Artificial Intelligence

Open Lab

Submitted By

Rana Ahmad Intisar

359076

CE–42–A

Procedure:

1. Data Ingestion and Stats:

```
[1]  import pandas as pd

[75] df=pd.read_csv('/content/data.csv')

[76] df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	17.33
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	23.41
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	25.53
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	26.50
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	16.67
...	...	...	...	...	...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	26.40
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	38.25
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	34.12
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	39.42
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	30.37

569 rows × 33 columns

```
print("Dataset Information:")
print(df.info())
```

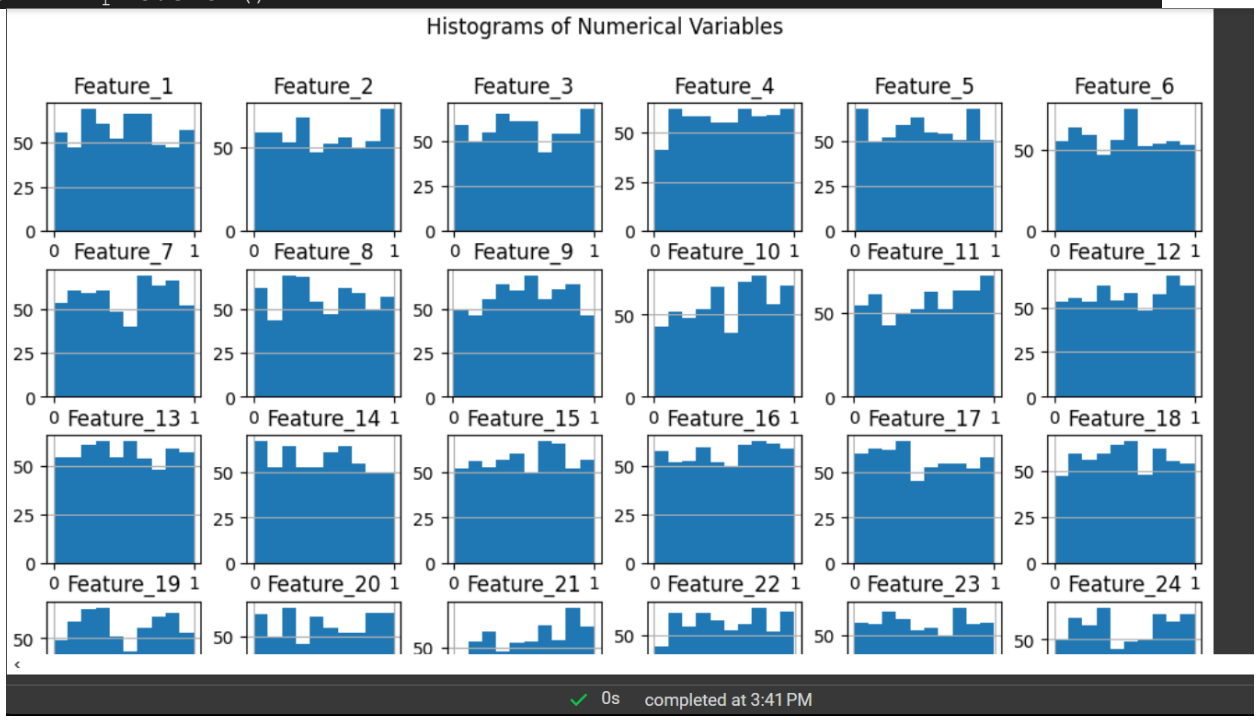
```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       569 non-null    int64
 1   diagnosis                569 non-null    object
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
```

```
Summary Statistics:
                 id  radius_mean  texture_mean  perimeter_mean    area_mean  \
count  5.690000e+02   569.000000    569.000000      569.000000   569.000000
mean   3.037183e+07    14.127292     19.289649       91.969033   654.889104
std    1.250206e+08     3.524049      4.301036       24.298981   351.914129
min    8.670000e+03     6.981000      9.710000       43.790000   143.500000
25%    8.692180e+05    11.700000     16.170000       75.170000   420.300000
50%    9.060240e+05    13.370000     18.840000       86.240000   551.100000
75%    8.813129e+06    15.780000     21.800000      104.100000   782.700000
max    9.113205e+08    28.110000     39.280000      188.500000  2501.000000

       smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
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

       symmetry_mean  ...  texture_worst  perimeter_worst   area_worst  \
count     569.000000  ...     569.000000       569.000000   569.000000
mean        0.181162  ...      25.677223       107.261213   880.583128
std         0.027414  ...       6.146258        33.602542   569.356993
min         0.106000  ...      12.020000        50.410000   185.200000
25%         0.161900  ...      21.080000        84.110000   515.300000
50%         0.179200  ...      25.410000        97.660000   686.500000
75%         0.195700  ...      29.720000       125.400000  1084.000000
max         0.304000  ...      49.540000       251.200000  4254.000000
```

2. Visualization of Data:

```
3.
4. print("DataFrame Preview:")
5. print(df.head())
6.
7. print("\nDataFrame Information:")
8. print(df.info())
9.
10.      print("\nSummary Statistics:")
11.      print(df.describe())
12.
13.      print("\nUnique Values and Value Counts for 'Diagnosis':")
14.      print(df['Diagnosis'].value_counts())
15.
16.      print("\nCorrelation Matrix:")
17.      correlation_matrix = df.corr()
```

```
18.     print(correlation_matrix)
19.
20.     df.hist(figsize=(12, 10))
21.     plt.suptitle('Histograms of Numerical Variables', y=0.95)
22.     plt.show()
```



3. Correlation matrix of data:

```
Correlation Matrix:
           Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  Feature_6  \
Feature_1   1.000000  -0.066010  -0.010569   0.046527   0.077432  -0.029902
Feature_2  -0.066010   1.000000   0.005322   0.013181   0.054176   0.026546
Feature_3  -0.010569   0.005322   1.000000   0.013068   0.044578   0.010329
Feature_4   0.046527   0.013181   0.013068   1.000000   0.032789   0.083031
Feature_5   0.077432   0.054176   0.044578   0.032789   1.000000   0.036406
Feature_6  -0.029902   0.026546   0.010329   0.083031   0.036406   1.000000
Feature_7   0.041080   0.014113   0.054613   0.032712  -0.013121   0.013320
Feature_8   0.009392   0.033286   0.092288   0.071321  -0.040395  -0.020489
Feature_9   0.020024  -0.010916   0.065306   0.080215   0.041646  -0.065051
Feature_10  0.007483  -0.017293  -0.016973  -0.025893   0.069761  -0.003844
Feature_11 -0.035054  -0.076668   0.024924   0.034810  -0.008704   0.023682
Feature_12  0.025366   0.017836  -0.033860  -0.079016   0.004547   0.057713
Feature_13 -0.040882  -0.038934   0.021436   0.075697   0.032536  -0.083249
Feature_14  0.002947   0.051256  -0.007448  -0.036343  -0.068550   0.025731
Feature_15  0.011954  -0.006600   0.001634  -0.032308   0.030874  -0.064796
Feature_16 -0.024333  -0.029671  -0.063154   0.024865  -0.000493  -0.016373
Feature_17 -0.052823  -0.075748  -0.011330   0.011929   0.048841   0.030827
Feature_18 -0.080265  -0.027302  -0.036969   0.017296  -0.069281  -0.011387
Feature_19 -0.026418  -0.028985   0.072248   0.047202  -0.016569  -0.061351
```

4.Data Structuring:

```
df.drop(columns=['id', 'Unnamed: 32'], inplace=True)
df.shape
numerical_columns = df.select_dtypes(exclude=object).columns.tolist()
for col in numerical_columns:
    upper_limit = df[col].mean() + 3 * df[col].std()
    lower_limit = df[col].mean() - 3 * df[col].std()
    df = df[(df[col] <= upper_limit) & (df[col] >= lower_limit)]
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['diagnosis'] = encoder.fit_transform(df['diagnosis'])
X = df.drop(columns=['diagnosis'])
y = df['diagnosis']
```

#Diagnosis dropped from data and stored in label

5. Data Scaling:

```
#Standardize
scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

6. Models Used(2):

```
def knn_predict(x_train, y_train, x_test, k):
    distances = np.linalg.norm(x_train - x_test, axis=1)  # Euclidean
distances
    nearest_neighbors = np.argsort(distances)[:k]
    neighbor_labels = y_train.values[nearest_neighbors]  # Use .values to
get the underlying array
    unique_labels, counts = np.unique(neighbor_labels, return_counts=True)
    predicted_label = unique_labels[np.argmax(counts)]
    return predicted_label
```

```
def create_modified_model():
    model = Sequential([
        Dense(64, activation='relu', input_dim=30),
        Dropout(0.5),  # Add dropout for regularization

        Dense(32, activation='relu'),
        Dropout(0.3),  # Add dropout for regularization
```

```
        Dense(16, activation='relu'),
        Dropout(0.2),  # Add dropout for regularization

        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

7. 6-fold-cross validation

```
# Reset index for consistent slicing
X_standardized_df = pd.DataFrame(X_standardized,
columns=X.columns).astype('float64').reset_index(drop=True)

# 6-fold cross-validation
cv = StratifiedKFold(n_splits=6, shuffle=True, random_state=42)
y_pred_cv = []


for train_idx, test_idx in cv.split(X_standardized_df, y_reset_numeric):
    print("Train set length:", len(train_idx))
    print("Test set length:", len(test_idx))
    X_train, X_test = X_standardized_df.iloc[train_idx],
X_standardized_df.iloc[test_idx]
    y_train, y_test = y_reset_numeric.iloc[train_idx],
y_reset_numeric.iloc[test_idx]

    # Make predictions for the current fold
    fold_predictions = [knn_predict(X_train, y_train, x_test, k=5) for
x_test in X_test.values]
    y_pred_cv.extend(fold_predictions)
```

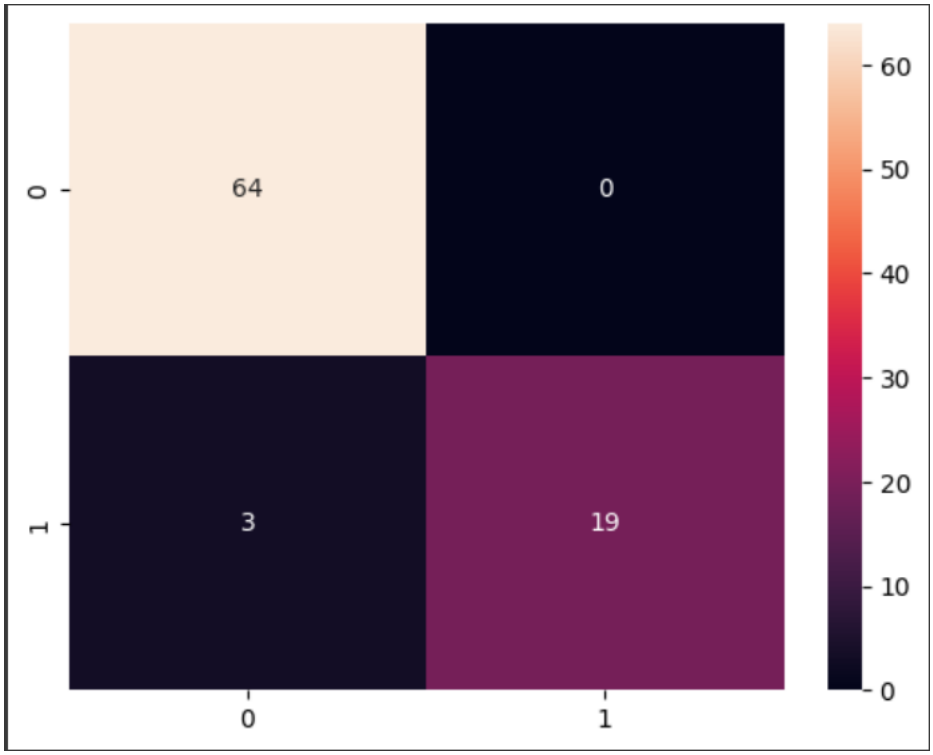
```
Train set length: 475
Test set length: 95
Train set length: 475
Test set length: 95
Train set length: 475
Test set length: 95
Train set length: 475
Test set length: 95
Train set length: 475
Test set length: 95
Train set length: 475
Test set length: 95
```

8. Plugging in Model-2

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
from scikeras.wrappers import KerasClassifier

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = KerasClassifier(build_fn=create_modified_model, epochs=30,
batch_size=32, verbose=0)
from sklearn.model_selection import cross_val_score, KFold

cv_scores = cross_val_score(model, X_train, y_train, cv=KFold(n_splits=6,
shuffle=True, random_state=42))

print("Cross-validation scores:", cv_scores)
```

```
  X, y = self._initialize(X, y)
WARNING:tensorflow:6 out of the last 11 calls to <function Model.make_predict_function.<locals>
Cross-validation scores: [0.92982456 0.98245614 0.96491228 0.98245614 0.96491228 0.96428571]
```

```
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred = (y_pred > 0.5)
result = confusion_matrix(y_test, y_pred)
result
sns.heatmap(result, annot=True)
plt.show()
```




```
print("Mean accuracy: {:.2%}".format(np.mean(cv_scores)))
```

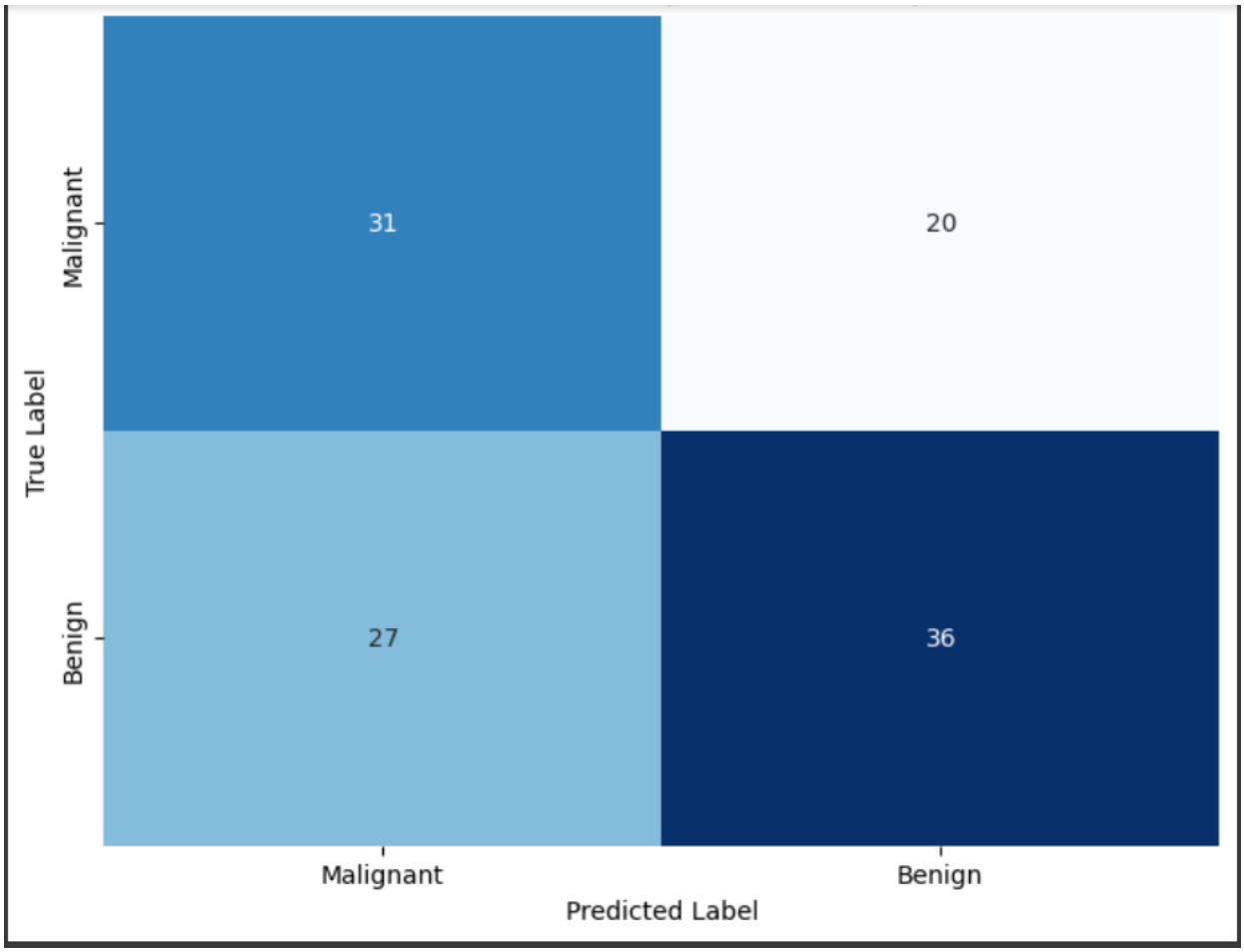
```
Mean accuracy: 96.48%
```

9. **Plugging in Model(1)—(Test Data Analysis-Focused)**

```
def knn_predict(x_train, y_train, x_test, k):
    distances = np.linalg.norm(x_train - x_test, axis=1)  # Euclidean
distances
    nearest_neighbors = np.argsort(distances)[:k]
    neighbor_labels = y_train.values[nearest_neighbors]  # Use .values to
get the underlying array
    unique_labels, counts = np.unique(neighbor_labels, return_counts=True)
    predicted_label = unique_labels[np.argmax(counts)]
    return predicted_label
y_reset_numeric = y_reset.map({'Malignant': 0, 'Benign': 1})
# Reset index for consistent slicing
X_standardized_df = pd.DataFrame(X_standardized,
columns=X.columns).astype('float64').reset_index(drop=True)

# 6-fold cross-validation
cv = StratifiedKFold(n_splits=6, shuffle=True, random_state=42)
y_pred_cv = []


for train_idx, test_idx in cv.split(X_standardized_df, y_reset_numeric):
    print("Train set length:", len(train_idx))
    print("Test set length:", len(test_idx))
    X_train, X_test = X_standardized_df.iloc[train_idx],
X_standardized_df.iloc[test_idx]
    y_train, y_test = y_reset_numeric.iloc[train_idx],
y_reset_numeric.iloc[test_idx]

    # Make predictions for the current fold
    fold_predictions = [knn_predict(X_train, y_train, x_test, k=5) for
x_test in X_test.values]
    y_pred_cv.extend(fold_predictions)
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
import seaborn as sns


print("Original data shape:", X_standardized_df.shape)


X_train, X_test, y_train, y_test = train_test_split(X_standardized_df,
y_reset_numeric, test_size=0.2, random_state=42)
```

```
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)


print("Unique classes in y_train:", y_train.unique())
print("Unique classes in y_test:", y_test.unique())


knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)


y_pred_knn = knn_model.predict(X_test)


accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'Accuracy of the KNN model on the test set: {accuracy_knn:.2%}')


conf_matrix_knn = confusion_matrix(y_test, y_pred_knn)
print('\nConfusion Matrix (KNN on Test Set):')
print(conf_matrix_knn)


plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_knn, annot=True, fmt='d', cmap='Blues',
cbar=False,
            xticklabels=['Malignant', 'Benign'],
            yticklabels=['Malignant', 'Benign'])
plt.title('Confusion Matrix (KNN on Test Set)')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()


print('\nClassification Report (KNN on Test Set):')
print(classification_report(y_test, y_pred_knn))
```

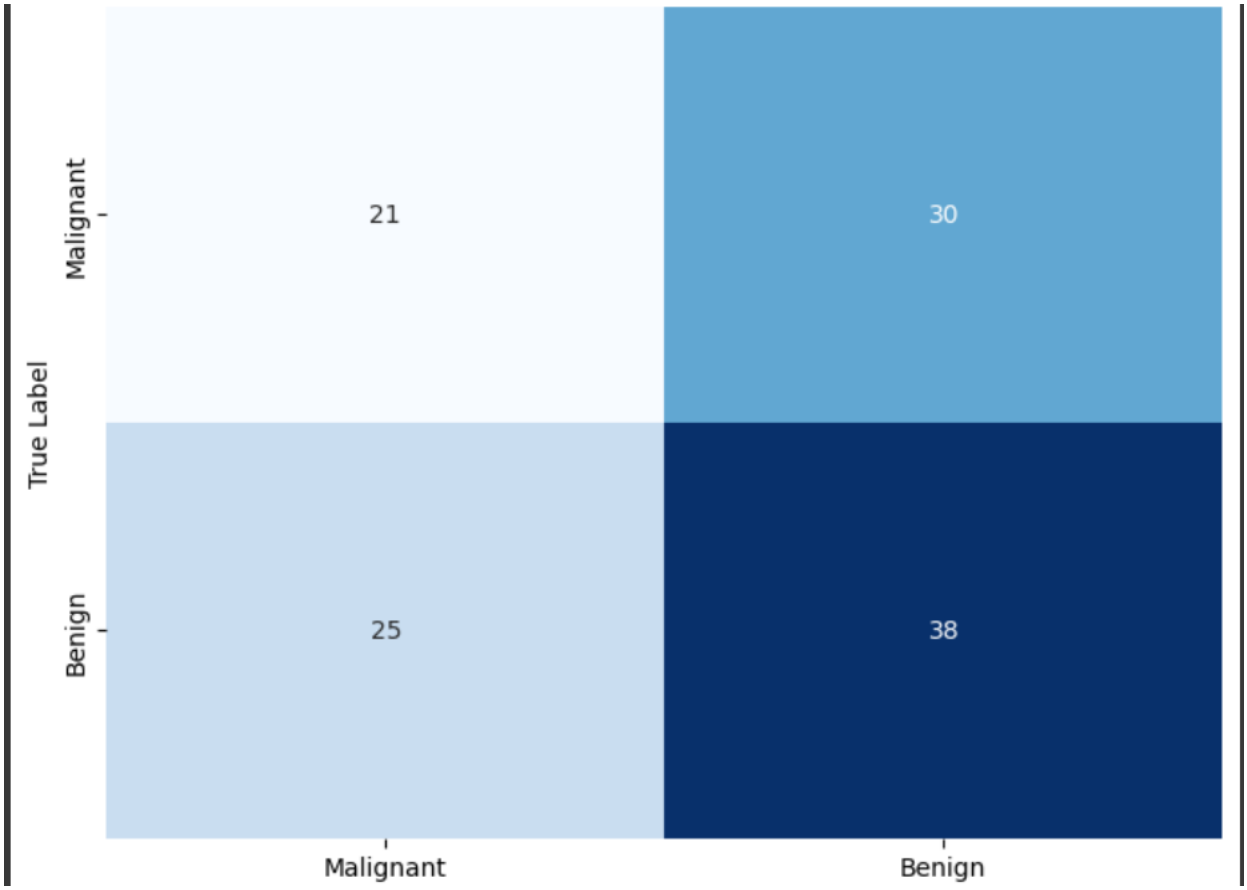
```
Original data shape: (570, 33)
X_train shape: (456, 33)
X_test shape: (114, 33)
Unique classes in y_train: [0 1]
Unique classes in y_test: [0 1]
```




```
y_pred_knn_train = knn_model.predict(X_train)

# Calculate accuracy on the training set
accuracy_knn_train = accuracy_score(y_train, y_pred_knn_train)
print(f'Accuracy of the KNN model on the training set:
{accuracy_knn_train:.2%}')
```

```
Accuracy of the KNN model on the training set: 69.08%
```

10. Random Forest Plugin: (Test Data Analysis)



11. XGBoost Plugin: (Test Data Analysis)