In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pandas import read_csv
import math
```

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 [ ]:
```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

In [ ]:
```
#Convert pandas dataframe to numpy array
dataset = adultcrit.values
# dataset.dtype
dataset = dataset.astype('float32') #Convert values to float
```

In [ ]:
```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1)) #Also try QuantileTransformer
dataset = scaler.fit_transform(dataset)
```

In [ ]:
```
# len(dataset) = 655, want to predict last month or last 4 weeks = 655- (7*4) = 627
train, test = dataset[:571], dataset[571:]
```

In [ ]:
```
def to_sequences(dataset, seq_size=1):
    x = []
    y = []

    for i in range(len(dataset)-seq_size-1):
        #print(i)
        window = dataset[i:(i+seq_size), 0]
        x.append(window)
        y.append(dataset[i+seq_size, 0])

    return np.array(x),np.array(y)
```

In [ ]:
```
#input size first week then predict the next
seq_size = 7
#Larger sequences (look further back) may improve forecasting.
trainX, trainY = to_sequences(train, seq_size)
testX, testY = to_sequences(test, seq_size)
```

In [ ]:
```
print("Shape of training set: {}".format(trainX.shape))
print("Shape of test set: {}".format(testX.shape))
```

```
Shape of training set: (563, 7)
Shape of test set: (76, 7)
```

In [ ]:
```
print('Build deep model...')
# create and fit dense model
model = Sequential()
model.add(Dense(64, input_dim=seq_size, activation='relu')) #12
model.add(Dense(32, activation='relu'))  #8
model.add(Dense(1)) # y value
model.compile(loss='mean_squared_error', optimizer='adam', metrics = ['acc'])
print(model.summary())
```

```
Build deep model...
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                512

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 2,625
Trainable params: 2,625
Non-trainable params: 0
_________________________________________________________________
None
```

In [ ]:
```
model.fit(trainX, trainY, validation_data=(testX, testY),
          verbose=2, epochs=20)
```

```
Epoch 1/20
18/18 - 2s - loss: 0.1422 - acc: 0.0018 - val_loss: 0.0379 - val_acc: 0.0000e+00 - 2s/epoch - 97ms/step
Epoch 2/20
18/18 - 0s - loss: 0.0079 - acc: 0.0036 - val_loss: 0.0038 - val_acc: 0.0000e+00 - 109ms/epoch - 6ms/step
Epoch 3/20
18/18 - 0s - loss: 0.0013 - acc: 0.0036 - val_loss: 0.0037 - val_acc: 0.0000e+00 - 100ms/epoch - 6ms/step
Epoch 4/20
18/18 - 0s - loss: 0.0010 - acc: 0.0036 - val_loss: 0.0033 - val_acc: 0.0000e+00 - 148ms/epoch - 8ms/step
Epoch 5/20
18/18 - 0s - loss: 8.5141e-04 - acc: 0.0036 - val_loss: 0.0030 - val_acc: 0.0000e+00 - 118ms/epoch - 7ms/step
Epoch 6/20
18/18 - 0s - loss: 7.5946e-04 - acc: 0.0036 - val_loss: 0.0029 - val_acc: 0.0000e+00 - 140ms/epoch - 8ms/step
Epoch 7/20
18/18 - 0s - loss: 6.6962e-04 - acc: 0.0036 - val_loss: 0.0026 - val_acc: 0.0000e+00 - 151ms/epoch - 8ms/step
Epoch 8/20
18/18 - 0s - loss: 5.8423e-04 - acc: 0.0036 - val_loss: 0.0021 - val_acc: 0.0000e+00 - 134ms/epoch - 7ms/step
Epoch 9/20
18/18 - 0s - loss: 5.1912e-04 - acc: 0.0036 - val_loss: 0.0019 - val_acc: 0.0000e+00 - 112ms/epoch - 6ms/step
Epoch 10/20
18/18 - 0s - loss: 4.4694e-04 - acc: 0.0036 - val_loss: 0.0017 - val_acc: 0.0000e+00 - 109ms/epoch - 6ms/step
Epoch 11/20
18/18 - 0s - loss: 3.8704e-04 - acc: 0.0036 - val_loss: 0.0015 - val_acc: 0.0000e+00 - 103ms/epoch - 6ms/step
Epoch 12/20
18/18 - 0s - loss: 3.4295e-04 - acc: 0.0036 - val_loss: 0.0015 - val_acc: 0.0000e+00 - 149ms/epoch - 8ms/step
Epoch 13/20
18/18 - 0s - loss: 3.3327e-04 - acc: 0.0036 - val_loss: 0.0011 - val_acc: 0.0000e+00 - 153ms/epoch - 8ms/step
Epoch 14/20
18/18 - 0s - loss: 3.0527e-04 - acc: 0.0036 - val_loss: 9.3465e-04 - val_acc: 0.0000e+00 - 115ms/epoch - 6ms/st
ep
Epoch 15/20
18/18 - 0s - loss: 2.5325e-04 - acc: 0.0036 - val_loss: 8.3894e-04 - val_acc: 0.0000e+00 - 112ms/epoch - 6ms/st
ep
Epoch 16/20
18/18 - 0s - loss: 2.1427e-04 - acc: 0.0036 - val_loss: 7.3255e-04 - val_acc: 0.0000e+00 - 139ms/epoch - 8ms/st
ep
Epoch 17/20
18/18 - 0s - loss: 1.8670e-04 - acc: 0.0036 - val_loss: 6.4762e-04 - val_acc: 0.0000e+00 - 83ms/epoch - 5ms/ste
p
Epoch 18/20
18/18 - 0s - loss: 1.6747e-04 - acc: 0.0036 - val_loss: 5.6382e-04 - val_acc: 0.0000e+00 - 102ms/epoch - 6ms/st
ep
Epoch 19/20
18/18 - 0s - loss: 1.8916e-04 - acc: 0.0036 - val_loss: 5.9840e-04 - val_acc: 0.0000e+00 - 137ms/epoch - 8ms/st
ep
Epoch 20/20
18/18 - 0s - loss: 1.4422e-04 - acc: 0.0036 - val_loss: 4.4577e-04 - val_acc: 0.0000e+00 - 131ms/epoch - 7ms/st
ep
```

Out[ ]:

```
<keras.callbacks.History at 0x7f0d037bed50>
```

In [ ]:

```
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

In [ ]:

```
# Estimate model performance
#SInce we used minmaxscaler we can now use scaler.inverse_transform
#to invert the transformation.

trainPredict = scaler.inverse_transform(trainPredict)
trainY_inverse = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY_inverse = scaler.inverse_transform([testY])
```

In [ ]:

```
def MAPE(Y_actual,Y_Predicted):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
print(MAPE(testY_inverse[0], testPredict[:,0]))
print(MAPE(testY_inverse[0], testPredict[:,0]))
```

```
4.456400047951194
4.456400047951194
```

In [ ]:

```
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY_inverse[0], trainPredict[:,0]))
print('Train Score: %.3f RMSE' % (trainScore))

testScore = math.sqrt(mean_squared_error(testY_inverse[0], testPredict[:,0]))
print('Test Score: %.3f RMSE' % (testScore))
```

```
Train Score: 9.827 RMSE
Test Score: 18.390 RMSE
```

In [ ]:

```
def mape(actual, pred):
    actual, pred = np.array(actual), np.array(pred)
    return np.mean(np.abs((actual - pred) / actual)) * 100
mape(testY_inverse[0], testPredict[:,0])
```

Out[ ]:

```
4.456400047951194
```

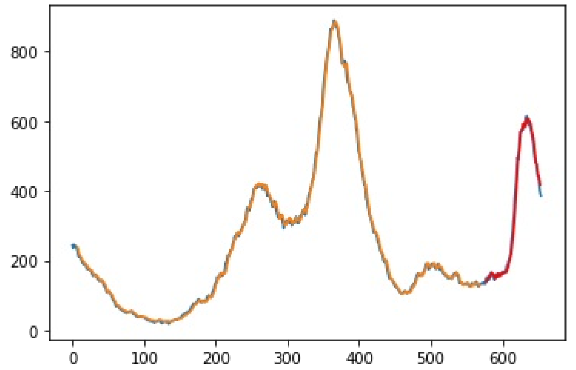
In [ ]:

```
# shift train predictions for plotting
```

In [ ]:
```
# shift train predictions for plotting
#we must shift the predictions so that they align on the x-axis with the original dataset.
trainPredictPlot = np.empty_like(dataset)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[seq_size:len(trainPredict)+seq_size, :] = trainPredict
```

In [ ]:
```
# shift test predictions for plotting
testPredictPlot = np.empty_like(dataset)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(seq_size*2)+1:len(dataset)-1, :] = testPredict
```

In [ ]:
```
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot, color = 'r')
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js