LSTM_arodriguezsans-Univariate_Multivariate_Bar

May 18, 2021

1 Barcelona

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]:
```
Bar=df_total.loc[df_total['sub_region_2'] == 'Barcelona']
#Bar.describe()
```

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

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

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(Bar.values[:, group])
    plt.title(Bar.columns[group], y=0.5, fontsize=10, loc='left')
    i += 1
plt.show()
```


num_casos.x
retail_and_recreation_percent_change_from_baseline
grocery_and_pharmacy_percent_change_from_baseline
parks_percent_change_from_baseline
workplaces_percent_change_from_baseline
residential_percent_change_from_baseline
total

1.5 LSTM - Univariate

[7]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data = Bar.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([[0.34230487],
       [0.31416687],
       [0.34132616],
       [0.32126254],
       [0.38879374]])
```

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

[8]:
```
array([[1424],
       [1309],
       [1420],
       [1338],
       [1614]], dtype=int64)
```

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

[9]: 290

[10]:
```
training_data_length
```

[10]: 273

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

[12]:
```python
# 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]:
```python
# list
x_train[0:2]
```

[13]:
```
[array([0.34230487, 0.31416687, 0.34132616, 0.32126254, 0.38879374,
        0.27868852, 0.25446538, 0.295816  , 0.27208221, 0.27746513,
        0.27477367, 0.24761439, 0.25935894, 0.19133839]),
 array([0.31416687, 0.34132616, 0.32126254, 0.38879374, 0.27868852,
        0.25446538, 0.295816  , 0.27208221, 0.27746513, 0.27477367,
        0.24761439, 0.25935894, 0.19133839, 0.23562515])]
```

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

[14]:
```
[0.23562515292390507, 0.22143381453388794]
```

[15]:
```python
# 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])
```

```
[[0.34230487 0.31416687 0.34132616 0.32126254 0.38879374 0.27868852
  0.25446538 0.295816   0.27208221 0.27746513 0.27477367 0.24761439
  0.25935894 0.19133839]
 [0.31416687 0.34132616 0.32126254 0.38879374 0.27868852 0.25446538
  0.295816   0.27208221 0.27746513 0.27477367 0.24761439 0.25935894
  0.19133839 0.23562515]]
------------------------------------------------------------
[0.23562515 0.22143381]
```

[16]:
```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,)
```

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

[17]:
```
array([[[0.34230487],
        [0.31416687],
        [0.34132616],
        [0.32126254],
        [0.38879374],
        [0.27868852],
        [0.25446538],
        [0.295816  ],
        [0.27208221],
        [0.27746513],
        [0.27477367],
        [0.24761439],
        [0.25935894],
        [0.19133839]],

       [[0.31416687],
        [0.34132616],
        [0.32126254],
        [0.38879374],
        [0.27868852],
        [0.25446538],
        [0.295816  ],
        [0.27208221],
        [0.27746513],
        [0.27477367],
        [0.24761439],
        [0.25935894],
        [0.19133839],
        [0.23562515]]])
```

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

[18]:
```
array([0.23562515, 0.22143381])
```

[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])
```

```
[[[0.22265721]
  [0.23562515]
  [0.21164668]
  [0.1866895 ]
  [0.18962564]
  [0.12698801]
  [0.09836066]
  [0.22559334]
  [0.13677514]
  [0.26694397]
  [0.28015659]
  [0.2879863 ]
  [0.18742354]
  [0.16344507]]

 [[0.23562515]
  [0.21164668]
  [0.1866895 ]
  [0.18962564]
  [0.12698801]
  [0.09836066]
  [0.22559334]
  [0.13677514]
  [0.26694397]
  [0.28015659]
  [0.2879863 ]
  [0.18742354]
  [0.16344507]
  [0.34719843]]]
------------------------------------------------------------
```

```
[0.34719843 0.35527282]
```

[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')
```

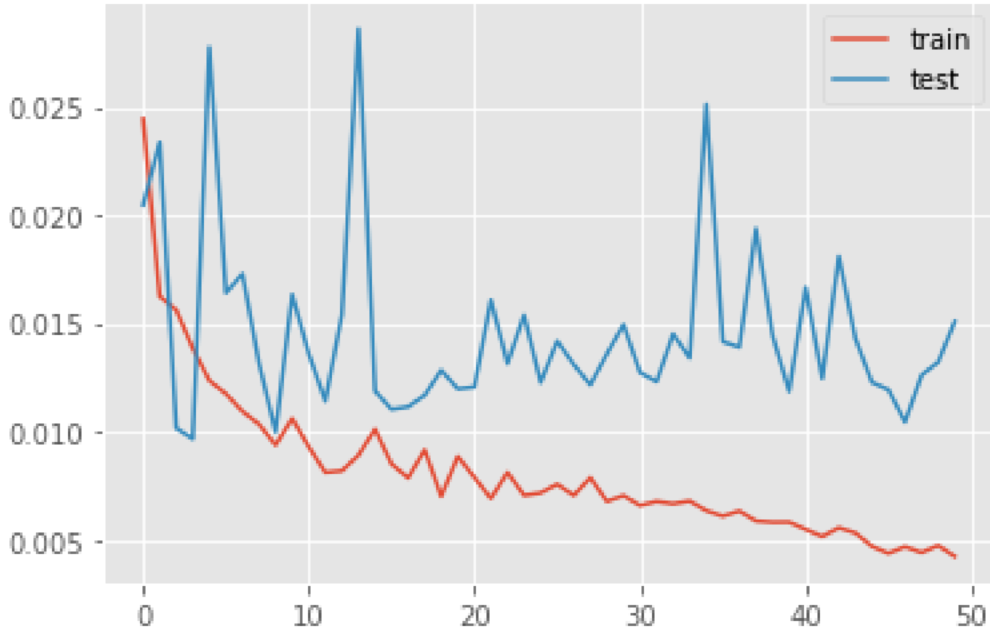
[22]:
```
# Training the model
#early_stop = EarlyStopping(monitor='loss', patience=2, verbose=1)
# fit network
history=model.fit(x_train,
                  y_train,
                  #callbacks=[early_stop],
                  batch_size=2,
                  epochs=50,
                  validation_data=(x_test,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

```
Epoch 1/50
130/130 - 10s - loss: 0.0245 - val_loss: 0.0205
Epoch 2/50
130/130 - 2s - loss: 0.0163 - val_loss: 0.0234
Epoch 3/50
130/130 - 1s - loss: 0.0156 - val_loss: 0.0102
Epoch 4/50
130/130 - 2s - loss: 0.0139 - val_loss: 0.0097
Epoch 5/50
130/130 - 2s - loss: 0.0124 - val_loss: 0.0278
Epoch 6/50
130/130 - 2s - loss: 0.0118 - val_loss: 0.0164
Epoch 7/50
130/130 - 2s - loss: 0.0109 - val_loss: 0.0173
Epoch 8/50
130/130 - 2s - loss: 0.0104 - val_loss: 0.0132
```

```
Epoch 9/50
130/130 - 2s - loss: 0.0094 - val_loss: 0.0100
Epoch 10/50
130/130 - 2s - loss: 0.0106 - val_loss: 0.0163
Epoch 11/50
130/130 - 2s - loss: 0.0093 - val_loss: 0.0136
Epoch 12/50
130/130 - 2s - loss: 0.0081 - val_loss: 0.0114
Epoch 13/50
130/130 - 2s - loss: 0.0082 - val_loss: 0.0154
Epoch 14/50
130/130 - 2s - loss: 0.0089 - val_loss: 0.0286
Epoch 15/50
130/130 - 2s - loss: 0.0101 - val_loss: 0.0119
Epoch 16/50
130/130 - 2s - loss: 0.0085 - val_loss: 0.0110
Epoch 17/50
130/130 - 2s - loss: 0.0079 - val_loss: 0.0112
Epoch 18/50
130/130 - 2s - loss: 0.0092 - val_loss: 0.0117
Epoch 19/50
130/130 - 2s - loss: 0.0070 - val_loss: 0.0128
Epoch 20/50
130/130 - 2s - loss: 0.0089 - val_loss: 0.0120
Epoch 21/50
130/130 - 2s - loss: 0.0079 - val_loss: 0.0121
Epoch 22/50
130/130 - 1s - loss: 0.0069 - val_loss: 0.0161
Epoch 23/50
130/130 - 1s - loss: 0.0081 - val_loss: 0.0131
Epoch 24/50
130/130 - 2s - loss: 0.0071 - val_loss: 0.0154
Epoch 25/50
130/130 - 2s - loss: 0.0072 - val_loss: 0.0123
Epoch 26/50
130/130 - 2s - loss: 0.0076 - val_loss: 0.0142
Epoch 27/50
130/130 - 2s - loss: 0.0071 - val_loss: 0.0131
Epoch 28/50
130/130 - 2s - loss: 0.0079 - val_loss: 0.0122
Epoch 29/50
130/130 - 2s - loss: 0.0068 - val_loss: 0.0136
Epoch 30/50
130/130 - 2s - loss: 0.0071 - val_loss: 0.0149
Epoch 31/50
130/130 - 2s - loss: 0.0066 - val_loss: 0.0127
Epoch 32/50
130/130 - 2s - loss: 0.0068 - val_loss: 0.0123
```

```
Epoch 33/50
130/130 - 2s - loss: 0.0067 - val_loss: 0.0145
Epoch 34/50
130/130 - 2s - loss: 0.0068 - val_loss: 0.0134
Epoch 35/50
130/130 - 2s - loss: 0.0064 - val_loss: 0.0251
Epoch 36/50
130/130 - 1s - loss: 0.0061 - val_loss: 0.0142
Epoch 37/50
130/130 - 2s - loss: 0.0063 - val_loss: 0.0139
Epoch 38/50
130/130 - 2s - loss: 0.0059 - val_loss: 0.0194
Epoch 39/50
130/130 - 2s - loss: 0.0058 - val_loss: 0.0145
Epoch 40/50
130/130 - 1s - loss: 0.0058 - val_loss: 0.0119
Epoch 41/50
130/130 - 2s - loss: 0.0055 - val_loss: 0.0167
Epoch 42/50
130/130 - 2s - loss: 0.0052 - val_loss: 0.0125
Epoch 43/50
130/130 - 2s - loss: 0.0056 - val_loss: 0.0181
Epoch 44/50
130/130 - 2s - loss: 0.0053 - val_loss: 0.0143
Epoch 45/50
130/130 - 2s - loss: 0.0047 - val_loss: 0.0123
Epoch 46/50
130/130 - 2s - loss: 0.0044 - val_loss: 0.0119
Epoch 47/50
130/130 - 2s - loss: 0.0047 - val_loss: 0.0105
Epoch 48/50
130/130 - 2s - loss: 0.0044 - val_loss: 0.0126
Epoch 49/50
130/130 - 2s - loss: 0.0047 - val_loss: 0.0132
Epoch 50/50
130/130 - 2s - loss: 0.0042 - val_loss: 0.0151
```

[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([[1078.8866],
       [1235.8221],
       [1461.0034],
       [1774.371 ],
       [2074.76  ],
       [1942.8036],
       [1796.0831],
       [1886.9486],
       [1748.0642],
       [2117.6973],
       [2270.7178],
       [2028.4099],
       [1533.3619],
       [1538.0133],
       [1923.3629],
       [2036.9807],
       [1969.241 ]], dtype=float32)
```

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

[26]:
```
array([[1444.],
       [1477.],
       [1320.],
       [1353.],
       [1477.],
       [1078.],
       [ 994.],
       [1785.],
       [1763.],
       [1616.],
       [1735.],
       [ 974.],
       [1262.],
       [1311.],
       [2440.],
       [2244.],
       [2197.]])
```

[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)))

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

```
MAE: 417.2
RMSE: 502.1
RMSE: 502.1
```

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

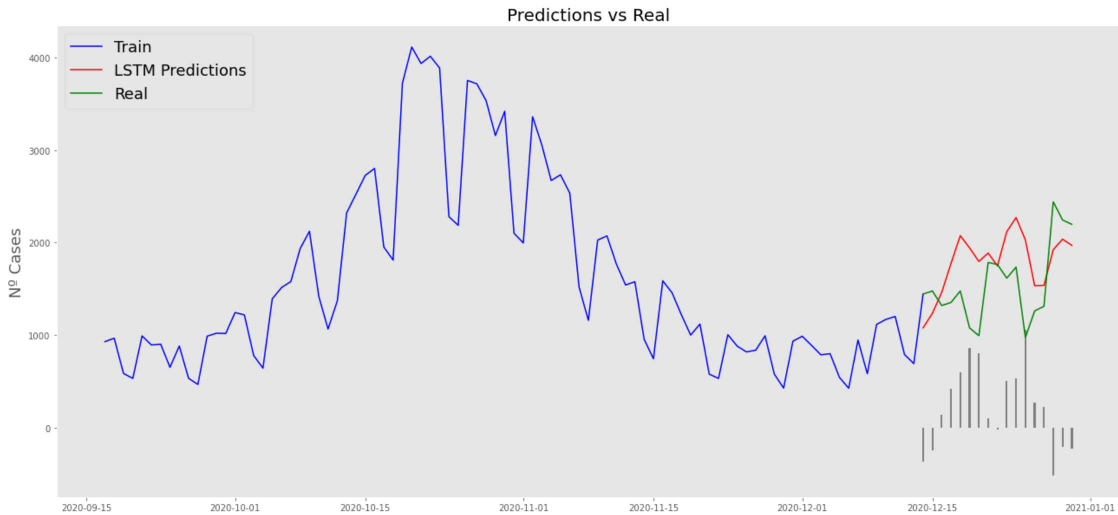
[31]:
```python
# Zoom-in to a closer timeframe
valid = valid[valid.index > display_start_date]
train = train[train.index > display_start_date]

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

[31]:

fecha	num_casos.x	Difference	Predictions
2020-12-14	1444	-365.113403	1078.886597
2020-12-15	1477	-241.177856	1235.822144
2020-12-16	1320	141.003418	1461.003418
2020-12-17	1353	421.370972	1774.370972
2020-12-18	1477	597.760010	2074.760010
2020-12-19	1078	864.803589	1942.803589
2020-12-20	994	802.083130	1796.083130
2020-12-21	1785	101.948608	1886.948608
2020-12-22	1763	-14.935791	1748.064209
2020-12-23	1616	501.697266	2117.697266
2020-12-24	1735	535.717773	2270.717773
2020-12-25	974	1054.409912	2028.409912
2020-12-26	1262	271.361938	1533.361938
2020-12-27	1311	227.013306	1538.013306
2020-12-28	2440	-516.637085	1923.362915
2020-12-29	2244	-207.019287	2036.980713
2020-12-30	2197	-227.759033	1969.240967

[32]:
```python
# Visualize the data
fig, ax1 = plt.subplots(figsize=(22, 10), sharex=True)

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

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

plt.plot(yt, color="blue", linewidth=1.5)
plt.plot(yv["Predictions"], color="red", linewidth=1.5)
plt.plot(yv["num_casos.x"], color="green", linewidth=1.5)
```

```
plt.legend(["Train", "LSTM Predictions", "Real"],
           loc="upper left", fontsize=18)

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



1.6 LSTM - 2 variables + infections reported

[33]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data_v2 = Bar.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([[0.34230487, 0.58      , 0.41473259],
       [0.31416687, 0.64      , 0.38446014],
       [0.34132616, 0.66      , 0.35418769],
       [0.32126254, 0.7       , 0.33551968],
       [0.38879374, 0.76      , 0.31685166]])
```

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

[34]:
```
array([[0.34230487],
       [0.31416687],
       [0.34132616],
       [0.32126254],
       [0.38879374]])
```

[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([[0.34230487, 0.58      , 0.41473259],
       [0.31416687, 0.64      , 0.38446014]])
```

[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([[[0.34230487, 0.58      , 0.41473259],
        [0.31416687, 0.64      , 0.38446014],
        [0.34132616, 0.66      , 0.35418769],
        [0.32126254, 0.7       , 0.33551968],
        [0.38879374, 0.76      , 0.31685166],
        [0.27868852, 0.58      , 0.16271443],
        [0.25446538, 0.48      , 0.00857719],
        [0.295816  , 0.7       , 0.15842583],
        [0.27208221, 0.72      , 0.30827447],
        [0.27746513, 0.74      , 0.3037336 ],
        [0.27477367, 0.78      , 0.29919273],
        [0.24761439, 0.8       , 0.21039354],
        [0.25935894, 0.6       , 0.12159435],
        [0.19133839, 0.5       , 0.15817356]],

       [[0.31416687, 0.64      , 0.38446014],
        [0.34132616, 0.66      , 0.35418769],
        [0.32126254, 0.7       , 0.33551968],
        [0.38879374, 0.76      , 0.31685166],
        [0.27868852, 0.58      , 0.16271443],
        [0.25446538, 0.48      , 0.00857719],
        [0.295816  , 0.7       , 0.15842583],
        [0.27208221, 0.72      , 0.30827447],
        [0.27746513, 0.74      , 0.3037336 ],
        [0.27477367, 0.78      , 0.29919273],
        [0.24761439, 0.8       , 0.21039354],
        [0.25935894, 0.6       , 0.12159435],
        [0.19133839, 0.5       , 0.15817356],
        [0.23562515, 0.78      , 0.19475277]]])
```

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

[40]:
```
array([0.23562515, 0.22143381])
```

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

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

[42]:
```
# Create the test data
test_data_v2 = scaled_data_v2[training_data_length_v2 - loop_back:, :]
```

```
print(test_data_v2.shape)

x_test_v2 = []
y_test_v2 = []
# The RNN needs data with the format of [samples, time steps, features].
# Here, we create N samples, N loop_back time steps per sample, and 3 features␣
 ↪(mobility + num_casos.x)
for i in range(loop_back, len(test_data_v2)):
    #print(i)
    #contains loop_back values ->>> 0-loop_back * columsn
    x_test_v2.append(test_data_v2[i-loop_back:i,:])
    #contains the prediction values for test / validation
    y_test_v2.append(test_data_v2[i, 0])

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

```
(31, 3)
```

[42]:
```
array([[[0.22265721, 0.22      , 0.47544568],
        [0.23562515, 0.24      , 0.6749075 ],
        [0.21164668, 0.24      , 0.87436932],
        [0.1866895 , 0.24      , 0.72262866],
        [0.18962564, 0.32      , 0.57088799],
        [0.12698801, 0.32      , 0.41914733],
        [0.09836066, 0.28      , 0.26740666],
        [0.22559334, 0.32      , 0.45021863],
        [0.13677514, 0.54      , 0.63303061],
        [0.26694397, 0.24      , 0.81584258],
        [0.28015659, 0.24      , 0.68276993],
        [0.2879863 , 0.32      , 0.54969728],
        [0.18742354, 0.3       , 0.41662462],
        [0.16344507, 0.26      , 0.28355197]],

       [[0.23562515, 0.24      , 0.6749075 ],
        [0.21164668, 0.24      , 0.87436932],
        [0.1866895 , 0.24      , 0.72262866],
        [0.18962564, 0.32      , 0.57088799],
        [0.12698801, 0.32      , 0.41914733],
        [0.09836066, 0.28      , 0.26740666],
        [0.22559334, 0.32      , 0.45021863],
        [0.13677514, 0.54      , 0.63303061],
        [0.26694397, 0.24      , 0.81584258],
        [0.28015659, 0.24      , 0.68276993],
        [0.2879863 , 0.32      , 0.54969728],
```

```
        [0.18742354, 0.3       , 0.41662462],
        [0.16344507, 0.26      , 0.28355197],
        [0.34719843, 0.22      , 0.48049109]]])
```

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

[43]:
```
array([0.34719843, 0.35527282])
```

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

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

[45]:
```
# Configure the neural network model
model_v2 = Sequential()

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

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

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

```
42 14 3
```

[46]:
```
# Training the model
early_stop_v2 = EarlyStopping(monitor='loss', patience=2, verbose=1)
history_v2 = model_v2.fit(x_train_v2,
                    y_train_v2,
                    batch_size=2,
                    validation_data=(x_test_v2, y_test_v2),
                    epochs=50
                    #callbacks=[early_stop_v2]
                         )
```

```
Epoch 1/50
130/130 [==============================] - 11s 28ms/step - loss: 0.0475 -
val_loss: 0.0281
```

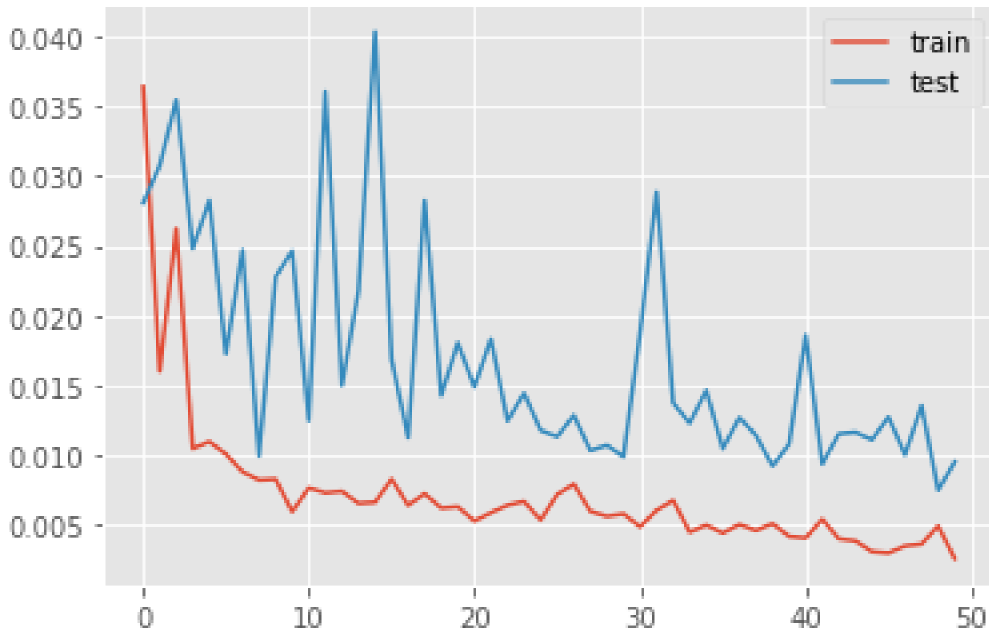
```
Epoch 2/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0159 -
val_loss: 0.0308
Epoch 3/50
130/130 [==============================] - 3s 23ms/step - loss: 0.0407 -
val_loss: 0.0355
Epoch 4/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0126 -
val_loss: 0.0248
Epoch 5/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0085 -
val_loss: 0.0283
Epoch 6/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0152 -
val_loss: 0.0173
Epoch 7/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0064 -
val_loss: 0.0247
Epoch 8/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0072 -
val_loss: 0.0100
Epoch 9/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0092 -
val_loss: 0.0228
Epoch 10/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0069 -
val_loss: 0.0247
Epoch 11/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0071 -
val_loss: 0.0125
Epoch 12/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0092 -
val_loss: 0.0361
Epoch 13/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0083 -
val_loss: 0.0150
Epoch 14/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0073 -
val_loss: 0.0218
Epoch 15/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0064 -
val_loss: 0.0404
Epoch 16/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0086 -
val_loss: 0.0169
Epoch 17/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0049 -
val_loss: 0.0113
```

```
Epoch 18/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0085 -
val_loss: 0.0283
Epoch 19/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0057 -
val_loss: 0.0143
Epoch 20/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0053 -
val_loss: 0.0181
Epoch 21/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0049 -
val_loss: 0.0149
Epoch 22/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0038 -
val_loss: 0.0184
Epoch 23/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0053 -
val_loss: 0.0125
Epoch 24/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0064 -
val_loss: 0.0145
Epoch 25/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0050 -
val_loss: 0.0118
Epoch 26/50
130/130 [==============================] - 2s 19ms/step - loss: 0.0045 -
val_loss: 0.0113
Epoch 27/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0079 -
val_loss: 0.0129
Epoch 28/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0062 -
val_loss: 0.0103
Epoch 29/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0062 -
val_loss: 0.0107
Epoch 30/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0065 -
val_loss: 0.0099
Epoch 31/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0074 -
val_loss: 0.0187
Epoch 32/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0043 -
val_loss: 0.0289
Epoch 33/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0098 -
val_loss: 0.0138
```

```
Epoch 34/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0028 -
val_loss: 0.0123
Epoch 35/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0046 -
val_loss: 0.0146
Epoch 36/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0028 -
val_loss: 0.0105
Epoch 37/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0063 -
val_loss: 0.0127
Epoch 38/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0051 -
val_loss: 0.0114
Epoch 39/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0056 -
val_loss: 0.0092
Epoch 40/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0056 -
val_loss: 0.0108
Epoch 41/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0034 -
val_loss: 0.0186
Epoch 42/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0036 -
val_loss: 0.0094
Epoch 43/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0047 -
val_loss: 0.0115
Epoch 44/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0036 -
val_loss: 0.0117
Epoch 45/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0030 -
val_loss: 0.0111
Epoch 46/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0041 -
val_loss: 0.0128
Epoch 47/50
130/130 [==============================] - 2s 13ms/step - loss: 0.0052 -
val_loss: 0.0100
Epoch 48/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0033 -
val_loss: 0.0136
Epoch 49/50
130/130 [==============================] - 2s 12ms/step - loss: 0.0050 -
val_loss: 0.0075
```

```
Epoch 50/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0025 -
val_loss: 0.0095
```

[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([[0.19510545],
       [0.23131315],
       [0.2506022 ],
       [0.23702295],
       [0.2179451 ],
       [0.19643812],
       [0.21949595],
       [0.3194039 ],
       [0.42641085],
       [0.56685257],
       [0.45487446],
       [0.35096836],
```

```
        [0.27229226],
        [0.29119015],
        [0.5153162 ],
        [0.6328563 ],
        [0.6494791 ]], 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: 343.5
RMSE: 399.1
```

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

[51]:
```
399.0607192982857
```

[52]:
```
# Date from which on the date is displayed
display_start_date_v2 = "2020-09-16"

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

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

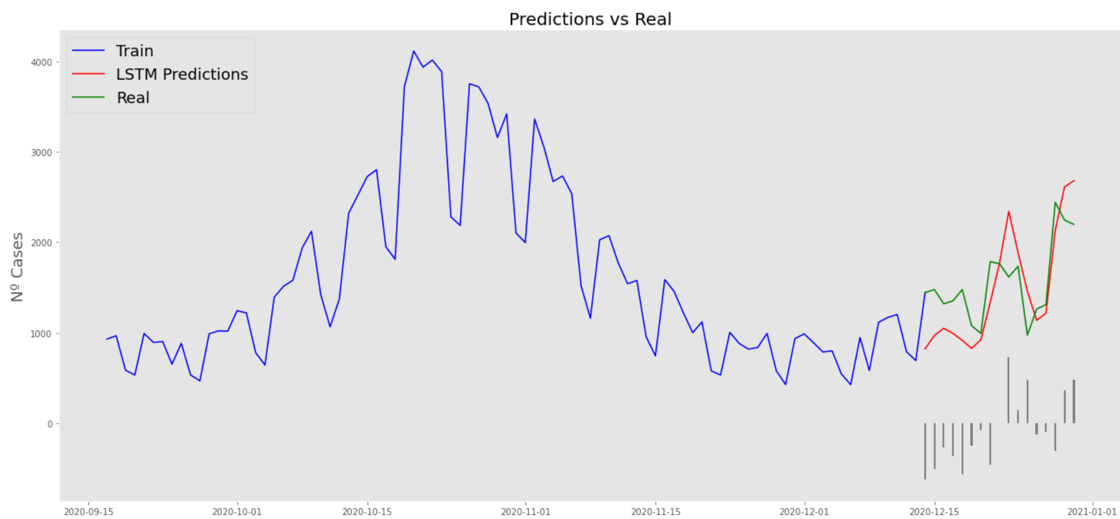
[54]:
```
# Zoom-in to a closer timeframe
valid_v2 = valid_v2[valid_v2.index > display_start_date_v2]
train_v2 = train_v2[train_v2.index > display_start_date_v2]

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

[54]:
```
            num_casos.x  Difference  Predictions  \
fecha
2020-12-14         1444 -621.604004   822.395996
```

```
2020-12-15         1477 -506.623108   970.376892
2020-12-16         1320 -270.788818  1049.211182
2020-12-17         1353 -359.287170   993.712830
2020-12-18         1477 -561.258362   915.741638
2020-12-19         1078 -250.157410   827.842590
2020-12-20          994  -71.920044   922.079956
2020-12-21         1785 -454.596313  1330.403687
2020-12-22         1763    4.741211  1767.741211
2020-12-23         1616  725.726318  2341.726318
2020-12-24         1735  149.071899  1884.071899
2020-12-25          974  485.407715  1459.407715
2020-12-26         1262 -124.141602  1137.858398
2020-12-27         1311  -95.905884  1215.094116
2020-12-28         2440 -308.902832  2131.097168
2020-12-29         2244  367.483643  2611.483643
2020-12-30         2197  482.420898  2679.420898

            residential_percent_change_from_baseline      total
fecha
2020-12-14                                       8.0  15.943333
2020-12-15                                       8.0  19.846667
2020-12-16                                       8.0  23.750000
2020-12-17                                       9.0  21.120000
2020-12-18                                      13.0  18.490000
2020-12-19                                      10.0  15.860000
2020-12-20                                       8.0  13.230000
2020-12-21                                       9.0  16.086667
2020-12-22                                      12.0  18.943333
2020-12-23                                      12.0  21.800000
2020-12-24                                      16.0  19.445000
2020-12-25                                      29.0  17.090000
2020-12-26                                      15.0  14.735000
2020-12-27                                       9.0  12.380000
2020-12-28                                      16.0  14.766667
2020-12-29                                      16.0  17.153333
2020-12-30                                      16.0  19.540000
```

[55]:
```
# Visualize the data
fig, ax1 = plt.subplots(figsize=(22, 10), sharex=True)

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

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

plt.plot(yt_v2, color="blue", linewidth=1.5)
plt.plot(yv_v2["Predictions"], color="red", linewidth=1.5)
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

[56]:
```
# New dataframe with only the 'num_casos.x' column
# Convert it to numpy array
data_v3 = Bar.filter(['num_casos.x',
                      'residential_percent_change_from_baseline',
                      'retail_and_recreation_percent_change_from_baseline',
                      'grocery_and_pharmacy_percent_change_from_baseline',
                      'parks_percent_change_from_baseline',
                      'transit_stations_percent_change_from_baseline',
                      'workplaces_percent_change_from_baseline',
                      'total'])
```

```
npdataset_v3 = data_v3.values

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

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

[56]:
```
array([[0.34230487, 0.58      , 0.15909091, 0.40769231, 0.14049587,
        0.26666667, 0.36734694, 0.41473259],
       [0.31416687, 0.64      , 0.14772727, 0.43846154, 0.14876033,
        0.22666667, 0.30612245, 0.38446014],
       [0.34132616, 0.66      , 0.13636364, 0.40769231, 0.14049587,
        0.2       , 0.28571429, 0.35418769],
       [0.32126254, 0.7       , 0.125     , 0.38461538, 0.12396694,
        0.18666667, 0.26530612, 0.33551968],
       [0.38879374, 0.76      , 0.10227273, 0.37692308, 0.10743802,
        0.16      , 0.25510204, 0.31685166]])
```

[57]:
```
# Creating a separate scaler that works on a single column for scaling␣
 ↪predictions
scaler_v3_pred = MinMaxScaler(feature_range=(0, 1))
df_cases = pd.DataFrame(Bar['num_casos.x'])
np_cases_scaled_v3 = scaler_v3_pred.fit_transform(df_cases)
np_cases_scaled_v3[0:5]
```

[57]:
```
array([[0.34230487],
       [0.31416687],
       [0.34132616],
       [0.32126254],
       [0.38879374]])
```

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

```
(273, 8)
```

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

[59]:
```
array([[0.34230487, 0.58      , 0.15909091, 0.40769231, 0.14049587,
        0.26666667, 0.36734694, 0.41473259],
       [0.31416687, 0.64      , 0.14772727, 0.43846154, 0.14876033,
```

```
         0.22666667, 0.30612245, 0.38446014]])
```

[60]:
```
training_data_length_v3
```

[60]:
```
273
```

[61]:
```
x_train_v3 = []
y_train_v3 = []
# The RNN needs data with the format of [samples, time steps, features].
# Here, we create N samples, N loop_back time steps per sample, and 8 features␣
 ↪(all)
for i in range(loop_back, training_data_length_v3):
    #print(i)
    #contains loop_back values ->>> 0-loop_back * columsn
    x_train_v3.append(train_data_v3[i-loop_back:i,:])
    #contains the prediction values for test / validation
    y_train_v3.append(train_data_v3[i, 0])

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

[61]:
```
array([[[0.34230487, 0.58      , 0.15909091, 0.40769231, 0.14049587,
         0.26666667, 0.36734694, 0.41473259],
        [0.31416687, 0.64      , 0.14772727, 0.43846154, 0.14876033,
         0.22666667, 0.30612245, 0.38446014],
        [0.34132616, 0.66      , 0.13636364, 0.40769231, 0.14049587,
         0.2       , 0.28571429, 0.35418769],
        [0.32126254, 0.7       , 0.125     , 0.38461538, 0.12396694,
         0.18666667, 0.26530612, 0.33551968],
        [0.38879374, 0.76      , 0.10227273, 0.37692308, 0.10743802,
         0.16      , 0.25510204, 0.31685166],
        [0.27868852, 0.58      , 0.05681818, 0.28461538, 0.04958678,
         0.10666667, 0.2755102 , 0.16271443],
        [0.25446538, 0.48      , 0.02272727, 0.11538462, 0.02479339,
         0.05333333, 0.25510204, 0.00857719],
        [0.295816  , 0.7       , 0.10227273, 0.32307692, 0.08264463,
         0.12      , 0.2244898 , 0.15842583],
        [0.27208221, 0.72      , 0.10227273, 0.33846154, 0.08264463,
         0.12      , 0.20408163, 0.30827447],
        [0.27746513, 0.74      , 0.09090909, 0.32307692, 0.08264463,
         0.12      , 0.20408163, 0.3037336 ],
        [0.27477367, 0.78      , 0.09090909, 0.32307692, 0.08264463,
         0.10666667, 0.19387755, 0.29919273],
        [0.24761439, 0.8       , 0.09090909, 0.32307692, 0.08264463,
         0.10666667, 0.19387755, 0.21039354],
        [0.25935894, 0.6       , 0.04545455, 0.25384615, 0.04132231,
```

```
 0.08      , 0.25510204, 0.12159435],
[0.19133839, 0.5       , 0.02272727, 0.1       , 0.02479339,
 0.04      , 0.2755102 , 0.15817356]],

[[0.31416687, 0.64      , 0.14772727, 0.43846154, 0.14876033,
  0.22666667, 0.30612245, 0.38446014],
 [0.34132616, 0.66      , 0.13636364, 0.40769231, 0.14049587,
  0.2       , 0.28571429, 0.35418769],
 [0.32126254, 0.7       , 0.125     , 0.38461538, 0.12396694,
  0.18666667, 0.26530612, 0.33551968],
 [0.38879374, 0.76      , 0.10227273, 0.37692308, 0.10743802,
  0.16      , 0.25510204, 0.31685166],
 [0.27868852, 0.58      , 0.05681818, 0.28461538, 0.04958678,
  0.10666667, 0.2755102 , 0.16271443],
 [0.25446538, 0.48      , 0.02272727, 0.11538462, 0.02479339,
  0.05333333, 0.25510204, 0.00857719],
 [0.295816  , 0.7       , 0.10227273, 0.32307692, 0.08264463,
  0.12      , 0.2244898 , 0.15842583],
 [0.27208221, 0.72      , 0.10227273, 0.33846154, 0.08264463,
  0.12      , 0.20408163, 0.30827447],
 [0.27746513, 0.74      , 0.09090909, 0.32307692, 0.08264463,
  0.12      , 0.20408163, 0.3037336 ],
 [0.27477367, 0.78      , 0.09090909, 0.32307692, 0.08264463,
  0.10666667, 0.19387755, 0.29919273],
 [0.24761439, 0.8       , 0.09090909, 0.32307692, 0.08264463,
  0.10666667, 0.19387755, 0.21039354],
 [0.25935894, 0.6       , 0.04545455, 0.25384615, 0.04132231,
  0.08      , 0.25510204, 0.12159435],
 [0.19133839, 0.5       , 0.02272727, 0.1       , 0.02479339,
  0.04      , 0.2755102 , 0.15817356],
 [0.23562515, 0.78      , 0.06818182, 0.25384615, 0.05785124,
  0.08      , 0.13265306, 0.19475277]]])
```

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

[62]:
```
array([0.23562515, 0.22143381])
```

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

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

[64]:
```
# Create the test data
test_data_v3 = scaled_data_v3[training_data_length_v3 - loop_back:, :]
print(test_data_v3.shape)

x_test_v3 = []
y_test_v3 = []
```

```python
# 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)
```

[64]:
```
array([[[0.22265721, 0.22      , 0.81818182, 0.69230769, 0.66115702,
         0.89333333, 0.67346939, 0.47544568],
        [0.23562515, 0.24      , 0.78409091, 0.7       , 0.59504132,
         0.88      , 0.67346939, 0.6749075 ],
        [0.21164668, 0.24      , 0.78409091, 0.71538462, 0.61157025,
         0.86666667, 0.69387755, 0.87436932],
        [0.1866895 , 0.24      , 0.79545455, 0.72307692, 0.60330579,
         0.89333333, 0.68367347, 0.72262866],
        [0.18962564, 0.32      , 0.69318182, 0.7       , 0.52066116,
         0.78666667, 0.67346939, 0.57088799],
        [0.12698801, 0.32      , 0.625     , 0.66153846, 0.49586777,
         0.73333333, 0.75510204, 0.41914733],
        [0.09836066, 0.28      , 0.56818182, 0.63076923, 0.56198347,
         0.69333333, 0.7244898 , 0.26740666],
        [0.22559334, 0.32      , 1.        , 0.82307692, 0.73553719,
         0.84      , 0.35714286, 0.45021863],
        [0.13677514, 0.54      , 0.64772727, 0.34615385, 0.6446281 ,
         0.54666667, 0.12244898, 0.63303061],
        [0.26694397, 0.24      , 0.78409091, 0.74615385, 0.55371901,
         0.85333333, 0.69387755, 0.81584258],
        [0.28015659, 0.24      , 0.79545455, 0.73846154, 0.58677686,
         0.88      , 0.68367347, 0.68276993],
        [0.2879863 , 0.32      , 0.69318182, 0.69230769, 0.49586777,
         0.77333333, 0.67346939, 0.54969728],
        [0.18742354, 0.3       , 0.64772727, 0.67692308, 0.55371901,
         0.78666667, 0.78571429, 0.41662462],
        [0.16344507, 0.26      , 0.63636364, 0.71538462, 0.60330579,
         0.74666667, 0.79591837, 0.28355197]],

       [[0.23562515, 0.24      , 0.78409091, 0.7       , 0.59504132,
```

```
        0.88      , 0.67346939, 0.6749075 ],
       [0.21164668, 0.24      , 0.78409091, 0.71538462, 0.61157025,
        0.86666667, 0.69387755, 0.87436932],
       [0.1866895 , 0.24      , 0.79545455, 0.72307692, 0.60330579,
        0.89333333, 0.68367347, 0.72262866],
       [0.18962564, 0.32      , 0.69318182, 0.7       , 0.52066116,
        0.78666667, 0.67346939, 0.57088799],
       [0.12698801, 0.32      , 0.625     , 0.66153846, 0.49586777,
        0.73333333, 0.75510204, 0.41914733],
       [0.09836066, 0.28      , 0.56818182, 0.63076923, 0.56198347,
        0.69333333, 0.7244898 , 0.26740666],
       [0.22559334, 0.32      , 1.        , 0.82307692, 0.73553719,
        0.84      , 0.35714286, 0.45021863],
       [0.13677514, 0.54      , 0.64772727, 0.34615385, 0.6446281 ,
        0.54666667, 0.12244898, 0.63303061],
       [0.26694397, 0.24      , 0.78409091, 0.74615385, 0.55371901,
        0.85333333, 0.69387755, 0.81584258],
       [0.28015659, 0.24      , 0.79545455, 0.73846154, 0.58677686,
        0.88      , 0.68367347, 0.68276993],
       [0.2879863 , 0.32      , 0.69318182, 0.69230769, 0.49586777,
        0.77333333, 0.67346939, 0.54969728],
       [0.18742354, 0.3       , 0.64772727, 0.67692308, 0.55371901,
        0.78666667, 0.78571429, 0.41662462],
       [0.16344507, 0.26      , 0.63636364, 0.71538462, 0.60330579,
        0.74666667, 0.79591837, 0.28355197],
       [0.34719843, 0.22      , 0.86363636, 0.68461538, 0.61157025,
        0.88      , 0.68367347, 0.48049109]]])
```

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

[65]:
```
array([0.34719843, 0.35527282])
```

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

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

[67]:
```
# Configure the neural network model
model_v3 = Sequential()

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

model_v3.add(LSTM(n_neurons_v3,
                  activation='relu',
                  return_sequences=True,
```

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

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

```
112 14 8
```

[68]:
```
# Training the model
early_stop_v3 = EarlyStopping(monitor='loss', patience=2, verbose=1)
history_v3 = model_v3.fit(x_train_v3,
                    y_train_v3,
                    batch_size=2,
                    validation_data=(x_test_v3, y_test_v3),
                    epochs=50
                    #callbacks=[early_stop_v2]
                         )
```

```
Epoch 1/50
130/130 [==============================] - 10s 26ms/step - loss: 0.0521 -
val_loss: 0.0092
Epoch 2/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0306 -
val_loss: 0.0113
Epoch 3/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0133 -
val_loss: 0.0138
Epoch 4/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0100 -
val_loss: 0.0181
Epoch 5/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0074 -
val_loss: 0.0188
Epoch 6/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0101 -
val_loss: 0.0177
Epoch 7/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0109 -
val_loss: 0.0105
Epoch 8/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0068 -
val_loss: 0.0107
Epoch 9/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0059 -
```

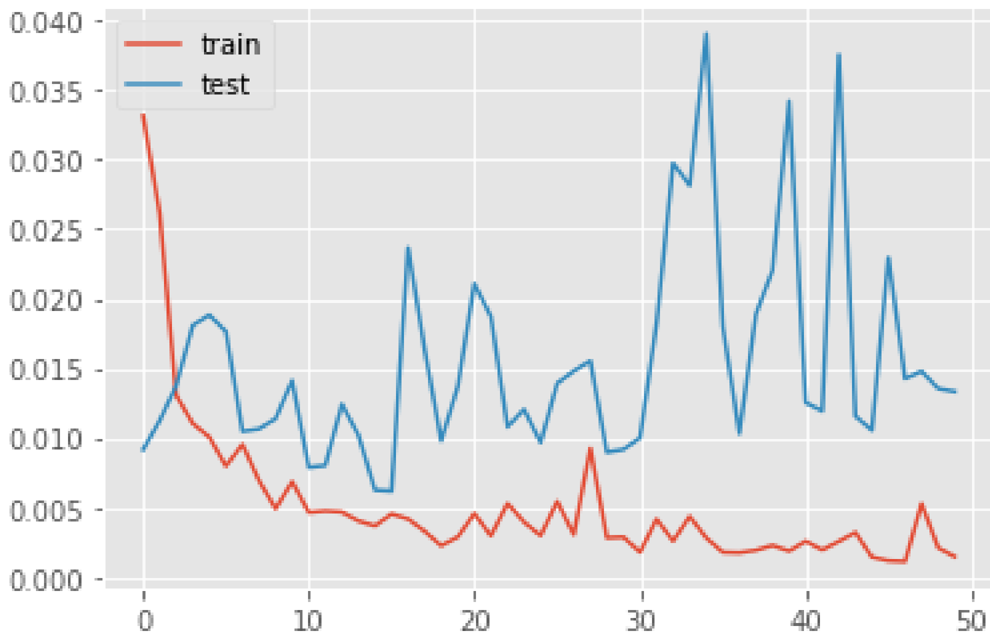
```
val_loss: 0.0114
Epoch 10/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0100 -
val_loss: 0.0141
Epoch 11/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0057 -
val_loss: 0.0079
Epoch 12/50
130/130 [==============================] - 3s 20ms/step - loss: 0.0059 -
val_loss: 0.0080
Epoch 13/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0044 -
val_loss: 0.0124
Epoch 14/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0041 -
val_loss: 0.0102
Epoch 15/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0035 -
val_loss: 0.0063
Epoch 16/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0026 -
val_loss: 0.0062
Epoch 17/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0041 -
val_loss: 0.0237
Epoch 18/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0030 -
val_loss: 0.0163
Epoch 19/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0030 -
val_loss: 0.0098
Epoch 20/50
130/130 [==============================] - 3s 21ms/step - loss: 0.0028 -
val_loss: 0.0137
Epoch 21/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0061 -
val_loss: 0.0211
Epoch 22/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0032 -
val_loss: 0.0187
Epoch 23/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0045 -
val_loss: 0.0108
Epoch 24/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0036 -
val_loss: 0.0121
Epoch 25/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0033 -
```

```
val_loss: 0.0097
Epoch 26/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0032 -
val_loss: 0.0139
Epoch 27/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0023 -
val_loss: 0.0148
Epoch 28/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0099 -
val_loss: 0.0155
Epoch 29/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0040 -
val_loss: 0.0090
Epoch 30/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0040 -
val_loss: 0.0092
Epoch 31/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0013 -
val_loss: 0.0100
Epoch 32/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0037 -
val_loss: 0.0182
Epoch 33/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0023 -
val_loss: 0.0298
Epoch 34/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0049 -
val_loss: 0.0281
Epoch 35/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0029 -
val_loss: 0.0390
Epoch 36/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0027 -
val_loss: 0.0180
Epoch 37/50
130/130 [==============================] - 2s 18ms/step - loss: 0.0013 -
val_loss: 0.0103
Epoch 38/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0018 -
val_loss: 0.0189
Epoch 39/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0020 -
val_loss: 0.0220
Epoch 40/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0017 -
val_loss: 0.0342
Epoch 41/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0026 -
```

```
val_loss: 0.0125
Epoch 42/50
130/130 [==============================] - 2s 16ms/step - loss: 0.0021 -
val_loss: 0.0119
Epoch 43/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0013 -
val_loss: 0.0375
Epoch 44/50
130/130 [==============================] - 2s 15ms/step - loss: 0.0047 -
val_loss: 0.0116
Epoch 45/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0016 -
val_loss: 0.0106
Epoch 46/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0011 -
val_loss: 0.0230
Epoch 47/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0013 -
val_loss: 0.0143
Epoch 48/50
130/130 [==============================] - 2s 17ms/step - loss: 0.0036 -
val_loss: 0.0148
Epoch 49/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0039 -
val_loss: 0.0135
Epoch 50/50
130/130 [==============================] - 2s 14ms/step - loss: 0.0014 -
val_loss: 0.0133
```

[69]:

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



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

[70]:
```
array([[0.19576609],
       [0.22180428],
       [0.24445327],
       [0.24590875],
       [0.22805966],
       [0.18211347],
       [0.1660583 ],
       [0.41708505],
       [0.47737867],
       [0.5845706 ],
       [0.53055453],
       [0.4080822 ],
       [0.2156617 ],
       [0.19537744],
       [0.39036286],
       [0.54433674],
       [0.5608298 ]], dtype=float32)
```

[71]:
```
# Get the predicted values
pred_unscaled_v3 = scaler_v3_pred.inverse_transform(predictions_v3)
y_test_v3_unscaled = scaler_v3_pred.inverse_transform(y_test_v3.reshape(-1, 1))
```

[72]:
```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: 407.9
RMSE: 471.9
```

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

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

[75]:
```python
# Zoom-in to a closer timeframe
valid_v3 = valid_v3[valid_v3.index > display_start_date_v3]
train_v3 = train_v3[train_v3.index > display_start_date_v3]

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

[75]:
```
            num_casos.x  Difference  Predictions  \
fecha
2020-12-14         1444 -618.903992   825.096008
2020-12-15         1477 -545.485901   931.514099
2020-12-16         1320 -295.919556  1024.080444
2020-12-17         1353 -322.970947  1030.029053
2020-12-18         1477 -519.920166   957.079834
2020-12-19         1078 -308.702271   769.297729
2020-12-20          994 -290.319702   703.680298
2020-12-21         1785  -55.373413  1729.626587
2020-12-22         1763  213.046631  1976.046631
2020-12-23         1616  798.139893  2414.139893
2020-12-24         1735  458.376221  2193.376221
2020-12-25          974  718.831909  1692.831909
2020-12-26         1262 -355.590637   906.409363
2020-12-27         1311 -487.492432   823.507568
2020-12-28         2440 -819.587036  1620.412964
2020-12-29         2244    5.704102  2249.704102
```

```
2020-12-30         2197  120.111328  2317.111328

            residential_percent_change_from_baseline  \
fecha
2020-12-14                                       8.0
2020-12-15                                       8.0
2020-12-16                                       8.0
2020-12-17                                       9.0
2020-12-18                                      13.0
2020-12-19                                      10.0
2020-12-20                                       8.0
2020-12-21                                       9.0
2020-12-22                                      12.0
2020-12-23                                      12.0
2020-12-24                                      16.0
2020-12-25                                      29.0
2020-12-26                                      15.0
2020-12-27                                       9.0
2020-12-28                                      16.0
2020-12-29                                      16.0
2020-12-30                                      16.0

            retail_and_recreation_percent_change_from_baseline  \
fecha
2020-12-14                                              -21.0
2020-12-15                                              -23.0
2020-12-16                                              -23.0
2020-12-17                                              -24.0
2020-12-18                                              -35.0
2020-12-19                                              -31.0
2020-12-20                                              -28.0
2020-12-21                                              -20.0
2020-12-22                                              -23.0
2020-12-23                                              -21.0
2020-12-24                                              -30.0
2020-12-25                                              -82.0
2020-12-26                                              -75.0
2020-12-27                                              -43.0
2020-12-28                                              -25.0
2020-12-29                                              -26.0
2020-12-30                                              -25.0

            grocery_and_pharmacy_percent_change_from_baseline  \
fecha
2020-12-14                                               -2.0
2020-12-15                                                1.0
2020-12-16                                                3.0
```

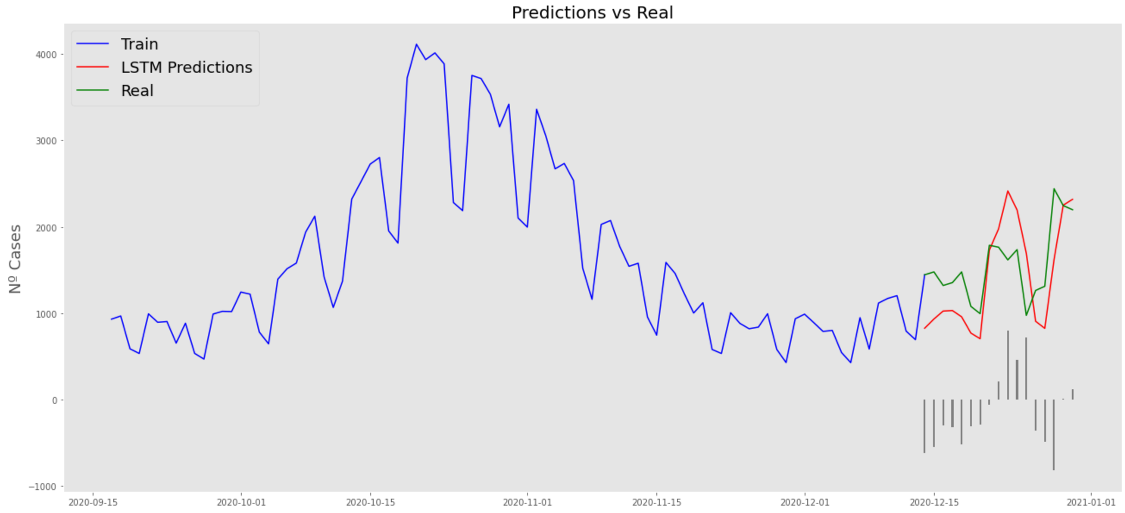
```
2020-12-17                                                    4.0
2020-12-18                                                   -7.0
2020-12-19                                                    3.0
2020-12-20                                                   21.0
2020-12-21                                                   14.0
2020-12-22                                                   19.0
2020-12-23                                                   39.0
2020-12-24                                                   34.0
2020-12-25                                                  -79.0
2020-12-26                                                  -69.0
2020-12-27                                                   19.0
2020-12-28                                                   11.0
2020-12-29                                                   15.0
2020-12-30                                                   31.0

            parks_percent_change_from_baseline  \
fecha
2020-12-14                               -19.0
2020-12-15                               -18.0
2020-12-16                               -17.0
2020-12-17                               -23.0
2020-12-18                               -41.0
2020-12-19                               -24.0
2020-12-20                               -19.0
2020-12-21                                -6.0
2020-12-22                               -10.0
2020-12-23                               -13.0
2020-12-24                               -28.0
2020-12-25                               -47.0
2020-12-26                               -35.0
2020-12-27                               -33.0
2020-12-28                               -25.0
2020-12-29                               -17.0
2020-12-30                               -19.0

            transit_stations_percent_change_from_baseline  \
fecha
2020-12-14                                          -25.0
2020-12-15                                          -23.0
2020-12-16                                          -24.0
2020-12-17                                          -24.0
2020-12-18                                          -32.0
2020-12-19                                          -25.0
2020-12-20                                          -27.0
2020-12-21                                          -21.0
2020-12-22                                          -25.0
2020-12-23                                          -25.0
```

```
2020-12-24                                          -37.0
2020-12-25                                          -67.0
2020-12-26                                          -50.0
2020-12-27                                          -38.0
2020-12-28                                          -37.0
2020-12-29                                          -34.0
2020-12-30                                          -35.0

            workplaces_percent_change_from_baseline      total
fecha
2020-12-14                                    -25.0  15.943333
2020-12-15                                    -25.0  19.846667
2020-12-16                                    -23.0  23.750000
2020-12-17                                    -24.0  21.120000
2020-12-18                                    -26.0  18.490000
2020-12-19                                    -12.0  15.860000
2020-12-20                                     -8.0  13.230000
2020-12-21                                    -30.0  16.086667
2020-12-22                                    -40.0  18.943333
2020-12-23                                    -42.0  21.800000
2020-12-24                                    -56.0  19.445000
2020-12-25                                    -87.0  17.090000
2020-12-26                                    -52.0  14.735000
2020-12-27                                    -19.0  12.380000
2020-12-28                                    -53.0  14.766667
2020-12-29                                    -53.0  17.153333
2020-12-30                                    -52.0  19.540000
```

[76]:

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

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

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

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

```python
# 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()
```



[ ]: