What is Feature Engineering?

It is the process of using domain knowledge to select and transform the most relevant variables from raw data to improve the performance of machine learning algorithms or statistical modeling.

> The goal of feature engineering and selection is to improve the performance of machine learning (ML) algorithms.

Types of Feature Engineering:

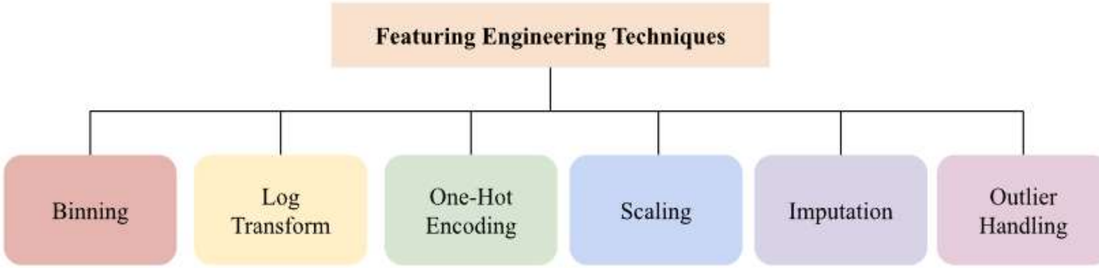
Feature engineering consists of creation, transformation, extraction, and selection of features(variables), to creating an accurate ML algorithm. These processes entail:

1. **Feature Construction:** Creating more useful features to improve the predictive model. It requires domain knowledge to manipulate Existing features via addition, subtraction, multiplication, and ratio, etc to create new derived features that have greater predictive power.

2. **Feature Transformations:** It is a mathematical transformation in which we apply a mathematical formula to a particular column (feature) and transform the values, which are useful for our further analysis e.g. avoiding computational errors by ensuring all features are within an acceptable range/ scale for the model.

3. **Feature Extraction:** Feature extraction aims to reduce data complexity (often known as "data dimensionality") while retaining as much relevant information as possible. This helps to improve the performance and efficiency of machine learning algorithms and simplify the analysis process using techniques such as Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-DSNE).

4. **Feature Selection:** Identify and select the most relevant features to improve model interpretability and efficiency using techniques like univariate feature selection or recursive feature elimination.



Feature Engineering in ML Lifecycle

Feature engineering involves transforming raw data into a format that enhances the performance of machine learning models. The key steps in feature engineering include:

1. Imputation

Missing values are among the most common problems faced when it requires getting your data prepared for machine learning. Imputation is a method of dealing with missing values. It is intended for managing anomalies inside the dataset. There are two kinds of imputations:

- *Numerical imputation*: Numerical imputation is implemented to fill gaps in assessments or polls when particular data bits are unavailable.
- *Categorical data imputation*: Missing values in categorical data imputation could be replaced with the highest value that occurs in a column.

2. Outlier Handling

A method to eliminate outliers from the data set is known as outlier handling. This technique can be applied to various levels to provide more precise data representations. This phase must be carried out before beginning model training. The Z-score and standard deviation might be used for identifying outliers. There are certain ways to handle outliers:

- *Removal*: The distribution is cleaned up by removing items that contain outliers.
- *Replacing Values*: Outliers can be interpreted as equivalent missing data and substituted with appropriate imputation.
- *Capping*: Replacing the largest and smallest numbers with an arbitrary value or that comes from a variable range.
- *Discretization*: It is the method of turning continuous variables, models, and functions into discrete variables, models, and functions.

3. One-Hot Encoding

One-hot encoding is a form of encoding wherein each member of a finite set is expressed by its index, with only one component having its index set to “1” and the remainder of the elements being given indices that fall within a specific range. It is a method that transforms categorical data into a format that machine learning algorithms can easily understand and use to produce accurate predictions.

4. Transformers:

Like Linear and Logistic regression, some data science models assume that the variables follow a normal distribution. More likely, variables in real datasets will follow a skewed distribution. By applying some transformations to these skewed variables, we can map this skewed distribution to a normal distribution to increase the performance of our models.

- *Log Transformation*: The log transform is commonly used to convert a skewed distribution into a normal or less skewed one. This transformation is not applied to those features which have negative values. This transformation is mostly applied `toright-skewed` data. Convert data from the addictive scale to multiplicative scale, i.e., linearly distributed data.

- *Reciprocal Transformation*: This transformation is not defined for zero. It is a powerful transformation with a radical effect. This transformation reverses the order among values of the same sign, so large values become smaller and vice-versa.

- *Square Transformation*: This transformation mostly applies to left-skewed data.

- *Square Root Transformation*: This transformation is defined only for positive numbers. This can be used for reducing the skewness of right-skewed data. This transformation is weaker than Log Transformation.

- *Custom Transformation*: A Function Transformer forwards its X (and optionally y) arguments to a user-defined function or function object and returns this function's result. The resulting transformer will not be pickleable if lambda is used as the function. This is useful for stateless transformations such as taking the log of frequencies, doing custom scaling, etc.

- *Power Transformations*: Power transforms are a family of parametric, monotonic transformations that make data more Gaussian-like. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood. This is useful for modeling issues related to non-constant variance or other situations where normality is desired. Currently, Power Transformer supports the Box-Cox transform and the Yeo-Johnson transform.

By default, zero-mean, unit-variance normalization is applied to the transformed data.

a. *Box-Cox Transformation*: It requires the input data to be strictly positive (not even zero is acceptable). Sqrt/sqr/log are the special cases of this transformation. b. *Yeo-Johnson Transformation*: It is a variation of the Box-Cox and supports both positive and negative data.

5. Feature Scaling

Standardize or normalize numerical features to ensure they are on a similar scale, improving model performance. There are two standard methods of scaling:

- *Normalization:* In this process, all values are scaled between a particular range between 0 and 1.

- *Standardization:* It is also known as Z-score normalization. It is the process of measuring values while considering their standard deviation.

6. Binning

Among the key problems that affect the effectiveness of the model in machine learning is overfitting, which happens because of more parameters and inaccurate data. Binning is a method for converting continuously varying variables to categorical variables. In this procedure, the continuous variable's spectrum of values is divided into numerous bins, and each one gets allocated a category value.

7. Text Data Processing

If dealing with text data, perform tasks such as tokenization, stemming, and removing stop words.

8. Time Series Features

Extract relevant timebased features such as lag features or rolling statistics for time series data.

9. Vector Features

Vector features are commonly used for training in machine learning. In machine learning, data is represented in the form of features, and these features are often organized into vectors. A vector is a mathematical object that has both magnitude and direction and can be represented as an array of numbers.

Feature engineering is often an iterative process; evaluate, refine, and repeat as needed to achieve optimal results. These steps may vary based on the nature of the data and the machine learning task, but collectively, they contribute to creating a robust and effective feature set for model training.

> > Cross-validation: Selecting features prior to cross-validation can introduce significant bias. Evaluate the impact of feature engineering on model performance using cross-validation techniques.

Importing and Understanding the Data

In [1]:

```
import pandas as pd
```

In [2]:

```
data = pd.read_csv("DAAssignment.csv")
print(data.columns)
print(data.shape)
```

```
Index(['Id', 'Requester id', 'Group', 'Status', 'Priority', 'Via',
       'Created at', 'Updated at', 'Assigned at', 'Initially assigned at',
       'Solved at', 'Resolution time', 'Satisfaction Score', 'Reopens',
       'Replies', 'First reply time in minutes within business hours',
       'First resolution time in minutes',
       'First resolution time in minutes within business hours',
       'Full resolution time in minutes',
       'Full resolution time in minutes within business hours',
       'Requester wait time in minutes',
       'Requester wait time in minutes within business hours',
       'Manual Tagging of Categories [list]'],
      dtype='object')
(16476, 23)
```

In [3]:

```
data = data.drop(columns=['Id', 'Created at', 'Updated at', 'Assigned at', 'Initially a
       'Solved at', 'Resolution time', 'Manual Tagging of Categories [list]'], axis= 1)

data
```

Out[3]:

	Requester id	Group	Status	Priority	Via	Satisfaction Score	Reopens	Replies	First reply time in minutes within business hours
0	10300555531549	Support	Solved	Low	Mail	Offered	11	11	173.0
1	10420228868125	Reimbursement Claims	Closed	Low	Mail	NaN	10	10	1527.0
2	10991633548957	Support	Solved	Low	Mail	4	10	11	61.0
3	10376247288477	Support	Solved	Low	Mail	Offered	9	9	381.0
4	7302858920989	Support	Closed	Low	Mail	4	9	9	37.0
...	...	...	...	...	...	...	...	...	...
16471	11283774362653	Support	Solved	Low	Mail	Offered	0	1	69.0
16472	7302298964893	Support	Solved	Low	Mail	Offered	0	3	90.0
16473	7303060862237	Support	Hold	Low	Mail	NaN	0	1	69.0
16474	10098421736733	Reimbursement Claims	Solved	Low	Mail	NaN	0	2	280.0
16475	11285001510813	Reimbursement Claims	Solved	Low	Mail	NaN	0	1	340.0

16476 rows × 15 columns

In [4]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16476 entries, 0 to 16475
Data columns (total 15 columns):
 #   Column                                             Non-Null Count  Dtype  
---  ------                                             --------------  -----  
 0   Requester id                                       16476 non-null  int64  
 1   Group                                              16476 non-null  object 
 2   Status                                             16476 non-null  object 
 3   Priority                                           16476 non-null  object 
 4   Via                                                16476 non-null  object 
 5   Satisfaction Score                                 5453 non-null   object 
 6   Reopens                                            16476 non-null  int64  
 7   Replies                                            16476 non-null  int64  
 8   First reply time in minutes within business hours  12760 non-null  float64
 9   First resolution time in minutes                   14834 non-null  float64
 10  First resolution time in minutes within business hours 14834 non-null  float64
 11  Full resolution time in minutes                    14409 non-null  float64
 12  Full resolution time in minutes within business hours  14409 non-null  float64
 13  Requester wait time in minutes                     16327 non-null  float64
 14  Requester wait time in minutes within business hours   16327 non-null  float64
dtypes: float64(7), int64(3), object(5)
memory usage: 1.9+ MB
```

Understanding Categorical data

In [5]:
```
data["Status"].value_counts()
```

Out[5]:
```
Closed     12270
Solved      2139
Hold         790
Pending      681
Open         470
New          126
Name: Status, dtype: int64
```

In [6]:
```
data["Priority"].value_counts()
```

Out[6]:
```
Low       16282
Urgent      182
Normal       12
Name: Priority, dtype: int64
```

In [7]:
```
data["Group"].value_counts()
```

Out[7]:
```
Endorsements            6449
Support                 5670
Reimbursement Claims    3827
Onboardings              530
Name: Group, dtype: int64
```

In [56]:
```
data["Via"].value_counts()
```

Out[56]:
```
Mail                      13081
Internal Communication     2028
OutBound                   1051
Closed Ticket               316
Name: Via, dtype: int64
```

In [8]:
```
data["Satisfaction Score"].value_counts()
```

Out[8]:
```
Offered    4640
5           409
4           311
3            64
1            15
2            14
Name: Satisfaction Score, dtype: int64
```

In [9]:
```
data["Satisfaction Score"].unique()
```

Out[9]:
```
array(['Offered', nan, '4', '1', '2', '3', '5'], dtype=object)
```

In [10]:
```
# Creating the target variable
data['Satisfaction Category'] = data['Satisfaction Score'].apply(lambda x: 'Offered' if
```

In [11]:
```
data
```

Out[11]:

	Requester id	Group	Status	Priority	Via	Satisfaction Score	Reopens	Replies	First reply time in minutes within business hours
0	10300555531549	Support	Solved	Low	Mail	Offered	11	11	173.0
1	10420228868125	Reimbursement Claims	Closed	Low	Mail	NaN	10	10	1527.0
2	10991633548957	Support	Solved	Low	Mail	4	10	11	61.0
3	10376247288477	Support	Solved	Low	Mail	Offered	9	9	381.0
4	7302858920989	Support	Closed	Low	Mail	4	9	9	37.0
...	...	...	...	...	...	...	...	...	...
16471	11283774362653	Support	Solved	Low	Mail	Offered	0	1	69.0
16472	7302298964893	Support	Solved	Low	Mail	Offered	0	3	90.0
16473	7303060862237	Support	Hold	Low	Mail	NaN	0	1	69.0
16474	10098421736733	Reimbursement Claims	Solved	Low	Mail	NaN	0	2	280.0
16475	11285001510813	Reimbursement Claims	Solved	Low	Mail	NaN	0	1	340.0

16476 rows × 16 columns

In [12]:
```
data["Satisfaction Category"].value_counts()
```

Out[12]:
```
Not Offered    11023
Offered         5453
Name: Satisfaction Category, dtype: int64
```

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [14]:
```
train_df, test_df, train_target, test_target = train_test_split(data.drop(columns=["Rec
```

Identifying Input and Target Columns

In [15]:
```
print("train_df.shape", train_df.shape)
print("test_df.shape", test_df.shape)
```

```
train_df.shape (13180, 14)
test_df.shape (3296, 14)
```

In [19]:
```
train_df
```

Out[19]:

	index	Group	Status	Priority	Via	Satisfaction Score	Reopens	Replies	First reply time in minutes within business hours	reso t m
0	14863	Endorsements	Closed	Low	Mail	NaN	0	1	3510.0	
1	33	Support	Solved	Low	Mail	4	6	8	72.0	
2	10872	Endorsements	Closed	Low	Mail	NaN	0	2	1603.0	
3	14364	Endorsements	Closed	Low	OutBound	NaN	0	0	NaN	
4	12239	Support	Closed	Low	Mail	Offered	0	2	181.0	
...	...	...	...	...	...	...	...	...	...	
13175	11284	Support	Closed	Low	Mail	Offered	0	1	192.0	
13176	11964	Endorsements	Closed	Low	Mail	NaN	0	2	1729.0	2
13177	5390	Support	Closed	Low	Mail	5	0	2	45.0	
13178	860	Support	Closed	Low	Mail	4	1	9	84.0	
13179	15795	Support	Hold	Low	Mail	NaN	0	1	20.0	

13180 rows × 15 columns

In [38]:
```
train_df.dropna(inplace=True)
test_df.dropna(inplace=True)
```

In [40]:
```
print("train_target Column: ", train_target)
print("test_target Column: ", test_target)
```

```
train_target Column:  14863    Not Offered
33           Offered
10872    Not Offered
14364    Not Offered
12239        Offered
            ...
11284        Offered
11964    Not Offered
5390         Offered
860          Offered
15795    Not Offered
Name: Satisfaction Category, Length: 13180, dtype: object
test_target Column:  12811    Not Offered
2816     Not Offered
10336    Not Offered
15619    Not Offered
2545     Not Offered
            ...
14092    Not Offered
753          Offered
8916         Offered
4359         Offered
12933    Not Offered
Name: Satisfaction Category, Length: 3296, dtype: object
```

Using `ColumnTransformer` to apply different preprocessing to different columns

In [41]:
```python
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
```

How to use `ColumnTransformer` ?

- select from DataFrame columns by name or by index
- passthrough or drop unspecified columns/ columns where preprocessing is not required

Q: Encode categorical features using OneHotEncoder or OrdinalEncoder

Two common ways to encode categorical features:

1. `OneHotEncoder` for unordered (nominal) data

2. `OrdinalEncoder` for ordered (ordinal) data

P.S. `LabelEncoder` is for labels, not features!

Q: For a one-hot encoded feature, what can you do if new data contains categories that weren't seen during training?

A: Set handle_unknown='ignore' to encode new categories as all zeros.

See example 👇

In [42]:
```
trf = ColumnTransformer(transformers=[
    # One Hot Encoding

    ("ohe_group", OneHotEncoder(sparse_output= False, handle_unknown="ignore", drop="fi
    # *Handle unknown categories with OneHotEncoder by encoding them as zeros*


    # Ordinal Encoding

    ("oe_status", OrdinalEncoder(categories=[['Solved', 'Closed', 'Hold', 'Pending', 'C
    ("oe_priority", OrdinalEncoder(categories=[['Low', 'Urgent', 'Normal']]), [2]),

    # One Hot Encoding

    ("ohe_via", OneHotEncoder(sparse_output= False, handle_unknown="ignore", drop="firs
    ('satisfaction_imputer', SimpleImputer(strategy='constant', fill_value='Not Mentior
    ('satisfaction_encoder', OrdinalEncoder(categories=[['Not Mentioned', '1', '2', '3'

    # Imputation Transformer

    ("imputing_first_reply_bh", SimpleImputer(strategy="mean"),[7]),
    ("imputing_first_resolution", SimpleImputer(strategy="mean"),[8]),
    ("imputing_first_resolution_bh", SimpleImputer(strategy="mean"),[9]),
    ("imputing_full_resolution", SimpleImputer(strategy="mean"),[10]),
    ("imputing_full_resolution_bh", SimpleImputer(strategy="mean"),[11]),
    ("imputing_requester_wait", SimpleImputer(strategy="mean"),[12]),
    ("imputing_requester_wait_bh", SimpleImputer(strategy="mean"),[13])
],remainder="passthrough")
```

In [43]:
```
trf.fit_transform(train_df)
```

Out[43]:
```
array([[0.0, 0.0, 0.0, ..., 3243.0, 6, 8],
       [1.0, 0.0, 0.0, ..., 232.0, 0, 2],
       [1.0, 0.0, 0.0, ..., 2091.0, 0, 4],
       ...,
       [1.0, 0.0, 0.0, ..., 2906.0, 0, 1],
       [1.0, 0.0, 0.0, ..., 88.0, 0, 2],
       [1.0, 0.0, 0.0, ..., 1331.0, 1, 9]], dtype=object)
```

In [47]:
```
trf
```

Out[47]:

ohe_group ▸ oe_status ▸ oe_priority ▸ ohe_via

OneHotEncoder ? (https://scikit-learn.org/1.4/modules/generated/...)

OrdinalEncoder ? (https://scikit-learn.org/1.4/modules/generated/...)

OrdinalEncoder ? (https://scikit-learn.org/1.4/modules/generated/...)

OneHotEncoder ? (http...

In [53]:
```
trf.transform(test_df)
```

Out[53]:
```
array([[1.0, 0.0, 1.0, ..., 61.0, 0, 2],
       [1.0, 0.0, 0.0, ..., 3333.0, 0, 4],
       [1.0, 0.0, 1.0, ..., 229.0, 0, 1],
       ...,
       [0.0, 0.0, 0.0, ..., 6467.0, 2, 5],
       [1.0, 0.0, 0.0, ..., 64.0, 0, 1],
       [1.0, 0.0, 0.0, ..., 1035.0, 0, 4]], dtype=object)
```

In [48]:
```
# Label Encoder (for Categorical Targets Only)
le = LabelEncoder()
le.fit(train_target)
```

Out[48]:

▾ LabelEncoder i ? (https://scikit-learn.org/1.4/modules/generated/sklearn.preprocessing.LabelEncoder.html)

LabelEncoder()

In [49]:
```
le.classes_
```

Out[49]:
```
array(['Not Offered', 'Offered'], dtype=object)
```

In [51]:
```
train_target = le.transform(train_target)
test_target = le.transform(test_target)
```

In [52]:
```
train_target
```

Out[52]:
```
array([0, 1, 0, ..., 1, 1, 0])
```

What is the difference between "fit" and "transform"?

-- `fit()` : Learn and estimate the parameters of the transformation

    * transformer.fit(train_data)

-- `transform()` : Apply the learned transformation to new data

    * transformed_data = estimator.transform(test_data)

-- `fit_transform()` : Learn the parameters and apply the transformation to new data

```
* transformed_data = estimator.fit_transform(data)
```

> > > > Note: Use "fit_transform" on training data, but "transform" (only) on testing/new data

> > *Want more tips? View all tips on GitHub.*

In [ ]: