2.1

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv("data/headbrain1.csv")

X = df["Head Size(cm^3)"]
y = df["Brain Weight(grams)"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=104, test_size=0.25, shuffle=True
)

print("X_train: ", X_train.head())
print("X_test: ", X_test.head())
print("y_train: ", y_train.head())
print("y_test: ", y_test.head())
```

```
X_train:  99     3478
52     4270
184    3479
139    3171
107    3399
Name: Head Size(cm^3), dtype: int64
X_test:  66     3415
113    3594
135    3436
227    4204
68     4430
Name: Head Size(cm^3), dtype: int64
y_train:  99     1270
52     1335
184    1160
139    1127
107    1226
Name: Brain Weight(grams), dtype: int64
y_test:  66     1310
113    1290
135    1235
227    1380
68     1510
Name: Brain Weight(grams), dtype: int64
```

In [2]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

df = pd.read_csv("data/headbrain1.csv")
print(df.shape)
print(df.head())
X = df["Head Size(cm^3)"]
```

```
y = df["Brain Weight(grams)"]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=104, train_size=0.8, shuffle=True
)
print("X_train.head(): \n", X_train.head())
print("X_train.shape: \n", X_train.shape)
print("X_test: \n", X_test.head())
print("X_test.shape: \n", X_test.shape)
print("y_train: \n", y_train.head())
print("y_train.shape: \n", y_train.shape)
print("y_test: \n", y_test.head())
print("y_test.shape: \n", y_test.shape)
```

```
(237, 2)
   Head Size(cm^3)  Brain Weight(grams)
0             4512                 1530
1             3738                 1297
2             4261                 1335
3             3777                 1282
4             4177                 1590
X_train.head():
 110    3695
164    3497
58     3935
199    3297
182    4005
Name: Head Size(cm^3), dtype: int64
X_train.shape:
 (189,)
X_test:
 66     3415
113    3594
135    3436
227    4204
68     4430
Name: Head Size(cm^3), dtype: int64
X_test.shape:
 (48,)
y_train:
 110    1310
164    1280
58     1330
199    1220
182    1280
Name: Brain Weight(grams), dtype: int64
y_train.shape:
 (189,)
y_test:
 66     1310
113    1290
135    1235
227    1380
68     1510
Name: Brain Weight(grams), dtype: int64
y_test.shape:
 (48,)
```

In [3]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

df = pd.read_csv("data/Real-estate.csv")
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.05, random_state=0
)
display("X_train: ", X_train)
display("y_train: ", y_train)
```

'X_train: '

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude
37	38	2013.167	12.0	1360.13900	1	24.95204	121.54842
334	335	2012.917	30.0	1013.34100	5	24.99006	121.53460
54	55	2013.083	16.1	289.32480	5	24.98203	121.54348
145	146	2012.917	2.1	451.24380	5	24.97563	121.54694
284	285	2012.917	15.0	383.28050	7	24.96735	121.54464
...	...	...	...	...	...	...	...
323	324	2013.417	28.6	197.13380	6	24.97631	121.54436
192	193	2013.167	43.8	57.58945	7	24.96750	121.54069
117	118	2013.000	13.6	4197.34900	0	24.93885	121.50383
47	48	2013.583	35.9	640.73910	3	24.97563	121.53715
172	173	2013.583	6.6	90.45606	9	24.97433	121.54310

393 rows × 7 columns

```
'y_train: '
37     25.3
334    22.8
54     51.7
145    45.5
284    34.4
       ...
323    42.5
192    42.7
117    13.0
47     61.5
172    58.1
Name: Y house price of unit area, Length: 393, dtype: float64
```

In [4]:
```
import numpy as np
from sklearn.model_selection import train_test_split
```

```
X = np.arange(1, 25).reshape(12, 2)
y = np.array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0])
display(X)
display(y)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=4, stratify=y, shuffle=True
)
display("X_train: ", X_train)
display("y_train: ", y_train)
display("X_test: ", X_test)
display("y_test: ", y_test)
```

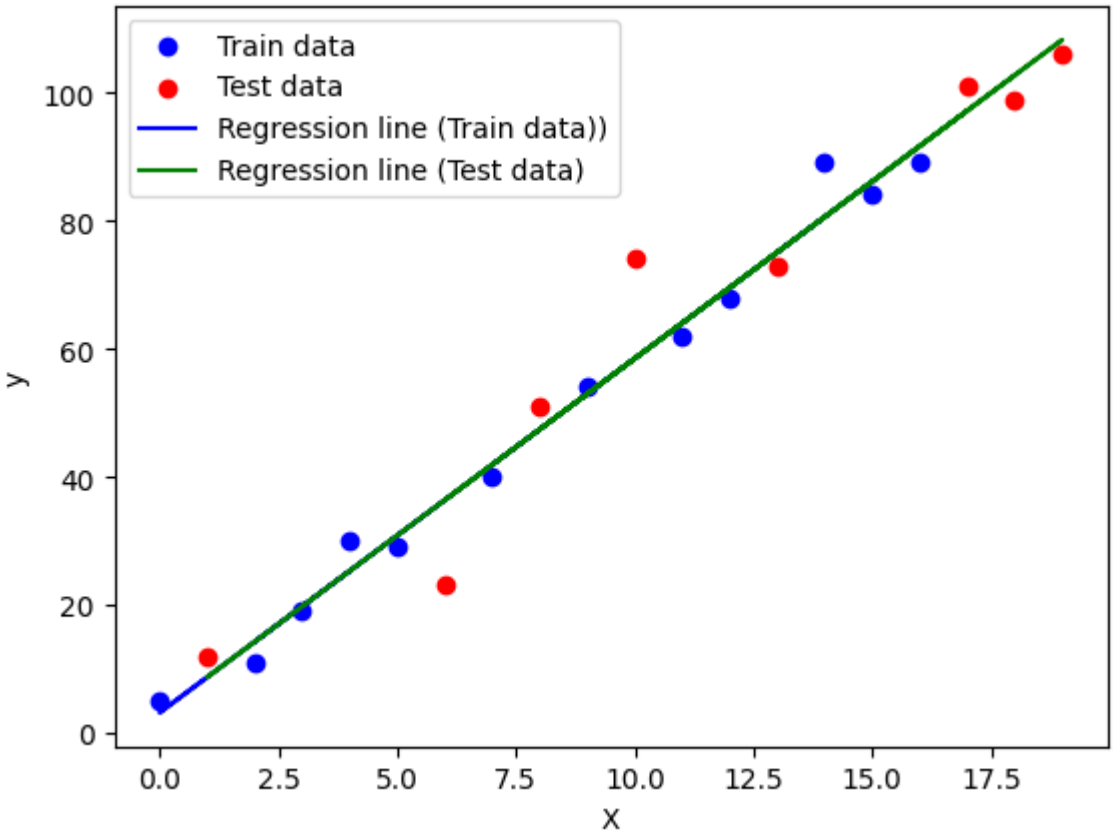
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12],
       [13, 14],
       [15, 16],
       [17, 18],
       [19, 20],
       [21, 22],
       [23, 24]])
array([1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0])
'X_train: '
array([[21, 22],
       [ 3,  4],
       [13, 14],
       [15, 16],
       [17, 18],
       [19, 20],
       [23, 24],
       [ 1,  2]])
'y_train: '
array([1, 0, 1, 0, 1, 0, 0, 1])
'X_test: '
array([[11, 12],
       [ 7,  8],
       [ 5,  6],
       [ 9, 10]])
'y_test: '
array([0, 0, 1, 1])
```

In [5]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

X = np.arange(20).reshape(-1, 1)
y = np.array(
    [5, 12, 11, 19, 30, 29, 23, 40, 51, 54, 74, 62, 68, 73, 89, 84, 89, 101, 99, 10
)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=8, random_state
```

```
model = LinearRegression().fit(X_train, y_train)

print("intercept_: ", model.intercept_)
print("model.coef_: ", model.coef_)
print("score_train: ", model.score(X_train, y_train))
print("score_test: ", model.score(X_test, y_test))

y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

plt.scatter(X_train, y_train, color="blue", label="Train data")
plt.scatter(X_test, y_test, color="red", label="Test data")

plt.plot(X_train, y_train_pred, color="b", label="Regression line (Train data))")
plt.plot(X_test, y_test_pred, color="g", label="Regression line (Test data)")

plt.xlabel("X")
plt.ylabel("y")
plt.legend(loc="upper left")

plt.show()
```

```
intercept_:  3.1617195496417523
model.coef_:  [5.53121801]
score_train:  0.9868175024574795
score_test:  0.9465896927715023
```



2.2

In [6]:
```
import pandas as pd

df = pd.read_csv("data/NaNDataset.csv")
display(df)
df.B = df.B.fillna(df.B.mean())
display("Cleaned dataset: ", df)
```

	A	B	C
0	1	2.0	3
1	4	NaN	6
2	7	NaN	9
3	10	11.0	12
4	13	14.0	15
5	16	17.0	18

'Cleaned dataset: '

	A	B	C
0	1	2.0	3
1	4	11.0	6
2	7	11.0	9
3	10	11.0	12
4	13	14.0	15
5	16	17.0	18

In [7]:
```
import pandas as pd

df = pd.read_csv("data/NaNDataset.csv")
df1 = df.dropna()
display(df1)
df2 = df.reset_index(drop=True)
display(df2)
```

	A	B	C
0	1	2.0	3
3	10	11.0	12
4	13	14.0	15
5	16	17.0	18

	A	B	C
0	1	2.0	3
1	4	NaN	6
2	7	NaN	9
3	10	11.0	12
4	13	14.0	15
5	16	17.0	18

In [8]:
```
import pandas as pd

df = pd.read_csv("data/DuplicateRows.csv")
display(df.duplicated(keep=False))

df.drop_duplicates(keep="first", inplace=True)
display(df)

df.drop_duplicates(subset=["A", "C"], keep="last", inplace=True)
display(df)
```

```
0    False
1     True
2     True
3    False
4    False
5     True
6     True
7    False
8    False
dtype: bool
```

	A	B	C
0	1	2	3
1	4	5	6
3	7	8	9
4	7	18	9
5	10	11	12
7	13	14	15
8	16	17	18

	A	B	C
0	1	2	3
1	4	5	6
4	7	18	9
5	10	11	12
7	13	14	15
8	16	17	18

In [9]:
```
import pandas as pd
from sklearn import preprocessing

df = pd.read_csv("data/NormalizeColumns.csv")
X = df.values.astype(float)
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(X)
df = pd.DataFrame(x_scaled, columns=df.columns)
display(df)
```

	A	B	C
0	0.6	0.000000	0.0
1	0.2	0.200000	0.2
2	0.4	0.266667	0.4
3	0.0	0.600000	0.6
4	0.8	0.800000	0.8
5	1.0	1.000000	1.0

In [10]:
```
import numpy as np
```

```
def outlier_iqr(data):
    q1, q3 = np.percentile(data, [25, 75])
    iqr = q3 - q1
    lower_bound = q1 - (iqr * 1.5)
    upper_bound = q3 + (iqr * 1.5)
    return np.where((data > upper_bound) | (data < lower_bound))
```

In [11]:
```
import pandas as pd

df = pd.read_csv("http://www.mosaic-web.org/go/datasets/galton.csv")
display(df.head())
```

	family	father	mother	sex	height	nkids
0	1	78.5	67.0	M	73.2	4
1	1	78.5	67.0	F	69.2	4
2	1	78.5	67.0	F	69.0	4
3	1	78.5	67.0	F	69.0	4
4	2	75.5	66.5	M	73.5	4

In [12]:
```
print("Outliers using outliers_iqr()")
print("==============================")
for i in outlier_iqr(df.height)[0]:
    display(df[i : i + 1])
```

```
Outliers using outliers_iqr()
==============================
```

	family	father	mother	sex	height	nkids
288	72	70.0	65.0	M	79.0	7

In [13]:
```
def outlier_z_score(data):
    threshold = 3
    mean = np.mean(data)
    std = np.std(data)
    z_scores = [(y - mean) / std for y in data]
    return np.where(np.abs(z_scores) > threshold)
```

In [14]:
```
print("Outliers using outliers_z_score()")
print("==================================")
for i in outlier_z_score(df.height)[0]:
    display(df[i : i + 1])
```

```
Outliers using outliers_z_score()
==================================
```

	family	father	mother	sex	height	nkids
125	35	71.0	69.0	M	78.0	5

	family	father	mother	sex	height	nkids
288	72	70.0	65.0	M	79.0	7

	family	father	mother	sex	height	nkids
672	155	68.0	60.0	F	56.0	7

2.3

In [15]:
```
# Import pandas package
import pandas as pd

# Assign data
data = {
    "Name": ["Jai", "Princi", "Gaurav", "Anuj", "Ravi", "Natasha", "Riya"],
    "Age": [17, 17, 18, 17, 18, 17, 17],
    "Gender": ["M", "F", "M", "M", "M", "F", "F"],
    "Marks": [90, 76, 75.2, 74, 65, 75.2, 71],
}

# Convert into DataFrame
df = pd.DataFrame(data)

# Display data
df
```

Out[15]:

	Name	Age	Gender	Marks
0	Jai	17	M	90.0
1	Princi	17	F	76.0
2	Gaurav	18	M	75.2
3	Anuj	17	M	74.0
4	Ravi	18	M	65.0
5	Natasha	17	F	75.2
6	Riya	17	F	71.0

In [16]:
```
df["Gender"] = df["Gender"].map({"M": 0, "F": 1}).astype(float)
df
```

Out[16]:

	Name	Age	Gender	Marks
0	Jai	17	0.0	90.0
1	Princi	17	1.0	76.0
2	Gaurav	18	0.0	75.2
3	Anuj	17	0.0	74.0
4	Ravi	18	0.0	65.0
5	Natasha	17	1.0	75.2
6	Riya	17	1.0	71.0

In [17]:
```
df = df[df["Marks"] >= 75].copy()

df.drop("Age", axis=1, inplace=True)
df
```

Out[17]:

	Name	Gender	Marks
0	Jai	0.0	90.0
1	Princi	1.0	76.0
2	Gaurav	0.0	75.2
5	Natasha	1.0	75.2

In [18]:
```
import pandas as pd

details = pd.DataFrame(
    {
        "ID": [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
        "NAME": [
            "Jagroop",
            "Praveen",
            "Harjot",
            "Pooja",
            "Rahul",
            "Nikita",
            "Saurabh",
            "Ayush",
            "Dolly",
            "Mohit",
        ],
        "BRANCH": [
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
```

```
            "CSE",
            "CSE",
            "CSE",
        ],
    }
)
display(details)
```

	ID	NAME	BRANCH
0	101	Jagroop	CSE
1	102	Praveen	CSE
2	103	Harjot	CSE
3	104	Pooja	CSE
4	105	Rahul	CSE
5	106	Nikita	CSE
6	107	Saurabh	CSE
7	108	Ayush	CSE
8	109	Dolly	CSE
9	110	Mohit	CSE

In [19]:
```
import pandas as pd

fees_status = pd.DataFrame(
    {
        "ID": [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
        "PENDING": [
            "5000",
            "250",
            "NIL",
            "9000",
            "15000",
            "NIL",
            "4500",
            "1800",
            "250",
            "NIL",
        ],
    }
)

display(fees_status)
```

	ID	PENDING
0	101	5000
1	102	250
2	103	NIL
3	104	9000
4	105	15000
5	106	NIL
6	107	4500
7	108	1800
8	109	250
9	110	NIL

In [20]:

```
import pandas as pd

details = pd.DataFrame(
    {
        "ID": [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
        "NAME": [
            "Jagroop",
            "Praveen",
            "Harjot",
            "Pooja",
            "Rahul",
            "Nikita",
            "Saurabh",
            "Ayush",
            "Dolly",
            "Mohit",
        ],
        "BRANCH": [
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
            "CSE",
        ],
    }
)

fees_status = pd.DataFrame(
    {
        "ID": [101, 102, 103, 104, 105, 106, 107, 108, 109, 110],
```

```
        "PENDING": [
            "5000",
            "250",
            "NIL",
            "9000",
            "15000",
            "NIL",
            "4500",
            "1800",
            "250",
            "NIL",
        ],
    }
)

merged_df = pd.merge(details, fees_status, on="ID")
display(merged_df)
```

	ID	NAME	BRANCH	PENDING
0	101	Jagroop	CSE	5000
1	102	Praveen	CSE	250
2	103	Harjot	CSE	NIL
3	104	Pooja	CSE	9000
4	105	Rahul	CSE	15000
5	106	Nikita	CSE	NIL
6	107	Saurabh	CSE	4500
7	108	Ayush	CSE	1800
8	109	Dolly	CSE	250
9	110	Mohit	CSE	NIL

In [21]:
```
import pandas as pd

car_selling_data = {
    "Brand": [
        "Maruti",
        "Maruti",
        "Maruti",
        "Maruti",
        "Hyundai",
        "Hyundai",
        "Toyota",
        "Mahindra",
        "Mahindra",
        "Ford",
        "Toyora",
        "Ford",
    ],
```

```
    "Year": [2010, 2011, 2009, 2013, 2010, 2011, 2011, 2010, 2013, 2010, 2010, 2011
    "Sold": [6, 7, 9, 8, 3, 5, 2, 8, 7, 2, 4, 2],
}
df = pd.DataFrame(car_selling_data)
display(df)
```

	Brand	Year	Sold
0	Maruti	2010	6
1	Maruti	2011	7
2	Maruti	2009	9
3	Maruti	2013	8
4	Hyundai	2010	3
5	Hyundai	2011	5
6	Toyota	2011	2
7	Mahindra	2010	8
8	Mahindra	2013	7
9	Ford	2010	2
10	Toyora	2010	4
11	Ford	2011	2

In [22]:
```
import pandas as pd

data1 = {
    "Name": ["Jai", "Princi", "Gaurav", "Anuj"],
    "Age": [27, 24, 22, 32],
    "Address": ["Delhi", "Kanpur", "Allahabad", "Kannauj"],
    "Qualification": ["Msc", "MA", "MCA", "Phd"],
    "Mobile No": [97, 91, 58, 76],
}

data2 = {
    "Name": ["Gaurav", "Anuj", "Dhiraj", "Hitesh"],
    "Age": [22, 31, 12, 52],
    "Address": ["Kanpur", "Allahabad", "Kannuaj", "Delhi"],
    "Qualification": ["MCA", "Phd", "Bcom", "B.hons"],
    "Salary": [1000, 2000, 3000, 4000],
}

df = pd.DataFrame(
    data1,
    index=[
        0,
        1,
        2,
        3,
    ],
```

```
)

df1 = pd.DataFrame(data2, index=[2, 3, 6, 7])
res = pd.concat([df, df1])
res
```

Out[22]:

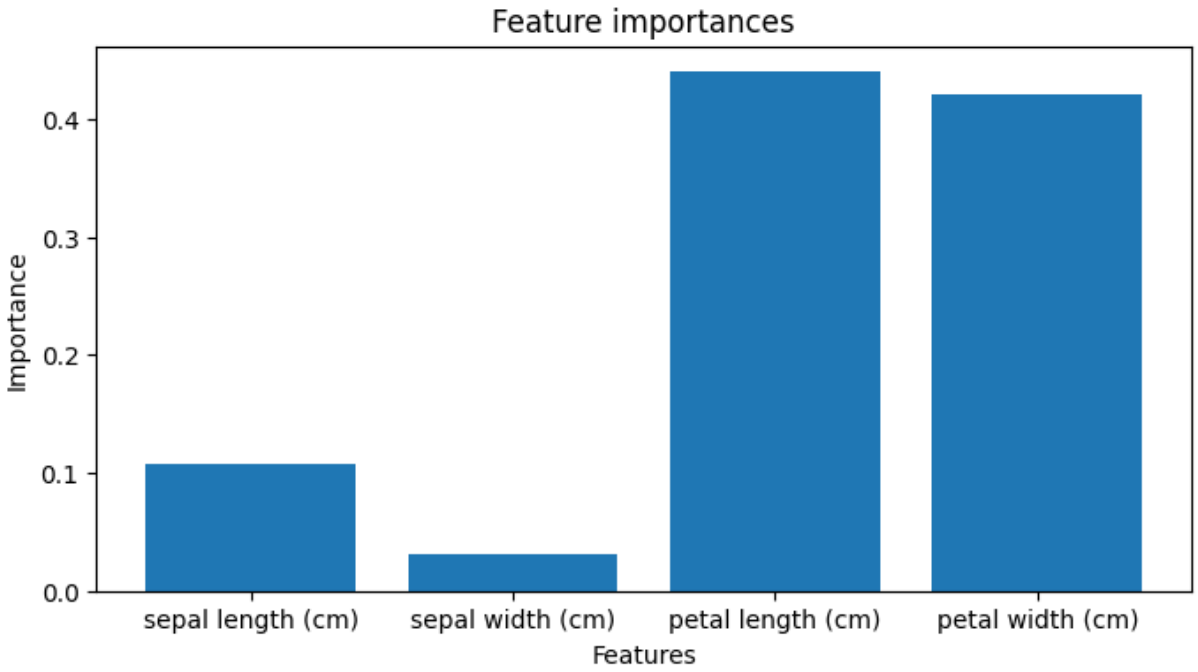
	Name	Age	Address	Qualification	Mobile No	Salary
0	Jai	27	Delhi	Msc	97.0	NaN
1	Princi	24	Kanpur	MA	91.0	NaN
2	Gaurav	22	Allahabad	MCA	58.0	NaN
3	Anuj	32	Kannauj	Phd	76.0	NaN
2	Gaurav	22	Kanpur	MCA	NaN	1000.0
3	Anuj	31	Allahabad	Phd	NaN	2000.0
6	Dhiraj	12	Kannuaj	Bcom	NaN	3000.0
7	Hitesh	52	Delhi	B.hons	NaN	4000.0

In [23]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
display(f"Accuracy: {accuracy:.2f}")
display("Confusion matrix:", conf_matrix)
display("Classification report:", class_report)
feature_importances = clf.feature_importances_
plt.figure(figsize=(8, 4))
plt.bar(
    range(len(iris.feature_names)), feature_importances, tick_label=iris.feature_na
)
plt.title("Feature importances")
plt.xlabel("Features")
plt.ylabel("Importance")
plt.show()
```

```
'Accuracy: 1.00'
```

```
'Confusion matrix:'
array([[10,  0,  0],
       [ 0,  9,  0],
       [ 0,  0, 11]], dtype=int64)
'Classification report:'
'              precision    recall  f1-score   support\n\n           0       1.00
1.00      1.00        10\n           1       1.00      1.00      1.00         9\n
2       1.00      1.00      1.00        11\n\n    accuracy
1.00        30\n   macro avg       1.00      1.00      1.00        30\nweighted avg
1.00      1.00      1.00        30\n'
```



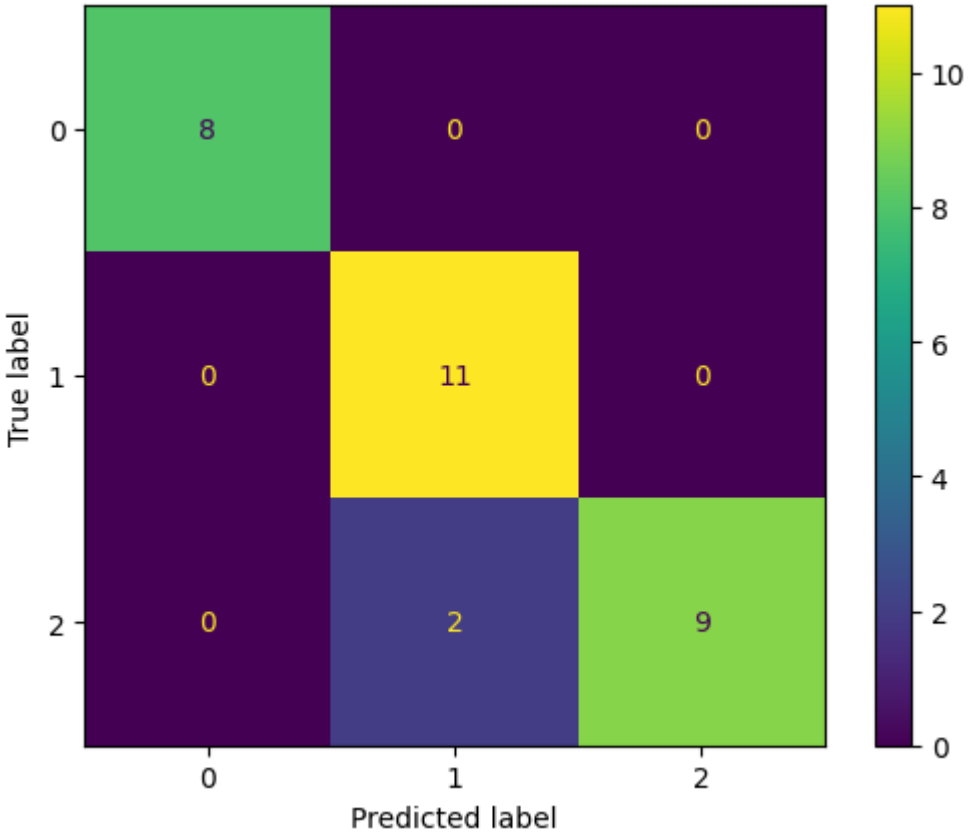
2.4

In [24]:

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn import metrics

iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=20
)
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

```
print("Precision:", precision_score(y_test, y_pred, average="weighted"))
print("Recall:", recall_score(y_test, y_pred, average="weighted"))
print("F1 score:", f1_score(y_test, y_pred, average="weighted"))

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)
cm_display = metrics.ConfusionMatrixDisplay(
    confusion_matrix=confusion_matrix, display_labels=[0, 1, 2]
)
cm_display.plot()
plt.show()
```

```
Accuracy: 0.9333333333333333
Precision: 0.9435897435897436
Recall: 0.9333333333333333
F1 score: 0.9327777777777778
```



In [25]:

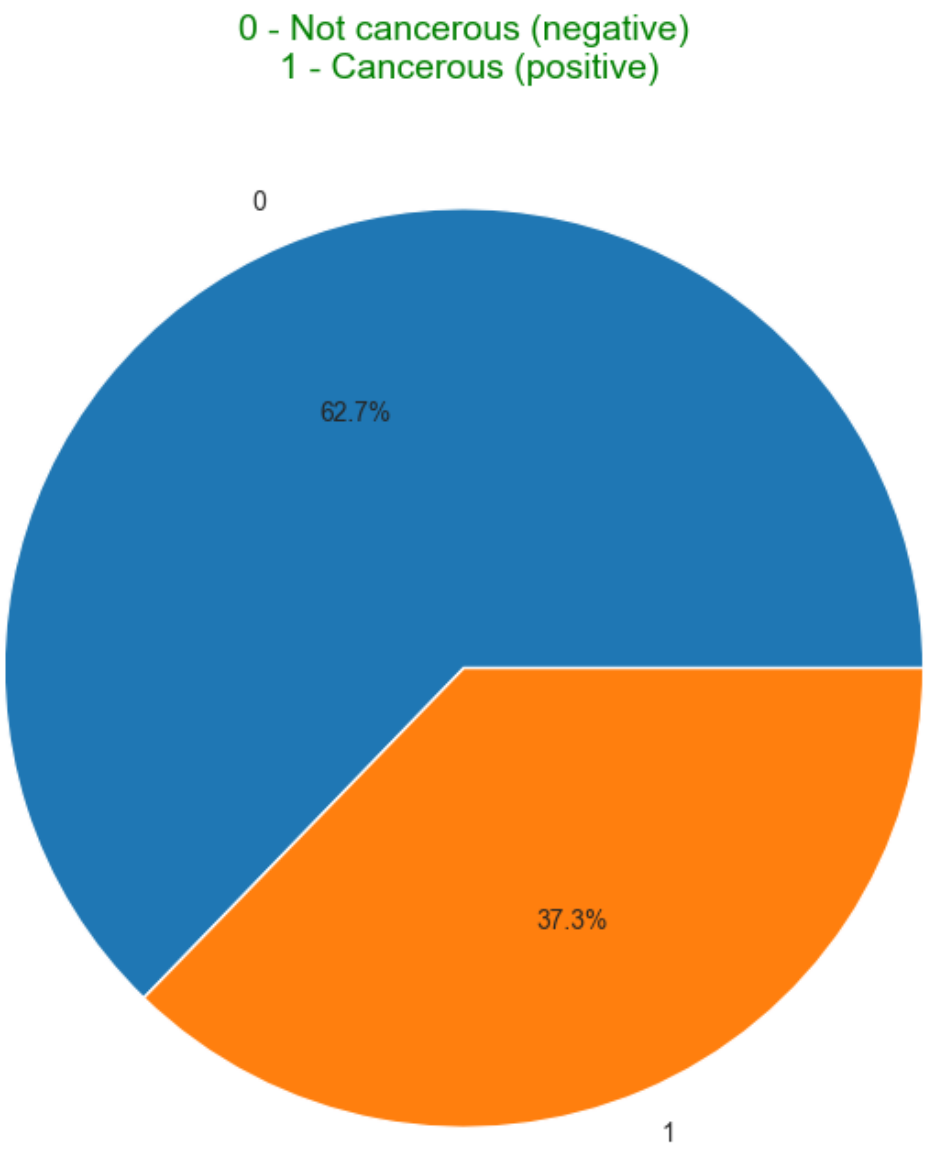
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import (
    mean_absolute_error,
    mean_squared_error,
    mean_absolute_percentage_error,
)

df = pd.read_csv("data/weather.csv")
X = df.iloc[:, 2].values
y = df.iloc[:, 3].values
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=20
```

```
)

X_train = X_train.reshape(-1, 1)
X_test = X_test.reshape(-1, 1)
regression = LinearRegression()
regression.fit(X_train, y_train)
y_pred = regression.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error:", mae)

rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error:", rmse)

mape = mean_absolute_percentage_error(
    y_test, y_pred, sample_weight=None, multioutput="uniform_average"
)
print("Mean Absolute Percentage Error:", mape)
```

```
Mean Absolute Error: 2.2312541290054972
Root Mean Squared Error: 2.627130808108865
Mean Absolute Percentage Error: 1.0589920324230513
```

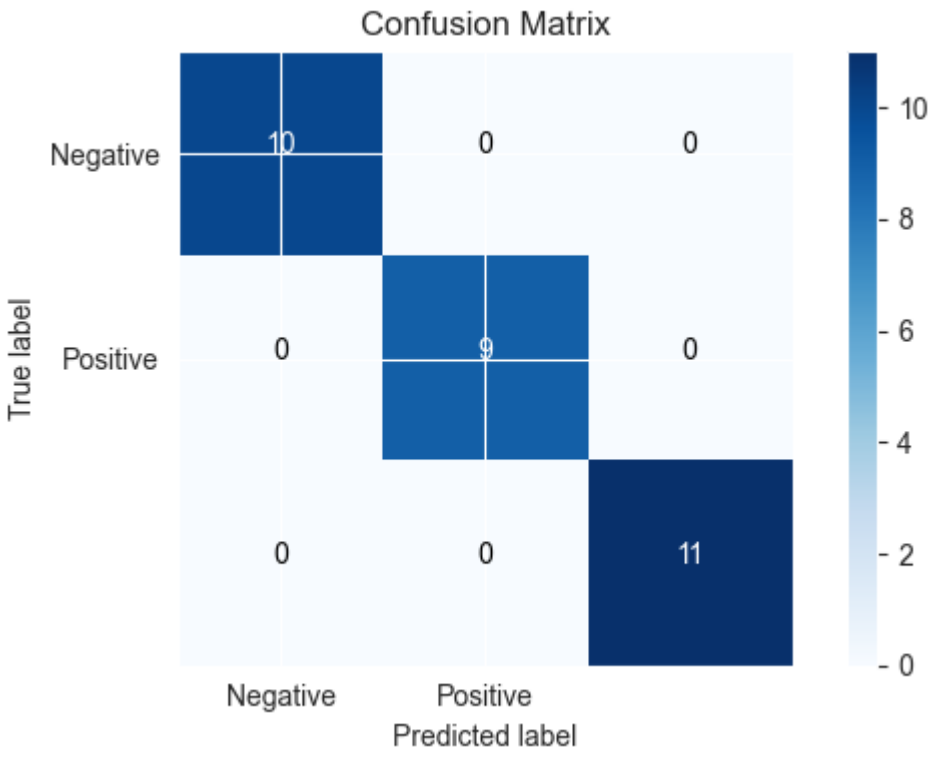
In [26]:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("darkgrid")

cancer = load_breast_cancer()

X = pd.DataFrame(cancer["data"], columns=cancer["feature_names"])
y = abs(pd.Series(cancer["target"]) - 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

model = RandomForestClassifier(random_state=1)
model.fit(X_train, y_train)
preds = model.predict(X_test)

plt.figure(figsize=(7, 7))
y.value_counts().plot.pie(autopct="%0.1f%%", ylabel=" ")
plt.title(f"0 - Not cancerous (negative)\n 1 - Cancerous (positive)", size=14, c="g
plt.tight_layout()
plt.show()
```



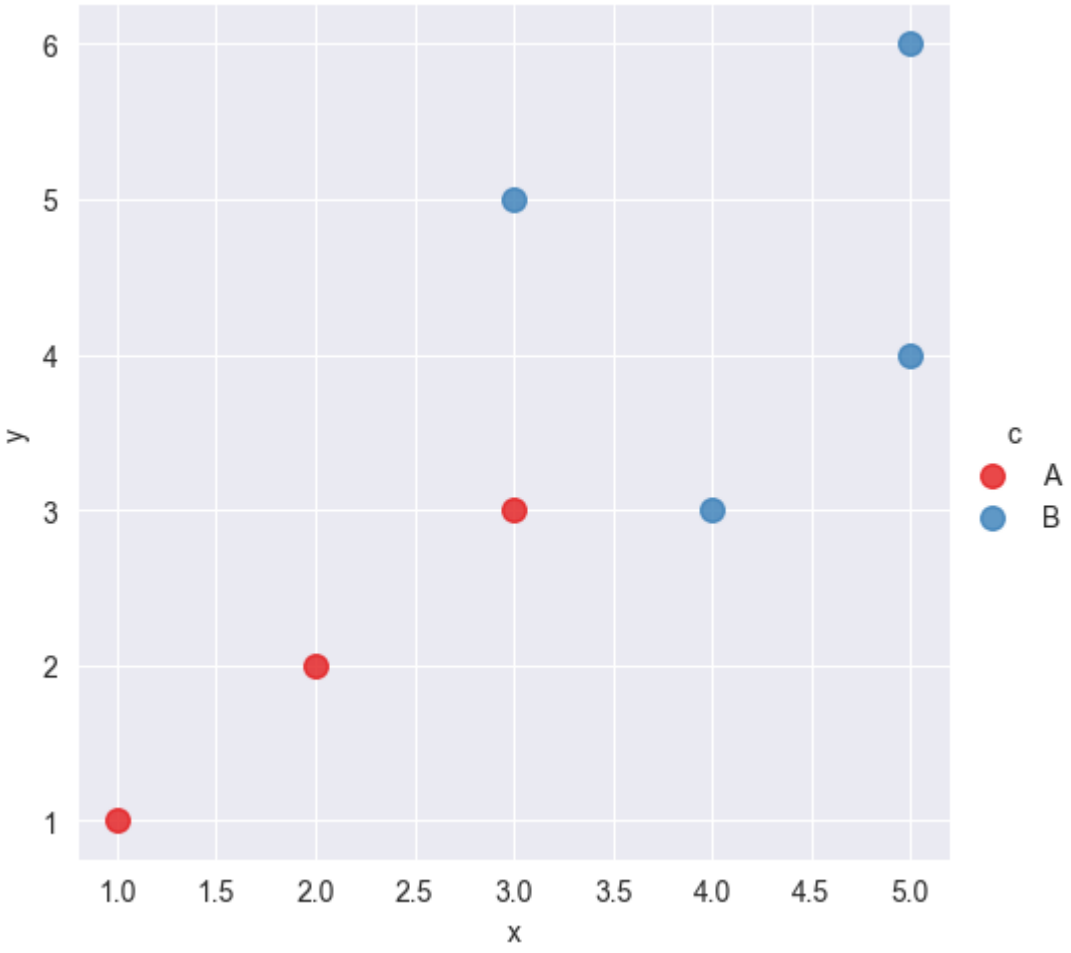
In [27]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import itertools

iris = load_iris()
X = iris.data
y = iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

clf = RandomForestClassifier(n_estimators=100, random_state=42)

clf.fit(X_train, y_train)
```

```
y_pred = clf.predict(X_test)

conf_matrix = confusion_matrix(y_test, y_pred)

class_labels = ["Negative", "Positive"]

plt.figure(figsize=(6, 4))
plt.imshow(conf_matrix, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.colorbar()
tick_marks = range(len(class_labels))
plt.xticks(tick_marks, class_labels)
plt.yticks(tick_marks, class_labels)

fmt = "d"
thresh = conf_matrix.max() / 2.0
for i, j in itertools.product(range(conf_matrix.shape[0]), range(conf_matrix.shape[
    plt.text(
        j,
        i,
        format(conf_matrix[i, j], fmt),
        horizontalalignment="center",
        color="white" if conf_matrix[i, j] > thresh else "black",
    )

plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.tight_layout()
plt.show()
```



In [28]:
```
import pandas as pd
import numpy as np
import operator
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv("data/knn.csv")
sns.lmplot(
    x="x",
    y="y",
    data=data,
    hue="c",
    palette="Set1",
    fit_reg=False,
    scatter_kws={"s": 70},
)
plt.show()
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\seaborn\axisgrid.py:118: User
Warning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



In [29]:
```
# Calculating the Distance between the points
def euclidean_distance(x1, x2, dimension):
    distance = 0
    for x in range(dimension):
```

```
        distance += np.square(x1[x] - x2[x])
    return np.sqrt(distance)


# Implementing the KNN model
def knn(training_points, test_point, k):
    distance = {}
    dimension = test_point.shape[1]

    for x in range(len(training_points)):
        dist = euclidean_distance(test_point, training_points.iloc[x], dimension)
        distance[x] = dist[0]

    sorted_d = sorted(distance.items(), key=operator.itemgetter(1))

    neighbors = []

    for x in range(k):
        neighbors.append(sorted_d[x][0])

    class_counter = {}

    for x in range(len(neighbors)):
        cls = training_points.iloc[neighbors[x]][-1]
        if cls in class_counter:
            class_counter[cls] += 1
        else:
            class_counter[cls] = 1

    sorted_counter = sorted(
        class_counter.items(), key=operator.itemgetter(1), reverse=True
    )

    return sorted_counter[0][0], neighbors
```

In [30]:
```
# Making Predictions
import pandas as pd
import numpy as np

test_set = [[3, 3.9]]
test = pd.DataFrame(test_set)

cls, neighbors = knn(data, test, 5)

print("Predicted Class: ", cls)
print("Nearest Neighbors: ", str(neighbors))
```

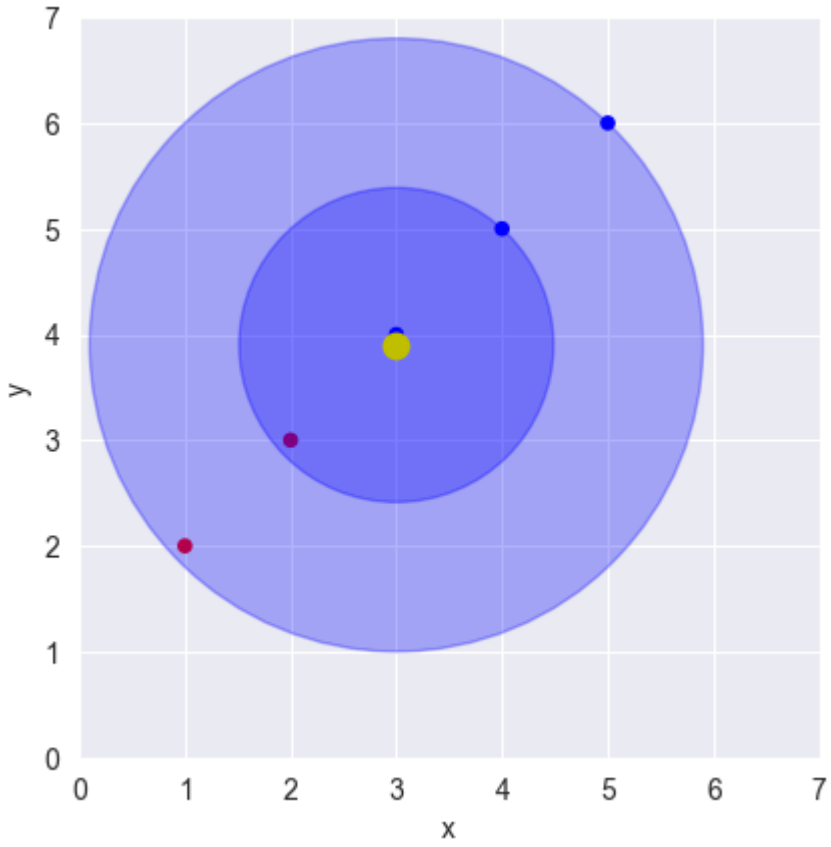
```
Predicted Class:  B
Nearest Neighbors:  [3, 4, 2, 6, 1]
```

2.5

In [31]:
```
import pandas as pd
import numpy as np
```

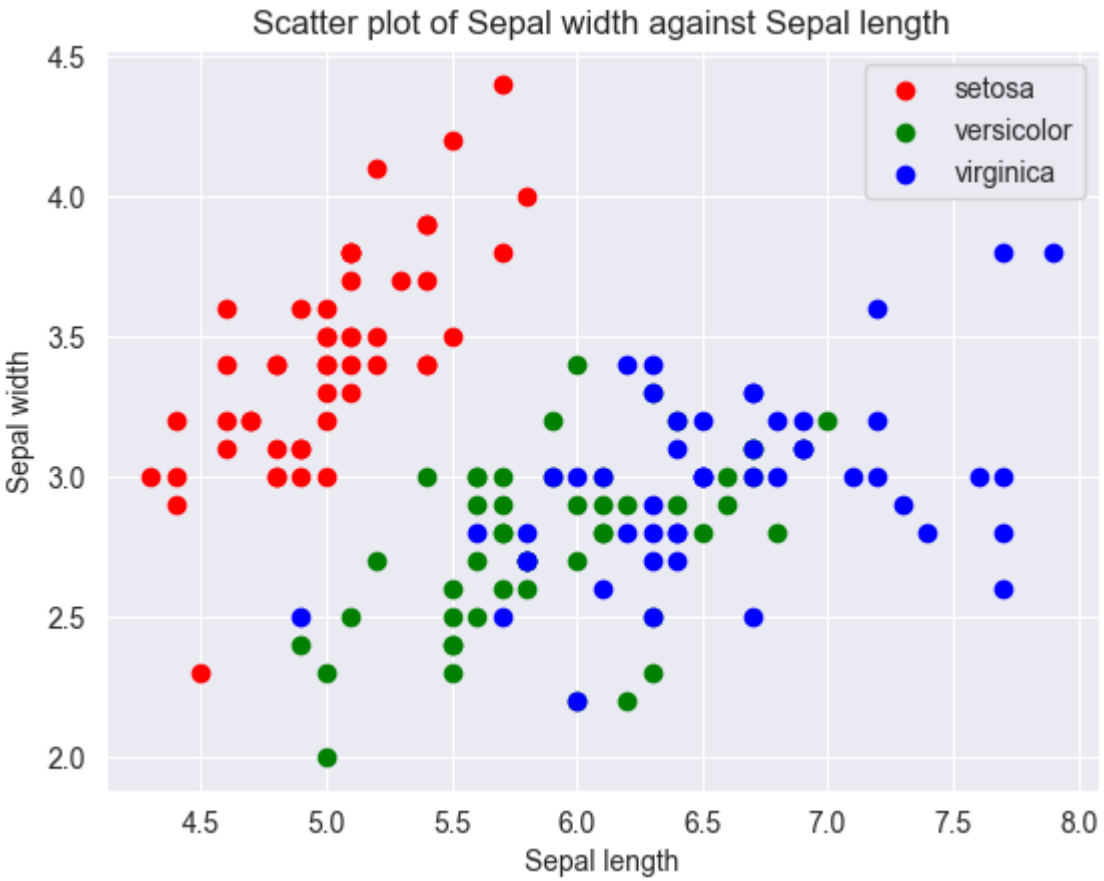
```
import matplotlib.pyplot as plt

def euclidean_distance(x1, x2, dimension):
    distance = 0
    for x in range(dimension):
        distance += np.square(x1[x] - x2[x])
    return np.sqrt(distance)

data = pd.DataFrame({
  'x': [1, 2, 3, 4, 5],
  'y': [2, 3, 4, 5, 6],
  'c': ['A', 'A', 'B', 'B', 'B']
})

test_set = [[3, 3.9]]
test = pd.DataFrame(test_set)

colors = ['r' if i == 'A' else 'b' for i in data['c']]

ax = data.plot(kind='scatter', x='x', y='y', color=colors)
plt.xlim(0, 7)
plt.ylim(0, 7)

plt.plot(test_set[0][0], test_set[0][1], "yo", markersize=9)

for k in range(7, 0, -2):  # You may consider using odd values like [7, 5, 3, 1] fo
    if k < 7:
        # Call your KNN function to classify the test data point
        cls, neighbors = knn(data, test, k)
        print("====================================")
        print("K =", k)
        print("Predicted Class:", cls)
        print("Nearest Neighbors:", str(neighbors))
        print(data.iloc[neighbors])
        print("====================================")
        furthest_point = data.iloc[neighbors].tail(1)
        radius = euclidean_distance(test.values[0], furthest_point.values[0], 2)
        circle_color = 'r' if cls == 'A' else 'b'

        circle = plt.Circle((test_set[0][0], test_set[0][1]), radius, color=circle_
        ax.add_patch(circle)

plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```

```
====================================
K = 5
Predicted Class: B
Nearest Neighbors: [2, 1, 3, 0, 4]
   x  y  c
2  3  4  B
1  2  3  A
3  4  5  B
0  1  2  A
4  5  6  B
====================================
====================================
K = 3
Predicted Class: B
Nearest Neighbors: [2, 1, 3]
   x  y  c
2  3  4  B
1  2  3  A
3  4  5  B
====================================
====================================
K = 1
Predicted Class: B
Nearest Neighbors: [2]
   x  y  c
2  3  4  B
====================================
```



In [32]:
```
# Using Scikit-learn's KNN Classifier
%matplotlib inline
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from sklearn import datasets, svm

iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target

colors = ['r', 'g', 'b']
for color, i, target in zip(colors, [0, 1, 2], iris.target_names):
    plt.scatter(X[y == i, 0], X[y == i, 1], color=color, label=target)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('Scatter plot of Sepal width against Sepal length')
plt.show()
```

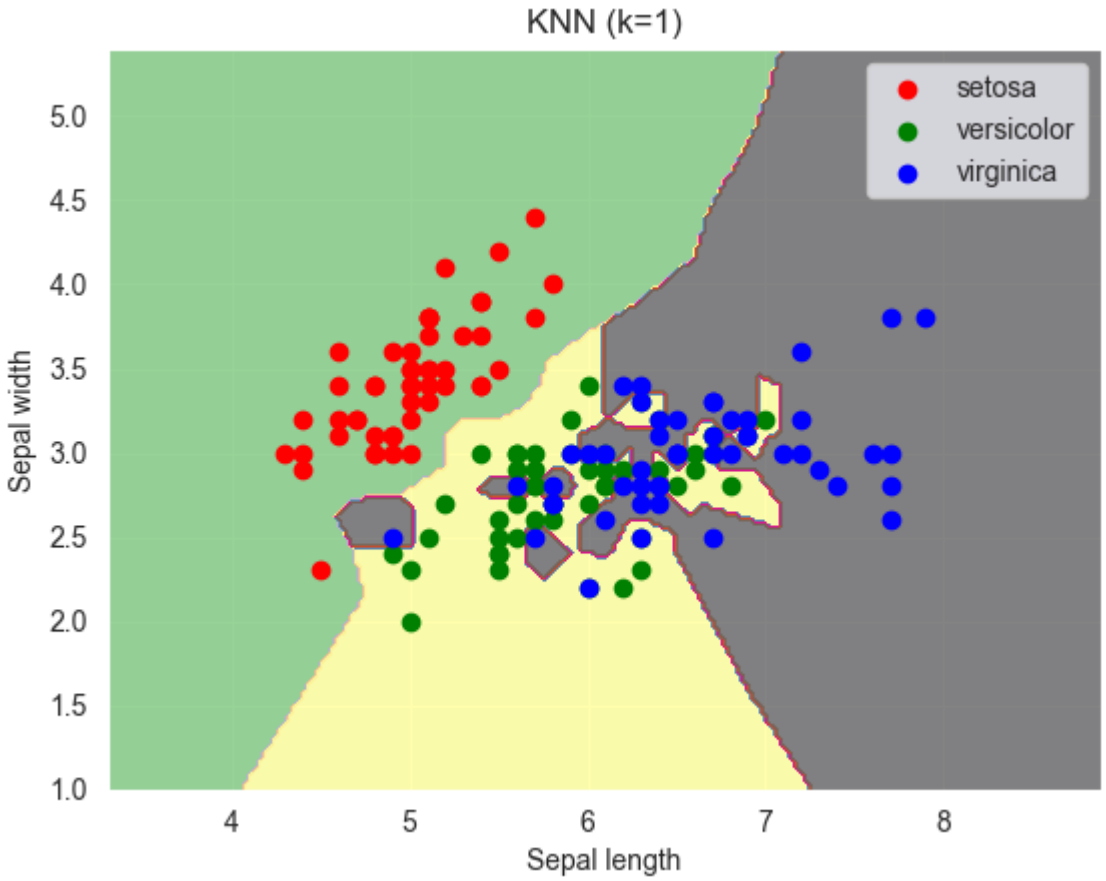


In [33]:

```
from sklearn.neighbors import KNeighborsClassifier
k = 1
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X, y)
X_min, X_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (X_max / X_min) / 100
xx, yy = np.meshgrid(np.arange(X_min, X_max, h), np.arange(y_min, y_max, h))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
plt.contourf(xx, yy, Z, cmap=plt.cm.Accent, alpha=0.8)
colors = ['r', 'g', 'b']
for color, i, target in zip(colors, [0, 1, 2], iris.target_names):
    plt.scatter(X[y == i, 0], X[y == i, 1], color=color, label=target)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title(f'KNN (k={k})')
plt.legend(loc='best', shadow=False, scatterpoints=1)
predictions = knn.predict(X)
print(np.unique(predictions, return_counts=True))
```

```
(array([0, 1, 2]), array([50, 60, 40], dtype=int64))
```



In [34]:

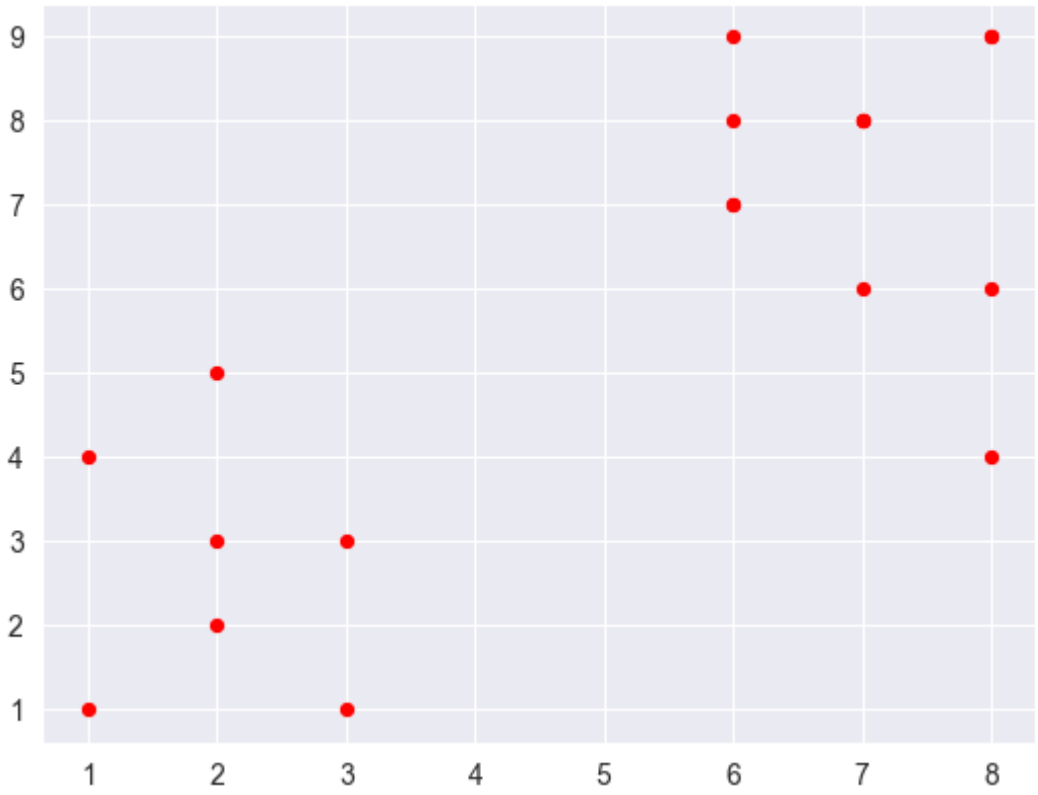
```
# Parameter Tuning K
from sklearn.model_selection import cross_val_score
cv_scores = []
X = iris.data[:, :4]
y = iris.target
folds = 10
ks = list(range(1, int(len(X) * ((folds - 1) / folds))))
ks = [k for k in ks if k % 3 != 0]
for k in ks:
  knn = KNeighborsClassifier(n_neighbors=k)
  scores = cross_val_score(knn, X, y, cv=folds, scoring='accuracy')
  mean = scores.mean()
  cv_scores.append(scores.mean())
  print(k, mean)
```

1 0.96
2 0.9533333333333334
4 0.9666666666666666
5 0.9666666666666668
7 0.9666666666666668
8 0.9666666666666668
10 0.9666666666666668
11 0.9666666666666668
13 0.9800000000000001
14 0.9733333333333334
16 0.9733333333333334
17 0.9733333333333334
19 0.9733333333333334
20 0.9800000000000001
22 0.9666666666666666
23 0.9733333333333334
25 0.9666666666666666
26 0.96
28 0.9533333333333334
29 0.9533333333333334
31 0.9466666666666667
32 0.9466666666666667
34 0.9466666666666667
35 0.9466666666666667
37 0.9466666666666667
38 0.9466666666666667
40 0.9533333333333334
41 0.9533333333333334
43 0.9466666666666667
44 0.9400000000000001
46 0.9333333333333333
47 0.9333333333333333
49 0.9400000000000001
50 0.9266666666666667
52 0.9333333333333333
53 0.9333333333333333
55 0.9333333333333333
56 0.9066666666666666
58 0.9133333333333334
59 0.9200000000000002
61 0.9199999999999999
62 0.9066666666666666
64 0.9
65 0.9
67 0.8866666666666667
68 0.8800000000000001
70 0.8866666666666667
71 0.8866666666666667
73 0.8933333333333333
74 0.8866666666666667
76 0.8800000000000001
77 0.8866666666666667
79 0.8866666666666667
80 0.8933333333333333
82 0.9000000000000001
83 0.8800000000000001

```
85 0.8733333333333334
86 0.8800000000000001
88 0.8733333333333334
89 0.8800000000000001
91 0.6599999999999999
92 0.6599999999999999
94 0.6599999999999999
95 0.6599999999999999
97 0.6599999999999999
98 0.6599999999999999
100 0.6599999999999999
101 0.6599999999999999
103 0.6599999999999999
104 0.6599999999999999
106 0.6599999999999999
107 0.6599999999999999
109 0.6599999999999999
110 0.6533333333333332
112 0.6466666666666665
113 0.6466666666666665
115 0.6466666666666665
116 0.6466666666666665
118 0.6399999999999999
119 0.6399999999999999
121 0.6399999999999999
122 0.6399999999999999
124 0.6399999999999999
125 0.6333333333333332
127 0.6266666666666666
128 0.6199999999999999
130 0.6066666666666667
131 0.5933333333333332
133 0.5666666666666667
134 0.5533333333333333
```

In [35]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data/kmeans.csv')
plt.scatter(df['x'], df['y'], c='r', s=18)
```

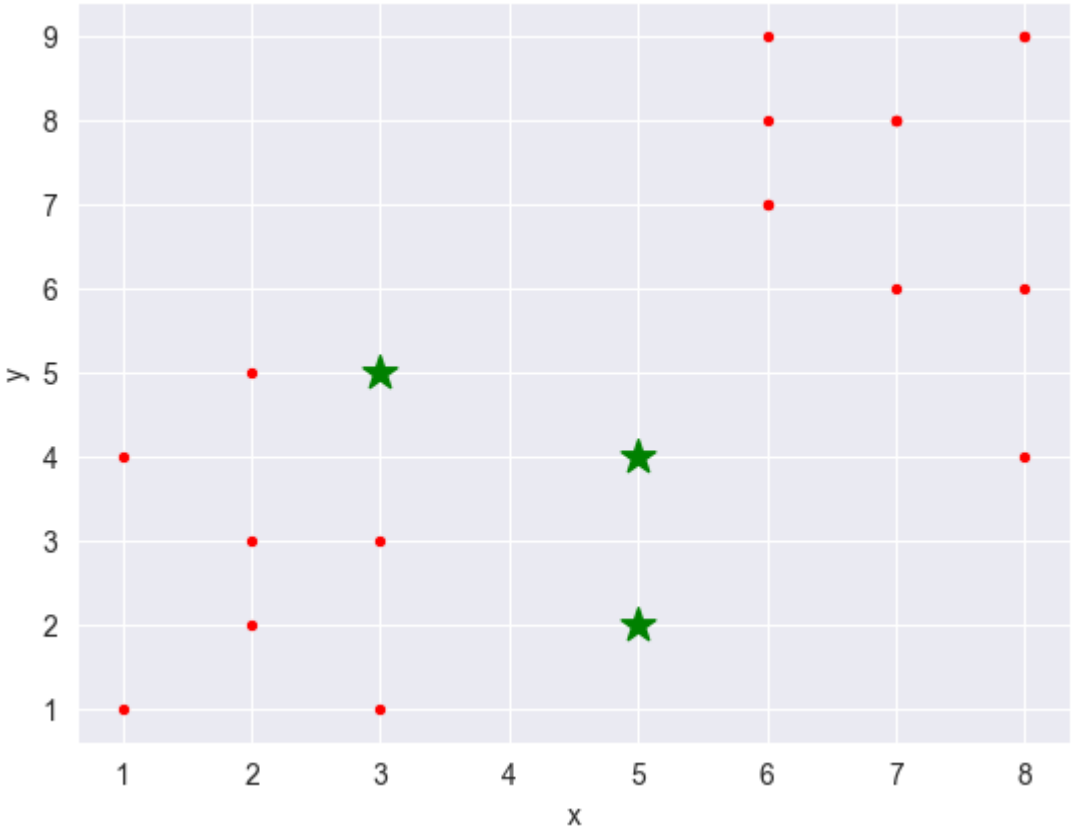
Out[35]:
```
<matplotlib.collections.PathCollection at 0x17f42475d90>
```



In [39]:
```
k = 3

X = np.array(list(zip(df['x'], df['y'])))

Cx = np.random.randint(np.min(X[:,0]), np.max(X[:,0]), size = k)
Cy = np.random.randint(np.min(X[:,1]), np.max(X[:,1]), size = k)

C = np.array(list(zip(Cx, Cy)), dtype=np.float64)
print(C)

plt.scatter(df['x'], df['y'], c='r', s=8)
plt.scatter(Cx, Cy, marker='*', c='g', s=160)
plt.xlabel("x")
plt.ylabel("y")
```

```
[[5. 4.]
 [5. 2.]
 [3. 5.]]
```

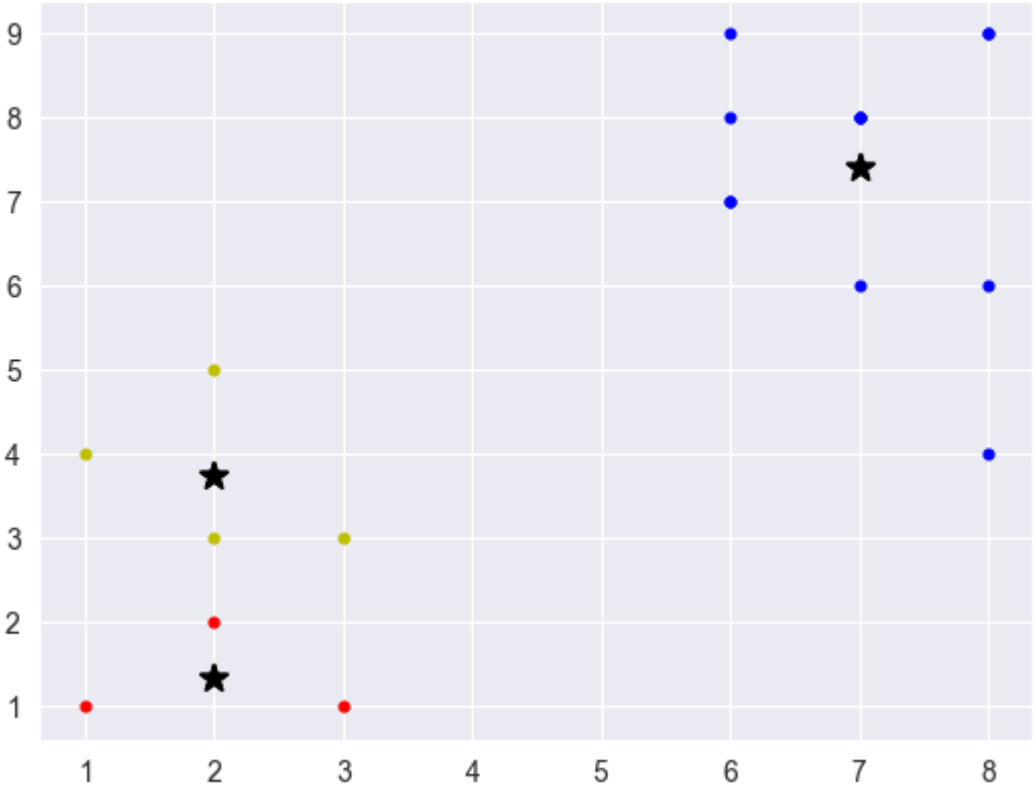
Out[39]: Text(0, 0.5, 'y')



In [40]:

```
from copy import deepcopy
def euclidean_distance(a, b, ax=1):
    return np.linalg.norm(a - b, axis=ax)
C_prev = np.zeros(C.shape)
clusters = np.zeros(len(X))
distance_differences = euclidean_distance(C, C_prev)
while distance_differences.any() != 0:
    for i in range(len(X)):
        distances = euclidean_distance(X[i], C)
        cluster = np.argmin(distances)
        clusters[i] = cluster
    C_prev = deepcopy(C)
    for i in range(k):
        points = [X[j] for j in range(len(X)) if clusters[j] == i]
        if len(points) != 0:
            C[i] = np.mean(points, axis=0)
    distance_differences = euclidean_distance(C, C_prev)
colors = ["b", "r", "y", "g", "c", "m"]
for i in range(k):
    points = np.array([X[j] for j in range(len(X)) if clusters[j] == i])
    if len(points) > 0:
        plt.scatter(points[:, 0], points[:, 1], s=10, c=colors[i])
    else:
        print("Please regenerate your centroids again.")
    plt.scatter(points[:, 0], points[:, 1], s=10, c=colors[i])
    plt.scatter(C[:, 0], C[:, 1], marker="*", s=100, c="black")
for i, cluster in enumerate(clusters):
    print("Point " + str(X[i]), "Cluster " + str(int(cluster)))
```

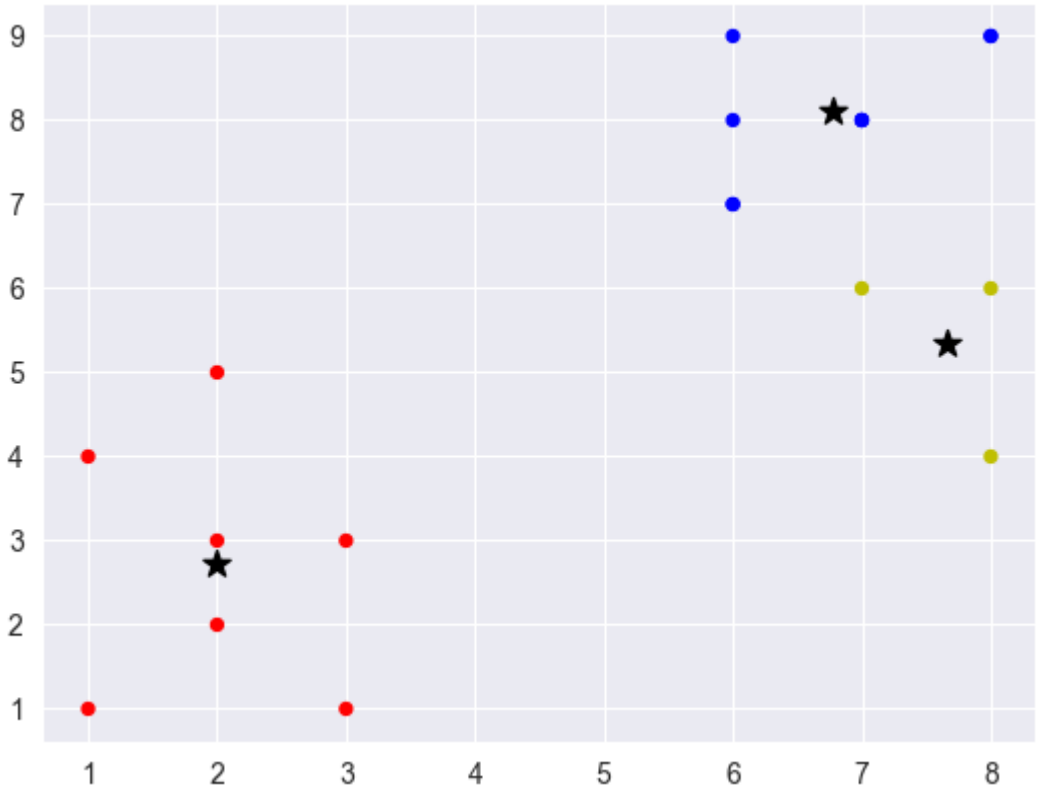
```
Point [1 1] Cluster 1
Point [2 2] Cluster 1
Point [2 3] Cluster 2
Point [1 4] Cluster 2
Point [3 3] Cluster 2
Point [6 7] Cluster 0
Point [7 8] Cluster 0
Point [6 8] Cluster 0
Point [7 6] Cluster 0
Point [6 9] Cluster 0
Point [2 5] Cluster 2
Point [7 8] Cluster 0
Point [8 9] Cluster 0
Point [6 7] Cluster 0
Point [7 8] Cluster 0
Point [3 1] Cluster 1
Point [8 4] Cluster 0
Point [8 6] Cluster 0
Point [8 9] Cluster 0
```



In [ ]:
```python
from sklearn.cluster import KMeans
k = 3
kmeans = KMeans(n_clusters=k)
kmeans = kmeans.fit(X)
labels = kmeans.predict(X)
centroids = kmeans.cluster_centers_
print(labels)
print(centroids)
c = ["b", "r", "y", "g", "c", "m"]
colors = [c[i] for i in labels]
plt.scatter(df["x"], df["y"], c=colors, s=18)
plt.scatter(centroids[:, 0], centroids[:, 1], marker="*", s=100, c="black")
```

```
cluster = kmeans.predict([[3, 4]])[0]
print(c[cluster])  # r
cluster = kmeans.predict([[7, 5]])[0]
print(c[cluster])  # y
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
[1 1 1 1 1 0 0 0 2 0 1 0 0 0 0 1 2 2 0]
[[6.77777778 8.11111111]
 [2.         2.71428571]
 [7.66666667 5.33333333]]
r
y
```



In [ ]:
```
from sklearn import metrics

silhouette_samples = metrics.silhouette_samples(X, kmeans.labels_)
print(silhouette_samples)

print("Average of Silhouette Coefficients for k =", k)
print("==========================================")
print("Silhouette mean:", silhouette_samples.mean())

print("Silhouette mean:", metrics.silhouette_score(X, kmeans.labels_))
```

```
[0.67534567 0.73722797 0.73455072 0.66254937 0.6323039  0.33332111
 0.63792468 0.58821402 0.29141777 0.59137721 0.50802377 0.63792468
 0.52511161 0.33332111 0.63792468 0.60168807 0.51664787 0.42831295
 0.52511161]
Average of Silhouette Coefficients for k = 3
============================================
Silhouette mean: 0.5578051985195768
Silhouette mean: 0.5578051985195768
```

In [ ]:

```python
silhouette_avgs = []
min_k = 2

# ---try k from 2 to maximum number of labels---
for k in range(min_k, len(X)):
    kmean = KMeans(n_clusters=k).fit(X)
    score = metrics.silhouette_score(X, kmean.labels_)
    print("Silhouette Coefficients for k =", k, "is", score)
    silhouette_avgs.append(score)

f, ax = plt.subplots(figsize=(7, 5))
ax.plot(range(min_k, len(X)), silhouette_avgs)

plt.xlabel("Number of clusters")
plt.ylabel("Silhouette Coefficients")

# ---the optimal k is the one with the highest average silhouette---
Optimal_K = silhouette_avgs.index(max(silhouette_avgs)) + min_k
print("Optimal K is ", Optimal_K)
```

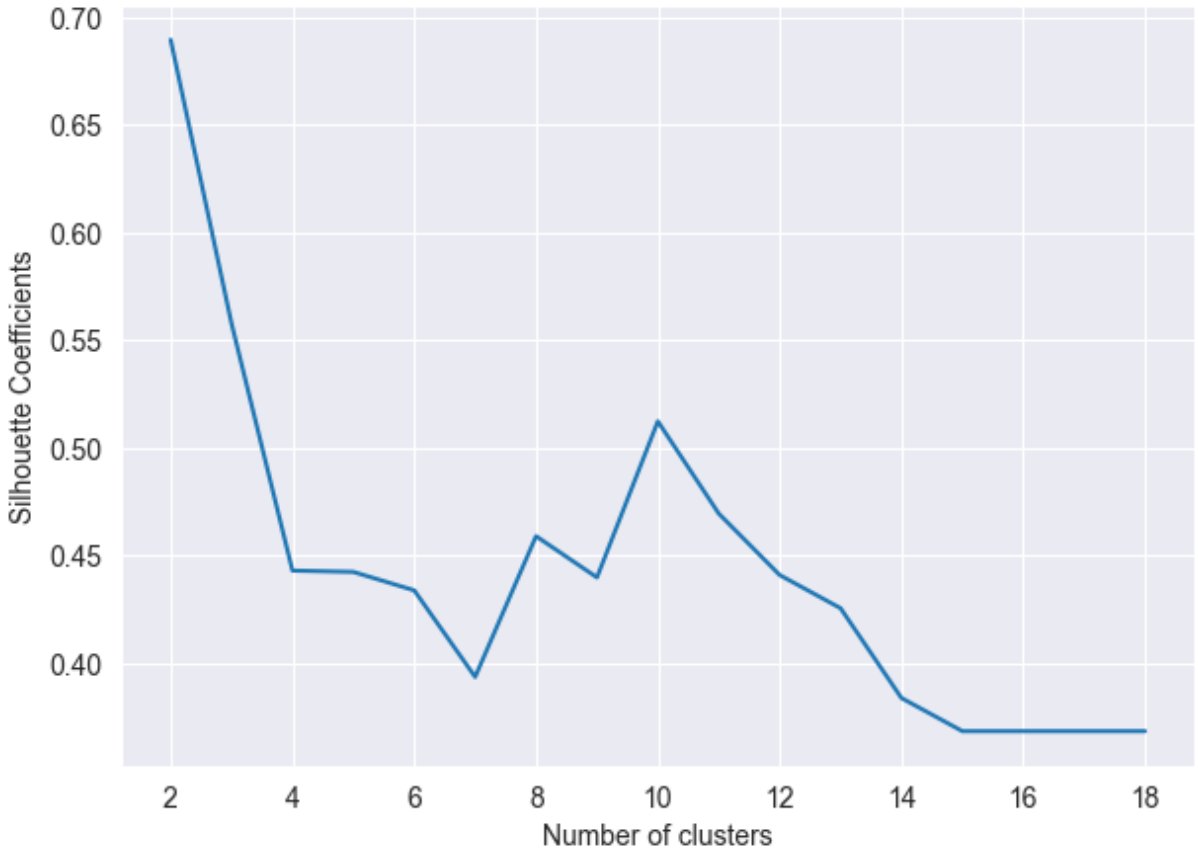
```
Silhouette Coefficients for k = 2 is 0.6897112069939448
Silhouette Coefficients for k = 3 is 0.5578051985195768
Silhouette Coefficients for k = 4 is 0.4430381814640289
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

```
Silhouette Coefficients for k = 5 is 0.4424248576948773
Silhouette Coefficients for k = 6 is 0.43385100789093656
Silhouette Coefficients for k = 7 is 0.3936180551723887
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 8 is 0.459039364508135
Silhouette Coefficients for k = 9 is 0.43985823083018905
Silhouette Coefficients for k = 10 is 0.5124113408422506
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 11 is 0.4695564671186216
Silhouette Coefficients for k = 12 is 0.4409831398126504
Silhouette Coefficients for k = 13 is 0.4255677072435213
Silhouette Coefficients for k = 14 is 0.383836485200708
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 15 is 0.3684210526315789
Silhouette Coefficients for k = 16 is 0.3684210526315789
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\base.py:1151: Converg
enceWarning: Number of distinct clusters (15) found smaller than n_clusters (16). Po
ssibly due to duplicate points in X.
  return fit_method(estimator, *args, **kwargs)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\base.py:1151: Converg
enceWarning: Number of distinct clusters (15) found smaller than n_clusters (17). Po
ssibly due to duplicate points in X.
  return fit_method(estimator, *args, **kwargs)
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 17 is 0.3684210526315789
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\base.py:1151: Converg
enceWarning: Number of distinct clusters (15) found smaller than n_clusters (18). Po
ssibly due to duplicate points in X.
  return fit_method(estimator, *args, **kwargs)
Silhouette Coefficients for k = 18 is 0.3684210526315789
Optimal K is  2
```



In [ ]:
```
%matplotlib inline
import numpy as np
import pandas as pd

df = pd.read_csv("data/BMX_G.csv")

print(df.shape)

df.isnull().sum()
```

```
(9338, 27)
```

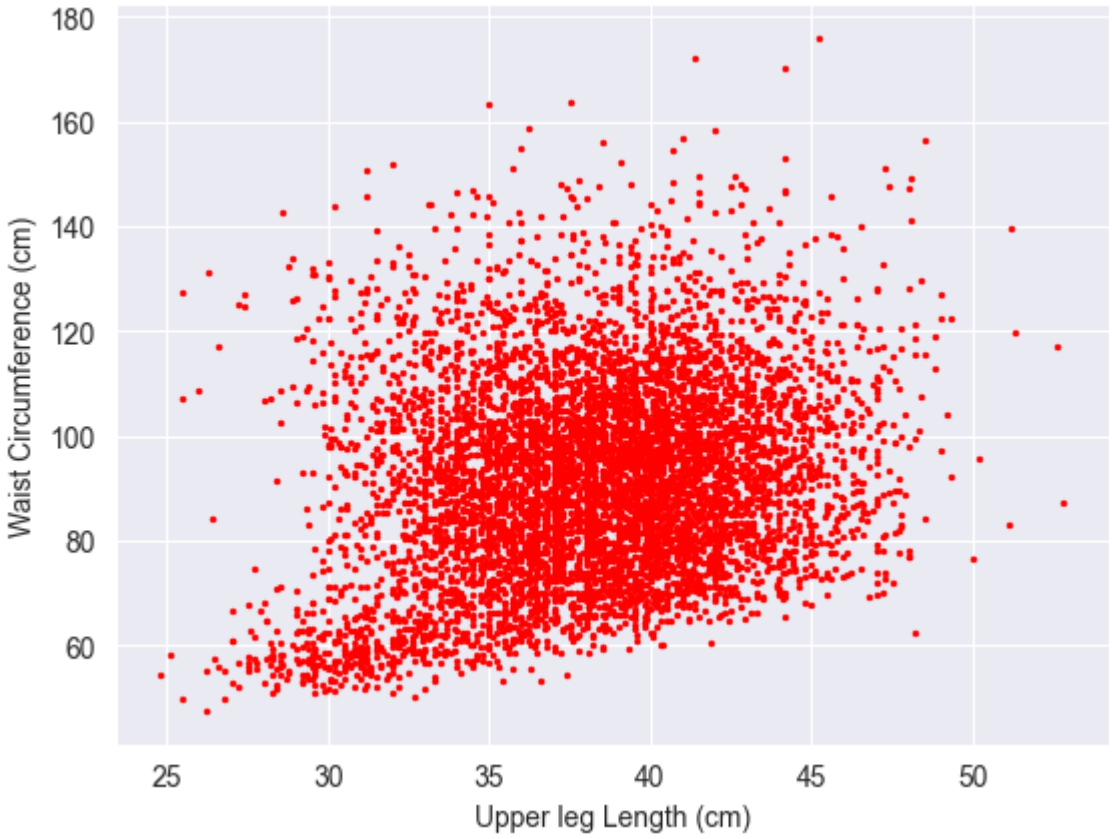
Out[ ]:
```
Unnamed: 0       0
seqn             0
bmdstats         0
bmxwt           95
bmiwt         8959
bmxrecum      8259
bmirecum      9307
bmxhead       9102
bmihead       9338
bmxht          723
bmiht         9070
bmxbmi         736
bmdbmic       5983
bmxleg        2383
bmileg        8984
bmxarml        512
bmiarml       8969
bmxarmc        512
bmiarmc       8965
bmxwaist      1134
bmiwaist      8882
bmxsad1       2543
bmxsad2       2543
bmxsad3       8940
bmxsad4       8940
bmdavsad      2543
bmdsadcm      8853
dtype: int64
```

In [ ]:
```
df = df.dropna(subset=['bmxleg','bmxwaist'])  # remove rows with NaNs
print(df.shape)
```

```
(6899, 27)
```

In [ ]:
```
import matplotlib.pyplot as plt

plt.scatter(df['bmxleg'],df['bmxwaist'], c='r', s=2)
plt.xlabel("Upper leg Length (cm)")
plt.ylabel("Waist Circumference (cm)")
```

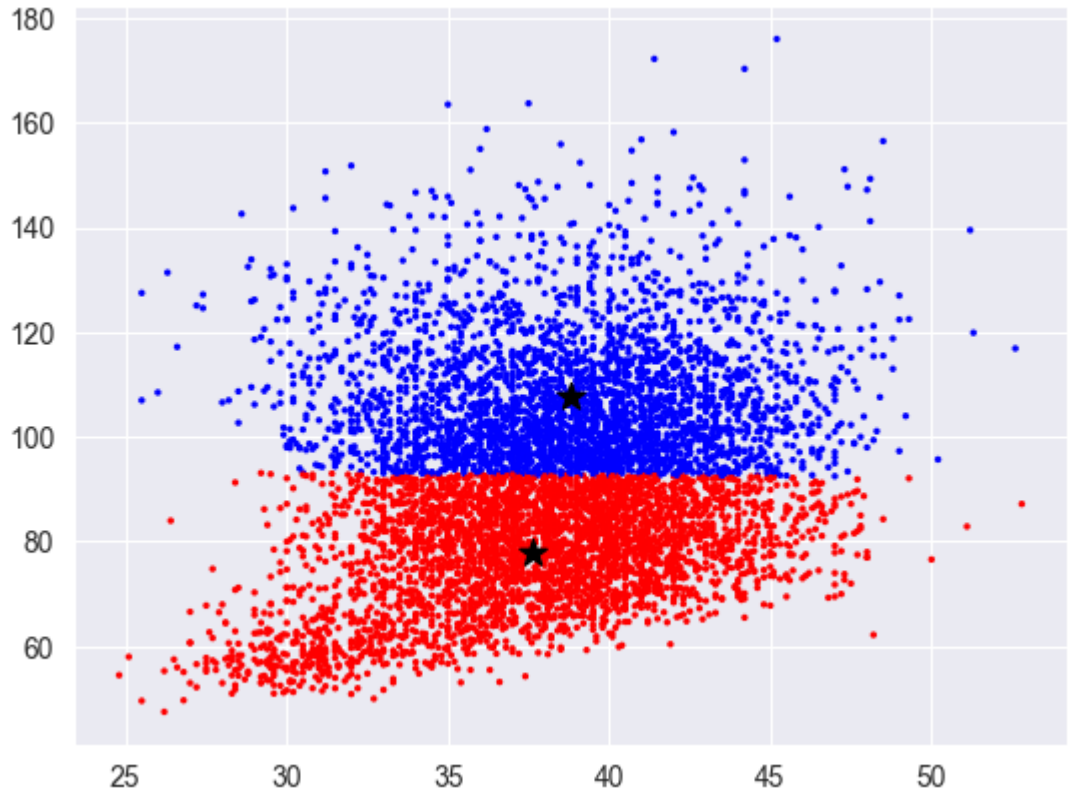
Out[ ]:
```
Text(0, 0.5, 'Waist Circumference (cm)')
```



In [ ]:
```
from sklearn.cluster import KMeans

k = 2
X = np.array(list(zip(df['bmxleg'],df['bmxwaist'])))

kmeans = KMeans(n_clusters=k)
kmeans = kmeans.fit(X)
labels = kmeans.predict(X)
centroids = kmeans.cluster_centers_

#---map the labels to colors---
c = ['b','r','y','g','c','m']
colors = [c[i] for i in labels]

plt.scatter(df['bmxleg'],df['bmxwaist'], c=colors, s=2)
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=100, c='black')
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[ ]:
```
<matplotlib.collections.PathCollection at 0x1c1c17b8e80>
```



In [ ]:
```
print(centroids)
```

```
[[ 38.82134278 107.78925162]
 [ 37.6453724   77.72853231]]
```

In [ ]:
```
from sklearn import metrics

silhouette_avgs = []
min_k = 2

#---try k from 2 to maximum number of labels---
for k in range(min_k, 10):
    kmean = KMeans(n_clusters=k).fit(X)
    score = metrics.silhouette_score(X, kmean.labels_)
    print("Silhouette Coefficients for k =", k, "is", score)
    silhouette_avgs.append(score)

#---the optimal k is the one with the highest average silhouette---
Optimal_K = silhouette_avgs.index(max(silhouette_avgs)) + min_k
print("Optimal K is", Optimal_K)
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 2 is 0.5165351076055341
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 3 is 0.47226905068761915
```

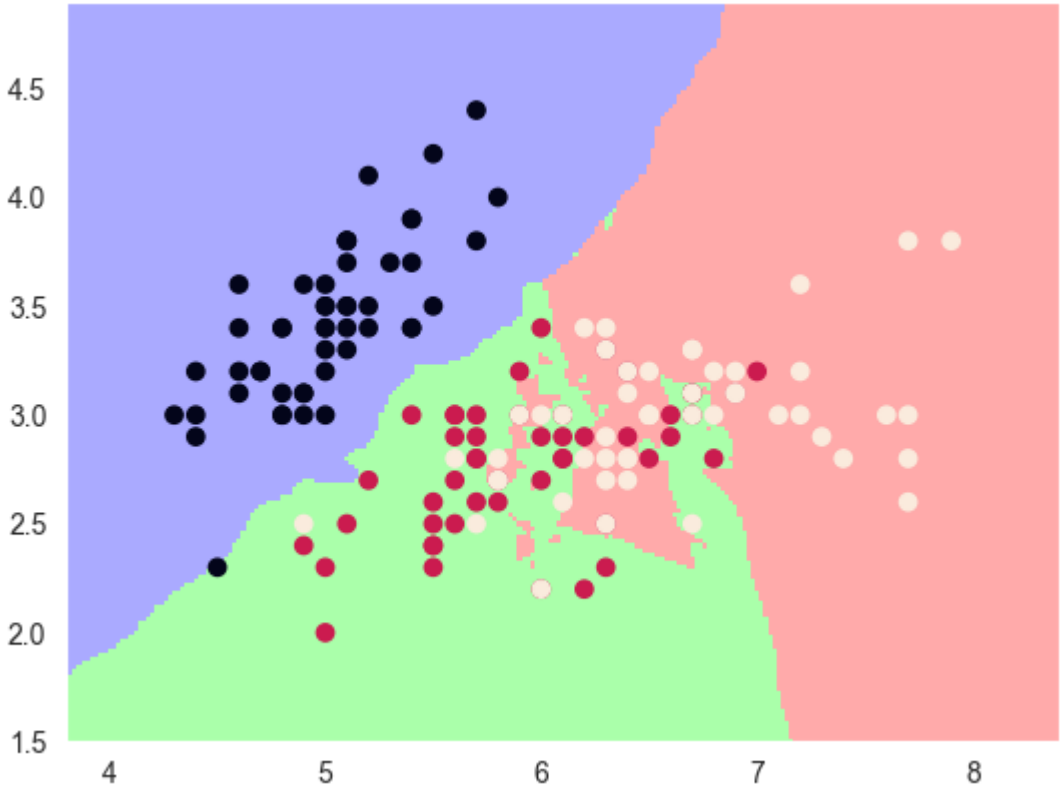
```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 4 is 0.43630749582817996
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 5 is 0.41912075493761086
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 6 is 0.39259265486319
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 7 is 0.3771085790364022
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 8 is 0.35709972188478173
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
Silhouette Coefficients for k = 9 is 0.3418641949563242
Optimal K is 2
```

2.6

In [42]:

```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier

# Tải dữ liệu Iris từ thư viện sklearn
iris = datasets.load_iris()

# Lấy dữ liệu của hai thuộc tính đầu tiên (chiều dài và chiều rộng của cánh hoa)
x = iris.data[:, :2]  # Trục X - chiều dài và chiều rộng của cánh hoa
y = iris.target  # Trục Y - loài hoa Iris

# Xác định giới hạn trục X và Y
x_min, x_max = x[:, 0].min() - 0.5, x[:, 0].max() + 0.5
y_min, y_max = x[:, 1].min() - 0.5, x[:, 1].max() + 0.5

# Tạo lưới (mesh)
```

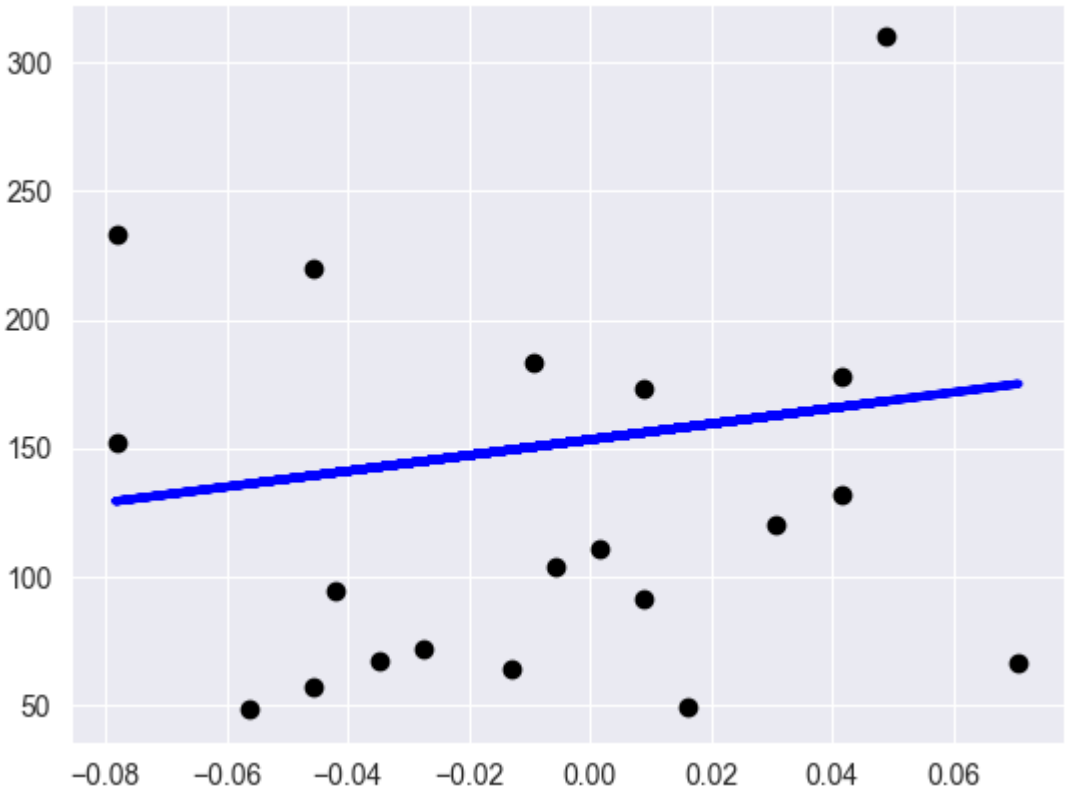
```
cmap_light = ListedColormap(['#AAAAFF', '#AAFFAA', '#FFAAAA'])
h = 0.02
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Tạo mô hình phân loại K-Nearest Neighbors (KNN) và huấn luyện mô hình
knn = KNeighborsClassifier()
knn.fit(x, y)

# Dự đoán lớp cho từng điểm trên lưới
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Tạo biểu đồ
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)  # Vẽ màu cho các khu vực dự đoán
plt.scatter(x[:, 0], x[:, 1], c=y)  # Vẽ các điểm dữ liệu huấn luyện
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

# Hiển thị biểu đồ
plt.show()
```



2.7

In [45]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import datasets
```

```
# Tải dữ liệu về bệnh tiểu đường từ sklearn
diabetes = datasets.load_diabetes()

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
x_train = diabetes.data[:-20]  # Dữ liệu huấn luyện (loại bỏ 20 mẫu cuối)
y_train = diabetes.target[:-20]  # Nhãn tương ứng với tập huấn luyện
x_test = diabetes.data[-20:]  # Dữ liệu kiểm tra (20 mẫu cuối)
y_test = diabetes.target[-20:]  # Nhãn tương ứng với tập kiểm tra

# Chọn chỉ một thuộc tính (chiều) của dữ liệu để thực hiện hồi quy tuyến tính
x0_test = x_test[:, 0]  # Lấy chỉ số 0 của dữ liệu kiểm tra
x0_train = x_train[:, 0]  # Lấy chỉ số 0 của dữ liệu huấn luyện

# Chuyển đổi x0_test và x0_train từ vector hàng thành vector cột
x0_test = x0_test[:, np.newaxis]
x0_train = x0_train[:, np.newaxis]

# Tạo một mô hình hồi quy tuyến tính
linreg = linear_model.LinearRegression()

# Huấn luyện mô hình trên dữ liệu huấn luyện (chỉ sử dụng một thuộc tính)
linreg.fit(x0_train, y_train)

# Dự đoán giá trị đầu ra trên dữ liệu kiểm tra (chỉ sử dụng một thuộc tính)
y = linreg.predict(x0_test)

# Vẽ biểu đồ để so sánh kết quả dự đoán với dữ liệu thực tế
plt.scatter(x0_test, y_test, color='k')  # Vẽ các điểm dữ liệu kiểm tra
plt.plot(x0_test, y, color='b', linewidth=3)  # Vẽ đường dự đoán của mô hình

# Hiển thị biểu đồ
plt.show()
```

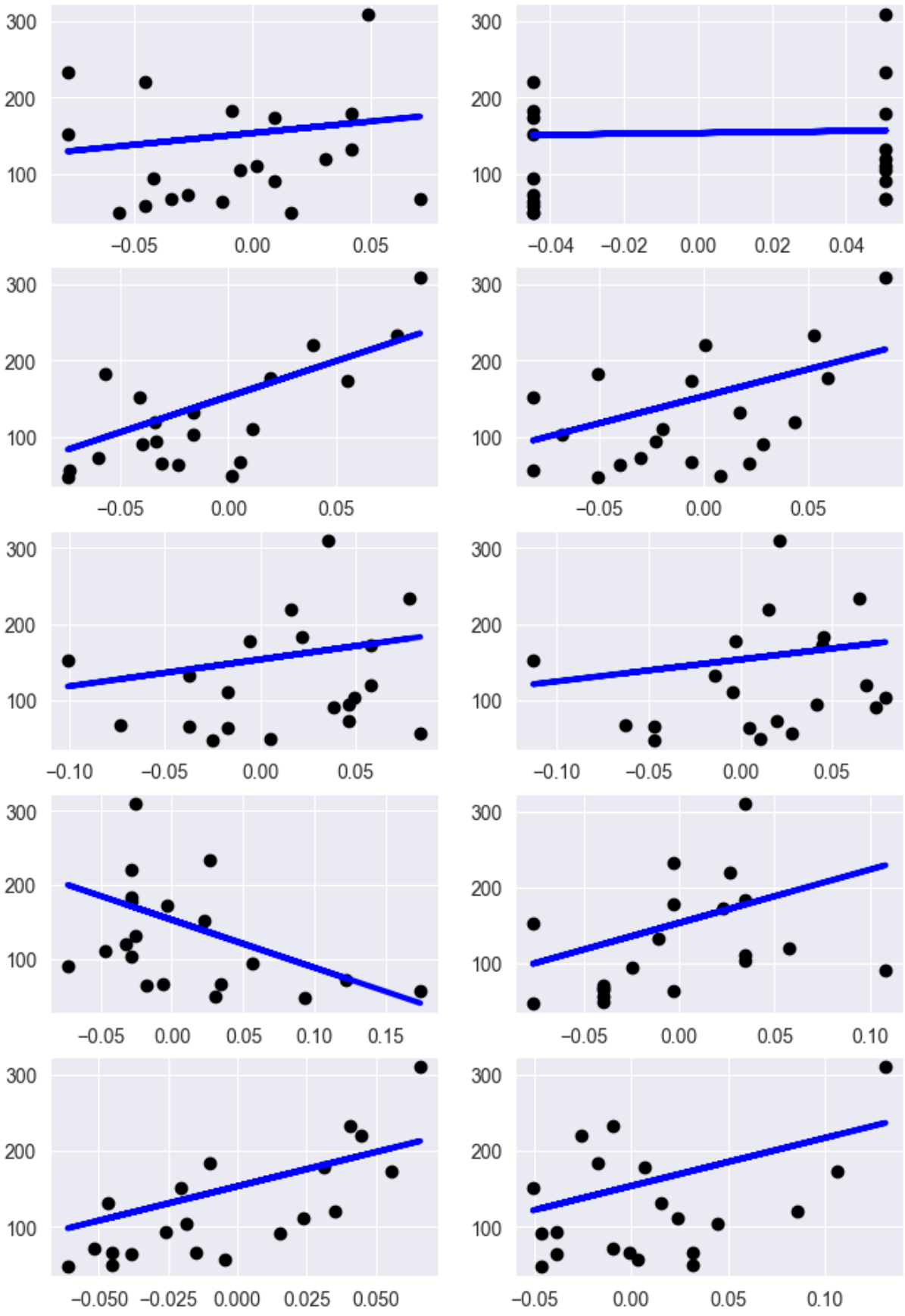


2.8

In [48]:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import datasets

# Tải dữ liệu bệnh tiểu đường từ thư viện sklearn
diabetes = datasets.load_diabetes()

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
x_train = diabetes.data[:-20]  # Dữ liệu huấn luyện (loại bỏ 20 mẫu cuối)
y_train = diabetes.target[:-20]  # Nhãn tương ứng với tập huấn luyện
x_test = diabetes.data[-20:]  # Dữ liệu kiểm tra (20 mẫu cuối)
y_test = diabetes.target[-20:]  # Nhãn tương ứng với tập kiểm tra

# Tạo một hình (figure) để chứa các biểu đồ con
plt.figure(figsize=(8, 12))

# Lặp qua các thuộc tính từ 0 đến 9 (có tổng cộng 10 thuộc tính)
for f in range(0, 10):
    xi_test = x_test[:, f]  # Lấy dữ liệu kiểm tra của thuộc tính thứ f
    xi_train = x_train[:, f]  # Lấy dữ liệu huấn luyện của thuộc tính thứ f

    # Chuyển đổi các mảng xi_test và xi_train từ vector hàng thành vector cột
    xi_test = xi_test[:, np.newaxis]
    xi_train = xi_train[:, np.newaxis]
```

```
    # Tạo mô hình hồi quy tuyến tính
    linreg = linear_model.LinearRegression()

    # Huấn luyện mô hình trên dữ liệu huấn luyện sử dụng chỉ một thuộc tính
    linreg.fit(xi_train, y_train)

    # Dự đoán giá trị đầu ra trên dữ liệu kiểm tra sử dụng chỉ một thuộc tính
    y = linreg.predict(xi_test)

    # Vẽ biểu đồ con (subplot) để so sánh kết quả dự đoán với dữ liệu kiểm tra
    plt.subplot(5, 2, f + 1)
    plt.scatter(xi_test, y_test, color='k')  # Vẽ các điểm dữ liệu kiểm tra
    plt.plot(xi_test, y, color='b', linewidth=3)  # Vẽ đường dự đoán của mô hình

# Hiển thị biểu đồ
plt.show()
```


300
200
100
−0.05
0.00
0.05
300
200
100
−0.04
−0.02
0.00
0.02
0.04
300
200
100
−0.05
0.00
0.05
300
200
100
−0.05
0.00
0.05
300
200
100
−0.10
−0.05
0.00
0.05
300
200
100
−0.10
−0.05
0.00
0.05
300
200
100
−0.05
0.00
0.05
0.10
0.15
300
200
100
−0.05
0.00
0.05
0.10
300
200
100
−0.050
−0.025
0.000
0.025
0.050
300
200
100
−0.05
0.00
0.05
0.10

2.9

In [51]:
```
# import the necessary libraries
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# load the breast cancer dataset
X, y = load_breast_cancer(return_X_y=True)
# split the train and test dataset
X_train, X_test,\
        y_train, y_test = train_test_split(X, y,
                                                                    test_size=0
                                                                    random_stat

# LogisticRegression
clf = LogisticRegression(random_state=0)
clf.fit(X_train, y_train)
# Prediction
y_pred = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print("Logistic Regression model accuracy (in %):", acc*100)
```

```
Logistic Regression model accuracy (in %): 96.49122807017544
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\linear_model\_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

In [53]:
```
from sklearn.model_selection import train_test_split
from sklearn import datasets, linear_model, metrics

# load the digit dataset
digits = datasets.load_digits()

# defining feature matrix(X) and response vector(y)
X = digits.data
y = digits.target

# splitting X and y into training and testing sets
X_train, X_test,\
        y_train, y_test = train_test_split(X, y,
                                                                    test_size=0
                                                                    random_stat


# create logistic regression object
reg = linear_model.LogisticRegression()

# train the model using the training sets
reg.fit(X_train, y_train)

# making predictions on the testing set
```

```
y_pred = reg.predict(X_test)

# comparing actual response values (y_test)
# with predicted response values (y_pred)
print("Logistic Regression model accuracy(in %):",
        metrics.accuracy_score(y_test, y_pred)*100)
```

```
Logistic Regression model accuracy(in %): 96.52294853963839
```

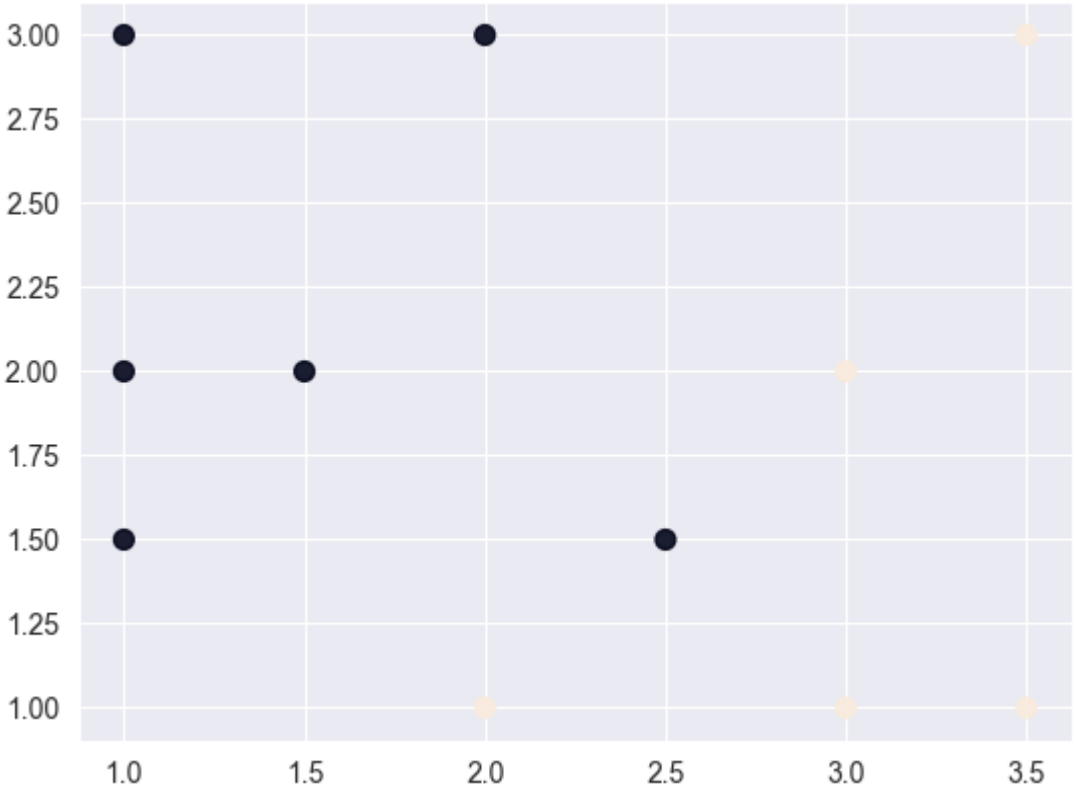
```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\linear_model\_logisti
c.py:460: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

2.10

In [58]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm

# Tạo dữ liệu mẫu với hai thuộc tính (features) và nhãn (labels)
x = np.array(
    [
        [1, 3],
        [1, 2],
        [1, 1.5],
        [1.5, 2],
        [2, 3],
        [2.5, 1.5],
        [2, 1],
        [3, 1],
        [3, 2],
        [3.5, 1],
        [3.5, 3],
    ]
)

# Nhãn tương ứng với từng mẫu dữ liệu: 0 cho các điểm màu xanh, 1 cho các điểm màu
y = [0] * 6 + [1] * 5

# Vẽ biểu đồ phân tán (scatter plot) của dữ liệu
plt.scatter(x[:, 0], x[:, 1], c=y, s=50, alpha=0.9)

# Trong biểu đồ này:
# - x[:, 0] là giá trị của thuộc tính thứ nhất trên trục X
# - x[:, 1] là giá trị của thuộc tính thứ hai trên trục Y
# - c=y xác định màu sắc của từng điểm dữ liệu dựa trên nhãn (0 cho màu xanh, 1 cho
# - s=50 xác định kích thước của điểm trên biểu đồ
# - alpha=0.9 xác định độ mờ của điểm, giá trị càng gần 1 thì điểm càng rõ nét
```

Out[58]: <matplotlib.collections.PathCollection at 0x17f47f079d0>

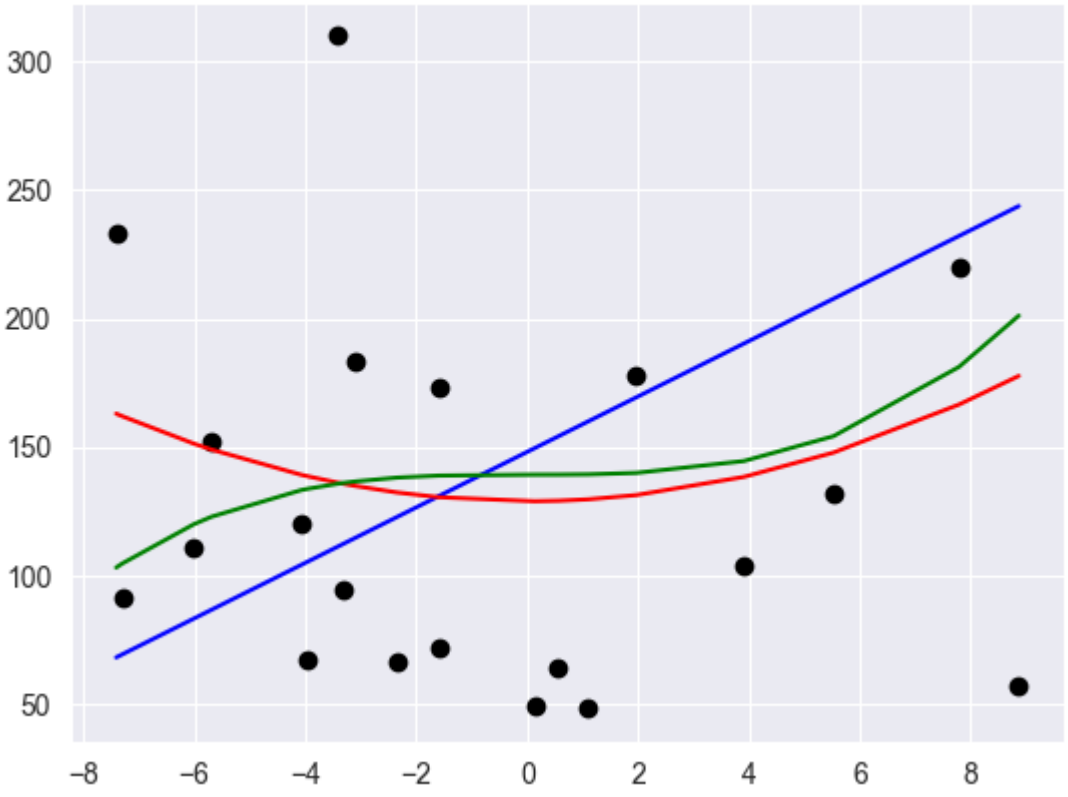


2.11

In [60]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn import datasets

# Tải dữ liệu về bệnh tiểu đường từ thư viện sklearn
diabetes = datasets.load_diabetes()

# Chia dữ liệu thành tập huấn luyện và tập kiểm tra
x_train = diabetes.data[:-20]  # Dữ liệu huấn luyện (loại bỏ 20 mẫu cuối)
y_train = diabetes.target[:-20]  # Nhãn tương ứng với tập huấn luyện
x_test = diabetes.data[-20:]  # Dữ liệu kiểm tra (20 mẫu cuối)
y_test = diabetes.target[-20:]  # Nhãn tương ứng với tập kiểm tra

# Lấy chỉ một thuộc tính (thuộc tính thứ 3) để thực hiện hồi quy
x0_test = x_test[:, 2]  # Lấy dữ liệu kiểm tra của thuộc tính thứ 3
x0_train = x_train[:, 2]  # Lấy dữ liệu huấn luyện của thuộc tính thứ 3

# Chuyển đổi các mảng x0_test và x0_train từ vector hàng thành vector cột
x0_test = x0_test[:, np.newaxis]
x0_train = x0_train[:, np.newaxis]

# Sắp xếp các giá trị trên trục X của dữ liệu kiểm tra
x0_test.sort(axis=0)
```

```
# Tăng giá trị của dữ liệu kiểm tra và dữ liệu huấn luyện lên 100 lần (scaling)
x0_test = x0_test * 100
x0_train = x0_train * 100

# Tạo mô hình Support Vector Regression (SVR) với kernel tuyến tính
svr = svm.SVR(kernel='linear', C=1000)

# Tạo mô hình SVR với kernel đa thức bậc 2
svr2 = svm.SVR(kernel='poly', C=1000, degree=2)

# Tạo mô hình SVR với kernel đa thức bậc 3
svr3 = svm.SVR(kernel='poly', C=1000, degree=3)

# Huấn luyện các mô hình trên dữ liệu huấn luyện
svr.fit(x0_train, y_train)
svr2.fit(x0_train, y_train)
svr3.fit(x0_train, y_train)

# Dự đoán giá trị đầu ra trên dữ liệu kiểm tra
y = svr.predict(x0_test)
y2 = svr2.predict(x0_test)
y3 = svr3.predict(x0_test)

# Vẽ biểu đồ để so sánh kết quả dự đoán với dữ liệu kiểm tra
plt.scatter(x0_test, y_test, color='k')  # Vẽ các điểm dữ liệu kiểm tra
plt.plot(x0_test, y, color='b')  # Vẽ đường dự đoán của mô hình SVR tuyến tính
plt.plot(x0_test, y2, c='r')  # Vẽ đường dự đoán của mô hình SVR đa thức bậc 2
plt.plot(x0_test, y3, c='g')  # Vẽ đường dự đoán của mô hình SVR đa thức bậc 3
```

Out[60]: [<matplotlib.lines.Line2D at 0x17f480271c0>]



2.12

In [63]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans
```

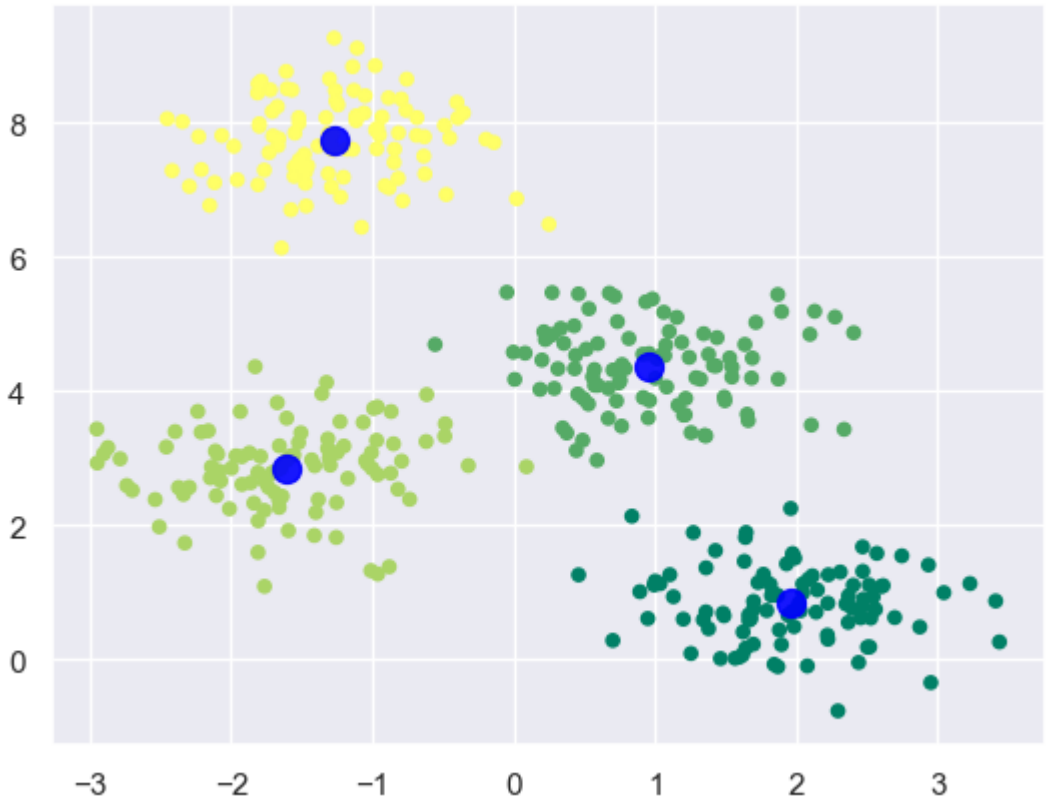
In [65]:
```
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=400, centers=4, cluster_std=0.60, random_state=0)
```

In [66]:
```
plt.scatter(X[:, 0], X[:, 1], s=20)
plt.show()
```

In [67]:
```
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.
4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

In [68]:
```
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=20, cmap='summer')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9)
plt.show()
```



In [69]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
from sklearn.cluster import KMeans
```

In [70]:
```
from sklearn.datasets import load_digits
digits = load_digits()
digits.data.shape
```

Out[70]:
```
(1797, 64)
```

In [71]:
```
kmeans = KMeans(n_clusters=10, random_state=0)
clusters = kmeans.fit_predict(digits.data)
kmeans.cluster_centers_.shape
```

```
c:\Users\tien2\miniconda3\envs\intel\lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[71]:
```
(10, 64)
```

In [73]:
```
fig, ax = plt.subplots(2, 5, figsize=(8, 3))
centers = kmeans.cluster_centers_.reshape(10, 8, 8)
for axi, center in zip(ax.flat, centers):
    axi.set(xticks=[], yticks=[])
    axi.imshow(center, interpolation="nearest", cmap=plt.cm.binary)
```