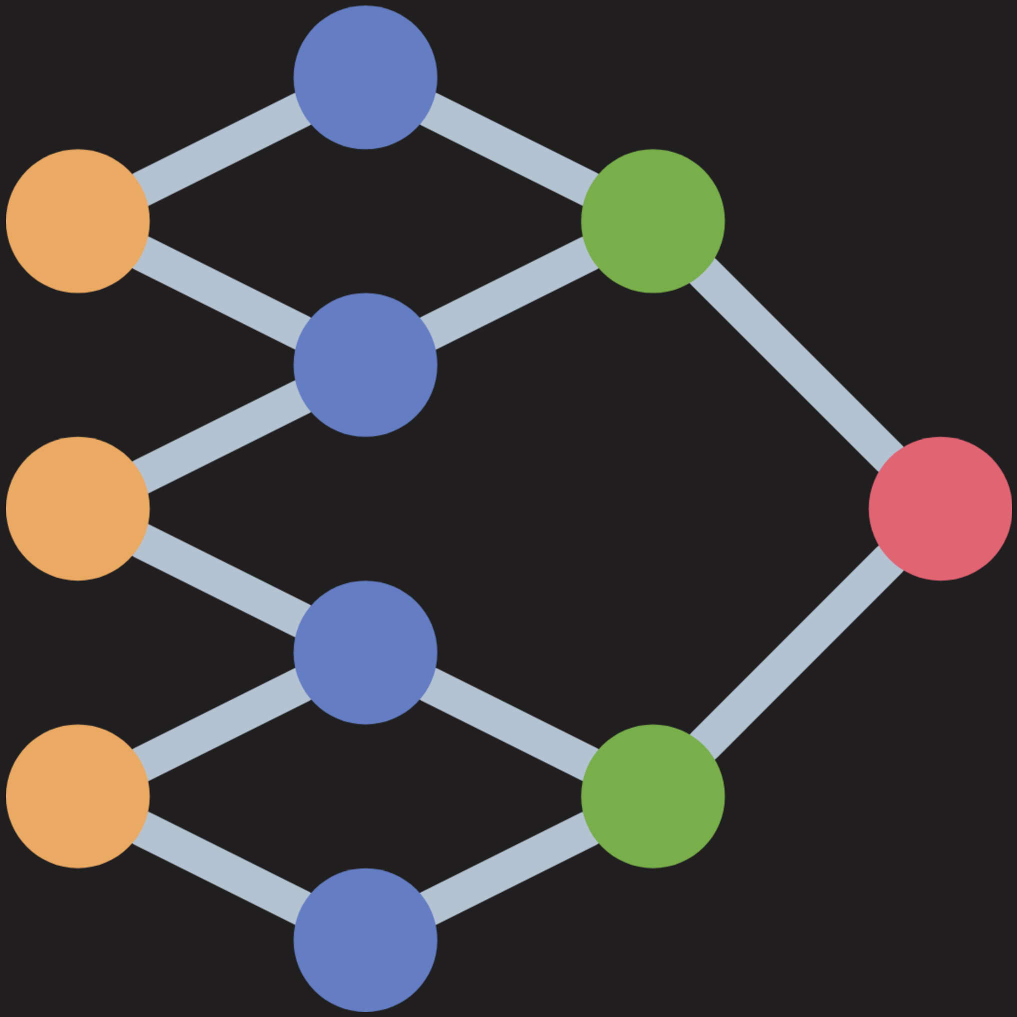
Linear algebra for AI



By Sutton

Linear algebra for AI

Chapter 1: Introduction to Linear Algebra

What is Linear Algebra?

Linear Algebra is a branch of mathematics focused on vectors, vector spaces, linear transformations, and systems of linear equations. It is fundamental to many scientific disciplines, including engineering, physics, computer science, and particularly, machine learning and AI.

Importance of Linear Algebra in AI

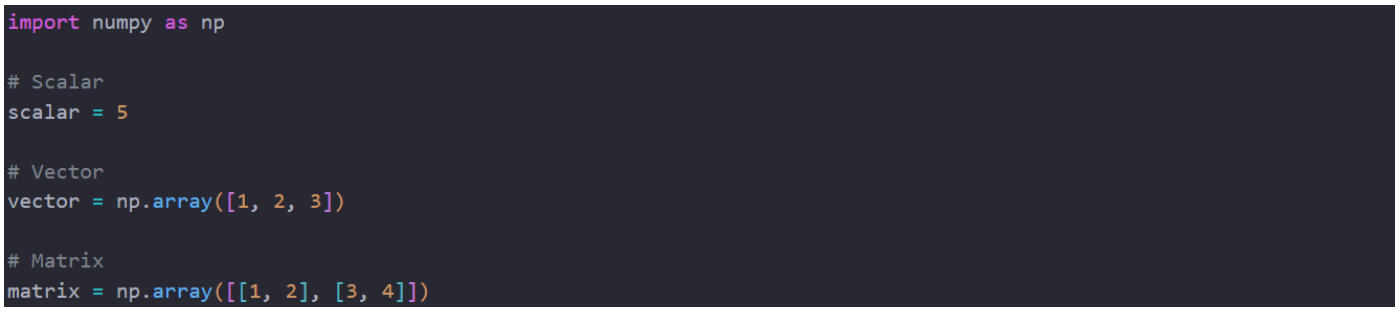
Linear Algebra provides the mathematical foundation for understanding and designing algorithms in AI and machine learning. Techniques such as data transformations, dimensionality reduction, and optimization are heavily based on linear algebra concepts.

Overview of Python in Linear Algebra

Python, with libraries like NumPy and SciPy, offers powerful tools for performing linear algebra operations. These libraries simplify complex mathematical computations, making Python a popular choice for implementing AI algorithms.

Chapter 2: Basic Concepts and Operations

Scalar: A single number.
Vector: An ordered list of numbers.
Matrix: A 2-dimensional array of numbers.



```python
import numpy as np

# Scalar
scalar = 5

# Vector
vector = np.array([1, 2, 3])

# Matrix
matrix = np.array([[1, 2], [3, 4]])
```

Matrix Addition and Subtraction

Matrices of the same dimensions can be added or subtracted element-wise.

```python
import numpy as np

matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])

# Addition
matrix_sum = matrix_a + matrix_b

# Subtraction
matrix_diff = matrix_a - matrix_b
```

Scalar Multiplication

Each element of a matrix is multiplied by a scalar.

```python
import numpy as np

scalar = 3
matrix = np.array([[1, 2], [3, 4]])

# Scalar Multiplication
matrix_scaled = scalar * matrix
```

Matrix Multiplication

The product of two matrices is calculated using the dot product of rows and columns.

```python
import numpy as np

matrix_a = np.array([[1, 2], [3, 4]])
matrix_b = np.array([[5, 6], [7, 8]])

# Matrix Multiplication
matrix_product = np.dot(matrix_a, matrix_b)
```

Transpose of a Matrix

Flipping a matrix over its diagonal, switching the row and column indices.

```python
import numpy as np

matrix = np.array([[1, 2], [3, 4]])

# Transpose
matrix_transpose = np.transpose(matrix)
```

Chapter 3: Vector Spaces

Definition and Examples

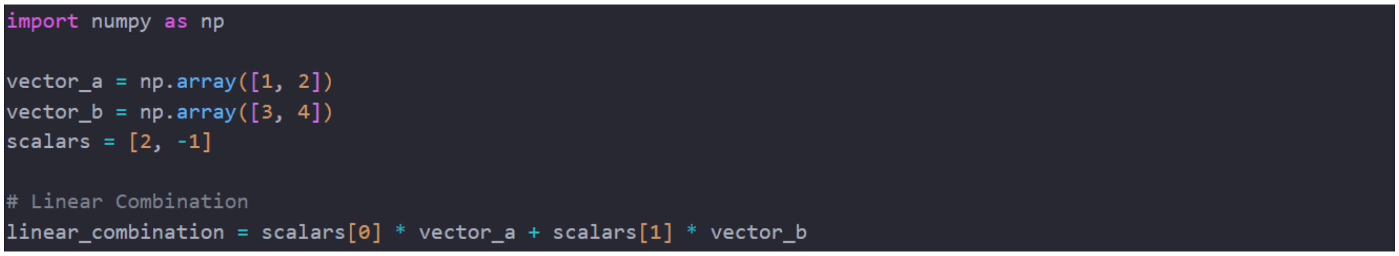
A vector space is a collection of vectors that can be added together and multiplied by scalars.

Subspaces

A subspace is a subset of a vector space that is also a vector space.

Linear Combinations

A linear combination involves adding scaled vectors together.



```python
import numpy as np

vector_a = np.array([1, 2])
vector_b = np.array([3, 4])
scalars = [2, -1]

# Linear Combination
linear_combination = scalars[0] * vector_a + scalars[1] * vector_b
```

Basis and Dimension

The basis of a vector space is a set of vectors that are linearly independent and span the vector space. The number of vectors in the basis is the dimension.

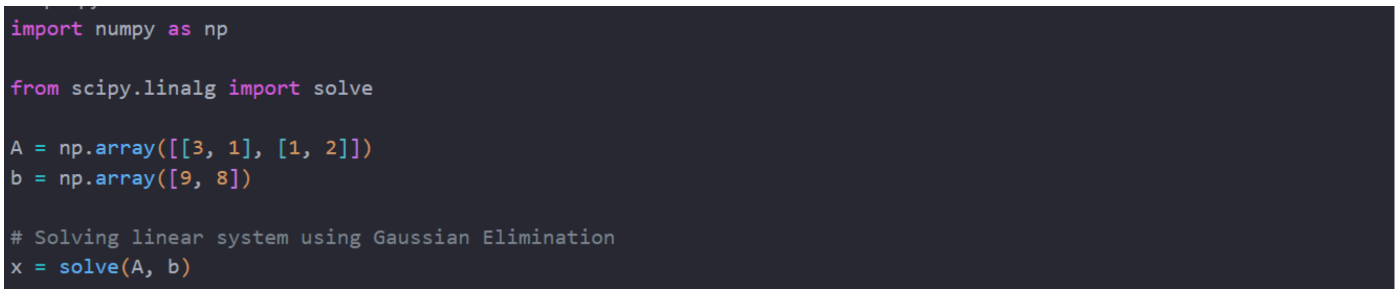
Chapter 4: Solving Linear Systems

Systems of Linear Equations

A system of linear equations can be written in matrix form as Ax = b, where A is the matrix of coefficients, x is the vector of variables, and b is the vector of constants.

Gaussian Elimination

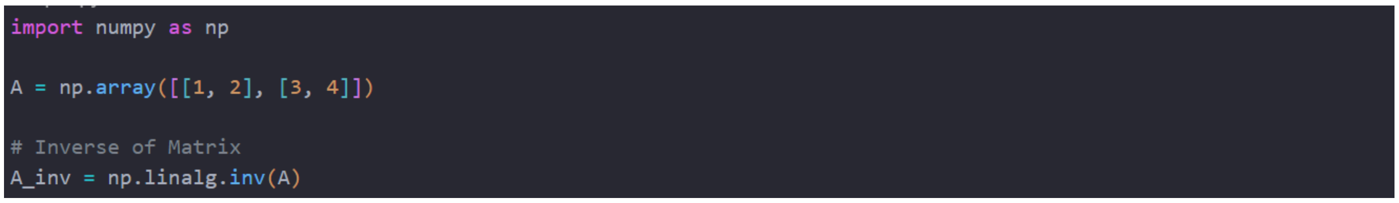
A method to solve a system of linear equations by transforming the matrix into row echelon form.



```python
import numpy as np

from scipy.linalg import solve

A = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])

# Solving linear system using Gaussian Elimination
x = solve(A, b)
```

Matrix Inversion

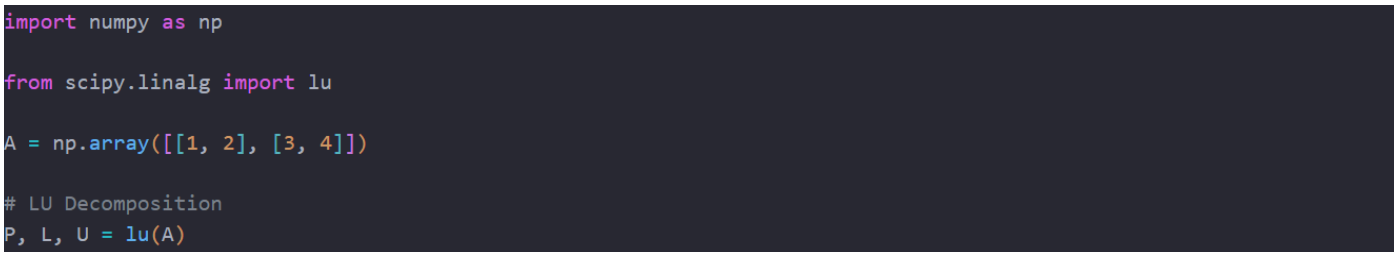
The inverse of a matrix A is a matrix B such that AB = BA = I, where I is the identity matrix.



```python
import numpy as np

A = np.array([[1, 2], [3, 4]])

# Inverse of Matrix
A_inv = np.linalg.inv(A)
```

LU Decomposition

Factoring a matrix as the product of a lower triangular matrix and an upper triangular matrix.



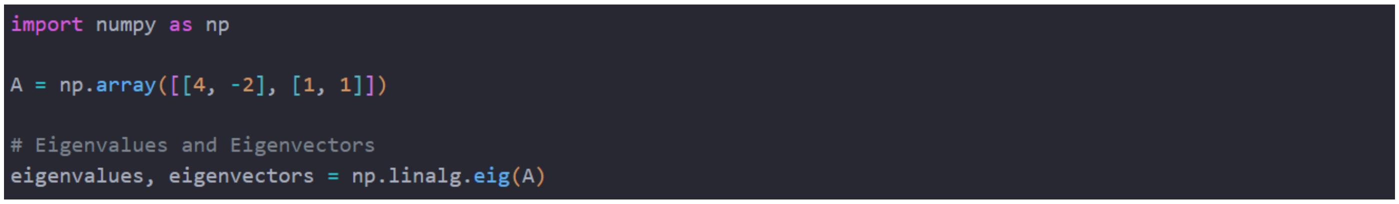
```python
import numpy as np

from scipy.linalg import lu

A = np.array([[1, 2], [3, 4]])

# LU Decomposition
P, L, U = lu(A)
```

Chapter 5: Eigenvalues and Eigenvectors

Definition and Properties

Eigenvalues and eigenvectors are properties of a square matrix. For a matrix A A, if there is a scalar λ λ and a non-zero vector v v such that $A v = \lambda v$ Av=λv, then λ λ is an eigenvalue and v v is an eigenvector of A A.

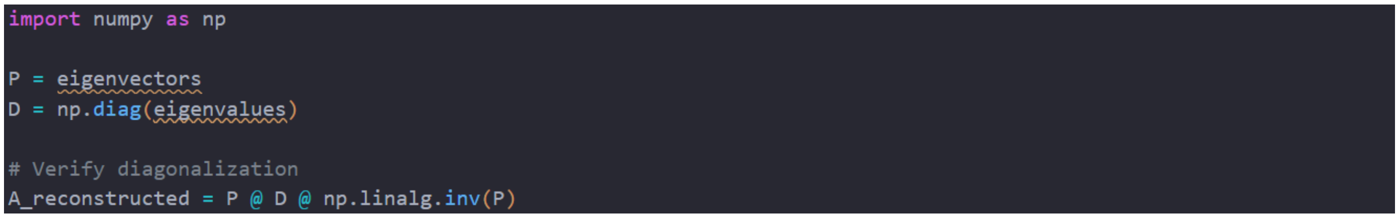
Calculation of Eigenvalues and Eigenvectors



```python
import numpy as np

A = np.array([[4, -2], [1, 1]])

# Eigenvalues and Eigenvectors
eigenvalues, eigenvectors = np.linalg.eig(A)
```

Diagonalization

A matrix A A can be diagonalized if it can be written as $A = P D P^{-1}$ A=PDP −1 , where P P is a matrix of eigenvectors and D D is a diagonal matrix of eigenvalues.



```python
import numpy as np

P = eigenvectors
D = np.diag(eigenvalues)

# Verify diagonalization
A_reconstructed = P @ D @ np.linalg.inv(P)
```

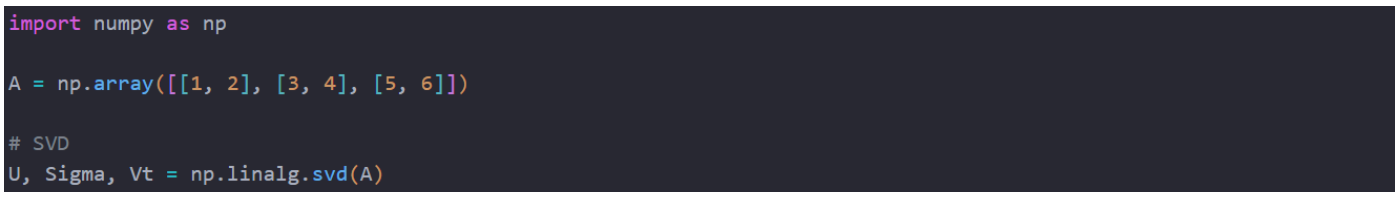
Applications in AI

Eigenvalues and eigenvectors are used in Principal Component Analysis (PCA), which is essential for dimensionality reduction in machine learning.

Chapter 6: Matrix Decompositions

Singular Value Decomposition (SVD)

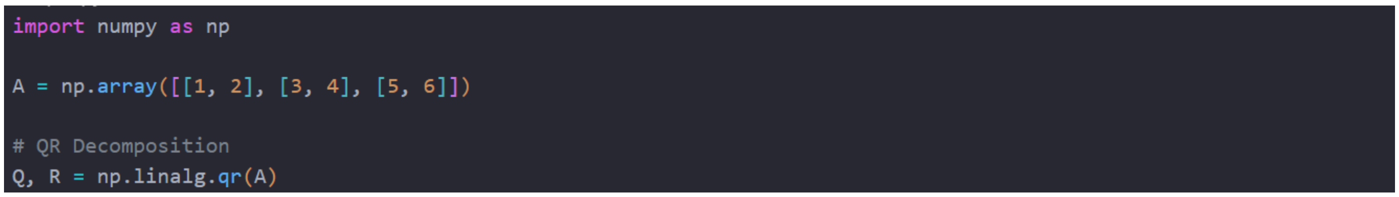
SVD decomposes a matrix A A into three matrices: $A = U \Sigma V^T$ A=UΣV T , where U U and V V are orthogonal matrices, and Σ Σ is a diagonal matrix.



```python
import numpy as np

A = np.array([[1, 2], [3, 4], [5, 6]])

# SVD
U, Sigma, Vt = np.linalg.svd(A)
```

QR Decomposition

QR decomposition factorizes a matrix A A into a product of an orthogonal matrix Q Q and an upper triangular matrix R R.



```python
import numpy as np

A = np.array([[1, 2], [3, 4], [5, 6]])

# QR Decomposition
Q, R = np.linalg.qr(A)
```

Applications in Data Compression and Dimensionality Reduction

SVD is used in data compression techniques like image compression, while both SVD and QR decomposition are used in dimensionality reduction techniques like PCA.

Chapter 7: Advanced Topics

Norms and Distance Measures

Norms measure the size or length of a vector. Common norms include the L1 norm (Manhattan distance) and the L2 norm (Euclidean distance).

```python
import numpy as np

vector = np.array([1, 2, 3])

# L1 Norm
l1_norm = np.linalg.norm(vector, 1)

# L2 Norm
l2_norm = np.linalg.norm(vector)
```

Orthogonality and Projections

Two vectors are orthogonal if their dot product is zero. Projection of vector a a onto vector b b is the shadow of a a on b b.

```python
import numpy as np

a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# Orthogonality
dot_product = np.dot(a, b)
orthogonal = np.isclose(dot_product, 0)

# Projection
projection = (np.dot(a, b) / np.dot(b, b)) * b
```

Positive Definite Matrices

A matrix is positive definite if all its eigenvalues are positive. Positive definite matrices are used in optimization problems.

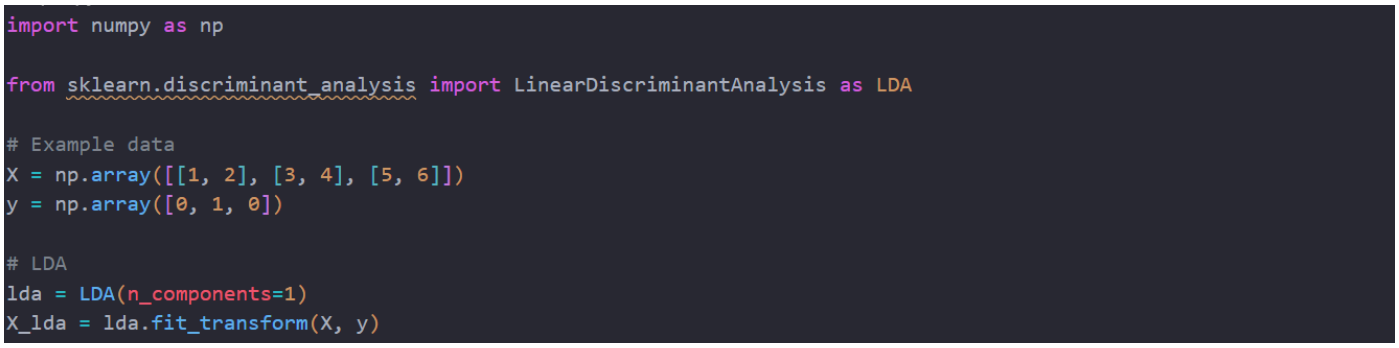
```python
import numpy as np

A = np.array([[2, -1], [-1, 2]])

# Check if matrix is positive definite
eigenvalues = np.linalg.eigvals(A)
is_positive_definite = np.all(eigenvalues > 0)
```

Advanced Applications in AI (PCA, LDA, etc.)

Principal Component Analysis (PCA)

```python
import numpy as np

from sklearn.decomposition import PCA

data = np.array([[1, 2], [3, 4], [5, 6]])

# PCA
pca = PCA(n_components=1)
data_reduced = pca.fit_transform(data)
```

Linear Discriminant Analysis (LDA)



```python
import numpy as np

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# Example data
X = np.array([[1, 2], [3, 4], [5, 6]])
y = np.array([0, 1, 0])

# LDA
lda = LDA(n_components=1)
X_lda = lda.fit_transform(X, y)
```

Chapter 8: Python for Linear Algebra

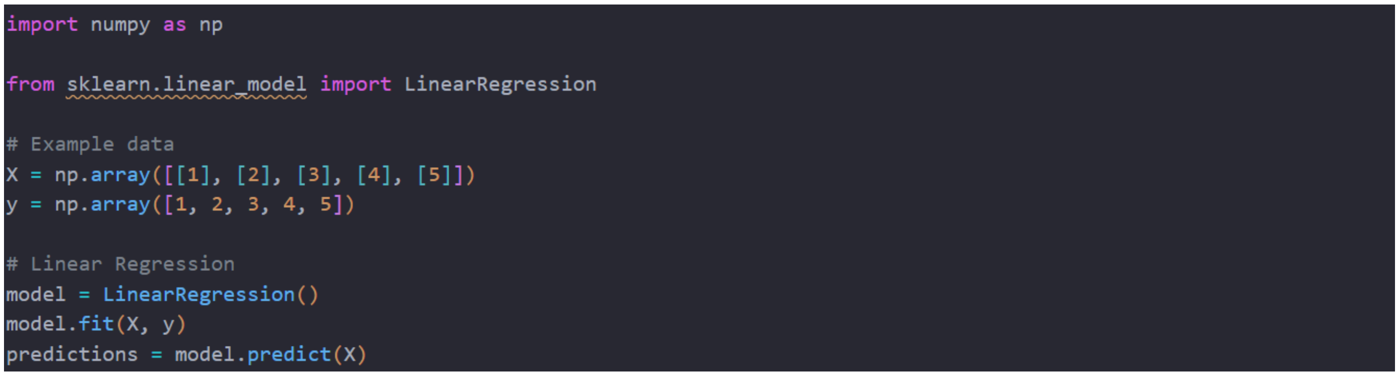
Introduction to NumPy

NumPy is the fundamental package for scientific computing in Python, providing support for arrays, matrices, and many mathematical functions.

```python
import numpy as np

# Creating Arrays
array = np.array([1, 2, 3])
matrix = np.array([[1, 2], [3, 4]])

# Basic Operations
sum_array = np.sum(array)
mean_array = np.mean(array)
```

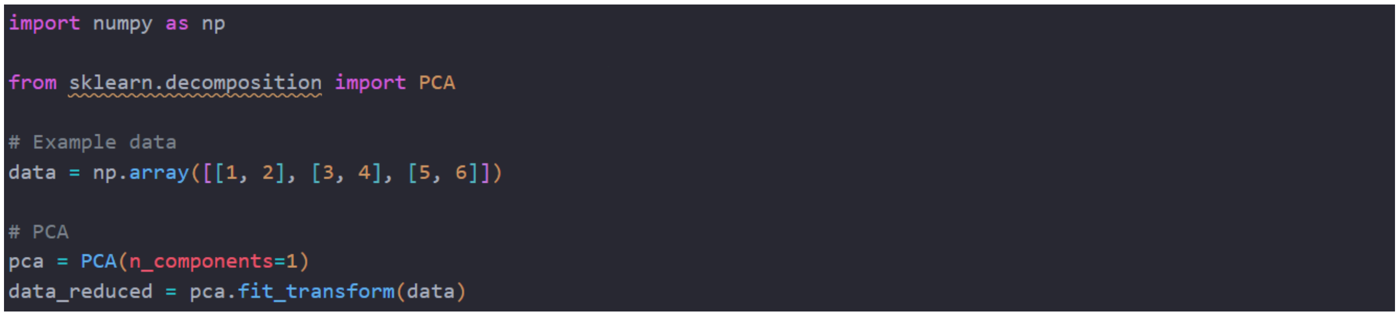
Chapter 9: Machine Learning Applications

Linear Regression



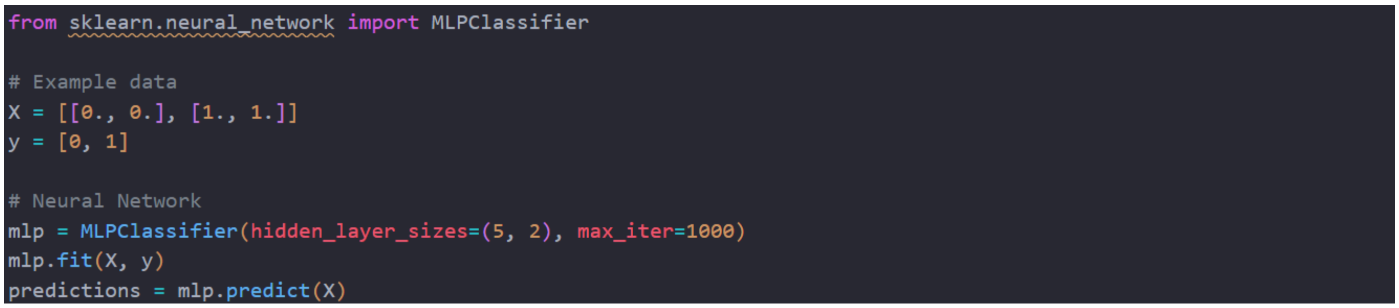
```python
import numpy as np

from sklearn.linear_model import LinearRegression

# Example data
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([1, 2, 3, 4, 5])

# Linear Regression
model = LinearRegression()
model.fit(X, y)
predictions = model.predict(X)
```

Principal Component Analysis (PCA)



```python
import numpy as np

from sklearn.decomposition import PCA

# Example data
data = np.array([[1, 2], [3, 4], [5, 6]])

# PCA
pca = PCA(n_components=1)
data_reduced = pca.fit_transform(data)
```

Neural Networks



```python
from sklearn.neural_network import MLPClassifier

# Example data
X = [[0., 0.], [1., 1.]]
y = [0, 1]

# Neural Network
mlp = MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=1000)
mlp.fit(X, y)
predictions = mlp.predict(X)
```

Chapter 10: Case Studies and Projects

Real-world AI Applications

1. **Recommendation Systems:** Using matrix factorization techniques to recommend products to users.
2. **Image Compression:** Applying SVD to compress images.

Hands-on Projects

1. **Project 1:** Predicting Housing Prices
   Use linear regression to predict housing prices based on various features.

2. **Project 2:** Dimensionality Reduction with PCA
   Apply PCA to reduce the dimensionality of a dataset and visualize the results.

Integrating Linear Algebra in AI Models

Detailed steps on incorporating linear algebra techniques into machine learning models for improved performance and efficiency.

Chapter 11: Further Reading and Resources

Recommended Books and Articles

1. "Linear Algebra and Its Applications" by Gilbert Strang "
2. Introduction to Linear Algebra" by Gilbert Strang
3. "Python for Data Analysis" by Wes McKinney

Online Courses and Tutorials

1. **Coursera:** Linear Algebra for Machine Learning
2. **edX:** Linear Algebra - Foundations to Frontiers

Python Libraries for Linear Algebra and AI

1. **NumPy:** Fundamental library for linear algebra operations.
2. **SciPy:** Additional linear algebra functionalities.
3. **scikit-learn:** Tools for machine learning and data mining.