LSTM_arodriguezsans-Univariate_Multivariate_Mad

May 18, 2021

1 Madrid

1.1 Load libraries needed

[1]:
```python
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]:
```python
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

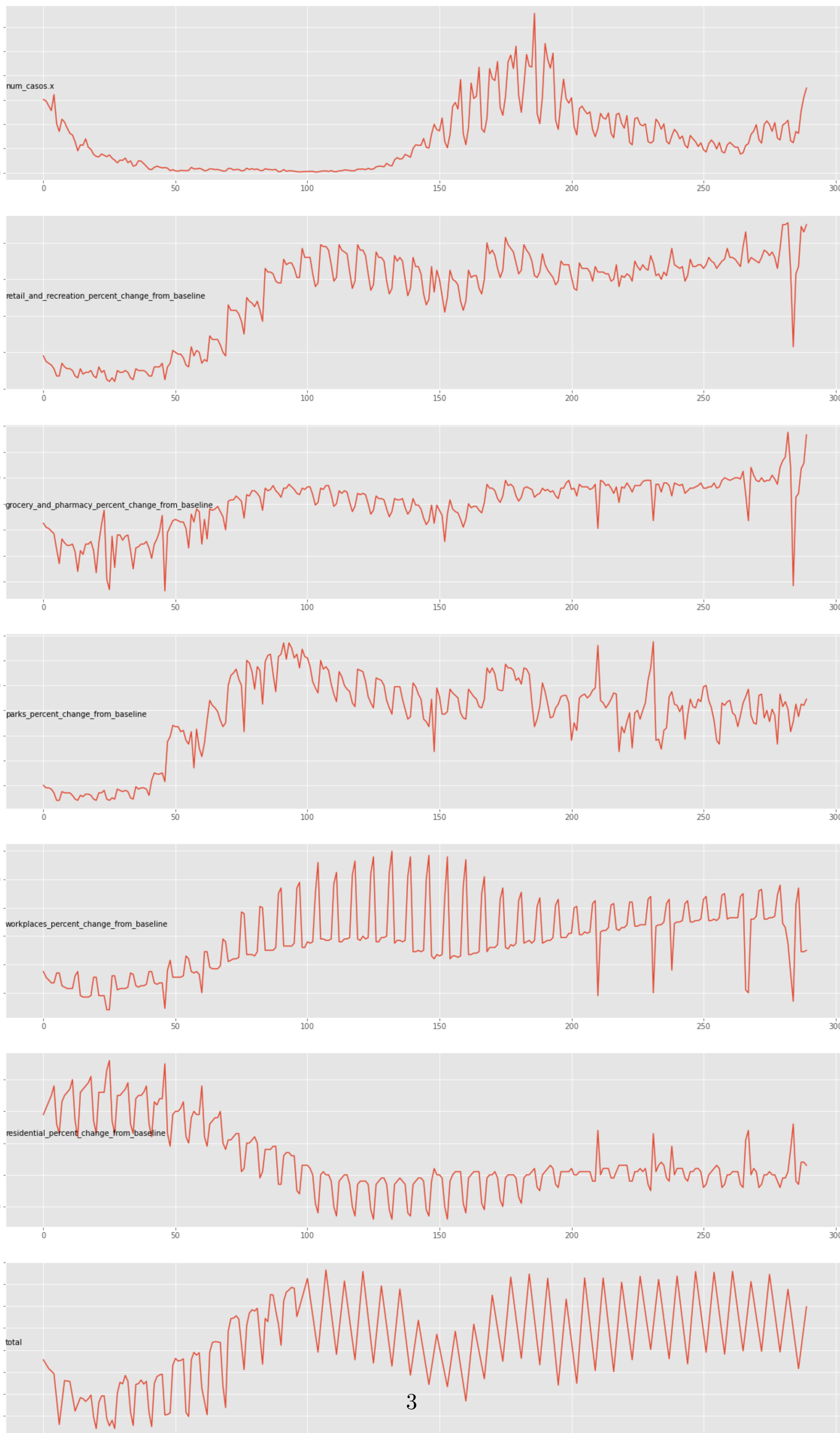
[3]:
```python
Mad=df_total.loc[df_total['sub_region_2'] == 'Madrid']
```

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

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

1.4 Plots

[6]:
```python
# 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(Mad.values[:, group])
    plt.title(Mad.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]:
```python
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data = Mad.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([[0.4569506 ],
       [0.4483866 ],
       [0.41718917],
       [0.3873681 ],
       [0.48830096]])
```

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

[8]:
```
array([[3014],
       [2958],
       [2754],
       [2559],
       [3219]], dtype=int64)
```

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

[9]: 290

[10]:
```python
training_data_length
```

[10]: 273

[11]:
```python
# 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([0.4569506 , 0.4483866 , 0.41718917, 0.3873681 , 0.48830096,
        0.30478666, 0.25600245, 0.3327726 , 0.31487995, 0.27725952,
        0.24407402, 0.23336902, 0.19131366, 0.13411837]),
 array([0.4483866 , 0.41718917, 0.3873681 , 0.48830096, 0.30478666,
        0.25600245, 0.3327726 , 0.31487995, 0.27725952, 0.24407402,
        0.23336902, 0.19131366, 0.13411837, 0.17112708])]
```

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

[14]:
```
[0.17112708365193455, 0.169139012081358]
```

[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])
```

```
[[0.4569506  0.4483866  0.41718917 0.3873681  0.48830096 0.30478666
  0.25600245 0.3327726  0.31487995 0.27725952 0.24407402 0.23336902
  0.19131366 0.13411837]
 [0.4483866  0.41718917 0.3873681  0.48830096 0.30478666 0.25600245
  0.3327726  0.31487995 0.27725952 0.24407402 0.23336902 0.19131366
  0.13411837 0.17112708]]
------------------------------------------------------------
[0.17112708 0.16913901]
```

[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]:
```python
x_train[0:2]
```

[17]:
```
array([[[0.4569506 ],
        [0.4483866 ],
        [0.41718917],
        [0.3873681 ],
        [0.48830096],
        [0.30478666],
        [0.25600245],
        [0.3327726 ],
        [0.31487995],
        [0.27725952],
        [0.24407402],
        [0.23336902],
        [0.19131366],
        [0.13411837]],

       [[0.4483866 ],
        [0.41718917],
        [0.3873681 ],
        [0.48830096],
        [0.30478666],
        [0.25600245],
        [0.3327726 ],
        [0.31487995],
        [0.27725952],
        [0.24407402],
        [0.23336902],
        [0.19131366],
        [0.13411837],
        [0.17112708]]])
```

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

[18]:
```
array([0.17112708, 0.16913901])
```

[19]:
```python
# 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])
```

```
[[[0.17036244]
  [0.18718458]
  [0.17158587]
  [0.15614008]
  [0.15721058]
  [0.11286129]
  [0.12142529]
  [0.16806851]
  [0.17862058]
  [0.23658052]
  [0.25768466]
  [0.29836366]
  [0.2038538 ]
  [0.18229087]]

 [[0.18718458]
  [0.17158587]
  [0.15614008]
  [0.15721058]
  [0.11286129]
  [0.12142529]
  [0.16806851]
  [0.17862058]
  [0.23658052]
  [0.25768466]
  [0.29836366]
  [0.2038538 ]
  [0.18229087]
  [0.29943416]]]
------------------------------------------------------------
```

```
[0.29943416 0.32206759]
```

[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)
```

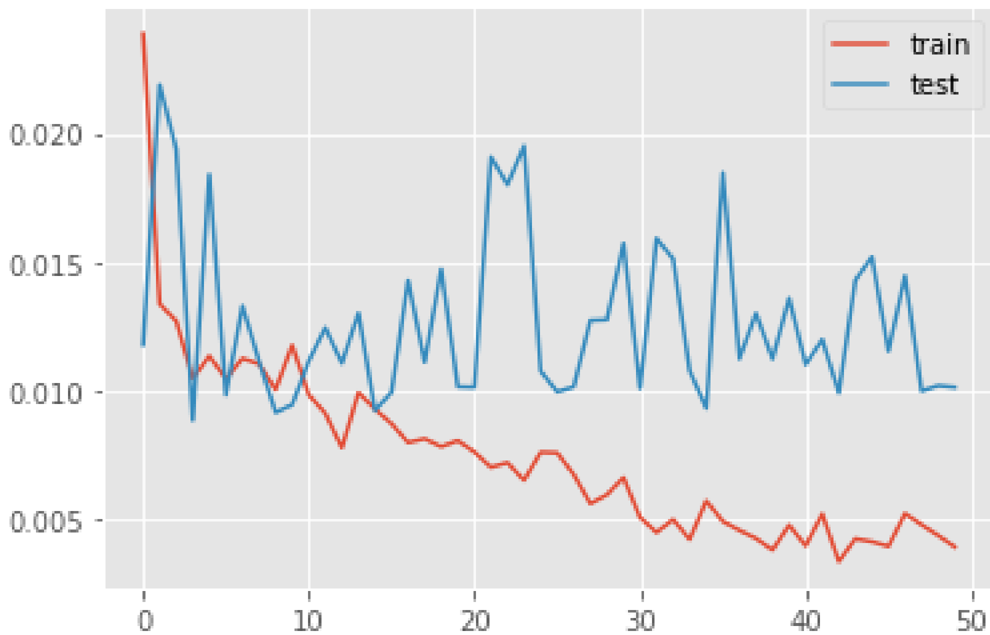
```
Epoch 1/50
130/130 - 12s - loss: 0.0239 - val_loss: 0.0118
Epoch 2/50
130/130 - 3s - loss: 0.0134 - val_loss: 0.0219
Epoch 3/50
130/130 - 2s - loss: 0.0127 - val_loss: 0.0194
Epoch 4/50
130/130 - 2s - loss: 0.0105 - val_loss: 0.0088
Epoch 5/50
130/130 - 2s - loss: 0.0113 - val_loss: 0.0184
Epoch 6/50
130/130 - 2s - loss: 0.0105 - val_loss: 0.0098
Epoch 7/50
130/130 - 3s - loss: 0.0112 - val_loss: 0.0133
Epoch 8/50
130/130 - 2s - loss: 0.0110 - val_loss: 0.0111
```

```
Epoch 9/50
130/130 - 2s - loss: 0.0100 - val_loss: 0.0091
Epoch 10/50
130/130 - 2s - loss: 0.0118 - val_loss: 0.0094
Epoch 11/50
130/130 - 2s - loss: 0.0098 - val_loss: 0.0111
Epoch 12/50
130/130 - 2s - loss: 0.0091 - val_loss: 0.0124
Epoch 13/50
130/130 - 2s - loss: 0.0078 - val_loss: 0.0111
Epoch 14/50
130/130 - 2s - loss: 0.0099 - val_loss: 0.0130
Epoch 15/50
130/130 - 2s - loss: 0.0093 - val_loss: 0.0092
Epoch 16/50
130/130 - 2s - loss: 0.0087 - val_loss: 0.0099
Epoch 17/50
130/130 - 2s - loss: 0.0080 - val_loss: 0.0143
Epoch 18/50
130/130 - 2s - loss: 0.0081 - val_loss: 0.0111
Epoch 19/50
130/130 - 2s - loss: 0.0078 - val_loss: 0.0147
Epoch 20/50
130/130 - 2s - loss: 0.0081 - val_loss: 0.0101
Epoch 21/50
130/130 - 2s - loss: 0.0076 - val_loss: 0.0101
Epoch 22/50
130/130 - 2s - loss: 0.0070 - val_loss: 0.0191
Epoch 23/50
130/130 - 2s - loss: 0.0072 - val_loss: 0.0180
Epoch 24/50
130/130 - 2s - loss: 0.0065 - val_loss: 0.0195
Epoch 25/50
130/130 - 2s - loss: 0.0076 - val_loss: 0.0108
Epoch 26/50
130/130 - 2s - loss: 0.0076 - val_loss: 0.0099
Epoch 27/50
130/130 - 2s - loss: 0.0067 - val_loss: 0.0101
Epoch 28/50
130/130 - 2s - loss: 0.0056 - val_loss: 0.0127
Epoch 29/50
130/130 - 2s - loss: 0.0059 - val_loss: 0.0128
Epoch 30/50
130/130 - 2s - loss: 0.0066 - val_loss: 0.0157
Epoch 31/50
130/130 - 2s - loss: 0.0051 - val_loss: 0.0101
Epoch 32/50
130/130 - 2s - loss: 0.0045 - val_loss: 0.0159
```

```
Epoch 33/50
130/130 - 2s - loss: 0.0050 - val_loss: 0.0151
Epoch 34/50
130/130 - 2s - loss: 0.0042 - val_loss: 0.0108
Epoch 35/50
130/130 - 2s - loss: 0.0057 - val_loss: 0.0093
Epoch 36/50
130/130 - 2s - loss: 0.0049 - val_loss: 0.0185
Epoch 37/50
130/130 - 2s - loss: 0.0046 - val_loss: 0.0112
Epoch 38/50
130/130 - 2s - loss: 0.0042 - val_loss: 0.0130
Epoch 39/50
130/130 - 2s - loss: 0.0038 - val_loss: 0.0112
Epoch 40/50
130/130 - 2s - loss: 0.0048 - val_loss: 0.0136
Epoch 41/50
130/130 - 2s - loss: 0.0040 - val_loss: 0.0110
Epoch 42/50
130/130 - 2s - loss: 0.0052 - val_loss: 0.0120
Epoch 43/50
130/130 - 2s - loss: 0.0033 - val_loss: 0.0099
Epoch 44/50
130/130 - 2s - loss: 0.0042 - val_loss: 0.0143
Epoch 45/50
130/130 - 2s - loss: 0.0041 - val_loss: 0.0152
Epoch 46/50
130/130 - 2s - loss: 0.0039 - val_loss: 0.0115
Epoch 47/50
130/130 - 2s - loss: 0.0052 - val_loss: 0.0145
Epoch 48/50
130/130 - 3s - loss: 0.0048 - val_loss: 0.0100
Epoch 49/50
130/130 - 2s - loss: 0.0044 - val_loss: 0.0102
Epoch 50/50
130/130 - 2s - loss: 0.0039 - val_loss: 0.0101
```

[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([[1819.023 ],
       [2130.1182],
       [2302.509 ],
       [2383.669 ],
       [2374.3674],
       [2209.901 ],
       [2053.8474],
       [2018.8616],
       [1946.8655],
       [2041.8727],
       [2036.5544],
       [2040.9673],
       [1663.7659],
       [1442.6469],
       [1702.0618],
       [1907.1042],
       [2015.448 ]], dtype=float32)
```

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

[26]:
```
array([[1984.],
       [2132.],
       [1982.],
       [1704.],
       [2064.],
       [1426.],
       [1345.],
       [1962.],
       [2036.],
       [2158.],
       [1313.],
       [1234.],
       [1692.],
       [1620.],
       [2555.],
       [3105.],
       [3485.]])
```

[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: 499.3
RMSE: 657.9
```

[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]
```

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

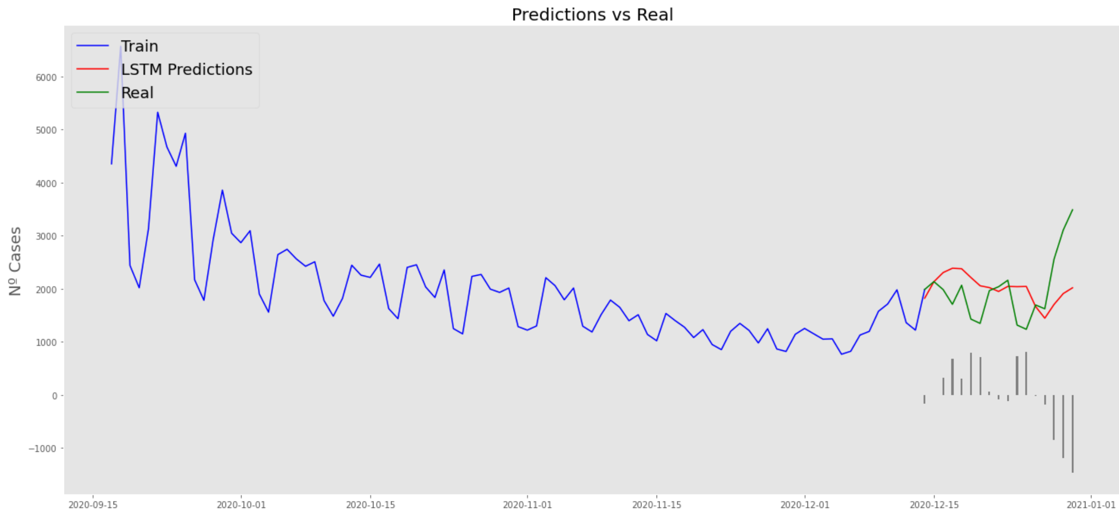
[30]:

```
            num_casos.x   Difference  Predictions
fecha
2020-12-14         1984  -164.977051  1819.022949
2020-12-15         2132    -1.881836  2130.118164
2020-12-16         1982   320.509033  2302.509033
2020-12-17         1704   679.668945  2383.668945
2020-12-18         2064   310.367432  2374.367432
2020-12-19         1426   783.900879  2209.900879
2020-12-20         1345   708.847412  2053.847412
2020-12-21         1962    56.861572  2018.861572
2020-12-22         2036   -89.134521  1946.865479
2020-12-23         2158  -116.127319  2041.872681
2020-12-24         1313   723.554443  2036.554443
2020-12-25         1234   806.967285  2040.967285
2020-12-26         1692   -28.234131  1663.765869
2020-12-27         1620  -177.353149  1442.646851
2020-12-28         2555  -852.938232  1702.061768
2020-12-29         3105 -1197.895752  1907.104248
2020-12-30         3485 -1469.552002  2015.447998
```

[31]:

```python
# 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 = Mad.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([[0.4569506 , 0.66      , 0.43362832],
       [0.4483866 , 0.7       , 0.40625   ],
       [0.41718917, 0.74      , 0.37887168],
       [0.3873681 , 0.78      , 0.36255531],
       [0.48830096, 0.84      , 0.34623894]])
```

[34]:
```python
# 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(Mad['num_casos.x'])
np_cases_scaled_v2 = scaler_v2_pred.fit_transform(df_cases)
np_cases_scaled_v2[0:5]
```

[34]:
```
array([[0.4569506 ],
       [0.4483866 ],
       [0.41718917],
       [0.3873681 ],
       [0.48830096]])
```

[35]:
```python
# Create the training data
train_data_v2 = scaled_data_v2[0:training_data_length_v2, :]
print(train_data_v2.shape)
```

```
(273, 3)
```

[36]:
```python
train_data_v2[0:2]
```

[36]:
```
array([[0.4569506 , 0.66      , 0.43362832],
       [0.4483866 , 0.7       , 0.40625   ]])
```

[37]:
```python
training_data_length_v2
```

[37]:
```
273
```

[38]:
```python
loop_back
```

[38]:
```
14
```

[39]:
```python
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]
```

[39]:
```
array([[[0.4569506 , 0.66      , 0.43362832],
        [0.4483866 , 0.7       , 0.40625   ],
        [0.41718917, 0.74      , 0.37887168],
        [0.3873681 , 0.78      , 0.36255531],
        [0.48830096, 0.84      , 0.34623894],
        [0.30478666, 0.6       , 0.18611726],
        [0.25600245, 0.54      , 0.02599558],
        [0.3327726 , 0.74      , 0.16454646],
        [0.31487995, 0.78      , 0.30309735],
        [0.27725952, 0.8       , 0.30033186],
        [0.24407402, 0.82      , 0.29756637],
        [0.23336902, 0.88      , 0.20436947],
        [0.19131366, 0.62      , 0.11117257],
        [0.13411837, 0.52      , 0.15348451]],

       [[0.4483866 , 0.7       , 0.40625   ],
        [0.41718917, 0.74      , 0.37887168],
        [0.3873681 , 0.78      , 0.36255531],
        [0.48830096, 0.84      , 0.34623894],
        [0.30478666, 0.6       , 0.18611726],
        [0.25600245, 0.54      , 0.02599558],
        [0.3327726 , 0.74      , 0.16454646],
        [0.31487995, 0.78      , 0.30309735],
        [0.27725952, 0.8       , 0.30033186],
        [0.24407402, 0.82      , 0.29756637],
        [0.23336902, 0.88      , 0.20436947],
        [0.19131366, 0.62      , 0.11117257],
        [0.13411837, 0.52      , 0.15348451],
        [0.17112708, 0.8       , 0.19579646]]])
```

[40]:
```
y_train_v2[0:2]
```

[40]:
```
array([0.17112708, 0.16913901])
```

[41]:
```
print(x_train_v2.shape, y_train_v2.shape)
```

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

[42]:
```
# 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)
```

```
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)
```

[42]:
```
array([[[0.17036244, 0.28      , 0.64362094],
        [0.18718458, 0.28      , 0.81766224],
        [0.17158587, 0.3       , 0.99170354],
        [0.15614008, 0.3       , 0.86628872],
        [0.15721058, 0.3       , 0.74087389],
        [0.11286129, 0.22      , 0.61545907],
        [0.12142529, 0.2       , 0.49004425],
        [0.16806851, 0.5       , 0.63569322],
        [0.17862058, 0.56      , 0.78134218],
        [0.23658052, 0.28      , 0.92699115],
        [0.25768466, 0.32      , 0.81706305],
        [0.29836366, 0.3       , 0.70713496],
        [0.2038538 , 0.2       , 0.59720686],
        [0.18229087, 0.22      , 0.48727876]],

       [[0.18718458, 0.28      , 0.81766224],
        [0.17158587, 0.3       , 0.99170354],
        [0.15614008, 0.3       , 0.86628872],
        [0.15721058, 0.3       , 0.74087389],
        [0.11286129, 0.22      , 0.61545907],
        [0.12142529, 0.2       , 0.49004425],
        [0.16806851, 0.5       , 0.63569322],
        [0.17862058, 0.56      , 0.78134218],
        [0.23658052, 0.28      , 0.92699115],
        [0.25768466, 0.32      , 0.81706305],
        [0.29836366, 0.3       , 0.70713496],
        [0.2038538 , 0.2       , 0.59720686],
        [0.18229087, 0.22      , 0.48727876],
        [0.29943416, 0.28      , 0.64896755]]])
```

[43]:
```
y_test_v2[0:2]
```

[43]:
```
array([0.29943416, 0.32206759])
```

[44]:
```
print(x_test_v2.shape, y_test_v2.shape)
```

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

[45]:
```
# 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
```

[46]:
```
# 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 [==============================] - 10s 22ms/step - loss: 0.0264 -
val_loss: 0.0151
Epoch 2/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0115 -
val_loss: 0.0145
Epoch 3/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0090 -
val_loss: 0.0136
Epoch 4/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0081 -
```

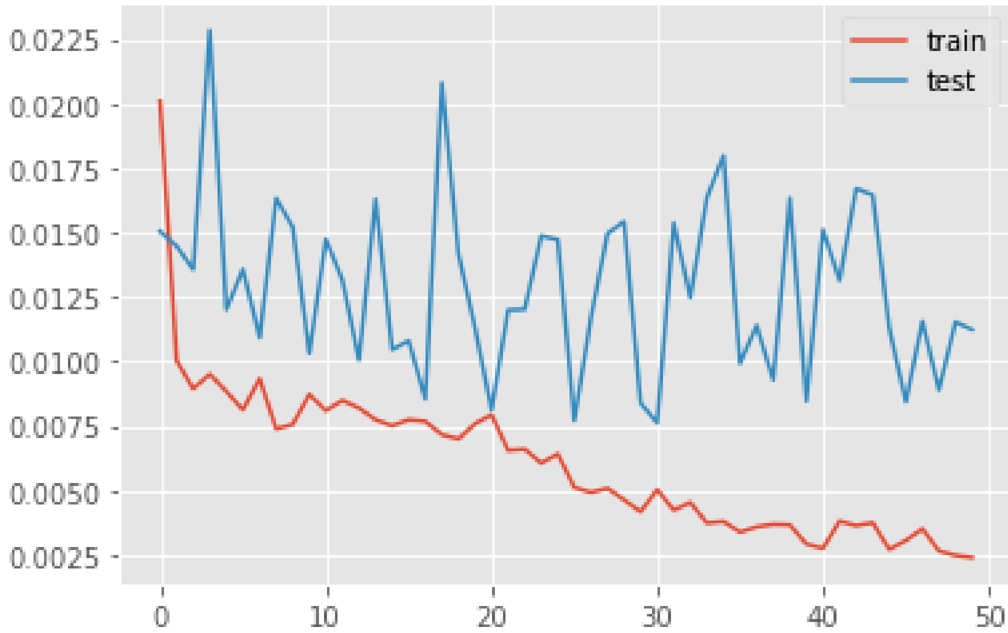
```
val_loss: 0.0229
Epoch 5/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0088 -
val_loss: 0.0120
Epoch 6/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0080 -
val_loss: 0.0136
Epoch 7/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0075 -
val_loss: 0.0109
Epoch 8/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0068 -
val_loss: 0.0163
Epoch 9/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0067 -
val_loss: 0.0152
Epoch 10/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0092 -
val_loss: 0.0103
Epoch 11/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0089 -
val_loss: 0.0147
Epoch 12/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0112 -
val_loss: 0.0132
Epoch 13/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0078 -
val_loss: 0.0101
Epoch 14/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0106 -
val_loss: 0.0163
Epoch 15/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0069 -
val_loss: 0.0105
Epoch 16/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0071 -
val_loss: 0.0108
Epoch 17/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0066 -
val_loss: 0.0086
Epoch 18/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0082 -
val_loss: 0.0208
Epoch 19/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0075 -
val_loss: 0.0142
Epoch 20/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0068 -
```

```
val_loss: 0.0113
Epoch 21/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0057 -
val_loss: 0.0081
Epoch 22/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0079 -
val_loss: 0.0120
Epoch 23/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0055 -
val_loss: 0.0120
Epoch 24/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0079 -
val_loss: 0.0149
Epoch 25/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0054 -
val_loss: 0.0147
Epoch 26/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0051 -
val_loss: 0.0077
Epoch 27/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0039 -
val_loss: 0.0117
Epoch 28/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0052 -
val_loss: 0.0150
Epoch 29/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0049 -
val_loss: 0.0154
Epoch 30/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0052 -
val_loss: 0.0084
Epoch 31/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0043 -
val_loss: 0.0076
Epoch 32/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0057 -
val_loss: 0.0154
Epoch 33/50
130/130 [==============================] - 3s 19ms/step - loss: 0.0033 -
val_loss: 0.0125
Epoch 34/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0034 -
val_loss: 0.0164
Epoch 35/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0043 -
val_loss: 0.0180
Epoch 36/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0037 -
```

```
val_loss: 0.0099
Epoch 37/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0036 -
val_loss: 0.0114
Epoch 38/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0032 -
val_loss: 0.0093
Epoch 39/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0039 -
val_loss: 0.0163
Epoch 40/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0024 -
val_loss: 0.0085
Epoch 41/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0018 -
val_loss: 0.0151
Epoch 42/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0042 -
val_loss: 0.0132
Epoch 43/50
130/130 [==============================] - 3s 19ms/step - loss: 0.0031 -
val_loss: 0.0167
Epoch 44/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0034 -
val_loss: 0.0165
Epoch 45/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0023 -
val_loss: 0.0113
Epoch 46/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0027 -
val_loss: 0.0084
Epoch 47/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0031 -
val_loss: 0.0116
Epoch 48/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0026 -
val_loss: 0.0089
Epoch 49/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0027 -
val_loss: 0.0115
Epoch 50/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0030 -
val_loss: 0.0112
```

[47]:
```
# Plot history
plt.plot(history_v2.history['loss'], label='train')
plt.plot(history_v2.history['val_loss'], label='test')
```

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



[48]:
```
# Get the predicted values
predictions_v2 = model_v2.predict(x_test_v2)
predictions_v2
```

[48]:
```
array([[0.18166205],
       [0.20725884],
       [0.22302486],
       [0.23229797],
       [0.2187805 ],
       [0.17981257],
       [0.16391294],
       [0.24567704],
       [0.27510265],
       [0.30017307],
       [0.30900496],
       [0.3050025 ],
       [0.20926845],
       [0.18603739],
       [0.24476385],
       [0.28222364],
       [0.29648367]], dtype=float32)
```

[49]:
```
# 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))
```

[50]:
```
# 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: 579.0
RMSE: 693.4
```

[51]:
```
mean_absolute_error(y_test_v2_unscaled, pred_unscaled_v2)
np.sqrt(mean_squared_error(y_test_v2_unscaled,pred_unscaled_v2))
```

[51]: 693.4279522799272

[52]:
```
# 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:]
```

[53]:
```
valid_v2.insert(1, "Predictions", pred_unscaled_v2, True)
valid_v2.insert(1, "Difference", valid_v2["Predictions"] - valid_v2["num_casos.
↪x"], True)
```

[54]:
```
# 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
```

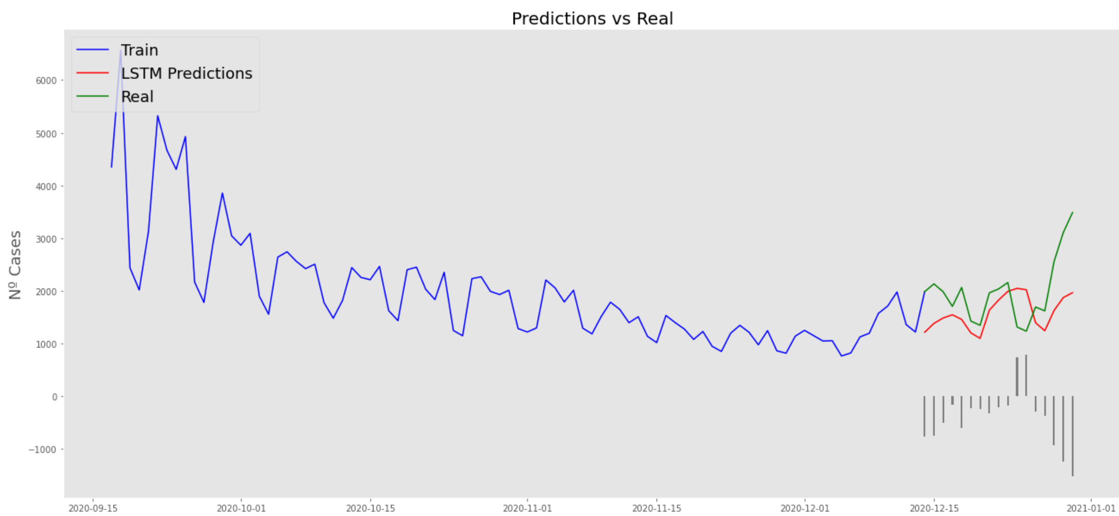
[54]:
```
            num_casos.x  Difference  Predictions  \
fecha
2020-12-14         1984 -770.111938  1213.888062
2020-12-15         2132 -750.734497  1381.265503
2020-12-16         1982 -497.640503  1484.359497
2020-12-17         1704 -159.003662  1544.996338
2020-12-18         2064 -607.394287  1456.605713
2020-12-19         1426 -224.205688  1201.794312
2020-12-20         1345 -247.173340  1097.826660
```

```
2020-12-21          1962  -329.517944  1632.482056
2020-12-22          2036  -211.103882  1824.896118
2020-12-23          2158  -169.168335  1988.831665
2020-12-24          1313   733.583374  2046.583374
2020-12-25          1234   786.411377  2020.411377
2020-12-26          1692  -297.593628  1394.406372
2020-12-27          1620  -377.501587  1242.498413
2020-12-28          2555  -928.489258  1626.510742
2020-12-29          3105 -1233.539673  1871.460327
2020-12-30          3485 -1520.293335  1964.706665

            residential_percent_change_from_baseline      total
fecha
2020-12-14                                      10.0  17.763333
2020-12-15                                      10.0  20.686667
2020-12-16                                      11.0  23.610000
2020-12-17                                      10.0  21.402500
2020-12-18                                      10.0  19.195000
2020-12-19                                       8.0  16.987500
2020-12-20                                       6.0  14.780000
2020-12-21                                       9.0  17.156667
2020-12-22                                       9.0  19.533333
2020-12-23                                      11.0  21.910000
2020-12-24                                      18.0  19.650000
2020-12-25                                      26.0  17.390000
2020-12-26                                       8.0  15.130000
2020-12-27                                       7.0  12.870000
2020-12-28                                      14.0  15.220000
2020-12-29                                      14.0  17.570000
2020-12-30                                      13.0  19.920000
```

[55]:

```
# 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)
```

```python
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

```python
[56]: # New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data_v3 = Mad.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]
```

[56]:
```
array([[0.4569506 , 0.66      , 0.16091954, 0.42622951, 0.09448819,
        0.26865672, 0.24107143, 0.43362832],
       [0.4483866 , 0.7       , 0.12643678, 0.40163934, 0.07874016,
        0.20895522, 0.20535714, 0.40625   ],
       [0.41718917, 0.74      , 0.11494253, 0.39344262, 0.07874016,
        0.17910448, 0.1875    , 0.37887168],
       [0.3873681 , 0.78      , 0.10344828, 0.37704918, 0.07086614,
        0.1641791 , 0.16964286, 0.36255531],
       [0.48830096, 0.84      , 0.08045977, 0.36065574, 0.04724409,
        0.14925373, 0.16964286, 0.34623894]])
```

[57]:
```
# 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(Mad['num_casos.x'])
np_cases_scaled_v3 = scaler_v3_pred.fit_transform(df_cases)
np_cases_scaled_v3[0:5]
```

[57]:
```
array([[0.4569506 ],
       [0.4483866 ],
       [0.41718917],
       [0.3873681 ],
       [0.48830096]])
```

[58]:
```
# Create the training data
train_data_v3 = scaled_data_v3[0:training_data_length_v3, :]
print(train_data_v3.shape)
```

```
(273, 8)
```

[59]:
```
train_data_v3[0:2]
```

[59]:
```
array([[0.4569506 , 0.66      , 0.16091954, 0.42622951, 0.09448819,
        0.26865672, 0.24107143, 0.43362832],
       [0.4483866 , 0.7       , 0.12643678, 0.40163934, 0.07874016,
        0.20895522, 0.20535714, 0.40625   ]])
```

[60]:
```
training_data_length_v3
```

[60]:
```
273
```

[61]:
```
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):
    #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]
```

[61]:
```
array([[[0.4569506 , 0.66      , 0.16091954, 0.42622951, 0.09448819,
         0.26865672, 0.24107143, 0.43362832],
        [0.4483866 , 0.7       , 0.12643678, 0.40163934, 0.07874016,
         0.20895522, 0.20535714, 0.40625   ],
        [0.41718917, 0.74      , 0.11494253, 0.39344262, 0.07874016,
         0.17910448, 0.1875    , 0.37887168],
        [0.3873681 , 0.78      , 0.10344828, 0.37704918, 0.07086614,
         0.1641791 , 0.16964286, 0.36255531],
        [0.48830096, 0.84      , 0.08045977, 0.36065574, 0.04724409,
         0.14925373, 0.16964286, 0.34623894],
        [0.30478666, 0.6       , 0.03448276, 0.26229508, 0.        ,
         0.08955224, 0.23214286, 0.18611726],
        [0.25600245, 0.54      , 0.03448276, 0.17213115, 0.        ,
         0.07462687, 0.23214286, 0.02599558],
        [0.3327726 , 0.74      , 0.11494253, 0.32786885, 0.05511811,
         0.14925373, 0.15178571, 0.16454646],
        [0.31487995, 0.78      , 0.09195402, 0.30327869, 0.04724409,
         0.13432836, 0.14285714, 0.30309735],
        [0.27725952, 0.8       , 0.08045977, 0.28688525, 0.04724409,
         0.11940299, 0.13392857, 0.30033186],
        [0.24407402, 0.82      , 0.08045977, 0.28688525, 0.04724409,
         0.11940299, 0.13392857, 0.29756637],
        [0.23336902, 0.88      , 0.06896552, 0.29508197, 0.03149606,
         0.10447761, 0.13392857, 0.20436947],
        [0.19131366, 0.62      , 0.03448276, 0.24590164, 0.00787402,
         0.08955224, 0.21428571, 0.11117257],
        [0.13411837, 0.52      , 0.02298851, 0.12295082, 0.        ,
         0.05970149, 0.24107143, 0.15348451]],

       [[0.4483866 , 0.7       , 0.12643678, 0.40163934, 0.07874016,
         0.20895522, 0.20535714, 0.40625   ],
```

```
        [0.41718917, 0.74      , 0.11494253, 0.39344262, 0.07874016,
         0.17910448, 0.1875    , 0.37887168],
        [0.3873681 , 0.78      , 0.10344828, 0.37704918, 0.07086614,
         0.1641791 , 0.16964286, 0.36255531],
        [0.48830096, 0.84      , 0.08045977, 0.36065574, 0.04724409,
         0.14925373, 0.16964286, 0.34623894],
        [0.30478666, 0.6       , 0.03448276, 0.26229508, 0.        ,
         0.08955224, 0.23214286, 0.18611726],
        [0.25600245, 0.54      , 0.03448276, 0.17213115, 0.        ,
         0.07462687, 0.23214286, 0.02599558],
        [0.3327726 , 0.74      , 0.11494253, 0.32786885, 0.05511811,
         0.14925373, 0.15178571, 0.16454646],
        [0.31487995, 0.78      , 0.09195402, 0.30327869, 0.04724409,
         0.13432836, 0.14285714, 0.30309735],
        [0.27725952, 0.8       , 0.08045977, 0.28688525, 0.04724409,
         0.11940299, 0.13392857, 0.30033186],
        [0.24407402, 0.82      , 0.08045977, 0.28688525, 0.04724409,
         0.11940299, 0.13392857, 0.29756637],
        [0.23336902, 0.88      , 0.06896552, 0.29508197, 0.03149606,
         0.10447761, 0.13392857, 0.20436947],
        [0.19131366, 0.62      , 0.03448276, 0.24590164, 0.00787402,
         0.08955224, 0.21428571, 0.11117257],
        [0.13411837, 0.52      , 0.02298851, 0.12295082, 0.        ,
         0.05970149, 0.24107143, 0.15348451],
        [0.17112708, 0.8       , 0.08045977, 0.25409836, 0.03149606,
         0.08955224, 0.08928571, 0.19579646]]])
```

[62]:
```
y_train_v3[0:2]
```

[62]:
```
array([0.17112708, 0.16913901])
```

[63]:
```
print(x_train_v3.shape, y_train_v3.shape)
```

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

[64]:
```
# 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
```

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

```
(31, 8)
```

[64]:
```
array([[[0.17036244, 0.28      , 0.83908046, 0.70491803, 0.62204724,
         0.92537313, 0.57142857, 0.64362094],
        [0.18718458, 0.28      , 0.7816092 , 0.69672131, 0.62992126,
         0.92537313, 0.58035714, 0.81766224],
        [0.17158587, 0.3       , 0.7816092 , 0.70491803, 0.5511811 ,
         0.91044776, 0.58035714, 0.99170354],
        [0.15614008, 0.3       , 0.77011494, 0.71311475, 0.53543307,
         0.89552239, 0.58035714, 0.86628872],
        [0.15721058, 0.3       , 0.74712644, 0.71311475, 0.46456693,
         0.89552239, 0.58035714, 0.74087389],
        [0.11286129, 0.22      , 0.72413793, 0.70491803, 0.53543307,
         0.92537313, 0.71428571, 0.61545907],
        [0.12142529, 0.2       , 0.85057471, 0.75409836, 0.61417323,
         0.89552239, 0.73214286, 0.49004425],
        [0.16806851, 0.5       , 0.94252874, 0.60655738, 0.65354331,
         0.59701493, 0.125     , 0.63569322],
        [0.17862058, 0.56      , 0.74712644, 0.44262295, 0.7007874 ,
         0.53731343, 0.10714286, 0.78134218],
        [0.23658052, 0.28      , 0.7816092 , 0.77868852, 0.53543307,
         0.89552239, 0.57142857, 0.92699115],
        [0.25768466, 0.32      , 0.77011494, 0.7295082 , 0.48818898,
         0.88059701, 0.57142857, 0.81706305],
        [0.29836366, 0.3       , 0.75862069, 0.69672131, 0.48031496,
         0.91044776, 0.58928571, 0.70713496],
        [0.2038538 , 0.2       , 0.74712644, 0.68852459, 0.66141732,
         1.        , 0.75      , 0.59720686],
        [0.18229087, 0.22      , 0.7816092 , 0.71311475, 0.66929134,
         0.88059701, 0.75892857, 0.48727876]],

       [[0.18718458, 0.28      , 0.7816092 , 0.69672131, 0.62992126,
         0.92537313, 0.58035714, 0.81766224],
        [0.17158587, 0.3       , 0.7816092 , 0.70491803, 0.5511811 ,
         0.91044776, 0.58035714, 0.99170354],
        [0.15614008, 0.3       , 0.77011494, 0.71311475, 0.53543307,
         0.89552239, 0.58035714, 0.86628872],
        [0.15721058, 0.3       , 0.74712644, 0.71311475, 0.46456693,
```

```
        0.89552239, 0.58035714, 0.74087389],
       [0.11286129, 0.22      , 0.72413793, 0.70491803, 0.53543307,
        0.92537313, 0.71428571, 0.61545907],
       [0.12142529, 0.2       , 0.85057471, 0.75409836, 0.61417323,
        0.89552239, 0.73214286, 0.49004425],
       [0.16806851, 0.5       , 0.94252874, 0.60655738, 0.65354331,
        0.59701493, 0.125     , 0.63569322],
       [0.17862058, 0.56      , 0.74712644, 0.44262295, 0.7007874 ,
        0.53731343, 0.10714286, 0.78134218],
       [0.23658052, 0.28      , 0.7816092 , 0.77868852, 0.53543307,
        0.89552239, 0.57142857, 0.92699115],
       [0.25768466, 0.32      , 0.77011494, 0.7295082 , 0.48818898,
        0.88059701, 0.57142857, 0.81706305],
       [0.29836366, 0.3       , 0.75862069, 0.69672131, 0.48031496,
        0.91044776, 0.58928571, 0.70713496],
       [0.2038538 , 0.2       , 0.74712644, 0.68852459, 0.66141732,
        1.        , 0.75      , 0.59720686],
       [0.18229087, 0.22      , 0.7816092 , 0.71311475, 0.66929134,
        0.88059701, 0.75892857, 0.48727876],
       [0.29943416, 0.28      , 0.82758621, 0.68852459, 0.51968504,
        0.92537313, 0.57142857, 0.64896755]]])
```

[65]:
```
y_test_v3[0:2]
```

[65]:
```
array([0.29943416, 0.32206759])
```

[66]:
```
print(x_test_v3.shape, y_test_v3.shape)
```

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

[67]:
```
# 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
```

[68]:
```
# 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]
                         )
```

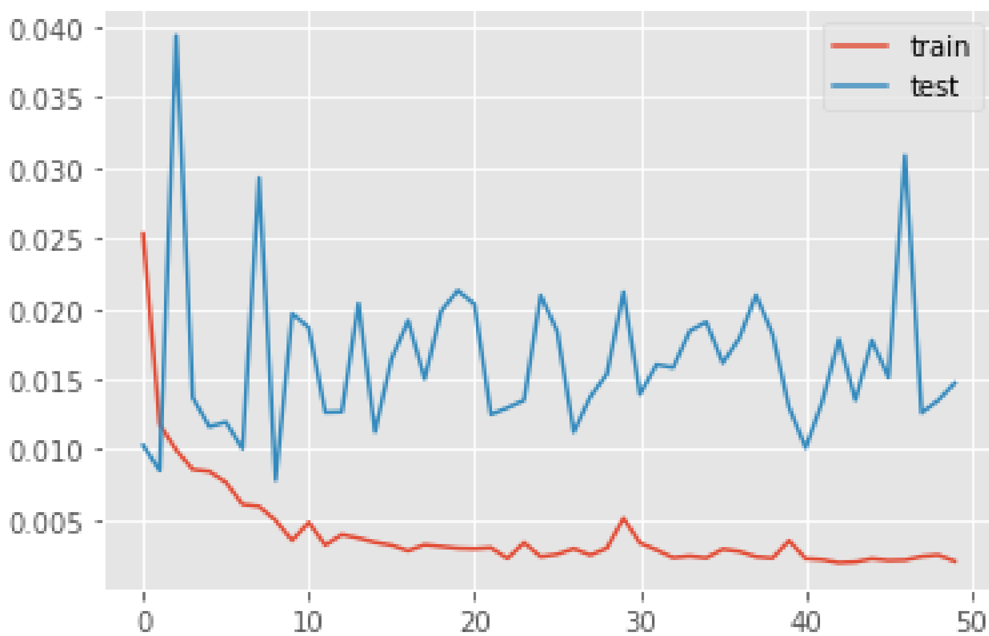
```
Epoch 1/50
130/130 [==============================] - 10s 23ms/step - loss: 0.0376 -
val_loss: 0.0103
Epoch 2/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0113 -
val_loss: 0.0085
Epoch 3/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0096 -
val_loss: 0.0394
Epoch 4/50
130/130 [==============================] - ETA: 0s - loss: 0.0096 - 3s 25ms/step
- loss: 0.0096 - val_loss: 0.0137
Epoch 5/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0087 -
val_loss: 0.0116
Epoch 6/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0075 -
val_loss: 0.0120
Epoch 7/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0077 -
val_loss: 0.0101
Epoch 8/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0068 -
val_loss: 0.0293
Epoch 9/50
130/130 [==============================] - 3s 19ms/step - loss: 0.0053 -
val_loss: 0.0078
Epoch 10/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0043 -
val_loss: 0.0197
Epoch 11/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0045 -
```

```
val_loss: 0.0187
Epoch 12/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0025 -
val_loss: 0.0127
Epoch 13/50
130/130 [==============================] - 3s 26ms/step - loss: 0.0047 -
val_loss: 0.0127
Epoch 14/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0044 -
val_loss: 0.0204
Epoch 15/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0029 -
val_loss: 0.0113
Epoch 16/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0032 -
val_loss: 0.0165
Epoch 17/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0023 -
val_loss: 0.0192
Epoch 18/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0040 -
val_loss: 0.0150
Epoch 19/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0029 -
val_loss: 0.0198
Epoch 20/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0023 -
val_loss: 0.0213
Epoch 21/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0033 -
val_loss: 0.0203
Epoch 22/50
130/130 [==============================] - 4s 32ms/step - loss: 0.0028 -
val_loss: 0.0125
Epoch 23/50
130/130 [==============================] - 3s 19ms/step - loss: 0.0034 -
val_loss: 0.0130
Epoch 24/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0041 -
val_loss: 0.0135
Epoch 25/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0030 -
val_loss: 0.0210
Epoch 26/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0024 -
val_loss: 0.0184
Epoch 27/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0027 -
```

```
val_loss: 0.0113
Epoch 28/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0024 -
val_loss: 0.0138 0s - loss: 0.00
Epoch 29/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0028 -
val_loss: 0.0154
Epoch 30/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0076 -
val_loss: 0.0212
Epoch 31/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0033 -
val_loss: 0.0139
Epoch 32/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0029 -
val_loss: 0.0160
Epoch 33/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0031 -
val_loss: 0.0158
Epoch 34/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0021 -
val_loss: 0.0184
Epoch 35/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0019 -
val_loss: 0.0191
Epoch 36/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0020 -
val_loss: 0.0162
Epoch 37/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0023 -
val_loss: 0.0179
Epoch 38/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0046 -
val_loss: 0.0210
Epoch 39/50
130/130 [==============================] - 4s 28ms/step - loss: 0.0019 -
val_loss: 0.0182
Epoch 40/50
130/130 [==============================] - 4s 29ms/step - loss: 0.0047 -
val_loss: 0.0130
Epoch 41/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0017 -
val_loss: 0.0101
Epoch 42/50
130/130 [==============================] - 3s 27ms/step - loss: 0.0023 -
val_loss: 0.0134
Epoch 43/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0022 -
```

```
val_loss: 0.0178
Epoch 44/50
130/130 [==============================] - 4s 27ms/step - loss: 0.0018 -
val_loss: 0.0135
Epoch 45/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0022 -
val_loss: 0.0178
Epoch 46/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0018 -
val_loss: 0.0151
Epoch 47/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0014 -
val_loss: 0.0309
Epoch 48/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0031 -
val_loss: 0.0126
Epoch 49/50
130/130 [==============================] - 4s 30ms/step - loss: 0.0023 -
val_loss: 0.0135
Epoch 50/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0023 -
val_loss: 0.0147
```

[69]:
```
# Plot history
plt.plot(history_v3.history['loss'], label='train')
plt.plot(history_v3.history['val_loss'], label='test')
plt.legend()
plt.show()
```



[70]:
```
# Get the predicted values
predictions_v3 = model_v3.predict(x_test_v3)
predictions_v3
```

[70]:
```
array([[0.17464152],
       [0.19089714],
       [0.20474234],
       [0.20114093],
       [0.1985077 ],
       [0.17207125],
       [0.16147244],
       [0.26272416],
       [0.2736012 ],
       [0.22000238],
       [0.22974896],
       [0.2301571 ],
       [0.18734385],
       [0.18080507],
       [0.22070475],
       [0.2314376 ],
       [0.24732636]], dtype=float32)
```

[71]:
```
# 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))
```

[72]:
```
# 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: 642.5
RMSE: 793.5
```

[73]:
```
# 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:]
```

[74]:
```python
valid_v3.insert(1, "Predictions", pred_unscaled_v3, True)
valid_v3.insert(1, "Difference", valid_v3["Predictions"] - valid_v3["num_casos.
↪x"], True)
```

[75]:
```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
```

[75]:
```
            num_casos.x   Difference  Predictions  \
fecha
2020-12-14         1984  -816.019165  1167.980835
2020-12-15         2132  -857.723633  1274.276367
2020-12-16         1982  -617.189819  1364.810181
2020-12-17         1704  -362.739502  1341.260498
2020-12-18         2064  -739.958252  1324.041748
2020-12-19         1426  -274.826172  1151.173828
2020-12-20         1345  -263.131714  1081.868286
2020-12-21         1962  -218.046753  1743.953247
2020-12-22         2036  -220.921753  1815.078247
2020-12-23         2158  -693.404419  1464.595581
2020-12-24         1313   215.328491  1528.328491
2020-12-25         1234   296.997314  1530.997314
2020-12-26         1692  -440.958618  1251.041382
2020-12-27         1620  -411.715698  1208.284302
2020-12-28         2555 -1085.811646  1469.188354
2020-12-29         3105 -1565.629639  1539.370361
2020-12-30         3485 -1841.732910  1643.267090

            residential_percent_change_from_baseline  \
fecha
2020-12-14                                      10.0
2020-12-15                                      10.0
2020-12-16                                      11.0
2020-12-17                                      10.0
2020-12-18                                      10.0
2020-12-19                                       8.0
2020-12-20                                       6.0
2020-12-21                                       9.0
2020-12-22                                       9.0
2020-12-23                                      11.0
2020-12-24                                      18.0
2020-12-25                                      26.0
2020-12-26                                       8.0
2020-12-27                                       7.0
```

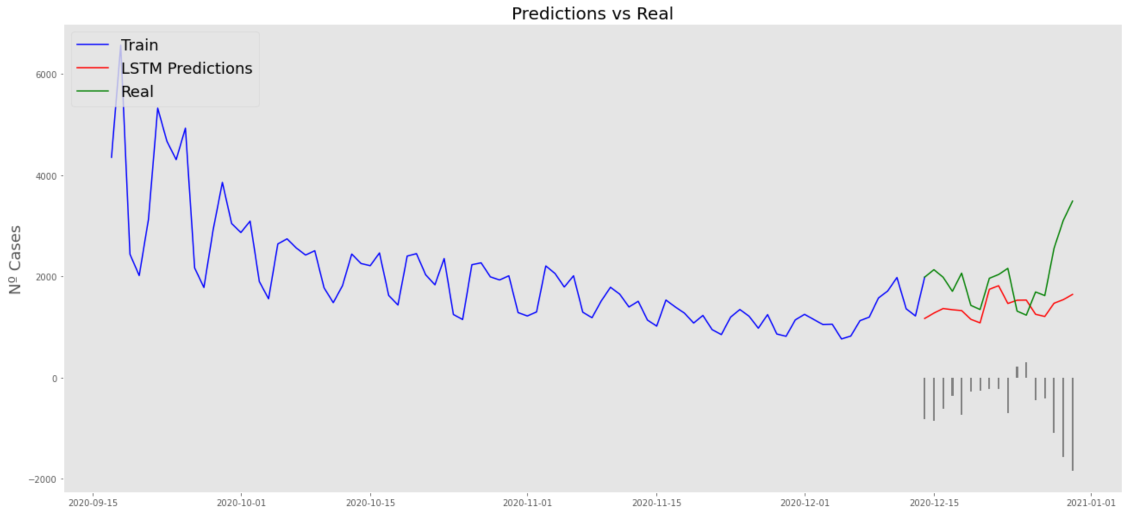
```
2020-12-28                                    14.0
2020-12-29                                    14.0
2020-12-30                                    13.0

            retail_and_recreation_percent_change_from_baseline  \
fecha
2020-12-14                                              -24.0
2020-12-15                                              -25.0
2020-12-16                                              -27.0
2020-12-17                                              -25.0
2020-12-18                                              -28.0
2020-12-19                                              -34.0
2020-12-20                                              -23.0
2020-12-21                                              -10.0
2020-12-22                                              -10.0
2020-12-23                                               -9.0
2020-12-24                                              -35.0
2020-12-25                                              -77.0
2020-12-26                                              -37.0
2020-12-27                                              -33.0
2020-12-28                                              -11.0
2020-12-29                                              -14.0
2020-12-30                                              -10.0

            grocery_and_pharmacy_percent_change_from_baseline  \
fecha
2020-12-14                                              -3.0
2020-12-15                                              -2.0
2020-12-16                                              -2.0
2020-12-17                                               2.0
2020-12-18                                              -1.0
2020-12-19                                              -5.0
2020-12-20                                               8.0
2020-12-21                                              13.0
2020-12-22                                              16.0
2020-12-23                                              35.0
2020-12-24                                               8.0
2020-12-25                                             -83.0
2020-12-26                                             -15.0
2020-12-27                                             -12.0
2020-12-28                                               7.0
2020-12-29                                              11.0
2020-12-30                                              33.0

            parks_percent_change_from_baseline  \
fecha
2020-12-14                               -26.0
```

```
2020-12-15                                -20.0
2020-12-16                                -29.0
2020-12-17                                -19.0
2020-12-18                                -24.0
2020-12-19                                -47.0
2020-12-20                                 -7.0
2020-12-21                                -17.0
2020-12-22                                -13.0
2020-12-23                                -19.0
2020-12-24                                -37.0
2020-12-25                                -29.0
2020-12-26                                -15.0
2020-12-27                                -25.0
2020-12-28                                -15.0
2020-12-29                                -16.0
2020-12-30                                -11.0

            transit_stations_percent_change_from_baseline  \
fecha
2020-12-14                                          -31.0
2020-12-15                                          -31.0
2020-12-16                                          -32.0
2020-12-17                                          -31.0
2020-12-18                                          -29.0
2020-12-19                                          -31.0
2020-12-20                                          -29.0
2020-12-21                                          -29.0
2020-12-22                                          -30.0
2020-12-23                                          -34.0
2020-12-24                                          -54.0
2020-12-25                                          -72.0
2020-12-26                                          -37.0
2020-12-27                                          -41.0
2020-12-28                                          -40.0
2020-12-29                                          -41.0
2020-12-30                                          -39.0

            workplaces_percent_change_from_baseline      total
fecha
2020-12-14                                    -28.0  17.763333
2020-12-15                                    -28.0  20.686667
2020-12-16                                    -27.0  23.610000
2020-12-17                                    -27.0  21.402500
2020-12-18                                    -26.0  19.195000
2020-12-19                                    -11.0  16.987500
2020-12-20                                     -4.0  14.780000
2020-12-21                                    -31.0  17.156667
```

```
2020-12-22                                      -34.0  19.533333
2020-12-23                                      -45.0  21.910000
2020-12-24                                      -66.0  19.650000
2020-12-25                                      -86.0  17.390000
2020-12-26                                      -18.0  15.130000
2020-12-27                                       -6.0  12.870000
2020-12-28                                      -51.0  15.220000
2020-12-29                                      -51.0  17.570000
2020-12-30                                      -50.0  19.920000
```

[76]:

```
# 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()
```


Predictions vs Real
Train
LSTM Predictions
Real
Nº Cases
6000
4000
2000
0
−2000
2020-09-15
2020-10-01
2020-10-15
2020-11-01
2020-11-15
2020-12-01
2020-12-15
2021-01-01

[ ]: