VERTICA-Machine Learning

BIG DATA – BUSINESS INTELLIGENCE – MACHINE LEARNING

stratebi
open business intelligence

Características avanzadas de Vertica: Funciones UDx y Machine Learning.

ÍNDICE

1. VERTICA & MACHINE LEARNING

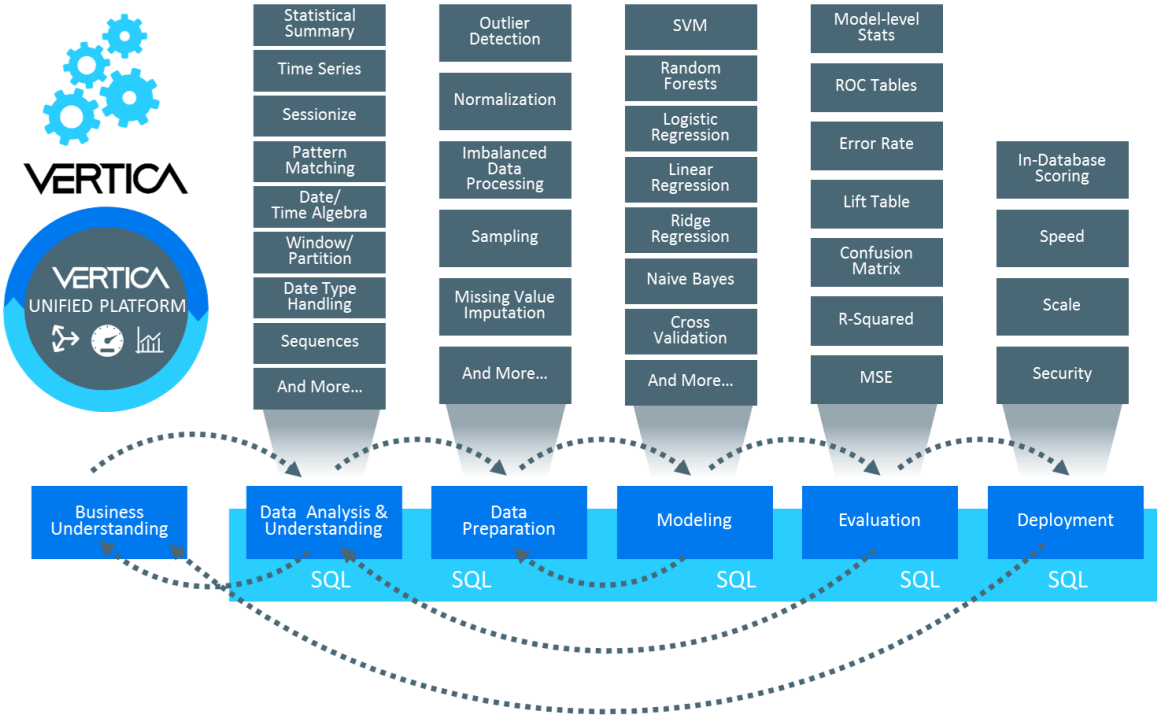
Funciones de Vertica para el Machine Learning. Ya vienen con Vertica, por lo que

i), no requieren programación ni instalación,

ii), son más eficientes que otras opciones como la integración con Python/Pandas,

iii) se aplican sobre tablas o vistas, iv) implementan los algoritmos más conocidos de Machine Learning. Además, son muy sencillas de usar y aprovechan la potencia de ejecución distribuida de un clúster Vertica.

En el siguiente diagrama se muestran las funciones nativas de Vertica para la analítica avanzada. Estas cubren todas las fases de una aplicación de Machine Learning (Pipeline de una aplicación ML): Análisis de datos origen para entender mejor el negocio, preparación de los datos, Modelado, Evaluación de los modelos generados y puesta en producción para la aplicación del modelo.



Alguno de los algoritmos incluidos en Vertica son:

- k-means
- Linear Regression
- Logistic Regression
- Naive Bayes
- Random Forest
- SVM Classification
- SVM Regression

En la página https://github.com/vertica/Machine-Learning-Examples tenemos ejemplos prácticos de funciones de Machine Learning que podemos probar en Vertica. No requieren instalar ningún componente adicional en Vertica, solo cargar los datos de ejemplo.

- Si no está instalado, instalar git para clonar el proyecto de ejemplo. Podemos omitir este paso si copiamos el código de los ejemplos por alguna otra vía.

```
sudo yum install git
git --version
git clone https://github.com/vertica/Machine-Learning-Examples
```

- Desde el cliente Vsql de Vertica ejecutar el script /data/load_ml_data.sql

```
\i load_ml_data.sql
```

EJEMPLO 1. REGRESIÓN LINEAL

La documentación de esta función se encuentra en el siguiente enlace:

- Creación del modelo de regresión lineal
  - https://my.vertica.com/docs/9.1.x/HTML/index.htm#Authoring/SQLReferenceManual/Functions/MachineLearning/LINEAR_REG.htm
- Generación de predicciones con un modelo creado
  - https://my.vertica.com/docs/9.1.x/HTML/index.htm#Authoring/SQLReferenceManual/Functions/MachineLearning/PREDICT_LINEAR_REG.htm

Para ponerla en práctica vamos a ejecutar el siguiente ejemplo del manual de Vertica

https://my.vertica.com/docs/9.1.x/HTML/index.htm#Authoring/AnalyzingData/MachineLearning/LinearRegression/ProgrammingExampleLinearReg.htm

La tabla de ejemplo faithful almacena muestras de los datos de tiempo (en minutos) de espera entra las erupciones (waiting) del geiser Old Faithful del parque nacional Yellowstone (EEUU), así como la duración de las mismas (eruptions). La duración de cada erupción está entre unos 1,5 y 5 minutos. Estos tiempos varían en cada muestra. Sin embargo, es posible estimar el tiempo de la próxima erupción basándonos en la duración de la erupción previa. De esta forma, el ejemplo muestra cómo construir un modelo predictivo para predecir el valor de la duración de la erupción (eruption) dado el valor de espera (waiting).

```
-- Borrar modelo predictivo y tabla de predicciones sobre conjunto de test si ya existe
DROP MODEL linear_reg_faithful;
drop table pred_faithful_results;


-- Creación del modelo predictivo usando regresion lineal y la tabla de datos de entrenamiento.
-- Argumentos nombreModelo, tabla de datos de training, variable a predecir, columna predictora 1, columnas predictora 2, ..., columna predictora N
-- USING PARAMETERS optimizer='BFGS' es opcional y sirve para seleccionar el método de optimización para el entrenamiento del modelo. Por defecto es CGD.
SELECT LINEAR_REG('linear_reg_faithful', 'faithful_training', 'eruptions', 'waiting' USING PARAMETERS optimizer='BFGS');


-- Podemos ejecutar la siguiente SQL para ver un sumario del modelo.
SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='linear_reg_faithful');



-- Aplicar el modelo creado sobre la tabla de datos de test faithful_testing para hacer la validación cruzada más abajo. La SQL crea una nueva tabla con los resultados de las predicciones sobre el conjunto de datos de test.
CREATE TABLE pred_faithful_results AS (SELECT id, eruptions, PREDICT_LINEAR_REG(waiting USING PARAMETERS model_name='linear_reg_faithful') AS pred FROM faithful_testing);


SELECT * FROM pred_faithful_results ORDER BY id;


-- Una vez construido el modelo, es posible evaluarlo siguiendo la técnica de la validación cruzada. Para ello se realizan predicciones sobre los datos de test y se comparan los valores predecidos con los valores reales. Por último, se calcula el MSE como métrica de evaluación de la calidad de los resultados. → Cuando más pequeño sea este valor, mejor es (la calidad) del modelo a la hora de hacer las predicciones.


SELECT MSE (eruptions::float, pred::float) OVER() FROM (SELECT eruptions, pred FROM pred_faithful_results) AS prediction_output;


-- Aplicación del modelo creado para generar las predicciones sobre las muestras
SELECT id, waiting, PREDICT_LINEAR_REG(waiting USING PARAMETERS model_name='linear_reg_faithful') AS pred FROM faithful;
```

EJEMPLO 2. CLUSTERING

La documentación de esta función se encuentra en el siguiente enlace:

- Ejecución del algoritmo K-Means sobre una tabla de entrada. El resultado es un modelo con una lista de centros de clúster.
    - https://my.vertica.com/docs/9.1.x/HTML/index.htm#Authoring/SQLReferenceManual/Functions/MachineLearning/KMEANS.htm
- Aplicación del modelo K-Means creado clusterizar las muestras
    - https://my.vertica.com/docs/9.1.x/HTML/index.htm#Authoring/SQLReferenceManual/Functions/MachineLearning/APPLY_KMEANS.htm

Para ponerla en práctica vamos a ejecutar el siguiente ejemplo del manual de Vertica

https://my.vertica.com/docs/9.1.x/HTML/index.htm#Authoring/AnalyzingData/MachineLearning/Kmeans/ClusteringDataUsingkmeans.htm%3FTocPath%3DAnalyzing%2520Data%7CMachine%2520Learning%2520for%2520Predictive%2520Analytics%7CClustering%2520Algorithms%7Ck-means%7C_____1

Este ejemplo de k-means usa dos pequeños conjuntos de datos (tablas) llamados agar_dish_1 y agar_dish_2. Usando los datos numéricos en el conjunto de datos agar_dish_1, puede agrupar los datos en k clusters. Luego, utilizando el modelo k-means creado, puede ejecutar APPLY_KMEANS en agar_dish_2 y asignarlos a los clústeres creados en su modelo original.

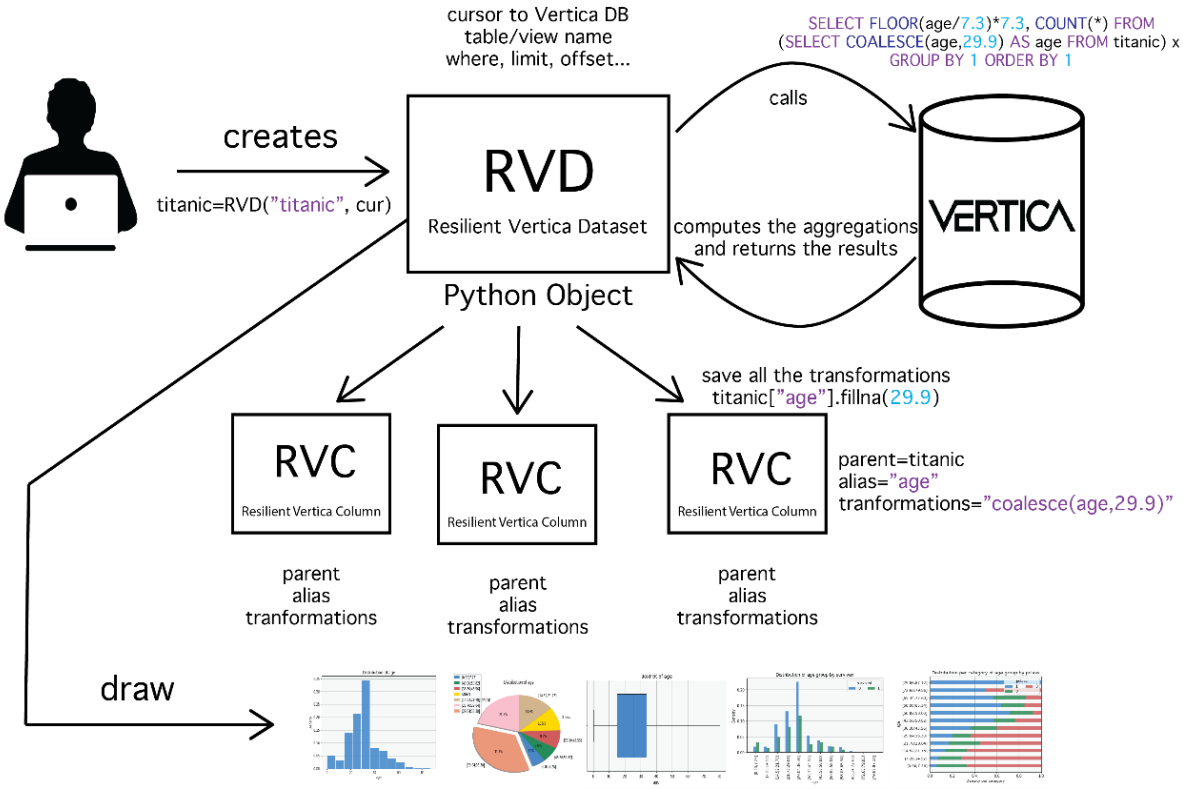
```
-- Borrar modelos y tablas si queremos volver a ejecutar el ejemplo.

DROP VIEW agar_1_view;
DROP MODEL agar_dish_kmeans;
DROP TABLE kmeans_results;

-- Creación del modelo KMEANS al que llamamos agar_dish_kmeans
-- Argumentos: nombre del modelo, tabla de entrada de entrenamiento, columnas, número de clusters a generar.
-- USING PARAMETERS sirve en este caso para excluir la columna id, marcar un número máximo de iteraciones (es importante dar un número suficiente para que el algoritmo pueda converger), vista de salida para los datos de entrenamiento clusterizados agar_1_view y columnas clave para asociar al clúster (ej id=1 cluster=5).
-- Nota: El ejemplo original daba error porque había que poner agar_1_view entre comillas
SELECT KMEANS('agar_dish_kmeans', 'agar_dish_1', '*', 5 USING PARAMETERS exclude_columns ='id', max_iterations=20, output_view='agar_1_view', key_columns='id');

-- Resultado de la table de entreamiento
SELECT * FROM agar_1_view;
SELECT cluster_id, COUNT(cluster_id) as Total_count FROM agar_1_view GROUP BY cluster_id;

SELECT GET_MODEL_SUMMARY(USING PARAMETERS model_name='agar_dish_kmeans');

-- Aplicar el modelo creado para asociar los datos de una table de entrada agar_dish_2 a cada clúster (id medición → cluster_id). El resultado es una nueva tabla
CREATE TABLE kmeans_results AS
    (SELECT id,
        APPLY_KMEANS(x, y
            USING PARAMETERS
                model_name='agar_dish_kmeans') AS cluster_id
     FROM agar_dish_2);

-- Análisis de los resultados de aplicar el modelo KMEANS creado 'agar_dish_kmeans' sobre la tabla -- agar_dish_2
SELECT cluster_id, COUNT(cluster_id) as Total_count FROM kmeans_results GROUP BY cluster_id;
```

Python para Machine Learning

Una opción alternativa al uso de las funciones nativas de Vertica para Machine Learning es la integración de Vertica con Python a través de las librerías de Machine Learning disponibles en la siguiente página:

https://github.com/vertica/Vertica-ML-Python

Estas librerías desarrolladas por el equipo de Vertica permiten desde nuestros programas Python acceder a tablas de Vertica para lectura y escritura. Los datos se pueden cargar en objetos propios de Python y ejecutar procesamiento local con librerías de ML como Pandas. Sin embargo, para mejorar el rendimiento es posible crear objetos RVD (Resilient Vertica Dataset). Estos objetos se crear desde Python pero se almacenan y procesan en Vertica, de forma que el código de Python (con librerías para ML como Pandas) se ejecuta en el clúster de Vertica, de forma transparente al usuario de Python, aprovechando el procesamiento distribuido de Vertica. Este planteamiento guarda bastantes similitudes con la tecnología Spark en clusters Hadoop, donde se hace uso de objetos denominados RDD (Resilient Distributed Dataset). En la siguiente imagen podemos ver cómo se integran Python y Vertica a través de los objetos RVD de Vertica.



La integración entre Vertica y Python no es tan eficiente como las funciones ML (built-in, nativas) de Vertica que vimos en el apartado anterior, sin embargo, proporcionan algunas ventajas frente al uso de estas:

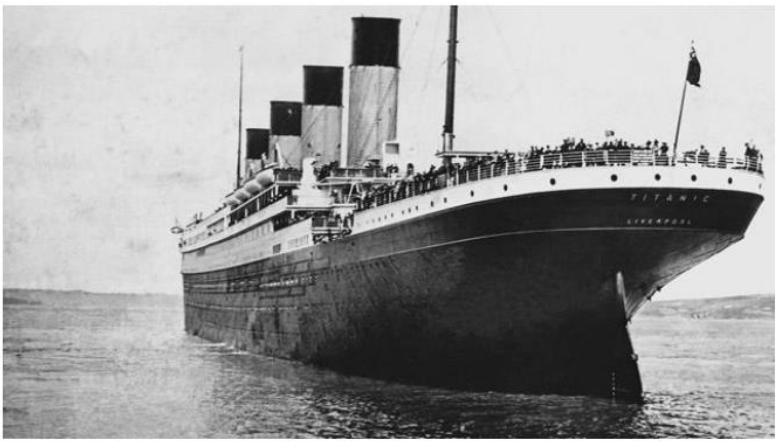
- Funcionalidad añadida: Podemos usar las librerías de Python para ML las cuales tienen un gran número de funcionalidades adicionales para casos de ML que no puedan cubrir las funciones ML nativas de Vertica.
- Capacidades de equipo de Data Science: Facilita a la amplia comunidad de Data scientist formados en Python la aplicación de sus conocimientos para la ejecución distribuida de ML aprovechando la potencia de Vertica.

Además de usarlo desde el shell de Python o programas Python compilados, se puede usar en combinación con la tecnología de visualización de notebooks Jupyter aportando el concepto de Notebook. En el siguiente enlace podemos un ejemplo del mismo, dónde se aplican distintos tipos de algoritmos de Regresión (Líneal, Random Forest) sobre el caso de estudio Titanic. En función de variables como el tipo de tarifa del pasajero, la clase social, estado civil,... se predice si un pasajero del Titanic vivirá o, por el contrario, morirá.

http://nbviewer.jupyter.org/github/vertica/vertica_ml_python/blob/master/notebooks/titanic.ipynb

VERTICA ML PYTHON

During the night of 14 April 1912, Something really unusual arrives. The passengers of the biggest boat in the world were enjoying their trip until shouts, sounds of running feet and alarms sounding removed their happiness. The Titanic sank and left behind it a lot of deaths. What happened ? This question is a mystery for the people who were not there. The purpose is to discover the truth thanks to the data we got before the embarkment. We know the information of 1300 passengers and we know if they survived after the sinking.



In order to use the Vertica ML Python Library and begin the searches, we need first to connect to the Vertica Database. We can use an ODBC connection using pyodbc.

We can now import the Vertica ML Python library and especially the RVD (Resilient Vertica Dataset) object.

In [4]:
```
from vertica_ml_python import RVD
from vertica_ml_python import read_csv # This function will help us to load the csv file in the Database.
from vertica_ml_python import drop_table # This function will help us to drop the unnecessary tables.
```

Let's now drop the table titanic from our Database (if it exists) in order to use the read_csv parser.

In [5]:
```
drop_table("titanic",cur)
```

```
The table titanic was successfully dropped.
```

Let's now try to parse the csv files. We can see that the type of boat is incorrect. However let's skip it, the parser will convert it automatically to varchar(30).

In [6]:
```
titanic=read_csv('../datasets/titanic.csv',cur)
```

```
The parser guess the following columns and types:
age: Numeric(6,3)
boat: Integer
body: Integer
cabin: Varchar(30)
embarked: Varchar(20)
fare: Numeric(10,5)
home.dest: Varchar(100)
```

In [11]:
```
import pandas
conn=pyodbc.connect("DSN=VerticaDSN")
titanicDF=pandas.read_sql("select * from titanic",conn)
import sys
print(sys.getsizeof(titanicDF))
```

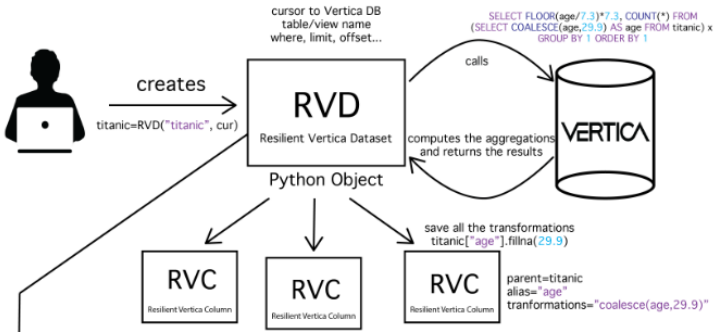
```
597049
```

In [12]:
```
print(sys.getsizeof(titanic))
```

```
56
```

As we don't load data in memory (everything is store inside the DB), this number will never change... It is a real difference.

Great, we have now our RVD object. All its methods will send SQL queries directly to Vertica in order to compute all the aggregations needed. In that case, nothing is loaded inside our machine except the final results computed by Vertica. All the users transformations are kept in mind and then used to send the final query. The following figure sums up the RVD object.



In [42]:
```
from vertica_ml_python import logistic_reg
from vertica_ml_python import drop_model
from vertica_ml_python import load_model
```

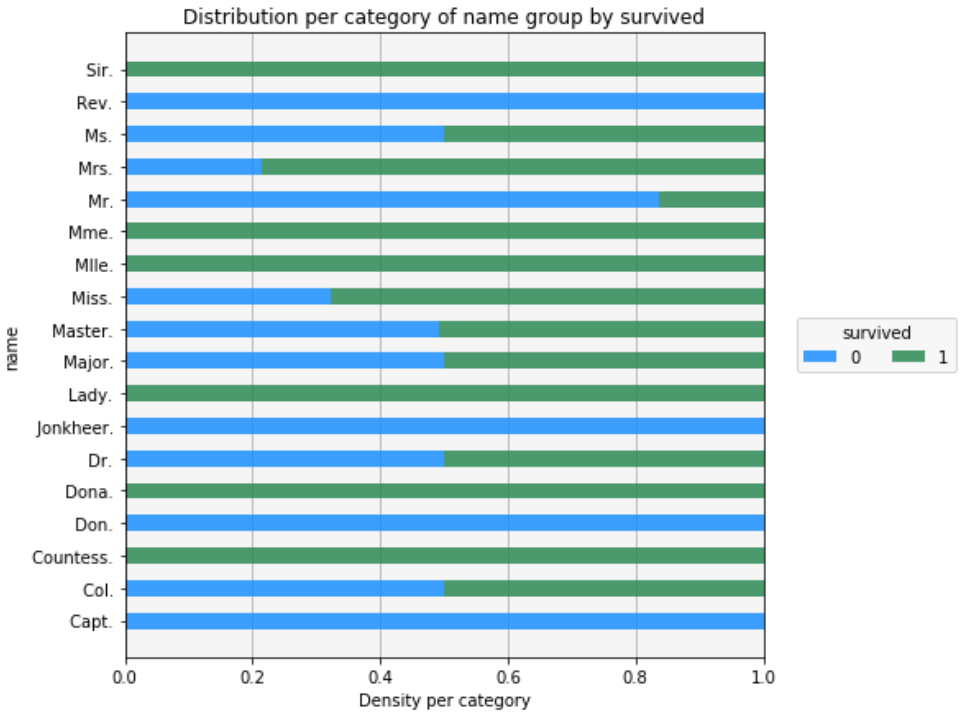
In [43]:
```
drop_model("lr_titanic",cur)
```

```
The model lr_titanic was successfully dropped.
```

In [44]:
```
logit=logistic_reg(model_name="lr_titanic",input_relation="train_titanic067",response_column="survived",
                   predictor_columns=["age","gender","family_size","embarked","fare","pclass"],cursor=cur)
print(logit)
coefficients=logit.details()
```

	coefficient	std_error	t_value	p_value
Intercept	4.88397485181215	0.586677365204033	8.32480532142844	8.44674447687093e-17
age	-0.0347705474951701	0.00768232526214005	-4.52604469463509	6.0097874973733e-06
gender	-2.39164697235406	0.188400948544658	-12.6944529251515	6.34719159375125e-37
family_size	-0.198100805188099	0.066719018884901	-2.96918043009368	0.00298595213155268
embarked	-0.224307360218314	0.138641253921166	-1.61789765942149	0.105684654790879
fare	0.00173264833035388	0.00202082681210399	0.857395755032531	0.391226204515912
pclass	-0.997092231510073	0.138741717111633	-7.18667933674052	6.63860752726033e-13

```
model_type='logistic_reg'
model_name='lr_titanic'
input_relation='train_titanic067'
response_column='survived'
predictor_columns='age,gender,family_size,embarked,fare,pclass'
regularization: none
lambda: 1.0
rejected_row_count: 0
accepted_row_count: 862
```



```
In [92]: titanic_test.select(['age','family_size','fare','embarked','survived','ticket'])
```

	age	family_size	fare	embarked	survived	ticket
0	44.0	1	7.925	S	1.0	STON/O 2. 3101269
1	32.0	1	56.4958	S	1.0	1601
2	32.0	1	56.4958	S	1.0	1601
3	31.0	1	7.925	S	1.0	STON/O 2. 3101288
4	29.0	1	9.5	S	1.0	345779
5	25.0	1	9.5	S	1.0	345768
6	25.0	1	7.2292	C	1.0	2654
7	22.0	1	7.225	C	1.0	2620
8	22.0	1	7.225	C	1.0	2658
9	20.0	2	7.925	S	1.0	STON/O 2. 3101285
10	20.0	1	7.925	S	1.0	SOTON/O2 3101284
11	19.0	1	8.05	S	1.0	A/5. 10482
12	18.0	1	8.05	S	1.0	A/5 3540
13	25.8630172413793	1	7.75	Q	1.0	367228
14	25.8630172413793	1	56.4958	S	1.0	1601
15	25.8630172413793	1	7.8875	S	1.0	3470
16	25.8630172413793	1	7.75	Q	1.0	368402
17	25.8630172413793	1	7.05	S	1.0	SOTON/O.Q. 3101308

El ejemplo probado y ejecutado en nuestro Vertica local, tanto con el shell de Python como desde Jupyter puede resumirse en la ejecución del siguiente código Python:

```
-- Conexión a Vertica. Requiere crear un conexión DNS en la máquina. También se podría usar JDBC

import pyodbc
cur=pyodbc.connect("DSN=VerticaDSN").cursor()

-- Importar las librerías de Machine Learning para la integración Vertica – Python y las de Regresion Lineal
from vertica_ml_python import RVD
from vertica_ml_python import read_csv # This function will help us to load the csv file in the Database.
from vertica_ml_python import drop_table # This function will help us to drop the unnecessary tables.
from vertica_ml_python import logistic_reg
from vertica_ml_python import drop_model
from vertica_ml_python import load_model

-- Usamos la conexión con Vertica para borrar la tabla Titanic desde Python
drop_table("titanic",cur)

-- Creamos una nueva tabla Titanic en Vertica desde CSV con los datos del ejemplo
titanic=read_csv('../datasets/titanic.csv',cur)

-- Cargamos la tabla titanic en un objeto RVD de Vertica usando la conexión por DSN VerticaDSN
titanic=RVD('titanic',dsn="VerticaDSN")
print(titanic)

-- Importar las librerías de ML de Python "PANDAS"
import pandas


-- Borrar el modelo de nombre "mi_modelo_regresion_titanic" si ya existía
drop_model("mi_modelo_regresion_titanic",cur)

-- Crear el modelo de regresión que llamaremos "mi_modelo_regresion_titanic" a partir del RVD de Vertica
que contiene los datos.
        logit=logistic_reg(model_name=" mi_modelo_regresion_titanic",
        input_relation="titanic",response_column="survived",
        predictor_columns=["age","fare","pclass"], cursor=cur)
```

Funciones UDx

No son para machine learning, sino para que el usuario cree nuevas funciones que encapsulen cálculos complejos (ej. un KPI complejo), nuevas funciones agregación,….. y que se pueden usar en las sentencias SQL de vertica (añaden funcionalidad a las funciones nativas de Vertica)

Existen varios tipos de funciones UDx según su propósito:

Tipo	Entrada	Salida	Lenguajes	Particularidad
Agregación (UDAF)	1 columna	1 columna	C++	
Analíticas (UDAnF)	1 o más filas	1 fila	C++ y Java	Es posible calcular los valores de salida a partir de múltiples filas
Escalares (UDSFs)	1 fila	1 valor	C++, Python, Java y R	Se pueden usar en cualquier sitio que se usa una función nativa de Vertica
Transformación (UDTFs)	1 o más particiones de tabla	1 tabla nueva	C++, Python, Java y R	
Extensiones de carga	Fuentes de datos	-	C++ y Java	Soportar la carga de nuevos tipos de fuentes de datos, filtros en la carga y parseadores del formato de entrada

Por ejemplo, las funciones escalares (UDSFs) devuelven un único valor por cada fila que reciben. Se pueden usar en cualquier parte de la sentencia SQL donde podamos aplicar una función nativa de Vertica. Este tipo de funciones son muy útiles para:

- Implementar cálculos muy complejos para ser realizados usando la sintaxis SQL
- Cuando la complejidad del cálculo hace que el uso de SQL y funciones nativas de Vertica afecte al rendimiento.
- Usar funciones diseñadas por terceros (otras empresas o departamentos) manteniendo el alto rendimiento de consulta en Vertica.

En la página https://github.com/vertica/UDx-Examples tenemos ejemplos prácticos de funciones UDx que podemos probar en Vertica. Uno de los ejemplos es una función escalar implementada con Python para hacer una conversión de divisas. Podemos ver el código de la misma en el siguiente enlace https://github.com/vertica/UDx-Examples/tree/master/Python/currency_conversion

Para hacerla funcionar tenemos que llevar a cabo los siguientes pasos:

- Instalar git para clonar el proyecto de ejemplo. Podemos omitir este paso si copiamos el código de los ejemplos por alguna otra vía.

```
sudo yum install git
git --version
git clone https://github.com/vertica/UDx-Examples.git
```

- Instalar Python en las máquinas donde está instalado Vertica. En principio no es necesario, pues la instalación de Vertica 9.1 ya lo tiene instalado.
- Cargar los datos de ejemplo ejecutando el script *currency_convert_load_data.sql*:
  - Lo hemos modificado porque no funcionaba correctamente debido a las rutas y otros pequeños problemas.

```
-- currency_convert Vertica Python UDx example

-- Creación de la función

-- Create the User Defined Scalar Function (UDSF)

CREATE OR REPLACE LIBRARY pylib AS
'/home/dbadmin/pruebasVertica/udx_python/UDx-
Examples/Python/currency_conversion/currency_convert.py' LANGUAGE 'Python';

CREATE OR REPLACE FUNCTION currency_convert AS LANGUAGE 'Python' NAME
'currency_convert_factory' LIBRARY pylib fenced;


-- Create the table and load the data

CREATE SCHEMA pruebasUDX;


CREATE TABLE pruebasUDX.items (product varchar(30), currency varchar(3), value
money);


\set input_file '''':t_pwd'/home/dbadmin/pruebasVertica/udx_python/UDx-
Examples/Python/currency_conversion/items.tbl'''

COPY pruebasUDX.items FROM :input_file DELIMITER '|' NULL '' DIRECT;



-- Return all the rows in items_cost table and then use the UDSF

-- to query the table.

SELECT product, currency, value FROM pruebasUDX.items;


-- currency es el nombre la columna, pero podemos darle otro de los definidos en
rates2USD, ej. 'EUR'

-- value es el nombre de la columna que queremos convertir

SELECT product, currency_convert(currency, value) AS cost_in_usd FROM
pruebasUDX.items;


-- por ejemplo, para convertir a euros

SELECT product, currency_convert('EUR', value) AS cost_in_usd FROM
pruebasUDX.items;


-- Borrar la table (si no queremos seguir ejecutando el ejemplo)

DROP TABLE pruebasUDX.items;
```

- La función currency_convert.py es la función escalar (UDSFs) implementada en Python:

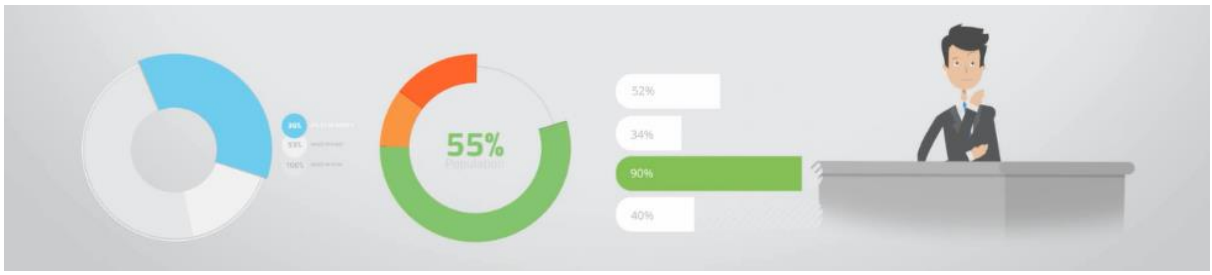
```
import vertica_sdk
import decimal

rates2USD = {'USD': 1.000,
          'EUR': 0.89977,
          'GBP': 0.68452,
          'INR': 67.0345,
          'AUD': 1.39187,
          'CAD': 1.30335,
          'ZAR': 15.7181}

""" Como argumento le pasamos el tipo de función UDX que vamos a definir. En este caso es una función de tipo scalar UDSFs """
class currency_convert(vertica_sdk.ScalarFunction):
   """Converts a money column to another currency. Returns a value in USD """
   def __init__(self):
      pass

   def setup(self, server_interface, col_types):
      pass

""" En esta función es donde se implementa la lógica del procesamiento. En el caso del ejemplo es una conversión de divisa """
   def processBlock(self, server_interface, arg_reader, res_writer):
      server_interface.log(" $$$ ")

""" Este bucle procesa mientras hayan filas """
      while(True):
""" Obtenemos el valor divisa a partir del parámetro pasado y el objeto definido arriba en el código rates2USD. ej. 'EUR' """
         currency = arg_reader.getString(0)

            """ Necesario try catch para gestión de errores"""
         try:
            rate = decimal.Decimal(rates2USD[currency])

         except KeyError:
            server_interface.log("ERROR: {} not in dictionary.".format(currency))
            # Scalar functions always need a value to move forward to the
            # next input row. Therefore, we need to assign it a value to
            # move beyond the error.
               """ Si no está la divisa en diccionario rates2USD o no se pasa divisa, entonces usar USD"""
            currency = 'USD'
            rate = decimal.Decimal(rates2USD[currency])
```

```
            """ Conversión de la divisa """
            """ Segundo argumento si damos el nombre value entonces coge el valor de fila que es lo que queremos """
        starting_value = arg_reader.getNumeric(1)
        converted_value = decimal.Decimal(starting_value  / rate)
res_writer.setNumeric(converted_value)
            """ Avance a siguiente fila """
        res_writer.next()
        if not arg_reader.next():
            break
def destroy(self, server_interface, col_types):
    pass

class currency_convert_factory(vertica_sdk.ScalarFunctionFactory):

""" Creación de la función Scalar """
  def createScalarFunction(self, srv):
    return currency_convert()

""" Definir argumentos y tipo de la función. Podemos tener hasta 1600 argumentos """
  def getPrototype(self, srv_interface, arg_types, return_type):
    arg_types.addVarchar()
    arg_types.addNumeric()
    return_type.addNumeric()

""" Definir tipo del valor devuelto por la función. Solo devuelve un valor"""
  def getReturnType(self, srv_interface, arg_types, return_type):
    return_type.addNumeric(9,4)
```

2. DESCRIPCIÓN

En Stratebi ofrecemos **gran cantidad de soluciones analíticas** por una compañía de **rápido crecimiento,** innovando en las áreas tecnológicas de mayor desarrollo en la actualidad: **Business Intelligence, Big Data y Social Intelligence**, muchas de ellas, basadas en soluciones **Open Source**.

Además, somos **Partners Certificados en Microsoft PowerBI y Vertica**, con gran número de proyectos con ámbas tecnologías



Desarrollamos nuevas soluciones analíticas basadas en Open Source, para la generación de Cuadros de Mando en tiempo real, con tecnologías IoT para SmartCities, machine learning, etc...

LinceBI
The best open source
BI based solution

3. TECNOLOGÍAS

4. DESCRIPCIÓN DE LAS SOLUCIONES

Stratebi son los creadores de la solución LinceBI.com sobre la que podrás desarrollar e innovar.

5. INFORMACIÓN SOBRE STRATEBI

Stratebi es una empresa española, con sede en Madrid, creada por un grupo de profesionales con amplia experiencia en sistemas de información, soluciones tecnológicas y procesos relacionados con soluciones de Open Source y de inteligencia de Negocio.

Esta experiencia, adquirida durante la participación en proyectos estratégicos en compañías de reconocido prestigio a nivel internacional, se ha puesto a disposición de nuestros clientes a través de Stratebi.

Stratebi es la única empresa española que ha estado presente en los 9 Pentaho Developers celebrados en Europa (Mainz-Alemania, Barcelona, Lisboa, Roma, Amsterdam, Sintra, Amberes (2) y Londres), habiendo organizado el de Barcelona.

En Stratebi nos planteamos como **objetivo** dotar a las compañías e instituciones, de herramientas escalables y adaptadas a sus necesidades, que conformen una estrategia Business Intelligence capaz de rentabilizar la información disponible. Para ello, nos basamos en el desarrollo de soluciones de Inteligencia de Negocio, mediante tecnología Open Source.

Stratebi **son** profesores y responsables de proyectos **del Master en Business Intelligence de la Universidad UOC.**

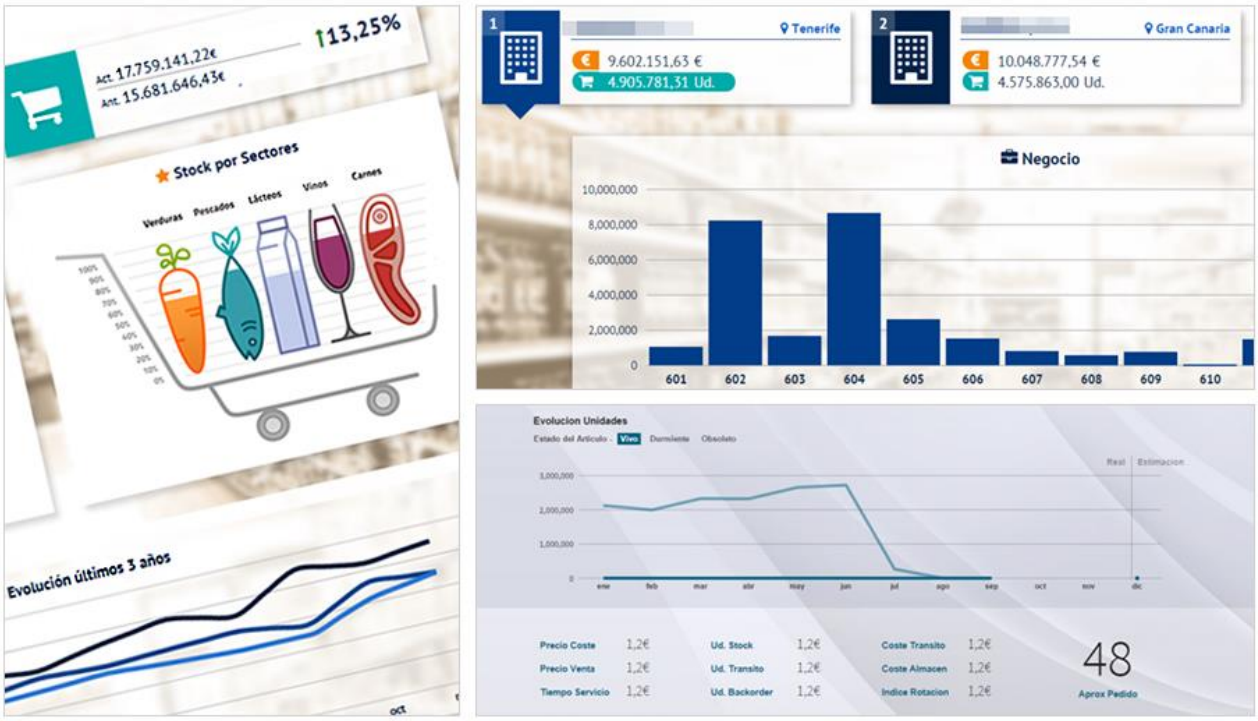
Los profesionales de Stratebi son los creadores y autores del primer weblog en español sobre el mundo del Business Intelligence, Data Warehouse, CRM, Dashboards, Scorecard y Open Source.

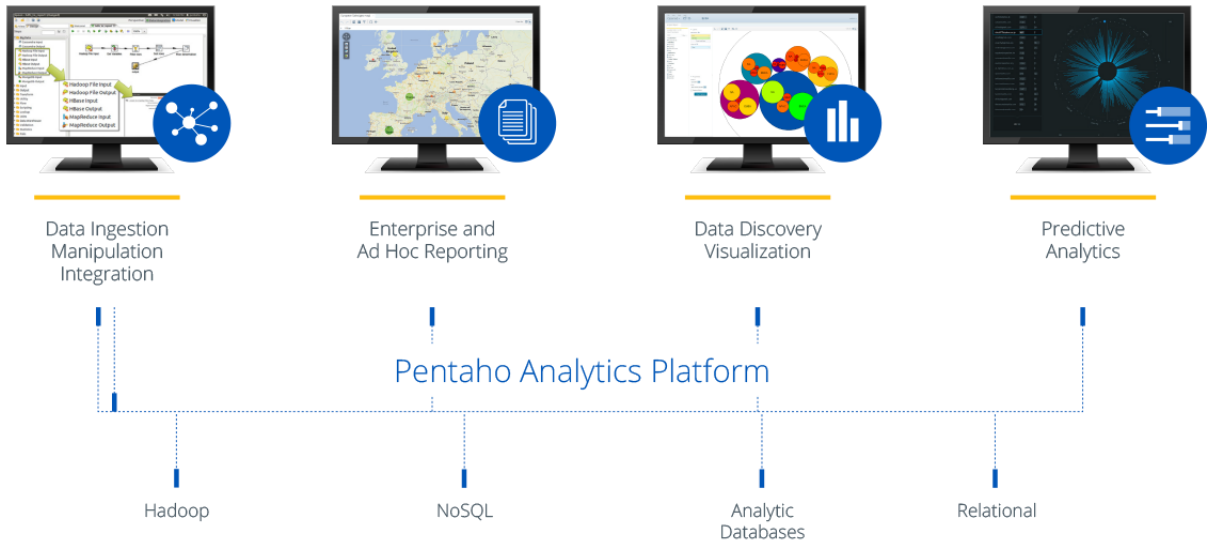
Stratebi es partner de las principales soluciones BI Open Source: Pentaho, Talend, Jedox, SQL Power…

Todo Bi, se ha convertido en una referencia para el conocimiento y divulgación del Business Intelligence en español.

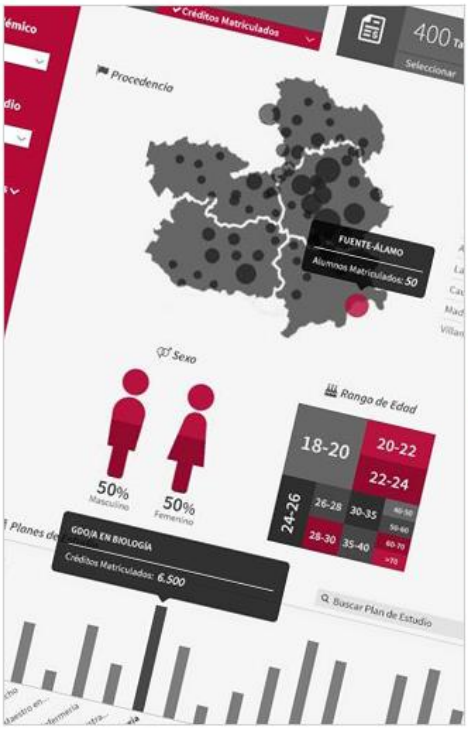
6. EJEMPLOS DE DESARROLLOS ANALYTICS

A continuación se presentan **ejemplos de algunos screenshots** de cuadros de mando diseñados por Stratebi, con el fin de dar a conocer lo que se puede llegar a obtener, así como Demos Online en la web de Stratebi:



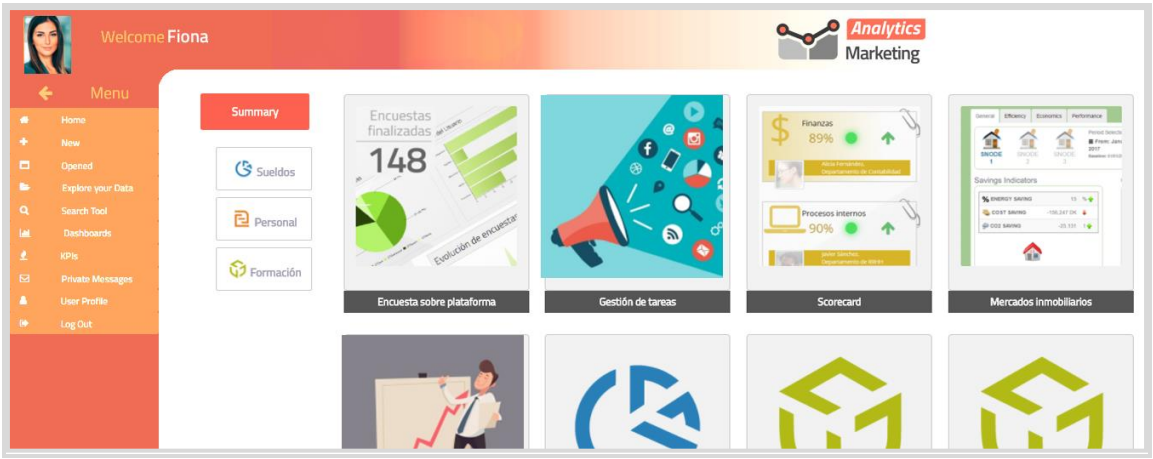



facebook DASHBOARD COMENTARIOS
Sexo
Sentimiento
Localización
Tipo de recurso
Evolución Comentarios por días
Smart MADRID
Distrito | Barrio
Última Actualización
18:24
TOP 5 PUNTOS DE MEDIDA
PROXIMIDAD DE PUNTOS DE MEDIDA
EVOLUCIÓN INTENSIDAD DE TRÁFICO


Créditos Matriculados
Procedencia
FUENTE-ÁLAMO
Alumnos Matriculados: 50
Sexo
50% Masculino
50% Femenino
Rango de Edad
18-20
20-22
22-24
24-26
26-28
28-30
30-35
35-40
GDO/A EN BIOLOGÍA
Créditos Matriculados: 6.500
Buscar Plan de Estudio


ITERÉS PLATAFORMA
181 Inscritos en Cursos
8 Desinscritos en Cursos
181 Inicios de Cursos
110 Inscritos en Eventos Virtuales
63 Inscritos en Debates
181 Inscritos en Talleres
USUARIOS ACTIVOS POR PAÍS
INSCRITOS EN CURSOS POR COMUNIDAD EDU


JORNADA COMPLETA
IDIOMAS
TIC
TIPO DE CENTRO
TIPO DE AUSENCIA
Nº de profesores por sexo
36%
64%
Nº total de profesores
1.587
Media de edad
43 años
Búsqueda
TERRITORIAL : Madrid - Capital


Welcome Fiona
Analytics
Marketing
Menu
Home
New
Opened
Explore your Data
Search Tool
Dashboards
KPIs
Private Messages
User Profile
Log Out
Summary
Sueldos
Personal
Formación
Encuestas finalizadas
148
Encuesta sobre plataforma
Gestión de tareas
Finanzas
89%
Procesos internos
90%
Scorecard
Mercados inmobiliarios


2,963
68