ViDeTTe Interactive Notebooks

Konstantinos Zarifis
University of California, San Diego
zarifis@cs.ucsd.edu

Yannis Papakonstantinou
University of California, San Diego
yannis@cs.ucsd.edu

ABSTRACT

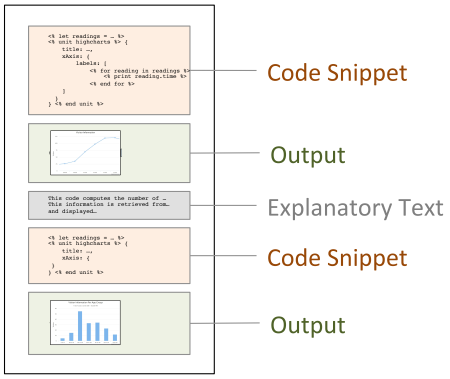
Interactive notebooks allow the use of popular languages, such as python, for composing data analytics projects. The interface they provide, enables data scientists to import data, analyze them and compose the results into easily readable report-like web pages, that can contain re-runnable code, visualizations and textual description of the entire process, all in one place. Scientists can then share such pages with other users in order to present their findings, collaborate and further explore the underlying data.

However, as we show in this work, interactive notebooks lack in interactivity and ease of use, both for the data scientist that composes the notebook and, even more so, for the reader of the resulting notebook. First, utilizing libraries that obtain, process and visualize data, requires technical expertise that often exceeds the skill-set of a typical data scientist. Second, while the user interface allows readers to rerun or extend the code included in the notebook, it does not allow them to directly interact with the generated visualizations in order to trigger additional computation and further explore the underlying data. This means that only code-literate readers can further interact with and extend such notebooks, while the rest can only passively read the provided report.

To address these issues, we propose ViDeTTe, an engine that enhances notebooks with capabilities that benefit both data scientists and non-technical notebook readers. ViDeTTe uses a declarative language to simplify data retrieval and data visualization for analysts. The generated visualizations are capable of collecting the reader's input and reacting to it. Additionally, the user input is utilized by ViDeTTe's propagation algorithm, to identify subsequent parts of the notebook that depend on it and cause their reevaluation. By doing this, ViDeTTe offers enhanced data exploratory capabilities to readers, without requiring any coding skills.

1. INTRODUCTION

The database, HCI and systems communities have proposed tools that simplify common tasks of analytical processes. Such tools, however, either focus on only individual stages (such as data retrieval or visualization) of a much bigger analytical pipeline or they focus on particular use cases (for instance pattern matching, finding correlations and so on). Despite their remarkable success, data analysts often want the flexibility that comes from combining diverse tools and computation libraries in order to perform an analysis. This need often pushes code-literate data analysts to the tried-and-true interactive notebooks, such as Jupyter [9].



```
<% let readings = … %>
<% unit highcharts %> {
    title: …,
    xAxis: {
        labels: [
            <% for reading in readings %>
                <% print reading.time %>
            <% end for %>
    ]
  }
} <% end unit %>
```

```
This code computes the number of …
This information is retrieved from…
and displayed…
```

```
<% let readings = … %>
<% unit highcharts %> {
    title: …,
    xAxis: {
 }
} <% end unit %>
```

Figure 1: Notebook interface

Interactive notebooks allow the use of popular, highly expressive imperative languages, such as Python, for describing tasks such as data retrieval, processing and visualization, all in one platform. Due to the popularity of the languages they support, there is a massive collection of third-party utility libraries that can be used to facilitate such tasks. Furthermore, the web environment of interactive notebooks enables collaboration between data analysts, since it allows them to develop and run code that processes data and generates visualizations directly on the browser. Lastly, after completing an analysis, data analysts can compose their findings into an interactive report-like page, that contains re-runnable code, visualizations and textual description of the analysis, which can also be shared with non-technical users. Figure 1, shows the interface of Jupyter notebooks. It comprises a sequence of blocks that are created "on-demand" by the data analyst, each block can contain explanatory text and re-runnable coding snippets that once evaluated print the result (which could be a visualization or plain text) into the subsequent block.

However, interactive notebooks are still suboptimal with regard to ease of use and productivity for data analysts. Retrieving, processing and visualizing data involves reading lengthy documentation pages that describe each library's programmatic API and writing complex imperative plumbing code in order combine their functionality. Both these steps, are not only very time consuming but also require technical expertise that exceeds the skill-set of a typical data analyst.

Besides the aforementioned limitations that impact data analysts, notebooks also have limitations that impact the, potentially non-technical, readers of the published notebooks. Particularly, while notebooks can contain visualizations that showcase important aspects of a data analysis, the readers cannot interact with the provided visualizations in the same way they could if these visualizations were part of a typical OLAP dashboard application. Specifically, even if the analyst that composed the notebook used third-party web-based visualization libraries that support user in-

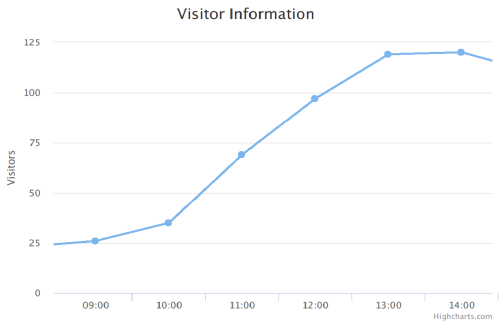


Figure 2: Line chart showing visitors per hour

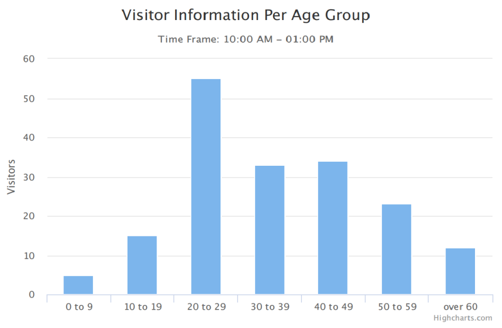


Figure 3: Bar chart showing age groups of visitors

Page Views					
id	vid	url	time	date	revenue

Visitors					
vid	name	lastname	username	age	gender

Table 1: Schema description of the two tables in our database.

teraction, this interaction can only cause local changes to the visualization the reader interacts with and cannot trigger additional data retrieval, computation or mutations to other visualizations that are included in the notebook. For this reason, only code-literate readers, capable of extending the notebook with more coding snippets, are able to further explore the underlying data (or examine additional hypotheses about the underlying data), while the rest are limited to passively reading the generated notebook.

Costas: Does this point belong to both the reader and the data analyst of the notebook? Regardless of the techinical expertise of the analyst, building an OLAP-like dashboard just cannot be done in a notebook.

Contributions. In order to resolve these issues, we extend Jupyter interactive notebooks with the ViDeTTe notebook engine. The main contributions of this extension are: (a) visual and data retrieval constructs, that simplify the process of data retrieval and visualization, for data analysts, (b) A declarative template language that can be used to declaratively describe computation, data transformations, user input collection, (c)A propagation algorithm, capable of propagating changes from visual and data retrieval constructs to subsequent such constructs and Python coding blocks, thus introducing a truly reactive behavior to notebooks. These reactive capabilities, enable non-technical, code illiterate readers to further explore the underlying data.

2. RUNNING EXAMPLE

In order to illustrate the issues with existing notebooks and describe the extensions in ViDeTTe we will use the following example of a potential analysis:

EXAMPLE 2.1. *Consider a data analyst, working for a news portal. The analyst wishes to convince the lead editor of the portal that by publishing more articles and advertisements that pertain to particular audiences during specific time-windows of the day, they can maximize the portal's revenue. In order to achieve this, the analyst intends to obtain and plot the number of readers that visit the website during the day; then for various time-windows she would like to obtain information about the readers' demographics (for instance, the visitors' age groups) and plot the result. Lastly, she would like to compute the portal's actual and predicted revenue (using a linear regression model) and plot the the results, side-by-side, thus illustrating whether there is room for improvement.*

For this analysis, we assume that the news portal maintains information about its reader-base in a Postgres database. Table 1 shows a potential schema, that could be used for storing visitor information. The database contains two tables, namely "Page Views" and "Visitors"; table "Visitors" contains information about each reader, such as the reader id, name, lastname, username, age and gender. The table "Page Views" maintains a tuple for each visit, and it consists of a visit id, the visitor id (foreign key referencing the visitor), the url of the visited page, the hour and date in which the user visited the website and the revenue that was collected during this visit. The reader information could have been retrieved, with their permission, from various social media services (such as Facebook, Google etc). In order to construct this notebook, the data analyst must (a) retrieve website access information from the database, by joining the two tables on the visitor id (b) generate a plot that shows the number of users that visit the website during the day, (c) issue a query that counts the number of visitors per age group, for each potential set of selected hours, (d) create a bar chart that shows the number of visitors per age group, (e) use an already trained predictive model in order to predict the expected revenue for the selected time window and (f) plot a bar chart, showing the actual and the predicted revenue generated in selected time window.

3. TRADITIONAL NOTEBOOKS

We will now describe how a data scientist would perform this analysis with a traditional interactive notebook that uses Python. The description will show the issues that arise.

Interacting with third-party utility libraries: Productivity and security concerns.

In order to retrieve website access information from the database, the data analyst, needs to employ third-party Python drivers. Employing such drivers, involves reading lengthy documentation pages that describe the exposed API calls of the respective driver. After identifying the appropriate API calls, the analyst needs to use imperative logic that interacts with the library, in order to issue the appropriate query. {During this step, the analyst has to also specify the credentials that will be used for accessing the database system. In most such libraries, the credentials have to be inlined into the method call that establishes the connection with that system. While this would not be a security concern in a python (or any other) application (since the code that includes the credentials would have been compiled into a binary file, thus hiding them), in a python notebook, the credentials would lie in plain sight for anyone, who has access to the notebook, to see. This secu-

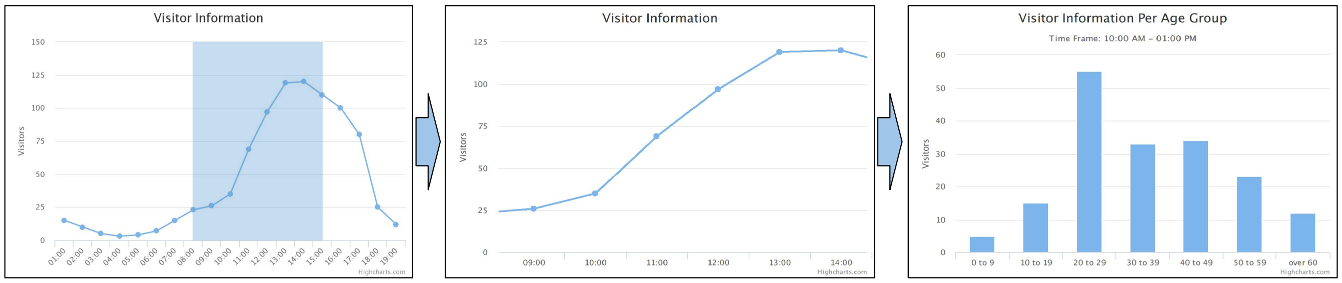


Figure 4: Demonstration of reactive charts. The reader's selection automatically updates both charts.

rity concern is also resolved in ViDeTTe.} Costas: Is this point weak? Should I remove it?

Data conversions. After, establishing the connection with the database system, and issuing the queries, the analyst, is able to consume the results by using the internal, to the library, data types. After doing so, the analyst has to again read documentation pages in order to infer the data types required by the visualization library she selected, and manually perform the appropriate conversions in order to construct the first visualization, by invoking the appropriate renderer function. Note that data conversions have to take place every time the analyst wishes to integrate a third-party library in her analysis, so the amount of plumbing code required, can quickly skyrocket.

Limited interactive exploratory capabilities. The next step, is for the analyst to issue a query that counts the visitors per age group for a set of selected hours and construct a bar chart showing the result. The obtained dataset will also go through a machine learning package (for instance, the scikit-learn package [14]), that employs a linear regression model to predict the expected revenue during the selected hours. The predicted revenue will be plotted next to the actual revenue produced during these hours. Note that selecting a meaningful time frame, is of utmost importance, because it will affect all the remaining steps of this analysis, and ultimately the insight, the portal editor will gain, about potential ways to increase the revenue.

Unfortunately, there is no straightforward way for selecting a meaningful time frame. The analyst will have to go through a process of trial and error, by issuing an arbitrary number of such queries and plotting the results, until she finds a set that produces valuable insight. Furthermore, even if the analyst goes through this process for the data corresponding to a particular day, the time frame selection she made, might not produce any valuable insight for the dataset corresponding to another day. Most importantly the non-technical reader cannot use the notebook to test similar hypotheses, she is limited to passively reading the ones that were hardcoded by the analyst. This aspect of introducing the human in the loop is improved dramatically with ViDeTTe notebooks.

4. REACTIVE NOTEBOOKS

Ideal reactive behavior and functionality. Co-dependent reactive charts are tremendously useful in this scenario. If the analyst was able to introduce a chart that allows the reader to select a particular range of hours, by directly interacting with it, and use that input to retrieve and plot only the user demographics for this particular range, she could enable the reader to further explore the underlying data without requiring any code literacy. Figure 4 graphically depicts this process (note that the charts that appear side-by-side in Figure 4 actually appear after the respective coding blocks that generated them in the notebook). The first figure shows the reader of the notebook, selecting a particular time frame; this causes the line chart to zoom in, thus showing only the selected time frame. After the reader performs the selection on the first chart, the subsequent bar chart is also updated thus only showing the age groups of the users that visited the portal during the selected hours. This feature adds useful exploratory capabilities to the notebook, which is of great value to readers. It enables them to further analyze the underlying dataset used by the notebook by simply interacting with the provided visualizations.

This kind of reactive behavior cannot be expressed in traditional interactive notebooks. It is important to note that this feature cannot be implemented currently in interactive notebooks. Instead, the analyst would have to implement a full blown web application in order to enable non-technical users to interact with visualizations in this way. This requires more time and effort as well as technical expertise with web frameworks that data analysts, might lack. This reactive behavior requires the implementation of actions that have to be invoked when particular mouse events take place on the pixels of the browser that correspond to the visualization. Additionally, such events have to take place in a specific order (mousedown-mousemove-mouseup) for the action to be invoked. The developer of such applications must install observers that listen for such events and then provide the imperative application logic that asynchronously accesses back-end databases to retrieve new data based on the user's selection, process them appropriately and cause mutations to the respective visualizations that depend on them.

YannisP: IMPORTANT: All figures have the problem that they do not look like interface pages, which sends the clear message to the reviewer that you have not even started yet. Make figures that show how the INTERFACE looks on the browser. *Costas: Should I replace the figures with this:* `https://czarifis.github.io/vidette-prototype/Thesis%20Proposal`

5. VIDETTE NOTEBOOKS

ViDeTTe notebooks provide constructs that simplify the process of building truly reactive notebooks. Specifically, they support two types of modules that streamline data access and visualizations, namely: **source wrappers** and **visual units**. These modules, enable data access and visualization without requiring any imperative logic or individual API invocations. Additionally, ViDeTTe introduces a **declarative template language**, that facilitates schema conversions between visual units and source wrappers, as well as invocation of reusable Python functions (that can describe an unbounded number of computations). Lastly, templates interface with visual units to allow for data collection from user input. The collected variables can be used in subsequent parts of the notebook, thus describing parameterized queries, computations or visualizations. When mutations are observed by ViDeTTe in Python variables (either as a result of user interaction, or due to changes in underlying source), a change propagation algorithm, ensures the propagation of these mutations to all dependent parts of the notebook, thus adding a reactive element to notebooks.

```
<% let readings =
   SELECT count(time) as visits, time
   FROM (SELECT * FROM page_views pv
               join visitors v
         on pv.v_id = v.vid) AS joined_table
   GROUP BY time
   ORDER BY time ASC %>;
```

(a) Data retrieval

```
readings = [
   {visits: 15, time: '08:00'},
   {visits: 10, time: '09:00'},
   {visits: 25, time: '10:00'},  ...]
```

(b) Query Result

```
sources : [ {
     driver   : "postgres",
     host     : "edu.db.domain",
     expose   : [ {
      schema : 'website_info',
      tables: [visits, page_views] } ]
     port     : 5432,
     username : "dbadmin"
     password : 'myP@ss'
   }]
```

(c) DB Access Configuration file

```
1   <% unit highcharts %> {
2     title: 'Visitor information' ,
3     type: 'line',
4     xAxis : {
5       labels : ['08:00','09:00'...],
6       min : '08:00'
7       max : '22:00'
8     }
9     series: [{ data: [ {y:15}, {y:10}...] }]
10  } <% end unit %>
```

(d) Unit with evaluated unit state

```
<% unit highcharts %> {
  title: 'Visitor information',
  type: 'line',
  xAxis : {
    labels : [
      <% for reading in readings %>
        <% print reading.time %>
      <% end for %>],
    min : <% bind min_time %>,
    max : <% bind max_time %>
  }
  series: [{
    data: [ <% for reading in readings %>
          {
            y  : <% print reading.count %>
          }
      <% end for %> ]
  }]
} <% end unit %>
```

(e) Template `temp_view`

Figure 5: Declarative Data retrieval, DB Configuration, evaluated unit state and Template describing unit state for running example

Source wrappers

Source wrappers allow the use of source specific languages for querying database systems (for instance SQL can be used to query MySQL and PostgreSQL databases, N1QL for Couchbase etc.,). When using a ViDeTTe notebook, data analysts can directly issue queries from the notebook interface. This query is directed to the appropriate source wrapper, that evaluates it, propagates it to the respective data base system and assigns the result to a Python variable. This variable can later be used in subsequent parts of the notebook (as will be described later in this section). Figure 5a contains the first query of the analysis that retrieves the dataset required to plot the chart shown in Figure 2. Specifically, the query joins the two tables: $visitors$ and $page\_views$ on the id of the visitor, then groups the result on the $time$ attribute and runs a $count$ aggregate to count the visitors per unit of time. Lastly, it sorts the resulting dataset by time in ascending order and assigns the result the $readings$ variable. Once the respective active function evaluates the query, it obtains and automatically parses the result into a Python array of dictionaries (used to maintain the tuples), as shown in Figure 5b. Note, that the analyst no longer needs to waste any time reading documentation pages that describe how to issue queries and parse the results, all these steps are handled automatically by the respective source wrapper.

In order for source wrappers to connect to the respective database systems, the data analyst must also provide a configuration file, describing the information required for establishing a connection with each source. Such configuration files are added from the interface of the notebook (into a configuration file coding block). Once added, these files are hidden from the UI and encrypted thus diminishing any security concerns. *Costas: mark-start:* Figure 5c shows a sample configuration file that is used for accessing a postgres database that contains the data that will by used in our analysis. The configuration file must include the type of the database system, the host name, port and the credentials that will be used for obtaining a connection. Additionally, it must contain the database tables that will be accessible by queries. Only the tables that are explicitly defined in the configuration file will be accessible in the notebook. Once imported, this file is evaluated by the ViDeTTe engine and the schema of the accessible tables is displayed on the notebook, thus allowing the notebook user to get a glimpse of that information when generating the queries. *Costas: mark-end*

YannisP: aren't the configuration files virtually identical to those used by application servers in order to create virtual data sources? We spent too much space on it, turning the paper into manual. I'd like to cut and just point to the figure. Perhaps keep only the security and encryption aspect.

Costas: Agreed. Should I remove the marked content?

Visual Units

Visual units are constructs capable of generating fully reactive visualizations, that take advantage of the advanced interactive capabilities of modern browsers. In the eyes of an analyst a visual unit is simply a black box that takes as input a Python value that describes the state of the visualization. This value is called *unit instance*. The visual unit internally uses the unit instance to invoke the appropriate renderer calls that will generate the expected visualization, thus absolving the data analyst from having to perform these tasks manually. A particular instantiation of the unit can be described as **<% unit** U **%>** v **<% end unit %>**, where U the type of the visual unit and v the Python value corresponding to the unit instance. Figure 5d shows the unit instance of type `highcharts`, that once added to a coding block generates the visualization shown in Figure 2. The unit instance describes all the information that will be displayed in the visualization (such as the type and title of the chart, the labels on the x axis and so on). Each visual unit, comes with a unit instance schema that describes the format of the unit instance. As the reader interacts with visualizations, she can trigger mutations in the unit instance. For example, if the reader selects a particular area of the chart generated by the unit instance shown in Figure 5d, the min and max attributes (in lines 6-7) will be updated

accordingly. *YannisP: Two major problems:*
First, the figure "Template, template instance, and UAS configuration file for the running example" gives me no idea of how to put together an interactive notebook. Even before that, what is an interactive notebook? I see a "data retrieval" piece here, a "template" there, how do these pieces fit to the notebook? I'm lost.
Second, you trivialize the semantics. Is the semantics supposed to be MVVM? Is it supposed to be just a formatting tool for turning input into a certain format?

Costas: 1. I agree with this point. Is there a way to add this: `https://czarifis.github.io/vidette-prototype/Thesis%20Proposal%20Sample.html` *prototype into the paper? 2. Not sure how to answer this. The language with the constructs can describe data access, data processing/transformations and data visualization. The analyst doesn't have to know anything about the MVVM design pattern in order to create this analysis.*

Templates

Despite the fact that both source wrappers and visual units use Python variables, data scientists often need to perform additional computation or data transformations. In order to facilitate this process, without requiring the use of imperative logic, ViDeTTe provides a declarative template language that operates on Python variables. Note, that analysts are not forced to use this language, if they don't wish to. They can simply manually construct the same Python variables (using regular Python code) and invoke the appropriate visual units or source wrappers by providing the variables as arguments. The template language supports a set of *template directives*, all of which also operate on Python variables and simplify the task of creating nested data. Due to lack of space we do not include a formal definition of the grammar, instead we simply describe each of them in detail and provide a concrete example that illustrates their use.

Defining variables. A template may define variables that are added to the notebook's environment so that they can be used in subsequent statements. The **`<% let`** x **`=`** E **`%>`** directive defines variable x and assigns to it the result of the expression E. E can denote four types of expressions: (a) path navigation on nested Python variables, (b) invocation of a Python functions, (c) another subexpression containing directives that report data (defined in the next subsection) which generate nested values or (d) a source-specific language (such as an SQL query). For instance, the template shown in Figure 5a employs a **`let`** directive to create a variable `readings` containing the visitor information that will be displayed in the chart (retrieved from a relational DBMS through an SQL query).

YannisP: I stopped here

Reporting data. Value assignments and iterations over collections are specified by using the **`print`** and **`for`** directives. The **`<%print`** E**`%>`** directive evaluates the expression E and returns the result, while the **`<%for`** x **`in`** E**`%>`** B **`<%end for%>`** directive specifies that variable x iterates over the result of E and, in each iteration, it instanciates the body B of the **`for`** loop. Specifically, in Figure 5e, in lines 6-8 and 13-17 a **`for`** directive is used to iterate over the readings retrieved from the database and for each reading, it generates a new value, with the use of the **`print`** directive. Particularly, in line 7 the template generates a string (the time label), which is added to the labels array (which contains the labels that will appear on the x axis of the chart). In lines 14-16 it generates a Python dictionary of the form {y: ...} and adds it to the data array. The data array in highcharts contains the points that will appear in the chart.

Invoking Python Functions. A template can also describe the invocation of Python functions, thus allowing any arbitrary computation. For instance, the snippet shown in Figure 6f (which generates the bar-chart illustrating the predicted and actual revenue)

ALGORITHM 1: Change Propagation Algorithm

```
function change-propagation (Notebook N, Environment env, Set of Input Diffs D)
    for each statement s in each coding snippet of N do
        if s is <% let $x$ = $E$ %> then
            if There is $\Delta(t) \in D$ with t that targets any subexpression of E then
                Reevaluate $E$ and assign result $r$ to $x$;
                Update $env$ with new value of $x$;
                Construct $\Delta(x)$ and add it to D;
        else if s is <%unit $U$ %> $B$ <%end unit%> then
            if There is $\Delta(t) \in D$ with t that that targets any subexpression of B then
                Reevaluate $B$ and invoke unit U with result $r$ ;
        else if s is Python statement then
            if There is $\Delta(t) \in D$ with t that that targets any subexpression of s then
                if s is an assignment $x = E$ then
                    Reevaluate $E$ and assign result to $x$;
                    Update $env$ with new value of $x$;
                    Construct $\Delta(x)$ and add it to D;
                else
                    Reevaluate $s$ ;
    end
```

in lines 5 and 6 invokes two functions: the $predict\_revenue()$ and $sum\_revenue()$ by using the **`print`** directive while providing as input an array of dictionaries that is initialized in Figure 6a; function $predict\_revenue()$ (shown in 6c), inputs the array into a trained linear regression model and returns the revenue prediction, while function $sum\_revenue()$ (shown in 6d), simply computes the actual revenue.

Collecting data. The template's **`bind`** directive allows the analyst to specify user input collection. Specifically, the **`<% bind x %>`** directive describes a two-way binding between the part of the unit instance appearing on the left side of the directive and the variable x. Once the reader interacts with the generated visualization and causes mutations to the underlying unit instance, these mutations will be propagated to the respective bound variables. For instance, in Figure 5e in lines 9-10 we create a two-way binding between the min and max boundaries of the chart and the min and max values of the time labels that have been received from the database (namely min_time and max_time). As the reader interacts with the generated chart she can select a particular time frame by dragging and dropping the mouse over a region. This action updates the min and max attributes of the unit instance which in turn updates the values min_time and max_time.

Internal Data Model & Propagation Algorithm

Once a set of variables is mutated by a visual unit as a result of reader interaction, or by a source wrapper as a result of a push-event from an underlying database ViDeTTe is responsible for reflecting these changes to all other parts of the notebook that depend on those variables. For instance, in Figure 6a we show a parameterized query that collects all users that visited the website in the time-frame specified by min_time and max_time, categorizes them into age groups and lastly counts the users that correspond to each age group. The result of this query is assigned to the variable age_groups (Figure 6b shows the contents of this variable). The variable age_groups is then used in the template shown in Figure 6e (lines 7-9 and 12-16) to produce the bar chart appearing in Figure 3. In this scenario, if the reader's interaction with the chart shown in Figure 2, caused the mutation of variables min_time and max_time, then ViDeTTe must trigger a chain reaction that causes

```
<% let visits =
   SELECT * FROM page_views pv join visitors v
   on pv.v_id = v.vid where time BETWEEN
   <% print min_time %> and <% print max_time %>;
<% let age_groups =
   SELECT agegroup, count(*) AS total
   FROM (SELECT CASE
    WHEN age BETWEEN 0 AND 9 THEN '0 to 9'
    WHEN age BETWEEN 10 and 19 THEN '10 to 19'
     ...
    FROM  visits
   GROUP BY agegroup
   ORDER BY agegroup ASC %>;
```

(a) Data retrieval

```
age_groups = [
   {age_group: '0 to 9', total: 12},
   {age_group: '10 to 19', total: 67},
   {age_group: '20 to 29', total: 84},  ...]
```

(b) Query Result

```
def predict_revenue(input):
  y_pred = LinearRegression.predict(input)
  return sum(y_pred)
```

(c) Python function predicting revenue

```
def sum_revenue(input_list):
  return sum(item['revenue'] for item in input_list)
```

(d) Python function summing actual revenue

```
  <% unit highcharts %>
  {
    title: 'Visitor information',
    type: 'bar',
    xAxis : {
      labels : [
        <% for v in age_groups %>
          <% print v.age_group %>
        <% end for %>]
    }
    series: [{
      data: [ <% for v in age_groups %>
          {
            y  : <% print v.total %>
          }
        <% end for %> ]
    }]
  }
  <% end unit %>
```

(e) Template `temp_view`

```
<% unit highcharts %>
{title: 'Predicted and Actual revenue',
  ...
  data: [
    {y : <% print predict_revenue(visits) %> }
    {y : <% print sum_revenue(visits) %> }
  ]...}
```

(f) Actual and Predictive revenue visualization

Figure 6: Declaration of "reactive" parameterized view, data visualization specifications and Python function declaration and invocation

the reevaluation of both the query and the following visualization in order to reflect the changes. We will now describe this process.

Internal Data Model - Diffs

Once a variable is mutated by a unit, the ViDeTTe engine produces a diff that describes this mutation. A *diff* is the internal data model that is shared between source wrappers, visual units and the change propagation algorithm, this data model is hidden from data analysts and notebook readers. A diff that targets a nested value is of the form $\Delta(\hat{p})$, where $\hat{p}$ is the path to the value that is being modified.

Change Propagation Algorithm.

Once a diff is generated for each mutated variable ViDeTTe invokes the change propagation algorithm. Algorithm 1 summarizes this algorithm; it takes as input a set of diffs, and only invokes the reevaluation of statements that depend on the targeted variables. Specifically, it iterates over each template statement of each coding block that appears in the notebook (as shown in line 2) and performs the following checks:

- If the statement defines a new variable x (line 3), then the algorithm visits every sub-expression of the **`let`** directive and checks if it is targeted by any diff in D (line 4). If there is, the change propagation algorithm causes the reevaluation of the **`let`** statement (line 5), updates the notebook environment with the new value of x (line 6) and lastly constructs a new diff that describes the mutation to the variable x (line 7). Note that the Expression shown in Figure 6b is an SQL statement that contains subexpressions (parameters), so given a diff of type: $\Delta(min\hat{\_}time)$ (which could have been created as a result of the user's interaction with the first chart), the propagation engine will identify that there is a subexpression that is targetted by that diff and cause its reevaluation.
- If the statement constructs a visualization using a visual unit of type U with body B (line 8), then the algorithm visits every sub-expression in body B and checks if it is targeted by any diff in D (line 9). If there is, the change propagation algorithm causes the reevaluation of the entire body B, thus generating the unit instance r and then reinvokes the unit U with new unit instance (line 10). *Costas: add points about source wrappers for parameterized views and about python coding blocks*
- If the statement is a typical Python expression, the algorithm checks if there is a diff that matches any subexpression of it. If there is such a diff and this statement is an assignment, then the expression is evaluated in full, the environment is updated accordingly and a diff that targets the target of the assignment is created, in all other cases the statement is fully reevaluated.

6. RELATED WORK

Costas: add plotly and Bokeh

The specific template language used in our project, follows heavily the syntax of templates in the FORWARD project [6, 7]. However, the semantics are different in order to fit the needs of notebook operation. Namely, the templates operate as scripts that produce nested Python instances for the units and fully redisplay them in each round. Unlike FORWARD, it is predetermined which units will be shown on each snapshot of the notebook. Thus, at the present state, it is not yet possible to create an interface that is tantamount to, say, a data cube exploration - which is something that would be possible in FORWARD. On the positive side, ViDeTTe (unlike FORWARD) does not require the analyst to understand the notebook as an MVVM application, whereas in each snapshot the framework visualizes a page with potentially different units and/or partially modified unit instances.

Additional prior work related to ViDeTTe notebooks can be classified into the following categories:

Grammar-based visualizations. The concept of using a formal grammar to specify visualizations can be traced back to Wilkinson's "The Grammar of Graphics" [21]. Since then multiple tools, such as Polaris [18] (later commercialized as Tableau), ggplot2 [20], Vega [15, 19] and others [2, 4, 8] have adopted this approach, thus enabling the description of visualizations (as well as the user interaction with such visualizations [16, 22]) in a terse and declarative manner. While the expressiveness of the employed grammars varies, their main focus is to effectively describe the specifics of the visualizations that will appear (for instance the dimensions of the chart, the colors, etc.), and not to describe data access or data processing and/or conversions. For that reason, such tools do not provide constructs capable of retrieving data from various sources or performing data transformations. Instead, the user has to manually perform these tasks and convert the dataset she wants to visualize into the format that is expected by the respective tool in order to generate the visualization.

Data exploration. Tools that use the human factor as an integral part of data exploration have gained in popularity in the past years. Such tools provide a simple to use way that describes both computation and visualizations, thus providing the required toolkit to effectively explore data and steer computation. Some of these tools enable this process by providing a declarative language capable of performing such tasks (such as Zenvisage [17] and Devise [13]), while others (such as Vizdom [3], DataHub [11] and more[1, 5, 10, 12, 23]) rely on the user interaction with a front-end, in order to compose the appropriate queries. As a result, the former tools assume a type of user with deep understanding in databases and query languages, while the latter offer a more user-friendly way of performing an analysis but at the same time allow a predetermined set of possible analyses. ViDeTTe notebooks bridge the gap between these two types of users; to a data analyst it provides constructs that allow rapid creation of notebooks for an arbitrary number of analyses, while to a non-technical user it provides the capability to interact with the produced visualization in order to further explore the underlying data.

7. CONCLUSION

We presented ViDeTTe an interactive notebook engine, that enhances interactive notebooks with capabilities that pertain to both data scientist that create notebooks and non-technical notebook readers. Specifically, ViDeTTe streamlines arduous tasks that usually have to be performed by data analysts, that compose such notebooks, while at the same time offering enhanced data exploratory capabilities to the readers of the notebooks

REFERENCES

[1] A. Bhardwaj, A. Deshpande, A. J. Elmore, D. Karger, S. Madden, A. Parameswaran, H. Subramanyam, E. Wu, and R. Zhang. Collaborative data analytics with datahub. *Proceedings of the VLDB Endowment*, 8(12):1916–1919, 2015.

[2] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE transactions on visualization and computer graphics*, 15(6):1121–1128, 2009.

[3] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: interactive analytics through pen and touch. *Proceedings of the VLDB Endowment*, 8(12):2024–2027, 2015.

[4] D3: Data-driven documents. `https://d3js.org/`.

[5] M. Derthick, J. Kolojejchick, and S. F. Roth. An interactive visual query environment for exploring data. In *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 189–198. ACM, 1997.

[6] Y. Fu, K. Kowalczykowski, K. W. Ong, Y. Papakonstantinou, and K. K. Zhao. Ajax-based report pages as incrementally rendered views. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 567–578. ACM, 2010.

[7] Y. Fu, K. W. Ong, Y. Papakonstantinou, and M. Petropoulos. The sql-based all-declarative forward web application development framework. In *CIDR*, pages 69–78, 2011.

[8] ggvis: Interactive grammar of graphics for r. `http://ggvis.rstudio.com/`.

[9] Jupyter. `http://jupyter.org/`.

[10] N. Kamat, E. Wu, and A. Nandi. Trendquery: A system for interactive exploration of trends. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 12:1–12:4, New York, NY, USA, 2016. ACM.

[11] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.*, 9(12):948–959, Aug. 2016.

[12] E. Liarou and S. Idreos. dbtouch in action database kernels for touch-based data exploration. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1262–1265, 2014.

[13] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. Devise: integrated querying and visual exploration of large datasets. In *ACM SIGMOD Record*, volume 26, pages 301–312. ACM, 1997.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2017, 2015.

[16] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2016.

[17] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. G. Parameswaran. zenvisage: Effortless visual data exploration. *CoRR*, abs/1604.03583, 2016.

[18] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002.

[19] Vega, a visualization grammar. `http://trifacta.github.io/`.

[20] H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer Science & Business Media, 2009.

[21] L. Wilkinson. *The Grammar of Graphics (Statistics and Computing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[22] Y. Wu, J. M. Hellerstein, and E. Wu. A devil-ish approach to inconsistency in interactive visualizations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, HILDA '16, pages 15:1–15:6, New York, NY, USA, 2016. ACM.

[23] K. Zoumpatianos, S. Idreos, and T. Palpanas. Rinse: interactive data series exploration with ads+. *Proceedings of the VLDB Endowment*, 8(12):1912–1915, 2015.