Proton-Proton Collisions and Computational Analysis of CERN (CMS) Open Data

This project explores the “Primary dataset DoubleMu 2011A” from CERN’s Open Data Portal. As an educational dataset, certain regions have been enhanced to make the identification of meaningful signals—like Z boson decays—more approachable.

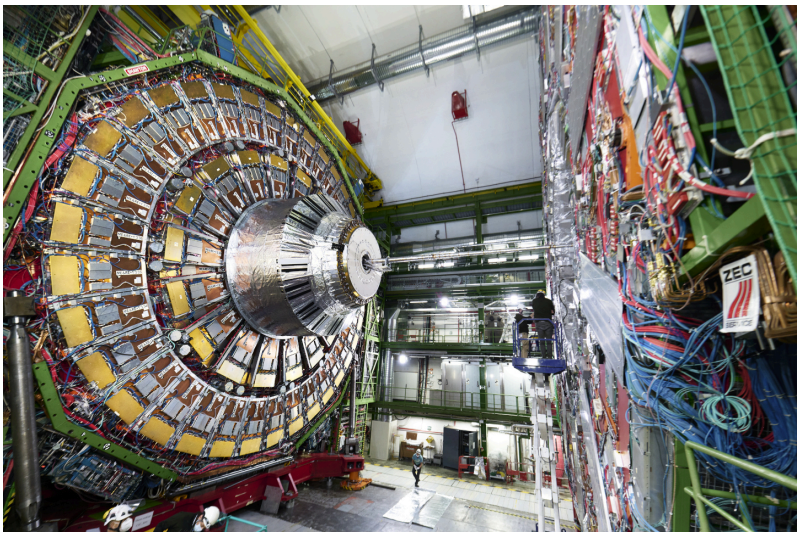
This analysis was carried out by a student in their final year of secondary school. While every effort was made to understand and apply the physics and data science correctly, it’s likely that mistakes, incorrect assumptions, or misinterpretations are present, and my understanding of physics concepts may be shallow and lacking. The project is meant as a learning experience—part physics, part programming, and a whole lot of curiosity.

Enjoy the journey, and don’t take the results too seriously :)

Link to Github repository:
https://github.com/notayan1505/computational-analysis-of-CERN-data/tree/main

1. Theory Behind CERN Open Data

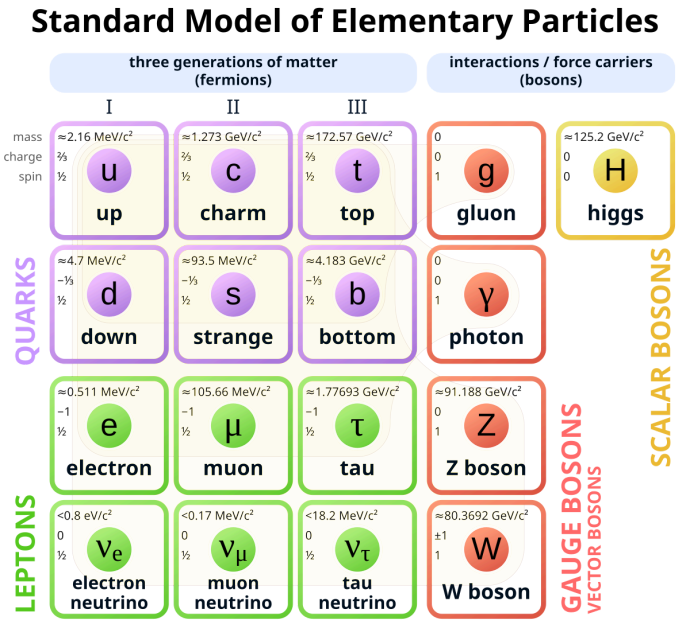
The CERN Compact Muon Solenoid - CMS - comes directly from their Large Hadron Collider - LHC, the world's largest and most powerful particle accelerator. While LHC deserves a whole essay for itself, today’s focus will be on what goes on inside the LHC. In particular, we will be looking at proton-proton collisions.



CMS CERN

The Standard Model of Physics

The standard model of physics is the most famous explanation behind the interaction of particles. Matter is made out of elementary particles, split into quarks and leptons. Both of these contain six particles each, split into pairs called generations, with the first generation being the most stable (and thus the most naturally appearing), and the later generations getting less stable.



[Standard Model (Wikipedia)]

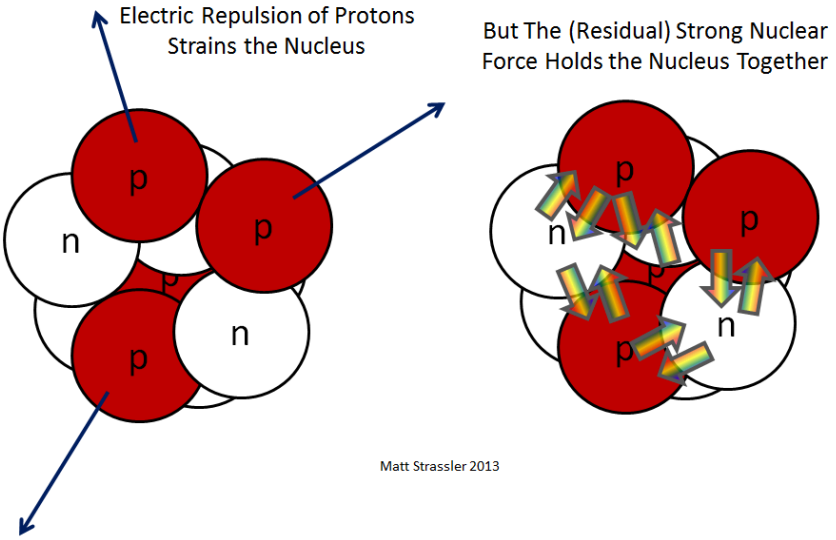
The Four Fundamental Forces

Four forces govern all types of interactions we currently know about, bar my social interactions. That one remains a mystery even quantum physics can't solve. These forces are gravity, electromagentism, the strong nuclear force and the weak nuclear force. At the microscopic level of particles, these forces play different roles. The electromagnetic and gravitational forces are quite well known, so here is an introduction to the other two:

- The strong nuclear force, as its name suggests, is the most powerful force at the atomic level. It is responsible for holding small subatomic particles together to form larger ones. It binds the protons and neutrons together to make the nucleus, and binds the quarks together using gluons (spoiler: they're the real MVPs inside protons) to form protons and neutrons.

  Imagine a nucleus. It has several positively charged protons. These protons repel each other, exerting a very strong repulsive force (that I cannot express in a metaphor because subatomic particles want to be all different), which is insane to think about at such a small scale. And only for two protons! Yet, this repulsion is overcome by magnitude by the strong nuclear force. And it has to be magnitudes, because it is extremely difficult to separate a proton from a nucleus despite electrostatic repulsion.

The strong force exists at an extremely short range - protons have to be very close to one another for it to have any effect. The force is mediated by gluons, a spin-1 boson - I will give a summary of this, don't worry - but how gluons do this is a bit above my level, closer to those crazy PhD people, so I will not be delving into that.



[Energy Education]

- The weak nuclear force is one of the four fundamental forces in nature, alongside gravity, electromagnetism, and the strong nuclear force. It is responsible for certain types of radioactive decay, such as beta decay, where a neutron transforms into a proton (or vice versa), emitting an electron (or positron) and a neutrino (or antineutrino), depending on the process.

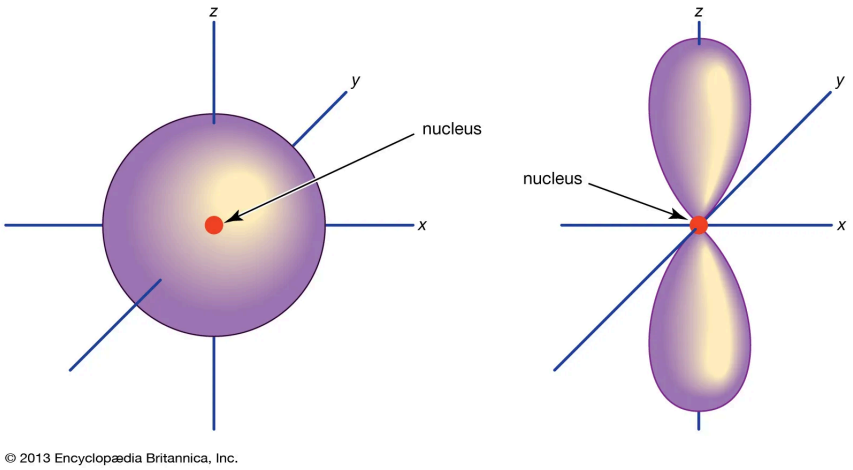
  This force acts on all fermions — a class of particles that includes quarks (which make up protons and neutrons) and leptons (like electrons and muons). The definition of a fermion is quite confusing, and I don't have the confidence to explain it, so here is a link for further reading.

  Every fermion carries a weak charge, which is mediated by three bosons: The W+, W- and Z boson. Unlike gluons, these bosons are enormous, and they really limit the range of the weak force, even considering the strong force.

  Compared to the strong force, the weak force is roughly a trillion times weaker. Lotta zeroes there. This is just a fun fact - the weak force is not in direct competition with the strong force. It is responsible for transforming particles like protons or neutrons, rather than pulling them apart against the strong force. The weak force is actually as strong as the electromagnetic force, it just acts at such a tiny range that it appears weak.

Basic Structure of an Atom and the Proton

Basics. The structure of an atom consists of the nucleus, made up of +1 charge protons and similarly weighted neutral neutrons. Around the nucleus exist electrons, much smaller and much lighter particles in comparison to the protons, with a -1 charge. Now, electrons are not neat round balls in circular orbits around the nucleus, like planets revolve around stars. Instead, they exist as wavefunctions, and a fuzzy cloud describes the probability of where an electron could be.



Electron Orbital - Britannica

Protons, apparently, also exist like this somewhat. While both of these particles have the wave-particle dual nature (so they exist as a wave and as a particle at the same time), all particles, including protons, exhibit wave-particle duality, which gives rise to probabilistic ‘clouds’ of their locations. This is because all matter exhibits uncertainty. However, this uncertainty is inversely proportional to mass, because of the de Broglie wavelength and Heisenberg’s uncertainty principle: heavier particles have smaller wavelengths and are more ‘localized.’What is Heisenberg’s Uncertainty Principle? Google it. And the mass of a proton is two thousand times greater than that of an electron. Even at that scale, the proton’s uncertainty is small enough for it to be considered a physical particle.

However, protons are not elementary particles. Elementary particles make up the base of reality, and are not made up of other, smaller particles. This distinction is key: particles we will discuss later - like bosons - are elementary particles, but they also decay - which might be confusing.

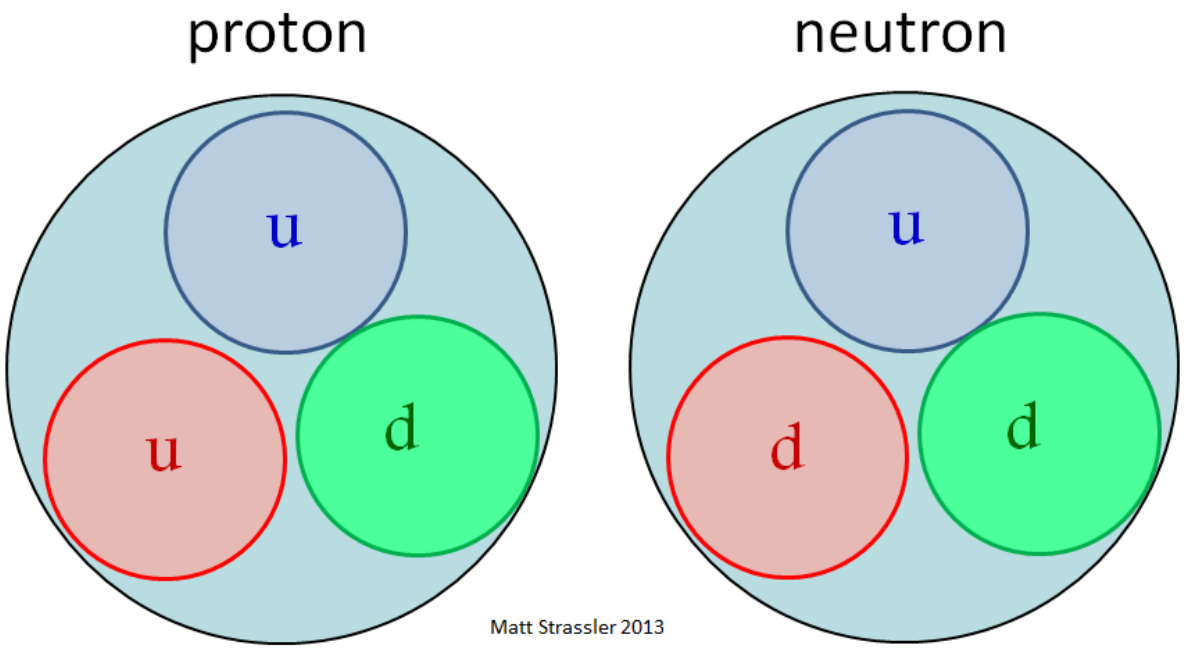
Protons themselves are made up of an elementary particle - quarks. In particular, two up quarks and one down quark. These quarks are held together by the strong force, which is enforced by particles called gluons. Also, protons contain a "sea" of quark-antiquark pairs and gluons that constantly pop in and out of existence. Trying to study this by yourself is equivalent to falling into a rabbit hole of madness. It is an experience.

Within protons, it is mostly empty space. The individual quarks and gluons inside are less than 1/10,000th the size of the entire proton.

What are Quarks, Bosons and Gluons?

Quarks are the very fundamental constituents of matter, which combine to produce other particles. There are six different types of quarks, called “flavors”: up, down, charm, strange, top, and bottom. The most important are the up and down quarks, as they are the first generation in the standard model, and the most abundant. Don’t worry too much about the rest. Quarks are never found alone, always in specific combinations that create hadrons. Protons and neutrons are a type of hadron. The “flavors” give a particle their electric charge. For example, in a proton:
There are two up quarks and one down quark. Charge of the up quark is +⅔. Charge of the down quark is -⅓. This sums up to +1, the charge of the proton.

Quarks have an important property named “color charges”. This is because quarks can have three unique charges: conveniently named after the three primary colors: red, green and blue. This is part of quantum chromodynamics (QCD), a theory of the strong force, one of the four fundamental forces of nature. When all the three colors are together, the overall color charge is neutral. Hadrons must be color neutral, which is why any drawing of a proton shows quarks of three different colors - it's not just to be colorful. When all three colors combine, the result is ‘color neutral’—a requirement for stable particles like protons.



https://profmattstrassler.com/articles-and-posts/particle-physics-basics/the-structure-of-matter/protons-and-neutrons/
But we cannot stop at just quarks. We also need to discuss their counterpart - antiquarks. Antiquarks are the antiparticles of quarks, which makes no sense if you don’t know what an antiparticle is. Simply, antiquarks are particles that have the same mass but opposite charge and other quantum numbers as quarks. For example, an up quark has a charge of +⅔ but an up antiquark has a charge of -⅔. They also have unique color channels: anti-red, anti-blue and anti-green.

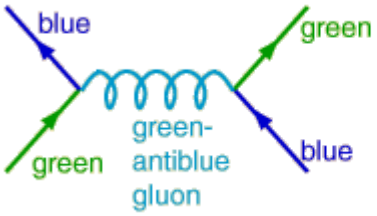
Antiquarks are important constituents of antimatter, and are also involved in the structure of the proton. However, they are not a stable element like quarks. Instead, at high energies

(like in the LHC), protons are seething with virtual particles: gluon radiation, quark-antiquark pairs, and quantum fluctuations. These fluctuations create a “quark sea” — which includes temporary antiquarks. So they just pop in and out of existence, aren’t very stable, and a part of some messy ‘quark-gluon sea’.

Next, let’s discuss bosons. If you’re ever confused if something is a boson, ask if it can carry one of the four fundamental forces. If it can, it's a boson. It is the Uber of forces. They are one of two broad categories of the Standard Model of Physics - the other being fermions.

Spin isn’t what you’re thinking. The particles themselves are not spinning like a ball. Some say it is more akin to rotation of waves (particles being the waves). But it's also related to angular momentum. Others say we don’t know and it is just some fancy math that ends up telling us something important in other calculations. I don’t know. I give up. I know it's not a scalar quantity, if that helps.

Two done. One to go. What are Gluons? Gluons are rather fascinating particles. A type of boson, they have no mass or charge, and are responsible for mediating the strong force - so they hold subatomic particles together. To understand how they do this, we return to color charge. Color charge must be conserved, the same way electric charge must be conserved. The strong force can be described as fluctuations in color charge of particles. Taking this into consideration, we can arrive at a conclusion: gluons must also have a color charge. Why? Because the color charge of quarks can change and the change in color charge causes the strong force. Therefore, the mediators of the strong force - gluons - must have a color charge so that it is conserved.



Gluon-mediated interaction between two quarks.

http://hyperphysics.phy-astr.gsu.edu/hbase/Forces/feyns.html
This is a Feynman Diagram, and helps deduce the color of the gluon. The x-axis represents time and the y-axis position. On the first quark on the left, we can see that a green quark enters the interaction, and leaves as a blue quark (following the position of arrows). To conserve charge, there must be green leaving the interaction as well. This gives the gluon a green color. However, the quark that leaves is blue, and there is no blue entering the interaction. Therefore, the gluon must also take on an anti-blue charge to cancel out the blue charge leaving. When we sum up all the colors, green enters and leaves the interaction, so color charge is conserved.

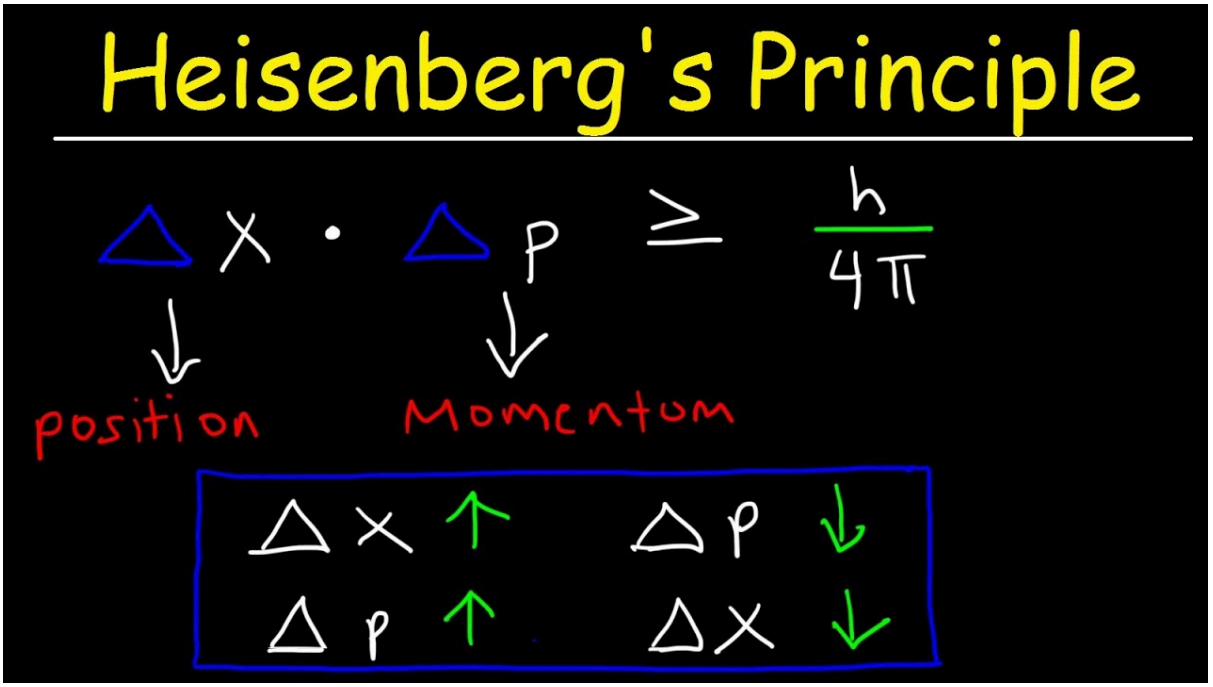
Gluons will always interact with particles with a color + anticolor pair of charge. So what's actually happening? Think of two quarks interacting via the strong force. One quark vibrates in a certain way, exciting the gluon field which then interacts with another quark, kind of linking them together. Gluons are representations of the field becoming excited, and that is why they are said to mediate the strong force.

Unstable Particles, Decay Width and Heisenberg's Uncertainty Principle

As the Standard Model describes, elementary particles are grouped into generations based on their mass and stability. In our proton-proton collision data, we observe muons—second-generation leptons that are heavier cousins of the electron. While muons decay in about 2.2 microseconds when at rest, at the energies inside the LHC they travel near the speed of light, experiencing significant time dilation according to Einstein's theory of special relativity. This allows them to travel far enough to be detected in particle detectors, even though they are unstable.

The particles we aim to study—such as the Z boson—are force carriers, not leptons, and they decay almost instantaneously (in ~10^{-25} seconds). Because of this, we cannot observe Z bosons directly, but instead infer their presence from the decay products—in this case, muon pairs.

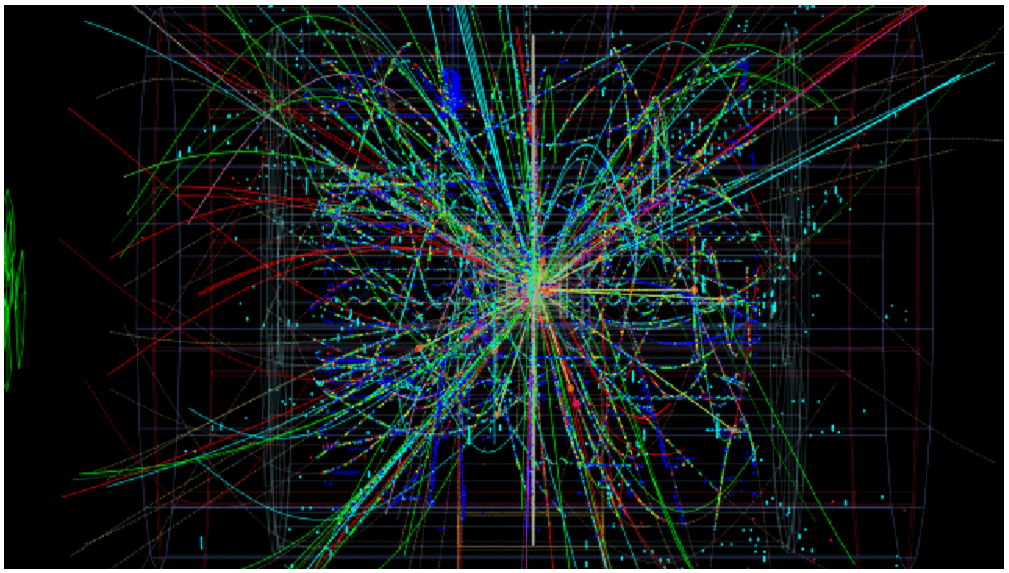
Due to their extremely short lifetimes, particles like the Z boson exhibit a property known as decay width. This is an intrinsic uncertainty in the particle's energy, and by extension, its mass, as $E=mc^2$. The less time a particle exists, the more uncertain its energy becomes—a direct consequence of the Heisenberg Uncertainty Principle, which states:



Credit: The Organic Chemistry Tutor - https://www.youtube.com/watch?v=BNYz5EKXVeI

The product of the uncertainty of the position of particles multiplied by the uncertainty in the mass of particles, must always be greater than Planck's Constant /4Pi. On the macroscopic scale, Planck's Constant is tiny: 6.626 x 10^-34 joule-seconds, so the uncertainty is pretty much negligible. On the scale of particles, however, it's not.

So how do proton-proton collisions occur?



[Proton Proton Collisions at CERN]

- The first step is to get a pure proton beam. The easiest way to do this is by using Hydrogen. Hydrogen has an atomic number of 1 and a mass number of 1. This means there are no neutrons in the nucleus of a hydrogen atom, only a proton. Through ionizing: $H \rightarrow H^+ + e^-$, you are left with a single proton. Take a lot of these to get a proton beam.

- These protons are then accelerated to near the speed of light in opposite directions around the circular tunnels of the LHC.

- Powerful magnets keep them in tightly focused beams, narrower than a human hair. These are superconducting magnets cooled to -271°C, just above absolute zero. They steer and squeeze the beams using electric and magnetic fields in perfect coordination.

- These beams are crossed at specific points—like at the center of the CMS detectors. These crossing points are engineered to maximize the chance of proton-proton collisions. Each crossing can involve billions of protons, but only a tiny fraction actually collide.

- Detectors record the debris from these collisions to study fundamental particles and forces, and we have our data.

Reasons For Speeding Up Protons

$E=mc^2$**.** Einstein's famous equation is the reason behind why the LHC speeds the protons to near light speed. It is to convert energy into mass.

One objective of the LHC is to study a vast array of different particles. Some of these particles have masses many many times those of the proton. At high speeds, the protons

have immense kinetic energy, which can then be converted to mass, to form heavier particles that we will be examining later. Most of these particles don't exist in our current conditions, and only stay for less than nanoseconds.

Another reason has to do with the equation $\lambda = h/p$. De Broglie's Wavelength. "h" is Planck's constant and "p" is momentum.

Everything has a De Broglie wavelength, as long as it is in motion (is a generic statement that has been hypothesised, and might not be true). From the equation, it is easy to see that momentum and wavelength are inversely proportional. A smaller wavelength allows LHC scientists to see even finer details.

The third reason is to recreate the conditions near the Big Bang, and to study the early universe. After the Big Bang, the universe was extremely hot, dense, and full of really hot particles. In fact, in the first few fractions of a second, the universe was so hot that quarks had so much energy that they couldn't even combine into protons.
By smashing protons at high speeds, CERN mimics those early conditions, helping physicists study how the universe began and evolved.

What Happens During and After These Collisions?

Collisions of protons are not exactly the same as colliding two round objects at a macroscopic scale. Protons are made up of empty space for the most part, and they do not have anything such as a defining surface or clear edges like a sphere does. You cannot collide empty space with itself. Instead, collisions means that the elementary particles within the proton pass close enough to each to interact.

At very high energies, these interactions can become strong enough to disturb the quantum fields that exist everywhere in space. When that happens, those fields can release energy by creating entirely new particles. So in particle physics, a collision is really about creating the right conditions for new physics to happen — not smashing particles like billiard balls.

If we are lucky, and the gluons and quarks and antiquarks carrying immense kinetic energy interact in a certain way that makes them excited, a successful collision achieve resonance - the creation of a temporary particle, as energy is turned into mass. Soon after, this particle will decay as it is highly unstable, leaving us with (in the scenario of our dataset) two muons, whose properties can be examined to find out the properties of the particle that was created. And that is what we will be doing.

Proton - proton collisions can create many, many different particles depending on the conditions. These resonances are rare, which is why CERN provides us with an educational dataset that amplifies the chances of these resonances for an easier time for us.

Finally, the rather long and frankly hard to get your head around introduction is done. That was an extremely basic overview of the theory, but there will be sources dotted around for further education if you so wish. Let's move on:

2. Understanding the Dataset “Events with two muons from 2011 (Primary dataset DoubleMu 2011A)”

This dataset contains information from 100,000 proton-proton collision events recorded at the LHC in 2011. Each event includes measurements of two muons that were detected after the collision. Not all of these events involve interesting new particles — in fact, most do not result in resonance, meaning no intermediate particle (like a Z boson) was formed and decayed into muons. However, in this educational dataset, the proportion of events showing clear resonance has been artificially increased. This makes it easier to observe and study phenomena like the production and decay of the Z boson.

Because of these modifications, the dataset is not suitable for professional research, but it’s excellent for learning and practicing data analysis techniques in particle physics — without the need to process terabytes of raw data.

The focus on muons is because they are:

- Elementary particles, belonging to the lepton family (like electrons, but heavier).
- Stable enough to reach the detector.
- Easily identifiable, making them ideal for studying the outcomes of high-energy collisions.

Sometimes, the muons are produced directly in the collision, but often they are the decay products of heavier particles, such as the Z boson. By analyzing the properties of these muons (like their momentum and angle), we can infer information about the original particle that created them.

These are the following quantities provided in the dataset, along with a brief description:

Dataset semantics

Variable	Type	Description
Run	The run number of the event.	
Event	The event number.	
type	Either T or G: whether the muon is a tracker or global muon.	
E	The total energy of the muon (GeV).	
px,py,pz	The components of the momemtum of the muon (GeV).	
pt	The transverse momentum of the muon (GeV).	
eta	The pseudorapidity of the muon.	
phi	The phi angle of the muon (rad).	
Q	The charge of the muon.	
M	The invariant mass of two muons (GeV).	

Credit: https://opendata.cern.ch/record/5201

Data was selected through the following process:
“An event was selected if there were two muons in the event, both with |eta| < 2.4, at least one muon was a global muon, the invariant mass of the two muons was > 0.3 GeV and < 300 GeV, and they have opposite-sign charge.”

3. Statistical Analysis

Firstly, from the data selection process, we can infer some information:

- The muons have opposite-sign charges. This means that the particle they originate from must have a sign charge of 0.
- An event must have one global muon. I will instead select only data from the dataset that has two global muons, and I will explain why later.
- The sum of the invariant masses of the two muons present us with a range for which to look for our particles. The invariant mass of the particle we are trying to find must equal to the sum of the invariant mass of the two muons.
  *Note: Invariant mass is mass of the particle that remains constant, regardless of the frame of reference. It is linked to Special Relativity.

Now that we have a basic idea of what we should do, let’s begin.

Setting Conditions for Unique Particles

Different particles will have unique invariant masses (I have specifically chosen a few particles so this is true). We must add labels to our data to seperate the muons that could have come from these particular particle decays, and those that haven’t. This is accomplished by analysing if the sum of the mass of the two muons is equivalent to the mass of the particle + some uncertainty.

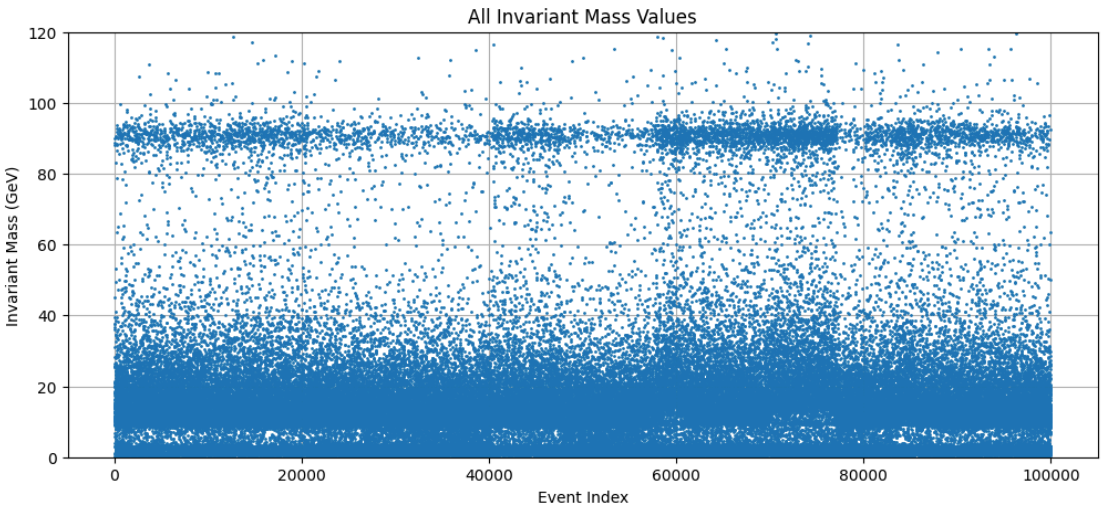
```python
# Nominal masses (GeV)
nominal_masses = {
    0: 3.10,    # J/ψ
    1: 3.685,   # ψ(2S)
    2: 9.45,    # ϒ(1S)
    3: 9.94,    # ϒ(2S)
    4: 10.265,  # ϒ(3S)
    5: 91.1876  # Z boson
}

# Relative uncertainties (fractional, e.g. 0.04 = 4%)
uncertainties = {
    0: 0.03,  # ~3% for J/ψ
    1: 0.03,  # ~3% for ψ(2S)
    2: 0.02,  # ~2% for ϒ(1S)
    3: 0.02,  # ~2% for ϒ(2S)
    4: 0.02,  # ~2% for ϒ(3S)
    5: 0.04   # 4% for Z boson (physics + detector combined)
}

def classify_mass(mass):
    for class_id, m_nom in nominal_masses.items():
        delta = uncertainties[class_id]
        lower = m_nom * (1 - delta)
        upper = m_nom * (1 + delta)
        if lower <= mass <= upper:
            return class_id
    return 6  # Background


# Apply classification to dataset
data['label'] = data['M'].apply(classify_mass)
```

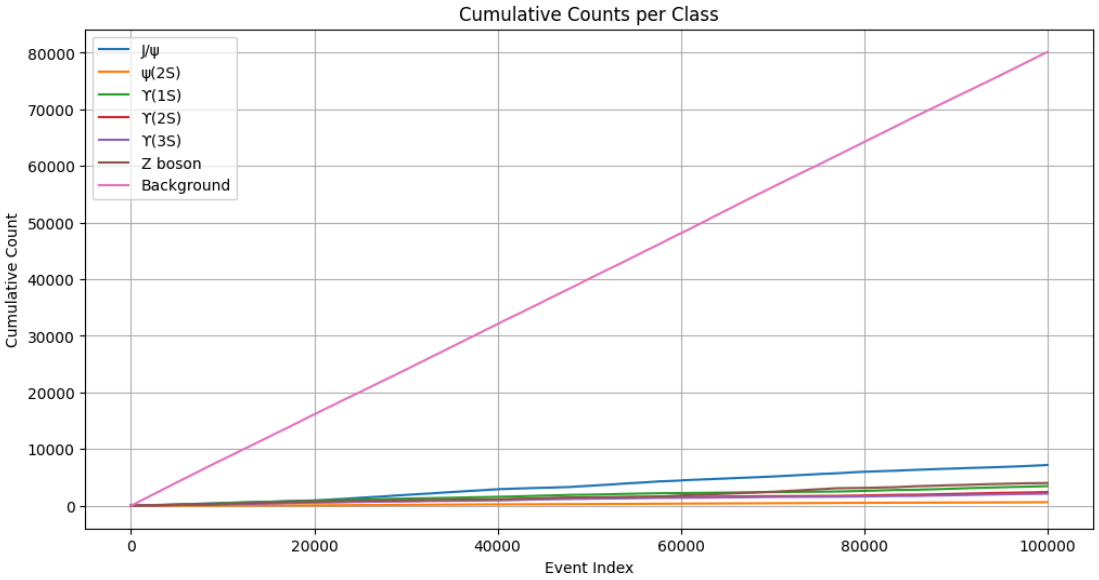
This means our dataset has been labelled - a new quantity was assigned to each event from 0 to 6, signifying where the muons could have originated from. I specifically use “could have” as we cannot be 100% sure until we do a few more things that help us clearly view the signal,

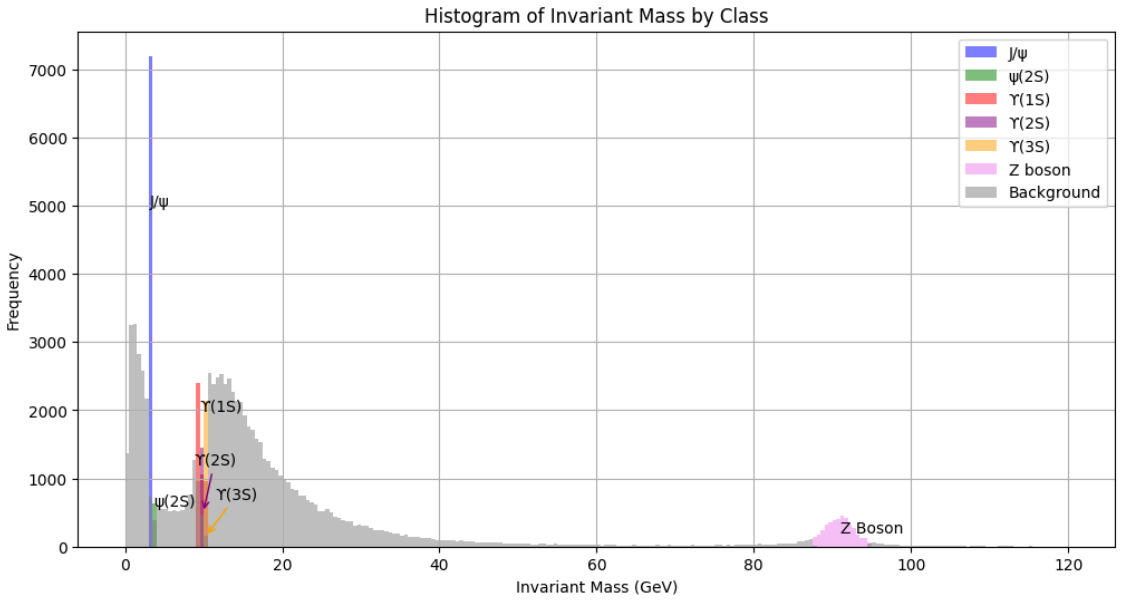
These are the particles I decided to look for:

- J/ψ(Psi) Meson of two unique energy states
- ϒ(Upsilon) Meson of three unique energy states
- Z Boson. This is the one I will be focusing on in particular.

Plot a few curves using matplotlib and we have an initial idea of what we are working with…


All Invariant Mass Values
Invariant Mass (GeV)
Event Index
0
20
40
60
80
100
120
0
20000
40000
60000
80000
100000


Cumulative Counts per Class
J/ψ
ψ(2S)
ϒ(1S)
ϒ(2S)
ϒ(3S)
Z boson
Background
Cumulative Count
Event Index
0
10000
20000
30000
40000
50000
60000
70000
80000
0
20000
40000
60000
80000
100000


Histogram of Invariant Mass by Class
J/ψ
ψ(2S)
ϒ(1S)
ϒ(2S)
ϒ(3S)
Z boson
Background
Frequency
Invariant Mass (GeV)
J/ψ
ϒ(1S)
ϒ(2S)
ψ(2S)
ϒ(3S)
Z Boson
0
1000
2000
3000
4000
5000
6000
7000
0
20
40
60
80
100
120

From here you can look into all the different types of particles, and use accurate metrics to be confident that the events you are identifying as particle decay actually are particle decay. This is a long process, which is why I will be focussing on the Z Boson.

The Z Boson

We know what bosons are and what they do. Let us discuss how the Z Boson forms and how it decays, along with its other properties. A spin 1 carrier of the weak force, Z bosons act as an intermediary step in the process of proton-proton collisions. These highly unstable particles last for an average of 3×10^{-25} seconds before decaying. Z bosons are only formed by the most influential collisions: their invariant mass of 91.2 GeV (the mass of subatomic particles can often be written in Gev - Gigaelectronvolts - as it makes the numbers we are working with more manageable. It is derived from $E=mc^2$), which is much much greater compared to the approximately 1 Gev of the colliding proton. A lot of energy is needed to simply create the particle.

Specific conditions were chosen to isolate and analyse Z boson events. These came directly from CERN, from “Luminosity determination using Z boson production at the CMS experiment” Section 3: Z boson candidate selection and efficiency determination. From there, the following criteria needed to be met to potentially consider the event a Z boson decay:

```
**Muon-Level Requirements (each muon in the pair):**
- Must be a **global muon**: `Type1 == 'G'` and `Type2 == 'G'`
- Transverse momentum **pT > 25 GeV** for both muons
- Pseudorapidity **|η| < 2.4**
- Muons must be of **opposite charge**: `Q1 * Q2 < 0`
```

Applying this, we were left with 4858 possible dimuon events that could be Z boson decay.

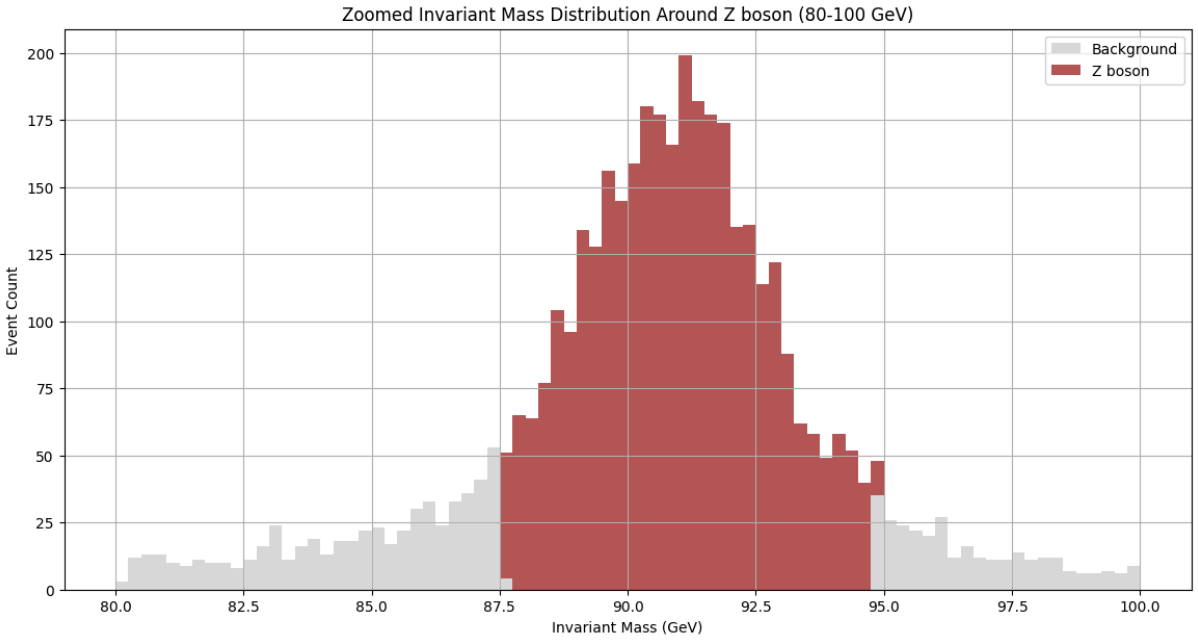
```
Filtered data size after muon ID and kinematic cuts: 4858 events
```

The mass range for the Z boson was taken by a sum of the uncertainty of the particle mass itself - it is hard to pinpoint the particle mass as it exists for such a short period of time. As such, there is always uncertainty related to its mass - and uncertainty due to the detector’s resolution:

```python
mZ_nominal = 91.1876  # GeV
uncertainty_physics = 0.03   # 3%
uncertainty_detector = 0.01  # 1%
total_uncertainty = uncertainty_physics + uncertainty_detector  # 4%

mass_lower = mZ_nominal * (1 - total_uncertainty)
mass_upper = mZ_nominal * (1 + total_uncertainty)

print(f"Z boson mass range with ±{total_uncertainty*100:.1f}% uncertainty:")
print(f"From {mass_lower:.3f} GeV to {mass_upper:.3f} GeV")
```

```
Z boson mass range with ±4.0% uncertainty:
From 87.540 GeV to 94.835 GeV
```

Finally, we end up with a possible 3357 events in the invariant mass range that are most likely to be events from Z boson decay.



4. Numerical Analysis

In this section, we compare the data of our events in order to get a more accurate representation.

A. Background Subtraction

From the classes, we can see that the data that does not match our invariant masses is labelled as 6 or “Background”. This is a rough initial step, as you observe in the histogram above, it is entirely possible that background events exist within the invariant mass range of our Z Boson. There is no perfect label for all the events, and there is no exact way of identifying how many background labels have been misclassified as Z bosons, therefore we use background subtraction - an approximation. To estimate how much of our Z peak is actually just noise, we fit a smooth curve—like a quadratic—through the 'quiet' regions on either side of the peak, then subtract this background estimate from the total event counts in each bin.

```
# --- 6. Background fit using sidebands ---
sideband_mask = ((bin_centers >= 80) & (bin_centers <= 85)) | ((bin_centers >= 97) & (bin_centers <= 100))
x_sb = bin_centers[sideband_mask]
y_sb = counts[sideband_mask]
```

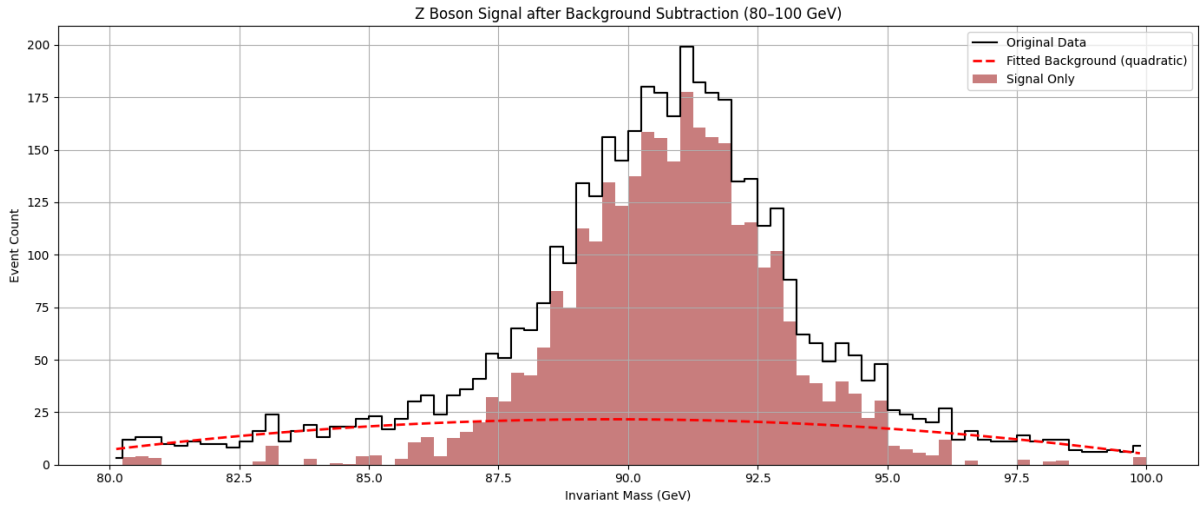
The sidebands are the little gray sections outside the Z boson invariant range. We use those to model how the background would continue into the Z boson range.

```
def poly_bg(x, a, b, c):
    return a * x**2 + b * x + c

popt, _ = curve_fit(poly_bg, x_sb, y_sb)
background_fit = poly_bg(bin_centers, *popt)
```

This defines the quadratic function, $f(x) = ax^2 + bx + c$.
'a' controls the curvature of the parabola (how steeply it bends).
'b' controls the slope or tilt of the parabola.
'c' is the y-intercept — where the curve crosses the y-axis.
In this case, poly_bg is the function we are trying to fit, with x values of 'x_sb', and y values of 'y_sb'.
x_sb is the 1D array of bin centers from the histogram.
y_sb is the count of events in each bin.
We do not set a, b and c manually, we use the function curve_fit from SciPi:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

Finally, we subtract the background, and plot the new histogram!

```
# --- 7. Subtract background and clip negative values ---
signal_only = counts - background_fit
signal_only = np.clip(signal_only, 0, None)
```



Now we have a new, and probably the most accurate count of Z boson decay event

candidates from the dataset:

```
✅ Total Z boson events after background subtraction: 2976
```

Just under 3000.

B. Modelling Z Boson Features

This section lets us compare the features of Z Boson from our data, to known features of Z bosons. Here is an example:
From all the events selected, we can calculate the mean invariant mass. Then, we can compare the mean invariant mass from our dataset to the real world value of 91.2 GeV, and see how close/far we are.

Now, such figures can be affected due to uncertainty as we discussed before: the detector resolution and the mass uncertainty, but it serves as a check to see if our filtering of Z boson events was somewhat accurate.

The difficult part lies in choosing the distribution. For modelling the Z boson, I took into consideration three distributions:

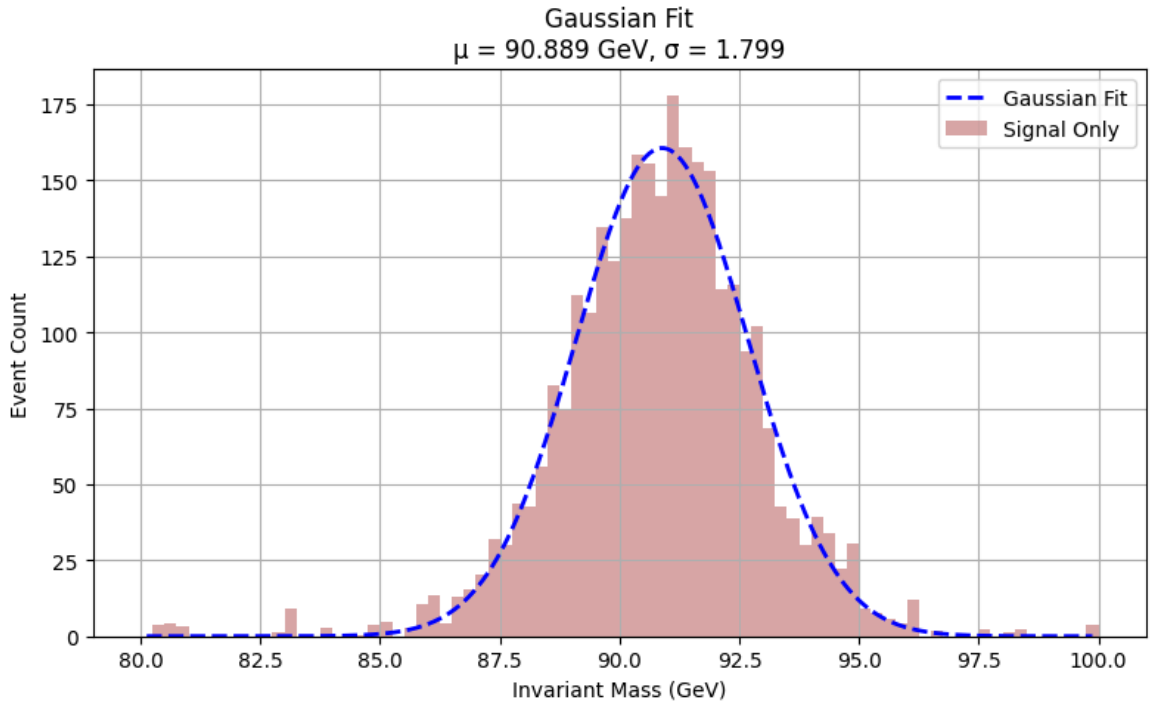
- The Gaussian / Normal
- The Lorentzian/Cauchy
- The Voigt Profile (A convolution of the previous two distributions)

*Note: The following steps involved a big leap into Statistics that if you are not wanting to learn, you should stick with the Normal as it is by far the most common distribution in statistics and can be understood by just basics.

The reason for using multiple distributions boils down to the fact that we cannot have a clear reading of the mass. Since there are multiple other parameters to account for outside the distribution of the mass, specific distributions are better than others at finding one of these parameters, and we compare them to see which is the most accurate, and which parameters affect our results the most.

Gaussian/Normal Distribution

A classic distribution where the data can be modelled so that the mean is equal to zero, and the standard deviation is 1. A characteristic of the Gaussian is the rapid fall of the tail, which indicates that the faster you move away from the mean, the closer to 0 probability of that event happening becomes. In theory, the Gaussian converges at infinity, which is useful when you carry out integration.



The mean invariant mass of 90.889 is ~0.3% away from the expected value of 91.2.
In the Gaussian distribution, the standard deviation is a result of detector smearing, due to the resolution of the detector. Detector smearing is ~ 2 GeV, which is close to the standard deviation of 1.8.

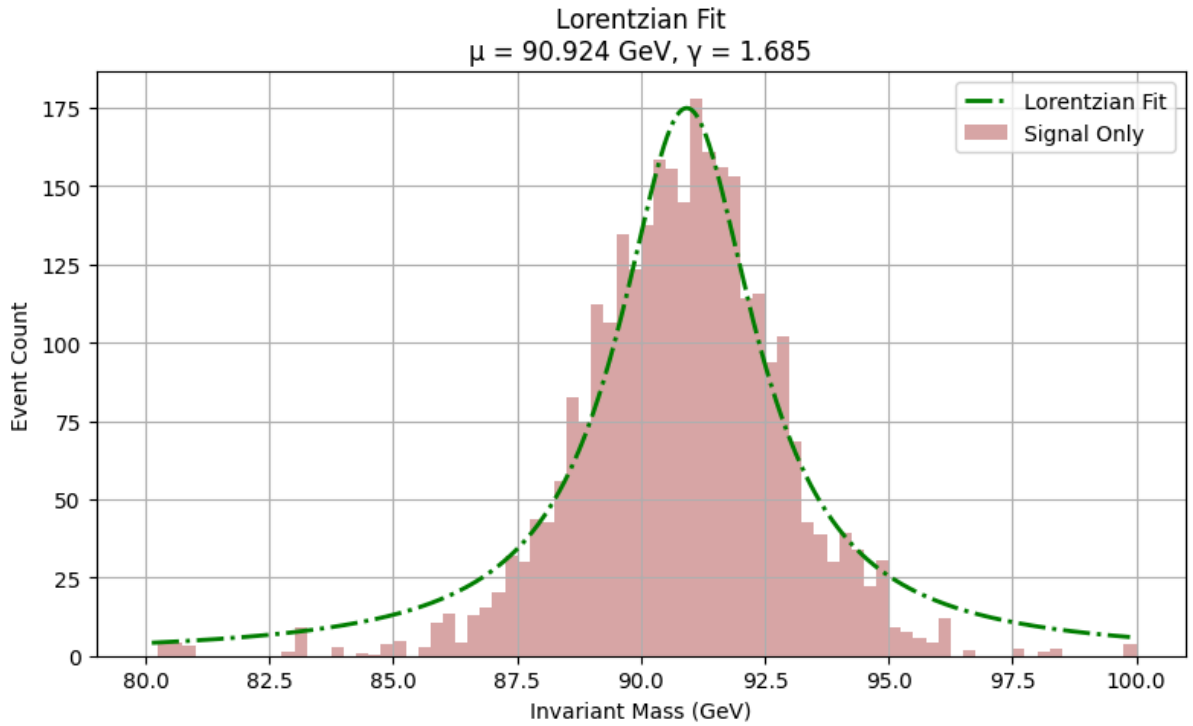
Detector smearing is a consequence due to the resolution of the detector compared to the mass of the particle, resulting in some percentage uncertainty. In the Gaussian graph, this is what gives the graph its width: σ.

Lorentzian/Cauchy

*Note: May also be referred to as the Breit Wigner Distribution. The two are closely linked.
The Lorentzian distribution has a key difference to the Gaussian - it does not converge. There is a much higher probability in the tails, so much so that even at infinity the probability is not 0. This gives two unique characteristics to the Lorentzian:

- It has no mean, since the mean is impossible to calculate if the distribution does not converge. Instead, the median is used instead. As a side effect, it also has no standard deviation.

- It is one of the few distributions that do not follow the Central Limit Theorem (C.L.T). The Central Limit Theorem suggests that if random samples are taken from a distribution, the mean of these samples will start to model a Gaussian distribution, no matter what the original distribution was. However, due to the high probability in the tails of the Lorentzian, a single sample containing a near infinity number will throw off the mean by an enormous margin.



The peak of the curve indicates the most probable mass of the Z Boson.

The γ of the Lorentzian is HWHM - Half Width at Half Maximum, a measure of half of the width of the curve at half of the height from the max. This value, 1.685, is a measure of half the value of the decay width of a particle - the Z boson.

The issue here is that when trying to determine the decay width, the detector smearing is still in effect, somewhat skewing results. The decay width determined from the Lorentzian, 3.37 GeV, is higher than the current accepted value of 2.5 GeV, which is evidence for the interference caused by detector smearing.

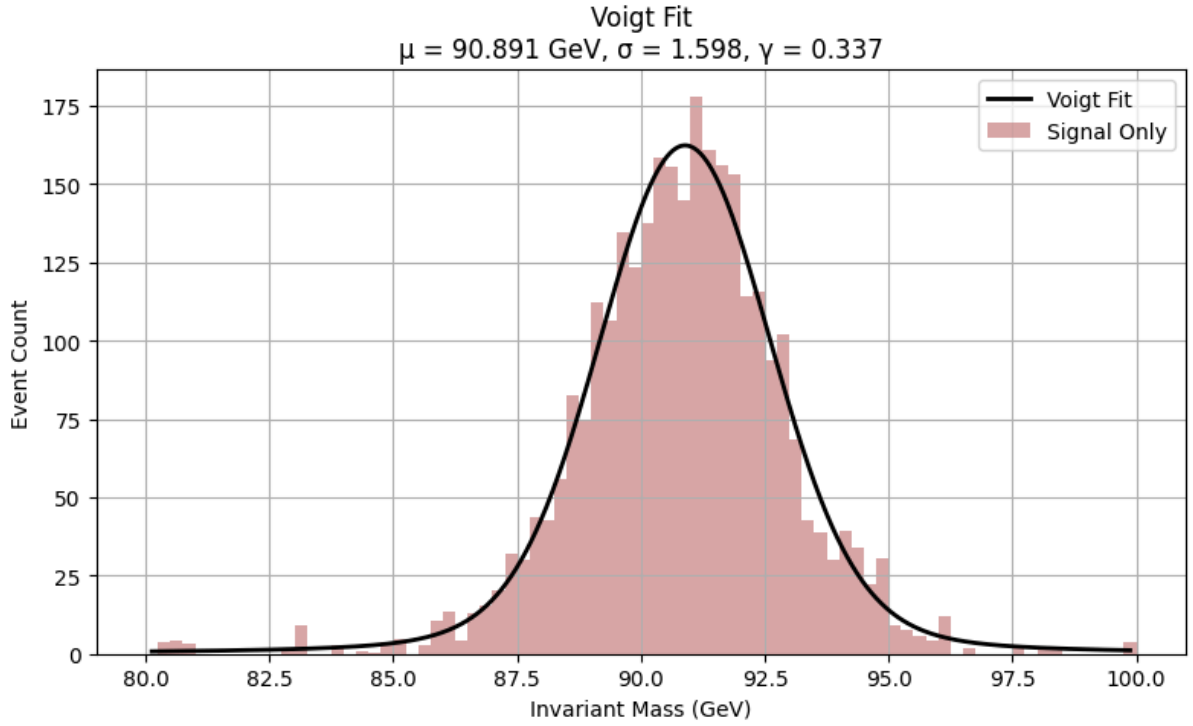
Voigt Profile

The Voigt profile is a combination (convolution) of the Gaussian and Lorentzian distributions. Why would we ever mix the two? Because both contribute to how particle data is spread:

- The Gaussian part accounts for detector resolution – how accurately the machinery measured the particle's properties. If your detector isn't perfect (spoiler: none are), the data gets smeared in a way that looks Gaussian.

- The Lorentzian part accounts for the natural width of the particle itself – a direct result of the Heisenberg Uncertainty Principle. Z bosons decay extremely fast (on the scale of 10^{-25} seconds), which means they don't have a sharply defined mass. Instead, there's a natural spread around the expected value, and that spread follows a Lorentzian curve.

So, what happens when you mix the two effects? You get the Voigt profile, which accurately models how the Z boson appears in real-life particle collisions, combining both physics and measurement effects into one curve.



Problems arise here as well. The decay width estimate is 0.674 GeV, a far way off from the true value of 2.5 GeV. This may be due to a more focused fitting of the Gaussian, since we had an aggressive filtering of possible Z boson events, especially outside the invariant mass range, the higher tails of the Lorentzian are less shown. Voigt Profile ends up favoring the Gaussian more. Also, there is overlap between detector smearing and decay width.

5. Monte Carlo Simulations

Named after the famous casino, the monte carlo simulation is a computational technique that involves using repeated random sampling to model a complex system or process, much like rolling dice.

By creating a bunch of fake experimental data using a distribution, you can simulate what you would expect in reality.

Basically, there are variables affecting the spread of Z Bosons like we mentioned before - detector smearing and decay width. By using the average known values for these variables, or in another sense saying how a Z Boson is meant to behave on average, a simulation can be run that shows the behavior of fake Z Bosons and how they are affected by the variables.Using that logic, we generate a large number of “fake” events — like Z bosons decaying into muons — each sampled from the theoretical Voigt distribution (a mix of the real decay width and the detector resolution). When we stack them all together, we get a histogram that should look a lot like real experimental data.

Defining the Voigt Profile, and Creating Fake Samples

```
def voigt_pdf(x, mu, sigma, gamma):
    z = ((x - mu) + 1j*gamma) / (sigma * np.sqrt(2))
    return np.real(wofz(z)) / (sigma * np.sqrt(2*np.pi))

# Parameters
mu = 91.1876
gamma = 2.495 / 2        # HWHM of Z decay width
sigma = 1.598            # Detector resolution from your fit
```

This is how the voigt profile is calculated from its parameters. Our variables are known scientific values for the decay width of the Z Boson and the detector resolution.

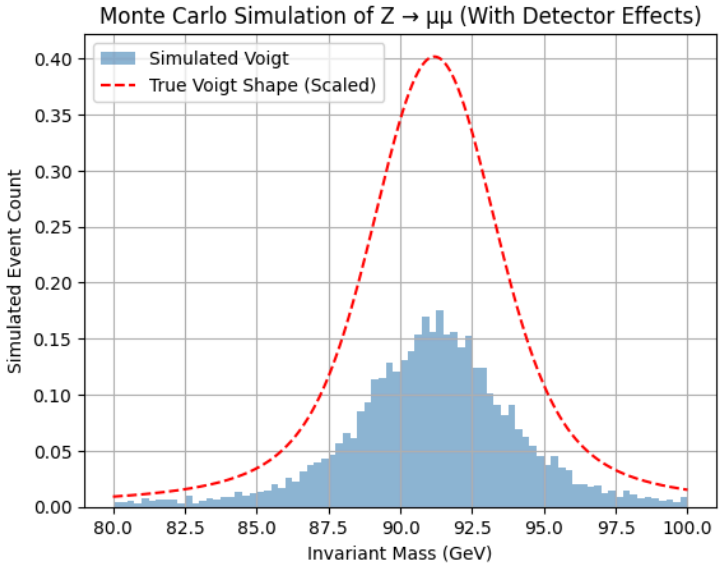
```
# Mass range and probability distribution
x_vals = np.linspace(80, 100, 1000)
voigt_probs = voigt_pdf(x_vals, mu, sigma, gamma)
voigt_probs /= voigt_probs.sum()  # Normalize

# Generate Monte Carlo sample
n_events = 10000
samples = np.random.choice(x_vals, size=n_events, p=voigt_probs)
```

The voigt distribution is spread over a mass range of 80-100 GeV, and normalised in order to be a proper probability distribution. 10,000 fake samples are created within the range.

Each one of these samples represents one fake Z Boson decay, taking into account the decay width and smearing.

Graphing

The true voigt shape is overlaid on top of the histogram distribution of the generated samples. This is comparing the fake created data to the IDEAL Voigt shape.



Looks pretty good.

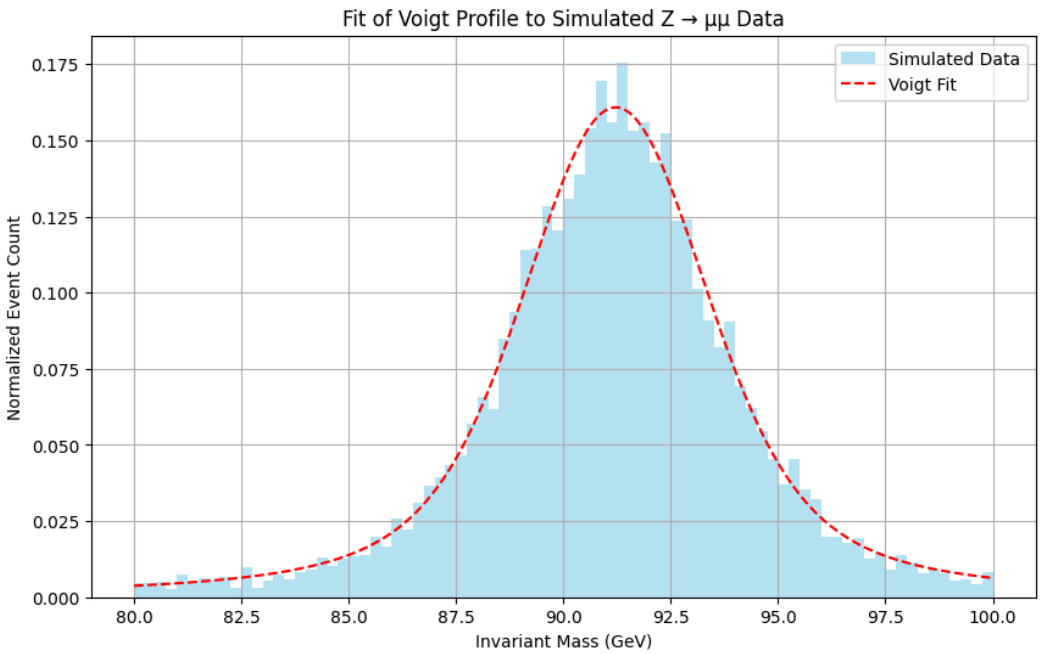
Finally, we model a NEW voigt profile onto our simulated data, and measure the parameters - they should be close to the ideal ones. This is a new voigt curve - the parameters will be found using the data, and it doesn't know the ones inputted before. Kinda like reverse engineering the parameters.

```python
from scipy.optimize import curve_fit

# --- Define Voigt profile as fit function ---
def voigt_func(x, mu, sigma, gamma, amplitude):
    z = ((x - mu) + 1j * gamma) / (sigma * np.sqrt(2))
    return amplitude * np.real(wofz(z)) / (sigma * np.sqrt(2 * np.pi))

# --- Create histogram from samples ---
hist_vals, bin_edges = np.histogram(samples, bins=80, range=(80, 100), density=True)
bin_centers = 0.5 * (bin_edges[1:] + bin_edges[:-1])

# --- Initial parameter guesses ---
initial_guess = [91.1876, 1.5, 1.42475, 0.15] # mu, sigma, gamma, amplitude

# --- Fit histogram to Voigt profile ---
popt, pcov = curve_fit(voigt_func, bin_centers, hist_vals, p0=initial_guess)
mu_fit, sigma_fit, gamma_fit, amp_fit = popt

# --- Extract fitted parameters ---
mu_fit, sigma_fit, gamma_fit, amp_fit = popt

# --- Plot the fit over histogram ---
x_fit = np.linspace(80, 100, 1000)
y_fit = voigt_func(x_fit, *popt)

plt.figure(figsize=(10, 6))
plt.hist(samples, bins=80, range=(80, 100), density=True, alpha=0.6, label='Simulated Data', color='skyblue')
plt.plot(x_fit, y_fit, 'r--', label='Voigt Fit')
plt.xlabel("Invariant Mass (GeV)")
plt.ylabel("Normalized Event Count")
plt.title("Fit of Voigt Profile to Simulated Z → μμ Data")
plt.legend()
plt.grid(True)
plt.show()
print("Fitted Parameters from Monte Carlo Simulation:")
print(f"μ (mean)      = {mu_fit:.4f} GeV")
print(f"σ (Gaussian) = {sigma_fit:.4f} GeV")
print(f"γ (Lorentzian) = {gamma_fit:.4f} GeV")
print(f"Amplitude     = {amp_fit:.4f}")
```

#Note: Wofz is the Faddeeva function mentioned earlier.

- voigt_func is the function that will be fit to the data. We provide the initial parameters. A good guess makes it easier to converge.

- curve_fit tries to find the optimal parameters (mu, sigma, gamma, amplitude) that make your Voigt function best match the histogram data.



Parameter	Monte Carlo Fit	True Simulation Input
μ (mean)	91.218 GeV	91.188 GeV
σ (Gaussian)	1.559 GeV	1.598 GeV
γ (Lorentzian)	1.293 GeV	1.247 GeV

And these are the monte carlo parameters - quite close to the ideal ones. If we compare them with our parameters from the experimental fitting:

Voigt Fit
μ = 90.891 GeV, σ = 1.598, γ = 0.337

Again, there is a mismatch with the Lorentzian part of the voigt profile.
The Monte Carlo simulation used the “true” decay width of the Z boson (2.495 GeV FWHM → 1.2475 GeV HWHM).

The experimental fit gives a much narrower width — around 0.337 GeV.

This is likely because the background was not fully subtracted in the experimental data, and the signal is blurred with noise, causing the Lorentzian (which captures the “tails” of the distribution) to shrink artificially.

6. Machine Learning

Here, a multiclass classification model will be built to predict the class (type of decay) of the muon.

The dataset will be split at random into train and test data, and be labelled with its corresponding class (to compare to the model's predictions). An 80-20 split of train and test data will be taken.

Class Imbalance

Now, because the probability of different particles emerging from the proton-proton collisions is different, and even with the signals enhanced, the background is dominant. This means we will be adding class weights in order to make up for this balance.

```
# Compute class weights
y_train_np = y_train.numpy()
classes = np.arange(7)
weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train_np)
class_weights = torch.tensor(weights, dtype=torch.float32)
```

Class weights are determined by:

$$w_c = \frac{n_{\text{samples}}}{n_{\text{classes}} \times n_c}$$

Why? Because if we take a look at our original dataset, 80,000 of the 100,000 events are background. This means that if the model only predicts the background 100% of the time, it will still end up with a 80% accuracy.

For this project, we would rather have a lower accuracy as long as the model can predict the particles accurately, a.k.a no false negatives, even if it does end up with more false positives. The loss now penalizes mistakes on minority classes more, encouraging the model to learn features from all classes. This helps reduce the dominance of the background class in training by emphasizing the importance of underrepresented classes.

The Model

```
import torch
from torch import nn
class ImprovedBoson(nn.Module):
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_shape, hidden_units),
            nn.BatchNorm1d(hidden_units),
            nn.LeakyReLU(),
            nn.Dropout(0.3),

            nn.Linear(hidden_units, hidden_units),
            nn.BatchNorm1d(hidden_units),
            nn.LeakyReLU(),
            nn.Dropout(0.3),

            nn.Linear(hidden_units, hidden_units),
            nn.BatchNorm1d(hidden_units),
            nn.LeakyReLU(),
            nn.Dropout(0.3),

            nn.Linear(hidden_units, output_shape)
        )
    def forward(self, x):
        return self.model(x)
```

nn.Module is subclassed to create a custom neural network.

def__init__ sets up the architecture, with the input shape being the number of useful input variables, and the output shape being the number of classes. nn.Sequential simply makes it easier to stack the layers.

We have three hidden blocks, here's what they do:

- nn.Linear(...): A fully connected (dense) layer.

- nn.BatchNorm1d(...): Batch normalization, which stabilizes and speeds up training by normalizing activations across the batch.

- nn.LeakyReLU(): A non-linear activation function; unlike ReLU, it allows a small gradient when the unit is inactive (helps prevent "dying neurons").

- nn.Dropout(0.3): Randomly disables 30% of neurons during training to reduce overfitting.

The output is raw logits for each class, which is because of the loss function.

```
# Model
model = ImprovedBoson(input_shape=14, hidden_units=64, output_shape=7)
loss_fn = nn.CrossEntropyLoss(weight=class_weights)
optimizer = torch.optim.Adam(model.parameters(), lr=0.010, weight_decay=0.0005)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=50, gamma=0.5)
```

Here is a brief explanation on what each of these are:

- Model: An algorithm that is capable of learning, by finding patterns in information, and creating their own methods. A neural network is used here, which is a model that kind of replicates a human brain.
- Loss function: A measure of how far the predicted value from the model is from the real one. Here, the loss function used is CrossEntropyLoss, which uses raw logits for

the loss calculations (the class weights calculated earlier are used here).

- Optimizer: An algorithm that changes the parameter (weights and biases) of the neural network, to help it predict better. If some input variables are more important than others, the optimizer tells the neural network to take these variables more seriously. The optimizer used here is Adam.

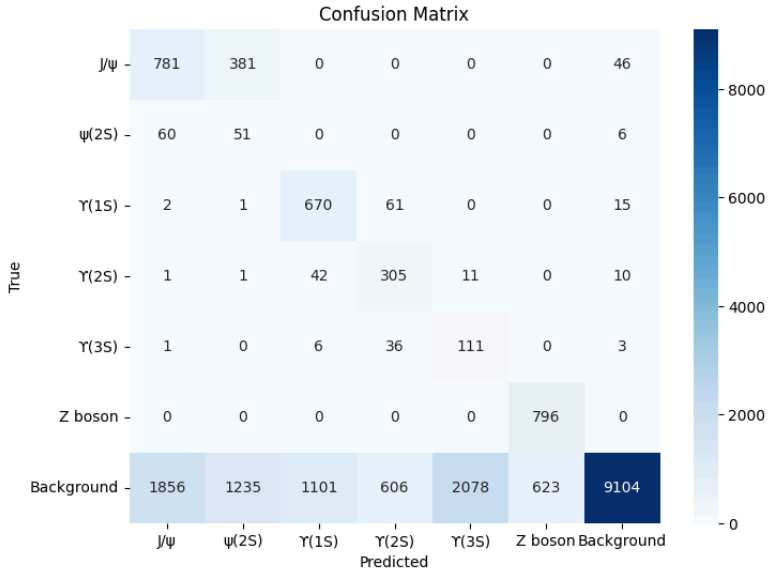
- Scheduler: Automatically decreases the learning rate as the epochs go by. The learning rate is a representation of how big the changes the optimizer can make to the parameters. Initially, we want to make big changes as the model is far away from the ideal parameters. However, as epochs go by, the learning rate should decrease in order for the optimizer to hone into the ideal parameters, instead of jumping around and never finding them. Here, after 50 epochs, the learning rate decreases by half.

Training and Testing Loop

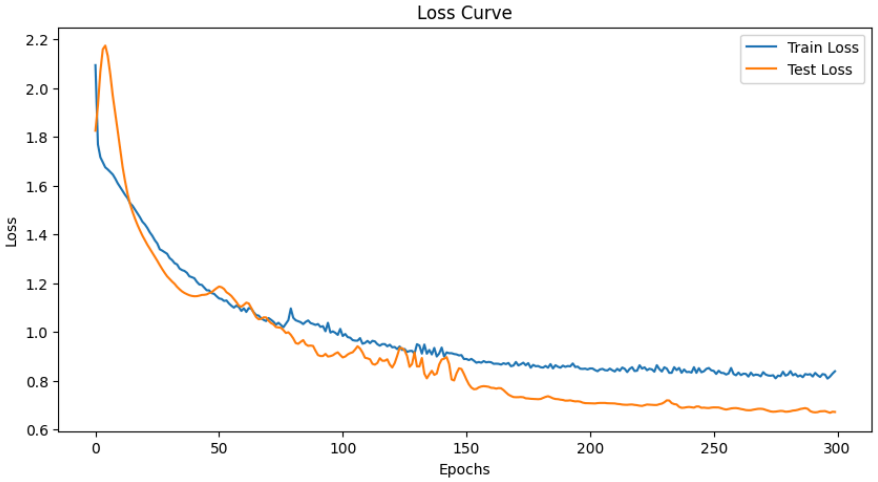
```python
# Accuracy function
def accuracy_fn(y_true, y_pred):
    return (torch.eq(y_true, y_pred).sum().item() / len(y_pred)) * 100

# Training loop
train_losses, test_losses = [], []
train_accuracy, test_accuracy = [], []

torch.manual_seed(42)
epochs = 300

for epoch in range(epochs):
    model.train()
    logits = model(X_train)
    preds = torch.argmax(logits, dim=1)
    loss = loss_fn(logits, y_train)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    acc = accuracy_fn(y_train, preds)
    train_losses.append(loss.item())
    train_accuracy.append(acc)

    model.eval()
    with torch.no_grad():
        test_logits = model(X_test)
        test_preds = torch.argmax(test_logits, dim=1)
        test_loss = loss_fn(test_logits, y_test)
        test_acc = accuracy_fn(y_test, test_preds)
        test_losses.append(test_loss.item())
        test_accuracy.append(test_acc)

    scheduler.step()
```

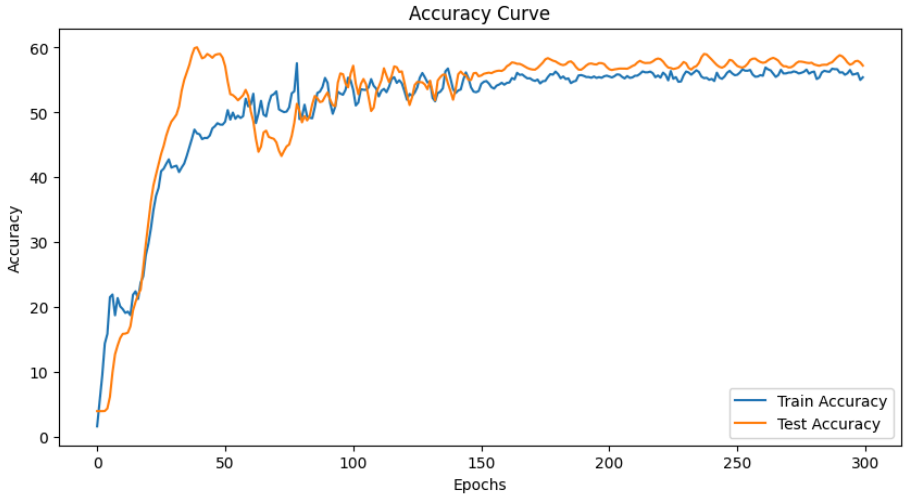
For everything we have done so far, including training, here is a link to Pytorch's explanation. Pytorch is the library being used to build the model, and a lot of the code: https://docs.pytorch.org/tutorials/beginner/introyt/trainingyt.html

Understanding our Model's Outputs



This is a confusion matrix, which shows the actual class against the class our model predicts .The confusion matrix shows substantial overlap between the Upsilon states (ϒ(1S), ϒ(2S), ϒ(3S)), likely due to their similar kinematic signatures and low event counts, making them hard to distinguish even for a neural network.





Loss appears to continually decrease whereas accuracy plateaus, which might suggest that the model has taken to just predict the majority classes, despite the class weights.

To counter this, class-specific probability thresholds will be applied. This basically tells the model that not only do you have to predict the class with the highest probability, but the probability you are guessing with has to be higher than the threshold (which is set by us for different classes), and if it doesn't meet such a requirement, that class will not be chosen.

```
Classification Report:
              precision    recall  f1-score   support

         J/ψ       0.29      0.65      0.40      1208
       ψ(2S)       0.03      0.44      0.06       117
       ϒ(1S)       0.37      0.89      0.52       749
       ϒ(2S)       0.30      0.82      0.44       370
       ϒ(3S)       0.05      0.71      0.09       157
     Z boson       0.56      1.00      0.72       796
  Background       0.99      0.55      0.71     16603

    accuracy                           0.59     20000
   macro avg       0.37      0.72      0.42     20000
weighted avg       0.88      0.59      0.67     20000

Multiclass ROC-AUC: 0.9354
```

This classification report, taken from the first loop, will help in determining the threshold boundaries.

- Precision,True Positive / (True Positive + False Positive), is a measure of false positives: Low precision means many false positives.

- Recall, True Positive / (True Positive + False Negative), is a measure of false negatives: Low recall means many false negatives.

- The F1-Score is the harmonic mean of precision and recall, and is calculated by

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

  It is used over regular mean as it penalises extreme values more, so that both precision and recall have to be high in order for a high f1-score - one cannot have too large of an effect on the mean if it is a very high value.

A lower precision roughly correlates to a higher threshold boundary, as the model is predicting a certain class too often (when it is not that class), and a higher threshold boundary raises the requirement for the model to predict that particular class.

Working with the results above, the threshold boundaries chosen were:

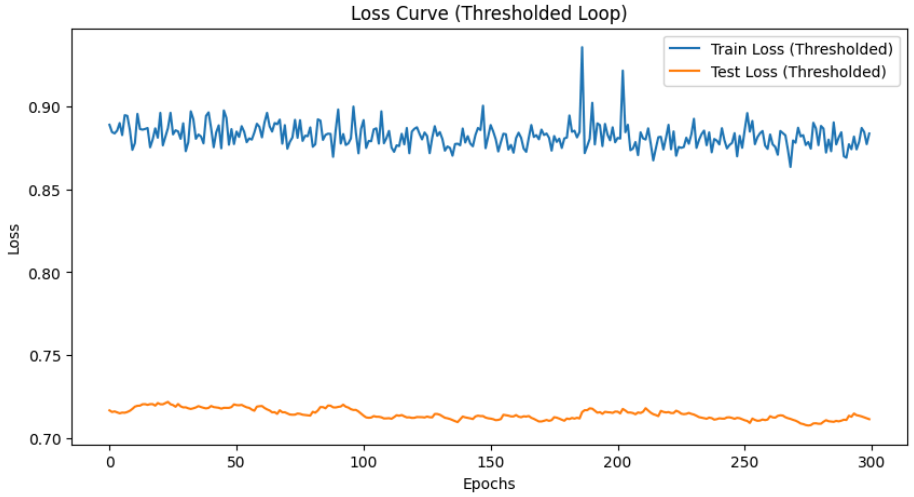
```
# Define custom thresholds (can be tuned)
thresholds = np.array([0.4,  # J/ψ
                       0.5,  # ψ(2S)
                       0.4,  # ϒ(1S)
                       0.375,  # ϒ(2S)
                       0.625,  # ϒ(3S)
                       0.6,  # Z boson
                       0.30]) # Background
```

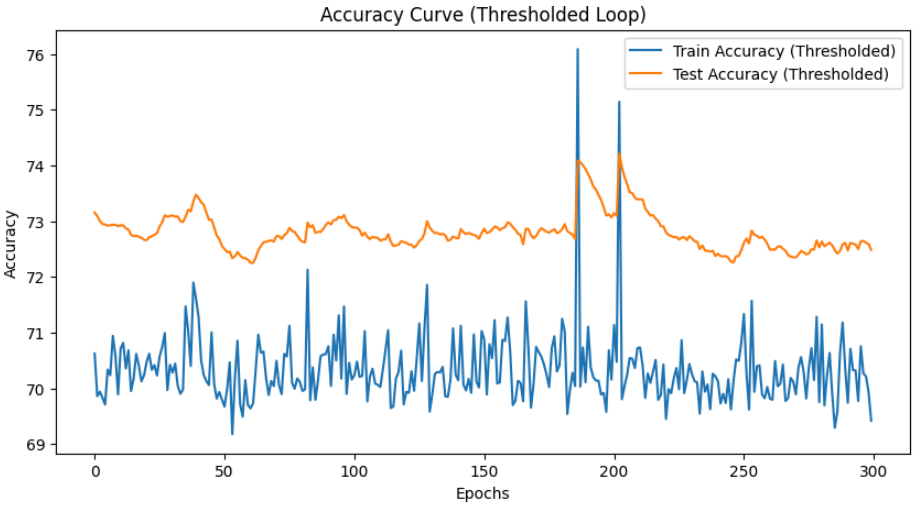
The highest threshold boundary is for the Upsilon 3 meson, which is due to its extremely low precision. Rerun the training-test loop, but instead of using torch.argmax, we will be comparing predicted probabilities to threshold boundaries.

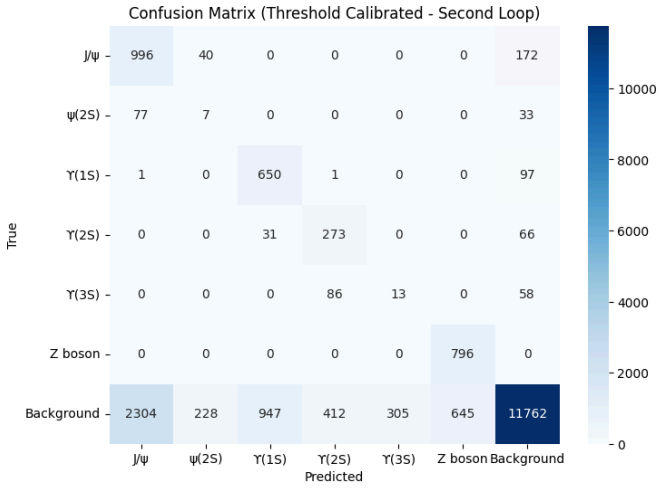
```
for epoch in range(epochs):
    model.train()
    logits_train_thresh = model(X_train)
    probs_train_thresh = torch.softmax(logits_train_thresh, dim=1).detach().cpu().numpy()
    y_train_np = y_train.cpu().numpy()

    # Threshold-based train predictions
    adjusted_train_preds = []
    for prob in probs_train_thresh:
        above_thresh = (prob >= thresholds)
        if np.any(above_thresh):
            adjusted_train_preds.append(np.argmax(prob * above_thresh))
        else:
            adjusted_train_preds.append(6)  # Default to Background
    adjusted_train_preds = torch.tensor(adjusted_train_preds)

    # Compute loss and backprop (use raw logits and original labels)
    loss_train_thresh = loss_fn(logits_train_thresh, y_train)
    optimizer.zero_grad()
    loss_train_thresh.backward()
    optimizer.step()

    # Train accuracy using thresholded preds
    acc_train_thresh = (adjusted_train_preds == y_train).sum().item() / len(y_train) * 100

    threshold_train_losses.append(loss_train_thresh.item())
    threshold_train_accuracy.append(acc_train_thresh)
```

```
    # Final thresholded predictions after the second loop
    model.eval()
    with torch.no_grad():
        final_test_logits = model(X_test)
        test_probs_thresh = torch.softmax(final_test_logits, dim=1).cpu().numpy()
        y_true_thresh = y_test.cpu().numpy()

        adjusted_preds_thresh = []
        for prob in test_probs_thresh:
            above_thresh = (prob >= thresholds)
            if np.any(above_thresh):
                adjusted_preds_thresh.append(np.argmax(prob * above_thresh))
            else:
                adjusted_preds_thresh.append(6)

        adjusted_test_preds = torch.tensor(adjusted_preds_thresh)

        loss_test_thresh = loss_fn(final_test_logits, y_test)
        acc_test_thresh = (adjusted_test_preds == y_test).sum().item() / len(y_test) * 100

        threshold_test_losses.append(loss_test_thresh.item())
        threshold_test_accuracy.append(acc_test_thresh)

    scheduler.step()
```

And the output:








```
Classification Report (Threshold Calibrated - Second Loop):
              precision    recall  f1-score   support

         J/ψ       0.29      0.82      0.43      1208
       ψ(2S)       0.03      0.06      0.04       117
       ϒ(1S)       0.40      0.87      0.55       749
       ϒ(2S)       0.35      0.74      0.48       370
       ϒ(3S)       0.04      0.08      0.05       157
     Z boson       0.55      1.00      0.71       796
  Background       0.97      0.71      0.82     16603

    accuracy                           0.72     20000
   macro avg       0.38      0.61      0.44     20000
weighted avg       0.86      0.72      0.76     20000

Multiclass ROC-AUC (Second Loop): 0.9340
```

The big issue here was that the model plateaus - it isn't learning. The loss and accuracy are almost a straight line. However, at an accuracy of 73%, this is where I ended my run. If you were trying to predict everything correctly, you would most likely need to increase the samples of the less represented classes in order to make the model train on them more. However, since the focus of this project is on Z Bosons - we catch all real Z Bosons, which is the most useful scenario for us. False Positives are acceptable as long as there are no False Negatives, because we filter out the true Z Boson events in the end anyways.

And there you have it — a rather chaotic, hands-on, and experimental dive into CERN data, deep learning, and particle physics. Was everything perfectly implemented?No. Some ideas (like Monte Carlo simulations and angular distributions) were ambitiously started and then abandoned when the graphs looked like two protons colliding instead of a curve. But the core mission — catching Z bosons using real collision data and machine learning — was a success. Along the way, I learned a ridiculous amount about muons, decay widths, detector smearing, and what happens when your neural network just gives up and predicts background every time. Would I improve the model further with simulations or fancier physics inputs? Maybe — but for now, I'm happy to have made it this far.