-----------Feature Engineering----------

**1. Handling Missing Values**

* **Mean/Median Imputation**: Replace missing values with the mean or median, useful for numeric data with minimal impact on variance.
* **Mode Imputation**: For categorical data, replace missing values with the most frequent category.
* **Predictive Imputation**: Use algorithms to predict and fill missing values, preserving data distribution.
* import pandas as pd
* import numpy as np
* from sklearn.impute import SimpleImputer
* *# Sample dataset with missing values*
* data = {
* 'NumericFeature': [10, 15, None, 30, None, 45],
* 'CategoricalFeature': ['A', 'B', 'A', None, 'B', None],
* 'Target': [1, 0, 1, 1, 0, 0]
* }
* df = pd.DataFrame(data)
* *# Replace None with np.nan*
* df.replace({None: np.nan}, inplace=True)
* *# Display initial data*
* print("Original Data:")
* print(df)
* *# 1. Mean/Median Imputation for Numeric Features*
* *# Mean Imputation*
* mean\_imputer = SimpleImputer(strategy='mean')
* df['NumericFeature\_Mean'] = mean\_imputer.fit\_transform(df[['NumericFeature']])
* *# Median Imputation*
* median\_imputer = SimpleImputer(strategy='median')
* df['NumericFeature\_Median'] = median\_imputer.fit\_transform(df[['NumericFeature']])
* *# 2. Mode Imputation for Categorical Features*
* mode\_imputer = SimpleImputer(strategy='most\_frequent')
* df['CategoricalFeature\_Mode'] = mode\_imputer.fit\_transform(df[['CategoricalFeature']]).flatten()
* *# Display results*
* print("\nData after Imputation:")
* print(df)

**2. Encoding Categorical Variables**

* **Label Encoding**: Assign a unique number to each category, useful for ordinal variables.
* import pandas as pd
* from sklearn.preprocessing import LabelEncoder
* *# Sample data*
* data = {'Size': ['Small', 'Medium', 'Large', 'Medium', 'Small']}
* df = pd.DataFrame(data)
* *# Label Encoding*
* label\_encoder = LabelEncoder()
* df['Size\_Encoded'] = label\_encoder.fit\_transform(df['Size'])
* print(df)
* **One-Hot Encoding**: Creates a binary column for each category, best for nominal variables with fewer unique values.
* import pandas as pd
* *# Sample data*
* data = {'Color': ['Red', 'Blue', 'Green', 'Blue', 'Red']}
* df = pd.DataFrame(data)
* *# One-Hot Encoding*
* df\_encoded = pd.get\_dummies(df, columns=['Color'])
* print(df\_encoded)
* **Target Encoding**: Replace categories with the mean of the target variable for each category.
* import pandas as pd
* *# Sample data*
* data = {'Category': ['A', 'B', 'A', 'A', 'B', 'C'],
* 'Target': [1, 2, 1, 2, 3, 2]}
* df = pd.DataFrame(data)
* *# Target Encoding*
* target\_mean = df.groupby('Category')['Target'].mean()
* df['Category\_Target\_Encoded'] = df['Category'].map(target\_mean)
* print(df)



**3. Feature Scaling**

* **Standardization (Z-score scaling)**: Transform features to have a mean of 0 and standard deviation of 1, important for algorithms sensitive to feature scale (e.g., SVM, k-NN).
* This technique transforms features to have a mean of 0 and a standard deviation of 1. It’s especially useful for algorithms like SVM and k-NN.
* import pandas as pd
* from sklearn.preprocessing import StandardScaler
* *# Sample data*
* data = {'Feature1': [10, 20, 30, 40, 50],
* 'Feature2': [15, 25, 35, 45, 55]}
* df = pd.DataFrame(data)
* *# Standardization*
* scaler = StandardScaler()
* df\_standardized = pd.DataFrame(scaler.fit\_transform(df), columns=df.columns)
* print("Standardized Data:")
* print(df\_standardized)
* **Normalization (Min-Max scaling)**: Rescales features to a range, often [0, 1], useful for neural networks.