**DURAN DUMAN 190101037**

**Decision Tree Classification Report**

**1- Introduction**

My main goal for this project was to create a decision tree classifier that can determine the acceptability of cars. The dataset I worked with contains different features that have influence on car acceptability, including **price**, **maintenance cost**, **number of doors**, **passenger capacity**, **luggage size**, and **safety**.

The aim for the project was to train a model that could predict car acceptability based on these features. To achieve this, I used a dataset divided into two subsets: one for training the model **(trainDATA.xlsx)** and the other for evaluating its performance **(testDATA.xlsx)**.

**1.1 Features and Target Variable**

***Features (X):***

* Price
* Maintenance cost
* Number of doors
* Passenger capacity
* Luggage size
* Safety

***Target Variable (y):***

* Car acceptability

**1.2 Libraries Used**

I used two main Python libraries for this project:

* **Pandas**: I used Pandas for data manipulation and analysis. Pandas helped me read the dataset from excel files into DataFrames, making it easy to preprocess the data.
* **Numpy**: I used Numpy especially in the calculation of entropy and information gain within the decision tree algorithm.
* import pandas as pd  
  import numpy as np

**2- Data Reading and Preparation:**

To begin, I read the dataset from excel files into pandas DataFrames. I specified column names and then separated the features (X) from the target variable (y) for both the training and test sets.

col\_names = ["Price", "MaintPrice", "NoofDoors", "Persons", "Lug\_size", "Safety", "Car Acceptibility"]  
train\_data = pd.read\_excel(r"trainDATA.xlsx", names=col\_names)  
test\_data = pd.read\_excel(r"testDATA.xlsx", names=col\_names)  
  
X\_train = train\_data.drop("Car Acceptibility", axis=1)  
y\_train = train\_data["Car Acceptibility"]  
X\_test = test\_data.drop("Car Acceptibility", axis=1)  
y\_test = test\_data["Car Acceptibility"]

This separation was a must to enable the model to learn from the features and predict the target variable accurately.

**3- Decision Tree Implementation Details**

In this section, I showed a thorough breakdown of each function in the decision tree implementation.

***3.1 entropy\_hesapla***

# Veri kümesinin entropisini hesaplayan fonksiyon  
def entropy\_hesapla(y):

# kaç farklı class olduğunu ve bu classlardan kaç tane olduğunu hesaplar  
 unique\_classes, class\_counts = np.unique(y, return\_counts=True)

# her classın veri kümesindeki oranını hesaplar  
 probabilities = class\_counts / len(y)

# entropi hesaplar  
 entropy\_value = -np.sum(probabilities \* np.log2(probabilities))   
 return entropy\_value

**Purpose**: Calculates the entropy of a given set of labels (y), representing the uncertainty in the set.

**Process:**

* **unique\_classes, class\_counts = np.unique(y, return\_counts=True):** Calculates the unique classes and their corresponding counts in the input set.
* **probabilities = class\_counts / len(y):** Computes the probabilities of each class.
* **entropy\_value = -np.sum(probabilities \* np.log2(probabilities**)): Calculates the entropy using the standard formula for entropy.

**Output:** Returns the computed entropy value.

***3.2 information\_gain\_hesapla***

# Belirli bir feature için information gain hesaplayan fonksiyon  
def information\_gain\_hesapla(X, y, feature):

# mevcut durumun entropisini hesaplar  
 entropy\_before = entropy\_hesapla(y)

# feature'daki unique değerleri alır  
 unique\_values = np.unique(X[feature])  
 information\_after = 0  
  
 for value in unique\_values: # her bir unique değer için

# feature'ın bu değere sahip olduğu alt kümesi  
 subset\_y = y[X[feature] == value]

# information'ı hesaplar  
 information\_after += len(subset\_y) / len(y) \*

entropy\_hesapla(subset\_y)

# information gain'i hesaplar  
 information\_gain\_value = entropy\_before - information\_after   
 return information\_gain\_value

**Purpose**: Computes the information gain for a specific feature in the dataset.

**Process:**

* **entropy\_before = entropy\_hesapla(y):** Computes the entropy of the dataset before the split.
* **unique\_values = np.unique(X[feature]):** Finds unique values for the specified feature.
* **information \_after = 0**: Initialize the variable to store the information after the split
* ***For each unique value of the feature:***
  + **subset\_y = y[X[feature] == value]:** Selects the subset of target variable values for the current feature value.
  + **information \_after += len(subset\_y) / len(y) \* entropy\_hesapla(subset\_y):** Calculates the information after the split.
* **information\_gain\_value = entropy\_before - information \_after:** Computes the information gain.

**Output:** Returns the information gain value for the specified feature.

***3.3 find\_best\_split***

# En iyi bölünme noktasını bulan fonksiyon (Gain'e bakarak)  
def find\_best\_split(X, y):

best\_feature = None # en iyi feature için başlangıç değeri

# en iyi information gain için başlangıç değeri  
 best\_information\_gain = 0

# verilen her bir feature için information gain hesaplar  
 for feature in X.columns:

current\_information\_gain = information\_gain\_hesapla(X, y, feature)

#current information gain, şu ana kadar en iyisiyse best feature'ı

güncelle  
 if current\_information\_gain > best\_information\_gain:   
 best\_information\_gain = current\_information\_gain  
 best\_feature = feature  
  
 return best\_feature # best feature'ı döndürür

**Purpose:** It Identifies the best feature to split on based on the maximum information gain.

**Process:**

* **best\_feature = None:** Initializes the best feature as None.
* **best\_information\_gain = 0:** Initializes the best information gain as 0.
* ***For each feature in the feature matrix:***
  + **current\_information\_gain = information\_gain\_hesapla(X, y, feature):** Calculates the information gain for the current feature.
  + Updates **best\_feature** and **best\_information**\_gain if the current information gain is higher.
* **Output: Returns the best feature for splitting.**

***3.4 id3\_build\_tree***

# ID3 Algoritmasını kullanarak Decision Tree (Karar Ağacı) oluşturan fonksiyon.  
def id3\_build\_tree(X, y):

# eğer veri kümesinde yalnızca bir class varsa, bir leaf node oluştur

ve class'ı döndür  
 if len(np.unique(y)) == 1:   
 return {'class': y.iloc[0]}

# best feature'u bulur (gain'e göre)

best\_feature = find\_best\_split(X, y)

# eğer best feature'u yoksa, bir leaf node oluştur ve en çok tekrar

eden class'ı döndür  
 if best\_feature is None:  
 return {'class': np.argmax(np.bincount(y))}  
  
 tree = {'feature': best\_feature, 'branches': {}} # ağacı oluştur

# best feature'un her bir unique değeri için sub tree oluşturur  
 for value in np.unique(X[best\_feature]):  
 subset\_X = X[X[best\_feature] == value].drop(best\_feature, axis=1)  
 subset\_y = y[X[best\_feature] == value]

# sub tree oluşturur ve ana tree'nin alt dallarına ekler (recursive

devam eder işlemler)  
 tree['branches'][value] = id3\_build\_tree(subset\_X, subset\_y)  
  
 return tree

**Process:**

* If all samples have the same label, a leaf node is created with the class label.
* **best\_feature = find\_best\_split(X, y):** Finds the best feature for splitting.
* If no suitable feature is found for splitting, a leaf node is created with the most frequent class label in the current subset.
* Initializes a tree dictionary with the best feature and an empty 'branches' dictionary.
* *For each unique value of the best feature:*
* Subset the data where the selected feature has the current unique value.
* Recursively call id3\_build\_tree on the subset of data for that feature value.
* Populate the 'branches' dictionary with the feature value as the key and the recursively constructed subtree as the value.

**Output**: Returns the constructed decision tree.

***3.5 predict***

# Karar ağacını kullanarak bir test data için tahmin yapar.  
def predict(tree, sample):

# eğer bu düğüm bir leaf node ise, class'ı döndür  
 if 'class' in tree:   
 return tree['class']  
  
 feature\_value = sample[tree['feature']]

# feature'ın değeri ağaçta yoksa, "None" döndür  
 if feature\_value not in tree['branches']:   
 return None  
  
 branch = tree['branches'][feature\_value]

# tahminler ağacın geri kalanı için recursive şekilde devam eder  
 return predict(branch, sample)

**Process:**

* If the current node is a leaf node, returns the class label.
* Checks the value of the feature in the current node of the tree.
* If the feature value is not in the branches, returns “None” indicating that the algorithm cannot make a prediction for this specific path in the tree.
* Recursively calls predict with the subtree corresponding to the feature value.

**Output:** Returns the predicted class label for the input sample.

***3.6 accuracy\_hesapla***

# Doğruluk hesaplayan fonksiyon  
def accuracy\_hesapla(predictions, actual):

# doğru tahmin sayısını bulur  
 correct\_predictions = np.sum(predictions == actual)

# toplam örnek sayısını bulur  
 total\_samples = len(actual)

# doğruluk yüzdesini hesaplar  
 accuracy\_percentage = (correct\_predictions / total\_samples) \* 100

return accuracy\_percentage

**Process:**

* **correct\_predictions = np.sum(predictions == actual):** Counts the number of correct predictions.
* **total\_samples = len(actual):** Computes the total number of samples.
* **accuracy\_percentage = (correct\_predictions / total\_samples) \* 100:** Calculates the accuracy percentage.

**Output:** Returns the accuracy percentage.

**3.7 print\_tree**

def print\_tree(tree, indent=0):  
 if "class" in tree:  
 print("|\_\_ Sınıf:", tree["class"])  
 else:  
 print("|\_\_ Özellik:", tree["feature"])  
 for value, subtree in tree["branches"].items():  
 print("| " \* indent + "|\_\_ {}: ".format(value), end="")  
 print\_tree(subtree, indent + 1)

**Process:**

* If the current node is a leaf node, prints the class label.
* If the current node is a decision node, prints the feature and iterates through branches.
* Recursively calls print\_tree for each branch with increased indentation.

**Output:** Prints the structure of the decision tree.

**3.8 Results Handling and Reporting**

# Karar ağacını oluşturur.  
tree = id3\_build\_tree(X\_train, y\_train)  
  
# Test veri seti için tahminler yapar.  
predictions = []  
for \_, sample in X\_test.iterrows():  
 predictions.append(predict(tree, sample))

# Tahminleri bir DataFrame'e dönüştürür.  
result\_df = pd.DataFrame({'Prediction': predictions})  
  
# Tahmin sonuçlarını "tahminler.xlsx" dosyasına yazar.  
result\_df.to\_excel("tahminler.xlsx", index=False)

# Ağacı printler  
print\_tree(tree)  
  
# Test data ve Tahminler arasındaki doğruluk değerini hesaplar.  
accuracy = accuracy\_hesapla(predictions, y\_test)  
print(f"Accuracy: {accuracy}")

* The decision tree is constructed using the training data.
* **Predictions** are made on the test set using the predict function.
* **Predictions** are stored in a **Pandas DataFrame** for further analysis.

**Results Writing:** The DataFrame containing predictions is written to an Excel file named ***"tahminler.xlsx"*** using the to\_excel method. **In that Excel file there are null predictions, those are the one that are returned as None when classifying. (In the code, you can see that part in the “predict” function.)**

**Accuracy Calculation:** The accuracy of the model is calculated using the accuracy\_hesapla function.

**Printing Tree Structure:** The structure of the decision tree is printed to the console using the print\_tree function.

**4 - Accuracy Calculation and Results**

The accuracy of the decision tree classifier on the test set was determined to be approximately 80.95%. This accuracy metric is a crucial measure of the model's performance.

**Accuracy Calculation:**

The accuracy was calculated using the accuracy\_hesapla function:

# Doğruluk hesaplar.  
accuracy = accuracy\_hesapla(predictions, y\_test)  
print(f"Doğruluk: {accuracy}")

The accuracy\_hesapla function takes the predicted labels (predictions) and the actual labels (y\_test) as inputs. It then computes the percentage of correctly predicted instances and prints the accuracy to the console.

**Results:**

Doğruluk: 80.95238095238095

An accuracy of 80.95% shows that the decision tree model performed well in classifying car acceptability based on the provided features. This means that, in approximately 80.95% of the cases, the model correctly predicted whether a car was acceptable or not.

In summary, an accuracy of 80.95% indicates an effective decision tree model for the given dataset.

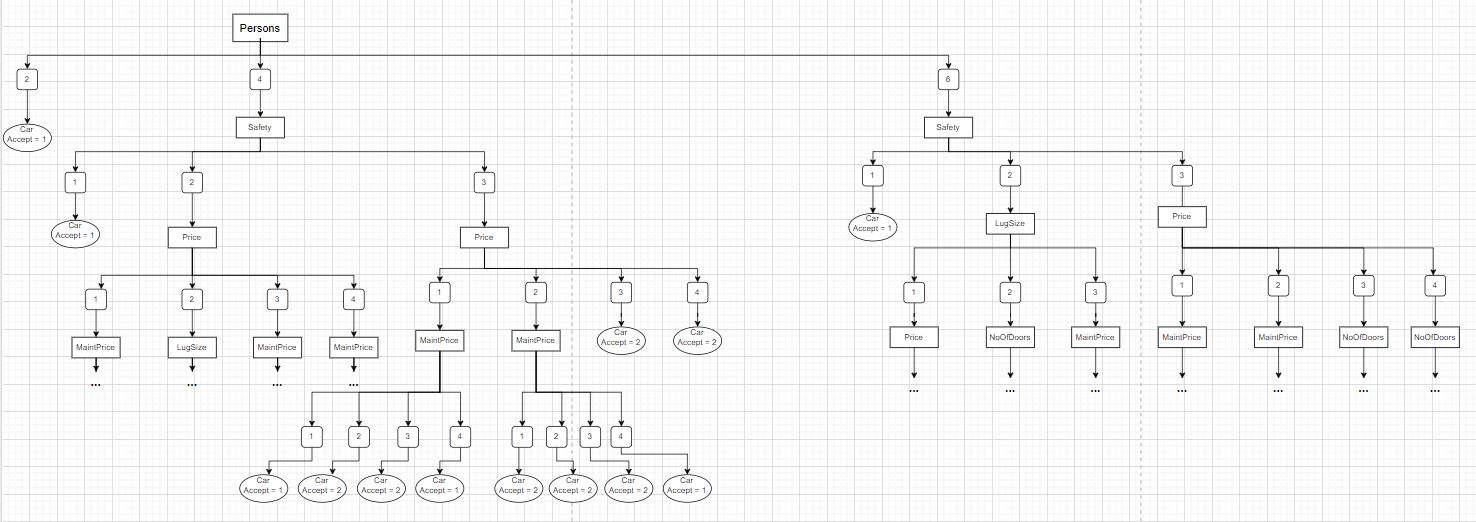
**Note : “tahminler.xlsx contains the predictions in the project file”**

**5 -Decision Tree Visualization**

To visualize the tree, I used this function :

# Karar ağacını çizen fonksiyon  
def print\_tree(tree, indent=0):  
 if "class" in tree:  
 print("|\_\_ Sınıf:", tree["class"])  
 else:  
 print("|\_\_ Özellik:", tree["feature"])  
 for value, subtree in tree["branches"].items():  
 print("| " \* indent + "|\_\_ {}: ".format(value), end="")  
 print\_tree(subtree, indent + 1)

**To see the some levels of decision tree please zoom in to the picture provided. (If you want to see from jpg format, you can find the jpg file in the project zip)**



|\_\_ Özellik: Persons  
|\_\_ 2: |\_\_ Sınıf: 1  
|\_\_ 4: |\_\_ Özellik: Safety  
| |\_\_ 1: |\_\_ Sınıf: 1  
| |\_\_ 2: |\_\_ Özellik: Price  
| | |\_\_ 1: |\_\_ Özellik: MaintPrice  
| | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Özellik: Lug\_size  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | | | |\_\_ 3: |\_\_ Sınıf: 1  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Sınıf: 1  
| | |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | |\_\_ 2: |\_\_ Özellik: MaintPrice  
| | | | |\_\_ 1: |\_\_ Özellik: NoofDoors  
| | | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | | |\_\_ 3: |\_\_ Sınıf: 1  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Özellik: NoofDoors  
| | | | | |\_\_ 3: |\_\_ Sınıf: 1  
| | | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 1  
| | | |\_\_ 3: |\_\_ Özellik: MaintPrice  
| | | | |\_\_ 1: |\_\_ Sınıf: 2  
| | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 1  
| | |\_\_ 3: |\_\_ Özellik: MaintPrice  
| | | |\_\_ 1: |\_\_ Sınıf: 2  
| | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Özellik: Lug\_size  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | | | |\_\_ 3: |\_\_ Sınıf: 1  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | |\_\_ 4: |\_\_ Özellik: MaintPrice  
| | | |\_\_ 1: |\_\_ Özellik: Lug\_size  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | | | |\_\_ 3: |\_\_ Sınıf: 1  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Özellik: Lug\_size  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| |\_\_ 3: |\_\_ Özellik: Price  
| | |\_\_ 1: |\_\_ Özellik: MaintPrice  
| | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Sınıf: 1  
| | |\_\_ 2: |\_\_ Özellik: MaintPrice  
| | | |\_\_ 1: |\_\_ Sınıf: 2  
| | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Sınıf: 1  
| | |\_\_ 3: |\_\_ Sınıf: 2  
| | |\_\_ 4: |\_\_ Sınıf: 2  
|\_\_ 6: |\_\_ Özellik: Safety  
| |\_\_ 1: |\_\_ Sınıf: 1  
| |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | |\_\_ 1: |\_\_ Özellik: Price  
| | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | |\_\_ 3: |\_\_ Özellik: MaintPrice  
| | | | |\_\_ 1: |\_\_ Sınıf: 2  
| | | | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 1  
| | | |\_\_ 4: |\_\_ Özellik: MaintPrice  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 1  
| | | | |\_\_ 4: |\_\_ Sınıf: 1  
| | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | |\_\_ 2: |\_\_ Özellik: Price  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | |\_\_ 3: |\_\_ Sınıf: 1  
| | | | |\_\_ 4: |\_\_ Özellik: MaintPrice  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | | |\_\_ 4: |\_\_ Sınıf: 1  
| | | |\_\_ 3: |\_\_ Özellik: MaintPrice  
| | | | |\_\_ 1: |\_\_ Özellik: Price  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Özellik: Price  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Özellik: Price  
| | | | |\_\_ 1: |\_\_ Özellik: MaintPrice  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | | |\_\_ 4: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | |\_\_ 5: |\_\_ Özellik: Price  
| | | | |\_\_ 1: |\_\_ Özellik: MaintPrice  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | | |\_\_ 4: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Özellik: MaintPrice  
| | | | | |\_\_ 1: |\_\_ Sınıf: 2  
| | | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | | |\_\_ 4: |\_\_ Sınıf: 1  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | |\_\_ 3: |\_\_ Özellik: MaintPrice  
| | | |\_\_ 1: |\_\_ Özellik: Price  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Özellik: Price  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| |\_\_ 3: |\_\_ Özellik: Price  
| | |\_\_ 1: |\_\_ Özellik: MaintPrice  
| | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | | |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Özellik: NoofDoors  
| | | | |\_\_ 2: |\_\_ Sınıf: 1  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Sınıf: 1  
| | |\_\_ 2: |\_\_ Özellik: MaintPrice  
| | | |\_\_ 1: |\_\_ Özellik: NoofDoors  
| | | | |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | |\_\_ 2: |\_\_ Özellik: NoofDoors  
| | | | |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Özellik: NoofDoors  
| | | | |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | | |\_\_ 5: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Sınıf: 1  
| | |\_\_ 3: |\_\_ Özellik: NoofDoors  
| | | |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | |\_\_ 5: |\_\_ Sınıf: 2  
| | |\_\_ 4: |\_\_ Özellik: NoofDoors  
| | | |\_\_ 2: |\_\_ Özellik: Lug\_size  
| | | | |\_\_ 1: |\_\_ Sınıf: 1  
| | | | |\_\_ 2: |\_\_ Sınıf: 2  
| | | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 3: |\_\_ Sınıf: 2  
| | | |\_\_ 4: |\_\_ Sınıf: 2  
| | | |\_\_ 5: |\_\_ Sınıf: 2