Importing Necessary libraries

```
import numpy as np
import pandas as pd
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from sklearn import  metrics, model_selection, preprocessing
from sklearn.ensemble import  RandomForestClassifier
from sklearn.tree import export_graphviz
import graphviz
```

```
data = pd.read_csv('/content/car_evaluation.csv')
```

```
data.head()
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	vhigh	vhigh	2	2	small	med	unacc
1	vhigh	vhigh	2	2	small	high	unacc
2	vhigh	vhigh	2	2	med	low	unacc
3	vhigh	vhigh	2	2	med	med	unacc
4	vhigh	vhigh	2	2	med	high	unacc

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   vhigh    1727 non-null   object
 1   vhigh.1  1727 non-null   object
 2   2        1727 non-null   object
 3   2.1      1727 non-null   object
 4   small    1727 non-null   object
 5   low      1727 non-null   object
 6   unacc    1727 non-null   object
dtypes: object(7)
memory usage: 94.6+ KB
```

```
data.columns
```

```
Index(['vhigh', 'vhigh.1', '2', '2.1', 'small', 'low', 'unacc'], dtype='object')
```

Identify the predictor variables and encode any string variables to equivalent integer codes

```
for i in list(data.columns):
  data[i],_ = pd.factorize(data[i])
data.head()
```

	vhigh	vhigh.1	2	2.1	small	low	unacc
0	0	0	0	0	0	0	0
1	0	0	0	0	0	1	0
2	0	0	0	0	1	2	0
3	0	0	0	0	1	0	0
4	0	0	0	0	1	1	0

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1727 entries, 0 to 1726
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   vhigh    1727 non-null   int64
 1   vhigh.1  1727 non-null   int64
 2   2        1727 non-null   int64
 3   2.1      1727 non-null   int64
 4   small    1727 non-null   int64
 5   low      1727 non-null   int64
 6   unacc    1727 non-null   int64
dtypes: int64(7)
memory usage: 94.6 KB
```

Select the predictor feature and select the target variable

```
X = data.iloc[:,:-1]
y = data.iloc[:,-1]
```

Split data randomly into 70% training and 30% test

```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.3,
model = RandomForestClassifier(random_state=1)
model.fit(X_train, y_train)
```

```
▾      RandomForestClassifier    i ?
RandomForestClassifier(random_state=1)
```

```
y_pred = model.predict(X_test)
```

```
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Misclassified samples: 12
Accuracy: 0.98
```

create the classifier with n_estimators = 100

```
clf = RandomForestClassifier(n_estimators=100, random_state=0)
```

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

```
▾      RandomForestClassifier    i ?
RandomForestClassifier(random_state=0)
```

```
feature_scores = pd.Series(clf.feature_importances_, index=X_train.columns).sort_values(a
```

```
feature_scores
```

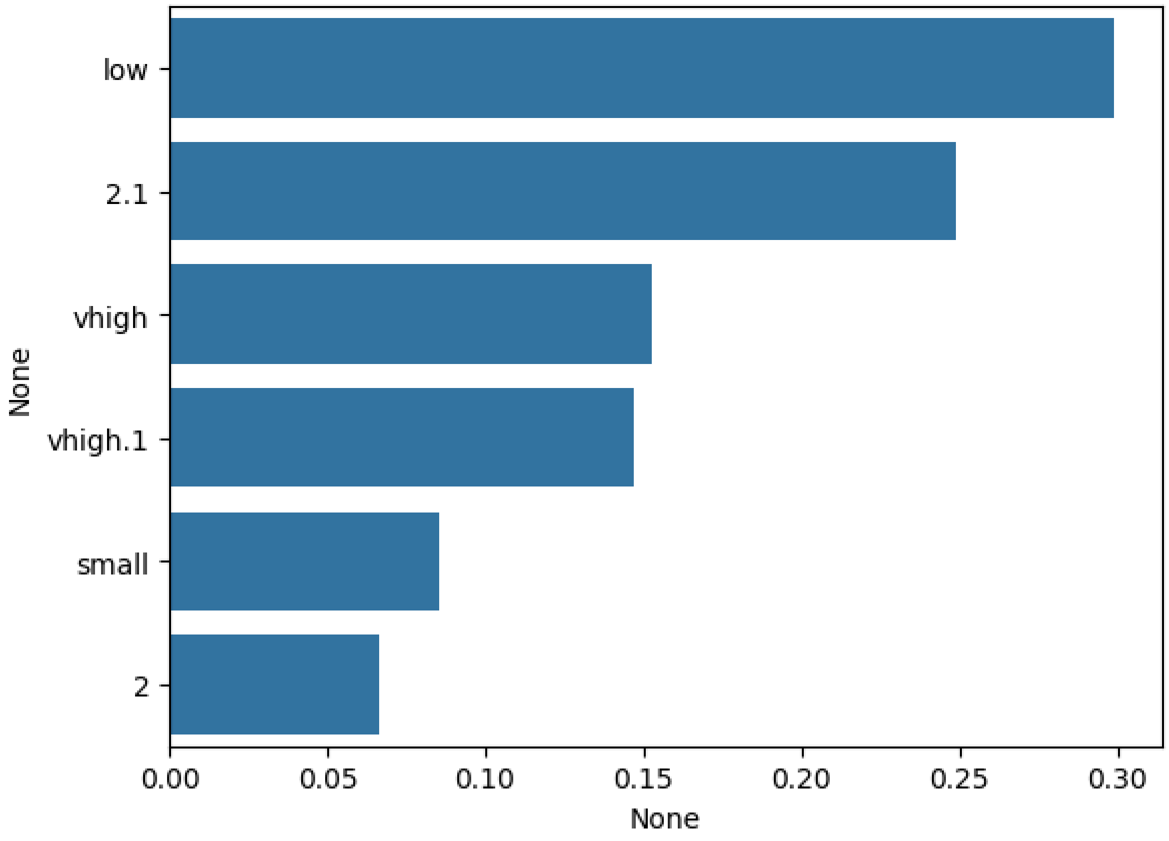
```
                 0
low       0.299280
2.1       0.249282
vhigh     0.152733
vhigh.1   0.146798
small     0.085398
2         0.066508

dtype: float64
```

Creating a seaborn bar plot

```
import seaborn as sns
sns.barplot(x=feature_scores, y=feature_scores.index)
```

```
<Axes: xlabel='None', ylabel='None'>
```



```
plt.xlabel('Feature Importance Score')
```

```
plt.ylabel('Features')
```

Show hidden output

```
plt.title("Visualizing Important Features")
```

```
plt.show()
```

Find the most important feature

```
feature_importances = clf.feature_importances_
best_feature_index = np.argmax(feature_importances)
best_feature_name = X_train.columns[best_feature_index]
```

```
print(f"The best feature is: {best_feature_name}")
```

```
The best feature is: low
```

Create a decision tree using the best feature

```
best_feature_data = X_train.iloc[:, best_feature_index].values
tree_classifier = RandomForestClassifier(n_estimators=1, random_state=0)  # Create a sing
tree_classifier.fit(best_feature_data.reshape(-1, 1), y_train)
```

```
▾                RandomForestClassifier                i ?
RandomForestClassifier(n_estimators=1, random_state=0)
```

Visualize the decision tree

```
dot_data = export_graphviz(tree_classifier.estimators_[0], out_file=None,
                           feature_names=[best_feature_name], class_names=data.cc
                           filled=True, rounded=True, special_characters=True)


graph = graphviz.Source(dot_data)
graph.render("/content/best_feature_tree")
graph.view("best_feature_tree")
```

```
'best_feature_tree.pdf'
```

Start coding or generate with AI.