In [ ]:
```
import tensorflow as tf
import os
import pandas as pd
import numpy as np
from pandas import read_csv
```

In [ ]:
```
url = 'https://raw.githubusercontent.com/Jneny/Hospitalcapacity/main/Data/icu_beds.csv'
data = read_csv(url, header=0, parse_dates=[0], index_col=0)
data = data.asfreq('d')
adultcrit = pd.DataFrame(data, columns=['adult_icu_crci_patients'])
sadultcrit = pd.Series(adultcrit.adult_icu_crci_patients)
```

In [ ]:
```
len(adultcrit)
```

Out[ ]:
```
655
```

In [ ]:
```
adtrain = adultcrit.iloc[:571]
adtest = adultcrit.iloc[571:]
```

In [ ]:
```
#from statsmodels.tsa.seasonal import seasonal_decompose
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
#convert dataset to scale of 0 to 1/ normalize because do not want magnitude to get confused due to different r
```

In [ ]:
```
scaler.fit(adtrain)
scaled_train = scaler.transform(adtrain)
scaled_test = scaler.transform(adtest)
```

In [ ]:
```
from keras.preprocessing.sequence import TimeseriesGenerator
```

In [ ]:
```
#Using 7days/ 1 week as input to predict the next day
```

In [ ]:
```
#Define generator, feeding input as one week then predict next day (input 7 days for 8th day output)
#feeding values to NN to create batches to predict next time range
n_input = 7
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input, batch_size =1)
```

In [ ]:
```
X,y = generator[0]
print(f'Given the Arrary: \n{X.flatten()}')
print(f'Predict this y: \n {y}')
```

```
Given the Arrary:
[0.25947187 0.25028703 0.26176808 0.25832377 0.25832377 0.25487945
 0.24799082]
Predict this y:
 [[0.24339839]]
```

In [ ]:
```
X.shape
```

Out[ ]:
```
(1, 7, 1)
```

In [ ]:
```
from keras.models import Sequential
from keras.layers.core import Dense
from keras.layers import LSTM
```

In [ ]:
```
# Define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_input, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss = 'mse')
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 100)               40800

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
# from tensorflow.keras.callbacks import EarlyStopping
# early_stopping = EarlyStopping()
```

In [ ]:
```
#fit model
model.fit(generator, epochs=20)

# es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)
# model.fit(generator, epochs=30, callbacks=[early_stopping], )
```

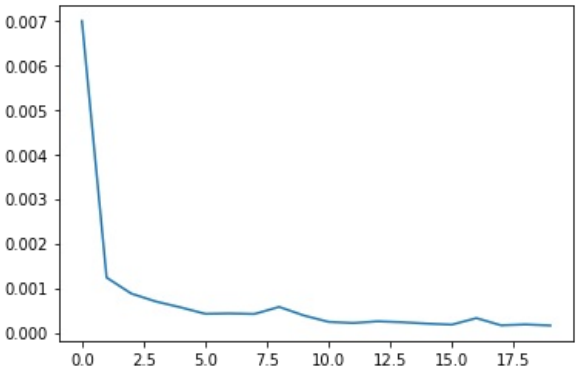
```
Epoch 1/20
564/564 [==============================] - 3s 4ms/step - loss: 0.0070
Epoch 2/20
564/564 [==============================] - 2s 4ms/step - loss: 0.0012
Epoch 3/20
564/564 [==============================] - 2s 4ms/step - loss: 8.7577e-04
Epoch 4/20
564/564 [==============================] - 2s 4ms/step - loss: 6.9725e-04
Epoch 5/20
564/564 [==============================] - 2s 4ms/step - loss: 5.6630e-04
Epoch 6/20
564/564 [==============================] - 2s 4ms/step - loss: 4.2179e-04
Epoch 7/20
564/564 [==============================] - 2s 4ms/step - loss: 4.2942e-04
Epoch 8/20
564/564 [==============================] - 2s 4ms/step - loss: 4.1827e-04
Epoch 9/20
564/564 [==============================] - 2s 4ms/step - loss: 5.7661e-04
Epoch 10/20
564/564 [==============================] - 2s 4ms/step - loss: 3.8442e-04
Epoch 11/20
564/564 [==============================] - 2s 4ms/step - loss: 2.4140e-04
Epoch 12/20
564/564 [==============================] - 2s 4ms/step - loss: 2.1516e-04
Epoch 13/20
564/564 [==============================] - 2s 4ms/step - loss: 2.5345e-04
Epoch 14/20
564/564 [==============================] - 2s 4ms/step - loss: 2.3291e-04
Epoch 15/20
564/564 [==============================] - 2s 4ms/step - loss: 2.0184e-04
Epoch 16/20
564/564 [==============================] - 2s 4ms/step - loss: 1.7882e-04
Epoch 17/20
564/564 [==============================] - 2s 4ms/step - loss: 3.2521e-04
Epoch 18/20
564/564 [==============================] - 2s 4ms/step - loss: 1.6406e-04
Epoch 19/20
564/564 [==============================] - 2s 4ms/step - loss: 1.8370e-04
Epoch 20/20
564/564 [==============================] - 2s 4ms/step - loss: 1.5981e-04
```

Out[ ]:
```
<keras.callbacks.History at 0x7f84b7999e50>
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
#plot loss to see when is good to stop epochs
loss_per_epoch = model.history.history['loss']
plt.plot(range(len(loss_per_epoch)), loss_per_epoch)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f84b763c350>]
```



In [ ]:
```
#formatting input so that using the last 30 of train to start prediction on first day of test set which is the
# 30 doesn't work take last 7 days of train to make prediction on first day of test set
last_train_batch = scaled_train[-7:]
```

In [ ]:
```
last_train_batch = last_train_batch.reshape((1, n_input, n_features))
```

In [ ]:
```
model.predict(last_train_batch) # check first prediction of test vs actual
```

Out[ ]:
```
array([[0.12857565]], dtype=float32)
```

In [ ]:
```
scaled_test[0]
```

Out[ ]:
```
array([0.13203215])
```

In [ ]:
```
test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))

for i in range(len(adtest)):

    # get the prediction value for the first batch
    current_pred = model.predict(current_batch)[0]

    # append the prediction into the array
    test_predictions.append(current_pred)

    # use the prediction to update the batch and remove the first value
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

In [ ]:
```
true_predictions = scaler.inverse_transform(test_predictions)
```

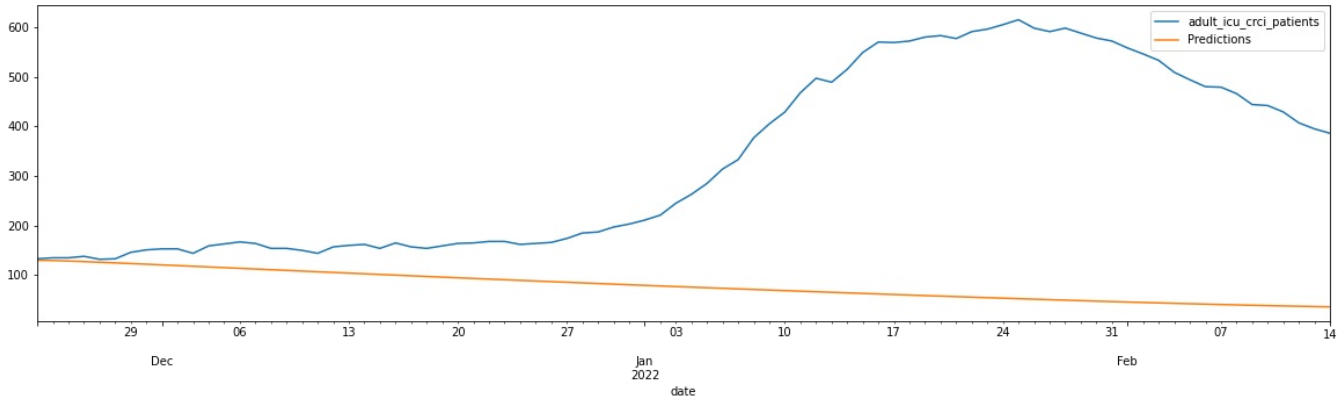
In [ ]:
```
adtest['Predictions'] = true_predictions
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [ ]:
```
adtest.plot(figsize=(20,5))
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f84b75760d0>
```



In [ ]:
```
# calculate root mean squared error
from sklearn.metrics import mean_squared_error
import math
from math import sqrt
rmse= math.sqrt(mean_squared_error(adtest['adult_icu_crci_patients'], adtest['Predictions']))
print(rmse)
```

```
324.70117816300467
```

In [ ]:
```
# calculate mean absolute percentage error MAPE
def mape(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.mean(np.abs((actual - pred) / actual)) * 100
mape(adtest['adult_icu_crci_patients'], adtest['Predictions'])
```

Out[ ]:
```
61.30077977561895
```

In [ ]: