LSTM_arodriguezsans-Univariate_Multivariate_Mal

May 18, 2021

1 Málaga

1.1 Load libraries needed

[1]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
matplotlib.style.use('ggplot')
import seaborn as sns
import math
from datetime import date, timedelta
from pandas import read_csv
from pandas.plotting import register_matplotlib_converters
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import RobustScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.preprocessing import MinMaxScaler
```

1.2 Load "Total" dataset

[2]:
```
df_total = pd.read_excel('Total.xls')
# Edit columns names + Lower case column names
df_total.columns = map(str.lower, df_total.columns)
df_total.columns
```

[2]:
```
Index(['sub_region_2', 'fecha', 'provincia_iso', 'num_casos.x',
       'num_casos_prueba_pcr', 'num_casos_prueba_test_ac',
       'num_casos_prueba_ag', 'num_casos_prueba_elisa',
       'num_casos_prueba_desconocida', 'num_casos.y', 'num_hosp', 'num_uci',
       'num_def', 'retail_and_recreation_percent_change_from_baseline',
       'grocery_and_pharmacy_percent_change_from_baseline',
       'parks_percent_change_from_baseline',
       'transit_stations_percent_change_from_baseline',
```

```
 'workplaces_percent_change_from_baseline',
 'residential_percent_change_from_baseline', 'total'],
dtype='object')
```

1.3 Dataframe under observation

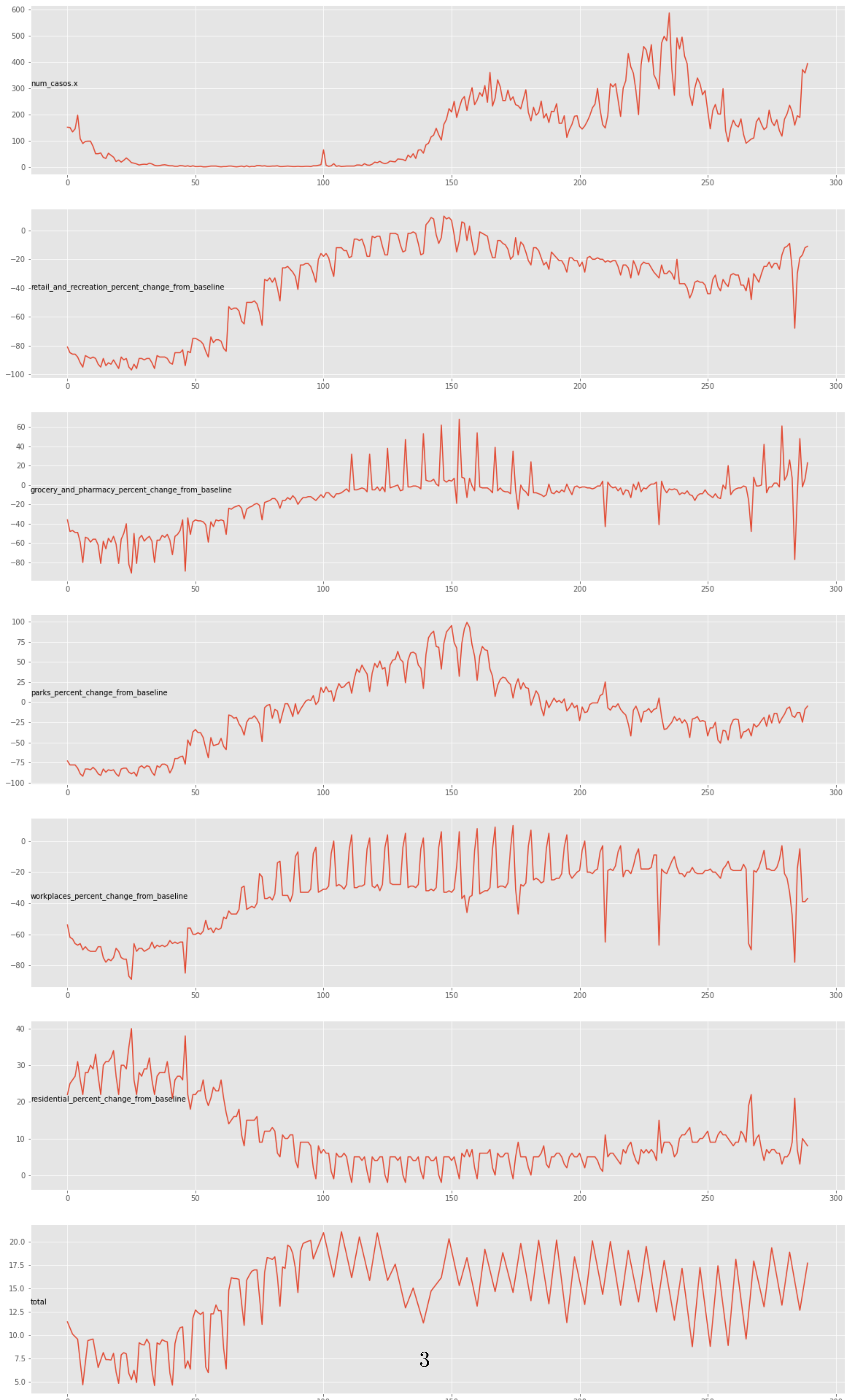
[3]:
```
Mal=df_total.loc[df_total['sub_region_2'] == 'Málaga']
```

[4]:
```
# Set index
Mal = Mal.set_index('fecha')
```

[5]:
```
# We select columns of interest (mobility ones)
Mal=Mal[['num_casos.x'] + list(Mal.loc[:
↪,'retail_and_recreation_percent_change_from_baseline':'total'])]
```

1.4 Plots

[6]:
```
# Columns to plot (mobility ones)
groups = [0, 1, 2, 3, 5, 6, 7]
i = 1
# plot each column
plt.figure(figsize=(20,35))
for group in groups:
    plt.subplot(len(groups), 1, i)
    ## Change "Bar" by any other region for the other cases ##
    plt.plot(Mal.values[:, group])
    plt.title(Mal.columns[group], y=0.5, fontsize=10, loc='left')
    i += 1
plt.show()
```


num_casos.x
retail_and_recreation_percent_change_from_baseline
grocery_and_pharmacy_percent_change_from_baseline
parks_percent_change_from_baseline
workplaces_percent_change_from_baseline
residential_percent_change_from_baseline
total

1.5 LSTM - Univariate

[7]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data = Mal.filter(['num_casos.x'])
npdataset = data.values

# Get the number of rows to train the model
# 94% of the data in order to have the same scenario like in ARIMA
# Train 273 - Test 17
training_data_length = math.ceil(len(npdataset) * 0.94)

# Transform features by scaling each feature to a range between 0 and 1
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(npdataset)
scaled_data[0:5]
```

[7]:
```
array([[0.2572402 ],
       [0.25553663],
       [0.22657581],
       [0.24701874],
       [0.33560477]])
```

[8]:
```
npdataset[0:5]
```

[8]:
```
array([[151],
       [150],
       [133],
       [145],
       [197]], dtype=int64)
```

[9]:
```
len(scaled_data)
```

[9]: 290

[10]:
```
training_data_length
```

[10]: 273

[11]:
```
# We create the scaled training data set
train_data = scaled_data[0:training_data_length, :]
# Nº of previous days check for forecast
loop_back = 14
```

[12]:
```
# Split the data into x_train and y_train data sets
# We create a supervised "problem"
x_train = []
y_train = []
trainingdatasize = len(train_data)
for i in range(loop_back, trainingdatasize):
    #print(i)
    #contains loop_back values 0-loop_back
    x_train.append(train_data[i-loop_back: i, 0])
    #contains all other values
    y_train.append(train_data[i, 0])
```

[13]:
```
# list
x_train[0:2]
```

[13]:
```
[array([0.2572402 , 0.25553663, 0.22657581, 0.24701874, 0.33560477,
        0.18228279, 0.1516184 , 0.16524702, 0.1669506 , 0.1669506 ,
        0.13287905, 0.08517888, 0.08517888, 0.09028961]),
 array([0.25553663, 0.22657581, 0.24701874, 0.33560477, 0.18228279,
        0.1516184 , 0.16524702, 0.1669506 , 0.1669506 , 0.13287905,
        0.08517888, 0.08517888, 0.09028961, 0.06132879])]
```

[14]:
```
# list
y_train[0:2]
```

[14]:
```
[0.06132879045996593, 0.054514480408858604]
```

[15]:
```
# Convert the x_train and y_train to numpy arrays
x_train = np.array(x_train)
y_train = np.array(y_train)
print(x_train[0:2])
print("------------------------------------------------------------")
print(y_train[0:2])
```

```
[[0.2572402  0.25553663 0.22657581 0.24701874 0.33560477 0.18228279
  0.1516184  0.16524702 0.1669506  0.1669506  0.13287905 0.08517888
  0.08517888 0.09028961]
 [0.25553663 0.22657581 0.24701874 0.33560477 0.18228279 0.1516184
  0.16524702 0.1669506  0.1669506  0.13287905 0.08517888 0.08517888
  0.09028961 0.06132879]]
------------------------------------------------------------
[0.06132879 0.05451448]
```

[16]:
```
# Reshape the data
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
print(x_train.shape)
print(y_train.shape)
```

```
(259, 14, 1)
(259,)
```

[17]:
```
x_train[0:2]
```

[17]:
```
array([[[0.2572402 ],
        [0.25553663],
        [0.22657581],
        [0.24701874],
        [0.33560477],
        [0.18228279],
        [0.1516184 ],
        [0.16524702],
        [0.1669506 ],
        [0.1669506 ],
        [0.13287905],
        [0.08517888],
        [0.08517888],
        [0.09028961]],

       [[0.25553663],
        [0.22657581],
        [0.24701874],
        [0.33560477],
        [0.18228279],
        [0.1516184 ],
        [0.16524702],
        [0.1669506 ],
        [0.1669506 ],
        [0.13287905],
        [0.08517888],
        [0.08517888],
        [0.09028961],
        [0.06132879]]])
```

[18]:
```
y_train[0:2]
```

[18]:
```
array([0.06132879, 0.05451448])
```

[19]:
```
# Create a new array containing scaled test values
test_data = scaled_data[training_data_length - loop_back:, :]
#test_data
#test_data.shape

# Create the data sets x_test and y_test
x_test = []
y_test = []
```

```python
#y_test = npdataset[training_data_length:, :]
#y_test = scaled_data[training_data_length:, :]
for i in range(loop_back, len(test_data)):
    x_test.append(test_data[i-loop_back:i, 0])
    y_test.append(test_data[i, 0])

# Convert the data to a numpy array
x_test = np.array(x_test)
y_test = np.array(y_test)

# Reshape the data, so that we get an array with multiple test datasets
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

print(x_test[0:2])
print("------------------------------------------------------------")
print(y_test[0:2])
```

```
[[[0.25042589]
  [0.3032368 ]
  [0.27086882]
  [0.25894378]
  [0.31175468]
  [0.21124361]
  [0.15332198]
  [0.1669506 ]
  [0.18057922]
  [0.18739353]
  [0.29131175]
  [0.31856899]
  [0.27597956]
  [0.24190801]]

 [[0.3032368 ]
  [0.27086882]
  [0.25894378]
  [0.31175468]
  [0.21124361]
  [0.15332198]
  [0.1669506 ]
  [0.18057922]
  [0.18739353]
  [0.29131175]
  [0.31856899]
  [0.27597956]
  [0.24190801]
  [0.2572402 ]]]
------------------------------------------------------------
```

```
[0.2572402  0.36797274]
```

[20]:
```
print(x_test.shape)
print(y_test.shape)
```

```
(17, 14, 1)
(17,)
```

As stated by **Brownlee (2018)**… "

Stochastic Gradient Descent

- Stochastic Gradient Descent, or SGD for short, is an optimization algorithm used to train machine learning algorithms, most notably artificial neural networks used in deep learning.
- The job of the algorithm is to find a set of internal model parameters that perform well against some performance measure such as logarithmic loss or mean squared error.
- Optimization is a type of searching process and you can think of this search as learning. The optimization algorithm is called "gradient descent", where "gradient" refers to the calculation of an error gradient or slope of error and "descent" refers to the moving down along that slope towards some minimum level of error.
- The algorithm is iterative. This means that the search process occurs over multiple discrete steps, each step hopefully slightly improving the model parameters.
- Each step involves using the model with the current set of internal parameters to make predictions on some samples, comparing the predictions to the real expected outcomes, calculating the error, and using the error to update the internal model parameters.
- This update procedure is different for different algorithms, but in the case of artificial neural networks, the backpropagation update algorithm is used.

What Is a Sample?

- A sample is a single row of data.
- It contains inputs that are fed into the algorithm and an output that is used to compare to the prediction and calculate an error.
- A training dataset is comprised of many rows of data, e.g. many samples. A sample may also be called an instance, an observation, an input vector, or a feature vector.
- Now that we know what a sample is, let's define a batch.

What Is a Batch?

- The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters.
- Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g. move down along the error gradient.
- A training dataset can be divided into one or more batches.

- When all training samples are used to create one batch, the learning algorithm is called batch gradient descent. When the batch is the size of one sample, the learning algorithm is called stochastic gradient descent. When the batch size is more than one sample and less than the size of the training dataset, the learning algorithm is called mini-batch gradient descent.
  - Batch Gradient Descent. Batch Size = Size of Training Set
  - Stochastic Gradient Descent. Batch Size = 1
  - Mini-Batch Gradient Descent. 1 < Batch Size < Size of Training Set

What Is an Epoch?

- The number of epochs is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.
- One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, as above, an epoch that has one batch is called the batch gradient descent learning algorithm.
- You can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified "batch size" number of samples.
- The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized. You may see examples of the number of epochs in the literature and in tutorials set to 10, 100, 500, 1000, and larger.
- It is common to create line plots that show epochs along the x-axis as time and the error or skill of the model on the y-axis. These plots are sometimes called learning curves. These plots can help to diagnose whether the model has over learned, under learned, or is suitably fit to the training dataset.

Worked Example

- Finally, let's make this concrete with a small example.
- Assume you have a dataset with 200 samples (rows of data) and you choose a batch size of 5 and 1,000 epochs.
- This means that the dataset will be divided into 40 batches, each with five samples. The model weights will be updated after each batch of five samples.
- This also means that one epoch will involve 40 batches or 40 updates to the model.
- With 1,000 epochs, the model will be exposed to or pass through the whole dataset 1,000 times. That is a total of 40,000 batches during the entire training process.

…"

Brownlee, J., 2018. Difference Between a Batch and an Epoch in a Neural Network. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ [Accessed 12 May 2021].

[21]:
```
# Configure / setup the neural network model - LSTM
model = Sequential()
```

```
# Model with Neurons
# Inputshape = neurons -> Timestamps
neurons= x_train.shape[1]
model.add(LSTM(14,
               activation='relu',
               return_sequences=True,
               input_shape=(x_train.shape[1], 1)))
model.add(LSTM(50,
               activation='relu',
               return_sequences=True))
model.add(LSTM(25,
               activation='relu',
               return_sequences=False))
model.add(Dense(5, activation='relu'))
model.add(Dense(1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')
```

[22]:
```
# Training the model
#early_stop = EarlyStopping(monitor='loss', patience=2, verbose=1)
# fit network
history=model.fit(x_train,
                  y_train,
                  #callbacks=[early_stop],
                  batch_size=2,
                  epochs=50,
                  validation_data=(x_test, y_test),
                  verbose=2)
```

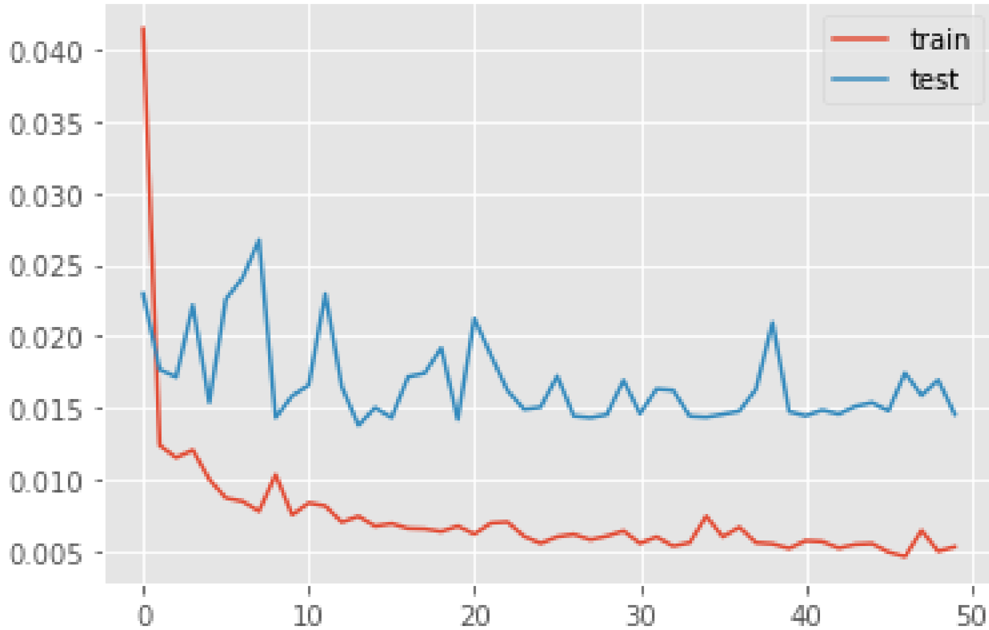
```
Epoch 1/50
130/130 - 12s - loss: 0.0415 - val_loss: 0.0230
Epoch 2/50
130/130 - 2s - loss: 0.0124 - val_loss: 0.0177
Epoch 3/50
130/130 - 1s - loss: 0.0115 - val_loss: 0.0171
Epoch 4/50
130/130 - 2s - loss: 0.0120 - val_loss: 0.0222
Epoch 5/50
130/130 - 2s - loss: 0.0100 - val_loss: 0.0154
Epoch 6/50
130/130 - 2s - loss: 0.0087 - val_loss: 0.0226
Epoch 7/50
130/130 - 1s - loss: 0.0085 - val_loss: 0.0241
Epoch 8/50
130/130 - 2s - loss: 0.0078 - val_loss: 0.0267
```

```
Epoch 9/50
130/130 - 2s - loss: 0.0103 - val_loss: 0.0143
Epoch 10/50
130/130 - 2s - loss: 0.0075 - val_loss: 0.0158
Epoch 11/50
130/130 - 2s - loss: 0.0083 - val_loss: 0.0166
Epoch 12/50
130/130 - 2s - loss: 0.0081 - val_loss: 0.0230
Epoch 13/50
130/130 - 2s - loss: 0.0070 - val_loss: 0.0165
Epoch 14/50
130/130 - 2s - loss: 0.0074 - val_loss: 0.0138
Epoch 15/50
130/130 - 2s - loss: 0.0067 - val_loss: 0.0150
Epoch 16/50
130/130 - 2s - loss: 0.0069 - val_loss: 0.0143
Epoch 17/50
130/130 - 1s - loss: 0.0066 - val_loss: 0.0172
Epoch 18/50
130/130 - 2s - loss: 0.0065 - val_loss: 0.0174
Epoch 19/50
130/130 - 2s - loss: 0.0063 - val_loss: 0.0192
Epoch 20/50
130/130 - 1s - loss: 0.0067 - val_loss: 0.0142
Epoch 21/50
130/130 - 2s - loss: 0.0062 - val_loss: 0.0212
Epoch 22/50
130/130 - 2s - loss: 0.0069 - val_loss: 0.0187
Epoch 23/50
130/130 - 2s - loss: 0.0070 - val_loss: 0.0162
Epoch 24/50
130/130 - 2s - loss: 0.0060 - val_loss: 0.0149
Epoch 25/50
130/130 - 2s - loss: 0.0055 - val_loss: 0.0150
Epoch 26/50
130/130 - 2s - loss: 0.0060 - val_loss: 0.0172
Epoch 27/50
130/130 - 2s - loss: 0.0061 - val_loss: 0.0144
Epoch 28/50
130/130 - 2s - loss: 0.0058 - val_loss: 0.0143
Epoch 29/50
130/130 - 2s - loss: 0.0060 - val_loss: 0.0145
Epoch 30/50
130/130 - 2s - loss: 0.0064 - val_loss: 0.0169
Epoch 31/50
130/130 - 2s - loss: 0.0055 - val_loss: 0.0146
Epoch 32/50
130/130 - 2s - loss: 0.0060 - val_loss: 0.0163
```

```
Epoch 33/50
130/130 - 2s - loss: 0.0053 - val_loss: 0.0162
Epoch 34/50
130/130 - 2s - loss: 0.0056 - val_loss: 0.0144
Epoch 35/50
130/130 - 2s - loss: 0.0074 - val_loss: 0.0143
Epoch 36/50
130/130 - 2s - loss: 0.0060 - val_loss: 0.0145
Epoch 37/50
130/130 - 3s - loss: 0.0066 - val_loss: 0.0148
Epoch 38/50
130/130 - 2s - loss: 0.0056 - val_loss: 0.0163
Epoch 39/50
130/130 - 2s - loss: 0.0055 - val_loss: 0.0210
Epoch 40/50
130/130 - 2s - loss: 0.0052 - val_loss: 0.0147
Epoch 41/50
130/130 - 2s - loss: 0.0057 - val_loss: 0.0144
Epoch 42/50
130/130 - 2s - loss: 0.0056 - val_loss: 0.0148
Epoch 43/50
130/130 - 2s - loss: 0.0052 - val_loss: 0.0146
Epoch 44/50
130/130 - 2s - loss: 0.0054 - val_loss: 0.0151
Epoch 45/50
130/130 - 2s - loss: 0.0055 - val_loss: 0.0154
Epoch 46/50
130/130 - 2s - loss: 0.0049 - val_loss: 0.0148
Epoch 47/50
130/130 - 2s - loss: 0.0046 - val_loss: 0.0174
Epoch 48/50
130/130 - 2s - loss: 0.0064 - val_loss: 0.0159
Epoch 49/50
130/130 - 2s - loss: 0.0050 - val_loss: 0.0169
Epoch 50/50
130/130 - 2s - loss: 0.0053 - val_loss: 0.0145
```

[23]:

```
# Plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```



[24]:
```
# Get the predicted values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

[25]:
```
predictions
```

[25]:
```
array([[160.29161],
       [174.05699],
       [194.69504],
       [216.32848],
       [231.6236 ],
       [239.87944],
       [237.45021],
       [223.6806 ],
       [208.85999],
       [198.76729],
       [197.30379],
       [206.60167],
       [215.70836],
       [225.38106],
       [234.5193 ],
       [251.30954],
       [273.92502]], dtype=float32)
```

[26]:
```
y_test = y_test.reshape(-1,1)
y_test = scaler.inverse_transform(y_test)
y_test
```

[26]:
```
array([[151.],
       [216.],
       [172.],
       [157.],
       [180.],
       [138.],
       [117.],
       [182.],
       [202.],
       [235.],
       [209.],
       [159.],
       [195.],
       [188.],
       [371.],
       [358.],
       [394.]])
```

[27]:
```
# Calculate the mean absolute error (MAE)
mae = mean_absolute_error(y_test, predictions)
print('MAE: ' + str(round(mae, 1)))

# Calculate the root mean squarred error (RMSE)
rmse = np.sqrt(mean_squared_error(y_test,predictions))
print('RMSE: ' + str(round(rmse, 1)))
```

```
MAE: 57.2
RMSE: 70.8
```

[28]:
```
# Date from which on the date is displayed
display_start_date = "2020-09-16"

# Add the difference between the valid and predicted prices
train = data[:training_data_length + 1]
valid = data[training_data_length:]
```

[29]:
```
valid.insert(1, "Predictions", predictions, True)
valid.insert(1, "Difference", valid["Predictions"] - valid["num_casos.x"], True)
```

[30]:
```
# Zoom-in to a closer timeframe
valid = valid[valid.index > display_start_date]
train = train[train.index > display_start_date]
```

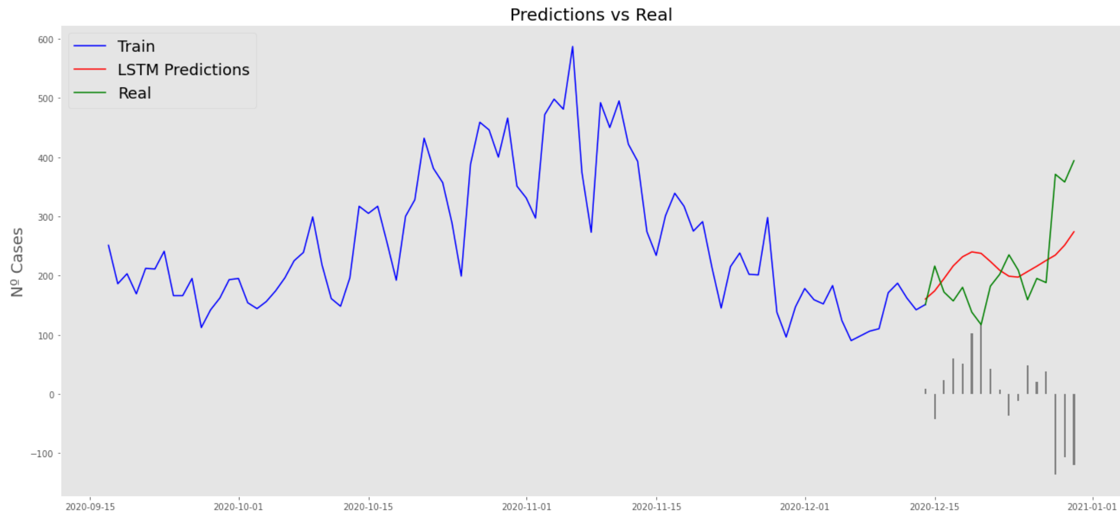
```
# Show the test / valid and predicted prices
valid
```

[30]:

	num_casos.x	Difference	Predictions
fecha			
2020-12-14	151	9.291611	160.291611
2020-12-15	216	-41.943008	174.056992
2020-12-16	172	22.695038	194.695038
2020-12-17	157	59.328476	216.328476
2020-12-18	180	51.623596	231.623596
2020-12-19	138	101.879440	239.879440
2020-12-20	117	120.450211	237.450211
2020-12-21	182	41.680603	223.680603
2020-12-22	202	6.859985	208.859985
2020-12-23	235	-36.232712	198.767288
2020-12-24	209	-11.696213	197.303787
2020-12-25	159	47.601669	206.601669
2020-12-26	195	20.708359	215.708359
2020-12-27	188	37.381058	225.381058
2020-12-28	371	-136.480698	234.519302
2020-12-29	358	-106.690460	251.309540
2020-12-30	394	-120.074982	273.925018

[31]:
```
# Visualize the data
fig, ax1 = plt.subplots(figsize=(22, 10), sharex=True)

# Data - Train
xt = train.index;
yt = train[["num_casos.x"]]
# Data - Test / validation
xv = valid.index;
yv = valid[["num_casos.x", "Predictions"]]

# Plot
plt.title("Predictions vs Real", fontsize=20)
plt.ylabel("Nº Cases", fontsize=18)

plt.plot(yt, color="blue", linewidth=1.5)
plt.plot(yv["Predictions"], color="red", linewidth=1.5)
plt.plot(yv["num_casos.x"], color="green", linewidth=1.5)
plt.legend(["Train", "LSTM Predictions", "Real"],
           loc="upper left", fontsize=18)

# Bar plot with the differences
x = valid.index
y = valid["Difference"]
plt.bar(x, y, width=0.2, color="grey")
```

```
plt.grid()
plt.show()
```



1.6 LSTM - 2 variables + infections reported

[32]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data_v2 = Mal.filter(['num_casos.x',
                      'residential_percent_change_from_baseline',
                      'total'])
npdataset_v2 = data_v2.values

# Get the number of rows to train the model
# 94% of the data in order to have the same scenario like in ARIMA
# Train 273 - Test 17
training_data_length_v2 = math.ceil(len(npdataset_v2) * 0.94)

# Transform features by scaling each feature to a range between 0 and 1
scaler_v2 = MinMaxScaler(feature_range=(0, 1))
scaled_data_v2 = scaler_v2.fit_transform(npdataset_v2)
scaled_data_v2[0:5]
```

[32]:
```
array([[0.2572402 , 0.57142857, 0.41419042],
       [0.25553663, 0.64285714, 0.37446938],
       [0.22657581, 0.66666667, 0.33474833],
       [0.24701874, 0.69047619, 0.31807156],
       [0.33560477, 0.78571429, 0.30139478]])
```

[33]:
```
# Creating a separate scaler that works on a single column for scaling␣
↪predictions
scaler_v2_pred = MinMaxScaler(feature_range=(0, 1))
df_cases = pd.DataFrame(Mal['num_casos.x'])
np_cases_scaled_v2 = scaler_v2_pred.fit_transform(df_cases)
np_cases_scaled_v2[0:5]
```

[33]:
```
array([[0.2572402 ],
       [0.25553663],
       [0.22657581],
       [0.24701874],
       [0.33560477]])
```

[34]:
```
# Create the training data
train_data_v2 = scaled_data_v2[0:training_data_length_v2, :]
print(train_data_v2.shape)
```

```
(273, 3)
```

[35]:
```
train_data_v2[0:2]
```

[35]:
```
array([[0.2572402 , 0.57142857, 0.41419042],
       [0.25553663, 0.64285714, 0.37446938]])
```

[36]:
```
training_data_length_v2
```

[36]:
```
273
```

[37]:
```
loop_back
```

[37]:
```
14
```

[38]:
```
x_train_v2 = []
y_train_v2 = []
# The RNN needs data with the format of [samples, time steps, features].
# Here, we create N samples, N loop_back time steps per sample, and 2 features␣
↪(all mobility)
for i in range(loop_back, training_data_length_v2):
    #print(i)
    #contains loop_back values ->>> 0-loop_back * columsn
    x_train_v2.append(train_data_v2[i-loop_back:i,:])
    #contains the prediction values for test / validation
    y_train_v2.append(train_data_v2[i, 0])

# Convert the x_train and y_train to numpy arrays
x_train_v2, y_train_v2 = np.array(x_train_v2), np.array(y_train_v2)
x_train_v2[0:2]
```

[38]:
```
array([[[0.2572402 , 0.57142857, 0.41419042],
        [0.25553663, 0.64285714, 0.37446938],
        [0.22657581, 0.66666667, 0.33474833],
        [0.24701874, 0.69047619, 0.31807156],
        [0.33560477, 0.78571429, 0.30139478],
        [0.18228279, 0.66666667, 0.15342632],
        [0.1516184 , 0.57142857, 0.00545785],
        [0.16524702, 0.71428571, 0.14918132],
        [0.1669506 , 0.71428571, 0.29290479],
        [0.1669506 , 0.76190476, 0.29775622],
        [0.13287905, 0.73809524, 0.30260764],
        [0.08517888, 0.83333333, 0.21012735],
        [0.08517888, 0.69047619, 0.11764706],
        [0.09028961, 0.57142857, 0.16616131]],

       [[0.25553663, 0.64285714, 0.37446938],
        [0.22657581, 0.66666667, 0.33474833],
        [0.24701874, 0.69047619, 0.31807156],
        [0.33560477, 0.78571429, 0.30139478],
        [0.18228279, 0.66666667, 0.15342632],
        [0.1516184 , 0.57142857, 0.00545785],
        [0.16524702, 0.71428571, 0.14918132],
        [0.1669506 , 0.71428571, 0.29290479],
        [0.1669506 , 0.76190476, 0.29775622],
        [0.13287905, 0.73809524, 0.30260764],
        [0.08517888, 0.83333333, 0.21012735],
        [0.08517888, 0.69047619, 0.11764706],
        [0.09028961, 0.57142857, 0.16616131],
        [0.06132879, 0.76190476, 0.21467556]]])
```

[39]:
```
y_train_v2[0:2]
```

[39]:
```
array([0.06132879, 0.05451448])
```

[40]:
```
print(x_train_v2.shape, y_train_v2.shape)
```

```
(259, 14, 3) (259,)
```

[41]:
```
# Create the test data
test_data_v2 = scaled_data_v2[training_data_length_v2 - loop_back:, :]
print(test_data_v2.shape)

x_test_v2 = []
y_test_v2 = []
# The RNN needs data with the format of [samples, time steps, features].
# Here, we create N samples, N loop_back time steps per sample, and 3 features␣
↪(mobility + num_casos.x)
```

```python
for i in range(loop_back, len(test_data_v2)):
    #print(i)
    #contains loop_back values ->>> 0-loop_back * columsn
    x_test_v2.append(test_data_v2[i-loop_back:i,:])
    #contains the prediction values for test / validation
    y_test_v2.append(test_data_v2[i, 0])

# Convert the x_train and y_train to numpy arrays
x_test_v2, y_test_v2 = np.array(x_test_v2), np.array(y_test_v2)
x_test_v2[0:2]
#len(x_train_v2)
```

```
(31, 3)
```

[41]:
```
array([[[0.25042589, 0.26190476, 0.44734182],
        [0.3032368 , 0.23809524, 0.63391955],
        [0.27086882, 0.26190476, 0.82049727],
        [0.25894378, 0.26190476, 0.69102486],
        [0.31175468, 0.33333333, 0.56155246],
        [0.21124361, 0.30952381, 0.43208005],
        [0.15332198, 0.26190476, 0.30260764],
        [0.1669506 , 0.5       , 0.47180109],
        [0.18057922, 0.57142857, 0.64099454],
        [0.18739353, 0.23809524, 0.81018799],
        [0.29131175, 0.28571429, 0.73559733],
        [0.31856899, 0.30952381, 0.66100667],
        [0.27597956, 0.21428571, 0.58641601],
        [0.24190801, 0.14285714, 0.51182535]],

       [[0.3032368 , 0.23809524, 0.63391955],
        [0.27086882, 0.26190476, 0.82049727],
        [0.25894378, 0.26190476, 0.69102486],
        [0.31175468, 0.33333333, 0.56155246],
        [0.21124361, 0.30952381, 0.43208005],
        [0.15332198, 0.26190476, 0.30260764],
        [0.1669506 , 0.5       , 0.47180109],
        [0.18057922, 0.57142857, 0.64099454],
        [0.18739353, 0.23809524, 0.81018799],
        [0.29131175, 0.28571429, 0.73559733],
        [0.31856899, 0.30952381, 0.66100667],
        [0.27597956, 0.21428571, 0.58641601],
        [0.24190801, 0.14285714, 0.51182535],
        [0.2572402 , 0.21428571, 0.63998383]]])
```

[42]:
```python
y_test_v2[0:2]
```

[42]:
```
array([0.2572402 , 0.36797274])
```

[43]:
```
print(x_test_v2.shape, y_test_v2.shape)
```

```
(17, 14, 3) (17,)
```

[44]:
```
# Configure the neural network model
model_v2 = Sequential()

# Model with N "loop_back" Neurons
# Inputshape >>> loop_back Timestamps, each with x_train.shape[2] variables
n_neurons_v2 = x_train_v2.shape[1] * x_train_v2.shape[2]
print(n_neurons_v2, x_train_v2.shape[1], x_train_v2.shape[2])

model_v2.add(LSTM(n_neurons_v2,
                  activation='relu',
                  return_sequences=True,
                  input_shape=(x_train_v2.shape[1],
                               x_train_v2.shape[2])))
model_v2.add(LSTM(50, activation='relu', return_sequences=True))
model_v2.add(LSTM(25, activation='relu',return_sequences=False))
model_v2.add(Dense(5, activation='relu'))
model_v2.add(Dense(1))

# Compile the model
model_v2.compile(optimizer='adam', loss='mean_squared_error')
```

```
42 14 3
```

[45]:
```
# Training the model
early_stop_v2 = EarlyStopping(monitor='loss', patience=2, verbose=1)
history_v2 = model_v2.fit(x_train_v2,
                    y_train_v2,
                    batch_size=2,
                    validation_data=(x_test_v2, y_test_v2),
                    epochs=50
                    #callbacks=[early_stop_v2]
                         )
```

```
Epoch 1/50
130/130 [==============================] - 12s 24ms/step - loss: 0.0741 -
val_loss: 0.0373
Epoch 2/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0079 -
val_loss: 0.0136
Epoch 3/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0085 -
val_loss: 0.0126
Epoch 4/50
130/130 [==============================] - 4s 30ms/step - loss: 0.0090 -
```

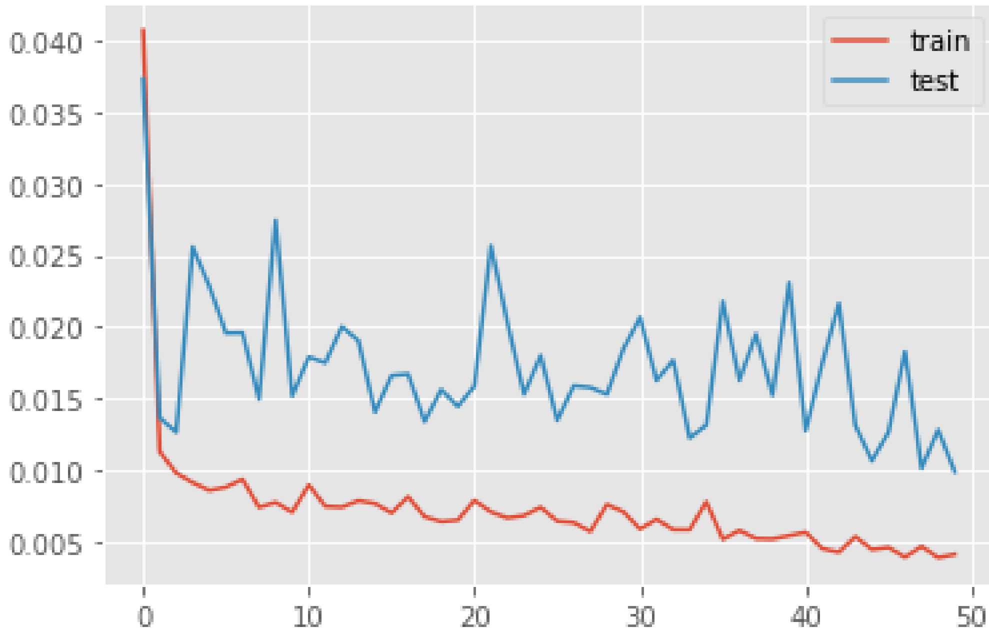
```
val_loss: 0.0256
Epoch 5/50
130/130 [==============================] - 3s 19ms/step - loss: 0.0074 -
val_loss: 0.0228
Epoch 6/50
130/130 [==============================] - 4s 34ms/step - loss: 0.0085 -
val_loss: 0.0196
Epoch 7/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0065 -
val_loss: 0.0196
Epoch 8/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0061 -
val_loss: 0.0149
Epoch 9/50
130/130 [==============================] - ETA: 0s - loss: 0.006 - 2s 17ms/step
- loss: 0.0067 - val_loss: 0.0275
Epoch 10/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0068 -
val_loss: 0.0152
Epoch 11/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0096 -
val_loss: 0.0179
Epoch 12/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0072 -
val_loss: 0.0175
Epoch 13/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0087 -
val_loss: 0.0200
Epoch 14/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0075 -
val_loss: 0.0190
Epoch 15/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0079 -
val_loss: 0.0141
Epoch 16/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0060 -
val_loss: 0.0166
Epoch 17/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0085 -
val_loss: 0.0167
Epoch 18/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0065 -
val_loss: 0.0134
Epoch 19/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0087 -
val_loss: 0.0156
Epoch 20/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0067 -
```

```
val_loss: 0.0144: 0s - loss: 0 - ETA: 0s - loss: 0.0
Epoch 21/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0078 -
val_loss: 0.0159
Epoch 22/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0043 -
val_loss: 0.0257
Epoch 23/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0061 -
val_loss: 0.0203
Epoch 24/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0059 -
val_loss: 0.0153
Epoch 25/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0075 -
val_loss: 0.0180
Epoch 26/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0079 -
val_loss: 0.0135
Epoch 27/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0059 -
val_loss: 0.0159
Epoch 28/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0054 -
val_loss: 0.0158
Epoch 29/50
130/130 [==============================] - 4s 28ms/step - loss: 0.0070 -
val_loss: 0.0153
Epoch 30/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0068 -
val_loss: 0.0185
Epoch 31/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0066 -
val_loss: 0.0207
Epoch 32/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0062 -
val_loss: 0.0163
Epoch 33/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0055 -
val_loss: 0.0177
Epoch 34/50
130/130 [==============================] - 4s 33ms/step - loss: 0.0046 -
val_loss: 0.0122
Epoch 35/50
130/130 [==============================] - 3s 26ms/step - loss: 0.0062 -
val_loss: 0.0131
Epoch 36/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0044 -
```

```
val_loss: 0.0218
Epoch 37/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0046 -
val_loss: 0.0163
Epoch 38/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0050 -
val_loss: 0.0195
Epoch 39/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0052 -
val_loss: 0.0152
Epoch 40/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0049 -
val_loss: 0.0231
Epoch 41/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0077 -
val_loss: 0.0127
Epoch 42/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0051 -
val_loss: 0.0174
Epoch 43/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0049 -
val_loss: 0.0216
Epoch 44/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0063 -
val_loss: 0.0131
Epoch 45/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0064 -
val_loss: 0.0107
Epoch 46/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0039 -
val_loss: 0.0126
Epoch 47/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0037 -
val_loss: 0.0183
Epoch 48/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0047 -
val_loss: 0.0101
Epoch 49/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0036 -
val_loss: 0.0128
Epoch 50/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0045 -
val_loss: 0.0099
```

[77]:
```
# Plot history
plt.plot(history_v2.history['loss'], label='train')
plt.plot(history_v2.history['val_loss'], label='test')
```

```
plt.legend()
plt.show()
```



[47]:
```
# Get the predicted values
predictions_v2 = model_v2.predict(x_test_v2)
predictions_v2
```

[47]:
```
array([[0.24143285],
       [0.24418049],
       [0.26096645],
       [0.26745617],
       [0.30008662],
       [0.30643493],
       [0.31680372],
       [0.3675339 ],
       [0.3988172 ],
       [0.36995304],
       [0.36462146],
       [0.3655196 ],
       [0.35172394],
       [0.33953038],
       [0.38438243],
       [0.4510553 ],
       [0.50198305]], dtype=float32)
```

[48]:
```python
# Get the predicted values
pred_unscaled_v2 = scaler_v2_pred.inverse_transform(predictions_v2)
y_test_v2_unscaled = scaler_v2_pred.inverse_transform(y_test_v2.reshape(-1, 1))
```

[49]:
```python
# Calculate the mean absolute error (MAE)
mae_v2 = mean_absolute_error(pred_unscaled_v2, y_test_v2_unscaled)
print('MAE: ' + str(round(mae_v2, 1)))

# Calculate the root mean squarred error (RMSE)
rmse_v2 = np.sqrt(mean_squared_error(y_test_v2_unscaled,pred_unscaled_v2))
print('RMSE: ' + str(round(rmse_v2, 1)))
```

```
MAE: 42.4
RMSE: 58.3
```

[50]:
```python
# Date from which on the date is displayed
display_start_date_v2 = "2020-09-16"

# Add the difference between the valid and predicted prices
train_v2 = data_v2[:training_data_length_v2 + 1]
valid_v2 = data_v2[training_data_length_v2:]
```

[51]:
```python
valid_v2.insert(1, "Predictions", pred_unscaled_v2, True)
valid_v2.insert(1, "Difference", valid_v2["Predictions"] - valid_v2["num_casos.
 ↪x"], True)
```

[52]:
```python
# Zoom-in to a closer timeframe
valid_v2 = valid_v2[valid_v2.index > display_start_date_v2]
train_v2 = train_v2[train_v2.index > display_start_date_v2]

# Show the test / valid and predicted prices
valid_v2
```

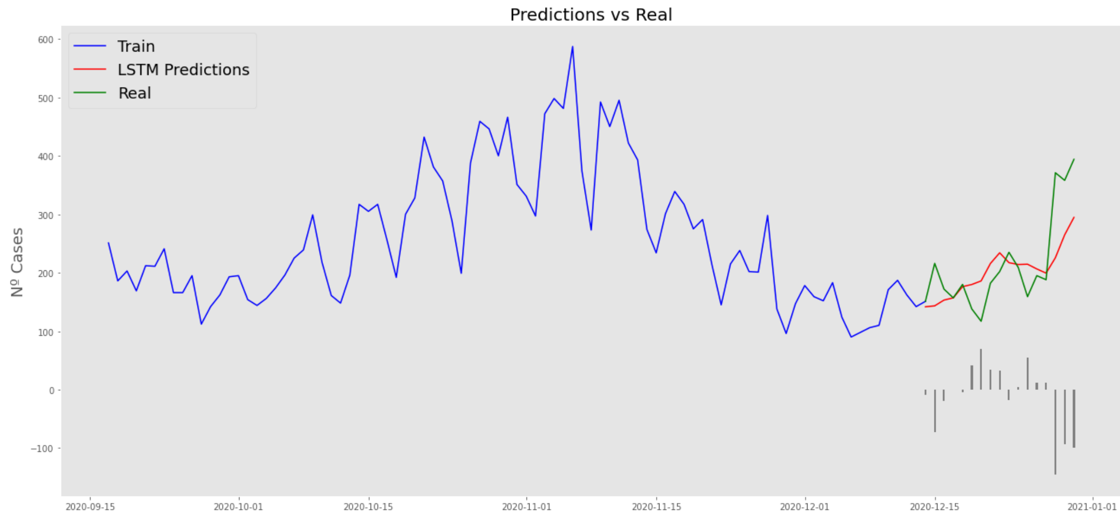
[52]:

fecha	num_casos.x	Difference	Predictions	\
2020-12-14	151	-9.278915	141.721085	
2020-12-15	216	-72.666061	143.333939	
2020-12-16	172	-18.812698	153.187302	
2020-12-17	157	-0.003220	156.996780	
2020-12-18	180	-3.849152	176.150848	
2020-12-19	138	41.877304	179.877304	
2020-12-20	117	68.963791	185.963791	
2020-12-21	182	33.742401	215.742401	
2020-12-22	202	32.105698	234.105698	
2020-12-23	235	-17.837570	217.162430	
2020-12-24	209	5.032791	214.032791	
2020-12-25	159	55.560013	214.560013	

```
2020-12-26          195   11.461960   206.461960
2020-12-27          188   11.304337   199.304337
2020-12-28          371 -145.367508   225.632492
2020-12-29          358  -93.230560   264.769440
2020-12-30          394  -99.335938   294.664062

            residential_percent_change_from_baseline      total
fecha
2020-12-14                                       7.0  15.113333
2020-12-15                                       6.0  17.226667
2020-12-16                                       7.0  19.340000
2020-12-17                                       7.0  17.800000
2020-12-18                                       6.0  16.260000
2020-12-19                                       6.0  14.720000
2020-12-20                                       3.0  13.180000
2020-12-21                                       5.0  15.073333
2020-12-22                                       5.0  16.966667
2020-12-23                                       6.0  18.860000
2020-12-24                                       9.0  17.302500
2020-12-25                                      21.0  15.745000
2020-12-26                                       7.0  14.187500
2020-12-27                                       3.0  12.630000
2020-12-28                                      10.0  14.316667
2020-12-29                                       9.0  16.003333
2020-12-30                                       8.0  17.690000
```

[53]:
```
# Visualize the data
fig, ax1 = plt.subplots(figsize=(22, 10), sharex=True)

# Data - Train
xt_v2 = train_v2.index;
yt_v2 = train_v2[["num_casos.x"]]
# Data - Test / validation
xv_v2 = valid_v2.index;
yv_v2 = valid_v2[["num_casos.x", "Predictions"]]

# Plot
plt.title("Predictions vs Real", fontsize=20)
plt.ylabel("Nº Cases", fontsize=18)

plt.plot(yt_v2, color="blue", linewidth=1.5)
plt.plot(yv_v2["Predictions"], color="red", linewidth=1.5)
plt.plot(yv_v2["num_casos.x"], color="green", linewidth=1.5)
plt.legend(["Train", "LSTM Predictions", "Real"],
           loc="upper left", fontsize=18)

# Bar plot with the differences
```

```
x_v2 = valid_v2.index
y_v2 = valid_v2["Difference"]
plt.bar(x_v2, y_v2, width=0.2, color="grey")
plt.grid()
plt.show()
```



1.7 LSTM - All variables

[54]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data_v3 = Mal.filter(['num_casos.x',
                      'residential_percent_change_from_baseline',
                      'retail_and_recreation_percent_change_from_baseline',
                      'grocery_and_pharmacy_percent_change_from_baseline',
                      'parks_percent_change_from_baseline',
                      'transit_stations_percent_change_from_baseline',
                      'workplaces_percent_change_from_baseline',
                      'total'])
npdataset_v3 = data_v3.values

# Get the number of rows to train the model
# 94% of the data in order to have the same scenario like in ARIMA
# Train 273 - Test 17
training_data_length_v3 = math.ceil(len(npdataset_v3) * 0.94)

# Transform features by scaling each feature to a range between 0 and 1
scaler_v3 = MinMaxScaler(feature_range=(0, 1))
scaled_data_v3 = scaler_v3.fit_transform(npdataset_v3)
scaled_data_v3[0:5]
```

[54]:
```
array([[0.2572402 , 0.57142857, 0.14953271, 0.34591195, 0.09947644,
         0.3375    , 0.35353535, 0.41419042],
        [0.25553663, 0.64285714, 0.11214953, 0.27044025, 0.07329843,
         0.2625    , 0.27272727, 0.37446938],
        [0.22657581, 0.66666667, 0.10280374, 0.27672956, 0.07329843,
         0.2375    , 0.26262626, 0.33474833],
        [0.24701874, 0.69047619, 0.10280374, 0.26415094, 0.07329843,
         0.2125    , 0.23232323, 0.31807156],
        [0.33560477, 0.78571429, 0.08411215, 0.26415094, 0.05235602,
         0.1625    , 0.22222222, 0.30139478]])
```

[55]:
```
# Creating a separate scaler that works on a single column for scaling␣
 ↪predictions
scaler_v3_pred = MinMaxScaler(feature_range=(0, 1))
df_cases = pd.DataFrame(Mal['num_casos.x'])
np_cases_scaled_v3 = scaler_v3_pred.fit_transform(df_cases)
np_cases_scaled_v3[0:5]
```

[55]:
```
array([[0.2572402 ],
        [0.25553663],
        [0.22657581],
        [0.24701874],
        [0.33560477]])
```

[56]:
```
# Create the training data
train_data_v3 = scaled_data_v3[0:training_data_length_v3, :]
print(train_data_v3.shape)
```

```
(273, 8)
```

[57]:
```
train_data_v3[0:2]
```

[57]:
```
array([[0.2572402 , 0.57142857, 0.14953271, 0.34591195, 0.09947644,
         0.3375    , 0.35353535, 0.41419042],
        [0.25553663, 0.64285714, 0.11214953, 0.27044025, 0.07329843,
         0.2625    , 0.27272727, 0.37446938]])
```

[58]:
```
training_data_length_v3
```

[58]:
```
273
```

[59]:
```
x_train_v3 = []
y_train_v3 = []
# The RNN needs data with the format of [samples, time steps, features].
# Here, we create N samples, N loop_back time steps per sample, and 8 features␣
 ↪(all)
for i in range(loop_back, training_data_length_v3):
```

```python
    #print(i)
    #contains loop_back values ->>> 0-loop_back * columsn
    x_train_v3.append(train_data_v3[i-loop_back:i,:])
    #contains the prediction values for test / validation
    y_train_v3.append(train_data_v3[i, 0])

# Convert the x_train and y_train to numpy arrays
x_train_v3, y_train_v3 = np.array(x_train_v3), np.array(y_train_v3)
x_train_v3[0:2]
```

[59]:

```
array([[[0.2572402 , 0.57142857, 0.14953271, 0.34591195, 0.09947644,
         0.3375    , 0.35353535, 0.41419042],
        [0.25553663, 0.64285714, 0.11214953, 0.27044025, 0.07329843,
         0.2625    , 0.27272727, 0.37446938],
        [0.22657581, 0.66666667, 0.10280374, 0.27672956, 0.07329843,
         0.2375    , 0.26262626, 0.33474833],
        [0.24701874, 0.69047619, 0.10280374, 0.26415094, 0.07329843,
         0.2125    , 0.23232323, 0.31807156],
        [0.33560477, 0.78571429, 0.08411215, 0.26415094, 0.05235602,
         0.1625    , 0.22222222, 0.30139478],
        [0.18228279, 0.66666667, 0.04672897, 0.20125786, 0.01570681,
         0.1125    , 0.23232323, 0.15342632],
        [0.1516184 , 0.57142857, 0.01869159, 0.06918239, 0.        ,
         0.0625    , 0.19191919, 0.00545785],
        [0.16524702, 0.71428571, 0.09345794, 0.2327044 , 0.04712042,
         0.1125    , 0.21212121, 0.14918132],
        [0.1669506 , 0.71428571, 0.08411215, 0.22641509, 0.04712042,
         0.1125    , 0.19191919, 0.29290479],
        [0.1669506 , 0.76190476, 0.07476636, 0.20125786, 0.04188482,
         0.1       , 0.18181818, 0.29775622],
        [0.13287905, 0.73809524, 0.08411215, 0.22012579, 0.05759162,
         0.1       , 0.18181818, 0.30260764],
        [0.08517888, 0.83333333, 0.07476636, 0.22012579, 0.04188482,
         0.075     , 0.18181818, 0.21012735],
        [0.08517888, 0.69047619, 0.03738318, 0.18238994, 0.01570681,
         0.05      , 0.21212121, 0.11764706],
        [0.09028961, 0.57142857, 0.01869159, 0.06289308, 0.0052356 ,
         0.025     , 0.21212121, 0.16616131]],

       [[0.25553663, 0.64285714, 0.11214953, 0.27044025, 0.07329843,
         0.2625    , 0.27272727, 0.37446938],
        [0.22657581, 0.66666667, 0.10280374, 0.27672956, 0.07329843,
         0.2375    , 0.26262626, 0.33474833],
        [0.24701874, 0.69047619, 0.10280374, 0.26415094, 0.07329843,
         0.2125    , 0.23232323, 0.31807156],
        [0.33560477, 0.78571429, 0.08411215, 0.26415094, 0.05235602,
         0.1625    , 0.22222222, 0.30139478],
```

```
       [0.18228279, 0.66666667, 0.04672897, 0.20125786, 0.01570681,
        0.1125    , 0.23232323, 0.15342632],
       [0.1516184 , 0.57142857, 0.01869159, 0.06918239, 0.        ,
        0.0625    , 0.19191919, 0.00545785],
       [0.16524702, 0.71428571, 0.09345794, 0.2327044 , 0.04712042,
        0.1125    , 0.21212121, 0.14918132],
       [0.1669506 , 0.71428571, 0.08411215, 0.22641509, 0.04712042,
        0.1125    , 0.19191919, 0.29290479],
       [0.1669506 , 0.76190476, 0.07476636, 0.20125786, 0.04188482,
        0.1       , 0.18181818, 0.29775622],
       [0.13287905, 0.73809524, 0.08411215, 0.22012579, 0.05759162,
        0.1       , 0.18181818, 0.30260764],
       [0.08517888, 0.83333333, 0.07476636, 0.22012579, 0.04188482,
        0.075     , 0.18181818, 0.21012735],
       [0.08517888, 0.69047619, 0.03738318, 0.18238994, 0.01570681,
        0.05      , 0.21212121, 0.11764706],
       [0.09028961, 0.57142857, 0.01869159, 0.06289308, 0.0052356 ,
        0.025     , 0.21212121, 0.16616131],
       [0.06132879, 0.76190476, 0.07476636, 0.20754717, 0.04712042,
        0.0875    , 0.14141414, 0.21467556]]])
```

[60]:
```
y_train_v3[0:2]
```

[60]:
```
array([0.06132879, 0.05451448])
```

[61]:
```
print(x_train_v3.shape, y_train_v3.shape)
```

```
(259, 14, 8) (259,)
```

[62]:
```
# Create the test data
test_data_v3 = scaled_data_v3[training_data_length_v3 - loop_back:, :]
print(test_data_v3.shape)

x_test_v3 = []
y_test_v3 = []
# The RNN needs data with the format of [samples, time steps, features].
# Here, we create N samples, N loop_back time steps per sample, and 3 features␣
 ↪(mobility + num_casos.x)
for i in range(loop_back, len(test_data_v3)):
    #print(i)
    #contains loop_back values ->>> 0-loop_back * columsn
    x_test_v3.append(test_data_v3[i-loop_back:i,:])
    #contains the prediction values for test / validation
    y_test_v3.append(test_data_v3[i, 0])

# Convert the x_train and y_train to numpy arrays
x_test_v3, y_test_v3 = np.array(x_test_v3), np.array(y_test_v3)
```

```python
x_test_v3[0:2]
#len(x_train_v3)
```

```
(31, 8)
```

[62]:
```
array([[[0.25042589, 0.26190476, 0.61682243, 0.50943396, 0.32984293,
         0.7125    , 0.71717172, 0.44734182],
        [0.3032368 , 0.23809524, 0.62616822, 0.53459119, 0.36649215,
         0.7625    , 0.70707071, 0.63391955],
        [0.27086882, 0.26190476, 0.61682243, 0.54716981, 0.37172775,
         0.75      , 0.70707071, 0.82049727],
        [0.25894378, 0.26190476, 0.61682243, 0.55345912, 0.36649215,
         0.7375    , 0.70707071, 0.69102486],
        [0.31175468, 0.33333333, 0.55140187, 0.55345912, 0.2460733 ,
         0.6875    , 0.70707071, 0.56155246],
        [0.21124361, 0.30952381, 0.55140187, 0.56603774, 0.28795812,
         0.6       , 0.74747475, 0.43208005],
        [0.15332198, 0.26190476, 0.51401869, 0.55974843, 0.29319372,
         0.575     , 0.71717172, 0.30260764],
        [0.1669506 , 0.5       , 0.59813084, 0.47169811, 0.30890052,
         0.4875    , 0.23232323, 0.47180109],
        [0.18057922, 0.57142857, 0.45794393, 0.27044025, 0.2617801 ,
         0.425     , 0.19191919, 0.64099454],
        [0.18739353, 0.23809524, 0.62616822, 0.62264151, 0.34031414,
         0.75      , 0.70707071, 0.81018799],
        [0.29131175, 0.28571429, 0.59813084, 0.56603774, 0.31937173,
         0.7       , 0.6969697 , 0.73559733],
        [0.31856899, 0.30952381, 0.57009346, 0.56603774, 0.33507853,
         0.6875    , 0.72727273, 0.66100667],
        [0.27597956, 0.21428571, 0.62616822, 0.57232704, 0.36125654,
         0.725     , 0.77777778, 0.58641601],
        [0.24190801, 0.14285714, 0.6728972 , 0.83647799, 0.38219895,
         0.725     , 0.83838384, 0.51182535]],

       [[0.3032368 , 0.23809524, 0.62616822, 0.53459119, 0.36649215,
         0.7625    , 0.70707071, 0.63391955],
        [0.27086882, 0.26190476, 0.61682243, 0.54716981, 0.37172775,
         0.75      , 0.70707071, 0.82049727],
        [0.25894378, 0.26190476, 0.61682243, 0.55345912, 0.36649215,
         0.7375    , 0.70707071, 0.69102486],
        [0.31175468, 0.33333333, 0.55140187, 0.55345912, 0.2460733 ,
         0.6875    , 0.70707071, 0.56155246],
        [0.21124361, 0.30952381, 0.55140187, 0.56603774, 0.28795812,
         0.6       , 0.74747475, 0.43208005],
        [0.15332198, 0.26190476, 0.51401869, 0.55974843, 0.29319372,
         0.575     , 0.71717172, 0.30260764],
        [0.1669506 , 0.5       , 0.59813084, 0.47169811, 0.30890052,
```

```
 0.4875    , 0.23232323, 0.47180109],
[0.18057922, 0.57142857, 0.45794393, 0.27044025, 0.2617801 ,
 0.425     , 0.19191919, 0.64099454],
[0.18739353, 0.23809524, 0.62616822, 0.62264151, 0.34031414,
 0.75      , 0.70707071, 0.81018799],
[0.29131175, 0.28571429, 0.59813084, 0.56603774, 0.31937173,
 0.7       , 0.6969697 , 0.73559733],
[0.31856899, 0.30952381, 0.57009346, 0.56603774, 0.33507853,
 0.6875    , 0.72727273, 0.66100667],
[0.27597956, 0.21428571, 0.62616822, 0.57232704, 0.36125654,
 0.725     , 0.77777778, 0.58641601],
[0.24190801, 0.14285714, 0.6728972 , 0.83647799, 0.38219895,
 0.725     , 0.83838384, 0.51182535],
[0.2572402 , 0.21428571, 0.6728972 , 0.52201258, 0.32460733,
 0.7375    , 0.71717172, 0.63998383]]])
```

[63]:
```
y_test_v3[0:2]
```

[63]:
```
array([0.2572402 , 0.36797274])
```

[64]:
```
print(x_test_v3.shape, y_test_v3.shape)
```

```
(17, 14, 8) (17,)
```

[65]:
```
# Configure the neural network model
model_v3 = Sequential()

# Model with N "loop_back" Neurons
# Inputshape >>> loop_back Timestamps, each with x_train.shape[2] variables
n_neurons_v3 = x_train_v3.shape[1] * x_train_v3.shape[2]
print(n_neurons_v3, x_train_v3.shape[1], x_train_v3.shape[2])

model_v3.add(LSTM(n_neurons_v3,
                  activation='relu',
                  return_sequences=True,
                  input_shape=(x_train_v3.shape[1],
                               x_train_v3.shape[2])))
model_v3.add(LSTM(50, activation='relu', return_sequences=True))
model_v3.add(LSTM(25, activation='relu',return_sequences=False))
model_v3.add(Dense(5, activation='relu'))
model_v3.add(Dense(1))

# Compile the model
model_v3.compile(optimizer='adam', loss='mean_squared_error')
```

```
112 14 8
```

[66]:
```
# Training the model
early_stop_v3 = EarlyStopping(monitor='loss', patience=2, verbose=1)
history_v3 = model_v3.fit(x_train_v3,
                    y_train_v3,
                    batch_size=2,
                    validation_data=(x_test_v3, y_test_v3),
                    epochs=50
                    #callbacks=[early_stop_v2]
                         )
```

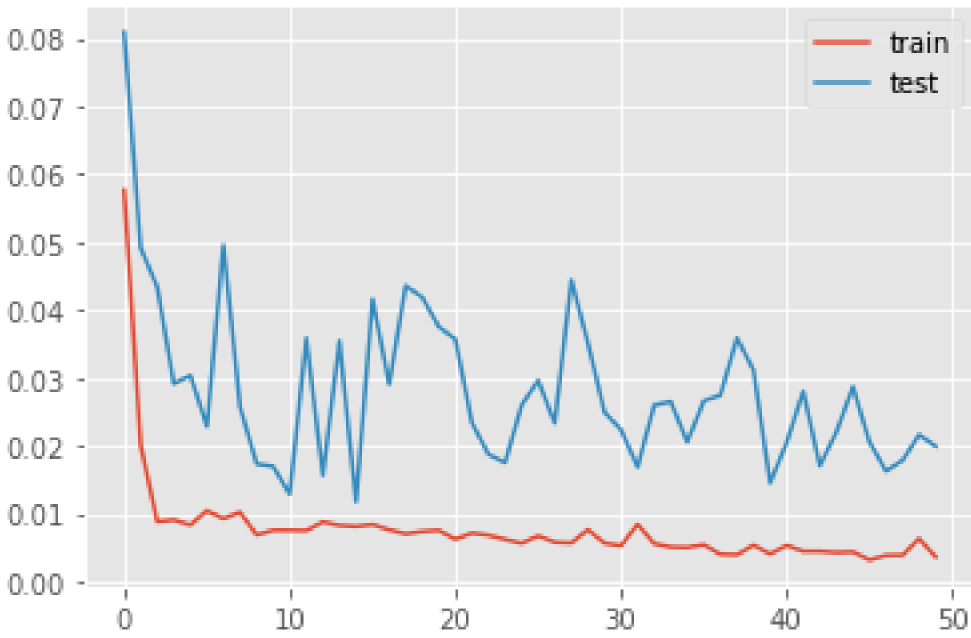
```
Epoch 1/50
130/130 [==============================] - 10s 24ms/step - loss: 0.0551 -
val_loss: 0.0809
Epoch 2/50
130/130 [==============================] - 4s 34ms/step - loss: 0.0318 -
val_loss: 0.0492
Epoch 3/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0092 -
val_loss: 0.0434
Epoch 4/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0089 -
val_loss: 0.0291
Epoch 5/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0078 -
val_loss: 0.0304
Epoch 6/50
130/130 [==============================] - 4s 31ms/step - loss: 0.0081 -
val_loss: 0.0229
Epoch 7/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0102 -
val_loss: 0.0496
Epoch 8/50
130/130 [==============================] - 4s 27ms/step - loss: 0.0111 -
val_loss: 0.0258
Epoch 9/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0081 -
val_loss: 0.0174
Epoch 10/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0076 -
val_loss: 0.0170
Epoch 11/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0062 -
val_loss: 0.0129
Epoch 12/50
130/130 [==============================] - 4s 30ms/step - loss: 0.0094 -
val_loss: 0.0359
Epoch 13/50
```

```
130/130 [==============================] - 4s 31ms/step - loss: 0.0089 -
val_loss: 0.0156
Epoch 14/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0096 -
val_loss: 0.0355
Epoch 15/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0080 -
val_loss: 0.0118
Epoch 16/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0075 -
val_loss: 0.0416
Epoch 17/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0090 -
val_loss: 0.0291
Epoch 18/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0046 -
val_loss: 0.0436
Epoch 19/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0064 -
val_loss: 0.0419
Epoch 20/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0087 -
val_loss: 0.0376- loss: 0.00
Epoch 21/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0071 -
val_loss: 0.0357
Epoch 22/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0059 -
val_loss: 0.0234
Epoch 23/50
130/130 [==============================] - 3s 19ms/step - loss: 0.0065 -
val_loss: 0.0188
Epoch 24/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0067 -
val_loss: 0.0175
Epoch 25/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0050 -
val_loss: 0.0260
Epoch 26/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0067 -
val_loss: 0.0296
Epoch 27/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0051 -
val_loss: 0.0233
Epoch 28/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0041 -
val_loss: 0.0445
Epoch 29/50
```

```
130/130 [==============================] - 2s 17ms/step - loss: 0.0082 -
val_loss: 0.0352
Epoch 30/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0052 -
val_loss: 0.0250
Epoch 31/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0048 -
val_loss: 0.0224
Epoch 32/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0100 -
val_loss: 0.0169
Epoch 33/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0048 -
val_loss: 0.0260
Epoch 34/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0060 -
val_loss: 0.0265
Epoch 35/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0046 -
val_loss: 0.0206
Epoch 36/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0064 -
val_loss: 0.0266
Epoch 37/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0030 -
val_loss: 0.0275
Epoch 38/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0026 -
val_loss: 0.0359
Epoch 39/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0059 -
val_loss: 0.0312
Epoch 40/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0038 -
val_loss: 0.0145
Epoch 41/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0048 -
val_loss: 0.0205
Epoch 42/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0035 -
val_loss: 0.0281
Epoch 43/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0050 -
val_loss: 0.0171
Epoch 44/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0038 -
val_loss: 0.0220
Epoch 45/50
```

```
130/130 [==============================] - 2s 17ms/step - loss: 0.0047 -
val_loss: 0.0287
Epoch 46/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0029 -
val_loss: 0.0206
Epoch 47/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0035 -
val_loss: 0.0163
Epoch 48/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0029 -
val_loss: 0.0179
Epoch 49/50
130/130 [==============================] - 4s 34ms/step - loss: 0.0069 -
val_loss: 0.0217
Epoch 50/50
130/130 [==============================] - 4s 30ms/step - loss: 0.0046 -
val_loss: 0.0199
```

[67]:
```
# Plot history
plt.plot(history_v3.history['loss'], label='train')
plt.plot(history_v3.history['val_loss'], label='test')
plt.legend()
plt.show()
```



[68]:
```
# Get the predicted values
predictions_v3 = model_v3.predict(x_test_v3)
```

```
predictions_v3
```

[68]:
```
array([[0.22447166],
       [0.22952017],
       [0.23865345],
       [0.23687044],
       [0.24292697],
       [0.23321125],
       [0.23039919],
       [0.25799808],
       [0.26973978],
       [0.27906656],
       [0.29576597],
       [0.2936736 ],
       [0.27171102],
       [0.26072407],
       [0.3059105 ],
       [0.32754675],
       [0.37200475]], dtype=float32)
```

[69]:
```
# Get the predicted values
pred_unscaled_v3 = scaler_v3_pred.inverse_transform(predictions_v3)
y_test_v3_unscaled = scaler_v3_pred.inverse_transform(y_test_v3.reshape(-1, 1))
```

[70]:
```
# Calculate the mean absolute error (MAE)
mae_v3 = mean_absolute_error(pred_unscaled_v3, y_test_v3_unscaled)
print('MAE: ' + str(round(mae_v3, 1)))

# Calculate the root mean squarred error (RMSE)
rmse_v3 = np.sqrt(mean_squared_error(y_test_v3_unscaled,pred_unscaled_v3))
print('RMSE: ' + str(round(rmse_v3, 1)))
```

```
MAE: 59.1
RMSE: 82.9
```

[71]:
```
# Date from which on the date is displayed
display_start_date_v3 = "2020-09-16"

# Add the difference between the valid and predicted prices
train_v3 = data_v3[:training_data_length_v3 + 1]
valid_v3 = data_v3[training_data_length_v3:]
```

[72]:
```
valid_v3.insert(1, "Predictions", pred_unscaled_v3, True)
valid_v3.insert(1, "Difference", valid_v3["Predictions"] - valid_v3["num_casos.
↪x"], True)
```

[73]:
```python
# Zoom-in to a closer timeframe
valid_v3 = valid_v3[valid_v3.index > display_start_date_v3]
train_v3 = train_v3[train_v3.index > display_start_date_v3]

# Show the test / valid and predicted prices
valid_v3
```

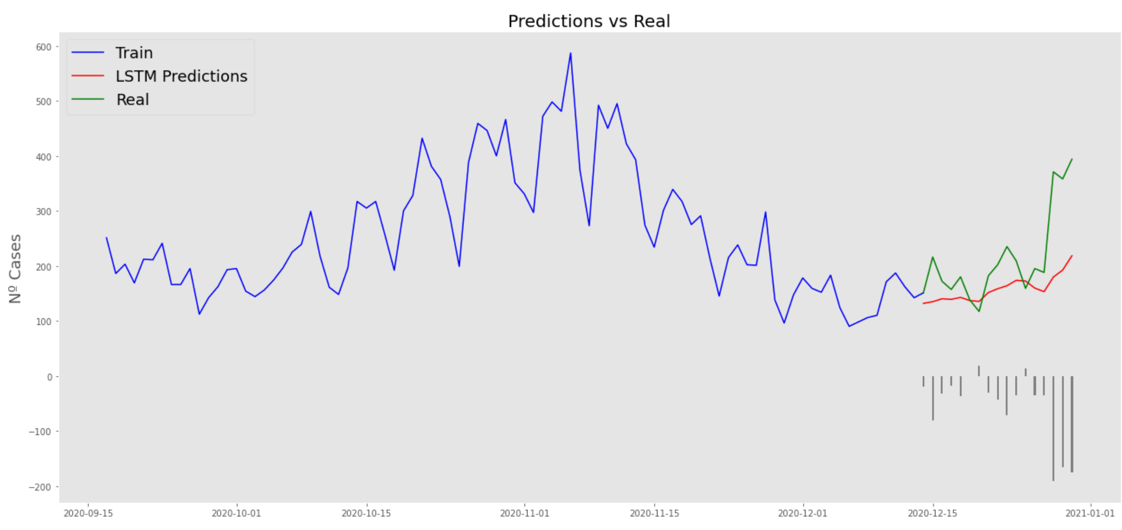
[73]:
```
            num_casos.x  Difference  Predictions  \
fecha
2020-12-14          151  -19.235138   131.764862
2020-12-15          216  -81.271652   134.728348
2020-12-16          172  -31.910431   140.089569
2020-12-17          157  -17.957047   139.042953
2020-12-18          180  -37.401871   142.598129
2020-12-19          138   -1.104996   136.895004
2020-12-20          117   18.244324   135.244324
2020-12-21          182  -30.555130   151.444870
2020-12-22          202  -43.662750   158.337250
2020-12-23          235  -71.187927   163.812073
2020-12-24          209  -35.385376   173.614624
2020-12-25          159   13.386414   172.386414
2020-12-26          195  -35.505630   159.494370
2020-12-27          188  -34.954971   153.045029
2020-12-28          371 -191.430542   179.569458
2020-12-29          358 -165.730057   192.269943
2020-12-30          394 -175.633209   218.366791

            residential_percent_change_from_baseline  \
fecha
2020-12-14                                       7.0
2020-12-15                                       6.0
2020-12-16                                       7.0
2020-12-17                                       7.0
2020-12-18                                       6.0
2020-12-19                                       6.0
2020-12-20                                       3.0
2020-12-21                                       5.0
2020-12-22                                       5.0
2020-12-23                                       6.0
2020-12-24                                       9.0
2020-12-25                                      21.0
2020-12-26                                       7.0
2020-12-27                                       3.0
2020-12-28                                      10.0
2020-12-29                                       9.0
2020-12-30                                       8.0
```

```
           retail_and_recreation_percent_change_from_baseline  \
fecha
2020-12-14                                              -25.0
2020-12-15                                              -22.0
2020-12-16                                              -26.0
2020-12-17                                              -23.0
2020-12-18                                              -23.0
2020-12-19                                              -27.0
2020-12-20                                              -17.0
2020-12-21                                              -12.0
2020-12-22                                              -11.0
2020-12-23                                               -9.0
2020-12-24                                              -27.0
2020-12-25                                              -68.0
2020-12-26                                              -30.0
2020-12-27                                              -19.0
2020-12-28                                              -17.0
2020-12-29                                              -12.0
2020-12-30                                              -11.0

           grocery_and_pharmacy_percent_change_from_baseline  \
fecha
2020-12-14                                               -8.0
2020-12-15                                               -2.0
2020-12-16                                               -2.0
2020-12-17                                                2.0
2020-12-18                                                2.0
2020-12-19                                               -2.0
2020-12-20                                               61.0
2020-12-21                                                5.0
2020-12-22                                               10.0
2020-12-23                                               26.0
2020-12-24                                                7.0
2020-12-25                                              -77.0
2020-12-26                                              -10.0
2020-12-27                                               48.0
2020-12-28                                               -2.0
2020-12-29                                                6.0
2020-12-30                                               23.0

           parks_percent_change_from_baseline  \
fecha
2020-12-14                              -30.0
2020-12-15                              -16.0
2020-12-16                              -26.0
2020-12-17                              -14.0
2020-12-18                              -14.0
```

```
2020-12-19                              -26.0
2020-12-20                              -20.0
2020-12-21                              -15.0
2020-12-22                               -8.0
2020-12-23                               -6.0
2020-12-24                              -17.0
2020-12-25                              -19.0
2020-12-26                              -13.0
2020-12-27                              -13.0
2020-12-28                              -25.0
2020-12-29                               -9.0
2020-12-30                               -5.0

            transit_stations_percent_change_from_baseline  \
fecha
2020-12-14                                          -34.0
2020-12-15                                          -27.0
2020-12-16                                          -30.0
2020-12-17                                          -28.0
2020-12-18                                          -24.0
2020-12-19                                          -26.0
2020-12-20                                          -26.0
2020-12-21                                          -23.0
2020-12-22                                          -20.0
2020-12-23                                          -18.0
2020-12-24                                          -41.0
2020-12-25                                          -68.0
2020-12-26                                          -32.0
2020-12-27                                          -27.0
2020-12-28                                          -33.0
2020-12-29                                          -26.0
2020-12-30                                          -26.0

            workplaces_percent_change_from_baseline      total
fecha
2020-12-14                                    -18.0  15.113333
2020-12-15                                    -18.0  17.226667
2020-12-16                                    -19.0  19.340000
2020-12-17                                    -19.0  17.800000
2020-12-18                                    -17.0  16.260000
2020-12-19                                    -12.0  14.720000
2020-12-20                                     -3.0  13.180000
2020-12-21                                    -21.0  15.073333
2020-12-22                                    -24.0  16.966667
2020-12-23                                    -33.0  18.860000
2020-12-24                                    -48.0  17.302500
2020-12-25                                    -78.0  15.745000
```

2020-12-26	-18.0	14.187500
2020-12-27	-5.0	12.630000
2020-12-28	-39.0	14.316667
2020-12-29	-39.0	16.003333
2020-12-30	-37.0	17.690000

[74]:
```python
# Visualize the data
fig, ax1 = plt.subplots(figsize=(22, 10), sharex=True)

# Data - Train
xt_v3 = train_v3.index;
yt_v3 = train_v3[["num_casos.x"]]
# Data - Test / validation
xv_v3 = valid_v3.index;
yv_v3 = valid_v3[["num_casos.x", "Predictions"]]

# Plot
plt.title("Predictions vs Real", fontsize=20)
plt.ylabel("Nº Cases", fontsize=18)

plt.plot(yt_v3, color="blue", linewidth=1.5)
plt.plot(yv_v3["Predictions"], color="red", linewidth=1.5)
plt.plot(yv_v3["num_casos.x"], color="green", linewidth=1.5)
plt.legend(["Train", "LSTM Predictions", "Real"],
           loc="upper left", fontsize=18)

# Bar plot with the differences
x_v3 = valid_v3.index
y_v3 = valid_v3["Difference"]
plt.bar(x_v3, y_v3, width=0.2, color="grey")
plt.grid()
plt.show()
```



```
[ ]:
```