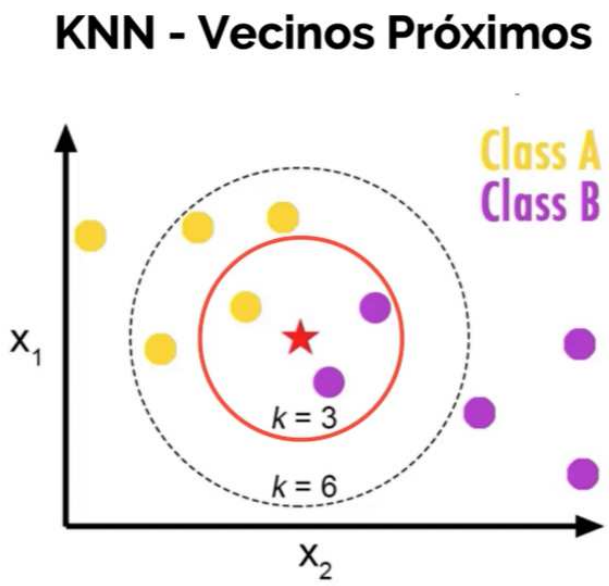
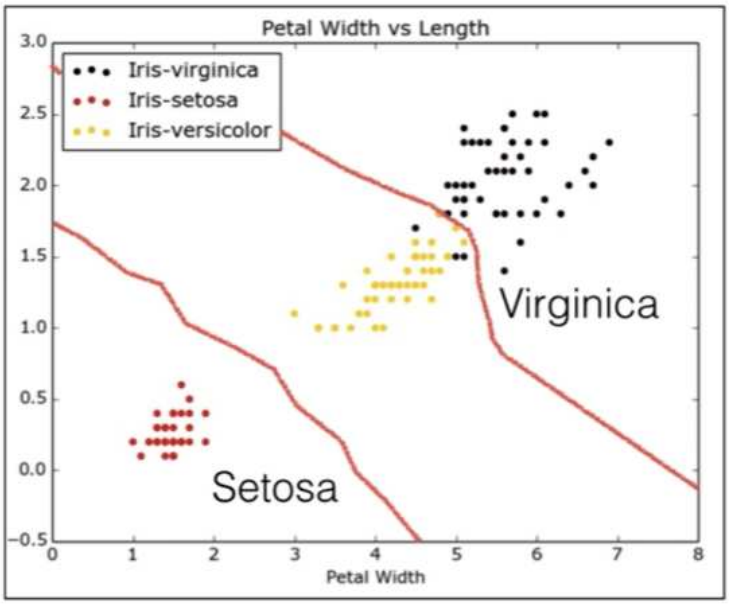
INSTRUCTIVO

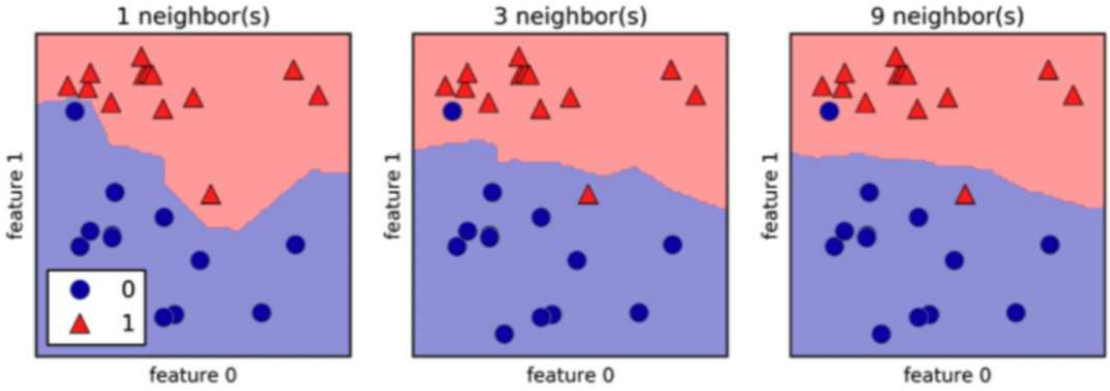
Bases Teóricas



Algoritmo de clasificación

Predice la etiqueta utilizando el numero K de vecinos próximos y hacer una votación





A continuación, mostramos los pasos para iniciar la exploración de K-Vecinos

1. Importamos las librerías base y aparte la base de datos de iris.

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

iris = sns.load_dataset("iris")
from sklearn.model_selection import train_test_split
iris.head()

X = iris.drop('species', axis = 1)
y = iris[['species']]
```

2. Ahora preparamos la separación de los grupos de X como Y para entrenamiento y test.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=30)
```

3. Importamos la librería del algoritmo y entrenamos al modelo con los grupos separados.

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
```

Se está utilizando el criterio de un solo vecino próximo (n_neightbors=1)

```
In [3]: from sklearn.neighbors import
KNeighborsClassifier
   ...: knn = KNeighborsClassifier(n_neighbors=1)
   ...: knn.fit(X_train, y_train)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn
\neighbors\_classification.py:179:
DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return self._fit(X, y)
Out[3]: KNeighborsClassifier(n_neighbors=1)
```

Presenta una sugerencia de uso en caso de ploteo (referencia de shape of….) pero fue pasado (obviado). Es una advertencia propia de la base de datos no de una generalidad en el script.

4. Ahora procedemos a generar las predicciones y matriz de confusión

```
from sklearn.metrics import classification_report, confusion_matrix
#predicciones
pred = knn.predict(X_test)
```

```
Terminal 1/A
   ...: pred = knn.predict(X_test)

In [5]: print(pred)
['setosa' 'setosa' 'setosa' 'virginica' 'versicolor'
'versicolor'
 'virginica' 'virginica' 'versicolor' 'virginica'
'setosa' 'virginica'
 'versicolor' 'versicolor' 'setosa' 'versicolor'
'setosa' 'setosa'
 'setosa' 'virginica' 'versicolor' 'setosa' 'setosa'
'setosa' 'virginica'
 'virginica' 'virginica' 'virginica' 'setosa'
'versicolor' 'virginica'
```

Generamos el reporte de métricas para analizar el resultado.

```
report = classification_report(y_test, pred)
tabla = confusion_matrix(y_test, pred)

print(report)
print(tabla)
```

```
 'virginica' 'virginica']
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       0.91      0.77      0.83        13
   virginica       0.86      0.95      0.90        19

    accuracy                           0.91        45
   macro avg       0.92      0.91      0.91        45
weighted avg       0.91      0.91      0.91        45

[[13  0  0]
 [ 0 10  3]
 [ 0  1 18]]
```

RECORDAR DEL INSTRUCTIVO PASADO:

Matriz de confusión		Estimado por el modelo		
		Negativo (N)	*Positivo (P)*	
Real	*Negativo*	a: (TN)	b: (FP)	
	Positivo	c: (FN)	d: (TP)	**Precisión** ***("precision")*** Porcentaje predicciones positivas correctas: d/(b+d)
		Sensibilidad, exhaustividad ***("Recall")*** Porcentaje casos positivos detectados	**Especifidad** ***(Specifity)*** Porcentaje casos negativos detectados	**Exactitud ("accuracy")** Porcentaje de predicciones correctas *(No sirve en datasets poco equilibrados)*
		d/(d+c)	a/(a+b)	(a+d)/(a+b+c+d)

Observamos la exactitud, según la matriz obtenida:

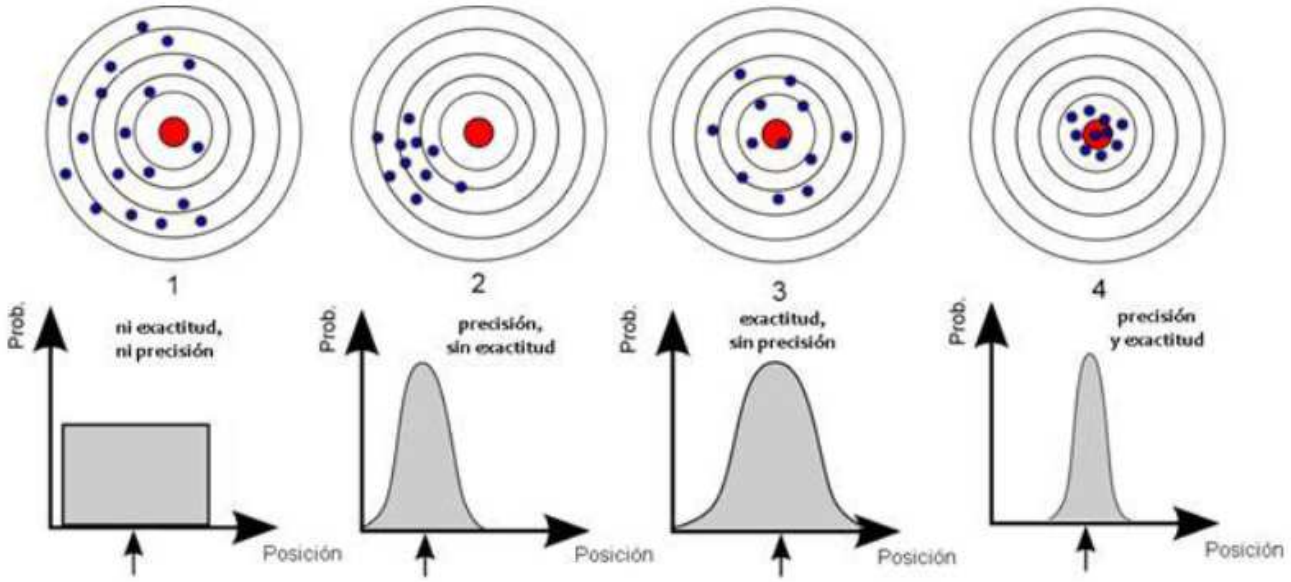
```
#Calculamos exactitud:
print(13+10+18)
print(13+10+18+3+1)
print(41/45)
```

```
In [7]: print(13+10+18)
41

In [8]: print(13+10+18+3+1)
45

In [9]: print(41/45)
0.9111111111111111
```

En una matriz de 3x3 y más los valores obtenidos son la comparativa entre test y las predicciones.



Cálculo de puntuación (score) a nivel de valores de los valores de test

```
#Calculamos la puntuación
knn.score(X_test, y_test)
```

```
In [10]: knn.score(X_test, y_test)
Out[10]: 0.9111111111111111

In [11]:
```

Cálculo de puntuación (score) a nivel de valores de los valores de train

```
#Calculamos la puntuación a nivel train
knn.score(X_train, y_train)
```

```
In [11]: knn.score(X_train, y_train)
Out[11]: 1.0
```

5. Ahora procedemos a generar la cantidad vecinos de 20 y matrices vacías para registrar los grupos de vecinos.

```
#Ahora establecemos un número de vecinos
vecinos = np.arange(1,20)
print(vecinos)

#Crear 2 matrices vacía en base a la cantidad de vecinos
#los valores serán muy pequeños que se asume que son nulos
train_2 = np.empty(len(vecinos))
test_2 = np.empty(len(vecinos))
print(train_2)
print(test_2)
```

```
    ...: print(train_2)
    ...: print(test_2)
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
17 18 19]
[6.23042070e-307 3.56043053e-307 1.60219306e-306
7.56571288e-307
 1.89146896e-307 1.37961302e-306 1.05699242e-307
8.01097889e-307
 1.78020169e-306 7.56601165e-307 1.02359984e-306
1.69118108e-306
 7.56593696e-307 1.69118108e-306 6.89804132e-307
1.69118787e-306
 8.34451503e-308 8.34402698e-308 3.91792476e-317]
[6.23042070e-307 3.56043053e-307 1.60219306e-306
7.56571288e-307
```

6. Ahora procedemos a generar el bucle con los vecinos para entrenamiento como test.

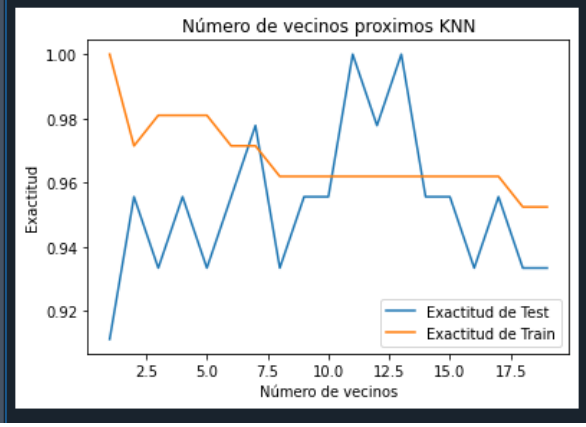
```
#Generamos un bucle para registrar los datos en la
#matrices
for i, k in enumerate(vecinos):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_2[i] = knn.score(X_train, y_train)
    test_2[i] = knn.score(X_test, y_test)

print(train_2)
print(test_2)
```

```
passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  return self._fit(X, y)
[1.         0.97142857 0.98095238 0.98095238 0.98095238 0.97142857
 0.97142857 0.96190476 0.96190476 0.96190476 0.96190476 0.96190476
 0.96190476 0.96190476 0.96190476 0.96190476 0.96190476 0.95238095
 0.95238095]
[0.91111111 0.95555556 0.93333333 0.95555556 0.93333333 0.95555556
 0.97777778 0.93333333 0.95555556 0.95555556 1.         0.97777778
 1.         0.95555556 0.95555556 0.93333333 0.95555556 0.93333333
 0.93333333]
```

7. Ahora vamos a crear un gráfico de vecinos versus test, como vecinos versus train_test

```
plt.title('Número de vecinos proximos KNN')
plt.plot(vecinos, test_2, label='Exactitud de Test')
plt.plot(vecinos, train_2, label='Exactitud de Train')
plt.legend()
plt.xlabel('Número de vecinos')
plt.ylabel('Exactitud')
plt.show()
```



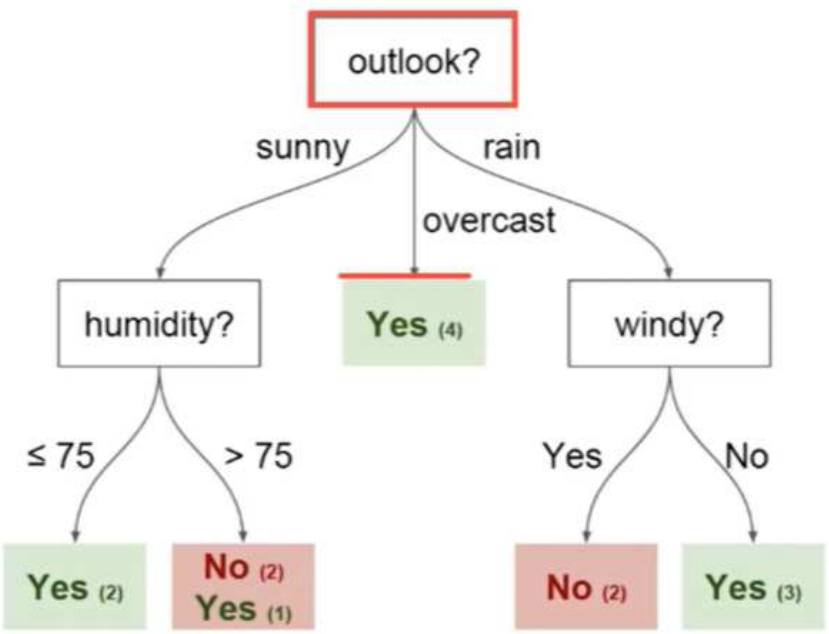
8. **Conclusión**: Los valores obtenidos de KNN-vecinos por test como de entrenamiento, no se ajusta en similitud. Para evaluar con mayores cambios delimitar los valores de la fuente.

ARBOLES DE DECISIÓN

Y BOSQUES ALEATORIOS

Bases Teóricas:

Temperature	Outlook	Humidity	Windy	Played?
Mild	Sunny	80	No	Yes
Hot	Sunny	75	Yes	**No**
Hot	Overcast	77	No	Yes
Cool	Rain	70	No	Yes
Cool	Overcast	72	Yes	Yes
Mild	Sunny	77	No	**No**



Beneficios

Para mejorar el rendimiento:

- Creamos bosques aleatorios
- Utilizaremos parte de las variables en cada árbol para cada corte

A continuación, los pasos para utilizar el árbol de decisión:

1. Importamos las librerías correspondientes y la carga de base de datos similar al instructivo pasado. Preparamos los grupos de entrenamiento y de test.

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

import sklearn

iris = sns.load_dataset("iris")

from sklearn.model_selection import train_test_split

X = iris.drop('species', axis = 1)
y = iris[['species']]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=30)
```

2. Importamos la librería de árboles de decisión para luego entrenar. Finalmente dibujamos el árbol.

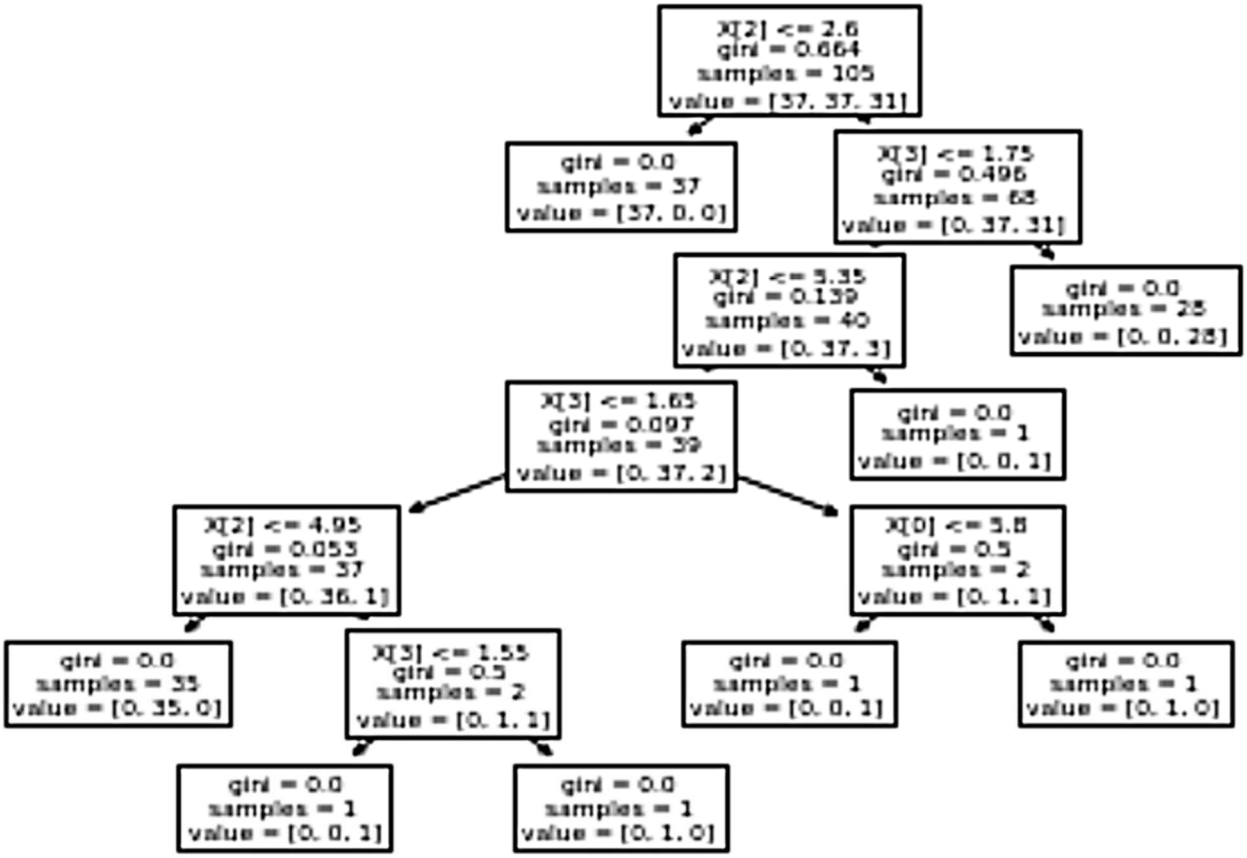
```
from sklearn.tree import DecisionTreeClassifier

arbol = DecisionTreeClassifier()

arbol.fit(X_train, y_train)

#Ahora procedemos a mostrar el Árbol
from sklearn import tree

tree.plot_tree(arbol)
```



3. Como podrá observarse el gráfico es menos nítido, por tanto, vamos a extraer las partes del grafo. Para ello, vamos a generar una lista de los valores X y los valores de las clases de salida. Luego se procede a crear el gráfico en base a los nombres y clases estipuladas como salida.

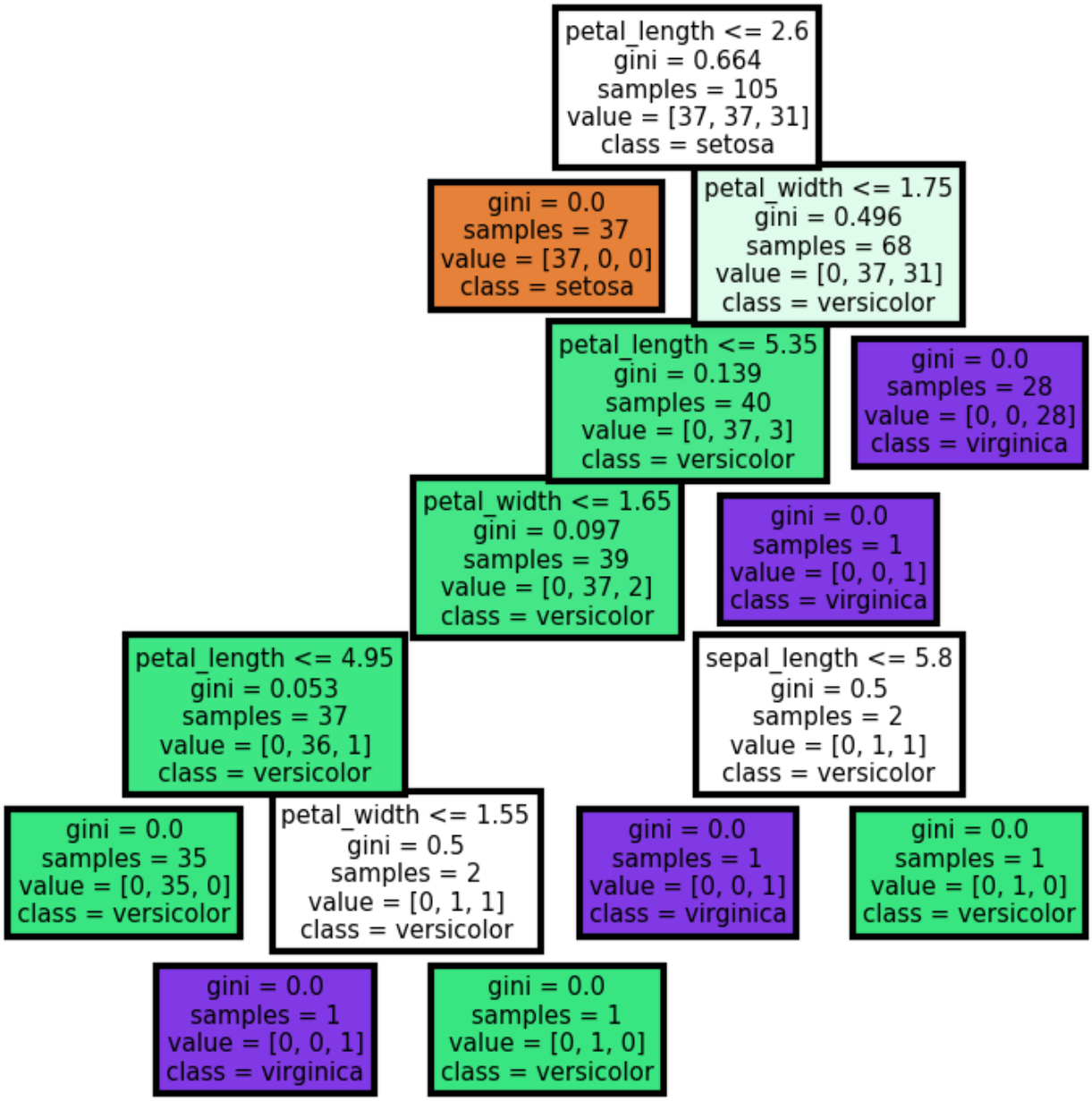
```
X_nombre = list(X.columns)

classes = ['setosa','versicolor','virginica']

fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (3,3), dpi = 300)

tree.plot_tree(arbol, feature_names = X_nombre, class_names= classes, filled = True)

fig.savefig('imagen.png')
```



4. Ahora vamos a crear las predicciones y nuestra matriz de confusión con los datos resultantes.

```
#Creamos las predicciones
pred = arbol.predict(X_test)
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
[[13  0  0]
 [ 0 12  1]
 [ 0  0 19]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      0.92      0.96        13
   virginica       0.95      1.00      0.97        19

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.98        45
weighted avg       0.98      0.98      0.98        45
```

5. Observemos con la misma base de datos y utilizando árboles aleatorios.

```
#Ahora utilizando árboles Aleatorios

from sklearn.ensemble import RandomForestClassifier

#El número 20 es número de bosques aleatorios
rfc = RandomForestClassifier(n_estimators=20, random_state=33)

rfc.fit(X_train, y_train)

rfc_pred = rfc.predict(X_test)

print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

```
[[13  0  0]
 [ 0 12  1]
 [ 0  0 19]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        13
  versicolor       1.00      0.92      0.96        13
   virginica       0.95      1.00      0.97        19

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.98        45
weighted avg       0.98      0.98      0.98        45
```