Feature Engineering

Categorical encoding

```
One-Hot encoding (OHE)
Ordinal encoding
Count and Frequency encoding
Target encoding / Mean encoding
```

```
It is a commonly used technique for encoding categorical variables. It
basically creates binary variables for each category present in the
categorical variable. These binary variables will have 0 if it is absent
in the category or 1 if it is present. Each new variable is called a dummy
variable or binary variable.
```

In [2]:
```
import pandas as pd
```

In [3]:
```
data = {
    'Color': ['Red', 'Green', 'Blue']
}
df = pd.DataFrame(data)

# One-Hot Encoding
one_hot_encoded_df = pd.get_dummies(df, columns=['Color'])
print(one_hot_encoded_df)
```

```
   Color_Blue  Color_Green  Color_Red
0       False        False       True
1       False         True      False
2        True        False      False
```

Ordinal Encoding

```
It simply means a categorical variable whose categories can be ordered and
that too meaningfully.
```

In [15]:
```
from sklearn.preprocessing import *
import pandas as pd
le = preprocessing.LabelEncoder()
```

In [40]:
```python
data = {
    'Size': ['Small', 'Medium', 'Large']
}
df = pd.DataFrame(data)

ordinal_encoder = OrdinalEncoder(categories=[['Small', 'Medium', 'Large']])
df['Size'] = ordinal_encoder.fit_transform(df[['Size']])

print(df)
inverse_transformed = ordinal_encoder.inverse_transform([[0], [2], [1]])
print(list(inverse_transformed.flatten()))
```

```
   Size
0   0.0
1   1.0
2   2.0
['Small', 'Large', 'Medium']
```

Label Encoding

Label encoding assigns a unique integer to each category. This is useful for nominal (unordered) categories. However, it should be used with caution as it can sometimes introduce unintended ordinal relationships between categories.

In [43]:
```python
data = {
    'Animal': ['Cat', 'Dog', 'Mouse']
}
df = pd.DataFrame(data)

# Label Encoding
label_encoder = LabelEncoder()
df['Animal'] = label_encoder.fit_transform(df['Animal'])

print(df)

# Inverse Transform
inverse_transformed = label_encoder.inverse_transform(df['Animal'])
print(list(inverse_transformed))
```

```
   Animal
0       0
1       1
2       2
['Cat', 'Dog', 'Mouse']
```

Count and Frequency Encoding

Count Encoding replaces each category with the count of its occurrences.Replacement can also be done with the frequency of the percentage of observations in the dataset.Suppose, if 30 of 100 genders are male we can replace male with 30 or by 0.3.

In [41]:
```
data = {
    'City': ['New York', 'Paris', 'New York', 'London', 'Paris', 'Paris']
}
df = pd.DataFrame(data)

# Count Encoding
count_encoding = df['City'].value_counts()
df['City_Count'] = df['City'].map(count_encoding)

# Frequency Encoding
frequency_encoding = df['City'].value_counts(normalize=True)
df['City_Frequency'] = df['City'].map(frequency_encoding)

print(df)
```

```
       City  City_Count  City_Frequency
0  New York           2        0.333333
1     Paris           3        0.500000
2  New York           2        0.333333
3    London           1        0.166667
4     Paris           3        0.500000
5     Paris           3        0.500000
```

Target / Mean Encoding

```
Target Encoding replaces each category with the mean of the target
variable for that category. This is useful when dealing with high
cardinality categorical features.But it may cause over-fitting to the
model
```

In [42]:
```
data = {
    'City': ['New York', 'Paris', 'New York', 'London', 'Paris', 'Paris'],
    'Price': [100, 200, 150, 300, 250, 200]
}
df = pd.DataFrame(data)

# Target Encoding
target_mean = df.groupby('City')['Price'].mean()
df['City_Target_Encoded'] = df['City'].map(target_mean)

print(df)
```

```
       City  Price  City_Target_Encoded
0  New York    100           125.000000
1     Paris    200           216.666667
2  New York    150           125.000000
3    London    300           300.000000
4     Paris    250           216.666667
5     Paris    200           216.666667
```

Variable Transformation

```
Machine learning algorithms like linear and logistic regression assume
that the variables are normally distributed. If a variable is not normally
distributed, sometimes it is possible to find a mathematical
transformation so that the transformed variable is Gaussian.
```

In [ ]:
```
#Logarithm transformation - log(x)
np.log10(data)
# Square root transformation - sqrt(x)
np.sqrt(data)
# Reciprocal transformation - 1 / x
1/data
# Exponential transformation - exp(x)
np.exp(data)
```

Discretization

```
Discretization, also known as binning, is the process of transforming
continuous data into discrete categories or bins. This can be useful for
simplifying data, handling outliers, or preparing data for certain types
of models, such as decision trees, which can work better with categorical
data.
```

```
Equal Width Binning: Divides the data range into equal-sized interv
als.
Equal Frequency Binning: Divides the data so that each bin has the
same number of points.
Custom Binning: Allows you to specify custom bin edges.
KBinsDiscretizer: Offers more advanced binning techniques (uniform,
quantile, k-means).
```

Equal Width Binning

This method divides the range of the data into equal-sized intervals

In [45]:

```
# Sample data
data = {
    'Value': [1, 10, 20, 30, 40]
}
df = pd.DataFrame(data)

# Equal width binning
df['Binned_Value'] = pd.cut(df['Value'], bins=3)

print(df)
```

```
   Value   Binned_Value
0      1  (0.961, 14.0]
1     10  (0.961, 14.0]
2     20   (14.0, 27.0]
3     30   (27.0, 40.0]
4     40   (27.0, 40.0]
```

Equal Frequency Binning

This method divides the data into bins such that each bin contains the same number of data points.

In [48]:

```
# Sample data
data = {
    'Value': [1, 10, 20, 30, 40]
}
df = pd.DataFrame(data)

# Equal frequency binning
df['Binned_Value'] = pd.qcut(df['Value'], q=3)

print(df)
```

```
   Value      Binned_Value
0      1   (0.999, 13.333]
1     10   (0.999, 13.333]
2     20  (13.333, 26.667]
3     30    (26.667, 40.0]
4     40    (26.667, 40.0]
```

Custom Binning

Custom binning allows you to define your own bin edges.

In [51]:

```

# Sample data
data = {
    'Value': [1, 10, 20, 30, 40]
}
df = pd.DataFrame(data)

# Custom binning
bins = [0, 10, 30, 40]
labels = ['(0, 10]', '(10, 30]', '(30, 40]']
df['Binned_Value'] = pd.cut(df['Value'], bins=bins, labels=labels, include_l

print(df)
```

```
   Value Binned_Value
0      1      (0, 10]
1     10      (0, 10]
2     20     (10, 30]
3     30     (10, 30]
4     40     (30, 40]
```

Discretization using KBinsDiscretizer

KBinsDiscretizer from sklearn can be used for more advanced binning techniques, such as uniform, quantile, and k-means binning.

In [50]:

```
from sklearn.preprocessing import KBinsDiscretizer

# Sample data
data = {
    'Value': [1, 10, 20, 30, 40]
}
df = pd.DataFrame(data)

# KBinsDiscretizer
kbins = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
df['Binned_Value'] = kbins.fit_transform(df[['Value']])

print(df)
```

```
   Value  Binned_Value
0      1           0.0
1     10           0.0
2     20           1.0
3     30           2.0
4     40           2.0
```

Types of data

Qualitative data - non-numerical data (categorical data)

Quantitative data - numerical data

Qualitative data (categorical data)

Nominal (eye color) =>(defined categories)

Ordinal (high,mid,low) =>(ordered categories)

Binary (Yes/No)

Quantitative data

Continuous data

(measurements that are rounded =>measured characteristics)

Interval (distance between two points is standardized and equal) eg: temperature since it can move below and above 0

ratio (scale starts at 0.0 & all the properties of interval data & true zero) eg: wight,height

Interval data vs. ratio data Ratio data gets its name because the ratio of two measurements can be interpreted meaningfully, whereas two measurements cannot be directly compared with intervals.Similarly, 40º is not twice as hot as 20º. Saying uses 0º as a reference point to compare the two temperatures, which is incorrect.

Discrete data

(exact values or whole numbers that are not rounded => counted items)

Count Data Data representing counts of occurrences or events.

Binary data (0 /1)

Nominal data =>Data where values are discrete and represent categories without inherent order. (1,3,8,2)

Ordinal data Data where values are discrete and represent ordered categories. => (1,2,3,4)