Preprocessing

In this section of our project, we will apply necessary preprocessing to the dataset to prepare it for downstream modeling. Our modeling strategy will include:

- Baseline model: **Random Forest Classifier**, effective for small datasets and handling nonlinear relationships and class imbalance.
- Second model: **Support Vector Machine (SVM)**, motivated by observed distinct class separation during EDA (t-SNE visualization).
- Third model: A boosted tree model such as **XGBoost, CatBoost, or LightGBM** for potentially improved performance.
- Final model: A **Neural Network** for further experimentation and comparison.

Overview of Preprocessing

Section I: Loading Configuration and Dataset

In this section, we load the necessary configuration files to set up the paths and parameters for the notebook. We then import the Breast Cancer Wisconsin dataset, removing irrelevant columns such as IDs and unnamed index columns.

In [1]:
```
# Loading the configuation and the dataset
import numpy as np
import pandas as pd
import yaml
import pickle

# Loading the configuration files for the notebook
with open("../notebook_config.yaml", 'r') as f:
    config = yaml.safe_load(f)
dataset_path = config['paths']['dataset_path']
eng_features_path = config['paths']['feature_set_eng_path']
raw_features_path = config['paths']['feature_set_raw_path']

# Loading the dataset
df = pd.read_csv(dataset_path)
df = df.drop(columns=['id', 'Unnamed: 32'])
```

Section II: Adding Engineered Features

Here, we augment the raw dataset with additional domain-informed engineered features by combining existing variables. These new features aim to enrich the dataset and

potentially improve model performance in downstream tasks.

In [2]:
```
# Adding the engineered features
df['con_worst_per_area'] = df['concavity_worst'] / df['area_worst']
df['con_mean_per_area'] = df['concavity_mean'] / df['area_mean']

df['fd_worst_perimeter'] = df['fractal_dimension_worst']*df['perimeter_worst']
df['fd_mean_perimeter'] = df['fractal_dimension_mean']*df['perimeter_mean']

df['sym_worst_compactness'] = df['symmetry_worst']*df['compactness_worst']
df['sym_mean_compactness'] = df['symmetry_mean']*df['compactness_mean']

df['smooth_worst_radius'] = df['smoothness_worst']*df['radius_worst']
df['smooth_mean_radius'] = df['smoothness_mean']*df['radius_mean']
```

Section III: Loading Feature Sets

We load the predefined lists of engineered and raw features, which were selected based on previous exploratory data analysis and feature selection.

In [3]:
```
# Loading the two feature sets
with open(eng_features_path, 'rb') as f:
    eng_feature_lst = pickle.load(f)
with open(raw_features_path, 'rb') as f:
    raw_feature_lst = pickle.load(f)
```

Section IV: Preprocessing Pipeline Setup

This section describes our preprocessing strategy where we apply a robust scaler to all numeric features. Robust scaling helps reduce the impact of outliers by using statistics that are robust to extreme values.

In [4]:
```
# Creating a preprocessing pipeline for our dataset
from sklearn.preprocessing import RobustScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

# Lets apply robust scaling to all of the numeric features
scaler = RobustScaler()
numerical_cols = df.select_dtypes(include=['number']).columns.to_list()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

Section V: Splitting Dataset into Train, Validation, and Test Sets

We split the dataset into training, validation, and test subsets with stratification on the target to maintain class balance. Additionally, we prepare separate feature matrices for engineered and raw feature sets for model development and evaluation.

In [5]:
```
# splitting as features (X) and target label (y)
X = df.drop(columns=['diagnosis'])
y = df['diagnosis']
y = y.map({"B": 0, "M": 1})
```

```
# Lets split the data based on the feature sets into two sets of temp/test split
X_temp, X_test, y_temp, y_test = train_test_split(X, y, stratify=y, train_size=0
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, stratify=y_tem

X_train_eng = X_train[eng_feature_lst]
X_val_eng = X_val[eng_feature_lst]
X_test_eng = X_test[eng_feature_lst]
y_train_eng = y_train
y_val_eng = y_val
y_test_eng = y_test

X_train_raw = X_train[raw_feature_lst]
X_val_raw = X_val[raw_feature_lst]
X_test_raw = X_test[raw_feature_lst]
y_train_raw = y_train
y_val_raw = y_val
y_test_raw = y_test
```

Section VI: Saving Processed Splits

To facilitate reproducibility and further experimentation, all train, validation, and test splits—with their respective engineered and raw feature subsets—are saved as CSV files. This ensures a consistent and easy-to-share data pipeline for modeling stages.

In [6]:
```
# Lets Save the splits as csv files
X_train_eng.to_csv('../dataset/X_train_eng.csv', index=False)
X_val_eng.to_csv('../dataset/X_val_eng.csv', index=False)
X_test_eng.to_csv("../dataset/X_test_eng.csv", index=False)
y_train_eng.to_csv('../dataset/y_train_eng.csv', index=False)
y_val_eng.to_csv('../dataset/y_val_eng.csv', index=False)
y_test_eng.to_csv("../dataset/y_test_eng.csv", index=False)

# Save raw features and labels
X_train_raw.to_csv('../dataset/X_train_raw.csv', index=False)
X_val_raw.to_csv('../dataset/X_val_raw.csv', index=False)
X_test_raw.to_csv("../dataset/X_test_raw.csv", index=False)
y_train_raw.to_csv('../dataset/y_train_raw.csv', index=False)
y_val_raw.to_csv('../dataset/y_val_raw.csv', index=False)
y_test_raw.to_csv("../dataset/y_test_raw.csv", index=False)

# Save main splits as well
X_train.to_csv("../dataset/X_train.csv", index=False)
X_val.to_csv("../dataset/X_val.csv", index=False)
X_test.to_csv("../dataset/X_test.csv", index=False)
y_train.to_csv("../dataset/y_train.csv", index=False)
y_val.to_csv("../dataset/y_val.csv", index=False)
y_test.to_csv("../dataset/y_test.csv", index=False)
```