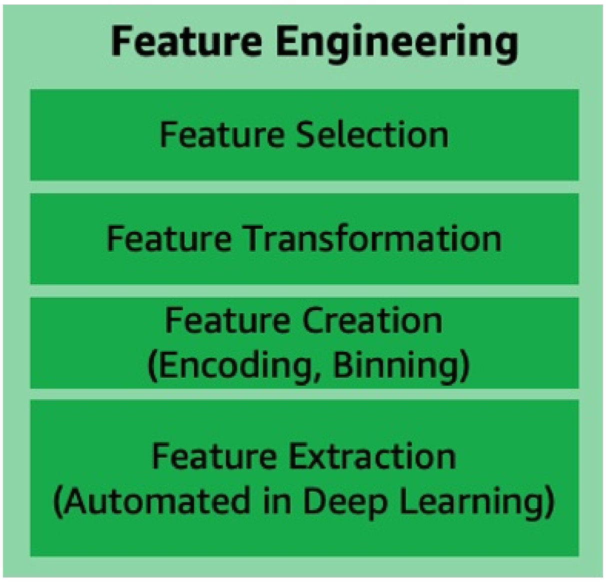
Encoding Categorical variable - Ordinal encoding and One Hot Encoding

ML lifecycle:

1. Frame the problem
2. Gathering the data/importing the data from various sources
3. Data Preprocessing
4. Exploratory Data Analysis
5. Feature Engineering and Selection
6. Model Training and Evaluation
7. Model Deployment
8. Testing
9. Optimize



Data types:

1. Qualitative - Nominal & ordinal data
2. Quantitative - Interval & ration

Types of data on the basis of measurement

Scale	True Zero	Equal Intervals	Order	Category	Example
Nominal	No	No	No	Yes	Marital Status, Sex, Gender, Ethnicity
Ordinal	No	No	Yes	Yes	Student Letter Grade, NFL Team Rankings
Interval	No	Yes	Yes	Yes	Temperature in Fahrenheit, SAT Scores, IQ, Year
Ratio	Yes	Yes	Yes	Yes	Age, Height, Weight

EXAMPLE:

In [1]:
```python
import numpy as np
import pandas as pd
```

In [2]:
```python
df=pd.read_csv(r"C:\Users\USER\Downloads\customer.csv")
df.sample(5)
```

Out[2]:

	age	gender	review	education	purchased
38	45	Female	Good	School	No
5	31	Female	Average	School	Yes
6	18	Male	Good	School	No
16	59	Male	Poor	UG	Yes
27	69	Female	Poor	PG	No

NOTE: Here in above data, what we can see - age,gender,review,eduaction and even target column purchased or not is also categorical column.

Now check : which is nominal and which is ordinal? - Nominal(gender) and ordinal(review and education)

Currently, we are focusing on ordinal encoding and label encoding so for now, lets ignore gender because here we have to use one hot encoding or we can use column transfer for all column at same time by creating pipelines.

In [3]:
```python
df1 = df[['review', 'education', 'purchased']]
```

In [4]:
```python
df1.head()
```

Out[4]:

	review	education	purchased
0	Average	School	No
1	Poor	UG	No
2	Good	PG	No
3	Good	PG	No
4	Average	UG	No

Always note, before doing any feature engineering, firstly, we have to do train_test_split

In [5]:
```python
X=df1.iloc[:,0:2]
```

In [6]:
```python
print(X)
```

```
     review education
0   Average    School
1      Poor        UG
2      Good        PG
3      Good        PG
4   Average        UG
5   Average    School
6      Good    School
7      Poor    School
8   Average        UG
9      Good        UG
10     Good        UG
11     Good        UG
12     Poor    School
13  Average    School
14     Poor        PG
15     Poor        UG
16     Poor        UG
17     Poor        UG
18     Good    School
19     Poor        PG
20  Average    School
21  Average        PG
22     Poor        PG
23     Good    School
24  Average        PG
25     Good    School
26     Poor        PG
27     Poor        PG
28     Poor    School
29  Average        UG
30  Average        UG
31     Poor    School
32  Average        UG
33     Good        PG
34  Average    School
35     Poor    School
36     Good        UG
37  Average        PG
38     Good    School
39     Poor        PG
40     Good    School
41     Good        PG
42     Good        PG
43     Poor        PG
44  Average        UG
45     Poor        PG
46     Poor        PG
47     Good        PG
48     Good        UG
49     Good        UG
```

In [7]:
```
y=df1.iloc[:,-1]
```

In [8]:
```
from sklearn.model_selection import train_test_split
```

In [9]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2)
```

In [10]:
```
from sklearn.preprocessing import OrdinalEncoder
```

Now,create object for OrdinalEncoder class and pass list from column where we want to do ordinal encoding.

In [11]:
```
oe = OrdinalEncoder(categories=[['Poor','Average','Good'],['School','UG','PG']])
```

NOTE: In ML, we always fit on X_train and later we transform on both X_train & X_test

In [12]:
```
oe.fit(X_train)
```

Out[12]:
```
OrdinalEncoder
OrdinalEncoder(categories=[['Poor', 'Average', 'Good'], ['School', 'UG', 'PG']])
```

In [13]:
```
X_train = oe.transform(X_train)
X_test = oe.transform(X_test)
```

In [14]:
```
#lets see values
X_train
```

Out[14]:
```
array([[2., 0.],
       [1., 0.],
       [0., 1.],
       [0., 0.],
       [1., 0.],
       [2., 2.],
       [2., 0.],
       [2., 1.],
       [0., 0.],
       [0., 2.],
       [0., 2.],
       [2., 2.],
       [2., 2.],
       [2., 0.],
       [0., 2.],
       [0., 1.],
       [0., 1.],
       [0., 1.],
       [2., 1.],
       [1., 2.],
       [0., 2.],
       [2., 1.],
       [2., 0.],
       [2., 0.],
       [1., 2.],
       [0., 2.],
       [0., 2.],
       [2., 2.],
       [0., 0.],
       [2., 2.],
       [1., 1.],
       [2., 2.],
       [0., 2.],
       [1., 0.],
       [1., 0.],
       [2., 1.],
       [1., 0.],
       [2., 1.],
       [0., 0.],
       [0., 2.]])
```

Similarly, lets do label encoding for target column

In [15]:
```
from sklearn.preprocessing import LabelEncoder
```

In [16]:
```
le = LabelEncoder()
```

In [17]:
```
le.fit(y_train)
```

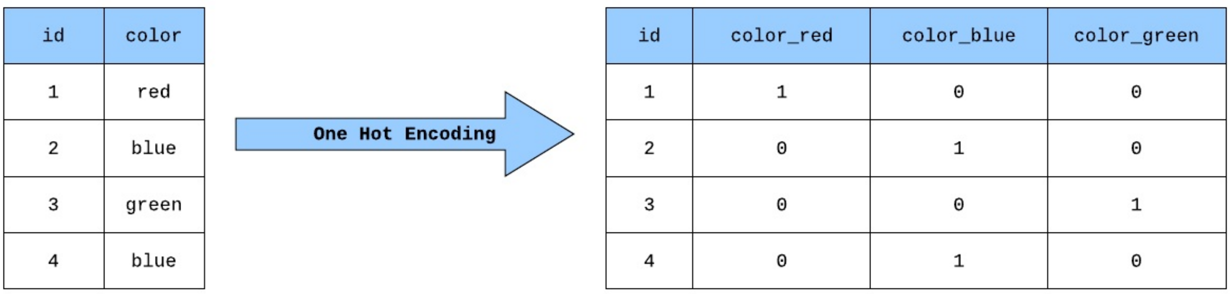
Out[17]:
```
▾ LabelEncoder
LabelEncoder()
```

In [18]:
```
y_train = le.transform(y_train)
y_test = le.transform(y_test)
```

In [19]:
```
#lets check
y_test
```

Out[19]:
```
array([0, 0, 0, 1, 1, 1, 0, 0, 0, 0])
```

One Hot Encoding



id	color
1	red
2	blue
3	green
4	blue

id	color_red	color_blue	color_green
1	1	0	0
2	0	1	0
3	0	0	1
4	0	1	0

In this way: above screenshot: In that way we divided column for One hot encoding for categorical data (Nominal data), but for further calculation - we have to remove one column example:color_red or color_blue or color_green will be remove otherwise we will face multicollineraty problem or dummy variable trap.

Why is it a problem?

In ML we need information about the features. We need to know if they are useful to predict the outcome or not.

Every feature is a Clue and we need to figure out its usefulness.

But if Clue 2 and Clue 3 are just saying the same thing as Clue 1, you can't really tell how much each clue is adding. They're all mixed up! From Clue 2 & 3 you can easily predict what Clue 1 will be.

In ML terms: The features are perfectly correlated, leading to unstable coefficient estimation.

In [20]:
```
import numpy as np
import pandas as pd
```

In [21]:
```
df = pd.read_csv(r"C:\Users\USER\Downloads\cars.csv")
df.head()
```

Out[21]:

	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

if want to do one hot encoding only in data analysis with pandas then process known as get_dummies

In [22]:
```
pd.get_dummies(df,columns=['fuel','owner'])
```

Out[22]:

	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 - To remove multicollenarity

In [23]:
```
pd.get_dummies(df,columns=['fuel','owner'],drop_first=True)
```

Out[23]:

	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

USING ML - Column Transform

In [24]:
```
df.head()
```
Out[24]:

	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 [33]:
```
df['fuel'].value_counts()
```
Out[33]:
```
Diesel    4402
Petrol    3631
CNG         57
LPG         38
Name: fuel, dtype: int64
```

In [34]:
```
df['owner'].value_counts()
```
Out[34]:
```
First Owner             5289
Second Owner            2105
Third Owner              555
Fourth & Above Owner     174
Test Drive Car             5
Name: owner, dtype: int64
```

Here we can use ordinal encoding for owner because we can give rank there but, for fuel its nominal encoding or One Hot encoding

In [40]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(df.iloc[:,0:4],df.iloc[:,-1],test_size=0.2,random_state=2)
```

In [41]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import OrdinalEncoder
```

In [42]:
```
transformer = ColumnTransformer(transformers=[
    ('tnf1',OrdinalEncoder(categories= [['First Owner','Second Owner','Third Owner','Fourth & Above Owner','Tes
    ('tnf2',OneHotEncoder(sparse=False,drop='first'),['fuel'])
],remainder='passthrough')
```

In [43]:
```
transformer.fit_transform(X_train).shape
```
```
C:\Users\USER\anaconda3\Lib\site-packages\sklearn\preprocessing\_encoders.py:972: FutureWarning: `sparse` was r
enamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leav
e `sparse` to its default value.
  warnings.warn(
```
Out[43]:
```
(6502, 6)
```

In [44]:
```
transformer.transform(X_test).shape
```
Out[44]:
```
(1626, 6)
```

In [ ]:
```
This is all process, we will see one more example in PART B.
```