Student exam perdiction

Import necessary libraries

Pandas:

For loading, exploring, and manipulating structured data (like .csv files). Used for .read_csv(), .head(), .describe(), etc.

numpy

Provides numerical computing support, especially for arrays and math functions. Often used under the hood by other libraries

Seaborn

Built on matplotlib; simplifies beautiful statistical plots (histograms, boxplots, heatmaps). Useful for Univariate/Bivariate Analysis

matplotlib

A plotting library for basic charts (line, bar, scatter). Used for understanding distributions and trends

Data Collection and understanding

In [137…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import pickle
import os  # To check for file existence
```

Data Collection and Understanding

In [138…
```
class DataLoader:
    def __init__(self, filepath):
        self.filepath = filepath

    def describe(self):
        return self.data.describe()

    def info(self):
        return self.data.info()

    def head(self):
        return self.data.head()

    def tail(self):
        return self.data.tail()
    def load_data(self):
        self.data = pd.read_csv(self.filepath)
        return self.data

    def show_summary(self):
        return self.data.describe()

    def show_info(self):
        return self.data.info()
```

Data Preprocessing & Analysis

In [139…
```
class DataPreprocessor:
    def __init__(self, data):
        self.data = data

    def clean_data(self):
        self.data = self.data.dropna()
        return self.data

    def rename_columns(self):
        self.data = self.data.rename(columns={
            'race/ethnicity': 'race_ethnicity',
            'parental level of education': 'parental_education',
            'test preparation course': 'prep_course',
            'math score': 'math_score',
            'reading score': 'reading_score',
            'writing score': 'writing_score'
        })
        return self.data

    def transform_gender(self):
        self.data['gender'] = self.data['gender'].map({'male': 'M', 'female': 'F'})
        return self.data

    def drop_columns(self):
        columns_to_drop = ['race_ethnicity', 'parental_education', 'lunch', 'prep_c
        self.data = self.data.drop(columns=columns_to_drop)
```

```
        return self.data

    def mean(self):
        return {
            "math_mean": self.data['math_score'].mean(),
            "reading_mean": self.data['reading_score'].mean(),
            "writing_mean": self.data['writing_score'].mean()
        }

    def mode(self):
        return {
            "math_mode": self.data['math_score'].mode().values.tolist(),
            "reading_mode": self.data['reading_score'].mode().values.tolist(),
            "writing_mode": self.data['writing_score'].mode().values.tolist()
        }

    def median(self):
        return {
            "math_median": self.data['math_score'].median(),
            "reading_median": self.data['reading_score'].median(),
            "writing_median": self.data['writing_score'].median()
        }
```

Graph

In [140…

```
class Graph:
    def __init__(self, df):
        self.df = df

    def plot_histogram(self, bins=10):
        plt.figure(figsize=(10, 6))
        plt.hist(self.df['math_score'], bins=bins, color='skyblue', edgecolor='blac
        plt.title('Math Scores Distribution', fontsize=16)
        plt.xlabel('Math Score', fontsize=14)
        plt.ylabel('Number of Students', fontsize=14)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)
        plt.tight_layout()
        plt.show()

    def plot_bar(self):
        avg_scores = self.df.groupby('gender')[['math_score', 'reading_score', 'wri
        fig, axes = plt.subplots(1, 3, figsize=(18, 8))

        axes[0].bar(avg_scores.index, avg_scores['math_score'], color='skyblue')
        axes[0].set_title('Average Math Score by Gender')
        axes[0].set_ylabel('Score')
        axes[0].set_xlabel('Gender')

        axes[1].bar(avg_scores.index, avg_scores['reading_score'], color='lightgree
        axes[1].set_title('Average Reading Score by Gender')
        axes[1].set_ylabel('Score')
        axes[1].set_xlabel('Gender')
```

```
        axes[2].bar(avg_scores.index, avg_scores['writing_score'], color='salmon')
        axes[2].set_title('Average Writing Score by Gender')
        axes[2].set_ylabel('Score')
        axes[2].set_xlabel('Gender')

        plt.tight_layout()
        plt.show()

    def plot_pie(self):
        gender_counts = self.df['gender'].value_counts()
        plt.figure(figsize=(6, 6))
        plt.pie(
            gender_counts,
            labels=gender_counts.index,
            autopct='%1.1f%%',
            startangle=90
        )
        plt.title('Gender Distribution')
        plt.axis('equal')
        plt.show()

    def plot_box(self):
        plt.figure(figsize=(6, 5))
        sns.boxplot(x='gender', y='reading_score', data=self.df)
        plt.title('Reading Score Distribution by Gender')
        plt.xlabel('Gender')
        plt.ylabel('Reading Score')
        plt.show()

    def plot_scatter(self):
        plt.figure(figsize=(6, 5))
        sns.scatterplot(x='math_score', y='reading_score', hue='gender', data=self.
        plt.title('Math vs Reading Scores by Gender')
        plt.xlabel('Math Score')
        plt.ylabel('Reading Score')
        plt.legend(title='Gender')
        plt.show()

    def plot_regression(self, x_col, y_col):
        plt.figure(figsize=(8, 6))
        sns.regplot(x=self.df[x_col], y=self.df[y_col])
        plt.title(f'Regression Plot: {x_col} vs {y_col}')
        plt.xlabel(x_col)
        plt.ylabel(y_col)
        plt.show()

    def plot_heatmap(self):
        plt.figure(figsize=(10, 8))
        sns.heatmap(self.df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
        plt.title("Correlation Heatmap")
        plt.show()
```

Univariate Analysis

In [141…
```
class Univariate(Graph):
    def __init__(self, df):
        super().__init__(df)

    def plot_histogram(self, column, bins=10):
        plt.figure(figsize=(10, 6))
        plt.hist(self.df[column], bins=bins, color='skyblue', edgecolor='black')
        plt.title(f'Distribution of {column}', fontsize=16)
        plt.xlabel(column, fontsize=14)
        plt.ylabel('Frequency', fontsize=14)
        plt.grid(True, linestyle='--', alpha=0.7)
        plt.xticks(fontsize=12)
        plt.yticks(fontsize=12)
        plt.tight_layout()
        plt.show()

    def plot_box(self, column):
        plt.figure(figsize=(6, 5))
        sns.boxplot(y=self.df[column])
        plt.title(f'Box Plot of {column}')
        plt.ylabel(column)
        plt.show()

    def plot_kde(self, column):
        plt.figure(figsize=(8, 6))
        sns.kdeplot(self.df[column], fill=True)
        plt.title(f'Kernel Density Estimation of {column}')
        plt.xlabel(column)
        plt.ylabel('Density')
        plt.show()
```

Bivariate Analysis

In [142…
```
class Bivariate(Graph):
    def __init__(self, df):
        super().__init__(df)

    def plot_scatter(self, x_col, y_col, hue=None):
        plt.figure(figsize=(8, 6))
        sns.scatterplot(x=self.df[x_col], y=self.df[y_col], hue=self.df[hue] if hue
        plt.title(f'Scatter Plot: {x_col} vs {y_col}')
        plt.xlabel(x_col)
        plt.ylabel(y_col)
        plt.legend(title=hue)
        plt.show()

    def plot_regression(self, x_col, y_col):
        plt.figure(figsize=(8, 6))
        sns.regplot(x=self.df[x_col], y=self.df[y_col])
        plt.title(f'Regression Plot: {x_col} vs {y_col}')
        plt.xlabel(x_col)
        plt.ylabel(y_col)
        plt.show()
```

```
    def plot_heatmap(self):
        plt.figure(figsize=(10, 8))
        sns.heatmap(self.df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
        plt.title("Correlation Heatmap")
        plt.show()
```

Data Spilting

In [143…

```
class DataSplitter:

    def __init__(self, X, y):

        self.X = X
        self.y = y

    def split(self, test_size=0.2, random_state=42):

        print(f"Splitting data with test_size={test_size}, random_state={random_sta
        X_train, X_test, y_train, y_test = train_test_split( self.X, self.y, test_s
        print(f"X_train shape: {X_train.shape}, X_test shape: {X_test.shape}")
        print(f"y_train shape: {y_train.shape}, y_test shape: {y_test.shape}")
        return X_train, X_test, y_train, y_test
```

Model Training

In [144…

```
class ModelTrainer:
    def __init__(self, kernel='linear'):
        from sklearn.svm import SVC
        self.model = SVC(kernel=kernel)

    def train(self, X_train, y_train):
        self.model.fit(X_train, y_train)
        print("Model training complete.")
        return self.model

    def evaluate(self, X_test, y_test):
        y_pred = self.model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        print(f"Model Accuracy: {acc:.4f}")

        cm = confusion_matrix(y_test, y_pred)
        print("\nConfusion Matrix:")
        print(cm)

        # Plot confusion matrix
        plt.figure(figsize=(6,5))
        sns.heatmap(cm, annot=True, fmt="d", cmap='Blues', cbar=False)
        plt.xlabel('Predicted')
        plt.ylabel('Actual')
        plt.title('Confusion Matrix')
        plt.show()
```

```
        return acc, cm
```

Model Storage

In [145…

```
class ModelStorage:
    @staticmethod
    def save_model(model, filename='svm_model.pkl'):
        try:
            with open(filename, 'wb') as f:
                pickle.dump(model, f)
            print(f"Model saved successfully to {filename}")
        except Exception as e:
            print(f"Error saving model: {e}")

    @staticmethod
    def load_model(filename='svm_model.pkl'):
        if not os.path.exists(filename):
            print(f"Error: Model file not found at {filename}")
            return None
        try:
            with open(filename, 'rb') as f:
                model = pickle.load(f)
            print(f"Model loaded successfully from {filename}")
            return model
        except Exception as e:
            print(f"Error loading model: {e}")
            return None
```

Main Excuetion

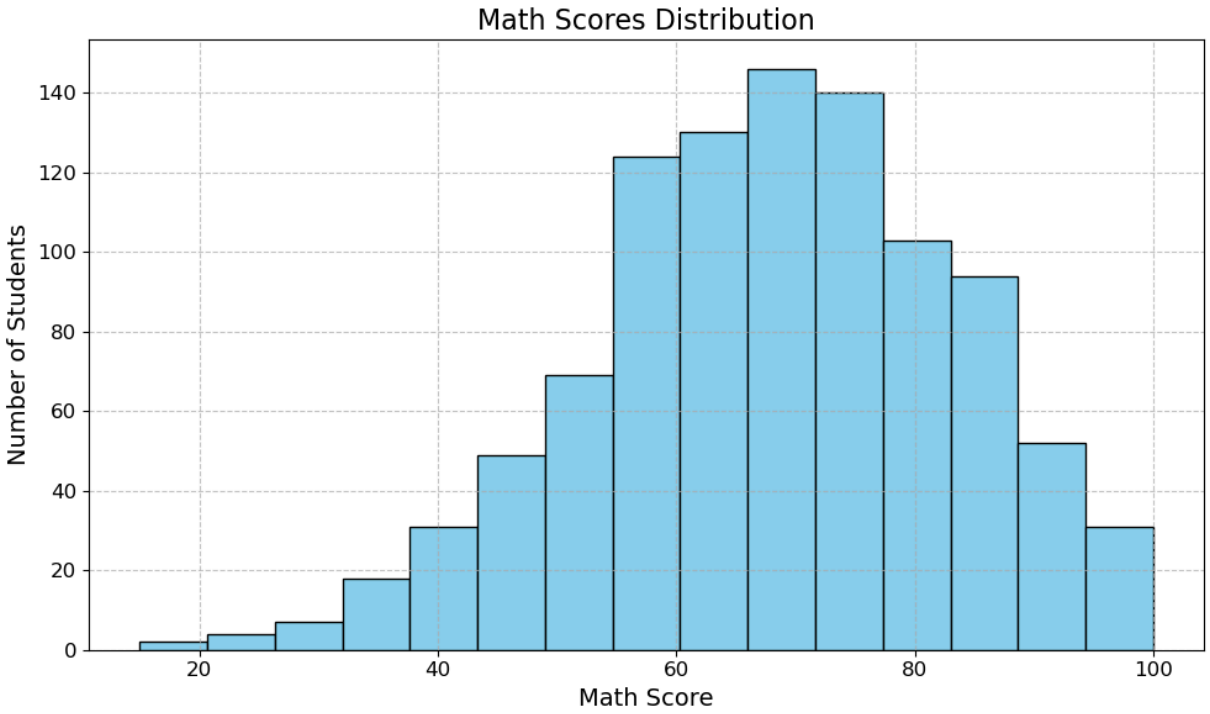
In [146…

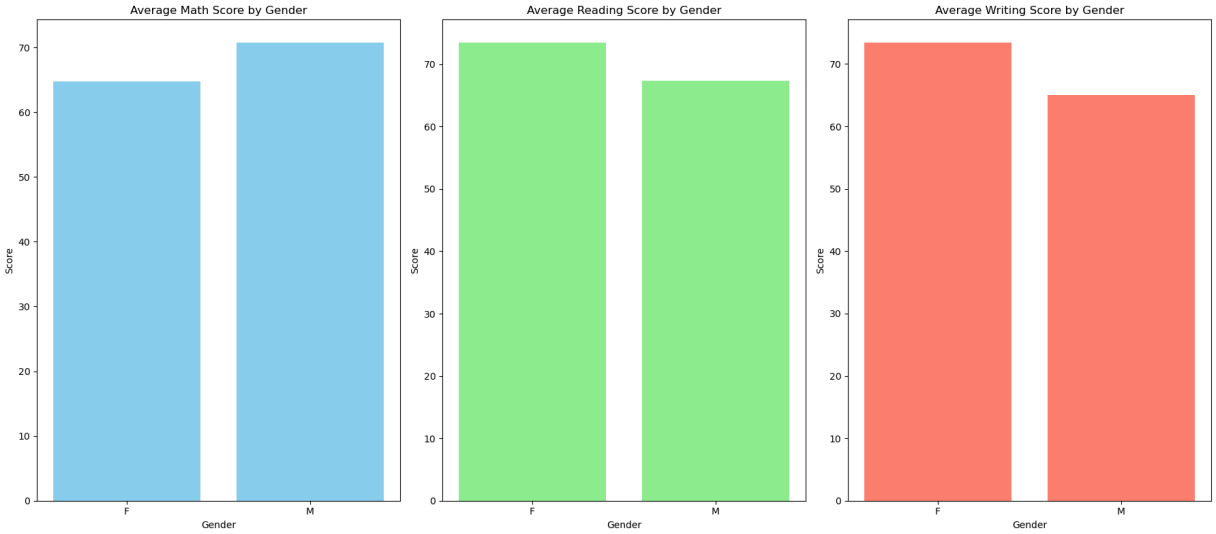
```
if __name__ == "__main__":
    data_file_path = "exams.csv"

    try:
        print("--- Loading Data ---")
        loader = DataLoader(filepath=data_file_path)
        raw_data = loader.load_data()

        if raw_data is None:
            print("Data loading failed. Exiting.")
        else:
            print("Data loaded successfully.")


            print("\n--- Preprocessing Data ---")
            preprocessor = DataPreprocessor(raw_data.copy())
            preprocessor.rename_columns()
            preprocessor.transform_gender()
            processed_data = preprocessor.data

            if 'gender' in processed_data.columns:
                processed_data['gender_encoded'] = processed_data['gender'].map({'M
```

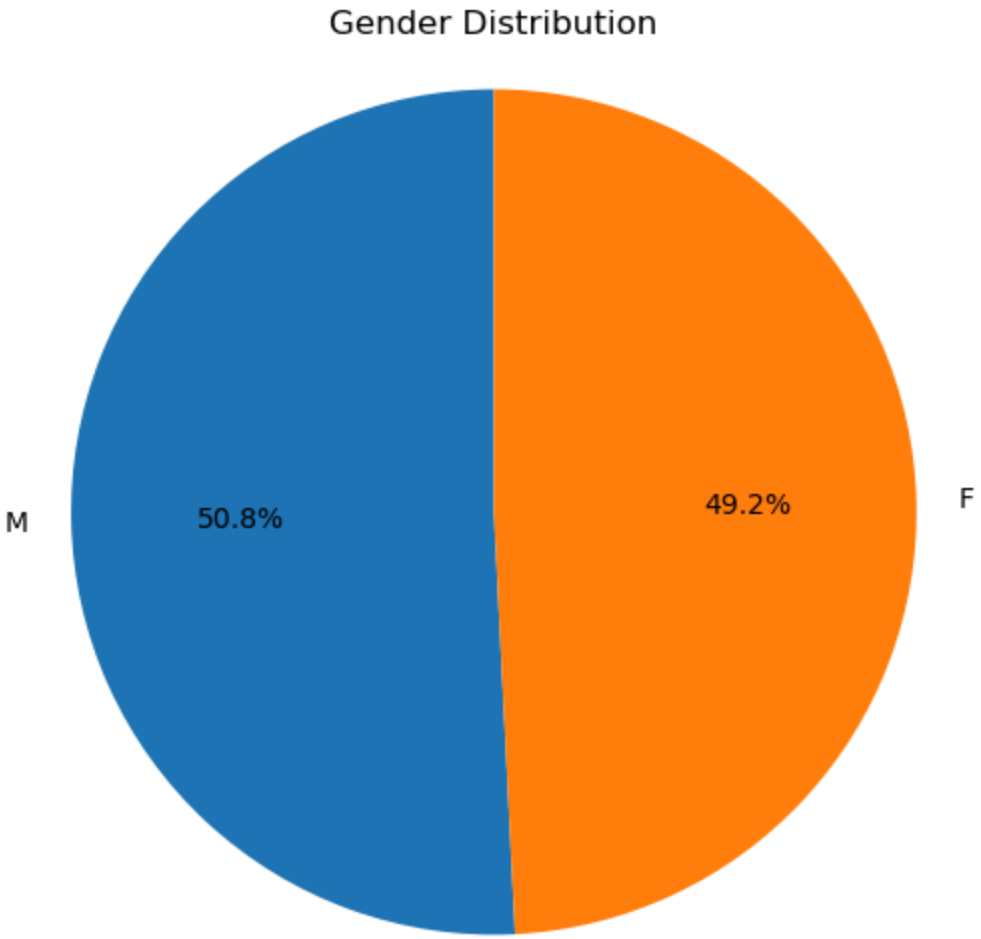
```
        else:
            raise KeyError("'gender' column not found in data.")

        required_cols = ['gender', 'math_score', 'reading_score', 'writing_scor
        if all(col in processed_data.columns for col in required_cols):
            graph_visualizer = Graph(df=processed_data)
            graph_visualizer.plot_histogram(bins=15)
            graph_visualizer.plot_bar()
            graph_visualizer.plot_pie()
            graph_visualizer.plot_box()
            graph_visualizer.plot_scatter()

            univariate_analyzer = Univariate(df=processed_data)
            univariate_analyzer.plot_histogram(column='reading_score', bins=20)
            univariate_analyzer.plot_box(column='writing_score')
            univariate_analyzer.plot_kde(column='math_score')

            bivariate_analyzer = Bivariate(df=processed_data)
            bivariate_analyzer.plot_scatter(x_col='math_score', y_col='writing_
            bivariate_analyzer.plot_regression(x_col='reading_score', y_col='wr

            numeric_cols_df = processed_data.select_dtypes(include=np.number)
            if not numeric_cols_df.empty:
                bivariate_numeric_analyzer = Bivariate(df=numeric_cols_df)
                bivariate_numeric_analyzer.plot_heatmap()
        else:
            print("Skipping visualization: some required columns missing.")

        print("\n--- Starting Model Training ---")


        X = processed_data[['math_score', 'reading_score', 'writing_score']]
        y = processed_data['gender_encoded']


        splitter = DataSplitter(X, y)
        X_train, X_test, y_train, y_test = splitter.split(test_size=0.2, random

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


        trainer = ModelTrainer(kernel='linear')
        model = trainer.train(X_train_scaled, y_train)

        print("\n--- Evaluating Model ---")
        y_pred = model.predict(X_test_scaled)

        acc = accuracy_score(y_test, y_pred)
        print(f"Accuracy: {acc:.4f}")

        print("\nConfusion Matrix:")
        print(confusion_matrix(y_test, y_pred))

        print("\nClassification Report:")
```

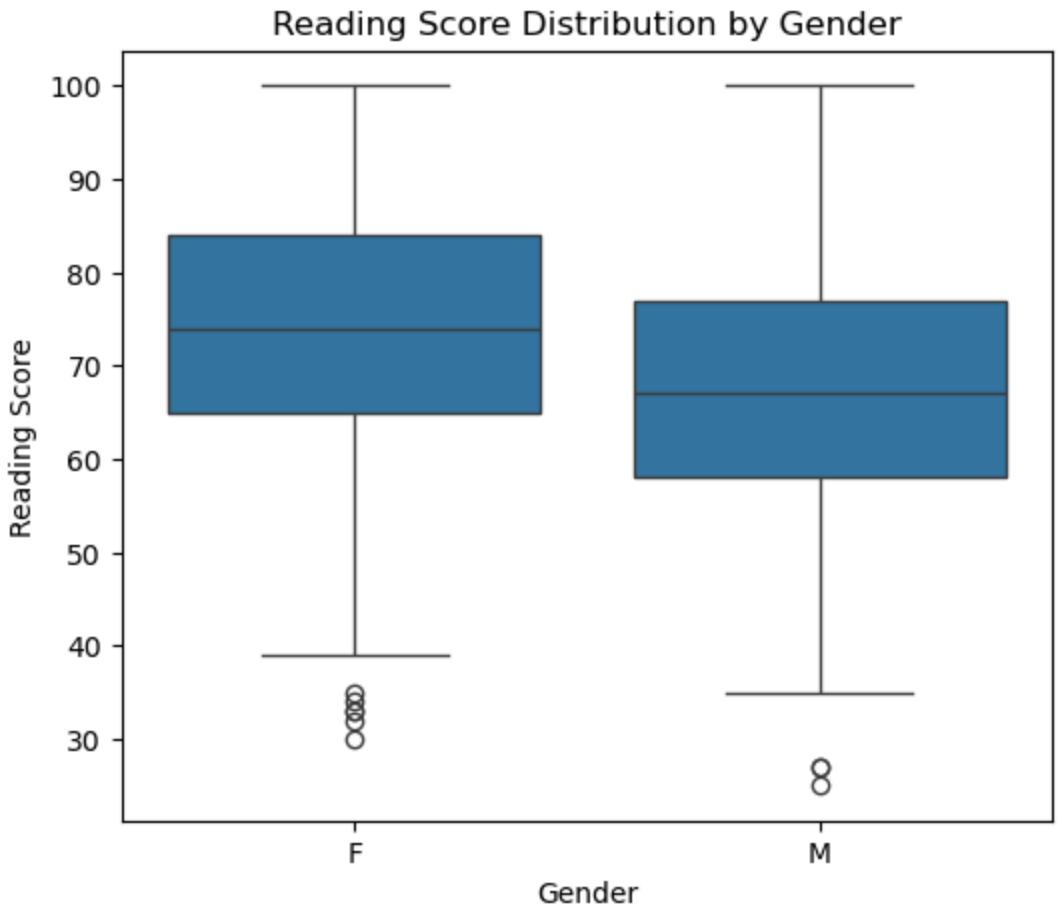
```
            print(classification_report(y_test, y_pred))

            # Save model
            ModelStorage.save_model(model,"student_exam_svc_model.pkl" )
            print(f"\nModel saved to {"student_exam_svc_model.pkl"}")

            # Load and test model
            loaded_model = ModelStorage.load_model("student_exam_svc_model.pkl")
            if loaded_model:
                sample_preds = loaded_model.predict(X_test_scaled[:5])
                print(f"\nSample Predictions: {sample_preds}")
                print(f"Actual Values: {y_test.iloc[:5].values}")

    except FileNotFoundError:
        print(f"File not found: {data_file_path}")
    except pd.errors.EmptyDataError:
        print(f"Data file is empty: {data_file_path}")
    except KeyError as e:
        print(f"Missing key in data: {e}")
    except Exception as e:
        print(f"Unexpected error: {e}")
```

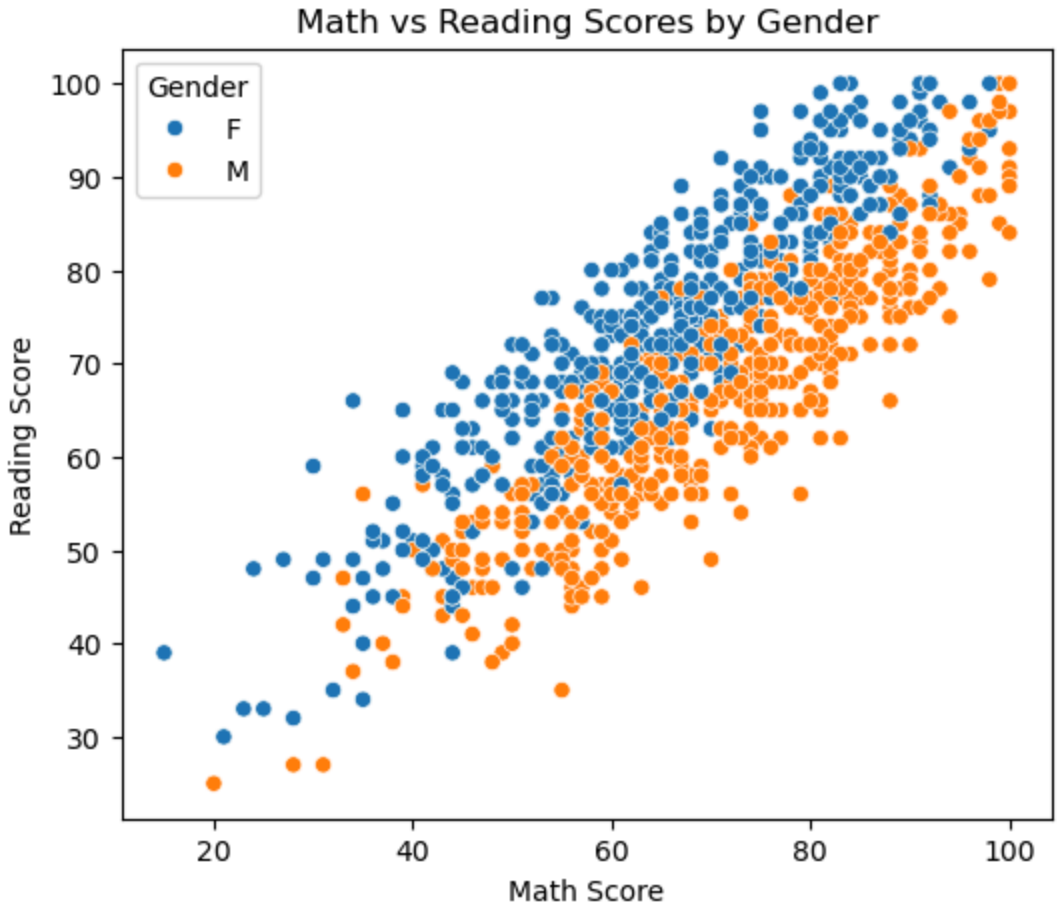
```
--- Loading Data ---
Data loaded successfully.

--- Preprocessing Data ---
```

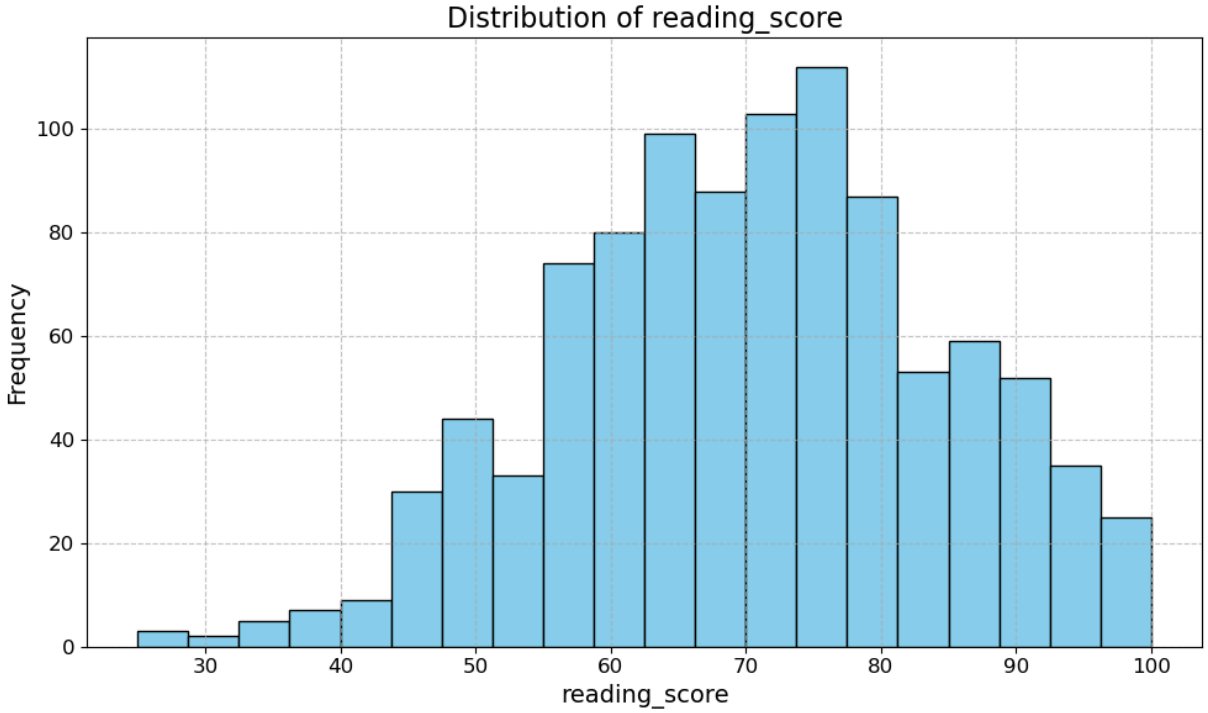


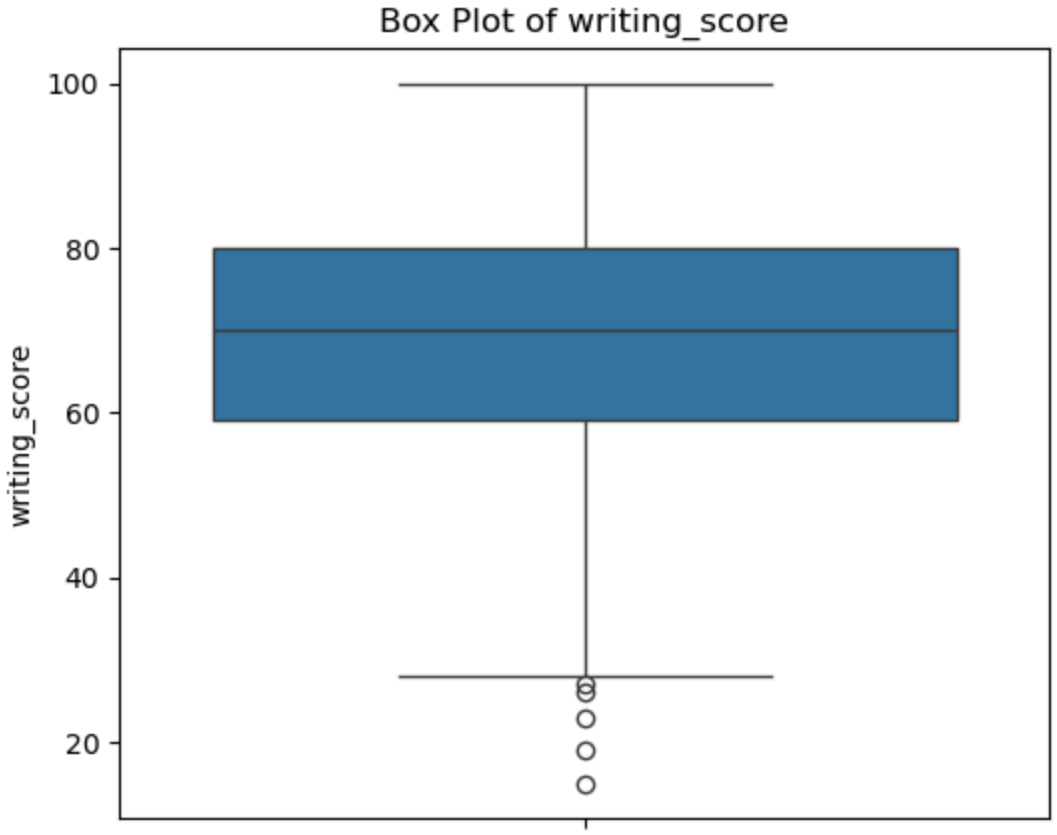

Average Math Score by Gender
Average Reading Score by Gender
Average Writing Score by Gender
Score
Gender
F
M
0
10
20
30
40
50
60
70

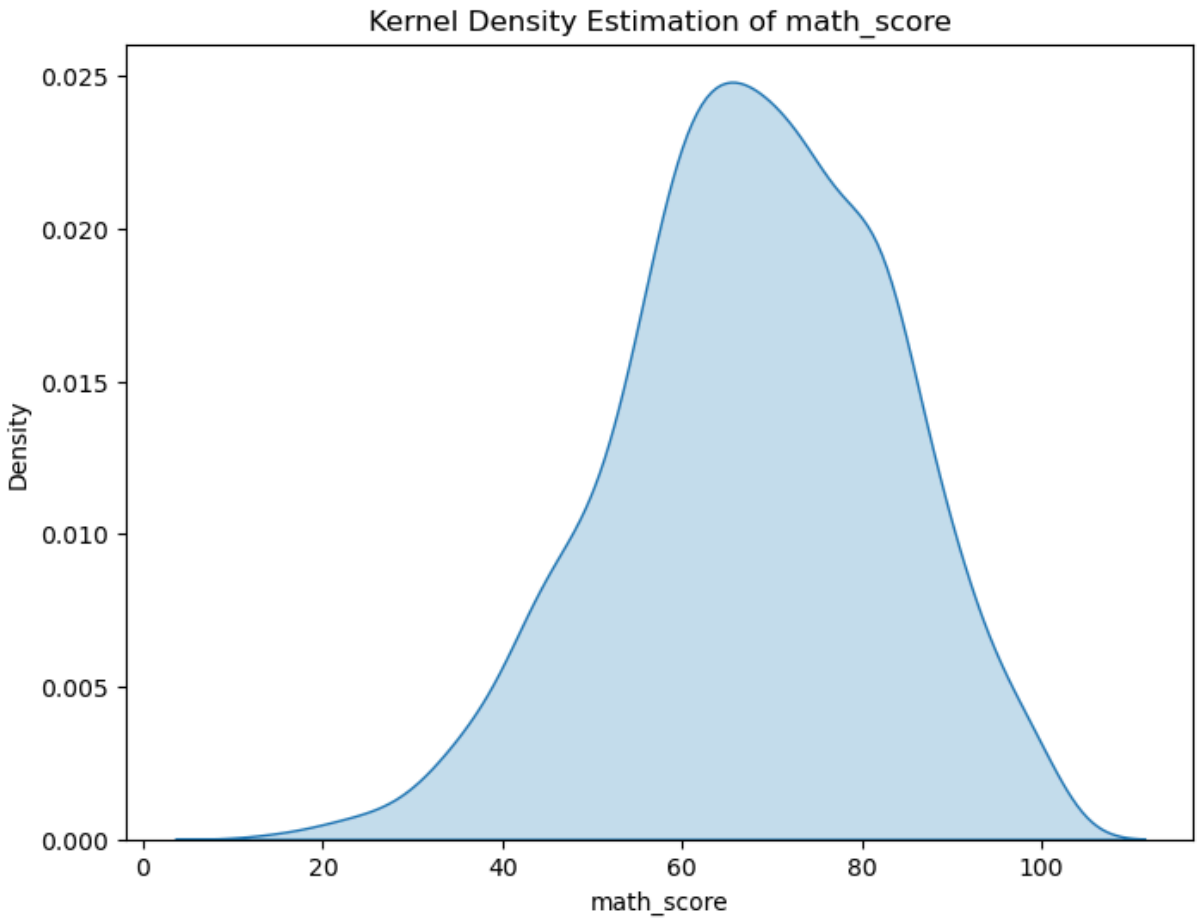

Gender Distribution
M
F
50.8%
49.2%

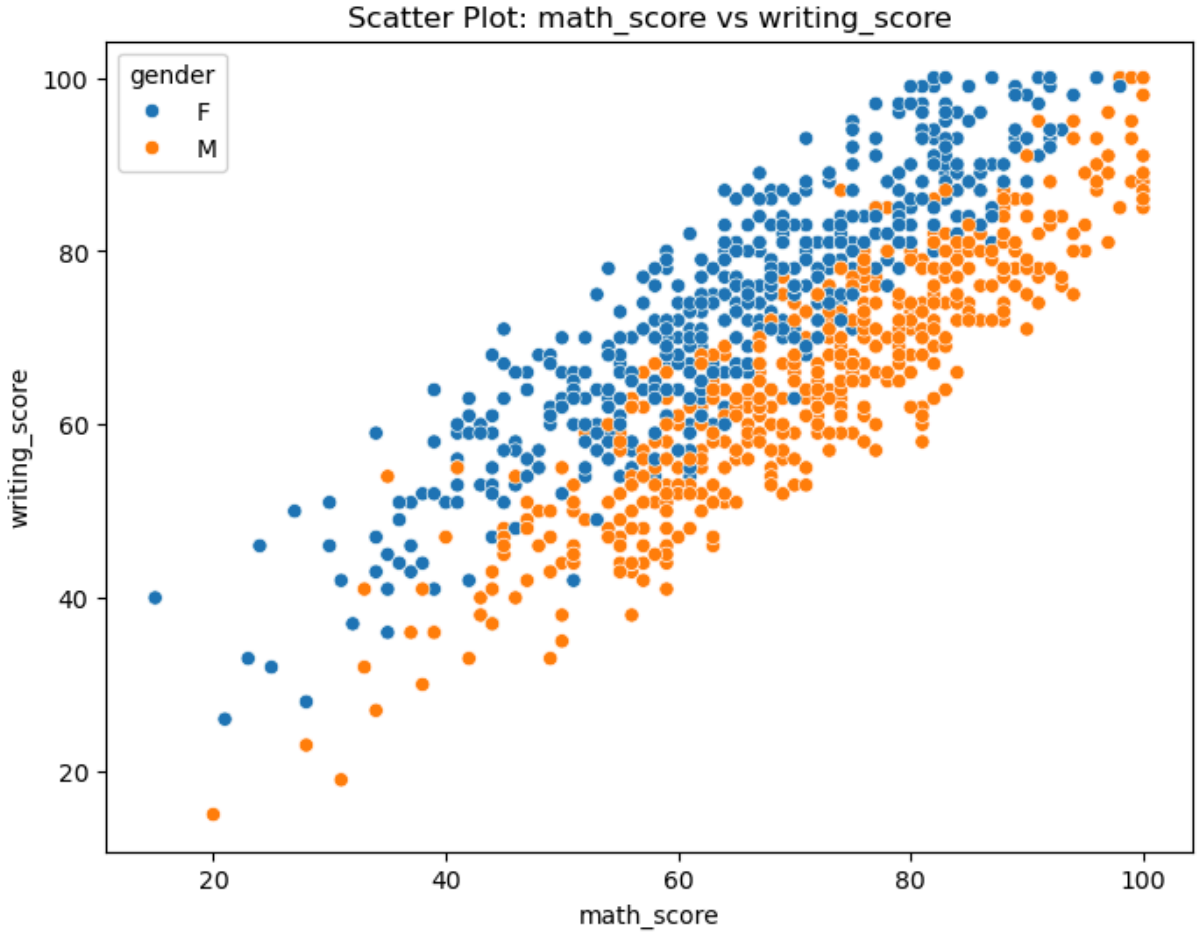

Reading Score Distribution by Gender
Reading Score
100
90
80
70
60
50
40
30
F
M
Gender

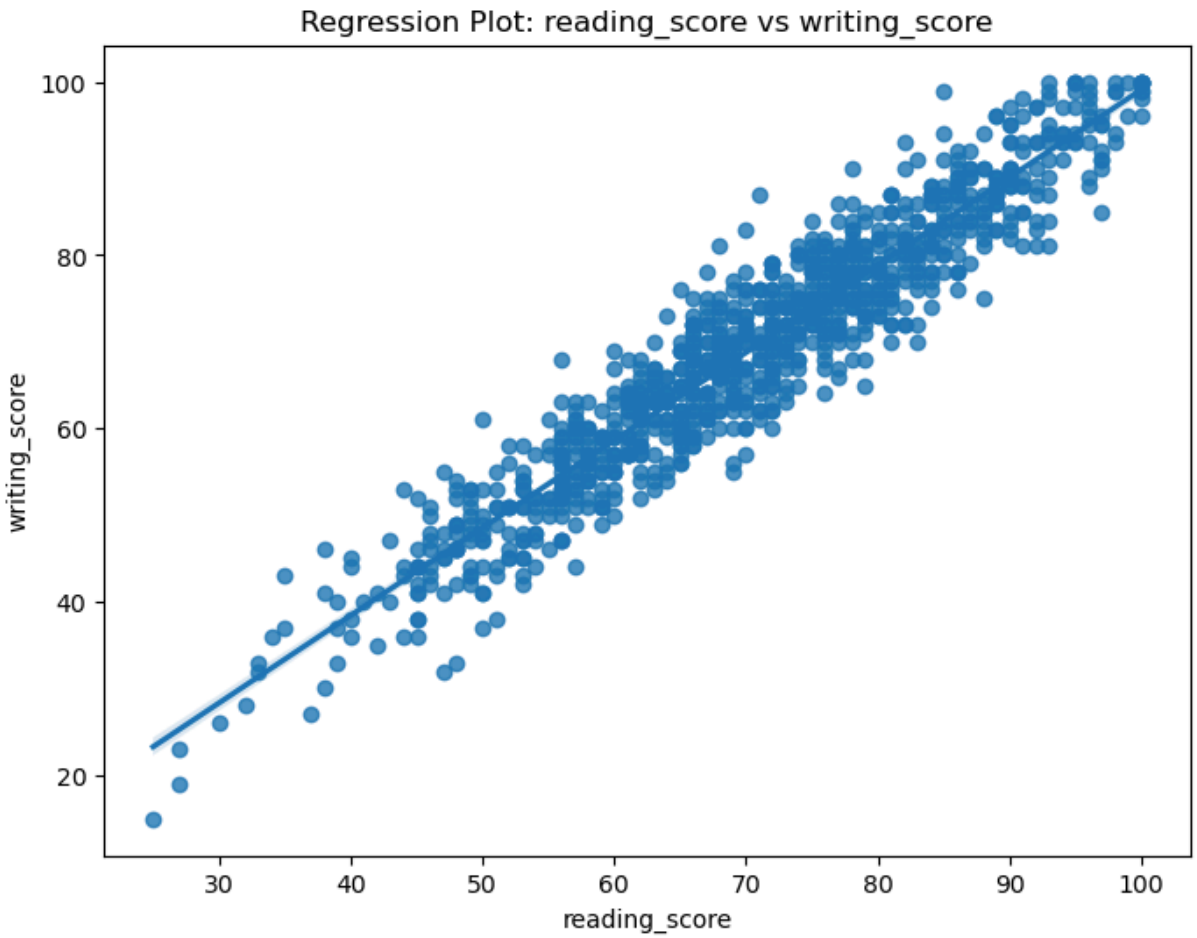

Math vs Reading Scores by Gender
Gender
F
M
Reading Score
100
90
80
70
60
50
40
30
20
40
60
80
100
Math Score

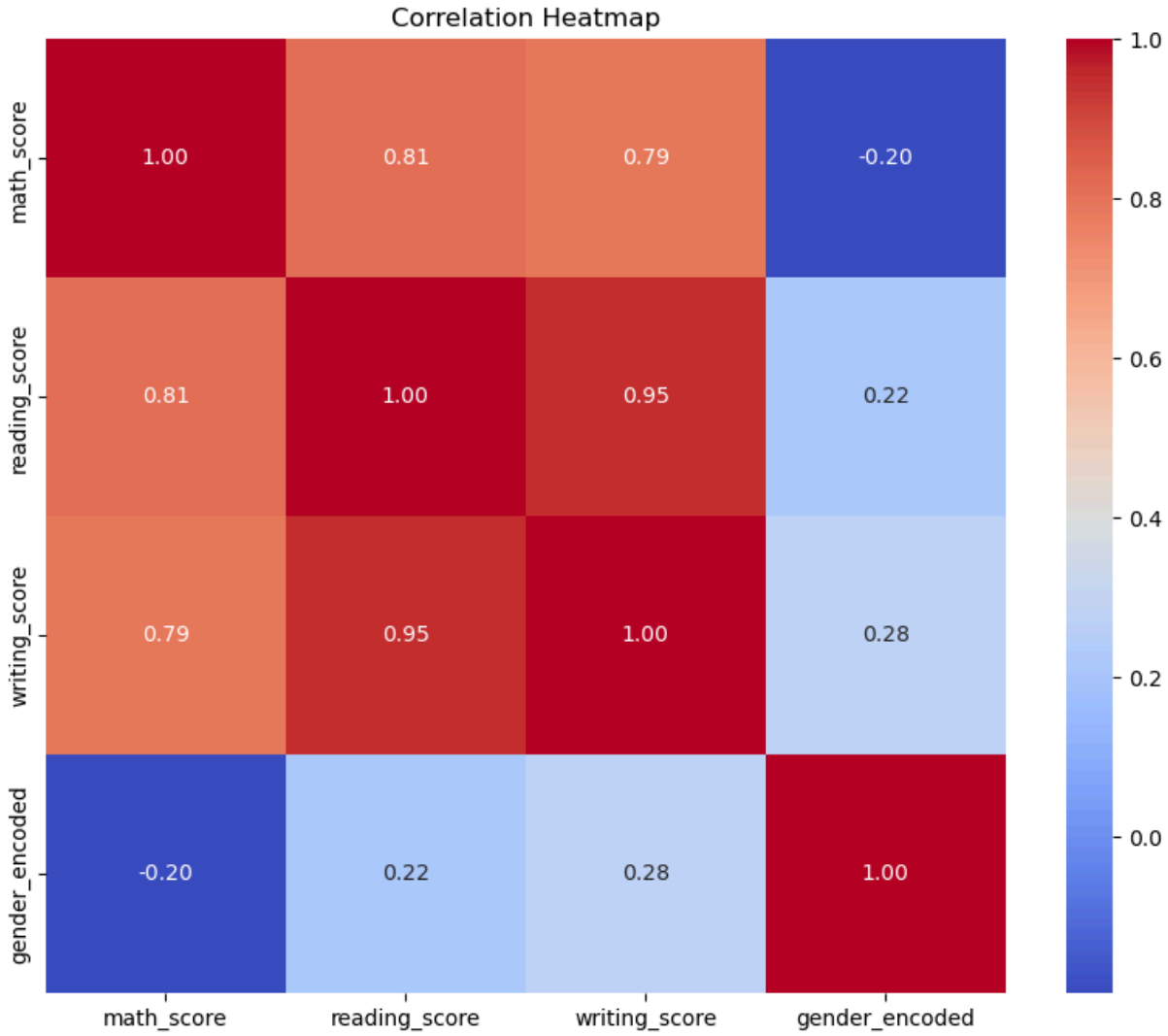

Distribution of reading_score
Frequency
reading_score
0
20
40
60
80
100
30
40
50
60
70
80
90
100


Box Plot of writing_score
writing_score
20
40
60
80
100


Kernel Density Estimation of math_score
Density
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
math_score


Scatter Plot: math_score vs writing_score
gender
F
M
writing_score
100
80
60
40
20
20
40
60
80
100
math_score


Regression Plot: reading_score vs writing_score
writing_score
reading_score
100
80
60
40
20
30
40
50
60
70
80
90
100


Correlation Heatmap
math_score
reading_score
writing_score
gender_encoded
1.00
0.81
0.79
-0.20
0.81
1.00
0.95
0.22
0.79
0.95
1.00
0.28
-0.20
0.22
0.28
1.00
math_score
reading_score
writing_score
gender_encoded
1.0
0.8
0.6
0.4
0.2
0.0

```
--- Starting Model Training ---
Splitting data with test_size=0.2, random_state=42
X_train shape: (800, 3), X_test shape: (200, 3)
y_train shape: (800,), y_test shape: (200,)
Model training complete.

--- Evaluating Model ---
Accuracy: 0.9100

Confusion Matrix:
[[93 12]
 [ 6 89]]

Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.89      0.91       105
           1       0.88      0.94      0.91        95

    accuracy                           0.91       200
   macro avg       0.91      0.91      0.91       200
weighted avg       0.91      0.91      0.91       200

Model saved successfully to student_exam_svc_model.pkl

Model saved to student_exam_svc_model.pkl
Model loaded successfully from student_exam_svc_model.pkl

Sample Predictions: [1 1 1 0 0]
Actual Values: [0 0 1 0 0]
```

In [ ]: