A case study using iris dataset for KNN algorithm

In [1]:
```
# import modules for this project
from sklearn import datasets
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# load iris dataset
iris = datasets.load_iris()
data, labels = iris.data, iris.target

# training testing split
res = train_test_split(data, labels,
                       train_size=0.8,
                       test_size=0.2,
                       random_state=12)
train_data, test_data, train_labels, test_labels = res

# Create and fit a nearest-neighbor classifier
from sklearn.neighbors import KNeighborsClassifier
# classifier "out of the box", no parameters
knn = KNeighborsClassifier()
knn.fit(train_data, train_labels)

# print some interested metrics
print("Predictions from the classifier:")
learn_data_predicted = knn.predict(train_data)
print(learn_data_predicted)
print("Target values:")
print(train_labels)
print(accuracy_score(learn_data_predicted, train_labels))

# re-do KNN using some specific parameters.
knn2 = KNeighborsClassifier(algorithm='auto',
                            leaf_size=30,
                            metric='minkowski',
                            p=2,         # p=2 is equivalent to euclidian distance
                            metric_params=None,
                            n_jobs=1,
                            n_neighbors=5,
                            weights='uniform')

knn.fit(train_data, train_labels)
test_data_predicted = knn.predict(test_data)
accuracy_score(test_data_predicted, test_labels)
```

```
Predictions from the classifier:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 1 2 2 1
 1 1 2 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 2 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
Target values:
[0 1 2 0 2 0 1 1 0 1 1 0 0 0 0 0 0 0 2 0 2 1 1 1 0 2 1 1 2 0 2 0 2 2 2 2 1
 1 1 1 2 0 2 2 0 1 0 2 2 0 1 1 0 0 1 1 1 1 2 1 2 0 0 1 1 1 0 2 1 0 2 2 1 2
 2 0 0 2 1 1 2 0 1 1 0 1 1 2 2 1 0 2 0 2 0 0 1 2 2 1 2 2 0 1 1 0 2 2 2 1 2
 2 2 0 0 1 0 2 2 1]
0.975
```

Out[1]:
```
0.9666666666666667
```

Use this command to help with choice of paramters in the `KNeighborsClassifier` function.

In [24]:
```
help(KNeighborsClassifier)
```

```
Help on class KNeighborsClassifier in module sklearn.neighbors._classification:

class KNeighborsClassifier(sklearn.neighbors._base.KNeighborsMixin, sklearn.base.ClassifierMixin, sklearn.neighbors._base.NeighborsBase)
 |  KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None)
 |
 |  Classifier implementing the k-nearest neighbors vote.
 |
 |  Read more in the :ref:`User Guide <classification>`.
 |
 |  Parameters
 |  ----------
 |  n_neighbors : int, default=5
 |      Number of neighbors to use by default for :meth:`kneighbors` queries.
 |
 |  weights : {'uniform', 'distance'} or callable, default='uniform'
 |      Weight function used in prediction.  Possible values:
 |
 |      - 'uniform' : uniform weights.  All points in each neighborhood
 |        are weighted equally.
 |      - 'distance' : weight points by the inverse of their distance.
 |        in this case, closer neighbors of a query point will have a
 |        greater influence than neighbors which are further away.
 |      - [callable] : a user-defined function which accepts an
 |        array of distances, and returns an array of the same shape
 |        containing the weights.
 |
 |  algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'
 |      Algorithm used to compute the nearest neighbors:
 |
 |      - 'ball_tree' will use :class:`BallTree`
 |      - 'kd_tree' will use :class:`KDTree`
 |      - 'brute' will use a brute-force search.
 |      - 'auto' will attempt to decide the most appropriate algorithm
 |        based on the values passed to :meth:`fit` method.
 |
 |      Note: fitting on sparse input will override the setting of
 |      this parameter, using brute force.
 |
 |  leaf_size : int, default=30
 |      Leaf size passed to BallTree or KDTree.  This can affect the
 |      speed of the construction and query, as well as the memory
 |      required to store the tree.  The optimal value depends on the
 |      nature of the problem.
 |
 |  p : int, default=2
 |      Power parameter for the Minkowski metric. When p = 1, this is
 |      equivalent to using manhattan_distance (l1), and euclidean_distance
 |      (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.
 |
 |  metric : str or callable, default='minkowski'
 |      The distance metric to use for the tree.  The default metric is
 |      minkowski, and with p=2 is equivalent to the standard Euclidean
 |      metric. For a list of available metrics, see the documentation of
 |      :class:`~sklearn.metrics.DistanceMetric`.
 |      If metric is "precomputed", X is assumed to be a distance matrix and
 |      must be square during fit. X may be a :term:`sparse graph`,
 |      in which case only "nonzero" elements may be considered neighbors.
 |
 |  metric_params : dict, default=None
 |      Additional keyword arguments for the metric function.
 |
 |  n_jobs : int, default=None
 |      The number of parallel jobs to run for neighbors search.
 |      ``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
 |      ``-1`` means using all processors. See :term:`Glossary <n_jobs>`
 |      for more details.
 |      Doesn't affect :meth:`fit` method.
 |
 |  Attributes
 |  ----------
 |  classes_ : array of shape (n_classes,)
 |      Class labels known to the classifier
 |
 |  effective_metric_ : str or callble
 |      The distance metric used. It will be same as the `metric` parameter
 |      or a synonym of it, e.g. 'euclidean' if the `metric` parameter set to
 |      'minkowski' and `p` parameter set to 2.
 |
 |  effective_metric_params_ : dict
 |      Additional keyword arguments for the metric function. For most metrics
 |      will be same with `metric_params` parameter, but may also contain the
 |      `p` parameter value if the `effective_metric_` attribute is set to
 |      'minkowski'.
 |
```

```
 |  n_features_in_ : int
 |      Number of features seen during :term:`fit`.
 |
 |      .. versionadded:: 0.24
 |
 |  feature_names_in_ : ndarray of shape (`n_features_in_`,)
 |      Names of features seen during :term:`fit`. Defined only when `X`
 |      has feature names that are all strings.
 |
 |      .. versionadded:: 1.0
 |
 |  n_samples_fit_ : int
 |      Number of samples in the fitted data.
 |
 |  outputs_2d_ : bool
 |      False when `y`'s shape is (n_samples, ) or (n_samples, 1) during fit
 |      otherwise True.
 |
 |  See Also
 |  --------
 |  RadiusNeighborsClassifier: Classifier based on neighbors within a fixed radius.
 |  KNeighborsRegressor: Regression based on k-nearest neighbors.
 |  RadiusNeighborsRegressor: Regression based on neighbors within a fixed radius.
 |  NearestNeighbors: Unsupervised learner for implementing neighbor searches.
 |
 |  Notes
 |  -----
 |  See :ref:`Nearest Neighbors <neighbors>` in the online documentation
 |  for a discussion of the choice of ``algorithm`` and ``leaf_size``.
 |
 |  .. warning::
 |
 |     Regarding the Nearest Neighbors algorithms, if it is found that two
 |     neighbors, neighbor `k+1` and `k`, have identical distances
 |     but different labels, the results will depend on the ordering of the
 |     training data.
 |
 |  https://en.wikipedia.org/wiki/K-nearest_neighbor_algorithm
 |
 |  Examples
 |  --------
 |  >>> X = [[0], [1], [2], [3]]
 |  >>> y = [0, 0, 1, 1]
 |  >>> from sklearn.neighbors import KNeighborsClassifier
 |  >>> neigh = KNeighborsClassifier(n_neighbors=3)
 |  >>> neigh.fit(X, y)
 |  KNeighborsClassifier(...)
 |  >>> print(neigh.predict([[1.1]]))
 |  [0]
 |  >>> print(neigh.predict_proba([[0.9]]))
 |  [[0.666... 0.333...]]
 |
 |  Method resolution order:
 |      KNeighborsClassifier
 |      sklearn.neighbors._base.KNeighborsMixin
 |      sklearn.base.ClassifierMixin
 |      sklearn.neighbors._base.NeighborsBase
 |      sklearn.base.MultiOutputMixin
 |      sklearn.base.BaseEstimator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=Non
e, n_jobs=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  fit(self, X, y)
 |      Fit the k-nearest neighbors classifier from the training dataset.
 |
 |      Parameters
 |      ----------
 |      X : {array-like, sparse matrix} of shape (n_samples, n_features) or                 (n_samples, n_samples) if metric='prec
omputed'
 |          Training data.
 |
 |      y : {array-like, sparse matrix} of shape (n_samples,) or                 (n_samples, n_outputs)
 |          Target values.
 |
 |      Returns
 |      -------
 |      self : KNeighborsClassifier
 |          The fitted k-nearest neighbors classifier.
 |
 |  predict(self, X)
 |      Predict the class labels for the provided data.
```

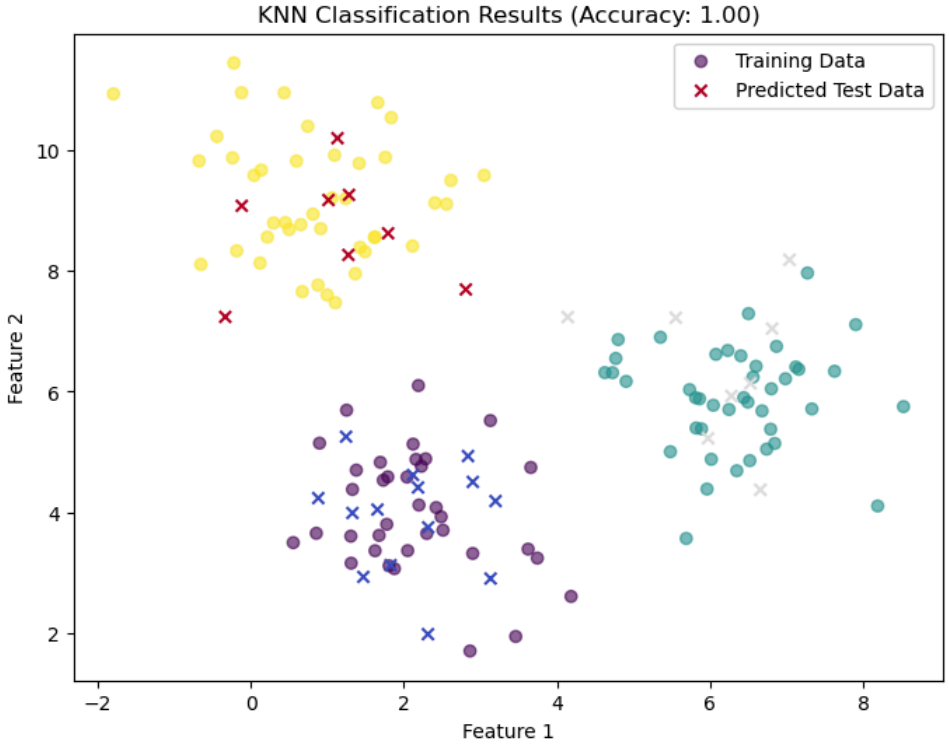
```
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_queries, n_features),                or (n_queries, n_indexed) if metric == 'precomputed'
 |          Test samples.
 |
 |      Returns
 |      -------
 |      y : ndarray of shape (n_queries,) or (n_queries, n_outputs)
 |          Class labels for each data sample.
 |
 |  predict_proba(self, X)
 |      Return probability estimates for the test data X.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_queries, n_features),                or (n_queries, n_indexed) if metric == 'precomputed'
 |          Test samples.
 |
 |      Returns
 |      -------
 |      p : ndarray of shape (n_queries, n_classes), or a list of n_outputs                of such arrays if n_outputs > 1.
 |          The class probabilities of the input samples. Classes are ordered
 |          by lexicographic order.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __abstractmethods__ = frozenset()
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.neighbors._base.KNeighborsMixin:
 |
 |  kneighbors(self, X=None, n_neighbors=None, return_distance=True)
 |      Find the K-neighbors of a point.
 |
 |      Returns indices of and distances to the neighbors of each point.
 |
 |      Parameters
 |      ----------
 |      X : array-like, shape (n_queries, n_features),             or (n_queries, n_indexed) if metric == 'precomputed',
default=None
 |          The query point or points.
 |          If not provided, neighbors of each indexed point are returned.
 |          In this case, the query point is not considered its own neighbor.
 |
 |      n_neighbors : int, default=None
 |          Number of neighbors required for each sample. The default is the
 |          value passed to the constructor.
 |
 |      return_distance : bool, default=True
 |          Whether or not to return the distances.
 |
 |      Returns
 |      -------
 |      neigh_dist : ndarray of shape (n_queries, n_neighbors)
 |          Array representing the lengths to points, only present if
 |          return_distance=True.
 |
 |      neigh_ind : ndarray of shape (n_queries, n_neighbors)
 |          Indices of the nearest points in the population matrix.
 |
 |      Examples
 |      --------
 |      In the following example, we construct a NearestNeighbors
 |      class from an array representing our data set and ask who's
 |      the closest point to [1,1,1]
 |
 |      >>> samples = [[0., 0., 0.], [0., .5, 0.], [1., 1., .5]]
 |      >>> from sklearn.neighbors import NearestNeighbors
 |      >>> neigh = NearestNeighbors(n_neighbors=1)
 |      >>> neigh.fit(samples)
 |      NearestNeighbors(n_neighbors=1)
 |      >>> print(neigh.kneighbors([[1., 1., 1.]]))
 |      (array([[0.5]]), array([[2]]))
 |
 |      As you can see, it returns [[0.5]], and [[2]], which means that the
 |      element is at distance 0.5 and is the third element of samples
 |      (indexes start at 0). You can also query for multiple points:
 |
 |      >>> X = [[0., 1., 0.], [1., 0., 1.]]
 |      >>> neigh.kneighbors(X, return_distance=False)
 |      array([[1],
 |             [2]]...)
 |
 |  kneighbors_graph(self, X=None, n_neighbors=None, mode='connectivity')
```

```
 |      Compute the (weighted) graph of k-Neighbors for points in X.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_queries, n_features),                or (n_queries, n_indexed) if metric == 'precomputed',
default=None
 |          The query point or points.
 |          If not provided, neighbors of each indexed point are returned.
 |          In this case, the query point is not considered its own neighbor.
 |          For ``metric='precomputed'`` the shape should be
 |          (n_queries, n_indexed). Otherwise the shape should be
 |          (n_queries, n_features).
 |
 |      n_neighbors : int, default=None
 |          Number of neighbors for each sample. The default is the value
 |          passed to the constructor.
 |
 |      mode : {'connectivity', 'distance'}, default='connectivity'
 |          Type of returned matrix: 'connectivity' will return the
 |          connectivity matrix with ones and zeros, in 'distance' the
 |          edges are distances between points, type of distance
 |          depends on the selected metric parameter in
 |          NearestNeighbors class.
 |
 |      Returns
 |      -------
 |      A : sparse-matrix of shape (n_queries, n_samples_fit)
 |          `n_samples_fit` is the number of samples in the fitted data.
 |          `A[i, j]` gives the weight of the edge connecting `i` to `j`.
 |          The matrix is of CSR format.
 |
 |      See Also
 |      --------
 |      NearestNeighbors.radius_neighbors_graph : Compute the (weighted) graph
 |          of Neighbors for points in X.
 |
 |      Examples
 |      --------
 |      >>> X = [[0], [3], [1]]
 |      >>> from sklearn.neighbors import NearestNeighbors
 |      >>> neigh = NearestNeighbors(n_neighbors=2)
 |      >>> neigh.fit(X)
 |      NearestNeighbors(n_neighbors=2)
 |      >>> A = neigh.kneighbors_graph(X)
 |      >>> A.toarray()
 |      array([[1., 0., 1.],
 |             [0., 1., 1.],
 |             [1., 0., 1.]])
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from sklearn.neighbors._base.KNeighborsMixin:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.ClassifierMixin:
 |
 |  score(self, X, y, sample_weight=None)
 |      Return the mean accuracy on the given test data and labels.
 |
 |      In multi-label classification, this is the subset accuracy
 |      which is a harsh metric since you require for each sample that
 |      each label set be correctly predicted.
 |
 |      Parameters
 |      ----------
 |      X : array-like of shape (n_samples, n_features)
 |          Test samples.
 |
 |      y : array-like of shape (n_samples,) or (n_samples, n_outputs)
 |          True labels for `X`.
 |
 |      sample_weight : array-like of shape (n_samples,), default=None
 |          Sample weights.
 |
 |      Returns
 |      -------
 |      score : float
 |          Mean accuracy of ``self.predict(X)`` wrt. `y`.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from sklearn.base.BaseEstimator:
```

```
|
|  __getstate__(self)
|
|  __repr__(self, N_CHAR_MAX=700)
|      Return repr(self).
|
|  __setstate__(self, state)
|
|  get_params(self, deep=True)
|      Get parameters for this estimator.
|
|      Parameters
|      ----------
|      deep : bool, default=True
|          If True, will return the parameters for this estimator and
|          contained subobjects that are estimators.
|
|      Returns
|      -------
|      params : dict
|          Parameter names mapped to their values.
|
|  set_params(self, **params)
|      Set the parameters of this estimator.
|
|      The method works on simple estimators as well as on nested objects
|      (such as :class:`~sklearn.pipeline.Pipeline`). The latter have
|      parameters of the form ``<component>__<parameter>`` so that it's
|      possible to update each component of a nested object.
|
|      Parameters
|      ----------
|      **params : dict
|          Estimator parameters.
|
|      Returns
|      -------
|      self : estimator instance
|          Estimator instance.
```

Use the following code to generate an artificial dataset which contain three classes. Conduct a similar KNN analysis to the dataset and report your accuracy.

In [2]:
```python
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
import numpy as np

centers = [[2, 4], [6, 6], [1, 9]]
n_classes = len(centers)
data, labels = make_blobs(n_samples=150,
                          centers=np.array(centers),
                          random_state=1)

# do a 80-20 split of the data
train_data, test_data, train_labels, test_labels = train_test_split(data, labels, train_size=0.8, test_size=0.2, random_state=12)

# perform a KNN analysis of the simulated data
knn = KNeighborsClassifier(n_neighbors=5, metric='euclidean')
knn.fit(train_data, train_labels)

# Predictions
test_predictions = knn.predict(test_data)

# output accuracy score
accuracy = accuracy_score(test_labels, test_predictions)

# Plot the training data and test predictions
plt.figure(figsize=(8, 6))
plt.scatter(train_data[:, 0], train_data[:, 1], c=train_labels, cmap='viridis', alpha=0.6, label="Training Data")
plt.scatter(test_data[:, 0], test_data[:, 1], c=test_predictions, cmap='coolwarm', marker='x', label="Predicted Test Data")
plt.title(f"KNN Classification Results (Accuracy: {accuracy:.2f})")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()

# Print accuracy
print(f"Test Accuracy: {accuracy:.2f}")
```




```
Test Accuracy: 1.00
```