École Centrale Nantes

Info IA - PIIA Marées
Rapport

Rapport 2 - PIIA1 Marées

Élèves :
Daniel MACEDO GALEMBECK
Nicolas CONTRERAS
Alexandre THOMASSIN
Liushuangfei XIE

Enseignant :
Mr. ROUX

17 février 2023

Table des matières

1 Rapport des deux semaines passées

Ces deux semaines ont été consacrées à l'apprentissage de TensorFlow et à la recherche de solutions.

1.1 Préparation des données

Afin que les données soit utilisables par des librairies comme TensorFlow ou ScikitLearn, il faut d'abord transformer les fichier .txt en .csv. On en profite alors pour les regroupé selon un nombre d'année chosit (ici 10ans) pour avoir des ensembles de données plus fiables.

Code Python - TXT to CSV :

```
""" Il faut executer le script dans le dossier contenant les fichier .txt"""

# Intervalle de regroupement en annees
years_inter = 10

# Annee du premier fichier
start_year = 1846

# Annee du dernier fichier
end_year = 2022

# Variables pour savoir si il faut creer un nouveau fichier
cur_file = "1846_1855.csv"
cur_year = 0

# On ecrit les labels des valeurs dans le premier fichier
csv_file = open(cur_file, "w")
csv_file.write("Date,Valeur,Source")
csv_file.close()

for i in range(start_year, end_year):
    if cur_year == years_inter:
        cur_file = f"{i}_{min(i+9, end_year)}.csv"
        cur_year = 0

        # On ecrit les labels des valeurs
        csv_file = open(cur_file, "w")
        csv_file.write("Date,Valeur,Source")
        csv_file.close()

    try:
        csv_file = open(cur_file, "a")
        csv_file.write("\n") # Saute une ligne suite a l'ouverture en mode '
    append'

        with open(f"3_{i}.txt") as txt_file:
            lines = txt_file.readlines()
            for line in lines:
                if not line[0] == "#":
                    elt = line.split(";")
                    csv_file.write(f"{elt[0]},{elt[1]},{elt[2]}")

```

```
        csv_file.close()

    except FileNotFoundError:
        print(f"Fichier {i}.txt non trouv ")

    # print(cur_year)
    cur_year += 1
```

1.2 Étude du module TensorFlow

Comme expliqué dans le rapport de la semaine dernière, nous avons choisi de nous familiarisé avec la librairie TensorFlow.

Dans cette optique, nous avons suivi quelques tutoriel afin d'essayer de prédire des données et les comparé aux données que l'on a déjà.

On a donc put écrire un model d'apprentissage avec TensorFlow (le notebook complet est en annexe) :

Formatage des données :

On commence d'abord par importer nos données d'entrainement et de test :

```
import tensorflow as tf
import pandas as pd
import numpy as np
import time
import datetime

history_size = 10
```

```
dataframe = pd.read_csv("Data/1996_2005.csv", parse_dates = True)
```

```
#print(dataframe)
value_serie = dataframe["Valeur"]
date = dataframe["Date"]
n = len(value_serie.values)
train_values = value_serie.values[:n//2].reshape(-1,1)
test_values = value_serie.values[n//2:].reshape(-1,1)
```

On implémente ensuite une fonction permettant d'adapter les données a l'utilisation de TensorFlow. En effet les données ont besoin d'être sous forme de "fenêtres" contenant les valeurs d'entrainement et un label servant de resultats a l'apprentissage.

```
def create_window(dataset, start_index, end_index, history_size):
    data = []
    labels = []

    start_index = start_index + history_size
    if end_index is None:
        end_index = len(dataset)

    for i in range(start_index, end_index-history_size):
        indices = range(i, i+history_size)
        data.append(np.reshape(dataset[indices], (history_size, 1)))
        labels.append(dataset[i+history_size])

    return np.array(data), np.array(labels)
```

```
train_features, train_labels = create_window(train_values, 0, None, history_size)
test_features, test_labels = create_window(test_values, 0, None,history_size)
```

Depuis ces données, on va pouvoir créer des **Dataset** reconnu par TensorFlow :

```
train_dataset = tf.data.Dataset.from_tensor_slices((train_features, train_labels)).batch(100).repeat()
test_dataset = tf.data.Dataset.from_tensor_slices((test_features, test_labels)).batch(100).repeat()
```

Création du modèle et entrainement :

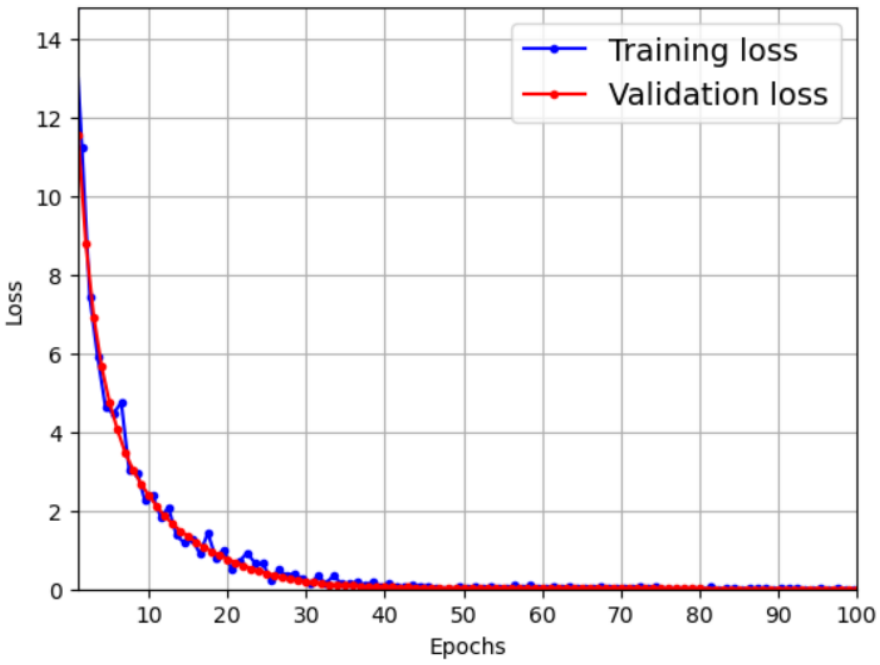
Maintenant que l'on à les données, on va pouvoir initialiser notre modèle et l'entrainer :

```
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(32, input_shape=(history_size, 1)),
    tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
```

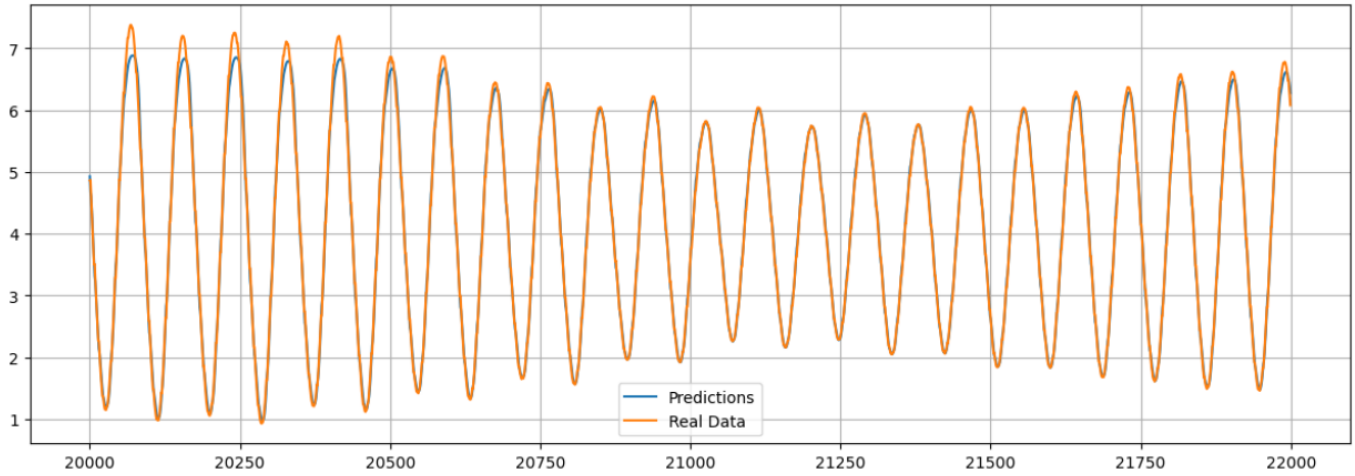
```
history = model.fit(
    train_dataset,
    epochs=100,
    steps_per_epoch=20,
    validation_data=test_dataset,
    validation_steps=3
)
```

Résultats :

Tout d'abord on peut regarder la fonction de pertes lors de l'entrainement comparé aux test :



Puis on peut comparer les données prédites avec les données réelles :



On remarque que dans l'ensemble les données prédites sont très proches des données réelles.

1.3 Étude du module scikit-lean

Nous avons étudié et analysé le module `scikit-learn` qui dispose d'outils simples et efficaces pour l'analyse prédictive des données. Les méthodes de ce module qui sont utiles pour le projet sont les suivantes :

KFold
Fournit des indices de *train/test* pour diviser les données en ensembles de *train/test*.
Nous pouvons l'importer de la manière suivante :

```
from sklearn.model_selection import KFold
```

LinearRegression
`LinearRegression` ajuste un modèle linéaire avec des coefficients $w = (w_1, ..., w_p)$ pour minimiser la somme résiduelle des carrés entre les cibles observées dans l'ensemble de données, et les cibles prédites par l'approximation linéaire.
Nous pouvons l'importer de la manière suivante :

```
from sklearn.linear_model import LinearRegression
```

Dans les **annexes**, on peut voir un modèle de régression linéaire réalisé avec un ensemble de données sur les personnes infectées par le covid. Cet exemple est homologue à notre projet car il utilise les paramètres `datetime` et `float`, c'est-à-dire le même type de paramètres que nous devons utiliser dans le projet.

2 Objectif des deux prochaines semaines

2.1 Programmation d'autres méthodes de prédiction

2.1.1 Modèle Multi-étapes avec TensorFlow

Le modèle créé lors des test de la librairies était un modèle à sortie unique (une seule valeurs d'avances étais prédite a chaque itération). L'idée de ces deux prochaine semaine est de développé un modèle multi-étapes afin de pouvoir prédire plusieurs données à chaque itération (ex : prédire les 10 prochaines données à partir des 10 données précédentes)

2.2 Modèle de régression linéaire avec les données de marées

Nous allons tester le modèle de régression linéaire effectué sur une partie des données de hauteur de marées. Nous utiliserons d'abord une petite quantité de données, afin de vérifier que le modèle génère les résultats attendus et de corriger les erreurs que cela implique. Si tout fonctionne bien, nous testerons avec les données complètes, en les séparant en sections de training et de testing.

3 Annexes

Prédictions avec Tensorflow

Test TF

October 28, 2022

[2]:
```
import tensorflow as tf
import pandas as pd
import numpy as np
import time
import datetime

history_size = 10
```

[3]:
```
dataframe = pd.read_csv("Data/1996_2005.csv", parse_dates = True)
```

[5]:
```
#print(dataframe)
value_serie = dataframe["Valeur"]
date = dataframe["Date"]
n = len(value_serie.values)
train_values = value_serie.values[:n//2].reshape(-1,1)
test_values = value_serie.values[n//2:].reshape(-1,1)
```

[87]:
```
print(f"Valeurs d'entrainement : {train_values}")
print(f"Valeurs de test : {test_values}")
```

```
Valeurs d'entrainement : [[5.88]
 [5.89]
 [5.9 ]
 …
 [5.13]
 [5.  ]
 [4.93]]
Valeurs de test : [[4.75 ]
 [4.68 ]
 [4.5  ]
 …
 [1.91 ]
 [2.087]
 [2.276]]
```

[90]:
```
def create_window(dataset, start_index, end_index, history_size):
    data = []
    labels = []
```

```
    start_index = start_index + history_size
    if end_index is None:
        end_index = len(dataset)

    for i in range(start_index, end_index-history_size):
        indices = range(i, i+history_size)
        data.append(np.reshape(dataset[indices], (history_size, 1)))
        labels.append(dataset[i+history_size])

    return np.array(data), np.array(labels)
```

[91]:
```
train_features, train_labels = create_window(train_values, 0, None,␣
 ↪history_size)
test_features, test_labels = create_window(test_values, 0, None,history_size)

#print(train_features)
#print(train_labels)
```

[92]:
```
train_dataset = tf.data.Dataset.from_tensor_slices((train_features,␣
 ↪train_labels)).batch(100).repeat()
test_dataset = tf.data.Dataset.from_tensor_slices((test_features, test_labels)).
 ↪batch(100).repeat()
```

[159]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(32, input_shape=(history_size, 1)),
    tf.keras.layers.Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
```

[160]:
```
history = model.fit(
    train_dataset,
    epochs=100,
    steps_per_epoch=20,
    validation_data=test_dataset,
    validation_steps=3
)
```

```
Epoch 1/100
20/20 [==============================] - 3s 42ms/step - loss: 21.7799 -
val_loss: 14.8530
Epoch 2/100
20/20 [==============================] - 1s 30ms/step - loss: 12.4477 -
val_loss: 7.7351
Epoch 3/100
20/20 [==============================] - 0s 14ms/step - loss: 5.3771 - val_loss:
4.0230
```

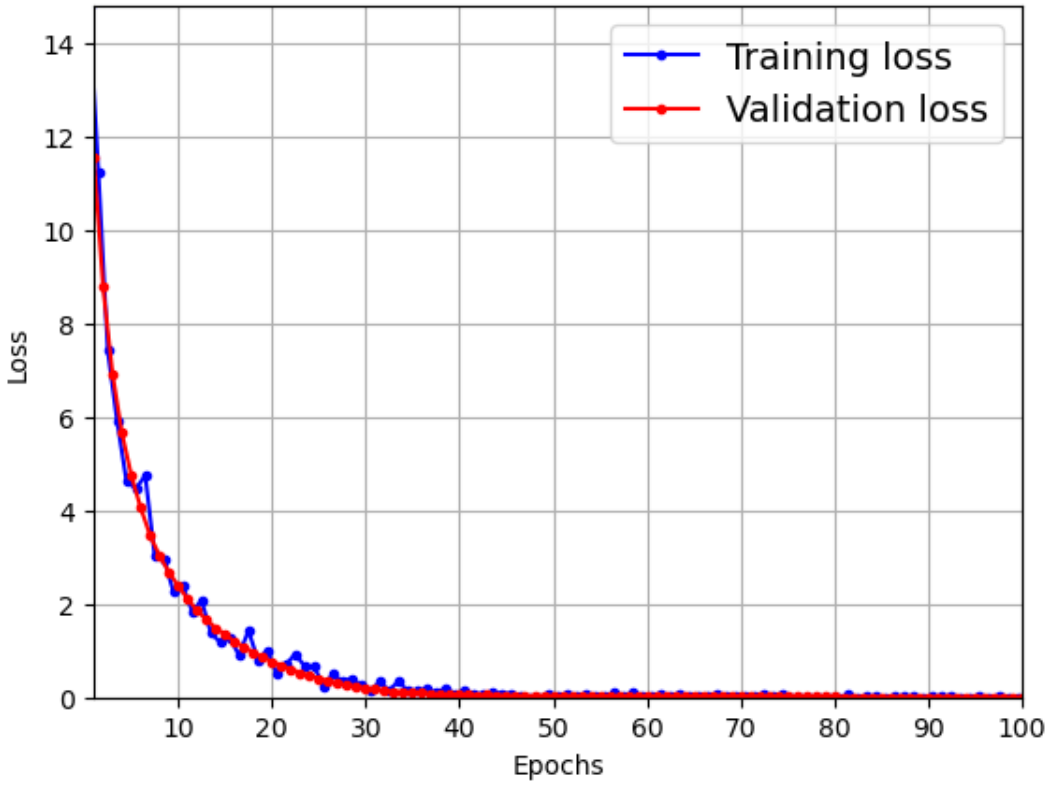
```
Epoch 4/100
20/20 [==============================] - 0s 15ms/step - loss: 3.2727 - val_loss:
2.4784
Epoch 5/100
20/20 [==============================] - 0s 13ms/step - loss: 2.1472 - val_loss:
1.7449
Epoch 6/100
20/20 [==============================] - 0s 15ms/step - loss: 1.7119 - val_loss:
1.1010
Epoch 7/100
20/20 [==============================] - 0s 14ms/step - loss: 1.3927 - val_loss:
0.5262
Epoch 8/100
20/20 [==============================] - 0s 25ms/step - loss: 0.4476 - val_loss:
0.3147
Epoch 9/100
20/20 [==============================] - 0s 22ms/step - loss: 0.3673 - val_loss:
0.2036
Epoch 10/100
20/20 [==============================] - 0s 20ms/step - loss: 0.3748 - val_loss:
0.1583
Epoch 11/100
20/20 [==============================] - 0s 18ms/step - loss: 0.2324 - val_loss:
0.1164
Epoch 12/100
20/20 [==============================] - 0s 19ms/step - loss: 0.1369 - val_loss:
0.0969
Epoch 13/100
20/20 [==============================] - 0s 18ms/step - loss: 0.1805 - val_loss:
0.0865
Epoch 14/100
20/20 [==============================] - 0s 17ms/step - loss: 0.0827 - val_loss:
0.0717
Epoch 15/100
20/20 [==============================] - 1s 32ms/step - loss: 0.0903 - val_loss:
0.0683
Epoch 16/100
20/20 [==============================] - 1s 31ms/step - loss: 0.1062 - val_loss:
0.0587
Epoch 17/100
20/20 [==============================] - 0s 21ms/step - loss: 0.0492 - val_loss:
0.0486
Epoch 18/100
20/20 [==============================] - 0s 18ms/step - loss: 0.1240 - val_loss:
0.0497
Epoch 19/100
20/20 [==============================] - 0s 17ms/step - loss: 0.0462 - val_loss:
0.0386
```

```
Epoch 100/100
20/20 [==============================] - 0s 12ms/step - loss: 0.0039 - val_loss:
0.0026
```

[97]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl

def plot_history(history):
    loss = history.history["loss"]
    val_loss = history.history["val_loss"]
    plt.plot(np.arange(len(loss)) + 0.5, loss, "b.-", label="Training loss")
    plt.plot(np.arange(len(val_loss)) + 1, val_loss, "r.-", label="Validation␣
 ↪loss")
    plt.gca().xaxis.set_major_locator(mpl.ticker.MaxNLocator(integer=True))
    plt.axis([1, 100, 0, max(max(loss), max(val_loss))])
    plt.legend(fontsize=14)
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.grid(True)
    plt.show()
```

[98]:
```
#print(history.history)
plot_history(history)
```



[132]:
```
predictions = model.predict(train_features)

print(predictions)
```

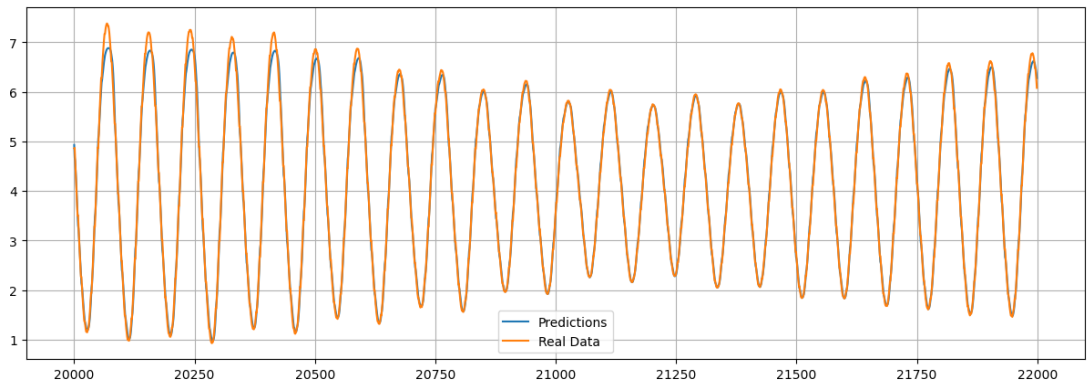
```
9881/9881 [==============================] - 58s 6ms/step
[[5.0789795]
 [4.9838033]
 [4.899341 ]
 …
 [5.0377192]
 [5.0130267]
 [4.924764 ]]
```

[158]:
```
begin = 20_000
end = 22_000
time = [i for i in range(end)]


fig = plt.figure(figsize=(15,5))

ax = fig.add_subplot(1, 1, 1)
ax.grid(True)

plt.plot(time[begin:end], predictions[begin:end], label = "Predictions")
plt.plot(time[begin:end], value_serie.values[begin+history_size*2:
  ↪end+history_size*2], label = "Real Data")
plt.legend()
plt.show()
```



[ ]:
```

```

Prédictions avec Scikit

Predictions

October 28, 2022

[2]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[3]:
```
%cd 'gdrive/MyDrive/Colab Notebooks/PIIA/References/predictions_types/'
!ls
```

```
/content/gdrive/MyDrive/Colab Notebooks/PIIA/References/predictions_types
dataset  Predictions.ipynb
```

[4]:
```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
```

[5]:
```
path = 'dataset/Infected.csv'
ds = pd.read_csv(path, index_col=0)
ds.head(18)
```

[5]:

Date	num_of_date	num_of_patients	Date_test	num_of_date_test
2020-01-21	1	1	2020-02-05	16.0
2020-01-22	2	1	2020-02-16	27.0
2020-01-23	3	1	2020-02-22	33.0
2020-01-24	4	2	2020-02-27	38.0
2020-01-25	5	2	2020-03-03	43.0
2020-01-26	6	5	2020-03-05	45.0
2020-01-27	7	5	2020-03-12	52.0
2020-01-28	8	5	2020-03-23	63.0
2020-01-29	9	5	2020-03-30	70.0

```
2020-01-30          10                5  2020-04-05              76.0
2020-01-31          11                7  2020-04-10              80.0
2020-02-01          12                8  2020-04-19              89.0
2020-02-02          13                8         NaN               NaN
2020-02-03          14               11         NaN               NaN
2020-02-04          15               11         NaN               NaN
2020-02-06          17               11         NaN               NaN
2020-02-07          18               11         NaN               NaN
2020-02-08          19               11         NaN               NaN
```

```
            num_of_patients_test
Date
2020-01-21                  11.0
2020-01-22                  13.0
2020-01-23                  15.0
2020-01-24                  58.0
2020-01-25                 118.0
2020-01-26                 217.0
2020-01-27                1663.0
2020-01-28               43847.0
2020-01-29              161807.0
2020-01-30              312237.0
2020-01-31              460252.0
2020-02-01              657996.0
2020-02-02                   NaN
2020-02-03                   NaN
2020-02-04                   NaN
2020-02-06                   NaN
2020-02-07                   NaN
2020-02-08                   NaN
```

[6]:
```
x = ds['num_of_date']
y = ds['num_of_patients']
x_test_patient = ds['num_of_date_test'][:12]  # data for the officiel test
y_test_patient = ds['num_of_patients_test'][:12]
```

[7]:
```
x
```

[7]:
```
Date
2020-01-21     1
2020-01-22     2
2020-01-23     3
2020-01-24     4
2020-01-25     5
              ..
2020-04-21    91
2020-04-22    92
```

```
2020-04-23    93
2020-04-24    94
2020-04-25    95
Name: num_of_date, Length: 83, dtype: int64
```

##Linear Regression

[8]:
```
yy = np.log10(y)

scores = []
```

[9]:
```
yy
```

[9]:
```
Date
2020-01-21    0.000000
2020-01-22    0.000000
2020-01-23    0.000000
2020-01-24    0.301030
2020-01-25    0.301030
                 …
2020-04-21    5.842530
2020-04-22    5.858897
2020-04-23    5.875226
2020-04-24    5.882302
2020-04-25    5.896655
Name: num_of_patients, Length: 83, dtype: float64
```

[10]:
```
Linear_Regression = LinearRegression()
```

[14]:
```
print(KFold.__doc__[:1070])
cv = KFold(n_splits=10, random_state=1, shuffle=True)  # helps to separate the␣
 ↪dataset in training and testing
separated = cv.split(x)
print(next(separated))
print(next(separated))
print(next(separated))
print(next(separated))
print(next(separated))
# we can do this 10 times
```

```
K-Folds cross-validator

    Provides train/test indices to split data in train/test sets. Split
    dataset into k consecutive folds (without shuffling by default).

    Each fold is then used once as a validation while the k - 1 remaining
    folds form the training set.
```

```
    Read more in the :ref:`User Guide <k_fold>`.

    Parameters
    ----------
    n_splits : int, default=5
        Number of folds. Must be at least 2.

        .. versionchanged:: 0.22
            ``n_splits`` default value changed from 3 to 5.

    shuffle : bool, default=False
        Whether to shuffle the data before splitting into batches.
        Note that the samples within each split will not be shuffled.

    random_state : int, RandomState instance or None, default=None
        When `shuffle` is True, `random_state` affects the ordering of the
        indices, which controls the randomness of each fold. Otherwise, this
        parameter has no effect.
        Pass an int for reproducible output across multiple function calls.
        See :term:`Glossary <random_state>`.

(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 41, 42, 43, 44, 45, 46, 47, 49, 50, 51, 52, 53,
       54, 55, 56, 58, 59, 61, 62, 63, 64, 67, 68, 69, 70, 71, 72, 74, 75,
       76, 77, 78, 79, 80, 81]), array([10, 40, 48, 57, 60, 65, 66, 73, 82]))
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 28, 29, 30, 32, 33, 34, 37,
       38, 39, 40, 41, 42, 43, 44, 45, 47, 48, 49, 50, 51, 52, 53, 54, 55,
       56, 57, 59, 60, 61, 62, 63, 64, 65, 66, 68, 69, 70, 71, 72, 73, 74,
       75, 78, 79, 80, 81, 82]), array([27, 31, 35, 36, 46, 58, 67, 76, 77]))
(array([ 0,  1,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 16, 17, 18,
       19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 35, 36, 37,
       38, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 53, 54, 55, 56, 57, 58,
       59, 60, 61, 62, 63, 64, 65, 66, 67, 69, 70, 71, 72, 73, 74, 75, 76,
       77, 78, 79, 80, 81, 82]), array([ 2, 15, 33, 34, 39, 47, 51, 52, 68]))
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 38, 39, 40, 41, 42, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
       55, 56, 57, 58, 59, 60, 61, 63, 64, 65, 66, 67, 68, 70, 71, 72, 73,
       74, 75, 76, 77, 79, 80, 82]), array([19, 26, 43, 44, 62, 69, 78, 81]))
(array([ 0,  1,  2,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
       36, 37, 39, 40, 41, 43, 44, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
       57, 58, 59, 60, 61, 62, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
       75, 76, 77, 78, 79, 81, 82]), array([ 3, 21, 38, 42, 45, 56, 63, 80]))
```

[15]:
```
for train_index, test_index in cv.split(x):
    X_train, X_test, y_train, y_test , yy_train, yy_test= x[train_index],␣
 ↪x[test_index], y[train_index], y[test_index], yy[train_index], yy[test_index]
    Linear_Regression.fit(X_train.values.reshape(-1,1), yy_train)
    scores.append(Linear_Regression.score(X_test.values.reshape(-1,1), yy_test))
```

[16]:
```
print("Average score for Linear Regression:", sum(scores) / len(scores))
```

```
Average score for Linear Regression: 0.9481866470449497
```

0.0.1 Evaluation model

[24]:
```
x.values.reshape(-1,1)[0], x.values.reshape(-1,1)[1]   # transpose matrix
```

[24]:
```
(array([1]), array([2]))
```

[29]:
```
x.values.reshape(-1,1)[-1]  # the last datetime is 95
```

[29]:
```
array([95])
```

[25]:
```
yy
```

[25]:
```
Date
2020-01-21    0.000000
2020-01-22    0.000000
2020-01-23    0.000000
2020-01-24    0.301030
2020-01-25    0.301030
                 …
2020-04-21    5.842530
2020-04-22    5.858897
2020-04-23    5.875226
2020-04-24    5.882302
2020-04-25    5.896655
Name: num_of_patients, Length: 83, dtype: float64
```

[27]:
```
#  Train the model with logaritmi values
Linear_Regression.fit(x.values.reshape(-1, 1), yy)  # use log data y
```

[ ]:
```
y_test_patient_log = np.log10(y_test_patient)  # calcule log of y_test oficial
evaluation_1 = Linear_Regression.predict(x_test_patient.values.reshape(-1, 1))
score = Linear_Regression.score(x_test_patient.values.reshape(-1, 1),␣
 ↪y_test_patient_log)
```

[36]:
```
print("Final Evaluation Score for Linear Regression :", score * 100, "%")
```

```
Final Evaluation Score for Linear Regression : 93.03292029481322 %
```

###Prediction for new days

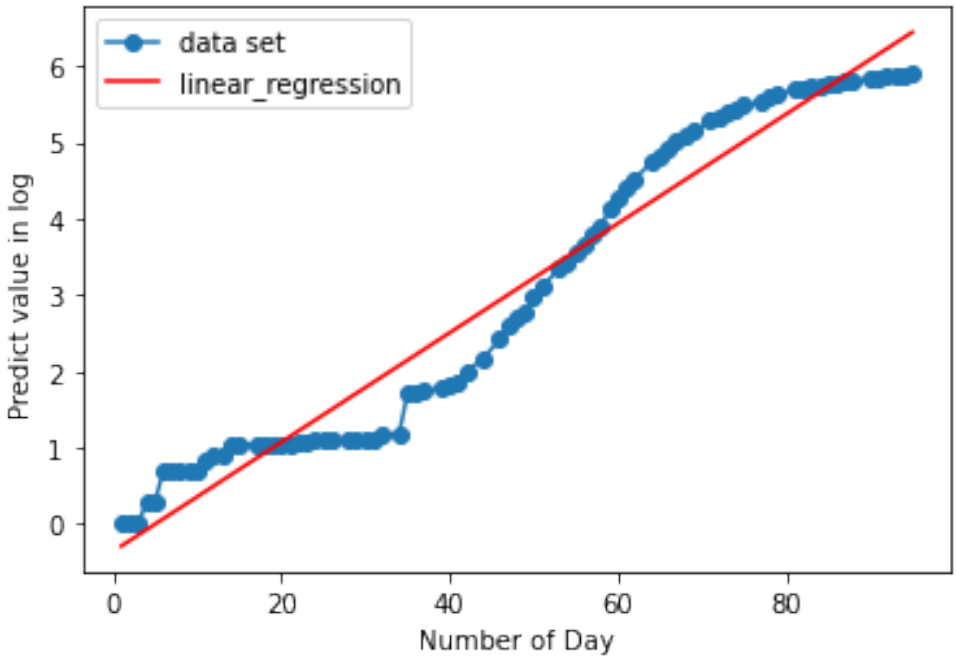
[30]:
```
# the last datetime is 95, so :
x_prediction = [[95],[96],[97],[98],[99],[100],[101]]  # matrix like a vector
```

[31]:
```
print('Evaluation for expecting 6 days in future in Linear_Regression:')
for predict in x_prediction:
  print('day', predict, '=', int(10 ** Linear_Regression.predict([predict])))
```

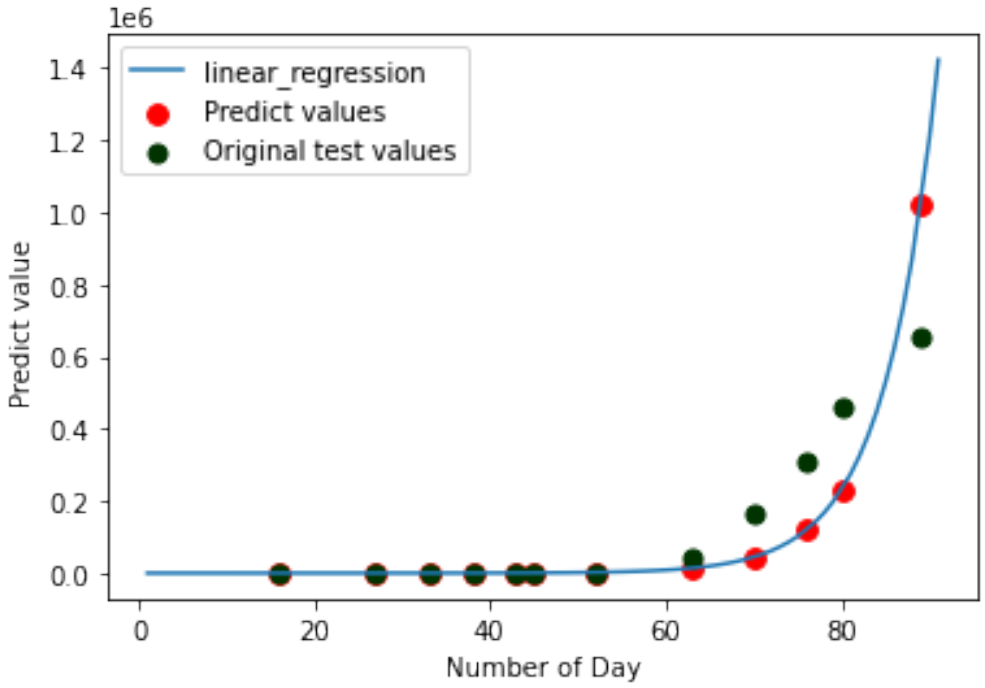
```
Evaluation for expecting 6 days in future in Linear_Regression:
day [95] = 2750230
day [96] = 3242796
day [97] = 3823580
day [98] = 4508382
day [99] = 5315832
day [100] = 6267896
day [101] = 7390475
```

###Plot the info

[37]:
```
predicted1 = Linear_Regression.predict(x.values.reshape(-1, 1))
predicted11 = 10 ** Linear_Regression.predict(x.values.reshape(-1,1))
plt.plot(x, yy, 'o-', label='data set')
plt.plot(x, predicted1, c='#ff0000', label='linear_regression')
plt.legend()
plt.xlabel('Number of Day')
plt.ylabel('Predict value in log ')
plt.show()
```



[34]:
```
plt.plot(x[:79], predicted11[:79],label='linear_regression')
plt.scatter(x_test_patient, 10 ** evaluation_1, s=60, c='#ff0000',␣
 ↪label='Predict values')
plt.scatter(x_test_patient, y_test_patient, s=50, c='#003300', label='Original␣
 ↪test values')
plt.legend()
plt.xlabel('Number of Day')
plt.ylabel('Predict value ')
plt.show()
```



[ ]: