Chapter 2

In [1]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_iris
```

In [2]:
```
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['species'] = data.target
df.head()
```

Out[2]:

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

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

In [4]:
```
# Multiple column features matrix to convert to NumPy Array
df.loc[:, feature_names]
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [5]:

```
# Convert to numpy array
x = df.loc[:, feature_names].values
```

In [6]:

```
# Make sure NumPy array is two dimensional
x.shape
```

Out[6]:

```
(150, 4)
```

In [7]:

```
# Pandas series to convert to NumPy Array
df.loc[:, 'species']
```

Out[7]:

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
```

```
147    2
148    2
149    2
Name: species, Length: 150, dtype: int32
```

In [8]:
```
y = df.loc[:, 'species'].values
```

In [9]:
```
y.shape
```

Out[9]:
```
(150,)
```

2_3

In [10]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

In [11]:
```
df = pd.read_csv("data/linear.csv")
df.head()
```

Out[11]:

	x	y
0	0.000000	-51.000000
1	25.000000	-12.000000
2	117.583220	134.907414
3	108.922466	134.085180
4	69.887445	NaN

In [12]:
```
# Look at the shape of the dataframe
df.shape
```

Out[12]: (102, 2)

In [13]:
```
# There are missing values in the y column which is what we will predict
df.isnull().sum()
```

Out[13]:
```
x    0
y    8
dtype: int64
```

In [14]:
```
# Remove entire rows from dataframe if they contain any nans in them or 'all'
# this may not be the best strategy for our dataset
df = df.dropna(how = 'any')
```

In [15]:
```
# There are no more missing values
df.isnull().sum()
```

Out[15]:
```
x    0
y    0
dtype: int64
```

In [16]:
```
df.shape
```

Out[16]: (94, 2)

In [17]:
```
# Convert x column to numpy array
X = df.loc[:, ['x']].values
```

In [18]:
```
# Features Matrix needs to be at 2 dimensional
X.shape
```

Out[18]: (94, 1)

In [19]:
```
y = df.loc[:, 'y'].values
```

In [20]:
```
y.shape
```

Out[20]:
```
(94,)
```

In [21]:
```
# Make a linear regression instance
reg = LinearRegression(fit_intercept=True)
```

In [22]:
```
reg.fit(X,y)
```

Out[22]:
```
LinearRegression()
```

In [23]:
```
# Input needs to be two dimensional (reshape makes input two dimensional )
reg.predict(X[0].reshape(-1,1))
```

Out[23]:
```
array([-50.99119328])
```

In [24]:
```
reg.predict(X[0:10])
```

Out[24]:
```
array([-50.99119328, -11.39905237, 135.223663  , 121.50775193,
       102.37289634,  31.0056196 ,   4.46431068,  74.84474012,
        20.82088826,  72.16749711])
```

In [25]:
```
score = reg.score(X, y)
print(score)
```

```
0.979881836115762
```

In [26]:
```
reg.coef_
```

Out[26]:
```
array([1.58368564])
```

In [27]:
```
reg.intercept_
```

Out[27]:
```
-50.99119328333397
```

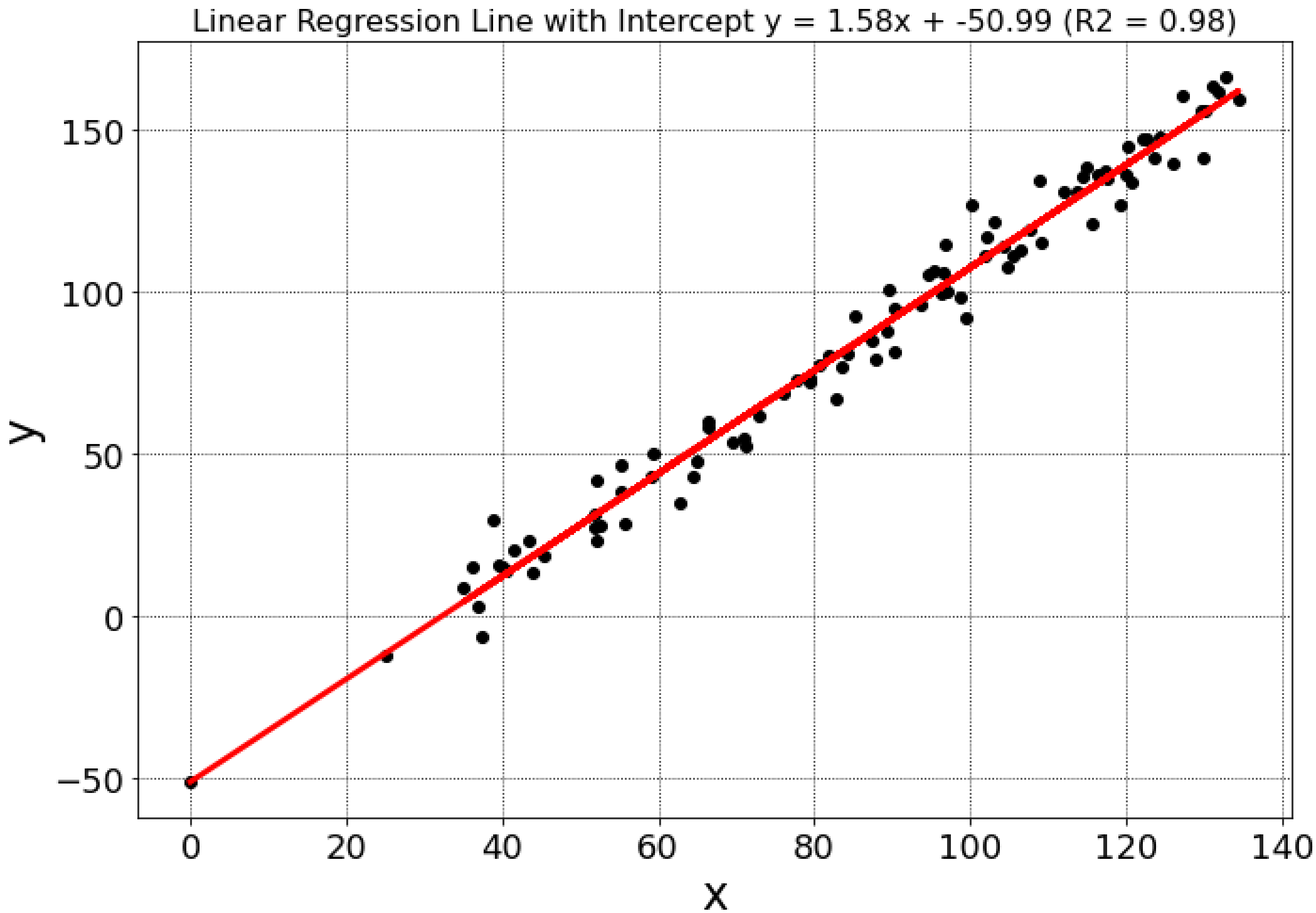
In [28]:
```
m = reg.coef_[0]

b = reg.intercept_
```

```
# following slope intercept form
print("formula: y = {:.2f}x + {:.2f}".format(m, b) )
```

```
formula: y = 1.58x + -50.99
```

In [29]:

```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10,7));

ax.scatter(X, y, color='black');
ax.plot(X, reg.predict(X), color='red',linewidth=3);
ax.grid(True,
        axis = 'both',
        zorder = 0,
        linestyle = ':',
        color = 'k')
ax.tick_params(labelsize = 18)
ax.set_xlabel('x', fontsize = 24)
ax.set_ylabel('y', fontsize = 24)
ax.set_title("Linear Regression Line with Intercept y = {:.2f}x + {:.2f} (R2 = {:.2f})".format(m, b, score), fontsize = 1
fig.tight_layout()
#fig.savefig('images/linearregression', dpi = 300)
```



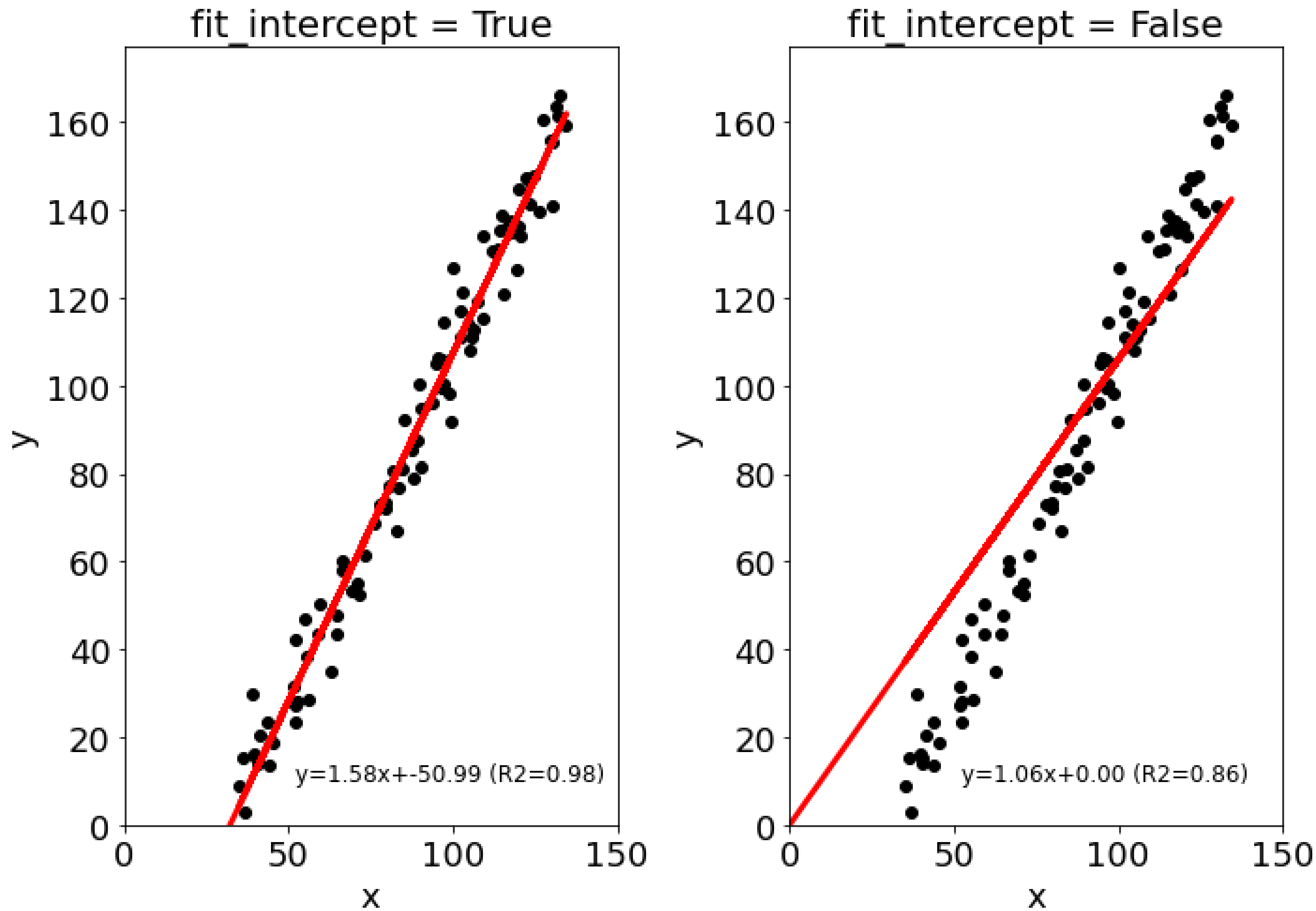
In [30]:

```
# Model with Intercept (like earlier in notebook)
reg_inter = LinearRegression(fit_intercept=True)
reg_inter.fit(X,y)
predictions_inter = reg_inter.predict(X)
score_inter = reg_inter.score(X, y)
```

In [31]:

```
fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (10,7));

for index, model in enumerate([LinearRegression(fit_intercept=True), LinearRegression(fit_intercept=False)]):
    model.fit(X,y)
    predictions = model.predict(X)
```

```
    score = model.score(X, y)
    m = model.coef_[0]
    b = model.intercept_

    ax[index].scatter(X, y, color='black');
    ax[index].plot(X, model.predict(X), color='red',linewidth=3);

    ax[index].tick_params(labelsize = 18)
    ax[index].set_xlabel('x', fontsize = 18)
    ax[index].set_ylabel('y', fontsize = 18)
    ax[index].set_xlim(left = 0, right = 150)
    ax[index].set_ylim(bottom = 0)

    ax[index].text(50, 10, " y={:.2f}x+{:.2f} (R2={:.2f})".format(m, b, score), fontsize = 12)

ax[0].set_title('fit_intercept = True', fontsize = 20)
ax[1].set_title('fit_intercept = False',  fontsize = 20)
fig.tight_layout()
```



Train test split

In [32]:

```
%matplotlib inline

import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_boston

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression
```

In [33]:

```
data = load_boston()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
```

Out[33]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [34]:

```
X = df.loc[:, ['RM', 'LSTAT', 'PTRATIO']].values
```

In [35]:

```
y = df.loc[:, 'target'].values
```

In [36]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=3)
```

In [37]:

```
# Make a linear regression instance
reg = LinearRegression(fit_intercept=True)

# Train the model on the training set.
reg.fit(X_train, y_train)
```

Out[37]:

```
LinearRegression()
```

Model Performance

In [ ]:

```
# Test the model on the testing set and evaluate the performance
score = reg.score(X_test, y_test)
print(score)
```

2_5

In [38]:
```
%matplotlib inline

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn import metrics
```

In [39]:
```
df = pd.read_csv('data/modifiedIris2Classes.csv')
```

In [40]:
```
df.shape
```

Out[40]:
```
(100, 5)
```

In [41]:
```
X_train, X_test, y_train, y_test = train_test_split(df[['petal length (cm)']], df['target'], random_state=0)
```

In [42]:
```
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [43]:
```
clf = LogisticRegression()
```

In [44]:
```
clf.fit(X_train, y_train)
```

Out[44]:
```
LogisticRegression()
```

In [45]:
```
# One observation's petal length after standardization
X_test[0].reshape(1,-1)
```

Out[45]:
```
array([[-0.12093628]])
```

In [46]:
```
print('prediction', clf.predict(X_test[0].reshape(1,-1))[0])
print('probability', clf.predict_proba(X_test[0].reshape(1,-1)))
```

```
prediction 0
probability [[0.52720087 0.47279913]]
```

In [47]:
```
example_df = pd.DataFrame()
example_df.loc[:, 'petal length (cm)'] = X_test.reshape(-1)
example_df.loc[:, 'target'] = y_test.values
example_df['logistic_preds'] = pd.DataFrame(clf.predict_proba(X_test))[1]
```

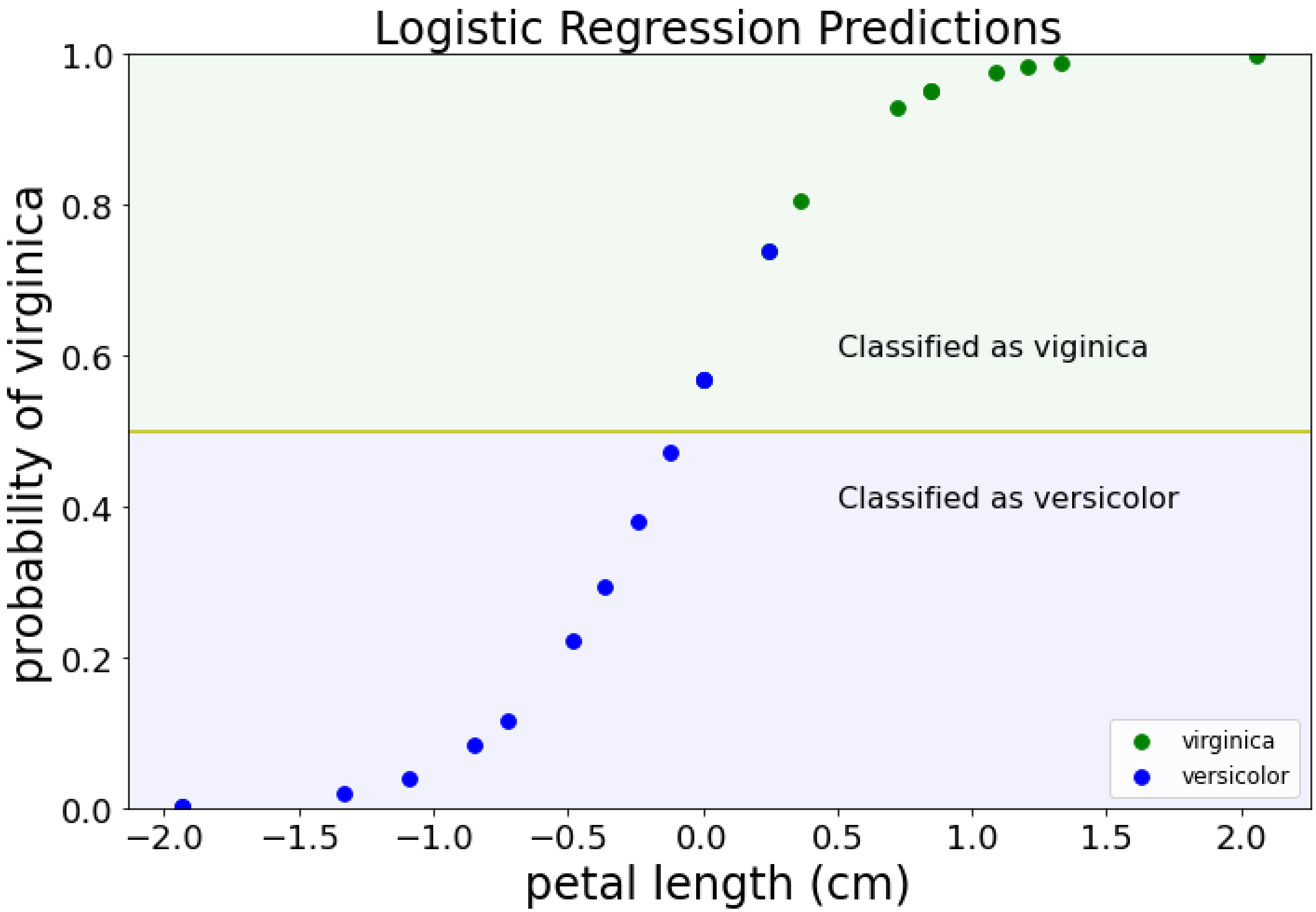
In [48]:
```
example_df.head()
```

Out[48]:

	petal length (cm)	target	logistic_preds
0	-0.120936	0	0.472799
1	0.846554	1	0.950658
2	0.000000	0	0.568197
3	2.055917	1	0.998879
4	1.330299	1	0.988926

In [49]:
```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10,7));


virginicaFilter = example_df['target'] == 1
versicolorFilter = example_df['target'] == 0

ax.scatter(example_df.loc[virginicaFilter, 'petal length (cm)'].values,
```

```
            example_df.loc[virginicaFilter, 'logistic_preds'].values,
           color = 'g',
           s = 60,
           label = 'virginica')


ax.scatter(example_df.loc[versicolorFilter, 'petal length (cm)'].values,
            example_df.loc[versicolorFilter, 'logistic_preds'].values,
           color = 'b',
           s = 60,
           label = 'versicolor')

ax.axhline(y = .5, c = 'y')

ax.axhspan(.5, 1, alpha=0.05, color='green')
ax.axhspan(0, .4999, alpha=0.05, color='blue')
ax.text(0.5, .6, 'Classified as viginica', fontsize = 16)
ax.text(0.5, .4, 'Classified as versicolor', fontsize = 16)

ax.set_ylim(0,1)
ax.legend(loc = 'lower right', markerscale = 1.0, fontsize = 12)
ax.tick_params(labelsize = 18)
ax.set_xlabel('petal length (cm)', fontsize = 24)
ax.set_ylabel('probability of virginica', fontsize = 24)
ax.set_title('Logistic Regression Predictions', fontsize = 24)
fig.tight_layout()
```



In [50]:

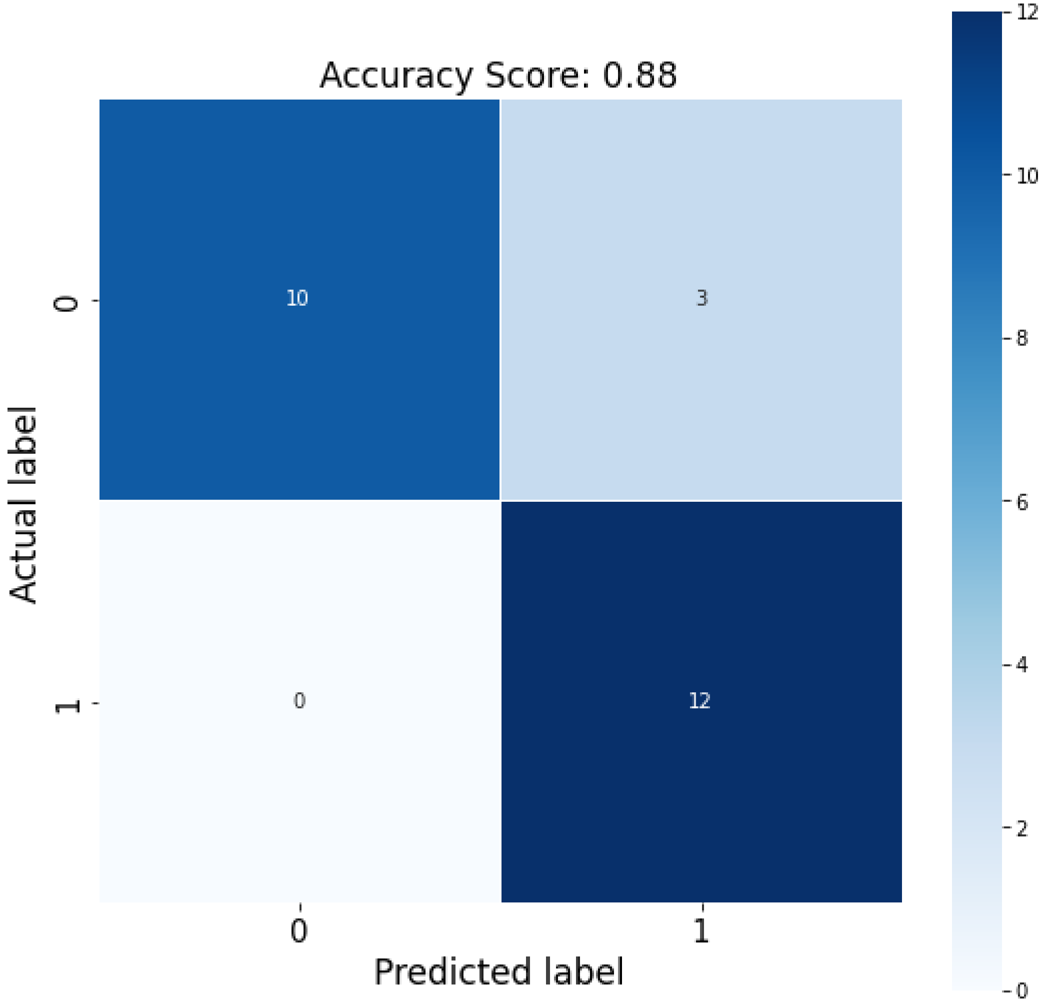
```
score = clf.score(X_test, y_test)
print(score)
```

```
0.88
```

In [51]:

```
cm = metrics.confusion_matrix(y_test, clf.predict(X_test))

plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True,
            fmt=".0f",
            linewidths=.5,
            square = True,
```

```
            cmap = 'Blues');
plt.ylabel('Actual label', fontsize = 17);
plt.xlabel('Predicted label', fontsize = 17);
plt.title('Accuracy Score: {}'.format(score), size = 17);
plt.tick_params(labelsize= 15)
```

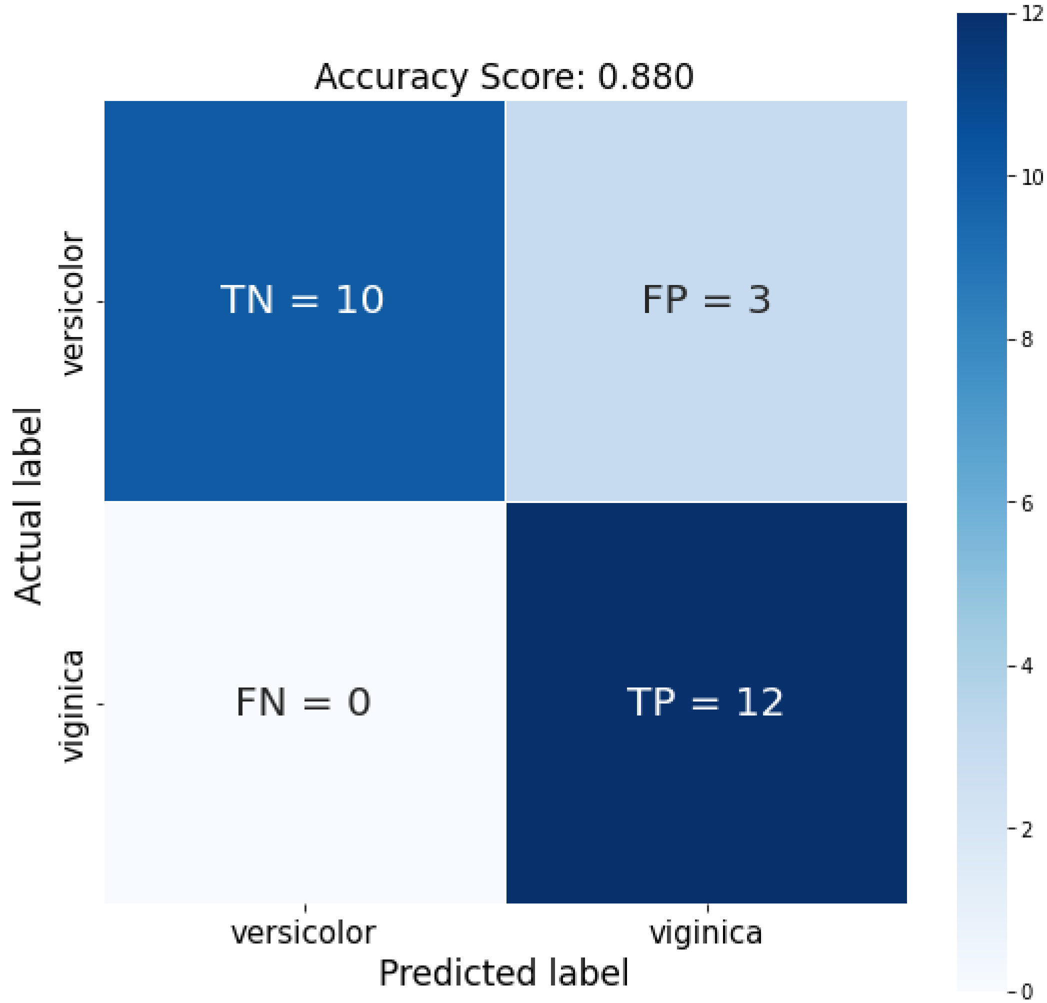


In [52]:
```
# ignore this code

modified_cm = []
for index,value in enumerate(cm):
    if index == 0:
        modified_cm.append(['TN = ' + str(value[0]), 'FP = ' + str(value[1])])
```

```
    if index == 1:
        modified_cm.append(['FN = ' + str(value[0]), 'TP = ' + str(value[1])])
```

In [53]:

```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=np.array(modified_cm),
            fmt="",
            annot_kws={"size": 20},
            linewidths=.5,
            square = True,
            cmap = 'Blues',
            xticklabels = ['versicolor', 'viginica'],
            yticklabels = ['versicolor', 'viginica'],
            );

plt.ylabel('Actual label', fontsize = 17);
plt.xlabel('Predicted label', fontsize = 17);
plt.title('Accuracy Score: {:.3f}'.format(score), size = 17);
plt.tick_params(labelsize= 15)
```



2_6 one vs Rest

In [54]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression
```

In [55]:
```
df = pd.read_csv('data/modifiedDigits4Classes.csv')
```

In [56]:
```
df.head()
```

Out[56]:

	0	1	2	3	4	5	6	7	8	9	...	55	56	57	58	59	60	61	62	63	label
0	0	0	5	13	9	1	0	0	0	0	...	0	0	0	6	13	10	0	0	0	0
1	0	0	0	12	13	5	0	0	0	0	...	0	0	0	0	11	16	10	0	0	1
2	0	0	0	4	15	12	0	0	0	0	...	0	0	0	0	3	11	16	9	0	2
3	0	0	7	15	13	1	0	0	0	8	...	0	0	0	7	13	13	9	0	0	3
4	0	0	1	9	15	11	0	0	0	0	...	0	0	0	1	10	13	3	0	0	0

5 rows × 65 columns

In [57]:
```
df.shape
```

Out[57]:
```
(720, 65)
```

Visualize Each Digit

In [59]:
```
pixel_colnames = df.columns[:-1]
```

In [60]:
```
pixel_colnames
```

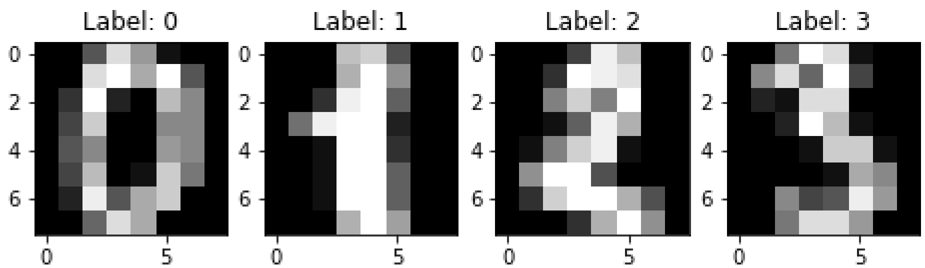
Out[60]:
```
Index(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12',
       '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24',
       '25', '26', '27', '28', '29', '30', '31', '32', '33', '34', '35', '36',
       '37', '38', '39', '40', '41', '42', '43', '44', '45', '46', '47', '48',
       '49', '50', '51', '52', '53', '54', '55', '56', '57', '58', '59', '60',
       '61', '62', '63'],
      dtype='object')
```

In [61]:
```
# Get all columns except the label column for the first image
```

```
image_values = df.loc[0, pixel_colnames].values
```

In [62]:

```
plt.figure(figsize=(10,2))
for index in range(0, 4):

    plt.subplot(1, 5, 1 + index )
    image_values = df.loc[index, pixel_colnames].values
    image_label = df.loc[index, 'label']
    plt.imshow(image_values.reshape(8,8), cmap ='gray')
    plt.title('Label: ' + str(image_label))
```



In [63]:

```
X_train, X_test, y_train, y_test = train_test_split(df[pixel_colnames], df['label'], random_state=0)
```

In [64]:

```
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [65]:

```
# multi_class is specifying one versus rest
clf = LogisticRegression(solver='liblinear',
                         multi_class='ovr',
                         random_state = 0)

clf.fit(X_train, y_train)
print('Training accuracy:', clf.score(X_train, y_train))
print('Test accuracy:', clf.score(X_test, y_test))
```

```
Training accuracy: 1.0
Test accuracy: 1.0
```

In [66]:

```
clf.intercept_
```

Out[66]:

```
array([-2.712674  , -3.54379096, -3.18367757, -2.623974  ])
```

In [67]:

```
clf.coef_.shape
```

Out[67]:

```
(4, 64)
```

In [68]:

```
clf.predict_proba(X_test[0:1])
```

Out[68]:

```
array([[0.00183123, 0.98368966, 0.00536378, 0.00911533]])
```

In [69]:

```
clf.predict(X_test[0:1])
```

Out[69]:

```
array([1], dtype=int64)
```

2_7 Decision trees

In [71]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

In [72]:

```
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
```

Out[72]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target

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

In [73]:

```
X_train, X_test, y_train, y_test = train_test_split(df[data.feature_names], df['target'], random_state=0)
```

In [74]:

```
clf = DecisionTreeClassifier(max_depth = 2,
                             random_state = 0)
```

In [75]:

```
clf.fit(X_train, y_train)
```

Out[75]:

```
DecisionTreeClassifier(max_depth=2, random_state=0)
```

In [76]:

```
# Predict for One Observation
clf.predict(X_test.iloc[0].values.reshape(1, -1))
```

Out[76]:

```
array([2])
```

In [77]:

```
clf.predict(X_test[0:10])
```

Out[77]:

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1])
```

In [78]:

```
score = clf.score(X_test, y_test)
print(score)
```

```
0.8947368421052632
```

In [79]:

```
# List of values to try for max_depth:
max_depth_range = list(range(1, 6))
```

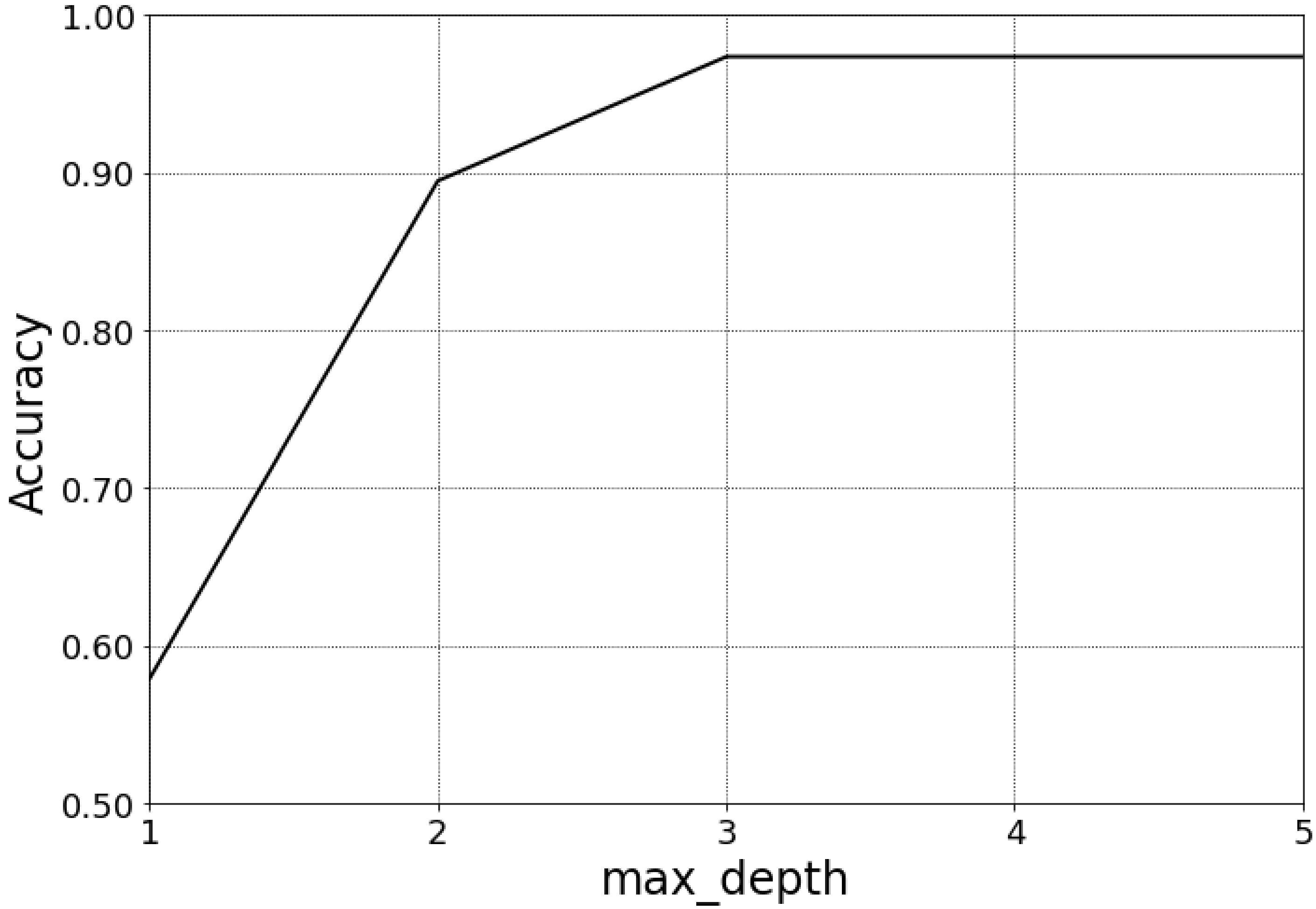
```
# List to store the average RMSE for each value of max_depth:
accuracy = []

for depth in max_depth_range:

    clf = DecisionTreeClassifier(max_depth = depth,
                                 random_state = 0)
    clf.fit(X_train, y_train)

    score = clf.score(X_test, y_test)
    accuracy.append(score)
```

In [80]:

```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10,7));

ax.plot(max_depth_range,
        accuracy,
        lw=2,
        color='k')

ax.set_xlim([1, 5])
ax.set_ylim([.50, 1.00])
ax.grid(True,
        axis = 'both',
        zorder = 0,
        linestyle = ':',
        color = 'k')

yticks = ax.get_yticks()

y_ticklist = []
for tick in yticks:
    y_ticklist.append(str(tick).ljust(4, '0')[0:4])
ax.set_yticklabels(y_ticklist)
ax.tick_params(labelsize = 18)
ax.set_xticks([1,2,3,4,5])
ax.set_xlabel('max_depth', fontsize = 24)
ax.set_ylabel('Accuracy', fontsize = 24)
fig.tight_layout()
#fig.savefig('images/max_depth_vs_accuracy.png', dpi = 300)
```

```
C:\Users\aadar\AppData\Local\Temp/ipykernel_992/1914883130.py:21: UserWarning: FixedFormatter should only be used togethe
r with FixedLocator
```

```
ax.set_yticklabels(y_ticklist)
```



2_8 Modeling pattern

In [81]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn import tree
```

In [83]:

```
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
df.head()
```

Out[83]:

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

In [84]:

```
X_train, X_test, Y_train, Y_test = train_test_split(df[data.feature_names], df['target'], random_state=0)
```

In [85]:

```
clf = DecisionTreeClassifier(max_depth = 2,
                             random_state = 0)
```

In [86]:

```
clf.fit(X_train, Y_train)
```

Out[86]:

```
DecisionTreeClassifier(max_depth=2, random_state=0)
```

In [87]:

```
clf.predict(X_test[0:10])
```

Out[87]:

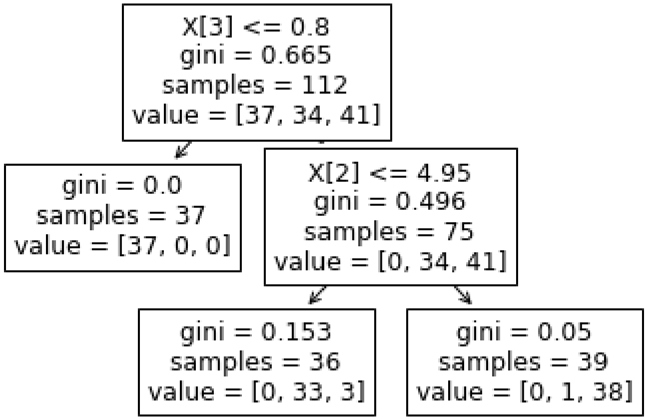
```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1])
```

In [88]:

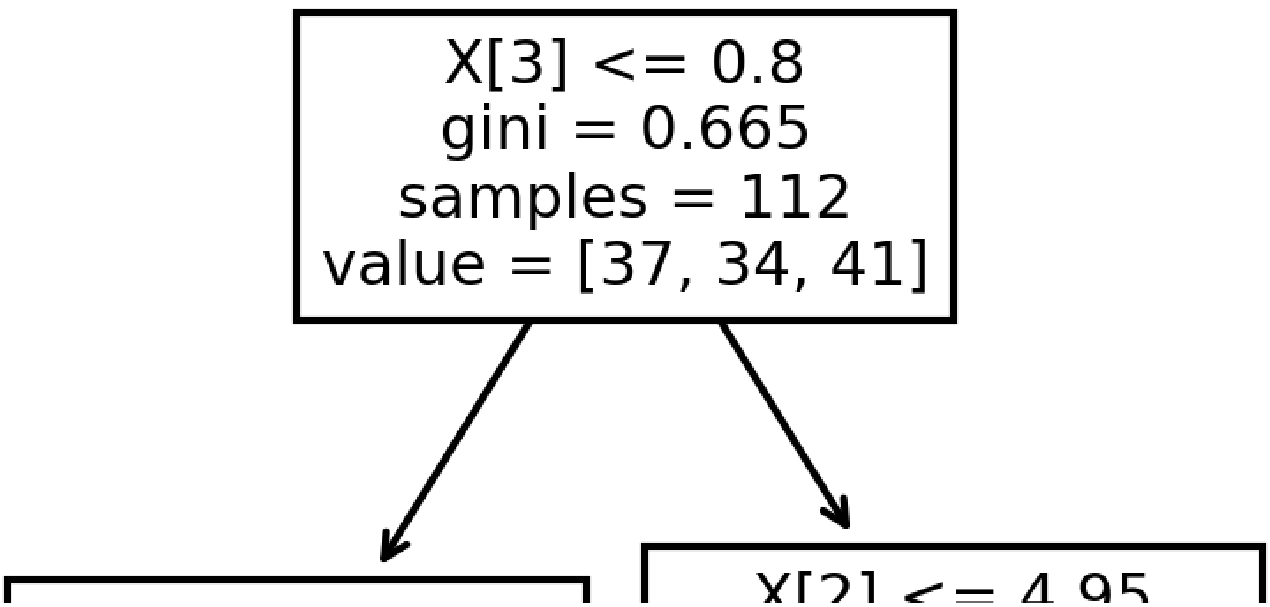
```
score = clf.score(X_test, Y_test)
print(score)
```

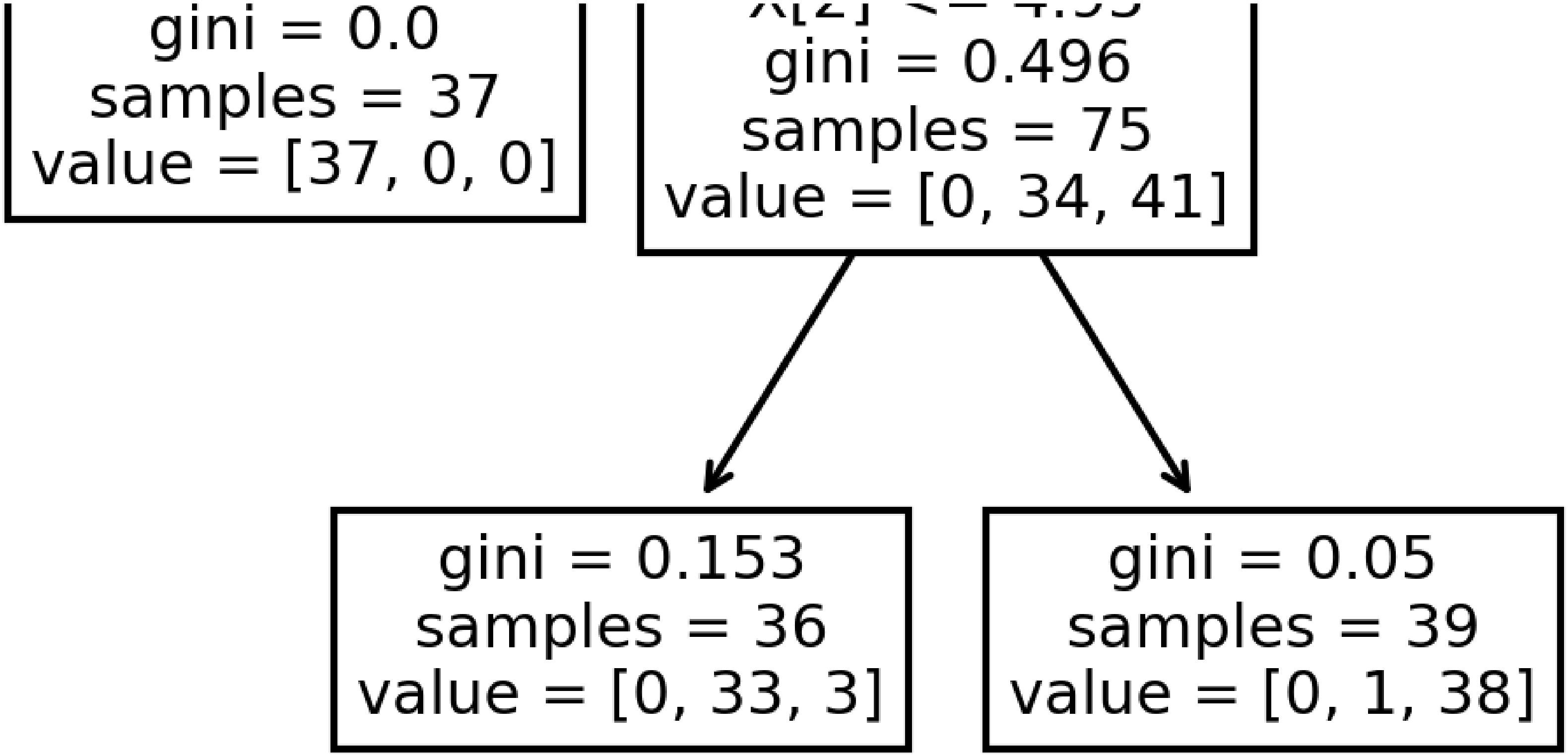
```
0.8947368421052632
```

In [89]:
```
tree.plot_tree(clf);
```



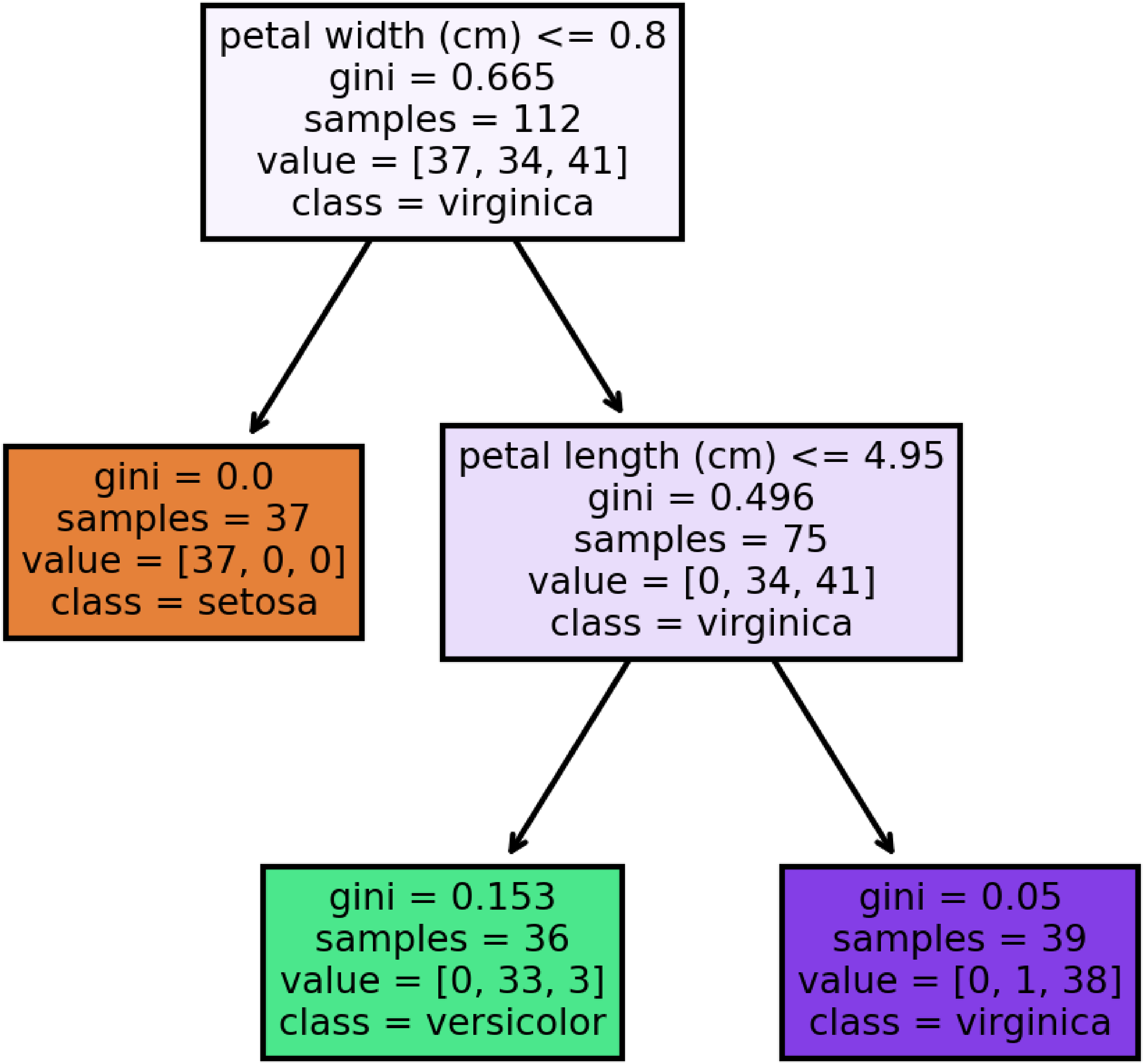
In [91]:
```
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4,4), dpi = 300)

tree.plot_tree(clf);
```





In [92]:
```
# Putting the feature names and class names into variables
fn = ['sepal length (cm)','sepal width (cm)','petal length (cm)','petal width (cm)']
cn = ['setosa', 'versicolor', 'virginica']
```

In [93]:
```
fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4,4), dpi = 300)

tree.plot_tree(clf,
               feature_names = fn,
               class_names=cn,
               filled = True);
fig.savefig('images/plottreefncn.png')
```


petal width (cm) <= 0.8
gini = 0.665
samples = 112
value = [37, 34, 41]
class = virginica
gini = 0.0
samples = 37
value = [37, 0, 0]
class = setosa
petal length (cm) <= 4.95
gini = 0.496
samples = 75
value = [0, 34, 41]
class = virginica
gini = 0.153
samples = 36
value = [0, 33, 3]
class = versicolor
gini = 0.05
samples = 39
value = [0, 1, 38]
class = virginica

Bagged Trees

In [94]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split

# Bagged Trees Regressor
from sklearn.ensemble import BaggingRegressor
```

In [95]:

```
df = pd.read_csv('data/kc_house_data.csv')

df.head()
```

Out[95]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_b
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

5 rows × 21 columns

In [96]:

```
# This notebook only selects a couple features for simplicity
# However, I encourage you to play with adding and substracting more features
```

```
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors']

X = df.loc[:, features]

y = df.loc[:, 'price'].values
```

In [97]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [98]:

```
reg = BaggingRegressor(n_estimators=100,
                       random_state = 0)
```

In [99]:

```
reg.fit(X_train, y_train)
```

Out[99]:

```
BaggingRegressor(n_estimators=100, random_state=0)
```

In [100…

```
# Returns a NumPy Array
# Predict for One Observation
reg.predict(X_test.iloc[0].values.reshape(1, -1))
```

Out[100…

```
array([353334.6])
```

In [101…

```
reg.predict(X_test[0:10])
```

Out[101…

```
array([ 353334.6 , 1011004.77,  450212.76,  418593.  ,  772871.7 ,
        405436.5 ,  361353.02,  720323.9 ,  580438.82, 1623570.8 ])
```

In [102…

```
score = reg.score(X_test, y_test)
print(score)
```

```
0.5786196798753096
```

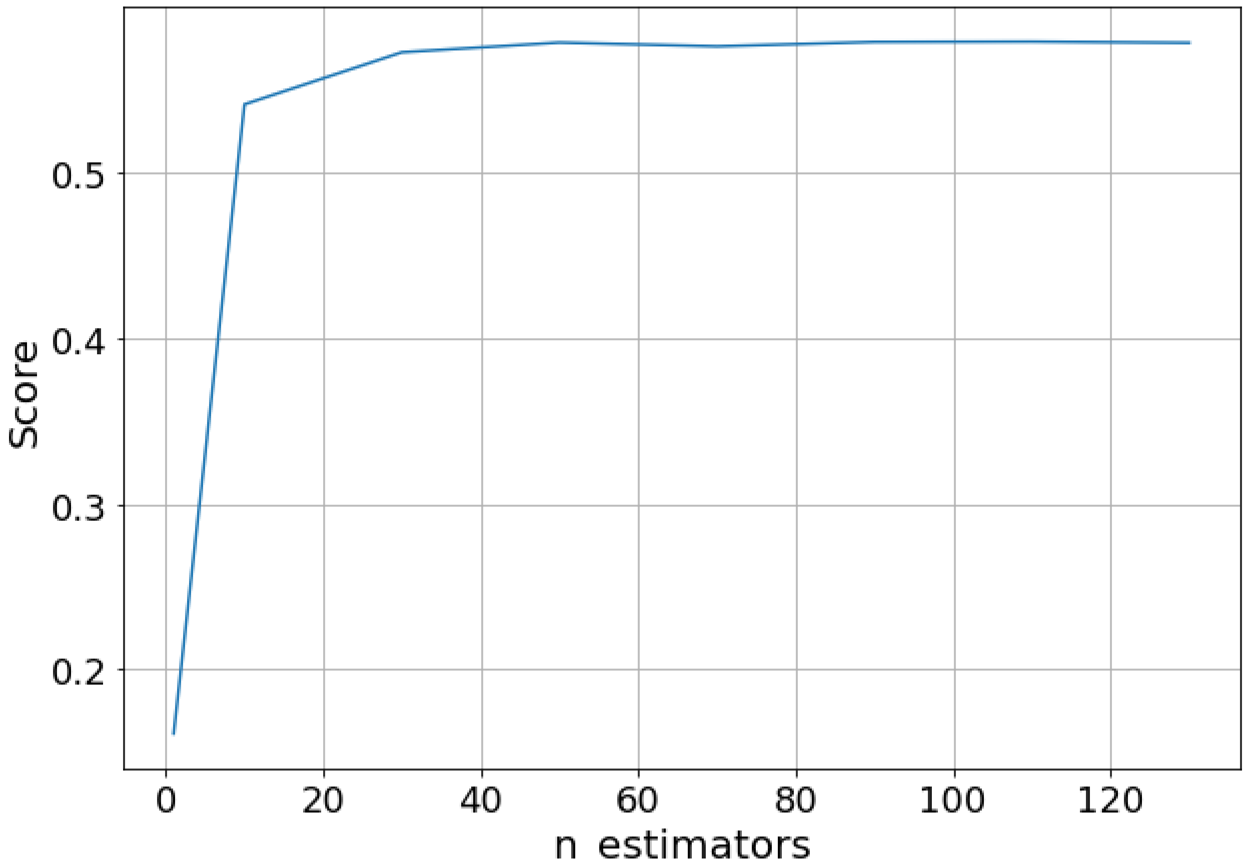
In [103…

```
# List of values to try for n_estimators:
estimator_range = [1] + list(range(10, 150, 20))

scores = []

for estimator in estimator_range:
```

```
    reg = BaggingRegressor(n_estimators=estimator, random_state=0)
    reg.fit(X_train, y_train)
    scores.append(reg.score(X_test, y_test))
```

In [104…

```
plt.figure(figsize = (10,7))
plt.plot(estimator_range, scores);

plt.xlabel('n_estimators', fontsize =20);
plt.ylabel('Score', fontsize = 20);
plt.tick_params(labelsize = 18)
plt.grid()
```



In [105…

```
### 2_10_Random_Forest
```

In [106…

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Regression Models
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor

# Classifer Models
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier

# To visualize individual decision trees
from sklearn import tree
from sklearn.tree import export_text
```

In [107…

```
df = pd.read_csv('data/kc_house_data.csv')

df.head()
```

Out[107…

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_b
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

5 rows × 21 columns

In [108…

```
# This notebook only selects a couple features for simplicity
# However, I encourage you to play with adding and substracting features
```

```python
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors']

X = df.loc[:, features]

y = df.loc[:, 'price'].values
```

In [109…

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

In [110…

```python
reg = RandomForestRegressor(n_estimators=100, random_state = 0)
```

In [111…

```python
reg.fit(X_train, y_train)
```

Out[111…

```
RandomForestRegressor(random_state=0)
```

In [112…

```python
reg.predict(X_test[0:10])
```

Out[112…

```
array([ 354008.56,  999809.  ,  443760.25,  426332.  ,  760570.2 ,
        408775.5 ,  360030.14,  714794.4 ,  585902.14, 1665779.  ])
```

In [113…

```python
score = reg.score(X_test, y_test)
print(score)
```

```
0.577684658845681
```

In [114…

```python
# Load the Iris Dataset
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df[data.feature_names], df['target'], random_state=0)
```

In [115…

```python
# Fir Bagged Tree Model
btc = BaggingClassifier(n_estimators=100,
                        random_state = 1)

btc.fit(X_train, y_train)
```

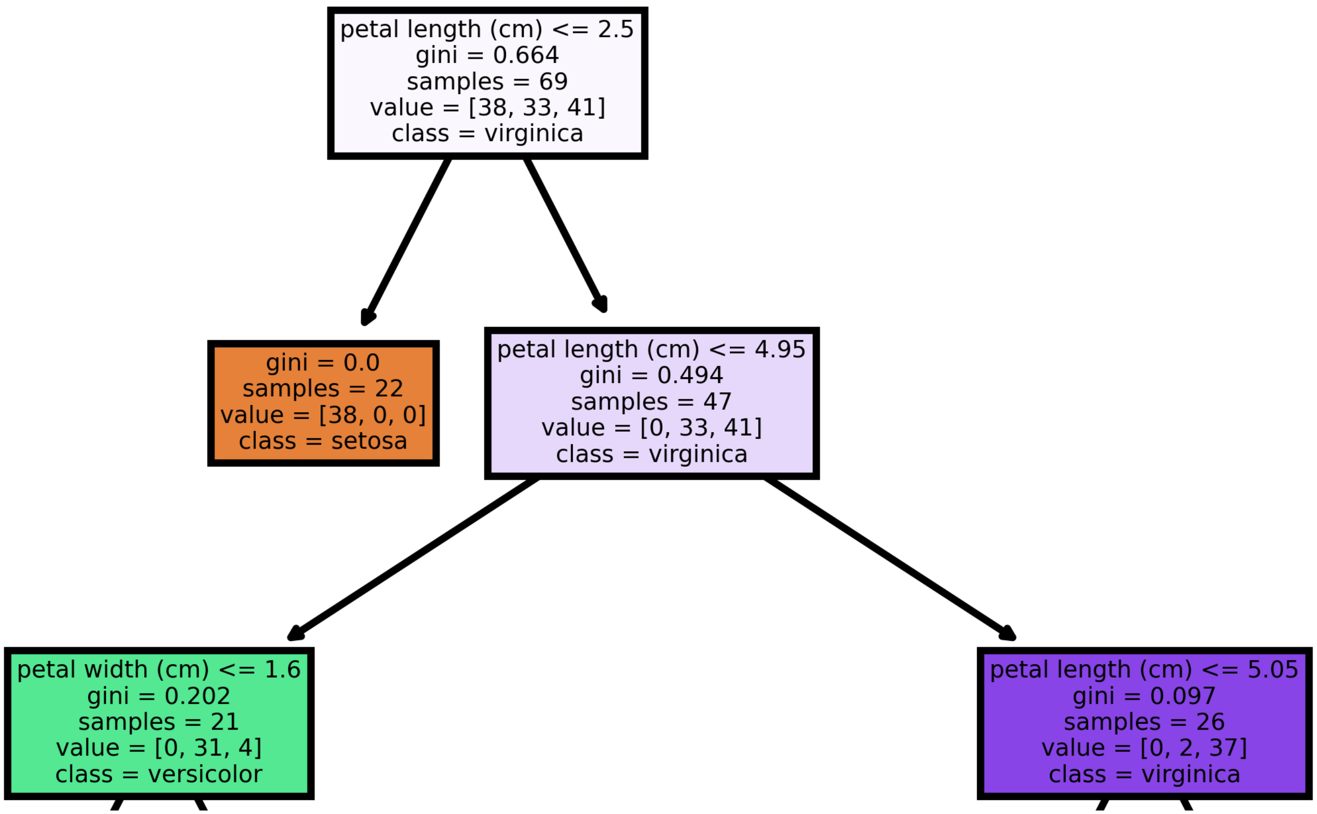
```
# Fit Random Forest Model
rfc = RandomForestClassifier(n_estimators=100, random_state = 1)

rfc.fit(X_train, y_train)
```

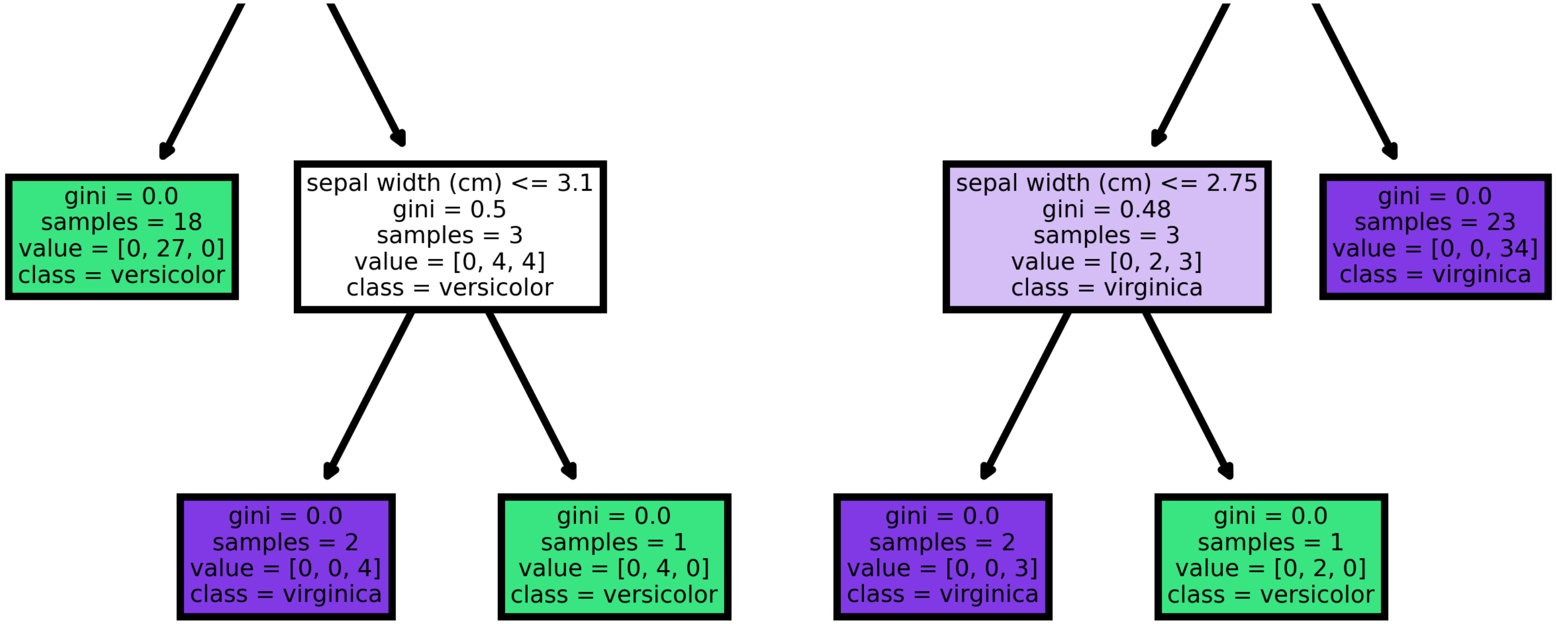
Out[115…

```
RandomForestClassifier(random_state=1)
```

In [116…

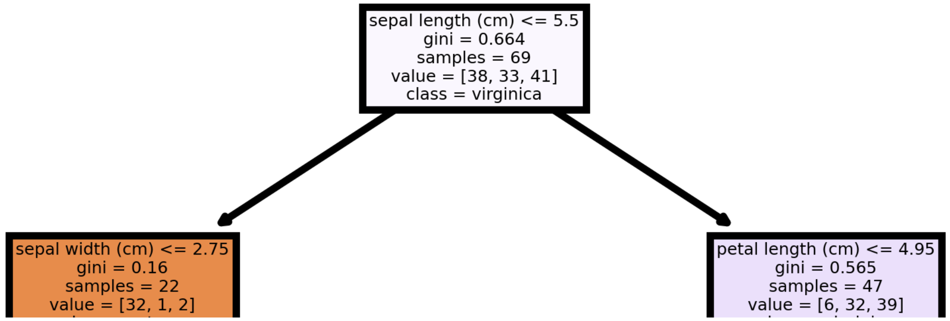
```
# Get the first decision tree for bagged tree model
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(btc.estimators_[0],
               feature_names = data.feature_names,
               class_names=data.target_names,
               filled = True);
```

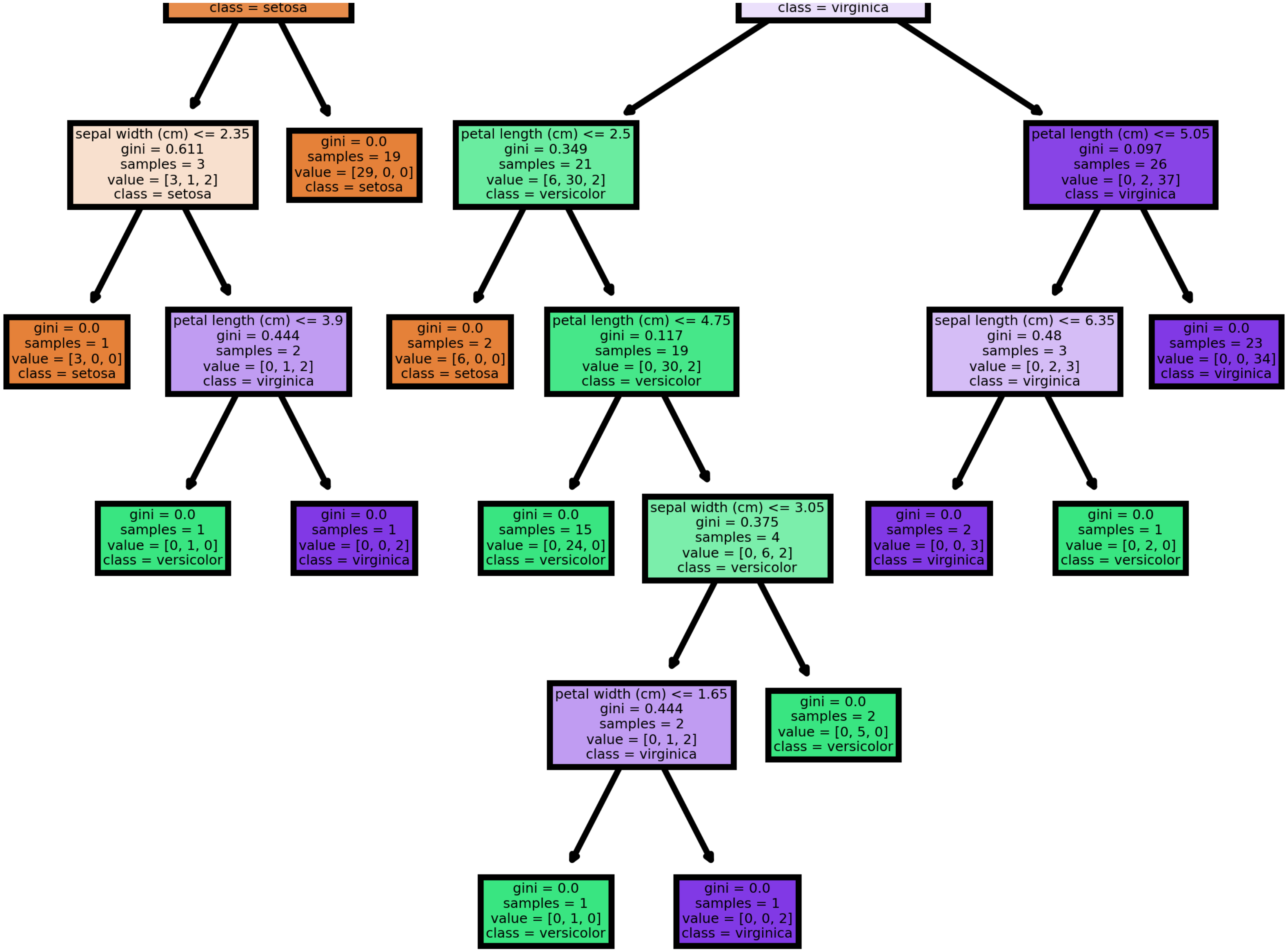




In [117…

```
# Get the first decision tree for a random forest model
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(rfc.estimators_[0],
               feature_names = data.feature_names,
               class_names=data.target_names,
               filled = True);
```





In [118…

```
importances = pd.DataFrame({'feature':X_train.columns,'importance':np.round(rfc.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False)
```

In [119...

```
importances
```

Out[119...

	feature	importance
2	petal length (cm)	0.453
3	petal width (cm)	0.385
0	sepal length (cm)	0.139
1	sepal width (cm)	0.023

Chapter 3 KMeans

In [120...

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans
```

In [121...

```
data = load_iris()
df = pd.DataFrame(data.data, columns=data.feature_names)
df.head()
```

Out[121...

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

In [122...

```
features = ['petal length (cm)','petal width (cm)']

# Create features matrix
x = df.loc[:, features].values
```

In [123...

```
y = data.target
```

In [124...

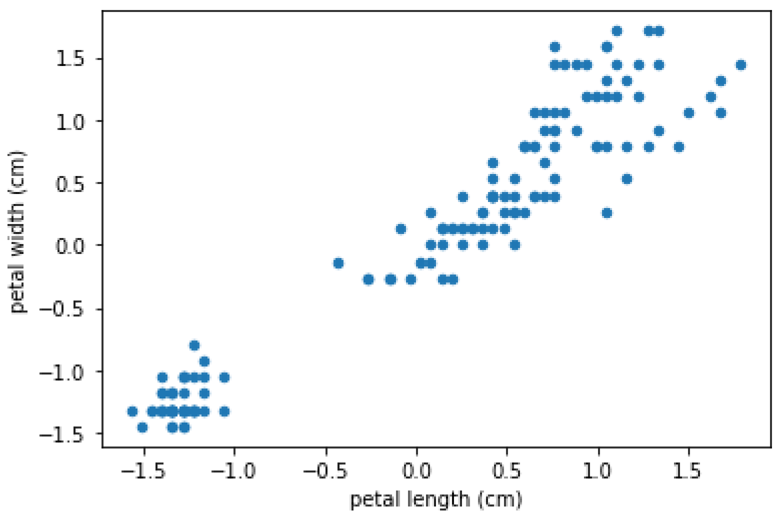
```
# Apply Standardization to features matrix X
x = df.loc[:, features].values
```

In [125...

```
x = StandardScaler().fit_transform(x)
```

In [126...

```
# Plot
pd.DataFrame(x, columns = features).plot.scatter('petal length (cm)','petal width (cm)' )

# Add labels
plt.xlabel('petal length (cm)');
plt.ylabel('petal width (cm)');
```



In [127...

```
# Make an instance of KMeans with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=1)
```

```
# Fit only on a features matrix
kmeans.fit(x)
```

Out[127…

```
KMeans(n_clusters=3, random_state=1)
```

In [128…

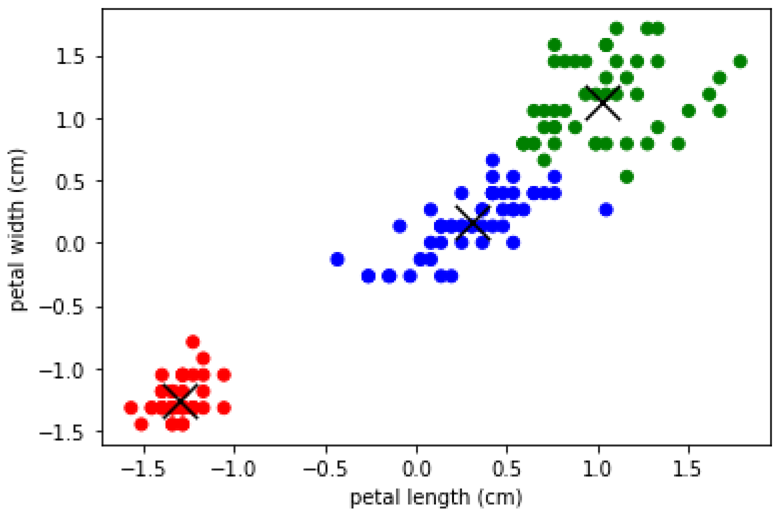
```
# Get labels and cluster centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
```

In [129…

```
x = pd.DataFrame(x, columns = features)
```

In [130…

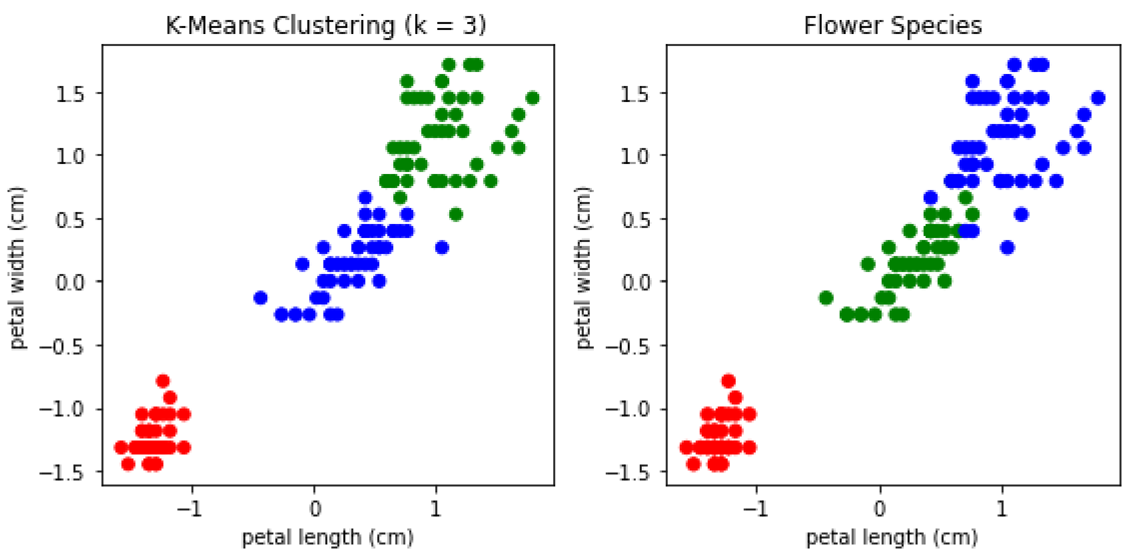
```
colormap = np.array(['r', 'g', 'b'])
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=colormap[labels])
plt.scatter(centroids[:,0], centroids[:,1], s = 300, marker = 'x', c = 'k')

plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)');
```



In [131…

```
plt.figure(figsize=(8,4))

plt.subplot(1, 2, 1)
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=colormap[labels])
plt.xlabel('petal length (cm)')
```

```
plt.ylabel('petal width (cm)');
plt.title('K-Means Clustering (k = 3)')

plt.subplot(1, 2, 2)
plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=colormap[y], s=40)
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)');
plt.title('Flower Species')

plt.tight_layout()
```



3_3 PCA

In [132…

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd

from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA
```

In [133…

```
data = load_iris()
```

```
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target
```

In [134…

```
speciesDict = {0: 'setosa', 1:'versicolor', 2:'virginica'}

df.loc[:,'target'] = df.loc[:, 'target'].apply(lambda x: speciesDict[x])
```

In [135…

```
df.head()
```

Out[135…

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

In [136…

```
# Apply Standardization to features matrix X
features = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)','petal width (cm)']
x = df.loc[:, features].values
y = df.loc[:,['target']].values
```

In [137…

```
x = StandardScaler().fit_transform(x)
```

In [138…

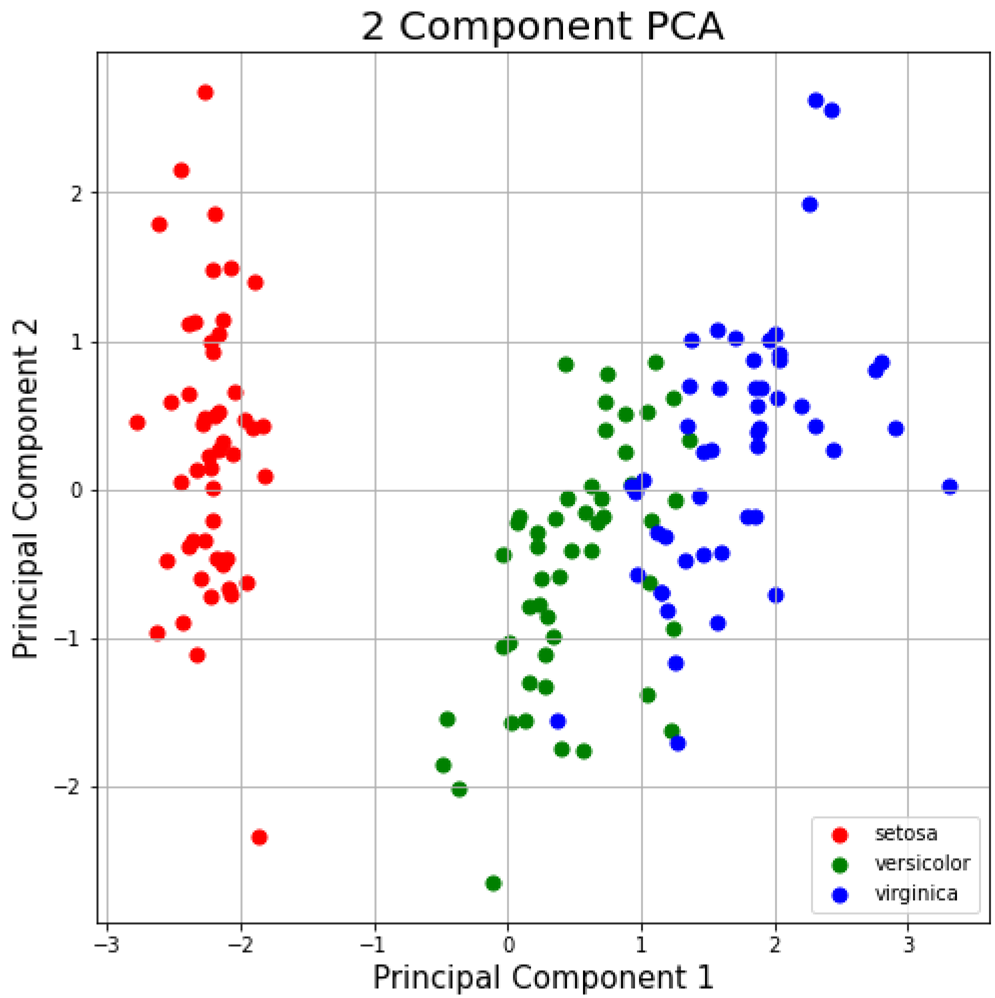
```
# Make an instance of PCA
pca = PCA(n_components=2)

# Fit and transform the data
principalComponents = pca.fit_transform(x)

principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])
```

In [139…

```
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
```

In [140…

```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (8,8));
targets = df.loc[:, 'target'].unique()
colors = ['r', 'g', 'b']

for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)

ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA', fontsize = 20)
ax.legend(targets)
ax.grid()
```



In [141…

```
pca.explained_variance_ratio_
```

Out[141…

```
array([0.72962445, 0.22850762])
```

In [142…

```
sum(pca.explained_variance_ratio_)
```

Out[142…

```
0.9581320720000164
```

#3_04

In [143…

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

In [144…

```
df = pd.read_csv('data/MNISTonly0_1.csv')
```

In [145…

```
df.head()
```

Out[145…

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	label
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

5 rows × 785 columns

In [146…

```
pixel_colnames = df.columns[:-1]
```

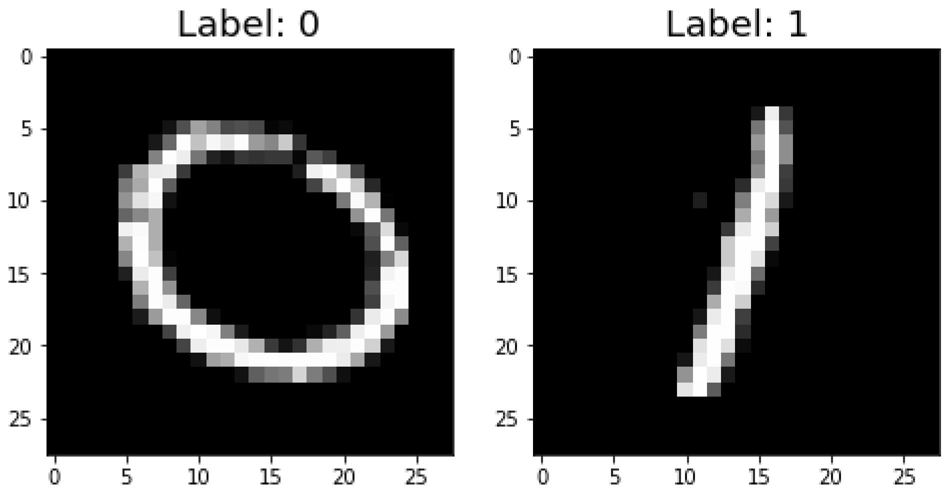
In [147…

```
# Get all columns except the label column for the first image
image_values = df.loc[0, pixel_colnames].values
```

In [148…

```
plt.figure(figsize=(8,4))
for index in range(0, 2):
```

```
    plt.subplot(1, 2, 1 + index )
    image_values = df.loc[index, pixel_colnames].values
    image_label = df.loc[index, 'label']
    plt.imshow(image_values.reshape(28,28), cmap ='gray')
    plt.title('Label: ' + str(image_label), fontsize = 18)
```



In [149…

```
X_train, X_test, y_train, y_test = train_test_split(df[pixel_colnames], df['label'], random_state=0)
```

In [150…

```
scaler = StandardScaler()

# Fit on training set only.
scaler.fit(X_train)

# Apply transform to both the training set and the test set.
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [151…

```
# Variable created for demonstational purposes in the notebook
scaledTrainImages = X_train.copy()
```

In [152…

```
"""
n_components = .90 means that scikit-learn will choose the minimum number
of principal components such that 90% of the variance is retained.
"""
```

```
pca = PCA(n_components = .90)

# Fit PCA on training set only
pca.fit(X_train)

# Apply the mapping (transform) to both the training set and the test set.
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Logistic Regression
clf = LogisticRegression()
clf.fit(X_train, y_train)

print('Number of dimensions before PCA: ' + str(len(pixel_colnames)))
print('Number of dimensions after PCA: ' + str(pca.n_components_))
print('Classification accuracy: ' + str(clf.score(X_test, y_test)))
```

```
Number of dimensions before PCA: 784
Number of dimensions after PCA: 104
Classification accuracy: 0.997
```

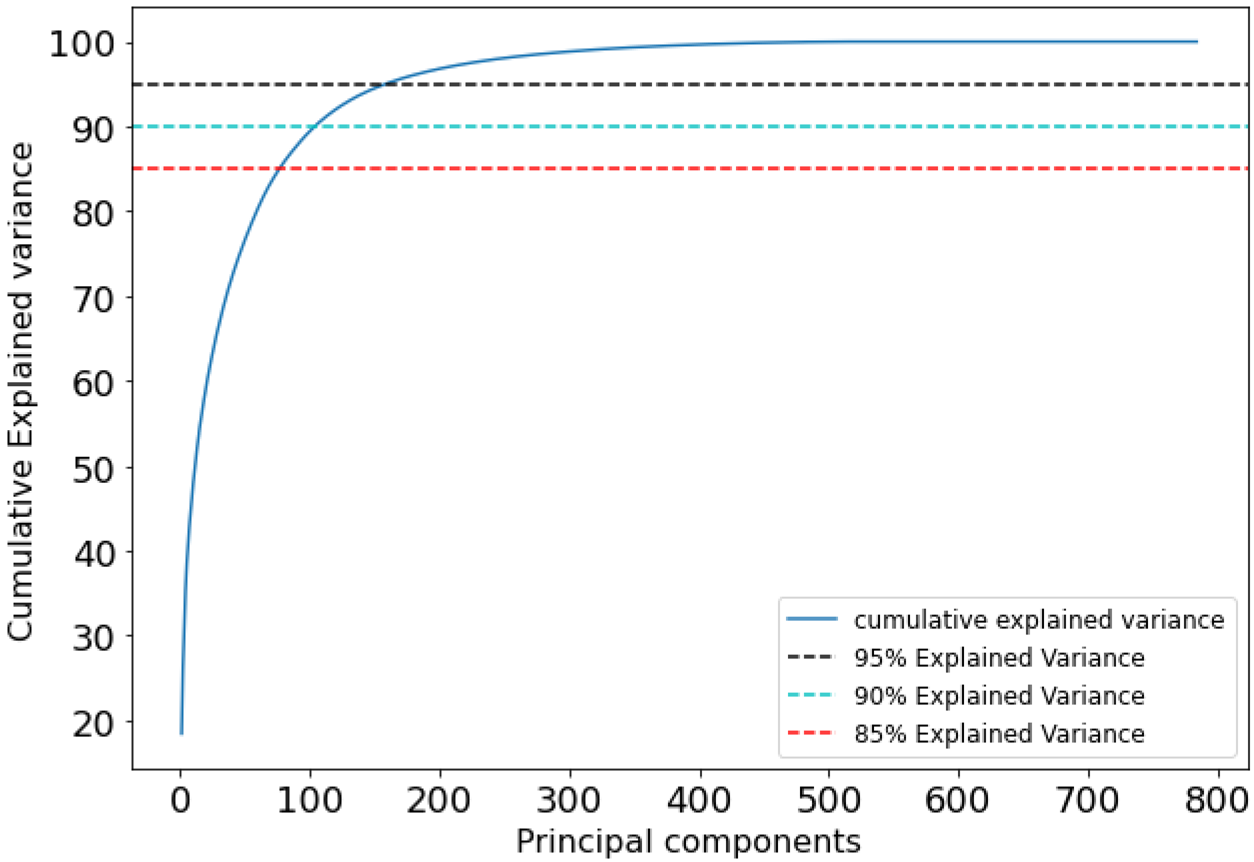
In [153…

```
# if n_components is not set, all components are kept (784 in this case)
pca = PCA()

pca.fit(scaledTrainImages)

# Summing explained variance
tot = sum(pca.explained_variance_)

var_exp = [(i/tot)*100 for i in sorted(pca.explained_variance_, reverse=True)]

# Cumulative explained variance
cum_var_exp = np.cumsum(var_exp)

# PLOT OUT THE EXPLAINED VARIANCES SUPERIMPOSED
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10,7));
ax.tick_params(labelsize = 18)
ax.plot(range(1, 785), cum_var_exp, label='cumulative explained variance')
ax.set_ylabel('Cumulative Explained variance', fontsize = 16)
ax.set_xlabel('Principal components', fontsize = 16)
ax.axhline(y = 95, color='k', linestyle='--', label = '95% Explained Variance')
ax.axhline(y = 90, color='c', linestyle='--', label = '90% Explained Variance')
```

```
ax.axhline(y = 85, color='r', linestyle='--', label = '85% Explained Variance')
ax.legend(loc='best', markerscale = 1.0, fontsize = 12)
```

Out[153…

```
<matplotlib.legend.Legend at 0x1a0693ba4f0>
```



Chapter 3_05 pipelines

In [154…

```
%matplotlib inline

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

from sklearn.pipeline import Pipeline
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
```

In [155…

```
df = pd.read_csv('data/MNISTonly0_1.csv')
```

In [156…

```
df.head()
```

Out[156…

	0	1	2	3	4	5	6	7	8	9	...	775	776	777	778	779	780	781	782	783	label
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

5 rows × 785 columns

In [157…

```
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(df[df.columns[:-1]], df['label'], random_state=0)

# Standardize Data
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# Apply PCA
pca = PCA(n_components = .90, random_state=0)
pca.fit(X_train)
X_train = pca.transform(X_train)
X_test = pca.transform(X_test)

# Apply Logistic Regression
clf = LogisticRegression()
```

```python
clf.fit(X_train, y_train)

# Get Model Performance
print(clf.score(X_test, y_test))
```

0.997

In [158…

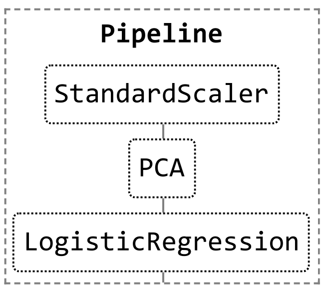
```python
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(df[df.columns[:-1]], df['label'], random_state=0)

# Create a pipeline
pipe = Pipeline([('scaler', StandardScaler()),
                 ('pca', PCA(n_components = .90, random_state=0)),
                 ('logistic', LogisticRegression())])

pipe.fit(X_train, y_train)

# Get Model Performance
print(pipe.score(X_test, y_test))
```

0.997

In [159…

```python
from sklearn import set_config

set_config(display='diagram')
pipe
```

Out[159…



In [ ]: