RooFit Development - Pythonization ideas

The deliverables of pythonization project group in the following areas outlined below. Please let us know if you have any further ideas or suggestions!

1. Pythonizations of RooFit functions that take command arguments

Pythonizations of RooFit functions that take command arguments, as tracked in this issue (https://github.com/root-project/root/issues/7217).
Primary goal is to pythonize all functions that accept RooCmdArgs, such that they take keyword arguments instead.
We can find the these function patterns for example with `git grep "RooCmdArg::none()"`.

The code will become more beautiful on the python side, for example

```
pdf.fitTo(data, ROOT.RooFit.Range("sideband"))
```

becomes

```
pdf.fitTo(data, range="sideband")
```

We decided to capitalize the keyword arguemnts to be consistent with the doxygen documentation.

Special case: nested command arguments

The idea of replacing the RooCmdArg functions with keyword arguments runs into problems when the functions that create a RooCmdArg accept a RooCmdArg itself. Here is an example from the rf302_utilfuncs.py tutorial (https://github.com/root-project/root/blob/master/tutorials/roofit/rf302_utilfuncs.py):

```
model.createHistogram("hist", x,
                      ROOT.RooFit.Binning(50),
                      ROOT.RooFit.YVar(y, ROOT.RooFit.Binning(50)))
```

The naive solution does not work, becuase we can't use keyword arguments in tuples:

```
model.createHistogram("hist", x, Binning=50, YVar=(y, Binning=50))
```

One possible pythonic interface would be to use dictionaries (...but it's problematic in python 2 where dictionaries are not ordered):

```
model.createHistogram("hist", x, Binning=50, YVar=dict(var=y, Binning=50))
```

2. Simple pythonizations to make RooFit in pyROOT less verbose and more pythonic

Simple pythonizations to make RooFit in pyROOT less verbose and more pythonic that we can identify by looking at the RooFit tutorials (https://github.com/root-project/root/tree/master/tutorials/roofit) and find reoccuring non-pythonic patterns.

One pattern to pythonize are for example colors:

```
model.plotOn(xframe, ROOT.RooFit.LineColor(ROOT.kMagenta))
```

We can use the well known color conventions from matplotlib (https://matplotlib.org/stable/gallery/color/named_colors.html) to simplify this:

```
model.plotOn(xframe, ROOT.RooFit.LineColor("m"))
```

However, this "pythonization" can also be done in the C++ code, so also C++ users can profit from this familiar notation.

Another good indicator for patters that can be pythonized is the usage of enums:

```
ROOT.RooDecay("decay_gm", "decay", dt, tau, gm, ROOT.RooDecay.DoubleSided)
```

Keyword arguments with string values would be more pythonic here:

```
ROOT.RooDecay("decay_gm", "decay", dt, tau, gm, DecayType="DoubleSided")
```

In general, the **usage of C++ enums** seems to be a good indicator that a pythonization is due. We hope to find more such indicators throughout the project.

3. Correct handling of object ownership in Python

- This is one of the largest issues in pyROOT. See RooFit/Stats mattermost for ongoing discussion.

4. More complicated Pythonizations for use cases that are challenging to implement without expert C++ knowledge

See for example this forum post (https://root-forum.cern.ch/t/combining-roodatasets-in-pyroot/43615). Here, the problem is that one has to create a C++ ``std::unordered_mapin `python to pass to theImport` command argument function:

```
dsmap = ROOT.std.map('string, RooDataSet*')()

# this is necessary to keep the RooDataSets alive,
# because the dsmap only contains pointers.
dsmap.keepalive = list()

# from python dictionary to std::unordered_map
for c, d in dsdict.items():
    dsmap.keepalive.append(d)
    dsmap[c] = d

ds = ROOT.RooDataSet("data","data", ROOT.RooArgSet(x), Index=cat, Import=dsmap)
```

It should be possible to pass python dictionaries directly to `Import`:

```
ds = ROOT.RooDataSet("data","data", ROOT.RooArgSet(x), Index=cat,
Import=dsdict)
```

The same pattern can also occur in the creation of a RooDataHist, as demonstrated in this tutorial (https://github.com/root-project/root/blob/master/tutorials/roofit/rf401_importttreethx.py#L82):

In general, things get complicated when one is forced to use C++ STL classes from Python. So for example in the case where a `std::unordered_map` is expected, it would be good to have a pythonization that accepts a Python dictionary (both `std::unordered_map` and the Python dictionary are hash tables).

Other more complicated opportunities for Pythonization are functions that take a RooArgList or a RooArgSet. These should be Pythonized to accept a simple Python list (or maybe iterable in general).

For example, this code:

```
fy_1 = ROOT.RooFormulaVar("fy_1", "a0-a1*sqrt(10*abs(y))",
                          ROOT.RooArgList(y, a0, a1))
```

might become

```
fy_1 = ROOT.RooFormulaVar("fy_1", "a0-a1*sqrt(10*abs(y))",[y, a0, a1])
```

5. Work on documentation

- We have to find a way to add doxygen code at function level for the pythonizations, not only as class level.
- We also need to think about what to do with the Python docstrings.