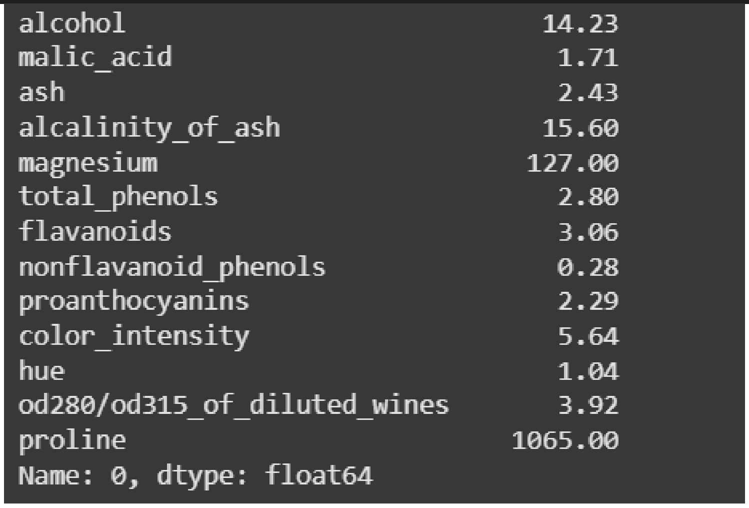
Homework coding 1

	Student first and last name	Course number and section
1.	Narindra Balkissoon	CSC 547 UT 2
2.	Chang Wang	CSC 547 UT 2
3.	Quinn Sturm	CSC 547 UT 2
4.	Shuyun Shen	CSC 447 UT 1

PART 1. Wine data classification using SVM

```
# Import necessary libraries
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc_curve, roc_auc_score
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
import seaborn as sns
# Load the Wine dataset
data = load_wine()
X = data.data  # Features (13 attributes)
y = data.target  # Target classes (3 classes)
import pandas as pd
# Create a DataFrame with the feature data
df = pd.DataFrame(data=data.data, columns=data.feature_names)
# Display one example
print(df.iloc[0])
```



```
alcohol                           14.23
malic_acid                         1.71
ash                                2.43
alcalinity_of_ash                 15.60
magnesium                        127.00
total_phenols                      2.80
flavanoids                         3.06
nonflavanoid_phenols               0.28
proanthocyanins                    2.29
color_intensity                    5.64
hue                                1.04
od280/od315_of_diluted_wines       3.92
proline                         1065.00
Name: 0, dtype: float64
```

Figure 1 represents 1 example in the wine data set

```
# data split 70/30
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Initialize the SVM classifier with custom C and gamma values
svm = SVC(
    C=15,                        # Custom regularization parameter
    kernel='linear',
    gamma=0.0015,                    # Custom gamma value
    probability=True,             # Enable probability estimates for ROC
    random_state=42,  # Random state for reproducibility
     decision_function_shape='ovr'
)
# Train the model
svm.fit(X_train, y_train)
# Make predictions on the test data
y_pred = svm.predict(X_test)
y_prob = svm.predict_proba(X_test)[:, 1]
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of SVM classifier: {accuracy * 100:.2f}%")
# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix using Seaborn
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=data.target_names, yticklabels=data.target_names)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

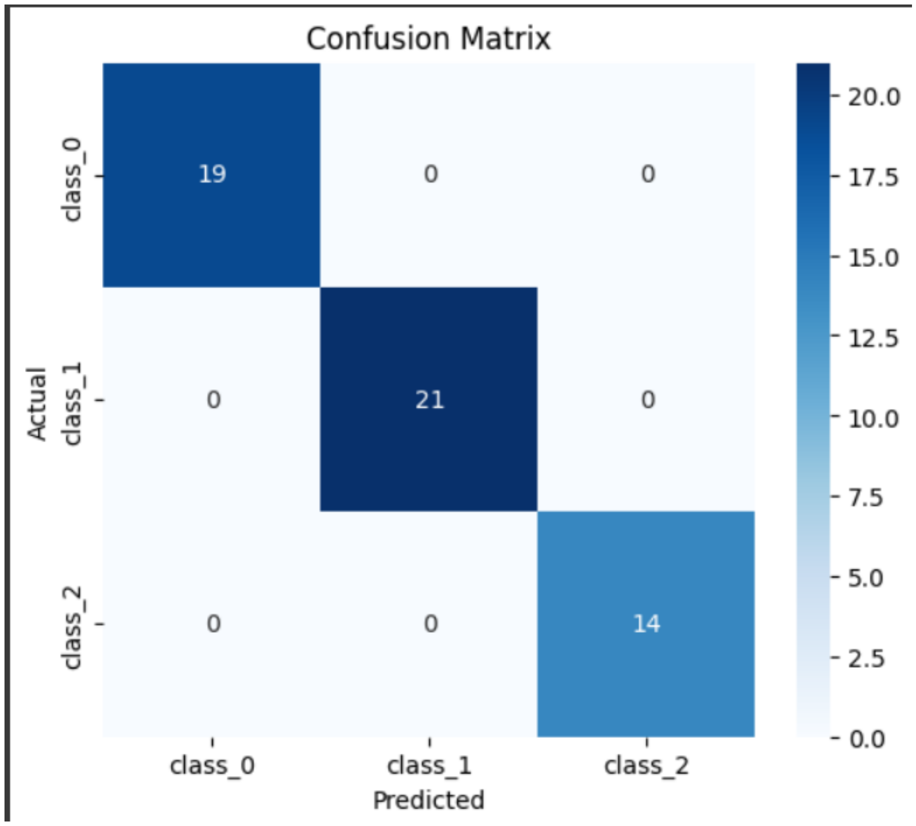


Figure 2 represents the confusion matrix for the wine classification

```
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
from sklearn.preprocessing import label_binarize
import numpy as np
y_binarized = label_binarize(y, classes=np.unique(y))
n_classes = y_binarized.shape[1]

# Compute ROC curve and ROC AUC for each class
y_test_binarized = label_binarize(y_test, classes=np.unique(y)) # Use y_test instead of y
y_score = svm.decision_function(X_test) # Get decision function scores for multi-class ROC
plt.figure(figsize=(8, 6))
colors = ['blue', 'green', 'red']
for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_test_binarized[:, i], y_score[:, i]) # Use y_test_binarized and y_score
    auc = roc_auc_score(y_test_binarized[:, i], y_score[:, i])
    plt.plot(fpr, tpr, color=colors[i], label=f'Class {data.target_names[i]} (AUC = {auc:.2f})')



# Plot random classifier line
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.title("Receiver Operating Characteristic (ROC) Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Recall)")
plt.legend(loc='lower right')
plt.show()
```

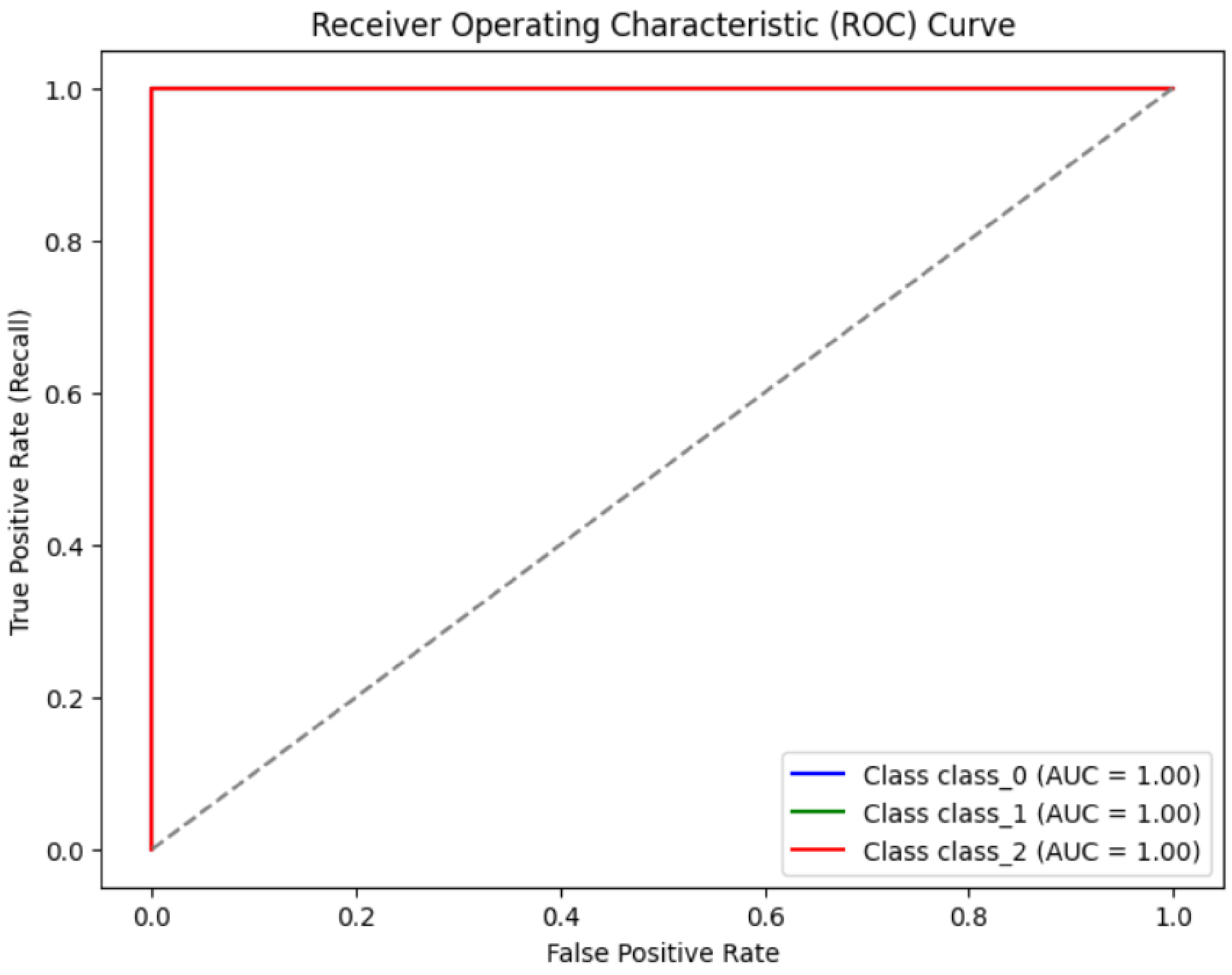
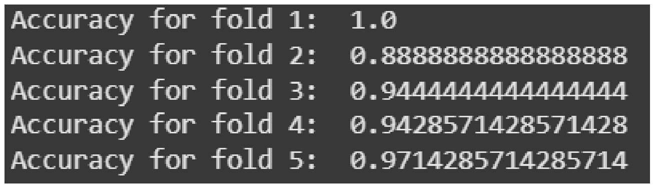


Figure 3 shows the ROC-AUC curve which is perfect for each class

```
from sklearn.model_selection import KFold
# Initialize KFold for 5-fold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Store the trained models and their corresponding test set indices
trained_models = []
test_indices = []
accuracies = []  # Store accuracies for each fold
# Perform 5-fold cross-validation manually
```

```
fold_num = 1
for train_index, test_index in kf.split(X):
    # Split the data into training and testing sets
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Train the model on the training data
    svm.fit(X_train, y_train)

    # Store the trained model and the test set index
    trained_models.append(svm)
    test_indices.append(test_index)

    # Test the model on the test data and print the score
    y_pred = svm.predict(X_test)
    accuracy=accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)  # Store accuracy
    print(f"Accuracy for fold {fold_num}: ", accuracy)
    fold_num += 1
```



```
Accuracy for fold 1:  1.0
Accuracy for fold 2:  0.8888888888888888
Accuracy for fold 3:  0.9444444444444444
Accuracy for fold 4:  0.9428571428571428
Accuracy for fold 5:  0.9714285714285714
```

Figure 4 shows the accuracy for each fold

```
# Calculate and print the average accuracy
#print(f"\nAverage Accuracy: {(accuracies)}")
average_accuracy = np.mean(accuracies)
print(f"\nAverage Accuracy: {average_accuracy}")
```

Average Accuracy: 0.9495238095238095

google colab (SVM):

https://colab.research.google.com/drive/13CpGOBNNZREKOzGLnD6i3THWsGp5UJ9U?usp=sharing

PART 2. Predict Diabetes progression using KNN regression

I. Data loading and Training

```python
# Import necessary libraries
from sklearn.datasets import load_diabetes
from sklearn.model_selection import train_test_split, KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score,
accuracy_score, make_scorer
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pandas as pd
from sklearn.model_selection import KFold


# Load the diabetes dataset
data = load_diabetes()

# Split the dataset into features (X) and target (y)
X = data.data
y = data.target

# Create a Pandas DataFrame from the dataset
df = pd.DataFrame(data=data.data, columns=data.feature_names)

# Add the target variable (diabetes progression)
df['target'] = data.target

# Display the first row of the DataFrame to understand its structure
print(df.iloc[0])
print(df.describe().T)
```

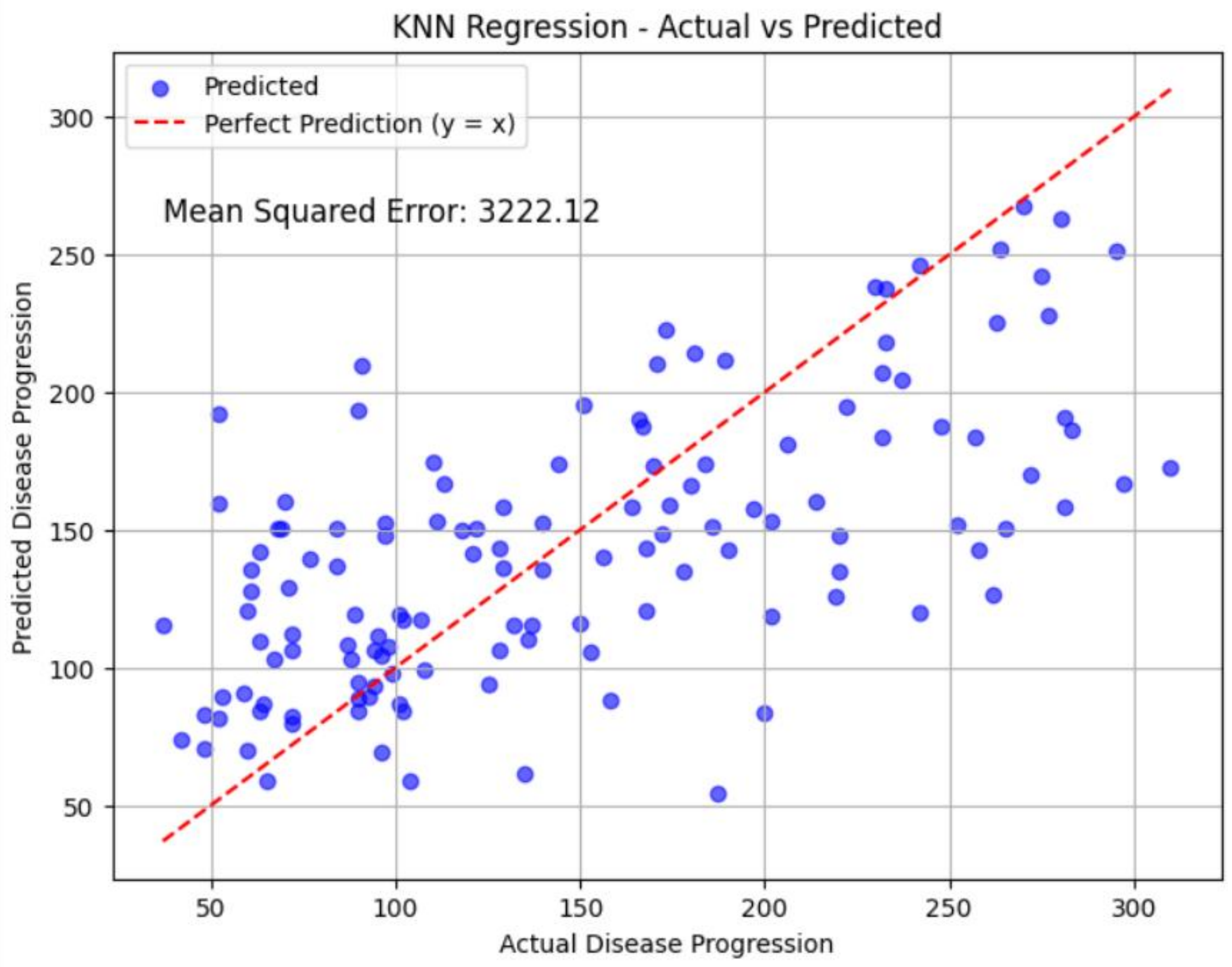
```
age          0.038076
sex          0.050680
bmi          0.061696
bp           0.021872
s1          -0.044223
s2          -0.034821
s3          -0.043401
s4          -0.002592
s5           0.019907
s6          -0.017646
target     151.000000
Name: 0, dtype: float64
        count          mean        std         min        25%         50%  \
age     442.0 -2.511817e-19   0.047619  -0.107226  -0.037299    0.005383
sex     442.0  1.230790e-17   0.047619  -0.044642  -0.044642   -0.044642
bmi     442.0 -2.245564e-16   0.047619  -0.090275  -0.034229   -0.007284
bp      442.0 -4.797570e-17   0.047619  -0.112399  -0.036656   -0.005670
s1      442.0 -1.381499e-17   0.047619  -0.126781  -0.034248   -0.004321
s2      442.0  3.918434e-17   0.047619  -0.115613  -0.030358   -0.003819
s3      442.0 -5.777179e-18   0.047619  -0.102307  -0.035117   -0.006584
s4      442.0 -9.042540e-18   0.047619  -0.076395  -0.039493   -0.002592
s5      442.0  9.293722e-17   0.047619  -0.126097  -0.033246   -0.001947
s6      442.0  1.130318e-17   0.047619  -0.137767  -0.033179   -0.001078
target  442.0  1.521335e+02  77.093005  25.000000  87.000000  140.500000

               75%         max
age       0.038076    0.110727
sex       0.050680    0.050680
bmi       0.031248    0.170555
bp        0.035644    0.132044
s1        0.028358    0.153914
s2        0.029844    0.198788
s3        0.029312    0.181179
s4        0.034309    0.185234
s5        0.032432    0.133597
s6        0.027917    0.135612
target  211.500000  346.000000
```

```python
# Split the dataset into training and test data (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Standardize the data to improve KNN performance
#scaler = StandardScaler()
#X_train_scaled = scaler.fit_transform(X_train)
#X_test_scaled = scaler.transform(X_test)
# Initialize KNN Regressor with K=5
```

```python
knn = KNeighborsRegressor(n_neighbors=5)

# Train the model
knn.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn.predict(X_test)
#print(y_pred)

# Evaluate model performance using Mean Squared Error (MSE) and RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print(f"Mean Squared Error: {mse}")
print(f"Root Mean Squared Error: {rmse}")
```

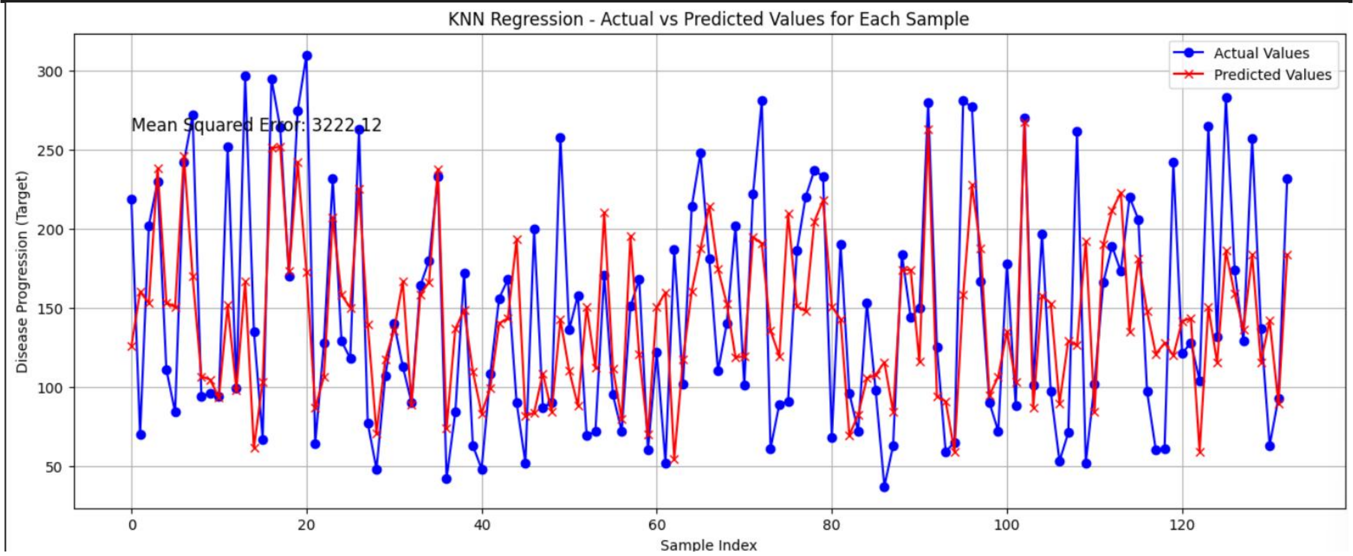
```
Mean Squared Error: 3222.117894736842
Root Mean Squared Error: 56.763702264183244
```

```python
# Plot predictions vs actual values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, color='blue', alpha=0.6, label='Predicted')

# Plot perfect prediction line (y = x)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red',
linestyle='--', label="Perfect Prediction (y = x)")

# Set labels and title
plt.xlabel("Actual Disease Progression")
plt.ylabel("Predicted Disease Progression")
plt.title("KNN Regression - Actual vs Predicted")

# Display Mean Squared Error on the plot
plt.text(min(y_test), max(y_pred) - 5, f'Mean Squared Error: {mse:.2f}',
color='black', fontsize=12)

# Add legend and grid
plt.legend()
plt.grid(True)
plt.show()
```

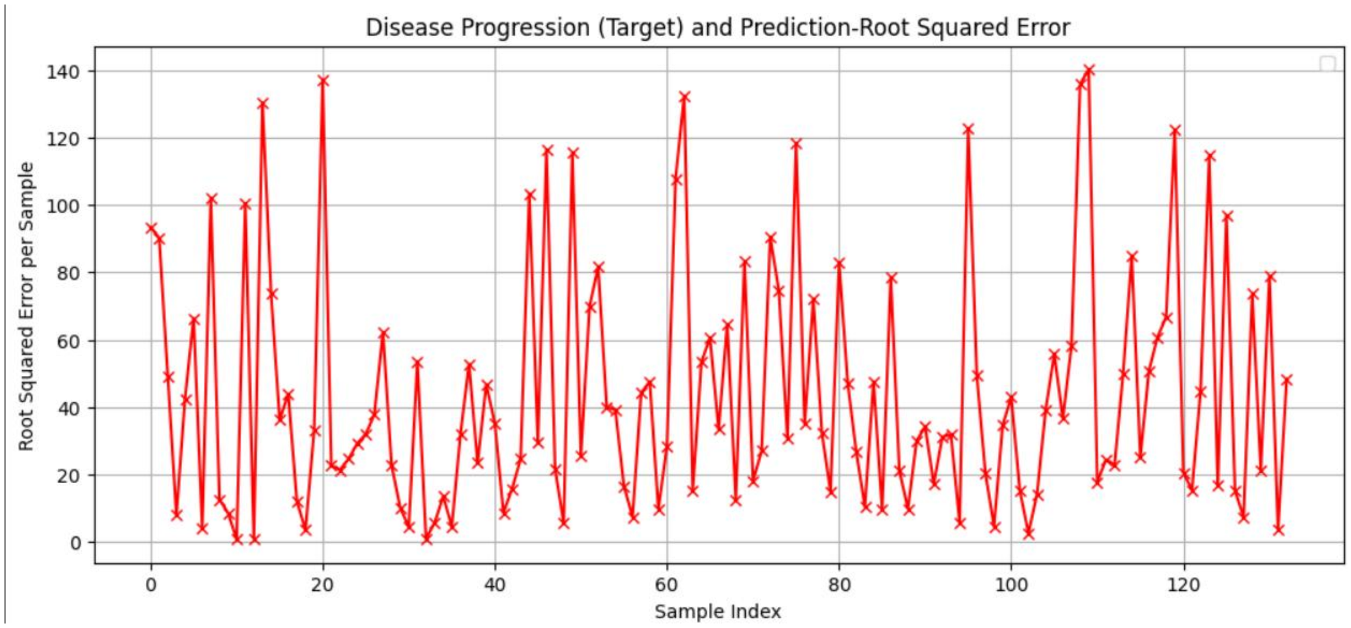


```
# Plot actual vs predicted values for each sample
plt.figure(figsize=(16, 6))

# Plot the actual values
plt.plot(range(len(y_test)), y_test, label="Actual Values", color='blue',
marker='o')

# Plot the predicted values
plt.plot(range(len(y_pred)), y_pred, label="Predicted Values", color='red',
marker='x')

# Set labels and title
plt.xlabel("Sample Index")
plt.ylabel("Disease Progression (Target)")
plt.title("KNN Regression - Actual vs Predicted Values for Each Sample")
```

```
# Display Mean Squared Error on the plot
plt.text(0, max(y_pred) - 5, f'Mean Squared Error: {mse:.2f}', color='black',
fontsize=12)

# Add legend and grid
plt.legend()
plt.grid(True)

# Show plot
plt.show()
```



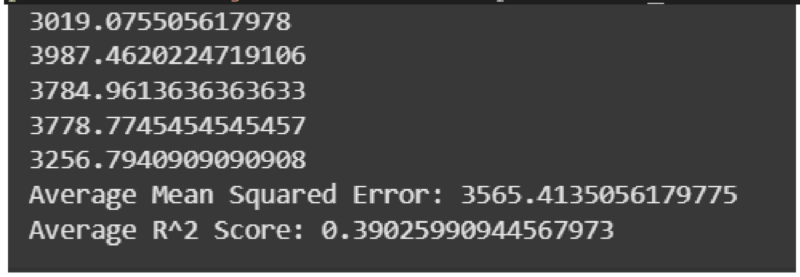
```
# Calculate Root Mean Squared Error

# Calculate RMSE
rmse = root_mean_squared_error(y_test, y_pred)
print(f"Root Mean Squared Error: {rmse:.2f}")

# calculate Root Squared Error for each sample in the test data
sample_rmse = np.sqrt((y_test - y_pred.flatten())**2)

# Plot the RSE per sample
plt.figure(figsize=(12, 5))
plt.plot(range(len(sample_rmse)), sample_rmse, color='red', marker='x')
#plt.plot(sample_rmse)
plt.title('Disease Progression (Target) and Prediction-Root Squared Error')
plt.xlabel('Sample Index')
plt.ylabel('Root Squared Error per Sample')

# Add legend and grid
plt.legend()
plt.grid(True)

# Show plot
plt.show()
```



```
# Define the number of folds (k)
k = 5

# Create a KFold object
kf = KFold(n_splits=k, shuffle=True, random_state=42)
# Custom function to compute "accuracy" for regression
def regression_accuracy(y_true, y_pred):
    # Round predictions and true values to integers
    y_pred_rounded = np.round(y_pred).astype(int)
    y_true_rounded = np.round(y_true).astype(int)
    return accuracy_score(y_true_rounded, y_pred_rounded)

# Create a custom scorer
custom_scorer = make_scorer(regression_accuracy)

# Perform k-fold CV with the custom scorer
scores = cross_val_score(knn, X_test, y_pred, cv=kf, scoring=custom_scorer)

print("Custom 'Accuracy' Scores:", scores)
print("Average Custom 'Accuracy':", np.mean(scores))
```

```
Custom 'Accuracy' Scores: [0.         0.         0.03703704 0.03846154 0.        ]
Average Custom 'Accuracy': 0.0150997150997151
```

```python
r2_scores = []   # R^2 scores for each fold
# Initialize an empty list to store the MSE scores
mse_scores = []
r2_scores = []   # R^2 scores for each fold
# Iterate through each fold
for train_index, test_index in kf.split(X):
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    # Create and train the k-NN regressor
    knn = KNeighborsRegressor(n_neighbors=5)  # Adjust n_neighbors as needed
    knn.fit(X_train, y_train)

    # Make predictions
    y_pred = knn.predict(X_test)
    #cross_val_score() # This line was causing the error. Removing it to
calculate MSE directly in the loop
    # Calculate MSE
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    mse_scores.append(mse)
    r2_scores.append(r2)
    print(mse)

# Print the average MSE
print(f"Average Mean Squared Error: {np.mean(mse_scores)}")
print(f"Average R^2 Score: {np.mean(r2_scores)}")
```



```
3019.075505617978
3987.4620224719106
3784.9613636363633
3778.7745454545457
3256.7940909090908
Average Mean Squared Error: 3565.4135056179775
Average R^2 Score: 0.39025990944567973
```

google colab (KNN):

https://colab.research.google.com/drive/1zNma2oX04updv3IEW2vlAIihuHLKb7iC?usp=sharing