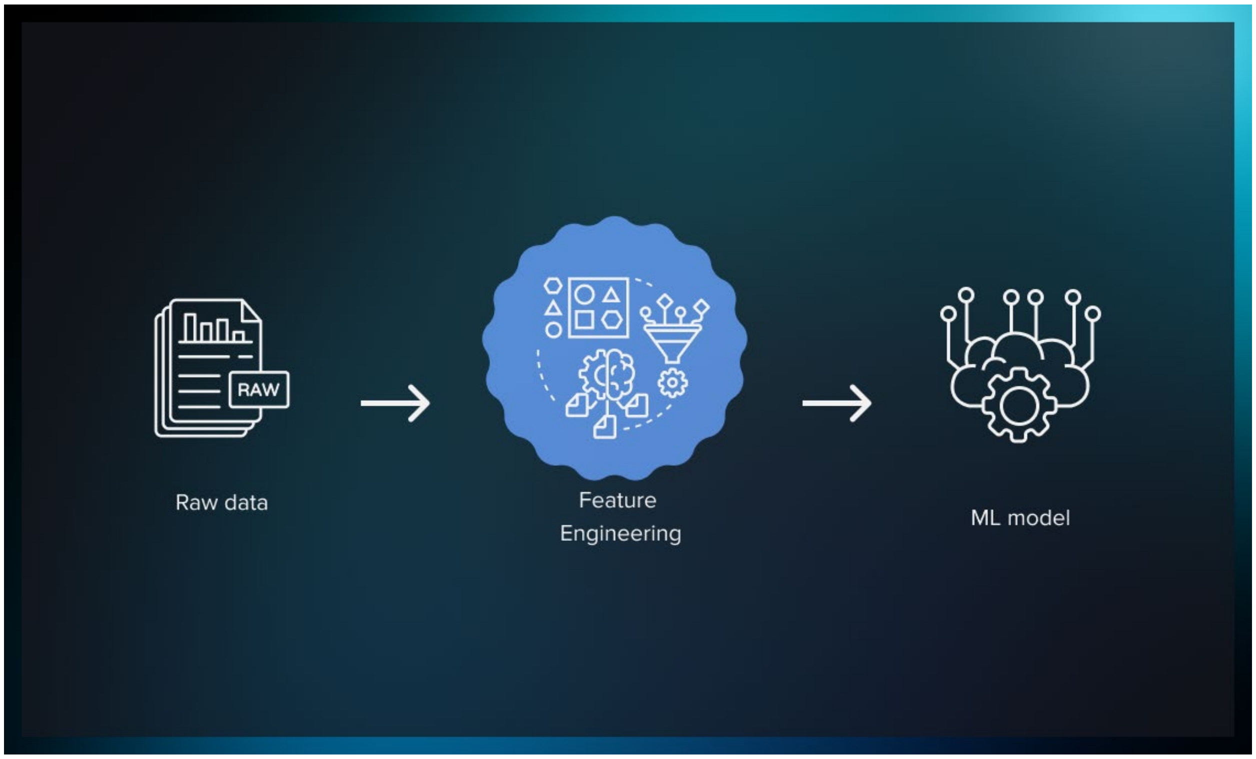
FEATURE ENGINEERING

TEAM MEMBERS

ABHIJITH
ANN MARYA
ANSIL
ILLIYAS
MUHAMMED ANSHEER
SHERIN
YADU

FEATURE ENGINEERING

Feature Engineering is the process of **creating new features or transforming existing features to improve the performance of a machine-learning model**. It involves **selecting relevant information from raw data and transforming it into a format that can be easily understood by a model**. The goal is to improve model accuracy by providing more meaningful and relevant information.



Processes Involved in Feature Engineering

Feature engineering in Machine learning consists of mainly 5 processes: **Feature Creation, Feature Transformation, Feature Extraction, Feature Selection, and Feature Scaling.** It is an iterative process that requires experimentation and testing to find the best combination of features for a given problem. The success of a machine learning model largely depends on the quality of the features used in the model.

1 Feature Creation

Feature Creation is the process of **generating new features based on domain knowledge or by observing patterns** in the data. It is a form of feature engineering that can significantly improve the performance of a machine-learning model.

1.1 Types of Feature Creation:

1.1.1 Domain-Specific:

Creating new features based on domain knowledge, such as **creating features based on business rules or industry standards**. Here are some examples of domain-specific feature creation in various fields:

1.1.1.1 Healthcare:

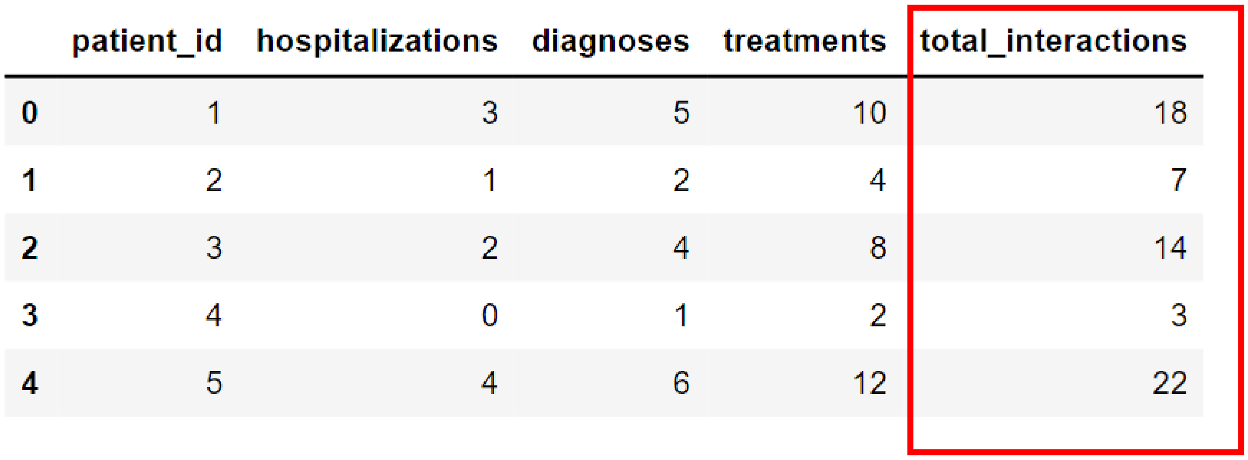
1.1.1.1.1 Patient History:

Create features based on a patient's medical history, such as the number of previous hospitalizations, diagnoses, or treatments.

Example:

This example demonstrates how to create new features based on a patient's medical history, including the number of previous hospitalizations, diagnoses, and treatments. These features provide insights into the patient's medical journey and complexity.

```
df['total_interactions'] = df['hospitalizations'] + df['diagnoses'] + df['treatments']
```



	patient_id	hospitalizations	diagnoses	treatments	total_interactions
0	1	3	5	10	18
1	2	1	2	4	7
2	3	2	4	8	14
3	4	0	1	2	3
4	5	4	6	12	22

Usage:

Useful for understanding the overall health history of patients and their interactions with the healthcare system.

1.1.1.1.2 Vital Sign Trends:

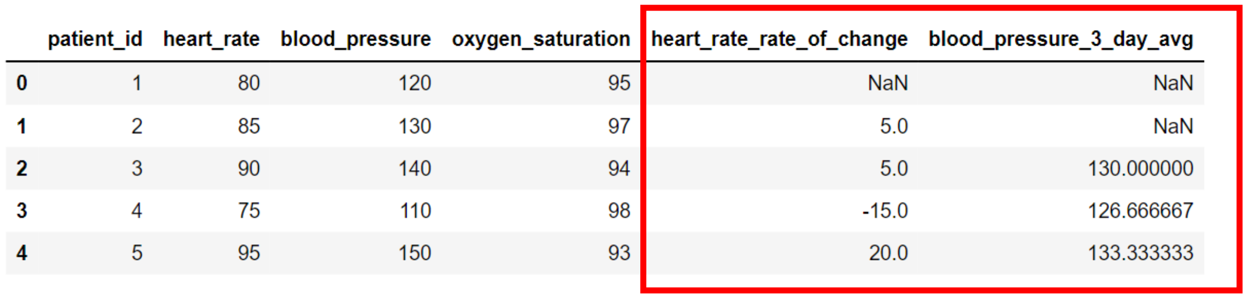
Calculate features that capture trends and changes in vital signs over time.

Example:

This example calculates and generates features capturing trends and changes in vital signs (heart rate, blood pressure, and oxygen saturation) over time. Features like rate of change and moving averages help identify patterns and anomalies in patient vitals.

```
df['heart_rate_rate_of_change'] = df['heart_rate'].diff()
```

```
df['blood_pressure_3_day_avg'] = df['blood_pressure'].rolling(window=3).mean()
```



	patient_id	heart_rate	blood_pressure	oxygen_saturation	heart_rate_rate_of_change	blood_pressure_3_day_avg
0	1	80	120	95	NaN	NaN
1	2	85	130	97	5.0	NaN
2	3	90	140	94	5.0	130.000000
3	4	75	110	98	-15.0	126.666667
4	5	95	150	93	20.0	133.333333

Usage:

Valuable for monitoring patients' health conditions and identifying sudden changes or gradual trends in vital signs.

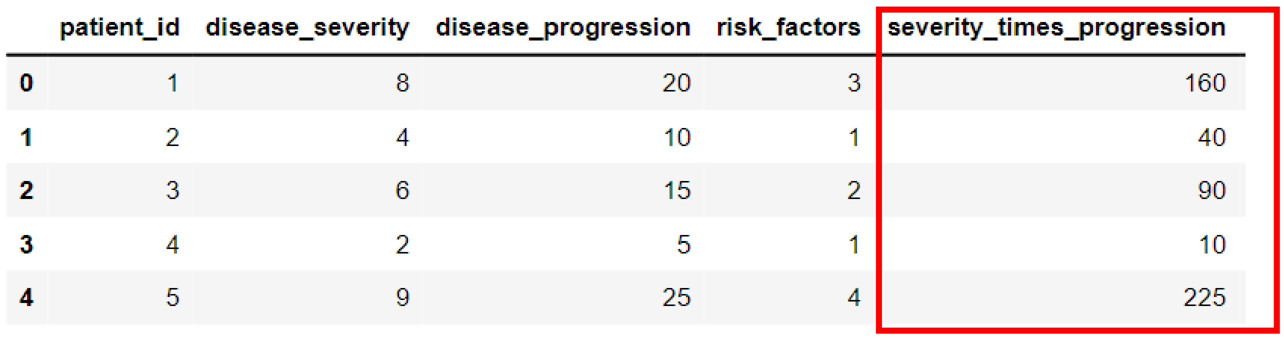
1.1.1.1.3 Disease-Specific Metrics:

For specific diseases, generate features that measure disease severity, progression, or risk factors.

Example:

This code snippet showcases how to create disease-specific features, such as disease severity, progression, and risk factors. By combining these metrics, you can generate features that provide insights into disease severity, progression, and overall risk for each patient.

```
# Calculate disease severity * disease progression as a feature
df['severity_times_progression'] = df['disease_severity'] * df['disease_progression']
```



	patient_id	disease_severity	disease_progression	risk_factors	severity_times_progression
0	1	8	20	3	160
1	2	4	10	1	40
2	3	6	15	2	90
3	4	2	5	1	10
4	5	9	25	4	225

Usage:

Helpful for predicting disease outcomes, understanding disease dynamics, and assessing patient risk factors.

1.1.1.2 Finance:

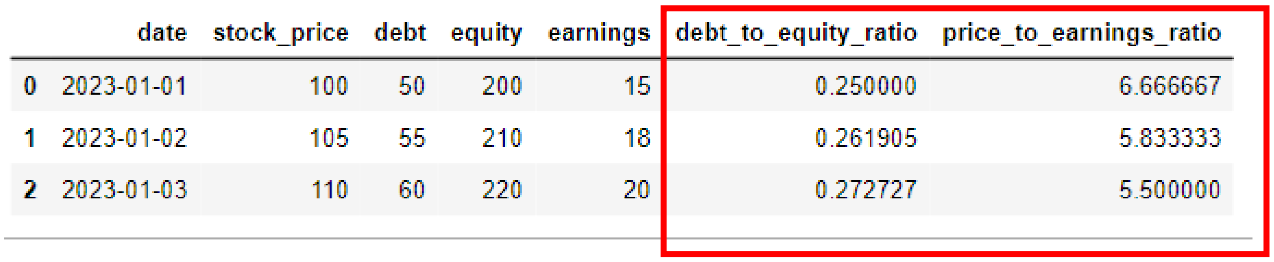
1.1.1.2.1 Financial Ratios:

Compute ratios like debt-to-equity ratio, price-to-earnings ratio, and return on investment as features for predicting stock prices or investment outcomes.

Example:

Financial ratios provide insights into a company's financial health and performance. Ratios like debt-to-equity ratio, price-to-earnings ratio, and return on investment can serve as features for predicting stock prices or investment outcomes.

```
# Create financial ratios as features

df['debt_to_equity_ratio'] = df['debt'] / df['equity']

df['price_to_earnings_ratio'] = df['stock_price'] / df['earnings']
```



	date	stock_price	debt	equity	earnings	debt_to_equity_ratio	price_to_earnings_ratio
0	2023-01-01	100	50	200	15	0.250000	6.666667
1	2023-01-02	105	55	210	18	0.261905	5.833333
2	2023-01-03	110	60	220	20	0.272727	5.500000

Usage:

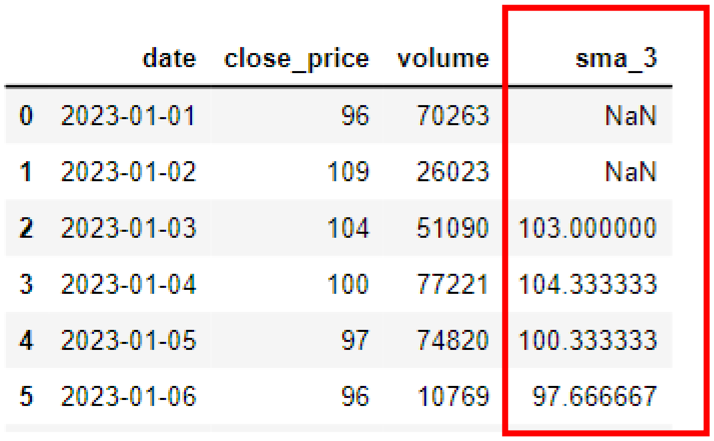
Financial ratios provide insights into a company's financial health and performance. They can be used as features in stock price prediction models, investment decision-making, and risk assessment. For instance, a low debt-to-equity ratio might indicate lower financial risk, while a high price-to-earnings ratio could suggest an overvalued stock.

1.1.1.2.2 Trading Indicators:

Generate technical indicators like moving averages, relative strength index (RSI), and moving average convergence divergence (MACD).

Example:

Technical indicators are mathematical calculations based on historical price and volume data. Examples include moving averages, relative strength index (RSI), and moving average convergence divergence (MACD). These indicators can provide insights into market trends and momentum.



	date	close_price	volume	sma_3
0	2023-01-01	96	70263	NaN
1	2023-01-02	109	26023	NaN
2	2023-01-03	104	51090	103.000000
3	2023-01-04	100	77221	104.333333
4	2023-01-05	97	74820	100.333333
5	2023-01-06	96	10769	97.666667

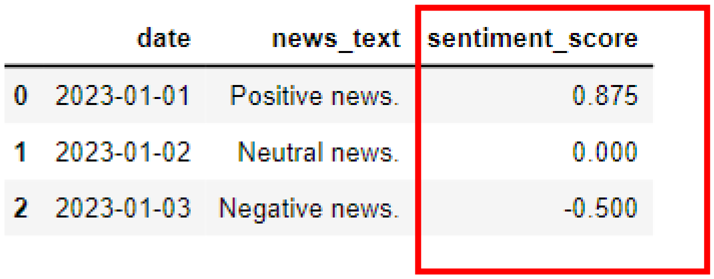
1.1.1.2.3 Market Sentiment:

Incorporate sentiment scores from financial news or social media to capture market sentiment as a feature.

Example:

Incorporating sentiment scores from financial news or social media can help capture market sentiment as a feature. Sentiment analysis can gauge the overall positive or negative sentiment toward a particular asset, which could influence price movements.

```
# Perform sentiment analysis and create sentiment feature

df['sentiment_score'] = df['news_text'].apply(lambda x: TextBlob(x).sentiment.polarity)
```



	date	news_text	sentiment_score
0	2023-01-01	Positive news.	0.875
1	2023-01-02	Neutral news.	0.000
2	2023-01-03	Negative news.	-0.500

Usage:

Market sentiment features derived from sentiment analysis of news or social media can provide additional context for decision-making. Positive sentiment might be associated with optimism and potential price increases, while negative sentiment could suggest caution and potential price declines. These sentiment scores can complement other features in predicting market movements.

1.1.1.3 E-Commerce:

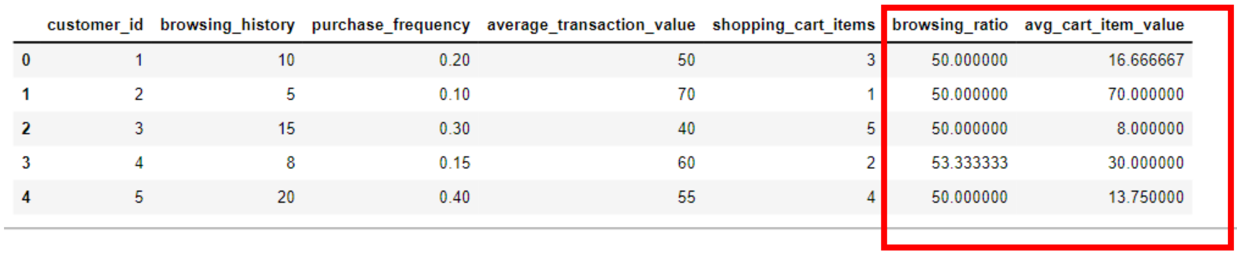
1.1.1.3.1 Customer Behavior:

Create features related to customer browsing history, purchase frequency, average transaction value, and shopping cart contents.

Example:

Customer behavior features provide insights into how customers interact with an e-commerce platform. These features can help understand browsing habits, purchase behavior, and transaction patterns.

```
# Create customer behavior features

df['browsing_ratio'] = df['browsing_history'] / df['purchase_frequency']

df['avg_cart_item_value'] = df['average_transaction_value'] / df['shopping_cart_items']
```



	customer_id	browsing_history	purchase_frequency	average_transaction_value	shopping_cart_items	browsing_ratio	avg_cart_item_value
0	1	10	0.20	50	3	50.000000	16.666667
1	2	5	0.10	70	1	50.000000	70.000000
2	3	15	0.30	40	5	50.000000	8.000000
3	4	8	0.15	60	2	53.333333	30.000000
4	5	20	0.40	55	4	50.000000	13.750000

Usage: Customer behavior features provide insights into how customers interact with your e-commerce platform. These features can be used to personalize recommendations, optimize marketing campaigns, and tailor user experiences. For instance, understanding the browsing-to-purchase ratio can help identify customers who are more likely to convert.

1.1.1.3.2 Product Features:

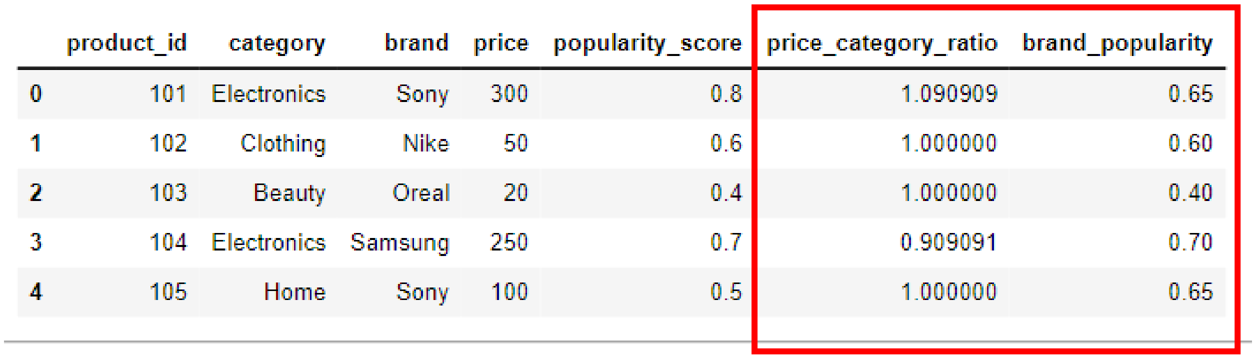
Generate features based on product attributes like category, brand, price range, and popularity.

Example:

Product features are derived from attributes associated with each product. These attributes can include product category, brand, price, and popularity, which help capture product characteristics and trends.

```
# Create product features

df['price_category_ratio'] = df['price'] /df.groupby('category')['price'].transform('mean')
```

```
df['brand_popularity'] = df.groupby('brand')['popularity_score'].transform('mean')
```



	product_id	category	brand	price	popularity_score	price_category_ratio	brand_popularity
0	101	Electronics	Sony	300	0.8	1.090909	0.65
1	102	Clothing	Nike	50	0.6	1.000000	0.60
2	103	Beauty	Oreal	20	0.4	1.000000	0.40
3	104	Electronics	Samsung	250	0.7	0.909091	0.70
4	105	Home	Sony	100	0.5	1.000000	0.65

Usage:

Product features capture attributes and characteristics of items in your e-commerce catalog. These features are valuable for personalized product recommendations, understanding brand preferences, and pricing strategies. For example, the price-to-category ratio can highlight whether a product's price is above or below the average for its category.

1.1.1.4 Manufacturing and Industrial Processes:

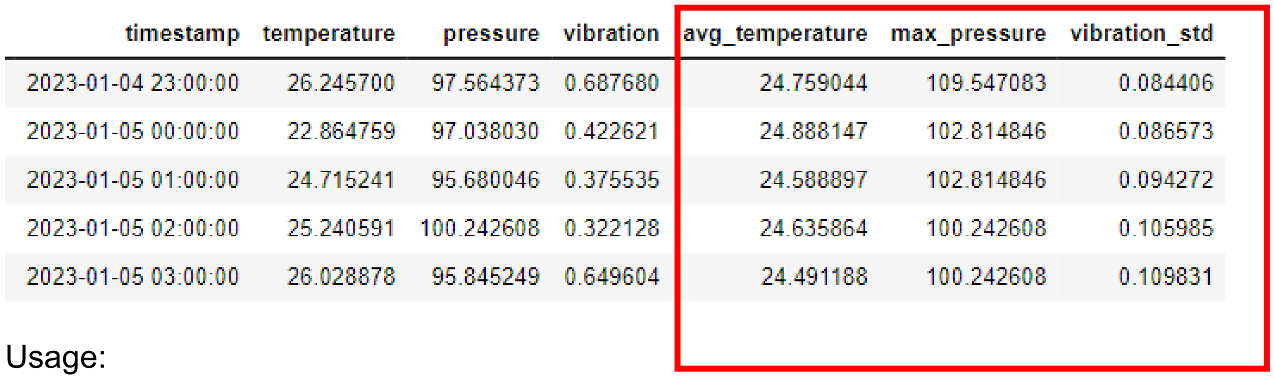
1.1.1.4.1 Sensor Data:

Generate features from sensor measurements, capturing variables like temperature, pressure, and vibration.

Example:

Generate features like average temperature, maximum pressure, and standard deviation of vibration readings over a specific time window. These features can be used for predictive maintenance, where deviations from normal sensor values could indicate impending equipment issues.

```
# Create sensor data features

df['avg_temperature'] = df['temperature'].rolling(window=10).mean()

df['max_pressure'] = df['pressure'].rolling(window=5).max()

df['vibration_std'] = df['vibration'].rolling(window=15).std()
```



timestamp	temperature	pressure	vibration	avg_temperature	max_pressure	vibration_std
2023-01-04 23:00:00	26.245700	97.564373	0.687680	24.759044	109.547083	0.084406
2023-01-05 00:00:00	22.864759	97.038030	0.422621	24.888147	102.814846	0.086573
2023-01-05 01:00:00	24.715241	95.680046	0.375535	24.588897	102.814846	0.094272
2023-01-05 02:00:00	25.240591	100.242608	0.322128	24.635864	100.242608	0.105985
2023-01-05 03:00:00	26.028878	95.845249	0.649604	24.491188	100.242608	0.109831

Usage:

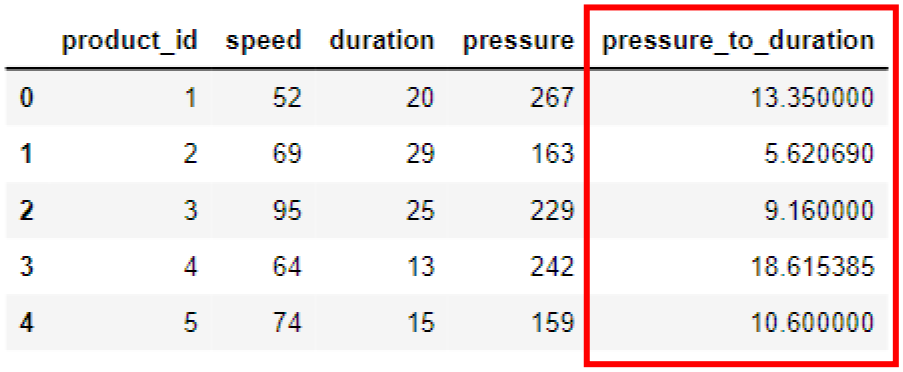
Sensor data features are essential for monitoring the health and performance of machinery and equipment. By creating features from sensor measurements such as temperature, pressure, and vibration, you can gain insights into the operational condition of machines.

1.1.1.4.2 Process Parameters:

Create features based on parameters used during manufacturing processes, such as speed, duration, and pressure levels.

Example:

Calculate the ratio of pressure to duration for different manufacturing processes. If higher pressure-to-duration ratios consistently lead to lower defect rates, this could indicate an optimal parameter setting for quality improvement.



	product_id	speed	duration	pressure	pressure_to_duration
0	1	52	20	267	13.350000
1	2	69	29	163	5.620690
2	3	95	25	229	9.160000
3	4	64	13	242	18.615385
4	5	74	15	159	10.600000

Usage:

Process parameter features are valuable for process optimization and quality control. By creating features based on parameters like speed, duration, and pressure levels used during manufacturing, you can assess their impact on the final product.

1.1.1.4.3 Anomaly Detection:

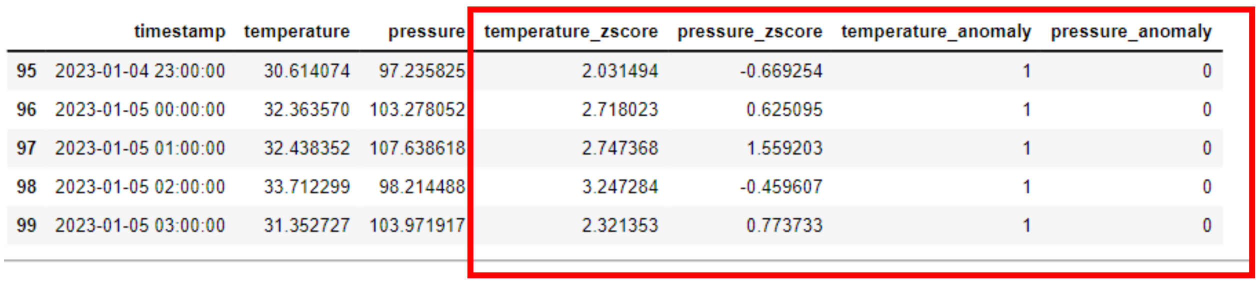
Develop features that detect anomalies in the production process, highlighting deviations from expected patterns.

Example:

Create features that compare the current sensor values to historical data or predefined thresholds. If any feature exceeds the established limits, it triggers an alert, helping operators address potential issues early and reduce scrap or rework.

```
# Create anomaly detection features

df['temperature_zscore'] = (df['temperature'] - df['temperature'].mean()) /
df['temperature'].std()

df['pressure_zscore'] = (df['pressure'] - df['pressure'].mean()) / df['pressure'].std()
```

```
# Detect anomalies using z-score threshold

anomaly_threshold = 2

df['temperature_anomaly'] = df['temperature_zscore'].apply(lambda x: 1 if abs(x) >
anomaly_threshold else 0)

df['pressure_anomaly'] = df['pressure_zscore'].apply(lambda x: 1 if abs(x) >
anomaly_threshold else 0)
```



	timestamp	temperature	pressure	temperature_zscore	pressure_zscore	temperature_anomaly	pressure_anomaly
95	2023-01-04 23:00:00	30.614074	97.235825	2.031494	-0.669254	1	0
96	2023-01-05 00:00:00	32.363570	103.278052	2.718023	0.625095	1	0
97	2023-01-05 01:00:00	32.438352	107.638618	2.747368	1.559203	1	0
98	2023-01-05 02:00:00	33.712299	98.214488	3.247284	-0.459607	1	0
99	2023-01-05 03:00:00	31.352727	103.971917	2.321353	0.773733	1	0

Usage:

Anomaly detection features are critical for maintaining product quality and preventing defects. By developing features that detect anomalies in production processes, you can identify deviations from expected patterns and take corrective action.

1.1.1.5 Social Media and User Behavior:

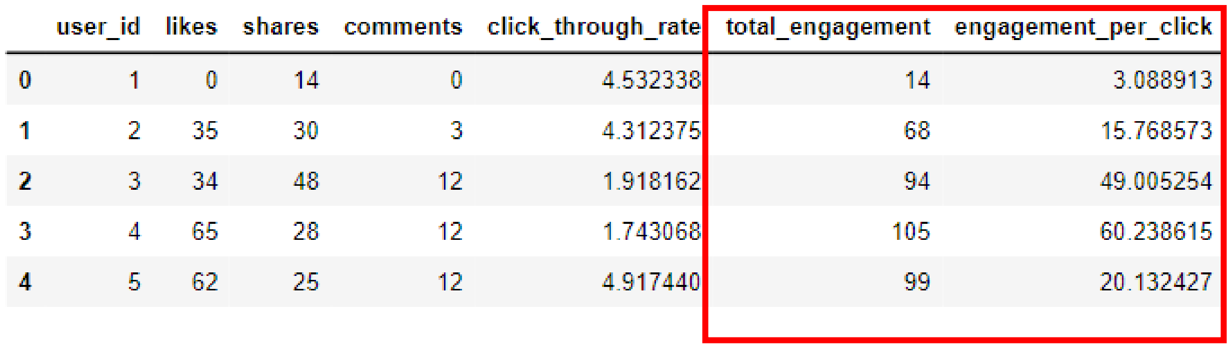
1.1.1.5.1 Engagement Metrics:

Create features based on user engagement metrics like likes, shares, comments, and click-through rates.

Example:

Engagement metrics features involve creating data-driven measurements based on user interactions with social media content. These metrics provide insights into user engagement, interactions, and the effectiveness of content.

```
# Create engagement metrics features

df['total_engagement'] = df['likes'] + df['shares'] + df['comments']

df['engagement_per_click'] = df['total_engagement'] / df['click_through_rate']
```



	user_id	likes	shares	comments	click_through_rate	total_engagement	engagement_per_click
0	1	0	14	0	4.532338	14	3.088913
1	2	35	30	3	4.312375	68	15.768573
2	3	34	48	12	1.918162	94	49.005254
3	4	65	28	12	1.743068	105	60.238615
4	5	62	25	12	4.917440	99	20.132427

Usage:

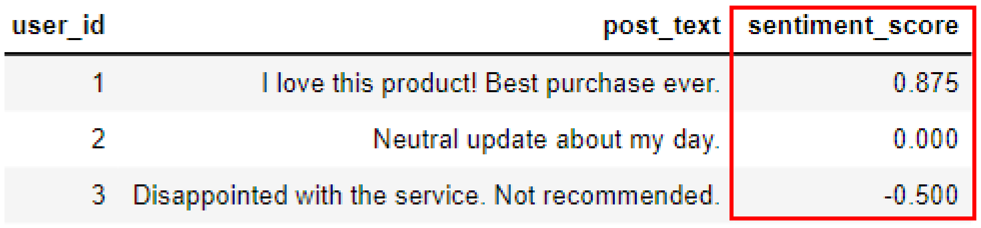
Engagement metrics features allow social media managers to quantify user interaction and content effectiveness. Total engagement reveals which posts generate the most interactions, while engagement per click indicates how well content converts clicks into meaningful actions. These insights help refine content strategies and optimize campaigns for maximum user engagement.

1.1.1.5.2 Sentiment Analysis:

Generate sentiment features based on user-generated content to capture user opinions and emotions.

Example:

Sentiment analysis features involve assessing the sentiment expressed in user-generated content, such as posts or comments. Sentiment scores indicate whether the content carries a positive, neutral, or negative sentiment.



user_id	post_text	sentiment_score
1	I love this product! Best purchase ever.	0.875
2	Neutral update about my day.	0.000
3	Disappointed with the service. Not recommended.	-0.500

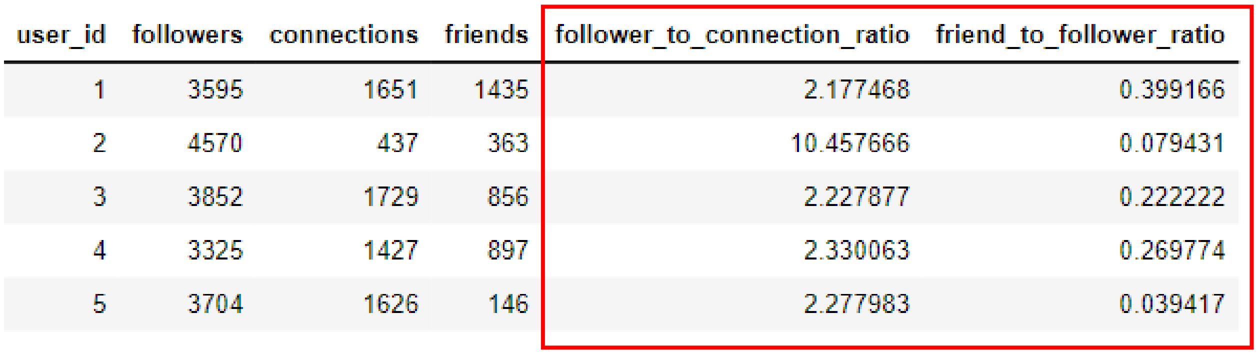
Usage: Sentiment analysis features empower brands to understand user opinions and emotions. Positive sentiment can identify successful campaigns, while negative sentiment can alert to issues. Businesses can tailor marketing strategies based on user sentiment, address concerns promptly, and enhance brand perception.

1.1.1.5.3 Network Features:

Capture network-related features like the number of connections, followers, or friends in social networks.

Example:

Network features revolve around user relationships within social networks. They provide insights into user influence, connectivity, and the dynamics of their connections.



user_id	followers	connections	friends	follower_to_connection_ratio	friend_to_follower_ratio
1	3595	1651	1435	2.177468	0.399166
2	4570	437	363	10.457666	0.079431
3	3852	1729	856	2.227877	0.222222
4	3325	1427	897	2.330063	0.269774
5	3704	1626	146	2.277983	0.039417

Usage:

Network features enable brands to identify influential users and understand their relationships. The follower-to-connection ratio points out users with high engagement relative to their network size, making them potential influencers. The friend-to-follower ratio helps comprehend user networking patterns and potential brand advocates, guiding influencer collaborations and targeting strategies.

1.1.2 Data-Driven:

Creating new features by observing patterns in the data, such as calculating aggregations or creating interaction features.

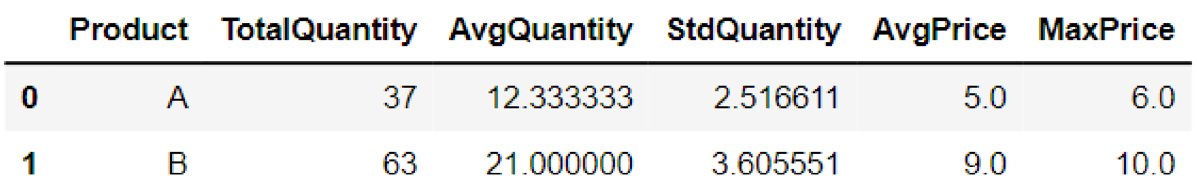
1.1.2.1 Aggregation Features:

Calculating summary statistics like mean, median, minimum, maximum, and standard deviation over groups of data points. This is particularly useful for summarizing information within categories or time intervals.

Aggregating features can **provide insights into the distribution and central tendencies of data within specified groups**, helping models understand patterns.

1.1.2.1.1 Example:

Suppose you're working with an e-commerce dataset containing information about customer purchases. You can create aggregation features by calculating the average purchase amount, maximum purchase amount, and total number of purchases for each customer. This can help capture the spending behavior of customers over time.



	Product	TotalQuantity	AvgQuantity	StdQuantity	AvgPrice	MaxPrice
0	A	37	12.333333	2.516611	5.0	6.0
1	B	63	21.000000	3.605551	9.0	10.0

1.1.2.1.2 Usage:

Suppose you're building a recommendation system for an e-commerce platform. You've created aggregation features representing average purchase amounts for each customer. You can use these features to model customer behavior and preferences, helping your recommendation system suggest products that align with their historical spending patterns.

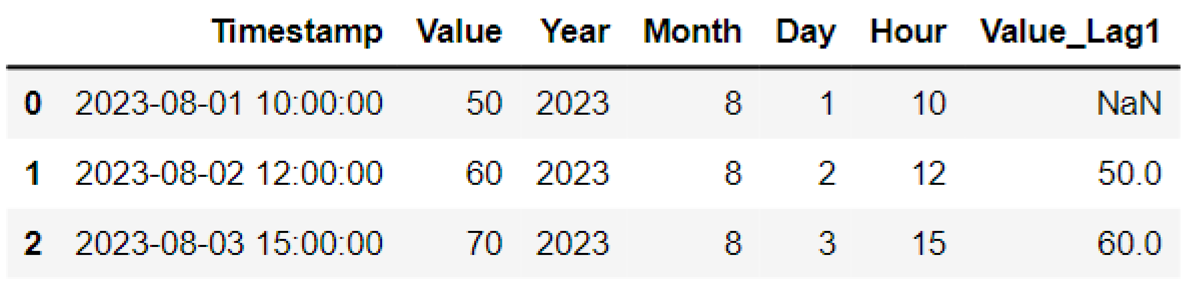
1.1.2.2 Temporal Features:

Extracting **temporal information from timestamps, such as day of the week, month, quarter, or time of day.**

Capturing seasonality and periodic patterns in time-series data, which can be crucial for forecasting and understanding cyclic behavior.

1.1.2.2.1 Example:

In a retail sales dataset, you can extract temporal features by converting timestamp information. For example, you could create a feature indicating the day of the week when a purchase was made. This feature might help capture differences in purchasing patterns on weekends versus weekdays.

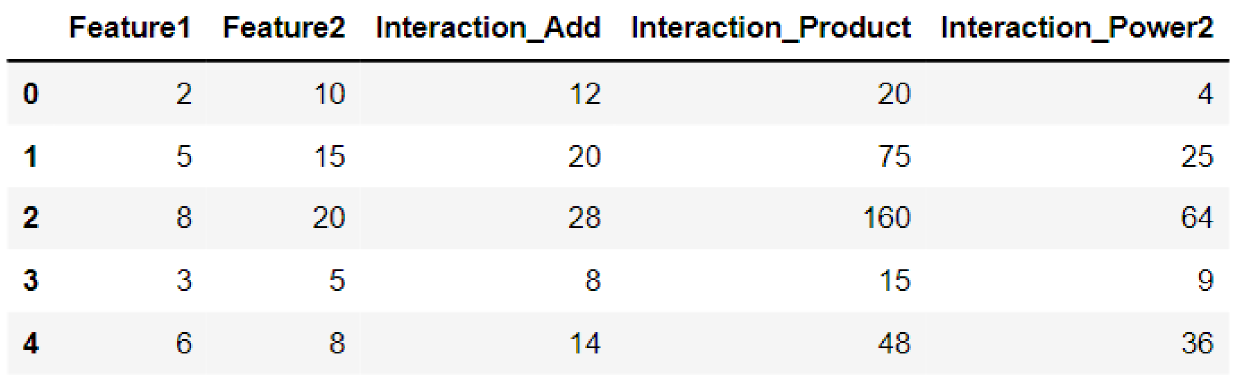


	Timestamp	Value	Year	Month	Day	Hour	Value_Lag1
0	2023-08-01 10:00:00	50	2023	8	1	10	NaN
1	2023-08-02 12:00:00	60	2023	8	2	12	50.0
2	2023-08-03 15:00:00	70	2023	8	3	15	60.0

1.1.2.2.2 Usage:
Imagine you're working on a demand forecasting model for a retail store. You've generated temporal features indicating the day of the week when purchases were made. These features can be crucial for capturing weekly purchasing trends, helping your model accurately predict customer demand for specific days.

1.1.2.3 Interaction Features:
Combining two or more features through mathematical operations (e.g., multiplication, division, addition) to capture relationships between them.

This subcategory can help models learn complex interactions that might not be apparent from individual features.



	Feature1	Feature2	Interaction_Add	Interaction_Product	Interaction_Power2
0	2	10	12	20	4
1	5	15	20	75	25
2	8	20	28	160	64
3	3	5	8	15	9
4	6	8	14	48	36

1.1.2.3.1 Example:
Imagine you're analyzing housing data and have features for both the number of bedrooms and the square footage of a house. By creating an interaction feature that multiplies the number of bedrooms by the square footage, you can capture the potential impact of the house's size on its desirability.

1.1.2.3.2 Usage:
In a real estate price prediction task, you've engineered an interaction feature by multiplying the number of bedrooms by the square footage of a house. This interaction feature can help the model capture the potential impact of both factors on the house price, providing a more nuanced understanding of the relationship.

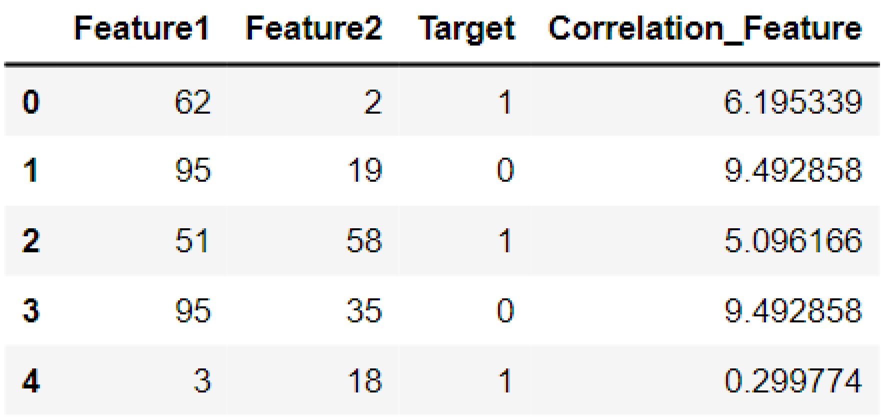
1.1.2.4 Correlation-based Features:

Creating new features that quantify the relationships between pairs of variables, often using correlation coefficients like Pearson's correlation or Spearman's rank correlation.

These features can provide insight into how strongly or weakly two variables are related and whether their relationship changes over time or across groups.

1.1.2.4.1 Example:

Suppose you're working with a financial dataset containing stock market data. You can create correlation-based features by calculating the correlation coefficient between two different stock prices. This can help your model understand how changes in one stock might relate to changes in another.



	Feature1	Feature2	Target	Correlation_Feature
0	62	2	1	6.195339
1	95	19	0	9.492858
2	51	58	1	5.096166
3	95	35	0	9.492858
4	3	18	1	0.299774

1.1.2.4.2 Usage:

Suppose you're analyzing stock market data and predicting the movement of a particular stock. The correlation-based feature you've created between two stocks can serve as an additional input to your model, helping it capture potential dependencies between the two stocks' behaviors.

1.1.2.5 Outlier Features:

Generating binary features that indicate the presence or absence of outliers in a dataset.

Outlier features can help models differentiate between normal and anomalous data points, which is valuable for anomaly detection tasks.

1.1.2.5.1 Example:

In a sensor data dataset monitoring machinery performance, you could create a binary outlier feature based on measurements that exceed a certain threshold. This feature could indicate whether a data point is an outlier (1) or not (0), which can be valuable for detecting equipment malfunctions.

1.1.2.5.2 Usage:
For a predictive maintenance scenario in manufacturing, you've incorporated an outlier feature indicating whether a sensor reading is considered an outlier. This feature can be used to enhance the accuracy of your model's predictions by accounting for anomalous data points that might signify potential equipment failures.

1.1.3 **Synthetic:**
Generating new features by **combining existing features or synthesizing new data points.**

1.1.3.1 Arithmetic Operation Features:
1.1.3.1.1 Example:
In a manufacturing process, you have 'raw_material_cost' and 'labor_cost'. You can create a 'total_cost' feature by adding these two costs together.

1.1.3.1.2 Usage:
The 'total_cost' feature provides an overall picture of production expenses, which can aid in cost analysis and decision-making.

1.1.3.2 Polynomial Expansion Features:
1.1.3.2.1 Example:
For a dataset with 'temperature' readings, you can create 'temperature_squared' and 'temperature_cubed' features to capture potential nonlinear relationships with other variables.

1.1.3.2.2 Usage:
These features might be beneficial in climate modeling, where temperature effects aren't strictly linear.

1.1.3.3 Feature Crosses:
1.1.3.3.1 Example:
Consider an e-commerce dataset with 'item_category' and 'customer_segment' features. You can create a 'category_segment' feature by combining these two, indicating preferences for specific items within customer segments.

1.1.3.3.2 Usage:
This feature can guide personalized product recommendations based on past preferences within each segment.

1.1.3.4 Scaling-based Features:

1.1.3.4.1 Example:

In a survey with 'income' and 'age' features, scaling 'income' to a common range (e.g., 0 to 1) ensures both features contribute equally to clustering or regression algorithms.

1.1.3.4.2 Usage:

This feature scaling prevents income from dominating the age feature in distance-based calculations.

1.1.3.5 Embedding-based Features:

1.1.3.5.1 Example:

In a social network analysis, you can use node embeddings generated from graph data to capture community structures and relationships between individuals.

1.1.3.5.2 Usage:

These embeddings can assist in identifying influential nodes or clusters in a network.

1.1.3.6 Time-Series Transformation Features:

1.1.3.6.1 Example:

For a stock price dataset, you can create a feature representing the difference between the current price and the price from a week ago.

1.1.3.6.2 Usage:

This feature can help capture weekly price trends and short-term fluctuations for trading strategies.

1.1.3.7 Categorical Interaction Features:

1.1.3.7.1 Example:

In a hotel booking system, you might cross 'destination_type' and 'booking_channel' to create a feature capturing preferred booking channels for specific destination types.

1.1.3.7.2 Usage:

This interaction feature can guide marketing strategies by tailoring channel promotions to destination preferences.

1.1.3.8 Text Length Features:

1.1.3.8.1 Example:

In sentiment analysis of product reviews, creating a 'review_length' feature (word count or character count) can help differentiate between concise and detailed feedback.

1.1.3.8.2 Usage:

Longer reviews might provide more valuable insights into customer opinions.

1.1.3.9 Bin Count Features:

1.1.3.9.1 Example:

In an online shopping dataset, you can create features representing the count of purchases falling into different price range bins.

1.1.3.9.2 Usage:

These bin count features can help identify price preferences and target specific price ranges with promotions.

1.1.3.10 Rate Calculation Features:

1.1.3.10.1 Example:

For a website analytics dataset, calculating the 'bounce rate' by dividing the number of one-page visits by the total visits provides insight into page engagement.

1.1.3.11 Usage:

Bounce rate helps assess the effectiveness of website content and layout.

1.1.3.12 **Benefits of Feature Creation:**

- **Improves Model Performance:** By providing additional and more relevant information to the model, feature creation can increase the accuracy and precision of the model.
- **Increases Model Robustness:** By adding additional features, the model can become more robust to outliers and other anomalies.
- **Improves Model Interpretability:** By creating new features, it can be easier to understand the model’s predictions.
- **Increases Model Flexibility:** By adding new features, the model can be made more flexible to handle different types of data.

2 Feature Transformation

Feature Transformation is the process of **transforming the features into a more suitable representation for the machine learning model**. This is done to ensure that the model can effectively learn from the data.

2.1 Types of Feature Transformation:

2.1.1 Normalization:

Rescaling the features to have a similar range, such as between 0 and 1, to prevent some features from dominating others.

2.1.2 Scaling:

Rescaling the features to have a similar scale, such as having a standard deviation of 1, to make sure the model considers all features equally.

2.1.3 Encoding:

Transforming categorical features into a numerical representation. Examples are one-hot encoding and label encoding.

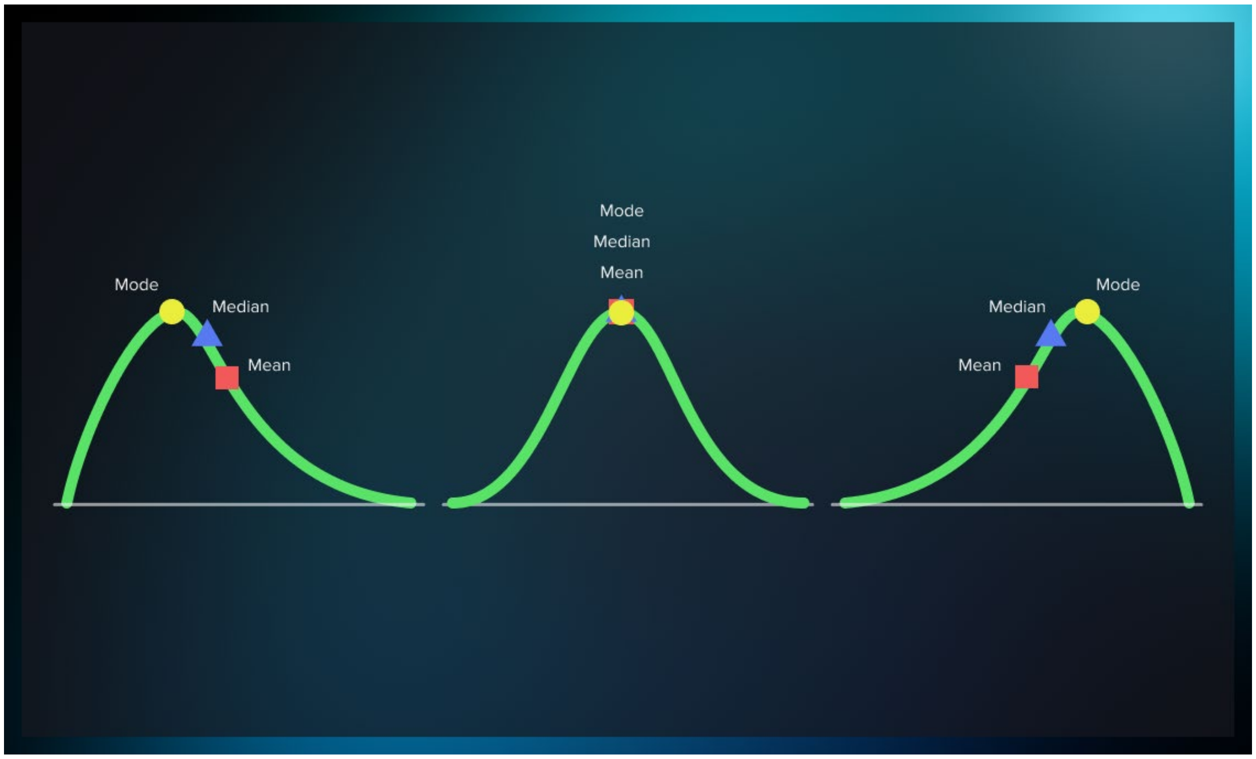
2.1.4 Transformation:

Transforming the features using mathematical operations to change the distribution or scale of the features. Examples are logarithmic, square root, and reciprocal transformations.

There are 3 types of Feature transformation techniques:

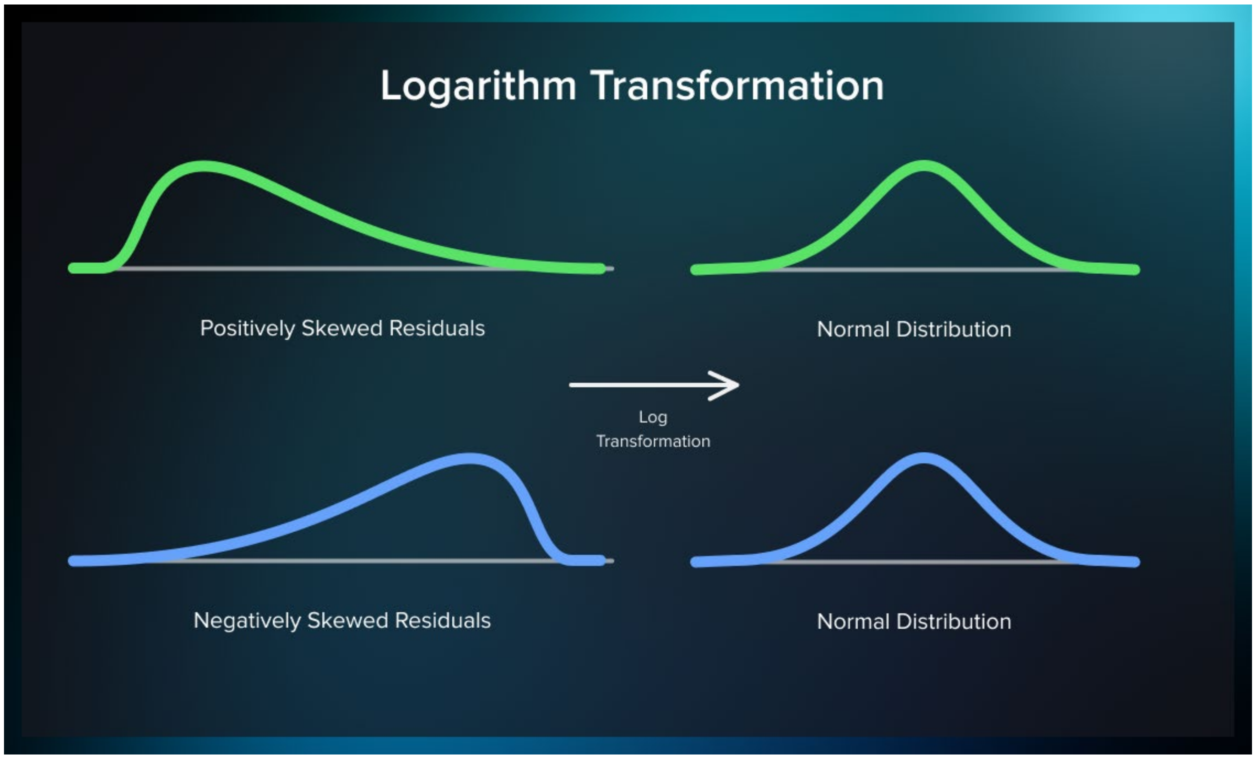
2.1.4.1 Function Transformers

Function transformers are the type of feature transformation technique that uses a particular **function to transform the data to the normal distribution.** Here the particular function is applied to the data observations.



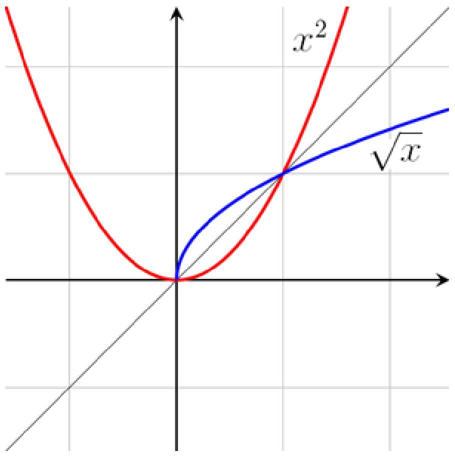
2.1.4.1.1 Log Transform

Log transform is one of the simplest transformations on the data in which the **log is applied to every single distribution** of the data and the result from the log is considered the final day to feed the machine learning algorithms.



2.1.4.1.2 Square Transform

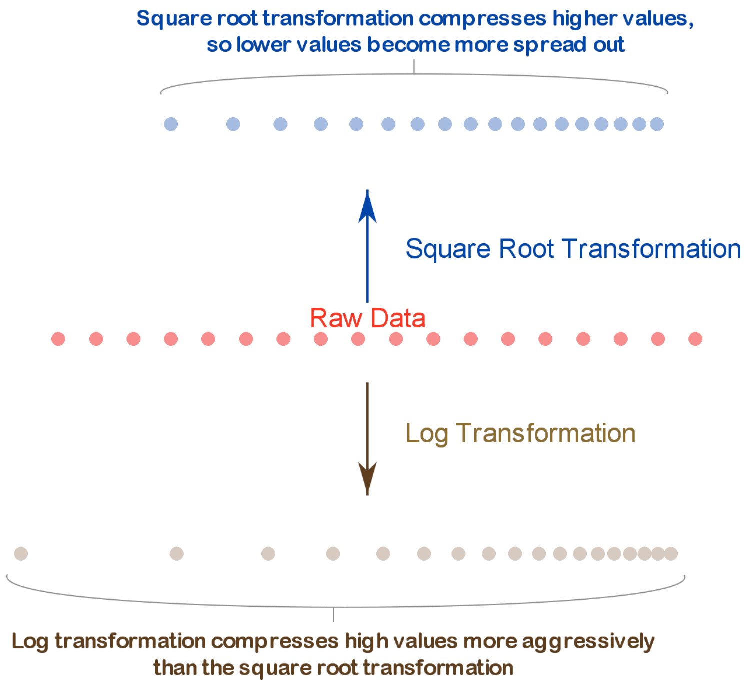
Square transform is the type of transformer in which the **square of the data is considered instead of the normal data.** In simple words, in this transformed the data is applied with the square function, where the square of every single observation will be considered as the final transformed data.



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

2.1.4.1.3 Square Root Transform

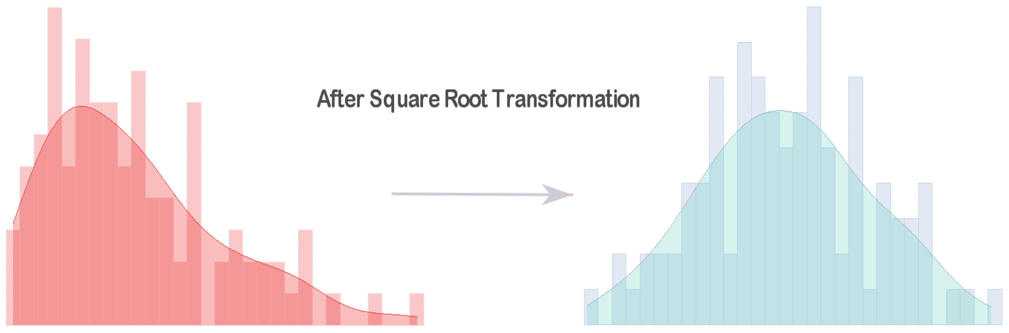
In this transform, the **square root of the data is calculated**. This transform performs so well on the left-skewed data and efficiently transformed the left-skewed data into normally distributed data.


Square root transformation compresses higher values, so lower values become more spread out
Square Root Transformation
Raw Data
Log Transformation
Log transformation compresses high values more aggressively than the square root transformation

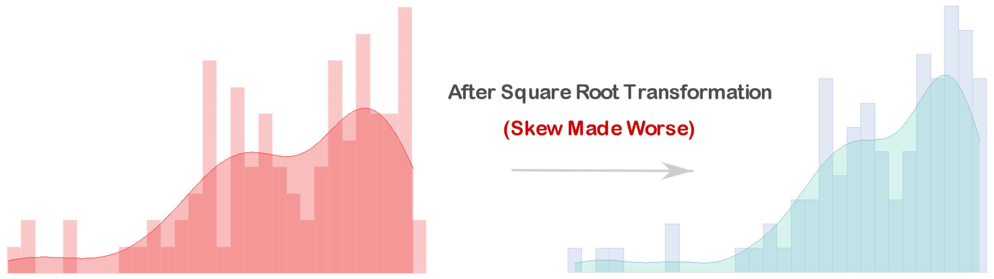
A square root transformation can be useful for:

1. Normalizing a skewed distribution:

   If your variable has a right skew, you can try a square root transformation in order to normalize it.



The square root transformation will not fix all skewed variables. **Variables with a left skew, for instance, will become worst after a square root transformation**. As discussed above, this is a consequence of compressing high values and stretching out the ones on the lower end.

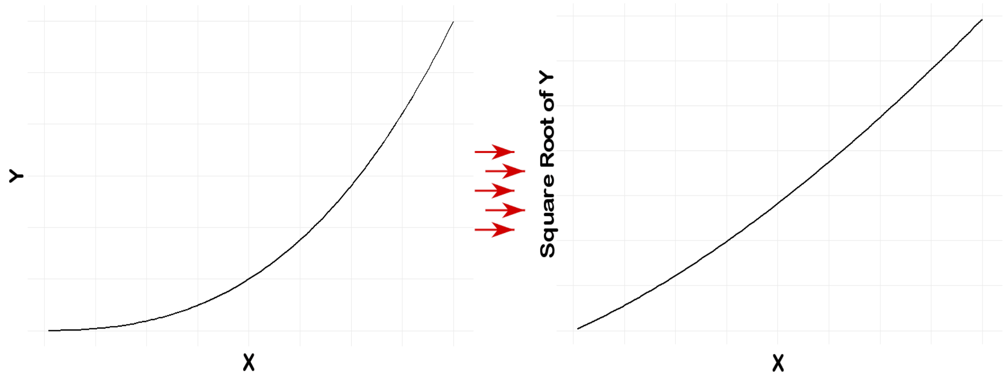


In order to normalize left skewed distributions, you can try a quadratic, cube or exponential transformation..

2. Transforming a non-linear relationship between 2 variables into a linear one

   When running a linear regression, the most important assumption is that the dependent and independent variable have a linear relationship.

   One solution to fix a non-linear relationship between X and Y, is to try a log or square root transformation.



After the transformation, the relationship looks linear enough to run a linear regression

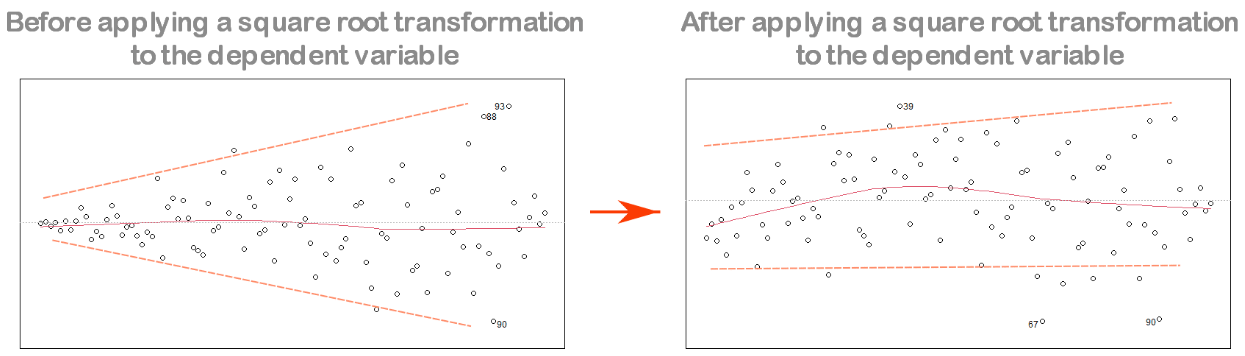
3. Reducing heteroscedasticity of the residuals in linear regression

   Another assumption of linear regression is that the residuals should have equal variance (often referred to as homoscedasticity of the residuals).

When the plot of residuals versus fitted values shows a funnel shape (as seen in the left-hand plot of the figure below), it is considered a sign of non-equal variance of the residuals (i.e. a sign of heteroscedasticity).

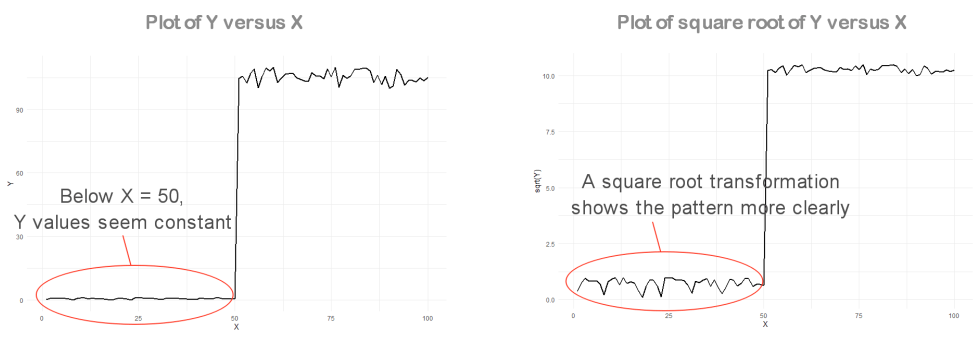
To address this issue, one solution is to use a logarithmic or square root transformation on the outcome Y:

Plots showing residuals versus fitted values:



4. Focusing on visualizing certain parts of your data:

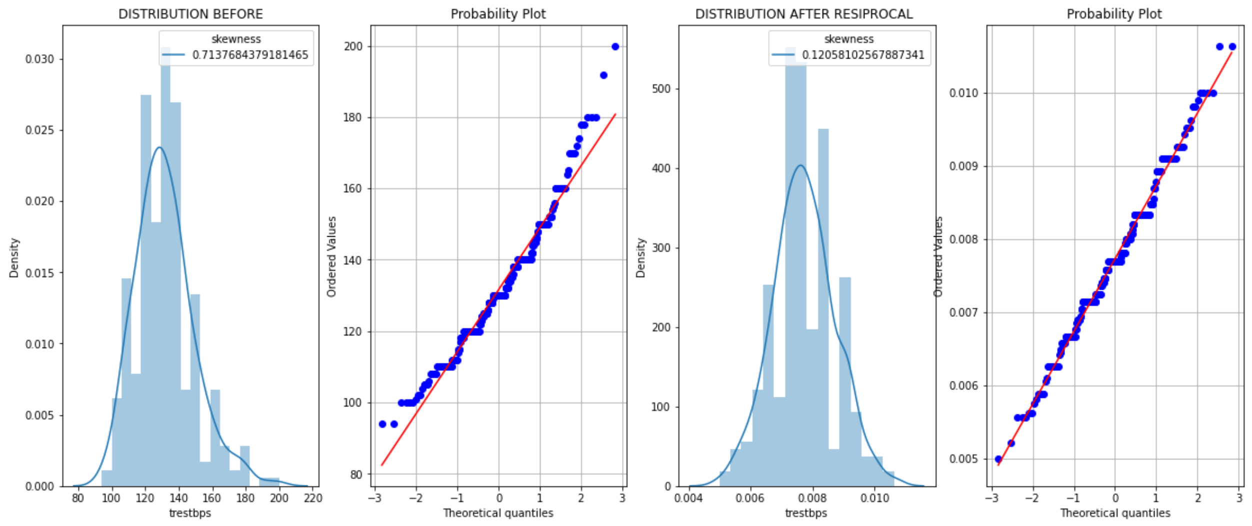
Some variables will inherently have very low and very high values when measured at different times. This will make them a little bit harder to visualize in a single plot.
An example of such variable is the blood CRP (C-Reactive Protein), which is a marker of inflammation in the body. Its normal range is less than 6 mg/L. But in case of bacterial infection, it could be as high as 10 times the normal value (so > 60 mg/L).
Below is an example of a variable Y plotted against X. As you may have noticed in the left-hand plot, low values of Y cannot be distinguished from one another and appear to be constant below X = 50.
In this case, a square root transformation may help visualize the pattern of these lower values of Y (right-hand plot).



2.1.4.1.4 Reciprocal Transform

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.

In this transformation, x will replace by the inverse of x (1/x). The reciprocal transformation will give little effect on the shape of the distribution. This transformation can be only used for non-zero values.



In the above plot There is Distribution of before applying Resiprocal transformation and after applying Resiprocal Transformation. As we can **see the Skewness got reduced.**

2.1.4.1.5 Custom Transform

In every dataset, the log and square root transforms can not be used, as every data can have different patterns and complexity. Based on the domain knowledge of the data, custom transformations can be applied to transform the data into a normal distribution. **The custom transforms here can be any function or parameter like sin, cos, tan, cube, etc.**

2.1.4.2 Power Transformers

Power Transformation techniques are the type of feature transformation technique where the **power is applied to the data observations for transforming the data.**

There are two types of Power Transformation techniques:

2.1.4.2.1 Box-Cox Transform:

This transform technique is mainly used for transforming the data observations by applying power to them. The power of the data observations is denoted by Lambda(λ). There are mainly two conditions associated with the power in this transform, which is lambda equals zero and not equal to zero. The mathematical formulation of this transform is as follows:

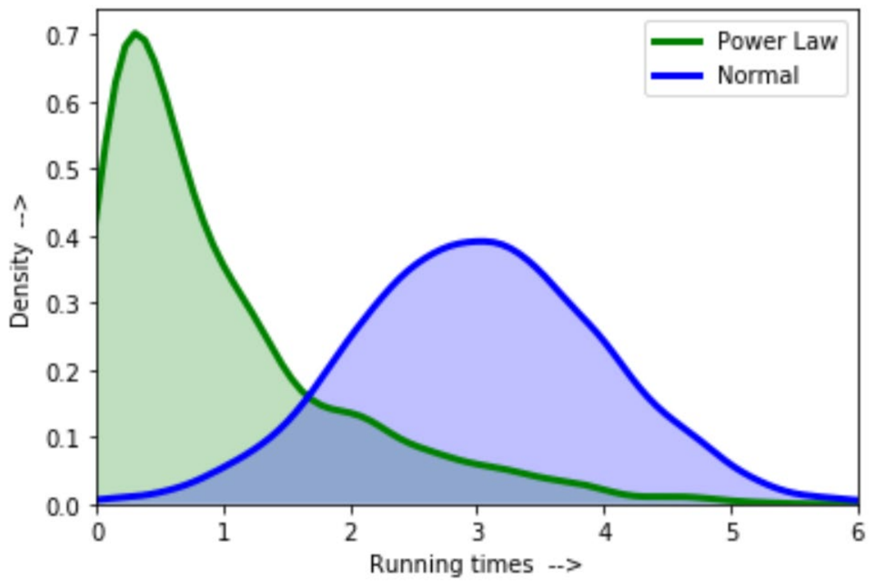
$$X_i^\lambda = \begin{cases} \ln X_i & ; \quad \text{for } \lambda = 0 \\ \frac{X_i^\lambda - 1}{\lambda} & ; \quad \text{for } \lambda \neq 0 \end{cases}$$

Here the lambda is the power applied to every data observation. Based upon the iteration technique every single value of the lambda is examined and the best fit value of the lambda is then applied to the data to transform it.

Here the transformed value of every data observation will lie between 5 to -5. One major disadvantage associated with this transformation technique is that this technique can only be applied to positive observations. it is not applicable for negative and zero values of the data observations.

Example:

Imagine you are watching a horse race and like any other race, there are fast runners and slow runners. So, logically speaking, the horse which came first and the fast horses along with it will have the smaller difference of completion time whereas the slowest ones will have a larger difference in their completion time.



Here in our example, there is an inconsistent variance (Heteroscedasticity) between the fast horses and the slow horses because there will be small variations for shorter completion time and vice versa.

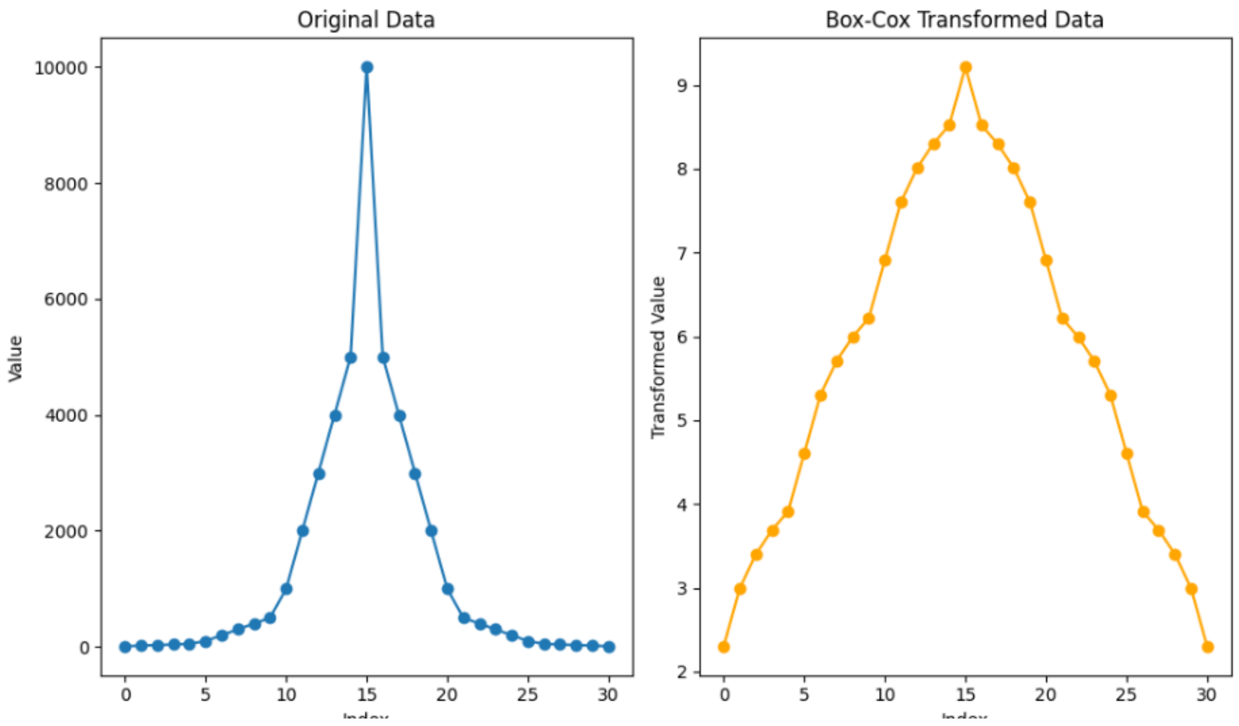
Hence, the distribution for our data will not be a bell curve or normally distributed as there will be a longer tail on the right side. These types of distributions follow **Power law** or **80-20 rule** where the relative change in one quantity varies as the power of another.

Mathematics behind Box-Cox Transformation:

When a log transformation is applied to non-normal distribution, it tries to expand the differences between the smaller values because the slope for the logarithmic function is steeper for smaller values whereas the differences between the larger values can be reduced because, for large values, log distribution has a moderate slope.

$$X_i^{\lambda} = \begin{cases} \ln X_i & ; \quad \text{for } \lambda = 0 \\ \frac{X_i^{\lambda}-1}{\lambda} & ; \quad \text{for } \lambda \neq 0 \end{cases}$$

Box-cox Transformation only cares about computing the value of   which varies from – 5 to 5. A value of   is said to be best if it is able to approximate the non-normal curve to a normal curve. This function **requires input to be positive**.



2.1.4.2.2 Yeo-Johnson Transform

This transformation technique is also a power transform technique, where the power of the data observations is applied to transform the data. **This is an advanced form of a box cox transformations technique where it can be applied to even zero and negative values of data observations also.**

The mathematical formulations of this transformations technique are as follows:

$$y(\lambda) = \begin{cases} \left((y+1)^{\lambda} - 1\right)/\lambda, & \text{if } y \geq 0 \text{ and } \lambda \neq 0 \\ -\left((-y+1)^{2-\lambda} - 1\right)/(2-\lambda), & \text{if } y < 0 \text{ and } \lambda \neq 2 \\ \log(y+1), & \text{if } \lambda = 0 \text{ and } y \geq 0 \\ -\log(-y+1), & \text{if } \lambda = 0 \text{ and } y < 0 \end{cases}$$

1. **Mathematics behind Yeo-Johnson Transformation:**

   - When λ=0:

     the Yeo-Johnson transformation behaves like a logarithmic transformation, and it can expand the differences between smaller values and reduce the differences between larger values.

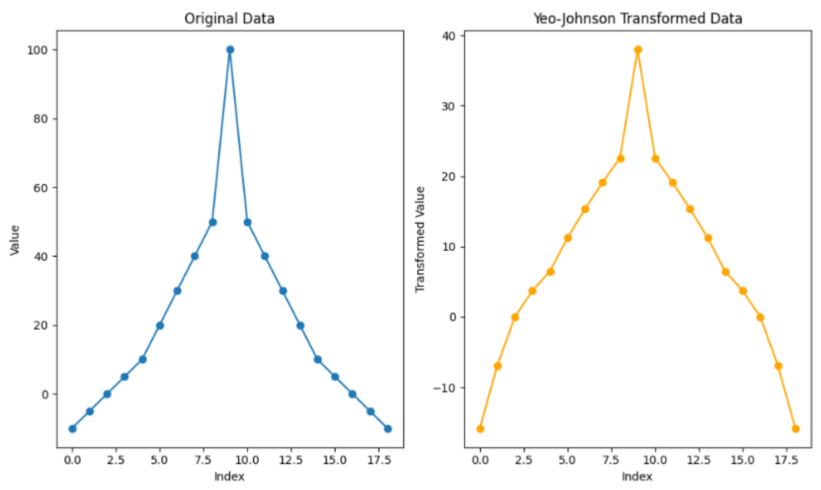
   - When λ=1:

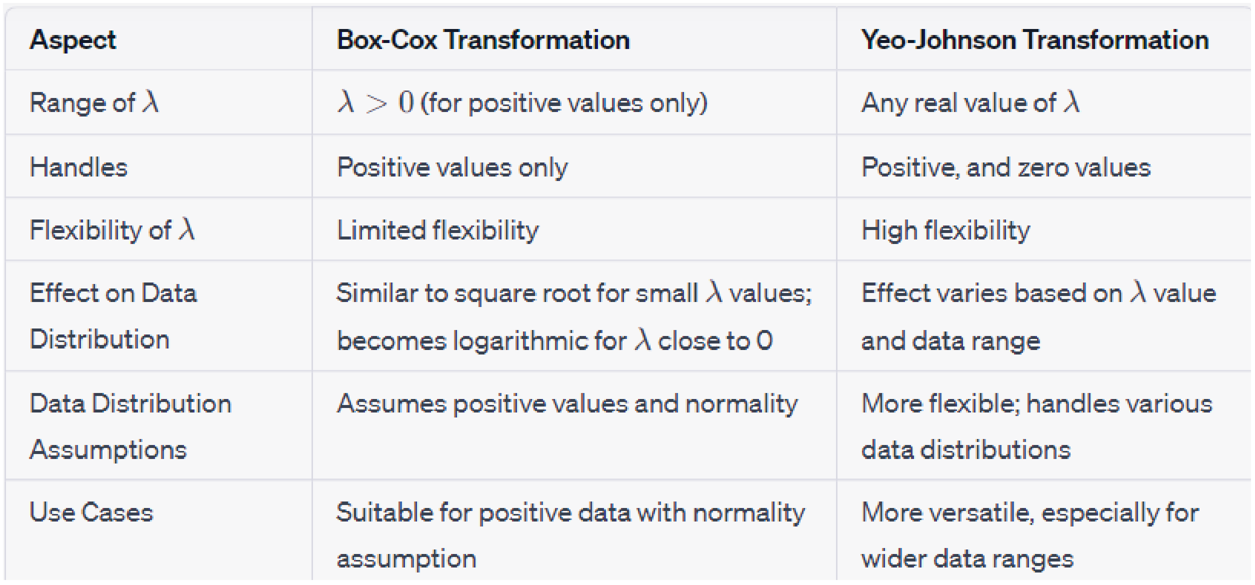
     the Yeo-Johnson transformation is a linear transformation that does not change the data's distribution significantly.

   - When λ=2
     the Yeo-Johnson transformation behaves like a inverse square root transformation. which can help stabilize variance and normalize the data.

   - When λ takes values between 0 and 1 or between 1 and 2:

     the Yeo-Johnson transformation introduces non-linearity that can have varying effects on different value ranges.





Aspect	Box-Cox Transformation	Yeo-Johnson Transformation
Range of λ	$\lambda > 0$ (for positive values only)	Any real value of λ
Handles	Positive values only	Positive, and zero values
Flexibility of λ	Limited flexibility	High flexibility
Effect on Data Distribution	Similar to square root for small λ values; becomes logarithmic for λ close to 0	Effect varies based on λ value and data range
Data Distribution Assumptions	Assumes positive values and normality	More flexible; handles various data distributions
Use Cases	Suitable for positive data with normality assumption	More versatile, especially for wider data ranges

2.1.4.3 Quantile Transformers:

The Quantile Transformer is a data transformation technique **used to map the data distribution to a specified probability distribution, often a uniform or Gaussian distribution.** This transformation is especially useful when you want to make your data adhere to certain distributional assumptions, which can be helpful for certain statistical methods or machine learning algorithms.

The general idea of the Quantile Transformer involves mapping the original data values to quantiles of the desired target distribution. **This can be useful for data that does not follow a normal distribution or when you want to mitigate the impact of outliers.**

Here's how the Quantile Transformer works:

Ranking the Data:

The first step involves ranking the original data values. This is done by sorting the data and assigning each value a rank based on its position in the sorted order.

Mapping to Quantiles:

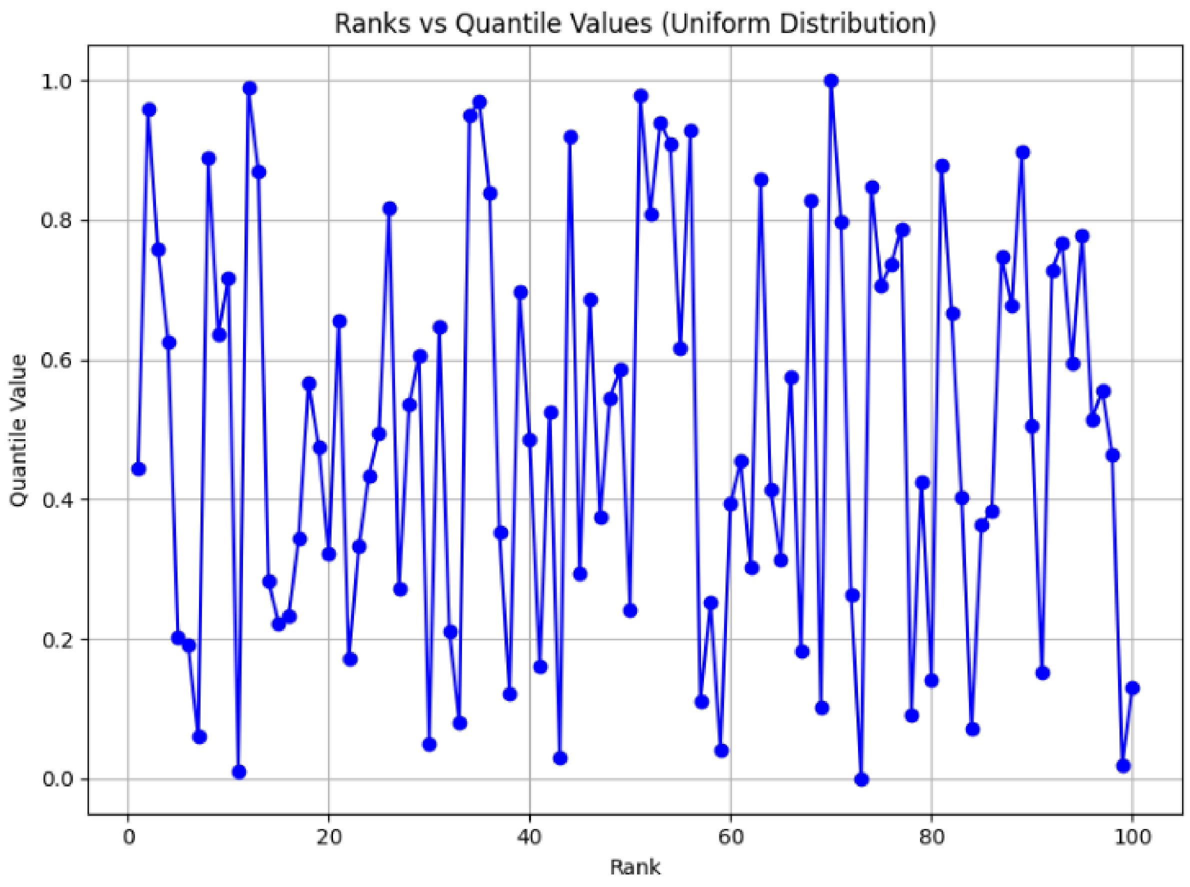
The ranked values are then mapped to quantiles of the desired target distribution. These quantiles are chosen to be uniformly spaced (for a uniform distribution) or based on percentiles (for a Gaussian distribution).

For uniform distribution,

$$\boldsymbol{Quantile\ Value} = \frac{(\boldsymbol{Rank} - \mathbf{0.5})}{\boldsymbol{Total\ Observations}}$$

Eg:

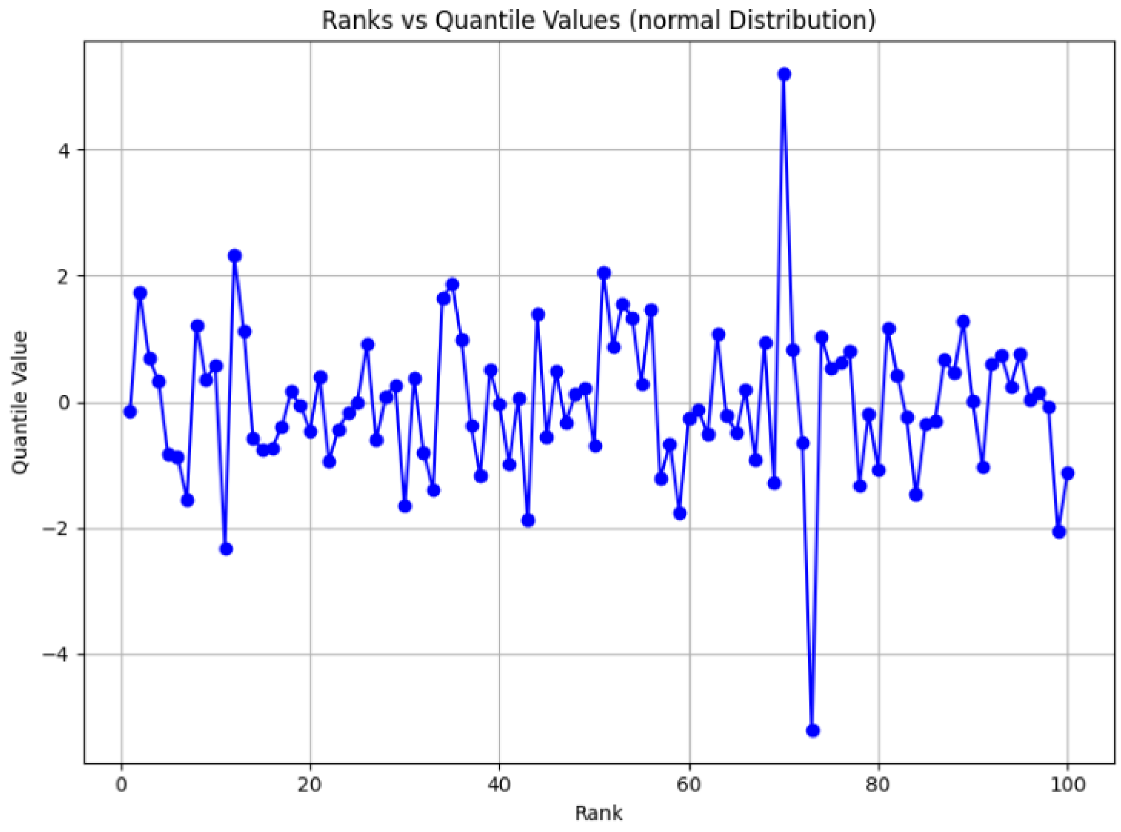
- Rank 1:
  Quantile Value = (1 - 0.5) / 6 = 0.083
- Rank 2:
  Quantile Value = (2 - 0.5) / 6 = 0.25



For a standard normal distribution (mean = 0, standard deviation = 1), the formula to transform a rank (percentile) into a quantile value is:

$$\boldsymbol{Quantile\ Value = PPF(Total\ ObservationsRank - 0.5)}$$

Where PPF represents the percent point function (quantile function) of the standard normal distribution.



Applying Inverse Transformation:

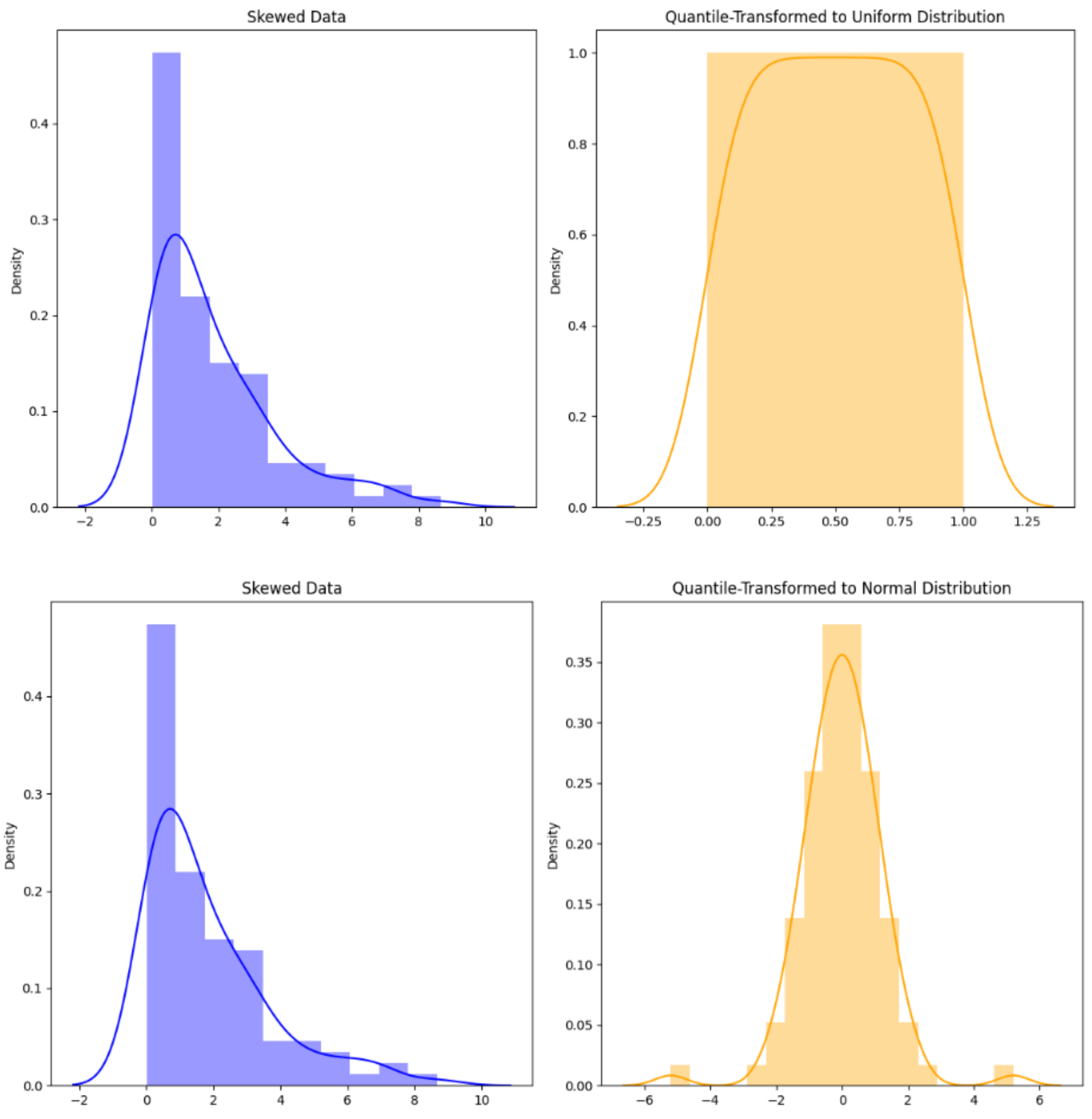
To transform the data back to its original scale, an inverse transformation is applied. This involves mapping the quantile-transformed values back to the original data distribution.

For both uniform distribution and normal distribution ,

Original Value = Quantile Value * (Max Value - Min Value) + Mini Value

Quantile Value 0.083: Original Value = 0.083 * (6 - 1) + 1 = 1.5

Quantile Value 0.25: Original Value = 0.25 * (6 - 1) + 1 = 2.25



2.2 Benefits of Feature Transformation:

1. **Improves Model Performance:** By transforming the features into a more suitable representation, the model can learn more meaningful patterns in the data.
2. **Increases Model Robustness:** Transforming the features can make the model more robust to outliers and other anomalies.
3. **Improves Computational Efficiency:** The transformed features often require fewer computational resources.
4. **Improves Model Interpretability:** By transforming the features, it can be easier to understand the model's predictions.

3 Feature Extraction

Feature Extraction is the process of creating new features from existing ones to provide more relevant information to the machine learning model. This is done by transforming, combining, or aggregating existing features.

3.1 Types of Feature Extraction:

3.1.1 Dimensionality Reduction:

Reducing the number of features by transforming the data into a lower-dimensional space while retaining important information. Examples are PCA and t-SNE.

3.1.1.1 Why Reduce Dimensions?

Discover hidden correlations: Situation where we would like to discover latent dimension along which the data varies

Remove redundant and noisy features: In order to get rid of noisy features or noisy columns from a large dataset. By removing such redundant features, we reduce the dimension of the overall matrix, at the same time we still preserve most of the features that are highly influential to the overall model.

Interpretation and visualization: It becomes easier to plot and visualize the features by reducing high dimensional data to either two or three dimension axis.

Easier storage and processing of data: By reducing dimensionality we shrink the data size and it becomes easy to store, analyze and process data.

3.1.1.2 PCA (Principal Component Analysis)

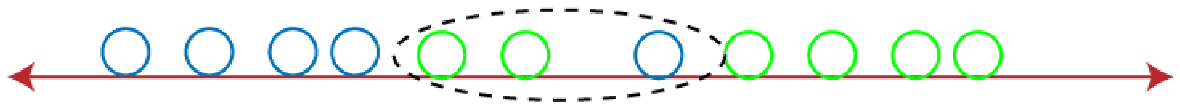
It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components.

3.1.1.3 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is one of the commonly used dimensionality reduction techniques in machine learning to solve more than two-class classification problems. It is also known as Normal Discriminant Analysis (NDA) or Discriminant Function Analysis (DFA).

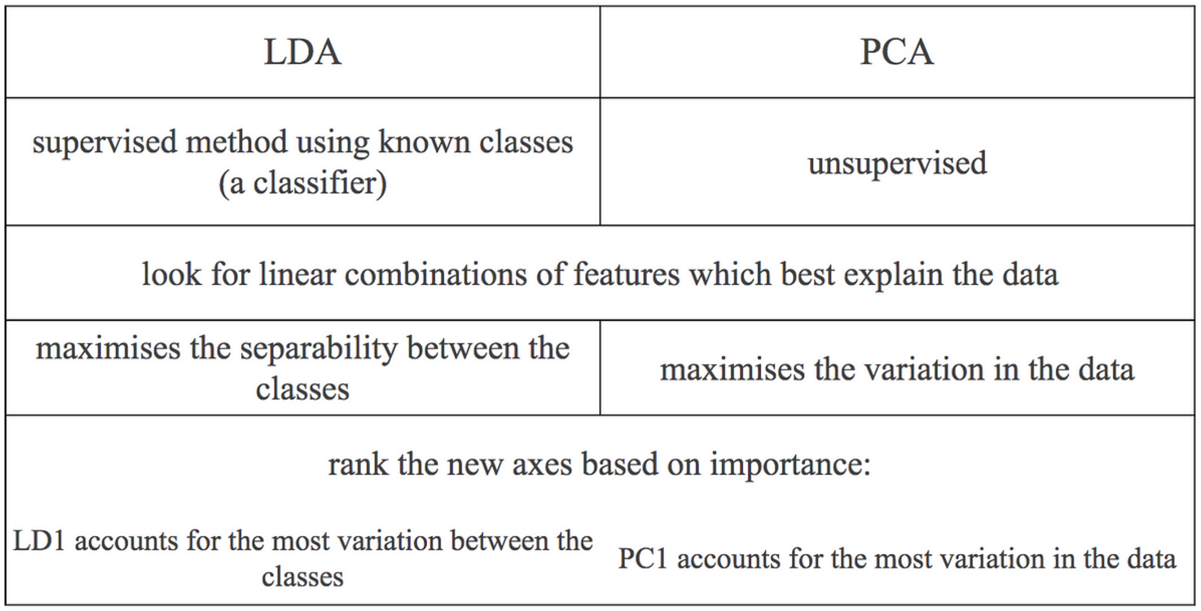
Although the logistic regression algorithm is limited to only two-class, linear Discriminant analysis is applicable for more than two classes of classification problems.

Whenever there is a requirement to separate two or more classes having multiple features efficiently, the Linear Discriminant Analysis model is considered the most common technique to solve such classification problems. For e.g., if we have two classes with multiple features and need to separate them efficiently. When we classify them using a single feature, then it may show overlapping.



Overlapping

To overcome the overlapping issue in the classification process, we must increase the number of features regularly.



LDA	PCA
supervised method using known classes (a classifier)	unsupervised
look for linear combinations of features which best explain the data	
maximises the separability between the classes	maximises the variation in the data
rank the new axes based on importance:	
LD1 accounts for the most variation between the classes	PC1 accounts for the most variation in the data

3.1.1.4 t-SNE:

3.1.1.4.1 What is t-SNE?

t-SNE is a nonlinear dimensionality reduction technique that is well suited for embedding high dimension data into lower dimensional data (2D or 3D) for data visualization.

t-SNE stands for t-distributed Stochastic Neighbor Embedding, which tells the following :

Stochastic → not definite but random probability

Neighbor →concerned only about retaining the variance of neighbor points

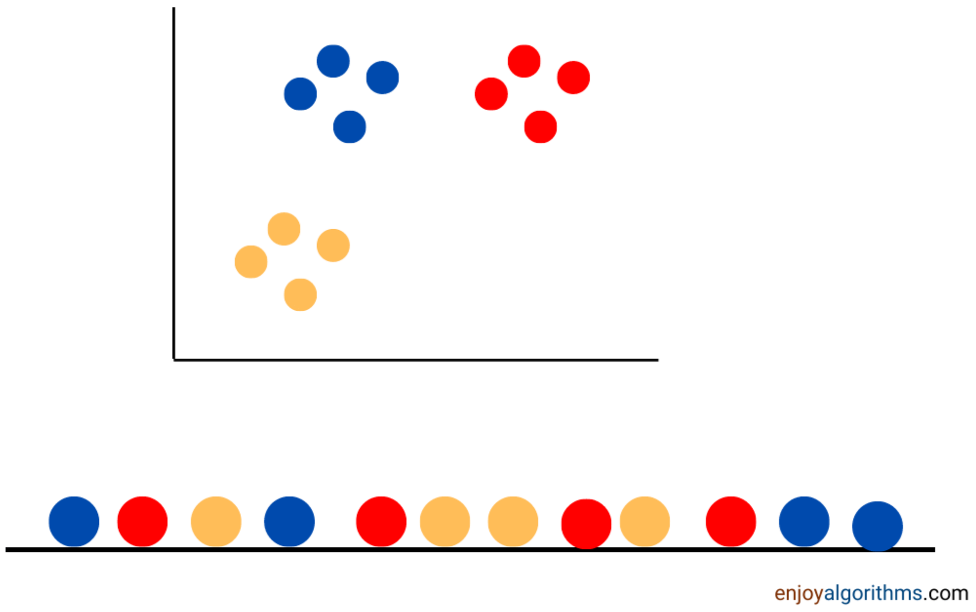
Embedding → plotting data into lower dimensions

3.1.1.4.2 Hyperparameter tuning

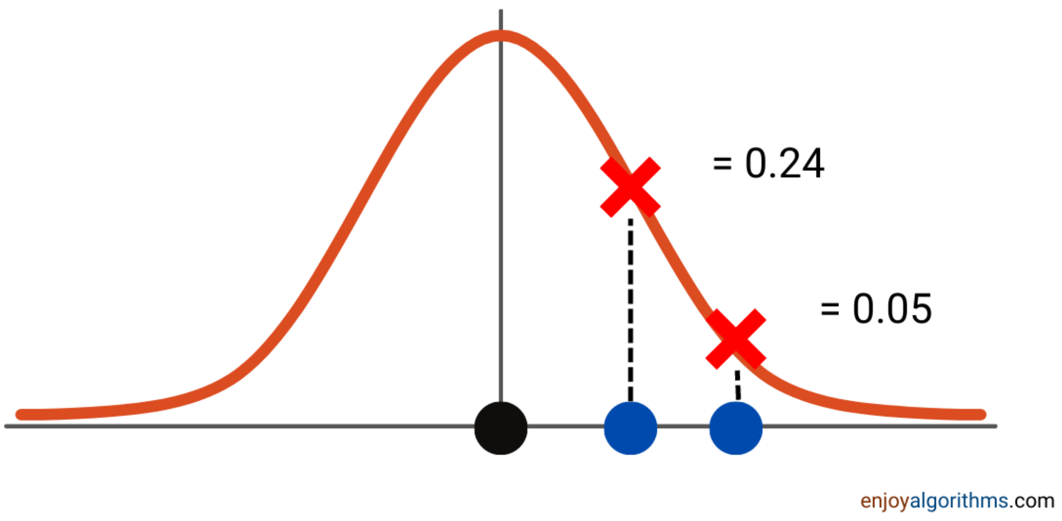
2 parameters that can highly influence the results are
a) n_iter: The number of iterations that the algorithm runs
b) perplexity: This can be thought of as the number of neighboring points t-SNE must consider

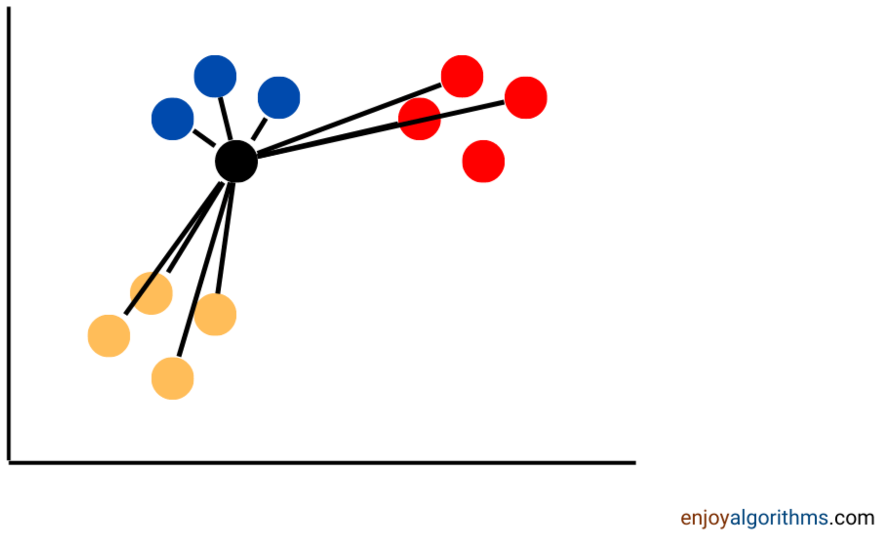
Stepwise working of the tsne:

Step 1: Initialization of Points
We plot all the points randomly on a 1d line shown in the image above. Next, we have to move them step-by-step such that points of similar characteristics gather together, whereas points of different characteristics move far apart. These points will be moved based on the similarity that we will calculate.

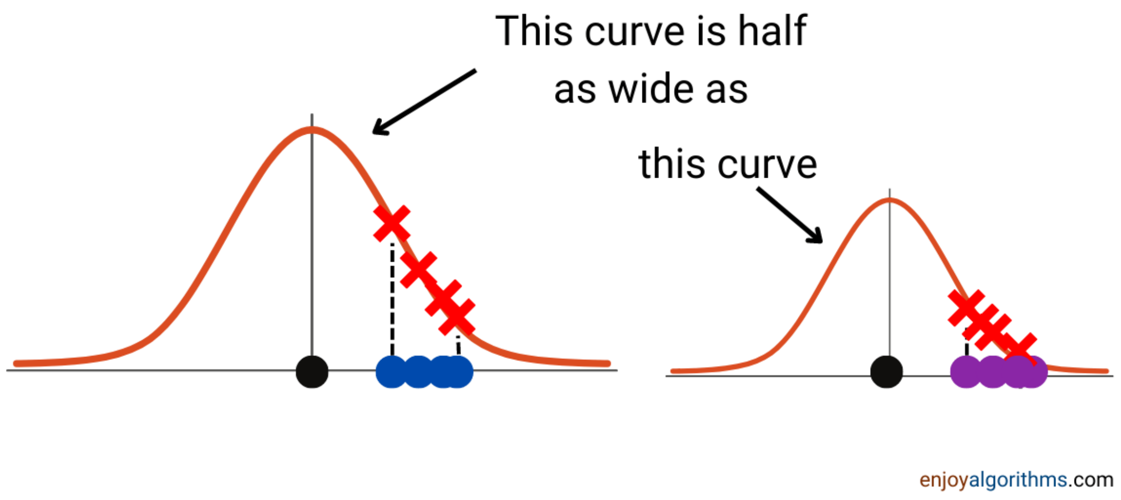


Step 2: Concept of Similarity and its use:
The similarity between the two points is defined in terms of a probability distribution. Here, the probability distribution for two points, A to B, is defined as a conditional probability that A will pick B as its neighbor.

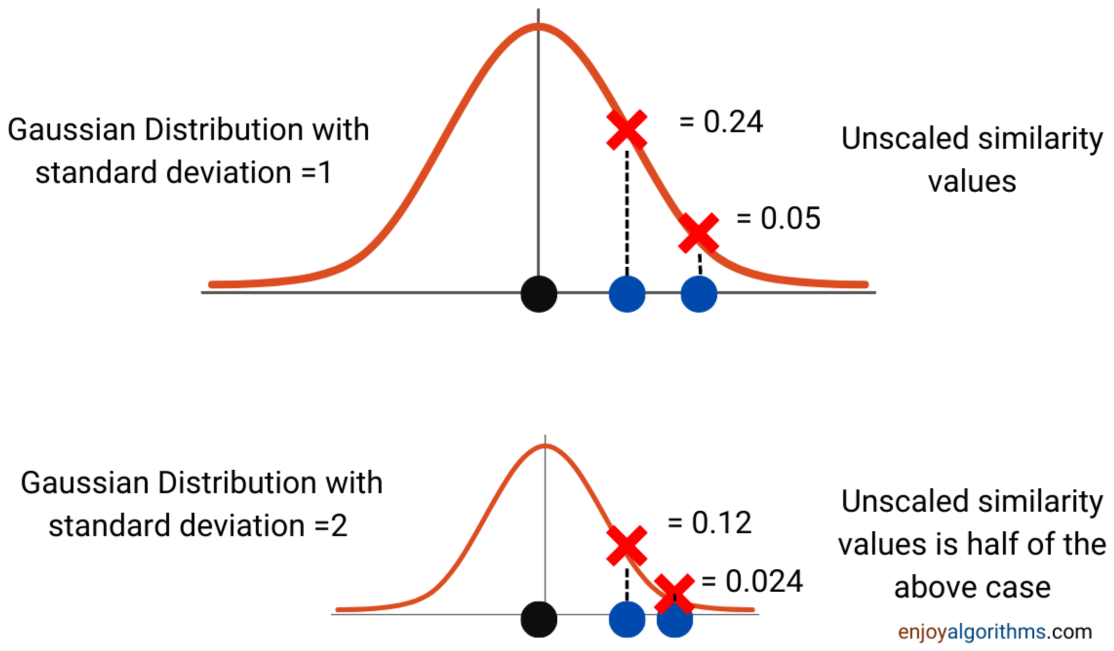




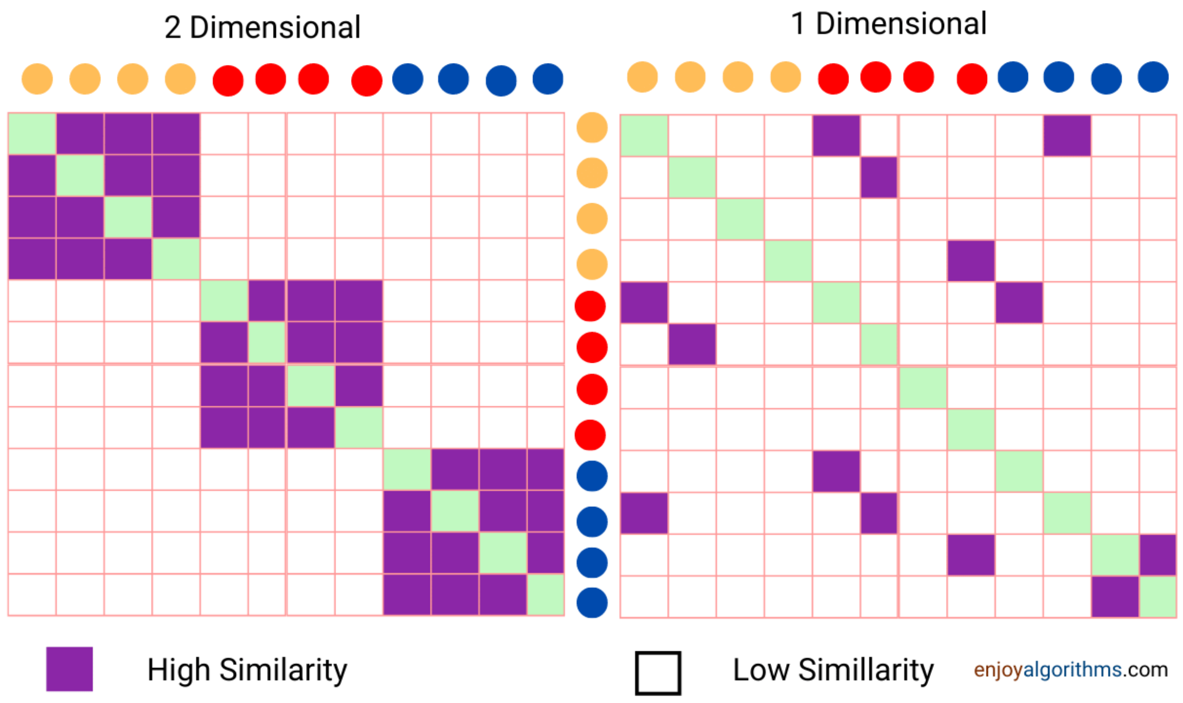
Step 3: t-SNE perplexity:
Perplexity tells the density of points relative to a particular point. If 4 points of similar characteristics are densely clustered, they will have higher perplexity than those not. Points with less density around them have flatter normal curves than curves formed for points with more density. In the following case of the figure below, purple points are sparse.

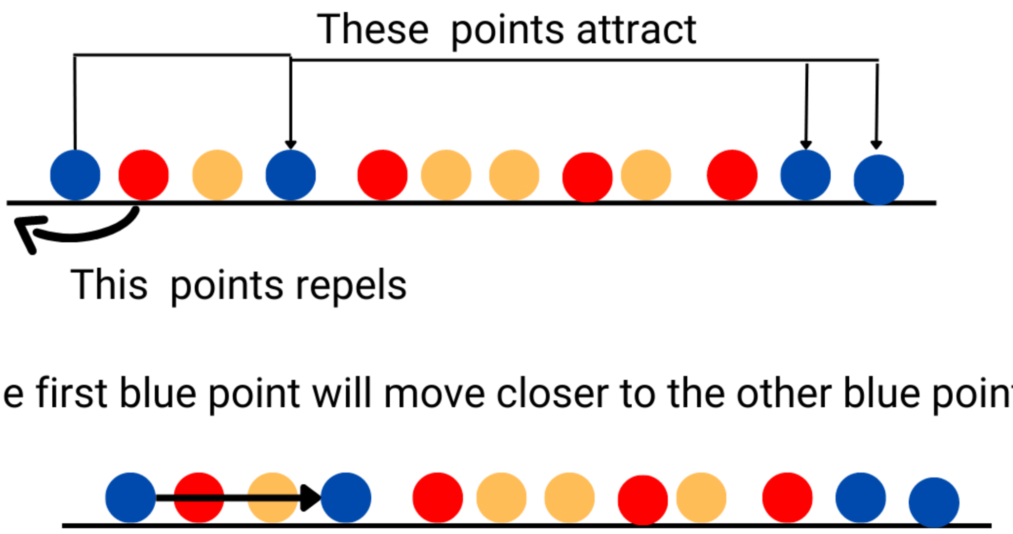


Step 4: Normalizing Perplexities:
Suppose we consider 4 points for denser plots and 4 points for less dense plots but having a similar distance ratio (dense has values double that of less dense). In simple terms, we have two sets, and the distance average in the denser group is half the distance average in the less dense group. These two sets of 4 points are the same if seen separately and deserve the same treatment. But with the existing approach, that will not be the case as y values will differ, i.e., similarity differs.


Gaussian Distribution with standard deviation =1
= 0.24
= 0.05
Unscaled similarity values
Gaussian Distribution with standard deviation =2
= 0.12
= 0.024
Unscaled similarity values is half of the above case
enjoyalgorithms.com

Step 5: Making Similarity Matrix:
We end up with a similarity matrix where the points with high similarity scores mean they belong to the same cluster. Please note that the matrix defines the similarity of a point to itself also (green squares), but it does not make any sense to do that, so t-SNE defines these similarity scores as 0.




These points attract
This points repels
So ithe first blue point will move closer to the other blue points

Advantages and Disadvantages of the t-SNE Algorithm:
Advantages,

1. Handles Non-linear Data, unlike PCA, which takes linear data
2. Preserves Local Structure:
   In PCA, we don't maintain local structure after dimension reduction as it cares about variance. But here, the local structure is maintained, i.e., points nearby in the original dimensions will also be nearby after dimension reduction. So it helps us to capture the essence of the data.

Disadvantages,

1. Computational Complexity:
   t-SNE involves complex calculations as it calculates the pairwise conditional probability for each point. Due to this, it takes more time as the number of data points increases.
2. Non-Deterministic:
   We may get different results with the same hyperparameters. This means that even though code and data points are the same in each iteration, we may get different results because of the randomization involved in the process.

3.1.1.4.3 Application of t-SNE

t-SNE has a wide range of applications in machine learning, particularly in the field of data visualization. Here are some of the most common applications of t-SNE:

1. **Image and Video Processing:** t-SNE can be used to analyze and visualize large sets of images and videos. By reducing the dimensionality of the image or video features, t-SNE can help to cluster similar images and identify patterns in large data sets. This makes it a useful tool for categorizing, segmenting, and retrieving images and videos.
2. **Natural Language Processing:**

   Natural language processing software frequently makes use of t-SNE. It may be used to illustrate the semantic connections between words in a sizable collection of textual information. By reducing the dimensionality of word embeddings, t-SNE can help to cluster words that have similar meanings, making it easier to identify patterns in the data.

3. **Biological Data Analysis:**
   t-SNE has many applications in the field of biology, particularly in the analysis of high-dimensional gene expression data. By reducing the dimensionality of gene expression data, t-SNE can help to identify patterns in the data and cluster genes that have similar expression profiles. This can lead to a better understanding of the biological processes that underlie disease and other complex phenotypes.

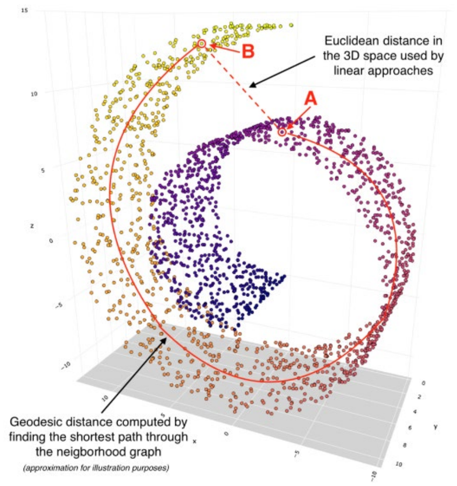
4. **Anomaly Detection:**

   With huge data sets, abnormalities can be found using t-SNE. By visualizing the data in a low-dimensional space, t-SNE can help to identify clusters of data points that are different from the rest of the data. This can be used to identify potential fraud or other anomalies in financial data or to identify outliers in other types of data sets.

5. **Recommender Systems:**
   t-SNE can also be used in recommender systems to help identify similar items based on their features. By reducing the dimensionality of the item features, t-SNE can help to cluster items that are similar to one another, making it easier to recommend similar items to users based on their preferences.

6. **Social Network Analysis:**
   t-SNE can be used to visualize the social networks of large groups of people. By reducing the dimensionality of social network features, t-SNE can help to identify clusters of people that are connected to one another, making it easier to identify influential people or groups within the network

3.1.1.5 Isomap:



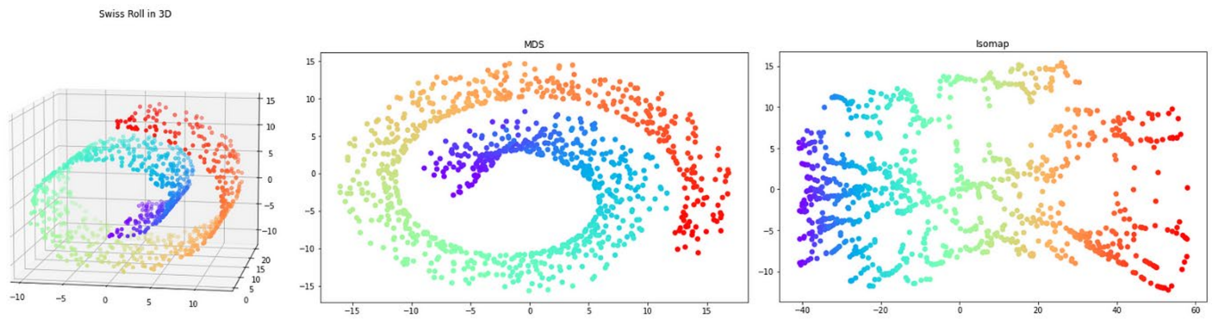
Isomap, short for Isometric Feature Mapping, is another dimensionality reduction technique that focuses on preserving the intrinsic geometry and structure of data points in a lower-dimensional space. It was introduced by Tenenbaum, de Silva, and Langford in 2000 as an extension of earlier manifold learning methods.

The main idea behind Isomap is to capture the underlying manifold structure of high-dimensional data by considering the geodesic distances (shortest path distances) between data points on the manifold.

3.1.1.5.1 Here's how Isomap works:

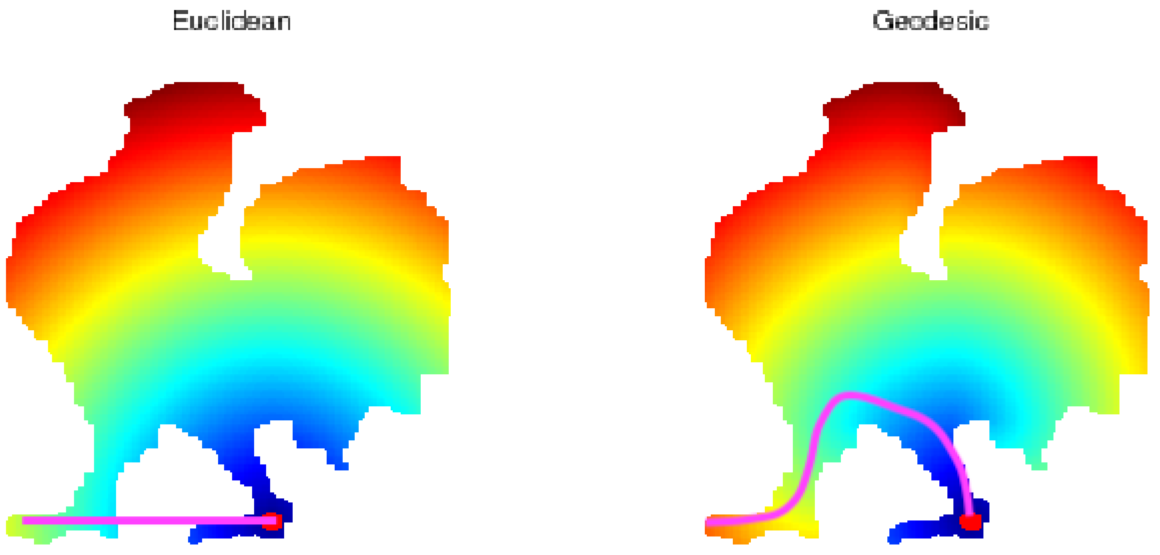
Nearest Neighbor Graph:

Isomap starts by constructing a nearest neighbor graph, where each data point is connected to its k nearest neighbors. The graph represents the local connectivity of the data.



Geodesic Distances:

Isomap computes the geodesic distances between all pairs of data points on the manifold. These distances represent the true distances along the manifold's structure, rather than the Euclidean distances in the original high-dimensional space.

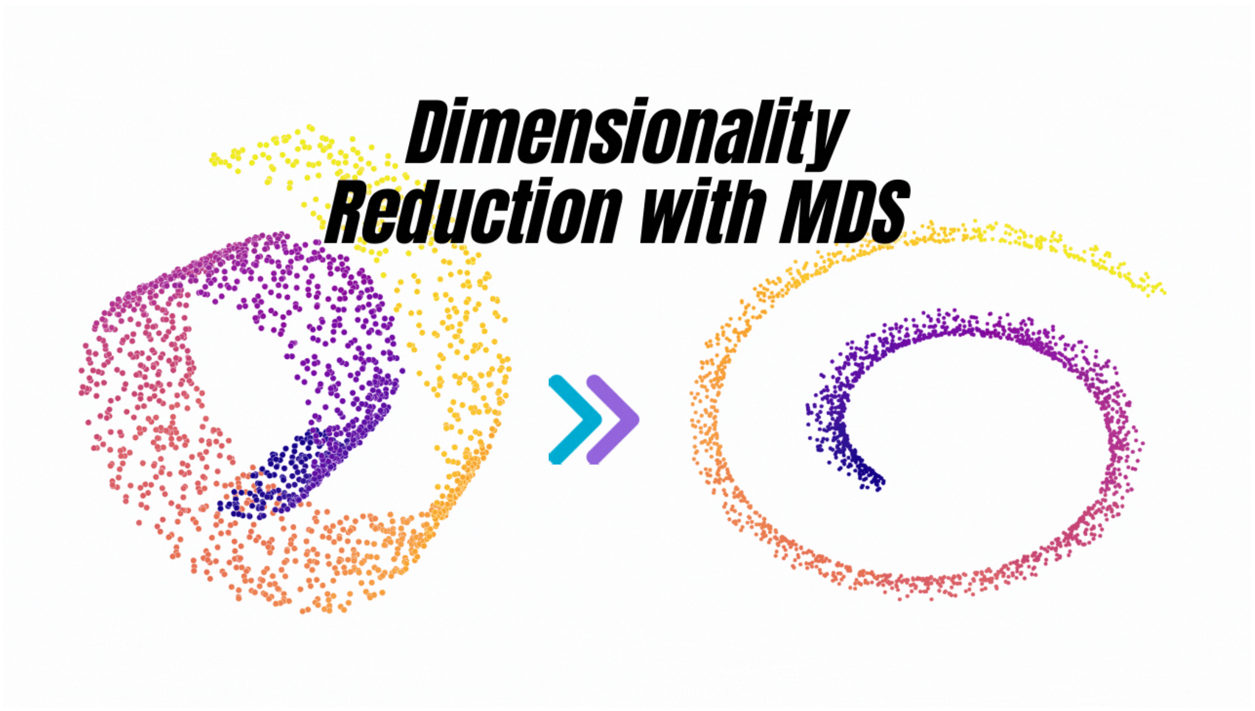


Graph Distance Matrix:

The geodesic distances are used to create a distance matrix that captures the pairwise distances between data points.

Multi-Dimensional Scaling (MDS):

Isomap then applies multi-dimensional scaling (MDS) to the distance matrix. MDS is a technique that aims to find a lower-dimensional representation of the data that preserves the pairwise distances as much as possible.



By using geodesic distances and MDS, Isomap attempts to unfold the data along the underlying manifold structure. This can be particularly useful when the high-dimensional data is intrinsically non-linear and contains curved or twisted patterns.

3.1.1.5.2 strengths and limitations

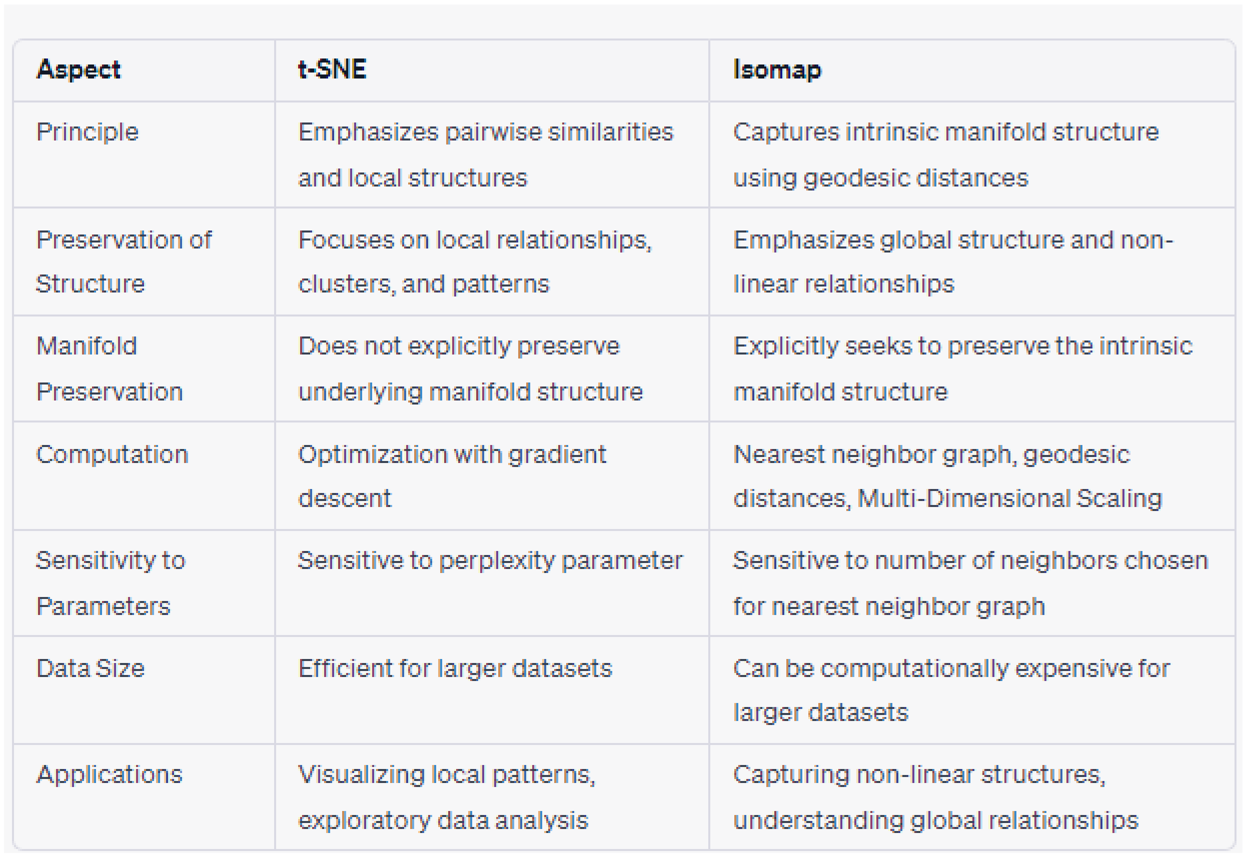
Like other dimensionality reduction techniques, Isomap has its strengths and limitations:

Advantages:

- It can capture non-linear relationships and preserve the intrinsic geometry of the data.
- Isomap is less sensitive to the curse of dimensionality compared to methods like PCA.
- It's especially useful for visualizing and understanding complex data structures.

Limitations:

- Isomap's performance can degrade if the nearest neighbor graph is not properly constructed or if there are outliers.
- It can be computationally expensive, especially for large datasets.
- The quality of results might be influenced by the choice of hyperparameters, such as the number of nearest neighbors.



Aspect	t-SNE	Isomap
Principle	Emphasizes pairwise similarities and local structures	Captures intrinsic manifold structure using geodesic distances
Preservation of Structure	Focuses on local relationships, clusters, and patterns	Emphasizes global structure and non-linear relationships
Manifold Preservation	Does not explicitly preserve underlying manifold structure	Explicitly seeks to preserve the intrinsic manifold structure
Computation	Optimization with gradient descent	Nearest neighbor graph, geodesic distances, Multi-Dimensional Scaling
Sensitivity to Parameters	Sensitive to perplexity parameter	Sensitive to number of neighbors chosen for nearest neighbor graph
Data Size	Efficient for larger datasets	Can be computationally expensive for larger datasets
Applications	Visualizing local patterns, exploratory data analysis	Capturing non-linear structures, understanding global relationships

3.1.2 Feature Combination:
Combining two or more existing features to create a new one. For example, the interaction between two features.

3.1.3 Feature Aggregation:
Aggregating features to create a new one. For example, calculating the mean, sum, or count of a set of features.

3.1.4 Feature Transformation:
Transforming existing features into a new representation. For example, log transformation of a feature with a skewed distribution.

3.2 Benefits of Feature Extraction:

1. **Improves Model Performance:** By creating new and more relevant features, the model can learn more meaningful patterns in the data.
2. **Reduces Overfitting:** By reducing the dimensionality of the data, the model is less likely to overfit the training data.
3. **Improves Computational Efficiency:** The transformed features often require fewer computational resources.
4. **Improves Model Interpretability:** By creating new features, it can be easier to understand the model’s predictions.

4 Feature selection

In feature selection, we select a subset of features from the data set to train machine learning algorithms. Feature selection techniques differ from dimensionality reduction in that they do not alter the original representation of the variables but merely select a smaller set of features

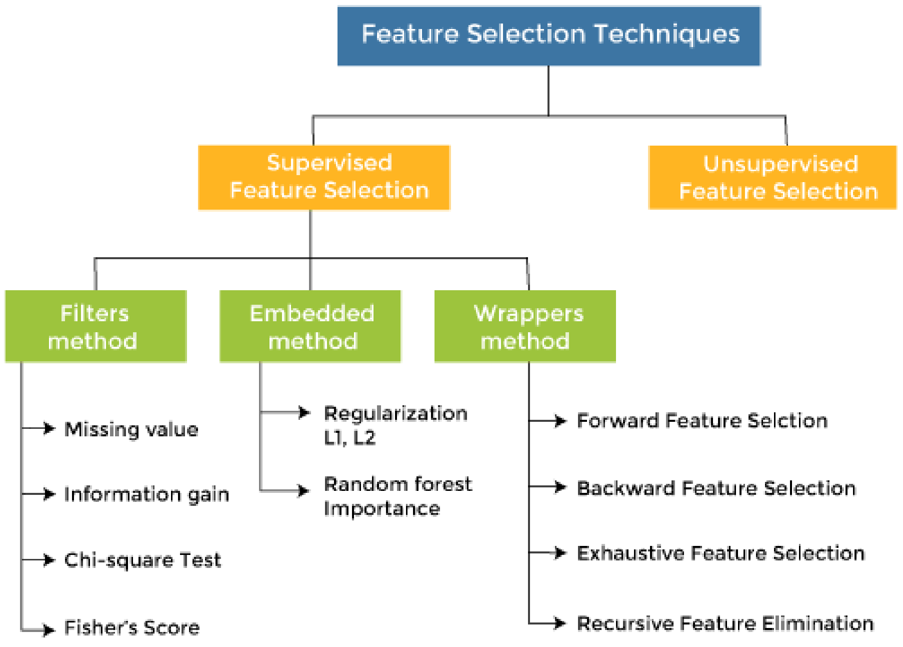
benefits of using feature selection in machine learning:

- It helps in avoiding the curse of dimensionality.
- It helps in the simplification of the model so that it can be easily interpreted by the researchers.
- It reduces the training time.
- It reduces overfitting hence enhance the generalization.

4.1 Feature Selection Techniques

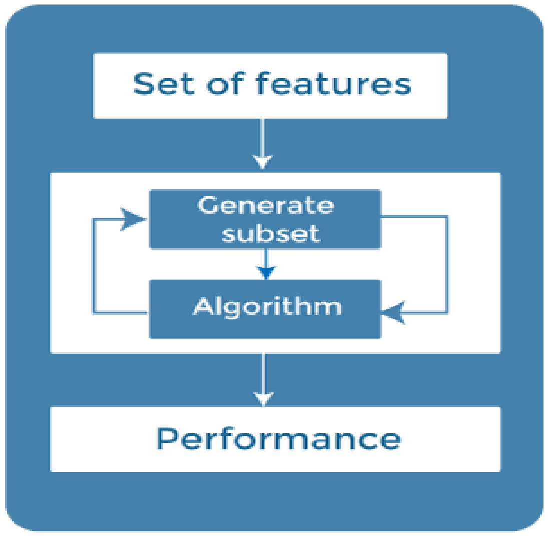
There are mainly two types of Feature Selection techniques, which are:

1. *Supervised Feature Selection technique*
   Supervised Feature selection techniques consider the target variable and can be used for the labelled dataset.
2. *Unsupervised Feature Selection technique*
   Unsupervised Feature selection techniques ignore the target variable and can be used for the unlabelled dataset.



4.1.1 Wrapper Methods

Wrapper methods, also referred to as greedy algorithms train the algorithm by using a subset of features in an iterative manner. Based on the conclusions made from training in prior to the model, addition and removal of features takes place. Stopping criteria for selecting the best subset are usually pre-defined by the person training the model such as when the performance of the model decreases or a specific number of features has been achieved.



4.1.1.1 Forward selection

Forward selection is an iterative process, which begins with an empty set of features. After each iteration, it keeps adding on a feature and evaluates the performance to check whether it is improving the performance or not. The process continues until the addition of a new variable/feature does not improve the performance of the model.

4.1.1.2 Backward elimination

Backward elimination is also an iterative approach, but it is the opposite of forward selection. This technique begins the process by considering all the features and removes the least significant feature. This elimination process continues until removing the features does not improve the performance of the model.

4.1.1.3 Bi-directional elimination

This method uses both forward selection and backward elimination technique simultaneously to reach one unique solution.

4.1.1.4 Exhaustive Feature Selection

It is one of the best feature selection methods, which evaluates each feature set as brute-force. It means this method tries & make each possible combination of features and return the best performing feature set.

4.1.1.5 Recursive Feature Elimination

Recursive feature elimination is a recursive greedy optimization approach, where features are selected by recursively taking a smaller and smaller subset of features. Now, an estimator is trained with each set of features, and the importance of each feature is determined using *coef_attribute* or through a *feature_importances_attribute.*

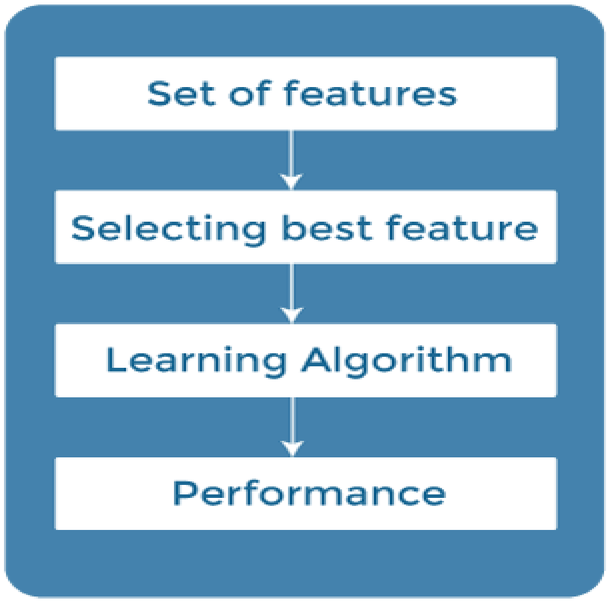
	Forward Selection	**Backward Elimination**	**Bi-directional Elimination**	**Exhaustive Feature Selection**	**Recursive Feature Elimination**
Description	Forward selection starts with an empty set and iteratively adds features that improve model performance.	Backward elimination begins with all features and removes them iteratively to simplify the model.	Bi-directional elimination combines forward and backward steps to iteratively add and remove features.	Exhaustive feature selection evaluates all possible feature combinations to find the optimal subset.	Recursive feature elimination removes the least important feature in each step using model performance.

Use Cases	Small to moderate feature space, when building the model iteratively.	Small to moderate feature space, when simplifying the model iteratively.	Balancing benefits of both forward and backward selection, when feature interactions are important.	Small feature space, exhaustive search for optimal feature subset.	Moderate feature space, leveraging model performance for feature selection.
Complexity	Moderate	Moderate	Moderate to High	High	Moderate
Agnostic	Yes	Yes	Yes	Yes	Yes
Interactions	No	No	Yes	Yes	No
Interpretability	Moderate	Moderate	Moderate	Low	Moderate
Pros	- Can capture complex feature interactions over iterations.	- Can help simplify the model while retaining key features.	- Can capture both individual feature importance and interactions.	- Guaranteed to find the globally optimal subset.	- Leverages the actual model performance for selection.
Cons	- Prone to local optima if a globally optimal subset requires removing previously added features.	- Similar to Forward Selection, it may not find globally optimal subsets.	- Computationally more expensive due to bidirectional nature.	- Computationally expensive and infeasible for large feature spaces.	- Might not capture complex interactions if removed features play a role in conjunction with others.

1. **Model Agnostic (Agnostic):**
    - This refers to whether the feature selection method can be applied regardless of the specific machine learning algorithm you intend to use. If a method is "agnostic," it means it doesn't rely on any particular algorithm and can work with a wide range of models.
2. **Handles Feature Interactions (Interactions):**
    - Feature interactions occur when the effect of one feature on the target variable depends on the values of other features. For example, the impact of age on salary might differ based on the level of education. If a method "handles feature interactions," it means it takes into account these complex relationships between features when selecting them.
3. **Interpretability:**
    - This refers to how easily the selected features can be understood and explained. Interpretability is categorized into three levels:
        - **High**: The selected features are easily understandable and can be explained clearly.
        - **Moderate**: Some level of interpretation is possible, but it may require more effort to explain the selections.
        - **Low**: The selected features are less intuitive, making them challenging to explain or understand.

4.1.2 Filter Methods

In Filter Method, features are selected on the basis of statistics measures. These methods are generally used while doing the pre-processing step. These methods select features from the dataset irrespective of the use of any machine learning algorithm. In terms of computation, they are very fast and inexpensive and are very good for removing duplicated, correlated, redundant features but these methods do not remove multicollinearity.



Some common techniques of Filter methods are as follows:

4.1.2.1 Information Gain:

It is defined as the amount of information provided by the feature for identifying the target value and measures reduction in the entropy values. Information gain of each attribute is calculated considering the target values for feature selection..

4.1.2.2 Chi-square Test:

Chi-square test is a technique to determine the relationship between the categorical variables. The chi-square value is calculated between each feature and the target variable, and the desired number of features with the best chi-square value is selected.

$$X^2 = \sum \frac{(Observed\ value - Expected\ value)^2}{Expected\ value}$$

4.1.2.3 Fisher's Score:

Fisher's score is one of the popular supervised technique of features selection. It returns the rank of the variable on the fisher's criteria in descending order. Then we can select the variables with a large fisher's score.

4.1.2.4 Missing Value Ratio:

The value of the missing value ratio can be used for evaluating the feature set against the threshold value. The formula for obtaining the missing value ratio is the number of missing values in each column divided by the total number of observations. The variable is having more than the threshold value can be dropped.

$$\text{Missing Value Ratio}=\frac{\textit{Number of Missing values}*100}{\textit{Total number of observations}}$$

4.1.2.5 Correlation Coefficient

Pearson's Correlation Coefficient is a measure of quantifying the association between the two continuous variables and the direction of the relationship with its values ranging from -1 to 1.

	Pearson Correlation Coefficient	**Spearman Rank Correlation Coefficient**
Type of Relationship	Measures linear relationship between continuous variables	Assesses monotonic relationship between variables
Data Type	Interval or ratio scale data (numeric with equal intervals)	Ordinal, interval, or ratio scale data
Assumptions	Assumes linear relationship and normal distribution	Does not assume specific distribution or linearity
Strength of Relationship	Captures linear relationships well	More robust to outliers and non-linear relationships
Example Use Case	Height vs. weight of individuals	Rank of exercise frequency vs. overall health

4.1.2.6 Variance Threshold

It is an approach where all features are removed whose variance doesn't meet the specific threshold. By default, this method removes features having zero variance. The assumption made using this method is higher variance features are likely to contain more information.

4.1.2.7 Mean Absolute Difference (MAD)

This method is similar to variance threshold method but the difference is there is no square in MAD. This method calculates the mean absolute difference from the mean value.

4.1.2.8 Dispersion Ratio

Dispersion ratio is defined as the ratio of the Arithmetic mean (AM) to that of Geometric mean (GM) for a given feature. Its value ranges from +1 to ∞ as AM ≥ GM for a given feature. Higher dispersion ratio implies a more relevant feature.

4.1.2.9 Mutual Dependence

This method measures if two variables are mutually dependent, and thus provides the amount of information obtained for one variable on observing the other variable. Depending on the presence/absence of a feature, it measures the amount of information that feature contributes to making the target prediction.

4.1.2.10 Relief

This method measures the quality of attributes by randomly sampling an instance from the dataset and updating each feature and distinguishing between instances that are near to each other based on the difference between the selected instance and two nearest instances of same and opposite classes.

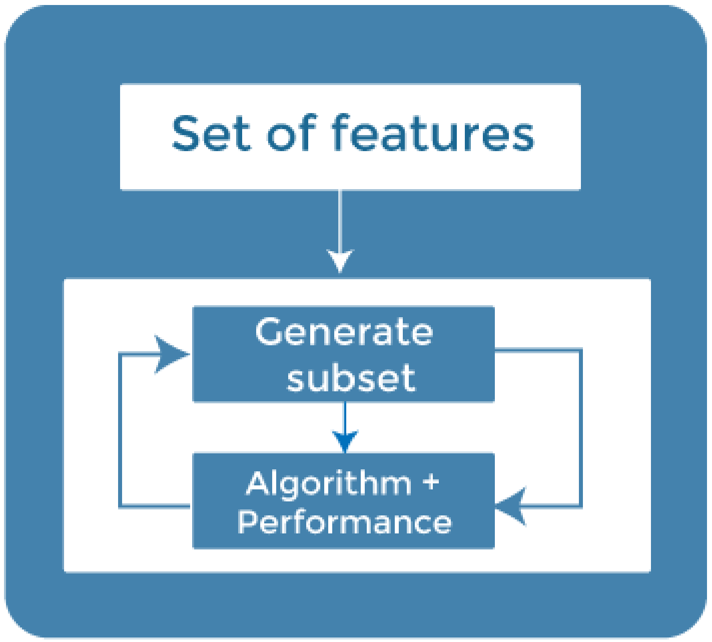
Method	Description	Suitable Use Cases	Pros	Cons
Information Gain	Measures the reduction in uncertainty of the target variable after including a feature.	Categorical target and features, identifying relevant categorical attributes (e.g., text classification).	Simple to compute, handles categorical data.	Ignores feature interactions, biases towards high-cardinality features.
Chi-square Test	Assesses the independence between categorical features and the target variable.	Categorical features and target, exploring associations in categorical data (e.g., survey analysis).	Effective for categorical data.	Limited for continuous features, assumes independence.
Fisher's Score	Measures the ratio of variance between classes to variance within classes for each feature.	Binary classification, numeric features, seeking discriminative attributes (e.g., medical diagnosis).	Captures class separability.	Limited to binary classification, sensitive to class imbalance.

Missing Value Ratio	Evaluates features based on the percentage of missing values.	Features with varying missingness, identifying attributes with data completeness issues.	Simple and quick.	Ignores feature quality, might discard useful information.
Correlation Coefficient	Measures the linear relationship between numerical features and the target variable.	Numeric features, linear relationships, understanding linear dependencies (e.g., financial modeling).	Captures linear dependencies.	Ignores non-linear relationships.
Variance Threshold	Removes features with variance below a specified threshold.	Numeric features with varying variances, reducing noise in data (e.g., sensor readings).	Removes low-variance noise.	Ignores feature interactions, not effective for binary classification.
Mean Absolute Difference	Compares the distribution of a feature's values for different classes.	Numeric features, binary classification, identifying class-specific patterns (e.g., medical diagnosis).	Captures class-discriminative patterns.	Sensitive to noise, limited to binary classification.
Dispersion Ratio	Measures the ratio of within-class variance to between-class variance.	Numeric features, multi-class classification, identifying attributes with different class distributions.	Captures within/between-class variance.	Limited to multi-class classification, assumes normality.

Mutual Dependence	Assesses the dependence between a feature and the target using mutual information or similar metrics.	Versatile across feature and target types, detecting non-linear dependencies (e.g., customer segmentation).	Considers non-linear relationships.	Computationally expensive for high-dimensional data.
Relief	Evaluates feature importance by comparing feature values for nearest neighbors with the same/different class.	Numeric/categorical features, classification, identifying locally important features (e.g., image recognition).	Captures local feature relevance.	Computationally expensive for large datasets, sensitive to K value.

4.1.3 Embedded Methods

Embedded methods combined the advantages of both filter and wrapper methods by considering the interaction of features along with low computational cost. These are fast processing methods similar to the filter method but more accurate than the filter method.



These methods are also iterative, which evaluates each iteration, and optimally finds the most important features that contribute the most to training in a particular iteration. Some techniques of embedded methods are:

4.1.3.1 Regularization-

Regularization adds a penalty term to different parameters of the machine learning model for avoiding overfitting in the model. This penalty term is added to the coefficients; hence it shrinks some coefficients to zero. Those features with zero coefficients can be removed from the dataset. The types of regularization techniques are L1 Regularization (Lasso Regularization) or Elastic Nets (L1 and L2 regularization).

4.1.3.2 Random Forest Importance

Different tree-based methods of feature selection help us with feature importance to provide a way of selecting features. Here, feature importance specifies which feature has more importance in model building or has a great impact on the target variable. Random Forest is such a tree-based method, which is a type of bagging algorithm that aggregates a different number of decision trees. It automatically ranks the nodes by their performance or decrease in the impurity (Gini impurity) over all the trees. Nodes are arranged as per the impurity values, and thus it allows to pruning of trees below a specific node. The remaining nodes create a subset of the most important features.

4.1.4 Hybrid methods

Hybrid methods for feature selection combine elements from different types of feature selection techniques, often incorporating the strengths of multiple methods to achieve improved feature selection performance. These methods leverage the advantages of each individual approach and aim to mitigate their limitations. Hybrid methods can provide more robust and effective feature selection by leveraging diverse strategies. Here are a few popular hybrid methods:

4.1.4.1 Recursive Feature Elimination with Cross-Validation (RFECV):

RFECV is an extension of Recursive Feature Elimination (RFE) that incorporates cross-validation to find the optimal number of features. It starts by recursively eliminating less important features and then uses cross-validation to assess model performance with different feature subsets. It helps prevent overfitting and optimizes the number of features to retain.

4.1.4.2 Genetic Algorithms for Feature Selection:

Genetic algorithms mimic the process of natural selection to search for the best feature subset. They generate and evolve populations of potential feature subsets over multiple iterations. Genetic algorithms can explore a wide search space and can handle both continuous and categorical features. They are particularly useful when dealing with a large number of features.

4.1.4.3 Embedded Filters with Wrapper Techniques:

This approach combines embedded filter methods (such as L1 regularization or tree-based feature selection) with wrapper techniques. The embedded filter identifies important features during model training, and then a wrapper method like Recursive Feature Elimination (RFE) is applied to refine the subset further. This hybrid method leverages the efficiency of embedded filters while benefiting from the iterative nature of wrapper methods.

4.1.4.4 Meta-Feature Selection:

Meta-feature selection involves running multiple feature selection algorithms and combining their results. For instance, you could apply Information Gain, Chi-square Test, and Recursive Feature Elimination, and then aggregate their outcomes to make a final feature selection decision. Meta-feature selection can provide a more balanced and robust approach.

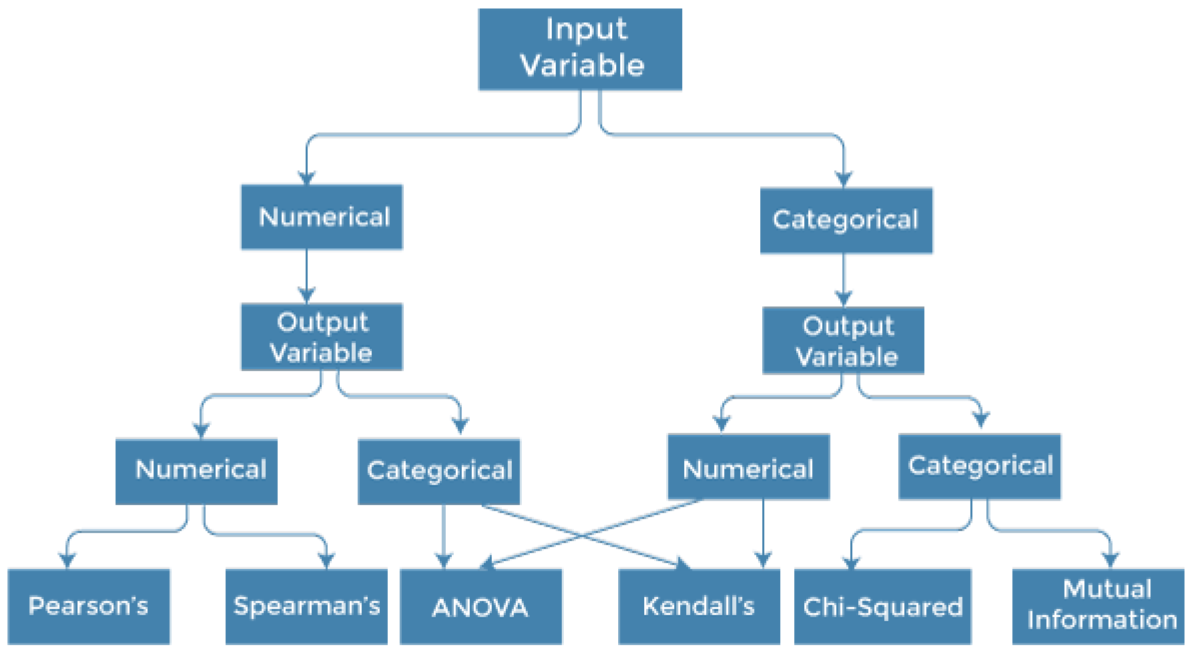
4.1.4.5 Feature Clustering and Selection:

This method involves clustering similar features together and selecting representative features from each cluster. It's particularly useful when dealing with high-dimensional data. By grouping correlated or similar features, you can reduce redundancy and improve computational efficiency.

4.1.4.6 Ensemble Feature Selection:

Ensemble methods create multiple models and combine their predictions. Similarly, ensemble feature selection combines the results of different feature selection techniques. For instance, you could use Information Gain, Mutual Dependence, and Relief, and then aggregate their rankings to select features.

4.2 How to choose a Feature Selection Method?



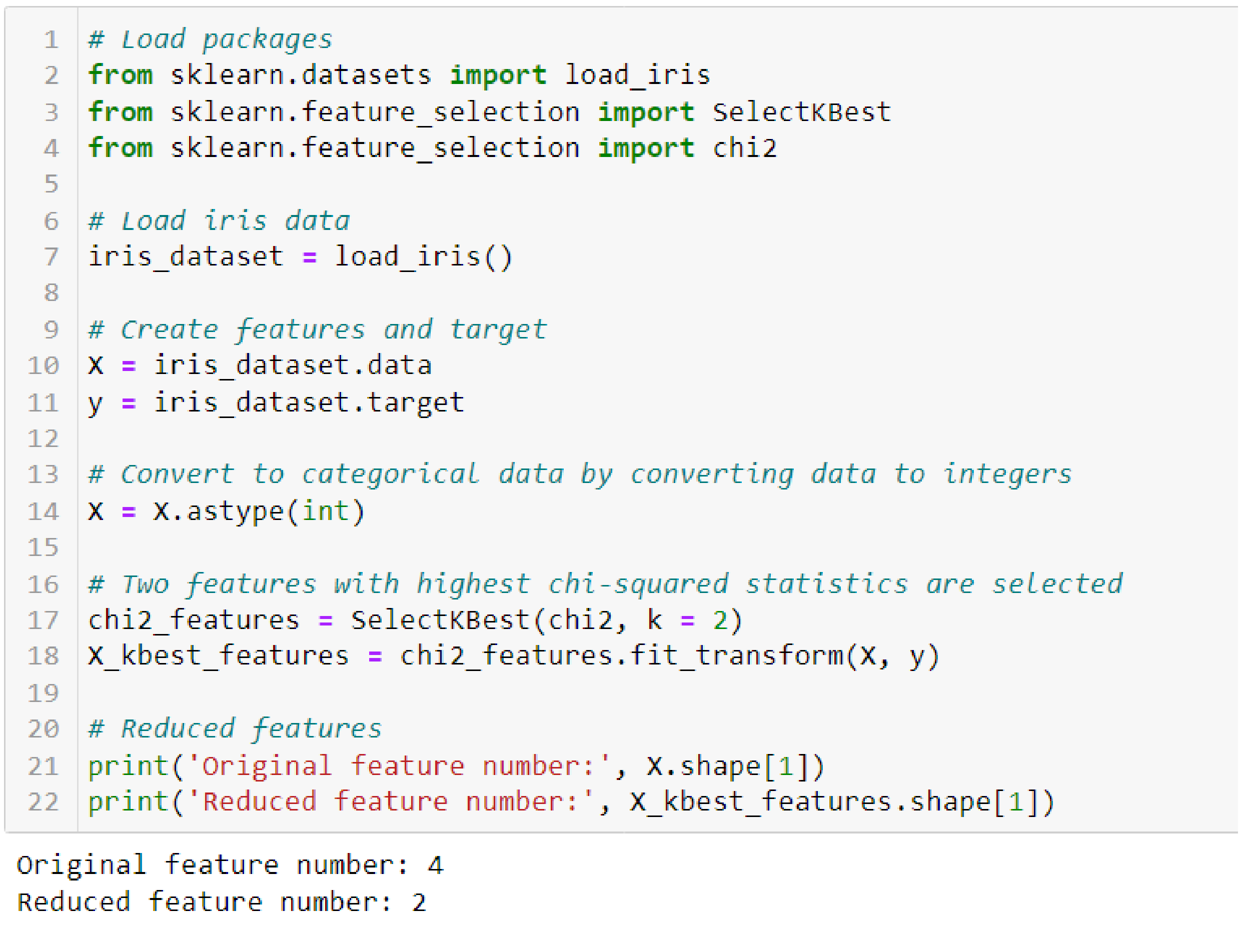
Input Variable	Output Variable	Feature Selection technique
Numerical	Numerical	○ Pearson's correlation coefficient (For linear Correlation). ○ Spearman's rank coefficient (for non-linear correlation).
Numerical	Categorical	○ ANOVA correlation coefficient (linear). ○ Kendall's rank coefficient (nonlinear).
Categorical	Numerical	○ Kendall's rank coefficient (linear). ○ ANOVA correlation coefficient (nonlinear).
Categorical	Categorical	○ Chi-Squared test (contingency tables). ○ Mutual Information.

4.3 Common methods for feature selection are:

4.3.1 Univariate Selection

Statistical tests can help to select independent features that have the strongest relationship with the target feature in your dataset. For example, the chi-squared test.

The Scikit-learn library provides the **SelectKBest** class that can be used with a suite of different statistical tests to select a specific number of features.
In the following example we use the **SelectKBest** class with the chi-squired test to find best feature for the Iris dataset:



```python
# Load packages
from sklearn.datasets import load_iris
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# Load iris data
iris_dataset = load_iris()

# Create features and target
X = iris_dataset.data
y = iris_dataset.target

# Convert to categorical data by converting data to integers
X = X.astype(int)

# Two features with highest chi-squared statistics are selected
chi2_features = SelectKBest(chi2, k = 2)
X_kbest_features = chi2_features.fit_transform(X, y)

# Reduced features
print('Original feature number:', X.shape[1])
print('Reduced feature number:', X_kbest_features.shape[1])
```

```
Original feature number: 4
Reduced feature number: 2
```

As you can see chi-squared test helps us to select **two important independent features** out of the original 4 that have the strongest relationship with the target feature. You can learn more about chi-squared test here: "A Gentle Introduction to the Chi-Squared Test for Machine Learning".

4.3.2 Feature Importance

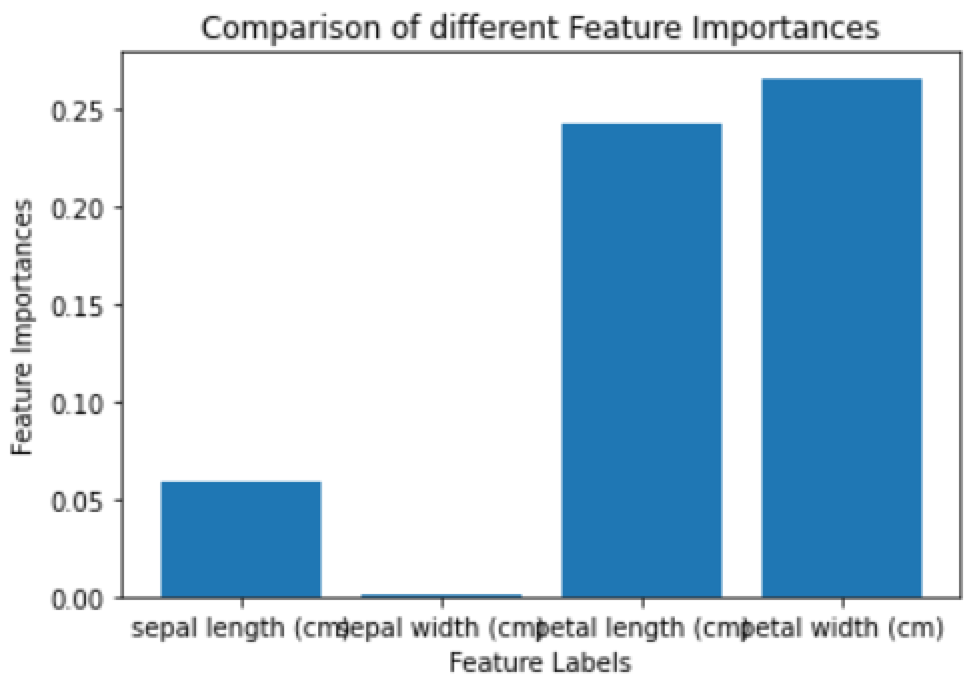
Feature importance gives you a score for each feature of your data. The higher the score, the more **important** or **relevant** that feature is to your target feature.

Feature importance is an inbuilt class that comes with tree-based classifiers such as:

- Random Forest Classifiers
- Extra Tree Classifiers

In the following example, we will train the extra tree classifier into the iris dataset and use the inbuilt class **.feature_importances_** to compute the importance of each feature:

```
# Load libraries
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier

# Load iris data
iris_dataset = load_iris()

# Create features and target
X = iris_dataset.data
y = iris_dataset.target

# Convert to categorical data by converting data to integers
X = X.astype(int)

 # Building the model
extra_tree_forest = ExtraTreesClassifier(n_estimators = 5,
                                        criterion ='entropy', max_features = 2)

# Training the model
extra_tree_forest.fit(X, y)

# Computing the importance of each feature
feature_importance = extra_tree_forest.feature_importances_

# Normalizing the individual importances
feature_importance_normalized = np.std([tree.feature_importances_ for tree in
                                        extra_tree_forest.estimators_],
                                        axis = 0)

# Plotting a Bar Graph to compare the models
plt.bar(iris_dataset.feature_names, feature_importance_normalized)
plt.xlabel('Feature Labels')
plt.ylabel('Feature Importances')
plt.title('Comparison of different Feature Importances')
plt.show()
```



Important features

The above graph shows that the most important features are ***petal length (cm)*** and ***petal width (cm)***, and that the least important feature is ***sepal width (cms)***. This means you can use the most important features to train your model and get best performance.

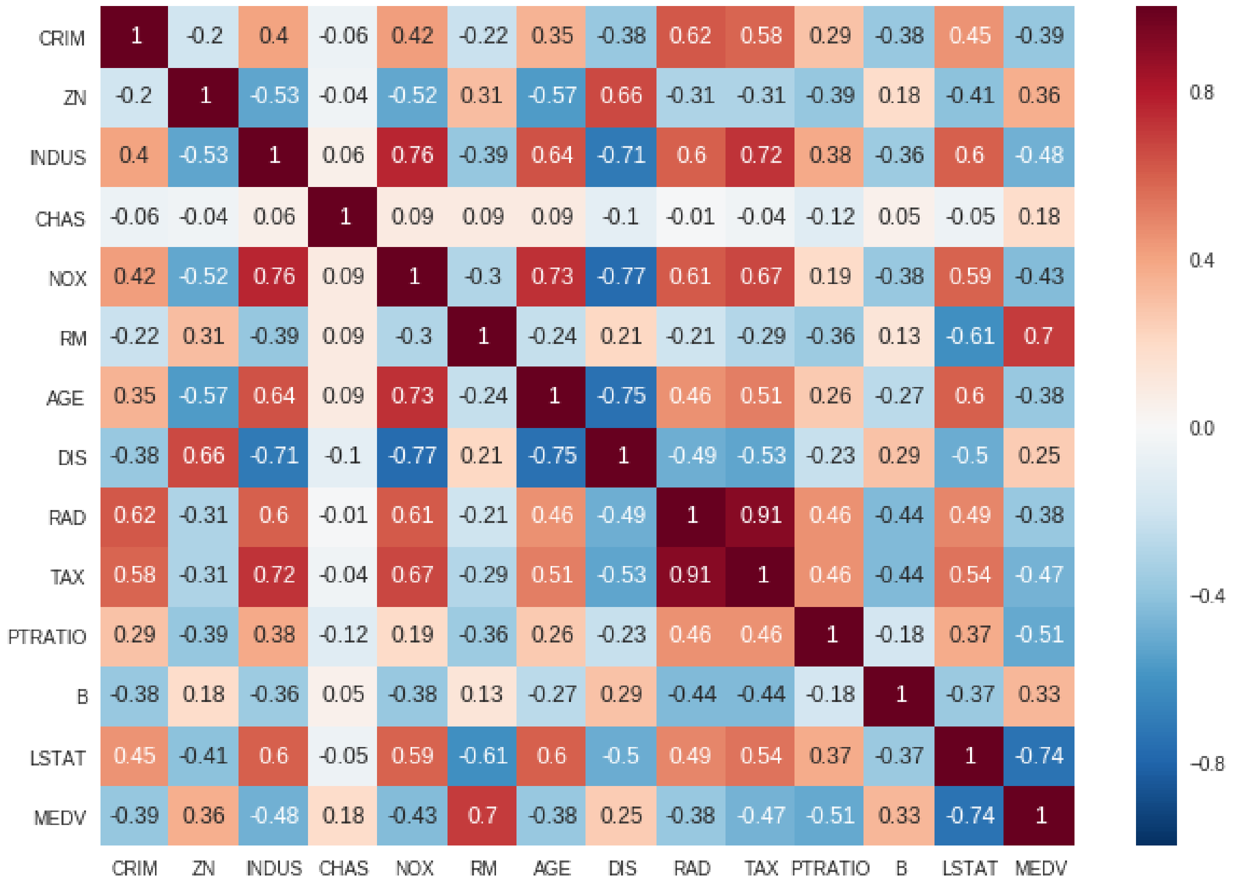
4.3.3 Correlation Matrix Heatmap

Correlation shows how the features are related to each other or the target feature.

Correlation can be positive (an increase in one value of the feature increases the value of the target variable) or negative (an increase in one value of the feature decreases the value of the target variable).

In the following example, we will use the Boston house prices dataset from the Scikit-learn library and the **corr()** method from pandas to find the pairwise correlation of all features in the dataframe:

```python
# Load libraries
from sklearn.datasets import load_boston
import matplotlib.pyplot as plt
import seaborn as sns

# load boston data
boston_dataset = load_boston()

# create a daframe for boston data
boston = pd.DataFrame(boston_dataset.data, columns=boston_dataset.feature_names)

# Convert to categorical data by converting data to integers
#X = X.astype(int)

#ploting the heatmap for correlation
ax = sns.heatmap(boston.corr().round(2), annot=True)
```



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
CRIM	1	-0.2	0.4	-0.06	0.42	-0.22	0.35	-0.38	0.62	0.58	0.29	-0.38	0.45	-0.39
ZN	-0.2	1	-0.53	-0.04	-0.52	0.31	-0.57	0.66	-0.31	-0.31	-0.39	0.18	-0.41	0.36
INDUS	0.4	-0.53	1	0.06	0.76	-0.39	0.64	-0.71	0.6	0.72	0.38	-0.36	0.6	-0.48
CHAS	-0.06	-0.04	0.06	1	0.09	0.09	0.09	-0.1	-0.01	-0.04	-0.12	0.05	-0.05	0.18
NOX	0.42	-0.52	0.76	0.09	1	-0.3	0.73	-0.77	0.61	0.67	0.19	-0.38	0.59	-0.43
RM	-0.22	0.31	-0.39	0.09	-0.3	1	-0.24	0.21	-0.21	-0.29	-0.36	0.13	-0.61	0.7
AGE	0.35	-0.57	0.64	0.09	0.73	-0.24	1	-0.75	0.46	0.51	0.26	-0.27	0.6	-0.38
DIS	-0.38	0.66	-0.71	-0.1	-0.77	0.21	-0.75	1	-0.49	-0.53	-0.23	0.29	-0.5	0.25
RAD	0.62	-0.31	0.6	-0.01	0.61	-0.21	0.46	-0.49	1	0.91	0.46	-0.44	0.49	-0.38
TAX	0.58	-0.31	0.72	-0.04	0.67	-0.29	0.51	-0.53	0.91	1	0.46	-0.44	0.54	-0.47
PTRATIO	0.29	-0.39	0.38	-0.12	0.19	-0.36	0.26	-0.23	0.46	0.46	1	-0.18	0.37	-0.51
B	-0.38	0.18	-0.36	0.05	-0.38	0.13	-0.27	0.29	-0.44	-0.44	-0.18	1	-0.37	0.33
LSTAT	0.45	-0.41	0.6	-0.05	0.59	-0.61	0.6	-0.5	0.49	0.54	0.37	-0.37	1	-0.74
MEDV	-0.39	0.36	-0.48	0.18	-0.43	0.7	-0.38	0.25	-0.38	-0.47	-0.51	0.33	-0.74	1

The correlation coefficient ranges from -1 to 1. If the value is close to 1, it means that there is a strong positive correlation between the two features. When it is close to -1, the features have a strong negative correlation.

In the figure above, you can see that the **TAX** and **RAD** features have a s*trong positive correlation* and the **DIS** and **NOX** features have a *strong negative correlation.*

If you find out that there are some features in your dataset that are correlated to each other, means that they convey the same information. It is recommended to remove one of them.

4.4 Benefits of Feature Selection:

1. **Reduces Overfitting:** By using only the most relevant features, the model can generalize better to new data.
2. **Improves Model Performance:** Selecting the right features can improve the accuracy, precision, and recall of the model.
3. **Decreases Computational Costs:** A smaller number of features requires less computation and storage resources.
4. **Improves Interpretability:** By reducing the number of features, it is easier to understand and interpret the results of the model.

5 Feature Scaling

Feature Scaling is the process of transforming the features so that they have a similar scale. This is important in machine learning because the scale of the features can affect the performance of the model.

5.1 Types of Feature Scaling:

5.1.1 Min-Max Scaling:

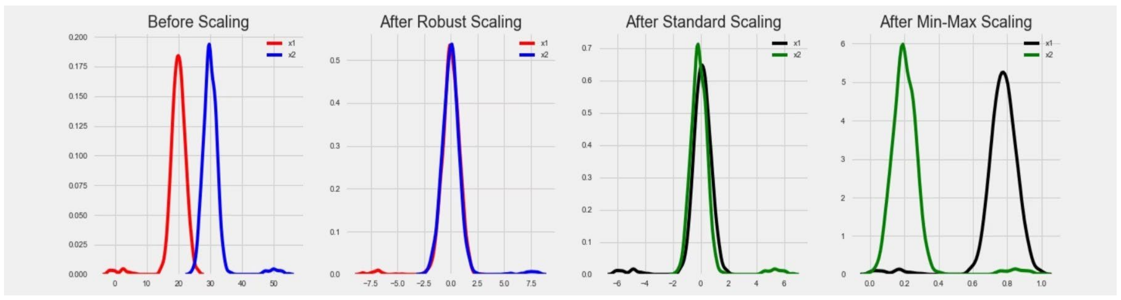
Rescaling the features to a specific range, such as between 0 and 1, by subtracting the minimum value and dividing by the range.

5.1.2 Standard Scaling:

Rescaling the features to have a mean of 0 and a standard deviation of 1 by subtracting the mean and dividing by the standard deviation.

5.1.3 Robust Scaling:

Rescaling the features to be robust to outliers by dividing them by the interquartile range.



.

5.1.4 Comparison

Scaling Technique	Suitable for Algorithms	Interpretability	Handling Outliers	Distribution	Suitable for Non-Gaussian Data
Min-Max Scaling	KNN, Gradient Descent	May lose interpretability	Sensitive	Preserves shape	Yes
Standardization	Gaussian-based, Gradient Descent	Retains interpretability	Sensitive	Shifts & scales	Yes
Robust Scaling	KNN, Clustering	Retains interpretability	Less sensitive	Preserves shape	Yes

Max Absolute Scaling	X_max		Gradient Descent, Distance-based	Retains interpretability	Less sensitive
Quantile Transformation	Gaussian-based	Retains interpretability	Not directly	Changes distribution	Yes
Log Transformation	Linear Regression, Clustering	Retains interpretability	Not directly	Shifts & scales	Yes
Power Transformation	Gaussian-based (Box-Cox)	Retains interpretability	Not directly	Changes distribution	Yes (Box-Cox)
Unit Vector Scaling	KNN, Clustering	Retains interpretability	Not applicable	Not applicable	Yes

5.2 Benefits of Feature Scaling:

1. **Improves Model Performance:** By transforming the features to have a similar scale, the model can learn from all features equally and avoid being dominated by a few large features.
2. **Increases Model Robustness:** By transforming the features to be robust to outliers, the model can become more robust to anomalies.
3. **Improves Computational Efficiency:** Many machine learning algorithms, such as k-nearest neighbors, are sensitive to the scale of the features and perform better with scaled features.
4. **Improves Model Interpretability:** By transforming the features to have a similar scale, it can be easier to understand the model's predictions.

6 Few Best Feature Engineering Tools

There are many tools which will help you in automating the entire feature engineering process and producing a large pool of features in a short period of time for both classification and regression tasks. So let's have a look at some of the features engineering tools.

6.1 FeatureTools

Featuretools is a framework to perform automated feature engineering. It excels at transforming temporal and relational datasets into feature matrices for machine learning. Featuretools integrates with the machine learning pipeline-building tools you already have. In a fraction of the time it would take to do it manually, you can load in pandas dataframes and automatically construct significant features.

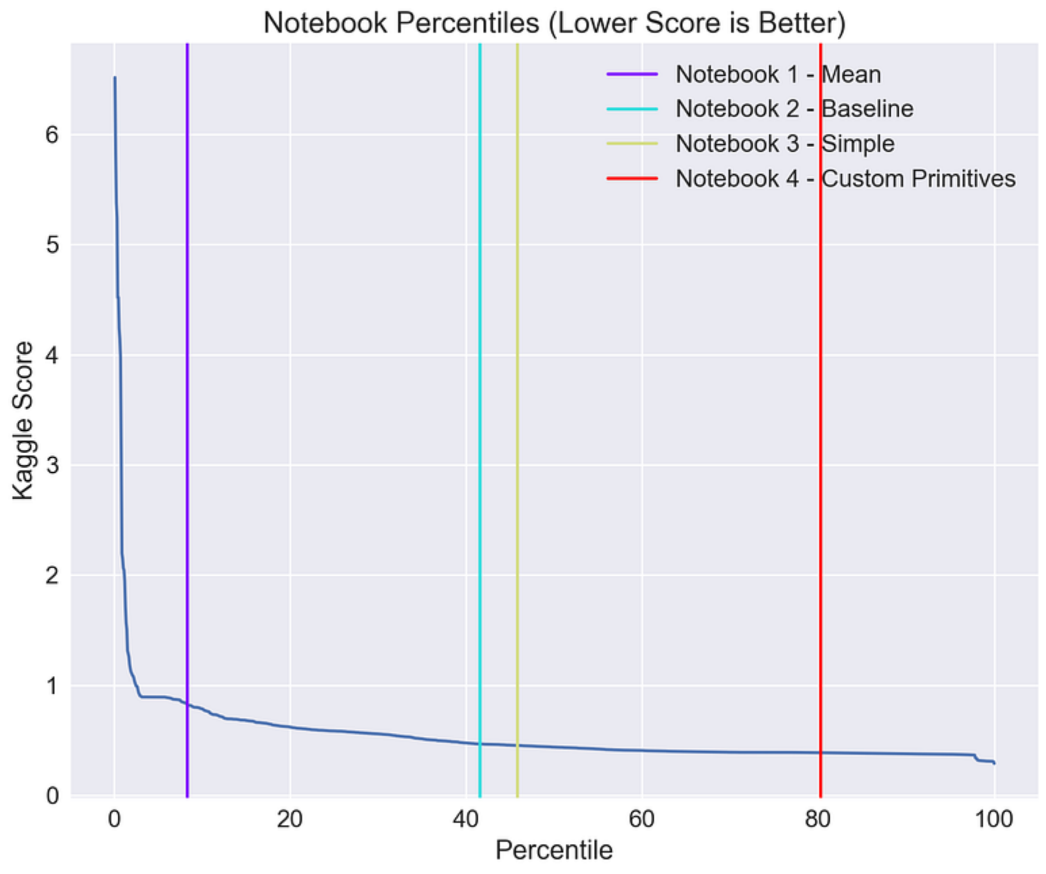


Image Source

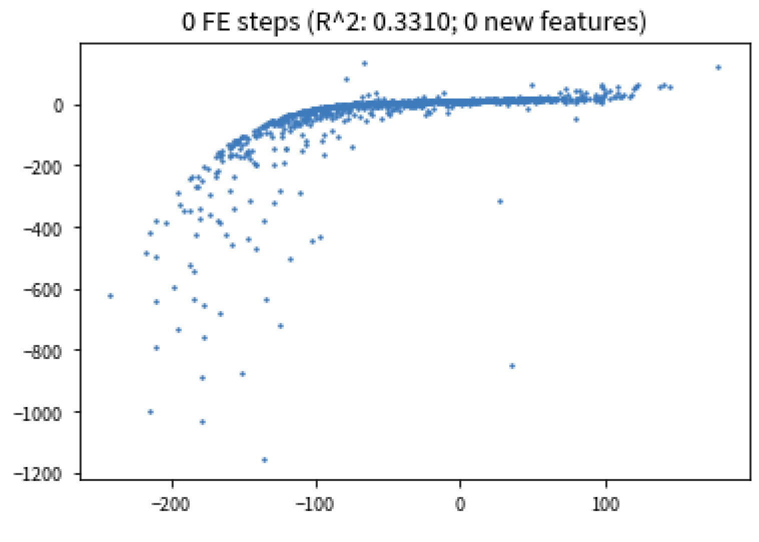
FeatureTools Summary

- Easy to get started, good documentation and community support
- It helps you construct meaningful features for machine learning and predictive modelling by combining your raw data with what you know about your data.
- It provides APIs to verify that only legitimate data is utilised for calculations, preventing label leakage in your feature vectors.
- Featuretools includes a low-level function library that may be layered to generate features.
- Its AutoML library(EvalML) helps you build, optimize, and evaluate machine learning pipelines.
- Good at handling relational databases.

Learn More About FeatureTools.

6.2 AutoFeat

AutoFeat helps to perform Linear Prediction Models with Automated Feature Engineering and Selection. AutoFeat allows you to select the units of the input variables in order to avoid the construction of physically nonsensical features.



Source

AutoFeat Summary

- AutoFeat can easily handle categorical features with One hot encoding.
- The AutoFeatRegressor and AutoFeatClassifier models in this package have a similar interface to scikit-learn models

- General purpose automated feature engineering which is Not good at handling relational data.
- It is useful in logistical data

Learn More About AutoFeat.

6.3 TsFresh

tsfresh is a python package. It calculates a huge number of time series characteristics, or features, automatically. In addition, the package includes methods for assessing the explanatory power and significance of such traits in regression and classification tasks.

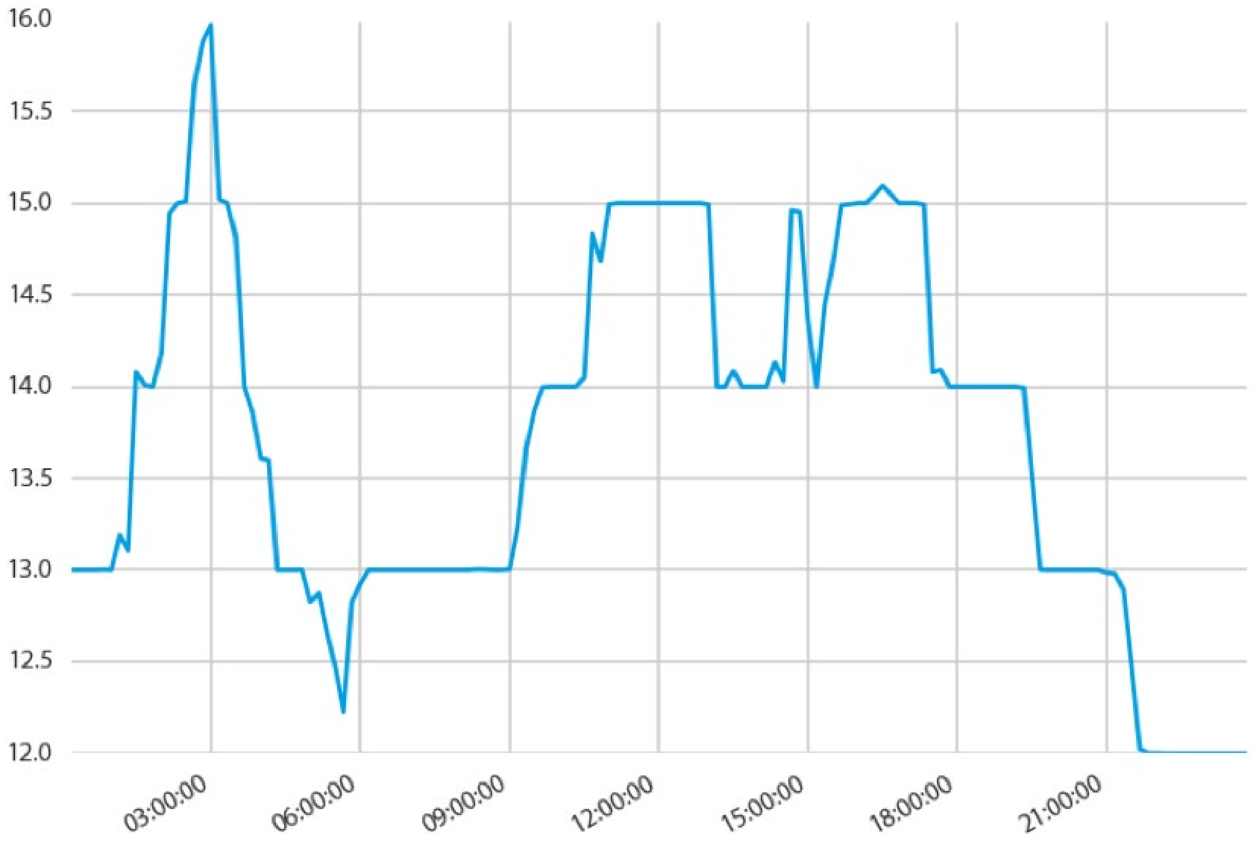


Image Source

TsFresh Summary

- It is Best open source python tool available for time series classification and regression.
- It helps to extract things such as the number of peaks, average value, maximum value, time reversal symmetry statistic, etc.
- It can be integrated with FeatureTools.

Learn More About TsFresh .

6.4 OneBM

OneBM interacts directly with a database's raw tables. It slowly joins the tables, taking different paths on the relational tree. It recognises simple data types (numerical or categorical) and complicated data types (set of numbers, set of categories, sequences, time series, and texts) in the joint results and applies pre-defined feature engineering approaches to the supplied types.

- Both relational and non-relational data are supported.
- When compared to FeatureTools, it generates both simple and complicated features.
- It was put to the test in Kaggle competitions, and it outperformed state-of-the-art models.

Learn More About OneBM

6.5 ExploreKit

Based on the idea that extremely informative features are typically the consequence of manipulating basic ones, ExploreKit identifies common operators to alter each feature independently or combine multiple of them. Instead of running feature selection on all developed features, which can be quite huge, meta learning is used to rank candidate features.

Learn More About ExploreKit

6.6 H2O.ai

H2O.ai is an open-source machine learning platform that includes feature engineering as one of its capabilities. It provides a range of automated feature engineering techniques, such as feature scaling, imputation, and encoding, as well as manual feature engineering capabilities for more advanced users. Some of its features include:

- Automatic and manual feature engineering options.
- Support for structured and unstructured data, including text and image data.
- Integration with popular data sources, such as CSV files and databases.
- Interactive visualization of the generated features and models.
- Collaboration and sharing tools for teams working on machine learning projects.

6.7 Alteryx

Alteryx is a data preparation and automation tool that includes feature engineering as one of its features. It provides a visual interface for creating data pipelines that can extract, transform, and generate features from multiple data sources. Some of its features include:

- Support for handling structured and unstructured data.
- Integration with popular data sources, such as Excel and databases.
- Pre-built tools for feature extraction and transformation.
- Support for custom scripting and code integration.
- Collaboration and sharing tools for teams working on data projects.

6.8 DataRobot

DataRobot is a machine learning automation platform that includes feature engineering as one of its capabilities. It uses automated machine learning techniques to generate new features and select the best combination of features and models for a given dataset. Some of its features include:

- Automatic feature engineering using machine learning algorithms.
- Support for handling time-dependent and text data.
- Integration with popular Python libraries, such as pandas and scikit-learn.
- Interactive visualization of the generated models and features.
- Collaboration tools for teams working on machine learning projects.

6.9 TPOT

TPOT (Tree-based Pipeline Optimization Tool) is an automated machine learning tool that includes feature engineering as one of its components. It uses genetic programming to search for the best combination of features and machine learning algorithms for a given dataset. Some of its features include:

- Automatic feature selection and transformation.
- Support for multiple types of machine learning models, including regression, classification, and clustering.
- Ability to handle missing data and categorical variables.
- Integration with popular Python libraries, such as scikit-learn and pandas.
- Interactive visualization of the generated pipelines.

6.10 Feature Selector

As the name suggests, Feature Selector is a Python library for choosing features. It determines attribute significance based on missing data, single unique values, collinear or insignificant features. For that, it uses “lightgbm” tree-based learning methods. The package also includes a set of visualization techniques that can provide more information about the dataset.

6.11 PyCaret

PyCaret is a Python-based open-source library. Although it is not a dedicated tool for automated feature engineering, it does allow for the automatic generation of features before model training. Its advantage is that it lets you replace hundreds of code lines with just a handful, thus increasing productivity and exponentially speeding up the experimentation cycle.

Some other useful tools for feature engineering include:

- the NumPy library with numeric and matrix operations;
- Pandas where you can find the DataFrame, one of the most important elements of data science in Python;
- Scikit-learn framework, a general ML package with multiple models and feature transformers;
- Matplotlib and Seaborn that will help you with plotting and visualization

7 Interview Questions Feature Engineering:

1.How to perform Feature Engineering on Unknown features?

Feature engineering for unknown features involves several steps. Begin by understanding the domain and context to infer potential relevance. Analyze correlations with existing features and the target variable. Utilize techniques like binning, scaling, or encoding categorical data. Impute missing values using mean, median, or advanced methods. Create new features through transformations like logarithms or interactions. Leverage domain expertise to derive meaningful features. Regularly validate and refine the engineered features with cross-validation. Employ dimensionality reduction if needed. Lastly, monitor model performance to assess the impact of engineered features and iterate as necessary.

2.Can you give an example of a real-world feature engineering problem and how you tackled it?
Example: Predicting customer churn in a telecom company. I identified relevant features like call duration, contract length, and usage patterns. I transformed categorical features using one-hot encoding and created interaction features between call duration and contract length. Afterward, I applied feature scaling and selected the most important features using feature importance scores from a random forest model.

3.**How does feature engineering impact model interpretability?**
Well-engineered features can enhance model interpretability by presenting clearer relationships between features and the target variable, making it easier to understand the model's decision-making process.

4.**How can you deal with imbalanced classes in classification tasks during feature engineering?**
Techniques include oversampling the minority class, undersampling the majority class, generating synthetic samples (SMOTE), or using specialized algorithms designed for imbalanced data.

5.**How can domain knowledge aid in feature engineering?**
Domain knowledge helps in identifying relevant features, understanding their relationships, and creating meaningful new features that capture the underlying characteristics of the data.

6.What is feature extraction?
Feature extraction involves transforming raw data into a lower-dimensional representation using techniques like Fourier transforms or extracting meaningful features from text using techniques like TF-IDF.

7.How do you perform End of Tail Imputation?

End of Tail Imputation involves replacing extreme outliers in a feature with values at the tail of the distribution, typically beyond a certain quantile (e.g., 95th percentile). This helps mitigate the impact of outliers on the model while preserving the distribution's general shape.

8.Can you explain what dimensionality reduction is?

Dimensionality reduction is the process of reducing the number of features in a dataset while still retaining as much information as possible. This can be done through a variety of methods, such as feature selection or feature extraction.

7.1 Feature Creation:

1.How can you create interaction features, and why are they valuable?

Interaction features are created by combining two or more existing features. They capture synergistic relationships that individual features might not capture, providing deeper insights for the model.

2.What are some common techniques for creating new features?

Some common techniques include polynomial expansion, interaction features, binning, one-hot encoding, and creating statistical summaries such as mean, median, and standard deviation.

3.Why might creating too many features lead to overfitting?

Creating too many features can lead to overfitting because the model may start to memorize noise in the training data, causing it to perform poorly on new, unseen data. It's important to strike a balance between creating informative features and avoiding excessive complexity.

4.What is the curse of dimensionality, and how does it relate to feature creation?

The curse of dimensionality refers to the challenges that arise when dealing with high-dimensional data. As the number of features increases, the amount of data needed to effectively train a model grows exponentially. Feature creation can exacerbate this issue by introducing more dimensions, so careful selection and dimensionality reduction techniques are crucial.

5.How would you validate the effectiveness of the features you've created?

Feature effectiveness can be validated through techniques like cross-validation, where the model is trained and evaluated on different subsets of the data. Additionally, analyzing feature importance scores from models like decision trees or random forests can provide insights into which features contribute most to the model's performance.

7.2 Feature Transformation:

1.What are some good rules of thumb for applying transformations to numeric data?

A good rule of thumb is to always start with the simplest possible transformation and then move on to more complex ones if needed. For example, if you have a dataset with a lot of outliers, you might start by trying to remove them. If that doesn't work, you could try transforming the data to make it more normally distributed. And so on.

2.How do you decide which feature transformation technique to apply to a specific dataset?

The choice depends on the distribution of the data, the type of relationships you're trying to capture, and the characteristics of the machine learning algorithm you intend to use.

3.Can you explain how the quantile transformation works?

The quantile transformation maps the data to follow a specified probability distribution, often the normal distribution. It helps in achieving more Gaussian-like data and stabilizing variance.

4.What is the purpose of applying polynomial transformations to features?

Polynomial transformations involve raising features to different powers to capture nonlinear relationships between variables. This can make the model more flexible and able to fit complex data patterns.

5.What role does the data distribution play when choosing a feature transformation technique?

The data distribution influences the choice of transformation technique. For example, if the data has a skewed distribution, transformations that reduce skewness might be preferred.

7.3 Feature Selection:

1.What's the difference between *Forward Feature Selection* and *Backward Feature Selection*?

- **Forward Feature selection** involves evaluating *each individual feature*, starting from the feature with a **higher score** and then adding one feature at a time so that the extended subset improves on the *selected metric*. We can keep adding features as long as the selected set of features **reaches a threshold** for the value of the metric, which is selected according to the context of the problem or using the **random feature technique** to obtain a *cutoff value*.

- **Backward Feature selection**, on the other hand, involves starting from the *full set* and evaluates the metric for the set without each feature. At each stage, the set is shrunk by the feature that produces the smallest reduction to the target metric. We can keep dropping features as long as the performance improves or stays the same. That is, **stop when it gets worse**.

2.Name some benefits of *Feature Selection* .

- Many features and low samples/features ratios will introduce noise into your dataset. In such a case your classification algorithm is likely to overfit and give you a false feeling of good performance.

- Reducing the number of features will reduce the running time in the later stages. That in turn will enable you to use algorithms of higher complexity, search for more hyperparameters or do more evaluations.

- A smaller set of features is more comprehendible to humans. That will enable you to focus on the main sources of predictability and do more exact feature engineering. If you will have to explain your model to a client, you are better at presenting a model with 5 features than a model with 200 features.

3.How do you use the *F-test* to *select features*?

F-Test is a **statistical test** used to compare models and check if the difference is *significant* between them. The hypothesis testing uses a model **X** and **Y**, where:

- **X** is a model created by just a *constant*.
- **Y** is the model created by a *constant* and a *feature*.

We calculate the *least square errors* in both models and compare and check if the difference in errors between model **X** and **Y** are *significant* or *introduced by chance*. This significance returns the importance of each feature, so we select the features that return **high F-values** and use those for further analysis.

4.Can we use *PCA* for feature selection?

Feature selection refers to **choosing a subset** of the features from the complete set of features.

In **PCA**, we obtain **Principal Components axis**, this is a **linear combination** of **all** the **original set** of feature variables which defines a new set of axes that explain most of the **variations** in the data.
Therefore while **PCA** performs well in many practical settings, it does not result in the development of a model that relies upon a **small set** of the original features and so for this reason, **PCA** is **not** a feature selection technique.

5.Explain the concept of regularization and its role in feature selection.

Regularization techniques, like L1 regularization (Lasso), penalize the magnitude of feature coefficients. This encourages the model to automatically perform feature selection by shrinking less important feature coefficients towards zero.

7.4 Feature Extraction:

1.What are some common techniques for feature extraction?

Common techniques include Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), t-Distributed Stochastic Neighbor Embedding (t-SNE), and autoencoders in neural networks.

2. When might non-linear dimensionality reduction techniques like t-SNE be preferred over linear techniques like PCA?

Non-linear techniques like t-SNE are preferred when the underlying relationships between features are complex and cannot be captured well by linear transformations. t-SNE is particularly useful for data visualization and exploring local structure.

3.Can you explain the curse of dimensionality and how feature extraction techniques help mitigate it?

The curse of dimensionality refers to the challenges posed by high-dimensional data, such as increased computational requirements and sparse data distribution. Feature extraction techniques like PCA can help mitigate this by reducing dimensionality while retaining most of the data's variance.

4.What are some potential downsides of using feature extraction techniques?

Feature extraction techniques can introduce complexity, require parameter tuning, and might not always improve model performance. Additionally, the interpretability of extracted features may be limited compared to using the original features.

7.5 Feature Scaling:

1.How does feature scaling affect algorithms like k-nearest neighbors (KNN) and support vector machines (SVM)?

Algorithms like KNN and SVM use distances between data points, which can be greatly influenced by feature scales. Scaling ensures that features contribute equally to the distance calculations.

2.What is the impact of not scaling features on algorithms like gradient descent?

Not scaling features in gradient descent can lead to slow convergence or even divergence. The algorithm might take larger steps in directions of features with larger scales, making the optimization process unstable.

3.Can you explain why scaling is crucial for clustering algorithms?

Clustering algorithms use distance metrics to group data points. Without proper scaling, features with larger scales may dominate the distance calculations, leading to biased clustering results.

4.How does feature scaling impact the interpretability of model coefficients?

Feature scaling doesn't affect the interpretability of the relationships between features and the target variable. However, coefficients will be more comparable when features are on the same scale.

5.Are there cases where feature scaling might not be necessary?

Some algorithms inherently handle different feature scales, like Naive Bayes or algorithms based on rules. However, applying feature scaling generally improves stability and convergence across various algorithms.