Data Encoding

```
1.Nominal/OHE Encoding


2.Label and Ordinal Encoding


3.Target Guided Ordinal Encoding
```

Nominal/OHE Encoding

One hot encoding, also known as nominal encoding, is a technique used to represent categorical data as numerical data, which is more suitable for machine learning algorithms. In this technique, each category is represented as a binary vector where each bit corresponds to a unique category. For example, if we have a categorical variable "color" with three possible value(red,green,blue), we can represent it one hot encoding as follows:

1.Red:[1,0,0] 2.Green:[0,1,0] 3.Blue:[0,0,1]

In [1]:

```python
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
```

In [2]:

```python
## Create a simple DataFrame
df = pd.DataFrame({'color':['red', 'blue','green', 'green', 'red','blue']

})
```

In [3]:

```python
df.head()
```

Out[3]:

	color
0	red
1	blue
2	green
3	green
4	red

In [10]:

```python
## Create an instance of OneHotEncoder
encoder = OneHotEncoder()
```

In [11]:

```python
## Perform fit and Transform
encoded = encoder.fit_transform(df[['color']]).toarray()
```

In [12]:

```
import pandas as pd
encoded_df=pd.DataFrame(encoded, columns=encoder.get_feature_names_out())
```

In [13]:

```
encoded_df
```

Out[13]:

	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 [14]:

```
## For new data
encoder.transform([['blue']]).toarray()
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:409: UserWarning: X doe
s not have valid feature names, but OneHotEncoder was fitted with feature names
  warnings.warn(
```

Out[14]:

```
array([[1., 0., 0.]])
```

In [16]:

```
encoder.transform([['green']]).toarray()
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:409: UserWarning: X doe
s not have valid feature names, but OneHotEncoder was fitted with feature names
  warnings.warn(
```

Out[16]:

```
array([[0., 1., 0.]])
```

In [17]:

```
encoder.transform([['red']]).toarray()
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:409: UserWarning: X doe
s not have valid feature names, but OneHotEncoder was fitted with feature names
  warnings.warn(
```

Out[17]:

```
array([[0., 0., 1.]])
```

In [18]:

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

Out[18]:

	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

Label Encoding

Label encoding and ordinal encoding are two techniques used to encode categorical data as numerical data.

Label encoding involves assigning a unique numerical label to each category in thevariable. The labels are usually assigned in alphabetical order or based on the frequency of the categories. For example, if we have a categorical variable "color" with three possible values ('red', 'green', 'blue'), we can represent it using label encoding as follows:

1.Red: 1 2.Green: 2 3.Blue: 3

In [19]:
```
df.head()
```

Out[19]:

	color
0	red
1	blue
2	green
3	green
4	red

In [20]:
```
from sklearn.preprocessing import LabelEncoder
lbl_encoder=LabelEncoder()
```

In [21]:
```
lbl_encoder.fit_transform(df[['color']])
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/preprocessing/_label.py:116: Da
taConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[21]:
```
array([2, 0, 1, 1, 2, 0])
```

In [22]:
```
lbl_encoder.transform([['red']])
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/preprocessing/_label.py:134: Da
taConversionWarning: A column-vector y was passed when a 1d array was expected.
Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
```

Out[22]:
```
array([2])
```

In [23]:
```
lbl_encoder.transform([['blue']])
```

Out[23]:
```
array([0])
```

In [24]:
```
lbl_encoder.transform([['green']])
```

Out[24]:
```
array([1])
```

Ordinal Encoding

It is used to encode categorical data that have an intrinsic order or ranking. In this technique, each category is assigned a numerical value based on its position in the order. For example, if we have a ctegorical variable "education level" with four possible values (high school, college, graduate, post-graduate), we can represnt it using ordinal encoding as follows:

High School: 1 College: 2 Graduate: 3 Post-graduate: 4

In [25]:
```
## Ordinal encoding
from sklearn.preprocessing import OrdinalEncoder
```

In [26]:
```
## Create a sample dataframe with an ordinal variable
df = pd.DataFrame({
    'size':['small','medium','large','medium','small','large']
})
```

In [27]:
```
df
```

Out[27]:

	size
0	small
1	medium
2	large
3	medium
4	small
5	large

In [28]:
```
## Create an insatnce for OrdinalEncoder and then fit_transform
encoder=OrdinalEncoder(categories=[['small','medium','large']])
```

In [29]:
```
encoder.fit_transform(df[['size']])
```

Out[29]:
```
array([[0.],
       [1.],
       [2.],
       [1.],
       [0.],
       [2.]])
```

In [30]:
```
encoder.transform([['small']])
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:409: UserWarning: X doe
s not have valid feature names, but OrdinalEncoder was fitted with feature name
s
  warnings.warn(
```

Out[30]:
```
array([[0.]])
```

In [31]:
```
encoder.transform([['medium']])
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:409: UserWarning: X doe
s not have valid feature names, but OrdinalEncoder was fitted with feature name
s
  warnings.warn(
```

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

Target Guided Ordinal Encoding

It is a technique used to encode categorical variables based on their relationship with the target variable. This encoding technique is useful when we have a categorical variable with a large number of unique categories, and we want to use this variable as a feature in our machine learning model.

In Target Guided Ordinal Encoding, we replace each category in the the categorical variable with a numerical value based on the mean or median of the target variable for that category. This creates a monotonic relationship between the categorical variable and the target variable, which can improve the predictive power of our model.

In [32]:
```python
import pandas as pd

# Create a sample dataframe with a categorical variable and a target variable

df = pd.DataFrame({
    'city': ['New York', 'London', 'Paris', 'Tokyo', 'New York', 'Paris'],
    'price': [200,150,300,250,180,320]
})
```

In [33]:
```python
df
```

Out[33]:

	city	price
0	New York	200
1	London	150
2	Paris	300
3	Tokyo	250
4	New York	180
5	Paris	320

In [34]:
```python
mean_price= df.groupby('city')['price'].mean().to_dict()
```

In [35]:
```python
mean_price
```

Out[35]:
```
{'London': 150.0, 'New York': 190.0, 'Paris': 310.0, 'Tokyo': 250.0}
```

In [36]:
```python
df['city_encoded']=df['city'].map(mean_price)
```

In [37]:
```python
df
```

Out[37]:

	city	price	city_encoded
0	New York	200	190.0
1	London	150	150.0
2	Paris	300	310.0
3	Tokyo	250	250.0
4	New York	180	190.0
5	Paris	320	310.0

In [ ]:

In [ ]:

In [ ]:

In [ ]: