TRAVAIL PRATIQUE DE SERIE TEMPORELLE

Réalisé par :

- *MEKA Moïse Christian Junior (21T2561)*
- *Douanla Sonhafo Champlain (21T2657)*
- *CHAMENI LEUDJIEU MIDREL STIVE (21T2372)*
- *NANA ORNELLA (21T2650)*

1- INITIALISATION

Pour la réalisation de ce travail pratique, nous allons commencer par importer les packages suivant pour :

- Realiser les opérations matricielles (*numpy*)
- Importer et traiter les données des fichiers (*pandas*)
- Réaliser des graphiques (*matplotlib*)
- Visualiser les erreurs dans des tableaux (*tabulate* et *termcolor*)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tabulate import tabulate
from termcolor import colored
```

Dans le cadre de ce TP nous avons utilisé les cinq (5) jeux de données suivants :

- AirPassengers.csv : Qui contient l'évolution mensuelle du nombre de passagers d'une compagnie aerienne
- BTC-EUR.csv : Qui contient l'évolution jounalière du volume de bitcoin
- Month_Value_1.csv : Qui contient l'évolution mensuelle des ventes d'un supermarché
- SARS-Cov2-PCRdata.csv : Qui contient l'évolution par semaine épidémiologique du taux de positivité au diagnostic du SARS-Cov2
- annual_rainfall_dallas.csv : Qui contient l'évolution annuelle de la pluviométrie d'une région

In [2]:
```
headers=["Type de lissage", "DataSet 1", "DataSet 2", "DataSet 3", "DataSet 4", "DataSet 5"]
Files_List=[ 'AirPassengers.csv', 'BTC-EUR.csv', 'Month_Value_1.csv',  'SARS-Cov2-PCRdata.csv', 'annual_rainfall_da
Features_List=['#Passengers','Volume', 'Revenue', 'Tx', 'Total']
SEP=[',',',',',',',',';',',',']
NB_DF=5
HORIZON=2
all_alpha = [0.1, 0.5, 0.9]
all_results={}
prop_data=0.9
```

In [3]:
```
all_df={}
taille_list={}
for i in range(NB_DF):
    df=pd.read_csv(f"Datasets/{Files_List[i]}", sep=SEP[i])
    df=df.dropna()
    all_df[f"{headers[i+1]}"]=df
    taille_list[f"{headers[i+1]}"]=int(len(df)*prop_data)
```

2- LISSAGE SIMPLE

Pour commencer nous implémentons le lissage simple en créant une classe qui va permettre de générer le lissage simple d'une serie

In [4]:
```
class Lissage_Simple:
    def __init__(self):
        self.L1=[]

    def __str__(self):
        return f"{self.L1}"
```

```
    def predict_simple_expo_lissage(self, data, alpha, taille):
        for i in range(taille):
            if i==0:
                self.L1.append((1-alpha)*data[i])
            else:
                tmp=(1-alpha)*data[i]+alpha*self.L1[i-1]
                self.L1.append(tmp)
        return self

    def predict(self, length):
        if len(self.L1)>0:
            last=self.L1[-1]
            for i in range(length):
                self.L1.append(last)
        return self
```

On effectue le lissage simple de chacune de nos cinq séries

In [5]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    all_L1={}
    for alpha in all_alpha:
        L1=Lissage_Simple()
        all_L1[f"{alpha}"]=L1.predict_simple_expo_lissage(df[FEATURE], alpha, taille_list[key])
    all_results[key]={"Lissage_Simple":all_L1}
    i+=1
```

3- LISSAGE DOUBLE

Dans ce bloc nous implémentons le lissage double en créant une classe qui va permettre de générer le lissage double d'une serie

In [6]:
```
class Lissage_Double:
    def __init__(self):
        self.x_hat=[]
        self.a1=[]
```

```
        self.a2=[]

    def predict_lissage_double(self, data, alpha, length, horizon):
        for i in range(length):
            if i==0:
                self.a1.append(data[i])
                self.a2.append(data[i+1]-data[i])
            else :
                self.a1.append((alpha**2)*(self.a1[-1]+self.a2[-1])+(1-alpha**2)*(data[i]))
                self.a2.append(((1-alpha)**2)*(data[i])-((1-alpha)**2)*self.a1[-2]+(2*alpha-(alpha**2))*self.a2[-1]
            self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon)
        return self

    def predict(self, length):
        #Ici on va tracer la droite Y(x)=a1+a2*x
        a1=self.a1[-1]
        a2=self.a2[-1]
        for x in range(length):
            y=a1+a2*x
            self.x_hat.append(y)
        return self
```

In [7]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    all_LissDouble={}
    for alpha in all_alpha:
        x_hat=Lissage_Double()
        all_LissDouble[f"{alpha}"]=x_hat.predict_lissage_double(df[FEATURE], alpha, taille_list[key], HORIZON)
    all_results[key]["Lissage_Double"]=all_LissDouble
    i+=1
```

4- HOLT-WINTERS

4.1- SANS SAISONNALITE

In [8]:
```
class Lissage_HW_SS:
    def __init__(self):
        self.x_hat=[]
        self.a1=[]
        self.a2=[]

    def predict_lissage_hw_ss(self, data, beta, gamma, length, horizon):
        for i in range(length):
            if i==0:
                self.a1.append(data[i])
                self.a2.append(data[i+1]-data[i])
            else :
                self.a1.append(beta*data[i]+(1-beta)*(self.a1[-1]+self.a2[-1]))
                self.a2.append(gamma*(self.a1[-1]-self.a1[-2])+(1-gamma)*self.a2[-1])
            self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon)
        return self

    def predict(self, length):
        #Ici on va tracer la droite Y(x)=a1+a2*x
        a1=self.a1[-1]
        a2=self.a2[-1]
        for x in range(length):
            y=a1+a2*x
            self.x_hat.append(y)
        return self
```

In [9]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    holt_winters_ss={}
    for beta in all_alpha:
        for gamma in all_alpha:
            key1=f"{beta} {gamma}"
            hw_ss=Lissage_HW_SS()
            holt_winters_ss[key1]=hw_ss.predict_lissage_hw_ss(df[FEATURE], beta, gamma, taille_list[key], HORIZON)
    all_results[key]["HW_SS"]=holt_winters_ss
    i+=1
```

4.2- AVEC SAISONNALITE

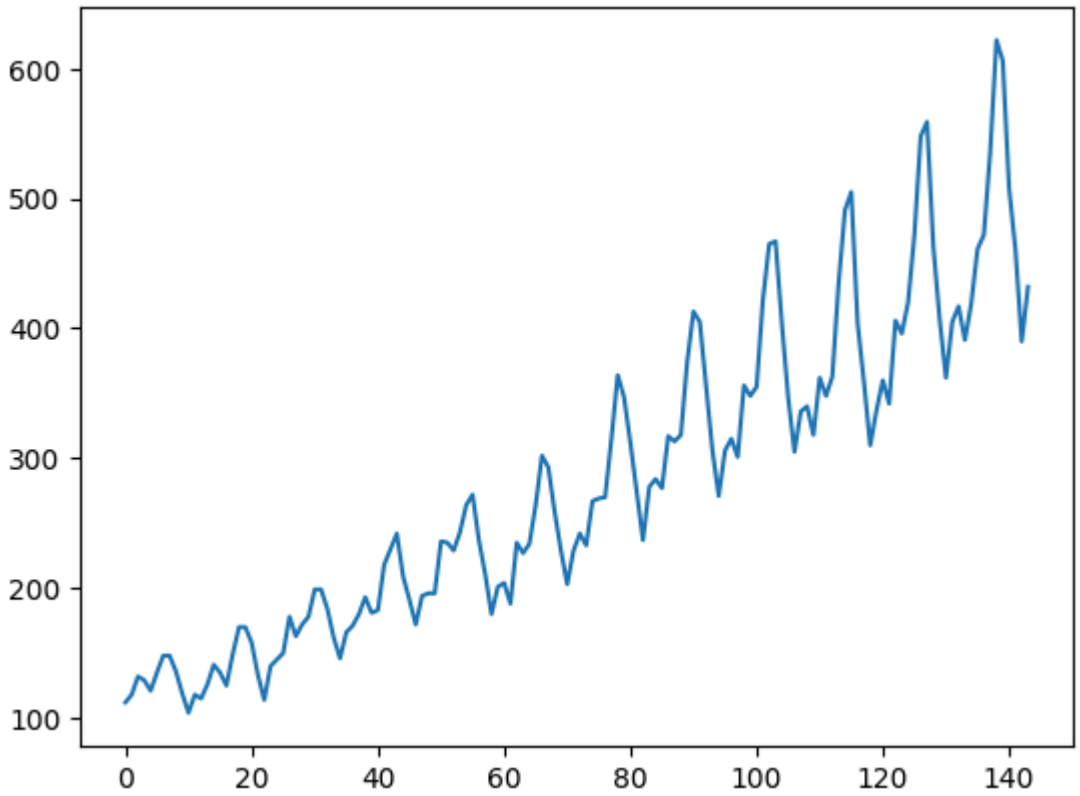
4.2.1- VERIFICATION DE LA SAISONNALITE

Afin de pouvoir réaliser les lissage de holt winters avec saisonnalité, nous avons d'abord voir si notre série est saisonnière et ensuite identifier la période.
NB : Vu que le TP demande de réaliser toutes les méthodes de lissage à tous les jeux de données nous prendrons T=2 pour les séries non saisonnière

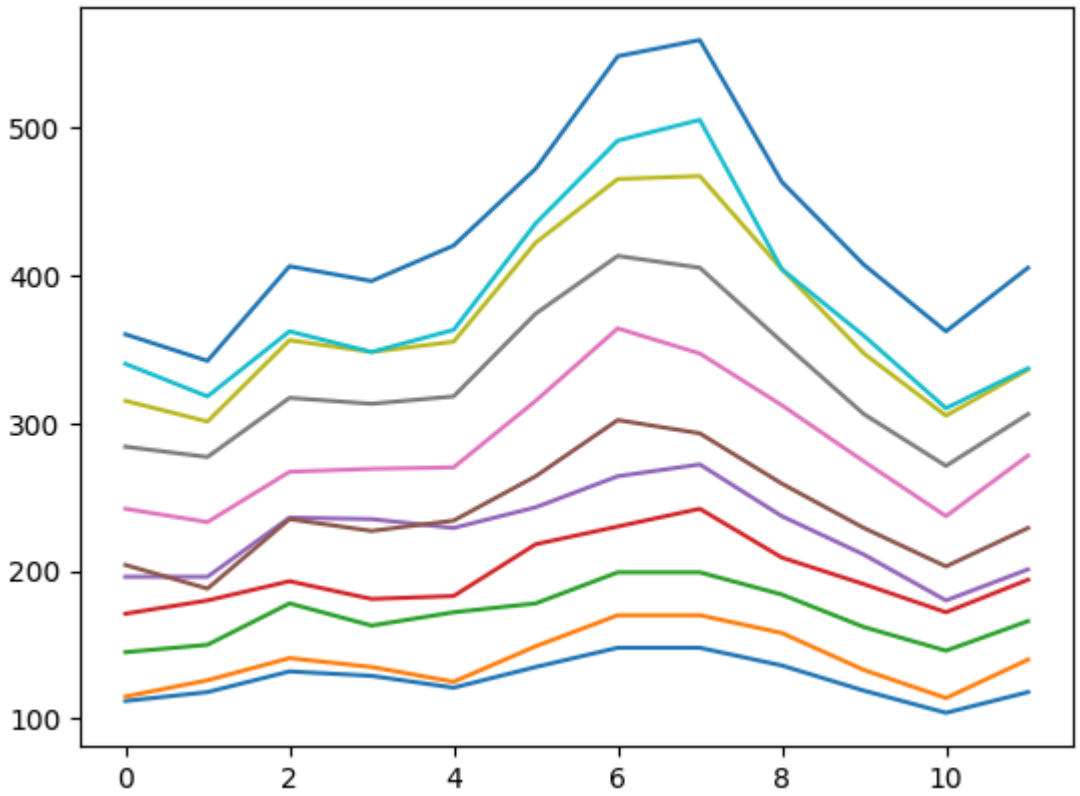
In [10]:
```
def season_check(data,periode):
    for i in range(periode,len(data),periode):
        x=np.linspace(0,periode-1,periode)
        if i==0:
            plt.plot(x,data[:periode])
        else:
            plt.plot(x,data[i-periode:i])
```

In [11]:
```
i=0
plt.plot(all_df[f"DataSet {i+1}"][Features_List[i]])
```

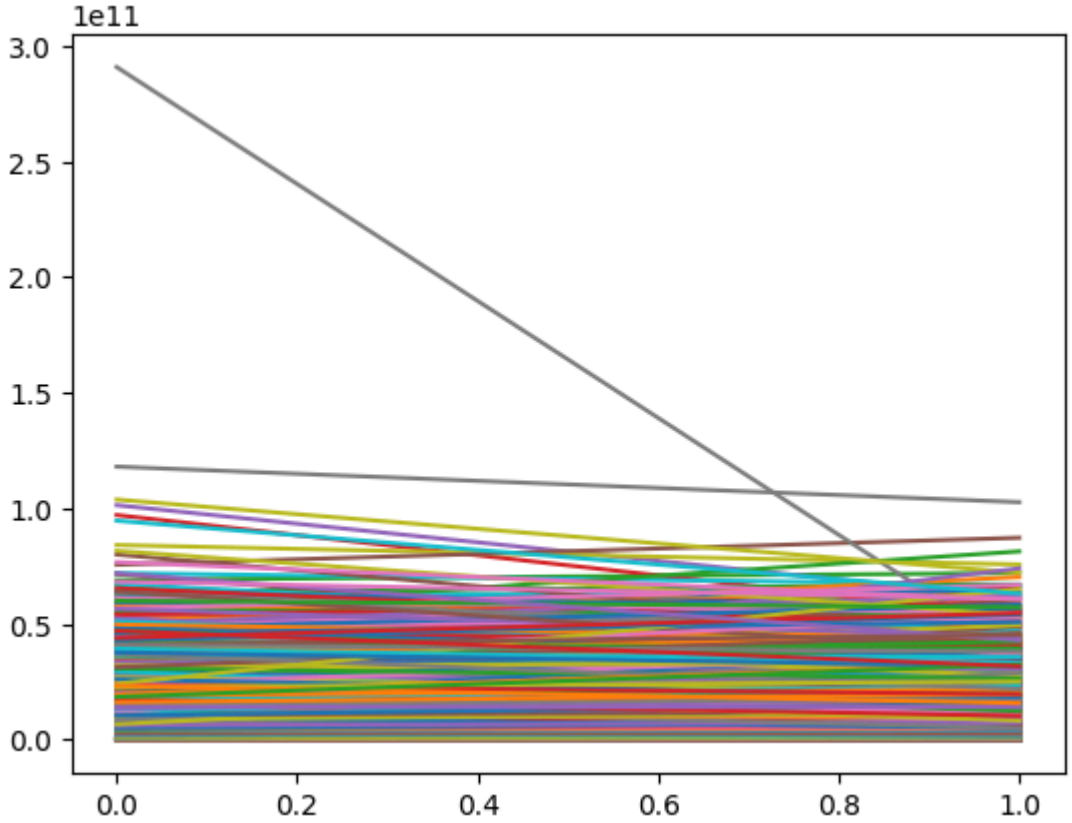
Out[11]:
```
[<matplotlib.lines.Line2D at 0x11afec560>]
```



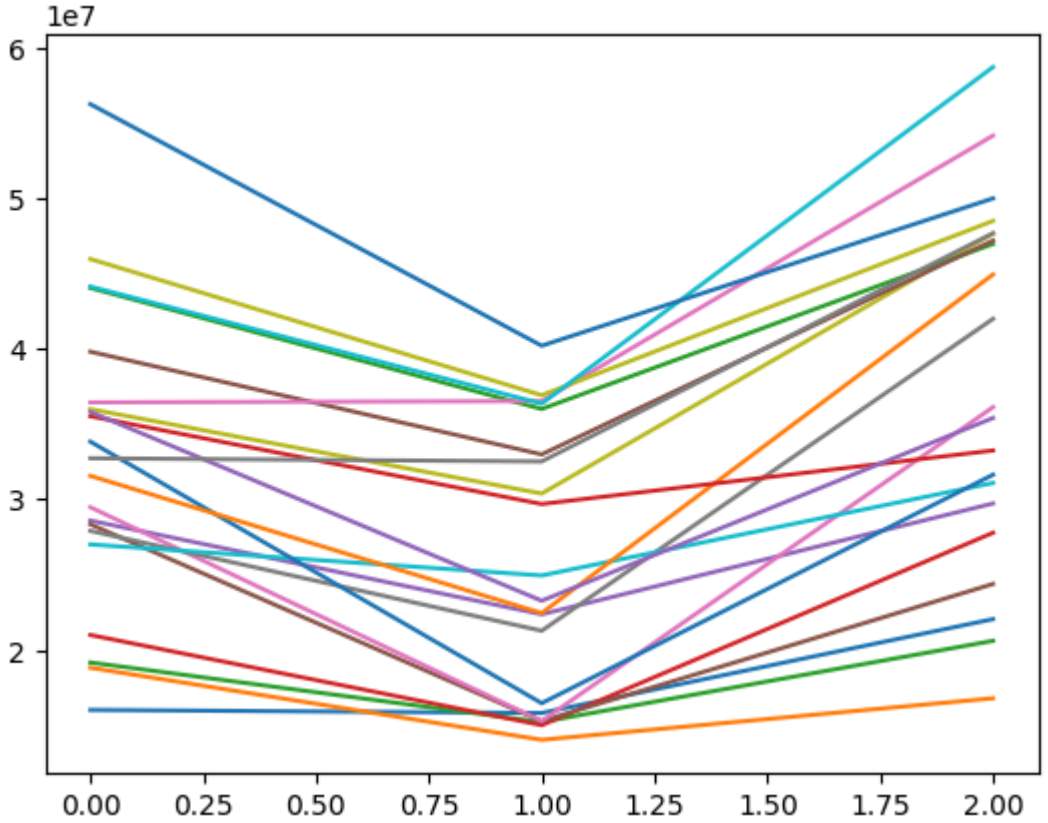
In [12]:
```
i=0
season_check(all_df[f"DataSet {i+1}"][Features_List[i]],12)
```



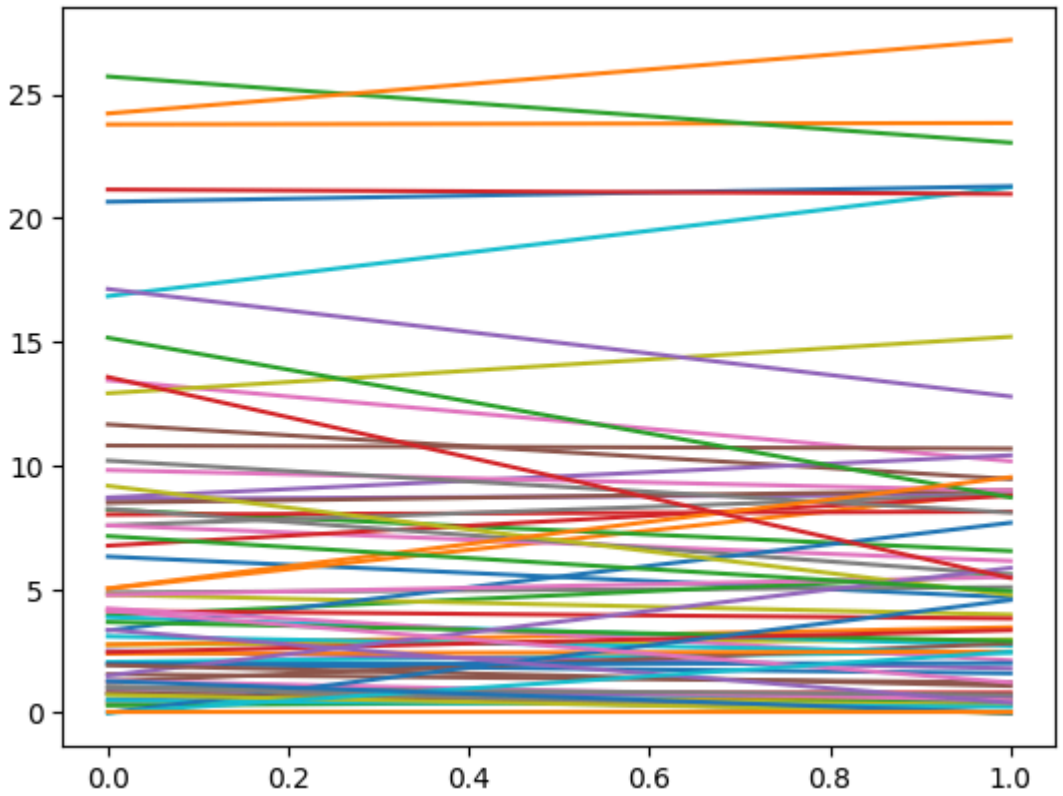
In [13]:
```
i=1
season_check(all_df[f"DataSet {i+1}"][Features_List[i]],2)
```



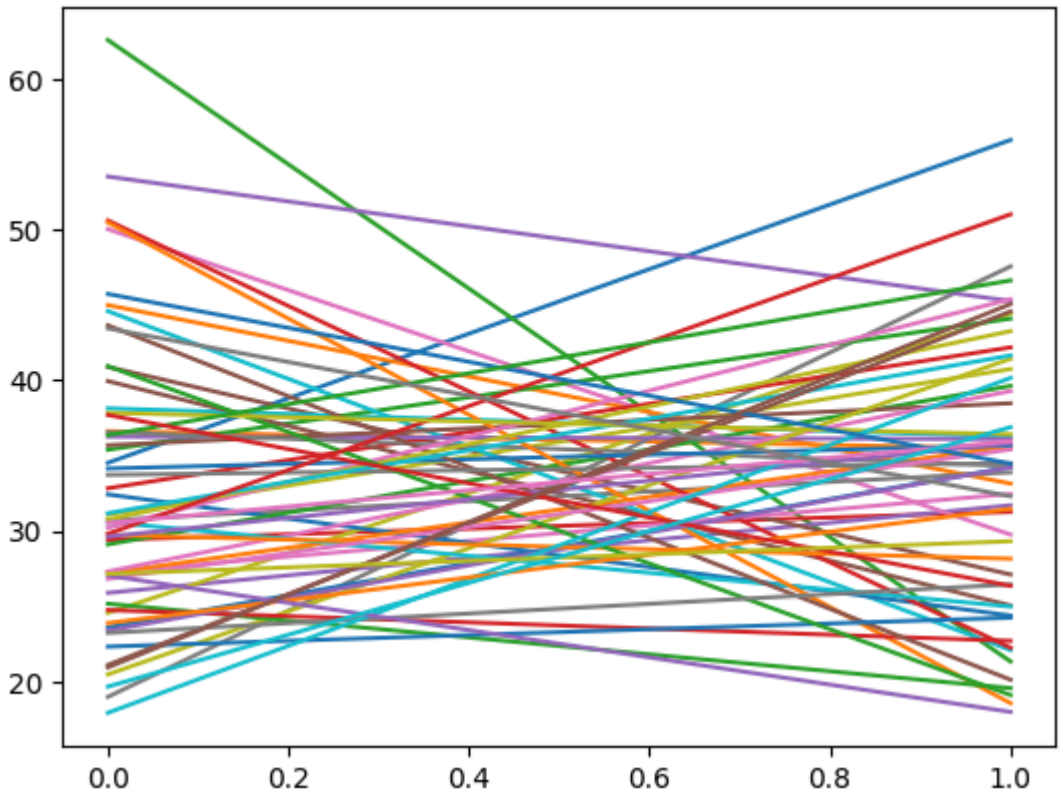
In [14]:
```
i=2
season_check(all_df[f"DataSet {i+1}"][Features_List[i]],3)
```



In [15]:
```
i=3
season_check(all_df[f"DataSet {i+1}"][Features_List[i]],2)
```



In [16]:
```
i=4
season_check(all_df[f"DataSet {i+1}"][Features_List[i]],2)
```



Après avoir vérifié visuellement la saisonnalité, voici les périodes obtenues

In [17]:
```
periode=[12,2,3,2,2]
```

4.2.2- SAISONNALITE ADDITIVE

Implémentation de Holt-Winters avec saisonnalité additive

In [18]:
```
class Lissage_HW_SA:
    def __init__(self, periode):
        self.x_hat=[]
        self.a1=[]
        self.a2=[]
        self.st=[]
```

```
        self.T=periode

    def initialize(self, data, horizon):
        self.a1.append(np.mean(data[:self.T]))
        tmp1=np.array(data[:self.T])
        tmp2=np.array(data[self.T:2*self.T])
        self.a2.append(np.mean((tmp2-tmp1)/self.T))
        for i in range(self.T):
            self.st.append(data[i]-self.a1[0])
        self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[0])

    def predict_lissage_hw_sa(self, data, beta, gamma, omega,length, horizon):
        self.initialize(data, horizon)
        for i in range(1, length):
            if i < self.T:
                self.a1.append(beta*(data[i]-self.st[i])+(1-beta)*(self.a1[-1]+self.a2[-1]))
                self.a2.append(gamma*(self.a1[-1]-self.a1[-2])+(1-gamma)*self.a2[-1])
                if horizon < self.T:
                    self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[i+horizon-self.T])
                else :
                    self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[i+horizon-2*self.T])
            else:
                self.a1.append(beta*(data[i]-self.st[i-self.T])+(1-beta)*(self.a1[-1]+self.a2[-1]))
                self.a2.append(gamma*(self.a1[-1]-self.a1[-2])+(1-gamma)*self.a2[-1])
                self.st.append(omega*(data[i]-self.a1[-1])+(1-omega)*self.st[i-self.T])
                if horizon < self.T:
                    self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[i+horizon-self.T])
                else :
                    self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[i+horizon-2*self.T])
        return self

    def predict(self, length):
        #Ici on va tracer la droite Y(x)=a1+a2*x
        a1=self.a1[-1]
        a2=self.a2[-1]
        for x in range(length-1):
            y=a1+a2*x+self.st[x-self.T]
            self.x_hat.append(y)
        return self
```

In [19]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    holt_winters_sa={}
    for beta in all_alpha:
        for gamma in all_alpha:
            for omega in all_alpha:
                key1=f"{beta} {gamma} {omega}"
                hw_sa=Lissage_HW_SA(periode[i])
                taille=int()
                holt_winters_sa[key1]=hw_sa.predict_lissage_hw_sa(df[FEATURE], beta, gamma, omega, taille_list[key]
    all_results[key]["HW_SA"]=holt_winters_sa
    i+=1
```

4.2.3- SAISONNALITE MULTIPLICATIVE

Implémentation de Holt-Winters avec saisonnalité multiplicatiVE

In [20]:
```
class Lissage_HW_SM:
    def __init__(self, periode):
        self.x_hat=[]
        self.a1=[]
        self.a2=[]
        self.st=[]
        self.T=periode

    def initialize(self, data, horizon):
        self.a1.append(np.mean(data[:self.T]))
        tmp1=np.array(data[:self.T])
        tmp2=np.array(data[self.T:2*self.T])
        self.a2.append(np.mean((tmp2-tmp1)/self.T))
        for i in range(self.T):
            self.st.append(data[i]/self.a1[0])
        self.x_hat.append((self.a1[0]+self.a2[0]*horizon)*self.st[0])

    def predict_lissage_hw_sm(self, data, beta, gamma, omega,length, horizon):
        self.initialize(data, horizon)
        for i in range(1, length):
```

```
            if i < self.T:
                self.a1.append(beta*(data[i]/self.st[i])+(1-beta)*(self.a1[-1]+self.a2[-1]))
                self.a2.append(gamma*(self.a1[-1]-self.a1[-2])+(1-gamma)*self.a2[-1])
                if horizon < self.T:
                    self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[i+horizon-self.T])
                else :
                    self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[i+horizon-2*self.T])
            else:
                self.a1.append(beta*(data[i]/self.st[i-self.T])+(1-beta)*(self.a1[-1]+self.a2[-1]))
                self.a2.append(gamma*(self.a1[-1]-self.a1[-2])+(1-gamma)*self.a2[-1])
                self.st.append(omega*(data[i]/self.a1[-1])+(1-omega)*self.st[i-self.T])
                if horizon < self.T:
                    self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[i+horizon-self.T])
                else :
                    self.x_hat.append(self.a1[-1]+self.a2[-1]*horizon+self.st[i+horizon-2*self.T])
        return self

    def predict(self, length):
        #Ici on va tracer la droite Y(x)=a1+a2*x
        a1=self.a1[-1]
        a2=self.a2[-1]
        for x in range(length-1):
            y=(a1+a2*x)*self.st[x-self.T]
            self.x_hat.append(y)
        return self
```

In [21]:

```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    holt_winters_sm={}
    for beta in all_alpha:
        for gamma in all_alpha:
            for omega in all_alpha:
                key1=f"{beta} {gamma} {omega}"
                hw_sm=Lissage_HW_SM(periode[i])
                holt_winters_sm[key1]=hw_sm.predict_lissage_hw_sm(df[FEATURE], beta, gamma, omega, taille_list[key]
    all_results[key]["HW_SM"]=holt_winters_sm
    i+=1
```

5- EVALUATION DES MODELES DE LISSAGE

Pour évaluer les modèles construient plus haut nous avons utilisé la somme des carrés des erreurs

In [22]:
```
def sum_square_error(real, predic):
    result=real-predic
    result=result**2
    return np.sum(result)
```

In [23]:
```
all_errors={}
```

5.1- LISSAGE SIMPLE

In [24]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    all_error_L1={}
    all_L1=all_results[key]["Lissage_Simple"]
    for alpha in all_alpha:
        real=np.array(df[FEATURE])
        predic=np.array(all_L1[f"{alpha}"].L1)
        all_error_L1[f"{alpha}"]=sum_square_error(real[:taille_list[key]], predic)
    all_errors[key]={"Lissage_Simple":all_error_L1}
    i+=1
```

5.2- LISSAGE DOUBLE

In [25]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    all_LissDouble=all_results[key]['Lissage_Double']
    all_error_LissDouble={}
    for alpha in all_alpha:
        real=np.array(df[FEATURE])
```

```
        predic=np.array(all_LissDouble[f"{alpha}"].x_hat)
        all_error_LissDouble[f"{alpha}"]=sum_square_error(real[:taille_list[key]], predic)
    all_errors[key]["Lissage_Double"]=all_error_LissDouble
    i+=1
```

5.3- HW SANS SAISONNALITE

In [26]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    holt_error_winters_ss={}
    holt_winters_ss=all_results[key]["HW_SS"]
    for beta in all_alpha:
        for gamma in all_alpha:
            real=np.array(df[FEATURE])
            predic=np.array(holt_winters_ss[f"{beta} {gamma}"].x_hat)
            holt_error_winters_ss[f"{beta} {gamma}"]=sum_square_error(real[:taille_list[key]], predic)
    all_errors[key]["HW_SS"]=holt_error_winters_ss
    i+=1
```

5.4- HW SAISONNALITE ADDITIVE

In [27]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    holt_error_winters_sa={}
    holt_winters_sa=all_results[key]["HW_SA"]
    for beta in all_alpha:
        for gamma in all_alpha:
            for omega in all_alpha:
                real=np.array(df[FEATURE])
                predic=np.array(holt_winters_sa[f"{beta} {gamma} {omega}"].x_hat)
                holt_error_winters_sa[f"{beta} {gamma} {omega}"]=sum_square_error(real[:taille_list[key]], predic)
    all_errors[key]["HW_SA"]=holt_error_winters_sa
    i+=1
```

5.5- HW SAISONNALITE MULTIPLICATIVE

In [28]:
```
i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    holt_error_winters_sm={}
    holt_winters_sm=all_results[key]["HW_SM"]
    for beta in all_alpha:
        for gamma in all_alpha:
            for omega in all_alpha:
                real=np.array(df[FEATURE])
                predic=np.array(holt_winters_sm[f"{beta} {gamma} {omega}"].x_hat)
                holt_error_winters_sm[f"{beta} {gamma} {omega}"]=sum_square_error(real[:taille_list[key]], predic)
    all_errors[key]["HW_SM"]=holt_error_winters_sm
    i+=1
```

On construit maintenant un tableau pour visualiser les différentes erreurs

In [29]:
```
d=['Lissage_Simple', 'Lissage_Double', 'HW_SS', 'HW_SA', 'HW_SM']
dd=['DataSet 1', 'DataSet 2', 'DataSet 3', 'DataSet 4', 'DataSet 5']
i=0
print_r=[]
for key in d:
    for key2 in all_errors['DataSet 1'][key].keys():
        line=[f"{key} ({key2})"]
        for key1 in dd:
            min_value=min(all_errors[key1][key].values())
            if all_errors[key1][key][key2]==min_value:
                line.append(colored(all_errors[key1][key][key2],'light_cyan',attrs=['bold']))
            else:
                line.append(all_errors[key1][key][key2])
        print_r.append(line)
```

In [30]:
```
print(tabulate(print_r,tablefmt="fancy_grid",headers=all_errors.keys(),colalign='center',))
```

	DataSet 1	DataSet 2	DataSet 3	DataSet 4	DataSet 5
Lissage_Simple (0.1)	**1415.25**	**2.1004e+21**	**5.83197e+13**	**11.2013**	**187.273**
Lissage_Simple (0.5)	49163.9	4.24696e+22	1.0837e+15	386.26	3631.68
Lissage_Simple (0.9)	319344	1.40432e+23	5.00712e+15	3065.88	13510.1
Lissage_Double (0.1)	359518	5.10194e+23	1.31123e+16	2990.47	43325.6
Lissage_Double (0.5)	**77985.9**	**3.08513e+22**	**4.80964e+14**	**736.679**	**6482.36**
Lissage_Double (0.9)	159214	1.10529e+23	2.18803e+15	3073.15	182479
HW_SS (0.1 0.1)	206625	1.54559e+23	2.75136e+15	5962.13	293996
HW_SS (0.1 0.5)	228793	1.94256e+23	2.65184e+15	5263.57	79023.7
HW_SS (0.1 0.9)	300418	2.27367e+23	3.03281e+15	5788.26	52250.2
HW_SS (0.5 0.1)	48494	4.0073e+22	8.96951e+14	567.364	24808.4
HW_SS (0.5 0.5)	166320	6.1677e+22	1.11509e+15	1320.19	**9267.29**
HW_SS (0.5 0.9)	298321	1.3917e+23	1.74917e+15	2380.57	10753.5
HW_SS (0.9 0.1)	**11917.8**	**4.21368e+21**	**8.15911e+13**	**167.766**	11182.5
HW_SS (0.9 0.5)	161454	1.1216e+23	2.24638e+15	1357.77	10611.9
HW_SS (0.9 0.9)	371515	4.42532e+23	1.05481e+16	3013.43	35947
HW_SA (0.1 0.1 0.1)	205030	1.46986e+23	4.55561e+15	3128.54	19504.1
HW_SA (0.1 0.1 0.5)	271786	1.57571e+23	5.47433e+15	2166.65	18135.9
HW_SA (0.1 0.1 0.9)	297566	2.00989e+23	5.81918e+15	1955.06	20968.9
HW_SA (0.1 0.5 0.1)	201621	1.70538e+23	4.51848e+15	3906.02	11331

HW_SA (0.1 0.5 0.5)	271827	1.62585e+23	5.57738e+15	2264.66	11765.5
HW_SA (0.1 0.5 0.9)	308490	2.01101e+23	5.95641e+15	1880.85	14717.9
HW_SA (0.1 0.9 0.1)	213841	1.94891e+23	4.94868e+15	4409.87	10490.1
HW_SA (0.1 0.9 0.5)	310847	1.69332e+23	5.91773e+15	2469.59	11046.9
HW_SA (0.1 0.9 0.9)	586954	2.03158e+23	6.22705e+15	1967.04	13866.4
HW_SA (0.5 0.1 0.1)	66756.9	4.09593e+22	3.03244e+15	477.121	4631.24
HW_SA (0.5 0.1 0.5)	163018	4.58603e+22	4.3725e+15	441.676	4710.06
HW_SA (0.5 0.1 0.9)	247170	5.3604e+22	4.88281e+15	429.774	5356.19
HW_SA (0.5 0.5 0.1)	77307.2	6.06515e+22	3.61336e+15	1228.19	3323.22
HW_SA (0.5 0.5 0.5)	127891	5.85021e+22	4.9139e+15	1032.48	3296.12
HW_SA (0.5 0.5 0.9)	302741	5.9673e+22	5.67835e+15	900.606	3589.82
HW_SA (0.5 0.9 0.1)	158626	1.35462e+23	4.4317e+15	2239.65	5964.94
HW_SA (0.5 0.9 0.5)	197298	1.24677e+23	5.8695e+15	1900.56	5876.35
HW_SA (0.5 0.9 0.9)	302077	1.21701e+23	7.3061e+15	1666.86	6295.08
HW_SA (0.9 0.1 0.1)	**35072.1**	4.19814e+21	**1.89622e+15**	147.94	**800.819**
HW_SA (0.9 0.1 0.5)	38436.2	4.1399e+21	2.91212e+15	144.435	807.146
HW_SA (0.9 0.1 0.9)	45230.9	**4.0881e+21**	3.52156e+15	**141.12**	814.828
HW_SA (0.9 0.5 0.1)	119933	1.12578e+23	3.48388e+15	1352.15	10083.6
HW_SA (0.9 0.5 0.5)	103416	1.14393e+23	4.50024e+15	1320.38	8283.4
HW_SA (0.9 0.5 0.9)	94222	1.1665e+23	5.10969e+15	1300.82	8343.6
HW_SA (0.9 0.9 0.1)	265548	4.46707e+23	7.55908e+15	3078.47	49323.3

HW_SA (0.9 0.9 0.5)	234386	4.64806e+23	8.53665e+15	3025.35	40173.5
HW_SA (0.9 0.9 0.9)	211752	4.86597e+23	9.26785e+15	3035.3	41508.2
HW_SM (0.1 0.1 0.1)	208869	9.13234e+23	3.70744e+15	3975.84	3.36419e+08
HW_SM (0.1 0.1 0.5)	222310	4.45804e+34	1.01478e+16	44407.8	697483
HW_SM (0.1 0.1 0.9)	241961	1.86378e+58	1.47226e+16	5.29107e+07	2.45743e+06
HW_SM (0.1 0.5 0.1)	212110	1.60301e+24	2.83754e+15	407873	9177.63
HW_SM (0.1 0.5 0.5)	210835	4.1238e+28	2.95796e+15	1.16198e+06	18195.1
HW_SM (0.1 0.5 0.9)	210895	2.65188e+66	2.8821e+15	1.12144e+06	34812.6
HW_SM (0.1 0.9 0.1)	226895	1.60664e+24	3.27219e+15	1.68156e+06	8167.75
HW_SM (0.1 0.9 0.5)	240301	1.35223e+24	2.79101e+15	1.26793e+06	8659.52
HW_SM (0.1 0.9 0.9)	566183	9.22201e+65	2.46814e+15	1.2179e+06	395807
HW_SM (0.5 0.1 0.1)	127492	3.01972e+24	1.87406e+15	672.984	**3537.99**
HW_SM (0.5 0.1 0.5)	171005	1.72305e+32	1.82288e+15	76149.3	6481.19
HW_SM (0.5 0.1 0.9)	212596	5.43776e+48	1.90247e+15	20291.1	102021
HW_SM (0.5 0.5 0.1)	63404.7	**2.37025e+23**	2.47607e+15	1000.55	4804.08
HW_SM (0.5 0.5 0.5)	129992	2.68578e+27	2.29028e+15	446046	21939.2
HW_SM (0.5 0.5 0.9)	349833	9.58245e+43	2.70604e+15	12662	549255
HW_SM (0.5 0.9 0.1)	**57348.3**	1.27248e+24	3.36266e+15	1926.31	9225.02
HW_SM (0.5 0.9 0.5)	71670.7	1.26793e+24	3.24029e+15	5456.44	24157.7
HW_SM (0.5 0.9 0.9)	247293	1.73206e+38	4.33889e+15	20670.5	727511

HW_SM (0.9 0.1 0.1)	98484.5	6.2114e+23	**1.76229e+15**	**499.381**	4895.21
HW_SM (0.9 0.1 0.5)	96274.3	1.84015e+26	2.09268e+15	609.964	5718.79
HW_SM (0.9 0.1 0.9)	96061.7	3.69192e+28	2.20476e+15	13924.6	15124.9
HW_SM (0.9 0.5 0.1)	98893.4	4.40116e+23	3.83211e+15	1944.66	21433.3
HW_SM (0.9 0.5 0.5)	83462.4	2.82129e+25	4.51113e+15	1988.94	23893.1
HW_SM (0.9 0.5 0.9)	73214	2.35661e+27	4.64947e+15	25870.3	49345.1
HW_SM (0.9 0.9 0.1)	175174	5.83628e+23	9.05121e+15	4248.79	67801.1
HW_SM (0.9 0.9 0.5)	150279	4.88089e+24	1.10113e+16	4544.73	74506.1
HW_SM (0.9 0.9 0.9)	132583	2.6017e+26	1.09101e+16	59186.6	133933

In [31]:
```
min_error=[]
for n in range(1,6):
    l1=[]
    ld=[]
    ss=[]
    sa=[]
    sm=[]
    for i in range(len(print_r)):
        if i< 3:
            l1.append(print_r[i][n])
        elif i>=3 and i< 3+3:
            ld.append(print_r[i][n])
        elif i>= 3+3 and i< 3+3+3*3 :
            ss.append(print_r[i][n])
        elif i>= 3+3+3*3 and i< 3+3+3*3+3*3*3 :
            sa.append(print_r[i][n])
        elif i>= 3+3+3*3+3*3*3 and i< 3+3+3*3+3*3*3+3*3*3 :
            sm.append(print_r[i][n])
    line=[f'DataSet {n}']
    index=list(all_errors[f'DataSet {n}']['Lissage_Simple'].keys())
    line.append(index[np.argmin(l1)])
    index=list(all_errors[f'DataSet {n}']['Lissage_Double'].keys())
```

```
    line.append(index[np.argmin(ld)])
    index=list(all_errors[f'DataSet {n}']['HW_SS'].keys())
    line.append(index[np.argmin(ss)])
    index=list(all_errors[f'DataSet {n}']['HW_SA'].keys())
    line.append(index[np.argmin(sa)])
    index=list(all_errors[f'DataSet {n}']['HW_SM'].keys())
    line.append(index[np.argmin(sm)])
    min_error.append(line)
```

Les lissages avec les plus petites valeurs d'erreurs pour chaque DataSet sont :

In [32]:
```
print(tabulate(min_error,tablefmt="fancy_grid",headers=all_errors['DataSet 1'].keys(),colalign='center'))
```

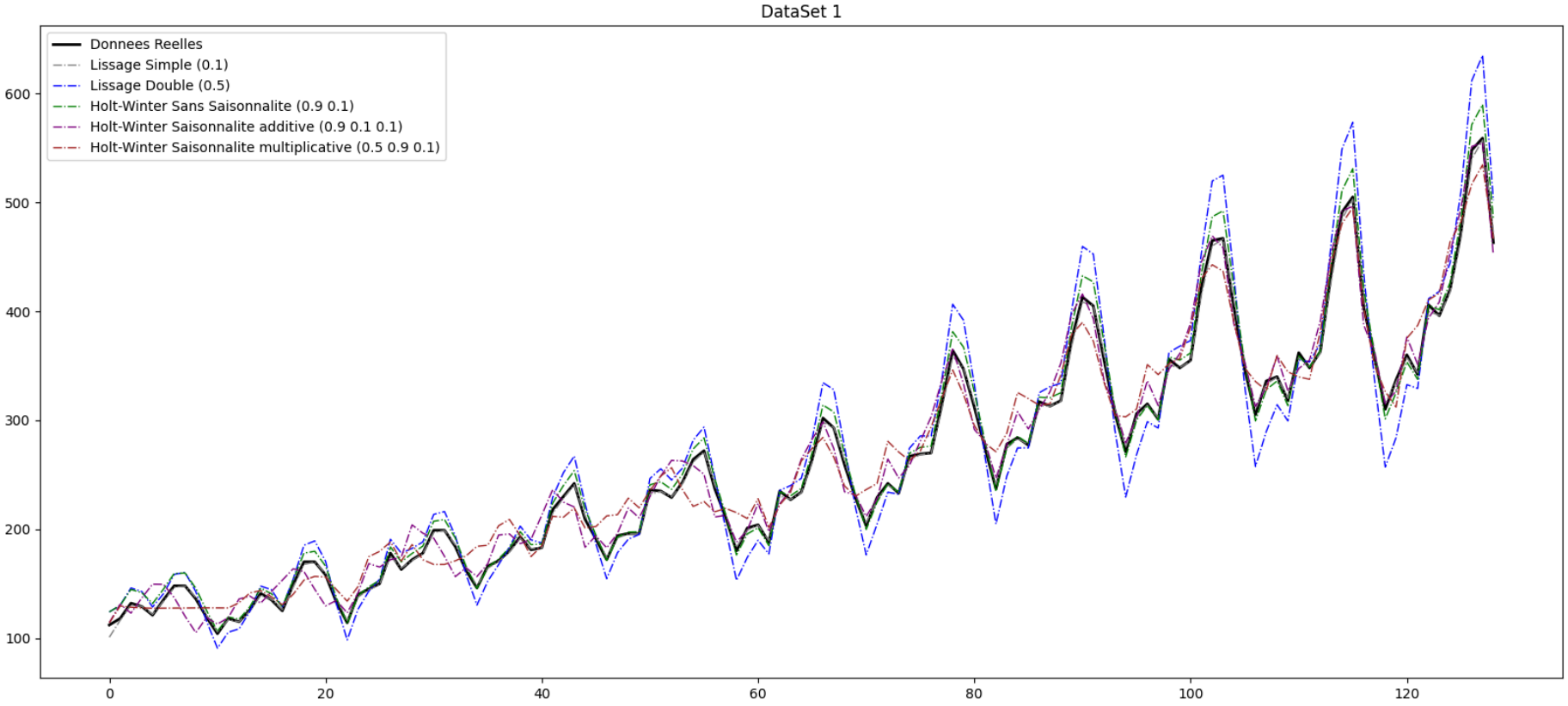
	Lissage_Simple	Lissage_Double	HW_SS	HW_SA	HW_SM
DataSet 1	0.1	0.5	0.9 0.1	0.9 0.1 0.1	0.5 0.9 0.1
DataSet 2	0.1	0.5	0.9 0.1	0.9 0.1 0.9	0.5 0.5 0.1
DataSet 3	0.1	0.5	0.9 0.1	0.9 0.1 0.1	0.9 0.1 0.1
DataSet 4	0.1	0.5	0.9 0.1	0.9 0.1 0.9	0.9 0.1 0.1
DataSet 5	0.1	0.5	0.5 0.5	0.9 0.1 0.1	0.5 0.1 0.1

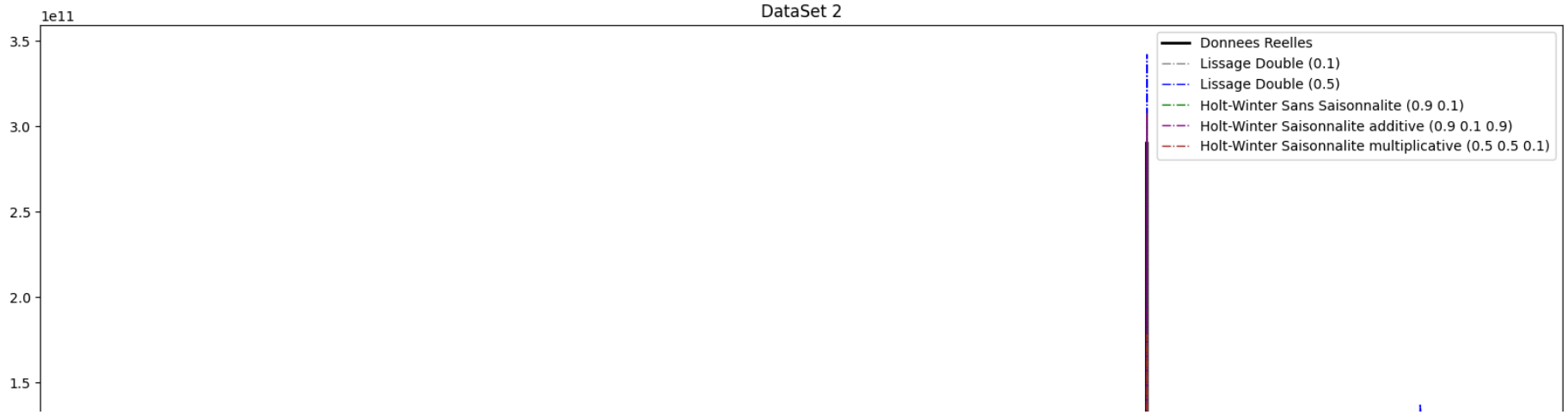
6- VISUALISATION DES RESULTATS

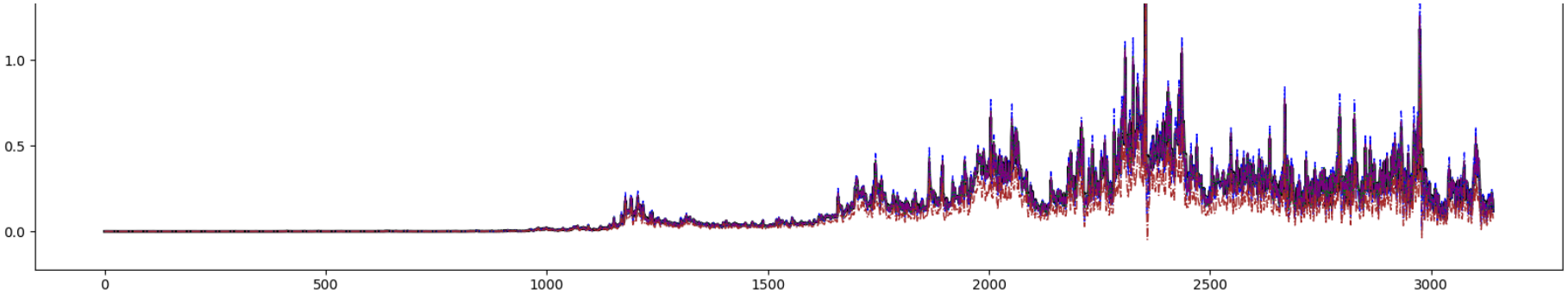
6.1- SUR LES 90% DES DONNEES

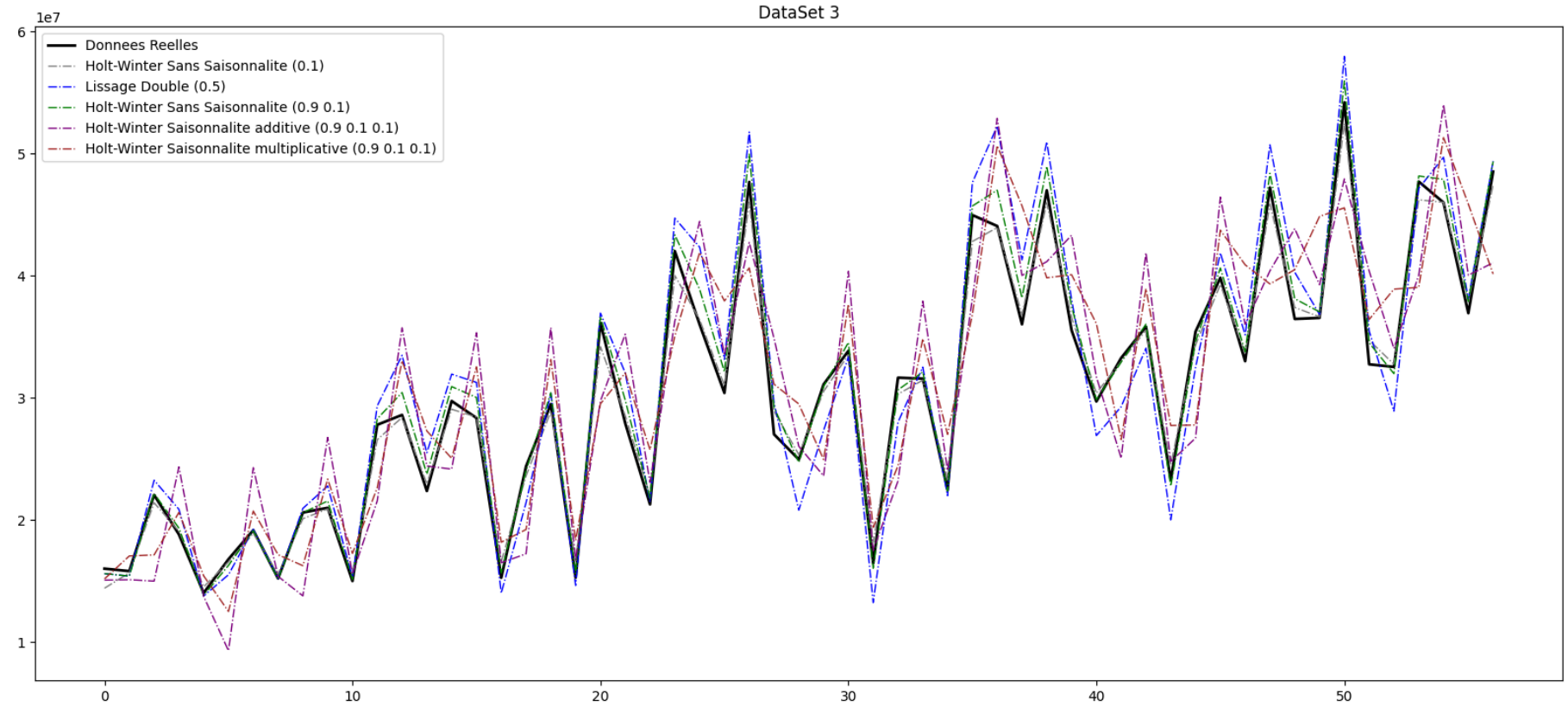
In [33]:
```
plt.figure(figsize=[20,50])
i=1
method_names=['Lissage_Simple', 'Lissage_Double', 'HW_SS', 'HW_SA', 'HW_SM']
method_full_names=['Lissage Simple','Lissage Double','Holt-Winter Sans Saisonnalite','Holt-Winter Saisonnalite addi
                   'Holt-Winter Saisonnalite multiplicative']
colors=['grey','blue','green','purple','brown']
for key in all_df.keys():
```

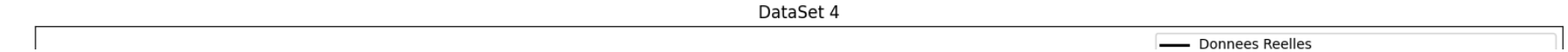
```
    df=all_df[key]
    FEATURE=Features_List[i-1]
    plt.subplot(5,1,i)
    plt.plot(df[FEATURE][:taille_list[key]] , label='Donnees Reelles', c='black', ls='-', lw=2)
    plt.plot(all_results[key][method_names[0]][min_error[i-1][1]].L1 , label=method_full_names[i-1]+f" ({min_error[
    for j in range(2,6):
            plt.plot(all_results[key][method_names[j-1]][min_error[i-1][j]].x_hat , label=method_full_names[j-1]+f"
    plt.title(key)
    plt.legend()
    i+=1
```

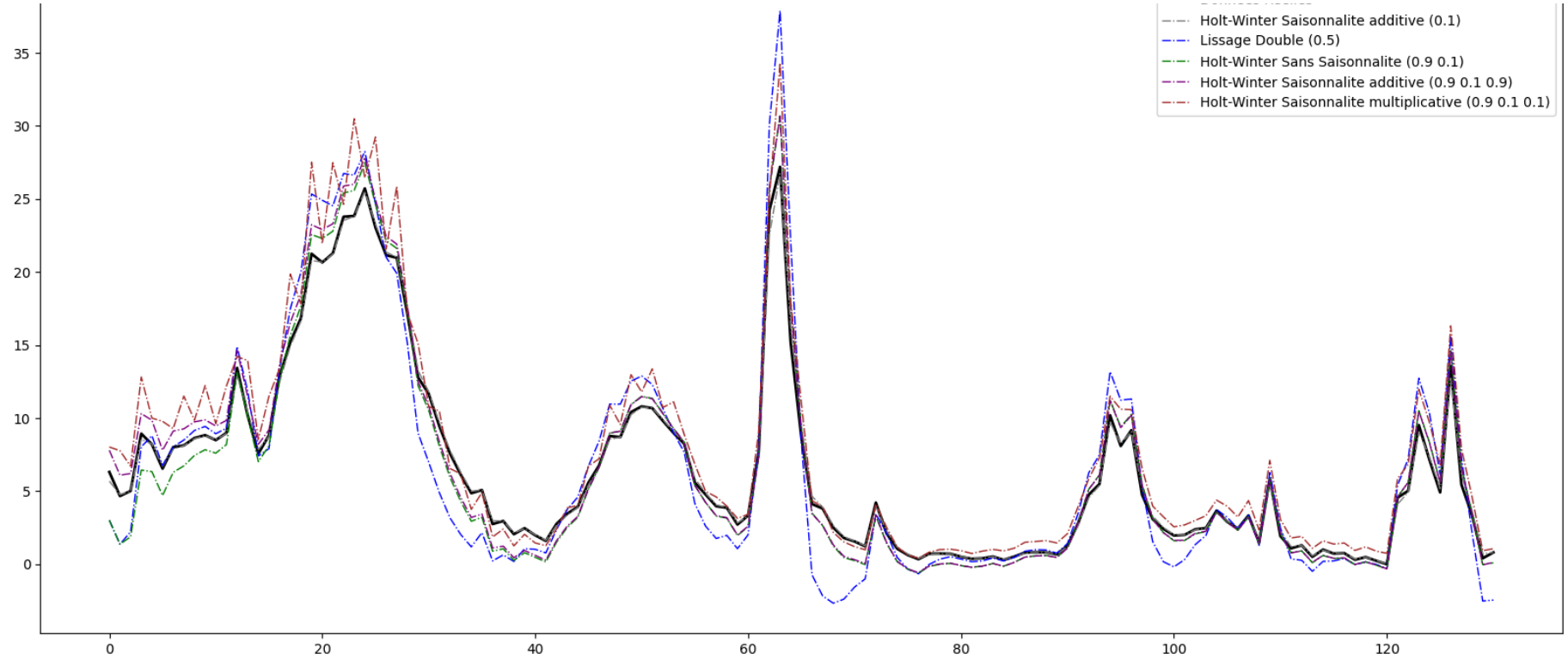

DataSet 1
Donnees Reelles
Lissage Simple (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.9 0.1)
Holt-Winter Saisonnalite additive (0.9 0.1 0.1)
Holt-Winter Saisonnalite multiplicative (0.5 0.9 0.1)
600
500
400
300
200
100
0
20
40
60
80
100
120


DataSet 2
1e11
3.5
3.0
2.5
2.0
1.5
Donnees Reelles
Lissage Double (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.9 0.1)
Holt-Winter Saisonnalite additive (0.9 0.1 0.9)
Holt-Winter Saisonnalite multiplicative (0.5 0.5 0.1)

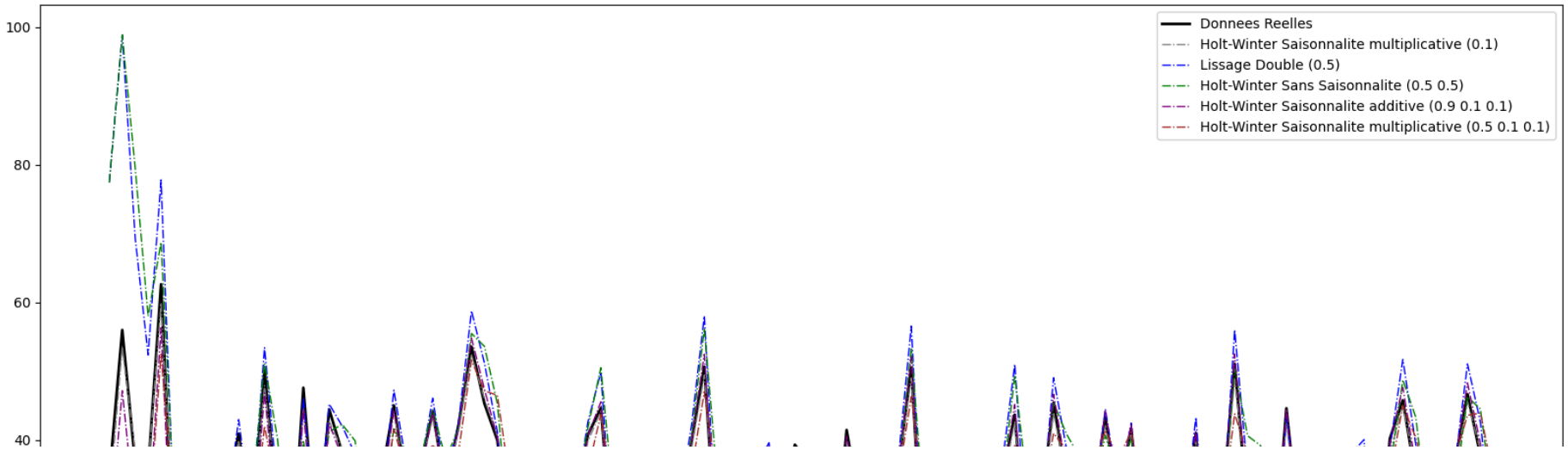

1.0
0.5
0.0
0
500
1000
1500
2000
2500
3000

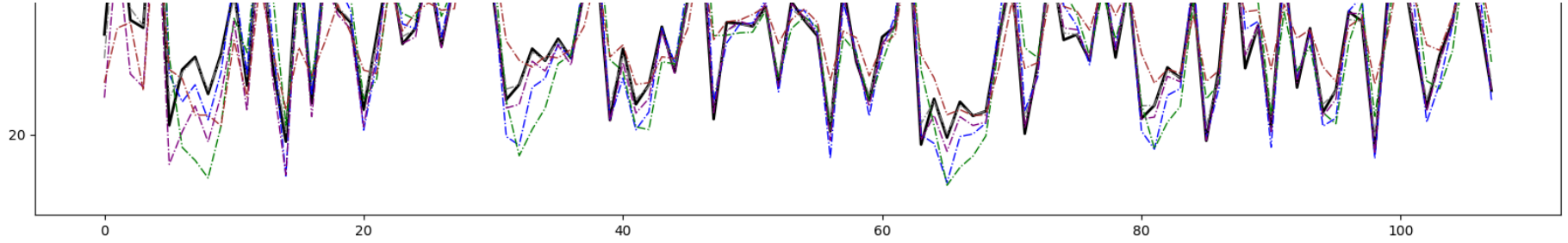

DataSet 3
1e7
Donnees Reelles
Holt-Winter Sans Saisonnalite (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.9 0.1)
Holt-Winter Saisonnalite additive (0.9 0.1 0.1)
Holt-Winter Saisonnalite multiplicative (0.9 0.1 0.1)
6
5
4
3
2
1
0
10
20
30
40
50


DataSet 4
Donnees Reelles


Holt-Winter Saisonnalite additive (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.9 0.1)
Holt-Winter Saisonnalite additive (0.9 0.1 0.9)
Holt-Winter Saisonnalite multiplicative (0.9 0.1 0.1)

DataSet 5


Donnees Reelles
Holt-Winter Saisonnalite multiplicative (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.5 0.5)
Holt-Winter Saisonnalite additive (0.9 0.1 0.1)
Holt-Winter Saisonnalite multiplicative (0.5 0.1 0.1)



6.2- SUR LES 10% DES DONNEES

In [34]:
```
method_names=['Lissage_Simple', 'Lissage_Double', 'HW_SS', 'HW_SA', 'HW_SM']
method_full_names=['Lissage Simple','Lissage Double','Holt-Winter Sans Saisonnalite','Holt-Winter Saisonnalite addi
                   'Holt-Winter Saisonnalite multiplicative']
colors=['grey','blue','green','purple','brown']
i=1
for key in all_df.keys():
    df=all_df[key]
    all_results[key][method_names[0]][min_error[i-1][1]].predict(len(df)-taille_list[key])
    i+=1
```

In [35]:
```
i=1
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i-1]
    all_results[key][method_names[1]][min_error[i-1][2]].predict(int(len(df)*0.1))
    i+=1
```

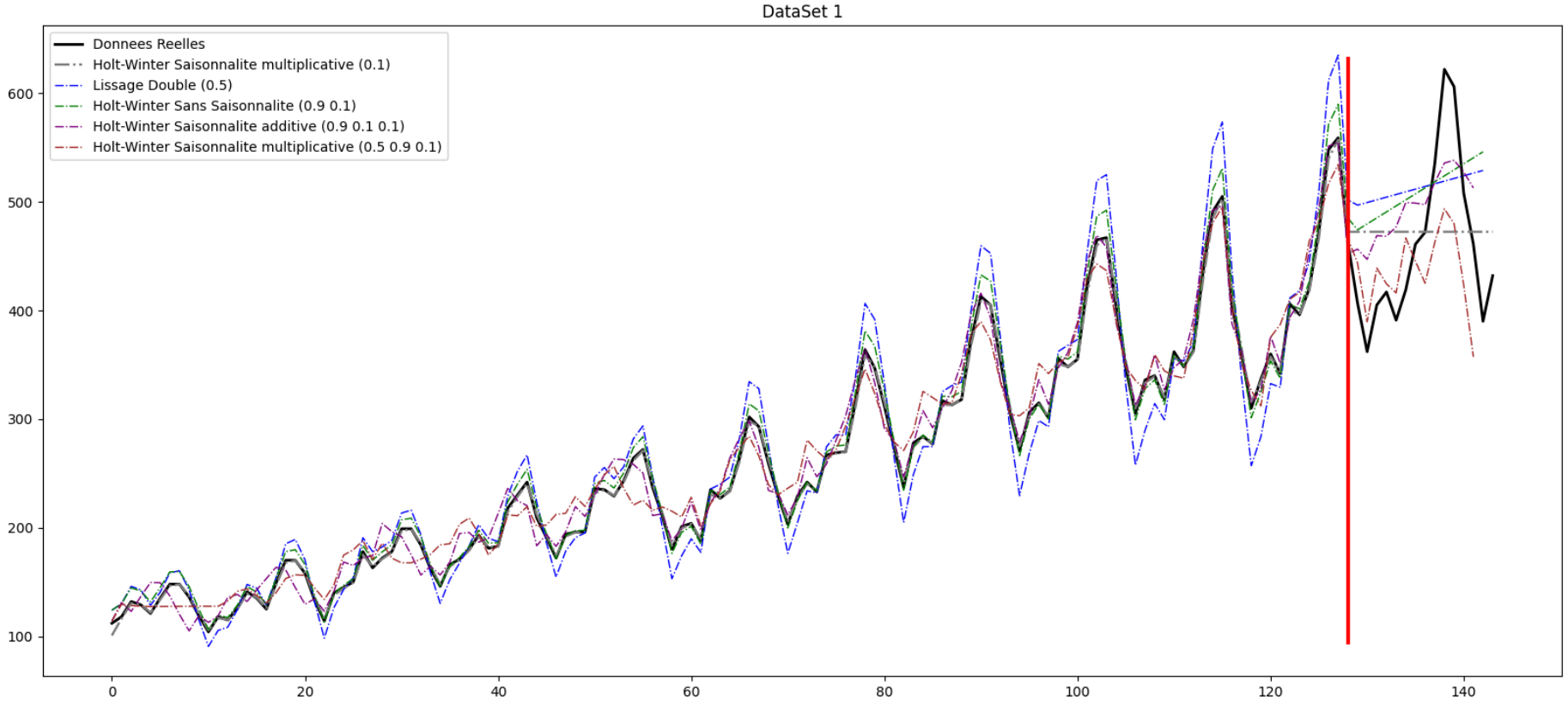
In [36]:
```
i=1
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i-1]
    all_results[key][method_names[2]][min_error[i-1][3]].predict(int(len(df)*0.1))
    i+=1
```

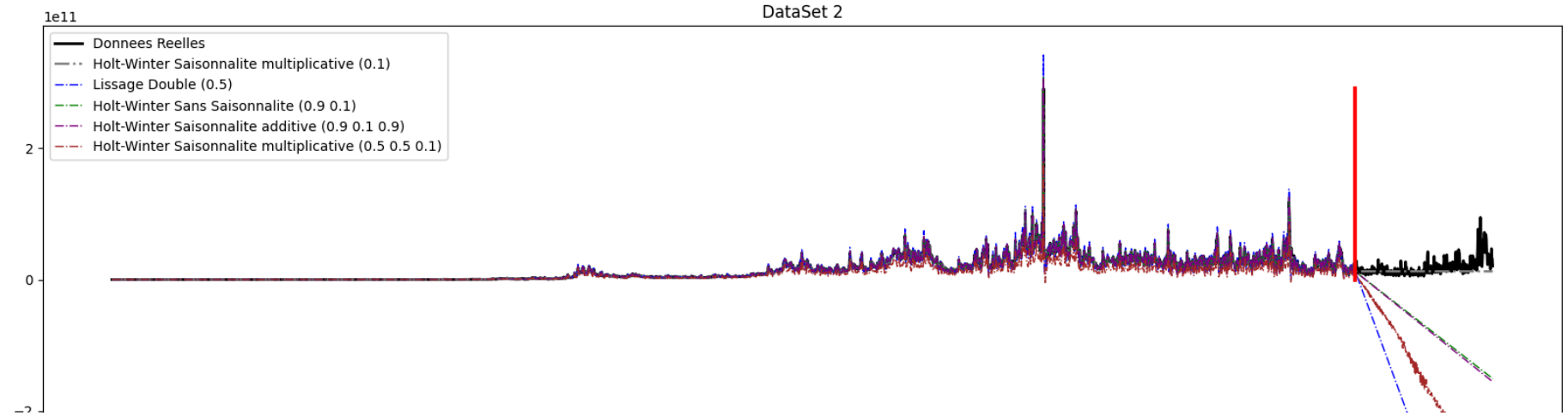
In [37]:
```
i=1
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i-1]
```

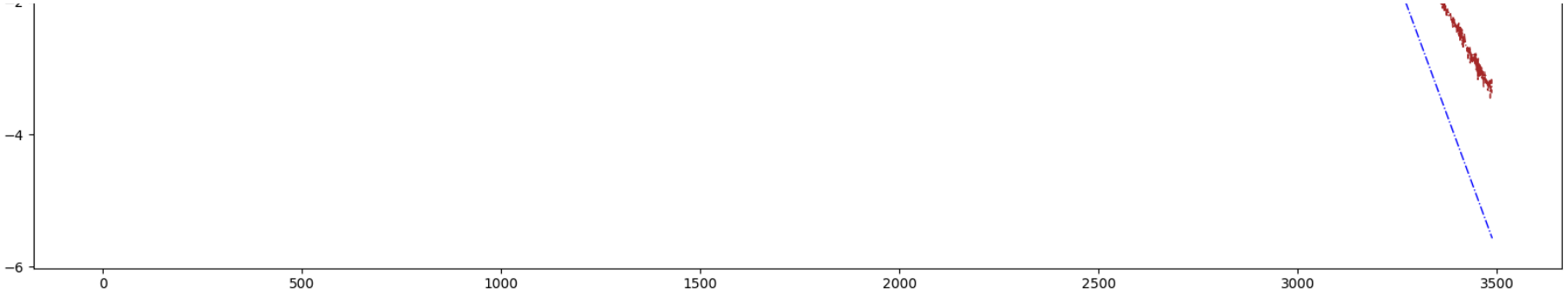
```
    all_results[key][method_names[3]][min_error[i-1][4]].predict(int(len(df)*0.1))
    i+=1
```

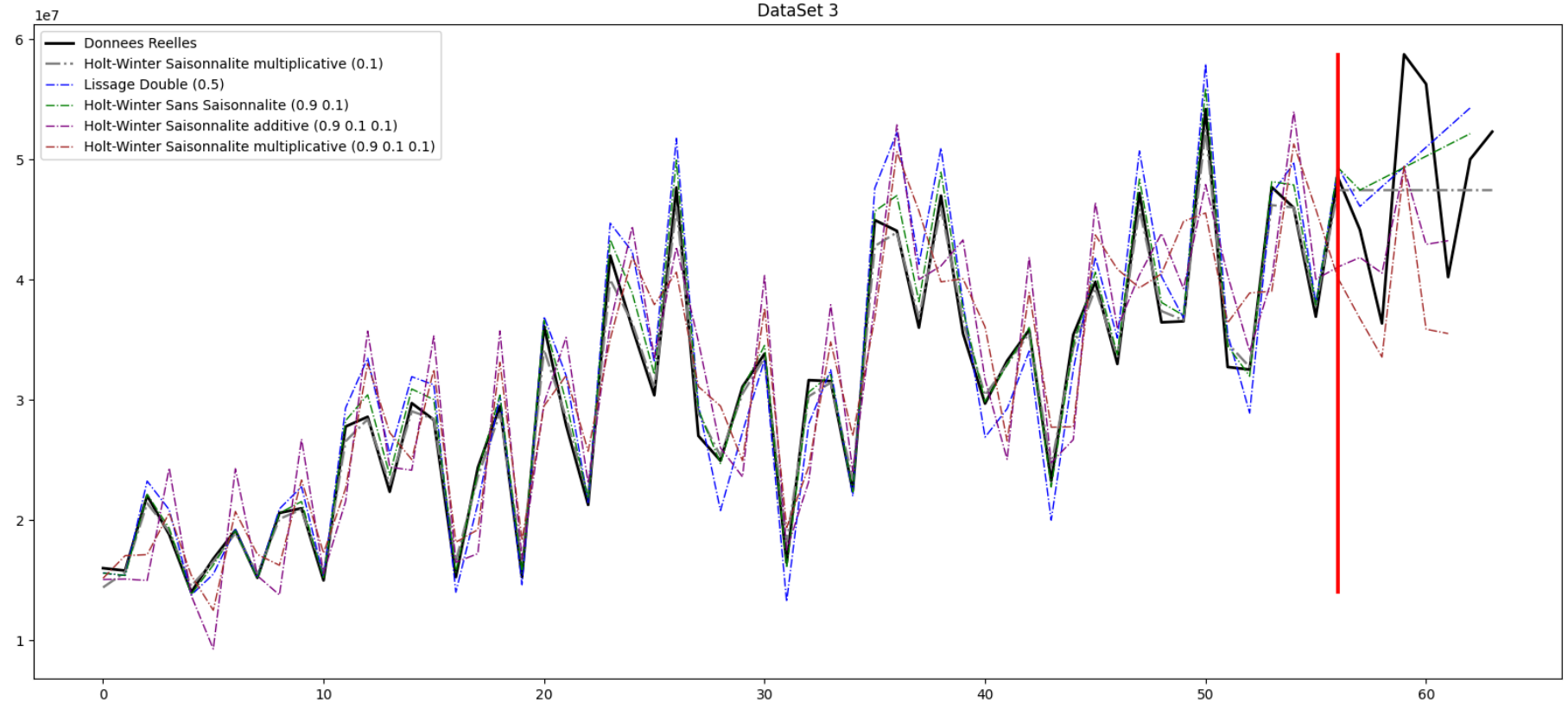
In [38]:
```
i=1
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i-1]
    all_results[key][method_names[4]][min_error[i-1][5]].predict(int(len(df)*0.1))
    i+=1
```

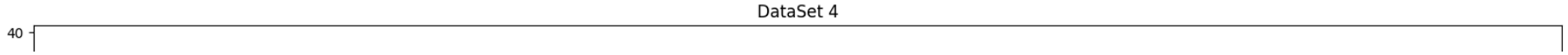
In [39]:
```
plt.figure(figsize=[20,50])
i=1
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i-1]
    plt.subplot(5,1,i)
    plt.plot(df[FEATURE] , label='Donnees Reelles', c='black', ls='-', lw=2)
    plt.plot(all_results[key][method_names[0]][min_error[i-1][1]].L1 , label=method_full_names[j-1]+f" ({min_error[
    for j in range(2,6):
            plt.plot(all_results[key][method_names[j-1]][min_error[i-1][j]].x_hat , label=method_full_names[j-1]+f"
    m1=np.max(df[FEATURE]+10)
    m2=np.min(df[FEATURE]-10)
    x=taille_list[key]-1
    y=np.linspace(m2,m1,100)
    x=np.ones(y.shape)*x
    plt.plot(x,y, c="red", lw=2.7, ls="-")
    plt.title(key)
    plt.legend()
    i+=1
```

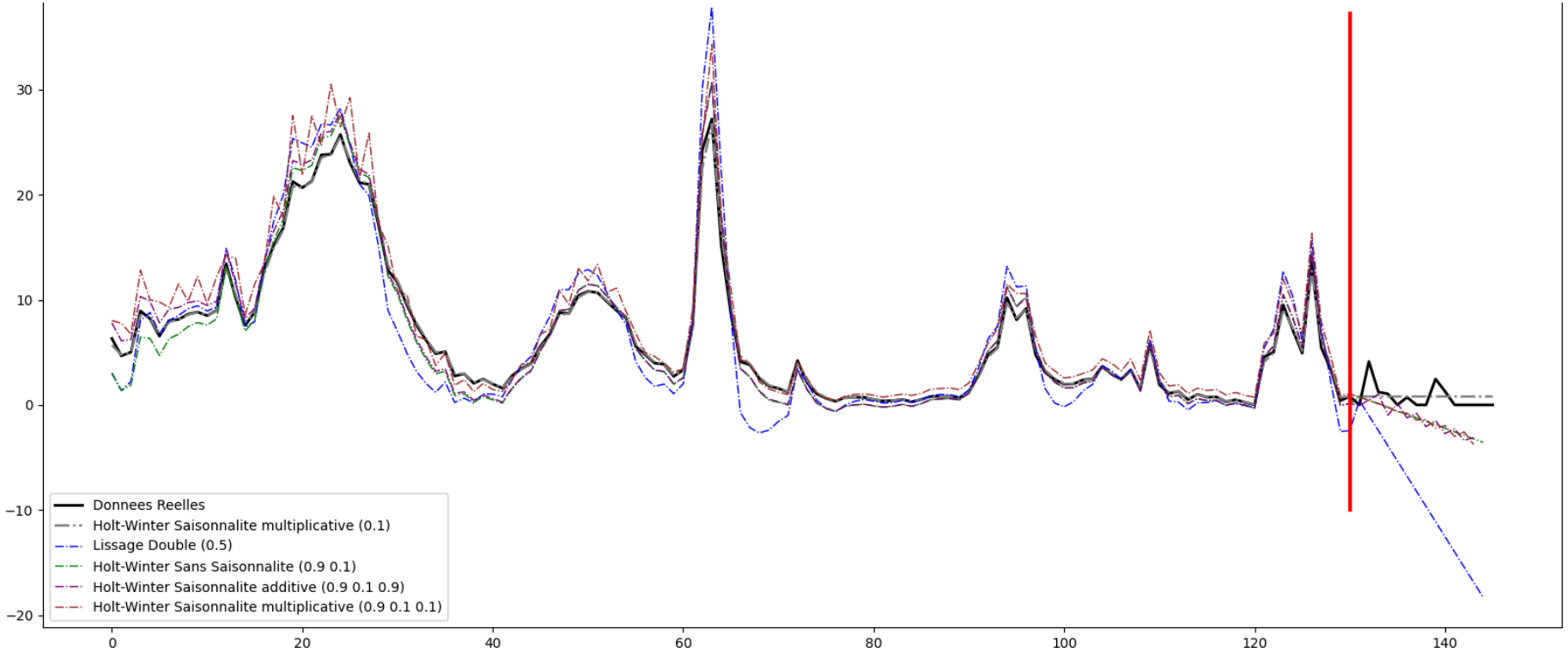

DataSet 1
Donnees Reelles
Holt-Winter Saisonnalite multiplicative (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.9 0.1)
Holt-Winter Saisonnalite additive (0.9 0.1 0.1)
Holt-Winter Saisonnalite multiplicative (0.5 0.9 0.1)
600
500
400
300
200
100
0
20
40
60
80
100
120
140

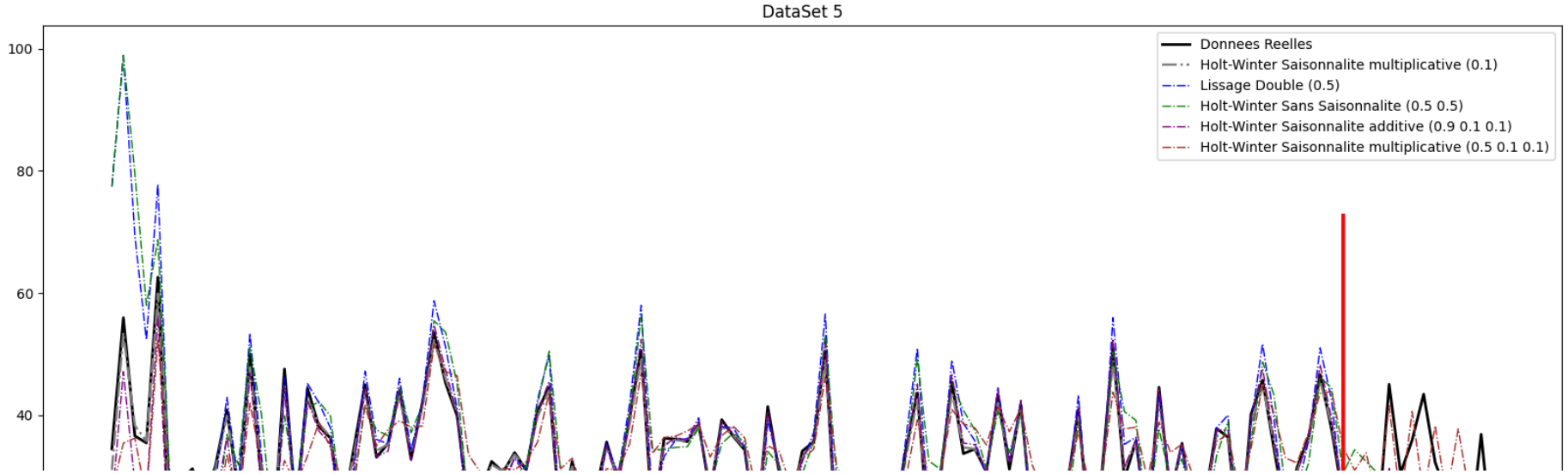

DataSet 2
1e11
Donnees Reelles
Holt-Winter Saisonnalite multiplicative (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.9 0.1)
Holt-Winter Saisonnalite additive (0.9 0.1 0.9)
Holt-Winter Saisonnalite multiplicative (0.5 0.5 0.1)
2
0

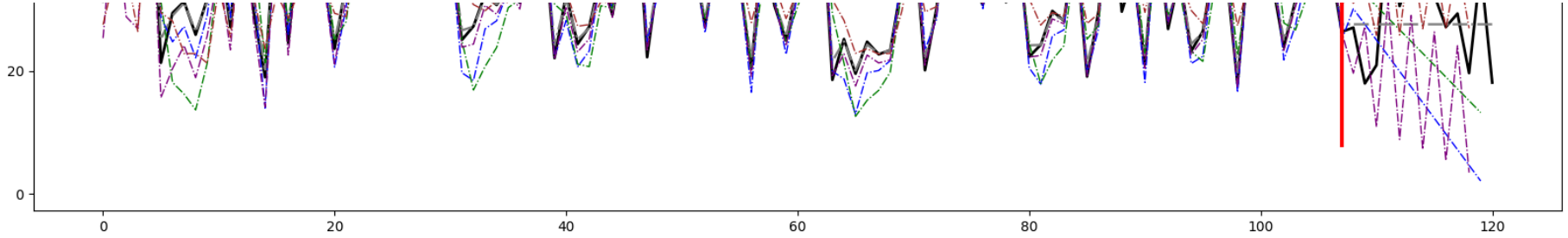

−4
−6
0
500
1000
1500
2000
2500
3000
3500


DataSet 3
1e7
Donnees Reelles
Holt-Winter Saisonnalite multiplicative (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.9 0.1)
Holt-Winter Saisonnalite additive (0.9 0.1 0.1)
Holt-Winter Saisonnalite multiplicative (0.9 0.1 0.1)
6
5
4
3
2
1
0
10
20
30
40
50
60


DataSet 4
40


Donnees Reelles
Holt-Winter Saisonnalite multiplicative (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.9 0.1)
Holt-Winter Saisonnalite additive (0.9 0.1 0.9)
Holt-Winter Saisonnalite multiplicative (0.9 0.1 0.1)


DataSet 5
Donnees Reelles
Holt-Winter Saisonnalite multiplicative (0.1)
Lissage Double (0.5)
Holt-Winter Sans Saisonnalite (0.5 0.5)
Holt-Winter Saisonnalite additive (0.9 0.1 0.1)
Holt-Winter Saisonnalite multiplicative (0.5 0.1 0.1)



In [ ]:

```
i=0
all_errors2={}
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    all_error_L1={}
    all_L1=all_results[key]["Lissage_Simple"]
    for alpha in all_alpha:
        real=np.array(df[FEATURE])
        predic=np.array(all_L1[f"{alpha}"].L1)
        all_error_L1[f"{alpha}"]=sum_square_error(real[taille_list[key]:], predic[taille_list[key]:])
    all_errors2[key]={"Lissage_Simple":all_error_L1}
    i+=1

i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    all_LissDouble=all_results[key]['Lissage_Double']
    all_error_LissDouble={}
    for alpha in all_alpha:
        real=np.array(df[FEATURE])
        predic=np.array(all_LissDouble[f"{alpha}"].x_hat)
        all_error_LissDouble[f"{alpha}"]=sum_square_error(real[taille_list[key]:], predic[taille_list[key]:])
    all_errors2[key]["Lissage_Double"]=all_error_LissDouble
    i+=1

i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
```

```
    holt_error_winters_ss={}
    holt_winters_ss=all_results[key]["HW_SS"]
    for beta in all_alpha:
        for gamma in all_alpha:
            real=np.array(df[FEATURE])
            predic=np.array(holt_winters_ss[f"{beta} {gamma}"].x_hat)
            holt_error_winters_ss[f"{beta} {gamma}"]=sum_square_error(real[taille_list[key]:], predic[taille_list[k
    all_errors2[key]["HW_SS"]=holt_error_winters_ss
    i+=1

i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    holt_error_winters_sa={}
    holt_winters_sa=all_results[key]["HW_SA"]
    for beta in all_alpha:
        for gamma in all_alpha:
            for omega in all_alpha:
                real=np.array(df[FEATURE])
                predic=np.array(holt_winters_sa[f"{beta} {gamma} {omega}"].x_hat)
                holt_error_winters_sa[f"{beta} {gamma} {omega}"]=sum_square_error(real[taille_list[key]:], predic[t
    all_errors2[key]["HW_SA"]=holt_error_winters_sa
    i+=1

i=0
for key in all_df.keys():
    df=all_df[key]
    FEATURE=Features_List[i]
    holt_error_winters_sm={}
    holt_winters_sm=all_results[key]["HW_SM"]
    for beta in all_alpha:
        for gamma in all_alpha:
            for omega in all_alpha:
                real=np.array(df[FEATURE])
                predic=np.array(holt_winters_sm[f"{beta} {gamma} {omega}"].x_hat)
                holt_error_winters_sm[f"{beta} {gamma} {omega}"]=sum_square_error(real[taille_list[key]:], predic[t
    all_errors2[key]["HW_SM"]=holt_error_winters_sm
    i+=1
```

In [ ]:
```
d=['Lissage_Simple', 'Lissage_Double', 'HW_SS', 'HW_SA', 'HW_SM']
dd=['DataSet 1', 'DataSet 2', 'DataSet 3', 'DataSet 4', 'DataSet 5']
i=0
print_r=[]
for key in d:
    for key2 in all_errors2['DataSet 1'][key].keys():
        line=[f"{key} ({key2})"]
        for key1 in dd:
            line.append(all_errors[key1][key][key2])
        print_r.append(line)
print(tabulate(print_r,tablefmt="fancy_grid",headers=all_errors.keys(),colalign='center'))
```

In [ ]:
```

```