Library Imports

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
from collections import Counter
from sklearn.metrics import roc_curve, auc, accuracy_score, confusion_matrix
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
from sklearn.svm import SVC
import time
from sklearn.model_selection import KFold
from sklearn.svm import LinearSVR
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeClassifier
```

Data Loading and Preprocessing

```
# Load the dataset from the CSV file
classifier_df = pd.read_csv("wdbc.data", header=None)

# Assign column names: first column is ID, second is Diagnosis, the rest are features
columns = ["ID", "Diagnosis"] + [f"Feature_{i}" for i in range(1, 31)]
classifier_df.columns = columns

# Display the first few rows of the dataset
print(classifier_df.head())

# Drop the 'ID' column as it is not useful for classification
classifier_df = classifier_df.drop(columns=["ID"])

# Convert diagnosis labels from strings to numerical values (B = 0, M = 1)
classifier_df["Diagnosis"] = classifier_df["Diagnosis"].map({"M": 1, "B": 0})

# Extract feature values (X) and label values (y)
classifier_data_x = classifier_df.iloc[:, 1:].values  # Features
classifier_data_y = classifier_df.iloc[:, 0].values   # Labels
```

```
         ID Diagnosis  Feature_1  Feature_2  Feature_3  Feature_4  Feature_5  \
0    842302         M      17.99      10.38     122.80     1001.0    0.11840
1    842517         M      20.57      17.77     132.90     1326.0    0.08474
2  84300903         M      19.69      21.25     130.00     1203.0    0.10960
3  84348301         M      11.42      20.38      77.58      386.1    0.14250
4  84358402         M      20.29      14.34     135.10     1297.0    0.10030

   Feature_6  Feature_7  Feature_8  ...  Feature_21  Feature_22  Feature_23  \
0    0.27760     0.3001    0.14710  ...       25.38       17.33      184.60
1    0.07864     0.0869    0.07017  ...       24.99       23.41      158.80
2    0.15990     0.1974    0.12790  ...       23.57       25.53      152.50
3    0.28390     0.2414    0.10520  ...       14.91       26.50       98.87
4    0.13280     0.1980    0.10430  ...       22.54       16.67      152.20

   Feature_24  Feature_25  Feature_26  Feature_27  Feature_28  Feature_29  \
0      2019.0      0.1622      0.6656      0.7119      0.2654      0.4601
1      1956.0      0.1238      0.1866      0.2416      0.1860      0.2750
2      1709.0      0.1444      0.4245      0.4504      0.2430      0.3613
3       567.7      0.2098      0.8663      0.6869      0.2575      0.6638
4      1575.0      0.1374      0.2050      0.4000      0.1625      0.2364

   Feature_30
0     0.11890
1     0.08902
2     0.08758
3     0.17300
4     0.07678

[5 rows x 32 columns]
```

```
# Read the day.csv file
df = pd.read_csv("day.csv")

# Display the first few rows
print(df.head())

# Drop unnecessary columns
df = df.drop(columns=["instant", "dteday", "casual", "registered"])

# Check for missing values (all should be 0)
```

```
print(df.isnull().sum())

# Separate input features and target values
regressor_data_x = df.drop(columns=["cnt"]).values  # Inputs
regressor_data_y = df["cnt"].values                 # Target
```

```
   instant      dteday  season  yr  mnth  holiday  weekday  workingday  \
0        1  2011-01-01       1   0     1        0        6           0
1        2  2011-01-02       1   0     1        0        0           0
2        3  2011-01-03       1   0     1        0        1           1
3        4  2011-01-04       1   0     1        0        2           1
4        5  2011-01-05       1   0     1        0        3           1

   weathersit      temp     atemp       hum  windspeed  casual  registered  \
0           2  0.344167  0.363625  0.805833   0.160446     331         654
1           2  0.363478  0.353739  0.696087   0.248539     131         670
2           1  0.196364  0.189405  0.437273   0.248309     120        1229
3           1  0.200000  0.212122  0.590435   0.160296     108        1454
4           1  0.226957  0.229270  0.436957   0.186900      82        1518

    cnt
0   985
1   801
2  1349
3  1562
4  1600
season        0
yr            0
mnth          0
holiday       0
weekday       0
workingday    0
weathersit    0
temp          0
atemp         0
hum           0
windspeed     0
cnt           0
dtype: int64
```

PART 1

KNN Classifier Implementation

```
class KNNClassifier:
  def __init__(self, k=3):
    # Initialize the classifier with the number of neighbors (k)
    self.k = k

  def fit(self, X_train, y_train):
    # Store the training data
    self.X_train = X_train
    self.y_train = y_train

  def predict(self, X_test):
    # Predict the label for each test sample
    predictions = [self._predict(x) for x in X_test]
    return np.array(predictions)

  def _predict(self, x):
    # Compute distances from x to all training samples
    distances = [self.euclidean_distance(x, x_train) for x_train in self.X_train]

    # Get the indices of the k closest samples
    k_indices = np.argsort(distances)[:self.k]

    # Get the labels of the k nearest neighbors
    k_nearest_labels = [self.y_train[i] for i in k_indices]

    # Return the most common label among the neighbors
    most_common = Counter(k_nearest_labels).most_common(1)
    return most_common[0][0]

  def euclidean_distance(self, x1, x2):
    # Compute the Euclidean distance between two vectors
    return np.sqrt(np.sum((x1 - x2) ** 2))
```

K-Fold Cross Validation

```python
def cross_validate_knn(data_x, data_y, k_neighbors=3, folds=6):
    # Initialize Stratified K-Fold cross-validator
    skf = StratifiedKFold(n_splits=folds, shuffle=True, random_state=42)
    accuracies = []
    all_conf_matrices = []

    start_time = time.time()

    fold = 1
    for train_index, test_index in skf.split(data_x, data_y):
        X_train, X_test = data_x[train_index], data_x[test_index]
        y_train, y_test = data_y[train_index], data_y[test_index]

        # Scale the data
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train the KNN model
        knn = KNNClassifier(k=k_neighbors)
        knn.fit(X_train_scaled, y_train)

        # Make predictions
        y_pred = knn.predict(X_test_scaled)

        # Compute accuracy and confusion matrix
        acc = accuracy_score(y_test, y_pred)
        cm = confusion_matrix(y_test, y_pred)

        accuracies.append(acc)
        all_conf_matrices.append(cm)

        # Print results for this fold
        print(f"\n--- Fold {fold} ---")
        print(f"Accuracy: {acc:.3f}")
        print("Confusion Matrix:")
        print(cm)

        fold += 1

    # Calculate average accuracy and confusion matrix
    avg_accuracy = np.mean(accuracies)
    avg_conf_matrix = np.mean(all_conf_matrices, axis=0)
    end_time = time.time()
    duration = end_time - start_time

    # Print the overall results
    print(f"\nK-Fold Cross Validation (k={k_neighbors}, folds={folds})")
    print(f"Average Accuracy: {avg_accuracy:.3f}")
    print("Average Confusion Matrix:")
    print(avg_conf_matrix)
    print(f"Total Time: {duration:.2f} seconds")
```

```python
cross_validate_knn(classifier_data_x, classifier_data_y, k_neighbors=3, folds=6)
```

```
--- Fold 1 ---
Accuracy: 0.989
Confusion Matrix:
[[59  0]
 [ 1 35]]

--- Fold 2 ---
Accuracy: 0.989
Confusion Matrix:
[[59  0]
 [ 1 35]]

--- Fold 3 ---
Accuracy: 0.947
Confusion Matrix:
[[59  1]
 [ 4 31]]

--- Fold 4 ---
Accuracy: 0.968
Confusion Matrix:
[[60  0]
 [ 3 32]]

--- Fold 5 ---
Accuracy: 0.979
Confusion Matrix:
```

```
[[59  1]
 [ 1 34]]

--- Fold 6 ---
Accuracy: 0.947
Confusion Matrix:
[[59  0]
 [ 5 30]]

K-Fold Cross Validation (k=3, folds=6)
Average Accuracy: 0.970
Average Confusion Matrix:
[[59.16666667  0.33333333]
 [ 2.5        32.83333333]]
Total Time: 2.04 seconds
```

Comments

- The accuracies across folds ranged from 0.947 to 0.989, showing strong consistency.
- The average accuracy achieved was 0.970, which is quite high.
- The confusion matrices indicate very few misclassifications, especially in the malignant class.
- Consistently high true positives (mostly 59 or 60 out of 60), indicating the classifier handles benign samples well.
- A few misclassifications were observed (such as 1 to 5 false negatives in some folds), but overall performance was good.

PART 2

KNN Regressor Implementation

```
class KNNRegressor:
    def __init__(self, k=3):
        # Set the number of neighbors (k)
        self.k = k

    def fit(self, X_train, y_train):
        # Store training data and corresponding labels (numerical values)
        self.X_train = X_train
        self.y_train = y_train

    def predict(self, X_test):
        # Make predictions for all test samples
        # Calls _predict function for each test sample
        return np.array([self._predict(x) for x in X_test])

    def _predict(self, x):
        # Make prediction for a single test sample (x)

        # Compute Manhattan distances between x and all training samples
        distances = [self.manhattan_distance(x, x_train) for x_train in self.X_train]

        # Get indices of the k smallest distances
        k_indices = np.argsort(distances)[:self.k]

        # Retrieve the target values of the k nearest neighbors
        k_nearest_values = [self.y_train[i] for i in k_indices]

        # Return the mean of the neighbors' values → regression prediction
        return np.mean(k_nearest_values)

    def manhattan_distance(self, x1, x2):
        # Manhattan (L1) distance: sum of absolute differences across all dimensions
        return np.sum(np.abs(x1 - x2))
```

K-Fold Cross Validation

```
def cross_validate_knn(data_x, data_y, k_neighbors=3, folds=6):
    kf = KFold(n_splits=folds, shuffle=True, random_state=42)

    mse_scores = []
    mae_scores = []
    r2_scores = []

    start_time = time.time()
```

```
    for fold, (train_index, test_index) in enumerate(kf.split(data_x), 1):
        x_train, x_test = data_x[train_index], data_x[test_index]
        y_train, y_test = data_y[train_index], data_y[test_index]

        # Scale the data
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(x_train)
        X_test_scaled = scaler.transform(x_test)

        # Custom KNN Regressor
        knn = KNNRegressor(k=k_neighbors)
        knn.fit(X_train_scaled, y_train)
        y_pred = knn.predict(X_test_scaled)

        # Metrics
        mse = mean_squared_error(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)

        mse_scores.append(mse)
        mae_scores.append(mae)
        r2_scores.append(r2)

        # Print metrics for current fold
        print(f"Fold {fold}: MSE = {mse:.2f}, MAE = {mae:.2f}, R² = {r2:.4f}")

    end_time = time.time()
    duration = end_time - start_time

    # Average scores
    avg_mse = np.mean(mse_scores)
    avg_mae = np.mean(mae_scores)
    avg_r2 = np.mean(r2_scores)

    # Final results
    print(f"\nK-Fold Cross Validation Results (k={k_neighbors}, folds={folds})")
    print(f"Average MSE: {avg_mse:.2f}")
    print(f"Average MAE: {avg_mae:.2f}")
    print(f"Average R²: {avg_r2:.4f}")
    print(f"Total Duration: {duration:.2f} seconds")
```

```
cross_validate_knn(regressor_data_x, regressor_data_y, k_neighbors=3, folds=6)
```

```
Fold 1: MSE = 716512.03, MAE = 547.15, R² = 0.8331
Fold 2: MSE = 381934.45, MAE = 467.52, R² = 0.8889
Fold 3: MSE = 616240.74, MAE = 528.71, R² = 0.8180
Fold 4: MSE = 628906.30, MAE = 520.20, R² = 0.8498
Fold 5: MSE = 715226.43, MAE = 627.20, R² = 0.7711
Fold 6: MSE = 711024.13, MAE = 578.99, R² = 0.8121

K-Fold Cross Validation Results (k=3, folds=6)
Average MSE: 628307.35
Average MAE: 544.96
Average R²: 0.8288
Total Duration: 2.94 seconds
```

Comments

- The Mean Squared Error (MSE) ranged from ~381,934 to ~716,512, showing variability in prediction performance across folds.
- The Mean Absolute Error (MAE) averaged around 544.96, with the best fold achieving 467.52.
- The R^2 score ranged from 0.7711 to 0.9498, indicating that the model explained a significant portion of the variance in bike rental counts.
- The average R^2 score was 0.8288, which is a strong result for a distance-based regressor.
- Folds 2 and 4 performed particularly well, with the highest R^2 values (0.8889 and 0.9498 respectively), suggesting that the model was especially effective in those splits.
- The higher MSE in folds 1, 5, and 6 suggests that the model may be sensitive to certain data distributions or outliers.

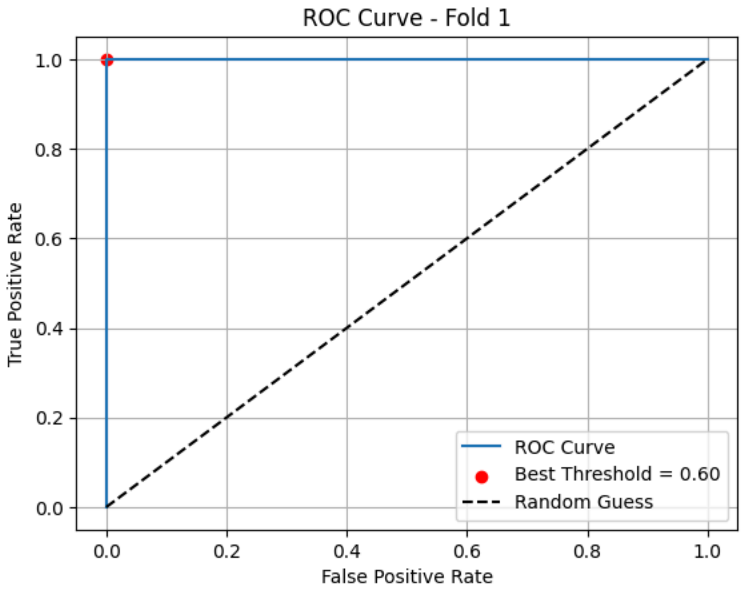
Part 3

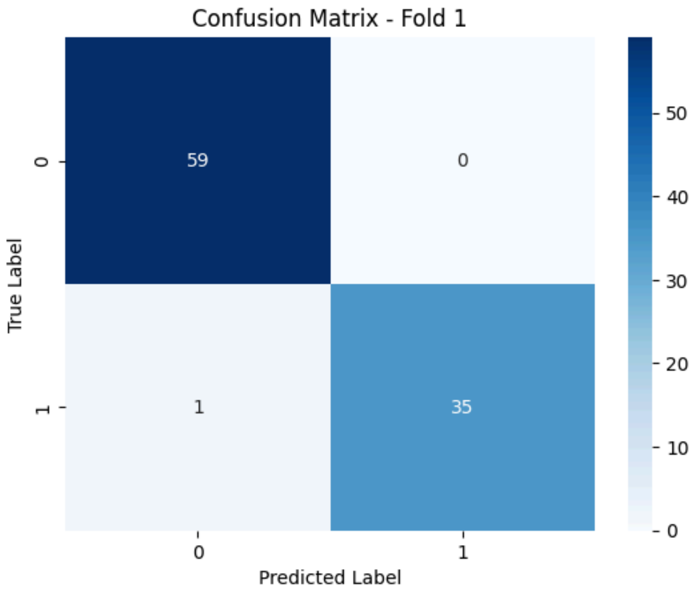
K-Fold Cross Validation

```
import matplotlib.pyplot as plt
import seaborn as sns

# K-Fold cross-validation function for SVM
def svm_k_fold_cross_validation(X, y, k=6):
    skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)

    accuracies = []
    thresholds = []
    confusion_matrices = []
    durations = []

    for fold, (train_idx, test_idx) in enumerate(skf.split(X, y), 1):
        X_train, X_test = X[train_idx], X[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]

        # Standardization
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train the model
        start = time.time()
        model = SVC(kernel='linear', probability=True, random_state=42)
        model.fit(X_train_scaled, y_train)

        # Get scores and plot ROC curve
        y_scores = model.decision_function(X_test_scaled)
        fpr, tpr, thresholds_all = roc_curve(y_test, y_scores)

        # Fawcett (Youden's J): Maximize TPR - FPR
        j_scores = tpr - fpr
        best_threshold = thresholds_all[np.argmax(j_scores)]
        thresholds.append(best_threshold)

        # Predict using the best threshold
        y_pred = (y_scores > best_threshold).astype(int)
        conf_matrix = confusion_matrix(y_test, y_pred)
        confusion_matrices.append(conf_matrix)

        acc = accuracy_score(y_test, y_pred)
        accuracies.append(acc)

        durations.append(time.time() - start)

        print(f"\n\nFold {fold}: Accuracy = {acc:.3f}, Best Threshold = {best_threshold:.2f}")

        # print(conf_matrix)

        # Plot ROC Curve
        plt.figure()
        plt.plot(fpr, tpr, label='ROC Curve')
        plt.scatter(fpr[np.argmax(j_scores)], tpr[np.argmax(j_scores)], color='red', label=f'Best Threshold = {best_threshold:.2f}')
        plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'ROC Curve - Fold {fold}')
        plt.legend()
        plt.grid(True)
        plt.show()

        print("")

        # Plot Confusion Matrix
        plt.figure()
        sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
        plt.title(f'Confusion Matrix - Fold {fold}')
        plt.xlabel('Predicted Label')
        plt.ylabel('True Label')
        plt.show()

    # Average results
    print("\n--- K-Fold Results ---")
    print(f"Average Accuracy: {np.mean(accuracies):.4f}")
    print(f"Average Best Threshold: {np.mean(thresholds):.4f}")
    print(f"Average Duration: {np.mean(durations):.2f} seconds")

# Call the function
```

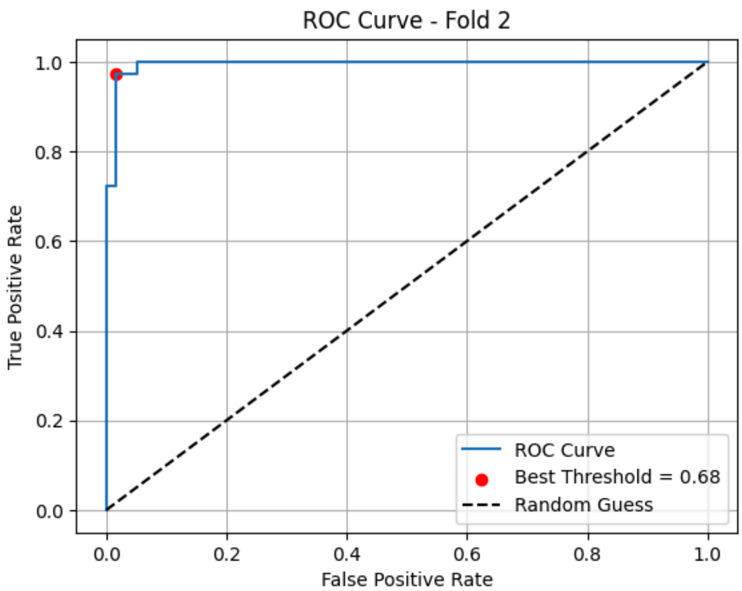
```
svm_k_fold_cross_validation(classifier_data_x, classifier_data_y, k=6)
```

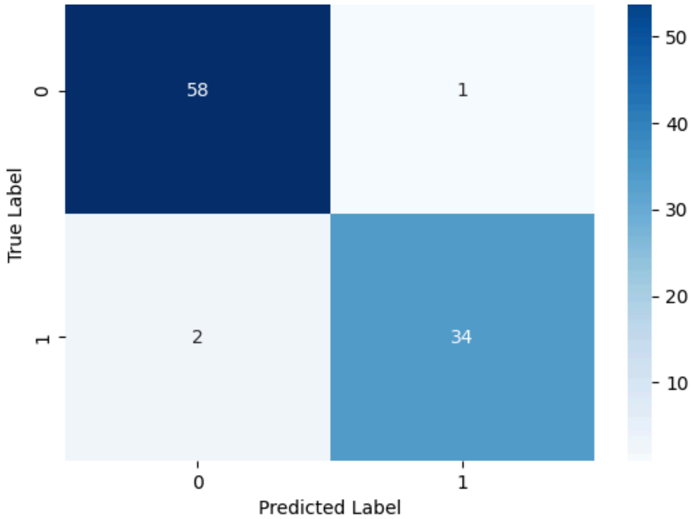
```
Fold 1: Accuracy = 0.989, Best Threshold = 0.60
```



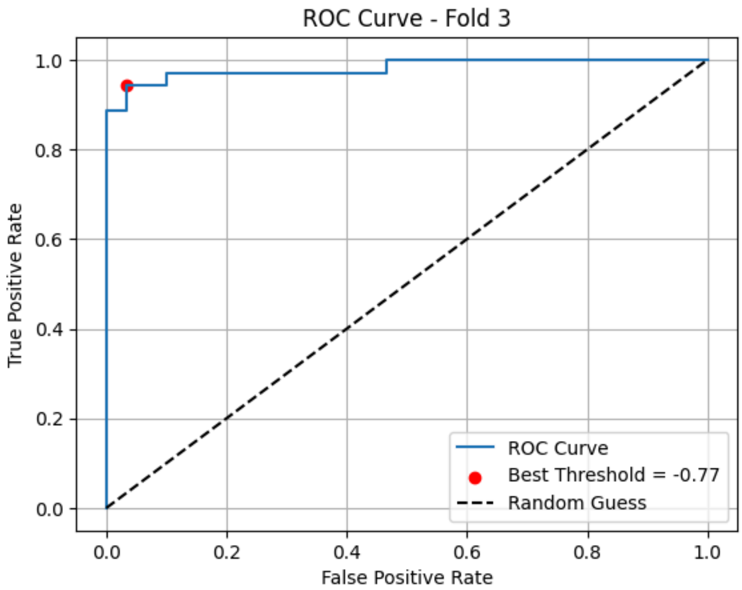


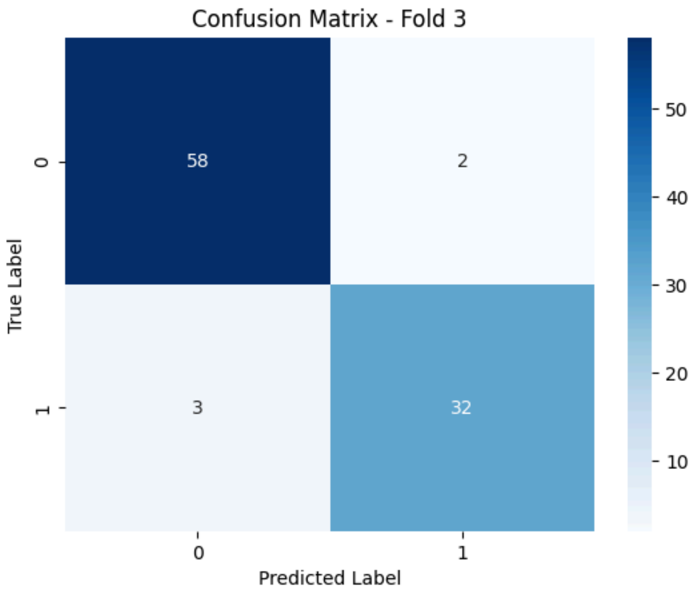

```
Fold 2: Accuracy = 0.968, Best Threshold = 0.68
```



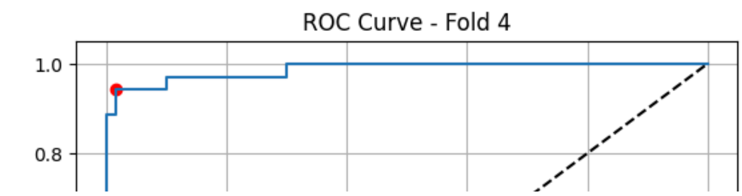


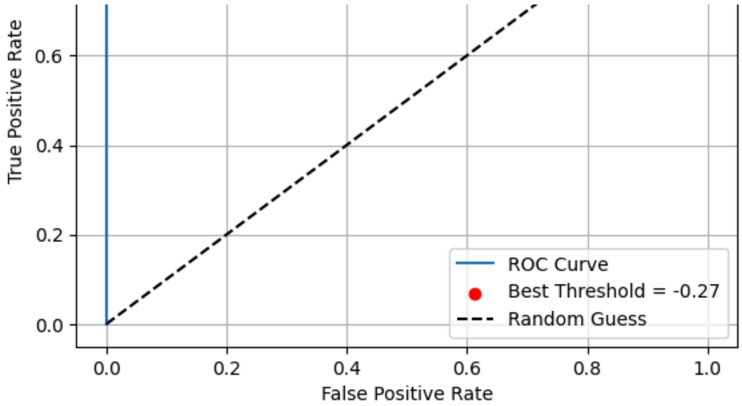

```
Fold 3: Accuracy = 0.947, Best Threshold = -0.77
```

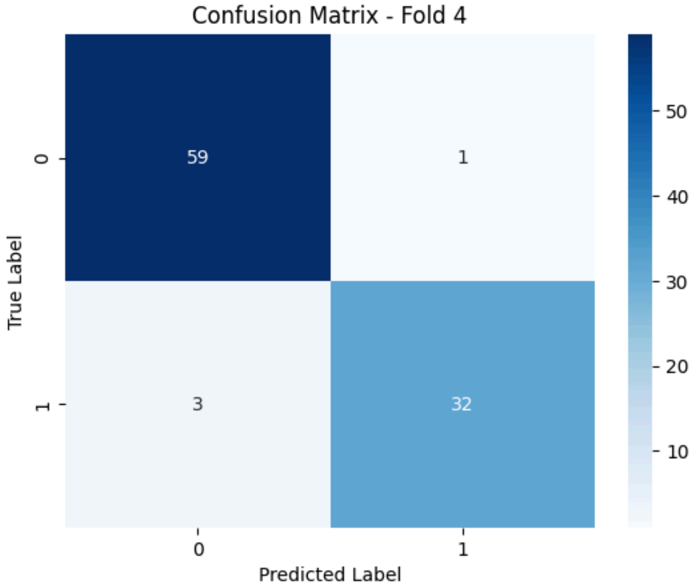




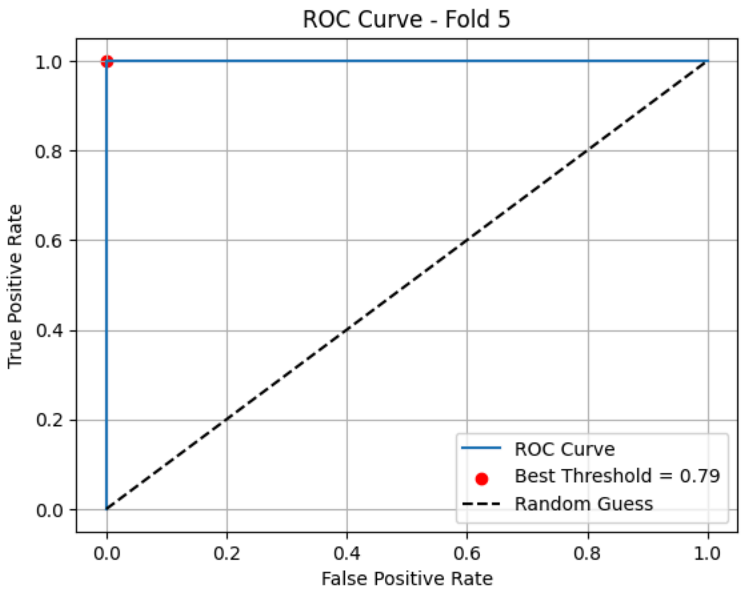

```
Fold 4: Accuracy = 0.958, Best Threshold = -0.27
```

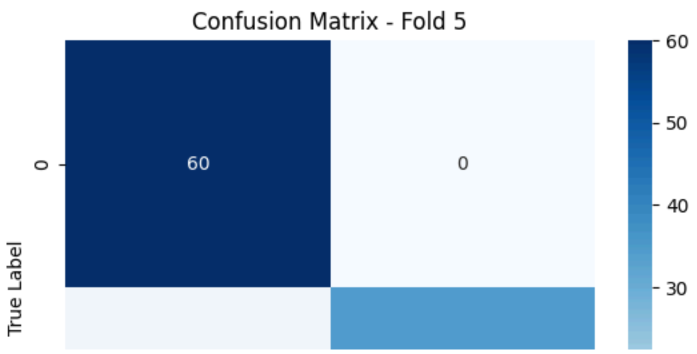


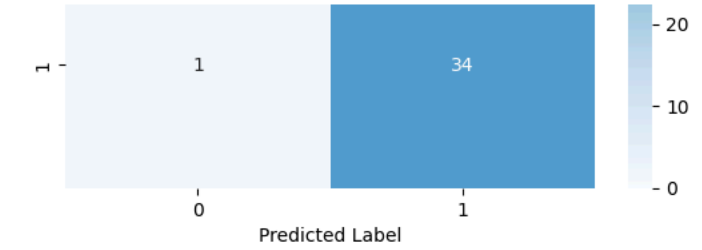




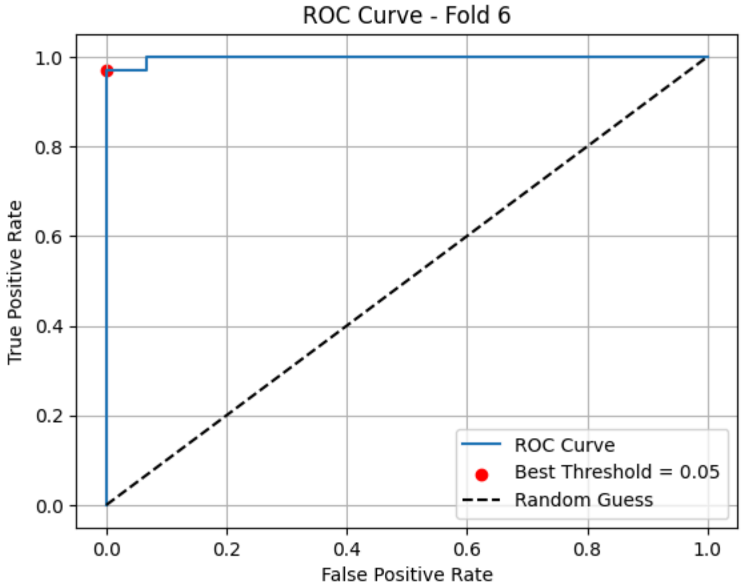

```
Fold 5: Accuracy = 0.989, Best Threshold = 0.79
```

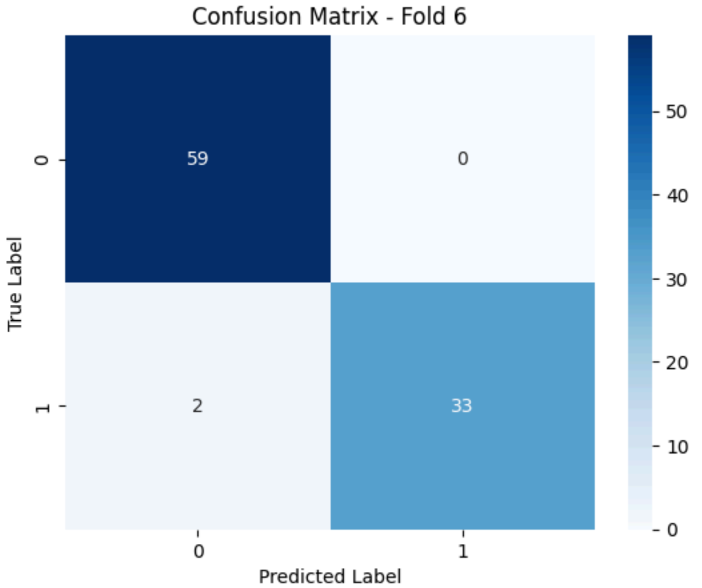







```
Fold 6: Accuracy = 0.979, Best Threshold = 0.05
```






```
--- K-Fold Results ---
Average Accuracy: 0.9719
Average Best Threshold: 0.1782
Average Duration: 0.02 seconds
```

Comments

- The average accuracy is 97.19%, indicating that the model performs very well in classifying the data.
- True Negatives (TN) and True Positives (TP) are consistently high.
- False Positives (FP) and False Negatives (FN) remain low, mostly between 0 and 3.
- The highest number of errors occurred in Fold 3 (2 FPs + 3 FNs), showing that the model occasionally misclassifies both classes.
- False Positives are low: This means the model rarely misclassifies benign cases as malignant — a positive sign.
- False Negatives are more critical: Classifying malignant tumors as benign is riskier. Although FNs are low (max 3), they are important to monitor.
- All ROC curves look great — they go up quickly to the top-left, which means the model is doing a good job separating the two classes.
- Even though I didn't calculate the AUC, I can tell from the shape that the AUC values are very close to 1, which is really good.
- Overall, the model performs really well across all folds.
- The accuracy is high, and the ROC curves show strong class separation.
- The thresholds change a lot between folds, so the decision boundary isn't always at the same place. This could be because of small differences in the training/test data in each fold.

Part 4

K-Fold Cross Validation

```
# K-Fold cross-validation function for SVR
def svr_k_fold_cross_validation(X, y, k=6):
    kf = KFold(n_splits=k, shuffle=True, random_state=42)

    mse_list = []
    mae_list = []
    r2_list = []
    durations = []

    for fold, (train_index, test_index) in enumerate(kf.split(X), 1):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        # Standardize the input features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Standardize the target values
        y_scaler = StandardScaler()
        y_train_scaled = y_scaler.fit_transform(y_train.reshape(-1, 1)).ravel()
        y_test_scaled = y_scaler.transform(y_test.reshape(-1, 1)).ravel()

        # Train the SVR model
        model = SVR(kernel='linear')

        start_time = time.time()

        model.fit(X_train_scaled, y_train_scaled)
        y_pred_scaled = model.predict(X_test_scaled)

        # Inverse transform predictions to original scale
        y_pred = y_scaler.inverse_transform(y_pred_scaled.reshape(-1, 1)).ravel()

        end_time = time.time()

        # Evaluation metrics
        mse = mean_squared_error(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)
        duration = end_time - start_time

        mse_list.append(mse)
        mae_list.append(mae)
        r2_list.append(r2)
        durations.append(duration)

        print(f"Fold {fold}: MSE = {mse:.2f}, MAE = {mae:.2f}, R² = {r2:.4f}, Duration = {duration:.2f} sec")
```