Linear Regression Analysis on Students' Performance Dataset

1. Introduction

The **StudentsPerformance.csv** dataset contains academic performance records of students along with their demographic information. This dataset is particularly valuable for understanding how different factors influence students' scores in three core subjects: **Math, Reading, and Writing**. Each entry represents an individual student, with attributes such as:

- **Gender**
- **Race/ethnicity**
- **Parental level of education**
- **Lunch type**
- **Test preparation course completion status**
- **Scores in Math, Reading, and Writing**

Libraries Used:

- **NumPy**: For numerical computations and array manipulations.
- **Pandas**: For data loading, cleaning, and preprocessing.
- **Matplotlib & Seaborn**: For data visualization (histograms, scatter plots, regression plots).
- **Scikit-learn (sklearn)**: For machine learning tasks, including:
    - Data splitting (train_test_split)
    - Model training (LinearRegression)
    - Model evaluation (R^2 score)

---

2. Exploratory Data Analysis (EDA)

Before applying any machine learning model, we conducted an Exploratory Data Analysis (EDA) to understand the dataset's structure, detect anomalies, and identify relationships between variables.

Key Steps in EDA

1. **Data Loading & Initial Inspection**
   - The dataset was loaded using pd.read_csv().
   - Basic information was extracted using:

     df.head() → First few records

     df.describe() → Statistical summary (mean, std, min, max, etc.)

     df.info() → Data types and missing values

---

2. **Data cleaning**

   - **No Missing Values**: Checked using isnull().sum() — no missing values found.
   - **No Duplicates**: Verified with duplicated().sum() — no duplicate rows detected.

   The data was clean and ready for analysis and modeling without any additional preprocessing.

---

3. **Data Visualization**

- **Histograms**

  Plotted the distributions of Math, Reading, and Writing scores to assess normality and detect outliers.
  Observation:
  The distributions appeared approximately normal, with no significant skewness or extreme outliers. No additional cleaning was required.

- **Correlation Heatmap**

  Used sns.heatmap() to visualize the correlation between the numeric score features.
  Key Findings:
  Reading ↔ Writing: Very strong positive correlation (≈ 0.95)
  Math↔ Reading/Writing: Strong positive correlation (≈ 0.80)

- **Scatter Plots**

  Plotted pairwise relationships between scores using sns.scatterplot() (Writing vs. Reading)

  Insight: Clear linear relationships were observed, supporting the use of Linear Regression techniques.

4. Data Preprocessing

To prepare the data for modeling, we performed the following steps:

A. Feature Selection

Selected relevant features (math score, reading score, and writing score). The target variable was set to writing score.

B. Feature scaling

Scaling was not applied because all numeric values in the dataset were already within a similar range (0–100).

C. Train-Test Split

The dataset was divided into:

a. Training set (80%) → Used to train the model.
b. Test set (20%) → Used to evaluate model performance.
c. Used train_test_split() from Scikit-learn.

5. Simple Linear Regression Model

we built a simple linear regression model using one numeric feature to predict the target.

Selected Feature:

The feature that showed the highest linear correlation with the target (writing score) was reading score.

Model Implementation:

- Linear Regression model was trained using only the reading score as input (X) and writing score as the target (y).
- The model learned a straight-line relationship between the two variables.

Evaluation:

- The model was evaluated using: R^2 Scorere: to measure the proportion of variance explained. Mean Squared Error (MSE): to measure average prediction error.
- A scatter plot with the regression line was used to visualize the fit.

6. Multiple Linear Regression Model

we extended the regression model to include multiple input features in order to improve prediction accuracy.

Selected Features:

- The input features used were:
  - math score
  - reading score
- The target variable remained writing score.

Model Implementation:

- A Linear Regression model was trained using both features simultaneously.
- This allowed the model to capture combined effects from multiple subjects on the writing score.

Evaluation:

- The model was evaluated using:
  - R^2 Score
  - Mean Squared Error (MSE)
- A comparison was made between the simple and multiple regression results to see which performed better.

Comparison between Simple and Multiple Linear Regression

Simple and Multiple Linear Regression models were applied to predict the writing score.

- Simple Linear Regression was based on a single feature: reading score.
- Multiple Linear Regression used multiple features: reading score and math score.

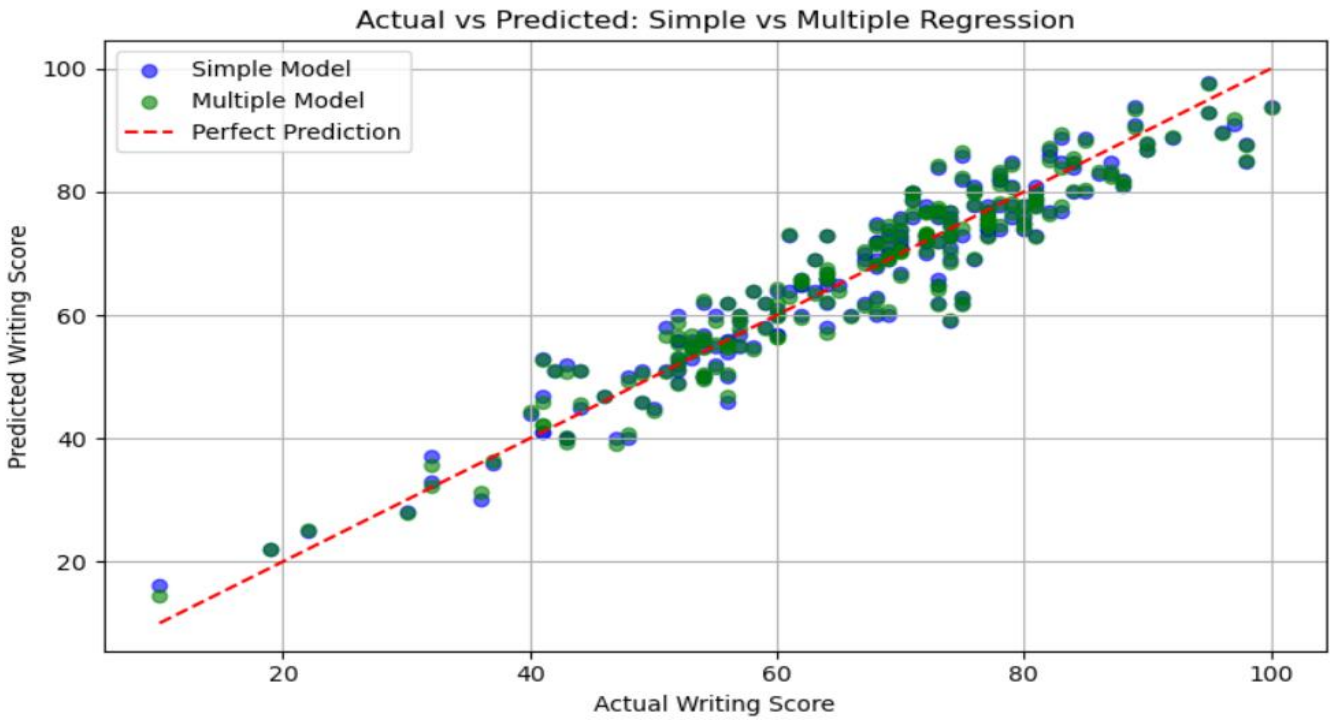
Observations from the Graph:

- The blue points represent predictions from the Simple Model.
- The green points represent predictions from the Multiple Model.
- The red dashed line indicates the perfect prediction line (where predicted = actual).

Key Comparison:

Criteria	Simple Regression	Multiple Regression
Features Used	Reading Score only	Reading + Math
Accuracy (Visual)	Good, but more scattered	Better alignment with the red line
Prediction Quality	Less precise	More accurate and consistent

The Multiple Regression Model provided more accurate predictions as it captured more information from the dataset by using multiple features. This is visually supported by how closely the green points align with the perfect prediction line, compared to the blue points.



7. Polynomial Regression

Polynomial Regression was used to investigate whether a non-linear model could better capture the relationship between reading scores and writing scores, compared to simple linear regression.

Selected Feature & Data Split

- Input: Reading score
- Target: Writing score
- Data split: 80% training, 20% testing
- No feature scaling was required since all values were on a consistent 0–100 scale.

Tested Degrees

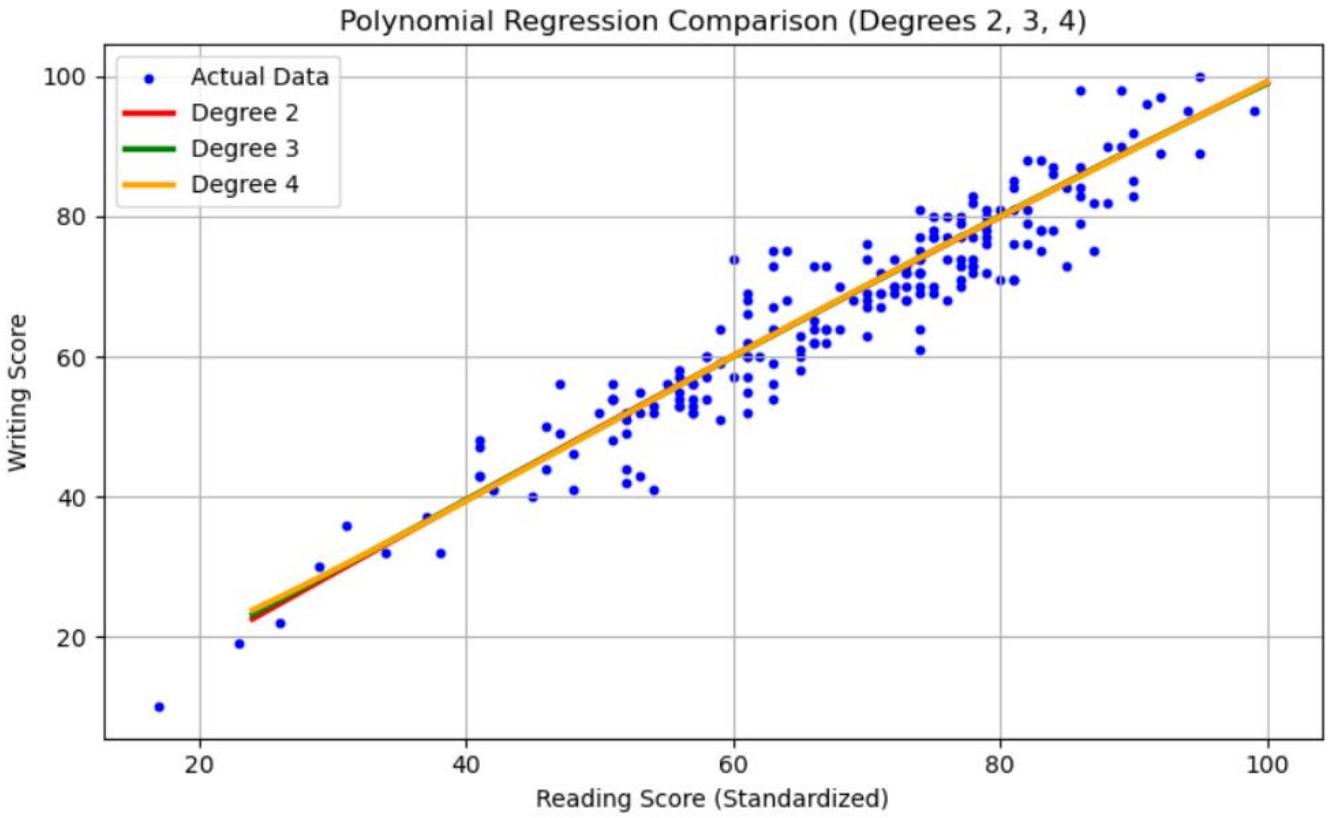
Polynomial regression models were built using degree 2, degree 3, and degree 4:
For each degree:

- Polynomial features were generated using PolynomialFeatures.
- A LinearRegression model was trained on the transformed features.
- Predictions were made on the test data.
- Model performance was evaluated using R^2 Score and Mean Squared Error (MSE).

Results & Visualization

Visual inspection showed that all polynomial curves were very close to a straight line, confirming a mostly linear relationship between the variables. The higher-degree models did not provide meaningful improvement in predictive power.

Degree	R^2 Score	MSE
2	0.901	23.92
3	0.900	24.00
4	0.900	24.13



Breast Cancer Dataset & Modeling Overview

1. Introduction

The Breast Cancer Wisconsin (Diagnostic) dataset contains medical records for 569 patients, with 30 numerical features describing tumor characteristics (like radius, texture, and area). The goal is to predict whether a tumor is benign or malignant.

To achieve that, we'll follow these steps:

- Preprocessing: Encode labels, scale features, and split data.
- EDA: Explore feature distributions and correlations.

- Modeling: Apply and compare multiple classifiers:
    - Logistic Regression
    - KNN
    - Cross Validation (5 fold)
    - SVM (Linear - Polynomial - Radial Basis Function)
    - Neural Network (NN)

Evaluation: Use metrics like Accuracy, Precision, Recall, F1-score, and ROC-AUC to compare model performance.

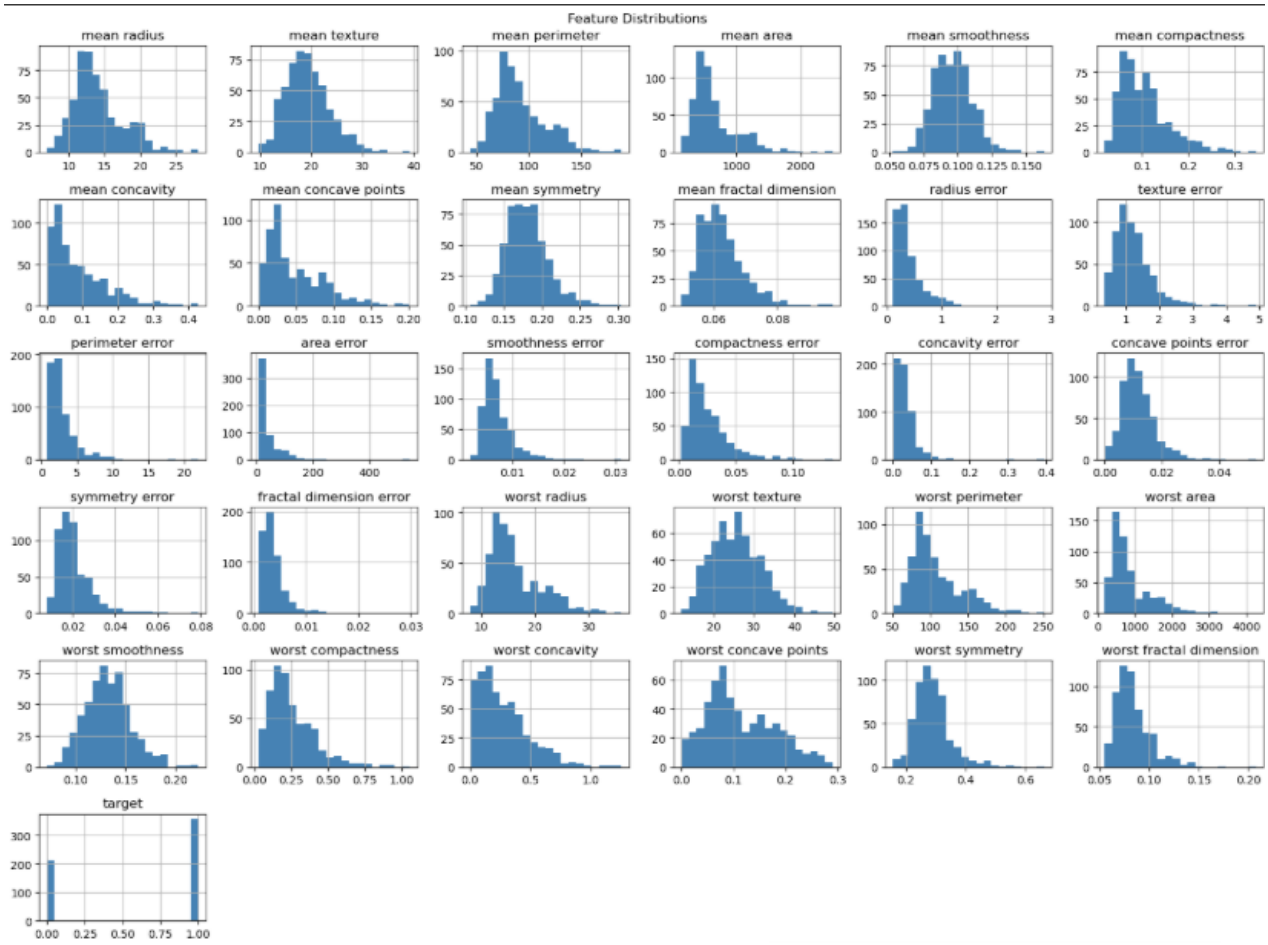
2. Data Preprocessing And EDA.

- **The dataset was loaded using** pd.read_csv().
- **Basic exploration was performed:**
    - df.head() → Previewed the first few rows.
    - df.describe() → Showed statistical summary of all numerical features.
    - df.info() → Displayed data types and confirmed no missing values.
- **No Missing Values**: Confirmed using df.isnull().sum() — all features were complete.
- **No Duplicates**: Checked with df.duplicated().sum() — no duplicate rows found.

The dataset was clean, well-structured, and required no additional preprocessing steps before analysis and modeling.

3. Data Visualization

- **Histogram** was plotted for each numerical feature in the data to examine the distribution of values. Most variables were right-skewed, indicating a few high values. Some features, such as worst smoothness and worst symmetry, were closer to a normal distribution. The plot helped understand the shape of the data, reveal the extent of dispersion, and identify potential scaling or transformations before building the model.



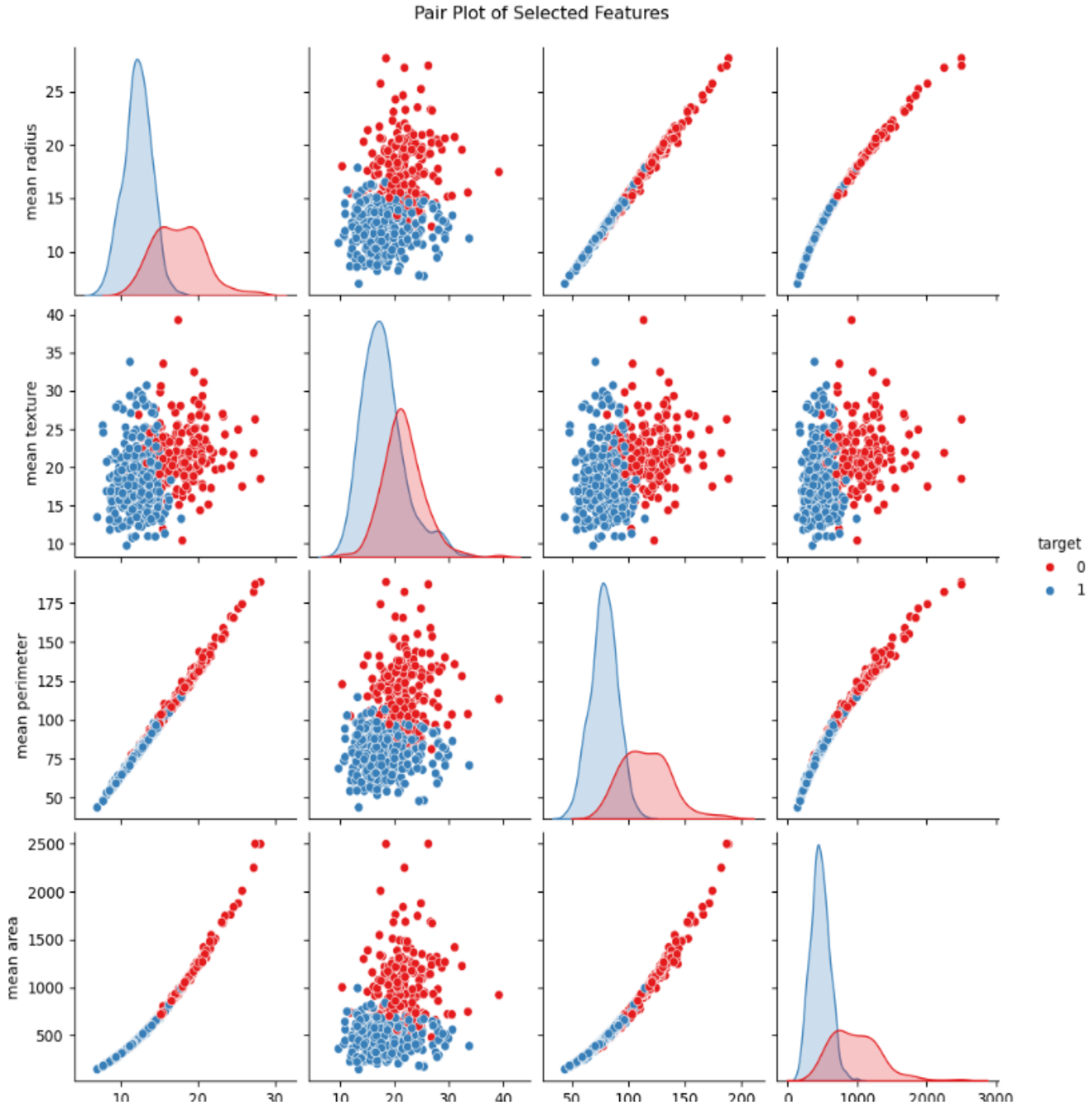
- **Pairplot Analysis**

  sns.pairplot() was used to display the pairwise relationships between some of the key features in the data, with points colored according to the target value.

  The plot showed a clear separation between the two types (benign and malignant) in some pairs, such as:

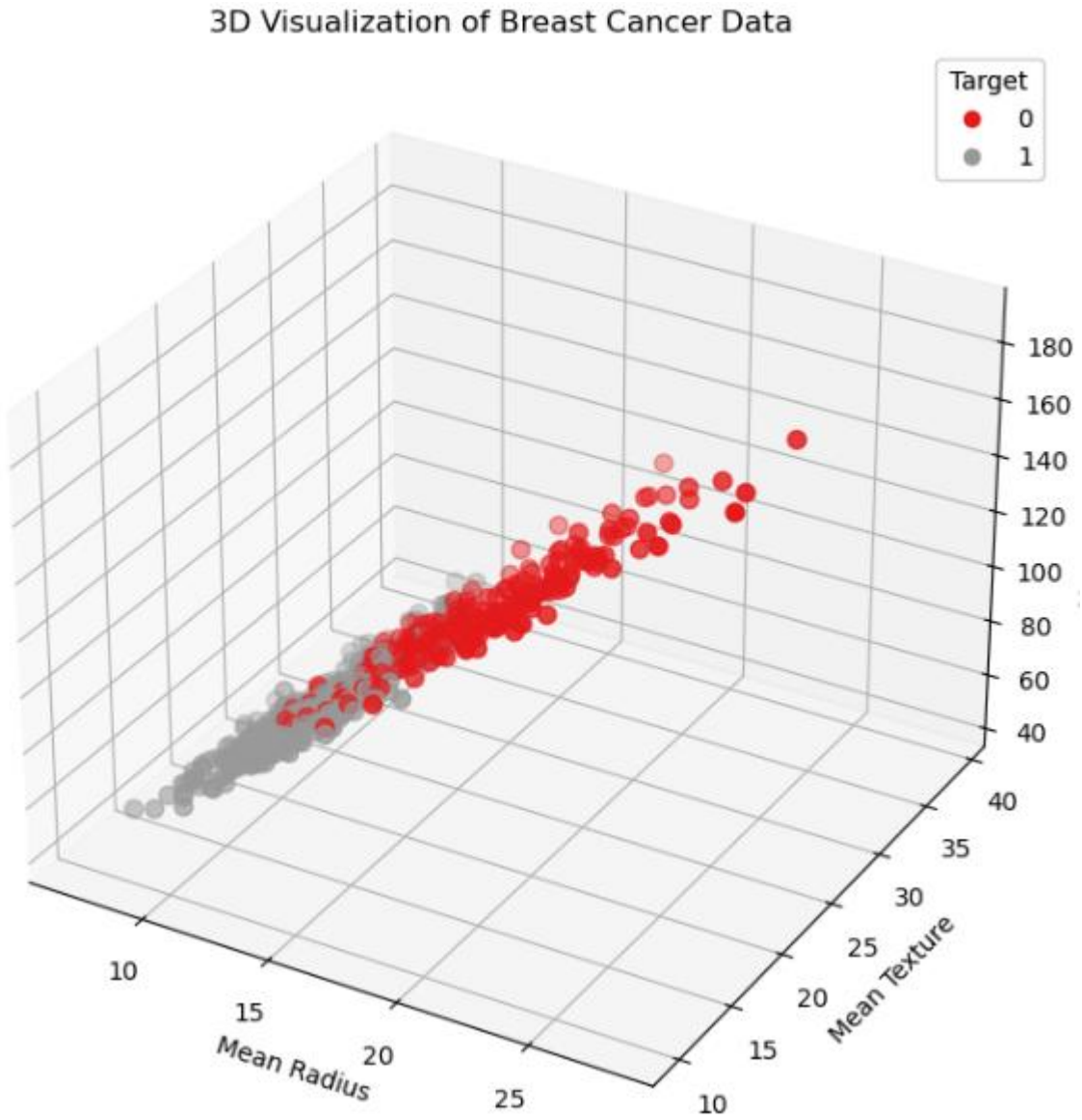
- mean radius vs. mean texture
- mean perimeter vs. mean area

This demonstrates that there are strong relationships between some variables, which is very useful in selecting appropriate features for building classification models later.



- **3D Scatter Plot Analysis** created to examine the relationships between three important features of the breast cancer dataset:
    - **Mean Radius**
    - **Mean Texture**
    - **Mean Perimeter**

  with points color-coded based on the target variable (benign or malignant). The plot clearly shows a distinct separation between the two tumor types, highlighting that these features play a significant role in distinguishing between benign and malignant tumors.



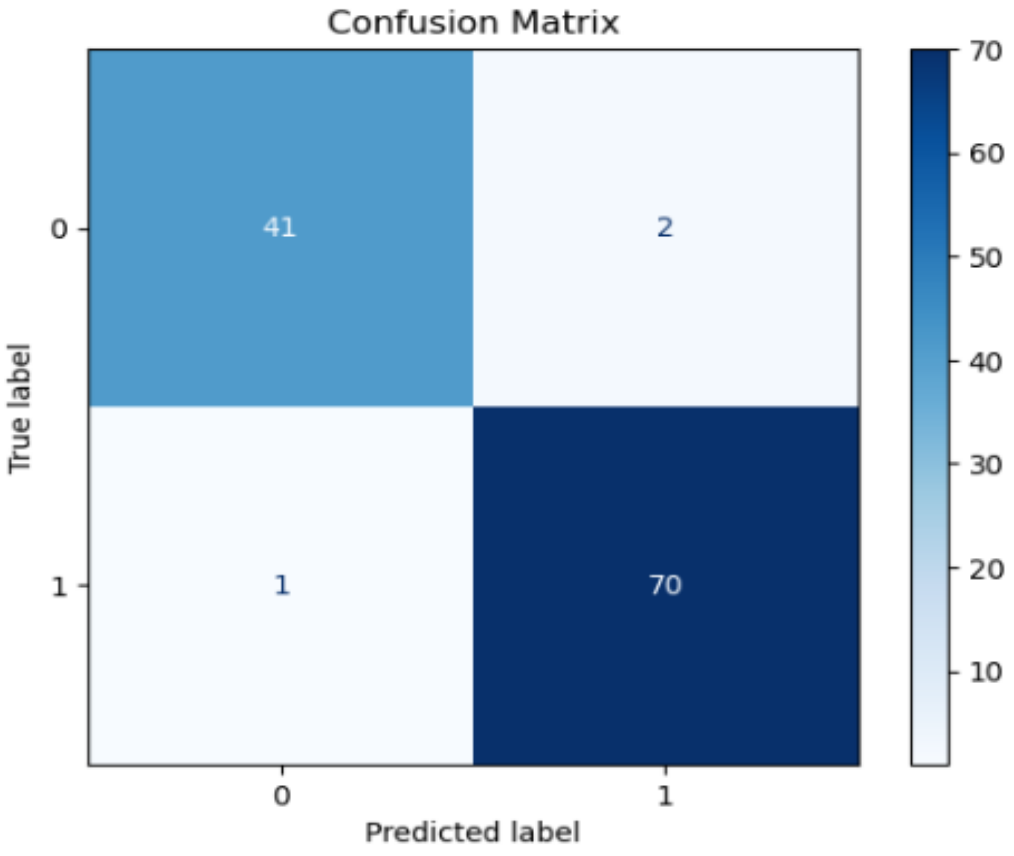
Models

1. Logistic Regression

In this step, we built and trained a Logistic Regression model.

Tasks Completed:

- Created a Logistic Regression model using LogisticRegression() from sklearn.
- Trained the model on the training data (X_train, y_train).
- Used the trained model to predict outcomes on the test data (X_test).
- Generated a confusion matrix using confusion_matrix() to compare predicted vs actual labels.
- Visualized the confusion matrix using a heatmap with seaborn for better clarity.
- The matrix helps us understand:
    - Correct predictions for both classes.
    - Misclassifications by the model.



2. KNN Model

- We experimented with different values of **K** (from 1 to 20) to find the best performing one. For each value of K, we trained the model and evaluated it on the validation set.

- The optimal **K** value was found to be **K=7**, which provided the highest accuracy on the validation set.

[151]:
```
# Hyperparameter Tuning
best_k = 1  # Initialize with a default value
best_accuracy = 0.0  # Initialize with 0

for k in range(1, 21):  # Try k values from 1 to 20
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_val = knn.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred_val)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k

print(f"Best k: {best_k}",",",f"Best_accuracy: {best_accuracy}")
```

```
Best k: 7 , Best_accuracy: 0.9649122807017544
```

- After identifying the optimal value of K through validation (K=7), we proceeded to train and evaluate the final KNN model using this configuration. The model's performance was assessed on the training, validation, and test sets using both Accuracy and Log Loss as .

```
Training Set Results:
==================================================
Training Accuracy: 0.9208211143695014
Training Loss: 0.1669480983983188
==================================================
Validation Set Results:
==================================================
Validation Accuracy: 0.9649122807017544
Validation Loss: 0.11915155373450227
==================================================
Testing Set Results:
==================================================
Test Accuracy: 0.9824561403508771
Test Loss: 0.07679061604088179
```

These results indicate that the model performs very well and generalizes effectively to unseen data. The relatively low training loss along with the high validation and test accuracy values suggest that the model is not overfitting. Instead, it maintains a good balance between bias and variance.

3. Cross-Validation

To ensure the model's generalization ability, we applied **5-Fold Cross-Validation**. In 5-fold cross-validation, the training data is divided into 5 equal parts. The model is trained on 4 parts and validated on the 5th, and this process is repeated 5 times.

- **Average Cross-Validation Accuracy**: The average accuracy across the 5 folds was **88.56%**, which was consistent with the validation and test set accuracies.
- **Model Performance Comparison**

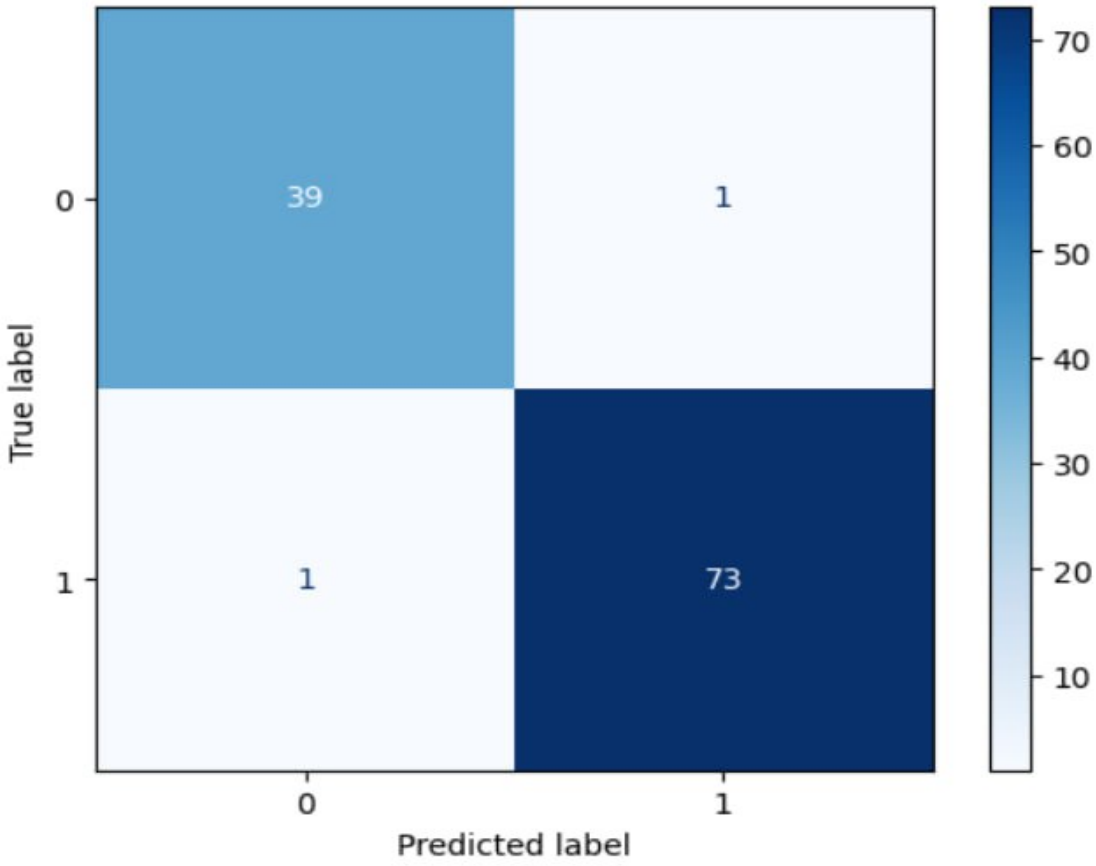
Evaluation Method	Accuracy	Loss (if available)
Cross-Validation (5-fold)	0.8857	N/A
Validation Set	0.9649	0.1192
Test Set	0.9825	0.0768

These consistent results demonstrate that the model is stable and not overfitting.

Confusion Matrix & Metrics

- After evaluating the final model on the test set, we generated a confusion matrix. The matrix showed that the model made very few misclassifications.
- We used **classification_report()** to compute the following metrics:
  - Accuracy: 0.9825 (98.25% of the samples were classified correctly).
  - Precision: 0.98 (The model is accurate in predicting positive cases).
  - Recall: 0.98 (The model identifies most positive cases).
  - F1-Score: 0.98 (Good balance between precision and recall).

- The confusion matrix heatmap clearly visualized the classification performance



Overfitting Discussion

To evaluate whether the KNN model suffers from overfitting, we compared its performance across the training, validation, and test sets:

- Training Accuracy: 92.08%
- Validation Accuracy: 96.49%
- Test Accuracy: 98.25%
- 5-Fold Cross-Validation Accuracy: 88.57%

From these results, we observe that the model performs slightly better on the validation and test sets than on the training set. This indicates that the model does not suffer from overfitting. In fact, the slightly lower training accuracy suggests that the model generalizes well and is not memorizing the training data.

4.Support Vector Machine (SVM)

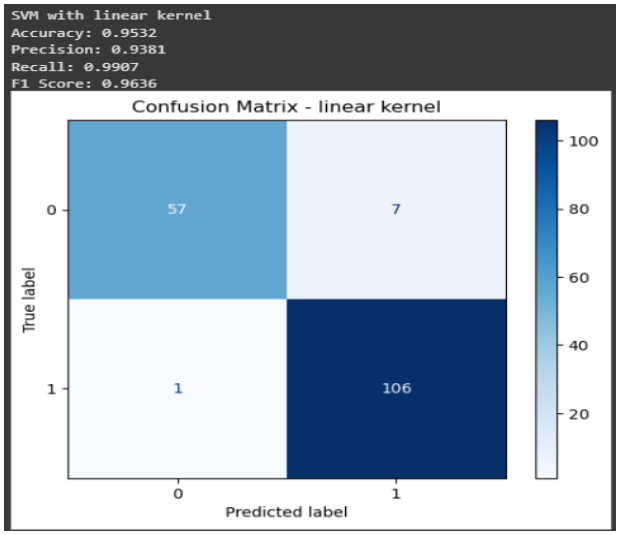
In this step, I implemented a Support Vector Machine (SVM) classifier using the scikit-learn library. I experimented with three different kernel types to evaluate their performance on the dataset:

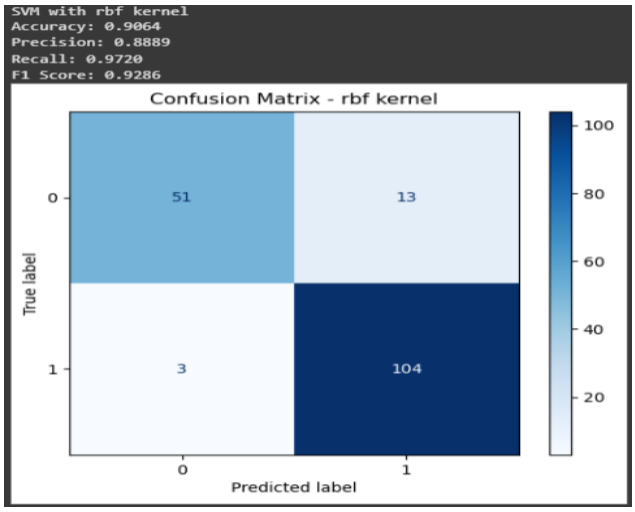
Kernels Used:

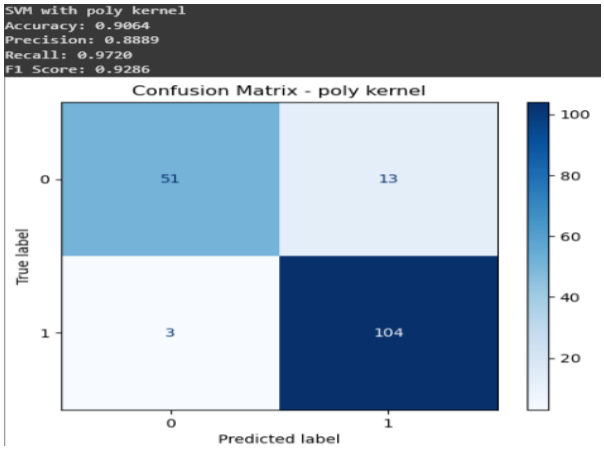
- Linear kernel
- Polynomial (poly) kernel
- Radial Basis Function (RBF) kernel

Model Training & Evaluation:

For each kernel, the model was trained on the training data and evaluated on the test set. I measured performance using accuracy and confusion matrix for a better understanding of model predictions.







Results Comparison:

Kernel Type	Test Accuracy
Linear	**0.9532**
Polynomial	0.9064
RBF	0.9064

5. Neural Network Implementation (NN)

This section describes the steps taken to build, train, and evaluate a Neural Network model for binary classification.

Model Architecture

- Built a **Sequential** model using Keras with the following layers:
  - Dense(64, activation='relu') — First hidden layer
  - Dropout(0.3) — Regularization to reduce overfitting
  - Dense(32, activation='relu') — Second hidden layer
  - Dense(1, activation='sigmoid') — Output layer for binary classification

---

Compilation

- Compiled the model using:
  - **Loss**: binary_crossentropy
  - **Optimizer**: Adam with a learning rate of 0.001
  - **Metric**: accuracy

Training

- Trained the model for up to 50 epochs with:
  - batch_size=32
  - validation_split=0.2
  - **EarlyStopping** callback: patience of 5 and restore_best_weights=True

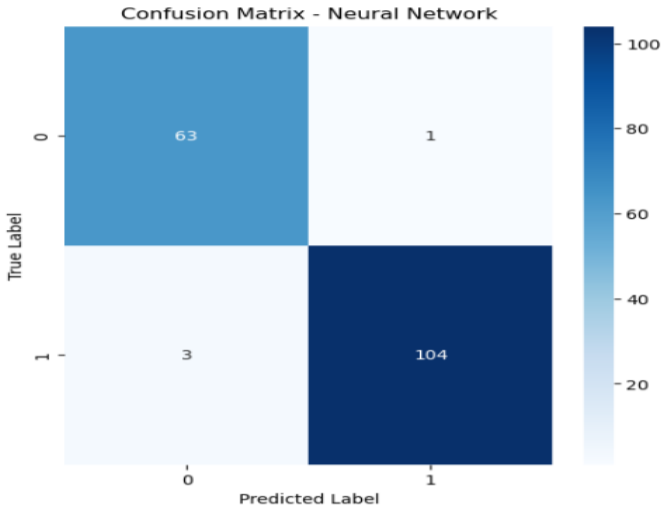
---

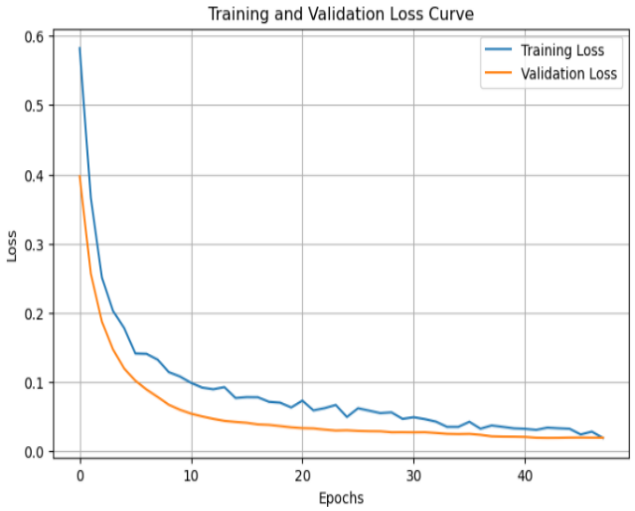
Evaluation

- Final **Test Accuracy**: **97.66%**
- Confusion Matrix:
  - True Negatives: 63
  - False Positives: 1
  - False Negatives: 3
  - True Positives: 104
- Classification Report:
  - **Precision**: 0.95 (class 0), 0.99 (class 1)
  - **Recall**: 0.98 (class 0), 0.97 (class 1)
  - **F1-score**: 0.97 (class 0), 0.98 (class 1)

Loss Curves

- Both **training** and **validation loss** decreased consistently.
- No signs of overfitting — validation loss remained lower or close to training loss throughout.



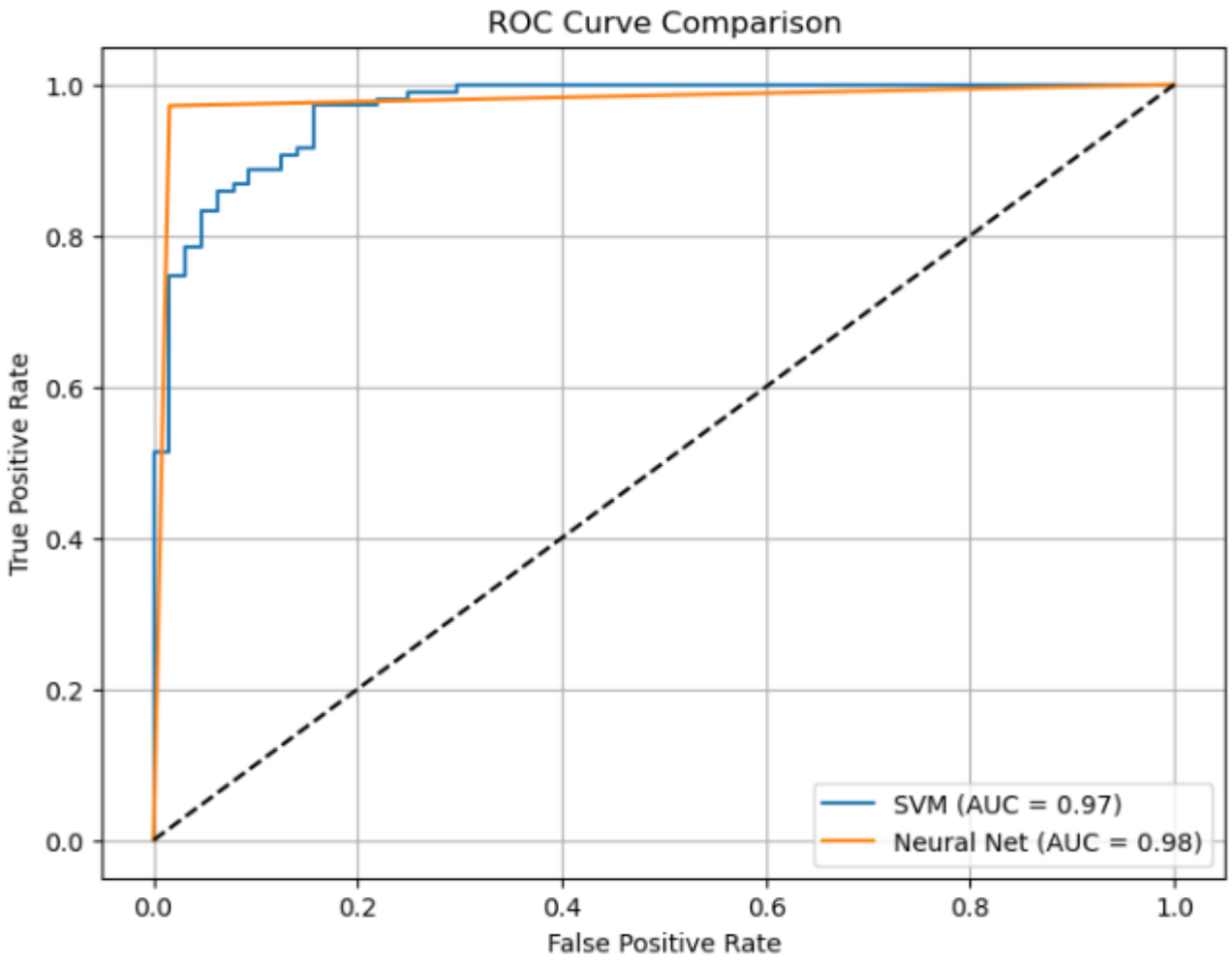


Comparison of SVM and Neural Network Performance

Model	Accuracy	Precision	Recall	F1-Score
SVM (Linear Kernel)	0.9532	0.9381	0.9907	0.9636
SVM (Poly Kernel)	0.9064	0.8889	0.9720	0.9286
SVM (RBF Kernel)	0.9064	0.8889	0.9720	0.9286
Neural Network	**0.9766**	**0.98**	**0.98**	**0.98**

Analysis and Observations

- The **Neural Network** outperformed all SVM models in terms of accuracy, precision, recall, and F1-score.
- The **SVM with a Linear Kernel** showed solid performance and was close to the neural network, while being computationally simpler.
- Both **SVM with Polynomial and RBF Kernels** performed similarly but worse than the linear kernel, which may suggest unnecessary complexity or slight overfitting.



Conclusion: Model Performance Comparison

Model	Accuracy	Precision	Recall	F1-Score
SVM (Linear Kernel)	0.9532	0.9381	0.9907	0.9636
SVM (Poly Kernel)	0.9064	0.8889	0.9720	0.9286
SVM (RBF Kernel)	0.9064	0.8889	0.9720	0.9286
Neural Network	0.9766	0.98	0.98	0.98
Logistic Regression	0.97	0.97	0.97	0.97
KNN	0.9825	0.9865	0.9865	0.9865
Cross-Validation Model	0.8857	-	-	-

Analysis and Observations

- Neural Network stands out as the best-performing model, achieving the highest accuracy (0.9766), precision (0.98), recall (0.98), and F1-score (0.98). It shows strong overall performance across all metrics.
- KNN is a close second with excellent precision, recall, and F1-score (all 0.9865), and a high test accuracy (0.9825). Its performance is very similar to the Neural Network, making it a strong contender.
- SVM (Linear Kernel) shows good performance with an accuracy of 0.9532, and its precision and recall scores are well-balanced. However, it does not outperform the neural network or KNN.
- SVM (Poly and RBF Kernels) perform similarly with an accuracy of 0.9064 and slightly lower precision, recall, and F1-score compared to the linear kernel, suggesting that more complex kernels didn't provide a significant improvement for this dataset.
- Logistic Regression performs well with an accuracy of 0.97, and its precision, recall, and F1-score are all consistent at 0.97. This is a solid model but not quite as strong as the neural network or KNN.
- The Cross-Validation Model provided a lower average accuracy (0.8857) compared to other models, which indicates that it may not be ideal for this particular dataset in terms of predictive power.
