Feature Transformation**

Encoding Categorical Data

Technique of converting categorical variables into numerical values, so that it could be easily fitted to a machine learning model.

There are two types of catgorical variables:

1. Ordinal categorical variables-->Ordinal Encoding
2. Nominal categorical variables-->One-Hot Encoding

Note : for encoding labelled(output) feature we use Label Encoding.

Nominal Encoding

Nominal Encoding is applied on nominal categorical input feature. As a reslt of it new columns are generated for these features which are known as dummy variables.

OneHot Encoding

Topics covered :

1. OneHotEncoding using Pandas
2. K-1 OneHotEncoding
3. OneHotEncoding using Sklearn
4. OneHotEncoding with Top Categories

Importing Dependencies

In [16]:

```
import pandas as pd
import numpy as np
```

In [17]:

```
#load data
df1=pd.read_csv('cars.csv')
#read first 5 rows
df1.head()
```

Out[17]:

	brand	km_driven	fuel	owner	selling_price
0	Maruti	145500	Diesel	First Owner	450000
1	Skoda	120000	Diesel	Second Owner	370000
2	Honda	140000	Petrol	Third Owner	158000
3	Hyundai	127000	Diesel	First Owner	225000
4	Maruti	120000	Petrol	First Owner	130000

In [18]:

```
#shape
df1.shape
```

Out[18]: (8128, 5)

In [19]:

```
#information of data
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8128 entries, 0 to 8127
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   brand          8128 non-null   object
 1   km_driven      8128 non-null   int64 
 2   fuel           8128 non-null   object
```

```
 3   owner          8128 non-null   object
 4   selling_price  8128 non-null   int64
dtypes: int64(2), object(3)
memory usage: 317.6+ KB
```

In [20]:

```python
#number of unique catgories in nominal categorical features
for fea in df1.columns:
    if(df1[fea].dtype=='O'):
        print(f'{fea} --> {df1[fea].nunique()}')
```

```
brand --> 32
fuel --> 4
owner --> 5
```

In [21]:

```python
#number of values for each category of 'fuel'
df1['fuel'].value_counts()
```

Out[21]:

```
Diesel    4402
Petrol    3631
CNG         57
LPG         38
Name: fuel, dtype: int64
```

In [22]:

```python
#number of values for each category of 'owner'
df1['owner'].value_counts()
```

Out[22]:

```
First Owner             5289
Second Owner            2105
Third Owner              555
Fourth & Above Owner     174
Test Drive Car             5
Name: owner, dtype: int64
```

OneHotEncoding using Pandas

Number of columns formed = Number of categories in particular feature

In [23]:

```python
#Creating Dummy Variables using pandas
pd.get_dummies(df1, columns=['fuel', 'owner'])
#Observation: 4-dummy variables for 'fuel'(as there are 4 categories for 'fuel')
#Observation: 5-dummy variables for 'owner'(as there are 5 categories for 'owner')
#number of columns=(4+5)=9
```

Out[23]:

	brand	km_driven	selling_price	fuel_CNG	fuel_Diesel	fuel_LPG	fuel_Petrol	owner_First Owner	owner_Fourth & Above Owner	owner_Second Owner	owner_Test Drive Car
0	Maruti	145500	450000	0	1	0	0	1	0	0	0
1	Skoda	120000	370000	0	1	0	0	0	0	1	0
2	Honda	140000	158000	0	0	0	1	0	0	0	0
3	Hyundai	127000	225000	0	1	0	0	1	0	0	0
4	Maruti	120000	130000	0	0	0	1	1	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
8123	Hyundai	110000	320000	0	0	0	1	1	0	0	0
8124	Hyundai	119000	135000	0	1	0	0	0	1	0	0
8125	Maruti	120000	382000	0	1	0	0	1	0	0	0
8126	Tata	25000	290000	0	1	0	0	1	0	0	0
8127	Tata	25000	290000	0	1	0	0	1	0	0	0

8128 rows × 12 columns

K-1 OneHotEncoding

Dummy variable trap : after one-hot encoding we drop first(or any) column, means if after one-hot encoding we got n-columns for any feature then we drop any-one of them and we are left with (n-1) columns

Multi-collinearity : input features(columns) must be independent(inter-dependence) to each other, but after one-hot encoding columns formed from encoded feature have relationship(sum of all columns for encoded feature = 1), and which creates an issue when algorithms(say linear regression or logistic regression) are applied o this data. And to overcome this issue we drop one of them, and the column that is dropped that category will have all values = 0(we can see in data below.)

Because of these dummy variables, there occured an issue of Multi-collinearity, that's why this is called Dummy Variable trap.

Why k-1 : because out of K-categories we are dropping 1.

K-1 OneHot Encoding : Number of columns formed = Number of categories in particular feature -1

In [24]:
```
#Creating Dummy Variables using pandas-->but dropped first column(due to multi-collinearity)
pd.get_dummies(df1, columns=['fuel', 'owner'], drop_first=True)
#Observation: 3-dummy variables for 'fuel'(as there are 4 categories for 'fuel')-->n-1
#Observation: 4-dummy variables for 'owner'(as there are 5 categories for 'owner')-->n-1
#number of new columns-->(4+5-2)=7
```

Out[24]:

	brand	km_driven	selling_price	fuel_Diesel	fuel_LPG	fuel_Petrol	owner_Fourth & Above Owner	owner_Second Owner	owner_Test Drive Car	owner_Third Owner
0	Maruti	145500	450000	1	0	0	0	0	0	0
1	Skoda	120000	370000	1	0	0	0	1	0	0
2	Honda	140000	158000	0	0	1	0	0	0	1
3	Hyundai	127000	225000	1	0	0	0	0	0	0
4	Maruti	120000	130000	0	0	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
8123	Hyundai	110000	320000	0	0	1	0	0	0	0
8124	Hyundai	119000	135000	1	0	0	1	0	0	0
8125	Maruti	120000	382000	1	0	0	0	0	0	0
8126	Tata	25000	290000	1	0	0	0	0	0	0
8127	Tata	25000	290000	1	0	0	0	0	0	0

8128 rows × 10 columns

OneHot Encoding using Sklearn

Why not to use Pandas?

Because Pandas don't remember which column was placed at what position, and we cannot apply this in Machine Learning projects, so we use class OneHotEncoder of sklearn.

In [25]:
```
#recommended to split data into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df1.iloc[:, :4], df1.iloc[:, -1], test_size=0.2, random_state
```

In [26]:
```
#OneHotEncoder
from sklearn.preprocessing import OneHotEncoder
#create object for OneHotEncoder(by-default OneHotEncoder creates Sparse matrix so for sparse=False it will conve
ohe=OneHotEncoder(drop='first',sparse=False,dtype=np.int32)
#
X_train_new = ohe.fit_transform(X_train[['fuel','owner']])
X_test_new = ohe.transform(X_test[['fuel','owner']])

#observation: since we have applied onehotencoding for 'fuel' and 'owner' only,
#so it after encoding it will create an array with dummy variable for these two features only
#so, we have to add rest of the features to this resulting array(dataframe), to do it all in one-step we use Colu
#which will be studied later...
```

In [27]:
```
#shape of train and test
X_train_new.shape, X_test_new.shape
```

Out[27]: ((6502, 7), (1626, 7))

In [28]:

In [28]:
```
#X_train ['fuel', 'owner'] before encoding
X_train[['fuel', 'owner']].head()
```

Out[28]:

	fuel	owner
5571	Diesel	First Owner
2038	Diesel	First Owner
2957	Petrol	First Owner
7618	Diesel	Second Owner
6684	Diesel	First Owner

In [29]:
```
#X_train ['fuel', 'owner'] after encoding
X_train_new
```

Out[29]:
```
array([[1, 0, 0, ..., 0, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       ...,
       [0, 0, 1, ..., 0, 0, 0],
       [1, 0, 0, ..., 1, 0, 0],
       [1, 0, 0, ..., 0, 0, 0]])
```

In [30]:
```
#X_test ['fuel', 'owner'] before encoding
X_test[['fuel', 'owner']].head()
```

Out[30]:

	fuel	owner
606	Petrol	First Owner
7575	Diesel	Second Owner
7705	Petrol	First Owner
4305	Petrol	Second Owner
2685	Diesel	Second Owner

In [31]:
```
#X_test ['fuel', 'owner'] after encoding
X_test_new
```

Out[31]:
```
array([[0, 0, 1, ..., 0, 0, 0],
       [1, 0, 0, ..., 1, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       ...,
       [0, 0, 1, ..., 0, 0, 0],
       [0, 0, 1, ..., 1, 0, 0],
       [1, 0, 0, ..., 0, 0, 0]])
```

In [34]:
```
#Complete X_train before encoding
X_train.head()
```

Out[34]:

	brand	km_driven	fuel	owner
5571	Hyundai	35000	Diesel	First Owner
2038	Jeep	60000	Diesel	First Owner
2957	Hyundai	25000	Petrol	First Owner
7618	Mahindra	130000	Diesel	Second Owner
6684	Hyundai	155000	Diesel	First Owner

In [32]:
```
#first covert this dataframe with rest features into array
X_train[['brand','km_driven']].values
```

Out[32]:
```
array([['Hyundai', 35000],
       ['Jeep', 60000],
       ['Hyundai', 25000],
       ...,
```

```
       ['Tata', 15000],
       ['Maruti', 32500],
       ['Isuzu', 121000]], dtype=object)
```

In [37]:
```
#combine X_train ['brand','km_driven'] array with X_train ['fuel', 'owner']
#hstack-->horizontally stack
X_train= np.hstack((X_train[['brand','km_driven']].values,X_train_new))
X_train
```

Out[37]:
```
array([['Hyundai', 35000, 1, ..., 0, 0, 0],
       ['Jeep', 60000, 1, ..., 0, 0, 0],
       ['Hyundai', 25000, 0, ..., 0, 0, 0],
       ...,
       ['Tata', 15000, 0, ..., 0, 0, 0],
       ['Maruti', 32500, 1, ..., 1, 0, 0],
       ['Isuzu', 121000, 1, ..., 0, 0, 0]], dtype=object)
```

In [39]:
```
#X_train = pd.DataFrame(X_train, columns=?)
```

OneHotEncoding with Top Categories

what if we have many categories for a nominal feature?? After applying One-Hot Encoding as it is, there will be created as many dummy variables as there are categories in a feature and as a result dimensionality of data will increase a lot resulting in slow processing.

Solution-->we create dummy variables for most frequent categories, and transform rest of all categories into new category(say, others).

Note : this technique is used when there is difference in frequencies of categories in a particular feature.

In [41]:
```
#number of unique categories in 'brands'
df1['brand'].nunique()
#observation: oh! 32, its huge, we can't create 32/31 dummy variables, so we will use most frequent categories or
```

Out[41]:
```
32
```

In [42]:
```
#number of values for each category of 'brand'
df1['brand'].value_counts()
```

Out[42]:
```
Maruti           2448
Hyundai          1415
Mahindra          772
Tata              734
Toyota            488
Honda             467
Ford              397
Chevrolet         230
Renault           228
Volkswagen        186
BMW               120
Skoda             105
Nissan             81
Jaguar             71
Volvo              67
Datsun             65
Mercedes-Benz      54
Fiat               47
Audi               40
Lexus              34
Jeep               31
Mitsubishi         14
Force               6
Land                6
Isuzu               5
Kia                 4
Ambassador          4
Daewoo              3
MG                  3
Peugeot             1
Opel                1
```

```
Ashok                1
Name: brand, dtype: int64
```

In [48]:

```
#create variable count which stores counts of all categories of 'brand'
counts=df1['brand'].value_counts()
#set, threshold=100
threshold=100
#so we will create a threshold(say, 100), so extract names of categories(index) with counts<=100
repl = counts[counts <= threshold].index
```

In [51]:

```
#replace those categories(with counts<=thersold) with name say 'uncommon' and get dummy columns(for now use Numpy
pd.get_dummies(df1['brand'].replace(repl, 'uncommon')).sample(5)
```

Out[51]:

	BMW	Chevrolet	Ford	Honda	Hyundai	Mahindra	Maruti	Renault	Skoda	Tata	Toyota	Volkswagen	uncommon
0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0

END of Document.