ROOT RDataFrame

RDataFrame documentation

- RDF is ROOT's high-level analysis interface.
- Users define their analysis as a sequence of operations to be performed on the data-frame object;
  - the framework takes care of the management of the loop over entries as well as low-level details such as I/O and parallelisation.
- RDataFrame provides methods to perform most common operations required by ROOT analyses:
  - at the same time, users can just as easily specify custom code that will be executed in the event loop.

HEP data analysis with RDataFrame

RDataFrame allows reading and writing trees, aiming at making HEP analysis easy to write and fast to perform.

In [1]:
```
import ROOT

treename = "dataset"
filename = "../../data/example_file.root"
df = ROOT.RDataFrame(treename, filename)

print(f"Columns in the dataset: {df.GetColumnNames()}")
```

```
Welcome to JupyROOT 6.30/04
Columns in the dataset: { "a", "b", "vec1", "vec2" }
```

Now we can `Define` new quantities, `Filter` rows based on custom expressions and retrieve some data aggregations such as a `Count` and a `Mean`:

In [2]:
```
def1 = df.Define("c", "a+b")

fil1 = def1.Filter("c < 0.5")

count = fil1.Count()
mean = fil1.Mean("c")
display = fil1.Display(["a","b","c"])

print(f"Number of rows after filter: {count.GetValue()}")
print(f"Mean of column c after filter: {mean.GetValue()}")
print("Dataset contents:")
display.Print()
```

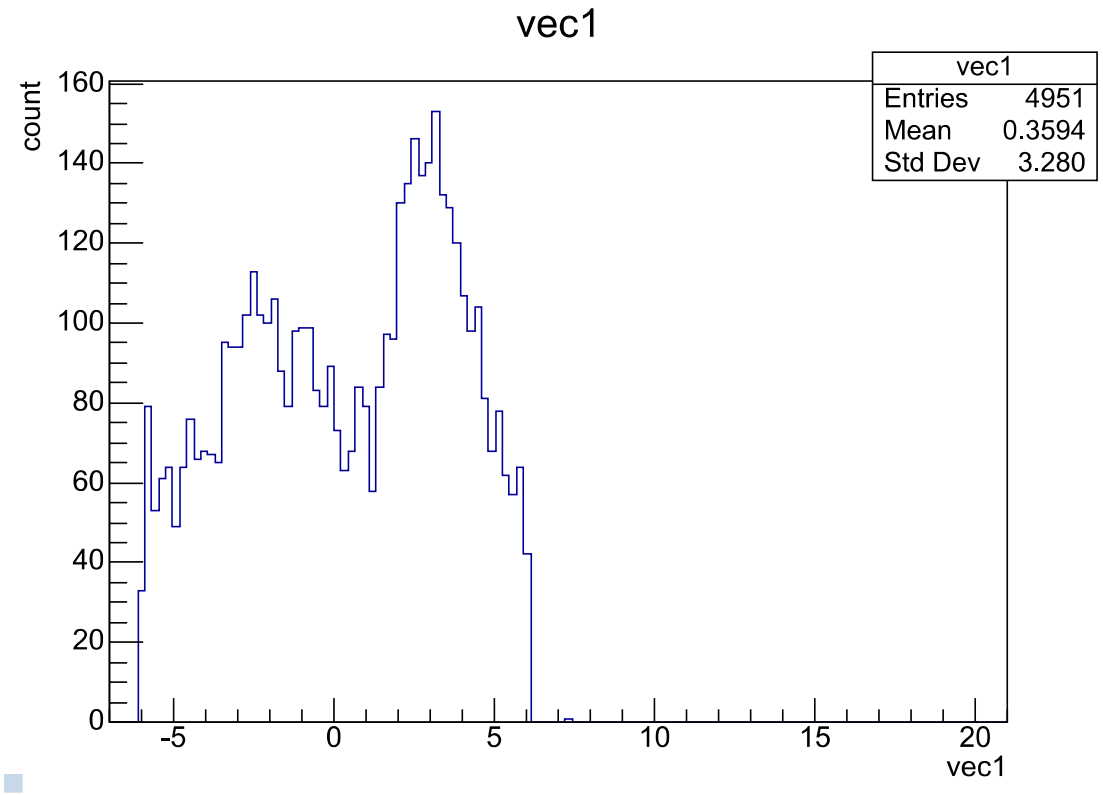
```
Number of rows after filter: 111
Mean of column c after filter: 0.38583512997804337
Dataset contents:
+-----+------------+-------------+------------+
| Row | a          | b           | c          |
+-----+------------+-------------+------------+
| 11  | 0.28843593 | 0.042303163 | 0.33073909 |
+-----+------------+-------------+------------+
| 13  | 0.25636993 | 0.12553929  | 0.38190921 |
+-----+------------+-------------+------------+
| 30  | 0.11045690 | 0.31782435  | 0.42828125 |
+-----+------------+-------------+------------+
| 92  | 0.26561222 | 0.17973985  | 0.44535206 |
+-----+------------+-------------+------------+
| 107 | 0.20609357 | 0.17557919  | 0.38167276 |
+-----+------------+-------------+------------+
```

Histograms with RDataFrame

RDataFrame helps you streamline the creation and filling of histogram objects from your data.

For example:

In [3]:
```
%jsroot on
c = ROOT.TCanvas()
h = df.Histo1D("vec1")
h.Draw()
c.Draw()
```



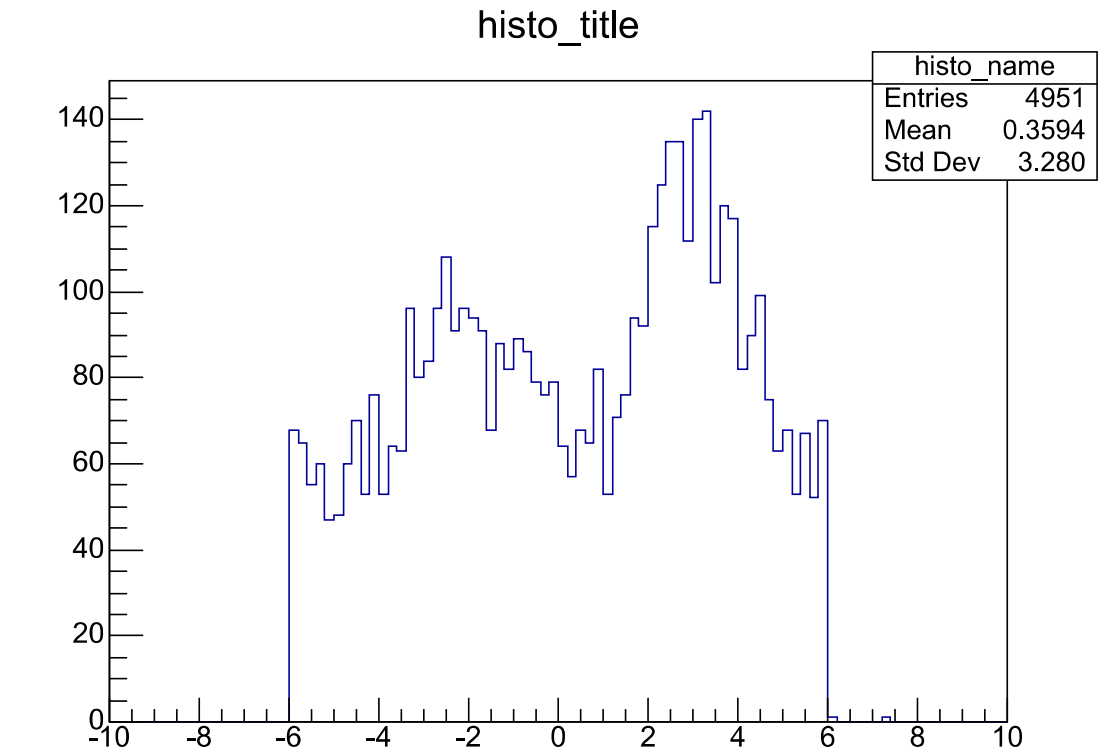
- `Histo1D` will create a one-dimensional histogram holding `double` values.
- `Histo{2,3}D` do the same in higher dimensions.
- These operations also accept a tuple with the same arguments that would be passed to the equivalent histogram object constructors.
- For example:

In [4]:
```
histo_name = "histo_name"
histo_title = "histo_title"
```

```
nbinsx = 100
xlow = -10
xup = 10

# The traditional TH1D constructor
# ROOT.TH1D(histo_name, histo_title, nbinsx, xlow, xup)

# With RDataFrame
c = ROOT.TCanvas()
h = df.Histo1D((histo_name, histo_title, nbinsx, xlow, xup), "vec1")
h.Draw()
c.Draw()
```



Think about data-flow

RDataFrame is built with a modular and flexible workflow in mind, summarised as follows:

- build a data-frame object by specifying your data-set
- apply a series of transformations to your data
  - filter (e.g. apply some cuts) or
  - define a new column (e.g. the result of an expensive computation on columns)
- apply actions to the transformed data to produce results (e.g. fill a histogram)

Important Note!

Make sure to **book all transformations and actions before** you access the contents of any of the results: this lets RDataFrame accumulate work and then produce all results at the same time, upon first access to any of them.

In [5]:
```
df_wrong = ROOT.RDataFrame(treename, filename)

h_a = df_wrong.Histo1D("a")
h_a_val = h_a.GetValue()

h_b = df_wrong.Histo1D("b")
h_b_val = h_b.GetValue()

h_vec1 = df_wrong.Histo1D("vec1")
h_vec1_val = h_vec1.GetValue()
```

```
print(f"The dataset was processed {df_wrong.GetNRuns()} times.")
```

```
The dataset was processed 3 times.
```

In [6]:
```
df_good = ROOT.RDataFrame(treename, filename)

h_a = df_good.Histo1D("a")
h_b = df_good.Histo1D("b")
h_vec1 = df_good.Histo1D("vec1")

h_a_val = h_a.GetValue()
h_b_val = h_b.GetValue()
h_vec1_val = h_vec1.GetValue()

print(f"The dataset was processed {df_good.GetNRuns()} time.")
```

```
The dataset was processed 1 time.
```

Operation categories in RDataFrame

There are 3 main types of operations you can perform on RDataFrames:

In [7]:
```
%%html
<style>
  th { font-size: 30px }
  td { font-size: 30px }
</style>
```

Transformations: manipulate the dataset, return a modified RDataFrame for further processing.

Transformation	Description
Alias()	Introduce an alias for a particular column name.
Define()	Creates a new column in the dataset.
Filter()	Filter rows based on user-defined conditions.

Actions: aggregate (parts of) the dataset into a result.

Action	Description
Count()	Return the number of events processed.

Action	Description
Display()	Provides a printable object representing the dataset contents.
Graph()	Fills a TGraph with the two columns provided.
Histo1D(), Histo2D(), Histo3D()	Fill a one-, two-, three-dimensional histogram with the processed column values.
Max(), Min()	Return the maximum(minimum) of processed column values.
Snapshot()	Writes processed data-set to a new TTree.

| ... | ...

Queries: these methods query information about your dataset and the RDataFrame status.

Operation	Description
GetColumnNames()	Get the names of all the available columns of the dataset.
GetColumnType()	Return the type of a given column as a string.
SaveGraph()	Export the computation graph of an RDataFrame in graphviz format for easy inspection.

Operation	Description
...	...