```python
#1: What is K-Nearest Neighbors (KNN) and how does it work in both classification and regression problems? 📌 K-Nearest Neighbor

# Import libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris, load_diabetes
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.metrics import accuracy_score, mean_squared_error

# ----------------------------
# 🔹 KNN for Classification
# ----------------------------
print("=== KNN for Classification (Iris Dataset) ===")

# Load dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features (important for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train KNN Classifier
knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train)

# Predictions
y_pred_clf = knn_clf.predict(X_test)

# Accuracy
print("Classification Accuracy:", accuracy_score(y_test, y_pred_clf))

# ----------------------------
# 🔹 KNN for Regression
# ----------------------------
print("\n=== KNN for Regression (Diabetes Dataset) ===")

# Load dataset
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train KNN Regressor
knn_reg = KNeighborsRegressor(n_neighbors=5)
knn_reg.fit(X_train, y_train)

# Predictions
y_pred_reg = knn_reg.predict(X_test)

# Mean Squared Error
print("Regression Mean Squared Error:", mean_squared_error(y_test, y_pred_reg))

# ----------------------------
# ✅ Explanation:
# - Classification → Majority voting among nearest neighbors
# - Regression → Average value among nearest neighbors
# - Scaling features is required since KNN relies on distances
# ----------------------------
```

```
=== KNN for Classification (Iris Dataset) ===
Classification Accuracy: 1.0

=== KNN for Regression (Diabetes Dataset) ===
Regression Mean Squared Error: 3047.449887640449
```

2: What is the Curse of Dimensionality and how does it affect KNN performance?

Ans- 📌 Curse of Dimensionality & Its Effect on KNN 🔹 What is the Curse of Dimensionality?

The curse of dimensionality refers to problems that arise when the number of features (dimensions) in the dataset becomes very large.

In high-dimensional space:

- Data becomes sparse → Points are far apart.
- Distance measures lose meaning → The difference between the nearest and farthest neighbor becomes very small.
- More data needed → The volume of space increases exponentially, so we need exponentially more data to maintain density.

🔹 How it Affects KNN Performance?

Since KNN relies on distance (Euclidean/Manhattan, etc.), the curse of dimensionality creates problems:

- Distances become less meaningful → All points seem equally far, so KNN struggles to identify true neighbors.
- Overfitting risk → With many irrelevant features, noise dominates the distance calculation.
- High computation cost → Distance calculation in high dimensions is very expensive.

🔹 Ways to Reduce the Effect:

- Feature Selection → Keep only relevant features.
- Dimensionality Reduction → Use PCA, t-SNE, or autoencoders.
- Scaling/Normalization → Helps but does not fully solve the issue.

3: What is Principal Component Analysis (PCA)? How is it different from feature selection?

Ans:- 📌 Principal Component Analysis (PCA) vs Feature Selection

🔹 What is PCA?

These components are linear combinations of the original features.

They are chosen to capture maximum variance in the data.

The first component captures the most variance, the second captures the next, and so on.

Aspect	PCA (Feature Extraction)	Feature Selection
Definition	Creates new features (principal components) as linear combinations of original features.	Selects a subset of the most important original features.
Approach	Transforms the feature space.	Keeps original features, removes irrelevant ones.
Interpretability	Components are harder to interpret (mix of many features).	Selected features remain interpretable.
Goal	Reduce dimensionality while retaining variance.	Reduce dimensionality by keeping only important features.
Example	Convert 100 correlated features into 10 components.	Pick top 10 features out of 100 based on importance.

4: What are eigenvalues and eigenvectors in PCA, and why are they important?

Ans:-📌 Eigenvalues & Eigenvectors in PCA

🔹 What are Eigenvalues & Eigenvectors?

- Eigenvectors: Directions along which data varies the most (principal components).
- Eigenvalues: The amount of variance captured along each eigenvector (importance/weight of each component).

👉 In PCA:

- Eigenvectors = new feature axes (principal components).
- Eigenvalues = how much variance (information) each component explains.

🔹 Why are They Important in PCA?

1.Eigenvectors determine the orientation of new axes (principal components).

2.Eigenvalues tell us how much information (variance) is retained by each component.

3.We use the largest eigenvalues → corresponding eigenvectors form the reduced feature space.

4.Helps in deciding how many components to keep (e.g., keep components that explain 95% variance).

5: How do KNN and PCA complement each other when applied in a single pipeline?

Ans- 📌 How KNN and PCA Complement Each Other

🔹 KNN Recap

- KNN is a distance-based algorithm.
- Performance depends heavily on feature space and distances.
- Struggles in high-dimensional data (curse of dimensionality).

🔹 PCA Recap

- PCA reduces dimensionality by creating new features (principal components).
- Removes noise and correlations between features.
- Retains maximum variance in fewer dimensions.

🔹 How They Work Together

- PCA before KNN → PCA reduces dimensionality, keeping only the most informative components.
- This makes distances more meaningful for KNN (less noise, less redundancy).
- PCA also reduces computation cost → KNN is faster with fewer features.
- PCA helps avoid overfitting in KNN by removing irrelevant/weak features.

```
#6: Train a KNN Classifier on the Wine dataset with and without features caling. Compare model accuracy in both cases.
import numpy as np
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load dataset
wine = load_wine()
X, y = wine.data, wine.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# -------------------------
# 1️⃣ KNN without Scaling
# -------------------------
knn_no_scaling = KNeighborsClassifier(n_neighbors=5)
knn_no_scaling.fit(X_train, y_train)
y_pred_no_scaling = knn_no_scaling.predict(X_test)
acc_no_scaling = accuracy_score(y_test, y_pred_no_scaling)

# -------------------------
# 2️⃣ KNN with Scaling
# -------------------------
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn_scaling = KNeighborsClassifier(n_neighbors=5)
knn_scaling.fit(X_train_scaled, y_train)
y_pred_scaling = knn_scaling.predict(X_test_scaled)
acc_scaling = accuracy_score(y_test, y_pred_scaling)

# -------------------------
# Results
# -------------------------
print("Accuracy without Scaling:", acc_no_scaling)
print("Accuracy with Scaling   :", acc_scaling)
```

```
Accuracy without Scaling: 0.7222222222222222
Accuracy with Scaling   : 0.9444444444444444
```

```
#7: Train a PCA model on the Wine dataset and print the explained variance ratio of each principal component.
import numpy as np
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load dataset
```

```
wine = load_wine()
X, y = wine.data, wine.target

# Standardize features (important for PCA)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA (keep all components)
pca = PCA()
pca.fit(X_scaled)

# Explained variance ratio
explained_variance = pca.explained_variance_ratio_

print("Explained Variance Ratio of each Principal Component:")
for i, var in enumerate(explained_variance):
    print(f"PC{i+1}: {var:.4f}")

print("\nTotal Variance Explained:", explained_variance.sum())
```

```
Explained Variance Ratio of each Principal Component:
PC1: 0.3620
PC2: 0.1921
PC3: 0.1112
PC4: 0.0707
PC5: 0.0656
PC6: 0.0494
PC7: 0.0424
PC8: 0.0268
PC9: 0.0222
PC10: 0.0193
PC11: 0.0174
PC12: 0.0130
PC13: 0.0080

Total Variance Explained: 1.0
```

```
#8: Train a KNN Classifier on the PCA-transformed dataset (retain top 2 components). Compare the accuracy with the original data
import numpy as np
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load dataset
wine = load_wine()
X, y = wine.data, wine.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# --------------------------
# 1️⃣ KNN on Original Dataset
# --------------------------
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn_original = KNeighborsClassifier(n_neighbors=5)
knn_original.fit(X_train_scaled, y_train)
y_pred_original = knn_original.predict(X_test_scaled)
acc_original = accuracy_score(y_test, y_pred_original)

# --------------------------
# 2️⃣ KNN on PCA-Transformed Dataset (Top 2 Components)
# --------------------------
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

knn_pca = KNeighborsClassifier(n_neighbors=5)
knn_pca.fit(X_train_pca, y_train)
y_pred_pca = knn_pca.predict(X_test_pca)
acc_pca = accuracy_score(y_test, y_pred_pca)
```

```
# -------------------------
# Results
# -------------------------
print("Accuracy on Original Dataset:", acc_original)
print("Accuracy on PCA (2 Components):", acc_pca)
```

```
Accuracy on Original Dataset: 0.9444444444444444
Accuracy on PCA (2 Components): 1.0
```

```
#9: Train a KNN Classifier with different distance metrics (euclidean, manhattan) on the scaled Wine dataset and compare the res
import numpy as np
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Load dataset
wine = load_wine()
X, y = wine.data, wine.target

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features (important for KNN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# -------------------------
# 1️⃣ KNN with Euclidean Distance (default)
# -------------------------
knn_euclidean = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn_euclidean.fit(X_train_scaled, y_train)
y_pred_euclidean = knn_euclidean.predict(X_test_scaled)
acc_euclidean = accuracy_score(y_test, y_pred_euclidean)

# -------------------------
# 2️⃣ KNN with Manhattan Distance
# -------------------------
knn_manhattan = KNeighborsClassifier(n_neighbors=5, metric='manhattan')
knn_manhattan.fit(X_train_scaled, y_train)
y_pred_manhattan = knn_manhattan.predict(X_test_scaled)
acc_manhattan = accuracy_score(y_test, y_pred_manhattan)

# -------------------------
# Results
# -------------------------
print("Accuracy with Euclidean Distance:", acc_euclidean)
print("Accuracy with Manhattan Distance:", acc_manhattan)
```

```
Accuracy with Euclidean Distance: 0.9444444444444444
Accuracy with Manhattan Distance: 0.9444444444444444
```

10: You are working with a high-dimensional gene expression dataset to classify patients with different types of cancer. Due to the large number of features and a small number of samples, traditional models overfit. Explain how you would: ● Use PCA to reduce dimensionality ● Decide how many components to keep ● Use KNN for classification post-dimensionality reduction ● Evaluate the model ● Justify this pipeline to your stakeholders as a robust solution for real-world biomedical data

Ans- 📌 Cancer Classification with PCA + KNN

🔹 Problem

- High-dimensional gene expression dataset (many features, few samples).
- Risk: Overfitting because traditional ML models cannot generalize well in such cases.

🔹 Step-by-Step Solution

1.Use PCA to Reduce Dimensionality

- Gene expression data may have thousands of features but only hundreds of patients.
- Apply PCA after scaling → compress features into a few principal components that capture most variance.

2.Decide How Many Components to Keep

- Look at explained variance ratio.
- Keep enough PCs to explain ~90–95% of variance (balance between information retention & noise reduction).
- Use a scree plot (elbow method) for visualization.

3.Use KNN for Classification Post-Dimensionality Reduction

- After PCA transformation, apply KNN classifier.
- KNN works better in reduced, denoised feature space since distances become more meaningful.

4.Evaluate the Model

- Use train-test split or cross-validation.
- Metrics: Accuracy, Precision, Recall, F1-score (important for medical diagnosis).
- Possibly use ROC-AUC for binary/multiclass evaluation.

5.Justify Pipeline to Stakeholders

- PCA → Handles high-dimensional data, reduces noise, avoids overfitting.
- KNN → Simple, interpretable, works well with reduced features.
- Evaluation → Cross-validation ensures robustness, metrics show reliability.
- This pipeline balances accuracy + interpretability, making it a strong choice for real-world biomedical datasets.

```
#10 ans-
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Simulate a high-dimensional gene expression dataset
X, y = make_classification(n_samples=200, n_features=1000, n_informative=50,
                           n_classes=3, random_state=42)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 1: Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 2: Apply PCA (keep 95% variance)
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

print("Number of components retained:", pca.n_components_)
print("Explained variance ratio sum:", np.sum(pca.explained_variance_ratio_))

# Step 3: Train KNN Classifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_pca, y_train)
y_pred = knn.predict(X_test_pca)

# Step 4: Evaluate
print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Number of components retained: 125
Explained variance ratio sum: 0.9525080991404814

Accuracy: 0.3333333333333333

Classification Report:
               precision    recall  f1-score   support

           0       0.29      0.42      0.34        19
           1       0.57      0.33      0.42        24
           2       0.22      0.24      0.23        17
```

```
    accuracy                           0.33        60
   macro avg       0.36      0.33      0.33        60
weighted avg       0.38      0.33      0.34        60
```

```
Start coding or generate with AI.
```