Program reskilling Data/AI PO
Specific session for SOCIETE GENERALE

2: Time Series Analysis

Eric Benhamou

Agenda

Session 1: Time Series Analysis - Understanding time series analysis

What is a time series?

- Concept
- How to generate value
- Machine to Machine (M2M) communication

AI modeling techniques

- Introducing the time series model – ARIMA
- Introducing neural networks – the secret sauce for accurately predicting demand
- Backpropagation - Neural network architecture - Using epochs for neural network training
- Scaling - Sampling

Demand forecasting using time series analysis using neural networks on Keras

- Data flow & Preprocessing the data (in the SQL database)
- Importing libraries and defining variables & Reading in data
- Preprocessing the data (in Python)
- Training and validating the model
- Testing the model
- Visualizing the test result
- Generating the model for production

Summary

Recall from previous session

- In the previous session, we **introduced AI, machine learning, and deep learning**.
- We also discovered how the banking sector functions and **how the use of AI can enhance banking processes**. We learned the importance of banking processes being easily accessible.
- Overall, the session provided **the necessary background for the application of machine learning in the banking industry to solve various business problems**.

Goal of this session

- In this session, we will learn about an algorithm that analyzes **historical data to forecast future behavior, known as time series analysis**.
- Time series analysis works on the basis of one variable —time. It emphasizes **the importance of chronology**
- It is the process of capturing data points, also known as observations, at specific time intervals.
- The goal of this session is to understand time series analysis in detail through examples and explain how **Machine-to-Machine (M2M) communication** can be helpful in the implementation of time series analysis.

What is a time series?

- A time series is technically defined as the ordered sequence of values of a variable captured over a uniformly spaced time interval.
  - Put simply, it is the method of capturing the value of a variable at specific time intervals.
  - It can be 1 hour, 1 day, or 20 minutes.
  - The captured values of a variable are also known as data points.
  - Time series analysis is performed in order to understand the structure of the underlying sources that produced the data. It is also used in forecasting, feedforward control, monitoring, and feedback.

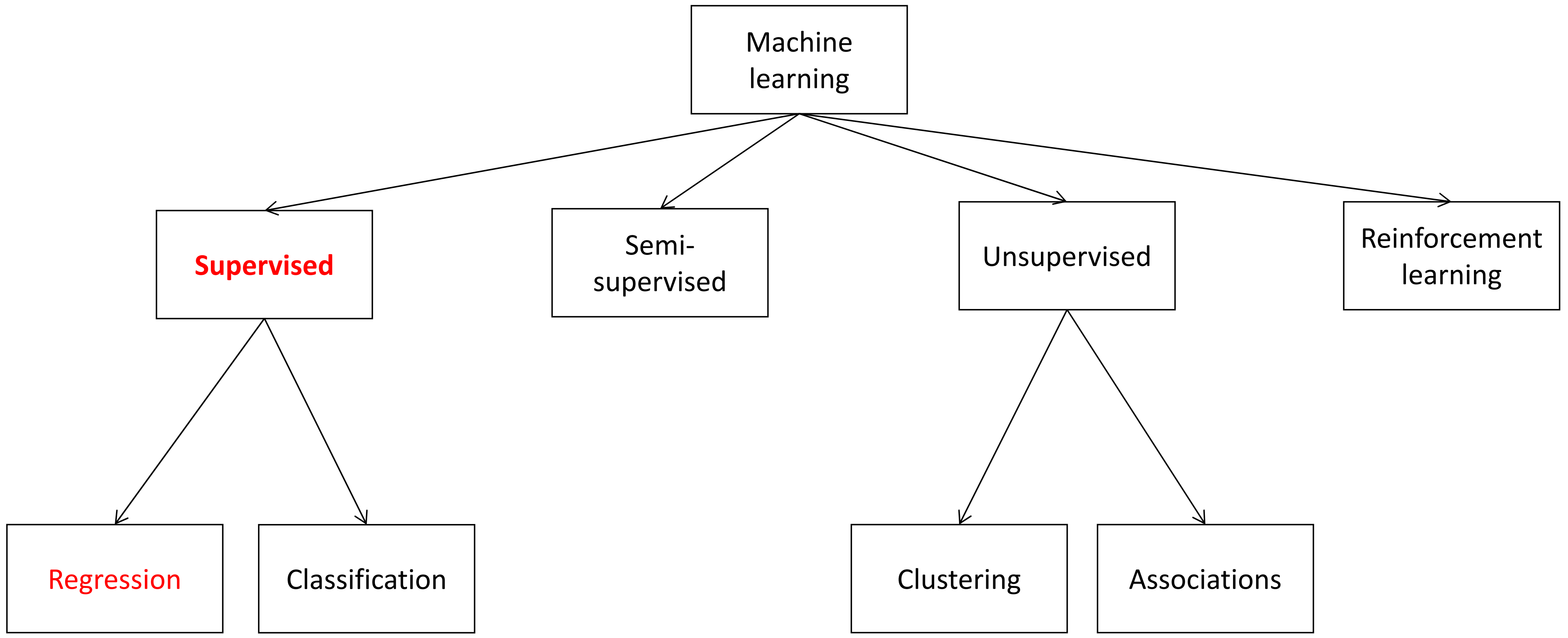
Type of time series analysis

- Can you cite applications of time series analysis?
  - Utility studies
  - Stock market analysis
  - Weather forecasting
  - Sales projections
  - Workload scheduling
  - Expenses forecasting
  - Budget analysis

How to extract value

- Time series analysis is achieved by applying various analytical methods to extract meaningful information from raw data that has been captured from various data sources
- Time series analysis is also useful for producing statistics and other characteristics of data—for example, the size of data, the type of data, the frequency of data, and more.
- In time series analysis, the capturing of a value is done at a point of observation.

ML taxonomy



Machine learning
Supervised
Semi-supervised
Unsupervised
Reinforcement learning
Regression
Classification
Clustering
Associations

Lab: forecasting demand for electricity consumption

- In this section, we will look at forecasting demand for electricity consumption, and predict energy expenses using time series analysis.
- Motivations: Today, electricity or energy is a very basic necessity for all of us. We use electricity and pay bills. Now, as a customer, we want to analyze electricity consumption and predict future consumption and predict energy expenses. This is the problem that we will solve in this section.
- Time series analysis is the optimal approach for solving this problem. Machine learning models need large datasets to be fed before the actual solution is derived. Here are the steps that we will follow:
    1. Downloading the data
    2. Preprocessing the data
    3. Model fitting the data

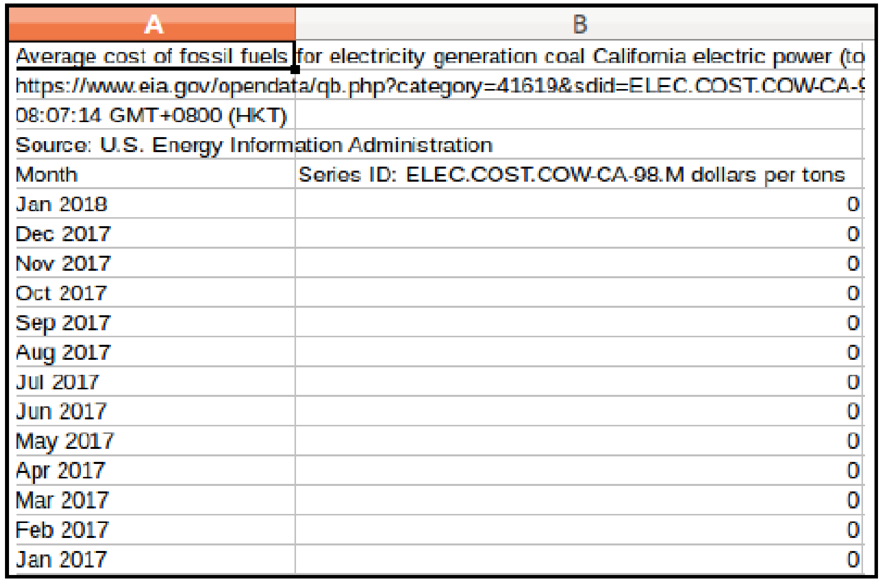
Downloading the data

- Start by downloading data regarding electricity consumption and energy expenses. Even though we can download data from public websites now, in a true production environment, it is not uncommon to download data from an internal database and pass it to users as a flat file (a text file with no database structure).

- You can download the files from the following paths:
  - **Consumption:** https://www.eia.gov/opendata/qb.php?category=873sdid= ELEC.CONS_TOT.NG-CA-98.M
  - **Cost:** https://www.eia.gov/opendata/qb.php?category=41625sdid=ELEC.COST. NG-CA-98.M
  - **Revenue:** https://www.eia.gov/opendata/qb.php?category=1007sdid=E LEC. REV.CA-RES.M

Preprocessing the data 1/2

- After we obtain the data, we align it together in the same time series, as the data we've downloaded can cover different periods of time. As data scientists, we strive to align our data in one single sheet of data, with all of the required data listed column by column (that is, cost, consumption, sales, and more):



A	B
Average cost of fossil fuels for electricity generation coal California electric power (to	
https://www.eia.gov/opendata/qb.php?category=41619&sdid=ELEC.COST.COW-CA-	
08:07:14 GMT+0800 (HKT)	
Source: U.S. Energy Information Administration	
Month	Series ID: ELEC.COST.COW-CA-98.M dollars per tons
Jan 2018	0
Dec 2017	0
Nov 2017	0
Oct 2017	0
Sep 2017	0
Aug 2017	0
Jul 2017	0
Jun 2017	0
May 2017	0
Apr 2017	0
Mar 2017	0
Feb 2017	0
Jan 2017	0

Preprocessing the data 2/2

- Each line (or row) of the data should represent a single month. Right before we feed our data for the machine to learn the patterns, we will need to set aside some data for testing and some for learning. With the testing data, we can see whether the model predicts well, without training on the learning data first.. We do not feed the testing dataset for ML/training. This is a fundamental step in all ML/predictive models

- In this program, we set aside the earliest 70% of data points as training data for the machine to learn and adapt to, while keeping the latter 30% of data points as testing data. This is data that will be used to compare against the prediction made by the model, not used to fit the data.

Model fitting the data

- In the part, we will first introduce the Autoregressive Integrated Moving Average (ARIMA), the most traditional type of forecasting model before going to deep networks to see the difference. We will also introduce a neural network model. ARIMA is a class of statistical models that is used to forecast a time series using past values.

- ARIMA is an acronym for the following:
    - AR (autoregression): Autoregression is a process that takes previous data values as inputs, applies this to the regression equation, and generates resultant prediction-based data values.
    - I (integrated): ARIMA uses an integrated approach by using differences in observations to make the time series equally spaced. This is done by subtracting the observation from an observation on a previous step or time value.
    - MA (moving average): A model that uses the observation and the residual error applied to past observations.

Some quick definition on ARIMA

- This ARIMA model belongs to parametric modelling—models that are fitted by known parameters. Normally, we classify this type of model as a statistical model because we need to make assumptions about what the data looks like. This is considerably different for wider machine learning models that do not have any preset assumptions about what the data looks like.
- However, in a real banking scenario, a statistical approach is still prevalent among the econometrics, quantitative finance, and risk management domains. This approach works when we have a handful of data points, for example, around 30 to 100 data points. However, when we have a wealth of data, this approach may not fare as well as other machine learning approaches.

- ARIMA assumes that there is a stationary trend that we can describe. The autoregressive terms, *p* and *d*, are each significant in their own way:
    - *p* means the number of past period(s) that is affecting the current period value (for example, *p = 1: Y current period = Y current -1 period * coefficient + constant*).
    - Non-seasonal difference (*d*) refers to the number of past periods progression impacting the current period values (for example, *d = 1*: the difference between *Y* now versus *Y* in the past period).
    - Lagged terms (*q*) means the number of the past period's forecast errors impacting the current period values.
- Consider an example in which *q = 1: Y* impacted by an error in the *t - 1* period—here, error refers to the difference between the actual and predicted values.
- In a nutshell, ARIMA specifies how the previous period's coefficient, constant, error terms, and even predicted values impact the current predicted values. It sounds scary, but it is, in fact, very understandable. After the model is fit, it will be asked to make a prediction and be compared against the actual testing data.
- The deviation of the prediction from the testing data will record the accuracy of the model. We will use a metric called the Mean Square Error (MSE) in this chapter to determine the fitness of the model to the data.

neural networks – the secret sauce for accurately predicting demand 1/2

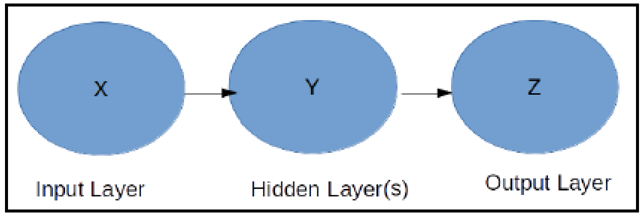
- A neural network is an attempt by a computer to **mimic how our brain works**—it works by connecting different computing points/neurons with different settings. Architecture-wise, it looks like layers of formulas. To make it very simple, if we denote by *Y* the **interested outcome** and *X*, the **input variable (features)**, with *b* being the coefficient and *c* being the constant term:

$$Y = b\,X + c$$

- *Y* is what we wish to predict on the left-hand side; on the right-hand side, *bX + c* are the forms that describe how the feature (*X*) is related to *Y*. In other words, *Y* is the output, while *X* is the input. The neural network describes the relationship between the input and the output.
- Suppose that *Z* **(the output)** is what we want to predict:

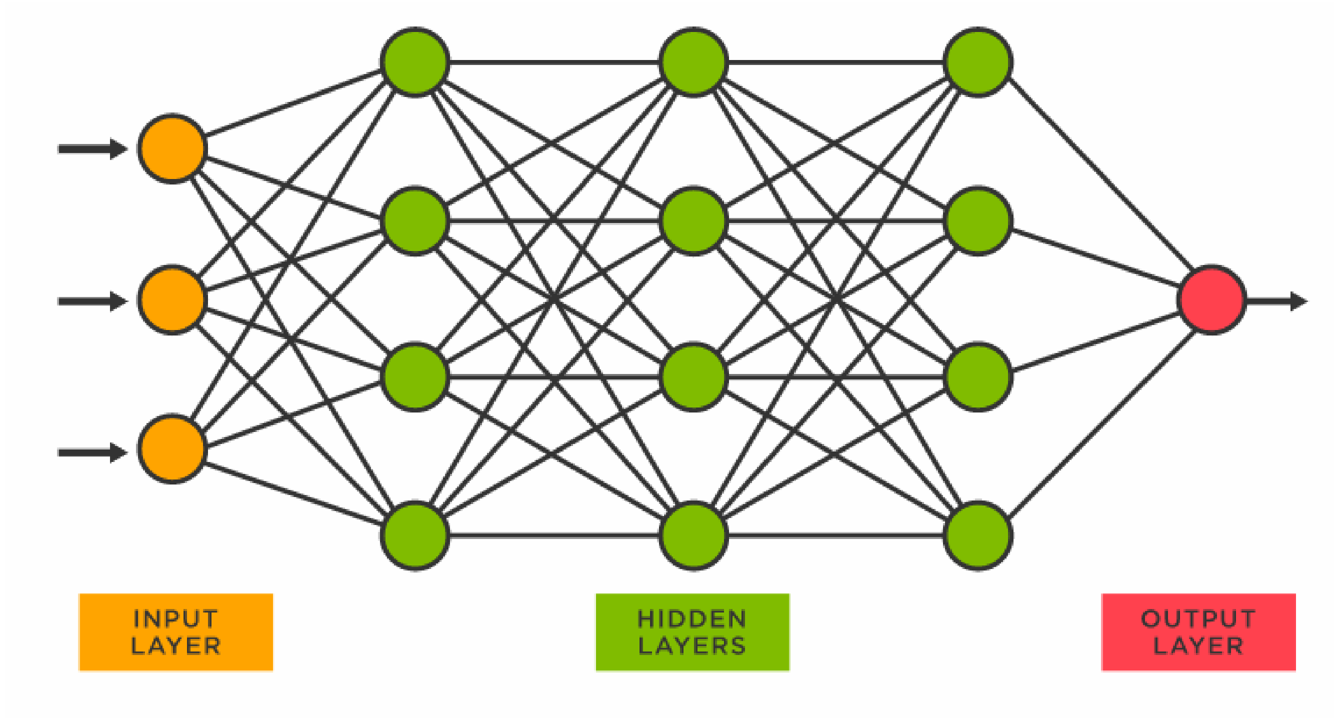
$$Z = d\,Y + e$$

- It seems that the formulas are linked:



neural networks 2/2

- This is the simplest form of a neural network, with one input layer, one hidden layer, and one output layer. Each of the layers has one neuron (point). Here we created linear relationship but we can make non linear **activation** thanks to ReLu, Sigmoid, etc…

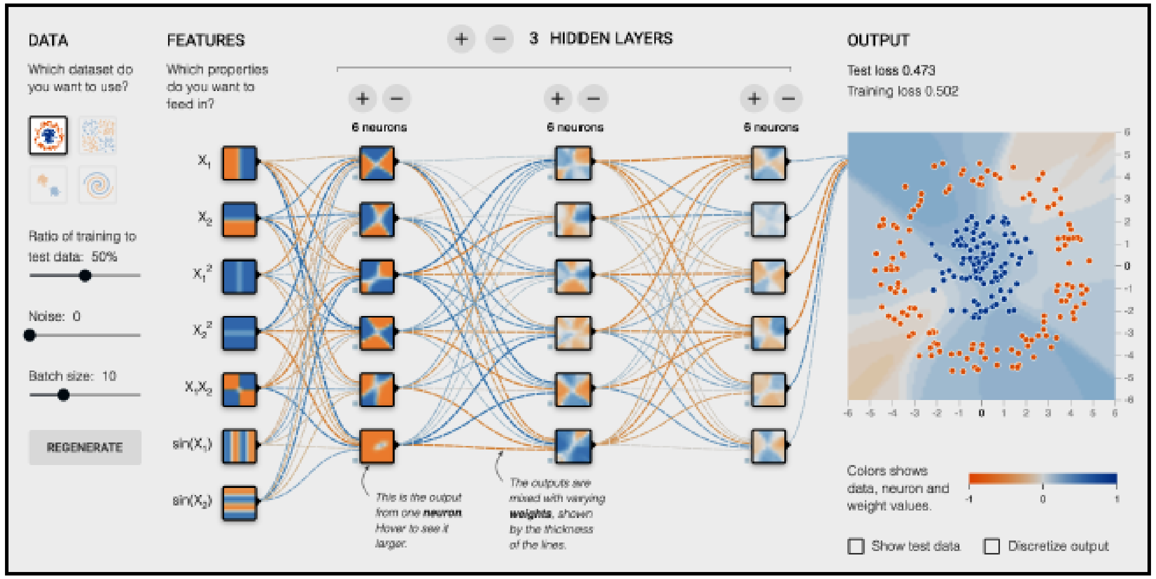


Backpropagation

- There are other concepts in neural networks, such as backpropagation. This refers to the feedback mechanism that fine-tunes the neural network's parameters, which mostly connect neurons within the network (except when it is a constant parameter at the layer). It works by comparing the output at output layer *Z* (predicted) versus the actual value of *Z* (actual). The wider the gap (or the loss) between actual and predicted, the more adjustment of *b*, *c*, *d*, and *e* is needed.

Neural network architecture

- Architecture concerns the layers and number of neurons at each layer, as well as how the neurons are interconnected in a neural network. The input layer is represented as **features**. The output layer can be a single number or a series of numbers (called a **vector**), which generates a number ranging from 0 to 1 or a continuous value—subject to the problem domain.
- For example, to understand the structure of a neural network, we can project that it will look like the following screenshot from TensorFlow Playground (https:/ / playground.tensorflow. org/), which is the visualization of another network with the same hidden layers—three layers with a size of 6:



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

*This is the output from one **neuron**. Hover to see it larger.*

*The outputs are mixed with varying **weights**, shown by the thickness of the lines.*

Test loss 0.473

Training loss 0.502

Colors shows data, neuron and weight values.

☐ Show test data ☐ Discretize output

Using epochs for neural network training

- Besides the design of the neural network, we also utilize the epoch parameter, which
- indicates the number of times the same set of data is fed to the neural network. We need to increase the number of epochs if we do not have enough data to satisfy the number of parameters in neural networks. Given that we have *X* parameters in the neural network, we need at least *X* data points to be fed to the network. Unfortunately, if our data point is only *X*/2, we need to set epoch to 2 in order to make sure that we can feed *X* data points (all of them are fed twice) to the network.

Scaling and train test

- Before feeding the features to the machine learning model, we will normalize the input features of different magnitudes to be of the same magnitude.

- For example, the price and volume of goods are different types of numeric data. The scaling process will make sure that both of them are scaled to the same range, from 0 to 1. In classical statistical modelling processes, this step is very important to avoid a particular feature of bigger scales that dominate the influence on the prediction.

- Apart from data column-level scaling, we also need to pay attention to the sampling bias of the model. Normally, we will set aside a portion of the data unseen by the machine while it is training and learning on another set of data—which is called a training set. Later on, the testing set (which is the dataset kept aside) will be used to check against the prediction made by the model.

Lab 1: doing ARIMA

- Once the data is clean, we will start training the machine to learn about the pattern. The training data will be fed to the machine as fitting. The model is like a shirt and the training data is like the body we're attempting to fit it to.

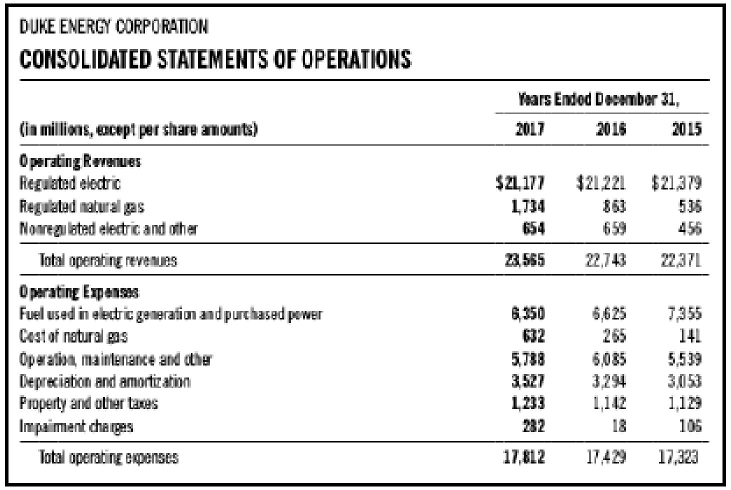
- Here are the steps to fit our data into an ARIMA model:
  - 1. For each data file/field in the consolidated file, we run *step 3* and *step 4* (which have been marked in the code file for the following code block).
  - 2. If the Boolean variable, parameter_op, is set to True, then we will run *step 5* (which is marked as well). This explores all the possible combinations of parameters in ARIMA with regard to p, d, and q, which are set as follows:
    - p: Ranging from 0 to 12
    - d: Ranging from 0 to 2
    - q: Ranging from 0 to 3
  - 3. For combinations of any of the preceding values, we calculate the fitness of the data to the actual pattern and measure the error values. The combinations with the lowest error values are selected as the chosen parameters of the ARIMA model.

Lab 2:Procuring commodities using neural networks on Keras

- In this example, we want to forecast the procurement of commodities based on historical data. The commodity that we are going to use is natural gas. In the case of natural gas, we do not have any control over its pricing because it is a hugely globalized commodity. However, we can still set up the internal procurement strategy when the pricing of the natural gas hits a certain range. The profitability ratio target will constrain the maximum pricing we can pay for the raw material to be profitable for the owners of the firm. We will track the profitability ratio, which is the ratio of cost of natural gas to sales.

Data

- Let's understand this pricing constraint with an example. In this example, we assume that for each dollar spent where the unit cost of natural gas (for electric power) increased, the cost of materials to sales of the energy company will increase by 9.18% (this is based on 3 years of data):



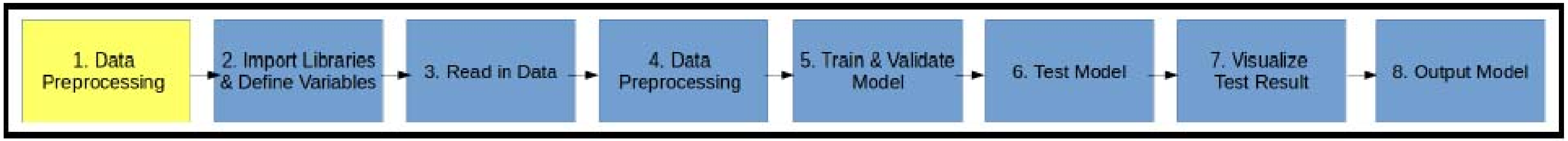
DUKE ENERGY CORPORATION

CONSOLIDATED STATEMENTS OF OPERATIONS

	Years Ended December 31,		
(in millions, except per share amounts)	**2017**	**2016**	**2015**
Operating Revenues			
Regulated electric	**$21,177**	$21,221	$21,379
Regulated natural gas	**1,734**	863	536
Nonregulated electric and other	**654**	659	456
Total operating revenues	**23,565**	22,743	22,371
Operating Expenses			
Fuel used in electric generation and purchased power	**6,350**	6,625	7,355
Cost of natural gas	**632**	265	141
Operation, maintenance and other	**5,788**	6,085	5,539
Depreciation and amortization	**3,527**	3,294	3,053
Property and other taxes	**1,233**	1,142	1,129
Impairment charges	**282**	18	106
Total operating expenses	**17,812**	17,429	17,323

Data flow

- The following data flow outlines the steps we need to take in order to prepare and generate the code to build the commodity procurement model. The first box denotes a script run on the SQLite database; the other boxes denote steps run on Python:



Pre-processing the data (in the SQL database)

- Data pre-processing means converting the data into the desired data features. We run it outside of Python coding to reduce the layers involved (that is, we interact directly with SQLite rather than using Python to interact with SQLite). Here are the steps involved in performing database operations:

  1. Create the SQLite database.
  2. Import the data as a staging table.
  3. Create the required table(s)—a one-time operation.
  4. Insert the staging table into the actual table with data type and format transformation.
  5. Create the view that does the feature engineering.
  6. Output the preprocessed view as CSV data.

Importing libraries and defining variables

- Import libraries and define variables to make sure that the relevant functions can be used. Import all of the relevant libraries:
    - pandas: This is for data storage before data is fed to the machine learning module.
    - keras: This an easy-to-use machine learning framework that has another library.
    - tensorflow: This is used as the backend.
    - sklearn: This is a very popular machine learning module that provides lots of data preprocessing toolkits along with some machine learning models that are easy to use. The models are not used in this example, as we wish to build up the foundation for the more extensive use of machine learning models afterward. In addition, sklearn also has metrics that appraise the performance of the models.
    - matplotlib: This is the default data visualization tool.

Code: importing libraries

```
'''***********************************
2. import all the libraries required
'''
import pandas as pd

from keras.models import Model
from keras.layers import Dense, Input
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

import matplotlib.pyplot as plt

import pickle

demand_model_path = 'demand_model.h5'
f_in_name = 'consumption_ng_exp.csv'
```

Code: reading the data

```
'''************************************
#3. Read in data
'''
pd_trade_history = pd.read_csv(f_in_name,header=0)
pd_trade_history = pd_trade_history.drop('date_d',1)
```

Code: Preprocessing the data

```
'''***********************************
4. Pre-processing data
'''
#4.A: select features and target
df_X = pd_trade_history.iloc[:,:-5]
df_Y = pd_trade_history.iloc[:,-4:]

np_X = df_X.values
np_Y = df_Y.values

#4.B: Prepare training and testing set
X_train, X_test, Y_train, Y_test = train_test_split(np_X, np_Y, test_size = 0.2)

#4.C: scaling the inputted features
sc_X = StandardScaler()
X_train_t = sc_X.fit_transform(X_train)
X_test_t = sc_X.fit_transform(X_test)
```

Code: Training and validating the model

```
'''**********************************
#5. Build the model
'''
inputs = Input(shape=(14,))
x = Dense(7, activation='relu')(inputs)
x = Dense(7, activation='relu')(x)
x = Dense(7, activation='relu')(x)
x = Dense(4, activation='relu')(x)
x = Dense(4, activation='relu')(x)
x = Dense(4, activation='relu')(x)
x = Dense(4, activation='relu')(x)
predictions = Dense(units=4, activation='linear')(x)
demand_model= Model(inputs=inputs,outputs=predictions)
demand_model.compile(loss='mse', optimizer='adam', metrics=['mae'])
demand_model.fit(X_train_t,Y_train, epochs=7000, validation_split=0.2)
Y_pred = demand_model.predict(X_test_t)

#conver numpy as dataframe for visualization
pd_Y_test = pd.DataFrame(Y_test)
pd_Y_pred = pd.DataFrame(Y_pred)
```

Code: Testing the model

```
'''*************************************
##6. Test model: Measure the model accuracy
combine both actual and prediction of test data into data
'''
data = pd.concat([pd_Y_test,pd_Y_pred], axis=1)
data_name = list(data)[0]
data.columns=['actual1','actual2','actual3','actual4','predicted1','predicted2','predicted3','predicted4']

error1 = mean_squared_error(data['actual1'],data['predicted1'])
print('Test MSE 1: %.3f' % error1)
error2 = mean_squared_error(data['actual2'],data['predicted2'])
print('Test MSE 1: %.3f' % error2)
error3 = mean_squared_error(data['actual3'],data['predicted3'])
print('Test MSE 1: %.3f' % error3)
error4 = mean_squared_error(data['actual4'],data['predicted4'])
print('Test MSE 1: %.3f' % error4)
```

Code: visualizing the prediction accuracy

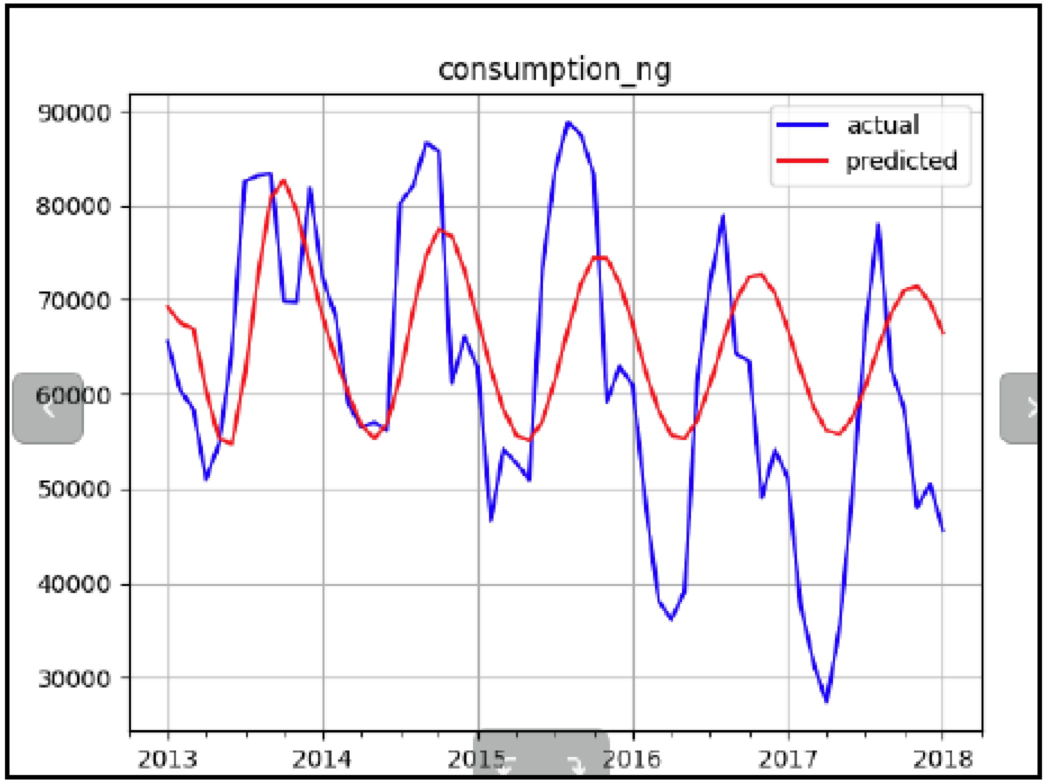
```
data.actual1.plot(color='blue',grid=True,label='actual1',title=data_name)
data.predicted1.plot(color='red',grid=True,label='predicted1')
plt.legend(); plt.show(); plt.close()

data.actual2.plot(color='blue',grid=True,label='actual2',title=data_name)
data.predicted2.plot(color='red',grid=True,label='predicted2')
plt.legend(); plt.show(); plt.close()

data.actual3.plot(color='blue',grid=True,label='actual3',title=data_name)
data.predicted3.plot(color='red',grid=True,label='predicted3')
plt.legend(); plt.show(); plt.close()

data.actual4.plot(color='blue',grid=True,label='actual4',title=data_name)
data.predicted4.plot(color='red',grid=True,label='predicted4')
plt.legend(); plt.show(); plt.close()
```

Expected result



Code: Generating model for production

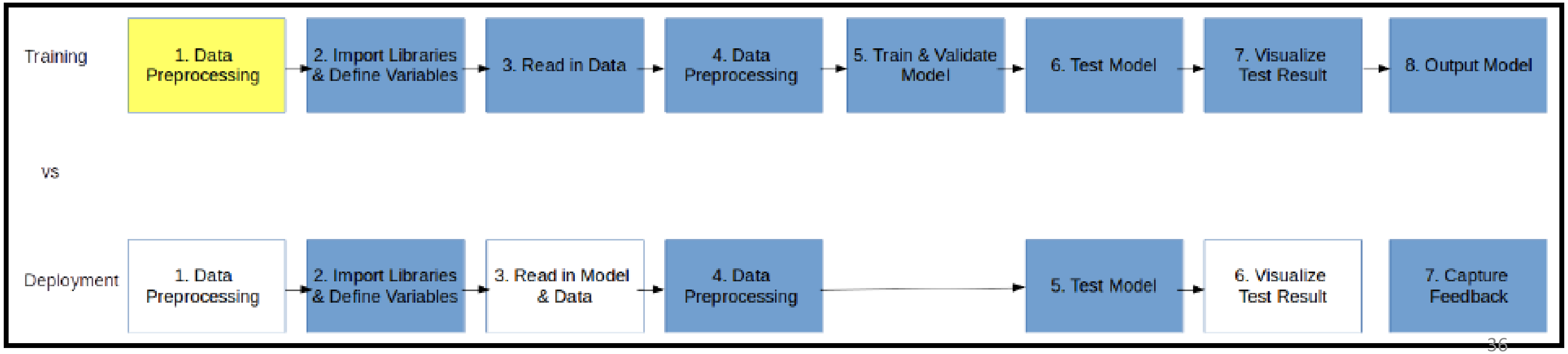
```
'''*************************************
#8. Output the models
'''
demand_model.summary()
demand_model.save(demand_model_path)
f_scaler=open('x_scaler.pkl',"wb+")
pickle.dump(sc_X, f_scaler)
```

Generating model for production

- The model that was trained and tested in *steps 5* and *6* will be output as a file for the production system to run on unseen production data. We will output two files
  - one for scaling the input features
  - and another one for the neural network

Well done!

- You **have now delivered a model** that can be used at the operational level to identify the quantity to order for this month's demand, next month, and the month after. The following diagram shows the steps in the training versus the deployment of machine learning models:



Summary

- In this chapter, you learned about time series analysis, the benefits of time series analysis for banking and two useful examples by defining the problem statement and deriving the solution step by step.

- We also learned about the basic concepts of time series analysis and a few techniques, such as ARIMA and deep learning