Cern3

https://opendata.cern.ch/record/30509
https://opendata.cern.ch/record/30509/files/CMS_Run2016G_MET_MINIAOD_UL2016_MiniAODv2-v2_1310000_file_index.json_0
(899.1 MB)

nano list_branches.py

```
import uproot
# Open the ROOT file
file = uproot.open("Cern3.root")
tree = file["Events"]
# List all available branches
print("Available branches:")
for name in tree.keys():
    print(name)
import uproot
files = [
    "/Users/bryanbarclay/Downloads/Cern1.root",
    "/Users/bryanbarclay/Downloads/Cern2.root",
    "/Users/bryanbarclay/Downloads/Cern3.root"
]
for f in files:
    with uproot.open(f) as file:
        print(f"\n==== {f} ====")
        tree = file["Events"]
        print(tree.keys())
```

nano step2_extract_features.py

```
import uproot
import awkward as ak
import pickle
# Open the ROOT file
file = uproot.open("Cern3.root")
tree = file["Events"]
# Only use branches that actually exist
branches = [
    "patPackedCandidates_packedPFCandidates__PAT.obj.packedPt_",
    "patPackedCandidates_packedPFCandidates__PAT.obj.packedEta_",
    "patPackedCandidates_packedPFCandidates__PAT.obj.packedPhi_",
    "patPackedCandidates_packedPFCandidates__PAT.obj.pdgId_"
]
# Extract arrays
arrays = tree.arrays(branches, library="ak")
# Save for Step 3
with open("step2_output.pkl", "wb") as f:
    pickle.dump(arrays, f)
print("✅ Step 2 complete — output saved to step2_output.pkl")
```

nano step3_clean_and_filter.py

```
import pickle
import awkward as ak
# Load the saved data from Step 2
with open("step2_output.pkl", "rb") as f:
    arrays = pickle.load(f)
# Extract individual arrays (use only valid fields)
pt = arrays["patPackedCandidates_packedPFCandidates__PAT.obj.packedPt_"]
eta = arrays["patPackedCandidates_packedPFCandidates__PAT.obj.packedEta_"]
phi = arrays["patPackedCandidates_packedPFCandidates__PAT.obj.packedPhi_"]
pdgId = arrays["patPackedCandidates_packedPFCandidates__PAT.obj.pdgId_"]
# Simple filter example: keep only charged particles (e.g., pdgId != 0)
charged_mask = pdgId != 0
# Apply mask to other variables
pt = pt[charged_mask]
eta = eta[charged_mask]
phi = phi[charged_mask]
pdgId = pdgId[charged_mask]
# Save the filtered arrays for next step
with open("step3_filtered.pkl", "wb") as f:
    pickle.dump({
        "pt": pt,
        "eta": eta,
        "phi": phi,
        "pdgId": pdgId
    }, f)
print("✅ Step 3 complete — filtered data saved to step3_filtered.pkl")
```

nano step4_physics_features.py

```
import pickle
import awkward as ak
import numpy as np
import pandas as pd
# Load filtered particle data
with open("step3_filtered.pkl", "rb") as f:
    arrays = pickle.load(f)
pt = arrays["pt"]
eta = arrays["eta"]
phi = arrays["phi"]
pdgId = arrays["pdgId"]
# Manual unique count per event (no ak.unique with axis)
def count_unique_per_event(arr):
    return ak.Array([len(set(event)) for event in arr])
df = pd.DataFrame({
    "event_id": np.arange(len(pt)),
    "num_particles": ak.num(pt),
    "mean_pt": ak.mean(pt, axis=1),
    "std_pt": ak.std(pt, axis=1),
    "sum_pt": ak.sum(pt, axis=1),
    "sum_eta": ak.sum(eta, axis=1),
    "sum_phi": ak.sum(phi, axis=1),
    "unique_pdgId_count": count_unique_per_event(pdgId)
})
# Save the output
with open("step4_features.pkl", "wb") as f:
    pickle.dump(df, f)
print("✅ Step 4 complete — physics features saved to step4_features.pkl")
```

nano step5_analyze_patterns.py

```
import pickle
import pandas as pd
# Load the physics features
with open("step4_features.pkl", "rb") as f:
    df = pickle.load(f)
# Define pulse pattern targets
pulse_numbers = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
# Build pattern match summary
results = {}
for col in ["num_particles", "sum_pt", "unique_pdgId_count"]:
    col_values = df[col].dropna().astype(int)
    matches = {}
    for pulse in pulse_numbers:
        count = (col_values == pulse).sum()
        matches[pulse] = int(count)
    results[col] = matches
# Output results
for col, match in results.items():
    print(f"\n--- {col} ---")
    for pulse, count in match.items():
        if count > 0:
            print(f"Matches pulse {pulse}: {count} events")
# Save clean dictionary format
with open("step5_patterns.pkl", "wb") as f:
    pickle.dump(results, f)
print("\n✅ Step 5 complete — pattern analysis saved to step5_patterns.pkl.")
```

nano step6_visualize_pulse.py

```
import pickle
import matplotlib.pyplot as plt
# Load the pattern results from Step 5
with open("step5_patterns.pkl", "rb") as f:
    data = pickle.load(f)
# Extract only the unique_pdgId_count results
unique_counts = data["unique_pdgId_count"]
# Prepare data for histogram
values = list(unique_counts.keys())
counts = list(unique_counts.values())
# Plot
plt.bar(values, counts, width=1.0, align='center', edgecolor='black')
plt.xlabel("Unique PDG ID Count")
plt.ylabel("Number of Events")
plt.title("Distribution of Unique PDG IDs per Event with Pulse Overlays")
# Overlay pulse lines
pulse_numbers = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
for pulse in pulse_numbers:
    plt.axvline(pulse, color='red', linestyle='--')
    plt.text(pulse, max(counts) * 0.95, str(pulse), color='red', rotation=90, ha='right', fontsize=8)
plt.tight_layout()
plt.savefig("cern_pulse_overlay.png")
plt.show()
print("\n✅ Step 6 complete — pulse visualization saved as cern_pulse_overlay.png")
```

nano step7_statistical_validation.py

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
# Load saved features from Step 4
with open("step4_features.pkl", "rb") as f:
    features = pickle.load(f)
unique_pdg_counts = features["unique_pdgId_count"]
# Define pulse numbers
pulse_values = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
# Histogram data
counts, bins = np.histogram(unique_pdg_counts, bins=range(0, max(unique_pdg_counts)+2))
bin_centers = bins[:-1]
hist_dict = dict(zip(bin_centers, counts))
# Total number of events for reference
total_events = len(unique_pdg_counts)
# Calculate expected mean and std
mean = np.mean(unique_pdg_counts)
std = np.std(unique_pdg_counts)
# Monte Carlo simulation
np.random.seed(42)
simulated_counts = []
for _ in range(100000):
    sim_data = np.random.normal(loc=mean, scale=std, size=total_events)
    sim_hist, _ = np.histogram(sim_data, bins=range(0, max(unique_pdg_counts)+2))
    sim_counts = [sim_hist[val] if val < len(sim_hist) else 0 for val in pulse_values]
    simulated_counts.append(sim_counts)
simulated_counts = np.array(simulated_counts)
# Print validation results
print("\n--- Statistical Validation of Pulse Hits ---\n")
for i, pulse in enumerate(pulse_values):
    actual = hist_dict.get(pulse, 0)
    sims = simulated_counts[:, i]
    sim_mean = np.mean(sims)
    sim_std = np.std(sims)
    if sim_std > 0:
        z = (actual - sim_mean) / sim_std
        p = 1 - norm.cdf(z)
    print(f"Pulse {pulse}: Actual = {actual}, Expected = {sim_mean:.2f}, Z = {z:.2f}, p = {p:.5f}")
    else:
        print(f"Pulse {pulse}: Actual = {actual}, Expected = {sim_mean:.2f}, Z = ∞ (no variation)")

# Optional save
with open("step7_stats_summary.txt", "w") as f_out:
    f_out.write("--- Statistical Validation of Pulse Hits ---\n\n")
    for i, pulse in enumerate(pulse_values):
        actual = hist_dict.get(pulse, 0)
        sims = simulated_counts[:, i]
        sim_mean = np.mean(sims)
        sim_std = np.std(sims)
        if sim_std > 0:
            z = (actual - sim_mean) / sim_std
            p = 1 - norm.cdf(z)
            f_out.write(f"Pulse {pulse}: Actual = {actual}, Expected = {sim_mean:.2f}, Z = {z:.2f}, p = {p:.5f}\n")
        else:
            f_out.write(f"Pulse {pulse}: Actual = {actual}, Expected = {sim_mean:.2f}, Z = ∞ (no variation)\n")

print("\n✅ Step 7 complete — statistical validation finished. Summary saved to 'step7_stats_summary.txt'.")
```

nano step8_montecarlo_expand.py

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
# Load pattern results from Step 5
with open("step5_patterns.pkl", "rb") as f:
    pattern_data = pickle.load(f)
# Use the count data for 'unique_pdgId_count'
pulse_hits = pattern_data["unique_pdgId_count"]
# Define pulse values
pulse_values = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
observed_counts = [pulse_hits.get(pulse, 0) for pulse in pulse_values]
# Monte Carlo simulation: 100,000 trials
total_events = sum(pulse_hits.values())
prob_distribution = np.ones(len(pulse_values)) / len(pulse_values)
np.random.seed(42)
simulations = np.random.multinomial(total_events, prob_distribution, size=100000)
# Compute Z-scores and p-values
print("\n--- Step 8: Monte Carlo Pulse Simulation ---\n")
for i, pulse in enumerate(pulse_values):
    actual = observed_counts[i]
    sim_vals = simulations[:, i]
    mean = np.mean(sim_vals)
    std = np.std(sim_vals)
    if std > 0:
        z = (actual - mean) / std
        p = 1 - (np.sum(sim_vals >= actual) / len(sim_vals))
        print(f"Pulse {pulse}: Actual = {actual}, Simulated Mean = {mean:.2f}, Z = {z:.2f}, p = {p:.5f}")
    else:
        print(f"Pulse {pulse}: Actual = {actual}, Simulated Mean = {mean:.2f}, Z = ∞ (no variation)")
# Optional: Plot histogram for one pulse
pulse_index = 0  # Example: pulse 7
plt.hist(simulations[:, pulse_index], bins=50, alpha=0.7, label="Simulated")
plt.axvline(observed_counts[pulse_index], color='r', linestyle='dashed', linewidth=2, label="Actual")
plt.title(f"Pulse {pulse_values[pulse_index]} — Observed vs Simulated Distribution")
plt.xlabel("Simulated Counts")
plt.ylabel("Frequency")
plt.legend()
plt.savefig("step8_pulse_simulation.png")
```

--- Statistical Validation of Pulse Hits ---
Pulse 7: Actual = 6908, Expected = 10576.25, Z = -44.94, p = 0.0
Pulse 12: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 21: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 28: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 42: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 49: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 70: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 91: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 112: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 133: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 144: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 153: Actual = 0, Expected = 0.00, Z = ∞ (no variation)
Pulse 343: Actual = 0, Expected = 0.00, Z = ∞ (no variation)

--- Step 8: Monte Carlo Pulse Simulation ---
Pulse 7: Actual = 6908, Simulated Mean = 531.39, Z = 287.07, p = 0.00001
Pulse 12: Actual = 0, Simulated Mean = 531.29, Z = -23.98, p = 0.00000
Pulse 21: Actual = 0, Simulated Mean = 531.39, Z = -23.99, p = 0.00000
Pulse 28: Actual = 0, Simulated Mean = 531.40, Z = -24.00, p = 0.00000
Pulse 42: Actual = 0, Simulated Mean = 531.47, Z = -24.10, p = 0.00000
Pulse 49: Actual = 0, Simulated Mean = 531.24, Z = -23.96, p = 0.00000
Pulse 70: Actual = 0, Simulated Mean = 531.46, Z = -24.07, p = 0.00000
Pulse 91: Actual = 0, Simulated Mean = 531.29, Z = -24.09, p = 0.00000
Pulse 112: Actual = 0, Simulated Mean = 531.48, Z = -23.90, p = 0.00000
Pulse 133: Actual = 0, Simulated Mean = 531.39, Z = -23.86, p = 0.00000
Pulse 144: Actual = 0, Simulated Mean = 531.48, Z = -24.01, p = 0.00000
Pulse 153: Actual = 0, Simulated Mean = 531.39, Z = -24.00, p = 0.00000
Pulse 343: Actual = 0, Simulated Mean = 531.33, Z = -24.03, p = 0.00000

CERN 1
https://opendata.cern.ch/record/31309
https://opendata.cern.ch/record/31309/files/MET_PFNano_29-Feb-24_Run2016G-UL2016_MiniAODv2_PFNanoAODv1_root_file_index.json_14
563.6 MB

nano step2_extract_features.py

```
import uproot
import awkward as ak
import pickle

# Open the ROOT file
file = uproot.open("Cern1.root")
tree = file["Events"]

# Use only branches that exist in Cern1.root
branches = [
    "PFCands_pt",
    "PFCands_eta",
    "PFCands_phi",
    "PFCands_pdgId"
]

# Extract arrays
arrays = tree.arrays(branches, library="ak")

# Save for Step 3
with open("step2_output.pkl", "wb") as f:
    pickle.dump(arrays, f)

print("✅ Step 2 complete — output saved to step2_output.pkl")
```

nano step3_clean_and_filter.py

```
import pickle
import awkward as ak

# Load the saved data from Step 2
with open("step2_output.pkl", "rb") as f:
    arrays = pickle.load(f)

# Extract individual arrays using correct Cern1 field names
pt = arrays["PFCands_pt"]
eta = arrays["PFCands_eta"]
phi = arrays["PFCands_phi"]
pdgId = arrays["PFCands_pdgId"]

# Simple filter: keep only charged particles (e.g., pdgId != 0)
charged_mask = pdgId != 0

# Apply mask to all arrays
pt = pt[charged_mask]
eta = eta[charged_mask]
phi = phi[charged_mask]
pdgId = pdgId[charged_mask]

# Save the filtered arrays
with open("step3_filtered.pkl", "wb") as f:
    pickle.dump({
        "pt": pt,
        "eta": eta,
        "phi": phi,
        "pdgId": pdgId
    }, f)

print("✅ Step 3 complete — filtered data saved to step3_filtered.pkl")
```

nano step4_physics_features.py

```
import pickle
import awkward as ak
import numpy as np
import pandas as pd
# Load filtered particle data
with open("step3_filtered.pkl", "rb") as f:
    arrays = pickle.load(f)
pt = arrays["pt"]
eta = arrays["eta"]
phi = arrays["phi"]
pdgId = arrays["pdgId"]
# Manual unique count per event
def count_unique_per_event(arr):
    return ak.Array([len(set(event)) for event in arr])
# Create DataFrame with physics features
df = pd.DataFrame({
    "event_id": np.arange(len(pt)),
    "num_particles": ak.num(pt),
    "mean_pt": ak.mean(pt, axis=1),
    "std_pt": ak.std(pt, axis=1),
    "sum_pt": ak.sum(pt, axis=1),
    "sum_eta": ak.sum(eta, axis=1),
    "sum_phi": ak.sum(phi, axis=1),
    "unique_pdgId_count": count_unique_per_event(pdgId)
})
# Save the feature set
with open("step4_features.pkl", "wb") as f:
    pickle.dump(df, f)
print("✅ Step 4 complete — physics features saved to step4_features.pkl")
```

nano step5_analyze_patterns.py

```
import pickle
import pandas as pd
# Load the physics features
with open("step4_features.pkl", "rb") as f:
    df = pickle.load(f)
# Define pulse pattern targets
pulse_numbers = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
# Build pattern match summary
results = {}
for col in ["num_particles", "sum_pt", "unique_pdgId_count"]:
    col_values = df[col].dropna().astype(int)
    matches = {}
    for pulse in pulse_numbers:
        count = (col_values == pulse).sum()
        matches[pulse] = int(count)
    results[col] = matches
# Output results
for col, match in results.items():
```

```
    print(f"\n--- {col} ---")
    for pulse, count in match.items():
        if count > 0:
            print(f"Matches pulse {pulse}: {count} events")
# Save clean dictionary format
with open("step5_patterns.pkl", "wb") as f:
    pickle.dump(results, f)
print("\n✅ Step 5 complete — pattern analysis saved to step5_patterns.pkl.")
```

nano step6_visualize_pulses.py

```
import pickle
import matplotlib.pyplot as plt

# Load the pattern results from Step 5
with open("step5_patterns.pkl", "rb") as f:
    data = pickle.load(f)

# Extract only the unique_pdgId_count results
unique_counts = data["unique_pdgId_count"]

# Prepare data for histogram
values = list(unique_counts.keys())
counts = list(unique_counts.values())

# Plot
plt.bar(values, counts, width=1.0, align='center', edgecolor='black')
plt.xlabel("Unique PDG ID Count")
plt.ylabel("Number of Events")
plt.title("Distribution of Unique PDG IDs per Event with Pulse Overlays")

# Overlay pulse lines
pulse_numbers = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
for pulse in pulse_numbers:
    plt.axvline(pulse, color='red', linestyle='--')
    plt.text(pulse, max(counts) * 0.95, str(pulse), color='red', rotation=90, ha='right', fontsize=8)

plt.tight_layout()
plt.savefig("cern_pulse_overlay.png")
plt.show()

print("\n✅ Step 6 complete — pulse visualization saved as cern_pulse_overlay.png")
```

nano step7_statistical_validation.py

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Load saved features from Step 4
with open("step4_features.pkl", "rb") as f:
    features = pickle.load(f)

unique_pdg_counts = features["unique_pdgId_count"]

# Define pulse numbers
pulse_values = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]

# Histogram actual data
counts, bins = np.histogram(unique_pdg_counts, bins=range(0, max(unique_pdg_counts)+2))
bin_centers = bins[:-1]
hist_dict = dict(zip(bin_centers, counts))

# Total number of events
total_events = len(unique_pdg_counts)

# Normal distribution parameters
mean = np.mean(unique_pdg_counts)
std = np.std(unique_pdg_counts)

# Monte Carlo simulation
np.random.seed(42)
simulated_counts = []
for _ in range(100000):
    sim_data = np.random.normal(loc=mean, scale=std, size=total_events)
    sim_hist, _ = np.histogram(sim_data, bins=range(0, max(unique_pdg_counts)+2))
    sim_counts = [sim_hist[val] if val < len(sim_hist) else 0 for val in pulse_values]
    simulated_counts.append(sim_counts)

simulated_counts = np.array(simulated_counts)

# Statistical Validation
print("\n--- Statistical Validation of Pulse Hits ---\n")
for i, pulse in enumerate(pulse_values):
    actual = hist_dict.get(pulse, 0)
    sims = simulated_counts[:, i]
    sim_mean = np.mean(sims)
    sim_std = np.std(sims)
    if sim_std > 0:
        z = (actual - sim_mean) / sim_std
```

```
        p = 1 - norm.cdf(z)
        print(f"Pulse {pulse}: Actual = {actual}, Simulated μ = {sim_mean:.2f}, σ = {sim_std:.2f}, z = {z:.2f}, p = {p:.4e}")
    else:
        print(f"Pulse {pulse}: Actual = {actual}, Sim std deviation = 0 — skipped")
```

nano step8_montecarlo_expand.py

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
# Load pattern results from Step 5
with open("step5_patterns.pkl", "rb") as f:
    pattern_data = pickle.load(f)
# Use the count data for 'unique_pdgId_count'
pulse_hits = pattern_data["unique_pdgId_count"]
# Define pulse values
pulse_values = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
observed_counts = [pulse_hits.get(pulse, 0) for pulse in pulse_values]
# Monte Carlo simulation: 100,000 trials
total_events = sum(pulse_hits.values())
prob_distribution = np.ones(len(pulse_values)) / len(pulse_values)
np.random.seed(42)
simulations = np.random.multinomial(total_events, prob_distribution, size=100000)
# Compute Z-scores and p-values
print("\n--- Step 8: Monte Carlo Pulse Simulation ---\n")
for i, pulse in enumerate(pulse_values):
    actual = observed_counts[i]
    sim_vals = simulations[:, i]
    mean = np.mean(sim_vals)
    std = np.std(sim_vals)
    if std > 0:
        z = (actual - mean) / std
        p = 1 - (np.sum(sim_vals >= actual) / len(sim_vals))
        print(f"Pulse {pulse}: Actual = {actual}, Simulated Mean = {mean:.2f}, Z = {z:.2f}, p = {p:.5f}")
    else:
        print(f"Pulse {pulse}: Actual = {actual}, Simulated Mean = {mean:.2f}, Z = ∞ (no variation)")
# Optional: Plot histogram for one pulse
pulse_index = 0  # Example: pulse 7
plt.hist(simulations[:, pulse_index], bins=50, alpha=0.7, label="Simulated")
plt.axvline(observed_counts[pulse_index], color='r', linestyle='dashed', linewidth=2, label="Actual")
plt.title(f"Pulse {pulse_values[pulse_index]} — Observed vs Simulated Distribution")
plt.xlabel("Simulated Counts")
plt.ylabel("Frequency")
plt.legend()
plt.savefig("step8_pulse_simulation.png")
```

--- Statistical Validation of Pulse Hits ---

Pulse 7: Actual = 11239, Simulated μ = 15210.56, σ = 98.28, z = -40.41, p = 1.0000e+00
Pulse 12: Actual = 0, Sim std deviation = 0 — skipped
Pulse 21: Actual = 0, Sim std deviation = 0 — skipped
Pulse 28: Actual = 0, Sim std deviation = 0 — skipped
Pulse 42: Actual = 0, Sim std deviation = 0 — skipped
Pulse 49: Actual = 0, Sim std deviation = 0 — skipped
Pulse 70: Actual = 0, Sim std deviation = 0 — skipped
Pulse 91: Actual = 0, Sim std deviation = 0 — skipped
Pulse 112: Actual = 0, Sim std deviation = 0 — skipped
Pulse 133: Actual = 0, Sim std deviation = 0 — skipped
Pulse 144: Actual = 0, Sim std deviation = 0 — skipped
Pulse 153: Actual = 0, Sim std deviation = 0 — skipped
Pulse 343: Actual = 0, Sim std deviation = 0 — skipped

--- Step 8: Monte Carlo Pulse Simulation ---

Pulse 7: Actual = 11239, Simulated Mean = 864.55, Z = 366.10, p = 1.00000
Pulse 12: Actual = 0, Simulated Mean = 864.45, Z = -30.68, p = 0.00000
Pulse 21: Actual = 0, Simulated Mean = 864.62, Z = -30.55, p = 0.00000
Pulse 28: Actual = 0, Simulated Mean = 864.63, Z = -30.51, p = 0.00000
Pulse 42: Actual = 0, Simulated Mean = 864.40, Z = -30.69, p = 0.00000
Pulse 49: Actual = 0, Simulated Mean = 864.49, Z = -30.62, p = 0.00000
Pulse 70: Actual = 0, Simulated Mean = 864.54, Z = -30.54, p = 0.00000
Pulse 91: Actual = 0, Simulated Mean = 864.60, Z = -30.61, p = 0.00000
Pulse 112: Actual = 0, Simulated Mean = 864.54, Z = -30.62, p = 0.00000
Pulse 133: Actual = 0, Simulated Mean = 864.51, Z = -30.65, p = 0.00000
Pulse 144: Actual = 0, Simulated Mean = 864.49, Z = -30.66, p = 0.00000
Pulse 153: Actual = 0, Simulated Mean = 864.68, Z = -30.71, p = 0.00000
Pulse 343: Actual = 0, Simulated Mean = 864.50, Z = -30.63, p = 0.00000

CERN 2
https://opendata.cern.ch/record/30508
https://opendata.cern.ch/record/30508/files/CMS_Run2016G_JetHT_MINIAOD_UL2016_MiniAODv2-v2_130000_file_index.json_1
123.3 MB

nano step2_extract_features.py

```
import uproot
import awkward as ak
import pickle

# Open the ROOT file
file = uproot.open("Cern2.root")
tree = file["Events"]

# Use verified branches
branches = [
    "patPackedCandidates_packedPFCandidates__PAT.obj.packedPt_",
    "patPackedCandidates_packedPFCandidates__PAT.obj.packedEta_",
    "patPackedCandidates_packedPFCandidates__PAT.obj.packedPhi_",
    "patPackedCandidates_packedPFCandidates__PAT.obj.pdgId_"
]

# Extract arrays
arrays = tree.arrays(branches, library="ak")

# Save for Step 3
with open("step2_output.pkl", "wb") as f:
    pickle.dump(arrays, f)

print("✅ Step 2 complete — output saved to step2_output.pkl")
```

nano step3_clean_and_filter.py

```
import pickle
import awkward as ak

# Load the saved data from Step 2
with open("step2_output.pkl", "rb") as f:
    arrays = pickle.load(f)

# Extract individual arrays (use only valid fields)
pt = arrays["patPackedCandidates_packedPFCandidates__PAT.obj.packedPt_"]
eta = arrays["patPackedCandidates_packedPFCandidates__PAT.obj.packedEta_"]
phi = arrays["patPackedCandidates_packedPFCandidates__PAT.obj.packedPhi_"]
pdgId = arrays["patPackedCandidates_packedPFCandidates__PAT.obj.pdgId_"]

# Simple filter example: keep only charged particles (e.g., pdgId != 0)
charged_mask = pdgId != 0

# Apply mask to other variables
pt = pt[charged_mask]
eta = eta[charged_mask]
phi = phi[charged_mask]
pdgId = pdgId[charged_mask]

# Save the filtered arrays for next step
with open("step3_filtered.pkl", "wb") as f:
    pickle.dump({
        "pt": pt,
        "eta": eta,
        "phi": phi,
        "pdgId": pdgId
    }, f)

print("✅ Step 3 complete — filtered data saved to step3_filtered.pkl")
```

nano step4_physics_features.py

```
import pickle
import awkward as ak
import numpy as np
import pandas as pd

# Load filtered particle data
with open("step3_filtered.pkl", "rb") as f:
    arrays = pickle.load(f)

pt = arrays["pt"]
eta = arrays["eta"]
phi = arrays["phi"]
pdgId = arrays["pdgId"]

# Manual unique count per event (no ak.unique with axis)
def count_unique_per_event(arr):
    return ak.Array([len(set(event)) for event in arr])

# Build feature dataframe
df = pd.DataFrame({
    "event_id": np.arange(len(pt)),
    "num_particles": ak.num(pt),
    "mean_pt": ak.mean(pt, axis=1),
    "std_pt": ak.std(pt, axis=1),
    "sum_pt": ak.sum(pt, axis=1),
    "sum_eta": ak.sum(eta, axis=1),
    "sum_phi": ak.sum(phi, axis=1),
    "unique_pdgId_count": count_unique_per_event(pdgId)
})

# Save the output
with open("step4_features.pkl", "wb") as f:
    pickle.dump(df, f)

print("✅ Step 4 complete — physics features saved to step4_features.pkl")
```

nano step5_analyze_patterns.py

```
import pickle
import pandas as pd

# Load the physics features
with open("step4_features.pkl", "rb") as f:
    df = pickle.load(f)

# Define pulse pattern targets
pulse_numbers = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]

# Build pattern match summary
results = {}
for col in ["num_particles", "sum_pt", "unique_pdgId_count"]:
    col_values = df[col].dropna().astype(int)
    matches = {}
    for pulse in pulse_numbers:
        count = (col_values == pulse).sum()
        matches[pulse] = int(count)
    results[col] = matches

# Output results
for col, match in results.items():
    print(f"\n--- {col} ---")
    for pulse, count in match.items():
        if count > 0:
            print(f"Matches pulse {pulse}: {count} events")

# Save clean dictionary format
with open("step5_patterns.pkl", "wb") as f:
    pickle.dump(results, f)

print("\n✅ Step 5 complete — pattern analysis saved to step5_patterns.pkl.")
```

nano step6_visualize_pulses.py

```
import pickle
import matplotlib.pyplot as plt

# Load the pattern results from Step 5
with open("step5_patterns.pkl", "rb") as f:
    data = pickle.load(f)

# Extract only the unique_pdgId_count results
unique_counts = data["unique_pdgId_count"]

# Prepare data for histogram
values = list(unique_counts.keys())
counts = list(unique_counts.values())

# Plot
plt.bar(values, counts, width=1.0, align='center', edgecolor='black')
plt.xlabel("Unique PDG ID Count")
plt.ylabel("Number of Events")
plt.title("Distribution of Unique PDG IDs per Event with Pulse Overlays")

# Overlay pulse lines
pulse_numbers = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
for pulse in pulse_numbers:
    plt.axvline(pulse, color='red', linestyle='--')
    plt.text(pulse, max(counts) * 0.95, str(pulse), color='red', rotation=90, ha='right', fontsize=8)

plt.tight_layout()
plt.savefig("cern2_pulse_overlay.png")
plt.show()

print("\n✅ Step 6 complete — pulse visualization saved as cern2_pulse_overlay.png")
```

nano step7_statistical_validation.py

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Load saved features from Step 4
with open("step4_features.pkl", "rb") as f:
    features = pickle.load(f)

unique_pdg_counts = features["unique_pdgId_count"]

# Define pulse numbers
pulse_values = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]

# Histogram data
counts, bins = np.histogram(unique_pdg_counts, bins=range(0, max(unique_pdg_counts)+2))
bin_centers = bins[:-1]
hist_dict = dict(zip(bin_centers, counts))

# Total number of events for reference
total_events = len(unique_pdg_counts)

# Calculate expected mean and std
mean = np.mean(unique_pdg_counts)
std = np.std(unique_pdg_counts)

# Monte Carlo simulation
np.random.seed(42)
simulated_counts = []
for _ in range(100000):
    sim_data = np.random.normal(loc=mean, scale=std, size=total_events)
    sim_hist, _ = np.histogram(sim_data, bins=range(0, max(unique_pdg_counts)+2))
    sim_counts = [sim_hist[val] if val < len(sim_hist) else 0 for val in pulse_values]
    simulated_counts.append(sim_counts)
simulated_counts = np.array(simulated_counts)

# Print validation results
print("\n--- Statistical Validation of Pulse Hits ---\n")
for i, pulse in enumerate(pulse_values):
    actual = hist_dict.get(pulse, 0)
    sims = simulated_counts[:, i]
    sim_mean = np.mean(sims)
    sim_std = np.std(sims)
    if sim_std > 0:
        z = (actual - sim_mean) / sim_std
        p = 1 - norm.cdf(z)
        print(f"Pulse {pulse}: Actual = {actual}, Simulated μ = {sim_mean:.2f}, σ = {sim_std:.2f}, z = {z:.2f}, p = {p:.5f}")
    else:
        print(f"Pulse {pulse}: Actual = {actual}, Sim std deviation = 0 — skipped")

# Optional save
with open("step7_stats_summary.txt", "w") as f_out:
    f_out.write("--- Statistical Validation of Pulse Hits ---\n\n")
    for i, pulse in enumerate(pulse_values):
        actual = hist_dict.get(pulse, 0)
```

```
        sims = simulated_counts[:, i]
        sim_mean = np.mean(sims)
        sim_std = np.std(sims)
        if sim_std > 0:
            z = (actual - sim_mean) / sim_std
            p = 1 - norm.cdf(z)
            f_out.write(f"Pulse {pulse}: Actual = {actual}, Expected = {sim_mean:.2f}, Z = {z:.2f}, p = {p:.5f}\n")
        else:
            f_out.write(f"Pulse {pulse}: Actual = {actual}, Expected = {sim_mean:.2f}, Z = ∞ (no variation)\n")

print("\n✅ Step 7 complete — statistical validation finished. Summary saved to 'step7_stats_summary.txt'.")
```

nano step8_montecarlo_expand.py

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
# Load pattern results from Step 5
with open("step5_patterns.pkl", "rb") as f:
    pattern_data = pickle.load(f)
# Use the count data for 'unique_pdgId_count'
pulse_hits = pattern_data["unique_pdgId_count"]
# Define pulse values
pulse_values = [7, 12, 21, 28, 42, 49, 70, 91, 112, 133, 144, 153, 343]
observed_counts = [pulse_hits.get(pulse, 0) for pulse in pulse_values]
# Monte Carlo simulation: 100,000 trials
total_events = sum(pulse_hits.values())
prob_distribution = np.ones(len(pulse_values)) / len(pulse_values)
np.random.seed(42)
simulations = np.random.multinomial(total_events, prob_distribution, size=100000)
# Compute Z-scores and p-values
print("\n--- Step 8: Monte Carlo Pulse Simulation ---\n")
for i, pulse in enumerate(pulse_values):
    actual = observed_counts[i]
    sim_vals = simulations[:, i]
    mean = np.mean(sim_vals)
    std = np.std(sim_vals)
    if std > 0:
        z = (actual - mean) / std
        p = 1 - (np.sum(sim_vals >= actual) / len(sim_vals))
        print(f"Pulse {pulse}: Actual = {actual}, Simulated Mean = {mean:.2f}, Z = {z:.2f}, p = {p:.5f}")
    else:
        print(f"Pulse {pulse}: Actual = {actual}, Simulated Mean = {mean:.2f}, Z = ∞ (no variation)")
# Optional: Plot histogram for one pulse
pulse_index = 0  # Example: pulse 7
plt.hist(simulations[:, pulse_index], bins=50, alpha=0.7, label="Simulated")
plt.axvline(observed_counts[pulse_index], color='r', linestyle='dashed', linewidth=2, label="Actual")
plt.title(f"Pulse {pulse_values[pulse_index]} — Observed vs Simulated Distribution")
plt.xlabel("Simulated Counts")
plt.ylabel("Frequency")
plt.legend()
plt.savefig("step8_pulse_simulation.png")
```

--- Statistical Validation of Pulse Hits ---

Pulse 7: Actual = 1100, Simulated μ = 1123.93, σ = 26.28, z = -0.91, p = 0.81871

Pulse 12: Actual = 0, Sim std deviation = 0 — skipped
Pulse 21: Actual = 0, Sim std deviation = 0 — skipped
Pulse 28: Actual = 0, Sim std deviation = 0 — skipped
Pulse 42: Actual = 0, Sim std deviation = 0 — skipped
Pulse 49: Actual = 0, Sim std deviation = 0 — skipped
Pulse 70: Actual = 0, Sim std deviation = 0 — skipped
Pulse 91: Actual = 0, Sim std deviation = 0 — skipped
Pulse 112: Actual = 0, Sim std deviation = 0 — skipped
Pulse 133: Actual = 0, Sim std deviation = 0 — skipped
Pulse 144: Actual = 0, Sim std deviation = 0 — skipped
Pulse 153: Actual = 0, Sim std deviation = 0 — skipped
Pulse 343: Actual = 0, Sim std deviation = 0 — skipped

--- Step 8: Monte Carlo Pulse Simulation ---

Pulse 7: Actual = 1100, Simulated Mean = 84.66, Z = 115.19, p = 1.00000
Pulse 12: Actual = 0, Simulated Mean = 84.59, Z = -9.60, p = 0.00000
Pulse 21: Actual = 0, Simulated Mean = 84.59, Z = -9.59, p = 0.00000
Pulse 28: Actual = 0, Simulated Mean = 84.63, Z = -9.60, p = 0.00000
Pulse 42: Actual = 0, Simulated Mean = 84.62, Z = -9.56, p = 0.00000
Pulse 49: Actual = 0, Simulated Mean = 84.57, Z = -9.56, p = 0.00000
Pulse 70: Actual = 0, Simulated Mean = 84.63, Z = -9.57, p = 0.00000
Pulse 91: Actual = 0, Simulated Mean = 84.67, Z = -9.59, p = 0.00000
Pulse 112: Actual = 0, Simulated Mean = 84.56, Z = -9.56, p = 0.00000
Pulse 133: Actual = 0, Simulated Mean = 84.62, Z = -9.58, p = 0.00000
Pulse 144: Actual = 0, Simulated Mean = 84.59, Z = -9.59, p = 0.00000
Pulse 153: Actual = 0, Simulated Mean = 84.65, Z = -9.59, p = 0.00000
Pulse 343: Actual = 0, Simulated Mean = 84.62, Z = -9.57, p = 0.00000