Nominal/OHE Encoding

Nominal encoding is a technique used to transform categorical variables that have no intrinsic ordering into numerical values that can be used in machine learning models. One common method for nominal encoding is one_hot encoding,which creates a binary vector for each category in the variable .

In [1]:
```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
```

In [2]:
```
df=pd.DataFrame({'color':['red','blue','green','green','red','blue']})
```

In [3]:
```
df
```

Out[3]:

	color
0	red
1	blue
2	green
3	green
4	red
5	blue

In [4]:
```
#create an instance of one hot encoder
encoder=OneHotEncoder()
```

In [5]:
```
encoder.fit_transform(df[['color']])
```

Out[5]:
```
<6x3 sparse matrix of type '<class 'numpy.float64'>'
	with 6 stored elements in Compressed Sparse Row format>
```

In [6]:
```
encoder.fit_transform(df[['color']]).toarray()
```

Out[6]:
```
array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

In [7]:
```
#fit the   encoder to the dataframe and transform the categorical variable
encoded=encoder.fit_transform(df[['color']])
```

In [8]:
```
import pandas as pd
encoded_df=pd.DataFrame(encoded.toarray(),columns=encoder.get_feature_names_out())
```

In [10]:
```
encoded_df
```

Out[10]:

	color_blue	color_green	color_red
0	0.0	0.0	1.0
1	1.0	0.0	0.0
2	0.0	1.0	0.0
3	0.0	1.0	0.0
4	0.0	0.0	1.0
5	1.0	0.0	0.0

In [9]:
```
pd.concat([df,encoded_df],axis=1)
```

Out[9]:

	color	color_blue	color_green	color_red
0	red	0.0	0.0	1.0
1	blue	1.0	0.0	0.0
2	green	0.0	1.0	0.0

3	green	0.0	1.0	0.0
4	red	0.0	0.0	1.0
5	blue	1.0	0.0	0.0

In [10]:

```
import seaborn as sns
df=sns.load_dataset('tips')
```

In [40]:

```
df
```

Out[40]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

In [ ]:

In [19]:

```
from sklearn.preprocessing import OneHotEncoder
```

In [20]:
```
encoder=OneHotEncoder()
```

In [21]:
```
df.head()
```

Out[21]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [34]:
```
encoded=encoder.fit_transform(df[['day']])
```

In [35]:
```
import pandas as pd
```

In [38]:
```
encoded_df1=pd.DataFrame(encoded.toarray(),columns=encoder.get_feature_names_out())
```

In [39]:
```
encoded_df1
```

Out[39]:

	day_Fri	day_Sat	day_Sun	day_Thur
0	0.0	0.0	1.0	0.0
1	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0
3	0.0	0.0	1.0	0.0
4	0.0	0.0	1.0	0.0

...	...	...	...	...
239	0.0	1.0	0.0	0.0
240	0.0	1.0	0.0	0.0
241	0.0	1.0	0.0	0.0
242	0.0	1.0	0.0	0.0
243	0.0	0.0	0.0	1.0

244 rows × 4 columns

In [41]:

```
pd.concat([df,encoded_df1],axis=1)
```

Out[41]:

	total_bill	tip	sex	smoker	day	time	size	day_Fri	day_Sat	day_Sun	day_Thur
0	16.99	1.01	Female	No	Sun	Dinner	2	0.0	0.0	1.0	0.0
1	10.34	1.66	Male	No	Sun	Dinner	3	0.0	0.0	1.0	0.0
2	21.01	3.50	Male	No	Sun	Dinner	3	0.0	0.0	1.0	0.0
3	23.68	3.31	Male	No	Sun	Dinner	2	0.0	0.0	1.0	0.0
4	24.59	3.61	Female	No	Sun	Dinner	4	0.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	0.0	1.0	0.0	0.0
240	27.18	2.00	Female	Yes	Sat	Dinner	2	0.0	1.0	0.0	0.0
241	22.67	2.00	Male	Yes	Sat	Dinner	2	0.0	1.0	0.0	0.0
242	17.82	1.75	Male	No	Sat	Dinner	2	0.0	1.0	0.0	0.0
243	18.78	3.00	Female	No	Thur	Dinner	2	0.0	0.0	0.0	1.0

244 rows × 11 columns

Label and Ordinal Encoder

Label for Label wise Encoding. Ordinal for Rank wise Encoding .

In [11]:
```
import pandas as pd
```

In [21]:
```
#Label encoding
```

In [12]:
```
from sklearn.preprocessing import LabelEncoder
```

In [13]:
```
df=pd.DataFrame({'color':['red','blue','green','green','red','blue']})
```

In [14]:
```
df.head()
```

Out[14]:

	color
0	red
1	blue
2	green
3	green
4	red

In [15]:
```
#create  an  instance of Labelencoder
labelencoder=LabelEncoder()
```

In [20]:
```
labelencoder.fit_transform(df['color'])
```

Out[20]:
```
array([2, 0, 1, 1, 2, 0])
```

In [22]:
```
## Ordinal Encding for Rank wise
```

In [23]:
```
from sklearn.preprocessing import OrdinalEncoder
```

In [24]:
```
#crate a sample dataframe with an ordinal variable
```

In [31]:
```
import pandas as pd
df=pd.DataFrame({'size':['small','medium','large','medium','small','large']})
```

In [32]:
```
df.head()
```

Out[32]:

	size
0	small
1	medium
2	large
3	medium
4	small

In [33]:
```
##create an instnce of the odinal encoder class and fit_transform
```

In [34]:
```
encoder=OrdinalEncoder(categories=[['small','medium','large']])
```

In [35]:
```
encoder.fit_transform(df[['size']])
```

Out[35]:
```
array([[0.],
       [1.],
       [2.],
       [1.],
       [0.],
       [2.]])
```

Target Guided Ordinal Encoding

It is a technique used to encode categorical varables based on their relationship with their target variables. IN Target Guided Ordinal

Encoding we replace each categorical variable based on their mean or median of the target variable for that category.

In [50]:
```
import pandas as pd
#create a simple dataframe with a categorical variable and a target variable
df=pd.DataFrame({'city':['New York','London','paris','Tokyo','New York','paris'],
                'price':[200,300,400,500,600,700]})
```

In [51]:
```
df
```

Out[51]:

	city	price
0	New York	200
1	London	300
2	paris	400
3	Tokyo	500
4	New York	600
5	paris	700

In [43]:
```
#Here price is the target variable and city is the categorical variable
```

In [52]:
```
##calculate the mean price for ecach city
mean_price=df.groupby('city')['price'].mean().to_dict()#for converting into dictionary
mean_price
```

Out[52]:
```
{'London': 300.0, 'New York': 400.0, 'Tokyo': 500.0, 'paris': 550.0}
```

In [45]:
```
##If there is outliers then we use the median value
```

In [48]:
```
##replace each city with their mean price
```

In [53]:

```
df['city_encoded']=df['city'].map(mean_price)
```

In [54]:

```
df
```

Out[54]:

	city	price	city_encoded
0	New York	200	400.0
1	London	300	300.0
2	paris	400	550.0
3	Tokyo	500	500.0
4	New York	600	400.0
5	paris	700	550.0

In [58]:

```
import seaborn as sns
df=sns.load_dataset('tips')
```

In [59]:

```
df.head()
```

Out[59]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [62]:

```
mean_price=df.groupby('day')['total_bill'].mean().to_dict()
mean_price
```

Out[62]:

```
{'Thur': 17.682741935483868,
 'Fri': 17.151578947368421
```

```
 'Sat': 20.44137931034483,
 'Sun': 21.41}
```

In [63]:
```
df['encoded_day']=df['day'].map(mean_price)
```

In [64]:
```
df
```

Out[64]:

	total_bill	tip	sex	smoker	day	time	size	encoded_day
0	16.99	1.01	Female	No	Sun	Dinner	2	21.410000
1	10.34	1.66	Male	No	Sun	Dinner	3	21.410000
2	21.01	3.50	Male	No	Sun	Dinner	3	21.410000
3	23.68	3.31	Male	No	Sun	Dinner	2	21.410000
4	24.59	3.61	Female	No	Sun	Dinner	4	21.410000
...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	20.441379
240	27.18	2.00	Female	Yes	Sat	Dinner	2	20.441379
241	22.67	2.00	Male	Yes	Sat	Dinner	2	20.441379
242	17.82	1.75	Male	No	Sat	Dinner	2	20.441379
243	18.78	3.00	Female	No	Thur	Dinner	2	17.682742

244 rows × 8 columns

In [67]:
```
mean_price_time=df.groupby('time')['tip'].mean().to_dict()
```

In [68]:
```
mean_price_time
```

Out[68]:
```
{'Lunch': 2.7280882352941176, 'Dinner': 3.102670454545455}
```

In [70]:
```
#replace ech time with their mean tip price
```

```
df['encoded_time']=df['time'].map(mean_price_time)
```

In [71]:
```
df
```

Out[71]:

	total_bill	tip	sex	smoker	day	time	size	encoded_day	encoded_time
0	16.99	1.01	Female	No	Sun	Dinner	2	21.410000	3.10267
1	10.34	1.66	Male	No	Sun	Dinner	3	21.410000	3.10267
2	21.01	3.50	Male	No	Sun	Dinner	3	21.410000	3.10267
3	23.68	3.31	Male	No	Sun	Dinner	2	21.410000	3.10267
4	24.59	3.61	Female	No	Sun	Dinner	4	21.410000	3.10267
...	...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	20.441379	3.10267
240	27.18	2.00	Female	Yes	Sat	Dinner	2	20.441379	3.10267
241	22.67	2.00	Male	Yes	Sat	Dinner	2	20.441379	3.10267
242	17.82	1.75	Male	No	Sat	Dinner	2	20.441379	3.10267
243	18.78	3.00	Female	No	Thur	Dinner	2	17.682742	3.10267

244 rows × 9 columns

In [74]:
```
mean_price_size=df.groupby('time')['size'].mean().to_dict()
```

In [75]:
```
mean_price_size
```

Out[75]: {'Lunch': 2.411764705882353, 'Dinner': 2.6306818181818183}

In [77]:
```
df['encoded_time_size']=df['time'].map(mean_price_size)
```

In [78]:
```
df
```

Out[78]:

	total_bill	tip	sex	smoker	day	time	size	encoded_day	encoded_time	encoded_time_size
0	16.99	1.01	Female	No	Sun	Dinner	2	21.410000	3.10267	2.630682
1	10.34	1.66	Male	No	Sun	Dinner	3	21.410000	3.10267	2.630682
2	21.01	3.50	Male	No	Sun	Dinner	3	21.410000	3.10267	2.630682
3	23.68	3.31	Male	No	Sun	Dinner	2	21.410000	3.10267	2.630682
4	24.59	3.61	Female	No	Sun	Dinner	4	21.410000	3.10267	2.630682
...	...	...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	20.441379	3.10267	2.630682
240	27.18	2.00	Female	Yes	Sat	Dinner	2	20.441379	3.10267	2.630682
241	22.67	2.00	Male	Yes	Sat	Dinner	2	20.441379	3.10267	2.630682
242	17.82	1.75	Male	No	Sat	Dinner	2	20.441379	3.10267	2.630682
243	18.78	3.00	Female	No	Thur	Dinner	2	17.682742	3.10267	2.630682

244 rows × 10 columns

In [ ]: