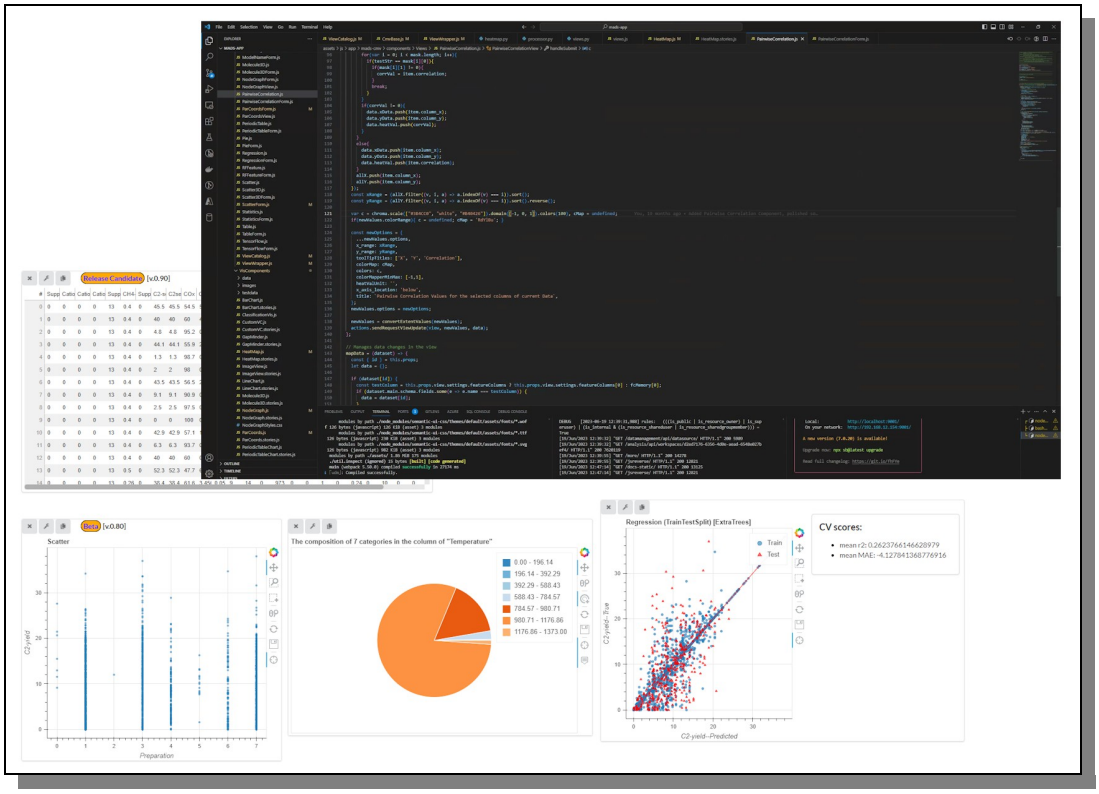
TAKAHASHI GROUP (情報化学研究室)

Hokkaido University



**CATALYST ACQUISITION BY DATA SCIENCE**

# CADS

Catalyst Acquisition by Data Science

# Developer Manual

ENGLISH

# Table of Contents

Chapter 1

# Introduction

*Get to know what CADS is and what it can do for you as a researcher.*

CADS is a catalyst analysis environment that implements a user-friendly graphic user interface(GUI) that aims to help scientist to analyze data in a more simple and effective way without the need of coding and/or setting up complex IT environments.

## *CADS Overview*

CADS provides three basic functions for users: “Data Management”, “Analysis”, and “Prediction”.

### Data Management

CADS provides the ability to upload and record individual data as well as data analysis procedures with the option for multi-user sessions, allowing users to share data and analysis reports with collaborators of their choosing. Meaning that not only can a researcher study and analyze their own data, but they can also compare and study other's shared data for a better and deeper understanding.

### Analysis

Data visualization tools (scatter plotting, histograms, etc.) and machine learning functions are available for catalyst data analysis and machine learning model selection. This is the core part of the CADS platform and one of its major strengths. In the workspace area a user can load a various buffet of components through which the user can then analyze and manipulate the data for make new findings and getting further understanding. The range of Components is steadily growing and the power of this section is increasing every month. Besides the classic visualization tools mentioned above, there are many more, ranging from image manipulation, machine learning and advanced analyzing and visualization tools, all with simple and straight forward user interfaces that guides the user to do complex analyzing with few and user-friendly operations.

This manual's primary target is to describe and explain how to develop new components for CADS.

### Prediction

Data prediction capabilities are available within the “Analysis” function and inside the Machine Learning components, allowing users to apply and store learning models for predicting new output (e.g. predicting new catalysts). Under the Prediction tab the user can then apply those pre-trained and saved models in order to predict the result with new data.

## *CADS Architecture*

On the surface to the average user the CADS system is just an ordinary website operated by basic mouse and keyboard operations. A few clicks and perhaps some simple typing of values and then submit to the CADS server, and voila, the data will be displayed in a new graphical and exciting way.

Under the hood the CADS system is built with Django and React JS and for easy maintenance and sound and familiar structure. The custom and generic Analyzing components are using state of the art technology from various high-end libraries for graphical display of data, e.g. Bokeh JS and Plotly JS. On the server side a wide range of support system is feeding the components with all sorts of data management capabilities like SciKit Learn and SciKit Image and much more. This is not needed to be known for the CADS user, but it might be good to know that under the hood are all the well-known complexity the research world is used to, but through CADS, most of the complexity is removed or toned down. Data analyzing should not just be for IT savvy people, it should be available to everyone.

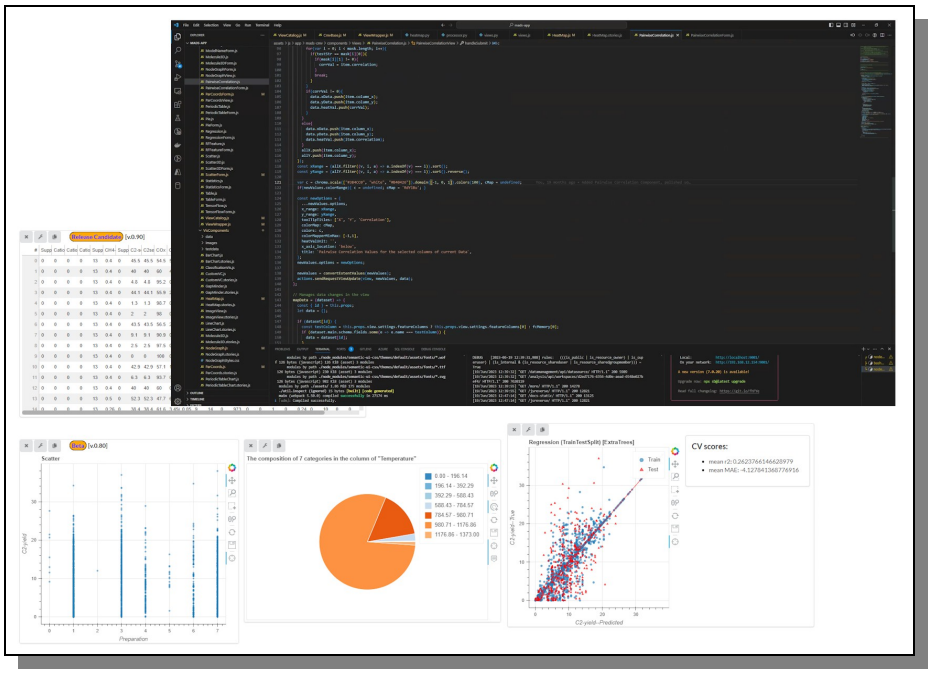
## *CADS Online in the Browser*

In order to use CADS to its fullest one need to have a user account in order to be able to save data and workspaces. Applying for an account is simple and quick and after confirming with a valid email address the CADS system is fully available to use. All interaction from hereon is based on familiar browser interactions using classic form fields and buttons and no need for any plugins or additional downloads are required. All work can be saved and if wanted to, relatively easily shared.

## *CADS Component Development*

For more details on CADS in general terms, please look for the CADS user Manual or component manual found on the CADS website. Here we will now dive deep into the world of CADS Component Developing and programming for those of you who run your own server and wish to add more features and components to enhance CADS even further.

Remember that if you ever create something worth sharing with the whole world via the official online version of CADS, just contact us at Takahashi Group and we can talk about it.



Chapter

2

# Component Developer's Manual

*So now you have your own local version of CADS running on your computer and you wish to implement your own Analyzing component, well let's do that then.*

In this chapter we will try to explain everything that you need to know about coding your own CADS analyzing Components as a programmer and developer. We will assume that you are at least average skilled in coding in both JavaScript and HTML as well as Python, and that you are familiar with the libraries and frameworks of Django and React. If not, we then first recommend you to go online and acquire the required skill-sets and knowledge before you continue reading.

Next, we will assume that you already have a local version of CADS running on your machine, but if you don't, yet, then please refer to the CADS Server Manual and make that happen before you continue reading.

Which programming tool you use is all up to you, but in this manual and in figures shown all coding and development is done with Visual Studio Code.

## *Starting The Development*

So at this point you should be all ready to go in all practical matters, but of course you should also have some kind of idea and vision on what kind of component you want to build, but with that we cannot help, that is all on you, but I am sure you got it covered.

### Basic Structure

We will focus only on the area and files of the CADS code project that makes up Visulization Components and ignore all the rest. In your code it is the assets/js/app/mads-cmv folder where we will find the JavaScript files we need to create and/or edit and also analysis/api/utils folder where we will add a python server side file. In Figure 1 below you can see which folders we are refering to.

Basically there are 3 files that need to be created using JavaScript and React in order to create a new component. An additional 4$^{th}$ file may be created for testing, but it is optional. In the same are there is also one file that need to be edited in order to make the new component show up properly.

The python file in the analysis utils area is optional, and only needed if the component need to ask the server for something such as doing some calculations using python libraries. But if such a file is added and used, than an addtional file need to be edited for the server to recognize the call.

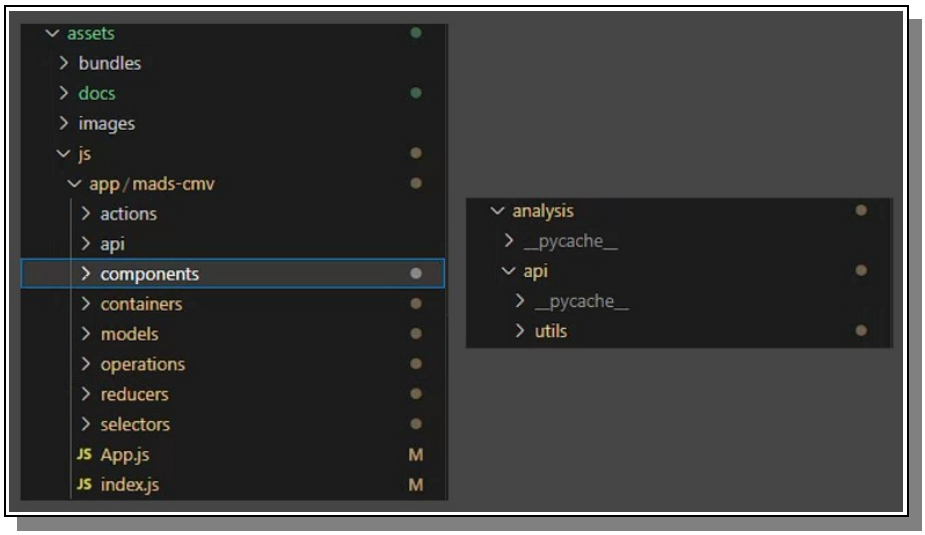



*Figure 1: The main focus folders for creating CADS components.*

So all in all a minimum of three new files and one edit file is needed for a component to appear, and a maximum of 5 new files and 2 edits for slightly more advanced ones.Most of our client side code will resite in the 'Views' and the 'VisComponents' folders under the 'js' section as seen in Figure 2 below.

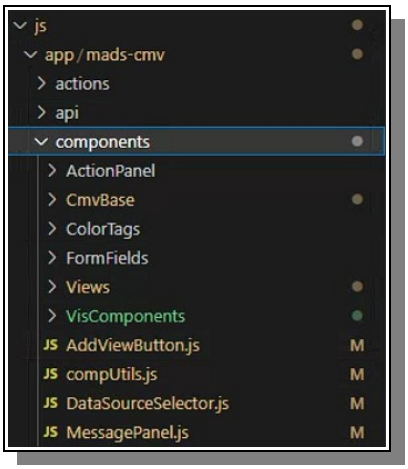



*Figure 2: The JavaScript based folder for hosting a component.*

## Creating the necessary files

When you open the 'Views' folder you find a whole bunch of component related files, 2 for each component in a matter of fact, like e.g. Bar.js and BarForm.js or HeatMap.js and HeatMapForm.js

The first file contains the code that the component need to manage and apply the data and the configurations to the components and call for server side calculations and finally call for the drawing of the component. This is the most important file and you need to create one for your own component. When you do give it a good name that make sense and give it a .js extension.

The second file you need to create draws, displayes and manages the configuration form for the component and is therefore named the same as the previous file but with the ending of 'Form' to make it clear what it contains. Also this file has .js as extension.

At the bottom if this folder there is a file called ViewCatalog.js which you need to edit for a component to appear, so open that file and we will get to it shortly.

Next we enter the 'VisComponent' folder and there you will se yet again a bunch of files related to

existing components. They are often 2 per component but sometimes only one. E.g. BarChart.js and BarChart.stories.js or HeatMap.js and Heatmap.stories.js

Also here you need to create at least one new file for your component. Give it a similar name as before so the connection is obvious and give it a .js extension.

If you are interesting in having a slightly independant test version of your component while you develop, we recommend that you also create a file called the same but with the ending of .stories (as seen above) and the extension of .js.
Further below in another section we will explain how to use it in a little more detail but you can already now read up on what storybooks are and how you basically use it at https://github.com/storybookjs/storybook.

If you know already now that you will be using some python library to drive the data manipulation of the component then you can create that file too. Lets open the analysis/api/utils folder mentione before and create a new file that you name either similar to the component or if you prefer something descriptive of what the file will intend to do. By looking at the names of the files already in that folder you probably get an idea of how to name your file too. Be aware that the extension of theis file should be .py since it is a server side Python file. You also need to edit the file in the same folder called processor.py so open that as well and we get back to it. If you are not yet sure if you need server side support for your component, just skip it for now and come back to it later if you change your mind.

## Understanding the Code

At this point we recommend that you visit and read and try to basically understand the code of some of the alrerady existing components e.g. The BarChart and the Scatter3D just to mention a few. If you are creating a component that is in some way similar to another already existing component, then you should definitely add that component to your list of studies. The more components you take a closer look at, the better understanding you get on what the code do in each file and the are similar or different, and the creating of your own component will be less difficult.

## Preparing the Files

When you know which already existing component is most similar to the component you wish to create you will use that as a template. If your component is very different, please still make a pick of a component to use as template, because that will speed things up and be less error-prone that if you write all by yourself from scratch. One criteria that can be important when picking a template is what library you are using when drawing the visualization. Will it be a Bokeh component or a Plotly component or will it be something else. (Some more information about the drawing libraries are found further on).

When the decision is made you copy the code from that existing component of each file for that component and paste it inside your, currently empty, files. Make sure the correct file get the correct code, form code into the form file, vis code into the vis file and python server code into the python server file.

Next, before we do anyhing else, we need to change the names of the main classes so that our new component is really new and do not use any of the copied templates. Do not bother to delete anything you think you do not need yet, that will be done at the very end when you are completely sure about that.

### *The Component Views File*

First change the importing of the other parts and sections of your component, the Vis-file and the Form file to be pointed at correctly.

For example (depending on which template you decided upon) change the following

```
import Bar from '../VisComponents/BarChart';
import BarForm from './BarForm';
```

to (for example depending on your component and file names)

```
import YourComponent from '../VisComponents/YourComponent';
import YourComponentForm from './YourComponentForm';
```

Next change the View Class name so that it correlates with your component and the other newly assigned names you are using.

For example change the following

```
export default class BarView extends withCommandInterface(Bar, BarForm) {
```

to

```
export default class YourComponentView extends
withCommandInterface(YourComponent, YourComponentForm) {
```

If you are planning to call the pyhon server after a form is submitted in order to manipulate the data some how using a pyhon librar like e.g. SciKitLearn, then make sure that the last line inside the 'handleSubmit' method is looking like below.

```
actions.sendRequestViewUpdate(view, newValues, data);
```

That makes sure the serer will get a say before the Visualization drawing is being applied.

If you are not intending to use any python library and do not need to access the server before drawing, then the following line will suffice instead.

```
updateView(id, newValues);
```

Some existing components calls the server even though it does not currently do something except send back the data unaltered, that is for a range of reasons, one being that possible future addons are considered. That is of course an option for your component too.

You could also do a search in the file for every string that uses the template component name and replace it with your component name, e.g. BarChartParameter, shoudl be myComponentParameter.

If that is all done correctly we are temporarily done with that file for now.

### *The Component Form File*

First change the form module method name. And make sure that you are using the name you have been using for this part of the component previously so that all is pointing where it should.

For example (depending on which template you decided upon) change the following

```
const BarForm = (props) => {
```

to (for example depending on your component and file names)

```
const YourComponentForm = (props) => {
```

You also need to edit at at the very end of the file where it would look more or less something like this.

```
export default reduxForm({
```

```
  form: 'Bar',
  validate,
})(BarForm);
```

to

```
export default reduxForm({
  form: 'YourComponent',
  validate,
})(YourComponentForm);
```

You could also do a search in the file for every string that uses the template component name and replace it with your component name as mentioned earlier.

And when all that is done correctly we are temporarily done with that file too, for now.

### *The Component Visualization File*

First we need to change the component class name so it matches what we have used before.

For example (depending on which template you decided upon) change the following

```
export default class HeatMap extends Component {
```

to (for example depending on your component and file names)

```
export default class YourComponent extends Component {
```

At the end of the file you will need to change two lines for two objects used

```
HeatMap.propTypes = {
```

to

```
YourComponent.propTypes = {
```

and

```
HeatMap.defaultProps = {
```

to

```
YourComponent.defaultProps = {
```

You could also do a search in the file for every string that uses the template component name and replace it with your component name as mentioned earlier.

And when all that is done correctly we are temporarily done with that file too, for now.

### *The Component Storybook File*

The use of this file is all optional, but if you decide to use Storybook as a developmet support and tool, then do the same thing for that file.

First change the import to point at your Vis component.

For example (depending on which template you decided upon) change the following

```
import HeatMap from './HeatMap';
```

to (for example depending on your component and file names)

```
import YourComponent from './YourComponent';
```

Then at more or less the bottom of the file you need to change the following line.

```
const stories = storiesOf('HeatMap', module);
```

to

```
const stories = storiesOf('YourComponent', module);
```

Below that line are the tags call for each version of the component that the story book is capable of displaying. Those tags should be changed from e.g.

```
<HeatMap />
```

to

```
< YourComponent />
```

You could also do a search in the file for every string that uses the template component name and replace it with your component name as mentioned earlier.

And when all that is done correctly we are temporarily done with that file too, for now.

### *The ViewCatalog.js file*

This file is the file that manages and in some way control all existing components, and for your component to be seen inside the application you most tell this file about it.

Be careful that you only edit where and what you are supposed to, and that the block of code you add is properly closed and contain the same way all other components are inserted.

This is also a good time to decide which category your component best fit and place it accordingly inside this file.

Categories are not hard coded and 'real', it is just a way of grouping components better and helping the user to find useful components easier. So technically you could, if you really feel the need to, add your own Categories and they would show up in the application automatically. We recommend you avoid that unless it makes fully sense to do so.

First edit and add a new import to point at your component.

For example add the following

```
import YourComponentView from './YourComponent';
```

Pay attention to that is the View file you are pointing to, and not the Vis file.

Next if you scroll down, you see that each and every component has their own little JSON like object part like seen in the example below for the Table component.

```
// Table - A Customizable Data Table
  //---------------------------------------
  {
    type: 'table',
    name: 'Table',
    category: 'Visualization',
    version: 1.0,
```

```
    devStage: "Stable Release",
    component: TableView,
    settings: {
      columns: [],
      options: {
        extent: {
          width: 800,
          height: 400,
        },
      },
    },
  },
  //----------------------------------------
```

You need to add your component in a similar way like seen below.

```
// YourComponent - A New Great Component
  //----------------------------------------
  {
    type: 'your-component',
    name: 'Your Component Name',
    category: 'Visualization',
    version: 0.4,
    devStage: "Draft",
    component: YourComponentView,
    settings: {
      options: {
        extent: {
          width: 800,
          height: 400,
        },
      },
    },
  },
  //----------------------------------------
```

the 'type' is a name that the python server will recognize so give it a typename that fits and that you can use later if you add server side features. The 'name' is what ever you want your component to be called when seen inside the website. 'category' is what you think fits best, and you should sort the json to appear accordingly in the file as well for easy finding. Make sure you type correctly. The 'version' is any number you think fit the current version of your new component. Anything below. 1.0 will be flagged accordingly in the application for easy spotting. 'devStage' is just a string to describe the state developing of the component. ”Stable Release” should be used for any component that has a version higher or equal to 1.0, below that ”Beta” or ”Draft” i usually used.

'settings' and 'options' is for parameters that you wish to preset for a component when it is loaded for the first time. You might come back to this part of the file later when you have added more parametrs in you component form. There is no need to preset every available parameter, only those you want to. Basically all components have a 'width' and 'height' value under the 'extent' tag and you should probably do too. It is the initial size of the component after being loaded for the first time.

And when all that is done correctly we are temporarily done with that file too, for now.

### *The Component Server Side Python File*

If you are using a server side method containing some python code in order to manipulate the data

before it is returned to the Visualization drawing then you need to edit that too with correct names.

It is basically only one line that needs to change for now and that is the name of the method.

For example (depending on which template you decided upon) change the following

```
def get_bar(data):
```

to (for example depending on your component and file names)

```
def get_your_component_something(data):
```

You could also do a search in the file for every string that uses the template component name and replace it with your component name, e.g. BarChartParameter, shoudl be myComponentParameter.

You should leave the rest as is for now, before you test that it loads as planned.

### *The processor.py file*

In order for the system and more precisely the server to find the correct method for your component, you also need to edit this file properly.

First you need to import the file method you named above. At the beginning of this file you find the import calls.

For example add the following

```
from .your_component import get_your_component_something
```

Next you need to connect the type name you gave in the ViewCatalog.js file earlier with the python method you wish to associate with it. Just close below from the import lines there is an object called 'processor_map = {'. There you add your component, with the key that is called the same as the type name you used before, and the value that is the same as the python method name you used.

For example something like this

```
'your-component': get_your_component_something,
```

And when all that is done correctly we are basically done with this file.

## First Test Run

This is the time you should compile and run the code for the first time to make sure that all is working so far as planned. (How to run the code is of course part of the CADS Server Manual, so if you need an update on how to do that, look there.)

If all was edited perfectly fine then your component should appear in the website list and when you load it it should be loaded and basically look and behave like the template you used.

If you get any errors, most likely that is related to names, and it probably says so in the development console or the terminal depending on if it is a front end or back end issue. It most likely tells that it does not find something called something, and that should be enough information you need to track down the error and fix it.

After all names and pointers are correct and the draft component loads as it should you can move on and edit in your own code.

## *Hardcore Development*

Yes, it is hardcore, because we are not holding your hand through this process, it is all up to you, and what you need to learn and understand, in regard to do this coding, is more or less all up to you. If you use BokehJS, you need to understand BokehJS, if you use PlotlyJS you need to understand PlotlyJS. All information you need is out there online for that. Of course the same goes if you decide to add a new library on your own.

What we will do in this section is give a couple of generic pointers and tips for each file you need to edit, but anything beyond that is probably most likely something you need to discover on your own.

In regards of testing and debugging the code while you develop, it is often helped by using many console.log() messages spread out in the code to help you identify issues and better understand the flow in the client side part and logger.info() on the server side (Which is written in the terminal, not the console)

Which file you start with is of course all up to you, but we often begin with the configuration form file, so we do that here.

### Component Configuration Form File (e.g. 'YourComponentForm.js')

As can be seen at the import section at the top, you need to be familiar with React, Redux-Form and Semantic-UI-React to fully be able to code this file, so make sure you are.

For examples of how things are done, any of the other component form files are valuable to study, but one of the more complex ones is the 'ImageViewForm.js' with many solutions and examples that can be of help.

Besides that most thing with this file is pretty basic and straight forward when you grasp how it works.

What you need to change depends on what parameters you wish to expose to the user, but basically adding and changing form fields the values that should be contained within.

After all your stuff is added and the unused template stuff is cleaned away, you should test run the code and see that it works so far in the process as you expect.

### The Component View File (e.g. YourComponent.js)

For editing this file it is most likely that you need to be able to use pandas-js library as can be seen at the import section at the top of many of the already existing component view files, so make sure you are.

This file basically only have two major methods that need to be managed, first is the 'handleSubmit' that will get the form data after submit is clicked and where you may prepare those parameters in some way if needed before you send the away to the next in line. Second is the 'mapData' which is the method where the data itself may be restructured or at least checked before it is handed over to the visualization and rendering file. If the data was managed and altered in the server side that will be seen here. Pay attention to that 'data' is the data your component is manipulating and restructuring, but the 'dataset' is the actual original data used.

As before, any of the other component View files are valuable to study, but 'bar.js' is covering a lot to help you understand what is needed to be done.

### The Server Side Python Method File (e.g. your_component.py)

This file is of course optional as mentioned earlier, but if you use it here are some pointers.

To edit this file you need to be able to code in Python of course, but depending on what libraries you are

using, e.g. 'SciKitLearn' you need to know those as well.

The method within basically get a JSON data package as sent from the view file at the end of 'handleSubmit' function that contains the data and the form configuration inputs. You job is to unpack the parts you need, manipulate the way you need and then return the result.

As being repeated, any of the other component Python files are valuable to study, but 'regression.py' is covering a lot to help you understand what is needed to be done.

Remember that any print output or logger.info() will be written into the terminal (and not the console since this happening on the server side). Also add the flush parameter to any print command since otherwise the output will not happen in real time.

## The Component Visualization File (e.g. YourComponent.js (visComponents folder))

Besides React, depending on your needs, you might need to be knowledgeable in a few other libraries such as lodash, deepEqual, Jquery and more, but we leave that to you to decide. You also need to be understanding of the rendering library you are using, which mainly so far in existing components are either plain HTML (sometimes), or BokehJS (mostly) or PlotlyJS (occasionally).

Worth noting though is that documentations for the JS versions of the above mentioned chart drawing libraries is really undeveloped, and most information you will find is in relation to the python version, but as soon as you grasp the differences in syntax you will know how to alter the code to be working with JavaScript.

If you are familiar with the python version of any of those, remember that the JavaScript version is slightly different in how they work and you should be aware of those differences in mainly syntax.

As stated over and over again, and it cannot be stressed enough, you should study all other visualization files in order to understand the flow and process of things, their similarities and differences. All are worth looking into, but basically the one using the rendering library you intend to use is of course most important.

Also be aware that some components use the old, original syntax for React and others the newer one using hooks. We recommend that you use hooks if you can, but either should work fine.

Study each method carefully in order to know what to edit, add and remove based on your needs. Your main target methods is of course 'createEmptyChart' and 'CreateChart'. Also remember that the 'propTypes' and 'defaultProps' objects at the end of the file needs to correlate to your component, so add, edit and remove what you need there.

## The ViewCatalog.js File

At the end you should also return to this file and make sure that your component's initial state is preset as you wish by adding the needed parameters to the JSON structure. Only those parameters that need presets need to be added here, not every existing parameter.

## The Storybook File (e.g. YourComponent.stories.js)

This is ones again completely optional, but while you develop the visualization part of your component, a storybook version is really helpful for debugging and testing, since it does not involve any data and form related code. Only the visualization code is applied, and in the storybook file you control what sample data you want it to react to. So if you want to use that, you should of course read up on how 'Storybook' works.

Once again, study the other .stories.js files to understand what is happening, how and why. 'BarChart.stories.js' is probably one of files that cover most. The base concept is that you create a chunk

of fake sample data and then send that to the visualization part of the component and that will be drawn inside the Storybook web page, and you make as many versions you wish to test as many aspects of the component you like.

## *Final Words*

Basically all the knowledge and know-how you need to code your own CADS component are hidden inside the already existing components available inside the CADS system. So by just studying those files thoroughly in combination with learning the various libraries that in play you should be able to create your own component, if not with ease, at least without any major hiccups. So whenever feeling a bit lost on what to do, return to the source and see what has been done before. Since the code is local and running on your machine, you can of course play around freely with all components to see how the behave to specific changes and by that learning to do what you need to do.

If you ever get really stuck, then you can of course mail the CADS support team with questions or maybe even better post an issue query in GitHub and someone will get back to you to help.

## *Summary*

We wish that this manual have given you the support and foundation you need in order to develop your own CADS Analysis Components for your own personal benefit as well as for the community of CADS users at large. Feel free to return here whenever your memory fails you on what to do and when.

Also, please remember to return to our other detailed manuals on how to use CADS in general terms, on how to use each individual Analysis components currently available, and/or on how to setup and maintain your own CADS server.

Also, if you think you have created something worth sharing with the whole world via the official online version of CADS, just contact us at Takahashi Group and we can talk about it.

Congratulations on your newly gained status as a ‘CADS-Component-Developer’. Now it is time to do some serious development. Enjoy!

# Index

# Figure Index

# *Thanks for your support*

CADS

Catalyst Acquisition by Data Science