#Importing and Prepocessing DataSets

Lien pour importer des fichiers à partir du drive : https://neptune.ai/blog/google-colab-dealing-with-files

Le but de ce code est de monter le répertoire "drive" dans l'environnement de travail,car les données que nous souhaitons utiliser sont stockées dans un dossier situé sur le drive.

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

```
Drive already mounted at /content/drive/; to attempt to forcibly
remount, call drive.mount("/content/drive/", force_remount=True).
```

Lien des Datasets : https://www.football-data.co.uk/englandm.php Nous avons pris les données de 2007 à 2022. Nous n'avons pas pris les données de 2023 car elles sont incomplètes. De plus, le jeu de données de la saison 2014-2015 contient 381 matchs. On remarque que la ligne 380 pour les données de la saison 2014-2015 ne contient que des valeurs nulles. Nous pouvons supprimer cette ligne.

Les variables utilisées sont :

- HomeTeam
- Away Team
- B365H = Bet365 home win odds
- B365D = Bet365 draw odds
- B365A = Bet365 away win odds
- HS = Home Team Shots
- AS = Away Team Shots
- HST = Home Team Shots on Target
- AST = Away Team Shots on Target
- HC = Home Team Corners
- AC = Away Team Corners
- HF = Home Team Fouls Committed
- AF = Away Team Fouls Committed
- HY = Home Team Yellow Cards
- AY = Away Team Yellow Cards
- HR = Home Team Red Cards
- AR = Away Team Red Cards
- FTR = Full Time Result (H=Home Win, D=Draw, A=Away Win)

```
import pandas as pd

all_datasets = ['2007_2008','2008_2009','2009_2010', '2010_2011',
'2011_2012', '2012_2013', '2013_2014', '2014_2015',
                 '2015_2016', '2016_2017', '2017_2018', '2018_2019',
'2019_2020', '2020_2021', '2021_2022']
```

```
all_seasons_frames = []
for x in all_datasets:
  season =
pd.DataFrame(pd.read_csv('/content/drive/MyDrive/Projet_Intelligence_A
rtificielle /Data_{}.csv'.format(x)),
                                  columns=['HomeTeam','AwayTeam',
'B365H', 'B365D','B365A', 'HS', 'AS', 'HST',
                                           'AST', 'HC', 'AC', 'HF',
'AF', 'HY', 'AY', 'HR', 'AR', 'FTR'])
  if(x == '2014_2015') :
    season = season.drop(season[season.HomeTeam.isnull()].index)

  all_seasons_frames.append(pd.DataFrame(season))
```

Types de données

Nous allons déterminer le type de nos données

```
df_all_frames = pd.concat(all_seasons_frames)
print(df_all_frames.dtypes)
```

```
HomeTeam     object
AwayTeam     object
B365H       float64
B365D       float64
B365A       float64
HS          float64
AS          float64
HST         float64
AST         float64
HC          float64
AC          float64
HF          float64
AF          float64
HY          float64
AY          float64
HR          float64
AR          float64
FTR          object
dtype: object
```

Ainsi, on a 15 variables continues et 3 variables catégorielles (HomeTeam, AwayTeam, FTR).

Puis, on va visualiser nos données

```
df_all_frames.head()
```

```
        HomeTeam    AwayTeam  B365H  B365D  B365A    HS    AS  HST
AST    HC  \
0    Aston Villa   Liverpool   4.00   3.25   1.90  10.0  17.0  6.0
7.0   4.0
```

```
1          Bolton   Newcastle   2.50   3.20   2.75  13.0   7.0  9.0
5.0   4.0
2           Derby  Portsmouth   2.80   3.25   2.40  12.0  12.0  5.0
6.0   6.0
3         Everton       Wigan   1.66   3.40   5.50  12.0  14.0  8.0
4.0   6.0
4  Middlesbrough   Blackburn   2.37   3.25   2.87  10.0   4.0  6.0
4.0  13.0

    AC    HF    AF   HY   AY   HR   AR FTR
0  2.0  18.0  11.0  4.0  2.0  0.0  0.0   A
1  3.0  15.0  16.0  1.0  1.0  0.0  0.0   A
2  6.0  14.0  17.0  1.0  2.0  0.0  0.0   D
3  2.0   8.0  13.0  0.0  0.0  0.0  0.0   H
4  3.0  16.0  16.0  3.0  4.0  0.0  0.0   A
```

Étude des données

On va spécifier les données qui serviront à l'entraînement

```
frames_trainingSet = all_seasons_frames.copy()
frames_trainingSet.pop(len(frames_trainingSet) - 1)
trainingSet = pd.concat(frames_trainingSet)
```

Voici les statistiques descriptives pour les variables continues.

En particulier nous obtenons le total, la moyenne, l'écart type, le minimum et le maximum.

```
trainingSet.describe()
```

```
             B365H        B365D        B365A           HS           AS
\
count  5320.000000  5320.000000  5320.000000  5320.000000  5320.000000

mean      2.840320     4.077624     4.966761    13.891541    11.145113

std       2.050143     1.236430     4.265884     5.447461     4.706581

min       1.060000     3.000000     1.120000     0.000000     0.000000

25%       1.660000     3.400000     2.380000    10.000000     8.000000

50%       2.200000     3.600000     3.500000    13.000000    11.000000

75%       3.100000     4.200000     5.500000    17.000000    14.000000

max      23.000000    17.000000    41.000000    43.000000    30.000000


               HST          AST           HC           AC           HF
\
```

count	5320.000000	5320.000000	5320.000000	5320.000000	5320.000000
mean	5.975564	4.782331	6.008083	4.783835	10.802632
std	3.411460	2.864422	3.129428	2.738987	3.497985
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	3.000000	3.000000	4.000000	3.000000	8.000000
50%	5.000000	4.000000	6.000000	4.000000	11.000000
75%	8.000000	6.000000	8.000000	6.000000	13.000000
max	24.000000	20.000000	20.000000	19.000000	33.000000

	AF	HY	AY	HR	AR
count	5320.000000	5320.000000	5320.000000	5320.000000	5320.000000
mean	11.224060	1.463910	1.755639	0.060902	0.084211
std	3.666357	1.192749	1.273793	0.248427	0.290306
min	1.000000	0.000000	0.000000	0.000000	0.000000
25%	9.000000	1.000000	1.000000	0.000000	0.000000
50%	11.000000	1.000000	2.000000	0.000000	0.000000
75%	14.000000	2.000000	3.000000	0.000000	0.000000
max	26.000000	7.000000	9.000000	2.000000	2.000000

On remarque que le minimum pour les tirs (HS, AS) est de 0. Ceci est très peu probable car en général durant un match chacune des équipes a au moins une opportunité de tir même si ceux-ci ne sont pas cadrés. Nous pouvons observer de plus près le(s) match(s) en question.

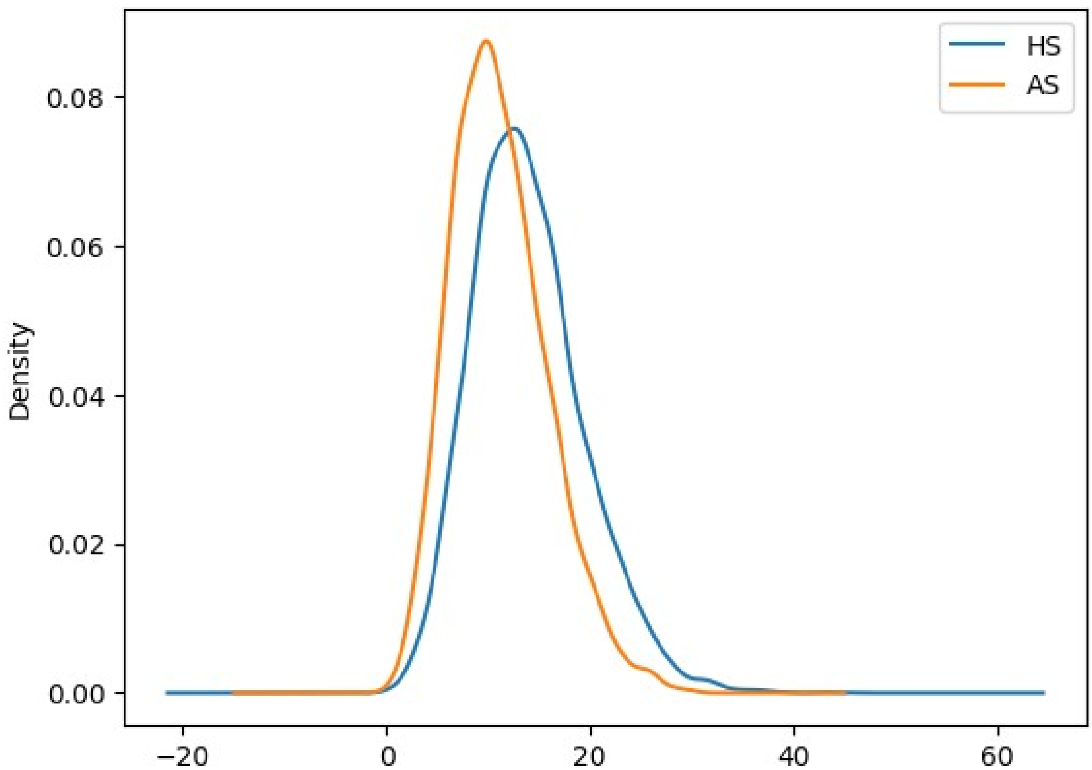
Voici les statistiques descriptives pour les variables catégorielles

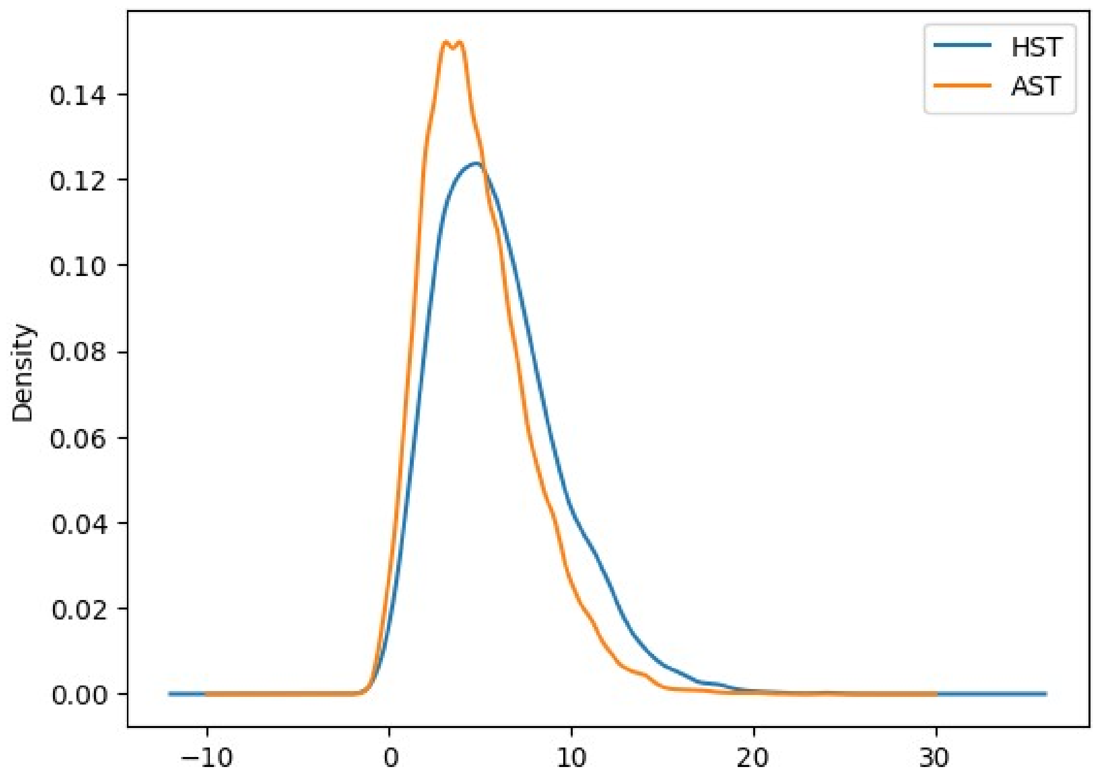
```
trainingSet.describe(include=[object])
```

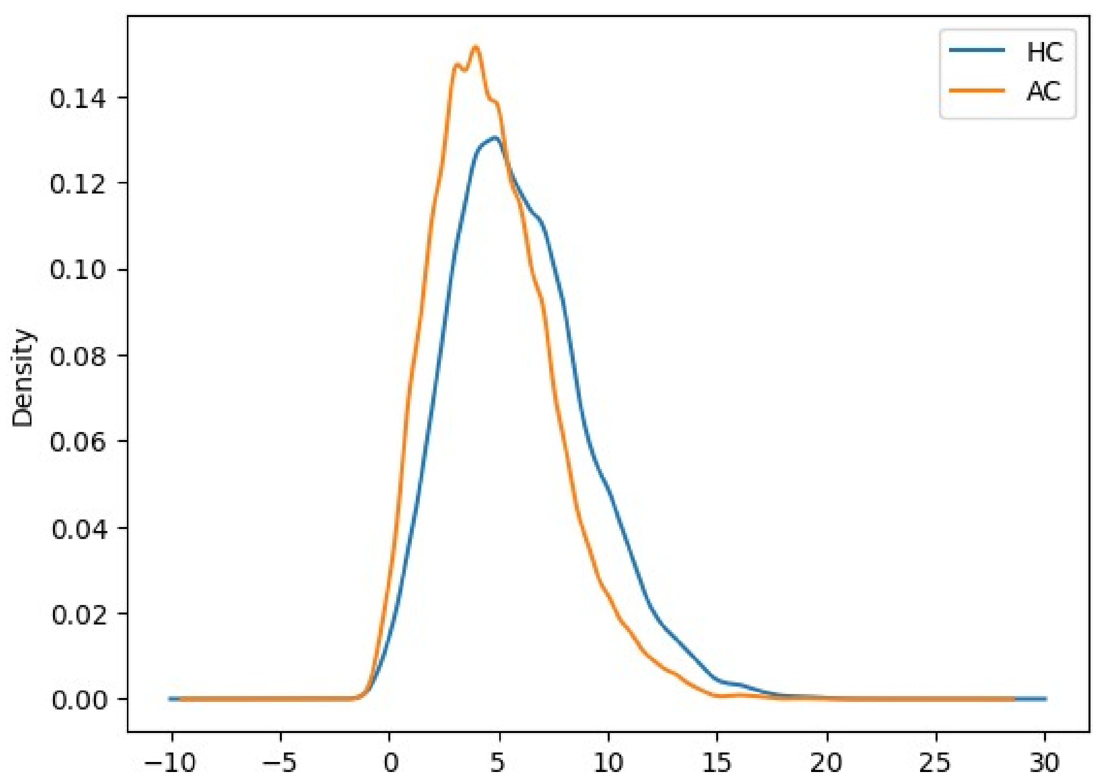
```
         HomeTeam   AwayTeam   FTR
count        5320       5320  5320
unique         39         39     3
top     Liverpool  Liverpool     H
freq          266        266  2423
```

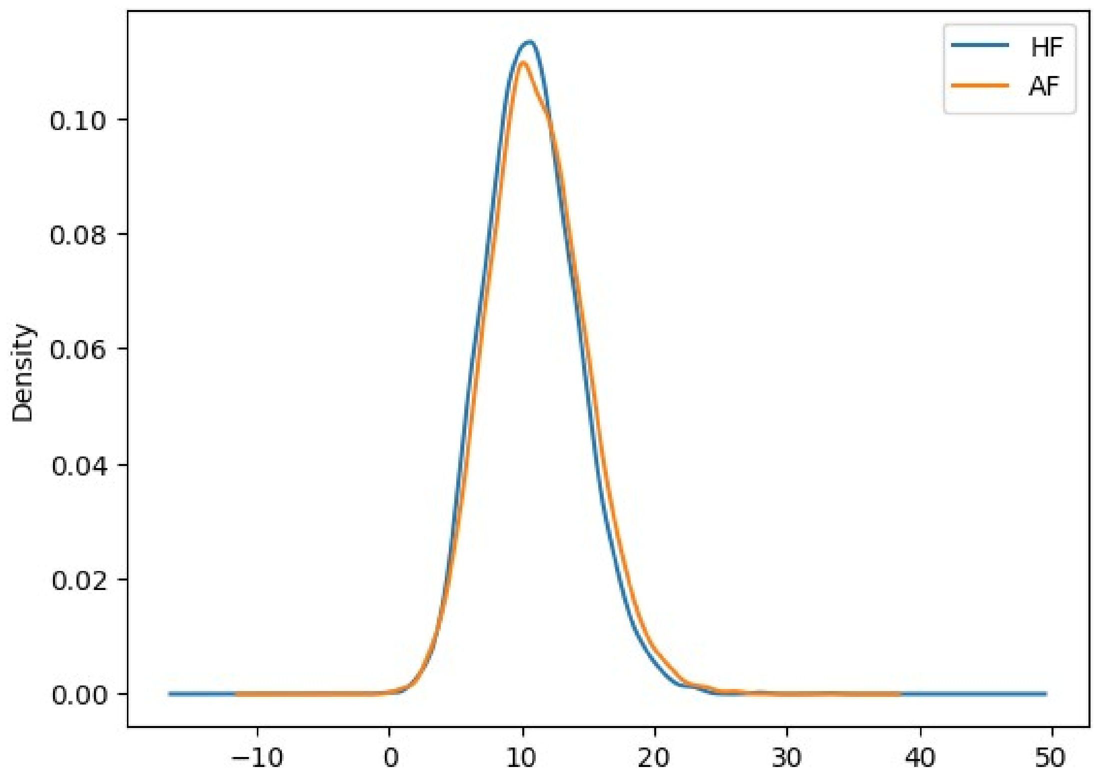
On va comparer la distribution de certaines variables entre équipe à domicile et équipe à l'extérieur

```
comparaisons = [['HS', 'AS'],['HST', 'AST'], ['HC', 'AC'],['HF',
'AF']]
for x in comparaisons :
  trainingSet.loc[:,x].plot.kde()
```




HST
AST
Density
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
−10
0
10
20
30


HC
AC
Density
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
−10
−5
0
5
10
15
20
25
30

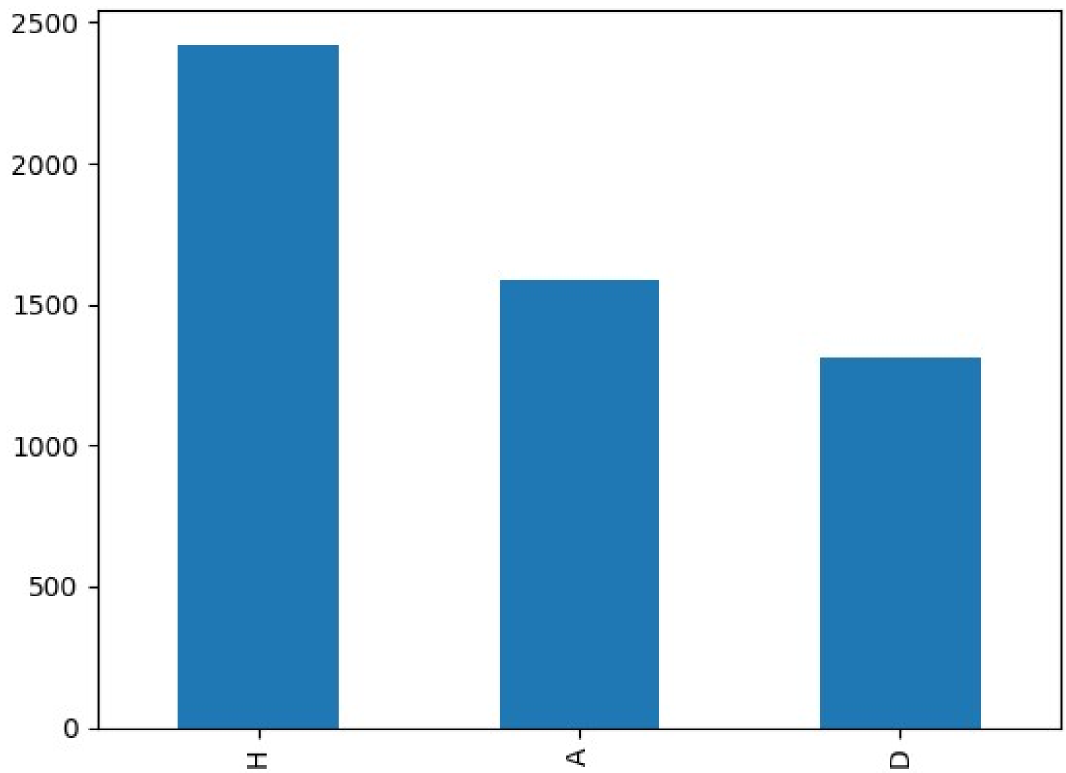


- On remarque que en ce concerne les occasion de tirs et de corners, la courbe pour les équipe à domicile est décalée vers la droite. Ceci suggère que les équipes jouant à domicile ont tendance à avoir plus d'occasion de tirs et de corners, on peut en déduire qu'elles adoptent un jeu offensif. Par contre, nous devons faire un test statistique pour savoir si cette différence est significative.
- En ce qui concerne les fautes, les distributions se superposent presque parfaitement ce qui ne nous donne pas vraiment d'intuition sur la stratégie de jeu des équipes.

Voici la distribution des résultats des matchs en fonction de H,A et D.

```
trainingSet['FTR'].value_counts().plot(kind='bar')
```

```
<Axes: >
```



- Les équipes jouant à domiciles gagnent environs 1.5 fois plus souvent que leurs adversaires jouant à l'extérieur.
- Les matchs nulles apparaissent en plus faibles proportions.

On va transformer les variables catégorielles.

En premier, on va transformer les variables pour les équipes en variables binaires.

Puis, on va transformer la variable pour le résultat du match en valeurs numériques (0: A, 1: D, 2: H

```
from sklearn.preprocessing import LabelEncoder

df_all_frames_modif = pd.get_dummies(df_all_frames,
columns=['HomeTeam', 'AwayTeam'])

label_make = LabelEncoder()
df_all_frames_modif["FTR_code"] =
label_make.fit_transform(df_all_frames_modif["FTR"])
df_all_frames_modif[["FTR", "FTR_code"]].head(11)
```

```
   FTR  FTR_code
0    A         0
1    A         0
2    D         1
3    H         2
4    A         0
```

```
5    H         2
6    A         0
7    H         2
8    H         2
9    D         1
10   A         0
```

```
df_all_frames_modif = df_all_frames_modif.drop(['FTR'], axis=1)

season_21_22_modif = df_all_frames_modif[5320:5701]

trainingSet_modif = df_all_frames_modif[:5320]
```

Modèles de base pour des données de football

On va évaluer la performance de quelques modèles de base pour les données de la saison 2021-2022:

- Premier modèle : toutes les prédictions sont à H (HomeWin)

```
import copy
finalsetaSet = copy.deepcopy(season_21_22_modif)
finalset['pred'] = 2
err_finalset_mod1 = (sum(finalset.FTR_code!
=finalset.pred)/len(finalset))*100
print("err_finalset_mod1: ", err_finalset_mod1, "%")
```

```
err_finalset_mod1:  57.10526315789474 %
```

- Deuxième modèle : toutes les prédictions sont à A (AwayWin)

```
finalset = copy.deepcopy(season_21_22_modif)
finalset['pred'] = 0
err_finalset_mod2 = (sum(finalset.FTR_code!
=finalset.pred)/len(finalset))*100
print("err_finalset_mod2: ", err_finalset_mod2, "%")
```

```
err_finalset_mod2:  66.05263157894737 %
```

- Troisième modèle : toutes les prédictions sont à D (Draw)

```
finalset = copy.deepcopy(season_21_22_modif)
finalset['pred'] = 1
err_finalset_mod3 = (sum(finalset.FTR_code!
=finalset.pred)/len(finalset))*100
print("err_finalset_mod3: ", err_finalset_mod3, "%")
```

```
err_finalset_mod3:  76.84210526315789 %
```

Splitting training and data sets

```
import numpy as np
from sklearn.model_selection import train_test_split

df_x = pd.DataFrame(trainingSet_modif.iloc[:,0:95])
x = pd.DataFrame(df_x).to_numpy()
y = np.array(trainingSet_modif.iloc[:,95])

X_train, X_test, y_train, y_test = train_test_split(x,y, shuffle=True,
test_size=0.2, random_state=1234)
```

MLP

Finding the best netwotk depth

On va fixer la profondeur du réseau et on évalue la taille optimale des couches avec notre échantillon de validation (testset dans ce cas)

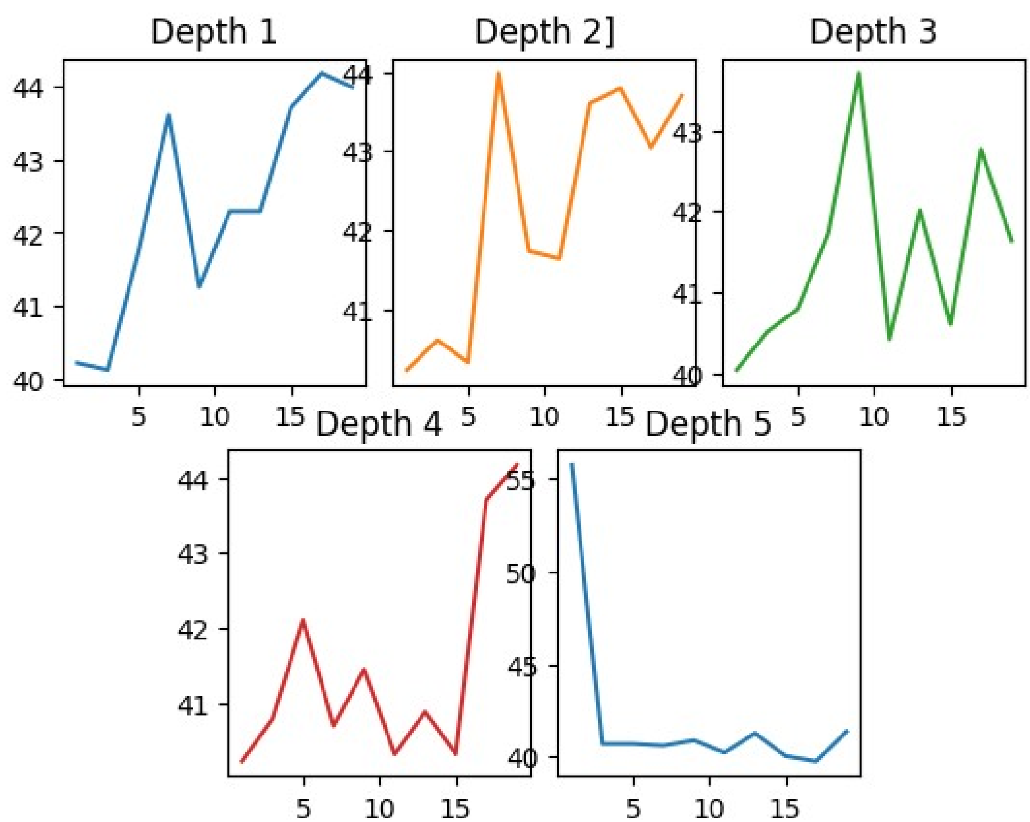
On cherche un optimum local en considérant 5 profondeurs range(1, 2, 3, 4, 5) et des tailles dans range(1, 21, 2) (même taille entre les couches d'un même modèle)

```
import warnings
from sklearn.exceptions import ConvergenceWarning

with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=ConvergenceWarning)

    from sklearn.neural_network import MLPClassifier

    c = ['tailleCouche', 'error_nn_test']
    df1 = pd.DataFrame(columns=c)
    df2 = pd.DataFrame(columns=c)
    df3 = pd.DataFrame(columns=c)
    df4 = pd.DataFrame(columns=c)
    df5 = pd.DataFrame(columns=c)


    for a in range(1, 21, 2):

      clf1 = MLPClassifier(hidden_layer_sizes=(a,),
                           activation='logistic',
                           solver='lbfgs',
                           random_state=0,
                           max_iter=500,
                           tol=1e-7).fit(X_train, y_train)

      clf2 = MLPClassifier(hidden_layer_sizes=(a,a,),
                           activation='logistic',
```

```
                            solver='lbfgs',
                            random_state=0,
                            max_iter=500,
                            tol=1e-7).fit(X_train, y_train)

      clf3 = MLPClassifier(hidden_layer_sizes=(a,a,a,),
                            activation='logistic',
                            solver='lbfgs',
                            random_state=0,
                            max_iter=500,
                            tol=1e-7).fit(X_train, y_train)

      clf4 = MLPClassifier(hidden_layer_sizes=(a,a,a,a,),
                            activation='logistic',
                            solver='lbfgs',
                            random_state=0,
                            max_iter=500,
                            tol=1e-7).fit(X_train, y_train)

      clf5 = MLPClassifier(hidden_layer_sizes=(a,a,a,a,a,),
                              activation='logistic',
                              solver='lbfgs',
                              random_state=0,
                              max_iter=500,
                              tol=1e-7).fit(X_train, y_train)

      df1 = pd.concat([df1,pd.DataFrame([[a, (1 - clf1.score(X_test,
y_test))*100 ]], columns=c)], axis = 0)
      df2 = pd.concat([df2,pd.DataFrame([[a, (1 - clf2.score(X_test,
y_test))*100 ]], columns=c)], axis = 0)
      df3 = pd.concat([df3,pd.DataFrame([[a, (1 - clf3.score(X_test,
y_test))*100 ]], columns=c)], axis = 0)
      df4 = pd.concat([df4,pd.DataFrame([[a, (1 - clf4.score(X_test,
y_test))*100 ]], columns=c)], axis = 0)
      df5 = pd.concat([df5,pd.DataFrame([[a, (1 - clf5.score(X_test,
y_test))*100 ]], columns=c)], axis = 0)


import matplotlib.pyplot as plt
import matplotlib as mpl
fig = plt.figure()
spec = mpl.gridspec.GridSpec(ncols=6, nrows=2)

ax1 = fig.add_subplot(spec[0,0:2])
ax2 = fig.add_subplot(spec[0,2:4])
ax3 = fig.add_subplot(spec[0,4:])
ax4 = fig.add_subplot(spec[1,1:3])
ax5 = fig.add_subplot(spec[1,3:5])

ax1.plot(df1.tailleCouche, df1.error_nn_test)
```

```
ax1.set_title('Depth 1',)
ax2.plot(df2.tailleCouche, df2.error_nn_test, 'tab:orange')
ax2.set_title('Depth 2]')
ax3.plot(df3.tailleCouche, df3.error_nn_test, 'tab:green')
ax3.set_title('Depth 3')
ax4.plot(df4.tailleCouche, df4.error_nn_test, 'tab:red')
ax4.set_title('Depth 4')
ax5.plot(df5.tailleCouche, df5.error_nn_test)
ax5.set_title('Depth 5')
```

```
Text(0.5, 1.0, 'Depth 5')
```



MLP with activation relu

```
clf = MLPClassifier(hidden_layer_sizes=(3,3,3,),
                          activation='relu',
                          solver='lbfgs',
                          random_state=0,
                          max_iter=500,
                          tol=1e-7).fit(X_train, y_train)

pred_nn_train = clf.predict(X_train)
pred_nn_test = clf.predict(X_test)

error_nn_train = (1 - clf.score(X_train, y_train))*100
error_nn_test =  (1 - clf.score(X_test, y_test))*100
```

```python
print("Error train MLPClassifier: ", error_nn_train, "%")
print("Error test MLPClassifier: ",  error_nn_test, "%")
```

```
Error train MLPClassifier:  38.88627819548872 %
Error test MLPClassifier:  40.22556390977443 %

/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:541: ConvergenceWarning: 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
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res,
self.max_iter)
```

MLP with activation tanh

```python
clf = MLPClassifier(hidden_layer_sizes=(3,3,3,),
                          activation='tanh',
                          solver='lbfgs',
                          random_state=0,
                          max_iter=500,
                          tol=1e-7).fit(X_train, y_train)

pred_nn_train = clf.predict(X_train)
pred_nn_test = clf.predict(X_test)

error_nn_train = (1 - clf.score(X_train, y_train))*100
error_nn_test =  (1 - clf.score(X_test, y_test))*100

print("Error train MLPClassifier: ", error_nn_train, "%")
print("Error test MLPClassifier: ",  error_nn_test, "%")
```

```
Error train MLPClassifier:  38.32236842105263 %
Error test MLPClassifier:  40.22556390977443 %

/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:541: ConvergenceWarning: 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
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res,
self.max_iter)
```

MLP with activation logistic and solver lbfgs

```
clf = MLPClassifier(hidden_layer_sizes=(3,3,3,),
                          activation='logistic',
                          solver='lbfgs',
                          random_state=0,
                          max_iter=500,
                          tol=1e-7).fit(X_train, y_train)

pred_nn_train = clf.predict(X_train)
pred_nn_test = clf.predict(X_test)

error_nn_train = (1 - clf.score(X_train, y_train))*100
error_nn_test =  (1 - clf.score(X_test, y_test))*100

print("Error train MLPClassifier: ", error_nn_train, "%")
print("Error test MLPClassifier: ",  error_nn_test, "%")
```

```
Error train MLPClassifier:  38.580827067669176 %
Error test MLPClassifier:  40.50751879699248 %
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/neural_network/
_multilayer_perceptron.py:541: ConvergenceWarning: 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
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res,
self.max_iter)
```

MLP with activation logistic and adam

```
clf = MLPClassifier(hidden_layer_sizes=(3,3,3,),
                          activation='logistic',
                          solver='adam',
                          random_state=0,
                          max_iter=500,
                          tol=1e-7).fit(X_train, y_train)

pred_nn_train = clf.predict(X_train)
pred_nn_test = clf.predict(X_test)

error_nn_train = (1 - clf.score(X_train, y_train))*100
error_nn_test =  (1 - clf.score(X_test, y_test))*100

print("Error train MLPClassifier: ", error_nn_train, "%")
print("Error test MLPClassifier: ",  error_nn_test, "%")
```

```
Error train MLPClassifier:  37.59398496240601 %
Error test MLPClassifier:  39.28571428571429 %
```

Net

On a utilisé l'article suivant : https://medium.com/@andreluiz_4916/pytorch-neural-networks-to-predict-matches-results-in-soccer-championships-part-ii-3d02b2ddd538

```
import torch
class Net(torch.nn.Module):
    def __init__(self, input_size, hidden_size):
        super(Net, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.fc1 = torch.nn.Linear(self.input_size, self.hidden_size)
        self.relu = torch.nn.ReLU()
        self.fc2 = torch.nn.Linear(self.hidden_size, 1)
        self.sigmoid = torch.nn.Sigmoid()
    def forward(self, x):
        hidden = self.fc1(x)
        sig = self.sigmoid(hidden)
        relu = self.relu(sig)
        output = self.fc2(relu)
        output = self.sigmoid(output)
        return output
```

```
#convert to tensors
training_input = torch.FloatTensor(X_train)
training_output = torch.FloatTensor(y_train)
test_input = torch.FloatTensor(X_test)
test_output = torch.FloatTensor(y_test)
```

```
input_size = training_input.size()[1] # number of features selected
hidden_size = 20 # number of nodes/neurons in the hidden layer
model = Net(input_size, hidden_size) # create the model
criterion = torch.nn.BCELoss() # works for binary classification
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)
```

```
model.eval()
y_pred = model(test_input)
before_train = criterion(y_pred.squeeze(), test_output)
print('Test loss before training' , before_train.item())
```

```
Test loss before training 0.7460142970085144
```

```
model.train()
epochs = 500
errors = []
training_input
y_pred
def closure():
    optimizer.zero_grad()
    y_pred = model(training_input)
    loss = criterion(y_pred.squeeze(), training_output)
```

```
    loss.backward()
    return loss

for epoch in range(epochs):
    optimizer.step(closure)
```

```
model.eval()
y_pred = model(test_input)
after_train = criterion(y_pred.squeeze(), test_output)
print('Test loss after Training' , after_train.item())
```

```
Test loss after Training -3.0839781761169434
```

```
# Pred saison 2021-2022
finalset = season_21_22_modif.drop(['FTR_code'], axis=1)
finalset = finalset.drop(['pred'], axis=1)
finalset = torch.FloatTensor(pd.DataFrame(finalset).to_numpy())
pred_season_21_22 = model(finalset)
pred_season_21_22 = pd.DataFrame(pred_season_21_22.detach().numpy())
pred_season_21_22.columns = ['pred']
pred_season_21_22.head()
```

```
       pred
0  0.996602
1  1.000000
2  0.999699
3  1.000000
4  1.000000
```

```
label_make = LabelEncoder()
season_21_22_eval = all_seasons_frames[len(all_seasons_frames)-1]
season_21_22_eval["FTR_code"] =
label_make.fit_transform(season_21_22_eval["FTR"])
season_21_22_eval['Pred'] = pred_season_21_22['pred'].astype(int)
```

```
#Calculons l'erreur de généralisation de notre modèle (pour les
données de la saison 2021_2022)

error_nn_season_21_22 =  (sum(season_21_22_eval.FTR_code!
=season_21_22_eval.Pred)/len(season_21_22_eval))*100

print("Erreur saison 2019-2020: ", error_nn_season_21_22, "%")
```

```
Erreur saison 2019-2020:  66.05263157894737 %
```

DNN

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import StandardScaler
```

```python
from tensorflow.keras.callbacks import EarlyStopping

#Normalize the input data

scaler = StandardScaler()
X_train3 = scaler.fit_transform(X_train)
X_test3 = scaler.transform(X_test)
y_train3 = y_train
y_test3 = y_test

#Define the model

model = Sequential([
Dense(128, activation='relu', input_shape=(X_train3.shape[1],)),
Dropout(0.2),
Dense(64, activation='relu'),
Dropout(0.2),
Dense(32, activation='relu'),
Dropout(0.2),
Dense(1, activation='softmax')
])

#Compile the model

model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

#Use early stopping to prevent overfitting

early_stop = EarlyStopping(monitor='val_loss', patience=5)

#Train the model

history = model.fit(X_train3, y_train3, epochs=50, batch_size=64,
validation_data=(X_test3, y_test3), callbacks=[early_stop])

#Evaluate the model

score_train = model.evaluate(X_train3, y_train3, verbose=0)
score_test = model.evaluate(X_test3, y_test3, verbose=0)
error_nn_train = (1 - score_train[1]) * 100
error_nn_test = (1 - score_test[1]) * 100

print("Error train DNN: ", error_nn_train, "%")
print("Error test DNN: ", error_nn_test, "%")
```

```
Epoch 1/50
67/67 [==============================] - 2s 7ms/step - loss: -4.4110 -
accuracy: 0.2446 - val_loss: -17.8163 - val_accuracy: 0.2547
```

```
Epoch 2/50
67/67 [==============================] - 0s 4ms/step - loss: -166.3313
- accuracy: 0.2446 - val_loss: -509.2274 - val_accuracy: 0.2547
Epoch 3/50
67/67 [==============================] - 0s 3ms/step - loss: -
1981.0226 - accuracy: 0.2446 - val_loss: -4282.3145 - val_accuracy:
0.2547
Epoch 4/50
67/67 [==============================] - 0s 4ms/step - loss: -
10316.4961 - accuracy: 0.2446 - val_loss: -17918.6660 - val_accuracy:
0.2547
Epoch 5/50
67/67 [==============================] - 0s 4ms/step - loss: -
34044.3320 - accuracy: 0.2446 - val_loss: -51822.9492 - val_accuracy:
0.2547
Epoch 6/50
67/67 [==============================] - 0s 4ms/step - loss: -
87321.0078 - accuracy: 0.2446 - val_loss: -120546.5625 - val_accuracy:
0.2547
Epoch 7/50
67/67 [==============================] - 0s 4ms/step - loss: -
185667.6094 - accuracy: 0.2446 - val_loss: -238490.4375 -
val_accuracy: 0.2547
Epoch 8/50
67/67 [==============================] - 0s 4ms/step - loss: -
343406.6250 - accuracy: 0.2446 - val_loss: -426003.0312 -
val_accuracy: 0.2547
Epoch 9/50
67/67 [==============================] - 0s 4ms/step - loss: -
583324.4375 - accuracy: 0.2446 - val_loss: -704157.3125 -
val_accuracy: 0.2547
Epoch 10/50
67/67 [==============================] - 0s 4ms/step - loss: -
926664.1875 - accuracy: 0.2446 - val_loss: -1084633.0000 -
val_accuracy: 0.2547
Epoch 11/50
67/67 [==============================] - 0s 4ms/step - loss: -
1415801.1250 - accuracy: 0.2446 - val_loss: -1610999.1250 -
val_accuracy: 0.2547
Epoch 12/50
67/67 [==============================] - 0s 4ms/step - loss: -
2058333.8750 - accuracy: 0.2446 - val_loss: -2291031.0000 -
val_accuracy: 0.2547
Epoch 13/50
67/67 [==============================] - 0s 3ms/step - loss: -
2894863.2500 - accuracy: 0.2446 - val_loss: -3146399.5000 -
val_accuracy: 0.2547
Epoch 14/50
67/67 [==============================] - 0s 3ms/step - loss: -
3887600.7500 - accuracy: 0.2446 - val_loss: -4223735.0000 -
```

```
val_accuracy: 0.2547
Epoch 15/50
67/67 [==============================] - 0s 4ms/step - loss: -
5171026.0000 - accuracy: 0.2446 - val_loss: -5520573.5000 -
val_accuracy: 0.2547
Epoch 16/50
67/67 [==============================] - 0s 4ms/step - loss: -
6659486.0000 - accuracy: 0.2446 - val_loss: -7068045.0000 -
val_accuracy: 0.2547
Epoch 17/50
67/67 [==============================] - 0s 4ms/step - loss: -
8431964.0000 - accuracy: 0.2446 - val_loss: -8928163.0000 -
val_accuracy: 0.2547
Epoch 18/50
67/67 [==============================] - 1s 8ms/step - loss: -
10679835.0000 - accuracy: 0.2446 - val_loss: -11061502.0000 -
val_accuracy: 0.2547
Epoch 19/50
67/67 [==============================] - 1s 9ms/step - loss: -
13070059.0000 - accuracy: 0.2446 - val_loss: -13542967.0000 -
val_accuracy: 0.2547
Epoch 20/50
67/67 [==============================] - 0s 7ms/step - loss: -
15800342.0000 - accuracy: 0.2446 - val_loss: -16402278.0000 -
val_accuracy: 0.2547
Epoch 21/50
67/67 [==============================] - 1s 8ms/step - loss: -
19059104.0000 - accuracy: 0.2446 - val_loss: -19630490.0000 -
val_accuracy: 0.2547
Epoch 22/50
67/67 [==============================] - 1s 9ms/step - loss: -
22774296.0000 - accuracy: 0.2446 - val_loss: -23261188.0000 -
val_accuracy: 0.2547
Epoch 23/50
67/67 [==============================] - 0s 6ms/step - loss: -
27096766.0000 - accuracy: 0.2446 - val_loss: -27475072.0000 -
val_accuracy: 0.2547
Epoch 24/50
67/67 [==============================] - 0s 4ms/step - loss: -
31608506.0000 - accuracy: 0.2446 - val_loss: -31970916.0000 -
val_accuracy: 0.2547
Epoch 25/50
67/67 [==============================] - 0s 4ms/step - loss: -
36900032.0000 - accuracy: 0.2446 - val_loss: -37113624.0000 -
val_accuracy: 0.2547
Epoch 26/50
67/67 [==============================] - 0s 4ms/step - loss: -
42702412.0000 - accuracy: 0.2446 - val_loss: -42775756.0000 -
val_accuracy: 0.2547
Epoch 27/50
```

```
67/67 [==============================] - 0s 6ms/step - loss: -
48880412.0000 - accuracy: 0.2446 - val_loss: -48919704.0000 -
val_accuracy: 0.2547
Epoch 28/50
67/67 [==============================] - 0s 6ms/step - loss: -
55964760.0000 - accuracy: 0.2446 - val_loss: -55722588.0000 -
val_accuracy: 0.2547
Epoch 29/50
67/67 [==============================] - 0s 5ms/step - loss: -
63447108.0000 - accuracy: 0.2446 - val_loss: -63051636.0000 -
val_accuracy: 0.2547
Epoch 30/50
67/67 [==============================] - 0s 6ms/step - loss: -
71870264.0000 - accuracy: 0.2446 - val_loss: -70970392.0000 -
val_accuracy: 0.2547
Epoch 31/50
67/67 [==============================] - 0s 5ms/step - loss: -
80001680.0000 - accuracy: 0.2446 - val_loss: -79606856.0000 -
val_accuracy: 0.2547
Epoch 32/50
67/67 [==============================] - 0s 6ms/step - loss: -
89624648.0000 - accuracy: 0.2446 - val_loss: -88930904.0000 -
val_accuracy: 0.2547
Epoch 33/50
67/67 [==============================] - 0s 6ms/step - loss: -
100462040.0000 - accuracy: 0.2446 - val_loss: -99102208.0000 -
val_accuracy: 0.2547
Epoch 34/50
67/67 [==============================] - 0s 6ms/step - loss: -
112332856.0000 - accuracy: 0.2446 - val_loss: -109895920.0000 -
val_accuracy: 0.2547
Epoch 35/50
67/67 [==============================] - 0s 5ms/step - loss: -
123772648.0000 - accuracy: 0.2446 - val_loss: -121410608.0000 -
val_accuracy: 0.2547
Epoch 36/50
67/67 [==============================] - 0s 5ms/step - loss: -
136750656.0000 - accuracy: 0.2446 - val_loss: -133813224.0000 -
val_accuracy: 0.2547
Epoch 37/50
67/67 [==============================] - 0s 5ms/step - loss: -
147801504.0000 - accuracy: 0.2446 - val_loss: -146705872.0000 -
val_accuracy: 0.2547
Epoch 38/50
67/67 [==============================] - 0s 6ms/step - loss: -
163703632.0000 - accuracy: 0.2446 - val_loss: -160650336.0000 -
val_accuracy: 0.2547
Epoch 39/50
67/67 [==============================] - 0s 5ms/step - loss: -
177709920.0000 - accuracy: 0.2446 - val_loss: -175319456.0000 -
```

```
val_accuracy: 0.2547
Epoch 40/50
67/67 [==============================] - 0s 3ms/step - loss: -
196417344.0000 - accuracy: 0.2446 - val_loss: -191164640.0000 -
val_accuracy: 0.2547
Epoch 41/50
67/67 [==============================] - 0s 4ms/step - loss: -
214042672.0000 - accuracy: 0.2446 - val_loss: -207428096.0000 -
val_accuracy: 0.2547
Epoch 42/50
67/67 [==============================] - 0s 4ms/step - loss: -
229162304.0000 - accuracy: 0.2446 - val_loss: -225313232.0000 -
val_accuracy: 0.2547
Epoch 43/50
67/67 [==============================] - 0s 4ms/step - loss: -
250678064.0000 - accuracy: 0.2446 - val_loss: -243719344.0000 -
val_accuracy: 0.2547
Epoch 44/50
67/67 [==============================] - 0s 4ms/step - loss: -
271242016.0000 - accuracy: 0.2446 - val_loss: -263141600.0000 -
val_accuracy: 0.2547
Epoch 45/50
67/67 [==============================] - 0s 4ms/step - loss: -
288970848.0000 - accuracy: 0.2446 - val_loss: -283601472.0000 -
val_accuracy: 0.2547
Epoch 46/50
67/67 [==============================] - 0s 4ms/step - loss: -
310073824.0000 - accuracy: 0.2446 - val_loss: -304392128.0000 -
val_accuracy: 0.2547
Epoch 47/50
67/67 [==============================] - 0s 4ms/step - loss: -
338458816.0000 - accuracy: 0.2446 - val_loss: -326956736.0000 -
val_accuracy: 0.2547
Epoch 48/50
67/67 [==============================] - 0s 4ms/step - loss: -
362714688.0000 - accuracy: 0.2446 - val_loss: -351120832.0000 -
val_accuracy: 0.2547
Epoch 49/50
67/67 [==============================] - 0s 4ms/step - loss: -
386731616.0000 - accuracy: 0.2446 - val_loss: -375610976.0000 -
val_accuracy: 0.2547
Epoch 50/50
67/67 [==============================] - 0s 4ms/step - loss: -
409900928.0000 - accuracy: 0.2446 - val_loss: -401152800.0000 -
val_accuracy: 0.2547
Error train DNN:  75.54041296243668 %
Error test DNN:  74.53007400035858 %
```

```
#Importance des variables pour le réseau de neurone (plus grand =>
plus important)
```

```
from sklearn.inspection import permutation_importance
r = permutation_importance(clf, X_test, y_test,
n_repeats=30,random_state=0)
print(r.importances_mean)
```

```
[ 4.13220551e-02  2.88220551e-03  3.19548872e-02  2.80701754e-02
  1.21867168e-02  1.00877193e-01  8.51817043e-02  1.59774436e-02
  2.04573935e-02  3.82205514e-03  7.83208020e-04  5.41979950e-03
  1.12781955e-03  9.49248120e-03  1.30012531e-02  8.14536341e-04
  1.47243108e-03  1.44110276e-03 -1.00250627e-03 -3.13283208e-05
 -9.39849624e-05 -3.44611529e-04  0.00000000e+00  1.37844612e-03
  1.66040100e-03 -6.26566416e-05  3.75939850e-04  1.44110276e-03
  6.26566416e-04 -5.95238095e-04  9.08521303e-04  3.13283208e-05
  2.60025063e-03  9.39849624e-04  1.87969925e-04 -6.26566416e-05
  2.50626566e-03  1.66040100e-03 -2.22044605e-17  3.19548872e-03
  6.26566416e-05  3.07017544e-03  1.00250627e-03  1.56641604e-04
  1.00250627e-03 -4.07268170e-04 -9.39849624e-04  1.25313283e-04
  8.14536341e-04 -1.00250627e-03 -6.26566416e-05  1.44110276e-03
  2.19298246e-04  6.89223058e-04  1.87969925e-04  8.14536341e-04
  1.09649123e-03  6.26566416e-05 -2.22044605e-17 -3.13283208e-05
  1.15914787e-03 -2.59052039e-17  0.00000000e+00 -7.51879699e-04
  1.66040100e-03  1.22180451e-03  6.26566416e-04  2.22431078e-03
  1.15914787e-03  1.19047619e-03 -6.26566416e-04  9.39849624e-05
 -3.13283208e-05 -7.83208020e-04  3.13283208e-03  2.19298246e-04
 -2.96059473e-17  7.20551378e-04  2.50626566e-03 -4.07268170e-04
  6.26566416e-05  3.13283208e-05  1.53508772e-03 -9.39849624e-05
 -3.13283208e-05 -2.81954887e-04 -5.63909774e-04  8.45864662e-04
  3.44611529e-04  1.53508772e-03  1.87969925e-04  1.15914787e-03
 -2.19298246e-04 -5.32581454e-04 -6.89223058e-04]
```

- Variables les plus importantes: Tirs cadrés de l'équipe à domicile, Tirs cadrés de l'équipe à l'extérieur, cote de victoire de l'équipe à domicile

Predict the best model

On a decidé de prédire avec le modèle MLP et la fonction d'activation logistic et le solver adam étant donné qu'il s'agit du modèle avec le taux d'erreur le plus faible.

```
finalset = season_21_22_modif.drop(['FTR_code'], axis=1)
finalset = finalset.drop(['pred'], axis=1)
pred_season_21_22 = clf.predict(finalset)
pred_season_21_22 = pd.DataFrame(pred_season_21_22)
pred_season_21_22.columns = ['pred']
pred_season_21_22.head()
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:432:
UserWarning: X has feature names, but MLPClassifier was fitted without
feature names
  warnings.warn(
```

```
   pred
0     2
1     2
2     0
3     2
4     2
```

```
label_make = LabelEncoder()
season_21_22_eval = all_seasons_frames[len(all_seasons_frames)-1]
season_21_22_eval["FTR_code"] =
label_make.fit_transform(season_21_22_eval["FTR"])
season_21_22_eval['Pred'] = pred_season_21_22['pred'].astype(int)
season_21_22_eval.head(15)
```

```
          HomeTeam        AwayTeam  B365H  B365D  B365A  HS  AS  HST
AST  HC  \
0        Brentford         Arsenal   4.00   3.40   1.95   8  22    3
4   2
1       Man United           Leeds   1.53   4.50   5.75  16  10    8
3   5
2          Burnley        Brighton   3.10   3.10   2.45  14  14    3
8   7
3          Chelsea  Crystal Palace   1.25   5.75  13.00  13   4    6
1   5
4          Everton     Southampton   1.90   3.50   4.00  14   6    6
3   6
5        Leicester          Wolves   1.66   3.80   5.25   9  17    5
3   5
6          Watford     Aston Villa   3.10   3.20   2.37  13  11    7
2   2
7          Norwich       Liverpool   9.00   5.75   1.30  14  19    3
8   3
8        Newcastle        West Ham   3.20   3.50   2.20  17   8    3
9   7
9        Tottenham        Man City   5.50   4.20   1.60  13  18    3
4   3
10       Liverpool         Burnley   1.18   7.50  13.00  27   9    9
3   8
11     Aston Villa       Newcastle   1.80   3.75   4.33  10   9    2
1   3
12  Crystal Palace       Brentford   2.55   3.20   2.87   7  14    2
3   3
13           Leeds         Everton   2.37   3.40   3.00  17  17    4
8   8
14        Man City         Norwich   1.08  11.00  26.00  16   1    4
0   6

    AC  HF  AF  HY  AY  HR  AR FTR  FTR_code  Pred
0    5  12   8   0   0   0   0   H         2     2
1    4  11   9   1   2   0   0   H         2     2
```

```
2     6  10   7   2   1   0   0   A         0     0
3     2  15  11   0   0   0   0   H         2     2
4     8  13  15   2   0   0   0   H         2     2
5     4   6  10   1   2   0   0   H         2     2
6     4  18  13   3   1   0   0   H         2     2
7    11   4  14   1   1   0   0   A         0     0
8     6   4   3   1   0   0   0   A         0     0
9    11  11   8   2   1   0   0   H         2     2
10    4   6  12   0   0   0   0   H         2     2
11    4   8  18   3   4   0   0   H         2     2
12    5  12   9   3   1   0   0   D         1     0
13    5   6  13   2   4   0   0   D         1     0
14    1  13   7   1   0   0   0   H         2     2
```

On va calculer l'erreur de généralisation de notre modèle pour les données de la saison 2021_2022

```
error_nn_season_21_22 =  (sum(season_21_22_eval.FTR_code!
=season_21_22_eval.Pred)/len(season_21_22_eval))*100

print("Erreur saison 2019-2020: ", error_nn_season_21_22, "%")
```

```
Erreur saison 2019-2020:  35.26315789473684 %
```

```
data_HomeWin=pd.DataFrame(season_21_22_eval.loc[season_21_22_eval.Pred
==
2,'HomeTeam'].value_counts()).reindex(season_21_22_eval.HomeTeam.uniqu
e(), fill_value=0)
data_HomeWin.columns = ['HomeWin']

data_AwayWin=pd.DataFrame(season_21_22_eval.loc[season_21_22_eval.Pred
==
0,'AwayTeam'].value_counts()).reindex(season_21_22_eval.AwayTeam.uniqu
e(), fill_value=0)
data_AwayWin.columns = ['AwayWin']

data_HomeDraw=pd.DataFrame(season_21_22_eval.loc[season_21_22_eval.Pre
d ==
1,'HomeTeam'].value_counts()).reindex(season_21_22_eval.HomeTeam.uniqu
e(), fill_value=0)
data_HomeDraw.columns = ['HomeDraw']

data_AwayDraw=pd.DataFrame(season_21_22_eval.loc[season_21_22_eval.Pre
d ==
1,'AwayTeam'].value_counts()).reindex(season_21_22_eval.AwayTeam.uniqu
e(), fill_value=0)
data_AwayDraw.columns = ['AwayDraw']

data_HomeLose=pd.DataFrame(season_21_22_eval.loc[season_21_22_eval.Pre
d ==
```

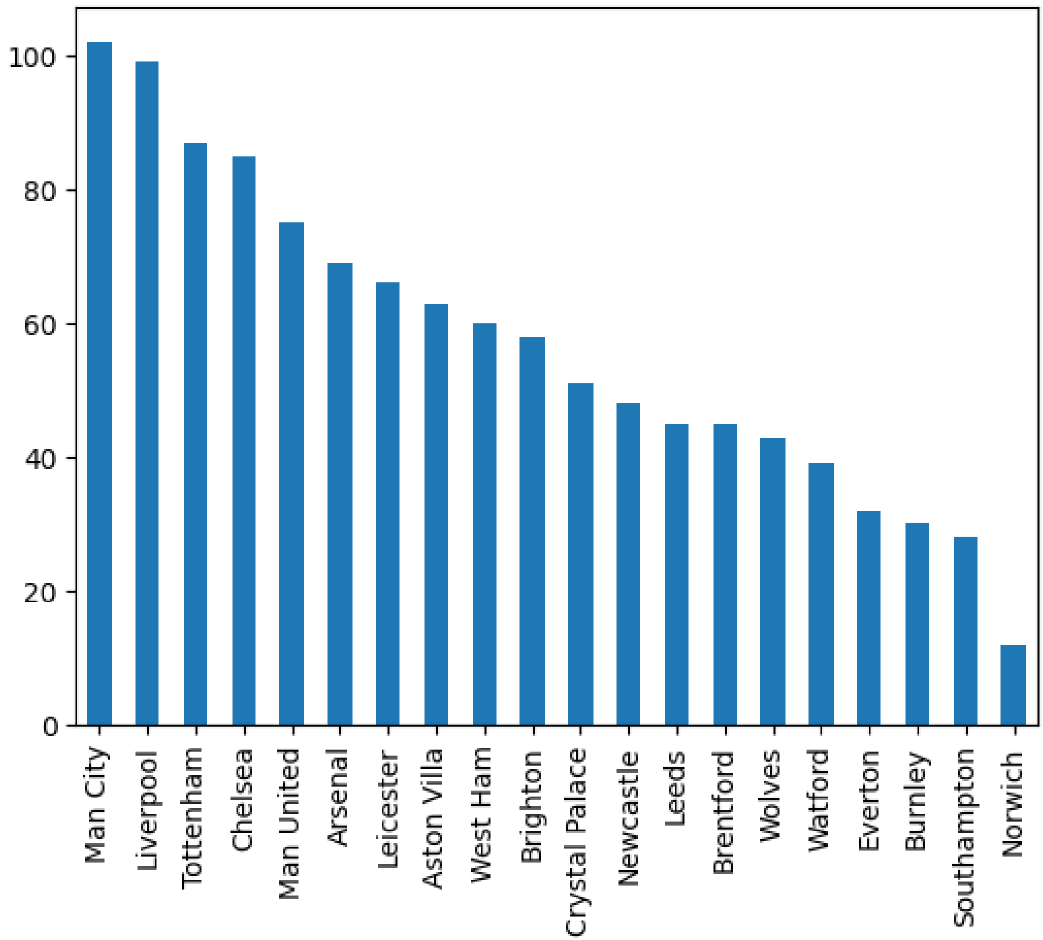
```
0,'HomeTeam'].value_counts()).reindex(season_21_22_eval.HomeTeam.uniqu
e(), fill_value=0)
data_HomeLose.columns = ['HomeLose']

data_AwayLose=pd.DataFrame(season_21_22_eval.loc[season_21_22_eval.Pre
d ==
2,'AwayTeam'].value_counts()).reindex(season_21_22_eval.AwayTeam.uniqu
e(), fill_value=0)
data_AwayLose.columns = ['AwayLose']


output = pd.concat([data_HomeWin, data_AwayWin, data_HomeDraw,
data_AwayDraw, data_HomeLose, data_AwayLose], axis=1, join='inner')
output
```

```
                HomeWin  AwayWin  HomeDraw  AwayDraw  HomeLose
AwayLose
Brentford             9        6         0         0        10
13
Man United           15       10         0         0         4
9
Burnley               8        2         0         0        11
17
Chelsea              16       12         0         1         3
6
Everton               6        4         1         1        12
14
Leicester            13        9         0         0         6
10
Watford               7        6         0         0        12
13
Norwich               3        1         0         0        16
18
Newcastle            10        6         0         0         9
13
Tottenham            16       13         0         0         3
6
Liverpool            18       15         0         0         1
4
Aston Villa          11       10         0         0         8
9
Crystal Palace        8        9         0         0        11
10
Leeds                 5       10         0         0        14
9
Man City             18       16         0         0         1
3
Brighton              8       11         1         0        10
8
Southampton           5        4         0         1        14
```

```
14
Wolves                8        6         1         0         10
13
Arsenal              14        9         0         0          5
10
West Ham             11        9         0         0          8
10
```

```
output['Points'] = output['HomeWin']*3 + output['AwayWin']*3 +
output['HomeDraw']*1 + output['AwayDraw']*1

output['Points'].sort_values(ascending=False).plot(kind='bar')
```

```
<Axes: >
```




```
output.sort_values(by=['Points'], ascending=False, inplace=True)
output
```

```
                HomeWin  AwayWin  HomeDraw  AwayDraw  HomeLose
AwayLose  \
Man City             18       16         0         0         1
3
```

Liverpool 4	18	15	0	0	1
Tottenham 6	16	13	0	0	3
Chelsea 6	16	12	0	1	3
Man United 9	15	10	0	0	4
Arsenal 10	14	9	0	0	5
Leicester 10	13	9	0	0	6
Aston Villa 9	11	10	0	0	8
West Ham 10	11	9	0	0	8
Brighton 8	8	11	1	0	10
Crystal Palace 10	8	9	0	0	11
Newcastle 13	10	6	0	0	9
Leeds 9	5	10	0	0	14
Brentford 13	9	6	0	0	10
Wolves 13	8	6	1	0	10
Watford 13	7	6	0	0	12
Everton 14	6	4	1	1	12
Burnley 17	8	2	0	0	11
Southampton 14	5	4	0	1	14
Norwich 18	3	1	0	0	16

	Points
Man City	102
Liverpool	99
Tottenham	87
Chelsea	85
Man United	75
Arsenal	69
Leicester	66
Aston Villa	63
West Ham	60
Brighton	58

Crystal Palace	51
Newcastle	48
Leeds	45
Brentford	45
Wolves	43
Watford	39
Everton	32
Burnley	30
Southampton	28
Norwich	12