Task 1

In [130]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import cv2
import sklearn.linear_model
```

Make sure to complete all the tasks in assignment 1. It is important to check if you have already generated all the four categories of data frames (for 2 images) so that you can perform ML tasks in this assignment.

In [97]:
```
# Block featureVectors
image01=pd.read_csv("random_merged_image01.csv")
del image01['Unnamed: 0']
print(image01.shape)
image012=pd.read_csv("random_merged_image012.csv")
del image012['Unnamed: 0']
print(image012.shape)
```

```
(288, 257)
(496, 257)
```

In [98]:
```
sw=pd.read_csv("sliding_window_feature_vectors.csv") #sw=sliding window
sw.shape
```

Out[98]:
```
(1829, 257)
```

In [99]:
```
# Sliding window featureVectors
sw_image01= sw[sw["256"].isin([0, 1])]
print(sw_image01.shape)
sw_image012= sw[sw["256"].isin([0, 1,2])]
print(sw_image012.shape)
```

```
(1054, 257)
(1829, 257)
```

In [100]:
```
# Randomize the placement of the Sliding window featureVectors
sw_image01=sw_image01.sample(frac=1).reset_index(drop=True)
print(sw_image01.shape)
sw_image012=sw_image012.sample(frac=1).reset_index(drop=True)
print(sw_image012.shape)
```

```
(1054, 257)
(1829, 257)
```

Divide the data domain of the datasets into 80:20, where 80% is assigned to training datasets and 20% is assigned to test datasets. Save these subsets for all the categories of data frames with suitable file names.

File Name:

1. image01_train.csv
2. image01_test.csv
3. image012_train.csv
4. image012_test.csv
5. sw_image01_train.csv
6. sw_image01_test.csv

7. sw_image012_train.csv
8. sw_image012_test.csv

- image01

In [101]:
```
print(image01.shape)
image01.head(5)
```

```
(288, 257)
```

Out[101]:

	0	1	2	3	4	5	6	7	8	9	...	247	248	249	250	251	252	253	254	255
0	118	118	117	117	116	115	114	114	113	114	...	114	114	112	112	113	113	113	113	113
1	137	109	108	107	106	102	95	101	102	110	...	95	107	97	102	109	110	111	111	110
2	65	64	65	65	64	65	65	65	65	66	...	70	70	70	68	69	68	68	72	74
3	228	228	228	227	227	209	195	237	226	194	...	126	127	128	129	129	125	121	127	121
4	108	108	108	109	109	113	116	118	118	118	...	122	123	123	121	118	117	115	114	114

5 rows × 257 columns

In [102]:
```
row,col=image01.shape
TR=round(row*0.8)
print(TR)
TT=row-TR
print(TT)

image01_train=image01.iloc[0:TR,:]
print(image01_train.shape)
image01_train.to_csv('image01_train.csv', index=False)

image01_test=image01.iloc[TR:row,:]
print(image01_test.shape)
image01_test.to_csv('image01_test.csv', index=False)
```

```
230
58
(230, 257)
(58, 257)
```

- image012

In [103]:
```
print(image012.shape)
image012.head(5)
```

```
(496, 257)
```

Out[103]:

	0	1	2	3	4	5	6	7	8	9	...	247	248	249	250	251	252	253	254	255
0	199	163	150	138	134	153	130	165	162	132	...	137	135	136	136	137	136	137	137	137
1	36	36	35	34	34	34	34	33	33	34	...	33	32	34	32	32	33	32	32	32
2	135	130	129	128	129	130	136	139	138	137	...	135	134	132	133	134	132	132	132	132
3	113	114	113	117	116	116	117	117	117	116	...	215	215	215	207	206	207	217	219	219
4	73	73	72	71	71	70	70	69	70	69	...	79	78	78	76	76	76	74	73	73

5 rows × 257 columns

In [104]:
```
row,col=image012.shape
TR=round(row*0.8)
print(TR)
TT=row-TR
print(TT)

image012_train=image012.iloc[0:TR,:]
print(image012_train.shape)
image012_train.to_csv('image012_train.csv', index=False)

image012_test=image012.iloc[TR:row,:]
print(image012_test.shape)
image012_test.to_csv('image012_test.csv', index=False)
```

```
397
99
(397, 257)
(99, 257)
```

- sw_image01

In [105]:
```
print(sw_image01.shape)
sw_image01.head(5)
```

```
(1054, 257)
```

Out[105]:

	0	1	2	3	4	5	6	7	8	9	...	247	248	249	250	251	252	253	254	255
0	231	233	230	222	193	173	160	159	143	129	...	82	81	81	81	81	80	80	80	82
1	218	217	216	216	220	217	222	224	223	222	...	198	198	199	199	200	203	203	207	210
2	150	150	151	154	159	160	162	160	160	164	...	160	161	161	161	163	164	164	163	162
3	145	152	145	115	120	129	129	132	135	135	...	208	211	204	142	145	157	185	179	191
4	111	114	114	114	132	206	205	188	146	145	...	185	187	194	196	193	187	143	139	132

5 rows × 257 columns

In [106]:
```
row,col=sw_image01.shape
TR=round(row*0.8)
print(TR)
TT=row-TR
print(TT)

sw_image01_train=sw_image01.iloc[0:TR,:]
print(sw_image01_train.shape)
sw_image01_train.to_csv('sw_image01_train.csv', index=False)

sw_image01_test=sw_image01.iloc[TR:row,:]
print(sw_image01_test.shape)
sw_image01_test.to_csv('sw_image01_test.csv', index=False)
```

```
843
211
(843, 257)
(211, 257)
```

- sw_image012

In [107]:
```
print(sw_image012.shape)
sw_image012.head(5)
```

```
(1829, 257)
```

Out[107]:

	0	1	2	3	4	5	6	7	8	9	...	247	248	249	250	251	252	253	254	255
0	66	64	60	54	47	43	40	35	33	34	...	25	26	26	25	25	24	25	25	25
1	119	182	194	201	206	190	177	158	123	97	...	131	159	174	166	172	179	170	180	193
2	29	19	206	216	192	198	0	28	28	24	...	228	184	157	116	118	95	164	155	170
3	161	141	99	121	123	125	173	203	184	170	...	108	104	88	87	81	85	93	97	111
4	32	34	33	33	34	32	31	31	31	31	...	36	35	36	35	36	35	34	35	34

5 rows × 257 columns

In [108]:
```
row,col=sw_image012.shape
TR=round(row*0.8)
print(TR)
TT=row-TR
print(TT)

sw_image012_train=sw_image012.iloc[0:TR,:]
print(sw_image012_train.shape)
sw_image012_train.to_csv('sw_image012_train.csv', index=False)

sw_image012_test=sw_image012.iloc[TR:row,:]
print(sw_image012_test.shape)
sw_image012_test.to_csv('sw_image012_test.csv', index=False)
```

```
1463
366
(1463, 257)
(366, 257)
```

Select two features in each category of training and testing sets and plot their histograms to see if they follow the same distribution – you may determine that from the shape, mean values, and variance.

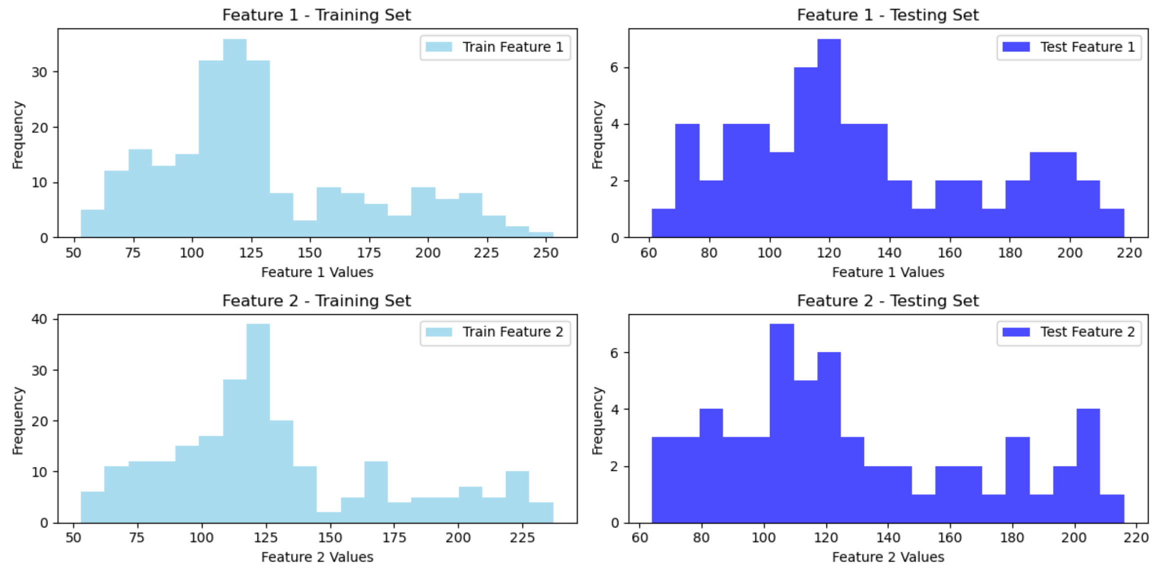
Selected feature: 4 and 7

- image01_train
- image01_test

In [109]:

```
feature_1_train = image01_train.iloc[:, 4]  # Replace 0 with the index of your chosen
feature_2_train = image01_train.iloc[:, 7]  # Replace 1 with the index of your chosen

feature_1_test = image01_test.iloc[:, 4]  # Testing set equivalent for feature 1
feature_2_test = image01_test.iloc[:, 7]  # Testing set equivalent for feature 2

# Compute statistics
train_stats = {
    "Feature 1": (np.mean(feature_1_train), np.var(feature_1_train)),
    "Feature 2": (np.mean(feature_2_train), np.var(feature_2_train))
}
test_stats = {
    "Feature 1": (np.mean(feature_1_test), np.var(feature_1_test)),
    "Feature 2": (np.mean(feature_2_test), np.var(feature_2_test))
}

print("Training set statistics:")
print( train_stats)
print("Testing set statistics:")
print(test_stats)

# Plot histograms
plt.figure(figsize=(12, 6))

# Feature 1
plt.subplot(2, 2, 1)
plt.hist(feature_1_train, bins=20, alpha=0.7, color='skyblue', label="Train Feature 1
plt.xlabel("Feature 1 Values")
plt.ylabel("Frequency")
plt.title("Feature 1 - Training Set")
plt.legend()

plt.subplot(2, 2, 2)
plt.hist(feature_1_test, bins=20, alpha=0.7, color='blue', label="Test Feature 1")
plt.xlabel("Feature 1 Values")
plt.ylabel("Frequency")
plt.title("Feature 1 - Testing Set")
plt.legend()

# Feature 2
plt.subplot(2, 2, 3)
plt.hist(feature_2_train, bins=20, alpha=0.7, color='skyblue', label="Train Feature 2
plt.xlabel("Feature 2 Values")
plt.ylabel("Frequency")
plt.title("Feature 2 - Training Set")
plt.legend()

plt.subplot(2, 2, 4)
plt.hist(feature_2_test, bins=20, alpha=0.7, color='blue', label="Test Feature 2")
plt.xlabel("Feature 2 Values")
plt.ylabel("Frequency")
plt.title("Feature 2 - Testing Set")
plt.legend()

plt.tight_layout()
plt.show()
```

```
Training set statistics:
{'Feature 1': (127.8391304347826, 1931.6132514177693), 'Feature 2': (128.87391304347
827, 1952.3971455576552)}
Testing set statistics:
{'Feature 1': (130.08620689655172, 1649.1132580261597), 'Feature 2': (128.5517241379
3105, 1747.9024970273483)}
```

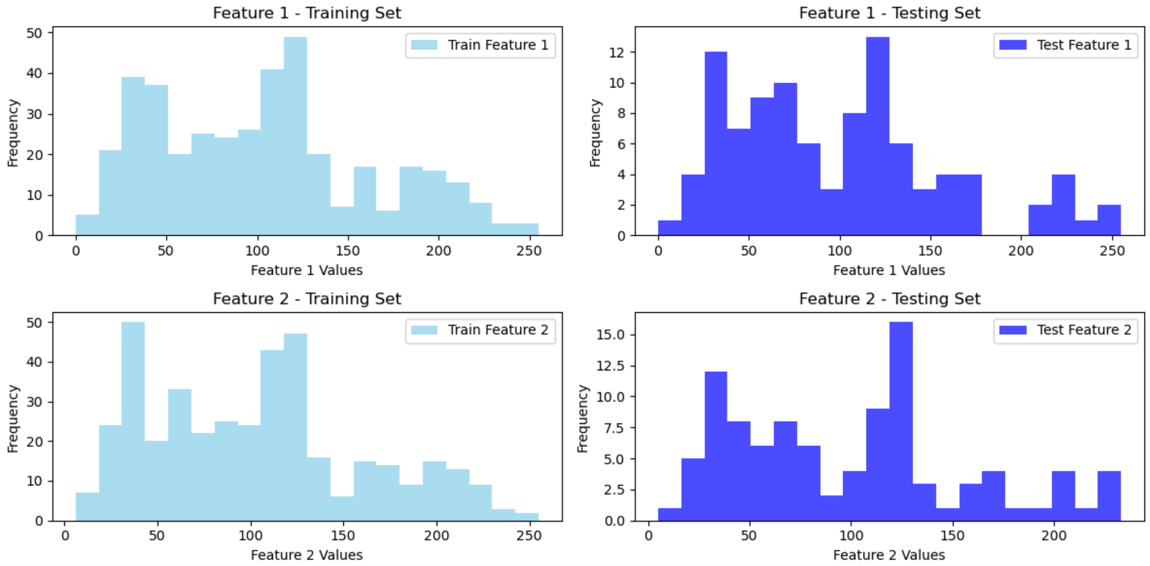


- image012_train
- image012_test

In [110]:

```
feature_1_train = image012_train.iloc[:, 4]  # Replace 0 with the index of your chose
feature_2_train = image012_train.iloc[:, 7]  # Replace 1 with the index of your chose

feature_1_test = image012_test.iloc[:, 4]  # Testing set equivalent for feature 1
feature_2_test = image012_test.iloc[:, 7]  # Testing set equivalent for feature 2

# Compute statistics
train_stats = {
    "Feature 1": (np.mean(feature_1_train), np.var(feature_1_train)),
    "Feature 2": (np.mean(feature_2_train), np.var(feature_2_train))
}
test_stats = {
    "Feature 1": (np.mean(feature_1_test), np.var(feature_1_test)),
    "Feature 2": (np.mean(feature_2_test), np.var(feature_2_test))
}

print("Training set statistics:")
print( train_stats)
print("Testing set statistics:")
print(test_stats)

# Plot histograms
plt.figure(figsize=(12, 6))

# Feature 1
plt.subplot(2, 2, 1)
plt.hist(feature_1_train, bins=20, alpha=0.7, color='skyblue', label="Train Feature 1
plt.xlabel("Feature 1 Values")
plt.ylabel("Frequency")
plt.title("Feature 1 - Training Set")
plt.legend()

plt.subplot(2, 2, 2)
plt.hist(feature_1_test, bins=20, alpha=0.7, color='blue', label="Test Feature 1")
plt.xlabel("Feature 1 Values")
plt.ylabel("Frequency")
plt.title("Feature 1 - Testing Set")
plt.legend()

# Feature 2
plt.subplot(2, 2, 3)
plt.hist(feature_2_train, bins=20, alpha=0.7, color='skyblue', label="Train Feature 2
plt.xlabel("Feature 2 Values")
plt.ylabel("Frequency")
plt.title("Feature 2 - Training Set")
plt.legend()

plt.subplot(2, 2, 4)
plt.hist(feature_2_test, bins=20, alpha=0.7, color='blue', label="Test Feature 2")
plt.xlabel("Feature 2 Values")
plt.ylabel("Frequency")
plt.title("Feature 2 - Testing Set")
plt.legend()

plt.tight_layout()
plt.show()
```

```
Training set statistics:
{'Feature 1': (102.639798488665, 3421.5755445437735), 'Feature 2': (103.198992443324
94, 3351.7563717808002)}
Testing set statistics:
{'Feature 1': (100.0909090909091, 3400.951331496786), 'Feature 2': (99.6363636363636
4, 3228.514233241506)}
```

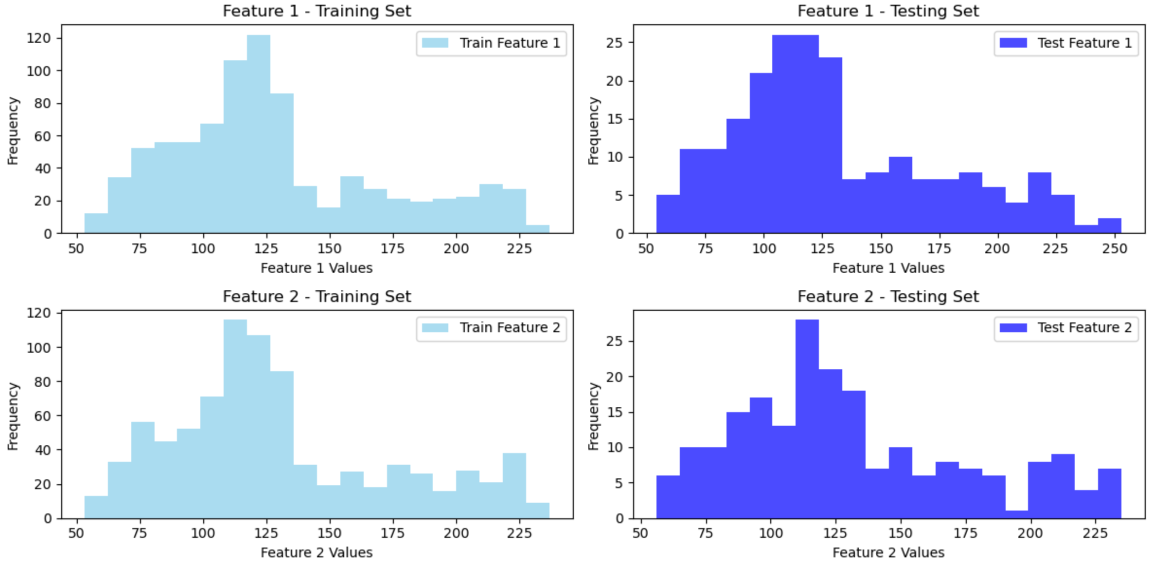


- sw_image01_train
- sw_image01_test

In [111]:

```
feature_1_train = sw_image01_train.iloc[:, 4]  # Replace 0 with the index of your cho
feature_2_train = sw_image01_train.iloc[:, 7]  # Replace 1 with the index of your cho

feature_1_test = sw_image01_test.iloc[:, 4]  # Testing set equivalent for feature 1
feature_2_test = sw_image01_test.iloc[:, 7]  # Testing set equivalent for feature 2

# Compute statistics
train_stats = {
    "Feature 1": (np.mean(feature_1_train), np.var(feature_1_train)),
    "Feature 2": (np.mean(feature_2_train), np.var(feature_2_train))
}
test_stats = {
    "Feature 1": (np.mean(feature_1_test), np.var(feature_1_test)),
    "Feature 2": (np.mean(feature_2_test), np.var(feature_2_test))
}

print("Training set statistics:")
print( train_stats)
print("Testing set statistics:")
print(test_stats)

# Plot histograms
plt.figure(figsize=(12, 6))

# Feature 1
plt.subplot(2, 2, 1)
plt.hist(feature_1_train, bins=20, alpha=0.7, color='skyblue', label="Train Feature 1
plt.xlabel("Feature 1 Values")
plt.ylabel("Frequency")
plt.title("Feature 1 - Training Set")
plt.legend()

plt.subplot(2, 2, 2)
plt.hist(feature_1_test, bins=20, alpha=0.7, color='blue', label="Test Feature 1")
plt.xlabel("Feature 1 Values")
plt.ylabel("Frequency")
plt.title("Feature 1 - Testing Set")
plt.legend()

# Feature 2
plt.subplot(2, 2, 3)
plt.hist(feature_2_train, bins=20, alpha=0.7, color='skyblue', label="Train Feature 2
plt.xlabel("Feature 2 Values")
plt.ylabel("Frequency")
plt.title("Feature 2 - Training Set")
plt.legend()

plt.subplot(2, 2, 4)
plt.hist(feature_2_test, bins=20, alpha=0.7, color='blue', label="Test Feature 2")
plt.xlabel("Feature 2 Values")
plt.ylabel("Frequency")
plt.title("Feature 2 - Testing Set")
plt.legend()

plt.tight_layout()
plt.show()
```

```
Training set statistics:
{'Feature 1': (128.15183867141164, 1777.8678109727864), 'Feature 2': (129.9062870699
8814, 1914.9461407811748)}
Testing set statistics:
{'Feature 1': (130.32701421800948, 2026.836189663305), 'Feature 2': (131.19905213270
14, 2034.5575346465725)}
```

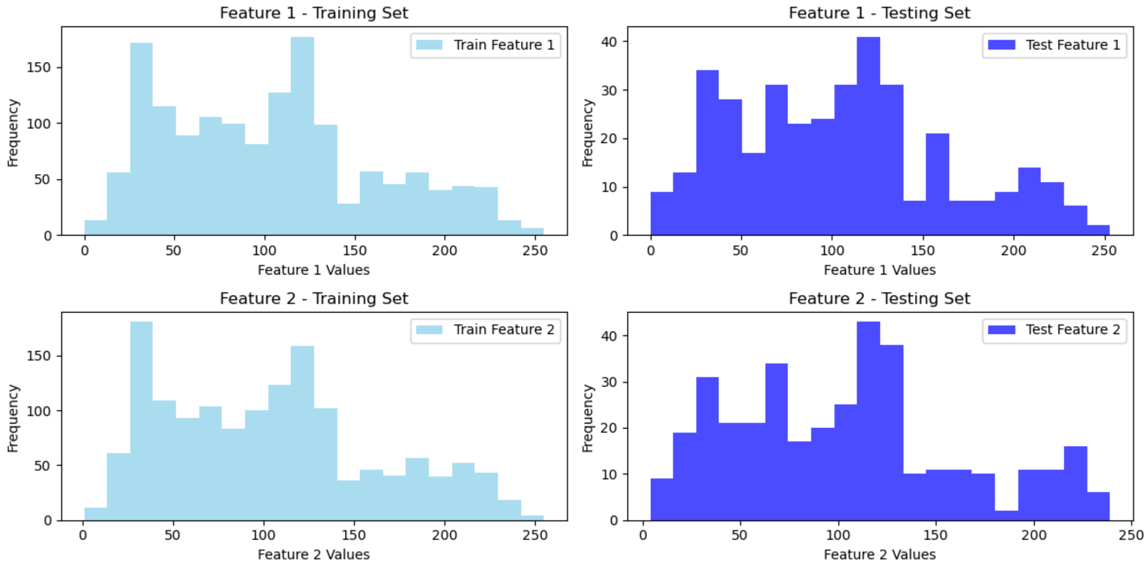


- sw_image012_train
- sw_image012_test

In [112]:

```
feature_1_train = sw_image012_train.iloc[:, 4]  # Replace 0 with the index of your ch
feature_2_train = sw_image012_train.iloc[:, 7]  # Replace 1 with the index of your ch

feature_1_test = sw_image012_test.iloc[:, 4]  # Testing set equivalent for feature 1
feature_2_test = sw_image012_test.iloc[:, 7]  # Testing set equivalent for feature 2

# Compute statistics
train_stats = {
    "Feature 1": (np.mean(feature_1_train), np.var(feature_1_train)),
    "Feature 2": (np.mean(feature_2_train), np.var(feature_2_train))
}
test_stats = {
    "Feature 1": (np.mean(feature_1_test), np.var(feature_1_test)),
    "Feature 2": (np.mean(feature_2_test), np.var(feature_2_test))
}

print("Training set statistics:")
print( train_stats)
print("Testing set statistics:")
print(test_stats)

# Plot histograms
plt.figure(figsize=(12, 6))

# Feature 1
plt.subplot(2, 2, 1)
plt.hist(feature_1_train, bins=20, alpha=0.7, color='skyblue', label="Train Feature 1
plt.xlabel("Feature 1 Values")
plt.ylabel("Frequency")
plt.title("Feature 1 - Training Set")
plt.legend()

plt.subplot(2, 2, 2)
plt.hist(feature_1_test, bins=20, alpha=0.7, color='blue', label="Test Feature 1")
plt.xlabel("Feature 1 Values")
plt.ylabel("Frequency")
plt.title("Feature 1 - Testing Set")
plt.legend()

# Feature 2
plt.subplot(2, 2, 3)
plt.hist(feature_2_train, bins=20, alpha=0.7, color='skyblue', label="Train Feature 2
plt.xlabel("Feature 2 Values")
plt.ylabel("Frequency")
plt.title("Feature 2 - Training Set")
plt.legend()

plt.subplot(2, 2, 4)
plt.hist(feature_2_test, bins=20, alpha=0.7, color='blue', label="Test Feature 2")
plt.xlabel("Feature 2 Values")
plt.ylabel("Frequency")
plt.title("Feature 2 - Testing Set")
plt.legend()

plt.tight_layout()
plt.show()
```

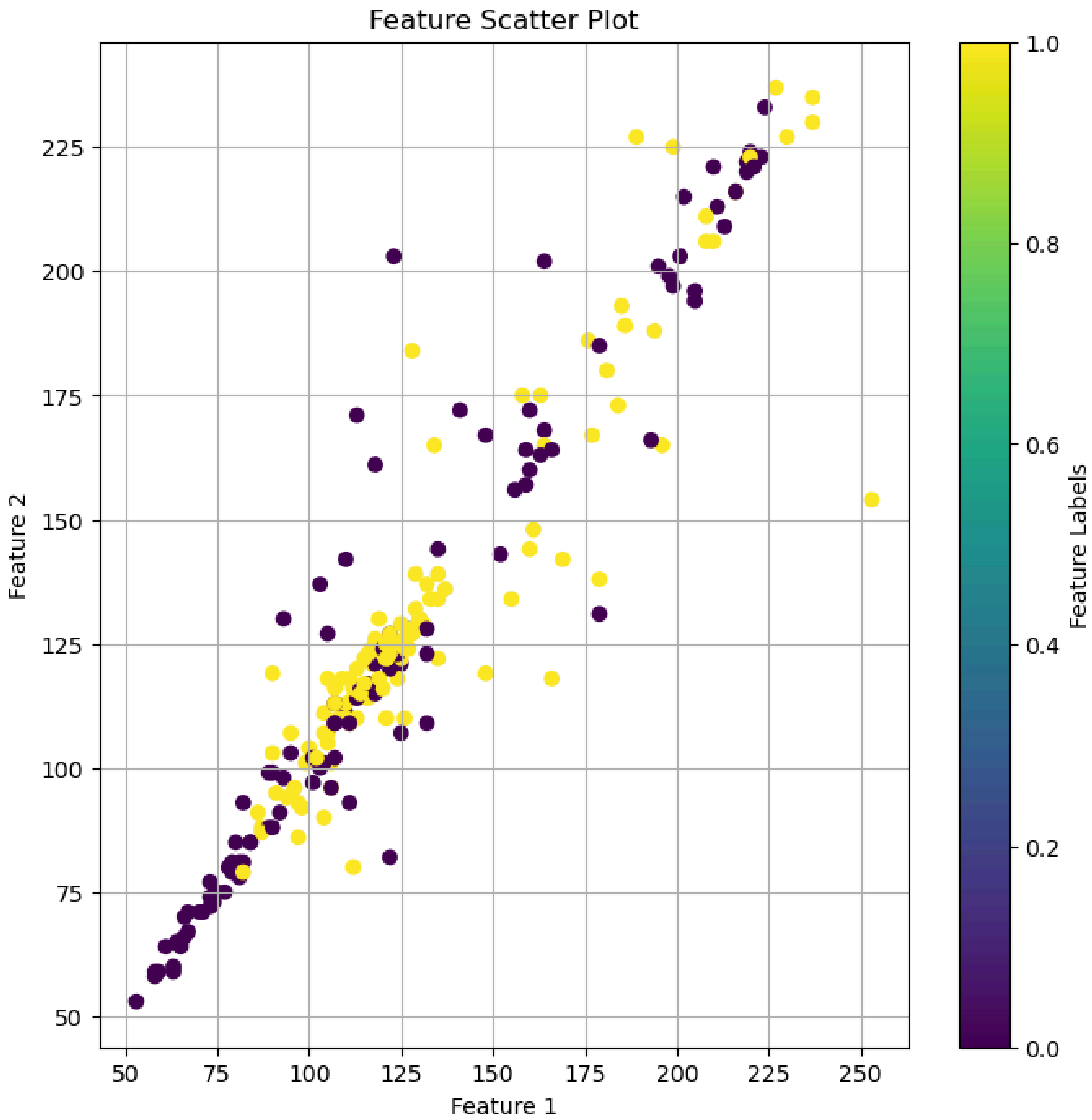
```
Training set statistics:
{'Feature 1': (103.11346548188654, 3345.099907539307), 'Feature 2': (103.86534518113
466, 3435.078928913665)}
Testing set statistics:
{'Feature 1': (104.30601092896175, 3424.256084087313), 'Feature 2': (103.67486338797
814, 3368.5363626862586)}
```



Use the same two features to generate scatter plots for each category of training and testing datasets as well. Highlight the corresponding labels with distinct colors. If the plot is too dense then use subsets.

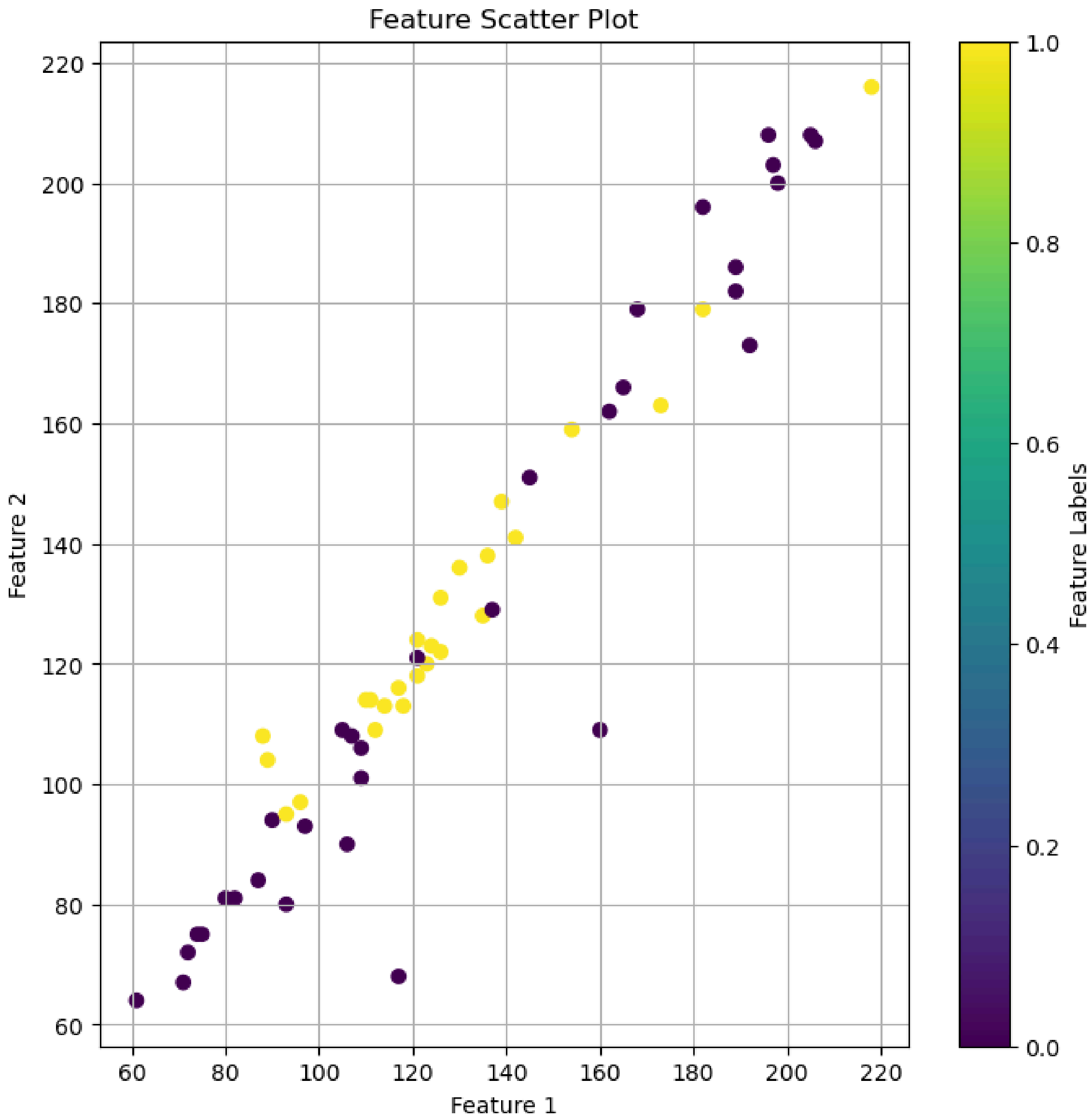
- image01_train

In [113]:
```
y_image01_train=image01_train['256']

plt.figure(figsize=(8, 8))
plt.scatter(image01_train.iloc[:, 4], image01_train.iloc[:, 7], c=y_image01_train, cm
plt.title('Feature Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Feature Labels')  # Add a colorbar to indicate the meaning of the
plt.grid(True)
plt.show()
```



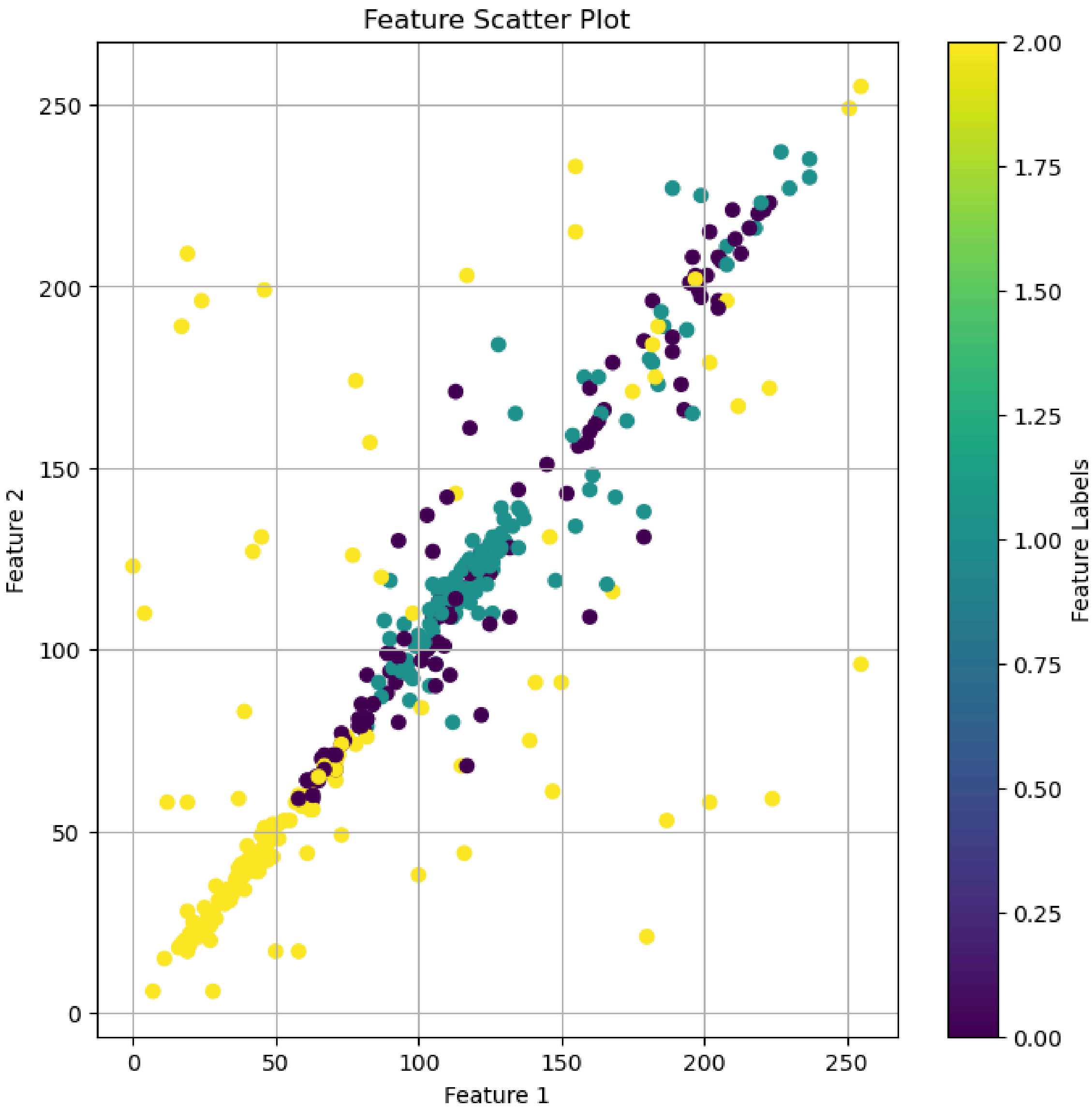
- image01_test

In [114]:
```
y_image01_test=image01_test['256']

plt.figure(figsize=(8, 8))
plt.scatter(image01_test.iloc[:, 4], image01_test.iloc[:, 7], c=y_image01_test, cmap=
plt.title('Feature Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Feature Labels')  # Add a colorbar to indicate the meaning of the
plt.grid(True)
plt.show()
```



- image012_train

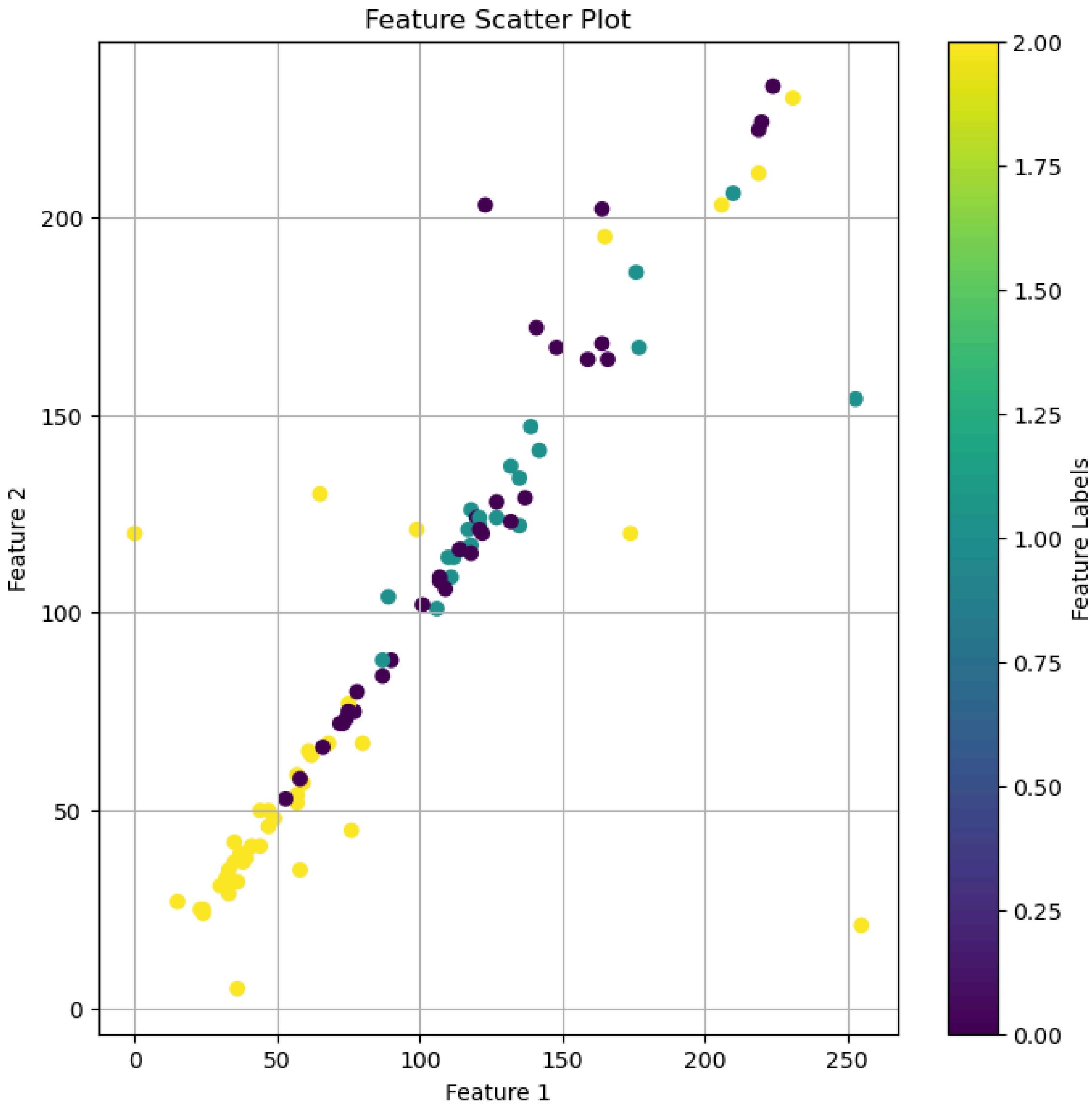
In [115]:
```
y_image012_train=image012_train['256']

plt.figure(figsize=(8, 8))
plt.scatter(image012_train.iloc[:, 4], image012_train.iloc[:, 7], c=y_image012_train,
plt.title('Feature Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Feature Labels')  # Add a colorbar to indicate the meaning of the
plt.grid(True)
plt.show()
```



- image012_test

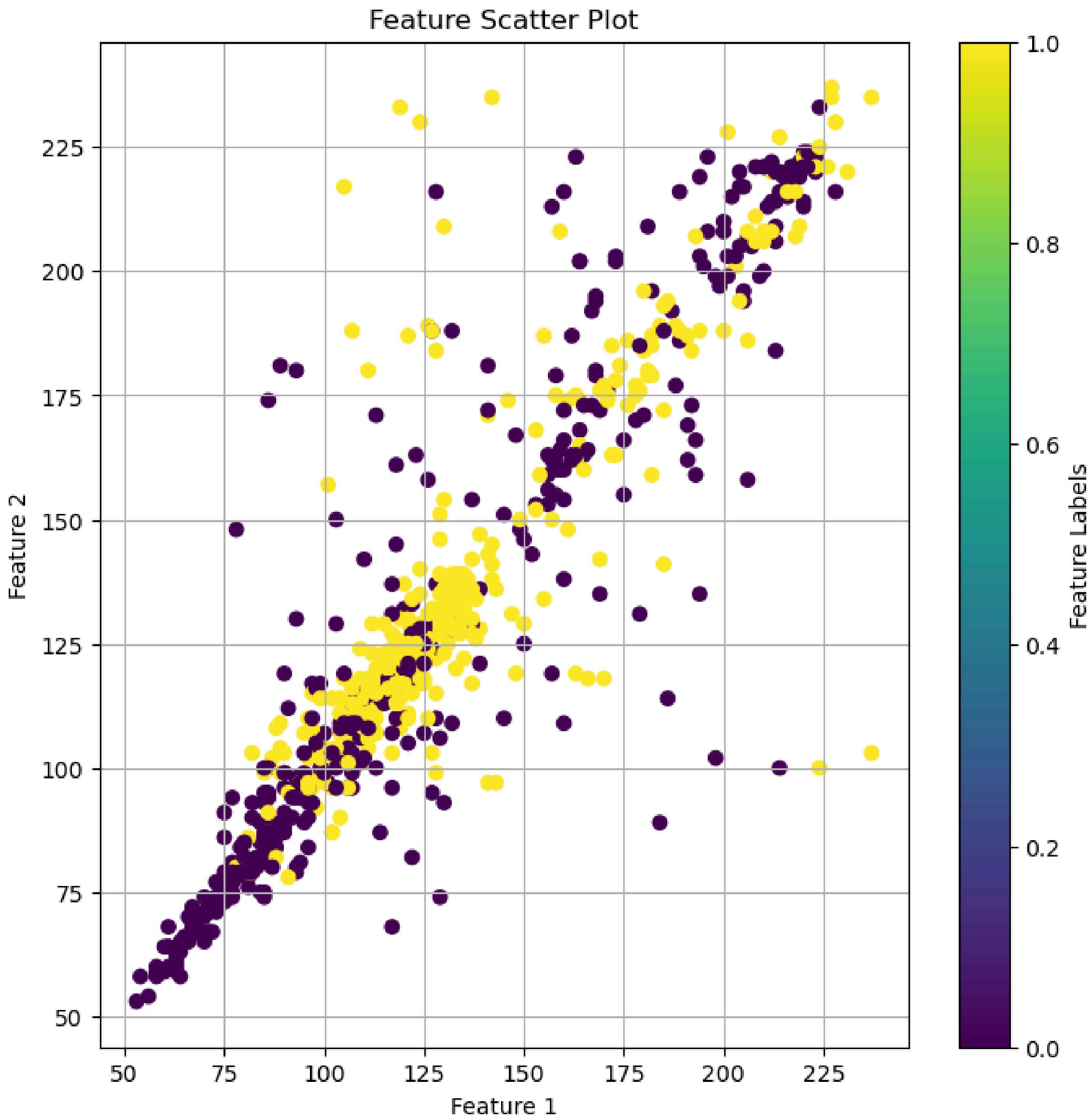
In [116]:

```
y_image012_test=image012_test['256']

plt.figure(figsize=(8, 8))
plt.scatter(image012_test.iloc[:, 4], image012_test.iloc[:, 7], c=y_image012_test, cm
plt.title('Feature Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Feature Labels')  # Add a colorbar to indicate the meaning of the
plt.grid(True)
plt.show()
```



- sw_image01_train

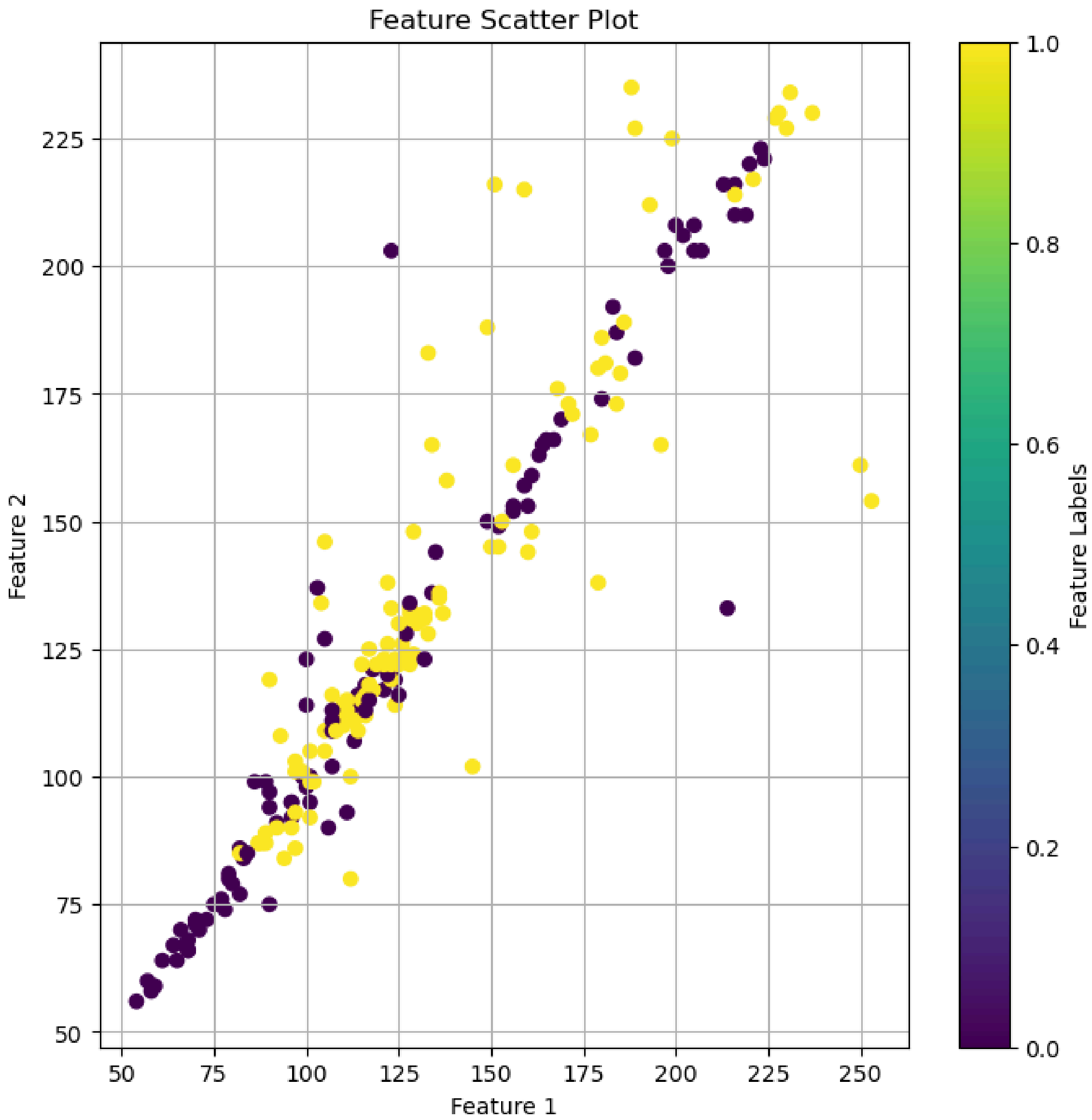
In [117]:
```
y_sw_image01_train=sw_image01_train['256']

plt.figure(figsize=(8, 8))
plt.scatter(sw_image01_train.iloc[:, 4], sw_image01_train.iloc[:, 7], c=y_sw_image01_
plt.title('Feature Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Feature Labels')  # Add a colorbar to indicate the meaning of the
plt.grid(True)
plt.show()
```



- sw_image01_test

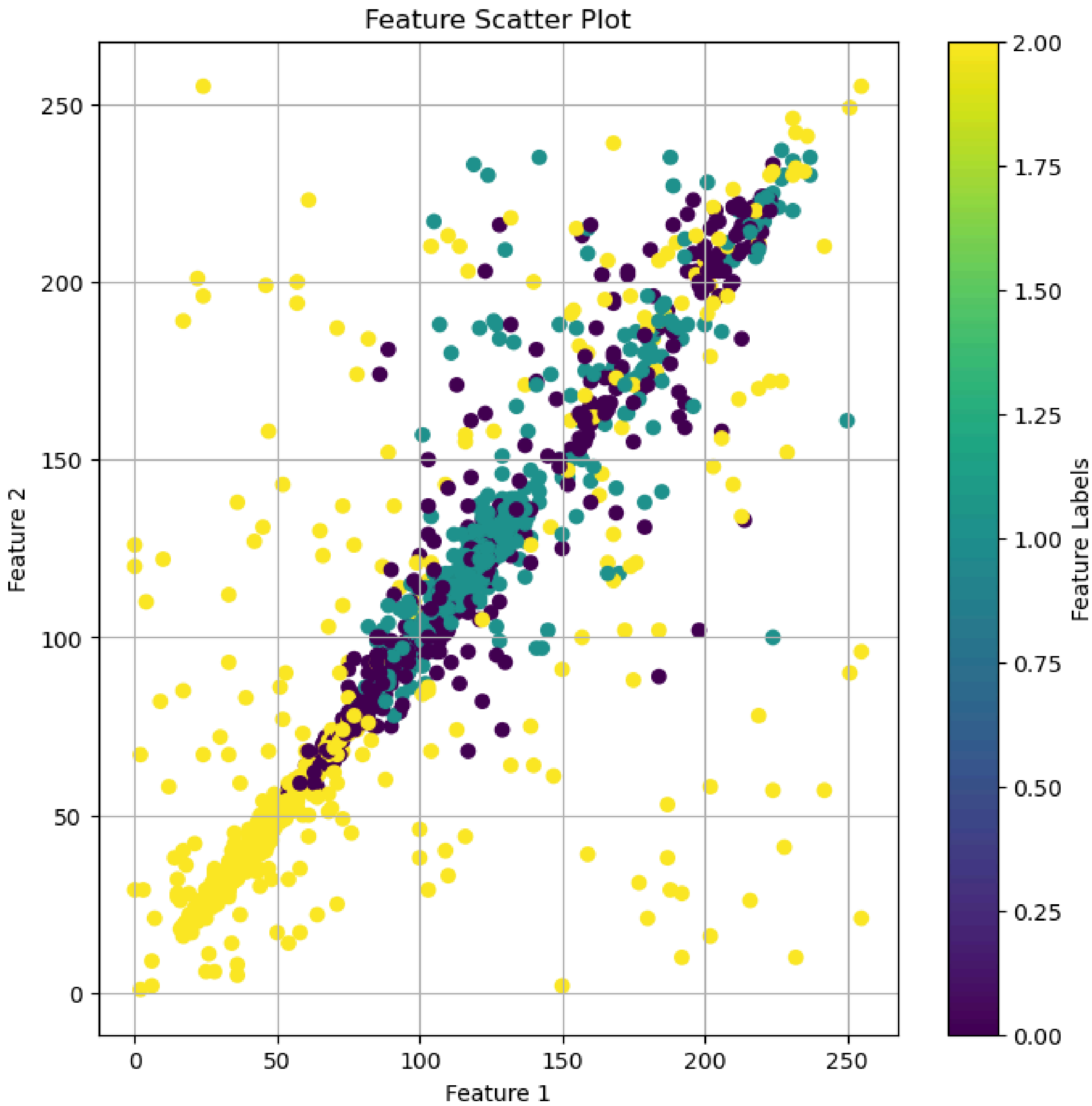
In [118]:

```
y_sw_image01_test=sw_image01_test['256']

plt.figure(figsize=(8, 8))
plt.scatter(sw_image01_test.iloc[:, 4], sw_image01_test.iloc[:, 7], c=y_sw_image01_te
plt.title('Feature Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Feature Labels')  # Add a colorbar to indicate the meaning of the
plt.grid(True)
plt.show()
```



- sw_image012_train

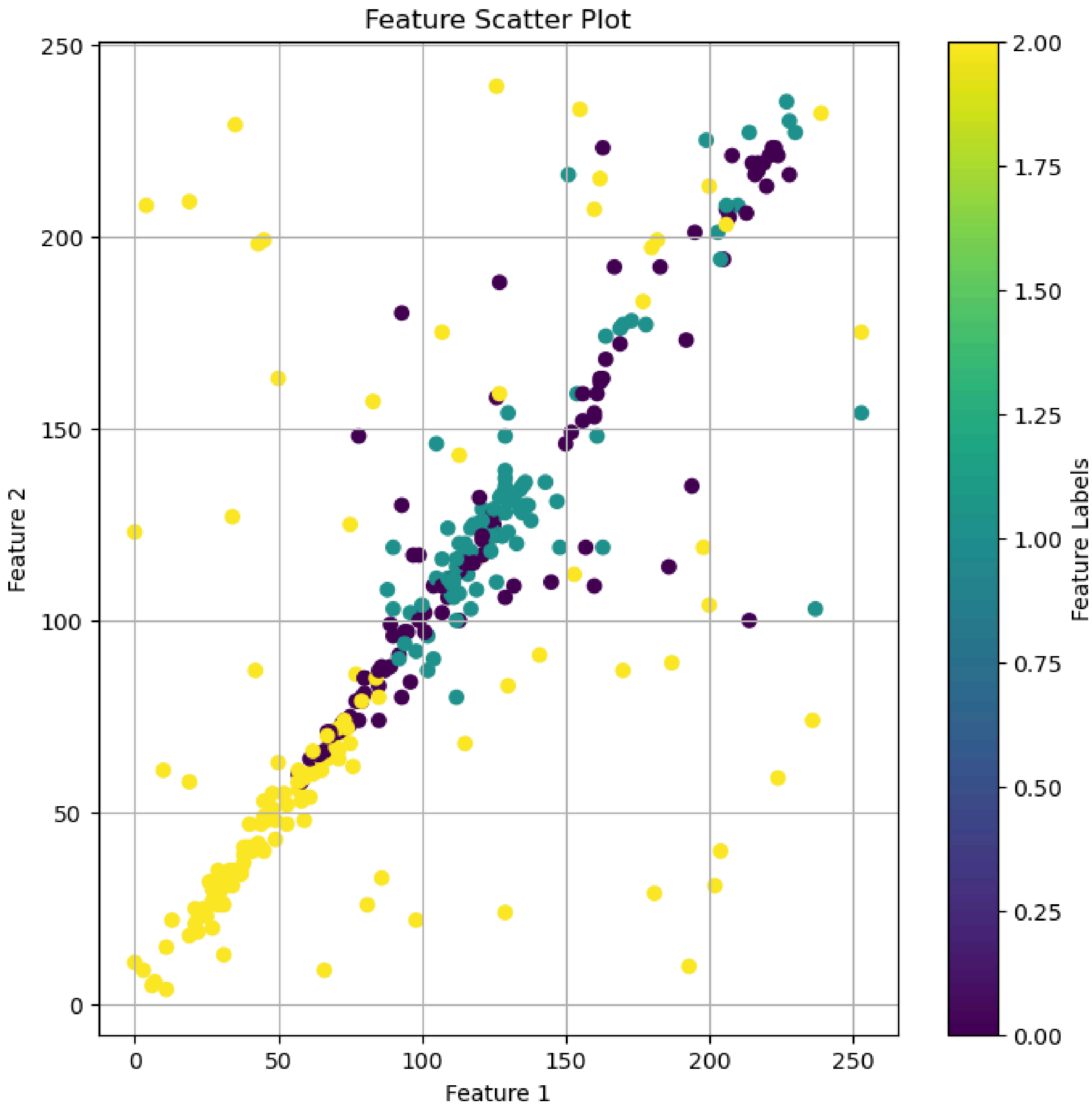
In [119]:
```
y_sw_image012_train=sw_image012_train['256']

plt.figure(figsize=(8, 8))
plt.scatter(sw_image012_train.iloc[:, 4], sw_image012_train.iloc[:, 7], c=y_sw_image0
plt.title('Feature Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Feature Labels')  # Add a colorbar to indicate the meaning of the
plt.grid(True)
plt.show()
```



- sw_image012_test

In [120]:

```
y_sw_image012_test=sw_image012_test['256']

plt.figure(figsize=(8, 8))
plt.scatter(sw_image012_test.iloc[:, 4], sw_image012_test.iloc[:, 7], c=y_sw_image012
plt.title('Feature Scatter Plot')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.colorbar(label='Feature Labels')  # Add a colorbar to indicate the meaning of the
plt.grid(True)
plt.show()
```



Task 2

Implement and train lasso regression or elastic-net regression as a two-class classifier using the training sets of the feature vectors (feature spaces) that you created.

image01

In [135]:
```
from sklearn.linear_model import Lasso
```

In [377]:
```
x_image01_train=image01_train.iloc[:,0:256]
#x_image01_train
y_image01_train=image01_train["256"]
#y_image01_train

reg = Lasso(alpha=0.08)
model_image01 = reg.fit(x_image01_train, y_image01_train)
model_image01
```

```
C:\Users\saksh\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:647: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations, check the scale of the features or consider increasing regu
larisation. Duality gap: 4.060e-01, tolerance: 5.743e-03
  model = cd_fast.enet_coordinate_descent(
```

Out[377]: Lasso(alpha=0.08)

image012

In [151]:
```
x_image012_train=image012_train.iloc[:,0:256]
y_image012_train=image012_train["256"]

reg = Lasso(alpha=0.08)
model_image012 = reg.fit(x_image012_train, y_image012_train)
model_image012
```

```
C:\Users\saksh\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:647: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations, check the scale of the features or consider increasing regu
larisation. Duality gap: 4.957e+00, tolerance: 2.674e-02
  model = cd_fast.enet_coordinate_descent(
```

Out[151]: Lasso(alpha=0.08)

sw_image01

In [137]:
```
x_sw_image01_train=sw_image01_train.iloc[:,0:256]
y_sw_image01_train=sw_image01_train["256"]

reg = Lasso(alpha=0.08)
model_sw_image01 = reg.fit(x_sw_image01_train, y_sw_image01_train)
model_sw_image01
```

```
C:\Users\saksh\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:647: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations, check the scale of the features or consider increasing regu
larisation. Duality gap: 3.618e-01, tolerance: 2.107e-02
  model = cd_fast.enet_coordinate_descent(
```

Out[137]: Lasso(alpha=0.08)

sw_image012

In [138]:
```
x_sw_image012_train=sw_image012_train.iloc[:,0:256]
y_sw_image012_train=sw_image012_train["256"]

reg = Lasso(alpha=0.08)
model_sw_image012 = reg.fit(x_sw_image012_train, y_sw_image012_train)
model_sw_image012
```

```
C:\Users\saksh\anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:647: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations, check the scale of the features or consider increasing regu
larisation. Duality gap: 3.785e+00, tolerance: 1.011e-01
  model = cd_fast.enet_coordinate_descent(
```

Out[138]:
```
Lasso(alpha=0.08)
```

Apply the trained models to the test sets of all the categories of datasets and add the predicted labels next to their actual labels in the corresponding spreadsheets.

image01

In [286]:
```
x_image01_test=image01_test.iloc[:,0:256]
y_image01_test=image01_test["256"]

yhat_image01_test_lasso = model_image01.predict(x_image01_test)
yhat_image01_test_lasso = yhat_image01_test_lasso.round()

yhat_image01_test_lasso=pd.DataFrame(yhat_image01_test_lasso)

image01_test = image01_test.reset_index(drop=True)
yhat_image01_test_lasso = yhat_image01_test_lasso.reset_index(drop=True)

image01_test_predict=pd.concat([image01_test,yhat_image01_test_lasso], axis=1)
image01_test_predict.columns.values[-1] = 'predict'
image01_test_predict.to_csv('image01_test_predict.csv', index=False)
```

image012

In [241]:
```
x_image012_test=image012_test.iloc[:,0:256]
y_image012_test=image012_test["256"]

yhat_image012_test_lasso = model_image012.predict(x_image012_test)
yhat_image012_test_lasso = yhat_image012_test_lasso.round()

yhat_image012_test_lasso=pd.DataFrame(yhat_image012_test_lasso)

image012_test = image012_test.reset_index(drop=True)
yhat_image012_test_lasso = yhat_image012_test_lasso.reset_index(drop=True)

image012_test_predict=pd.concat([image012_test,yhat_image012_test_lasso], axis=1)
image012_test_predict.columns.values[-1] = 'predict'
image012_test_predict.to_csv('image012_test_predict.csv', index=False)
```

sw_image01

In [242]:
```
x_sw_image01_test=sw_image01_test.iloc[:,0:256]
y_sw_image01_test=sw_image01_test["256"]

yhat_sw_image01_test_lasso = model_sw_image01.predict(x_sw_image01_test)
yhat_sw_image01_test_lasso = yhat_sw_image01_test_lasso.round()

yhat_sw_image01_test_lasso=pd.DataFrame(yhat_sw_image01_test_lasso)

sw_image01_test = sw_image01_test.reset_index(drop=True)
yhat_sw_image01_test_lasso = yhat_sw_image01_test_lasso.reset_index(drop=True)

sw_image01_test_predict=pd.concat([sw_image01_test,yhat_sw_image01_test_lasso], axis=
sw_image01_test_predict.columns.values[-1] = 'predict'
sw_image01_test_predict.to_csv('sw_image01_test_predict.csv', index=False)
```

sw_image012

In [243]:
```
x_sw_image012_test=sw_image012_test.iloc[:,0:256]
y_sw_image012_test=sw_image012_test["256"]

yhat_sw_image012_test_lasso = model_sw_image012.predict(x_sw_image012_test)
yhat_sw_image012_test_lasso = yhat_sw_image012_test_lasso.round()

yhat_sw_image012_test_lasso=pd.DataFrame(yhat_sw_image012_test_lasso)

sw_image012_test = sw_image012_test.reset_index(drop=True)
yhat_sw_image012_test_lasso = yhat_sw_image012_test_lasso.reset_index(drop=True)

sw_image012_test_predict=pd.concat([sw_image012_test,yhat_sw_image012_test_lasso], ax
sw_image012_test_predict.columns.values[-1] = 'predict'
sw_image012_test_predict.to_csv('sw_image012_test_predict.csv', index=False)
```

Construct confusion matrices using the responses in the actual and predicted label columns of the test datasets for all the four categories of data frames. Don’t forget to save these confusion matrices…!!!

In [180]:
```
from sklearn.metrics import confusion_matrix
```

image01

In [244]:
```
cm_image01 = confusion_matrix(y_image01_test, yhat_image01_test_lasso, labels=[0,1])
cm_image01
```

Out[244]:
```
array([[16, 17],
       [ 6, 18]], dtype=int64)
```

image012

In [245]:
```
cm_image012 = confusion_matrix(y_image012_test, yhat_image012_test_lasso, labels=[0,1
cm_image012
```

Out[245]:
```
array([[ 8, 23,  2],
       [ 6, 15,  1],
       [ 5, 12, 21]], dtype=int64)
```

sw_image01

In [246]:
```
cm_sw_image01 = confusion_matrix(y_sw_image01_test, yhat_sw_image01_test_lasso, label
cm_sw_image01
```

Out[246]:
```
array([[55, 43],
       [55, 58]], dtype=int64)
```

sw_image012

In [247]:
```
cm_sw_image012 = confusion_matrix(y_sw_image012_test, yhat_sw_image012_test_lasso, la
cm_sw_image012
```

Out[247]:
```
array([[31, 77,  1],
       [20, 83,  0],
       [16, 40, 92]], dtype=int64)
```

Select two qualitative measures that use the concept of confusion matrix to quantify the performance of a machine learning model. Use them to compare the model's performance for all the categories of datasets.

There are four different qualitative measures:

1. accuracy
2. precision
3. sensitivity
4. specificity

Accuracy: It describes the performance of the model based on the proportionality between the false positive and the true positive. If the accuracy is high, then it means that the classification of both classes are highly accurate (it is indicated by the double lines), and the false negative and false positives are ignorable.

Precision: It describes the performance of the model based on the proportionality between the false positives and the true positives. If the precision is high, then it means that the classification of A is precisely high (double line) with low false negatives

image01

In [306]:
```
TN = cm_image01[1,1]
FP = cm_image01[1,0]
FN = cm_image01[0,1]
TP = cm_image01[0,0]
FPFN = FP+FN
TPTN = TP+TN

Accuracy_image01_lasso = 1/(1+(FPFN/TPTN))
print("Accuracy:",Accuracy_image01_lasso)

Precision_image01_lasso = 1/(1+(FP/TP))
print("Precision:",Precision_image01_lasso)
```

```
Accuracy: 0.5964912280701754
Precision: 0.7272727272727273
```

image012

In [331]:

```
TP0=cm_image012[0,0]
TP1=cm_image012[1,1]
TP2=cm_image012[2,2]
E10=cm_image012[1,0]
E20=cm_image012[2,0]
E21=cm_image012[2,1]
E01=cm_image012[0,1]
E02=cm_image012[0,2]
E12=cm_image012[1,2]

n=TP0+TP1+TP2+E10+E20+E21+E01+E02+E12
Accuracy_image012_lasso = (TP0+TP1+TP2)/n
print("Accuracy:",Accuracy_image012_lasso)

p0=TP0/(TP0+E10+E20)
p1=TP1/(TP1+E01+E21)
p2=TP2/(TP2+E02+E12)
Precision_image012_lasso =(p0+p1+p2)/3
print("Precision Macro:",Precision_image012_lasso)
```

```
Accuracy: 0.4731182795698925
Precision Macro: 0.5320175438596492
```

sw_image01

In [308]:

```
TN = cm_sw_image01[1,1]
FP = cm_sw_image01[1,0]
FN = cm_sw_image01[0,1]
TP = cm_sw_image01[0,0]
FPFN = FP+FN
TPTN = TP+TN

Accuracy_sw_image01_lasso = 1/(1+(FPFN/TPTN))
print("Accuracy:",Accuracy_sw_image01_lasso)

Precision_sw_image01_lasso = 1/(1+(FP/TP))
print("Precision:",Precision_sw_image01_lasso)
```

```
Accuracy: 0.5355450236966824
Precision: 0.5
```

sw_image012

In [332]:
```
TP0=cm_sw_image012[0,0]
TP1=cm_sw_image012[1,1]
TP2=cm_sw_image012[2,2]
E10=cm_sw_image012[1,0]
E20=cm_sw_image012[2,0]
E21=cm_sw_image012[2,1]
E01=cm_sw_image012[0,1]
E02=cm_sw_image012[0,2]
E12=cm_sw_image012[1,2]

n=TP0+TP1+TP2+E10+E20+E21+E01+E02+E12
Accuracy_sw_image012_lasso = (TP0+TP1+TP2)/n
print("Accuracy:",Accuracy_sw_image012_lasso)

p0=TP0/(TP0+E10+E20)
p1=TP1/(TP1+E01+E21)
p2=TP2/(TP2+E02+E12)
Precision_sw_image012_lasso =(p0+p1+p2)/3
print("Precision Macro:",Precision_sw_image012_lasso)
```

```
Accuracy: 0.5722222222222222
Precision Macro: 0.6223112929973788
```

Task 3

Use the sklearn.ensemble, keras.models, or keras.layers libraries and implement the random forest or the sequential (simple deep learning) technique as two-class and three-class classifiers using the training sets of the feature vectors that you generated.

In [215]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=0, n_estimators=500, oob_score=True,
n_jobs=-1)
```

image01

In [216]:
```
model_RF_image01 = rf.fit(x_image01_train,y_image01_train)
```

image012

In [221]:
```
model_RF_image012 = rf.fit(x_image012_train,y_image012_train)
```

sw_image01

In [220]:
```
model_RF_sw_image01 = rf.fit(x_sw_image01_train,y_sw_image01_train)
```

sw_image012

In [219]:
```
model_RF_sw_image012 = rf.fit(x_sw_image012_train,y_sw_image012_train)
```

Apply the trained model to the test sets of all the categories of datasets and add the predicted labels next to their actual labels in the corresponding spreadsheets as before.

image01

In [252]:
```
#x_image01_test=image01_test.iloc[:,0:256]
#y_image01_test=image01_test["256"]

yhat_image01_test_rf = model_RF_image01.predict(x_image01_test)
yhat_image01_test_rf = yhat_image01_test_rf.round()

yhat_image01_test_rf=pd.DataFrame(yhat_image01_test_rf)

image01_test = image01_test.reset_index(drop=True)
yhat_image01_test_rf = yhat_image01_test_rf.reset_index(drop=True)

image01_test_predict=pd.concat([image01_test,yhat_image01_test_rf], axis=1)
image01_test_predict.columns.values[-1] = 'rf predict'
image01_test_predict.to_csv('image01_test_predict_rf.csv', index=False)
```

image012

In [253]:
```
#x_image01_test=image01_test.iloc[:,0:256]
#y_image01_test=image01_test["256"]

yhat_image012_test_rf = model_RF_image012.predict(x_image012_test)
yhat_image012_test_rf = yhat_image012_test_rf.round()

yhat_image012_test_rf=pd.DataFrame(yhat_image012_test_rf)

image012_test = image012_test.reset_index(drop=True)
yhat_image012_test_rf = yhat_image012_test_rf.reset_index(drop=True)

image012_test_predict=pd.concat([image012_test,yhat_image012_test_rf], axis=1)
image012_test_predict.columns.values[-1] = 'rf predict'
image012_test_predict.to_csv('image012_test_predict_rf.csv', index=False)
```

sw_image01

In [254]:
```
#x_image01_test=image01_test.iloc[:,0:256]
#y_image01_test=image01_test["256"]

yhat_sw_image01_test_rf = model_RF_sw_image01.predict(x_sw_image01_test)
yhat_sw_image01_test_rf = yhat_sw_image01_test_rf.round()

yhat_sw_image01_test_rf=pd.DataFrame(yhat_sw_image01_test_rf)

sw_image01_test = sw_image01_test.reset_index(drop=True)
yhat_sw_image01_test_rf = yhat_sw_image01_test_rf.reset_index(drop=True)

sw_image01_test_predict=pd.concat([sw_image01_test,yhat_sw_image01_test_rf], axis=1)
sw_image01_test_predict.columns.values[-1] = 'rf predict'
sw_image01_test_predict.to_csv('sw_image01_test_predict_rf.csv', index=False)
```

sw_image012

In [255]:
```
#x_image01_test=image01_test.iloc[:,0:256]
#y_image01_test=image01_test["256"]

yhat_sw_image012_test_rf = model_RF_sw_image012.predict(x_sw_image012_test)
yhat_sw_image012_test_rf = yhat_sw_image012_test_rf.round()

yhat_sw_image012_test_rf=pd.DataFrame(yhat_sw_image012_test_rf)

sw_image012_test = sw_image012_test.reset_index(drop=True)
yhat_sw_image012_test_rf = yhat_sw_image012_test_rf.reset_index(drop=True)

sw_image012_test_predict=pd.concat([sw_image012_test,yhat_sw_image012_test_rf], axis=
sw_image012_test_predict.columns.values[-1] = 'rf predict'
sw_image012_test_predict.to_csv('sw_image012_test_predict_rf.csv', index=False)
```

Construct confusion matrices using the responses in the actual and predicted label columns in the test datasets for all the four categories of datasets. Once again, don’t forget to save these confusion matrices!!!

image01

In [256]:
```
cm_image01_rf = confusion_matrix(y_image01_test, yhat_image01_test_rf, labels=[0,1])
cm_image01_rf
```

Out[256]:
```
array([[31,  2],
       [ 3, 22]], dtype=int64)
```

image012

In [257]:
```
cm_image012_rf = confusion_matrix(y_image012_test, yhat_image012_test_rf, labels=[0,1
cm_image012_rf
```

Out[257]:
```
array([[24,  8,  2],
       [ 5, 17,  0],
       [ 6,  0, 37]], dtype=int64)
```

sw_image01

In [258]:
```
cm_sw_image01_rf = confusion_matrix(y_sw_image01_test, yhat_sw_image01_test_rf, label
cm_sw_image01_rf
```

Out[258]:
```
array([[80, 17],
       [39, 74]], dtype=int64)
```

sw_image012

In [259]:
```
cm_sw_image012_rf = confusion_matrix(y_sw_image012_test, yhat_sw_image012_test_rf, la
cm_sw_image012_rf
```

Out[259]:
```
array([[ 87,  21,   1],
       [ 21,  82,   0],
       [ 10,   2, 142]], dtype=int64)
```

Adapt the same two measures used in Task 2 to quantify the performance of the models (random forest or deep learning) that you selected. Compare the model’s performance for all the categories of datasets.

1. Accuracy
2. Precision

image01

In [310]:
```
TN = cm_image01_rf[1,1]
FP = cm_image01_rf[1,0]
FN = cm_image01_rf[0,1]
TP = cm_image01_rf[0,0]
FPFN = FP+FN
TPTN = TP+TN

Accuracy_image01_rf = 1/(1+(FPFN/TPTN))
print("Accuracy:",Accuracy_image01_rf)

Precision_image01_rf = 1/(1+(FP/TP))
print("Precision:",Precision_image01_rf)
```

```
Accuracy: 0.9137931034482759
Precision: 0.911764705882353
```

image012

In [333]:
```
TP0=cm_image012_rf[0,0]
TP1=cm_image012_rf[1,1]
TP2=cm_image012_rf[2,2]
E10=cm_image012_rf[1,0]
E20=cm_image012_rf[2,0]
E21=cm_image012_rf[2,1]
E01=cm_image012_rf[0,1]
E02=cm_image012_rf[0,2]
E12=cm_image012_rf[1,2]

n=TP0+TP1+TP2+E10+E20+E21+E01+E02+E12
Accuracy_image012_rf = (TP0+TP1+TP2)/n
print("Accuracy:",Accuracy_image012_rf)

p0=TP0/(TP0+E10+E20)
p1=TP1/(TP1+E01+E21)
p2=TP2/(TP2+E02+E12)
Precision_image012_rf =(p0+p1+p2)/3
print("Precision Macro:",Precision_image012_rf)
```

```
Accuracy: 0.7878787878787878
Precision Macro: 0.7714774114774116
```

sw_image01

In [312]:
```
TN = cm_sw_image01_rf[1,1]
FP = cm_sw_image01_rf[1,0]
FN = cm_sw_image01_rf[0,1]
TP = cm_sw_image01_rf[0,0]
FPFN = FP+FN
TPTN = TP+TN

Accuracy_sw_image01_rf = 1/(1+(FPFN/TPTN))
print("Accuracy:",Accuracy_sw_image01_rf)

Precision_sw_image01_rf = 1/(1+(FP/TP))
print("Precision:",Precision_sw_image01_rf)
```

```
Accuracy: 0.7333333333333333
Precision: 0.6722689075630252
```

sw_image012

In [334]:
```
TP0=cm_sw_image012_rf[0,0]
TP1=cm_sw_image012_rf[1,1]
TP2=cm_sw_image012_rf[2,2]
E10=cm_sw_image012_rf[1,0]
E20=cm_sw_image012_rf[2,0]
E21=cm_sw_image012_rf[2,1]
E01=cm_sw_image012_rf[0,1]
E02=cm_sw_image012_rf[0,2]
E12=cm_sw_image012_rf[1,2]

n=TP0+TP1+TP2+E10+E20+E21+E01+E02+E12
Accuracy_sw_image012_rf = (TP0+TP1+TP2)/n
print("Accuracy:",Accuracy_sw_image012_rf)

p0=TP0/(TP0+E10+E20)
p1=TP1/(TP1+E01+E21)
p2=TP2/(TP2+E02+E12)
Precision_sw_image012_rf =(p0+p1+p2)/3
print("Precision Macro:",Precision_sw_image012_rf)
```

```
Accuracy: 0.8497267759562842
Precision Macro: 0.8370825031841981
```

Task 4

Find some built-in measures that are available in software libraries like metrics.accuracy_score in Python's sklearn environment. Use such two measures to compare the performance of the models.

In [281]:
```
from sklearn import metrics
```

Lasso Regression Model

image01

In [359]:
```
BuiltIn_Accuracy_image01_lasso=metrics.accuracy_score(y_image01_test, yhat_image01_te
print("BuiltIn_Accuracy:",BuiltIn_Accuracy_image01_lasso)
BuiltIn_Precision_image01_lasso=metrics.precision_score(y_image01_test, yhat_image01_
print("BuiltIn_Precision:",BuiltIn_Precision_image01_lasso)
```

```
BuiltIn_Accuracy: 0.5862068965517241
BuiltIn_Precision: 0.6354679802955665
```

image012

In [358]:
```
BuiltIn_Accuracy_image012_lasso=metrics.accuracy_score(y_image012_test, yhat_image012
print("BuiltIn_Accuracy:",BuiltIn_Accuracy_image012_lasso)
BuiltIn_Precision_image012_lasso=metrics.precision_score(y_image012_test, yhat_image0
print("BuiltIn_Precision:",BuiltIn_Precision_image012_lasso)
```

```
BuiltIn_Accuracy: 0.4444444444444444
BuiltIn_Precision: 0.5913211057947899
```

sw_image01

In [319]:
```
BuiltIn_Accuracy_sw_image01_lasso=metrics.accuracy_score(y_sw_image01_test, yhat_sw_i
print("BuiltIn_Accuracy:",BuiltIn_Accuracy_sw_image01_lasso)
BuiltIn_Precision_sw_image01_lasso=metrics.precision_score(y_sw_image01_test, yhat_sw
print("BuiltIn_Precision:",BuiltIn_Precision_sw_image01_lasso)
```

```
BuiltIn_Accuracy: 0.5355450236966824
BuiltIn_Precision: 0.5397681948289615
```

sw_image012

In [320]:
```
BuiltIn_Accuracy_sw_image012_lasso=metrics.accuracy_score(y_sw_image012_test, yhat_sw
print("BuiltIn_Accuracy:",BuiltIn_Accuracy_sw_image012_lasso)
BuiltIn_Precision_sw_image012_lasso=metrics.precision_score(y_sw_image012_test, yhat_
print("BuiltIn_Precision:",BuiltIn_Precision_sw_image012_lasso)
```

```
BuiltIn_Accuracy: 0.5628415300546448
BuiltIn_Precision: 0.6708249230666691
```

Random forest

image01

In [321]:
```
BuiltIn_Accuracy_image01_rf=metrics.accuracy_score(y_image01_test, yhat_image01_test_
print("BuiltIn_Accuracy:",BuiltIn_Accuracy_image01_rf)
BuiltIn_Precision_image01_rf=metrics.precision_score(y_image01_test, yhat_image01_tes
print("BuiltIn_Precision:",BuiltIn_Precision_image01_rf)
```

```
BuiltIn_Accuracy: 0.9137931034482759
BuiltIn_Precision: 0.9138776200135226
```

image012

In [322]:
```
BuiltIn_Accuracy_image012_rf=metrics.accuracy_score(y_image012_test, yhat_image012_te
print("BuiltIn_Accuracy:",BuiltIn_Accuracy_image012_rf)
BuiltIn_Precision_image012_rf=metrics.precision_score(y_image012_test, yhat_image012_
print("BuiltIn_Precision:",BuiltIn_Precision_image012_rf)
```

```
BuiltIn_Accuracy: 0.7878787878787878
BuiltIn_Precision: 0.7986783586783587
```

sw_image01

In [325]:
```
BuiltIn_Accuracy_sw_image01_rf=metrics.accuracy_score(y_sw_image01_test, yhat_sw_imag
print("BuiltIn_Accuracy:",BuiltIn_Accuracy_sw_image01_rf)
BuiltIn_Precision_sw_image01_rf=metrics.precision_score(y_sw_image01_test, yhat_sw_im
print("BuiltIn_Precision:",BuiltIn_Precision_sw_image01_rf)
```

```
BuiltIn_Accuracy: 0.7298578199052133
BuiltIn_Precision: 0.7477367906696036
```

sw_image012

In [327]:
```
BuiltIn_Accuracy_sw_image012_rf=metrics.accuracy_score(y_sw_image012_test, yhat_sw_im
print("BuiltIn_Accuracy:",BuiltIn_Accuracy_sw_image012_rf)
BuiltIn_Precision_sw_image012_rf=metrics.precision_score(y_sw_image012_test, yhat_sw_
print("BuiltIn_Precision:",BuiltIn_Precision_sw_image012_rf)
```

```
BuiltIn_Accuracy: 0.8497267759562842
BuiltIn_Precision: 0.8571737129530962
```

Describe the quantitative differences between the built-in measures and the confusion-matrix-based measures that you used in the analysis to compare the models with the four categories of datasets.

Lasso Regression Model

image01

In [329]:
```
difference_Accuracy_image01_lasso=Accuracy_image01_lasso-BuiltIn_Accuracy_image01_las
print(difference_Accuracy_image01_lasso)
difference_Precision_image01_lasso=Precision_image01_lasso-BuiltIn_Precision_image01_
print(difference_Precision_image01_lasso)
```

```
0.0102843315184513 17
0.09180474697716079
```

image012

In [343]:
```
difference_Accuracy_image012_lasso=Accuracy_image012_lasso-BuiltIn_Accuracy_image012_
print(difference_Accuracy_image012_lasso)
difference_Precision_image012_lasso=Precision_image012_lasso-BuiltIn_Precision_image0
print(abs(difference_Precision_image012_lasso))
```

```
0.028673835125448077
0.05930356193514075
```

sw_image01

In [344]:
```
difference_Accuracy_sw_image01_lasso=Accuracy_sw_image01_lasso-BuiltIn_Accuracy_sw_im
print(difference_Accuracy_sw_image01_lasso)
difference_Precision_sw_image01_lasso=Precision_sw_image01_lasso-BuiltIn_Precision_sw
print(abs(difference_Precision_sw_image01_lasso))
```

```
0.0
0.039768194828961545
```

sw_image012

In [345]:
```
difference_Accuracy_sw_image012_lasso=Accuracy_sw_image012_lasso-BuiltIn_Accuracy_sw_
print(difference_Accuracy_sw_image012_lasso)
difference_Precision_sw_image012_lasso=Precision_sw_image012_lasso-BuiltIn_Precision_
print(abs(difference_Precision_sw_image012_lasso))
```

```
0.00938069216757742
0.048513630069290326
```

Random forest

image01

In [346]:
```
difference_Accuracy_image01_rf=Accuracy_image01_lasso-BuiltIn_Accuracy_image01_rf
print(abs(difference_Accuracy_image01_rf))
difference_Precision_image01_rf=Precision_image01_rf-BuiltIn_Precision_image01_rf
print(abs(difference_Precision_image01_rf))
```

```
0.3173018753781005
0.002112914131169541
```

image012

In [348]:
```
difference_Accuracy_image012_rf=Accuracy_image012_rf-BuiltIn_Accuracy_image012_rf
print(difference_Accuracy_image012_rf)
difference_Precision_image012_rf=Precision_image012_rf-BuiltIn_Precision_image012_rf
print(abs(difference_Precision_image012_rf))
```

```
0.0
0.027200947200947123
```

sw_image01

In [349]:
```
difference_Accuracy_sw_image01_rf=Accuracy_sw_image01_rf-BuiltIn_Accuracy_sw_image01_
print(difference_Accuracy_sw_image01_rf)
difference_Precision_sw_image01_rf=Precision_sw_image01_rf-BuiltIn_Precision_sw_image
print(abs(difference_Precision_sw_image01_rf))
```

```
0.003475513428120025
0.07546788310657848
```

sw_image012

In [350]:
```
difference_Accuracy_sw_image012_rf=Accuracy_sw_image012_rf-BuiltIn_Accuracy_sw_image0
print(difference_Accuracy_sw_image012_rf)
difference_Precision_sw_image012_rf=Precision_sw_image012_rf-BuiltIn_Precision_sw_ima
print(abs(difference_Precision_sw_image012_rf))
```

```
0.0
0.02009120976889811
```

Compare all the results (qualitative measures) and determine which pair (a model and a category of feature vectors) is superior among the ones that you considered. Use your data and findings to support this result.

Lasso Regression Model:-

- (Using built-in measures)

Feature vectors	Accuracy	Precision
image01	0.586	0.635
image012	0.444	0.591
sw_image01	0.535	0.539
sw_image012	0.562	0.670

Random forest:-

- (Using built-in measures)

Feature vectors	Accuracy	Precision
image01	0.913	0.913

Feature vectors	Accuracy	Precision
image012	0.787	0.798
sw_image01	0.729	0.747

- **Lasso Regression Model**: This model showing lower accuracy and precision scores across all feature vector categories. These scores suggest that the Lasso model might be struggling to fit the data as effectively as the Random Forest model.
- **Random Forest Model**: Random Forest consistently outperforms Lasso in both accuracy and precision across all feature vector categories. The highest scores for both accuracy (0.913) and precision (0.913) occur with the image01 feature vector, indicating strong predictive power and reliable performance when using this feature set.
- **Conclusion**: The **Random Forest model** using the **image01 feature vector** is the superior pairing among those evaluated. This feature set likely contains highly predictive information that Random Forest can leverage better than Lasso, which struggles with lower scores on this and other feature sets.

Classification Report:-

In [367]:
```
from sklearn.metrics import classification_report
```

Lasso Regression Model:-

image01

In [368]:
```
print(classification_report(y_image01_test, yhat_image01_test_lasso,labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.73      0.48      0.58        33
           1       0.51      0.72      0.60        25

   micro avg       0.60      0.59      0.59        58
   macro avg       0.62      0.60      0.59        58
weighted avg       0.64      0.59      0.59        58
```

image012

In [369]:
```
print(classification_report(y_image012_test, yhat_image012_test_lasso,labels=[0,1,2])
```

```
              precision    recall  f1-score   support

           0       0.42      0.24      0.30        34
           1       0.30      0.68      0.42        22
           2       0.88      0.49      0.63        43

   micro avg       0.47      0.44      0.46        99
   macro avg       0.53      0.47      0.45        99
weighted avg       0.59      0.44      0.47        99
```

sw_image01

In [370]:
```
print(classification_report(y_sw_image01_test, yhat_sw_image01_test_lasso,labels=[0,1
```

```
              precision    recall  f1-score   support

           0       0.50      0.56      0.53        98
           1       0.57      0.51      0.54       113

    accuracy                           0.54       211
   macro avg       0.54      0.54      0.54       211
weighted avg       0.54      0.54      0.54       211
```

sw_image012

In [371]:
```
print(classification_report(y_sw_image012_test, yhat_sw_image012_test_lasso,labels=[0
```

```
              precision    recall  f1-score   support

           0       0.46      0.28      0.35       109
           1       0.41      0.81      0.55       103
           2       0.99      0.60      0.74       154

   micro avg       0.57      0.56      0.57       366
   macro avg       0.62      0.56      0.55       366
weighted avg       0.67      0.56      0.57       366
```

Random forest:-

image01

In [372]:
```
print(classification_report(y_image01_test, yhat_image01_test_rf,labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.91      0.94      0.93        33
           1       0.92      0.88      0.90        25

    accuracy                           0.91        58
   macro avg       0.91      0.91      0.91        58
weighted avg       0.91      0.91      0.91        58
```

image012

In [373]:
```
print(classification_report(y_image012_test, yhat_image012_test_rf,labels=[0,1,2]))
```

```
              precision    recall  f1-score   support

           0       0.69      0.71      0.70        34
           1       0.68      0.77      0.72        22
           2       0.95      0.86      0.90        43

    accuracy                           0.79        99
   macro avg       0.77      0.78      0.77        99
weighted avg       0.80      0.79      0.79        99
```

sw_image01

In [374]:
```
print(classification_report(y_sw_image01_test, yhat_sw_image01_test_rf,labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.67      0.82      0.74        98
           1       0.81      0.65      0.73       113

   micro avg       0.73      0.73      0.73       211
   macro avg       0.74      0.74      0.73       211
weighted avg       0.75      0.73      0.73       211
```

sw_image012

In [375]:
```
print(classification_report(y_sw_image012_test, yhat_sw_image012_test_rf,labels=[0,1,
```

```
              precision    recall  f1-score   support

           0       0.74      0.80      0.77       109
           1       0.78      0.80      0.79       103
           2       0.99      0.92      0.96       154

    accuracy                           0.85       366
   macro avg       0.84      0.84      0.84       366
weighted avg       0.86      0.85      0.85       366
```