K Nearest Neighbors (KNN)

Imports

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Loading Dataset

In [2]:
```
from sklearn.datasets import load_iris
```

In [3]:
```
iris = load_iris()
```

In [4]:
```
type(iris)
```

Out[4]:
```
sklearn.utils.Bunch
```

See all the keys

In [5]:
```
iris.keys()
```

Out[5]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filenam
e'])
```

Description

In [6]:
```
print(iris["DESCR"])
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
```

```
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

Data

In [7]:
```
iris["data"][:5] # Independent columns
```

Out[7]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

In [8]:
```
iris["feature_names"] # Column names
```

Out[8]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [9]:
```
np.array(iris["target"]) # Dependent column
```

Out[9]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
```

```
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

In [10]:
```
iris["target_names"]
# So, 0 is setosa, 1 is versicolor and 2 is virginica
```

Out[10]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Create Dataframe

In [11]:
```
df = pd.DataFrame(data=iris["data"], columns=iris["feature_names"])
df["target"] = iris["target"]
df.head()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [12]:
```
df["target"].unique()
```

Out[12]:
```
array([0, 1, 2])
```

In [13]:
```
df["target"].replace(range(3), iris["target_names"], inplace=True)
```

In [14]:
```
df.head()
```

Out[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [15]:
```
df["target"].unique()
```

Out[15]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

EDA

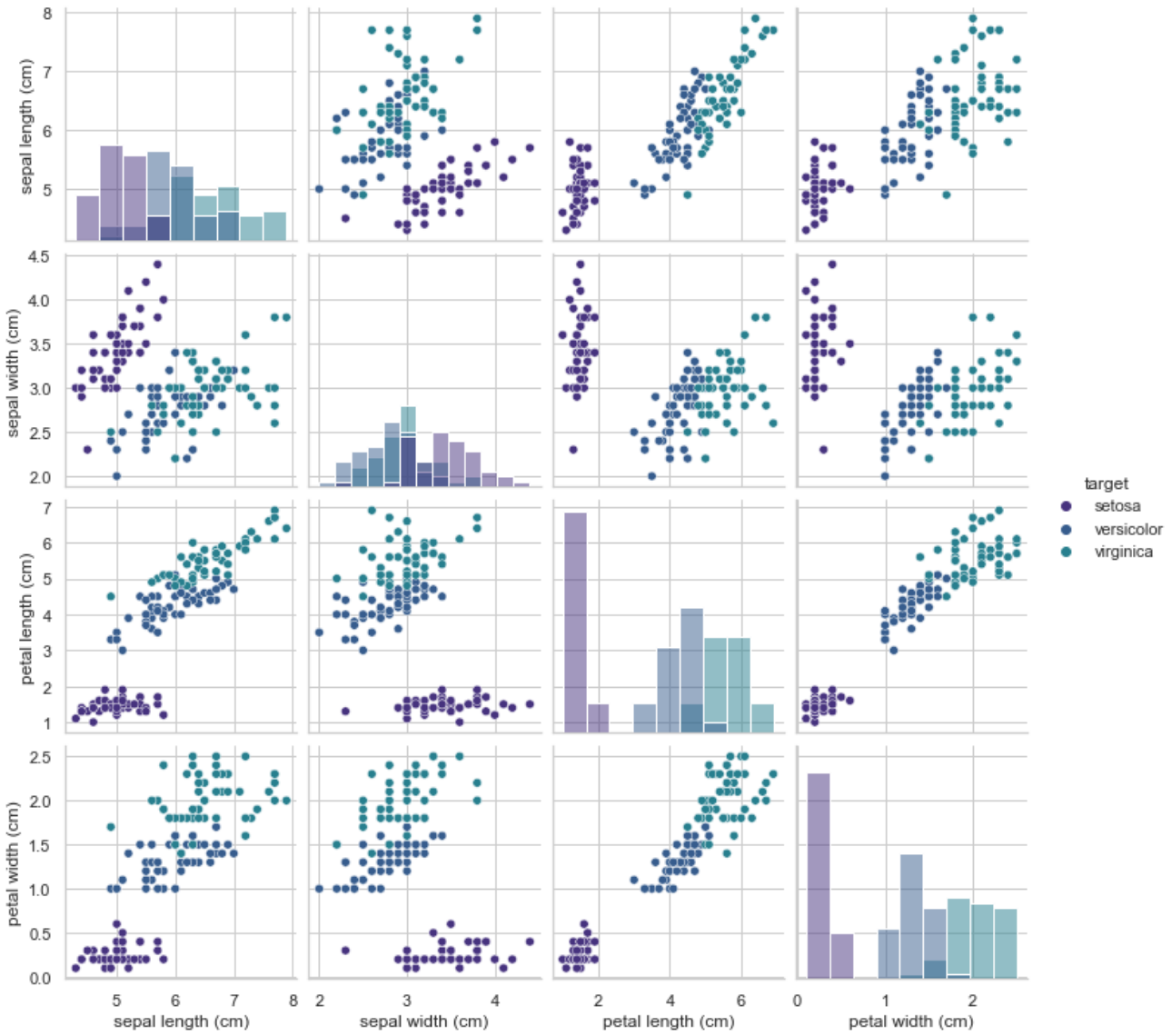
In [16]:
```
sns.set(style="whitegrid", palette="viridis")
```

In [17]:
```

```

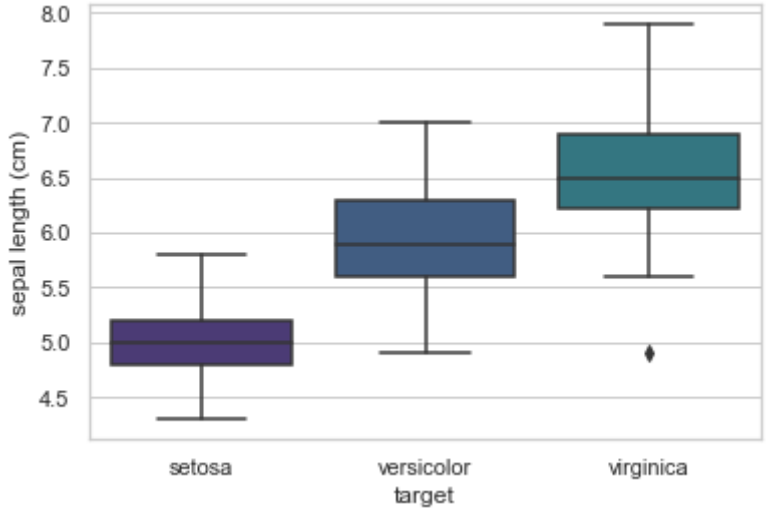
```
sns.pairplot(df, hue="target", diag_kind='hist')
```

Out[17]: <seaborn.axisgrid.PairGrid at 0x1fa7452ba60>



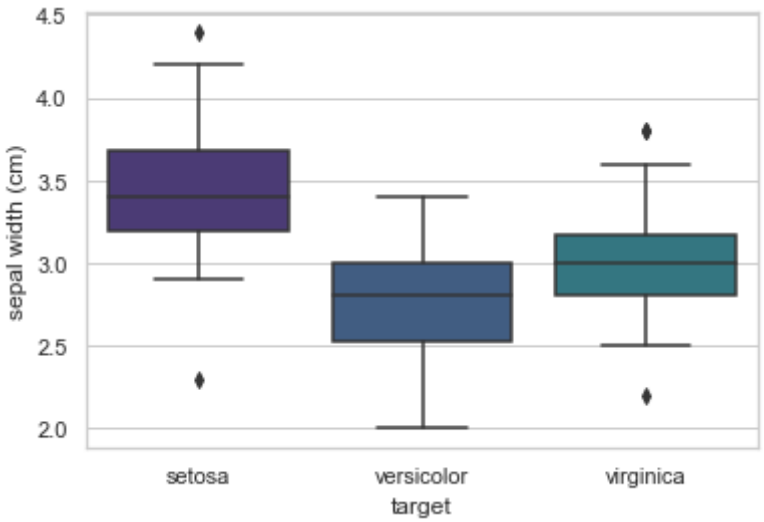
In [18]:
```
sns.boxplot(x="target", y="sepal length (cm)", data=df)
```

Out[18]: <AxesSubplot:xlabel='target', ylabel='sepal length (cm)'>



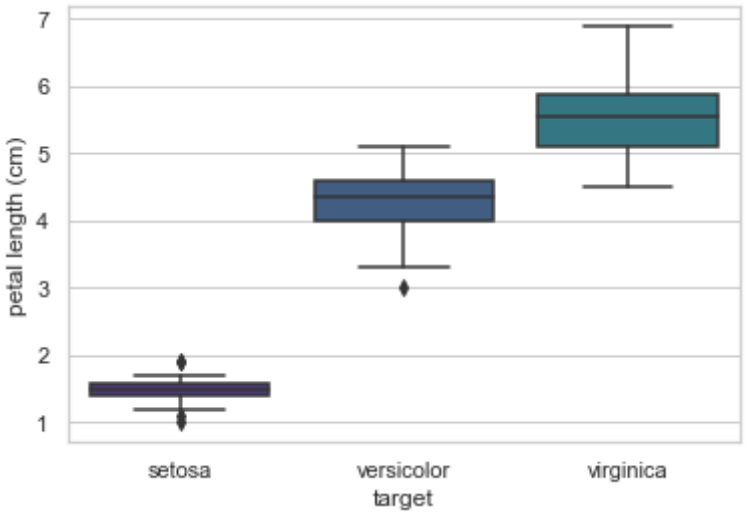
In [19]:
```
sns.boxplot(x="target", y="sepal width (cm)", data=df)
```

Out[19]: <AxesSubplot:xlabel='target', ylabel='sepal width (cm)'>



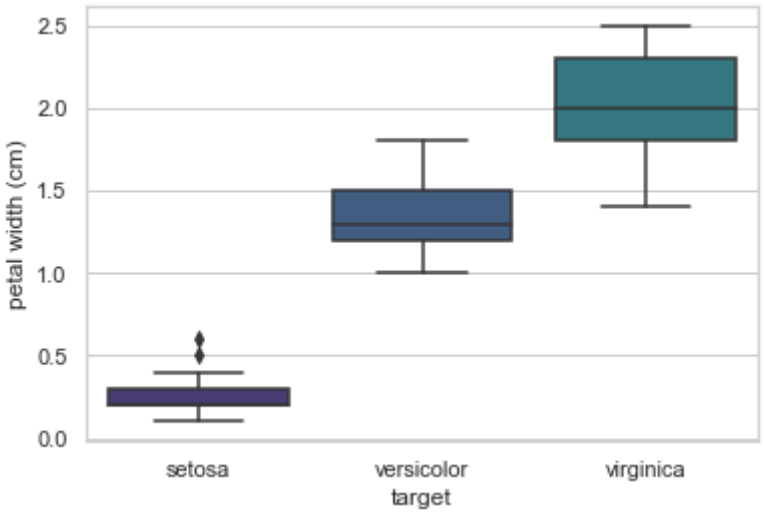
In [20]:
```
sns.boxplot(x="target", y="petal length (cm)", data=df)
```

Out[20]: <AxesSubplot:xlabel='target', ylabel='petal length (cm)'>



In [21]:
```
sns.boxplot(x="target", y="petal width (cm)", data=df)
```

Out[21]: <AxesSubplot:xlabel='target', ylabel='petal width (cm)'>



Feature Selection

In [22]:
```
df.head()
```

Out[22]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [23]:
```
X = df.drop("target", axis=1) # Independent
y = df["target"] # Dependent
```

Train Test Split

In [24]:
```
from sklearn.model_selection import train_test_split
```

In [25]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=202
```

In [26]:
```
X_train[:5]
```

Out[26]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
120	6.9	3.2	5.7	2.3
117	7.7	3.8	6.7	2.2
76	6.8	2.8	4.8	1.4
86	6.7	3.1	4.7	1.5
57	4.9	2.4	3.3	1.0

In [27]:
```
y_train[:5]
```

Out[27]:
```
120     virginica
117     virginica
76     versicolor
86     versicolor
57     versicolor
Name: target, dtype: object
```

K-Nearest Neighbors

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. It stores all the available cases and classifies the new data or case based on a similarity measure. It is mostly used to classifies a data point based on how its neighbours are classified. The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

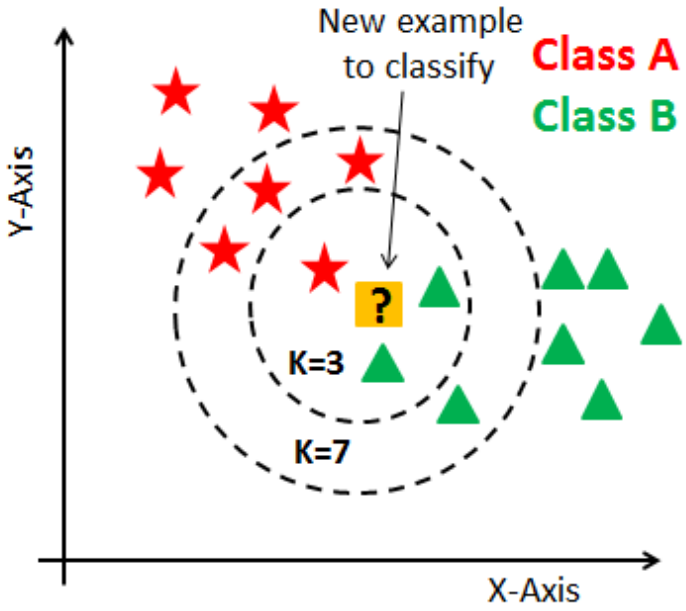
The KNN Algorithm

1. Load the data.
2. Initialize K to your chosen number of neighbors.
3. For each example in the data, calculate the distance between the query example and the current example from the data and add the distance and the index of the example to an ordered collection.
4. Sort the ordered collection of distances and indices in ascending order by the distances.
5. Pick the first K entries from the sorted collection.
6. Get the labels of the selected K entries.
7. If regression, return the mean of the K labels. If classification, return the mode of the K labels.

Choosing the right value for K

To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

- As we decrease the value of K to 1, our predictions become less stable.
- Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.



Advantages

- The algorithm is simple and easy to implement.
- There's no need to build a model, tune several parameters, or make additional assumptions.
- The algorithm is versatile. It can be used for classification, regression, and search.

Disadvantages

- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

- Need to determine the value of parameter K (number of nearest neighbors)
- Computation cost is quite high because we need to compute the distance of each query instance to all training samples.

Article Links: Link 1, Link 2

In [28]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [29]:
```
knn_model = KNeighborsClassifier(n_neighbors=5)
```

In [30]:
```
knn_model.fit(X_train, y_train)
```

Out[30]:
```
KNeighborsClassifier()
```

In [31]:
```
knn_pred = knn_model.predict(X_test)
```

In [32]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [33]:
```
print(confusion_matrix(y_test, knn_pred))
```

```
[[18  0  0]
 [ 0 12  1]
 [ 0  2 12]]
```

In [34]:
```
print(classification_report(y_test, knn_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       0.86      0.92      0.89        13
   virginica       0.92      0.86      0.89        14

    accuracy                           0.93        45
   macro avg       0.93      0.93      0.93        45
weighted avg       0.93      0.93      0.93        45
```

In [35]:
```
print(list(knn_pred != y_test))
```

```
[False, False, False, False, False, True, False, False, False, False, False, True, False,
False, False, False, False, False, False, False, False, False, True, False, False, False,
False, False, False, False, False, False, False, False, False, False, False, Fals
e, False, False, False, False, False, False]
```

In [36]:
```
np.mean(knn_pred != y_test)
```

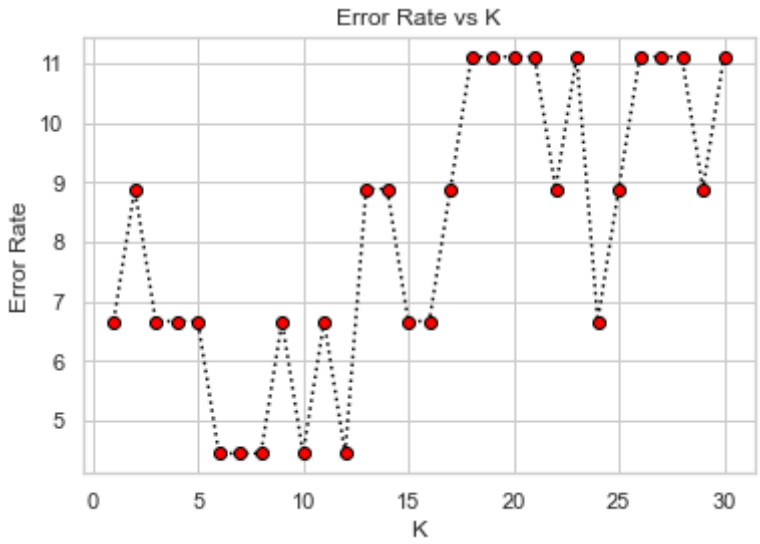
Out[36]:
```
0.06666666666666667
```

In [37]:
```
error = []
for i in range(1, 31):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
```

```
    error.append(np.mean(pred != y_test))

# Alternatively, you can use accuracy score instead of calculating error manually
# Keep in mind, in this case lower value is better
# In case of accuracy score, higher value will be better
```

In [38]:
```
plt.plot(range(1, 31), np.array(error) * 100, color='k', ls=':', marker='o', markerfaceco
plt.title('Error Rate vs K')
plt.xlabel('K')
plt.ylabel('Error Rate')
plt.show()
```



In [39]:
```
knn_model = KNeighborsClassifier(n_neighbors=7)
```

In [40]:
```
knn_model.fit(X_train, y_train)
```

Out[40]:
```
KNeighborsClassifier(n_neighbors=7)
```

In [41]:
```
knn_pred = knn_model.predict(X_test)
```

In [42]:
```
print(confusion_matrix(y_test, knn_pred))
```

```
[[18  0  0]
 [ 0 12  1]
 [ 0  1 13]]
```

In [43]:
```
print(classification_report(y_test, knn_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        18
  versicolor       0.92      0.92      0.92        13
   virginica       0.93      0.93      0.93        14

    accuracy                           0.96        45
   macro avg       0.95      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```

Exercise

Write a function that will take a list of models and will return a list of accuracy scores correponding to each model.

In [ ]:
```
def func(models):
    score = []
    # Write your answer here
    return score

models = [
    LogisticRegression(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    KNeighborsClassifier(n_neighbors=7)
    # Assume that list will contain an arbitrary no. of models
]

print(func(models))
```

Solution:

In [44]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

def func(models):
    score = []
    for model in models:
        model.fit(X_train,y_train)
        pred = model.predict(X_test)
        score.append(accuracy_score(y_test, pred))
    return score

models = [
    LogisticRegression(),
    DecisionTreeClassifier(),
    RandomForestClassifier(),
    KNeighborsClassifier(n_neighbors=7)
]
print(func(models))
```

```
[0.9111111111111111, 0.9111111111111111, 0.9111111111111111, 0.9555555555555556]
```