

Decision Tree Solutions



AML ML Practical List.pdf
PDF

Give me the solutions to all the questions in the PDF. Give a precise and simple solution, do not complicate it

Below is one possible “precise and simple” set of solution-templates for all 70 practicals. (Each solution is meant to be a starting point—you may need to adjust file names, parameters, or minor details to suit your local setup.)

1. Decision Tree on diabetes.csv (max_depth=3)

python

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

df = pd.read_csv('diabetes.csv')
X = df.drop('label', axis=1)
y = df['label']

clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X, y)

plt.figure(figsize=(12,8))
plot_tree(clf, feature_names=X.columns, filled=True)
plt.show()
```

2. Decision Tree with Feature Importance on diabetes.csv

python

```

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('diabetes.csv')
X = df.drop('label', axis=1)
y = df['label']

clf = DecisionTreeClassifier(random_state=42)
clf.fit(X, y)

importances = clf.feature_importances_
sns.barplot(x=X.columns, y=importances)
plt.title('Feature Importances')
plt.show()

```

3. Compare Decision Tree vs. SVM on Iris dataset

python

```

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
acc_dt = accuracy_score(y_test, dt.predict(X_test))

svm = SVC(random_state=42)
svm.fit(X_train, y_train)

```

```
acc_svm = accuracy_score(y_test, svm.predict(X_test))
```

```
print("Decision Tree Accuracy:", acc_dt)
```

```
print("SVM Accuracy:", acc_svm)
```

4. Two Decision Trees on Iris (max_leaf_nodes=8 vs. min_samples_leaf=5)

python

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

dt1 = DecisionTreeClassifier(max_leaf_nodes=8, random_state=42)
acc1 = accuracy_score(y_test, dt1.fit(X_train, y_train).predict(X_test))

dt2 = DecisionTreeClassifier(min_samples_leaf=5, random_state=42)
acc2 = accuracy_score(y_test, dt2.fit(X_train, y_train).predict(X_test))

print("Max Leaf Nodes=8 Accuracy:", acc1)
print("Min Samples per Leaf=5 Accuracy:", acc2)
```

5. Decision Tree vs. Logistic Regression on diabetes.csv

python

```
import pandas as pd
from sklearn.model_selection import train_test_split
```

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df = pd.read_csv('diabetes.csv')
X = df.drop('label', axis=1)
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

dt = DecisionTreeClassifier(random_state=42)
lr = LogisticRegression(max_iter=1000)

dt.fit(X_train, y_train)
lr.fit(X_train, y_train)

print("Decision Tree Accuracy:", accuracy_score(y_test, dt.predict(X_test)))
print("Logistic Regression Accuracy:", accuracy_score(y_test, lr.predict(X_test)))

```

6. Random dataset (350×3) – Compare DT, RF, and Logistic Regression

python

```

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

np.random.seed(42)
data = np.random.rand(350, 3)
df = pd.DataFrame(data, columns=['A', 'B', 'C'])
df['target'] = (df.sum(axis=1) > 1.5).astype(int)

X = df[['A', 'B', 'C']]
y = df['target']

```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

models = {
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Logistic Regression': LogisticRegression(max_iter=1000)
}

accuracies = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    accuracies[name] = accuracy_score(y_test, model.predict(X_test))

plt.bar(accuracies.keys(), accuracies.values(), color=['blue', 'green', 'red'])
plt.ylabel('Accuracy')
plt.title('Model Comparison')
plt.ylim(0,1)
plt.show()

print(accuracies)

```

7. Plot ccp_alpha vs. Tree Depth on diabetes.csv

python

```

import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

df = pd.read_csv('diabetes.csv')
X = df.drop('label', axis=1)
y = df['label']

clf = DecisionTreeClassifier(random_state=42)
path = clf.cost_complexity_pruning_path(X, y)
ccp_alphas = path.ccp_alphas

depths = []

```

```

for alpha in ccp_alphas:
    clf_temp = DecisionTreeClassifier(random_state=42, ccp_alpha=alpha)
    clf_temp.fit(X, y)
    depths.append(clf_temp.get_depth())

plt.plot(ccp_alphas, depths, marker='o')
plt.xlabel("ccp_alpha")
plt.ylabel("Tree Depth")
plt.title("ccp_alpha vs Tree Depth")
plt.show()

```

8. Elbow Method on a 14-point dataset

python

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

np.random.seed(42)
X = np.random.rand(14, 2)

sse = []
for k in range(1, 10):
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X)
    sse.append(km.inertia_)

plt.plot(range(1, 10), sse, marker='o')
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.title("Elbow Method")
plt.show()

```

9. K-Medoids clustering on synthetic urban area data

(Requires the `scikit-learn-extra` package.)

python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn_extra.cluster import KMedoids

np.random.seed(42)
df = pd.DataFrame({
    'population_density': np.random.randint(1000, 10000, 100),
    'average_income': np.random.randint(30000, 100000, 100),
    'amenities': np.random.randint(1, 50, 100)
})

X = df.values
kmedoids = KMedoids(n_clusters=4, random_state=42)
df['cluster'] = kmedoids.fit_predict(X)

plt.scatter(df['population_density'], df['average_income'], c=df['cluster'],
            cmap='viridis')
plt.xlabel("Population Density")
plt.ylabel("Average Income")
plt.title("Urban Areas Clustering (K-Medoids)")
plt.show()
```

10. K-Medoids clustering on customer data (age, spending_score)

python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn_extra.cluster import KMedoids

np.random.seed(42)
df = pd.DataFrame({
```

```

    'age': np.random.randint(18, 70, 100),
    'spending_score': np.random.randint(1, 100, 100)
})

X = df.values
kmedoids = KMedoids(n_clusters=3, random_state=42)
df['cluster'] = kmedoids.fit_predict(X)

plt.scatter(df['age'], df['spending_score'], c=df['cluster'], cmap='plasma')
plt.xlabel("Age")
plt.ylabel("Spending Score")
plt.title("Customer Segmentation (K-Medoids)")
plt.show()

print(df.groupby('cluster').mean())

```

11. K-means clustering on e-commerce customer data

python

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

np.random.seed(42)
df = pd.DataFrame({
    'total_spending': np.random.randint(100, 1000, 100),
    'frequency': np.random.randint(1, 50, 100),
    'basket_size': np.random.randint(1, 20, 100)
})

X = df.values
kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(X)

plt.scatter(df['total_spending'], df['frequency'], c=df['cluster'], cmap='Accent')
plt.xlabel("Total Spending")
plt.ylabel("Frequency")

```



```
plt.title("E-commerce Customer Segmentation (K-Means)")
plt.show()

print(df.groupby('cluster').mean())
```

12. K-means on make_blobs with clusters 2-7

python

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

X, _ = make_blobs(n_samples=300, centers=4, random_state=42)
cluster_range = [2,3,4,5,6,7]

plt.figure(figsize=(12,8))
for i, k in enumerate(cluster_range, 1):
    plt.subplot(2,3,i)
    y_pred = KMeans(n_clusters=k, random_state=42).fit_predict(X)
    plt.scatter(X[:,0], X[:,1], c=y_pred, cmap='viridis')
    plt.title(f"{k} Clusters")
plt.tight_layout()
plt.show()
```

13. K-means clustering on simulated grocery transactions

python

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
import seaborn as sns
import matplotlib.pyplot as plt
```

```

np.random.seed(42)
# Simulate 100 transactions over 10 products (binary purchase matrix)
df = pd.DataFrame(np.random.randint(0,2,(100,10)),
                  columns=[f'Product_{i+1}' for i in range(10)])
df['cluster'] = KMeans(n_clusters=3, random_state=42).fit_predict(df)

print(df.groupby('cluster').mean())
sns.heatmap(df.groupby('cluster').mean(), annot=True)
plt.title("Average Purchase Pattern by Cluster")
plt.show()

```

14. K-means on 35-point dataset; impact of varying cluster count

python

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

np.random.seed(42)
X = np.random.rand(35, 2)
cluster_range = [2,3,4,5]

plt.figure(figsize=(12,8))
for i, k in enumerate(cluster_range, 1):
    plt.subplot(2,2,i)
    y_pred = KMeans(n_clusters=k, random_state=42).fit_predict(X)
    plt.scatter(X[:,0], X[:,1], c=y_pred, cmap='Set1')
    plt.title(f"{k} Clusters")
plt.tight_layout()
plt.show()

# Note: Increasing k may capture more detail but risks overfitting to noise.

```

15. DBSCAN clustering on make_moons dataset

python

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN

X, _ = make_moons(n_samples=300, noise=0.05, random_state=42)
labels = DBSCAN(eps=0.3, min_samples=5).fit_predict(X)

plt.scatter(X[:,0], X[:,1], c=labels, cmap='plasma')
plt.title("DBSCAN on make_moons")
plt.show()
```

16. DBSCAN on selected columns (3rd & 4th) from random data

python

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN

np.random.seed(42)
data = np.random.rand(100, 5) * 10
X = data[:,2:4]
labels = DBSCAN(eps=5, min_samples=5).fit_predict(X)

plt.scatter(X[:,0], X[:,1], c=labels, cmap='coolwarm')
plt.xlabel("Feature 3")
plt.ylabel("Feature 4")
plt.title("DBSCAN on Selected Columns")
plt.show()
```

17. DBSCAN for customer segmentation (frequency & transaction amount)

python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN

np.random.seed(42)
df = pd.DataFrame({
    'frequency': np.random.randint(1, 50, 100),
    'transaction_amount': np.random.randint(10, 500, 100)
})
df['cluster'] = DBSCAN(eps=50,
min_samples=5).fit_predict(df[['frequency', 'transaction_amount']])

plt.scatter(df['frequency'], df['transaction_amount'], c=df['cluster'],
cmap='viridis')
plt.xlabel("Frequency")
plt.ylabel("Transaction Amount")
plt.title("DBSCAN on Customer Data")
plt.show()

print(df.groupby('cluster').mean())
```

18. DBSCAN parametric analysis on 2-feature dataset

python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN

np.random.seed(42)
df = pd.DataFrame({
    'spending': np.random.randint(100, 1000, 100),
    'feature': np.random.randint(1, 100, 100)
})
X = df.values
```

```

eps_values = [10, 20, 30, 40, 50]
n_clusters = []
for eps in eps_values:
    labels = DBSCAN(eps=eps, min_samples=5).fit_predict(X)
    clusters = len(set(labels)) - (1 if -1 in labels else 0)
    n_clusters.append(clusters)

plt.plot(eps_values, n_clusters, marker='o')
plt.xlabel("eps value")
plt.ylabel("Number of clusters")
plt.title("Effect of eps on DBSCAN")
plt.show()

```

19. DBSCAN: effect of varying eps (1–5) and min_samples on 1st & 3rd columns

python

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN

np.random.seed(42)
data = np.random.rand(100, 4) * 10
X = data[:, [0,2]]

# Vary eps from 1 to 5
eps_range = [1, 2, 3, 4, 5]
clusters_eps = []
for eps in eps_range:
    labels = DBSCAN(eps=eps, min_samples=5).fit_predict(X)
    clusters = len(set(labels)) - (1 if -1 in labels else 0)
    clusters_eps.append(clusters)

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(eps_range, clusters_eps, marker='o')

```

```

plt.xlabel("eps")
plt.ylabel("Clusters")
plt.title("Effect of eps on DBSCAN")

# Vary min_samples
min_samples_range = [3, 5, 7, 10]
clusters_minpts = []
for m in min_samples_range:
    labels = DBSCAN(eps=3, min_samples=m).fit_predict(X)
    clusters = len(set(labels)) - (1 if -1 in labels else 0)
    clusters_minpts.append(clusters)

plt.subplot(1,2,2)
plt.plot(min_samples_range, clusters_minpts, marker='o', color='red')
plt.xlabel("min_samples")
plt.ylabel("Clusters")
plt.title("Effect of min_samples on DBSCAN")
plt.tight_layout()
plt.show()

```

20. Mall Customer Clustering – DBSCAN & Agglomerative (with dendrogram)

python

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN, AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram, linkage

np.random.seed(42)
n = 200
df = pd.DataFrame({
    'CustomerID': range(1, n+1),
    'Genre': np.random.choice(['Male', 'Female'], n),
    'Age': np.random.randint(18, 70, n),
    'AnnualIncome': np.random.randint(15, 150, n),

```

```

    'SpendingScore': np.random.randint(1, 100, n)
})
X = df[['Age', 'AnnualIncome', 'SpendingScore']].values

# DBSCAN clustering
df['DBSCAN_cluster'] = DBSCAN(eps=5, min_samples=5).fit_predict(X)
plt.figure()
plt.scatter(df['AnnualIncome'], df['SpendingScore'], c=df['DBSCAN_cluster'],
            cmap='Accent')
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score")
plt.title("DBSCAN Clustering")
plt.show()

# Dendrogram using Agglomerative Clustering
linked = linkage(X, method='ward')
plt.figure(figsize=(10,7))
dendrogram(linked)
plt.title("Agglomerative Clustering Dendrogram")
plt.xlabel("Samples")
plt.ylabel("Distance")
plt.show()

```

21. k-Fold CV on SVM (synthetic dataset)

python

```

from sklearn.datasets import make_classification
from sklearn.model_selection import KFold, cross_val_score
from sklearn.svm import SVC

X, y = make_classification(n_samples=100, n_features=20, random_state=42)
svm = SVC(kernel='rbf', gamma='scale', random_state=42)
kf = KFold(n_splits=10, shuffle=True, random_state=42)
scores = cross_val_score(svm, X, y, cv=kf)

print("Fold Accuracies:", scores)
print("Average Accuracy:", scores.mean())

```

22. Bootstrap Sampling with Random Forest on synthetic data

python

```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

np.random.seed(42)
X = np.random.rand(150, 10)
y = np.random.choice([0,1,2], 150)

n_bootstraps = 10
accuracies = []
for i in range(n_bootstraps):
    indices = np.random.choice(range(150), size=150, replace=True)
    X_boot = X[indices]
    y_boot = y[indices]
    rf = RandomForestClassifier(random_state=42)
    rf.fit(X_boot, y_boot)
    acc = accuracy_score(y, rf.predict(X))
    accuracies.append(acc)
    print(f"Bootstrap {i+1} Accuracy: {acc}")
print("Average Accuracy:", np.mean(accuracies))
```

23. LOOCV with Decision Tree on 100×5 dataset (binary target)

python

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import accuracy_score

np.random.seed(42)
X = np.random.rand(100, 5)
y = np.random.choice([0,1], 100)
```



```

loo = LeaveOneOut()
predictions = []
for train_idx, test_idx in loo.split(X):
    clf = DecisionTreeClassifier(random_state=42)
    clf.fit(X[train_idx], y[train_idx])
    predictions.append(clf.predict(X[test_idx])[0])

print("LOOCV Accuracy:", accuracy_score(y, predictions))

```

24. LOOCV with Decision Tree on 150×10 dataset (3 classes, noisy)

python

```

import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import LeaveOneOut
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

np.random.seed(42)
X = np.random.rand(150, 10)
y = np.random.choice([0,1,2], 150)
X_noisy = X + np.random.normal(0, 0.1, X.shape)

loo = LeaveOneOut()
preds = []
for train_idx, test_idx in loo.split(X_noisy):
    clf = DecisionTreeClassifier(random_state=42)
    clf.fit(X_noisy[train_idx], y[train_idx])
    preds.append(clf.predict(X_noisy[test_idx])[0])

print("LOOCV Accuracy:", accuracy_score(y, preds))
print("Confusion Matrix:\n", confusion_matrix(y, preds))
print("Classification Report:\n", classification_report(y, preds))

```

25. Stratified K-Fold CV on Random Forest (200×15, 4 classes with imbalance)

python

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

X, y = make_classification(n_samples=200, n_features=15, n_classes=4,
                          weights=[0.5, 0.2, 0.2, 0.1], random_state=42)
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
preds_all = np.zeros_like(y)
accs = []

for train_idx, test_idx in skf.split(X, y):
    clf = RandomForestClassifier(random_state=42)
    clf.fit(X[train_idx], y[train_idx])
    y_pred = clf.predict(X[test_idx])
    accs.append(np.mean(y_pred==y[test_idx]))
    preds_all[test_idx] = y_pred

print("Fold Accuracies:", accs)
print("Average Accuracy:", np.mean(accs))
print("Classification Report:\n", classification_report(y, preds_all))
cm = confusion_matrix(y, preds_all)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.show()
```

26. Grid Search CV on Iris with SVM

python

```

from sklearn.datasets import load_iris
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

iris = load_iris()
X, y = iris.data, iris.target
param_grid = {'C': [0.1, 1, 10, 100],
              'kernel': ['linear', 'rbf'],
              'gamma': ['scale', 'auto']}
grid = GridSearchCV(SVC(random_state=42), param_grid, cv=5)
grid.fit(X, y)
print("Best Hyperparameters:", grid.best_params_)
best_model = grid.best_estimator_
y_pred = best_model.predict(X)
print("Accuracy:", accuracy_score(y, y_pred))
cm = confusion_matrix(y, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='viridis')
plt.title("Confusion Matrix (Best Model)")
plt.show()

```

27. PCA on Iris (reduce to 2D and plot)

python

```

from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

iris = load_iris()
X, y = iris.data, iris.target
X_pca = PCA(n_components=2).fit_transform(X)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='Set1')
plt.xlabel("PC1")
plt.ylabel("PC2")

```

```
plt.title("PCA of Iris Dataset")
plt.show()
```

28. Curse of Dimensionality with k-NN (varying dimensions)

python

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

dims = [2, 5, 10, 20, 50, 100]
accuracies = []
for d in dims:
    X, y = make_classification(n_samples=1000, n_features=d, random_state=42)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)
    accuracies.append(knn.score(X_test, y_test))

plt.plot(dims, accuracies, marker='o')
plt.xlabel("Number of Dimensions")
plt.ylabel("Accuracy")
plt.title("k-NN Accuracy vs. Dimensionality")
plt.show()
```

29. Impact of increasing feature dimensions (SVM discussion)

python

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

```
# Simple bar plot to illustrate volume increase
dims = ['3D', '12D']
volumes = [1**3, 1**12] # Conceptual: volume increases exponentially

plt.bar(dims, volumes, color=['blue', 'orange'])
plt.ylabel("Relative Volume")
plt.title("Increase in Feature Space Volume")
plt.show()

print("""Explanation:
Increasing from 3 to 12 dimensions expands the feature space volume exponentially.
This leads to sparser data, increased risk of overfitting, higher computational
cost,
and diminished meaning of distance metrics (curse of dimensionality).""")
```

30. Synthetic blobs with 4 features, PCA to 3D, 3D plots side by side

python

```
from sklearn.datasets import make_blobs
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

X, y = make_blobs(n_samples=300, centers=4, n_features=4, random_state=42)
X_pca = PCA(n_components=3).fit_transform(X)

fig = plt.figure(figsize=(12,6))
ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(X[:,0], X[:,1], X[:,2], c=y, cmap='viridis')
ax1.set_title("Original Data (First 3 Features)")

ax2 = fig.add_subplot(122, projection='3d')
ax2.scatter(X_pca[:,0], X_pca[:,1], X_pca[:,2], c=y, cmap='viridis')
ax2.set_title("PCA Reduced Data")
plt.show()
```

31. Random Forest on Breast Cancer dataset with evaluation

python

```
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report

data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1-score:", f1_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

32. k-NN on Wine dataset (from CSV) with performance metrics

python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

df = pd.read_csv('wine.csv') # ensure your CSV has a 'target' column
X = df.drop('target', axis=1)
y = df['target']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,  
random_state=42)
```

```
knn = KNeighborsClassifier()  
knn.fit(X_train, y_train)  
y_pred = knn.predict(X_test)  
print("Classification Report:\n", classification_report(y_test, y_pred))
```

33. PCA on Iris (4→2 dimensions, 2D scatter)

python

```
from sklearn.datasets import load_iris  
from sklearn.decomposition import PCA  
import matplotlib.pyplot as plt  
  
iris = load_iris()  
X, y = iris.data, iris.target  
X_pca = PCA(n_components=2).fit_transform(X)  
  
plt.scatter(X_pca[:,0], X_pca[:,1], c=y, cmap='Set1')  
plt.xlabel("PC1")  
plt.ylabel("PC2")  
plt.title("PCA on Iris Dataset")  
plt.show()
```

34. LDA on Iris (4→2 dimensions, 2D scatter)

python

```
from sklearn.datasets import load_iris  
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA  
import matplotlib.pyplot as plt  
  
iris = load_iris()
```

```
X, y = iris.data, iris.target
X_lda = LDA(n_components=2).fit_transform(X, y)

plt.scatter(X_lda[:,0], X_lda[:,1], c=y, cmap='Set1')
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.title("LDA on Iris Dataset")
plt.show()
```

35. LDA on Wine dataset with visualization and evaluation

python

```
import pandas as pd
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

df = pd.read_csv('wine.csv')
X = df.drop('target', axis=1)
y = df['target']

X_lda = LDA(n_components=2).fit_transform(X, y)
plt.scatter(X_lda[:,0], X_lda[:,1], c=y, cmap='viridis')
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.title("LDA on Wine Dataset")
plt.show()

# Simple evaluation using Logistic Regression
X_train, X_test, y_train, y_test = train_test_split(X_lda, y, test_size=0.3,
random_state=42)
clf = LogisticRegression()
clf.fit(X_train, y_train)
print(classification_report(y_test, clf.predict(X_test)))
```


36. LDA on Iris (with label encoding if needed) and 2D plot

python

```
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.pyplot as plt

iris = load_iris()
X, y = iris.data, iris.target
X_lda = LDA(n_components=2).fit_transform(X, y)

plt.scatter(X_lda[:,0], X_lda[:,1], c=y, cmap='rainbow')
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.title("LDA on Iris Dataset")
plt.show()
```

37. LDA on Breast Cancer (reduce to 1D and plot)

python

```
from sklearn.datasets import load_breast_cancer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.pyplot as plt

data = load_breast_cancer()
X, y = data.data, data.target
X_lda = LDA(n_components=1).fit_transform(X, y)

plt.scatter(X_lda, [0]*len(X_lda), c=y, cmap='coolwarm', marker='o')
plt.xlabel("LD1")
plt.title("LDA on Breast Cancer Dataset")
plt.show()
```

38. Compare PCA and LDA on Iris dataset

python

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.pyplot as plt

iris = load_iris()
X, y = iris.data, iris.target

X_pca = PCA(n_components=2).fit_transform(X)
X_lda = LDA(n_components=2).fit_transform(X, y)

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.scatter(X_pca[:,0], X_pca[:,1], c=y, cmap='Set1')
plt.title("PCA on Iris")
plt.xlabel("PC1")
plt.ylabel("PC2")

plt.subplot(1,2,2)
plt.scatter(X_lda[:,0], X_lda[:,1], c=y, cmap='Set1')
plt.title("LDA on Iris")
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.tight_layout()
plt.show()
```

39. Linear Regression for House Prices (original vs. PCA-reduced)

python

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
```

```

df = pd.read_csv('HousePrices.csv')
# Example: use square_feet and bedrooms as features
X = df[['square_feet', 'bedrooms']]
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Original data
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print("Original Data Metrics:")
print("MAE:", mean_absolute_error(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))

# PCA reduced model
pca = PCA(n_components=1)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
lr_pca = LinearRegression()
lr_pca.fit(X_train_pca, y_train)
y_pred_pca = lr_pca.predict(X_test_pca)
print("\nPCA Reduced Data Metrics:")
print("MAE:", mean_absolute_error(y_test, y_pred_pca))
print("MSE:", mean_squared_error(y_test, y_pred_pca))
print("R2:", r2_score(y_test, y_pred_pca))

```

40. PCA then Regression on House Prices

python

```

import pandas as pd
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

```

```

df = pd.read_csv('HousePrices.csv')
X = df.drop('price', axis=1)
y = df['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

lr = LinearRegression()
lr.fit(X_train_pca, y_train)
y_pred = lr.predict(X_test_pca)
print("MAE:", mean_absolute_error(y_test, y_pred))
print("MSE:", mean_squared_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))

```

41. Two SVC models on Social_Netwok_Ads.csv (linear & RBF)

python

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

df = pd.read_csv('Social_Netwok_Ads.csv')
X = df[['Age', 'EstimatedSalary']]
y = df['Purchased']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

svc_linear = SVC(kernel='linear', random_state=42)
svc_linear.fit(X_train, y_train)
acc_linear = accuracy_score(y_test, svc_linear.predict(X_test))

svc_rbf = SVC(kernel='rbf', random_state=42)
svc_rbf.fit(X_train, y_train)
acc_rbf = accuracy_score(y_test, svc_rbf.predict(X_test))

```

```
print("Linear SVC Accuracy:", acc_linear)
print("RBF SVC Accuracy:", acc_rbf)
```

42. SVC on Breast Cancer with PCA visualization

python

```
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

svc = SVC(kernel='rbf', probability=True, random_state=42)
svc.fit(X_train, y_train)
print("Test Accuracy:", svc.score(X_test, y_test))

pca = PCA(n_components=2)
X_vis = pca.fit_transform(X_test)
plt.scatter(X_vis[:,0], X_vis[:,1], c=svc.predict(X_test), cmap='coolwarm')
plt.title("SVC Predictions (PCA Visualization)")
plt.show()
```

43. SVM for Breast Cancer with StandardScaler & Grid Search

python

```
from sklearn.datasets import load_breast_cancer
from sklearn.svm import SVC
```

```

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report

data = load_breast_cancer()
X, y = data.data, data.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=42)
param_grid = {'C': [0.1, 1, 10], 'kernel': ['linear','rbf'], 'gamma':
['scale','auto']}
grid = GridSearchCV(SVC(random_state=42), param_grid, cv=5)
grid.fit(X_train, y_train)
print("Best Params:", grid.best_params_)
y_pred = grid.predict(X_test)
print(classification_report(y_test, y_pred))

```

44. Gaussian Naive Bayes on Breast Cancer dataset

python

```

from sklearn.datasets import load_breast_cancer
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

gnb = GaussianNB()
gnb.fit(X_train, y_train)
print("Accuracy:", accuracy_score(y_test, gnb.predict(X_test)))

```

45. Gaussian Naive Bayes on PIMA.csv for diabetes prediction

python

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

df = pd.read_csv('PIMA.csv')
print(df.describe())
X = df.drop('Diabetes', axis=1)
y = df['Diabetes']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

gnb = GaussianNB()
gnb.fit(X_train, y_train)
print("Accuracy:", accuracy_score(y_test, gnb.predict(X_test)))

# Note: Check for biases (e.g., imbalanced classes, demographic factors) that might
affect performance.
```

46. Sentiment Analysis on IMDB_Dataset.csv with Multinomial Naive Bayes

python

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

df = pd.read_csv('IMDB_Dataset.csv')
vectorizer = TfidfVectorizer(stop_words='english')
```

```
X = vectorizer.fit_transform(df['Review'])
y = df['Sentiment'].map({'positive':1, 'negative':0})
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

nb = MultinomialNB()
nb.fit(X_train, y_train)
print("Accuracy:", accuracy_score(y_test, nb.predict(X_test)))
```

47. Multinomial Naive Bayes on 20 Newsgroups dataset

python

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

newsgroups = fetch_20newsgroups(subset='all')
vectorizer = CountVectorizer(stop_words='english')
X = vectorizer.fit_transform(newsgroups.data)
y = newsgroups.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

nb = MultinomialNB()
nb.fit(X_train, y_train)
print("Accuracy:", accuracy_score(y_test, nb.predict(X_test)))
```

48. Logistic Regression for Churn Prediction (small dataset)

python


```

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = {
    'Age': [25, 34, 45, 29, 50, 38, 42, 35, 48, 55],
    'Account_Length': [12, 24, 36, 18, 48, 30, 42, 24, 36, 60],
    'Monthly_Charges': [70, 90, 80, 100, 60, 80, 90, 70, 80, 100],
    'Churn': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]
}
df = pd.DataFrame(data)
X = df[['Age', 'Account_Length', 'Monthly_Charges']]
y = df['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

lr = LogisticRegression()
lr.fit(X_train, y_train)
print("Accuracy:", accuracy_score(y_test, lr.predict(X_test)))

```

49. Logistic Regression on customer_churn.csv

python

```

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('customer_churn.csv')
le = LabelEncoder()
df['gender'] = le.fit_transform(df['gender'])
df['international_plan'] = le.fit_transform(df['international_plan'])
df['voice_mail_plan'] = le.fit_transform(df['voice_mail_plan'])
df['churn'] = le.fit_transform(df['churn'])

```

```
X = df.drop(['customer_id', 'churn'], axis=1)
y = df['churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
print("Accuracy:", accuracy_score(y_test, lr.predict(X_test)))
```

50. Iris binary classification (Iris-Virginica vs. rest) with threshold 0.6

python

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np

iris = load_iris()
X, y = iris.data, iris.target
y_bin = (y == 2).astype(int) # 1 if Iris-Virginica

X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size=0.3,
random_state=42)
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
probs = lr.predict_proba(X_test)[:,:1]
preds = (probs >= 0.6).astype(int)

print("Predicted probabilities:", probs)
print("Predictions (threshold=0.6):", preds)
```

51. Simple Linear Regression on Salary.csv with residual plot

python

```

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import seaborn as sns

df = pd.read_csv('Salary.csv')
X = df[['years_of_experience']]
y = df['salary']

lr = LinearRegression()
lr.fit(X, y)
y_pred = lr.predict(X)
df['residuals'] = y - y_pred

plt.figure(figsize=(14,5))
plt.subplot(1,2,1)
plt.scatter(X, y, color='blue')
plt.plot(X, y_pred, color='red')
plt.title("Fitted Model")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")

plt.subplot(1,2,2)
sns.residplot(x=y_pred, y=df['residuals'], lowess=True)
plt.title("Residual Plot")
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.show()

```

52. Polynomial Regression on HousePrices.csv (square_feet vs. price)

python

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

```

```

from sklearn.metrics import mean_squared_error

df = pd.read_csv('HousePrices.csv')
X = df[['square_feet']]
y = df['price']

poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)
model = LinearRegression()
model.fit(X_poly, y)
y_pred = model.predict(X_poly)
residuals = y - y_pred

plt.figure(figsize=(14,5))
plt.subplot(1,2,1)
plt.scatter(X, y, color='blue')
plt.plot(X, y_pred, color='red')
plt.title("Polynomial Regression Fit")
plt.xlabel("Square Feet")
plt.ylabel("Price")

plt.subplot(1,2,2)
plt.scatter(y_pred, residuals)
plt.hlines(0, min(y_pred), max(y_pred), colors='red')
plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()

```

53. Multiple Linear Regression on CarPrices.csv (age, mileage vs. price)

python

```

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
df = pd.read_csv('CarPrices.csv')
X = df[['age', 'mileage']]

```

```

y = df['price']
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
residuals = y - y_pred

plt.figure(figsize=(14,5))
plt.subplot(1,2,1)
plt.scatter(y, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs. Predicted")
plt.subplot(1,2,2)
plt.scatter(y_pred, residuals)
plt.hlines(0, min(y_pred), max(y_pred), colors='red')
plt.xlabel("Predicted Prices")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()

```

54. Linear Regression via Gradient Descent on HousePrices.csv (size vs. price)

python

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = pd.read_csv('HousePrices.csv')
X = df['size'].values.reshape(-1,1)
y = df['price'].values

m, b = 0, 0
learning_rate = 0.0001
epochs = 1000

for _ in range(epochs):

```

```

y_pred = m * X + b
error = y_pred.flatten() - y
m_grad = (2/len(X)) * np.sum(error * X.flatten())
b_grad = (2/len(X)) * np.sum(error)
m -= learning_rate * m_grad
b -= learning_rate * b_grad

print("Slope:", m, "Intercept:", b)
plt.scatter(X, y, color='blue')
plt.plot(X, m*X+b, color='red')
plt.title("Gradient Descent Linear Regression")
plt.xlabel("Size")
plt.ylabel("Price")
plt.show()

```

55. Linear Regression on CarPrices.csv (horsepower, age vs. price) evaluation

python

```

import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

df = pd.read_csv('CarPrices.csv')
X = df[['horsepower', 'age']]
y = df['price']
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

print("MSE:", mean_squared_error(y, y_pred))
print("R2:", r2_score(y, y_pred))
print("MAE:", mean_absolute_error(y, y_pred))

```

56. Linear Regression on HouseData.csv (num_rooms, square_feet, location vs. house_price)

python

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

df = pd.read_csv('HouseData.csv')
X = df[['num_rooms', 'square_feet', 'location']]
y = df['house_price']
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)

print("MSE:", mean_squared_error(y, y_pred))
print("R2:", r2_score(y, y_pred))
print("MAE:", mean_absolute_error(y, y_pred))
```

57. Linear Regression on EmployeeData.csv (experience, education_level vs. salary)

python

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv('EmployeeData.csv')
le = LabelEncoder()
df['education_level'] = le.fit_transform(df['education_level'])
X = df[['years_of_experience', 'education_level']]
y = df['salary']
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
```

```
print("MSE:", mean_squared_error(y, y_pred))
print("R2:", r2_score(y, y_pred))
print("MAE:", mean_absolute_error(y, y_pred))
```

58. Lasso Regression on HousingData.csv (num_rooms, area, age vs. price)

python

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
import numpy as np

df = pd.read_csv('HousingData.csv')
X = df[['num_rooms', 'area', 'age']]
y = df['price']

lasso = Lasso(alpha=0.1)
lasso.fit(X, y)
y_pred = lasso.predict(X)
residuals = y - y_pred

print("Coefficients:", lasso.coef_)
plt.bar(X.columns, lasso.coef_)
plt.title("Lasso Coefficients")
plt.show()
plt.scatter(y_pred, residuals)
plt.hlines(0, min(y_pred), max(y_pred), colors='red')
plt.xlabel("Predicted Price")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```


59. Ridge Regression on Diabetes.csv (all features vs. diabetes_progression)

python

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
import numpy as np
from sklearn.metrics import mean_squared_error

df = pd.read_csv('Diabetes.csv')
X = df.drop('diabetes_progression', axis=1)
y = df['diabetes_progression']

ridge = Ridge(alpha=1.0)
ridge.fit(X, y)
y_pred = ridge.predict(X)
residuals = y - y_pred

print("Coefficients:", ridge.coef_)
plt.bar(X.columns, ridge.coef_)
plt.title("Ridge Coefficients")
plt.show()
plt.scatter(y_pred, residuals)
plt.hlines(0, min(y_pred), max(y_pred), colors='red')
plt.xlabel("Predicted Progression")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

60. Lasso Regression on WineQuality.csv (all features vs. quality)

python

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
import numpy as np
```

```

from sklearn.metrics import mean_squared_error

df = pd.read_csv('WineQuality.csv')
X = df.drop('quality', axis=1)
y = df['quality']

lasso = Lasso(alpha=0.1)
lasso.fit(X, y)
y_pred = lasso.predict(X)
residuals = y - y_pred

print("Coefficients:", lasso.coef_)
plt.bar(X.columns, lasso.coef_)
plt.title("Lasso Coefficients")
plt.xticks(rotation=90)
plt.show()
plt.scatter(y_pred, residuals)
plt.hlines(0, min(y_pred), max(y_pred), colors='red')
plt.xlabel("Predicted Quality")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()

```

61. Lasso Regression on HealthData.csv (all features vs. health_score)

python

```

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso
import numpy as np
from sklearn.metrics import mean_squared_error

df = pd.read_csv('HealthData.csv')
X = df.drop('health_score', axis=1)
y = df['health_score']

lasso = Lasso(alpha=0.1)
lasso.fit(X, y)

```

```
y_pred = lasso.predict(X)
residuals = y - y_pred

print("Coefficients:", lasso.coef_)
plt.bar(X.columns, lasso.coef_)
plt.title("Lasso Coefficients")
plt.xticks(rotation=45)
plt.show()
plt.scatter(y_pred, residuals)
plt.hlines(0, min(y_pred), max(y_pred), colors='red')
plt.xlabel("Predicted Health Score")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

62. PCA on 100 samples with 3 features (random data)

python

```
import numpy as np
from sklearn.decomposition import PCA

X = np.random.rand(100,3)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
print("Explained Variance Ratios:", pca.explained_variance_ratio_)
```

63. PCA on a chosen dataset with explained variance printed

python

```
import numpy as np
from sklearn.decomposition import PCA

X = np.random.rand(200,5)
```

```
pca = PCA(n_components=5)
pca.fit(X)
print("Explained Variance Ratios:", pca.explained_variance_ratio_)
```

64. Synthetic blobs with 3 features & 3 centers, PCA to 2D, side-by-side plots

python

```
from sklearn.datasets import make_blobs
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

X, y = make_blobs(n_samples=300, centers=3, n_features=3, random_state=42)
X_pca = PCA(n_components=2).fit_transform(X)

fig = plt.figure(figsize=(12,5))
ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(X[:,0], X[:,1], X[:,2], c=y, cmap='viridis')
ax1.set_title("Original 3D Data")
ax2 = fig.add_subplot(122)
plt.scatter(X_pca[:,0], X_pca[:,1], c=y, cmap='viridis')
plt.title("PCA Reduced 2D Data")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

65. PCA on Iris (reduce to 2D)

python

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

```
iris = load_iris()
X, y = iris.data, iris.target
X_pca = PCA(n_components=2).fit_transform(X)
plt.scatter(X_pca[:,0], X_pca[:,1], c=y, cmap='Set1')
plt.title("PCA on Iris")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.show()
```

66. LDA on Wine dataset (reduce to 2D)

python

```
from sklearn.datasets import load_wine
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.pyplot as plt

wine = load_wine()
X, y = wine.data, wine.target
X_lda = LDA(n_components=2).fit_transform(X, y)
plt.scatter(X_lda[:,0], X_lda[:,1], c=y, cmap='Set1')
plt.title("LDA on Wine")
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.show()
```

67. LDA on Breast Cancer dataset (reduce to 1D)

python

```
from sklearn.datasets import load_breast_cancer
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.pyplot as plt
```

```
data = load_breast_cancer()
X, y = data.data, data.target
X_lda = LDA(n_components=1).fit_transform(X, y)
plt.scatter(X_lda, [0]*len(X_lda), c=y, cmap='coolwarm', marker='o')
plt.xlabel("LD1")
plt.title("LDA on Breast Cancer (1D)")
plt.show()
```

68. LDA on Digits dataset (reduce to 2D)

python

```
from sklearn.datasets import load_digits
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.pyplot as plt

digits = load_digits()
X, y = digits.data, digits.target
X_lda = LDA(n_components=2).fit_transform(X, y)
plt.scatter(X_lda[:,0], X_lda[:,1], c=y, cmap='Spectral')
plt.title("LDA on Digits")
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.show()
```

69. LDA on Iris (reduce to 2D, plot)

python

```
from sklearn.datasets import load_iris
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
import matplotlib.pyplot as plt

iris = load_iris()
X, y = iris.data, iris.target
```

```
X_lda = LDA(n_components=2).fit_transform(X, y)
plt.scatter(X_lda[:,0], X_lda[:,1], c=y, cmap='Set1')
plt.title("LDA on Iris")
plt.xlabel("LD1")
plt.ylabel("LD2")
plt.show()
```

70. SVC to classify Iris species using sepal/petal dimensions

python

```
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
svc = SVC(kernel='rbf', probability=True, random_state=42)
svc.fit(X_train, y_train)
print("Test Accuracy:", svc.score(X_test, y_test))

# For visualization, use first two features (sepal length and width)
plt.scatter(X_test[:,0], X_test[:,1], c=svc.predict(X_test), cmap='viridis')
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.title("SVC Classification on Iris")
plt.show()
```

Each snippet above represents one “precise and simple” solution. You can use these as starting points and adjust them further as needed. Happy coding!