The Iris Dataset in Python - Implementing KNN with Different Distance Measures

1. Importing necessary libraries

First, we need to import the libraries required for the KNN implementation and distance calculations.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from scipy.spatial import distance
```

2. Loading the dataset and preparing the data

Here, we are using the Iris dataset available under the `sklearn` module. We load it and split it into training and testing datasets.

```
# Loading the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Splitting the dataset into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. KNN implementation with Euclidean distance

We define a KNN class that uses the Euclidean distance, and then test the implementation using the testing dataset:

```
class KNNEuclidean:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        # Computing distances using Euclidean distance
        distances = distance.cdist([x], self.X_train, metric='euclidean')[0]

        # Getting the indices of the k nearest neighbors
        k_indices = np.argsort(distances)[:self.k]

        # Extracting the labels of the k nearest neighbors
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # Returning the most common class label
        most_common = np.bincount(k_nearest_labels).argmax()
        return most_common


# Training and testing KNN with Euclidean distance
knn_euclidean = KNNEuclidean(k=3)
knn_euclidean.fit(X_train, y_train)
y_pred_euclidean = knn_euclidean.predict(X_test)
print("Euclidean Distance Accuracy:", accuracy_score(y_test, y_pred_euclidean))
```

```
Euclidean Distance Accuracy: 1.0
```

4. KNN implementation with Manhattan distance

Next, we define a KNN class that uses the Manhattan distance, and then test the implementation using the testing dataset:

```
class KNNManhattan:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        # Computing distances using Manhattan distance
        distances = distance.cdist([x], self.X_train, metric='cityblock')[0]

        # Getting the indices of the k nearest neighbors
        k_indices = np.argsort(distances)[:self.k]

        # Extracting the labels of the k nearest neighbors
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # Returning the most common class label
        most_common = np.bincount(k_nearest_labels).argmax()
        return most_common


# Training and testing KNN with Manhattan distance
knn_manhattan = KNNManhattan(k=3)
knn_manhattan.fit(X_train, y_train)
y_pred_manhattan = knn_manhattan.predict(X_test)
print("Manhattan Distance Accuracy:", accuracy_score(y_test, y_pred_manhattan))
```

```
Manhattan Distance Accuracy: 1.0
```

5. KNN implementation with Mahalanobis distance

Finally, we define a KNN class that uses the Mahalanobis (or statistical) distance, and then test the implementation using the testing dataset:

```
class KNNMahalanobis:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
        # Computing the covariance matrix and its inverse
        self.cov_matrix = np.cov(X.T)
        self.inv_cov_matrix = np.linalg.inv(self.cov_matrix)

    def predict(self, X):
        predictions = [self._predict(x) for x in X]
        return np.array(predictions)

    def _predict(self, x):
        # Computing distances using Mahalanobis distance
        distances = [distance.mahalanobis(x, x_train, self.inv_cov_matrix) for x_train in self.X_train]

        # Getting the indices of the k nearest neighbors
        k_indices = np.argsort(distances)[:self.k]

        # Extracting the labels of the k nearest neighbors
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # Returning the most common class label
        most_common = np.bincount(k_nearest_labels).argmax()
        return most_common


# Training and testing KNN with Mahalanobis distance
knn_mahalanobis = KNNMahalanobis(k=3)
knn_mahalanobis.fit(X_train, y_train)
y_pred_mahalanobis = knn_mahalanobis.predict(X_test)
print("Mahalanobis Distance Accuracy:", accuracy_score(y_test, y_pred_mahalanobis))
```

```
Mahalanobis Distance Accuracy: 0.9333333333333333
```