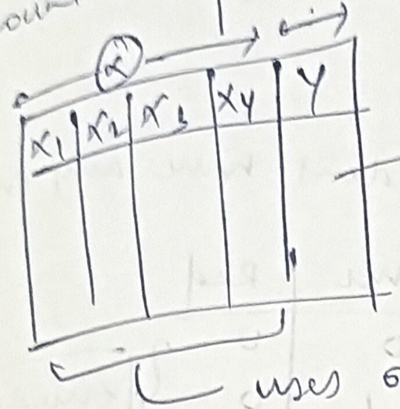
categorical data

Nominal	Ordinal
- no order	- ordered data.
- countries	- grades



category → Label encoder
- used for o/p of the dataframe

uses ordinal encoding

Education
HS - 0
UG - 1
PG - 2
HS - 0
UG - 1
PG - 2

HS - 0
UG - 1
PG - 2
⇒ ORDINAL ENCODING

* Before encoding split the data into training & testing

Review	education	Purchase
Poor	School	Yes
Avg	UG	No
Good	PG	Yes
		No

code

```
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder (categories=[['Poor','Avg','Good'],
        ['School','UG','PG']])
oe.fit (x_train)
x_train = oe.transform (x_train)
x_test = oe.transform (x_test)
```

(The numbers 0, 1, 2 are written above 'Poor', 'Avg', 'Good' and above 'School', 'UG', 'PG'; an arrow labelled "order" runs under the second list.)

for o/p column

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(y_train)
```

- Label encoding is used to encode target column only.

One Hot Encoding

- used for nominal data (which don't have any order)

	Yes	Blue	Red	
Yellow	1	0	0	encoded data
Blue	0	1	0	
Red	0	0	1	

Y - Yellow
B - Blue
R - Red

Dummy Variable trap ⇒ Multicollinearity

owner	fuel	selling_price

ohe using pandas

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

not preferred

ohe using scikit learn

- split the data

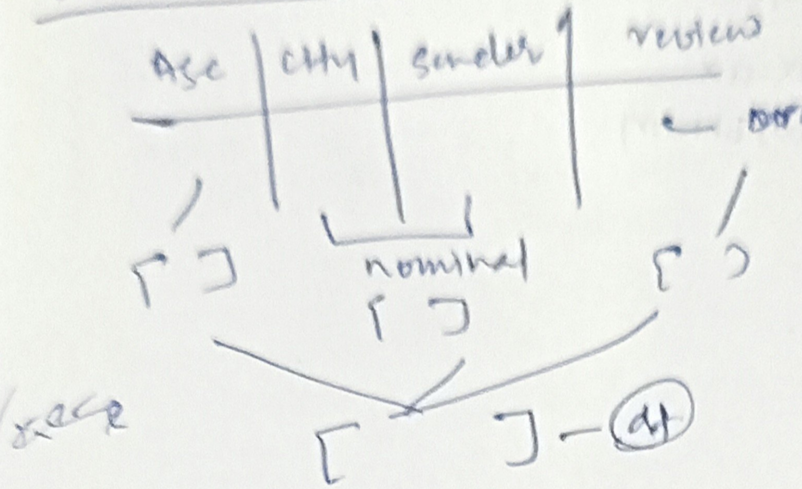
```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
ohe.fit_transform(X_train[['fuel','owner']]).toarray()
ohe.transform(X_test[['fuel','owner']]).toarray()
```

append them with the original df.

np.hstack(())
— concates two array side by side

sno | car + owner | fuel = sno | car | owner | fuel

columnTransformer

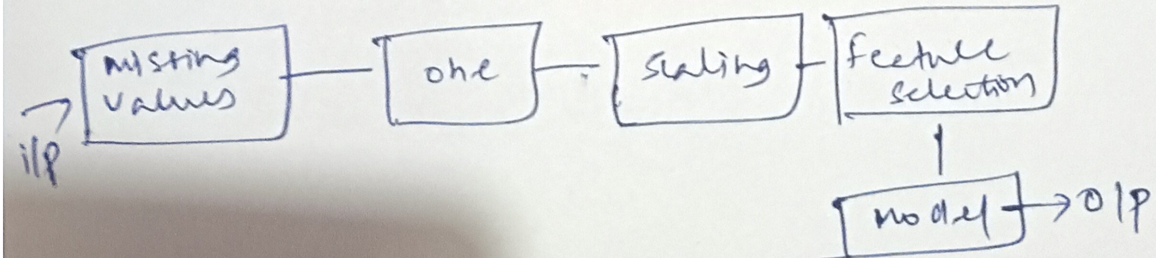


Simple imputer
- used to replace missing values

```
from sklearn.compose import columnTransformer
transformer = columnTransformer(transformers=[
        ('tnf1', SimpleImputer(), ['fever']),
        ('tnf2', OrdinalEncoder(categories=[['mild'],
              ['strong']]), ['cough']),
        ('tnf3', Ohe(sparse=False, drop='first'),
              ['Gender', 'city'])
    ], remainder='passthrough')
```

Pipelines

- pipelines chains together multiple steps so that the output of each step is used as i/p to the next step
- pipelines makes it easy to apply the same preprocessing to train & test.



Sklearn

- Function transformer
  - Logarithm
  - Reciprocal
  - Sq/Sqrt
  - custom
- Power transformer
  - Box-cox
  - Yeo-Johnson
- Quantile transformer