

TONG Ngoc Chung

4A STI – Big Data – INSA CVL

## PROJET IA

### Apprentissage machine pour mesurer la tendance d'évolution du Covid-19

## I, Introduction

### Pays sélectionné :

J'utilise les données de Covid-19 de la **France** (colonne **France** dans le DataFrame ).

### Fichiers des données utilisées :

*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 ([sources](#))

### Fichiers du code:

*Covid\_19\_prediction.ipynb* : Jupyter notebook pour analyser des données et les différences models de prédiction la tendance d'évolution du Covid-19 de la France.

Dans ce notebook, il y a 5 partie différences : **Data Analysis**, **Linear regression**, **Support vector machine regression(SVR)**, **Long Short-Term Memory network (LSTM) – univariate**, et **LSTM-Multivariate**.

## II, Analyse et prétraitement des données

Importer des données :

```
1 path_data = "https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/"
2 data_files = ["time_series_covid19_confirmed_global.csv", "time_series_covid19_deaths_global.csv", "time_series_covid19_recovered_global.csv"]
3
4 url_confirmed = os.path.join(path_data, data_files[0])
5 url_deaths = os.path.join(path_data, data_files[1])
6 url_recovered = os.path.join(path_data, data_files[2])
7
8 df_confirmed = pd.read_csv(url_confirmed)
9 df_deaths = pd.read_csv(url_deaths)
10 df_recovered = pd.read_csv(url_recovered)
11
```

*df\_confirmed* : Date et nombre de cas confirmés par jour global.

*df\_deaths* : Date vs nombre de décès par jours global.

*df\_recovered* : Date et nombre de guéri par jours global.

### Fonction pour le prétraitement des données :

```
1 def preprocessing_data(df):
2     # La latitude et la longitude ne contribuent pas à notre analyse
3     # nous allons donc supprimer ces 2 colonnes.
4     df.drop(['Lat', 'Long'], axis=1, inplace = True)
5
6     # nous devons donc fusionner les données
7     # pour obtenir le infection totale de chaque pays par jour.
8     df = df.groupby('Country/Region').sum()
9
10    df = df.transpose().reset_index().rename(columns={'index':'Date'})
11    df.rename_axis(None, axis=1,inplace=True)
12    df['Date'] = pd.to_datetime(df['Date'])
13    return df
14
```

La latitude et la longitude ne contribuent pas à notre analyse, nous allons donc supprimer ces 2 colonnes.

D'après la description de notre ensemble de données, nous pouvons voir que nous avons 284 lignes, mais le nombre de valeurs uniques de la colonne **Country/Region** est de 198 (puisque certains pays sont répartis entre leurs provinces), nous devons donc fusionner leurs données pour obtenir l'infection totale de chaque pays par jour.

### Fonction pour choisir le pays :

Ici, on choisit **la France**.

```
1 # C - confirmed; D - death; R - recovered
2 data_C, data_D, data_R = choose_country("France")
```

```
1 # Choose 1 country for prediction
2 def choose_country(name_country):
3     data_C = df_confirmed[['Date',name_country]]
4     data_C.rename(columns={name_country:"Confirmed"}, inplace =True)
5
6     data_D = df_deaths[['Date',name_country]]
7     data_D.rename(columns={name_country:"Deaths"}, inplace =True)
8
9     data_R = df_recovered[['Date',name_country]]
10    data_R.rename(columns={name_country:"Recovered"}, inplace =True)
11
12    return data_C, data_D, data_R
```

*data\_C* : Date et nombre de cas confirmés de la France.

*data\_D* : Date vs nombre de décès de la France.

*data\_R* : Date vs nombre de guéri de la France.

Préparer les ensembles de données d'apprentissage pour chaque série temporelle (85% pour le training et 15% pour le test )

J'utilise fonction **train\_test\_split** de **sklearn.model\_selection** pour diviser train et test set.

### III, Linear regression model

J'utilise polinomial regression ([sources](#)).

Après avoir essayé de nombreuses valeurs différentes, **degree = 6** est la meilleure valeur pour la prédiction.

J'utilise **fit\_transform** pour normaliser les données.

```
1 def linear_regression(X_train, X_test, y_train, y_test):
2     # poly features
3     pf = PolynomialFeatures(degree=6, include_bias=True)
4
5     # normalize splitted data
6     poly_X_train = pf.fit_transform(X_train)
7     poly_X_test = pf.fit_transform(X_test)
8     poly_future_pre = pf.fit_transform(days_start_to_futures)
9     #print(poly_X_train)
10
11    # fit a Linear Regression model
12    lin_regr = LinearRegression(normalize=True, fit_intercept=True)
13    lin_regr.fit(poly_X_train, y_train)
14
15    coef = lin_regr.coef_
16    print("coef: ", coef)
```

J'utilise l'erreur absolue moyenne (**MAE**) et l'erreur quadratique moyenne (**RMSE**) pour calculer le test l'erreur.

```
# Calcul error
mae = mean_absolute_error(y_pred, y_test)
mse = mean_squared_error(y_pred, y_test)
rmse = np.sqrt(mse)
print("RMSE of ", lin_regr.__class__.__name__, round(rmse, 1))
print("MAE of ", lin_regr.__class__.__name__, round(mae, 1), '\n')
```

Récupérer le nombre de jour et ajouter 15 jours pour prédire l'évolution de Covid-19 dans le future.

```
numbers_of_dates_confirmed = data_C.index.values.reshape(-1, 1)
numbers_of_dates_deaths = data_D.index.values.reshape(-1, 1)
numbers_of_dates_recovered = data_R.index.values.reshape(-1, 1)

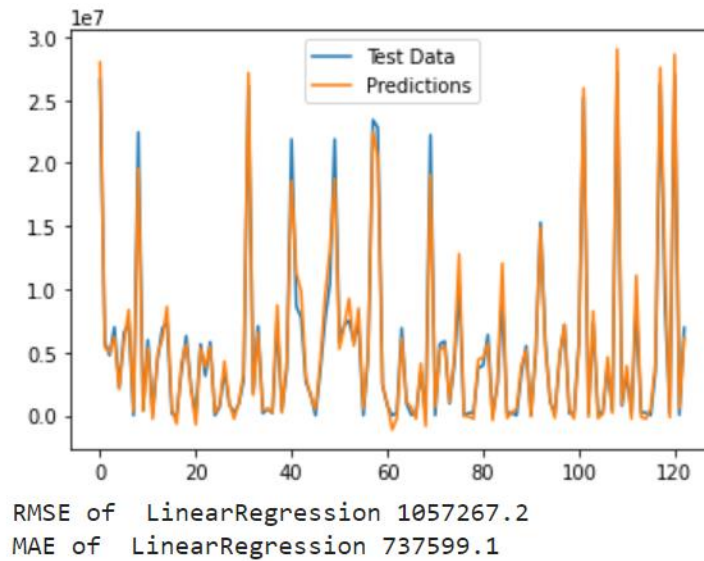
# Prediction Covid_confirmed cases for the next n days
nb_future_pre = 15 # Prediction for next 15 days
days_start_to_futures = np.array([i for i in range(data_C.shape[0] + nb_future_pre)]).reshape(-1, 1)
```

Variable **days\_start\_to\_futures** : tableau contenant les dates du début au 15ème jour dans le futur.

## Résultats :

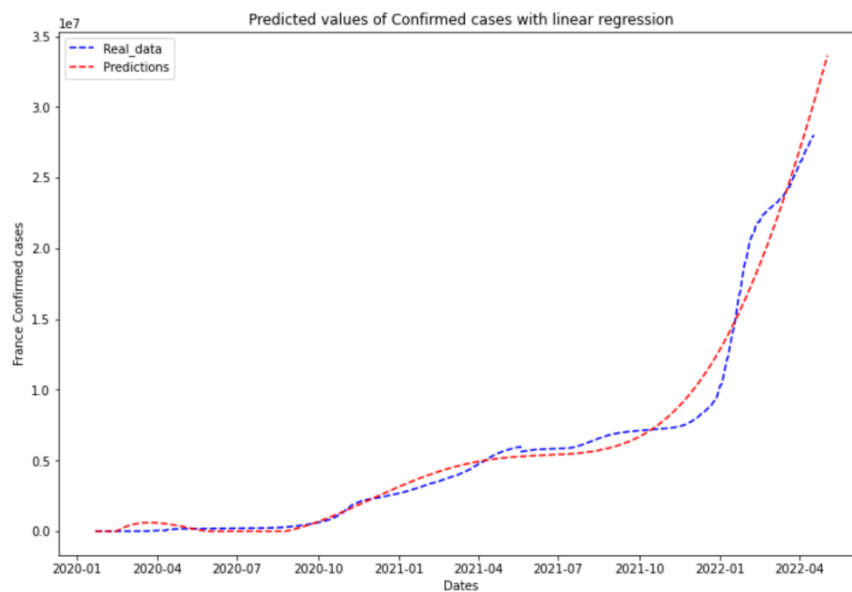
- Prédiction des cas confirmés.

Plot prédiction sur test set :

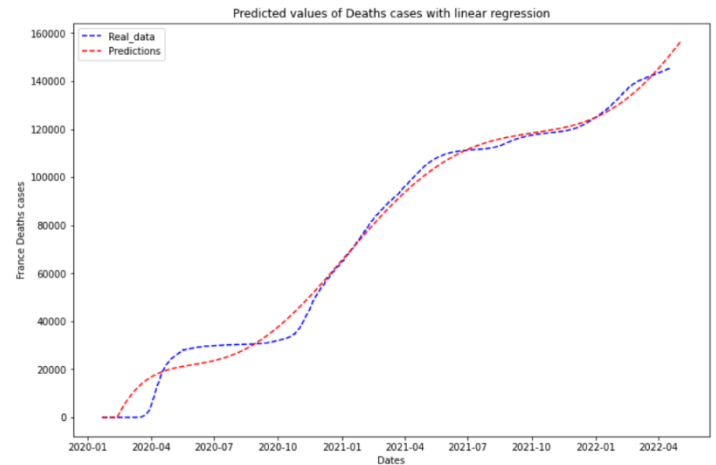
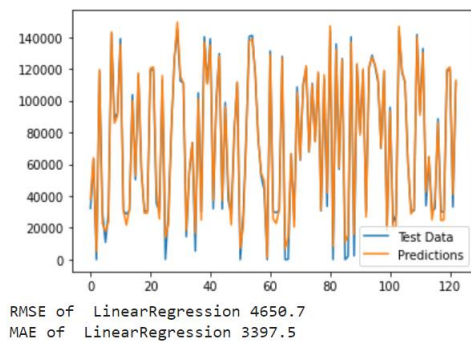


Bien que nous voyions de grandes erreurs, mais les données ont de grandes valeurs ( $10^7 - 10^7$ ) donc l'algorithme fonctionne bien.

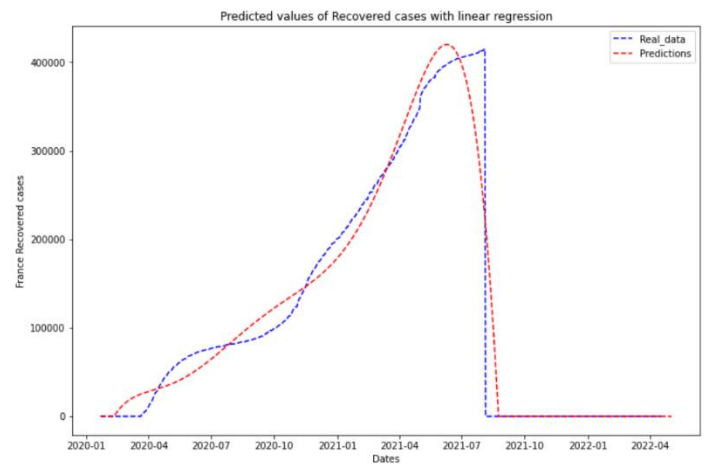
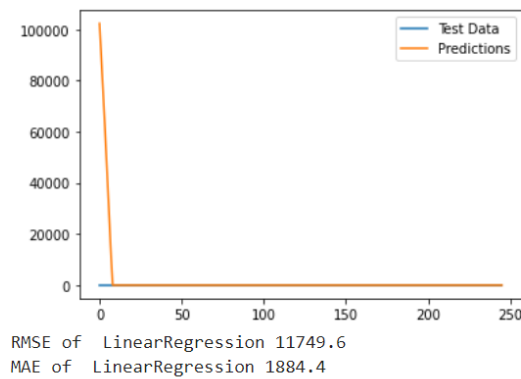
Plot prédiction sur tous les données et dans 15 jours au future:



- Prédiction des cas de décès.



- Prédiction des cas de guéri.



En effet, la prédiction a les valeurs négatives, mais les données doivent être positives, je donc ajoute la restriction des valeurs comme **ReLU** activation.

```
# all values of predictions must > 0
for i in range(len(poly_pred)):
    if poly_pred[i] < 0:
        poly_pred[i] = 0
```

On voit que les résultats sont bien sur 3 ensemble de données.

## IV, Support vector machine regression(SVR) model.

Créer le modèle SVR ([sourceSVR](#)).

Les paramètre:

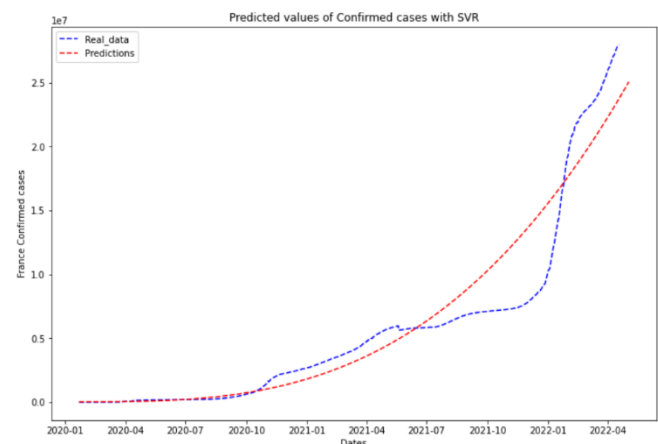
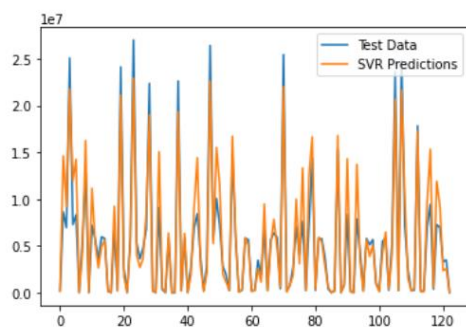
- **kernel** : Spécifie le type de noyau à utiliser dans l'algorithme.
- **C** : Paramètre de régularisation. La force de la régularisation est inversement proportionnelle à C.
- **epsilon** : Il spécifie le tube epsilon dans lequel aucune pénalité n'est associée dans la fonction de perte d'entraînement avec des points prévus à une distance epsilon de la valeur réelle.

```
1 def svm_model(X_train, y_train, X_test, y_test):
2     # SVM Model
3     svm_reg = SVR(kernel='poly', C=0.5, gamma=0.01, epsilon=0.01)
4     svm_reg.fit(X_train, y_train)
5
6     y_pred = svm_reg.predict(X_test)
7     svm_pred = svm_reg.predict(days_start_to_futures)
8     # Plot
9     plt.plot(y_test)
10    plt.plot(y_pred)
11    plt.legend(['Test Data', 'SVR Predictions'])
12
13    mae=mean_absolute_error(y_pred, y_test)
14    mse=mean_squared_error(y_pred, y_test)
15    rmse = np.sqrt(mse)
16    print("RMSE of ", svm_reg.__class__.__name__, round(rmse,1))
17    print("MAE of ", svm_reg.__class__.__name__, round(mae,1), '\n')
18
19    return svm_pred
```

### Résultats:

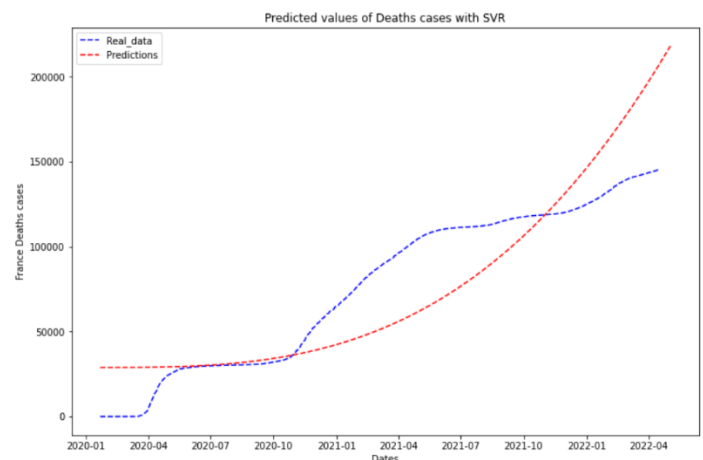
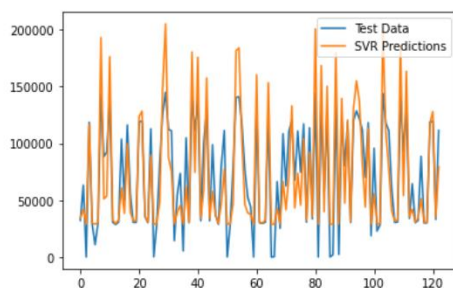
- Prédiction des cas confirmés

RMSE of SVR 2261141.1  
MAE of SVR 1407351.9



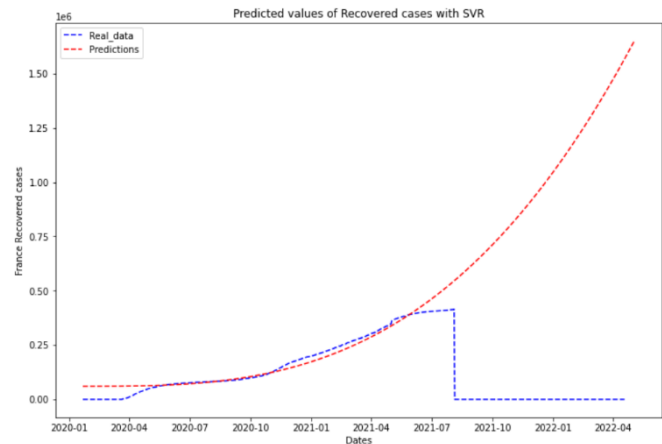
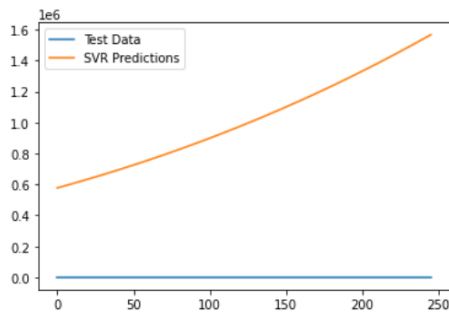
- Prédiction des cas de décès.

RMSE of SVR 25591.5  
MAE of SVR 20119.7



## - Prédiction des cas de guéri

RMSE of SVR 1054537.9  
MAE of SVR 1014781.1



On voit que le résultat est assez bien sur des cas confirmés, mais pas bien sur des cas de décès et en particulier pour des cas de guéri (en effet les données de guéri ont le problème, les values sont 0 à partir 2021-08-05)

## V, Long Short-Term Memory network (LSTM) – univariate

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: [1, 2, 3, 4, 5, 6, 7, 8, 9]

Nous pouvons diviser la séquence en plusieurs modèles d'entrée / sortie appelés échantillons, où time-steps = 2 sont utilisés (2 valeurs sont utilisées pour prédire la valeur suivante). Par exemple :

<b>x</b>	<b>y</b>
[1, 2]	[3]
[2, 3]	[4]
[3, 4]	[5]
...	...

### Définir les variables :

On peut définir la longueur de chaque entrée par le variable **nb\_steps** et le nombre d'attribut par le variable **nb\_features**.

```
1 #Choose timesteps, features and number of days in future, ...
2 nb_steps = 2
3 nb_features = 1
4 nb_future_pre = 30
5 nb_node = 150
6 percent_split = 2/3
```

**nb\_future\_pre** : nombre de jour dans l'avenir pour prédiction.

**nb\_node** : nombre de node dans le modèle LSMT.

## Diviser des données :

J'utilise **MinMaxScaler()** pour normaliser les données.

```
# Split data
normalizing = False
def split_data(data, nb_steps):
    if normalizing:
        # Normalizing data
        normal = MinMaxScaler()
        normal = normal.fit(data.reshape(-1,1))
        normal = normal.transform(data.reshape(-1,1))
    else : normal = data

    data_X, data_y = list(), list()
    #The batches would be [[t_1,...,t_{nb_steps}],predict=tt_{n_steps+1}].
    for i in range(len(normal)):
        if i + nb_steps >= len(normal):
            break
        seq_x, seq_y = normal[i: i+nb_steps], normal[i+nb_steps]
        data_X.append(seq_x)
        data_y.append(seq_y)

    return np.asarray(data_X), np.asarray(data_y)
```

## Créer le modèle LSMT ([source LSTM](#)) :

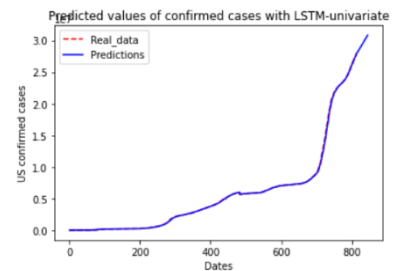
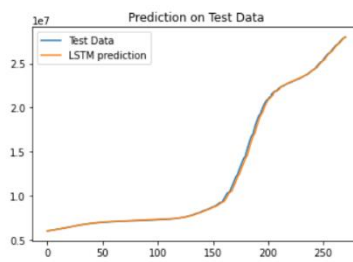
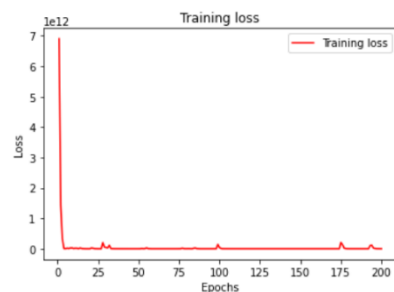
Model: "sequential\_11"

Layer (type)	Output Shape	Param #
lstm_11 (LSTM)	(None, 150)	91200
dense_22 (Dense)	(None, 75)	11325
dense_23 (Dense)	(None, 1)	76
Total params: 102,601		
Trainable params: 102,601		
Non-trainable params: 0		

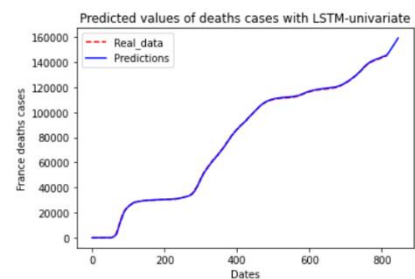
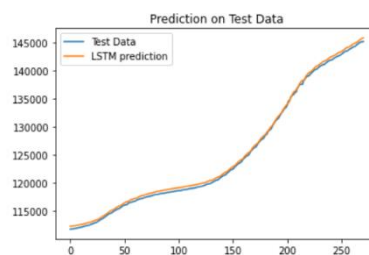
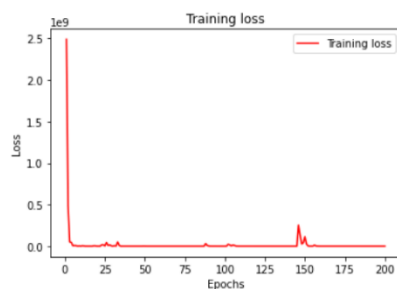
```
# Build model
def build_model(nb_node):
    model = Sequential()
    model.add(LSTM(nb_node, activation='relu', input_shape=(nb_steps, nb_features) ))
    model.add(Dense(75, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer="adam", loss="mse")
    model.summary()
    return model
```

## Résultats :

### - Prédiction des cas confirmés

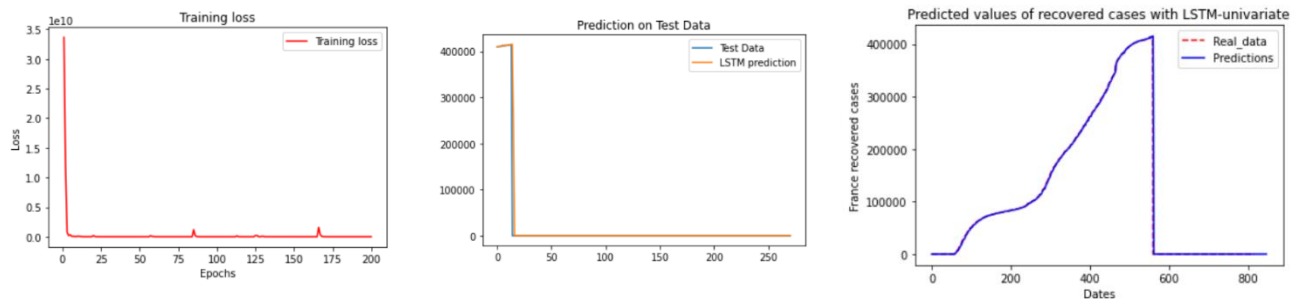


### - Prédiction des cas de décès





## - Prédiction des cas de guéri



On voit que avec nb\_steps = 2, le modèle fonctionne très bien pour les 3 ensembles des données.

Les erreurs sont très faible (approche à 0)

## VI, LSTM-Multivariate modèle

- Fusionner les données :

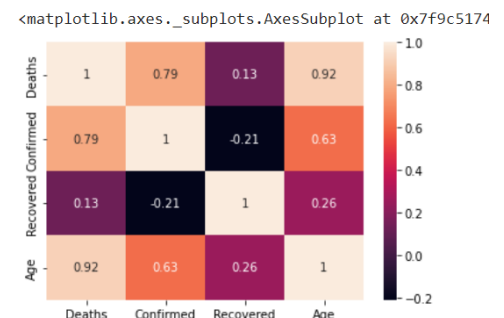
```
1 df_merge = pd.merge(data_C, data_D, how='left', on=['Date'])
2 df_merge = pd.merge(df_merge, data_R, how='left', on=['Date'])
3
4 df_merge
```

- On ajoutera colonne Age dans nos données. D'après les données d'Internet, on a population âgée de 65 ans et plus pour France (pourcentage) : 20,8 % pour 2020 et 21.1 % pour 2021, et on utilisera également 21.1 % pour 2022.

```
1 df_merge['Age'] = 20.8
2 df_merge.loc[df_merge['Date'].dt.year.isin([2021,2022]), 'Age'] = 21.1
3
```

- Correlation entre les features:

On voit que la corrélation entre colonne Deaths et Recovered est faible, alors pour prédire les cas de Décès, on va utiliser 3 colonnes Décès, Confirmé et Âge et on supprimera la colonne Guéri.



- Normaliser des données :

```
values = df_merge.values
# normalise data
scaler = MinMaxScaler()
scaled = scaler.fit_transform(values)
```

Fonction pour convertir les données sous forme supervisée :

```
def to_supervised(data, time_step = 2):
    df = pd.DataFrame(data)
    column = []
    column.append(df)
    for i in range(1, time_step+1):
        column.append(df.shift(-i))
    df = pd.concat(column, axis=1)
    df.dropna(inplace = True)
    nb_features = data.shape[1]
    data = df.values
    supervised_data = data[:, :nb_features*time_step]
    supervised_data = np.column_stack([supervised_data, data[:, nb_features*time_step]])
    return supervised_data
```

**Model LSTM :**

Il y a 3 layers (1 LSTM et 2 Dense)

```
1 model = build_model(nb_node = 150)
2 # fit model
3 history = model.fit(X_train, y_train, validation_data = (X_test, y_test), epochs=200, batch_size = 72, verbose=0)
```

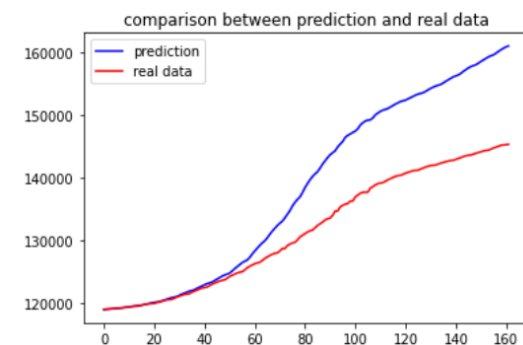
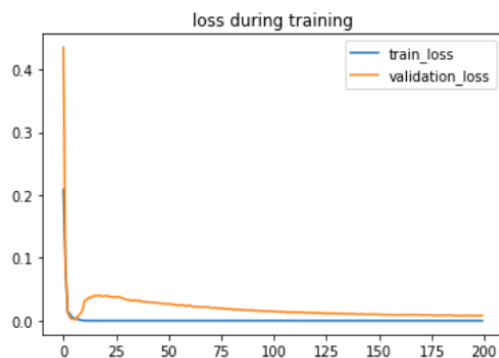
Model: "sequential\_25"

Layer (type)	Output Shape	Param #
lstm_25 (LSTM)	(None, 150)	92400
dense_50 (Dense)	(None, 75)	11325
dense_51 (Dense)	(None, 1)	76

=====  
Total params: 103,801  
Trainable params: 103,801  
Non-trainable params: 0

J'utilise 80% des données pour le training (**percent\_split = 0.8**).


**Résultats :**



Mean absolute error : 6723.766346848305  
Mean squared error : 77028791.93191461

- **Ajout du paramètre de confinement :**

On utilise les données de [https://en.wikipedia.org/wiki/COVID-19\\_lockdowns](https://en.wikipedia.org/wiki/COVID-19_lockdowns)

 France	Nationwide	2020-03-17 <sup>[252]</sup>	2020-05-11 <sup>[253]</sup>	55	2020-10-30 <sup>[254]</sup>	2020-12-15 <sup>[255]</sup>	46	2021-04-04 <sup>[256]</sup>	2021-05-03 <sup>[256]</sup>	29
	Paris							2021-03-19 <sup>[257]</sup>	2021-04-18	30

On a 3 fois de confinement.

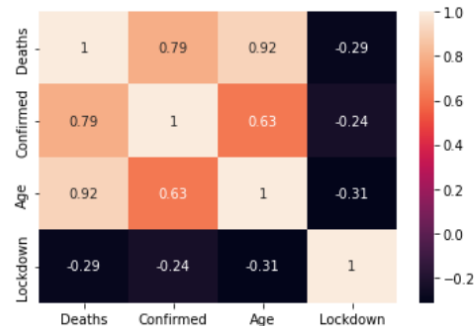
```
1 date_format = "%Y-%m-%d"
2
3 lockdown_1_start = dt.strptime('2020-03-17', date_format)
4 lockdown_1_end = dt.strptime('2020-05-11', date_format)
5
6 lockdown_2_start = dt.strptime('2020-10-30', date_format)
7 lockdown_2_end = dt.strptime('2020-12-15', date_format)
8
9 lockdown_3_start = dt.strptime('2021-04-04', date_format)
10 lockdown_3_end = dt.strptime('2021-05-03', date_format)
11
12 lockdown_1 = pd.date_range(start = lockdown_1_start, end = lockdown_1_end)
13 lockdown_2 = pd.date_range(start = lockdown_2_start, end = lockdown_2_end)
14 lockdown_3 = pd.date_range(start = lockdown_3_start, end = lockdown_3_end)
15
16 lockdown = lockdown_1.union(lockdown_2).union(lockdown_3)
```

J'ajoute la colonne **Lockdown** . La valeur 1 signifie ce jour n'appartient pas le confinement et la valeur 2 signifie l'appartenance.

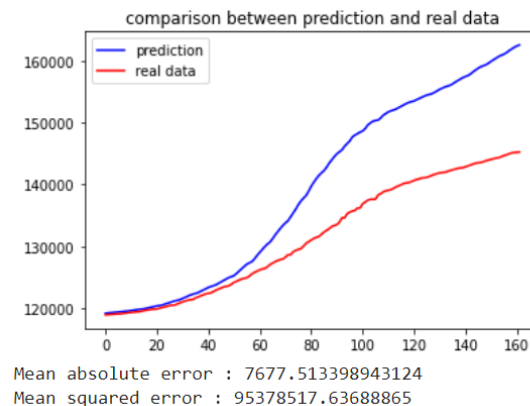
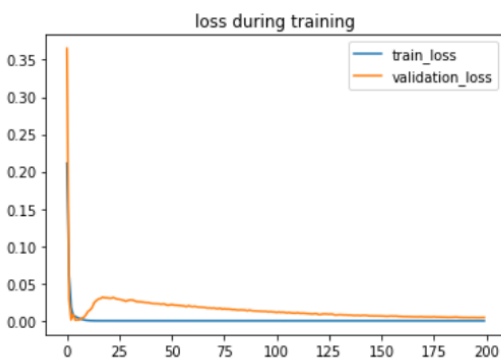
```
1 df_merge['Lockdown'] = 1
2
3 df_merge.loc[df_merge.index.isin(lockdown), 'Lockdown'] = 2
4
```

Nouvelle corrélation :

On voit que la corrélation entre colonne Deaths et Lockdown est très faible.



Résultats:



En comparant les erreurs (**MAE** et **RMSE**) de modèle avant et après l'ajout de paramètre **Confinement**, on voit que l'erreur après est plus grand qu'avant. La raison est la corrélation entre Deaths et Lockdown est faible.

## VII, Conclusion

Avec ce type de données, les 2 modèles **LSTM-univariate** et **Polynomial Regression** fonctionnent très bien, mieux que le modèle **SVR**.

Le modèle **LSTM-multivariate** fonctionne pas bien et si on ajoute le paramètre de confinement, l'erreur est plus grand parce que la corrélation est faible.