Assignment_P2

May 29, 2019

1 194.049 Energy-efficient Distributed Systems

1.1 Assignment Part 2: Simulation infrastructure and preliminary implementation

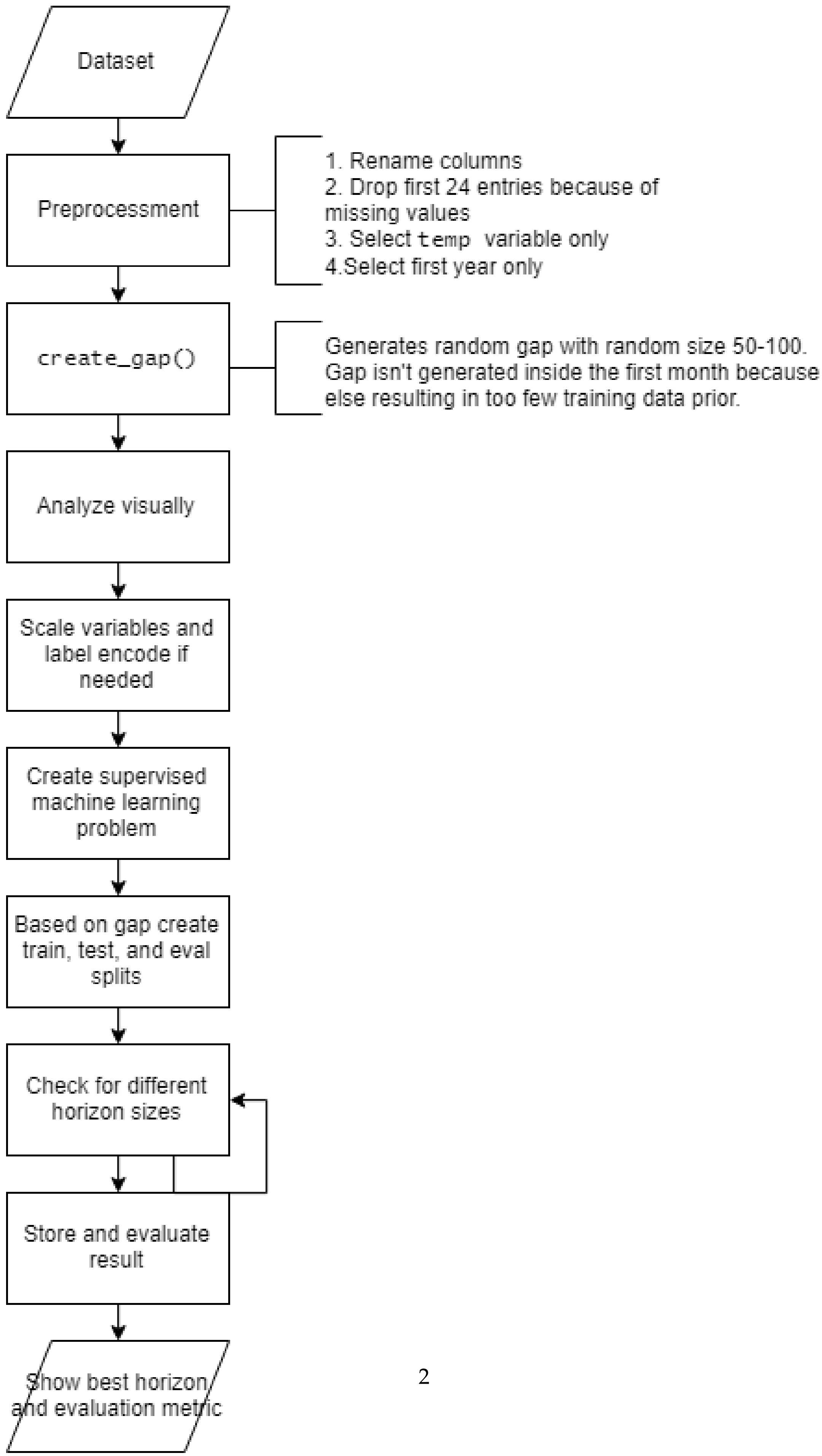
1.1.1 Gent Rexha (11832486), Princ Mullatahiri (11846033), Ilir Osmanaj (11770999)

29.05.2019

1.2 Infrastructure Simulation Flow Chart

```
In [32]: from IPython.display import Image
         from IPython.core.display import HTML

         PATH = "C:/Projects/University/Semester 2 Projects/Energy-efficient Distributed Syster
         Image(filename = PATH + "Infrastructure Simulation Flow Chart.png")
```

Out[32]:


Dataset
Preprocessment
1. Rename columns
2. Drop first 24 entries because of missing values
3. Select temp variable only
4.Select first year only
create_gap()
Generates random gap with random size 50-100. Gap isn't generated inside the first month because else resulting in too few training data prior.
Analyze visually
Scale variables and label encode if needed
Create supervised machine learning problem
Based on gap create train, test, and eval splits
Check for different horizon sizes
Store and evaluate result
Show best horizon and evaluation metric

In [17]:
```
from datetime import datetime
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
import pandas as pd
from pathlib import Path
from keras import Sequential
from keras.layers import LSTM, Dense
from matplotlib import pyplot
import numpy as np
from math import sqrt
from sklearn.metrics import mean_squared_error
from plotnine import *
```

1.3 Preprocessment

In [18]:
```
import platform
if platform.system() == 'Darwin':
    data_path = Path('../data')
else:
    data_path = Path('C:/Projects/University/Semester 2 Projects/Energy-efficient Dis
```

1.3.1 Data Preparation

In [19]:
```
def parse(x):
    return datetime.strptime(x, '%Y %m %d %H')

df = pd.read_csv(data_path / 'poll.csv',  parse_dates = [['year', 'month', 'day', 'hou
df.drop('No', axis=1, inplace=True)

# Manually specify column names
df.columns = ['pollution', 'dew', 'temp', 'press', 'wnd_dir', 'wnd_spd', 'snow', 'rai
df.index.name = 'date'

# Mark all NA values with 0
df['pollution'].fillna(0, inplace=True)

# Drop the first 24 hours because all of them have missing values
df = df[24:]

# Summarize first 5 rows
display(df.head())

# Save to file
df.to_csv(data_path / 'pollution.csv')
```

```
            pollution  dew  temp   press wnd_dir  wnd_spd  snow  rain
```

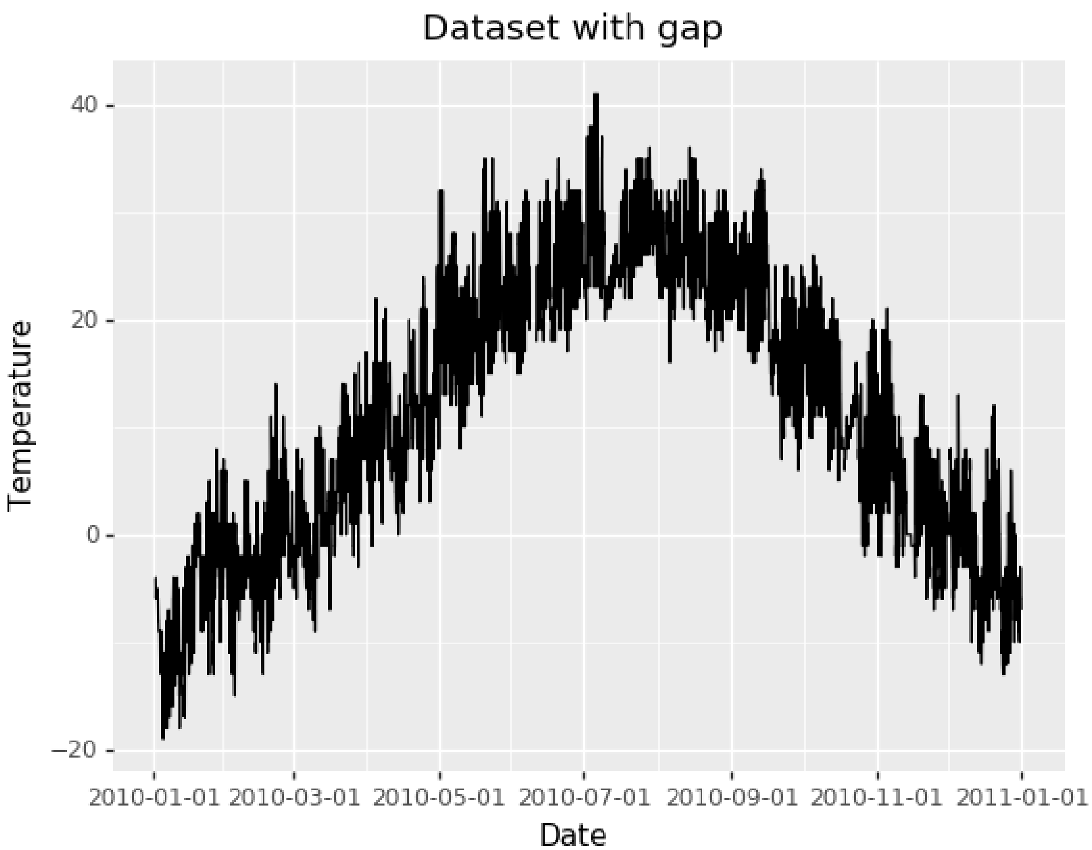
```
date
2010-01-02 00:00:00      129.0  -16  -4.0  1020.0      SE     1.79     0     0
2010-01-02 01:00:00      148.0  -15  -4.0  1020.0      SE     2.68     0     0
2010-01-02 02:00:00      159.0  -11  -5.0  1021.0      SE     3.57     0     0
2010-01-02 03:00:00      181.0   -7  -5.0  1022.0      SE     5.36     1     0
2010-01-02 04:00:00      138.0   -7  -5.0  1022.0      SE     6.25     2     0
```

```
In [20]: import random
         from copy import deepcopy

         # Select only temp and first year for forecast
         df_pred = df.loc[:, ['temp']]
         df_pred = df_pred.iloc[0:24*7*52]


         def create_gap(df):
             """Creates a artificial made gap in the first column of the dataset with a size f

             Args:
                 df (pd.Dataframe): dataframe where the gap should be created

             Output:
                 dataset (pd.Dataframe): dataframe with random gap
             """
             dataset = deepcopy(df)
             # Leave one month before and at the end so we have enough data to train the model
             gap = random.randint(24*7*4, len(df.index)-24*7*4)
             gap_size = random.randint(50,101)
             dataset.iloc[gap:gap+gap_size] = np.nan
             return dataset


         df_gap = create_gap(df_pred)
```

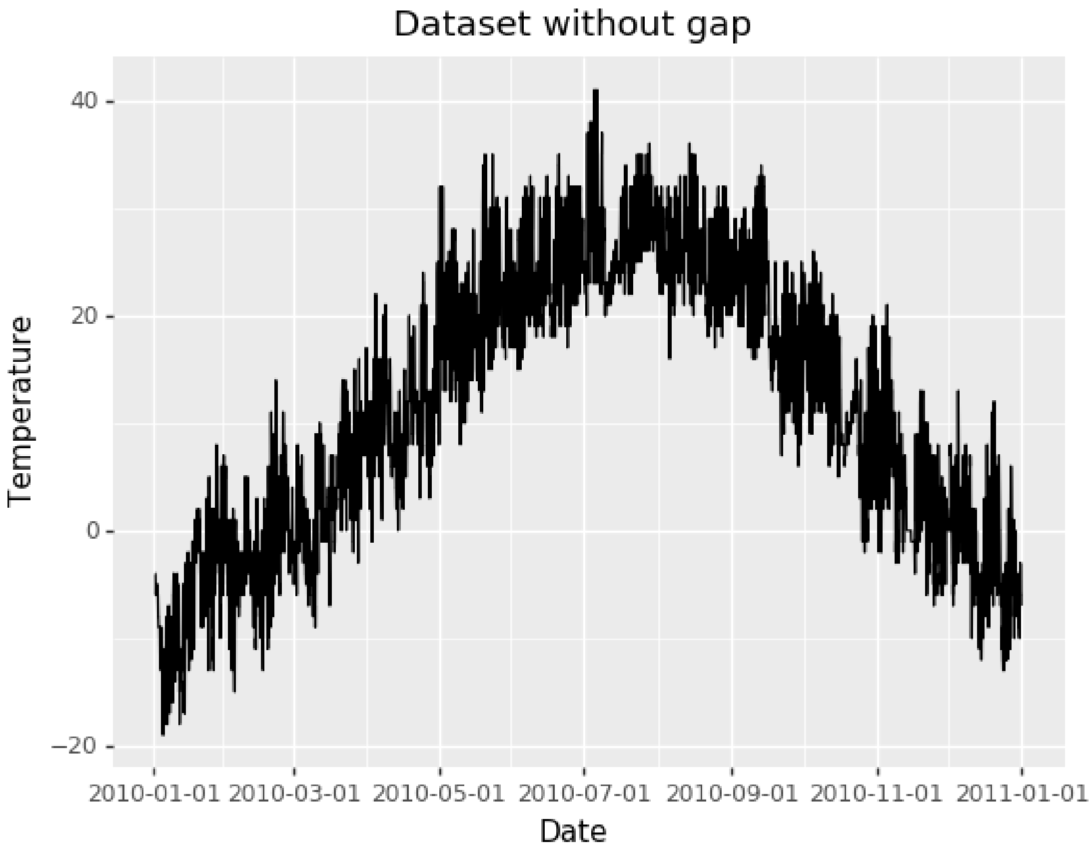
1.3.2 Visualizations

```
In [21]: # df_gap
         g1 = ggplot(df_gap, aes('df_gap.index', 'temp')) + geom_line() + labs(x='Date', y='Te
         print(g1)

         # df_pred
         g2 = ggplot(df_pred, aes('df_pred.index', 'temp')) + geom_line() + labs(x='Date', y='
         print(g2)

         # Find gap
         nan_indexes = pd.isnull(df_gap).any(1).nonzero()[0].tolist()
```

```
# Zoomed in version of df_gap
dataset = df_gap.iloc[nan_indexes[0]-24*7:nan_indexes[-1]+24*7]
g3 = ggplot(dataset, aes('dataset.index', 'temp')) + geom_line() + labs(x='Date', y='T
print(g3)

# Zoomed in version of df_pred
dataset = df_pred.iloc[nan_indexes[0]-24*7:nan_indexes[-1]+24*7]
g4 = ggplot(dataset, aes('dataset.index', 'temp')) + geom_line() + labs(x='Date', y='T
print(g4)
```

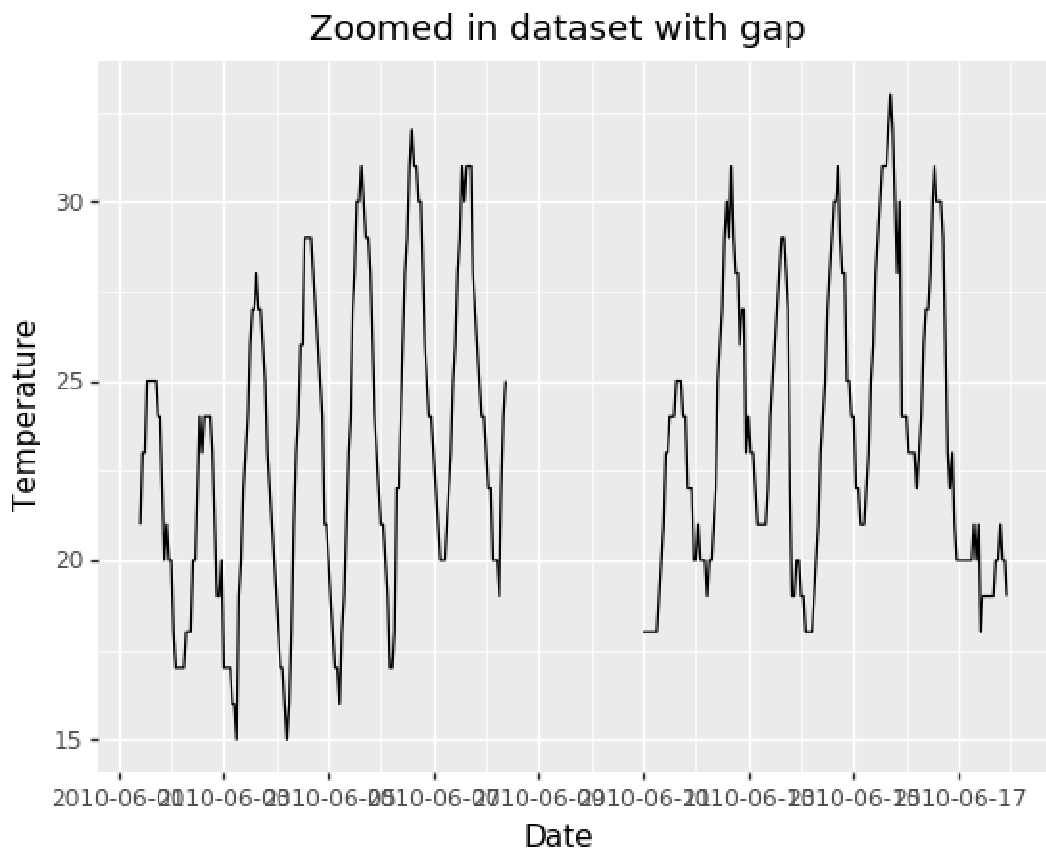



```
<ggplot: (-9223371902415579234)>
```

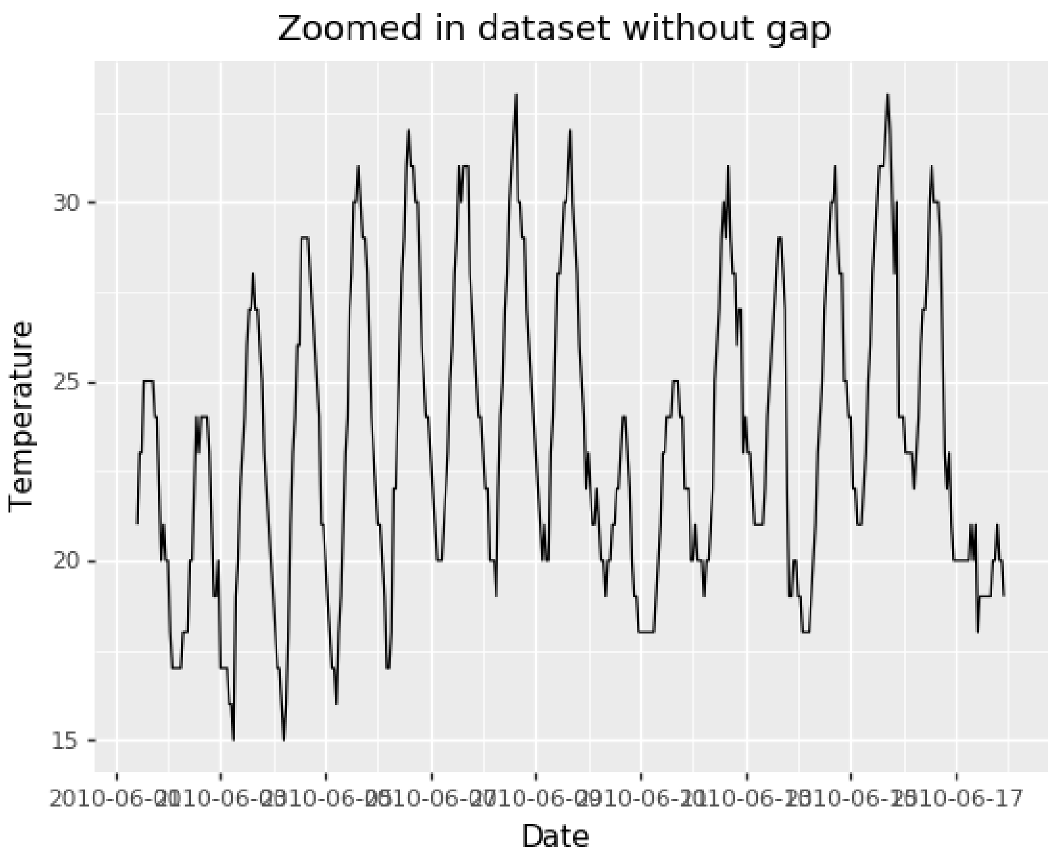



```
<ggplot: (-9223371902415236201)>
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:10: FutureWarning: Series.nonz
  # Remove the CWD from sys.path while we load stuff.
```




```
<ggplot: (134431158059)>
```




```
<ggplot: (134439583794)>
```

1.3.3 Detect gap indexes

This assumes that there is only one continuous gap in the dataset

```
In [22]: # get the indexes with nan values on the temp field
         nan_indexes = pd.isnull(df_gap).any(1).nonzero()[0].tolist()
         gap_idx_start = nan_indexes[0]
         gap_length = len(nan_indexes)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: Series.nonze
```

```
In [23]: def perfrom_scaling(values):
             scaler = MinMaxScaler(feature_range=(0, 1))
             return scaler.fit_transform(values)
```

1.3.4 Label Enconding & Scaling

```
In [24]: # Load dataset
         values = df_gap.values
         values_original = df_pred.values

         # Normalize features
         scaler = MinMaxScaler(feature_range=(0, 1))
         values = scaler.fit_transform(values)
         values_original = scaler.fit_transform(values_original)

         # Encoded & scaled values
         display(values)
```

```
array([[0.25      ],
       [0.25      ],
       [0.23333333],
       ...,
       [0.21666667],
       [0.2       ],
       [0.2       ]])
```

1.3.5 Transforming Time Series Data into a Supervised Machine Learning Problem

```
In [25]: def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
             """Frame a time series as a supervised learning dataset.

             Args:
                 data: Sequence of observations as a list or NumPy array.
                 n_in: Number of lag observations as input (X).
                 n_out: Number of observations as output (y).
                 dropnan: Boolean whether or not to drop rows with NaN values.
             Returns:
                 pd.Dataframe: The return value. True for success, False otherwise.

             """
             n_vars = 1 if type(data) is list else data.shape[1]
             df = deepcopy(pd.DataFrame(data))
             cols = list()
             names = list()

             # input sequence (t-n, ... t-1)
             for i in range(n_in, 0, -1):
                 cols.append(df.shift(i))
                 names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]

             # forecast sequence (t, t+1, ... t+n)
             for i in range(0, n_out):
```

```
            cols.append(df.shift(-i))
            if i == 0:
                names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
            else:
                names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]

        # put it all together
        agg = pd.concat(cols, axis=1)
        agg.columns = names

        return agg
```

In [26]:
```
# frame as supervised learning
reframed = series_to_supervised(values, 1, 1)
reframed_original = series_to_supervised(values_original, 1, 1)
print(reframed.head())
```

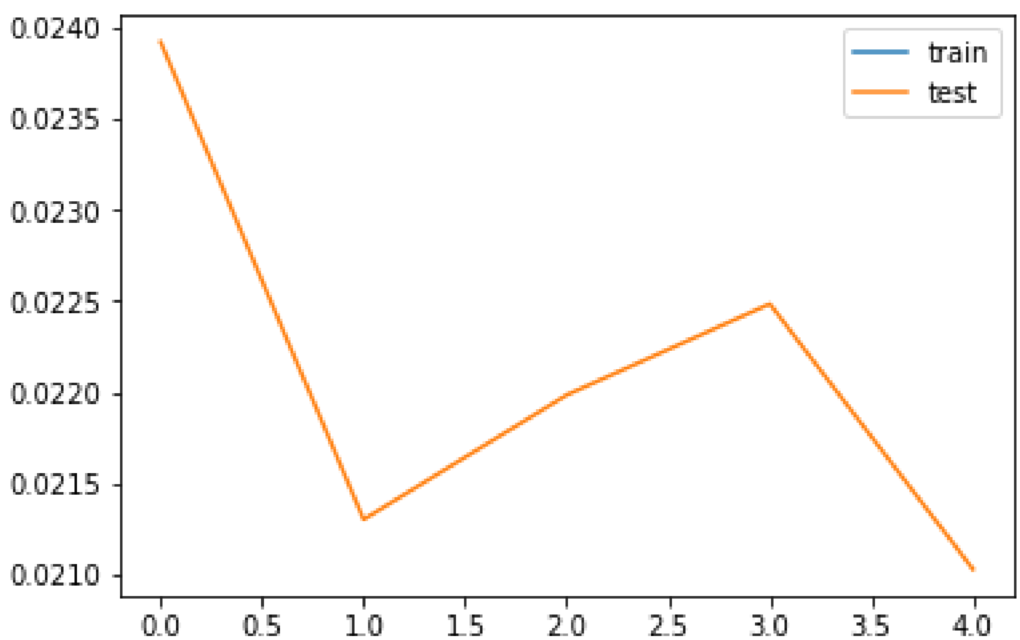
```
   var1(t-1)   var1(t)
0        NaN  0.250000
1   0.250000  0.250000
2   0.250000  0.233333
3   0.233333  0.233333
4   0.233333  0.233333
```

In [27]:
```
values = reframed.values
values_original = reframed_original.values

# from all the non-na data, use some for train and some for testing (100 values for n
all_data = np.concatenate([values[1:gap_idx_start + 1, :], values[gap_idx_start + gap

train = all_data[list(range(0, 100)) + list(range(200, all_data.shape[0] - 1)),:]
# train = train[~np.isnan(train), :]
test = all_data[list(range(100, 200)),:]

# test contains only data from the gap (but we use the orignal dataset - since they a
validation = values_original[gap_idx_start: gap_idx_start + gap_length, :]


# split into input and outputs
train_X, train_y = train[:, :-1], train[:, -1]
test_X, test_y = test[:, :-1], test[:, -1]
validation_X, validation_y = validation[:, :-1], validation[:, -1]

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], 1, train_X.shape[1]))
test_X = test_X.reshape((test_X.shape[0], 1, test_X.shape[1]))
validation_X = validation_X.reshape((validation_X.shape[0], 1, validation_X.shape[1]))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

```
(8571, 1, 1) (8571,) (100, 1, 1) (100,)
```

```
In [28]: model = Sequential()
         model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
         model.add(Dense(1))
         model.compile(loss='mae', optimizer='adam')

         # fit network
         history = model.fit(train_X, train_y, epochs=5, batch_size=1, validation_data=(test_X

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

```
Train on 8571 samples, validate on 100 samples
Epoch 1/5
 - 11s - loss: nan - val_loss: 0.0239
Epoch 2/5
 - 10s - loss: nan - val_loss: 0.0213
Epoch 3/5
 - 9s - loss: nan - val_loss: 0.0220
Epoch 4/5
 - 9s - loss: nan - val_loss: 0.0225
Epoch 5/5
 - 10s - loss: nan - val_loss: 0.0210
```

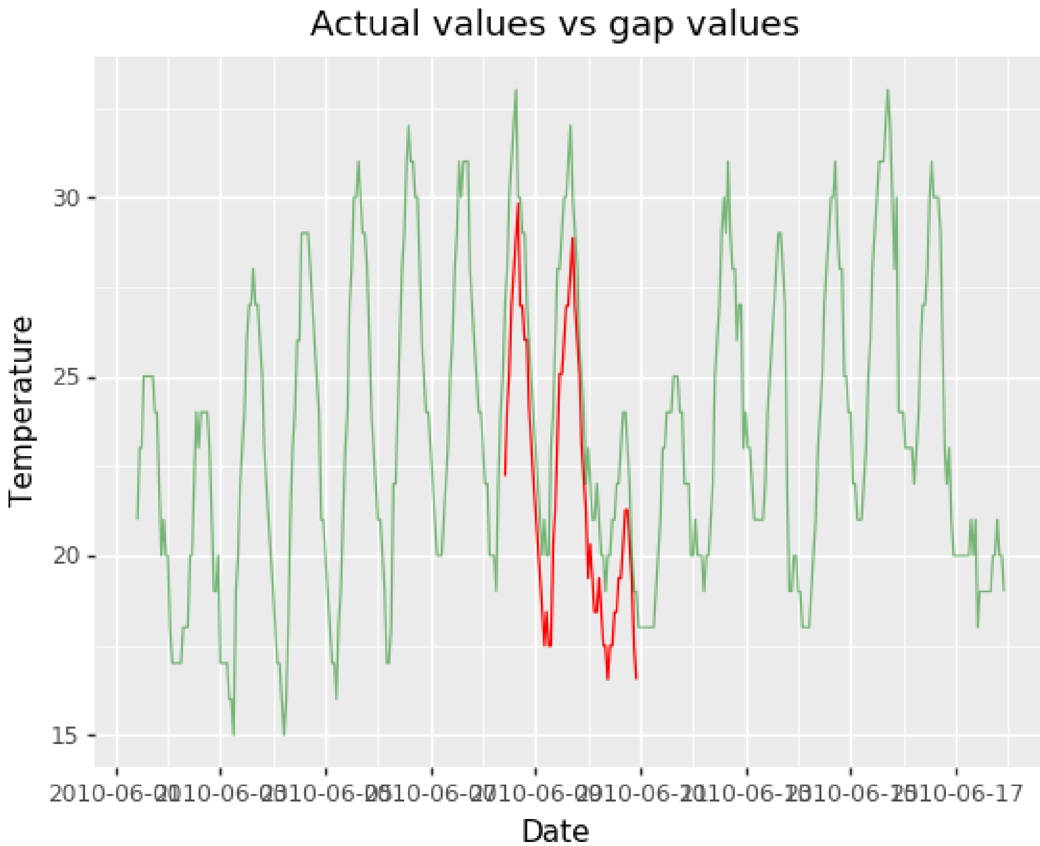


In [29]:
```
# make a prediction
yhat = model.predict(validation_X)
validation_X = validation_X.reshape((validation_X.shape[0], validation_X.shape[2]))

# invert scaling for forecast
inv_yhat = np.concatenate((yhat, validation_X[:, 1:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:,0]

# invert scaling for actual
validation_y = validation_y.reshape((len(validation_y), 1))
inv_y = np.concatenate((validation_y, validation_X[:, 1:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:,0]

# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Gap RMSE: %.3f' % rmse)
```

```
Gap RMSE: 2.910
```

1.3.6 Visualizing predicted vs actual values

In [30]:
```
# Insert predicted values into df_gap
df_gap.iloc[nan_indexes[0]:nan_indexes[-1]+1, 0] = inv_yhat
df1 = df_gap.iloc[nan_indexes[0]:nan_indexes[-1]]
df2 = df_pred.iloc[nan_indexes[0]-24*7:nan_indexes[-1]+24*7]

g1 = ggplot() + geom_line(df1, aes('df1.index', 'temp'), color='red') + geom_line(df2
print(g1)
```




```
<ggplot: (-9223371902412265570)>
```