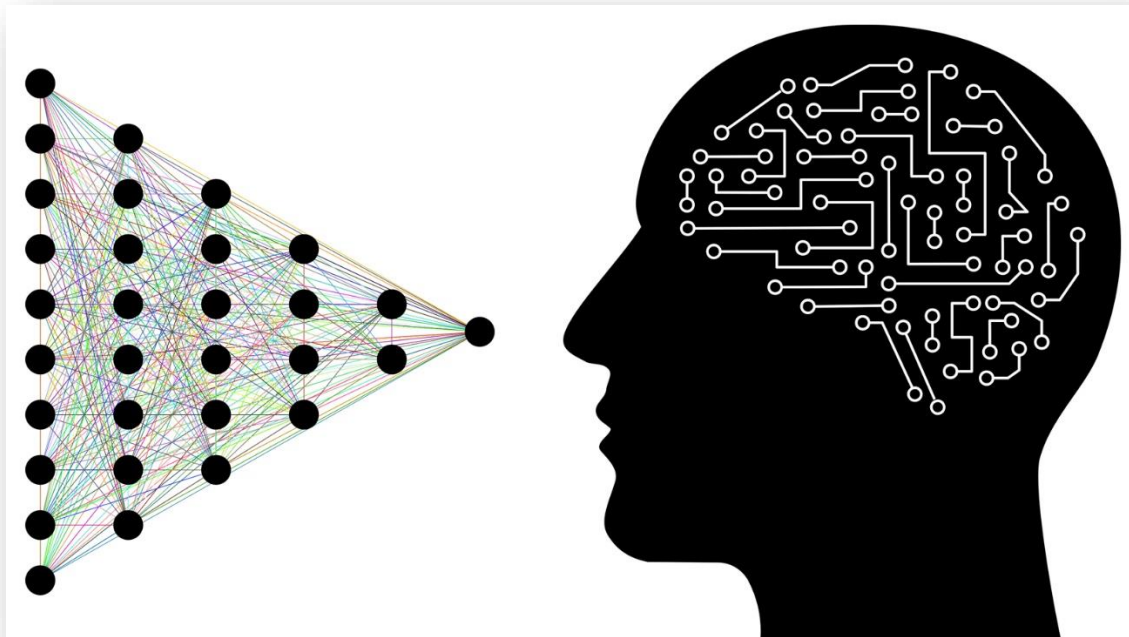

CI - Logiciels et systèmes intelligents (LSI)

Département Informatique

Devoir 3 : Réseaux de neurones



Réalisé par :

Basma Akarmass

Achraf El Krissi

Nafia Akdi

Encadré par : MR. Mohammed Ait

Remerciement

Au terme de ce travail, nous tenons à exprimer notre très vive reconnaissance à notre cher professeur et encadrant M. M'hamed AIT KBIR pour son suivi et pour son énorme soutien qu'il n'a cessé de nous prodiguer au long de cette semestre.

L'Objectif :

Développer deux applications pour implémenter les solutions apportées par les modèles de réseaux de neurones artificiels aux deux problèmes en bas.

1~ Utilisation des réseaux RBF (Radial basis function) pour la prédiction de la consommation de carburant en miles par gallon, les données de la base des exemples d'apprentissage sont disponibles sur ce lien :

<https://archive.ics.uci.edu/ml/datasets/auto+mpg>

2~ Utilisation des réseaux multi-couches pour la prédiction du coup à jouer dans le jeu de Ti-Tac-Toe.

Expliquer la démarche adoptée pour former la base des exemples d'apprentissage. Développer l'application qui permet de jouer une partie de Tic-Tac-Toe Homme-Machine, la décision prise par la machine est le résultat prédit par le PMC.

Table des matières

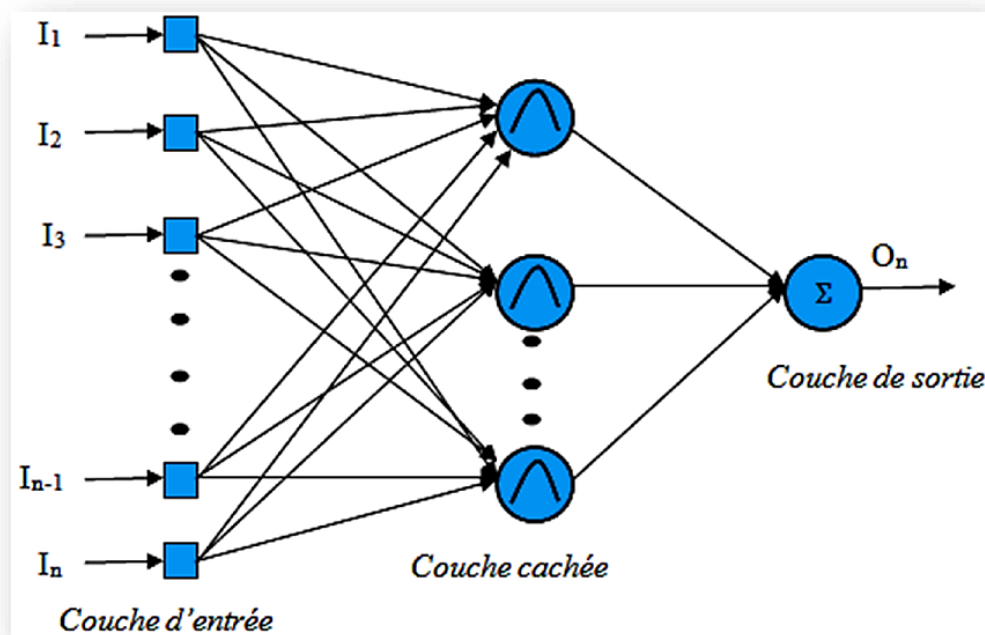
1) Exercice 1.....	6
1.1 Enoncé :	6
1.2 Réseau RBF :	7
1.3 La démarche à suivre :	8
1.4 Importer le fichier « auto-mpg.csv » dans Jupyter :	9
1.5 Visualiser les données :	9
1.6 Convertir les données en un format numérique :	11
1.7 Compter le nombre de valeurs non nulles (NaN).....	12
1.8 Supprimer les lignes contenant des valeurs manquantes (NaN)	14
1.9 Calculer des statistiques de base sur les colonnes numériques	16
1.10 Suppression de la colonne « car name » :	18
1.11 Calcule les corrélations entre toutes les colonnes du DataFrame	19
1.12 Créer un Histogramme d'une colonne	21
1.13 Graphe de boîte (boxplot)	23
1.14 Normaliser les données	25
1.15 Sélectionner les données d'entrées (Features).....	27
1.16 Sélectionner la variable cible (Target)	28
1.17 Séparez les données en jeux d'entraînement et de test :	29
1.18 Le jeu d'entraînement pour entraîner un modèle RBF :	30
1.19 Effectuer La prédiction sur les données de Test :	31
1.20 Comparer visuellement les valeurs réelles avec les valeurs de la prédiction :	32
1.21 Tracer la consommation de carburant prévue par rapport à la consommation de carburant réelle :	33
1.22 Afficher la précision du modèle sur l'ensemble de données d'entraînement.....	34
1.23 Le score de précision du modèle sur les données de test.	35
1.24 1.10 Le score de précision R^2 pour les données de Test :	36
1.4 Exercice 2 :	37
2.1 Enoncé :	37
2.2 La démarche à suivre :	38
2.3 Former la base des exemples d'apprentissage :	39
2.4 Entraîner le modèle PMC.....	43
2.4.1 Importer le fichier « data.csv » dans Jupyter	43

2.4.2	Visualiser les données.....	43
2.4.3	Séparer les données en un ensemble d'entraînement et un ensemble de test	45
2.4.4	Créer le modèle PMC	46
2.4.5	Entraîner le modèle PMC	47
2.4.6	Effectuer La prédiction sur les données de Test :.....	48
2.4.7	Comparer visuellement les valeurs réelles avec les valeurs de la prédiction	49
2.4.8	La précision :.....	50
2.5	L'application :	51
2.5.1	La bibliothèque utilisée :	51
2.5.2	L'interface du jeu :	52
2.5.3	Home Vs Machine :	53

1) Exercice 1

1.1 Enoncé :

Utilisation des réseaux RBF (Radial basis function) pour la prédiction de la consommation de carburant en miles par gallon :



- ❖ Les données de la base des exemples d'apprentissage sont disponibles sur le lien : <https://archive.ics.uci.edu/ml/datasets/auto+mpg>

1.2 Réseau RBF :

Les réseaux de fonctions de base radiales (RBF) sont un type de réseau de neurones artificiels qui utilisent des fonctions de base radiales pour modéliser les relations entre les entrées et les sorties.

- Les RBF se basent sur l'idée que les données peuvent être approximées par un certain nombre de fonctions de base, chacune centrée sur un point de données particulier. Ces fonctions de base sont ensuite combinées pour produire une sortie globale.
- Les paramètres de ces fonctions de base sont appris à partir des données d'entraînement en utilisant des algorithmes d'optimisation tels que le gradient à pas variable.
- Les réseaux RBF sont souvent utilisés pour des tâches de classification et de régression, et ils ont montré de bons résultats dans de nombreux domaines tels que la reconnaissance de la parole, l'analyse d'images et la reconnaissance d'écriture manuscrite.

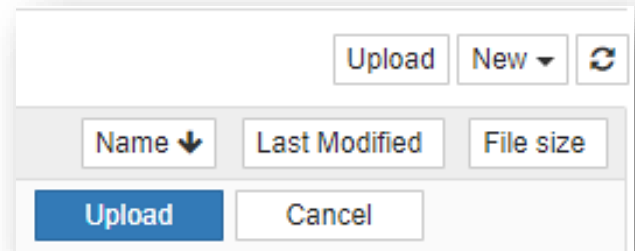
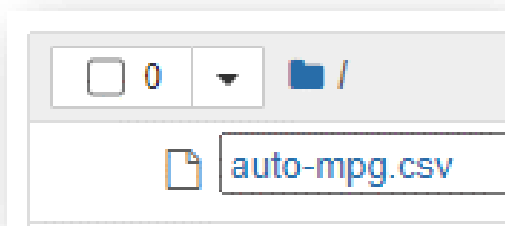
1.3 La démarche à suivre :

Pour développer une application utilisant les réseaux RBF pour prédire la consommation de carburant en miles par gallon :

- Téléchargement de données de la base d'exemples d'apprentissage à partir de l'adresse :
<https://archive.ics.uci.edu/ml/datasets/auto+mpg>
- Charger les données et stocker dans une variable.
- Utilisez les méthodes de Pandas pour nettoyer et prétraiter les données, telles que la suppression des colonnes inutiles ou la normalisation des données.
- Séparez les données en jeux d'entraînement et de test.
- Utilisez le jeu d'entraînement pour entraîner un modèle RBF.
- Utilisez les méthodes de prédiction sur des données réelles pour évaluer la performance de votre modèle.

1.4 Importer le fichier « auto-mpg.csv » dans Jupyter :

Pour entraîner un modèle, vous devez d'abord importer le fichier CSV dans Jupyter en utilisant la bibliothèque Pandas, qui vous permet de lire les données d'un fichier CSV et de les stocker dans une structure de données appelée DataFrame.



1.5 Visualiser les données :

```
df = pd.read_csv("auto-mpg.csv")
df.head(15)
```

- Cette commande permettra d'importer les données contenues dans le fichier "**auto_mpg.csv**" en utilisant la fonction **read_csv** de Pandas.
- La commande **df.head(15)** permet de visualiser les 15 premières lignes du **DataFrame**, vous permettant de vérifier que les données ont bien été importées et de vérifier leur format. Si vous voulez voir plus de lignes, vous pouvez augmenter le nombre dans la fonction **head()**, pour voir toutes les lignes vous pouvez utiliser **df** sans **head()**.

➤ Les 15 premières lignes du DataFrame :

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140	3449	10.5	70	1	ford torino
5	15.0	8	429.0	198	4341	10.0	70	1	ford galaxie 500
6	14.0	8	454.0	220	4354	9.0	70	1	chevrolet impala
7	14.0	8	440.0	215	4312	8.5	70	1	plymouth fury iii
8	14.0	8	455.0	225	4425	10.0	70	1	pontiac catalina
9	15.0	8	390.0	190	3850	8.5	70	1	amc ambassador dpl
10	15.0	8	383.0	170	3563	10.0	70	1	dodge challenger se
11	14.0	8	340.0	160	3609	8.0	70	1	plymouth 'cuda 340
12	15.0	8	400.0	150	3761	9.5	70	1	chevrolet monte carlo
13	14.0	8	455.0	225	3086	10.0	70	1	buick estate wagon (sw)
14	24.0	4	113.0	95	2372	15.0	70	3	toyota corona mark ii

1.6 Convertir les données en un format numérique :

```
# Rendre la colonne horsepower numérique  
df['horsepower'] = pd.to_numeric(df['horsepower'], errors = 'coerce')
```

Cette commande utilise la bibliothèque Pandas pour convertir la colonne 'horsepower' en un format numérique.

- La fonction "**pd.to_numeric**" prend en paramètre la colonne 'horsepower' et l'option "**errors='coerce'**".
- Cette option permet de convertir les valeurs non valides en NaN (**Not a Number**). Cela signifie que s'il y a des valeurs non numériques dans la colonne 'horsepower' elles seront remplacées par NaN. Cette étape est importante car pour entraîner un modèle de réseau de neurones il est nécessaire que les données soient numériques.

1.7 Compter le nombre de valeurs non nulles (NaN)

```
In [8]: # La colonne HP contient 6 entrées non valides
df.info()
df.count()
```

- La commande "**df.info()**" permet d'obtenir des informations sur le DataFrame
- La commande "**df.count()**" permet de compter le nombre de valeurs non nulles pour chaque colonne dans le DataFrame.

```
Out[8]: mpg          398
cylinders          398
displacement       398
horsepower         392
weight            398
acceleration       398
model year        398
origin            398
car name          398
dtype: int64
```

```
In [15]: # Afficher les 6 lignes à supprimer
df[df['horsepower'].isna()]
```

En utilisant cette commande, vous pouvez afficher les 6 lignes de données qui contiennent des valeurs manquantes pour la colonne "horsepower"

- La commande `df[df['horsepower'].isna()]` utilise la méthode `isna()` pour sélectionner toutes les lignes où la colonne "horsepower" contient des valeurs manquantes (NaN).

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
32	25.0	4	98.0	NaN	2046	19.0	71	1	ford pinto
126	21.0	6	200.0	NaN	2875	17.0	74	1	ford maverick
330	40.9	4	85.0	NaN	1835	17.3	80	2	renault lecar deluxe
336	23.6	4	140.0	NaN	2905	14.3	80	1	ford mustang cobra
354	34.5	4	100.0	NaN	2320	15.8	81	2	renault 18i
374	23.0	4	151.0	NaN	3035	20.5	82	1	amc concord dl

Cela vous permet de vérifier les données et de décider si vous souhaitez les supprimer ou les remplacer par une valeur valide.

1.8 Supprimer les lignes contenant des valeurs manquantes (NaN)

```
# Supprimer toutes les lignes non valides  
df = df.dropna(axis = 0)  
# Si la valeur d'une cellule est NaN supprimer toute la ligne  
df.info()
```

La commande **df = df.dropna(axis = 0)** utilise la méthode **dropna()** de Pandas pour supprimer toutes les lignes (axis = 0) contenant des valeurs manquantes (NaN) dans le DataFrame df.

- La méthode **dropna()** par défaut supprime les lignes ou les colonnes entières qui contiennent des valeurs manquantes.
- En utilisant **axis = 0**, cela indique à la fonction de **supprimer toutes les lignes entières qui contiennent des valeurs manquantes**.
- La dernière commande **df.info()** donne des informations générales sur les données présentes dans le DataFrame df, comme le nombre de lignes et de colonnes, les types de données de chaque colonne, etc. Cela permet de vérifier que les lignes contenant des valeurs manquantes ont bien été supprimées.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 9 columns):
#   Column          Non-Null Count  Dtype
---  -
0   mpg              392 non-null    float64
1   cylinders         392 non-null    int64
2   displacement      392 non-null    float64
3   horsepower        392 non-null    float64
4   weight            392 non-null    int64
5   acceleration      392 non-null    float64
6   model year       392 non-null    int64
7   origin            392 non-null    int64
8   car name         392 non-null    object
dtypes: float64(4), int64(4), object(1)
memory usage: 30.6+ KB
```

```
# La colonne 'NAME' n'est pas discriminante
# 392/301 ~ 1.30, trop de valeurs uniques, donc peu de valeurs qui se répètent
df['car name'].value_counts()
```

- La colonne "**car name**" n'est pas utile pour la prédiction de consommation de carburant en miles par gallon.
- La commande "**df['car name'].value_counts()**" utilise la fonction "**value_counts()**" de Pandas pour compter le nombre d'occurrences de chaque valeur unique dans la colonne "car name".
- Le résultat est que 392 lignes contiennent des valeurs uniques différentes dans la colonne "car name" sur un total de 392 lignes, ce qui signifie qu'il y a très peu de valeurs qui se répètent, donc cette colonne n'est pas discriminante pour la prédiction de consommation de carburant en miles par gallon.

1.9 Calculer des statistiques de base sur les colonnes numériques

```
#stats = df.describe(percentiles = [0.1, 0.5, 0.8])  
stats = df.describe()  
stats|
```

- La commande **df.describe()** permet de calculer des statistiques de base sur les colonnes numériques du DataFrame df.
- Cela inclut les informations telles que la moyenne, l'écart-type, la valeur minimale, la valeur maximale, et les quartiles. Le résultat de cette commande est un DataFrame qui contient ces informations pour chaque colonne numérique.
- La commande `stats = df.describe()` crée une variable stats qui contient le résultat de cette fonction.

Résultat :

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
count	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000	392.000000
mean	23.445918	5.471939	194.411990	104.469388	2977.584184	15.541327	75.979592	1.576531
std	7.805007	1.705783	104.644004	38.491160	849.402560	2.758864	3.683737	0.805518
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.000000	4.000000	105.000000	75.000000	2225.250000	13.775000	73.000000	1.000000
50%	22.750000	4.000000	151.000000	93.500000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	275.750000	126.000000	3614.750000	17.025000	79.000000	2.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000

Les statistiques calculées incluent :

- La moyenne (mean)
- L'écart-type (std)
- Le minimum (min)
- Le premier quartile (25%)
- Le médian (50%)
- Le troisième quartile (75%)
- Le maximum (max).

Les résultats sont affichés sous forme de tableau, avec les noms de colonnes en haut de chaque colonne.

1.10 Suppression de la colonne « car name » :

```
df = df.drop('car name', axis = 1)  
# Colonne supprimée % des 392 enregistrements.
```

Cette commande utilise la fonction "drop" de Pandas pour supprimer la colonne "car name" du DataFrame "df".

- La colonne est spécifiée en utilisant l'argument "**axis = 1**", ce qui signifie que la colonne est supprimée en fonction de l'axe des colonnes (par opposition à l'axe des lignes).
- Enfin, l'opération est affectée à **la variable df**, c'est à dire que on modifie directement le **dataframe en cours**.

1.11 Calcule les corrélations entre toutes les colonnes du DataFrame .

```
df.corr()
```

La commande **df.corr()** calcule les corrélations entre toutes les colonnes du **DataFrame df**.

- Elle renvoie une matrice de corrélations, où chaque cellule représente la corrélation entre les deux colonnes correspondant aux lignes et aux colonnes de la matrice.
- Les valeurs de corrélation vont **de -1 à 1**, où **-1** indique une **corrélation négative parfaite**, **0** indique l'**absence de corrélation** et **1** indique une **corrélation positive parfaite**.

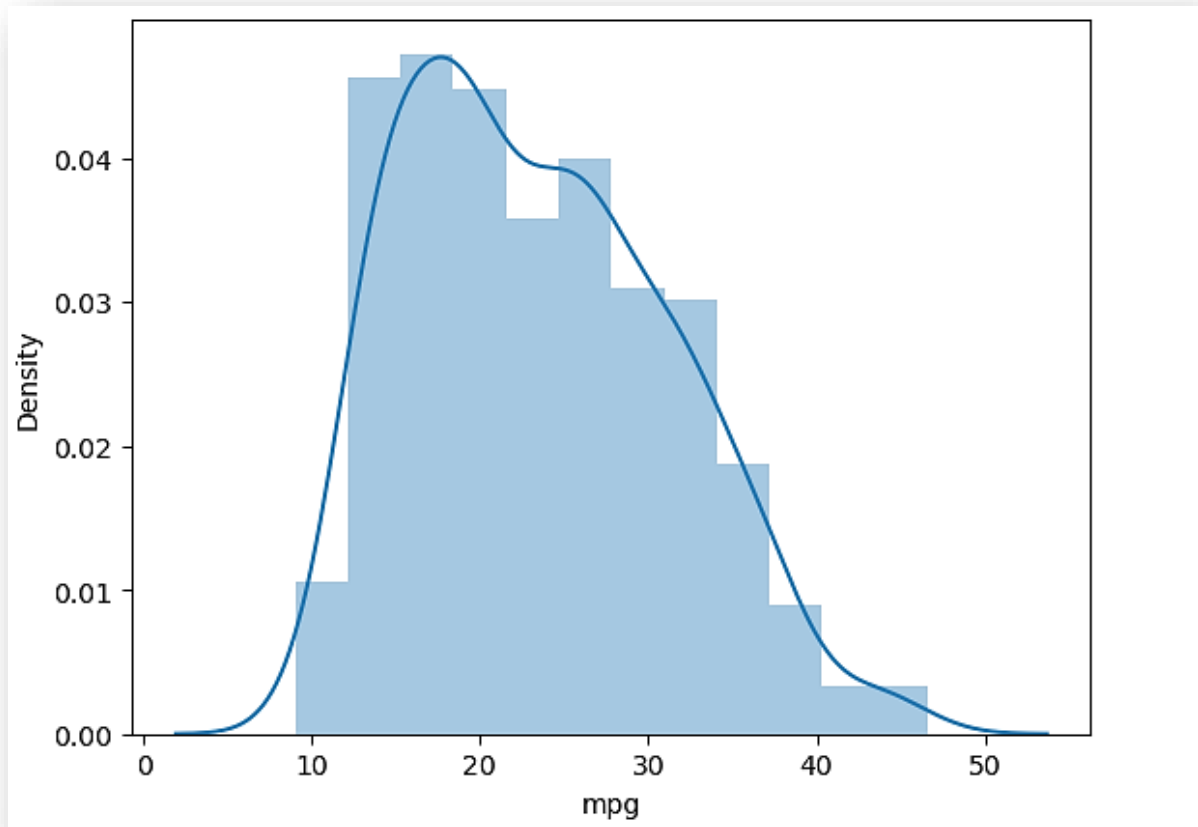
	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
mpg	1.000000	-0.777618	-0.805127	-0.778427	-0.832244	0.423329	0.580541	0.565209
cylinders	-0.777618	1.000000	0.950823	0.842983	0.897527	-0.504683	-0.345647	-0.568932
displacement	-0.805127	0.950823	1.000000	0.897257	0.932994	-0.543800	-0.369855	-0.614535
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361	-0.455171
weight	-0.832244	0.897527	0.932994	0.864538	1.000000	-0.416839	-0.309120	-0.585005
acceleration	0.423329	-0.504683	-0.543800	-0.689196	-0.416839	1.000000	0.290316	0.212746
model year	0.580541	-0.345647	-0.369855	-0.416361	-0.309120	0.290316	1.000000	0.181528
origin	0.565209	-0.568932	-0.614535	-0.455171	-0.585005	0.212746	0.181528	1.000000

- Par exemple, la cellule en haut à gauche indique la corrélation entre la colonne "mpg" et elle-même qui est 1, cela indique que la colonne "mpg" est complètement corrélée à elle-même
- La cellule en haut à droite de la diagonale principale indique la corrélation entre "mpg" et "cylinders" qui est -0.777618, cela indique qu'il existe une forte corrélation négative entre les deux colonnes, c'est-à-dire que lorsque les cylindres augmentent, la consommation de carburant (mpg) diminue.

1.12 Créer un Histogramme d'une colonne

```
# cela implique qu'il y a plus de voitures  
# qui ont un faible mpg que celles qui ont un mpg élevé.  
sns . distplot (df[ 'mpg' ])
```

- La commande "**sns.distplot(df['mpg'])**" utilise la librairie **Seaborn** pour créer un histogramme de la colonne "mpg" dans le DataFrame "df".
- Cela permet de visualiser la distribution des valeurs de consommation **de carburant (mpg) dans les données**.



- L'histogramme montre comment les valeurs sont réparties dans les données, ce qui peut aider à identifier des tendances ou des patterns dans les données.
- **Dans ce cas précis, l'histogramme indique qu'il y a plus de voitures qui ont un faible mpg que celles qui ont un mpg élevé.**

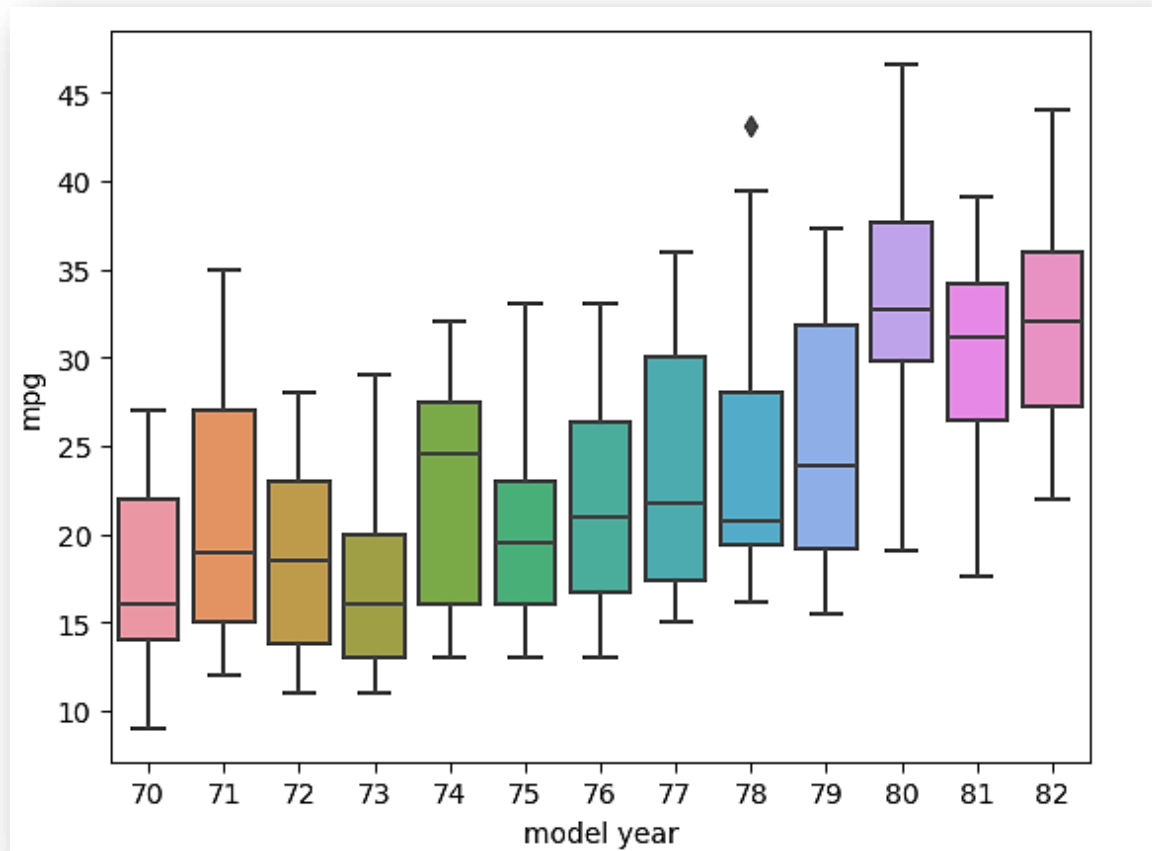
1.13 Graphe de boîte (boxplot)

Un graphique de boîte, également appelé diagramme de boîte à moustache, est un type de graphique utilisé pour afficher les données statistiques d'une série de données numériques.

Il est également connu sous le nom de boîte à moustache, car il ressemble à une boîte avec une moustache à chaque extrémité. Le graphique de boîte montre la répartition des données en affichant la médiane (ligne centrale), la quartile supérieur et inférieur (les boîtes) et les données extrêmes (les moustaches ou les whiskers). Il permet de visualiser rapidement les tendances, la dispersion et les anomalies dans les données.

```
# voyons l'efficacité du mpg au fil des ans  
sns . boxplot (x='model year', y='mpg', data=df)
```

- Cette commande utilise la librairie seaborn pour créer un graphique de boîtes (boxplot) qui montre **la distribution de la colonne "mpg" en fonction de la colonne "model year"**.
- Le graphique de boîtes montre la médiane, les quartiles et les valeurs extrêmes de la colonne "mpg" pour chaque année de modèle. Les points en dehors des boîtes sont les valeurs atypiques.



- Cela permet de visualiser **comment l'efficacité de la consommation de carburant (mpg) a évolué au fil des années** et de voir si certaines années ont une efficacité de consommation de carburant significativement différente des autres.

1.14 Normaliser les données

```
# normalisation (valeur-moyenne)/std  
df_normalized = ((df-df.mean())/df.std())  
df_normalized|
```

Cette commande permet de normaliser les données en utilisant la formule **(valeur-moyenne)/std** pour chaque colonne de la DataFrame.

- Cela signifie que pour chaque valeur dans chaque colonne, la valeur est soustraite de la moyenne de cette colonne et divisée par l'écart type de cette colonne.
- Cela permet de mettre toutes les données sur la même échelle, ce qui est utile pour les comparaisons et les analyses ultérieures.

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
0	-0.697747	1.482053	1.075915	0.663285	0.619748	-1.283618	-1.623241	-0.715726
1	-1.082115	1.482053	1.486832	1.572585	0.842258	-1.464852	-1.623241	-0.715726
2	-0.697747	1.482053	1.181033	1.182885	0.539692	-1.646086	-1.623241	-0.715726
3	-0.953992	1.482053	1.047246	1.182885	0.536160	-1.283618	-1.623241	-0.715726
4	-0.825870	1.482053	1.028134	0.923085	0.554997	-1.827320	-1.623241	-0.715726
...
393	0.455359	-0.862911	-0.519972	-0.479835	-0.220842	0.021267	1.634321	-0.715726
394	2.633448	-0.862911	-0.930889	-1.363154	-0.997859	3.283479	1.634321	0.525711
395	1.095974	-0.862911	-0.567753	-0.531795	-0.803605	-1.428605	1.634321	-0.715726
396	0.583482	-0.862911	-0.711097	-0.661694	-0.415097	1.108671	1.634321	-0.715726
397	0.967851	-0.862911	-0.720653	-0.583754	-0.303253	1.398646	1.634321	-0.715726

392 rows × 8 columns

- La commande normalise les données en calculant (valeur-moyenne)/std pour chaque valeur dans chaque colonne du jeu de données.
- Cela signifie que pour chaque colonne, la moyenne **devient 0** et la **déviatiion standard devient 1**.
- Cela permet de mettre les données sur une échelle commune, ce qui peut être utile pour les comparaisons et les analyses ultérieures.

1.15 Sélectionner les données d'entrées (Features)

```
x= df.drop('mpg', axis = 1)
```

- La commande **x= df.drop('mpg', axis = 1)** permet de sélectionner toutes les colonnes de la dataframe **sauf la colonne "mpg"** et de les stocker dans la variable **"x"**.
- La variable **"x"** sera utilisée comme les entrées (features) pour l'entraînement de notre modèle de machine learning.

1.16 Sélectionner la variable cible (Target)

```
y=df['mpg'].values  
y=(np rint(y)).astype(int)  
.
```

- La **commande** `y=df['mpg'].values` permet de sélectionner la colonne "mpg" de la dataframe et de stocker ses valeurs dans la variable "y".
- La variable "y" sera utilisée comme la variable cible (target) pour l'entraînement de notre modèle de machine learning.
- La **commande** `y=(np rint(y)).astype(int)` permet de arrondir les valeurs de la variable cible "y" à l'entier le plus proche.

1.17 Séparez les données en jeux d'entraînement et de test :

```
|  
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.15,random_state=0)
```

- La commande :
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.15,random_state=0) permet de diviser aléatoirement les données en un jeu de données **d'entraînement (x_train, y_train)** et un jeu de **données de test (x_test, y_test)**.
- La taille du jeu de données de test est **spécifiée à 15% du jeu de données total en utilisant le paramètre test_size=0.15**.
- Le **paramètre random_state=0** garantit que les données sont divisées de manière reproductible.

1.18 Le jeu d'entrainement pour entrainer un modèle RBF :

Utilisation des réseaux RBF pour faire le jeu d'entrainement

```
# Non-Linear regression model using  
#RBF SVM kernel, default value of gamma.  
|  
model = SVC(kernel='rbf',C=10000000)  
model.fit(x_train, y_train)  
  
SVC(C=10000000)
```

- Cette commande utilise l'algorithme SVM (Support Vector Machine) pour entraîner un modèle de régression non linéaire en utilisant le noyau RBF (Radial Basis Function) avec une valeur par défaut de gamma.
- Le modèle est entraîné en utilisant les données d'entraînement pour x (toutes les colonnes sauf "mpg") et y (la colonne "mpg"). Le modèle est ensuite ajusté aux données d'entraînement pour pouvoir faire des prédictions sur de nouvelles données.
- Le paramètre C est utilisé pour contrôler la complexité du modèle et la valeur de gamma est utilisée pour contrôler la largeur du noyau. La valeur C plus élevée signifie une complexité plus faible et une tolérance plus grande aux erreurs dans la classification. Cela signifie que le modèle peut être plus flexible et moins sensible aux données d'entraînement.

Dans cette commande, le paramètre gamma est défini par défaut et C est défini à un très grand nombre (10000000)

1.19 Effectuer La prédiction sur les données de Test :

```
pred=model.predict(x_test)
```

- La commande **pred=model.predict(x_test)** utilise le modèle d'entraînement SVC pour prédire les valeurs de la variable cible (mpg) pour les données de test (x_test).
- La sortie sera un tableau de valeurs prédites pour chaque élément de x_test.

1.20 Comparer visuellement les valeurs réelles avec les valeurs de la prédiction :

```
a=pd.DataFrame({'Real':y_test.reshape(-1),  
                |'Predict':pred.reshape(-1)})  
a.head(25)
```

	Real	Predict
0	28	30
1	22	20
2	12	13
3	38	34
4	34	32
5	19	22
6	38	32
7	30	27

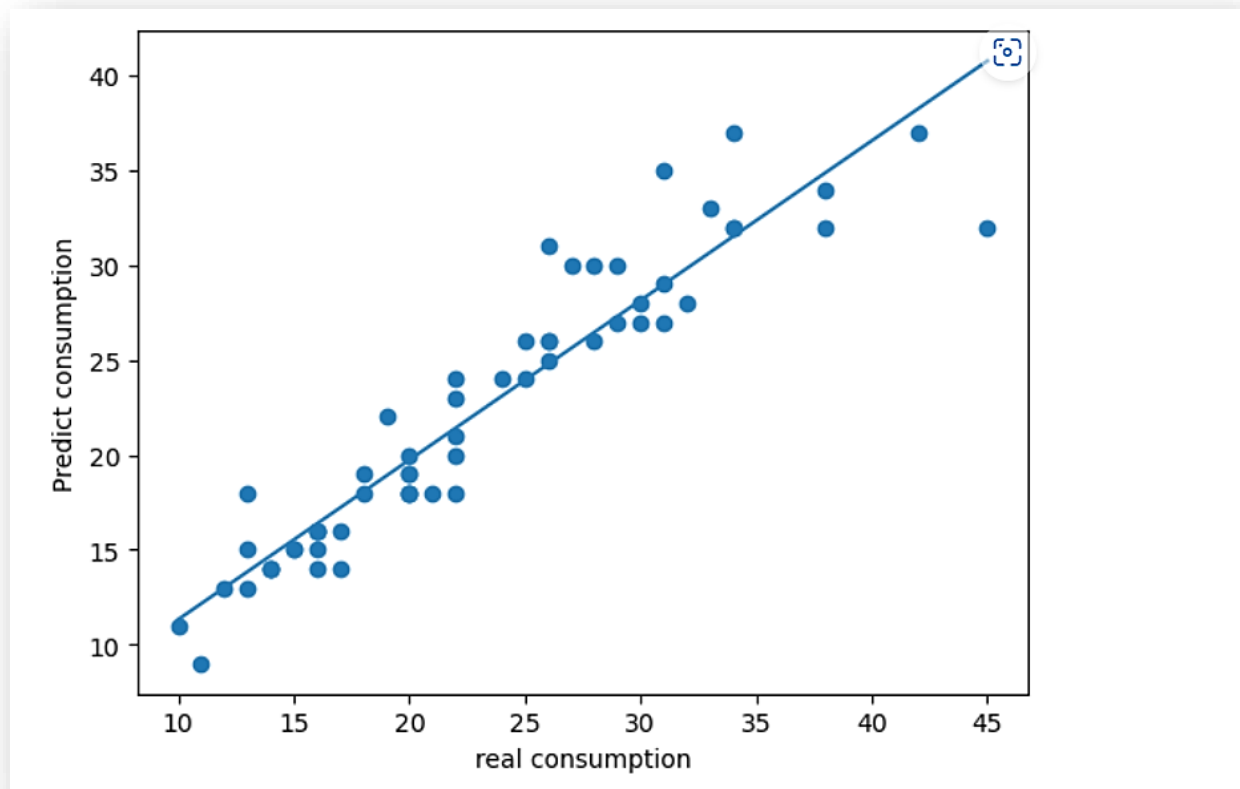
18	26	26
19	28	26
20	13	18
21	16	15
22	20	19
23	25	24
24	42	37

La commande suivante **a=pd.DataFrame({'Real':y_test.reshape(-1),'Predict':pred.reshape(-1)})** crée un dataframe qui contient les valeurs réelles de la variable cible pour les données de test (y_test) et les valeurs prédites par le modèle pour les mêmes données (pred).

La commande **a.head(25)** affiche les 25 premières lignes de ce dataframe, permettant de comparer les valeurs réelles et les valeurs prédites pour les 25 premières observations de données de test.

1.21 Tracer la consommation de carburant prévue par rapport à la consommation de carburant réelle :

```
# Tracer la consommation de carburant prévue par rapport à la consommation de carburant réelle
plt.scatter(y_test,pred)
plt.xlabel("real consumption")
plt.ylabel("Predict consumption")
plt.plot (np.unique(y_test),np.poly1d(np.polyfit(y_test,pred, 1))(np.unique(y_test) ))|
plt. show ( )
```



1.22 Afficher la précision du modèle sur l'ensemble de données d'entraînement

```
: # ensemble de données d'entraînement
print("training dataset % ",model.score(x_train,y_train))

training dataset % 0.7747747747747747
```

- La commande **"print("training dataset %",model.score(x_train,y_train))"** permet d'afficher la précision du modèle sur l'ensemble de données d'entraînement. Cela signifie qu'elle calcule la proportion de prédictions correctes faites par le modèle sur les données d'entraînement par rapport au nombre total de données d'entraînement. Ce résultat est exprimé en pourcentage.
- Dans ce cas, la précision sur l'ensemble de données d'entraînement est de 77,5%, ce qui signifie que le modèle a réussi à prédire correctement 77,5% des valeurs de l'ensemble de données d'entraînement.

1.23 Le score de précision du modèle sur les données de test.

```
print("testing",model.score(x_test,y_test))  
testing 0.2542372881355932
```

- La commande `print("testing",model.score(x_test,y_test))` calcule le score de précision du modèle sur les données de test.
- Le score de précision est un indicateur de la performance du modèle, qui est défini comme le nombre de prédictions correctes divisé par le nombre total de prédictions.
- Le score de précision obtenu est de 0,254, ce qui signifie que le modèle prédit correctement 25,4% des valeurs cibles des données de test.

1.24 1.10 Le score de précision R^2 pour les données de Test :

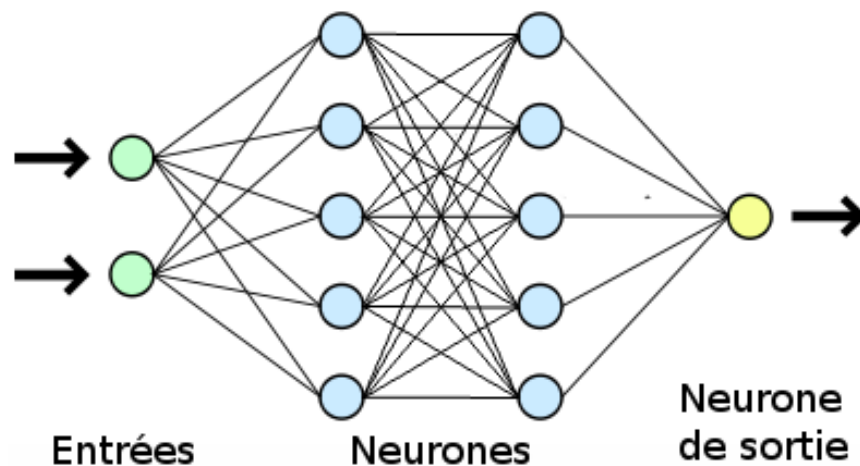
```
: # précision  
print("accuracy",r2_score(y_test,pred))  
  
accuracy 0.8738873575729911
```

- La commande "print("accuracy",r2_score(y_test,pred))" calcule et imprime le score de précision R^2 pour les données de test en comparant les valeurs réelles de "y_test" avec les valeurs prédites "pred".
- Le R^2 est une mesure de la performance de la régression qui varie entre 0 et 1, où 1 signifie une prédiction parfaite. Plus le score est proche de 1, meilleure est la précision du modèle.
- Plus le r^2 score est proche de 1, plus le modèle est précis. Dans ce cas, le modèle a un r^2 score de 0.87, ce qui signifie qu'il prédit correctement 87% de la variance des données cibles.

1.4 Exercice 2 :

2.1 Enoncé :

- ❖ Utilisation des réseaux multi-couches pour la prédiction du coup à jouer dans le jeu de Ti-Tac-Toe.



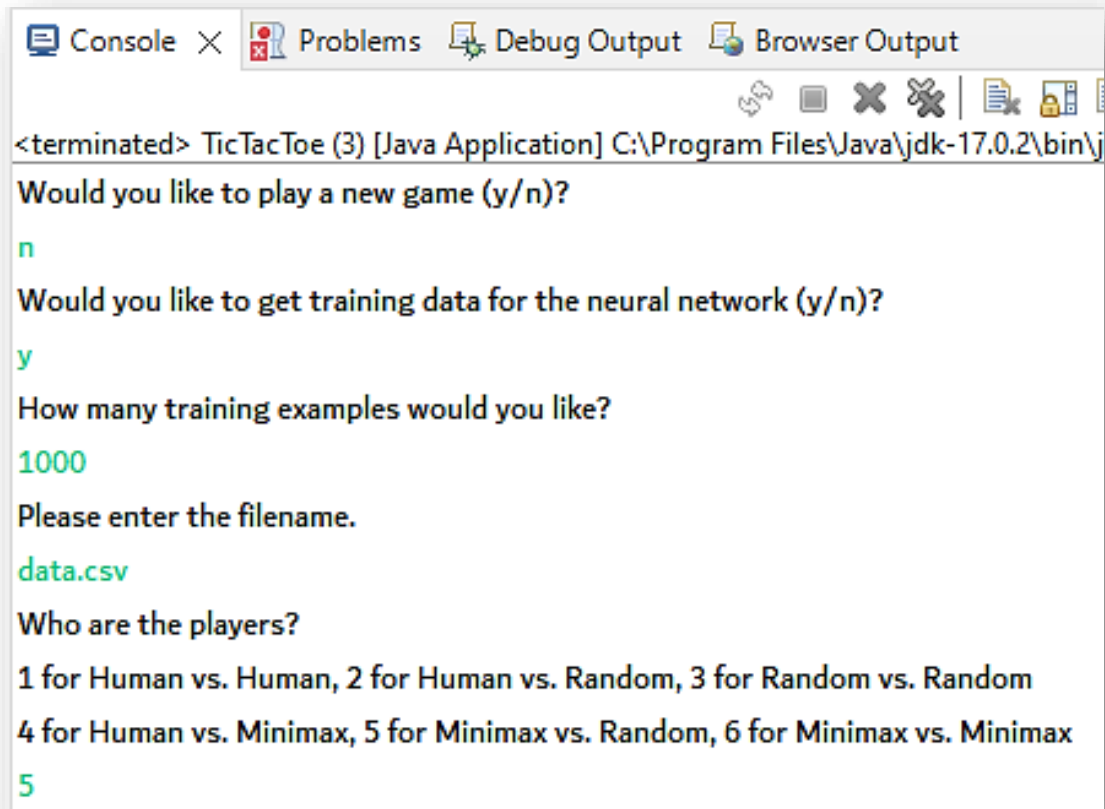
- ❖ Expliquer la démarche adoptée pour former la base des exemples d'apprentissage. Développer l'application qui permet de jouer une partie de Tic-Tac-Toe Homme-Machine, la décision prise par la machine est le résultat prédit par le PMC.

2.2 La démarche à suivre :

- ❖ Former une base de données d'exemples d'apprentissage.
une façon possible de le faire serait de générer des jeux aléatoires de Tic-Tac-Toe, en enregistrant l'état de chaque plateau de jeu et la prochaine action jouée (coup) pour chaque tour de jeu. Ces exemples peuvent ensuite être utilisés pour entraîner un réseau de neurones multi-couches pour prédire le coup à jouer.
- ❖ Créer un réseau de neurone PMC est l'entraîner toute en utilisant la base de données d'exemples
- ❖ Une fois que le modèle est entraîné, il peut être utilisé pour développer une application de Tic-Tac-Toe Homme-Machine. L'application permettra à un utilisateur humain de jouer contre la machine, en utilisant l'interface graphique pour placer les jetons sur le plateau de jeu. À chaque tour, le modèle de réseau de neurones prédira le prochain coup à jouer pour la machine. La décision prise par la machine sera le résultat prédit par le modèle de réseau de neurones.

2.3 Former la base des exemples d'apprentissage :

Une façon possible de le faire serait de générer des jeux aléatoires de Tic-Tac-Toe, en enregistrant l'état de chaque plateau de jeu et la prochaine action jouée (coup) pour chaque tour de jeu.



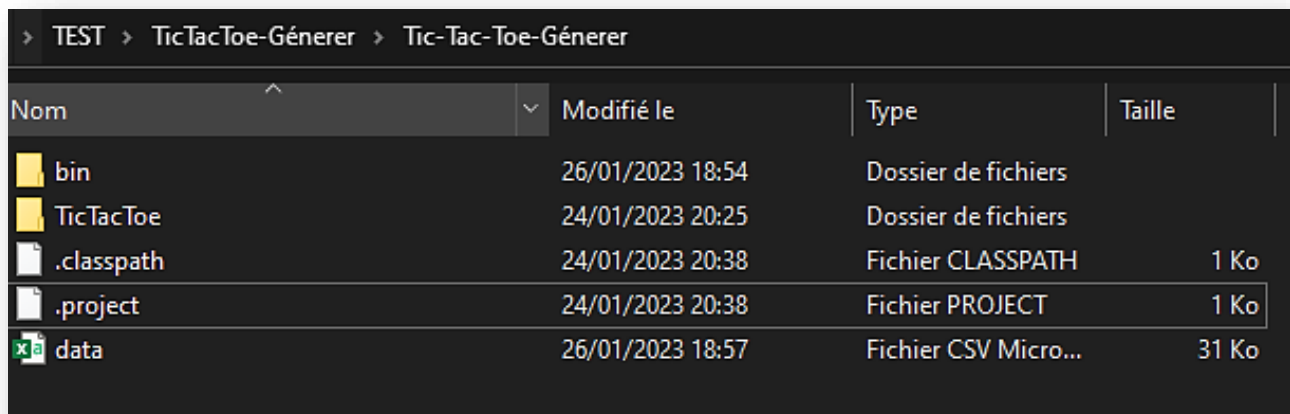
```
<terminated> TicTacToe (3) [Java Application] C:\Program Files\Java\jdk-17.0.2\bin\j
Would you like to play a new game (y/n)?
n
Would you like to get training data for the neural network (y/n)?
y
How many training examples would you like?
1000
Please enter the filename.
data.csv
Who are the players?
1 for Human vs. Human, 2 for Human vs. Random, 3 for Random vs. Random
4 for Human vs. Minimax, 5 for Minimax vs. Random, 6 for Minimax vs. Minimax
5
```

D'abord La console demande si vous voulez jouer à un nouveau jeu (y/n). Si vous répondez "n", elle vous demandera si vous voulez obtenir des données d'entraînement pour un réseau de neurones (y/n). Si vous répondez "y", elle vous demandera combien d'exemples d'entraînement vous voulez, puis le nom du fichier où enregistrer ces données.

Dans ce cas j'ai choisi 1000 exemples et le nom du fichier ou enregistrer ces données s'appelle « data.csv »

Enfin, elle vous demandera qui sont les joueurs :

- 1 : pour humain contre humain
 - 2 : pour humain contre aléatoire
 - 3 : pour aléatoire contre aléatoire
 - 4 : pour humain contre Minimax
 - 5 : pour Minimax contre aléatoire
 - 6 : pour Minimax contre Minimax.
-
- Je recommanderais de choisir Minimax contre Aléatoire comme joueurs pour générer rapidement des exemples d'entraînement.
-
- Il est important de noter que lorsque vous entrez ces données, un fichier sera ajouté à votre répertoire sous le nom de "data.csv".



The screenshot shows a file explorer window with the following structure:

- Path: > TEST > TicTacToe-Générer > Tic-Tac-Toe-Générer
- Columns: Nom, Modifié le, Type, Taille
- Files and folders:
 - bin (Dossier de fichiers, 26/01/2023 18:54)
 - TicTacToe (Dossier de fichiers, 24/01/2023 20:25)
 - .classpath (Fichier CLASSPATH, 24/01/2023 20:38, 1 Ko)
 - .project (Fichier PROJECT, 24/01/2023 20:38, 1 Ko)
 - data (Fichier CSV Micro..., 26/01/2023 18:57, 31 Ko)

Le fichier "data.csv" contient 1000 lignes exactement comme j'ai commandé au début

	A	B
1	0, 0, 0, 0, 0, 0, -1, 0, 0, 5	
2	0, 0, -1, 0, 1, 0, -1, 0, 0, 2	
3	-1, 1, -1, 0, 1, 0, -1, 0, 0, 8	
4	0, 0, 0, 0, -1, 0, 0, 0, 0, 1	
5	1, 0, -1, 0, -1, 0, 0, 0, 0, 7	
6	1, 0, -1, 0, -1, 0, 1, 0, -1, 4	
7	0, 0, 0, 0, 0, 0, 0, 0, 0, 1	
...
988	0, 0, 0, 0, 0, 0, 0, 0, 0, 1	
989	1, 0, 0, 0, 0, 0, 0, -1, 0, 3	
990	1, 0, 1, 0, -1, 0, 0, -1, 0, 2	
991	0, 0, 0, 0, 0, 0, 0, 0, 0, 1	
992	1, -1, 0, 0, 0, 0, 0, 0, 0, 4	
993	1, -1, -1, 1, 0, 0, 0, 0, 0, 7	
994	0, 0, 0, 0, 0, 0, 0, 0, 0, 1	
995	1, 0, 0, 0, 0, 0, -1, 0, 0, 2	
996	1, 1, 0, 0, 0, -1, -1, 0, 0, 3	
997	0, 0, 0, 0, 0, -1, 0, 0, 0, 3	
998	0, -1, 1, 0, 0, -1, 0, 0, 0, 4	
999	0, -1, 1, 1, 0, -1, 0, -1, 0, 5	
1000	-1, -1, 1, 1, 1, -1, 0, -1, 0, 7	

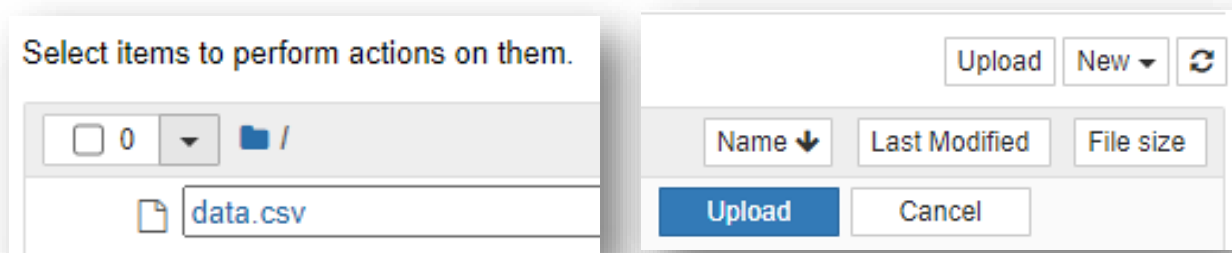
Les données d'entraînement sont formatées de la manière suivante :

- Chaque ligne représente un nouvel exemple d'entraînement avec des données séparées par des virgules.
- Les neuf premiers nombres de chaque ligne représentent la configuration actuelle du plateau ($X(i)$).
- Le dernier nombre représente le coup que le joueur un a sélectionné après avoir analysé la configuration actuelle du plateau ($Y(i)$).

2.4 Entraîner le modèle PMC

2.4.1 Importer le fichier « data.csv » dans Jupyter

Pour entraîner un modèle, vous devez d'abord importer le fichier CSV dans Jupyter en utilisant la bibliothèque Pandas, qui vous permet de lire les données d'un fichier CSV et de les stocker dans une structure de données appelée DataFrame.



2.4.2 Visualiser les données

```
In [2]: #importer les données
df = pd.read_csv("data.csv")
df.head(25)
```

Cette commande permettra d'importer les données contenues dans le fichier **"data.csv"** en utilisant la fonction **read_csv** de **Pandas**. La commande **df.head(25)** permet de visualiser les 25 premières lignes du DataFrame, vous permettant de vérifier que les données ont bien été importées et de vérifier leur format. Si vous voulez voir plus de lignes, vous pouvez augmenter le nombre dans la fonction **head()**, pour voir toutes les lignes vous pouvez utiliser **df** sans **head()**.

Voilà les 25 premières lignes du DataFrame :

out[2]:

	P1	P2	P3	P4	P5	P6	P7	P8	P9	MOVE
0	0	0	0	0	0	0	-1	0	0	5
1	0	0	0	-1	1	0	-1	0	0	1
2	1	-1	0	-1	1	0	-1	0	0	9
3	0	0	0	0	0	0	0	0	0	1
4	1	0	0	0	0	0	0	-1	0	3
5	1	0	1	0	-1	0	0	-1	0	2
6	-1	0	0	0	0	0	0	0	0	5
7	-1	0	0	0	1	0	0	0	-1	2
8	-1	1	0	-1	1	0	0	0	-1	8
9	1	0	-1	0	0	0	0	0	0	4
10	1	0	-1	1	0	0	0	-1	0	7
11	1	0	0	0	0	-1	0	0	0	3
12	1	0	1	0	-1	-1	0	0	0	2
13	1	0	0	-1	1	-1	-1	0	0	9
14	-1	1	0	0	1	0	-1	0	-1	8
15	1	-1	0	0	0	0	0	0	0	4
16	1	-1	0	1	0	0	0	-1	0	7
17	1	-1	0	1	0	0	0	0	-1	7
18	-1	0	0	0	1	-1	0	0	0	2
19	-1	1	0	0	1	-1	0	-1	0	7
20	-1	1	-1	0	1	-1	1	-1	0	9
21	0	0	0	0	-1	0	0	0	0	1
22	1	0	0	0	-1	-1	0	0	0	4
23	1	0	-1	1	-1	-1	0	0	0	7
24	0	0	0	0	0	0	0	-1	0	2

2.4.3 Séparer les données en un ensemble d'entraînement et un ensemble de test

```
# Les exemples:  
x= df.drop('MOVE', axis = 1)  
# Les valeurs désirées  
y=df['MOVE'].values  
# 95% données d'entraînement, 5% données de test  
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.05,random_state=0)|
```

Ces commandes permettent de séparer les données en utilisant la fonction **train_test_split** de la **bibliothèque scikit-learn**.

- ❖ La première **commande x= df.drop('MOVE', axis = 1)** permet de sélectionner toutes les colonnes du DataFrame sauf celle qui a pour nom "MOVE" et de les stocker dans une variable x qui sera utilisée pour les exemples d'entraînement.
- ❖ La deuxième commande **y=df['MOVE'].values** permet de sélectionner la colonne "MOVE" et de stocker les valeurs de cette colonne dans une variable y qui sera utilisée pour les valeurs désirées.
- ❖ La troisième commande **x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=0.05,random_state=0)** permet de diviser les données en utilisant la fonction **train_test_split**. Le paramètre **test_size=0.05** signifie que 5% des données seront utilisées pour les données de test, et 95% pour les données d'entraînement. Le paramètre **random_state=0** permet de fixer un état aléatoire pour s'assurer que les données sont divisées de la même manière chaque fois que vous exécutez le code.

2.4.4 Créer le modèle PMC

```
from sklearn.neural_network import MLPClassifier
# MLP: création + entraînement
mlp = MLPClassifier(hidden_layer_sizes=(30,20), random_state=1, max_iter=2014)
```

ces commandes permettent de créer et d'entraîner un réseau de neurones multi-couches (MLP) en utilisant la bibliothèque scikit-learn.

- ❖ La première commande **from sklearn.neural_network import MLPClassifier** importe la classe **MLPClassifier** de la bibliothèque scikit-learn qui permet de créer et d'entraîner un MLP.
- ❖ La deuxième commande **mlp = MLPClassifier(hidden_layer_sizes=(30,20), random_state=1, max_iter=2014)** crée une instance de la classe **MLPClassifier** avec des paramètres spécifiques.
- ❖ Le paramètre **hidden_layer_sizes=(30,20)** spécifie que le MLP aura 2 couches cachées avec 30 et 20 neurones respectivement.
- ❖ Le paramètre **random_state=1** permet de fixer un état aléatoire pour s'assurer que les poids sont initialisés de la même manière chaque fois que vous exécutez le code. Et Le paramètre **max_iter=2014** permet de définir le nombre maximal d'itérations lors de l'entraînement du MLP. Cela signifie que l'algorithme d'apprentissage s'arrêtera après 2014 itérations

2.4.5 Entraîner le modèle PMC

```
In [5]: mlp.fit(x_train.values, y_train)
```

```
Out[5]: MLPClassifier(hidden_layer_sizes=(30, 20), max_iter=2014, random_state=1)
```

Cette commande utilise la fonction `fit()` pour entraîner le MLP sur les données d'entraînement.

- ❖ La fonction `fit` prend en entrée les données d'entraînement pour les exemples (`x_train.values`) et les valeurs désirées (`y_train`)
- ❖ Cela permet d'optimiser les poids du modèle en utilisant un algorithme d'apprentissage automatique pour minimiser l'erreur entre les prédictions du modèle et les valeurs désirées.
- ❖ Une fois que l'entraînement est terminé, le modèle peut être utilisé pour faire des prédictions sur des données de test en utilisant la fonction `predict()` ou pour évaluer sa performance en utilisant des métriques d'évaluation telles que l'accuracy.

2.4.6 Effectuer La prédiction sur les données de Test :

```
# test1 (ensemble des exemples)  
pred=mlp.predict(x_test)|
```

- ❖ Cette commande utilise la fonction predict() pour effectuer des prédictions sur les données de test.
- ❖ La fonction predict() prend en entrée les données de test (x_test) et retourne les prédictions du modèle (pred) pour ces données.
- ❖ Les prédictions sont des valeurs prédites pour les valeurs désirées (y_test) correspondant aux exemples de test (x_test).

2.4.7 Comparer visuellement les valeurs réelles avec les valeurs de la prédiction

```
a=pd.DataFrame({'Real':y_test.reshape(-1),'Predict':pred.reshape(-1)})  
a.head(100)|
```

Cette commande crée un DataFrame qui contient les valeurs réelles `y_test` et les valeurs prédites `pred`. La fonction `reshape(-1)` est utilisée pour remettre les données sous forme de vecteur 1D.

- ❖ En utilisant la fonction `head(100)`, vous pouvez afficher les premières 100 lignes de ce DataFrame, qui vous permet de comparer visuellement les valeurs réelles avec les prédictions du modèle. Cela peut vous aider à évaluer la performance du modèle et à détecter éventuels problèmes.

Out[9]:

	Real	Predict
0	6	6
1	3	3
2	1	2
3	2	2
4	8	8
5	4	5
6	9	9

2.4.8 La précision :

```
: # la precision
print("accuracy",r2_score(y_test,pred))

accuracy 0.8373870743571925
```

Cette commande utilise la fonction `r2_score()` de scikit-learn pour calculer la précision (ou l'accuracy) du modèle.

- ❖ La fonction `r2_score()` prend en entrée les valeurs réelles (`y_test`) et les valeurs prédites (`pred`) et retourne un score compris entre -1 et 1.
- ❖ Un score proche de 1 indique une performance élevée, c'est-à-dire que les prédictions sont proches des valeurs réelles, tandis qu'un score proche de -1 indique une performance faible, c'est-à-dire que les prédictions sont éloignées des valeurs réelles.

2.5 L'application :

Cette application va utiliser le modèle entraîné PMC Il s'agit d'un script Python qui crée une interface graphique pour un jeu de Tic-Tac-Toe (Morpion).

2.5.1 La bibliothèque utilisée :

La bibliothèque utilisée est « **Tkinter** » pour créer une fenêtre avec un tableau de boutons pour représenter les cases du jeu.

- Il y a également un label pour afficher le tour actuel et un bouton « restart » pour redémarrer le jeu.
- La logique du jeu est implémentée à l'aide de plusieurs fonctions, comme :
 - "next_turn" pour passer au prochain tour
 - "check_win" pour vérifier si quelqu'un a gagné
 - "check_empty_spaces" pour vérifier s'il reste des cases vides
 - "start_new_game" pour redémarrer le jeu.
 - Il y a aussi une partie de l'IA qui est utilisée pour jouer contre l'homme, **cette partie utilise un modèle de réseau de neurones pour prédire le meilleur coup à jouer.**

2.5.2 L'interface du jeu :

Voici l'interface du notre jeu Tic-Tac-Toe :

- Button « restart » pour redémarrer le jeu

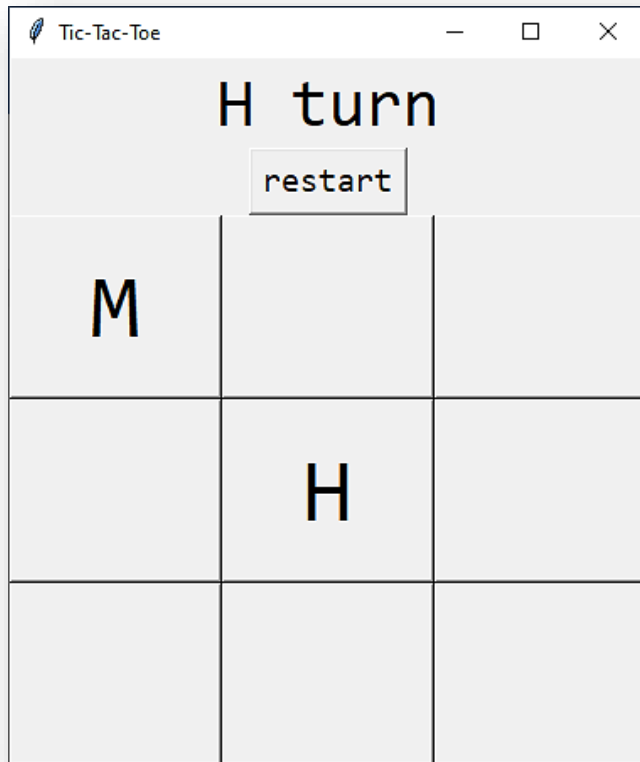


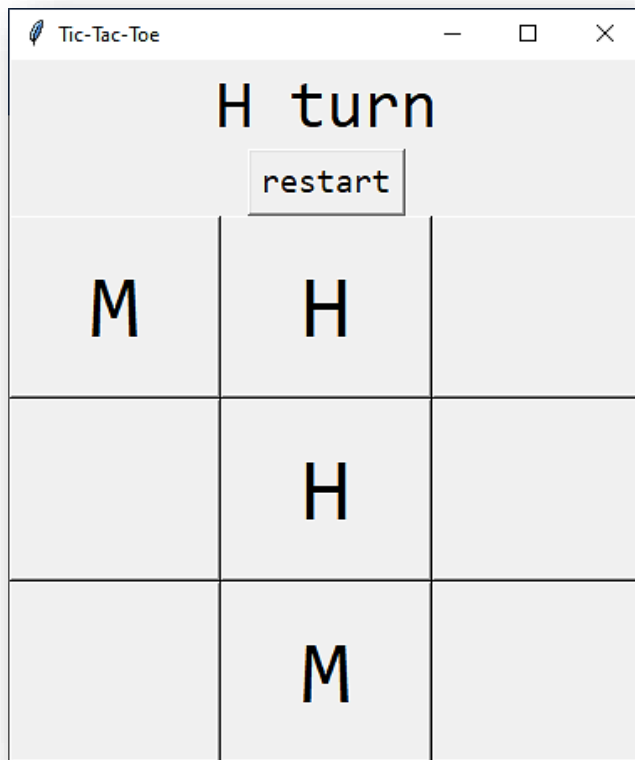
2.5.3 Home Vs Machine :

L'interface indique que c'est le tour de « Homme » pour jouer



Une fois que « Homme » à jouer, la machine joue :

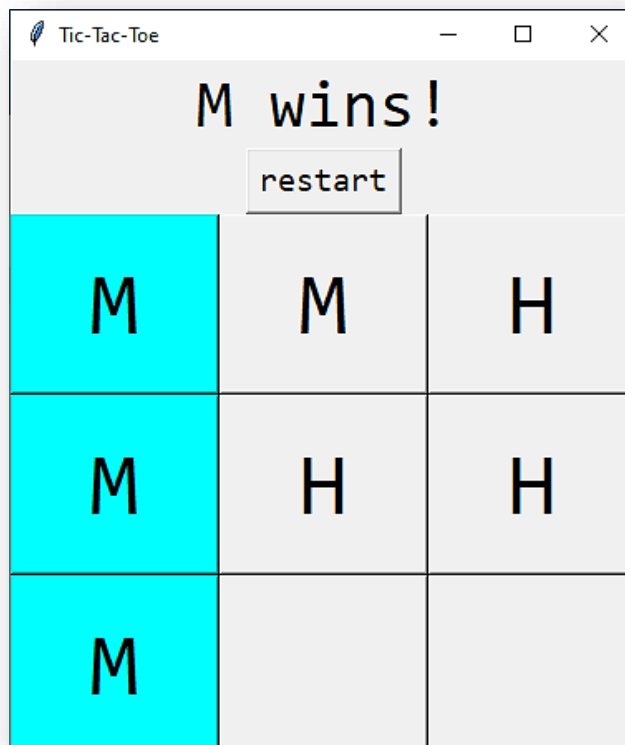




➤ Dans le cas où il n'existe aucune gagnant :



➤ Dans le cas où la machine gagne :



FIN