WEEK 7

September 25, 2024

[1]:
```python
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

[2]:
```python
iris = pd.read_csv("Iris.csv")
```

[3]:
```python
iris.head()
```

[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

[4]:
```python
iris.info()
```

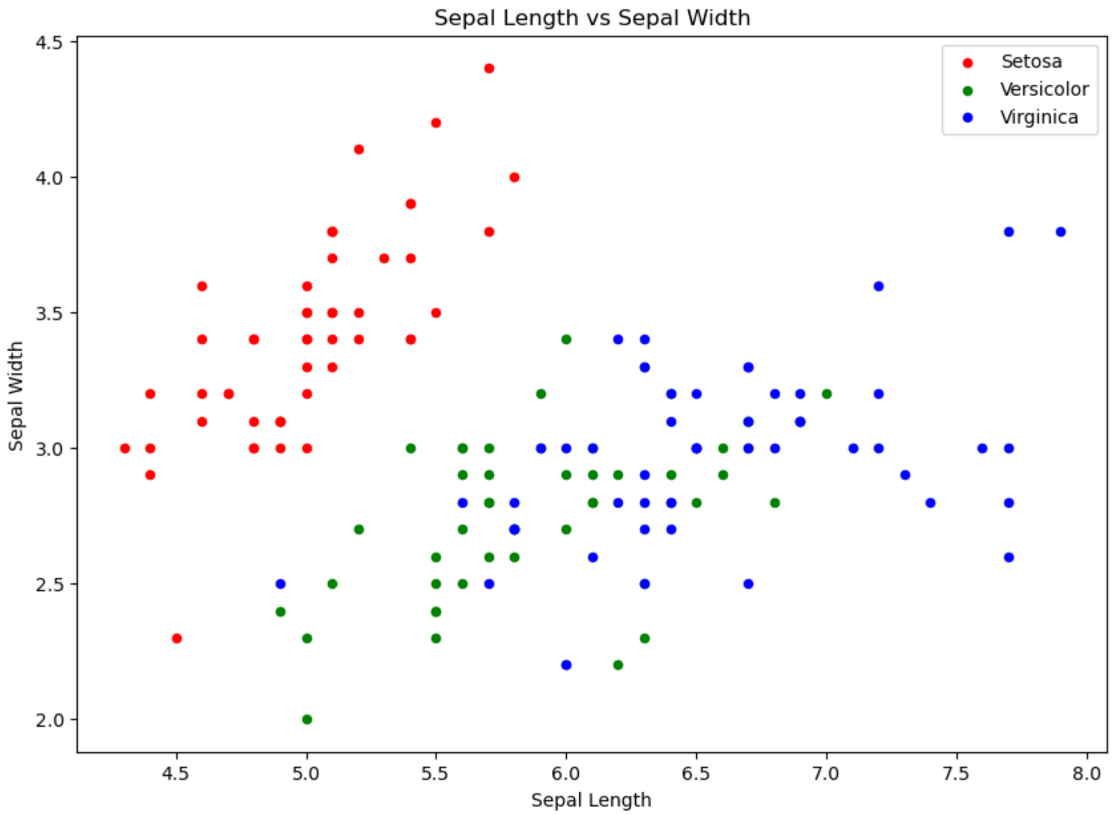
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

1 Removing Unneed column

[5]:
```
iris.drop("Id", axis=1, inplace = True)
```

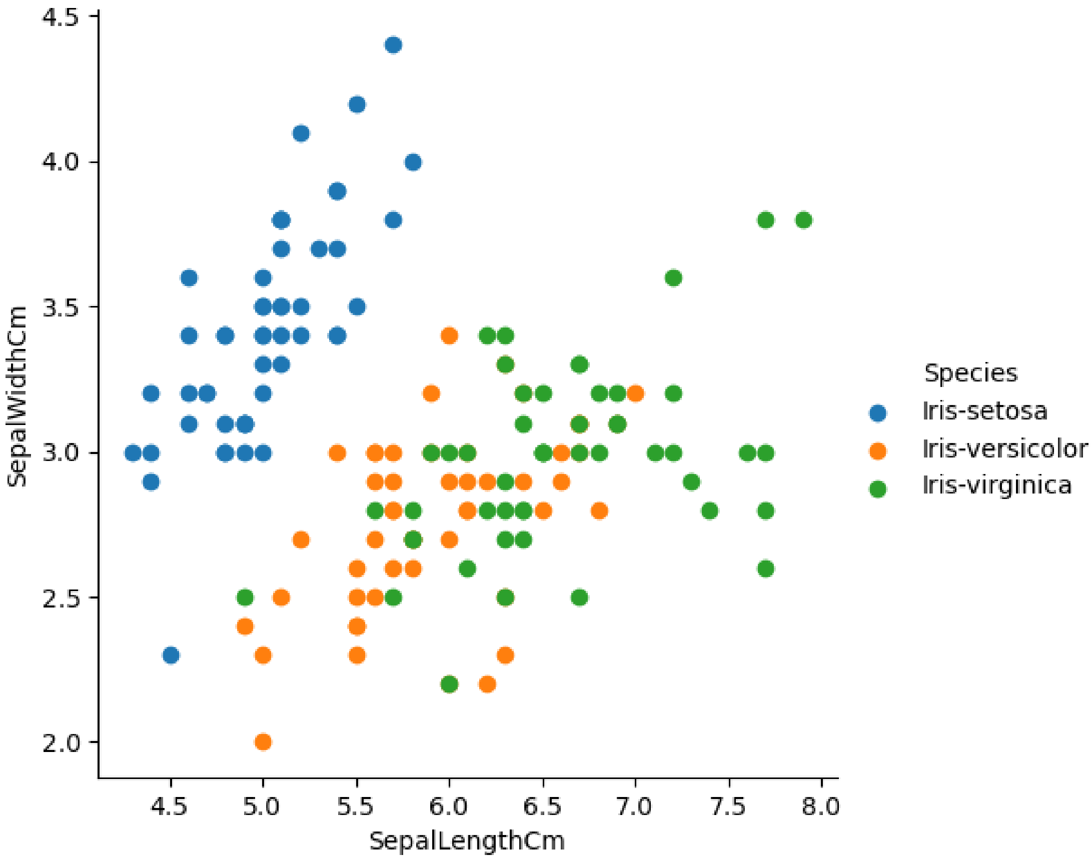
2 Some EDA with iris

[7]:
```
fig, ax = plt.subplots(figsize=(10, 7))
iris[iris.Species == 'Iris-setosa'].plot(kind='scatter', x='SepalLengthCm',␣
 ↪y='SepalWidthCm', color='red', label='Setosa', ax=ax)
iris[iris.Species == 'Iris-versicolor'].plot(kind='scatter', x='SepalLengthCm',␣
 ↪y='SepalWidthCm', color='green', label='Versicolor', ax=ax)
iris[iris.Species == 'Iris-virginica'].plot(kind='scatter', x='SepalLengthCm',␣
 ↪y='SepalWidthCm', color='blue', label='Virginica', ax=ax)

ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_title('Sepal Length vs Sepal Width')
ax.legend()
plt.show()
```



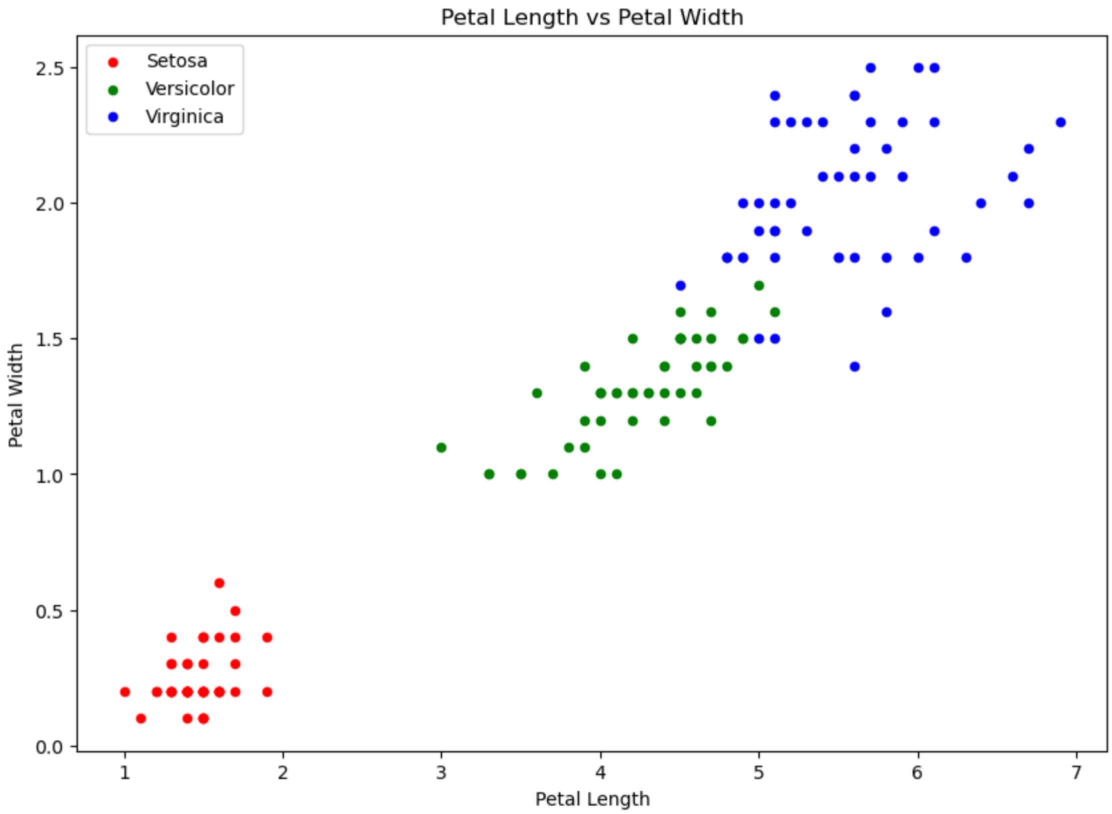
[10]:

```
sns.FacetGrid(iris, hue='Species', height=5) \
    .map(plt.scatter, 'SepalLengthCm', 'SepalWidthCm') \
    .add_legend()

plt.show()
```

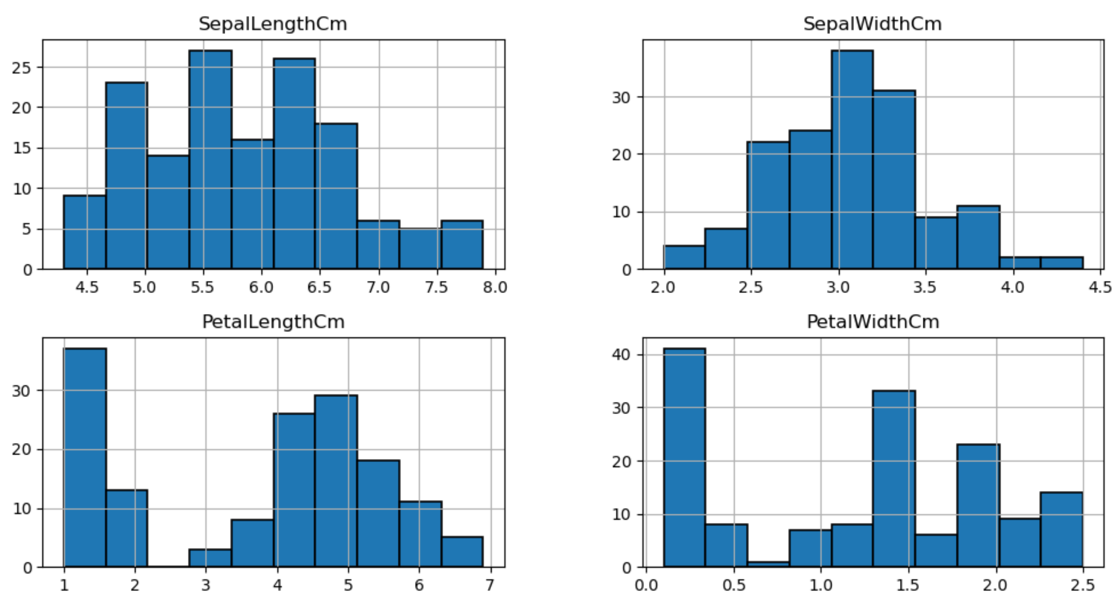


[11]:

```
fig, ax = plt.subplots(figsize=(10, 7))
iris[iris.Species == 'Iris-setosa'].plot(kind='scatter', x='PetalLengthCm',␣
 ↪y='PetalWidthCm', color='red', label='Setosa', ax=ax)
iris[iris.Species == 'Iris-versicolor'].plot(kind='scatter', x='PetalLengthCm',␣
 ↪y='PetalWidthCm', color='green', label='Versicolor', ax=ax)
iris[iris.Species == 'Iris-virginica'].plot(kind='scatter', x='PetalLengthCm',␣
 ↪y='PetalWidthCm', color='blue', label='Virginica', ax=ax)
ax.set_xlabel('Petal Length')
ax.set_ylabel('Petal Width')
ax.set_title('Petal Length vs Petal Width')
ax.legend()
plt.show()
```

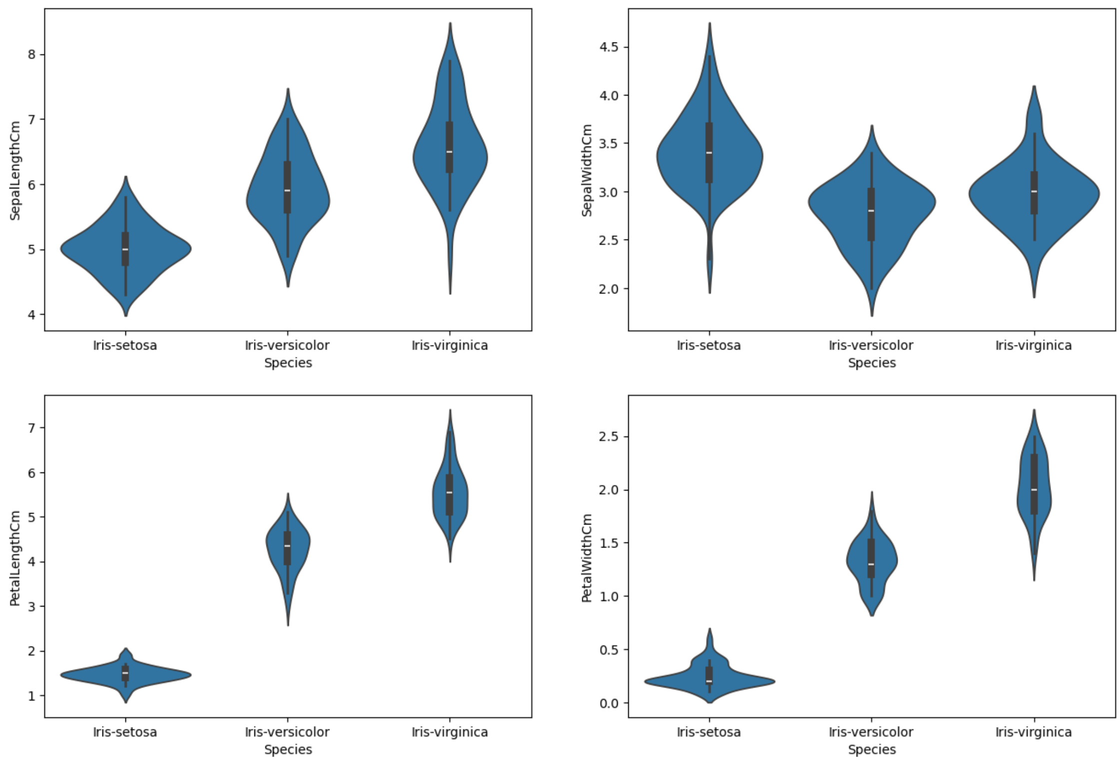


[12]:
```
iris.hist(edgecolor='black', linewidth=1.2)
fig = plt.gcf()
fig.set_size_inches(12,6)
plt.show()
```



[13]:
```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.violinplot(x='Species', y = 'SepalLengthCm', data=iris)
plt.subplot(2,2,2)
sns.violinplot(x='Species', y = 'SepalWidthCm', data=iris)
plt.subplot(2,2,3)
sns.violinplot(x='Species', y = 'PetalLengthCm', data=iris)
plt.subplot(2,2,4)
sns.violinplot(x='Species', y = 'PetalWidthCm', data=iris)
```

[13]:
```
<Axes: xlabel='Species', ylabel='PetalWidthCm'>
```



3 Now the given problem is a classification problem..

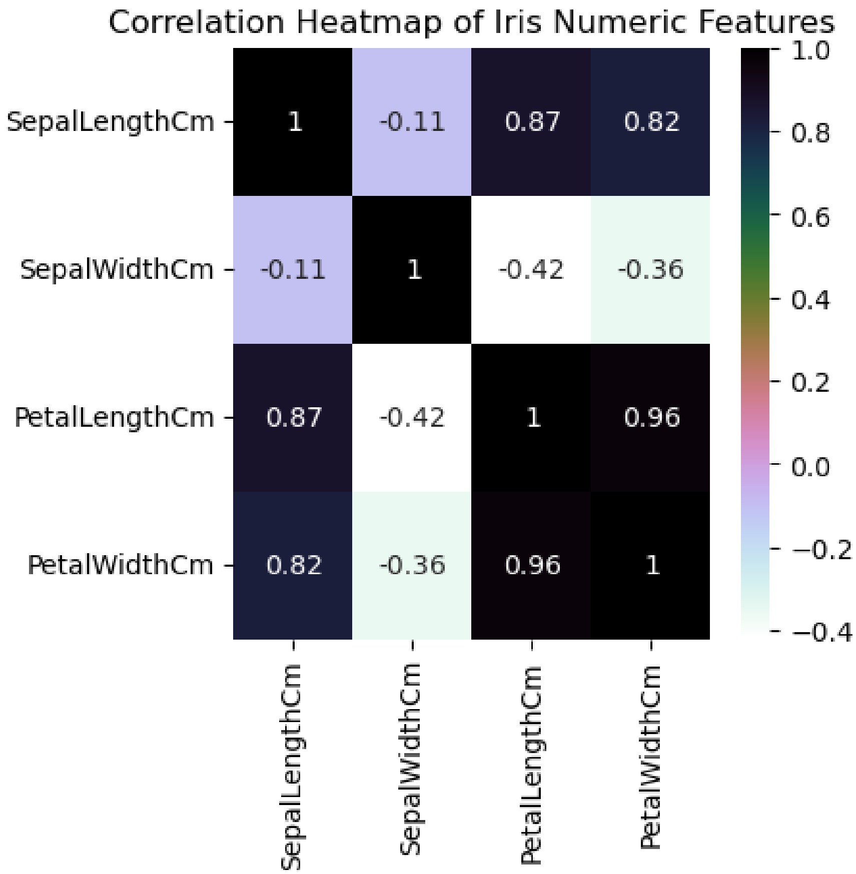
Thus we will be using the classification algorithms to build a model. Classification: Samples belong to two or more classes and we want to learn from already labeled data how to predict the class of unlabeled data Regression: If the desired output consists of one or more continuous variables, then the task is called regression. An example of a regression problem would be the prediction of the length of a salmon as a function of its age and weight. Before we start, we need to clear some ML notations. attributes–>An attribute is a property of an instance that may be used to determine its classification. In the following dataset, the attributes are the petal and sepal length and width. It is also known as Features. Target variable, in the machine learning context is the variable that is or should be the output. Here the target variables are the 3 flower species.

[16]:
```
from sklearn.linear_model import LogisticRegression  # for Logistic Regression␣
↪Algorithm
from sklearn.model_selection import train_test_split  # to split the dataset␣
↪for training and testing
from sklearn.neighbors import KNeighborsClassifier  # KNN classifier
from sklearn import svm  # for Support Vector Machine algorithm
from sklearn import metrics # for checking the model accuracy
from sklearn.tree import DecisionTreeClassifier # for using DTA
```

[17]:
```
iris.shape
```

[17]:
```
(150, 5)
```

[24]:
```
import matplotlib.pyplot as plt
import seaborn as sns
iris_numeric = iris.select_dtypes(include=['float64', 'int64'])
plt.figure(figsize=(4, 4))
sns.heatmap(iris_numeric.corr(), annot=True, cmap='cubehelix_r')
plt.title('Correlation Heatmap of Iris Numeric Features')
plt.show()
```



[25]:
```
train, test = train_test_split(iris, test_size=0.3)
print(train.shape)
print(test.shape)
```

```
(105, 5)
(45, 5)
```

[26]:
```
train_X = train[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
train_y = train.Species

test_X = test[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']]
test_y = test.Species
```

[27]:
```
train_X.head()
```

[27]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
104	6.5	3.0	5.8	2.2
119	6.0	2.2	5.0	1.5
71	6.1	2.8	4.0	1.3
45	4.8	3.0	1.4	0.3

```
5              5.4           3.9            1.7           0.4
```

```
[29]: test_X.head()
```

```
[29]:      SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
      50             7.0           3.2            4.7           1.4
      105            7.6           3.0            6.6           2.1
      72             6.3           2.5            4.9           1.5
      91             6.1           3.0            4.6           1.4
      134            6.1           2.6            5.6           1.4
```

```
[30]: train_y.head()
```

```
[30]: 104     Iris-virginica
      119     Iris-virginica
      71     Iris-versicolor
      45         Iris-setosa
      5          Iris-setosa
      Name: Species, dtype: object
```

4 Support Vector Machine SVM

```
[35]: model = svm.SVC() # select the svm algorithm

      # we train the algorithm with training data and training output
      model.fit(train_X, train_y)

      # we pass the testing data to the stored algorithm to predict the outcome
      prediction = model.predict(test_X)
      print('The accuracy of the SVM is: ', metrics.accuracy_score(prediction,␣
       ↪test_y)) #
       #we pass the predicted output by the model and the actual output
```

```
The accuracy of the SVM is:  0.9777777777777777
```

5 Logistic Regression

```
[36]: from sklearn.linear_model import LogisticRegression
      from sklearn.metrics import accuracy_score

      # Assuming train_X, train_y, test_X, test_y are already defined

      # Create and train the model
      model = LogisticRegression()
      model.fit(train_X, train_y)
```

```
# Make predictions on the test set
prediction = model.predict(test_X)

# Calculate and print the accuracy
accuracy = accuracy_score(test_y, prediction)
print('The accuracy of Logistic Regression is:', accuracy)
```

```
The accuracy of Logistic Regression is: 0.9777777777777777
```

6 Decision Trees

[37]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Assuming train_X, train_y, test_X, test_y are already defined

# Create and train the model
model = DecisionTreeClassifier()
model.fit(train_X, train_y)

# Make predictions on the test set
prediction = model.predict(test_X)

# Calculate and print the accuracy
accuracy = accuracy_score(test_y, prediction)
print('The accuracy of Decision Tree is:', accuracy)
```

```
The accuracy of Decision Tree is: 0.9333333333333333
```

7 K nearest Neighbour

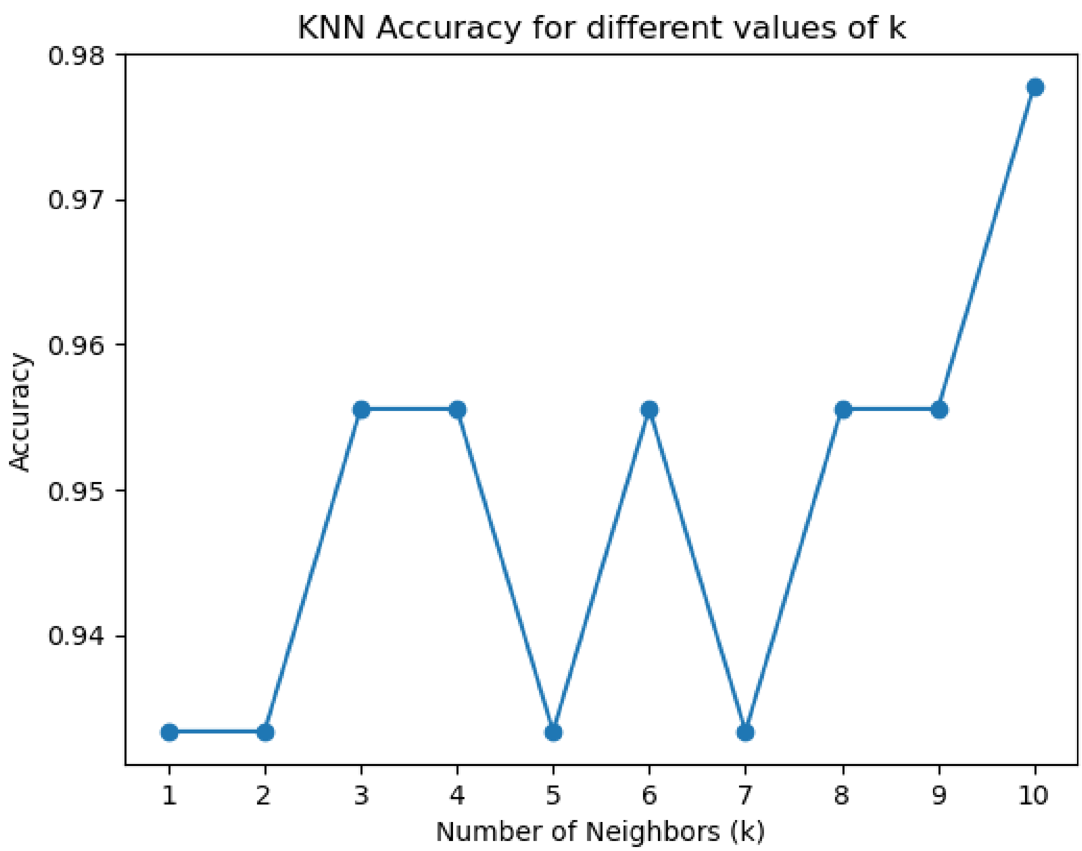
[38]:
```
model = KNeighborsClassifier(n_neighbors=3)
model.fit(train_X, train_y)
prediction = model.predict(test_X)
print('The accuracy of KNN is: ', metrics.accuracy_score(prediction, test_y))
```

```
The accuracy of KNN is:  0.9555555555555556
```

8 Let's check the accuracy for various values of n for K-Nearest Neighbors

[51]:
```
a_index = list(range(1, 11))
accuracy_scores = []
for i in a_index:
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(train_X, train_y)
```

```
    prediction = model.predict(test_X)

    accuracy_scores.append(accuracy_score(test_y, prediction))
a = pd.Series(accuracy_scores)
plt.plot(a_index, a, marker='o')
plt.xticks(a_index)
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('KNN Accuracy for different values of k')
plt.show()
```



[47]:
```
train_X.head(5)
```

[47]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
104	6.5	3.0	5.8	2.2
119	6.0	2.2	5.0	1.5
71	6.1	2.8	4.0	1.3
45	4.8	3.0	1.4	0.3
5	5.4	3.9	1.7	0.4

[48]:
```
train_y.head(5)
```

[48]:
```
104     Iris-virginica
119     Iris-virginica
71     Iris-versicolor
45         Iris-setosa
5          Iris-setosa
Name: Species, dtype: object
```

9 Creatig Petals and sepals training data

[52]:
```
petal = iris[['PetalLengthCm','PetalWidthCm','Species']]
sepal = iris[['SepalLengthCm','SepalWidthCm','Species']]
```

10 For IRIS petals

[53]:
```
train_p,test_p = train_test_split(petal, test_size=0.3, random_state=0) #petals
train_x_p = train_p[['PetalWidthCm','PetalLengthCm']]
train_y_p = train_p.Species
test_x_p = test_p[['PetalWidthCm','PetalLengthCm']]
test_y_p = test_p.Species
```

11 For IRIS sepals

[54]:
```
train_s,test_s = train_test_split(sepal, test_size=0.3, random_state=0) #sepals
train_x_s = train_s[['SepalWidthCm','SepalLengthCm']]
train_y_s = train_s.Species
test_x_s = test_s[['SepalWidthCm','SepalLengthCm']]
test_y_s = test_s.Species
```

12 SVM Algorithm

[55]:
```
model = svm.SVC()
model.fit(train_x_p, train_y_p)
prediction_p = model.predict(test_x_p)
print('The accuracy of the SVM using Petals is:', accuracy_score(test_y_p,
 ↪prediction_p))
model = svm.SVC()
model.fit(train_x_s, train_y_s)
prediction_s = model.predict(test_x_s)
print('The accuracy of the SVM using Sepals is:', accuracy_score(test_y_s,
 ↪prediction_s))
```

```
The accuracy of the SVM using Petals is: 0.9777777777777777
The accuracy of the SVM using Sepals is: 0.8
```

13 Logistic Regression

[56]:
```
model = LogisticRegression()
model.fit(train_x_p, train_y_p)
prediction_p = model.predict(test_x_p)
print('The accuracy of the Logistic Regression using Petals is:',␣
    ↪accuracy_score(test_y_p, prediction_p))
model = LogisticRegression()
model.fit(train_x_s, train_y_s)
prediction_s = model.predict(test_x_s)
print('The accuracy of the Logistic Regression using Sepals is:',␣
    ↪accuracy_score(test_y_s, prediction_s))
```

```
The accuracy of the Logistic Regression using Petals is: 0.9777777777777777
The accuracy of the Logistic Regression using Sepals is: 0.8222222222222222
```

14 Decision Tree

[57]:
```
model = DecisionTreeClassifier()
model.fit(train_x_p, train_y_p)
prediction_p = model.predict(test_x_p)
print('The accuracy of the Decision Tree using Petals is:',␣
    ↪accuracy_score(test_y_p, prediction_p))
model.fit(train_x_s, train_y_s)
prediction_s = model.predict(test_x_s)
print('The accuracy of the Decision Tree using Sepals is:',␣
    ↪accuracy_score(test_y_s, prediction_s))
```

```
The accuracy of the Decision Tree using Petals is: 0.9555555555555556
The accuracy of the Decision Tree using Sepals is: 0.6444444444444445
```

15 K Nearest Neighbor

[58]:
```
model = KNeighborsClassifier(n_neighbors=3)
model.fit(train_x_p, train_y_p)
prediction_p = model.predict(test_x_p)
print('The accuracy of the KNN using Petals is:', accuracy_score(test_y_p,␣
    ↪prediction_p))
model.fit(train_x_s, train_y_s)
prediction_s = model.predict(test_x_s)
print('The accuracy of the KNN using Sepals is:', accuracy_score(test_y_s,␣
    ↪prediction_s))
```

```
The accuracy of the KNN using Petals is: 0.9777777777777777
The accuracy of the KNN using Sepals is: 0.7333333333333333
```

[59]:
```
import numpy as np
from sklearn import datasets
iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
```

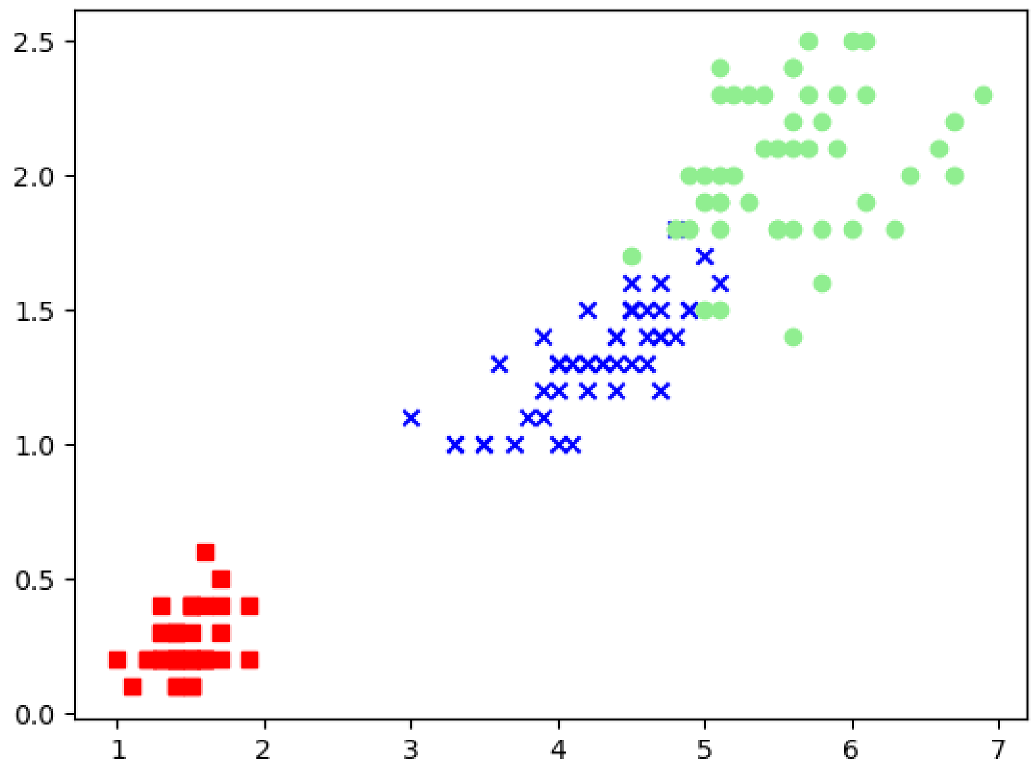
[60]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=0)
print('There are {} samples in the training set and {} samples in the test set'.
 ↪format(
X_train.shape[0], X_test.shape[0]))
```

```
There are 105 samples in the training set and 45 samples in the test set
```

[61]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
X_combined_std = np.vstack((X_train_std, X_test_std))
y_combined = np.hstack((y_train, y_test))
```

[64]:
```
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
markers = ('s', 'x', 'o')
colors = ('red', 'blue', 'lightgreen')
cmap = ListedColormap(colors[:len(np.unique(y_test))])
for idx, cl in enumerate(np.unique(y)):
    plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
    c=cmap(idx), marker=markers[idx], label=cl)
```

```
C:\Users\skand\AppData\Local\Temp\ipykernel_27496\2015837998.py:7: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
```




```
[65]: def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
          markers = ('s', 'x', 'o', '^', 'v')
          colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
          cmap = ListedColormap(colors[:len(np.unique(y))])
          x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
          x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
          xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                                 np.arange(x2_min, x2_max, resolution))
          Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
          Z = Z.reshape(xx1.shape)
          plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
          plt.xlim(xx1.min(), xx1.max())
          plt.ylim(xx2.min(), xx2.max())

          for idx, cl in enumerate(np.unique(y)):
              plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                          alpha=0.8, c=cmap(idx),
                          marker=markers[idx], label=cl)
          if test_idx:
              X_test, y_test = X[test_idx, :], y[test_idx]
              plt.scatter(X_test[:, 0], X_test[:, 1], c='',
                          alpha=1.0, linewidth=1, marker='o',
```

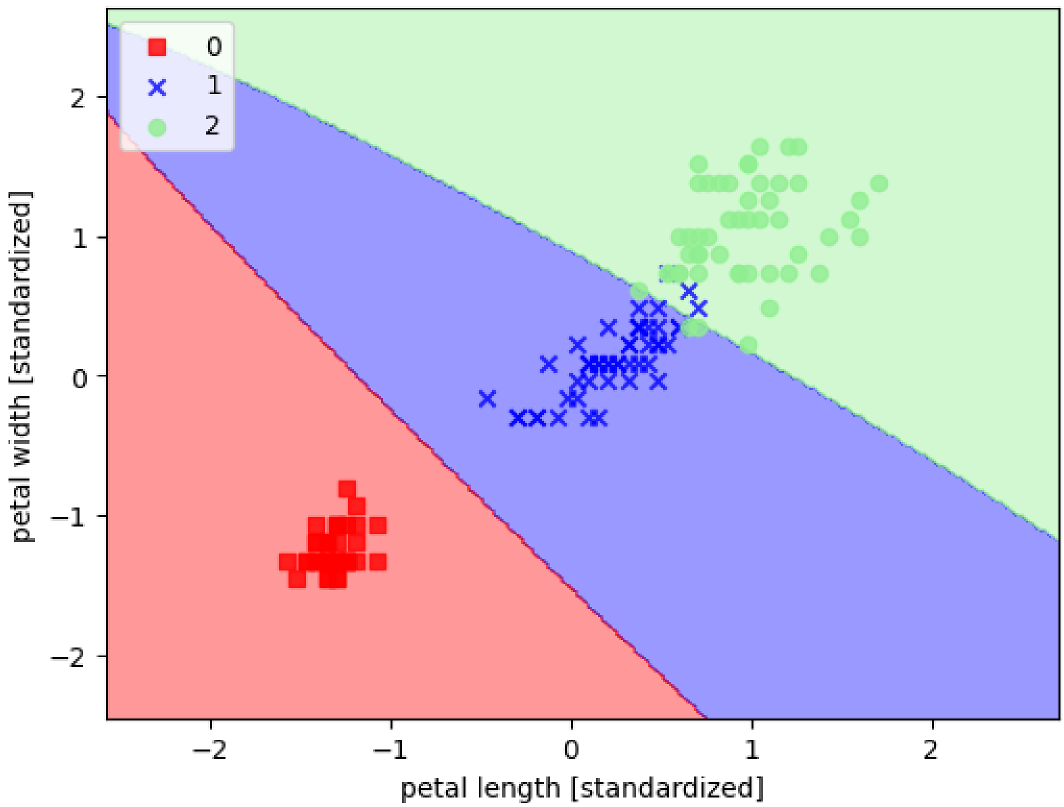
```
                     s=55, label="test set")
```

[67]:
```
from sklearn.svm import SVC
svm = SVC(kernel='rbf', random_state=0, gamma=0.10, C=1.0)
svm.fit(X_train_std, y_train)
print('The accuracy of the SVM classifier on training data is {:.2f} out of 1'.
 ↪format(svm.score(X_train_std, y_train)))
print('The accuracy of the SVM classifier on test data is {:.2f} out of 1'.
 ↪format(svm.score(X_test_std, y_test)))
```

```
The accuracy of the SVM classifier on training data is 0.95 out of 1
The accuracy of the SVM classifier on test data is 0.98 out of 1
```

[69]:
```
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=svm,␣
 ↪test_idx=range(105,15))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

```
C:\Users\skand\AppData\Local\Temp\ipykernel_27496\696551294.py:16: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
single row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
```



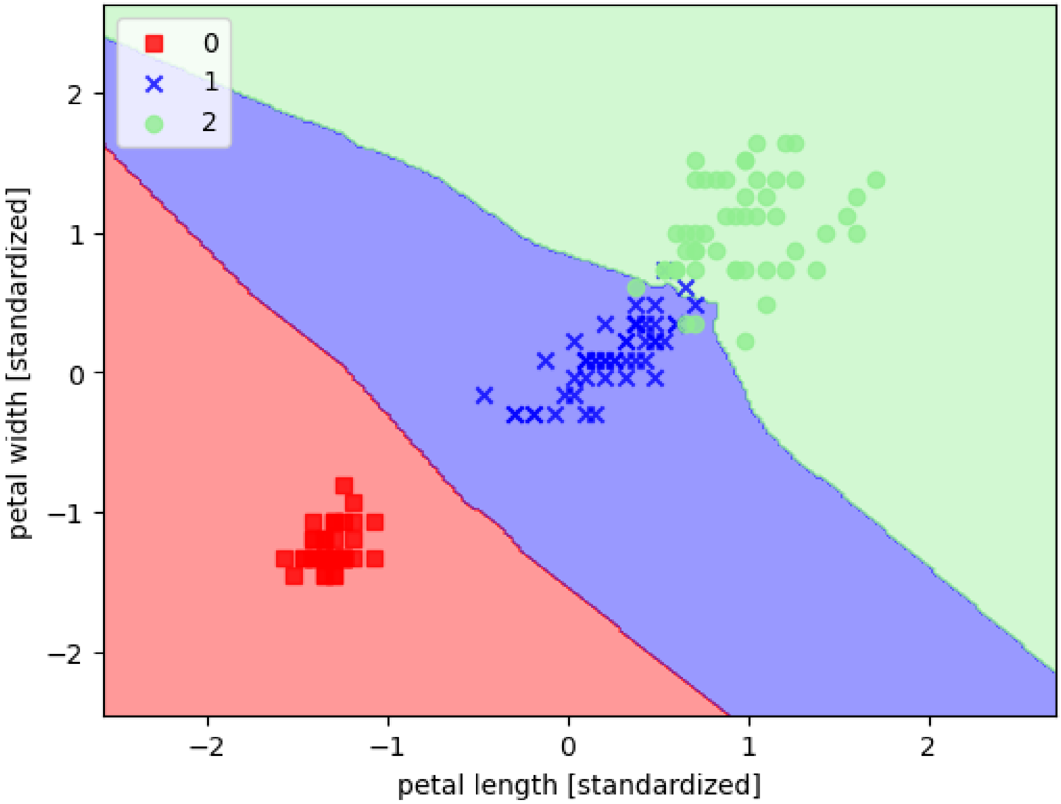
[70]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
knn.fit(X_train_std, y_train)
print('The accuracy of the KNN classifier on training data is {:.2f} out of 1'.
  ↪format(knn.score(X_train_std, y_train)))
print('The accuracy of the KNN classifier on test data is {:.2f} out of 1'.
  ↪format(knn.score(X_test_std, y_test)))
```

```
The accuracy of the KNN classifier on training data is 0.95 out of 1
The accuracy of the KNN classifier on test data is 1.00 out of 1
```

[72]:
```
plot_decision_regions(X=X_combined_std, y=y_combined, classifier=knn,␣
  ↪test_idx=range(105,15))
plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

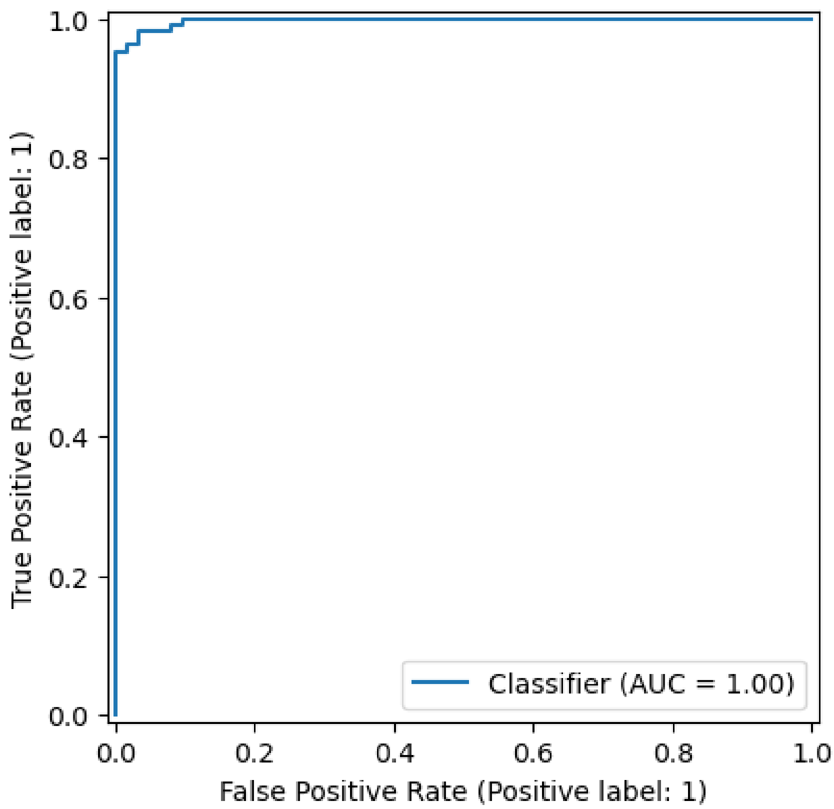
```
C:\Users\skand\AppData\Local\Temp\ipykernel_27496\696551294.py:16: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a
```

```
single row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
```



[6]:
```python
from sklearn.metrics import RocCurveDisplay
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# Load dataset
data = load_breast_cancer()
X, y = data.data, data.target

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)

# Train model
model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)
```

```
# Get prediction scores
y_score = model.decision_function(X_test)

# Compute ROC curve
roc_display = RocCurveDisplay.from_predictions(y_test, y_score)

# Show plot
plt.show()
```



[7]:
```
data = load_breast_cancer()
data
```

[7]:
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, …, 2.654e-01, 4.601e-01,
          1.189e-01],
         [2.057e+01, 1.777e+01, 1.329e+02, …, 1.860e-01, 2.750e-01,
          8.902e-02],
         [1.969e+01, 2.125e+01, 1.300e+02, …, 2.430e-01, 3.613e-01,
          8.758e-02],
         …,
         [1.660e+01, 2.808e+01, 1.083e+02, …, 1.418e-01, 2.218e-01,
```

```
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, …, 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, …, 0.000e+00, 2.871e-01,
        7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 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, 0, 0, 0, 0, 0, 0, 1]),
 'frame': None,
 'target_names': array(['malignant', 'benign'], dtype='<U9'),
 'DESCR': '.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
dataset\n--------------------------------------------\n\n**Data Set
Characteristics:**\n\n:Number of Instances: 569\n\n:Number of Attributes: 30
numeric, predictive attributes and the class\n\n:Attribute Information:\n    -
radius (mean of distances from center to points on the perimeter)\n    - texture
(standard deviation of gray-scale values)\n    - perimeter\n    - area\n    -
smoothness (local variation in radius lengths)\n    - compactness (perimeter^2 /
area - 1.0)\n    - concavity (severity of concave portions of the contour)\n
- concave points (number of concave portions of the contour)\n    - symmetry\n
- fractal dimension ("coastline approximation" - 1)\n\n    The mean, standard
error, and "worst" or largest (mean of the three\n    worst/largest values) of
these features were computed for each image,\n    resulting in 30 features.  For
instance, field 0 is Mean Radius, field\n    10 is Radius SE, field 20 is Worst
```

```
Radius.\n\n    - class:\n            - WDBC-Malignant\n            - WDBC-
Benign\n\n:Summary Statistics:\n\n===================================== ======
======\n                                       Min
Max\n===================================== ====== ======\nradius (mean):
6.981  28.11\ntexture (mean):                      9.71   39.28\nperimeter
(mean):                   43.79  188.5\narea (mean):
143.5  2501.0\nsmoothness (mean):                   0.053  0.163\ncompactness
(mean):                 0.019  0.345\nconcavity (mean):
0.0    0.427\nconcave points (mean):               0.0    0.201\nsymmetry
(mean):                    0.106  0.304\nfractal dimension (mean):
0.05   0.097\nradius (standard error):             0.112  2.873\ntexture
(standard error):            0.36   4.885\nperimeter (standard error):
0.757  21.98\narea (standard error):               6.802  542.2\nsmoothness
(standard error):         0.002  0.031\ncompactness (standard error):
0.002  0.135\nconcavity (standard error):          0.0    0.396\nconcave points
(standard error):     0.0    0.053\nsymmetry (standard error):            0.008
0.079\nfractal dimension (standard error):   0.001  0.03\nradius (worst):
7.93   36.04\ntexture (worst):                     12.02  49.54\nperimeter
(worst):                   50.41  251.2\narea (worst):
185.2  4254.0\nsmoothness (worst):                  0.071  0.223\ncompactness
(worst):                 0.027  1.058\nconcavity (worst):
0.0    1.252\nconcave points (worst):              0.0    0.291\nsymmetry
(worst):                    0.156  0.664\nfractal dimension (worst):
0.055  0.208\n===================================== ====== ======\n\n:Missing
Attribute Values: None\n\n:Class Distribution: 212 - Malignant, 357 -
Benign\n\n:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L.
Mangasarian\n\n:Donor: Nick Street\n\n:Date: November, 1995\n\nThis is a copy of
UCI ML Breast Cancer Wisconsin (Diagnostic)
datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image
of a fine needle\naspirate (FNA) of a breast mass.  They
describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating
plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K.
P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of
the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp.
97-101, 1992], a classification method which uses linear\nprogramming to
construct a decision tree.  Relevant features\nwere selected using an exhaustive
search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual
linear program used to obtain the separating plane\nin the 3-dimensional space
is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust
Linear\nProgramming Discrimination of Two Linearly Inseparable
Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is
also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-
prog/cpo-dataset/machine-learn/WDBC/\n\n.. dropdown:: References\n\n  - W.N.
Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction\n    for
breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on\n
Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n    San
Jose, CA, 1993.\n  - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast
```

```
cancer diagnosis and\n    prognosis via linear programming. Operations Research,
43(4), pages 570-577,\n    July-August 1995.\n  - W.H. Wolberg, W.N. Street, and
O.L. Mangasarian. Machine learning techniques\n    to diagnose breast cancer
from fine-needle aspirates. Cancer Letters 77 (1994)\n    163-171.\n',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'mean
area',
        'mean smoothness', 'mean compactness', 'mean concavity',
        'mean concave points', 'mean symmetry', 'mean fractal dimension',
        'radius error', 'texture error', 'perimeter error', 'area error',
        'smoothness error', 'compactness error', 'concavity error',
        'concave points error', 'symmetry error',
        'fractal dimension error', 'worst radius', 'worst texture',
        'worst perimeter', 'worst area', 'worst smoothness',
        'worst compactness', 'worst concavity', 'worst concave points',
        'worst symmetry', 'worst fractal dimension'], dtype='<U23'),
 'filename': 'breast_cancer.csv',
 'data_module': 'sklearn.datasets.data'}
```

[8]:
```
data.keys()
```

[8]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names',
'filename', 'data_module'])
```

[9]:
```
data.DESCR
```

[9]:
```
'.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic)
dataset\n--------------------------------------------\n\n**Data Set
Characteristics:**\n\n:Number of Instances: 569\n\n:Number of Attributes: 30
numeric, predictive attributes and the class\n\n:Attribute Information:\n    -
radius (mean of distances from center to points on the perimeter)\n    - texture
(standard deviation of gray-scale values)\n    - perimeter\n    - area\n    -
smoothness (local variation in radius lengths)\n    - compactness (perimeter^2 /
area - 1.0)\n    - concavity (severity of concave portions of the contour)\n
- concave points (number of concave portions of the contour)\n    - symmetry\n
- fractal dimension ("coastline approximation" - 1)\n\n    The mean, standard
error, and "worst" or largest (mean of the three\n    worst/largest values) of
these features were computed for each image,\n    resulting in 30 features.  For
instance, field 0 is Mean Radius, field\n    10 is Radius SE, field 20 is Worst
Radius.\n\n    - class:\n            - WDBC-Malignant\n            - WDBC-
Benign\n\n:Summary Statistics:\n\n===================================== ======
======\n                                           Min
Max\n===================================== ====== ======\nradius (mean):
6.981  28.11\ntexture (mean):                       9.71   39.28\nperimeter
(mean):                     43.79  188.5\narea (mean):
143.5  2501.0\nsmoothness (mean):                    0.053  0.163\ncompactness
(mean):                   0.019  0.345\nconcavity (mean):
0.0    0.427\nconcave points (mean):                0.0    0.201\nsymmetry
```

```
(mean):                      0.106  0.304\nfractal dimension (mean):
0.05   0.097\nradius (standard error):              0.112  2.873\ntexture
(standard error):             0.36   4.885\nperimeter (standard error):
0.757  21.98\narea (standard error):                6.802  542.2\nsmoothness
(standard error):          0.002  0.031\ncompactness (standard error):
0.002  0.135\nconcavity (standard error):           0.0    0.396\nconcave points
(standard error):      0.0    0.053\nsymmetry (standard error):            0.008
0.079\nfractal dimension (standard error):   0.001  0.03\nradius (worst):
7.93   36.04\ntexture (worst):                      12.02  49.54\nperimeter
(worst):                    50.41  251.2\narea (worst):
185.2  4254.0\nsmoothness (worst):                   0.071  0.223\ncompactness
(worst):                   0.027  1.058\nconcavity (worst):
0.0    1.252\nconcave points (worst):               0.0    0.291\nsymmetry
(worst):                     0.156  0.664\nfractal dimension (worst):
0.055  0.208\n===================================== ====== ======\n\n:Missing
Attribute Values: None\n\n:Class Distribution: 212 - Malignant, 357 -
Benign\n\n:Creator:  Dr. William H. Wolberg, W. Nick Street, Olvi L.
Mangasarian\n\n:Donor: Nick Street\n\n:Date: November, 1995\n\nThis is a copy of
UCI ML Breast Cancer Wisconsin (Diagnostic)
datasets.\nhttps://goo.gl/U2Uwz2\n\nFeatures are computed from a digitized image
of a fine needle\naspirate (FNA) of a breast mass.  They
describe\ncharacteristics of the cell nuclei present in the image.\n\nSeparating
plane described above was obtained using\nMultisurface Method-Tree (MSM-T) [K.
P. Bennett, "Decision Tree\nConstruction Via Linear Programming." Proceedings of
the 4th\nMidwest Artificial Intelligence and Cognitive Science Society,\npp.
97-101, 1992], a classification method which uses linear\nprogramming to
construct a decision tree.  Relevant features\nwere selected using an exhaustive
search in the space of 1-4\nfeatures and 1-3 separating planes.\n\nThe actual
linear program used to obtain the separating plane\nin the 3-dimensional space
is that described in:\n[K. P. Bennett and O. L. Mangasarian: "Robust
Linear\nProgramming Discrimination of Two Linearly Inseparable
Sets",\nOptimization Methods and Software 1, 1992, 23-34].\n\nThis database is
also available through the UW CS ftp server:\n\nftp ftp.cs.wisc.edu\ncd math-
prog/cpo-dataset/machine-learn/WDBC/\n\n.. dropdown:: References\n\n  - W.N.
Street, W.H. Wolberg and O.L. Mangasarian. Nuclear feature extraction\n    for
breast tumor diagnosis. IS&T/SPIE 1993 International Symposium on\n
Electronic Imaging: Science and Technology, volume 1905, pages 861-870,\n    San
Jose, CA, 1993.\n  - O.L. Mangasarian, W.N. Street and W.H. Wolberg. Breast
cancer diagnosis and\n    prognosis via linear programming. Operations Research,
43(4), pages 570-577,\n    July-August 1995.\n  - W.H. Wolberg, W.N. Street, and
O.L. Mangasarian. Machine learning techniques\n    to diagnose breast cancer
from fine-needle aspirates. Cancer Letters 77 (1994)\n    163-171.\n'
```

[10]:
```python
df = pd.DataFrame(data.data,
                  columns = data.feature_names)
 # Add the target columns, and fill it with the target data
df["target"] = data.target
```

```python
# Show the dataframe
df
```

[10]:

```
     mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0          17.99         10.38          122.80     1001.0          0.11840
1          20.57         17.77          132.90     1326.0          0.08474
2          19.69         21.25          130.00     1203.0          0.10960
3          11.42         20.38           77.58      386.1          0.14250
4          20.29         14.34          135.10     1297.0          0.10030
..           …            …              …          …               …
564        21.56         22.39          142.00     1479.0          0.11100
565        20.13         28.25          131.20     1261.0          0.09780
566        16.60         28.08          108.30      858.1          0.08455
567        20.60         29.33          140.10     1265.0          0.11780
568         7.76         24.54           47.92      181.0          0.05263

     mean compactness  mean concavity  mean concave points  mean symmetry  \
0             0.27760         0.30010              0.14710         0.2419
1             0.07864         0.08690              0.07017         0.1812
2             0.15990         0.19740              0.12790         0.2069
3             0.28390         0.24140              0.10520         0.2597
4             0.13280         0.19800              0.10430         0.1809
..                …              …                    …             …
564           0.11590         0.24390              0.13890         0.1726
565           0.10340         0.14400              0.09791         0.1752
566           0.10230         0.09251              0.05302         0.1590
567           0.27700         0.35140              0.15200         0.2397
568           0.04362         0.00000              0.00000         0.1587

     mean fractal dimension  …  worst texture  worst perimeter  worst area  \
0                   0.07871  …          17.33           184.60      2019.0
1                   0.05667  …          23.41           158.80      1956.0
2                   0.05999  …          25.53           152.50      1709.0
3                   0.09744  …          26.50            98.87       567.7
4                   0.05883  …          16.67           152.20      1575.0
..                      …   …            …               …           …
564                 0.05623  …          26.40           166.10      2027.0
565                 0.05533  …          38.25           155.00      1731.0
566                 0.05648  …          34.12           126.70      1124.0
567                 0.07016  …          39.42           184.60      1821.0
568                 0.05884  …          30.37            59.16       268.6

     worst smoothness  worst compactness  worst concavity  \
0             0.16220            0.66560           0.7119
1             0.12380            0.18660           0.2416
2             0.14440            0.42450           0.4504
3             0.20980            0.86630           0.6869
```

```
4            0.13740          0.20500          0.4000
..               …                …              …
564          0.14100          0.21130          0.4107
565          0.11660          0.19220          0.3215
566          0.11390          0.30940          0.3403
567          0.16500          0.86810          0.9387
568          0.08996          0.06444          0.0000

     worst concave points  worst symmetry  worst fractal dimension  target
0                  0.2654          0.4601                  0.11890       0
1                  0.1860          0.2750                  0.08902       0
2                  0.2430          0.3613                  0.08758       0
3                  0.2575          0.6638                  0.17300       0
4                  0.1625          0.2364                  0.07678       0
..                    …               …                        …      …
564                0.2216          0.2060                  0.07115       0
565                0.1628          0.2572                  0.06637       0
566                0.1418          0.2218                  0.07820       0
567                0.2650          0.4087                  0.12400       0
568                0.0000          0.2871                  0.07039       1

[569 rows x 31 columns]
```

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   mean radius              569 non-null    float64
 1   mean texture             569 non-null    float64
 2   mean perimeter           569 non-null    float64
 3   mean area                569 non-null    float64
 4   mean smoothness          569 non-null    float64
 5   mean compactness         569 non-null    float64
 6   mean concavity           569 non-null    float64
 7   mean concave points      569 non-null    float64
 8   mean symmetry            569 non-null    float64
 9   mean fractal dimension   569 non-null    float64
 10  radius error             569 non-null    float64
 11  texture error            569 non-null    float64
 12  perimeter error          569 non-null    float64
 13  area error               569 non-null    float64
 14  smoothness error         569 non-null    float64
 15  compactness error        569 non-null    float64
 16  concavity error          569 non-null    float64
 17  concave points error     569 non-null    float64
```

```
 18  symmetry error          569 non-null    float64
 19  fractal dimension error 569 non-null    float64
 20  worst radius            569 non-null    float64
 21  worst texture           569 non-null    float64
 22  worst perimeter         569 non-null    float64
 23  worst area              569 non-null    float64
 24  worst smoothness        569 non-null    float64
 25  worst compactness       569 non-null    float64
 26  worst concavity         569 non-null    float64
 27  worst concave points    569 non-null    float64
 28  worst symmetry          569 non-null    float64
 29  worst fractal dimension 569 non-null    float64
 30  target                  569 non-null    int32
dtypes: float64(30), int32(1)
memory usage: 135.7 KB
```

[12]:
```
df.isna().sum()
```

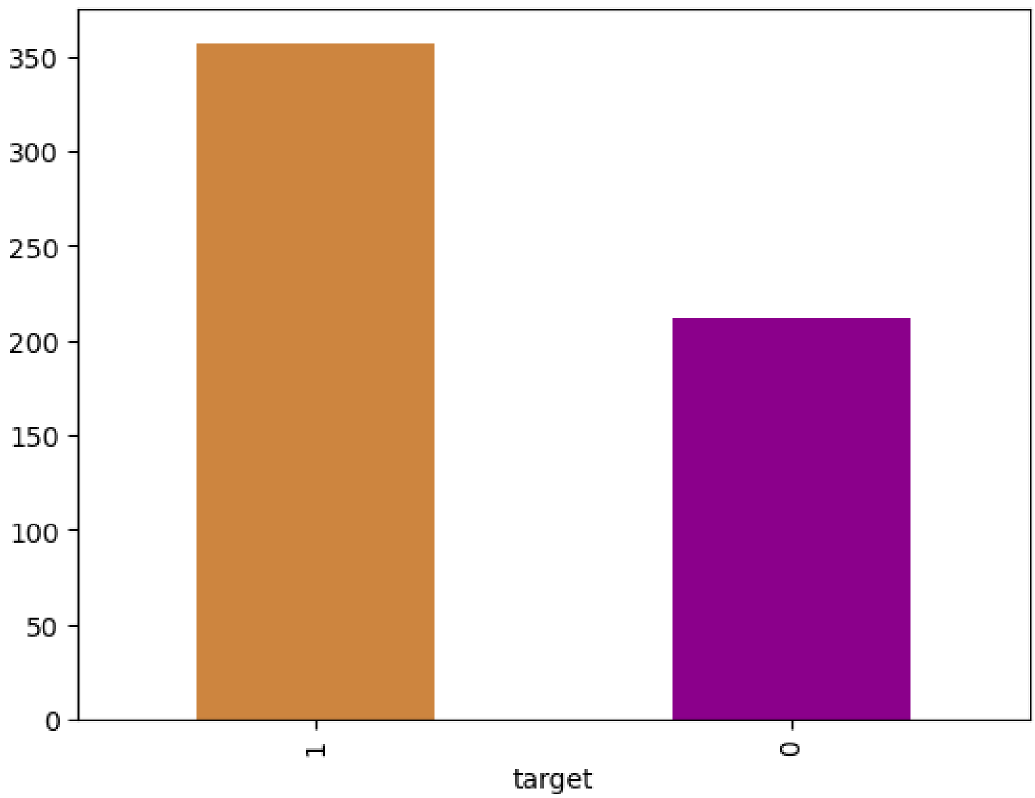
[12]:
```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
```

```
worst fractal dimension    0
target                     0
dtype: int64
```

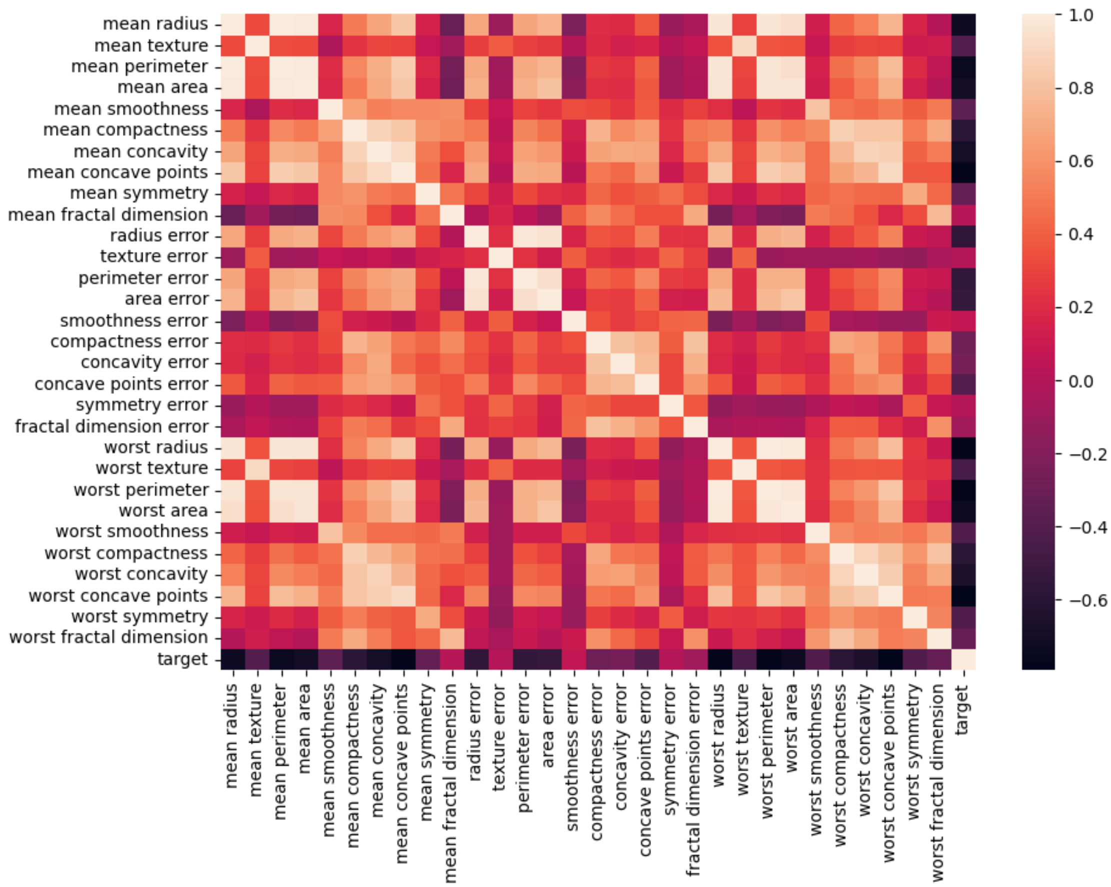
[13]:
```
df["target"].value_counts()
```

[13]:
```
target
1    357
0    212
Name: count, dtype: int64
```

[14]:
```
df["target"].value_counts().plot(kind="bar", color=["peru", "darkmagenta"]);
```



[15]:
```
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize=(10, 7))
ax = sns.heatmap(corr_matrix)
```



[16]:
```
data = load_breast_cancer()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

16 Apples and Oranges CSV

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

[5]:
```
import pandas as pd
df = pd.read_csv('apples_and_oranges.csv')
print(df.head())
```

```
   Weight  Size   Class
```

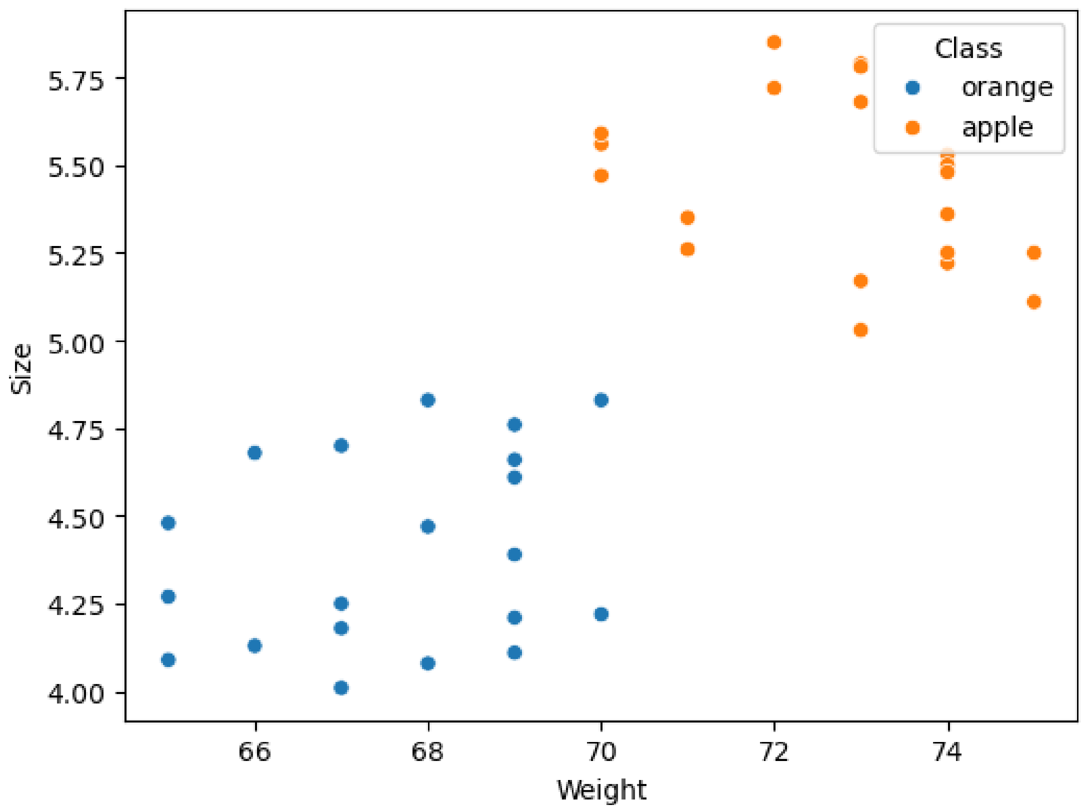
```
0       69  4.39  orange
1       69  4.21  orange
2       65  4.09  orange
3       72  5.85   apple
4       67  4.70  orange
```

[7]:
```python
data = df.copy()
print(data.head())
```

```
   Weight  Size   Class
0       69  4.39  orange
1       69  4.21  orange
2       65  4.09  orange
3       72  5.85   apple
4       67  4.70  orange
```

[10]:
```python
import seaborn as sns
sns.scatterplot(x="Weight", y="Size", hue="Class", data=data)
```

[10]: <Axes: xlabel='Weight', ylabel='Size'>



[11]:
```python
from sklearn.model_selection import train_test_split
#training_set, test_set = train_test_split(data, test_size=0.2, random_state =
  ↪1)
training_set,test_set = train_test_split(data,test_size=0.2,random_state=1)
print("train:",training_set)
print("test:",test_set)
```

```
train:     Weight  Size   Class
19      74  5.50   apple
26      67  4.01  orange
32      72  5.72   apple
17      75  5.25   apple
30      73  5.78   apple
36      69  4.76  orange
33      73  5.17   apple
28      74  5.25   apple
4       67  4.70  orange
14      74  5.22   apple
10      73  5.79   apple
35      69  4.11  orange
23      68  4.08  orange
24      67  4.25  orange
34      68  4.83  orange
20      66  4.13  orange
18      67  4.18  orange
25      71  5.35   apple
6       70  5.56   apple
13      68  4.47  orange
7       75  5.11   apple
38      70  5.59   apple
1       69  4.21  orange
16      69  4.66  orange
0       69  4.39  orange
15      65  4.48  orange
5       73  5.68   apple
11      70  5.47   apple
9       65  4.27  orange
8       74  5.36   apple
12      74  5.53   apple
37      74  5.48   apple
test:     Weight  Size   Class
2       65  4.09  orange
31      66  4.68  orange
3       72  5.85   apple
21      70  4.83  orange
27      70  4.22  orange
29      71  5.26   apple
22      69  4.61  orange
```

```
39       73  5.03    apple
```

[12]:
```
x_train = training_set.iloc[:,0:2].values  # data
y_train = training_set.iloc[:,2].values  # target
x_test = test_set.iloc[:,0:2].values  # data
y_test = test_set.iloc[:,2].values  # target
print(x_train,y_train)
print(x_test,y_test)
```

```
[[74.    5.5 ]
 [67.    4.01]
 [72.    5.72]
 [75.    5.25]
 [73.    5.78]
 [69.    4.76]
 [73.    5.17]
 [74.    5.25]
 [67.    4.7 ]
 [74.    5.22]
 [73.    5.79]
 [69.    4.11]
 [68.    4.08]
 [67.    4.25]
 [68.    4.83]
 [66.    4.13]
 [67.    4.18]
 [71.    5.35]
 [70.    5.56]
 [68.    4.47]
 [75.    5.11]
 [70.    5.59]
 [69.    4.21]
 [69.    4.66]
 [69.    4.39]
 [65.    4.48]
 [73.    5.68]
 [70.    5.47]
 [65.    4.27]
 [74.    5.36]
 [74.    5.53]
 [74.    5.48]] ['apple' 'orange' 'apple' 'apple' 'apple' 'orange' 'apple'
'apple'
 'orange' 'apple' 'apple' 'orange' 'orange' 'orange' 'orange' 'orange'
 'orange' 'apple' 'apple' 'orange' 'apple' 'apple' 'orange' 'orange'
 'orange' 'orange' 'apple' 'apple' 'orange' 'apple' 'apple' 'apple']
[[65.    4.09]
 [66.    4.68]
 [72.    5.85]
```

```
 [70.    4.83]
 [70.    4.22]
 [71.    5.26]
 [69.    4.61]
 [73.    5.03]] ['orange' 'orange' 'apple' 'orange' 'orange' 'apple' 'orange'
'apple']
```

[13]:
```
from sklearn.svm import SVC
classifier = SVC(kernel='rbf',random_state=1,C=1,gamma='auto')
classifier.fit(x_train,y_train)
```

[13]:
```
SVC(C=1, gamma='auto', random_state=1)
```

[14]:
```
y_pred = classifier.predict(x_test)
print(y_pred)
```

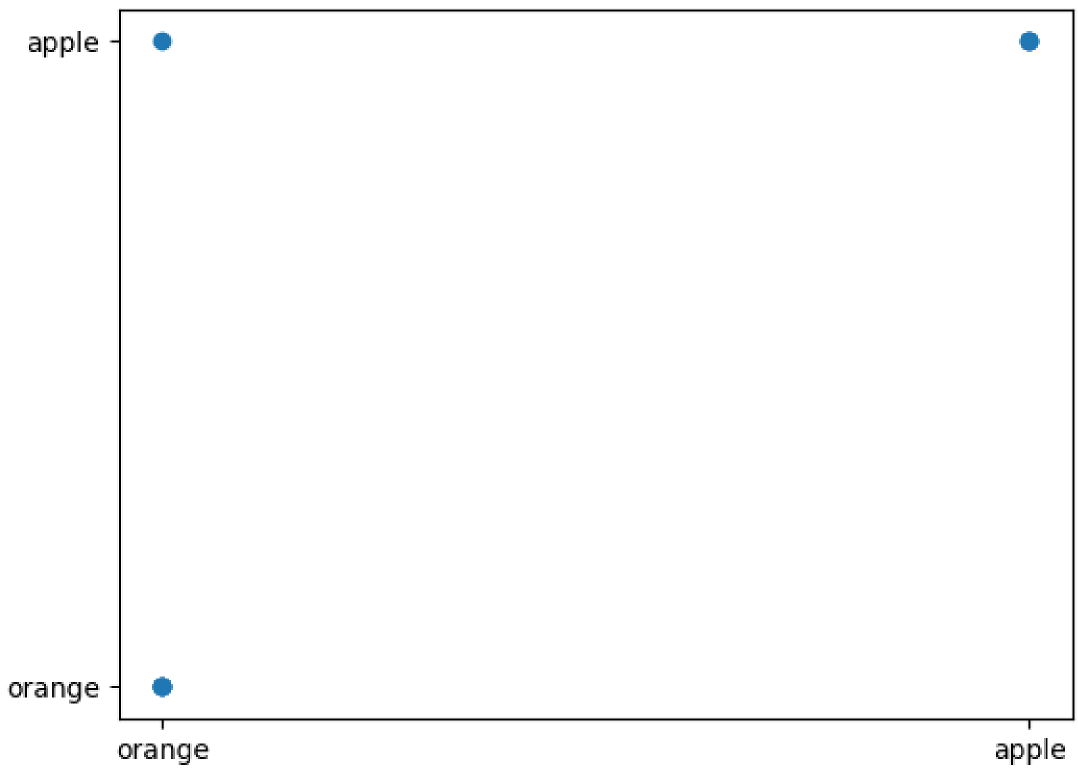
```
['orange' 'orange' 'apple' 'apple' 'orange' 'apple' 'orange' 'apple']
```

[15]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print(cm)
accuracy = float(cm.diagonal().sum())/len(y_test)
print('model accuracy is:',accuracy*100,'%')
```

```
[[3 0]
 [1 4]]
model accuracy is: 87.5 %
```

[16]:
```
import matplotlib.pyplot as plt
plt.scatter(y_test,y_pred)
```

[16]:
```
<matplotlib.collections.PathCollection at 0x1b4e92abe60>
```


apple
orange
orange
apple

[ ]: