In [1]:

```
# Data analysis of TCGA-BRCA dataset on Breast Invasive Carcinoma by Zahra

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV, cross_v
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassi
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.linear_model import LassoCV
from sklearn.metrics import accuracy_score, classification_report, confusic
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
# dataset from
# Kaggle: https://www.kaggle.com/datasets/0425b3af5246404d92316a6887a58e404
# original dataset processed by https://rbabaei82.github.io/MultiOmics_TCGA
# From NCI https://portal.gdc.cancer.gov/projects/TCGA-BRCA
```

In [3]:

```
# Step 1: Data Loading and Preprocessing
# A: Load the dataset
df = pd.read_csv('brca_data_w_subtypes.csv')
df.head(5)
```

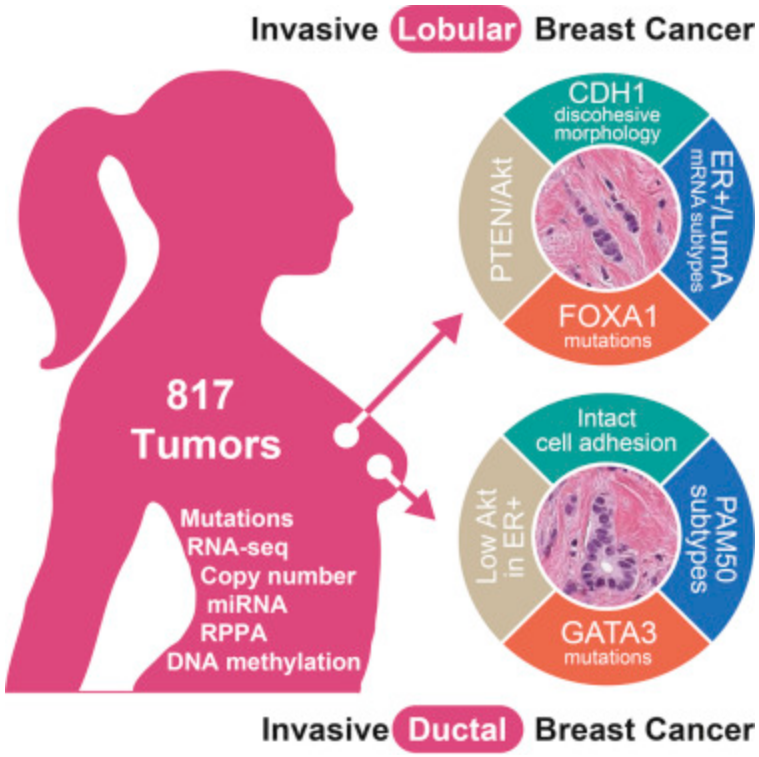
Out[3]:

	rs_CLEC3A	rs_CPB1	rs_SCGB2A2	rs_SCGB1D2	rs_TFF1	rs_MUCL1	rs_GSTM1	
0	0.892818	6.580103	14.123672	10.606501	13.189237	6.649466	10.520335	10.
1	0.000000	3.691311	17.116090	15.517231	9.867616	9.691667	8.179522	7.
2	3.748150	4.375255	9.658123	5.326983	12.109539	11.644307	10.517330	5.
3	0.000000	18.235519	18.535480	14.533584	14.078992	8.913760	10.557465	13.
4	0.000000	4.583724	15.711865	12.804521	8.881669	8.430028	12.964607	6.

5 rows × 1941 columns

In [4]:
```
# Content
# There are 705 breast cancer samples. The dataset contains four different
# From https://www.cell.com/cell/fulltext/S0092-8674(15)01195-2
# Comprehensive Molecular Portraits of Invasive Lobular Breast Cancer by Gi
# cn: copy number variations (n=860)
# mu: mutations (n=249)
# rs: gene expression (n=604)
# pp: protein levels (n=223)
# Acknowledgements
# The dataset was lifted from this analysis by rbabaei https://rbabaei82.gi
from IPython.display import Image
Image(filename='BRCA_IntroImage.jpg')
```

Out[4]:



In [5]:
```
# B: Checking the dataset
df.dtypes
```

Out[5]:
```
rs_CLEC3A            float64
rs_CPB1              float64
rs_SCGB2A2           float64
rs_SCGB1D2           float64
rs_TFF1              float64
                      ...
vital.status          int64
PR.Status            object
ER.Status            object
HER2.Final.Status    object
histological.type    object
Length: 1941, dtype: object
```

In [6]:
```
df.shape
```

Out[6]:
```
(705, 1941)
```

In [7]:
```
with pd.option_context('display.max_rows', None, 'display.max_columns', Nor
    print(df.isnull().sum(axis = 0))
```

```
rs_CLEC3A                       0
rs_CPB1                         0
rs_SCGB2A2                      0
rs_SCGB1D2                      0
rs_TFF1                         0
rs_MUCL1                        0
rs_GSTM1                        0
rs_PIP                          0
rs_ADIPOQ                       0
rs_ADH1B                        0
rs_S100A7                       0
rs_HMGCS2                       0
rs_CYP2B7P1                     0
rs_ANKRD30A                     0
rs_PRAME                        0
rs_TAT                          0
rs_SERPINA6                     0
rs_AGR3                         0
rs_TFAP2B                       0
```

In [9]:
```
# C: Drop row with missing  the dataset
new_df=df.dropna(how='any')
new_df.shape
```

Out[9]:
```
(560, 1941)
```

In [10]:
```
new_df.dtypes
```

Out[10]:
```
rs_CLEC3A            float64
rs_CPB1              float64
rs_SCGB2A2           float64
rs_SCGB1D2           float64
rs_TFF1              float64
                      ...
vital.status          int64
PR.Status            object
ER.Status            object
HER2.Final.Status    object
histological.type    object
Length: 1941, dtype: object
```

In [11]:
```
# Count values in column 'histological.type'
histo_count = new_df['histological.type'].value_counts()

print(histo_count)
```

```
infiltrating ductal carcinoma     509
infiltrating lobular carcinoma     51
Name: histological.type, dtype: int64
```

In [12]:
```
# Count values in column for human epidermal growth factor receptor 2 'HER2
HER2_count = new_df['HER2.Final.Status'].value_counts()

print(HER2_count)
```

```
Negative         457
Positive          86
Equivocal          9
Not Available      8
Name: HER2.Final.Status, dtype: int64
```

In [13]:
```
# Count values in column 'vital.status'
vita_count = new_df['vital.status'].value_counts()

print(vita_count)
```

```
0    487
1     73
Name: vital.status, dtype: int64
```

In [14]:
```
# Count values in column Estrogen receptor'ER.status'
erS_count = new_df['ER.Status'].value_counts()

print(erS_count)
```

```
Positive                       401
Negative                       128
Not Performed                   27
Performed but Not Available      2
Indeterminate                    2
Name: ER.Status, dtype: int64
```

In [15]:
```
# Count values in column Progesteron receptor'PR.status'
prS_count = new_df['PR.Status'].value_counts()

print(prS_count)
```

```
Positive                       342
Negative                       184
Not Performed                   28
Indeterminate                    4
Performed but Not Available      2
Name: PR.Status, dtype: int64
```

In [16]:

```
# Step 2: Principal Component Analysis (PCA)
# We will be using K-Means Clustering on PCA-Transformed dataset new_df
# Features will be X =new_df['vital.status','PR.Status','ER.Status','HER2.F
```

In [17]:

```
# Convert object columns into int using pd.get_dummies instead of one-hot e
# in the features
import pandas as pd
# within features in new_df['PR.Status','ER.Status','HER2.Final.Status','hi

categorical_columns = new_df.select_dtypes(include=['object']).columns.toli
new_df2=pd.get_dummies(new_df, columns=categorical_columns)
new_df2
```

Out[17]:

	rs_CLEC3A	rs_CPB1	rs_SCGB2A2	rs_SCGB1D2	rs_TFF1	rs_MUCL1	rs_GSTM1	
0	0.892818	6.580103	14.123672	10.606501	13.189237	6.649466	10.520335	1
1	0.000000	3.691311	17.116090	15.517231	9.867616	9.691667	8.179522	
2	3.748150	4.375255	9.658123	5.326983	12.109539	11.644307	10.517330	
3	0.000000	18.235519	18.535480	14.533584	14.078992	8.913760	10.557465	1
4	0.000000	4.583724	15.711865	12.804521	8.881669	8.430028	12.964607	
...	...	...	...	...	...	...	...	
644	0.000000	14.652475	6.430018	2.487152	10.896235	2.487152	1.507262	
645	0.000000	4.071531	3.128508	5.467426	0.000000	0.718438	7.640049	
647	3.186199	11.624534	8.096817	4.858956	9.683010	9.276432	0.000000	
648	15.582967	10.151592	15.638541	14.126165	12.447517	16.541921	2.528146	1
649	0.000000	5.093780	15.999704	13.844006	11.539494	13.139496	1.073820	1

560 rows × 1953 columns

In [18]:
```
with pd.option_context('display.max_rows', None, 'display.max_columns', Nor
    print(new_df2.dtypes)
```

```
rs_CLEC3A                                          float64
rs_CPB1                                            float64
rs_SCGB2A2                                         float64
rs_SCGB1D2                                         float64
rs_TFF1                                            float64
rs_MUCL1                                           float64
rs_GSTM1                                           float64
rs_PIP                                             float64
rs_ADIPOQ                                          float64
rs_ADH1B                                           float64
rs_S100A7                                          float64
rs_HMGCS2                                          float64
rs_CYP2B7P1                                        float64
rs_ANKRD30A                                        float64
rs_PRAME                                           float64
rs_TAT                                             float64
rs_SERPINA6                                        float64
rs_AGR3                                            float64
rs_TFAP2B                                          float64
```

In [19]:
```
with pd.option_context('display.max_rows', None, 'display.max_columns', Nor
    print(new_df2.isnull().sum(axis = 0))
```

```
rs_CLEC3A                                          0
rs_CPB1                                            0
rs_SCGB2A2                                         0
rs_SCGB1D2                                         0
rs_TFF1                                            0
rs_MUCL1                                           0
rs_GSTM1                                           0
rs_PIP                                             0
rs_ADIPOQ                                          0
rs_ADH1B                                           0
rs_S100A7                                          0
rs_HMGCS2                                          0
rs_CYP2B7P1                                        0
rs_ANKRD30A                                        0
rs_PRAME                                           0
rs_TAT                                             0
rs_SERPINA6                                        0
rs_AGR3                                            0
rs_TFAP2B                                          0
```

In [20]:
```
features = ['vital.status','PR.Status_Indeterminate','PR.Status_Negative','

X = new_df2[features]

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [21]:
```
# Set a PCA object and PCA transform the scaled data
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
```

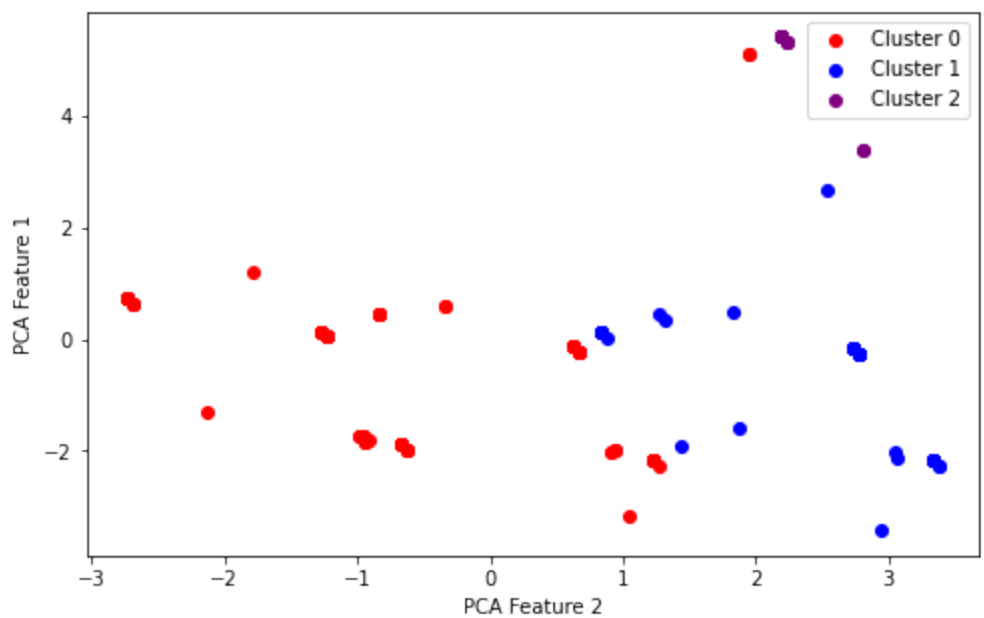
In [22]:
```
# Step 3: K-means clustering: cluster amount and random state for reproduci
# Invasive lobular carcinoma (ILC) is the second most prevalent histologic
# To profile the breast tumors, after dropping NAN values, we have includir
# and 88 mixed IDC/ILC.

count_ILC = new_df2['histological.type_infiltrating lobular carcinoma'].val
print("Number of 'ILC cases':", count_ILC)
count_IDC = new_df2['histological.type_infiltrating ductal carcinoma'].valu
print("Number of 'IDC cases':", count_IDC)
count_IDC_ILC = (new_df2['histological.type_infiltrating ductal carcinoma']
#(df['col2'] == True) & (df['col3'] == True)
print("Number of 'IDC_ILC case':", count_IDC_ILC)
```

```
Number of 'ILC cases': 51
Number of 'IDC cases': 509
Number of 'IDC_ILC case': 49
```

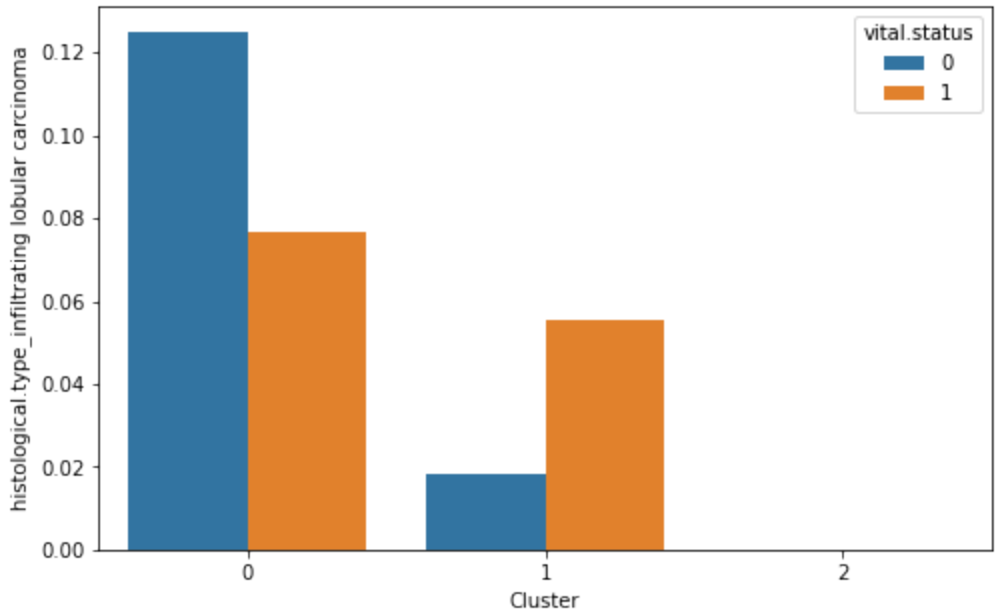
In [23]:

```
kmeans = KMeans(n_clusters=3, random_state=24)

clusters = kmeans.fit_predict(X_pca) # Cluster assignment
new_df2['Cluster'] = clusters

# Define cluster colors
cluster_colors = {0: 'red', 1: 'blue', 2: 'purple'}
plt.figure(figsize=(8, 5))

# Create separate scatter plots for each cluster
for cluster_id, color in cluster_colors.items():
     cluster_mask = np.where(clusters == cluster_id)
     plt.scatter(X_pca[cluster_mask, 0], X_pca[cluster_mask, 1], c=color, l

# Add a legend
plt.legend()
plt.ylabel('PCA Feature 1')
plt.xlabel('PCA Feature 2')
plt.show()

# Adding PCA dimensions to the data for visualization
new_df2['PCA_1'] = X_pca[:, 0]
new_df2['PCA_2'] = X_pca[:, 1]
```

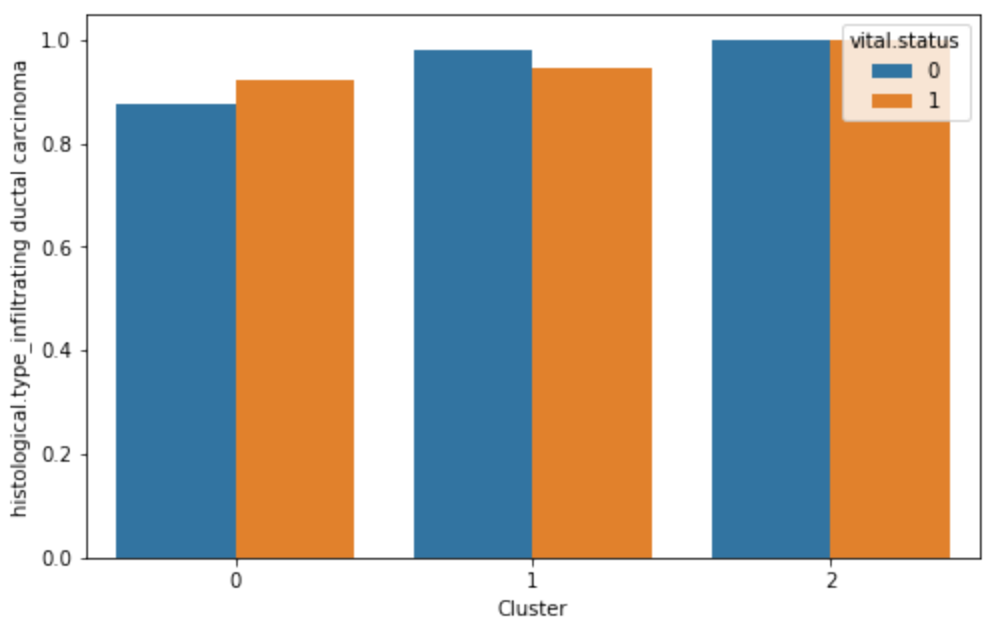
```
C:\Users\Zahra\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:133
4: UserWarning: KMeans is known to have a memory leak on Windows with MK
L, when there are less chunks than available threads. You can avoid it by
setting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
```



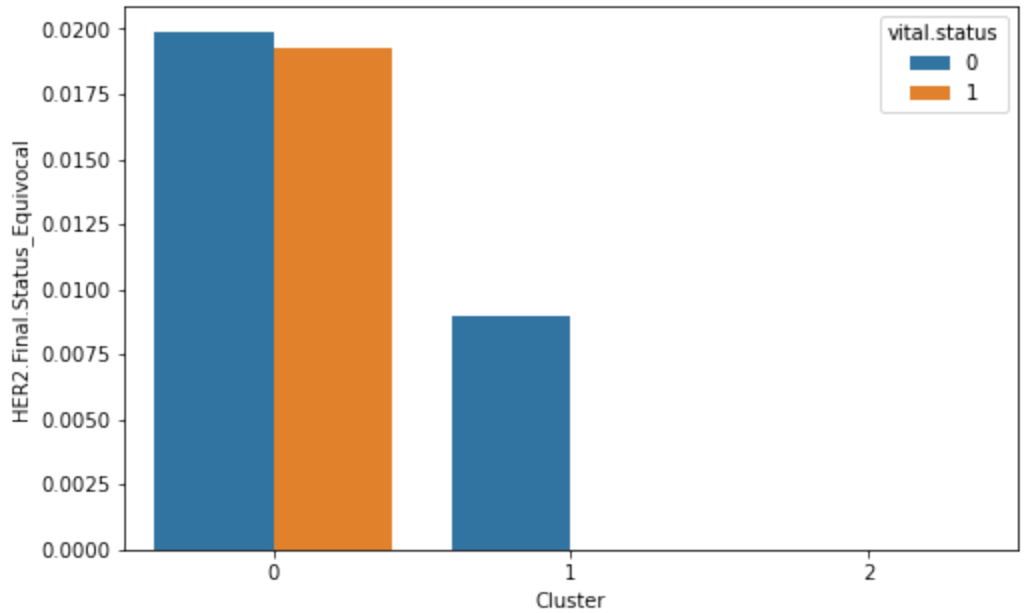
In [24]:
```
# Step 4: Visulization
# Boxplot of ILC with vital status outcome for each cluster
plt.figure(figsize=(8, 5))
sns.barplot(x='Cluster', y='histological.type_infiltrating lobular carcinom
plt.show()
```



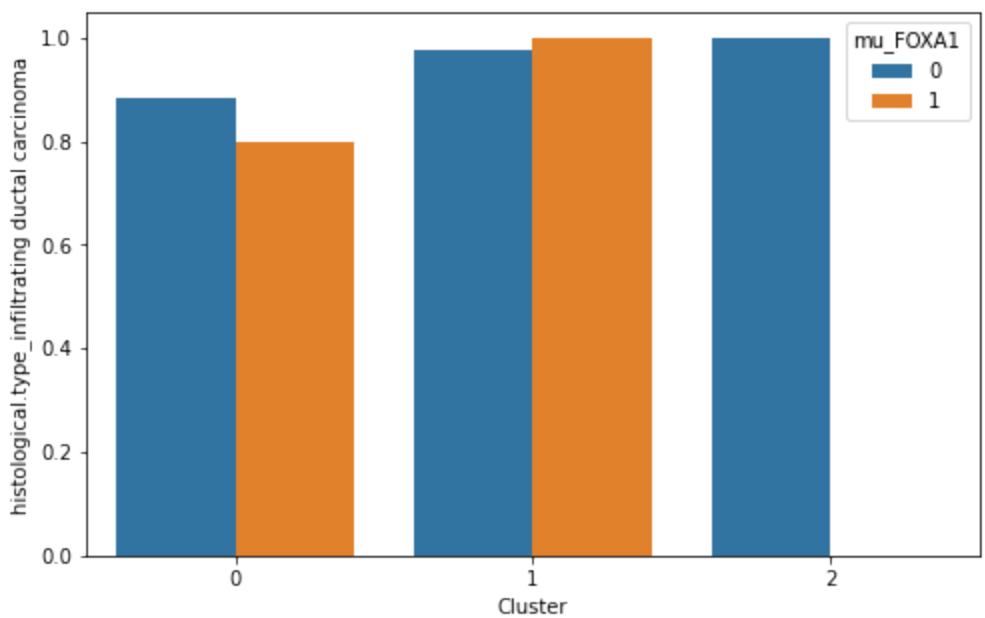
In [25]:
```
# Boxplot of IDC with vital status outcome for each cluster
plt.figure(figsize=(8, 5))
sns.barplot(x='Cluster', y='histological.type_infiltrating ductal carcinoma
plt.show()
```



In [26]:
```
# The boxplot of vital status outcomes of ILC vs IDC shows that there is lo
plt.figure(figsize=(8, 5))
sns.barplot(x='Cluster', y='HER2.Final.Status_Equivocal', data=new_df2, hue
plt.show()
```

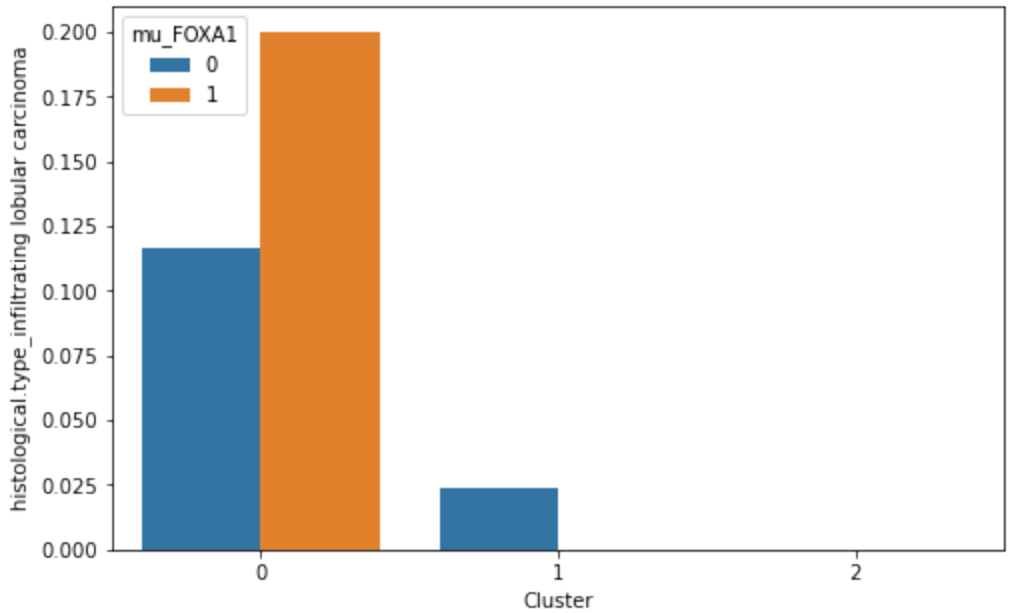


In [27]:
```
# Boxplot of IDC with FOXA1 mutations  outcome for each cluster
plt.figure(figsize=(8, 5))
sns.barplot(x='Cluster', y='histological.type_infiltrating ductal carcinoma
plt.show()
```



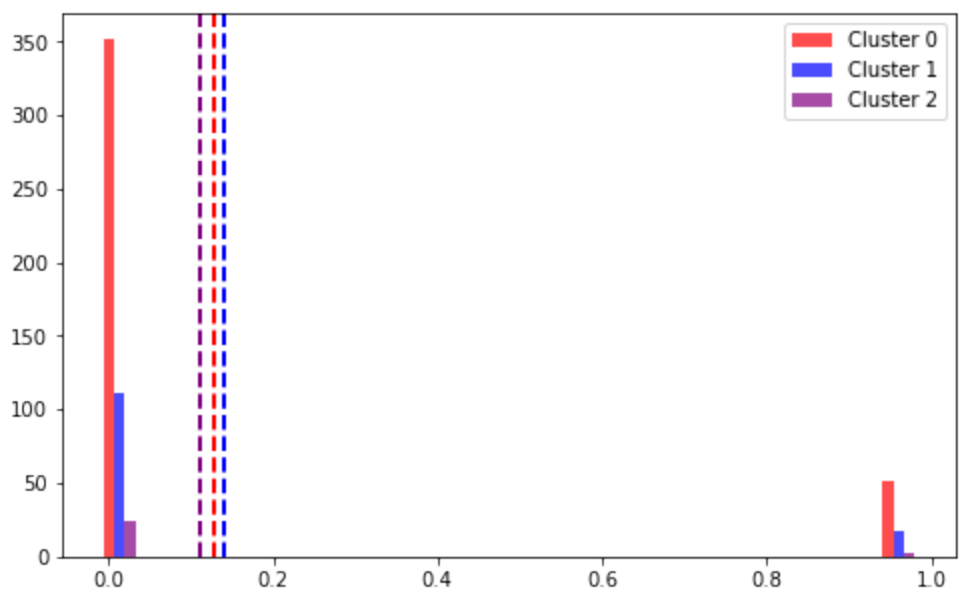
In [28]:

```
# Boxplot of ILC with FOXA1 mutations  outcome for each cluster
plt.figure(figsize=(8, 5))
sns.barplot(x='Cluster', y='histological.type_infiltrating lobular carcinom
plt.show()
```



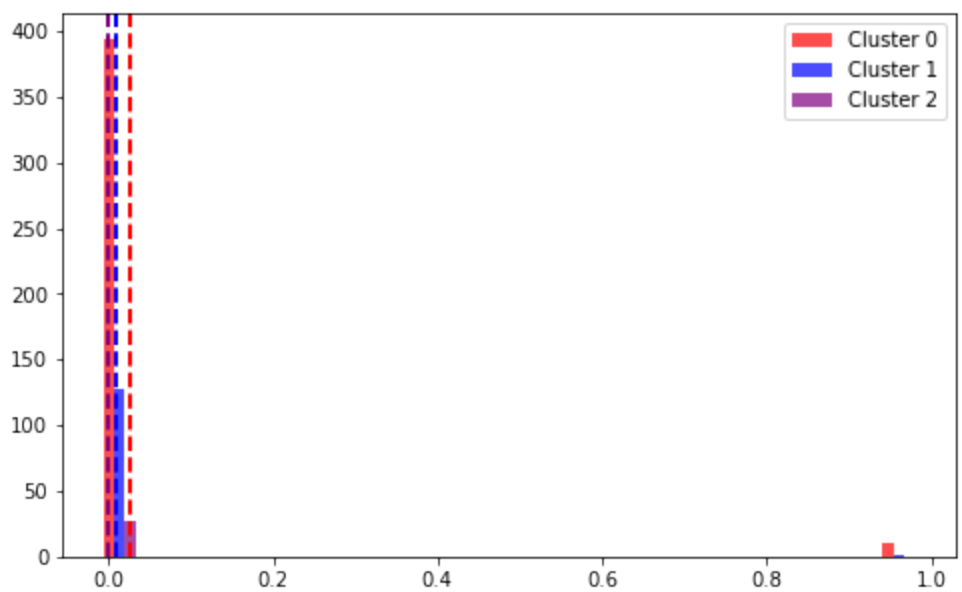
In [35]:

```
#Histograms for target variables
cluster_colors = {0: 'red', 1: 'blue', 2: 'purple'}
plt.figure(figsize=(8, 5))
clusters = [0, 1, 2]
feature = 'vital.status'  # The feature to plot

# Create bins with a common range for all histograms
min_value = new_df2[feature].min()
max_value = new_df2[feature].max()
bins = np.linspace(min_value, max_value, 20)

# Set up the plot
width = np.diff(bins)[0] / (len(clusters) + 1)  # Width of each bar

# Plot histograms for each cluster
for i, cluster in enumerate(clusters):
    cluster_data = new_df2[new_df2['Cluster'] == cluster][feature]
    cluster_mean = cluster_data.mean()  # Calculate mean for the cluster
    bar_positions = bins[:-1] + (i * width)  # Position bars side by side
    plt.bar(bar_positions, np.histogram(cluster_data, bins=bins)[0],
            width=width, color=cluster_colors[i], alpha=0.7,
            label=f'Cluster {cluster}')
    plt.axvline(cluster_mean, color=cluster_colors[i], linestyle='--', line

# Finalize and show the plot
plt.legend()
plt.show()
```



In [30]:

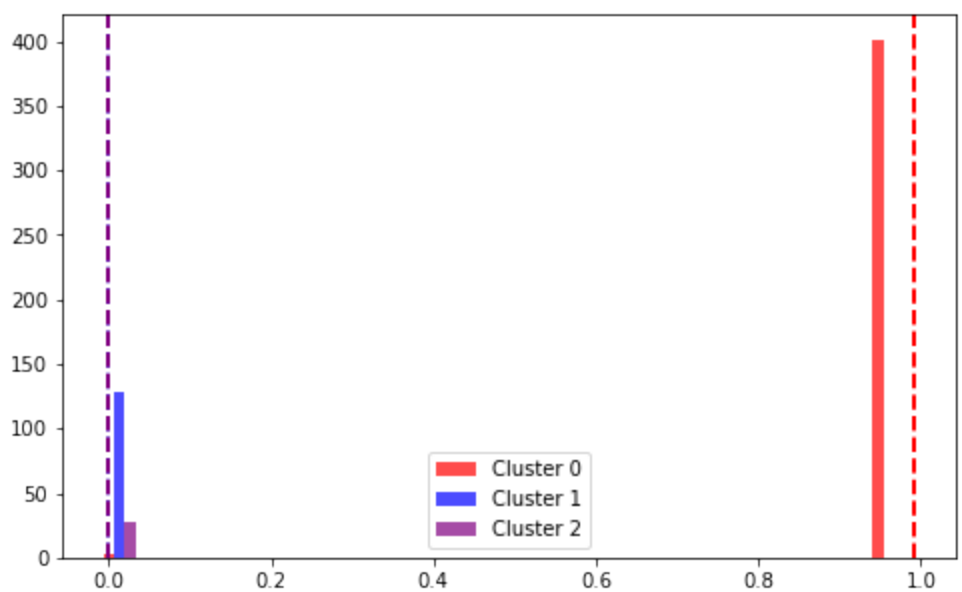
```
#Histograms for target variables
cluster_colors = {0: 'red', 1: 'blue', 2: 'purple'}
plt.figure(figsize=(8, 5))
clusters = [0, 1, 2]
feature = 'mu_FOXA1'  # The feature to plot

# Create bins with a common range for all histograms
min_value = new_df2[feature].min()
max_value = new_df2[feature].max()
bins = np.linspace(min_value, max_value, 20)

# Set up the plot
width = np.diff(bins)[0] / (len(clusters) + 1)  # Width of each bar

# Plot histograms for each cluster
for i, cluster in enumerate(clusters):
    cluster_data = new_df2[new_df2['Cluster'] == cluster][feature]
    cluster_mean = cluster_data.mean()  # Calculate mean for the cluster
    bar_positions = bins[:-1] + (i * width)  # Position bars side by side
    plt.bar(bar_positions, np.histogram(cluster_data, bins=bins)[0],
            width=width, color=cluster_colors[i], alpha=0.7,
            label=f'Cluster {cluster}')
    plt.axvline(cluster_mean, color=cluster_colors[i], linestyle='--', line

# Finalize and show the plot
plt.legend()
plt.show()
```



In [31]:

```
#Histograms for target variables
cluster_colors = {0: 'red', 1: 'blue', 2: 'purple'}
plt.figure(figsize=(8, 5))
clusters = [0, 1, 2]
feature = 'ER.Status_Positive'  # The feature to plot

# Create bins with a common range for all histograms
min_value = new_df2[feature].min()
max_value = new_df2[feature].max()
bins = np.linspace(min_value, max_value, 20)

# Set up the plot
plt.figure(figsize=(8, 5))
width = np.diff(bins)[0] / (len(clusters) + 1)  # Width of each bar

# Plot histograms for each cluster
for i, cluster in enumerate(clusters):
    cluster_data = new_df2[new_df2['Cluster'] == cluster][feature]
    cluster_mean = cluster_data.mean()  # Calculate mean for the cluster
    bar_positions = bins[:-1] + (i * width)  # Position bars side by side
    plt.bar(bar_positions, np.histogram(cluster_data, bins=bins)[0],
            width=width, color=cluster_colors[i], alpha=0.7,
            label=f'Cluster {cluster}')
    plt.axvline(cluster_mean, color=cluster_colors[i], linestyle='--', line

# Finalize and show the plot
plt.legend()
plt.show()
```

```
<Figure size 576x360 with 0 Axes>
```



In [32]:

```
# Cluster 2 has the lowest vital status, ER.status_Positive, and a lower mu
# which has the highest ER.status_Positive.
# While cluster 1 has the highest vital status, no ER.status (negative or u
# compared to cluster 2.
# Studies have shown that FOXA1 governs the estrogen-regulated transcripton
# It is unsurprising that Cluster 2 has the lower, ER.status_Positive, and
# FOXA1 is required for estrogen expression.
# Hurtado A.et al FOXA1 is a key determinant of estrogen receptor function
# Robinson et al. FoxA1 is a Key Mediator of Hormonal Response in Breast ar
```

In [33]:

```
# Step 5: Data Loading and Preprocessing

# Separate features and target variable
X =new_df2.drop('vital.status', axis=1)
y =new_df2['vital.status']

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=

# Function to evaluate model performance
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy: {accuracy}')
    print('Classification Report:')
    print(classification_report(y_test, y_pred))
    print('Confusion Matrix:')
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
    return accuracy

# Step 2: Model Training and Evaluation

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
print("Random Forest:")
rf_accuracy = evaluate_model(rf, X_test, y_test)


# Random Forest Accuracy Plot

# Plotting Random Forest accuracy as a bar with value on top
plt.figure(figsize=(6, 6))
rf_accuracy_value = rf_accuracy

# Creating the bar plot
plt.bar(['Random Forest'], [rf_accuracy_value])

# Annotating the bar with accuracy value
plt.text(0, rf_accuracy_value + 0.01, f'{rf_accuracy_value:.4f}', ha='cente

# Adding labels and title
plt.ylim(0, 1)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Random Forest Accuracy', fontsize=14)
plt.show()


# Neural Network
```

```
nn = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500, random_state=42
nn.fit(X_train, y_train)
print("Neural Network:")
nn_accuracy = evaluate_model(nn, X_test, y_test)


# Neural Network Accuracy Bar Plot

# Plotting the accuracy of the Neural Network model
plt.figure(figsize=(6, 6))
sns.barplot(x=['Neural Network'], y=[nn_accuracy])

# Annotating the bar with accuracy value
plt.text(0, nn_accuracy + 0.01, f'{nn_accuracy:.4f}', ha='center', va='bott

# Set plot labels and title
plt.ylim(0, 1)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Neural Network Accuracy', fontsize=14)

# Display the plot
plt.show()


# Support Vector Machine (SVM)
svm = SVC(kernel='linear', probability=True)
svm.fit(X_train, y_train)
print("Support Vector Machine (SVM):")
svm_accuracy = evaluate_model(svm, X_test, y_test)


# SVM Accuracy Bar Plot
plt.figure(figsize=(6, 6))
sns.barplot(x=["SVM"], y=[svm_accuracy], palette="viridis")

# Annotating the bar with accuracy value
plt.text(0, svm_accuracy + 0.01, f'{svm_accuracy:.4f}', ha='center', fontsi

# Customizing the plot
plt.ylim(0, 1)
plt.title('SVM Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Model')
plt.show()


# Gradient Boosting
gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb.fit(X_train, y_train)
print("Gradient Boosting:")
gb_accuracy = evaluate_model(gb, X_test, y_test)

# Gradient Boosting
gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb.fit(X_train, y_train)
print("Gradient Boosting:")
gb_accuracy = evaluate_model(gb, X_test, y_test)
```

```
# Plotting the Gradient Boosting accuracy
plt.figure(figsize=(6, 6))
sns.barplot(x=['Gradient Boosting'], y=[gb_accuracy], palette='viridis')

# Annotating the bar with the accuracy value
plt.text(0, gb_accuracy + 0.01, f'{gb_accuracy:.4f}', ha='center', fontsize

plt.ylim(0, 1)  # Set y-axis limits to make sure the bar and text fit well
plt.title('Gradient Boosting Model Accuracy', fontsize=16)
plt.ylabel('Accuracy', fontsize=14)
plt.xlabel('Model', fontsize=14)
plt.show()

# Step 3: Results and Visualization
# Plotting the results
models = ['Random Forest', 'Neural Network', 'SVM', 'Gradient Boosting']
accuracies = [rf_accuracy, nn_accuracy, svm_accuracy, gb_accuracy]


plt.figure(figsize=(10, 6))
sns.barplot(x=models, y=accuracies)

# Annotating the bars with accuracy values
for index, value in enumerate(accuracies):
    plt.text(index, value + 0.01, f'{value:.4f}', ha='center')

plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy')
plt.xlabel('Model')
plt.show()

# Identifying the Best Performing Model
best_model_index = np.argmax(accuracies)
best_model_name = models[best_model_index]
best_model_accuracy = accuracies[best_model_index]

print(f'The best performing model is {best_model_name} with an accuracy of

# Additional metrics and visualizations
def plot_roc_curve(model, X_test, y_test):
    y_pred_prob = model.predict_proba(X_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    auc = roc_auc_score(y_test, y_pred_prob)
    plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend()
    plt.show()

print("Random Forest ROC Curve:")
plot_roc_curve(rf, X_test, y_test)

print("Neural Network ROC Curve:")
plot_roc_curve(nn, X_test, y_test)
```

```
print("Support Vector Machine (SVM) ROC Curve:")
plot_roc_curve(svm, X_test, y_test)

print("Gradient Boosting ROC Curve:")
plot_roc_curve(gb, X_test, y_test)
```

```
Random Forest:
Accuracy: 0.9017857142857143
Classification Report:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        98
           1       1.00      0.21      0.35        14

    accuracy                           0.90       112
   macro avg       0.95      0.61      0.65       112
weighted avg       0.91      0.90      0.87       112

Confusion Matrix:
```

In [34]:

```
#The best performing model is Random Forest with an accuracy of 0.901785714
```

In [ ]:

In [ ]: