age	department	sex	salary
50	HR	Male	10000
29	HR	Male	50000
34	Account	Female	10000
47	HR	Male	30000
52	Sales	Female	70000
36	Account	Male	24000

Why do we need encoding for categorical data?

1. **Most ML algorithms require numerical input to perform their calculations**. Categorical data needs to be transformed into a numerical format for these algorithms to use effectively.

2. **Model Performance**: Proper encoding can improve accuracy, reduce bias, and ensure correct model interpretation, especially for models sensitive to numerical relationships (like linear models or k-NN).

Label encoding

Label encoding assigns a unique integer to each category in a feature column.

age	department	salary
41	HR	10000
52	HR	50000
63	Account	10000
74	HR	30000
65	Sales	70000
56	Account	24000

Here we assign

HR → 0

Account → 1

Sales → 2

age	department	salary
41	0	10000
52	0	50000
63	1	10000
74	0	30000
65	2	70000
56	1	24000

Label encoding

Label encoding assigns a unique integer to each category in a feature column.

qty	color	price
2	red	10
4	green	50
6	red	10
3	green	30
2	red	70
7	yellow	26

Here we assign
red → 0
green → 1
yellow → 2

qty	color	price
2	0	10
4	1	50
4	0	10
3	1	30
2	0	70
7	2	26

Label encoding: Python code

```python
import pandas as pd

df = pd.DataFrame({
    'department': ['HR', 'HR', 'Account', 'HR', 'Sales', 'Sales', 'Account'],
    'salary': [50000, 45000, 55000, 52000, 48000, 43000, 28000]
})

print(df)
```

```
  department  salary
0         HR   50000
1         HR   45000
2    Account   55000
3         HR   52000
4      Sales   48000
5      Sales   43000
6    Account   28000
```

Label encoding: Python code

```python
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder() # create an instance
df['department_encoded'] = le.fit_transform(df['department'])

print(df)
```

```
  department  salary  department_encoded
0         HR   50000                   1
1         HR   45000                   1
2    Account   55000                   0
3         HR   52000                   1
4      Sales   48000                   2
5      Sales   43000                   2
6    Account   28000                   0
```

Label encoding: Python code

```python
# Lets see the label mappings
print("\nclasses:", le.classes_)
print("\nvalues:", le.transform(le.classes_))
print("\nLabel mapping:", dict(zip(le.classes_, le.transform(le.classes_))))
```

```
classes: ['Account' 'HR' 'Sales']

values: [0 1 2]

Label mapping: {'Account': 0, 'HR': 1, 'Sales': 2}
```

1 Hot Encoding

Converts categorical data into a binary format where each category is represented by a separate column:
1 indicating its presence and 0s for all other categories.

age	depart	salary
31	HR	10000
52	HR	50000
23	Account	10000
44	HR	30000
55	Sales	70000

age	depart_HR	depart_Account	depart_Sales	salary
31	1	0	0	10000
52	1	0	0	50000
23	0	1	0	10000
44	1	0	0	30000
55	0	0	1	70000

After 1-hot encoding there would be 2 additional columns

1 Hot Encoding

If u have categorical columns with **only 2 values,** then after performing 1-hot encoding, we can safely drop 1 column to **avoid duplicate information** and reduce dimensionality.

EmpID	sex	salary
1	male	10000
2	male	50000
3	female	10000
4	male	30000
5	female	70000

EmpID	sex_male	sex_female	salary
1	1	0	10000
2	1	0	50000
3	0	1	10000
4	1	0	30000
5	0	1	70000

After 1-hot encoding

EmpID	sex_male	salary
1	1	10000
2	1	50000
3	0	10000
4	1	30000
5	0	70000

After dropping column sex_female

1 Hot Encoding: Python Code

```python
import pandas as pd

# Create sample data
df = pd.DataFrame({
    'empID': [10, 12, 13, 14, 15, 16, 17],
    'department': ['HR', 'HR', 'Account', 'HR', 'Sales', 'Sales', 'Account'],
    'salary': [50000, 45000, 55000, 52000, 48000, 34000, 23000]
})

print(df)
```

```
   empID department  salary
0     10         HR   50000
1     12         HR   45000
2     13    Account   55000
3     14         HR   52000
4     15      Sales   48000
5     16      Sales   34000
6     17    Account   23000
```

1 Hot Encoding: Python Code

```python
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse_output=False, dtype=int)
encoded = encoder.fit_transform(df[['department']])

# Convert to DataFrame
encoded_df = pd.DataFrame(encoded, columns=encoder.get_feature_names_out(['department']))
print(encoded_df)
```

```
   department_Account  department_HR  department_Sales
0                   0              1                 0
1                   0              1                 0
2                   1              0                 0
3                   0              1                 0
4                   0              0                 1
5                   0              0                 1
6                   1              0                 0
```

1 Hot Encoding: Python Code

```python
# Combine with salary
final_df = pd.concat([encoded_df, df[['empID','salary']]], axis=1)
print(final_df)
```

```
   department_Account  department_HR  department_Sales  empID  salary
0                   0              1                 0     10   50000
1                   0              1                 0     12   45000
2                   1              0                 0     13   55000
3                   0              1                 0     14   52000
4                   0              0                 1     15   48000
5                   0              0                 1     16   34000
6                   1              0                 0     17   23000
```

How is 1-hot encoding better that label encoding?

Label encoding approach can create problems because it might suggest an order or ranking among categories that doesn't actually exist.
For example, assigning 0 to Red, 1 to Green, and 2 to yellow could make the model think that yellow is greater than green. This misunderstanding can negatively affect the model's performance.

qty	color	price
3	red	10
4	green	50
5	red	10
3	green	30
4	red	70
5	yellow	26

Label Encoding:
red → 0
green → 1
yellow → 2

qty	color	price
3	0	10
4	1	50
5	0	10
3	1	30
4	0	70
5	2	26

How is 1-hot encoding better that label encoding?

One-hot encoding solves this problem by creating a separate binary column for each category. This way, the model can see that each category is distinct and unrelated to the others.

qty	color	price
1	red	10
2	green	50
4	red	10
1	green	30
2	red	70
4	yellow	26

qty	color_red	color_green	color_yellow	price
1	1	0	0	10
2	0	1	0	50
4	1	0	0	10
1	0	1	0	30
2	1	0	0	70
4	0	0	1	26

When do prefer One-Hot encoding over Label encoding?

1) Categories are nominal (no intrinsic order):
Example: ["red", “yellow", "green"]
These are just labels with no hierarchy or order.

qty	color	price
1	red	10
2	green	50
4	red	10
1	green	30
2	red	70
4	yellow	26

qty	color_red	color_green	color_yellow	price
1	1	0	0	10
2	0	1	0	50
4	1	0	0	10
1	0	1	0	30
2	1	0	0	70
4	0	0	1	26

When do prefer One-Hot encoding over Label encoding?

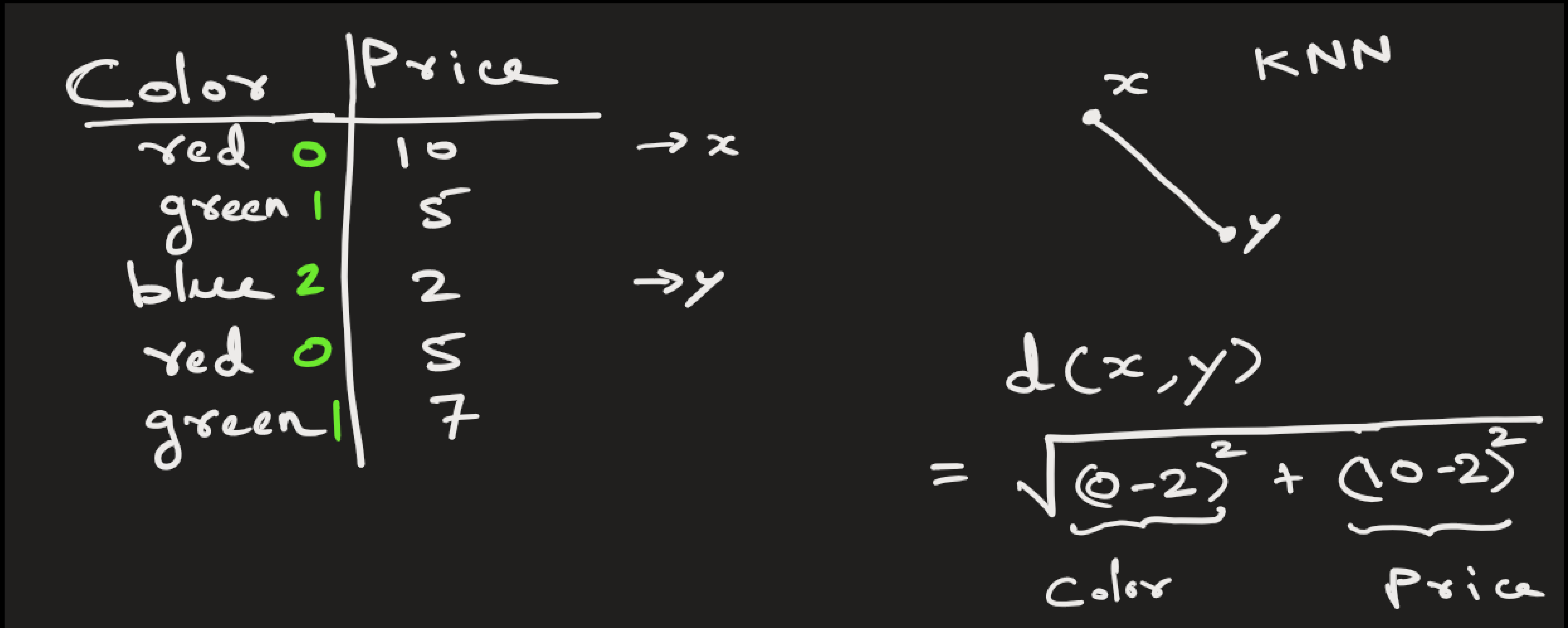
2) The number of categories is small to moderate:
One-hot encoding creates a new column for each category, so it is best when number of categories is small.

rain	size	city
5	20	Delhi
0	50	Tokyo
1	50	Delhi
2	21	Moscow
		⋮

rain	size	city_Delhi	City_Tokyo	City_Moscow	..
5	20	1	0	0	..
0	50	0	1	0	..
1	50	1	0	0	..
2	21	0	0	1	..

When do prefer One-Hot Encoding over Label encoding?

3) You are using models that assume numeric distance has meaning:
Example: K-NN, SVM, Linear Regression, Neural Network, etc.
Label encoding would wrongly imply that one category is greater than another:
(e.g., red=0, green=1, blue=2 might suggest blue > green > red, which isn't true).



When do we prefer Label Encoding over 1-hot encoding:

1) Categories are ordinal (have a natural order):
Example: ["low", "medium", "high"]
Here, label encoding preserves the ordinal relationship (low=0, medium=1, high=2).

size	price
M	1
L	5
M	2
S	3
L	3

S → 0
M → 1
L → 2

size	price
1	1
2	5
1	2
0	3
2	3

When do we prefer Label Encoding over 1-hot encoding:

2) The number of categories is very high (high cardinality):
One-hot would create too many columns, making it computationally expensive and sparse.
Example: ZIP codes, user IDs, or product SKUs.

zip-code	price
61022	1
61027	3
62055	6
. .	. .

When do we prefer Label Encoding over 1-hot encoding:

3) You are using tree-based models:
Decision trees, random forests, XGBoost, LightGBM can handle label encoded features well.
These models don’t rely on the numerical order as much since they split on feature values and thresholds.


Which encoding should we use? (p3)

Scenario	**Preferred Encoding**	**Why**
Nominal (unordered) categories	One-hot encoding	Prevents false ordinal relationships
Ordinal (ordered) categories	Label encoding	Preserves order
Small number of categories	One-hot encoding	Manageable feature expansion
Large number of categories	Label encoding	Prevents explosion in dimensionality
Using tree-based models	Label encoding	Trees are robust to numeric labels
Using linear or distance-based models	One-hot encoding	Prevents model misinterpreting category distances