DSC630FinalProject

May 29, 2025

[52]:

```
#DSC630 Final Project
# Armin Heldovac

# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score, confusion_matrix, classification_report

# Load dataset
df = pd.read_csv('\\Users\\armin\\Downloads\\surveylungcancer.csv')
df.head()
```

[52]:

```
  GENDER  AGE  SMOKING  YELLOW_FINGERS  ANXIETY  PEER_PRESSURE  \
0      M   69        1               2        2              1
1      M   74        2               1        1              1
2      F   59        1               1        1              2
3      M   63        2               2        2              1
4      F   63        1               2        1              1

   CHRONIC DISEASE  FATIGUE   ALLERGY   WHEEZING  ALCOHOL CONSUMING  COUGHING  \
0                1         2         1         2                  2         2
1                2         2         2         1                  1         1
2                1         2         1         2                  1         2
3                1         1         1         1                  2         1
4                1         1         1         2                  1         2

   SHORTNESS OF BREATH  SWALLOWING DIFFICULTY  CHEST PAIN LUNG_CANCER
0                    2                      2           2         YES
1                    2                      2           2         YES
2                    2                      1           2          NO
```

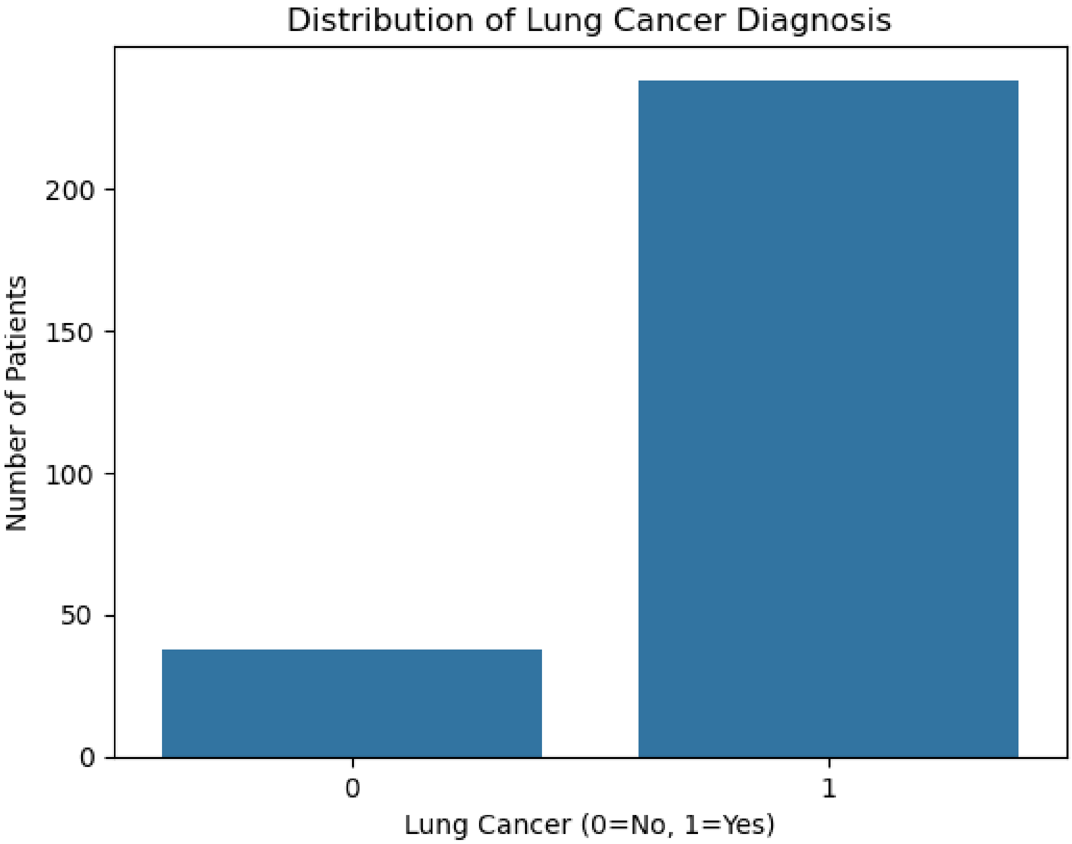
```
3                     1                       2            2           NO
4                     2                       1            1           NO
```

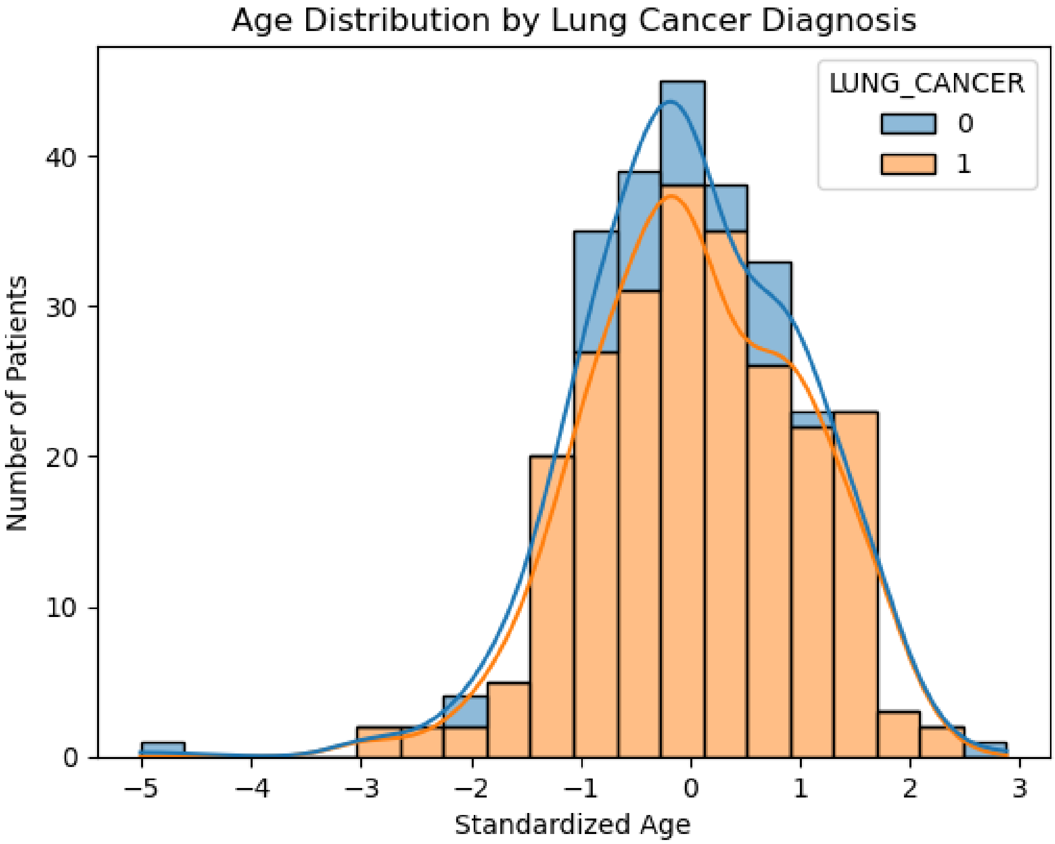
[54]:

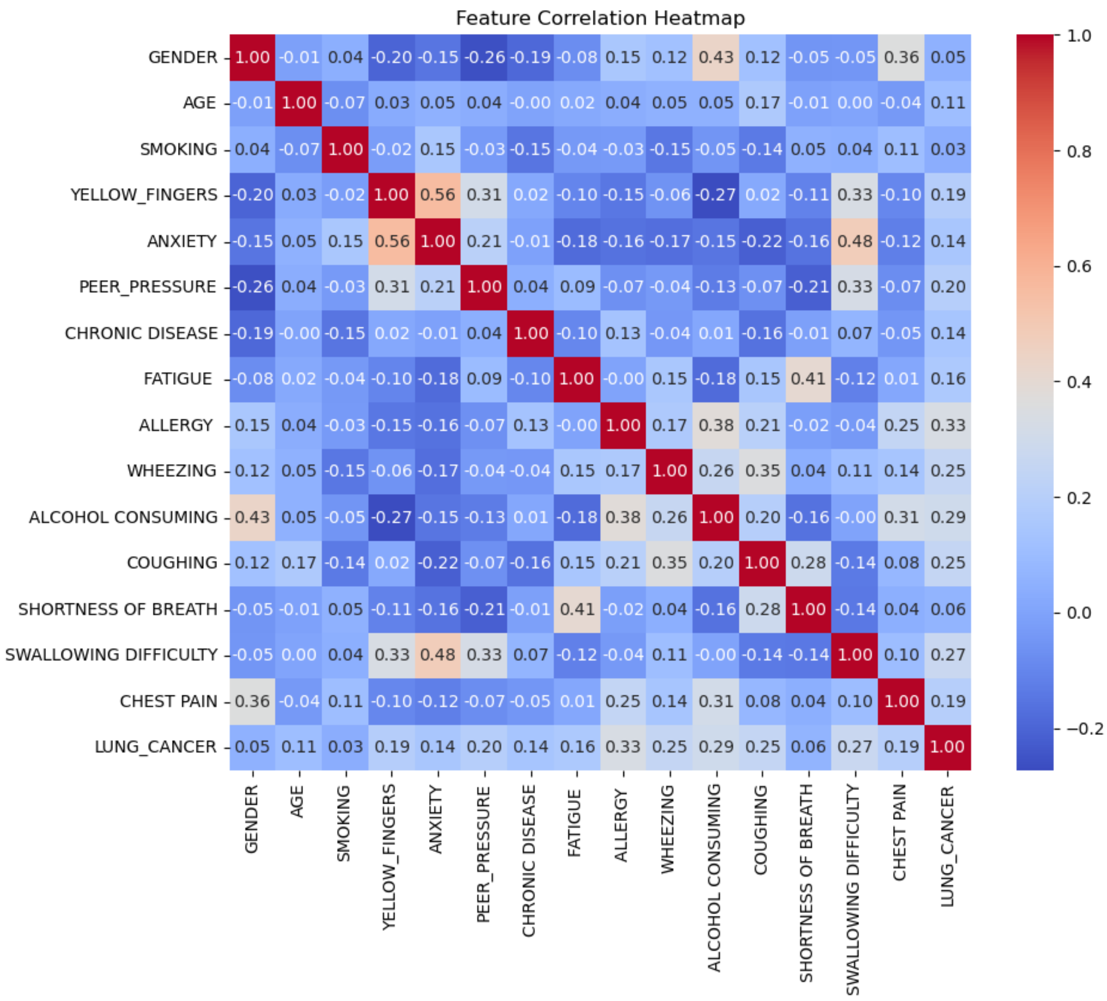
```
# Remove duplicates
df = df.drop_duplicates()

# Check and handle missing values (example: remove rows with missing values)
df = df.dropna()

# Encode categorical variables
label_encoders = {}
for column in ['GENDER', 'SMOKING', 'ALCOHOL CONSUMING', 'COUGHING', 'SHORTNESS␣
 ↪OF BREATH', 'CHEST PAIN', 'LUNG_CANCER']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Feature scaling for numerical variables
scaler = StandardScaler()
df['AGE'] = scaler.fit_transform(df[['AGE']])
```

[56]:

```
#Data Visualization

# Class distribution
sns.countplot(x='LUNG_CANCER', data=df)
plt.title('Distribution of Lung Cancer Diagnosis')
plt.xlabel('Lung Cancer (0=No, 1=Yes)')
plt.ylabel('Number of Patients')
plt.show()

# Age distribution by diagnosis
sns.histplot(data=df, x='AGE', hue='LUNG_CANCER', kde=True, multiple='stack')
plt.title('Age Distribution by Lung Cancer Diagnosis')
plt.xlabel('Standardized Age')
plt.ylabel('Number of Patients')
plt.show()

# Correlation heatmap
plt.figure(figsize=(10,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Feature Correlation Heatmap')
plt.show()
```


Distribution of Lung Cancer Diagnosis
Number of Patients
0
50
100
150
200
0
1
Lung Cancer (0=No, 1=Yes)


Age Distribution by Lung Cancer Diagnosis
LUNG_CANCER
0
1
Number of Patients
0
10
20
30
40
Standardized Age
−5
−4
−3
−2
−1
0
1
2
3



[57]:

```
#Model Building and Evaluation

# Define features and target
X = df.drop('LUNG_CANCER', axis=1)
y = df['LUNG_CANCER']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize and train logistic regression model
lr_model = LogisticRegression(class_weight='balanced', random_state=42)
lr_model.fit(X_train, y_train)

# Make predictions
y_pred = lr_model.predict(X_test)
```

```
y_prob = lr_model.predict_proba(X_test)[:,1]
```

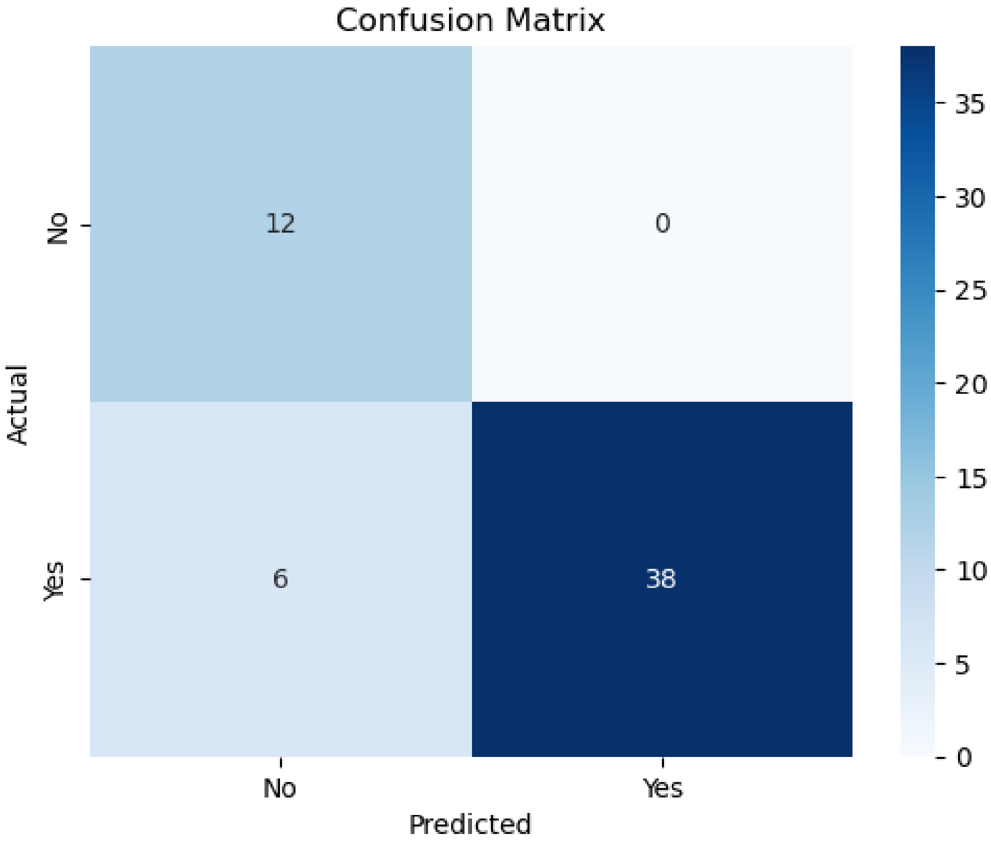
[58]:

```
# Evaluate model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_prob))
```

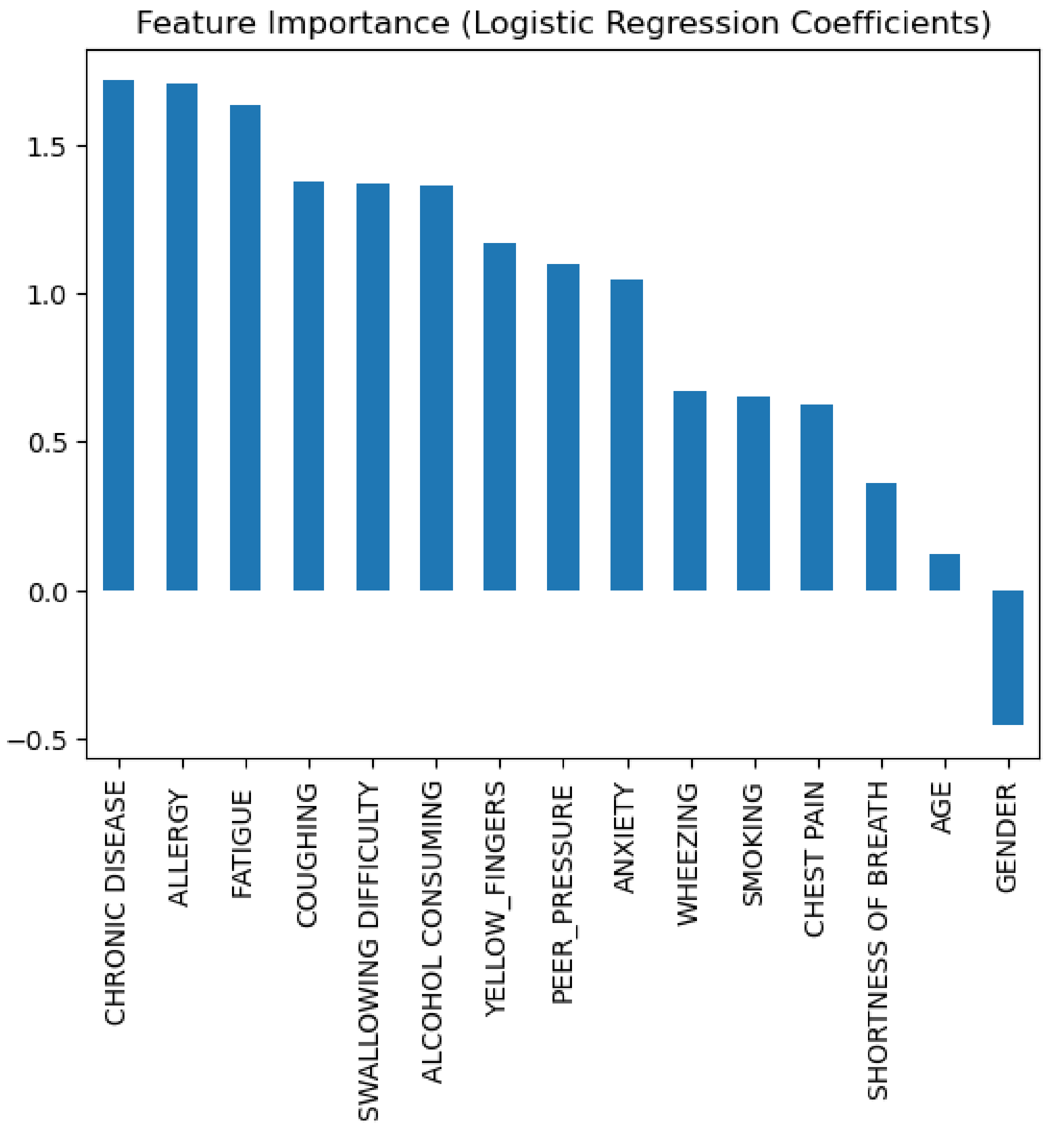
```
Accuracy: 0.8928571428571429
Precision: 1.0
Recall: 0.8636363636363636
F1 Score: 0.926829268292683
ROC-AUC Score: 0.9791666666666667
```

[62]:

```
# Confusion matrix visualization
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No', 'Yes'],␣
 ↪yticklabels=['No', 'Yes'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



[64]:
```
# Feature importance
feature_importance = pd.Series(lr_model.coef_[0], index=X.columns).
 ↪sort_values(ascending=False)
feature_importance.plot(kind='bar')
plt.title('Feature Importance (Logistic Regression Coefficients)')
plt.show()
```



[66]:
```
# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

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

           0       0.67      1.00      0.80        12
           1       1.00      0.86      0.93        44

    accuracy                           0.89        56
   macro avg       0.83      0.93      0.86        56
weighted avg       0.93      0.89      0.90        56
```

[68]:

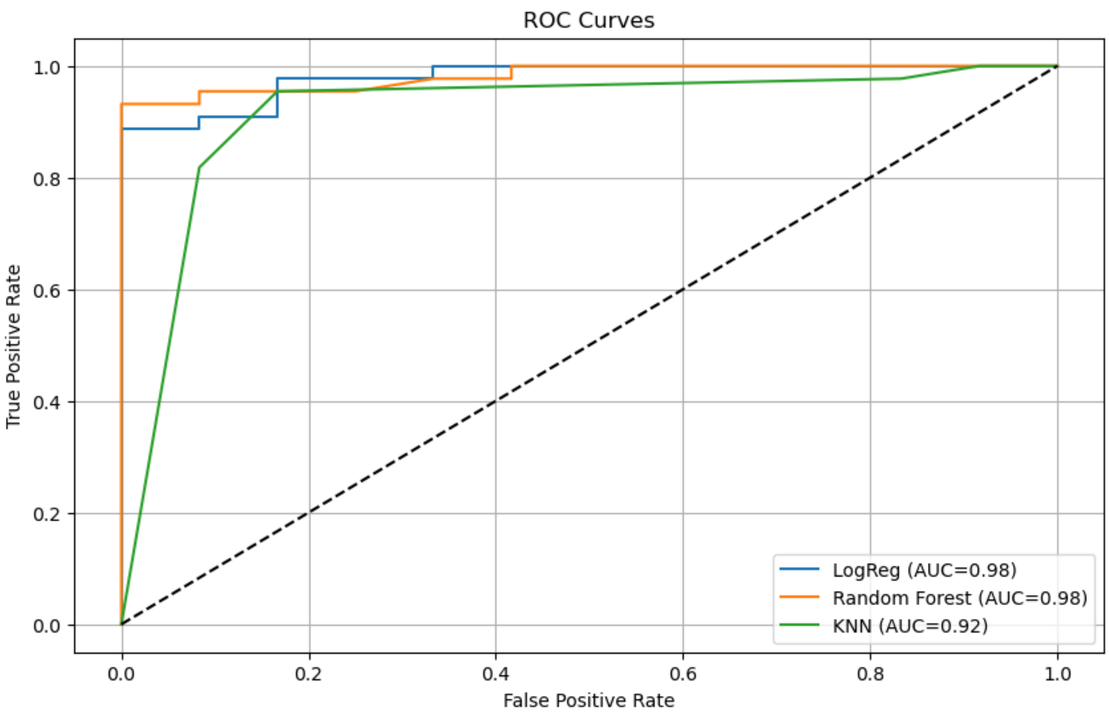
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import cross_val_score, GridSearchCV


# Train Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_preds = rf_model.predict(X_test)
rf_probs = rf_model.predict_proba(X_test)[:, 1]

# Train KNN
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
knn_preds = knn_model.predict(X_test)
knn_probs = knn_model.predict_proba(X_test)[:, 1]

# Evaluate
print("Random Forest:")
print("Accuracy:", accuracy_score(y_test, rf_preds))
print("Precision:", precision_score(y_test, rf_preds))
print("Recall:", recall_score(y_test, rf_preds))
print("F1 Score:", f1_score(y_test, rf_preds))
print("ROC-AUC:", roc_auc_score(y_test, rf_probs))

print("\nK-Nearest Neighbors:")
print("Accuracy:", accuracy_score(y_test, knn_preds))
print("Precision:", precision_score(y_test, knn_preds))
print("Recall:", recall_score(y_test, knn_preds))
print("F1 Score:", f1_score(y_test, knn_preds))
print("ROC-AUC:", roc_auc_score(y_test, knn_probs))
```

```
Random Forest:
Accuracy: 0.8571428571428571
Precision: 0.8461538461538461
```

```
Recall: 1.0
F1 Score: 0.9166666666666666
ROC-AUC: 0.9820075757575757

K-Nearest Neighbors:
Accuracy: 0.8035714285714286
Precision: 0.8113207547169812
Recall: 0.9772727272727273
F1 Score: 0.8865979381443299
ROC-AUC: 0.9176136363636364
```

[70]:

```
# ROC Curves
fpr_lr, tpr_lr, _ = roc_curve(y_test, lr_model.predict_proba(X_test)[:,1])
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_probs)
fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_probs)
plt.figure(figsize=(10,6))
plt.plot(fpr_lr, tpr_lr, label='LogReg (AUC={:.2f})'.
  ↪format(roc_auc_score(y_test, lr_model.predict_proba(X_test)[:,1])))
plt.plot(fpr_rf, tpr_rf, label='Random Forest (AUC={:.2f})'.
  ↪format(roc_auc_score(y_test, rf_probs)))
plt.plot(fpr_knn, tpr_knn, label='KNN (AUC={:.2f})'.
  ↪format(roc_auc_score(y_test, knn_probs)))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend()
plt.grid()
plt.show()
```



[72]:

```
# Cross-validation Recall
print("Logistic Regression CV Recall:", cross_val_score(lr_model, X, y, cv=5,␣
  ↪scoring='recall').mean())
print("Random Forest CV Recall:", cross_val_score(rf_model, X, y, cv=5,␣
  ↪scoring='recall').mean())
print("KNN CV Recall:", cross_val_score(knn_model, X, y, cv=5,␣
  ↪scoring='recall').mean())

# Grid Search for RF
param_grid_rf = {'n_estimators': [50, 100], 'max_depth': [None, 10],␣
  ↪'min_samples_split': [2, 5]}
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf,␣
  ↪cv=5, scoring='recall')
grid_rf.fit(X_train, y_train)
print("Best RF Params:", grid_rf.best_params_)
print("Best RF CV Recall:", grid_rf.best_score_)

# Grid Search for KNN
param_grid_knn = {'n_neighbors': [3, 5, 7]}
grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5,␣
  ↪scoring='recall')
grid_knn.fit(X_train, y_train)
print("Best KNN Params:", grid_knn.best_params_)
print("Best KNN CV Recall:", grid_knn.best_score_)
```

```
Logistic Regression CV Recall: 0.8780141843971631
Random Forest CV Recall: 0.9454787234042554
KNN CV Recall: 0.962145390070922
Best RF Params: {'max_depth': None, 'min_samples_split': 5, 'n_estimators': 100}
Best RF CV Recall: 0.9690958164642375
Best KNN Params: {'n_neighbors': 7}
Best KNN CV Recall: 0.9794871794871796
```

[ ]:

```
```

[ ]:

```
```