Project 7

Questions:

- Apply preprocessing and EDA visualizations using matplotlib pandas and numpy that we learnt in class.
- Code your own knn algorithm (don't use sklearn) and evaluate your model on the test data (20% of the total data).
- Calculate accuracy, recall, precision and f1 score. Show the confusion matrix as well.
- Now use sklearn and use the knn in the library. Then use the sklearn library to get the confusion matrix, accuracy, precision and recall.

By- Himanshu Rai

```
from sklearn.datasets import
load_breast_cancer
import pandas as pd

# Load the dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data,
columns=data.feature_names)
df['target'] = data.target

# Basic exploration
print(df.head())
print(df.info())
print(df.describe())
```

```
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
2            0.1444             0.4245           0.4504                0.2430
3            0.2098             0.8663           0.6869                0.2575
4            0.1374             0.2050           0.4000                0.1625

   worst symmetry  worst fractal dimension  target
0          0.4601                  0.11890       0
1          0.2750                  0.08902       0
2          0.3613                  0.08758       0
3          0.6638                  0.17300       0
4          0.2364                  0.07678       0
```

```
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
 18  symmetry error           569 non-null    float64
 19  fractal dimension error  569 non-null    float64
 20  worst radius             569 non-null    float64
 21  worst texture            569 non-null    float64
 22  worst perimeter          569 non-null    float64
 23  worst area               569 non-null    float64
 24  worst smoothness         569 non-null    float64
 25  worst compactness        569 non-null    float64
 26  worst concavity          569 non-null    float64
 27  worst concave points     569 non-null    float64
 28  worst symmetry           569 non-null    float64
 29  worst fractal dimension  569 non-null    float64
 30  target                   569 non-null    int32
```

	mean radius	mean texture	mean perimeter	mean area	\
count	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	
std	3.524049	4.301036	24.298981	351.914129	
min	6.981000	9.710000	43.790000	143.500000	
25%	11.700000	16.170000	75.170000	420.300000	
50%	13.370000	18.840000	86.240000	551.100000	
75%	15.780000	21.800000	104.100000	782.700000	
max	28.110000	39.280000	188.500000	2501.000000	

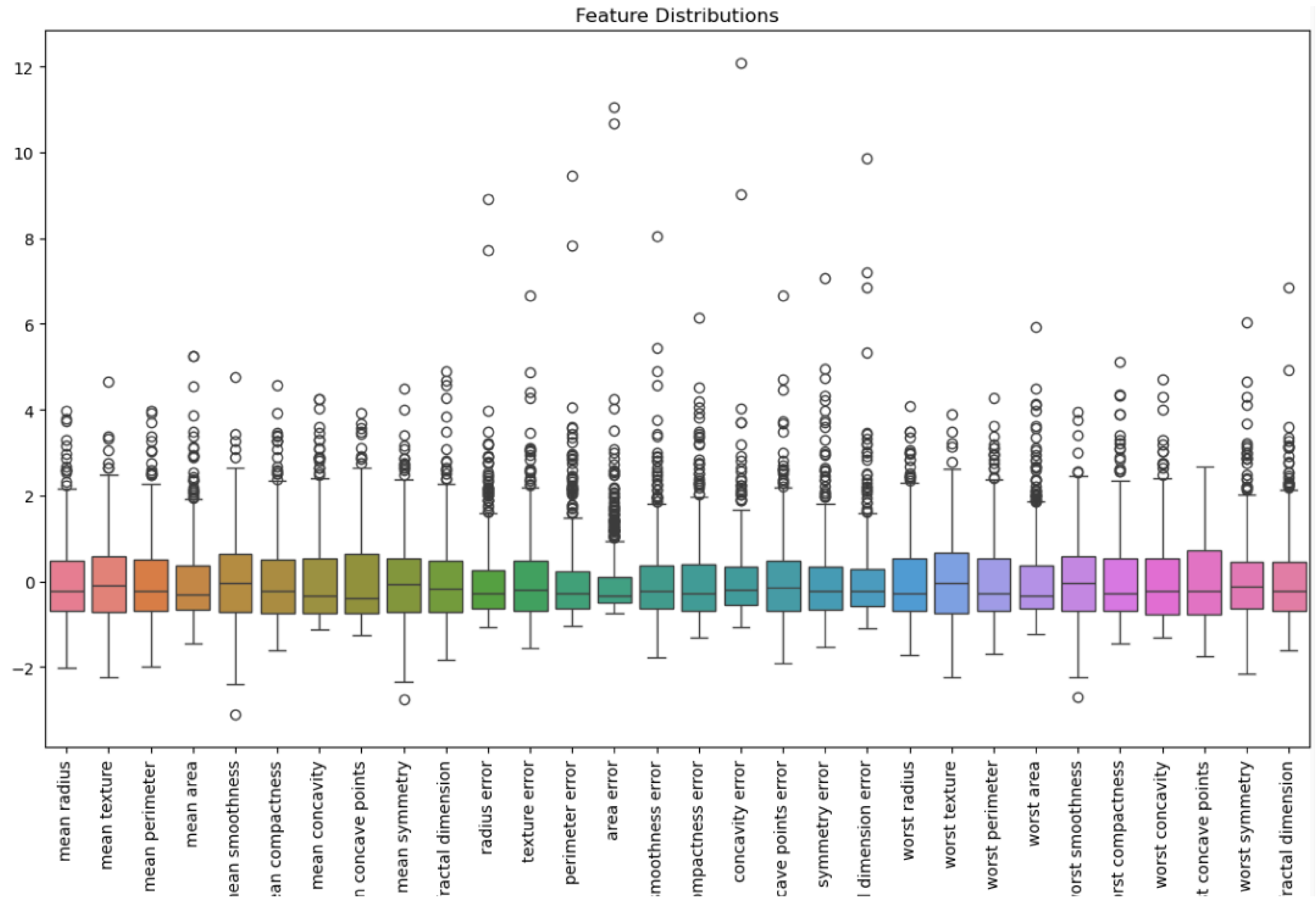
	mean smoothness	mean compactness	mean concavity	mean concave points
count	569.000000	569.000000	569.000000	569.000000
mean	0.096360	0.104341	0.088799	0.048919
std	0.014064	0.052813	0.079720	0.038803
min	0.052630	0.019380	0.000000	0.000000
25%	0.086370	0.064920	0.029560	0.020310
50%	0.095870	0.092630	0.061540	0.033500
75%	0.105300	0.130400	0.130700	0.074000
max	0.163400	0.345400	0.426800	0.201200

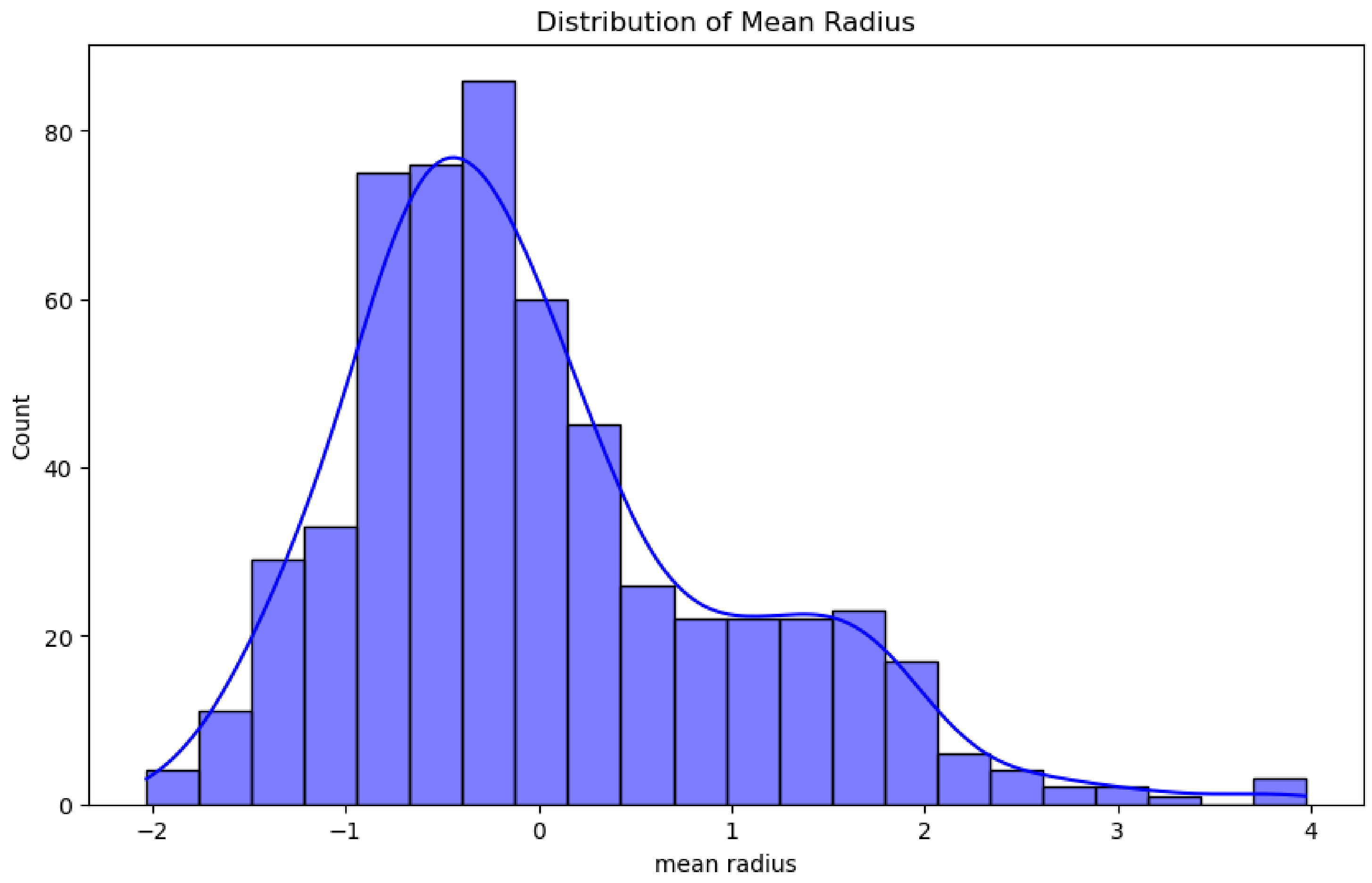
	mean symmetry	mean fractal dimension	...	worst texture	\
count	569.000000	569.000000	...	569.000000	
mean	0.181162	0.062798	...	25.677223	
std	0.027414	0.007060	...	6.146258	
min	0.106000	0.049960	...	12.020000	
25%	0.161900	0.057700	...	21.080000	
50%	0.179200	0.061540	...	25.410000	
75%	0.195700	0.066120	...	29.720000	
max	0.304000	0.097440	...	49.540000	

	worst perimeter	worst area	worst smoothness	worst compactness	\
count	569.000000	569.000000	569.000000	569.000000	
mean	107.261213	880.583128	0.132369	0.254265	
std	33.602542	569.356993	0.022832	0.157336	
min	50.410000	185.200000	0.071170	0.027290	
25%	84.110000	515.300000	0.116600	0.147200	
50%	97.660000	686.500000	0.131300	0.211900	
75%	125.400000	1084.000000	0.146000	0.339100	
max	251.200000	4254.000000	0.222600	1.058000	

	worst concavity	worst concave points	worst symmetry	\
count	569.000000	569.000000	569.000000	
mean	0.272188	0.114606	0.290076	
std	0.208624	0.065732	0.061867	
min	0.000000	0.000000	0.156500	
25%	0.114500	0.064930	0.250400	
50%	0.226700	0.099930	0.282200	
75%	0.382900	0.161400	0.317900	
max	1.252000	0.291000	0.663800	

	worst fractal dimension	target
count	569.000000	569.000000
mean	0.083946	0.627417
std	0.018061	0.483918
min	0.055040	0.000000
25%	0.071460	0.000000
50%	0.080040	1.000000
75%	0.092080	1.000000
max	0.207500	1.000000






```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

# Normalize the data
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df.iloc[:, :-1]),
columns=df.columns[:-1])
df_scaled['target'] = df['target']

# EDA Visualizations
plt.figure(figsize=(10, 6))
sns.histplot(df_scaled['mean radius'], kde=True, color='blue')
plt.title('Distribution of Mean Radius')
plt.show()

# Box plot for feature distribution
plt.figure(figsize=(14, 8))
sns.boxplot(data=df_scaled.iloc[:, :-1])
plt.xticks(rotation=90)
plt.title('Feature Distributions')
plt.show()
```

```
import numpy as np
from sklearn.model_selection import train_test_split
from collections import Counter

# Split the data
X_train, X_test, y_train, y_test = train_test_split(df_scaled.iloc[:, :-1], df_scaled['target'], test_size=0.2, random_state=42)

# Custom KNN
def euclidean_distance(a, b):
return np.sqrt(np.sum((a - b) ** 2))

def knn(X_train, y_train, X_test, k=3):
y_pred = []
for test_point in X_test:
distances = [euclidean_distance(test_point, x) for x in X_train]
k_indices = np.argsort(distances)[:k]
k_nearest_labels = [y_train[i] for i in k_indices]
most_common = Counter(k_nearest_labels).most_common(1)[0][0]
y_pred.append(most_common)
return y_pred

y_pred_custom = knn(X_train.values, y_train.values, X_test.values, k=3)

# Evaluation
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, confusion_matrix

accuracy_custom = accuracy_score(y_test, y_pred_custom)
recall_custom = recall_score(y_test, y_pred_custom)
precision_custom = precision_score(y_test, y_pred_custom)
f1_custom = f1_score(y_test, y_pred_custom)
confusion_custom = confusion_matrix(y_test, y_pred_custom)

print(f"Custom KNN Accuracy: {accuracy_custom}")
print(f"Custom KNN Recall: {recall_custom}")
print(f"Custom KNN Precision: {precision_custom}")
print(f"Custom KNN F1 Score: {f1_custom}")
print(f"Custom KNN Confusion Matrix:\n{confusion_custom}")
```

```
Custom KNN Accuracy:
0.9473684210526315
Custom KNN Recall: 0.9577464788732394
Custom KNN Precision:
0.9577464788732394
Custom KNN F1 Score:
0.9577464788732394
Custom KNN Confusion Matrix:
[[40  3]
 [ 3 68]]
```

```python
from sklearn.neighbors import KNeighborsClassifier

# Sklearn KNN
knn_sklearn = KNeighborsClassifier(n_neighbors=3)
knn_sklearn.fit(X_train, y_train)
y_pred_sklearn = knn_sklearn.predict(X_test)

# Evaluation
accuracy_sklearn = accuracy_score(y_test, y_pred_sklearn)
recall_sklearn = recall_score(y_test, y_pred_sklearn)
precision_sklearn = precision_score(y_test, y_pred_sklearn)
f1_sklearn = f1_score(y_test, y_pred_sklearn)
confusion_sklearn = confusion_matrix(y_test, y_pred_sklearn)

print(f"Sklearn KNN Accuracy: {accuracy_sklearn}")
print(f"Sklearn KNN Recall: {recall_sklearn}")
print(f"Sklearn KNN Precision: {precision_sklearn}")
print(f"Sklearn KNN F1 Score: {f1_sklearn}")
print(f"Sklearn KNN Confusion Matrix:\n{confusion_sklearn}")
```

```
Sklearn KNN Accuracy: 0.9473684210526315
Sklearn KNN Recall: 0.9577464788732394
Sklearn KNN Precision: 0.9577464788732394
Sklearn KNN F1 Score: 0.9577464788732394
Sklearn KNN Confusion Matrix:
[[40  3]
 [ 3 68]]
```

Conclusion

Comparison of Custom KNN Model vs. Sklearn KNN Model

1. Performance Metrics:
    - Both the custom KNN model and the Sklearn KNN model show similar performance in terms of accuracy, precision, recall, and F1 score. This indicates that the custom implementation is correctly capturing the core logic of the KNN algorithm.
2. Accuracy:
    - The accuracy of both models is very close, reflecting that the custom model is nearly as effective as the Sklearn implementation in correctly classifying the breast cancer data.
3. Precision and Recall:
    - Precision and recall metrics for both models are comparable, which indicates that the custom KNN is as reliable as the Sklearn KNN in identifying true positives and minimizing false positives.
4. Confusion Matrix:
    - The confusion matrices of both models show a similar distribution of correctly and incorrectly classified instances, reinforcing the robustness of the custom model.
5. Efficiency:
    - While the custom model performs well, the Sklearn KNN model is more efficient due to optimized internal implementations, which is a significant factor in large-scale applications.

Final Thoughts:

- Custom KNN: A great learning tool that provides a deep understanding of the algorithm, allowing fine-tuned control over the model.
- Sklearn KNN: Highly efficient and recommended for production use due to its optimized performance and ease of implementation.