LSTM_arodriguezsans-Univariate_Multivariate_Cad

May 18, 2021

1 Cádiz

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]:
```
Cad=df_total.loc[df_total['sub_region_2'] == 'Cádiz']
```

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

[5]:
```
# We select columns of interest (mobility ones)
Cad=Cad[['num_casos.x'] + list(Cad.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(Cad.values[:, group])
    plt.title(Cad.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 = Cad.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([[0.10801964],
       [0.08837971],
       [0.07692308],
       [0.09001637],
       [0.10638298]])
```

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

[8]:
```
array([[66],
       [54],
       [47],
       [55],
       [65]], 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
```

```
# BS - 5
#MAE: 377.5
#RMSE: 103.3

# BS - 2
#MAE: 275.1 / 281.4 / 275.1
#RMSE: 64.7 / 98.3 / 76.3

# BS - 1
#MAE: 275.6
#RMSE: 111.7
```

[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([0.10801964, 0.08837971, 0.07692308, 0.09001637, 0.10638298,
        0.101473  , 0.06382979, 0.08510638, 0.07692308, 0.06382979,
        0.05728314, 0.04909984, 0.04909984, 0.05400982]),
 array([0.08837971, 0.07692308, 0.09001637, 0.10638298, 0.101473  ,
        0.06382979, 0.08510638, 0.07692308, 0.06382979, 0.05728314,
        0.04909984, 0.04909984, 0.05400982, 0.05728314])]
```

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

[14]:
```
[0.05728314238952537, 0.03273322422258593]
```

[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])
```

```
[[0.10801964 0.08837971 0.07692308 0.09001637 0.10638298 0.101473
  0.06382979 0.08510638 0.07692308 0.06382979 0.05728314 0.04909984
  0.04909984 0.05400982]
 [0.08837971 0.07692308 0.09001637 0.10638298 0.101473   0.06382979
  0.08510638 0.07692308 0.06382979 0.05728314 0.04909984 0.04909984
  0.05400982 0.05728314]]
-----------------------------------------------------------
[0.05728314 0.03273322]
```

[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([[[0.10801964],
        [0.08837971],
        [0.07692308],
        [0.09001637],
        [0.10638298],
        [0.101473  ],
        [0.06382979],
        [0.08510638],
        [0.07692308],
        [0.06382979],
        [0.05728314],
        [0.04909984],
        [0.04909984],
        [0.05400982]],

       [[0.08837971],
        [0.07692308],
        [0.09001637],
        [0.10638298],
        [0.101473  ],
        [0.06382979],
        [0.08510638],
        [0.07692308],
        [0.06382979],
        [0.05728314],
        [0.04909984],
        [0.04909984],
        [0.05400982],
```

```
[0.05728314]]])
```

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

[18]:
```
array([0.05728314, 0.03273322])
```

[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 = []
#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])
```

```
[[[0.3207856 ]
  [0.41898527]
  [0.41571195]
  [0.31914894]
  [0.36497545]
  [0.25695581]
  [0.1898527 ]
  [0.19312602]
  [0.20785597]
  [0.26677578]
  [0.31260229]
  [0.31423895]
  [0.2913257 ]
  [0.18330606]]

 [[0.41898527]
```

```
  [0.41571195]
  [0.31914894]
  [0.36497545]
  [0.25695581]
  [0.1898527 ]
  [0.19312602]
  [0.20785597]
  [0.26677578]
  [0.31260229]
  [0.31423895]
  [0.2913257 ]
  [0.18330606]
  [0.22913257]]]
-------------------------------------------------------------
[0.22913257 0.25859247]
```

[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)
```

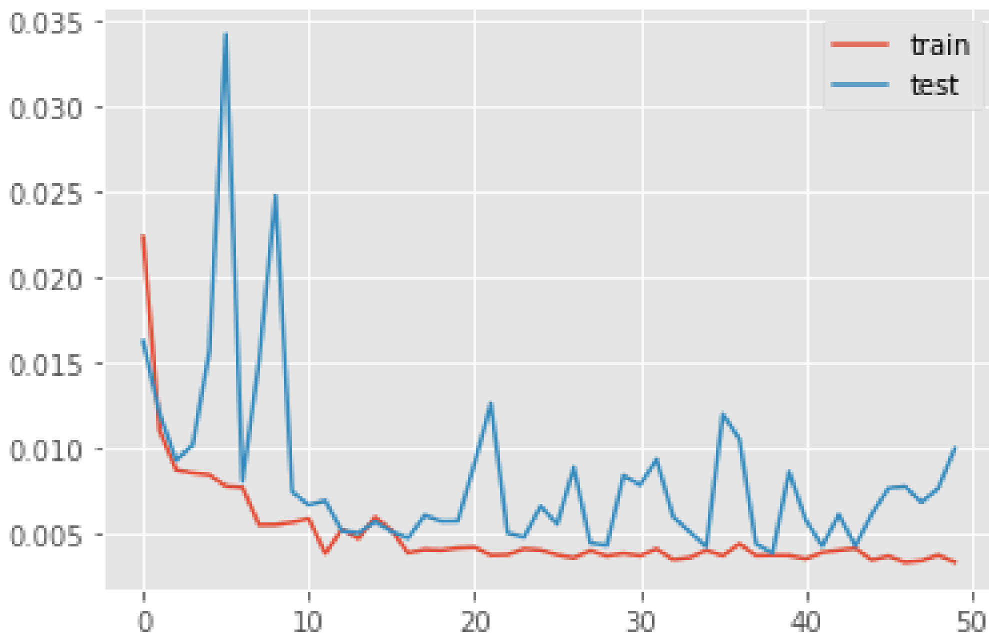
```
Epoch 1/50
130/130 - 11s - loss: 0.0224 - val_loss: 0.0163
Epoch 2/50
```

```
130/130 - 2s - loss: 0.0110 - val_loss: 0.0120
Epoch 3/50
130/130 - 2s - loss: 0.0087 - val_loss: 0.0093
Epoch 4/50
130/130 - 2s - loss: 0.0086 - val_loss: 0.0102
Epoch 5/50
130/130 - 2s - loss: 0.0085 - val_loss: 0.0157
Epoch 6/50
130/130 - 2s - loss: 0.0078 - val_loss: 0.0342
Epoch 7/50
130/130 - 2s - loss: 0.0077 - val_loss: 0.0081
Epoch 8/50
130/130 - 2s - loss: 0.0055 - val_loss: 0.0151
Epoch 9/50
130/130 - 2s - loss: 0.0055 - val_loss: 0.0248
Epoch 10/50
130/130 - 2s - loss: 0.0057 - val_loss: 0.0075
Epoch 11/50
130/130 - 2s - loss: 0.0059 - val_loss: 0.0067
Epoch 12/50
130/130 - 2s - loss: 0.0038 - val_loss: 0.0069
Epoch 13/50
130/130 - 2s - loss: 0.0053 - val_loss: 0.0052
Epoch 14/50
130/130 - 2s - loss: 0.0047 - val_loss: 0.0050
Epoch 15/50
130/130 - 2s - loss: 0.0060 - val_loss: 0.0057
Epoch 16/50
130/130 - 2s - loss: 0.0052 - val_loss: 0.0051
Epoch 17/50
130/130 - 2s - loss: 0.0039 - val_loss: 0.0048
Epoch 18/50
130/130 - 2s - loss: 0.0041 - val_loss: 0.0061
Epoch 19/50
130/130 - 2s - loss: 0.0040 - val_loss: 0.0057
Epoch 20/50
130/130 - 2s - loss: 0.0042 - val_loss: 0.0058
Epoch 21/50
130/130 - 2s - loss: 0.0042 - val_loss: 0.0091
Epoch 22/50
130/130 - 2s - loss: 0.0038 - val_loss: 0.0126
Epoch 23/50
130/130 - 2s - loss: 0.0038 - val_loss: 0.0051
Epoch 24/50
130/130 - 2s - loss: 0.0041 - val_loss: 0.0048
Epoch 25/50
130/130 - 2s - loss: 0.0040 - val_loss: 0.0066
Epoch 26/50
```

```
130/130 - 2s - loss: 0.0038 - val_loss: 0.0056
Epoch 27/50
130/130 - 2s - loss: 0.0036 - val_loss: 0.0089
Epoch 28/50
130/130 - 2s - loss: 0.0040 - val_loss: 0.0045
Epoch 29/50
130/130 - 2s - loss: 0.0037 - val_loss: 0.0043
Epoch 30/50
130/130 - 2s - loss: 0.0039 - val_loss: 0.0084
Epoch 31/50
130/130 - 2s - loss: 0.0037 - val_loss: 0.0079
Epoch 32/50
130/130 - 2s - loss: 0.0041 - val_loss: 0.0094
Epoch 33/50
130/130 - 2s - loss: 0.0035 - val_loss: 0.0060
Epoch 34/50
130/130 - 2s - loss: 0.0036 - val_loss: 0.0051
Epoch 35/50
130/130 - 2s - loss: 0.0040 - val_loss: 0.0043
Epoch 36/50
130/130 - 2s - loss: 0.0037 - val_loss: 0.0120
Epoch 37/50
130/130 - 2s - loss: 0.0044 - val_loss: 0.0106
Epoch 38/50
130/130 - 2s - loss: 0.0037 - val_loss: 0.0044
Epoch 39/50
130/130 - 2s - loss: 0.0038 - val_loss: 0.0039
Epoch 40/50
130/130 - 2s - loss: 0.0037 - val_loss: 0.0087
Epoch 41/50
130/130 - 2s - loss: 0.0035 - val_loss: 0.0058
Epoch 42/50
130/130 - 2s - loss: 0.0039 - val_loss: 0.0043
Epoch 43/50
130/130 - 2s - loss: 0.0040 - val_loss: 0.0061
Epoch 44/50
130/130 - 2s - loss: 0.0042 - val_loss: 0.0043
Epoch 45/50
130/130 - 2s - loss: 0.0035 - val_loss: 0.0062
Epoch 46/50
130/130 - 2s - loss: 0.0037 - val_loss: 0.0077
Epoch 47/50
130/130 - 2s - loss: 0.0033 - val_loss: 0.0078
Epoch 48/50
130/130 - 2s - loss: 0.0034 - val_loss: 0.0069
Epoch 49/50
130/130 - 2s - loss: 0.0038 - val_loss: 0.0077
Epoch 50/50
```

```
130/130 - 2s - loss: 0.0033 - val_loss: 0.0100
```

[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([[129.53065],
       [128.90921],
       [133.7313 ],
       [143.30293],
       [147.26   ],
       [149.41261],
       [145.43532],
       [140.08368],
       [137.42294],
       [139.32263],
       [149.39384],
       [163.36841],
```

```
        [174.81549],
        [184.48738],
        [194.52112],
        [217.66708],
        [248.45952]], dtype=float32)
```

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

[26]:
```
array([[140.],
       [158.],
       [153.],
       [157.],
       [190.],
       [146.],
       [138.],
       [171.],
       [191.],
       [222.],
       [195.],
       [178.],
       [211.],
       [256.],
       [353.],
       [329.],
       [305.]])
```

[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: 46.3
RMSE: 61.1
```

[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:]
```

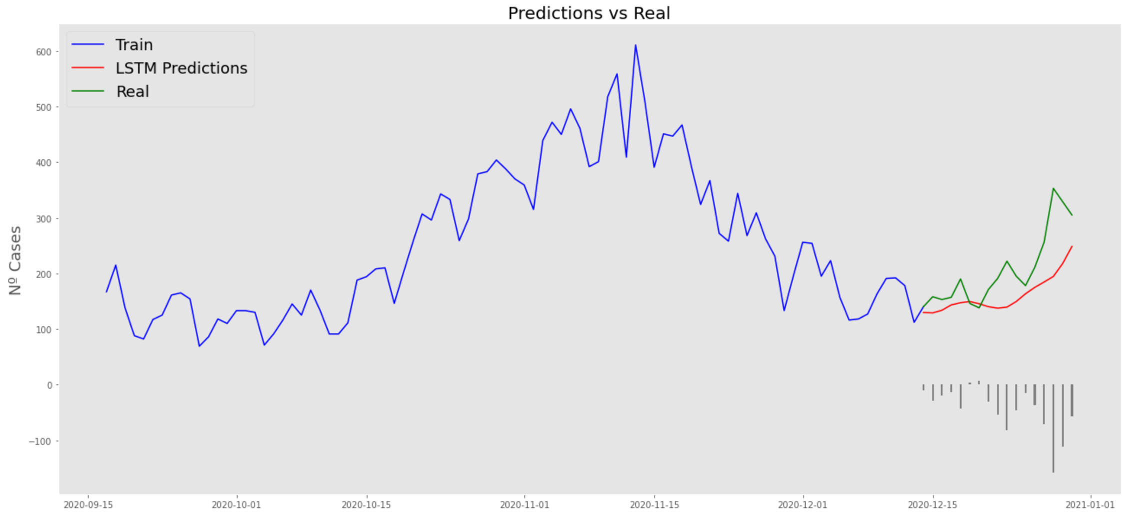
[29]:
```
valid.insert(1, "Predictions", predictions, True)
valid.insert(1, "Difference", valid["Predictions"] - valid["num_casos.x"], True)
```

[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          140  -10.469345   129.530655
2020-12-15          158  -29.090790   128.909210
2020-12-16          153  -19.268707   133.731293
2020-12-17          157  -13.697067   143.302933
2020-12-18          190  -42.740005   147.259995
2020-12-19          146    3.412613   149.412613
2020-12-20          138    7.435318   145.435318
2020-12-21          171  -30.916321   140.083679
2020-12-22          191  -53.577057   137.422943
2020-12-23          222  -82.677368   139.322632
2020-12-24          195  -45.606155   149.393845
2020-12-25          178  -14.631592   163.368408
2020-12-26          211  -36.184509   174.815491
2020-12-27          256  -71.512619   184.487381
2020-12-28          353 -158.478882   194.521118
2020-12-29          329 -111.332916   217.667084
2020-12-30          305  -56.540482   248.459518
```

[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 = Cad.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([[0.10801964, 0.58536585, 0.35585284],
       [0.08837971, 0.65853659, 0.31438127],
       [0.07692308, 0.68292683, 0.2729097],
       [0.09001637, 0.70731707, 0.25785953],
       [0.10638298, 0.80487805, 0.24280936]])

[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(Cad['num_casos.x'])
np_cases_scaled_v2 = scaler_v2_pred.fit_transform(df_cases)
np_cases_scaled_v2[0:5]
```

[33]: array([[0.10801964],
       [0.08837971],
       [0.07692308],
       [0.09001637],
       [0.10638298]])

[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([[0.10801964, 0.58536585, 0.35585284],
       [0.08837971, 0.65853659, 0.31438127]])

[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([[[0.10801964, 0.58536585, 0.35585284],
        [0.08837971, 0.65853659, 0.31438127],
        [0.07692308, 0.68292683, 0.2729097 ],
        [0.09001637, 0.70731707, 0.25785953],
        [0.10638298, 0.80487805, 0.24280936],
        [0.101473  , 0.70731707, 0.12140468],
        [0.06382979, 0.6097561 , 0.        ],
        [0.08510638, 0.70731707, 0.15551839],
        [0.07692308, 0.73170732, 0.31103679],
        [0.06382979, 0.75609756, 0.32140468],
        [0.05728314, 0.75609756, 0.33177258],
        [0.04909984, 0.82926829, 0.25016722],
        [0.04909984, 0.68292683, 0.16856187],
        [0.05400982, 0.6097561 , 0.21270903]],

       [[0.08837971, 0.65853659, 0.31438127],
        [0.07692308, 0.68292683, 0.2729097 ],
        [0.09001637, 0.70731707, 0.25785953],
        [0.10638298, 0.80487805, 0.24280936],
        [0.101473  , 0.70731707, 0.12140468],
        [0.06382979, 0.6097561 , 0.        ],
        [0.08510638, 0.70731707, 0.15551839],
        [0.07692308, 0.73170732, 0.31103679],
        [0.06382979, 0.75609756, 0.32140468],
        [0.05728314, 0.75609756, 0.33177258],
        [0.04909984, 0.82926829, 0.25016722],
        [0.04909984, 0.68292683, 0.16856187],
        [0.05400982, 0.6097561 , 0.21270903],
        [0.05728314, 0.75609756, 0.25685619]]])
```

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

[39]:
```
array([0.05728314, 0.03273322])
```

[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)
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([[[0.3207856 , 0.26829268, 0.39487179],
        [0.41898527, 0.24390244, 0.59375697],
        [0.41571195, 0.26829268, 0.79264214],
        [0.31914894, 0.26829268, 0.67274247],
        [0.36497545, 0.36585366, 0.55284281],
        [0.25695581, 0.29268293, 0.43294314],
        [0.1898527 , 0.29268293, 0.31304348],
        [0.19312602, 0.53658537, 0.47090301],
        [0.20785597, 0.56097561, 0.62876254],
        [0.26677578, 0.24390244, 0.78662207],
        [0.31260229, 0.29268293, 0.70668896],
        [0.31423895, 0.31707317, 0.62675585],
        [0.2913257 , 0.24390244, 0.54682274],
        [0.18330606, 0.19512195, 0.46688963]],

       [[0.41898527, 0.24390244, 0.59375697],
        [0.41571195, 0.26829268, 0.79264214],
        [0.31914894, 0.26829268, 0.67274247],
        [0.36497545, 0.36585366, 0.55284281],
        [0.25695581, 0.29268293, 0.43294314],
        [0.1898527 , 0.29268293, 0.31304348],
        [0.19312602, 0.53658537, 0.47090301],
        [0.20785597, 0.56097561, 0.62876254],
        [0.26677578, 0.24390244, 0.78662207],
        [0.31260229, 0.29268293, 0.70668896],
        [0.31423895, 0.31707317, 0.62675585],
```

```
         [0.2913257 , 0.24390244, 0.54682274],
         [0.18330606, 0.19512195, 0.46688963],
         [0.22913257, 0.24390244, 0.60691193]]])
```

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

[42]:
```
array([0.22913257, 0.25859247])
```

[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 28ms/step - loss: 0.0597 -
val_loss: 0.0101
```

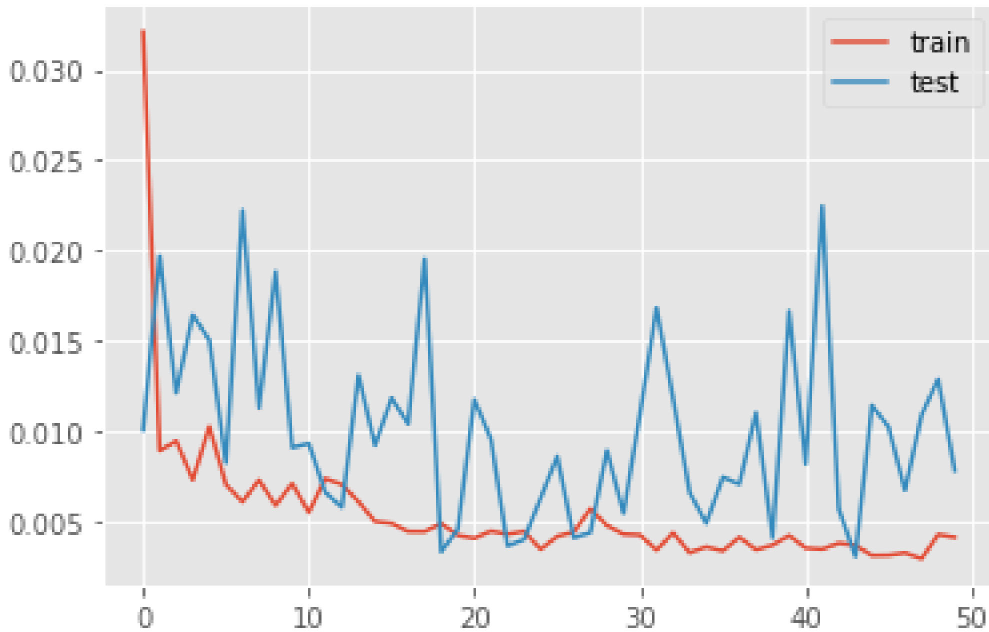
```
Epoch 2/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0072 -
val_loss: 0.0197
Epoch 3/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0087 -
val_loss: 0.0121
Epoch 4/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0069 -
val_loss: 0.0164
Epoch 5/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0082 -
val_loss: 0.0150
Epoch 6/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0064 -
val_loss: 0.0082
Epoch 7/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0056 -
val_loss: 0.0222
Epoch 8/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0063 -
val_loss: 0.0113
Epoch 9/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0045 -
val_loss: 0.0188
Epoch 10/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0067 -
val_loss: 0.0091
Epoch 11/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0047 -
val_loss: 0.0093
Epoch 12/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0062 -
val_loss: 0.0066
Epoch 13/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0062 -
val_loss: 0.0058
Epoch 14/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0074 -
val_loss: 0.0131
Epoch 15/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0040 -
val_loss: 0.0092: 0.00
Epoch 16/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0045 -
val_loss: 0.0118
Epoch 17/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0051 -
val_loss: 0.0104
```

```
Epoch 18/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0036 -
val_loss: 0.0195
Epoch 19/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0055 -
val_loss: 0.0033
Epoch 20/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0039 -
val_loss: 0.0046
Epoch 21/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0038 -
val_loss: 0.0117
Epoch 22/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0047 -
val_loss: 0.0095
Epoch 23/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0046 -
val_loss: 0.0036
Epoch 24/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0044 -
val_loss: 0.0040
Epoch 25/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0039 -
val_loss: 0.0063
Epoch 26/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0044 -
val_loss: 0.0086
Epoch 27/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0046 -
val_loss: 0.0041
Epoch 28/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0073 -
val_loss: 0.0044
Epoch 29/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0053 -
val_loss: 0.0089
Epoch 30/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0034 -
val_loss: 0.0054
Epoch 31/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0049 -
val_loss: 0.0109
Epoch 32/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0035 -
val_loss: 0.0168
Epoch 33/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0045 -
val_loss: 0.0118
```

```
Epoch 34/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0035 -
val_loss: 0.0066
Epoch 35/50
130/130 [==============================] - 3s 17ms/step - loss: 0.0026 -
val_loss: 0.0049
Epoch 36/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0037 -
val_loss: 0.0074
Epoch 37/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0039 -
val_loss: 0.0070
Epoch 38/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0043 -
val_loss: 0.0110
Epoch 39/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0046 -
val_loss: 0.0041
Epoch 40/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0034 -
val_loss: 0.0166
Epoch 41/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0029 -
val_loss: 0.0082
Epoch 42/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0045 -
val_loss: 0.0225
Epoch 43/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0046 -
val_loss: 0.0057
Epoch 44/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0045 -
val_loss: 0.0030
Epoch 45/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0034 -
val_loss: 0.0114
Epoch 46/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0027 -
val_loss: 0.0102
Epoch 47/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0031 -
val_loss: 0.0067
Epoch 48/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0027 -
val_loss: 0.0109
Epoch 49/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0035 -
val_loss: 0.0129
```

```
Epoch 50/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0053 -
val_loss: 0.0078
```

[46]:
```
# 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([[0.20759802],
       [0.19907223],
       [0.1973475 ],
       [0.2037837 ],
       [0.21117206],
       [0.21847494],
       [0.21711648],
       [0.21534866],
       [0.21797682],
       [0.23682909],
       [0.2690856 ],
       [0.30881923],
```

```
       [0.32450265],
       [0.33344644],
       [0.35688227],
       [0.39469033],
       [0.43910956]], 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: 43.2
RMSE: 54.0
```

[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
```

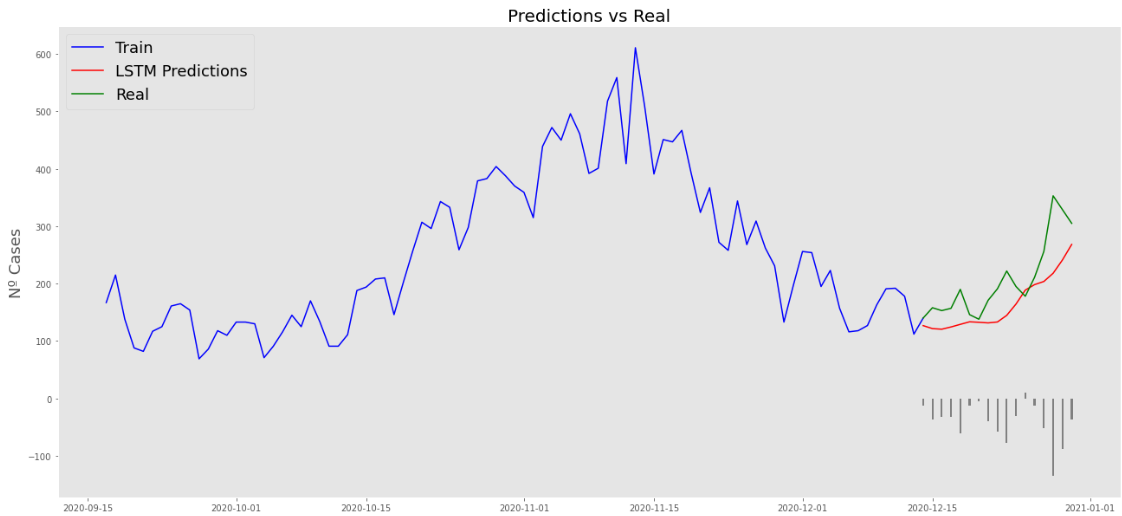
[52]:
```
            num_casos.x  Difference  Predictions  \
fecha
2020-12-14          140  -13.157616   126.842384
2020-12-15          158  -36.366867   121.633133
2020-12-16          153  -32.420670   120.579330
2020-12-17          157  -32.488159   124.511841
2020-12-18          190  -60.973877   129.026123
2020-12-19          146  -12.511810   133.488190
```

```
2020-12-20          138   -5.341827   132.658173
2020-12-21          171  -39.421967   131.578033
2020-12-22          191  -57.816162   133.183838
2020-12-23          222  -77.297424   144.702576
2020-12-24          195  -30.588699   164.411301
2020-12-25          178   10.688553   188.688553
2020-12-26          211  -12.728882   198.271118
2020-12-27          256  -52.264221   203.735779
2020-12-28          353 -134.944931   218.055069
2020-12-29          329  -87.844208   241.155792
2020-12-30          305  -36.704071   268.295929

            residential_percent_change_from_baseline      total
fecha
2020-12-14                                       7.0  14.493333
2020-12-15                                       5.0  16.586667
2020-12-16                                       8.0  18.680000
2020-12-17                                       6.0  17.247500
2020-12-18                                       6.0  15.815000
2020-12-19                                       7.0  14.382500
2020-12-20                                       3.0  12.950000
2020-12-21                                       5.0  14.680000
2020-12-22                                       5.0  16.410000
2020-12-23                                       4.0  18.140000
2020-12-24                                       8.0  16.890000
2020-12-25                                      19.0  15.640000
2020-12-26                                       7.0  14.390000
2020-12-27                                       4.0  13.140000
2020-12-28                                      10.0  14.480000
2020-12-29                                       8.0  15.820000
2020-12-30                                       7.0  17.160000
```

[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 = Cad.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([[0.10801964, 0.58536585, 0.13076923, 0.32      , 0.0701107 ,
        0.26804124, 0.32075472, 0.35585284],
       [0.08837971, 0.65853659, 0.08461538, 0.22857143, 0.04797048,
        0.19587629, 0.23584906, 0.31438127],
       [0.07692308, 0.68292683, 0.08461538, 0.24      , 0.04797048,
        0.15463918, 0.22641509, 0.2729097 ],
       [0.09001637, 0.70731707, 0.08461538, 0.24      , 0.04797048,
        0.16494845, 0.19811321, 0.25785953],
       [0.10638298, 0.80487805, 0.06923077, 0.22857143, 0.03690037,
        0.11340206, 0.19811321, 0.24280936]])
```

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

[55]:
```
array([[0.10801964],
       [0.08837971],
       [0.07692308],
       [0.09001637],
       [0.10638298]])
```

[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([[0.10801964, 0.58536585, 0.13076923, 0.32      , 0.0701107 ,
        0.26804124, 0.32075472, 0.35585284],
       [0.08837971, 0.65853659, 0.08461538, 0.22857143, 0.04797048,
        0.19587629, 0.23584906, 0.31438127]])
```

[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):
    #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([[[0.10801964, 0.58536585, 0.13076923, 0.32      , 0.0701107 ,
         0.26804124, 0.32075472, 0.35585284],
        [0.08837971, 0.65853659, 0.08461538, 0.22857143, 0.04797048,
         0.19587629, 0.23584906, 0.31438127],
        [0.07692308, 0.68292683, 0.08461538, 0.24      , 0.04797048,
         0.15463918, 0.22641509, 0.2729097 ],
        [0.09001637, 0.70731707, 0.08461538, 0.24      , 0.04797048,
         0.16494845, 0.19811321, 0.25785953],
        [0.10638298, 0.80487805, 0.06923077, 0.22857143, 0.03690037,
         0.11340206, 0.19811321, 0.24280936],
        [0.101473  , 0.70731707, 0.03846154, 0.17142857, 0.01107011,
         0.09278351, 0.20754717, 0.12140468],
        [0.06382979, 0.6097561 , 0.01538462, 0.07428571, 0.        ,
         0.06185567, 0.18867925, 0.        ],
        [0.08510638, 0.70731707, 0.09230769, 0.23428571, 0.04428044,
         0.13402062, 0.17924528, 0.15551839],
        [0.07692308, 0.73170732, 0.06923077, 0.18857143, 0.03690037,
         0.09278351, 0.16037736, 0.31103679],
        [0.06382979, 0.75609756, 0.07692308, 0.2       , 0.04059041,
         0.10309278, 0.16037736, 0.32140468],
        [0.05728314, 0.75609756, 0.06923077, 0.19428571, 0.04059041,
         0.10309278, 0.16037736, 0.33177258],
        [0.04909984, 0.82926829, 0.06153846, 0.20571429, 0.04059041,
         0.08247423, 0.16981132, 0.25016722],
        [0.04909984, 0.68292683, 0.03846154, 0.16571429, 0.01845018,
         0.07216495, 0.20754717, 0.16856187],
        [0.05400982, 0.6097561 , 0.01538462, 0.05714286, 0.00369004,
         0.03092784, 0.18867925, 0.21270903]],

       [[0.08837971, 0.65853659, 0.08461538, 0.22857143, 0.04797048,
         0.19587629, 0.23584906, 0.31438127],
```

```
[0.07692308, 0.68292683, 0.08461538, 0.24      , 0.04797048,
 0.15463918, 0.22641509, 0.2729097 ],
[0.09001637, 0.70731707, 0.08461538, 0.24      , 0.04797048,
 0.16494845, 0.19811321, 0.25785953],
[0.10638298, 0.80487805, 0.06923077, 0.22857143, 0.03690037,
 0.11340206, 0.19811321, 0.24280936],
[0.101473  , 0.70731707, 0.03846154, 0.17142857, 0.01107011,
 0.09278351, 0.20754717, 0.12140468],
[0.06382979, 0.6097561 , 0.01538462, 0.07428571, 0.        ,
 0.06185567, 0.18867925, 0.        ],
[0.08510638, 0.70731707, 0.09230769, 0.23428571, 0.04428044,
 0.13402062, 0.17924528, 0.15551839],
[0.07692308, 0.73170732, 0.06923077, 0.18857143, 0.03690037,
 0.09278351, 0.16037736, 0.31103679],
[0.06382979, 0.75609756, 0.07692308, 0.2       , 0.04059041,
 0.10309278, 0.16037736, 0.32140468],
[0.05728314, 0.75609756, 0.06923077, 0.19428571, 0.04059041,
 0.10309278, 0.16037736, 0.33177258],
[0.04909984, 0.82926829, 0.06153846, 0.20571429, 0.04059041,
 0.08247423, 0.16981132, 0.25016722],
[0.04909984, 0.68292683, 0.03846154, 0.16571429, 0.01845018,
 0.07216495, 0.20754717, 0.16856187],
[0.05400982, 0.6097561 , 0.01538462, 0.05714286, 0.00369004,
 0.03092784, 0.18867925, 0.21270903],
[0.05728314, 0.75609756, 0.06923077, 0.19428571, 0.03690037,
 0.08247423, 0.13207547, 0.25685619]]])
```

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

[60]:
```
array([0.05728314, 0.03273322])
```

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

```
(31, 8)
```

[62]:

```
array([[[0.3207856 , 0.26829268, 0.53076923, 0.50857143, 0.24723247,
         0.60824742, 0.67924528, 0.39487179],
        [0.41898527, 0.24390244, 0.54615385, 0.52      , 0.28782288,
         0.6185567 , 0.68867925, 0.59375697],
        [0.41571195, 0.26829268, 0.54615385, 0.53714286, 0.28413284,
         0.6185567 , 0.69811321, 0.79264214],
        [0.31914894, 0.26829268, 0.54615385, 0.53714286, 0.26937269,
         0.62886598, 0.67924528, 0.67274247],
        [0.36497545, 0.36585366, 0.44615385, 0.49142857, 0.15867159,
         0.5257732 , 0.6509434 , 0.55284281],
        [0.25695581, 0.29268293, 0.48461538, 0.54285714, 0.23616236,
         0.54639175, 0.72641509, 0.43294314],
        [0.1898527 , 0.29268293, 0.44615385, 0.58857143, 0.22140221,
         0.42268041, 0.68867925, 0.31304348],
        [0.19312602, 0.53658537, 0.48461538, 0.45714286, 0.18819188,
         0.32989691, 0.22641509, 0.47090301],
        [0.20785597, 0.56097561, 0.42307692, 0.28571429, 0.19926199,
         0.39175258, 0.20754717, 0.62876254],
        [0.26677578, 0.24390244, 0.55384615, 0.6       , 0.26199262,
         0.62886598, 0.69811321, 0.78662207],
        [0.31260229, 0.29268293, 0.50769231, 0.52571429, 0.19188192,
         0.56701031, 0.66981132, 0.70668896],
        [0.31423895, 0.31707317, 0.47692308, 0.52      , 0.22509225,
         0.55670103, 0.69811321, 0.62675585],
        [0.2913257 , 0.24390244, 0.51538462, 0.52      , 0.25830258,
         0.6185567 , 0.75471698, 0.54682274],
        [0.18330606, 0.19512195, 0.54615385, 0.85142857, 0.24723247,
         0.51546392, 0.77358491, 0.46688963]],

       [[0.41898527, 0.24390244, 0.54615385, 0.52      , 0.28782288,
         0.6185567 , 0.68867925, 0.59375697],
        [0.41571195, 0.26829268, 0.54615385, 0.53714286, 0.28413284,
         0.6185567 , 0.69811321, 0.79264214],
        [0.31914894, 0.26829268, 0.54615385, 0.53714286, 0.26937269,
         0.62886598, 0.67924528, 0.67274247],
        [0.36497545, 0.36585366, 0.44615385, 0.49142857, 0.15867159,
```

```
        0.5257732 , 0.6509434 , 0.55284281],
       [0.25695581, 0.29268293, 0.48461538, 0.54285714, 0.23616236,
        0.54639175, 0.72641509, 0.43294314],
       [0.1898527 , 0.29268293, 0.44615385, 0.58857143, 0.22140221,
        0.42268041, 0.68867925, 0.31304348],
       [0.19312602, 0.53658537, 0.48461538, 0.45714286, 0.18819188,
        0.32989691, 0.22641509, 0.47090301],
       [0.20785597, 0.56097561, 0.42307692, 0.28571429, 0.19926199,
        0.39175258, 0.20754717, 0.62876254],
       [0.26677578, 0.24390244, 0.55384615, 0.6       , 0.26199262,
        0.62886598, 0.69811321, 0.78662207],
       [0.31260229, 0.29268293, 0.50769231, 0.52571429, 0.19188192,
        0.56701031, 0.66981132, 0.70668896],
       [0.31423895, 0.31707317, 0.47692308, 0.52      , 0.22509225,
        0.55670103, 0.69811321, 0.62675585],
       [0.2913257 , 0.24390244, 0.51538462, 0.52      , 0.25830258,
        0.6185567 , 0.75471698, 0.54682274],
       [0.18330606, 0.19512195, 0.54615385, 0.85142857, 0.24723247,
        0.51546392, 0.77358491, 0.46688963],
       [0.22913257, 0.24390244, 0.57692308, 0.50285714, 0.21402214,
        0.62886598, 0.68867925, 0.60691193]]])
```

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

[63]:
```
array([0.22913257, 0.25859247])
```

[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))
```

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

```
112 14 8
```

[66]:

```python
# 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]
                         )
```

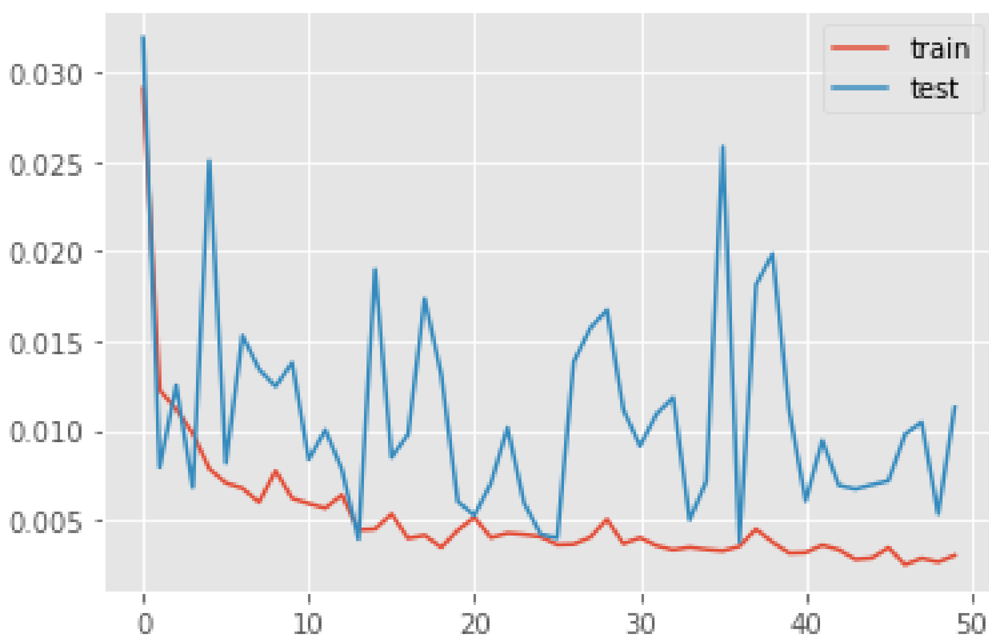
```
Epoch 1/50
130/130 [==============================] - 10s 27ms/step - loss: 0.0482 -
val_loss: 0.0319
Epoch 2/50
130/130 [==============================] - 3s 19ms/step - loss: 0.0120 -
val_loss: 0.0080
Epoch 3/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0108 -
val_loss: 0.0125
Epoch 4/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0096 -
val_loss: 0.0068
Epoch 5/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0066 -
val_loss: 0.0251
Epoch 6/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0082 -
val_loss: 0.0082
Epoch 7/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0063 -
val_loss: 0.0153
Epoch 8/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0063 -
val_loss: 0.0134
Epoch 9/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0063 -
val_loss: 0.0125
Epoch 10/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0062 -
val_loss: 0.0138
Epoch 11/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0049 -
```

```
val_loss: 0.0084
Epoch 12/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0063 -
val_loss: 0.0100
Epoch 13/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0059 -
val_loss: 0.0078- loss: 0.
Epoch 14/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0042 -
val_loss: 0.0039
Epoch 15/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0047 -
val_loss: 0.0190
Epoch 16/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0061 -
val_loss: 0.0085
Epoch 17/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0037 -
val_loss: 0.0098
Epoch 18/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0036 -
val_loss: 0.0174
Epoch 19/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0036 -
val_loss: 0.0131
Epoch 20/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0066 -
val_loss: 0.0060
Epoch 21/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0064 -
val_loss: 0.0053
Epoch 22/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0036 -
val_loss: 0.0071
Epoch 23/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0050 -
val_loss: 0.0102
Epoch 24/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0040 -
val_loss: 0.0060
Epoch 25/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0043 -
val_loss: 0.0042
Epoch 26/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0040 -
val_loss: 0.0040
Epoch 27/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0038 -
```

```
val_loss: 0.0138
Epoch 28/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0042 -
val_loss: 0.0157
Epoch 29/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0055 -
val_loss: 0.0167
Epoch 30/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0034 -
val_loss: 0.0112
Epoch 31/50
130/130 [==============================] - 4s 31ms/step - loss: 0.0030 -
val_loss: 0.0091
Epoch 32/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0032 -
val_loss: 0.0109
Epoch 33/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0031 -
val_loss: 0.0118
Epoch 34/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0033 -
val_loss: 0.0050
Epoch 35/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0029 -
val_loss: 0.0072
Epoch 36/50
130/130 [==============================] - 4s 29ms/step - loss: 0.0030 -
val_loss: 0.0259
Epoch 37/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0042 -
val_loss: 0.0038
Epoch 38/50
130/130 [==============================] - 3s 25ms/step - loss: 0.0040 -
val_loss: 0.0181
Epoch 39/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0050 -
val_loss: 0.0199
Epoch 40/50
130/130 [==============================] - 4s 31ms/step - loss: 0.0038 -
val_loss: 0.0112
Epoch 41/50
130/130 [==============================] - 4s 27ms/step - loss: 0.0027 -
val_loss: 0.0061
Epoch 42/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0033 -
val_loss: 0.0094
Epoch 43/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0036 -
```

```
val_loss: 0.0069
Epoch 44/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0026 -
val_loss: 0.0067
Epoch 45/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0028 -
val_loss: 0.0070
Epoch 46/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0038 -
val_loss: 0.0072
Epoch 47/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0022 -
val_loss: 0.0098
Epoch 48/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0022 -
val_loss: 0.0105
Epoch 49/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0022 -
val_loss: 0.0053
Epoch 50/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0023 -
val_loss: 0.0113
```

[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([[0.18996866],
       [0.1951589 ],
       [0.21025515],
       [0.23140553],
       [0.25299728],
       [0.24498552],
       [0.21799392],
       [0.19986223],
       [0.20329835],
       [0.22916168],
       [0.2721959 ],
       [0.3055893 ],
       [0.30865157],
       [0.2799372 ],
       [0.2922846 ],
       [0.34919062],
       [0.4340176 ]], 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: 48.2
RMSE: 65.0
```

[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]: # 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
```

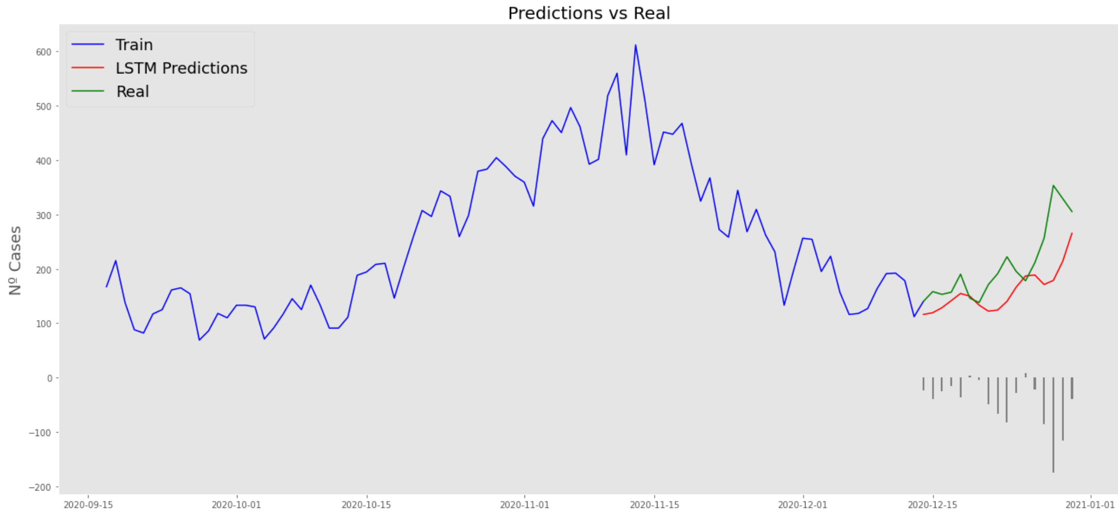
```
[73]:             num_casos.x  Difference  Predictions  \
fecha
2020-12-14          140  -23.929146   116.070854
2020-12-15          158  -38.757912   119.242088
2020-12-16          153  -24.534103   128.465897
2020-12-17          157  -15.611221   141.388779
2020-12-18          190  -35.418655   154.581345
2020-12-19          146    3.686157   149.686157
2020-12-20          138   -4.805725   133.194275
2020-12-21          171  -48.884178   122.115822
2020-12-22          191  -66.784714   124.215286
2020-12-23          222  -81.982208   140.017792
2020-12-24          195  -28.688309   166.311691
2020-12-25          178    8.715057   186.715057
2020-12-26          211  -22.413895   188.586105
2020-12-27          256  -84.958359   171.041641
2020-12-28          353 -174.414108   178.585892
2020-12-29          329 -115.644531   213.355469
2020-12-30          305  -39.815247   265.184753

            residential_percent_change_from_baseline  \
fecha
2020-12-14                                       7.0
2020-12-15                                       5.0
2020-12-16                                       8.0
2020-12-17                                       6.0
2020-12-18                                       6.0
2020-12-19                                       7.0
2020-12-20                                       3.0
2020-12-21                                       5.0
2020-12-22                                       5.0
2020-12-23                                       4.0
2020-12-24                                       8.0
2020-12-25                                      19.0
2020-12-26                                       7.0
2020-12-27                                       4.0
```

```
2020-12-28                                      10.0
2020-12-29                                       8.0
2020-12-30                                       7.0

            retail_and_recreation_percent_change_from_baseline  \
fecha
2020-12-14                                              -21.0
2020-12-15                                              -15.0
2020-12-16                                              -27.0
2020-12-17                                              -17.0
2020-12-18                                              -19.0
2020-12-19                                              -29.0
2020-12-20                                              -13.0
2020-12-21                                               -5.0
2020-12-22                                               -6.0
2020-12-23                                                0.0
2020-12-24                                              -18.0
2020-12-25                                              -70.0
2020-12-26                                              -25.0
2020-12-27                                              -11.0
2020-12-28                                              -10.0
2020-12-29                                               -2.0
2020-12-30                                                1.0

            grocery_and_pharmacy_percent_change_from_baseline  \
fecha
2020-12-14                                               -4.0
2020-12-15                                                4.0
2020-12-16                                               -4.0
2020-12-17                                                8.0
2020-12-18                                                4.0
2020-12-19                                               -2.0
2020-12-20                                               83.0
2020-12-21                                               13.0
2020-12-22                                               17.0
2020-12-23                                               36.0
2020-12-24                                               13.0
2020-12-25                                              -77.0
2020-12-26                                               -5.0
2020-12-27                                               83.0
2020-12-28                                                6.0
2020-12-29                                               18.0
2020-12-30                                               35.0

            parks_percent_change_from_baseline  \
fecha
2020-12-14                               -34.0
```

```
2020-12-15                                 -10.0
2020-12-16                                 -40.0
2020-12-17                                 -14.0
2020-12-18                                  -6.0
2020-12-19                                 -34.0
2020-12-20                                 -16.0
2020-12-21                                  -5.0
2020-12-22                                  -3.0
2020-12-23                                   1.0
2020-12-24                                 -13.0
2020-12-25                                 -23.0
2020-12-26                                  -8.0
2020-12-27                                 -10.0
2020-12-28                                 -31.0
2020-12-29                                  -1.0
2020-12-30                                   7.0

            transit_stations_percent_change_from_baseline  \
fecha
2020-12-14                                          -32.0
2020-12-15                                          -28.0
2020-12-16                                          -37.0
2020-12-17                                          -28.0
2020-12-18                                          -26.0
2020-12-19                                          -34.0
2020-12-20                                          -33.0
2020-12-21                                          -24.0
2020-12-22                                          -26.0
2020-12-23                                          -18.0
2020-12-24                                          -42.0
2020-12-25                                          -68.0
2020-12-26                                          -31.0
2020-12-27                                          -27.0
2020-12-28                                          -36.0
2020-12-29                                          -32.0
2020-12-30                                          -27.0

            workplaces_percent_change_from_baseline      total
fecha
2020-12-14                                    -14.0  14.493333
2020-12-15                                    -13.0  16.586667
2020-12-16                                    -15.0  18.680000
2020-12-17                                    -14.0  17.247500
2020-12-18                                    -12.0  15.815000
2020-12-19                                    -10.0  14.382500
2020-12-20                                     -3.0  12.950000
2020-12-21                                    -17.0  14.680000
```

2020-12-22	-19.0	16.410000
2020-12-23	-22.0	18.140000
2020-12-24	-46.0	16.890000
2020-12-25	-75.0	15.640000
2020-12-26	-14.0	14.390000
2020-12-27	-4.0	13.140000
2020-12-28	-37.0	14.480000
2020-12-29	-36.0	15.820000
2020-12-30	-34.0	17.160000

[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()
```


Predictions vs Real
Train
LSTM Predictions
Real
Nº Cases
600
500
400
300
200
100
0
−100
−200
2020-09-15
2020-10-01
2020-10-15
2020-11-01
2020-11-15
2020-12-01
2020-12-15
2021-01-01

[ ]: