Tran Triet
4A STI – Big Data

PROJET IA: Apprentissage machine pour mesurer la tendance d'évolution du Covid-19

Pays sélectionné:
Le travail utilise les données de Covid-19 de l'américaine (US).

Fichiers du code inclus:
US_Corona_train_test_prediction.ipynb : Utiliser le modèle SVR (85 % train, 15 % test) pour faire la prédiction de nombre de cas confirmés/décès/guéri.

US_Corona_SVR_KernelRidge.ipynb : Utiliser le modèle SVR et KernelRidge (100 % train) pour faire la prédiction de nombre de cas confirmés/décès/guéri dans 10 jours dans le future.

US_Corona_LSTM_Univariate.ipynb : Utiliser un modèle LSTM (85 % train, 15 % test) pour faire la prédiction de nombre de cas confirmés/décès/guéri dans 10 jours dans le future.

US_Corona_LSTM_Multivariate.ipynb:Utiliser un modèle LSTM avec l'ajout de paramètre de confinement pour faire la prédiction de nombre de cas confirmés/décès/guéri dans 10 jours dans le future.

Fichiers des données inclus :
time_series_covid19_confirmed_global.csv : Date et nombre de cas confirmés par pays.

time_series_covid19_deaths_global.csv : Date vs nombre de décès par jours

time_series_covid19_recovered_global.csv : Date et nombre de guéri par jours

Les données sont obtenues du dépôt Github de l'université de Johns Hopkins
(source)

(Note : Pour la question 4b, je pense que les données ont 2 problèmes que nous ne devons pas utiliser pour former notre modèle:
1 :Les dernières données datent de 2018 (alors que nous prédisons le virus Conora en 2020).
2 :Les données ne sont pas changées pendant un an. Par exemple, supposons que nous ayons le pourcentage de la population âgée de 65 ans et plus dans 2020, et que nous voulons mettre cette valeur dans une couche LSTM pour prédire la tendance du Covid-19 chaque jour, ces données ne seront pas utiles car il s'agit d'une valeur constante tandis que la couche LSTM besoin d'avoir une entrée comme valeur de time-steps: la valeur varie chaque jour.)

Code pour mettre à jour automatiquement les données dans les fichiers CSV :

```
#refresh data from github of university Johns Hopkins

base_data_path = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_
data_files = ["time_series_covid19_confirmed_global.csv", "time_series_covid19_deaths_global.csv", "time_series_covi

for file in data_files:
    if os.path.exists(file):
        os.remove(file)
    wget.download(os.path.join(base_data_path, file))
```

Les données sur Covid-19 sont mises à jour chaque jour, à partir du 22/1/2020.
Jusqu'à présent, il n'y avait qu'environ 90-> 100 points de données (chaque point pour chaque jour).
Comme nous le voyons, nous avons 2 problèmes ici:
1: nous aurons peu de données pour former notre modèle
2: les données des derniers jours (par centaines de milliers) sont plus importantes que les premiers jours (le nombre d'infections dans les premiers jours est très faible comme 1, 3, 4, 7, …)
Je vais donc séparer les données avec le ratio 85% pour le training et 15% pour le test.

On a
data_confirmed : Date et nombre de cas confirmés de l'US.
data_deaths : Date vs nombre de décès par jours de l'US.
data_recovered : Date et nombre de guéri par jours de l'US

Dans les 2 fichiers **US_Corona_train_test_prediction** et **US_Corona_SVR_KernelRidge**, le méthode pour séparer les données est **sklearn.model_selection.train_test_split**
L'attribut **shuffle = True** parce que la valeur de Y dépend uniquement de la valeur de x (les 2 modèle SVR et KernelRidge ne sont pas des modèles de réseau de neurones récurrents, donc l'ordre de valeur n'est pas important). L'utilisation de shuffle ici est une astuce pour éviter le overfitting.

```
#split data for train and test of confirmed, deaths and recovered
X_train_C, X_test_C, y_train_C, y_test_C = train_test_split(days, data_confirmed, test_size=0.15, shuffle=True)
X_train_D, X_test_D, y_train_D, y_test_D = train_test_split(days, data_deaths, test_size=0.15, shuffle=True)
X_train_R, X_test_R, y_train_R, y_test_R = train_test_split(days, data_recovered, test_size=0.15, shuffle=True)
```

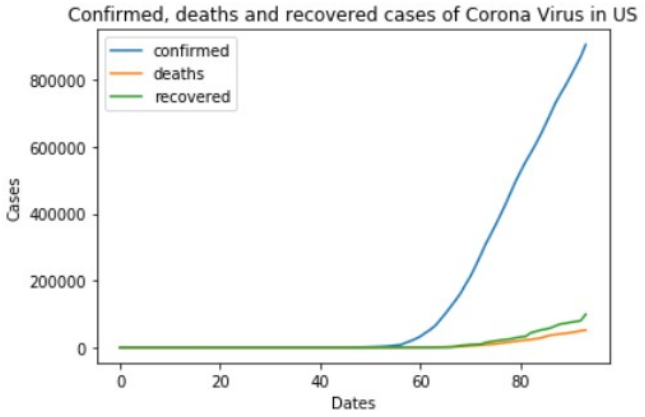
Dans les 2 fichiers **US_Corona_LSTM_Univariate** et **US_Corona_LSTM_Multivariate,** le modèle LSTM est un modèles de réseau de neurones récurrents, donc l'ordre de valeur est important. On va donc prendre les premiers 85% des données pour le training et 15 % derniers pour le test.

```
#percent split train test data
percent_split = 0.85

#get data
data_train_C = data_confirmed[:int(percent_split*len(data_confirmed))]
data_test_C = data_confirmed[int(percent_split*len(data_confirmed)):]

data_train_D = data_deaths[:int(percent_split*len(data_deaths))]
data_test_D = data_deaths[int(percent_split*len(data_deaths)):]

data_train_R = data_recovered[:int(percent_split*len(data_recovered))]
data_test_R = data_recovered[int(percent_split*len(data_recovered)):]
```

Un aperçu de l'évolution de Covid-19 donné par le graphique

```
#plot data: confirmed, deaths and recovered
plt.plot(days, data_confirmed, label='confirmed')
plt.plot(days, data_deaths, label='deaths')
plt.plot(days, data_recovered, label='recovered')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Confirmed, deaths and recovered cases of Corona Virus in US')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb812d2d6d0>
```



I. US_Corona_train_test_prediction

Déclarer les StandardScaler pour normaliser les données :

```
sc_X_C = StandardScaler()
sc_X_D = StandardScaler()
sc_X_R = StandardScaler()

sc_y_C = StandardScaler()
sc_y_D = StandardScaler()
sc_y_R = StandardScaler()
```

Normaliser les données :

```
#normalize splitted data
#normalize confirmed cases
X_train_C_transformed = sc_X_C.fit_transform(X_train_C)
X_test_C_transformed  = sc_X_C.transform(X_test_C)

y_train_C_transformed = sc_y_C.fit_transform(y_train_C)
y_test_C_transformed = sc_y_C.transform(y_test_C)

#normalize deaths cases
X_train_D_transformed = sc_X_D.fit_transform(X_train_D)
X_test_D_transformed  = sc_X_D.transform(X_test_D)

y_train_D_transformed = sc_y_D.fit_transform(y_train_D)
y_test_D_transformed = sc_y_D.transform(y_test_D)

#normalize recovered cases
X_train_R_transformed = sc_X_R.fit_transform(X_train_R)
X_test_R_transformed  = sc_X_R.transform(X_test_R)

y_train_R_transformed = sc_y_R.fit_transform(y_train_R)
y_test_R_transformed = sc_y_R.transform(y_test_R)
```

Créer les modèles SVR (sourceSVR)

On a les paramètre:

kernel : Spécifie le type de noyau à utiliser dans l'algorithme.

C : Paramètre de régularisation. La force de la régularisation est inversement proportionnelle à C.

epsilon : Il spécifie le tube epsilon dans lequel aucune pénalité n'est associée dans la fonction de perte d'entraînement avec des points prévus à une distance epsilon de la valeur réelle.

Après avoir essayé de nombreuses valeurs différentes, les valeurs kernel='poly', C=1.0, epsilon=0.1 m'ont donné un assez bon résultat

```
#Building SVR models for Confirmed, Deaths and Recovered Cases

clf_C = SVR(kernel='poly', C=1.0, epsilon=0.1)
clf_C.fit(X_train_C_transformed, y_train_C_transformed)

clf_D = SVR(kernel='poly', C=1.0, epsilon=0.1)
clf_D.fit(X_train_D_transformed, X_train_D_transformed)

clf_R = SVR(kernel='poly', C=1.0, epsilon=0.1)
clf_R.fit(X_train_R_transformed, X_train_R_transformed)
```

Utiliser le modèle pour prédire les données dans l'ensemble de test et apercevoir le résultat par le graphique.

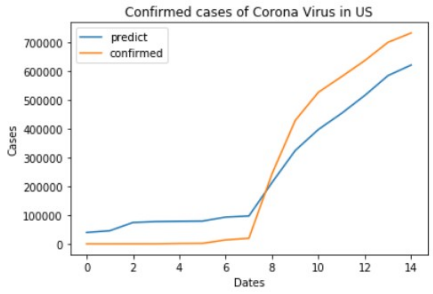
```
#Prediction for models
pred_C_transformed = clf_C.predict(X_test_C_transformed)
pred_C = sc_y_C.inverse_transform(pred_C_transformed)

pred_D_transformed = clf_D.predict(X_test_D_transformed)
pred_D = sc_y_D.inverse_transform(pred_D_transformed)

pred_R_transformed = clf_C.predict(X_test_R_transformed)
pred_R = sc_y_R.inverse_transform(pred_R_transformed)
```

```
#Sorted data in time order
p_C, r_C = zip(*sorted(zip(pred_C, y_test_C)))
p_D, r_D = zip(*sorted(zip(pred_D, y_test_D)))
p_R, r_R = zip(*sorted(zip(pred_R, y_test_R)))
```

Predict confirmed cases VS Real confirmed cases

```
#plot data: Confirmed cases
plt.plot(p_C, label='predict')
plt.plot(r_C, label='confirmed')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Confirmed cases of Corona Virus in US')
plt.legend()
```

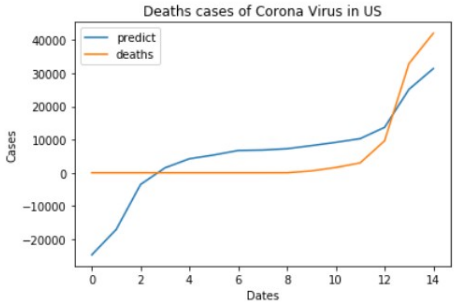
```
<matplotlib.legend.Legend at 0x7fb812c251d0>
```



Predict deaths cases VS Real deaths cases

```
#plot data: Deaths cases
plt.plot(p_D, label='predict')
plt.plot(r_D, label='deaths')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Deaths cases of Corona Virus in US')
plt.legend()
```

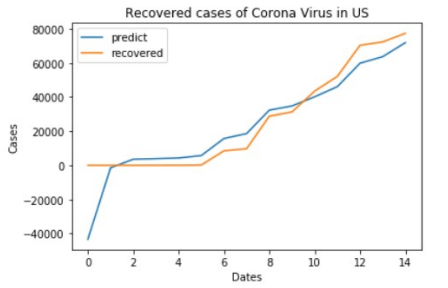
```
<matplotlib.legend.Legend at 0x7fb812baafd0>
```



Predict recovered cases VS Real recovered cases

```
#plot data: Recovered cases
plt.plot(p_R, label='predict')
plt.plot(r_R, label='recovered')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Recovered cases of Corona Virus in US')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fb812b30b90>
```



On voit que les résultats sont assez bien, sauf qu'il y a quelques valeurs négatives pour les premiers jours (qui devraient toujours être positives)

II. US_Corona_SVR_KernelRidge

Récupérer le nombre de jour et ajouter 10 jours pour prédire l'évolution de Covid-19 dans le future.

```
#get number of days and create an array of days
nb_days = len(data_confirmed)
days = np.arange(nb_days).reshape(-1,1)

#suppose we want to predict number of corona cases in the next 10 days
nb_days_future = 10
days_include_future = np.arange(nb_days + nb_days_future).reshape(-1,1)
```

Déclarer les StandardScaler pour normaliser les données :

```
sc_X = StandardScaler()
sc_y_confirmed = StandardScaler() #StandardScaler for confirmed cases
sc_y_deaths = StandardScaler() #StandardScaler for deaths cases
sc_y_recovered = StandardScaler() #StandardScaler for recovered cases
```

Normaliser les données :

```
days_transformed = sc_X.fit_transform(days)
days_include_future_transformed = sc_X.transform(days_include_future)

data_confirmed_transformed = sc_y_confirmed.fit_transform(data_confirmed)
data_deaths_transformed = sc_y_deaths.fit_transform(data_deaths)
data_recovered_transformed = sc_y_recovered.fit_transform(data_recovered)
```

Créer les modèles SVR (sourceSVR) et KernelRidge (sourceKernelRidge) et faire la prédiction de l'évolution de Covid-19

SVR model

source

```
#Create SVR model for Confirmed Death and Recovered
clf_C = SVR(kernel='poly', C=1.0, epsilon=0.1)
clf_C.fit(days_transformed, data_confirmed_transformed)

clf_D = SVR(kernel='poly', C=1.0, epsilon=0.1)
clf_D.fit(days_transformed, data_deaths_transformed)

clf_R = SVR(kernel='poly', C=1.0, epsilon=0.1)
clf_R.fit(days_transformed, data_recovered_transformed)
```

```
#Prediction for models
pred_C_transformed = clf_C.predict(days_include_future_transformed)
pred_C = sc_y_confirmed.inverse_transform(pred_C_transformed)
pred_C_clipped = np.clip(pred_C, a_min=0, a_max=None).astype(int)

pred_D_transformed = clf_D.predict(days_include_future_transformed)
pred_D = sc_y_deaths.inverse_transform(pred_D_transformed)
pred_D_clipped = np.clip(pred_D, a_min=0, a_max=None).astype(int)

pred_R_transformed = clf_C.predict(days_include_future_transformed)
pred_R = sc_y_recovered.inverse_transform(pred_R_transformed)
pred_R_clipped = np.clip(pred_R, a_min=0, a_max=None).astype(int)
```

KernelRidge model

Source

```
#Create SVR model for Confirmed Death and Recovered
clf_C = KernelRidge(kernel='poly', alpha=0.1)
clf_C.fit(days_transformed, data_confirmed_transformed)

clf_D = KernelRidge(kernel='poly', alpha=0.1)
clf_D.fit(days_transformed, data_deaths_transformed)

clf_R = KernelRidge(kernel='poly', alpha=0.1)
clf_R.fit(days_transformed, data_recovered_transformed)
```

```
#Prediction for models
pred_C_transformed = clf_C.predict(days_include_future_transformed)
pred_C = sc_y_confirmed.inverse_transform(pred_C_transformed)
pred_C_clipped = np.clip(pred_C, a_min=0, a_max=None).astype(int)

pred_D_transformed = clf_D.predict(days_include_future_transformed)
pred_D = sc_y_deaths.inverse_transform(pred_D_transformed)
pred_D_clipped = np.clip(pred_D, a_min=0, a_max=None).astype(int)

pred_R_transformed = clf_C.predict(days_include_future_transformed)
pred_R = sc_y_recovered.inverse_transform(pred_R_transformed)
pred_R_clipped = np.clip(pred_R, a_min=0, a_max=None).astype(int)
```

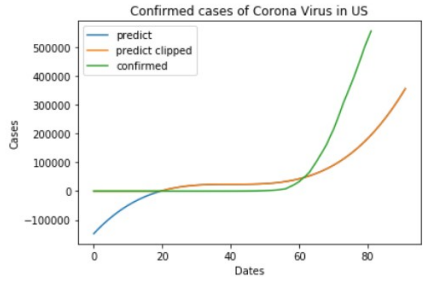
Apercevoir le résultat par le graphique.

SVR

SVR model for Confirmed Cases

```
#plot data: Confirmed cases
plt.plot(pred_C, label='predict')
plt.plot(pred_C_clipped, label='predict clipped')
plt.plot(data_confirmed, label='confirmed')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Confirmed cases of Corona Virus in US')
plt.legend()
```

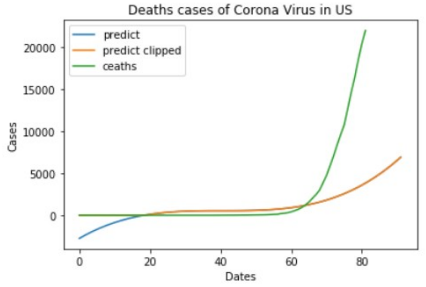
```
<matplotlib.legend.Legend at 0x7f8f0edbf610>
```



SVR model for Deaths Cases

```
#plot data: Confirmed cases
plt.plot(pred_D, label='predict')
plt.plot(pred_D_clipped, label='predict clipped')
plt.plot(data_deaths, label='ceaths')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Deaths cases of Corona Virus in US')
plt.legend()
```

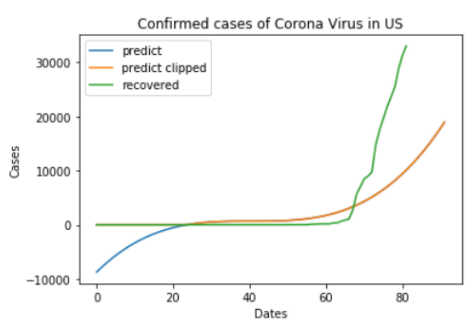
```
<matplotlib.legend.Legend at 0x7f8f0ed0f350>
```



SVR model for Recovered Cases

```
#plot data: Confirmed cases
plt.plot(pred_R, label='predict')
plt.plot(pred_R_clipped, label='predict clipped')
plt.plot(data_recovered, label='recovered')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Confirmed cases of Corona Virus in US')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8f0ec80890>
```

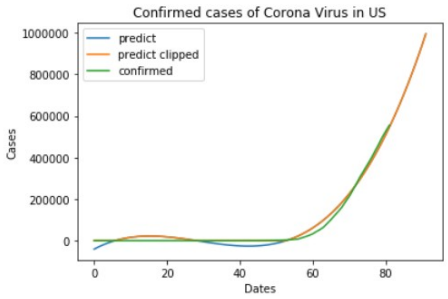


KernelRidge

KernelRidge model for Confirmed Cases

```
#plot data: Confirmed cases
plt.plot(pred_C, label='predict')
plt.plot(pred_C_clipped, label='predict clipped')
plt.plot(data_confirmed, label='confirmed')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Confirmed cases of Corona Virus in US')
plt.legend()
```

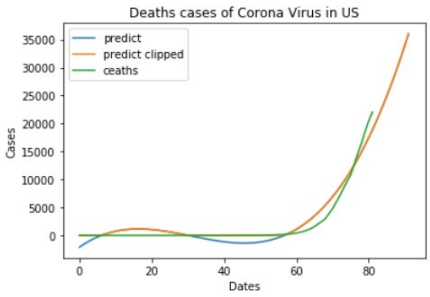
```
<matplotlib.legend.Legend at 0x7f8f0ec0cdd0>
```



KernelRidge model for Deaths Cases ¶

```
#plot data: Confirmed cases
plt.plot(pred_D, label='predict')
plt.plot(pred_D_clipped, label='predict clipped')
plt.plot(data_deaths, label='ceaths')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Deaths cases of Corona Virus in US')
plt.legend()
```

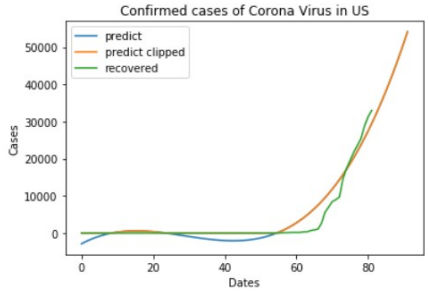
```
<matplotlib.legend.Legend at 0x7f8f0ea53d90>
```



KernelRidge model for Recovered Cases

```
#plot data: Confirmed cases
plt.plot(pred_R, label='predict')
plt.plot(pred_R_clipped, label='predict clipped')
plt.plot(data_recovered, label='recovered')
plt.xlabel('Dates')
plt.ylabel('Cases')
plt.title('Confirmed cases of Corona Virus in US')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f8f0e9e9a50>
```



On peut voir que les performances du modèle KernelRidge sont meilleures que celles du modèle SVR, la ligne de prédiction du KernelRidge est plus lisse et plus exacte.

III. US_Corona_LSTM_Univariate

Le modèle LSTM utilisé est un modèle de réseau de neurones récurrents.
Le modèle LSTM apprendra une fonction qui mappe une séquence d'observations passées comme entrée à une observation de sortie. En tant que tel, la séquence d'observations doit être transformée en plusieurs exemples à partir desquels le LSTM peut apprendre.
Par exemple, considérons une séquence univariée donnée:
[10, 20, 30, 40, 50, 60, 70, 80, 90]

Nous pouvons diviser la séquence en plusieurs modèles d'entrée / sortie appelés échantillons, où trois time-steps sont utilisés comme entrée et un time-steps est utilisé comme sortie pour la prédiction en une étape qui est apprise.

X,	y
[10, 20, 30]	[40]
[20, 30, 40]	[50]
[30, 40, 50]	[60]

…

La fonction split_sequence() ci-dessous implémente ce comportement et divisera une séquence univariée donnée en plusieurs échantillons où chaque échantillon a un nombre spécifié de time-steps et la sortie est un time-steps.

```
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

On peut définir la longeur de chaque entré par le variable n_steps et le nombre d'attribut par le variable n_features.
Après avoir essayé de nombreuses valeurs différentes de n_steps, la valeur n_steps = 1 (signifie que la valeur Y d'un jour dépend uniquement de la valeur Y de la journée précédente) m'ont donné le meilleur résultat.

```
#Choose timesteps, features and number of days in future
n_steps = 1
n_features = 1
days_future = 10
```

Diviser les données du train et les données de test :

```
# split into samples
X_train_C, y_train_C = split_sequence(data_train_C, n_steps)
X_test_C, y_test_C = split_sequence(data_test_C, n_steps)
X_C, y_C = split_sequence(data_confirmed, n_steps)

X_train_D, y_train_D = split_sequence(data_train_D, n_steps)
X_test_D, y_test_D = split_sequence(data_test_D, n_steps)
X_D, y_D = split_sequence(data_deaths, n_steps)

X_train_R, y_train_R = split_sequence(data_train_R, n_steps)
X_test_R, y_test_R = split_sequence(data_test_R, n_steps)
X_R, y_R = split_sequence(data_recovered, n_steps)
```

Créer le modèle LSTM (sourceLSTM) avec la fonction build_model()

```
#build model
def build_model(nb_node):
    model = Sequential()
    model.add(LSTM(nb_node, activation='relu', input_shape=(n_steps, n_features)))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.summary()
    return model
```

Train le modèle LSTM avec les données de cas confirmés/décès/guéri

Confirmés

```
nb_node = 50
model_C = build_model(nb_node)
#compile model
model_C.compile(optimizer='adam', loss='mse')

# fit model
model_C.fit(X_train_C, y_train_C, epochs=200, verbose=0)
```

Décès

```
nb_node = 50
model_D = build_model(nb_node)
#compile model
model_D.compile(optimizer='adam', loss='mse')

# fit model
model_D.fit(X_train_D, y_train_D, epochs=200, verbose=0)
```

Guéri

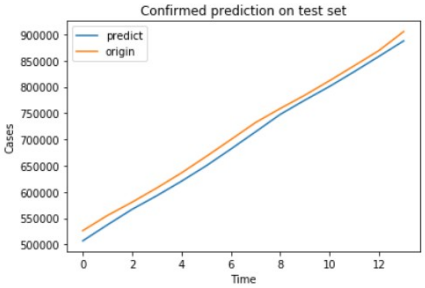
```
nb_node = 100
model_R = build_model(nb_node)
#compile model
model_R.compile(optimizer='adam', loss='mse')

# fit model
model_R.fit(X_train_R, y_train_R, epochs=200, verbose=0)
```

Évaluer la performace sur l’ensemble de test

Confirmés

```
y_test_pred_C = model_C.predict(X_test_C)
```

```
plt.plot(y_test_pred_C.reshape(-1,1), label='predict')
plt.plot(y_test_C, label='origin')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Cases')
plt.title('Confirmed prediction on test set')
plt.show()
```



Décès

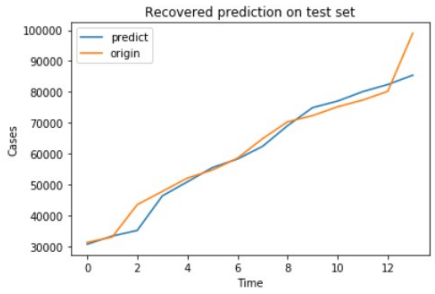
```
y_test_pred_D = model_D.predict(X_test_D)
```

```
plt.plot(y_test_pred_D.reshape(-1,1), label='predict')
plt.plot(y_test_D, label='origin')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Cases')
plt.title('Deaths prediction on test set')
plt.show()
```

Guéri

```
y_test_pred_R = model_R.predict(X_test_R)
```

```
plt.plot(y_test_pred_R.reshape(-1,1), label='predict')
plt.plot(y_test_R, label='origin')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Cases')
plt.title('Recovered prediction on test set')
plt.show()
```



Ce modèle fonctionne très bien, lorsqu'il est testé sur l'ensemble de test, l'erreur entre la prédiction et la réalité est très faible.

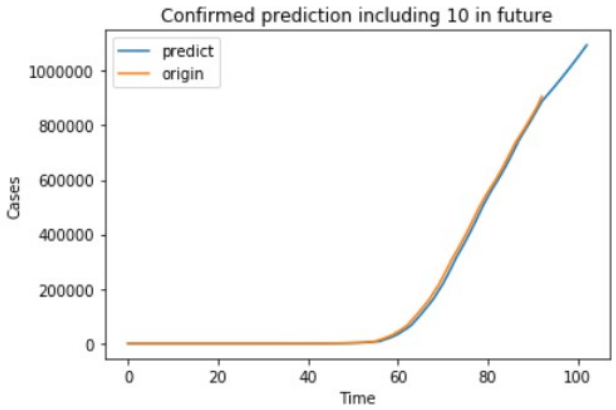
Prédire la tendance de Covid-19 dans les 10 prochains jours en utilisant la fonction predict_include_future() : après avoir prédit le nombre de cas d'un jour, nous utiliserons cette valeur prédite pour prédire le nombre de cas du jour suivant.

```
def predict_include_future(model, data, nb_days, n_steps):
    y_preds = model.predict(data)
    for i in range(nb_days):
        seq = y_preds[-n_steps:].reshape(1,n_steps,n_features)
        pred_day = model.predict(seq)
        y_preds = np.concatenate((y_preds, pred_day))
    return y_preds
```

Confirmés

```
y_pred_C = predict_include_future(model_C, X_C, days_future, n_steps)
```

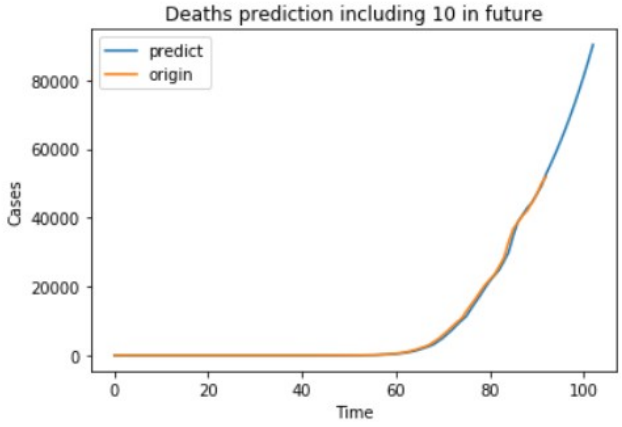
```
plt.plot(y_pred_C.reshape(-1,1), label='predict')
plt.plot(data_confirmed[n_steps:], label='origin')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Cases')
plt.title('Confirmed prediction including '+str(days_future)+' in future')
plt.show()
```



Décès

```
y_pred_D = predict_include_future(model_D, X_D, days_future, n_steps)
```

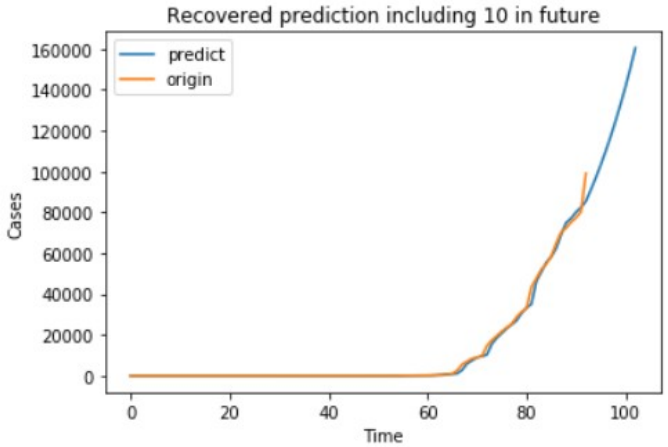
```
plt.plot(y_pred_D.reshape(-1,1), label='predict')
plt.plot(data_deaths[n_steps:], label='origin')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Cases')
plt.title('Deaths prediction including '+str(days_future)+' in future')
plt.show()
```



Guéri

```
y_pred_R = predict_include_future(model_R, X_R, days_future, n_steps)
```

```
plt.plot(y_pred_R.reshape(-1,1), label='predict')
plt.plot(data_recovered[n_steps:], label='origin')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Cases')
plt.title('Recovered prediction including '+str(days_future)+' in future')
plt.show()
```



La ligne de prédiction est assez lisse et cela semble une bonne prédiction qui nous aide à visualiser la tendance de Covid-19.

IV. US_Corona_LSTM_Multivariate

A côté du paramètre de nombre de cas mis à jour chaque jour, ajoutons un paramètre de confinement. La valeur 0 correspond à une journée normale et 1 à une journée de confinement.
Les états-unis commencent le confinement du 15/3/2020 au 30/4/2020, alors on va prédire la tendance de Covid-19 jusqu'au 5/5/2020 (jusqu'à 5 jours après le confinement).
Nous avons un array de confinement: la valeur des jours entre le 15/3/2020 et le 30/4/2020 est définie par 1 et les autres par 0 ([0,0,0,..,1,1...1,1,1,0,0,0,0,0])

La fonction split_sequence() ci-dessous implémente ce comportement et divisera une séquence multivariée donnée en plusieurs échantillons où chaque échantillon a un nombre spécifié de time-steps et la sortie est un time-steps.

```
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
        X.append((seq_x))
        y.append(seq_y)
    return array(X), array(y)
```

Créer le modèle LSTM (sourceLSTM) avec la fonction build_model()

```
#build model
def build_model(nb_node):
    model = Sequential()
    model.add(LSTM(nb_node, activation='relu', input_shape=(n_steps, n_features)))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.summary()
    return model
```

Train le modèle LSTM avec les données de cas confirmés

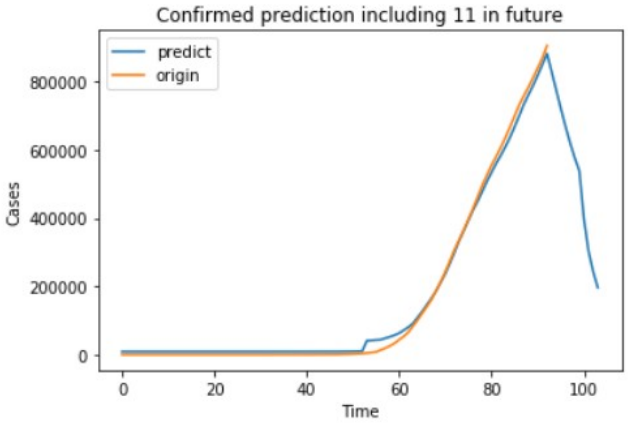
```
nb_node = 50
model_C = build_model(nb_node)
#compile model
model_C.compile(optimizer='adam', loss='mse')

# fit model
model_C.fit(X, y, epochs=200, verbose=0)
```

Prédire la tendance de Covid-19 dans les 10 prochains jours en utilisant la fonction predict_include_future() : après avoir prédit le nombre de cas d'un jour, nous utiliserons cette valeur prédite pour prédire le nombre de cas du jour suivant

```
def predict_include_future(model, data, nb_days, n_steps):
    k = len(data)
    y_preds = model.predict(data)
    for i in range(nb_days):
        data_1 = quarant_include_future[k + 1 + i - n_steps: k + 1 + i ].reshape(-1,1)
        data_2 = y_preds[-n_steps:]
        seq = hstack((data_1,data_2)).reshape(1,n_steps,n_features)
        pred_day = model.predict(seq)
        y_preds = np.concatenate((y_preds, pred_day))
    return y_preds
```

```
y_pred_C = y_pred_C_transformed*std[2]+mean[2]
```

```
plt.plot(y_pred_C.reshape(-1,1), label='predict')
plt.plot(data_confirmed[n_steps:], label='origin')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Cases')
plt.title('Confirmed prediction including '+str(days_future)+' in future')
plt.show()
```



Le résultat n'est pas comme prévu, car le nombre d'infections ne peut qu'augmenter mais pas diminuer. Parce qu'ici nous avons 2 propriétés différentes, j'ai donc standardisé les données avant mais le résultat est encore assez mauvais.

Conclusion

Avec ce type de données, les 2 modèles LSTM et KernelRige fonctionnent très bien, mieux que le modèle SVR.
Le modèle LSTM ne fonctionne bien si on ajoute le paramètre de confinement.