# Data Imbalance Handling

GROKKERS

AI FOR EVERYONE

# Pre-reqs

- Python
- NumPy and PANDAS, SciPy, Visualizations
- Elementary stats and maths
- Some preprocessing steps – may need ML as well, for advanced topics

# Background

**Numeric Data:** Preprocessing involves handling missing values, scaling to a similar range, and possibly normalizing the distribution.

Text Data**:** Common preprocessing steps include text cleaning (removing stop words, punctuation, etc.), tokenization, and vectorization (converting text into numerical form, such as TF-IDF or word embeddings).

**Image Data**: Techniques like resizing, normalization of pixel values, and data augmentation (creating variations of existing images) are often used.

**Time Series Data**: Dealing with temporal aspects, handling missing values over time, and creating lag features are important steps in preprocessing time series data.

## Topics

- About Data, feature types, tabular form
- General inspection of data quality
- Handling duplicates in data
- Missing value analysis (2 parts)
- Handling Outliers
- Cardinality assessment
- Encoding of discrete data
- Scaling and Normalization
- Data Imbalance Handling
- **Data Splitting**

## Attribute transformations

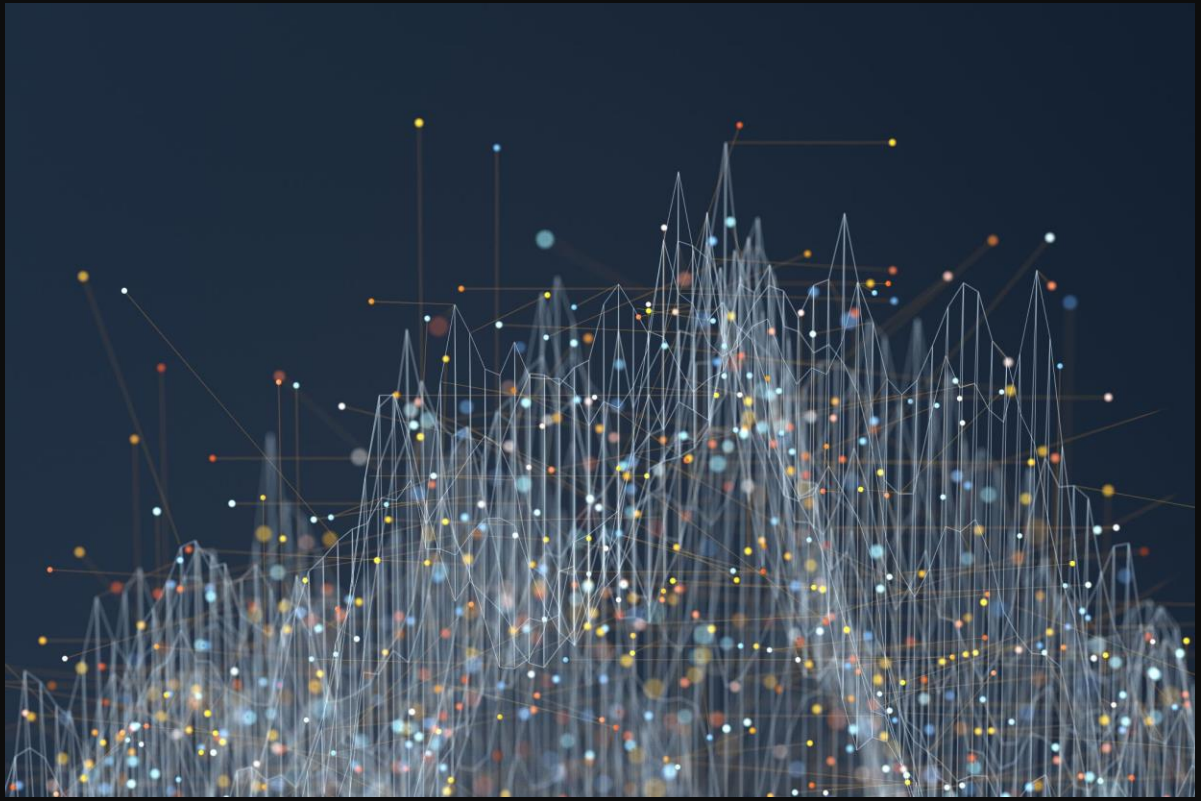
modifying the original features (attributes) in a dataset to create new representations

# Logarithmic Transformation

- Logarithmic transformation is a mathematical operation that involves taking the logarithm of a variable.

- useful when dealing with data that spans multiple orders of magnitude or exhibits exponential growth.

- most common logarithmic transformation is the natural logarithm (base e), denoted as $log_e(x)$ or simply $\ln(x)$.



## How?

- **Formula:**
  - Logarithmic Transformation: $y = \log_e(x)$
- **Example:**
  - Consider a dataset with the following values for a variable
    - $x$: [1,10,100,1000]
  - Applying the natural logarithm:
    - $\ln(x) = [\ln(1), \ln(10), \ln(100), \ln(1000)]$
- The transformed values would be:
  - [0,2.3026,4.6052,6.9078]

## Benefits

**Mitigating Skewness:**

Logarithmic transformation can help mitigate right-skewness in the distribution of a variable, making the data more symmetric.

**Stabilizing Variance:**

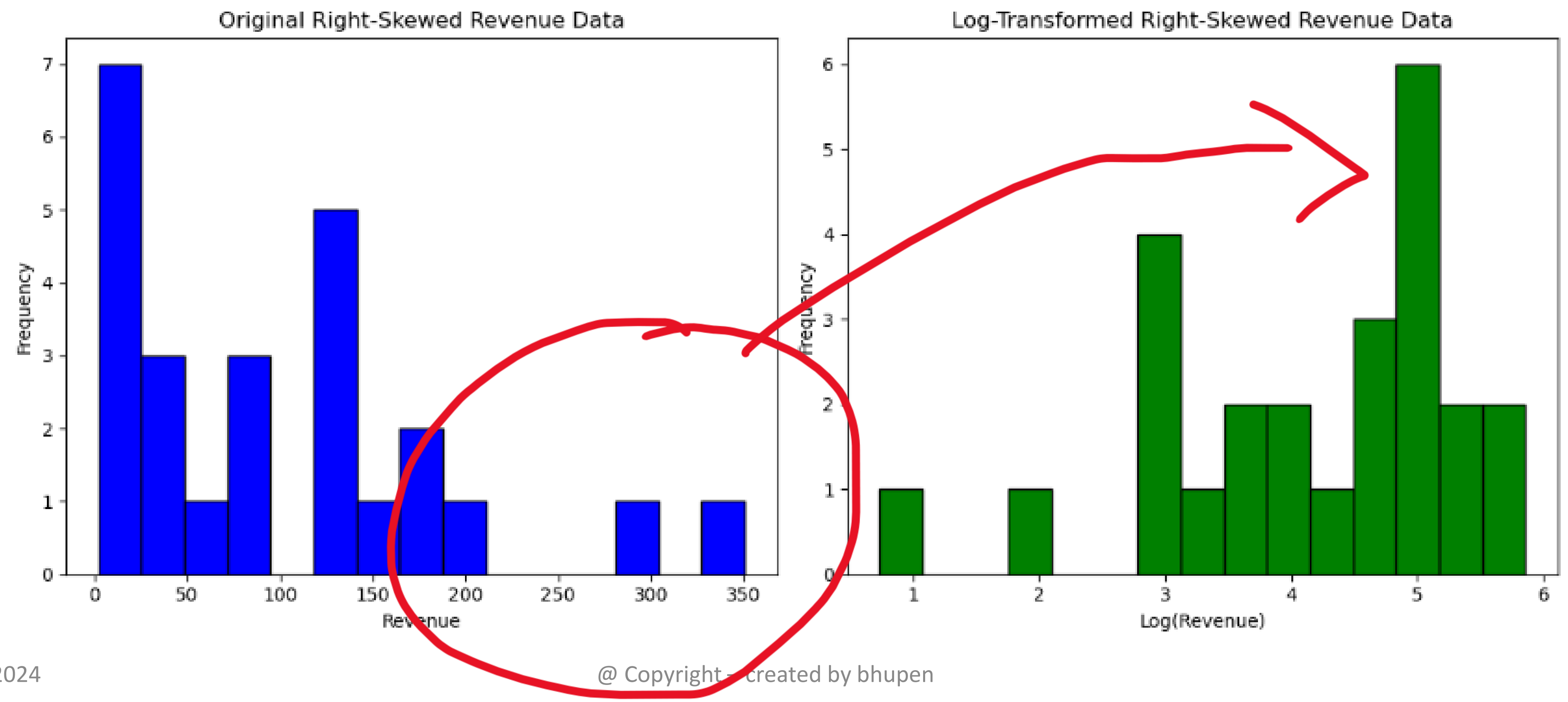
It can stabilize the variance across different levels of a variable, particularly when the data exhibits heteroscedasticity.

**Handling Multiplicative Relationships:**

Useful when dealing with relationships that involve multiplication, as the logarithm transforms multiplication into addition.

# Example

the synthetic dataset to have more density on the right-skewed side.



# square root transformation

01

involves taking the square root of each value in a dataset.

02

often used as a data transformation technique to mitigate the impact of right-skewed distributions, especially when the data contains values that vary across several orders of magnitude.

03

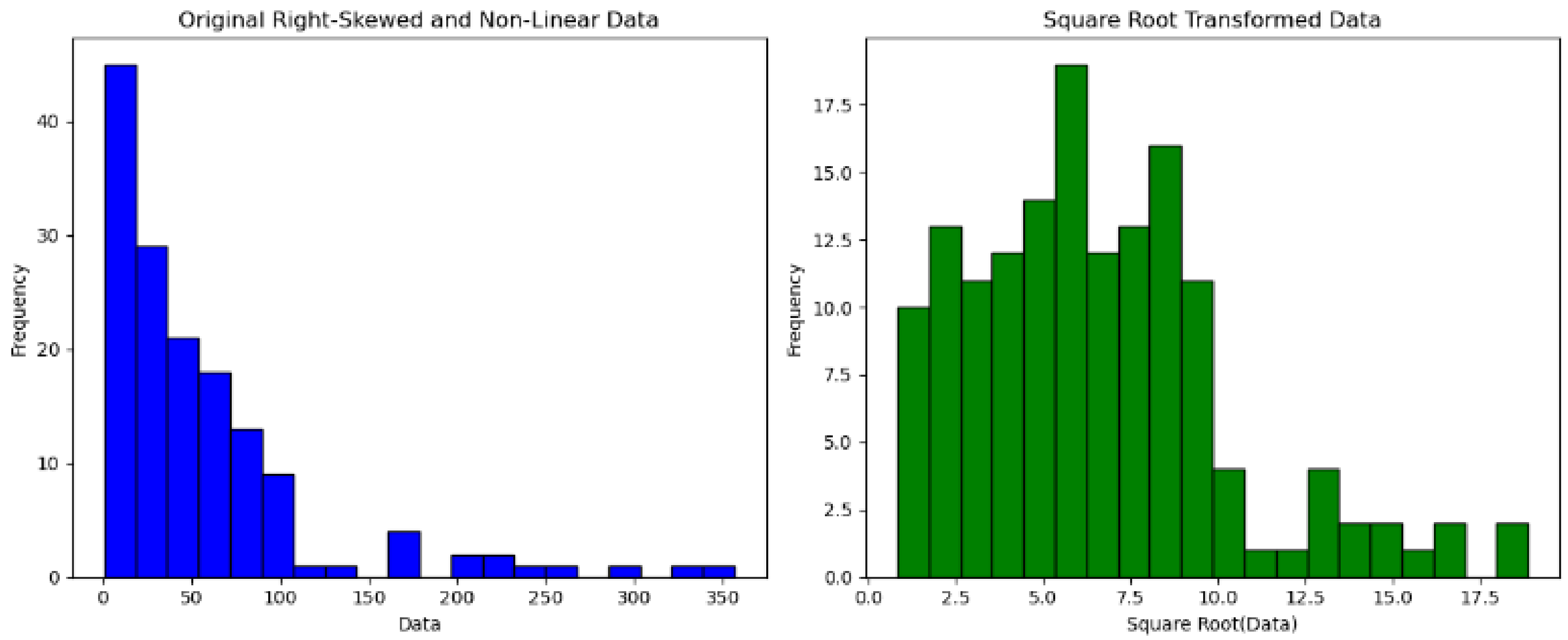
useful when the data exhibits a nonlinear relationship, and it can help make the distribution more symmetric.

# Example

Consider a dataset with the following values for a variable *x*:[1,4,9,16,25]

Applying the square root transformation: $\sqrt{x} = [\sqrt{1}, \sqrt{4}, \sqrt{9}, \sqrt{16}, \sqrt{25}]$

The transformed values would be: [1,2,3,4,5]



General idea of data imbalance, what it is , cover the outline of a simple data generation technique to handle imbalance

Addressing the data imbalance requires ML models (even the modern generative models)

## Scope of discussion

# Define data imbalance

1

Data imbalance occurs when the distribution of classes in a classification dataset is uneven, with one or more classes having significantly fewer samples than others.

2

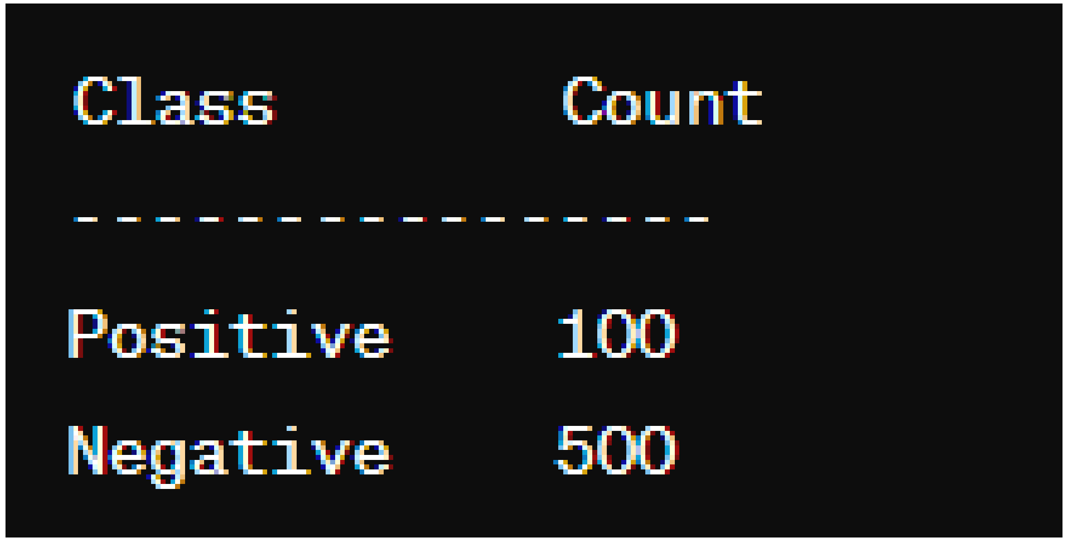
Handling data imbalance is important because it can lead to biased models that perform well on the majority class but poorly on the minority class.

# how data imbalance is measured

**Class Distribution**:

count the number of instances for each class in the dataset.

If one class has significantly fewer instances than the other, it indicates an imbalance.



```
Class         Count
------------------
Positive     100
Negative     500
```

## Example

- The "Positive" class has 100 instances.
- The "Negative" class has 500 instances.

$$\text{Class Proportion} = \frac{\text{Number of instances in the Positive class}}{\text{Number of instances in the Negative class}} = \frac{100}{500} = 0.2$$

$$\text{Imbalance Ratio} = \frac{\text{Number of instances in the Negative class}}{\text{Number of instances in the Positive class}} = \frac{500}{100} = 5$$

- The class proportion is 0.2, indicating that the Positive class is underrepresented compared to the Negative class.
- The imbalance ratio is 5, suggesting that the Negative class is five times larger than the Positive class.

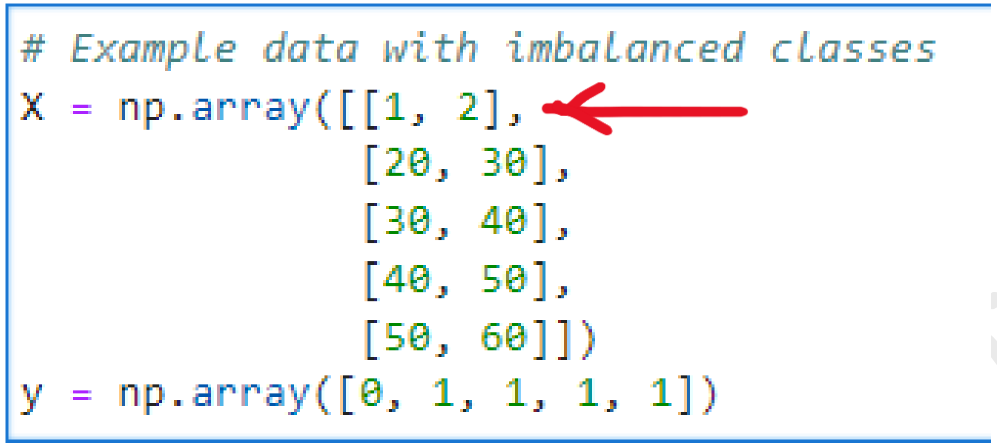
## Techniques for handling data imbalance

- Resampling Techniques
- Weighted Classes
- Ensemble Methods
- Data Augmentation
- Transfer Learning
- Generate Synthetic Data
- Anomaly Detection

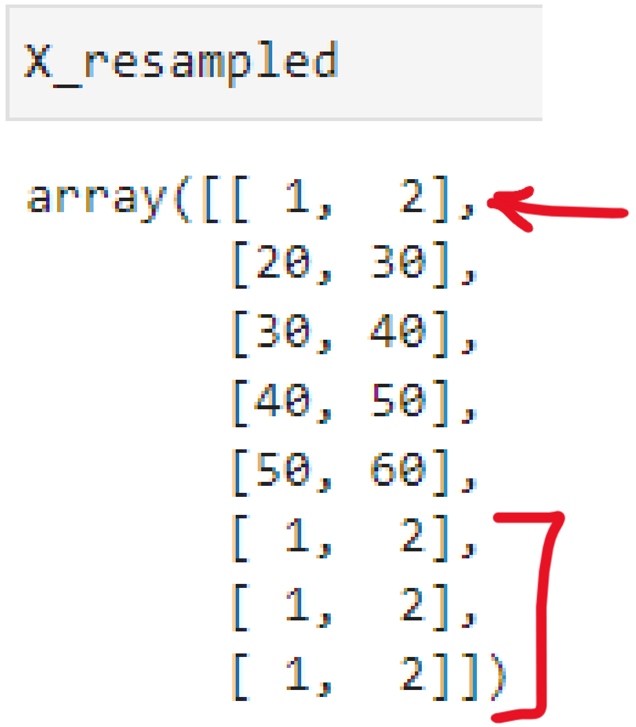
## Naive random over-sampling

- generate new samples in the classes which are under-represented.
- The naivest strategy is to generate new samples by randomly sampling with replacement the current available samples

## How does it work



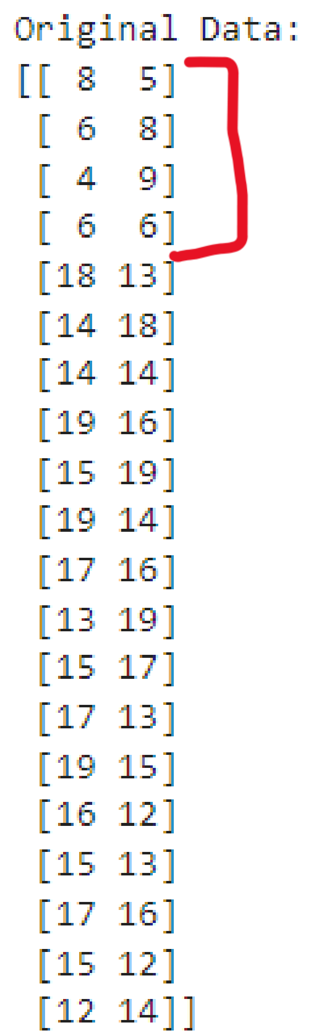
```
# Example data with imbalanced classes
X = np.array([[1, 2],
              [20, 30],
              [30, 40],
              [40, 50],
              [50, 60]])
y = np.array([0, 1, 1, 1, 1])
```



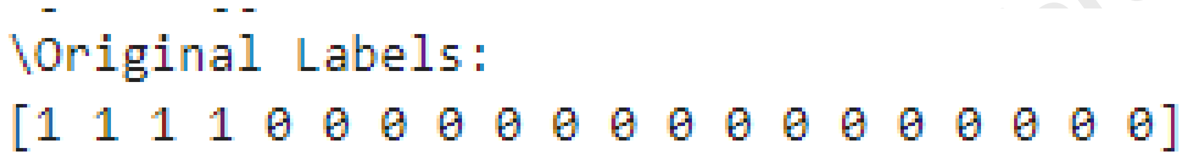
```
X_resampled
```

```
array([[ 1,  2],
       [20, 30],
       [30, 40],
       [40, 50],
       [50, 60],
       [ 1,  2],
       [ 1,  2],
       [ 1,  2]])
```

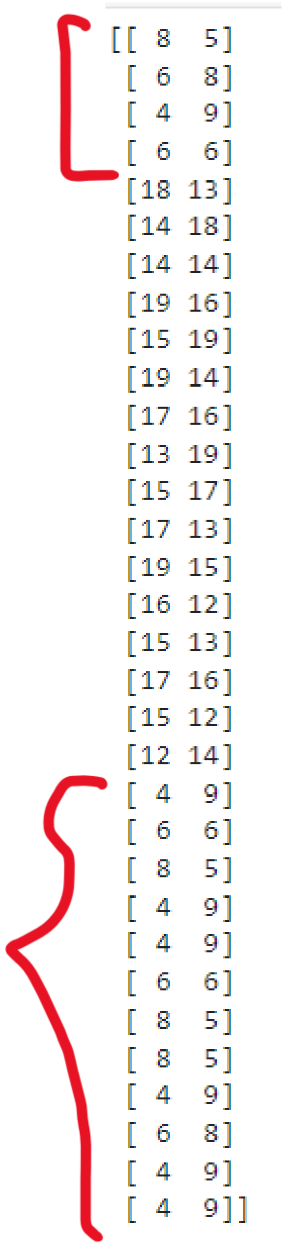
# Another example



```
Original Data:
[[ 8  5]
 [ 6  8]
 [ 4  9]
 [ 6  6]
 [18 13]
 [14 18]
 [14 14]
 [19 16]
 [15 19]
 [19 14]
 [17 16]
 [13 19]
 [15 17]
 [17 13]
 [19 15]
 [16 12]
 [15 13]
 [17 16]
 [15 12]
 [12 14]]
```



```
\Original Labels:
[1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```



```
[[ 8  5]
 [ 6  8]
 [ 4  9]
 [ 6  6]
 [18 13]
 [14 18]
 [14 14]
 [19 16]
 [15 19]
 [19 14]
 [17 16]
 [13 19]
 [15 17]
 [17 13]
 [19 15]
 [16 12]
 [15 13]
 [17 16]
 [15 12]
 [12 14]
 [ 4  9]
 [ 6  6]
 [ 8  5]
 [ 4  9]
 [ 4  9]
 [ 6  6]
 [ 8  5]
 [ 8  5]
 [ 4  9]
 [ 6  8]
 [ 4  9]
 [ 4  9]]
```

## Is random over sampler at all useful?

**Risk of Overfitting:**

- Duplicating instances from the minority class may lead to overfitting, as the model could memorize the duplicated instances instead of learning generalizable patterns. This is a common concern with any form of oversampling.

**No Generation of New Information:**

- Random OverSampler does not generate new synthetic samples. If the minority class is diverse and lacks representation, more advanced oversampling methods like SMOTE or ADASYN might be more beneficial.

**May Not Address Data Distribution Gaps:**

- Random OverSampler may not address distribution gaps or specific patterns in the minority class.

## SMOTE

SMOTE, which stands for Synthetic Minority Over-sampling Technique, is an oversampling technique designed to address class imbalance by generating synthetic samples for the minority class.

Unlike Random OverSampler, SMOTE does not duplicate existing instances but creates new synthetic instances by interpolating between neighboring instances in the minority class.

## Step 1 - Identify Minority Class Instances

This step involves determining which class in the dataset has fewer instances.

In the context of SMOTE, this is typically the class you want to oversample, referred to as the minority class.

# Step 2 and 3

| | |
|---|---|
| **Select a Minority Instance:** | A single instance is randomly chosen from the minority class. |
| **Find k-Nearest Neighbors:** | The k-nearest neighbors of the selected instance within the minority class are identified. The choice of k is a parameter set by the user, defining the number of neighbors to consider. |

## Step 4

**Generate Synthetic Instances:**

- For each neighbor, a synthetic instance is created by calculating the vector between the selected instance and the neighbor.
- This vector is scaled by a random value between 0 and 1.
- The scaled vector is added to the selected instance to generate a new synthetic instance.

## Illustrating step 4

- Suppose we have a minority class instance A with two features:
  - Instance A: (x_A, y_A) = (3, 5)
- Let's say we choose k=2, and the two nearest neighbors of instance A in the minority class are B and C:
  - Neighbor B: (x_B, y_B) = (4, 4)
  - Neighbor C: (x_C, y_C) = (2, 6)
- Now, we'll generate synthetic instances for instance A based on its neighbors.

## Generate synthetic data

1. **Calculate Vectors:**
   1. Vector BA = (x_A - x_B, y_A - y_B) = (3 - 4, 5 - 4) = (-1, 1)
   2. Vector CA = (x_A - x_C, y_A - y_C) = (3 - 2, 5 - 6) = (1, -1)
2. **Scale Vectors:**
   1. Choose a random value between 0 and 1, let's say r = 0.3
   2. Scaled Vector BA = r * Vector BA = 0.3 * (-1, 1) = (-0.3, 0.3)
   3. Scaled Vector CA = r * Vector CA = 0.3 * (1, -1) = (0.3, -0.3)
3. **Generate Synthetic Instances:**
   1. Synthetic Instance 1 = Instance A + Scaled Vector BA = (3, 5) + (-0.3, 0.3) = (2.7, 5.3)
   2. Synthetic Instance 2 = Instance A + Scaled Vector CA = (3, 5) + (0.3, -0.3) = (3.3, 4.7)

## Step 5

**Repeat the Process:**

- Steps 2-4 are repeated for a specified number of times or until the desired level of oversampling is achieved.

## Key idea

- synthetic instances (2.7, 5.3) and (3.3, 4.7) are the new instances generated for the minority class instance A
- In a more realistic scenario, the feature space would have more dimensions, and the process would involve calculating vectors and scaling them in each dimension.
- The idea is to create diverse synthetic instances around the minority class instance to improve model generalization.

# Thanks !!

AI FOR EVERYONE

Grokkers

CONCEPTS AND EQUATIONS MADE SIMPLE