Feature Engineering

Feature engineering is the process of using domain knowledge to extract features (characteristics, properties, attributes) from raw data. The motivation is to use these extra features to improve the quality of results from a machine learning process, compared with supplying only the raw data to the machine learning process.

It involves selecting, transforming, extracting, combining, and manipulating raw data to generate the desired variables for analysis or predictive modeling. It is a crucial step in developing a machine learning model.

Domain knowledge refers to the understanding and expertise of a specific field or industry that a data scientist or analyst possesses. This knowledge helps them make informed decisions when working with data, creating new features, and building machine learning models. In the context of feature engineering, domain knowledge allows the data scientist to identify relevant variables, understand the relationships between them, and create new, meaningful features that can improve the performance of a machine learning model.

There are several techniques that can be used in feature engineering to create new features or transform existing ones. Some common techniques include:

- **Feature scaling**: This involves transforming numeric features to have a similar scale. Common methods include min-max scaling and standardization (z-score normalization).
- **One-hot encoding**: This is used to convert categorical variables into a binary vector representation that can be used in machine learning models.
- **Binning**: This involves grouping continuous variables into discrete bins or categories. This can help reduce the impact of small variations in the data and make it easier for machine learning models to learn relationships between variables.
- **Feature transformation**: This involves applying mathematical functions (e.g., log, square root) to features to change their distribution or relationship with other variables.
- **Feature interaction**: This involves creating new features by combining or multiplying existing features. This can help capture complex relationships between variables that may not be apparent when considering each feature individually.

Data Encoding:

Data Encoding is the process of converting data from one format to another. In the context of machine learning and data analysis, encoding often refers to the process of converting categorical data into numerical data that can be used in a model. There are several methods for encoding categorical data, including one-hot encoding, label encoding, and binary encoding. Each method has its own advantages and disadvantages and the choice of method depends on the specific problem at hand. Encoding is an important step in preparing data for machine learning algorithms, as many algorithms cannot handle categorical data in its raw form.

Some common techniques used in feature engineering include:

- **Numerical transformations:** These can include scaling, taking logarithms, or calculating ratios to transform numerical data into more useful forms for analysis.
- **Categorical encoding:** This involves converting categorical data into numerical form so that it can be used in machine learning models. Common techniques include one-hot encoding and target encoding.
- **Clustering:** This technique can be used to group similar data points together and create new features based on the cluster assignments. Group aggregated values: This involves calculating summary statistics such as the mean, median, or standard deviation for groups of data points and using these values as new features.
- **Principal component analysis (PCA):** This technique can be used to reduce the dimensionality of numerical data by creating new features that capture the most important sources of variation in the data.
- **Feature construction:** This involves using domain knowledge to create new features that are relevant to the problem at hand. For example, in physics, construction of dimensionless numbers such as Reynolds number in fluid dynamics or Nusselt number in heat transfer. These are just a few examples of the many techniques that can be used in feature engineering to transform raw data into more useful forms for analysis and modeling.
- **One-hot encoding** is a technique used to convert categorical data into numerical form so that it can be used in machine learning models. It works by creating a new binary column for each unique category in the data. For example, if you have a categorical variable from the employee_df with respect to the Education variable which has three categories: “Bachelors”,

"Masters", and "PHD", one-hot encoding would create three new binary columns: one for "Bachelors", one for "Masters", and one for "PHD". Each row in the data would then have a 1 in the column corresponding to its category and 0s in the other columns. This allows machine learning algorithms to treat categorical data as numerical data and use it in their calculations.

One-hot encoding can be useful when working with categorical data that has no inherent order or hierarchy. However, it can also result in a large increase in the number of columns in the data, especially if the categorical variable has many unique categories. This can lead to increased computational complexity and memory usage when training machine learning models.

- **Target encoding** is another technique used to convert categorical data into numerical form for use in machine learning models. It works by replacing each category in a categorical variable with the average value of the target variable for that category. For example, if you have a binary classification problem with a categorical variable "color" and a target variable "outcome", you could calculate the mean outcome for each color and use these values to replace the original color categories. There are several alternatives to one-hot encoding for converting categorical data into numerical form for use in machine learning models.

Some common alternatives include:

- **Ordinal encoding:** This technique involves assigning a unique integer value to each category in the data. For example, if you have a categorical variable with three categories: "low", "medium", and "high", you could assign the values 1, 2, and 3 to these categories respectively. This approach can be useful when the categorical variable has an inherent order or hierarchy.
- **Binary encoding:** This technique involves converting each category into a binary code and representing it using multiple binary columns. For example, if you have a categorical variable with four categories: ["A", "B", "C", and "D"], you could represent these categories using two binary columns as follows: "A" = 00, "B" = 01, "C" = 10, and "D" = 11. This approach can be useful when the categorical variable has many unique categories, as it can result in fewer new columns than one-hot encoding.
- **Count encoding:** This technique involves replacing each category with the count of how many times it appears in the data. For example, if you have a categorical variable with three

categories: “red”, “green”, and “blue”, and the counts of these categories are 10, 20, and 30 respectively, you could replace the original categories with these counts.

- **Target encoding:** as mentioned earlier, target encoding involves replacing each category with the average value of the target variable for that category. These are just a few examples of the many techniques that can be used to convert categorical data into numerical form for use in machine learning models. The choice of technique will depend on the specific characteristics of the data and the problem at hand.

  Target encoding can be useful when working with high cardinality categorical variables, where one-hot encoding would result in a large number of new columns. It can also help capture the relationship between the categorical variable and the target variable. However, it’s important to be careful when using target encoding to avoid overfitting or data leakage. **One common approach is to use cross-validation or a holdout set to calculate the target means, rather than using the entire dataset.**

The choice of technique will depend on the specific problem and data at hand, as well as the domain knowledge of the data scientist or analyst.

To enhance the feature engineering process, you can consider the following techniques and transformations:

 # Binning 'Age' into age groups

- *employee_df['AgeGroup'] = pd.cut(employee_df['Age'], bins=[20, 30, 40, 50, 60], labels=['20-29', '30-39', '40-49', '50-59'])*

Here's a detailed explanation of the code:

1. **employee_df['AgeGroup']**: A new column 'AgeGroup' is created in the employee_df DataFrame to store the binned age groups.
2. **pd.cut()**: The pandas **cut()** function is used to bin the 'Age' feature into discrete intervals.
3. **employee_df['Age']**: This is the input data (the 'Age' column) for the **cut()** function.
4. **bins=[20, 30, 40, 50, 60]**: The **bins** parameter specifies the edges of the bins or intervals. In this case, the bins are defined as (20, 30], (30, 40], (40, 50], and (50, 60]. Note that the left bracket is exclusive, while the right bracket is inclusive.
5. **labels=['20-29', '30-39', '40-49', '50-59']**: The **labels** parameter assigns a custom label for each bin, making the output more readable. In this example, the labels correspond to the age groups 20-29, 30-39, 40-49, and 50-59.

After executing the code, the employee_df DataFrame will have a new column 'AgeGroup' that contains the corresponding age group for each employee's age. This new feature can be used in the machine learning model to potentially improve its performance.

In the context of machine learning, linear models, such as linear regression, assume a linear relationship between the input features and the target variable. However, real-world data often contain non-linear relationships that linear models might not capture effectively.

By transforming features to better represent non-linear relationships, we can potentially improve the performance of the machine learning model.

Binning is one technique that can help capture non-linear relationships. By converting a numerical feature into discrete categories (bins), the model can identify patterns or trends within these categories that might not be apparent when using the raw numerical data.

For our use case we are going to use one-hot encoding for the categorical variables.

One Hot Encoding

One-hot encoding is a technique used to convert categorical data (variables or features) into binary (dummy) variables (numerical form), which take the values 0 or 1 so that it can be used in machine learning models. Each category within a categorical variable is represented by a separate binary variable, indicating the presence or absence of that category in an observation.

For example, if you have a categorical variable from the employee_df with respect to the Education variable which has three categories: “Bachelors”, “Masters”, and “PHD”, one-hot encoding would create three new binary columns: one for “Bachelors”, one for “Masters”, and one for “PHD”. Each row in the data would then have a 1 in the column corresponding to its category and 0s in the other columns. This allows machine learning algorithms to treat categorical data as numerical data and use it in their calculations.

Secondly, for the categorical variable "Gender", with labels 'female' and 'male', we can generate the boolean variable "female", which takes 1 if the person is 'female' or 0 otherwise, or we can generate the variable "male", which takes 1 if the person is 'male' and 0 otherwise.

Example: Encoding Gender, City, Variables

For the categorical variable "Gender", with labels 'female' and 'male', we can generate a binary variable "female", which takes 1 if the person is 'female' or 0 otherwise, or we can generate a binary variable "male", which takes 1 if the person is 'male' and 0 otherwise.

For the categorical variable "colour" with values 'red', 'blue' and 'green', we can create 3 new variables called "red", "blue" and "green". These variables will take the value 1, if the observation is of the said colour or 0 otherwise.

Encoding into k-1 dummy variables.

****When one hot encoding categorical variables, we create k - 1 binary variables****

Most machine learning algorithms, consider the entire data set while being fit. Therefore, encoding categorical variables into k - 1 binary variables, is better, as it avoids introducing redundant information.

Note however, that for the variable "colour", by creating 2 binary variables, say "red" and "blue", we already encode **ALL** the information.

- if the observation is red, it will be captured by the variable "red" (red = 1, blue = 0)

- if the observation is blue, it will be captured by the variable "blue" (red = 0, blue = 1)

- if the observation is green, it will be captured by the combination of "red" and "blue" (red = 0, blue = 0)

We do not need to add a third variable "green" to capture that the observation is green.

More generally, a categorical variable should be encoded by creating k-1 binary variables, where k is the number of distinct categories. In the case of gender, k=2 (male / female), therefore we need to create only 1 (k - 1 = 1) binary variable. In the case of colour, which has 3 different categories (k=3), we need to create 2 (k - 1 = 2) binary variables to capture all the information.

One hot encoding into k-1 binary variables takes into account that we can use 1 less dimension and still represent the whole information: if the observation is 0 in all the binary variables, then it must be 1 in the final (not present) binary variable.

 # Exception: One hot encoding into k dummy variables

There are a few occasions when it is better to encode variables into k dummy variables:

- when building tree-based algorithms

- when doing feature selection by recursive algorithms

- when interested in determine the importance of each single category

Tree based algorithms, as opposed to the majority of machine learning algorithms, **do not evaluate the entire dataset while being trained.**

They randomly extract a subset of features from the data set at each node for each tree.

Therefore, if we want a tree-based algorithm to consider **all the categories, we need to encode categorical variables into **k binary variables**.**

If we are planning to do feature selection by recursive elimination (or addition), or if we want to evaluate the importance of each single category of the categorical variable, then we will also need the entire set of binary variables (k) to let the machine learning model select which ones have the most predictive power.

Additionally, if we are planning to do feature selection by recursive elimination or addition, or if we want to evaluate the importance of each single category of the categorical variable, then we will also need the entire set of binary variables (k) to let the machine learning model select which ones have the most predictive power. This is because using k-1 binary variables would not allow the model to consider all categories individually.

Advantages of one hot encoding

- Straightforward to implement

- Makes no assumption about the distribution or categories of the categorical variable

- Keeps all the information of the categorical variable

- Suitable for linear models

Limitations

- Expands the feature space

- Does not add extra information while encoding

- Many dummy variables may be identical, introducing redundant information

 # Notes

If our datasets contain a few highly cardinal variables, we will end up very soon with datasets with thousands of columns, which may make training of our algorithms slow, and model interpretation hard.

In addition, many of these dummy variables may be similar to each other, since it is not unusual that 2 or more variables share the same combinations of 1 and 0s. Therefore, one hot encoding may introduce redundant or duplicated information even if we encode into k-1.

In this demo:

We will see how to perform one hot encoding with:

- pandas

- Scikit-learn

- Feature-Engine

The 'PaymentTier' feature already has values of 1, 2, and 3, hence there is no need to apply ordinal encoding.
In this case, the feature is already in a suitable numerical format for machine learning algorithms. Ordinal encoding is necessary when the values of an ordinal feature are in a non-numeric format, such as strings (ordinal_encoding_payment_tier = **{'Tier 1': 1,'Tier 2': 2,'Tier 3': 3}** or labels that indicate an order or rank.

```
# One-hot encoding of City and Education
cat_features = ['City', 'Education']
employee_df = pd.get_dummies(employee_df, columns=cat_features, drop_first=True)

# Binarizing Gender and EverBenched features:
employee_df['Gender'] = employee_df['Gender'].replace('Male', 1).replace('Female', 0)
employee_df['EverBenched'] = employee_df['EverBenched'].replace('Yes', 1).replace('No', 0)
```

The **'pd.get_dummies()' function is used to one-hot encode the 'City' feature in the 'employee_df' DataFrame.**

The 'columns' parameter is set to 'cat_features', which indicates that only the 'City' column will be one-hot encoded.

The 'drop_first=True' parameter ensures that the first category for the 'City' feature is dropped, to avoid multicollinearity.

When one-hot encoding is applied to a categorical feature, it creates a new binary feature for each category in the original feature. However, including all of these binary features in a model can lead to multicollinearity, which is when two or more features are highly correlated. In the presence of multicollinearity, it becomes difficult for the model to determine the individual effect of each feature on the target variable, as they can be easily explained by other correlated features. This can lead to unstable estimates and reduced interpretability of the model.

To avoid multicollinearity, one common practice is to drop one of the binary features created during one-hot encoding. The rationale behind this is that if all other categories have a value of 0, the dropped category is implicitly represented as 1. For example, **when one-hot encoding the 'City' feature with categories 'Bangalore', 'Pune', and 'New Delhi', we create three binary features: 'City_Bangalore', 'City_Pune', and 'City_New Delhi'. By dropping the first category (e.g., 'City_Bangalore'), we can still infer its value based on the other two categories. If 'City_Pune' and 'City_New Delhi' are both 0, it means the city must be 'Bangalore'.**

In the code, the **'drop_first=True'** parameter in the **'pd.get_dummies()'** function is used to drop the first category of the **'City' feature.** This helps to prevent multicollinearity by reducing the number of correlated binary features in the model.

- **pd.get_dummies()** is a function in the pandas library that is used to convert categorical variables into a numerical format known as one-hot encoding or dummy variables. When you have a categorical feature with multiple categories, one-hot encoding creates new binary features for each category in the original feature, with a value of 1 if the category is present for that observation and 0 otherwise.

Explanation of how pd.get_dummies() works:

1. The function takes a pandas DataFrame as input, along with some optional parameters.
2. It identifies the categorical columns that need to be one-hot encoded. You can either let the function automatically detect these columns based on their data type or explicitly specify the columns to encode using the **columns** parameter.
3. For each categorical column, it creates new binary columns for each unique category, named by combining the original column name with the category (e.g., 'City_Bangalore', 'City_Pune', 'City_New Delhi').

4. It replaces the original categorical columns in the DataFrame with the new binary columns.
5. If **drop_first=True** is specified, it drops the first binary column created for each categorical feature to avoid multicollinearity.

python Copy code

```python
import pandas as pd

data = {'City': ['Bangalore', 'Pune', 'New Delhi', 'Pune']}
df = pd.DataFrame(data)

# One-hot encoding the 'City' column
df_encoded = pd.get_dummies(df, columns=['City'])

print(df_encoded)
```

Output:

Copy code

```
   City_Bangalore  City_New Delhi  City_Pune
0               1               0          0
1               0               0          1
2               0               1          0
3               0               0          1
```