



ONE HEALTH



Comparaison entre méthodes de
Machine Learning et Deep Learning pour
prédire l'indice de qualité de l'air à
partir de données météo



SOMMAIRE



1. présentation du jeu de données
2. modèle ARIMA
3. modèle Random Forest
4. modèle LSTM
5. comparaison des méthodes



PRÉSENTATION DU JEU DE DONNÉES

- données temporelles (année, mois, jour, heure)
- 12 stations à Pékin
- 6 facteurs météorologiques (température, pluviométrie...)
- 6 facteurs de pollution (concentrations en CO, O3...)


➔ Indice de Qualité de l'Air (6 modalités)

> 420 000 données



DES MÉTHODES DE PRÉVISION ADAPTÉES AUX SÉRIES TEMPORELLES



- ✿ ARIMA: un modèle d'apprentissage statistique pour l'étude des séries temporelles
 - ✿ Les forêts aléatoires: ensemble d'arbres de décision est robuste aux relations non linéaires et peut capturer des interactions complexes dans les séries temporelles avec des variables exogènes
 - ✿ Réseaux neuronaux récurrents: capturent efficacement les motifs locaux dans les séries temporelles et sont bien adaptés à l'extraction d'informations à partir de données séquentielles
- 



TRAITEMENT INITIAL DES DONNÉES



- ✿ Import et réorganisation du jeu de données
- ✿ Imputation des données avec la méthode MissRanger
- ✿ Calcul des indices de qualité de l'air

MODÈLE ARIMA

AutoRegressive Integrated Moving Average

$$X_t = c + \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t$$

$$X_t = \mu + \varepsilon_t + \sum_{i=1}^q \theta_i \varepsilon_{t-i}$$

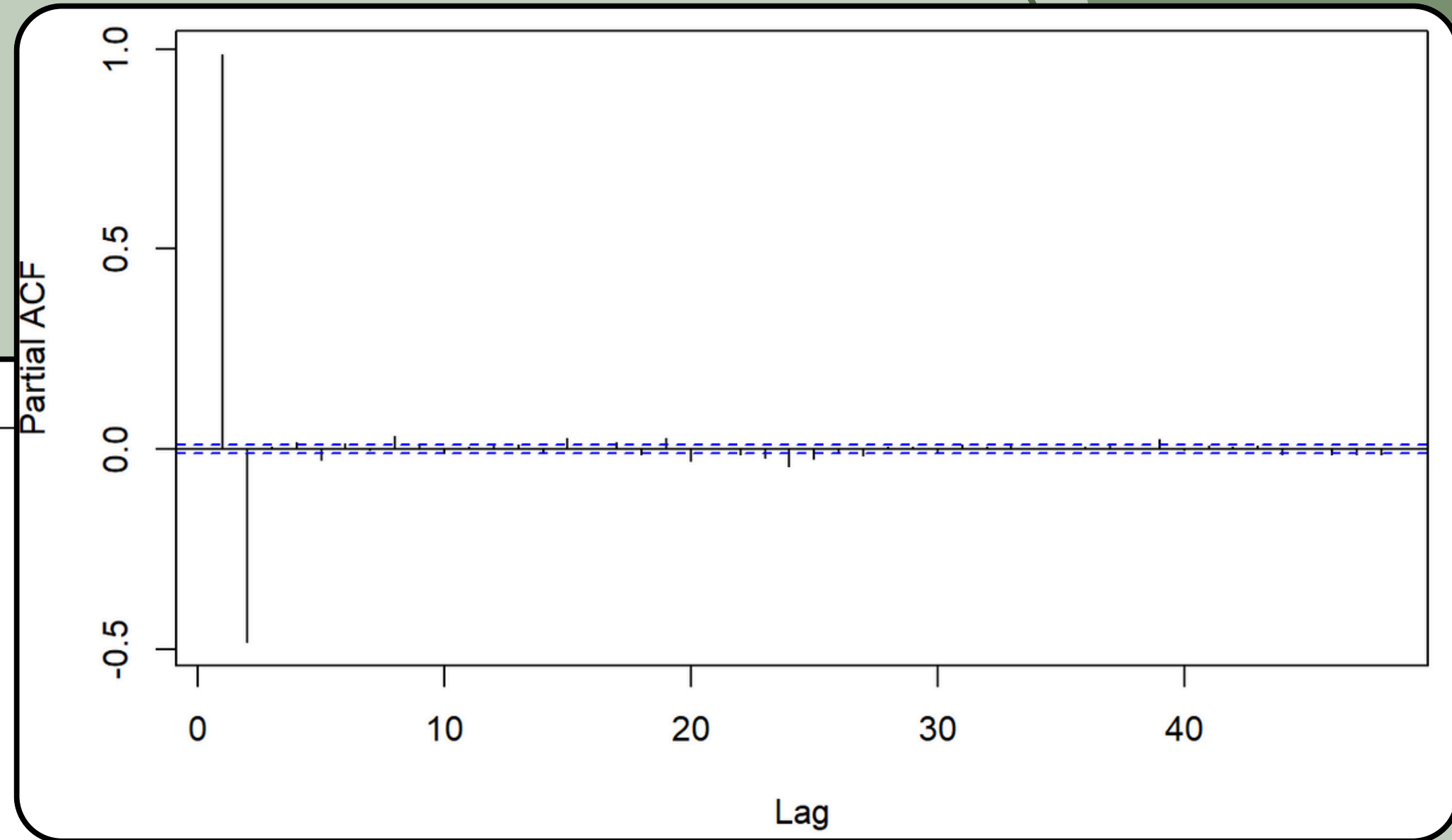
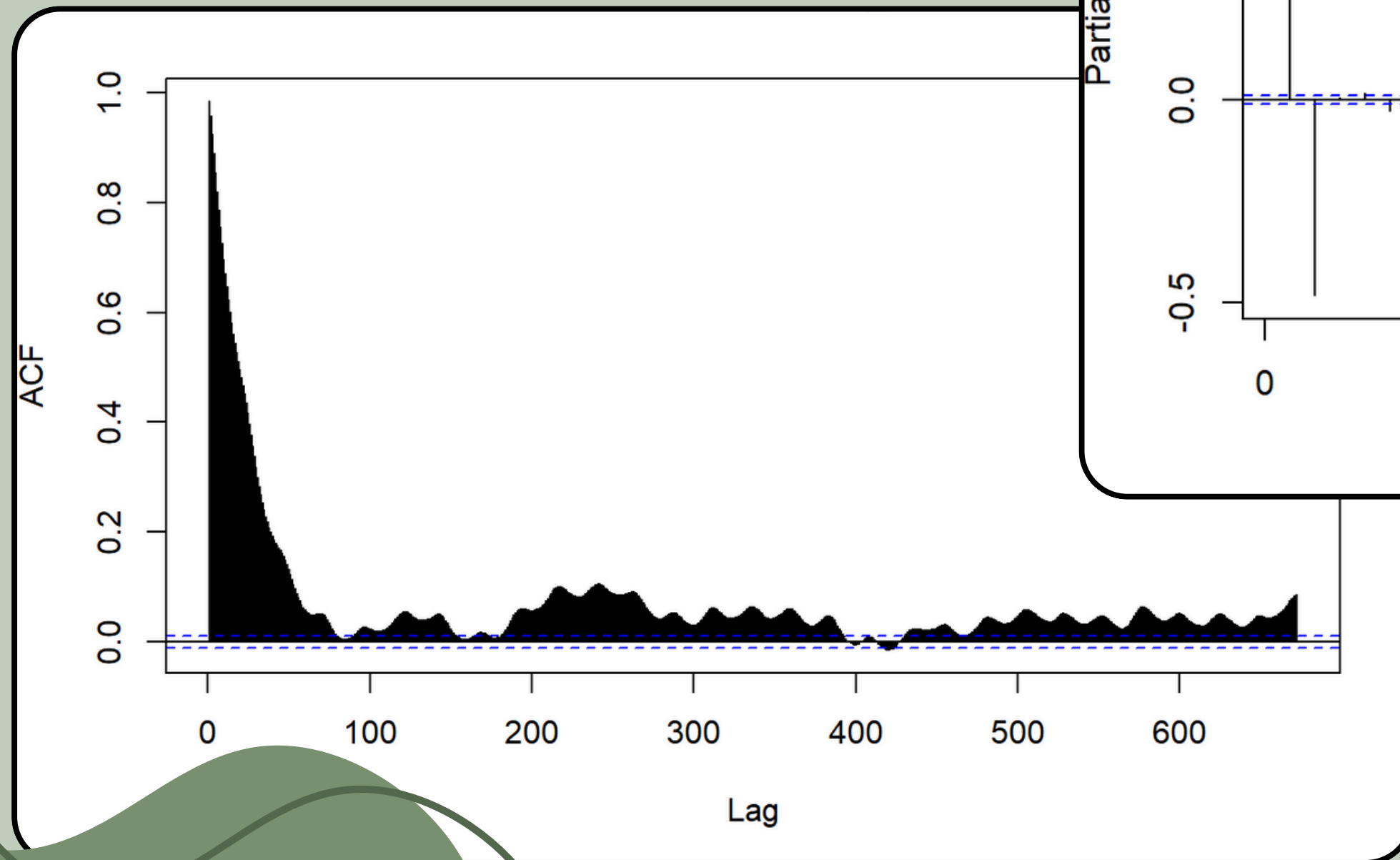
$$X_t - \alpha_1 X_{t-1} - \dots - \alpha_{p'} X_{t-p'} = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q},$$

$$\left(1 - \sum_{i=1}^{p'} \alpha_i L^i\right) X_t = \left(1 + \sum_{i=1}^q \theta_i L^i\right) \varepsilon_t$$

$$\left(1 - \sum_{i=1}^p \varphi_i L^i\right) (1 - L)^d X_t = \delta + \left(1 + \sum_{i=1}^q \theta_i L^i\right) \varepsilon_t$$

MODÈLE ARIMA

paramétrage de ARIMA



q

p

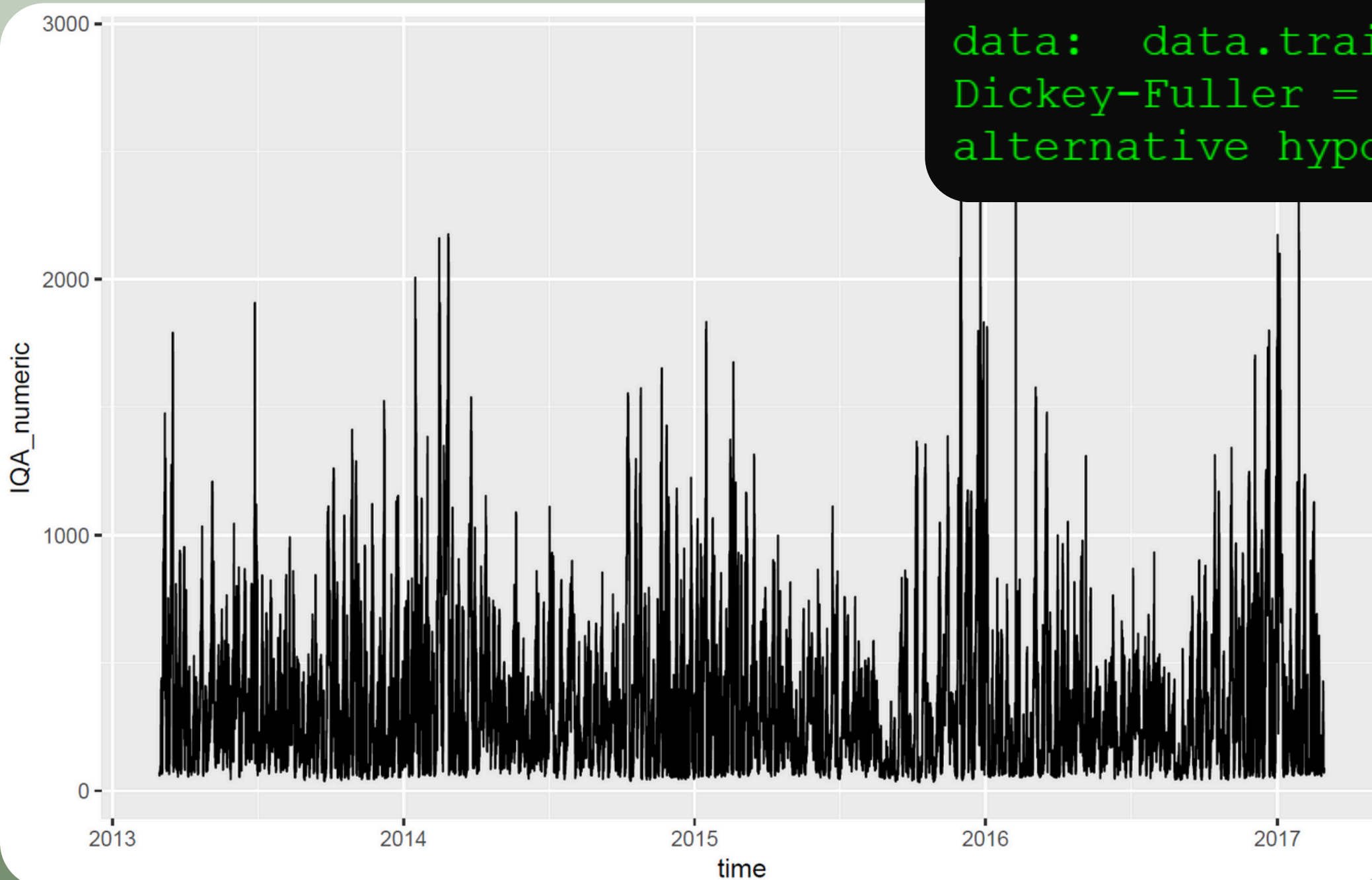
MODÈLE ARIMA

paramétrage de ARIMA

Augmented Dickey-Fuller Test

```
data: data.train$IQA_numeric  
Dickey-Fuller = -20.667, Lag order = 31, p-value = 0.01  
alternative hypothesis: stationary
```

$d = 0$



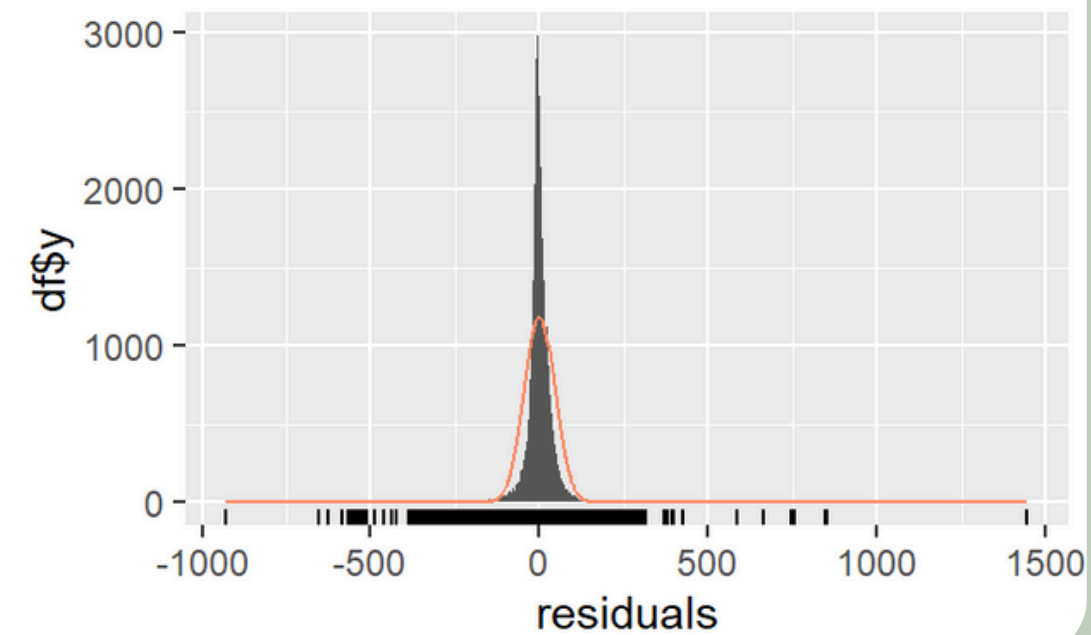
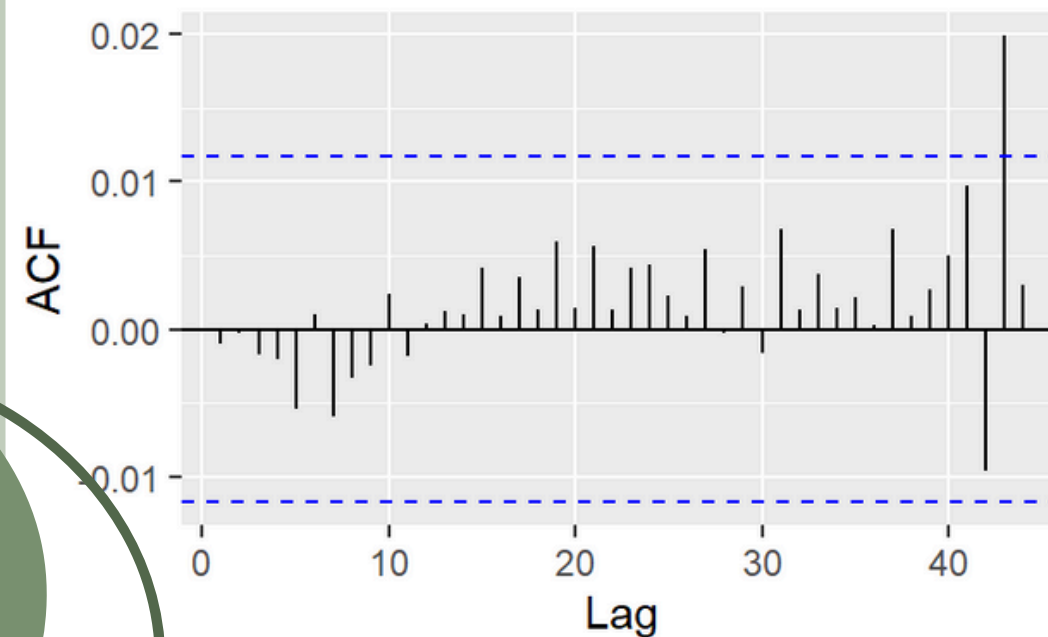
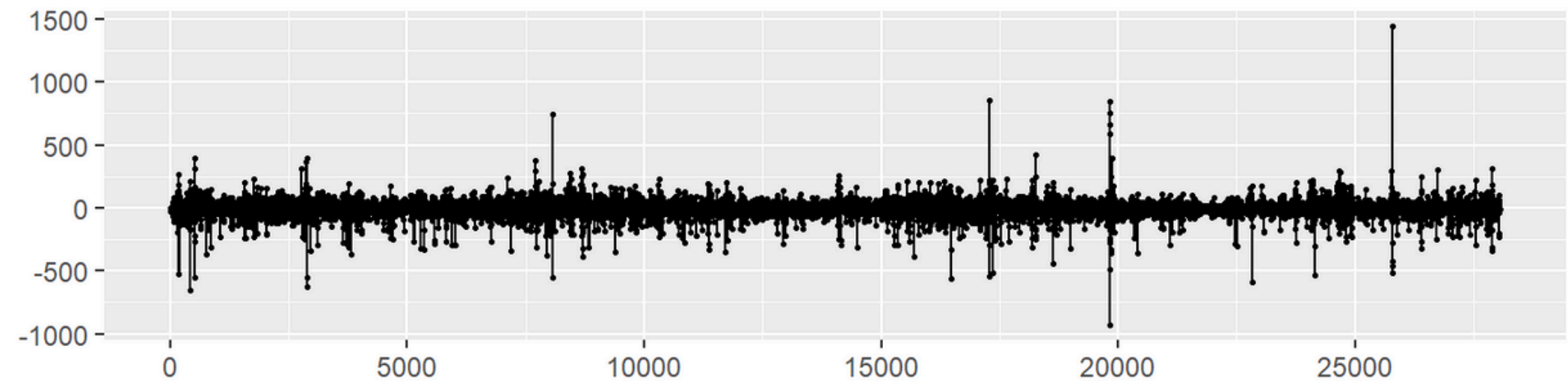
MODÈLE ARMA

Ljung-Box test

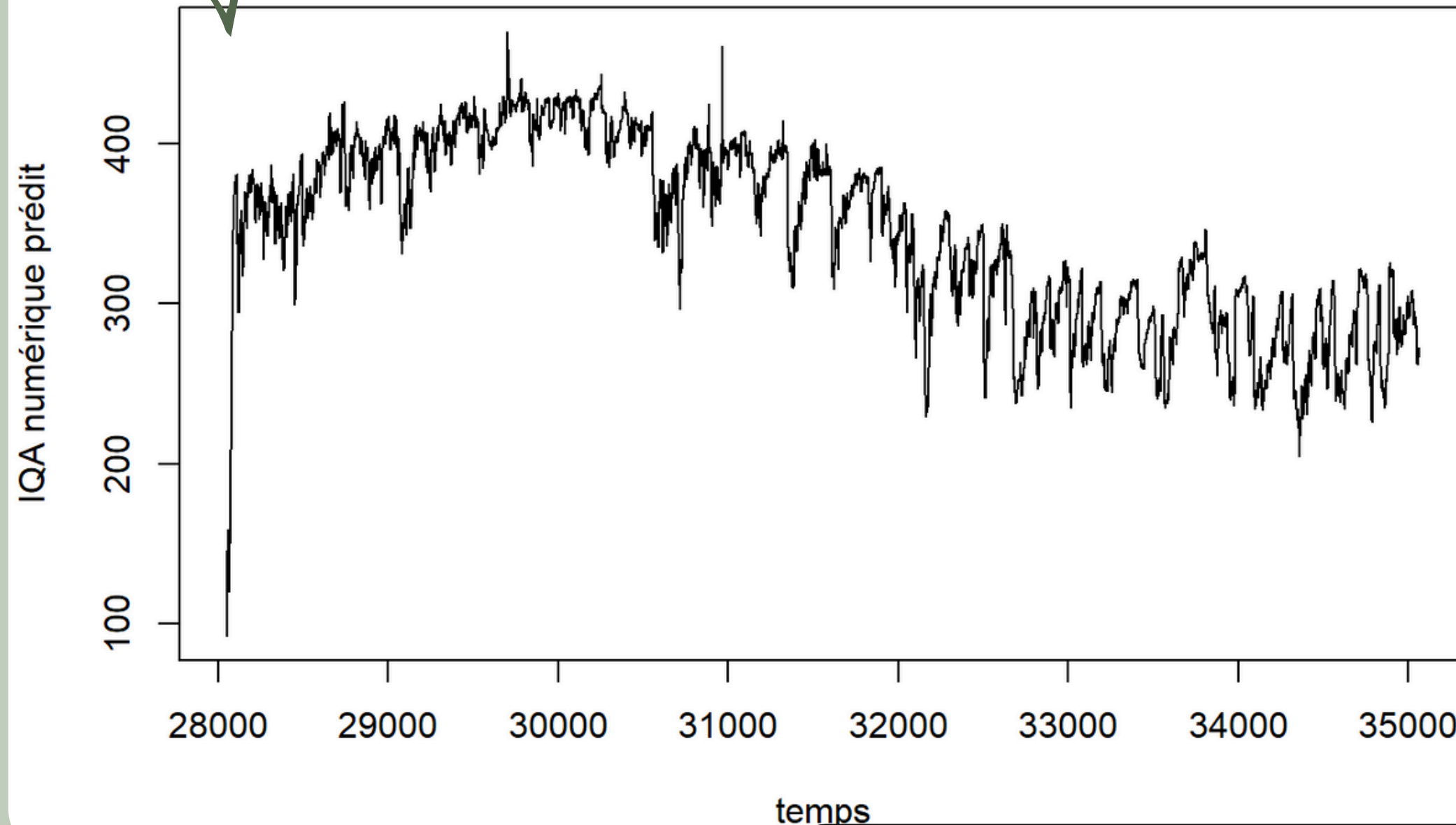
data: Residuals from Regression with ARIMA(2,0,40) errors
 $Q^* = 34.274$, $df = 3$, $p\text{-value} = 1.735e-07$

Model df: 42. Total lags used: 45

Residuals from Regression with ARIMA(2,0,40) errors

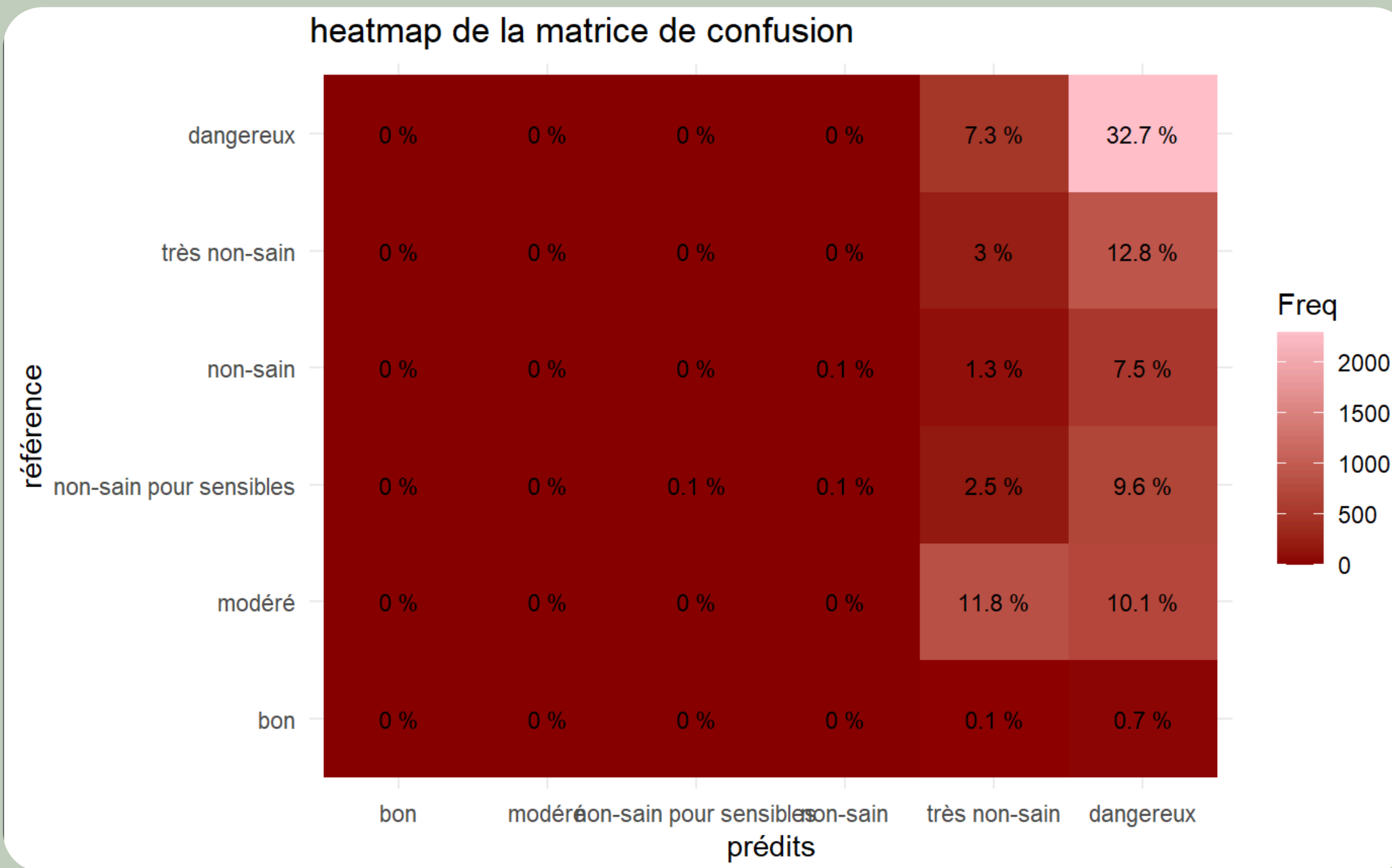


MODÈLE ARMA



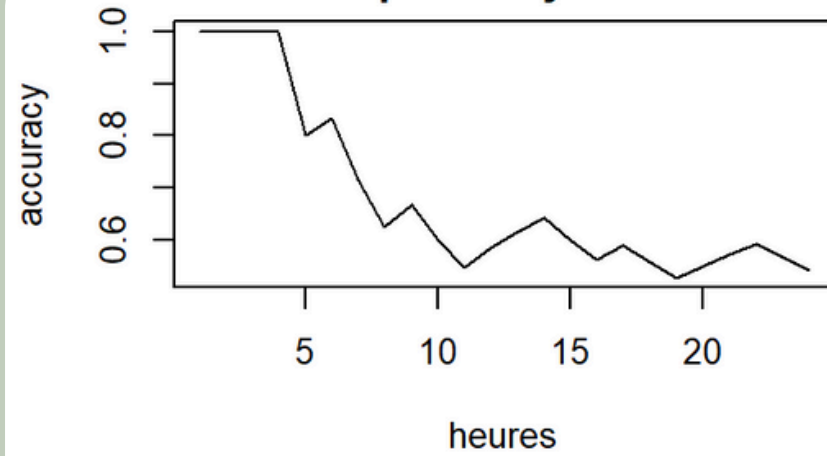
```
[1] "-----accuracy par plage de temps-----"  
accuracy 24h: 54.2 %  
accuracy 48h: 27.1 %  
accuracy 72h: 18.1 %  
accuracy 168h: 22 %  
accuracy globale: 36 %
```

MODÈLE ARMA

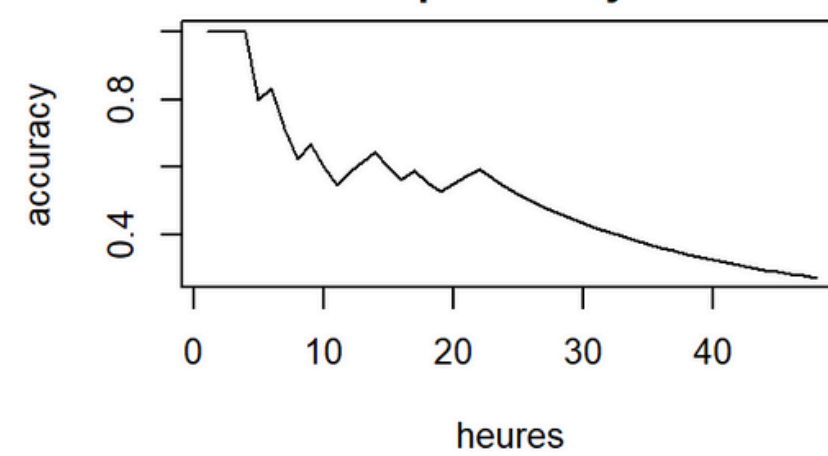


MODÈLE ARMA

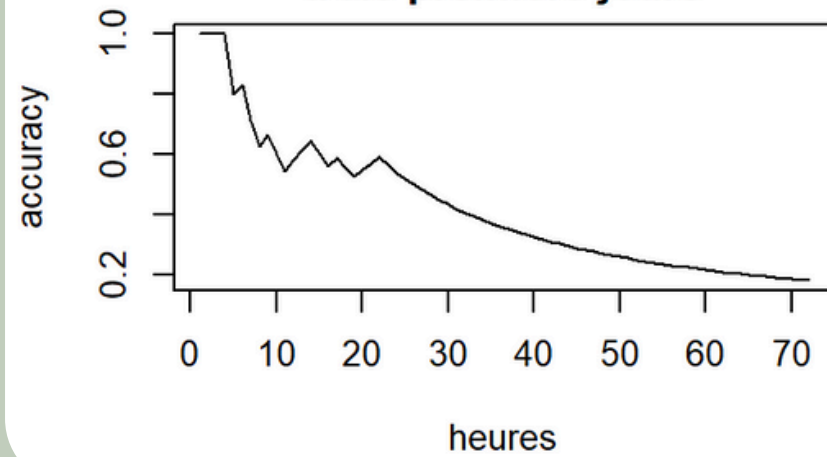
premier jour



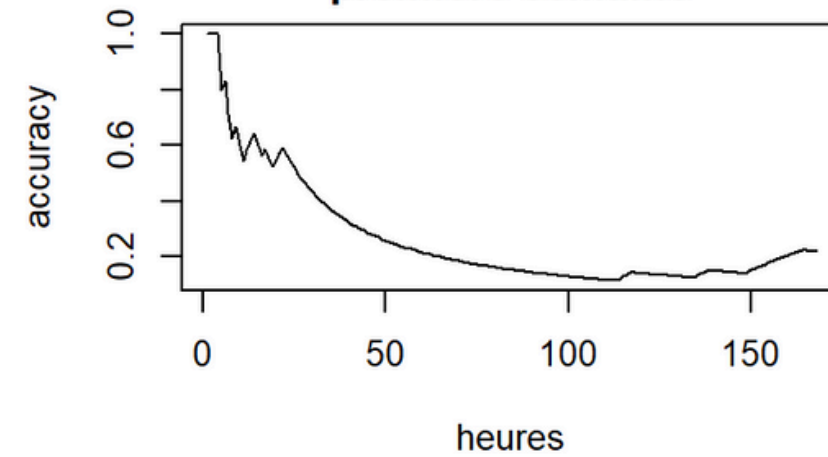
deux premiers jours



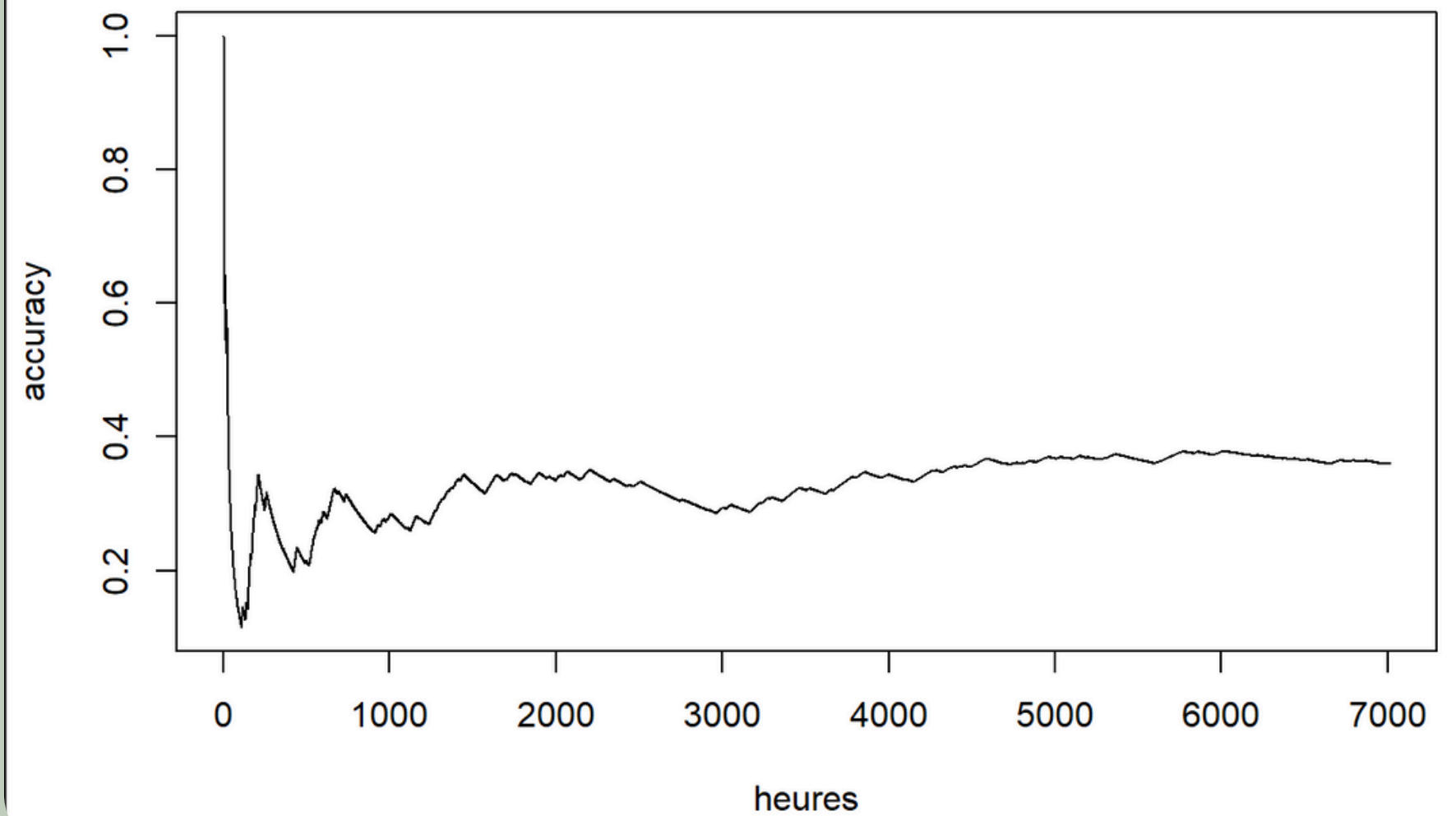
trois premiers jours



première semaine



l'accuracy cumulative au cours du temps sur tout le dataset data.test

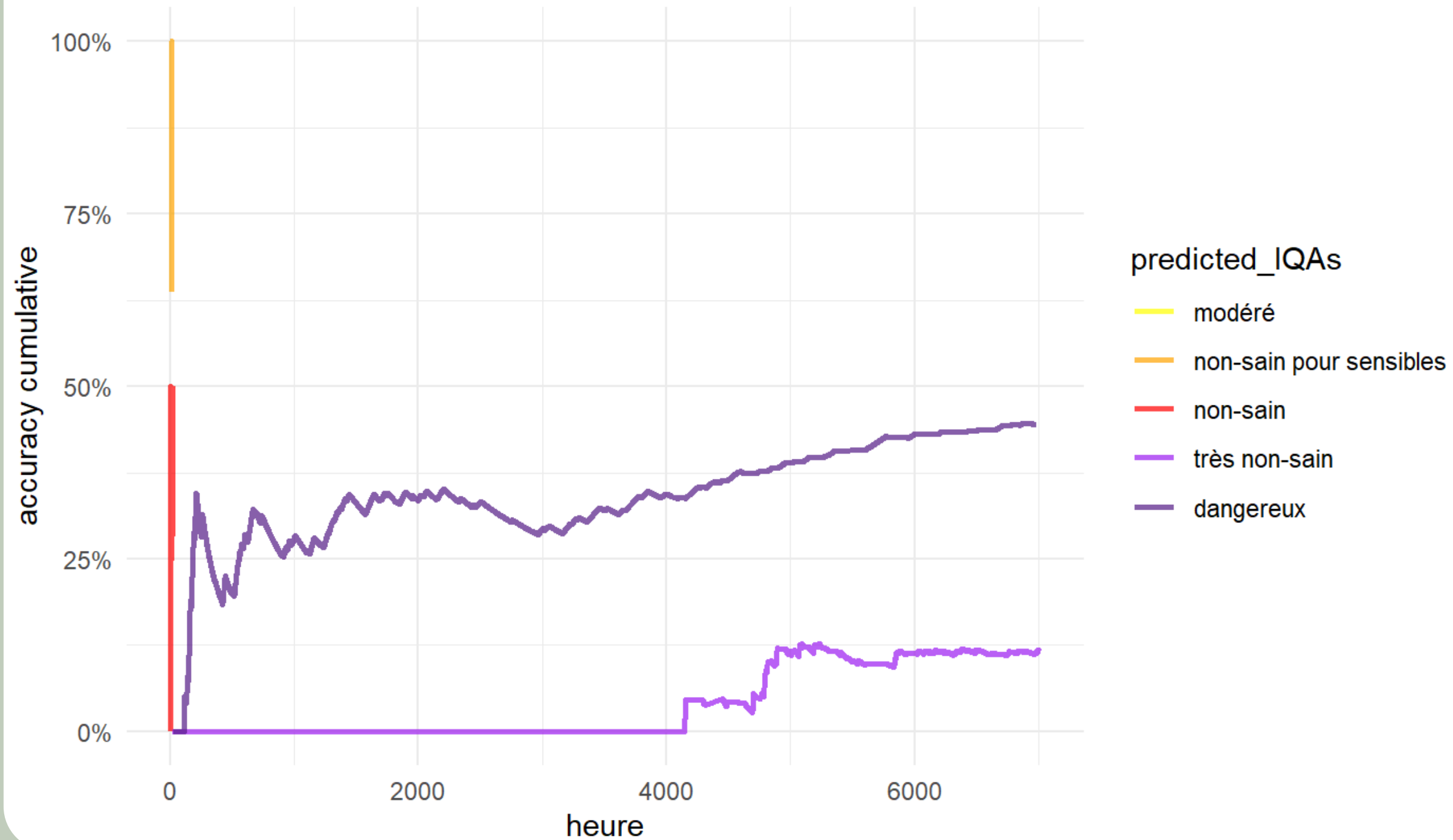


MODÈLE ARMA

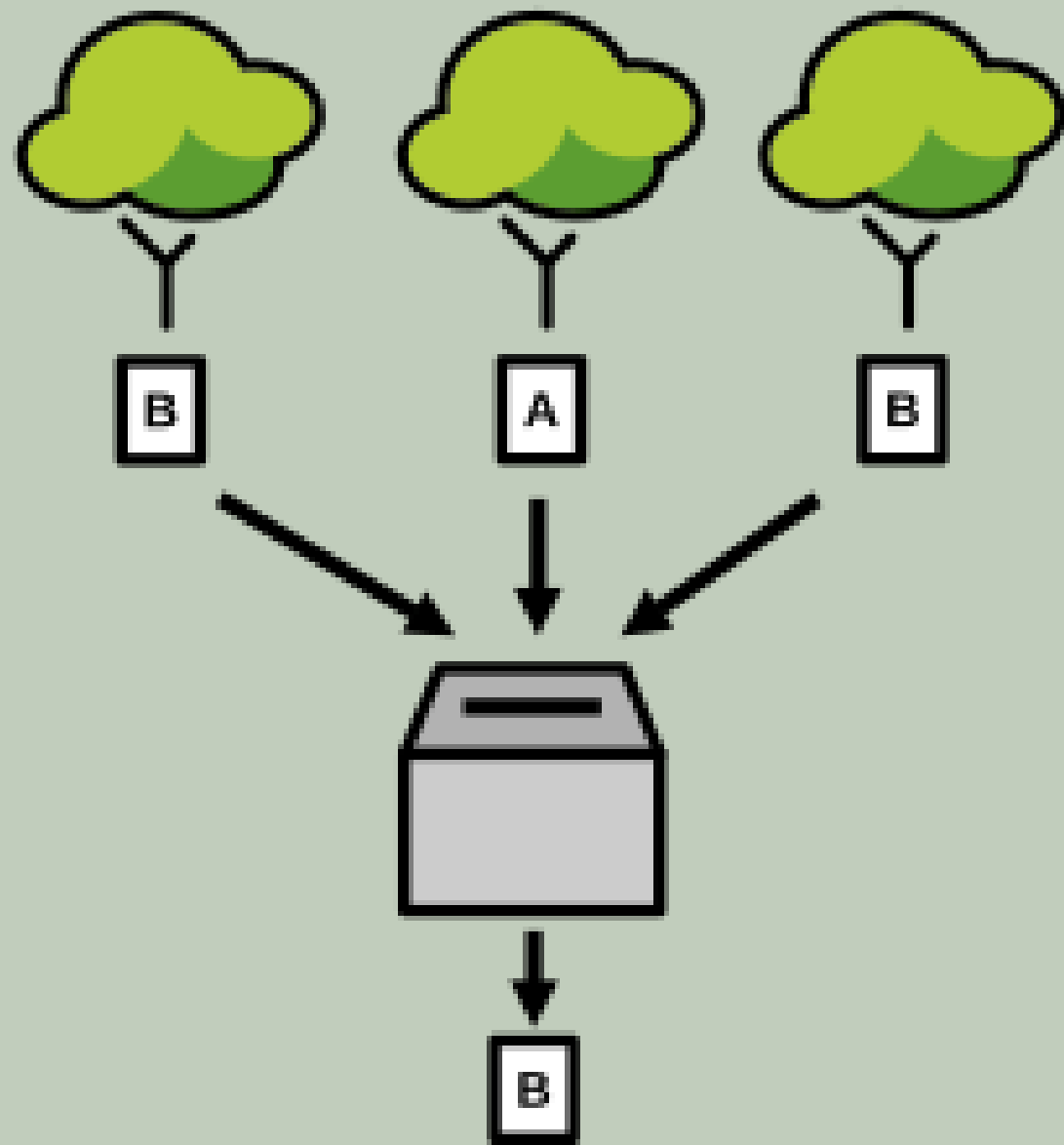
```
> head(predicted_IQAs,72)
[1] modéré non-sain pour sensibles non-sain pour sensibles
[4] non-sain pour sensibles non-sain non-sain
[7] non-sain non-sain non-sain pour sensibles
[10] non-sain pour sensibles non-sain pour sensibles non-sain pour sensibles
[13] non-sain pour sensibles non-sain pour sensibles non-sain pour sensibles
[16] non-sain pour sensibles non-sain non-sain
[19] non-sain non-sain non-sain
[22] non-sain très non-sain très non-sain
[25] très non-sain très non-sain très non-sain
[28] très non-sain très non-sain très non-sain
[31] très non-sain dangereux dangereux
[34] dangereux dangereux dangereux
[37] dangereux dangereux dangereux
[40] dangereux dangereux dangereux
[43] dangereux dangereux dangereux
[46] dangereux dangereux dangereux
[49] dangereux dangereux dangereux
[52] dangereux dangereux dangereux
[55] dangereux dangereux dangereux
[58] dangereux dangereux dangereux
[61] dangereux dangereux dangereux
[64] dangereux dangereux très non-sain
[67] très non-sain très non-sain très non-sain
[70] très non-sain très non-sain très non-sain
6 Levels: bon modéré non-sain pour sensibles non-sain ... dangereux
```

MODÈLE ARMA

accuracy cumulative de la prédiction du IQA au cours du temps par niveau



MODÈLE RANDOM FOREST



- Produire plusieurs arbres de décision
- Choisir la réponse majoritaire
- Hyperparamètres :
 - `ntree` = nombre d'arbres
 - `mtry` =
 - `min.node.size` = nb min d'individus par noeud

MODÈLE RANDOM FOREST

```
class_weights <- 1 / (table(data$IQA))
class_weights <- class_weights/sum(class_weights)

foret <- ranger(IQA ~ ., data = data.train, probability = TRUE,
               class.weights = class_weights, importance = "impurity")

prediction <- predict(foret, data.test)

pred_class <- apply(prediction$predictions, 1, which.max)
```

Type:	Probability estimation
Number of trees:	500
Sample size:	336615
Number of independent variables:	60
Mtry:	7
Target node size:	10
Variable importance mode:	impurity
Splitrule:	gini
OOB prediction error (Brier s.):	0.281381

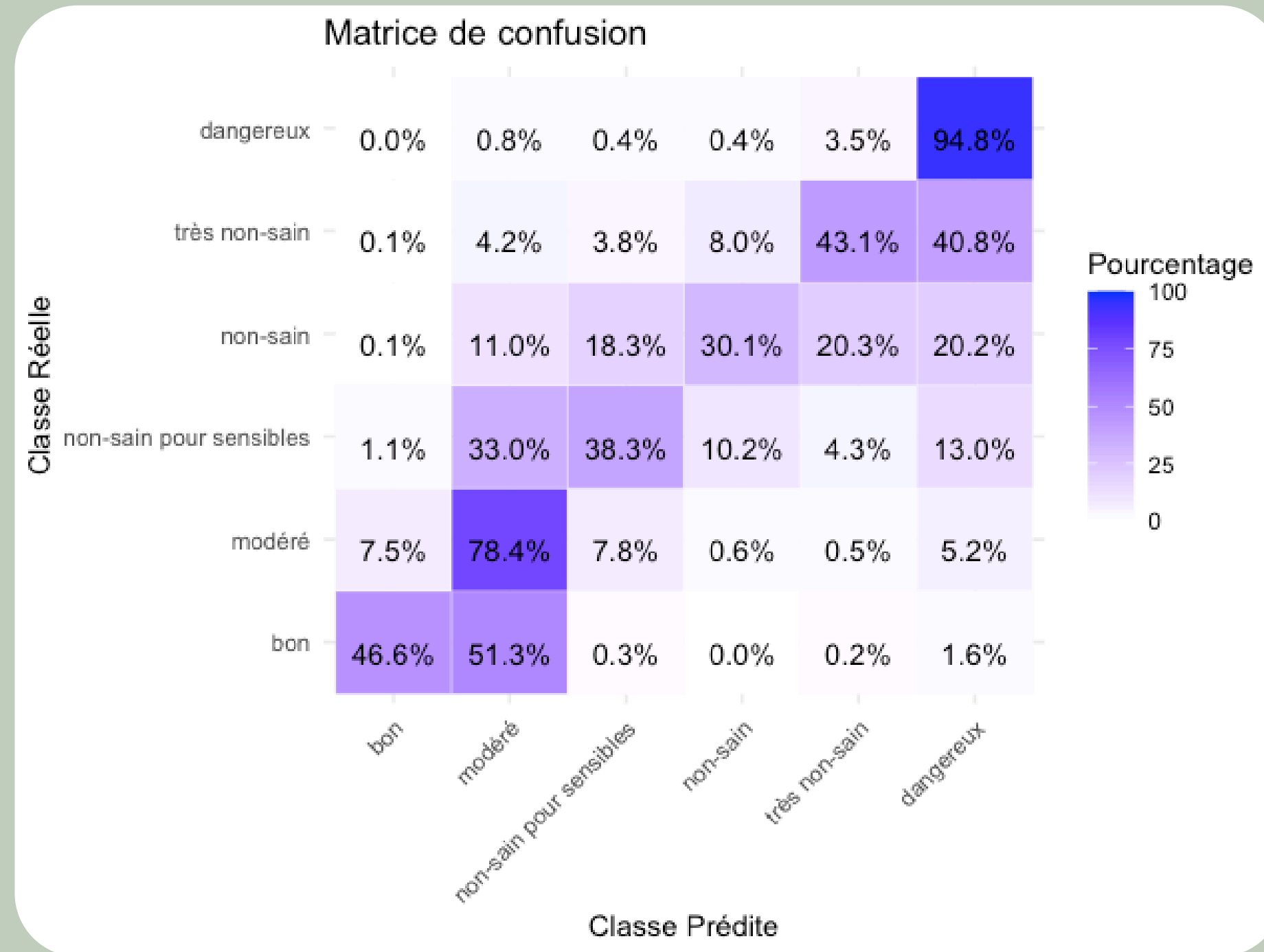


MODÈLE RANDOM FOREST



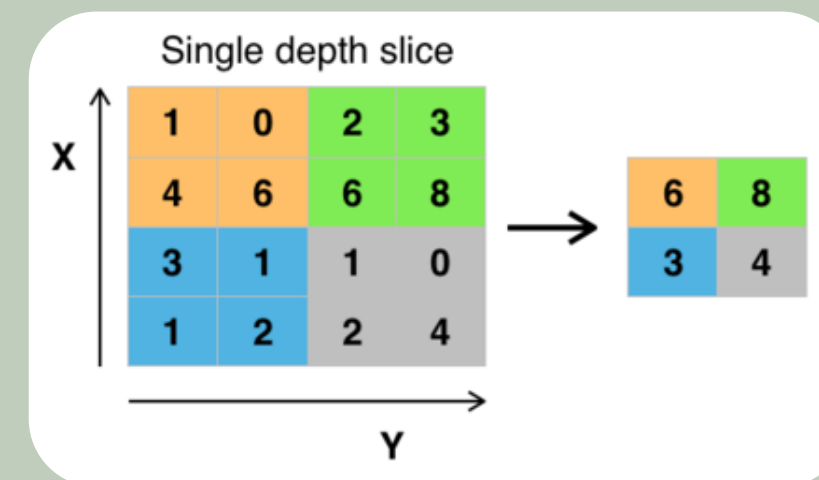
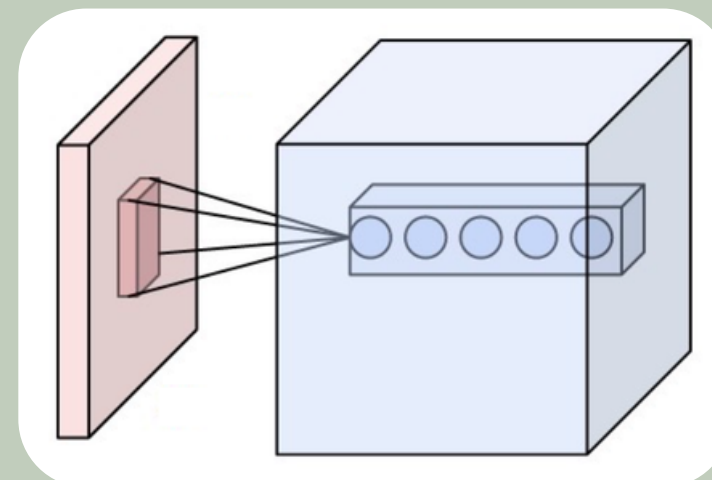
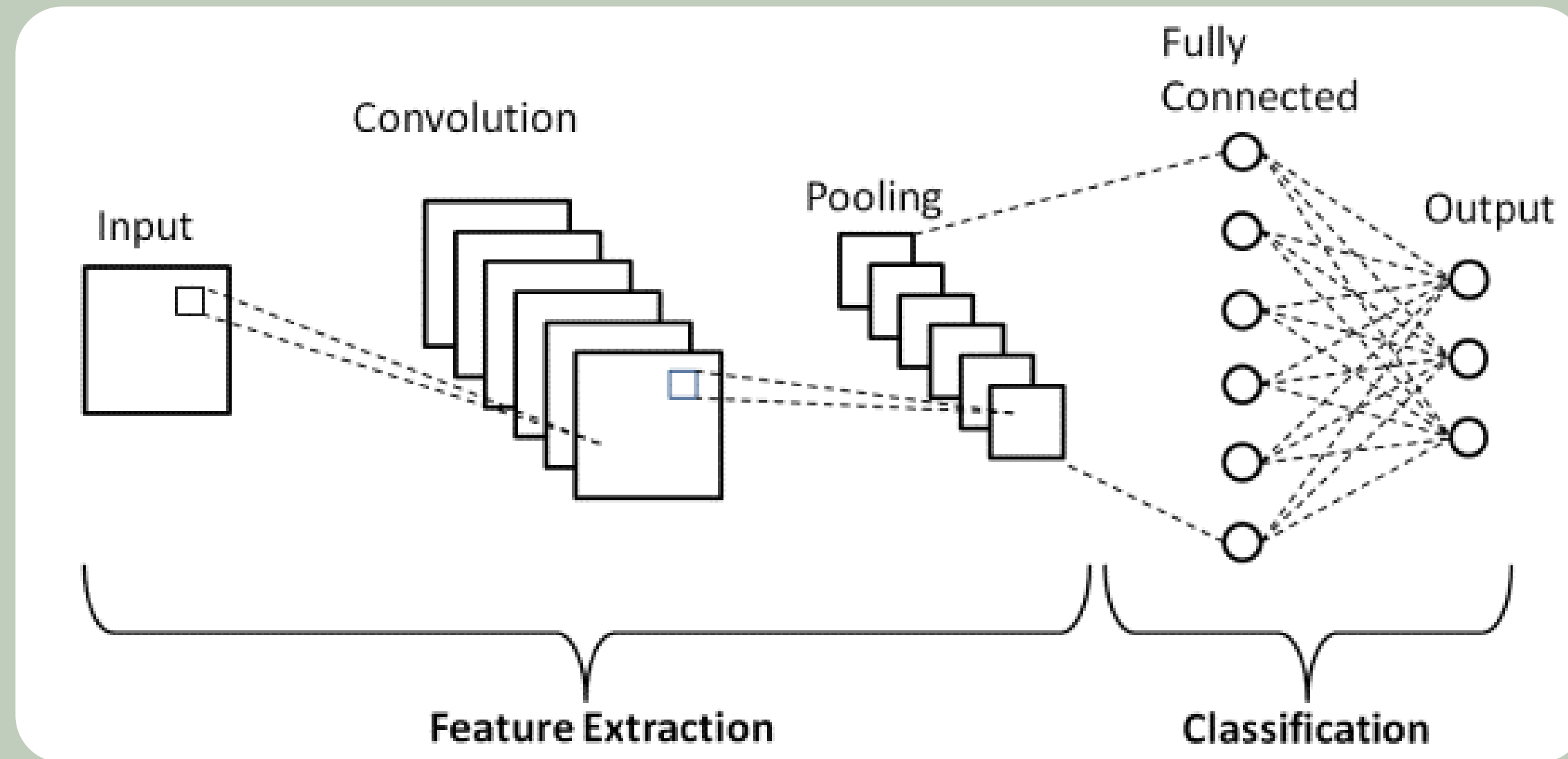
PERFORMANCE DE PREDICTION

Pourcentage
=
Sensibilité



LES CNN

(CONVOLUTIONAL NEURON NETWORK)



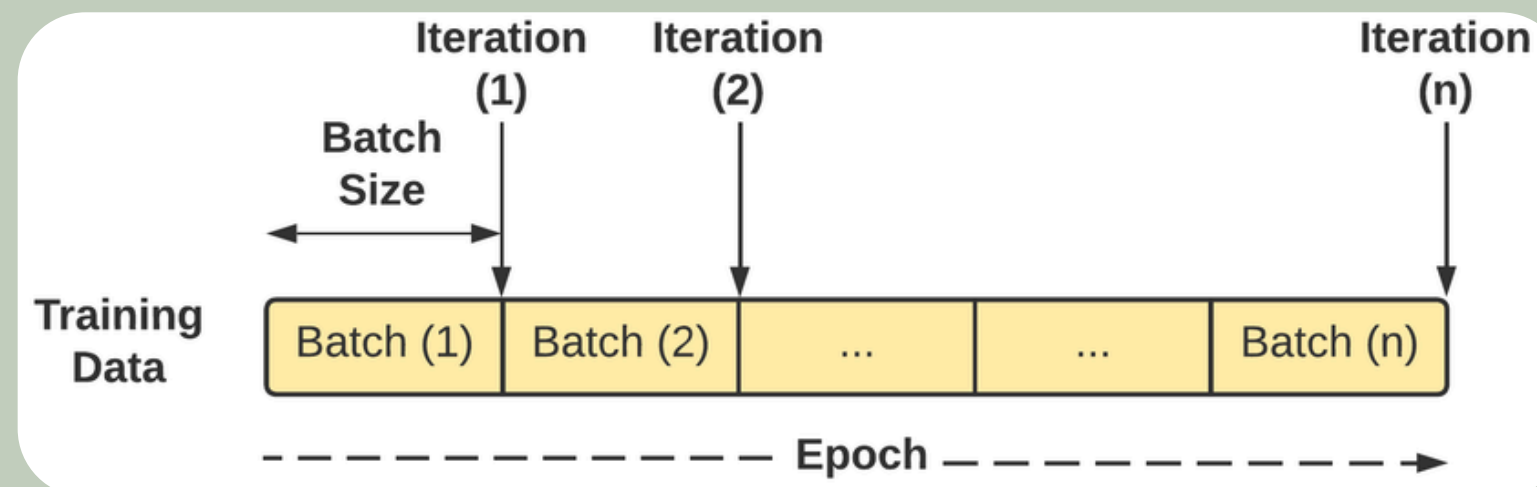
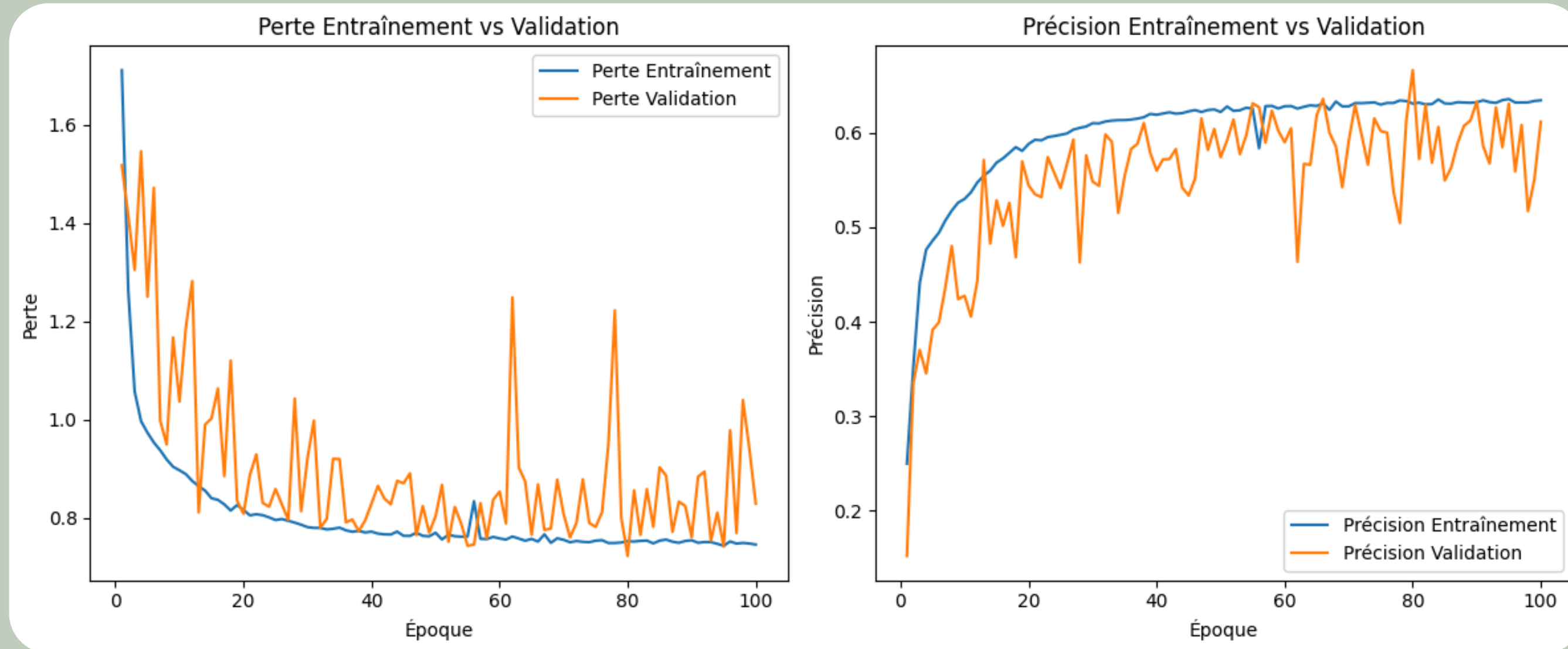
LES CNN EN PRATIQUE

AVEC PYTORCH

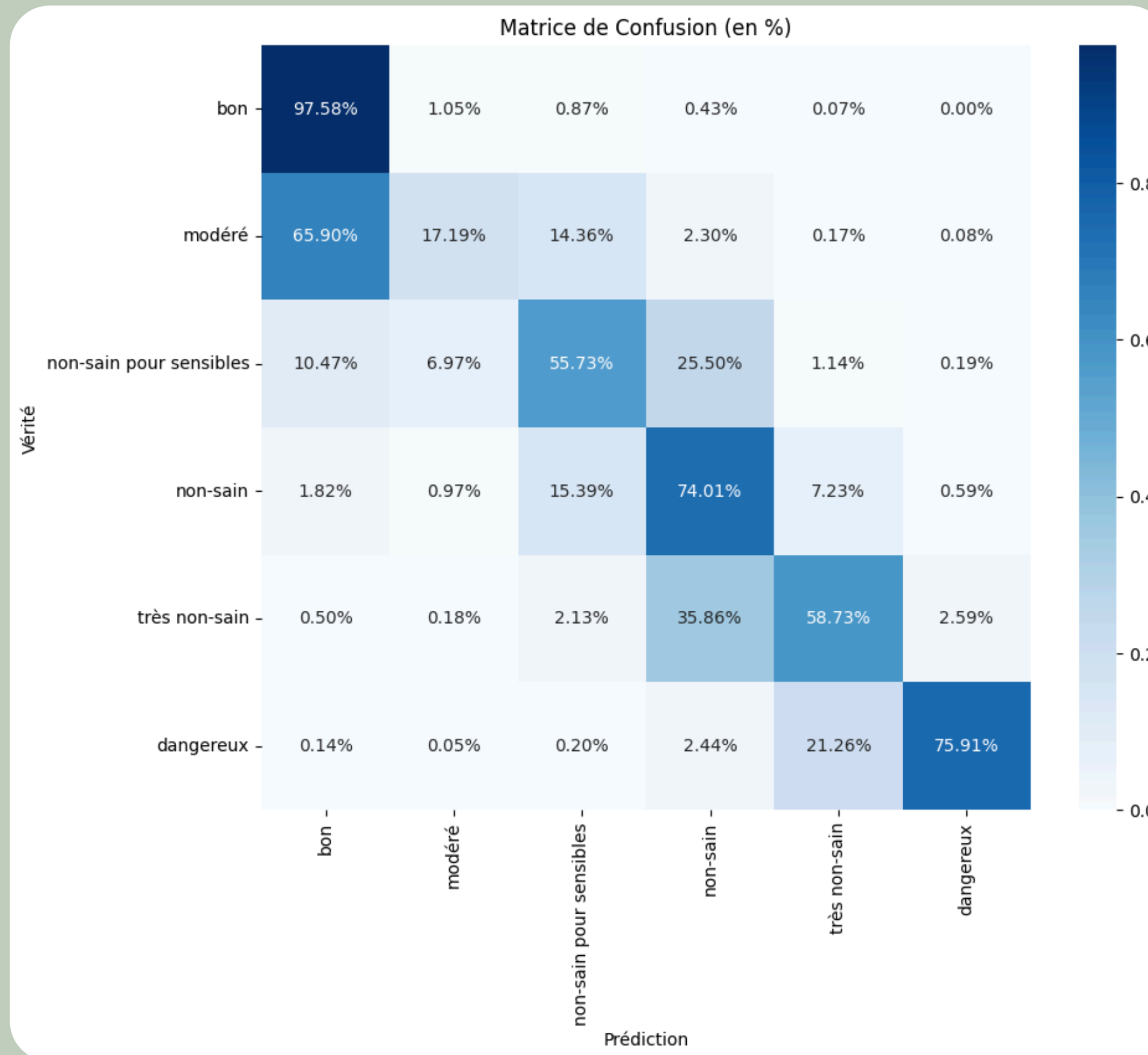
```
class Conv1DModel(nn.Module): 2 usages
    def __init__(self, input_channels, num_classes):
        super(Conv1DModel, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=input_channels, out_channels=64, kernel_size=3)
        self.conv2 = nn.Conv1d(in_channels=64, out_channels=64, kernel_size=3)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)
        self.flatten = nn.Flatten()
        # Calculer la taille après les convolutions
        # Chaque convolution sans padding réduit la longueur de la séquence de (kernel_size - 1)
        # Après deux convolutions : sequence_length - 2*(3-1) = sequence_length - 4
        self.fc1 = nn.Linear(64 * (sequence_length - 4), out_features=100)
        self.fc2 = nn.Linear(in_features=100, num_classes)

    def forward(self, x):
        # x shape: (batch_size, sequence_length, n_features)
        x = x.permute(0, 2, 1) # (batch_size, n_features, sequence_length) pour Conv1D
        x = self.relu(self.conv1(x)) # (batch_size, 64, L1)
        x = self.relu(self.conv2(x)) # (batch_size, 64, L2)
        x = self.flatten(x)         # (batch_size, 64 * L2)
        x = self.relu(self.fc1(x))  # (batch_size, 100)
        x = self.dropout(x)
        x = self.fc2(x)             # (batch_size, num_classes)
        return x
```

L'APPRENTISSAGE

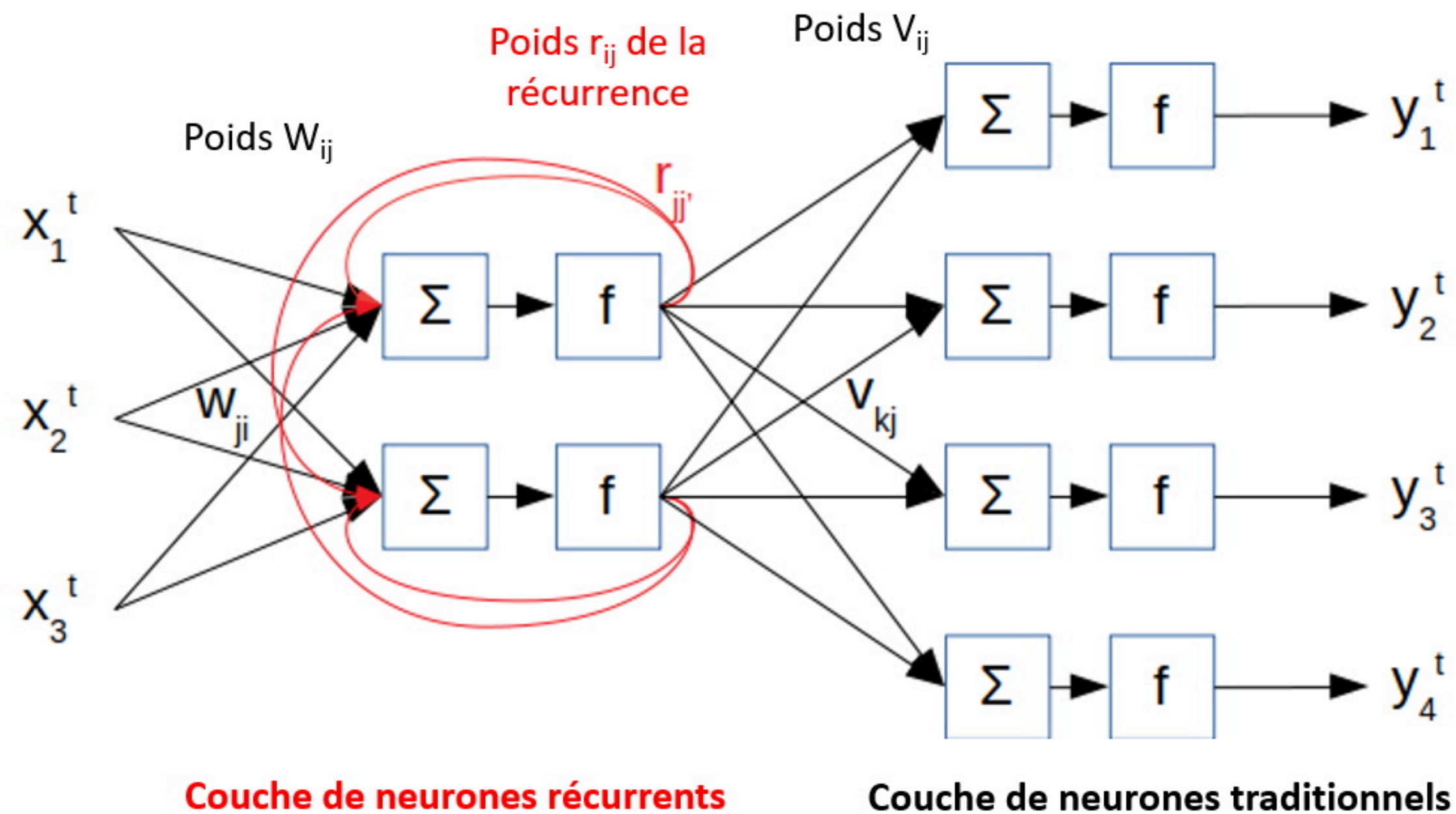


PERFORMANCES DE PRÉDICTION



PISTE D'AMÉLIORATION: LES RNN

(RECURRENT NEURON NETWORK)



COMPARAISON

	ACC	ACC/CLASSE	F1 SCORE	Ressources nécessaires	Nombre d'hyperparamètre
ARIMA	36%	<ul style="list-style-type: none">▪ bon: 0%▪ modéré: 0%▪ non-sain pour sensibles: 0%▪ non-sain: 0.1%▪ très non-sain: 3%▪ dangereux: 32.7%	<ul style="list-style-type: none">▪ bon: NA▪ modéré: 0.13%▪ non-sain pour sensibles: 1.6%▪ non-sain: 0.16%▪ très non-sain: 14.5%▪ dangereux: 57.6%	1h	2 (p, q)
Random Forest	70%	<ul style="list-style-type: none">• Bon : 0.72475• Modéré : 0.8407• Non-sain pour sensibles : 0.67107• Non-sain : 0.63657• Très non-sain : 0.69357• Dangereux : 0.8879	<ul style="list-style-type: none">• Bon : 0.53559• Modéré : 0.7002• Non-sain pour sensibles : 0.44656• Non-sain : 0.37445• Très non-sain : 0.50758• Dangereux : 0.8698	FORET : 12 MIN PREDICTION : 2 MIN	
Convolutional Neuron Network	60%	Classe bon: 0.2893 Classe modéré: 0.7921 Classe non-sain pour sensibles: 0.5593 Classe non-sain: 0.4049 Classe très non-sain: 0.4792 Classe 5 dangereux: 0.9848	Classe bon: 0.4463 Classe modéré: 0.2825 Classe non-sain pour sensibles: 0.5583 Classe non-sain: 0.5234 Classe très non-sain: 0.5272 Classe dangereux: 0.8573	3h sans GPU 30min avec GPU	4



CONCLUSION





QUESTIONS?



**MERCI POUR VOTRE
ÉCOUTE**

AUTEURS



Derya
Kapisiz



Amélie
Brejot



Achille
Gausserès