SOFTWARE TOOL ARTICLE

Plotly.NET: A fully featured charting library for .NET programming languages [version 1; peer review: awaiting peer review]

Kevin Schneider, Benedikt Venn, Timo Mühlhaus

Computational Systems Biology, TU Kaiserslautern, 67663 Kaiserslautern, Germany

Abstract
Data visualization is integral for exploratory data analysis and data presentation. Consequently, every programming environment suitable for data analysis needs a feature-rich visualization library. In this paper, we present Plotly.NET - a fully featured charting library for the .NET ecosystem. Plotly.NET requires little ceremony to use, provides multiple API (application programming interface) layers for both low-code and maximum customizability, and can be used in many different visualization workflows - from interactive scripting and notebooks to user interface applications. Plotly.NET brings the charting capabilities of plotly.js to the .NET ecosystem, successfully combining the flexibility of the plotly grammar with the type-safety and expressiveness of .NET.

Keywords
.NET, F#, C#, data visualization

Corresponding author: Timo Mühlhaus (muehlhaus@bio.uni-kl.de)

Author roles: Schneider K: Conceptualization, Methodology, Software, Validation, Visualization, Writing – Original Draft Preparation; Venn B: Resources, Software, Visualization, Writing – Original Draft Preparation; Mühlhaus T: Conceptualization, Funding Acquisition, Methodology, Project Administration, Software, Supervision, Writing – Original Draft Preparation

Competing interests: No competing interests were disclosed.

Grant information: This work was supported by the Deutsche Forschungsgemeinschaft (SFB/TRR175, project INF) and the National Research Data Infrastructure Germany (NFDI) DataPLANT, awarded to Timo Mühlhaus. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Copyright: © 2022 Schneider K et al. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

How to cite this article: Schneider K, Venn B and Mühlhaus T. Plotly.NET: A fully featured charting library for .NET programming languages [version 1; peer review: awaiting peer review] F1000Research 2022, 11:1094 https://doi.org/10.12688/f1000research.123971.1

First published: 23 Sep 2022, 11:1094 https://doi.org/10.12688/f1000research.123971.1
Introduction
Proper visualization of scientific results or data in general is an essential element for teaching, the discussion of analysis results, or the distribution of new insights within the research community. Well structured figures help to improve visibility, explorability, and interpretability of the underlying data, consequently saving tedious searching in order to be able to interpret the presented data. Algorithmic diversity is paramount in exploratory data analysis, but ultimately the goal will always be to produce meaningful figures that represent the obtained information in an optimal way. Even during exploration, figures help to guide the researcher to hidden properties that become apparent when inspected in the form of a figure, especially when the figure is interactive. Therefore, the ability for convenient and streamlined plotting strategies/possibilities is not only important for distributing results, but also helps to get new insights during data analysis, and is an invaluable tool in the data science stacks of any programming language.

plotly.js is a popular open source graphing JavaScript library. It provides a well defined grammar for creating visualizations via the figure data structure - a tree with named nodes called attributes. The figure data structure can be represented via JavaScript Object Notation (JSON) - a widespread standard for representation of structured data which can be parsed and created by any modern programming language (see Figure 1 for an example). Due to this interoperability, libraries for creating plotly.js figures are available in many programming languages, most notably in Python and R which see widespread adoption in the scientific programming landscape.

In this paper we present Plotly.NET - a library for creating plotly.js-conform JSON formatted figures for the set of statically typed .NET languages (F#, C#, VB.NET). Plotly.NET is written in F# and leverages the static type system by providing type-safe domain specific languages (DSLs) for plotly.js attributes and objects, as well as several abstraction layers for ease-of-use. Plotly.NET is feature complete regarding plotly.js API (application programming interface) coverage and is compatible with the .NET kernel for Jupyter, therefore offering a complete suite of data visualization tools for exploratory data analysis unprecedented in the .NET environment.

Methods
Implementation
Plotly.NET is implemented in the open source F# programming language. It creates objects which are serialized into JSON (using the Newtonsoft.Json package) which can be used by a rendering engine to create the visual equivalent. This is either realized by embedding the generated JSON in a HTML scaffold where plotly.js is used for rendering the plot in a browser (or equivalent environment where JavaScript can be executed), or via an interface for custom rendering engines.

The API is designed to enable different styles of creating charts and contains several abstraction layers. All (as of v2.6.3) plotly.js trace types have type safe equivalents with additional convenience layers to create derivative charts from these base traces.

Core API layers
Plotly.NET provides multiple API layers with increasing abstraction levels: (1) DynamicObject, (2) plotly.js object mappings, and (3) the high-level Chart API.

(1) DynamicObject: The low-level API builds on top of the DynamicObj package which provides a way of setting dynamic members in the statically typed .NET environment. Instances of this class, as well as instances of classes inheriting from it, provide the ?-operator which enables member assignment on runtime. This is usually prevented by the static type system of .NET languages, but necessary to support the highly dynamically typed nature of the JSON objects expected by plotly.js. Static classes are not flexible enough for this, especially because almost all attributes are optional and can be of multiple types.

This API layer is closest to plotly.js and can be used to directly translate javascript examples, as attribute names must be mirrored exactly. This also means that extensive knowledge of plotly.js, or at least its JSON schema is necessary to effectively use it, but makes it easy to implement new features should the underlying plotly.js version be updated. Figure 2 shows an example on how to use this API layer to generate the same plot shown in Figure 1.

(2) plotly.js object mappings: plotly.js attributes that are themselves objects are abstracted as classes built on top of (1) and provide static init and style methods, as well as object-specific methods. These methods have a plethora of optional typed arguments which are equivalent to all attributes supported in the respective plotly.js JSON. Additionally, type-safe DSLs for attributes/values used on multiple occasions such as style parameters, color, or enums are provided in the StyleParam namespace. This layer comes with the additional advantage of providing options for auto-completing argument names as well as documentation for them, leading to less plotly.js knowledge being required for its usage. Figure 3 shows an example on how to use this API layer to generate the same plot shown in Figure 1.
Note that this API layer provides type-safe abstractions, but still needs insights about how plotly.js structures figures: While allowed attribute names and types are abstracted, the user still needs to know that for example axis styles are to be set on a Layout object (see equivalent lines 10-13 in Figure 1). Lines 25-27 create the figure object and render it in a browser. Right: The respective rendered graph.

(3) The high-level Chart API provides type safe abstractions for chart creation and styling without knowledge of the structure of plotly.js JSON. Type-safe methods for creating figure structures for all trace types and derived charts as well as various styling methods are provided.
The plotly.js figure schema is completely abstracted away. The API surface is only concerned with ‘creating and styling a chart’. Consider the following example: In plotly.js JSON, the axis style has to be set on the layout object. It also must be specified which type of axis (x, y, z, etc.) it is. Styling the x axis usually requires knowledge of this. The Chart API however provides a Chart.withXAxisStyle function, which also provides arguments for the relevant attributes, and can be applied to any Chart object.

Using this API layer also significantly decreases the lines of code necessary to create figures, as proper creation of nested objects and argument naming is implicitly done by the Chart methods. See Figure 4 for an example on how to use this API layer to generate the same plot shown in Figure 1. Note that there is a single flow of the initial Chart object, without the need of creating additional objects for axis styling.

Using this API layer also significantly decreases the lines of code necessary to create figures, as proper creation of nested objects and argument naming is implicitly done by the Chart methods. See Figure 4 for an example on how to use this API layer to generate the same plot shown in Figure 1. Note that there is a single flow of the initial Chart object, without the need of creating additional objects for axis styling.

The plotly.js figure schema is completely abstracted away. The API surface is only concerned with ‘creating and styling a chart’. Consider the following example: In plotly.js JSON, the axis style has to be set on the layout object. It also must be specified which type of axis (x, y, z, etc.) it is. Styling the x axis usually requires knowledge of this. The Chart API however provides a Chart.withXAxisStyle function, which also provides arguments for the relevant attributes, and can be applied to any Chart object.

Using this API layer also significantly decreases the lines of code necessary to create figures, as proper creation of nested objects and argument naming is implicitly done by the Chart methods. See Figure 4 for an example on how to use this API layer to generate the same plot shown in Figure 1. Note that there is a single flow of the initial Chart object, without the need of creating additional objects for axis styling.

Plotly.NET however does not enforce the usage of any API layer. The low-level, abstraction-less layer is exposed on purpose as this gradual abstraction via API layers enables users with any kind of knowledge about plotly.js to use it. Additionally, this allows for both imperative and declarative creation of figures. The dynamic assignment of members however might seem not intuitive to .NET users unexposed to dynamically typed languages. The high-level Chart API on the other hand provides type-safe abstractions with the trade-off of reduced customizability. Extending this layer needs a lot of domain knowledge in both plotly.js and the Plotly.NET equivalent DSL. We believe that users are offered the best of both worlds by providing low-code abstractions for common use-cases, while retaining the ability to take full control where needed.

**Extensions**

In addition to the core API, three extension packages - Plotly.NET.ImageExport, Plotly.NET.Interactive, and Plotly.NET.CSharp - are provided. Plotly.NET.ImageExport and Plotly.NET.Interactive cover two specific use cases - static image export and usage with the .NET Jupyter Kernel .NET Interactive - while the aim of Plotly.NET.CSharp is providing a native, idiomatic API surface for consuming the core API for the object-oriented C# language (note that the core API can be consumed in any .NET programming language, this package just provides a more idiomatic API).

Plotly.NET.ImageExport provides an interface for rendering engines that can produce PNG, JPG, or SVG files from plotly.js JSON figures. A reference implementation is provided that uses PuppeteerSharp to create a headless Chromium browser to render the figure and save it to the disk in the desired format.

Plotly.NET.Interactive is a formatting extension for .NET interactive, the Jupyter kernel for .NET languages. It enables automatic rendering of figures created with Plotly.NET in the notebook output cells without the need to leave the analysis environment.
environment by switching to the browser. Having the charts tied to the code producing them is especially helpful in the context of sharing and comprehension of visualizations. Examples of such notebooks are included in the Plotly.NET repository (see the Data availability section).

**Operation**

Plotly.NET targets the netstandard2.0 framework and is therefore supported on a plethora of compatible target frameworks across the .NET ecosystem. At the time of writing this article, this means that Plotly.NET can be used on any modern desktop operating system, as well as multiple mobile operating systems. It can be either referenced as nuget package or be built from source.

**Using Plotly.NET nuget packages**

Plotly.NET is available as pre-compiled binary package on nuget.org (see here), which is the central package registry in the .NET ecosystem. It can be incorporated into any .NET project, script or notebook via the respective package reference. A target framework compatibility matrix is available on the nuget package page.

**Building Plotly.NET from source**

The source code as well as detailed instructions on how to build it are available on GitHub (see Software availability). An installation of the .NET SDK of version ≥ 6.0.1 (freely available to readers and reviewers here) is mandatory.

**Use cases**

Plotly.NET aims to be a general-purpose visualization library for the .NET ecosystem. It targets interactive workflows such as visualization in scripting or in notebook environments, as well as visualizations in any kind of application, be it for desktop, mobile, or the web. Therefore, we demonstrate some of the visualization capabilities in the following section instead of specifying use cases.

Figure 5 shows a few more complex charts that can be generated with Plotly.NET:

- **Figure 5A** shows various variations of box and violin plots for the purpose of visualizing shapes and properties of distributions.

- **Figure 5B** is a combination of a temperature time course as line chart and a superimposed heatmap indicating correlation coefficients derived by continuous wavelet transform using FSharp.Stats (see Ref. 11 and Underlying data). High values (reddish) indicate high correlation to the frequency given as row indicators on the right. High negative values (bluish) indicate high anti-correlation.
Figure 5. Example charts. (A) Box and violin plots to describe properties and shapes of distributions. (B) Combined chart of temperature measurements and their frequency representation as heatmap. (C) Splom chart of the iris data set. (D) Surface chart of a three dimensional wave function.

- Figure 5C shows a scatter plot matrix of the classic iris data set (see Ref. 12 and Underlying data) providing an overview of how the respective features in it correlate with each other.

- Figure 5D shows a surface chart of the three dimensional wave function \( f(x,y) = \sin \left( \sqrt{x^2 + y^2} \right) \).

The respective code is provided as interactive notebooks in the repository on GitHub (see Software availability).  

**Data availability**

**Underlying data**

The data underlying Figure 5 are contained in the ‘publication’ folder in the Plotly .NET GitHub and Zenodo repositories (see Software availability below) as follows:

- iris-data.csv (contains species label, sepal length/width, and petal length/width of observed iris flowers. This data was originally obtained from Ref. 12).

- waveletData.txt (contains water temperature measurements conducted by the authors. Additional correlation coefficients were derived by continuous wavelet transform using Marr wavelets with frequencies specified in the row annotations. This data was made available previously in Ref. 11).
Software availability

- Software available from: https://nuget.org/packages/Plotly.NET
- Source code and notebooks generating the example figures are available from: https://github.com/plotly/Plotly.NET
- Archived source code at time of publication: https://doi.org/10.5281/zenodo.6822582
- License: MIT

Acknowledgements

The authors would like to thank all independent open source contributors on the GitHub repository, no matter how small the contributions. Special thanks to @WhiteBlackGoose for the extensive conceptual discussions about all things .NET.

References

1. Plotly Technologies Inc: Collaborative data science. 2015. Reference Source
The benefits of publishing with F1000Research:

• Your article is published within days, with no editorial bias
• You can publish traditional articles, null/negative results, case reports, data notes and more
• The peer review process is transparent and collaborative
• Your article is indexed in PubMed after passing peer review
• Dedicated customer support at every stage

For pre-submission enquiries, contact research@f1000.com