Machine Learning Models IRIS Dataset

April 25, 2023

1 Machine Learning Models IRIS Dataset

We will build Logistic Regression Classifier, KNN, Random Forest Classifier, and SVM Classifier models in this lab.

The following is the common imports for all models.

[39]:
```
import warnings
# Filter out warnings
warnings.filterwarnings("ignore")
```

[40]:
```
import numpy as np
import pandas as pd
import pylab
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
import re

from time import time
from datetime import datetime
from fancyimpute import IterativeImputer, KNN
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import scale, label_binarize
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from keras.models import Sequential, load_model, model_from_json
from keras.layers import Dense, Dropout, Flatten
from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score
from sklearn import linear_model, datasets
```

```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
from sklearn.model_selection import train_test_split
import plotly
import plotly.offline as py

plotly.offline.init_notebook_mode()

%matplotlib inline
```

We will use the Iris data set that is already available in sklearn.

Please see the "Exercises" directory for a very detailed Exploratory Data Analysis on the Iris set.

[2]:
```
iris = datasets.load_iris()
print("Iris data set description:", iris["DESCR"])
```

```
Iris data set description: .. _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 …
```

[3]:
```
iris.feature_names
```

[3]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

[4]:
```
iris.target_names
```

[4]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

[5]:
```
print(iris.data.shape)
```

```
(150, 4)
```

[6]:
```
iris.target
```

[6]:
```
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, 2, 2])
```

[7]:
```
X = iris.data
Y = labels = iris.target

feature_names = iris.feature_names
Y_names = iris.target_names

n_labels = len(Y_names)

n_samples, n_features = X.shape

print("n_labels=%d \t n_samples=%d \t n_features=%d" % (n_labels, n_samples,␣
 ↪n_features))
```

```
n_labels=3       n_samples=150    n_features=4
```

1.1 Exploratory Data Analysis : Iris Dataset

[8]:
```
# Load the Iris dataset
iris = datasets.load_iris()

# Convert the dataset to a Pandas dataframe
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target']  = iris.target

# Print the first few rows of the dataframe
df.head()
```

[8]:

	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

[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
```

Data Insights:

- All columns are not having any Null Entries
- Four columns are numerical type
- Only Single column categorical type which is target columns

Statistical Insight

[10]:
```
# Print some basic statistics about the dataset
df.describe()
```

[10]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

Checking the data set whether is balance or not

[11]:
```
# Print the number of instances of each target class
print(df['target'].value_counts())
```

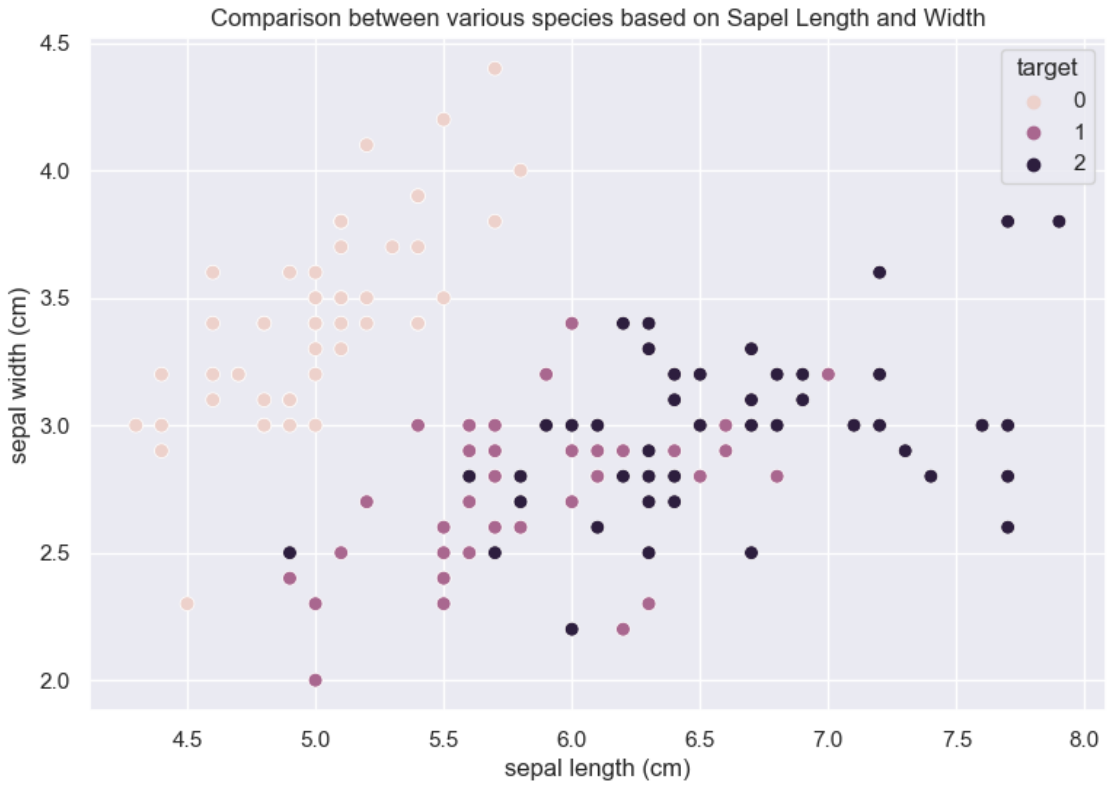
```
0    50
1    50
2    50
Name: target, dtype: int64
```

1.1.1 Data Visualization

Species = 0: 'setosa', 1: 'versicolor', 2: 'virginica'

[12]:
```
plt.figure(figsize=(9,6))
plt.title('Comparison between various species based on Sapel Length and Width')
sns.scatterplot(x = 'sepal length (cm)', y = 'sepal width (cm)', data = df, hue␣
↪='target',s=50)
```

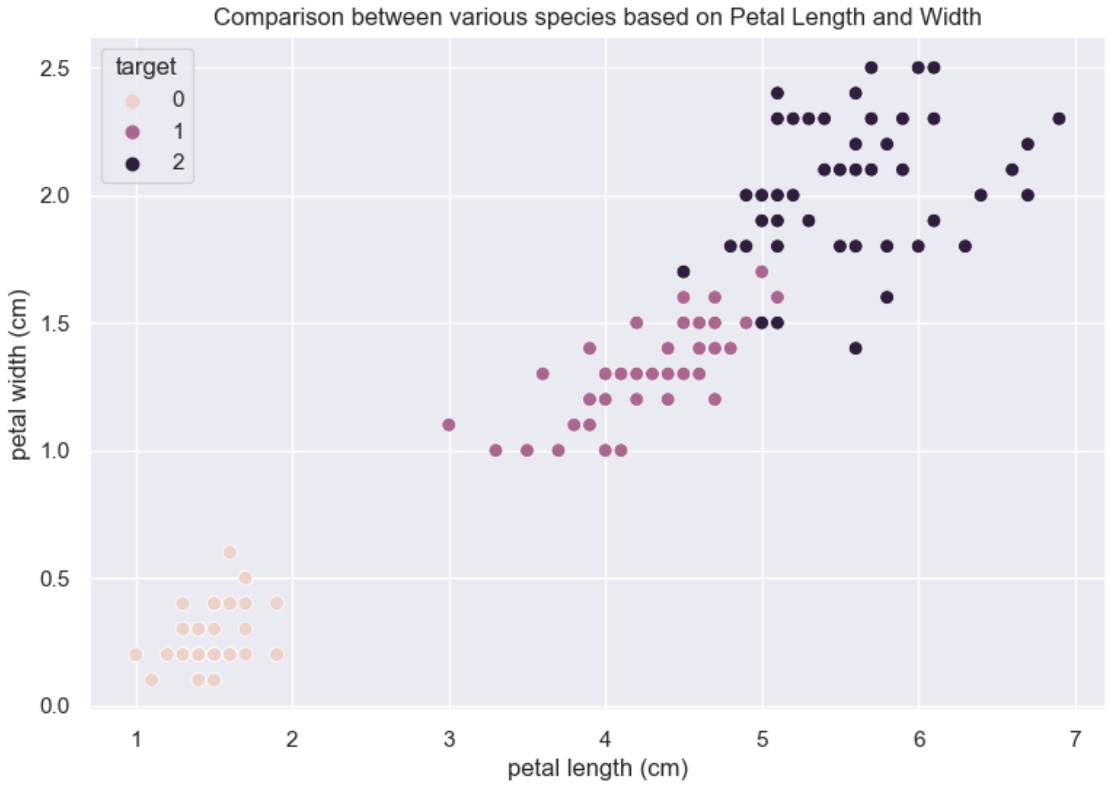
[12]:
```
<Axes: title={'center': 'Comparison between various species based on Sapel
Length and Width'}, xlabel='sepal length (cm)', ylabel='sepal width (cm)'>
```



[13]:
```
plt.figure(figsize=(9,6))
plt.title('Comparison between various species based on Petal Length and Width')
```

```
sns.scatterplot(x='petal length (cm)', y='petal width (cm)', data = df, hue␣
↪='target',s=50)
```

[13]:
```
<Axes: title={'center': 'Comparison between various species based on Petal
Length and Width'}, xlabel='petal length (cm)', ylabel='petal width (cm)'>
```



The Iris flower dataset includes three species of flowers: **Setosa, Versicolor, and Virginica**. Each species has distinctive characteristics in terms of sepal length and width.

Setosa has a smaller sepal length compared to the other two species, but its sepal width is relatively higher. In other words, the sepals of Setosa flowers are wider in proportion to their length.

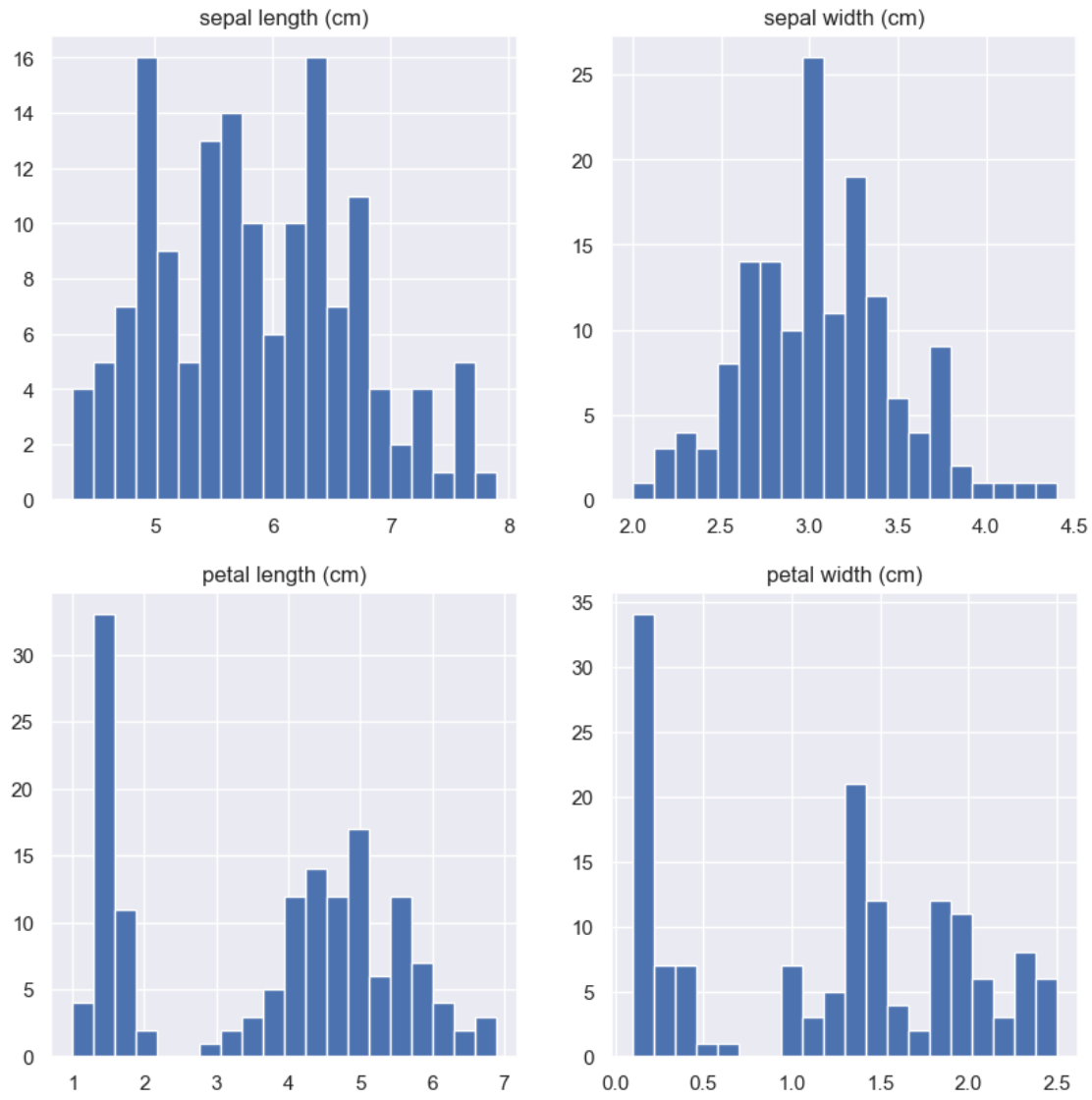
Versicolor lies somewhere in the middle in terms of both sepal length and width. Its sepals are not as long as those of Virginica, but not as short as those of Setosa. Similarly, its sepals are not as wide as those of Setosa, but not as narrow as those of Virginica.

Virginica has larger sepal lengths than the other two species, but its sepal widths are relatively smaller. In contrast to Setosa, Virginica flowers have narrower sepals compared to their length.

These differences in sepal length and width are important characteristics that can be used to distinguish between the three Iris species.

[14]:
```
# Plot the distribution of each feature
import matplotlib.pyplot as plt
```

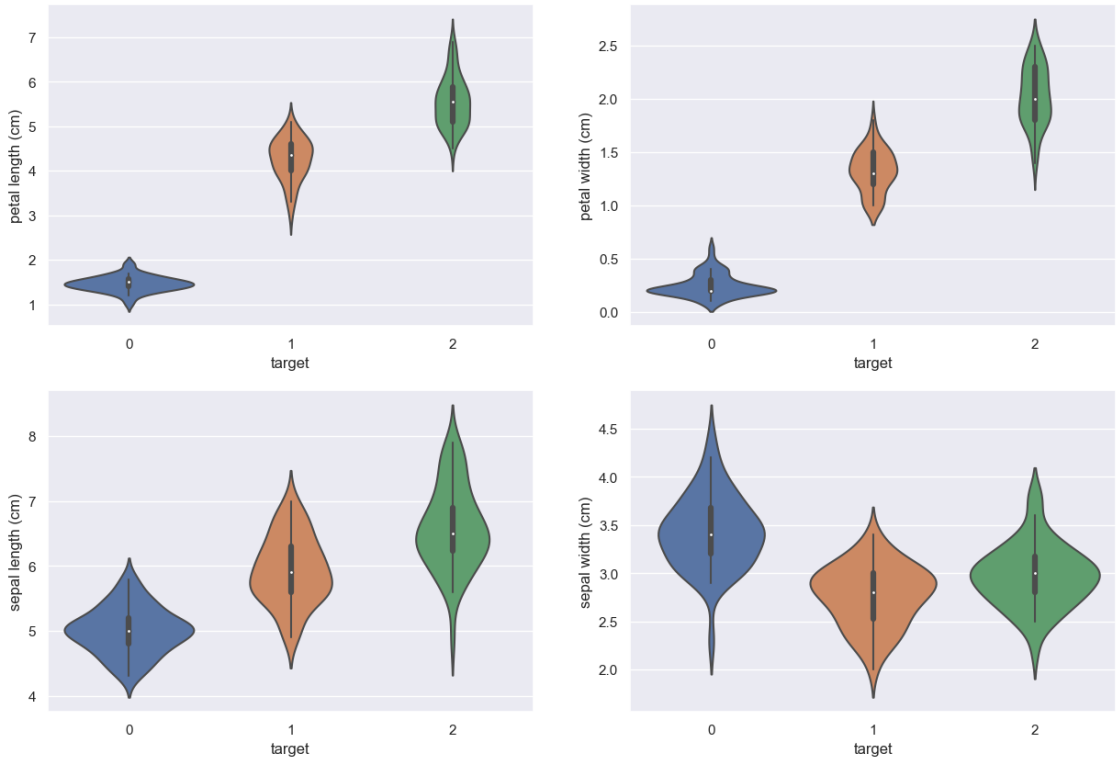
```
fig, axes = plt.subplots(2, 2, figsize=(10, 10))
for i, ax in enumerate(axes.flat):
    feature_name = iris.feature_names[i]
    ax.hist(df[feature_name], bins=20)
    ax.set(title=feature_name)
plt.show()
```



[15]:
```
## {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
plt.figure(figsize=(15,10))

plt.subplot(2,2,1)
sns.violinplot(x='target',y='petal length (cm)',data=df)
```

```
plt.subplot(2,2,2)
sns.violinplot(x='target',y='petal width (cm)',data=df)

plt.subplot(2,2,3)
sns.violinplot(x='target',y='sepal length (cm)',data=df)

plt.subplot(2,2,4)
sns.violinplot(x='target',y='sepal width (cm)',data=df)
```

[15]: <Axes: xlabel='target', ylabel='sepal width (cm)'>

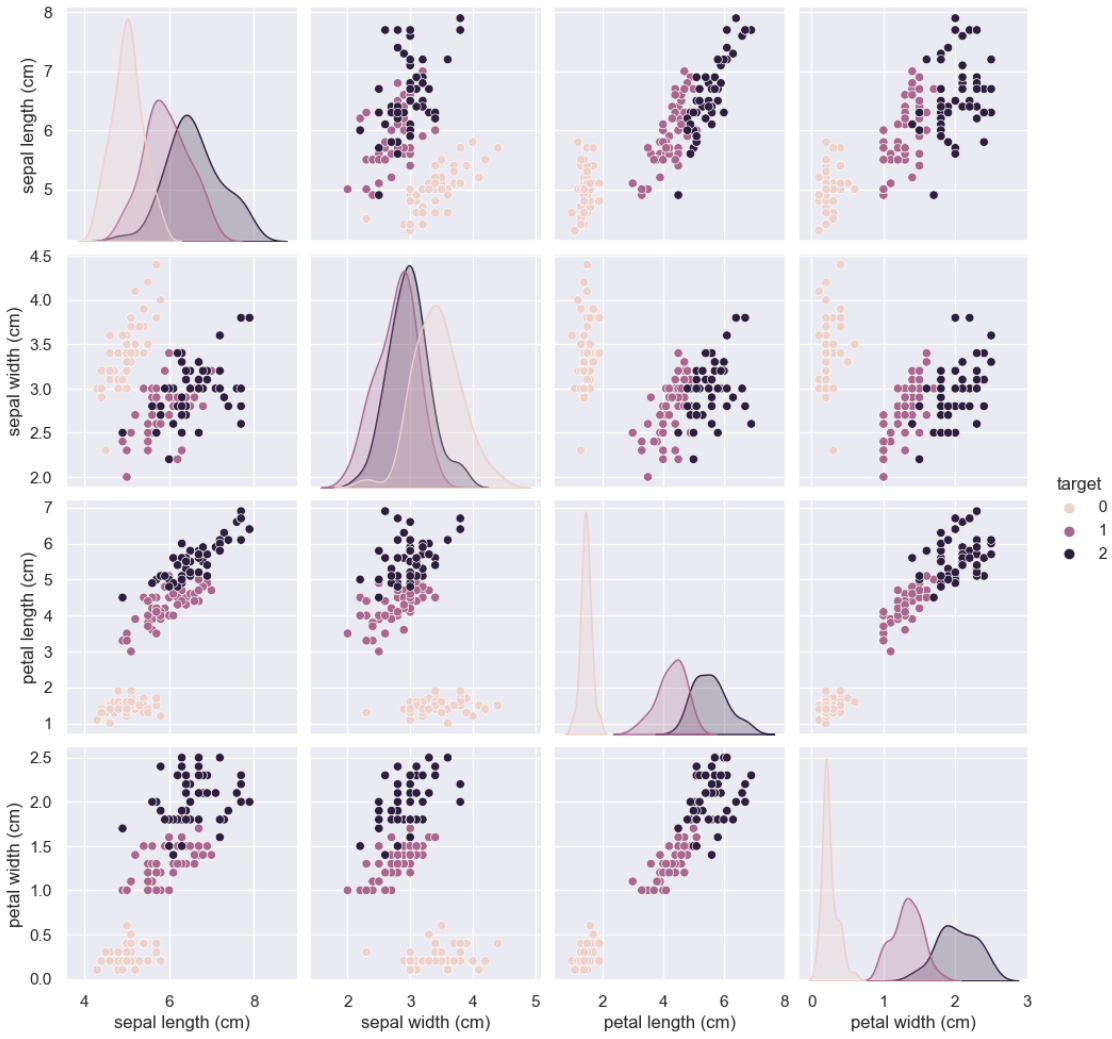


The violin plot shows density of the length and width in the species. The thinner part denotes that there is less density whereas the fatter part conveys higher density.

- Setosa is having less distribution and density in case of petal length & width
- Versicolor is distributed in a average manner and average features in case of petal length & width
- Virginica is highly distributed with large no .of values and features in case of sepal length & width
- High density values are depicting the mean/median values, for example: Iris Setosa has highest density at 5.0 cm (sepal length feature) which is also the median value(5.0) as per

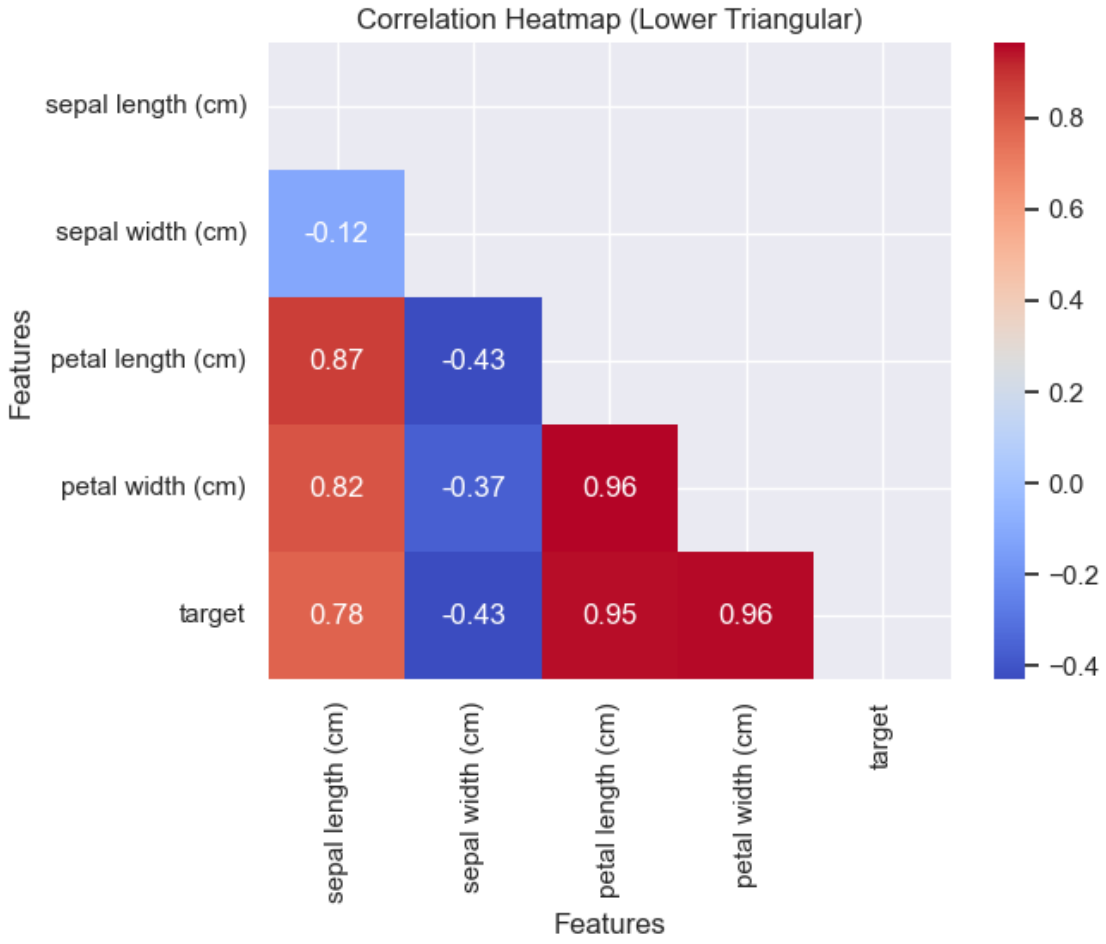
the table

[16]:
```
# Plot the pairwise scatterplot of the features
sns.pairplot(df, hue='target') ## {0: 'setosa', 1: 'versicolor', 2: 'virginica'}
plt.show()
```



- High co relation between petal length and width columns.
- Setosa has both low petal length and width
- Versicolor has both average petal length and width
- Virginica has both high petal length and width.
- Sepal width for setosa is high and length is low.
- Versicolor have average values for for sepal dimensions.
- Virginica has small width but large sepal length

[17]:
```
# Compute the correlation matrix
corr = df.corr()

# Create a mask for the upper triangular part of the matrix
mask = np.triu(np.ones_like(corr, dtype=bool))

# Create the heatmap with the mask
sns.heatmap(corr, cmap='coolwarm', annot=True, mask=mask)

# Set the title and labels
plt.title('Correlation Heatmap (Lower Triangular)')
plt.xlabel('Features')
plt.ylabel('Features')

plt.show()
```



- High co relation between petal length and width columns
- Sepal Length and Sepal Width features are slightly correlated with each other

1.2 Modeling

We will build many models, and we will need to evaluate them.

First we will use only the first 2 features:

[18]:
```
X = df.drop(["target"], axis=1)
Y = df["target"]
```

[19]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.8,␣
↪random_state=0)
```

We can use the StandardScaler to normalize the training and test sets, and train a logistic regression model on the normalized data.

[20]:
```
# Normalize the data using the StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

1.2.1 Logistic Regressions

[60]:
```
logreg = LogisticRegression(C = 1e5, random_state=42)
logreg.fit(X_train, Y_train)
```

[60]:
```
LogisticRegression(C=100000.0, random_state=42)
```

Now let's predict the test set using this model:

[61]:
```
Y_train_pred_logreg = logreg.predict(X_train)
```

Let's check the performance metrics of our train model:

[62]:
```
accuracy_logreg_train = accuracy_score(Y_train, Y_train_pred_logreg)
print("Accuracy of Training: {0:.2f}".format(accuracy_logreg_train))
```

```
Accuracy of Training: 0.98
```

Let's check the performance metrics of our test model:

[63]:
```
Y_test_pred_logreg = logreg.predict(X_test)
```

[64]:
```
accuracy_logreg_test = accuracy_score(Y_test_pred_logreg, Y_test)
print("Accuracy of Testing data: {0:.2f}".format(accuracy_logreg_test))
```

```
Accuracy of Testing data: 1.00
```

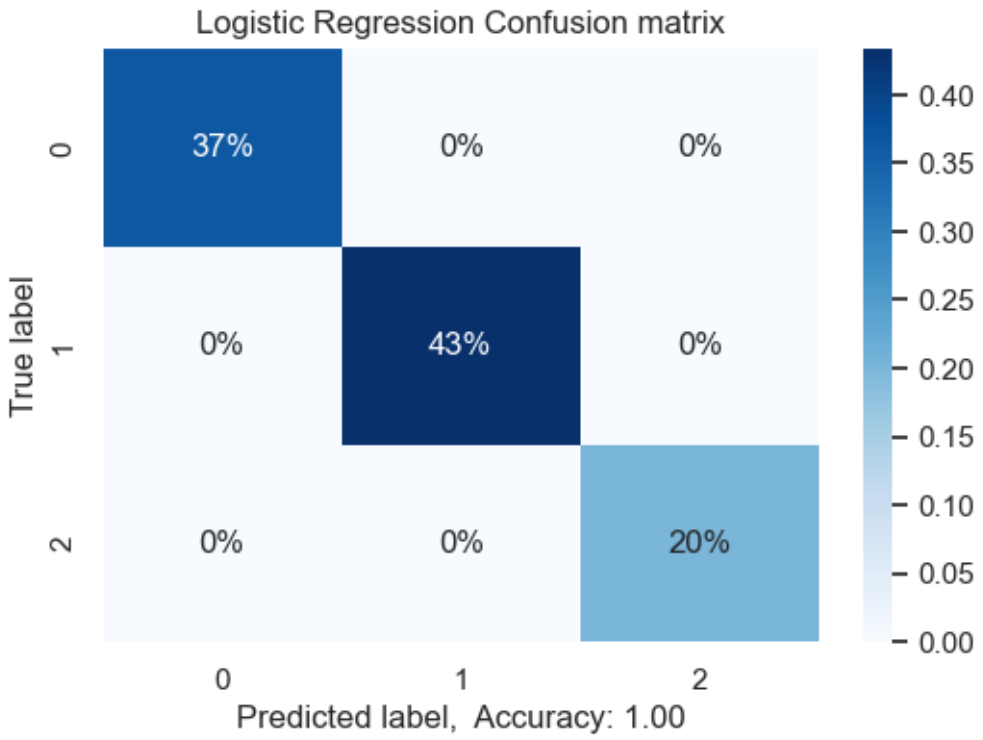
[65]:
```
# Print the classification report for the training set
print("Training Set Classification Report:")
print(classification_report(Y_train, Y_train_pred_logreg, target_names=iris.
↪target_names))
```

```
# Print the classification report for the testing set
print("Testing Set Classification Report:")
print(classification_report(Y_test, Y_test_pred_logreg, target_names=iris.
 ↪target_names))
```

```
Training Set Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        39
  versicolor       0.97      0.97      0.97        37
   virginica       0.98      0.98      0.98        44

    accuracy                           0.98       120
   macro avg       0.98      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120

Testing Set Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

[79]:
```
fig, ax = plt.subplots(figsize=(6, 4))
print("Logistic Regression Confusion Matrix:")
cm_logreg = confusion_matrix(Y_test, Y_test_pred_logreg)
print(cm)
sns.heatmap(cm_logreg/np.sum(cm_logreg), annot=True, fmt='.0%', cmap='Blues')
plt.title('Logistic Regression Confusion matrix')
plt.ylabel('True label')
plt.xlabel("Predicted label,  Accuracy: {0:.2f}".format(accuracy_logreg_test))
plt.show()
```

```
Logistic Regression Confusion Matrix:
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```



Everything looks good!

1.2.2 KNNeighborsClassifier

[67]:
```
# Create a KNN classifier
knn = KNeighborsClassifier(n_neighbors = 5)

# Fit a KNN classifier
knn.fit(X_train, Y_train)

#check the performance metrics of our train model
Y_train_pred_knn = knn.predict(X_train)
accuracy_knn_train = accuracy_score(Y_train, Y_train_pred_knn)
print("Accuracy of Training data: {0:.2f}".format(accuracy_knn_train))

#check the performance metrics of our test model
Y_test_pred_knn = knn.predict(X_test)
accuracy_knn_test = accuracy_score(Y_test_pred_knn, Y_test)
print("Accuracy of Testing data: {0:.2f}".format(accuracy_knn_test))
```

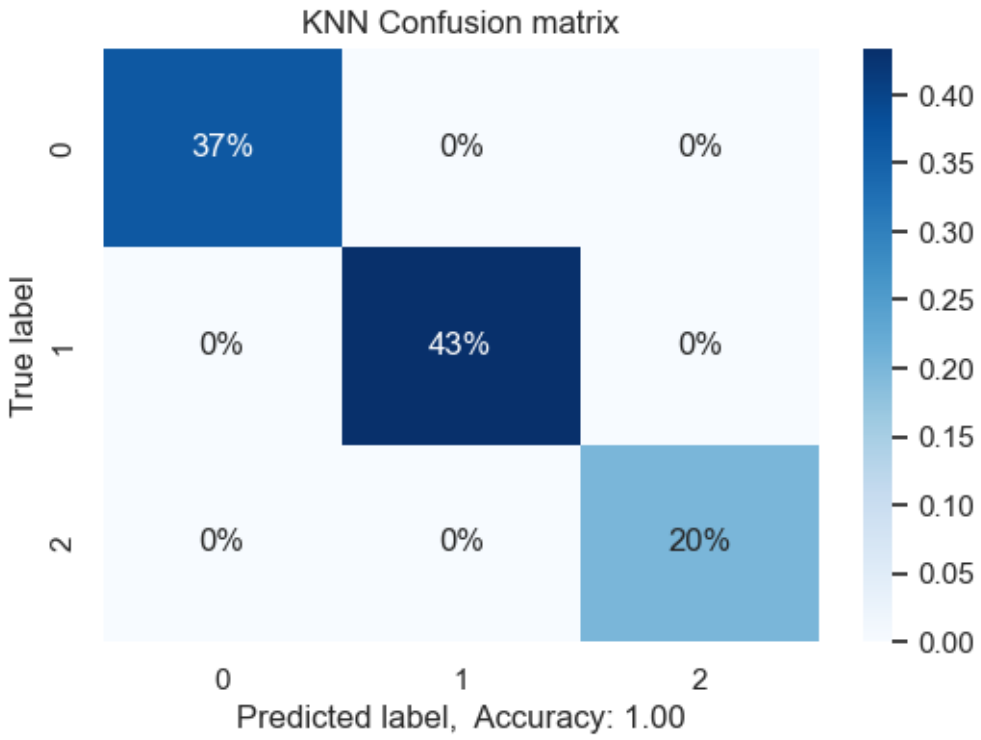
```
Accuracy of Training data: 0.96
Accuracy of Testing data: 1.00
```

[68]:
```
# Print the classification report for the training set
print("Training Set Classification Report:")
print(classification_report(Y_train, Y_train_pred_knn, target_names=iris.
 ↪target_names))

# Print the classification report for the testing set
print("Testing Set Classification Report:")
print(classification_report(Y_test, Y_test_pred_knn, target_names=iris.
 ↪target_names))
```

```
Training Set Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        39
  versicolor       0.94      0.92      0.93        37
   virginica       0.93      0.95      0.94        44

    accuracy                           0.96       120
   macro avg       0.96      0.96      0.96       120
weighted avg       0.96      0.96      0.96       120

Testing Set Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

[80]:
```
fig, ax = plt.subplots(figsize=(6, 4))
print("KNN Confusion Matrix:")
cm_knn = confusion_matrix(Y_test, Y_test_pred_knn)
print(cm)
sns.heatmap(cm_knn/np.sum(cm_knn), annot=True, fmt='.0%', cmap='Blues')
plt.title('KNN Confusion matrix')
plt.ylabel('True label')
plt.xlabel("Predicted label,  Accuracy: {0:.2f}".format(accuracy_knn_test))
plt.show()
```

```
KNN Confusion Matrix:
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```



1.2.3 Random Forrest Classifier

[70]:
```
# Create a Random Forrest Classifier
rf = RandomForestClassifier(n_estimators = 200)

# Fit a Random Forrest Classifier
rf.fit(X_train, Y_train)

#check the performance metrics of our train model
Y_train_pred_rf = rf.predict(X_train)
accuracy_rf_train = accuracy_score(Y_train, Y_train_pred_rf)
print("Accuracy of Training data: {0:.2f}".format(accuracy_rf_train))

#check the performance metrics of our test model
Y_test_pred_rf = rf.predict(X_test)
accuracy_rf_test = accuracy_score(Y_test_pred_rf, Y_test)
print("Accuracy of Testing data: {0:.2f}".format(accuracy_rf_test))
```

```
Accuracy of Training data: 1.00
Accuracy of Testing data: 1.00
```

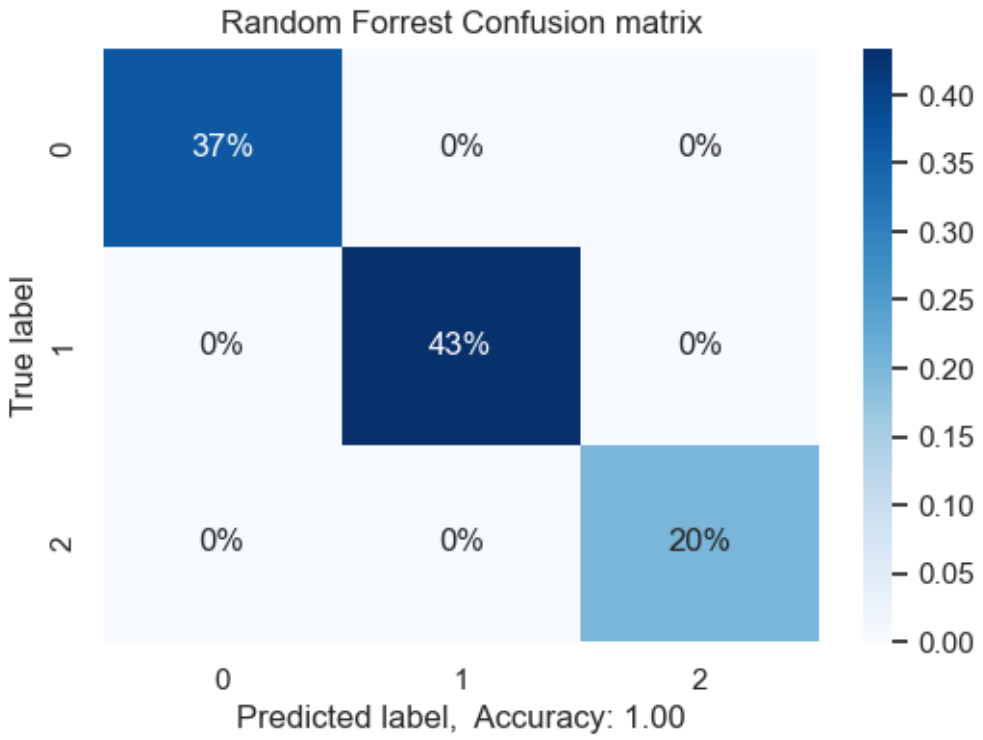
[71]:
```
# Print the classification report for the training set
print("Training Set Classification Report:")
```

```
print(classification_report(Y_train, Y_train_pred_rf, target_names=iris.
 ↪target_names))

# Print the classification report for the testing set
print("Testing Set Classification Report:")
print(classification_report(Y_test, Y_test_pred_rf, target_names=iris.
 ↪target_names))
```

```
Training Set Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        39
  versicolor       1.00      1.00      1.00        37
   virginica       1.00      1.00      1.00        44

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120

Testing Set Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

[81]:
```
fig, ax = plt.subplots(figsize=(6, 4))
print("Random Forrest Confusion Matrix:")
cm_rf = confusion_matrix(Y_test, Y_test_pred_rf)
print(cm)
sns.heatmap(cm_rf/np.sum(cm_rf), annot=True, fmt='.0%', cmap='Blues')
plt.title('Random Forrest Confusion matrix')
plt.ylabel('True label')
plt.xlabel("Predicted label,  Accuracy: {0:.2f}".format(accuracy_rf_test))
plt.show()
```

```
Random Forrest Confusion Matrix:
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```



1.2.4 SVM Classifier

[73]:
```
# Create a SVM Classifier
svm = SVC(probability = True)

# Fit a SVM Classifier
svm.fit(X_train, Y_train)

#check the performance metrics of our train model
Y_train_pred_svm = svm.predict(X_train)
accuracy_svm_train = accuracy_score(Y_train, Y_train_pred_svm)
print("Accuracy of Training data: {0:.2f}".format(accuracy_svm_train))

#check the performance metrics of our test model
Y_test_pred_svm = svm.predict(X_test)
accuracy_svm_test = accuracy_score(Y_test_pred_svm, Y_test)
print("Accuracy of Testing data: {0:.2f}".format(accuracy_svm_test))
```

```
Accuracy of Training data: 0.96
Accuracy of Testing data: 1.00
```

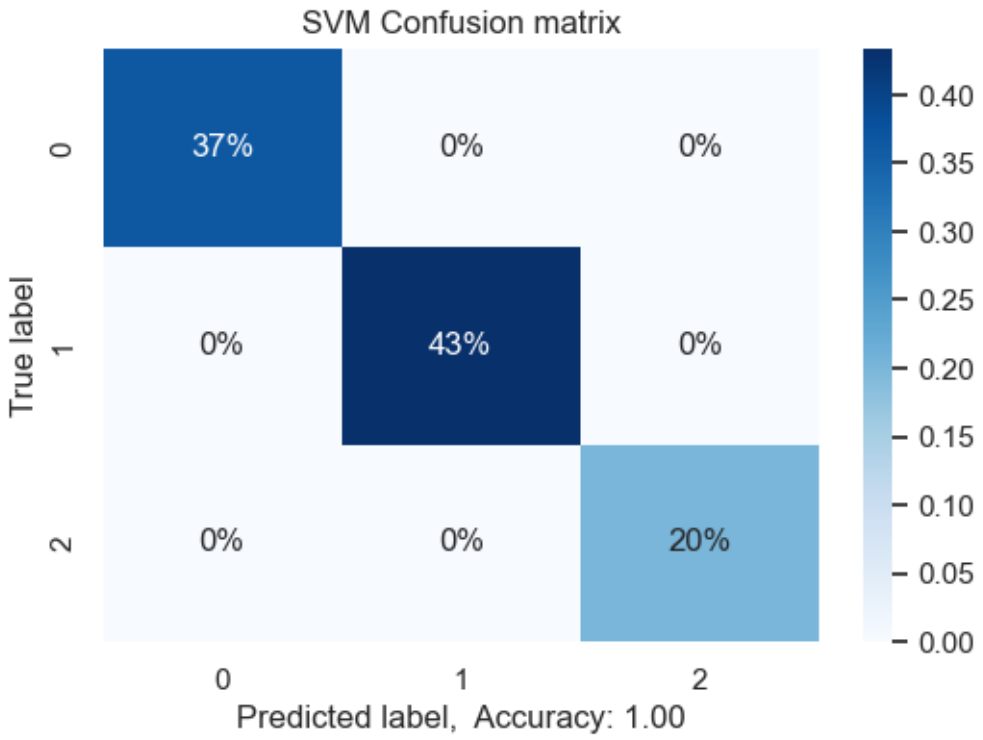
[74]:
```
# Print the classification report for the training set
print("Training Set Classification Report:")
```

```
print(classification_report(Y_train, Y_train_pred_svm, target_names=iris.
 ↪target_names))

# Print the classification report for the testing set
print("Testing Set Classification Report:")
print(classification_report(Y_test, Y_test_pred_svm, target_names=iris.
 ↪target_names))
```

```
Training Set Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        39
  versicolor       0.92      0.95      0.93        37
   virginica       0.95      0.93      0.94        44

    accuracy                           0.96       120
   macro avg       0.96      0.96      0.96       120
weighted avg       0.96      0.96      0.96       120

Testing Set Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        11
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00         6

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

[82]:
```
fig, ax = plt.subplots(figsize=(6, 4))
print("SVM Confusion Matrix:")
cm_svm = confusion_matrix(Y_test, Y_test_pred_svm)
print(cm)
sns.heatmap(cm_svm/np.sum(cm_svm), annot=True, fmt='.0%', cmap='Blues')
plt.title('SVM Confusion matrix')
plt.ylabel('True label')
plt.xlabel("Predicted label,  Accuracy: {0:.2f}".format(accuracy_svm_test))
plt.show()
```

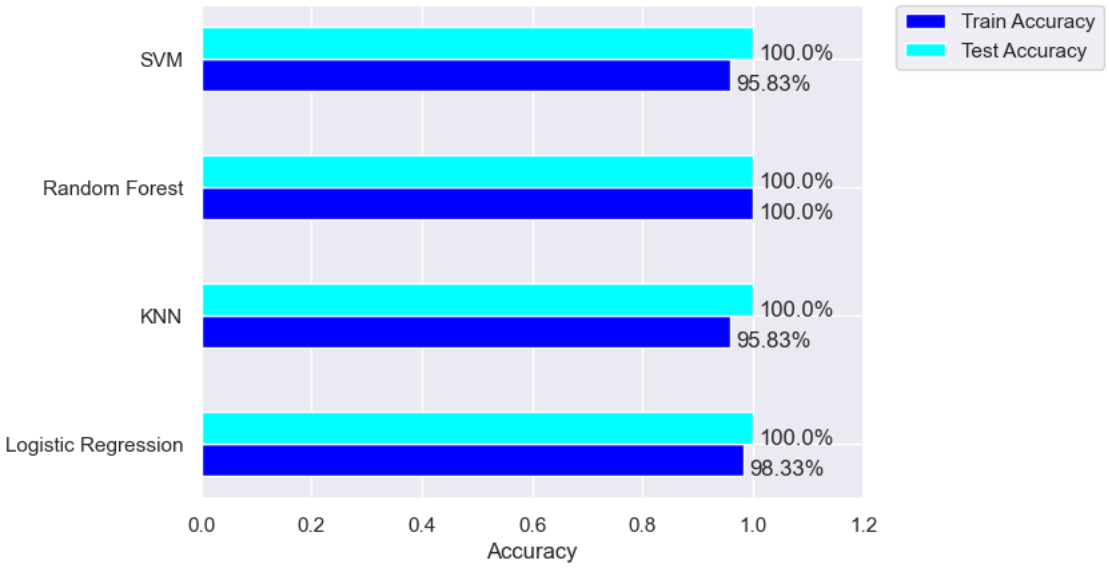
```
SVM Confusion Matrix:
[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
```



Accuracy is already close to perfect on both training and testing data, it might not be necessary to perform `Cross-validation`. However, cross-validation is generally a good practice to ensure that your model is not overfitting to the training data and to obtain a more accurate estimate of the model's performance. Cross-validation can also help to identify the best hyperparameters for your model. Therefore, it is recommended to perform cross-validation even if the accuracy is high.

[76]:
```
import numpy as np
import matplotlib.pyplot as plt

# Define the models to be compared
models = ['Logistic Regression', 'KNN', 'Random Forest', 'SVM']

# Define the train and test accuracy for each model
train_acc = [accuracy_logreg_train, accuracy_knn_train, accuracy_rf_train,␣
 ↪accuracy_svm_train]
test_acc = [accuracy_logreg_test, accuracy_knn_test, accuracy_rf_test,␣
 ↪accuracy_svm_test]

# Set the position of the bars on the x-axis
r1 = np.arange(len(models))

# Plot the horizontal bars
plt.barh(r1, train_acc, height=0.25, color='blue', label='Train Accuracy')
```

```
plt.barh(r1 + 0.25, test_acc, height=0.25, color='cyan', label='Test Accuracy')

# Add accuracy labels to the right of each bar
for i, (train, test) in enumerate(zip(train_acc, test_acc)):
    plt.text(test+0.01, i+0.12, str(round(test*100, 2))+'%')
    plt.text(train+0.01, i-0.12, str(round(train*100, 2))+'%')

# Add xticks on the middle of the group bars
plt.yticks(r1 + 0.125, models)
plt.xlabel('Accuracy')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left', borderaxespad=0.)

# Set the x-axis limits to fit the labels
plt.xlim(0, 1.2)

# Show the plot
plt.show()
```



Based on the results of the models, it appears that all four models **(Logistic Regression, KNN, Random Forest, and SVM)** performed very well on the given dataset. The training accuracy for all four models was high, with **Random Forest** achieving a perfect score of 1.0. The test accuracy for all four models was also perfect at 1.0.

However, it is important to note that these results may be somewhat limited, as we did not perform cross-validation, regularization, or feature selection/engineering. It is possible that with further analysis and refinement, the performance of these models could be further improved.

In conclusion, based on these results, it appears that the **Random Forest** model may be the best choice for this particular dataset. However, further analysis and refinement would be needed to

fully assess the performance of these models.