Data Science & Machine Learning

Eman Raslan

Data Exploration & Preparation

Feature Transformation

According to a study by Gartner, 87% of data science projects never go into **production**.

And again, studies show that preparing the dataset for the model is the most time-consuming task (80%).

The most important thing in a machine learning model is the features.

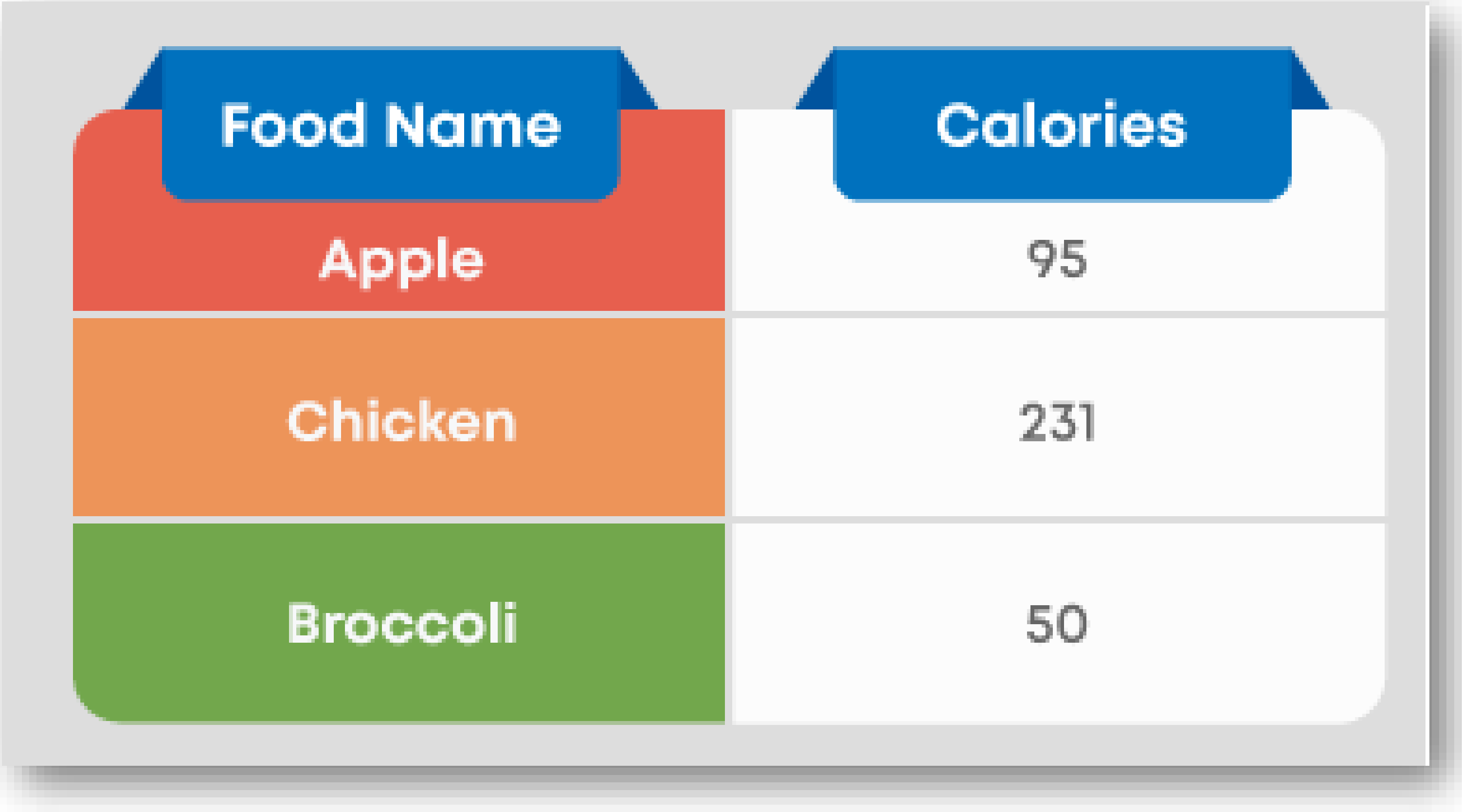
Feature Transformation

- Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.
- It is the secret weapon that advanced data scientists use to extract the most accurate results from algorithms.

Why Transformation?

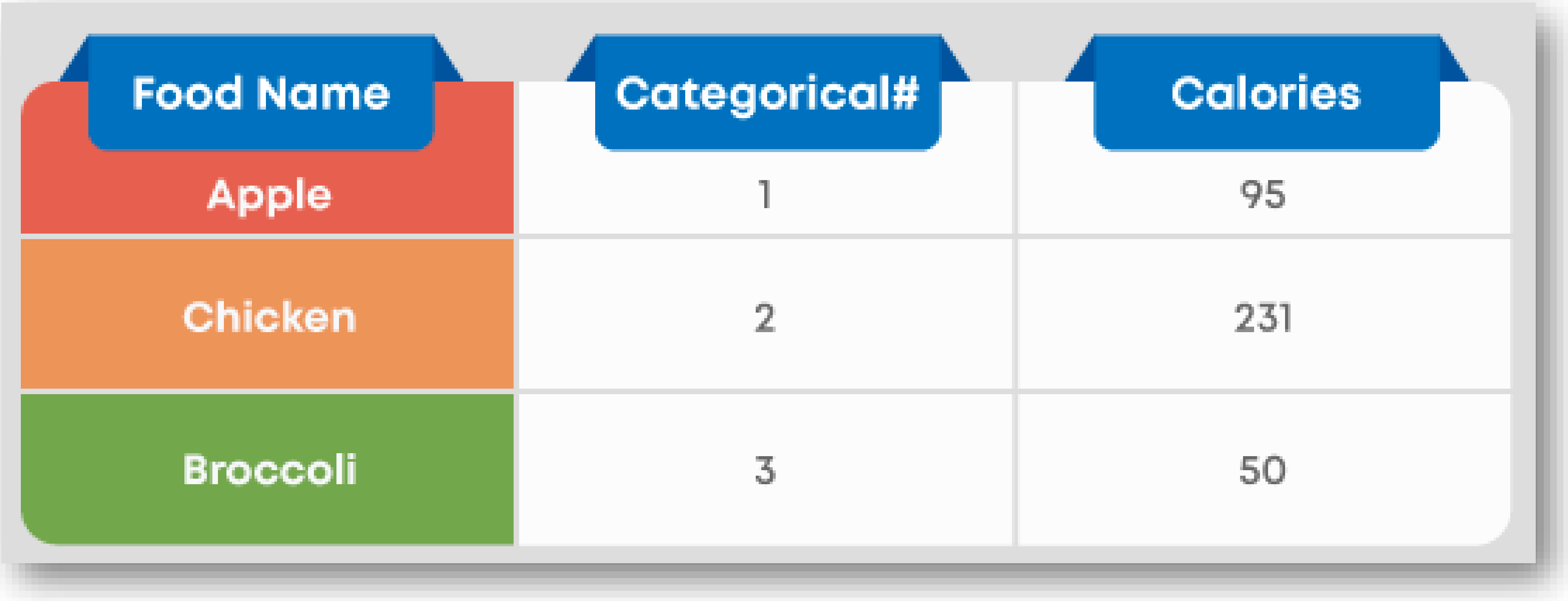
- It is mandatory to digitize categorical features for models to work properly. This is called **encoding**.
- The **scale** between features in the dataset can be very different from each other (or they may have different units). For example, while the “Age” feature varies between 0–100, “Car Price” can vary between 0–1000000. Some machine learning methods are affected by these scale differences, so normalizing the difference will contribute to the success of the model.
- Models like KNN and SVM are **distance-based** algorithms means that the distance between points is used to obtain clusters or to find out similarities. The distance of unscaled features would also be unscaled and will be misleading the model. In addition, models using gradient descent optimization such as linear regression, logistic regression, or neural networks are also negatively affected by unscaled data. Coefficients of linear models are also affected by unscaled features.
- Transformation decreases the effects of **outliers** since the variability is reduced by scaling.
- It improves model **performance** regarding the non-linear relationship between the target feature and the independent feature.
- Some machine learning models are based on the assumption that the features are **normally distributed**. However, in real-life problems, the data is usually not normally distributed. In this case, we apply transformations to approximate these skewed data to the normal distribution so that the models can yield better results.

Transforming Categorical Data



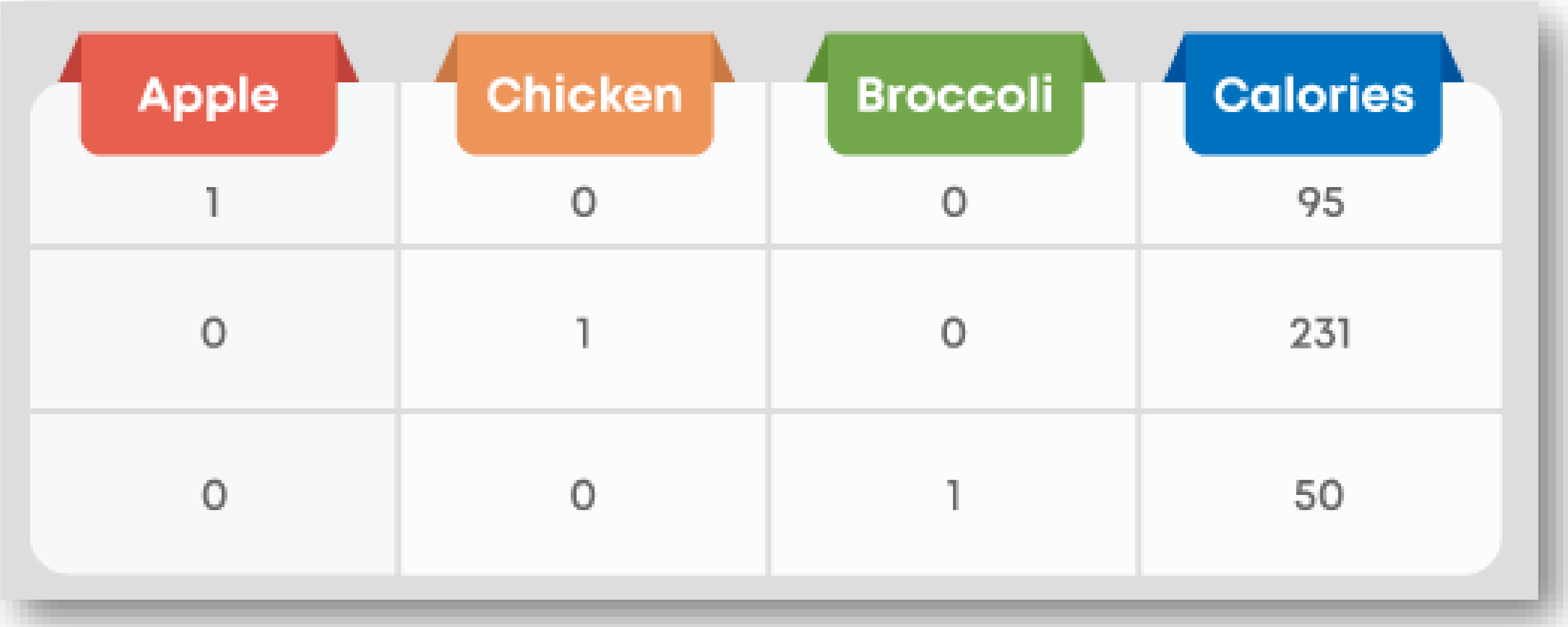
Food Name	Calories
Apple	95
Chicken	231
Broccoli	50

Label Encoding



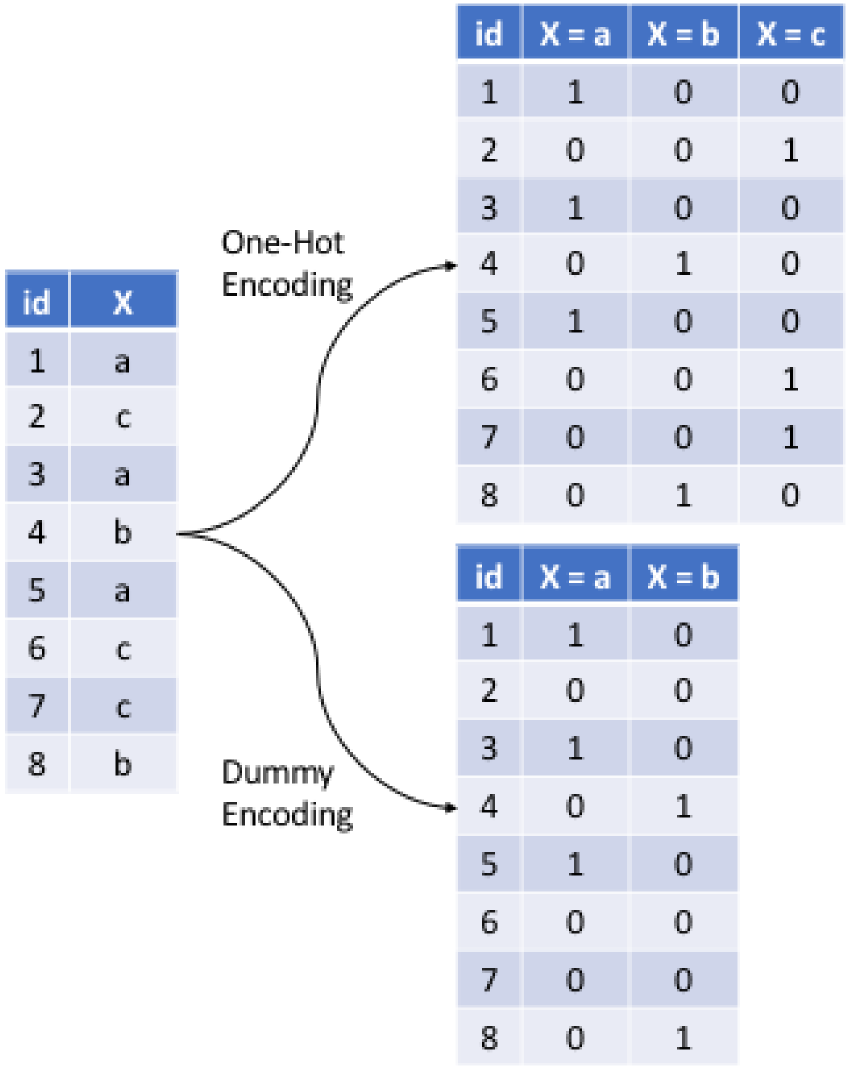
Food Name	Categorical#	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

One-Hot Encoding



Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

One-Hot Encoding



id	X
1	a
2	c
3	a
4	b
5	a
6	c
7	c
8	b

id	X = a	X = b	X = c
1	1	0	0
2	0	0	1
3	1	0	0
4	0	1	0
5	1	0	0
6	0	0	1
7	0	0	1
8	0	1	0

id	X = a	X = b
1	1	0
2	0	0
3	1	0
4	0	1
5	1	0
6	0	0
7	0	0
8	0	1

OneHotEncoder object stores the information about categories from the training dataset.
make sure that the final features will be the same as the training data.

get_dummies method doesn’t store the information about train data categories.

Label Encoding Vs One-Hot Encoding

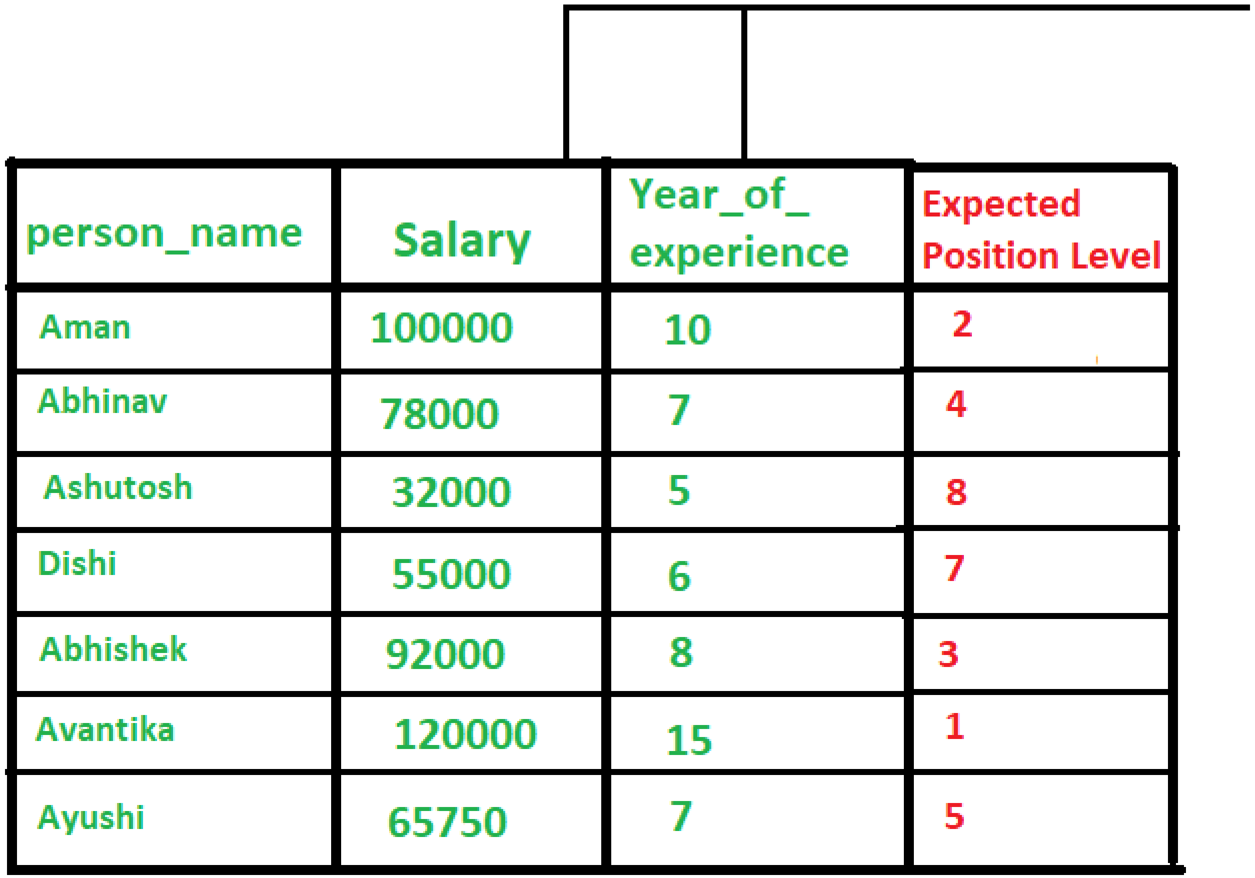
- **Use one-hot encoding**
  1. When the categorical feature has no order or is not ordinal.
  2. When the number of unique categorical features is less. This is because more features increase the model's complexity and training time.
- **Use label encoding**
  1. When the categorical feature has some order or is ordinal.
  2. When the number of categories is large.

Is it Mandatory to encode categorical variables before modeling?

- The answer is **Not really**.
- Some machine learning algorithms handle the categorical data automatically by themselves whereas the rest of the algorithms expect only the numerical columns. CatBoost, Light GBM, and XGBoost are the machine learning models where you can pass your data set having numerical as well as categorical data to the model. You just need to specify the index of categorical columns so that model handles it internally.
- Whereas models like Logistic Regression, Linear Regression, Random Forest **etc** can work only on numerical dataset.

- Categorical Lab

Feature Scaling



person_name	Salary	Year_of_ experience	Expected Position Level
Aman	100000	10	2
Abhinav	78000	7	4
Ashutosh	32000	5	8
Dishi	55000	6	7
Abhishek	92000	8	3
Avantika	120000	15	1
Ayushi	65750	7	5

The attributes salary and year_of_experience are on different scale and hence attribute salary can take high priority over attribute year_of_experience in the model.

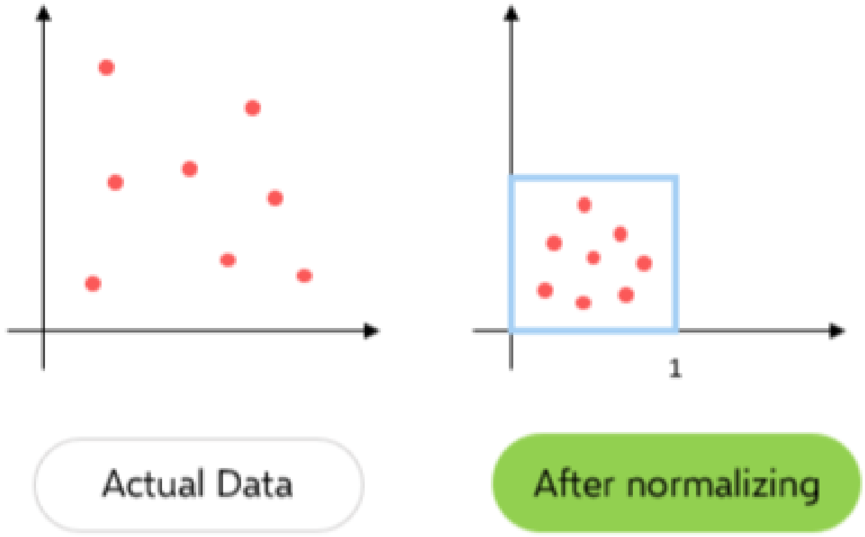
Helps the model learn appropriate weights for each feature. Without feature scaling, the model will pay too much attention to the features having a wider range.

Which to Scale

- Algorithms that require Feature Scaling:
  - Linear Regression
  - Logistic Regression
  - KNN
  - SVM
  - KMeans
  - Hierarchical clustering
  - Neural Networks
  - PCA
- Algorithms that donot require Feature Scaling:
  - Naive Bayes
  - Decision Tree
  - Random Forest
  - AdaBoost
  - XGBoost

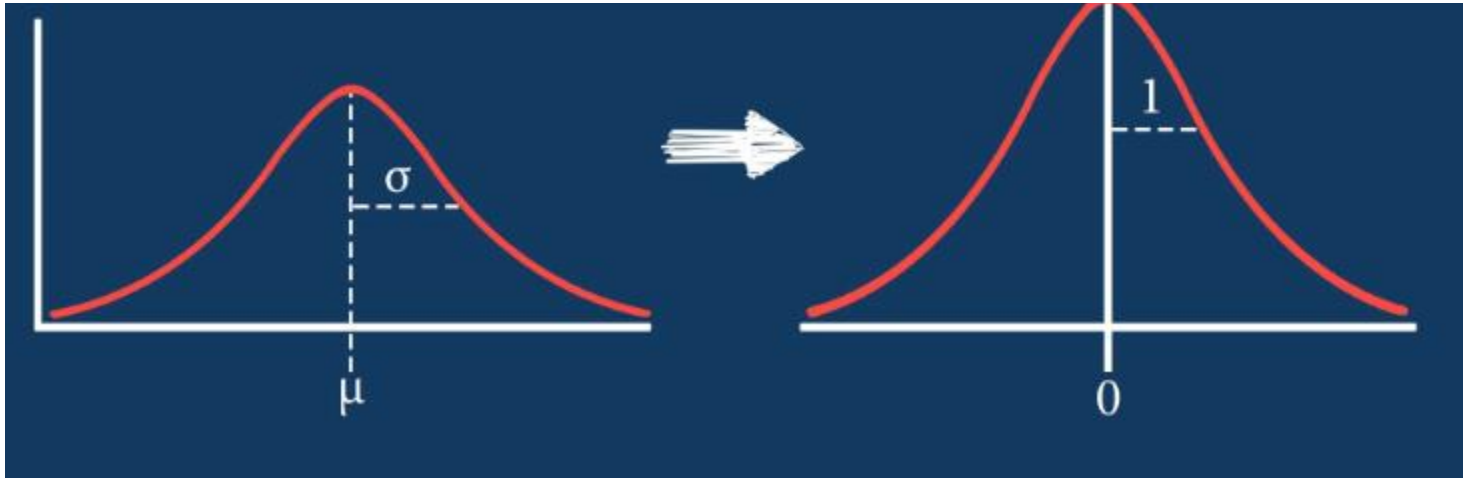
Min-Max Normalization

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

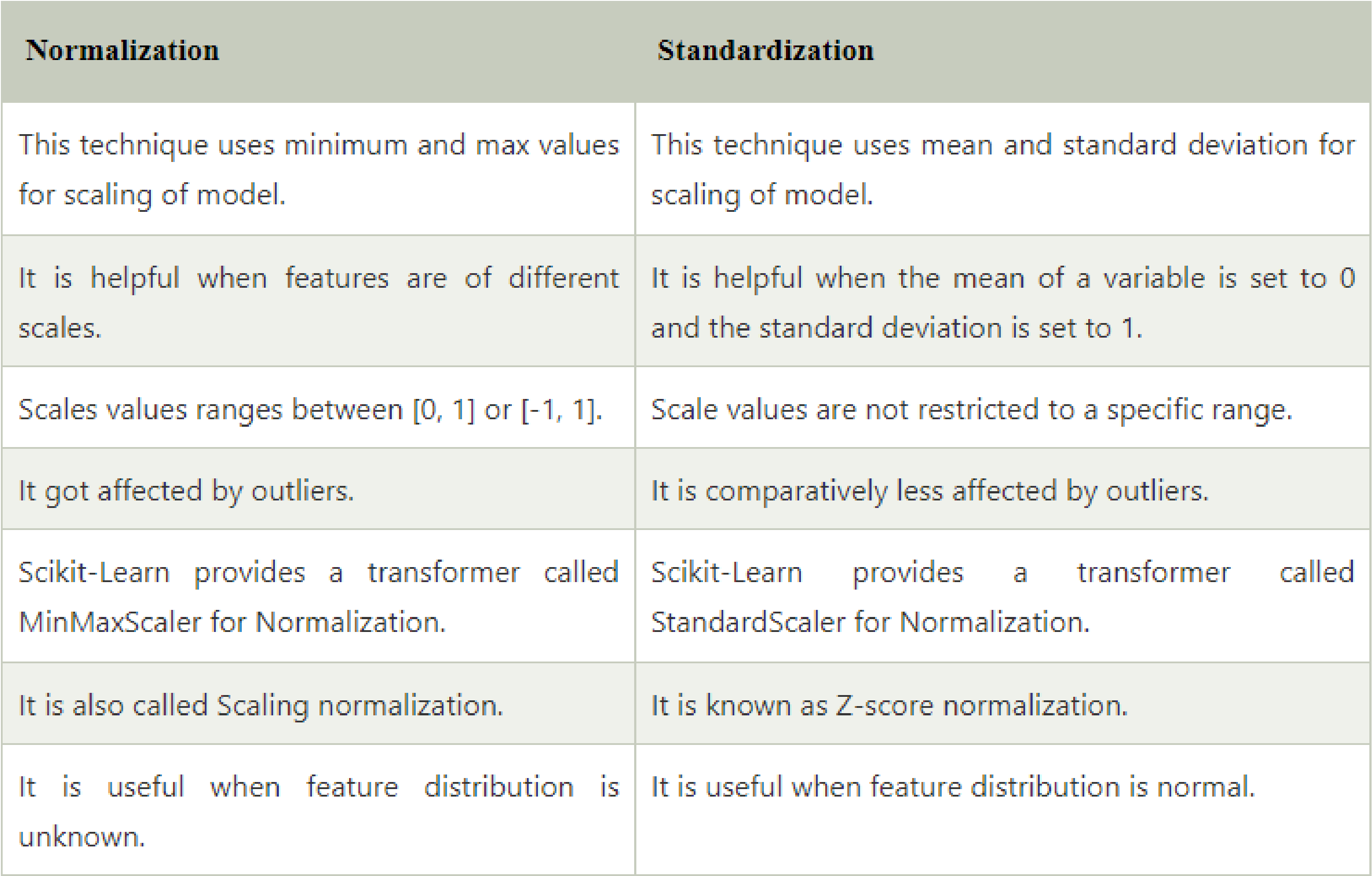


Z-Score Normalization

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$$



Normalization Vs Standardization



Normalization	Standardization
This technique uses minimum and max values for scaling of model.	This technique uses mean and standard deviation for scaling of model.
It is helpful when features are of different scales.	It is helpful when the mean of a variable is set to 0 and the standard deviation is set to 1.
Scales values ranges between [0, 1] or [-1, 1].	Scale values are not restricted to a specific range.
It got affected by outliers.	It is comparatively less affected by outliers.
Scikit-Learn provides a transformer called MinMaxScaler for Normalization.	Scikit-Learn provides a transformer called StandardScaler for Normalization.
It is also called Scaling normalization.	It is known as Z-score normalization.
It is useful when feature distribution is unknown.	It is useful when feature distribution is normal.

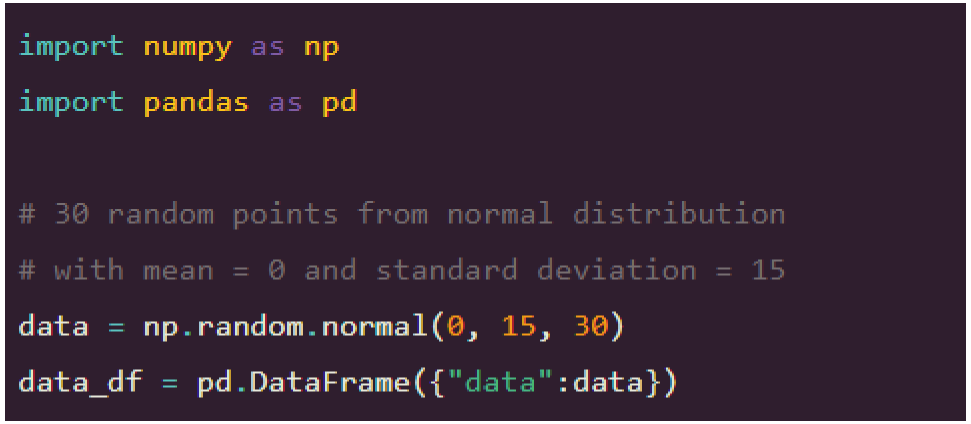
Robust Scaler

Robust scaler changes the median to 0 and IQR to 1.

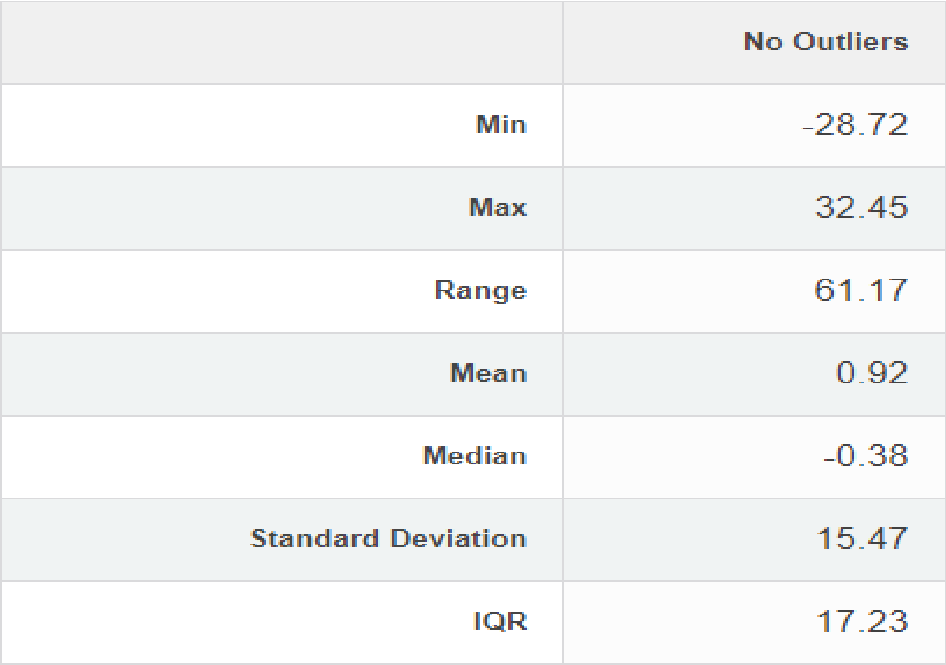
$$X_{\text{scale}} = \frac{x_i - x_{\text{med}}}{x_{75} - x_{25}}$$

extreme values (outliers) can overshadow other data points for a given feature. That can negatively influence standard scaling.

Robust Scaler



```
import numpy as np
import pandas as pd

# 30 random points from normal distribution
# with mean = 0 and standard deviation = 15
data = np.random.normal(0, 15, 30)
data_df = pd.DataFrame({"data":data})
```



	No Outliers
Min	-28.72
Max	32.45
Range	61.17
Mean	0.92
Median	-0.38
Standard Deviation	15.47
IQR	17.23

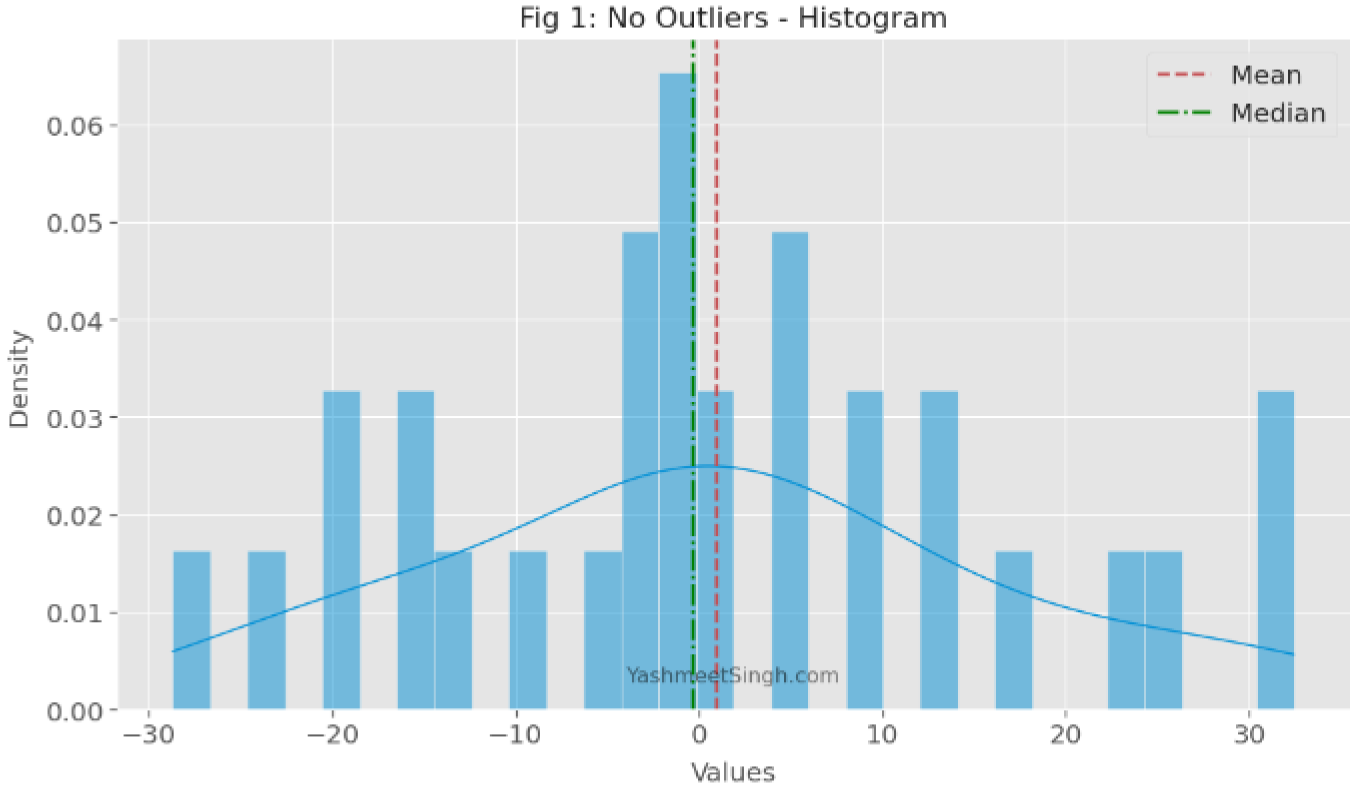
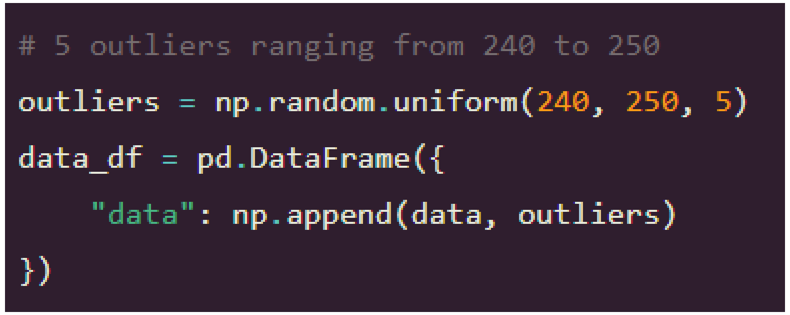
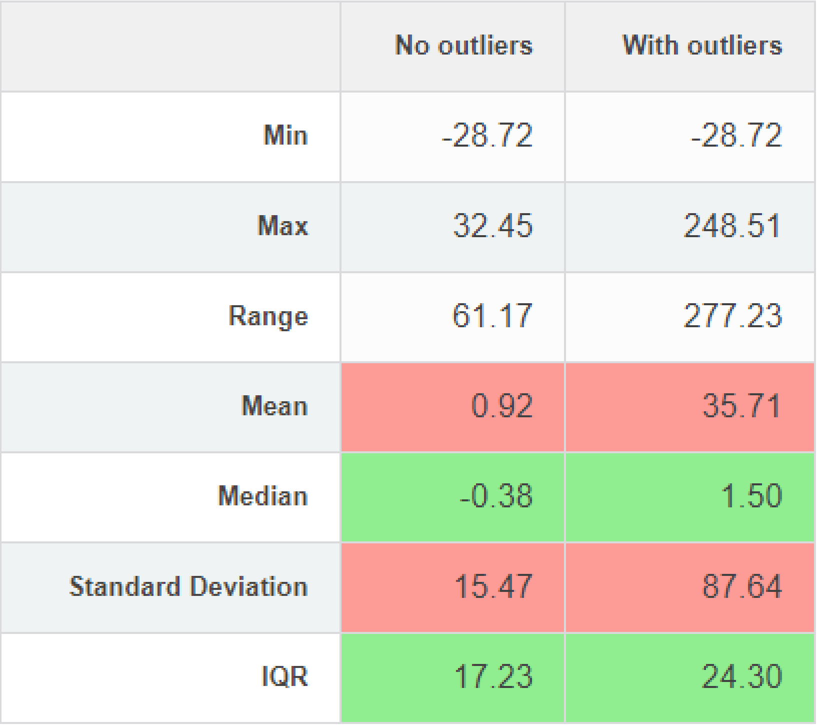


Fig 1: No Outliers - Histogram

Robust Scaler



```
# 5 outliers ranging from 240 to 250
outliers = np.random.uniform(240, 250, 5)
data_df = pd.DataFrame({
    "data": np.append(data, outliers)
})
```



	No outliers	With outliers
Min	-28.72	-28.72
Max	32.45	248.51
Range	61.17	277.23
Mean	0.92	35.71
Median	-0.38	1.50
Standard Deviation	15.47	87.64
IQR	17.23	24.30

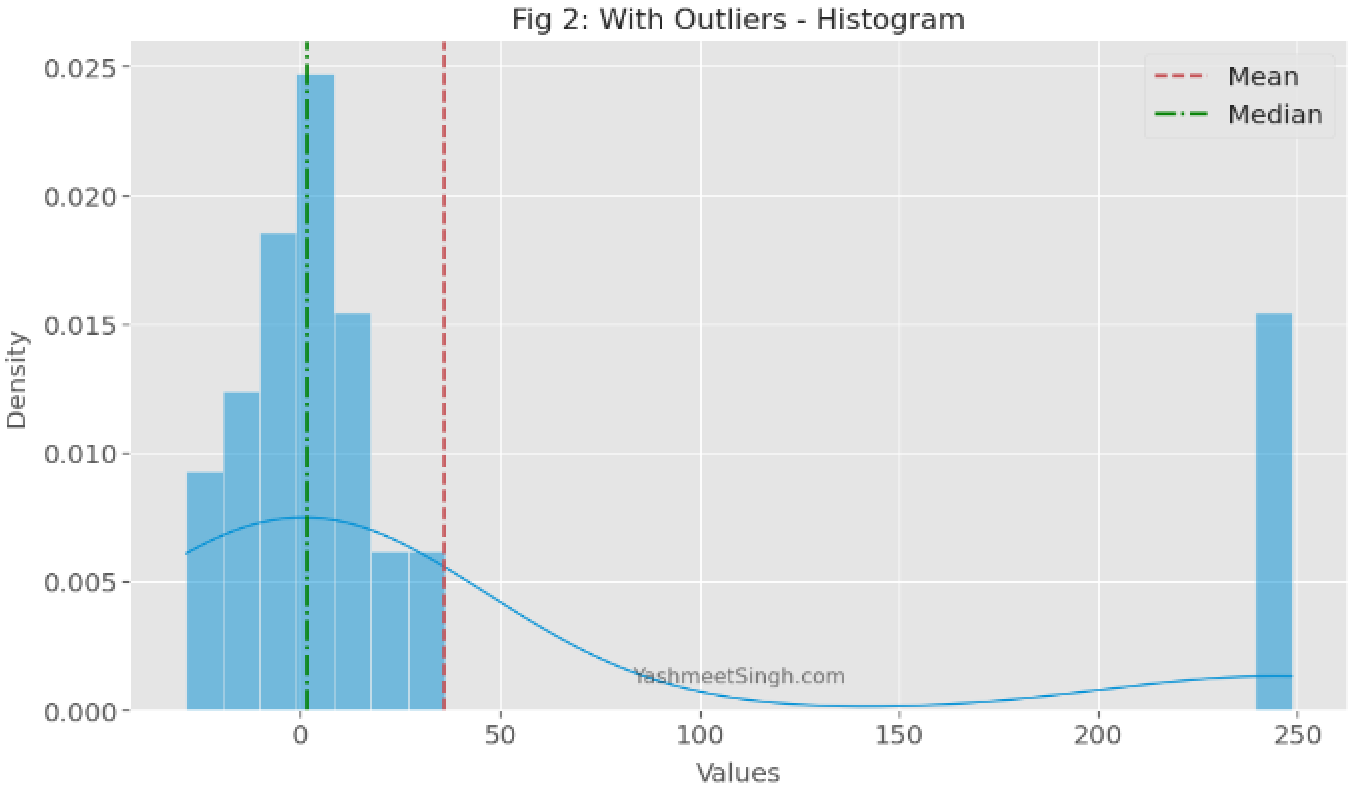
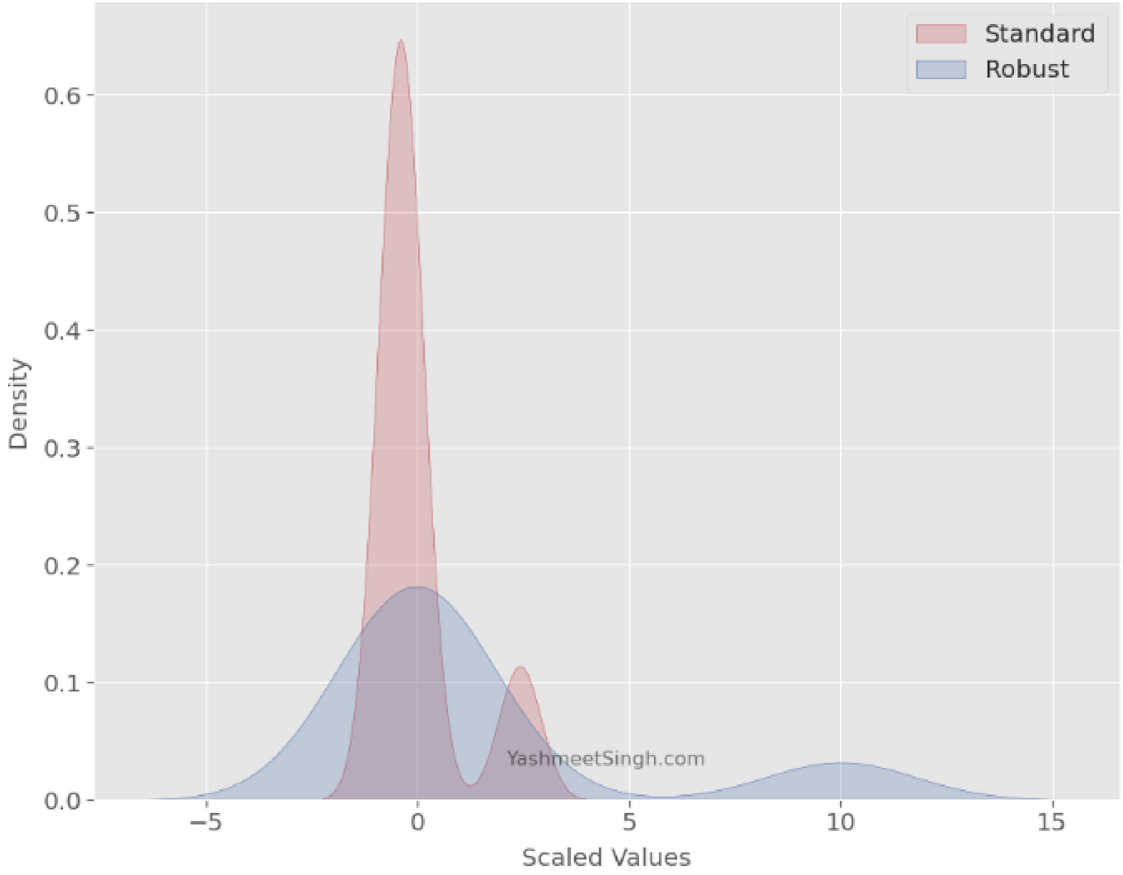


Fig 2: With Outliers - Histogram

Robust Scaler VS Standard Scaler



The **robust** scaler produces a much wider range of values than the **standard** scaler.

Robust scaler resists the pull of outliers.

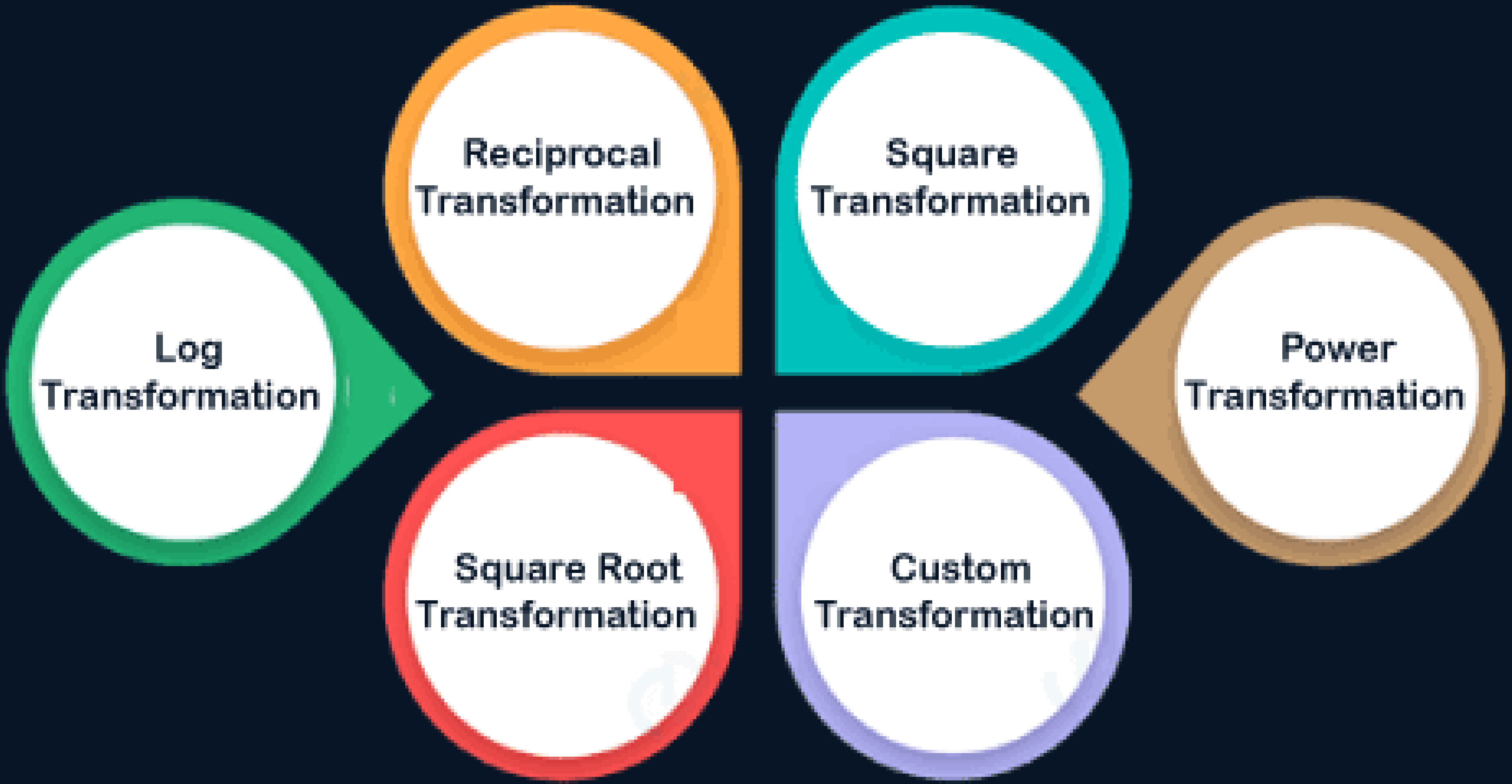
MaxAbs Scaler

$$X_i' = \frac{X_i}{abs(X_{max})}$$

- All values in the feature are divided by the absolute maximum value of that feature.
- If we want to keep 0 values as 0, we can use this method.
- If only positive values are present, the range is [0, 1].
- If only negative values are present, the range is [-1, 0].
- If both negative and positive values are present, the range is [-1, 1].
- On positive only data, both **MinMaxScaler** and **MaxAbsScaler** behave similarly.
- **MaxAbsScaler** also suffers from the presence of large outliers.

- EX7 Feature Scaling

Feature Transformation Techniques



Reciprocal Transformation

Square Transformation

Log Transformation

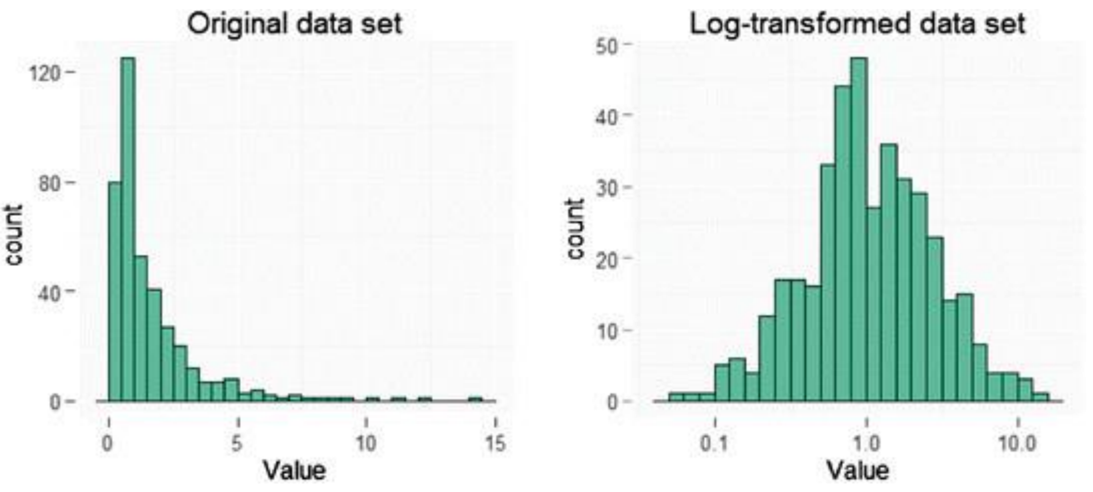
Power Transformation

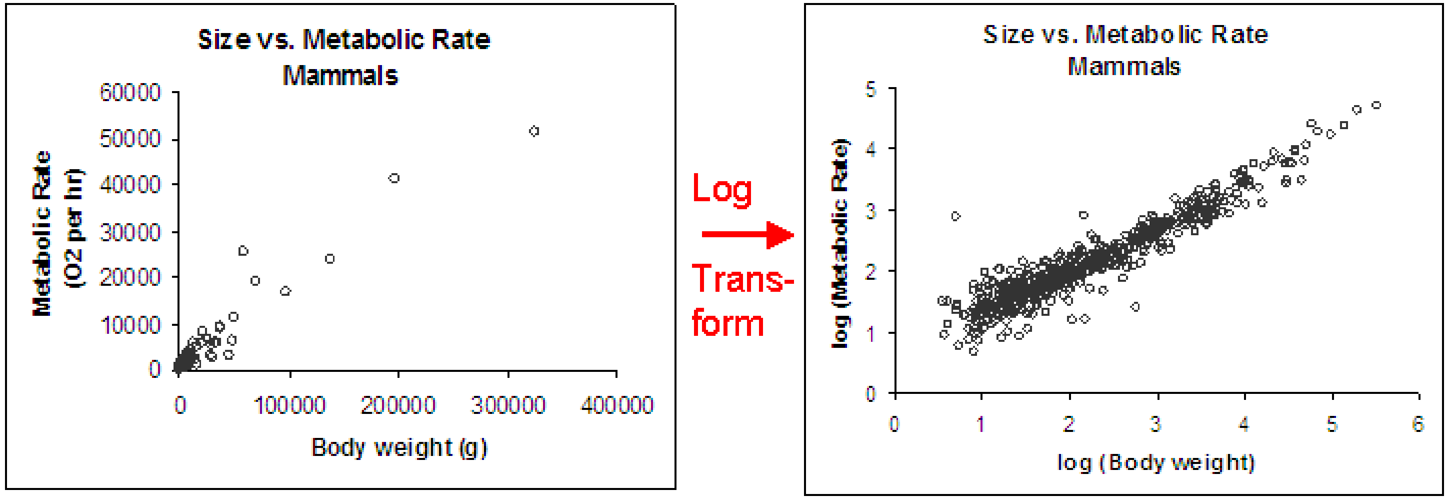
Square Root Transformation

Custom Transformation

Log Transformation

- It helps in handling skewed data, and after transformation, the distribution becomes more similar to normal.
- The effect of outliers is decreased and the model becomes robust.





Reciprocal Transformation

- It cannot be applied to the value of zero.
- It reverses the order between data with the same sign, i.e., larger values become small and vice versa (or like rewinding the time).
- It can be applied to right-skewed data.

Square Transformation

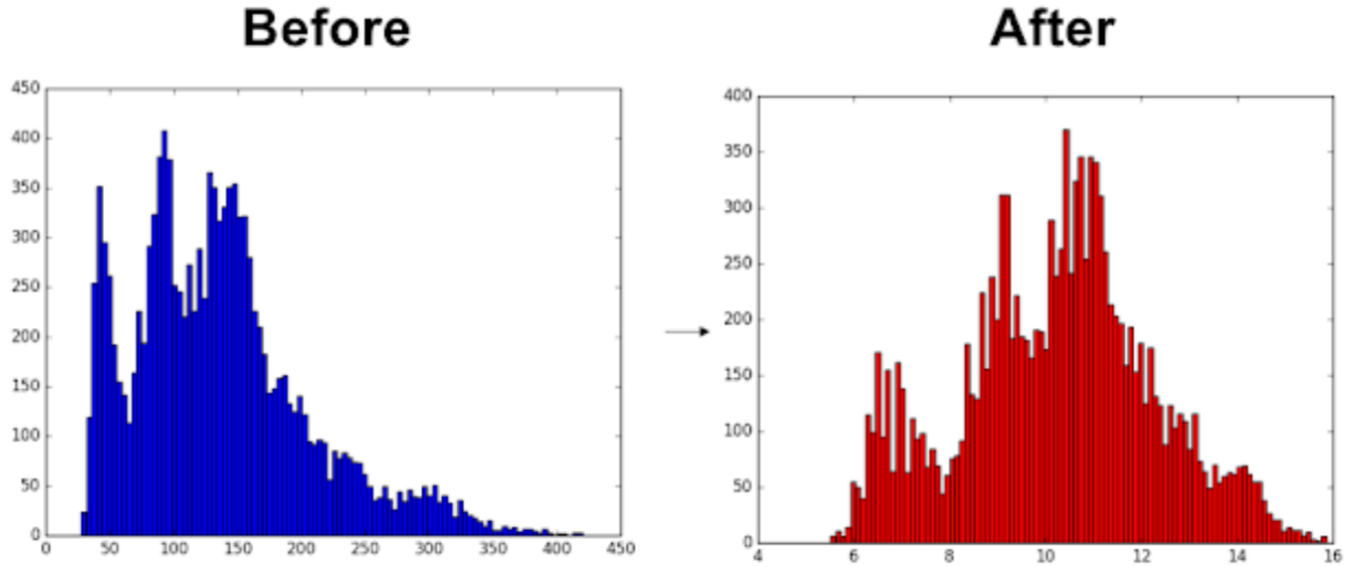
- It is generally applied to **left-skewed** data.

Square Root Transformation

- It is only valid for **positive** numbers and is generally applied to **right-skewed** data.

Power Transformer

applies a power transformation to each feature to make the data more Gaussian-like in order to stabilize variance and minimize skewness.

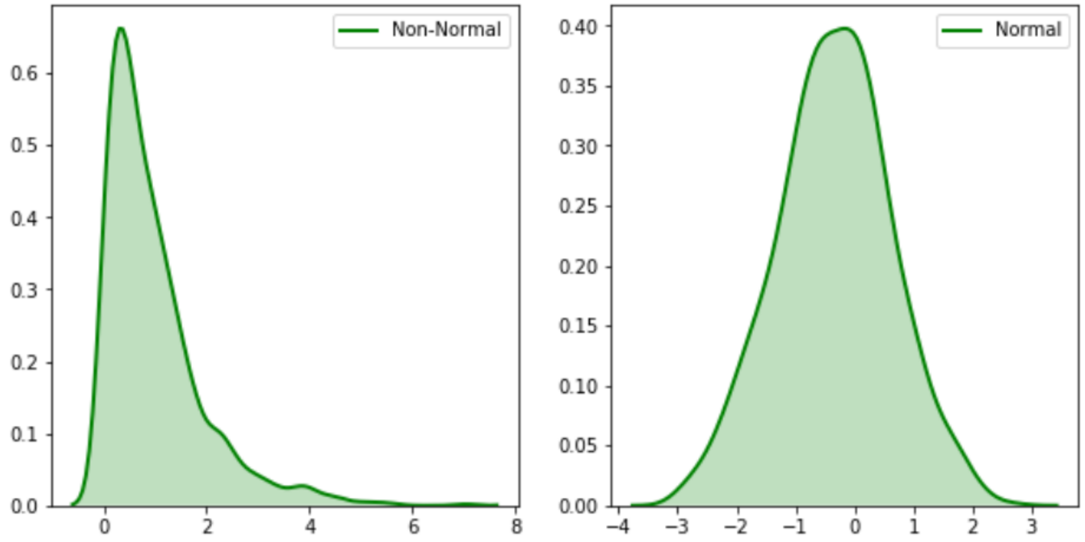


Box-Cox Transformation

- This is included in the concept of power transformations.
- Data must be positive.
- Lambda, λ is a value between -5 and 5. An optimal lambda value should be selected (like hyperparameter tuning).

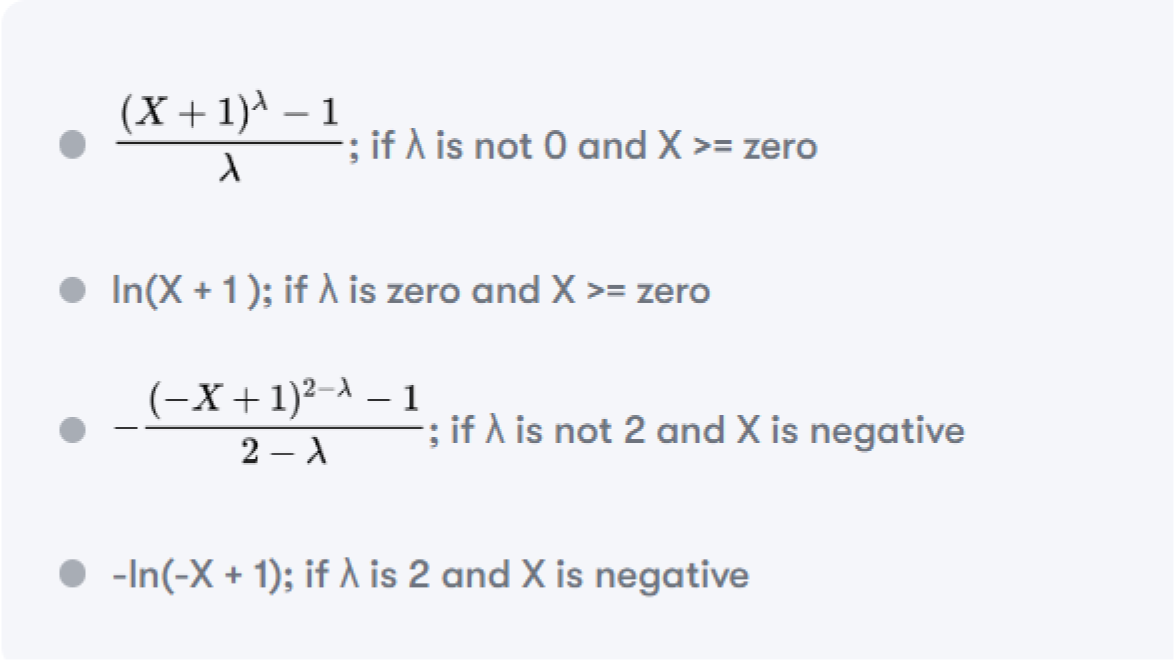
$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^{(\lambda)} - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(y_i) & \text{if } \lambda = 0 \end{cases}$$

```
Lambda value used for Transformation: 0.30656155175590766
```



Yeo-Johnson Transformation

- The power transformation technique if the data contains zero or negative values.

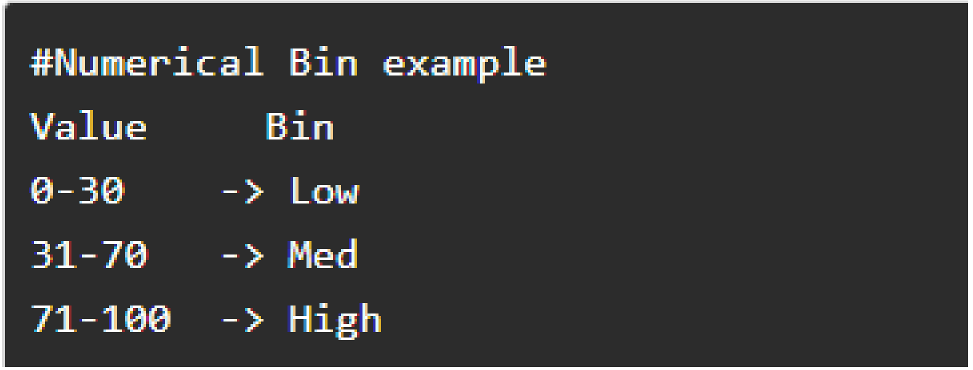


- $\frac{(X+1)^{\lambda}-1}{\lambda}$; if λ is not 0 and X >= zero
- ln(X + 1); if λ is zero and X >= zero
- $-\frac{(-X+1)^{2-\lambda}-1}{2-\lambda}$; if λ is not 2 and X is negative
- -ln(-X + 1); if λ is 2 and X is negative

- EX Log Transformation

Binning

- Data binning, also called discrete binning or bucketing, is a data pre-processing technique used to reduce the effects of minor observation errors.
- It is a form of quantization.
- Binning or discretization is used to transform a continuous or numerical variable into a categorical feature.



```
#Numerical Bin example
Value     Bin
0-30    -> Low
31-70   -> Med
71-100  -> High
```

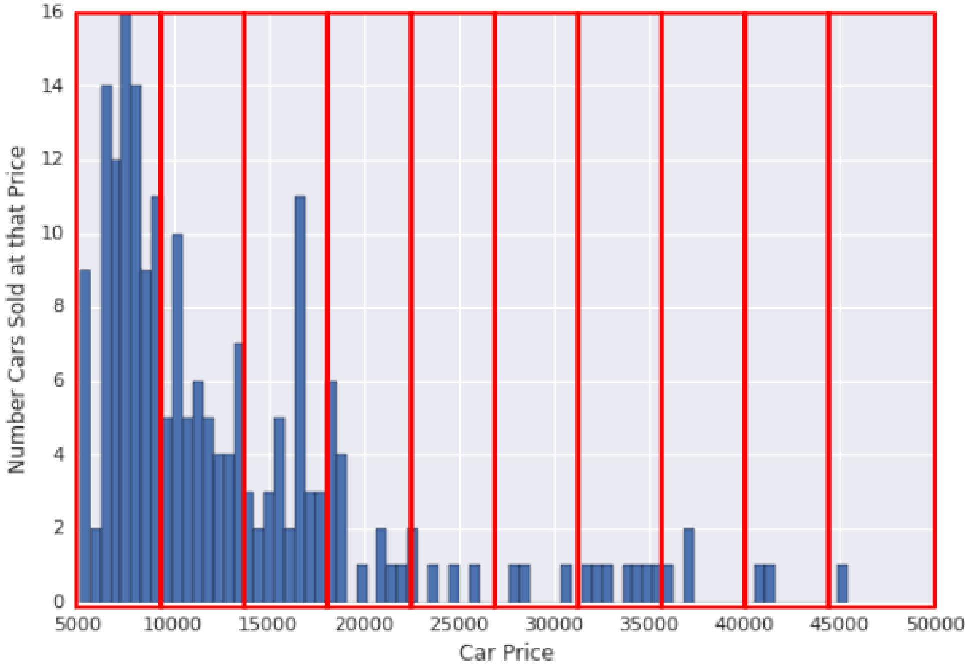
Why is Binning Used?

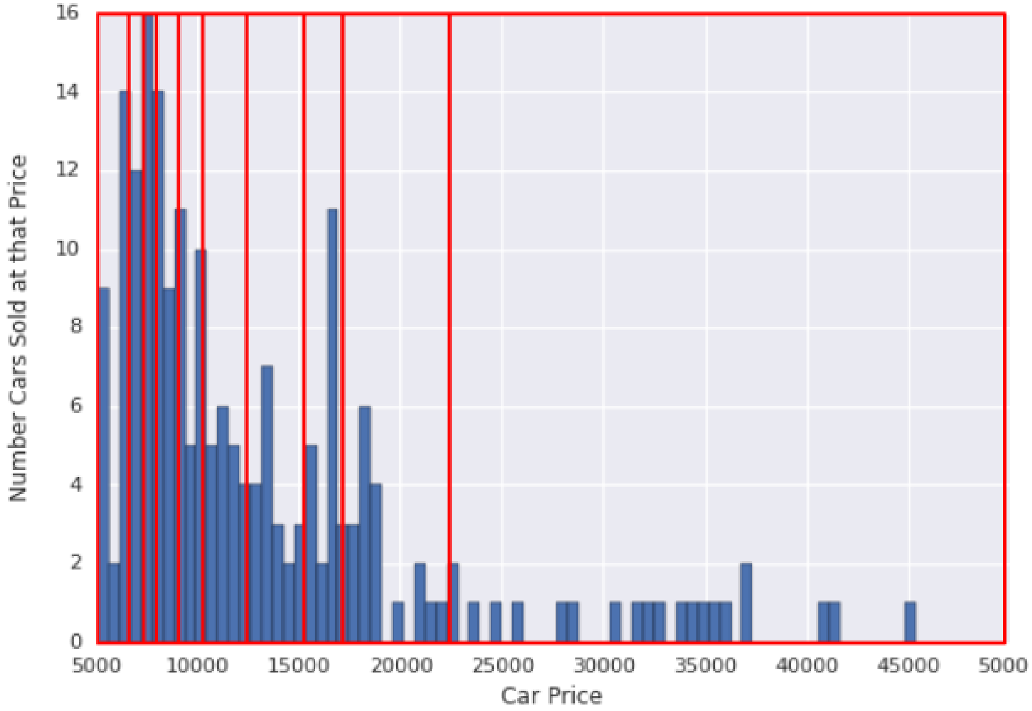
- Binning groups related values together in bins to reduce the number of distinct values.
- This has a soothing effect on the input data and may also reduce the chances of overfitting in the case of small datasets and tends to improve the performance of the model.

How do you Binning Data?

1. Equal Frequency Binning: Divide the data into several equal parts, and the number of data in each equal part is the same.

2. Equal Width Binning: Divide the data into equal parts with the same width.

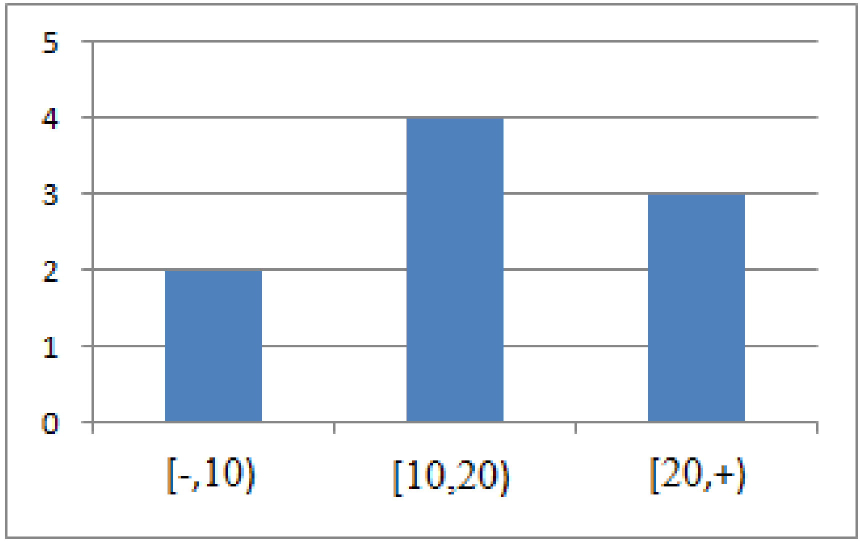




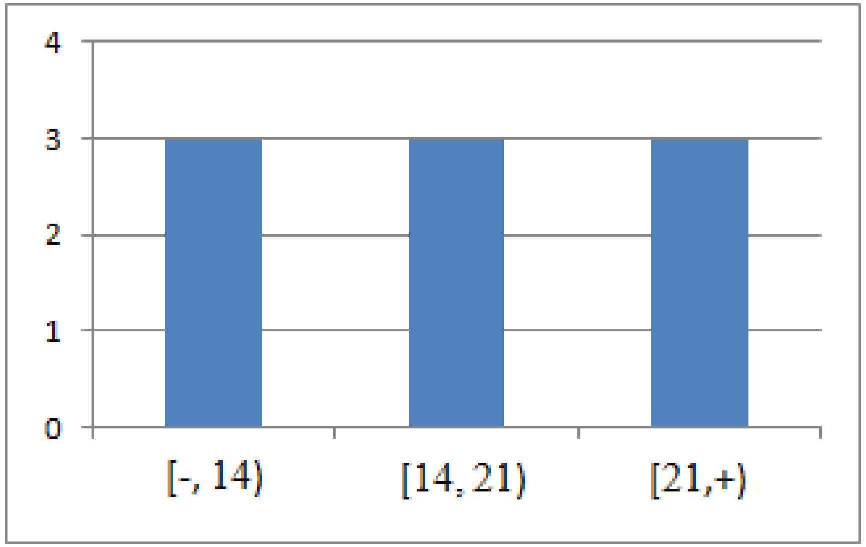
Exercise

0, 4, 12, 16, 16, 18, 24, 26, 28

How do you Binning Data?

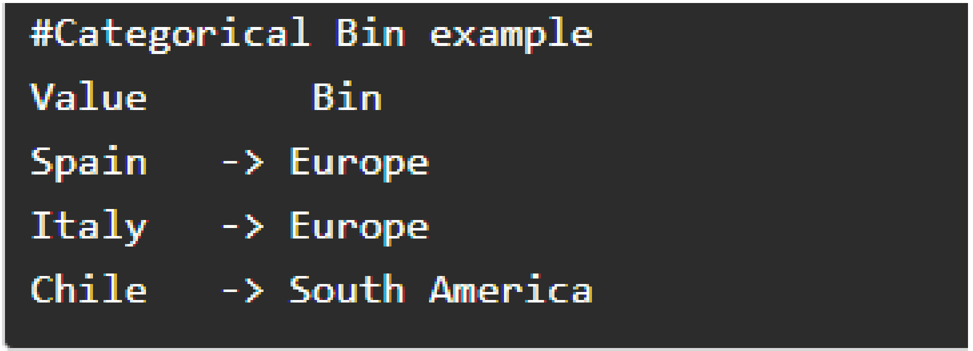


Equal width



Equal frequency

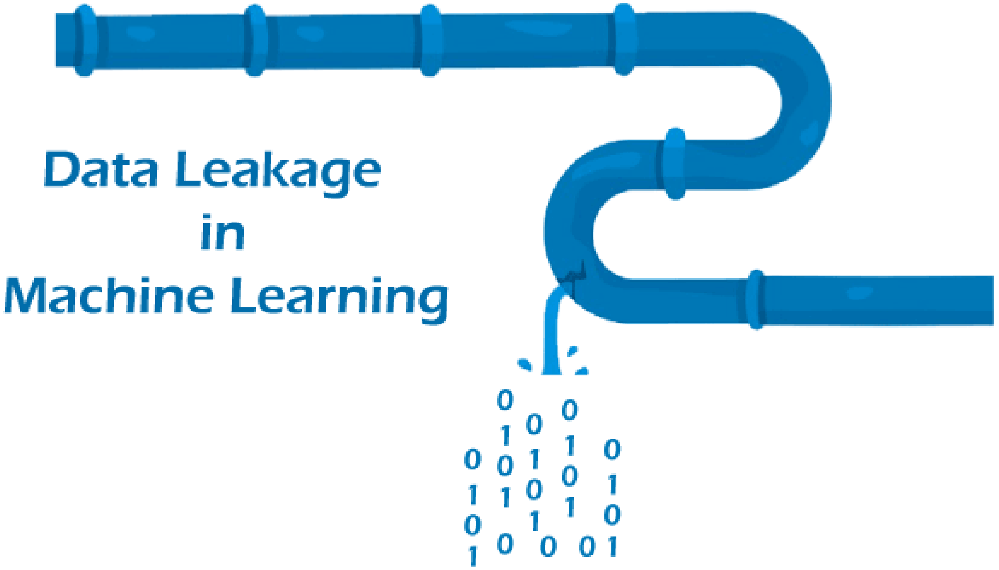
Binning



```
#Categorical Bin example
Value       Bin
Spain   -> Europe
Italy   -> Europe
Chile   -> South America
```

Data Leakage

- **Data leakage** (or **leakage**) happens when your training data contains information about the target, but similar data will not be available when the model is used for prediction.



This leads to high performance on the training set (and possibly even the validation data), but the model will perform poorly in production.

How to detect data leakage?

- **Case -1:** Firstly, if we find that our model is performing very well, i.e., predicted outcome and actual outcomes are the same, then we must get suspicious of the occurrence of data leakage problem. In such cases, the model memorizes the relations between training data and test data instead of generalizing on the unseen data.

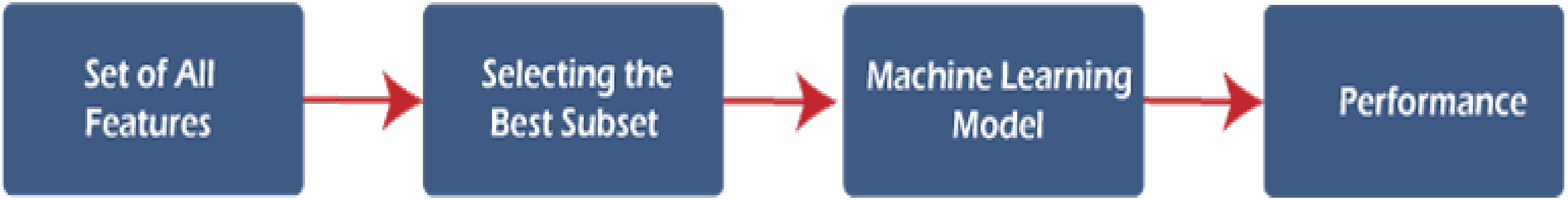
- **Case-2:** While performing Exploratory Data Analysis EDA, we may find some highly correlated features with the target variable. Although some features can be more correlated than others, if there is an exceptionally high correlation between them, it should be checked to avoid data leakage.

- **Case-3:** After building the model, check if there is some unusual feature behavior in the fitted model. For example, exceptionally high feature weights or extensive information associated with a variable. Further, we should check for the model's overall performance, i.e., if it is unexpected. It means we need to look carefully at the events and features that have a high impact on the model if the results of the model's evaluation are significantly greater than those of similar or comparable situations and datasets.

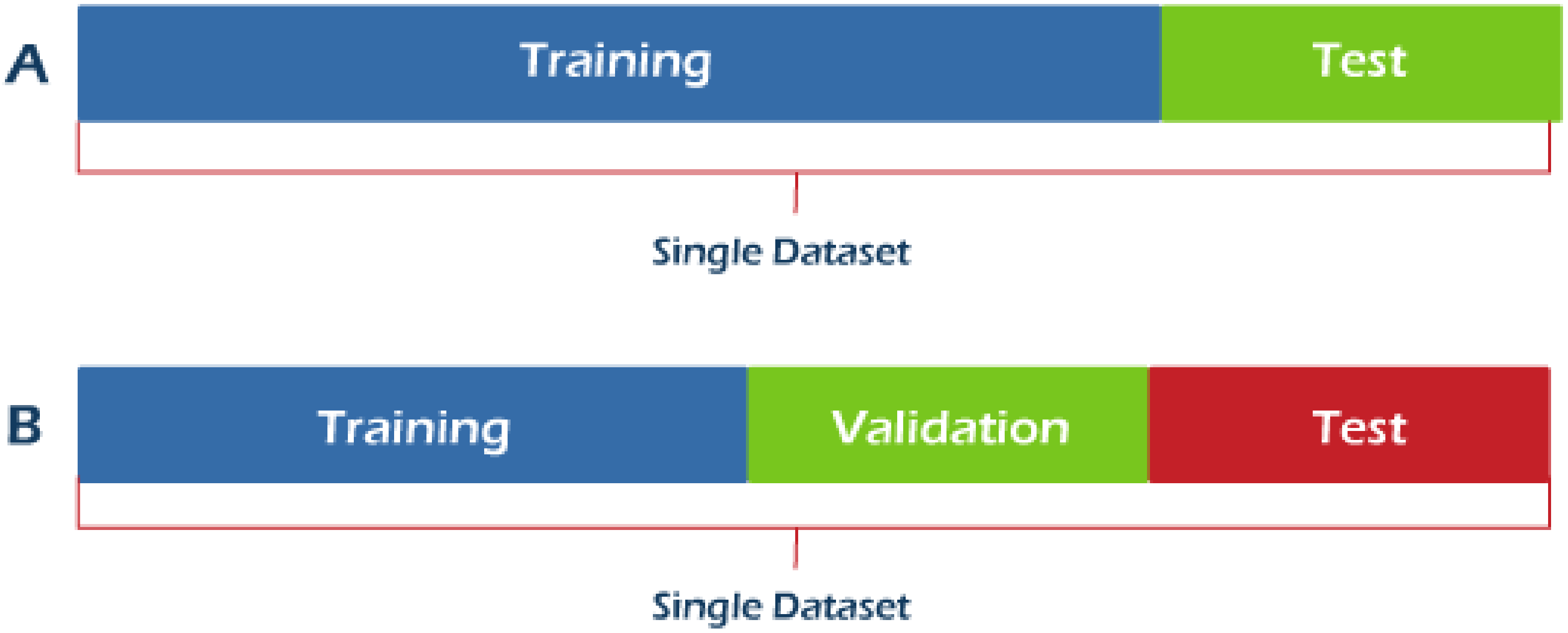
How to Fix Data Leakage Problem in Machine Learning?

- Extract the appropriate set of features
- Add an individual validation dataset.
- Apply data pre-processing separately to both data sets
- Time-series data
- Cross-Validation

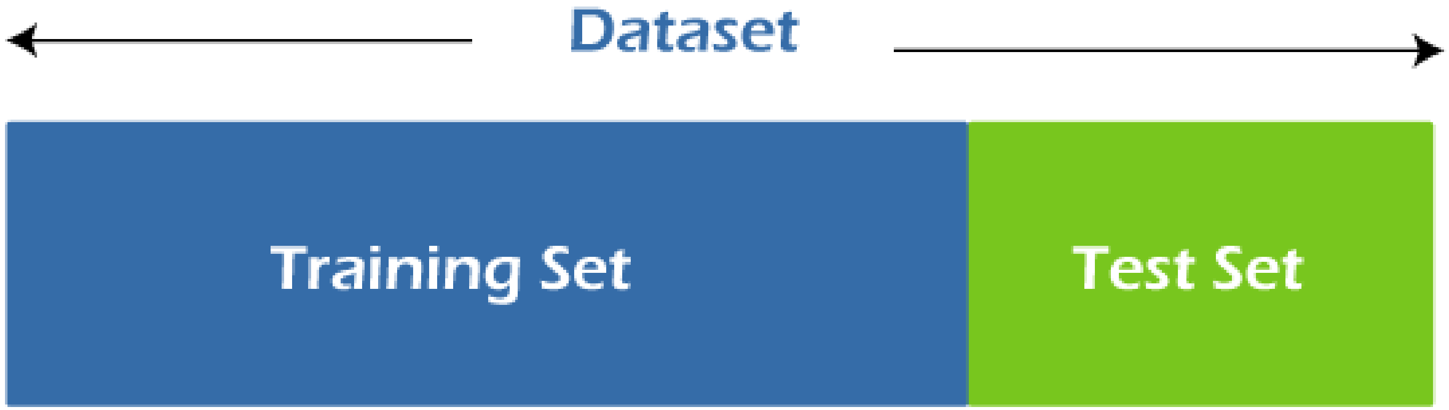
1. Extract the appropriate set of features



2. Add an individual validation set



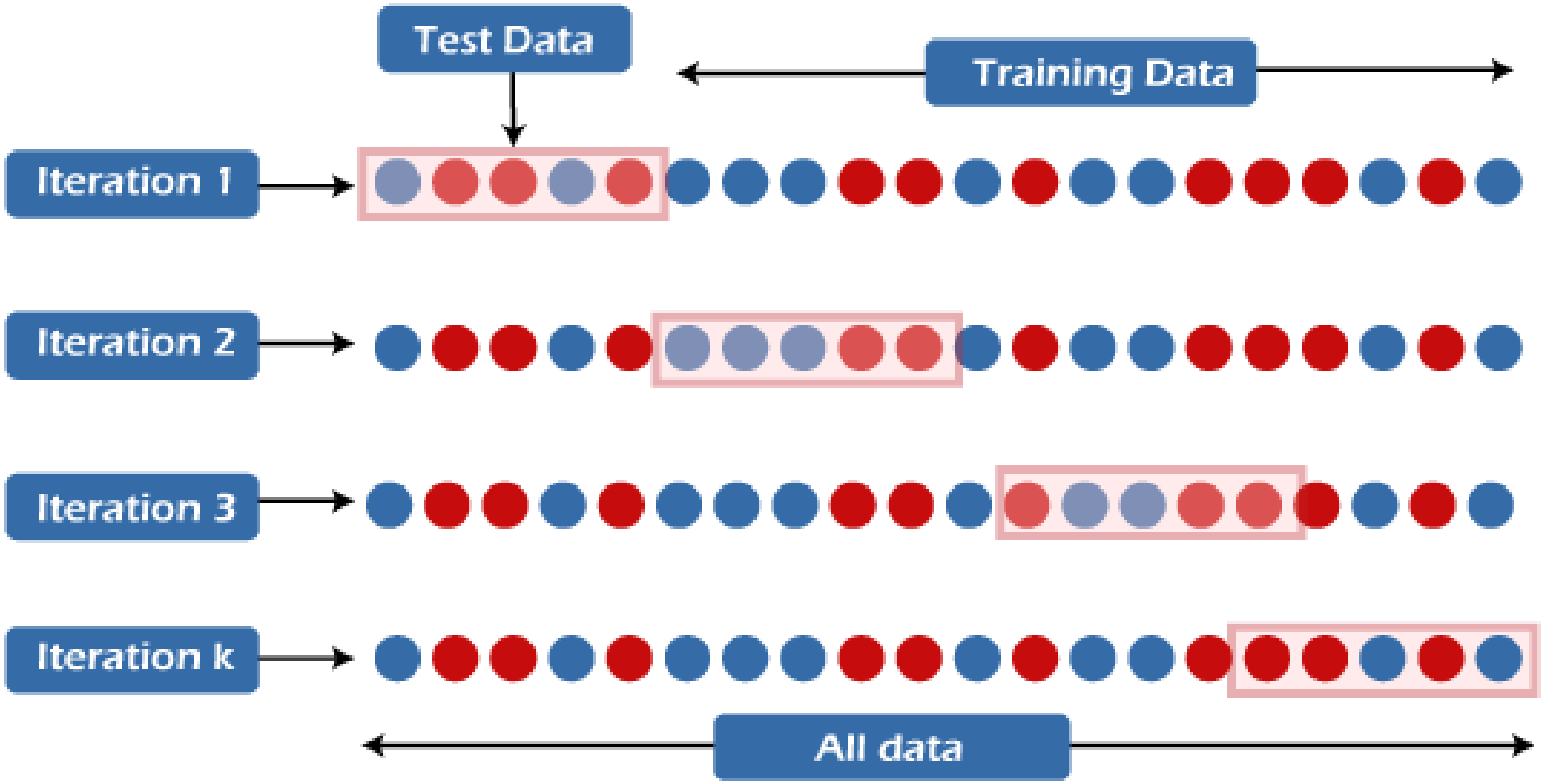
3. Apply data pre-processing separately to both datasets



4. Time-series data

- When working with time-series data, we should have more attention to data leakage issues in the models. Data leakage in time-series data is due to randomly splitting into test and training data sets.
- To minimize the data leakage in time-series data, always put a cut-off value on time as it helps you avoid getting any information after the time of prediction.

5. Cross-Validation



Dimensionality Reduction

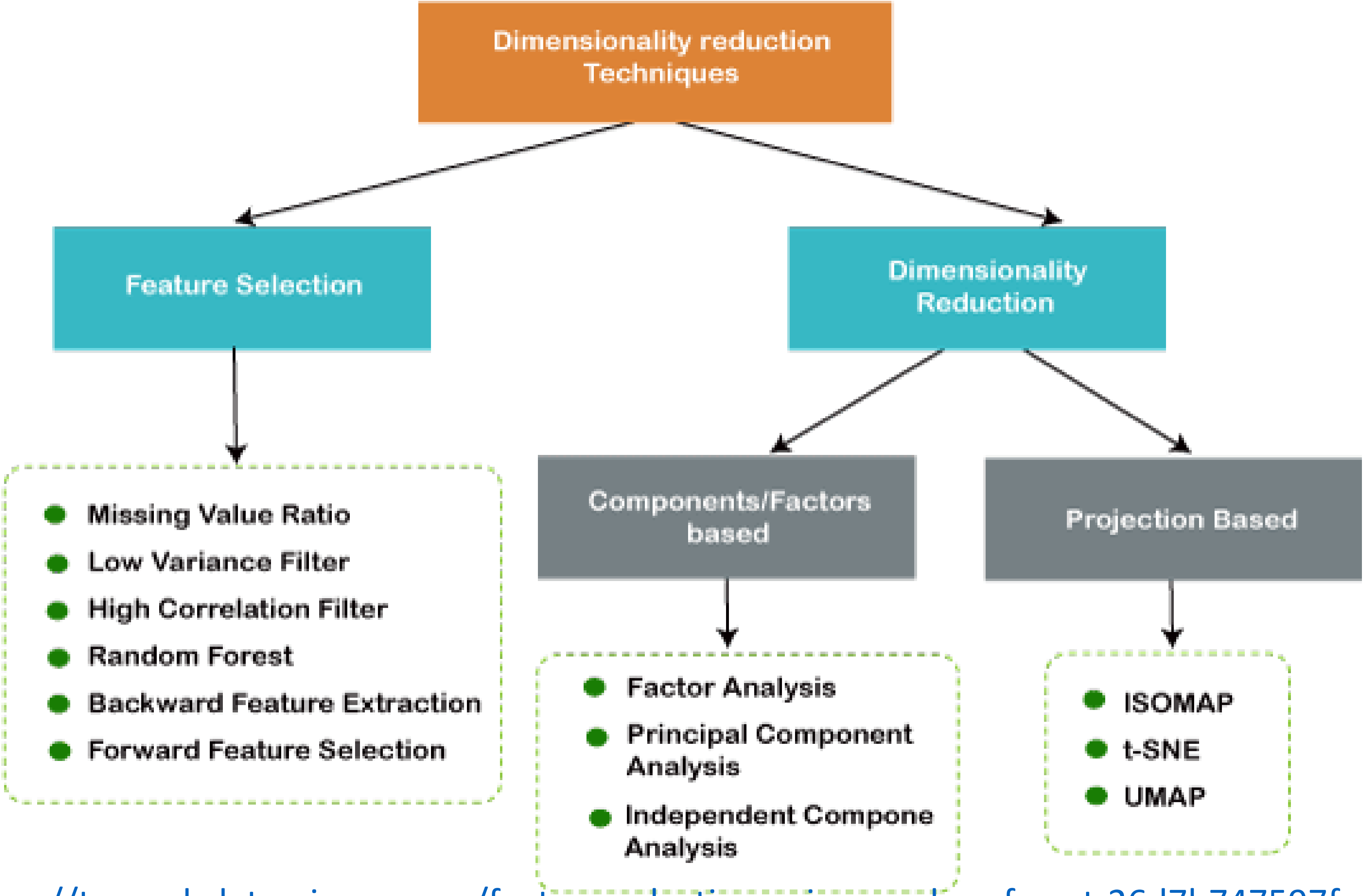
- Dimensionality reduction is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data.

The Curse of Dimensionality

Benefits of applying Dimensionality Reduction

- By reducing the dimensions of the features, the space required to store the dataset also gets reduced.
- Less Computation training time is required for reduced dimensions of features.
- Reduced dimensions of features of the dataset help in visualizing the data quickly.
- It removes the redundant features (if present).

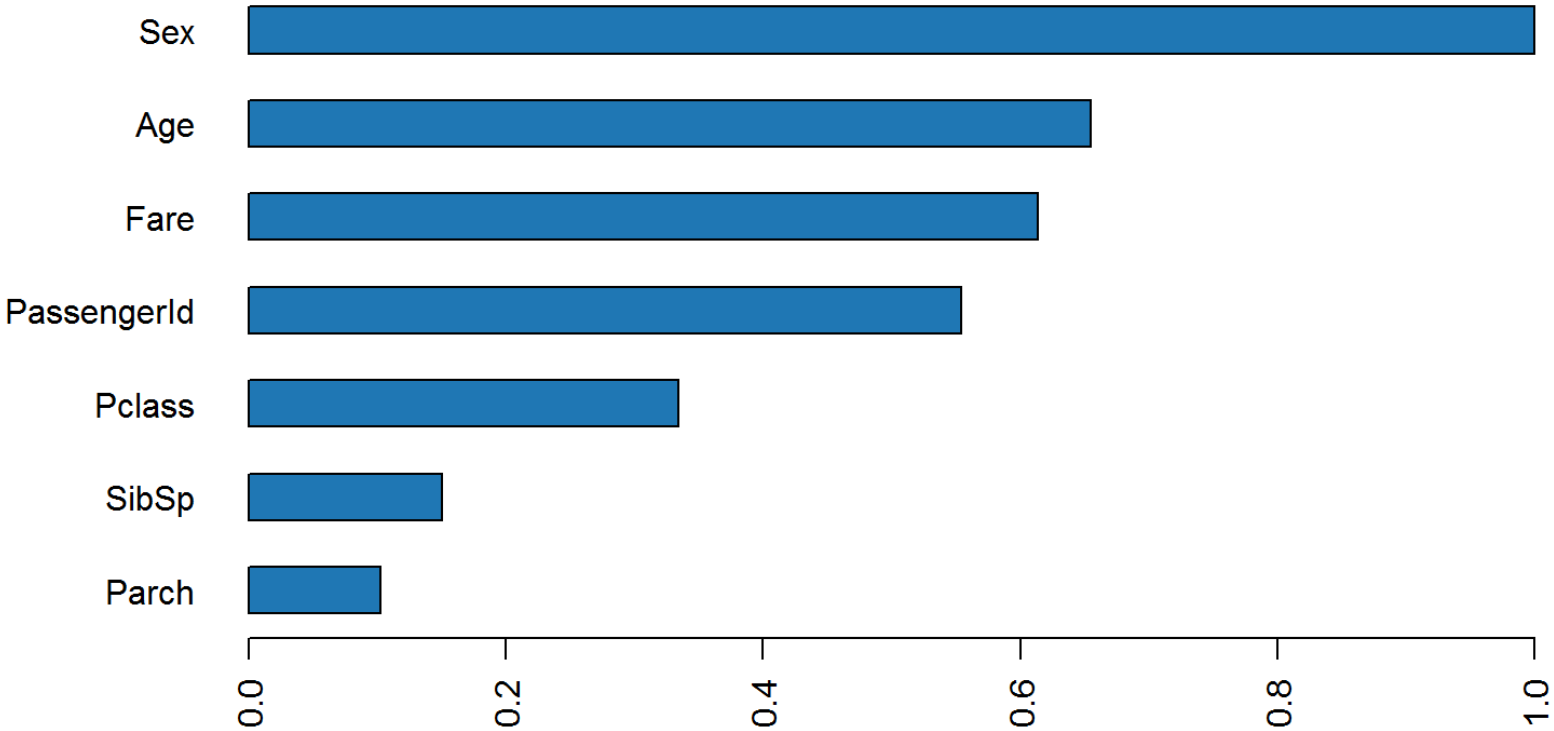
Dimensionality Reduction



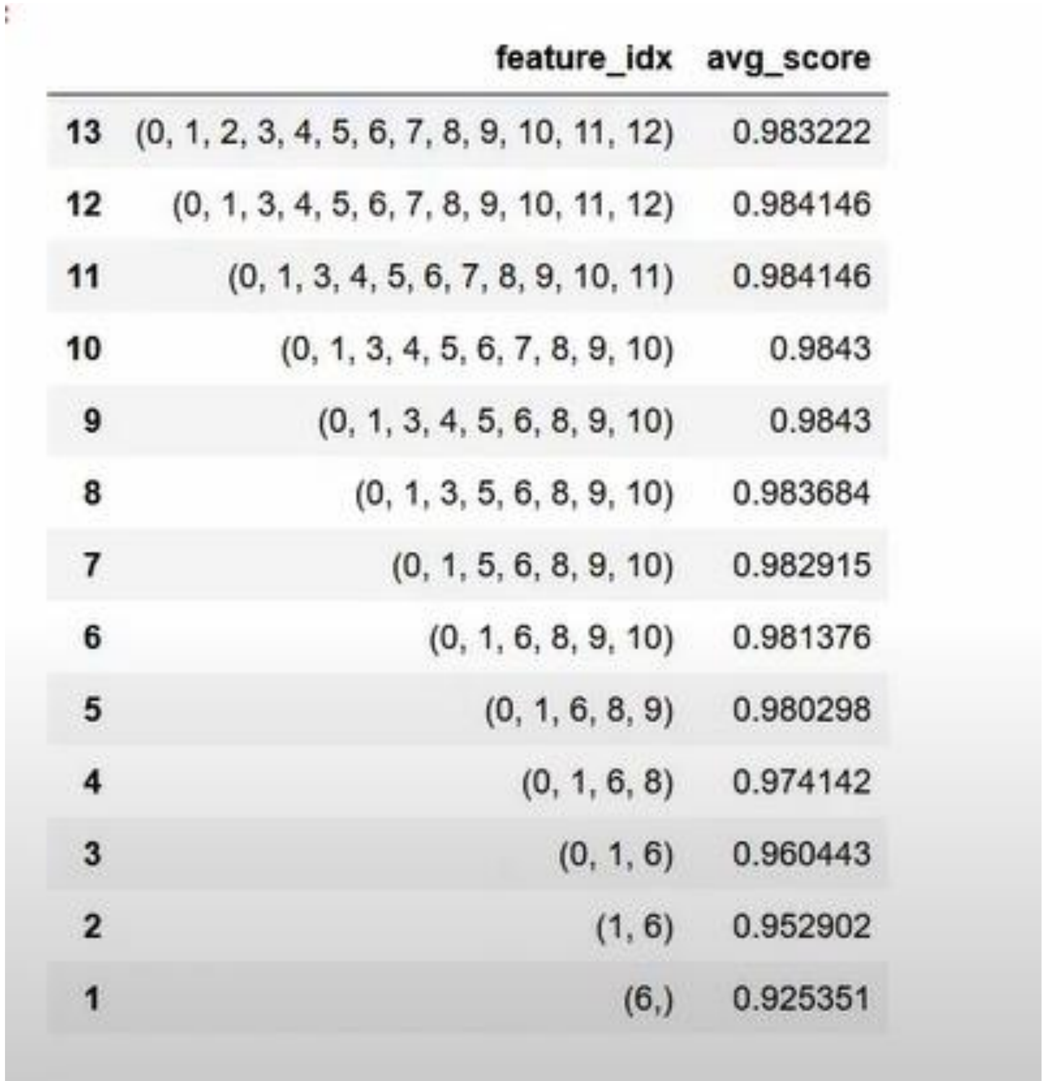
https://towardsdatascience.com/feature-selection-using-random-forest-26d7b747597f

Feature Importance using RF

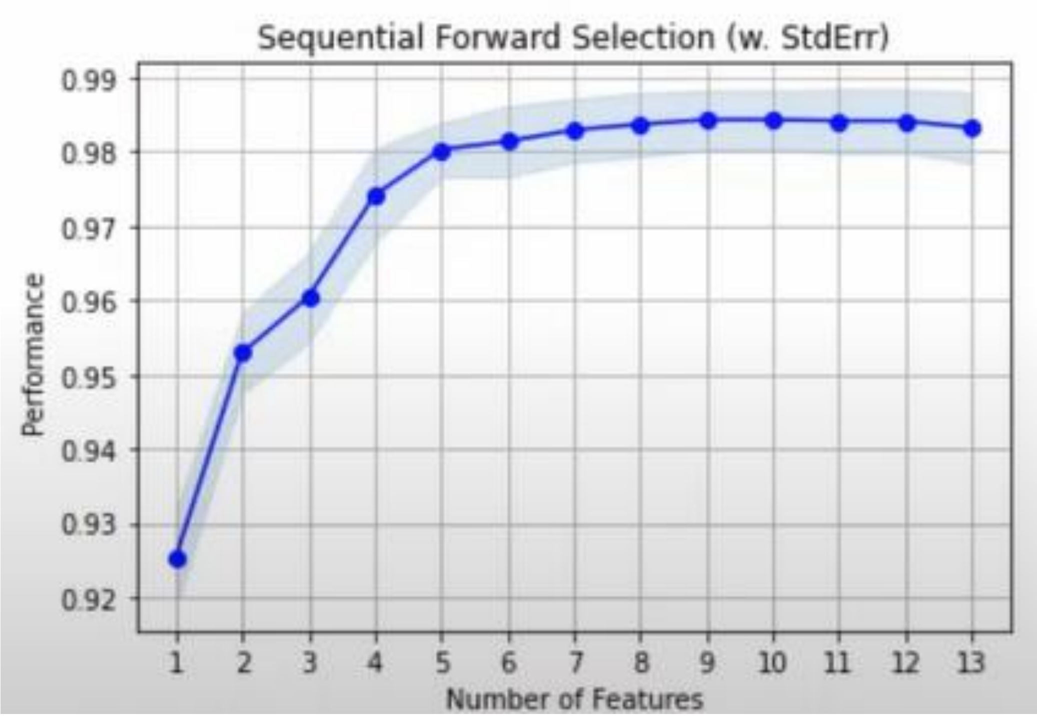
```
from sklearn.ensemble import RandomForestClassifier as RClf

model = RClf(n_estimators = 100)
model.fit(X, y)
importances = model.feature_importances_
```



Backward Feature Selection

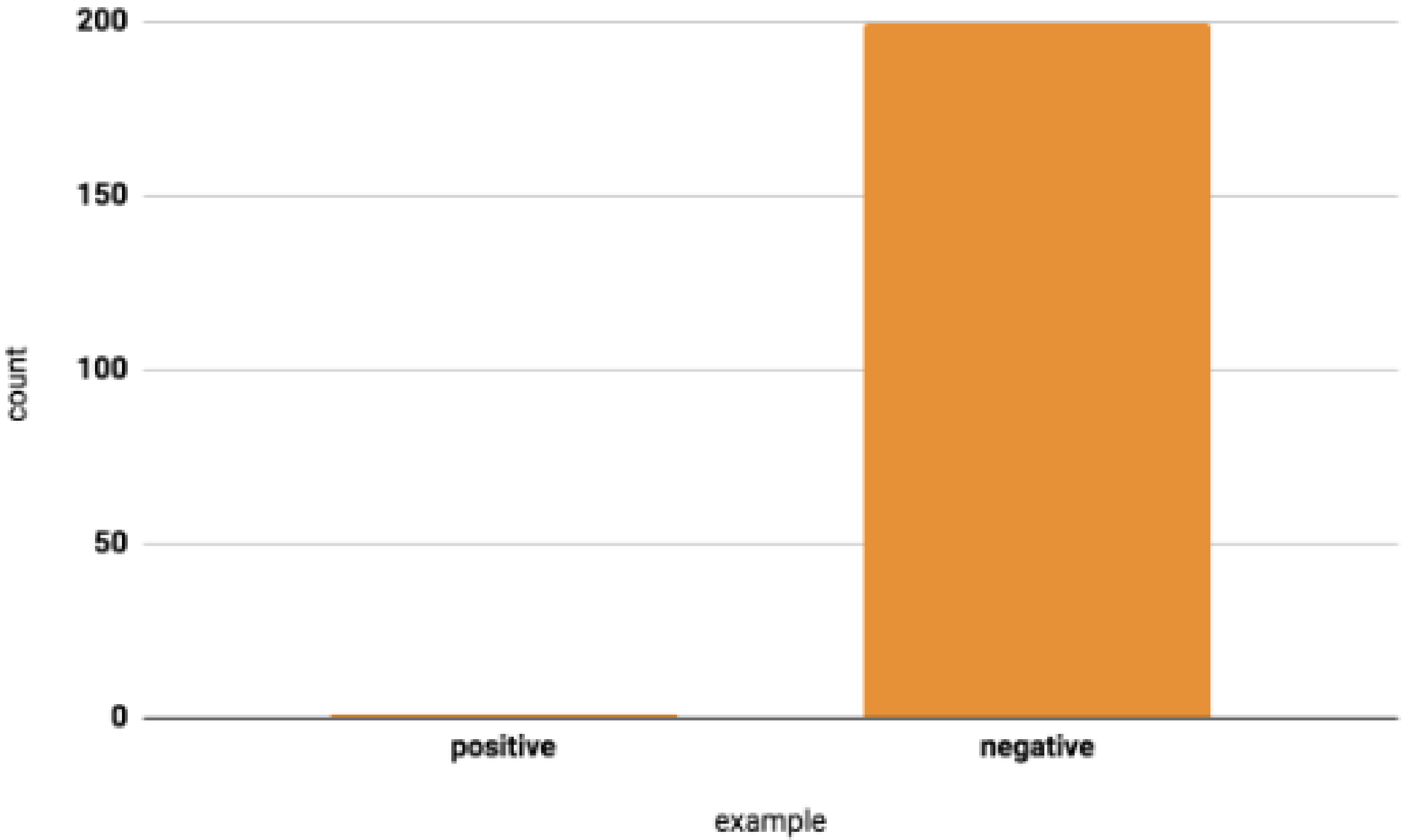


	feature_idx	avg_score
13	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)	0.983222
12	(0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)	0.984146
11	(0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11)	0.984146
10	(0, 1, 3, 4, 5, 6, 7, 8, 9, 10)	0.9843
9	(0, 1, 3, 4, 5, 6, 8, 9, 10)	0.9843
8	(0, 1, 3, 5, 6, 8, 9, 10)	0.983684
7	(0, 1, 5, 6, 8, 9, 10)	0.982915
6	(0, 1, 6, 8, 9, 10)	0.981376
5	(0, 1, 6, 8, 9)	0.980298
4	(0, 1, 6, 8)	0.974142
3	(0, 1, 6)	0.960443
2	(1, 6)	0.952902
1	(6,)	0.925351

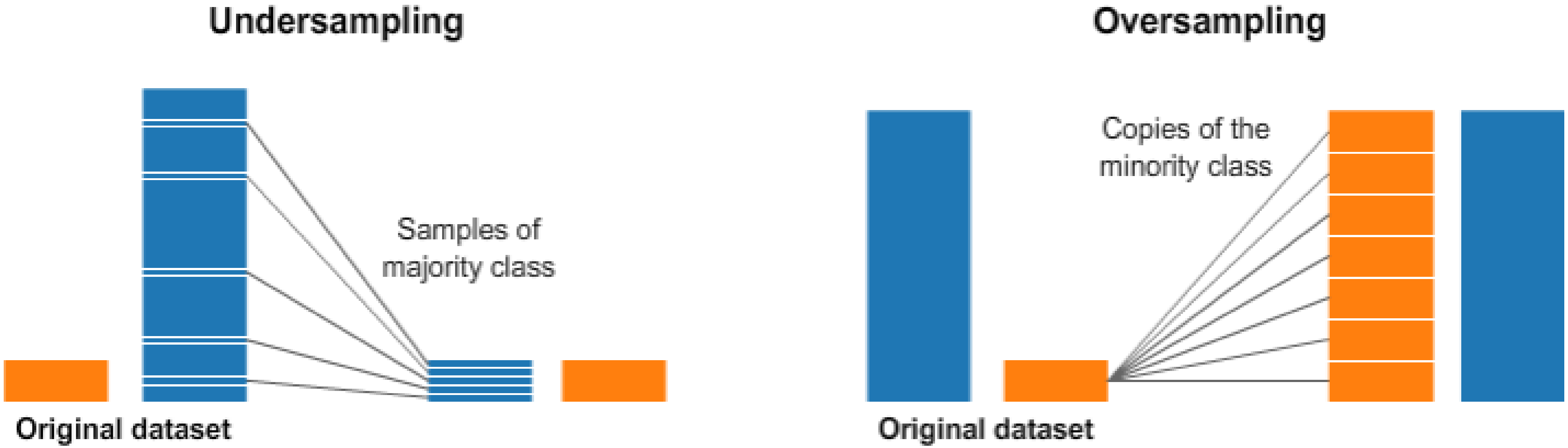


"UNBALANCED"
11/09

Imbalanced Dataset



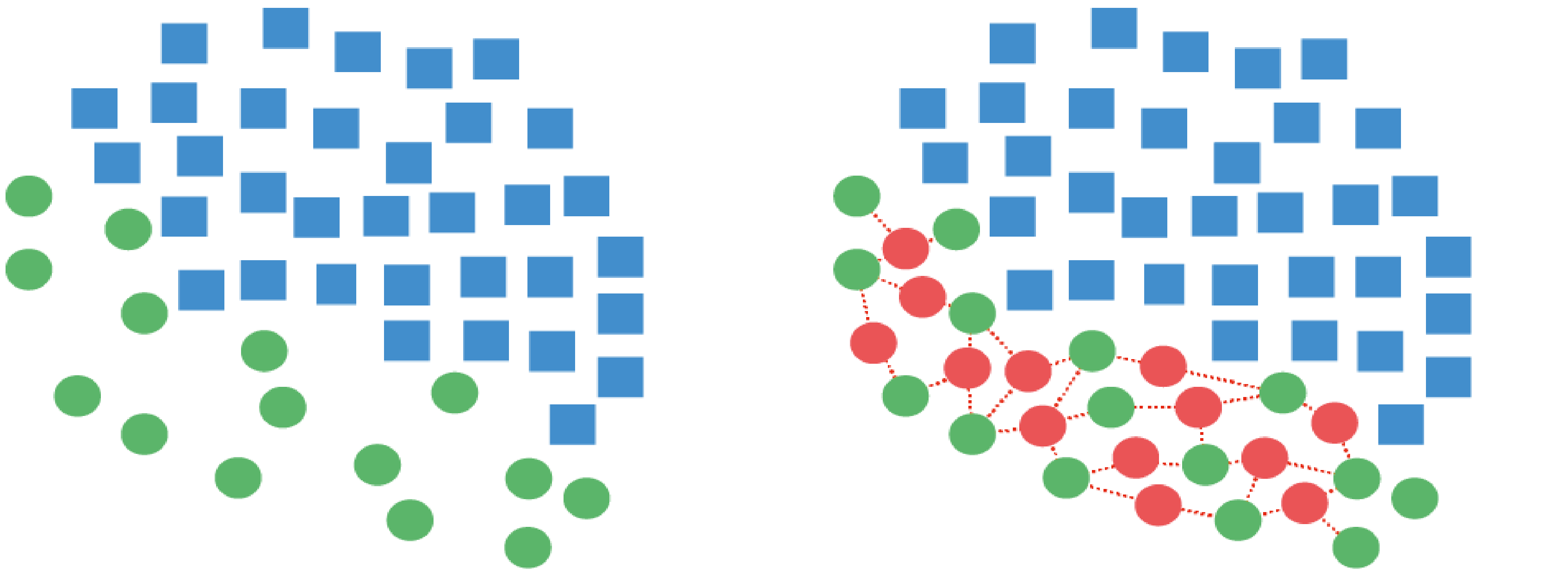
Imbalance Dataset



Accuracy is not the best metric to use when evaluating imbalanced datasets as it can be misleading

Imbalance Dataset

Synthetic Minority Oversampling Technique SMOTE



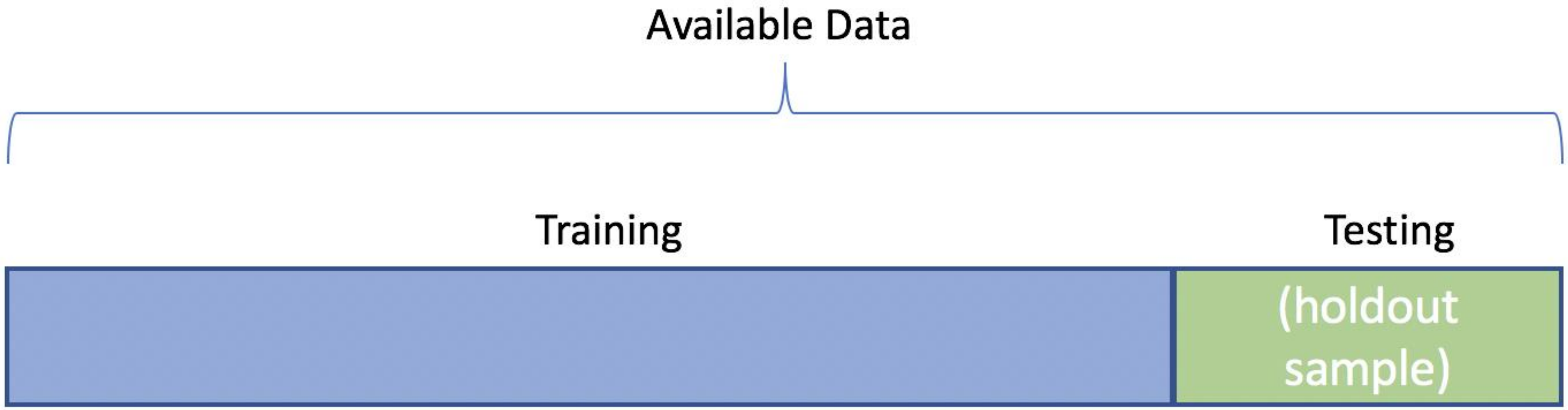
Original Dataset

Generating Samples

SMOTE works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point.

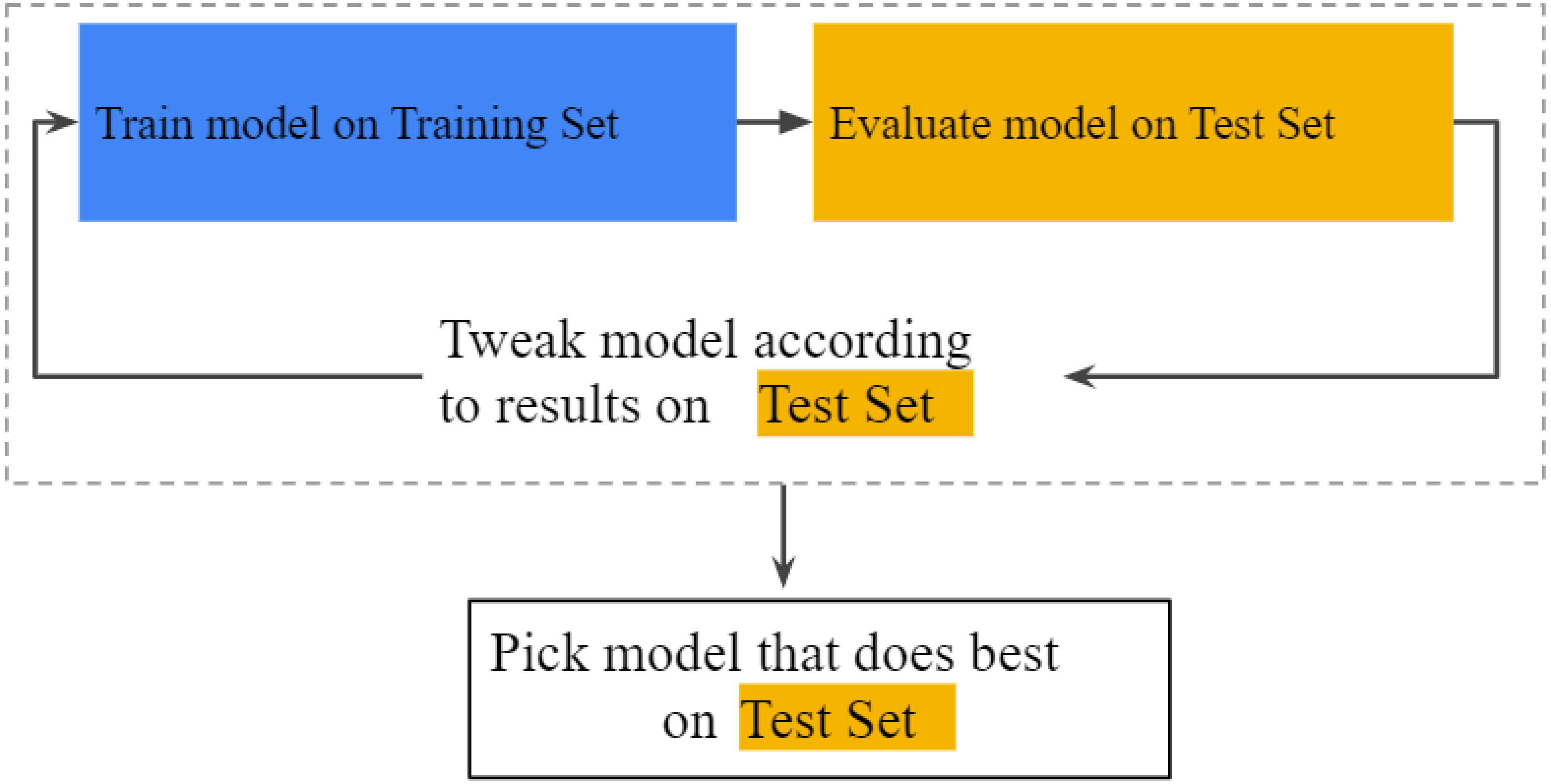
How we split data in Machine Learning?

- Make sure that your test set meets the following two conditions:
  - Is large enough to yield statistically meaningful results.
  - Is representative of the data set as a whole. In other words, don't pick a test set with different characteristics than the training set.



Never train on test data. If you are seeing surprisingly good results on your evaluation metrics, it might be a sign that you are accidentally training on the test set. For example, high accuracy might indicate that test data has leaked into the training set.

How we split data in Machine Learning?



Train model on Training Set

Evaluate model on Test Set

Tweak model according to results on Test Set

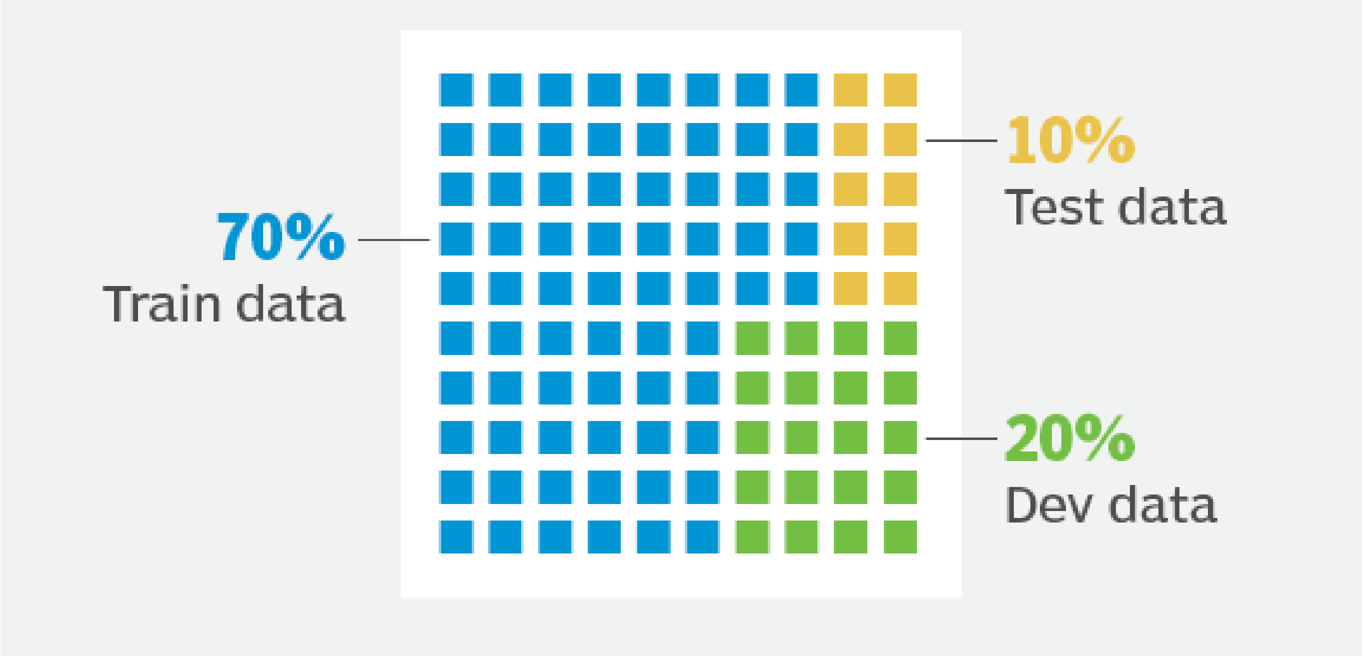
Pick model that does best on Test Set

Data Splitting

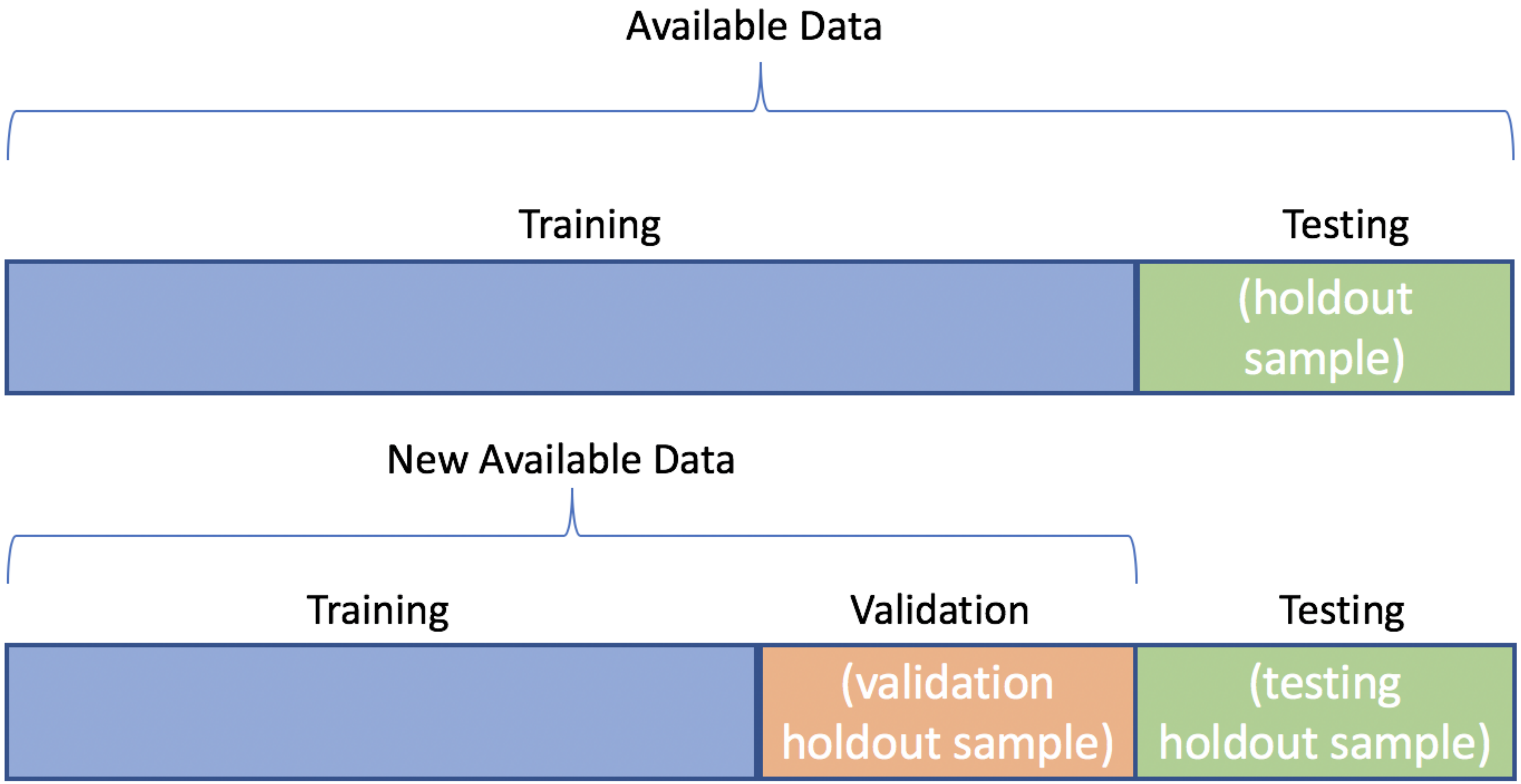
- We looked at a process of using a test set and a training set to drive iterations of model development. On each iteration, we'd train on the training data and evaluate on the test data, using the evaluation results on test data to guide choices of and changes to various model hyperparameters like learning rate and features. **Is there anything wrong with this approach?**

Yes indeed! The more often we evaluate on a given test set, the more we are at risk for implicitly overfitting to that one test set

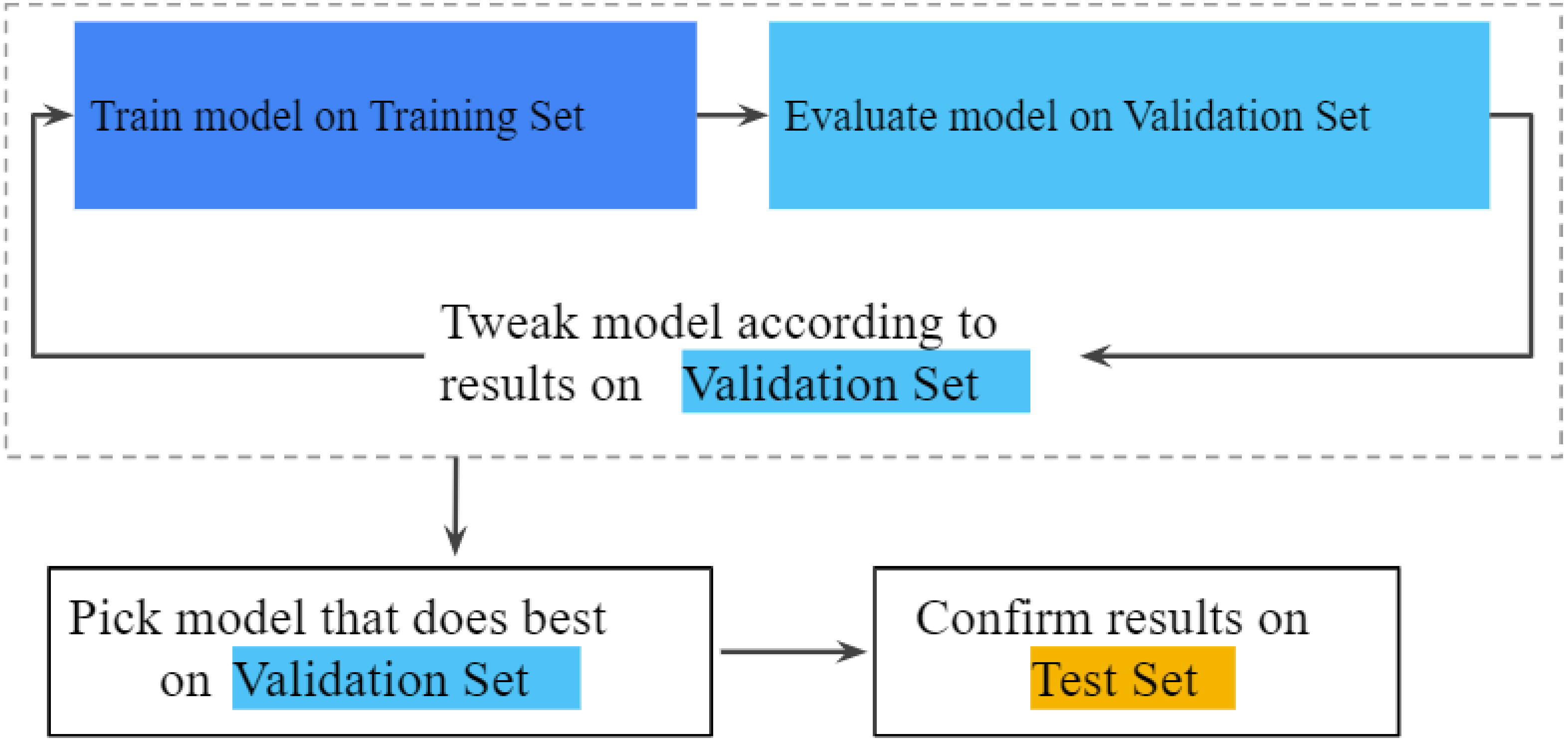
Data Splitting



How we split data in Machine Learning?



How we split data in Machine Learning?



Train model on Training Set

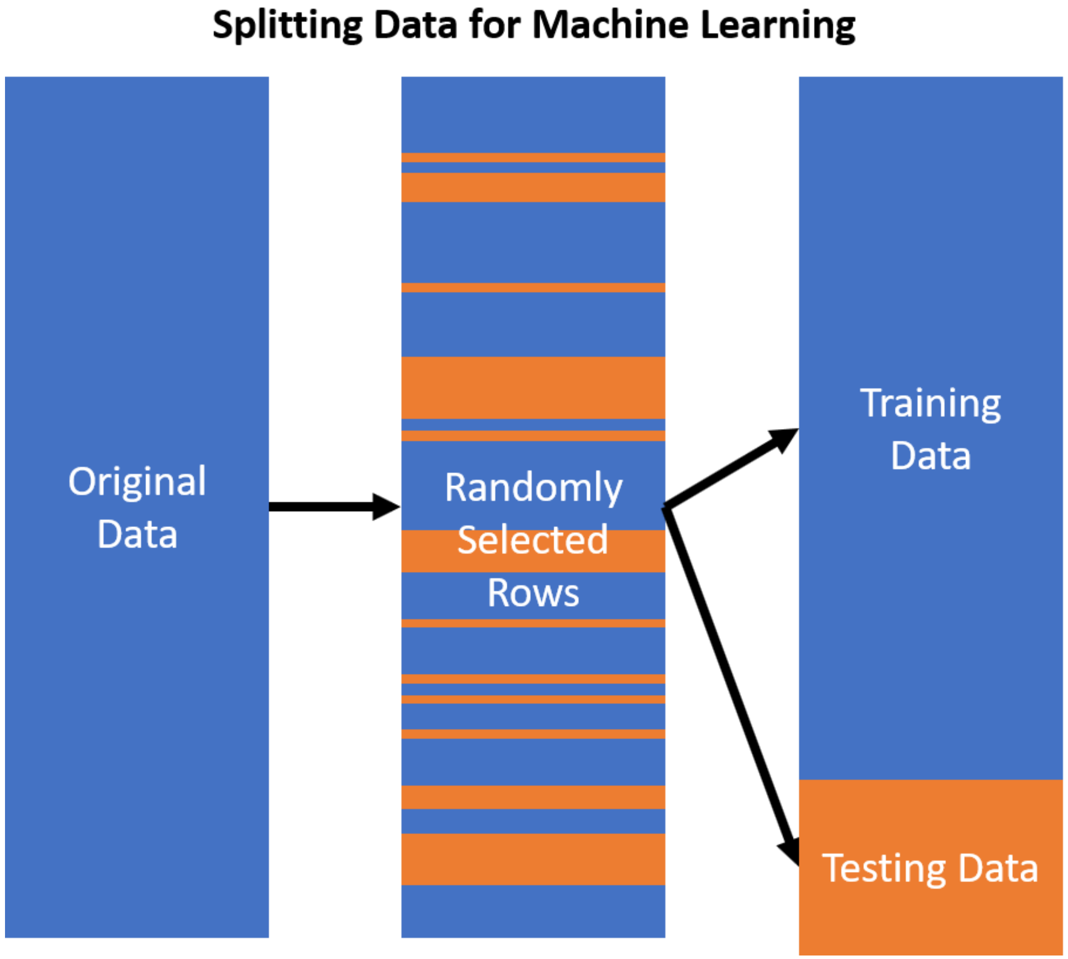
Evaluate model on Validation Set

Tweak model according to results on Validation Set

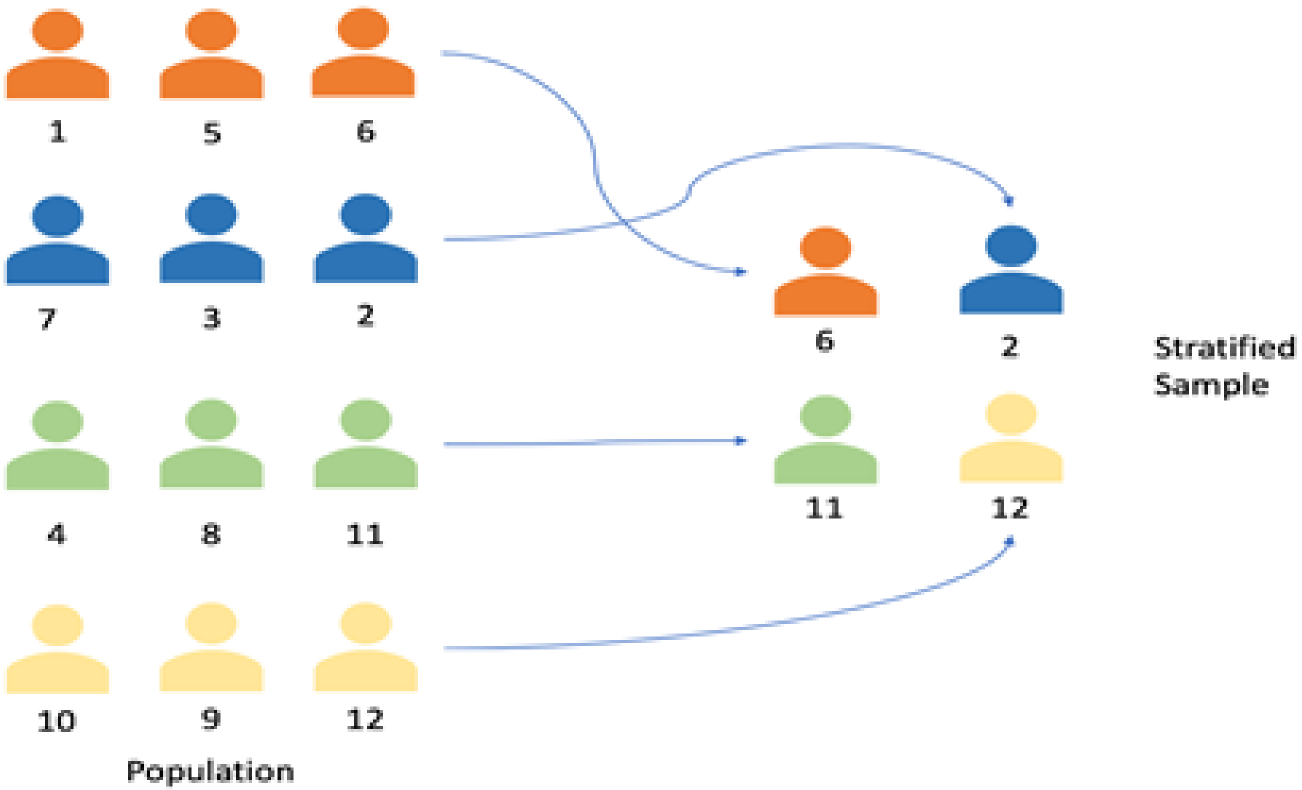
Pick model that does best on Validation Set

Confirm results on Test Set

How we split data in Machine Learning?



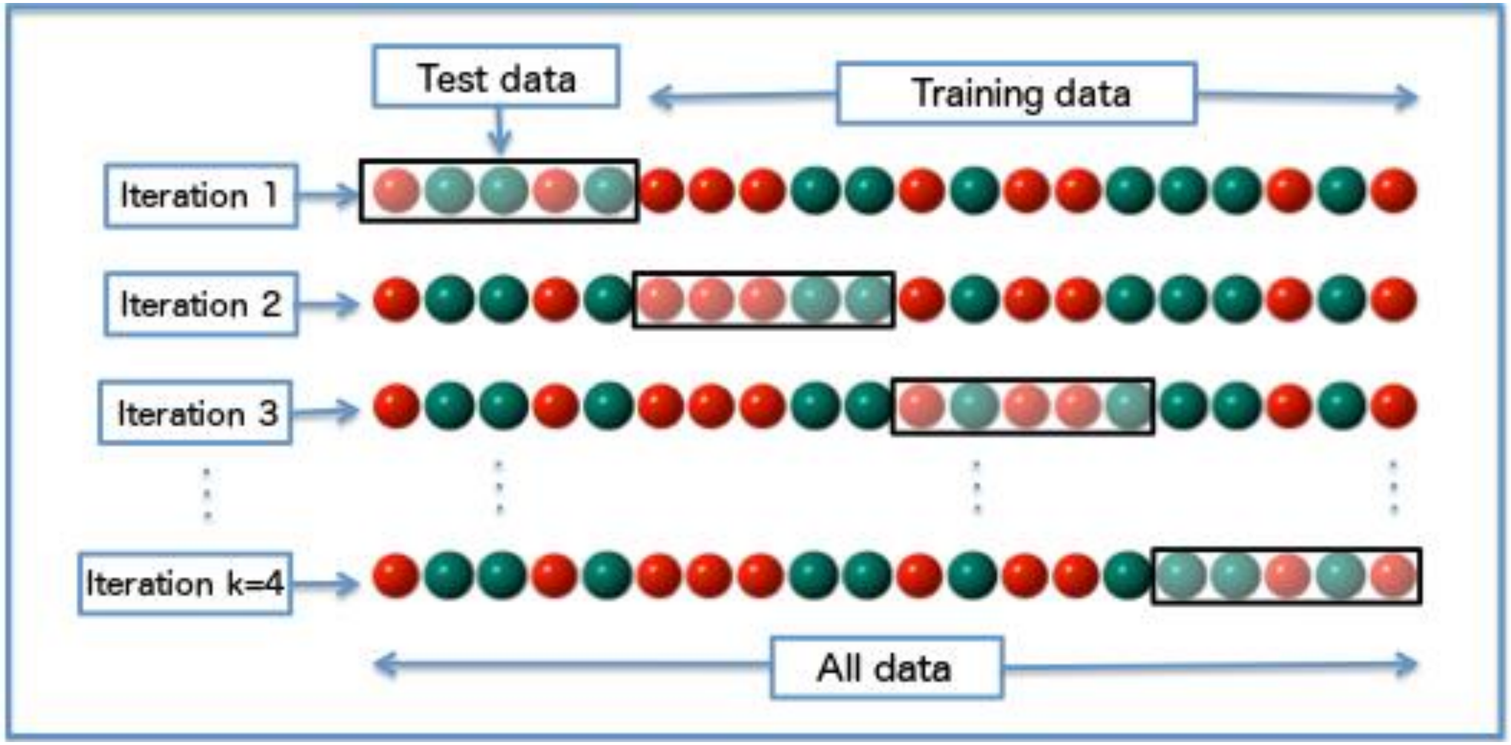
Stratified Split



class proportions are preserved when **splitting**

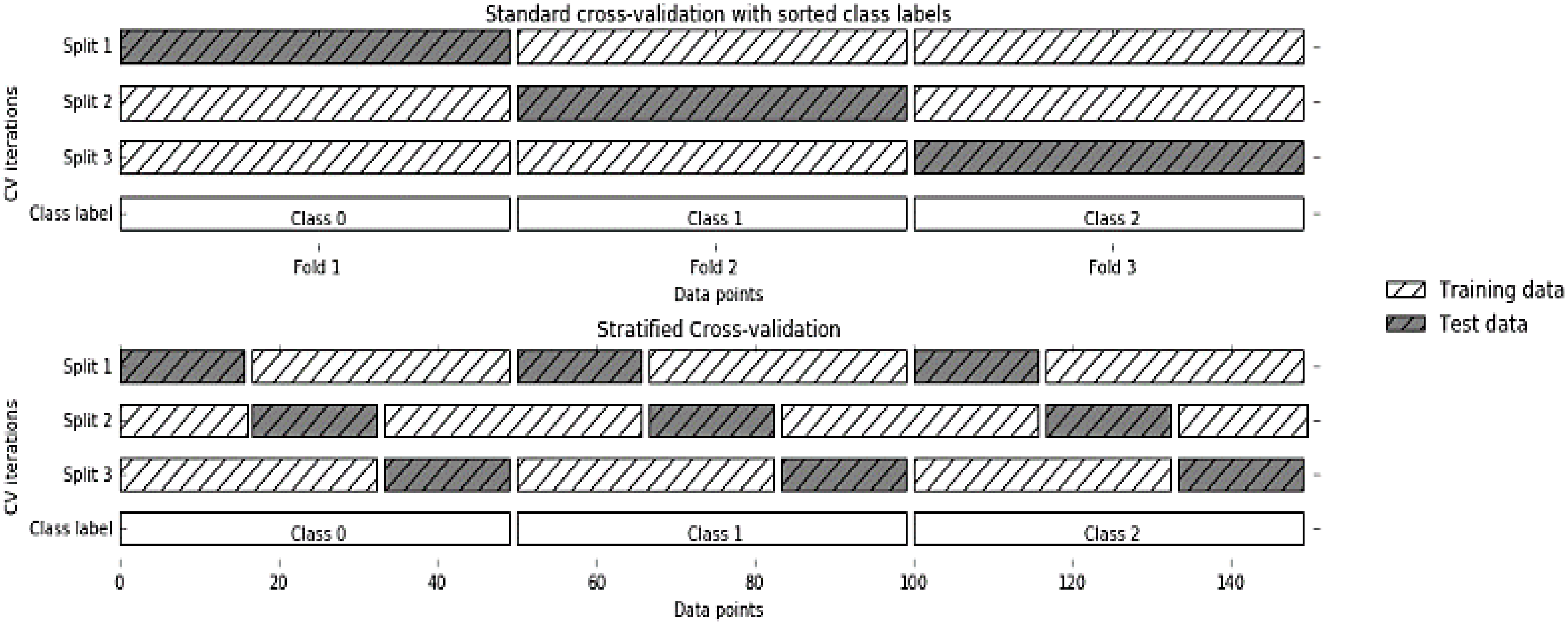
Stratified splitting becomes important when you have class **imbalances**.

K-Cross Validation

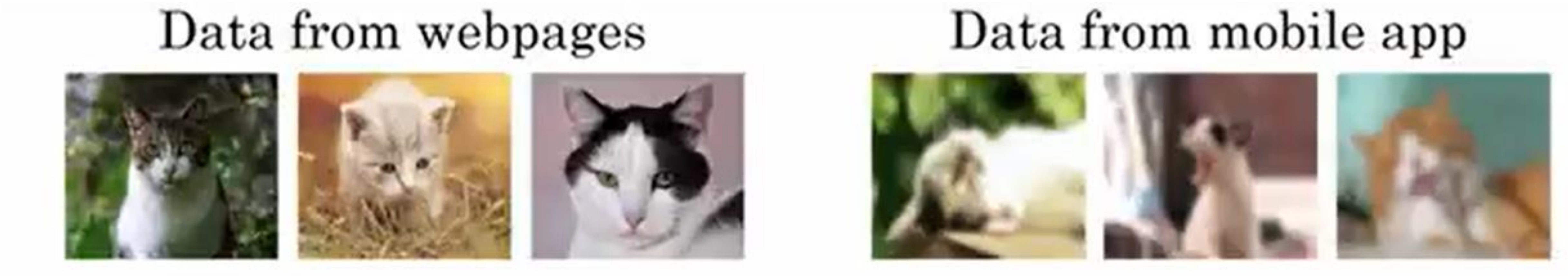


http://puremonkey2010.blogspot.com/2017/03/intro2ml-ch6-model-evaluation-and.html

Stratified K-fold Cross-validation

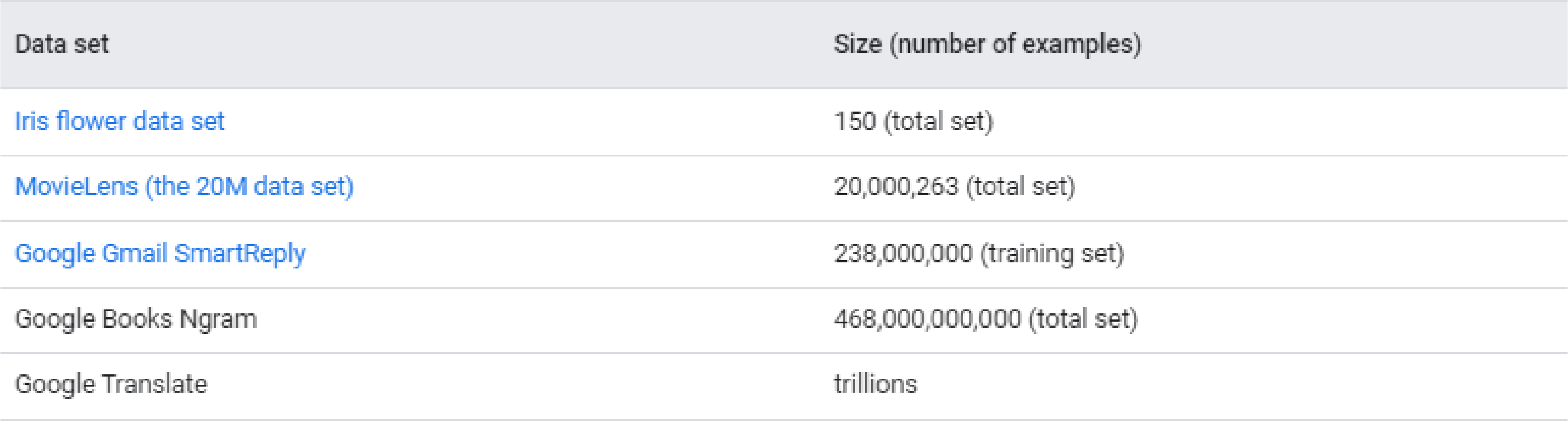


Training and Testing on Different Distribution

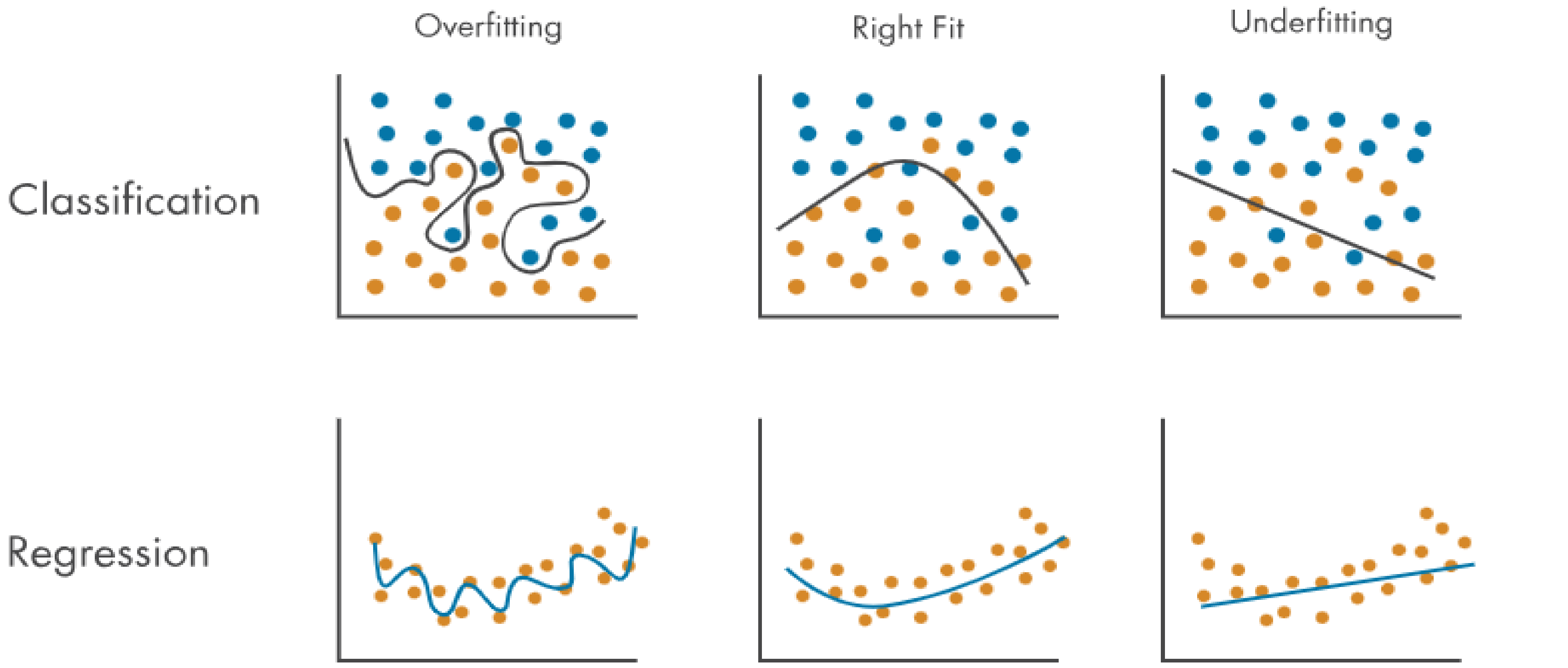


The Size of a Data Set

How much data do you need to get useful results?



Data set	Size (number of examples)
Iris flower data set	150 (total set)
MovieLens (the 20M data set)	20,000,263 (total set)
Google Gmail SmartReply	238,000,000 (training set)
Google Books Ngram	468,000,000,000 (total set)
Google Translate	trillions



Error	Overfitting	Right Fit	Underfitting
Training	Low	Low	High
Test	High	Low	High

Techniques to fight underfitting and overfitting

Underfitting	Overfitting
More complex model	More simple model
Less regularization	More regularization
Larger quantity of features	Smaller quantity of features
More data can't help	More data can help