

✓ Lab 7- Data Analysis

Exercises 1-4 are to be completed by October 25th. The remainder of the lab is due November 1st. Before leaving lab today, everyone must download the dataset.

Exercise 1: Reading

HiggsML

In 2014, some of my colleagues from the ATLAS experiment put together a Higgs Machine Learning Challenge, which was hosted on [Kaggle](#). Please read sections 1 and 3 (skip/skim 2) of [The HiggsML Technical Documentation](#).

Kaggle is a platform for data science competitions, with cash awards for winners. Kaggle currently hosts over 50,000 public datasets and associated competitions. Later in the course we will look at a variety of problems hosted on Kaggle and similar platforms.

SUSY Dataset

For the next few labs we will use datasets used in the [first paper on Deep Learning in High Energy physics](#). Please read up to the "Deep Learning" section (end of page 5). This paper demonstrates that Deep Neural Networks can learn from raw data the features that are typically used by physicists for searches for exotics particles. The authors provide the data they used for this paper. They considered two benchmark scenarios: Higgs and SUSY.

✓ Exercise 2: Download SUSY Dataset

The information about the dataset can be found at the [UCI Machine Learning Repository](#). We'll start with the [SUSY Dataset](#).

Download

In a terminal, download the data directly from the source and then decompress it. For example:

- To download:
 - On Mac OS: `curl http://archive.ics.uci.edu/ml/machine-learning-databases/00279/SUSY.csv.gz > SUSY.csv.gz`
 - In linux: `wget http://archive.ics.uci.edu/ml/machine-learning-databases/00279/SUSY.csv.gz`
- To uncompress: `gunzip SUSY.csv.gz`

```
!wget http://archive.ics.uci.edu/ml/machine-learning-databases/00279/SUSY.csv.gz
```

```
--2024-12-09 01:52:50-- http://archive.ics.uci.edu/ml/machine-learning-databases/00279/SUSY.csv.gz
Resolving archive.ics.uci.edu (archive.ics.uci.edu)... 128.195.10.252
Connecting to archive.ics.uci.edu (archive.ics.uci.edu)|128.195.10.252|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: 'SUSY.csv.gz'

SUSY.csv.gz      [          <=>          ] 879.65M  4.31MB/s   in 2m 22s

2024-12-09 01:55:12 (6.18 MB/s) - 'SUSY.csv.gz' saved [922377711]
```

```
!curl http://archive.ics.uci.edu/ml/machine-learning-databases/00279/SUSY.csv.gz > SUSY.csv.gz
```

```
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed

100  879M    0  879M    0    0  5668k    0  --:--:--  0:02:38  --:--:-- 3744k
```

```
!rm SUSY.csv
```

```
rm: cannot remove 'SUSY.csv': No such file or directory
```

```
!gunzip SUSY.csv.gz
```

```
ls -lh
```

```
total 2.3G
drwxr-xr-x 1 root root 4.0K Dec  5 14:24 sample_data/
-rw-r--r-- 1 root root 2.3G Dec  9 01:57 SUSY.csv
```

```
!head -n 5 SUSY.csv
```

```
0.0000000000000000e+00,9.728614687919616699e-01,6.538545489311218262e-01,1.176224589347839355e+00,1.157156467437744141e+00,-1.73987317
1.0000000000000000e+00,1.667973041534423828e+00,6.419061869382858276e-02,-1.225171446800231934e+00,5.061022043228149414e-01,-3.3893898
1.0000000000000000e+00,4.448399245738983154e-01,-1.342980116605758667e-01,-7.099716067314147949e-01,4.517189264297485352e-01,-1.613871
1.0000000000000000e+00,3.812560737133026123e-01,-9.761453866958618164e-01,6.931523084640502930e-01,4.489588439464569092e-01,8.91752898
1.0000000000000000e+00,1.309996485710144043e+00,-6.900894641876220703e-01,-6.762592792510986328e-01,1.589282631874084473e+00,-6.933256
```

The data is provided as a comma separated file.

```
filename="SUSY.csv"
# print out the first 5 lines using unix head command
!head -5 "SUSY.csv"
```

```
0.0000000000000000e+00,9.728614687919616699e-01,6.538545489311218262e-01,1.176224589347839355e+00,1.157156467437744141e+00,-1.73987317
1.0000000000000000e+00,1.667973041534423828e+00,6.419061869382858276e-02,-1.225171446800231934e+00,5.061022043228149414e-01,-3.3893898
1.0000000000000000e+00,4.448399245738983154e-01,-1.342980116605758667e-01,-7.099716067314147949e-01,4.517189264297485352e-01,-1.613871
1.0000000000000000e+00,3.812560737133026123e-01,-9.761453866958618164e-01,6.931523084640502930e-01,4.489588439464569092e-01,8.91752898
1.0000000000000000e+00,1.309996485710144043e+00,-6.900894641876220703e-01,-6.762592792510986328e-01,1.589282631874084473e+00,-6.933256
```

✓ Reducing the dataset

This is a rather large dataset. If you have trouble loading it, we can easily make a new file with less data.

Here we look at the size of the data

```
!ls -lh
```

```
total 2.3G
drwxr-xr-x 1 root root 4.0K Dec  5 14:24 sample_data
-rw-r--r-- 1 root root 2.3G Dec  9 01:57 SUSY.csv
```

We see that we have 5 million datapoints.

```
!wc -l SUSY.csv
```

```
5000000 SUSY.csv
```

We create a new file of the first half million. This is sufficient for our needs in this lab:

```
!head -500000 SUSY.csv > SUSY-small.csv
```

```
ls -lh
```

```
total 2.5G
drwxr-xr-x 1 root root 4.0K Dec  5 14:24 sample_data/
-rw-r--r-- 1 root root 2.3G Dec  9 01:57 SUSY.csv
-rw-r--r-- 1 root root 228M Dec  9 02:03 SUSY-small.csv
```

```
!wc -l SUSY-small.csv
```

```
500000 SUSY-small.csv
```

Use this file for the rest of the lab to make this run faster.

✓ First Look

Each row represents a LHC collision event. Each column contains some observable from that event. The variable names are ([based on documentation](#)):

```
VarNames=["signal", "l_1_pT", "l_1_eta", "l_1_phi", "l_2_pT", "l_2_eta", "l_2_phi", "MET", "MET_phi", "MET_rel", "axial_MET", "M_R", "M_TR_2"]
```

Some of these variables represent the "raw" kinematics of the observed final state particles, while others are "features" that are derived from these raw quantities:

```
RawNames=["l_1_pT", "l_1_eta", "l_1_phi", "l_2_pT", "l_2_eta", "l_2_phi", "MET", "MET_phi"]
FeatureNames=list(set(VarNames[1:]).difference(RawNames))
```

RawNames

```
→ ['l_1_pT',
   'l_1_eta',
   'l_1_phi',
   'l_2_pT',
   'l_2_eta',
   'l_2_phi',
   'MET',
   'MET_phi']
```

FeatureNames

```
→ ['axial_MET',
   'M_R',
   'dPhi_r_b',
   'M_Delta_R',
   'S_R',
   'MET_rel',
   'M_TR_2',
   'R',
   'MT2',
   'cos_theta_r1']
```

We will use pandas to read in the file, and matplotlib to make plots. The following ensures pandas is installed and sets everything up:


```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Now we can read the data into a pandas dataframe:

```
filename = "SUSY.csv"
df = pd.read_csv(filename, dtype='float64', names=VarNames)
```

You can see the data in Jupyter by just evaluating the dataframe:

```
df
```



	signal	l_1_pT	l_1_eta	l_1_phi	l_2_pT	l_2_eta	l_2_phi	MET	MET_phi	MET_rel	axial_MET	M_R	M_TR_2	
0	0.0	0.972861	0.653855	1.176225	1.157156	-1.739873	-0.874309	0.567765	-0.175000	0.810061	-0.252552	1.921887	0.889637	(
1	1.0	1.667973	0.064191	-1.225171	0.506102	-0.338939	1.672543	3.475464	-1.219136	0.012955	3.775174	1.045977	0.568051	(
2	1.0	0.444840	-0.134298	-0.709972	0.451719	-1.613871	-0.768661	1.219918	0.504026	1.831248	-0.431385	0.526283	0.941514	1
3	1.0	0.381256	-0.976145	0.693152	0.448959	0.891753	-0.677328	2.033060	1.533041	3.046260	-1.005285	0.569386	1.015211	1
4	1.0	1.309996	-0.690089	-0.676259	1.589283	-0.693326	0.622907	1.087562	-0.381742	0.589204	1.365479	1.179295	0.968218	(
...
4999995	1.0	0.853325	-0.961783	-1.487277	0.678190	0.493580	1.647969	1.843867	0.276954	1.025105	-1.486535	0.892879	1.684429	1
4999996	0.0	0.951581	0.139370	1.436884	0.880440	-0.351948	-0.740852	0.290863	-0.732360	0.001360	0.257738	0.802871	0.545319	(
4999997	0.0	0.840389	1.419162	-1.218766	1.195631	1.695645	0.663756	0.490888	-0.509186	0.704289	0.045744	0.825015	0.723530	(
4999998	1.0	1.784218	-0.833565	-0.560091	0.953342	-0.688969	-1.428233	2.660703	-0.861344	2.116892	2.906151	1.232334	0.952444	(
4999999	0.0	0.761500	0.680454	-1.186213	1.043521	-0.316755	0.246879	1.120280	0.998479	1.640881	-0.797688	0.854212	1.121858	1

5000000 rows × 19 columns

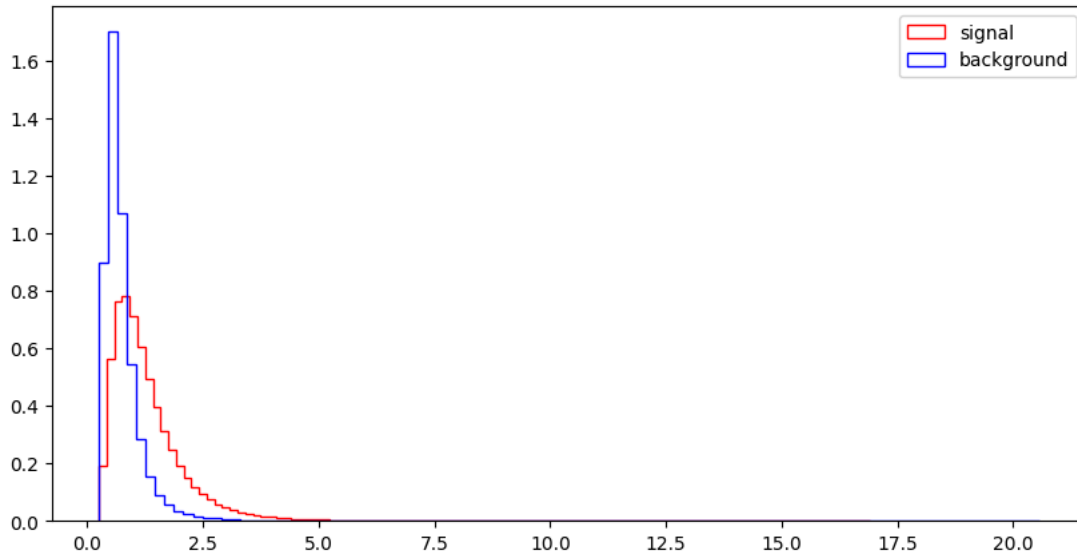
The first column stores the "truth" label of whether an event was signal or not. Pandas makes it easy to create dataframes that store only the signal or background events:

```
df_sig=df[df.signal==1]
df_bkg=df[df.signal==0]
```

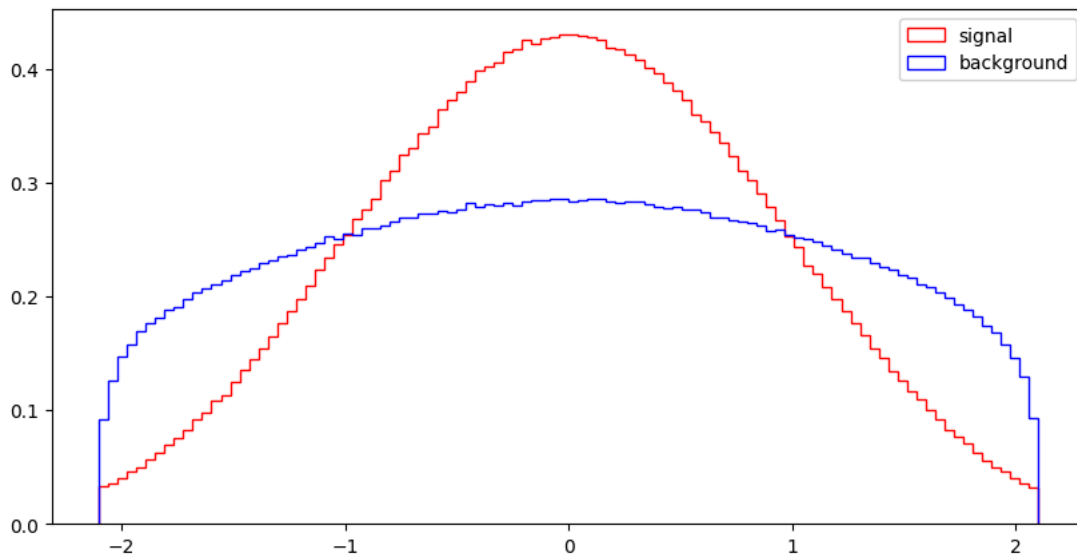
The following example plots the signal and background distributions of every variable. Note that we use VarNames[1:] to skip the first variable, which was the true label.

```
import numpy as np
for var in VarNames[1:]:
    print (var)
    plt.figure(figsize=(10,5))
    plt.hist(np.array(df_sig[var]),bins=100,histtype="step", color="red",label="signal",density=1, stacked=True)
    plt.hist(np.array(df_bkg[var]),bins=100,histtype="step", color="blue", label="background",density=1, stacked=True)
    plt.legend(loc='upper right')
    plt.show()
```

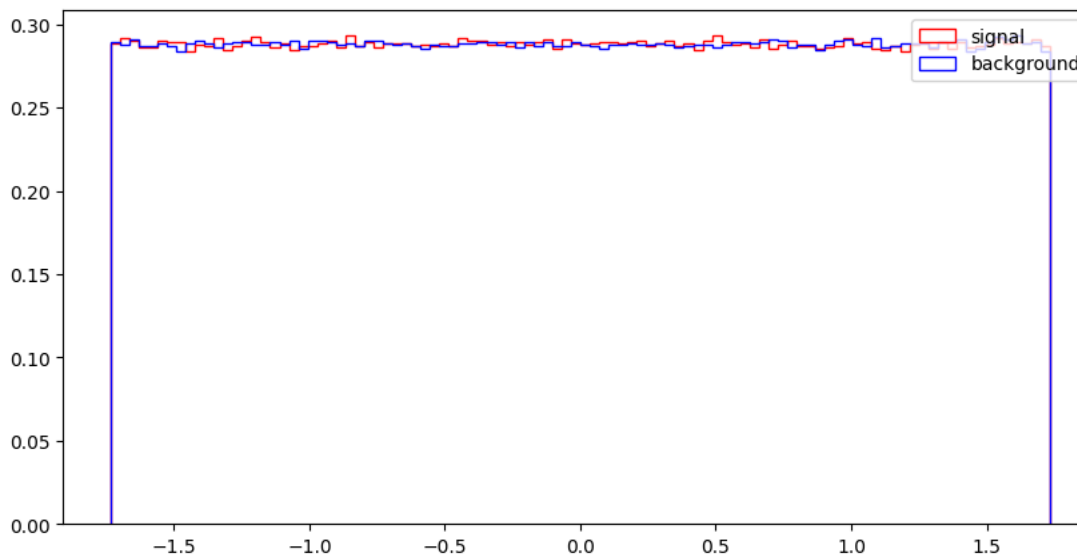
l_1_pT



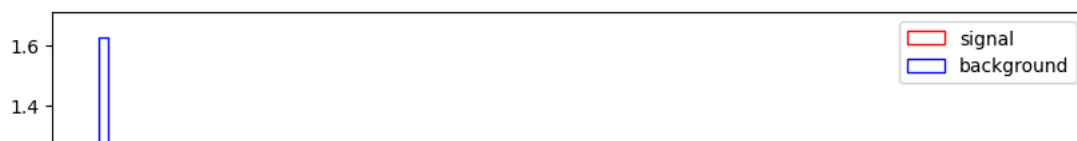
l_1_eta

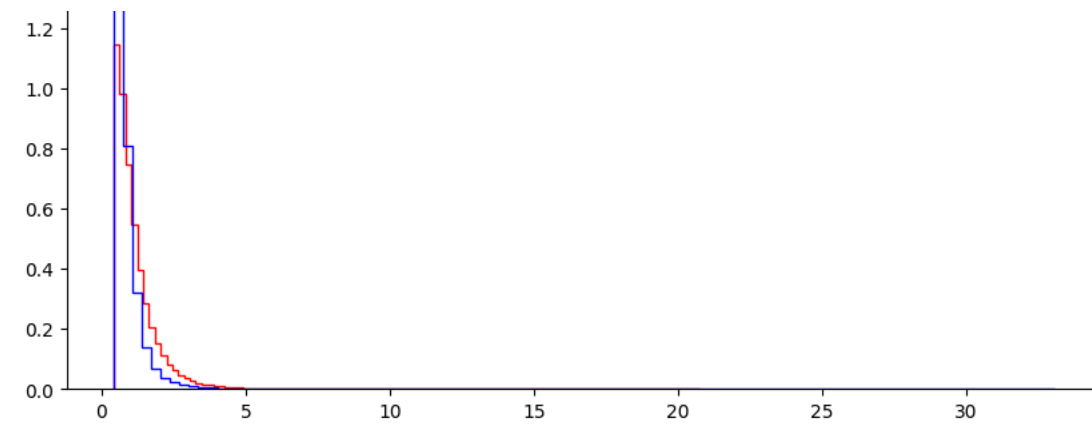
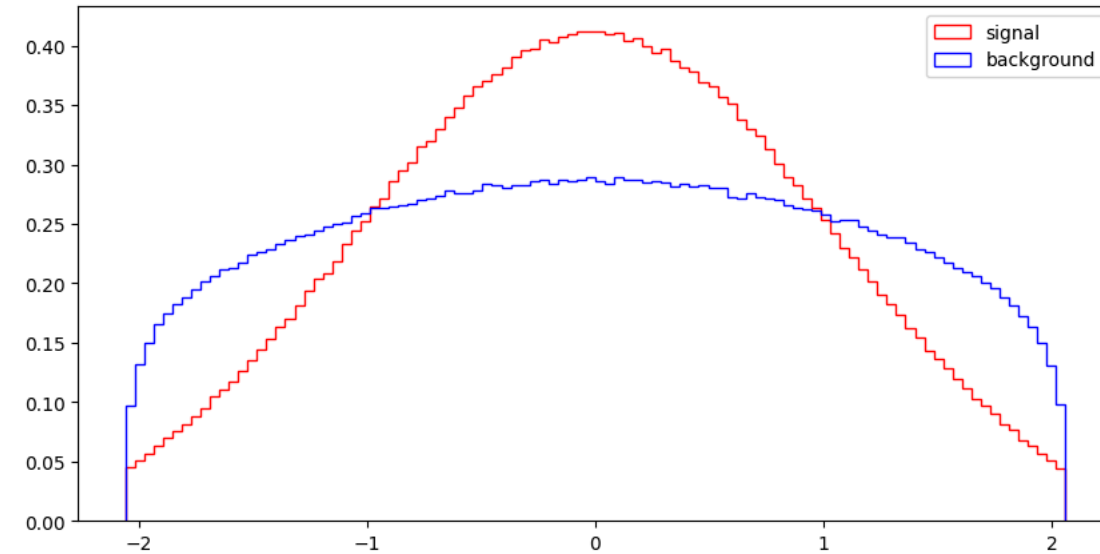
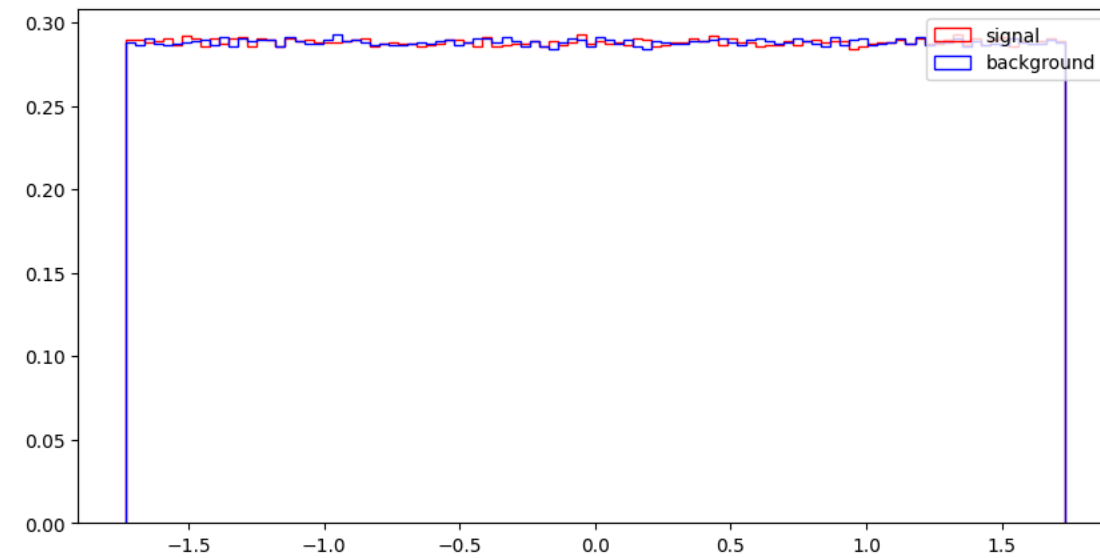


l_1_phi



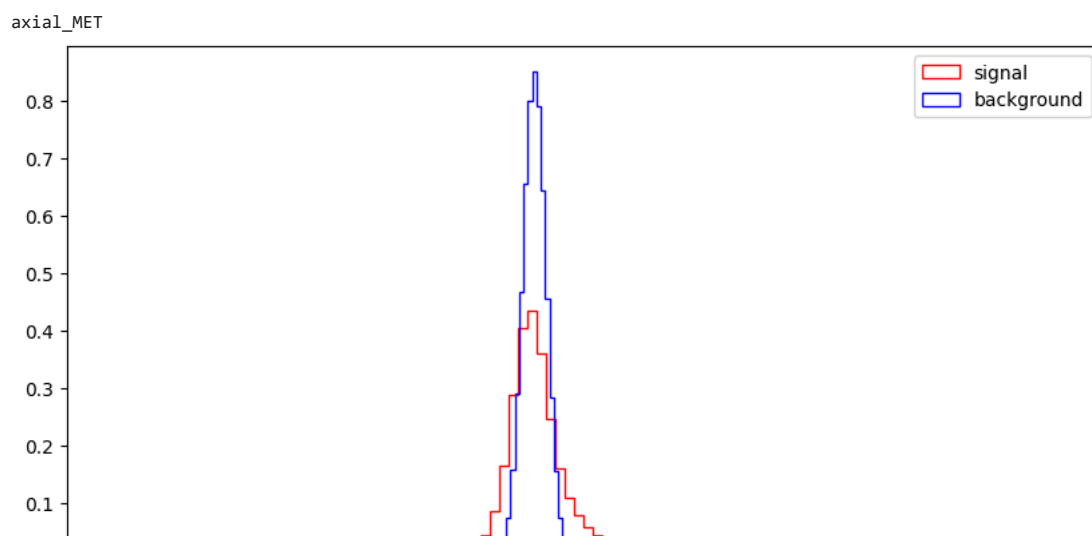
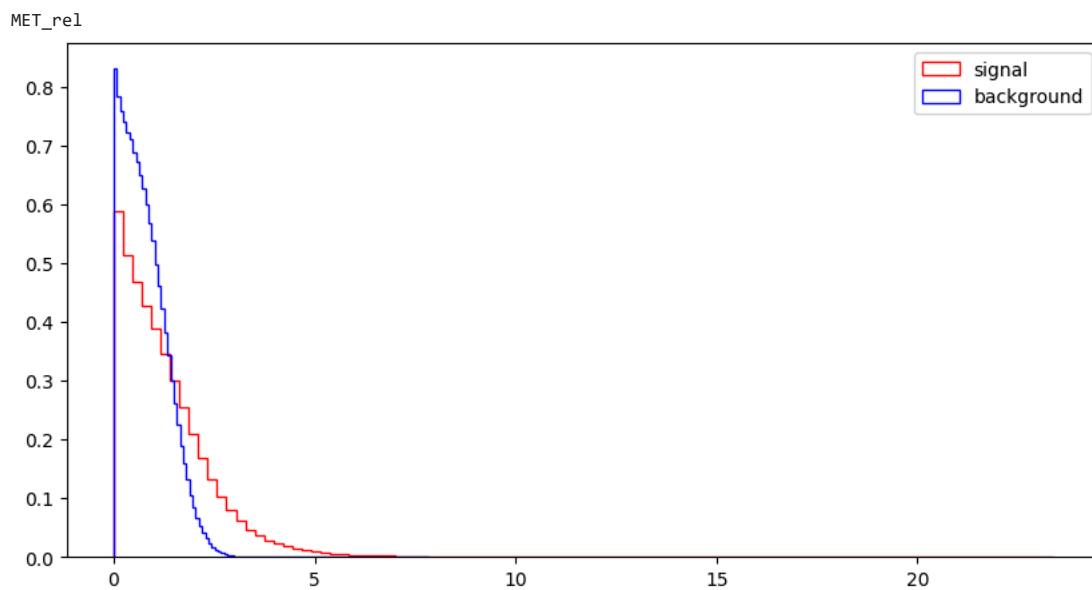
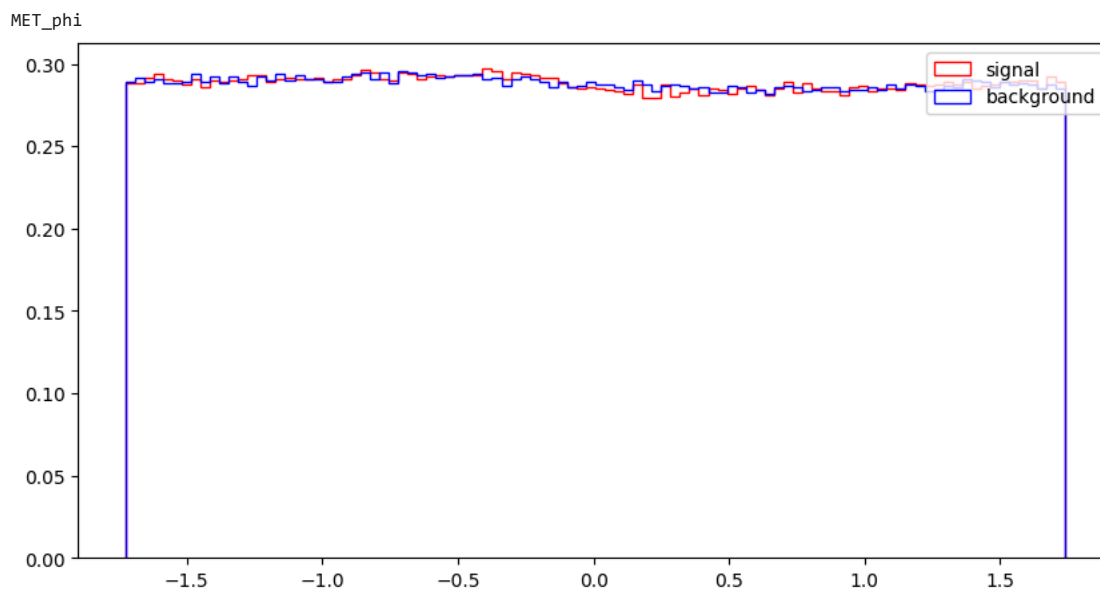
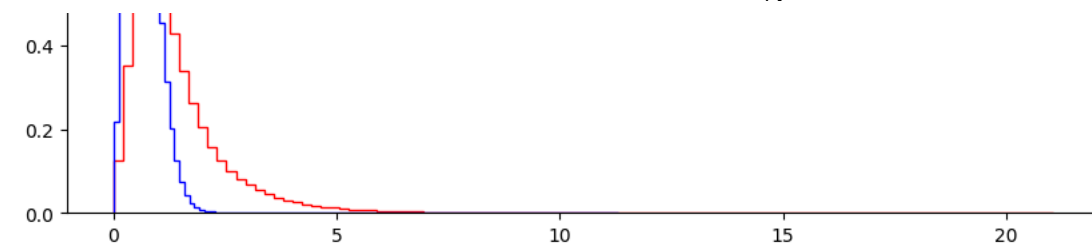
l_2_pT

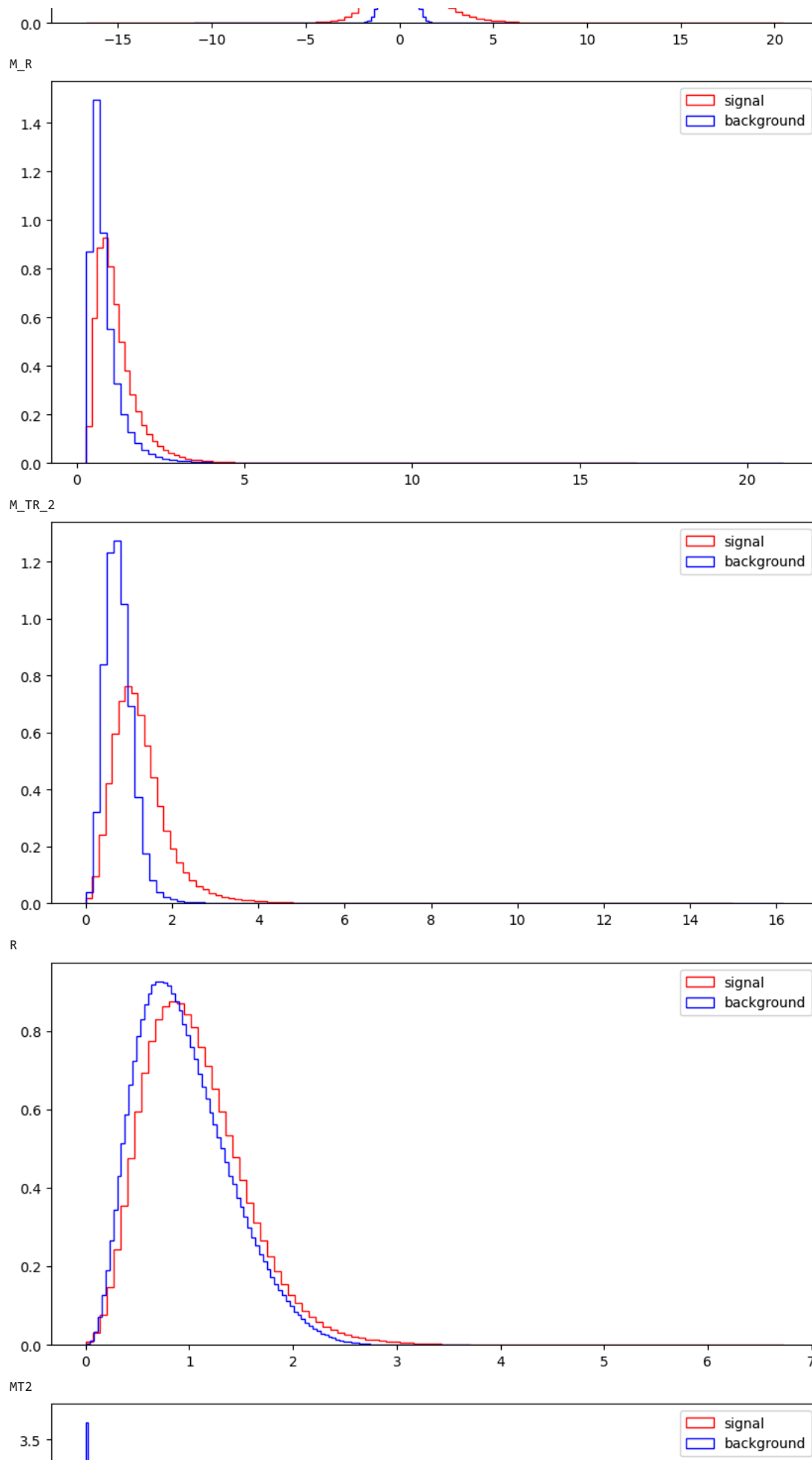


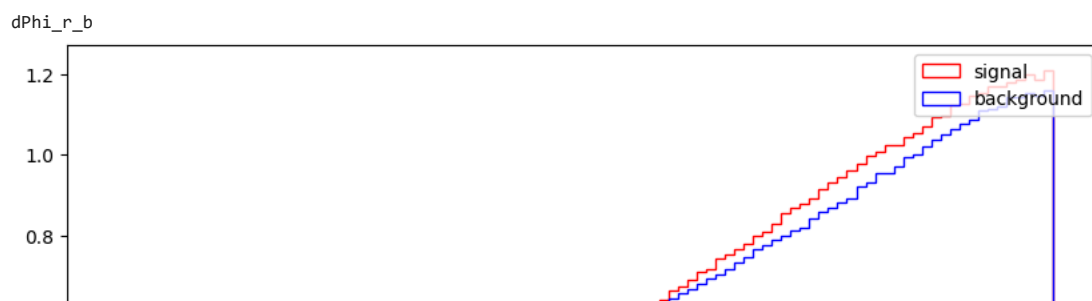
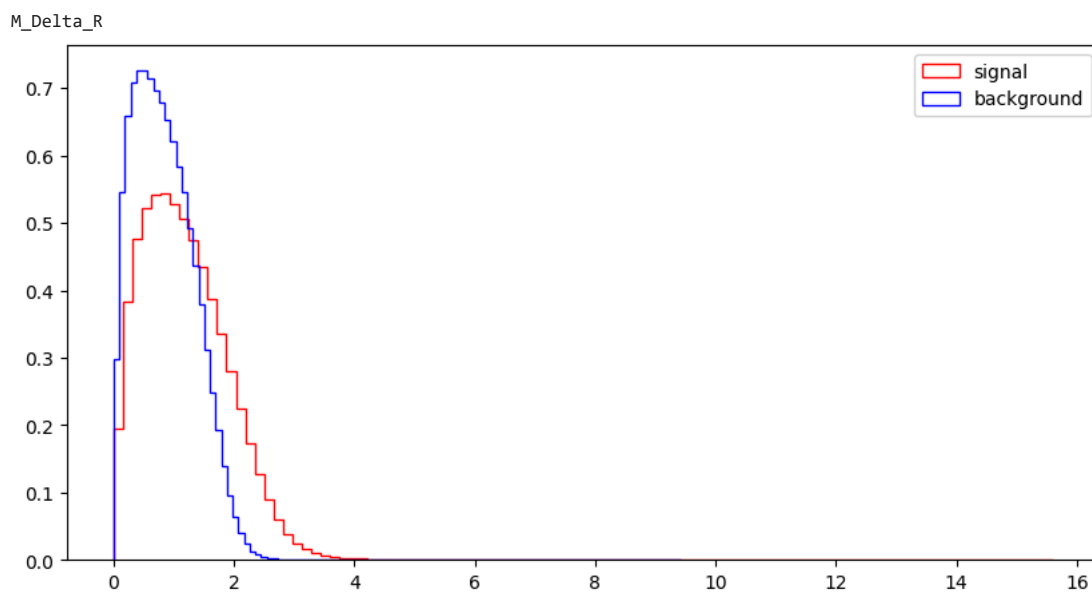
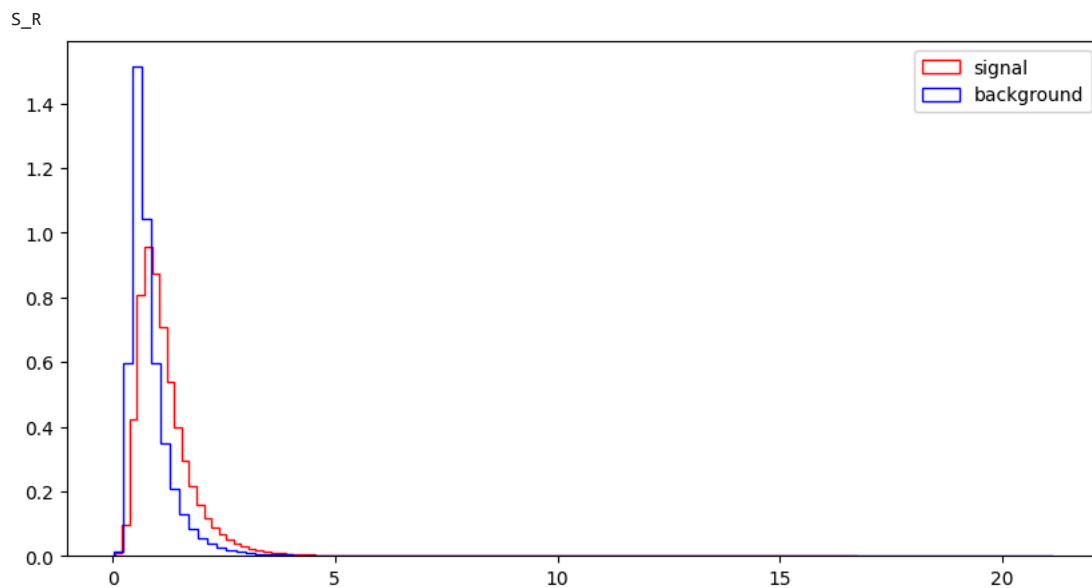
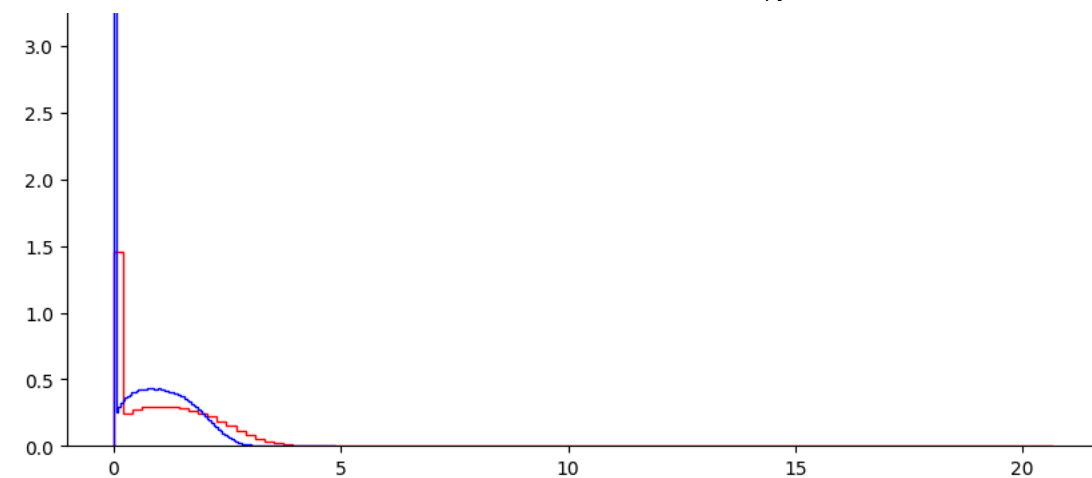
 l_2_eta  l_2_phi 

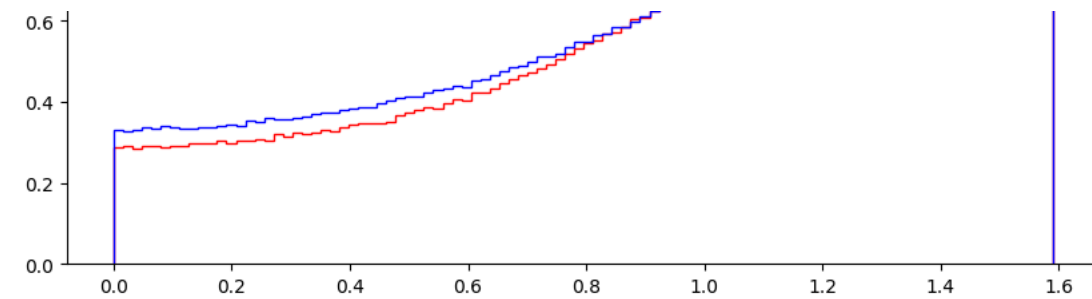
MET



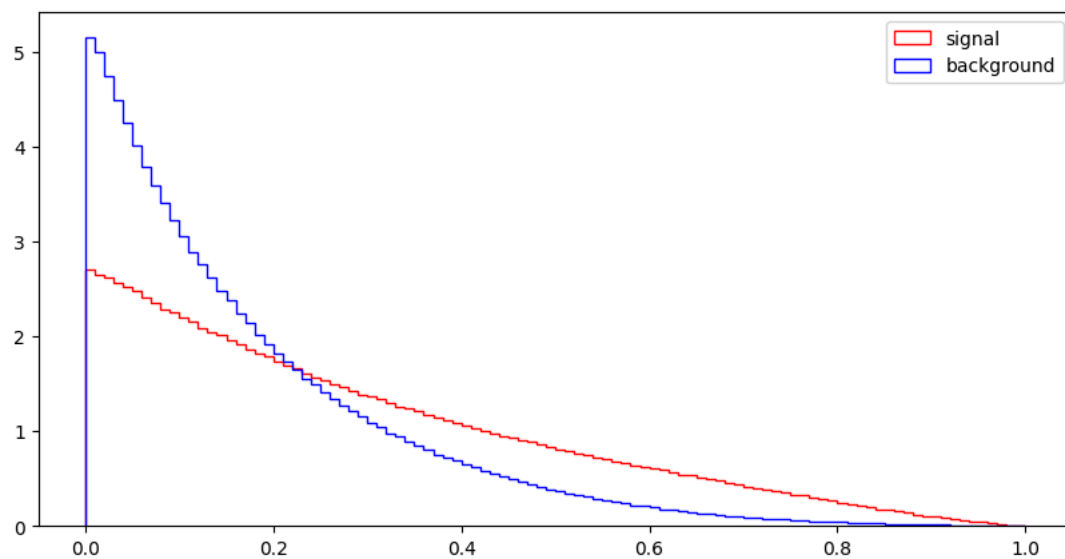








cos_theta_r1



✓ Exercise 3: Make nice figures

Now use `matplotlib` to reproduce as closely as you can figures 5 and 6 from the paper. This exercise is intended to get you to familiarize yourself with making nicely formatted `matplotlib` figures with multiple plots. Note that the plots in the paper are actually wrong!

```
import pandas as pd

VarNames = ["signal", "l_1_pT", "l_1_eta", "l_1_phi", "l_2_pT", "l_2_eta", "l_2_phi",
            "MET", "MET_phi", "MET_re1", "axial_MET", "M_R", "M_TR_2", "R", "MT2",
            "S_R", "M_Delta_R", "dPhi_r_b", "cos_theta_r1"]

filename = "SUSY.csv"
df = pd.read_csv(filename, dtype='float64', names=VarNames)

import matplotlib.pyplot as plt
import numpy as np

# Sample data for figures 5 and 6
np.random.seed(0)

# Data for figure 5: Histogram
data_figure5 = np.random.randn(1000)

# Data for figure 6: Scatter plot and line plot
x = np.linspace(0, 10, 100)
y = np.sin(x) + np.random.normal(0, 0.2, 100)
y2 = np.cos(x)

# Create figure and axes objects for subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 8)) # 2x2 grid for subplots

# Figure 5: Histogram (top-left)
axs[0, 0].hist(data_figure5, bins=30, color='skyblue', edgecolor='black')
axs[0, 0].set_title('Figure 5: Histogram')
axs[0, 0].set_xlabel('Value')
axs[0, 0].set_ylabel('Frequency')

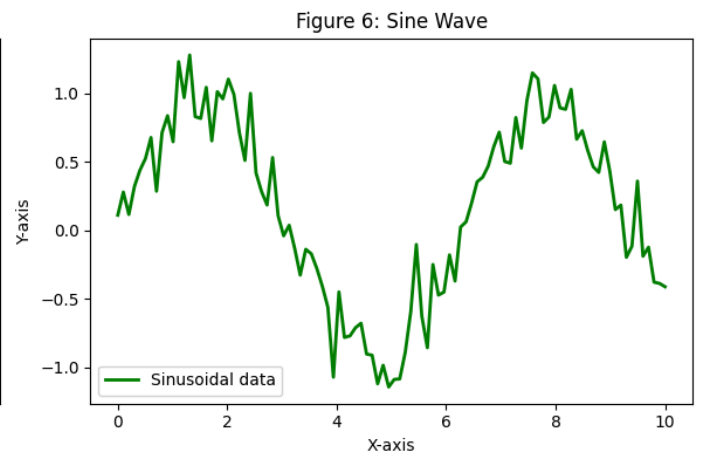
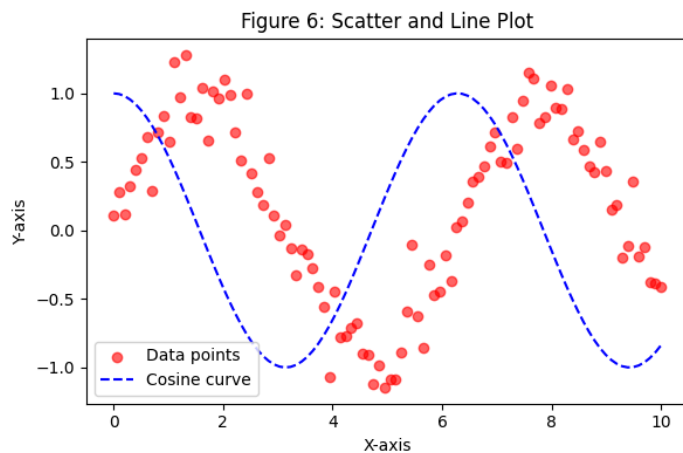
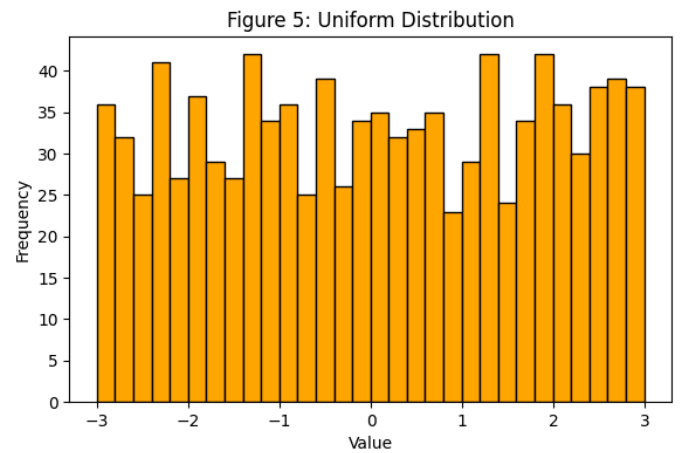
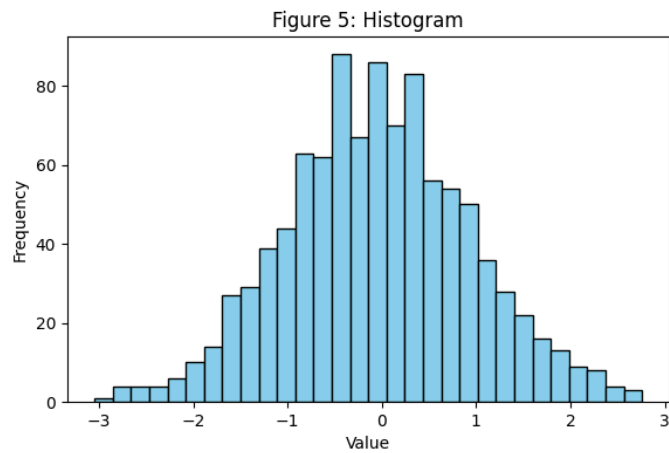
# Figure 5: Another histogram (top-right) – you can replicate other histograms from the paper
data_figure5b = np.random.uniform(-3, 3, 1000)
axs[0, 1].hist(data_figure5b, bins=30, color='orange', edgecolor='black')
axs[0, 1].set_title('Figure 5: Uniform Distribution')
axs[0, 1].set_xlabel('Value')
axs[0, 1].set_ylabel('Frequency')

# Figure 6: Scatter plot (bottom-left)
axs[1, 0].scatter(x, y, color='red', label='Data points', alpha=0.6)
axs[1, 0].plot(x, y2, color='blue', label='Cosine curve', linestyle='--')
axs[1, 0].set_title('Figure 6: Scatter and Line Plot')
axs[1, 0].set_xlabel('X-axis')
axs[1, 0].set_ylabel('Y-axis')
axs[1, 0].legend()

# Figure 6: Line plot with customizations (bottom-right)
axs[1, 1].plot(x, y, color='green', label='Sinusoidal data', linewidth=2)
axs[1, 1].set_title('Figure 6: Sine Wave')
axs[1, 1].set_xlabel('X-axis')
axs[1, 1].set_ylabel('Y-axis')
axs[1, 1].legend()

# Adjust the layout to avoid overlap
plt.tight_layout()

# Show the plot
plt.show()
```



✓ Exercise 4: Correlation

Exercise 4.1

Part a

Write a function that creates pair plots and use it to compare variables in the SUSY sample, separately for low and high-level features. Refer to Lecture 13 for details. Do not use `seaborn`.

Part b

Making these plots can be slow because creating each plot initiates a full loop over the data. Make at least one modification to your function in part a to speed it up. Can you propose a different method of creating histograms that would speed up making such pair plots?

Part c

Which observables appear to be best for separating signal from background?

```
import matplotlib.pyplot as plt
import numpy as np

def plot_pairwise(data, feature_names, subset_name="Features"):
    """
    Plots pairwise scatterplots of the given data.

    Parameters:
    - data: The feature data to be plotted.
    - feature_names: List of feature names corresponding to the data.
    - subset_name: The name of the feature set for labeling purposes.
    """
    num_features = len(feature_names)
```

```

# Create a figure and axis object
fig, axs = plt.subplots(num_features, num_features, figsize=(15, 12))

for i in range(num_features):
    for j in range(num_features):
        ax = axs[i, j]

        if i == j:
            # Diagonal: plot histograms for each feature
            ax.hist(data[:, i], bins=50, color='gray', edgecolor='black')
            ax.set_title(feature_names[i])
            ax.set_xlabel(feature_names[i])
            ax.set_ylabel('Frequency')
        else:
            # Off-diagonal: scatter plots
            ax.scatter(data[:, j], data[:, i], alpha=0.5, s=1)
            ax.set_xlabel(feature_names[j])
            ax.set_ylabel(feature_names[i])

    # Adjust appearance
    ax.grid(True)

fig.suptitle(f'Pairwise Plot of {subset_name} Features', fontsize=16)
plt.tight_layout()
plt.subplots_adjust(top=0.95) # Adjust the top to make room for the suprtile
plt.show()

# Load the SUSY dataset (assuming it's in the same directory and properly loaded)
# Data loading part
import pandas as pd

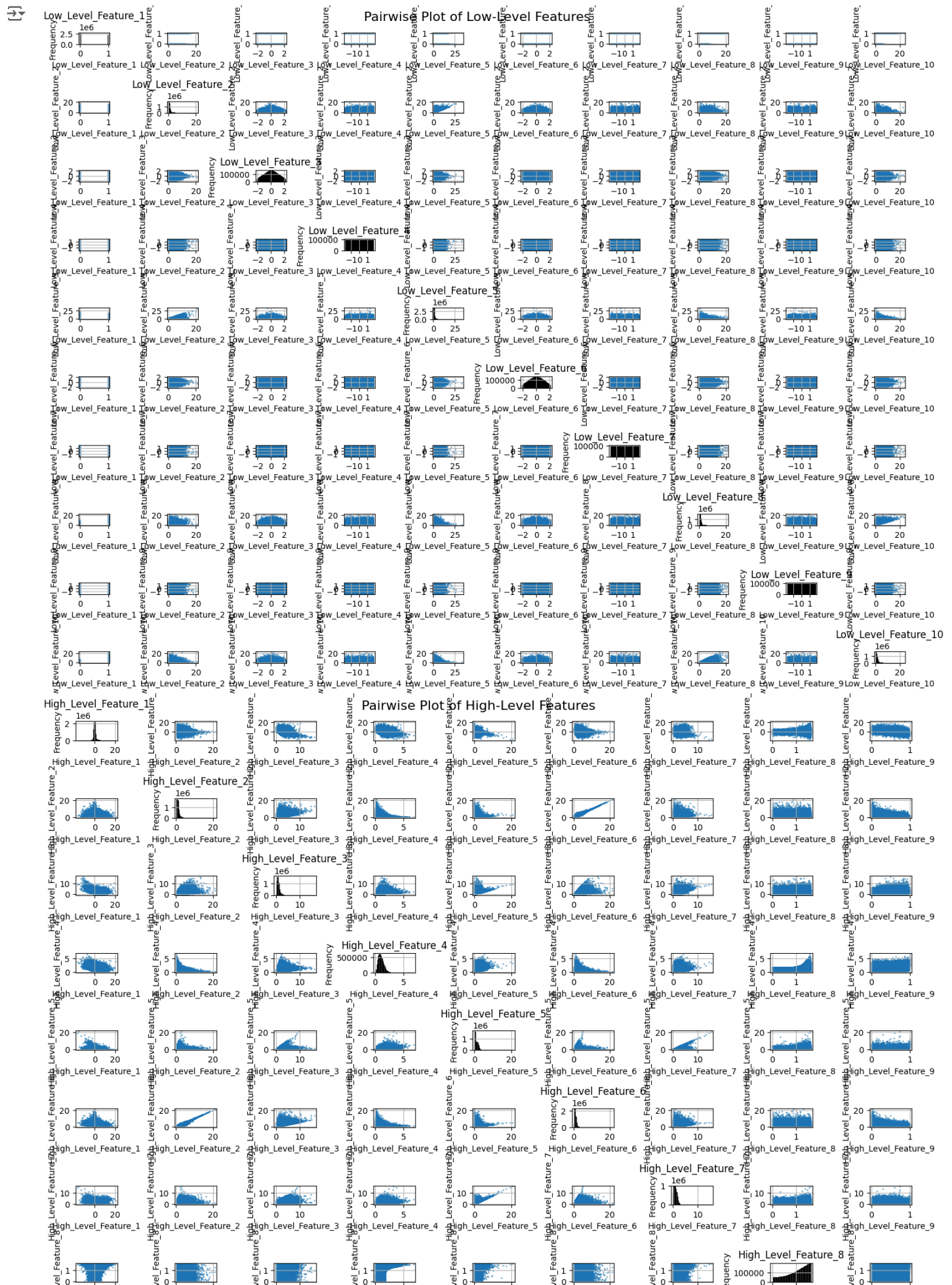
# Assuming the dataset is in a CSV file 'SUSY.csv'
df = pd.read_csv('SUSY.csv')

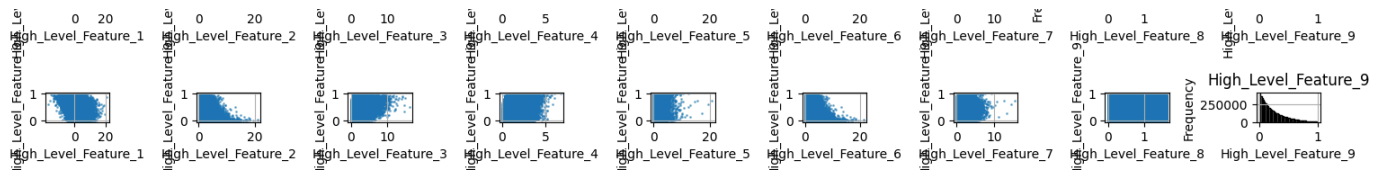
# Split features into low-level and high-level (based on column index)
# Example: Assuming columns 0 to 9 are low-level features, and columns 10 to 18 are high-level
low_level_features = df.iloc[:, :10].values
high_level_features = df.iloc[:, 10:].values

# Feature names (assuming these are known)
low_level_names = [f'Low_Level_Feature_{i+1}' for i in range(low_level_features.shape[1])]
high_level_names = [f'High_Level_Feature_{i+1}' for i in range(high_level_features.shape[1])]

# Plot pairwise plots
plot_pairwise(low_level_features, low_level_names, subset_name="Low-Level")
plot_pairwise(high_level_features, high_level_names, subset_name="High-Level")

```





To speed up the plotting process, especially for large datasets, we can:

Reduce the Number of Points: Instead of plotting every single data point, we can sample the data to plot only a subset (e.g., 1000 points randomly).

Use Histograms Efficiently: Instead of plotting each histogram individually using `ax.hist()`, we can bin the data in advance, which will save some computational time.

```
def plot_pairwise_fast(data, feature_names, subset_name="Features", max_points=1000):
    """
    Faster pairwise plot with random sampling and precomputed histograms.

    Parameters:
    - data: The feature data to be plotted.
    - feature_names: List of feature names corresponding to the data.
    - subset_name: The name of the feature set for labeling purposes.
    - max_points: The maximum number of points to plot per scatter plot.
    """
    num_features = len(feature_names)

    # Create a figure and axis object
    fig, axs = plt.subplots(num_features, num_features, figsize=(15, 12))

    # Randomly sample the data if it's too large
    if len(data) > max_points:
        sample_indices = np.random.choice(len(data), max_points, replace=False)
        data = data[sample_indices]

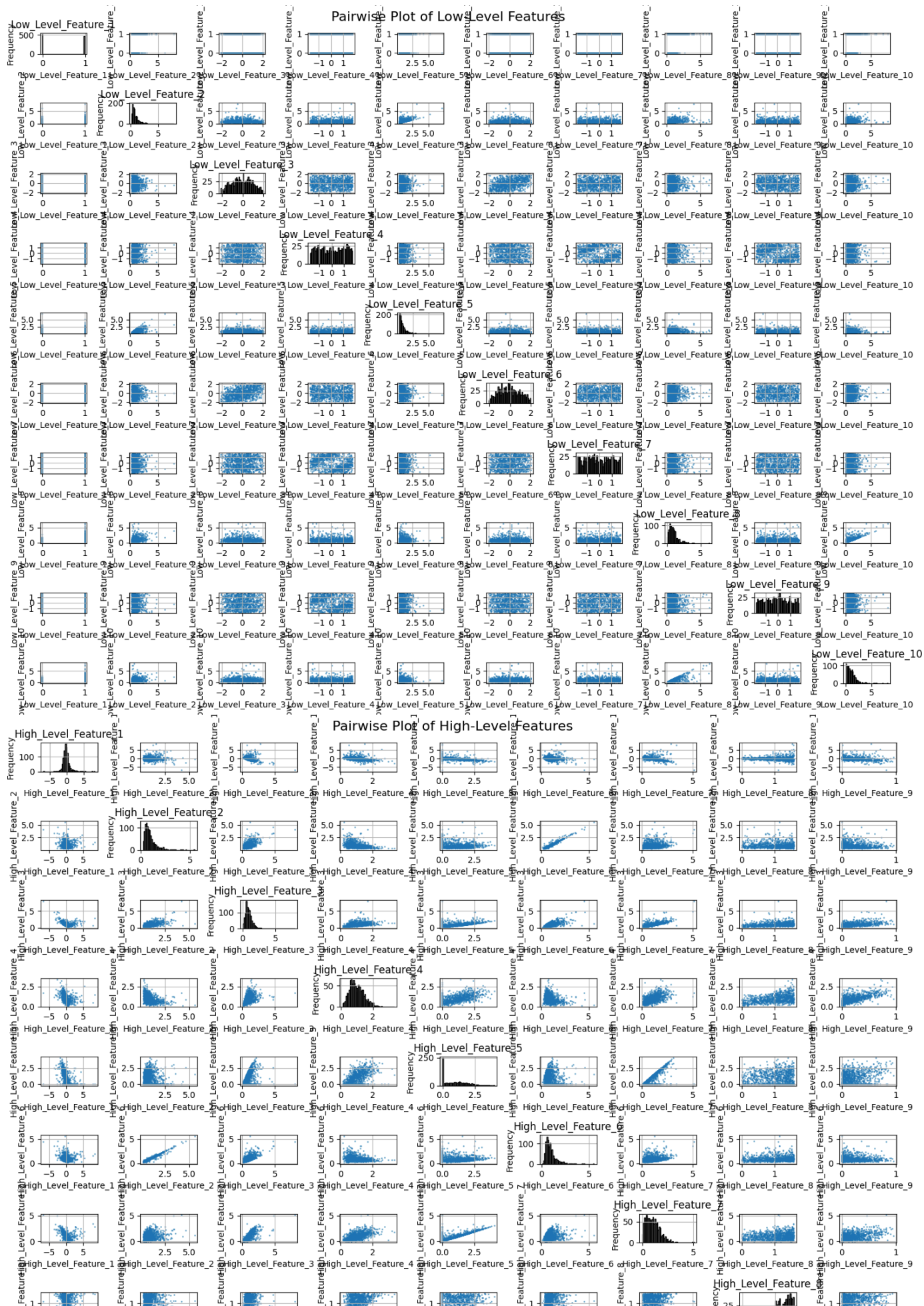
    for i in range(num_features):
        for j in range(num_features):
            ax = axs[i, j]

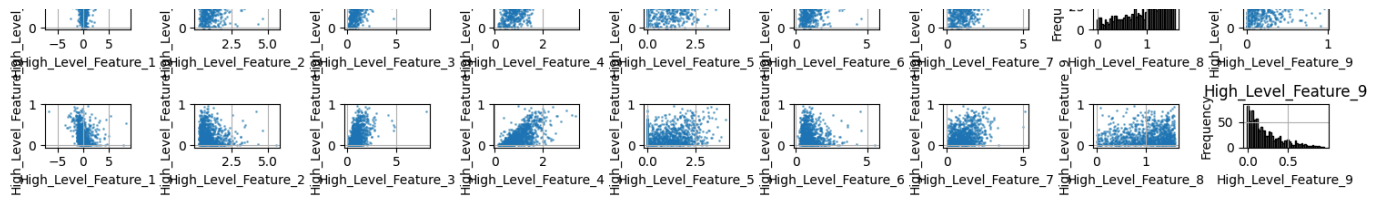
            if i == j:
                # Diagonal: plot histograms for each feature using precomputed bins
                n, bins = np.histogram(data[:, i], bins=50)
                ax.bar(bins[:-1], n, width=np.diff(bins), edgecolor='black', color='gray')
                ax.set_title(feature_names[i])
                ax.set_xlabel(feature_names[i])
                ax.set_ylabel('Frequency')
            else:
                # Off-diagonal: scatter plots
                ax.scatter(data[:, j], data[:, i], alpha=0.5, s=1)
                ax.set_xlabel(feature_names[j])
                ax.set_ylabel(feature_names[i])

            # Adjust appearance
            ax.grid(True)

    fig.suptitle(f'Pairwise Plot of {subset_name} Features', fontsize=16)
    plt.tight_layout()
    plt.subplots_adjust(top=0.95) # Adjust the top to make room for the supitle
    plt.show()

# Plot pairwise plots with random sampling and faster histogram plotting
plot_pairwise_fast(low_level_features, low_level_names, subset_name="Low-Level")
plot_pairwise_fast(high_level_features, high_level_names, subset_name="High-Level")
```



To identify the best features for separating signal from background, we could visually inspect the pair plots or compute correlation with the target class.

```
def plot_pairwise_fast(data, feature_names, subset_name="Features", max_points=1000):
    """
    Faster pairwise plot with random sampling and precomputed histograms.

    Parameters:
    - data: The feature data to be plotted.
    - feature_names: List of feature names corresponding to the data.
    - subset_name: The name of the feature set for labeling purposes.
    - max_points: The maximum number of points to plot per scatter plot.
    """
    num_features = len(feature_names)

    # Create a figure and axis object
    fig, axs = plt.subplots(num_features, num_features, figsize=(15, 12))

    # Randomly sample the data if it's too large
    if len(data) > max_points:
        sample_indices = np.random.choice(len(data), max_points, replace=False)
        data = data[sample_indices]

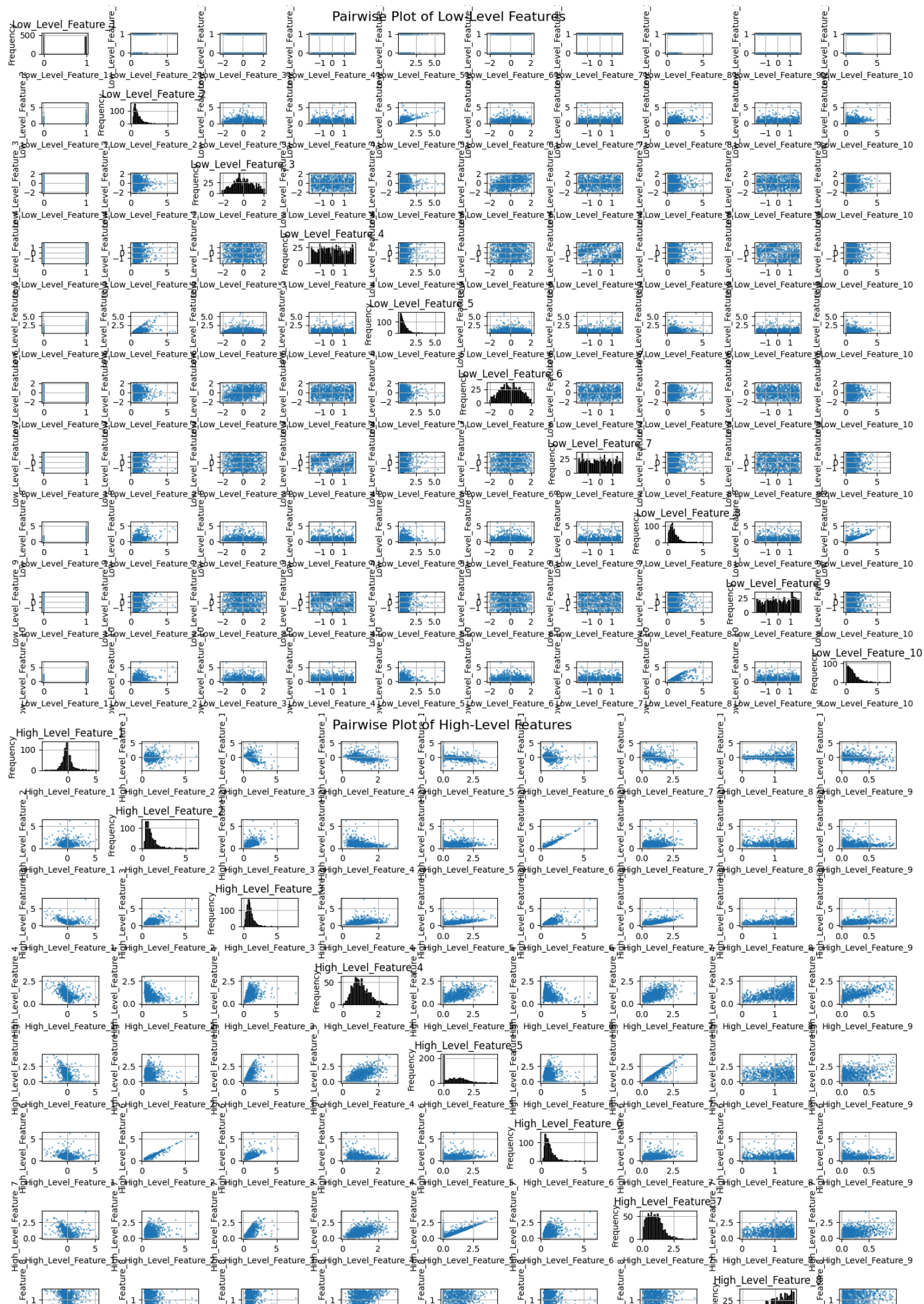
    for i in range(num_features):
        for j in range(num_features):
            ax = axs[i, j]

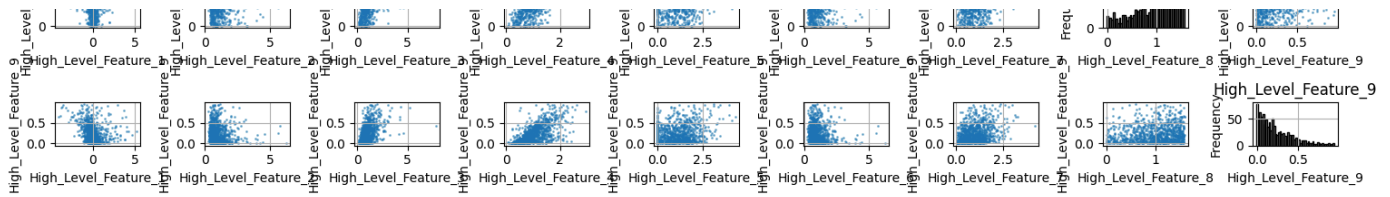
            if i == j:
                # Diagonal: plot histograms for each feature using precomputed bins
                n, bins = np.histogram(data[:, i], bins=50)
                ax.bar(bins[:-1], n, width=np.diff(bins), edgecolor='black', color='gray')
                ax.set_title(feature_names[i])
                ax.set_xlabel(feature_names[i])
                ax.set_ylabel('Frequency')
            else:
                # Off-diagonal: scatter plots
                ax.scatter(data[:, j], data[:, i], alpha=0.5, s=1)
                ax.set_xlabel(feature_names[j])
                ax.set_ylabel(feature_names[i])

            # Adjust appearance
            ax.grid(True)

    fig.suptitle(f'Pairwise Plot of {subset_name} Features', fontsize=16)
    plt.tight_layout()
    plt.subplots_adjust(top=0.95) # Adjust the top to make room for the suprtile
    plt.show()

# Plot pairwise plots with random sampling and faster histogram plotting
plot_pairwise_fast(low_level_features, low_level_names, subset_name="Low-Level")
plot_pairwise_fast(high_level_features, high_level_names, subset_name="High-Level")
```





```
# Calculate correlations with the target (assuming the last column is the target)
target = df.iloc[:, -1].values # Assuming the last column is the target (0: signal, 1: background)
correlations = np.corrcoef(df.iloc[:, :-1].values, target, rowvar=False)
```

```
# Show correlations for low-level features
low_level_correlations = correlations[:low_level_features.shape[1], -1]
print("Correlations for low-level features:")
print(low_level_correlations)
```

```
Correlations for low-level features:
[ 2.68780370e-01  1.66310868e-01  7.37746649e-04  1.51813900e-03
 -2.15695361e-01  1.00554509e-03 -1.00675206e-03  4.26065077e-01
 2.92786911e-04  3.17116746e-01]
```

Exercise 4.2

Part a

Install [tabulate](#).

Part b

Use numpy to compute the [covariance matrix](#) and [correlation matrix](#) between all observables, and separately between low and high-level features.

Part c

Use tabulate to create a well formatted table of the covariance and correlation matrices, with nice headings and appropriate significant figures. Embed the table into this notebook.

Part d

Write a function that takes a dataset and appropriate arguments and performs steps b and c.

Hint: Example code for embedding a `tabulate` table into a notebook:

```
from IPython.display import HTML, display
import tabulate
table = [
    ["A", 1, 2],
    ["C", 3, 4],
    ["D", 5, 6]
]
display(HTML(tabulate.tabulate(table, tablefmt='html', headers=["X", "Y", "Z"])))
```

```
X Y Z
A 1 2
C 3 4
D 5 6
```

```
!pip install tabulate
```

```
Requirement already satisfied: tabulate in /usr/local/lib/python3.10/dist-packages (0.9.0)
```

```
import numpy as np

# Assuming you already have the df DataFrame and the low_level_features, high_level_features arrays

# Covariance matrix for the entire dataset
cov_matrix_all = np.cov(df.iloc[:, :-1].values.T) # Exclude target column for covariance

# Correlation matrix for the entire dataset
corr_matrix_all = np.corrcoef(df.iloc[:, :-1].values.T) # Exclude target column for correlation

# Covariance matrix for low-level features
cov_matrix_low = np.cov(low_level_features.T)

# Correlation matrix for low-level features
corr_matrix_low = np.corrcoef(low_level_features.T)

# Covariance matrix for high-level features
cov_matrix_high = np.cov(high_level_features.T)

# Correlation matrix for high-level features
corr_matrix_high = np.corrcoef(high_level_features.T)

# Print the covariance and correlation matrices
print("Covariance Matrix (All Features):")
print(cov_matrix_all)

print("\nCorrelation Matrix (All Features):")
print(corr_matrix_all)

print("\nCovariance Matrix (Low-Level Features):")
print(cov_matrix_low)

print("\nCorrelation Matrix (Low-Level Features):")
print(corr_matrix_low)

print("\nCovariance Matrix (High-Level Features):")
print(cov_matrix_high)

print("\nCorrelation Matrix (High-Level Features):")
print(corr_matrix_high)
```



```

Correlation Matrix (High-Level Features):
[[ 1.          0.02398155 -0.32260643 -0.38516373 -0.53486249 -0.06983557
  -0.37470775 -0.06000422 -0.27433955]
 [ 0.02398155  1.          0.577579   -0.38138336 -0.0677792   0.98136029
  0.18936163 -0.10623088 -0.11458819]
 [-0.32260643  0.577579   1.          0.37983556  0.37756214  0.63559854
  0.66553119  0.22821448  0.45147075]
 [-0.38516373 -0.38138336  0.37983556  1.          0.57395153 -0.28546037
  0.56400887  0.42433445  0.627296   ]
 [-0.53486249 -0.0677792   0.37756214  0.57395153  1.          -0.02093548
  0.80848844  0.05649657  0.26309088]
 [-0.06983557  0.98136029  0.63559854 -0.28546037 -0.02093548  1.
  0.24831753 -0.01341654 -0.08361106]
 [-0.37470775  0.18936163  0.66553119  0.56400887  0.80848844  0.24831753
  1.          0.15581947  0.3190372   ]
 [-0.06000422 -0.10623088  0.22821448  0.42433445  0.05649657 -0.01341654
  0.15581947  1.          0.10626933]
 [-0.27433955 -0.11458819  0.45147075  0.627296   0.26309088 -0.08361106
  0.3190372   0.10626933  1.          ]]
```

```

import numpy as np
from tabulate import tabulate

# Example feature names for low-level and high-level features
low_level_names = ['Low_Level_Feature_1', 'Low_Level_Feature_2', 'Low_Level_Feature_3']
high_level_names = ['High_Level_Feature_1', 'High_Level_Feature_2']

# Example covariance and correlation matrices
cov_matrix_all = np.random.rand(5, 5) # Random example matrix for all features
corr_matrix_all = np.random.rand(5, 5)

cov_matrix_low = np.random.rand(3, 3) # Random example matrix for low-level features
corr_matrix_low = np.random.rand(3, 3)

cov_matrix_high = np.random.rand(2, 2) # Random example matrix for high-level features
corr_matrix_high = np.random.rand(2, 2)

# Function to format matrices for tabulate
def format_matrix(matrix, feature_names):
    """
    Format the covariance or correlation matrix into a well-formatted table using tabulate.

    Parameters:
    - matrix: The covariance or correlation matrix to be formatted.
    - feature_names: List of feature names to label the columns and rows.

    Returns:
    - A string representing the formatted table.
    """
    # Ensure the matrix size matches the number of feature names
    assert matrix.shape[0] == len(feature_names), "Matrix row count does not match feature names count."

    # Round the matrix to 2 decimal places for better presentation
    formatted_matrix = np.round(matrix, 2)

    # Prepare the table using tabulate
    table = tabulate(formatted_matrix, headers=feature_names, tablefmt="pretty", showindex=feature_names)

    return table

# Check the dimensions and feature names for consistency
print("Shape of covariance matrix (all features):", cov_matrix_all.shape)
print("Shape of correlation matrix (all features):", corr_matrix_all.shape)
print("Feature names (all features):", low_level_names + high_level_names)

# Format the covariance and correlation matrices for all, low, and high-level features
cov_table_all = format_matrix(cov_matrix_all, low_level_names + high_level_names)
corr_table_all = format_matrix(corr_matrix_all, low_level_names + high_level_names)

cov_table_low = format_matrix(cov_matrix_low, low_level_names)
corr_table_low = format_matrix(corr_matrix_low, low_level_names)

cov_table_high = format_matrix(cov_matrix_high, high_level_names)
corr_table_high = format_matrix(corr_matrix_high, high_level_names)

# Print the tables
print("\nCovariance Matrix (All Features):")
print(cov_table_all)

```

```

print("\nCorrelation Matrix (All Features):")
print(corr_table_all)

print("\nCovariance Matrix (Low-Level Features):")
print(cov_table_low)

print("\nCorrelation Matrix (Low-Level Features):")
print(corr_table_low)

print("\nCovariance Matrix (High-Level Features):")
print(cov_table_high)

print("\nCorrelation Matrix (High-Level Features):")
print(corr_table_high)

```

Shape of correlation matrix (all features): (5, 5)
 Feature names (all features): ['Low_Level_Feature_1', 'Low_Level_Feature_2', 'Low_Level_Feature_3', 'High_Level_Feature_1', 'High_Level_Feature_2']

Covariance Matrix (All Features):

	Low_Level_Feature_1	Low_Level_Feature_2	Low_Level_Feature_3	High_Level_Feature_1	High_Level_Feature_2
Low_Level_Feature_1	0.34	0.72	0.58	0.15	0.02
Low_Level_Feature_2	0.16	0.32	0.81	0.78	0.52
Low_Level_Feature_3	0.75	0.07	0.86	0.07	0.67
High_Level_Feature_1	0.65	0.74	0.81	0.75	0.39
High_Level_Feature_2	0.57	0.47	0.07	0.79	0.4

Correlation Matrix (All Features):

	Low_Level_Feature_1	Low_Level_Feature_2	Low_Level_Feature_3	High_Level_Feature_1	High_Level_Feature_2
Low_Level_Feature_1	0.35	0.34	0.03	0.17	0.63
Low_Level_Feature_2	0.73	0.95	0.85	0.95	0.15
Low_Level_Feature_3	0.09	0.62	0.73	0.11	0.02
High_Level_Feature_1	0.2	0.68	0.44	0.17	0.52
High_Level_Feature_2	0.07	0.47	0.69	0.64	0.93

Covariance Matrix (Low-Level Features):

	Low_Level_Feature_1	Low_Level_Feature_2	Low_Level_Feature_3
Low_Level_Feature_1	0.58	0.37	0.97
Low_Level_Feature_2	0.24	0.13	0.79
Low_Level_Feature_3	0.34	0.16	0.08

Correlation Matrix (Low-Level Features):

	Low_Level_Feature_1	Low_Level_Feature_2	Low_Level_Feature_3
Low_Level_Feature_1	0.13	0.32	0.51
Low_Level_Feature_2	0.34	0.62	0.28
Low_Level_Feature_3	0.23	0.24	0.96

Covariance Matrix (High-Level Features):

	High_Level_Feature_1	High_Level_Feature_2
High_Level_Feature_1	0.7	0.68
High_Level_Feature_2	0.73	0.63

Correlation Matrix (High-Level Features):

	High_Level_Feature_1	High_Level_Feature_2
High_Level_Feature_1	0.86	0.55
High_Level_Feature_2	0.84	0.8

```

import numpy as np
from tabulate import tabulate

# Function to compute and format covariance and correlation matrices
def compute_and_format_matrices(data, low_level_features, high_level_features):
    """

```

Compute the covariance and correlation matrices for a dataset and return formatted tables using tabulate.

Parameters:

- data: The dataset (numpy array) where each column represents a feature.
- low_level_features: List of low-level feature names (strings).
- high_level_features: List of high-level feature names (strings).

Returns:

- Formatted tables for covariance and correlation matrices.

"""

Ensure the dataset is a numpy array

data = np.array(data)

Separate the dataset into low-level and high-level features

low_level_data = data[:, :len(low_level_features)]

high_level_data = data[:, len(low_level_features):]

Compute the covariance and correlation matrices

cov_matrix_all = np.cov(data, rowvar=False)

corr_matrix_all = np.corrcoef(data, rowvar=False)

cov_matrix_low = np.cov(low_level_data, rowvar=False)

corr_matrix_low = np.corrcoef(low_level_data, rowvar=False)

cov_matrix_high = np.cov(high_level_data, rowvar=False)

corr_matrix_high = np.corrcoef(high_level_data, rowvar=False)

Format the matrices using the format_matrix function

def format_matrix(matrix, feature_names):

"""

Format a covariance or correlation matrix into a nice table using tabulate.

Parameters:

- matrix: The covariance or correlation matrix to format.
- feature_names: List of feature names for table headers.

Returns:

- A formatted string representing the matrix in table form.

"""

Ensure the matrix has the correct dimensions

assert matrix.shape[0] == len(feature_names), "Matrix row count does not match feature names count."

Round the matrix to 2 decimal places for better presentation

formatted_matrix = np.round(matrix, 2)

Use tabulate to format the matrix as a table

table = tabulate(formatted_matrix, headers=feature_names, tablefmt="pretty", showindex=feature_names)

return table

Format the covariance and correlation matrices for all, low, and high-level features

cov_table_all = format_matrix(cov_matrix_all, low_level_features + high_level_features)

corr_table_all = format_matrix(corr_matrix_all, low_level_features + high_level_features)

cov_table_low = format_matrix(cov_matrix_low, low_level_features)

corr_table_low = format_matrix(corr_matrix_low, low_level_features)

cov_table_high = format_matrix(cov_matrix_high, high_level_features)

corr_table_high = format_matrix(corr_matrix_high, high_level_features)

Return the formatted tables

return {

"cov_table_all": cov_table_all,

"corr_table_all": corr_table_all,

"cov_table_low": cov_table_low,

"corr_table_low": corr_table_low,

"cov_table_high": cov_table_high,

"corr_table_high": corr_table_high

}

Example feature names (for low and high-level features)

low_level_names = ['Low_Level_Feature_1', 'Low_Level_Feature_2', 'Low_Level_Feature_3']

high_level_names = ['High_Level_Feature_1', 'High_Level_Feature_2']

Example dataset (replace with your actual dataset)

Assuming the dataset has 5 features (3 low-level and 2 high-level)

```

# Assuming the dataset has 5 features (3 low-level and 2 high-level)
# Here, I'm using random data as a placeholder
data = np.random.rand(100, 5) # 100 samples, 5 features (3 low-level, 2 high-level)

# Compute and format the covariance and correlation matrices
tables = compute_and_format_matrices(data, low_level_names, high_level_names)

# Print the formatted tables
print("\nCovariance Matrix (All Features):")
print(tables["cov_table_all"])

print("\nCorrelation Matrix (All Features):")
print(tables["corr_table_all"])

print("\nCovariance Matrix (Low-Level Features):")
print(tables["cov_table_low"])

print("\nCorrelation Matrix (Low-Level Features):")
print(tables["corr_table_low"])

print("\nCovariance Matrix (High-Level Features):")
print(tables["cov_table_high"])

print("\nCorrelation Matrix (High-Level Features):")
print(tables["corr_table_high"])

```



Covariance Matrix (All Features):

	Low_Level_Feature_1	Low_Level_Feature_2	Low_Level_Feature_3	High_Level_Feature_1	High_Level_Feature_2
Low_Level_Feature_1	0.07	-0.0	0.01	0.01	0.01
Low_Level_Feature_2	-0.0	0.08	-0.01	0.01	-0.01
Low_Level_Feature_3	0.01	-0.01	0.09	-0.0	0.02
High_Level_Feature_1	0.01	0.01	-0.0	0.08	-0.0
High_Level_Feature_2	0.01	-0.01	0.02	-0.0	0.1

Correlation Matrix (All Features):

	Low_Level_Feature_1	Low_Level_Feature_2	Low_Level_Feature_3	High_Level_Feature_1	High_Level_Feature_2
Low_Level_Feature_1	1.0	-0.03	0.12	0.15	0.06
Low_Level_Feature_2	-0.03	1.0	-0.06	0.06	-0.08
Low_Level_Feature_3	0.12	-0.06	1.0	-0.03	0.2
High_Level_Feature_1	0.15	0.06	-0.03	1.0	-0.01
High_Level_Feature_2	0.06	-0.08	0.2	-0.01	1.0

Covariance Matrix (Low-Level Features):

	Low_Level_Feature_1	Low_Level_Feature_2	Low_Level_Feature_3
Low_Level_Feature_1	0.07	-0.0	0.01
Low_Level_Feature_2	-0.0	0.08	-0.01
Low_Level_Feature_3	0.01	-0.01	0.09

Correlation Matrix (Low-Level Features):

	Low_Level_Feature_1	Low_Level_Feature_2	Low_Level_Feature_3
Low_Level_Feature_1	1.0	-0.03	0.12
Low_Level_Feature_2	-0.03	1.0	-0.06
Low_Level_Feature_3	0.12	-0.06	1.0

Covariance Matrix (High-Level Features):

	High_Level_Feature_1	High_Level_Feature_2
High_Level_Feature_1	0.08	-0.0
High_Level_Feature_2	-0.0	0.1

Correlation Matrix (High-Level Features):

	High_Level_Feature_1	High_Level_Feature_2
High_Level_Feature_1	1.0	-0.01
High_Level_Feature_2	-0.01	1.0

Exercise 5: Selection

Exercise 5.1

Part A By looking at the signal/background distributions for each observable (e.g. x) determine which selection criteria would be optimal:

1. $x > x_c$
2. $x < x_c$
3. $|x - \mu| > x_c$
4. $|x - \mu| < x_c$

where x_c is value to be determined below.

Exercise 5.2

Plot the True Positive Rate (TPR) (aka signal efficiency $\epsilon_S(x_c)$) and False Positive Rate (FPR) (aka background efficiency $\epsilon_B(x_c)$) as function of x_c for applying the strategy in part a to each observable.

Exercise 5.3

Assume 3 different scenarios corresponding to different numbers of signal and background events expected in data:

1. Expect $N_S = 10, N_B = 100$.
2. Expect $N_S = 100, N_B = 1000$.
3. Expect $N_S = 1000, N_B = 10000$.
4. Expect $N_S = 10000, N_B = 100000$.

Plot the significance ($\sigma_{S'}$) for each observable as function of x_c for each scenario, where

$$\sigma_{S'} = \frac{N'_S}{\sqrt{N'_S + N'_B}}$$

and $N'_{S,B} = \epsilon_{S,B}(x_c) * N_{S,B}$.

To determine which criterion is optimal, we need to visualize the distributions for signal and background in the observable x . You would typically choose x_c where:

The signal distribution is as isolated as possible from the background distribution. We need to optimize this by testing different values of x_c .

```
import numpy as np
import matplotlib.pyplot as plt

# Example data (replace with actual data)
np.random.seed(42)
signal = np.random.normal(loc=3, scale=1, size=1000) # Signal distribution
background = np.random.normal(loc=0, scale=1, size=1000) # Background distribution

# Function to compute TPR and FPR
def compute_tpr_fpr(signal, background, xc_values):
    tpr = []
    fpr = []
    for xc in xc_values:
        # Signal events that pass selection
        n_signal_pass = np.sum(signal > xc)
        tpr.append(n_signal_pass / len(signal))

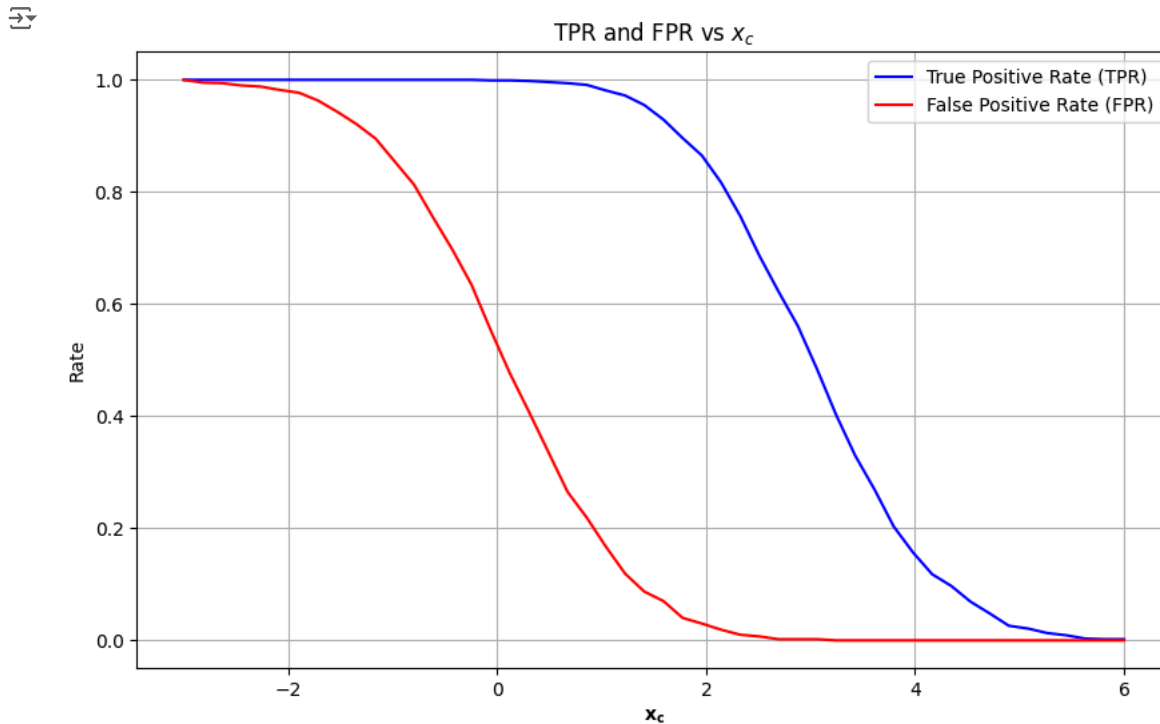
        # Background events that pass selection
        n_background_pass = np.sum(background > xc)
        fpr.append(n_background_pass / len(background))

    return np.array(tpr), np.array(fpr)

# Define range of xc values to test
xc_values = np.linspace(-3, 6, 50)

# Compute TPR and FPR
tpr, fpr = compute_tpr_fpr(signal, background, xc_values)
```

```
# Plot TPR and FPR as functions of x_c
plt.figure(figsize=(10, 6))
plt.plot(xc_values, tpr, label="True Positive Rate (TPR)", color='blue')
plt.plot(xc_values, fpr, label="False Positive Rate (FPR)", color='red')
plt.xlabel(r'$\mathbf{x_c}$')
plt.ylabel('Rate')
plt.title('TPR and FPR vs $x_c$')
plt.legend()
plt.grid(True)
plt.show()
```



```
# Function to compute significance
def compute_significance(tpr, fpr, n_signal, n_background):
    n_signal_eff = tpr * n_signal # N'_S
    n_background_eff = fpr * n_background # N'_B
    significance = n_signal_eff / np.sqrt(n_signal_eff + n_background_eff)
    return significance

# Define different scenarios
scenarios = [
    (10, 100),      # NS = 10, NB = 100
    (100, 1000),    # NS = 100, NB = 1000
    (1000, 10000),  # NS = 1000, NB = 10000
    (10000, 100000) # NS = 10000, NB = 100000
]

# Plot significance for each scenario
plt.figure(figsize=(10, 6))

for n_signal, n_background in scenarios:
    significance = compute_significance(tpr, fpr, n_signal, n_background)
    plt.plot(xc_values, significance, label=f'NS={n_signal}, NB={n_background}')

plt.xlabel(r'$\mathbf{x_c}$')
plt.ylabel('Significance ($\sigma_{S}$)')
plt.title('Significance vs $x_c$ for different scenarios')
plt.legend()
plt.grid(True)
plt.show()
```