import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

!pip install tensorflow

Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0) Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (f Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.1 Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from te Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (fr Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (fr Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (f Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tens Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,< Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from ten Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from te Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (fr Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-pack Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3. Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packa Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10 Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages ( Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-package Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist- Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (fro Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3 Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (fro Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packag Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-package Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-pack Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-pack Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from r Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages ( Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages ( Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (f Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (fro

import pandas as pd

df = pd.read\_csv("BankChurn.csv") df.head()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CustomerId | CreditScore | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCar |
| 0 15634602 | 619 | Female | 42 | 2 | 0.00 | 1 | Ye |
| 1 15647311 | 608 | Female | 41 | 1 | 83807.86 | 1 | N |
| 2 15619304 | 502 | Female | 42 | 8 | 159660.80 | 3 | Ye |
| 3 15701354 | 699 | Female | 39 | 1 | 0.00 | 2 | N |
| 4 15737888 | 850 | Female | 43 | 2 | 125510.82 | 1 | Ye |



from sklearn.preprocessing import LabelEncoder le\_gender = LabelEncoder()

le\_Hascard = LabelEncoder() le\_Active = LabelEncoder()

df['Gender\_num'] = le\_gender.fit\_transform(df['Gender'])

df['Hascard\_num'] = le\_Hascard.fit\_transform(df['HasCrCard'])

df['Active\_num'] = le\_Active.fit\_transform(df['IsActiveMember'])

newdf = df.drop(['Gender','HasCrCard','IsActiveMember'],axis='columns') newdf

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CustomerId | CreditScore | Age | Tenure | Balance | NumOfProducts | Complain | Rati |
| 0 | 15634602 | 619 | 42 | 2 | 0.00 | 1 | Yes |  |
| 1 | 15647311 | 608 | 41 | 1 | 83807.86 | 1 | Yes |  |
| 2 | 15619304 | 502 | 42 | 8 | 159660.80 | 3 | Yes |  |
| 3 | 15701354 | 699 | 39 | 1 | 0.00 | 2 | No |  |
| 4 | 15737888 | 850 | 43 | 2 | 125510.82 | 1 | No |  |
| ... | ... | ... | ... | ... | ... | ... | ... |  |
| 415 | 15809616 | 626 | 26 | 8 | 0.00 | 2 | No |  |
| 416 | 15720559 | 487 | 61 | 5 | 110368.03 | 1 | Yes |  |
| 417 | 15695632 | 556 | 39 | 9 | 89588.35 | 1 | No |  |
| 418 | 15659843 | 643 | 46 | 6 | 0.00 | 2 | No |  |
| 419 | 15615624 | 605 | 28 | 6 | 0.00 | 2 | No |  |

420 rows × 13 columns

import pandas as pd import numpy as np

# Load your dataset

# Replace 'your\_dataset.csv' with the path to your dataset df = pd.read\_csv('BankChurn.csv')

# Separate the majority and minority classes majority\_class = df[df['Exited'] == 0]

minority\_class = df[df['Exited'] == 1]

# Determine the class with fewer samples

minority\_class\_count = len(minority\_class) majority\_class\_count = len(majority\_class)

# Oversample the minority class by randomly duplicating samples

oversampled\_minority\_class = minority\_class.sample(n=majority\_class\_count, replace=True, random\_sta

# Combine the oversampled minority class with the original majority class

oversampled\_df = pd.concat([majority\_class, oversampled\_minority\_class], axis=0)

# Shuffle the oversampled dataset

oversampled\_df = oversampled\_df.sample(frac=1, random\_state=42).reset\_index(drop=True)

# Check the class distribution after oversampling print("Class Distribution after Oversampling:") print(oversampled\_df['Exited'].value\_counts())

Class Distribution after Oversampling: Exited

1 335

0 335

Name: count, dtype: int64

Start coding or ge nerate with AI.

from sklearn.preprocessing import LabelEncoder le\_gender = LabelEncoder()

le\_Hascard = LabelEncoder() le\_Active = LabelEncoder()

oversampled\_df['Gender\_num'] = le\_gender.fit\_transform(oversampled\_df['Gender'])

oversampled\_df['Hascard\_num'] = le\_Hascard.fit\_transform(oversampled\_df['HasCrCard'])

oversampled\_df['Active\_num'] = le\_Active.fit\_transform(oversampled\_df['IsActiveMember'])

oversampled\_df

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CustomerId | CreditScore | Gender | Age | Tenure | Balance | NumOfProducts | HasCrCard | IsActiveMe |
| 0 | 15619955 | 733 | Male | 34 | 3 | 100337.96 | 3 | Yes |  |
| 1 | 15811127 | 521 | Male | 35 | 6 | 96423.84 | 1 | Yes |  |
| 2 | 15600974 | 516 | Male | 50 | 5 | 0.00 | 1 | No |  |
| 3 | 15723488 | 668 | Male | 47 | 7 | 106854.21 | 1 | No |  |
| 4 | 15659420 | 659 | Male | 32 | 3 | 107594.11 | 2 | Yes |  |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| 665 | 15731511 | 808 | Male | 45 | 7 | 118626.55 | 2 | Yes |  |
| 666 | 15712543 | 789 | Male | 39 | 7 | 124828.46 | 2 | Yes |  |
| 667 | 15763859 | 840 | Female | 43 | 7 | 0.00 | 2 | Yes |  |
| 668 | 15658929 | 683 | Male | 29 | 0 | 133702.89 | 1 | Yes |  |
| 669 | 15740404 | 758 | Female | 34 | 3 | 0.00 | 2 | Yes |  |



670 rows × 16 columns

newdf = oversampled\_df.drop(['Gender','HasCrCard','IsActiveMember'],axis='columns') newdf

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CustomerId | CreditScore | Age | Tenure | Balance | NumOfProducts | Complain | Rating | EstimatedSa |
| 0 | 15619955 | 733 | 34 | 3 | 100337.96 | 3 | Yes | 4 | 485 |
| 1 | 15811127 | 521 | 35 | 6 | 96423.84 | 1 | No | 4 | 104 |
| 2 | 15600974 | 516 | 50 | 5 | 0.00 | 1 | Yes | 5 | 1461 |
| 3 | 15723488 | 668 | 47 | 7 | 106854.21 | 1 | Yes | 3 | 1579 |
| 4 | 15659420 | 659 | 32 | 3 | 107594.11 | 2 | No | 3 | 1024 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| 665 | 15731511 | 808 | 45 | 7 | 118626.55 | 2 | No | 1 | 1471 |
| 666 | 15712543 | 789 | 39 | 7 | 124828.46 | 2 | No | 2 | 1244 |
| 667 | 15763859 | 840 | 43 | 7 | 0.00 | 2 | No | 2 | 909 |
| 668 | 15658929 | 683 | 29 | 0 | 133702.89 | 1 | Yes | 1 | 555 |
| 669 | 15740404 | 758 | 34 | 3 | 0.00 | 2 | No | 2 | 1242 |

670 rows × 13 columns

X = newdf.drop(['CustomerId','Complain','Exited'],axis='columns') y =newdf['Exited']

X

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | CreditScore | Age | Tenure | Balance | NumOfProducts | Rating | EstimatedSalary | Gender\_num | Hasc |
| 0 | 733 | 34 | 3 | 100337.96 | 3 | 4 | 48559.19 | 1 |  |
| 1 | 521 | 35 | 6 | 96423.84 | 1 | 4 | 10488.44 | 1 |  |
| 2 | 516 | 50 | 5 | 0.00 | 1 | 5 | 146145.93 | 1 |  |
| 3 | 668 | 47 | 7 | 106854.21 | 1 | 3 | 157959.02 | 1 |  |
| 4 | 659 | 32 | 3 | 107594.11 | 2 | 3 | 102416.84 | 1 |  |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |  |
| 665 | 808 | 45 | 7 | 118626.55 | 2 | 1 | 147132.46 | 1 |  |
| 666 | 789 | 39 | 7 | 124828.46 | 2 | 2 | 124411.08 | 1 |  |
| 667 | 840 | 43 | 7 | 0.00 | 2 | 2 | 90908.95 | 0 |  |
| 668 | 683 | 29 | 0 | 133702.89 | 1 | 1 | 55582.54 | 1 |  |
| 669 | 758 | 34 | 3 | 0.00 | 2 | 2 | 124226.16 | 0 |  |



670 rows × 10 columns

import pandas as pd import numpy as np

from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train) X\_test = scaler.transform(X\_test)

# Build the ANN model model = Sequential([

Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)), Dropout(0.5),

Dense(64, activation='relu'), Dropout(0.5),

Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=20, batch\_size=32, validation\_split=0.2)

Epoch 1/20

9/9 [==============================] - 1s 20ms/step - loss: 0.6529 - accuracy: 0.6306 - val\_lo Epoch 2/20

9/9 [==============================] - 0s 6ms/step - loss: 0.5868 - accuracy: 0.7537 - val\_los

Epoch 3/20

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 9/9 [==============================]  Epoch 4/20 | - 0s | 4ms/step | - loss: | 0.5313 | - accuracy: | 0.8097 | - val\_los |
| 9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.5431 | - accuracy: | 0.7948 | - val\_los |
| Epoch 5/20  9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.5431 | - accuracy: | 0.8097 | - val\_los |
| Epoch 6/20  9/9 [==============================] | - 0s | 6ms/step | - loss: | 0.4887 | - accuracy: | 0.8022 | - val\_los |
| Epoch 7/20  9/9 [==============================] | - 0s | 6ms/step | - loss: | 0.5128 | - accuracy: | 0.8060 | - val\_los |
| Epoch 8/20 |  |  |  |  |  |  |  |
| 9/9 [==============================] | - 0s | 3ms/step | - loss: | 0.5153 | - accuracy: | 0.8060 | - val\_los |
| Epoch 9/20  9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.5151 | - accuracy: | 0.8097 | - val\_los |
| Epoch 10/20  9/9 [==============================] | - 0s | 6ms/step | - loss: | 0.4859 | - accuracy: | 0.8022 | - val\_los |
| Epoch 11/20 |  |  |  |  |  |  |  |
| 9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.4736 | - accuracy: | 0.8134 | - val\_los |
| Epoch 12/20  9/9 [==============================] | - 0s | 6ms/step | - loss: | 0.4846 | - accuracy: | 0.8097 | - val\_los |
| Epoch 13/20  9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.4777 | - accuracy: | 0.8097 | - val\_los |
| Epoch 14/20  9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.4445 | - accuracy: | 0.8134 | - val\_los |
| Epoch 15/20 |  |  |  |  |  |  |  |
| 9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.4572 | - accuracy: | 0.8172 | - val\_los |
| Epoch 16/20  9/9 [==============================] | - 0s | 5ms/step | - loss: | 0.4509 | - accuracy: | 0.8097 | - val\_los |
| Epoch 17/20  9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.4490 | - accuracy: | 0.8060 | - val\_los |
| Epoch 18/20 |  |  |  |  |  |  |  |
| 9/9 [==============================]  Epoch 19/20 | - 0s | 6ms/step | - loss: | 0.4494 | - accuracy: | 0.8097 | - val\_los |
| 9/9 [==============================] | - 0s | 6ms/step | - loss: | 0.4655 | - accuracy: | 0.8134 | - val\_los |
| Epoch 20/20  9/9 [==============================] | - 0s | 4ms/step | - loss: | 0.4402 | - accuracy: | 0.8209 | - val\_los |

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test) print('Test Loss:', test\_loss)

print('Test Accuracy:', test\_accuracy)

3/3 [==============================] - 0s 3ms/step - loss: 0.4969 - accuracy: 0.7857 Test Loss: 0.4968760013580322

Test Accuracy: 0.7857142686843872

import pandas as pd

from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler from sklearn.pipeline import Pipeline

# Load the dataset

df = pd.read\_csv('BankChurn.csv')

# Split the data 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)

# Create a pipeline with a scaler and SVM classifier pipeline = Pipeline([

('scaler', StandardScaler()), ('svm', SVC())

])

# Define the parameter grid for hyperparameter tuning param\_grid = {

'svm C': [0.1, 1, 10, 100], # Regularization parameter 'svm kernel': ['linear', 'rbf', 'poly'], # Kernel type 'svm gamma': ['scale', 'auto'] # Kernel coefficient

}

# Choose the search strategy (GridSearchCV or RandomizedSearchCV)

# grid\_search = GridSearchCV(pipeline, param\_grid, cv=5, verbose=1)

grid\_search = RandomizedSearchCV(pipeline, param\_grid, n\_iter=10, cv=5, verbose=1)

# Perform hyperparameter tuning

grid\_search.fit(X\_train, y\_train)

# Print the best hyperparameters found

print("Best hyperparameters:", grid\_search.best\_params\_)

# Evaluate the model on the test set

test\_accuracy = grid\_search.score(X\_test, y\_test) print("Test Accuracy:", test\_accuracy)

Fitting 5 folds for each of 10 candidates, totalling 50 fits

Best hyperparameters: {'svm kernel': 'rbf', 'svm gamma': 'auto', 'svm C': 10} Test Accuracy: 0.9253731343283582

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.metrics import confusion\_matrix

# Make predictions on the test set

y\_pred = grid\_search.predict(X\_test)

# Compute confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

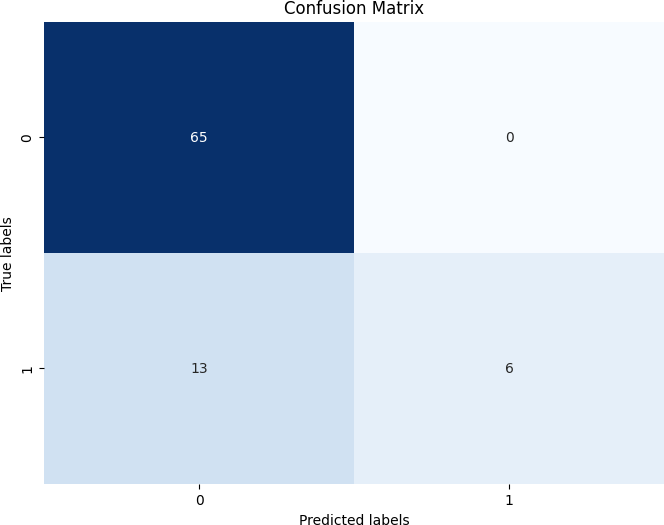
# Plot confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False) plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix') plt.show()



from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_mat

# Create a Support Vector Machine classifier with specified C parameter svm\_classifier = SVC(kernel='linear', C=1.0)

# Train the classifier on the training data svm\_classifier.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred\_svm = svm\_classifier.predict(X\_test)

# Evaluate the performance of the classifier

accuracy\_svm = accuracy\_score(y\_test, y\_pred\_svm)

precision\_svm = precision\_score(y\_test, y\_pred\_svm) recall\_svm = recall\_score(y\_test, y\_pred\_svm)

f1\_svm = f1\_score(y\_test, y\_pred\_svm)

confusion\_matrix\_svm = confusion\_matrix(y\_test, y\_pred\_svm)

print("Support Vector Machine Classifier Metrics:") print("Accuracy:", accuracy\_svm)

print("Precision:", precision\_svm) print("Recall:", recall\_svm)

print("F1 Score:", f1\_svm)

print("Confusion Matrix:\n", confusion\_matrix\_svm)

Support Vector Machine Classifier Metrics:

Accuracy: 0.5671641791044776

Precision: 0.5441176470588235

Recall: 0.578125

F1 Score: 0.5606060606060606

Confusion Matrix:

[[39 31]

[27 37]]

# Get the number of support vectors for each class n\_support\_vectors = svm\_classifier.n\_support\_

print("Number of support vectors for each class:", n\_support\_vectors)

Number of support vectors for each class: [135 137]

from sklearn.metrics import confusion\_matrix

# Calculate confusion matrix for the classifier

conf\_matrix = confusion\_matrix(y\_test, y\_pred\_svm)

# Extract true negatives (TN) and false positives (FP) tn = conf\_matrix[0, 0]

fp = conf\_matrix[0, 1]

# Calculate specificity

specificity = tn / (tn + fp)

print("Specificity:", specificity)

Specificity: 0.5571428571428572

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.metrics import confusion\_matrix

# Calculate confusion matrix for the SVM model

conf\_matrix\_svm = confusion\_matrix(y\_test, y\_pred\_svm)

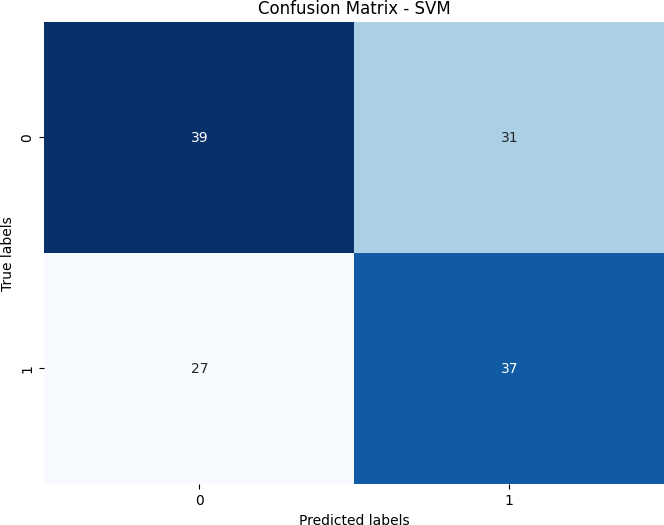
# Plot confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_svm, annot=True, fmt='d', cmap='Blues', cbar=False) plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix - SVM') plt.show()



from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, confusion\_mat

# Create a K-Nearest Neighbors classifier

knn\_classifier = KNeighborsClassifier(n\_neighbors=5)

# Train the classifier on the training data knn\_classifier.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred\_knn = knn\_classifier.predict(X\_test)

# Evaluate the performance of the classifier

accuracy\_knn = accuracy\_score(y\_test, y\_pred\_knn)

precision\_knn = precision\_score(y\_test, y\_pred\_knn) recall\_knn = recall\_score(y\_test, y\_pred\_knn)

f1\_knn = f1\_score(y\_test, y\_pred\_knn)

confusion\_matrix\_knn = confusion\_matrix(y\_test, y\_pred\_knn)

print("K-Nearest Neighbors Classifier Metrics:") print("Accuracy:", accuracy\_knn)

print("Precision:", precision\_knn) print("Recall:", recall\_knn)

print("F1 Score:", f1\_knn)

print("Confusion Matrix:\n", confusion\_matrix\_knn)

K-Nearest Neighbors Classifier Metrics: Accuracy: 0.7686567164179104

Precision: 0.7142857142857143

Recall: 0.859375

F1 Score: 0.7801418439716311

Confusion Matrix:

[[48 22]

[ 9 55]]

from sklearn.model\_selection import GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

# Define the parameter grid

param\_grid = {'n\_neighbors': [3, 5, 7, 9, 11]} # Values of k to try

# Create a K-Nearest Neighbors classifier knn\_classifier = KNeighborsClassifier()

# Perform grid search with cross-validation

grid\_search = GridSearchCV(knn\_classifier, param\_grid, cv=5, scoring='accuracy') grid\_search.fit(X\_train, y\_train)

# Get the best model and its accuracy

best\_model = grid\_search.best\_estimator\_ best\_accuracy = grid\_search.best\_score\_

print("Optimal KNN Model:", best\_model)

print("Accuracy of Optimal Model:", best\_accuracy)

Optimal KNN Model: KNeighborsClassifier(n\_neighbors=3)

Accuracy of Optimal Model: 0.7500173070266529

import matplotlib.pyplot as plt import numpy as np

# Extract the mean cross-validated scores and corresponding values of n\_neighbors mean\_scores = grid\_search.cv\_results\_['mean\_test\_score']

neighbors\_values = param\_grid['n\_neighbors']

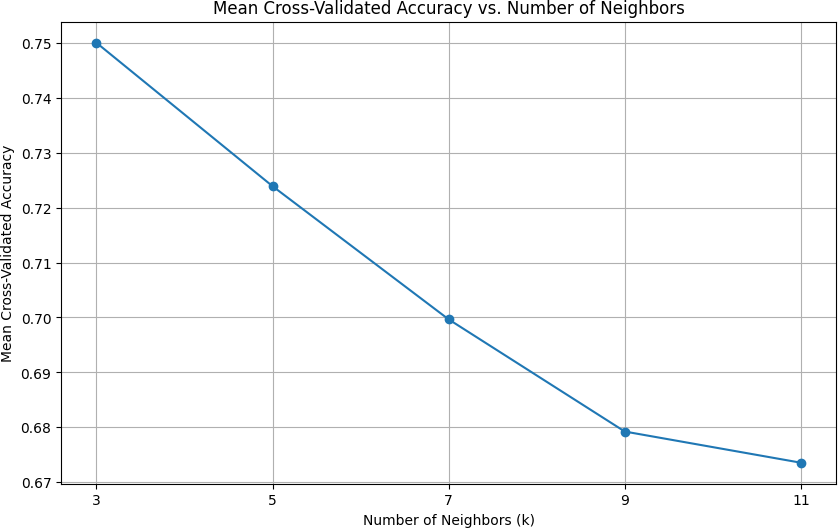
# Plot mean cross-validated accuracy vs. number of neighbors plt.figure(figsize=(10, 6))

plt.plot(neighbors\_values, mean\_scores, marker='o')

plt.title('Mean Cross-Validated Accuracy vs. Number of Neighbors') plt.xlabel('Number of Neighbors (k)')

plt.ylabel('Mean Cross-Validated Accuracy') plt.xticks(neighbors\_values)

plt.grid(True) plt.show()



from sklearn.metrics import confusion\_matrix

# Make predictions on the test data

y\_pred\_knn = best\_model.predict(X\_test)

# Calculate confusion matrix for the KNN model

conf\_matrix\_knn = confusion\_matrix(y\_test, y\_pred\_knn)

# Extract true negatives (TN) and false positives (FP) tn = conf\_matrix\_knn[0, 0]

fp = conf\_matrix\_knn[0, 1]

# Calculate specificity

specificity = tn / (tn + fp)

print("Specificity of KNN Model:", specificity)

Specificity of KNN Model: 0.6714285714285714

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.metrics import confusion\_matrix

# Calculate confusion matrix for the KNN model

conf\_matrix\_knn = confusion\_matrix(y\_test, y\_pred\_knn)

# Plot confusion matrix as a heatmap plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix\_knn, annot=True, fmt='d', cmap='Blues', cbar=False) plt.xlabel('Predicted labels')

plt.ylabel('True labels')

plt.title('Confusion Matrix - KNN') plt.show()

