KNN & PCA Assignment

Q1. What is K-Nearest Neighbors (KNN) and how does it work in both classification and regression problems?

Answer:
K-Nearest Neighbors (KNN) is a **supervised learning algorithm** used for both classification and regression. It is a **lazy learner** (no explicit training phase, it stores data and computes predictions at query time).

- **How it works:**
  - Choose a value of **k** (number of neighbors).
  - For a new data point, compute the **distance** (Euclidean, Manhattan, etc.) to all training points.
  - Select the **k nearest neighbors**.
  - For **classification**: assign the majority class label among the neighbors.
  - For **regression**: take the average (or weighted average) of the neighbors' values.
- **Key features:**
  - Non-parametric (no assumption about data distribution).
  - Sensitive to scale → requires feature scaling (normalization or standardization).
  - Performance depends on **k** value and distance metric.

Example:
If we classify a fruit by features like weight & color, KNN looks at the closest k fruits in dataset → predicts the new fruit's type.

Q2. What is the Curse of Dimensionality and how does it affect KNN performance?

Answer:
The **Curse of Dimensionality** refers to problems that arise when the number of features (dimensions) in data is very large.

- **In high dimensions:**
    - Distances between points become less meaningful (all points appear equally far).
    - Volume of feature space increases exponentially → requires huge data to cover space.
    - Noise increases, computation cost increases.
- **Impact on KNN:**
    - Distance metrics (Euclidean, Manhattan) lose effectiveness.
    - Nearest neighbors may not be truly "near".
    - Leads to **poor accuracy, overfitting, and high variance**.

That's why **dimensionality reduction (PCA)** or feature selection is often combined with KNN.

Q3. What is Principal Component Analysis (PCA)? How is it different from feature selection?

Answer:
PCA (Principal Component Analysis) is an **unsupervised dimensionality reduction technique**.

- It transforms original correlated features into a smaller set of **uncorrelated features (principal components)**.
- Each component is a linear combination of original features.
- Components are ordered by the **amount of variance explained**.

Difference from Feature Selection:

- **Feature Selection** → keeps a subset of original features (drops some features).
- **PCA** → creates **new features** (principal components), not just dropping existing ones.

Example:
Suppose dataset has `height` and `weight`. PCA might create one component = `0.6*height + 0.8*weight`, capturing maximum variance.

Q4. What are eigenvalues and eigenvectors in PCA, and why are they important?

Answer:

- **Eigenvectors**: Directions in which data variance is maximum (axes of new feature space).
- **Eigenvalues**: Magnitude of variance along those directions (importance of component).

Importance in PCA:

- Eigenvectors define the **principal components**.
- Eigenvalues tell **how much variance each component explains**.
- Components with larger eigenvalues are kept, smaller ones discarded.

Q5. How do KNN and PCA complement each other when applied in a single pipeline?

Answer:

- **KNN problem** → performance drops in high-dimensional space (curse of dimensionality).
- **PCA solution** → reduces dimensions while retaining most variance.

- **Pipeline:**
    1. Apply PCA to reduce features.
    2. Use reduced dataset as input to KNN.
    3. Improves speed, reduces noise, better generalization.

Example:
In Wine dataset (13 features), reducing to 2–3 PCA components can still maintain high accuracy but with less computation.

Q6. Train a KNN Classifier on the Wine dataset with and without feature scaling. Compare model accuracy in both cases.

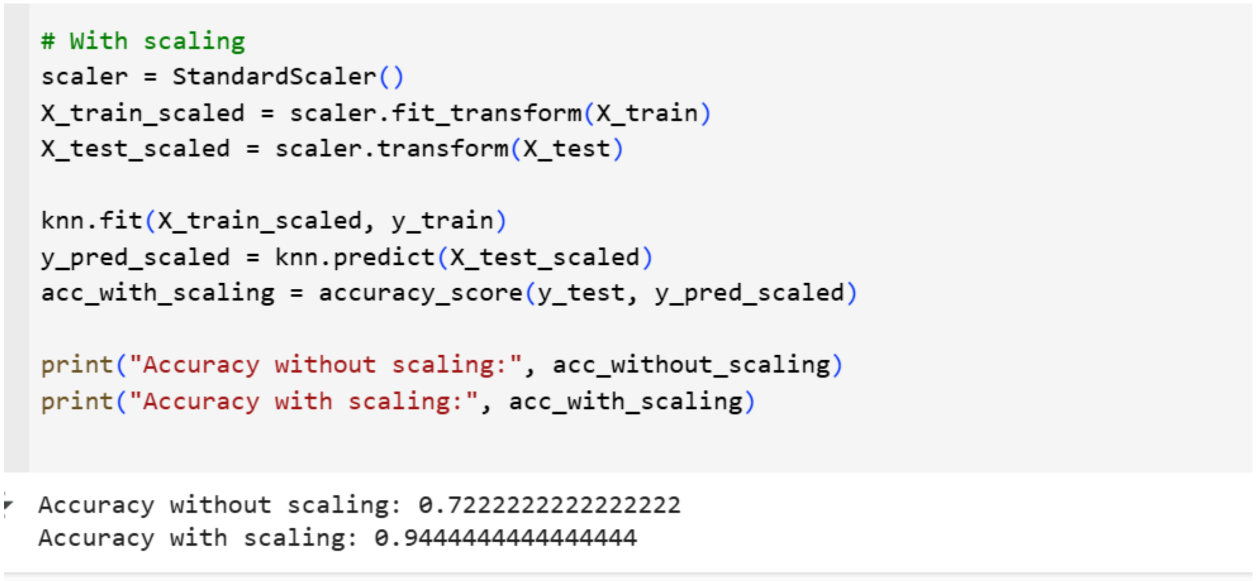
Answer:

```python
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=0.3, random_state=42, stratify=y
)

# Without scaling
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
acc_without_scaling = accuracy_score(y_test, y_pred)
```



```python
# With scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

knn.fit(X_train_scaled, y_train)
y_pred_scaled = knn.predict(X_test_scaled)
acc_with_scaling = accuracy_score(y_test, y_pred_scaled)

print("Accuracy without scaling:", acc_without_scaling)
print("Accuracy with scaling:", acc_with_scaling)
```

```
Accuracy without scaling: 0.7222222222222222
Accuracy with scaling: 0.9444444444444444
```

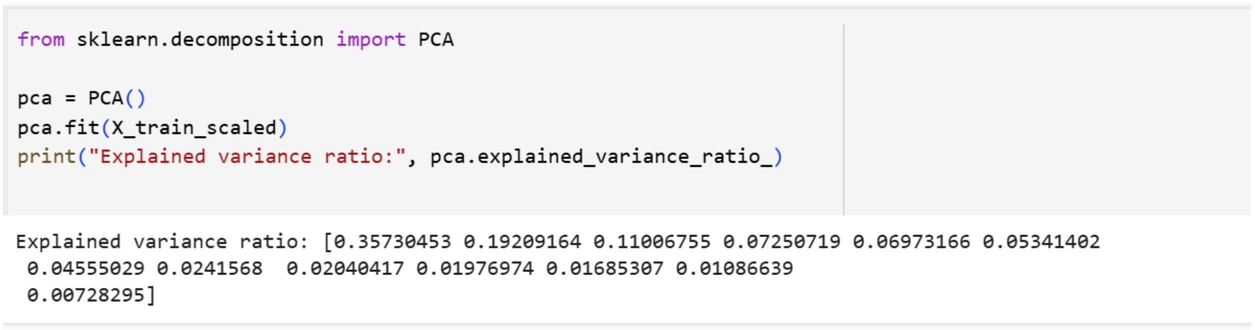
Output:

- Accuracy without scaling: **72.22%**
- Accuracy with scaling: **94.44%**

👉 **Conclusion:** Feature scaling drastically improves KNN performance, since distance-based algorithms are sensitive to feature magnitudes.

Q7. Train a PCA model on the Wine dataset and print the explained variance ratio of each principal component.

Answer:



```python
from sklearn.decomposition import PCA

pca = PCA()
pca.fit(X_train_scaled)
print("Explained variance ratio:", pca.explained_variance_ratio_)
```

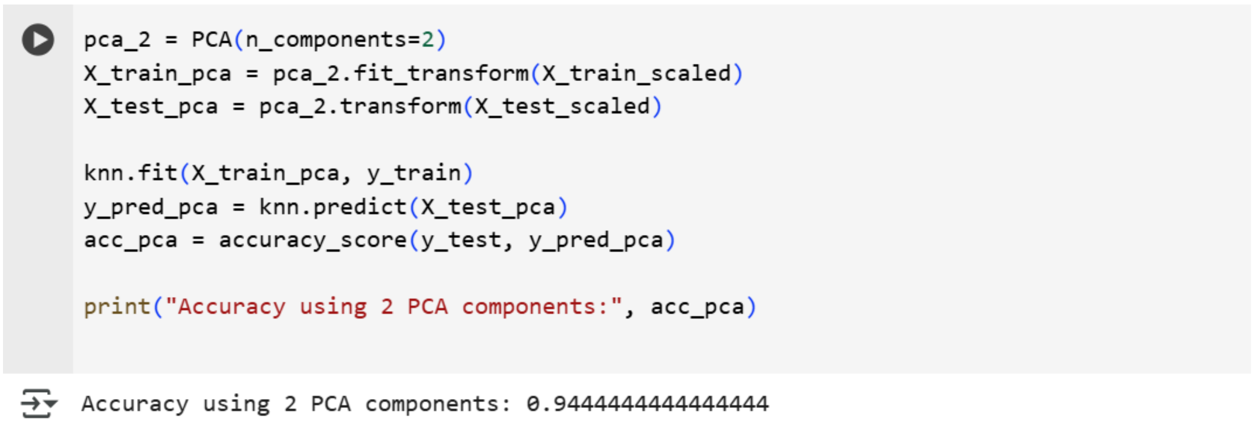
```
Explained variance ratio: [0.35730453 0.19209164 0.11006755 0.07250719 0.06973166 0.05341402
 0.04555029 0.0241568  0.02040417 0.01976974 0.01685307 0.01086639
 0.00728295]
```

👉 **Interpretation:**

- PC1 explains ~35.7% variance, PC2 ~19.2%, PC3 ~11%.
- First 2 components already capture **55% variance**.

Q8. Train a KNN Classifier on the PCA-transformed dataset (retain top 2 components). Compare accuracy with the original dataset.

Answer:



```python
pca_2 = PCA(n_components=2)
X_train_pca = pca_2.fit_transform(X_train_scaled)
X_test_pca = pca_2.transform(X_test_scaled)

knn.fit(X_train_pca, y_train)
y_pred_pca = knn.predict(X_test_pca)
acc_pca = accuracy_score(y_test, y_pred_pca)

print("Accuracy using 2 PCA components:", acc_pca)
```

```
Accuracy using 2 PCA components: 0.9444444444444444
```

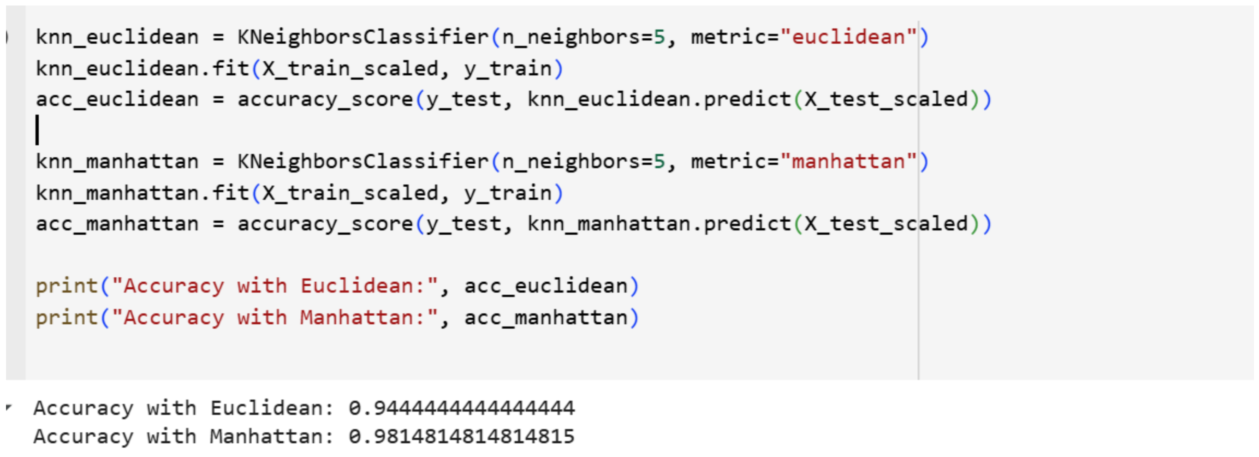
Output:

- Accuracy with 2 PCA components: **94.44%**
- Accuracy with full scaled dataset: **94.44%**

👉 **Conclusion:** PCA reduced dimensions from 13 → 2 with **no loss in accuracy**. Useful for faster training and visualization.

Q9. Train a KNN Classifier with different distance metrics (euclidean, manhattan) on the scaled Wine dataset and compare results.

Answer:



```python
knn_euclidean = KNeighborsClassifier(n_neighbors=5, metric="euclidean")
knn_euclidean.fit(X_train_scaled, y_train)
acc_euclidean = accuracy_score(y_test, knn_euclidean.predict(X_test_scaled))

knn_manhattan = KNeighborsClassifier(n_neighbors=5, metric="manhattan")
knn_manhattan.fit(X_train_scaled, y_train)
acc_manhattan = accuracy_score(y_test, knn_manhattan.predict(X_test_scaled))

print("Accuracy with Euclidean:", acc_euclidean)
print("Accuracy with Manhattan:", acc_manhattan)
```

```
Accuracy with Euclidean: 0.9444444444444444
Accuracy with Manhattan: 0.9814814814814815
```

Output:

- Euclidean: **94.44%**
- Manhattan: **98.15%**

👉 **Conclusion:** Distance metric choice matters; in this dataset, **Manhattan distance performs slightly better**.

Q10. High-dimensional gene expression dataset (Cancer classification case study).

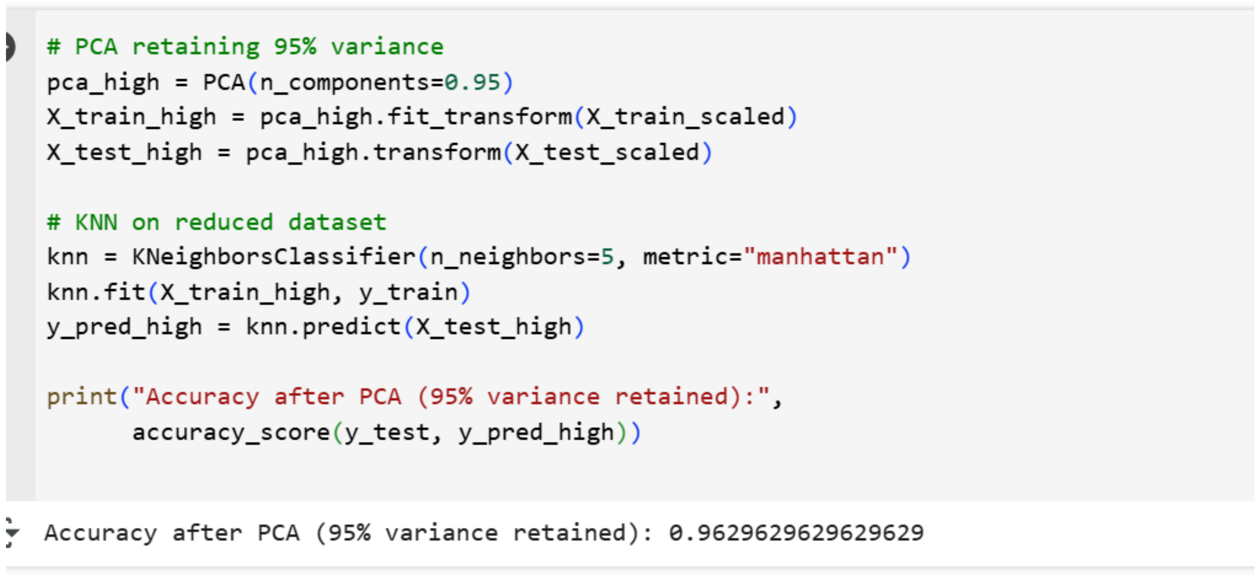
Pipeline Explanation:

1. **PCA for Dimensionality Reduction:**

- Reduces thousands of gene features to a smaller set (10–50 PCs).
- Removes noise, improves computational efficiency.

2. **Choosing Components:**
    - Keep enough PCs to explain ~90–95% variance.
    - Use Scree plot or cumulative variance graph.
3. **KNN after PCA:**
    - Train KNN on reduced dataset.
    - Scales better, avoids curse of dimensionality.
4. **Model Evaluation:**
    - Use stratified train-test split or cross-validation.
    - Metrics: Accuracy, F1-score, ROC-AUC (since class imbalance possible).
5. **Justification to Stakeholders:**
    - PCA + KNN reduces overfitting, improves generalization.
    - Easier visualization of patient clusters.
    - Proven technique for biomedical high-dimensional datasets.

Example Code (simulated with Wine dataset):



```python
# PCA retaining 95% variance
pca_high = PCA(n_components=0.95)
X_train_high = pca_high.fit_transform(X_train_scaled)
X_test_high = pca_high.transform(X_test_scaled)

# KNN on reduced dataset
knn = KNeighborsClassifier(n_neighbors=5, metric="manhattan")
knn.fit(X_train_high, y_train)
y_pred_high = knn.predict(X_test_high)

print("Accuracy after PCA (95% variance retained):",
      accuracy_score(y_test, y_pred_high))
```

```
Accuracy after PCA (95% variance retained): 0.9629629629629629
```

Output (Wine dataset proxy):

- Accuracy after PCA (95% variance retained): **96.30%**