LSTM_arodriguezsans-Univariate_Multivariate_Sev

May 18, 2021

1 Sevilla

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]:
```
Sev=df_total.loc[df_total['sub_region_2'] == 'Sevilla']
```

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

[6]:
```
# We select columns of interest (mobility ones)
Sev=Sev[['num_casos.x'] + list(Sev.loc[:
↪,'retail_and_recreation_percent_change_from_baseline':'total'])]
```

1.4 Plots

[7]:
```
# 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(Sev.values[:, group])
    plt.title(Sev.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

[8]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data = Sev.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]
```

[8]:
```
array([[0.07290867],
       [0.08211819],
       [0.06676899],
       [0.08749041],
       [0.11281658]])
```

[9]:
```
npdataset[0:5]
```

[9]:
```
array([[ 95],
       [107],
       [ 87],
       [114],
       [147]], dtype=int64)
```

[10]:
```
len(scaled_data)
```

[10]: 290

[11]:
```
training_data_length
```

[11]: 273

[12]:
```
# 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
```

[13]:
```
# 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])
```

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

[14]:
```
[array([0.07290867, 0.08211819, 0.06676899, 0.08749041, 0.11281658,
        0.07904835, 0.10590944, 0.10590944, 0.06830391, 0.09439754,
        0.06830391, 0.05832694, 0.02839601, 0.02916347]),
 array([0.08211819, 0.06676899, 0.08749041, 0.11281658, 0.07904835,
        0.10590944, 0.10590944, 0.06830391, 0.09439754, 0.06830391,
        0.05832694, 0.02839601, 0.02916347, 0.04451266])]
```

[15]:
```
# list
y_train[0:2]
```

[15]:
```
[0.044512663085188024, 0.03837298541826554]
```

[16]:
```
# 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])
```

```
[[0.07290867 0.08211819 0.06676899 0.08749041 0.11281658 0.07904835
  0.10590944 0.10590944 0.06830391 0.09439754 0.06830391 0.05832694
  0.02839601 0.02916347]
 [0.08211819 0.06676899 0.08749041 0.11281658 0.07904835 0.10590944
  0.10590944 0.06830391 0.09439754 0.06830391 0.05832694 0.02839601
  0.02916347 0.04451266]]
-----------------------------------------------------------
[0.04451266 0.03837299]
```

[17]:
```python
# 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,)
```

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

[18]:
```
array([[[0.07290867],
        [0.08211819],
        [0.06676899],
        [0.08749041],
        [0.11281658],
        [0.07904835],
        [0.10590944],
        [0.10590944],
        [0.06830391],
        [0.09439754],
        [0.06830391],
        [0.05832694],
        [0.02839601],
        [0.02916347]],

       [[0.08211819],
        [0.06676899],
        [0.08749041],
        [0.11281658],
        [0.07904835],
        [0.10590944],
        [0.10590944],
        [0.06830391],
        [0.09439754],
        [0.06830391],
        [0.05832694],
        [0.02839601],
        [0.02916347],
```

```
        [0.04451266]]])
```

[19]:
```
y_train[0:2]
```

[19]:
```
array([0.04451266, 0.03837299])
```

[20]:
```
# 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])
```

```
[[[0.13814275]
  [0.16116654]
  [0.13737529]
  [0.12586339]
  [0.11588642]
  [0.10744436]
  [0.05295472]
  [0.07751343]
  [0.06062932]
  [0.12586339]
  [0.14121259]
  [0.15272448]
  [0.12970069]
  [0.06369916]]

 [[0.16116654]
```

```
  [0.13737529]
  [0.12586339]
  [0.11588642]
  [0.10744436]
  [0.05295472]
  [0.07751343]
  [0.06062932]
  [0.12586339]
  [0.14121259]
  [0.15272448]
  [0.12970069]
  [0.06369916]
  [0.1143515 ]]]
-----------------------------------------------------------
[0.1143515  0.13814275]
```

[21]:
```python
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].

[22]:
```
# 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')
```

[23]:
```
# 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)
```

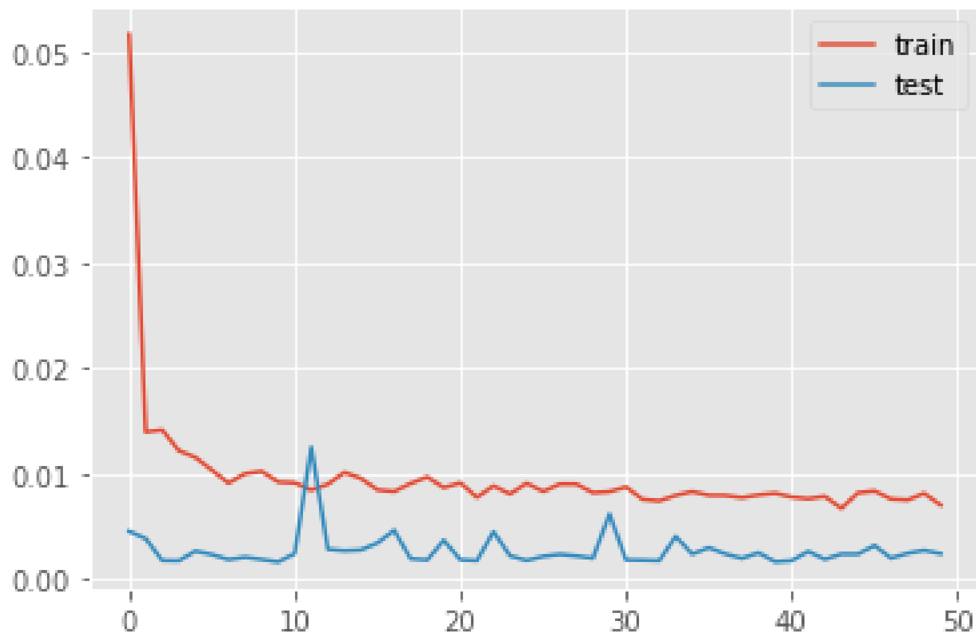
```
Epoch 1/50
130/130 - 11s - loss: 0.0517 - val_loss: 0.0045
Epoch 2/50
```

```
130/130 - 2s - loss: 0.0139 - val_loss: 0.0038
Epoch 3/50
130/130 - 2s - loss: 0.0141 - val_loss: 0.0017
Epoch 4/50
130/130 - 2s - loss: 0.0122 - val_loss: 0.0017
Epoch 5/50
130/130 - 2s - loss: 0.0115 - val_loss: 0.0026
Epoch 6/50
130/130 - 2s - loss: 0.0103 - val_loss: 0.0023
Epoch 7/50
130/130 - 2s - loss: 0.0091 - val_loss: 0.0018
Epoch 8/50
130/130 - 2s - loss: 0.0100 - val_loss: 0.0020
Epoch 9/50
130/130 - 2s - loss: 0.0102 - val_loss: 0.0018
Epoch 10/50
130/130 - 2s - loss: 0.0092 - val_loss: 0.0016
Epoch 11/50
130/130 - 2s - loss: 0.0091 - val_loss: 0.0024
Epoch 12/50
130/130 - 2s - loss: 0.0084 - val_loss: 0.0125
Epoch 13/50
130/130 - 2s - loss: 0.0090 - val_loss: 0.0028
Epoch 14/50
130/130 - 2s - loss: 0.0101 - val_loss: 0.0026
Epoch 15/50
130/130 - 3s - loss: 0.0095 - val_loss: 0.0027
Epoch 16/50
130/130 - 2s - loss: 0.0084 - val_loss: 0.0034
Epoch 17/50
130/130 - 1s - loss: 0.0083 - val_loss: 0.0046
Epoch 18/50
130/130 - 2s - loss: 0.0090 - val_loss: 0.0019
Epoch 19/50
130/130 - 2s - loss: 0.0097 - val_loss: 0.0018
Epoch 20/50
130/130 - 2s - loss: 0.0086 - val_loss: 0.0036
Epoch 21/50
130/130 - 2s - loss: 0.0091 - val_loss: 0.0018
Epoch 22/50
130/130 - 2s - loss: 0.0077 - val_loss: 0.0017
Epoch 23/50
130/130 - 2s - loss: 0.0088 - val_loss: 0.0045
Epoch 24/50
130/130 - 2s - loss: 0.0080 - val_loss: 0.0021
Epoch 25/50
130/130 - 2s - loss: 0.0091 - val_loss: 0.0017
Epoch 26/50
```

```
130/130 - 2s - loss: 0.0083 - val_loss: 0.0021
Epoch 27/50
130/130 - 2s - loss: 0.0090 - val_loss: 0.0023
Epoch 28/50
130/130 - 2s - loss: 0.0090 - val_loss: 0.0021
Epoch 29/50
130/130 - 2s - loss: 0.0082 - val_loss: 0.0019
Epoch 30/50
130/130 - 2s - loss: 0.0083 - val_loss: 0.0061
Epoch 31/50
130/130 - 2s - loss: 0.0087 - val_loss: 0.0018
Epoch 32/50
130/130 - 2s - loss: 0.0075 - val_loss: 0.0017
Epoch 33/50
130/130 - 2s - loss: 0.0074 - val_loss: 0.0017
Epoch 34/50
130/130 - 2s - loss: 0.0079 - val_loss: 0.0040
Epoch 35/50
130/130 - 2s - loss: 0.0083 - val_loss: 0.0023
Epoch 36/50
130/130 - 2s - loss: 0.0079 - val_loss: 0.0029
Epoch 37/50
130/130 - 2s - loss: 0.0079 - val_loss: 0.0023
Epoch 38/50
130/130 - 2s - loss: 0.0077 - val_loss: 0.0019
Epoch 39/50
130/130 - 2s - loss: 0.0079 - val_loss: 0.0024
Epoch 40/50
130/130 - 1s - loss: 0.0081 - val_loss: 0.0016
Epoch 41/50
130/130 - 2s - loss: 0.0078 - val_loss: 0.0017
Epoch 42/50
130/130 - 2s - loss: 0.0076 - val_loss: 0.0026
Epoch 43/50
130/130 - 2s - loss: 0.0078 - val_loss: 0.0018
Epoch 44/50
130/130 - 3s - loss: 0.0067 - val_loss: 0.0023
Epoch 45/50
130/130 - 2s - loss: 0.0081 - val_loss: 0.0023
Epoch 46/50
130/130 - 2s - loss: 0.0083 - val_loss: 0.0031
Epoch 47/50
130/130 - 2s - loss: 0.0076 - val_loss: 0.0019
Epoch 48/50
130/130 - 2s - loss: 0.0075 - val_loss: 0.0024
Epoch 49/50
130/130 - 2s - loss: 0.0081 - val_loss: 0.0027
Epoch 50/50
```

```
130/130 - 2s - loss: 0.0070 - val_loss: 0.0024
```

[24]:
```
# Plot history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```



[25]:
```
# Get the predicted values
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

[26]:
```
predictions
```

[26]:
```
array([[107.76372 ],
       [114.4487  ],
       [127.56039 ],
       [132.94547 ],
       [135.7882  ],
       [135.76784 ],
       [132.94547 ],
       [130.67831 ],
       [117.93382 ],
       [111.67485 ],
       [119.866905],
       [131.26279 ],
```

```
        [132.94547 ],
        [132.94547 ],
        [132.94547 ],
        [136.1526  ],
        [157.37598 ]], dtype=float32)
```

[27]:
```
y_test = y_test.reshape(-1,1)
y_test = scaler.inverse_transform(y_test)
y_test
```

[27]:
```
array([[149.],
       [180.],
       [150.],
       [138.],
       [139.],
       [102.],
       [ 72.],
       [129.],
       [198.],
       [205.],
       [190.],
       [114.],
       [135.],
       [157.],
       [229.],
       [233.],
       [288.]])
```

[28]:
```
# 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: 49.7
RMSE: 63.2
```

[29]:
```
# 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:]
```

[30]:
```
valid.insert(1, "Predictions", predictions, True)
valid.insert(1, "Difference", valid["Predictions"] - valid["num_casos.x"], True)
```

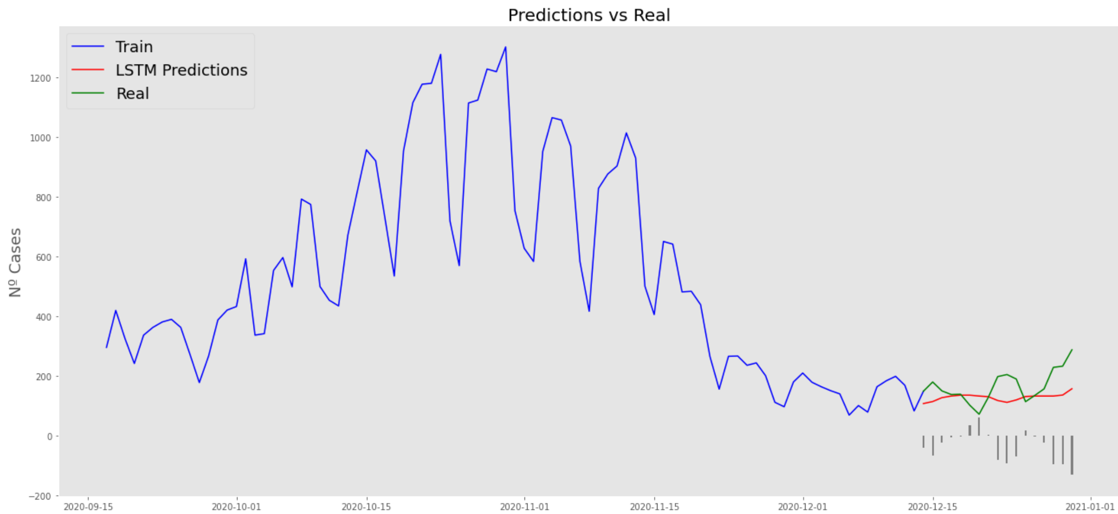
[31]:
```
# Zoom-in to a closer timeframe
valid = valid[valid.index > display_start_date]
train = train[train.index > display_start_date]

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

[31]:

fecha	num_casos.x	Difference	Predictions
2020-12-14	149	-41.236282	107.763718
2020-12-15	180	-65.551300	114.448700
2020-12-16	150	-22.439613	127.560387
2020-12-17	138	-5.054535	132.945465
2020-12-18	139	-3.211807	135.788193
2020-12-19	102	33.767838	135.767838
2020-12-20	72	60.945465	132.945465
2020-12-21	129	1.678314	130.678314
2020-12-22	198	-80.066177	117.933823
2020-12-23	205	-93.325150	111.674850
2020-12-24	190	-70.133095	119.866905
2020-12-25	114	17.262787	131.262787
2020-12-26	135	-2.054535	132.945465
2020-12-27	157	-24.054535	132.945465
2020-12-28	229	-96.054535	132.945465
2020-12-29	233	-96.847397	136.152603
2020-12-30	288	-130.624023	157.375977

[32]:
```
# 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

[33]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data_v2 = Sev.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]
```

[33]:
```
array([[0.07290867, 0.55319149, 0.40776119],
       [0.08211819, 0.63829787, 0.36358209],
       [0.06676899, 0.65957447, 0.31940299],
       [0.08749041, 0.68085106, 0.30597015],
       [0.11281658, 0.76595745, 0.29253731]])
```

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

[34]:
```
array([[0.07290867],
       [0.08211819],
       [0.06676899],
       [0.08749041],
       [0.11281658]])
```

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

```
(273, 3)
```

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

[36]:
```
array([[0.07290867, 0.55319149, 0.40776119],
       [0.08211819, 0.63829787, 0.36358209]])
```

[37]:
```
training_data_length_v2
```

[37]:
```
273
```

[38]:
```
loop_back
```

[38]:
```
14
```

[39]:
```
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([[[0.07290867, 0.55319149, 0.40776119],
        [0.08211819, 0.63829787, 0.36358209],
        [0.06676899, 0.65957447, 0.31940299],
        [0.08749041, 0.68085106, 0.30597015],
        [0.11281658, 0.76595745, 0.29253731],
        [0.07904835, 0.68085106, 0.14626866],
        [0.10590944, 0.57446809, 0.        ],
        [0.10590944, 0.65957447, 0.18179104],
        [0.06830391, 0.70212766, 0.36358209],
        [0.09439754, 0.70212766, 0.36029851],
        [0.06830391, 0.72340426, 0.35701493],
        [0.05832694, 0.80851064, 0.26149254],
        [0.02839601, 0.65957447, 0.16597015],
        [0.02916347, 0.55319149, 0.2158209 ]],

       [[0.08211819, 0.63829787, 0.36358209],
        [0.06676899, 0.65957447, 0.31940299],
        [0.08749041, 0.68085106, 0.30597015],
        [0.11281658, 0.76595745, 0.29253731],
        [0.07904835, 0.68085106, 0.14626866],
        [0.10590944, 0.57446809, 0.        ],
        [0.10590944, 0.65957447, 0.18179104],
        [0.06830391, 0.70212766, 0.36358209],
        [0.09439754, 0.70212766, 0.36029851],
        [0.06830391, 0.72340426, 0.35701493],
        [0.05832694, 0.80851064, 0.26149254],
        [0.02839601, 0.65957447, 0.16597015],
        [0.02916347, 0.55319149, 0.2158209 ],
        [0.04451266, 0.72340426, 0.26567164]]])
```

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

[40]:
```
array([0.04451266, 0.03837299])
```

[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:, :]
```

```python
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([[[0.13814275, 0.27659574, 0.40975124],
        [0.16116654, 0.27659574, 0.63084577],
        [0.13737529, 0.27659574, 0.8519403 ],
        [0.12586339, 0.27659574, 0.70223881],
        [0.11588642, 0.36170213, 0.55253731],
        [0.10744436, 0.31914894, 0.40283582],
        [0.05295472, 0.29787234, 0.25313433],
        [0.07751343, 0.53191489, 0.44338308],
        [0.06062932, 0.57446809, 0.63363184],
        [0.12586339, 0.27659574, 0.8238806 ],
        [0.14121259, 0.31914894, 0.72179104],
        [0.15272448, 0.34042553, 0.61970149],
        [0.12970069, 0.23404255, 0.51761194],
        [0.06369916, 0.21276596, 0.41552239]],

       [[0.16116654, 0.27659574, 0.63084577],
        [0.13737529, 0.27659574, 0.8519403 ],
        [0.12586339, 0.27659574, 0.70223881],
        [0.11588642, 0.36170213, 0.55253731],
        [0.10744436, 0.31914894, 0.40283582],
        [0.05295472, 0.29787234, 0.25313433],
        [0.07751343, 0.53191489, 0.44338308],
        [0.06062932, 0.57446809, 0.63363184],
        [0.12586339, 0.27659574, 0.8238806 ],
        [0.14121259, 0.31914894, 0.72179104],
        [0.15272448, 0.34042553, 0.61970149],
```

```
        [0.12970069, 0.23404255, 0.51761194],
        [0.06369916, 0.21276596, 0.41552239],
        [0.1143515 , 0.25531915, 0.54208955]]])
```

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

[43]:
```
array([0.1143515 , 0.13814275])
```

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

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

[45]:
```python
# 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]:
```python
# 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 [==============================] - 11s 32ms/step - loss: 0.0528 -
val_loss: 0.0064
```

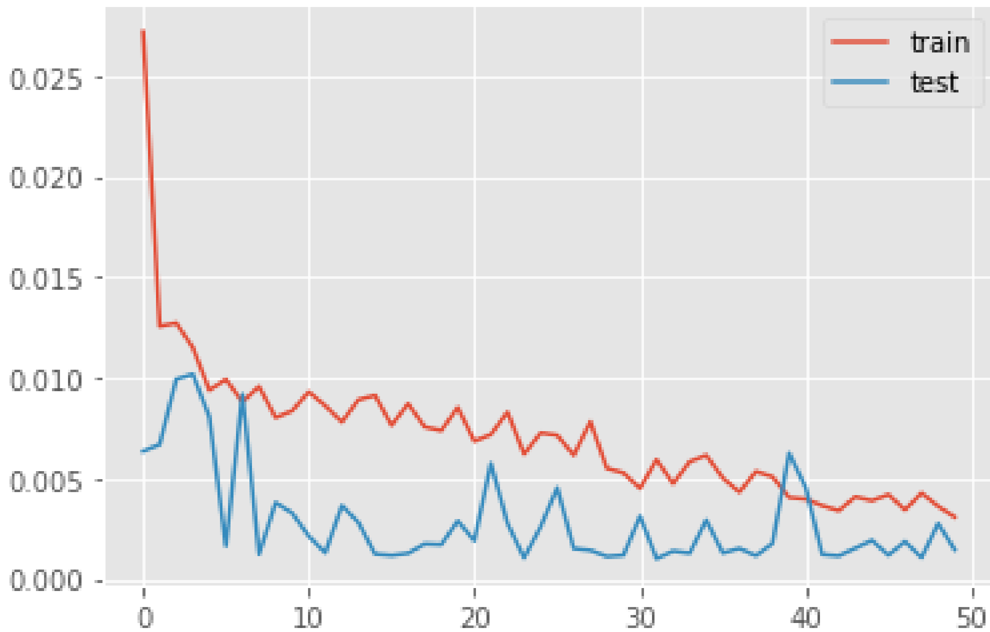
```
Epoch 2/50
130/130 [==============================] - 4s 28ms/step - loss: 0.0146 -
val_loss: 0.0067
Epoch 3/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0120 -
val_loss: 0.0099
Epoch 4/50
130/130 [==============================] - 3s 19ms/step - loss: 0.0129 -
val_loss: 0.0102
Epoch 5/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0107 -
val_loss: 0.0081
Epoch 6/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0096 -
val_loss: 0.0017
Epoch 7/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0083 -
val_loss: 0.0092
Epoch 8/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0077 -
val_loss: 0.0012
Epoch 9/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0064 -
val_loss: 0.0038
Epoch 10/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0086 -
val_loss: 0.0033
Epoch 11/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0088 -
val_loss: 0.0021
Epoch 12/50
130/130 [==============================] - 4s 33ms/step - loss: 0.0083 -
val_loss: 0.0013
Epoch 13/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0088 -
val_loss: 0.0037
Epoch 14/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0104 -
val_loss: 0.0028
Epoch 15/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0087 -
val_loss: 0.0013
Epoch 16/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0079 -
val_loss: 0.0012
Epoch 17/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0072 -
val_loss: 0.0013
```

```
Epoch 18/50
130/130 [==============================] - 4s 32ms/step - loss: 0.0065 -
val_loss: 0.0018
Epoch 19/50
130/130 [==============================] - 6s 47ms/step - loss: 0.0067 -
val_loss: 0.0017
Epoch 20/50
130/130 [==============================] - 3s 27ms/step - loss: 0.0081 -
val_loss: 0.0029
Epoch 21/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0078 -
val_loss: 0.0019
Epoch 22/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0074 -
val_loss: 0.0058
Epoch 23/50
130/130 [==============================] - 5s 40ms/step - loss: 0.0087 -
val_loss: 0.0028
Epoch 24/50
130/130 [==============================] - 3s 27ms/step - loss: 0.0056 -
val_loss: 0.0011
Epoch 25/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0082 -
val_loss: 0.0026
Epoch 26/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0064 -
val_loss: 0.0045
Epoch 27/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0067 -
val_loss: 0.0015
Epoch 28/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0061 -
val_loss: 0.0014
Epoch 29/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0064 -
val_loss: 0.0011
Epoch 30/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0046 -
val_loss: 0.0012
Epoch 31/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0047 -
val_loss: 0.0031
Epoch 32/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0051 -
val_loss: 0.0010
Epoch 33/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0038 -
val_loss: 0.0014
```

```
Epoch 34/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0072 -
val_loss: 0.0013
Epoch 35/50
130/130 [==============================] - 4s 30ms/step - loss: 0.0036 -
val_loss: 0.0029
Epoch 36/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0051 -
val_loss: 0.0013
Epoch 37/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0046 -
val_loss: 0.0015
Epoch 38/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0066 -
val_loss: 0.0012
Epoch 39/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0035 -
val_loss: 0.0018
Epoch 40/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0044 -
val_loss: 0.0063
Epoch 41/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0044 -
val_loss: 0.0046
Epoch 42/50
130/130 [==============================] - 5s 37ms/step - loss: 0.0027 -
val_loss: 0.0012
Epoch 43/50
130/130 [==============================] - 5s 36ms/step - loss: 0.0038 -
val_loss: 0.0012
Epoch 44/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0035 -
val_loss: 0.0015
Epoch 45/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0035 -
val_loss: 0.0019
Epoch 46/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0044 -
val_loss: 0.0012
Epoch 47/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0030 -
val_loss: 0.0019
Epoch 48/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0055 -
val_loss: 0.0011
Epoch 49/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0037 -
val_loss: 0.0028
```

```
Epoch 50/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0039 -
val_loss: 0.0015
```

[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([[0.09405538],
       [0.10105915],
       [0.10798825],
       [0.10372518],
       [0.09967114],
       [0.09578478],
       [0.08762793],
       [0.09877619],
       [0.10526647],
       [0.12263749],
       [0.12319554],
       [0.12236504],
```

```
        [0.11492747],
        [0.11019425],
        [0.11405569],
        [0.12008657],
        [0.12597911]], 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: 38.3
RMSE: 49.8
```

[51]:
```
# 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:]
```

[52]:
```
valid_v2.insert(1, "Predictions", pred_unscaled_v2, True)
valid_v2.insert(1, "Difference", valid_v2["Predictions"] - valid_v2["num_casos.
 ↪x"], True)
```

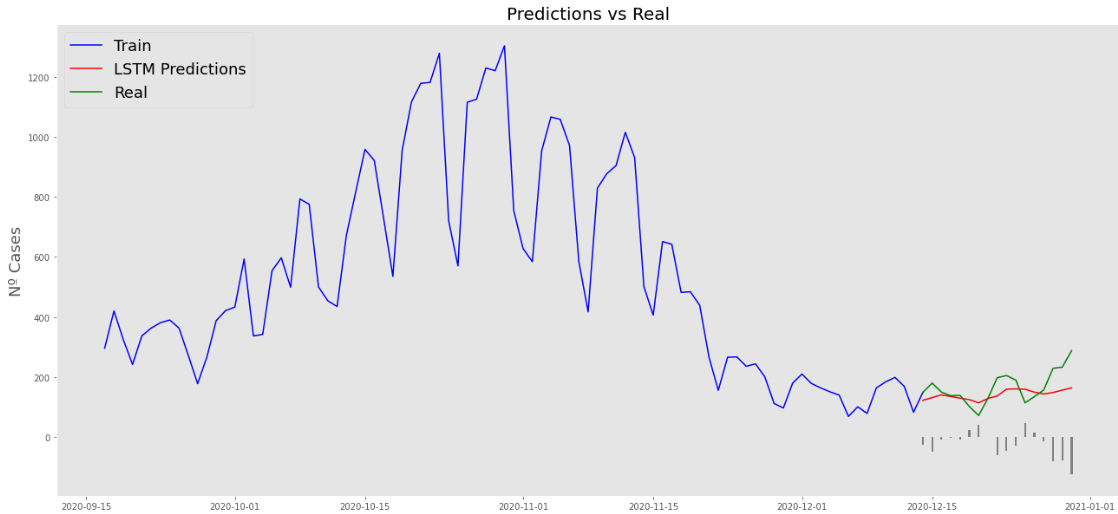
[53]:
```
# Zoom-in to a closer timeframe
valid_v2 = valid_v2[valid_v2.index > display_start_date_v2]
train_v2 = train_v2[train_v2.index > display_start_date_v2]

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

[53]:
```
            num_casos.x  Difference  Predictions  \
fecha
2020-12-14          149  -26.445847   122.554153
2020-12-15          180  -48.319916   131.680084
2020-12-16          150   -9.291321   140.708679
2020-12-17          138   -2.846085   135.153915
2020-12-18          139   -9.128510   129.871490
2020-12-19          102   22.807571   124.807571
```

```
2020-12-20           72   42.179199   114.179199
2020-12-21          129   -0.294617   128.705383
2020-12-22          198  -60.837784   137.162216
2020-12-23          205  -45.203354   159.796646
2020-12-24          190  -29.476212   160.523788
2020-12-25          114   45.441635   159.441635
2020-12-26          135   14.750488   149.750488
2020-12-27          157  -13.416885   143.583115
2020-12-28          229  -80.385437   148.614563
2020-12-29          233  -76.527206   156.472794
2020-12-30          288 -123.849213   164.150787

            residential_percent_change_from_baseline      total
fecha
2020-12-14                                       7.0  14.520000
2020-12-15                                       6.0  16.640000
2020-12-16                                       9.0  18.760000
2020-12-17                                       7.0  17.215000
2020-12-18                                       6.0  15.670000
2020-12-19                                       8.0  14.125000
2020-12-20                                       2.0  12.580000
2020-12-21                                       5.0  14.903333
2020-12-22                                       4.0  17.226667
2020-12-23                                       6.0  19.550000
2020-12-24                                       9.0  17.727500
2020-12-25                                      20.0  15.905000
2020-12-26                                       7.0  14.082500
2020-12-27                                       3.0  12.260000
2020-12-28                                       9.0  14.273333
2020-12-29                                       8.0  16.286667
2020-12-30                                       7.0  18.300000
```

[54]:
```python
# 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

[55]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data_v3 = Sev.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]
```

[55]:
```
array([[0.07290867, 0.55319149, 0.18888889, 0.37012987, 0.18367347,
        0.275     , 0.38383838, 0.40776119],
       [0.08211819, 0.63829787, 0.13333333, 0.28571429, 0.12244898,
        0.2       , 0.29292929, 0.36358209],
       [0.06676899, 0.65957447, 0.13333333, 0.29220779, 0.13265306,
        0.1625    , 0.27272727, 0.31940299],
       [0.08749041, 0.68085106, 0.12222222, 0.27272727, 0.12244898,
        0.15      , 0.25252525, 0.30597015],
       [0.11281658, 0.76595745, 0.1       , 0.27922078, 0.09183673,
        0.125     , 0.25252525, 0.29253731]])
```

[56]:
```
# 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(Sev['num_casos.x'])
np_cases_scaled_v3 = scaler_v3_pred.fit_transform(df_cases)
np_cases_scaled_v3[0:5]
```

[56]:
```
array([[0.07290867],
       [0.08211819],
       [0.06676899],
       [0.08749041],
       [0.11281658]])
```

[57]:
```
# Create the training data
train_data_v3 = scaled_data_v3[0:training_data_length_v3, :]
print(train_data_v3.shape)
```

```
(273, 8)
```

[58]:
```
train_data_v3[0:2]
```

[58]:
```
array([[0.07290867, 0.55319149, 0.18888889, 0.37012987, 0.18367347,
        0.275     , 0.38383838, 0.40776119],
       [0.08211819, 0.63829787, 0.13333333, 0.28571429, 0.12244898,
        0.2       , 0.29292929, 0.36358209]])
```

[59]:
```
training_data_length_v3
```

[59]:
```
273
```

```
[60]: 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]
```

```
[60]: array([[[0.07290867, 0.55319149, 0.18888889, 0.37012987, 0.18367347,
         0.275     , 0.38383838, 0.40776119],
        [0.08211819, 0.63829787, 0.13333333, 0.28571429, 0.12244898,
         0.2       , 0.29292929, 0.36358209],
        [0.06676899, 0.65957447, 0.13333333, 0.29220779, 0.13265306,
         0.1625    , 0.27272727, 0.31940299],
        [0.08749041, 0.68085106, 0.12222222, 0.27272727, 0.12244898,
         0.15      , 0.25252525, 0.30597015],
        [0.11281658, 0.76595745, 0.1       , 0.27922078, 0.09183673,
         0.125     , 0.25252525, 0.29253731],
        [0.07904835, 0.68085106, 0.04444444, 0.20779221, 0.02040816,
         0.0625    , 0.24242424, 0.14626866],
        [0.10590944, 0.57446809, 0.02222222, 0.09090909, 0.        ,
         0.05      , 0.21212121, 0.        ],
        [0.10590944, 0.65957447, 0.13333333, 0.27272727, 0.1122449 ,
         0.1375    , 0.23232323, 0.18179104],
        [0.06830391, 0.70212766, 0.11111111, 0.24025974, 0.1122449 ,
         0.125     , 0.21212121, 0.36358209],
        [0.09439754, 0.70212766, 0.1       , 0.22727273, 0.10204082,
         0.125     , 0.21212121, 0.36029851],
        [0.06830391, 0.72340426, 0.1       , 0.22727273, 0.10204082,
         0.1125    , 0.2020202 , 0.35701493],
        [0.05832694, 0.80851064, 0.08888889, 0.24025974, 0.08163265,
         0.1       , 0.21212121, 0.26149254],
        [0.02839601, 0.65957447, 0.05555556, 0.2012987 , 0.04081633,
         0.0625    , 0.24242424, 0.16597015],
        [0.02916347, 0.55319149, 0.02222222, 0.06493506, 0.01020408,
         0.0375    , 0.23232323, 0.2158209 ]],

       [[0.08211819, 0.63829787, 0.13333333, 0.28571429, 0.12244898,
         0.2       , 0.29292929, 0.36358209],
```

```
        [0.06676899, 0.65957447, 0.13333333, 0.29220779, 0.13265306,
         0.1625    , 0.27272727, 0.31940299],
        [0.08749041, 0.68085106, 0.12222222, 0.27272727, 0.12244898,
         0.15      , 0.25252525, 0.30597015],
        [0.11281658, 0.76595745, 0.1       , 0.27922078, 0.09183673,
         0.125     , 0.25252525, 0.29253731],
        [0.07904835, 0.68085106, 0.04444444, 0.20779221, 0.02040816,
         0.0625    , 0.24242424, 0.14626866],
        [0.10590944, 0.57446809, 0.02222222, 0.09090909, 0.        ,
         0.05      , 0.21212121, 0.        ],
        [0.10590944, 0.65957447, 0.13333333, 0.27272727, 0.1122449 ,
         0.1375    , 0.23232323, 0.18179104],
        [0.06830391, 0.70212766, 0.11111111, 0.24025974, 0.1122449 ,
         0.125     , 0.21212121, 0.36358209],
        [0.09439754, 0.70212766, 0.1       , 0.22727273, 0.10204082,
         0.125     , 0.21212121, 0.36029851],
        [0.06830391, 0.72340426, 0.1       , 0.22727273, 0.10204082,
         0.1125    , 0.2020202 , 0.35701493],
        [0.05832694, 0.80851064, 0.08888889, 0.24025974, 0.08163265,
         0.1       , 0.21212121, 0.26149254],
        [0.02839601, 0.65957447, 0.05555556, 0.2012987 , 0.04081633,
         0.0625    , 0.24242424, 0.16597015],
        [0.02916347, 0.55319149, 0.02222222, 0.06493506, 0.01020408,
         0.0375    , 0.23232323, 0.2158209 ],
        [0.04451266, 0.72340426, 0.1       , 0.21428571, 0.09183673,
         0.1       , 0.15151515, 0.26567164]]])
```

[61]:
```
y_train_v3[0:2]
```

[61]:
```
array([0.04451266, 0.03837299])
```

[62]:
```
print(x_train_v3.shape, y_train_v3.shape)
```

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

[63]:
```
# 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)
```

[63]:

```
array([[[0.13814275, 0.27659574, 0.74444444, 0.56493506, 0.69387755,
         0.8       , 0.72727273, 0.40975124],
        [0.16116654, 0.27659574, 0.74444444, 0.58441558, 0.7755102 ,
         0.8125    , 0.73737374, 0.63084577],
        [0.13737529, 0.27659574, 0.73333333, 0.5974026 , 0.74489796,
         0.7875    , 0.73737374, 0.8519403 ],
        [0.12586339, 0.27659574, 0.74444444, 0.61038961, 0.71428571,
         0.7875    , 0.73737374, 0.70223881],
        [0.11588642, 0.36170213, 0.65555556, 0.58441558, 0.47959184,
         0.7375    , 0.71717172, 0.55253731],
        [0.10744436, 0.31914894, 0.7       , 0.63636364, 0.65306122,
         0.7       , 0.81818182, 0.40283582],
        [0.05295472, 0.29787234, 0.6       , 0.61038961, 0.57142857,
         0.625     , 0.76767677, 0.25313433],
        [0.07751343, 0.53191489, 0.74444444, 0.53896104, 0.70408163,
         0.5125    , 0.22222222, 0.44338308],
        [0.06062932, 0.57446809, 0.57777778, 0.28571429, 0.6122449 ,
         0.4375    , 0.19191919, 0.63363184],
        [0.12586339, 0.27659574, 0.73333333, 0.64935065, 0.55102041,
         0.775     , 0.72727273, 0.8238806 ],
        [0.14121259, 0.31914894, 0.71111111, 0.6038961 , 0.51020408,
         0.7125    , 0.72727273, 0.72179104],
        [0.15272448, 0.34042553, 0.67777778, 0.62337662, 0.66326531,
         0.75      , 0.75757576, 0.61970149],
        [0.12970069, 0.23404255, 0.77777778, 0.61688312, 0.79591837,
         0.85      , 0.83838384, 0.51761194],
        [0.06369916, 0.21276596, 0.77777778, 0.8961039 , 0.63265306,
         0.8125    , 0.85858586, 0.41552239]],

       [[0.16116654, 0.27659574, 0.74444444, 0.58441558, 0.7755102 ,
         0.8125    , 0.73737374, 0.63084577],
        [0.13737529, 0.27659574, 0.73333333, 0.5974026 , 0.74489796,
         0.7875    , 0.73737374, 0.8519403 ],
        [0.12586339, 0.27659574, 0.74444444, 0.61038961, 0.71428571,
         0.7875    , 0.73737374, 0.70223881],
        [0.11588642, 0.36170213, 0.65555556, 0.58441558, 0.47959184,
```

```
 0.7375    , 0.71717172, 0.55253731],
[0.10744436, 0.31914894, 0.7       , 0.63636364, 0.65306122,
 0.7       , 0.81818182, 0.40283582],
[0.05295472, 0.29787234, 0.6       , 0.61038961, 0.57142857,
 0.625     , 0.76767677, 0.25313433],
[0.07751343, 0.53191489, 0.74444444, 0.53896104, 0.70408163,
 0.5125    , 0.22222222, 0.44338308],
[0.06062932, 0.57446809, 0.57777778, 0.28571429, 0.6122449 ,
 0.4375    , 0.19191919, 0.63363184],
[0.12586339, 0.27659574, 0.73333333, 0.64935065, 0.55102041,
 0.775     , 0.72727273, 0.8238806 ],
[0.14121259, 0.31914894, 0.71111111, 0.6038961 , 0.51020408,
 0.7125    , 0.72727273, 0.72179104],
[0.15272448, 0.34042553, 0.67777778, 0.62337662, 0.66326531,
 0.75      , 0.75757576, 0.61970149],
[0.12970069, 0.23404255, 0.77777778, 0.61688312, 0.79591837,
 0.85      , 0.83838384, 0.51761194],
[0.06369916, 0.21276596, 0.77777778, 0.8961039 , 0.63265306,
 0.8125    , 0.85858586, 0.41552239],
[0.1143515 , 0.25531915, 0.82222222, 0.57142857, 0.60204082,
 0.8375    , 0.73737374, 0.54208955]]])
```

[64]:
```
y_test_v3[0:2]
```

[64]:
```
array([0.1143515 , 0.13814275])
```

[65]:
```
print(x_test_v3.shape, y_test_v3.shape)
```

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

[66]:
```
# 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
```

[67]:
```
# 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]
                         )
```

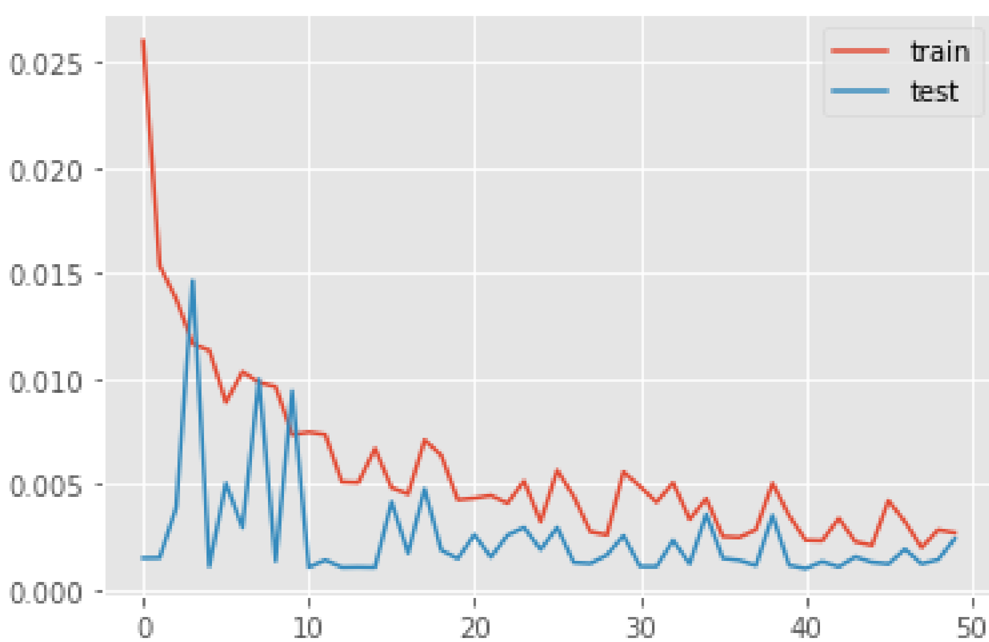
```
Epoch 1/50
130/130 [==============================] - 10s 21ms/step - loss: 0.0379 -
val_loss: 0.0015
Epoch 2/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0135 -
val_loss: 0.0015
Epoch 3/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0164 -
val_loss: 0.0039
Epoch 4/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0141 -
val_loss: 0.0146
Epoch 5/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0142 -
val_loss: 0.0011
Epoch 6/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0087 -
val_loss: 0.0051
Epoch 7/50
130/130 [==============================] - 5s 35ms/step - loss: 0.0123 -
val_loss: 0.0030
Epoch 8/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0059 -
val_loss: 0.0100
Epoch 9/50
130/130 [==============================] - 4s 32ms/step - loss: 0.0115 -
val_loss: 0.0014
Epoch 10/50
130/130 [==============================] - 6s 50ms/step - loss: 0.0080 -
val_loss: 0.0094
Epoch 11/50
130/130 [==============================] - 6s 44ms/step - loss: 0.0107 -
```

```
val_loss: 0.0011
Epoch 12/50
130/130 [==============================] - 3s 27ms/step - loss: 0.0097 -
val_loss: 0.0014
Epoch 13/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0041 -
val_loss: 0.0011
Epoch 14/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0050 -
val_loss: 0.0011
Epoch 15/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0108 -
val_loss: 0.0011
Epoch 16/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0036 -
val_loss: 0.0042
Epoch 17/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0037 -
val_loss: 0.0017
Epoch 18/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0057 -
val_loss: 0.0048
Epoch 19/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0054 -
val_loss: 0.0019
Epoch 20/50
130/130 [==============================] - 3s 24ms/step - loss: 0.0050 -
val_loss: 0.0015
Epoch 21/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0030 -
val_loss: 0.0026
Epoch 22/50
130/130 [==============================] - 4s 33ms/step - loss: 0.0043 -
val_loss: 0.0016
Epoch 23/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0033 -
val_loss: 0.0026
Epoch 24/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0047 -
val_loss: 0.0030
Epoch 25/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0034 -
val_loss: 0.0019
Epoch 26/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0063 -
val_loss: 0.0030
Epoch 27/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0034 -
```

```
val_loss: 0.0013
Epoch 28/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0019 -
val_loss: 0.0012
Epoch 29/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0031 -
val_loss: 0.0017
Epoch 30/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0036 -
val_loss: 0.0026
Epoch 31/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0078 -
val_loss: 0.0011
Epoch 32/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0027 -
val_loss: 0.0011
Epoch 33/50
130/130 [==============================] - 6s 49ms/step - loss: 0.0065 -
val_loss: 0.0023
Epoch 34/50
130/130 [==============================] - 5s 39ms/step - loss: 0.0039 -
val_loss: 0.0012
Epoch 35/50
130/130 [==============================] - 6s 50ms/step - loss: 0.0053 -
val_loss: 0.0036
Epoch 36/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0029 -
val_loss: 0.0015
Epoch 37/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0027 -
val_loss: 0.0014
Epoch 38/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0030 -
val_loss: 0.0012
Epoch 39/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0028 -
val_loss: 0.0035
Epoch 40/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0041 -
val_loss: 0.0012
Epoch 41/50
130/130 [==============================] - 5s 38ms/step - loss: 0.0020 -
val_loss: 0.0010
Epoch 42/50
130/130 [==============================] - 14s 108ms/step - loss: 0.0024 -
val_loss: 0.0013
Epoch 43/50
130/130 [==============================] - 6s 46ms/step - loss: 0.0049 -
```

```
val_loss: 0.0011
Epoch 44/50
130/130 [==============================] - 6s 43ms/step - loss: 0.0029 -
val_loss: 0.0016
Epoch 45/50
130/130 [==============================] - 5s 40ms/step - loss: 0.0019 -
val_loss: 0.0013
Epoch 46/50
130/130 [==============================] - 5s 37ms/step - loss: 0.0041 -
val_loss: 0.0012
Epoch 47/50
130/130 [==============================] - 4s 30ms/step - loss: 0.0031 -
val_loss: 0.0019
Epoch 48/50
130/130 [==============================] - 3s 22ms/step - loss: 0.0024 -
val_loss: 0.0012
Epoch 49/50
130/130 [==============================] - 5s 37ms/step - loss: 0.0021 -
val_loss: 0.0014
Epoch 50/50
130/130 [==============================] - 4s 31ms/step - loss: 0.0021 -
val_loss: 0.0024
```

[68]:
```
# Plot history
plt.plot(history_v3.history['loss'], label='train')
plt.plot(history_v3.history['val_loss'], label='test')
plt.legend()
plt.show()
```



[69]:
```python
# Get the predicted values
predictions_v3 = model_v3.predict(x_test_v3)
predictions_v3
```

[69]:
```
array([[0.07375626],
       [0.0737868 ],
       [0.07921986],
       [0.08665624],
       [0.08618884],
       [0.07932421],
       [0.06605157],
       [0.0643347 ],
       [0.07042015],
       [0.10393722],
       [0.11234134],
       [0.11699536],
       [0.1097768 ],
       [0.09997277],
       [0.10374874],
       [0.11485538],
       [0.11727484]], dtype=float32)
```

[70]:
```python
# 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))
```

[71]:
```python
# 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: 53.0
RMSE: 64.1
```

[72]:
```python
# 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:]
```

[73]:
```
valid_v3.insert(1, "Predictions", pred_unscaled_v3, True)
valid_v3.insert(1, "Difference", valid_v3["Predictions"] - valid_v3["num_casos.
↪x"], True)
```

[74]:
```
# 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
```

[74]:
```
            num_casos.x  Difference  Predictions  \
fecha
2020-12-14          149  -52.895599    96.104401
2020-12-15          180  -83.855797    96.144203
2020-12-16          150  -46.776527   103.223473
2020-12-17          138  -25.086914   112.913086
2020-12-18          139  -26.695946   112.304054
2020-12-19          102    1.359444   103.359444
2020-12-20           72   14.065201    86.065201
2020-12-21          129  -45.171890    83.828110
2020-12-22          198 -106.242546    91.757454
2020-12-23          205  -69.569794   135.430206
2020-12-24          190  -43.619232   146.380768
2020-12-25          114   38.444962   152.444962
2020-12-26          135    8.039169   143.039169
2020-12-27          157  -26.735474   130.264526
2020-12-28          229  -93.815399   135.184601
2020-12-29          233  -83.343445   149.656555
2020-12-30          288 -135.190887   152.809113

            residential_percent_change_from_baseline  \
fecha
2020-12-14                                       7.0
2020-12-15                                       6.0
2020-12-16                                       9.0
2020-12-17                                       7.0
2020-12-18                                       6.0
2020-12-19                                       8.0
2020-12-20                                       2.0
2020-12-21                                       5.0
2020-12-22                                       4.0
2020-12-23                                       6.0
2020-12-24                                       9.0
2020-12-25                                      20.0
2020-12-26                                       7.0
2020-12-27                                       3.0
```

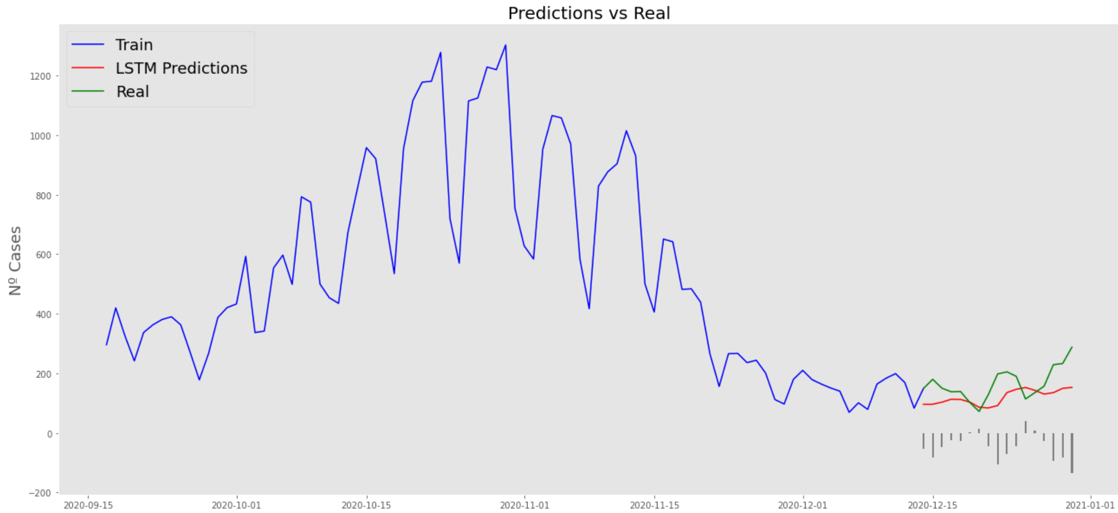
```
2020-12-28                                       9.0
2020-12-29                                       8.0
2020-12-30                                       7.0

            retail_and_recreation_percent_change_from_baseline  \
fecha
2020-12-14                                              -23.0
2020-12-15                                              -21.0
2020-12-16                                              -30.0
2020-12-17                                              -22.0
2020-12-18                                              -24.0
2020-12-19                                              -35.0
2020-12-20                                              -17.0
2020-12-21                                               -9.0
2020-12-22                                               -8.0
2020-12-23                                               -7.0
2020-12-24                                              -25.0
2020-12-25                                              -71.0
2020-12-26                                              -33.0
2020-12-27                                              -22.0
2020-12-28                                              -10.0
2020-12-29                                              -10.0
2020-12-30                                               -8.0

            grocery_and_pharmacy_percent_change_from_baseline  \
fecha
2020-12-14                                              -3.0
2020-12-15                                               4.0
2020-12-16                                              -4.0
2020-12-17                                               8.0
2020-12-18                                               4.0
2020-12-19                                              -7.0
2020-12-20                                              63.0
2020-12-21                                              11.0
2020-12-22                                              14.0
2020-12-23                                              34.0
2020-12-24                                              12.0
2020-12-25                                             -78.0
2020-12-26                                             -12.0
2020-12-27                                              50.0
2020-12-28                                               5.0
2020-12-29                                              12.0
2020-12-30                                              31.0

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

```
2020-12-15                                  -16.0
2020-12-16                                  -43.0
2020-12-17                                  -18.0
2020-12-18                                  -16.0
2020-12-19                                  -56.0
2020-12-20                                  -24.0
2020-12-21                                  -10.0
2020-12-22                                   -3.0
2020-12-23                                   -9.0
2020-12-24                                  -30.0
2020-12-25                                  -37.0
2020-12-26                                  -31.0
2020-12-27                                  -27.0
2020-12-28                                  -15.0
2020-12-29                                   -5.0
2020-12-30                                   -4.0

            transit_stations_percent_change_from_baseline  \
fecha
2020-12-14                                          -27.0
2020-12-15                                          -23.0
2020-12-16                                          -30.0
2020-12-17                                          -25.0
2020-12-18                                          -21.0
2020-12-19                                          -34.0
2020-12-20                                          -15.0
2020-12-21                                          -19.0
2020-12-22                                          -17.0
2020-12-23                                          -20.0
2020-12-24                                          -46.0
2020-12-25                                          -69.0
2020-12-26                                          -31.0
2020-12-27                                          -19.0
2020-12-28                                          -28.0
2020-12-29                                          -28.0
2020-12-30                                          -26.0

            workplaces_percent_change_from_baseline      total
fecha
2020-12-14                                    -17.0  14.520000
2020-12-15                                    -16.0  16.640000
2020-12-16                                    -17.0  18.760000
2020-12-17                                    -17.0  17.215000
2020-12-18                                    -16.0  15.670000
2020-12-19                                    -12.0  14.125000
2020-12-20                                      0.0  12.580000
2020-12-21                                    -20.0  14.903333
```

```
2020-12-22                                      -22.0  17.226667
2020-12-23                                      -33.0  19.550000
2020-12-24                                      -49.0  17.727500
2020-12-25                                      -79.0  15.905000
2020-12-26                                      -16.0  14.082500
2020-12-27                                       -4.0  12.260000
2020-12-28                                      -39.0  14.273333
2020-12-29                                      -38.0  16.286667
2020-12-30                                      -38.0  18.300000
```

[75]:

```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
1200
1000
800
600
400
200
0
−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


```
[ ]:
```