Q1. KNN Classifier on load_iris dataset:

```
from sklearn.datasets import load_iris from sklearn.model_selection import
train_test_split from sklearn.neighbors import KNeighborsClassifier from
sklearn.metrics import accuracy_score # Load dataset iris = load_iris() X = iris.data y
= iris.target # Split dataset into train and test sets X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42) # Initialize KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=5) # Train the classifier
knn_classifier.fit(X_train, y_train) # Predict on the test set y_pred =
knn_classifier.predict(X_test) # Calculate accuracy accuracy = accuracy_score(y_test,
y_pred) print("Accuracy:", accuracy)
```

Q2. KNN Regressor on load_boston dataset:

```
from sklearn.datasets import load_boston from sklearn.model_selection import
train_test_split from sklearn.neighbors import KNeighborsRegressor from
sklearn.metrics import mean_squared_error # Load dataset boston = load_boston()
X = boston.data y = boston.target # Split dataset into train and test sets X_train,
X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42) #
Initialize KNN regressor knn_regressor = KNeighborsRegressor(n_neighbors=5) #
Train the regressor knn_regressor.fit(X_train, y_train) # Predict on the test set
y_pred = knn_regressor.predict(X_test) # Calculate mean squared error mse =
mean_squared_error(y_test, y_pred) print("Mean Squared Error:", mse)
```

Q3. Finding optimal K for KNN Classifier using cross-validation on load_iris dataset:

```
from sklearn.datasets import load_iris from sklearn.model_selection import
cross_val_score from sklearn.neighbors import KNeighborsClassifier # Load dataset
iris = load_iris() X = iris.data y = iris.target # Initialize KNN classifier knn_classifier =
KNeighborsClassifier() # Perform cross-validation to find optimal K cv_scores = [] for
k in range(1, 31): knn_classifier.n_neighbors = k scores =
cross_val_score(knn_classifier, X, y, cv=5, scoring='accuracy')
cv_scores.append(scores.mean()) # Find optimal K optimal_k =
cv_scores.index(max(cv_scores)) + 1 print("Optimal K:", optimal_k)
```

Q4. KNN Regressor with feature scaling on load_boston dataset:

```
from sklearn.datasets import load_boston from sklearn.model_selection import
train_test_split from sklearn.neighbors import KNeighborsRegressor from
sklearn.preprocessing import StandardScaler from sklearn.metrics import
mean_squared_error # Load dataset boston = load_boston() X = boston.data y =
boston.target # Feature scaling scaler = StandardScaler() X_scaled =
scaler.fit_transform(X) # Split dataset into train and test sets X_train, X_test, y_train,
y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42) # Initialize
KNN regressor knn_regressor = KNeighborsRegressor(n_neighbors=5) # Train the
regressor knn_regressor.fit(X_train, y_train) # Predict on the test set y_pred =
knn_regressor.predict(X_test) # Calculate mean squared error mse =
mean_squared_error(y_test, y_pred) print("Mean Squared Error:", mse)
```

Q5. KNN Classifier with weighted voting on load_iris dataset:

```
from sklearn.datasets import load_iris from sklearn.model_selection import
train_test_split from sklearn.neighbors import KNeighborsClassifier from
sklearn.metrics import accuracy_score # Load dataset iris = load_iris() X = iris.data y
= iris.target # Split dataset into train and test sets X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42) # Initialize KNN classifier with
weighted voting knn_classifier = KNeighborsClassifier(n_neighbors=5,
weights='distance') # Train the classifier knn_classifier.fit(X_train, y_train) # Predict
on the test set y_pred = knn_classifier.predict(X_test) # Calculate accuracy accuracy
= accuracy_score(y_test, y_pred) print("Accuracy:", accuracy)
```

Q6. Function to standardize features before applying KNN Classifier:

```
from sklearn.preprocessing import StandardScaler def standardize_features(X):
scaler = StandardScaler() X_scaled = scaler.fit_transform(X) return X_scaled
```

Q7. Function to calculate Euclidean distance between two points:

pythonCopy code

```
import numpy as np
 def euclidean_distance(point1, point2):
return np.sqrt(np.sum((point1 - point2) ** 2))
```

Q8. Function to calculate Manhattan distance between two points:

```
import numpy as np

def manhattan_distance(point1, point2):
   return np.sum(np.abs(point1 - point2))
```