Product Requirements Document: Linear Regression Coefficient Estimation Using Dot Product

1. Executive Summary

This document outlines the requirements for developing a Python application that generates synthetic linear regression data and estimates regression coefficients using least squares formulas implemented with **dot product operations** for vectorized calculations.

2. Product Overview

2.1 Purpose

Create a statistical analysis application that demonstrates the least squares method for estimating linear regression parameters using dot product operations, with visualization of both true and estimated relationships.

2.2 Target Users

- Statistics and data science students learning regression analysis
- Educators teaching linear regression and vector operations
- Data analysts validating regression algorithms
- Researchers requiring synthetic dataset generation
- Software developers learning vectorized numerical computing

2.3 Product Author

Yair Levi

2.4 Key Innovation

This implementation specifically uses **dot product operations** for coefficient estimation, making the mathematical relationship between linear algebra and regression analysis explicit and educational.

3. Functional Requirements

3.1 Data Generation Requirements

3.1.1 Independent Variable (X) Generation

- **FR-001**: The application SHALL generate exactly 1000 random points for the X-axis
- **FR-002**: X values SHALL follow a normal distribution with mean $\mu = 0$

- **FR-003**: X values SHALL have standard deviation σ = 1
- **FR-004**: The application SHALL use NumPy's random number generation capabilities
- **FR-005**: X generation SHALL support optional random seed for reproducibility

3.1.2 Dependent Variable (Y) Generation

- **FR-006**: For each X point, the application SHALL calculate Y using: **$Y = \beta_0 + \beta_1 * X + \varepsilon$**
- **FR-007**: The true intercept parameter SHALL be $\beta_0 = 0.2$
- **FR-008**: The true slope parameter SHALL be $\beta_1 = 0.9$
- **FR-009**: Epsilon (ε) SHALL be random noise following normal distribution
- **FR-010**: Epsilon SHALL have mean = 0 and configurable standard deviation
- **FR-011**: Each Y value SHALL be calculated independently with fresh noise
- **FR-012**: The application SHALL store all (X, Y) pairs for analysis

3.2 Coefficient Estimation Requirements

3.2.1 Mathematical Foundation

The application SHALL implement ordinary least squares (OLS) estimation using the following formulas:

Formula 1 - Slope Estimation:

$$\beta_1 = \Sigma^{n}_{i=1}((Xi - \bar{X}) * (Yi - \bar{Y})) / \Sigma^{n}_{i=1}((Xi - \bar{X})^2)$$

Formula 2 - Intercept Estimation:

$$\beta_0 = \bar{Y} - \beta_1 * \bar{X}$$

Where:

- $\bar{X}$ = average of all X values
- $\bar{Y}$ = average of all Y values
- n = 1000 (number of points)

3.2.2 Dot Product Implementation Requirements

CRITICAL: All summation operations MUST be implemented using dot product operations.

3.2.2.1 Numerator Calculation for β_1

- **FR-013**: The numerator $\Sigma((Xi - \bar{X}) * (Yi - \bar{Y}))$ SHALL be calculated using **dot product**
- **FR-014**: Implementation SHALL use: `np.dot(X_dev, Y_dev)`
- **FR-015**: Where X_dev = X - $\bar{X}$ (deviation vector)
- **FR-016**: Where Y_dev = Y - $\bar{Y}$ (deviation vector)
- **FR-017**: The dot product operation SHALL replace element-wise multiplication + sum

3.2.2.2 Denominator Calculation for β_1

- **FR-018**: The denominator $\Sigma((Xi - \bar{X})^2)$ SHALL be calculated using **dot product**
- **FR-019**: Implementation SHALL use: `np.dot(X_dev, X_dev)`
- **FR-020**: This represents the dot product of X_dev with itself
- **FR-021**: The result equals sum of squared deviations

3.2.2.3 Mean Calculations

- **FR-022**: $\bar{X}$ (X average) SHALL be calculated using `np.mean(X)`
- **FR-023**: $\bar{Y}$ (Y average) SHALL be calculated using `np.mean(Y)`

3.2.2.4 Deviation Vector Calculations

- **FR-024**: X_dev SHALL be calculated as: `X - X_avg` (vectorized subtraction)
- **FR-025**: Y_dev SHALL be calculated as: `Y - Y_avg` (vectorized subtraction)
- **FR-026**: Deviation calculations SHALL use NumPy broadcasting
- **FR-027**: No explicit loops SHALL be used for deviation computation

3.2.3 Coefficient Calculation Sequence

- **FR-028**: Step 1: Calculate $\bar{X}$ = mean(X)
- **FR-029**: Step 2: Calculate $\bar{Y}$ = mean(Y)
- **FR-030**: Step 3: Calculate X_dev = X - $\bar{X}$
- **FR-031**: Step 4: Calculate Y_dev = Y - $\bar{Y}$
- **FR-032**: Step 5: Calculate β_1 = dot(X_dev, Y_dev) / dot(X_dev, X_dev)
- **FR-033**: Step 6: Calculate $\beta_0 = \bar{Y} - \beta_1 * \bar{X}$
- **FR-034**: All calculations SHALL be performed in vector form

3.2.4 No Loop Requirements

- **FR-035**: The application SHALL NOT use any explicit Python loops (for/while) in coefficient calculations
- **FR-036**: All operations SHALL be vectorized using NumPy
- **FR-037**: Dot product SHALL be the primary operation for summations

3.3 Visualization Requirements

3.3.1 Single Graph Requirement

- **FR-038**: The application SHALL display exactly ONE graph containing all visual elements
- **FR-039**: Multiple separate plots are NOT permitted
- **FR-040**: All visualization SHALL be integrated into a single figure

3.3.2 Required Visual Elements

- **FR-041**: The graph SHALL display scatter plot of all 1000 (X, Y) data points
- **FR-042**: The graph SHALL display the true regression line (Y = 0.2 + 0.9*X)
- **FR-043**: The graph SHALL display the estimated regression line (Y = β_0_est + β_1_est*X)
- **FR-044**: All three elements (points, true line, estimated line) MUST be visible simultaneously

3.3.3 Visual Styling Requirements

- **FR-045**: Data points SHALL be displayed with transparency (alpha < 1)
- **FR-046**: True regression line SHALL be solid and visually distinct
- **FR-047**: Estimated regression line SHALL be distinguishable from true line
- **FR-048**: Color scheme SHALL ensure clear differentiation:
  - Data points: Blue
  - True line: Red, solid
  - Estimated line: Green, dashed
- **FR-049**: Line width for true line SHALL be ≥ 2.5
- **FR-050**: Line width for estimated line SHALL be ≥ 2.0

3.3.4 Graph Labeling Requirements

- **FR-051**: X-axis SHALL be labeled "X"
- **FR-052**: Y-axis SHALL be labeled "Y"
- **FR-053**: Graph SHALL include descriptive title with equation
- **FR-054**: Legend SHALL identify all visual elements

- **FR-055**: Legend SHALL display actual coefficient values for both lines
- **FR-056**: Grid SHALL be displayed with appropriate transparency

3.3.5 Regression Line Specifications

- **FR-057**: True line SHALL represent: $Y = \beta_0\_true + \beta_1\_true * X$
- **FR-058**: Estimated line SHALL represent: $Y = \beta_0\_est + \beta_1\_est * X$
- **FR-059**: Both lines SHALL span the full range of X values in the data
- **FR-060**: Lines SHALL be computed using 100+ evenly spaced X values

3.4 Output Requirements

3.4.1 Console Output Requirements

- **FR-061**: Display author name: "Yair Levi"
- **FR-062**: Print application title and purpose
- **FR-063**: Display configuration parameters ($n=1000$, $\mu=0$, $\sigma=1$, $\beta_0=0.2$, $\beta_1=0.9$)
- **FR-064**: Print step-by-step execution progress
- **FR-065**: Display formulas being used with dot product notation
- **FR-066**: Show data generation statistics

3.4.2 Statistical Reporting Requirements

- **FR-067**: Print true coefficient values ($\beta_0=0.2$, $\beta_1=0.9$)
- **FR-068**: Print estimated coefficient values with 6 decimal places
- **FR-069**: Calculate and display absolute error for each coefficient
- **FR-070**: Calculate and display relative error as percentage
- **FR-071**: Show X and Y data ranges (min, max)
- **FR-072**: Display X and Y mean and standard deviation

3.4.3 Formula Display Requirements

- **FR-073**: Explicitly show: "Formula for β_1: dot(Xi - X_avg, Yi - Y_avg) / dot(Xi - X_avg, Xi - X_avg)"
- **FR-074**: Explicitly show: "Formula for β_0: Y_avg - β_1 * X_avg"
- **FR-075**: Use dot product notation in all formula displays
- **FR-076**: Emphasize the use of dot product operations

4. Technical Requirements

4.1 Programming Language and Environment

- **TR-001**: Implementation SHALL use Python 3.6 or higher
- **TR-002**: Code SHALL follow PEP 8 style guidelines
- **TR-003**: All functions SHALL include comprehensive docstrings
- **TR-004**: Variable names SHALL be mathematically intuitive

4.2 Required Libraries

- **TR-005**: NumPy SHALL be used for all numerical operations
- **TR-006**: Matplotlib SHALL be used for visualization
- **TR-007**: No scipy dependency is required
- **TR-008**: Only standard library and NumPy/Matplotlib SHALL be dependencies

4.3 Dot Product Implementation Requirements

- **TR-009**: All dot products SHALL use `np.dot(a, b)` function
- **TR-010**: Alternative syntax `a @ b` MAY be used if preferred
- **TR-011**: Dot product SHALL be the explicit method for:
  - Calculating $\Sigma((X_i - \bar{X}) * (Y_i - \bar{Y}))$
  - Calculating $\Sigma((X_i - \bar{X})^2)$
- **TR-012**: No use of `np.sum()` with element-wise multiplication in coefficient estimation
- **TR-013**: Dot product usage SHALL be clearly documented in code

4.4 Algorithm Requirements

- **TR-014**: Coefficient estimation SHALL use ordinary least squares (OLS) method
- **TR-015**: All mathematical operations SHALL be fully vectorized
- **TR-016**: Dot product operations SHALL replace loop-based summations
- **TR-017**: Memory efficiency SHALL be maintained through NumPy arrays
- **TR-018**: Numerical stability SHALL be considered in all calculations

4.5 Performance Requirements

- **TR-019**: Data generation SHALL complete within 0.5 seconds
- **TR-020**: Coefficient estimation SHALL complete within 0.1 seconds
- **TR-021**: Total execution time SHALL be under 3 seconds (excluding visualization interaction)
- **TR-022**: Memory usage SHALL not exceed 50MB during execution

- **TR-023**: Application SHALL efficiently handle the 1000-point dataset

4.6 Code Structure Requirements

- **TR-024**: Code SHALL be organized into distinct functions:
  - `generate_data()`: Data generation
  - `estimate_coefficients()`: Coefficient estimation using dot product
  - `plot_data_and_lines()`: Single graph visualization
  - `print_results()`: Results comparison
  - `main()`: Workflow orchestration
- **TR-025**: Configuration parameters SHALL be defined as module-level constants
- **TR-026**: Each function SHALL have single, clear responsibility
- **TR-027**: Error handling SHALL be implemented for edge cases

5. Mathematical Specifications

5.1 Data Generation Model

Model Definition:

$$X \sim \text{Normal}(\mu=0, \sigma=1)$$
$$\varepsilon \sim \text{Normal}(0, \sigma_\varepsilon)$$
$$Y = \beta_0 + \beta_1 * X + \varepsilon$$

True Parameters:
- $\beta_0 = 0.2$ (intercept)
- $\beta_1 = 0.9$ (slope)
- $\sigma_\varepsilon = 0.3$ (configurable noise level)

5.2 Least Squares Estimation with Dot Product

5.2.1 Mean Calculations

$$\bar{X} = (1/n) * \Sigma^{n}_{i=1} Xi$$
$$\bar{Y} = (1/n) * \Sigma^{n}_{i=1} Yi$$

NumPy Implementation:

python

```
X_avg = np.mean(X)
Y_avg = np.mean(Y)
```

5.2.2 Deviation Vectors

$$X\_dev = [X_1 - \bar{X}, X_2 - \bar{X}, ..., X_n - \bar{X}]$$
$$Y\_dev = [Y_1 - \bar{Y}, Y_2 - \bar{Y}, ..., Y_n - \bar{Y}]$$

NumPy Implementation (Vectorized):

```
python

X_dev = X - X_avg  # Broadcasting: subtract scalar from array
Y_dev = Y - Y_avg  # Broadcasting: subtract scalar from array
```

5.2.3 Slope Estimation Using Dot Product

$$\text{Numerator} = \Sigma^n_{i=1}((Xi - \bar{X}) * (Yi - \bar{Y}))$$
$$= X\_dev \cdot Y\_dev \quad \text{(dot product)}$$

$$\text{Denominator} = \Sigma^n_{i=1}((Xi - \bar{X})^2)$$
$$= X\_dev \cdot X\_dev \quad \text{(dot product with self)}$$

$$\beta_1 = \text{Numerator} / \text{Denominator}$$

NumPy Implementation (Dot Product):

```
python

numerator = np.dot(X_dev, Y_dev)      # Sum of products
denominator = np.dot(X_dev, X_dev)    # Sum of squares
beta_1_est = numerator / denominator
```

5.2.4 Intercept Estimation

$$\beta_0 = \bar{Y} - \beta_1 * \bar{X}$$

NumPy Implementation:

```
python
```

```
beta_0_est = Y_avg - beta_1_est * X_avg
```

5.3 Dot Product Properties

5.3.1 Dot Product Definition

For vectors $\mathbf{a} = [a_1, a_2, ..., a_n]$ and $\mathbf{b} = [b_1, b_2, ..., b_n]$:

$$a \cdot b = \Sigma^{n}_{i=1}(a_i * b_i) = a_1b_1 + a_2b_2 + ... + a_nb_n$$

5.3.2 Application to Regression

Numerator (Covariance-like term):

$$X\_dev \cdot Y\_dev = \Sigma^{n}_{i=1}((Xi - \bar{X}) * (Yi - \bar{Y}))$$

This measures how X and Y vary together.

Denominator (Variance of X):

$$X\_dev \cdot X\_dev = \Sigma^{n}_{i=1}((Xi - \bar{X})^2)$$

This is the sum of squared deviations (variance × n).

Slope Interpretation:

$$\beta_1 = (X\_dev \cdot Y\_dev) / (X\_dev \cdot X\_dev)$$

Represents how much Y changes per unit change in X.

5.4 Statistical Properties

- **Unbiasedness**: $E[\beta_0\_est] = \beta_0$, $E[\beta_1\_est] = \beta_1$
- **Consistency**: As $n \to \infty$, estimates converge to true values
- **Efficiency**: OLS estimators are BLUE (Best Linear Unbiased Estimators) under Gauss-Markov assumptions
- **Expected Accuracy**: With n=1000 and $\sigma\_\varepsilon=0.3$, estimates should be within 1-3% of true values

6. Quality Requirements

6.1 Accuracy Requirements

- **QR-001**: Dot product calculations SHALL produce numerically identical results to summation methods
- **QR-002**: With 1000 points, estimated β_1 SHALL typically be within 5% of true value
- **QR-003**: With 1000 points, estimated β_0 SHALL typically be within 10% of true value
- **QR-004**: Numerical precision SHALL be maintained to at least 6 decimal places
- **QR-005**: Dot product implementation SHALL be mathematically correct

6.2 Reliability Requirements

- **QR-006**: Application SHALL produce consistent results across multiple runs (with same seed)
- **QR-007**: Optional seed parameter SHALL enable exact reproducibility
- **QR-008**: Application SHALL handle edge cases gracefully
- **QR-009**: No runtime errors SHALL occur during normal operation
- **QR-010**: Dot product operations SHALL not cause overflow/underflow

6.3 Code Quality Requirements

- **QR-011**: Dot product usage SHALL be clearly commented
- **QR-012**: Mathematical relationship to formulas SHALL be documented
- **QR-013**: Code SHALL be readable and maintainable
- **QR-014**: Variable names SHALL match mathematical notation where possible

6.4 Educational Quality Requirements

- **QR-015**: Dot product operations SHALL be explicit and educational
- **QR-016**: Connection between linear algebra and regression SHALL be clear
- **QR-017**: Code SHALL serve as learning resource for vectorization
- **QR-018**: Output SHALL explain what dot products represent

7. Acceptance Criteria

7.1 Data Generation Acceptance Criteria

- [ ] Generates exactly 1000 X points from Normal(0, 1)
- [ ] Y values calculated using Y = 0.2 + 0.9*X + ε

- [ ] All 1000 (X, Y) pairs stored correctly
- [ ] Data statistics (mean, std) approximately match expected values

7.2 Dot Product Implementation Acceptance Criteria

- [ ] β_1 numerator calculated using `np.dot(X_dev, Y_dev)`
- [ ] β_1 denominator calculated using `np.dot(X_dev, X_dev)`
- [ ] No use of `np.sum()` with element-wise multiplication in coefficient formulas
- [ ] Dot product explicitly mentioned in code comments
- [ ] Dot product notation used in console output formulas
- [ ] No explicit Python loops in coefficient calculation
- [ ] All operations properly vectorized

7.3 Coefficient Calculation Acceptance Criteria

- [ ] X_avg and Y_avg calculated using `np.mean()`
- [ ] Deviation vectors computed using broadcasting
- [ ] β_1 formula implemented exactly as specified using dot product
- [ ] β_0 formula implemented exactly as specified
- [ ] Results are numerically accurate
- [ ] Estimated coefficients close to true values (typically within 5%)

7.4 Visualization Acceptance Criteria

- [ ] Exactly ONE graph displayed
- [ ] All 1000 data points visible as scatter plot
- [ ] True regression line (red, solid) clearly visible
- [ ] Estimated regression line (green, dashed) clearly visible
- [ ] Both lines distinguishable from each other
- [ ] Legend includes all elements with coefficient values
- [ ] Graph properly labeled (title, axes, legend)
- [ ] Lines span the full range of data

7.5 Output Acceptance Criteria

- [ ] Author "Yair Levi" displayed
- [ ] Configuration parameters printed
- [ ] Formulas displayed with dot product notation
- [ ] True parameters printed (β_0=0.2, β_1=0.9)
- [ ] Estimated parameters printed with 6 decimals
- [ ] Absolute errors calculated and displayed

- [ ] Relative errors shown as percentages
- [ ] Step-by-step progress messages shown

7.6 Code Quality Acceptance Criteria

- [ ] No explicit loops in coefficient estimation
- [ ] Dot product explicitly used and documented
- [ ] All functions have proper docstrings
- [ ] Code follows PEP 8 style guidelines
- [ ] Variable names are clear and mathematical
- [ ] Comments explain dot product usage

8. Implementation Specifications

8.1 Configuration Constants

```python
NUM_POINTS = 1000       # Number of data points
MU_X = 0                # Mean of X distribution
SIGMA_X = 1             # Std dev of X distribution
BETA_0 = 0.2            # True intercept
BETA_1 = 0.9            # True slope
EPSILON_SIGMA = 0.3     # Noise standard deviation
SEED = None             # Optional random seed
```

8.2 Critical Function: estimate_coefficients()

```python
```

```python
def estimate_coefficients(X, Y):
    """
    Estimate β₀ and β₁ using least squares formulas with dot product.

    Formulas:
    β₁ = dot(Xi - X_avg, Yi - Y_avg) / dot(Xi - X_avg, Xi - X_avg)
    β₀ = Y_avg - β₁ * X_avg

    Args:
        X (np.ndarray): X values (shape: (n,))
        Y (np.ndarray): Y values (shape: (n,))

    Returns:
        tuple: (beta_0_estimated, beta_1_estimated)
    """
    # Calculate averages
    X_avg = np.mean(X)
    Y_avg = np.mean(Y)

    # Calculate deviations (vectorized)
    X_dev = X - X_avg
    Y_dev = Y - Y_avg

    # Calculate β₁ using DOT PRODUCT
    numerator = np.dot(X_dev, Y_dev)      # Sum of cross-products
    denominator = np.dot(X_dev, X_dev)    # Sum of squared deviations
    beta_1_est = numerator / denominator

    # Calculate β₀
    beta_0_est = Y_avg - beta_1_est * X_avg

    return beta_0_est, beta_1_est
```

8.3 Visualization Specifications

Color and Style Scheme

```python
```

```
# Data points
color='blue', alpha=0.4, s=20, edgecolors='navy'

# True line
color='red', linestyle='-', linewidth=3

# Estimated line
color='green', linestyle='--', linewidth=2.5
```

Graph Properties

```python
figsize=(12, 8)
grid=True, alpha=0.3
xlabel='X', fontsize=14, fontweight='bold'
ylabel='Y', fontsize=14, fontweight='bold'
title fontsize=15, fontweight='bold'
legend fontsize=11
```

9. Testing Requirements

9.1 Unit Testing

Test 1: Dot Product Correctness

```python
def test_dot_product_equivalence():
    """Verify dot product gives same result as sum of products."""
    X_dev = np.array([1, 2, 3])
    Y_dev = np.array([4, 5, 6])

    # Using dot product
    result_dot = np.dot(X_dev, Y_dev)

    # Using sum (for verification)
    result_sum = np.sum(X_dev * Y_dev)

    assert result_dot == result_sum  # Should be equal: 32
```

Test 2: Known Data Test

```python
def test_perfect_fit():
    """Test with perfect linear relationship (no noise)."""
    X = np.array([1, 2, 3, 4, 5])
    Y = 0.2 + 0.9 * X  # Perfect fit

    beta_0, beta_1 = estimate_coefficients(X, Y)

    assert abs(beta_0 - 0.2) < 1e-10
    assert abs(beta_1 - 0.9) < 1e-10
```

Test 3: Self Dot Product

```python
def test_self_dot_product():
    """Verify X_dev · X_dev equals sum of squares."""
    X_dev = np.array([1, -2, 3])

    dot_result = np.dot(X_dev, X_dev)
    sum_sq_result = np.sum(X_dev ** 2)

    assert dot_result == sum_sq_result  # Should be: 14
```

9.2 Integration Testing

- **TEST-004**: Verify complete workflow from generation to visualization
- **TEST-005**: Test with seed for reproducibility
- **TEST-006**: Verify estimated coefficients close to true values
- **TEST-007**: Test with different noise levels
- **TEST-008**: Verify all three visual elements appear in single graph

9.3 Validation Testing

- **TEST-009**: Compare dot product method vs traditional sum method (should be identical)
- **TEST-010**: Verify estimation accuracy across 10 different random seeds
- **TEST-011**: Test with extreme noise levels (σ_ε = 0.1, 1.0, 2.0)
- **TEST-012**: Validate against known regression results from statistical software

9.4 Educational Testing

- **TEST-013**: Verify dot product usage is explicit in code
- **TEST-014**: Confirm formula display uses dot product notation
- **TEST-015**: Ensure code serves as clear teaching example
- **TEST-016**: Validate that linear algebra connection is clear

10. Documentation Requirements

10.1 Code Documentation

- **DOC-001**: Explain what dot product represents in regression context
- **DOC-002**: Document why dot product is used instead of sum
- **DOC-003**: Include mathematical formulas in docstrings
- **DOC-004**: Comment on the relationship between linear algebra and statistics

10.2 Formula Documentation

- **DOC-005**: Display formulas with dot product notation in output
- **DOC-006**: Explain numerator as dot product of deviation vectors
- **DOC-007**: Explain denominator as dot product of X_dev with itself
- **DOC-008**: Show connection to covariance and variance

10.3 Educational Documentation

- **DOC-009**: Explain advantages of dot product approach
- **DOC-010**: Show how vectorization improves performance
- **DOC-011**: Demonstrate connection between formulas and code
- **DOC-012**: Provide learning resources for linear algebra in statistics

11. Success Metrics

11.1 Implementation Success Metrics

- 100% use of dot product for summation operations in coefficient calculation
- Zero explicit Python loops in numerical computations
- Execution time < 1 second for 1000 points
- Memory usage < 20MB

11.2 Accuracy Success Metrics

- Dot product results numerically identical to traditional methods (within floating-point precision)

- Coefficient estimates within 5% of true values (95% of runs)
- Mean estimation error < 2% across 100 random runs

11.3 Educational Success Metrics

- Code clearly demonstrates dot product usage
- Linear algebra connection is explicit
- Suitable for teaching vectorization concepts
- Students can understand the implementation

11.4 Visualization Success Metrics

- Single graph contains all required elements
- Visual distinction between true and estimated lines is clear
- Graph is publication-quality
- Legend is informative and complete

12. Risk Assessment

12.1 Technical Risks

Risk	Probability	Impact	Mitigation
Dot product implementation error	Low	High	Thorough testing against known results
Numerical precision issues	Very Low	Medium	Use NumPy's stable implementations
Confusion between dot and element-wise	Medium	Low	Clear documentation and comments
Performance degradation	Very Low	Low	Dot product is highly optimized

12.2 Educational Risks

Risk	Probability	Impact	Mitigation
Dot product not clearly explained	Medium	Medium	Extensive documentation
Mathematical connection unclear	Medium	Medium	Add comments explaining formulas
Code too complex for learners	Low	Medium	Keep implementation simple and clear

13. Future Enhancements

13.1 Algorithm Enhancements

- Matrix formulation: β = (X^T X)^(-1) X^T Y

- Multiple regression using dot products
- Weighted least squares
- Ridge regression with penalty term

13.2 Educational Enhancements

- Interactive visualization showing dot product calculation
- Animation of dot product computation
- Comparison with loop-based implementation (speed test)
- Geometric interpretation of dot product

13.3 Feature Enhancements

- Confidence intervals using dot product formulations
- Residual analysis with vectorized operations
- R^2 calculation using dot products
- Hypothesis testing for coefficients

14. Compliance and Standards

14.1 Mathematical Standards

- Standard statistical notation (β, ε, μ, σ)
- Dot product notation: a · b or dot(a, b)
- Summation notation: $\Sigma^{n}_{i=1}$
- Consistent variable naming

14.2 Code Standards

- PEP 8 compliance
- NumPy documentation standards
- Clear mathematical variable names
- Explicit dot product usage

14.3 Educational Standards

- Clear explanation of concepts
- Connection to theoretical foundations
- Suitable for undergraduate statistics courses

- Demonstrates best practices in scientific computing

Document Version: 2.0
Author: Yair Levi
Last Updated: October 3, 2025
Key Innovation: Explicit use of dot product operations for coefficient estimation
Status: Approved for Implementation
Next Review: Upon implementation completion