Project File 5

- Continue working on cleaning the data, and solving for other models.

New Plans

- Perform pairplots of the top 30 columns to visualize the relationships between variables.
- Train the model again using the new clean data.

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot as plt
import seaborn as sns
```

In [ ]:
```
df = pd.read_csv('data/project_data.csv')
df.head()
```

Out[ ]:

	Flow ID	Source IP	Source Port	Destination IP	Destination Port	Protocol	Timestamp	Flow Duration
0	192.168.10.5-104.16.207.165-54865-443-6	104.16.207.165	443	192.168.10.5	54865	6	7/7/2017 3:30	3
1	192.168.10.5-104.16.28.216-55054-80-6	104.16.28.216	80	192.168.10.5	55054	6	7/7/2017 3:30	109
2	192.168.10.5-104.16.28.216-55055-80-6	104.16.28.216	80	192.168.10.5	55055	6	7/7/2017 3:30	52
3	192.168.10.16-104.17.241.25-46236-443-6	104.17.241.25	443	192.168.10.16	46236	6	7/7/2017 3:30	34
4	192.168.10.5-104.19.196.102-54863-443-6	104.19.196.102	443	192.168.10.5	54863	6	7/7/2017 3:30	3

5 rows × 85 columns

Problem with the Dataset

- Every row that is labeled DDoS is TCP
- There are no rows that are UDP that are DDoS

In [ ]:
```
df.columns = df.columns.str.strip() # many columns have preceeding and trailing whites

print(f'Dataframe number of rows    : {len(df)}')
ddos_rows = df[(df['Label'] == 'DDoS')]
tcp_rows = df[(df['Protocol'] == 6)]
udp_rows = df[(df['Protocol'] == 17)]
ddos_tcp_rows = df[(df['Label'] == 'DDoS') & (df['Protocol'] == 6)]
ddos_udp_rows = df[(df['Label'] == 'DDoS') & (df['Protocol'] == 17)]
print(f'Number of rows that are DDoS: {len(ddos_rows)} --> {len(ddos_rows)/len(df)*100
print(f'Number of rows that are TCP : {len(tcp_rows)} --> {len(tcp_rows)/len(df)*100:.
print(f'Number of rows that are UDP : {len(udp_rows)} --> {len(udp_rows)/len(df)*100:.
print(f'Number of rows that are TCP and DDoS: {len(ddos_tcp_rows)}')
print(f'Number of rows that are UDP and DDoS: {len(ddos_udp_rows)}')
print('\nCONCERN: All rows labeled DDoS ARE also TCP')
```

```
Dataframe number of rows    : 225745
Number of rows that are DDoS: 128027 --> 56.71%
Number of rows that are TCP : 192820 --> 85.41%
Number of rows that are UDP : 32871 --> 14.56%
Number of rows that are TCP and DDoS: 128027
Number of rows that are UDP and DDoS: 0

CONCERN: All rows labeled DDoS ARE also TCP
```

In [ ]:
```
df.replace([np.inf, -np.inf], np.nan, inplace=True) # There is an infinity value hidir
print(f'Total number of cells that  are empty: {df.isnull().sum().sum()}')
df.dropna(axis=0, inplace=True)
print(f'New total number of empty cells      : {df.isnull().sum().sum()}')
```

```
Total number of cells that  are empty: 68
New total number of empty cells      : 0
```

In [ ]:
```
#df.drop(columns=['Source IP'], inplace=True)
#df.drop(columns=['Destination IP'], inplace=True)
#df.drop(columns=['Source Port'], inplace=True)
#df.drop(columns=['Destination Port'], inplace=True)
#df.drop(columns=['Flow ID'], inplace=True)
#df.drop(columns=['Timestamp'], inplace=True)

drop_columns = ['Source IP', 'Destination IP', 'Source Port', 'Destination Port', 'Flo
df.drop(columns=drop_columns, inplace=True)
df.head()
```

Out[ ]:

	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	Bwd Packet Length Max	...	min_
0	3	2	0	12	0	6	6	6.0	0.0	0	...	
1	109	1	1	6	6	6	6	6.0	0.0	6	...	
2	52	1	1	6	6	6	6	6.0	0.0	6	...	
3	34	1	1	6	6	6	6	6.0	0.0	6	...	
4	3	2	0	12	0	6	6	6.0	0.0	0	...	

5 rows × 78 columns

In [ ]:
```
df['Label_encoded'] = df['Label'].map({'BENIGN': 0, 'DDoS': 1})
df.drop(columns=['Label'], inplace=True)
df.head()
```

Out[ ]:

	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	Bwd Packet Length Max	...	min_
0	3	2	0	12	0	6	6	6.0	0.0	0	...	
1	109	1	1	6	6	6	6	6.0	0.0	6	...	
2	52	1	1	6	6	6	6	6.0	0.0	6	...	
3	34	1	1	6	6	6	6	6.0	0.0	6	...	
4	3	2	0	12	0	6	6	6.0	0.0	0	...	

5 rows × 78 columns

In [ ]:
```
df.describe()
```

Out[ ]:

	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	F L
count	2.257110e+05	225711.000000	225711.000000	225711.000000	2.257110e+05	225711.000000	225
mean	1.624410e+07	4.875389	4.573424	939.603147	5.961369e+03	538.615499	
std	3.152612e+07	15.423986	21.756929	3249.628245	3.922122e+04	1864.258043	
min	-1.000000e+00	1.000000	0.000000	0.000000	0.000000e+00	0.000000	
25%	7.123800e+04	2.000000	1.000000	26.000000	0.000000e+00	6.000000	
50%	1.453164e+06	3.000000	4.000000	30.000000	1.640000e+02	20.000000	
75%	8.806652e+06	5.000000	5.000000	64.000000	1.160100e+04	34.000000	
max	1.199999e+08	1932.000000	2942.000000	183012.000000	5.172346e+06	11680.000000	1

8 rows × 78 columns

Normalization

Each column scales drastically. Some have values upward in the millions, and some have values where the mean is only 4.\ Normalization is necessary.

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler()
normalized_data = min_max_scaler.fit_transform(df)
normalized_df = pd.DataFrame(normalized_data, columns=df.columns)
normalized_df.head()
```

Out[ ]:

	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	Fwd Packet Length Min	Fwd Packet Length Mean	Fwd Packet Length Std	Bwd Packet Length Max
0	3.333335e-08	0.000518	0.00000	0.000066	0.000000	0.000514	0.004076	0.001552	0.0	0.000000
1	9.166671e-07	0.000000	0.00034	0.000033	0.000001	0.000514	0.004076	0.001552	0.0	0.000514
2	4.416669e-07	0.000000	0.00034	0.000033	0.000001	0.000514	0.004076	0.001552	0.0	0.000514
3	2.916668e-07	0.000000	0.00034	0.000033	0.000001	0.000514	0.004076	0.001552	0.0	0.000514
4	3.333335e-08	0.000518	0.00000	0.000066	0.000000	0.000514	0.004076	0.001552	0.0	0.000000

5 rows × 78 columns

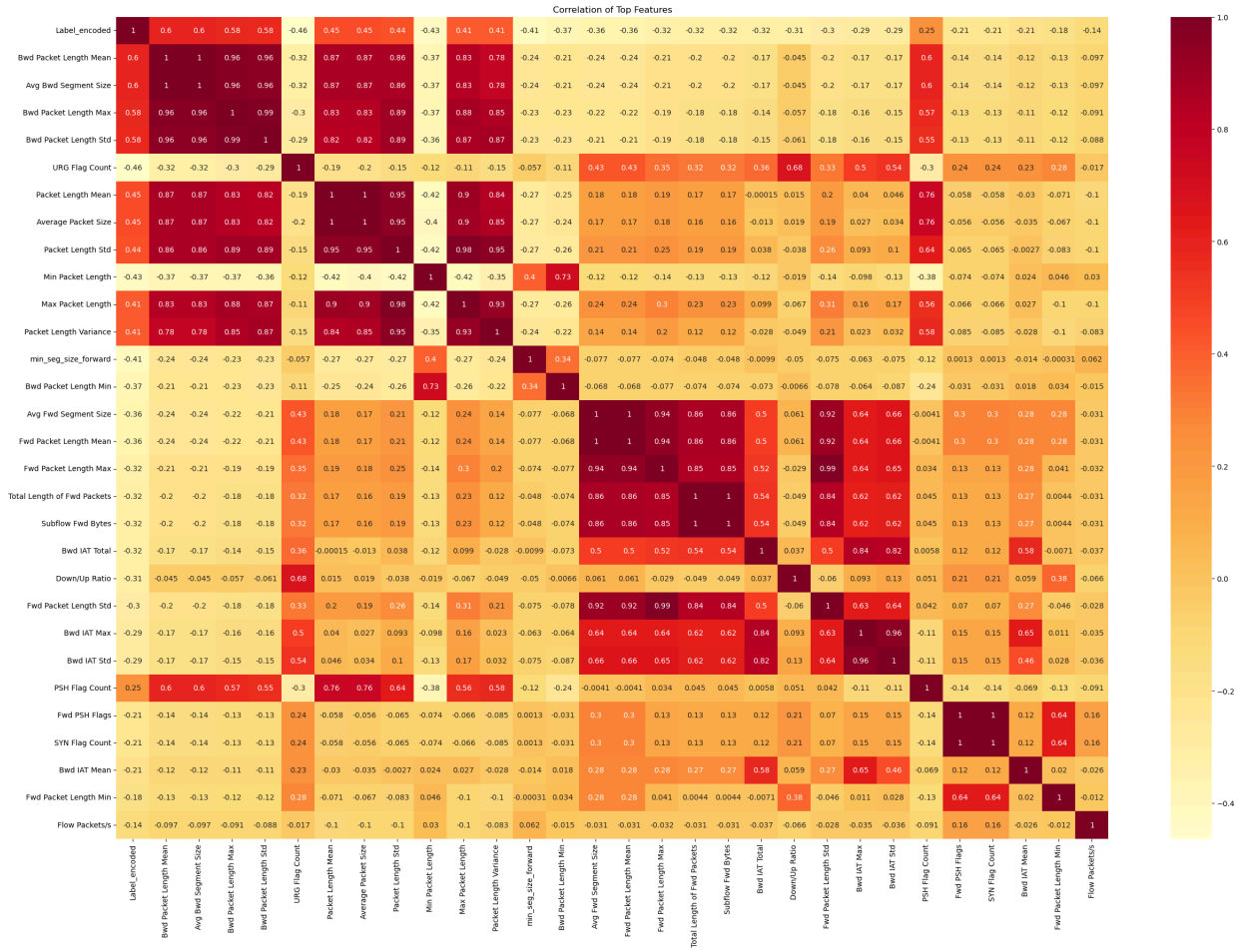
In [ ]:
```
normalized_df.describe()
```

Out[ ]:

	Flow Duration	Total Fwd Packets	Total Backward Packets	Total Length of Fwd Packets	Total Length of Bwd Packets	Fwd Packet Length Max	225
count	225711.000000	225711.000000	225711.000000	225711.000000	225711.000000	225711.000000	
mean	0.135368	0.002007	0.001555	0.005134	0.001153	0.046114	
std	0.262718	0.007988	0.007395	0.017756	0.007583	0.159611	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.000594	0.000518	0.000340	0.000142	0.000000	0.000514	
50%	0.012110	0.001036	0.001360	0.000164	0.000032	0.001712	
75%	0.073389	0.002071	0.001700	0.000350	0.002243	0.002911	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

8 rows × 78 columns

In [ ]:
```
corr_matrix = normalized_df.corr().abs()
top_features = corr_matrix['Label_encoded'].sort_values(ascending=False).head(30).inde
top_corr = normalized_df[top_features].corr()

plt.figure(figsize=(30, 20))
sns.heatmap(top_corr, annot=True, cmap='YlOrRd')
plt.title('Correlation of Top Features')
plt.show()
```



In [ ]:
```
print(top_features)
df_top_features = normalized_df[top_features]
```

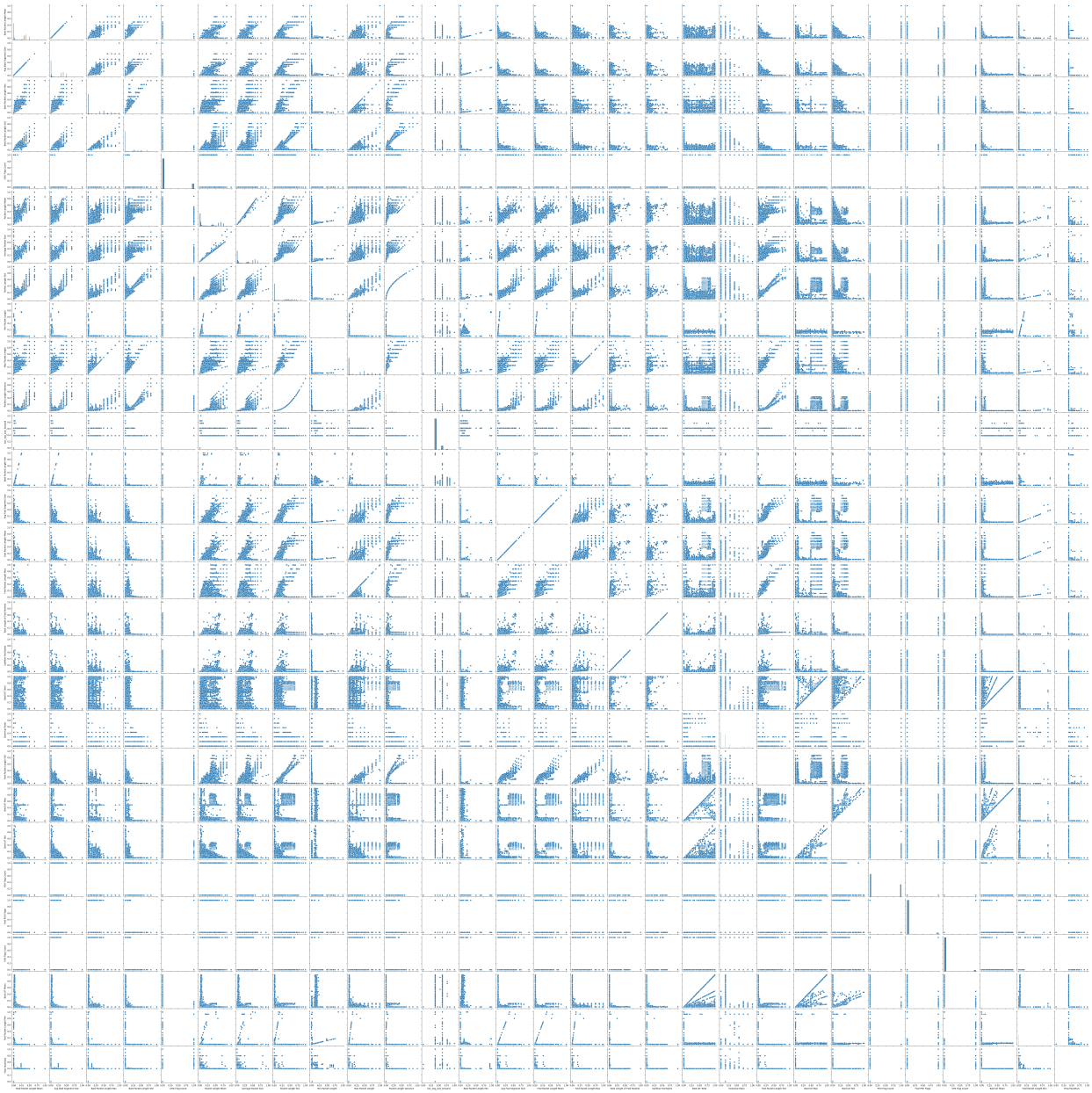
```
Index(['Label_encoded', 'Bwd Packet Length Mean', 'Avg Bwd Segment Size',
       'Bwd Packet Length Max', 'Bwd Packet Length Std', 'URG Flag Count',
       'Packet Length Mean', 'Average Packet Size', 'Packet Length Std',
       'Min Packet Length', 'Max Packet Length', 'Packet Length Variance',
       'min_seg_size_forward', 'Bwd Packet Length Min', 'Avg Fwd Segment Size',
       'Fwd Packet Length Mean', 'Fwd Packet Length Max',
       'Total Length of Fwd Packets', 'Subflow Fwd Bytes', 'Bwd IAT Total',
       'Down/Up Ratio', 'Fwd Packet Length Std', 'Bwd IAT Max', 'Bwd IAT Std',
       'PSH Flag Count', 'Fwd PSH Flags', 'SYN Flag Count', 'Bwd IAT Mean',
       'Fwd Packet Length Min', 'Flow Packets/s'],
      dtype='object')
```

Pairplots of top correlated features

In [ ]:
```
top_features = [feature for feature in corr_matrix['Label_encoded'].sort_values(ascend
df_top_features = normalized_df[top_features]

sns.pairplot(df_top_features)
```

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

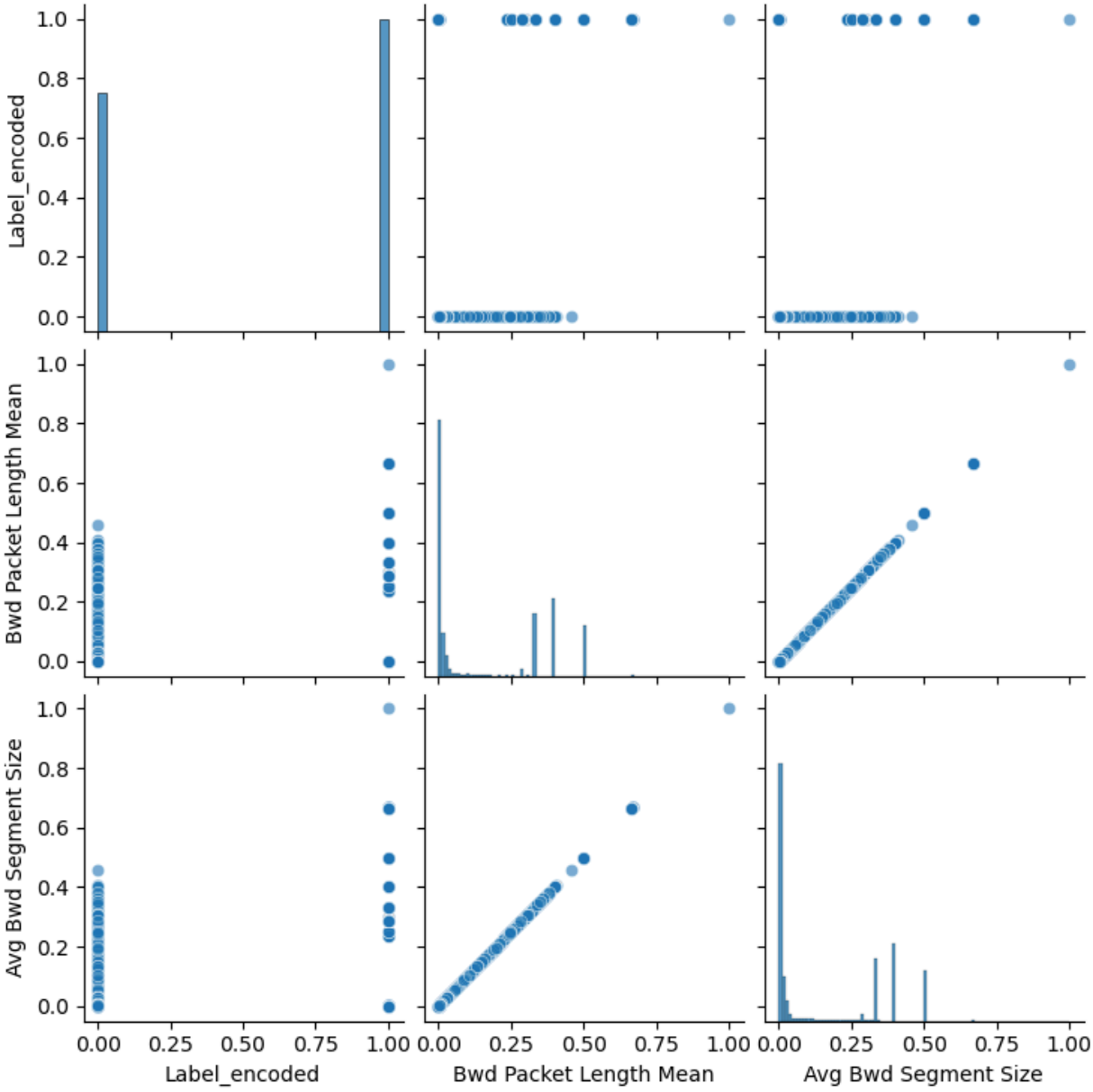
Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x18fa769b2d0>
```



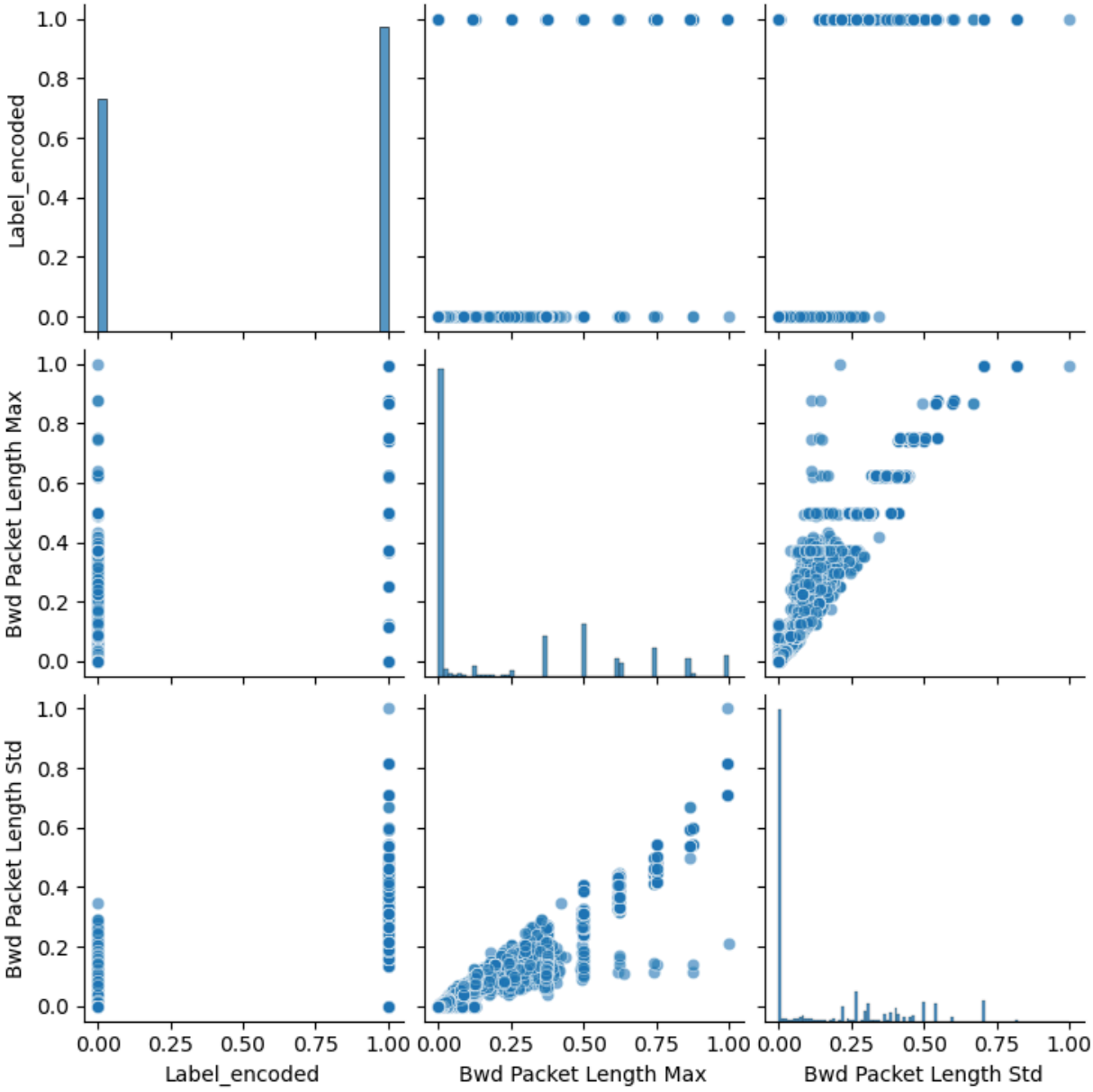
Implemented script to generate pairplots for subsets of top correlated features with the Label_encoded, with each graph displaying three features.

In [ ]:
```
top_features = [feature for feature in corr_matrix['Label_encoded'].sort_values(ascend
subsets = [['Label_encoded'] + top_features[i:i+2] for i in range(0, len(top_features)

# Create pairplots for each subset
for subset in subsets:
    sns.pairplot(normalized_df, x_vars=subset, y_vars=subset, plot_kws={'alpha': 0.6})
    plt.show()
```

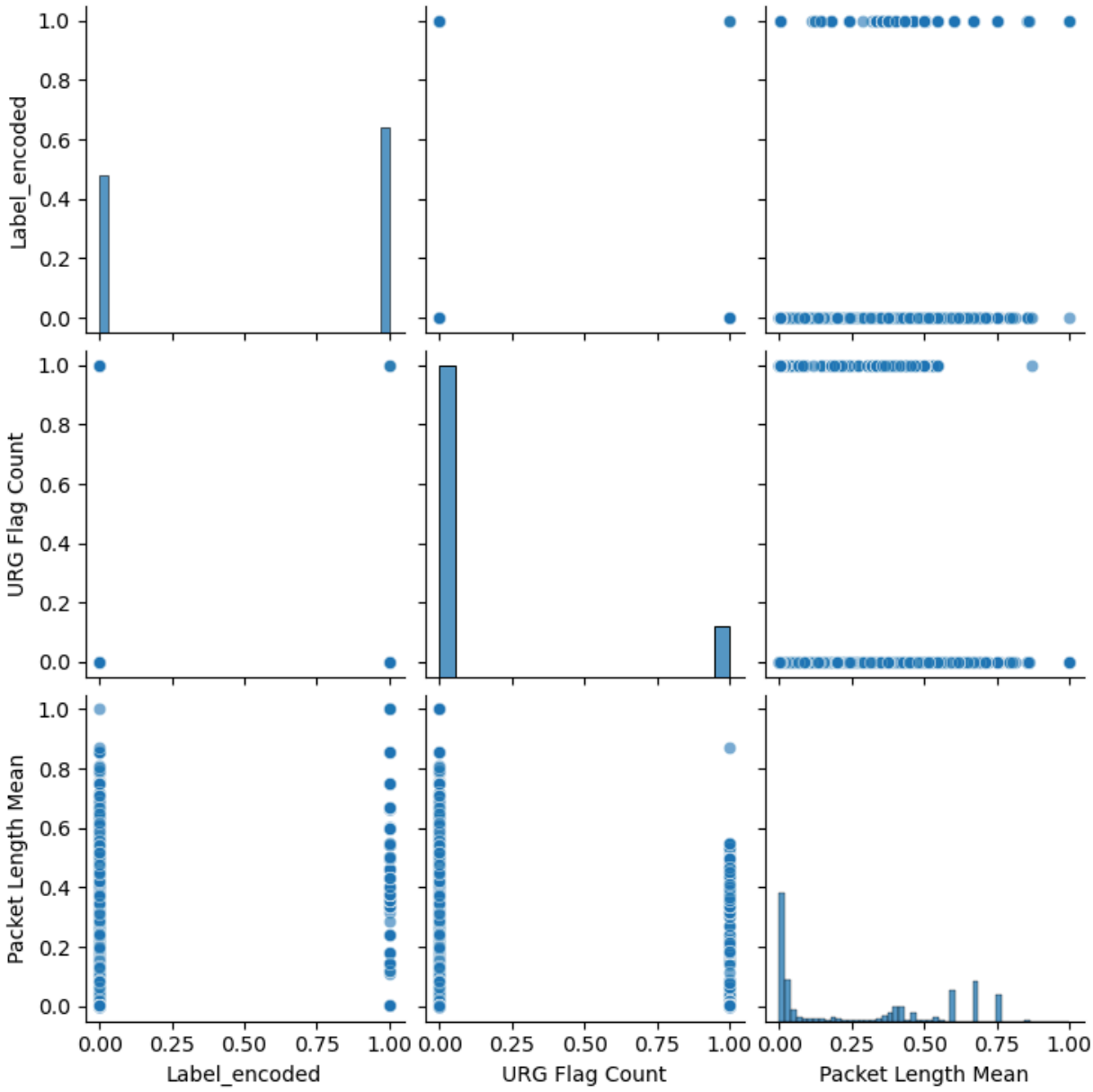
```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



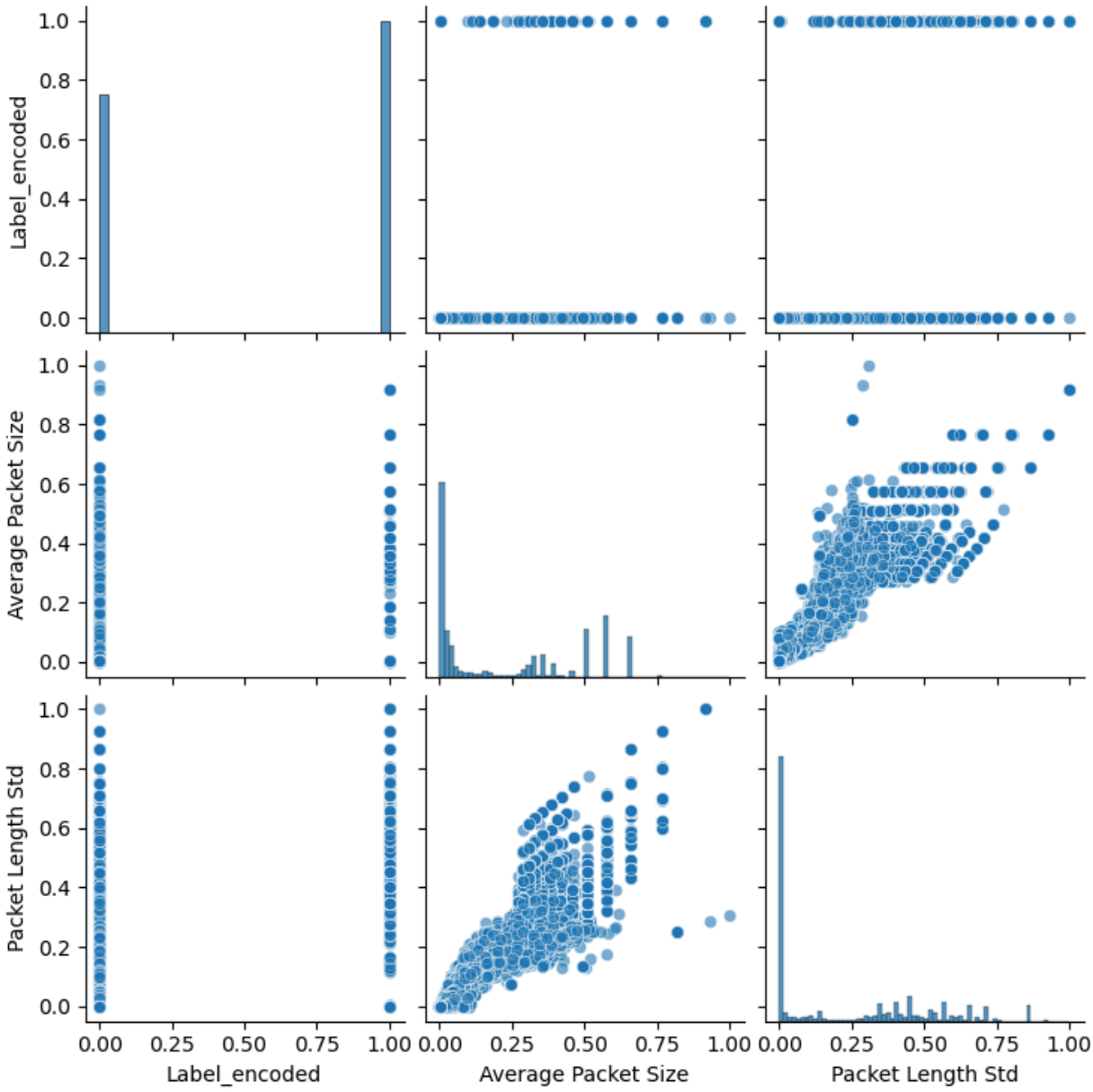

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



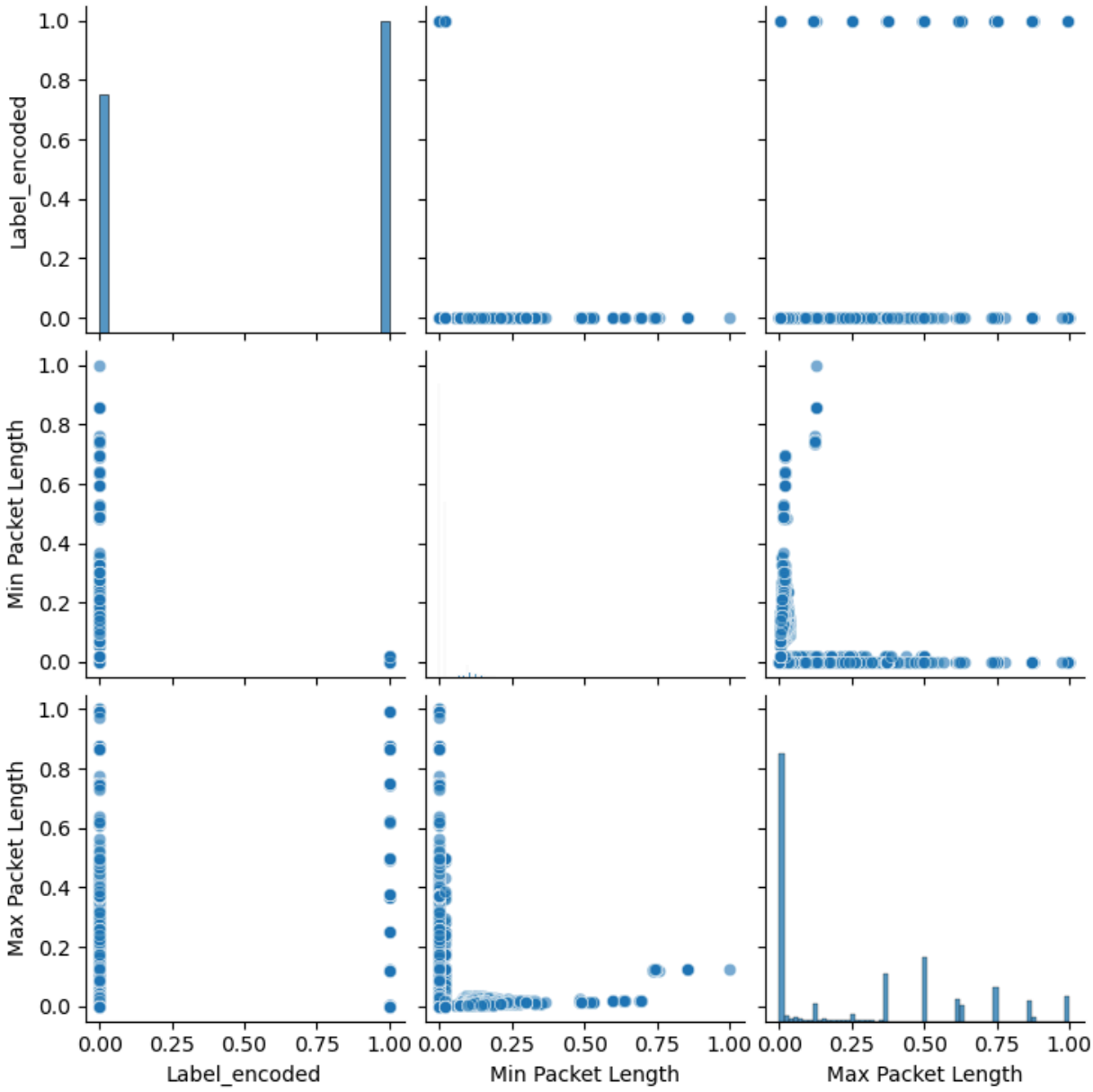

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



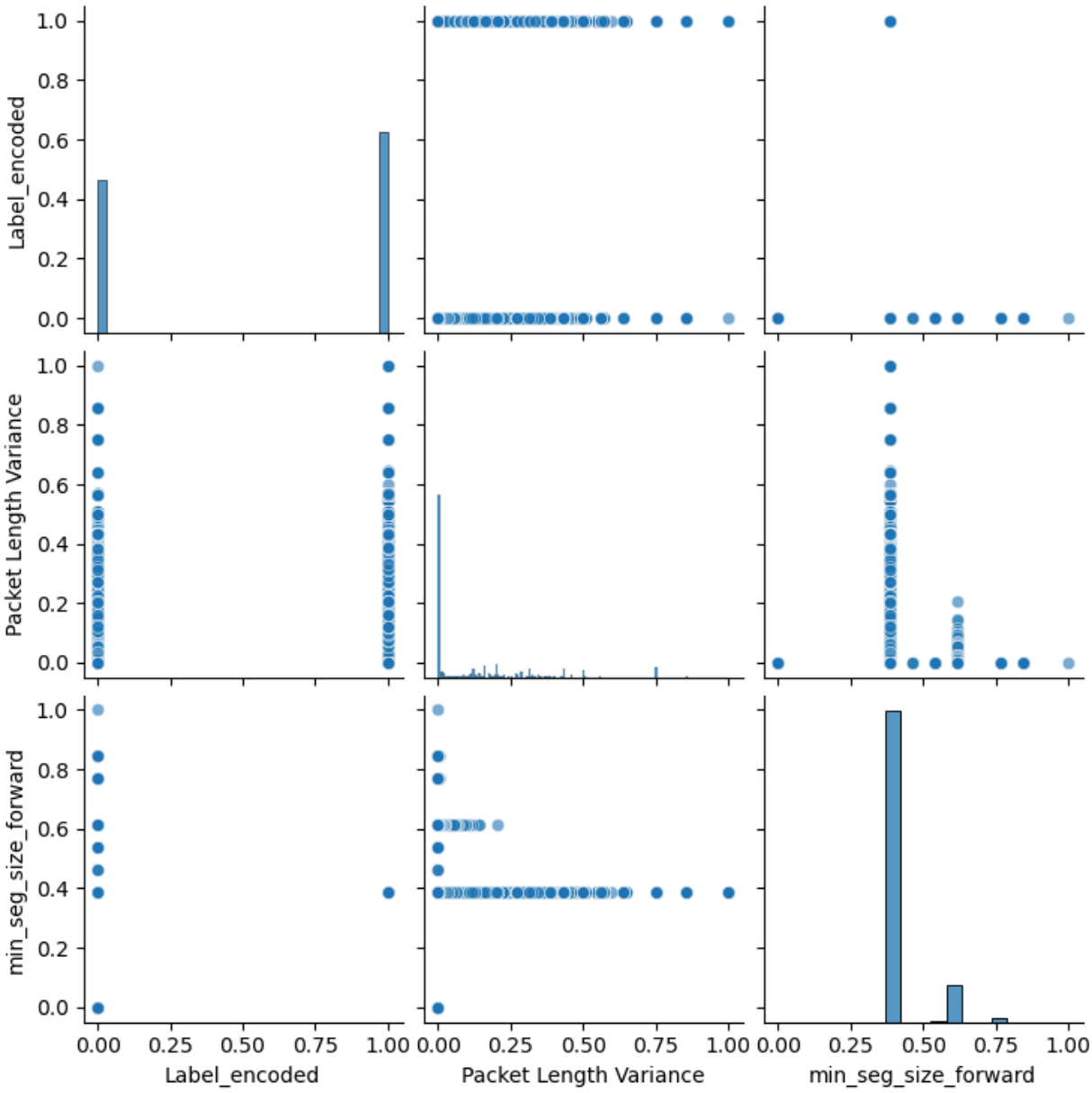

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



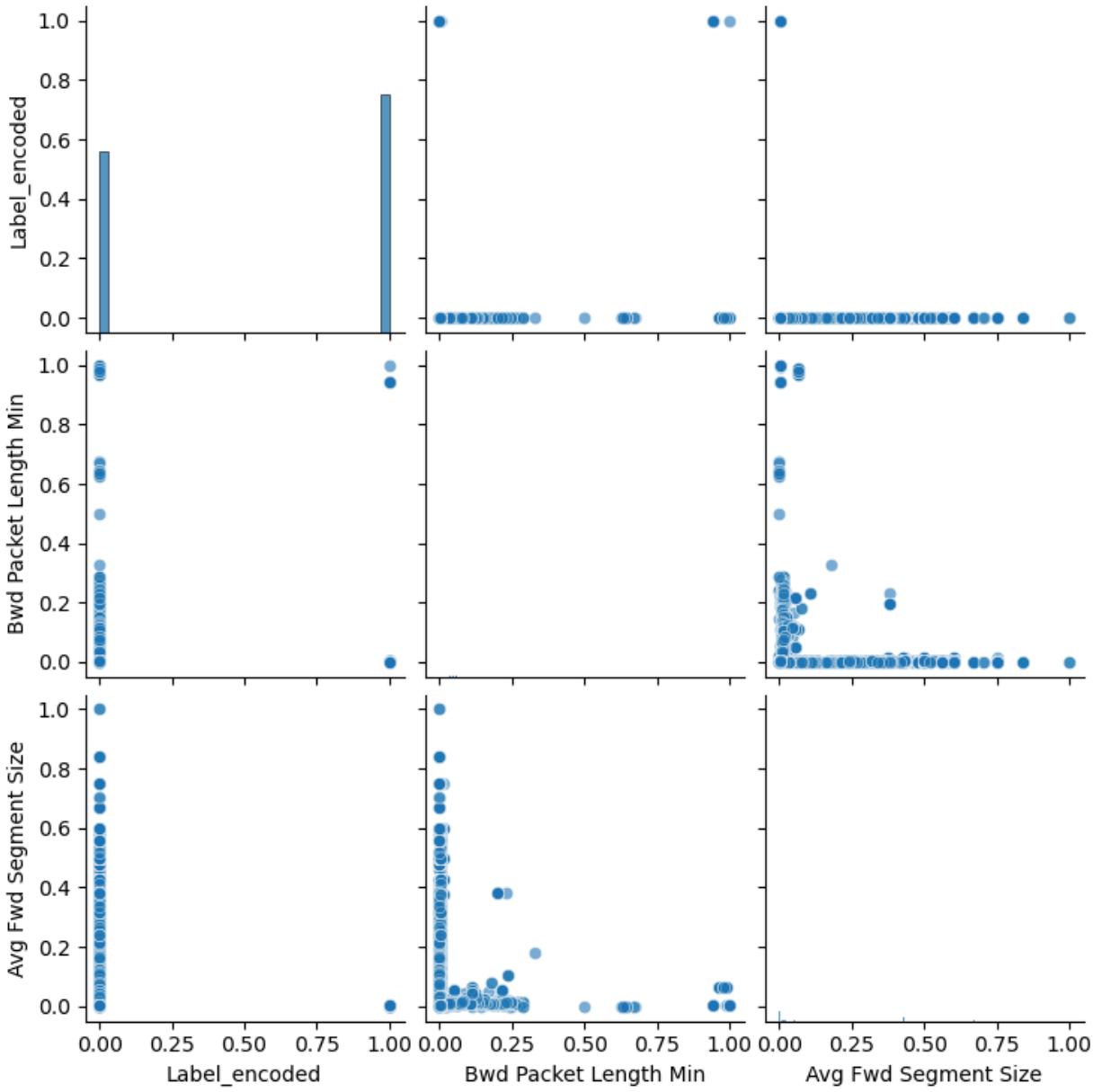

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



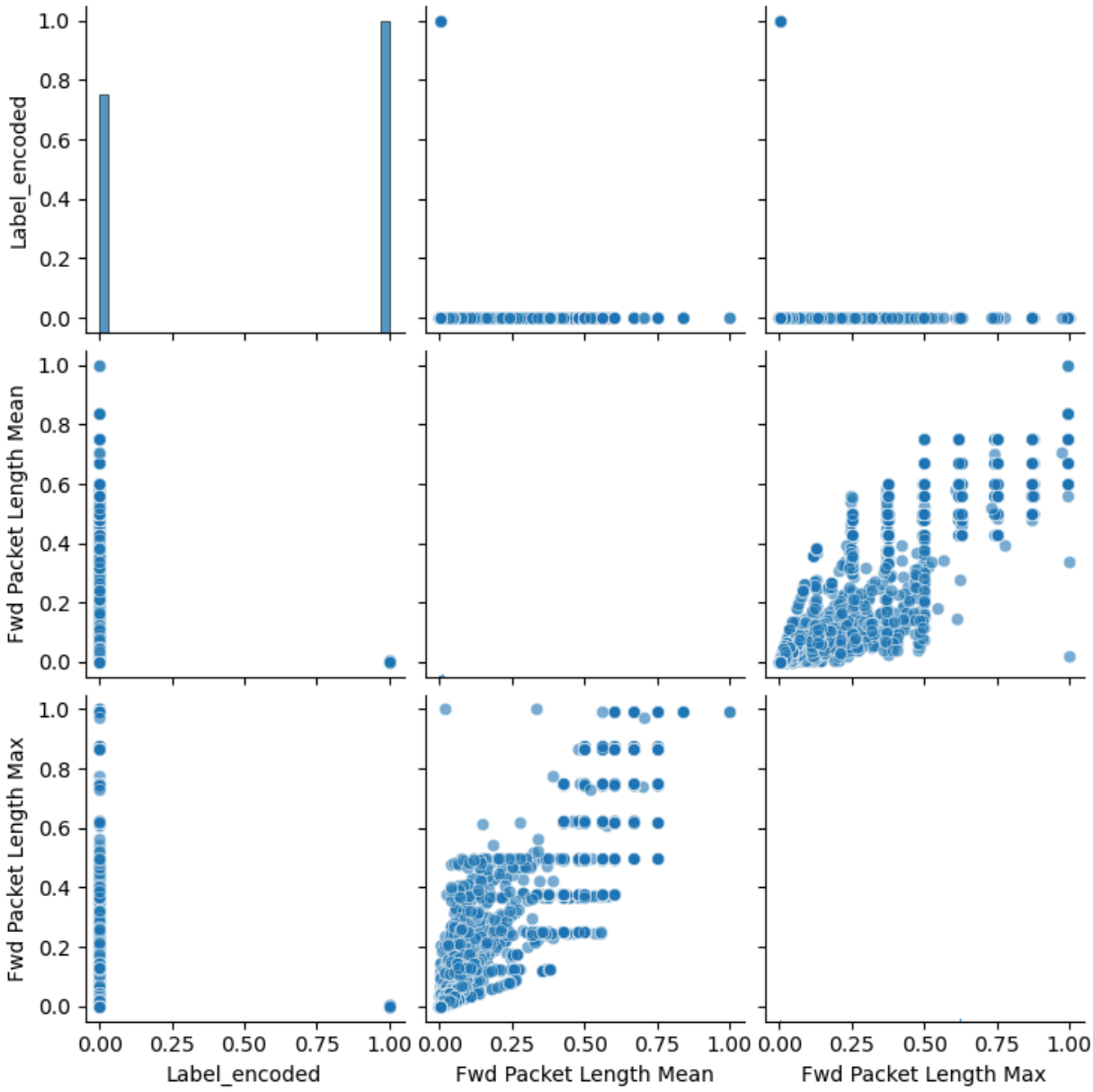

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



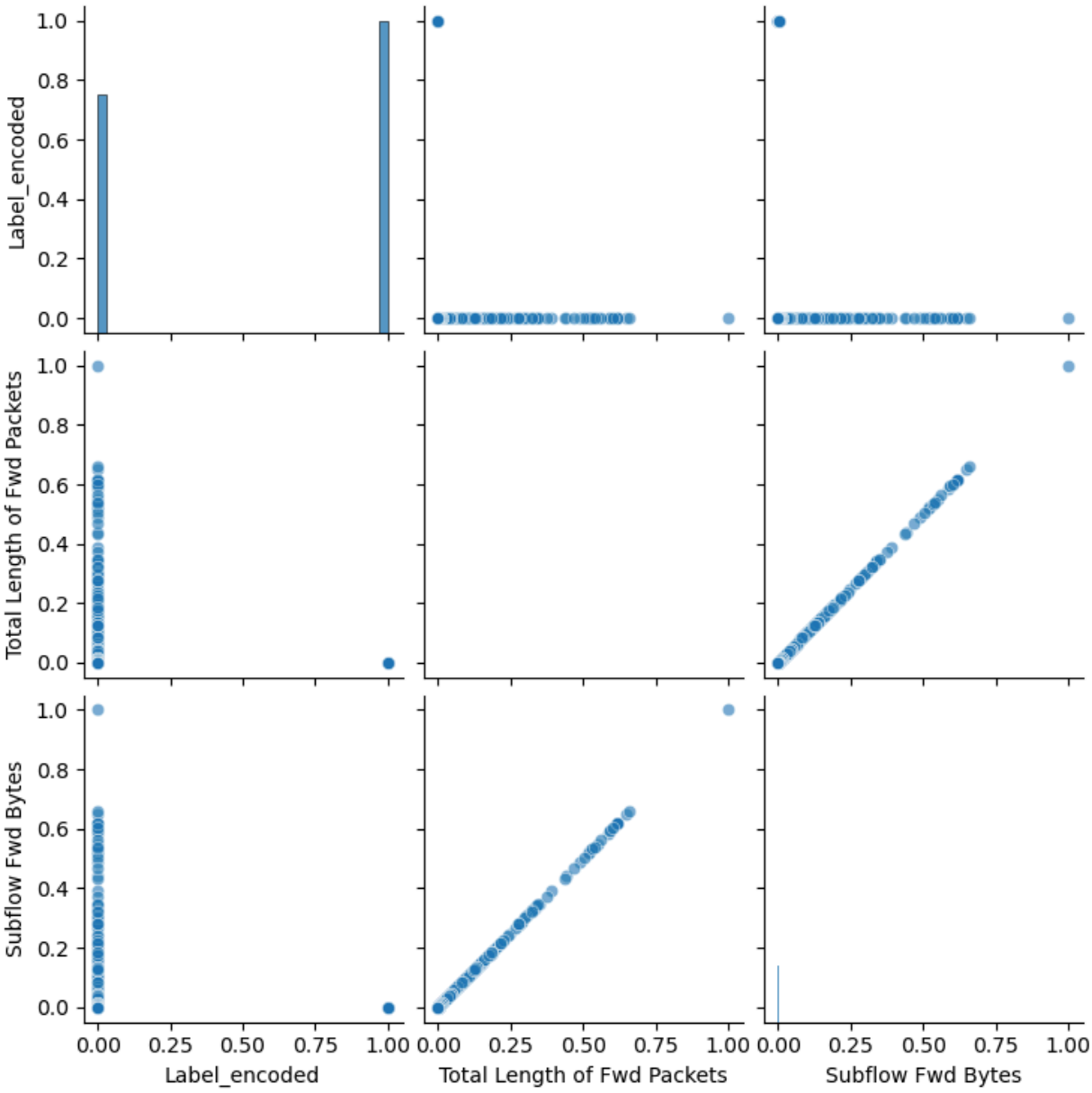

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



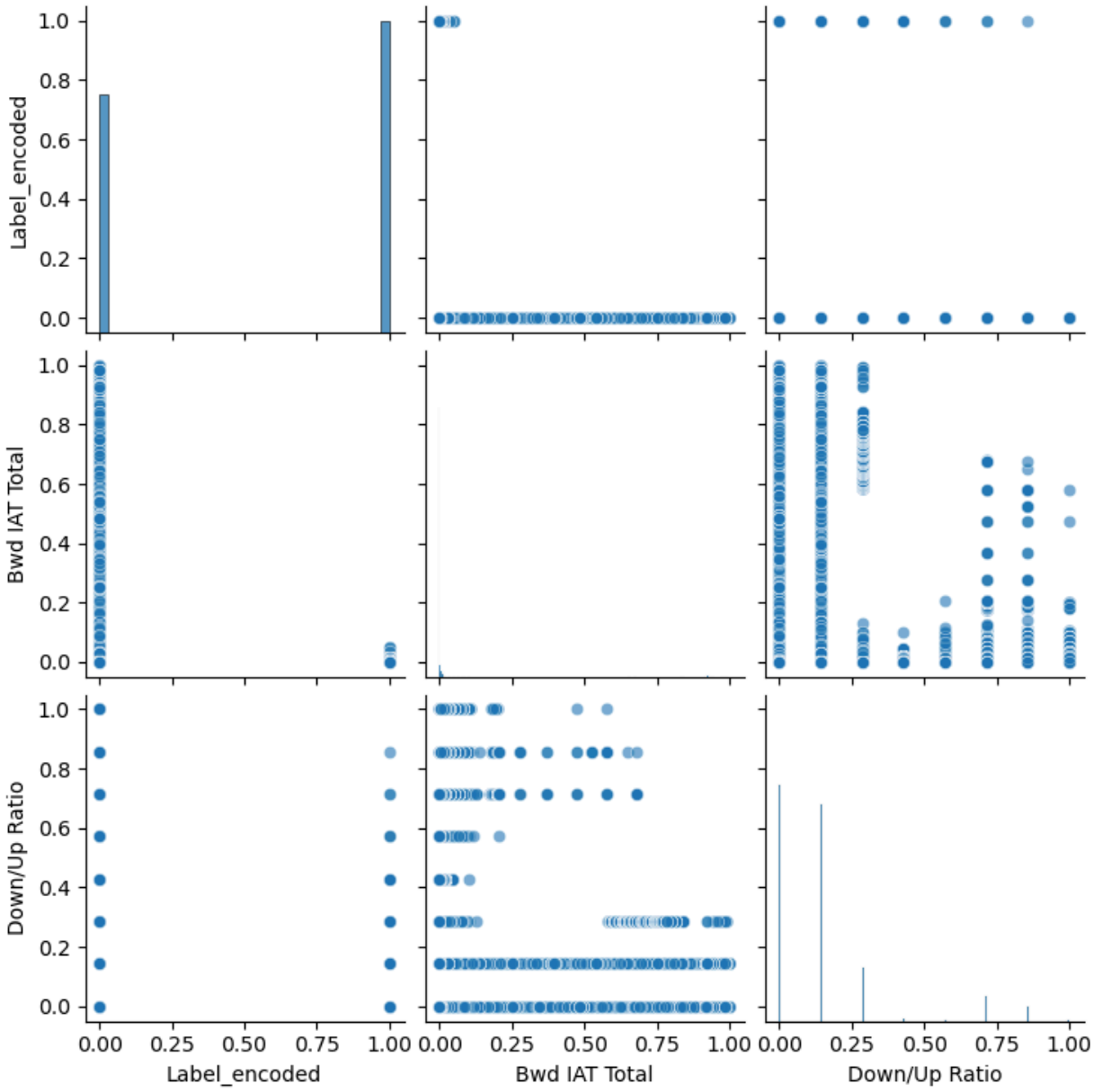

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



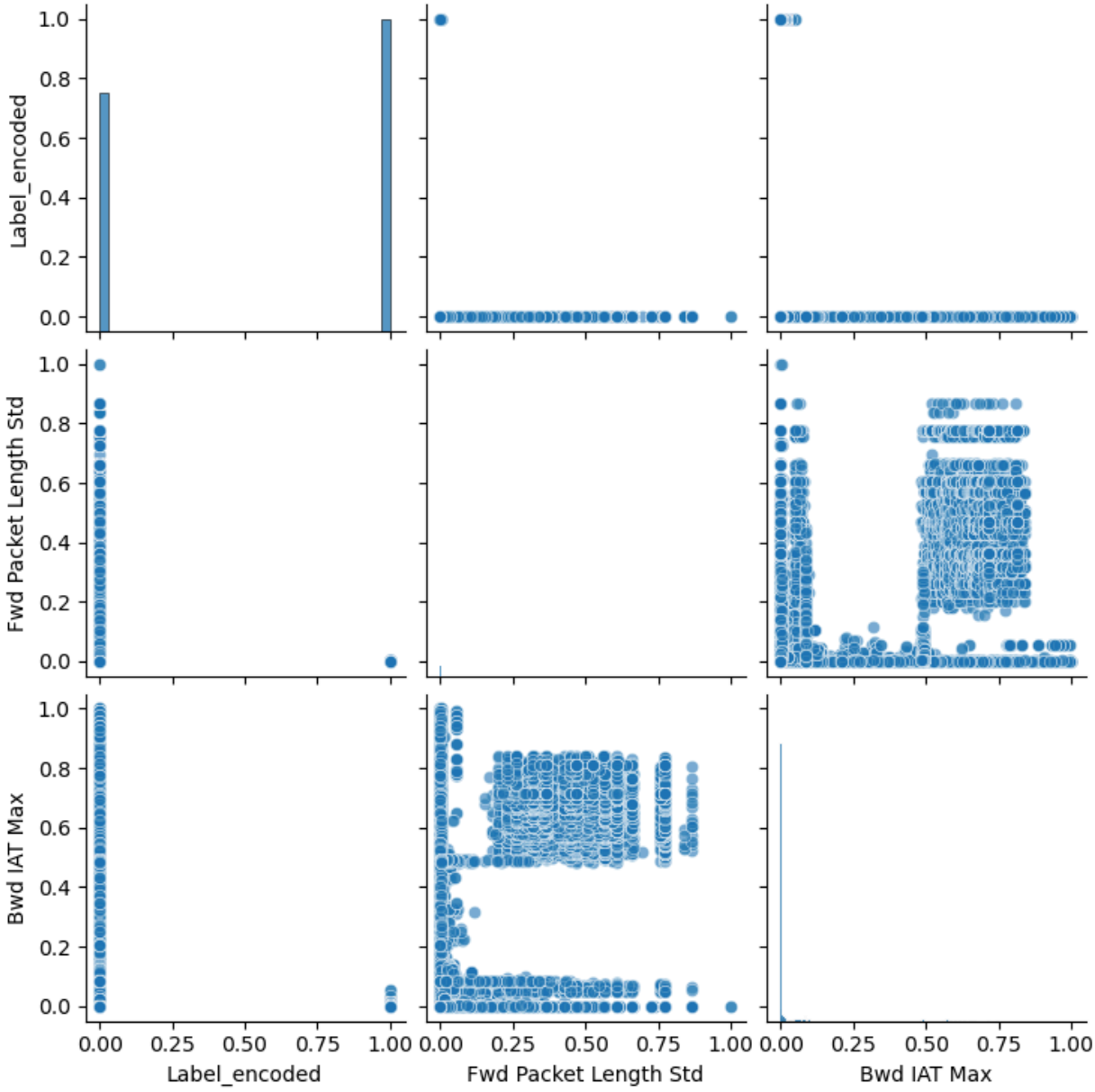

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



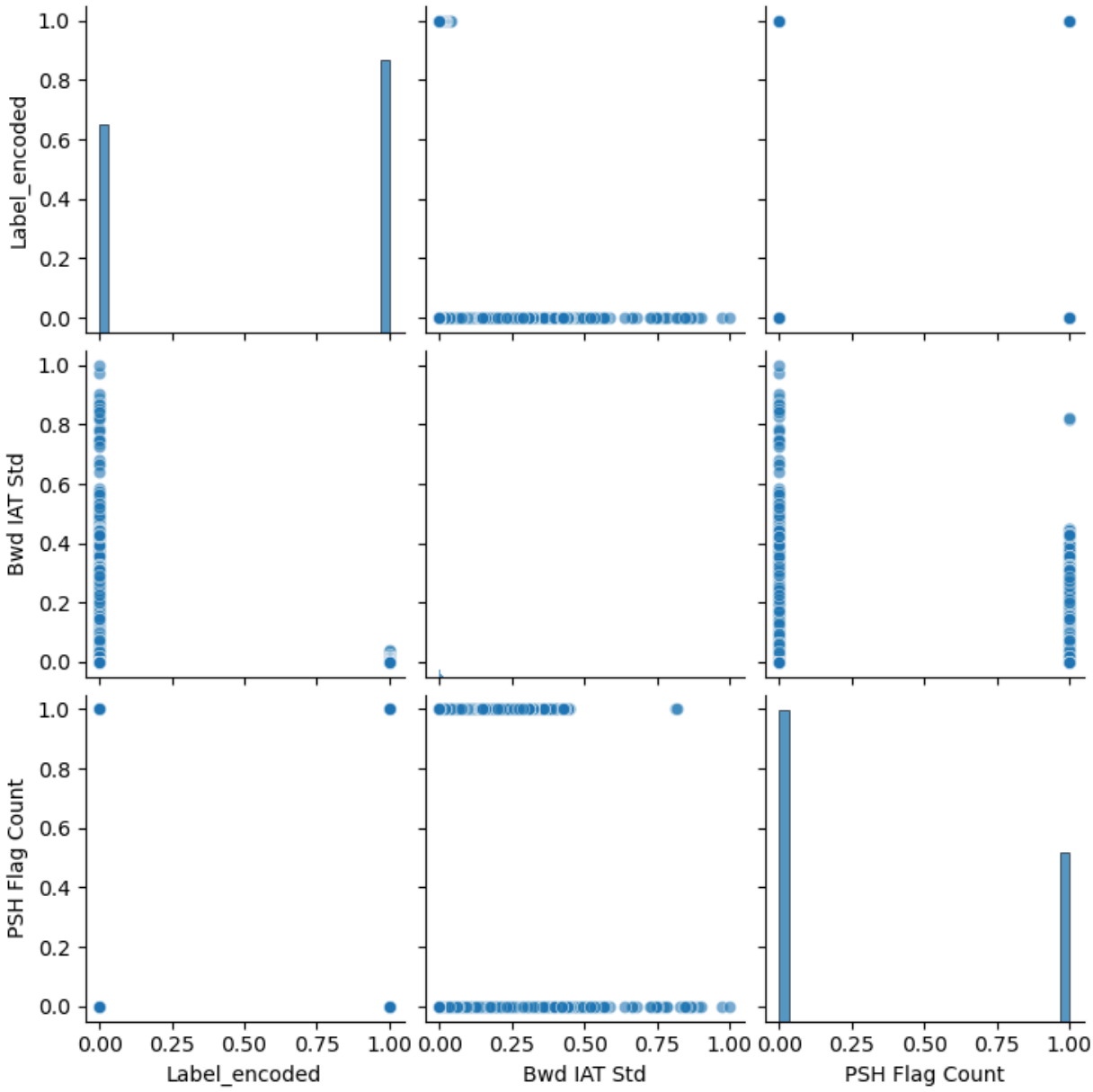

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



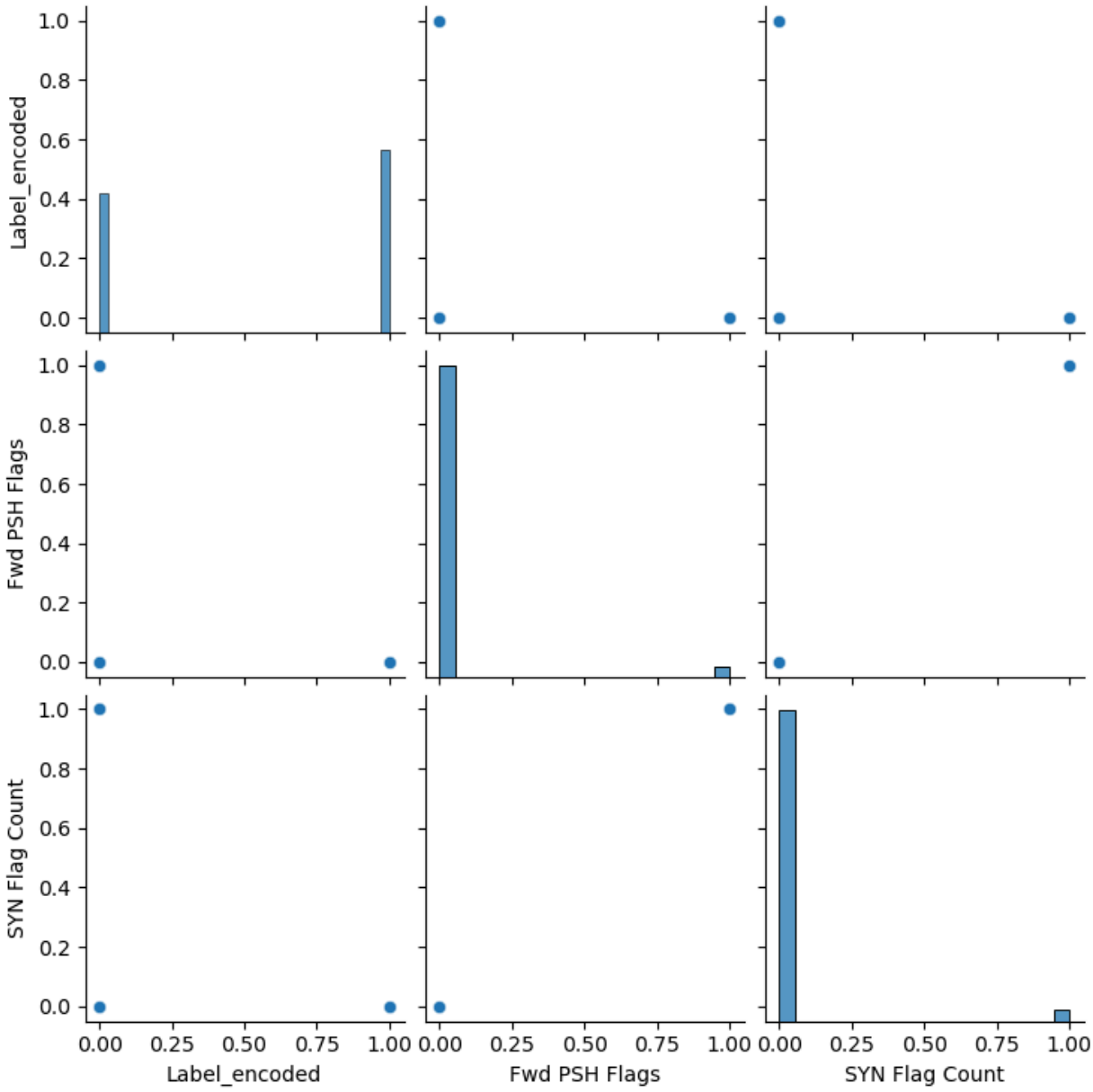

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



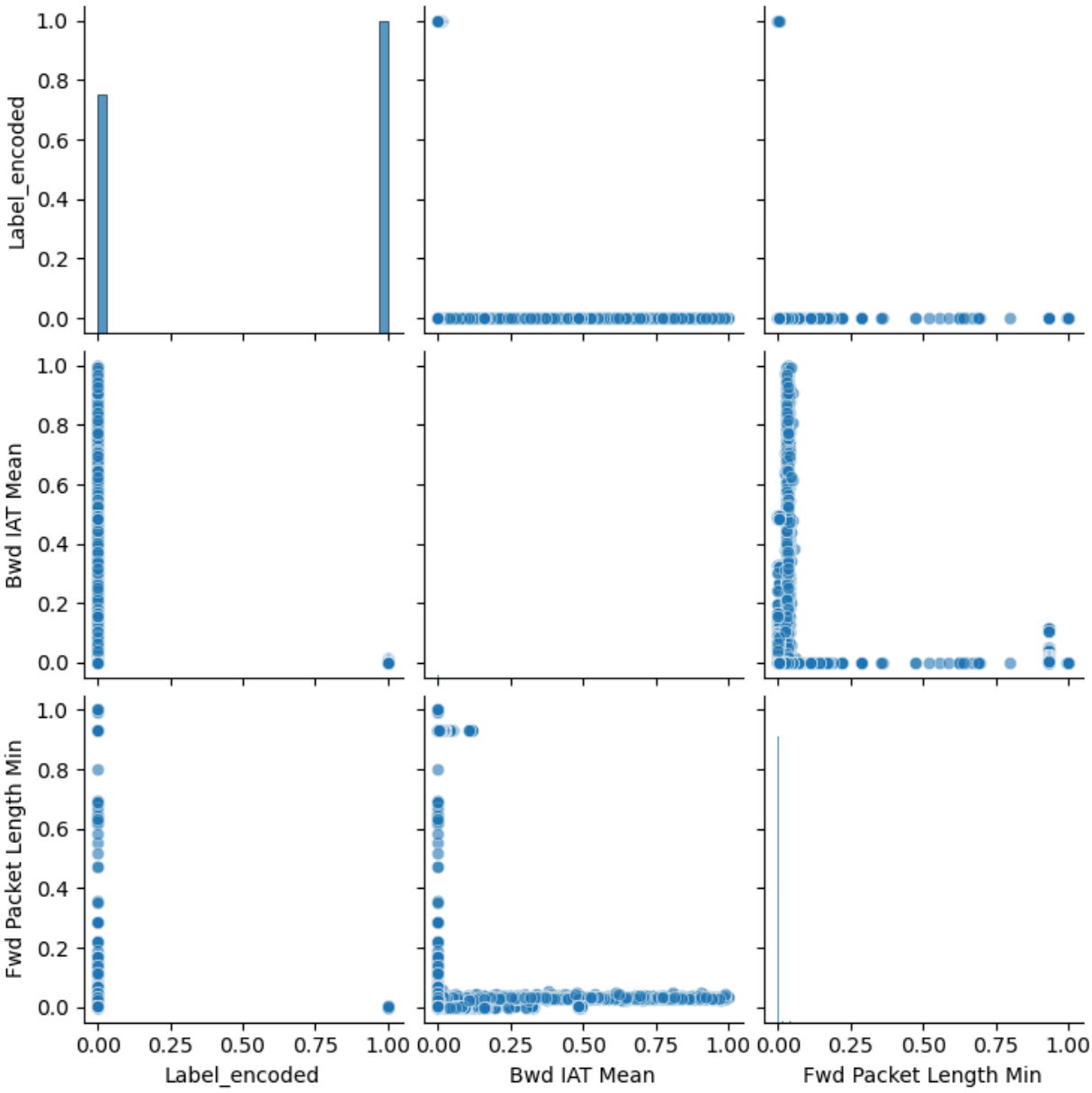

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



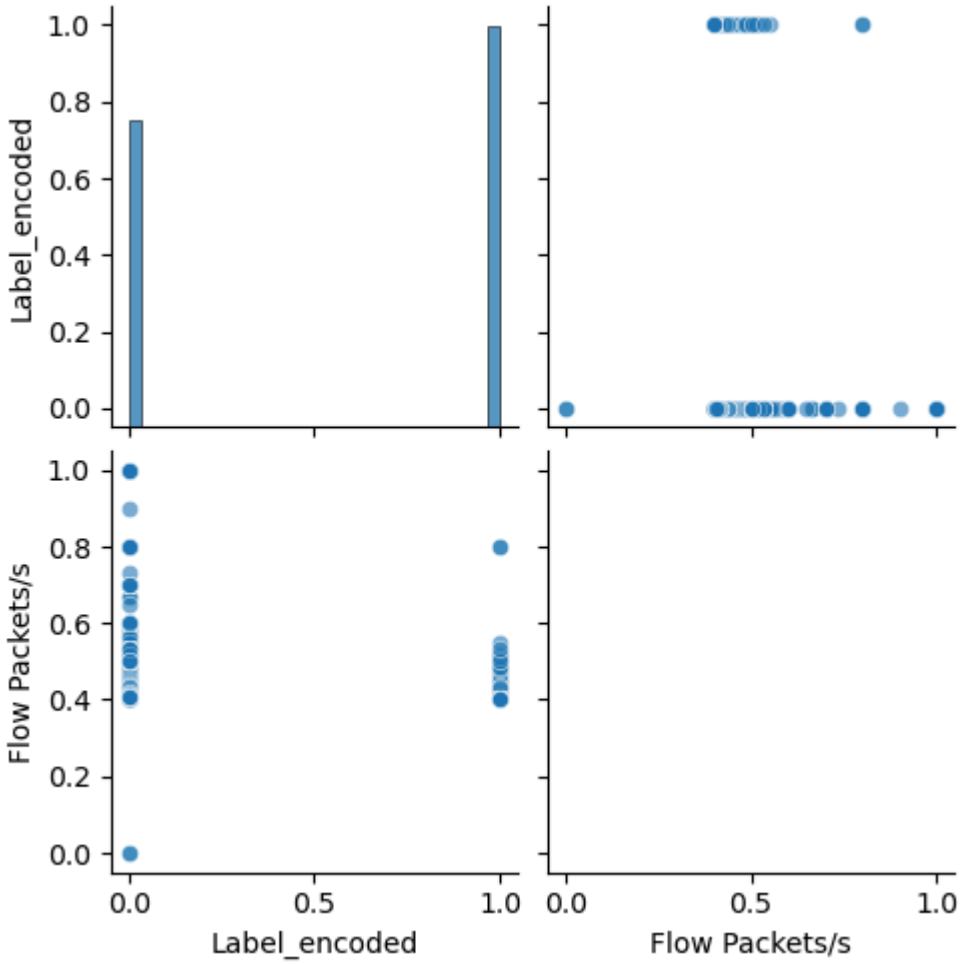

```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```




```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```




```
c:\Users\muham\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The
figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Logistic regression after data normalization

In [ ]:
```
y = df_top_features['Label_encoded']
X = df_top_features.drop(columns=['Label_encoded'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state
print(f'X Shape: {X.shape}')
print(f'y Shape: {y.shape}')
```

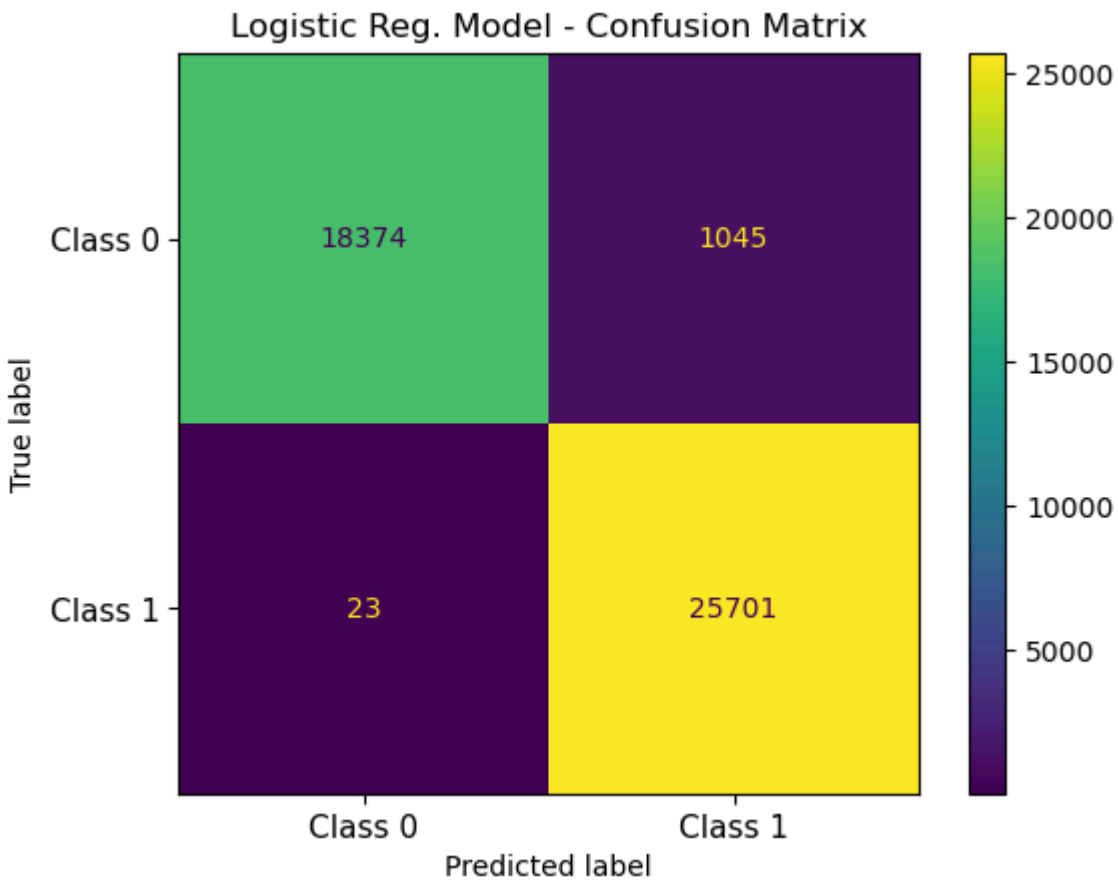
```
X Shape: (225711, 29)
y Shape: (225711,)
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

# Create a logistic regression model using sklearn library
clf=LogisticRegression()
clf.fit(X_train,y_train)

#print score for test data
print(clf.score(X_test,y_test))
```

```
0.9763418470194715
```

In [ ]:
```
from sklearn.metrics import ConfusionMatrixDisplay

cm = ConfusionMatrixDisplay.from_estimator(clf,X_test, y_test)
```

```
#plt.figure()
#plot_confusion_matrix(cm, figsize=(12,8), hide_ticks=True, cmap=plt.cm.Blues)
plt.title("Logistic Reg. Model - Confusion Matrix")
plt.xticks(range(2), ["Class 0","Class 1"], fontsize=11)
plt.yticks(range(2), ["Class 0","Class 1"], fontsize=11)
plt.show()
```



In [ ]:
```
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.98
```