Bertazioli Dario
Borroni Alessandro
D'Intinosante Fabrizio
Perletti Massimiliano

DATA SCIENCE LAB
// a.a. 2018/2019


VERVIEW

CONTEXT

GME Gestore Mercati Energetici

Operates in **power**, **gas** and **environmental** markets.
It is the exchange place for electricity and natural gas spot trading in Italy

In the power market platform, *producers* and *purchasers* sell and buy wholesale electricity

An auction for every hour of the day

GME releases data in files .xml with a delay of one week

Forecasting this Supply Function could be interesting for every energy producer

OUR GOAL

Predict the «**supply function**» for auctions in the Italian electricity market exploiting **functional time series forecasting**, through both statistical methods and machine learning

1. The supply function is a "curve" that relates the amount of electricity (in MWh) with the price (in €)

2. The results might provide the electricity producers with an improvement for their bidding strategy

RAW DATASET

- PURPOSE_CD
- STATUS_CD
- UNIT_REFERENCE_NO
- INTERVAL_NO
- BID_OFFER_DATE_DT
- QUANTITY_NO
- AWARDED_QUANTITY_NO
- ENERGY_PRICE_NO
- MERIT_ORDER_NO
- PARTIAL_QTY_ACCEPTED_IN
- ADJ_QUANTITY_NO
- ZONE_CD
- AWARDED_PRICE_NO
- OPERATORE

54 M rows
(4 GB)

OFF = selling offer
BID = purchase offer

Day hour

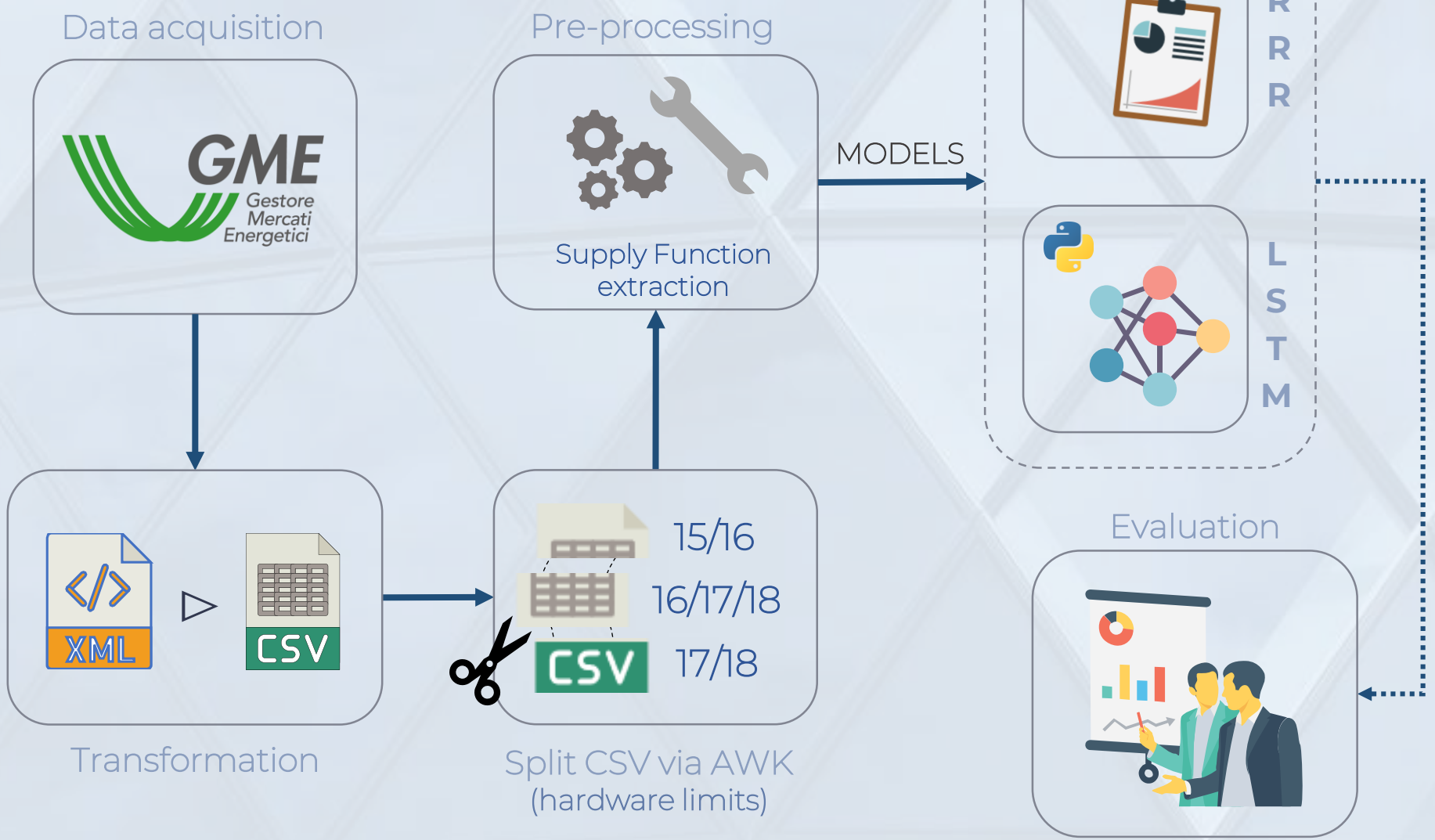
Offer date

Quantity offered
or demanded

Price offered
or demanded

- **PURPOSE_CD**
- STATUS_CD
- UNIT_REFERENCE_NO
- **INTERVAL_NO**
- **BID_OFFER_DATE_DT**
- **QUANTITY_NO**
- AWARDED_QUANTITY_NO
- **ENERGY_PRICE_NO**
- MERIT_ORDER_NO
- PARTIAL_QTY_ACCEPTED_IN
- ADJ_QUANTITY_NO
- ZONE_CD
- AWARDED_PRICE_NO
- OPERATORE

PIPELINE


Data acquisition
GME
Gestore Mercati Energetici
Pre-processing
Supply Function extraction
MODELS
R
R R R
L S T M
XML
CSV
Transformation
15/16
16/17/18
17/18
CSV
Split CSV via AWK (hardware limits)
Evaluation


PRE-PROCESSING

FOCUS

Target on 2015-2016 data:

- Dimensionality of the dataset
- Data density

DATA MANIPULATION

Due to hours change during the year some days have **23** or **25** INTERVALS

- 25 » Delete the 25th hour
- 23 » Replicate the 23rd to obtain the 24th

INTERVAL_NO and BID_OFFER_DATE_DT are concatenated into *datetime format*

GRID

The **grid** is realized using 50-iles of an interval going from 0 to 260 (cap. price in the Italian market, based on this paper*)

*M. Pelagatti, «*Supply Function Prediction in Electricity Auctions*»

FOCUS

Target on 2015-2016 data:

- Dimensionality of the dataset
- Data density

GRID

The **grid** is realized using 50-iles of an interval going from 0 to 260 (cap. price in the Italian market, based on this paper*)

Due to hours change during the year some days have **23** or **25** INTERVALS

- 25 » Delete the 25th hour
- 23 » Replicate the 23rd to obtain the 24th

INTERVAL_NO and BID_OFFER_DATE_DT are concatenated into *datetime format*

DATA MANIPULATION

*M. Pelagatti, «*Supply Function Prediction in Electricity Auctions*»

FOCUS

Target on 2015-2016 data:

- Dimensionality of the dataset
- Data density

Due to hours change during the year some days have **23** or **25** INTERVALS

- 25 » Delete the 25th hour
- 23 » Replicate the 23rd to obtain the 24th

INTERVAL_NO and BID_OFFER_DATE_DT are concatenated into *datetime format*

GRID

The **grid** is realized using 50-iles of an interval going from 0 to 260 (cap. price in the Italian market, based on this paper*)

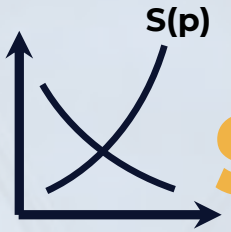
DATA MANIPULATION

*M. Pelagatti, «*Supply Function Prediction in Electricity Auctions*»

Apply `groupby` on prices in order to take only unique values

The offers are ordered with respect to the price cumulating the quantities

From each supply function 51 prices based on the grid values are sampled, obtaining 51 time series



SUPPLY FUNCTION

LOG-TRANSFORMATION

Transform the original ordinate points preserving positivity and non-decreasing monotonicity

$$q_{i,t} := \begin{cases} \log S_t(p_i), & \text{for } i = 0; \\ \log\left(S_t(p_i) - S_t(p_{i-1}) + c\right), & \text{for } i = 1, \ldots, 50. \end{cases}$$

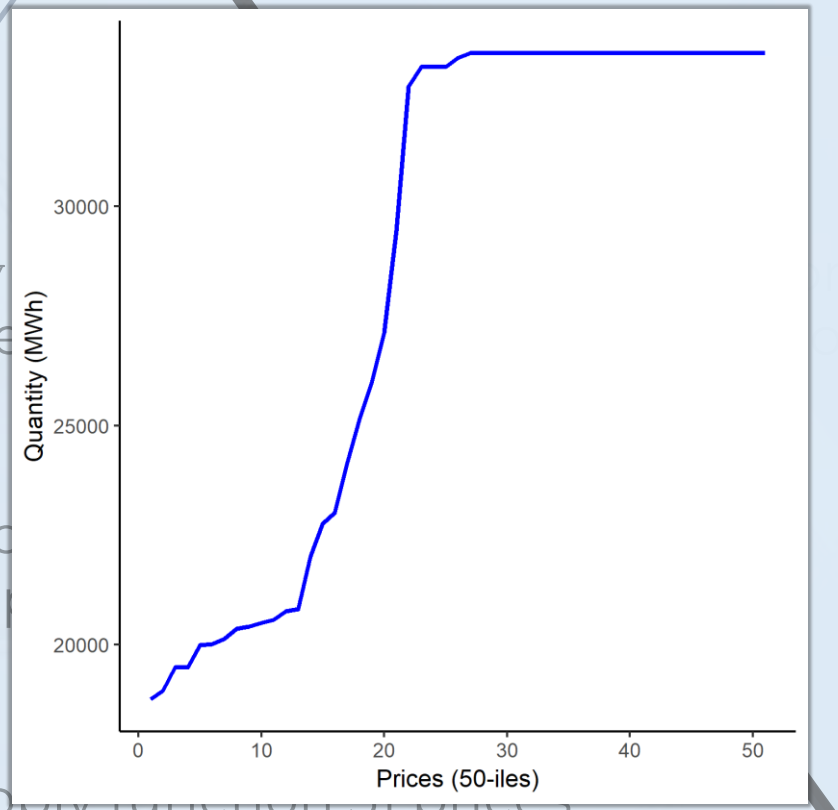
After the prediction phase, the inverse transformation is applied to the results.

$$\hat{S}_t(p_i) = \begin{cases} \exp\left(\hat{q}_{i,t} + s_{i,t}^2/2\right), & \text{for } i = 0; \\ \exp\left(\hat{q}_{i,t} + s_{i,t}^2/2\right) + \hat{S}_t(p_{i-1}) - c, & \text{for } i = 1, \ldots, 50. \end{cases}$$

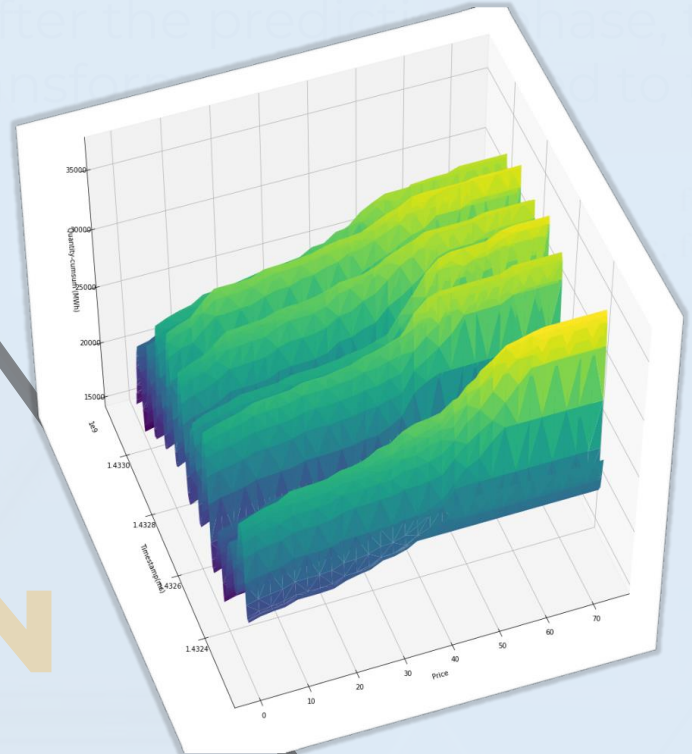
Apply `groupby`
in order to take
unique values

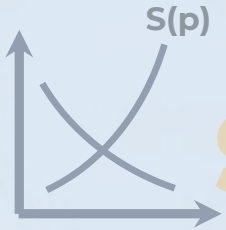
The offers are c
respect to the p
the quantities

From each supply function 51 prices
based on the grid values are sampled,
obtaining 51 time series

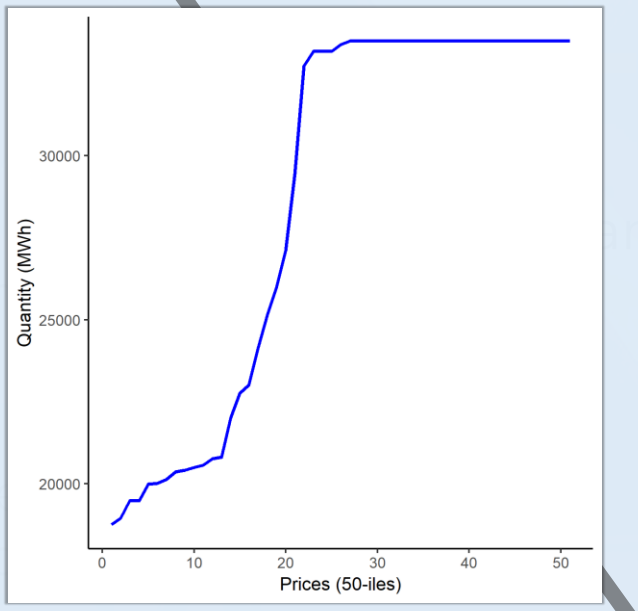


The cumulative quantity based on the ordered prices is taken in order to obtain a Supply Function

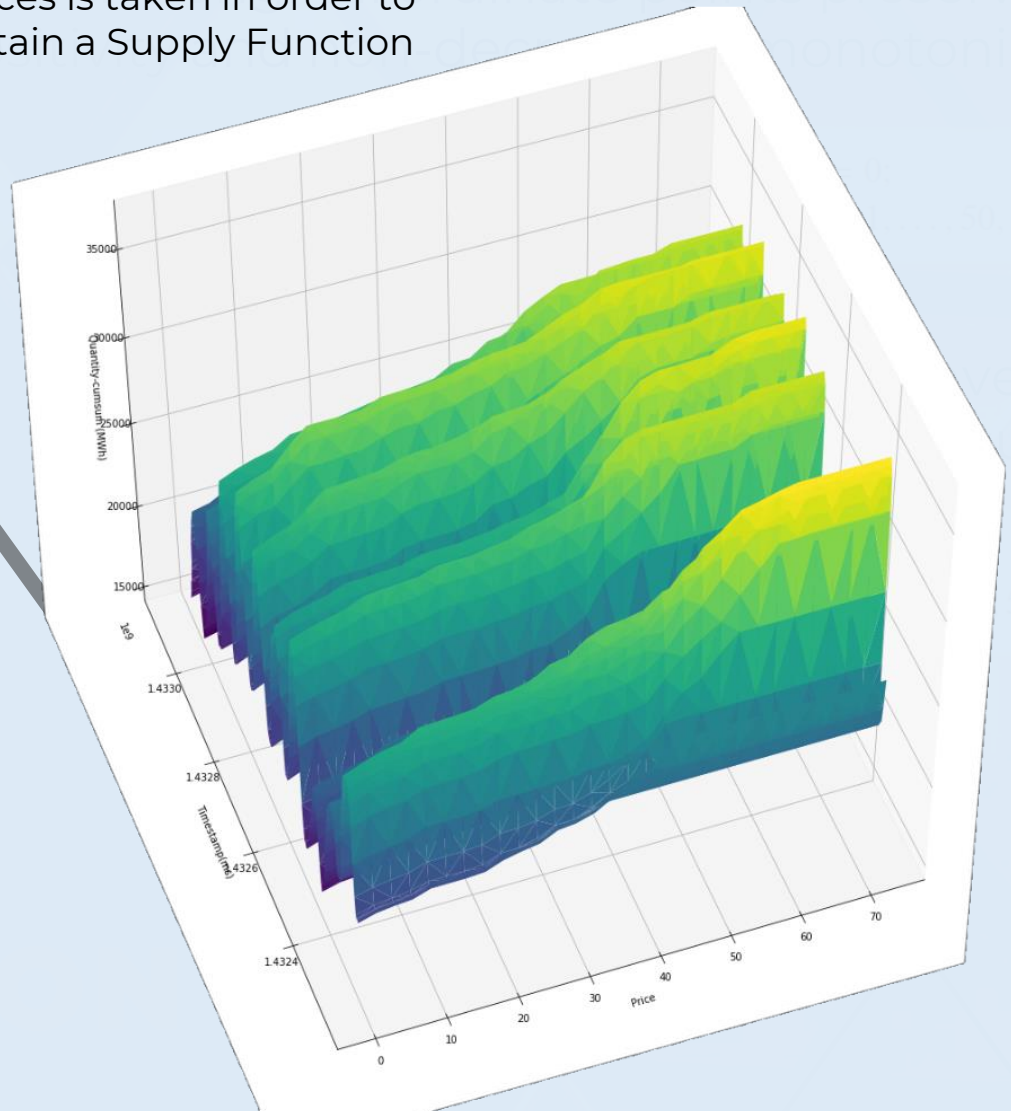




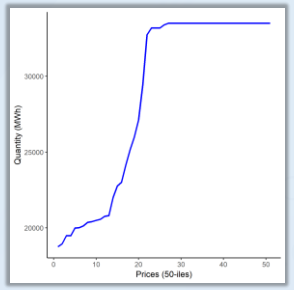
SUPPLY FUNCTION



The cumulative quantity based on the ordered prices is taken in order to obtain a Supply Function



There will be a curve for each time interval in the dataset

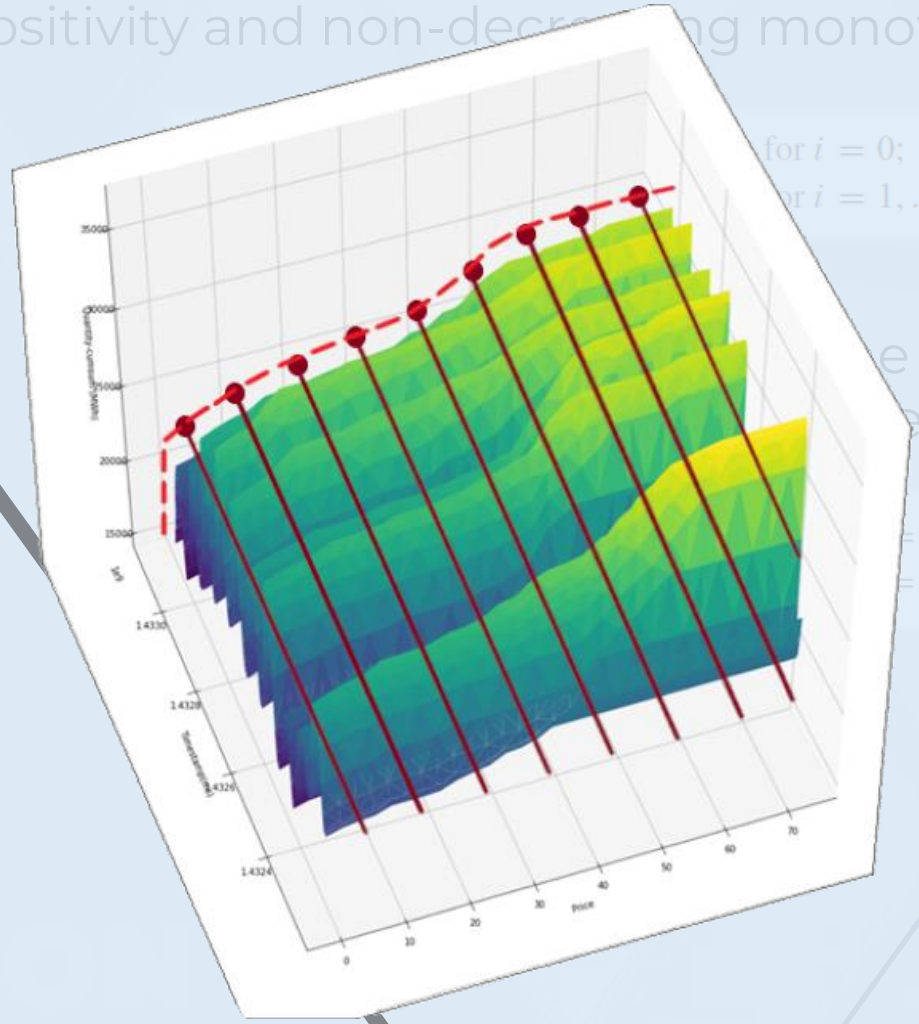


The cumulative quantity based on the ordered prices is taken in order to obtain a Supply Function

There will be a curve for each time interval in the dataset

Objective: to predict this curve in a future time

The **estimated Supply Function** is obtained by linking together the predictions of each time series



LOG-TRANSFORMATION

Apply `groupby` on prices in order to take only unique values

The offers are ordered with respect to the price accumulating the quantities

From each supply function 51 prices based on the grid values are sampled, obtaining 51 time series of ordinate points

Transform the original ordinate points preserving positivity and non-decreasing monotonicity

$$q_{i,t} := \begin{cases} \log S_t(p_i), & \text{for } i = 0; \\ \log\left(S_t(p_i) - S_t(p_{i-1}) + c\right), & \text{for } i = 1, \ldots, 50, \end{cases}$$

After the prediction phase, the inverse transformation is applied to the results

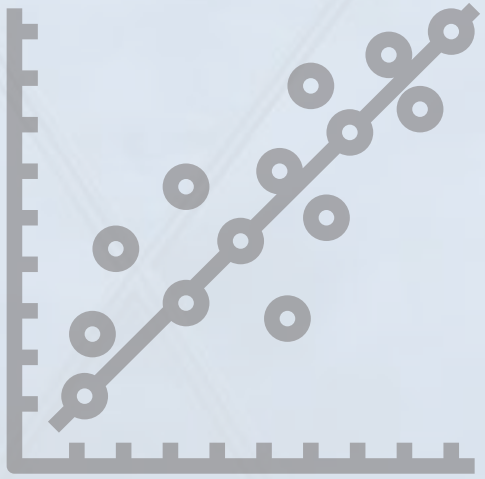
$$\hat{S}_t(p_i) = \begin{cases} \exp\left(\hat{q}_{i,t} + s_{i,t}^2/2\right), & \text{for } i = 0; \\ \exp\left(\hat{q}_{i,t} + s_{i,t}^2/2\right) + \hat{S}_t(p_{i-1}) - c, & \text{for } i = 1, \ldots, 50. \end{cases}$$

SUPPLY FUNCTION


MODELS

RRR

The **Reduced Rank Regression** (RRR) model is a multivariate regression model with a coefficient matrix of reduced rank



Reducing the rank of the coefficient matrix means that few linear combinations of the regressors are enough to take into account all the variability of dependent variable

The optimal rank is chosen evaluating iteratively minimizing the out-of-sample MSE or RMSE using Cross Validation

RRR

The applied models include **1**-hour, **24**-hours and **168**-hours (1 week) forecasting

Every forecasting model make use of the corresponding **Lag** in time and the subsequent ones

(e.g. $y_t \; on \; y_{t-1}, y_{t-24}, y_{t-168}$)

Each model is performed also on 3 different matrices of **deterministic variables**, called Reg. 1, Reg. 2 and Reg. 3

RRR

Reg. 1

- Sinusoids for annual seasonality

Reg. 2

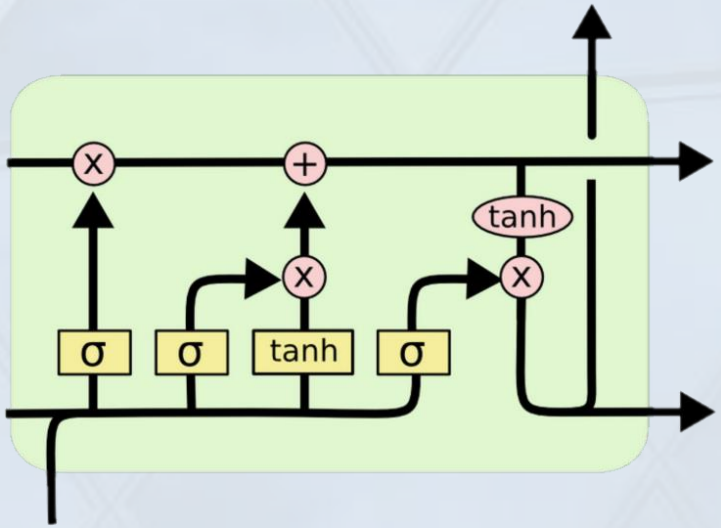
- Sinusoids for annual seasonality
- Dummies indicating Saturday, Sunday and Monday

Reg. 3

- Sinusoids for annual seasonality
- Dummies indicating Saturday, Sunday and Monday
- Sinusoids for daily seasonality

LSTM

The Long Short-Term Memory network (LSTM) is a **recurrent neural network** trained using backpropagation
This family of NN is particularly suitable for sequence modelling



The core concept of LSTM is the cell state, and it's various gates

The cell state acts as a transport channel that propagates information all the way down the sequence chain

In order to emulate a reduced rank effect (like RRR models), the network flow is **compressed** using a hidden layer with a reduced number of neurons in order to re-create a kind of information bottleneck

LSTM - The Model

Loss function = MSE

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout, Flatten
from keras.layers import CuDNNLSTM
from keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam, SGD, Nadam
from time import time
#!pip install livelossplot
#from livelossplot import PlotLossesKeras
from keras.layers.advanced_activations import LeakyReLU, PReLU
import tensorflow as tf
from tensorflow.python.client import device_lib
from sklearn.preprocessing import StandardScaler


  def create_model(look_back, bott, reg=True):
      model = Sequential()
      #1 layer
      model.add(CuDNNLSTM(500, input_shape=(look_back, n_features), return_sequences=True, kernel_initializer='TruncatedNormal'))
      if reg:
        model.add(BatchNormalization())
      model.add(LeakyReLU())
      if reg:
        model.add(Dropout(rate=0.1))
      #2
      for _ in range(1):
          model.add(CuDNNLSTM(128, kernel_initializer='TruncatedNormal', return_sequences=True))
          if reg:
            model.add(BatchNormalization())
          model.add(LeakyReLU())
          if reg:
            model.add(Dropout(rate=0.1))
      #3
      for _ in range(1):
        model.add(CuDNNLSTM(51, kernel_initializer='TruncatedNormal', return_sequences=False))
        if reg:
```

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout, Flatten
from keras.layers import CuDNNLSTM
from keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam, SGD, Nadam
from time import time
#!pip install livelossplot
#from livelossplot import PlotLossesKeras
from keras.layers.advanced_activations import LeakyReLU, PReLU
import tensorflow as tf
from tensorflow.python.client import device_lib
from sklearn.preprocessing import StandardScaler


  def create_model(look_back, bott, reg=True):
      model = Sequential()
      #1 layer
      model.add(CuDNNLSTM(500, input_shape=(look_back, n_features), return_sequences=True, kernel_initializer='TruncatedNormal'))
      if reg:
        model.add(BatchNormalization())
      model.add(LeakyReLU())
      if reg:
        model.add(Dropout(rate=0.1))
      #2
      for _ in range(1):
          model.add(CuDNNLSTM(128, kernel_initializer='TruncatedNormal', return_sequences=True))
          if reg:
            model.add(BatchNormalization())
          model.add(LeakyReLU())
          if reg:
            model.add(Dropout(rate=0.1))
      #3
      for _ in range(1):
        model.add(CuDNNLSTM(51, kernel_initializer='TruncatedNormal', return_sequences=False))
        if reg:
          model.add(BatchNormalization())
        model.add(LeakyReLU())
        if reg:
          model.add(Dropout(rate=0.1))
      #4
      for _ in range(1):
          model.add(Dense(rank))
          if reg:
            model.add(BatchNormalization())
          model.add(LeakyReLU())
          if reg:
            model.add(Dropout(rate=0.1))
      #5
      model.add(Dense(n_features))

      return model
```

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout, Flatten
from keras.layers import CuDNNLSTM
from keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam, SGD, Nadam
from time import time
#!pip install livelossplot
#from livelossplot import PlotLossesKeras
from keras.layers.advanced_activations import LeakyReLU, PReLU
import tensorflow as tf
```

```
model.add(CuDNNLSTM(500, input_shape=(look_back, n_features),
if reg:
  model.add(BatchNormalization())
model.add(LeakyReLU())
if reg:
  model.add(Dropout(rate=0.1))
```

```
        if reg:
          model.add(BatchNormalization())
        model.add(LeakyReLU())
        if reg:
          model.add(Dropout(rate=0.1))
    #3
    for _ in range(1):
      model.add(CuDNNLSTM(51, kernel_initializer='TruncatedNormal', return_sequences=False))
      if reg:
        model.add(BatchNormalization())
      model.add(LeakyReLU())
      if reg:
        model.add(Dropout(rate=0.1))
    #4
    for _ in range(1):
        model.add(Dense(rank))
        if reg:
          model.add(BatchNormalization())
        model.add(LeakyReLU())
        if reg:
          model.add(Dropout(rate=0.1))
    #5
    model.add(Dense(n_features))

    return model
```

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout, Flatten
from keras.layers import CuDNNLSTM
from keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam, SGD, Nadam
from time import time
#!pip install livelossplot
#from livelossplot import PlotLossesKeras
from keras.layers.advanced_activations import LeakyReLU, PReLU
import tensorflow as tf
from tensorflow.python.client import device_lib
from sklearn.preprocessing import StandardScaler


  def create_model(look_back, bott, reg=True):
      model = Sequential()
      #1 layer
      model.add(CuDNNLSTM(500, input_shape=(look_back, n_features), return_sequences=True, kernel_initializer='TruncatedNormal'))
      if reg:
        model.add(BatchNormalization())
      model.add(LeakyReLU())
      if reg:
        model.add(Dropout(rate=0.1))
      #2
      for _ in range(1):
          model.add(CuDNNLSTM(128, kernel_initializer='TruncatedNormal', return_sequences=True))
          if reg:
            model.add(BatchNormalization())
          model.add(LeakyReLU())
          if reg:
            model.add(Dropout(rate=0.1))
      #3
      for _ in range(1):
        model.add(CuDNNLSTM(51, kernel_initializer='TruncatedNormal', return_sequences=False))
        if reg:
          model.add(BatchNormalization())
        model.add(LeakyReLU())
        if reg:
          model.add(Dropout(rate=0.1))
      #4
      for _ in range(1):
          model.add(Dense(rank))
          if reg:
            model.add(BatchNormalization())
          model.add(LeakyReLU())
          if reg:
            model.add(Dropout(rate=0.1))
      #5
      model.add(Dense(n_features))

      return model
```

```
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout, Flatten
from keras.layers import CuDNNLSTM
from keras.callbacks import EarlyStopping, TensorBoard, ModelCheckpoint
from keras.layers.normalization import BatchNormalization
from keras.optimizers import Adam, SGD, Nadam
from time import time
#!pip install livelossplot
#from livelossplot import PlotLossesKeras
from keras.layers.advanced_activations import LeakyReLU, PReLU
import tensorflow as tf
from tensorflow.python.client import device_lib
from sklearn.preprocessing import StandardScaler


  def create_model(look_back, bott, reg=True):
      model = Sequential()
      #1 layer
      model.add(CuDNNLSTM(500, input_shape=(look_back, n_features), return_sequences=True, kernel_initializer='TruncatedNormal'))
      if reg:
        model.add(BatchNormalization())
      model.add(LeakyReLU())
      if reg:
        model.add(Dropout(rate=0.1))
      #2
      for _ in range(1):
          model.add(CuDNNLSTM(128, kernel_initializer='TruncatedNormal', return_sequences=True))
          if reg:
            model.add(BatchNormalization())
          model.add(LeakyReLU())
          if reg:
```

```
for _ in range(1):
    model.add(Dense(rank))
    if reg:
      model.add(BatchNormalization())
    model.add(LeakyReLU())
    if reg:
      model.add(Dropout(rate=0.1))
```

```
      model.add(Dense(n_features))

      return model
```

RESULTS

RESULTS

Each model is applied to a **training set** of 12’992 records and evaluated on a **test set** of 1’000 records

The out-of-sample evaluations are performed using **RMSE** (Root Mean Square Error) and **MAPE** (Mean Absolute Percentage Error) as metrics

EVALUATION - RRR

Table 1: Out-of-sample root mean square error for the three forecast and 9 models

	Reg. 1		Reg. 2		Reg. 3	
	Rank	RMSE	Rank	RMSE	Rank	RMSE
1-step	51	3494.8	51	3733.5	51	4555.2
24-step	50	4678.9	50	4415.4	50	5179.5
168-step	51	5567.5	51	5440.9	50	4775.4

Table 2: Out-of-sample mean absolute percentage error computed for each day of the week

Pred.	Reg.	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Ave.
1-step	Reg. 1	10.6	11.9	**11.1**	11.6	11.4	9.94	10.1	11.0
	Reg. 2	12.5	12.1	13.1	12.6	12.7	**7.7**	**6.4**	11.0
	Reg. 3	**9.8**	**10.0**	11.5	**9.4**	**10.0**	10.0	9.5	**10.0**
24-step	Reg. 1	14.5	14.4	**11.6**	14.2	13.3	11.7	10.3	12.9
	Reg. 2	15.4	**12.1**	12.0	13.0	12.5	**11.1**	**9.4**	12.2
	Reg. 3	**14.4**	12.4	12.8	**10.7**	**10.7**	11.9	11.8	**12.1**
168-step	Reg. 1	15.1	13.5	13.4	15.9	13.6	14.8	**12.0**	14.0
	Reg. 2	14.9	12.9	12.4	15.0	12.8	14.9	13.5	13.8
	Reg. 3	**13.6**	**12.3**	**11.8**	**11.4**	**10.7**	**13.4**	**12.0**	**12.2**

EVALUATION - LSTM

Table 3: Out-of-sample root mean square error

	Rank	RMSE
1-step	35	**1074.8**
24-step	30	2245.1
168-step	35	3538.9

Table 4: Out-of-sample mean absolute percentage error computed for each day of the week

Pred.	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Ave.
1-step	**4.0**	**3.3**	**3.2**	**3.2**	**3.0**	**2.9**	**4.7**	**3.5**
24-step	6.1	5.2	5.8	5.5	5.1	4.5	6.9	5.6
168-step	14.4	9.3	6.9	11.2	8.6	7.0	10.4	9.7


CONCLUSIONS

CONSIDERATIONS

- RRR is faster and simpler (+)
- RRR potentially interpretable (+)
- RRR allows to use cross validation in order to find the best rank (+)
- RRR achieves poor performances (−)

- LSTM has better performances (+)
- LSTM is more complex to implement (−)
- LSTM as NN is not interpretable (−)
- LSTM tends to overfit the train set, ideally needs more data (−)

IMPROVEMENTS

More continuous data

More time

Specific models for time series forecasting

More powerful hardware


THANK YOU