

```
In [ ]: import lightgbm as lgb
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
```

## Import Dataset

```
In [ ]: car_data = pd.read_csv("data/car.data", header=None, names=["buying",
                                                                    "maint",
                                                                    "doors",
                                                                    "persons",
                                                                    "lug_boot",
                                                                    "safety",
                                                                    "class"])

# Remove persons column because it is not required
del car_data['persons']
car_data
```

```
Out[ ]:
```

	buying	maint	doors	lug_boot	safety	class
0	vhigh	vhigh	2	small	low	unacc
1	vhigh	vhigh	2	small	med	unacc
2	vhigh	vhigh	2	small	high	unacc
3	vhigh	vhigh	2	med	low	unacc
4	vhigh	vhigh	2	med	med	unacc
...	...	...	...	...	...	...
1723	low	low	5more	med	med	good
1724	low	low	5more	med	high	vgood
1725	low	low	5more	big	low	unacc
1726	low	low	5more	big	med	good
1727	low	low	5more	big	high	vgood

1728 rows × 6 columns

## Data Exploration

```
In [ ]: car_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   buying      1728 non-null   object
1   maint       1728 non-null   object
2   doors       1728 non-null   object
3   lug_boot    1728 non-null   object
4   safety      1728 non-null   object
5   class       1728 non-null   object
dtypes: object(6)
memory usage: 81.1+ KB
```

```
In [ ]: # Check for distribution
car_data["buying"].value_counts()
```

```
Out[ ]: vhigh    432
high     432
med      432
low      432
Name: buying, dtype: int64
```

```
In [ ]: # All the columns are categorical
# Apply one-hot encoding
enc_car_data = pd.get_dummies(car_data.drop("buying", axis=1))
enc_car_data.head()
```

```
Out[ ]:   maint_high  maint_low  maint_med  maint_vhigh  doors_2  doors_3  doors_4  doors_5more  lu
0           0           0           0           1         1         0         0           0
1           0           0           0           1         1         0         0           0
2           0           0           0           1         1         0         0           0
3           0           0           0           1         1         0         0           0
4           0           0           0           1         1         0         0           0
```

```
In [ ]: # Encode target variable using Label Encoder
le = LabelEncoder()
le.fit(car_data["buying"])
buying_price = le.transform(car_data["buying"])
```

```
In [ ]: le.classes_
```

```
Out[ ]: array(['high', 'low', 'med', 'vhigh'], dtype=object)
```

```
In [ ]: buying_price
```

```
Out[ ]: array([3, 3, 3, ..., 1, 1, 1])
```

## Splitting dataset

```
In [ ]: x_train, x_test, y_train, y_test = train_test_split(
        enc_car_data, car_data["buying"], test_size=0.2
    )
```

```
In [ ]: x_train.shape, y_train.shape
```

```
Out[ ]: ((1382, 18), (1382,))
```

```
In [ ]: x_test.shape, y_test.shape
```

```
Out[ ]: ((346, 18), (346,))
```

# Training models

## Decision Tree

```
In [ ]: # Hyperparameter tuning
        for depth in [2, 3, 4, 5]:
            dt_model = DecisionTreeClassifier(criterion="gini", max_depth=depth)
            # Cross-validation
            cv_results = cross_validate(dt_model, x_train, y_train, cv=5)
            print(
                f"Max depth: {depth} --- Accuracy results: {round(cv_results['test_score']
                )
```

```
Max depth: 2 --- Accuracy results: 0.3061
Max depth: 3 --- Accuracy results: 0.3162
Max depth: 4 --- Accuracy results: 0.3111
Max depth: 5 --- Accuracy results: 0.296
```

```
In [ ]: # Best model
        dt_model = DecisionTreeClassifier(criterion="gini", max_depth=3)
        dt_model.fit(x_train, y_train)
```

```
Out[ ]: DecisionTreeClassifier(max_depth=3)
```

```
In [ ]: # Check accuracy on test set
        y_pred = dt_model.predict(x_test)
        accuracy = accuracy_score(y_pred, y_test)
        print("Model accuracy score: {0:0.4f}".format(accuracy_score(y_test, y_pred)))
```

```
Model accuracy score: 0.3092
```

```
In [ ]: columns = y_test.unique()
        pd.DataFrame(
            confusion_matrix(y_test, y_pred, labels=columns),
            columns=columns + "_pred",
            index=columns + "_true",
        )
```

```
Out[ ]:
```

	low_pred	vhigh_pred	high_pred	med_pred
<b>low_true</b>	16	52	0	23
<b>vhigh_true</b>	0	62	0	18
<b>high_true</b>	0	57	0	25
<b>med_true</b>	10	54	0	29

## Gradient-boosted decision tree

```
In [ ]:
```

```
# Hyperparameter tuning
best_model = {"accuracy": 0}
for leaves in [2, 3, 4, 6, 8]:
    for lr in [0.5, 0.1, 0.05, 0.01]:
        for n_est in [100, 200, 500]:
            lgb_model = lgb.LGBMClassifier(
                objective="multiclass",
                num_leaves=leaves,
                boosting_type="dart",
                learning_rate=lr,
                n_estimators=n_est,
                num_threads=4,
            )
            cv_results = cross_validate(lgb_model, x_train, y_train, cv=5)
            if cv_results["test_score"].mean() > best_model["accuracy"]:
                best_model["accuracy"] = cv_results["test_score"].mean()
                best_model["leaves"] = leaves
                best_model["lr"] = lr
                best_model["n_est"] = n_est
            print(
                f"Leaves: {leaves}, lr: {lr}, n_est: {n_est} --- Accuracy resu
            )
```

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



Out[ ]:

	low_pred	vhigh_pred	high_pred	med_pred
<b>low_true</b>	19	52	2	18
<b>vhigh_true</b>	0	62	15	3
<b>high_true</b>	0	57	21	4
<b>med_true</b>	12	54	15	12

## Prediction (Answering question)

- Maintenance = High
- Number of doors = 4
- Lug Boot Size = Big
- Safety = High
- Class Value = Good

In [ ]:

```

sample = {
    "maint_high": 1,
    "maint_low": 0,
    "maint_med": 0,
    "maint_vhigh": 0,
    "doors_2": 0,
    "doors_3": 0,
    "doors_4": 1,
    "doors_5more": 0,
    "lug_boot_big": 1,
    "lug_boot_med": 0,
    "lug_boot_small": 0,
    "safety_high": 1,
    "safety_low": 0,
    "safety_med": 0,
    "class_acc": 0,
    "class_good": 1,
    "class_unacc": 0,
    "class_vgood": 0,
}
prediction = lgb_model.predict([list(sample.values())])
print(f"Predicted buying price: {prediction[0]}")

```

Predicted buying price: low