In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split

%matplotlib inline
```

ATENÇÃO!

- A função train_test_split é usada para separar os conjuntos de dados em treino e teste.
- Apenas os conjuntos de treino devem ser utilizados para ajustar os modelos: não vale mudar parâmetros e ir verificando se a métrica melhora no conjunto de teste.
- O conjunto de teste é utilizado apenas para calcular a métrica final.

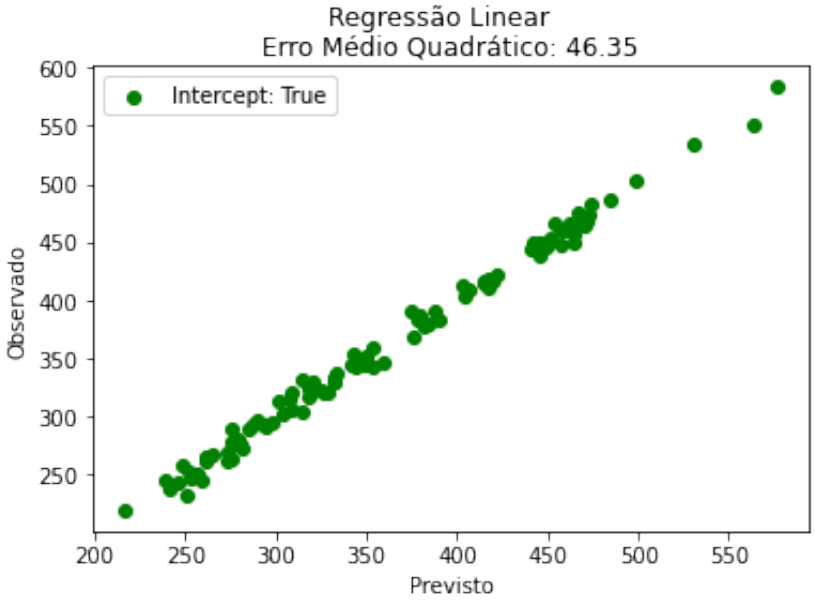
Exercício 1

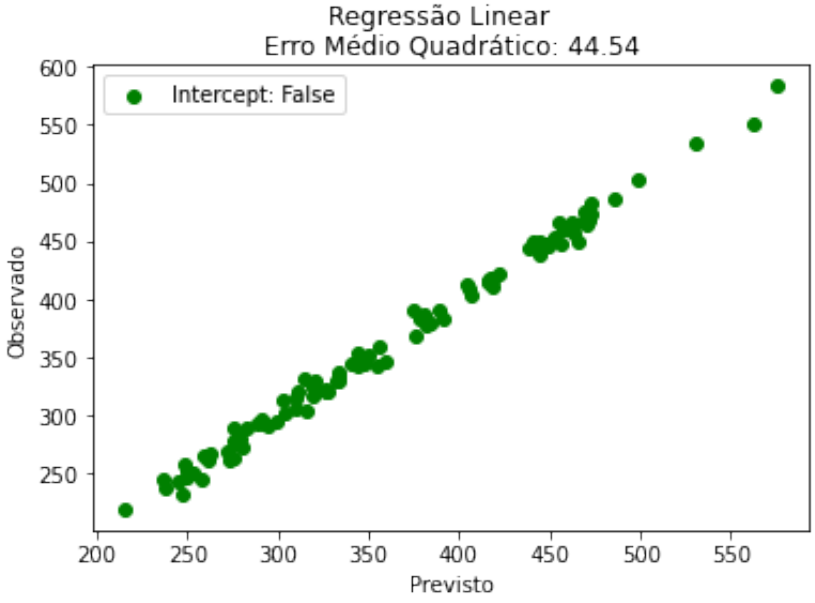
Ajuste um modelo linear e um modelo não-linear à sua escolha ao conjunto de treino abaixo (X_train e y_train).

1. Calcule o erro médio quadrático de cada modelo no conjunto de teste.
2. Plote suas predições e o valor real observado, qual modelo você julga mais indicado e porquê?
3. Teste alguns parâmetros para os modelos e verifique o que muda.

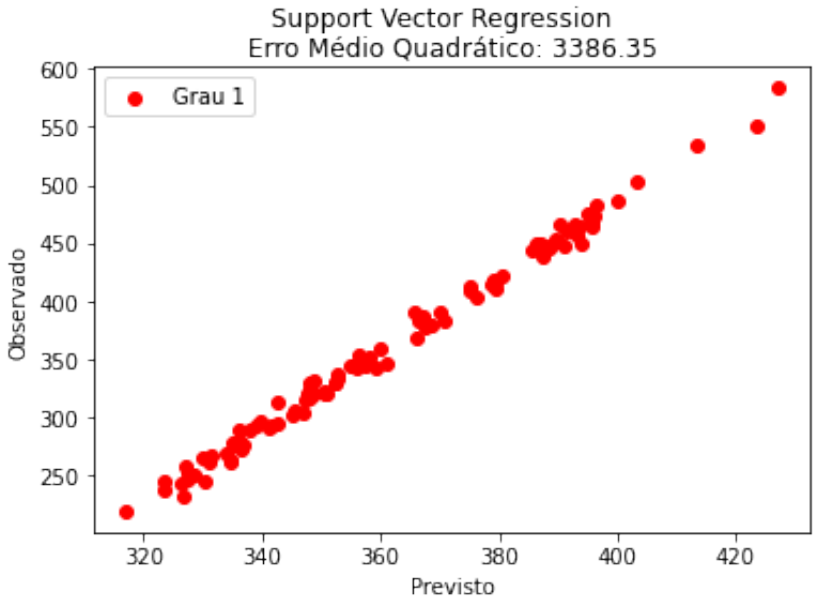
In [21]:
```
# Importações
from sklearn.linear_model import LinearRegression
from sklearn import svm
from sklearn.metrics import mean_squared_error # erro médio quadrático

# Carrega dados
sp = pd.read_csv('SP500.csv')

# Separa e treina
X, y = sp.drop('S&P500NextWeek',axis=1), sp['S&P500NextWeek']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
```

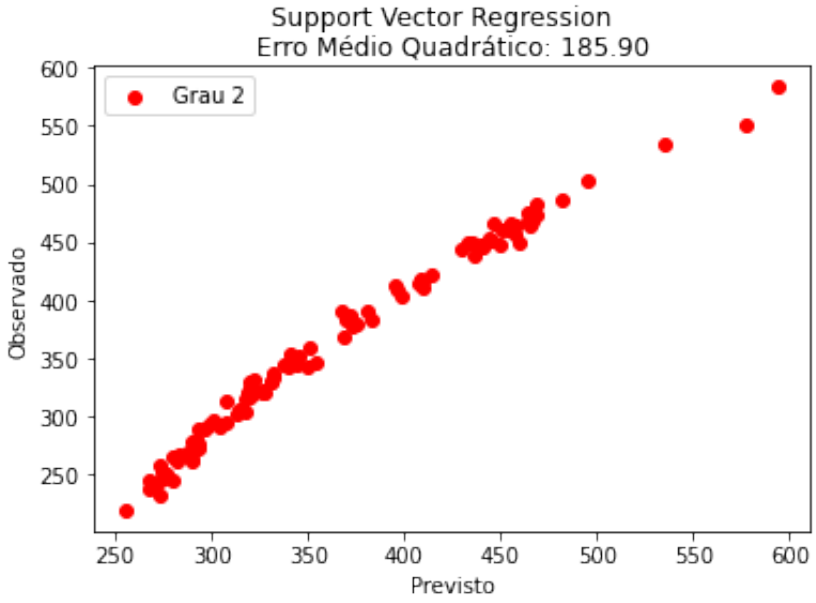
In [94]:
```
# Ajuste do Modelo Linear

intercepts = [True, False]

for intercept in intercepts:
    linearReg = LinearRegression(fit_intercept=intercept)
    linearReg.fit(X_train,y_train)
    y_pred = linearReg.predict(X_test) # Prevê os valores

    plt.scatter(y_pred, y_test, color='g', label='Intercept: %r' % intercep
    plt.title('Erro Médio Quadrático: %.2f' % mean_squared_error(y_test,y_p
    plt.suptitle('Regressão Linear')
    plt.xlabel('Previsto')
    plt.ylabel('Observado')
    plt.legend()
    plt.show()
```

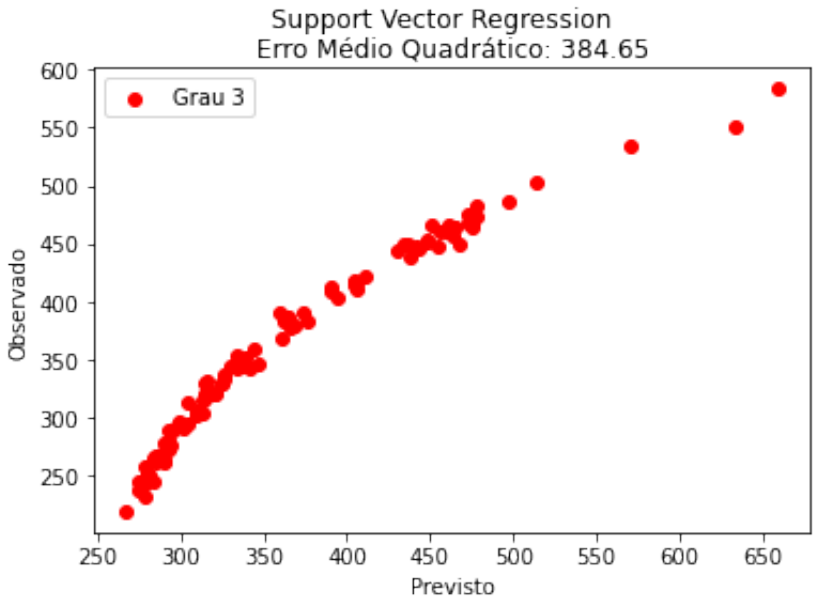




In [83]:
```
# Ajuste do Modelo Não Linear

graus = [1,2,3]

for grau in graus:
    supportReg = svm.SVR(kernel='poly', degree=grau)
    supportReg.fit(X_train, y_train)
    y_pred = supportReg.predict(X_test) # Prevê os valores

    plt.scatter(y_pred, y_test, color='r', label='Grau %s' % supportReg.dec
    plt.title('Erro Médio Quadrático: %.2f' % mean_squared_error(y_test,y_p
    plt.suptitle('Support Vector Regression')
    plt.xlabel('Previsto')
    plt.ylabel('Observado')
    plt.legend()
    plt.show()
```


Support Vector Regression
Erro Médio Quadrático: 3386.35
Grau 1
Observado
Previsto
600
550
500
450
400
350
300
250
320
340
360
380
400
420


Support Vector Regression
Erro Médio Quadrático: 185.90
Grau 2
Observado
Previsto
600
550
500
450
400
350
300
250
250
300
350
400
450
500
550
600


Support Vector Regression
Erro Médio Quadrático: 384.65
Grau 3
Observado
Previsto
600
550
500
450
400
350
300
250
250
300
350
400
450
500
550
600
650

Resposta:

Estou julgando o modelo linear mais vantajoso pois apresentou um erro médio quadrático menor que o modelo polinomial, indicando que o valor previsto é mais próximo do valor observado no modelo linear que no modelo polinomial.

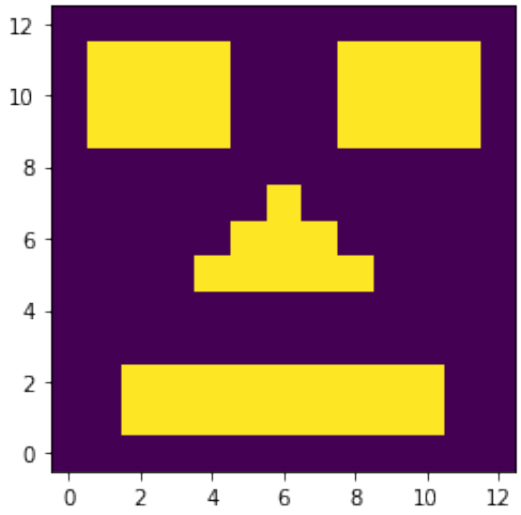
Exercício 2

Máscara de halloween. Crie um modelo que seja capaz de reconstruir a máscara do dataset 'mask.csv', usando apenas o conjunto de dados de treino como entrada.

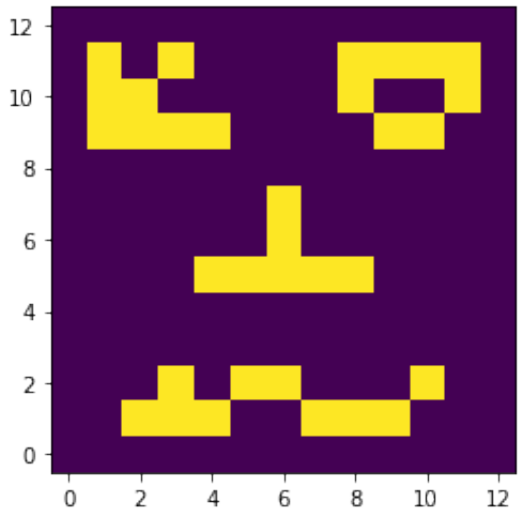
Teste pelo menos 3 métodos diferentes (Regressão Logística, KNN, SVM, Redes Neurais...), e compare com o desenho original. Qual modelo você escolheria?

Calcule a acurácia do modelo para te ajudar na decisão.

In [2]:
```
def show_mask(df):
    m = np.empty((13,13)) #
    for i, col in df.iterrows():
        m[col['x1'],col['x0']] = col['y']

    plt.imshow(m, origin='lower')

mask = pd.read_csv('mask.csv')
show_mask(mask)
```



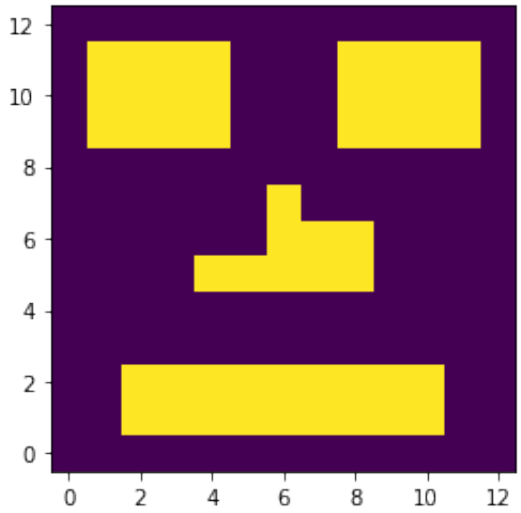
In [6]:
```
train = mask.sample(frac=0.7,random_state=3)
test  = mask.drop(train.index)

show_mask(train)

X_train, y_train = train.drop('y',axis=1), train['y']
X_test,  y_test  = test.drop('y',axis=1),  test['y']
```



In [75]:
```
# Ajuste seu modelo, e passe a saída prevista para a variável pred para pl
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error # erro médio quadrático
```

In [78]:
```
# Ajuste dos Modelos
modelos = [KNeighborsClassifier(),
           LogisticRegression(),
           GaussianNB(),
           MLPClassifier(),
           GradientBoostingClassifier(),
           DecisionTreeClassifier(),
           RandomForestClassifier()]

maiorScore = 0
menorMSE = 99999999

for modelo in modelos:
    X_train, y_train = train.drop('y',axis=1), train['y']
    X_test,  y_test  = test.drop('y',axis=1),  test['y']

    model = modelo
    model.fit(X_train, y_train)
    modelPred = model.predict(X_test)

    score = model.score(X_train, y_train)
    mse = mean_squared_error(y_test,modelPred)

    print('Modelo: %s' % type(model).__name__)
    print('Score %.2f' % score)
    print('MSE %.2f' % mse)
    print()

    X_test['y'] = modelPred
    new_mask = pd.concat([train,X_test])
    show_mask(new_mask)
```

```
Modelo: KNeighborsClassifier
Score 0.92
MSE 0.20

Modelo: LogisticRegression
Score 0.72
MSE 0.35

Modelo: GaussianNB
Score 0.72
MSE 0.35

/usr/local/lib/python3.8/site-packages/sklearn/neural_network/_multilayer_p
erceptron.py:582: ConvergenceWarning: Stochastic Optimizer: Maximum iterati
ons (200) reached and the optimization hasn't converged yet.
  warnings.warn(
Modelo: MLPClassifier
Score 0.73
MSE 0.35

Modelo: GradientBoostingClassifier
Score 0.98
MSE 0.04

Modelo: DecisionTreeClassifier
Score 1.00
MSE 0.04

Modelo: RandomForestClassifier
Score 1.00
MSE 0.04
```



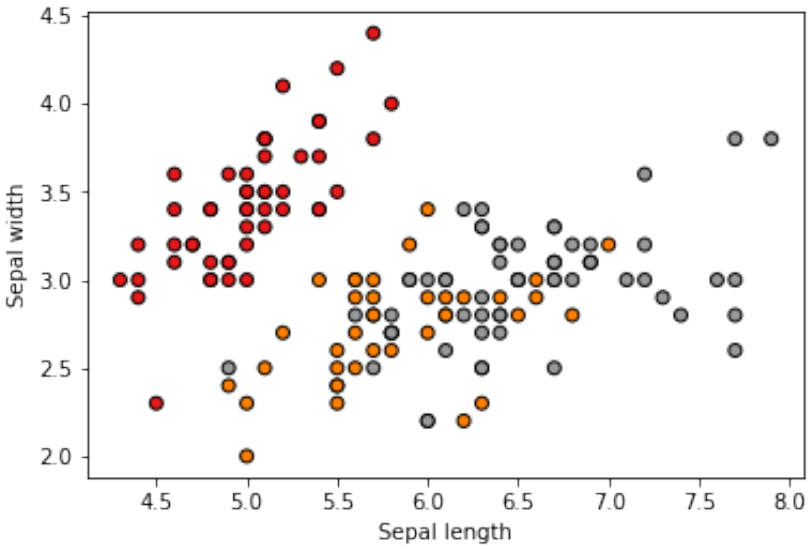
Exercício 3

Conjunto de dados Iris.

1. Ajuste um modelo de classificação ao conjunto de dados iris.
2. Use a função sklearn.metrics.classification_report para avaliar seu modelo.
3. Analise e explique a saída do relatório.
4. Teste modelos e parâmetros diferentes.

In [3]:
```
# Importações
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree

# Carrega dados
iris = load_iris()

X = iris.data
y = iris.target

# Separa e treina
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, r
```

In [4]:
```
# Plota gráfico inicial
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Set1,
            edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```



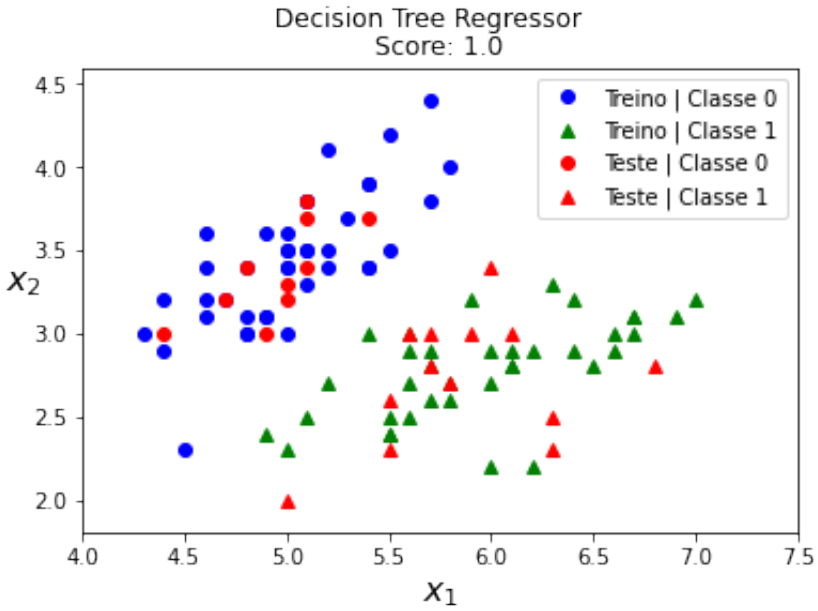
In [14]:
```
# Ajuste do Modelo de Regressão Logística

solvers = ['liblinear', 'sag', 'saga']

# Separa e treina
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, 

for slv in solvers:
    print('Modelo: ' + slv)
    logReg = LogisticRegression(solver=slv)
    logReg.fit(X_train, y_train)
    y_pred = logReg.predict(X_test)
    print('Score: ' + str(logReg.score(X_train, y_train)))
    print(classification_report(y_test, y_pred))
    print()
```

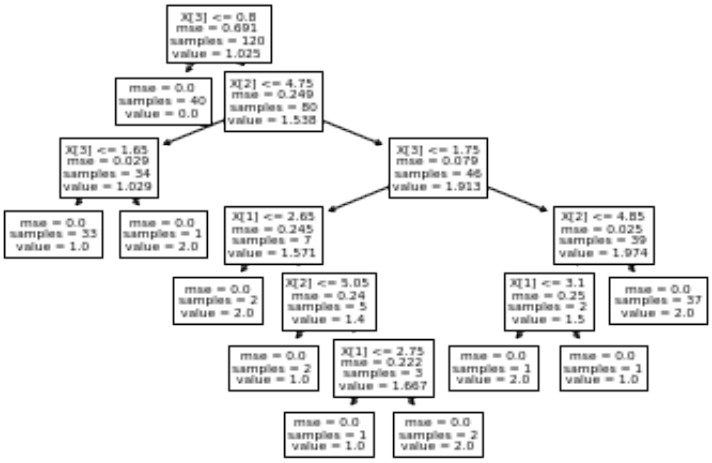
```
Modelo: liblinear
Score: 0.95
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.77      0.87        13
           2       0.70      1.00      0.82         7

    accuracy                           0.90        30
   macro avg       0.90      0.92      0.90        30
weighted avg       0.93      0.90      0.90        30


Modelo: sag
Score: 0.9833333333333333
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        13
           2       0.88      1.00      0.93         7

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30


Modelo: saga
Score: 0.975
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.92      0.96        13
           2       0.88      1.00      0.93         7

    accuracy                           0.97        30
   macro avg       0.96      0.97      0.96        30
weighted avg       0.97      0.97      0.97        30


/usr/local/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:329: Co
nvergenceWarning: The max_iter was reached which means the coef_ did not co
nverge
  warnings.warn("The max_iter was reached which means "
/usr/local/lib/python3.8/site-packages/sklearn/linear_model/_sag.py:329: Co
nvergenceWarning: The max_iter was reached which means the coef_ did not co
nverge
  warnings.warn("The max_iter was reached which means "
```

In [7]:

```
def plot_dataset(X, y, axes, marker=['bo','g^'], label='Treino'):
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], marker[0], label=label + ' | Cla
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], marker[1], label=label + ' | Cla
    plt.axis(axes)
    plt.xlabel(r"$x_1$", fontsize=16)
    plt.ylabel(r"$x_2$", fontsize=16, rotation=0)
    plt.legend()
```

In [22]:
```
# Verificações
# np.amax(X_train)
# np.amin(X_train)
# np.amax(y_train)
# np.amin(y_train)

# Separa e treina
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, r

# Ajuste do Modelo de Árvore de Decisão (Decision Tree Regressor)
treeReg = DecisionTreeRegressor()
treeReg.fit(X_train,y_train)
y_pred = treeReg.predict(X_test)

plot_dataset(X_train, y_train, [4, 7.5, 1.8, 4.6])
plot_dataset(X_test, y_test, [4, 7.5, 1.8, 4.6], marker=['ro','r^'], label=
plt.suptitle('Decision Tree Regressor')
plt.title('Score: ' + str(treeReg.score(X_train, y_train)))
plt.show()

tree.plot_tree(treeReg)
plt.show()

print()

print('\nRelatório de Classificação:\n')
print(classification_report(y_test, y_pred))
```





Relatório de Classificação:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	10
1	1.00	0.92	0.96	13
2	0.88	1.00	0.93	7
accuracy			0.97	30
macro avg	0.96	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

Resposta:

- O **recall** significa "quantos desta classe foram encontrados em todo o número de elementos desta classe"

Todas as classes obtiveram um bom desempenho

- A **precisão** será "quantos estão corretamente classificados entre essa classe"

Somente a classe 2 teve uma precisão baixa, indicando que sua classificação teve mais erros

- A **pontuação f1** é a média harmônica entre precisão e recuperação
- O **suporte** é o número de ocorrências de determinada classe em seu conjunto de dados

Pode-se perceber que as classes 0 e 1 possuem praticamente a mesma quantidade de ocorrências; ou seja, estão bem balanceadas. Porém a classe 2 está defasada

É possível verificar, através do gráfico mostrado anteriormente, que usando o modelo *DecistionTreeRegressor*, obtivemos um score de **1.0** e as classes foram classificadas corretamente

In [ ]: