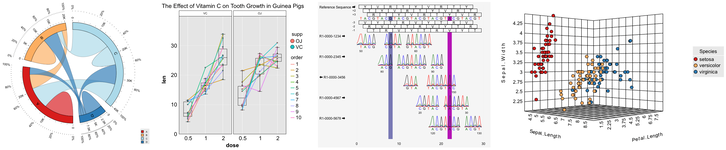
Introducing CanvasXpress for Python!

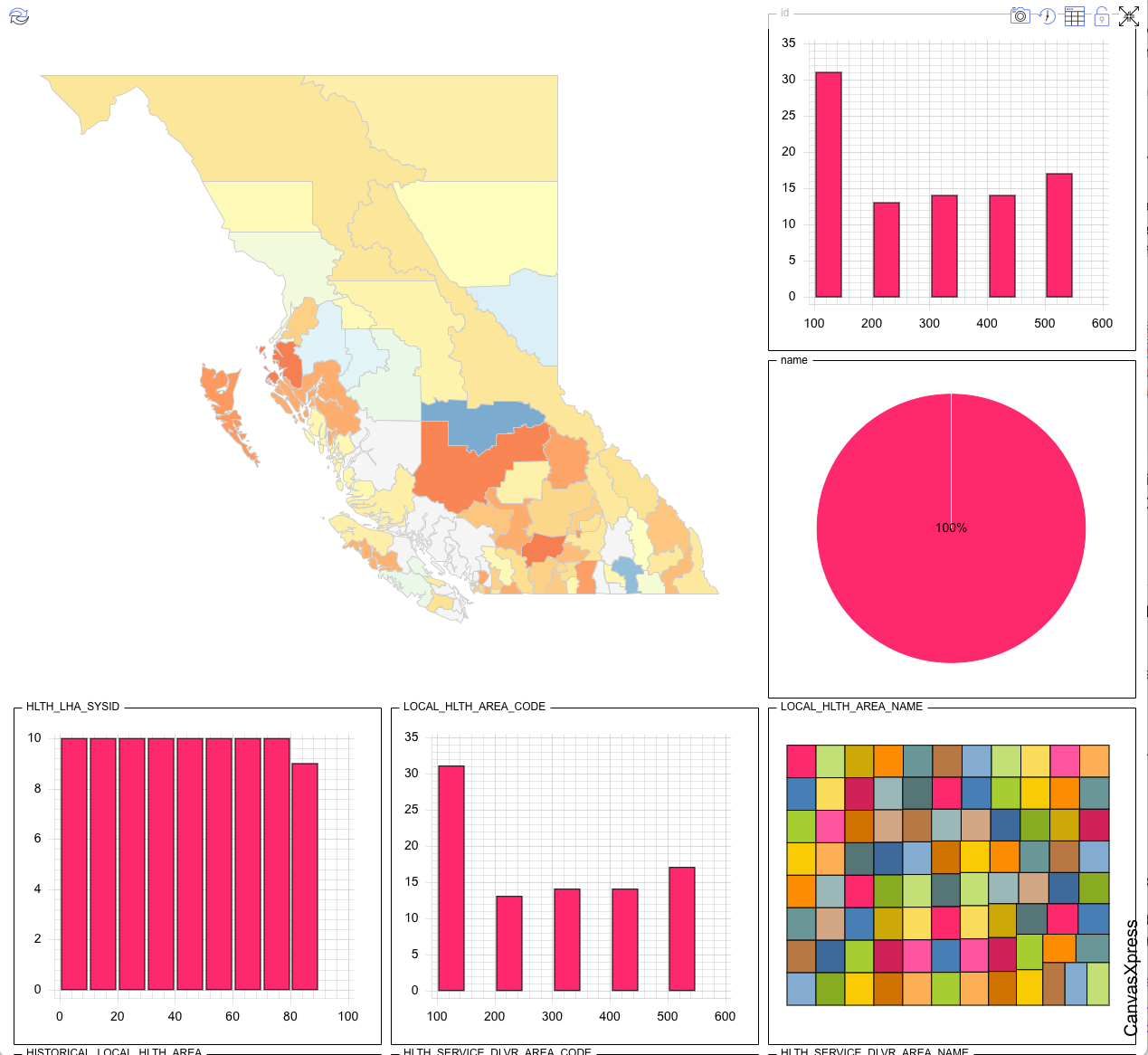
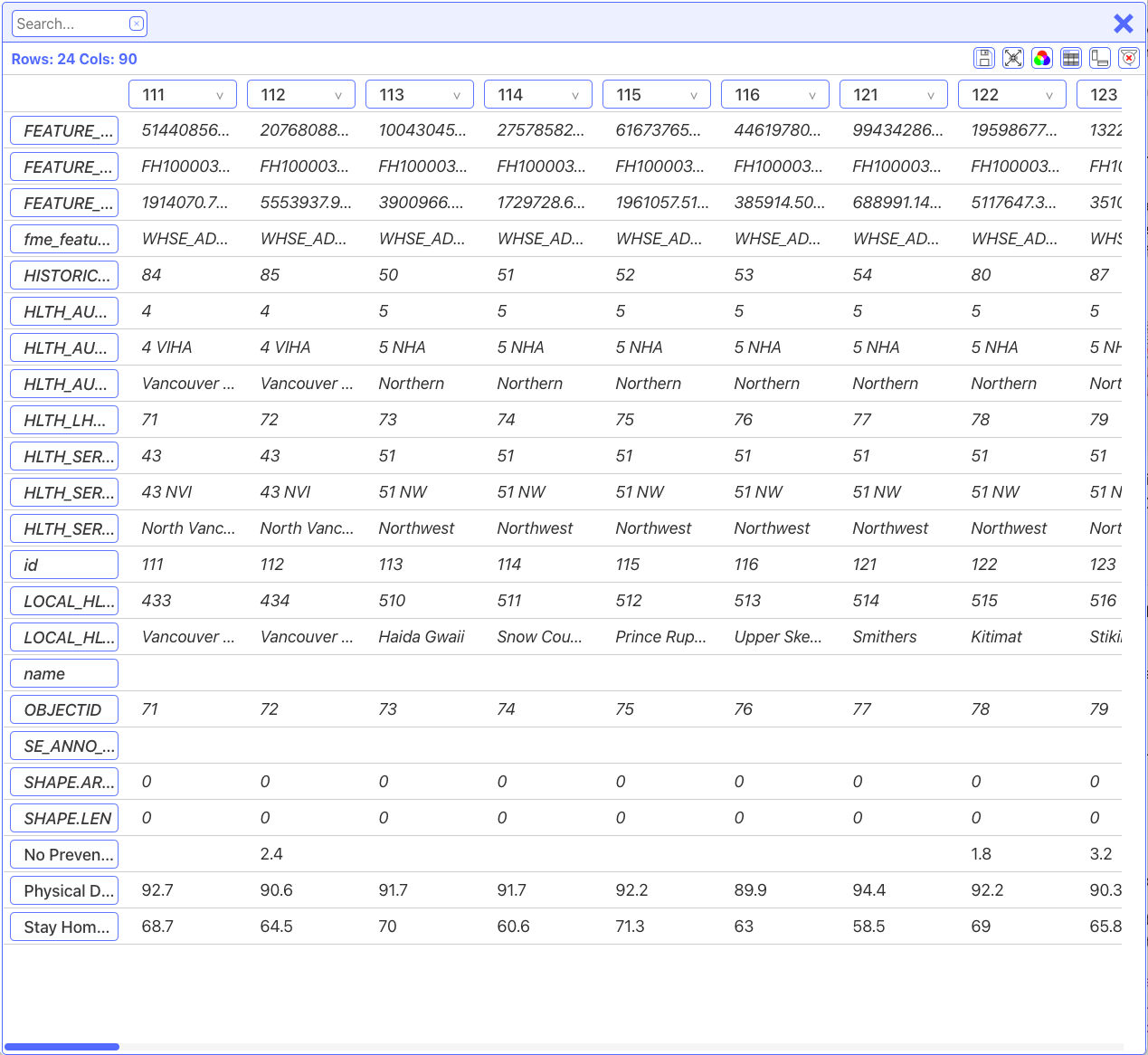
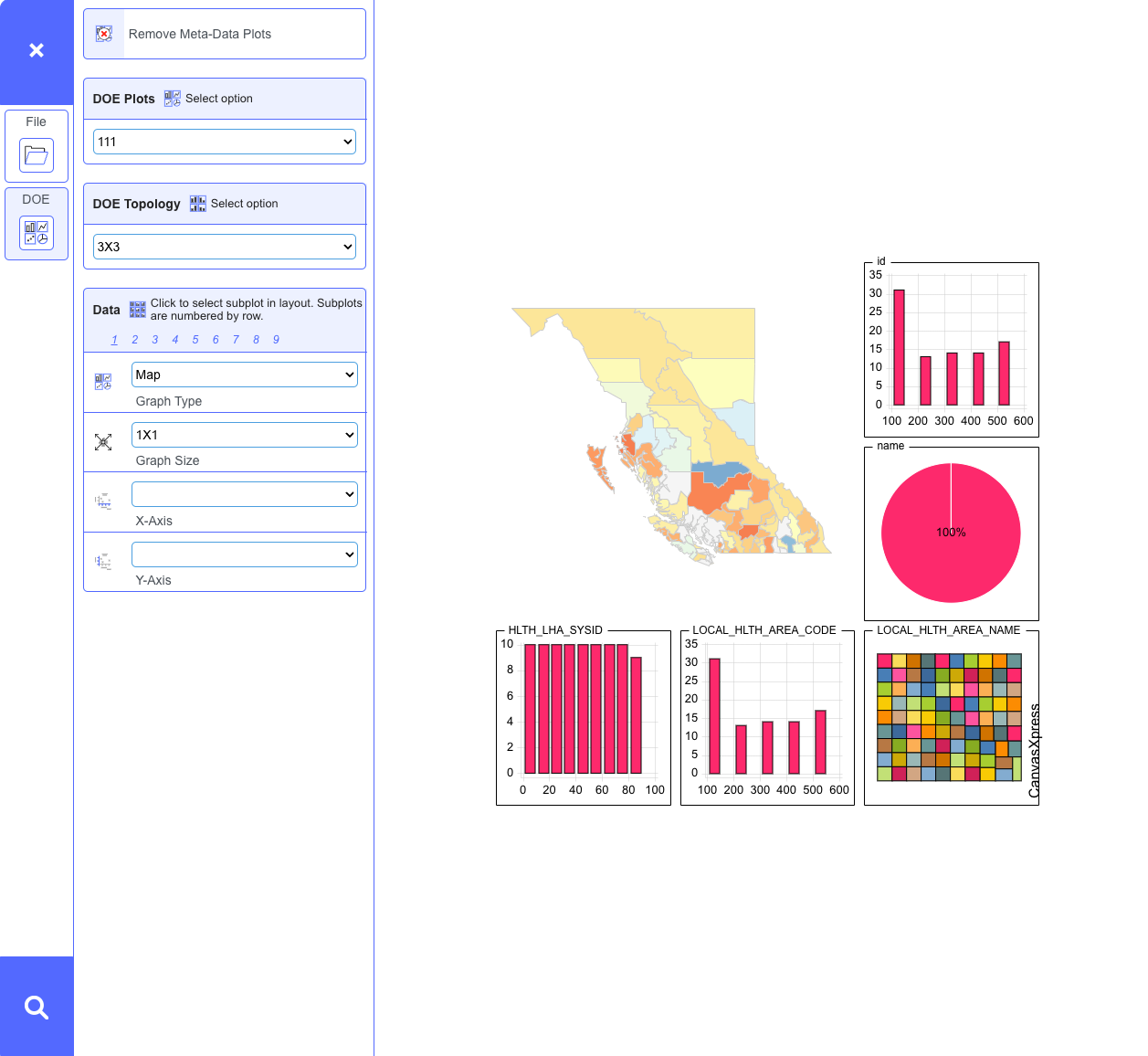
## A High Performance Plotting Library

I’m pleased to introduce [CanvasXpress for Python](https://pypi.org/project/canvasxpress/): an extension to the highly performant [JavaScript Library](https://canvasxpress.org) developed by Dr. Isaac Nuehaus for producing chart illustrations and facilitating data exploration.

Scientific computing often requires that very large data sets be visualized to facilitate analysis and research. Unfortunately, most modern charting libraries for Web browsers use methods that cannot efficiently illustrate, if even at all, data sets larger than a few thousand data points. CanvasXpress, which was originally developed as the core visualization component for bioinformatics and systems biology at Bristol-Myers Squibb, uses a very different approach to rendering that enables it to produce beautiful, interactive illustrations at scale—for example, million-point heat maps. CanvasXpress does this for a rich variety of chart types, examples of which are presented here:



Upon being rendered in a Web environment, the charts can then be explored in detail using a variety of tools. The user can use menu and configuration tools to review the data table; segregate or group data fields; shape or color points based on primary or meta properties; or even re-illustrate the chart altogether, such as changing a bar chart into a tree map. Many of these exploration features would normally require the use of standalone and potentially expensive applications, such as Tableau, but CanvasXpress treats interactive data review as a first-class feature standard in every chart. Here is a dashboard chart with its data table and configuration tools presented:

Not only can each chart be explored in detail, but charts on the same page broadcast changes to their peers. For example, if two or more charts on the same page use the same data set then data selection in one chart will update peers so that the same data is selected in the other charts—even if the types of charts are different. This is a powerful means by which complex data analysis can be facilitated by breaking complex data into simpler charts that better highlight related but illustratively unique data characteristics from the overall set.

## Platform Native Approaches

Our company, [Aggregate Genius Inc.](https://www.aggregate-genius.com/) specializes in custom analytics solutions for the scientific and business communities, and of course visualizations for complex data is a key part of our work. CanvasXpress’ ability to rapidly illustrate very large data *and* provide a comprehensive interactive environment is perfectly aligned to our client’s needs, and as such we have worked with Dr. Nuehaus over the years to both maintain the R package and develop a Python package to ease integration with the world’s most popular analytics platforms.

The Python edition was first released to [PyPI](https://pypi.org/project/canvasxpress/) in April 2021, and since that time we have expanded its functionality to support contexts such as general Python environments, Jupyter Notebooks, Flask, Plotly’s Dash, and Snowflake’s Streamlit. We continue to refine the package based on our use and the community’s feedback to simplify integration, configuration, and adoption. With such broad support for popular frameworks available, it seems appropriate to highlight CanvasXpress for Python to the greater community so that every Pythonista can take advantage of this excellent option for scientific and business data presentation and exploration tool!

The rest of this article will show how CanvasXpress can be added to a Python project, integrated into a Jupyter Notebook, and customized for visual and interactive interest. Follow-up articles will go into depth for platform-specific integration, such as Dash, and the CanvasXpress PyPI homepage provides an [example chart for each platform](https://pypi.org/project/canvasxpress/).

## Installing CanvasXpress for Python

CanvasXpress for Python is maintained at PyPI and github:

* PyPI: https://pypi.org/project/canvasxpress/
* Github: https://github.com/docinfosci/canvasxpress-python

As of October 2023, renderers are provided for:

* A standard HTML output, such as what would be used by Flask
* Jupyter Notebook / Lab
* Plotly Dash
* Snowflake Streamlit
* A local pop-up Web browser

We are also developing a Shiny for Python edition, which we hope to release later this year.

pip is supported for package installation. To manage third party dependencies, which can become quite large depending on the platform, CanvasXpress for Python can be installed using one of several profiles:

* default minimal package (CLI, flask, etc.): pip install canvasxpress
* all profiles in one: pip install canvasxpress[all]
* Jupyter specific: pip install canvasxpress[jupyter]
* Plotly Dash specific: pip install canvasxpress[dash]
* Snowflake Streamlit specific: pip install canvasxpress[streamlit]

This article will focus on the Jupyter package while touching on other platforms, and future articles will discuss in detail the use of CanvasXpress in popular dashboard environments. The PyPI project page also includes examples for each alternate profile and platform.

## The Essential CanvasXpress Chart

Let’s begin by considering a simple example. Given this minimal tabular (matrix) data set:

|  | Sample 1 | Sample 2 | Sample 3 |
| --- | --- | --- | --- |
| Amount | 33 | 44 | 55 |

We can create a bar chart:

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Great! Let’s now explore how we can use CanvasXpress for Python to create our own charts.

## Data

CanvasXpress generally works with data conceptually organized in a tabular (*aka* matrix) structure, but in many cases broken down into a JSON object. First, consider a simple matrix:

|  | Sample 1 | Sample 2 |
| --- | --- | --- |
| variable 1 | value | value |
| variable 2 | value | value |

CanvasXpress treats each column as a data sample. For example, in a demographic survey we might have samples for Canada, the United States, and Mexico. Each row represents a variable being tracked across the samples. In our demographic survey we might be interested in the population and birth rates.

Reworking our matrix example:

|  | Canada | The United States |
| --- | --- | --- |
| population | 38,250,000 | 329,500,000 |
| birth rate | 1.4 | 1.64 |

CanvasXpress translates that into JSON using object properties data, samples (smps, effectively the columns), and variables (vars, effectively the rows). These properties get put into a root level object property referenced as y:

{  
 "y": {  
 "data": [  
 [38250000, 329500000],  
 [1.4, 1.64],  
 ],  
 "smps": ["Canada", "The United States"],  
 "vars": ["population", "birth rate"],  
 }  
}

data is a list of lists: a master list representing the data matrix and each sub-list representing a row of the matrix. smps describe each column, or put another way the ordered list items in each sub-list. vars identifies each sub-list in order. It’s a good idea to use explicit identifiers for smps and vars when learning how to create charts, but CanvasXpress can in reasonable instances assign default values for these if omitted from the JSON.

(Note that there are also specialized JSON structures for Venn, Network, and Genomics charts, but as those are specialized use cases we will address them in a future article.)

In terms of chart data, anything that can be saved using the default json.dumps() function can be placed into the dict representing the JSON. Of course, it’s possible to write translating extensions for Python’s dumps() function, but unless the resultant value is native to JavaScript JSON such an approach will probably not be of use as CanvasXpress only understands JavaScript types.

Although we’re focusing on the JSON object, we can start with other data formats and either provide such directly to CanvasXpress for Python or convert them into a JSON compliant dict. For example, let’s create a Pandas DataFrame and convert it into a compliant dict:

import json  
from pandas import DataFrame  
  
# Establish a DataFrame based on our nation matrix from earlier.  
df\_example = DataFrame(  
 data=[  
 ["Canada", 38250000, 1.4],  
 ["The United States", 329500000, 1.64],  
 ],  
 columns=["country", "population", "birth rate"],  
)  
  
# Get the values as appropriate from the DataFrame and place them into a dict.  
dict\_conversion = {  
 "y": {  
 "smps": df\_example["country"].values.tolist(),  
 "vars": ["population", "birth rate"],  
 "data": df\_example[["population", "birth rate"]].values.tolist()  
 }  
}  
print(json.dumps(dict\_conversion, indent=2))

{  
 "y": {  
 "smps": [  
 "Canada",  
 "The United States"  
 ],  
 "vars": [  
 "population",  
 "birth rate"  
 ],  
 "data": [  
 [  
 38250000.0,  
 1.4  
 ],  
 [  
 329500000.0,  
 1.64  
 ]  
 ]  
 }  
}

CanvasXpress for Python also provides support for importing DataFrame, JSON text, CSV text, and other formats:

from canvasxpress.data.matrix import CXDataframeData  
  
...  
  
df\_example = DataFrame(  
 data=[  
 ["Canada", 38250000, 1.4],  
 ["The United States", 329500000, 1.64],  
 ],  
 columns=["country", "population", "birth rate"],  
)  
  
# You can create a CX-P data wrapper object:  
data = CXDataFrame(df\_example)  
  
# Or reference the DataFrame directly when specifying the chart:  
chart = CanvasXpress(  
 data = df\_example,  
 ...  
)

But in this introductory article we will focus on the dict object for the remainder of the examples.

## Configuration

CanvasXpress formats charts using a configuration JSON. In this case, a key-value relationship is used. For example, the following configures a bar chart with various attributes:

{  
 "graphOrientation": "vertical",  
 "plotBox": True,  
 "showLegend": False,  
 "smpLabelRotate": 90,  
 "smpTitle": "Samples",  
 "theme": "CanvasXpress",  
 "title": "Bar Graph Title",  
 "xAxis": ["V1"],  
 "xAxisTitle": "Value",  
 "graphType": "Bar",  
}

Some configuration entries will be encountered with most charts, such as graphType. Other configuration entries will only be present if the author wants to explictly show other information, such as a title, or control the character of the visualization, such as colours. CanvasXpress provides a [comprehensive documentation page](https://canvasxpress.org/api/general.html) where the options are described for each kind of chart, and an [examples page](https://canvasxpress.org/examples/area-1.html) where many of these options can be considered in the context of specific visualizations.

## Rendering

CanvasXpress charts are rendered in an HTML using JavaScript, but creating and placing those instructions into viewing containers requires the use of CanvasXpress for Python rendering components. The concept is simple enough:

1. Identify the Web container being used to view the charts, such as a Jupyter Notebook or Plotly Dash application.
2. Import the relevant render component.
3. Configure a chart and pass the object to the renderer.

Let’s consider a Jupyter example:

from canvasxpress.canvas import CanvasXpress  
from canvasxpress.render.jupyter import CXNoteBook  
  
# Aggregate one or more charts for rendering in a notebook cell:  
notebook\_charts = CXNoteBook(  
 CanvasXpress(  
 render\_to="bar1",  
 data={  
 "y": {  
 "vars": ["V1"],  
 "smps": ["S1", "S2", "S3"],  
 "data": [[33, 44, 55]],  
 }  
 },  
 config={  
 "graphOrientation": "vertical",  
 "plotBox": True,  
 "showLegend": False,  
 "smpLabelRotate": 90,  
 "smpTitle": "Samples",  
 "theme": "CanvasXpress",  
 "title": "Bar Graph Title",  
 "xAxis": ["V1"],  
 "xAxisTitle": "Value",  
 "graphType": "Bar",  
 },  
 width=609,  
 height=609,  
 )   
)  
  
# Render the charts into the cell:  
notebook\_charts.render()

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

The import statement from canvasxpress.render.jupyter import CXNoteBook indicates that we want to target a Jupyter Notebook context, and the CXNoteBook object is what will create the output for our charts.

Depending on the greater logic, such as for a Flask application, the collection of charts being rendered might be used multiple times, so the renderer accepts one or more CanvasXpress objects in its constructor. After initialization, the render() function can be called to create the appropriate output.

Where appropriate, render() also accepts parameters to control the chart layout or take additional actions such as save the generated HTML into a local file. Each renderer includes documentation as appropriate, which can be accessed in a Python interpreter using Python’s help() function:

help("canvasxpress.render.jupyter.CXNoteBook.render")

Help on function render in canvasxpress.render.jupyter.CXNoteBook:  
  
canvasxpress.render.jupyter.CXNoteBook.render = render(self, \*\*kwargs: Any)  
 Renders the associated CanvasXpress object appropriate for display in  
 an IPython (e.g., Jupyter NoteBook/Lab) environment. Charts cannot  
 have the same name, so render\_to will be updated with a uuid for each  
 conflicting chart.  
 :param kwargs: `Any`  
 \* Supports `columns` for any positive `int` of `1` or greater, with a  
 default value of `1`. Values less that `1` are ignored. `columns`  
 indicates how many charts should be rendered horizontally in the  
 Jupyter Notebook if more than one chart is being tracked.  
 \* Supports `output\_file` as a string for a path at which the output  
 should be saved. If a file exists at the specified path then  
 it will be overwritten. This permits Jupyter sessions to render  
 output that is saved and accessible in later sessions.  
 \* Supports `debug` for displaying the output source. True indicates  
 that the HTML code shall be displayed prior to the parsed output.  
 Default is False.

The chart configuration works the same for each context, but the render required depends on the nature of the viewing environment. For example, Plotly Dash has different requirements than a terminal session that uses a local Web browser on a developer machine. Reusing the Jupyter example, here’s the same chart configuration but with a local Web browser render object that instead would launch a Web browser on the local system to display the chart:

from canvasxpress.canvas import CanvasXpress  
from canvasxpress.render.popup import CXBrowserPopup  
  
# Aggregate one or more charts for rendering in a local browser:  
local\_browser\_charts = CXBrowserPopup(  
 CanvasXpress(  
 render\_to="bar1",  
 data={  
 "y": {  
 "vars": ["V1"],  
 "smps": ["S1", "S2", "S3"],  
 "data": [[33, 44, 55]],  
 }  
 },  
 config={  
 "graphOrientation": "vertical",  
 "plotBox": True,  
 "showLegend": False,  
 "smpLabelRotate": 90,  
 "smpTitle": "Samples",  
 "theme": "CanvasXpress",  
 "title": "Bar Graph Title",  
 "xAxis": ["V1"],  
 "xAxisTitle": "Value",  
 "graphType": "Bar",  
 },  
 width=609,  
 height=609,  
 )   
)  
  
# Render the charts in a browser launched on the local system:  
local\_browser\_charts.render()

## JavaScript Identifiers

Given that a CanvasXpress chart will render using JavaScript in a Web context, identifiers can be assigned to each chart for reference by JavaScript functions. render\_to accepts a JavaScript compliant string value, and it can be used later in the Python code to ascertain the ID of a chart object. If no identifier is provided then the chart is considered anonymous and CanvasXpress for Python will assign a unique ID prior to rendering, and the render\_to property can be used to determine what ID has been assigned. Note that some contexts—such as Dash—use the JavaScript React library for component rendering, and in these contexts no chart ID can be reused. In such cases, all charts provided by CanvasXpress for Python to those contexts will be assigned a unique ID to prevent rendering issues within the HTML container.

## Sizing

Charts can be sized according to pixel dimensions using the width and height properties. If those are not specified then the default CanvasXpress chart size will be used, which at the time of writing is 500 x 500 pixels.

## Chart Metadata

CanvasXpress allows for metadata describing the samples or variables, which can be used to improve formatting or permit the user to explore the chart in greater detail through enhanced formatting, tooltips, data segregation, and more.

Metadata is provided as key-value pairs with list lengths matching the number of samples or variables being described:

|  | *vars Metadata 1* | Sample 1 | Sample 2 |
| --- | --- | --- | --- |
| *smps metadata 1* |  | *metadata 1 value* | *metadata 1 value* |
| *smps metadata 2* |  | *metadata 2 value* | *metadata 2 value* |
| variable 1 | *vars metadata 1 value* | value | value |
| variable 2 | *vars metadata 1 value* | value | value |

Revisiting our demographic example from earlier, this might translate into:

|  | *metric type* | Canada | The United States |
| --- | --- | --- | --- |
| *continent* |  | *North America* | *North America* |
| population | *demographic* | 38,250,000 | 329,500,000 |
| birth rate | *demographic* | 1.4 | 1.64 |

Given multiple continents, a default chart with all countries represented might be presented; however, the chart could be later filtered by the user to focus on a specific region of countries.

To include metadata in the data dictionary, use x for samples and z for variables. For example:

{  
 "y": {  
 "data": [  
 [38250000, 329500000],  
 [1.4, 1.64],  
 ],  
 "smps": ["Canada", "The United States"],  
 "vars": ["population", "birth rate"],  
 },  
 "x": {  
 "continents": ["North America", "North America"],  
 },  
 "z": {  
 "metric type": ["demographic", "demographic"],  
 }  
}

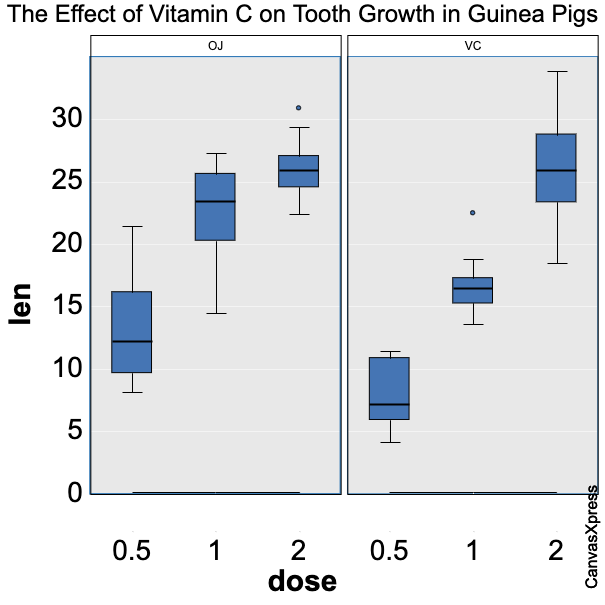
Here’s a box plot example that includes a number of sample metadata:

from canvasxpress.canvas import CanvasXpress  
 from canvasxpress.render.jupyter import CXNoteBook  
   
 cx = CanvasXpress(  
 render\_to = "metadata\_example\_chart",  
 data = {  
 "y": {  
 "smps": ["Var1","Var2","Var3","Var4","Var5","Var6","Var7","Var8","Var9","Var10","Var11","Var12","Var13","Var14","Var15","Var16","Var17","Var18","Var19","Var20","Var21","Var22","Var23","Var24","Var25","Var26","Var27","Var28","Var29","Var30","Var31","Var32","Var33","Var34","Var35","Var36","Var37","Var38","Var39","Var40","Var41","Var42","Var43","Var44","Var45","Var46","Var47","Var48","Var49","Var50","Var51","Var52","Var53","Var54","Var55","Var56","Var57","Var58","Var59","Var60"],  
 "data": [  
 [4.2,11.5,7.3,5.8,6.4,10,11.2,11.2,5.2,7,16.5,16.5,15.2,17.3,22.5,17.3,13.6,14.5,18.8,15.5,23.6,18.5,33.9,25.5,26.4,32.5,26.7,21.5,23.3,29.5,15.2,21.5,17.6,9.7,14.5,10,8.2,9.4,16.5,9.7,19.7,23.3,23.6,26.4,20,25.2,25.8,21.2,14.5,27.3,25.5,26.4,22.4,24.5,24.8,30.9,26.4,27.3,29.4,23]  
 ],  
 "vars": ["len"]  
 },  
 "x": {  
 "supp": ["VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","VC","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ","OJ"],  
 "order": [1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10],  
 "dose": [0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2]  
 }  
 },  
 config = {  
 "axisTickScaleFontFactor": 1.8,  
 "axisTitleFontStyle": "bold",  
 "axisTitleScaleFontFactor": 1.8,  
 "graphOrientation": "vertical",  
 "graphType": "Boxplot",  
 "groupingFactors": ["dose"],  
 "plotBox": True,  
 "showLegend": False,  
 "smpLabelRotate": 90,  
 "smpLabelScaleFontFactor": 1.8,  
 "smpTitle": "dose",  
 "smpTitleFontStyle": "bold",  
 "smpTitleScaleFontFactor": 1.8,  
 "summaryType": "iqr",  
 "theme": "CanvasXpress",  
 "title": "The Effect of Vitamin C on Tooth Growth in Guinea Pigs",  
 "xAxis": ["len"],  
 "xAxisTitle": "len"  
 },  
 width = 609,  
 height = 609  
 )  
   
 display = CXNoteBook(cx)  
 display.render()

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

As this chart has metadata, such as supplement and dose, the user can use the context-menu to make adjustments such as segregating by supplement and ordering by dose within each supplement perspective. It’s the same data as before, but additional clarity has been provided by user exploration.



## Chart Events

CanvasXpress supports JavaScript events, with a default event handler provided to display tooltip information on various aspects of a chart. For example, in a basic bar chart the tooltip will display the variable and sample labels plus intersecting value when the mouse however over a bar.

CanvasXpress for Python supports the creation of custom events using a CXEvent object:

from canvasxpress.js.function import CXEvent  
  
CXEvent(  
 id="mousemove",  
 script="t.showInfoSpan(e, '<pre>' + t.prettyJSON(o) + '</pre>');",  
)

id is the name of the JavaScript event that will trigger the logic. script is the content of the event function to be used by CanvasXpress.

CanvasXpress provides each event function with three parameters:

* o is the data provided to the CanvasXpress chart
* e is the JavaScript object for the triggering event
* t is the full JavaScript CanvasXpress object (e.g., including the configuration)

Events are added to CanvasXpress for Python objects using a list of events:

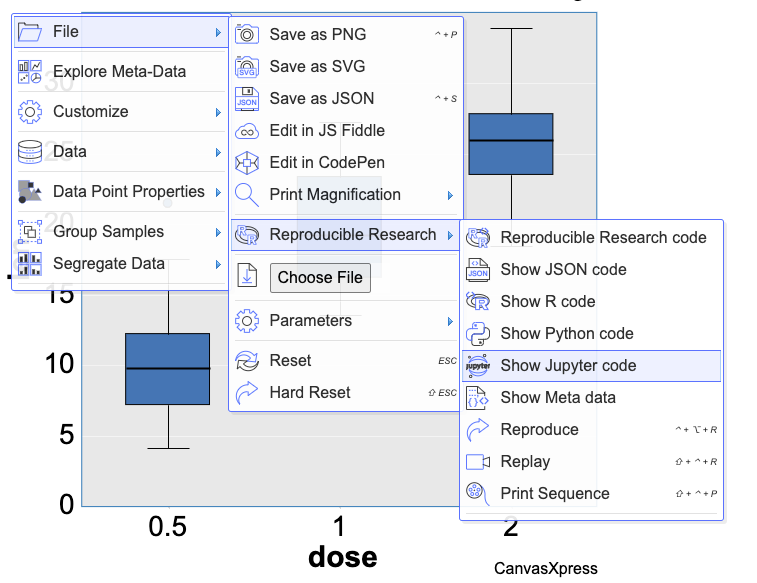
from canvasxpress.canvas import CanvasXpress  
from canvasxpress.render.jupyter import CXNoteBook  
from canvasxpress.js.function import CXEvent  
  
cx = CanvasXpress(  
 render\_to="bar2",  
 data={  
 "y": {  
 "vars": ["V1"],  
 "smps": ["S1", "S2", "S3"],  
 "data": [[33, 44, 55]],  
 }  
 },  
 config={  
 "graphOrientation": "vertical",  
 "plotBox": True,  
 "showLegend": False,  
 "smpLabelRotate": 90,  
 "smpTitle": "Samples",  
 "theme": "CanvasXpress",  
 "title": "Chart with Custom Event Showing X-Y Coordinates",  
 "xAxis": ["V1"],  
 "xAxisTitle": "Value",  
 "graphType": "Bar",  
 },  
 events=[  
 CXEvent(  
 id="mousemove",  
 script="t.showInfoSpan(e, '<pre>X: ' + e.clientX + '<br>Y: ' + e.clientY + '</pre>');",  
 ),  
 ],  
 width=500,  
 height=500,  
)  
  
display = CXNoteBook(cx)  
display.render()

<IPython.core.display.HTML object>

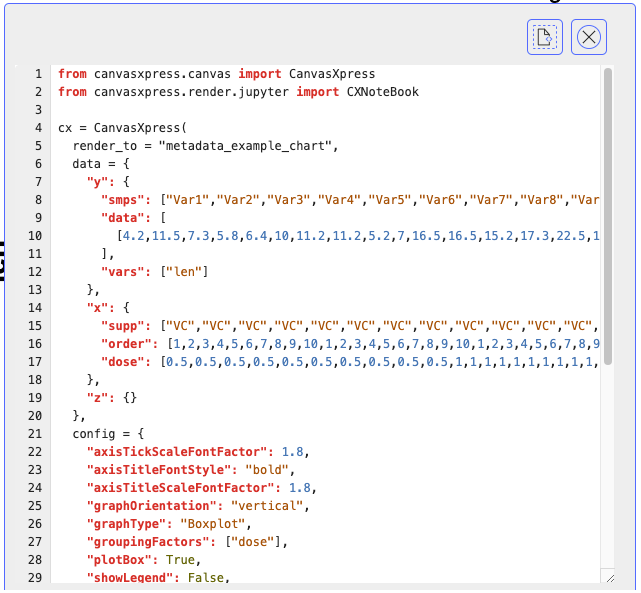
<IPython.core.display.HTML object>

## Obtaining Example Code from Any CanvasXpress Chart

CanvasXpress charts track their JSON data and configuration objects over their lifespan, and they can convert attributes such as those into code examples. On any CanvasXpress chart, such as the [examples at CanvasXpress.org](https://canvasxpress.org/examples/area-1.html), the right-click menu or configuration tool can be used to obtain code in one of the supported languages:



Selecting the menu option will bring up a code sample that captures the chart as currently configured. The best part about this feature is that a chart can be customized using menu or configuration tool selections and the code for the resultant chart can be captured, which is an excellent way to learn about the many options available for chart customization.



CanvasXpress currently supports Python code examples for a local development context with a pop-up browser and a Jupyter Notebook context. However, remember that using a chart configuration in another context, such as Streamlit, simply requires the use of the proper render object.

## Conclusion

CanvasXpress is a powerful and performant charting library with first class support for data exploration. For many scientific and business contexts, the ability to provide users with curated visualizations that can be further enhanced or explored by the user is a game changing capability that is sure to make solutions stand out from the crowd. We have enjoyed using CanvasXpress in our own client work, and we are proud to bring this excellent framework to the Python community for everyone to enjoy. This introduction provides a comprehensive overview of chart creation using Python, but be sure to visit the [CanvasXpress site](https://canvasxpress.org) to review the more than 50 types of charts and the numerous examples providing inspiration for each. Also be sure to check out our [PyPI page](https://pypi.org/project/canvasxpress/) for more information on how to adopt CanvasXpress into your own projects and experiments.