```
#importing required libraries
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.metrics import classification_report
```

```
#loading the datset
bc = load_breast_cancer()
```

```
# printing the names of the features
print("Features: ", bc.feature_names)

# printing the label type of cancer('malignant' 'benign')
print("Labels: ", bc.target_names)
```

```
Features:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Labels:  ['malignant' 'benign']
```

```
#splitting into features and target
X = bc.data
y = bc.target
```

```
#Splitting the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
#KNN classifier with defining hyperparameters
knn_classifier = KNeighborsClassifier()
param_grid = {
    'n_neighbors': range(1, 21),
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'p': [1, 2]  # for Manhattan and Euclidean distance metrics
}
```

```
# Random search for hyperparameter tuning
random_search = RandomizedSearchCV(estimator=knn_classifier, param_distributions=param_grid, n_iter=20, cv=5)
random_search.fit(X_train, y_train)

best_params = random_search.best_params_
best_model = random_search.best_estimator_
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'weights': 'uniform', 'p': 1, 'n_neighbors': 6, 'algorithm': 'auto'}
```

```
#Cross validation
cv_scores = cross_val_score(best_model, X_train, y_train, cv=10)
print("Cross-Validation Scores:", cv_scores)
print("Mean Cross-Validation Score:", np.mean(cv_scores))
```

```
Cross-Validation Scores: [0.97826087 0.97826087 0.91304348 0.86956522 0.95652174 1.
 0.93333333 0.91111111 0.95555556 0.91111111]
Mean Cross-Validation Score: 0.9406763285024156
```

```
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
```

```
# Calculate True Positives (TP), True Negatives (TN), False Positives (FP), False Negatives (FN)
TP = np.sum((y_pred == 1) & (y_test == 1))
TN = np.sum((y_pred == 0) & (y_test == 0))
FP = np.sum((y_pred == 1) & (y_test == 0))
FN = np.sum((y_pred == 0) & (y_test == 1))

accuracy = (TP + TN) / len(y_test)

precision = TP / (TP + FP)

sensitivity = TP / (TP + FN)

specificity = TN / (TN + FP)

print("Performance Metrics:")
print("Test Accuracy:", accuracy)
print("Precision:", precision)
print("Sensitivity (Recall):", sensitivity)
print("Specificity:", specificity)
```

```
Performance Metrics:
Test Accuracy: 0.956140350877193
Precision: 0.9459459459459459
Sensitivity (Recall): 0.9859154929577465
Specificity: 0.9069767441860465
```

```
report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", report)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.91      0.94        43
           1       0.95      0.99      0.97        71

    accuracy                           0.96       114
   macro avg       0.96      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114
```