In [66]:
```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame

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

In [67]:
```
iris = pd.read_csv("Iris.csv")
```

In [94]:
```
iris.head()
```

Out[94]:

	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

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

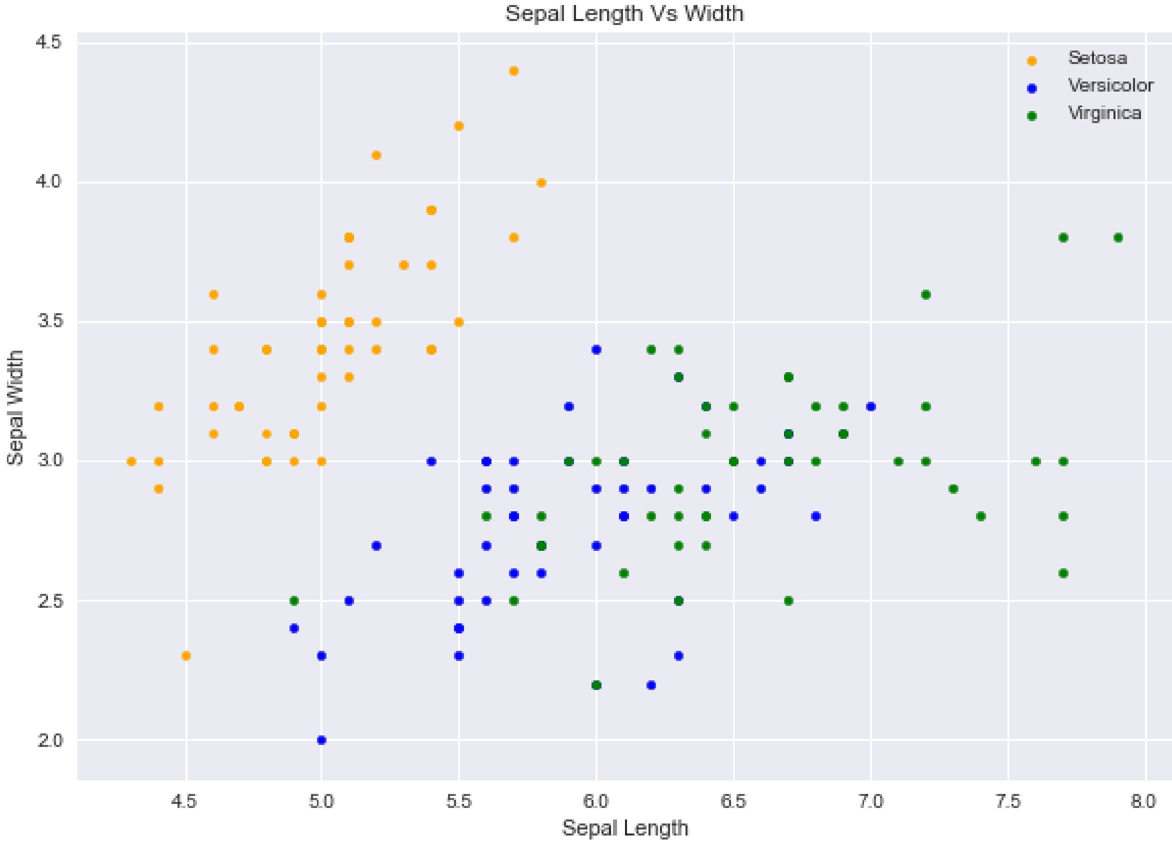
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
Id                 150 non-null int64
SepalLengthCm      150 non-null float64
SepalWidthCm       150 non-null float64
PetalLengthCm      150 non-null float64
PetalWidthCm       150 non-null float64
Species            150 non-null object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.1+ KB
```

removing unneeded column

In [6]:
```
iris.drop("Id", axis=1, inplace = True)
```

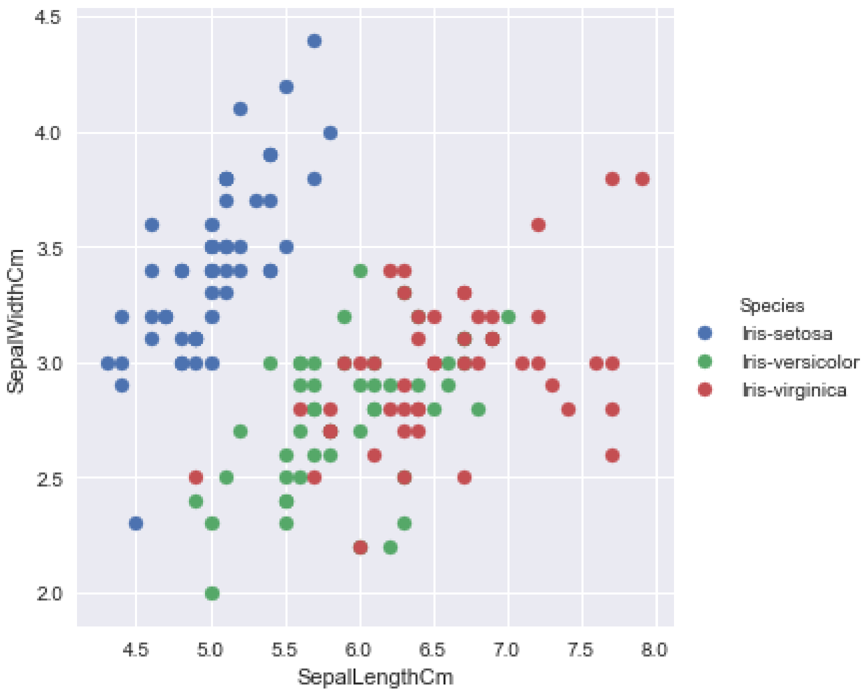
Some EDA with Iris

In [10]:
```
fig = iris[iris.Species == 'Iris-setosa'].plot(kind='scatter', x='SepalLengthCm', y
iris[iris.Species == 'Iris-versicolor'].plot(kind='scatter', x='SepalLengthCm', y=
iris[iris.Species == 'Iris-virginica'].plot(kind='scatter', x='SepalLengthCm', y='S

fig.set_xlabel('Sepal Length')
fig.set_ylabel('Sepal Width')
fig.set_title('Sepal Length Vs Width')

fig=plt.gcf()
fig.set_size_inches(10, 7)
plt.show()
```



In [12]:
```
sns.FacetGrid(iris, hue='Species', size=5)\
   .map(plt.scatter, 'SepalLengthCm', 'SepalWidthCm')\
   .add_legend()
```

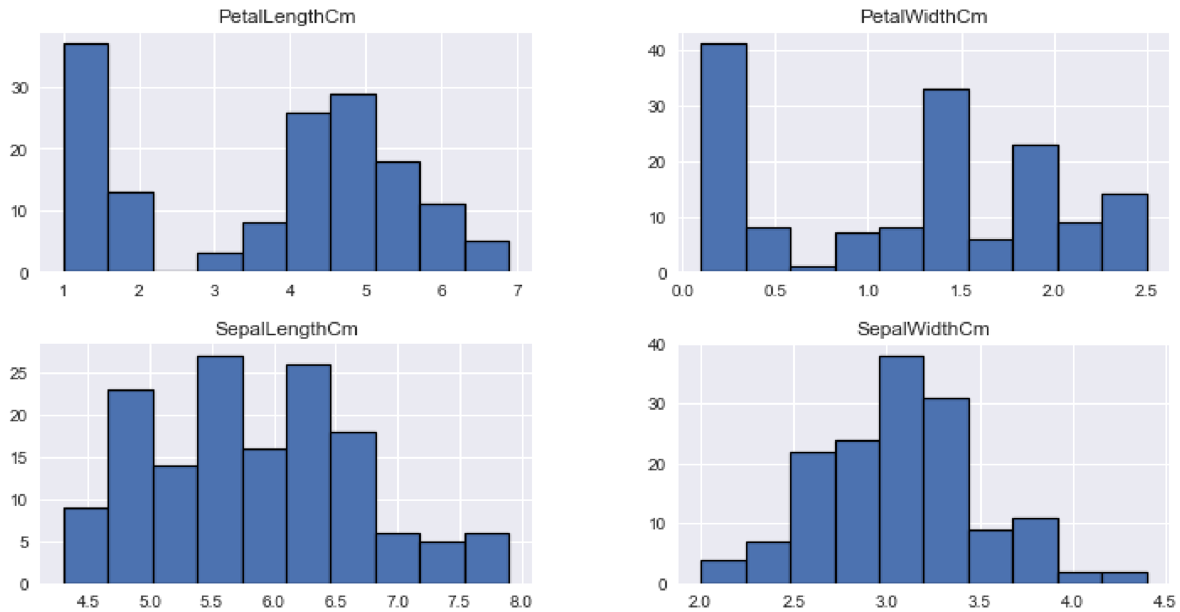
Out[12]: <seaborn.axisgrid.FacetGrid at 0xbd07748>



In [13]:
```
fig = iris[iris.Species == 'Iris-setosa'].plot(kind='scatter', x='PetalLengthCm', y
iris[iris.Species == 'Iris-versicolor'].plot(kind='scatter', x='PetalLengthCm', y=
iris[iris.Species == 'Iris-virginica'].plot(kind='scatter', x='PetalLengthCm', y='I

fig.set_xlabel('Petal Length')
fig.set_ylabel('Petal Width')
fig.set_title('Petal Length Vs Width')
```

```
fig=plt.gcf()
fig.set_size_inches(10, 7)
plt.show()
```

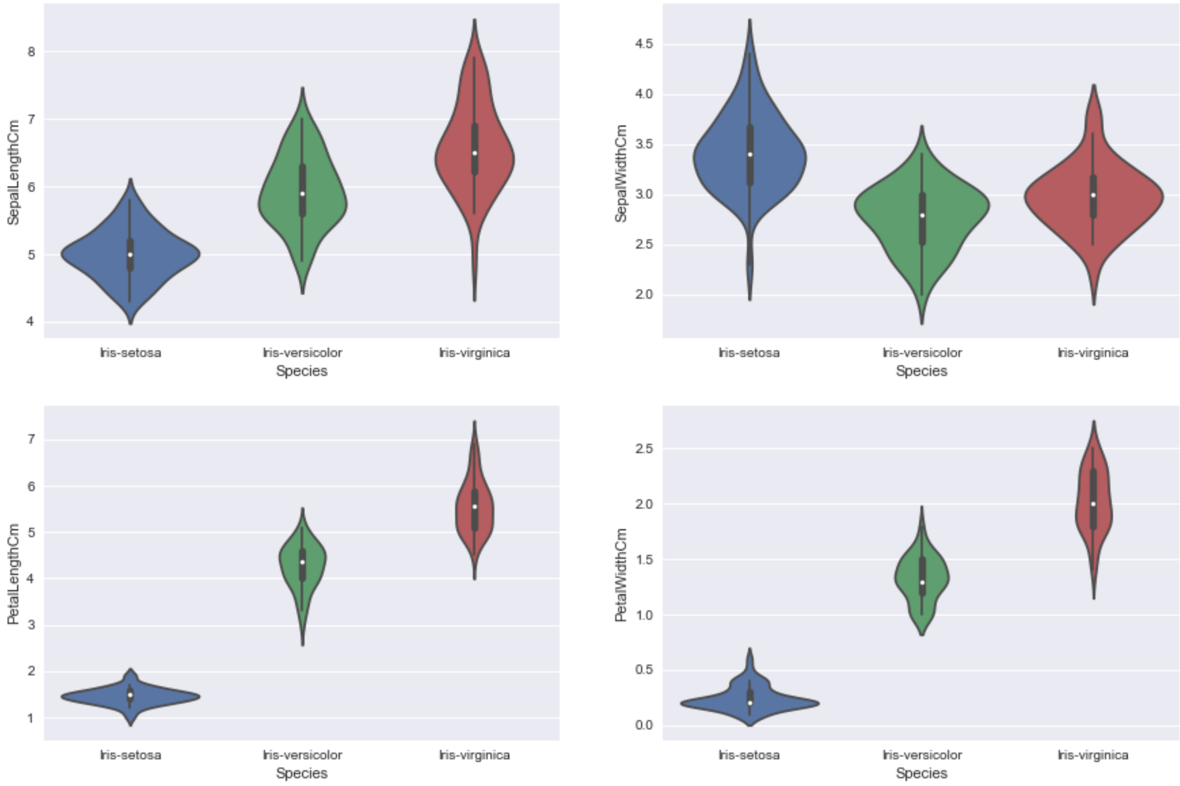
In [14]:
```
iris.hist(edgecolor='black', linewidth=1.2)
fig = plt.gcf()
fig.set_size_inches(12,6)
plt.show()
```



In [16]:
```
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)
```

Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0xd18bf60>



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.

In [18]:
```
# importing alll the necessary packages to use the various classification algorithm
from sklearn.linear_model import LogisticRegression # for Logistic Regression Algor
from sklearn.cross_validation import train_test_split # to split the dataset for tr
from sklearn.neighbors import KNeighborsClassifier # KNN classifier
from sklearn import svm # for suport vector machine algorithm
```

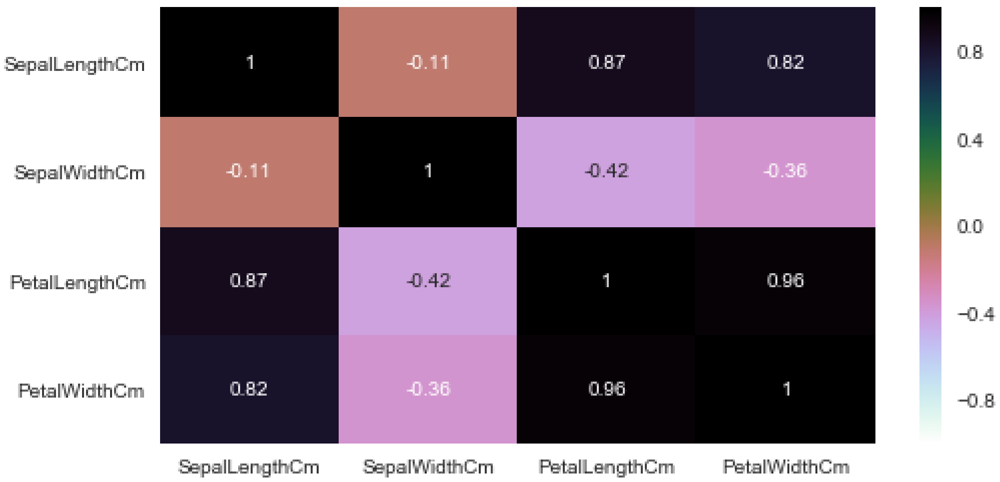
```
from sklearn import metrics # for checking the model accuracy
from sklearn.tree import DecisionTreeClassifier # for using DTA
```

In [19]:
```
iris.shape
```

Out[19]:
```
(150, 5)
```

Now, when we train any algorithm, the number of features and their correlation plays an important role. If there are features and many of the features are highly correlated, then training an algorithm with all the featues will reduce the accuracy. Thus features selection should be done carefully. This dataset has less featues but still we will see the correlation.

In [20]:
```
plt.figure(figsize=(8,4))
sns.heatmap(iris.corr(), annot=True, cmap='cubehelix_r') # draws heatmap with inpu
plt.show()
```



Observation---> The Sepal Width and Length are not correlated The Petal Width and Length are highly correlated We will use all the features for training the algorithm and check the accuracy.

Then we will use 1 Petal Feature and 1 Sepal Feature to check the accuracy of the algorithm as we are using only 2 features that are not correlated. Thus we can have a variance in the dataset which may help in better accuracy. We will check it later.

Steps To Be followed When Applying an Algorithm

Split the dataset into training and testing dataset. The testing dataset is generally smaller than training one as it will help in training the model better.

Select any algorithm based on the problem (classification or regression) whatever you feel may be good. Then pass the training dataset to the algorithm to train it. We use the .fit() method Then pass the testing data to the trained algorithm to predict the outcome. We use the .predict() method. We then check the accuracy by passing the predicted outcome and the actual output to the model.

Splitting The Data into Training And Testing Dataset

In [21]:
```
train, test = train_test_split(iris, test_size=0.3) # our main data split into tra
# the attribute test_size=0.3 splits the data into 70% and 30% ratio. train=70% and
```

```
print(train.shape)
print(test.shape)
```

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

In [92]:
```
train_X = train[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']] #
train_y = train.Species # output of the training data

test_X = test[['SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm']] # te
test_y = test.Species # output value of the test data
```

In [93]:
```
train_X.head()
```

Out[93]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
58	6.6	2.9	4.6	1.3
131	7.9	3.8	6.4	2.0
85	6.0	3.4	4.5	1.6
18	5.7	3.8	1.7	0.3
84	5.4	3.0	4.5	1.5

In [27]:
```
test_X.head()
```

Out[27]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
4	5.0	3.6	1.4	0.2
41	4.5	2.3	1.3	0.3
124	6.7	3.3	5.7	2.1
23	5.1	3.3	1.7	0.5
118	7.7	2.6	6.9	2.3

In [28]:
```
train_y.head()
```

Out[28]:
```
58     Iris-versicolor
131     Iris-virginica
85     Iris-versicolor
18         Iris-setosa
84     Iris-versicolor
Name: Species, dtype: object
```

Support Vector Machine SVM

In [39]:
```
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: ', 0.9555555555555556)
```

SVM is giving very good accuracy . We will continue to check the accuracy for different models.

Now we will follow the same steps as above for training various machine learning algorithms.

Logistic Regression

In [38]:
```
model = LogisticRegression()
model.fit(train_X, train_y)
prediction = model.predict(test_X)
print('The accuracy of Logistic Regression is: ', metrics.accuracy_score(predictio
```

```
('The accuracy of Logistic Regression is: ', 0.9555555555555556)
```

Decision Tree

In [40]:
```
model = DecisionTreeClassifier()
model.fit(train_X, train_y)
prediction = model.predict(test_X)
print('The accuracy of Decision Tree is: ', metrics.accuracy_score(prediction, tes
```

```
('The accuracy of Decision Tree is: ', 0.9333333333333335)
```

K-Nearest Neighbors

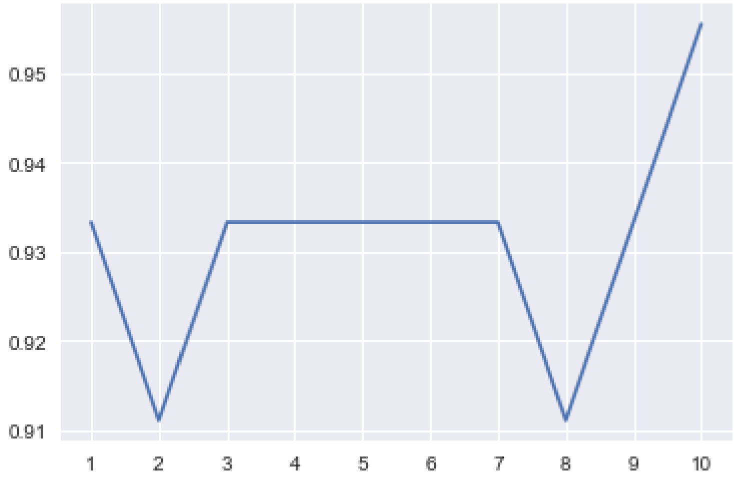
In [42]:
```
model = KNeighborsClassifier(n_neighbors=3) # this examines 3 neighbors for puttin
model.fit(train_X, train_y)
prediction = model.predict(test_X)
print('The accuracy of KNN is: ', metrics.accuracy_score(prediction, test_y))
```

```
('The accuracy of KNN is: ', 0.9333333333333335)
```

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

In [45]:
```
a_index = list(range(1,11))
a = pd.Series()
for i in list(range(1,11)):
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(train_X, train_y)
    prediction = model.predict(test_X)
    a = a.append(pd.Series(metrics.accuracy_score(prediction, test_y)))
plt.plot(a_index, a)
x = [1,2,3,4,5,6,7,8,9,10]
plt.xticks(x)
```

Out[45]:
```
([<matplotlib.axis.XTick at 0xe6fd940>,
  <matplotlib.axis.XTick at 0xe684160>,
  <matplotlib.axis.XTick at 0xe5dec88>,
  <matplotlib.axis.XTick at 0xe5ec240>,
  <matplotlib.axis.XTick at 0xe5ec898>,
  <matplotlib.axis.XTick at 0xe5ecef0>,
  <matplotlib.axis.XTick at 0xe5f8588>,
  <matplotlib.axis.XTick at 0xe5f8be0>,
  <matplotlib.axis.XTick at 0xe604278>,
  <matplotlib.axis.XTick at 0xe6048d0>],
 <a list of 10 Text xticklabel objects>)
```



Above is the graph showing the accuracy for the KNN models using different values of n.

We used all the features of iris in above models. Now we will use Petals and Sepals Seperately

Creating Petals And Sepals Training Data

In [103…
```
petal = iris[['PetalLengthCm','PetalWidthCm','Species']]
sepal = iris[['SepalLengthCm','SepalWidthCm','Species']]
```

For Iris Petal

In [ ]:
```
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
```

For Iris Sepal

In [105…
```
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
```

SVM Algorithm

In [109…
```
model=svm.SVC()
model.fit(train_x_p,train_y_p)
prediction=model.predict(test_x_p)
print('The accuracy of the SVM using Petals is:',metrics.accuracy_score(prediction

model=svm.SVC()
model.fit(train_x_s,train_y_s)
prediction=model.predict(test_x_s)
print('The accuracy of the SVM using Sepals is:',metrics.accuracy_score(prediction
```

```
('The accuracy of the SVM using Petals is:', 0.9777777777777775)
('The accuracy of the SVM using Sepals is:', 0.8000000000000004)
```

Logistic Regression

In [111…

```
model = LogisticRegression()
model.fit(train_x_p,train_y_p)
prediction=model.predict(test_x_p)
print('The accuracy of the Logistic Regression using Petals is:',metrics.accuracy_

model.fit(train_x_s,train_y_s)
prediction=model.predict(test_x_s)
print('The accuracy of the Logistic Regression using Sepals is:',metrics.accuracy_
```

```
('The accuracy of the Logistic Regression using Petals is:', 0.6888888888888888)
('The accuracy of the Logistic Regression using Sepals is:', 0.6444444444444449)
```

Decision Tree

In [112…

```
model=DecisionTreeClassifier()
model.fit(train_x_p,train_y_p)
prediction=model.predict(test_x_p)
print('The accuracy of the Decision Tree using Petals is:',metrics.accuracy_score(

model.fit(train_x_s,train_y_s)
prediction=model.predict(test_x_s)
print('The accuracy of the Decision Tree using Sepals is:',metrics.accuracy_score(
```

```
('The accuracy of the Decision Tree using Petals is:', 0.9555555555555556)
('The accuracy of the Decision Tree using Sepals is:', 0.6444444444444449)
```

K-Nearest Neighbors

In [113…

```
model=KNeighborsClassifier(n_neighbors=3)
model.fit(train_x_p,train_y_p)
prediction=model.predict(test_x_p)
print('The accuracy of the KNN using Petals is:',metrics.accuracy_score(prediction

model.fit(train_x_s,train_y_s)
prediction=model.predict(test_x_s)
print('The accuracy of the KNN using Sepals is:',metrics.accuracy_score(prediction
```

```
('The accuracy of the KNN using Petals is:', 0.9777777777777775)
('The accuracy of the KNN using Sepals is:', 0.7333333333333328)
```

Observations:

- Using Petals over Sepal for training the data gives a much better accuracy.
- This was expected as we saw in the heatmap above that the correlation between the Sepal Width and Length was very low whereas the correlation between Petal Width and Length was very high.

In [ ]:

In [4]:

```
import numpy as np
from sklearn import datasets

iris = datasets.load_iris()

X = iris.data[:, [2, 3]]
y = iris.target
```

In [5]:

```
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
```

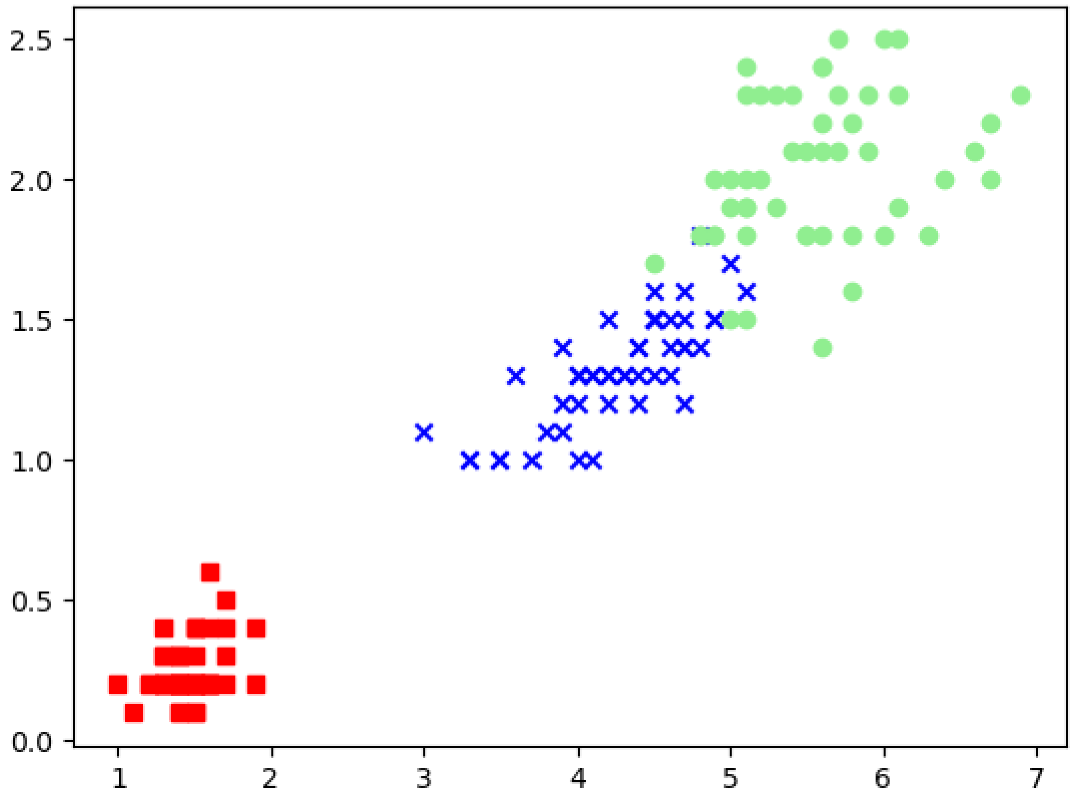
In [6]:

```
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))
```

In [7]:

```
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* 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 arr
ay with a single row if you intend to specify the same RGB or RGBA value for
all points.
*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 arr
ay with a single row if you intend to specify the same RGB or RGBA value for
all points.
*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 arr
ay with a single row if you intend to specify the same RGB or RGBA value for
all points.
```



In [8]:

```
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    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)

    # highlight test samples
    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")
```

In [11]:

```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=0, gamma=.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.s

print('The accuracy of the svm classifier on test data is {:.2f} out of 1'.format(svm.score
```

```
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
```

In [ ]:

```
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()
```

In [ ]:

```
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 is {:.2f} out of 1 on training data'.format(knn.s
print('The accuracy of the knn classifier is {:.2f} out of 1 on test data'.format(knn.score
```

In [ ]:

```
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()
```

In [15]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# import models from Scikit-learn
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

# import model evaluation tools
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import plot_roc_curve

# import data
from sklearn.datasets import load_breast_cancer
```

In [16]:

```
data = load_breast_cancer()

data
```

Out[16]:

```
{'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, 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, 1, 0, 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 (diagnosti
c) dataset\n--------------------------------------------\n\n**Data Set Chara
cteristics:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes:
30 numeric, predictive attributes and the class\n\n    :Attribute Informatio
n:\n        - radius (mean of distances from center to points on the perimet
er)\n        - texture (standard deviation of gray-scale values)\n        -
perimeter\n        - area\n        - smoothness (local variation in radius l
engths)\n        - compactness (perimeter^2 / area - 1.0)\n        - concavi
ty (severity of concave portions of the contour)\n        - concave points
(number of concave portions of the contour)\n        - symmetry\n        - f
```

```
ractal dimension ("coastline approximation" - 1)\n\n        The mean, standa
rd error, and "worst" or largest (mean of the three\n        worst/largest v
alues) of these features were computed for each image,\n        resulting in
30 features.  For instance, field 0 is Mean Radius, field\n        10 is Rad
ius SE, field 20 is Worst Radius.\n\n        - class:\n                - WDB
C-Malignant\n                - WDBC-Benign\n\n    :Summary Statistics:\n\n
===================================== ====== ======\n
Min    Max\n    ===================================== ====== ======\n    rad
ius (mean):                        6.981  28.11\n    texture (mean):
9.71   39.28\n    perimeter (mean):                     43.79  188.5\n    ar
ea (mean):                          143.5  2501.0\n    smoothness (mean):
0.053  0.163\n    compactness (mean):                   0.019  0.345\n    co
ncavity (mean):                     0.0    0.427\n    concave points (mean):
0.0    0.201\n    symmetry (mean):                      0.106  0.304\n    fr
actal dimension (mean):             0.05   0.097\n    radius (standard erro
r):              0.112  2.873\n    texture (standard error):            0.3
6   4.885\n    perimeter (standard error):          0.757  21.98\n    area
(standard error):                6.802  542.2\n    smoothness (standard erro
r):          0.002  0.031\n    compactness (standard error):         0.002
0.135\n    concavity (standard error):          0.0    0.396\n    concave p
oints (standard error):      0.0    0.053\n    symmetry (standard error):
0.008  0.079\n    fractal dimension (standard error):   0.001  0.03\n    rad
ius (worst):                       7.93   36.04\n    texture (worst):
12.02  49.54\n    perimeter (worst):                    50.41  251.2\n    ar
ea (worst):                         185.2  4254.0\n    smoothness (worst):
0.071  0.223\n    compactness (worst):                  0.027  1.058\n    co
ncavity (worst):                    0.0    1.252\n    concave points (wors
t):                0.0    0.291\n    symmetry (worst):                    0.
156  0.664\n    fractal dimension (worst):            0.055  0.208\n    ====
================================= ====== ======\n\n    :Missing Attribute Va
lues: 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\nFea
tures are computed from a digitized image of a fine needle\naspirate (FNA) o
f a breast mass.  They describe\ncharacteristics of the cell nuclei present
in the image.\n\nSeparating plane described above was obtained using\nMultis
urface 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 w
hich uses linear\nprogramming to construct a decision tree.  Relevant featur
es\nwere selected using an exhaustive search in the space of 1-4\nfeatures a
nd 1-3 separating planes.\n\nThe actual linear program used to obtain the se
parating plane\nin the 3-dimensional space is that described in:\n[K. P. Ben
nett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Tw
o Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23
-34].\n\nThis database is also available through the UW CS ftp server:\n\nft
p ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. topi
c:: References\n\n   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nucle
ar feature extraction \n     for breast tumor diagnosis. IS&T/SPIE 1993 Inte
rnational Symposium on \n     Electronic Imaging: Science and Technology, vo
lume 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 v
ia linear programming. Operations Research, 43(4), pages 570-577, \n     Jul
y-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machin
e learning techniques\n     to diagnose breast cancer from fine-needle aspir
ates. Cancer Letters 77 (1994) \n     163-171.',
 'feature_names': array(['mean radius', 'mean texture', 'mean perimeter', 'm
ean 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'}
```

In [17]:

```
data.keys()
```

Out[17]:

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_name
s', 'filename', 'data_module'])
```

In [18]:

```
data.DESCR
```

Out[18]:

```
'.. _breast_cancer_dataset:\n\nBreast cancer wisconsin (diagnostic) dataset
\n--------------------------------------------\n\n**Data Set Characteristic
s:**\n\n    :Number of Instances: 569\n\n    :Number of Attributes: 30 numer
ic, 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 con
cave portions of the contour)\n        - concave points (number of concave p
ortions of the contour)\n        - symmetry\n        - fractal dimension ("c
oastline approximation" - 1)\n\n        The mean, standard error, and "wors
t" or largest (mean of the three\n        worst/largest values) of these fea
tures 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 i
s Worst Radius.\n\n        - class:\n                - WDBC-Malignant\n
- WDBC-Benign\n\n    :Summary Statistics:\n\n    ===========================
========== ====== ======\n                                           Min
Max\n    ===================================== ====== ======\n    radius (me
an):                         6.981  28.11\n    texture (mean):
9.71   39.28\n    perimeter (mean):                     43.79  188.5\n    ar
ea (mean):                          143.5  2501.0\n    smoothness (mean):
0.053  0.163\n    compactness (mean):                   0.019  0.345\n    co
ncavity (mean):                     0.0    0.427\n    concave points (mean):
0.0    0.201\n    symmetry (mean):                      0.106  0.304\n    fr
actal dimension (mean):             0.05   0.097\n    radius (standard erro
r):             0.112  2.873\n    texture (standard error):             0.3
6   4.885\n    perimeter (standard error):           0.757  21.98\n    area
(standard error):                 6.802  542.2\n    smoothness (standard erro
r):         0.002  0.031\n    compactness (standard error):         0.002
0.135\n    concavity (standard error):           0.0    0.396\n    concave p
oints (standard error):       0.0    0.053\n    symmetry (standard error):
0.008  0.079\n    fractal dimension (standard error):   0.001  0.03\n    rad
ius (worst):                        7.93   36.04\n    texture (worst):
12.02  49.54\n    perimeter (worst):                    50.41  251.2\n    ar
ea (worst):                         185.2  4254.0\n    smoothness (worst):
0.071  0.223\n    compactness (worst):                  0.027  1.058\n    co
ncavity (worst):                    0.0    1.252\n    concave points (wors
t):              0.0    0.291\n    symmetry (worst):                     0.
156  0.664\n    fractal dimension (worst):            0.055  0.208\n    ====
================================= ====== ======\n\n    :Missing Attribute Va
lues: 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\nFea
tures are computed from a digitized image of a fine needle\naspirate (FNA) o
f a breast mass.  They describe\ncharacteristics of the cell nuclei present
in the image.\n\nSeparating plane described above was obtained using\nMultis
urface 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 w
hich uses linear\nprogramming to construct a decision tree.  Relevant featur
es\nwere selected using an exhaustive search in the space of 1-4\nfeatures a
nd 1-3 separating planes.\n\nThe actual linear program used to obtain the se
parating plane\nin the 3-dimensional space is that described in:\n[K. P. Ben
```

```
nett and O. L. Mangasarian: "Robust Linear\nProgramming Discrimination of Tw
o Linearly Inseparable Sets",\nOptimization Methods and Software 1, 1992, 23
-34].\n\nThis database is also available through the UW CS ftp server:\n\nft
p ftp.cs.wisc.edu\ncd math-prog/cpo-dataset/machine-learn/WDBC/\n\n.. topi
c:: References\n\n   - W.N. Street, W.H. Wolberg and O.L. Mangasarian. Nucle
ar feature extraction \n     for breast tumor diagnosis. IS&T/SPIE 1993 Inte
rnational Symposium on \n     Electronic Imaging: Science and Technology, vo
lume 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 v
ia linear programming. Operations Research, 43(4), pages 570-577, \n     Jul
y-August 1995.\n   - W.H. Wolberg, W.N. Street, and O.L. Mangasarian. Machin
e learning techniques\n     to diagnose breast cancer from fine-needle aspir
ates. Cancer Letters 77 (1994) \n     163-171.'
```

In [19]:

```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

# Show the dataframe
df
```

Out[19]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	me symme
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1
...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1

569 rows × 31 columns

In [20]:

```
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
```

In [21]:

```
df.isna().sum()
```

Out[21]:

```
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
```

In [22]:

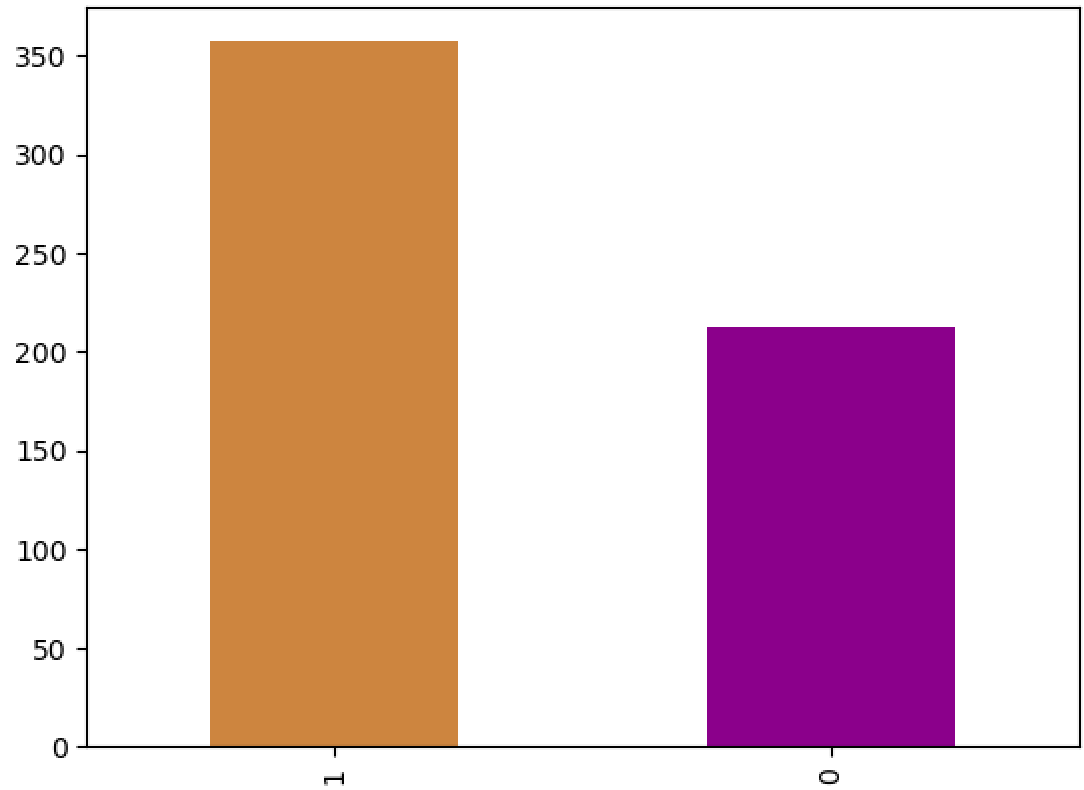
```
df["target"].value_counts()
```

Out[22]:

```
1    357
0    212
Name: target, dtype: int64
```

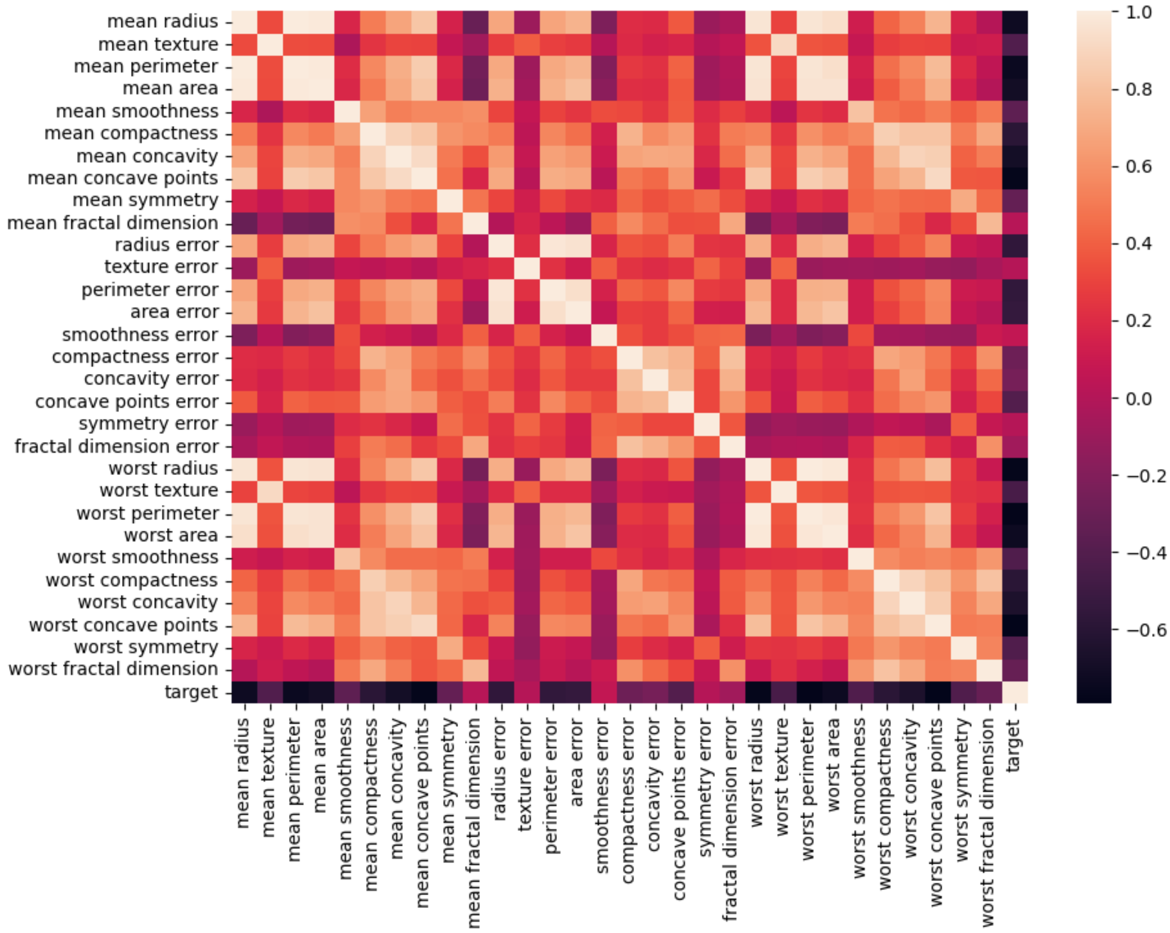
In [23]:

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



In [24]:

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



In [25]:

```
X = data.data
y = data.target

# Split the data using Scikit-Learn‘s train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.2,
                                                    random_state=0)
```

In [ ]:

In [2]:
```
# import required libraries

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
```

In [15]:
```
# read data from csv file
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
```

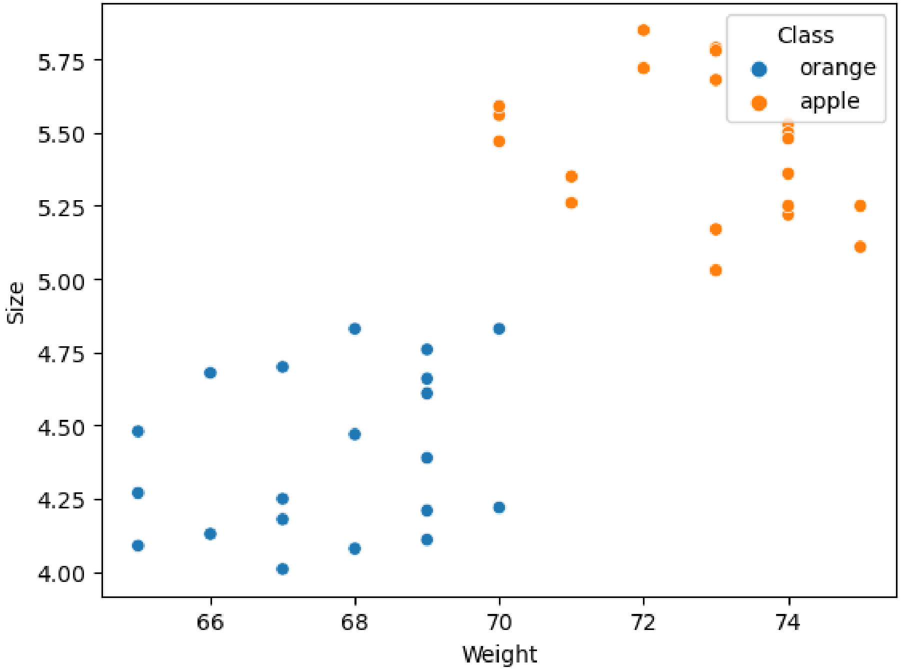
In [16]:
```
data=pd.DataFrame(data)
data.head()
```

Out[16]:

	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

In [14]:
```
import seaborn as sns
sns.scatterplot(x="Weight",
                    y="Size",
                    hue="Class",
                    data=data)
```

Out[14]: <AxesSubplot:xlabel='Weight', ylabel='Size'>



In [5]:
```python
# splitting data into training and test set
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
```

In [6]:
```python
# prepare data for applying it to svm
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']
```

In [7]:
```python
# fitting the data (train a model)
classifier = SVC(kernel='rbf',random_state=1,C=1,gamma='auto')
classifier.fit(x_train,y_train)
```

Out[7]:
```
SVC(C=1, gamma='auto', random_state=1)
```

In [8]:
```python
# perform prediction on x_test data
y_pred = classifier.predict(x_test)
#test_set['prediction']=y_pred
print(y_pred)
```

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

In [9]:
```python
# creating confusion matrix and accuracy calculation
cm = confusion_matrix(y_test,y_pred)
print(cm)
accuracy = float(cm.diagonal().sum())/len(y_test)
print('model accuracy is:',accuracy*100,'%')


#x1_test = [[73,6]] # for new data testing
```

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

In [10]:
```python
import matplotlib.pyplot as plt
plt.scatter(y_test,y_pred)
```

Out[10]:
```
<matplotlib.collections.PathCollection at 0x1c51558dc70>
```