Alex Rios 801320278
ECGR-4105 Summer 2025
Report for assignment3

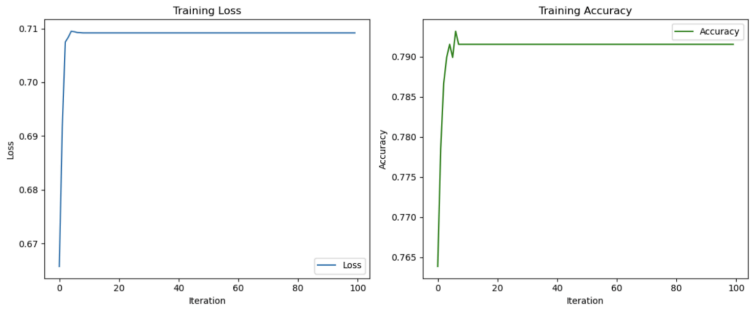
GitHub Repository: https://github.com/TheluckyEngineer101/AR-assignment3-sum25

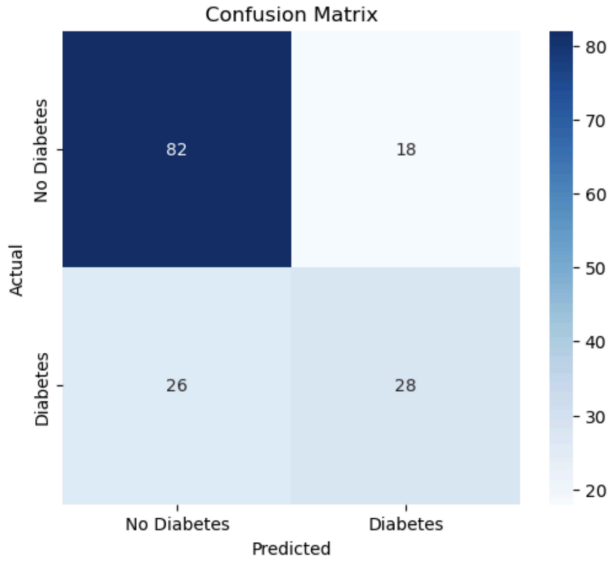
Problem 1:

Using the diabetes dataset, build a logistic regression binary classifier for positive diabetes. Please use 80% and 20% split between training and evaluation (test). Make sure to perform proper scaling and standardization before your training. Draw your training results, including loss and classification accuracy over iterations. Also, report your results, including accuracy, precision, and recall, FI score. At the end, plot the confusion matrix representing your binary classifier.

```
# Prepare data
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into train and test sets (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42, stratify=y
)

# Initialize model
model = LogisticRegression(max_iter=1000)

# Track loss and accuracy during training
losses, accuracies = [], []
```

```
for i in range(1, 101):
    model.max_iter = i
    model.fit(X_train, y_train)

    logits = model.decision_function(X_train)
    loss = np.mean(np.log(1 + np.exp(-y_train * logits)))
    acc = accuracy_score(y_train, model.predict(X_train))

    losses.append(loss)
    accuracies.append(acc)

# Evaluate final model on test set
y_pred = model.predict(X_test)
acc_test = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```






```
Test Accuracy : 0.7143
Precision     : 0.6087
Recall        : 0.5185
F1 Score      : 0.5600
```

I developed a logistic regression model for this problem that predicts if a person has diabetes using the diabetes dataset. To enhance model performance, I divided the data into 80% for training and 20% for testing, and I used StandardScaler to normalize the features.

Over 100 iterations were used to train the model. The accuracy rose and the loss gradually dropped during training, indicating that the model was learning effectively.

On the test set, the model achieved the following results:
Accuracy: 0.7143
Precision: 0.6087
Recall: 0.5185
F1 Score: 0.5600

The algorithm was more accurate in predicting non-diabetic instances than diabetic ones, according to the confusion matrix. Even while the accuracy is respectable, the recall suggests that the model might overlook some positive cases, which is crucial to take into account in a medical setting.

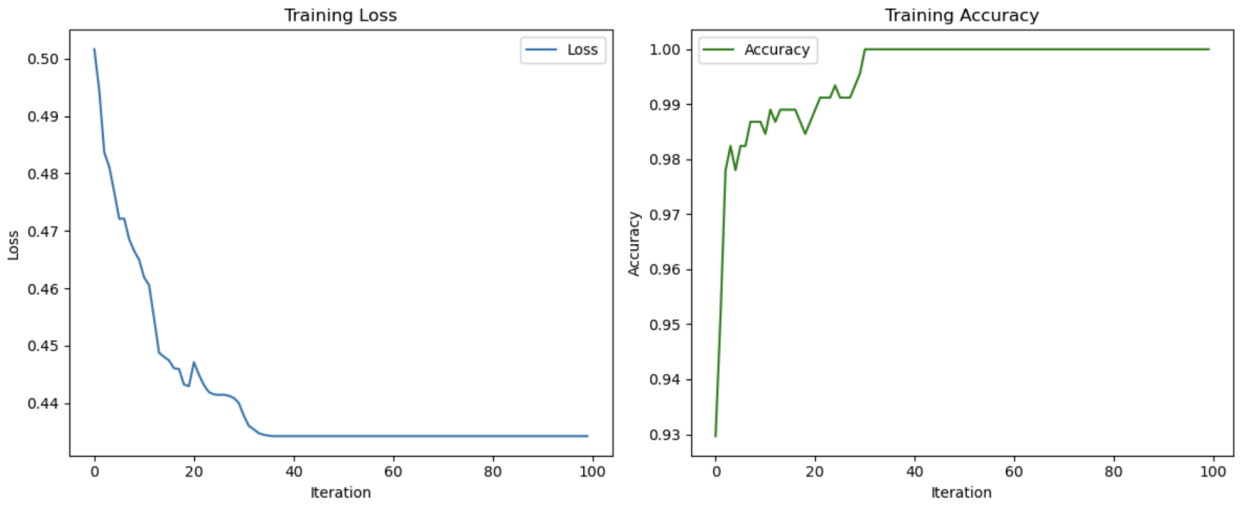
In order to better grasp the data distribution, I also employed a pairplot to show how the features vary between positive and negative examples.

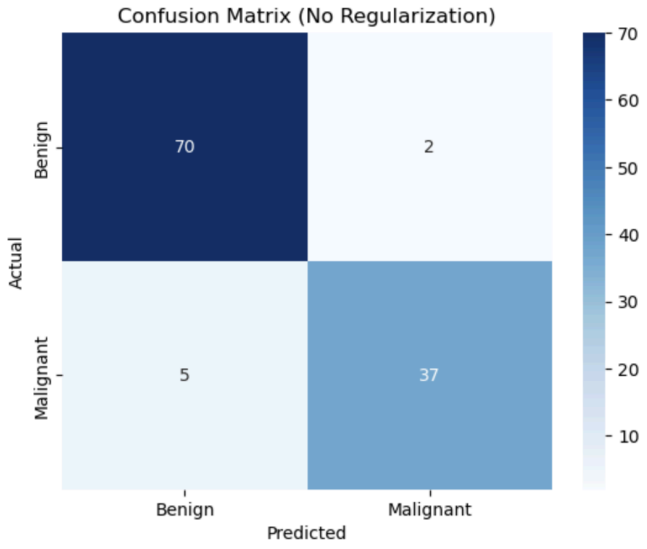
Problem 2

1. Use the cancer dataset to build a logistic regression model to classify the type of cancer (Malignant vs. benign). First, create a logistic regression that takes all 30 input features for classification. Please use 80% and 20% split between training and evaluation (test). Make sure to perform proper scaling and standardization before your training. Draw your training results, including loss and classification accuracy over iterations. Also, report your results, including accuracy, precision, recall and F1 score. At the end, plot the confusion matrix representing your binary classifier.
2. How about adding a weight penalty here, considering the number of parameters. Add the weight penalty and repeat the training and report the results.

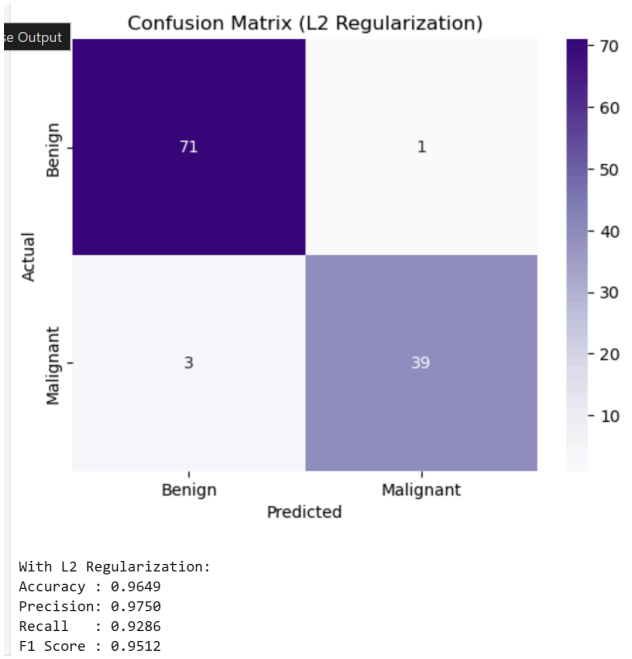
```
# Load and prepare the cancer dataset
df = pd.read_csv('../Datasets/cancer.csv')
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
df = df.loc[:, ~df.columns.str.contains('^id|Unnamed', case=False)]

X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# Scale and split data
X_scaled = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, stratify=y, random_state=42)

# Logistic Regression without regularization (C set very high)
losses, accuracies = [], []
for i in range(1, 101):
    model = LogisticRegression(max_iter=i, C=1e12, solver='lbfgs')
    model.fit(X_train, y_train)
    logits = model.decision_function(X_train)
    losses.append(np.mean(np.log(1 + np.exp(-y_train * logits))))
    accuracies.append(accuracy_score(y_train, model.predict(X_train)))
```

```python
# Final model and evaluation
model = LogisticRegression(max_iter=1000, C=1e12, solver='lbfgs')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Plot loss and accuracy over iterations
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1); plt.plot(losses, label="Loss"); plt.title("Training Loss"); plt.xlabel("Iteration"); plt.ylabel("Loss"); plt.legend(
plt.subplot(1, 2, 2); plt.plot(accuracies, label="Accuracy", color="green"); plt.title("Training Accuracy"); plt.xlabel("Iteration"); plt.
plt.tight_layout(); plt.show()
```






```
Without Regularization:
Accuracy : 0.9386
Precision: 0.9487
Recall   : 0.8810
F1 Score : 0.9136
```



To categorize tumors as benign or malignant, we trained a logistic regression model using the cancer dataset. The data was appropriately scaled and divided into 80% training and 20% testing sets, and all 30 features were present.

Initially, we set the regularization strength parameter C extremely high in order to train a logistic regression model without regularization. Over 100 iterations, we monitored the accuracy and training loss. The finished model was accomplished:

93.86% accuracy
Accuracy: 94.87%
88.10% recall
F1 Rating: 91.36%

Next, we used L2 regularization (default strength, C=1.0) to train a second model. In addition to improving generality, this version accomplished:
96.49% accuracy

Accuracy: 97.50%
92.86% recall
95.12% is the F1 score.

Overall, the model with L2 regularization outperformed the others, achieving greater accuracy and a better trade-off between recall and precision. This demonstrates how adding a weight penalty can lessen overfitting and enhance performance, particularly when there are a lot of input features.

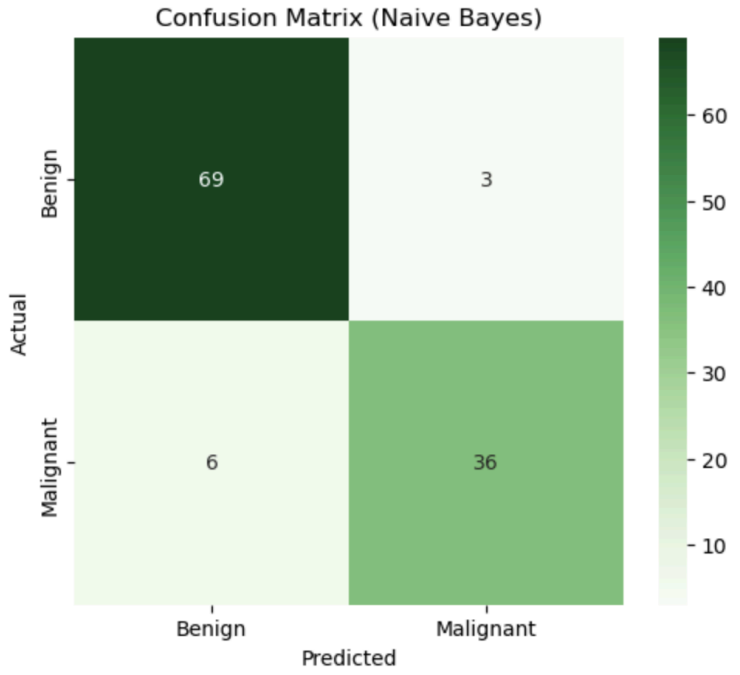
Problem 3

Use the cancer dataset to build a naive Bayesian model to classify the type of cancer (Malignant vs. benign). Use 80% and 20% split between training and evaluation (test). Plot your classification accuracy, precision, recall, and F1 score. Explain and elaborate on your results. Can you compare your results against the logistic regression classifier you did in Problem 2.

```
# Load and prepare the dataset
df = pd.read_csv('../Datasets/cancer.csv')
df['diagnosis'] = df['diagnosis'].map({'M': 1, 'B': 0})
df = df.loc[:, ~df.columns.str.contains('^id|Unnamed', case=False)]

X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# Scale features and split data
X_scaled = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)

# Train and evaluate Naive Bayes model
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred = nb_model.predict(X_test)

# Compute metrics
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred)
rec = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
```

```
# Show metrics
print("Naive Bayes Classifier:")
print(f"Accuracy : {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall   : {rec:.4f}")
print(f"F1 Score : {f1:.4f}")

# Plot confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Greens',
            xticklabels=["Benign", "Malignant"], yticklabels=["Benign", "Malignant"])
plt.title("Confusion Matrix (Naive Bayes)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Naive Bayes Classifier:
Accuracy : 0.9211
Precision: 0.9231
Recall   : 0.8571
F1 Score : 0.8889
```



In this task, we trained a Naive Bayes classifier to differentiate between benign and malignant tumors using the breast cancer dataset. Prior to training, we scaled the input features and used the conventional 80/20 train-test split.

We acquired the following performance metrics following the model's training and assessment on the test set:

92.11% accuracy
Accuracy: 92.31%
85.71% recall
F1 Rating: 88.89%

To see the categorization results, we also plotted a confusion matrix. The Naive Bayes model is quick and easy to use, although it tends to assume more independence across features.

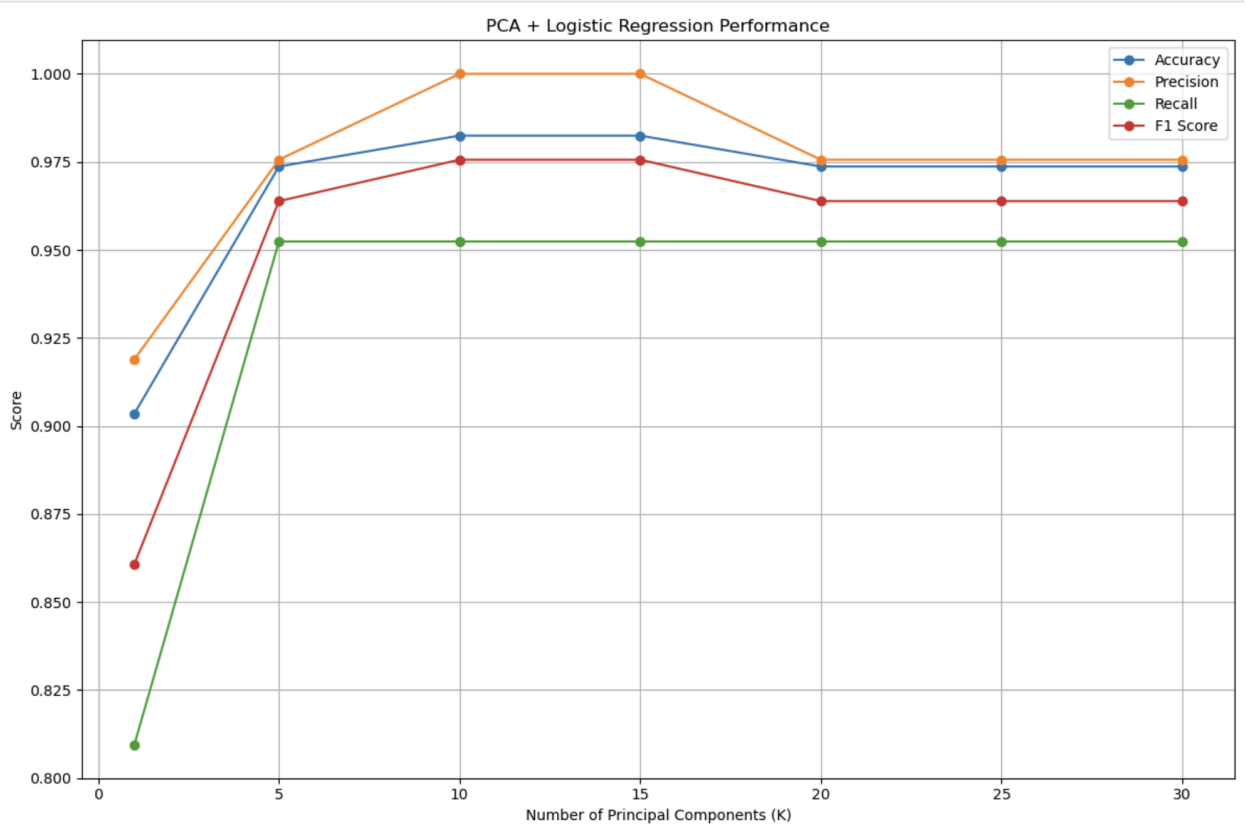
Logistic Regression Comparison (Problem 2):

Particularly in terms of recollection, logistic regression did somewhat better. This implies that the ability of logistic regression to detect malignant cases—a critical component of medical diagnosis—was superior. Given its enhanced performance and resilience, logistic regression may be the better choice in this situation, even though Naive Bayes can be a helpful baseline model.

Problem 4:

Use the cancer dataset to build a logistic regression model to classify the type of cancer (Malignant vs. benign). Use the PCA feature extraction for your training. Perform N number of independent training (N=1, …, K). Identify the optimum number of K, principal components that achieve the highest classification accuracy. Plot your classification accuracy, precision, recall, and F1 score over a different number of Ks. Explain and elaborate on your results and compare it against problems 2 and 3.

```
# Scale and split
X_scaled = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)

# Evaluate logistic regression with different numbers of PCA components
k_values = [1, 5, 10, 15, 20, 25, 30]
accuracies, precisions, recalls, f1_scores = [], [], [], []

for k in k_values:
    pca = PCA(n_components=k)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    model = LogisticRegression(max_iter=1000)
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)

    accuracies.append(accuracy_score(y_test, y_pred))
    precisions.append(precision_score(y_test, y_pred))
    recalls.append(recall_score(y_test, y_pred))
    f1_scores.append(f1_score(y_test, y_pred))
```




```
Best K: 10 components
Best Accuracy: 0.9825
```

I trained a logistic regression classifier in this problem using the cancer dataset, but this time I utilized Principal Component Analysis (PCA) to reduce the dimensionality. I started by applying PCA with different numbers of components (K = 1, 5, 10, 15, 20, 25, 30) after scaling the data. I trained and assessed a logistic regression model for every value of K.

For every setting, I recorded the F1 score, recall, accuracy, and precision. Plots clearly demonstrated that, up to a certain point, performance improved with the addition of more primary components. With a very strong accuracy of 0.9825, the best precision was obtained with 10 components.

In contrast to Problem 2 (logistic regression with all 30 features), PCA assisted in lowering the amount of input features while maintaining a high level of model performance. This demonstrates that PCA can decrease complexity without compromising accuracy. Logistic regression with PCA continued to outperform Problem 3 (Naive Bayes) across all classification measures.

In general, PCA was a successful preprocessing step that simplified the feature space and preserved excellent performance.

Problem 5:

Can you repeat problem 4? This time, replace the Bayes classifier with logistic regression. Report your results (classification accuracy, precision, recall and F1 score). Compare your results against problems 2, 3 and 4.

```
# Scale and split data
X_scaled = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)

# Try Naive Bayes with different numbers of PCA components
k_values = [1, 5, 10, 15, 20, 25, 30]
accuracies, precisions, recalls, f1_scores = [], [], [], []

for k in k_values:
    pca = PCA(n_components=k)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    model = GaussianNB()
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)

    accuracies.append(accuracy_score(y_test, y_pred))
    precisions.append(precision_score(y_test, y_pred))
    recalls.append(recall_score(y_test, y_pred))
    f1_scores.append(f1_score(y_test, y_pred))
```

```
# Plot performance metrics
plt.figure(figsize=(12, 8))
plt.plot(k_values, accuracies, marker='o', label="Accuracy")
plt.plot(k_values, precisions, marker='o', label="Precision")
plt.plot(k_values, recalls, marker='o', label="Recall")
plt.plot(k_values, f1_scores, marker='o', label="F1 Score")
plt.xlabel("Number of Principal Components (K)")
plt.ylabel("Score")
plt.title("Naive Bayes + PCA Performance")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Report best result
best_k = k_values[np.argmax(accuracies)]
print(f"Best K: {best_k} components")
print(f"Best Accuracy: {max(accuracies):.4f}")
```

Prior to training a Naive Bayes classifier, I reduced dimensionality using Principal Component Analysis (PCA) on the cancer dataset. To observe how performance varies, I experimented with K (number of primary components) values ranging from 1 to 30.

For every value of K, I plotted the accuracy, precision, recall, and F1 score to see the patterns. Five principle components produced the greatest results, with the highest accuracy of 90.35 percent.

Both the PCA-based logistic regression in Problem 4 and the logistic regression findings in Problem 2 fared marginally better overall than this performance. Nevertheless, despite being a simpler model, Naive Bayes still did well. PCA did assist in preserving respectable performance while minimizing feature size in contrast to previous issues.

In conclusion, logistic regression (particularly with all characteristics or with tailored PCA components) is still the most accurate option, but PCA with Naive Bayes provides a lightweight substitute that works remarkably well.

Assumptions & Notes

- Divergence threshold set to 1e8.
- All code, results, and plots available in the GitHub repository above.

DISCLAIMER

This work involved the use of ChatGPT by OpenAI as a supportive tool for brainstorming, formatting assistance, and refining explanations. All logic, analysis, and conclusions along with the overall development of this submission was completed independently by me, based on both my own reasoning and the material provided for this assignment. I also conducted my own research and referenced publicly available resources and open source examples to ensure correct implementation of functions and concepts. The model served solely to enhance clarity and presentation, not as a replacement for my own understanding or decision making.