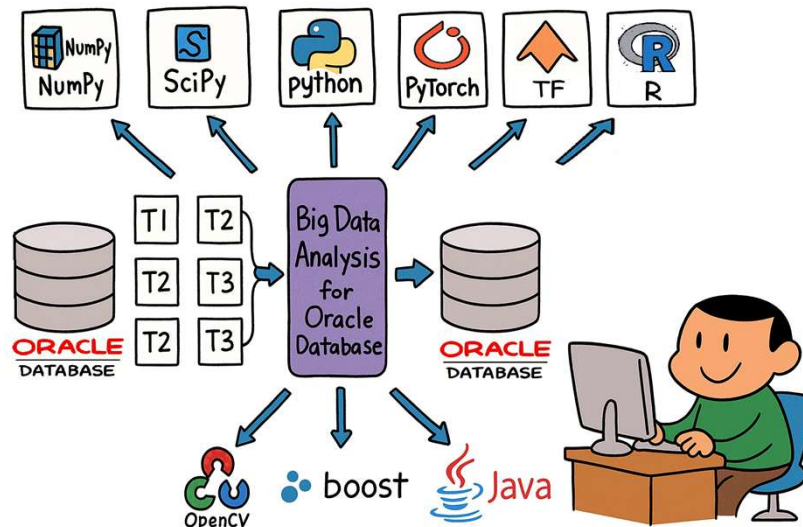


Big Data Analysis Eabler(BDAE™)

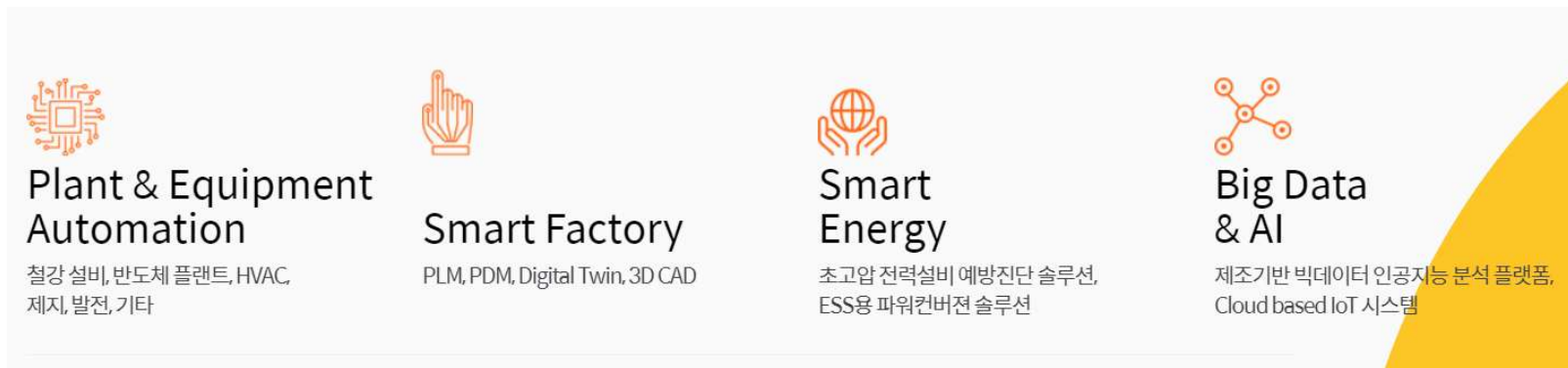
Introduction Summary



Raymond,
2025/05/16

1 | General

BDAE^(TM) can be used in most fields that use **Oracle Database^(TM)**, Non-stop installation/patching, algorithms or logic can be applied immediately to existing solutions without any downtime.

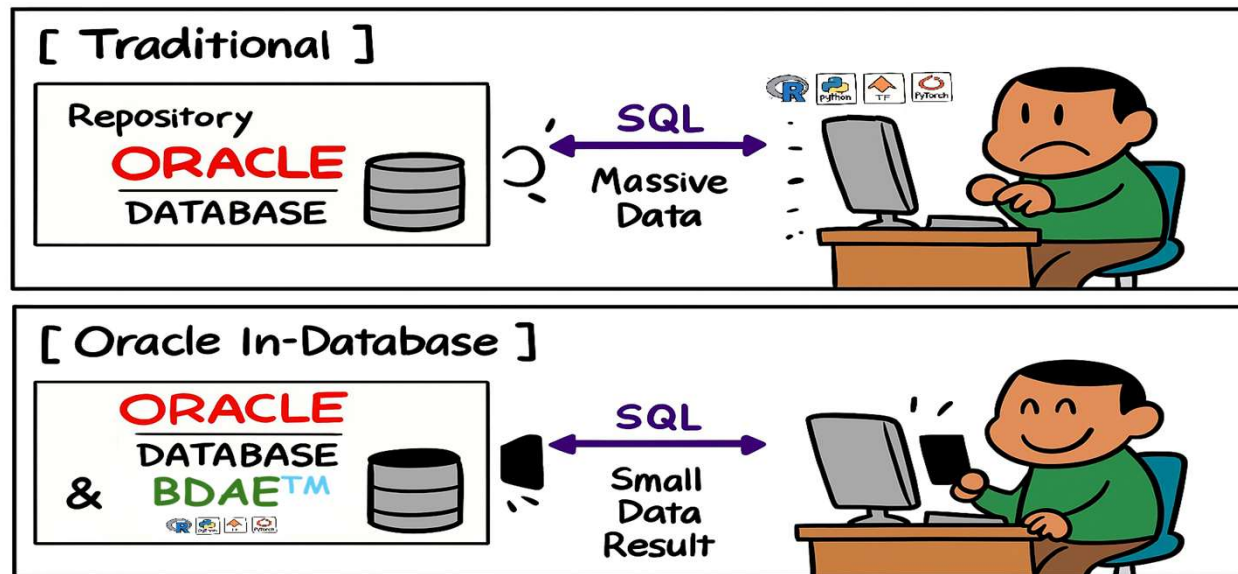


BDAE^(TM) is executed as a SQL statement and the results are retrieved like the results of a general SQL statement, so a separate application server is unnecessary, and all logic (including the backend) can be implemented in Python and R.

Available for use in all MES, SPC, FDC, YMS and Smart Factory configuration systems in manufacturing, finance, energy, etc.

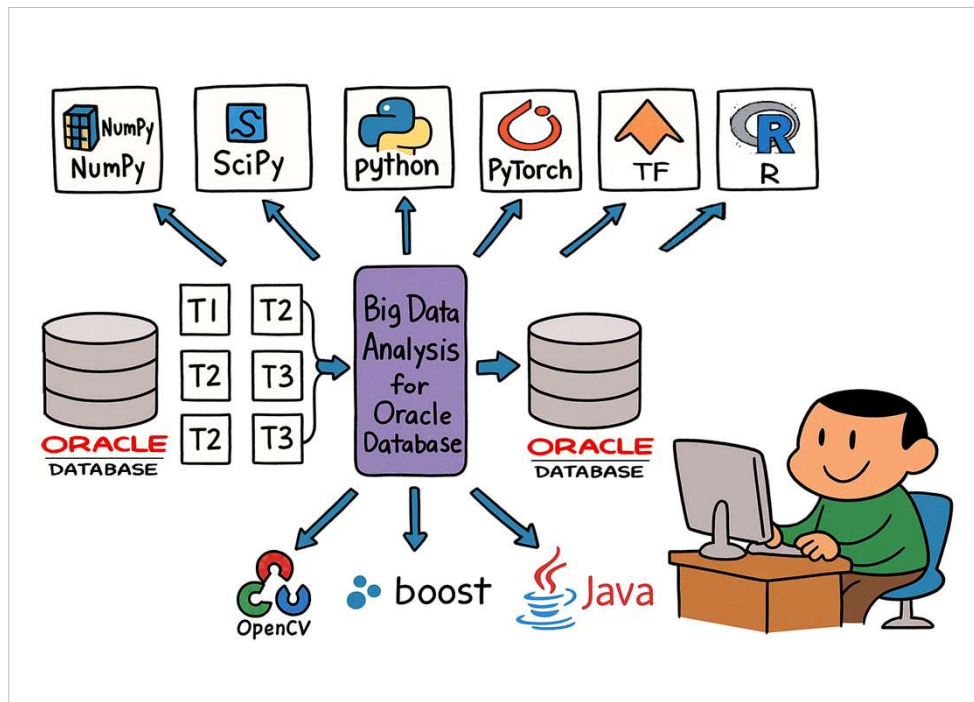
2 | BDAETM for What

BDAETM is built on Oracle In-Database technology and has platform features that enable Oracle DatabaseTM to be used not only as a simple storage for general AI tasks, but also as a non-stop operating environment without the overhead of data movement during learning and inference.



※ BDAETM advantages include development productivity (Python, R), parallel processing, and bringing performance into the real-time domain.

3 | BDAETM SW Configurations & Architecture

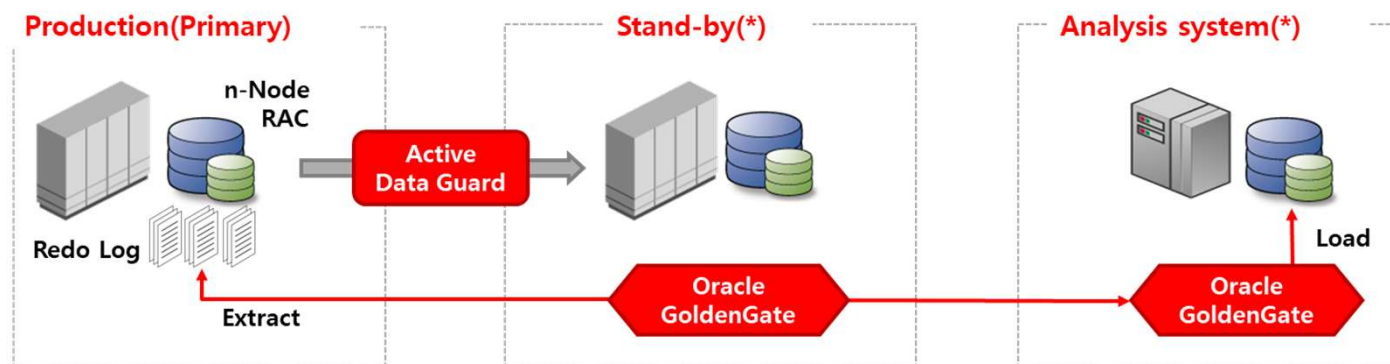


Left Image Shows the operating location of BDAETM in the form of **Oracle In-Database**.

Parallel distributed processing is a feature of **Oracle In-Database**, and analysts do not need to consider it in their modules, which increases the reusability of logic.

In addition, the fact that it can be integrated with various analysis engines can be seen as an advantage of BDAETM.

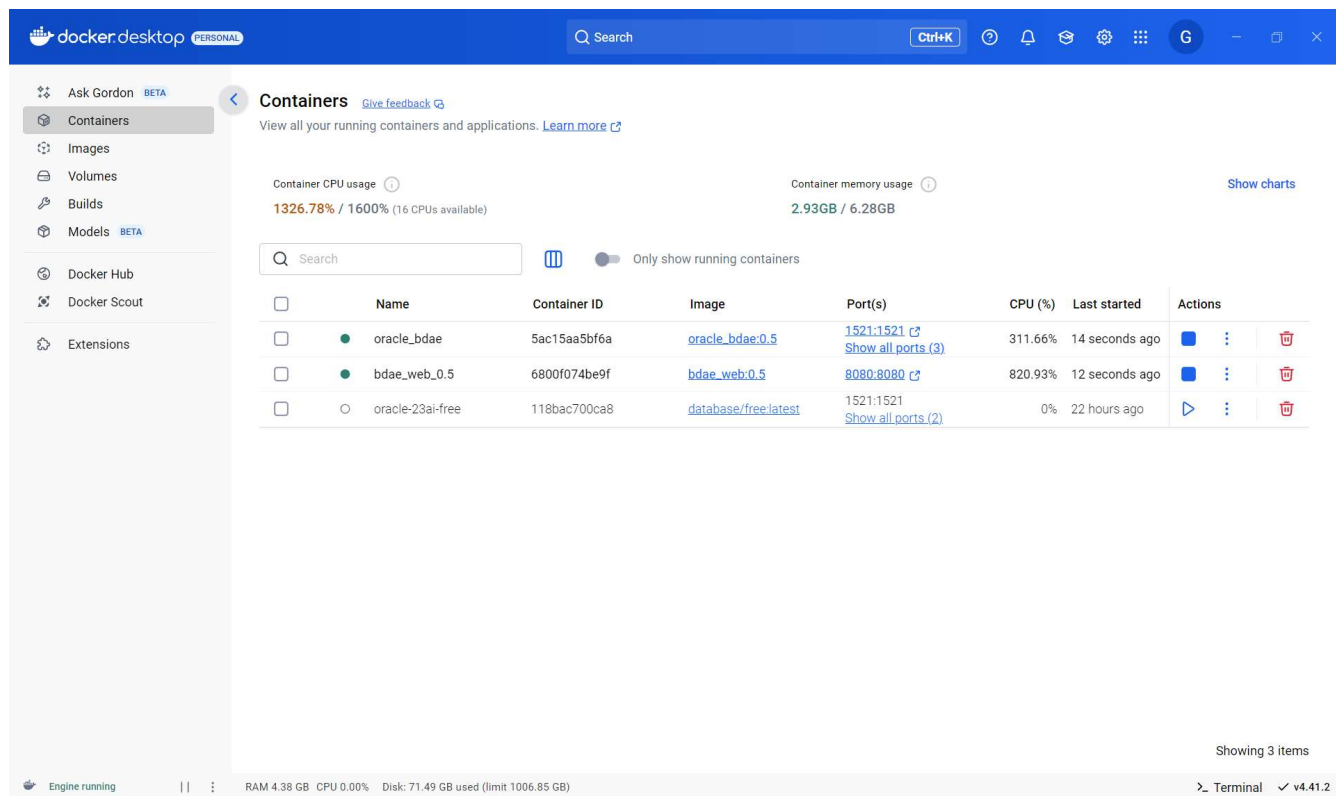
BDAETM can operate on any architectural configuration of **Oracle DatabaseTM**.



4 | BDAETM Docker-based Demo environment

BDAETM consists of two parts: oracle_bdae and bdae_web.

bdae_web is made with Spring Boot + JSP, and oracle_bdae is installed with BDAETM on the image distributed by Oracle, and then distributed to customers as an export file. (It can be delivered via USB storage device)



※ If necessary, please request to gracesjy@naver.com. I will provide you with the Google Cloud shared file URL.

5 | BDAE^(TM) Build, Packages

BDAE^(TM) is built on Oracle Linux^(TM) (OL, formerly OEL: Oracle Enterprise Linux^(TM)). BDAE^(TM) can run on any Linux or UNIX environment supported by Oracle Database^(TM).

As of now, BDAE^(TM) has been built on versions 6, 7, 8, and 9 of the RedHat series (including Oracle Linux). Once built, it is provided in the form of a library file along with PL/SQL, etc., and can be installed.

Oracle In-Database Programs(including BDAE^(TM)) must be written in C with PL/SQL, Type, Function, etc., so it needs to be built. Engines such as R and Python are all written in C, and can be viewed as being similar to the device driver of the operating system. (Oracle Database is guaranteed by OL/OEL and is related to the GLIBC version.)

※ Customers can install Python and R packages themselves, and on Nexus-based systems, the package location can be changed to Private Network.

※ Setting up parallel processing for a specific schema is as simple as creating a BDAE^(TM) Oracle Function, and you can refer to the existing built-in examples. (This will be explained in the future when introduced.)

6 | BDAE^(TM) Docker Installation

In a Windows environment, install Docker Desktop first and then proceed as follows. For Docker Desktop installation, refer to the relevant Internet document.

Import two Docker Images: Oracle Database^(TM) + BDAE^(TM) , BDAE^(TM) Web and run them respectively.

```
docker load -i oracle_bdae.tar # Oracle Database(TM) + BDAE(TM)
docker load -i bdae_web.tar   # BDAE(TM) Web

# check docker images above, and run belows
docker run --name oracle_bdae -p 1521:1521 -p 5500:5500 -p 8888:8888 oracle_bdae:0.5
docker run --name bdae_web -p 8080:8080 bdae_web:0.5
```

Oracle Database Access :

> Oracle Service Name : FREE, TNS Port : 1521, IP : 127.0.0.1

Web Server Access :

> <http://127.0.0.1:8080>

7 | Before using BDAE^(TM) ..

Oracle R Enterprise^(TM), a commercial product from Oracle Corporation, supports only the R language. Of course, great analysts have built their main algorithms into R packages, but the reality is that a lot of code needs to be written.

And the written code must be stored as a DB table and does not provide a GUI.

On the other hand, BDAE^(TM) supports for Python, R, and JAVA for ETL (mainly Hadoop, etc.), has a simple web screen, and inputs the customer's algorithm into the DB through this web editor.

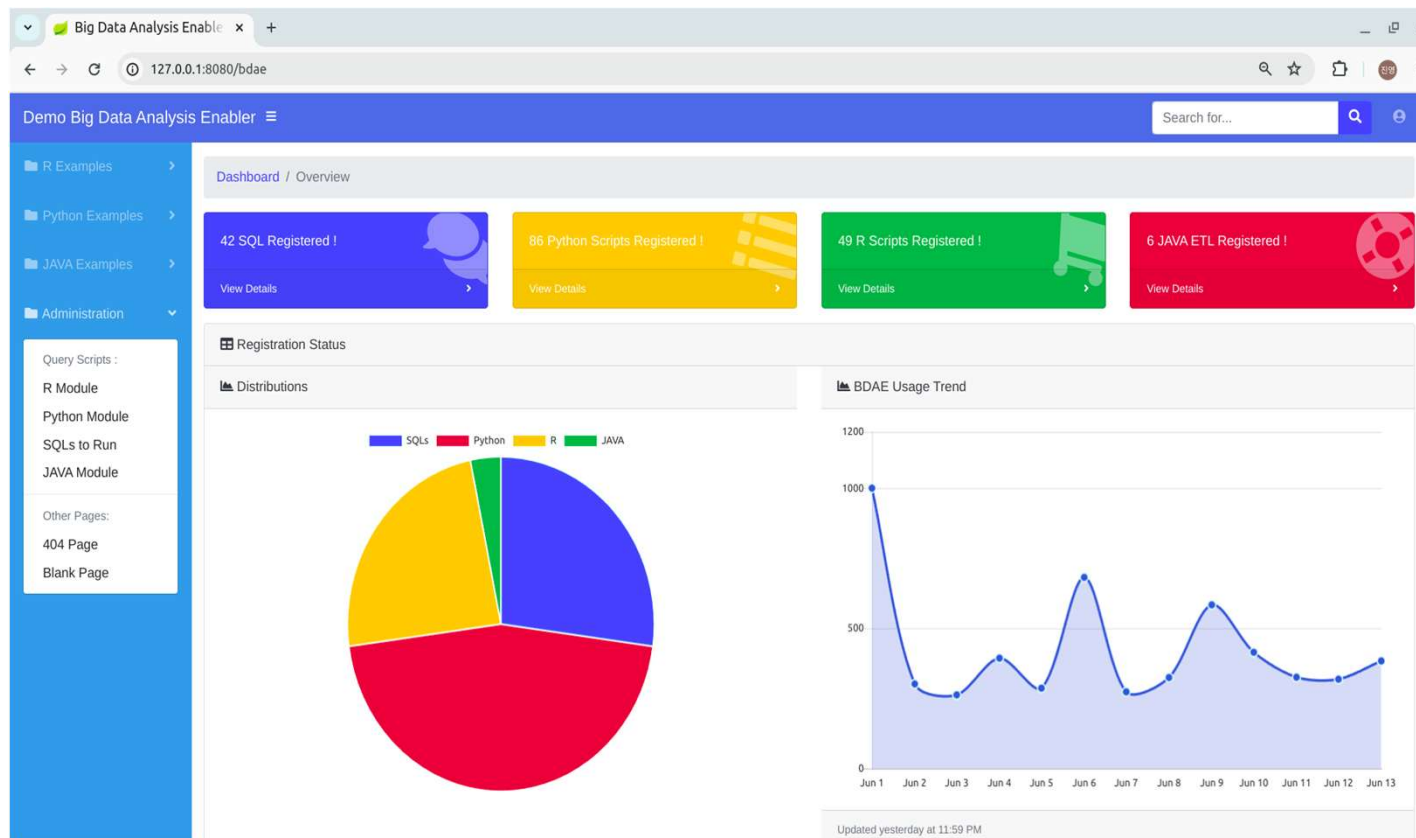
Normal development is often done in R Studio for R or Jupyter notebook for Python, and what is developed here can be copied and pasted into BDAE^(TM) Web.

※ BDAE^(TM) Web is a bundle, and it is intended for reference when building a basic CRUD function and a future customer-specific construction. (When used with Apache NIFI^(TM), it is effective for real-time, batch work, scheduling, etc.)

8 | BDAE^(TM) Web

You can register, search, and delete R Modules and Python Modules mainly through the Administration menu on the left, and you can register data and SQL for calls and check the results by executing them immediately.

(※ The basic purpose of BDAE^(TM) Web is to register and execute customers' algorithms in Python and R modules.)



9 | Oracle R Enterprise™ SQL

Below is the Embedded Script Execution form provided by Oracle R Enterprise™.

The bottom two (sys.rqScriptCreate, rqScriptDrop) are for module input and deletion, which are a bit inconvenient.

In the case of BDAE™, creation, modification, and deletion are done in the Web Editor, so it is convenient.

※ Oracle R Enterprise™ does not have version control capabilities for R code.

On the other hand, BDAE™ has a schema structure that allows version management and is customizable.

※ Looking at the contents of rqScriptCreate, it is INSERT INTO ..., but indentation is very important in Python, and since both Python and R use " ", ' ', etc., it takes more time to write SQL INSERT distinctions. In particular, in the case of Python, since it mixes """ , "" , ' ', it is almost impossible with the INSERT syntax.

Embedded Script Execution – SQL Interface	
SQL Interface function	Purpose
rqEval()	Invoke stand-alone R script
rqTableEval()	Invoke R script with full table as input
rqRowEval()	Invoke R script on one row at a time, or multiple rows in chunks
"rqGroupEval()"	Invoke R script on data partitioned by grouping column
sys.rqScriptCreate	Create named R script
sys.rqScriptDrop	Drop named R script

10 | BDAE^(TM) SQL

BDAE^(TM) supports both R and Python, providing a total of 8 functions, twice that of Oracle R EnterpriseTM. The arguments are the same and both Concept and Oracle In-Database follow the same architecture.

SQL Interface function	Description (Oracle R Enterprise TM 와 비교)
asEval()	Same rqEval()
asTableEval()	Same rqTableEval()
asRowEval()	Same rqRowEval()
asGroupEval()	Same rqGroupEval()
apEval()	Invoke stand-alone Python module
apTableEval()	Invoke Python module with full table as input
apRowEval()	Invoke Python module on one row at a time, or multiple rows in chunks
apGroupEval()	Invoke Python module on data partitioned by grouping column

11 | Register and Run, Using BDAE^(TM) Web

BDAE^(TM) can be registered and executed in 3 steps below.

1. Registering R or Python modules (reusable)
2. Registering BDAE^(TM) SQL statements for integration with DB data and registered R or Python modules.
3. Execute the registered SQL statement above and view the results.

That's all. It patches instantly in real time and doesn't affect existing versions that are running.

Since the data to be analyzed and output by both Oracle R Enterprise^(TM) and BDAE^(TM) exists within the Oracle Database^(TM), everything must be described in SQL statements.

That is, both input and output are in a table-like data format that Oracle Database^(TM) can understand, but the temporary part inside can be composed of various formats and codes.

※ Normally, after performing large-scale analysis or Hadoop or file-based analysis, the results are put into RDBMS, but BDAE^(TM) can automatically perform these tasks.

※ Because Oracle Optimizer uses statistical information and Data Dictionary for input and output when executing SQL and then determines a plan, you must explicitly inform it about input and output. This is why input and output are specified in Oracle R Enterprise^(TM) or BDAE^(TM) SQL format.

12 | R Coding Style (#1)

There are two R coding styles for **BDAETM**, the first of which is a functional form as follows: **BDAETM** automatically finds the function argument names data and args and calls them by inserting the data.

Therefore, the analyst can arbitrarily specify the data and args variable names.

```
function(data, args) {  
  # Registering various dependent libraries  
  library(xts)  
  library(quantmod)  
  library(RCurl)  
  library(logr)  
  library(xts)  
  library(quantmod)  
  library(RCurl)  
  
  ...  
  ... < Any data type, model, or algorithm code supported by R can be accommodated. > ...  
  ...  
  
  df <- data.frame(col1=c(...), col2=c(...), ... ,stringsAsFactors=FALSE)  
  return (df)  
}
```

12 | R Coding Style (#2) - BDAETM Unique Features

Another R coding style is descriptive, like this:

At this time, data is called by putting it in explicit variable names called **data** and **args** in BDAETM, so these two variable names are **Read-Only** and should not be used for other purposes.

Since there is no return at the end, you must write `df`, and this variable name (`df`) can be anything you want.

```
library(xts)
library(quantmod)
library(RCurl)
library(logr)
library(xts)
library(quantmod)
library(RCurl)
```

```
x <- data # The variable that contains the raw data to be processed is data.
```

```
y <- args # Data that contains additional data is args
```

```
...
```

```
... < Any data type, model, or algorithm code supported by R can be accommodated. > ...
```

```
...
```

```
df <- data.frame(col1=c(...), col2=c(...), ... ,stringsAsFactors=FALSE)
```

```
df
```

12 | R Coding Style (#3)

Check for syntax errors and runtime errors:

Analysts do not need to handle exceptions in the **BDAE^(TM)** analysis module (the analyst's R, Python code).

If there is a problem in **BDAE^(TM)**, it is returned as an SQL error code (ORA-20001 to 20500) and an R error message.

BDAE^(TM) first performs a syntax check for **R**, and then processes the data and calls the module only if there are no problems. If there are problems, the analyst resolves them and reruns them.

※ R has poor exception handling compared to Python. The API of the R engine, which is written in C, has limitations.

13 | Python Coding Style (#1)

Python Coding Style is different from R. Python requires a **module name** (usually a file name) and a **start function name** among many functions to be called. (Not the normal way to call `__main__`)

```
import pandas as pd
import numpy as np
import seaborn as sns
from StreamToLogger import StreamToLogger
import sys
import logging, logging.handlers

def describe(df):
    logger = logging.getLogger('TitanicDesc:describe')
    logger.setLevel(logging.DEBUG)
    socketHandler = logging.handlers.SocketHandler('localhost',
        logging.handlers.DEFAULT_TCP_LOGGING_PORT)
    logger.addHandler(socketHandler)

    sys.stdout = StreamToLogger(logger, logging.INFO)
    sys.stderr = StreamToLogger(logger, logging.ERROR)

    print('----- start -----')
    columns_ = df.columns.tolist()
    print(str(columns_))
    df.columns = list(map(str.lower, columns_))
    df_desc = df.describe()
    print(str(df_desc))
    df_desc.reset_index(inplace=True)
    df_desc.columns = ['vars', 'passengerid', 'survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
    df_melt = pd.melt(df_desc, id_vars=['vars'])
    print('----- end -----')

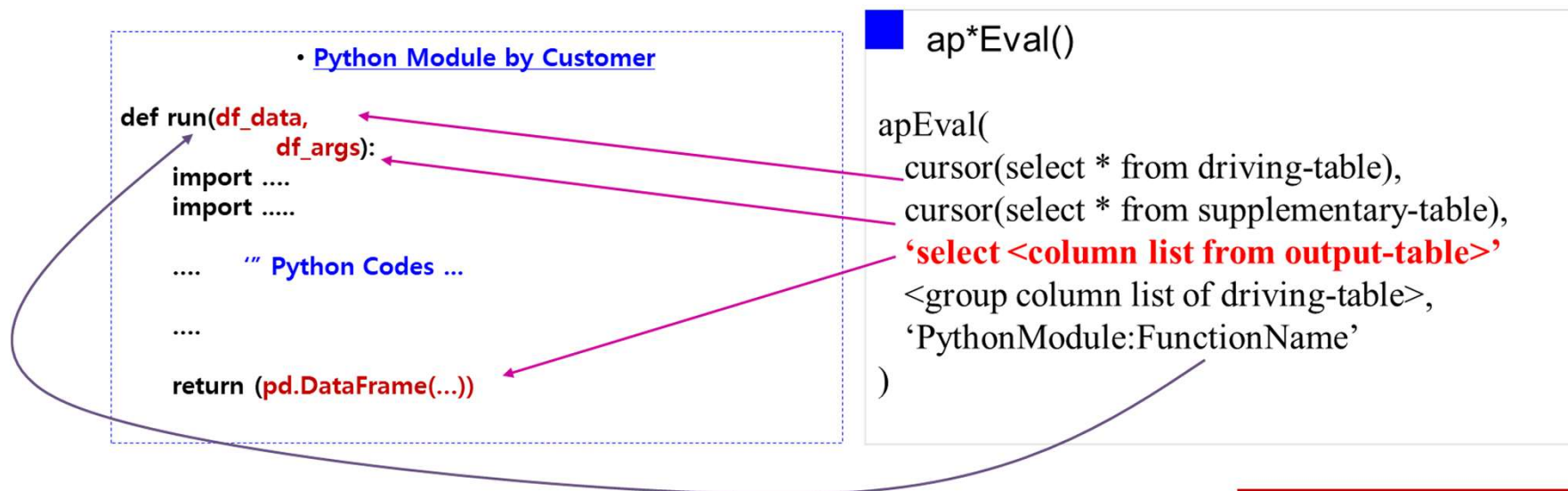
    socketHandler.close()
    logger.removeHandler(socketHandler)

    return df_melt
```

13 | Python Coding Style (#2)

The arguments of the Python module's start function are the same as the R function coding style, and are the same as two. The first one is the input data, and the second one is additional data. You can name it as you like. Unlike R, it does not support descriptive form.

- ※ In the case of parallel processing, the function name (ap(s)GroupEval) is different, and the parallel processing column names must be provided.
- ※ It is necessary to understand the SQL execution mechanism of **Oracle DatabaseTM**. The Oracle Optimizer creates a plan based on statistical information that is stored in the input and output format. Since Oracle does not automatically recognize the final Python output, you must inform it of this. (Red on the right)
- ※ If the output is complex, you can create it as an **Oracle View** and just write 'VIEW_NAME'. this is BDAE(TM) one of functionalities.
- ※ The part that automatically creates Python and R output as SELECT statements is on GitHub, and we will make it into a View.



14 | BDAETM Web Editor

BDAETM Web, the Web Editor part was imported and used as an open source package. It does not mean that you should develop it here. You can copy and paste what you developed in an existing development tool (R studio, Jupyter notebook, etc.).

※ If there is a line number in the case of a syntax error, you can use this, and the line number on the left is that line number.

```
R Modification / For enhancing your model.

R Module Name :  Script Type :  Description 

1 function(data, args1) {
2
3   library(xts)
4   library(quantmod)
5   library(RCurl)
6   library(logr)
7   library(xts)
8   library(quantmod)
9   library(RCurl)
10
11   sma1 = args1$SMA1
12   sma2 = args1$SMA2
13
14   data2 <- data.frame(data$open, data$high, data$low, data$close, data$volume, data$adjusted, row.names=data$row.names)
15   s1 <- as.xts(data2)
16   png(tf1 <- tempfile(fileext = ".png"), width=1920, height=1080)
17   taS <- sprintf("addMACD();addBBands();addSMA(%d);addSMA(%d,col='blue')", sma1[[1]], sma2[[1]])
18   chartSeries(s1['2016-03-10:'], up.col='red', dn.col='blue', theme='white', name="Samsung", TA=taS)
```

15 | BDAETM Python CRUD

In the case of **Python**, Module Name and Function Name (start function) must exist, and **BDAETM** executes both the case in DB and the file configuration, but DB is executed first.

That is, if it is not in DB, it means that the Module is searched in the PYTHONPATH location. (Customer sets it)

Python Script Modification / You can modify your model ...

Python Module Name : Python Function Name :
Description

```
1 from ultralytics import YOLO
2 import pandas as pd
3 import numpy as np
4 import os
5 os.environ['KMP_DUPLICATE_LIB_OK']='TRUE'
6 os.environ['PYTHONIOENCODING']='UTF-8'
7 import logging, logging.handlers
8 import sys
9 import base64
10 from os import listdir
11 from os.path import isfile, join
12 from StreamToLogger import StreamToLogger
13 import logging, logging.handlers
14
15 file_loc = []
16
17 def yolo_callback(x):
18     print("yolo callback called !")
19     file_loc.append(str(x))
```

16 | Python DataFrame, R data.frame

Since all RDBMS tables are 2D Tensors, we cannot escape from this. Therefore, both input and output values must be written in the [Python Pandas DataFrame](#) and [R data.frame](#) formats. (In the case of R, both data.frame and data.table outputs are supported.)

Input is done automatically by [BDAE\(TM\)](#), but the output must be created and returned by the analyst when returning. The case is an issue when entering, but if you do it as below, you can also use columns with mixed case.

If SELECT is "col1", .., [BDAE\(TM\)](#) will pass the mixed case form as it is.

Analysts should be aware of this part. Be careful when using Pandas DataFrame columns.

It would be better to change all columns to uppercase or lowercase and then work on them.

(※ df['col1'] is used in Python, R, etc., and case is important at this time.)

If you say **SELECT "Passenger" as Passenger**, it will be capitalized. This is not something that [BDAE\(TM\)](#) does, but rather something that the Data Dictionary API does, so you should be aware of it.

※ This part is not due to [BDAE\(TM\)](#), but is a characteristic of the [Oracle Database\(TM\)](#) SQL engine.

※ If you do SELECT * FROM .., the column case at the time the table was created will be applied as is.

17 | NA, NaN, Inf (#1)

In analysis, NA (Not Available), NaN (NULL), Inf (Infinity) are important. However, in **Oracle DatabaseTM**, NA and NaN are the same value NULL, and Inf is provided as follows. Infinity in R below is 1.0/0.0 is positive Infinity, -1.0/0.0 is negative Infinity.

1. In case of R

```
X <- c(1,2,3,NA,5)
Y <- c(1.1,-1.0/0,1.0/0,4.0,5.34)
```

```
df <- data.frame(X,Y)
df
```

2. In case of Python

```
import pandas as pd
import numpy as np
```

```
def returnNAN():
    df = pd.DataFrame([[ 'motor type',1, np.inf],
                       [np.nan, 2, 3.2],
                       ['RF', np.nan, 4.5]],
                      columns = list('abc'))

    return df
```

17 | NA, NaN, Inf (#2)

In Oracle, Infinity is returned as `binary_double_infinity`, but if you simply call this SQL in JAVA, an Overflow will occur, so be careful. (This is not an **Oracle DatabaseTM** or **BDAETM** problem.)

```
SELECT A, case when B=-binary_double_infinity
               then '-Infinity'
               when B=binary_double_infinity
               then '+Infinity'
               else TO_CHAR(B) end AS B
FROM (
  SELECT *
  FROM
  table(asEval(
    NULL,
    'SELECT 1 as A, 1.0 as B FROM dual',
    'R_infinity'))
);
```

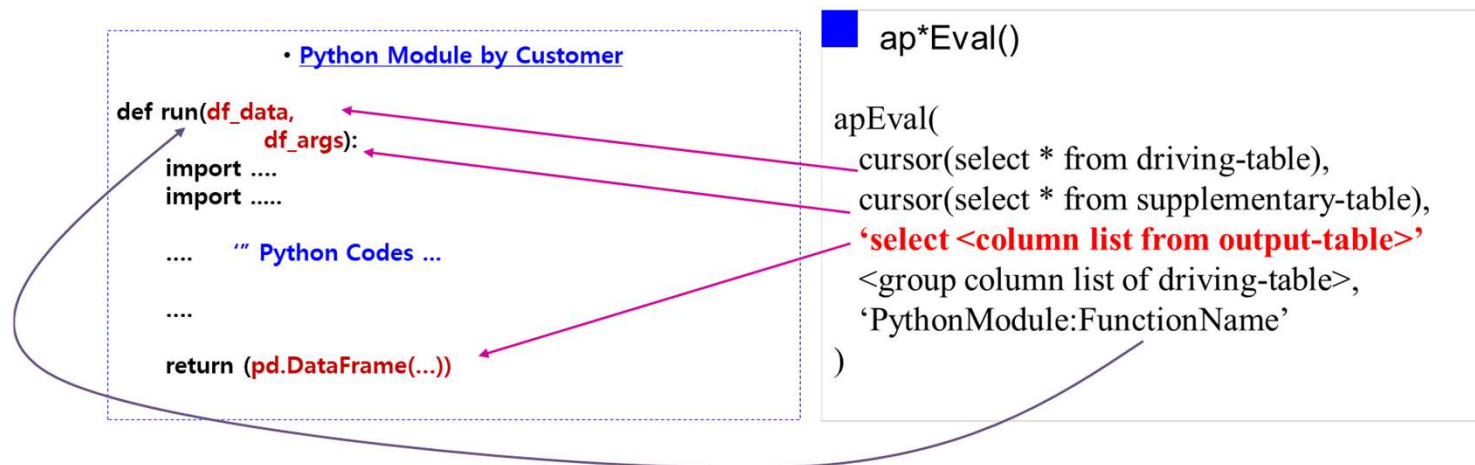

18 | Output DataFrame vs Oracle Type Matching

The red 'select <column list from output-table>' part on the right usually uses dual as follows. However, if there are many columns like below, it is annoying. Therefore, it is convenient to use it by making it a View. Just input 'V_OUTPUT1'. (※ This part is a function of **BDAE^(TM)**)

```
'SELECT CAST(''AA'' AS VARCHAR2(40)) EQP,  
      CAST(''AA'' AS VARCHAR2(40)) UNIT,  
      CAST(''AA'' AS VARCHAR2(40)) LOT,  
      CAST(''AA'' AS VARCHAR2(40)) WAFER,  
      CAST(''AA'' AS VARCHAR2(40)) RECIPE,  
      CAST(''AA'' AS VARCHAR2(40)) PARAM,  
      CAST(''AA'' AS VARCHAR2(100)) LOCATION  
FROM DUAL'
```



```
CREATE VIEW V_OUTPUT1 AS  
SELECT CAST(''AA'' AS VARCHAR2(40)) EQP,  
      CAST(''AA'' AS VARCHAR2(40)) UNIT,  
      CAST(''AA'' AS VARCHAR2(40)) LOT,  
      CAST(''AA'' AS VARCHAR2(40)) WAFER,  
      CAST(''AA'' AS VARCHAR2(40)) RECIPE,  
      CAST(''AA'' AS VARCHAR2(40)) PARAM,  
      CAST(''AA'' AS VARCHAR2(100)) LOCATION  
FROM DUAL;
```



19 | Real Time Logging

If there is a system error, you cannot see the logs in the Python or R modules.

(※ **BDAE**^(TM) is not executed one line at a time like the Python or R development environment.)

BDAE^(TM) transmits the error in the form of Description defined in ORA-20000 when there is a Python or R error.

If there is no error and you want to debug, it is better to use remote logging during development rather than doing it every time with a file. This part is included in GitHub.

In the case of Python, you can run the **LogServer.py**.

20 | Python Module Development (Input)

In a development environment without **BDAETM**, it is possible to use the Python **sqlalchemy** package. It does not work if you only install pip install sqlalchemy, cx_Oracle.

You need an Oracle Client, which you can download from the Oracle site.

The usage is as follows. With this, you can develop modules while checking columns.

```
import sqlalchemy
import pandas as pd
import pandas as pd
import cx_Oracle
import os
from sqlalchemy import create_engine

LOCATION = r"C:\Users\Admin\Downloads\instantclient-basic-windows.x64-23.8.0.25.04\instantclient_23_8"
os.environ["PATH"] = LOCATION + ";" + os.environ["PATH"]

oracle_connection_string = 'oracle+cx_Oracle://{username}:{password}@{hostname}:{port}'
DATABASE = "FREE"
SCHEMA = "rquser"
PASSWORD = "0000"

engine = create_engine(oracle_connection_string.format(username=SCHEMA,
                                                    password=PASSWORD, hostname='177.175.54.97', port='1521', database='FREE', ))

conn = engine.connect()
SQL = \
"""
SELECT ...
"""

df = pd.read_sql_query(SQL, conn)
```

21 | R Module Development (Input)

In a development environment without **BDAETM**, it is possible to use the **ROracle** package. ROracle installation is done using Github or the Internet, but the installation on Windows is on GitHub. Oracle Client is required, which is uploaded to the Cloud. (Installation file folder)
The usage is as follows. With this, development is performed by expecting **BDAETM** input.

```
# Oracle Installation Location
```

```
Sys.setenv("ORACLE_HOME"="/u01/app/oracle/product/12.2.0.1/db_1")
library(DBI)
library(ROracle)

driv <- dbDriver("Oracle")

connect.string <- paste(
  "(DESCRIPTION=",
  "(ADDRESS=(PROTOCOL = TCP)(HOST = 177.175.54.97)(PORT = 1521))",
  "(CONNECT_DATA=(SERVER = DEDICATED))",
  "(SERVICE_NAME = FREE)))", sep = "")

conn <- dbConnect(driv, username="rquser", password="0000", dbname=connect.string)
df <- dbGetQuery(conn,"SELECT * FROM FDC_TRACE WHERE ROWNUM < 10")
```

22 | Python Output to Oracle Data Type(#1)

In the end, I created a utility because it was difficult to manually do the SELECT ... FROM DUAL part of **BDAE^(TM)** SQL every time.

It is the Output part, and if you use the below, it will be created, and if you make it into a View again, it will be convenient.

```
def dtype_to_dbtype(typestr):
    return {
        'int64': lambda: '1',
        'object': lambda: "CAST('AA' AS VARCHAR2(40))",
        'float32': lambda: '1.0',
        'float64': lambda: '1.0',
        'datetime64[ns]': lambda: 'TO_TIMESTAMP(NULL)',
        'byte': lambda: 'TO_BLOB(NULL)'
    }.get(typestr, lambda: typestr + "not defined type.")()

def space_fill_underbar(column_name):
    return '_'.join(column_name.split(' '))
```

22 | Python Output to Oracle Data Type(#2)

Calling `generate_dual_select(df)` below will create a "SELECT ... FROM dual" statement.

```
def generate_dual_select(df):
    types = df.dtypes
    column_name_list = []
    column_type_list = []
    for i in range(len(types.index.tolist())):
        column_name_list.append(types.index.tolist()[i])
        column_type_list.append(str(types[types.index.tolist()[i]]))
        print("%-40s %s" %(types.index.tolist()[i], str(types[types.index.tolist()[i]])))

    sql = 'SELECT '
    last_index = len(column_name_list) - 1
    for i in range(last_index + 1):
        if i == last_index:
            sql = sql + ' ' + dtype_to_dbtype(column_type_list[i]) + ' ' + space_fill_underbar(column_name_list[i]) + '\nFROM dual'
        else:
            sql = sql + ' ' + dtype_to_dbtype(column_type_list[i]) + ' ' + space_fill_underbar(column_name_list[i]) + ',\n'

    return (sql)

sql = generate_dual_select(df)
print(sql)
```

23 | R Output to Oracle Data Type

In the case of R, calling `generate_dual_select(df)` below will create a "SELECT ... FROM dual" statement.

```
r_to_oracle_type <- function(r_type)
{
  switch(r_type,
    "character" = "VARCHAR2(4000)",
    "integer"   = "NUMBER(10)",
    "numeric"   = "FLOAT",
    "double"    = "FLOAT",
    "logical"   = "CHAR(1)",
    "Date"      = "DATE",
    "POSIXct"   = "TIMESTAMP",
    "factor"    = "VARCHAR2(4000)",
    "list"      = "CLOB",
    "unknown"   = "VARCHAR2(4000)",
    "VARCHAR2(4000)" # default
  )
}
```

```
generate_dual_select <- function(df)
{
  types <- sapply(df, function(col) class(col)[1])
  oracle_types <- sapply(types, r_to_oracle_type)

  select_parts <- mapply(function(col_name, ora_type) {
    paste0("CAST(NULL AS ", ora_type, ") AS ", col_name)
  }, names(df), oracle_types, USE.NAMES = FALSE)

  paste("SELECT", paste(select_parts, collapse = ", "), "FROM dual")
}

# 실행
query <- generate_dual_select(df)
cat(query)
```


24 | BDAE^(TM) SQL

* If you do not use Dynamic SQL statements, this means that you will be dependent on the customer's schema, which means that you will have to change the code every time to fit the customer.

However, BDAE^(TM) has been implemented to enable Dynamic SQL, which is a key differentiating feature. BDAE^(TM) has also been implemented to enable the same level as Oracle R Enterprise^(TM).

(※ The only RDBMS that fully supports Database Dictionary is Oracle Database^(TM).

The core of Dynamic SQL is that even if it becomes a SubQuery of a SubQuery .., it must transparently support the Dictionary of top-level column information. Oh, I'm talking about the API level. Not SQL execution ..)

* About Parallel processing

1. Oracle Optimizer checks the input query, output query, and statistical information and decides on parallel processing.
2. Therefore, Oracle Database^(TM) gives up parallel processing when it detects that an image (BLOB) or a string (CLOB) larger than 4000 bytes is included.
3. The same applies when using BDAE^(TM). However, there are many ways to circumvent this and use tricks to enable parallel processing, but this can only be done by using BDAE^(TM) rather than a regular SQL statement.

25 | DB-based development

Before creating **BDAE^(TM)** SQL, analysts should do the following:

Create a single module, a single function, without considering parallel processing.

Based on `import sqlalchemy`, let's first create a DataFrame with SQL and develop it.

SQL1 = "SELECT .. " # This SQL statement is used to create the query to be analyzed.

```
df_data = pd.read_sql_query(SQL1, conn)
```

There are two cases where the function of the Class you want to create requires arguments.

1) Combination of simple Scalars

SQL2 = "SELECT 1 as ARG1, 0.5 as ARG2 FROM DUAL"

2) Referenced data (reference tables, e.g. Reference tables for similarity measurement, inference tables)

SQL2 = "SELECT ... FROM TABLE "

```
df_args = pd.read_sql_query(SQL2, conn)
```

You can create a function for your purpose using those two, and register the module name and function name in **BDAE^(TM)**.

(※ You can work with R using ROracle, and the process is similar.)

26 | R Precautions

As mentioned earlier, R has two coding styles.

1. function() style : Input variables can be used with any name, explicit return like `return df`
2. In the description format, the input data name to be analyzed is `data`, and additional data is fixed as `args`.

You only need to write `df` once without a return statement, and any name other than `df` can be used.

However, both 1 and 2 return data.frame format and `stringsAsFactor=FALSE`.

The `stringsAsFactor` part will of course be familiar to R analysts.

If you create a data.frame without thinking, the Category column is saved as a Factor such as 1, 2, 3, etc.

3. The attributes of the data.frame returned like Python are generally strings, numbers, and DateTime. (converting - `as.character()`, `as.numeric()`, ..)

27 | Visualization 과 Serialization/DeSerialization

Both R and Python can create charts in three different forms.

1. Image Binary File Format (PNG, JPG, ...)
2. Image, String encoded in base64
3. Interactive Java Script String using plotly

When learning AI, there are many verification data results, various visualization charts, and parameters, and you want to see them. Since the SQL statement returns only once, you can put all of these in a DataFrame, data.frame, and that's it.

In particular, in the case of Model, it must be serialized, so it must be stored as an Oracle BLOB, and when performing real-time inference during later operation, it must be deserialized and used again.

It is quite easy and possible using **BDAE^(TM)**.

28 | BDAE^(TM) does not use storage space at runtime..

BDAE^(TM) is executed in memory, without any separate storage space, and after analysis via Python and R engines, everything is done in memory.

There are two ways to save the return results of BDAE^(TM) in a DB: one is to have the analyst create a separate session within Python or R, and the other is to use the following universal database techniques.

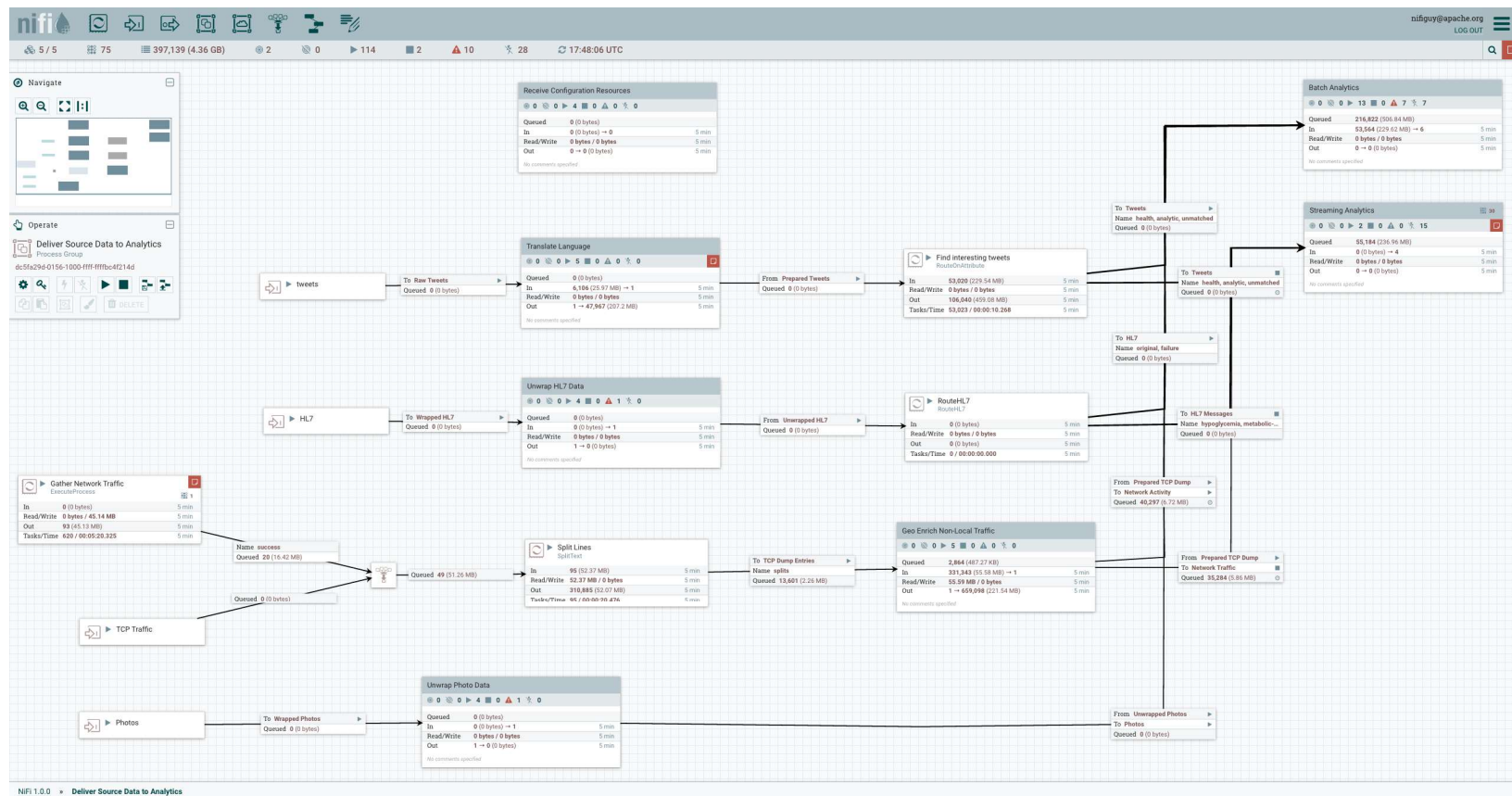
```
CREATE TABLE RESULT_TEMPORARY_TABLE AS SELECT ... ;  
INSERT INTO RESULT_PERMANENT_TABLE SELECT ... ;
```

※ Problems with Oracle In-Database Programs and Solutions with BDAE^(TM)

1. Since it is managed separately from the SGA, PGA area of Oracle Database^(TM), if Python or R internal code excessively uses a lot of memory, it can kill the server where the Oracle Instance is located. (Out of Memory in terms of OS)
2. Because we have seen this happen in real Oracle R Enterprise^(TM) operations, BDAE^(TM) is designed to set the maximum available memory for a single session memory and monitor it during execution (every 10 seconds for that session).

29 | BDAE™ and Apache NIFI™ Integration

Using Apache NIFI™ and BDAE™ together allows you to configure batch jobs with a more stable workflow. In particular, it is effective and can be organized neatly when it is SQL-based rather than file-based.



Utilizing BDAE^(TM)

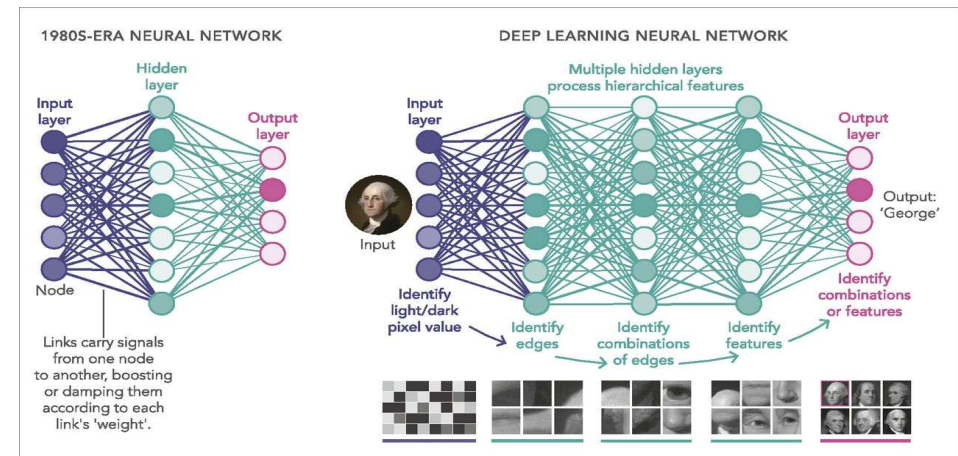
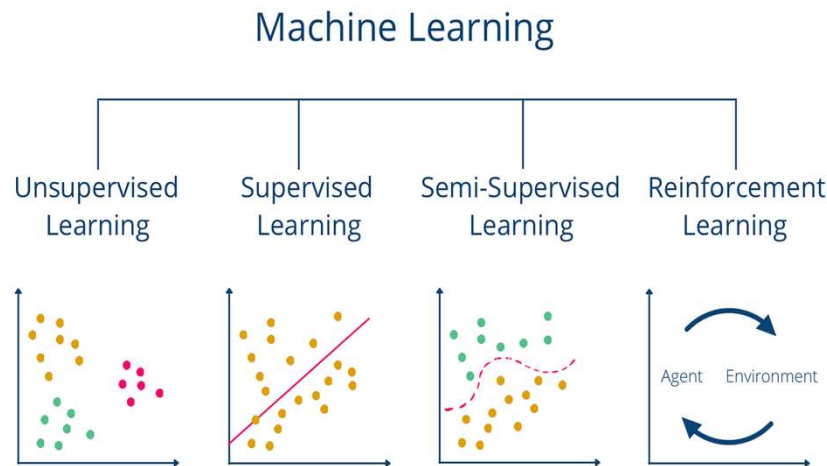
1 | 1. Various AI tasks performed with Python and R (#1)

The initial development goal of **BDAE^(TM)** was to find a way to easily apply the latest algorithms to existing solutions in manufacturing systems, such as SPC, FDC, and YMS.

In particular, there is a doubt about whether it is advisable to perform defect analysis, SPC, descriptive statistics, ANOVA, etc. through pattern recognition in backends such as JAVA, or to purchase separate expensive analysis products.

In actual high-tech fields, raw data is commonly in **Oracle Database^(TM)**, and R or Python scripts are run on a separate application server.

However, BDAE(TM) solves problems such as algorithm source management, parallel processing, and best performance on a separate server without massive data movement.



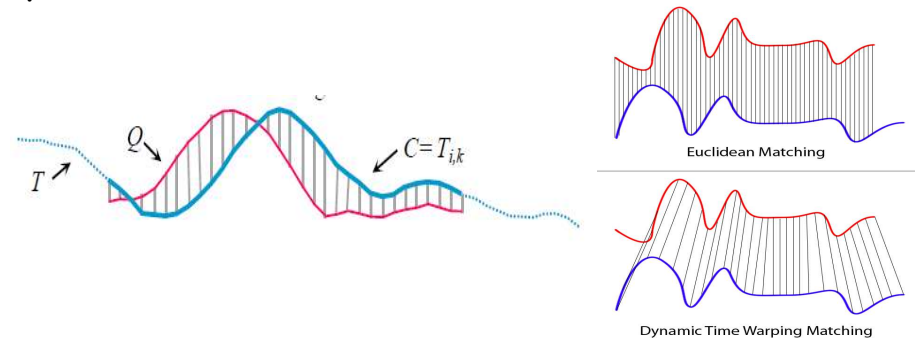
2 | Various AI tasks performed with Python and R (#2)

If you perform real-time analysis quickly and receive the results as a universal SQL query result like the one below, the application has a very simple structure.

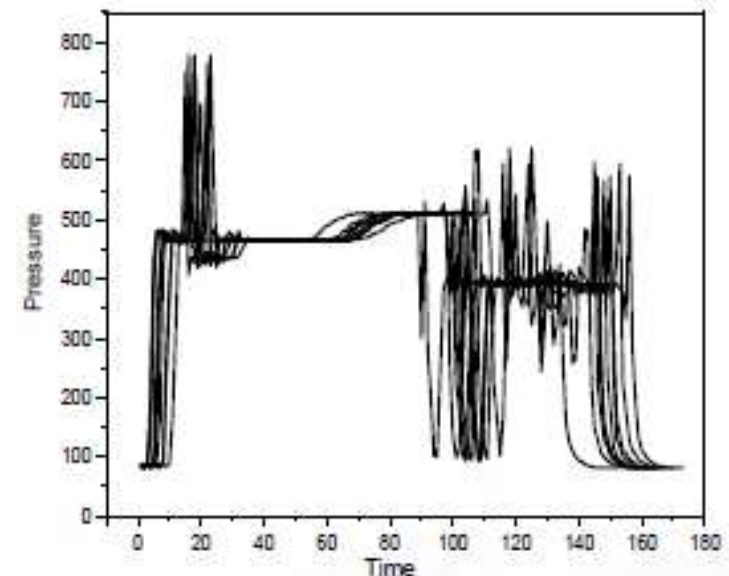
※ The second argument of **BDAETM** is used as
Query Data for comparison,
Model Data for inference,
or Argument for functions used in Python, R.

```
SQL> SELECT /*+ parallel(20) */
2 FROM TABLE (apGroupEvalParallel (
3     cursor (
4         SELECT *
5         FROM TRACE_DATA
6         WHERE EQP_ID = 'EPS001'
7             AND LOT_ID = 'LOT001'
8             AND ETC = '.....'
9     ),
10     cursor(SELECT * FROM GOLDEN_EQUIPMENT ...),
11     'SELECT CAST("A" AS VARCHAR2(40)) PARAMETER_ID,
12         1.0 SIMILARITY FROM DUAL',
13     'EQP_ID, LOT_ID, ....',
14     'DefectUtil:FastDTW');
```

PARAMETER_ID	SIMILARITY
RF_POWER_1	2.23
O2_PUMP_1	0.5
Ch1_TEMP_1	2.1
.....
.....

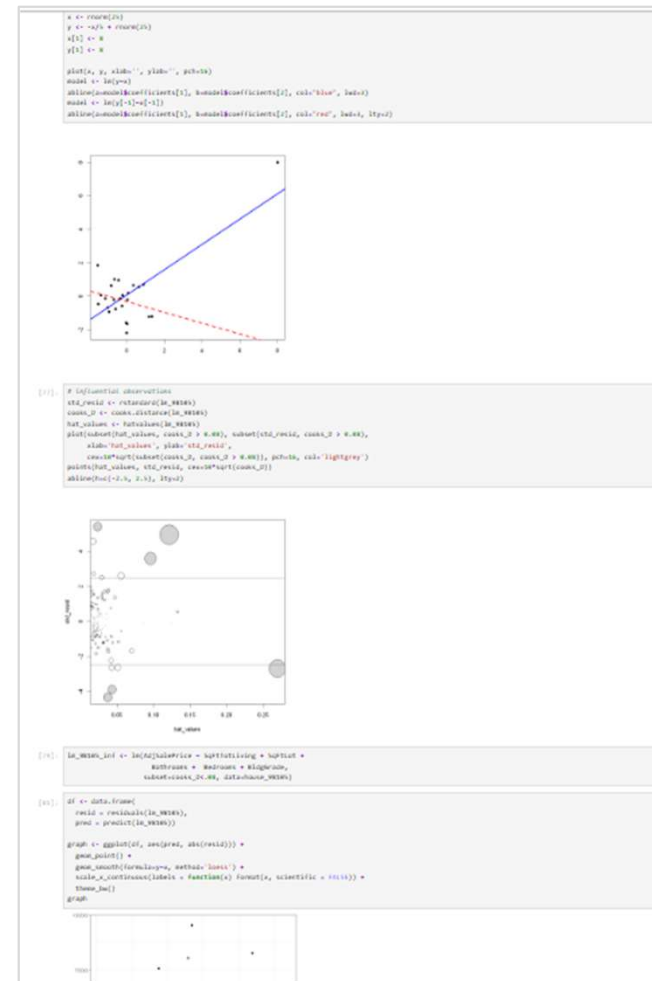
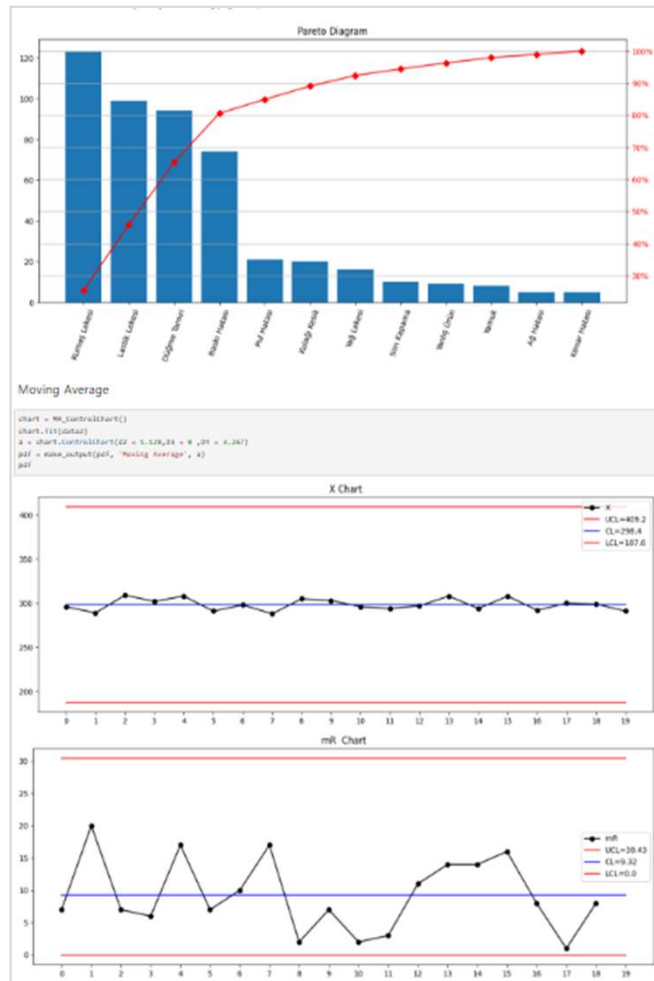


Pattern Matching Algorithms



3 | Batch and integrate analytics tasks

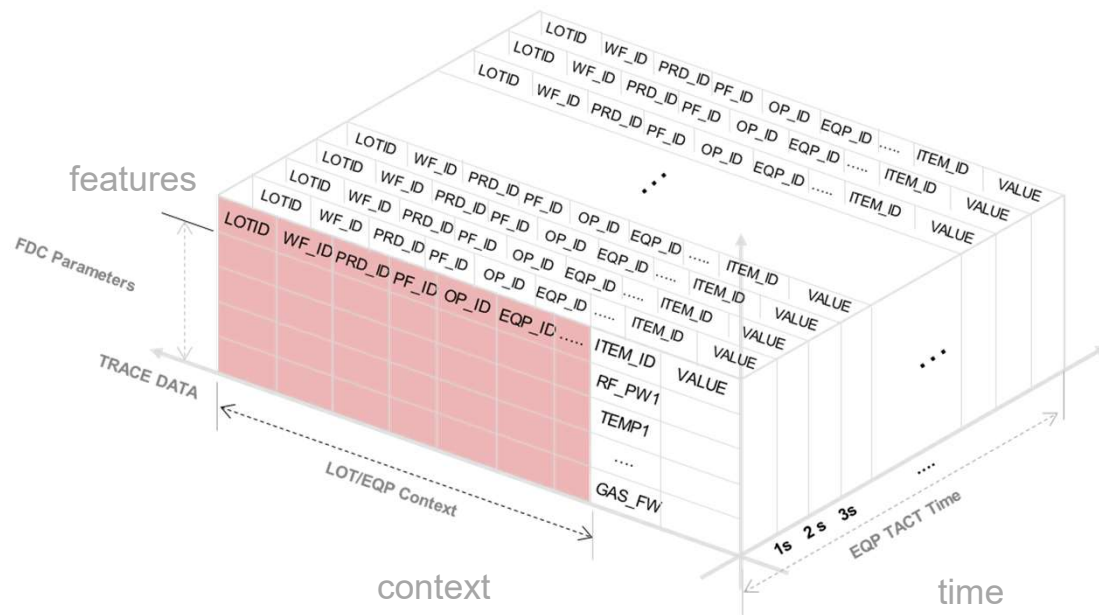
If you wrap the SQL statement above in Python or R and create a class, you can integrate various analysis tasks and perform them at once, but parallel processing is possible, and the results can also be received at once.



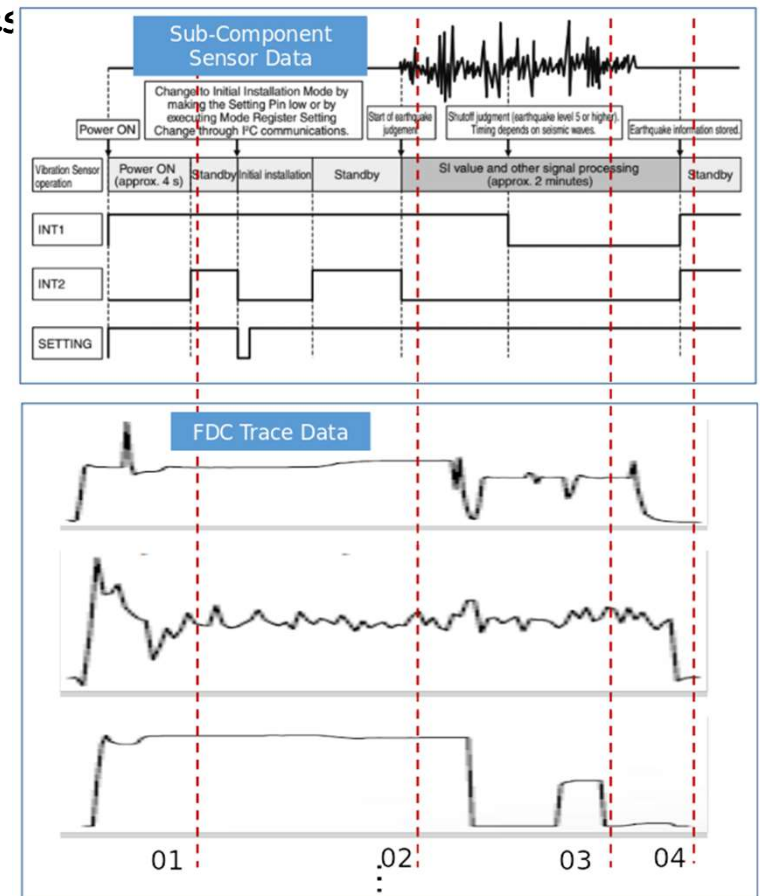
4 | Parallel distributed processing (#1)

When grouping large tables and applying the same algorithm at once to receive results, using parallel processing in Python or R can result in excessive reusability, performance, and memory usage.

This parallel processing part is left to Oracle In-Database, and the role of **BDAE^(TM)** is to make Python and R modules have a simple form that does not consider parallel processing, and the above problems are solved.



Trace Data per 1 LOT/1 EQP



4 | Parallel distributed processing (#2)

The following query is an example of parallel distributed processing of a large table in units of group columns in red. The Oracle Hint (`/*+ parallel(20) */`) section divides it into 20 and parallelizes it, and the Python module then takes the form of a simple module that does not consider parallel processing. **BDAETM** makes this possible.

```
SELECT /*+ parallel(20) */
FROM table(apGroupEvalParallel(
  CURSOR(
    WITH TARGET_TBLE AS
    (
      SELECT * FROM FDC_TRACE
      WHERE 1=1
      AND EQP_ID='EQP-200'
      AND UNIT_ID='UNIT-02'
    )
    SELECT EQP_ID, UNIT_ID, LOT_ID, WAFER_ID, RECIPE, PARAM_ID,
      (VALUE - (AVG(VALUE) OVER (PARTITION BY PARAM_ID)))
      / (STDDEV(VALUE) OVER (PARTITION BY PARAM_ID)) AS NORMALIZED_VALUE
      FROM TARGET_TBLE
  ),
  NULL,
  'SELECT CAST(''A'' AS VARCHAR2(40)) EQP_ID,
    CAST(''A'' AS VARCHAR2(40)) UNIT_ID,
    CAST(''A'' AS VARCHAR2(40)) LOT_ID,
    CAST(''A'' AS VARCHAR2(40)) WAFER_ID,
    CAST(''A'' AS VARCHAR2(40)) RECIPE,
    CAST(''A'' AS VARCHAR2(40)) RESULT
    FROM DUAL',
  'EQP_ID,UNIT_ID,LOT_ID,WAFER_ID,RECIPE,PARAM_ID',
  'Standardization:normalize'));
```

LOTID	WF_ID	PRD_ID	OP_ID	EQP_ID	UNIT_ID	...	TIME	ITEM_ID	VALUE
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_001	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_002	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_003	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_004	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_005	653
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_006	78
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_007	112
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_008	8
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_009	212
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_010	32
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_011	12
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_012	22
...									
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_683	112.22
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_684	0.212

LOTID	WF_ID	PRD_ID	OP_ID	EQP_ID	UNIT_ID	...	TIME	ITEM_ID	VALUE
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_001	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_002	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_003	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_004	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_005	653
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_006	78
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_007	112
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_008	8
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_009	212
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_010	32
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_011	12
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_012	22
...									
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_683	112.22
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_684	0.212

:

LOTID	WF_ID	PRD_ID	OP_ID	EQP_ID	UNIT_ID	...	TIME	ITEM_ID	VALUE
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_001	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_002	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_003	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_004	23
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_005	653
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_006	78
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_007	112
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_008	8
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_009	212
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_010	32
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_011	12
L1	W1	P1	OP1	EQP1	UNIT1	...	T1	ITEM_012	22
...									
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_683	112.22
L1	W1	P1	OP1	EQP1	UNIT1	...	T3600	ITEM_684	0.212



5 | PreProcessing

If Python or R performs preprocessing of a large portion of input data, memory and performance issues arise. **BDAE^(TM)** can accept and preprocess complex SQL statements such as the following, and this is because **Oracle Database^(TM)** has the most advanced SQL engine.

```
SELECT eqp_id, recipe_id, ..., time, parameter_name, sma, ema
FROM (
    SELECT eqp_id, recipe_id, time, ..., parameter_name, parameter_value
    FROM trace_data
    WHERE 1=1
        AND time between ... and ...
        AND step_seq = '...'
    ...
) a
MODEL
PARTITION BY (a.parameter_name, 2 / (1 + count(*) over (partition by a.parameter_name))
              smoothing_constant )
DIMENSION BY (row_number() over (partition by a.parameter_name order by a.time) rn)
MEASURES (a.time, a.parameter_value, sma, 0 ema)
(
    ema[1] = a.parameter_value[1],
    ema[rn > 1] order by rn = ( cv(smoothing_constant) * (parameter_value[cv()] - ema[cv() - 1]) ) + ema[cv() -
1]
)
ORDER BY eqp_id, a.recipe_id, ..., a.time, a.parameter_name;
```

6 | Smart Factory (#1)

Based on existing solutions such as statistical quality analysis, anomaly detection, SPC, etc., you can easily embed flexible functions into existing solutions as SQL statements without additional cost by using various packages provided by Python and R in **BDAE^(TM)**.

※ **BDAE^(TM)** is implemented as a SQL statement and runs in memory, so it can be added to any solution.

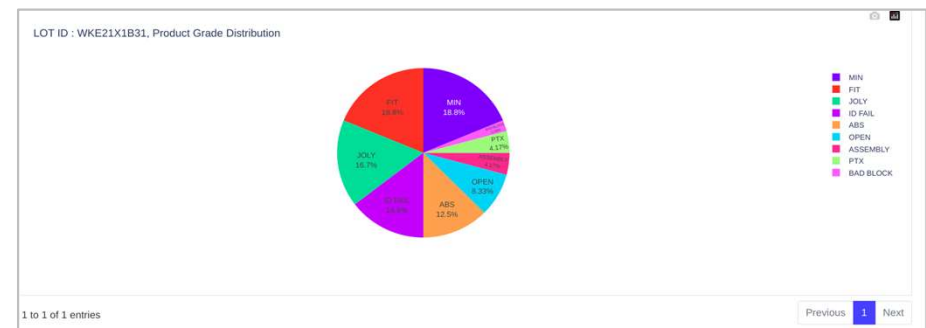


6 | Smart Factory (#2)

This level of functionality can be implemented in about 5 lines.

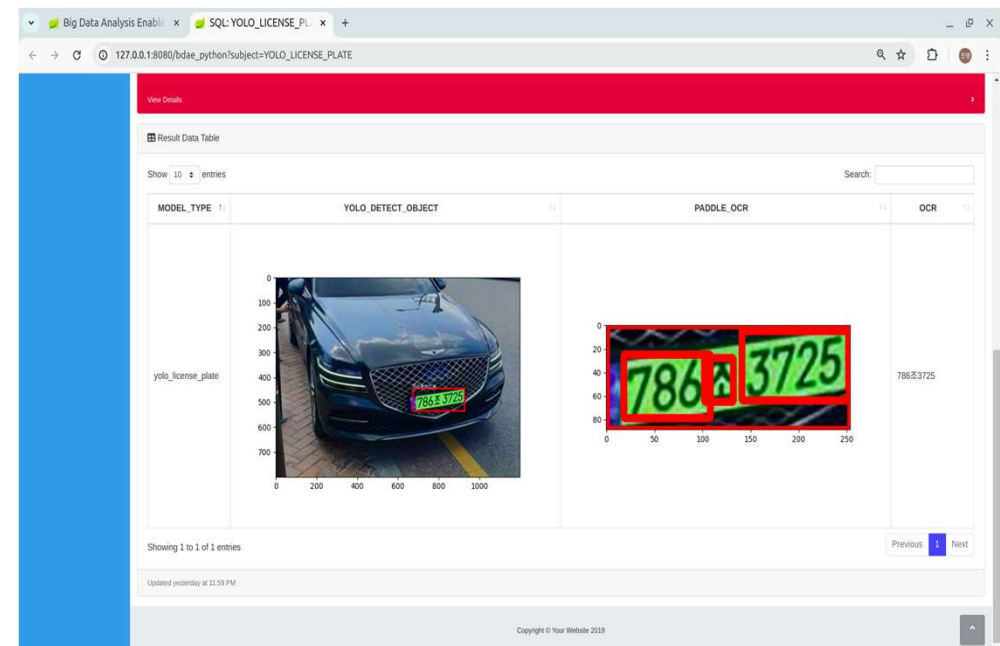
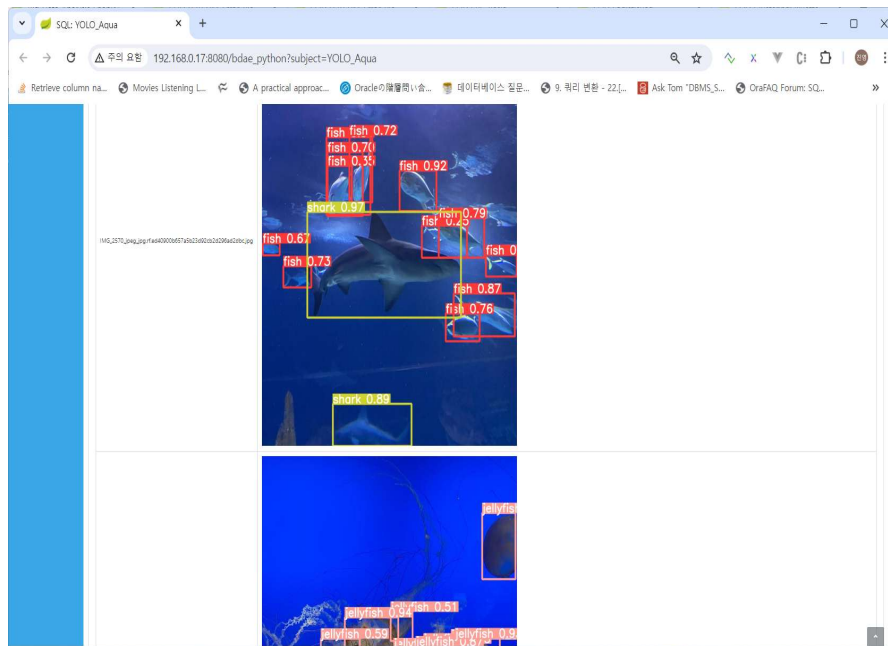
The plotly package exists for both Python and R and makes charts interactive.

```
WITH LOT_SUM_ONE AS (  
  SELECT LOT_ID,  
    CASE WHEN PROD_GRADE = 9 THEN 'BAD BLOCK'  
      WHEN PROD_GRADE = 8 THEN 'ASSEMBLY'  
      WHEN PROD_GRADE = 7 THEN 'PTX'  
      WHEN PROD_GRADE = 6 THEN 'ID FAIL'  
      WHEN PROD_GRADE = 5 THEN 'OPEN'  
      WHEN PROD_GRADE = 4 THEN 'ABS'  
      WHEN PROD_GRADE = 3 THEN 'JOLY'  
      WHEN PROD_GRADE = 2 THEN 'FIT'  
      WHEN PROD_GRADE = 1 THEN 'MIN'  
    ELSE  
      'N/A'  
    END PROD_GRADE_DESC  
  , COUNT(PROD_GRADE) CNT FROM table (  
    productExplodeEvalClob(cursor(  
      SELECT *  
      FROM LOT_SUM  
      WHERE LOT_ID = 'WKE21X1B31'  
        AND MACHINE_ID = '48PARA-03' ...DURABLE_ID = 'Z718'))))  
  GROUP BY LOT_ID, PROD_GRADE  
)  
SELECT *  
FROM table(apTableEval(  
  cursor(  
    SELECT * FROM LOT_SUM_ONE  
  ),  
  NULL,  
  'XML',  
  'LOTSUM_ERR_PIE:display'))
```



7 | Deep Learning Inference

In places where **BDAE^(TM)** is not installed for learning cost- or GPU-dependent algorithms, you can learn the model and use it for inference in real time using **BDAE^(TM)**.
(This is because the model may be different for each facility and product, and **BDAE^(TM)** supports DeSerialization.)



8 | Step by Step

BDAE^(TM) execution starts with a SQL statement, which can be called from any Oracle session-based DB tool or application.

You can create anything with **BDAE**^(TM), such as an image or document, and pass it as the result of a SQL query. You can also DeSerialize a model stored in a BLOB to make inferences, or you can separately index it.

This section 8 describes the process of creating Python, R modules, and **BDAE**^(TM) SQL to call them.

※ **BDAE**^(TM) SQL is not a special SQL. It is a kind of table function in Oracle Database, but it is a little special.

8 | R ML Example (#1 Select SQL Interface function)

R provides four functions. The most commonly used ones are `asTableEval()` and `asGroupEval()`. `asEval()` can be used for testing purposes, and `asRowEval` is useful for doing something for each defined number of rows.

The arguments of these functions are almost similar and intuitive.

※ The **Python** function is `ap` instead of `as`, and the functionality is exactly the same, and it also provides four functions.

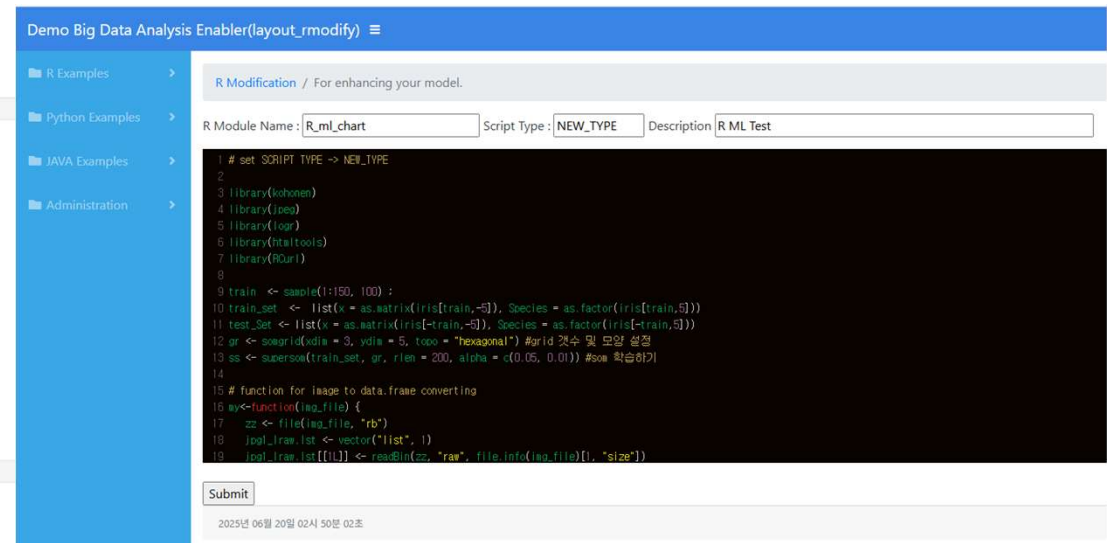
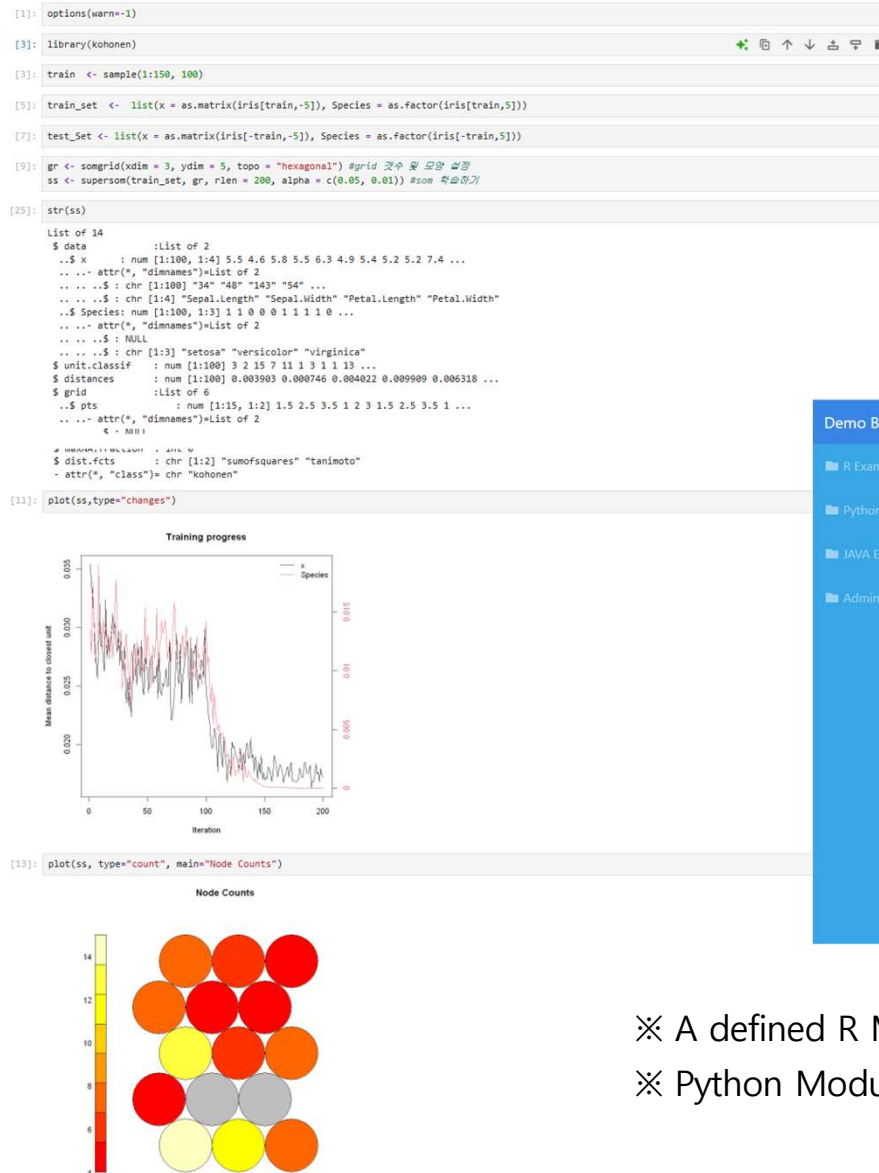
SQL Interface function	Description
<code>asEval()</code>	Provide No Data into R Module
<code>asTableEval()</code>	Provide Data into R Module
<code>asRowEval()</code>	Provide Splitted Data per rows into R Module
<code>asGroupEval()</code>	Provided Splitted Data per grouped into R Module

8 R ML Example (#2 Development Phase)

Typically, analysts develop interactively using R studio or Jupyter notebook during the development phase..

After completing this development work, if you want to run it on BDAE(TM),

1. Organize and move the source. (Copy & Paste) using **BDAE(TM)** Web Editor.
- 2) Name the R module. Save it.



※ A defined R Module can be reused.

※ Python Module is similar, but you also need to enter the function name.

8 | R ML Example (#2 Development Phase)

When transferring R source in the development stage to **BDAE^(TM)**, the output must be considered. Since it is eventually output in the form of a table, all data must be converted to the form of R's data.frame and transferred. Therefore, some source modification is necessary, but since this part is standardized, there will be no difficulty if you create and use reusable functions. In order to put all images in a data.frame, create a function (my) and insert it.

```
my<-function(img_file) {  
  zz <- file(img_file, "rb")  
  jpg1_lraw.lst <- vector("list", 1)  
  jpg1_lraw.lst[[1L]] <- readBin(zz, "raw", file.info(img_file)[1, "size"])  
  close(zz)  
  df <- data.frame(name=img_file,stringsAsFactors=FALSE)  
  df$blob <- jpg1_lraw.lst  
  unlink(img_file)  
  return (df)  
}
```

```
name <-c("/tmp/kohonen01.jpg","/tmp/kohonen02.jpg","/tmp/kohonen03.jpg","/tmp/kohonen04.jpg", "/tmp/kohonen05.jpg")
```

```
jpeg(name[1])  
plot(ss,type="changes")  
dev.off()  
jpeg(name[2])  
plot(ss, type="count", main="Node Counts")  
dev.off()
```

8 | R ML Example (#2 Development Phase)

. After merging each data.frame based on row using rbind, write down the final data.frame name.

```
df = my(name[1])  
for (i in 2:length(name)) {  
  df_tmp = my(name[i])  
  log_print(name[i])  
  df = rbind(df, df_tmp)  
}
```

df

. Save the above source as R_ml_chart using **BD&AETM** Web Editor. (Copy & Paste)

. To make the call, we now need to create **BD&AETM** SQL.

The data.frame created earlier consists of a file name and an image binary, so the output of **BD&AETM** is

```
SELECT CAST(NULL AS VARCHAR2(40) Name, TO_BLOB) IMG FROM DUAL
```

You can create it simply with . For complex data.frame, you can use the output function in this document. **(generate_dual_select)**

8 | R ML Example (#3 BDAE SQL Creation Phase)

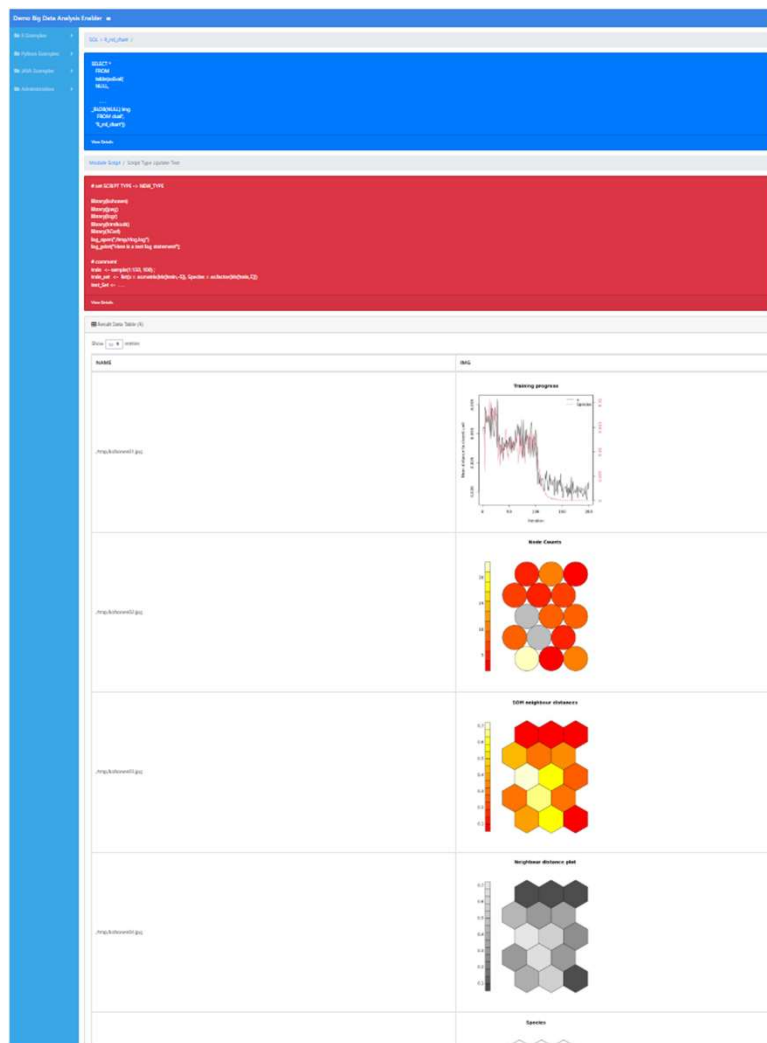
Finally, the SQL using asEval() is as follows.

1. Of the four functions, since we do not receive data from **BDAETM**, select asEval()
2. there is no additional data (e.g. arguments for R), so NULL,
3. the output part is the image name and the image binary (JPG), so red,
4. and finally, enter R_ml_chart, the name of the R module containing the algorithm.

```
SELECT *  
FROM  
table(asEval(  
  NULL,  
  'SELECT CAST(NULL AS VARCHAR2(40)) name, TO_BLOB(NULL) img FROM dual',  
  'R_ml_chart'))
```

※ In the case of Python, the start function name, such as P_ml_chart:display, must be separated by a comma.

8 | R ML Example (#4 Review the result)



◁ SQL executed (run)

◁ Used R Module

◁ SQL Result

8 | R ML Example (#4 Review the result)

For example, if you call SQL with the DBeaver(TM) tool, you will see the following results. You can see the chart image right away in the DBeaver(TM) tool, knowing that it is an image.

The screenshot displays the DBeaver 25.1.0 interface. The main window shows a SQL script in the editor:

```
SELECT * FROM table(asEval(
  NULL,
  'SELECT CAST(NULL AS VARCHAR2(100)) name, TO_BLOB(NULL) img
  FROM dual',
  'R_ml_chart'));
```

The results pane shows a table with 5 rows:

NAME	IMG
/tmp/kohonen01.jpg	[BLOB]
/tmp/kohonen02.jpg	[BLOB]
/tmp/kohonen03.jpg	[BLOB]
/tmp/kohonen04.jpg	[BLOB]
/tmp/kohonen05.jpg	[BLOB]

The right pane shows a chart titled "Training progress" with the following data:

Iteration	Mean distance to closest unit (x)	Species (Species)
0	0.035	0.015
50	0.030	0.010
100	0.025	0.005
150	0.020	0.002
200	0.020	0.002

8 | Python Example

Python is not much different from R.

R is an excellent language for data interpretation, and Python has already established itself as a main language for AI due to its fast performance and GPU usage.

From the BDAE(TM) perspective, Python is also recommended because of its good exception handling techniques.

8 Python Smart Factory concerned Example (#1 Table Example)

The LOT_SUM table originally looks like this, and what's special is the existence of the MAPPING column. The Grade part of the PRODUCTS in the LOT exists as a CLOB column.

QTY	OPERATOR	OUT_QTY	PROC_OPER_	PROD_SPEC_ID	MAPPING	TKIN_TIME	TKOUT_TIME
66	신혜선	654	M030	RAK37BHFAMA1-AGA	9,8,8,7,7,9,7,7,8,7,8,8,7,9,9,7,7,9,7,8	2020-08-06 06:32:38.000	2020-08-06 07:03:5
124	정은경	596	M020	RAK37BHFAMA1-AGA	8,9,8,9,9,7,9,7,7,7,7,8,8,7,8,9,9,9,8,8	2020-08-02 09:18:58.000	2020-08-02 09:50:1
5	최효주	43	M030	K9AFGD8H0A-T040MD1	7,1,2,8,5,5,3,7,3,3,9,2,2,5,6,2,4,6,6,5	2020-07-04 17:28:13.000	2020-07-04 17:59:1
3	한시우	45	M020	RAK37BHFAMA1-AGA	9,8,5,1,2,4,1,2,1,4,2,1,3,6,4,6,2,5,3,3	2020-07-11 04:20:25.000	2020-07-11 04:52:0
2	정은경	46	M030	K9AFGD8H0A-T040MD1	3,2,5,5,4,6,3,3,3,5,8,5,2,3,4,3,6,6,5,5	2020-07-11 13:56:08.000	2020-07-11 14:26:3
6	최효주	42	M020	K9AFGD8H0A-T040MD2	8,7,1,7,9,7,3,6,6,5,6,6,5,3,3,5,2,3,2,6,6	2020-08-14 23:47:44.000	2020-08-15 00:18:5
5	정은경	43	M030	RAK37AJDAMC1-AGB	9,4,8,6,8,3,6,6,3,5,2,7,3,2,6,1,3,2,5,1,2	2020-09-18 06:58:47.000	2020-09-18 07:30:4
25	신혜선	215	M030	K9AFGD8H0A-T040MD1	9,8,8,8,7,7,8,9,7,7,9,7,8,7,8,9,2,5,8,8,9	2020-08-29 11:49:45.000	2020-08-29 12:22:5
43	정은경	197	M020	K9AFGD8H0A-T040MD1	9,7,9,8,7,8,8,8,9,7,9,7,9,8,8,9,8,8,7,7,9	2020-09-13 23:57:11.000	2020-09-14 00:28:3
3	최효주	45	M030	K9AFGD8H0A-T040MD2	4,5,9,5,7,5,1,3,1,5,5,3,2,1,3,1,6,1,6,3,3	2020-09-07 12:42:59.000	2020-09-07 13:17:4
7	정은경	41	M030	RAK37AJDAMC1-AGB	9,3,7,7,2,3,9,3,4,1,5,2,3,7,5,5,5,4,2,9,2	2020-07-11 08:30:53.000	2020-07-11 09:05:0
3	정은경	717	M020	K9AFGD8H0A-T040MD2	2,1,8,6,2,7,3,1,1,6,2,2,1,2,2,1,2,3,2,6,6	2020-09-17 08:37:08.000	2020-09-17 09:09:2
27	정은경	213	M030	RAK37BHFAMA1-AGA	7,8,9,9,8,9,7,7,8,8,8,2,9,9,7,7,8,9,8,8,2	2020-07-19 06:40:07.000	2020-07-19 07:12:1
119	신혜선	601	M030	K9AFGD8H0A-T040MD2	7,7,7,9,8,8,8,8,8,8,9,9,9,8,7,9,8,8,9,9	2020-09-17 01:15:02.000	2020-09-17 01:48:2
25	정은경	215	M020	K9AFGD8H0A-T040MD2	7,7,9,8,8,8,9,8,8,7,9,7,8,8,5,6,8,9,4,9	2020-07-30 08:59:27.000	2020-07-30 09:30:0
11	최효주	229	M030	K9AFGD8H0A-T040MD1	8,7,9,7,2,8,3,9,8,3,8,2,2,9,3,6,7,3,5,1,6	2020-09-18 11:34:03.000	2020-09-18 12:08:4
11	한시우	229	M020	K9AFGD8H0A-T040MD1	8,9,9,2,7,6,9,7,9,7,5,5,6,7,1,2,5,3,3,7	2020-09-20 22:32:46.000	2020-09-20 23:04:0
37	최효주	203	M020	RAK37AJDAMC1-AGB	7,7,9,7,7,8,8,8,7,9,9,9,9,8,8,7,7,8,2,8	2020-09-30 13:15:32.000	2020-09-30 13:50:2
5	정은경	43	M030	RAK37AHDAPC1-APA	7,8,9,2,3,5,5,7,3,5,5,1,3,2,3,3,6,3,4,6	2020-07-27 22:39:59.000	2020-07-27 23:11:1
4	최효주	44	M020	K9AFGD8H0A-T040MD2	7,9,8,2,4,3,3,2,1,1,3,2,5,2,3,2,5,6,1,6	2020-07-04 16:39:30.000	2020-07-04 17:11:3

Let's assume we want to draw a PIE chart of Grade for a specific LOT below.

NG_QTY	OPERATOR	OUT_QTY	PROC_OPER_	PROD_SPEC_ID	MAPPING	TKIN_TIME	TKOUT_TIME
6	한시우	42	M030	K9AFGD8H0A-T040MD1	8,8,9,4,7,6,4,7,4,6,3,4,2,2,6,4,5,5,6,2,	2020-07-04 20:08:45.000	2020-07-04 20:43:22.000

8 | Python Smart Factory concerned Example (#2 Development Phase)

Drawing Pie Chart in Python is very simple. It only takes 2~3 lines.
(It is not much different in R. R also provides the same package.)

What is needed is that the information of the corresponding LOT is passed in df. This Pie Chart is interactive because it is automatically generated with JavaScript. (When the Mouse moves...)

However, it is passed in the form of <div> for the web server (backend).

```
import pandas as pd
import plotly.express as px
import plotly.offline as py

def display(df):
    fig = px.pie(df, values='CNT', names='PROD_GRADE_DESC', title='LOT ID : ' + df['LOT_ID'][0] + ', Product Grade Distribution')
    fig.update_traces(textposition='inside', textinfo='percent+label')
    total = py.offline.plot(fig, output_type='div')
    return (total)
```

Now, let's save the module name as LOTSUM_Error_Pie and the function name as display using the BDAE Web Editor.

8 | Python Smart Factory concerned Example (#3 EDA)

Preprocessing is required. Originally, LOT_SUM's Mapping is a CLOB column, and this needs to be resolved based on Rows. Therefore, it is easy to draw a chart by making it look like this.

	lot_id	prod_grade_desc	cnt
0	WKE21X1B31	ASSEMBLY	2
1	WKE21X1B31	BAD BLOCK	1
2	WKE21X1B31	ABS	6
3	WKE21X1B31	PTX	2
4	WKE21X1B31	ID FAIL	7
5	WKE21X1B31	JOLY	8
6	WKE21X1B31	FIT	9
7	WKE21X1B31	OPEN	4
8	WKE21X1B31	MIN	9

Is it possible to dynamically create a Mapping table based on rows?

You can create a separate table using PL/SQL, but it will take up a lot of space.

So what should we do?

8 | Python Smart Factory concerned Example (#4 Preprocessing)

BDAE^(TM) is equipped with a preprocessing module and can be created dynamically as follows. It is created dynamically and in this case, it uses purely Oracle Native functions without calling Python or R.

Results 1 x

SQL: `SELECT * FROM LOT_SUM WHERE LOT_ID = 1` Enter a SQL expression to filter results (use Ctrl+Space)

	NG_QTY	A-Z OPERATOR	123 OUT_QTY	A-Z PROC_OPER_	A-Z PROD_SPEC_ID	MAPPING	TKIN_TIME	TKOUT_TIME
1	6	한시우	42	M030	K9AFGD8H0A-T040MD1	8,8,9,4,7,6,4,7,4,6,3,4,2,2,6,4,5,5,6,2,	2020-07-04 20:08:45.000	2020-07-04 20:43:22.000

Results 1 x

SQL: `SELECT * from table (productExplodeEvalCL` Enter a SQL expression to filter results (use Ctrl+Space)

	OUT_QTY	A-Z PROC_OPER_ID	A-Z PROD_SPEC_ID	A-Z PROD_GRADE	123 PROD_IDX	A-Z TKIN_TIME	A-Z TKOUT_TIME
1	42	M030	K9AFGD8H0A-T040MD1	8	1	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
2	42	M030	K9AFGD8H0A-T040MD1	8	2	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
3	42	M030	K9AFGD8H0A-T040MD1	9	3	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
4	42	M030	K9AFGD8H0A-T040MD1	4	4	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
5	42	M030	K9AFGD8H0A-T040MD1	7	5	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
6	42	M030	K9AFGD8H0A-T040MD1	6	6	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
7	42	M030	K9AFGD8H0A-T040MD1	4	7	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
8	42	M030	K9AFGD8H0A-T040MD1	7	8	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
9	42	M030	K9AFGD8H0A-T040MD1	4	9	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
10	42	M030	K9AFGD8H0A-T040MD1	6	10	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
11	42	M030	K9AFGD8H0A-T040MD1	3	11	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
12	42	M030	K9AFGD8H0A-T040MD1	4	12	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
13	42	M030	K9AFGD8H0A-T040MD1	2	13	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
14	42	M030	K9AFGD8H0A-T040MD1	2	14	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
15	42	M030	K9AFGD8H0A-T040MD1	6	15	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
16	42	M030	K9AFGD8H0A-T040MD1	4	16	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
17	42	M030	K9AFGD8H0A-T040MD1	5	17	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000
18	42	M030	K9AFGD8H0A-T040MD1	5	18	2020-07-04 20:08:45.000000000	2020-07-04 20:43:22.000000000

8 | Python Smart Factory concerned Example (#4 Preprocessing)

This productExplodeEvalClob function is a function provided by **BDTMAE** and is made without Python or R in the case of Clob. This is because it cannot be solved with a general SQL statement when Clob exceeds 4000 bytes.

※ The problem is the case of complex schemas that cannot be created with a single SQL. **BDTMAE** can do it, but the backend will have to send DB Query several times to create it.

※ What about BLOB? In this case, **BDTMAE** Python or R must be used. Think about the inference.

```
SELECT *
FROM table (
  productExplodeEvalClob(
    cursor(
      SELECT *
      FROM LOT_SUM
      WHERE 1=1
        AND LOT_ID = 'WKE21X1B31'
        AND MACHINE_ID = '48PARA-03'
        AND DURABLE_ID = 'Z718')
    )
);
```

8 | Python Smart Factory concerned Example (#5 Choose function)

This productExplodeEvalClob function is a function provided by **BD AETM** and is made without Python or R in the case of Clob. This is because it cannot be solved with a general SQL statement when Clob exceeds 4000 bytes.

※ The problem is the case of complex schemas that cannot be created with a single SQL. **BD AETM** can do it, but the backend will have to send DB Query several times to create it.

※ What about BLOB? In this case, **BD AETM** Python or R must be used. Think about the inference.

SQL Interface function	Description
apEval()	Provide No Data into Python Module
apTableEval()	Provide Data into Python Module
apRowEval()	Provide Splitted Data per rows into Python Module
apGroupEval()	Provided Splitted Data per grouped into Python Module

8 | Python Smart Factory concerned Example (#6 make SQL)

You need to create an input SQL that matches the Python module.

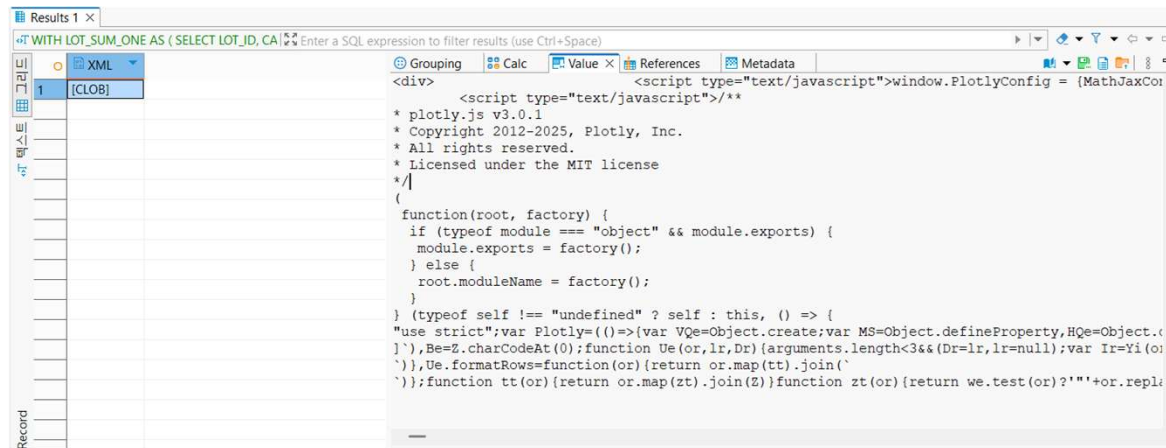
The red part is created with data as a Pandas DataFrame column and called.

```
WITH LOT_SUM_ONE AS (  
  SELECT LOT_ID,  
    CASE WHEN PROD_GRADE = 9 THEN 'BAD BLOCK'  
          WHEN PROD_GRADE = 8 THEN 'ASSEMBLY'  
          WHEN PROD_GRADE = 7 THEN 'PTX'  
          WHEN PROD_GRADE = 6 THEN 'ID FAIL'  
          WHEN PROD_GRADE = 5 THEN 'OPEN'  
          WHEN PROD_GRADE = 4 THEN 'ABS'  
          WHEN PROD_GRADE = 3 THEN 'JOLY'  
          WHEN PROD_GRADE = 2 THEN 'FIT'  
          WHEN PROD_GRADE = 1 THEN 'MIN'  
          ELSE  
            'N/A'  
        END PROD_GRADE_DESC  
  , COUNT(PROD_GRADE) CNT from table (  
    productExplodeEvalClob(cursor(  
      SELECT * FROM LOT_SUM WHERE LOT_ID = 'WKE21X1B31' AND MACHINE_ID = '48PARA-03' AND  
DURABLE_ID = 'Z718')))) GROUP BY LOT_ID, PROD_GRADE  
  )  
SELECT *  
FROM table(apTableEval(  
  cursor(  
    SELECT * FROM LOT_SUM_ONE  
  ),  
  NULL,  
  'XML' -- XML, PNG, JPG etc are BDAE™ Predefined Keywords.  
  'LOTSUM_ERR_PIE:display'))
```

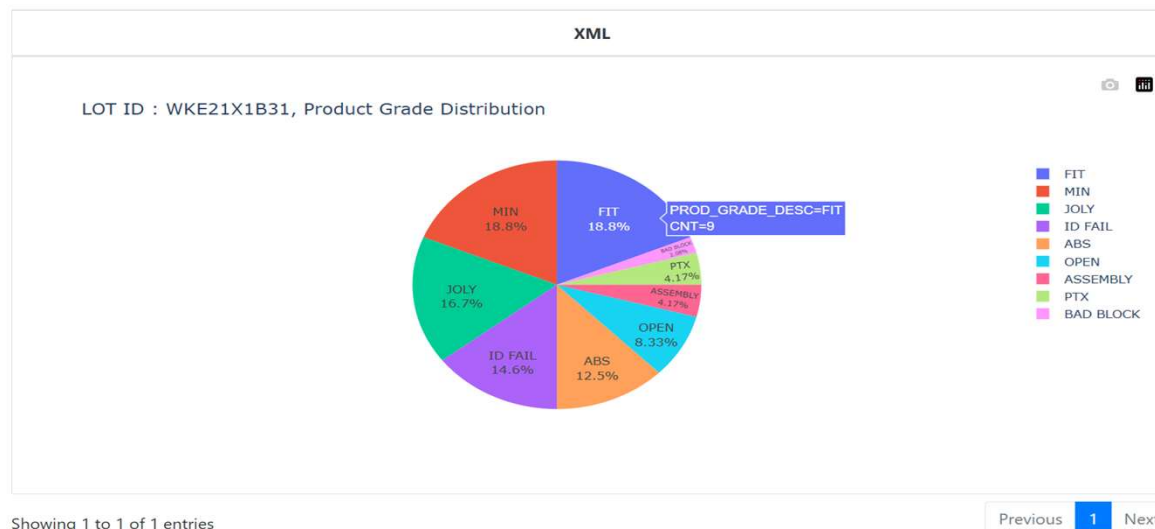
```
import pandas as pd  
import plotly.express as px  
import plotly.offline as py  
  
def display(df):  
    fig = px.pie(df, values='CNT', names='PROD_GRADE_DESC', title='LOT ID : ' + df['LOT_ID'][0] + ', Product Grade Distribution')  
    fig.update_traces(textposition='inside', textinfo='percent+label')  
    total = py.offline.plot(fig, output_type='div')  
    return (total)
```

8 Python Smart Factory concerned Example (#7 Result)

If you run the above SQL with a DB tool, the result will be displayed in the JavaScript structure on the right in CLOB format.



If you look at BDAE Web, you can see that interactive images work due to dependent CSS and Javascript. (This is because plotly package was used in the Python module.)



8 | Python Smart Factory concerned Example (#8 Parallelism)

If you want to get Pie Chart for all LOTs at TKIN/OUT time at once, not just one LOT?
The module should also be changed.

apGroupEval is automatically called for each LOT when Group Column is set to LOT_ID only.
So you don't have to think that multiple LOT_ID groups will come in. Just do your own.

In fact, this module is more reusable than the original. However, I just wanted to show the Predefined Keyword called 'XML' in the previous example.

```
import pandas as pd
import plotly.express as px
import plotly.offline as py

def display(df):
    fig = px.pie(df, values='CNT', names='PROD_GRADE_DESC', title='LOT ID : ' + df['LOT_ID'][0] + ', Product Grade Distribution')
    fig.update_traces(textposition='inside', textinfo='percent+label')
    total = py.offline.plot(fig, output_type='div')
    dict = { 'LOT_ID': [df['LOT_ID']], 'XML' : total }

    return (pd.DataFrame(dict))
```

8 | Python Smart Factory concerned Example (#8 Parallelism)

The SQL statement must also be changed. However, if it is the module in the previous chapter, the same Python module can be used in both `apTableEval()` and `apGroupEval()`. That means it is recycled.

```
WITH LOT_SUM_ALL AS (
  SELECT LOT_ID,
    CASE WHEN PROD_GRADE = 9 THEN ' BAD BLOCK '
         WHEN PROD_GRADE = 8 THEN ' ASSEMBLY '
         WHEN PROD_GRADE = 7 THEN ' PTX '
         WHEN PROD_GRADE = 6 THEN ' ID FAIL '
         WHEN PROD_GRADE = 5 THEN ' OPEN '
         WHEN PROD_GRADE = 4 THEN ' ABS '
         WHEN PROD_GRADE = 3 THEN ' JOLY '
         WHEN PROD_GRADE = 2 THEN ' FIT '
         WHEN PROD_GRADE = 1 THEN ' MIN '
         ELSE
           'N/A'
        END PROD_GRADE_DESC
  , COUNT(PROD_GRADE) CNT from table (
    productExplodeEvalClob(cursor(
      SELECT * FROM LOT_SUM WHERE BETWEEN TKIN ... AND ....))) GROUP BY LOT_ID, PROD_GRADE
)
SELECT *
FROM table(apGroupEval(
  cursor(
    SELECT * FROM LOT_SUM_ALL
  ),
  NULL,
  'SELECT CAST(NULL AS VARCHAR2(40) LOT_ID, TO_CLOB(NULL) XML FROM DUAL ' ,
  LOT_ID, -- Grouping Column
  'LOTSUM_ERR_PIE:display'))
```

The Beautiful Times ...

