```
import numpy as np
import pandas as pd

df = pd.read_csv('/content/customer.csv')

df.head()
```

	age	gender	review	education	purchased
0	30	Female	Average	School	No
1	68	Female	Poor	UG	No
2	70	Female	Good	PG	No
3	72	Female	Good	PG	No
4	16	Female	Average	UG	No

1. Ordinal Encoding

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:2],
df.iloc[:,-1], test_size=0.2)

X_train
```

	review	education
25	Good	School
32	Average	UG
33	Good	PG
23	Good	School
7	Poor	School
8	Average	UG
20	Average	School
4	Average	UG
29	Average	UG
15	Poor	UG
46	Poor	PG
5	Average	School
12	Poor	School
0	Average	School
34	Average	School
24	Average	PG
37	Average	PG
10	Good	UG
44	Average	UG
19	Poor	PG
47	Good	PG
27	Poor	PG
45	Poor	PG
49	Good	UG
2	Good	PG
14	Poor	PG

```
28     Poor    School
13  Average    School
38     Good    School
18     Good    School
43     Poor        PG
39     Poor        PG
1      Poor        UG
11     Good        UG
21  Average        PG
6      Good    School
40     Good    School
30  Average        UG
36     Good        UG
26     Poor        PG
```

```
# specify order
oe = OrdinalEncoder(categories=[['Poor','Average','Good'],
['School','UG','PG']])

X_train = oe.fit_transform(X_train)
X_test = oe.transform(X_test)

X_train
```

```
array([[2., 0.],
       [1., 1.],
       [2., 2.],
       [2., 0.],
       [0., 0.],
       [1., 1.],
       [1., 0.],
       [1., 1.],
       [1., 1.],
       [0., 1.],
       [0., 2.],
       [1., 0.],
       [0., 0.],
       [1., 0.],
       [1., 0.],
       [1., 2.],
       [1., 2.],
       [2., 1.],
       [1., 1.],
       [0., 2.],
       [2., 2.],
       [0., 2.],
       [0., 2.],
       [2., 1.],
       [2., 2.],
       [0., 2.],
```

```
       [0., 0.],
       [1., 0.],
       [2., 0.],
       [2., 0.],
       [0., 2.],
       [0., 2.],
       [0., 1.],
       [2., 1.],
       [1., 2.],
       [2., 0.],
       [2., 0.],
       [1., 1.],
       [2., 1.],
       [0., 2.]])
```

```
oe.categories_
```

```
[array(['Poor', 'Average', 'Good'], dtype=object),
 array(['School', 'UG', 'PG'], dtype=object)]
```

```
oe.feature_names_in_
```

```
array(['review', 'education'], dtype=object)
```

```
oe.n_features_in_
```

```
2
```

```
oe.inverse_transform(np.array([0,2]).reshape(1,2))
```

```
array([['Poor', 'PG']], dtype=object)
```

```
oe.get_feature_names_out()
```

```
array(['review', 'education'], dtype=object)
```

```
# handle unknown
oe.transform(np.array(['Poor','college']).reshape(1,2))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465:
UserWarning: X does not have valid feature names, but OrdinalEncoder
was fitted with feature names
  warnings.warn(
```

```
---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-118-12646ee05ea3> in <cell line: 2>()
      1 # handle unknown
----> 2 oe.transform(np.array(['Poor','college']).reshape(1,2))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_set_output.py
in wrapped(self, X, *args, **kwargs)
    155     @wraps(f)
    156     def wrapped(self, X, *args, **kwargs):
--> 157         data_to_wrap = f(self, X, *args, **kwargs)
    158         if isinstance(data_to_wrap, tuple):
    159             # only wrap the first output for cross
decomposition

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoder
s.py in transform(self, X)
   1584             Transformed input.
   1585         """
-> 1586         X_int, X_mask = self._transform(
   1587             X,
   1588             handle_unknown=self.handle_unknown,

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoder
s.py in _transform(self, X, handle_unknown, force_all_finite,
warn_on_unknown, ignore_category_indices)
    198                             " during transform".format(diff, i)
    199                     )
--> 200                     raise ValueError(msg)
    201                 else:
    202                     if warn_on_unknown:

ValueError: Found unknown categories ['college'] in column 1 during
transform
```

```
# set unknown value
oe = OrdinalEncoder(categories=[['Poor','Average','Good'],
['School','UG','PG']],
                    handle_unknown='use_encoded_value',
                    unknown_value=-1)
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:2],
df.iloc[:,-1], test_size=0.2)
X_train = oe.fit_transform(X_train)
oe.transform(np.array(['Poor','college']).reshape(1,2))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465:
UserWarning: X does not have valid feature names, but OrdinalEncoder
was fitted with feature names
  warnings.warn(

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

```
# handling infrequent categories
X = np.array([['dog'] * 5 + ['cat'] * 20 + ['rabbit'] * 10 +['snake']
* 3 + ['horse'] * 2], dtype=object).T
X
```

```
array([['dog'],
       ['dog'],
       ['dog'],
       ['dog'],
       ['dog'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['cat'],
       ['rabbit'],
       ['rabbit'],
       ['rabbit'],
       ['rabbit'],
       ['rabbit'],
       ['rabbit'],
       ['rabbit'],
       ['rabbit'],
       ['rabbit'],
       ['rabbit'],
       ['snake'],
       ['snake'],
       ['snake'],
       ['horse'],
       ['horse']], dtype=object)
```

```
enc = OrdinalEncoder(max_categories=3).fit(X)
```

```
enc.infrequent_categories_
```

```
[array(['dog', 'snake'], dtype=object)]
```

```
enc.transform(np.array([['cat','rabbit','snake','dog']]).reshape(4,1))
```

```
array([[0.],
       [1.],
       [2.],
       [2.]])
```

```
enc = OrdinalEncoder(min_frequency=4).fit(X)
enc.infrequent_categories_
```

```
[array(['horse', 'snake'], dtype=object)]
```

```
enc.transform(np.array([['cat','rabbit','snake','dog','horse']]).resha
pe(5,1))
```

```
array([[0.],
       [2.],
       [3.],
       [1.],
       [3.]])
```

```
# update sklearn if the above command doesn't work
!pip install --upgrade scikit-learn==1.3.2
```

```
# handling missing data

# Example categorical data with missing values
data = [['Cat'], [np.nan], ['Dog'], ['Fish'], [np.nan]]

# Setting encoded_missing_value to -1, indicating we want missing
values to be encoded as -1
encoder = OrdinalEncoder(encoded_missing_value=-1)

encoded_data = encoder.fit_transform(data)

print(encoded_data)
```

```
[[ 0.]
 [-1.]
 [ 1.]
 [ 2.]
 [-1.]]
```

LabelEncoder

```
df.head()
```

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

```
X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:2],
df.iloc[:,-1], test_size=0.2)

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.transform(y_test)

le.classes_
```

```
array(['No', 'Yes'], dtype=object)
```

```
le.inverse_transform(np.array([1,1,0]))
```

```
array(['Yes', 'Yes', 'No'], dtype=object)
```

2. OneHotEncoder

```
cars = pd.read_csv('cars.csv')
cars.head()
```

```
     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
```

```
cars.isnull().sum()
```

```
brand            0
km_driven        0
fuel             0
owner            0
selling_price    0
dtype: int64
```

```
X = cars.iloc[:,0:2]
y = cars.iloc[:,-1]

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.2,random_state=42)

X_train['fuel'].nunique()
```

```
4
```

```
from sklearn.preprocessing import OneHotEncoder

ohe = OneHotEncoder(sparse_output=False, dtype=np.int32)
```

```
ohe.fit_transform(X_train)
```

```
array([[0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 1, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 1],
       [0, 0, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 0, 0, 1]], dtype=int32)
```

```
X_train = ohe.fit_transform(X_train).toarray()
X_train
```

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

```
X_train.shape
```

```
(6502, 36)
```

```
ohe.categories_
```

```
[array(['Ambassador', 'Ashok', 'Audi', 'BMW', 'Chevrolet', 'Daewoo',
       'Datsun', 'Fiat', 'Force', 'Ford', 'Honda', 'Hyundai',
'Isuzu',
       'Jaguar', 'Jeep', 'Kia', 'Land', 'Lexus', 'MG', 'Mahindra',
       'Maruti', 'Mercedes-Benz', 'Mitsubishi', 'Nissan', 'Opel',
       'Peugeot', 'Renault', 'Skoda', 'Tata', 'Toyota', 'Volkswagen',
       'Volvo'], dtype=object),
 array(['CNG', 'Diesel', 'LPG', 'Petrol'], dtype=object)]
```

```
ohe.feature_names_in_
```

```
array(['brand', 'fuel'], dtype=object)
```

```
ohe.n_features_in_
```

```
2
```

```
ohe.get_feature_names_out()
```

```
array(['brand_Ambassador', 'brand_Ashok', 'brand_Audi', 'brand_BMW',
       'brand_Chevrolet', 'brand_Daewoo', 'brand_Datsun',
'brand_Fiat',
       'brand_Force', 'brand_Ford', 'brand_Honda', 'brand_Hyundai',
       'brand_Isuzu', 'brand_Jaguar', 'brand_Jeep', 'brand_Kia',
       'brand_Land', 'brand_Lexus', 'brand_MG', 'brand_Mahindra',
```

```
       'brand_Maruti', 'brand_Mercedes-Benz', 'brand_Mitsubishi',
       'brand_Nissan', 'brand_Opel', 'brand_Peugeot', 'brand_Renault',
       'brand_Skoda', 'brand_Tata', 'brand_Toyota',
'brand_Volkswagen',
       'brand_Volvo', 'fuel_CNG', 'fuel_Diesel', 'fuel_LPG',
       'fuel_Petrol'], dtype=object)
```

```
pd.DataFrame(ohe.fit_transform(X_train),columns=ohe.get_feature_names_
out())
```

```
      brand_Ashok  brand_Audi  brand_BMW  brand_Chevrolet
brand_Daewoo  \
0             0.0         0.0        0.0              0.0
0.0
1             0.0         0.0        0.0              0.0
0.0
2             0.0         0.0        0.0              0.0
0.0
3             0.0         0.0        0.0              0.0
0.0
4             0.0         0.0        0.0              0.0
0.0
...           ...         ...        ...              ...            ..
.
6497          0.0         0.0        0.0              0.0
0.0
6498          0.0         0.0        0.0              0.0
0.0
6499          0.0         0.0        0.0              0.0
0.0
6500          0.0         0.0        0.0              0.0
0.0
6501          0.0         0.0        0.0              0.0
0.0

      brand_Datsun  brand_Fiat  brand_Force  brand_Ford
brand_Honda  ...  \
0              0.0         0.0          0.0         0.0
0.0  ...
1              0.0         0.0          0.0         0.0
1.0  ...
2              0.0         0.0          0.0         0.0
0.0  ...
3              0.0         0.0          0.0         0.0
0.0  ...
4              0.0         0.0          0.0         0.0
0.0  ...
...            ...         ...          ...         ...          ...
...
6497           0.0         0.0          0.0         0.0
```

```
0.0  ...
6498           0.0         0.0           0.0         0.0
0.0  ...
6499           0.0         0.0           0.0         0.0
0.0  ...
6500           0.0         0.0           0.0         0.0
0.0  ...
6501           0.0         0.0           0.0         0.0
0.0  ...

      brand_Peugeot  brand_Renault  brand_Skoda  brand_Tata
brand_Toyota  \
0               0.0            0.0          0.0         1.0
0.0
1               0.0            0.0          0.0         0.0
0.0
2               0.0            0.0          0.0         0.0
0.0
3               0.0            0.0          0.0         0.0
0.0
4               0.0            0.0          0.0         0.0
0.0
...             ...            ...          ...         ...
...
6497            0.0            0.0          0.0         0.0
0.0
6498            0.0            0.0          0.0         0.0
0.0
6499            0.0            0.0          0.0         0.0
0.0
6500            0.0            0.0          0.0         0.0
0.0
6501            0.0            0.0          0.0         0.0
0.0

      brand_Volkswagen  brand_Volvo  fuel_Diesel  fuel_LPG
fuel_Petrol
0                  0.0          0.0          0.0       0.0
1.0
1                  0.0          0.0          0.0       0.0
1.0
2                  0.0          0.0          1.0       0.0
0.0
3                  0.0          0.0          1.0       0.0
0.0
4                  0.0          0.0          0.0       0.0
1.0
...                ...          ...          ...       ...          ..
.
```

```
6497                0.0          0.0          1.0        0.0
0.0
6498                0.0          0.0          1.0        0.0
0.0
6499                0.0          0.0          0.0        0.0
1.0
6500                0.0          0.0          1.0        0.0
0.0
6501                0.0          0.0          0.0        0.0
1.0

[6502 rows x 34 columns]
```

```
# form dataframe again if required

ohe.inverse_transform(np.array([0., 0., 1., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.,
       0., 0.]).reshape(1,34))
```

```
array([['BMW', 'CNG']], dtype=object)
```

```
# show sparse_array false

# show dtype

# drop
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.2,random_state=42)

ohe = OneHotEncoder(drop='first',sparse_output=False)
ohe.fit_transform(X_train).shape
```

```
(6502, 34)
```

```
ohe.drop_idx_
```

```
array([0, 0], dtype=object)
```

```
# handling rare categories
X_train['brand'].value_counts()
```

```
Maruti           1953
Hyundai          1127
Mahindra          635
Tata              586
Toyota            391
Honda             369
Ford              320
Chevrolet         185
Renault           183
```

```
Volkswagen        154
BMW                96
Skoda              82
Nissan             62
Jaguar             59
Volvo              54
Datsun             48
Mercedes-Benz      43
Fiat               35
Audi               30
Jeep               26
Lexus              22
Mitsubishi         13
Force               6
Land                5
Kia                 4
Ambassador          3
MG                  3
Daewoo              3
Isuzu               2
Ashok               1
Peugeot             1
Opel                1
Name: brand, dtype: int64
```

```
cars['fuel'].value_counts()
```

```
Diesel    4402
Petrol    3631
CNG         57
LPG         38
Name: fuel, dtype: int64
```

```
# using min frequency

ohe = OneHotEncoder(sparse_output=False, min_frequency=100)
ohe.fit_transform(X_train).shape
```

```
(6502, 14)
```

```
ohe.get_feature_names_out()
```

```
array(['brand_Chevrolet', 'brand_Ford', 'brand_Honda',
'brand_Hyundai',
       'brand_Mahindra', 'brand_Maruti', 'brand_Renault',
'brand_Tata',
       'brand_Toyota', 'brand_Volkswagen', 'brand_infrequent_sklearn',
       'fuel_Diesel', 'fuel_Petrol', 'fuel_infrequent_sklearn'],
      dtype=object)
```

```
# using max_categories
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore',
max_categories=15)
ohe.fit_transform(X_train).shape
```

```
(6502, 19)
```

```
ohe.get_feature_names_out()
```

```
array(['brand_BMW', 'brand_Chevrolet', 'brand_Ford', 'brand_Honda',
       'brand_Hyundai', 'brand_Jaguar', 'brand_Mahindra',
'brand_Maruti',
       'brand_Nissan', 'brand_Renault', 'brand_Skoda', 'brand_Tata',
       'brand_Toyota', 'brand_Volkswagen', 'brand_infrequent_sklearn',
       'fuel_CNG', 'fuel_Diesel', 'fuel_LPG', 'fuel_Petrol'],
dtype=object)
```

```
# how to handle unknown category
X_train,X_test,y_train,y_test =
train_test_split(X,y,test_size=0.2,random_state=42)

ohe = OneHotEncoder(drop='first',sparse_output=False)
ohe.fit_transform(X_train)

ohe.transform(np.array(['local','Petrol']).reshape(1,2))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465:
UserWarning: X does not have valid feature names, but OneHotEncoder
was fitted with feature names
  warnings.warn(

---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-97-fd88ce92ae1b> in <cell line: 7>()
      5 ohe.fit_transform(X_train)
      6
----> 7 ohe.transform(np.array(['local','Petrol']).reshape(1,2))

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_set_output.py
in wrapped(self, X, *args, **kwargs)
    155     @wraps(f)
    156     def wrapped(self, X, *args, **kwargs):
--> 157         data_to_wrap = f(self, X, *args, **kwargs)
    158         if isinstance(data_to_wrap, tuple):
    159             # only wrap the first output for cross
decomposition

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoder
s.py in transform(self, X)
```

```
   1025             "infrequent_if_exist",
   1026         }
-> 1027         X_int, X_mask = self._transform(
   1028             X,
   1029             handle_unknown=self.handle_unknown,

/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoder
s.py in _transform(self, X, handle_unknown, force_all_finite,
warn_on_unknown, ignore_category_indices)
    198                         " during transform".format(diff, i)
    199                     )
--> 200                     raise ValueError(msg)
    201                 else:
    202                     if warn_on_unknown:

ValueError: Found unknown categories ['local'] in column 0 during
transform
```

```
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
ohe.fit_transform(X_train)

ohe.transform(np.array(['local','Petrol']).reshape(1,2))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:465:
UserWarning: X does not have valid feature names, but OneHotEncoder
was fitted with feature names
  warnings.warn(

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

```
ohe.inverse_transform(np.array([0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.,
        0., 0., 0., 1.]).reshape(1,36))

array([[None, 'Petrol']], dtype=object)
```

LabelBinarizer

```
from sklearn.preprocessing import LabelBinarizer

# Sample target variable for a multi-class classification problem
y = ['cat', 'dog', 'fish', 'dog', 'cat']

# Initialize the LabelBinarizer
lb = LabelBinarizer()
```

```
# Fit and transform the target variable
y_binarized = lb.fit_transform(y)

print("Binarized labels:\n", y_binarized)

# Inverse transform to recover original labels
y_original = lb.inverse_transform(y_binarized)

print("Original labels:\n", y_original)
```

```
Binarized labels:
 [[1 0 0]
 [0 1 0]
 [0 0 1]
 [0 1 0]
 [1 0 0]]
Original labels:
 ['cat' 'dog' 'fish' 'dog' 'cat']
```

```
from sklearn.preprocessing import MultiLabelBinarizer

# Example multi-label data
y = [('red', 'blue'), ('blue', 'green'), ('green',), ('red',)]

# Initialize MultiLabelBinarizer
mlb = MultiLabelBinarizer()

# Fit and transform the data to binary matrix format
Y = mlb.fit_transform(y)

print("Binary matrix:\n", Y)
print("Class labels:", mlb.classes_)

# Inverse transform to recover original labels
y_inv = mlb.inverse_transform(Y)
print("Inverse transformed labels:", y_inv)
```

```
Binary matrix:
 [[1 0 1]
 [1 1 0]
 [0 1 0]
 [0 0 1]]
Class labels: ['blue' 'green' 'red']
Inverse transformed labels: [('blue', 'red'), ('blue', 'green'),
('green',), ('red',)]
```

3. Count Encoder/Frequency Encoder

```
!pip install category_encoders
```

```
Requirement already satisfied: category_encoders in
/usr/local/lib/python3.10/dist-packages (2.6.3)
Requirement already satisfied: numpy>=1.14.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.4.0)
Requirement already satisfied: scipy>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(0.14.1)
Requirement already satisfied: pandas>=1.0.5 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.5.3)
Requirement already satisfied: patsy>=0.5.1 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(0.5.6)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>category_encoders) (2023.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0-
>category_encoders) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0-
>category_encoders) (3.2.0)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0-
>category_encoders) (23.2)

# dataset generation
import pandas as pd
import numpy as np
import category_encoders as ce

# Simulating a dataset
data = {
    'Age': np.random.randint(20, 60, size=100).astype(float),  #
Random ages between 20 and 60
    'State': np.random.choice(['Karnataka', 'Tamil Nadu',
'Maharashtra', 'Delhi', 'Telangana'], size=100),
    'Education': np.random.choice(['High School', 'UG', 'PG'],
```

```
size=100),
    'Package': np.random.rand(100) * 100  # Random package values for
demonstration
}

# Introducing missing values in 'Age' column (5%)
np.random.seed(0)  # For reproducibility
missing_indices = np.random.choice(data['Age'].shape[0],
replace=False, size=int(data['Age'].shape[0] * 0.05))
data['Age'][missing_indices] = np.nan

df = pd.DataFrame(data)

df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 100,\n  \"fields\": [\
n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 11.43463824734759,\n
\"min\": 20.0,\n        \"max\": 59.0,\n        \"samples\": [\n
56.0,\n          47.0,\n          37.0\n        ],\n
\"num_unique_values\": 40,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"State\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"samples\": [\n          \"Delhi\",\n
\"Karnataka\",\n          \"Telangana\"\n        ],\n
\"num_unique_values\": 5,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Education\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"samples\": [\n          \"High School\",\n
\"PG\",\n          \"UG\"\n        ],\n        \"num_unique_values\":
3,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Package\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
28.649380287279953,\n        \"min\": 0.9688170796795847,\n
\"max\": 99.24295232856558,\n        \"samples\": [\n
95.37213806638319,\n          17.41119522271323,\n
87.71309658812383\n        ],\n        \"num_unique_values\": 100,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```


```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(df.drop(columns=['Package']), df['Package'],
test_size=0.2, random_state=42)

X_train.head()
```


```
{"summary":"{\n  \"name\": \"X_train\",\n  \"rows\": 80,\n
\"fields\": [\n    {\n      \"column\": \"Age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
```

```
11.418729674853827,\n        \"min\": 20.0,\n        \"max\": 59.0,\n
\"samples\": [\n          45.0,\n          56.0,\n          25.0\n
],\n        \"num_unique_values\": 36,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"State\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"samples\": [\n          \"Maharashtra\",\n
\"Karnataka\",\n          \"Delhi\"\n        ],\n
\"num_unique_values\": 5,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Education\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"samples\": [\n          \"High School\",\n
\"PG\",\n          \"UG\"\n        ],\n        \"num_unique_values\":
3,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"X_train"}
```


```
X_train['State'].value_counts()
```

```
Delhi          18
Tamil Nadu     17
Telangana      17
Maharashtra    14
Karnataka      14
Name: State, dtype: int64
```

```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd
import sklearn

class CountEncoder(BaseEstimator, TransformerMixin):
    def __init__(self, columns=None):
        self.columns = columns
        self.count_map = {}

    def fit(self, X, y=None):
        if self.columns is None:
            self.columns = X.columns
        for col in self.columns:
            self.count_map[col] = X[col].value_counts().to_dict()
        return self

    def transform(self, X):
        X = X.copy()
        for col in self.columns:
            X[col] = X[col].map(self.count_map[col]).fillna(0)
        return X

preprocessor = ColumnTransformer(
    transformers=[
```

```
        ('age_missing', SimpleImputer(strategy='mean'), ['Age']),
        ('cat_state', CountEncoder(), ['State']),
        ('education_ordinal', OrdinalEncoder(), ['Education'])
    ])

sklearn.set_config(transform_output="pandas")

preprocessor.fit_transform(X_train)
```


```
{"summary":"{\n  \"name\": \"preprocessor\",\n  \"rows\": 80,\n
\"fields\": [\n    {\n      \"column\": \"age_missing__Age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11.233238803143056,\n        \"min\": 20.0,\n        \"max\": 59.0,\n
\"samples\": [\n          48.0,\n          59.0,\n          36.0\n
],\n        \"num_unique_values\": 35,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"cat_state__State\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 2,\n        \"min\": 12,\n
\"max\": 19,\n        \"samples\": [\n          18,\n          19,\n
15\n        ],\n        \"num_unique_values\": 5,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"education_ordinal__Education\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.8220651672711271,\n        \"min\": 0.0,\n        \"max\": 2.0,\n
\"samples\": [\n          1.0,\n          0.0,\n          2.0\n
],\n        \"num_unique_values\": 3,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
# using category encoders
from category_encoders.count import CountEncoder

preprocessor = ColumnTransformer(
    transformers=[
        ('age_missing', SimpleImputer(strategy='mean'), ['Age']),
        ('cat_state', CountEncoder(normalize=True), ['State']),
        ('education_ordinal', OrdinalEncoder(), ['Education'])
    ])
sklearn.set_config(transform_output="pandas")

preprocessor.fit_transform(X_train)
```


```
{"summary":"{\n  \"name\": \"preprocessor\",\n  \"rows\": 80,\n
\"fields\": [\n    {\n      \"column\": \"age_missing__Age\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
11.05147144488219,\n        \"min\": 20.0,\n        \"max\": 59.0,\n
\"samples\": [\n          52.0,\n          26.0,\n          23.0\n
],\n        \"num_unique_values\": 37,\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"cat_state__State\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.02064764292184094,\n
```

\"min\": 0.175,\n   \"max\": 0.225,\n   \"samples\": [\n
0.2125,\n   0.175,\n   0.225\n   ],\n
\"num_unique_values\": 3,\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"education_ordinal__Education\",\n   \"properties\": {\n
\"dtype\": \"number\",\n   \"std\": 0.8335337311786837,\n
\"min\": 0.0,\n   \"max\": 2.0,\n   \"samples\": [\n
0.0,\n   1.0,\n   2.0\n   ],\n
\"num_unique_values\": 3,\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
# frequency encoding

# parameters
import pandas as pd
import numpy as np
import category_encoders as ce

# Simulating a dataset
np.random.seed(42)  # For reproducibility
data = {
    'State': np.random.choice(['Karnataka', 'Tamil Nadu',
'Maharashtra', 'Delhi', 'Telangana', np.NaN], size=100),
    'Education': np.random.choice(['High School', 'UG', 'PG', np.NaN],
size=100)
}
df = pd.DataFrame(data)

df.head(25)
```

{"summary":"{\n \"name\": \"df\",\n \"rows\": 100,\n \"fields\": [\
n   {\n   \"column\": \"State\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"samples\": [\n
\"Delhi\",\n   \"Telangana\",\n   \"Karnataka\"\n
],\n   \"num_unique_values\": 6,\n   \"semantic_type\":
\"\",\n   \"description\": \"\"\n   }\n   },\n   {\n
\"column\": \"Education\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"samples\": [\n   \"High
School\",\n   \"UG\",\n   \"PG\"\n   ],\n
\"num_unique_values\": 4,\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\
n}","type":"dataframe","variable_name":"df"}

```
df.isnull().sum()
```

```
State        0
Education    0
dtype: int64
```

```
# Initialize the CountEncoder with various parameters
encoder = ce.CountEncoder(
```

```
    cols=['State', 'Education'],  # Specify columns to encode. None
would automatically select categorical columns.
    handle_missing='error',  # Treat NaNs as a countable category
    handle_unknown='error',  # Treat unknown categories as NaNs (if
seen during transform but not in fit)
)

# Fit and transform the dataset
encoder.fit_transform(df)

#print(encoded_df.head(25))
```


```
{"summary":"{\n  \"name\": \"#print(encoded_df\",\n  \"rows\": 100,\n
\"fields\": [\n    {\n      \"column\": \"State\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
4,\n        \"min\": 11,\n        \"max\": 25,\n        \"samples\":
[\n          17,\n          19,\n          25\n        ],\n
\"num_unique_values\": 4,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Education\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 6,\n        \"min\": 16,\n
\"max\": 34,\n        \"samples\": [\n          27,\n          16,\n
34\n        ],\n        \"num_unique_values\": 4,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
encoder.mapping
```

```
{'State': Delhi          25
 Tamil Nadu     19
 Telangana      17
 nan            17
 Maharashtra    11
 Karnataka      11
 Name: State, dtype: int64,
 'Education': PG             34
 High School    27
 nan            23
 UG             16
 Name: Education, dtype: int64}
```

```
new_data = pd.DataFrame({'State': ['Bihar'], 'Education': ['UG']})

encoder.transform(new_data)
```

```
---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
<ipython-input-59-1f22ed1287ce> in <cell line: 3>()
      1 new_data = pd.DataFrame({'State': ['Bihar'], 'Education':
```

```
['UG']})
      2
----> 3 encoder.transform(new_data)

/usr/local/lib/python3.10/dist-packages/sklearn/utils/_set_output.py
in wrapped(self, X, *args, **kwargs)
    271     @wraps(f)
    272     def wrapped(self, X, *args, **kwargs):
--> 273         data_to_wrap = f(self, X, *args, **kwargs)
    274         if isinstance(data_to_wrap, tuple):
    275             # only wrap the first output for cross
decomposition

/usr/local/lib/python3.10/dist-packages/category_encoders/utils.py in
transform(self, X, override_return_df)
    486             return X
    487
--> 488         X = self._transform(X)
    489         return self._drop_invariants(X, override_return_df)
    490

/usr/local/lib/python3.10/dist-packages/category_encoders/count.py in
_transform(self, X)
    161                     and X[col].isna().any()
    162             ):
--> 163                 raise ValueError(f'Missing data found in
column {col} at transform time.')
    164         return X
    165

ValueError: Missing data found in column State at transform time.
```

```
np.random.seed(0)  # For reproducibility
data = {
    'Category': np.random.choice(['A', 'B', 'C', 'D', 'E', 'F',
np.nan], size=100, p=[0.3, 0.25, 0.15, 0.15, 0.05, 0.05, 0.05]),
    'Value': np.random.rand(100)
}

df = pd.DataFrame(data)

df.sample(10)
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Category\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"samples\": [\n          \"C\",\n
\"B\",\n          \"A\"\n        ],\n        \"num_unique_values\":
3,\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Value\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
```

0.28421869197152966,\n   \"min\": 0.14944830465799375,\n   \"max\": 0.855803342392611,\n   \"samples\": [\n   0.22392468806038013,\n   0.29007760721044407,\n   0.7044144019235328\n   ],\n   \"num_unique_values\": 10,\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

```
df['Category'].value_counts()
```

```
A      34
B      22
C      21
D      12
nan     5
F       4
E       2
Name: Category, dtype: int64
```

```
encoder = ce.CountEncoder(
    cols=['Category'],
    min_group_size=10,  # Groups with counts less than 5 will be combined
    min_group_name='salman',  # Use default naming for combined minimum groups
)

# Fit and transform the dataset
encoded_df = encoder.fit_transform(df['Category'])

# Display the original and encoded data for comparison
df['Encoded'] = encoded_df
print(df.head(20))
```

```
   Category     Value  Encoded
0         B  0.677817       22
1         D  0.270008       12
2         C  0.735194       21
3         B  0.962189       22
4         B  0.248753       22
5         C  0.576157       21
6         B  0.592042       22
7         E  0.572252       11
8       nan  0.223082       11
9         B  0.952749       22
10        D  0.447125       12
11        B  0.846409       22
12        C  0.699479       21
13        F  0.297437       11
14        A  0.813798       34
15        A  0.396506       34
```

```
16        A  0.881103       34
17        D  0.581273       12
18        D  0.881735       12
19        E  0.692532       11
```

```
encoder.mapping
```

```
{'Category': A         34
 B         22
 C         21
 D         12
 salman    11
 Name: Category, dtype: int64}
```

Binary Encoder

```
import pandas as pd
import category_encoders as ce

# Sample dataset
data = {
    'Item': ['Item1', 'Item2', 'Item3', 'Item4', 'Item5', 'Item6',
'Item7', 'Item8'],
    'Fruit': ['Apple', 'Banana', 'Cherry', 'Date', 'Elderberry',
'Fig', 'Grape', 'Honeydew']
}
df = pd.DataFrame(data)

df
```

```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"Item\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"samples\": [\n          \"Item2\",\n
\"Item6\",\n          \"Item1\"\n        ],\n
\"num_unique_values\": 8,\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Fruit\",\n      \"properties\": {\n        \"dtype\": \"string\",\n
\"samples\": [\n          \"Banana\",\n          \"Fig\",\n
\"Apple\"\n        ],\n        \"num_unique_values\": 8,\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```

```
# Initialize the Binary Encoder
encoder = ce.BinaryEncoder(cols=['Fruit'], return_df=True)

# Fit and transform the data
df_encoded = encoder.fit_transform(df)

# Display the original and encoded data
print(df_encoded)
```

```
    Item  Fruit_0  Fruit_1  Fruit_2  Fruit_3
0  Item1        0        0        0        1
1  Item2        0        0        1        0
2  Item3        0        0        1        1
3  Item4        0        1        0        0
4  Item5        0        1        0        1
5  Item6        0        1        1        0
6  Item7        0        1        1        1
7  Item8        1        0        0        0
```

Target Encoder

```
# using category_encoder

import pandas as pd
import category_encoders as ce

# Sample data
data = {
    'Feature': ['A', 'B', 'A', 'B', 'C', 'A', 'B', 'C'],
    'Target': [1, 0, 0, 1, 1, 1, 0, 1]
}
df = pd.DataFrame(data)

# Separating the feature and target columns
X = df.drop('Target', axis=1)
y = df['Target']

# Initialize the TargetEncoder
encoder = ce.TargetEncoder(cols=['Feature'])

# Fit the encoder using the feature data and target variable
encoder.fit(X, y)

# Transform the data
encoded = encoder.transform(X)

# Show the original and encoded data
print(pd.concat([df, encoded], axis=1))
```

```
   Feature  Target   Feature
0        A       1  0.631436
1        B       0  0.579948
2        A       0  0.631436
3        B       1  0.579948
4        C       1  0.678194
5        A       1  0.631436
6        B       0  0.579948
7        C       1  0.678194
```

```
encoder.mapping
```

```
{'Feature': Feature
  1    0.631436
  2    0.579948
  3    0.678194
 -1    0.625000
 -2    0.625000
 dtype: float64}
```

```
!pip install --upgrade scikit-learn==1.4.0
```

```
# using sklearn
import pandas as pd
from sklearn.preprocessing import TargetEncoder

# Sample data
data = {
    'Feature': ['A', 'B', 'A', 'B', 'C', 'A', 'B', 'C'],
    'Target': [1, 0, 0, 1, 1, 1, 0, 1]
}
df = pd.DataFrame(data)

# Separating the feature and target columns
X = df.drop('Target', axis=1)
y = df['Target']

# Initialize the TargetEncoder
encoder = TargetEncoder(smooth=0.0)

# Fit the encoder using the feature data and target variable
encoder.fit(X, y)

# Transform the data
encoded = encoder.transform(X)

encoded
```

```
array([[0.66666667],
       [0.33333333],
       [0.66666667],
       [0.33333333],
       [1.        ],
       [0.66666667],
       [0.33333333],
       [1.        ]])
```

Weight of Evidence

```
!pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 994.7 kB/s eta
0:00:00
ent already satisfied: numpy>=1.14.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.25.2)
Requirement already satisfied: scikit-learn>=0.20.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.2.2)
Requirement already satisfied: scipy>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.11.4)
Requirement already satisfied: statsmodels>=0.9.0 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(0.14.1)
Requirement already satisfied: pandas>=1.0.5 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(1.5.3)
Requirement already satisfied: patsy>=0.5.1 in
/usr/local/lib/python3.10/dist-packages (from category_encoders)
(0.5.6)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5-
>category_encoders) (2023.4)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-
packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0-
>category_encoders) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0-
>category_encoders) (3.2.0)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0-
>category_encoders) (23.2)
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

```python
import pandas as pd
import category_encoders as ce

# Example dataset
data = {
    'Feature': ['A', 'B', 'A', 'C', 'B', 'A', 'C', 'B', 'A', 'C'],
    'Target': [1, 0, 0, 1, 1, 0, 1, 0, 1, 0]
}
```

```
df = pd.DataFrame(data)

# Define the features and target
X = df[['Feature']]
y = df['Target']

# Initialize and fit the TargetEncoder
encoder = ce.WOEEncoder(cols=['Feature'])
X_encoded = encoder.fit_transform(X, y)

# Display the original and encoded data
df['Feature_Encoded'] = X_encoded
print(df)
```

```
  Feature  Target  Feature_Encoded
0       A       1         0.000000
1       B       0        -0.405465
2       A       0         0.000000
3       C       1         0.405465
4       B       1        -0.405465
5       A       0         0.000000
6       C       1         0.405465
7       B       0        -0.405465
8       A       1         0.000000
9       C       0         0.405465
```