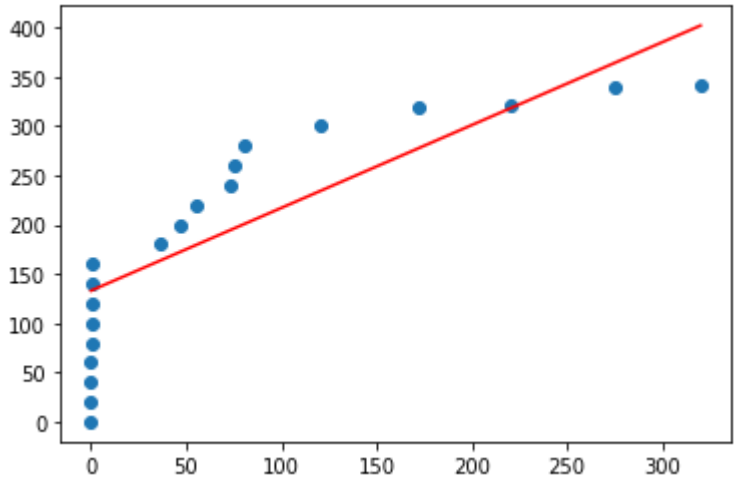
LINEAR REGRESSION

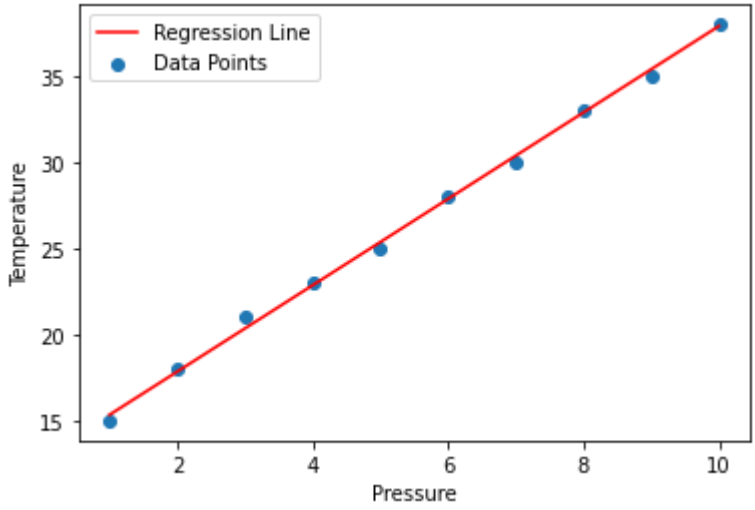
In [3]:
```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score

data={
    "pressure":[0.0002,0.0012,0.0060,0.0300,0.0900,0.1000,0.1100,0.1200,0.2
700,36.0000,47.0000,55.0000,73.0000,75.0000,80.7000,120.7000,172.0000,220.0
000,275.0000,320.0000],
    "temperature":[0,20,40,60,80,100,120,140,160,180,200,220,240,260,280,30
0,320,322,340,342]
    }

df=pd.DataFrame(data)

x=df.iloc[:,0].values.reshape(-1,1)
y=df.iloc[:,1].values.reshape(-1,1)

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_sta
te=42)

model=LinearRegression()
model.fit(x_train,y_train)
print(model.predict([[100]]))
y_pred = model.predict(x_test)
mae=mean_absolute_error(y_test,y_pred)
mse=mean_squared_error(y_test,y_pred)
r2=r2_score(y_test,y_pred)
print("mse: ",mse)
print("mae: ",mae)
print("R2: ",r2)

x_line = np.linspace(x.min(), x.max(), 100).reshape(-1, 1)
plt.plot(x_line, model.predict(x_line), 'r')
plt.scatter(x,y)
plt.show()
```

```
[[217.09926859]]
mse:  8689.589903453918
mae:  78.86069844924648
R2:  0.6182204055905927
```



In [22]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_sco
re

# Create custom dataset
pressure = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
temperature = [15, 18, 21, 23, 25, 28, 30, 33, 35, 38]

df = pd.DataFrame({
    "pressure": pressure,
    "temperature": temperature
})

# Prepare the data
X = df[["pressure"]].values  # 2D array of features
y = df["temperature"].values.reshape(-1, 1)  # target values

# Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
ndom_state=42)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions and evaluate
y_pred = model.predict(X_test)
print("MSE:", mean_squared_error(y_test, y_pred))
print("MAE:", mean_absolute_error(y_test, y_pred))
print("R2:", r2_score(y_test, y_pred))

# Plot the regression line and data points
x_line = np.linspace(min(pressure), max(pressure), 100).reshape(-1, 1)
plt.plot(x_line, model.predict(x_line), 'r', label="Regression Line")
plt.scatter(X, y, label="Data Points")
plt.xlabel("Pressure")
plt.ylabel("Temperature")
plt.legend()
plt.show()
```

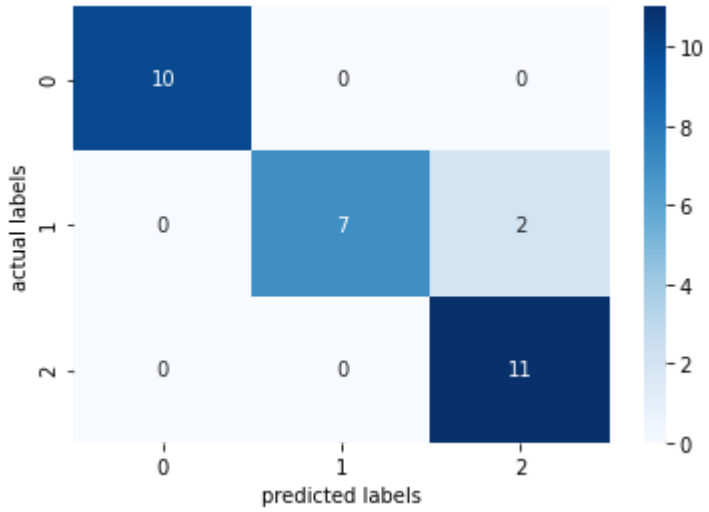
```
MSE: 0.09412158145065727
MAE: 0.28017241379310853
R2: 0.9986972791494718
```



LOGISTIC

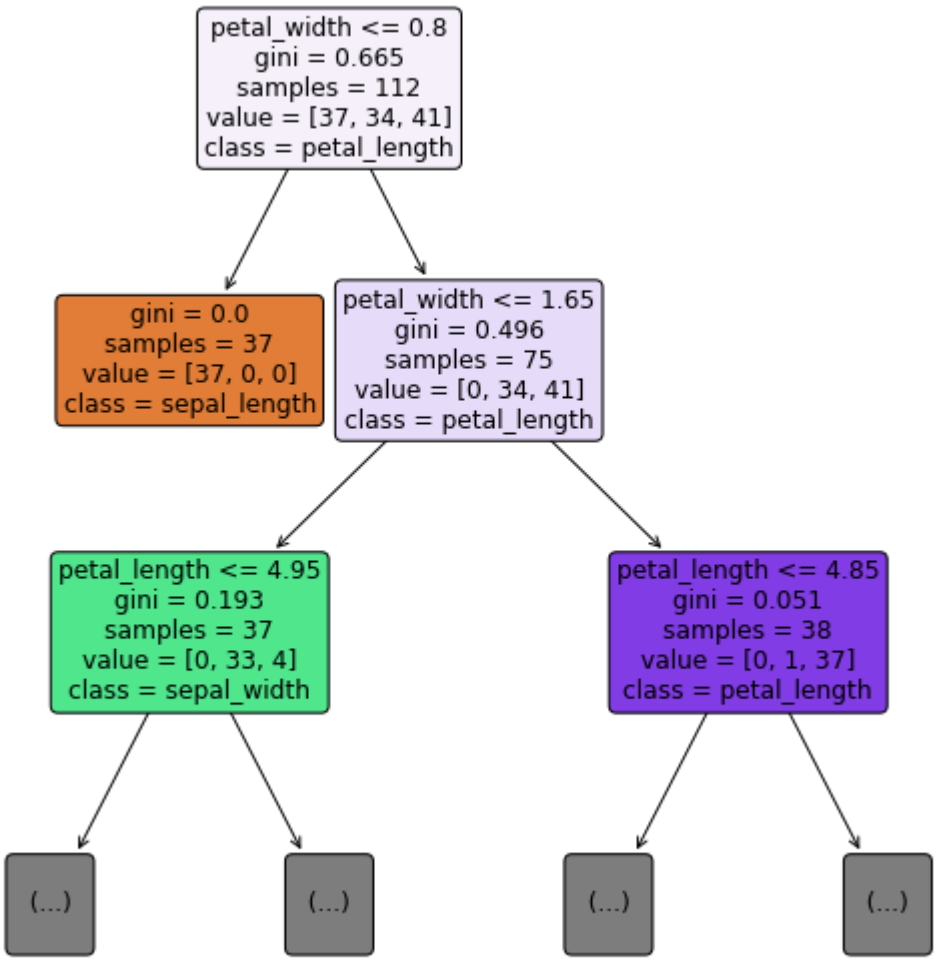
In [4]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusio
n_matrix

df=sns.load_dataset("iris")
x=df[["sepal_length","sepal_width","petal_length","petal_width"]]
y=df[["species"]]

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_sta
te=42)
model=LogisticRegression(C=0.01)
model.fit(x_train,y_train)
pred=model.predict(x_test)

print("accuracy: ",accuracy_score(y_test,pred))
print("\nClassification_report: ",classification_report(y_test,pred))
cm=confusion_matrix(y_test,pred)


sns.heatmap(cm,annot=True,cmap="Blues")
plt.xlabel("predicted labels")
plt.ylabel("actual labels")
plt.show()
```

```
C:\CYrus\anaconda\lib\site-packages\sklearn\utils\validation.py:63: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expecte
d. Please change the shape of y to (n_samples, ), for example using ravel
().
  return f(*args, **kwargs)

accuracy:  0.9333333333333333

Classification_report:                precision    recall  f1-score   supp
ort

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      0.78      0.88         9
   virginica       0.85      1.00      0.92        11

    accuracy                           0.93        30
   macro avg       0.95      0.93      0.93        30
weighted avg       0.94      0.93      0.93        30
```



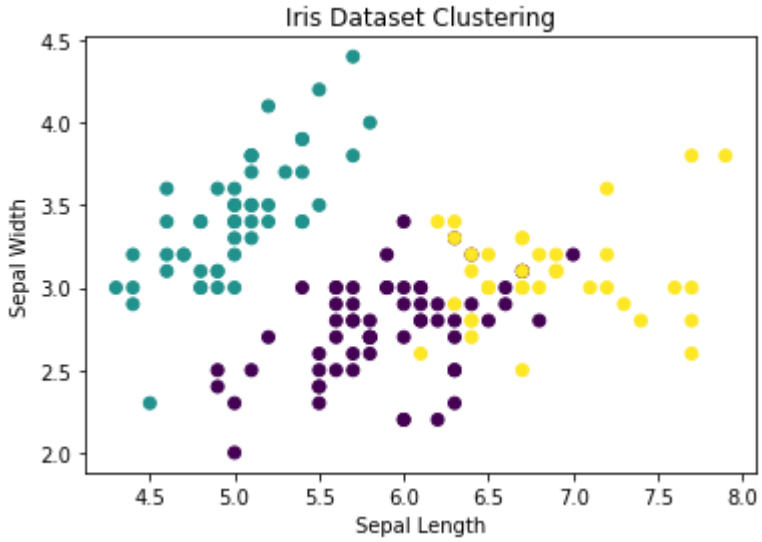
DECISION TREE

In [5]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

data=sns.load_dataset('iris')
x=data.values[:,0:4]
y=data.values[:,4]

x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=1)
model=DecisionTreeClassifier(random_state=54)
model.fit(x_train,y_train)

pred=model.predict(x_test)

plt.figure(figsize=(10,10))
plot_tree(model,filled=True,rounded=True,feature_names=data.columns[:4].to_
list(),class_names=data.columns[:5].unique().to_list(),max_depth=2)
plt.show()
```



KMEANS CLUSTERING

In [6]:
```
import seaborn as sns
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

data=sns.load_dataset('iris')
model=KMeans(n_clusters=3)
model.fit(data.drop('species',axis=1))
labels=model.labels_

plt.scatter(data['sepal_length'],data['sepal_width'],c=labels)
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Iris Dataset Clustering')
plt.show()
```



PERCEPTRON

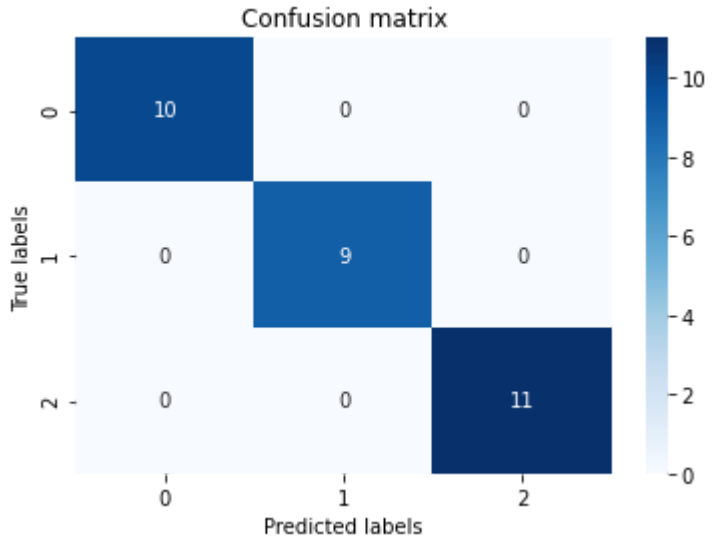
In [8]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
data = load_iris()
x, y = data.data[:100, :], data.target[:100]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, ra
ndom_state=42)
perceptron = Perceptron(random_state=42)
perceptron.fit(x_train, y_train)
y_pred = perceptron.predict(x_test)
print(f'Prediction: {y_pred}')
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```
Prediction: [1 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 1 1 0 0]
Accuracy: 1.0
```

RANDOM FOREST CLASSIFIER

In [9]:
```
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, classificatio
n_report
import seaborn as sns
import matplotlib.pyplot as plt
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, ra
ndom_state=42)
model = RandomForestClassifier()
clf=model.fit(x_train, y_train)
y_pred = clf.predict(x_test)
# print(y_pred)
accuracy=accuracy_score(y_test,y_pred)
print(f'Accuracy:{accuracy:.2f}')
print("Report: ", classification_report(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, cmap="Blues")
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.title("Confusion matrix")
plt.show()
```

```
Accuracy:1.00
Report:                precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

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



VOTING CLASSIFIER

In [10]:
```
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
ndom_state=42)
log_clf = LogisticRegression(max_iter=200)
tree_clf = DecisionTreeClassifier(random_state=42)
knn_clf = KNeighborsClassifier()

# Hard Voting Classifier (majority vote)
hard_voting_clf = VotingClassifier(estimators=[
    ('log_clf', log_clf),
    ('tree_clf', tree_clf),
    ('knn_clf', knn_clf)
], voting='hard')

# Soft Voting Classifier (average probabilities)
soft_voting_clf = VotingClassifier(estimators=[
    ('log_clf', log_clf),
    ('tree_clf', tree_clf),
    ('knn_clf', knn_clf)
], voting='soft')

# Train and evaluate Hard Voting Classifier
hard_voting_clf.fit(X_train, y_train)
y_pred_hard = hard_voting_clf.predict(X_test)
hard_acc = accuracy_score(y_test, y_pred_hard)
print(f"Hard Voting Classifier Accuracy: {hard_acc:.4f}")

# Train and evaluate Soft Voting Classifier
soft_voting_clf.fit(X_train, y_train)
y_pred_soft = soft_voting_clf.predict(X_test)
soft_acc = accuracy_score(y_test, y_pred_soft)
print(f"Soft Voting Classifier Accuracy: {soft_acc:.4f}")
```

```
Hard Voting Classifier Accuracy: 1.0000
Soft Voting Classifier Accuracy: 1.0000
```

APRIORI

In [11]:
```
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
dataset = [
    ['milk', 'bread', 'butter'],
    ['bread', 'butter'],
    ['milk', 'bread'],
    ['milk', 'bread', 'butter'],
    ['bread', 'butter']
]
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
print("\nFrequent Itemsets:")
print(frequent_itemsets)
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=
1.0)
print("\nAssociation Rules:")
print(rules)
print("\nNumber of association rules: ",len(rules))
```

```
Frequent Itemsets:
   support         itemsets
0      1.0          (bread)
1      0.8         (butter)
2      0.6           (milk)
3      0.8  (butter, bread)
4      0.6    (milk, bread)

Association Rules:
  antecedents consequents  antecedent support  consequent support  support
\
0    (butter)     (bread)                 0.8                 1.0      0.8
1     (bread)    (butter)                 1.0                 0.8      0.8
2      (milk)     (bread)                 0.6                 1.0      0.6
3     (bread)      (milk)                 1.0                 0.6      0.6

   confidence  lift  leverage  conviction
0         1.0   1.0       0.0         inf
1         0.8   1.0       0.0         1.0
2         1.0   1.0       0.0         inf
3         0.6   1.0       0.0         1.0

Number of association rules:  4
```

naive bayes

In [12]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split into training and test sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, ra
ndom_state=42)

# Initialize and train the Gaussian Naïve Bayes classifier
nb = GaussianNB()
nb.fit(X_train, y_train)

# Make predictions on the test set
y_pred = nb.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9777777777777777
```

svm

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report
)

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

X_2d = X[:, :2]

X_train, X_test, y_train, y_test = train_test_split(
    X_2d, y, test_size=0.3, random_state=42
)
clf = svm.SVC(kernel='linear', random_state=42)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.2f}")

cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

```
Accuracy: 0.80
Confusion Matrix:
[[19  0  0]
 [ 0  7  6]
 [ 0  3 10]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.70      0.54      0.61        13
           2       0.62      0.77      0.69        13

    accuracy                           0.80        45
   macro avg       0.78      0.77      0.77        45
weighted avg       0.81      0.80      0.80        45
```