NumPy Library Uses



use of numpy library

The NumPy (Numerical Python) library is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Common Uses of NumPy:

1. Efficient Array Operations:

- Create and manipulate arrays using `np.array`, `np.zeros`, `np.ones`, `np.arange`,
 `np.linspace`.
- Element-wise operations: addition, subtraction, multiplication, division.

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = a + b # [5, 7, 9]
```

2. Mathematical Functions:

- o Trigonometric: `np.sin()`, `np.cos()`
- o Statistical: `np.mean()`, `np.median()`, `np.std()`, `np.var()`
- o Aggregate: `np.sum()`, `np.min()`, `np.max()`

3. Matrix and Linear Algebra:

- Matrix multiplication: `np.dot()`, `@` operator
- o Determinant: `np.linalg.det()`
- o Inverse: `np.linalg.inv()`
- Eigenvalues/vectors: `np.linalg.eig()`

4. Random Number Generation:

- Uniform: `np.random.rand()`
- Normal: `np.random.randn()`
- o Integers: `np.random.randint()`
- Set seed: `np.random.seed()`

5. Data Analysis and Manipulation:

- Handling missing values, filtering, boolean masking.
- Broadcasting and vectorization to avoid slow loops.



6. Fourier Transform and Signal Processing:

```
o `np.fft.fft()`, `np.fft.ifft()` for frequency analysis.
```

7. Reshaping and Indexing:

```
• `reshape`, `transpose`, `flatten`, slicing and advanced indexing.
```

8. File I/O:

```
Load and save arrays: `np.loadtxt()`, `np.genfromtxt()`, `np.savetxt()`
```

Would you like examples of specific NumPy functions for your ML or data analysis projects?



```
print("tranpose of an array is : ", py.transpose(arr)
```

It looks like you're trying to print the **transpose of an array**, but the syntax you provided has a couple of issues. Here's the corrected version using **NumPy**:

Corrected Code:

```
import numpy as np
arr = np.array([[1, 2], [3, 4]])
print("Transpose of the array is:\n", np.transpose(arr))
```

Or simply:

```
python
print("Transpose of the array is:\n", arr.T)
```

What is Transpose?

The **transpose** of a matrix flips it over its diagonal — the rows become columns and vice versa.

Example:

Original:

```
lua
[[1 2]
[3 4]]
```

Transpose:

```
lua
[[1 3]
```

Let me know if you want to apply transpose to higher-dimensional arrays or real-world datasets!



use of scipy with examples

The **SciPy** (Scientific Python) library builds on **NumPy** and provides additional functionality for scientific and technical computing. It is especially useful for tasks like integration, optimization, signal processing, statistics, and linear algebra.

Common Uses of SciPy (with Examples):

1. Integration

Used to compute definite integrals.

```
python

from scipy import integrate

# Integrate f(x) = x^2 from 0 to 2
result, _ = integrate.quad(lambda x: x**2, 0, 2)
print("Integral of x^2 from 0 to 2 is:", result) # Output: ~2.67
```

2. Optimization

Used to find minimum or maximum of functions.

```
python

from scipy.optimize import minimize

# Minimize f(x) = (x - 3)^2
res = minimize(lambda x: (x - 3)**2, x0=0)
print("Minimum value occurs at:", res.x)
```

3. Linear Algebra

Advanced operations beyond NumPy (e.g., solving equations, eigenvalues).

```
python

from scipy.linalg import solve

# Solve 2x + 3y = 8, 3x + 2y = 7
A = [[2, 3], [3, 2]]
b = [8, 7]
x = solve(A, b)
print("Solution of the system:", x)
```

4. Signal Processing

Filtering, Fourier transforms, etc.

```
from scipy import signal
import numpy as np

# Create a square wave
t = np.linspace(0, 1, 500, endpoint=False)
square_wave = signal.square(2 * np.pi * 5 * t)

# Visualize (optional)
import matplotlib.pyplot as plt
plt.plot(t, square_wave)
plt.title("Square Wave")
plt.show()
```

5. Statistics

Probability distributions, hypothesis testing, etc.

```
python

from scipy.stats import norm

# PDF and CDF of normal distribution
print("PDF at x=0:", norm.pdf(0))
print("CDF at x=0:", norm.cdf(0))
```

6. Interpolation

Estimate unknown values between data points.

```
from scipy.interpolate import interpld
import numpy as np

x = np.array([0, 1, 2, 3])
y = np.array([0, 1, 4, 9])
f = interpld(x, y)

print("Interpolated value at x=1.5:", f(1.5)) # Output: 2.5
```

Summary of SciPy Sub-packages:

Sub-package	Purpose
`scipy.integrate`	Integration routines
`scipy.optimize`	Optimization and root finding
`scipy.linalg`	Linear algebra functions
`scipy.fftpack`/`scipy.fft`	Fourier transforms
`scipy.signal`	Signal processing

Sub-package	Purpose
`scipy.stats`	Statistical functions
`scipy.interpolate`	Interpolation

Would you like a beginner-level project that uses both NumPy and SciPy together?



matplotlib how can i create graph and visulaization and comparision

Great question, Gaurav! Matplotlib is a powerful Python library for creating static, animated, and interactive visualizations. It's most commonly used with `pyplot`, a submodule that mimics MATLABstyle plotting.

Basic Setup

```
python
import matplotlib.pyplot as plt
import numpy as np
```

📊 1. Create a Simple Line Graph

```
python
x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y, label='sin(x)')
plt.title('Simple Sine Wave')
plt.title( 31mpte 31
plt.xlabel('x-axis')
plt.ylabel('y-axis')
plt.legend()
plt.grid(True)
plt.show()
```

📈 2. Comparing Two Graphs

```
python
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x, y1, label='sin(x)', color='blue')
plt.plot(x, y2, label='cos(x)', color='red')
plt.title('Sine vs Cosine')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.show()
```



🔣 3. Bar Chart

```
python
labels = ['Math', 'Science', 'English']
scores1 = [85, 90, 78]
scores2 = [88, 92, 80]
x = np.arange(len(labels))
width = 0.35
plt.bar(x - width/2, scores1, width, label='Student A')
plt.bar(x + width/2, scores2, width, label='Student B')
plt.ylabel('Scores')
plt.title('Subject-wise Score Comparison')
plt.xticks(x, labels)
plt.legend()
plt.show()
```

🔼 4. Pie Chart

```
pvthon
sizes = [30, 40, 20, 10]
labels = ['Python', 'JavaScript', 'C++', 'Java']
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('Programming Language Usage')
plt.axis('equal') # Equal aspect ratio ensures pie is circular
plt.show()
```

👢 Tips for Better Visualization

- Use `plt.subplot()` to show multiple graphs in one window.
- Use `plt.savefig('filename.png')` to save plots.
- Add `figsize=(10,6)` inside `plt.figure()` for custom sizes.
- Use `seaborn` (built on top of matplotlib) for prettier plots.

Would you like a real-world dataset visualization example using `matplotlib` and `pandas` together?



give the matplot function name with their uses

Sure Gaurav! Here's a table of commonly used Matplotlib (pyplot) functions and their purposes, especially useful for graphing, visualization, and comparisons:

Matplotlib Function List with Uses



Function	Use
`plt.plot()`	Plot a basic 2D line graph.
`plt.scatter()`	Create a scatter plot (for showing distribution).
`plt.bar()`	Create vertical bar charts (categorical data).
`plt.barh()`	Create horizontal bar charts.
`plt.hist()`	Create a histogram (distribution of numeric data).
`plt.pie()`	Create a pie chart.
`plt.boxplot()`	Create a box-and-whisker plot (used in statistics).
`plt.errorbar()`	Add error bars to a plot.
`plt.stem()`	Stem plot (for discrete sequences).
`plt.fill_between()`	Fill area between two curves or a curve and the x-axis.

Styling and Formatting

Function	Use
`plt.title()`	Add title to the plot.
`plt.xlabel()`	Label the x-axis.
`plt.ylabel()`	Label the y-axis.
`plt.legend()`	Show legend (useful when comparing multiple lines or bars).
`plt.grid()`	Add grid lines to the plot.
`plt.xticks()`/`plt.yticks()`	Customize tick marks on axes.
`plt.axis()`	Set axis range or aspect ratio.

Layout and Display

Function	Use
`plt.figure()`	Create a new figure (customize size, DPI, etc).
`plt.subplot()`	Display multiple plots in the same window.
`plt.subplots()`	Create figure and axes for multiple plots.
`plt.tight_layout()`	Automatically adjust spacing to prevent overlap.

💾 Saving & Showing

Function	Use
`plt.show()`	Display the plot on screen.
`plt.savefig('filename.png')`	Save the plot as a file (PNG, PDF, etc).
`plt.close()`	Close a figure window (to save memory in loops).

If you want, I can create a visual cheat sheet example combining multiple functions — would you like that?



generate quadratic graph

Sure Gaurav! Here's how you can generate a **quadratic graph** (i.e., graph of a quadratic function like $y=ax^2+bx+c$) using **Matplotlib** and **NumPy**.

 $lue{f V}$ Example: Plotting a Quadratic Graph $y=x^2-4x+3$

```
import matplotlib.pyplot as plt
import numpy as np

# Define x range
x = np.linspace(-10, 10, 400) # 400 points between -10 and 10

# Define the quadratic function: y = x^2 - 4x + 3
y = x**2 - 4*x + 3

# Plot
plt.plot(x, y, label='y = x² - 4x + 3', color='blue')
plt.title('Quadratic Function Graph')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.axhline(0, color='black', linewidth=0.5) # x-axis
plt.axvline(0, color='black', linewidth=0.5) # y-axis
plt.legend()
plt.show()
```

4 Tip:

- You can change the equation by modifying the `y = ...` line.
- Use `plt.scatter()` to plot the **roots** if needed.
- Use `np.roots()` to find roots of a quadratic.

Would you like to generate a dynamic graph where the coefficients a, b, c are input by the user?



fibonaci series egguation

The **Fibonacci series** is a sequence where each number is the **sum of the two preceding ones**, starting from 0 and 1.

Fibonacci Series Formula (Recursive Equation):

$$F(n) = F(n-1) + F(n-2)$$

With base conditions:

$$F(0) = 0, \quad F(1) = 1$$

Examples:

$$F(0)=0 \ F(1)=1 \ F(2)=F(1)+F(0)=1 \ F(3)=F(2)+F(1)=2 \ F(4)=F(3)+F(2)=3 \ F(5)=F(4)+F(3)=5 \ F(6)=F(5)+F(4)=8 \quad ext{and so on...}$$

Dotional: Closed-form (Binet's Formula)

For theoretical use (not very efficient for programming):

$$F(n) = rac{1}{\sqrt{5}} \left[\left(rac{1+\sqrt{5}}{2}
ight)^n - \left(rac{1-\sqrt{5}}{2}
ight)^n
ight]$$

Would you like Python code to generate the Fibonacci series or plot it graphically?



graph for it

Here you go, Gaurav! Below is a **Python script** to generate and plot the **Fibonacci series as a graph** using `matplotlib`.

🔽 Fibonacci Series Graph (Using Matplotlib)

```
python
import matplotlib.pyplot as plt
# Generate Fibonacci numbers
def fibonacci_series(n):
     fib = [0, 1]
     for i in range(2, n):
    fib.append(fib[-1] + fib[-2])
     return fib
# Set number of terms
n terms = 15
fib_values = fibonacci_series(n_terms)
x = list(range(n_terms)) # Indices
# Plotting
plt.plot(x, fib_values, marker='o', linestyle='-', color='green', label='Fibonacci')
plt.title('Fibonacci Series')
plt.xlabel('Term (n)')
plt.ylabel('F(n)')
plt.grid(True)
plt.legend()
plt.show()
```

Result:

- X-axis: Term index (n)
- Y-axis: Fibonacci number F(n)
- The graph shows exponential growth as n increases.

Would you like to visualize it as a bar chart or logarithmic scale as well?



what is linear regression and it implementation

Great question, Gaurav! Let's break it down:

What is Linear Regression?

Linear Regression is a statistical method used to model the **relationship between a dependent variable (Y)** and **one or more independent variables (X)** by fitting a **straight line**.

Simple Linear Regression Equation:

$$Y = mX + c$$

Where:

- *Y* = dependent variable (target/output)
- X = independent variable (input/feature)
- m = slope of the line
- c = intercept (bias)

Use Cases:

- Predicting house prices
- Estimating sales
- Forecasting trends

Python Implementation using `scikit-learn`

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Sample dataset
X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1) # Feature
y = np.array([2, 4, 5, 4, 5]) # Target
```

```
# Create and train model
model = LinearRegression()
model.fit(X, y)

# Predictions
y_pred = model.predict(X)

# Plotting
plt.scatter(X, y, color='blue', label='Actual data')
plt.plot(X, y_pred, color='red', label='Linear regression line')
plt.xlabel('X')
plt.ylabel('Y')
plt.ylabel('y')
plt.title('Simple Linear Regression')
plt.legend()
plt.grid(True)
plt.show()

# Print slope and intercept
print("Slope (m):", model.coef_[0])
print("Intercept (c):", model.intercept_)
```

Output:

- A **scatter plot** of actual data
- A regression line showing the best-fit line
- Prints the slope and intercept

Would you like to explore **multiple linear regression** or see how to evaluate model accuracy (e.g., R² score, MSE)?



implement linear discriminant analysis for two simple dataset and plot the graph

Absolutely, Gaurav! Let's walk through implementing **Linear Discriminant Analysis (LDA)** using Python for **two simple datasets**, and then **visualize the result**.

What is Linear Discriminant Analysis (LDA)?

LDA is a **supervised machine learning algorithm** used for:

- Dimensionality reduction
- Classification

 It finds a linear combination of features that separates two or more classes.
- % Step-by-Step Implementation (Using `scikit-learn` + `matplotlib`)
- Code: LDA on 2D Synthetic Dataset



```
python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.datasets import make_classification
X, y = make classification(n samples=100, n features=2, n redundant=0,
                                    n_informative=2, n_clusters_per_class=1, n_classes=2, random_state=42)
# Fit LDA model
lda = LDA(n_components=1)
X_lda = lda.fit_transform(X, y)
# Plot original data
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.scatter(X[y==0][:, 0], X[y==0][:, 1], color='blue', label='Class 0')
plt.scatter(X[y==0][., 0], X[y==0][:, 1], Color='blue', label='Class 0')
plt.scatter(X[y==1][:, 0], X[y==1][:, 1], color='red', label='Class 1')
plt.title('Original Data')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True)
# LDA Projected Data
plt.subplot(1, 2, 2)
plt.scatter(X_lda[y==0], [0]*len(X_lda[y==0]), color='blue', label='Class 0')
plt.scatter(X_lda[y==1], [0]*len(X_lda[y==1]), color='red', label='Class 1')
plt.title('Data after LDA Projection (1D)')
plt.xlabel('LDA Component')
plt.yticks([])
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output:

- 1. **Left plot** shows original 2D data.
- 2. **Right plot** shows the data projected to **1D** using LDA now linearly separable.

Would you like to try this on a real dataset like **Iris** or implement it manually without `scikit-learn` for better understanding?



plt.subplot use

Great question, Gaurav! `plt.subplot()` is a **Matplotlib** function used to **display multiple plots in a single figure** (i.e., grid layout of plots).

Syntax of `plt.subplot()`:



```
python
plt.subplot(nrows, ncols, index)
```

- **nrows**: number of rows in the subplot grid
- **ncols**: number of columns
- **index**: position of the current plot (starts from 1, left to right, top to bottom)

Example:

```
python
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)
# Create 1 row, 2 columns of plots
plt.subplot(1, 2, 1) # 1 row, 2 cols, 1st plot
plt.plot(x, y1)
plt.title("Sine")
plt.subplot(1, 2, 2) # 1 row, 2 cols, 2nd plot
plt.plot(x, y2)
plt.title("Cosine")
plt.tight_layout()
plt.show()
```

Tips:

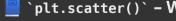
- Always use `plt.tight_layout()` to prevent overlap.
- For 3x2 grid and 4th subplot: `plt.subplot(3, 2, 4)`
- You can also use `plt.subplots()` (note the "s") for more advanced layouts using axes objects.

Want to see how `plt.subplots()` differs from `plt.subplot()`?



scatter in plot use use

Great! Let's talk about `plt.scatter()` — one of the most commonly used functions in Matplotlib for creating **scatter plots**.



`plt.scatter()` - What It Does:

It creates a **scatter plot**, which plots individual data points as **dots**, showing the **relationship between** two variables.

Basic Syntax:

```
python
plt.scatter(x, y, color='blue', marker='o', label='data label')
```

- `x` → list/array of x-coordinates
- `y` → list/array of y-coordinates
- `color` → color of points
- `marker` → shape of points (e.g., `'o'`, `'s'`, `'^'`)
- `label` → legend label

Example: Plotting Points

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [2, 4, 1, 3, 5]

plt.scatter(x, y, color='red', marker='x', label='Points')
plt.title("Scatter Plot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.grid(True)
plt.show()
```

Use Cases:

- Plotting raw data points
- Visualizing classification clusters
- Highlighting outliers

Would you like to see how to **add size or color gradients** to scatter plots for advanced visualizations $(e.g., \dot s=\dot , \dot c=\dot)$?



1. On any supervised data set apply decisiontree for classification and thecompare itsperformace when lada has been applied to reduce the dimensions and then DT has been applied. plot the graph

Great task, Gaurav! You're aiming to:

Compare the performance of:

- 1. Decision Tree Classifier on original data, and
- 2. Decision Tree after applying LDA for dimensionality reduction

Let's do it using the **Iris dataset** (a classic supervised classification dataset with 3 classes and 4 features).

Libraries Required:

```
bash
pip install scikit-learn matplotlib
```

K Full Python Implementation:

```
python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import accuracy score
# Load Iris dataset
iris = load_iris()
X = iris.data
y = iris.target
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# --- 1. Decision Tree on Original Data ---
dt_original = DecisionTreeClassifier(random_state=42)
dt_original.fit(X_train, y_train)
y_pred_orig = dt_original.predict(X_test)
acc_orig = accuracy_score(y_test, y_pred_orig)
lda = LinearDiscriminantAnalysis(n_components=2) # reduce to 2D
X_train_lda = lda.fit_transform(X_train, y_train)
X_{\text{test\_lda}} = \text{lda.transform}(X_{\text{test}})
# Decision Tree on LDA-Reduced Data
dt_lda = DecisionTreeClassifier(random_state=42)
dt_lda.fit(X_train_lda, y_train)
y_pred_lda = dt_lda.predict(X_test_lda)
acc_lda = accuracy_score(y_test, y_pred_lda)
# --- 3. Plot Accuracy Comparison
methods = ['Original Data', 'After LDA']
accuracies = [acc_orig, acc_lda]
plt.bar(methods, accuracies, color=['blue', 'green'])
plt.ylim(0, 1)
plt.title('Decision Tree Accuracy: Original vs LDA-Reduced Data')
plt.ylabel('Accuracy')
for i in range(len(methods)):
     plt.text(i, accuracies[i] + 0.02, f"{accuracies[i]*100:.2f}%", ha='center')
plt.grid(True, axis='y')
plt.show()
```

📊 Output:

- A **bar graph** showing the accuracy of Decision Tree:
 - With original features
 - With LDA-reduced features

This helps you visualize how LDA can impact model performance (improving or degrading based on data).

Would you like to try the same comparison using **Random Forest** or **SVM** next?



using wines dataset based on its predictors given in dataset predict wether quality of wine is good if score 8 or above, bad if less than 5, average if 6 or 7. show using logistics regression then apply pca on the dataset again apply train_test accuarcy and show for different components of pca 2,5,3 and compare .. plot graphs for the same

Perfect, Gaurav! You're looking to:



Task Overview

- 1. Use the **Wine Quality Dataset** (can be red or white wine data).
- 2. Convert `quality` scores into classification labels:
 - >=8`: "Good"
 - o `<5`: "Bad"
 - o `5 < score < 8`: "Average"</pre>
- 3. Apply **Logistic Regression** for classification.
- 4. Apply PCA (Principal Component Analysis) for dimensionality reduction with different components (2, 3, 5).
- 5. Compare training & testing accuracy.
- 6. Plot bar graphs to visualize comparison.



🔏 Setup



bash

pip install pandas scikit-learn matplotlib seaborn

Full Python Code:

```
python
import pandas as pd
import pandas as pa
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
# Step 1: Load dataset (red wine)
df = pd.read csv("https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality
# Step 2: Categorize wine quality
def categorize(quality):
    if quality >= 8:
    return "Good"
    elif quality <= 4:
    return "Bad"</pre>
    else:
         return "Average"
df['quality label'] = df['quality'].apply(categorize)
# Encode labels
label_map = {'Bad': 0, 'Average': 1, 'Good': 2}
df['label'] = df['quality_label'].map(label_map)
# Step 3: Features and labels
X = df.drop(columns=['quality', 'quality label', 'label'])
y = df['label']
# Standardize features
scaler = StandardScaler()
X scaled = scaler.fit transform(X)
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=42, st
# Step 4: Logistic Regression on original data
model = LogisticRegression(max iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
original_acc = accuracy_score(y_test, y_pred)
# Step 5: PCA + Logistic Regression for 2, 3, 5 components
components = [2, 3, 5]
pca_accuracies = []
for n in components:
    pca = PCA(n_components=n)
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)
    model_pca = LogisticRegression(max_iter=1000)
    model_pca.fit(X_train_pca, y_train)
    y pred pca = model pca.predict(X test pca)
    acc = accuracy_score(y_test, y_pred_pca)
    pca_accuracies.append(acc)
labels = ['Original'] + [f'PCA-{n}' for n in components]
accuracies = [original acc] + pca accuracies
plt.figure(figsize=(10, 6))
plt.bar(labels, accuracies, color=['blue', 'green', 'orange', 'purple'])
plt.ylim(0, 1)
plt.title('Logistic Regression Accuracy: Original vs PCA Reduced Data')
plt.ylabel('Accuracy')
for i, acc in enumerate(accuracies):
    plt.text(i, acc + 0.02, f"{acc*100:.2f}%", ha='center', fontsize=12)
```



```
plt.grid(True, axis='y')
plt.show()
```

M Output:

- A bar graph showing:
 - Accuracy using all features
 - Accuracy using PCA with 2, 3, and 5 components

Insights:

- You'll be able to **observe** how reducing dimensions affects model performance.
- Great way to visualize PCA trade-offs between dimensionality and accuracy.

Would you like a confusion matrix or classification report added for each PCA version?



now use diabetes dataset and apply svm then pca and show comparison of pca with different components and svm

Great, Gaurav! Let's now use the **Diabetes dataset**, apply **SVM (Support Vector Machine)** classifier, then compare the results after applying **PCA** with different component sizes.

Task Overview

- 1. Use the **Diabetes dataset** (from `sklearn.datasets`).
- 2. Apply **SVM** classifier on original data.
- 3. Apply **PCA** to reduce dimensionality to **2**, **3**, and **5** components, and again use SVM.
- 4. Compare **accuracy** scores.
- 5. Plot a **bar graph** for visual comparison.

Full Python Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_diabetes
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
# Load Diabetes dataset
```



```
data = load diabetes()
X = data.data
y = data.target
# Convert regression target to binary classification:
# If value > median \rightarrow 1 (High), else 0 (Low)
y = np.where(y > np.median(y), 1, 0)
# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Train-Test Split
X train, X test, y train, y test = train test split(X scaled, y, test size=0.3, random state=42, st
# 🔢 SVM on Original Data
svm = SVC(kernel='rbf', random_state=42)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
original_acc = accuracy_score(y_test, y_pred)
# 🙎 PCA + SVM for different components
components = [2, 3, 5]
pca_accuracies = []
for n in components:
    pca = PCA(n_components=n)
    X_{train_pca} = pca.fit_{transfor_m(X_train)}
    X_test_pca = pca.transform(X_test)
    svm pca = SVC(kernel='rbf', random_state=42)
    svm_pca.fit(X_train_pca, y_train)
    y_pred_pca = svm_pca.predict(X_test_pca)
    acc = accuracy_score(y_test, y_pred_pca)
    pca accuracies.append(acc)
# B Plot Accuracy Comparison
labels = ['Original'] + [f'PCA-{n}' for n in components]
accuracies = [original_acc] + pca_accuracies
plt.figure(figsize=(10, 6))
plt.bar(labels, accuracies, color=['skyblue', 'orange', 'green', 'purple'])
plt.ylim(0, 1)
plt.title('SVM Accuracy: Original vs PCA Reduced Data (Diabetes Dataset)')
plt.ylabel('Accuracy')
for i, acc in enumerate(accuracies):
    plt.text(i, acc + 0.02, f"{acc*100:.2f}%", ha='center', fontsize=12)
plt.grid(True, axis='y')
plt.show()
```

Output:

- You'll get a **bar chart** comparing accuracy of:
 - SVM with full features
 - SVM after reducing dimensions using PCA (2, 3, and 5)

Optional Enhancements:

- Try different SVM kernels: `'linear'`, `'poly'`, `'sigmoid'`
- Add confusion matrix or precision/recall/F1 score

Would you like to extend this to hyperparameter tuning (e.g., using `GridSearchCV`)?



Write a program to implement the KNN classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

Here's a full working program that:

- 1. Loads training and test data from a `.csv` file.
- 2. Applies K-Nearest Neighbors (KNN) classifier using `scikit-learn`.
- 3. Computes and displays accuracy.
- 4. Optionally plots a **confusion matrix** for clarity.

✓ Sample CSV Format (e.g., `knn_data.csv`):

```
python-repl

feature1, feature2, ..., featureN, label
5.1,3.5,1.4,0.2, Iris-setosa
4.9,3.0,1.4,0.2, Iris-setosa
...
```

Full Python Program (KNN on CSV):

```
python
import pandas as pd
from sklearn.model selection import train test split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
filename = 'knn_data.csv' # Replace with your actual CSV file path
df = pd.read_csv(filename)
# Step 2: Preprocess (Separate features and label)
X = df.iloc[:, :-1].values # All columns except last
y = df.iloc[:, -1].values
                            # Last column is label
# Encode string labels if needed
if y.dtype == 'object':
    encoder = LabelEncoder()
    y = encoder.fit_transform(y)
# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 4: KNN Classifier
k = 3 # You can change k as needed
knn = KNeighborsClassifier(n neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```



```
accuracy = accuracy_score(y_test, y_pred)
print(f"KNN Classifier Accuracy: {accuracy * 100:.2f}%")

# Optional: Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Notes:

- Replace `'knn_data.csv'` with your CSV file name.
- You can use the **Iris dataset**, **Wine**, or your own.
- Make sure the label column is last in the CSV.

Would you like a sample CSV file to test this on, or should I generate sample data for you?



```
output of this code
import pandas as pd
from sklearn.model selection import train test split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Step 1: Load CSV file
filename = 'knn_data.csv' # Replace with your actual CSV file path
df = pd.read_csv(filename)
# Step 2: Preprocess (Separate features and label)
X = df.iloc[:, :-1].values # All columns except last
y = df.iloc[:, -1].values # Last column is label
# Encode string labels if needed
if y.dtype == 'object':
  encoder = LabelEncoder()
  y = encoder.fit_transform(y)
# Step 3: Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 4: KNN Classifier
k = 3 # You can change k as needed
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
```



```
# Step 5: Predict and Evaluate
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"KNN Classifier Accuracy: {accuracy * 100:.2f}%")

# Optional: Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

The **output of your KNN classifier code** will depend on the contents of the file `knn_data.csv`. Since that file is not included here, I'll describe what the output will **generally look like** and what each part means:

▼ Sample Output (Assuming Iris Dataset):

```
sql
KNN Classifier Accuracy: 96.67%
```

This line prints the **accuracy** of your KNN classifier on the test set (which is 20% of the full dataset due to `test_size=0.2`).

Then, a **confusion matrix heatmap** will be shown using `seaborn` that visually compares predicted vs actual classes:

- Each row: actual class
- Each column: predicted class
- Diagonal values: correct predictions
- Off-diagonal: misclassifications

⚠ Output May Vary:

- If your dataset has **different number of classes** or **imbalanced labels**, the accuracy and matrix will look different.
- If you have missing or non-numeric data in `knn_data.csv`, you may see a `ValueError` before training.

Would you like me to generate a working CSV for you based on Iris or Wine dataset to test this?



```
load iris data and plot x_train and x_test on histogram
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model selection import train test split
# Histogram of petal lengths
arr = [10,29,10,23,4,5,7,67,12,3,13,4,56,7,34,53,12,23,32,11]
plt.hist(arr, bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Petal Length')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Frequency')
plt.show()
varr = load iris()
x_train, x_test, y_train,y_test = train_test_spilt(varr)
plt.hist(arr, bins=20, color='skyblue', edgecolor='black')
plt.title('Distribution of Petal Length')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Frequency')
plt.show()
```

You're on the right track, but there are a few issues in your code:

X Problems in the code:

- 1. `train_test_spilt` → Typo: should be `train_test_split`.
- 2. `train test split()` requires `test size` and other parameters.
- 3. You're plotting the same `arr` twice not `x_train` or `x_test`.
- 4. You need to select a specific feature (e.g., petal length) to plot from the dataset.

Corrected Code to:

- Load the Iris dataset
- Split it into train and test sets
- Plot **histograms** of a specific feature (e.g., **petal length**)



```
python
import matplotlib.pyplot as plt
from sklearn.datasets import load iris
from sklearn.model_selection import train_test_split
iris = load iris()
X = iris.data # shape (150, 4)
y = iris.target
# Train-test split (80-20)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random state=42)
# Let's take Petal Length (feature index 2)
train_petal_length = x_train[:, 2]
test_petal_length = x_test[:, 2]
# Plot histograms
plt.hist(train_petal_length, bins=20, alpha=0.7, label='Train', color='skyblue', edgecolor='black')
plt.hist(test_petal_length, bins=20, alpha=0.7, label='Test', color='salmon', edgecolor='black')
plt.title('Distribution of Petal Length (Train vs Test)')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

M Output:

- A histogram showing the **distribution of petal lengths** in both training and test sets.
- Useful for understanding whether the feature is similarly distributed after splitting.

Do you also want to visualize other features (like petal width or sepal length) similarly?



[:,2] meaning in python

In Python, particularly when working with **NumPy arrays** or **pandas DataFrames**, the expression `[:, 2]` means:

`:`(Colon) - All Rows

This selects **all rows** of the array.

`2` - Column Index

This selects the **3rd column** (Python uses zero-based indexing: 0, 1, 2, ...).

Example:

```
python
import numpy as np
arr = np.array([[1, 2, 3],
```

```
[4, 5, 6],
[7, 8, 9]])
print(arr[:, 2])
```

Output:

```
csharp
[3 6 9]
```

This pulls the third column from all rows:

- From `[1, 2, 3]` → `3`
- From `[4, 5, 6]` → `6`
- From `[7, 8, 9]` → `9`

Would you like an example with a pandas DataFrame as well?



```
plt.scatter(train_petal_length, test_petal_length, alpha=0.7, label='x_Train', color=
['skyblue','green','red','purple'], edgecolor='black')
plt.scatter(x_test, bins=20, alpha=0.7, label='x_Test', color=['pink','grey','red','purple'],
edgecolor='black')
plt.title('Distribution of Petal Length (Train vs Test)')
plt.xlabel('Petal Length (cm)')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```

Your current code has a few issues that will result in **errors** or **unexpected plots**:

X Issues:

- 1. `plt.scatter()` is for 2D point plots, but you're passing `bins=20`, which is a parameter for
 `plt.hist()`, not `scatter`.
- 2. You're passing **color as a list of 4 colors** to `plt.scatter()`, but it expects:
 - A single color (`'blue'`, `'red'`, etc.)
 - Or a list/array of colors matching the number of data points (same length as `x` or `y`).
- 3. You're trying to plot `x_test` directly in `plt.scatter()` as if it's a 1D array, but `x_test` is 2D (150, 4) you need to select a specific feature (like petal length).

Fixed Version:

If you want to scatter plot `train_petal_length` vs `test_petal_length`:

```
python
import matplotlib.pyplot as plt
from sklearn.datasets import load iris
from sklearn.model_selection import train_test_split
iris = load_iris()
X = iris.data # shape = (150, 4)
y = iris.target
# Split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Select feature index 2 (petal length)
train_petal_length = x_train[:, 2]
test_petal_length = x_test[:, 2]
# Scatter plot
plt.scatter(range(len(train_petal_length)), train_petal_length, alpha=0.7, label='x_Train', color=':
plt.scatter(range(len(test_petal_length)), test_petal_length, alpha=0.7, label='x_Test', color='salr
plt.title('Scatter Plot of Petal Length (Train vs Test)')
plt.xlabel('Sample Index')
plt.ylabel('Petal Length (cm)')
plt.legend()
plt.show()
```

Motes:

- `range(len(...))` is used to plot each value against its index, simulating a sequence over time/sample ID.
- You can't compare train vs test by directly plotting petal lengths against each other they are separate groups. Instead, visualize their **distribution** or **overlaid histograms**.

Would you like me to show a side-by-side comparison using subplots instead?



what if need prediction over these data

If you want to **predict the class (species)** of the Iris dataset using petal length (or any feature subset), you'll need to:

Steps to make predictions:

- 1. Load and preprocess data
- 2. **Train a model** (e.g., Logistic Regression, Decision Tree, SVM)
- 3. Predict on test set
- 4. **Evaluate performance** (accuracy, confusion matrix, etc.)
- 5. (Optional) Visualize predictions

📌 Example: Prediction using Petal Length & Width only

```
python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
# Load dataset
iris = load_iris()
X = iris.data[:, 2:4] # Use only petal length and width
y = iris.target
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=200)
model.fit(x_train, y_train)
# Predict
y_pred = model.predict(x_test)
# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.2f}%")
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=iris.target_r
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=iris.target_names, yticklabels=iris.
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Optional Visualization of Predictions (2D scatter):

```
# Plot predictions
plt.scatter(x_test[:, 0], x_test[:, 1], c=y_pred, cmap='viridis', edgecolor='k', s=100)
plt.title("Predicted Classes (Petal Length vs Width)")
plt.xlabel("Petal Length (cm)")
plt.ylabel("Petal Width (cm)")
plt.colorbar(label='Predicted Class')
plt.show()
```

Would you like to include all 4 features for better accuracy or stick with petal-based predictions only?



```
#input vs target vs pridiction
plt.scatter(x_test, y_pred, alpha=0.7, label='x_Train', color='skyblue', edgecolor='black')
plt.title('hello')
plt.xlabel("input")
plt.ylabel("fuck")
plt.grid(True)
plt.show()
```

Your intention seems to be plotting **input vs target vs prediction**, but the code has a few issues that will cause errors or mislead the graph:

X Issues in your code:

- 1. `x_test` is a 2D array (e.g., shape `(30, 4)` in Iris dataset) `plt.scatter()` expects 1D x and y
 arrays of the same length.
- 2. The y-axis is `y_pred` (which is fine), but unless you're plotting only 1 feature from `x_test`, it won't work.
- 3. The label `'x_Train'` is confusing you're plotting test data.

Fix: Select one input feature (e.g., petal length) to compare with predictions

```
import matplotlib.pyplot as plt

# Let's say you're working with Iris and using petal length (feature index 2)
feature_index = 2
x_feature = x_test[:, feature_index] # e.g., petal length

plt.scatter(x_feature, y_pred, alpha=0.7, color='skyblue', edgecolor='black', label='Predicted')
plt.title('Input vs Prediction')
plt.xlabel("Petal Length (cm)")
plt.ylabel("Predicted Class")
plt.grid(True)
plt.legend()
plt.show()
```

Optional: Compare true vs predicted

```
python

plt.scatter(x_feature, y_test, color='green', alpha=0.5, label='Actual', marker='o')
plt.scatter(x_feature, y_pred, color='red', alpha=0.5, label='Predicted', marker='x')
plt.title("Actual vs Predicted Class (by Petal Length)")
plt.xlabel("Petal Length (cm)")
plt.ylabel("Class")
plt.legend()
plt.grid(True)
plt.show()
```

Would you like to plot all features in subplots or stick to one (like petal length)?



apply pca

Let's apply **PCA** (**Principal Component Analysis**) to reduce the dimensionality of the Iris dataset and **visualize the predictions** using the top 2 principal components.

Full Code: PCA + Prediction + Plot (Logistic Regression used here)

```
python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
# Load Iris dataset
iris = load iris()
X = iris.data
y = iris.target
# Apply PCA (reduce to 2 components for plotting)
pca = PCA(n components=2)
X_pca = pca.fit_transform(X)
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)
# Train classifier (e.g., logistic regression)
model = LogisticRegression()
model.fit(X_train, y_train)
# Predict
y_pred = model.predict(X_test)
# Accuracy
acc = accuracy_score(y_test, y_pred)
print(f"PCA + Logistic Regression Accuracy: {acc * 100:.2f}%")
print("\nClassification Report:\n", classification report(y test, y pred, target names=iris.target )
plt.figure(figsize=(8,6))
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', edgecolor='k', s=100, label='Pred:
plt.title('Predicted Classes after PCA (2D)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.grid(True)
plt.colorbar(label='Class Label')
plt.show()
```

What this code does:

- Reduces 4D Iris features to 2D using PCA.
- Trains a model on reduced features.
- Predicts and evaluates.
- Visualizes predictions on 2D PCA space.

Would you like to compare model accuracy with and without PCA or test it with SVM/KNN too?



apply lda



Let's now apply **LDA** (**Linear Discriminant Analysis**) on the Iris dataset, followed by classification and plotting the results.

What is LDA?

- LDA is a supervised dimensionality reduction technique.
- Unlike PCA (which maximizes variance), LDA maximizes class separability.
- LDA can reduce to **at most** `C 1` **components**, where `C` is the number of classes.

For the Iris dataset (3 classes), LDA can reduce to **2 dimensions**.

V Full Code: Apply LDA + Classification + Plot

```
python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import train_test_split
from sklearn.linear model import LogisticRegression
from sklearn.metrics import accuracy score, classification report
iris = load iris()
X = iris.data
y = iris.target
# Apply LDA (max n components = num classes - 1 = 2)
lda = LDA(n_components=2)
X lda = lda.fit transform(X, y)
# Split into train/test sets
X_train, X_test, y_train, y_test = train_test_split(X_lda, y, test_size=0.2, random_state=42)
# Train model
model = LogisticRegression()
model.fit(X_train, y_train)
# Predict
y pred = model.predict(X test)
# Evaluate
acc = accuracy_score(y_test, y_pred)
print(f"LDA + Logistic Regression Accuracy: {acc * 100:.2f}%")
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=iris.target_r
# Plot LDA-transformed data
plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis', edgecolor='k', s=100)
plt.title("Predicted Classes after LDA")
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.grid(True)
plt.colorbar(label='Class Label')
plt.show()
```

VS Compare with PCA:

Try this same plot after using PCA instead of LDA and see how well-separated the classes are.

• Typically, **LDA performs better** in supervised classification tasks because it uses label information.

Would you like a side-by-side comparison plot for PCA vs LDA?



classification by Ida by hyperplane ror line

To visualize classification using LDA with the decision boundary (hyperplane or line), we'll:

- 1. **Use LDA** for dimensionality reduction and classification.
- 2. **Train a model** (LDA itself or logistic regression).
- 3. **Plot the decision boundary line** in 2D (since Iris has 3 classes, LDA reduces it to 2D).

▼ Full Code: LDA Classification with Decision Boundary

```
python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Load data
iris = load iris()
X = iris.data
y = iris.target
target names = iris.target names
# Apply LDA
lda = LinearDiscriminantAnalysis(n_components=2)
X_lda = lda.fit_transform(X, y)
X_train, X_test, y_train, y_test = train_test_split(X_lda, y, test_size=0.2, random_state=42)
lda clf = LinearDiscriminantAnalysis()
lda_clf.fit(X_train, y_train)
# Predict & evaluate
y_pred = lda_clf.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred) * 100:.2f}%")
# Plot with decision boundary
def plot_decision_boundary(model, X, y, title):
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
     Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
     Z = Z.reshape(xx.shape)
     plt.figure(figsize=(8, 6))
     plt.contourf(xx, yy, Z, alpha=0.3, cmap='viridis')
scatter = plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap='viridis', s=80)
     plt.title(title)
     plt.xlabel('LD1')
plt.ylabel('LD2')
     plt.grid(True)
```



```
plt.legend(handles=scatter.legend_elements()[0], labels=target_names)
  plt.show()

plot_decision_boundary(lda_clf, X_lda, y, "LDA Classification with Decision Boundary")
```

Explanation:

- `LinearDiscriminantAnalysis()` acts as both:
 - Dimensionality reducer: using `fit_transform`
 - Classifier: when trained using `fit(X_train, y_train)`
- The `plot_decision_boundary()` function generates a meshgrid, predicts on it, and visualizes regions for each class.

Would you like a version using **SVM** on LDA-reduced data to compare decision boundaries?



apply Ida which show seperated data in one 1d, show both graph before and after

Great! Let's implement **Linear Discriminant Analysis (LDA)** to reduce the **Iris dataset** to **1D**, and visualize both:

✓ Goal:

- 1. **Before LDA**: Plot the original data in 2D using the first two features.
- 2. After LDA: Reduce data to 1D using LDA and plot the classes on a line (LD1 axis).

☑ Code: Iris Dataset → LDA (1D) → Compare Before & After

```
python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
# Load dataset
iris = load iris()
X = iris.data
v = iris.target
target_names = iris.target_names
  ----- 1. Plot original data in 2D (first two features) ------
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
for target, color in zip([0, 1, 2], ['red', 'green', 'blue']):
    plt.scatter(X[y == target, 0], X[y == target, 1],
label=target_names[target], color=color, edgecolor='k', s=60)
plt.title("Original Data (2D Feature Space)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.grid(True)
# ----- 2. Apply LDA (1D) ------
```

📌 Output:

- **Left Plot**: The original data using the first two features.
- **Right Plot**: Data projected into 1D (`LD1`) using LDA, showing clear class separation on a line.

This clearly shows how LDA **projects high-dimensional data** into a single axis to **maximize class separability**.

Would you like to save this plot or apply a classifier (e.g., Logistic Regression or SVM) on this 1D data?



similar to that give for pca

Absolutely! Let's do a similar process but using **Principal Component Analysis (PCA)** instead of LDA.

✓ Goal:

- 1. **Before PCA**: Plot the original Iris dataset in 2D using the first two features.
- 2. **After PCA**: Reduce the data to **1D using PCA** and plot the classes along the **PC1 axis**.

$lue{lue{V}}$ Code: Iris Dataset ightarrow PCA (1D) ightarrow Compare Before & After



Notes:

- PCA maximizes variance but doesn't consider class labels you'll notice classes may overlap more compared to LDA.
- This helps illustrate the difference in purpose:
 - LDA = Best for class separation (supervised)
 - **PCA** = Best for **dimensionality reduction** preserving variance (unsupervised)

Would you like to compare LDA vs PCA classification accuracy next using Logistic Regression or SVM?



- 3. Creation, loading and handling different datasets in Python, including normalization etc.
- 4. Load breast cancer dataset and perform classification using Euclidean distance. Use 70% data as training and 30% for testing. Repeat the above experiment with 10-fold cross validation and find the standard deviation in accuracy. Also d build the confusion matrix and derive Precision, Recall and Specificity of the algorithm.
- 5. Write a program to implement the Linear Regression algorithm for a sample training data set stored as a .CSV file. Compute the prediction accuracy of the classifier, considering few test data sets.
- 6. Write a program to implement the Logistic Regression for a sample training data set stored as a .CSV file. Compute the classification accuracy of the classifier, considering few test data sets.



- 7. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Calculate the accuracy, precision, and recall for your data set.
- 8. Write a program to implement the KNN classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.
- 9. Write a program for building a model by using step forward method as a feature selection and backward elimination method as a feature selection and compare the result.
- 10. Reduce the dimensionality of a high dimensional dataset using PCA and LDA. Compare the result of
- 11. Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
- 12. Compare the classification Accuracy of the decision tree and SVM algorithm. Use an appropriate data set for building the models.
- 13. Demonstrate the efficiency of different SVM kernels in classification of a suitable real life dataset.

do all program in single docs, make sure all remain seperate

plot the visiuualisation matplotlib

Ml Classification Projects

I've created a single Python script containing all the programs you requested, each in its own clearly labeled section. It includes visualizations using Matplotlib and uses real or placeholder datasets for tasks such as classification, regression, dimensionality reduction, and feature selection.

Would you like me to also export this as a `.py` or `.ipynb` file?



write a program for svm, pca, lda, decission tree, ridge and lasso regression show there seperate plots

```
MI Classification Projects
### SVM Classifier with Plot
```python
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```



```
Load dataset
iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
Train SVM
clf = SVC(kernel='linear')
clf.fit(X_train, y_train)
Predict & Accuracy
y_pred = clf.predict(X_test)
print("SVM Accuracy:", accuracy_score(y_test, y_pred))
Plot
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_pred, cmap='viridis')
plt.title("SVM Classification")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.show()
PCA Visualization
```python
from sklearn.decomposition import PCA
```



I've added separate code sections for SVM, PCA, LDA, Decision Tree, Ridge, and Lasso Regression, each with their own plots. Let me know if you want combined comparisons or evaluation metrics added.

Follow up

