Feature Engineering

Feature engineering is the process of creating new features from existing ones or transforming existing features to improve the performance of a machine learning model. It involves selecting, modifying, or creating features to enhance the model's ability to learn patterns from the data. Effective feature engineering can lead to better model performance, faster training times, and improved interpretability.

Feature Engineering

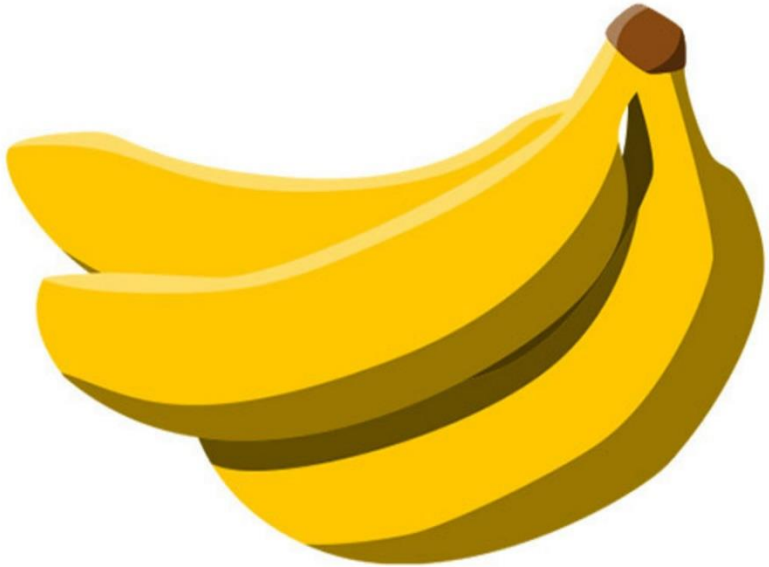
Feature engineering involves manipulating, selecting, or creating features (input variables) from raw data to improve model performance.

- **Transformation:** It includes transforming data through extraction, scaling, and other techniques to help models capture patterns and relationships.
- **Selection:** Choosing relevant features improves model simplicity, interpretability, and generalization by reducing noise and redundancy.
- **Creation:** New features can be engineered to capture important relationships or patterns not apparent in the original data.
- **Missing Data:** Handling missing values through techniques like imputation maintains data integrity.
- **Categorical Variables:** Encoding categorical variables as numerical values enables their use in machine learning algorithms.

Feature Engineering

- **Interaction Features:** Combining existing features to capture interactions between variables can enhance model performance.
- **Domain Expertise:** Feature engineering often requires domain knowledge to make informed decisions about creating meaningful features.
- **Impact:** Effective feature engineering can significantly enhance model accuracy and robustness.
- **Experimentation:** Iterative experimentation with different feature engineering techniques is key to finding optimal features for a given problem and dataset.

Feature Transformation



Feature Transformation

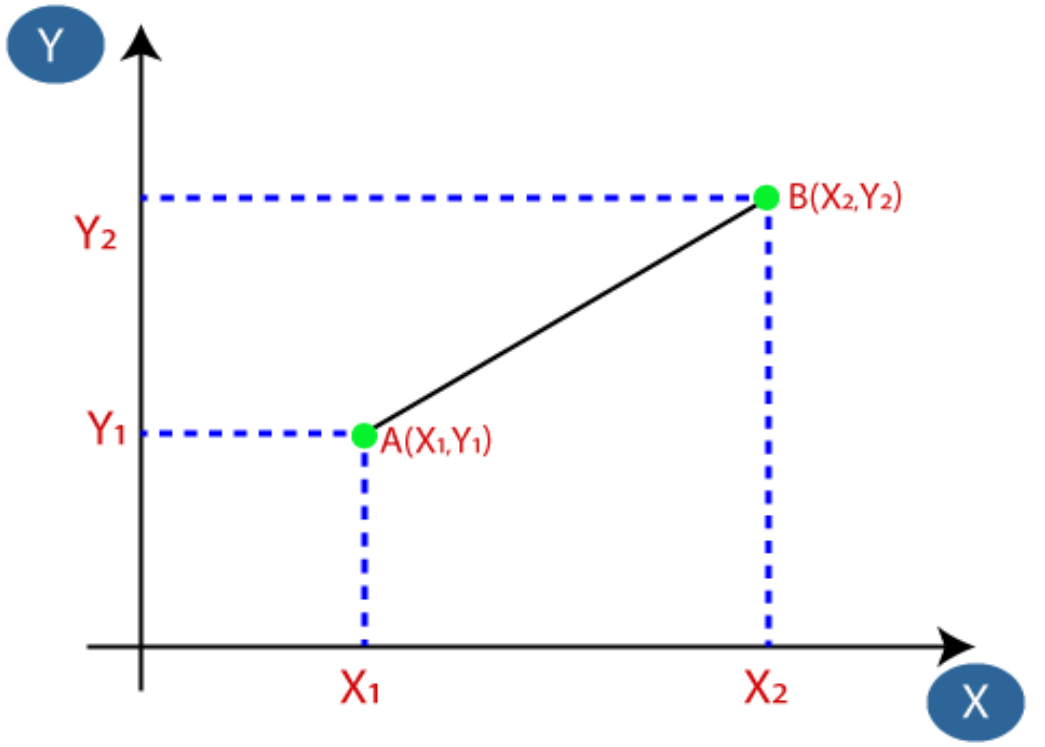
Transformation: It includes transforming data through extraction, scaling, and other techniques to help models capture patterns and relationships.

	cost	transform_cost
0	100	0.333333
1	200	0.666667
2	150	0.500000
3	300	1.000000
4	180	0.600000

Fig: transform data

Feature Transformation

The Euclidean Distance between two points is calculated using a simple formula.



Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

Feature Transformation

The Manhattan Distance between two points is calculated using a simple formula.

Manhattan Distance = $\left|x_1 - x_2\right| + \left|y_1 - y_2\right|$

Manhattan Distance = $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$

A few advantages of **feature scaling** the data are as follows:

- ❖ It makes your training faster.
- ❖ It prevents you from getting stuck in local optima.
- ❖ It gives you a better error surface shape.

However, there are few algorithms such as Tree-based algorithms and probability-based algo. that are not affected by the scaling of input data.

Feature Transformation

- **Tree-Based Models:** Feature scaling has minimal impact on models like Random Forests and Gradient Boosted Trees.
- **Neural Networks with Batch Normalization:** Modern neural networks can adapt to feature scales during training, especially with batch normalization.
- **Sparse Models:** Feature scaling may not significantly affect models designed for sparse data.
- **Models Robust to Outliers:** Robust models are less sensitive to the scale of features.
- **Ordinal or Count Data:** Features representing ordinal or count data may not need standardization.
- **Sparse Data with Feature Interactions:** Feature scaling may have limited influence when predictive power relies on feature interactions.
- **Feature Engineering Adequacy:** Additional feature transformations may not significantly improve model performance if existing features capture patterns effectively.

Feature Transformation

Examples of Algorithms where Feature Scaling matters:

- **K-Means** uses the Euclidean distance measure here to feature scaling matters.
- **K-Nearest – Neighbours** also require feature scaling.
- **Principal Component Analysis (PCA)**: Tries to get the feature with maximum variance, here feature scaling is required.
- **Normalization for Gradient Descent:** In optimization algorithms like gradient descent, feature scaling and normalization can significantly impact convergence speed.

Note: Naïve Bayes, Decision Tree, Random Forest & and All tree-based models are not affected by feature scaling.

Feature Transformation

Techniques to perform Feature Transformation:

- Normalization
- Standardization
- Log Transformation
- Robust Scaler
- Max Absolute Scaler

Feature Transformation

	Student	CGPA	Salary '000
0	1	3.0	60
1	2	3.0	40
2	3	4.0	40
3	4	4.5	50
4	5	4.2	52

Normalization: $$X_{new} = \frac{X_i - \min(X)}{\max(x) - \min(X)}$$

Python Implementation:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

Feature Transformation

	Student	CGPA	Salary '000
0	1	3.0	60
1	2	3.0	40
2	3	4.0	40
3	4	4.5	50
4	5	4.2	52

Standardization: $X_{new} = \frac{X_i - X_{mean}}{\text{Standard Deviation}}$

Standard Deviation: $\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$

σ = population standard deviation

N = the size of the population

x_i = each value from the population

μ = the population mean

Python Implementation:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

Feature Transformation

The log transform can be applied as follows:

1. Check if the feature has any zero or negative values. If so, consider using a modified version of the log transform (e.g., adding a constant value or using the logarithm of the absolute values).

2. Add a small constant value (e.g., 5) to the feature before applying the logarithm. This is done to avoid taking the logarithm of zero or close-to-zero values, which would result in undefined or infinite values.

3. Apply the natural logarithm function (base e) to each value of the feature.

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.FunctionTransformer.html

In simplest terms, the Max Absolute Scaler takes the absolute maximum value of each column and divides each value in the column by the maximum value.

Formula: $$xscaled = \frac{x}{max(x)}$$

Python Implementation:

```
from sklearn.preprocessing import MaxAbsScaler
scaler = MaxAbsScaler()
```

Robust Scaler are robust to outliers. It is used to scale the feature to median and quantiles Scaling using median and quantiles consists of subtracting the median to all the observations, and then dividing by the interquartile difference. The interquartile difference is the difference between the 75th and 25th quantile:

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

- IQR = 75th quantile - 25th quantile
- RobustScaler= (Xi – X.Median)/IQR

Python Implementation:

```
from sklearn.preprocessing import RobustScaler
RoSc=RobustScaler()
```

Video: https://youtu.be/U9N-ELpCpc8

Let’s do it with PYTHON