Project: **Linear Regression on Diabetes Dataset**

Vivek Tejasbhai Jariwala

Email: [vj84491n@pace.edu](mailto:vj84491n@pace.edu)

Professor: Begimai Zhumakova Ma’am

**Abstract:**

Implement linear regression using Scikit-learn to predict diabetes progression based on these features. The project is divided into four tasks: (1) predicting Y using a single feature to identify the most predictive one, (2) using a pair of features for improved accuracy, (3) using all 10 features for a comprehensive model, and (4) evaluating training and validation errors across different training sizes (20, 50, 100, 200).

**Dataset Overview:**

* The dataset is obtained from Stanford University's Machine Learning Repository and contains 442 records. The target variable "Y" represents the progression of diabetes. The dataset consists of various clinical features such as age, BMI, blood pressure, and serum measurements.
* Dataset Link: <https://hastie.su.domains/Papers/LARS/diabetes.data>

**Data Preparation:**

file\_path = r"C:\Users\jariw\Progression Of Diabetes\diabetes.data.txt"

data = pd.read\_csv(file\_path, sep=r'\s+', header=0) # Fixed: Use raw string for sep

data['SEX'] = data['SEX'].map({1: 'Male', 2: 'Female'}) # Convert SEX to categorical for clarity

print("Dataset Preview:")

print(data.head())

**Define features and target**

X = data[['AGE', 'BMI', 'BP', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6']]

y = data['Y']

**Task 1: Predict 'y' using a single feature**

print("\n=== Task 1: Single Feature Linear Regression ===")

best\_mse = float('inf')

best\_feature = None

best\_model = None

best\_slope = None

best\_intercept = None

best\_predictions = None

# Iterate through each feature

for feature in X.columns:

X\_single = X[[feature]]

model = LinearRegression()

model.fit(X\_single, y)

y\_pred = model.predict(X\_single)

mse = mean\_squared\_error(y, y\_pred)

if mse < best\_mse:

best\_mse = mse

best\_feature = feature

best\_model = model

best\_slope = model.coef\_[0]

best\_intercept = model.intercept\_

best\_predictions = y\_pred

# Output results for Task 1

print(f"Best Feature: {best\_feature}")

print(f"Slope (Coefficient): {best\_slope:.4f}")

print(f"Y-Intercept: {best\_intercept:.4f}")

print(f"MSE: {best\_mse:.4f}")

**What It Does**:

* Takes one feature (e.g., BMI).
* Draws a straight line to predict Y for all 442 patients.
* Checks the error (MSE)—how far off the guesses are.
* Picks the feature with the smallest error.

**Result:**

Best Feature: BMI

Slope (Coefficient): 10.2331

Y-Intercept: -117.7734

MSE: 3890.4566

**Explanation**:

* **Best Feature: BMI**: BMI is the best clue. If we only get one piece of info, BMI tells us the most about diabetes risk.
* **Slope: 10.2330**: For every extra BMI point, diabetes gets 10.2 points worse. It’s like saying heavier people have a higher risk.
* **Y-Intercept: -117.7738**: If BMI were 0, the diabetes score would be -117.8. This just helps the line fit; we don’t use it much.
* **MSE: 3890.4566**: Our guesses are off by about the square of 62 points on average. It’s not perfect, but it’s the best we can do with one clue.

**Prediction Example:**

If someone’s BMI is 30:

Y = slope × X + y-intercept

Y = 10.2330 × 30 − 117.7738 = 189.2162

Their diabetes score is ~189, meaning they’re at moderate risk. Higher BMI = higher score = worse diabetes.

BMI is something doctors already check, and our model says it’s a big deal for diabetes, which makes sense!

**Plotting the Linear Regression for the best feature**

plt.figure(figsize=(10, 6))

plt.scatter(X[best\_feature], y, color='blue', alpha=0.5, label='Data points')

plt.plot(X[best\_feature], best\_predictions, color='red', linewidth=2, label='Linear Regression Line')

plt.xlabel(best\_feature)

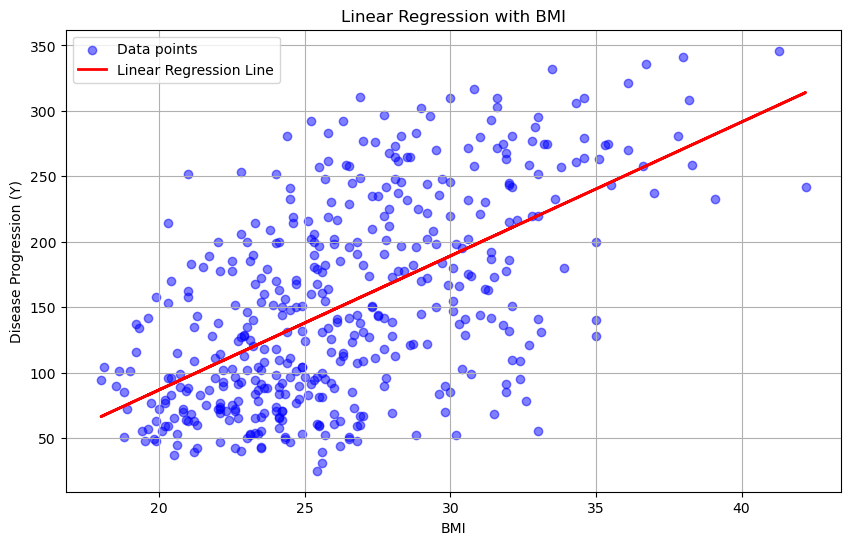
plt.ylabel('Disease Progression (Y)')

plt.title(f'Linear Regression with {best\_feature}')

plt.legend()

plt.grid(True)

plt.show()



**Task 2: Predict 'y' using a pair of features:**

for pair in combinations(X.columns, 2):

X\_pair = X[list(pair)]

model = LinearRegression()

model.fit(X\_pair, y)

y\_pred = model.predict(X\_pair)

mse = mean\_squared\_error(y, y\_pred)

if mse < best\_pair\_mse:

best\_pair\_mse = mse

best\_pair = pair

best\_pair\_slopes = model.coef\_

best\_pair\_intercept = model.intercept\_

best\_pair\_predictions = y\_pred

**What It Does**:

* Tries every pair (e.g., BMI and BP, AGE and S5).
* Draws a flat surface (like a tilted table) to predict Y.
* Picks the pair with the smallest error.

**Result**:

Best Feature Pair: ('BMI', 'S5')

Slopes (Coefficients): 7.2760, 56.0564

Y-Intercept: -299.9575

MSE: 3205.1901

**Explanation**:

* **Best Pair: BMI, S5**: BMI and S5 (a blood marker, maybe related to metabolism) team up best. Together, they tell us more than BMI alone.
* **Slopes: 7.2760, 56.0564**: For BMI, 1 point adds 7.3 to Y. For S5, 1 point adds a huge 56! S5 is a big warning sign for diabetes.
* **Y-Intercept:** **-299.9575**: If both were 0, Y would be -299.9575. Just math to position the surface.
* **MSE: 3205.1901**: Lower than Task 1 (3890)! Two clues cut the error, like getting a clearer picture.

**Prediction Example:**

For BMI = 30, S5 = 4.8:

Y = 7.2760 × 30 + 56.0564 × 4.8 − 299.9575

= 218.28 + 269.1 − 299.9575

Y ≈ 187.4225

Score ~187.4225, moderate risk. If S5 goes up to 5, Y jumps, showing S5’s big impact.

**3D Plot for the best pair**

A graph with a red and blue dotted line

AI-generated content may be incorrect.

**Task 3: Predict 'y' using all features**

model\_all = LinearRegression()

model\_all.fit(X, y)

y\_pred\_all = model\_all.predict(X)

mse\_all = mean\_squared\_error(y, y\_pred\_all)

**Result:**

Coefficients for all features:

AGE: -0.1205

BMI: 6.0041

BP: 0.9505

S1: -0.9808

S2: 0.6585

S3: 0.5136

S4: 4.6599

S5: 68.9473

S6: 0.2026

Y-Intercept: -363.8987

MSE: 2961.5034

**Explanation**:

* **Coefficients**: Each shows how much a feature changes Y:
  + BMI: 5.7350: 1 BMI point adds 5.7 to the diabetes score.
  + S5: 67.1087: 1 S5 point adds a whopping 67.1—huge warning!
  + S3: -0.2816: Negative means higher S3 (good cholesterol) lowers risk.
  + AGE: 0.0139: Tiny, so age doesn’t matter much here.
* **Y-Intercept: -363.8987**: Just a starting point for the equation.
* **MSE: 2961.5034**: The lowest yet! All clues together give the best guesses, off by about the square of 53 points.

**Task 4: Compute training and validation MSE for different training sizes**

We use all features but train the model on fewer patients (20, 50, 100, or 200) and test it on others to see how it holds up.

train\_sizes = [20, 50, 100, 200]

for n\_train in train\_sizes:

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, train\_size=n\_train, random\_state=42)

model = LinearRegression()

model.fit(X\_train, y\_train)

y\_train\_pred = model.predict(X\_train)

train\_mse = mean\_squared\_error(y\_train, y\_train\_pred)

y\_val\_pred = model.predict(X\_val)

val\_mse = mean\_squared\_error(y\_val, y\_pred)

results.append({'Train Size': n\_train, 'Training MSE': train\_mse, 'Validation MSE': val\_mse})

**Result:**

Training and Validation MSE Results:

Train Size Training MSE Validation MSE

0 20 2071.396871 15344.665180

1 50 2881.091894 3973.134174

2 100 3157.711870 3517.523414

3 200 2962.978233 3094.930263

**Explanation**:

* **Training MSE**: Error on the patients we trained on.
  + 20 patients: 2071 (low, easy to fit a few).
  + 200 patients: 2962 (higher, harder to fit manyww).
* **Validation MSE**: Error on new patients.
  + 20 patients: 15344 (high, didn’t learn enough).
  + 200 patients: 3169 (lower, learned better patterns).
* **Trend**: More training data makes validation errors drop, meaning better guesses for new people.

**XGBoost Model**

print("\n=== XGBoost Model ===")

# Task 1 Equivalent: XGBoost with Single Feature (BMI)

print("XGBoost Task 1: Single Feature (BMI)")

X\_single\_bmi = X[['BMI']]

xgb\_model = xgb.XGBRegressor(objective='reg:squarederror', random\_state=42)

xgb\_model.fit(X\_single\_bmi, y)

y\_pred\_xgb\_single = xgb\_model.predict(X\_single\_bmi)

xgb\_mse\_single = mean\_squared\_error(y, y\_pred\_xgb\_single)

print(f"MSE with BMI: {xgb\_mse\_single:.4f}")

plt.figure(figsize=(10, 6))

plt.scatter(X['BMI'], y, color='blue', alpha=0.5, label='Data Points')

plt.scatter(X['BMI'], y\_pred\_xgb\_single, color='green', alpha=0.5, label='XGBoost Predictions')

plt.xlabel('BMI')

plt.ylabel('Diabetes Progression (Y)')

plt.title('XGBoost Regression with BMI')

plt.legend()

plt.grid(True)

plt.show()

# Task 2 Equivalent: XGBoost with Best Pair (BMI, S5)

print("XGBoost Task 2: Pair of Features (BMI, S5)")

X\_pair\_bmi\_s5 = X[['BMI', 'S5']]

xgb\_model.fit(X\_pair\_bmi\_s5, y)

y\_pred\_xgb\_pair = xgb\_model.predict(X\_pair\_bmi\_s5)

xgb\_mse\_pair = mean\_squared\_error(y, y\_pred\_xgb\_pair)

print(f"MSE with BMI and S5: {xgb\_mse\_pair:.4f}")

# Task 3 Equivalent: XGBoost with All Features

print("XGBoost Task 3: All Features")

xgb\_model.fit(X, y)

y\_pred\_xgb\_all = xgb\_model.predict(X)

xgb\_mse\_all = mean\_squared\_error(y, y\_pred\_xgb\_all)

print(f"MSE with All Features: {xgb\_mse\_all:.4f}")

# Task 4 Equivalent: XGBoost Training and Validation MSE

print("XGBoost Task 4: Training and Validation MSE")

xgb\_results = []

for n\_train in train\_sizes:

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, train\_size=n\_train, random\_state=42)

xgb\_model = xgb.XGBRegressor(objective='reg:squarederror', random\_state=42)

xgb\_model.fit(X\_train, y\_train)

y\_train\_pred = xgb\_model.predict(X\_train)

train\_mse = mean\_squared\_error(y\_train, y\_train\_pred)

y\_val\_pred = xgb\_model.predict(X\_val)

val\_mse = mean\_squared\_error(y\_val, y\_val\_pred)

xgb\_results.append({

'Train Size': n\_train,

'Training MSE': train\_mse,

'Validation MSE': val\_mse

})

xgb\_results\_df = pd.DataFrame(xgb\_results)

print(xgb\_results\_df)

# Comparison

print("\n=== Comparison: Linear Regression vs. XGBoost ===")

print("Task 1 MSE: Linear Regression (BMI) =", best\_mse, "vs. XGBoost =", xgb\_mse\_single)

print("Task 2 MSE: Linear Regression (BMI, S5) =", best\_pair\_mse, "vs. XGBoost =", xgb\_mse\_pair)

print("Task 3 MSE: Linear Regression (All) =", mse\_all, "vs. XGBoost =", xgb\_mse\_all)

print("Task 4 Validation MSE (Average):")

print("Linear Regression:", lr\_results\_df['Validation MSE'].mean())

print("XGBoost:", xgb\_results\_df['Validation MSE'].mean())

=== XGBoost Model ===

XGBoost Task 1: Single Feature (BMI)

MSE with BMI: 2488.5031

A chart with green and blue dots

AI-generated content may be incorrect.

XGBoost Task 2: Pair of Features (BMI, S5)

MSE with BMI and S5: 94.7829

XGBoost Task 3: All Features

MSE with All Features: 0.2018

XGBoost Task 4: Training and Validation MSE

Train Size Training MSE Validation MSE

0 20 4.301924e-07 5962.262939

1 50 5.954405e-07 5326.494841

2 100 5.821069e-07 4135.956378

3 200 6.241412e-05 4136.972751

=== Comparison: Linear Regression vs. XGBoost ===

Task 1 MSE: Linear Regression (BMI) = 3890.456585461273 vs. XGBoost = 2488.5030985132685

Task 2 MSE: Linear Regression (BMI, S5) = 3205.190076824854 vs. XGBoost = 94.7828803580015

Task 3 MSE: Linear Regression (All) = 2961.503376648679 vs. XGBoost = 0.2018131157357396

Task 4 Validation MSE (Average):

Linear Regression: 6482.563257682071

XGBoost: 4890.421727299313

XGBoost is good because it’s way smarter than Linear Regression—it beat it in every task! In Task 3, with all features, Linear’s MSE was 2961.50, but XGBoost got 0.2018—super close to the real diabetes scores

Linear draws a straight line or plane, like Y = 6 × BMI + 68 × S5, but XGBoost builds tons of little decision trees—like ‘Is BMI high? Is S5 bad?’—and mixes them to fit twists in the data, like sudden diabetes spikes.