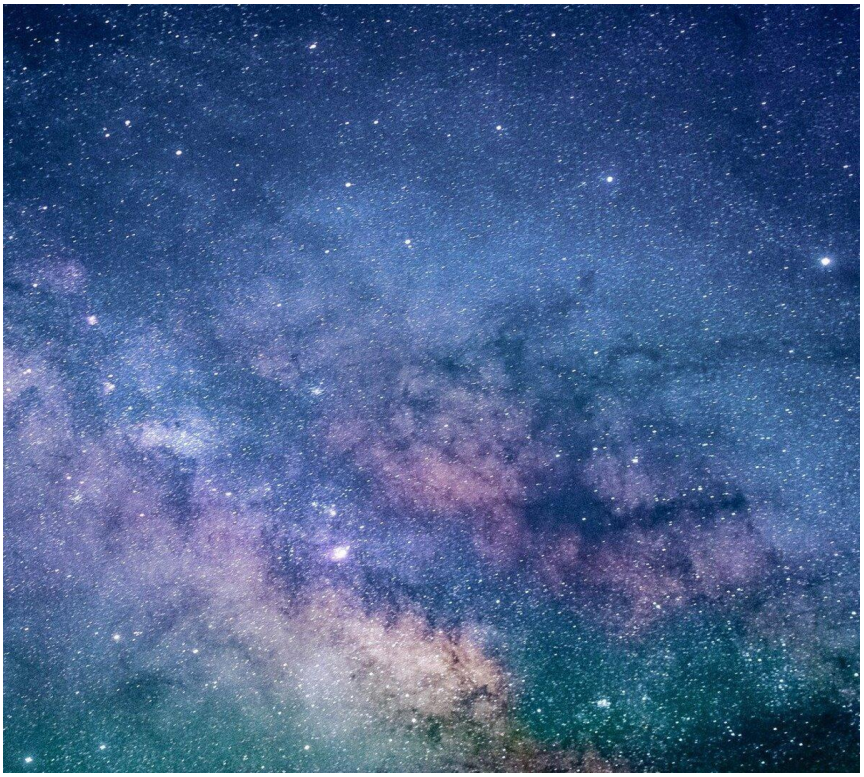
Stars Classification: Giants and Dwarfs



Part I

1.1 Introduction:

La classification stellaire utilise les données spectrales des étoiles pour les classer dans différentes catégories. Le système de classification stellaire moderne est connu sous le nom de système de classification Morgan-Keenan (MK). Il utilise l'ancien système de classification HR pour classer les étoiles selon leur chromaticité et utilise des chiffres romains pour classer la taille de l'étoile.

L'apprentissage automatique peut aider à la classification stellaire en permettant aux astronomes de traiter de grandes quantités de données spectrales d'étoiles de manière efficace et précise. Les algorithmes d'apprentissage automatique peuvent être formés sur des données spectrales d'étoiles préalablement classées selon le système de classification MK, afin de détecter des modèles et des caractéristiques dans les données qui permettent de les classer correctement.

Une fois que l'algorithme est formé, il peut être utilisé pour classer de nouvelles étoiles dont les données spectrales sont disponibles, mais qui n'ont pas encore été classées. Cela peut être particulièrement utile pour les grandes enquêtes astronomiques qui produisent des quantités massives de données, où la classification manuelle de chaque étoile peut être extrêmement chronophage.

En utilisant l'apprentissage automatique pour la classification stellaire, les astronomes peuvent accélérer le processus de classification et obtenir des résultats plus précis et cohérents.

1.2 Importation de données:

```
import pandas as pd
from google.colab import
drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
data = pd.read_csv("/content/drive/MyDrive/Star3642_balanced.csv",
decimal=',')
data
```

	Vmag	Plx	e_Plx	B-V	SpType	Amag	TargetClass
0	5.99	13.73	0.58	1.318	K5III	16.678352	0
1	8.7	2.31	1.29	-0.045	B1II	15.51806	0
2	5.77	5.5	1.03	0.855	G5III	14.471813	0
3	6.72	5.26	0.74	-0.015	B7V	15.324928	1
4	8.76	13.44	1.16	0.584	G0V	19.401997	1
...	...	...	...	...	...	...	...
3637	7.29	3.26	0.95	1.786	K4III	14.856089	0
3638	8.29	6.38	1.0	0.408	F2IV/V	17.314104	1
3639	6.11	2.42	0.79	1.664	M0/M1IIICNp	13.029078	0
3640	7.94	4.94	2.9	0.21	A5V	16.408636	1

```
3641 8.81    1.87 1.23    1.176     K1/K2III  15.169209           0

[3642 rows x 7 columns]
```

Les données se composent de 3642 observations et 7 colonnes.

Vmag: Visual Apparent Magnitude of the Star

Plx : La distance entre l'étoile et la terre.

e_Plx: Standard Error of Plx

B-V: B-V color index. (Une étoile chaude a un indice de couleur B-V proche de 0 ou négatif, tandis qu'une étoile froide a un indice de couleur B-V proche de 2,0. Les autres étoiles se situent quelque part entre les deux.)

SpType: Type spectral

Amag: Absolute Magnitude of the Star

TargetClass: L'étoile est de type Dwarf (0) ou bien Giant (1)

Pour SyType, les lettres romains correspond au type de l'étoile: *Dwarfs (I, II, III, VII)* Giants (IV, V, VI) On va supprimer donc la colonne SyType.

```
data = data.drop('SpType', inplace=False,
axis='columns') data.head()
```

```
   Vmag    Plx e_Plx     B-V       Amag TargetClass
0 5.99 13.73 0.58      1.318 16.678352
0
1   8.7   2.31 1.29  -0.045    15.51806           0
2 5.77     5.5  1.03   0.855 14.471813           0
3 6.72    5.26 0.74  -0.015  15.324928           1
4 8.76  13.44 1.16     0.584 19.401997           1
```

1.3 Description de données:

"dataset.shape" est utilisé pour connaître le nombre de lignes et de colonnes d'un ensemble de données que nous allons traiter. Dans cet exemple, l'ensemble de données a 3642 lignes et 7 fonctionnalités (colonnes).

```
data.shap

e (3642,

6)
```

"dataset.info()" est utilisé pour connaître les fonctionnalités disponibles dans un ensemble de données et pour fournir des informations sur la présence ou non de données manquantes dans chaque colonne. Cela est très important car les données manquantes ne peuvent pas être traitées et doivent être supprimées de l'ensemble de données. "dataset.info()" fournit également des informations sur le type de données pour chaque fonctionnalité dans un ensemble de données traité.

```
data.info()
```

```
<class
'pandas.core.frame.DataFrame'>
RangeIndex: 3642 entries, 0 to 3641
Data columns (total 6 columns):
 #   Column       Non-Null Count Dtype
 ---  -------      --------------  -----
 0   Vmag         3642 non-null   object
 1   Plx          3642 non-null   object
 2   e_Plx        3642 non-null   object
 3   B-V          3642 non-null   object
 4   Amag         3642 non-null   object
 5  TargetClass 3642 non-null
int64 dtypes: int64(1), object(5)
memory usage: 170.8+ KB
```

"dataset.describe()" est utilisé pour connaître les valeurs statistiques d'un ensemble de données. Ces valeurs comprennent le nombre d'observations, la moyenne, l'écart-type, le minimum, le premier quartile, la médiane, le troisième quartile et le maximum d'un ensemble de données.

```
data.describe()
```

```
       TargetClas
s count
3642.000000
mean      0.500000
std       0.500069
min       0.000000
25%       0.000000
50%       0.500000
75%       1.000000
max       1.000000
```

Comme "dataset.info()", "dataset.isnull().sum()" fournit également des informations sur le nombre de lignes qui ont des valeurs manquantes ou nulles pour chaque fonctionnalité/colonne dans un ensemble de données donné.

```
data.isnull().sum(
```

```
) Vmag         0

Plx            0
e_Plx          0
B-V            0
Amag           0
TargetClass    0
dtype: int64
```

Dans notre cas, on a pas de valeurs manquantes.

1.4 Preparation des données:

1.4.1 Encodage des valeurs de données d'objet en numérique

L'encodage de données est effectué car l'ensemble de données que nous traitons contient des fonctionnalités qui sont encore de type "objet", tandis que le type de données "objet" ne peut pas être traité et doit être converti en type de données numérique.

```
data.dtypes
```

```
Vmag           object
Plx            object
e_Plx          object
B-V            object
Amag           object
TargetClass
int64 dtype: object
```

```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
for c in data.columns[0:]:
    if(data[c].dtype=='object'):
        data[c] = encoder.fit_transform(data[c])
    else:
        data[c] = data[c]
```

```
data.dtypes
```

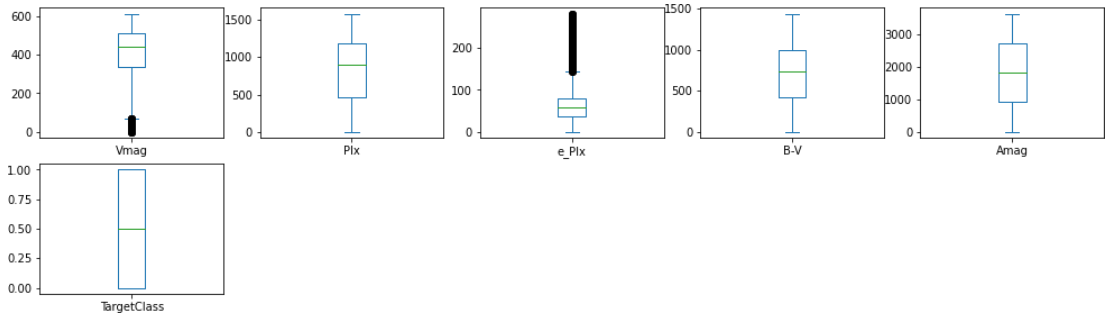
```
Vmag           int64
Plx            int64
e_Plx          int64
B-V            int64
Amag           int64
TargetClass    int64
dtype: object
```

1.4.4 Détection et traitement des valeurs aberrantes

L'une des étapes les plus importantes du prétraitement des données est la détection et le traitement des valeurs aberrantes (ou des données aberrantes), car elles peuvent avoir un impact négatif sur l'analyse statistique et le processus d'apprentissage d'un algorithme de machine learning, ce qui entraîne une précision moindre. On peut détecter les valeurs aberrantes en regardant les boîte à moustaches.

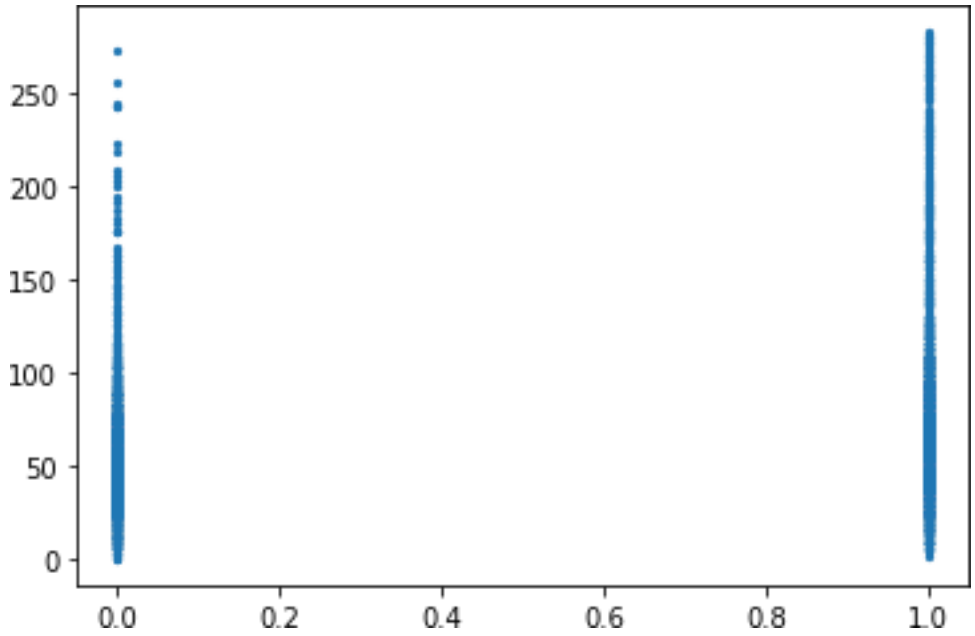
```
data.plot(kind='box', subplots=True, layout=(8,5), figsize=(17,20))
```

```
Vmag
AxesSubplot(0.125,0.799681;0.133621x0.0803191) Plx
AxesSubplot(0.285345,0.799681;0.133621x0.0803191) e_Plx
              AxesSubplot(0.44569,0.799681;0.133621x0.0803191)
B-V
AxesSubplot(0.606034,0.799681;0.133621x0.0803191) Amag
AxesSubplot(0.766379,0.799681;0.133621x0.0803191)
TargetClass      AxesSubplot(0.125,0.703298;0.133621x0.0803191)
dtype: object
```

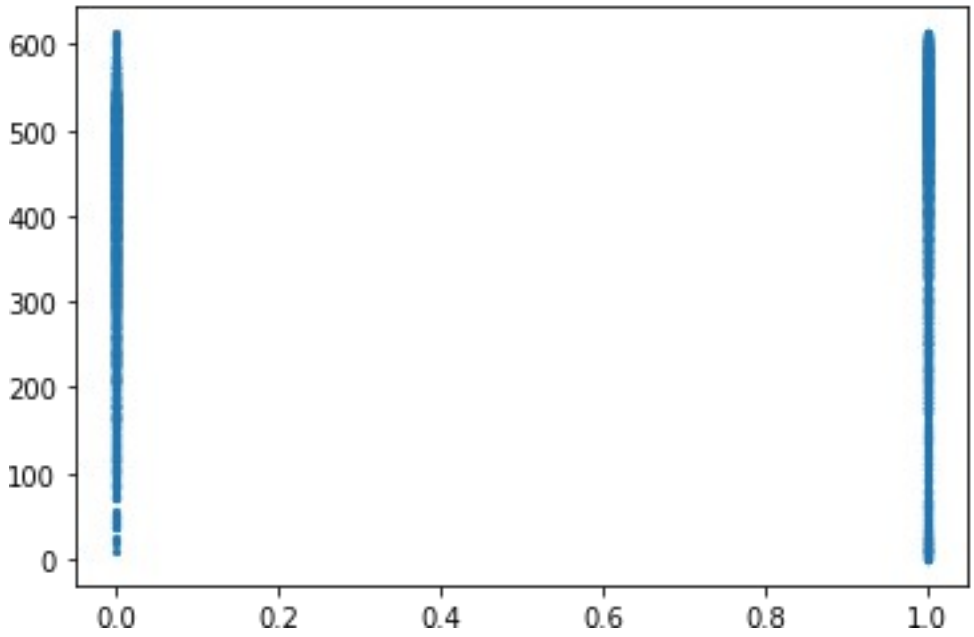


On peut les détecter aussi par scatter()

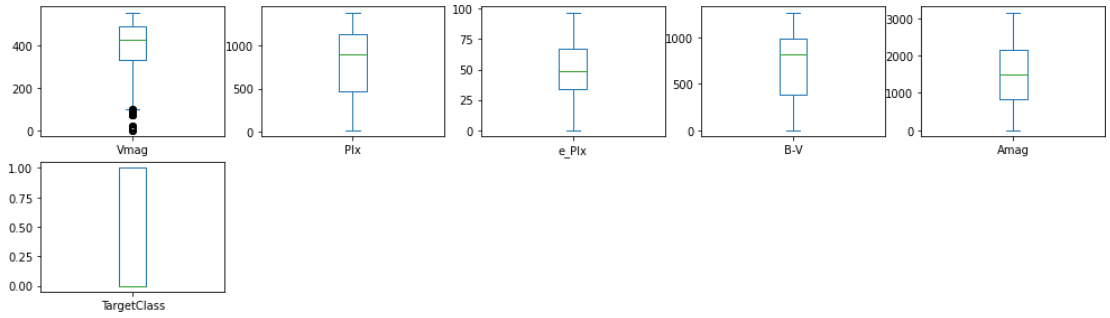
```
import matplotlib.pyplot as plt
fig = plt.scatter(data['TargetClass'], data['e_Plx'], s = 5)
```



```
fig = plt.scatter(data['TargetClass'], data['Vmag'], s = 5)
```




```
data=data[data['Vmag']<data['Vmag'].quantile(0.9)]
data=data[data['Plx']<data['Plx'].quantile(0.9)]
data=data[data['e_Plx']<data['e_Plx'].quantile(0.9)
] data=data[data['B-V']<data['B-V'].quantile(0.9)]
data=data[data['Amag']<data['Amag'].quantile(0.9)]

data.plot(kind='box', subplots=True, layout=(8,5), figsize=(17,20))

Vmag
AxesSubplot(0.125,0.799681;0.133621x0.0803191) Plx
AxesSubplot(0.285345,0.799681;0.133621x0.0803191) e_Plx
              AxesSubplot(0.44569,0.799681;0.133621x0.0803191)
B-V
AxesSubplot(0.606034,0.799681;0.133621x0.0803191) Amag
AxesSubplot(0.766379,0.799681;0.133621x0.0803191)
TargetClass      AxesSubplot(0.125,0.703298;0.133621x0.0803191)
dtype: object
```




```
data.shap

e (2144,

6)
```

On a juste supprimé les ouliers, et par conséquent la taille de notre dataset a été réduit.

1.5 Visualisation de données:

On commence par comprendre nos cibles classes:

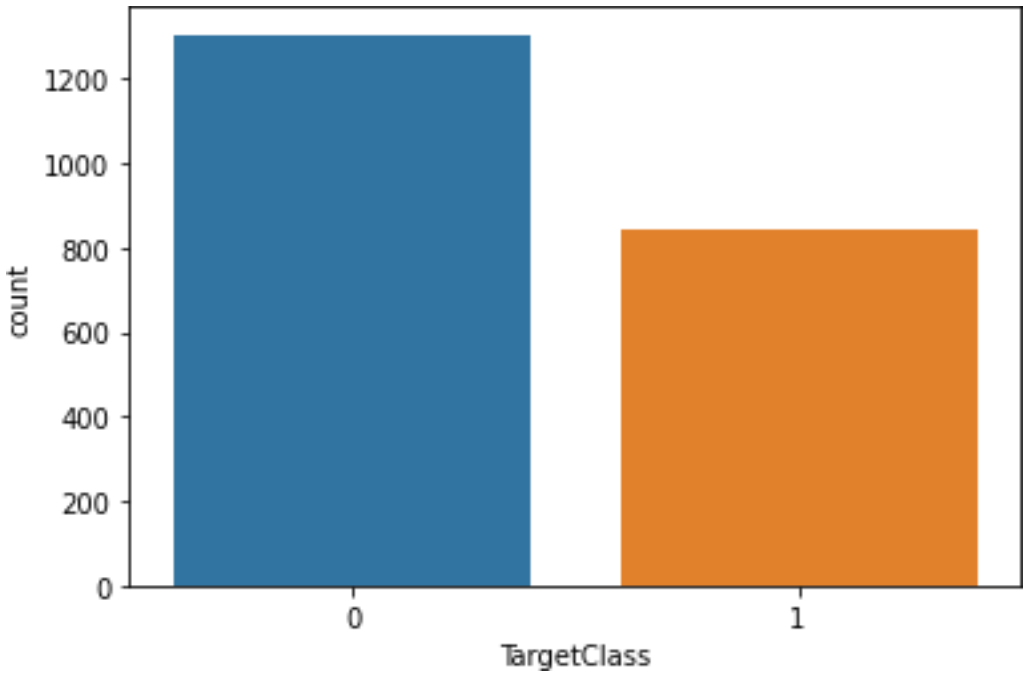
```
data['TargetClass'].value_counts(

) 0  1304

1     840
Name: TargetClass, dtype: int64
```

On a 1304 des étoiles de type Dwarf et 840 autres de type Giant.

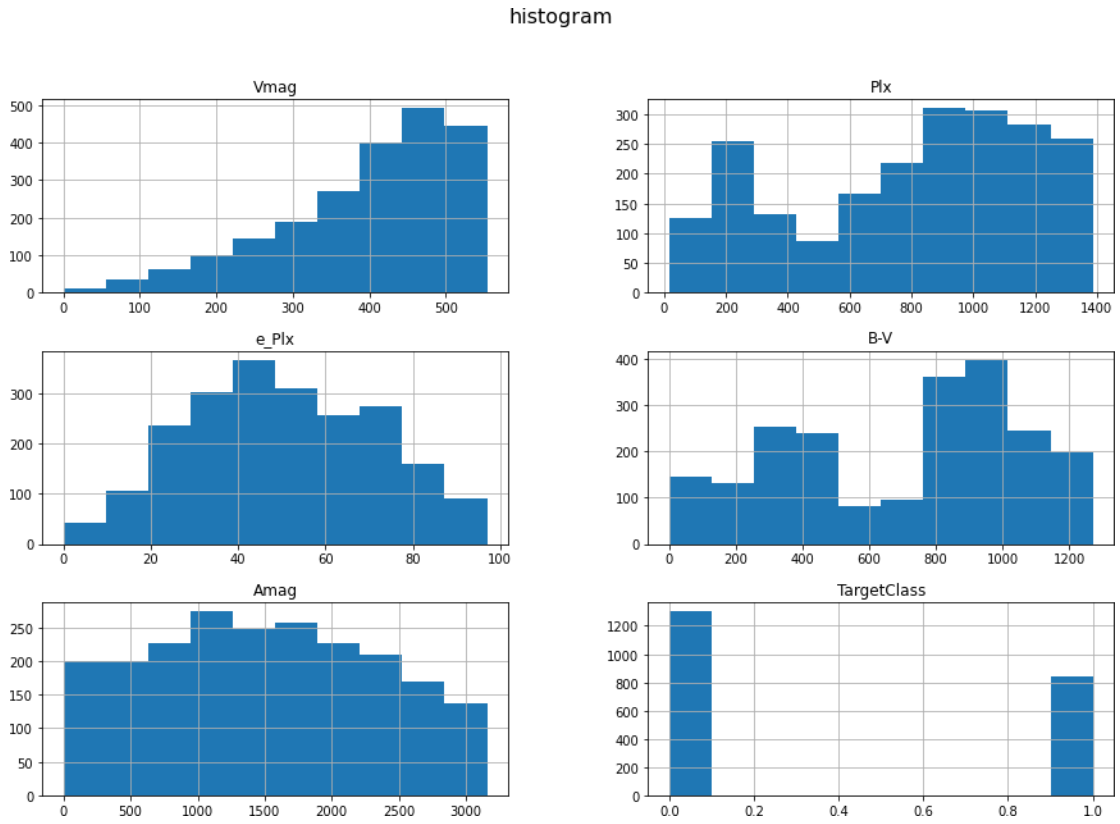
```
import seaborn as sns
sns.countplot(data['TargetClass'])
```

```
/usr/local/lib/python3.8/dist-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 explicit keyword will result in an
error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='TargetClass', ylabel='count'>
```




```
data.hist(figsize=(15,10))
plt.suptitle("histogram", fontsize=16)
plt.show()
```

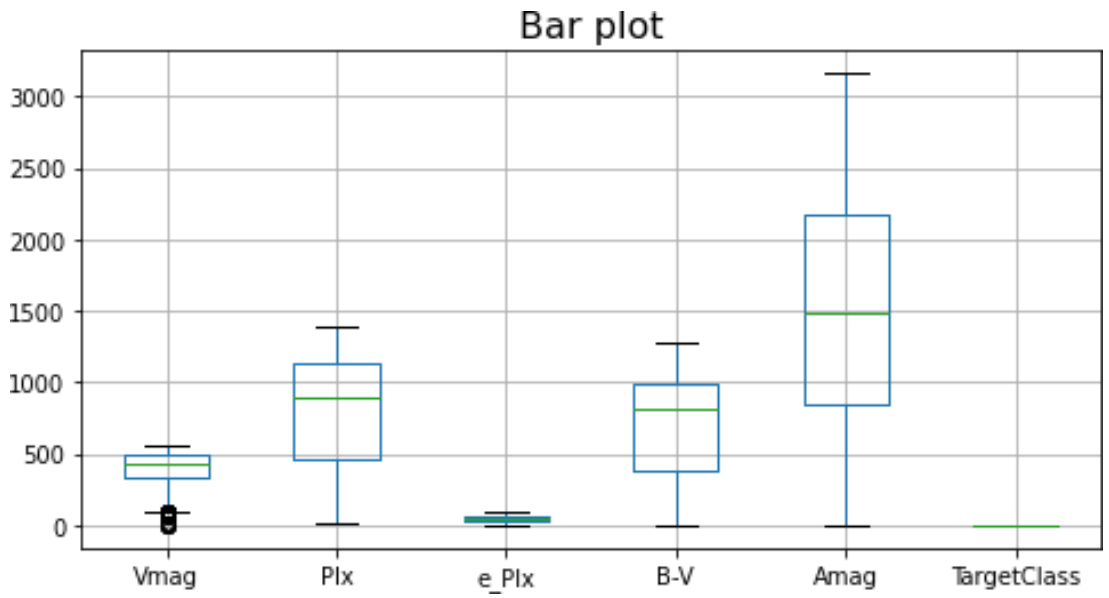


Les histogrammes des différentes variables ont des échelles différentes, cela peut indiquer que la normalisation des données est nécessaire. Par exemple, l'histogramme de ePlx a des valeurs qui varient entre 0 et 100, alors que l'histogramme de la variable Amag a des valeurs qui varient entre 0 et 3000, cela peut indiquer que la normalisation est nécessaire.

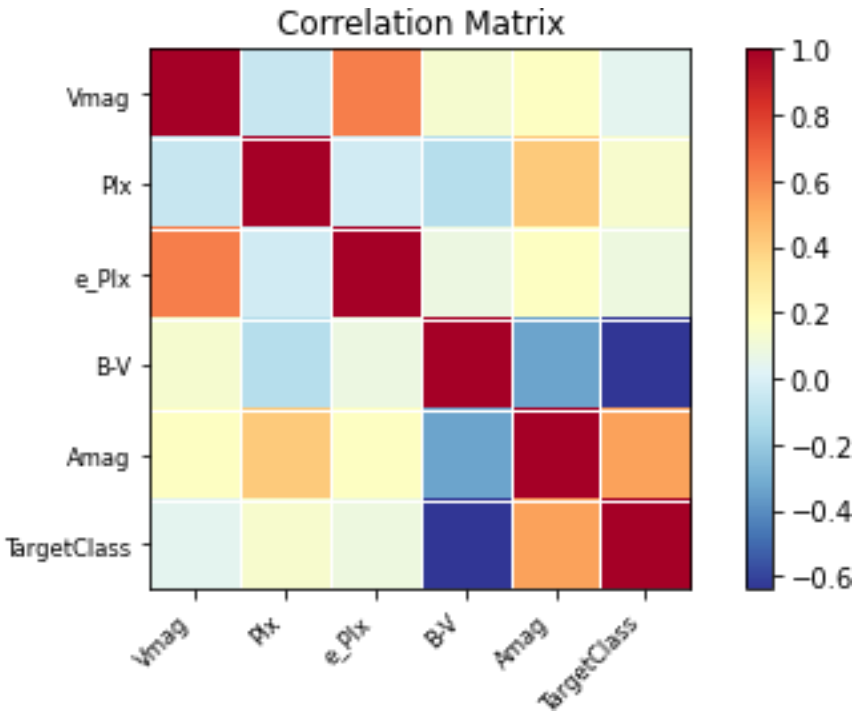
Ainsi les histogrammes sont asymétriques, sauf pour les variables ePlx et Amag , c'est-à-dire qu'il y a plus de valeurs dans une queue de la distribution que dans l'autre, cela peut indiquer que la normalisation est nécessaire.

Par exemple, k'histogramme de la variable Vmag a une queue longue et une concentration de valeurs plus faibles à gauche. Cela peut être un signe que la normalisation des données est nécessaire pour réduire l'influence des valeurs extrêmes lors du calcul de la distance entre les points en KNN.

```
data.boxplot(figsize=(8,4))
plt.title("Bar plot", fontsize=16)
plt.show()
```




```
#correlation matirx
corr = data.corr()
print(corr)
import statsmodels.api as sm
sm.graphics.plot_corr(corr,
xnames=list(corr.columns)) plt.show()
```



	Vmag	Plx	e_Plx	B-V	Amag	TargetClass
Vmag	1.000000	-0.059426	0.624642	0.131432	0.172578	0.039040
Plx	-0.059426	1.000000	-0.031150	-0.113369	0.414680	0.137238
e_Plx	0.624642	-0.031150	1.000000	0.080102	0.168931	0.087559
B-V	0.131432	-0.113369	0.080102	1.000000	-0.338770	0.641406
Amag	0.172578	0.414680	0.168931	-0.338770	1.000000	0.538003
TargetClass	0.039040	0.137238	0.087559	-0.641406	0.538003	1.000000

La matrice de corrélation est un outil utile pour explorer les relations entre les différentes variables explicatives dans notre jeu de données.

1- Les valeurs de corrélation : Les valeurs de corrélation dans la matrice varient de -1 à 1, où -1 indique une corrélation négative parfaite, 0 indique aucune corrélation et 1 indique une corrélation positive parfaite. Des valeurs proches de 1 ou -1 indiquent une forte relation linéaire entre les deux variables, tandis que des valeurs proches de 0 indiquent une faible relation ou aucune relation.

2- Les couleurs de la matrice : Les couleurs dans la matrice de corrélation peuvent aider à visualiser rapidement les corrélations les plus fortes ou les plus faibles. Les couleurs chaudes (rouge, orange) indiquent des corrélations positives, tandis que les couleurs froides (bleu, vert) indiquent des corrélations négatives.

3- Les variables les plus corrélées : Les variables les plus corrélées sont souvent importantes à identifier, car elles peuvent indiquer des relations causales potentielles ou

des variables qui sont des prédicteurs importants d'une variable cible. Les variables fortement corrélées peuvent également indiquer des problèmes de multicollinéarité, qui peuvent affecter la stabilité et l'interprétation des modèles de régression.

4- Les corrélations faibles ou nulles : Les corrélations faibles ou nulles peuvent également être importantes à identifier, car elles peuvent indiquer des variables qui n'ont pas de relation linéaire avec d'autres variables dans le jeu de données. Cela peut indiquer que ces variables ne sont pas importantes pour expliquer la variance dans la variable cible ou que leur relation avec la variable cible est non linéaire.

D'après la figure obtenue, on peut dire que Visual Apparent Magnitude of the Star et la distance entre la terre et l'étoile, ainsi que son écart type sont faiblement corrélé avec TargetClass, il n'y a pas une forte causalité.

Part II

1- Construction du modèle KNN (K-Nearest-Neighbors):

1.1 Normalisation des donnnées:

Il est souvent recommandé de normaliser les données avant d'utiliser l'algorithme KNN car la distance calculée entre deux points est sensible à l'échelle des variables. Si les variables ont des échelles différentes, celles qui ont des valeurs plus élevées peuvent avoir une influence disproportionnée sur la distance entre les points. La normalisation consiste à transformer les variables pour qu'elles aient une moyenne de zéro et une variance de un. Cela permet à toutes les variables d'avoir des échelles similaires et donc de contribuer de manière égale à la distance entre les points. La normalisation des données est particulièrement importante si les variables ont des unités différentes ou si elles ont des échelles de mesure différentes. Elle peut également être utile si certaines variables ont une variance beaucoup plus grande que d'autres, car cela peut entraîner une surpondération de certaines variables dans le calcul de la distance.

```
from sklearn.preprocessing import MinMaxScaler
mmscaler = MinMaxScaler()
cols = ['Vmag', 'Plx', 'e_Plx', 'B-V', 'Amag']
data[cols] = mmscaler.fit_transform(data[cols])
```

1.2 La standarisation:

La standardisation des données est souvent recommandée avant d'utiliser l'algorithme des k plus proches voisins (KNN). La raison en est que KNN utilise une mesure de distance pour trouver les voisins les plus proches, et les variables avec des échelles différentes peuvent avoir une influence disproportionnée sur cette mesure de distance. La standardisation des données est généralement recommandée pour l'algorithme KNN car elle peut améliorer la précision de l'algorithme en éliminant l'impact des différentes échelles de variables sur la mesure de distance. Dans les histogrammes (au dessus) on remarque que la plupart des varibales n'ont pas une distribution normale. Donc il faut les standariser.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
cols = ['Vmag', 'Plx', 'e_Plx', 'B-V', 'Amag']
data[cols] = sc.fit_transform(data[cols])
```

1.3 Le modèle KNN:

L'algorithme K-Nearest Neighbor est un algorithme d'apprentissage supervisé où le résultat de la nouvelle instance est classifié en fonction de la majorité des catégories des k-voisins les plus proches. L'objectif de cet algorithme est de classifier de nouveaux objets en fonction des attributs et des échantillons des données d'entraînement. L'algorithme k-

Nearest Neighbor utilise la classification de voisinage comme valeur de prédiction pour la nouvelle instance.

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.model_selection import
train_test_split from sklearn.datasets import
load_iris
X = data.drop(['TargetClass'], axis =
1) y = data["TargetClass"]
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X,y)
y_model =
model.predict(X)
accuracy_score(y,y_model)
```

```
0.9095149253731343
```

l'accuracy_score a une valeur de 1, cela signifie que le modèle a correctement prédit toutes les étiquettes de classe dans l'ensemble de données de test.

En d'autres termes, le modèle n'a commis aucune erreur de classification et a réussi à identifier avec précision toutes les instances de chaque classe. Cependant, il convient de noter que l'obtention d'une précision de 1 peut être indicative d'un surajustement (overfitting) du modèle aux données d'entraînement, ce qui signifie que le modèle peut ne pas généraliser bien aux nouvelles données.

2- L'approche de Holdout:

2.1- Splitting Data

La division des données est utile pour séparer chaque fonctionnalité en variables que nous voulons. Dans ce cas, les fonctionnalités sont séparées en variables x et y, qui sont ensuite utilisées comme données d'entraînement (train) et données de test (test).

```
from sklearn.model_selection import train_test_split

y = data['TargetClass']
x = data.drop(['TargetClass'], axis = 1)

# Split the dataset to train and test data
train_x, test_x, train_y, test_y = train_test_split(x, y, train_size
= 0.8, random_state = 0)
```

2.2- Holdout:

L'approche de Holdout en apprentissage supervisé est une technique de validation de modèle qui consiste à diviser le jeu de données en deux ensembles distincts : l'ensemble d'entraînement (training set) et l'ensemble de test (test set).

L'ensemble d'entraînement est utilisé pour entraîner le modèle, tandis que l'ensemble de test est utilisé pour évaluer la performance du modèle après l'entraînement. Le modèle est ajusté sur l'ensemble d'entraînement et ensuite évalué sur l'ensemble de test. Cette évaluation permet de mesurer les performances du modèle sur des données qu'il n'a jamais vues auparavant.

L'approche de Holdout est l'une des méthodes les plus simples pour évaluer la performance d'un modèle, mais elle nécessite de disposer d'un nombre suffisant de données pour permettre une division significative entre l'ensemble d'entraînement et l'ensemble de test. Lorsqu'il y a un nombre limité de données disponibles, il peut être préférable d'utiliser des méthodes de validation croisée (cross-validation) pour évaluer la performance du modèle.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
parameters = {
    "algorithm" : "auto",
    "leaf_size" : 30,
    "metric" :
    "euclidean",
    "metric_params" :
    None, "n_jobs" : 1,
    "n_neighbors" : 1,
    "p": 2,
    "weights" : "uniform"
}
KNN =
KNeighborsClassifier(**parameters)
KNN.fit(train_x, train_y)
KNN_Predictions = KNN.predict(test_x)
accuracy_score(test_y, KNN_Predictions)
```

```
0.8111888111888111
```

2.3- Tracer l'ajustement du modèle:

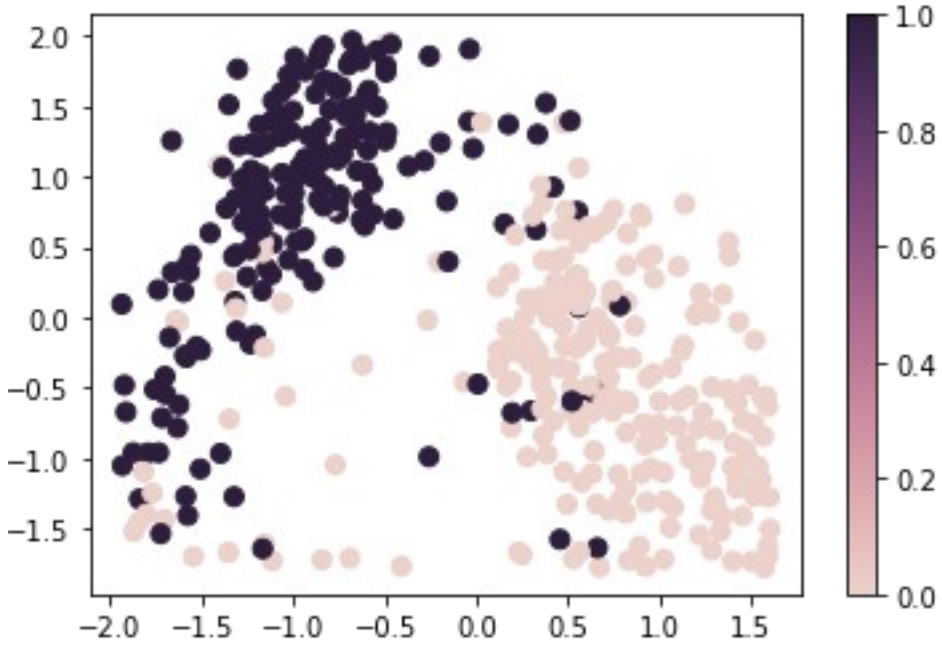
Avant de commencer à améliorer le modèle, il est important de comprendre comment il a appris en visualisant comment il a effectué ses prédictions.

Dans ce bloc de code, on a utilisé Seaborn pour créer un graphique de dispersion des 3ème et 4ème colonnes de test_x en sous-ensemble les tableaux test_x[:,3] et X_test[:,4]. Rappelez-vous que le 3ème et 4ème colonnes sont B-V et SpType. Elles sont fortement corrélées, comme vous l'avez vu dans le tableau des corrélations.

On a utilisé c pour spécifier que les valeurs prédites (KNN_Predictions) doivent être utilisées comme une barre de couleurs. L'argument s est utilisé pour spécifier la taille des points dans le graphique de dispersion. Et cmap pour spécifier la carte de couleurs cubehelix_palette.

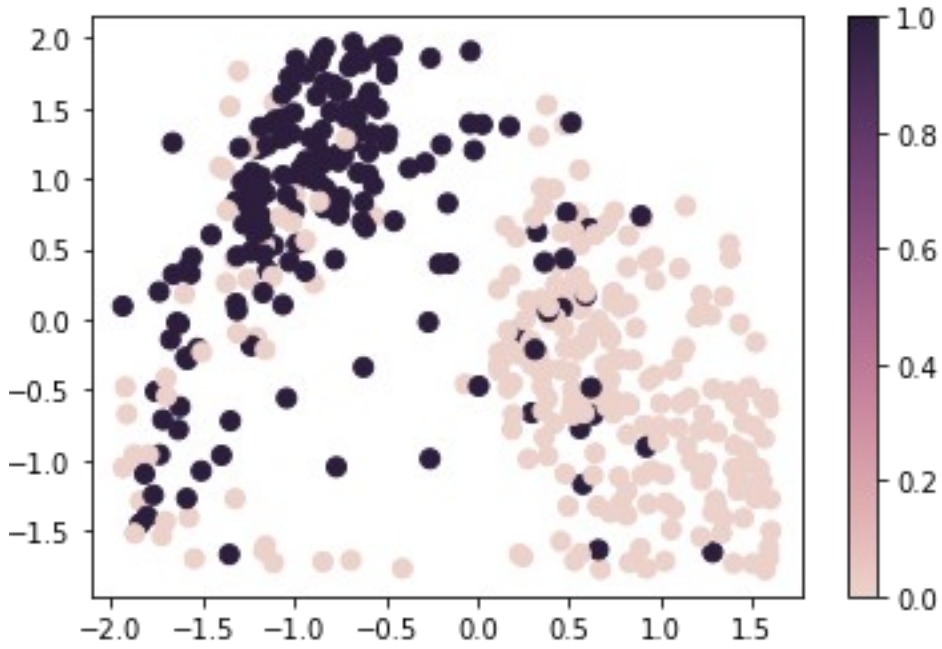
Sur ce graphique, chaque point représente une étoile de l'ensemble de test, avec sa vakeur de B-V et Vmag réel sur les axes X et Y, respectivement. La couleur du point reflète le type de l'étoile (Giant ou Dwarft).

```
import seaborn as sns

cmap = sns.cubehelix_palette(as_cmap = True)
f, ax = plt.subplots()
points = ax.scatter(test_x.iloc[:, 3], test_x.iloc[:, 4], c =
KNN_Predictions, s = 50, cmap=cmap)
f.colorbar(points)

plt.show()
```



Pour confirmer si cette tendance existe dans les données réelles, on peut faire la même chose pour les valeurs réelles en remplaçant simplement la variable utilisée pour c :

```
cmap2 = sns.cubehelix_palette(as_cmap =
True) f2, ax2 = plt.subplots()
points2 = ax2.scatter(test_x.iloc[:, 3], test_x.iloc[:, 4], c =
test_y, s = 50, cmap=cmap2)
f2.colorbar(points2)

<matplotlib.colorbar.Colorbar at 0x7f2dfa8df910>
```



Cela indique que la tendance de notre modèle a effectivement du sens.

3- Cross Validation:

La validation croisée est une technique utilisée en apprentissage automatique pour évaluer les performances d'un modèle sur un ensemble de données limité. Elle implique la partition des données en plusieurs sous-ensembles, ou "plis", puis l'utilisation de chaque pli tour à tour pour évaluer le modèle qui a été entraîné sur les données restantes.

Par exemple, dans la validation croisée k-fold, les données sont divisées en k plis de taille égale, avec un pli utilisé comme ensemble de validation et les k-1 autres plis utilisés pour l'entraînement. Ce processus est répété k fois, chaque pli étant utilisé une fois comme ensemble de validation. La performance du modèle est alors mesurée comme la moyenne de la performance sur tous les k plis.

La validation croisée est utile pour évaluer la capacité d'un modèle à généraliser à de nouvelles données non vues auparavant, et peut être particulièrement précieuse lorsque la quantité de données disponibles est limitée. Elle peut également aider à identifier des problèmes potentiels de surajustement, où un modèle fonctionne bien sur les données d'entraînement mais mal sur de nouvelles données. En évaluant le modèle sur plusieurs sous-ensembles des données, la validation croisée peut fournir une estimation plus fiable des performances réelles du modèle.

```
#5 fold
from sklearn.model_selection import cross_val_score
cross_val_score(model, X, y, cv = 5)
```

```
array([0.86013986, 0.84615385, 0.86713287, 0.86013986, 0.87616822])
```

Ce qui signifie que le modèle KNN a obtenu une précision de 81,81 % sur le premier pli, 81,58 % sur le deuxième pli, 82,98 % sur le troisième pli, 81,35 % sur le quatrième pli et 83,17 % sur le cinquième pli.

4- Leave-One-Out Cross Validation (LOOCV)

Leave-One-Out Cross Validation (LOOCV) est une méthode de validation croisée en apprentissage automatique où chaque instance dans l'ensemble de données est utilisée comme ensemble de validation unique et le reste des instances est utilisé pour l'entraînement du modèle.

Dans la LOOCV, le modèle est entraîné sur tous les exemples sauf un, puis la performance du modèle est évaluée en utilisant l'exemple laissé de côté comme ensemble de validation. Ce processus est répété pour chaque exemple dans l'ensemble de données, de sorte que chaque exemple est utilisé une fois comme ensemble de validation.

La LOOCV est utile pour les petits ensembles de données, car elle fournit une estimation de la performance du modèle qui est basée sur tous les exemples disponibles dans l'ensemble de données. Cependant, elle peut être coûteuse en temps de calcul, car elle nécessite l'entraînement d'un modèle pour chaque exemple dans l'ensemble de données.

En général, la LOOCV est plus précise que la validation croisée k-fold lorsque l'ensemble de données est petit, mais elle peut être moins précise que la validation croisée k-fold lorsque l'ensemble de données est grand.

```
#1OOCV
from sklearn.model_selection import LeaveOneOut
scores = cross_val_score(model, X, y, cv = LeaveOneOut() )
scores.mean()
print(scores)
print(scores.mean()
)
```

```
[1. 0. 1. ... 0. 1. 0.]
0.8568097014925373
```

La variable "scores" stocke un tableau de scores de précision pour chaque pli. Chaque élément du tableau "scores" représente la précision du modèle lors de la prédiction de la variable cible pour l'échantillon correspondant.

5- Évaluation du modèle:

Il est important de noter que l'accuracy score seul ne fournit pas toujours une image complète de la performance du modèle, car il ne prend pas en compte les erreurs de type I et de type II. Par conséquent, il peut être utile d'examiner d'autres mesures de performance telles que la matrice de confusion, le taux de faux positifs, le taux de faux négatifs et la précision et le rappel pour évaluer la performance globale du modèle.

```
from sklearn.model_selection import
train_test_split from sklearn.neighbors import
KNeighborsClassifier from sklearn.metrics import
accuracy_score
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
import numpy as np
X1, X2, y1, y2 = train_test_split(x,y, random_state= np.random,
train_size=0.7)
y_pred = model.fit(X1,y1).predict(X2)
y2_model = model.predict(X2)
cm1= confusion_matrix(y2,
y2_model) print("Confusion Matrix:
\n", cm1)
```

```
Confusion Matrix:
 [[353 35]
 [ 54 202]]
```

On a 101 parmi 644 sont mal prédites par le modèl, c'est à dire presque 16% des données de test ne sont pas bien prédite par le modèle.

```
print(accuracy_score(y2,

y2_model)) 0.8618012422360248
```

Et donc presque 84% des données de test sont bien prédites par le modèle. Elle est calculée en divisant le nombre de prédictions correctes par le nombre total de prédictions.

```
print(precision_score(y2, y2_model, average='weighted'))
```

```
0.8613586604780881
```

Precision_Score est une mesure de la précision des prédictions positives. Elle est calculée en divisant le nombre de vrais positifs par la somme des vrais positifs et des faux positifs.

```
print(recall_score(y2, y2_model, average='macro'))
```

```
0.8494281572164948
```

Recall : il s'agit d'une mesure de la capacité du modèle à identifier tous les exemples positifs. Elle est calculée en divisant le nombre de vrais positifs par la somme des vrais positifs et des faux négatifs. Elle peut être calculée à l'aide de la fonction.

```
print("Classification report :", classification_report(y2, y2_model))
```

```
Classificatio  report :              precision    recall  f1-score
n support

           0       0.87      0.91      0.89       388
           1       0.85      0.79      0.82       256

    accuracy                           0.86       644
   macro avg       0.86      0.85      0.85       644
weighted avg       0.86      0.86      0.86       644
```

6- Grid Search:

Jusqu'à présent, on a toujours travaillé avec k=3 dans l'algorithme kNN, mais la meilleure valeur pour k est quelque chose qu'on doit trouver empiriquement pour chaque ensemble de données.

Pour trouver la meilleure valeur pour k, on va utiliser un outil appelé GridSearchCV. C'est un outil qui est souvent utilisé pour ajuster les hyperparamètres des modèles d'apprentissage automatique. Dans notre cas, cela aidera en trouvant automatiquement la meilleure valeur de k pour notre ensemble de données.

```
#Grid search
from sklearn.model_selection import GridSearchCV
#creat a new KNN model
Knn2 = KNeighborsClassifier()
grid_param={'n_neighbors': range(1,50),
'weights' : ['uniform', 'distance'],
'metric' : ['euclidean', 'manhattan', 'minkowski']}
grid = GridSearchCV(Knn2, grid_param, cv = 10, scoring = 'accuracy')
grid.fit(X,y)
print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_
)
```

```
0.8792001738752445
{'metric': 'manhattan', 'n_neighbors': 18, 'weights': 'distance'}
KNeighborsClassifier(metric='manhattan', n_neighbors=18,
weights='distance')
```

Part III

L'arbre de décision est un modèle d'apprentissage automatique qui permet de représenter graphiquement des décisions et des actions possibles en fonction des caractéristiques ou des attributs d'un ensemble de données. Un arbre de décision se compose d'un ensemble de nœuds qui représentent les attributs ou caractéristiques de l'ensemble de données, ainsi que d'un ensemble de branches qui relient ces nœuds et représentent les décisions prises en fonction de ces attributs.

3.1 Decision Tree avec la méthode de Holdout:

```
#Hold out
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score , confusion_matrix,
precision_score, recall_score, classification_report
from sklearn.model_selection import
train_test_split DT = DecisionTreeClassifier()
X1, X2, y1, y2 = train_test_split(X,y, random_state= np.random,
train_size=0.7)
y_pred=DT.fit(X1,y1).predict(X2
)
print(accuracy_score(y2,y_pred)
)
```

```
0.8043478260869565
```

La précision du modèle est évaluée en utilisant la fonction accuracy_score() de Scikit-learn, qui calcule le taux de classification correcte, en comparant les valeurs réelles de l'ensemble de test (y2) avec les valeurs prédites (y_pred) par le modèle. Le résultat est 0,83, cela signifie que le modèle a prédit correctement la classe de 83% des échantillons dans l'ensemble de test.

3.2 Decision Tree avec Cross Validation:

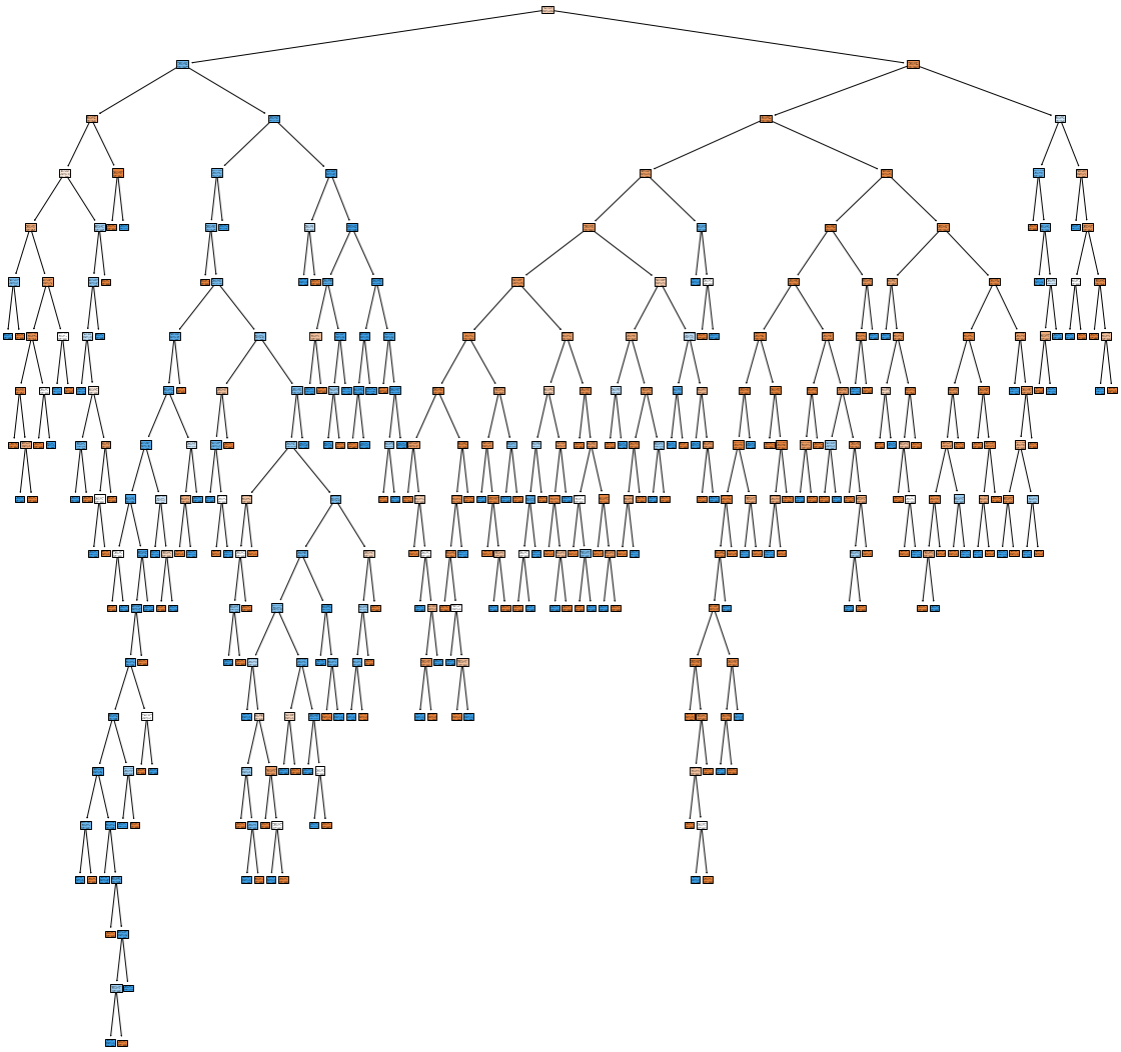
```
#5 fold with cross validation
from sklearn.model_selection import cross_val_score
cross_val_score(DT, X, y, cv=5)
```

```
array([0.83682984, 0.83682984, 0.81351981, 0.8041958 , 0.80607477])
```

Ici, on a utilisé la validation croisée en 5 folds, c'est une est donc une technique plus fiable pour évaluer les performances d'un modèle que la technique de validation "holdout", car elle permet d'utiliser efficacement les données disponibles et fournit une estimation plus précise de la performance du modèle. Les résultats de la validation croisée en 5-fold donne des précisions similaires, même si les valeurs de précision sont différentes.

C'est une bonne indication car cela vaut dire que données sont distribuées de manière uniforme et homogène dans l'ensemble de données, de sorte que les sous-ensembles de données générés par la validation croisée en 5-fold sont similaires en termes de distribution des données.

```
from sklearn import tree
fig = plt.figure(figsize=(20,20))
_ = tree.plot_tree(DT,
                   feature_names = ['Vmag', 'Plx', 'e_Plx', 'B-V',
'Amag'],                                    class_name
                                            s =
                                            ['Dwarf',
                                            'Giant'],
                                            filled=Tru
                                            e)
```



3.3 Decision Tree avec Grid Search:

On va effectuer une recherche de grille (grid search) pour optimiser les hyperparamètres du modèle. La recherche de grille consiste à tester toutes les combinaisons possibles d'hyperparamètres spécifiées dans le dictionnaire 'params', et à sélectionner la combinaison qui donne la meilleure performance en termes de précision (accuracy).

La fonction GridSearchCV de Scikit-learn est utilisée pour effectuer la recherche de grille, et elle prend plusieurs paramètres en entrée:

*Le premier paramètre est l'estimateur à optimiser, dans ce cas, DT2, un modèle d'arbre de décision.

*Le deuxième paramètre est le dictionnaire de paramètres à tester, qui comprend trois hyperparamètres différents pour ce modèle d'arbre de décision : la profondeur maximale de l'arbre (max_depth), le nombre minimum d'échantillons pour les feuilles (min_samples_leaf) et le critère de mesure de qualité de division de l'arbre (criterion).

*Le troisième paramètre est le nombre de "folds" dans la validation croisée, ici 10.

*Le quatrième paramètre spécifie la mesure de performance à utiliser pour évaluer chaque combinaison d'hyperparamètres, ici la précision (accuracy).

```
#Grid search
from sklearn.model_selection import GridSearchCV
#creat a new KNN model
DT2 =
DecisionTreeClassifier()
params = {
'max_depth': [2, 3, 5, 10, 20],
'min_samples_leaf': [5, 10, 20, 50, 100],
'criterion': ["gini", "entropy"]
}
grid = GridSearchCV(DT2, params, cv = 10, scoring = 'accuracy')
grid.fit(X,y)
print(grid.best_score_)
print(grid.best_params_)
print(grid.best_estimator_
)
```

```
0.8829297978700282
{'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 20}
DecisionTreeClassifier(max_depth=5, min_samples_leaf=20)
```

```
from sklearn.tree import export_graphviz
import graphviz
dot_data = tree.export_graphviz(grid.best_estimator_, feature_names =
['Vmag', 'Plx', 'e_Plx', 'B-V', 'Amag'], class_names = ['Dwarf',
'Giant'],
                                filled=True,
                                rounded=True,
                                special_characters=True)
graph =
graphviz.Source(dot_data) graph
```

Part IV

1. Filter Methods:

1.1 Mutual Information Score:

Ici, on doit identifier les caractéristiques les plus importantes dans notre ensemble de données et ainsi permettre de les sélectionner pour la modélisation, en éliminant les caractéristiques (features) qui ne contribuent pas de manière significative à la

classification de la variable cible(TargetClass). Cela peut également aider à améliorer la performance du modèle et à réduire les temps de calcul en éliminant les caractéristiques inutiles.

Pour cela, on va utiliser la fonction mutual_info_classif de la bibliothèque scikit-learn pour calculer les scores d'information mutuelle entre les caractéristiques (features) et la variable cible (TargetClass).

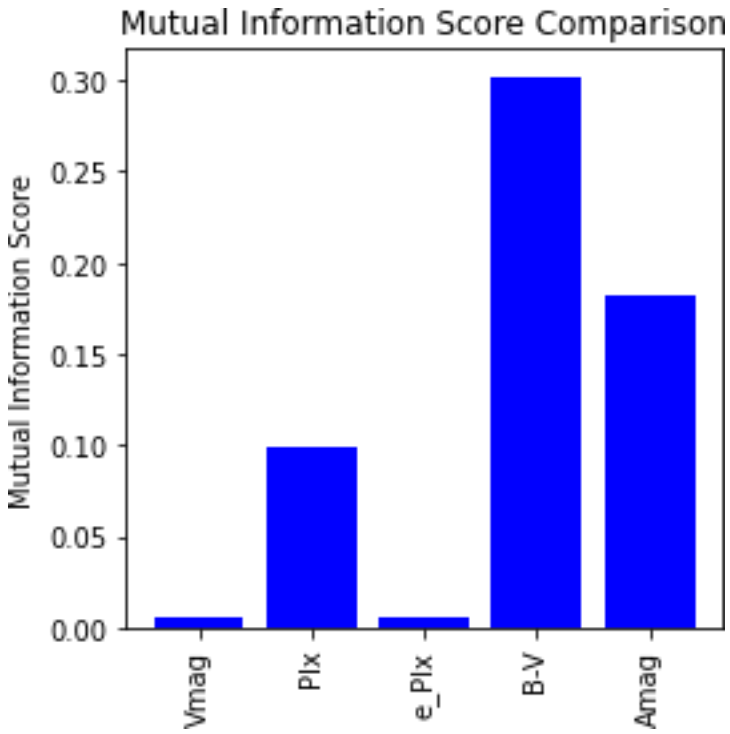
Les scores d'information mutuelle mesurent l'importance de chaque caractéristique pour la classification de la variable cible. Les caractéristiques qui ont des scores élevés sont considérées comme étant plus importantes pour la classification que celles avec des scores plus faibles.

```
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
feature_names = ['Vmag', 'Plx', 'e_Plx', 'B-V',
'Amag'] X = data.drop(['TargetClass'], axis = 1)
y = data["TargetClass"]
from sklearn.feature_selection import mutual_info_classif
MI_score = mutual_info_classif(X, y, random_state=0)
for feature in zip(feature_names, MI_score):
  print(feature)
```

```
('Vmag', 0.00547000866992664)
('Plx', 0.09860389118057045)
('e_Plx', 0.004931652792147023)
('B-V', 0.3022223698031603)
('Amag', 0.18251798454327184)
```

On peut bien voir que B-V, Amag, et Plx sont plus importants. On peut voir cela aussi dans la matrice de corrélation auparavant.

```
plt.figure(figsize=(4,4))
plt.bar(x = feature_names, height=MI_score, color='blue')
plt.xticks(rotation='vertical')
plt.ylabel('Mutual Information Score')
plt.title('Mutual Information Score Comparison')
plt.show()
```



2. Wrapper Methods:

2.1 Exhaustive Feature Selection:

```
!pip install mlxtend
```

```
!pip install sklearn
```

Les méthodes de Wrapping sélectionnent les feautures en entraînant le modèle sur chaque sous-ensemble possible et en évaluant sa performance à l'aide d'une métrique. Tandis que les méthodes filtrages sélectionnent les fonctionnalités en calculant des mesures statistiques telles que l'information mutuelle.

```
import joblib
import sys
sys.modules['sklearn.externals.joblib'] =
joblib import mlxtend
from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
efs = EFS(estimator=knn,
      min_features=1,
      max_features=4,
      scoring='accuracy'
      , cv=5)
efs = efs.fit(X, y)
print('Best accuracy score: %.2f' % efs.best_score_)
print('Best subset (corresponding names):', efs.best_feature_names_)
```

```
Features: 30/30

Best accuracy score: 0.88
Best subset (corresponding names): ('e_Plx', 'B-V', 'Amag')
```

La meilleure combinaison de caractéristiques pour classifier les étoiles aves le modèle KNN est obtenue en sélectionnant les features nommées ['e_Plx', 'B-V', 'Amag']. Cette combinaison donne une précision de prédiction de 0.88, ce qui est le meilleur score de précision atteint parmi toutes les combinaisons évaluées.

```
# Transform the dataset
X_new =
efs.transform(X) #
Print the results
print('Number of features before transformation:
{}'.format(X.shape[1]))
print('Number of features after transformation:
{}'.format(X_new.shape[1]))
```

```
Number of features before transformation: 5
Number of features after transformation: 3
```

La sélection exhaustive de caractéristiques a identifié un sous-ensemble de 3 caractéristiques qui donnent la meilleure précision de prédiction avec le modèle KNN. Ces 3 caractéristiques sont sélectionnées à partir des 5 caractéristiques initiales dans le jeu de données d'origine (X). Le nouveau jeu de données transformé (X_new) contient donc ces 3 caractéristiques uniquement. Le nombre de caractéristiques dans le jeu de données d'origine était 5, alors que le nouveau jeu de données contient seulement 3 caractéristiques. On a maintenant 3 features après la sélection.

Maintenant, on va afficher les performances de chaque sous-ensemble de caractéristiques évalué par l'algorithme EFS.

```
# Show the performance of each subset of features
efs_results = pd.DataFrame.from_dict(efs.get_metric_dict()).T
efs_results.sort_values(by='avg_score', ascending=True, inplace=True)
efs_results
```

```
     feature_idx                                          cv_scores
avg_score \
0           (0,)  [0.5594405594405595, 0.5337995337995338 0.559...
                                                        ,
0.551308
6         (0, 2)  [0.5291375291375291, 0.5874125874125874 0.594...
                                                        ,
0.566236
2           (2,)  [0.5524475524475524, 0.5944055944055944 0.543...
                                                        ,
0.573233
1           (1,)  [0.6643356643356644, 0.6480186480186481 0.627...
                                                        ,
0.65299
9         (1, 2)  [0.7156177156177156, 0.7132867132867133 0.713...
                                                        ,
0.708942
15     (0, 1, 2)  [0.7435897435897436, 0.7435897435897436 0.722...
```

,

0.734133

```
5         (0, 1)  [0.7668997668997669, 0.7552447552447552 0.769...
                                                         ,
0.757455
4           (4,)  [0.7645687645687645, 0.7482517482517482 0.778...
                                                         ,
0.757456
11        (1, 4)  [0.7668997668997669, 0.7645687645687645 0.776...
                                                         ,
0.76352
13        (2, 4)  [0.7692307692307693, 0.7715617715617715 0.769...
                                                         ,
0.767721
17     (0, 1, 4)  [0.7878787878787878, 0.7832167832167832 0.790...
                                                         ,
0.772839
8         (0, 4)  [0.7855477855477856, 0.7762237762237763 0.790...
                                                         ,
0.772841
26 (0, 1, 2,  4)  [0.7668997668997669, 0.7832167832167832 0.778...
                                                         ,
0.774712
19     (0, 2, 4)  [0.7762237762237763, 0.7785547785547785 0.783...
0.777047
                                                         ,
22         2, 4)  [0.7785547785547785, 0.7645687645687645 0.792...
(1,                                                      ,
0.778446
12        (2  3)  [0.8461538461538461, 0.8554778554778555 0.829...
          ,                                              ,
0.836752
7         (0  3)  [0.8578088578088578, 0.8461538461538461 0.822...
          ,                                              ,
0.837215
3           (3,)  [0.8461538461538461, 0.8531468531468531 0.848...
                                                         ,
0.837683
18     (0, 2, 3)  [0.8391608391608392, 0.8508158508158508 0.825...
                                                         ,
0.837686
10        (1, 3)  [0.8531468531468531, 0.8648018648018648 0.853...
                                                         ,
0.853074
16     (0, 1, 3)  [0.8717948717948718, 0.8717948717948718 0.855...
                                                         ,
0.864268
21     (1  2, 3)  [0.8531468531468531, 0.8764568764568764 0.869...
       ,                                                 ,
0.864742
25 (0, 1, 2,  3)  [0.8624708624708625, 0.8717948717948718 0.857...
                                                         ,
0.866139
27 (0, 1, 3,  4)  [0.8648018648018648, 0.8717948717948718 0.871...
                                                         ,
0.866607
29 (1, 2, 3,  4)  [0.8648018648018648, 0.8671328671328671 0.874...
                                                         ,
0.86847
14        (3, 4)  [0.8671328671328671, 0.8951048951048951 0.871...
```

,
0.871732
23 (1, 3, 4) [0.8671328671328671, 0.8717948717948718 0.876...
,
0.871738
20 (0, 3, 4) [0.8717948717948718, 0.8811188811188811, 0.871...
0.872202
28 (0, 2, 3, 4) [0.8741258741258742, 0.8811188811188811, 0.878...
0.875466
24 (2, 3, 4) [0.8694638694638694, 0.8881118881118881, 0.876...
0.876401

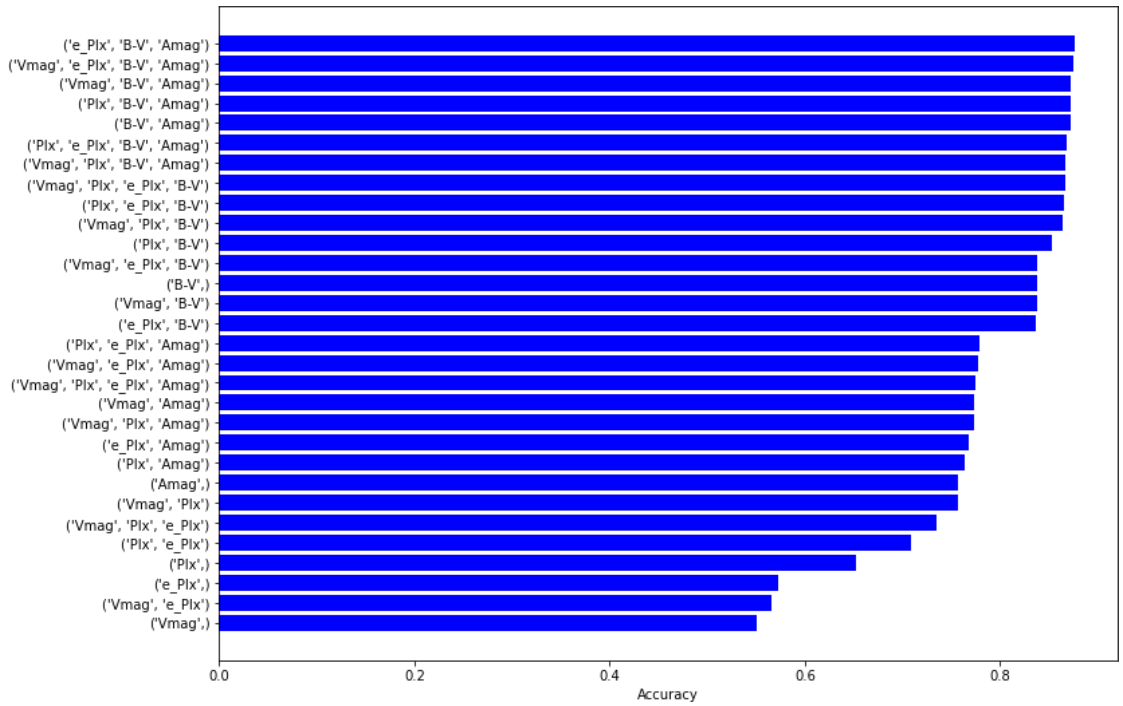
feature_names ci_bound std_dev std_err

0	(Vmag,)	0.01251	0.009736	0.004868
6	(Vmag, e_Plx)	0.03287	0.025576	0.012788
2	(e_Plx,)	0.02732	0.021261	0.010631
1	(Plx,)	0.01835	0.014281	0.00714
9	(Plx, e_Plx)	0.01916	0.014914	0.007457
15	(Vmag, Plx, e_Plx)	0.01644	0.012794	0.006397
5	(Vmag, Plx)	0.01310	0.010197	0.005099
4	(Amag,)	0.01636	0.012731	0.006365
11	(Plx, Amag)	0.01105	0.008599	0.0043
13	(e_Plx, Amag)	0.00431	0.00336	0.00168
17	(Vmag, Plx, Amag)	0.02427	0.018889	0.009444
8	(Vmag, Amag)	0.02134	0.016605	0.008303
26	(Vmag, Plx, e_Plx, Amag)	0.01448	0.011268	0.005634
19	(Vmag, e_Plx, Amag)	0.00749	0.005832	0.002916
22	(Plx, e_Plx, Amag)	0.01494	0.011626	0.005813
12	(e_Plx, B-V)	0.01664	0.012951	0.006476
7	(Vmag, B-V)	0.01664	0.012948	0.006474
3	(B-V,)	0.01989	0.015476	0.007738
18	(Vmag, e_Plx, B-V)	0.01049	0.008165	0.004083
10	(Plx, B-V)	0.00881	0.006857	0.003429
16	(Vmag, Plx, B-V)	0.00965	0.007511	0.003756
21	(Plx, e_Plx, B-V)	0.01251	0.009737	0.004868
25	(Vmag, Plx, e_Plx, B-V)	0.00667	0.005196	0.002598
27	(Vmag, Plx, B-V, Amag)	0.00930	0.007241	0.00362
29	(Plx, e_Plx, B-V, Amag)	0.00404	0.003146	0.001573
14	(B-V, Amag)	0.01578	0.012283	0.006141
23	(Plx, B-V, Amag)	0.00673	0.005242	0.002621
20	(Vmag, B-V, Amag)	0.00765	0.005953	0.002977
28	(Vmag, e_Plx, B-V, Amag)	0.00525	0.004086	0.002043
24	(e_Plx, B-V, Amag)	0.00980	0.007627	0.003814

Le tableau est trié par ordre croissant de la précision de prédiction moyenne (avg_score) des sous-ensembles de caractéristiques, afin de faciliter la visualisation des sous-ensembles qui donnent les moins bonnes performances.

Le sous-ensemble de caractéristiques qui a donné la meilleure précision de prédiction moyenne est celui qui a inclus les caractéristiques ['e_Plx', 'B-V', 'Amag'].

```
fig, ax =
plt.subplots(figsize=(12,9)) y_pos =
np.arange(len(efs_results))
ax.barh(y_pos,
efs_results['avg_score'],
color='blue')
ax.set_yticks(y_pos)
ax.set_yticklabels(efs_results['feature_names'])
ax.set_xlabel('Accuracy')
plt.show()
```



Le sous-ensemble de caractéristiques qui a donné la meilleure précision de prédiction moyenne (environ 0,85) est représenté par la barre horizontale la plus longue, correspondant au sous-ensemble ['e_Plx', 'B-V', 'Amag'].