DATA ANALYSIS WITH RUST NOTEBOOKS

DR. SHAHIN ROSTAMI

Biography

Dr. Shahin Rostami is a Senior Academic (Associate Professor) and Principal Consultant in Data Science, Artificial Intelligence, Digital Health, and Defence Systems. He is currently working with industry whilst on his sabbatical from his position at the Department of Computing and Informatics at the Bournemouth University, where he has been a faculty member since 2014.

Dr. Rostami holds a Ph.D. in the field of Computational Intelligence with applications to Concealed Weapon Detection. His research interests lie within Data Science and Artificial Intelligence, ranging from theory to their application to Digital Healthcare and Threat Detection. Currently, he is consulting for and supervising PhD research projects in Non-Contact Vital Sign Measurement and Multi-Objective Concealed Weapon Detection. He has published in many high-impact journals and conferences, and organised/chaired special sessions including the IEEE CIBCB 2017 "Machine Learning in Medical Diagnosis and Prognosis". He also leads the Computational Intelligence Research Initiative (CIRI), and currently supervises 5 Ph.D. and 3 Ms.c. students in related subjects.

Dr. Rostami has founded and held the position of Programme Leader for multiple programmes at the postgraduate level: MS.c. Data Science and Artificial Intelligence (DSAI); MS.c. Digital Health and Artificial Intelligence (DHAI); and MS.c. Applied Data Analytics (ADA). He has designed and taught both postgraduate and undergraduate curriculum, such as Search and Optimisation, Artificial Intelligence, and Data Mining and Analytic Technologies. He is also committed to teaching Data Science and Computer Science to anyone, regardless of their educational background, e.g. his public videos on YouTube.

shahinrostami.com
YT: ShahinRostami
@ShahinRostami
Github: shahinrostami
u/shahinrostami
Patreon: StamiLabs.com
IG: Data.Crayon

Table of Contents

version 2021.2.10

Preface

Preface

The Rust programming language has become a popular choice amongst software engineers since its release in 2010. Besides being something new and interesting, Rust promised to offer exceptional performance and reliability. In particular, Rust achieves memory-safety and thread-safety through its ownership model. Instead of runtime checks, this safety is assured at compile time by Rust's borrow checker. This prevents undefined behaviour such as dangling pointers!

```
println!("Hello World!");
```

```
Hello World!
```

I first encountered Rust sometime around 2015 when I was updating my teaching materials on memory management in C. A year later in 2016, I implemented a simple multi-objective evolutionary algorithm in Rust as an academic exercise (available: https://github.com/shahinrostami/simple_ea). I didn't have any formal training with Rust, nor did I complete any tutorial series, I just figured things out using the documentation as I progressed through my project.

Some example code from this project takes ZDT1 from its mathematical expression in Equation 1

$$
\begin{aligned}
f_1(x_1) &= x_1 \\
f_2(x) &= g \cdot h \\
g(x_2, \ldots, x_{\mathrm{D}}) &= 1 + 9 \cdot \sum_{d=2}^{\mathrm{D}} \frac{x_d}{(D-1)} \\
h(f_1, g) &= 1 - \sqrt{f1/g}
\end{aligned}
\tag{1}
$$

to the Rust implementation below.

```
pub fn zdt1(parameters: [f32; 30]) -> [f32; 2] {

    let f1 = parameters[0];
    let mut g = 1_f32;

    for i in 1..parameters.len() {
        g = g + ((9_f32 / (parameters.len() as f32 - 1_f32)) *
parameters[i]);
    }

    let h = 1_f32 - (f1 / g).sqrt();
    let f2 = g * h;

    return [f1, f2];
}
```

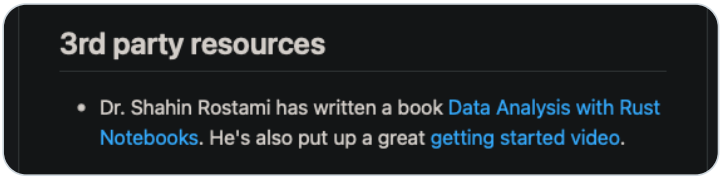
It was interesting to see that since writing this code in 2016, some of my dependencies have been deprecated and replaced.

My greatest challenge was breaking away from what I already knew. Until this point, I was familiar with languages such as C, C++, C#, Java, Python, MATLAB, etc., with the majority of my time spent working with memory managed languages. I found myself resisting Rust's intended usage, and it is still something I'm working on.

Now that I am about to commence my sabbatical from my University post, I've decided to try Rust again. This time, I'm going to write a book which focusses on using Rust and Jupyter Notebooks to implement algorithms and conduct experiments, most likely in the fields of search, optimisation, and machine learning. Can we write and execute all our code in a Jupyter Notebook? Yes! *Should* we? Probably not. However, I enjoy the workflow, and making this an enjoyable process is important to me.

> **Note**
>
> I aim to generate everything in this book through code. This means you will see the code for all my figures and tables, including things like flowcharts.
>
> This book is currently available in early access form. It is being actively worked on and updated.
>
> Every section is intended to be independent, so you will find some repetition as you progress from one section to another.

github.com/google/evcxr/c... #rustlang #rust



6:54 PM · Dec 7, 2020

7 1 Copy link to Tweet

Setup Anaconda, Jupyter, and Rust

Contents Download Source

Software Setup

We are taking a practical approach in the following sections. As such, we need the right tools and environments available in order to keep up with the examples and exercises. We will be using Rust along with packages that will form our scientific stack, such as `ndarray` (for multi-dimensional containers) and `plotly` (for interactive graphing), etc. We will write all of our code within a Jupyter Notebook, but you are free to use other IDEs.

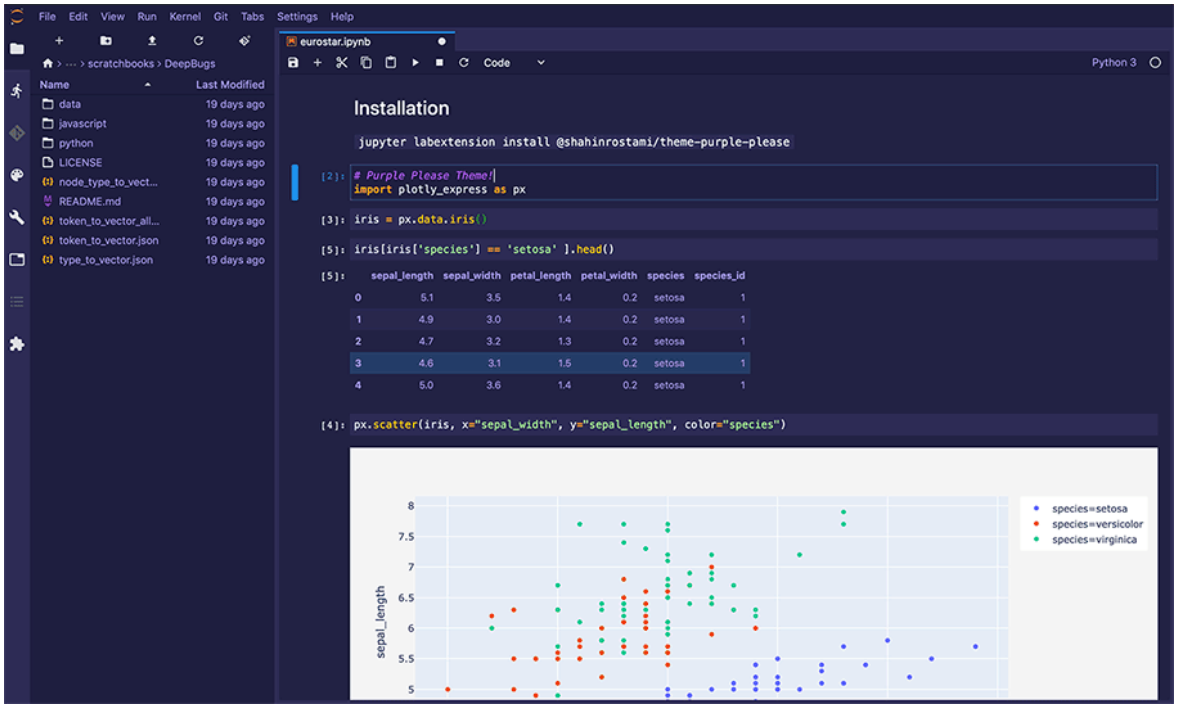


Figure 1 - A Jupyter Notebook being edited within Jupyter Lab.
Theme from https://github.com/shahinrostami/theme-purple-please

Install Miniconda

There are many different ways to get up and running with an environment that will facilitate our work. One approach I can recommend is to install and use Miniconda.

> Miniconda is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages they depend on, and a small number of other useful packages, including pip, zlib and a few others.
>
> — *https://docs.conda.io/en/latest/miniconda.html*

You can skip Miniconda entirely if you prefer and install Jupyter Lab directly, however, I prefer using it to manage other environments too.

You can find installation instructions for Miniconda on their website, but if you're using Linux (e.g. Ubuntu) you can execute the following commands from in your terminal:

```
wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
chmod +x Miniconda3-latest-Linux-x86_64.sh
./Miniconda3-latest-Linux-x86_64.sh
```

This will download the installation files and start the interactive installation process. Follow the process to the end, where you should see the following message:

> Thank you for installing Miniconda3!

All that's left is to close and re-open the terminal window.

Create Your Environment

Once Miniconda is installed, we need to create and configure our environment. If you added Miniconda to your PATH environment during the installation process, then you can run these commands directly from Terminal, Powershell, or CMD.

Now we can create and configure our conda environment using the following commands.

```
conda create -n darn python=3
```

You can replace `darn` (**D**ata **An**alytics with **R**ust **N**otebooks) with a name of your choosing.

This will create a conda environment named `darn` with the latest Python 3 package ready to go. You should be presented with a list of packages that will be installed and asked if you wish to proceed. To do so, just enter the character `y`. If this operation is successful, you should see the following output at the end:

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate darn
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

As the message suggests, you will need to type the following command to activate and start entering commands within our environment named `darn`.

```
conda activate darn
```

Once you do that, you should see your terminal prompt now leads with the environment name within parentheses:

```
(darn) melica:~ shahin$
```

> **Note**
>
> The example above shows the macOS machine name "melica" and the user "shahin". You will see something different on your machine, and it may appear in a different format on a different operating system such as Windows. As long as the prompt leads with "(darn)", you are on the right track.

This will allow you to identify which environment you are currently operating in. If you restart your machine, you should be able to use `conda activate darn` within your conda prompt to get back into the same environment.

Install Packages

If your environment was already configured and ready, you would be able to enter the command `jupyter lab` to launch an instance of the Jupyter Lab IDE in the current directory. However, if we try that in our newly created environment, we will receive an error:

```
(darn) melica:~ shahin$ jupyter lab
-bash: jupyter: command not found
```

So let's fix that. Let's install Jupyter Lab and use the `-y` option which automatically says "yes" to any questions asked during the installation process.

```
conda install -c conda-forge jupyterlab=2.2.9
```

We'll also need `cmake` later on.

```
conda install -c anaconda cmake -y
```

Finally, let's install nodejs. This is needed to run our Jupyter Lab extension in the next section.

```
conda install -c conda-forge nodejs=15 -y
```

Install Jupyer Lab Extensions

There's one last thing we need to do before we move on, and that's installing any Jupyter Lab extensions that we may need. One particular extension that we need is the plotly extension, which will allow our Jupyter Notebooks to render our Plotly visualisations. Within your conda environment, simply run the following command:

```
jupyter labextension install jupyterlab-plotly
```

This may take some time, especially when it builds your jupyterlab assets, so keep an eye on it until you're returned control over the conda prompt, i.e. when you see the following:

```
(darn) melica:~ shahin$
```

Optionally, you may wish to install the purple looking theme from Figure 1 above.

```
jupyter labextension install @shahinrostami/theme-purple-please
```

Now we're good to go!

Install Rust

Now we'll install Rust using rustup, but you can check out the other installation methods if you need them.

```
curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh
```

The code samples in this book will work in many versions of Rust, but I can confirm them to be working with version `1.42.0`. You can get the same version with:

```
rustup default 1.42.0
```

You will be given instructions for adding Cargo's bin directory to your PATH environment variable.

```
source $HOME/.cargo/env
```

This will work until your close your terminal, so make sure to add it to your shell profile. I use Z shell (Zsh) so this meant adding the following to `.zshrc`:

```
export PATH="$HOME/.cargo/bin:$PATH"
```

You can make sure everything works by closing and re-opening your terminal and typing `cargo`. If this returns the usage documentation then you're all set.

> **Note**
>
> Don't forget to activate your environment when opening the terminal.

Install the EvCxR Jupyter Kernel

Now we'll install the EvCxR Jupyter Kernel. If you're wondering how it's pronounced, it's been mentioned to be "Evic-ser". This is what will allow us to execute Rust code in a Jupyter Notebook.

You can get other installation methods methods for EvCxR if you need then, but we will be using:

```
cargo install evcxr_jupyter --version 0.5.3
evcxr_jupyter --install
```

A Quick Test

Let's test if everything is working as it should be. In your conda prompt, within your conda environment, run the following command

```
jupyter lab
```

This should start the Jupyter Lab server and launch a browser window with the IDE ready to use.

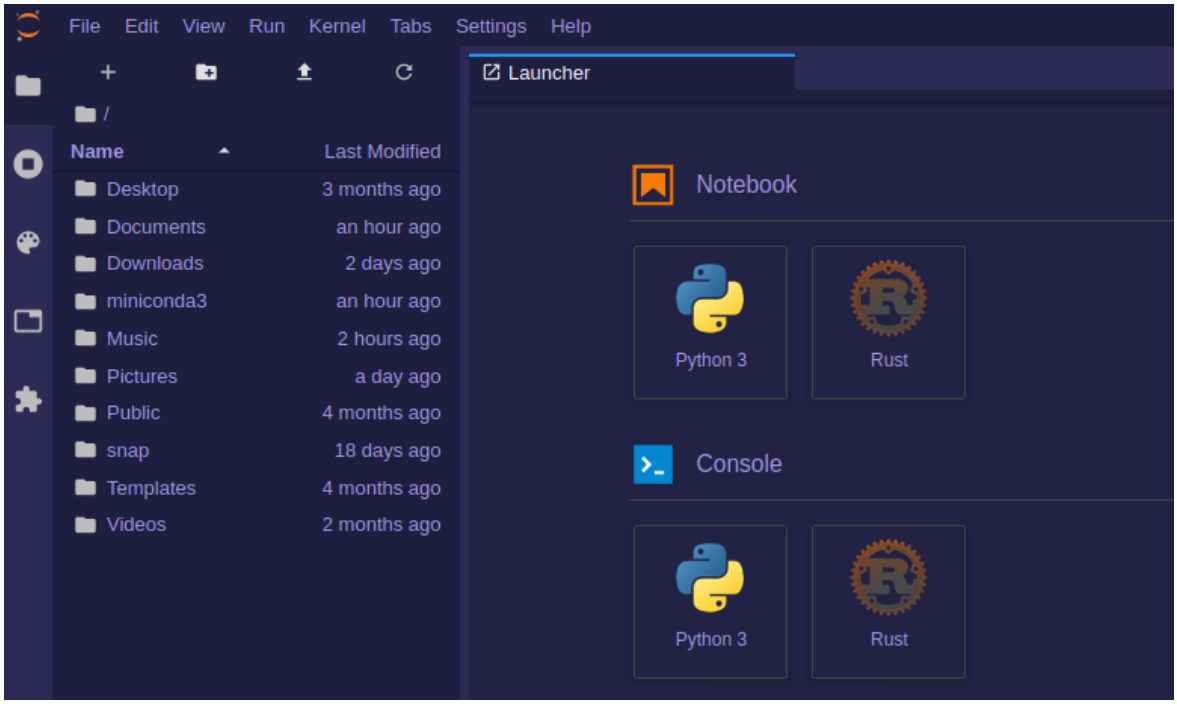


Figure 2 - A fresh installation of Jupyter Lab.

Let's create a new notebook. In the Launcher tab which has opened by default, click "Rust" under the Notebook heading. This will create a new and empty notebook named `Untitled.ipynb` in the current directory.

If everything is configured as it should be, you should see no errors. Type the following into the first cell and click the "play" button to execute it and create a new cell.

```
println!("Hello World!");
```

```
Hello World!
```

If we followed all the instructions and didn't encounter any errors, everything should be working. We should see "Hello World!" in the output cell.

Conclusion

In this section, we've downloaded, installed, configured, and tested our environment such that we're ready to run the following examples and experiments. If you ever find that you're missing Jupyter Lab packages, you can install them in the same way as we installed Jupyter Lab and the others in this section.

Multidimensional Arrays and Operations with NDArray

Contents Download Source

Preamble

```
:dep ndarray = {version = "0.13.1"}
extern crate ndarray;
```

This module contains the most used types, type aliases, traits and functions that you can import easily as a group:

```
use ndarray::prelude::*;
```

This gives us access to the following: `ArrayBase`, `Array`, `RcArray`, `ArrayView`, `ArrayViewMut`, `Axis`, `Dim`, `Dim`, `Dimension`, `Array0`, `Array1`, `Array2`, `Array3`, `Array4`, `Array5`, `Array6`, `ArrayD`, `ArrayView0`, `ArrayView1`, `ArrayView2`, `ArrayView3`, `ArrayView4`, `ArrayView5`, `ArrayView6`, `ArrayViewD`, `ArrayViewMut0`, `ArrayViewMut1`, `ArrayViewMut2`, `ArrayViewMut3`, `ArrayViewMut4`, `ArrayViewMut5`, `ArrayViewMut6`, `ArrayViewMutD`, `Ix0`, `Ix0`, `Ix1`, `Ix1`, `Ix2`, `Ix2`, `Ix3`, `Ix3`, `Ix4`, `Ix4`, `Ix5`, `Ix5`, `Ix6`, `Ix6`, `IxDyn`, `IxDyn`, `arr0`, `arr1`, `arr2`, `aview0`, `aview1`, `aview2`, `aview_mut1`, `ShapeBuilder`, `NdFloat`, and `AsArray`.

Introduction

The `ndarray` crate provides us with a multidimensional container that can contain general or numerical elements. If you're familiar with Python, then you can consider it to be similar to the `numpy` package. With `ndarray` we get our n-dimensional arrays, slicing, views, mathematical operations, and more. We'll need these in later sections to load in our datasets into containers that we can operate on and conduct our analyses.

Creating Arrays

From a Vector

Let's take a look at how we can create a two-dimensional ndarray `Array` from a `Vec` with the `arr2()` function.

```
arr2(&[[1.,2.,3.],
       [4.,5.,6.]])
```

```
[[1.0, 2.0, 3.0],
 [4.0, 5.0, 6.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

It's as easy as that, This has given us a 2 by 3 array with our desired floating point values. We can also use the `array!` macro as a shorthand for creating an array.

```
array![[1.,2.,3.],
       [4.,5.,6.]]
```

```
[[1.0, 2.0, 3.0],
 [4.0, 5.0, 6.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

Filled with Zeros

We can also construct an array filled with zeros, we can do this with the `zeros()` function and pass in our desired shape.

```
Array2::<f64>::zeros((4,4))
```

```
[[0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0, 0.0]], shape=[4, 4], strides=[4, 1], layout=C (0x1), const
ndim=2
```

Filled with Ones

Similarly, we can also construct an array filled with ones, we can do this with the `ones()` function and pass in our desired shape.

```
Array2::<f64>::ones((4,4))
```

```
[[1.0, 1.0, 1.0, 1.0],
 [1.0, 1.0, 1.0, 1.0],
 [1.0, 1.0, 1.0, 1.0],
 [1.0, 1.0, 1.0, 1.0]], shape=[4, 4], strides=[4, 1], layout=C (0x1), const
ndim=2
```

Let's create variables to store a 1D array and a 2D array for use in the following subsections.

```
let data_1D: Array1::<f32> = array![1.,2.,3.];

let data_2D: Array2::<f32> = array![[1.,2.,3.],
                                    [4.,5.,6.]];
```

Dimensions

It's often the case that we need to find out the dimensionality of our arrays. There are many ways to do this, and the following contains some of the common approaches.

From Length

We can use `Array.len()` to return the shape along a single axis.

```
data_1D.len()
```

```
3
```

This is simple enough if we have a one-dimensional array. However, for higher dimensions, we can see that for a `len()` returns the flattened length.

```
data_2D.len()
```

```
6
```

If we want to get the length along one of the axes instead, e.g. the second one, we can use `Array.len_of(Axis(n))`

```
data_2D.len_of(Axis(1))
```

```
3
```

From Shape

Another approach is to use `Array.shape()` which returns more information.

```
data_2D.shape()
```

```
[2, 3]
```

We can see it has returned an array that indicates the length along all of our axes. This can be indexed to get the length along a specific axis.

```
data_2D.shape()[1]
```

```
3
```

Indexing

Like most data structures, the indexing starts at 0. To access the first element in our one-dimensional arrays we can do the following.

```
data_1D[0]
```

```
1.0
```

For higher dimensions, we need to use a primitive array.

```
data_2D[[0,0]]
```

```
1.0
```

Likewise, to access the second element in our one-dimensional arrays we need to index with 1.

```
data_1D[1]
```

```
2.0
```

Again, for our higher dimensions, we use a primitive array..

```
data_2D[[0,1]]
```

```
2.0
```

To select the last element in our one-dimensional arrays we can index with `Array.len() -1` .

```
data_1D[data_1D.len() -1]
```

```
3.0
```

But for our multidimensional arrays we need to use a primitive array and use `Array.len_of(Axis(n))`.

```
data_2D[[0, data_2D.len_of(Axis(1)) -1]]
```

```
3.0
```

Alternatively, we could use `Array.shape()[n]`.

```
data_2D[[0, data_2D.shape()[1] - 1]]
```

```
3.0
```

Mathematics

Let's look at some common mathematical operations that can operate on our arrays.

Summing Array Elements

All elements in an array can be summed with `sum()`.

```
data_2D.sum()
```

```
21.0
```

We may instead wish to sum all elements along a specific axis in an array, e.g. the first axis.

```
data_2D.sum_axis(Axis(0))
```

```
[5.0, 7.0, 9.0], shape=[3], strides=[1], layout=CF (0x3), const ndim=1
```

Or the second axis:

```
data_2D.sum_axis(Axis(1))
```

```
[6.0, 15.0], shape=[2], strides=[1], layout=CF (0x3), const ndim=1
```

Element-wise Operations

It's quite common to apply mathematical operations to each element of an array. Let's have a look at some examples.

Addition

We can add values, e.g. 1.0, to every element.

```
&data_2D + 1.0
```

```
[[2.0, 3.0, 4.0],
 [5.0, 6.0, 7.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

We can also add the elements of one array to another.

```
&data_2D + &data_2D
```

```
[[2.0, 4.0, 6.0],
 [8.0, 10.0, 12.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

Finally, we can add a one-dimensional array to a two-dimensional array.

```
&data_2D + &data_1D
```

```
[[2.0, 4.0, 6.0],
 [5.0, 7.0, 9.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

> **Warning**
>
> When summing two arrays together they don't need to have the same shape, but their shapes must be compatible. This means we should be able to broadcast one array across another, i.e. they must be identical in the size of at least one dimension.

Subtraction

We can subtract values, e.g. 1.0, from every element.

```
&data_2D - 1.0
```

```
[[0.0, 1.0, 2.0],
 [3.0, 4.0, 5.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

We can also subtract elements of one array from another.

```
&data_2D - &data_2D
```

```
[[0.0, 0.0, 0.0],
 [0.0, 0.0, 0.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

Finally, we can subtract a one-dimensional array from a two-dimensional array array.

```
&data_2D - &data_1D
```

```
[[0.0, 0.0, 0.0],
 [3.0, 3.0, 3.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

Multiplication

We can multiply every element by a value, e.g. by 2.0.

```
&data_2D * 2.0
```

```
[[2.0, 4.0, 6.0],
 [8.0, 10.0, 12.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

We can also multiply every element of one array by another.

```
&data_2D * &data_1D
```

```
[[1.0, 4.0, 9.0],
 [4.0, 10.0, 18.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

Division

We can divide every element by a value, e.g. by 2.0.

```
&data_2D / 2.0
```

```
[[0.5, 1.0, 1.5],
 [2.0, 2.5, 3.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

We can also divide every element of one array by another.

```
&data_2D / &data_1D
```

```
[[1.0, 1.0, 1.0],
 [4.0, 2.5, 2.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const ndim=2
```

Power

We can raise the elements in an array to a power, e.g. of 3.0.

```
data_2D.mapv(|data_2D| data_2D.powi(3))
```

```
[[1.0, 8.0, 27.0],
 [64.0, 125.0, 216.0]], shape=[2, 3], strides=[3, 1], layout=C (0x1), const
ndim=2
```

Square root

We can calculate the square root of elements in an array. The specified data type must match.

```
data_2D.mapv(f32::sqrt)
```

```
[[1.0, 1.4142135, 1.7320508],
 [2.0, 2.236068, 2.4494898]], shape=[2, 3], strides=[3, 1], layout=C (0x1),
const ndim=2
```

Conclusion

In this section, we've introduced `ndarray` as a crate that gives us multidimensional containers and operations. We demonstrated how to create arrays, find out their dimensionality, index them, and how to invoke some basic mathematical operations.

Visualisation of Co-occurring Types

Contents Download Source

Preamble

```
:dep darn = {version = "0.3.0"}
:dep ndarray = {version = "0.13.1"}
:dep itertools = {version = "0.9.0"}
:dep chord = {Version = "0.1.6"}
extern crate ndarray;

use ndarray::prelude::*;
use itertools::Itertools;
use chord::{Chord, Plot};
```

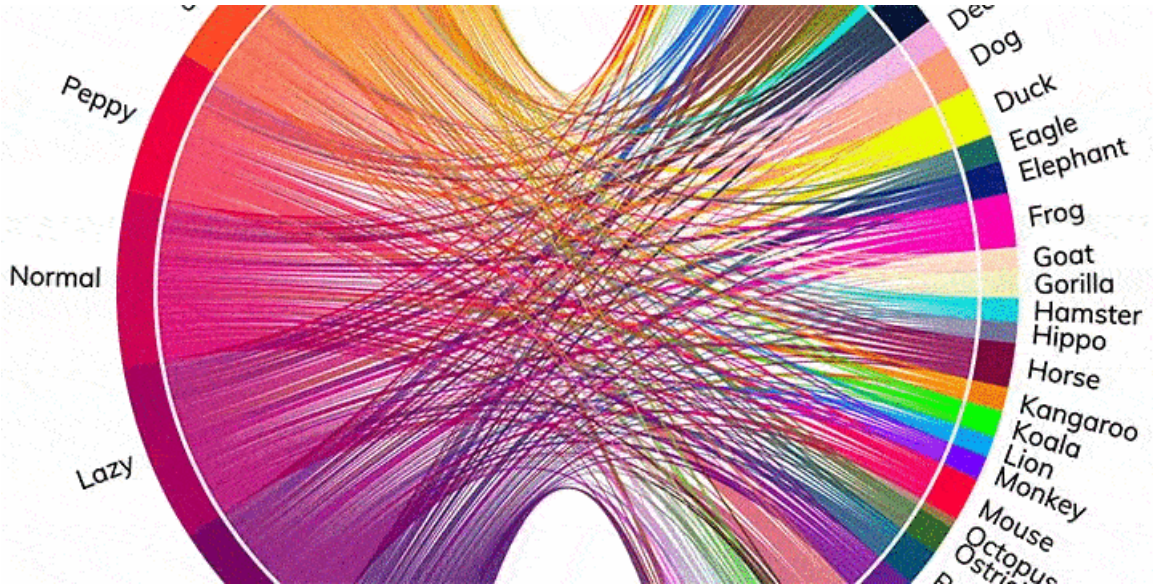
Introduction

In this section, we're going to use the Complete Pokemon Dataset dataset to visualise the co-occurrence of Pokémon types from generations one to eight. We'll make this happen using a *chord diagram*.

Chord Diagrams

In a chord diagram (or radial network), entities are arranged radially as segments with their relationships visualised by arcs that connect them. The size of the segments illustrates the numerical proportions, whilst the size of the arc illustrates the significance of the relationships[1].

Chord diagrams can be useful when trying to convey relationships between different entities, and they can be beautiful and eye-catching. They can get messy when considering many entities, so it's often beneficial to make them interactive and explorable.



The Chord Crate

I wasn't able to find any Rust crates for plotting chord diagrams, so I ported my own (based on d3-chord) from Python to Rust.

You can get the crate either from crates.io or from the GitHub repository. With your processed data, you should be able to plot something beautiful with just a single line, `Chord{ matrix : matrix, names : names, .. Chord::default() }.show()`. To enable the pro features of the `chord` crate check out Chord Pro.

The Dataset

The dataset documentation states that we can expect two *type* variables per each of the 1028 samples of the first eight generations, `type_1`, and `type_2`.

Let's download the mirrored dataset and have a look for ourselves.

```
let data =
darn::read_csv("https://datacrayon.com/datasets/pokemon_gen_1_to_8.csv");
```

```
darn::show_frame(&data.0, Some(&data.1));
```

	pokedex_number	name	german_name	japanese_name	generation	status
"0"	"1"	"Bulbasaur"	"Bisasam"	"フシギダネ (Fushigidane)"	"1"	"Normal"
"1"	"2"	"Ivysaur"	"Bisaknosp"	"フシギソウ (Fushigisou)"	"1"	"Normal"
"2"	"3"	"Venusaur"	"Bisaflor"	"フシギバナ (Fushigibana)"	"1"	"Normal"
"3"	"3"	"Mega Venusaur"	"Bisaflor"	"フシギバナ (Fushigibana)"	"1"	"Normal"
"4"	"4"	"Charmander"	"Glumanda"	"ヒトカゲ	"1"	"Normal"

				(Hitokage)"		
...	...	...	...	...	...	...
"1023"	"888"	"Zacian Hero of Many Battles"	""	""	"8"	"Legendary"
"1024"	"889"	"Zamazenta Crowned Shield"	""	""	"8"	"Legendary"
"1025"	"889"	"Zamazenta Hero of Many Battles"	""	""	"8"	"Legendary"
"1026"	"890"	"Eternatus"	""	""	"8"	"Legendary"
"1027"	"890"	"Eternatus Eternamax"	""	""	"8"	"Legendary"

It looks good so far, we can clearly see the two type columns. Let's confirm that we have 1028 samples.

```
&data.0.shape()
```

```
[1028, 51]
```

Perfect, that's exactly what we were expecting.

Data Wrangling

We need to do a bit of data wrangling before we can visualise our data. We can see from the column names that the Pokémon types are split between the columns `type_1` and `type_2`.

```
&data.1
```

```
["", "pokedex_number", "name", "german_name", "japanese_name", "generation",
"status", "species", "type_number", "type_1", "type_2", "height_m",
"weight_kg", "abilities_number", "ability_1", "ability_2", "ability_hidden",
"total_points", "hp", "attack", "defense", "sp_attack", "sp_defense", "speed",
"catch_rate", "base_friendship", "base_experience", "growth_rate",
"egg_type_number", "egg_type_1", "egg_type_2", "percentage_male", "egg_cycles",
"against_normal", "against_fire", "against_water", "against_electric",
"against_grass", "against_ice", "against_fight", "against_poison",
"against_ground", "against_flying", "against_psychic", "against_bug",
"against_rock", "against_ghost", "against_dragon", "against_dark",
"against_steel", "against_fairy"]
```

So let's select just these two columns and work with a list containing only them as we move forward.

```
let types = data.0.slice(s![.., 9..11]).into_owned();
darn::show_frame(&types, None);
```

"Grass"	"Poison"
"Grass"	"Poison"
"Grass"	"Poison"
"Grass"	"Poison"
"Fire"	""
...	...
"Fairy"	""
"Fighting"	"Steel"
"Fighting"	""
"Poison"	"Dragon"
"Poison"	"Dragon"

Our chord diagram will need two inputs: the co-occurrence matrix, and a list of names to label the segments.

First, we'll populate our list of type names by looking for the unique ones.

```
let mut names = types.iter().cloned().unique().collect_vec();
names
```

```
["Grass", "Poison", "Fire", "", "Flying", "Dragon", "Water", "Bug", "Normal",
"Dark", "Electric", "Psychic", "Ground", "Ice", "Steel", "Fairy", "Fighting",
"Rock", "Ghost"]
```

Let's sort this alphabetically.

```
names.sort();
names
```

```
["", "Bug", "Dark", "Dragon", "Electric", "Fairy", "Fighting", "Fire",
"Flying", "Ghost", "Grass", "Ground", "Ice", "Normal", "Poison", "Psychic",
"Rock", "Steel", "Water"]
```

We'll also remove the empty string that has appeared as a result of samples with only one type.

```
names.remove(0);
names
```

```
["Bug", "Dark", "Dragon", "Electric", "Fairy", "Fighting", "Fire", "Flying",
"Ghost", "Grass", "Ground", "Ice", "Normal", "Poison", "Psychic", "Rock",
"Steel", "Water"]
```

Now we can create our empty co-occurrence matrix with a shape that can hold co-occurrences between our types.

```
let type_count = names.len();
let mut matrix: Vec<Vec<f64>> = vec![vec![Default::default(); type_count];
type_count];
matrix
```

```
[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
```

We can populate a co-occurrence matrix with the following approach. Here, we're looping through every sample in our dataset and incrementing the corresponding matrix entry by one using the `type_1` and `type_2` indices from the `names` vector. To make sure we have a co-occurrence matrix, we're also doing the same in reverse, i.e. `type_2` and `type_1`.

```
for item in types.genrows() {
    if(!item[0].is_empty() && !item[1].is_empty()) {
        matrix[names.iter().position(|s| s == &item[1]).unwrap()]
              [names.iter().position(|s| s == &item[0]).unwrap()] += 1.0;
        matrix[names.iter().position(|s| s == &item[0]).unwrap()]
              [names.iter().position(|s| s == &item[1]).unwrap()] += 1.0;
    };
};
```

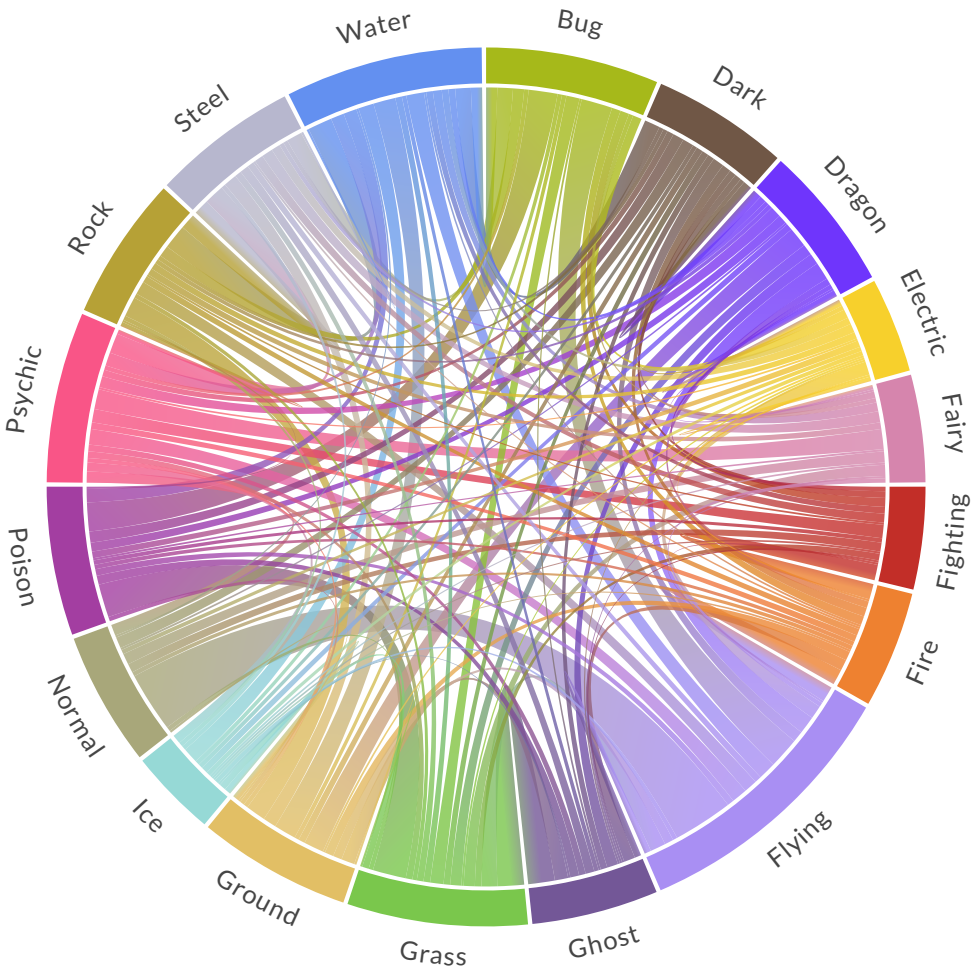
Chord Diagram

Time to visualise the co-occurrence of types using a chord diagram. We are going to use a list of custom colours that represent the types.

```
let colors: Vec<String> = vec![
    "#A6B91A", "#705746", "#6F35FC", "#F7D02C", "#D685AD",
    "#C22E28", "#EE8130", "#A98FF3", "#735797", "#7AC74C",
    "#E2BF65", "#96D9D6", "#A8A77A", "#A33EA1", "#F95587",
    "#B6A136", "#B7B7CE", "#6390F0"
]
.into_iter()
.map(String::from)
.collect();
```

Finally, we can put it all together.

```
Chord {
    matrix: matrix.clone(),
    names: names.clone(),
    colors: colors,
    margin: 30.0,
    wrap_labels: true,
    ..Chord::default()
}
.show();
```



Conclusion

In this section, we demonstrated how to conduct some data wrangling on a downloaded dataset to prepare it for a chord diagram. Our chord diagram is interactive, so you can use your mouse or touchscreen to investigate the co-occurrences!

Box Plots at the Olympics

Contents Download Source

Preamble

```
:dep darn = {version = "0.3.0"}
:dep ndarray = {version = "0.13.1"}
:dep itertools = {version = "0.9.0"}
:dep plotly = {version = "0.4.0"}
extern crate ndarray;

use ndarray::prelude::*;
use std::str::FromStr;
use itertools::Itertools;
use plotly::{Plot, Layout, BoxPlot};
use plotly::common::{Title, Font};
use plotly::layout::{Margin, Axis};
```

Introduction

In this section, we're going to use 120 years of Olympic history to create two visualisations. Let's set our sights on something that illustrates the age and height in athletes grouped by the different Olympic games.

The Dataset

We'll use the 120 years of Olympic history: athletes and results dataset, which we'll download and load with the `darn` crate. You're also welcome to use the mirrored that has been used in the following cell.

```
let data =
darn::read_csv("https://datacrayon.com/datasets/athlete_events_known_age.csv"
);
```

We'll take a peek at what we've downloaded to make sure there were no issues with the loading.

```
darn::show_frame(&data.0, Some(&data.1));
```

ID	Name	Sex	Age	Height	Weight	Team	NOC	Games	Year	Seas
"1"	"A Dijiang"	"M"	"24"	"180"	"80"	"China"	"CHN"	"1992 Summer"	"1992"	"Summe
"2"	"A Lamusi"	"M"	"23"	"170"	"60"	"China"	"CHN"	"2012 Summer"	"2012"	"Summe
"5"	"Christine Jacoba Aaftink"	"F"	"21"	"185"	"82"	"Netherlands"	"NED"	"1988 Winter"	"1988"	"Winte
"5"	"Christine Jacoba Aaftink"	"F"	"21"	"185"	"82"	"Netherlands"	"NED"	"1988 Winter"	"1988"	"Winte
"5"	"Christine Jacoba Aaftink"	"F"	"25"	"185"	"82"	"Netherlands"	"NED"	"1992 Winter"	"1992"	"Winte
...	...	...	...	...	...	...	...	...	...	
"135569"	"Andrzej ya"	"M"	"29"	"179"	"89"	"Poland-1"	"POL"	"1976 Winter"	"1976"	"Winte
"135570"	"Piotr ya"	"M"	"27"	"176"	"59"	"Poland"	"POL"	"2014 Winter"	"2014"	"Winte
"135570"	"Piotr ya"	"M"	"27"	"176"	"59"	"Poland"	"POL"	"2014 Winter"	"2014"	"Winte
"135571"	"Tomasz Ireneusz ya"	"M"	"30"	"185"	"96"	"Poland"	"POL"	"1998 Winter"	"1998"	"Winte
"135571"	"Tomasz	"M"	"34"	"185"	"96"	"Poland"	"POL"	"2002	"2002"	"Winte

```
                Ireneusz                                                Winter"
                   ya"
```

It looks like the data was loaded without any issues.

Data Wrangling

Let's assign the feature data to `games` and feature names to `headers` for readability.

```
let games = data.0;
let headers = data.1;
```

A quick look at the available features will give us the feature names we're after for the age and height of athletes.

```
println!("{}", &headers.iter().format("\n"));
```

```
ID
Name
Sex
Age
Height
Weight
Team
NOC
Games
Year
Season
```

We've confirmed that the two features we're after are named `Age` and `Height`, and that they're at index 3 and 4. However, it would be better to determine these indices programmatically instead of hard-coding them.

```
let idx_age = headers.iter().position(|x| x == "Age").unwrap();
let idx_height = headers.iter().position(|x| x == "Height").unwrap();
```

```
City
Sport
Event
Medal
```

Let's create an array of these indices and print them out to check.

```
let selected_features = [idx_age,idx_height];

println!("{}",selected_features.iter().format("\n"));
```

```
3
4
```

Now that we know the index of our age and height columns, let's prepare two collection variables, one named `features` to hold the numeric feature data, and one named `feature_headers` to hold the corresponding column names.

```
let mut features: Array2::<f32> =  Array2::<f32>::zeros((games.shape()
[0],0));
let mut feature_headers = Vec::<String>::new();
```

Now, we can copy and parse our feature data into initialised collections.

```
for &feature_index in selected_features.iter() {
    feature_headers.push(headers[feature_index].clone());
    features = ndarray::stack![Axis(1), features,
        games.column(feature_index as usize)
            .mapv(|elem| elem.parse::<f32>().unwrap())
            .insert_axis(Axis(1))
    ];
};
```

We'll take a peek to make sure there were no obvious issues with parsing.

```
darn::show_frame(&features, Some(&feature_headers));
```

Age	Height
24.0	180.0
23.0	170.0
21.0	185.0
21.0	185.0
25.0	185.0
...	...
29.0	179.0
27.0	176.0
27.0	176.0
30.0	185.0
34.0	185.0

Looking good. Next, we'll need to determine the different games available in our dataset - we'll be using these to group the age and height data.

```
let idx_sport = headers.iter().position(|x| x == "Sport").unwrap();
let unique_games =
games.column(idx_sport).iter().cloned().unique().collect_vec();

println!("{}",unique_games.iter().format(", "));
```

```
Basketball, Judo, Speed Skating, Cross Country Skiing, Athletics, Ice Hockey,
Badminton, Sailing, Biathlon, Gymnastics, Alpine Skiing, Handball,
Weightlifting, Wrestling, Luge, Rowing, Bobsleigh, Swimming, Football,
Equestrianism, Shooting, Taekwondo, Boxing, Fencing, Diving, Canoeing, Water
Polo, Tennis, Cycling, Hockey, Figure Skating, Softball, Archery, Volleyball,
Synchronized Swimming, Modern Pentathlon, Table Tennis, Nordic Combined,
Baseball, Rhythmic Gymnastics, Freestyle Skiing, Rugby Sevens, Trampolining,
Beach Volleyball, Triathlon, Ski Jumping, Curling, Golf, Snowboarding, Short
Track Speed Skating, Skeleton, Rugby, Art Competitions, Tug-Of-War
```

We now have the unique list of Olympic games - some of which you may not even have heard of!

Visualising the Data

Now that we have prepared our data, let's use all of our hard work in a box plot test.

Height of Athletes in Basketball

Let's see if we can create a box plot for the height of athletes in Basketball. To do so, we're going to build a list of row indices that correspond to Basketball data.

```
let mut count = -1;
let mut indices = Vec::<usize>::new();

let mask = games.column(idx_sport).map(|elem| {
    count += 1;
    if(elem == "Basketball") { indices.push(count as usize) };
    elem == "Basketball"
    }
);
```

Then, we'll use these indices to select from our feature data.

```
let basketball = features.select(Axis(0), &indices);
```

We'll take a peek to make sure there were no obvious issues with parsing.

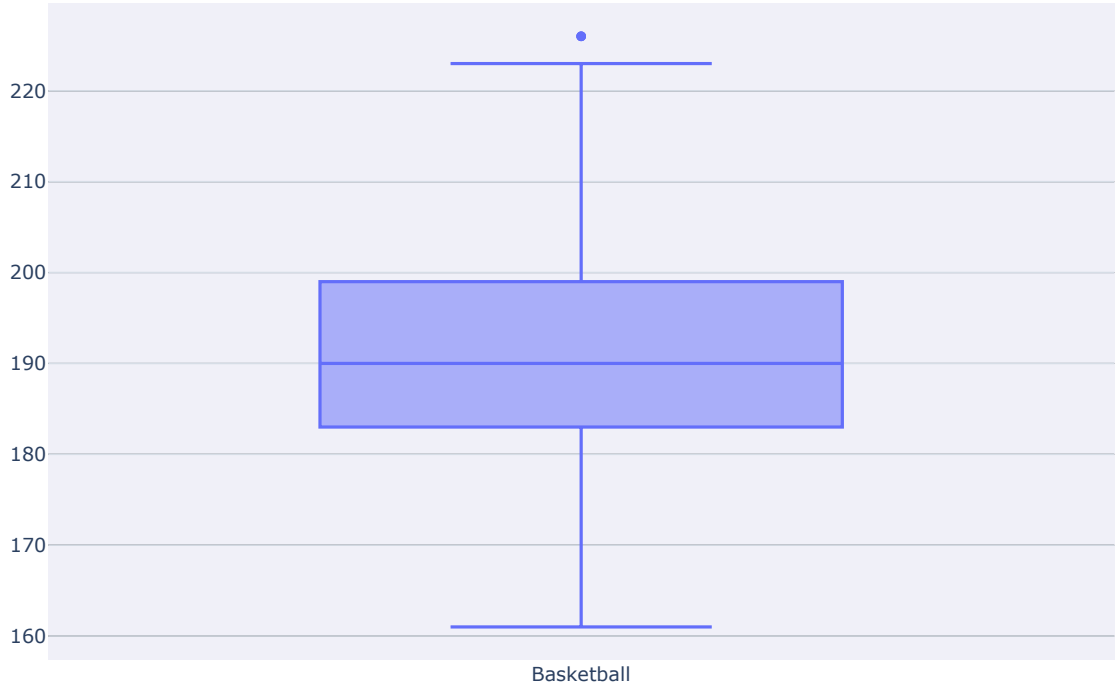
```
darn::show_frame(&basketball, Some(&feature_headers));
```

Age	Height
24.0	180.0

19.0	185.0
29.0	195.0
25.0	189.0
23.0	178.0
...	...
30.0	218.0
20.0	201.0
28.0	201.0
23.0	202.0
33.0	171.0

Finally, we'll create a box plot with just the height of the athletes in our dataset.

```
let mut plot = Plot::new();

let trace = BoxPlot::new(basketball.column(1).to_vec()).name("Basketball");

plot.add_trace(trace);

darn::show_plot(plot);
```



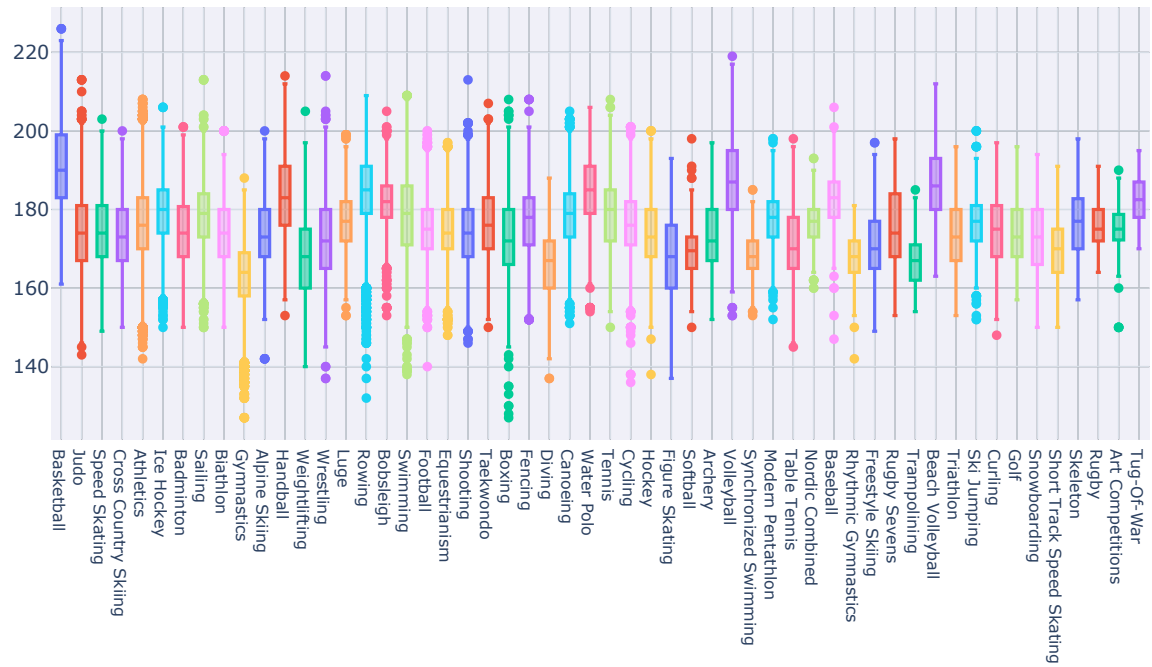
Looking good.

Athlete Height Grouped by Olympic Games

Now let's do the same as what we've just done for Basketball, but apply it to all the games in our dataset.

```
let mut plot = Plot::new();
let layout = Layout::new()
    .title(Title::new("Athlete height grouped by Olympic games."))
    .margin(Margin::new().left(30).right(0).bottom(140).top(40))
    .xaxis(Axis::new().show_grid(true).tick_font(Font::new().size(10)))
    .show_legend(false);

plot.set_layout(layout);

for name in unique_games.iter() {
    let mut count = -1;
    let mut indices = Vec::<usize>::new();
    let mask = games.column(idx_sport).map(|elem| {
        count += 1;
        if(elem == name) { indices.push(count as usize) };
        elem == "name"
        }
    );

    let game = features.select(Axis(0), &indices);
    let trace1 = BoxPlot::new(game.column(1).to_vec()).name(name);
    plot.add_trace(trace1);
};

darn::show_plot(plot);
```

Athlete height grouped by Olympic games.

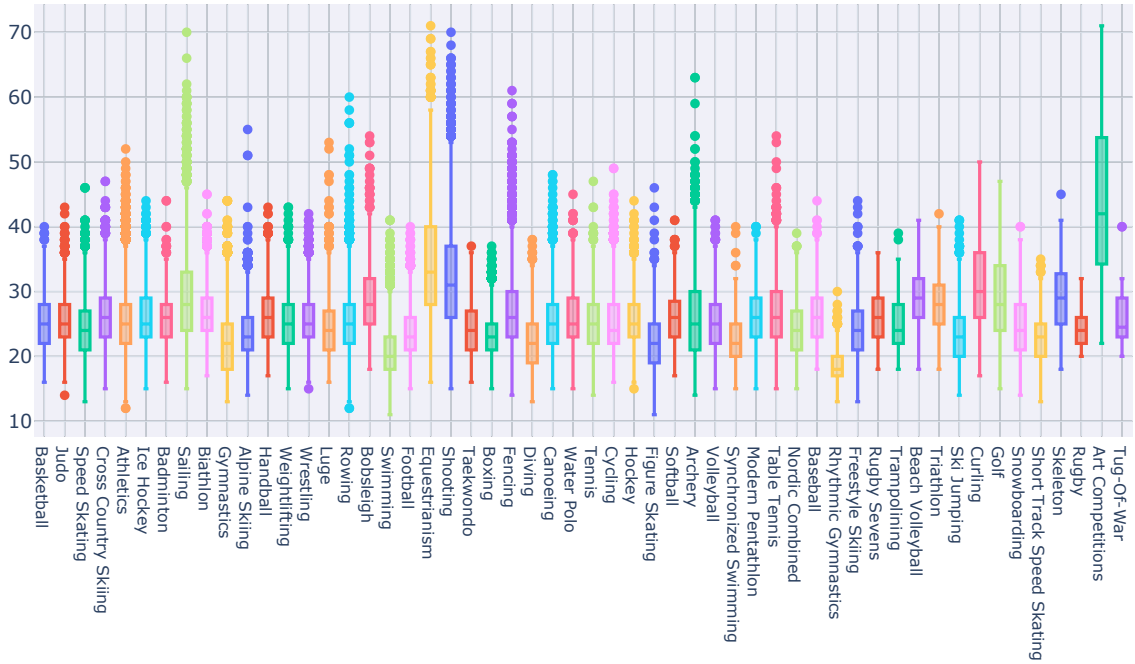


Athlete Age Grouped by Olympic Games

Let's repeat the last visualisation but this time for the age of athletes grouped by Olympic games.

```
let mut plot = Plot::new();
let layout = Layout::new()
    .title(Title::new("Athlete age grouped by Olympic games."))
    .margin(Margin::new().left(30).right(0).bottom(140).top(40))
    .xaxis(Axis::new().show_grid(true).tick_font(Font::new().size(10)))
    .show_legend(false);

plot.set_layout(layout);

for name in unique_games.iter() {
    let mut count = -1;
    let mut indices = Vec::<usize>::new();
    let mask = games.column(idx_sport).map(|elem| {
        count += 1;
        if(elem == name) { indices.push(count as usize) };
        elem == "name"
        }
    );

    let game = features.select(Axis(0), &indices);
    let trace1 = BoxPlot::new(game.column(0).to_vec()).name(name);
    plot.add_trace(trace1);
};

darn::show_plot(plot);
```

Athlete age grouped by Olympic games.



Conclusion

In this section, we worked towards illustrating the age and height of athletes grouped by games in the 120 years of Olympic history: athletes and results dataset. We avoided hard-coding where possible and presented the data in the form of multiple box plots.