Categorical to Numerical

- It is very important convert categorical data to Numerical data
- Because in Machine learning models the input is in the form numbers
- ML models developed by maths , so passing text data is not works.
- We have different types of methods are there
  - map method
  - np.where
  - One hot encoder
  - Label Encoder

Import the packages

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Read the data

In [2]:
```
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
visa_df.head(2)
```

Out[2]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training
0	EZYV01	Asia	High School	N	N
1	EZYV02	Asia	Master's	Y	N

In [3]:
```
# categorical columns
visa_df.select_dtypes(include='object').columns
```

Out[3]:
```
Index(['case_id', 'continent', 'education_of_employee', 'has_job_experience',
       'requires_job_training', 'region_of_employment', 'unit_of_wage',
       'full_time_position', 'case_status'],
      dtype='object')
```

map

- first get the unique labels for each coumn
- create a dictionary by providing a values to each label
- For example case status has two lables , we are provding two values
  - Assign 0 for certified

- Assign 1 for Denied

In [5]:
```
visa_df['case_status'].unique()
```

Out[5]:
```
array(['Denied', 'Certified'], dtype=object)
```

In [6]:
```
d={'Denied':1,'Certified':0}
d
```

Out[6]:
```
{'Denied': 1, 'Certified': 0}
```

In [ ]:
```

```

In [7]:
```
visa_df['case_status_new']=visa_df['case_status'].map(d)
```

In [8]:
```
visa_df
```

Out[8]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_t
0	EZYV01	Asia	High School	N	
1	EZYV02	Asia	Master's	Y	
2	EZYV03	Asia	Bachelor's	N	
3	EZYV04	Asia	Bachelor's	N	
4	EZYV05	Africa	Master's	Y	
...	...	...	...	...	
25475	EZYV25476	Asia	Bachelor's	Y	
25476	EZYV25477	Asia	High School	Y	
25477	EZYV25478	Asia	Master's	Y	
25478	EZYV25479	Asia	Master's	Y	
25479	EZYV25480	Asia	Bachelor's	Y	

25480 rows × 13 columns

In [9]:
```
# read the data again
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
d={'Denied':1,'Certified':0}
visa_df['case_status']=visa_df['case_status'].map(d)
visa_df
```

Out[9]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_t
0	EZYV01	Asia	High School	N	
1	EZYV02	Asia	Master's	Y	
2	EZYV03	Asia	Bachelor's	N	
3	EZYV04	Asia	Bachelor's	N	
4	EZYV05	Africa	Master's	Y	
...	...	...	...	...	
25475	EZYV25476	Asia	Bachelor's	Y	
25476	EZYV25477	Asia	High School	Y	
25477	EZYV25478	Asia	Master's	Y	
25478	EZYV25479	Asia	Master's	Y	
25479	EZYV25480	Asia	Bachelor's	Y	

25480 rows × 12 columns

In [11]:
```
d2={'Y':1,'N':0}
visa_df['has_job_experience']=visa_df['has_job_experience'].map(d2)
visa_df
```

Out[11]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_t
0	EZYV01	Asia	High School	0	
1	EZYV02	Asia	Master's	1	
2	EZYV03	Asia	Bachelor's	0	
3	EZYV04	Asia	Bachelor's	0	
4	EZYV05	Africa	Master's	1	
...	...	...	...	...	
25475	EZYV25476	Asia	Bachelor's	1	
25476	EZYV25477	Asia	High School	1	
25477	EZYV25478	Asia	Master's	1	
25478	EZYV25479	Asia	Master's	1	
25479	EZYV25480	Asia	Bachelor's	1	

25480 rows × 12 columns

task

- read the data again

- apply for loop
- Iterate through columns
- Get the unique lables of each column
- Get the count of unique lables
- Create a dictioanry
- apply in to the map

In [21]:
```
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
visa_df['case_status'].unique()
```

Out[21]:
```
array(['Denied', 'Certified'], dtype=object)
```

In [20]:
```
cols=visa_df.select_dtypes(include='object').columns
for i in cols:
    labels=list(visa_df[i].unique())
    values=[i for i in range(len(labels))]
    d=dict(zip(labels,values))
    visa_df[i]=visa_df[i].map(d)
visa_df
```

Out[20]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_trai
0	0	0	0	0	
1	1	0	1	1	
2	2	0	2	0	
3	3	0	2	0	
4	4	1	1	1	
...	...	...	...	...	
25475	25475	0	2	1	
25476	25476	0	0	1	
25477	25477	0	1	1	
25478	25478	0	1	1	
25479	25479	0	2	1	

25480 rows × 12 columns

Label encoder

- Label encoder is a package under **sickit learn**
- Sickit-learn is heart of Machine learning
- In side sickit-learn we have a method called **Preprocessing**

- Under preprocessing we have **LableEncoder**
- scikit-learn we will write as **sklearn**
- any sklearn package we have 3 steps
    - Step-1: Read the package
    - Step-2: Save the package
    - Step-3: Apply Fit transform

In [22]:
```
# read the data
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
```

In [23]:
```
# Step-1: Read the package
from sklearn.preprocessing import LabelEncoder
# Step-2: Save the package
le=LabelEncoder()
# Step-3: apply fit transform
visa_df['case_status']=le.fit_transform(visa_df['case_status'])
```

In [24]:
```
visa_df
```

Out[24]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_t
0	EZYV01	Asia	High School	N	
1	EZYV02	Asia	Master's	Y	
2	EZYV03	Asia	Bachelor's	N	
3	EZYV04	Asia	Bachelor's	N	
4	EZYV05	Africa	Master's	Y	
...	...	...	...	...	
25475	EZYV25476	Asia	Bachelor's	Y	
25476	EZYV25477	Asia	High School	Y	
25477	EZYV25478	Asia	Master's	Y	
25478	EZYV25479	Asia	Master's	Y	
25479	EZYV25480	Asia	Bachelor's	Y	

25480 rows × 12 columns

In [26]:
```
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
cols=visa_df.select_dtypes(include='object').columns

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in cols:
    visa_df[i]=le.fit_transform(visa_df[i])
```

```
visa_df
```

Out[26]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_trai
0	0	1	2	0	
1	1	1	3	1	
2	2	1	0	0	
3	3	1	0	0	
4	4	0	3	1	
...	...	...	...	...	
25475	17204	1	0	1	
25476	17205	1	2	1	
25477	17206	1	3	1	
25478	17207	1	3	1	
25479	17209	1	0	1	

25480 rows × 12 columns

inverse transform

- in order to see inverse of lables we need to perform individually again

In [30]:
```
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

visa_df['case_status']=le.fit_transform(visa_df['case_status'])
print(visa_df['case_status'].values) # why we are applying values
print(le.inverse_transform(visa_df['case_status']))
```

```
[1 0 1 ... 0 0 0]
['Denied' 'Certified' 'Denied' ... 'Certified' 'Certified' 'Certified']
```

LableEncoder will use most of the time

np.where

- np.where applicable only for binary
- np.where will take 3 arguments
  - conditon
  - True value
  - False value

senario

- For example case status value equlal to Certified replace with zero
- Other wise relpace with 1

In [31]:
```
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
con=visa_df['case_status']=='Certified'
visa_df['case_status']=np.where(con,0,1)
visa_df
```

Out[31]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_t
0	EZYV01	Asia	High School	N	
1	EZYV02	Asia	Master's	Y	
2	EZYV03	Asia	Bachelor's	N	
3	EZYV04	Asia	Bachelor's	N	
4	EZYV05	Africa	Master's	Y	
...	...	...	...	...	
25475	EZYV25476	Asia	Bachelor's	Y	
25476	EZYV25477	Asia	High School	Y	
25477	EZYV25478	Asia	Master's	Y	
25478	EZYV25479	Asia	Master's	Y	
25479	EZYV25480	Asia	Bachelor's	Y	

25480 rows × 12 columns

One hot encoder

- One hot means if one is ON another one is OFF
- ON represents with 1, and OFF represents with 0
- For example case status has two unique lables
  - Denied
  - Certified
- If we apply one hot encoder on case status column it will create two extra columns
  - case_status_Denied
  - case_status_Certified

case_status	case_status_denied	case_status_certified
Denied	1	0
Certified	0	1

Advantages

- In Machine learning we have one important property is **Independent of variables**
- Machine learning models expects variables are **independent each other**
- One hot coder creating new columns , the new columns are **independent each other**
- Indepente variables means both variables have **90 degrees phase shift**
- 90 degrees phase shift also called as **perpendicular each other**
- Perpendicular means **orthogonal each other**

Disadvantage

- One hot encoder creates new columns equal to number of unique lables
- If number of unique lables increase, new columns also increase
- The more columns leads data is more
- Then data is more the complexity will increase
- Complexity increases more computation power required
- Also time also incease
- The entire process names as **curse of dimenionality**

pd.get_dummies

- Read the data again
- choose one column
- apply pd.get_dummies

In [3]:
```
import pandas as pd
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
pd.get_dummies(visa_df['case_status'],dtype='int')
```

Out[3]:

	Certified	Denied
0	0	1
1	1	0
2	0	1
3	0	1
4	1	0
...	...	...
25475	1	0
25476	1	0
25477	1	0
25478	1	0
25479	1	0

25480 rows × 2 columns

In [4]:
```
import pandas as pd
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
pd.get_dummies(visa_df,dtype='int')
```

Out[4]:

	no_of_employees	yr_of_estab	prevailing_wage	case_id_EZYV01	case_id_EZYV02
0	14513	2007	592.2029	1	0
1	2412	2002	83425.6500	0	1
2	44444	2008	122996.8600	0	0
3	98	1897	83434.0300	0	0
4	1082	2005	149907.3900	0	0
...	...	...	...	...	...
25475	2601	2008	77092.5700	0	0
25476	3274	2006	279174.7900	0	0
25477	1121	1910	146298.8500	0	0
25478	1918	1887	86154.7700	0	0
25479	3195	1960	70876.9100	0	0

25480 rows × 25510 columns

Note

Always drop the id column

In [5]:
```
import pandas as pd
visa_df=pd.read_csv(r"C:\Users\omkar\OneDrive\Documents\Data science\Naresh IT\N
visa_df.drop('case_id',axis=1,inplace=True)
pd.get_dummies(visa_df,dtype='int')
```

Out[5]:

	no_of_employees	yr_of_estab	prevailing_wage	continent_Africa	continent_Asia
0	14513	2007	592.2029	0	1
1	2412	2002	83425.6500	0	1
2	44444	2008	122996.8600	0	1
3	98	1897	83434.0300	0	1
4	1082	2005	149907.3900	1	0
...	...	...	...	...	...
25475	2601	2008	77092.5700	0	1
25476	3274	2006	279174.7900	0	1
25477	1121	1910	146298.8500	0	1
25478	1918	1887	86154.7700	0	1
25479	3195	1960	70876.9100	0	1

25480 rows × 30 columns

- Generally we will prefer LabelEncoder
- One hot encoder when categorical data has less unique lables

In [ ]: