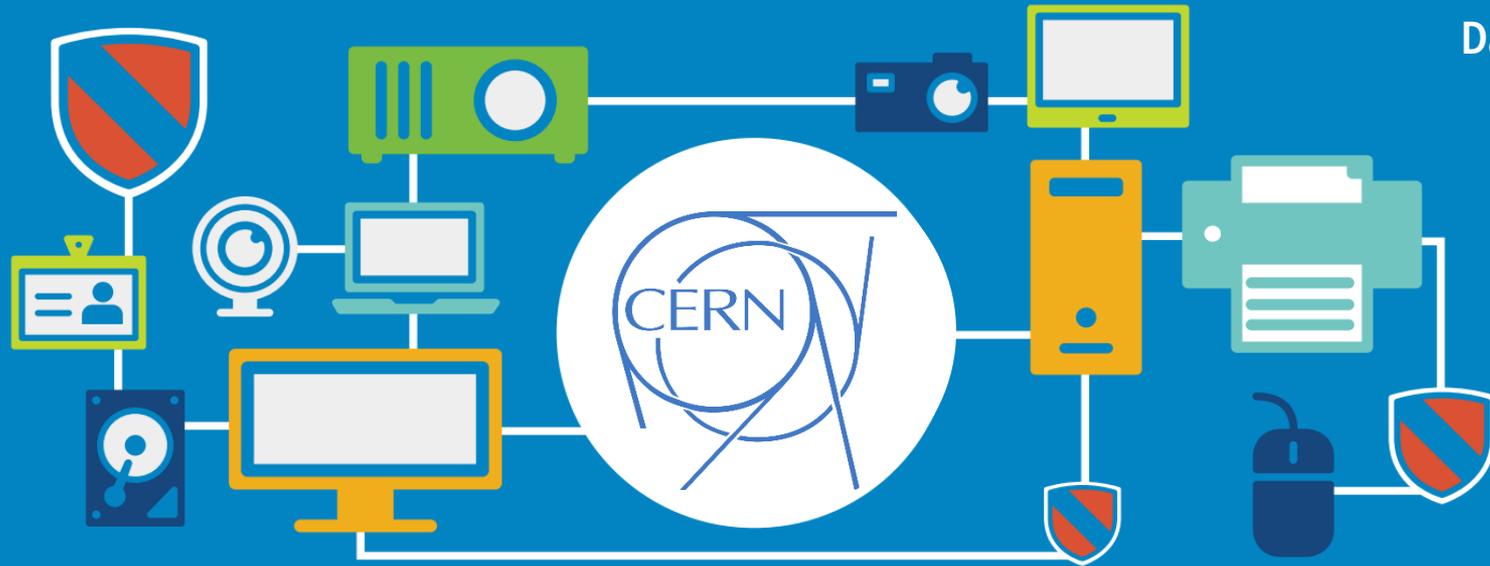




THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Presenter :
Harrison She

Supervisor:
Danilo Piparo



Summer Student 2017 - Optimising JS visualisation for notebooks

My Background



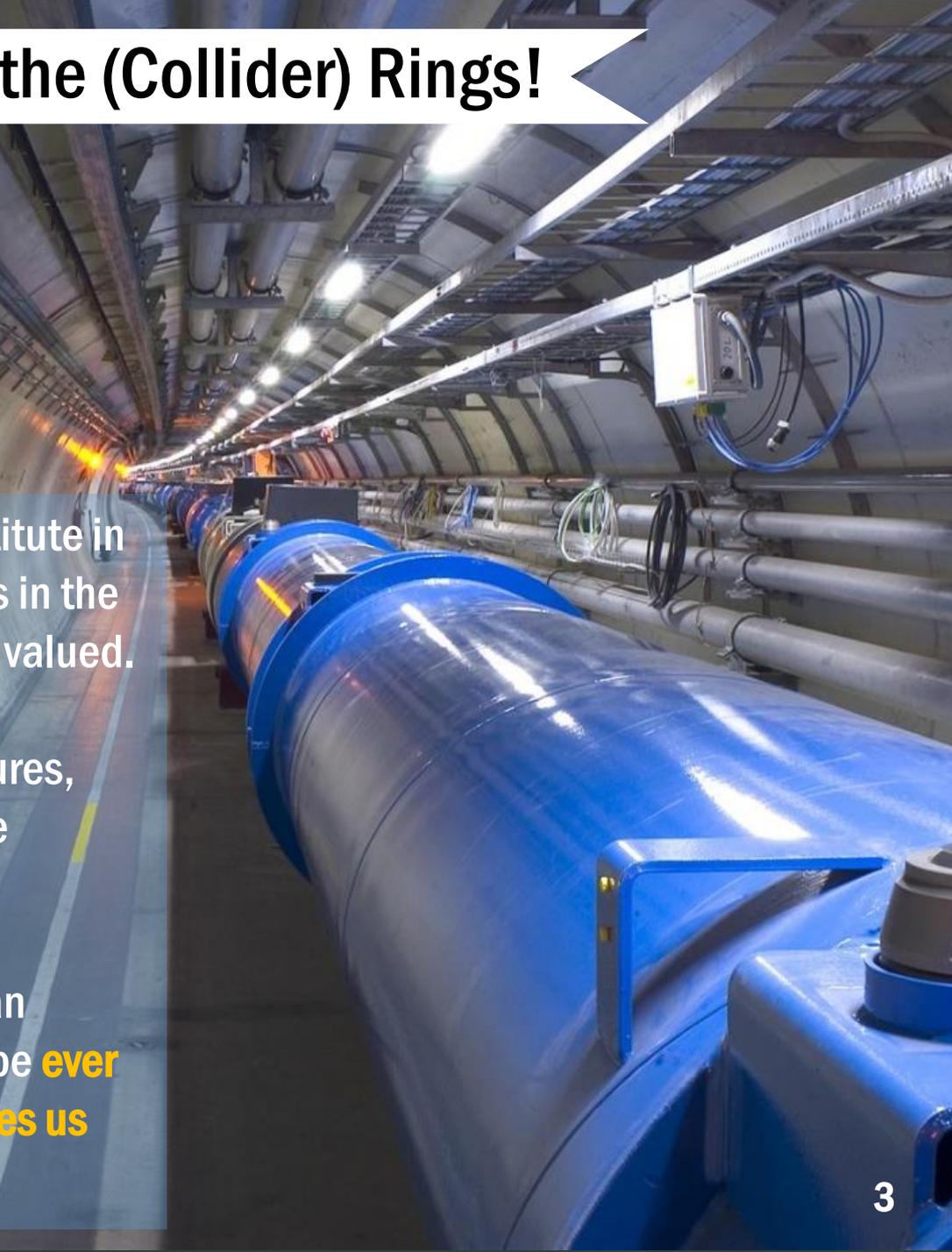
- Final year student from the University of Auckland, New Zealand.
- **Bachelor of Engineering (Honours)** specialising in Software Engineering, conjoint with a **Bachelor of Science** majoring in Physics.

Why did a kiwi apply for CERN **18,000km** away
(and enduring a **24 hr+** flight)?

The Lord of the (Collider) Rings!



- CERN is the preminent research institute in Computing, Engineering and Physics in the world. International collaboration is valued.
- Skills and knowledge gained in lectures, workshops and the project will prove invaluable.
- CERN represents an ideal that we can always strive to discover more, and be **ever curious about nature, it is what makes us human after all.**



First Steps

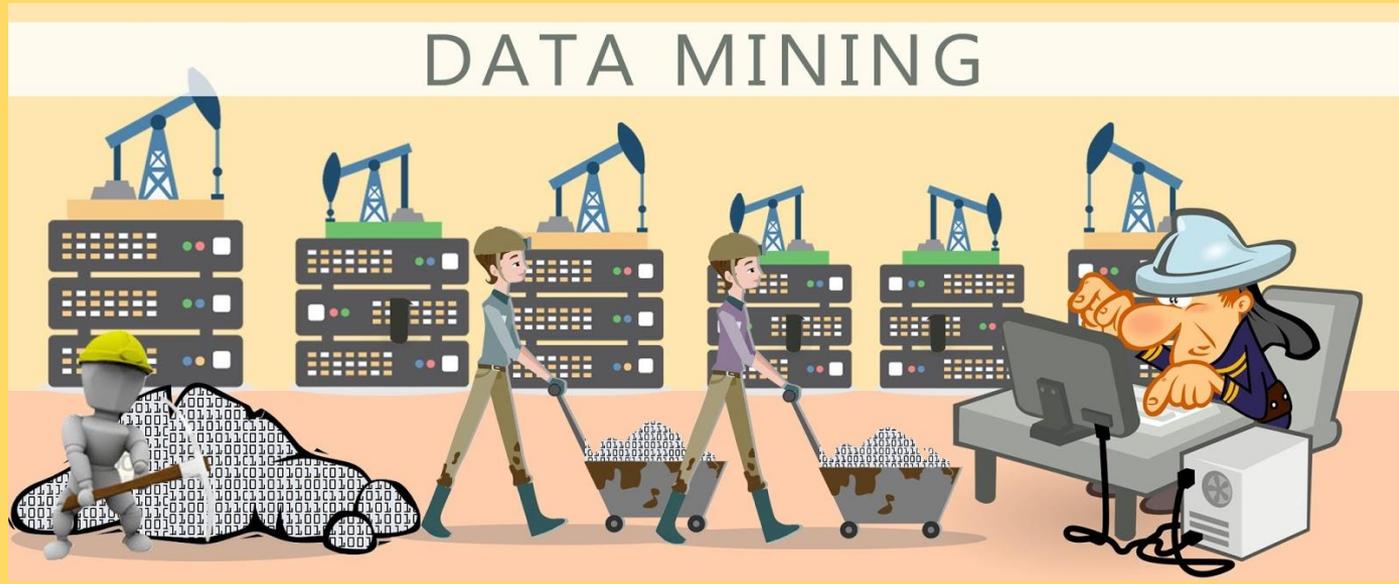


- **ROOT is the Data Analysis framework used extensively at CERN.**
- **It provides a rich set of functionality for big data processing, statistical analysis, visualization and storage.**
- **Had never dealt with ROOT or C++ before, learned through tutorials and primers.**



Jupyter Notebooks and SWAN

- Literate programming, human-friendly
- Efficient for **research, demonstrations, and teaching**
- An isolated and lightweight “sandbox”



The Problem

- A major component of Jupyter notebooks are graphs, histograms etc.
- These are contained within a TCanvas.
- For the Jupyter notebooks, ROOT objects are represented and serialized as JSON.
- Redundant information creates **bloated notebooks**.

```
{
  "typename": "TCanvas",
  "fUniqueID": 0,
  "fBits": 36896776,
  "fLineColor": 1,
  "fLineStyle": 1,
  "fLineWidth": 1,
  "fFillColor": 0,
  "fFillStyle": 1001,
  "fLeftMargin": 0.1,
  "fRightMargin": 0.1,
  "fBottomMargin": 0.1,
  "fTopMargin": 0.1,
  "fXfile": 2,
  "fYfile": 2,
  "fAfile": 1,
  "fXstat": 0.99,
  "fYstat": 0.99,
  "fAstat": 2,
  "fFrameFillColor": 0,
  "fFrameLineColor": 1,
  "fFrameFillStyle": 1001,
  "fFrameLineStyle": 1,
  "fFrameLineWidth": 1,
  "fFrameBorderSize": 1,
  "fFrameBorderMode": 0,
  "fX1": -1.25000010058284,
  "fY1": -7.00912558927577,
  "fX2": 12.2500001005828,
  "fY2": 72.0821255892758,
  "fXtoAbsPixelk": 64.4444986697517,
  "fXtoPixelk": 64.4444986697517,
  "fXtoPixel": 51.5555547873179,
  "fYtoAbsPixelk": 430.171058438474,
  "fYtoPixelk": 430.171058438474,
  "fYtoPixel": -5.96779027979266,
  "fUtoAbsPixelk": 5e-5,
  "fUtoPixelk": 5e-5,
  "fUtoPixel": 696,
  "fVtoAbsPixelk": 472.00005,
  "fVtoPixelk": 472,
  "fVtoPixel": -472,
  "fAbsPixeltoXk": -1.25000010058284,
  "fPixeltoXk": -1.25000010058284,
  "fPixeltoX": 0.0193965520131691,
  "fAbsPixeltoYk": 72.0821255892758,
  "fPixeltoYk": -7.00912558927577,
  "fPixeltoY": -0.16756621012405,
  "fXLowNDC": 0,
  "fYLowNDC": 0,
  "fXUpNDC": 0,
  "fYUpNDC": 0,
  "fWNDC": 1,
  "fHNDC": 1,
  "fAbsXLowNDC": 0,
  "fAbsYLowNDC": 0,
  "fAbsWNDC": 1,
  "fAbsHNDC": 1,
  "fUxmin": 0.1,
```

The Goal:

- Find the redundant information that is able to be trimmed out of the notebook, quantify the effects of this, in order to be able to **compress the size of the notebook**
- Identify candidates for **trimming**
- Prototype in python
 - Familiar language
 - Proof of concept
- If successful, implement in C++
 - Faster

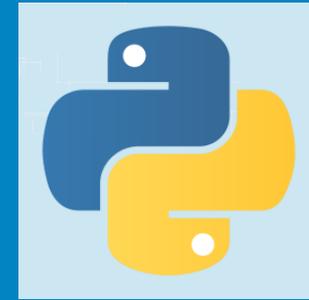


Main trimming candidate

- A main candidate suggested for large bloated notebooks was the List of TColors.
- TCanvas contains the entire list of TColors
- **649 colours** predefined by default palette at ROOT's initialization!
- However, this is in most cases storing this information is **completely redundant**.

```
    "_typename": "TObjArray",
    "name": "ListOfColors",
    "arr": [{
      "typename": "TColor",
      "fUniqueID": 0,
      "fBits": 50331648,
      "fName": "background",
      "fTitle": "",
      "fNumber": 0,
      "fRed": 1,
      "fGreen": 1,
      "fBlue": 1,
      "fHue": 0,
      "fLight": 1,
      "fSaturation": 0,
      "fAlpha": 1
    }, {
      "typename": "TColor",
      "fUniqueID": 0,
      "fBits": 50331648,
      "fName": "black",
      "fTitle": "",
      "fNumber": 1,
      "fRed": 0,
      "fGreen": 0,
      "fBlue": 0,
      "fHue": 0,
      "fLight": 0,
      "fSaturation": 0,
      "fAlpha": 1
    }, {
      "typename": "TColor",
      "fUniqueID": 0,
      "fBits": 50331648,
      "fName": "red",
      "fTitle": "",
      "fNumber": 2,
      "fRed": 1,
      "fGreen": 0,
      "fBlue": 0,
      "fHue": 360,
      "fLight": 0.5,
      "fSaturation": 1,
      "fAlpha": 1
    }, {
      "typename": "TColor",
      "fUniqueID": 0,
      "fBits": 50331648,
      "fName": "green",
      "fTitle": "",
      "fNumber": 3,
      "fRed": 0,
      "fGreen": 1,
```

Pythonic prototype



- Exploration of the serialized JSON hierarchy and structure of the TCanvas.
 - Discovered fixed nested hierarchy for List of TColors
- Introduced a new intercepting method `removeTColors()`
 - Perform checks and return original JSON if no TCanvas in notebook
 - Otherwise, deserialise the JSON
 - Traverse the hierarchy to find the List of TColors array
 - Delete each of the colors in the array
 - Reserialise the JSON
 - Pass the JSON back to be used in the notebook.
- Test the method using Python doctests
 - Usual white box testing:
 - Test if contained TCanvas
 - Test if doesn't contain TCanvas
 - Test if has TCanvas and List of TColors and vice-versa, etc.



implementation

Create a new method that is called from the Python side, analogous to the prototype mentioned previously

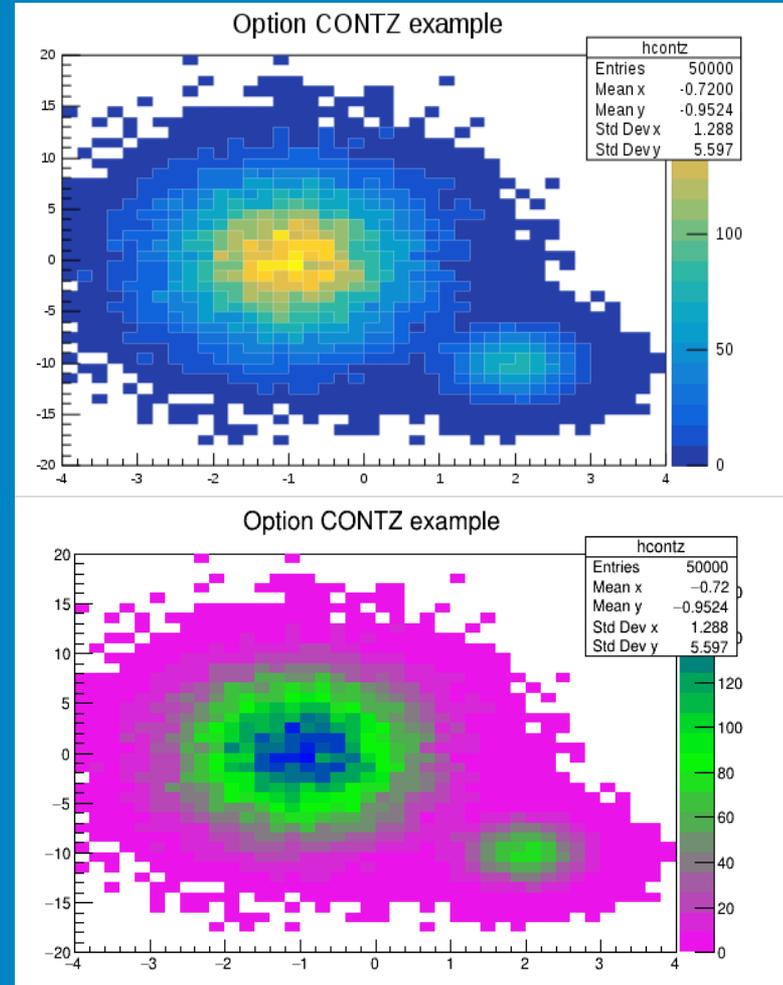
Options:

- Leverage TBufferJSON class, which is used to serialise data.
 - Create a new compression level passed to the 'compact' parameter of the method ConvertToJSON(...).
- Current levels of compression:
 - 0 – no compression (human readable)
 - 1 – exclude leading spaces
 - 2 – remove newlines
 - 3 – exclude all spaces where possible
 - New level: 4 – graphics compression for Jupyter Notebooks
- Initially, the first method was favoured, but later it was thought for maintainability and flexibility, it is better to leverage the 'compact' flag.



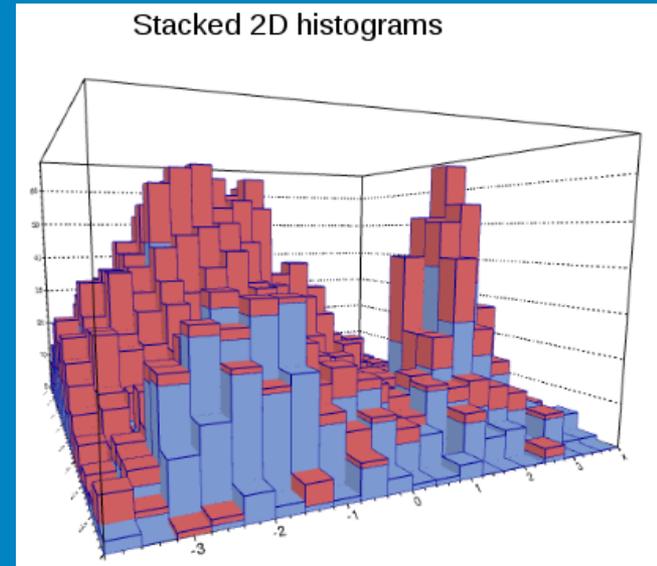
C++ implementation/Testing

- Testing
 - Manual Tests
 - Google Tests
- But:
 - Changing the palette:
`gStyle->SetPalette(...)`
did not work!
 - Actually, discovered a missing feature in JSROOT
 - Will now be addressed in due time by our team



C++ implementation/Testing

- Need to check if user wants to use a palette other than the default.
- As mentioned previously, nominally there are 649 colours that are predefined.
 - Changing palette will change this number by ~50 or more
 - Plotting some particular graphs (ie. Stacked histogram) will increase this by ~3
- Check if the definedColors have changed by more than a tolerance to decide to not trim.

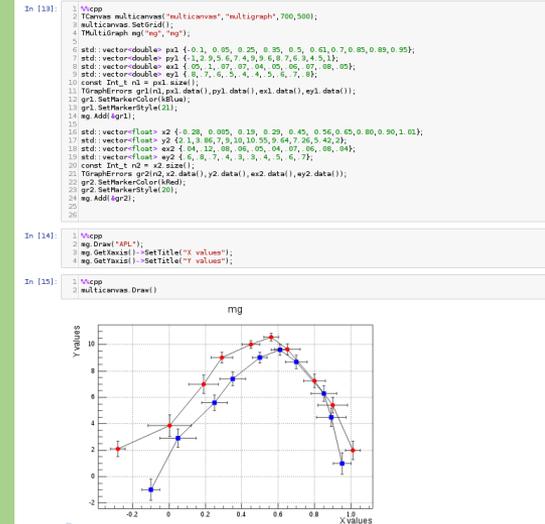


```
root [0] .x ListColors.C
```

Idx	Red	Green	Blue	Alpha	Color Name
0	1.000	1.000	1.000	1.000	background
1	0.000	0.000	0.000	1.000	black
2	1.000	0.000	0.000	1.000	red
3	0.000	1.000	0.000	1.000	green
4	0.000	0.000	1.000	1.000	blue
5	1.000	1.000	0.000	1.000	yellow
6	1.000	0.000	1.000	1.000	magenta
7	0.000	1.000	1.000	1.000	cyan
8	0.350	0.830	0.330	1.000	Color8
9	0.350	0.330	0.850	1.000	Color9
10	0.999	0.999	0.999	1.000	white
11	0.754	0.715	0.676	1.000	editcol
12	0.300	0.300	0.300	1.000	grey12
13	0.400	0.400	0.400	1.000	grey13
14	0.500	0.500	0.500	1.000	grey14
15	0.600	0.600	0.600	1.000	grey15
16	0.700	0.700	0.700	1.000	grey16

Evaluation and results

- The evaluation of how well the algorithm worked was done through creating a benchmark notebook of several different types of graphs such as 2-D and 3-D graphs and histograms.
- The **size** and **'correctness'** of the notebook and its output to the user were compared before and after the compression:

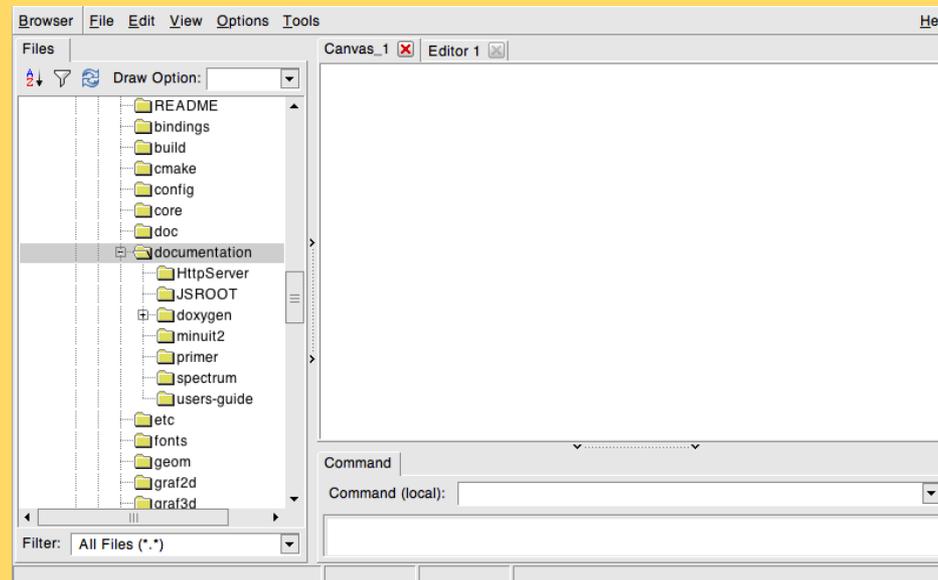


Notebook size before TColorCompression	Notebook size after TColorCompression	Total space saved:
1.2 MB	250 kB	1.05 MB (~80% reduction!)

Correctness: No difference in notebook's before and after in terms of displaying the graphics correctly.

Jupyter Notebook browser

- Currently there is a browser functionality in ROOT (TBrowser), that allows you to browse ROOT objects in local memory, such as graphs and histograms



- No ability previously to be able to browse the root objects in the Jupyter Notebook. The task is to allow for a Tbrowser-esque browser, callable from within a Jupyter notebook.

JSRoot browser

- No need to reinvent the wheel, and can leverage some of the online JSRootBrowser features that already exist.
- About JSROOT browser:
 - Launches a specialized THttp server to display the browser to the user.
 - Can manually set this up in ROOT
 - Allows users to browse local files on disk and see the graphs

Acknowledgements



A big thank you to my supervisor **Danilo Piparo** for all the guidance and mentorship, as well as others in the hallway!

And lastly a big thank you to the **EP-SFT** group for granting me this once in a lifetime opportunity to work for CERN, I am humbled and honoured by my selection!



I've enjoyed every second here
at CERN!



Thank you for listening!

