Making an RDBMS Data Scientist Friendly

Advanced In-database Interactive Analytics with Visualization Support

Joseph Vinish D'silva
joseph.dsilva@mail.mcgill.ca

Florestan De Moor
florestan.demoor@mail.mcgill.ca

Bettina Kemme
kemme@cs.mcgill.ca

School of Computer Science, McGill University
Montréal, Canada

ABSTRACT

We are currently witnessing the rapid evolution and adoption of various data science frameworks that function external to the database. Any support from conventional RDBMS implementations for data science applications has been limited to procedural paradigms such as user-defined functions (UDFs) that lack exploratory programming support. Therefore, the current status quo is that during the exploratory phase, data scientists usually use the database system as the "data storage" layer of the data science framework, whereby the majority of computation and analysis is performed outside the database, e.g., at the client node. We demonstrate AIDA, an in-database framework for data scientists. AIDA allows users to write interactive Python code using a development environment such as a Jupyter notebook. The actual execution itself takes place inside the database (near-data), where a server component of AIDA, that resides inside the embedded Python interpreter of the RDBMS, manages the data sets and computations. The demonstration will also show the visualization capabilities of AIDA where the progress of computation can be observed through live updates. Our evaluations show that AIDA performs several times faster compared to contemporary external data science frameworks, but is much easier to use for exploratory development compared to database UDFs.

PVLDB Reference Format:
Joseph Vinish D'silva, Florestan De Moor, Bettina Kemme. Making an RDBMS Data Scientist Friendly: Advanced In-database Interactive Analytics with Visualization Support. *PVLDB*, 12(12): 1930-1933, 2019.
DOI: https://doi.org/10.14778/3352063.3352102

1. INTRODUCTION

Data scientists typically need to work with data from a variety of sources, including those stored in conventional Relational Database Management System (RDBMS) implementations. However, traditional RDBMSes have been cautious to adapt to the needs of the data science community at large. This is because introducing support for many non-relational operations such as linear algebra require significant changes to the RDBMS engine. Further, SQL was never envisioned to be a language with the high level language (HLL) capabilities that a typical data science project requires.

This has resulted in the evolution of specialized data science frameworks that are external to the database [8, 3]. However, these systems incur data transfer delays from the database [6] and often have to duplicate the relational processing capabilities. On the other hand, when it comes to conventional RDBMS implementations, so far the only popular attempt to support the needs of the data science community has been to integrate interpreted HLLs, such as Python and R. This approach requires the users to write their processing logic in an HLL function which is then executed in the database space in the form of a *user defined function* (UDF) [6, 5]. While computationally efficient, this approach fares poorly when it comes to the easiness of use [6], as UDFs are quite restrictive in terms of supporting interactive exploratory work and visualization.

We demonstrate AIDA (Abstraction for **A**dvanced **In**-**d**atabase **A**nalytics) [1] that couples database engine and HLL in a quite different way. AIDA is a data science framework that performs all the computations inside the database while allowing users to write interactive Python code for exploration. The main features of AIDA are:

User-friendly syntax. AIDA's client API is written in Python and emulates the syntax and semantics of popular data science packages such as NumPy and pandas, thereby reducing the learning curve for users.

Thin client. Users can use a regular Python interpreter or data science environments like Jupyter notebooks, to use AIDA's client API package. All data transformations requested by the user are transparently sent to AIDA's server inside the RDBMS, ensuring *near-data* computation.

Exploratory programming paradigm. Unlike database UDFs that are strictly procedural, AIDA's programming semantics facilitate interactive programming in a similar way as most "outside-the-database" data science frameworks, allowing the users to analyze the outcome of a set of operations before deciding further transformations. To this extent, AIDA also supports both explanatory and exploratory data visualizations.

A unified abstraction for computation. Data sets in AIDA are represented as *TabularData* objects and can be used to encapsulate the data inside a database table or a *table-like* Python data structure. TablularData serves as a unified abstraction that supports both relational and linear algebra operations seamlessly. Relational operations are in-

Proceedings of the VLDB Endowment, Vol. 12, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3352063.3352102

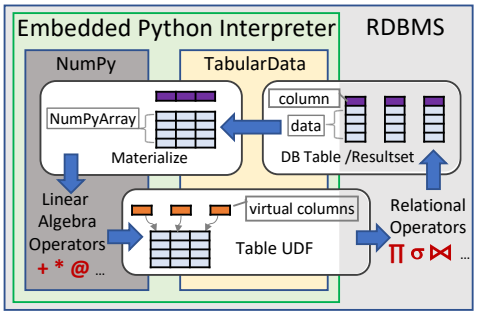

Figure 1: TabularData Abstraction [1].

ternally executed using the underlying RDBMS and linear algebra operations are performed using NumPy.

User Extensions. While AIDA provides the most commonly used operators for relational and linear algebra processing, users can write custom operators using a simple Python syntax, allowing them to integrate external data transformation libraries into their workflow.

Table 1 summarizes AIDA's capabilities in comparison with contemporary popular data science tools.

Table 1: AIDA Vs. Popular Data Science Tools.

	Languages	*Interactive*	*Incremental*	*Near-data*	*Visualization*	*Unified*
AIDA	Python	✓	✓	✓	✓	✓
DB UDF	Python, SQL	✗	✗	✓	✗	✓
pandas	Python	✓	✓	✗	✓	✗
Spark	Scala	✓	✓	✗	✓	✗

2. AIDA OVERVIEW

In this section, we provide a brief overview of AIDA's architecture and programming interface. Interested readers can find detailed information about AIDA in [1].

AIDA primarily consists of a data abstraction called *TabularData* and two software components (i) A client-side API that is a Python package which can be used with any regular Python interpreter or more popular data science programming environments such as Jupyter Notebooks. (ii) AIDA's server component, that resides in the embedded Python interpreter of the RDBMS and is responsible for overseeing computational operations and maintaining data sets.

2.1 TabularData

AIDA encapsulates all the data sets using an abstraction called *TabularData*. A TabularData object resides in the database and can be used to represent the data in a database table, the result set of a database query, or any *table-like* Python object such as a NumPy array (see Figure 1). Users can apply various data transformations on TabularData objects using a large repertoire of methods that AIDA's API provides. TablularData objects are immutable and transforming a TabularData object creates a new one. However, for optimization reasons, in many cases, AIDA does not "materialize" the data of a TabularData object unless there is a need for it.

2.2 User APIs

The detailed client-server architecture of AIDA is shown in Figure 2. AIDA's client-side API is Python based and

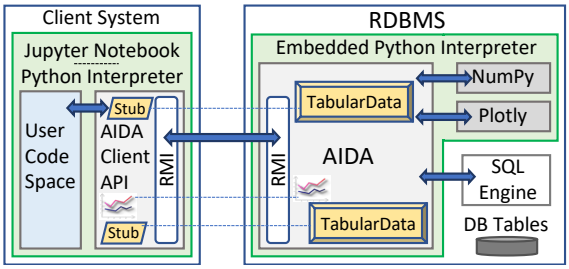

Figure 2: AIDA - High level architecture [1].

is intentionally modeled after popular data science packages such as NumPy and pandas for familiarity. Its interactive programming syntax supports both exploratory programming needs and incremental solution development. However, beyond the syntactic similarities, AIDA's client API serves the function of seamlessly shifting the computations to AIDA's server which performs these computations inside the database. This is a fundamental advantage that AIDA has over such packages which need to retrieve the data from the database to perform the computations.

Relational Operations. Users can perform relational operations on a TabularData object using an object-relational mapping (ORM) [4] based syntax. Below is an example of aggregating the supplier table of TPC-H, to compute the number of suppliers and the total account balance.

```
si = supplier.agg((,{COUNT('*'): 'numsup'}
                   ,{SUM('s_acctbal'): 'totsbal'}))
```

AIDA will keep track of such relational transformations instead of executing them immediately. When they need to be materialized, AIDA internally translates the ORM expressions into the equivalent SQL and leverages the RDBMS engine to execute the operation. If the original data set happens to be a Python object, it will dynamically generate a *table-UDF* [6] to expose it to the RDBMS for SQL processing (see Figure 1).

Linear Algebra. The linear algebra syntax for operating on TabularData objects are based on NumPy's, which itself is mathematically intuitive and compact. In the example below, we are performing a matrix based division on the above TabularData object to represent the number of suppliers in thousands and total account balance in millions.

```
res = si/numpy.asarray([1000, 1000000]);
```

If the original data set happens to be in the database, AIDA will internally execute a SQL to fetch it (see Figure 1). It is also capable of leveraging database optimizations like *zero-copy* [2] that minimizes any data transfer overheads.

User Extensions. Further, AIDA lets users extend the operator repertoire by allowing them to write custom operators. For example, in one of the demo scenarios, we will demonstrate how to perform a custom transformation to compute the geodesic distance between two coordinates using a third party Python package.

Data Visualizations. Visualizations are an important resource in any data science project. AIDA provides visualization support using plotly[1] which is a very popular Python based interactive graphing library. Users can, for example, use a Jupyter Notebook to interact with the visualization components of AIDA (see Figure 2). They can also observe

[1]https://dash.plot.ly/

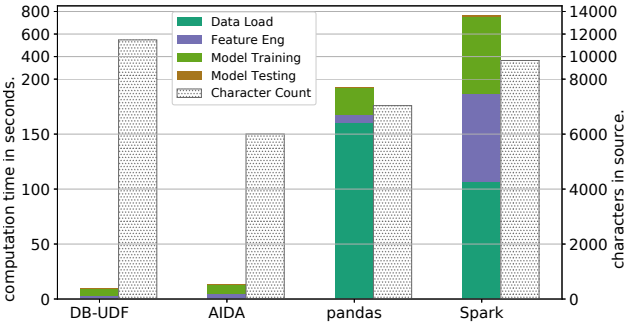

Figure 3: Linear regression on Bixi data set [1].

the progress of the training of a model by watching live updates of error rate graphs, a scenario that we will be covering in our demonstration.

2.3 Client-server Architecture

As previously mentioned, AIDA's server resides in the embedded Python interpreter of the RDBMS, thus facilitating the execution of the computation in the database address space. The transformations that the users initiate in the user code space are transparently sent by AIDA's client API to the server using remote method invocation (RMI) (see Figure 2). AIDA also has optimizations to reduce RMI overhead for iterative statements such as *loops* by shifting the complete iteration logic to the server side.

2.4 Performance and Usability Evaluation

We study the usability and performance of our system by performing an exploratory workflow analysis and compare it with the contemporary in-database approach of using UDFs and popular external-to-database approaches such as pandas and Spark. We use the Bixi bicycle trip data set[2] to perform a linear regression to predict the duration of a trip, given the distance between two bicycle stations. From the performance metrics in Figure 3 (left y-axis), we can see that both the in-database approaches are on par, while external systems such as pandas and Spark perform poorly, predominantly due to their data transfer overhead and less efficient relational operation implementations. Overall AIDA turns out to be 56 times faster than Spark and 14 times faster than pandas. On the usability analysis, we resort to comparing using the source code size given the variation in programming paradigms (right side y-axis). While AIDA is on par with pandas and Spark, database UDFs take about twice the amount of source code. Therefore, we can see that AIDA provides a highly user-friendly in-database programming paradigm without sacrificing performance.

3. DEMONSTRATION DESCRIPTION

In our demonstration, we will cover two primary use cases of AIDA. In the first one, we will go over a detailed data science workflow written in AIDA, consisting of various phases of a data science project such as data set exploration, feature engineering, data visualization, model training, etc. In the second scenario, we will demonstrate how easy it is to represent complex queries such as those found in the TPC-H using AIDA's API and how it supports developing interactive widgets capable of dynamically updating the results based on user inputs.

[2]https://www.kaggle.com/aubertsigouin/biximtl

3.1 User Interface

Users can load AIDA's client packages in any Python interpreter and perform their work. However, advanced data science environments like Jupyter notebook can help users leverage the full potential of AIDA such as using interactive visualization capabilities. Therefore, for our demonstration, we will use Jupyter notebook as the user interface of choice. Users can load AIDA's Python client package into a Jupyter notebook and login to the database.

3.2 Exploratory Data Science Using AIDA

In our primary demo workflow, we walk the users through a scenario where they have to explore the Bixi bicycle trip data set. Here the "data scientist" is in the process of developing a model that can predict the duration of a trip, using the distance between the bicycle stations. This is a concise workflow that covers various real-world aspects of exploratory data science such as analyzing the data sets including visualizing them, eliminating outliers and noisy data, performing feature engineering, training a model and monitoring the progress of error rate and finally comparing the performance of the model against the test data.

Once the users log into the database using AIDA's client API, they will be able to explore the data sets in the database. AIDA's connection object allows users access to the database tables through the TabularData abstraction. The underlying RMI framework provides *stubs* (see Figure 2) to these TabularData objects for the user code to operate upon.

In-database data exploration. Similar to contemporary data science packages, users can take a look at some of the sample records or a detailed summary of statistical characteristics of various attributes in the data set. Since providing such information incurs only minimal data transfers, this is a key advantage for AIDA that keeps the bulk of the data in the database unlike the external data science frameworks that need to take out all of the data from the database to work with.

Visualization. Further, similar to contemporary data science packages that allow users to programmatically visualize various aspects of the data set, users can visualize data in AIDA using the Plotly package support integrated into it. To this extent, users will learn how to visualize the geographical locations of the bicycle stations in the data set on a map along with the density distribution of those stations in the data set (see Figure 4).

Relational operations without SQL. As part of preprocessing the data set, users will observe how easy it is to perform relational processing operations such as *selection*, *join*, *aggregation*, etc., using AIDA's ORM syntax, eliminating the dependency on the knowledge of SQL. In this aspect, users will also experience how such processing can be done through an interactive widget. For example, users will be able to use a *histogram* chart and a *slider* to limit the range of outliers that we should accommodate in our working data set (see Figure 5).

User transforms for feature engineering. The ability to write custom transformations is key to the *feature engineering* aspect of a data science project. To facilitate this, AIDA supports the concept of user transformations. In our demo workflow, users will observe how they can use the geographical coordinates of bicycle stations to compute the geodesic distance [7] between them using a third party Python package to enrich the working data set.

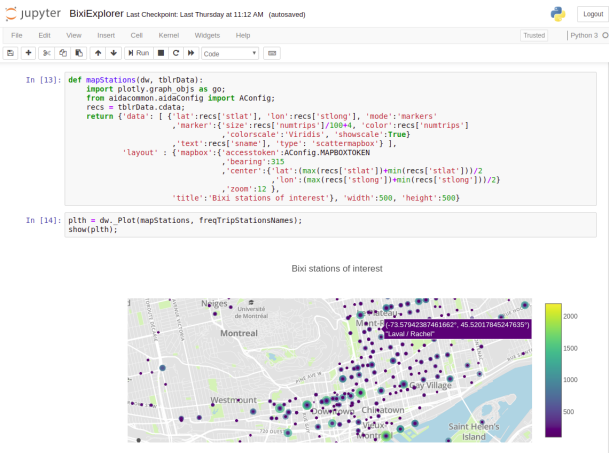

Figure 4: Visualizing the station locations in the data.

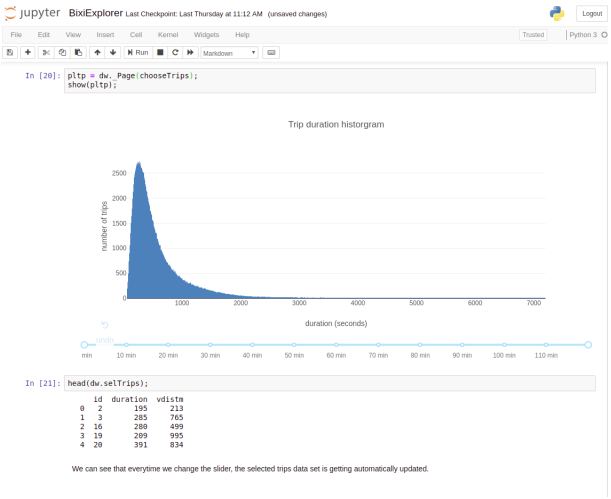

Figure 5: Updating a data set through visual interaction.

Linear algebra and model development. As we continue the exploratory workflow development, users will familiarize themselves on how to develop new models easily using AIDA's API. In this concise example, users will see how to define an *error* function and a *gradient descent* function that is required to develop a linear regression model. As these functions are primarily based on linear algebra, users will notice the simple syntax of AIDA's linear algebra API on par with contemporary data science packages, thus requiring no learning curve. Further, they will also observe the programming convenience of the TabularData unified abstraction that supports both relational and linear algebra operations on the underlying data sets without the users being burdened with moving data sets back and forth between the database and the statistical packages.

Live visualization for monitoring. As we move into the training phase, users have the opportunity to observe the error rates being updated on a live graph as the model training progresses. Such capabilities play a significant role in improving the overall productivity of the data scientist.

We will conclude the workflow by applying the trained model on our test data and verifying its predictions.

Finally, to understand the performance benefits of AIDA, which performs all the computations within the database, users will have the opportunity to compare its execution with a workflow written in pandas, a popular data science package, that has to first extract the data from the database before doing any analysis.

3.3 Conventional Analytics

AIDA's programming versatility and capability to support visualization and interactive widgets from within the database environment also opens up interesting options when it comes to providing conventional analytics capabilities such as those that are common in *decision support systems* used for *business intelligence*.

To this extent, users will be able to go through various workflows that are modeled over the TPC-H database. For example, in one such scenario, the TPC-H query 5 that looks at the revenue from each country based on the geographical regions, will be presented with its results visually overlayed over a world map. This is a fairly complex query consisting of 6 tables in the join along with selection conditions and aggregations and is easily expressible in AIDA without writing any SQL. The user can interactively change the date range of the query and switch the geographical region of interest and the query will automatically refresh the data displayed over the map.

As AIDA provides interactive visualizations and widgets through libraries that support contemporary web scripting languages, the results produced by AIDA can be easily incorporated into a web server. We are currently in the process of facilitating the additional platform support required for this. With this, a data scientist will be able to easily develop a workflow to analyze some new concept and only publish the end-user facing visualizations or interactive widgets to their user community without having to share the complex programming nuances associated with typical data science workflows. This will help streamline the *prototype - user validate* process flow of data science projects.

4. REFERENCES

[1] J. V. D'Silva, F. De Moor, and B. Kemme. AIDA-Abstraction for Advanced In-Database Analytics. *PVLDB*, 11(11):1400–1413, 2018.

[2] J. Lajus and H. Mühleisen. Efficient Data Management and Statistics with Zero-Copy Integration. In *SSDBM*, pages 12:1–12:10. ACM, 2014.

[3] W. McKinney. pandas: a Foundational Python Library for Data Analysis and Statistics. *Python for High Performance and Scientific Computing*, pages 1–9, 2011.

[4] S. Melnik, A. Adya, and P. A. Bernstein. Compiling Mappings to Bridge Applications and Databases. *Transactions on Database Systems*, 33(4):22, 2008.

[5] H. Mühleisen and T. Lumley. Best of Both Worlds: Relational Databases and Statistics. In *SSDBM*, pages 32:1–32:4. ACM, 2013.

[6] M. Raasveldt and H. Mühleisen. Vectorized UDFs in Column-Stores. In *SSDBM*, pages 16:1–16:12. ACM, 2016.

[7] T. Vincenty. Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations. *Survey Review*, 23(176):88–93, 1975.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster Computing With Working Sets. *HotCloud*, 10(10-10):95, 2010.