Classification of cells from Masaeli et al.

Please indicate at the top of your assignment whether or not you used any AI tools, such as MS Copilot. If you did use one of these tools, please provide a very brief explanation alongside each answer for how you confirmed the correctness of your solution.

- Used MS Copilot for some of the questions

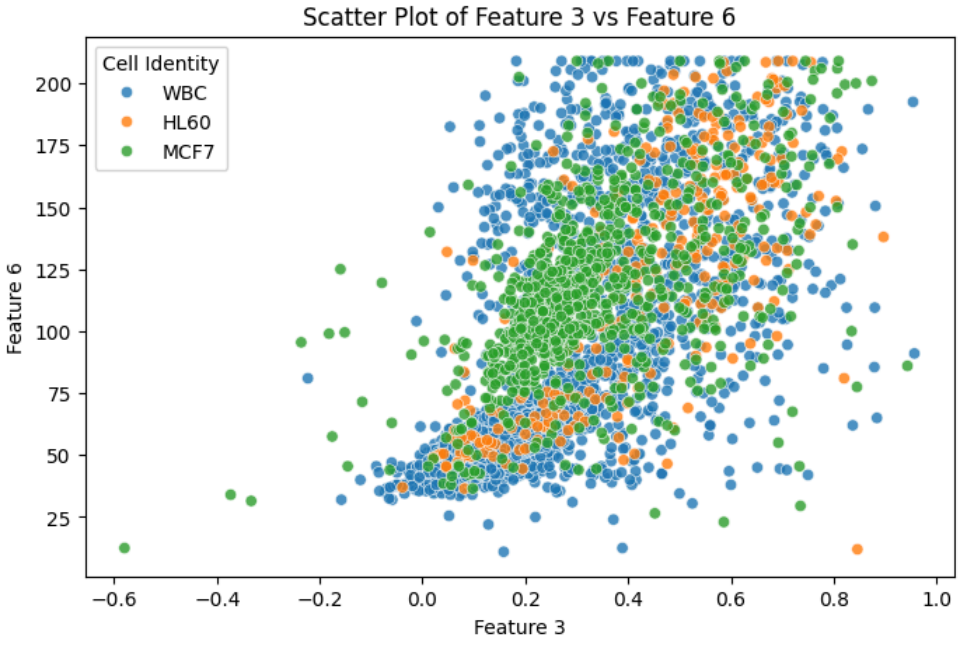
We're going to reimplement an SVM model from a Di Carlo lab's study of the mechanical properties of cells. With this, we'll then explore some of its properties.

In [5]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn.preprocessing import label_binarize
from sklearn.decomposition import PCA
import sklearn.metrics as metrics
from sklearn.svm import SVC, LinearSVC
from scipy import stats
from sklearn.cross_decomposition import PLSRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

mat = pd.read_csv('WBC.csv')
mat.drop('Unnamed: 0', axis=1, inplace=True)
X = mat.iloc[:, :-1]
y = mat.iloc[:, -1]


print(set(y))
print(X.shape, y.shape)
```

```
{'WBC', 'HL60', 'MCF7'}
(10108, 19) (10108,)
```

(1) Choose any two variables and plot them against cell identity (in color). Do you see clear separation of these classes? What does this tell you about whether or not you can classify the cells of differing type?

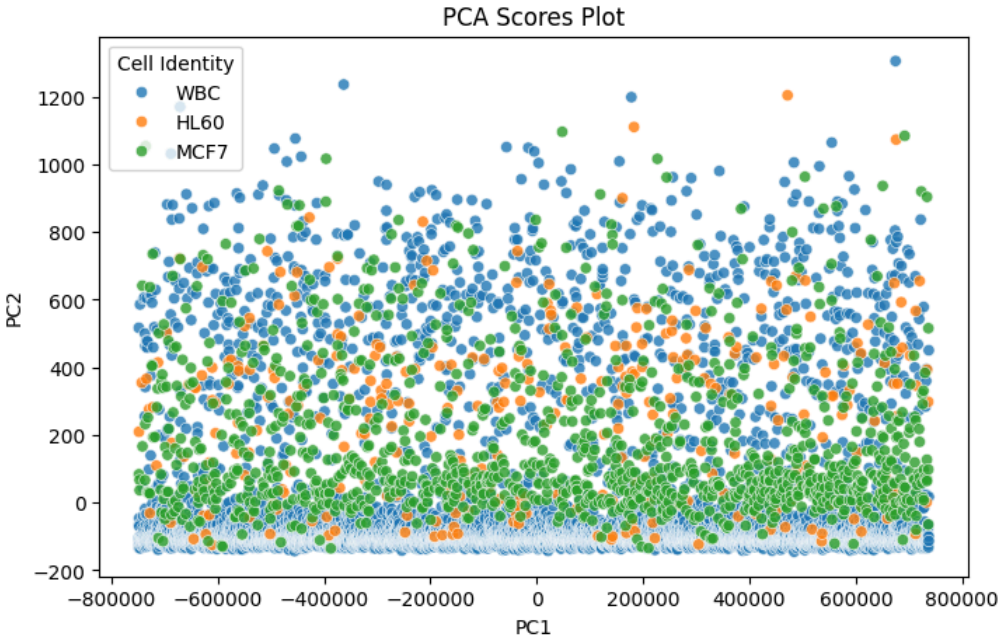
In [2]:
```
# Choosing two variables (columns)
feature3 = X.columns[2]
feature6 = X.columns[5]

# Scatter plot
plt.figure(figsize=(8, 5))
sns.scatterplot(x=X[feature3], y=X[feature6], hue=y, alpha=0.8)

# Add labels and title
plt.title(f"Scatter Plot of Feature 3 vs Feature 6")
plt.xlabel("Feature 3")
plt.ylabel("Feature 6")
plt.legend(title="Cell Identity")
plt.show()
```



(2) Use principal components analysis to visualize the variation in each variable. Plot the first two principal components against the cell type. Do you see better separation in this case? What does this tell you about your ability to distinguish the cell types?

In [3]:
```
# Perform PCA
pca = PCA(n_components=2, random_state=0)
scores = pca.fit_transform(X)
loadings = pca.components_.T

# Plot Scores
plt.figure(figsize=(8, 5))
sns.scatterplot(x=scores[:, 0], y=scores[:, 1], hue=y, alpha=0.8)
plt.title("PCA Scores Plot")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(title="Cell Identity")
```

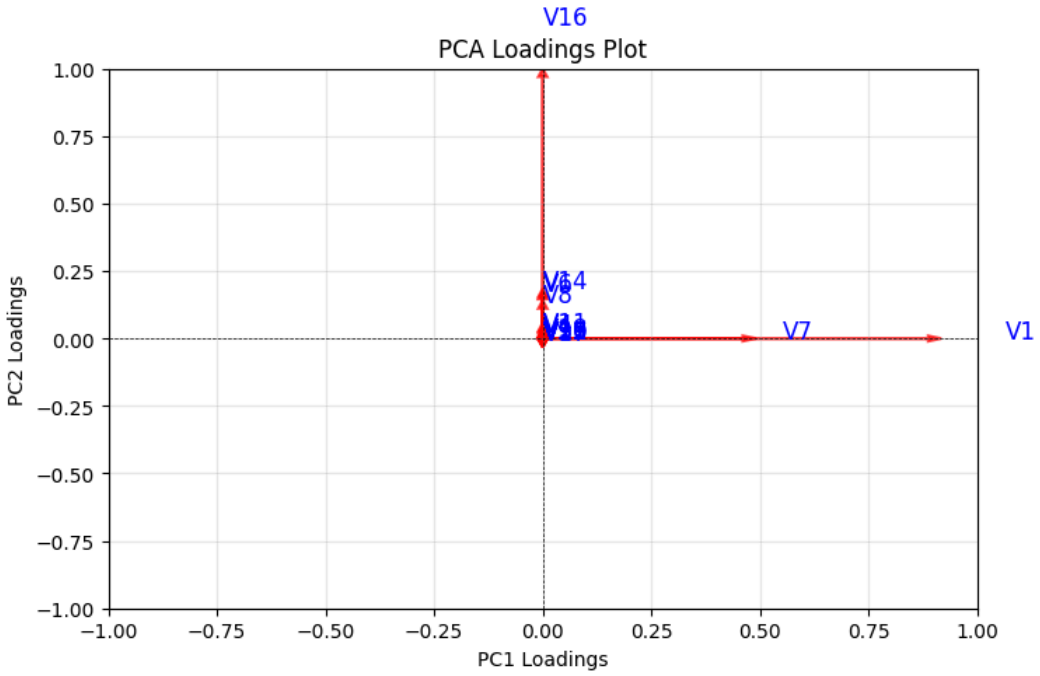
Out[3]:
```
<matplotlib.legend.Legend at 0x765d6fa10fe0>
```



- There is no clear seperation
- PC2 shows some seperation between the cell types with most of the WBCs closely associated with negative side of PC2
- MCF7 cells slighly positivley associated with PC2 and HL60 cells dispersed from negative to positive PC2

(3) Plot the loadings for the first two principal components. What do these tell you about the relative differences among the cells in (2)?

In [ ]:
```
# Plot Loadings
plt.figure(figsize=(8, 5))
for i, feature in enumerate(X.columns):
    plt.arrow(0, 0, pca.components_[0, i], pca.components_[1, i],
              color='red', alpha=0.7, head_width=0.02, linewidth=1.5)
    plt.text(pca.components_[0, i] * 1.2, pca.components_[1, i] * 1.2,
             feature, color='blue', fontsize=12)

plt.title("PCA Loadings Plot")
plt.xlabel("PC1 Loadings")
plt.ylabel("PC2 Loadings")
plt.axhline(0, color='black', linewidth=0.5, linestyle='--')
plt.axvline(0, color='black', linewidth=0.5, linestyle='--')
plt.grid(alpha=0.3)
plt.xlim(-1, 1)
plt.ylim(-1, 1)
plt.show()
```

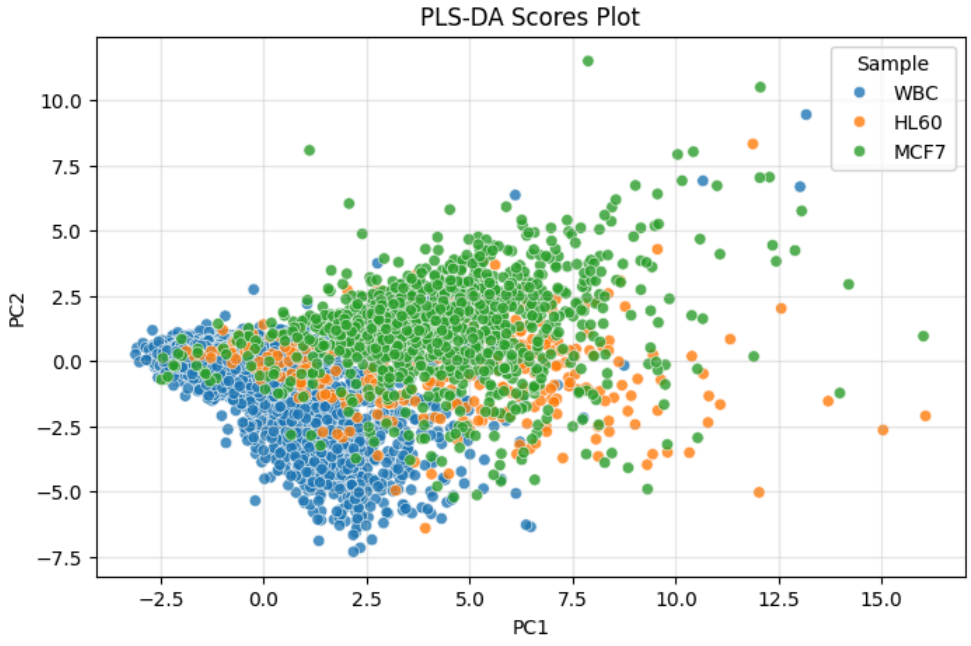


- V16 is strongly separated along PC2, which suggests that the cell identities (WBC, MCF7, HL60) could differ significantly based on these features on how they influence on these principal components
- PC1 does not clearly differentiate between the cells. Therefore, V1 and V7 which are highly correlated with PC1 may not be important in delineating the celltypes

(4) How does partial least squares regression fare in discriminating cancer and non-cancer cells (this is called PLS-DA)? How are these results similar or different to those in the previous question?

In [5]:
```
# This makes cancer=1 and not cancer=0
y_binary = np.logical_not(label_binarize(y, classes=['WBC', 'HL60', 'MCF7'])[:, 0])

plsr = PLSRegression(n_components=2)
plsr.fit(X, y_binary)
scores = plsr.x_scores_

# Plot Scores
plt.figure(figsize=(8, 5))
plt.grid(alpha=0.3)
plt.title("PLS-DA Scores Plot")
plt.xlabel("PC1")
plt.ylabel("PC2")
sns.scatterplot(x=scores[:, 0], y=scores[:, 1], hue=y, alpha=0.8)
```

Out[5]:
```
<Axes: title={'center': 'PLS-DA Scores Plot'}, xlabel='PC1', ylabel='PC2'>
```



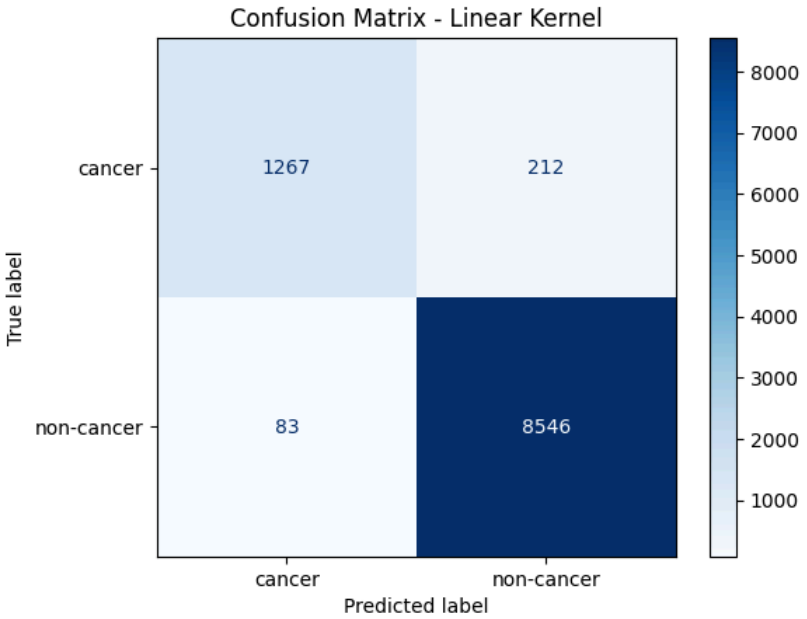
- With PLSR, the cancerous celltypes seem to be seperated well from the healthy leucocytes
- Comparatively better than PCA
- There is still considerable overlap between the HL60 and MCF70 cells (may need other mechanical and morphometric features to distinguish between these)

(5) Setup a support vector machine classifier (with linear kernel) to distinguish cancer and non-cancer.

Evaluate how well this performs.

In [7]:
```
# Convert target labels to binary format
cancer_type = np.array(["non-cancer" if label == 'WBC' else "cancer" for label in y])

# Encode labels as integers for training the SVM
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_encoded = le.fit_transform(cancer_type)  # "non-cancer" -> 0, "cancer" -> 1

# Train the SVM with a linear kernel
X_norm = stats.zscore(X)
svm_linear = SVC(random_state=42, kernel='linear')
svm_linear.fit(X_norm, y_encoded)  # Use numeric binary labels for training

# Predict using the trained model
y_pred_linear = svm_linear.predict(X_norm)

# Generate the confusion matrix and score
cm_linear = confusion_matrix(y_encoded, y_pred_linear)
print("Confusion Matrix - Linear SVM:\n", cm_linear)

score_linear = svm_linear.score(X_norm, y_encoded)
print("Linear SVM Accuracy: ", score_linear)

# Display the confusion matrix with original labels
disp_linear = ConfusionMatrixDisplay(confusion_matrix=cm_linear, display_labels=le.classes_)
disp_linear.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix - Linear Kernel")
plt.show()
```

```
Confusion Matrix - Linear SVM:
 [[1267  212]
 [  83 8546]]
Linear SVM Accuracy:  0.970815195884448
```

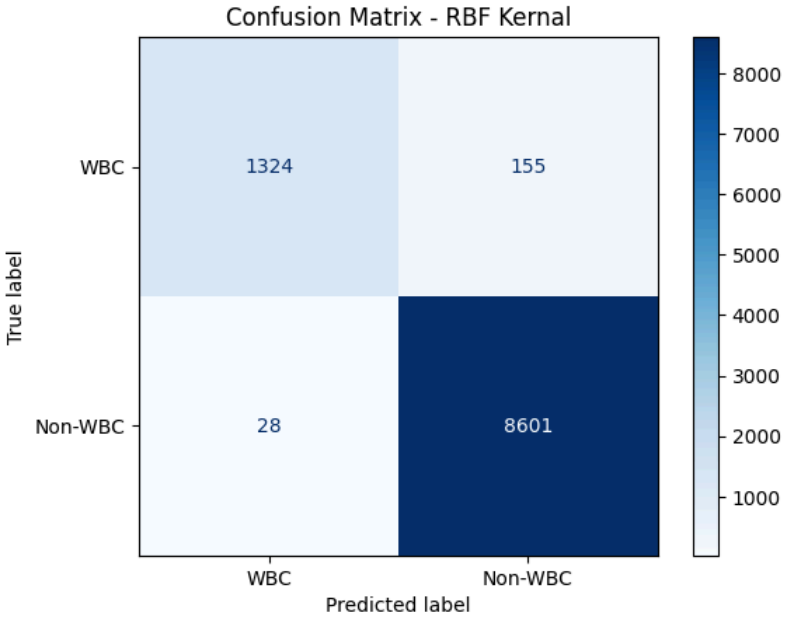


(6) Why might an alternative kernel, like `rbf`, be more effective in this application?

In [ ]:
```
# Normalize the data
X_norm = stats.zscore(X)

# Train the SVM with a rbf kernel
svm_rbf = SVC(kernel='rbf', random_state=42)
svm_rbf.fit(X_norm, cancer_type)

# Predict using the trained model
y_pred = svm_rbf.predict(X_norm)

# Generate the confusion matrix and score
cm = confusion_matrix(cancer_type, y_pred)
```

```
print(cm)

score = svm_rbf.score(X_norm, cancer_type)
print("RBF SVM: ", score)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["WBC", "Non-WBC"])
disp.plot(cmap="Blues", values_format="d")

plt.title("Confusion Matrix - RBF Kernal")
plt.show()
```

```
[[1324  155]
 [  28 8601]]
RBF SVM:  0.9818955282944203
```



- The RBF kernel might be helpful in casses where the data is not linearly separable. With the help of the RBF kernel, the data can be projected into a higher dimensional space where the points can are seperable

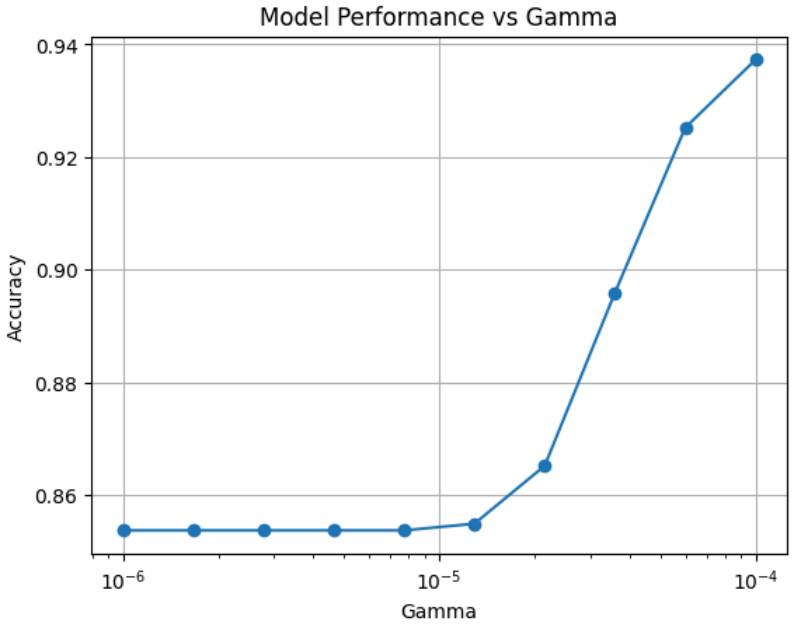
(7) Vary the radius parameter of your kernel and evaluate the ability of it to descriminate data both when fitting and on cross-validation. Is the choice of the best radius different based on whether you are performing cross-validation?

In [ ]:

```
from sklearn.model_selection import StratifiedKFold

# Normalize the data
X_norm = stats.zscore(X)

# Initialize Stratified K-Folds
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Define gamma values
gamma_values = np.logspace(-6, -4, 10)

score = np.zeros(len(gamma_values))
best_score = 0
best_gamma = 0

# Loop over gamma values
for i, gamma in enumerate(gamma_values):
    fold_scores = []

    # Cross-validation
    for train_index, test_index in kf.split(X_norm, y):
        trainX, testX = X_norm.iloc[train_index], X_norm.iloc[test_index]
        trainY, testY = y.iloc[train_index], y.iloc[test_index]

        # Train the model with RBF kernel and current gamma
        model = SVC(kernel='rbf', gamma=gamma, random_state=42)
        model.fit(trainX, trainY)
        fold_scores.append(model.score(testX, testY))

    # Store the mean score for this gamma
    score[i] = np.mean(fold_scores)
    print(f"For gamma = {gamma}, the score is: {score[i]:.4f}")

    # Track the best performing gamma
    if score[i] > best_score:
        best_gamma = gamma
        best_score = score[i]
```

```
print(f"Best Score with CV = {best_score:.4f} with gamma = {best_gamma}")

# Save scores to a DataFrame
score_df = pd.DataFrame({'Gamma': gamma_values, 'Score': score})
print(score_df)

# Plot Gamma vs Score
plt.plot(gamma_values, score, marker='o')
plt.xscale('log')
plt.xlabel('Gamma')
plt.ylabel('Accuracy')
plt.title('Model Performance vs Gamma')
plt.grid()
plt.show()
```

```
For gamma = 1e-06, the score is: 0.8537
For gamma = 1.6681005372000591e-06, the score is: 0.8537
For gamma = 2.782559402207126e-06, the score is: 0.8537
For gamma = 4.641588833612782e-06, the score is: 0.8537
For gamma = 7.742636826811277e-06, the score is: 0.8537
For gamma = 1.2915496650148827e-05, the score is: 0.8549
For gamma = 2.1544346900318823e-05, the score is: 0.8652
For gamma = 3.5938136638046256e-05, the score is: 0.8959
For gamma = 5.994842503189409e-05, the score is: 0.9252
For gamma = 0.0001, the score is: 0.9373
Best Score with CV = 0.9373 with gamma = 0.0001
      Gamma     Score
0  0.000001  0.853680
1  0.000002  0.853680
2  0.000003  0.853680
3  0.000005  0.853680
4  0.000008  0.853680
5  0.000013  0.854867
6  0.000022  0.865156
7  0.000036  0.895922
8  0.000060  0.925207
9  0.000100  0.937277
```



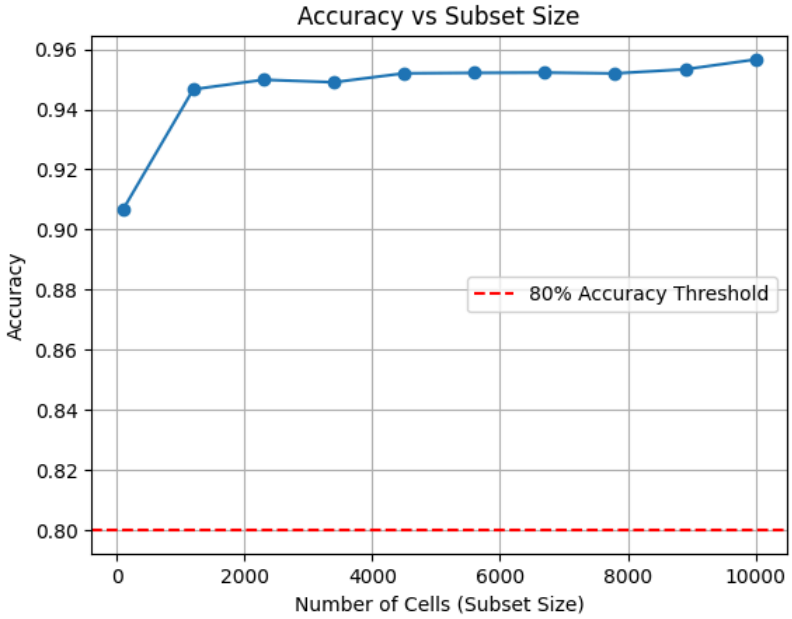
- A smaller gamma leads to a larger radius, and a larger gamma leads to a smaller radius.
- Low gamma - The model struggles to differentiate classes, as the decision boundary is too smooth (high bias, low variance)
- High gamma - The model tightly follows the training data but may perform poorly on new data due to overfitting (low bias, high variance)

(8) Your experimental collaborator asks you how many cells she needs to collect to build a classifier with 80% accuracy. Determine this number empirically.

In [11]:
```
subset_sizes = np.linspace(100, 10000, 10, dtype=int)
accuracies = []
threshold = 0.8

# Normalize the data
X_norm = stats.zscore(X)

# Iterate through different subset sizes
for size in subset_sizes:
    trial_accuracies = []

    for _ in range(5):
        # Subsample the data
        X_subset, _, y_subset, _ = train_test_split(X_norm, y, train_size=size, stratify=y, random_state=np.random.randint(1000))
```

```
        # Split the subset into train and test
        X_train, X_test, y_train, y_test = train_test_split(X_subset, y_subset, test_size=0.3, stratify=y_subset, random_state=42)

        # Train the classifier (SVM with RBF kernel)
        model = SVC(kernel='rbf', C=1, random_state=42)
        model.fit(X_train, y_train)

        # Evaluate accuracy
        y_pred = model.predict(X_test)
        trial_accuracies.append(accuracy_score(y_test, y_pred))

    # Average accuracy across trials for this subset size
    accuracies.append(np.mean(trial_accuracies))
    print(f"Subset size: {size}, Average accuracy: {accuracies[-1]:.4f}")

# Determine minimum sample size for >=80% accuracy
for size, acc in zip(subset_sizes, accuracies):
    if acc >= threshold:
        print(f"Minimum number of cells required for 80% accuracy: {size}")
        break

# Plot accuracy vs subset size
plt.plot(subset_sizes, accuracies, marker='o')
plt.axhline(y=threshold, color='r', linestyle='--', label='80% Accuracy Threshold')
plt.xlabel('Number of Cells (Subset Size)')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Subset Size')
plt.legend()
plt.grid()
plt.show()
```

```
Subset size: 100, Average accuracy: 0.9067
Subset size: 1200, Average accuracy: 0.9467
Subset size: 2300, Average accuracy: 0.9499
Subset size: 3400, Average accuracy: 0.9490
Subset size: 4500, Average accuracy: 0.9520
Subset size: 5600, Average accuracy: 0.9521
Subset size: 6700, Average accuracy: 0.9522
Subset size: 7800, Average accuracy: 0.9520
Subset size: 8900, Average accuracy: 0.9533
Subset size: 10000, Average accuracy: 0.9566
Minimum number of cells required for 80% accuracy: 100
```



- The classifier is able to classify the points with >80% accuracy even with a 100 cells
- This may be due to the huge class imbalance in the dataset with a lot more cancer cells than WBCs

(9) You want to save your fit model, but want to reduce it so that you don't have to carry around all the data points. Which points could you remove and still have the same SVM model in the end?

- You can save the support vectors and remove the rest of the points that are far away from the support vectors. Doing this, we will still have the same SVM model in the end