

---

## **Louis-Amaury PELISSIER**

amaurypelissier@gmail.com  
0662479067

# Météo Forecast

**Août 2022**

## **Introduction**

L'objectif de ce projet est d'utiliser des techniques de data science sur un jeu de données météorologiques afin de réaliser des prédictions.

Nous utiliserons deux types de modèles: features learning pour des prédictions à court terme et time series pour du plus long terme

Dans un premier temps nous explorerons les données et réaliserons leur prétraitement

Dans un second temps, nous évaluerons et améliorerons les différents modèles

## 0. Importation des bibliothèques

---

```
import pandas as pd
import sys
import streamlit as st
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
import seaborn as sns
import pickle
import time
import math
from sklearn.model_selection import train_test_split
from neuralprophet import NeuralProphet
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import statsmodels.api as sm
from sklearn.preprocessing import StandardScaler, normalize
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.metrics import classification_report_imbalanced, geometric_mean_score
from imblearn.under_sampling import RandomUnderSampler, ClusterCentroids
```

## 1. Exploration des données (1/2)

---

Le fichier contenant les données est ‘weatherAUS.csv’ qui contient plus de dix ans de données météorologiques journalières dans plusieurs villes d’Australie

Obtenons un premier aperçu:

```
df = pd.read_csv('./weatherAUS.csv')
df.head()
```

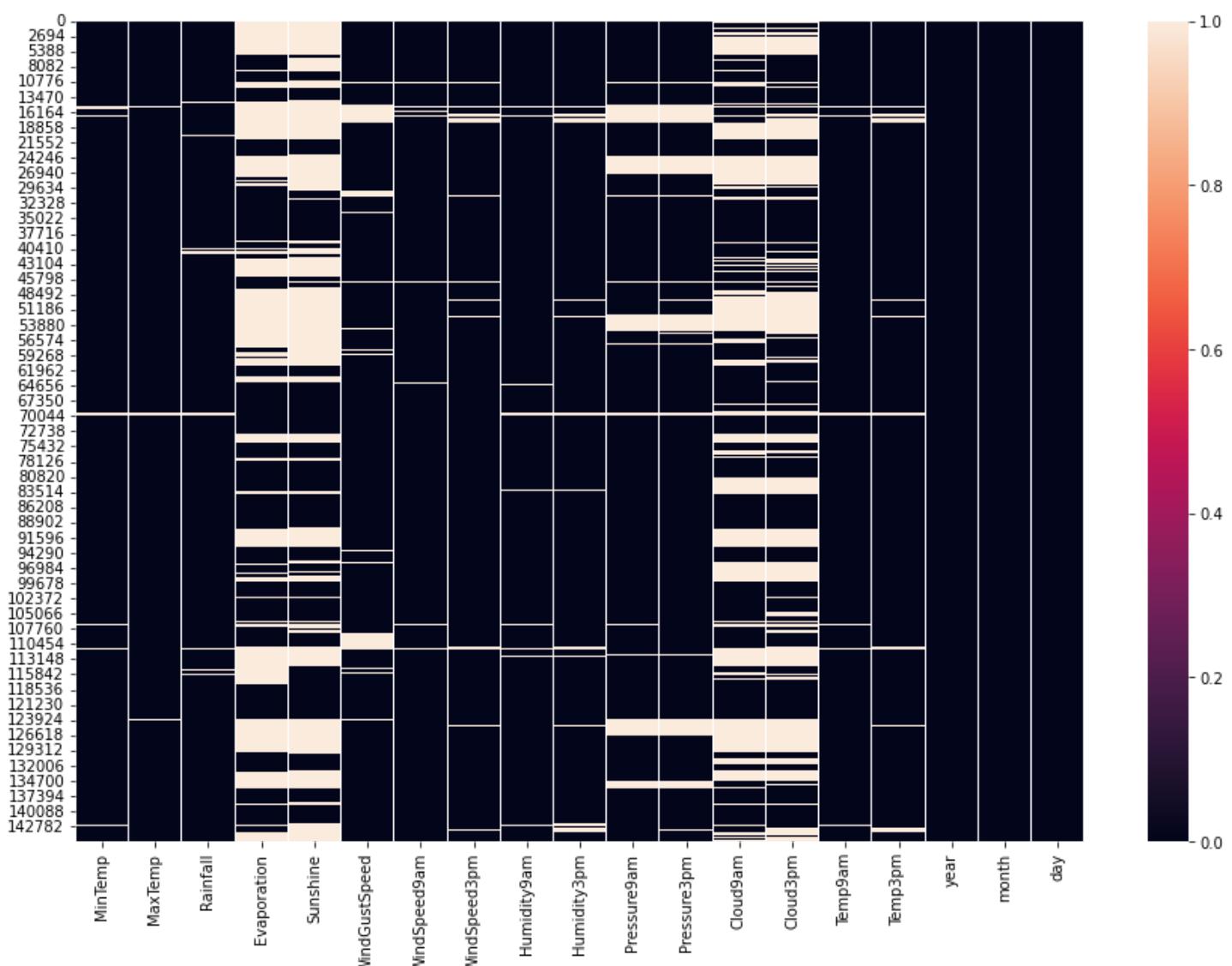
index	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	WindGustSpeed	WindDir9am	WindDir3pm	WindSpeed9am	W
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	44.0	W	WNW		20.0
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	44.0	NNW	WSW		4.0
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	46.0	W	WSW		19.0
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	24.0	SE	E		11.0
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	41.0	ENE	NW		7.0

WindDir3pm	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	Cloud9am	Cloud3pm	Temp3pm	Temp9am	Temp3pm	Temp9am	W
	20.0	24.0	71.0	22.0	1007.7	1007.1			8.0				
	4.0	22.0	44.0	25.0	1010.6	1007.8			NaN				
	19.0	26.0	38.0	30.0	1007.6	1008.7			NaN				
	11.0	9.0	45.0	16.0	1017.6	1012.8			NaN				
	7.0	20.0	82.0	33.0	1010.8	1006.0			7.0				

Il y a des données numériques (19), catégorielles (6), des valeurs non-déterminées

### Visualisation des valeurs “null”

```
plt.figure(figsize=(15,10))
sns.heatmap(df[numerical_features].isnull(),linecolor='white')
```



### Nettoyage des valeurs “null”

## Données catégorielles:

Remplacement par le “mode”

```
categorical_features = [column_name for column_name in df.columns if df[column_name].dtype == 'O']
print("Number of Categorical Features: {}".format(len(categorical_features)))
print("Categorical Features: ",categorical_features)

numerical_features = [column_name for column_name in df.columns if df[column_name].dtype != 'O']
print("Number of Numerical Features: {}".format(len(numerical_features)))
print("Numerical Features: ",numerical_features)
print(df[categorical_features].isnull().sum())

categorical_features_with_null = [feature for feature in categorical_features if df[feature].isnull().sum()]

for each_feature in categorical_features_with_null:
    mode_val = df[each_feature].mode()[0]
    df[each_feature].fillna(mode_val,inplace=True)
df[categorical_features].isnull().sum()
```

## Données numériques:

Remplacement par la moyenne des valeurs manquantes

Identification des outliers avec df.describe()

```
x = np.concatenate((x1 if isinstance(x1, np.ndarray), else np.asarray(
```

	MinTemp	MaxTemp	Rainfall	Evaporation	\
count	143975.000000	144199.000000	142199.000000	82670.000000	
mean	12.194034	23.221348	2.360918	5.468232	
std	6.398495	7.119049	8.478060	4.193704	
min	-8.500000	-4.800000	0.000000	0.000000	
25%	7.600000	17.900000	0.000000	2.600000	
50%	12.000000	22.600000	0.000000	4.800000	
75%	16.900000	28.200000	0.800000	7.400000	
max	33.900000	48.100000	371.000000	145.000000	
	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	\
count	75625.000000	135197.000000	143693.000000	142398.000000	
mean	7.611178	40.035230	14.043426	18.662657	
std	3.785483	13.607062	8.915375	8.809800	
min	0.000000	6.000000	0.000000	0.000000	
25%	4.800000	31.000000	7.000000	13.000000	
50%	8.400000	39.000000	13.000000	19.000000	
75%	10.600000	48.000000	19.000000	24.000000	
max	14.500000	135.000000	130.000000	87.000000	

	Humidity9am	Humidity3pm	Pressure9am	Pressure3pm	\
count	142806.000000	140953.000000	130395.000000	130432.000000	
mean	68.880831	51.539116	1017.64994	1015.255889	
std	19.029164	20.795902	7.10653	7.037414	
min	0.000000	0.000000	980.50000	977.100000	
25%	57.000000	37.000000	1012.90000	1010.400000	
50%	70.000000	52.000000	1017.60000	1015.200000	
75%	83.000000	66.000000	1022.40000	1020.000000	
max	100.000000	100.000000	1041.00000	1039.600000	
	Cloud9am	Cloud3pm	Temp9am	Temp3pm	year
count	89572.000000	86102.000000	143693.000000	141851.00000	145460.000000
mean	4.447461	4.509930	16.990631	21.68339	2012.769751
std	2.887159	2.720357	6.488753	6.93665	2.537684
min	0.000000	0.000000	-7.200000	-5.40000	2007.000000
25%	1.000000	2.000000	12.300000	16.60000	2011.000000
50%	5.000000	5.000000	16.700000	21.10000	2013.000000
75%	7.000000	7.000000	21.600000	26.40000	2015.000000
max	9.000000	9.000000	40.200000	46.70000	2017.000000
	month	day			
count	145460.000000	145460.000000			
mean	6.399615	15.712258			
std	3.427262	8.794789			
min	1.000000	1.000000			
25%	3.000000	8.000000			
50%	6.000000	16.000000			
75%	9.000000	23.000000			
max	12.000000	31.000000			

et remplacement de ces outliers par les limites des IQR (interquartile range)

```
features_with_outliers = ['MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'WindGustSpeed','WindSpeed9am', 'WindSpeed3pm', 'Humidity']

for feature in features_with_outliers:
    q1 = df[feature].quantile(0.25)
    q3 = df[feature].quantile(0.75)
    IQR = q3-q1
    lower_limit = q1 - (IQR*1.5)
    upper_limit = q3 + (IQR*1.5)
    df.loc[df[feature]<lower_limit,feature] = lower_limit
    df.loc[df[feature]>upper_limit,feature] = upper_limit

plt.figure(figsize=(15,10))
ax = df[features_with_outliers].plot(kind='box', title='boxplot')
for feature in numerical_features:

    plt.figure(figsize=(4,4))
    sns.boxplot(df[feature])
    plt.title(feature)

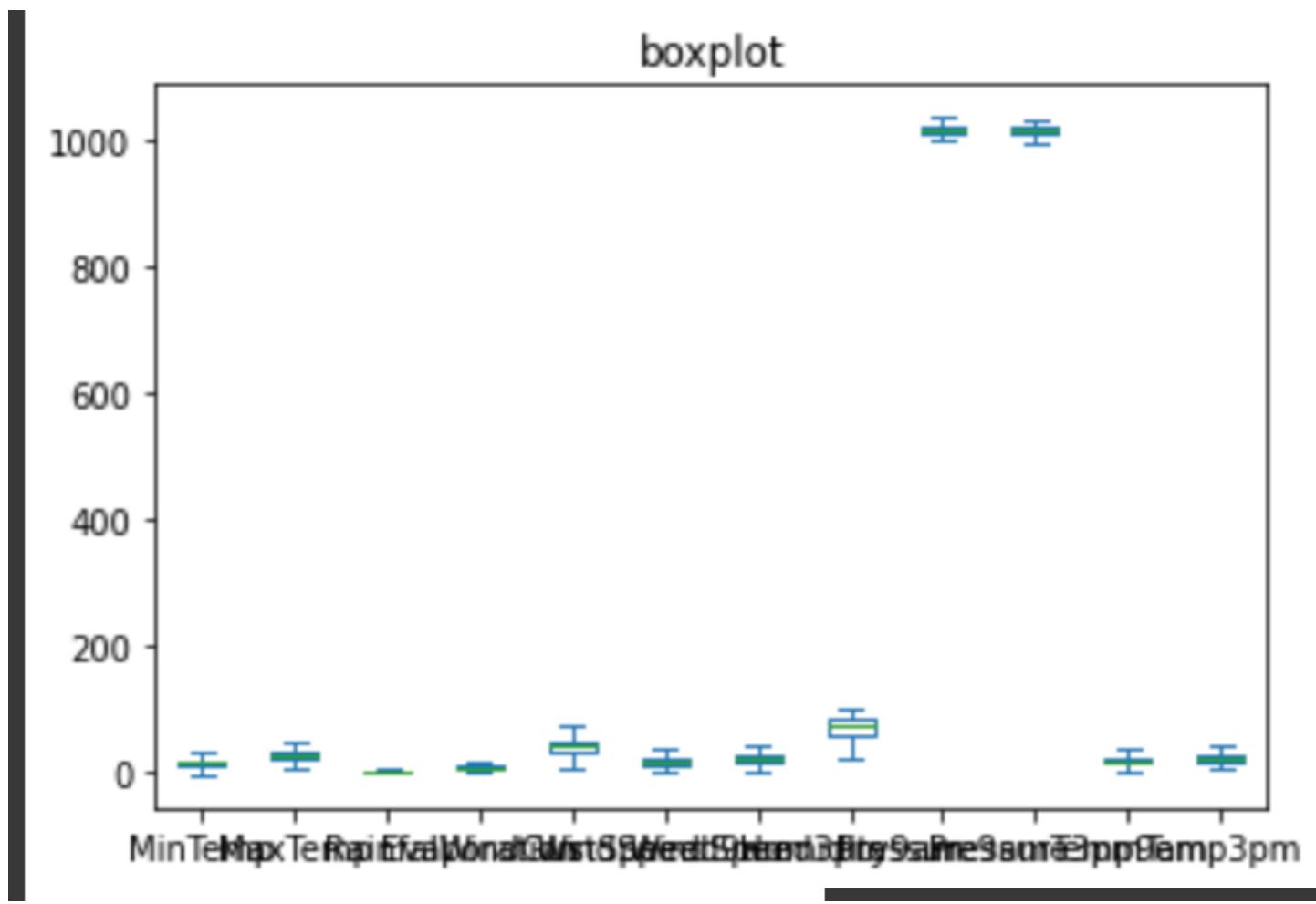
# list of numerical Features with Null values:

numerical_features_with_null = [feature for feature in numerical_features if df[feature].isnull().sum()]
# Filling null values using mean:

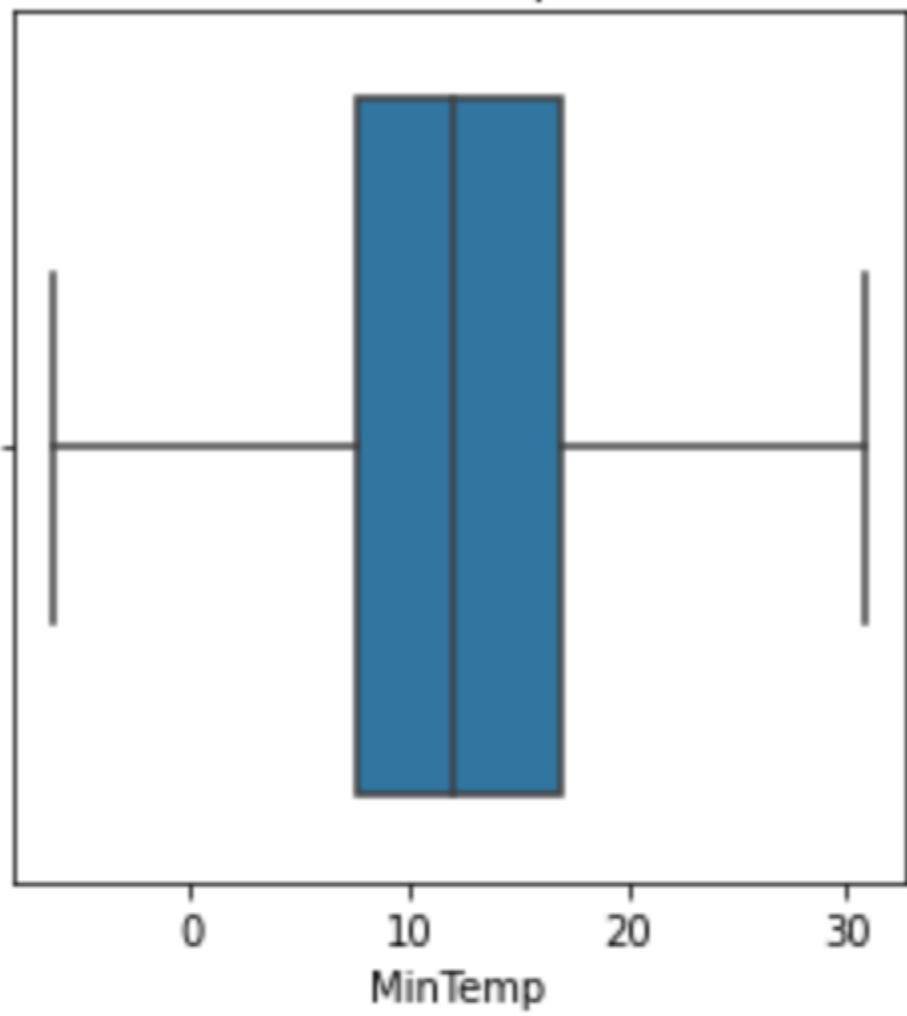
for feature in numerical_features_with_null:
    mean_value = df[feature].mean()
    df[feature].fillna(mean_value,inplace=True)
```

Les données numériques sont maintenant nettoyées comme l'attestent les graphiques boxplot.

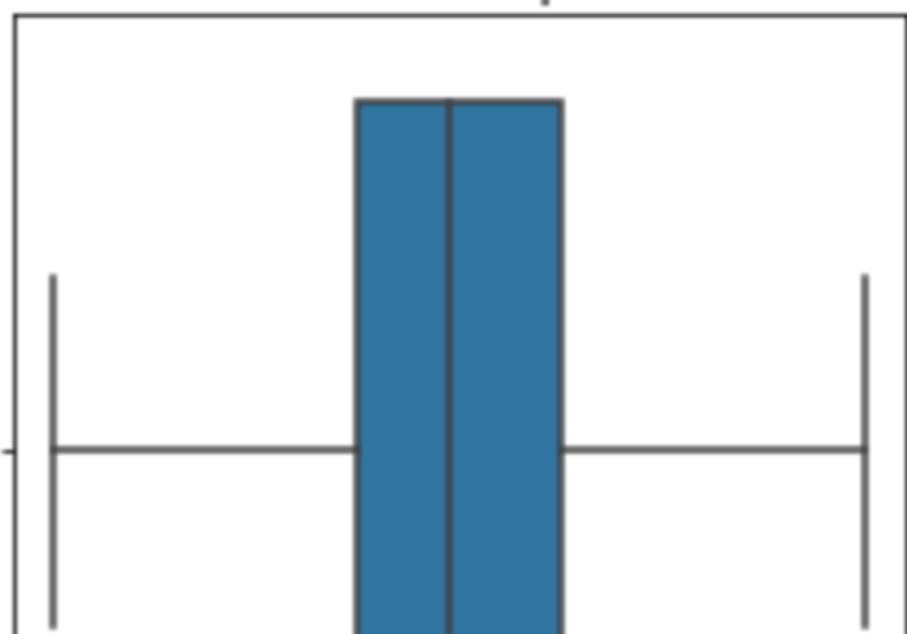
Toutefois une standardisation des données sera nécessaire car leur valeurs sont hétérogènes:



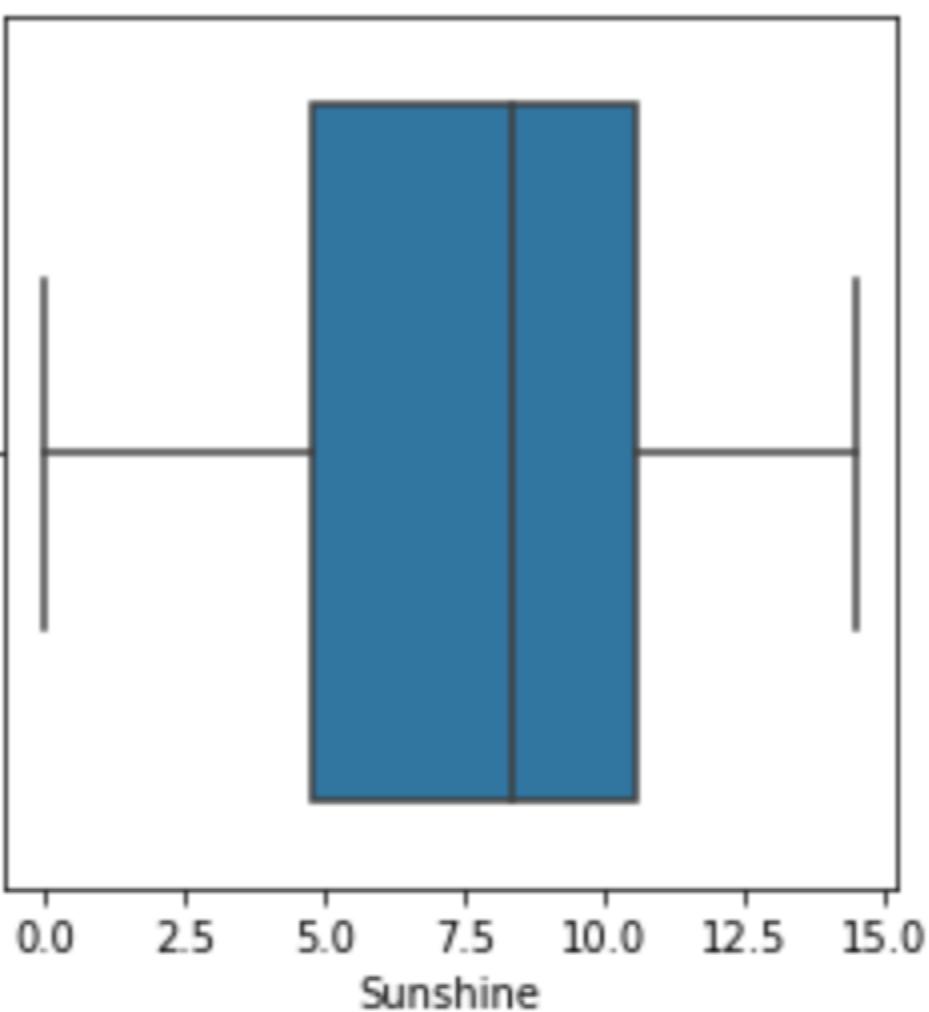
MinTemp



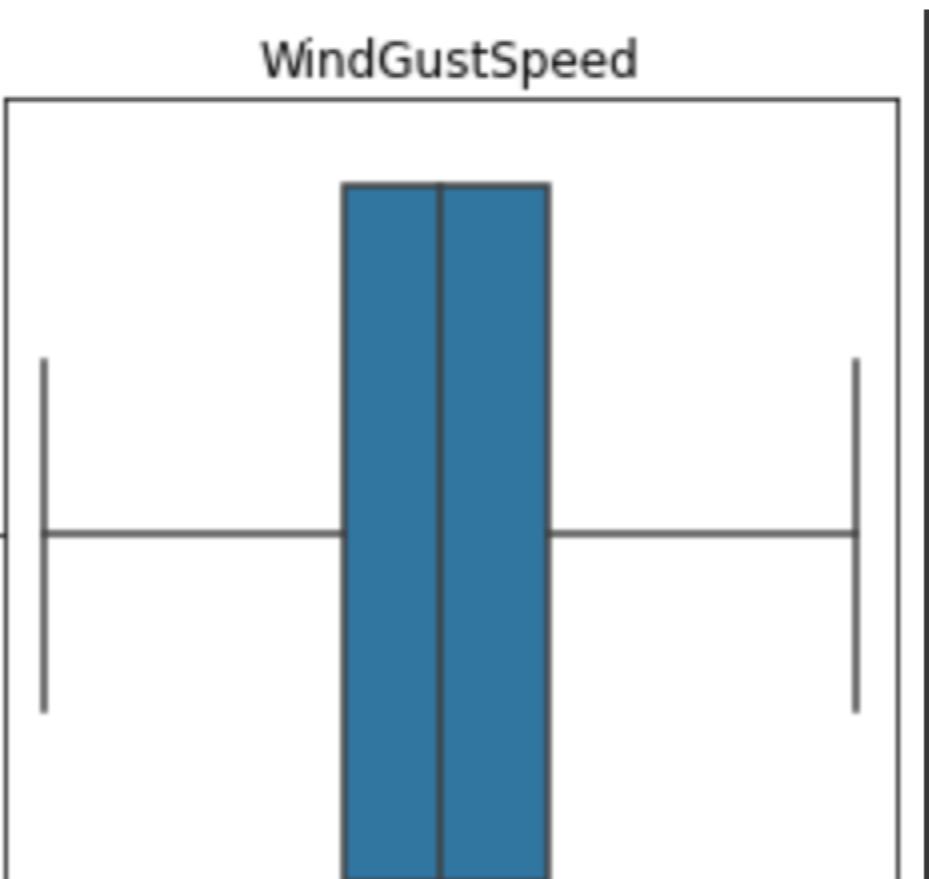
MaxTemp



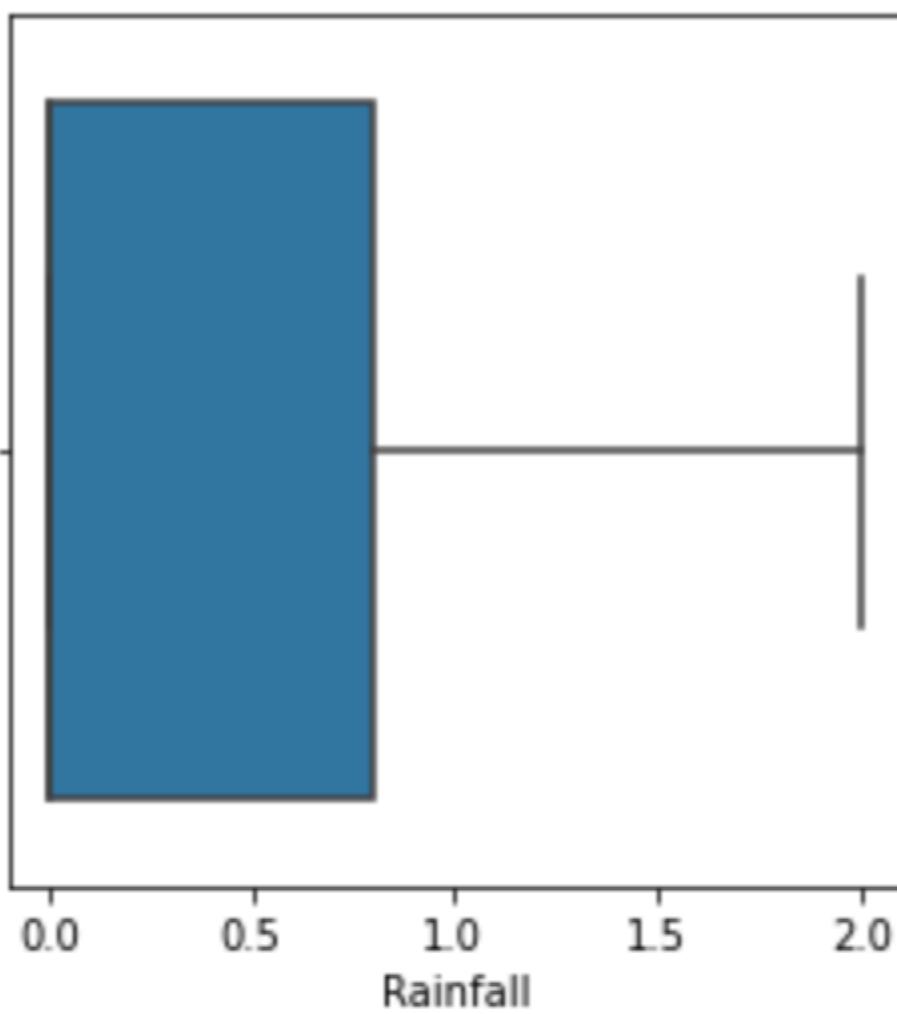
Sunshine



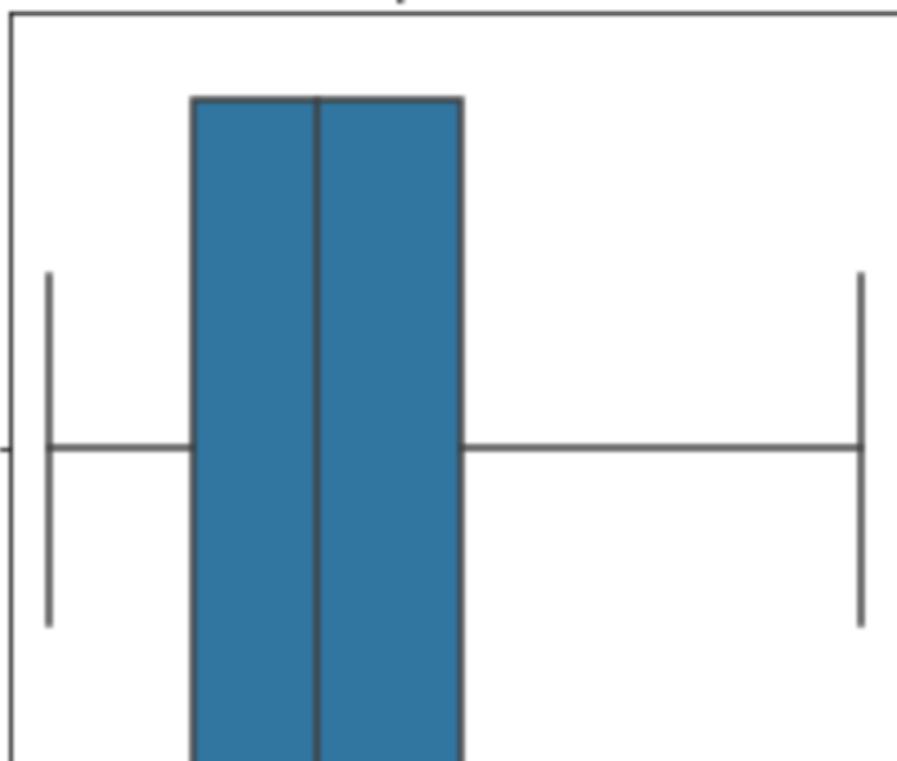
WindGustSpeed



Rainfall



Evaporation



## Valeurs catégorielles

On transforme les catégories en indicateurs numériques

```
df['RainToday'].replace({'No':0, 'Yes': 1}, inplace = True)

df['RainTomorrow'].replace({'No':0, 'Yes': 1}, inplace = True)
df['RainTomorrow'].value_counts().plot(kind='bar')
#
def encode_data(feature_name):
    mapping_dict = {}
    unique_values = list(df[feature_name].unique())
    for idx in range(len(unique_values)):
        mapping_dict[unique_values[idx]] = idx
    print(mapping_dict)
    return mapping_dict

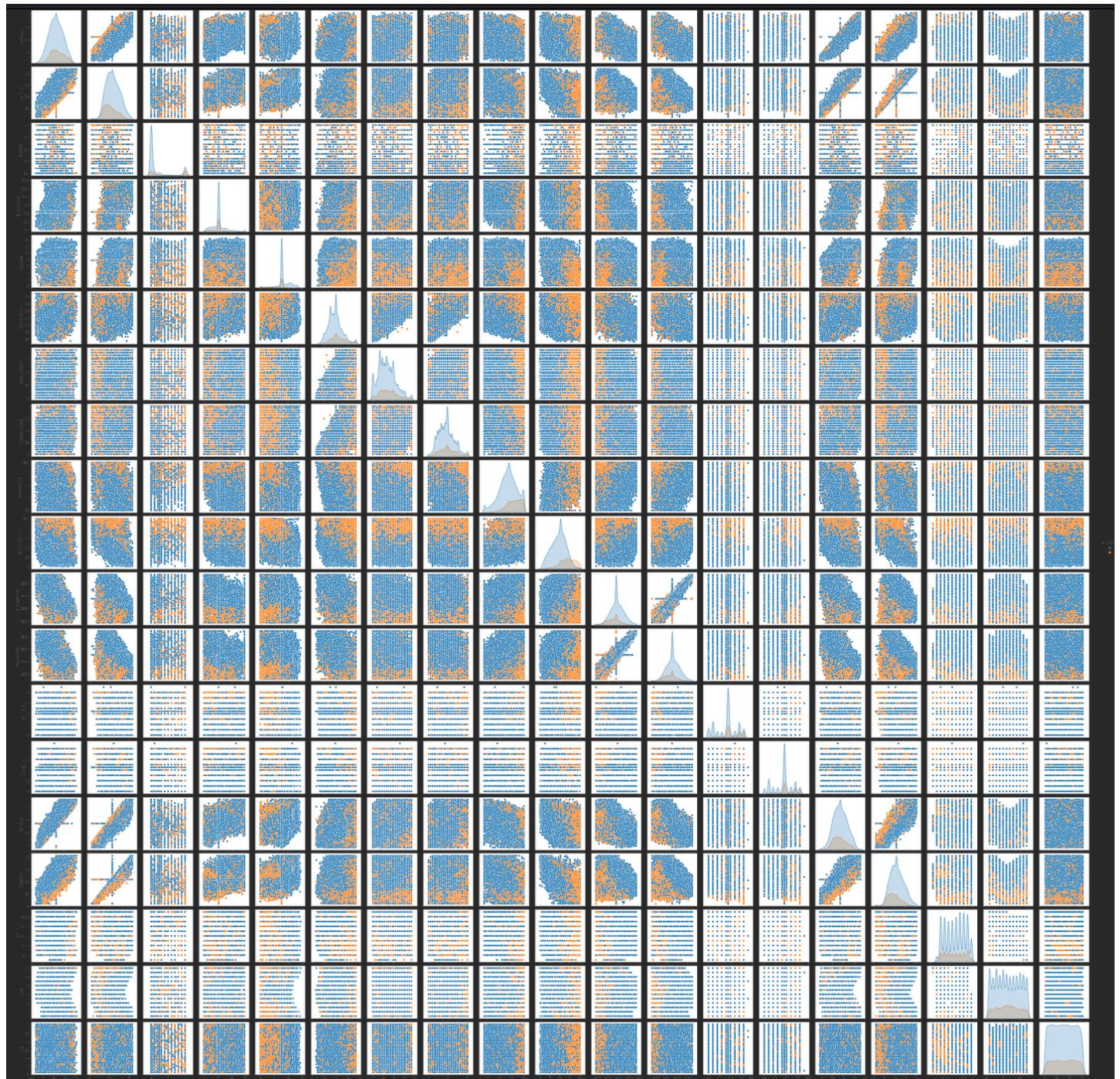
df['WindGustDir'].replace(encode_data('WindGustDir'),inplace = True)
df['WindDir9am'].replace(encode_data('WindDir9am'),inplace = True)
df['WindDir3pm'].replace(encode_data('WindDir3pm'),inplace = True)
df['Location'].replace(encode_data('Location'), inplace = True)
```



## **2. Exploration des données (2/2)**

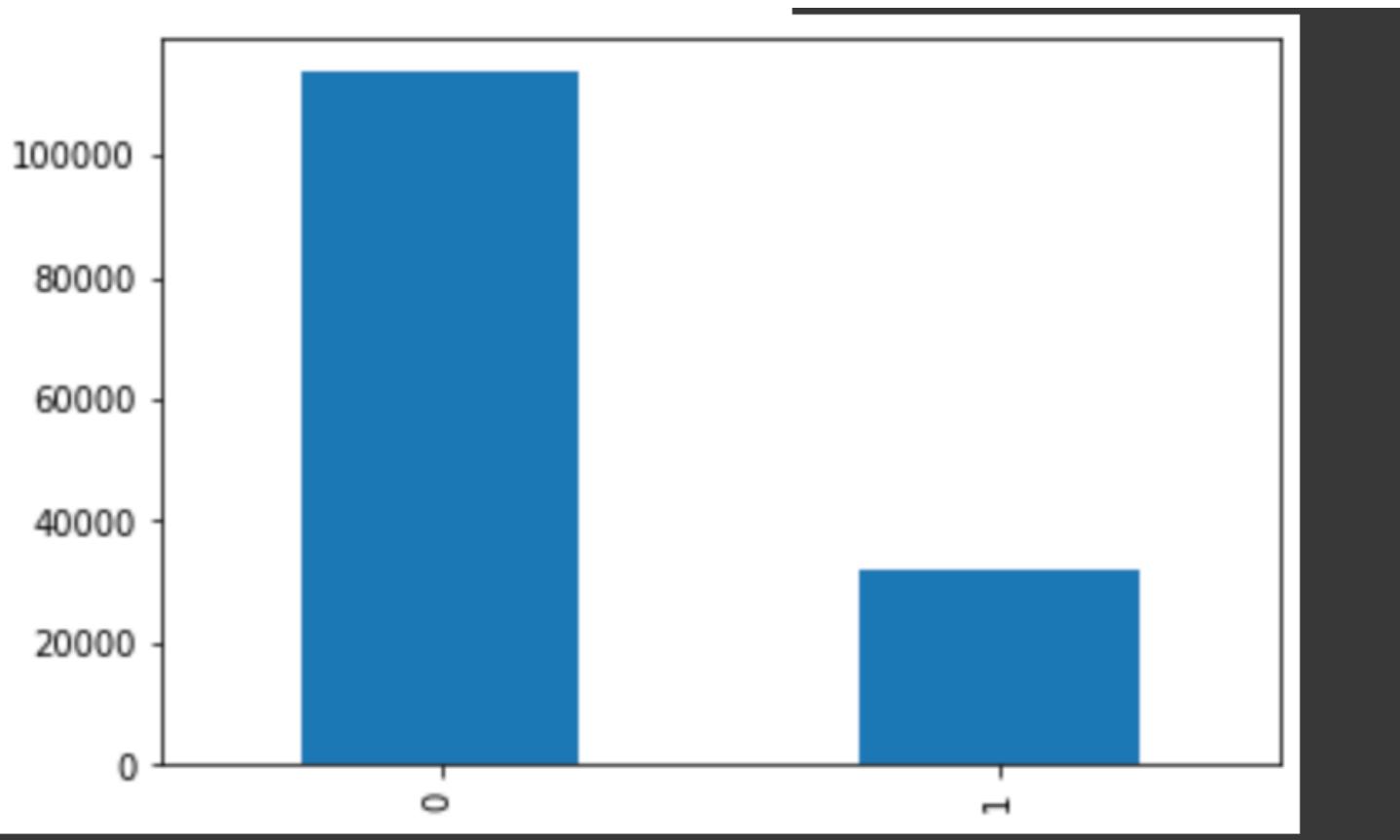
Maintenant que les données sont nettoyées, nous pouvons réaliser une visualisation des corrélations entre les différentes colonnes

A l'aide de Seaborn, exécutons un “pairplot” sur l'ensemble du dataset



Ici la variable colorée est “RainToday”. On remarque de nombreuses relations linéaires avec différentes variables, par exemple avec la “pression” et la “température”

Intéressons-nous à la variable cible “RainTomorrow”  
Notamment sa distribution:



On remarque que les classes de “RainTomorrow” sont déséquilibrées

On effectuera donc différents resampling par la suite.

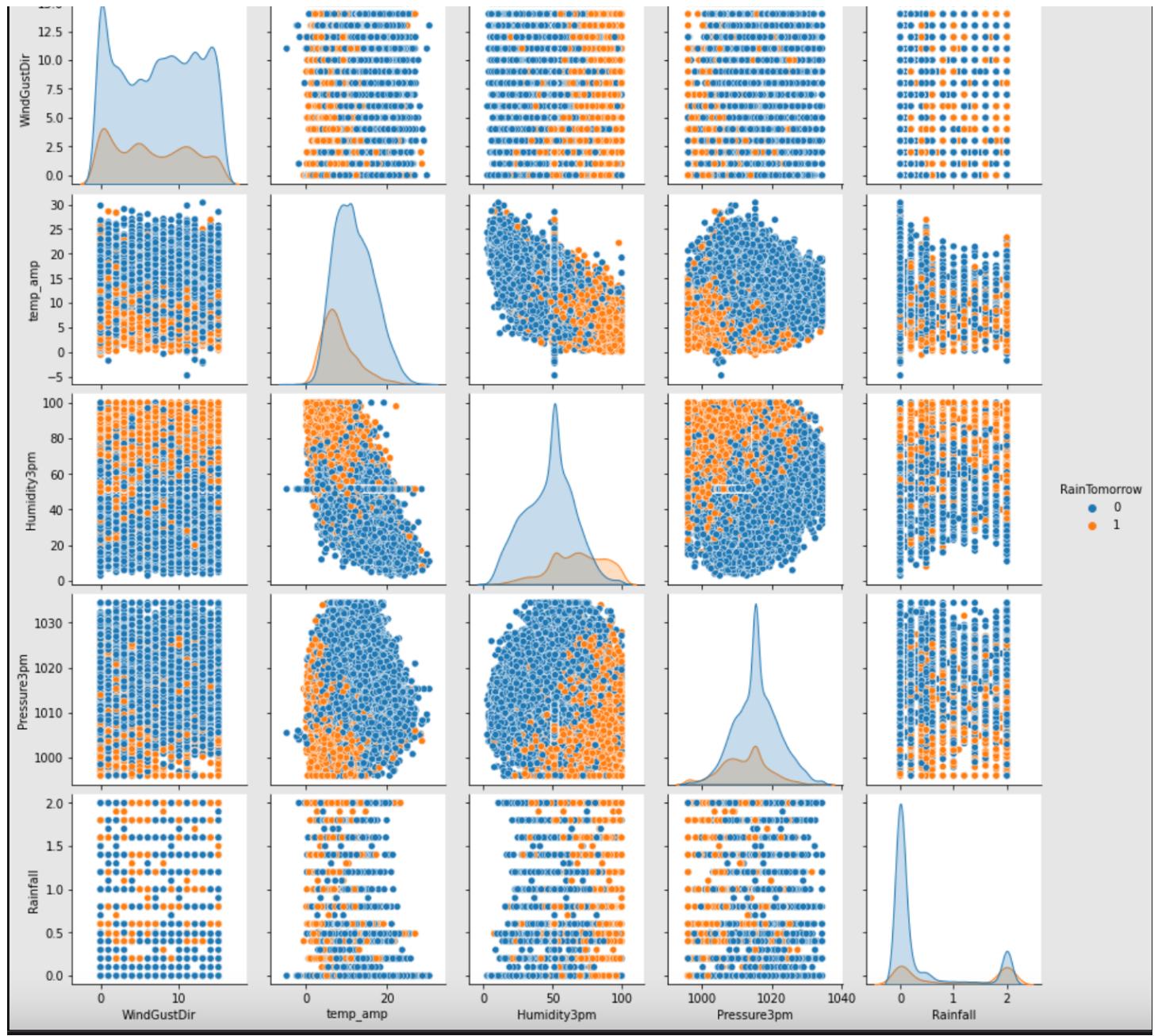
A l'issu d'un undersampling par centroid cluster, allégeons le dataset et filtrons les dates pour ne garder que quelques années et réexécutons un pairplot avec en couleur la variable cible:

```
#Centroids
cc = ClusterCentroids()
X_cc, y_cc = cc.fit_resample(X_train, y_train)
print('Classes échantillon CC :', dict(pd.Series(y_cc).value_counts()))

Classes échantillon undersampled : {0: 1426, 1: 1426}
Classes échantillon CC : {0: 1426, 1: 1426}
```

Réduisons aussi le nombre de colonnes en ne gardant que celles qui ont de l'importance d'après le premier pairplot et intuitivement.

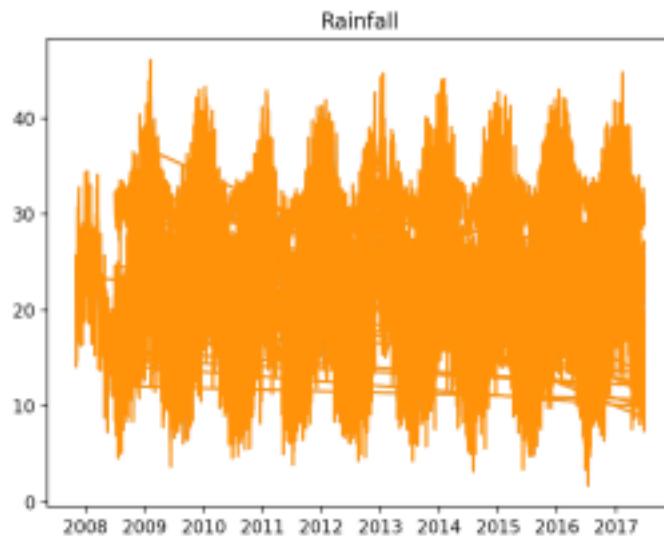
L'objectif étant de visualiser des relations entre la cible et les autres variables.



On remarque des relations linéaires entre la variable cible “RainTomorrow” et

les autres variables

Pour les données temporelles, nous pouvons tracer les précipitations en fonction du temps. On remarque une périodicité que nous pourrons exploiter avec le modèle pour les timeseries



### 3. Modélisation

#### Prédictions à court terme

Tout d'abord effectuons une standardisation des données

```
target = 'RainTomorrow'
y = df[target]
X = df.drop([target],axis=1)
print('logreg')
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2, random_state = 0)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

y.value_counts(normalize=True)
```

```
logreg
0      0.780854
1      0.219146
Name: RainTomorrow, dtype: float64
```

La surreprésentation de la classe 1 est ici confirmée

Effectuons tout de même une calibration avec une régression logistique:

```
classifier = LogisticRegression(solver='liblinear', random_state=0)
classifier.fit(X_train, y_train)
end_time = time.time()
y_pred = classifier.predict(X_test)
acc_score = accuracy_score(y_test,y_pred)
classifier_score_train = classifier.score(X_train, y_train)
classifier_score_test = classifier.score(X_test, y_test)
cm = confusion_matrix(y_test, y_pred)
print(classifier_score_train)
print(classifier_score_test)
print(cm)
sns.heatmap(cm)
```

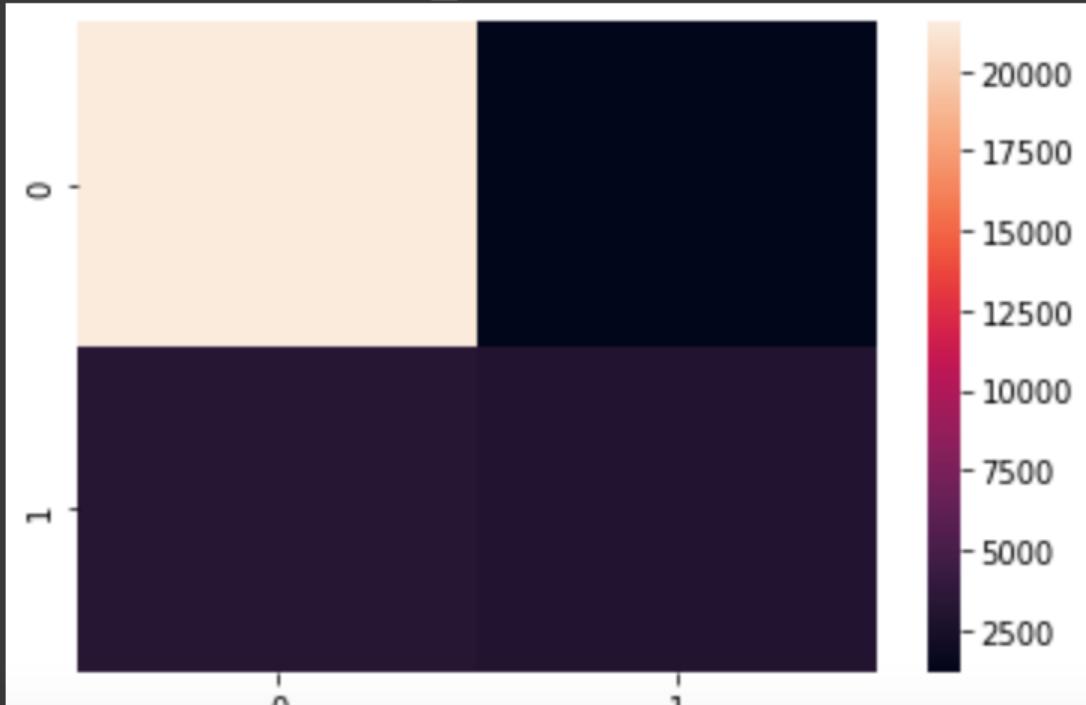
0.8437714835693662

0.8444933315000688

[[21536 1190]

[ 3334 3032]]

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa9b2c2a250



```
print(classification_report_imbalanced(y_test, y_pred))
```

	pre	rec	spe	f1	geo	iba	sup
0	0.87	0.95	0.48	0.90	0.67	0.47	22726
1	0.72	0.48	0.95	0.57	0.67	0.43	6366
avg / total	0.83	0.84	0.58	0.83	0.67	0.46	29092

Au vu des résultats, notamment le ‘recall’ et le f1 score, on ne peut pas garder le modèle tel quel.

Effectuons différent resampling et évaluons les résultats

### Oversampling et SMOTE:

```
rOs = RandomOverSampler()
X_ro, y_ro = rOs.fit_resample(X_train, y_train)
print('Classes échantillon oversampled :', dict(pd.Series(y_ro).value_counts()))
```

```
smo = SMOTE()
X_sm, y_sm = smo.fit_resample(X_train, y_train)
print('Classes échantillon SMOTE :', dict(pd.Series(y_sm).value_counts()))
```

```
Classes échantillon oversampled : {1: 90857, 0: 90857}
Classes échantillon SMOTE : {1: 90857, 0: 90857}
```

```
svm = SVC(gamma='scale')
svm.fit(X_ro, y_ro)

y_pred = svm.predict(X_test)
print(pd.crosstab(y_test, y_pred))

print(classification_report_imbalanced(y_test, y_pred))
```

col_0	0	1						
RainTomorrow								
0	18619	4107						
1	1439	4927						
	pre	rec	spe	f1	geo	iba	sup	
0	0.93	0.82	0.77	0.87	0.80	0.64	22726	
1	0.55	0.77	0.82	0.64	0.80	0.63	6366	
avg / total	0.84	0.81	0.78	0.82	0.80	0.64	29092	

## Undersampling et Cluster Centroïds

```

rUs = RandomUnderSampler()
X_ru, y_ru = rUs.fit_resample(X_train, y_train)
print('Classes échantillon undersampled :', dict(pd.Series(y_ru).value_counts()))

```

```

#Centroids
cc = ClusterCentroids()
X_cc, y_cc = cc.fit_resample(X_train, y_train)
print('Classes échantillon CC :', dict(pd.Series(y_cc).value_counts()))

```

```

Classes échantillon undersampled : {0: 1426, 1: 1426}
Classes échantillon CC : {0: 1426, 1: 1426}

```

```

svm = SVC(gamma='scale')
svm.fit(X_ru, y_ru)

y_pred = svm.predict(X_test)
print(pd.crosstab(y_test, y_pred))
print(classification_report_imbalanced(y_test, y_pred))

```

col_0	0	1						
RainTomorrow								
0	18355	4371						
1	1387	4979						
	pre	rec	spe	f1	geo	iba	sup	
0	0.93	0.81	0.78	0.86	0.79	0.63	22726	
1	0.53	0.78	0.81	0.63	0.79	0.63	6366	
avg / total	0.84	0.80	0.79	0.81	0.79	0.63	29092	

Les données sont équilibrées mais il n'y a qu'une légère amélioration (recall et f1 score)

Avec un GradientBoosting sur les données over samplées,  
On obtient de meilleurs résultats

```
classifier = GradientBoostingClassifier()
classifier.fit(X_sm, y_sm)
end_time = time.time()
|
y_pred = classifier.predict(X_test)
acc_score = accuracy_score(y_test,y_pred)
classifier_score_train = classifier.score(X_train, y_train)
classifier_score_test = classifier.score(X_test, y_test)
cm = confusion_matrix(y_test, y_pred)
print(classifier_score_train)
print(classifier_score_test)
print(cm)
sns.heatmap(cm)
```

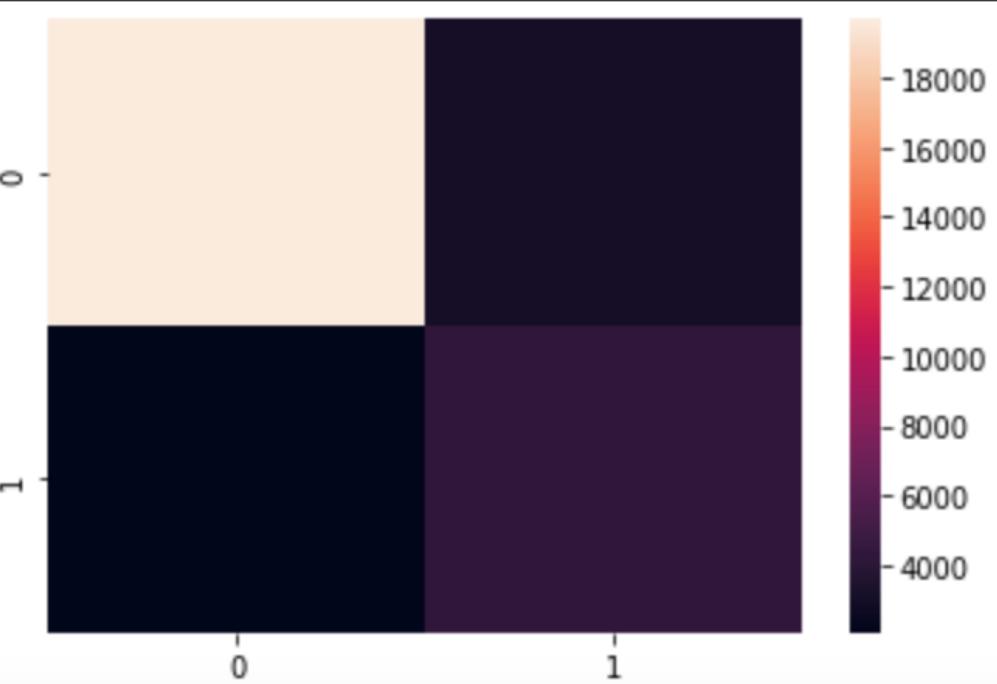
0.8227347724460333

0.8238347311975801

[[19668 3058]

[ 2067 4299]]

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fbf368bce50>



## 2. Analyse des Séries Temporelles

---

Nous évaluerons le modèle NeuralProphet (Facebook)

Considérons la ville de Sydney  
et gardons les valeurs temporelles et les cibles:  
Les précipitation (RainFall) puis les températures  
maximales (MaxTemp)

Et calibrons le model sur une périodes journalière

Le modèle NeuralProphet de facebook combine des méthodes classique d'analyse de séries temporelles (Fourier, arima, ..) et avec des layers de neurones pour estimer les coefficients d'autorégression.

En ce qui concerne les coefficients d'autorégression,  
nous pouvons déjà visualiser les différentiations d'ordre  
2 des transformées logarithmiques

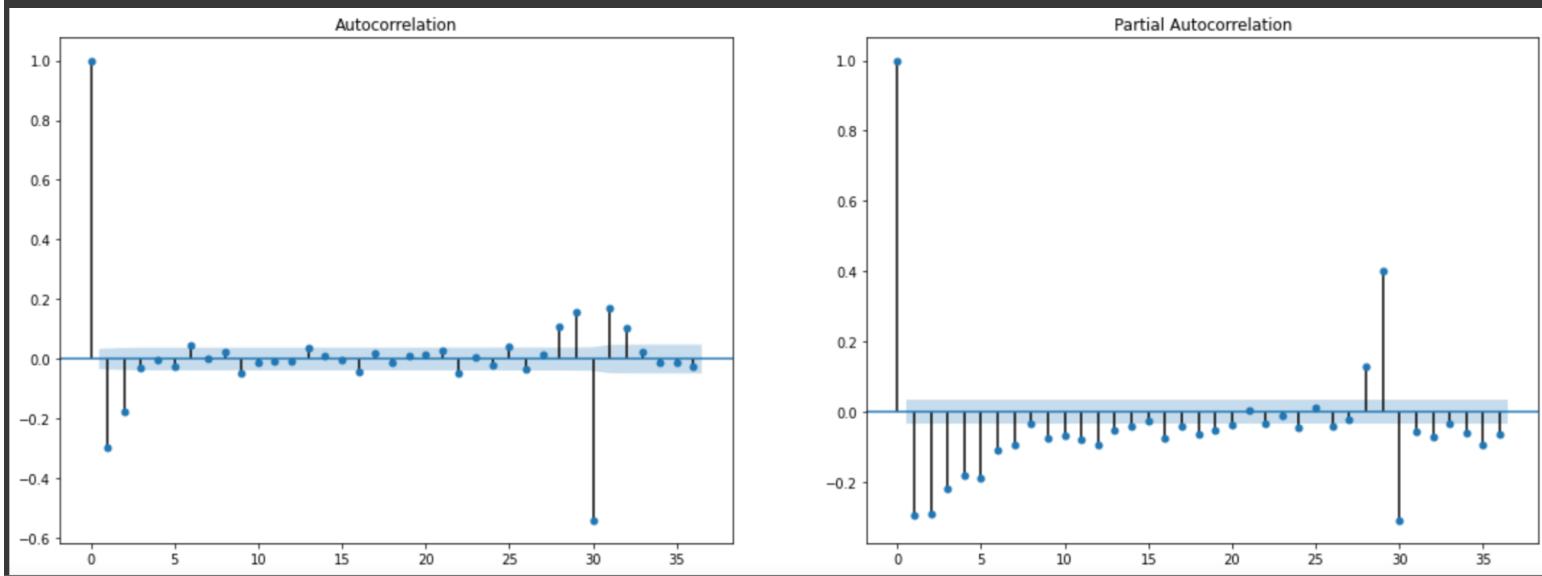
(ci dessous pour la variable “Rainfall”

```
rainfall_log = np.log(dfts['MaxTemp']) #Transformée logarithmique

rainfall_log_1 = rainfall_log.diff().dropna() #Differenciation simple

rainfall_log_2 = rainfall_log_1.diff(periods = 30).dropna() #Différenciation d'ordre 12
```

```
from statsmodels.graphics.tsplots import plot_acf, plot_pacf
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20,7))
plot_acf(rainfall_log_2, lags = 36, ax=ax1)
plot_pacf(rainfall_log_2, lags = 36, ax=ax2)
plt.show()
```



```
df = pd.read_csv('./weatherAUS.csv')

df.Location.unique()
df.columns
df['Date'] = pd.to_datetime(df['Date'])
plt.plot(df['Date'], df['Rainfall'])
df['Year'] = df['Date'].apply(lambda x: x.year)
data = df

df['Date'].dtype
df['year'] = df['Date'].dt.year
df['month'] = df['Date'].dt.month
df['day'] = df['Date'].dt.day

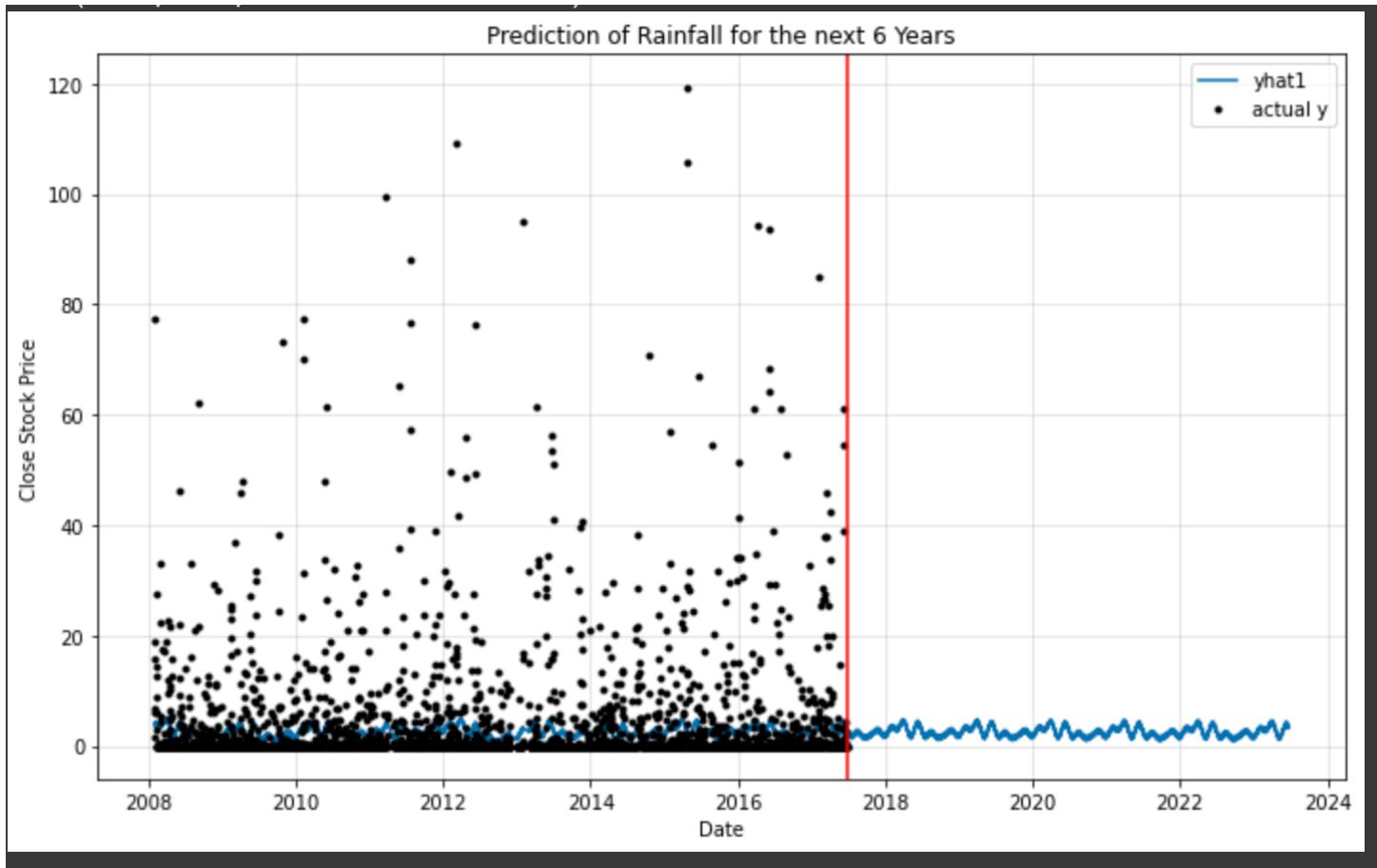
print(df.Location.unique())
df.columns
d=df
d = data[data['Location'] == 'Sydney']
# d = d[d['Year']<=2013]
d = d[['Date', 'Rainfall']]
d.dropna(inplace=True)
# data = data.drop_duplicates()

d.columns = ['ds', 'y']
print(d.shape)

mod = NeuralProphet()
model = mod.fit(d, freq='D')#, epochs=1000)
```

Effectuons maintenant une prédiction sur 6 ans (6\*365 jours)

Pour les précipitations



Pour les températures maximales:

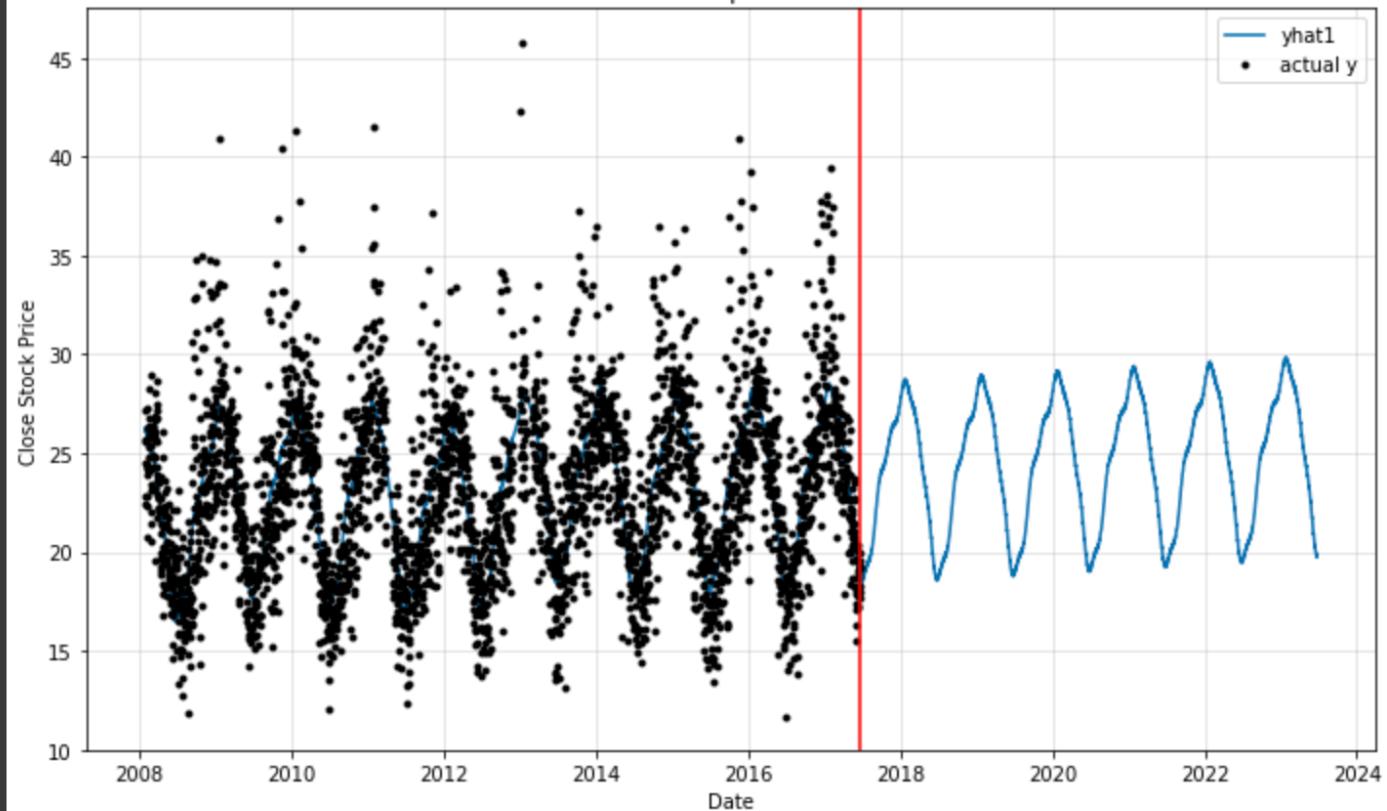
```
future = mod.make_future_dataframe(d, periods=6*365, n_historic_predictions=True)

prediction = mod.predict(future)
forecast = mod.plot(prediction)
# plt.plot(df_test['ds'], df_test['y'], color='purple')
plt.title("Prediction of MaxTemp for the next 6 Years")
plt.xlabel("Date")
plt.axvline(x=d['ds'].iloc[-1], color='r')

plt.ylabel("Close Stock Price")
```

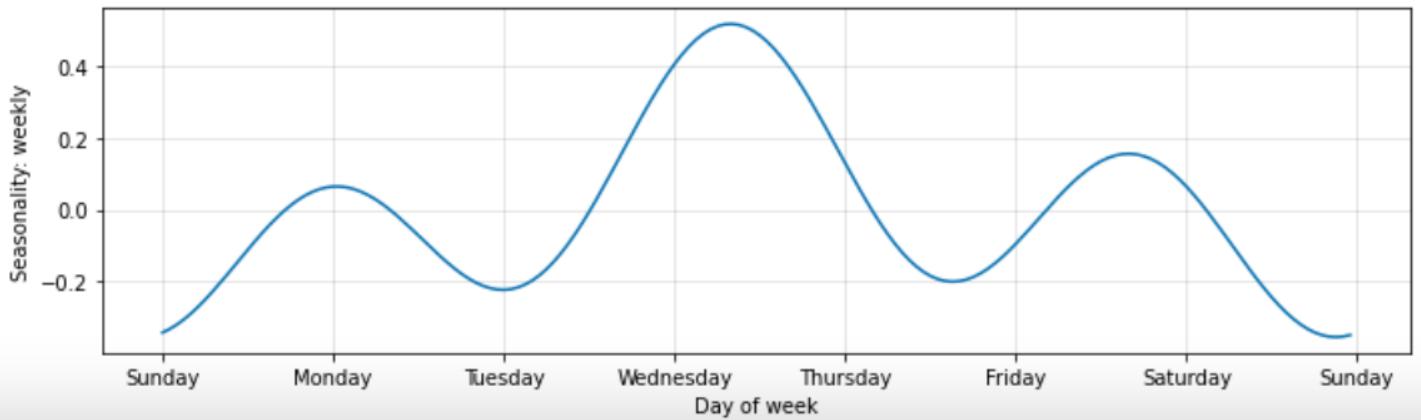
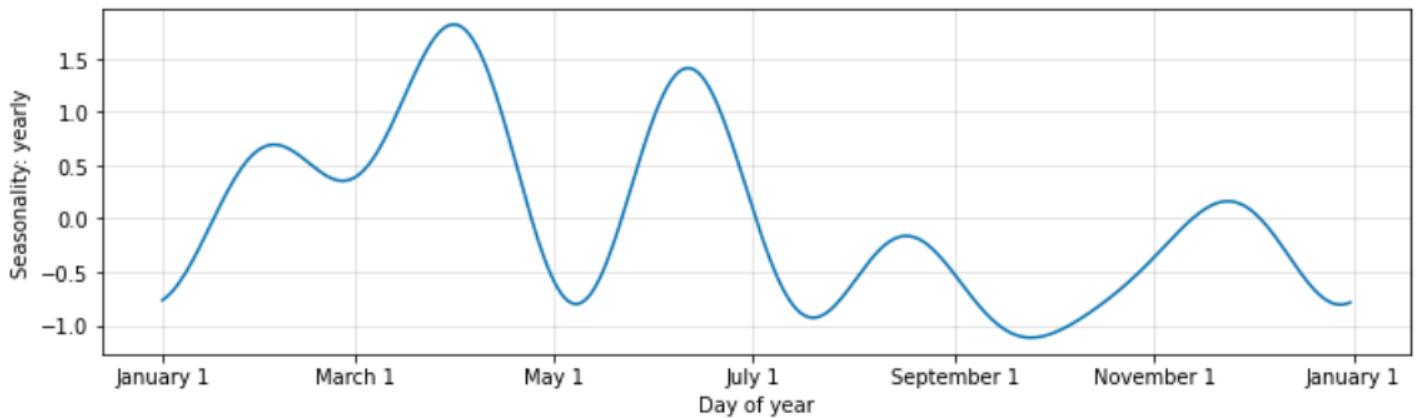
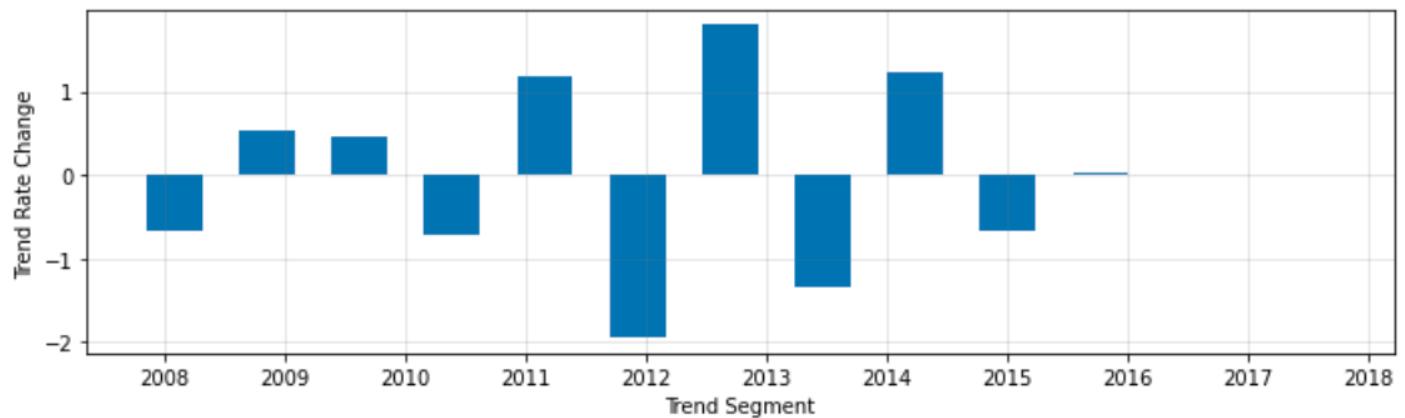
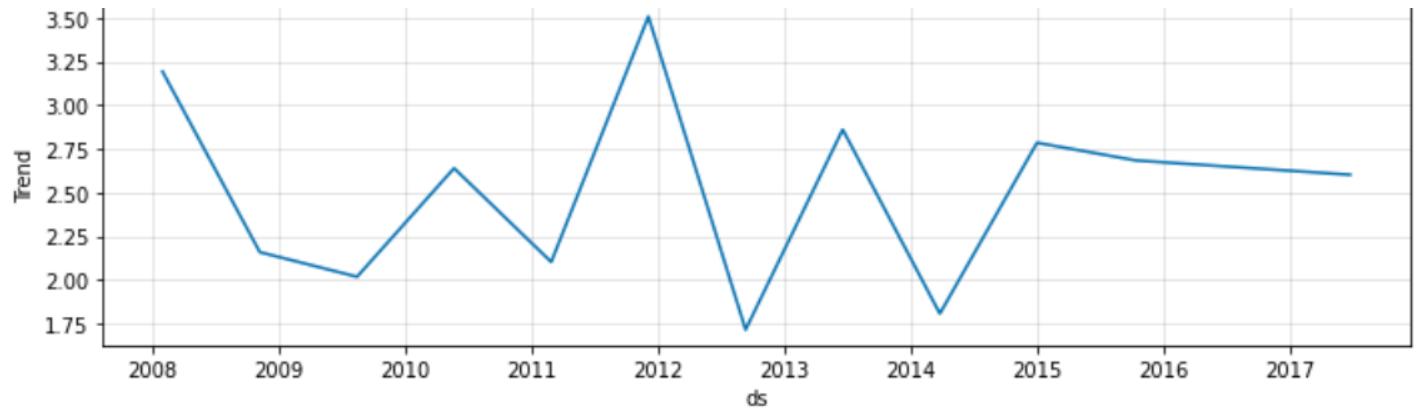
```
text(0.25, 0.5, "Close Stock Price")
```

Prediction of MaxTemp for the next 6 Years



De plus la visualisation des composantes du modèle nous permet d'afficher les tendances et saisonnalités (ci-dessous pour “Rainfall”)

Notamment on remarque qu'il pleut davantage entre janvier et juillet



## Conclusion

Pour les prédictions à court terme, malgré un déséquilibre dans la distribution de la variable cible, nous avons pu obtenir des résultats corrects. L'Australie étant un pays plutôt sec.

Pour les prévisions long terme, le modèle NeuralProphet permet de bien capter les saisonnalités

La combinaison des méthodes à court et long termes avec un meilleurs fine tuning des modèles et leur entraînement dans d'autre régions du monde pourraient donner de meilleurs résultats.

