

Power BI for Physics

Edmund Ting

August 7, 2025

Abstract

Documentation for the [Power BI for Physics](#) project. Presents a brief physics preliminary before showcasing each of the Power BI projects in the repository, with descriptions of how its suite of tools (M, DAX, etc.) were used to manipulate and produce visualisations of the data.

Contents

1 Preliminaries	1
1.1 Why Power BI?	2
1.2 Basic workflow	2
1.3 Preparing physics data for analysis in Power BI	2
2 Example: Plotting the Higgs peak	2
2.1 Applying filters using Power Query M	3
2.2 Using Python for visualisation	4
3 Example: Interactive data filtering using slicers	5
3.1 Creating binned data counts using DAX	5
3.2 Plotting “histograms”	6
3.3 Adding interactive slicer elements	7

1 Preliminaries

The Large Hadron Collider (LHC) at CERN facilitates the collision of two antiparallel proton beams at extremely high energies. As the protons collide, they can undergo interactions governed by our most complete subatomic theory to date: the Standard Model of particle physics, which describes the fundamental particles and forces in our universe.

The driving motivation behind the physics programme of the LHC is precisely to gain a deeper understanding of the Standard Model. Being built upon the foundation of quantum field theory, the Standard Model is inherently non-deterministic. In fact, there is only a *tiny* probability for something to happen when two protons are made to collide; furthermore, the nature of exactly *what* happens is also probabilistic, as there can be a plethora of interactions that are possible. Thus, statistics lies at the core of experimental high-energy physics (HEP), and the work of the experimentalist is to sift through extremely large volumes of collision data in order to draw meaningful insights about the Standard Model.

The examples in this *Power BI for Physics* project are based on [this educational resource](#) by the ATLAS collaboration. The premise is to analyse publicly available LHC data to rediscover the [Higgs boson](#), recreating its momentous discovery by the ATLAS and CMS collaborations in 2012.

1.1 Why Power BI?

Power BI (*Business Intelligence*) is Microsoft’s analytics platform (only available on Windows) optimised towards creating interactive and digestible reports from data in the business context. Still, at its core, it is a tool used for manipulating and visualising data, and so it is also possible to conduct a physics analysis using it. As we shall see in the examples presented later, despite not being a practical tool for analysing large-scale HEP datasets, there are actually some nice features built into the platform particularly when it comes to creating interactive visualisation elements.

1.2 Basic workflow

Using Power BI essentially boils down to a few simple steps:

1. Import the data (from e.g. a local file, SQL server, or data warehouse).
2. Construct a *semantic model* (i.e. one or more connected tables) from the data.
3. Arrange visualisations (histograms, charts, etc.) together to create a report.

There are a few things to keep in consideration for each of these steps. Firstly, the size of the `.pbix` project file will scale with the amount of data imported in step 1. Thus, it’s important to trim away as much of the irrelevant data as possible during this step. **Power Query M** is the premier tool for this: it allows the user to filter and transform the input data prior to loading it into Power BI.

Semantic models represent a structured view of the imported dataset. The idea is that since the data could have been fetched from multiple sources, there can be commonalities or redundancies between them. For example, two tables could contain the same column, which would establish a relationship between their other columns. The interface for doing this is simple (one just drags and drops arrows between tables in a graphical view). Power BI also allows the user to calculate new variables at this step using **DAX** (*data analysis expressions*).

Finally, **build visualisations** of the data. Click different buttons, arrange till pretty.

1.3 Preparing physics data for analysis in Power BI

Due to the complex structure of LHC collision events, HEP data is typically stored in the **ROOT** format. Unfortunately, this is incompatible with Power BI, so relevant information needs to first be extracted from the files and exported into another format.

To parse a ROOT file, the **uproot** package in Python can be used. This allows for branches in the ROOT file to be extracted as **Awkward** arrays, which can then be exported into, for example, a `csv` or **Apache Parquet** file. (Note: it is not recommended to export a `csv` if processing a large number of events, as the write rate is *extremely* slow.)

2 Example: Plotting the Higgs peak

As the first example, let’s start with the very basics of a typical analysis in HEP. The problem statement is this: we have data that is distributed across a multi-dimensional space, and the objective is to identify a region in this “phase space” where contributions from a “signal” process is statistically significant with respect to other “background” processes. In this case, the signal corresponds to the production of a Higgs boson that then decays into two photons, while backgrounds consist of other Standard Model processes that also produce photons.

As preparatory work, for all photons in each event, the following variables were exported into a Parquet file: the transverse momentum (p_T), pseudorapidity (η), azimuthal angle (ϕ), and Boolean flags indicating whether they pass identification and isolation requirements.

2.1 Applying filters using Power Query M

Importing the Parquet file into Power BI, we obtain the following table:

	photon_pt	photon_eta	photon_phi	photon_e	photon_isTightID	photon_ptcone20
1	List	List	List	List	List	List
2	List	List	List	List	List	List
3	List	List	List	List	List	List
4	List	List	List	List	List	List
5	List	List	List	List	List	List
6	List	List	List	List	List	List
7	List	List	List	List	List	List
8	List	List	List	List	List	List
9	List	List	List	List	List	List
10	List	List	List	List	List	List
11	List	List	List	List	List	List
12	List	List	List	List	List	List
13	List	List	List	List	List	List
14	List	List	List	List	List	List
15	List	List	List	List	List	List
16	List	List	List	List	List	List

Because each event can contain multiple photons, Power Query displays each entry in the table as a **List** object. These are sorted in decreasing order according to the photon p_T , i.e. the first element in each list corresponds to the highest p_T photon, the second element to the p_T -sub-leading photon, and so on.

The Power Query M formula language can be used to filter the source data. The idea is that we start with the full table from the source, select only a subset of its rows based on some condition, and define this as a new table. This procedure is repeated successively until all the desired filters are applied, and the final data table is then passed to Power BI for analysis.

The basic structure of M's syntax looks like this:

```

1 let
2 Source = ***point to the input source data***
3 #"New table" = Table.SelectRows(Source, ***filter condition***)
4 #"Second new table" = Table.SelectRows("#New table", ***another filter***)
5 in
6 #"Second new table"
```

Importantly, note that the hash symbol is not a comment, but rather how the table variables are defined. The line after `in` determines the table that is imported into Power BI.

For this example, we want to define one new variable (i.e. add a column to the source table) and apply six filter requirements:

- at least two photons in the event
- both of the two leading photons must pass the identification criteria
- the leading photon must have $p_T > 50 \text{ GeV}$, and the subleading photon $p_T > 30 \text{ GeV}$
- both of the two leading photons must pass an isolation check: $p_T^{\text{cone},20}/p_T < 0.055$
- both of the two leading photons must satisfy $|\eta| < 1.37$ or $|\eta| > 1.52$
- the invariant mass of the two leading photons, calculated as

$$m_{\gamma\gamma} = \sqrt{2p_{T,1}p_{T,2}(\cosh(\eta_1 - \eta_2) - \cos(\phi_1 - \phi_2))}$$

must be less than their respective $p_T/0.35$

The M code to calculate $m_{\gamma\gamma}$ and implement these selections is as follows:


```

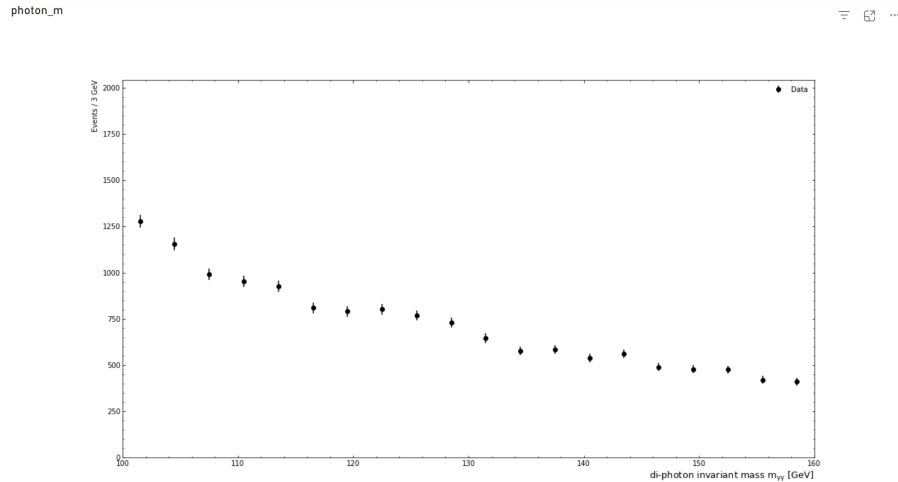
1  let
2  Source = Parquet.Document(File.Contents("photons.parquet"), [Compression=null,
    LegacyColumnNameEncoding=false, MaxDepth=null]),
3  #"Select two photons" = Table.SelectRows(Source, each (List.Count([photon_pt])
    >= 2)),
4  #"Select Tight ID" = Table.SelectRows("#Select two photons", each
    ([photon_isTightID]{0} = true and [photon_isTightID]{1} = true)),
5  #"Select minimum pT" = Table.SelectRows("#Select Tight ID", each ([photon_pt]{0}
    > 50 and [photon_pt]{1} > 30)),
6  #"Select isolated" = Table.SelectRows("#Select minimum pT", each
    ([photon_ptcone20]{0}/[photon_pt]{0} < 0.055 and
    [photon_ptcone20]{1}/[photon_pt]{1} < 0.055)),
7  #"Select eta" = Table.SelectRows("#Select isolated", each
    ((Number.Abs([photon_eta]{0}) < 1.37 or Number.Abs([photon_eta]{0}) > 1.52)
    and (Number.Abs([photon_eta]{1}) < 1.37 or Number.Abs([photon_eta]{1}) >
    1.52))),
8  #"Define invariant mass" = Table.AddColumn("#Select eta", "photon_m", each
    Number.Sqrt(2*[photon_pt]{0}*[photon_pt]{1} * (Number.Cosh([photon_eta]{0} -
    [photon_eta]{1}) - Number.Cos([photon_phi]{0} - [photon_phi]{1}))),
9  #"Select invariant mass isolation" = Table.SelectRows("#Define invariant mass",
    each ([photon_pt]{0}/[photon_m] > 0.35 and [photon_pt]{1}/[photon_m] >
    0.35))
10 in
11 #"Select invariant mass isolation"

```

2.2 Using Python for visualisation

For this example, all of the necessary filters are applied in Power Query, and the semantic model is simple, since there is only one table (corresponding to the di-photon invariant mass in each event).

We'll visualise the data using Power BI's integrated Python scripting functionality (the icon for this looks like ). This is a flexible option, since it allows one to leverage Python's large suite of plotting tools. In this instance, we plot the following histogram of the di-photon invariant mass: the excess of events around 125 GeV is the Higgs peak that we set out to recreate.



3 Example: Interactive data filtering using slicers

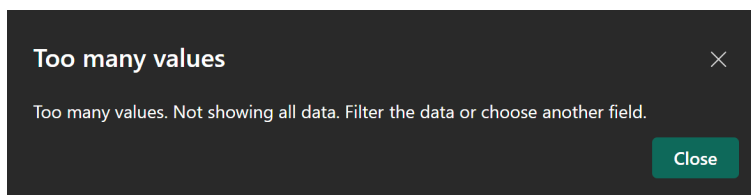
In the previous example, the goal was to familiarise ourselves with Power BI's data manipulation and visualisation capabilities. Using those basics, let's now lean into the report-building side of the application and construct an interactive dashboard that provides meaningful insights into the data.

In order to do this, data is imported into Power BI without applying any of the filters described in section 2.1. In addition, all of the input variables need to be imported (not just the invariant mass). Note that List objects cannot be imported into Power BI, so individual columns must be added to the table using Power Query like so:

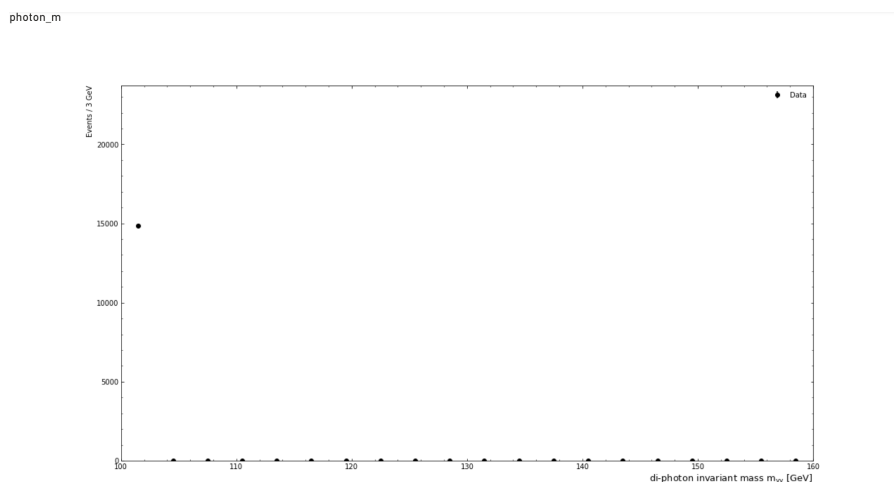
```
1 "#New column" = Table.AddColumn(Source, "photon_pt_0", each [photon_pt]{0})
```

Saving additional data will clearly have an impact on the resulting file size, and it turns out to be quite drastic: the final .pbix file footprint increases from 900 KB to around 750 MB when properties of the first two photons in each event is imported into Power BI.

A challenge rears its head as soon as we try to visualise this larger volume of data, for example with the same Python histogramming script used in section 2.2. Power BI shows this message:



and produces a histogram that looks like this:



There is a silly reason for this: it turns out that *every* entry in the histogram is counted as a visualisation element. So, for instance, here we see that the first bin contains $\sim 15\,000$ events. Power BI then thinks that we've created a visualisation with 15 000 data points in it, deems this as too crowded, and terminates the plot without throwing any errors. All subsequent bins are left unfilled, and there's no way to remove this limitation, because how on Earth would the user know better than Power BI, right?

3.1 Creating binned data counts using DAX

As a result of Power BI's very smart decisions, we'll need to take a few additional steps to recreate the output we want. I guess the upside of all this is that it gives us a prime opportunity to play

around with DAX. In particular, we want to achieve the following: given a range of values for $m_{\gamma\gamma}$ (corresponding to the bin ranges of the histogram we wish to plot), we wish to count the number of events (i.e. rows in our data table) that fall within this range, then use this singular value to represent a bin in the histogram-like visualisation.

Unfortunately, this gets messy for a couple of reasons. Firstly, DAX does *not* support any form of `for` loop, which means that a *lot* of code duplication will be unavoidable. Secondly, Power BI does *not* have a native histogram visualisation element. That’s right. The user has to do a bunch of hacky things just to draw a simple histogram. That’s why I said that we were making a “histogram-like” visualisation, because it’d actually be a column/scatter/etc. chart in disguise.

Anyway, the first step in all this is to create a new table containing the central bin values. This is done by clicking the “New Table” icon, then using the following DAX expression:

```
1 myy_bins = { 101.5, 104.5, 107.5, 110.5, 113.5, 116.5, ... }
```

Each of the numbers here represent a row in the new table. After creating this table, also go ahead and rename its single column to “bins” in order to stay consistent with the code below. (*Side note:* to define multiple columns in a new table, simply enclose each row in parentheses, e.g. `(col1,col2,col3)`. However, there might be issues with Power BI being oh-so-smart and automatically converting floating point values into integers. I could not find a way around this.)

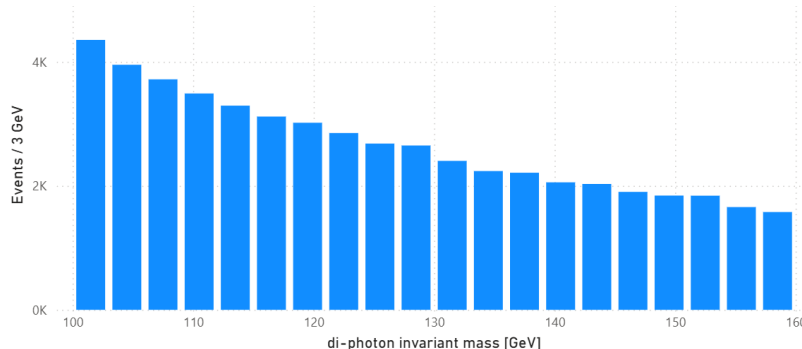
Next, we calculate the contents of each bin. This is done by selecting the main data table and creating a new measure, using the following DAX expression:

```
1 myy_binned = SWITCH( MAX('myy_bins'[bins]),
2   101.5, CALCULATE(COUNTROWS('photons'), ('photons'[photon_m] > 100) &&
3     ('photons'[photon_m] < 103)),
4   104.5, CALCULATE(COUNTROWS('photons'), ('photons'[photon_m] > 103) &&
5     ('photons'[photon_m] < 106)),
6   ...
7 )
```

Let’s unpack this. In line 1, we can see that this is a `SWITCH` statement, which is evaluating against the maximum of the bin values we just defined. In other words, if the bin value evaluates to (say) 101.5, then the expression on line 2 will be output of this switch case, and the return value is the number of events (i.e. rows) where the di-photon invariant mass is between 100 GeV and 103 GeV. The ellipsis indicates that the expression is completed by including all possible cases; namely, all of the possible bin values and their corresponding filtered counts.

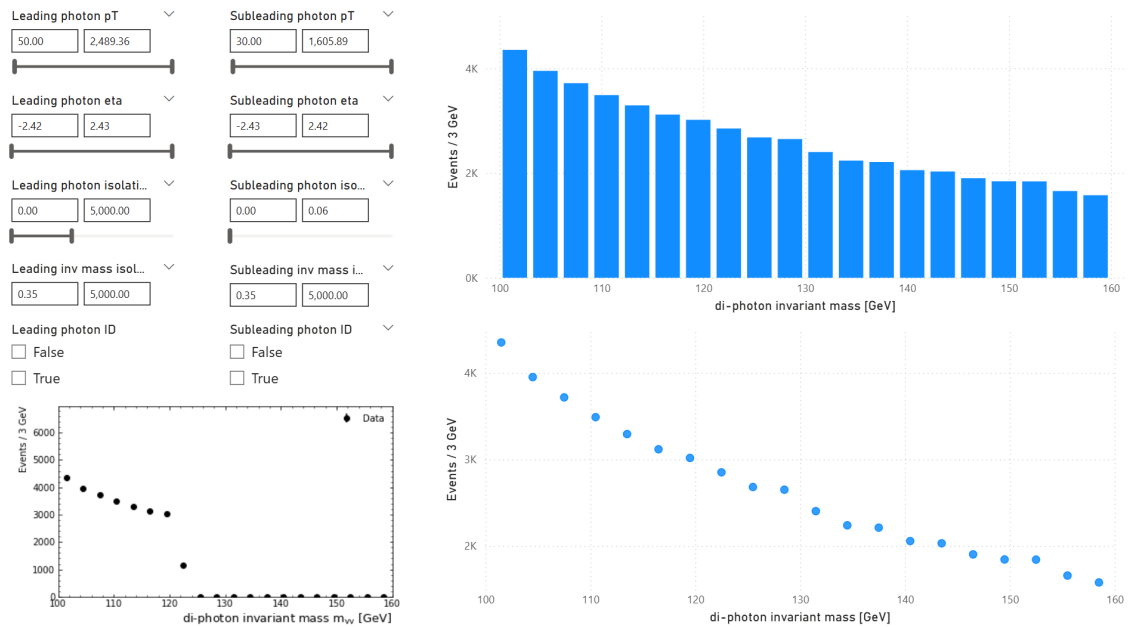
3.2 Plotting “histograms”

Visualisations can be produced using the bin values as the x -axis and calculated measure as the y -axis. Here is an example of how a stacked column chart looks like:



3.3 Adding interactive slicer elements

Slicers allow for filters to be applied dynamically to the visualised data. This is a pretty cool feature that allows the user to quickly study the impact of different selections on the overall data, which is useful for analysis prototyping.



Here you can see that I've added ways to adjust the kinematic selections that are imposed on the data (via entering upper/lower bounds or using the sliders). The Python approach to histogramming is plotted on the bottom left—where we can see that Power BI truncates past an invariant mass of 120 GeV or so—and on the right, two different visualisations of the aggregate approach is shown (a stacked column chart and a scatter chart).