MLOps

Feature Engineering

“Designing Artificial Intelligence Systems

- Introduction to ML system design
- Data Engineering fundamentals
- Training data & Feature Engineering
- Model development and offline evaluation
- Model deployment & prediction services
- Introduction to MLOps

“Feature Engineering

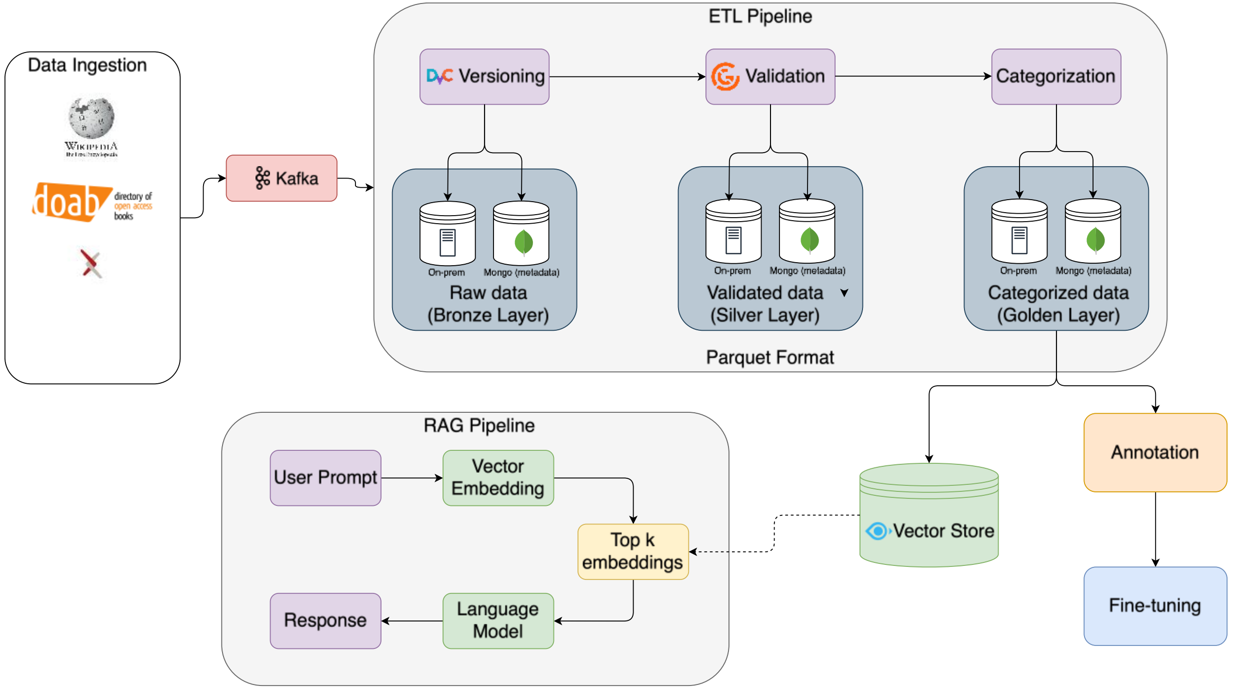


Figure 1. BrahmDAX pipeline, https://github.com/zenteiq-ai/BrahmDAX.

“Feature Engineering

- Learned Features Vs. Engineered Features
- Common Feature Handling Operations
- Data Leakage
- Engineering Good Features
- Summary

Feature Engineering

Lecture Summary

- Introduction to Feature Engineering
  - Importance in enhancing model performance
  - Role in extracting relevant information from raw data
  - Handling missing data and reducing dimensionality
  - Enabling model interpretability and improving robustness
  - Supporting domain knowledge incorporation and facilitating model deployment
- Learned vs. Engineered Features
  - Comparison between automatically derived and manually crafted features
  - Advantages and disadvantages of each approach
  - Importance of combining both for optimal

Feature Engineering

Lecture Summary

- Common Feature Engineering Operations
  - Handling missing values (deletion, imputation)
  - Scaling techniques (min-max, standardization, robust scaling)
  - Discretization methods
  - Encoding categorical features
  - Feature crossing and positional embeddings
- Data Leakage
  - Definition and causes of data leakage
  - Effects on model performance and evaluation
  - Prevention strategies and detection methods

Feature Engineering

Lecture Summary

- Engineering Good Features
  - Leveraging domain knowledge
  - Feature extraction and selection techniques
  - Transformation and encoding methods
  - Handling time series data Iterative process of feature engineering

Feature Engineering

Intended Learning Outcomes (ILOs)

- After completing this lecture, you should be able to:
  - Explain the importance of feature engineering in machine learning and its impact on model performance.
  - Differentiate between learned and engineered features, and understand when to apply each approach.
  - Apply common feature engineering operations, including handling missing values, scaling, discretization, and encoding categorical variables.
  - Identify potential sources of data leakage and implement strategies to prevent and detect it.
  - Design and implement effective feature engineering techniques tailored to specific problem domains and data types.

Feature Engineering

Importance

- Enhances Model Performance
  - Plays a crucial role in improving the performance of ML models
  - By **selecting**, **creating**, and **transforming** relevant features, we can provide valuable information to the model, enabling it to make more accurate predictions or classifications
- Extracts Relevant Information
  - Helps in extracting relevant information from raw data
  - By carefully selecting and transforming features
    - we can capture the underlying patterns and relationships in the data,
    - making it easier for the model to learn and make accurate predictions

Feature Engineering

Importance

- Handles Missing Data
  - Feature engineering techniques can effectively handle missing data
  - By imputing missing values or creating new features based on existing information, we can ensure that our models are not biased or inaccurate due to missing data points
- Reduces Dimensionality
  - Allows us to reduce the dimensionality of the data
  - By selecting or creating a subset of relevant features, we can eliminate noise or redundant information, making the model more efficient and less prone to overfitting

Feature Engineering

Importance

- Enables Model Interpretability
    - Helps in making models more interpretable
    - By transforming features into meaningful representations, we can understand the impact of different features on the model's output, gaining insights into the underlying factors driving the predictions
- Improves Robustness
    - Makes models more robust and generalizable
    - By creating features that capture different aspects of the problem, such as time-based trends or interactions between variables, we can improve the model's ability to handle new or unseen data

Feature Engineering

Importance

- Supports Domain Knowledge
  - Allows us to incorporate domain knowledge into the model
  - By leveraging our understanding of the problem domain, we can engineer features that are specifically tailored to the problem at hand, resulting in better model performance
- Facilitates Model Deployment
  - Well-engineered features simplify the deployment of machine learning models
  - By pre-processing and transforming features in a consistent manner, we can easily apply the same feature engineering techniques to new data during production, ensuring reliable and accurate predictions

Feature Engineering

Veracity Check

1. T/F: Feature engineering always reduces the dimensionality of the data.
2. T/F: Well-engineered features can make models more interpretable.
3. T/F: Feature engineering is only important for machine learning models, not for deep learning.
4. T/F: Incorporating domain knowledge is a key aspect of feature engineering.

Feature Engineering

Veracity Check

1. T/F: Feature engineering always reduces the dimensionality of the data.

   Answer: False. **Explanation**: While feature engineering can reduce dimensionality (e.g., through feature selection), it can also increase dimensionality through techniques like feature creation or crossing.

2. T/F: Well-engineered features can make models more interpretable.

   Answer: True. Explanation: Engineered features often have clear semantic meanings, making it easier to understand the model's decision-making process.

3. T/F: Feature engineering is only important for machine learning models, not for deep learning.

   Answer: False. Explanation: While deep learning can learn features automatically, feature engineering can still improve performance and efficiency, especially with limited data.

4. T/F: Incorporating domain knowledge is a key aspect of feature engineering.

   Answer: True. Explanation: Domain expertise helps create relevant features that capture important aspects of the problem, often leading to better model performance.

Feature Engineering

Self-assessment

1. Q: What is the primary goal of feature engineering in machine learning?

   A: The primary goal of feature engineering is to improve model performance by creating more informative and relevant features. **Explanation**: Feature engineering aims to transform raw data into a format that better represents the underlying problem to the model, thereby enhancing its ability to learn and make accurate predictions.

2. Q: How does feature engineering contribute to model interpretability?

   A: Feature engineering can create more meaningful and intuitive features, making it easier to understand the model's decision-making process. **Explanation**: By crafting features that have clear semantic meanings, we can more easily interpret how these features influence the model's predictions, enhancing overall model interpretability.

3. Q: In what way can feature engineering help handle missing data?

   A: Feature engineering techniques can be used to impute missing values or create new features that capture the pattern of missingness. **Explanation**: Methods like mean imputation, regression imputation, or creating binary indicators for missingness are all feature engineering techniques that address the challenge of missing data.

Feature Engineering

Self-assessment

4. Q: How does feature engineering support the incorporation of domain knowledge?

   A: Feature engineering allows domain experts to create features that capture important aspects of the problem based on their specialized knowledge. **Explanation**: Domain experts can use their understanding of the field to design features that represent key factors or relationships, which might not be immediately apparent from the raw data alone.

5. Q: What role does feature engineering play in dimensionality reduction?

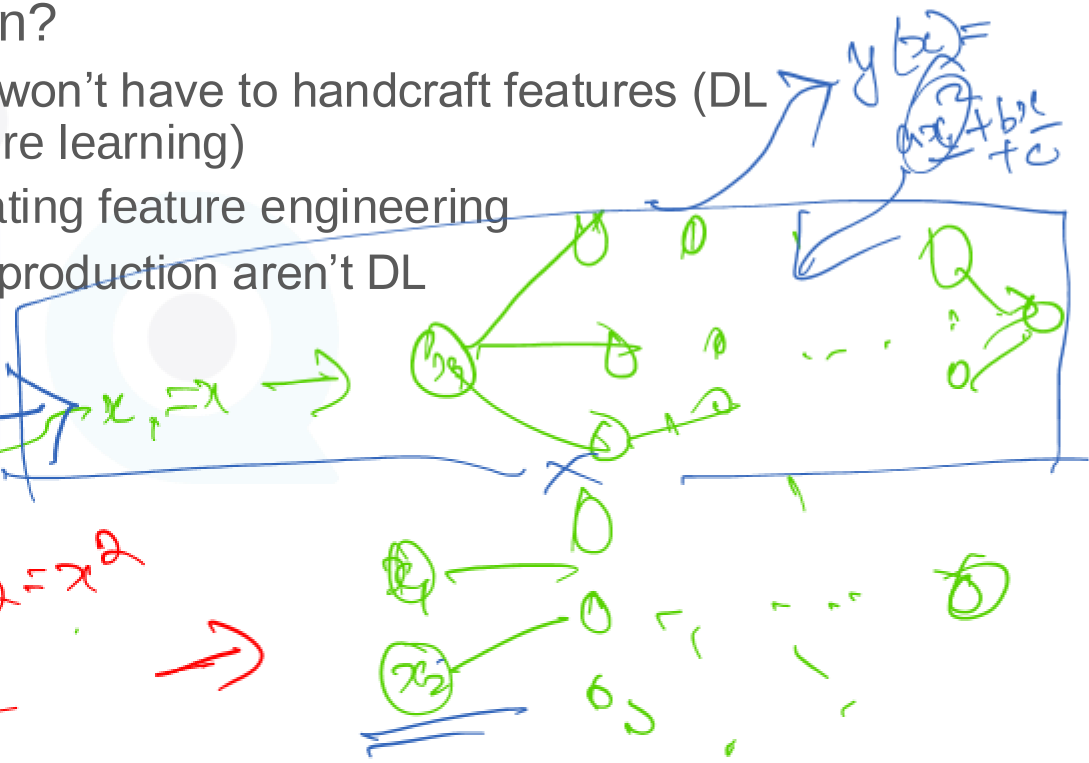
   A: Feature engineering can help reduce dimensionality by creating more compact representations of the data or by selecting the most relevant features. **Explanation**: Techniques like feature selection, principal component analysis (PCA), or creating composite features can all serve to reduce the number of dimensions while retaining important information.

“Learned Features Versus Engineered Features

Learned Features Vs. Engineered Features

Why do we need to worry?

- Doesn't DL promise to learn?
  - Promise of DL is that we won't have to handcraft features (DL is sometimes called feature learning)
  - In reality, far from automating feature engineering
  - Majority of ML models in production aren't DL



Engg features $x_1 = x$, $x_2 = x^2$

Learned Features Vs. Engineered Features

Difference

- Learned Features
  - Automatically derived by the machine learning model during the training process
  - Extracted directly from the raw data, without any explicit feature engineering by SMEs
- Engineered Features
  - Carefully designed and created by DEs based on their domain knowledge and understanding of the problem
  - These features are typically derived from the raw data through various techniques such as transformations, aggregations, or combinations of existing features

$x_1 = x$

$x_2 = x^2$

Learned Features Vs. Engineered Features

Example: Learned vs. Engineered Features in Image Classification

Let's consider an image classification task where the goal is to distinguish between images of cats and dogs.

- **Engineered Features:** In the engineered features approach, a computer vision expert might manually design features based on their knowledge of what distinguishes cats from dogs.
  - Some examples of engineered features could include:
    - **Ear shape**: A feature that measures the pointiness of ears (cats often have more triangular ears).
    - **Snout length:** A measurement of the length of the nose and mouth area (dogs typically have longer snouts).
    - **Eye shape:** A feature capturing the shape of the eyes (cats often have more almond-shaped eyes).
    - **Tail-to-body ratio:** The ratio of tail length to body length (cats often have longer tails relative to their body).
    - **Texture analysis**: Measures of fur texture (cats and dogs often have different fur patterns).

Learned Features Vs. Engineered Features

Example: Learned vs. Engineered Features in Image Classification

- **Learned Features:** In contrast, a deep learning approach using Convolutional Neural Networks (CNNs) would learn features automatically from the raw pixel data.
    - The process might look like this:
        - **Low-level features:** The initial layers might learn to detect edges, corners, and simple textures.
        - **Mid-level features:** Subsequent layers could combine these to recognize more complex shapes like circular eyes or pointy ears.
        - **High-level features:** Deeper layers might learn to identify entire body parts or characteristic patterns.
    - The exact nature of these learned features is not explicitly programmed but emerges during the training process.
- In practice, a combination of both approaches can be powerful.
    - For example, you might use a CNN to learn features automatically, but also include some engineered features (like the tail-to-body ratio) as additional inputs to the model's final layers.

Learned Features Vs. Engineered Features

Learned Features

- Offer flexibility as they can capture complex patterns and relationships that may not be immediately obvious to human experts
- Allows the model to adapt to the data and learn from the intricate interactions present in the dataset
- Can have high dimensionality since they are derived directly from the raw data
- Pose challenges in terms of computational complexity and model interpretability, especially when dealing with large datasets

Engineered Features

- Tend to be more interpretable as they are derived based on human intuition and understanding
- Often have a clear semantic meaning and can provide insights into the underlying factors driving the model's predictions
- Can help improve data efficiency
- By transforming the raw data into more informative representations, engineered features can reduce the amount of data needed for accurate modeling, making the process more efficient

Learned Features Vs. Engineered Features

Engineered Features

- Can be more transferable across different tasks or domains
- Once a set of well-engineered features is designed and validated for a specific problem, they can often be reused or adapted for related tasks, saving time and effort in feature engineering
- Leverage domain knowledge and expertise, allowing human experts to incorporate prior knowledge into the modeling process
- Can be particularly valuable in domains where interpretability, ***regulatory compliance***, or the ability to explain model decisions are important
- Combination of learned and engineered features can often result in the best performance
- The use of both learned and engineered features is often an iterative process. Initially, we may start with engineered features, then gradually introduce learned features as the model becomes more proficient at capturing relevant patterns in the data

Learned Features Vs. Engineered Features

Veracity Check

1. T/F: Learned features are always more effective than engineered features.
2. T/F: Engineered features tend to be more interpretable than learned features.
3. T/F: The combination of learned and engineered features often results in the best model performance.
4. T/F: Engineered features require less computational resources than learned features.

Learned Features Vs. Engineered Features

Veracity Check

1. T/F: Learned features are always more effective than engineered features.

   Answer: False. Explanation: The effectiveness depends on the specific problem, data availability, and model complexity. Sometimes engineered features outperform learned ones.

2. T/F: Engineered features tend to be more interpretable than learned features.

   Answer: True. Explanation: Engineered features are created based on human understanding, making them generally more interpretable than abstract learned features.

3. T/F: The combination of learned and engineered features often results in the best model performance.

   Answer: True. Explanation: Combining both approaches can leverage the strengths of human expertise and machine learning capabilities, often leading to optimal performance.

4. T/F: Engineered features require less computational resources than learned features.

   Answer: True. Explanation: Engineered features are typically pre-computed, while learning features requires additional computational resources during model training.

Learned Features Vs. Engineered Features

Self-assessment

1. Q: What is the main difference between learned and engineered features?

   A: Engineered features are manually crafted based on domain knowledge, while learned features are automatically extracted by the model during training. **Explanation**: Engineered features require human intervention and expertise, whereas learned features are derived by the model itself, often through techniques like deep learning.

2. Q: In what situations might engineered features be preferable to learned features?

   A: Engineered features might be preferable when dealing with limited data, when interpretability is crucial, or when there's strong domain knowledge available. **Explanation**: With small datasets, engineered features can encode important domain knowledge that the model might not learn on its own. They're also typically more interpretable and can leverage expert insights.

3. Q: How do learned features contribute to the promise of end-to-end learning in deep learning?

   A: Learned features enable models to automatically extract relevant representations from raw data, potentially capturing complex patterns without manual feature engineering. **Explanation**: This capability allows deep learning models to learn directly from raw inputs (like pixels or text) to outputs, without the need for extensive manual feature design.

Learned Features Vs. Engineered Features

Self-assessment

4. Q: What is a potential drawback of relying solely on learned features?

   A: Relying solely on learned features may require larger amounts of data and computational resources, and can sometimes result in less interpretable models. **Explanation**: Learning features from scratch often requires more data to capture relevant patterns, and the resulting features may be abstract and difficult to interpret compared to manually engineered features.

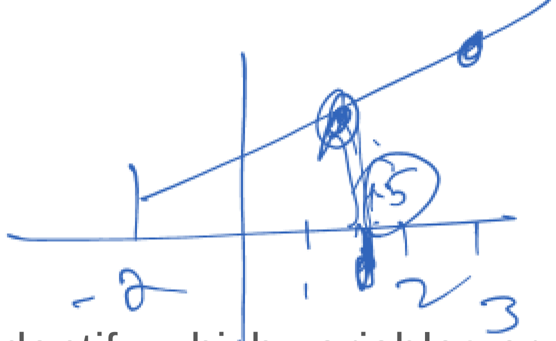
5. Q: How can the combination of learned and engineered features be beneficial?

   A: Combining learned and engineered features can leverage both human expertise and the model's ability to discover patterns, often leading to better overall performance. **Explanation**: This approach allows the model to benefit from domain knowledge encoded in engineered features while also discovering additional patterns through learned features, potentially capturing a more comprehensive representation of the data.

“Feature Engineering Operations

Feature Engineering Operations

Handling Missing Values



- Identify Missing Values
  - The first step in handling missing values is to identify which variables or features have missing data
  - This can be done by examining the dataset or using functions that detect missing values, such as isnull() or isna()
- Understand the Missingness
  - It is important to understand the patterns and reasons for missing values
  - Missingness can be categorized as Missing Completely at Random (MCAR), Missing at Random (MAR), or Missing Not at Random (MNAR).
  - This understanding helps in choosing the appropriate imputation strategy

Feature Engineering Operations

Handling Missing Values

1 **Missing Not at Random (MNAR)**

Missing is because of the true value itself

E.g.: Higher income person didn't disclose

[illegible] **Missing at Random (MAR)**

Not missing due to its value but another observed variable

E.g., Age not disclosed because of gender

[illegible] **Missing Completely at Random (MCAR)**

No pattern in missing

E.g.: Persons missed to provide

ID	Age	Gender	Income	Job
1	26	A	10	E
2	?	B		S
3	17	A	8	E
4	?	B	15	[illegible]
5	62	A		T

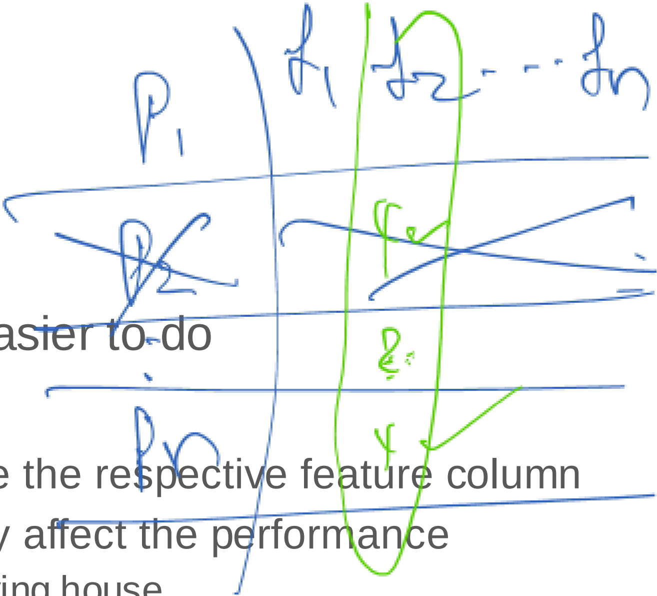
Handling Missing Values

Deletion, Imputation, Scaling, Discretization

Feature Engineering Operations

Handling Missing Values: Deletion

- Often, many tend to prefer deletion
- Not because of a better method, but because of easier to do
- Column Deletion
  - If a feature is missing in many samples, just remove the respective feature column
  - Might remove an important feature, which adversely affect the performance
    - E.g.; Marital feature might be highly correlated to buying house
- Row Deletion
  - If a row has a many missing features, remove the sample
  - Woks well for Missing values are completely at random (MCAR)
  - Might remove an important samples, which adversely affect the performance
    - E.g.; Don’t want to remove Gender B, which create bias



Feature Engineering Operations

Handling Missing Values: Imputation

- The process of filling in missing values in a dataset with estimated or imputed values
  - Missing values can occur in datasets due to various reasons such as data entry errors, sensor failures, or participant non-response
  - Imputation allows to retain the integrity of the dataset by avoiding the need to discard incomplete observations.
  - By filling in missing values, we can utilize the available data more effectively and minimize data loss
  - Missing values can introduce biases into statistical analyses or machine learning models if not appropriately handled
  - Helps mitigate bias by providing estimated values that preserve the overall distribution and relationships in the data
  - Enhances statistical power by increasing the sample size, which can lead to more precise and reliable statistical estimates

Feature Engineering Operations

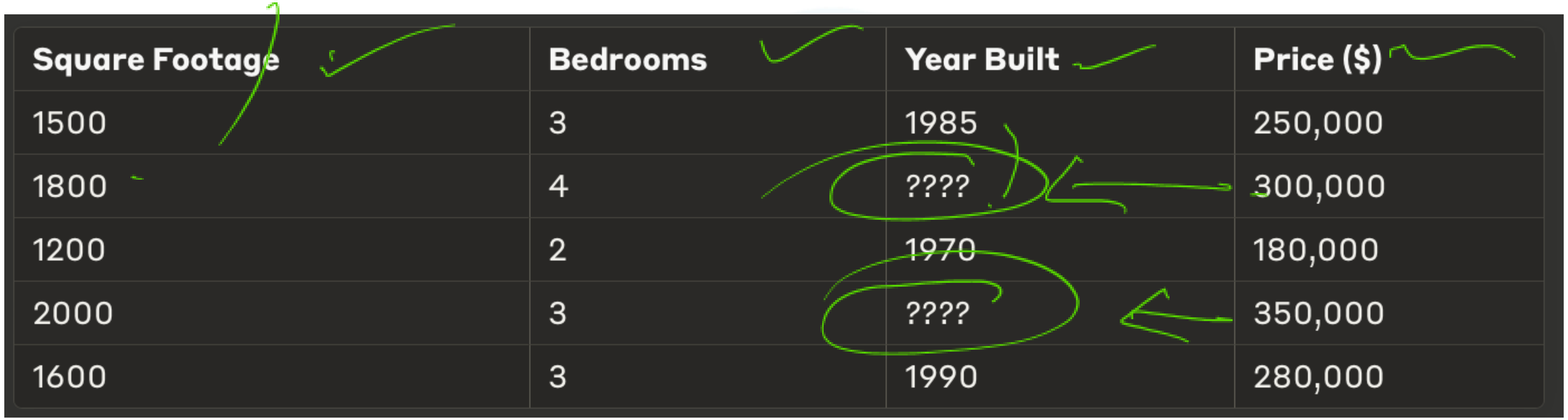
Handling Missing Values: Imputation

- Methods for Imputation
  - Various imputation methods exist to fill missing values, including mean, median, mode (the most common value), regression, hot deck (values from similar observed cases), and multiple (stochastic, accounts for the uncertainty associated)
  - The choice of method depends on the nature of the data, the missing data mechanism, and the goals of the analysis
  - Imputation methods make certain assumptions about the missing data mechanism and the relationship between variables
  - It is important to be aware of these assumptions and understand their implications while interpreting the imputed values

Feature Engineering Operations

Example: Imputation Methods for Missing Values

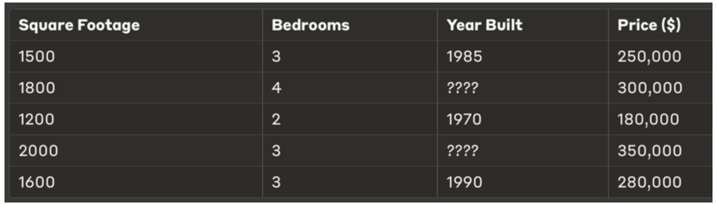
- Let's consider a dataset of house prices with the following features:
  - Square Footage, Number of Bedrooms, Year Built, and Price.
  - Some houses have missing values for the Year Built feature.



Square Footage	Bedrooms	Year Built	Price ($)
1500	3	1985	250,000
1800	4	????	300,000
1200	2	1970	180,000
2000	3	????	350,000
1600	3	1990	280,000

Feature Engineering Operations

Example: Imputation Methods for Missing Values



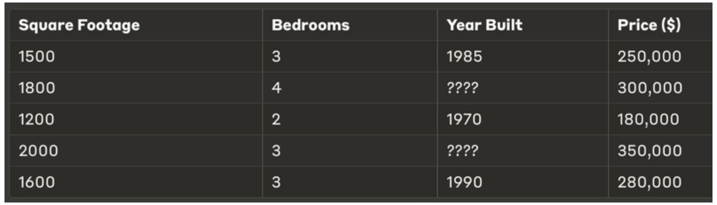
Square Footage	Bedrooms	Year Built	Price ($)
1500	3	1985	250,000
1800	4	????	300,000
1200	2	1970	180,000
2000	3	????	350,000
1600	3	1990	280,000

R

- Let's apply different imputation methods to fill in the missing Year Built values:
- **Mean Imputation:** Calculate the mean of the known Year Built values:
  - (1985 + 1970 + 1990) / 3 = 1981.67
  - Replace missing values with 1982 (rounded to nearest year)
- **Median Imputation:** Find the median of known Year Built values:
  - 1985
  - Replace missing values with 1985
- Mode Imputation: Find the most frequent Year Built value (in this small sample, there's no mode)
  - If we had more data and 1990 was the most common year, we'd use

Feature Engineering Operations

Example: Imputation Methods for Missing Values

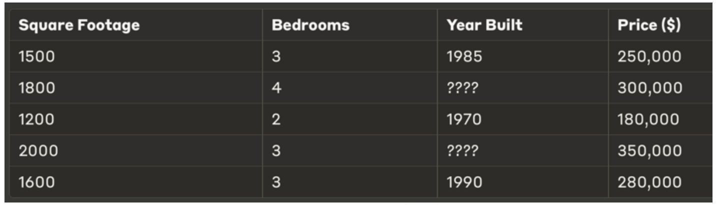


Square Footage	Bedrooms	Year Built	Price ($)
1500	3	1985	250,000
1800	4	????	300,000
1200	2	1970	180,000
2000	3	????	350,000
1600	3	1990	280,000

- **Regression Imputation:** Build a regression model to predict Year Built based on other features
    - For example, if larger, more expensive houses tend to be newer:
    - Predicted Year = 1950 + (0.01 * Square Footage) + (5 * Bedrooms) + (0.0001 * Price)
    - Use this model to predict missing values
- Hot Deck Imputation: For each house with a missing Year Built, find the most similar house with a known Year Built
    - For the 1800 sq ft house: might choose 1985 (from the 1500 sq ft house)
    - For the 2000 sq ft house: might choose 1990 (from the 1600 sq ft house)

Feature Engineering Operations

Example: Imputation Methods for Missing Values

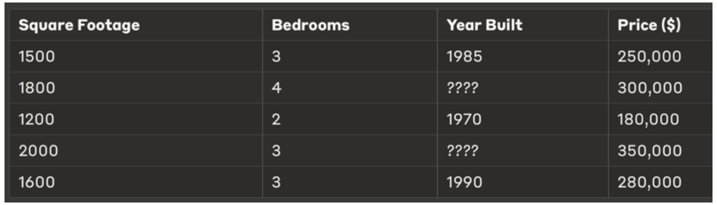


Square Footage	Bedrooms	Year Built	Price ($)
1500	3	1985	250,000
1800	4	????	300,000
1200	2	1970	180,000
2000	3	????	350,000
1600	3	1990	280,000

- **Multiple Imputation:** Generate multiple plausible imputed datasets, perhaps using a combination of the above methods
  - For example, for the 1800 sq ft house: Dataset 1: 1982 (mean) Dataset 2: 1985 (median) Dataset 3: 1988 (regression prediction)
  - Analyze each dataset separately and combine results, accounting for uncertainty

Feature Engineering Operations

Example: Imputation Methods for Missing Values



Square Footage	Bedrooms	Year Built	Price ($)
1500	3	1985	250,000
1800	4	????	300,000
1200	2	1970	180,000
2000	3	????	350,000
1600	3	1990	280,000

- Each method has its pros and cons:
  - Mean and median are simple but can distort relationships between variables
  - Regression can maintain relationships but may overfit
  - Hot deck preserves the distribution but may not work well with small datasets
  - Multiple imputation accounts for uncertainty but is more complex to implement
- The choice of method depends on the nature of your data, the amount of missing data, and the specific requirements of your analysis or model.

Feature Engineering Operations

Handling Missing Values: Imputation

- Sensitivity Analysis
  - Crucial when using imputed data
  - It involves examining the effects of different imputation methods on the results to assess the robustness and reliability of the conclusions drawn from the imputed dataset
- Incorporating Uncertainty
  - Imputation can introduce uncertainty due to the estimation of missing values
  - Techniques such as multiple imputation provide multiple plausible imputed datasets, allowing for the incorporation of uncertainty in subsequent analyses
- Documentation and Transparency
  - It is essential to document the imputation process thoroughly, including the method used, the rationale behind the choice of method, and any assumptions made
  - Promotes transparency, reproducibility, and the ability to validate the analysis

“Scaling

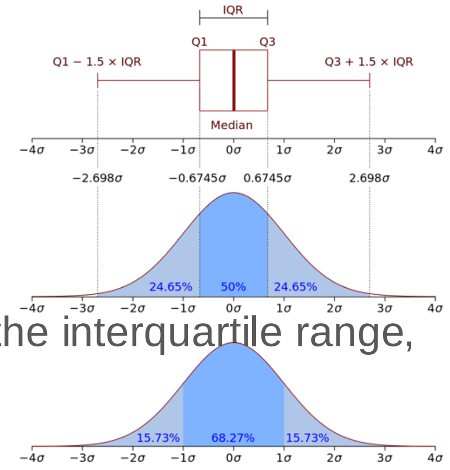
Feature Engineering Operations

Scaling

- Scaling refers to the process of transforming numerical variables to a standardized range or distribution, ensuring all features are on a similar scale
- Min-Max Scaling (normalization)
  - Transforms the data to a predefined range, typically between 0 and 1.
  - $\hat{x} = \dfrac{x - \min(x)}{\max(x) - \min(x)}$
- Crucial in machine learning because it helps prevent certain features from dominating others based solely on their magnitude & Ensures that all features contribute equally
- Standardization (Z-score Scaling)
  - Scales the data to have a mean of 0 and a unit variance
    - $\hat{x} = \dfrac{x - \bar{x}}{\sigma}$
    - Where $\bar{x}$ is the mean of x and σ is its standard deviation

Feature Engineering Operations

Scaling



- Robust Scaling
  - Useful when dealing with outliers
  - It scales the data by subtracting the median and dividing by the interquartile range, making it more resistant to the influence of extreme values
- Logarithmic Scaling
  - Employed when the data exhibits a wide range of values or a skewed distribution
  - Can help normalize the distribution and reduce the impact of extreme values
- Scaling is particularly important in neural networks
  - Input features that are not on a similar scale can lead to slow convergence or unstable training
  - Scaling the input data to a small range can improve the performance and stability of neural networks

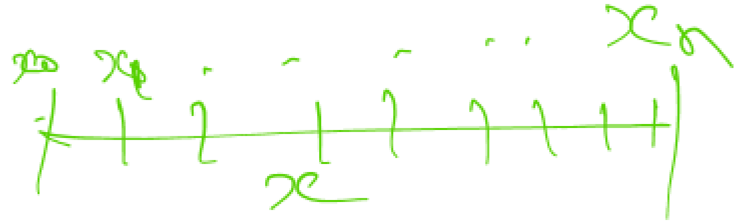
Feature Engineering Operations

Scaling

- Scaling for Distance-Based Algorithms
  - Distance-based algorithms, such as k-nearest neighbors (KNN) or clustering algorithms, are sensitive to the scale of the variables
  - Scaling the features ensures that the distance or similarity calculations are not skewed by variables with larger magnitudes
- Scaling for Regularization
  - Regularization techniques, such as ridge regression or Lasso, penalize large coefficients
  - Scaling the variables helps ensure that all features are penalized equally, preventing certain variables from dominating the regularization process
- When to Scale?
  - Generally recommended when using algorithms that rely on distance, magnitude, or regularization
  - However, it may not be necessary for all algorithms.
    - Decision trees or random forests, for example, are not influenced by variable scaling

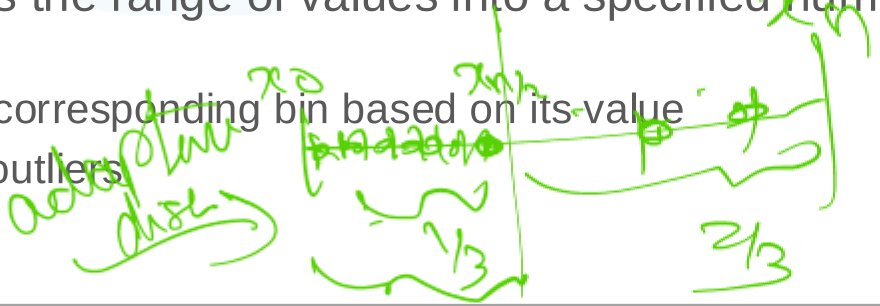
“Discretization

Feature Engineering Operations



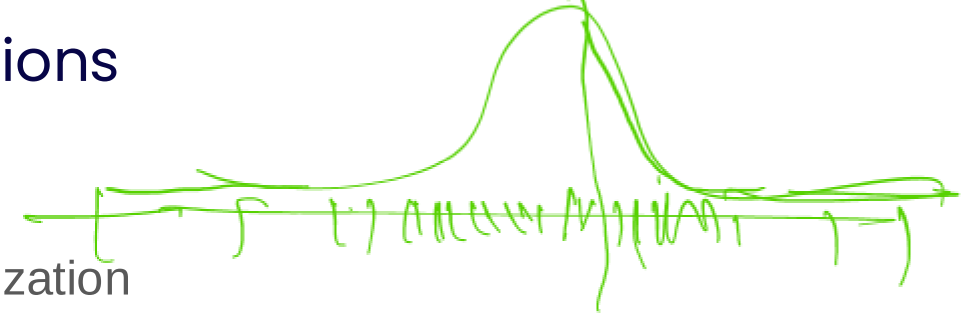
Discretization

- Process of transforming continuous numerical features into discrete bins or categories
- It helps simplify the representation of data and can be useful in feature selection
- Beneficial in feature selection as it can reduce the complexity of continuous variables, help identify non-linear relationships, and enable the use of certain algorithms that require discrete inputs
- Equal width discretization divides the range of values into a specified number of equal-width bins.
    - Assigns each data point to the corresponding bin based on its value
    - Simple but can be sensitive to outliers



Feature Engineering Operations

Discretization



- Equal Frequency (Quantile) Discretization
  - Equal frequency discretization divides the data into bins of approximately equal frequencies
  - Helps handle outliers and ensures a more balanced distribution across bins
- Custom Discretization
  - Allows for the creation of bins based on specific domain knowledge or requirements
  - Involves defining custom thresholds or ranges to partition the data into meaningful categories
  - Offers flexibility in capturing domain-specific patterns or characteristics
- Discretization for Feature Selection
  - Can aid feature selection by transforming continuous variables into discrete ones
  - Reduces the dimensionality of the data, simplifies the feature space, and facilitates the identification of relevant features

Feature Engineering Operations

Discretization

- Information Gain-Based Discretization
  - Measures the amount of information gained by dividing the data into different bins
  - Aims to find the optimal split points that maximize the information gain
- Discretization Impact on Feature Selection
  - Can have a significant impact on feature selection algorithms, especially when they rely on metrics such as information gain, chi-square, or mutual information
  - Can enhance the performance of these algorithms by capturing non-linear relationships between features and the target variable

Feature Engineering Operations

Discretization

- Discretization Challenges
  - Should be applied carefully, considering the nature of the data and the specific problem
  - Discretizing continuous variables can introduce information loss and reduce the precision of the data representation
  - Choosing the appropriate number of bins or thresholds is crucial to balance the granularity of the discretization and the preservation of valuable information
- Iterative Process
  - Discretization is often an iterative process in feature selection
  - Initially, continuous variables may be discretized to simplify the feature space
  - Subsequently, feature selection algorithms are applied on the discretized data to identify the most relevant features
  - This iterative approach helps refine the feature selection process and improve model performance

“Encoding Categorical Features

Feature Engineering Operations

Encoding Categorical Features

- People who haven't worked with data in production tend to assume that categories are static
- True for many categories
  - E.g: Income bracket and age bracket are unlikely to change but not the number of brands in an E-Commerce store
- Categorical Feature Encoding
  - Process of transforming categorical variables into numerical representations that machine learning models can understand
  - Categorical variables contain non-numerical values, such as labels or categories, which cannot be directly used by most machine learning algorithms
  - Encoding enables the inclusion of categorical features in the modeling process and helps capture the inherent information within them

Feature Engineering Operations

Encoding Categorical Features

- One-Hot Encoding
    - Popular technique that represents each category as a binary vector
    - Creates a binary column for each category, with a value of 1 indicating the presence of that category and 0 otherwise
    - Avoids imposing an artificial ordinal relationship among categories

Red (1 0 0)
Blue (0 1 0)
Black (0 0 1

Feature Engineering Operations

Encoding Categorical Features

- Label Encoding
  - Assigns a unique numerical label to each category
  - Maps each category to an integer value, allowing the model to interpret the encoded values as ordinal
  - However, label encoding may introduce unintended ordinality, which can impact the model's performance

Blue	0
Red	1
Black	2

Feature Engineering Operations

Encoding Categorical Features

- Ordinal Encoding
  - Assigns numerical values to categories based on their order or rank
  - Preserves the ordinal relationship among categories, which can be beneficial when the order carries meaningful information
  - However, ordinal encoding assumes a monotonic relationship between categories, which may not always be accurate

$$\begin{tikzcd}
\text{low} \arrow[r, dash] & 0 \arrow[d, bend left] \\
\text{Average} \arrow[r, dash] & 1 \arrow[d, bend left] \\
\text{High} \arrow[r, dash] & 2
\end{tikzcd}$$

Feature Engineering Operations

Encoding Categorical Features

- Count Encoding
  - Replaces each category with the count or frequency of its occurrences in the dataset
  - Represents the information about the distribution of categories within the dataset, which can be useful in certain scenarios

Feature Engineering Operations

Encoding Categorical Features

- Target Encoding
  - Uses the target variable's information to encode categorical features
  - Replaces each category with the average or probability of the target variable for that category
  - Captures the relationship between the categorical feature and the target, potentially enhancing predictive performance

0 Cat — D

0 Dog →

1 Lion

$\left\{ \begin{array}{l} \text{friendl} \\ \text{frien} \\ \text{£ \ unfri} \end{array} \right\}$

Feature Engineering Operations

Encoding Categorical Features

- Binary Encoding
  - Represents each category as a binary code
  - It assigns a unique binary pattern to each category, reducing the dimensionality compared to one-hot encoding while capturing information about category membership

Red $\rightarrow$ 001

Blue $\rightarrow$ 010

Blace $\rightarrow$ 011

Brwn $\rightarrow$ 111

$\rightarrow$

Cat 1	Cat 2	Cat 3
0	1	
1	0	
1	1	

Feature Engineering Operations

Encoding Categorical Features

- Feature Hashing
  - Also known as the hashing trick, maps categorical features to a fixed-length vector representation
  - Uses a hash function to convert the categories into numerical indices, which are then mapped to a predefined feature space
  - Feature hashing is efficient for large-scale datasets and can handle high-cardinality categorical variables

Feature Engineering Operations

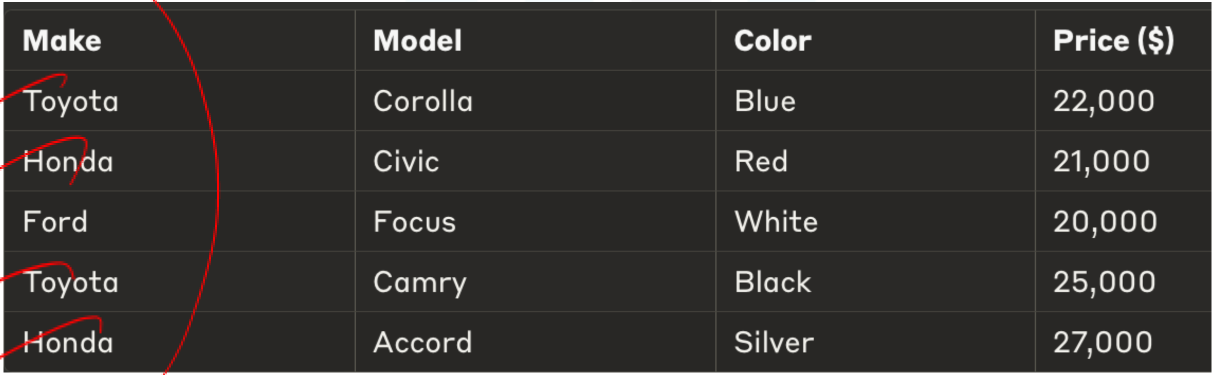
Encoding Categorical Features

- Encoding Considerations
  - When encoding categorical features, it is essential to consider the nature of the data, the number of categories, the potential relationships between categories, and the requirements of the machine learning algorithm
  - The choice of encoding technique can impact model performance, interpretability, and computational efficiency

Feature Engineering Operations

Example: Encoding Categorical Features

- Let's consider a dataset about car sales with the following features:
  - Make, Model, Color, and Price.
  - The Make, Model, and Color are categorical features that need to be encoded before they can be used in most machine learning models.
  - Here's a sample of our dataset:

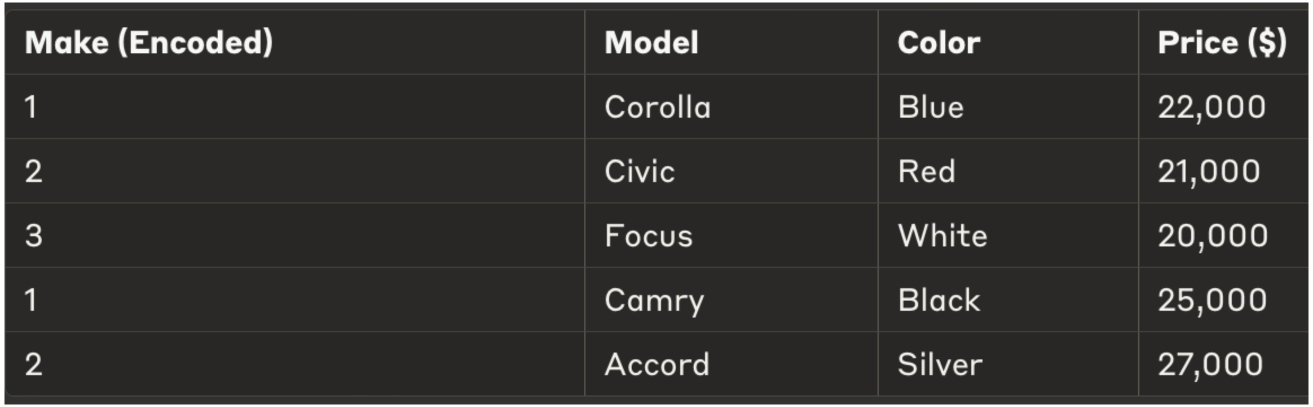


Make	Model	Color	Price ($)
Toyota	Corolla	Blue	22,000
Honda	Civic	Red	21,000
Ford	Focus	White	20,000
Toyota	Camry	Black	25,000
Honda	Accord	Silver	27,000

Feature Engineering Operations

Example: Encoding Categorical Features

- Let's apply different encoding methods to the 'Make' feature:
  - **Label Encoding:** Assign a unique integer to each category Toyota: 1, Honda: 2, Ford: 3



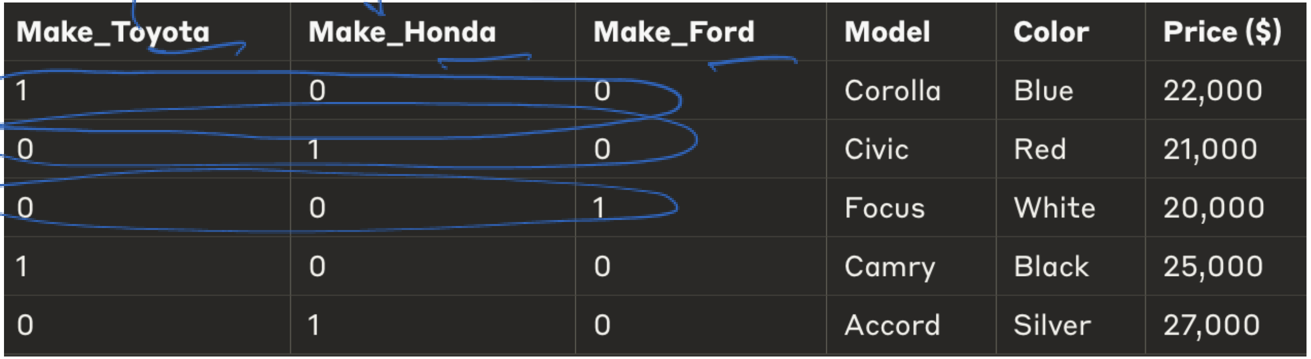
Make (Encoded)	Model	Color	Price ($)
1	Corolla	Blue	22,000
2	Civic	Red	21,000
3	Focus	White	20,000
1	Camry	Black	25,000
2	Accord	Silver	27,000

Feature Engineering Operations

Example: Encoding Categorical Features

- **One-Hot Encoding:**
  - Create a binary column for each category

3



Make_Toyota	Make_Honda	Make_Ford	Model	Color	Price ($)
1	0	0	Corolla	Blue	22,000
0	1	0	Civic	Red	21,000
0	0	1	Focus	White	20,000
1	0	0	Camry	Black	25,000
0	1	0	Accord	Silver	27,000

Feature Engineering Operations

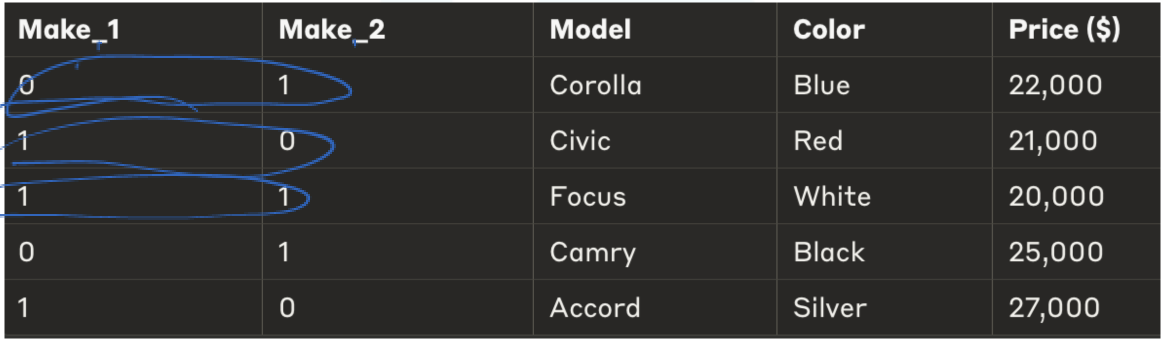
Example: Encoding Categorical Features

- **Binary Encoding:**
  - Represent each category as a binary code
  - Toyota: 01, Honda: 10, Ford: 11

T

H

F

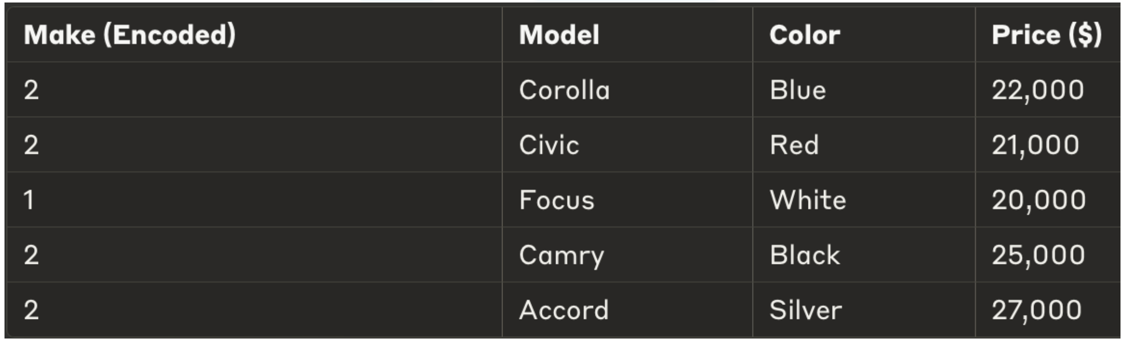


Make_1	Make_2	Model	Color	Price ($)
0	1	Corolla	Blue	22,000
1	0	Civic	Red	21,000
1	1	Focus	White	20,000
0	1	Camry	Black	25,000
1	0	Accord	Silver	27,000

Feature Engineering Operations

Example: Encoding Categorical Features

- **Count Encoding:**
  - Replace category with its frequency count
  - Toyota: 2, Honda: 2, Ford: 1



Make (Encoded)	Model	Color	Price ($)
2	Corolla	Blue	22,000
2	Civic	Red	21,000
1	Focus	White	20,000
2	Camry	Black	25,000
2	Accord	Silver	27,000

Feature Engineering Operations

Example: Encoding Categorical Features

- **Each method has its pros and cons:**
  - Label Encoding is simple but may imply an ordinal relationship that doesn't exist.
  - One-Hot Encoding avoids ordinal implications but can lead to high dimensionality with many categories.
  - Binary Encoding balances between information density and dimensionality.
  - Target Encoding can capture the relationship with the target variable but risks overfitting.
  - Count Encoding can provide information about category frequency but may not always be meaningful.

Feature Crossing

Feature Engineering Operations

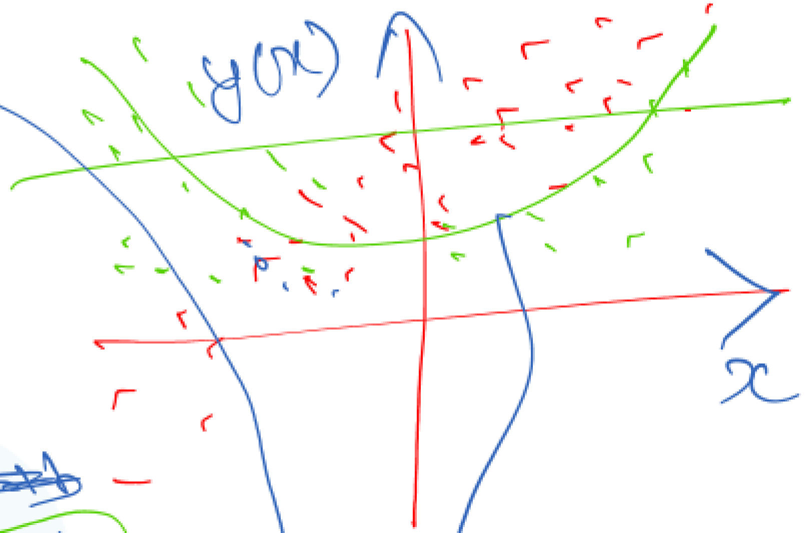
Feature Crossing

- Also known as feature interaction, is the process of creating new features by combining or crossing existing features
- Involves combining two or more features to capture potential non-linear relationships or interactions between them
- Importance of Feature Crossing
  - Can uncover hidden patterns and interactions that may not be captured by individual features alone
  - Allows ML models to learn complex relationships and improve predictive performance.
  - Many real-world relationships between features are non-linear and cannot be adequately represented by simple linear models
  - Helps capture these non-linear relationships by creating new features that incorporate interactions between different variables

Feature Engineering Operations

Feature Crossing

- Interaction Effects
  - Particularly valuable in detecting interaction effects, where the impact of one feature on the target variable varies depending on the value of another feature
  - By crossing features, we can explicitly capture and model these interaction effects, leading to more accurate predictions
- Polynomial Feature Crossing
  - Involves creating new features by multiplying or raising existing features to different powers
    - For example, crossing features "x" and "y" can create new features like "x^2," "xy," or "y^2.”
  - Effective in capturing curvature and complex relationships



x_i ~~y_i~~ x_i^2 label | target y_i

x_1^2

x_2^2

x_n^2

feature

$$\Rightarrow y(x) = \cancel{w x + b} = wx + b$$

$$\boxed{y(x_i) = w_1 x_i + w_2 x_i^2 + b}$$

a | b | $\downarrow$ new feature $c = ab$ | $y(a, b, c)$

Feature Engineering Operations

Feature Crossing

- Categorical Feature Crossing
  - Feature crossing is not limited to numerical features; it can also be applied to categorical features
  - By crossing categorical variables, we can capture interactions between different categories and uncover valuable information about their joint impact on the target variable
- Curse of Dimensionality
  - Feature crossing can significantly increase the dimensionality of the feature space, leading to the "curse of dimensionality."
  - As the number of crossed features grows, the data sparsity increases, and the model's performance may suffer
  - It is crucial to consider the trade-off between capturing complex interactions and managing dimensionality

Feature Engineering Operations

Feature Crossing

- Automated Feature Crossing
  - Automated feature engineering libraries or algorithms, can automatically generate and select relevant crossed features based on statistical significance or predictive power
  - These methods can save time and effort in manual feature engineering
- Interpretability Trade-off
  - Feature crossing can enhance predictive performance, but it may come at the cost of interpretability
  - As the number of crossed features increases, it becomes challenging to interpret the individual impact of each feature on the model's output
  - Balancing interpretability with predictive power is a crucial consideration

Feature Engineering Operations

Feature Crossing

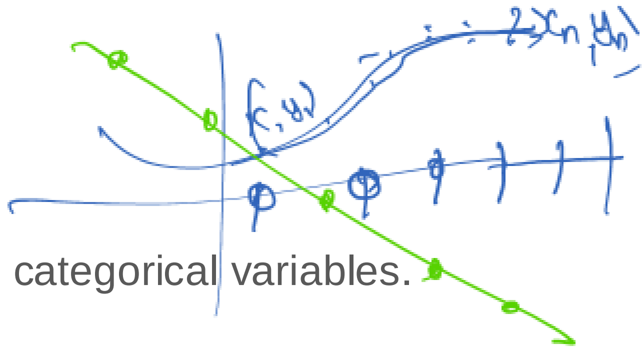
- Domain Knowledge and Experimentation
  - While automated feature crossing can be effective, domain knowledge and experimentation are valuable in identifying meaningful feature interactions
  - Understanding the underlying domain can guide the creation of relevant crossed features and improve the model's performance
  - Remember that feature crossing should be applied judiciously, considering the specific problem, dataset, and modeling goals
  - It is important to assess the impact of crossed features on model performance and interpretability and choose the most informative and relevant feature combinations

Feature Engineering Operations

Veracity Check

1. T/F: One-hot encoding is the only method for encoding categorical variables.
2. T/F: Min-max scaling always results in a distribution with a mean of 0 and standard deviation of 1.
3. T/F: Feature crossing can help capture non-linear relationships between features.
4. T/F: Discretization always improves model performance.

Feature Engineering Operations



Veracity Check

1. T/F: One-hot encoding is the only method for encoding categorical variables.

   Answer: False. Explanation: There are various encoding methods, including label encoding, target encoding, and binary encoding, among others.

2. T/F: Min-max scaling always results in a distribution with a mean of 0 and standard deviation of 1.

   Answer: False. Explanation: Min-max scaling results in values between 0 and 1. Standardization (z-score scaling) results in a mean of 0 and standard deviation of 1.

3. T/F: Feature crossing can help capture non-linear relationships between features.

   Answer: True. Explanation: By combining features, feature crossing can represent complex interactions and non-linear relationships that individual features cannot capture.

4. T/F: Discretization always improves model performance.

   Answer: False. Explanation: While discretization can be beneficial in some cases, it can also lead to information loss. Its effectiveness depends on the specific problem and data.

Feature Engineering Operations

Self-assessment

1. Q: What is the purpose of feature scaling, and name two common methods for it?

   A: Feature scaling aims to bring all features to a similar range. Two common methods are Min-Max scaling and Standardization (Z-score scaling). **Explanation**: Scaling ensures that features with larger magnitudes don't dominate the learning process. Min-Max scaling typically scales features to a range of [0,1], while Standardization scales features to have a mean of 0 and standard deviation of 1.

2. Q: How does one-hot encoding work, and when is it commonly used?

   A: One-hot encoding creates binary columns for each category in a categorical variable, with a 1 indicating the presence of that category and 0 otherwise. It's commonly used for nominal categorical variables. **Explanation**: This technique allows categorical variables to be represented numerically without imposing an ordinal relationship. It's particularly useful when the categories don't have a natural order.

3. Q: What is feature crossing, and how can it help in capturing non-linear relationships?

   A: Feature crossing involves creating new features by combining two or more existing features, often through multiplication. It can help capture non-linear relationships by representing interactions between features. **Explanation**: By creating these interaction terms, we allow the model to learn more complex patterns that depend on the combination of multiple features, which can be especially useful for linear models.

Feature Engineering Operations

Self-assessment

4. Q: Describe the process of discretization and provide an example of when it might be useful.

   A: Discretization involves converting continuous variables into categorical ones by binning values into discrete categories. It can be useful when there's a non-linear relationship between the feature and the target variable. **Explanation**: For example, age might have a non-linear relationship with purchasing behavior. Discretizing age into categories (e.g., child, teen, adult, senior) could capture this relationship more effectively than the raw, continuous age value.

5. Q: What is the difference between ordinal encoding and one-hot encoding?

   A: Ordinal encoding assigns a unique integer to each category based on its order, while one-hot encoding creates binary columns for each category. **Explanation**: Ordinal encoding is suitable for ordinal variables where there's a clear ordering of categories (e.g., education level), while one-hot encoding is better for nominal variables with no inherent order (e.g., color).

“Discrete and Continuous $\boxed{\text{Positional}}$ Embeddings

Discrete and Continuous Positional Embeddings

Definitions

- Positional embeddings are a fundamental component in sequence-based models, such as Transformers, for capturing the positional information of elements in a sequence

ULM

- Discrete Positional Embeddings
  - Assign a unique embedding vector to each position in the sequence
  - Each position is represented by a distinct vector, allowing the model to learn different positional relationships
  - Discrete positional embeddings are typically learned during the training process
- Continuous Positional Embeddings
  - Provide a continuous representation of position information
  - Instead of using discrete vectors, continuous positional embeddings use mathematical functions, such as sinusoidal functions, to generate smooth and continuous representations.
  - These embeddings are fixed and do not change during model training

Em − f ([illegible] (x), expert)

I love India — — —

I like cricket — —

India is Good at Cricket — —

Cricket is a Popular sports in India

words

(0.1, 0.3, 0.07, ...)

(1.2, 0.3, 0.1, 0.2)

Discrete and Continuous Positional Embeddings

Pros and Cons

- Explicitly encode the position of each element in the sequence, capturing precise positional relationships
- They can be learned from data, enabling the model to adapt to different sequence lengths and patterns
- Can increase the model's parameter count, especially for long sequences, as each position requires a unique embedding vector
- This can lead to higher memory

- Have a fixed size and do not increase the parameter count with sequence length
- Provide smooth representations, making it easier for the model to generalize positional relationships across different sequence lengths
- May not capture fine-grained positional information as precisely as discrete embeddings.
- Rely on periodic functions like sine and cosine to encode position, which can limit their ability to represent complex

Discrete and Continuous Positional Embeddings

Choosing between Discrete and Continuous

- The choice between discrete and continuous positional embeddings depends on the specific task, dataset, and model architecture
- Discrete embeddings are often preferred when precise positional information is crucial, while continuous embeddings are favored for their simplicity, generalizability, and computational efficiency
- Hybrid approaches can also be used, combining discrete and continuous positional embeddings
    - This approach combines the benefits of both methods by incorporating discrete embeddings for capturing fine-grained positional details and continuous embeddings for generalizing positional relationships

Discrete and Continuous Positional Embeddings

Impact

- Impact on Model Performance
  - The choice of positional embedding method can impact model performance, including accuracy, convergence speed, and generalization ability
  - It is important to experiment with different embedding techniques and evaluate their effects on specific tasks to choose the most suitable approach

Data Leakage

Data Leakage

What is it?

"features"

- Data leakage occurs when information from outside the training data leaks into the ~~model~~ during the modeling process

or training

- Leading to overly optimistic performance estimates and unreliable predictions

Data Leakage

Common Causes of Data Leakage

- Target Leakage
  - Information from the target variable is inadvertently incorporated into the features used for training a machine learning model
    - Target variable is present in the feature set due to the improper handling of data
- Train-Test Contamination
  - Occurs when information from the test set is inadvertently used during model training
  - This can happen when data preprocessing steps, such as feature scaling or imputation, are applied to the entire dataset before splitting into train and test sets.

80 $\boxed{20}$ %

60 %

Data Leakage

Effects of Data Leakage

- Overfitting
  - Data leakage can lead to overfitting, where the model becomes overly tuned to the training data, resulting in poor generalization to unseen data
  - The model may learn spurious patterns present in the leaked data, leading to inflated performance on the training set but poor performance on new data
- Inaccurate Performance Evaluation
  - Data leakage can artificially inflate the model's performance during evaluation, leading to misleading performance metrics
  - This can create a false sense of confidence in the model's effectiveness

Data Leakage

Preventing Data Leakage

80 : 20

- Proper Train-Test Split
  - Ensure that data splitting is performed before any preprocessing steps or feature engineering
  - This ensures that information from the test set does not contaminate the training process
- Feature Selection
  - Carefully select features based on prior knowledge and ensure that no information that would not be available during actual prediction is used

Data Leakage

Detecting and Resolving Data Leakage

- Visualize and Analyze Feature Relationships
  - Examine the relationships between features and the target variable to identify any suspicious patterns or correlations that may indicate data leakage
- Cross-Validation
  - Use proper cross-validation techniques, such as stratified k-fold or time-based splitting, to obtain reliable performance estimates and detect potential leakage
- Investigate Anomalies
  - Look for unexpected patterns or outliers in the data that may indicate leakage or information leakage

Data Leakage

Conclusion

- Data leakage can significantly impact the performance and reliability of machine learning models
- Understanding the causes, effects, and prevention strategies of data leakage is crucial for building accurate and trustworthy models
- Proper data handling, feature engineering, and model evaluation techniques are essential to mitigate the risk of data leakage and ensure ~~reliable~~ predictions.

Data Leakage

Veracity Check

1. T/F: Data leakage always results in overfitting.
2. T/F: Train-test contamination occurs when the test set influences the training process.
3. T/F: Data leakage can be completely eliminated by using cross-validation.
4. T/F: Target leakage happens when future information is inadvertently included in the training data.

Data Leakage

Veracity Check

1. T/F: Data leakage always results in overfitting.

   Answer: False. Explanation: While data leakage often leads to overfitting, it can also cause other issues like unreliable model evaluation or poor generalization.

2. T/F: Train-test contamination occurs when the test set influences the training process.

   Answer: True. Explanation: Train-test contamination happens when information from the test set leaks into the model training process, leading to overly optimistic performance estimates.

3. T/F: Data leakage can be completely eliminated by using cross-validation.

   Answer: False. Explanation: While cross-validation helps detect some forms of leakage, it doesn't guarantee elimination of all types of data leakage, especially target leakage.

4. T/F: Target leakage happens when future information is inadvertently included in the training data.

   Answer: True. Explanation: Target leakage occurs when the model is trained on information that wouldn't be available at the time of prediction in a real-world scenario.

Data Leakage

Self-assessment

1. Q: What is data leakage in the context of machine learning?

   A: Data leakage occurs when information that wouldn't be available at the time of prediction is used in the model training process. **Explanation**: This can lead to overly optimistic performance estimates and poor generalization, as the model is using information it shouldn't have access to in a real-world scenario.

2. Q: What is the difference between target leakage and train-test contamination?

   A: Target leakage occurs when predictive features include information about the target, while train-test contamination happens when information from the test set influences the training process. **Explanation**: Target leakage is about the features themselves containing future information, while train-test contamination is about the improper use of test data during model development.

3. Q: How can data leakage impact model evaluation?

   A: Data leakage can lead to overly optimistic performance estimates, giving a false sense of the model's predictive power. **Explanation**: When leakage occurs, the model has access to information it shouldn't, making it perform artificially well on the test set but poorly in real-world applications.

Data Leakage

Self-assessment

4. Q: What is a common cause of train-test contamination, and how can it be prevented?

   A: A common cause is performing data preprocessing (e.g., normalization, imputation) on the entire dataset before splitting into train and test sets. It can be prevented by splitting the data first, then applying preprocessing separately to each set. **Explanation**: By splitting first, we ensure that no information from the test set influences the preprocessing of the training set, maintaining the integrity of the model evaluation.

5. Q: How can domain knowledge help in identifying potential sources of data leakage?

   A: Domain experts can identify features that might inadvertently include future information or recognize subtle relationships between features that could lead to leakage. **Explanation**: For example, in a medical prediction task, a domain expert might recognize that a certain test result wouldn't be available at the time of prediction in practice, even if it's present in the historical data.

“Engineering Good Features

Engineering Good Features

Conclusion

1

- Importance of Feature Engineering
  - Feature engineering plays a vital role in ML as it involves creating new features or transforming existing ones to improve the performance and predictive power of models.

2

- Domain Knowledge
  - Leveraging domain knowledge is key to engineering good features
  - Understanding the problem domain helps identify relevant features that capture important aspects of the data and align with the problem's context

3

- Feature Extraction
  - Involves transforming raw data into a more suitable representation for the model
  - It can include techniques like extracting statistical measures, creating

Engineering Good Features

Conclusion

target

$$\underbrace{g(x, y, \cancel{z})}_{3\ \text{features}} \simeq w_1 x + w_2 y + \cancel{w_3 z}$$

Suppose $\{0.7, 0.25, 0.001\}$

low

- Feature Selection
  - Aims to identify the most informative and relevant features for the model
  - It helps reduce dimensionality, improve interpretability, and avoid overfitting
  - Techniques like statistical tests, correlation analysis, or model-based selection can be employed
- Feature Transformation
  - Involves scaling, normalizing, or applying mathematical operations to the features to make them more suitable for the model
  - It can help handle outliers, non-linear relationships, and improve the convergence of optimization algorithms

Engineering Good Features

Conclusion

- Encoding Categorical Features
  - Categorical features require encoding to convert them into numerical representations
  - Techniques like one-hot encoding, label encoding, or target encoding can be employed based on the nature of the data and the modeling requirements
- Handling Missing Data
  - Missing data can have a significant impact on model performance
  - Strategies like imputation, either by mean, median, or advanced methods, or considering missingness as a separate feature can be applied to handle missing values effectively

Engineering Good Features

Conclusion

- Time Series Features
  - Time series data often requires specific feature engineering techniques
  - Creating lag features, aggregating data over different time periods, or extracting seasonality patterns can help capture temporal dependencies and improve model performance.
- Feature Scaling
  - Feature scaling is important to ensure features are on similar scales
  - Techniques like standardization or normalization can be used to bring features to a similar range, preventing some features from dominating others

Engineering Good Features

Conclusion

- Iterative Process
  - Feature engineering is often an iterative process
  - It involves experimenting with different feature engineering techniques, evaluating their impact on model performance, and refining the feature set based on feedback from the model
  - Remember that good feature engineering requires a combination of creativity, analytical thinking, and domain expertise
  - It is crucial to understand the data, problem context, and the requirements of the machine learning algorithm to engineer effective features that capture the underlying patterns and improve model performance

Engineering Good Features

Veracity Check

1. T/F: Data leakage always results in overfitting.

   Answer: False. Explanation: While data leakage often leads to overfitting, it can also cause other issues like unreliable model evaluation or poor generalization.

2. T/F: Train-test contamination occurs when the test set influences the training process.

   Answer: True. Explanation: Train-test contamination happens when information from the test set leaks into the model training process, leading to overly optimistic performance estimates.

3. T/F: Data leakage can be completely eliminated by using cross-validation.

   Answer: False. Explanation: While cross-validation helps detect some forms of leakage, it doesn't guarantee elimination of all types of data leakage, especially target leakage.

4. T/F: Target leakage happens when future information is inadvertently included in the training data.

   Answer: True. Explanation: Target leakage occurs when the model is trained on information that wouldn't be available at the time of prediction in a real-world scenario.

Engineering Good Features

Self-assessment

1. Q: Why is domain knowledge important in feature engineering?

   A: Domain knowledge helps in creating relevant features that capture important aspects of the problem, often leading to more effective and interpretable models. **Explanation**: Experts in the field can identify key factors and relationships that might not be immediately apparent from the raw data, guiding the creation of informative features.

2. Q: What is feature selection, and why is it important in the feature engineering process?

   A: Feature selection is the process of choosing a subset of relevant features for use in model construction. It's important for reducing overfitting, improving model performance, and enhancing interpretability. **Explanation**: By selecting only the most relevant features, we can reduce noise in the data, speed up training, and create more parsimonious models.

3. Q: How does feature engineering differ for time series data compared to static data?

   A: Time series data often requires specific techniques like creating lag features, extracting trends and seasonality, or using rolling statistics, which are not typically used with static data. **Explanation**: These techniques help capture the temporal dependencies and patterns unique to time series data, which are crucial for accurate predictions in time-dependent problems.

Engineering Good Features

Self-assessment

1. Q: What is the purpose of feature scaling, and name two common methods for it?

   A: Feature scaling aims to bring all features to a similar range, preventing features with larger magnitudes from dominating the learning process. Two common methods are Min-Max scaling and Standardization (Z-score scaling). **Explanation**: Min-Max scaling typically scales features to a range of [0,1], while Standardization scales features to have a mean of 0 and standard deviation of 1. This can be crucial for many machine learning algorithms.

2. Q: Why is feature engineering often described as an iterative process?

   A: Feature engineering is iterative because it often involves creating features, testing their effectiveness, and refining them based on model performance and insights gained. **Explanation**: As we learn more about the problem and how different features impact model performance, we can continually improve our feature set, often going through multiple cycles of creation, evaluation, and refinement.

Feature Engineering

Conclusion

- Learned Features Vs. Engineered Features
- Common Feature Handling Operations
- Data Leakage
- Engineering Good Features
- Summary