imports

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

I- Statistiques descriptives

Chargement du dataset

In [2]:
```
df=pd.read_csv('/Users/mouadantari/Downloads/data_v1.0 (3).csv', index_col='i
df.drop(columns='Unnamed: 0', inplace=True)
df.head()
```

Out[2]:

index	date	cheveux	age	exp	salaire	sexe	diplome	specialite	note	dispo	embauch
0	2012-06-02	roux	25.0	9.0	26803.0	F	licence	geologie	97.08	non	
1	2011-04-21	blond	35.0	13.0	38166.0	M	licence	forage	63.86	non	
2	2012-09-07	blond	29.0	13.0	35207.0	M	licence	geologie	78.50	non	
3	2011-07-01	brun	NaN	12.0	32442.0	M	licence	geologie	45.09	non	
4	2012-08-07	roux	35.0	6.0	28533.0	F	licence	detective	81.91	non	

In [3]:
```
df.shape
```

Out[3]:
```
(20000, 11)
```

1-Description et retraitements du dataset

La colonne date

In [4]:
```
df.date
```

Out[4]:
```
index
0        2012-06-02
1        2011-04-21
2        2012-09-07
3        2011-07-01
4        2012-08-07
            ...
19995    2012-03-10
19996    2010-09-19
19997    2010-09-02
19998    2011-12-06
19999    2010-11-29
Name: date, Length: 20000, dtype: object
```

La colonne date est de type `object`, on la transforme en `datetime64` :

In [5]:
```
df.date=df.date.apply(pd.Timestamp)
```

In [6]:
```
df.date
```

Out[6]:
```
index
0       2012-06-02
1       2011-04-21
2       2012-09-07
3       2011-07-01
4       2012-08-07
           ...
19995   2012-03-10
19996   2010-09-19
19997   2010-09-02
19998   2011-12-06
19999   2010-11-29
Name: date, Length: 20000, dtype: datetime64[ns]
```

Description du dataset

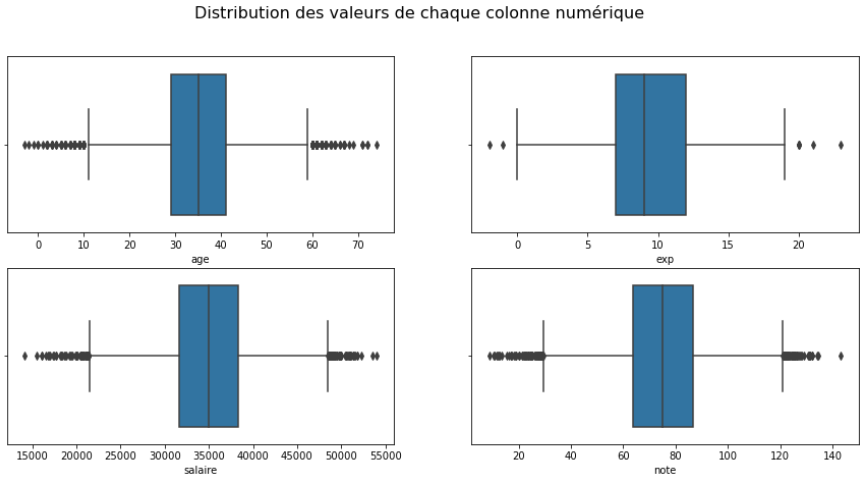
In [7]:
```
df.describe(include = [np.number,np.datetime64]).loc[['first', 'last', 'count
```

Out[7]:

	date	age	exp	salaire	note	embauche
first	2010-01-01 00:00:00	NaN	NaN	NaN	NaN	NaN
last	2014-12-31 00:00:00	NaN	NaN	NaN	NaN	NaN
count	19909	19909.000000	19904.000000	19905.000000	19886.000000	20000.000000
mean	NaN	35.004521	9.500352	34974.343934	75.168762	0.114600
std	NaN	9.609169	3.012109	5003.099616	17.095926	0.318547

De première vue, aucune des variables numériques ni la variable "date" ne contiennent 20000 valeurs, ils coniennent alors des valeurs vides.

In [8]:
```
fig, axes=plt.subplots(2,2,figsize=(15,7))
sns.boxplot(df.age,ax=axes[0,0])
sns.boxplot(df.exp,ax=axes[0,1])
sns.boxplot(df.salaire,ax=axes[1,0])
sns.boxplot(df.note,ax=axes[1,1])
fig.suptitle("Distribution des valeurs de chaque colonne numérique",fontsize=
plt.show()
```



- On remarque que la variable "âge" varie entre des valeurs négatives et des valeurs supérieures à 70 ans. Un nettoyage des valeurs négatives et des valeurs supérieures à 70 ans (âge maximal de travail en France).
- On remarque aussi que des notes dépassent la note maximale de l'exercice (100), ce qui n'est pas valide.
- On vérifie maintenant la différence entre les variables "âge" et "exp" (expériance). Vu que l'âge legal pour travailler en France est de 16 ans, on cherche les différences inférieures à cette valeur.

In [9]:
```
df=df.drop(index=df[df.age<0].index)
df=df.drop(index=df[df.age>=70].index)
df=df.drop(index=df[df.exp<0].index)
df=df.drop(index=df[df.note>100].index)
```

In [10]:
```
len(df[df.age-df.exp<16])
```

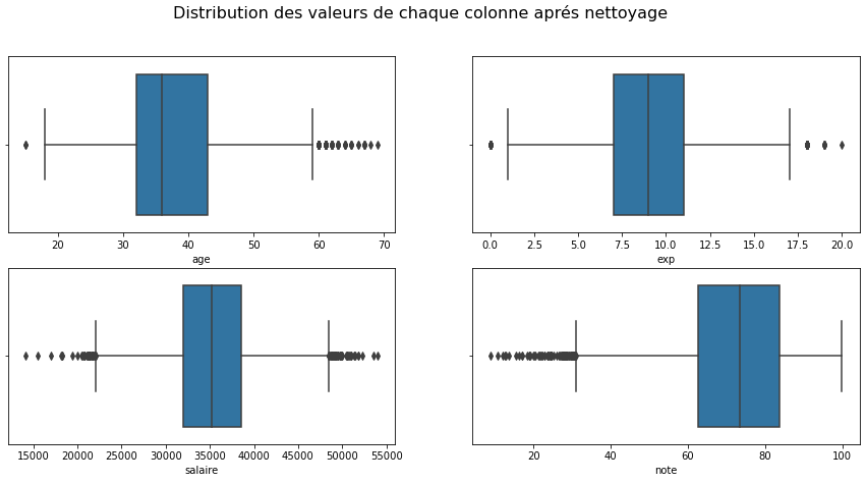
Out[10]:
```
2890
```

- On nettoie alors 2890 lignes du dataset ont une expérience supérieure à la durée dans laquelle ils ont été autorisés à travailler.

In [11]:
```
df=df.drop(index=df[df.age-df.exp<16].index)
```

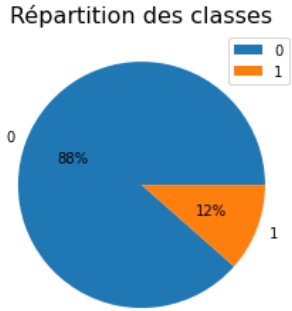
Distribution des données numériques aprés nettoyage:

In [12]:
```
fig, axes=plt.subplots(2,2,figsize=(15,7))
sns.boxplot(df.age,ax=axes[0,0])
sns.boxplot(df.exp,ax=axes[0,1])
sns.boxplot(df.salaire,ax=axes[1,0])
sns.boxplot(df.note,ax=axes[1,1])
fig.suptitle("Distribution des valeurs de chaque colonne aprés nettoyage",fon
plt.show()
```



Variable cible

In [13]:
```
pie=plt.pie(df.embauche.value_counts(),
            labels=df.embauche.value_counts().index, autopct='%.f%%')
plt.title('Répartition des classes', fontsize=16)
plt.legend()
plt.show()
```



On note que les données sont fortement déséquilibrés.

Aprés avoir traiter les attributs numériques, on se tourne vers les attributs catégoriques:

In [14]:
```
print('Les valeurs de la colonne "cheveux"   : ',df.cheveux.unique())
print('Les valeurs de la colonne "sexe"      : ',df.sexe.unique())
print('Les valeurs de la colonne "diplome"   : ',df.diplome.unique())
print('Les valeurs de la colonne "specialite": ',df.specialite.unique())
print('Les valeurs de la colonne "dispo"     : ',df.dispo.unique())
```

```
Les valeurs de la colonne "cheveux"   :  ['roux' 'blond' 'brun' 'chatain' nan]
Les valeurs de la colonne "sexe"      :  ['F' 'M' nan]
Les valeurs de la colonne "diplome"   :  ['licence' 'master' 'doctorat' 'bac'
nan]
Les valeurs de la colonne "specialite":  ['geologie' 'forage' 'detective' 'arc
heologie' nan]
Les valeurs de la colonne "dispo"     :  ['non' 'oui' nan]
```

Même si les attributs catégoriques contiennent des vides (les valeurs `nan`), on ne les supprimerons pas pour ne pas dégrader d'avantage le dataset, et parce que ces attributs seront encodés par le `OneHotEncoder` qu'on forcera à encoder les `nan` en des zéros dans toutes les catégories de l'attribut vide.

In [15]:
```
from sklearn.preprocessing import OneHotEncoder

categories=np.array([df.cheveux.unique()[:-1],df.sexe.unique()[:-1],
            df.diplome.unique()[:-1],df.specialite.unique()[:-1],
            df.dispo.unique()[:-1]])
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore', categories=cat
```

In [16]:
```
enc_data=encoder.fit_transform(df[['cheveux','sexe','diplome','specialite','d
enc_names=encoder.get_feature_names(['cheveux','sexe','diplome','specialite',
enc_features=pd.DataFrame(enc_data, columns=enc_names, index=df.index)
```

In [17]:
```
enc_df=pd.concat([enc_features,df[['date','age','exp','salaire','note','embau
```

Le dataset, et surtout les variables catégoriques, prennent la forme suivante:

In [18]:
```
enc_df.head()
```

Out[18]:

index	cheveux_roux	cheveux_blond	cheveux_brun	cheveux_chatain	sexe_F	sexe_M	diplon
0	1.0	0.0	0.0	0.0	1.0	0.0	
1	0.0	1.0	0.0	0.0	0.0	1.0	
2	0.0	1.0	0.0	0.0	0.0	1.0	
3	0.0	0.0	1.0	0.0	0.0	1.0	
4	1.0	0.0	0.0	0.0	1.0	0.0	

5 rows × 22 columns

Un exemple d'encodage de la valeur `nan`:

- Avant l'encodage:

In [19]:
```
df.loc[283:285]
```

Out[19]:

index	date	cheveux	age	exp	salaire	sexe	diplome	specialite	note	dispo	embauch
283	2013-05-09	brun	30.0	0.0	32969.0	M	doctorat	forage	96.81	oui	
284	2014-07-19	NaN	25.0	9.0	32263.0	M	master	forage	66.13	oui	
285	2011-03-27	brun	32.0	13.0	40195.0	M	licence	geologie	74.40	non	

- Après l'encodage:

In [20]:
```
enc_df.loc[283:285]
```

Out[20]:

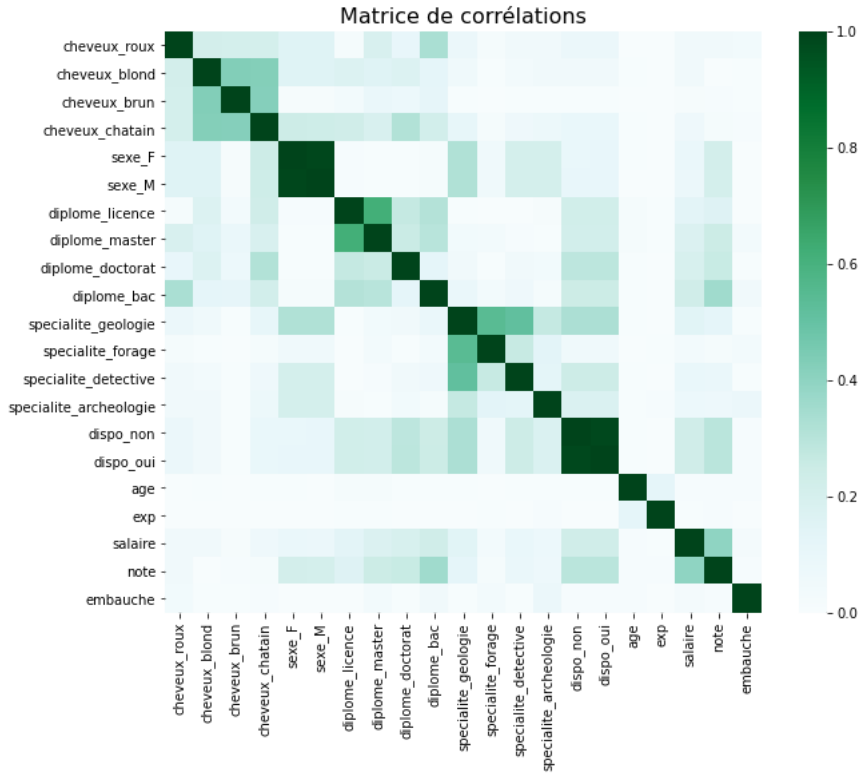
index	cheveux_roux	cheveux_blond	cheveux_brun	cheveux_chatain	sexe_F	sexe_M	diplon
283	0.0	0.0	1.0	0.0	0.0	1.0	
284	0.0	0.0	0.0	0.0	0.0	1.0	
285	0.0	0.0	1.0	0.0	0.0	1.0	

3 rows × 22 columns

2-Dépendance des paramètres

Corrélations entre les nouveaux attributs

In [21]:
```
plt.figure(figsize=(10,8))
sns.heatmap(enc_df.corr().abs(),cmap='BuGn',vmin=0,vmax=1)
plt.title("Matrice de corrélations", fontsize=16)
plt.show()
```



- On remarque que les variables ne sont fortement corrélées que si elles représentent le même attribut (dispo, sexe, cheveux). On remarque aussi que la variable cible (embauche) est trés faiblement corrélée à toutes les autres variables.

- Afin de vérifier les dépendences demandées à l'énoncé, on prépare les fonctions suivantes:

  - La fonction `anova` qui retourne la valeur statistique F et la "p-value" du test statistique "one-way ANOVA". Ce test verifie l'hypotèse nulle d'indépendence de deux variables: une variable numérique et une autre catégorique.
  - La fonction `vCramer` qui retourne le coefficient de Cramer qui représente une mesure de corrélation entre deux variables catégoriques.

In [22]:
```
from scipy.stats import f_oneway, chi2_contingency, chisquare
def anova(num_col,cat_col):
    return f_oneway(*[group[num_col].dropna() for key, group in df.groupby(by=

def vCramer(col1,col2):
    cross=pd.crosstab(df[col1],df[col2])
    chi2=chi2_contingency(cross)[0]
    V=np.sqrt(chi2/(len(df)*(min(cross.shape)-1)))
    return V
```

On réalise les mesures de dépendence demandés:

In [23]:
```
vCramer('sexe','specialite',)
```

Out[23]:
```
0.3603818191207106
```

In [24]:
```
anova('salaire','cheveux')
```

Out[24]:
```
F_onewayResult(statistic=38.99921247248099, pvalue=4.24508603502273e-25)
```

In [25]:
```
np.abs(enc_df.corr()['exp']['note'])
```

Out[25]:
```
0.015510256456495508
```

- Les variables "sexe" et "specialite" sont corrélées à V=0.36, ce qui n'est pas une forte valeur par rapport à la valeur de dépendence maximale (V=1).
- Les variables "salaire" et "cheveux" ne sont pas indépendantes (statistiquement parlant) puisque la `p-value=3.55e-25` ce qui est suffisemment inférieur à 0.05 pour rejeter l'hypotèse de l'indépendance.
- Les variables "exp" et "note" sont corrélées à 1.56% donc faible dépendence.

De première vue, et considérant la faible corrélation de la variable cible avec toutes les autres variables, aucune variable n'est à écarter.

Imputation des vides restants

- Les vides de la variable "exp" sera considéré comme un candidat sans expérience.
- Les vides des variables "salaire" et "note" seront remplis par la moyenne interne à chaque classe de chaque variable, selon la classe du candidat.
- Le reste des vides sera éliminé.

In [26]:
```
from datetime import datetime
try:
    enc_df.exp.fillna(0, inplace=True)
    enc_df[df.embauche==1].salaire\
    .fillna(enc_df[df.embauche==1].salaire.mean(), inplace=True)

    enc_df[df.embauche==0].salaire\
    .fillna(enc_df[df.embauche==0].salaire.mean(), inplace=True)

    enc_df[df.embauche==1].note\
    .fillna(enc_df[df.embauche==1].note.mean(), inplace=True)

    enc_df[df.embauche==0].note\
    .fillna(enc_df[df.embauche==0].note.mean(), inplace=True)

    enc_df.dropna(inplace=True)
    enc_df.date=enc_df.date.apply(datetime.toordinal)
    print('done')
except: pass
```

```
done
```

II- Modèle prédictif

In [27]:
```
from sklearn.model_selection import train_test_split
X=enc_df.iloc[:,:-1]
y=enc_df.iloc[:,-1]
```

1- Algorithme et paramètre

L'algorithme que je vais utiliser pour mon modèle est le KNN, vu qu'il considère la proximité des point et dans un problème de recrutement deux profils rapproché ont plus de chance d'avoir les mêmes résultats.

Mais avant d'appliquer le KNN, une standarisation des variables est necessaire afin que les varibles soient pris en compte équitablement dans le calcul des distances.

Et pour remédier au problème de déséquilibre des classes, on effectue un sur-echantillonage afin d'augmenter la classe minoritaire. L'algorithme utilisé pour le sur-echantillonage est `ADASYN`, qui a la particularité de rajouter des données synthétiques prés des points de la classe minoritaire faussement prédit par un KNN.

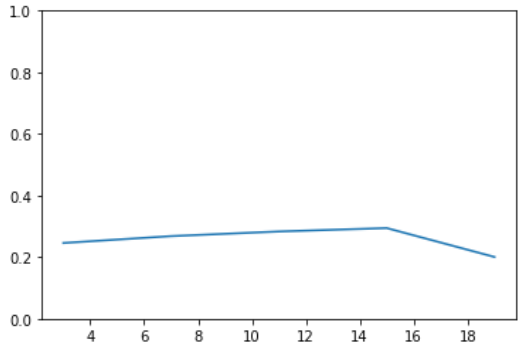
Finalement, afin de pouvoir faire une validation croisée, on insère les trois étapes de l'algorithme dans une pipeline qui sera le modèle à entrainer pour la validation croisée. Ceci a comme avantage de ne tester, à chaque itération, que sur des données originales et n'augmenter que les données dédiés à l'entrainement.

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_predict, cross_val_score
from sklearn.metrics import confusion_matrix
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import ADASYN
from sklearn.model_selection import KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import precision_score, classification_report

errors=np.zeros(5)
for k in range(3,22,4):
    predictor=KNeighborsClassifier(k)
    model= Pipeline([('std', StandardScaler()),
                     ('over',ADASYN(sampling_strategy=1)),
                     ('pred',predictor)])


    cv=KFold(n_splits=5, shuffle=True, random_state=0)
    score=cross_val_score(model, X, y, cv=cv).mean()
    #print(precision_score(y, pred))
    error=1-score
    i=(k-4)//4
    errors[i]=error
```

In [29]:
```
plt.plot(np.arange(3,22,4),errors,)
plt.ylim(0,1)
plt.show()
```



En traçant la courbe de l'erreur en fonction de K, on remarque que dans ce cas de figure ce paramètre n'influence pas le taux d'erreur. On choisit alors une valeur pas grande pour optimiser le temps d'entrainement.

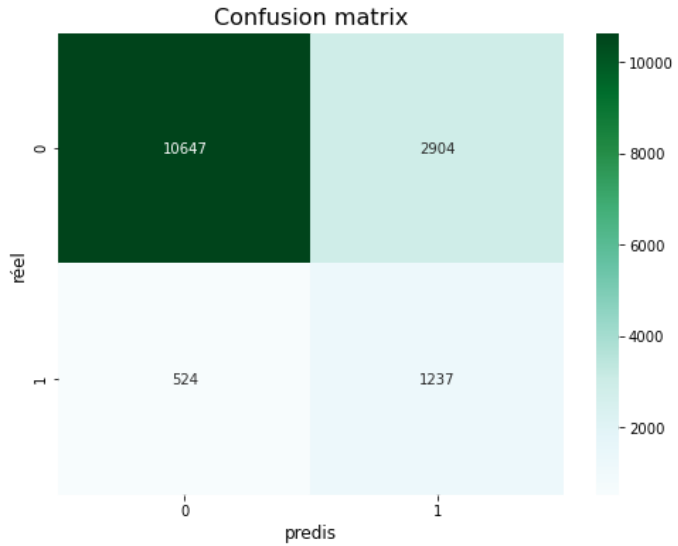
3- Critère de performance

En tant que modèle qui veut prédire si un profil est recruté ou pas, l'évaluation des performances de ce modèle se focalisera sur le taux des embauchés détectés par le modèle sur l'ensemble des embauchés, donc le critère de performance utilisé sera le "recall" sur le label 1. Le recall est le taux des vrais positifs prédis sur la totalité despositifs.

In [30]:
```
predictor=KNeighborsClassifier(5)
model= Pipeline([('std', StandardScaler()),
                 ('over',ADASYN(sampling_strategy=1)),
                 ('pred',predictor)])


cv=KFold(n_splits=5, shuffle=True, random_state=0)
pred=cross_val_predict(model, X, y, cv=cv)

plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y,pred)
            ,annot=True,fmt='g',cmap='BuGn')
plt.xlabel('predis',fontsize=12)
plt.ylabel('réel',fontsize=12)
plt.title('Confusion matrix',fontsize=16)
plt.show()
```



In [31]:
```
print(classification_report(y,pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.79      0.86     13551
           1       0.30      0.70      0.42      1761

    accuracy                           0.78     15312
   macro avg       0.63      0.74      0.64     15312
weighted avg       0.88      0.78      0.81     15312
```

On remarque que le recall pour la classe 1 est de 70%. Ce qui est satisfaisant pour un premier résultat d'un modèle améliorable.

4- Pistes d'amélioration

Le modèle utilisé dans cet exercice n'est qu'un premier resultat, certes améliorable. Comme pistes d'améliorations je propose:

- Etude de l'importance des variables du dataset puis pondération de ces variables dans le calcul de distance du KNN.
- Choisir une métrique du KNN en utilisant des techniques du metric-learning afin d'avoire une métrique plus adaptée et qui améliore les performences du modèle.

In [ ]:
```
```