Lecture 7

–

Feature Engineering

Mönchsgrasmücke / Eurasian blackcap

(*Sylvia atricapilla*)

- It has mainly olive-grey upperparts and pale grey underparts. Both sexes have a neat coloured cap to the head, black in the male and reddish-brown in the female.
- The male's typical song is a rich musical warbling, often ending in a loud high-pitched crescendo. Its rich and varied song has led to it being described as the "mock nightingale" and it has featured in literature, films and music.
- The motives of the song are learned by the young male birds and are thus handed down. They sing for the first time in autumn, this autumn song has shorter stanzas. Due to the large distribution area of the blackcap, several dialects have developed.
- The nest is a neat cup, built low in brambles or scrub, and the clutch is typically 4–6 mainly buff eggs, which hatch in about 11 days. The chicks fledge in 11–12 days, but are cared for by both adults for some time after leaving the nest.

Sources:
* Photo by Premek Hajekon pixabay
* https://en.wikipedia.org/wiki/Eurasian_blackcap

Feature Engineering

1. Feature Engineering, Domain Knowledge and ML Algorithms
2. Scaling, Normalization, Standardization
3. Identifying important Features
4. Encoding Categorical Features
5. Creating new Features

What is Feature Engineering?

- Feature engineering (feature extraction, feature discovery) is the process of using domain knowledge to extract features from raw data in order to improve the performance of machine learning models.

- Feature engineering options
  - Select Features
  - Modify existing Features
  - Create new Features

- Goals of Feature Engineering
  - Improve a model's predictive performance
  - Reduce computational effort
  - Reduce data needs
  - Improve interpretability of the results

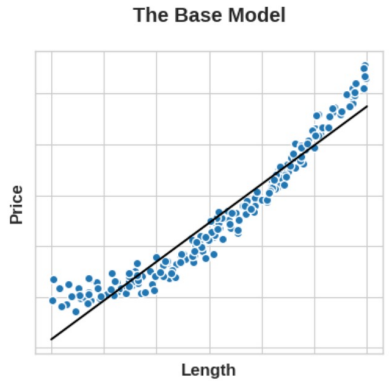
Simple Feature Engineering Examples based on Domain Knowledge

- Example 1
  - Target is Ice Cream Sales
  - Existing Features are Temperature, Humidity, Wind Speed
  - Domain Knowledge: People buy more Ice Cream, when it is “hotter”. Actually, when they feel hotter!
  - ➔ Temperature itself has a weaker relationship with target than expected
  - ➔ New Features should be based on perceived temperature, e.g. *heat index* or *wind chill factor*

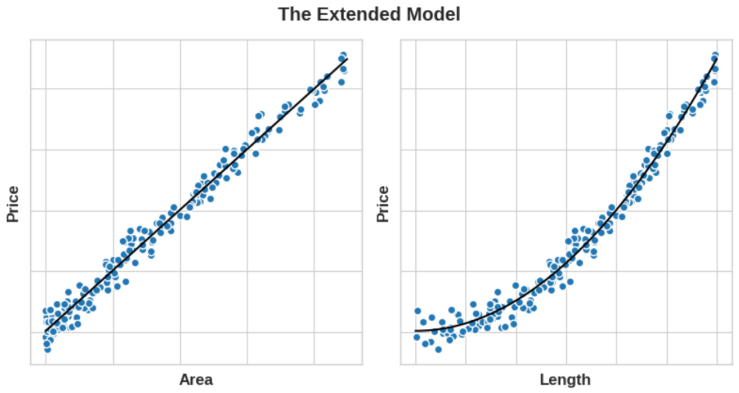
- Example 2
  - Target is quality produced cement, i.e., of the product of a chemical process
  - Existing features are amount of cement produced, amount of each raw ingredient, temperature, start timestamp of production, end timestamp of production
  - Domain Knowledge: absolute amounts of ingredient has a weak relationship with target and the start/end time also have a weak relationship with the target
  - ➔ New Features for ratios of ingredients will have a stronger relationship, as well as the duration of „baking“ the cement instead of start/end times

Simple Feature Engineering Examples based on ML Algorithm

- Example:
  - Target: predict the **Price** of square plots of land
  - Features: **Length** of one side
  - Algorithm: Linear Regression
  - Result: Poor! The relationship is not linear.



  - Feature Engineering: Replace Length by new feature **Area** (=Length*Length)
  - Result: Great! The relationship is now linear!



Source: https://www.kaggle.com/code/ryanholbrook/what-is-feature-engineering

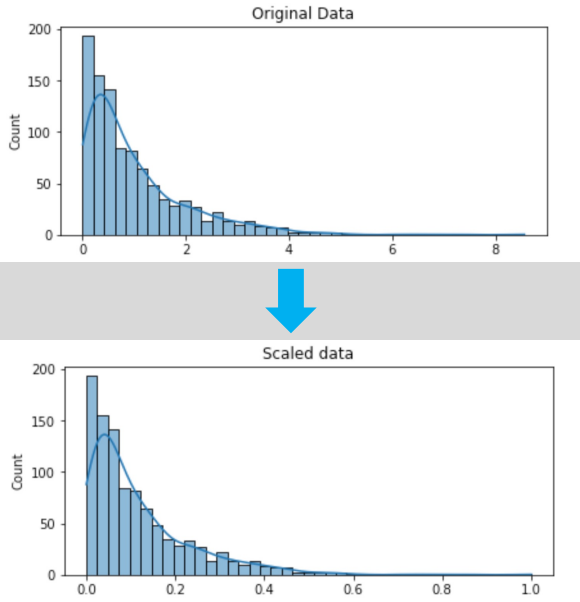
Feature Engineering

1. Feature Engineering, Domain Knowledge and ML Algorithms
2. Scaling, Normalization, Standardization
3. Identifying important Features
4. Encoding Categorical Features
5. Creating new Features

Scaling vs. Normalization

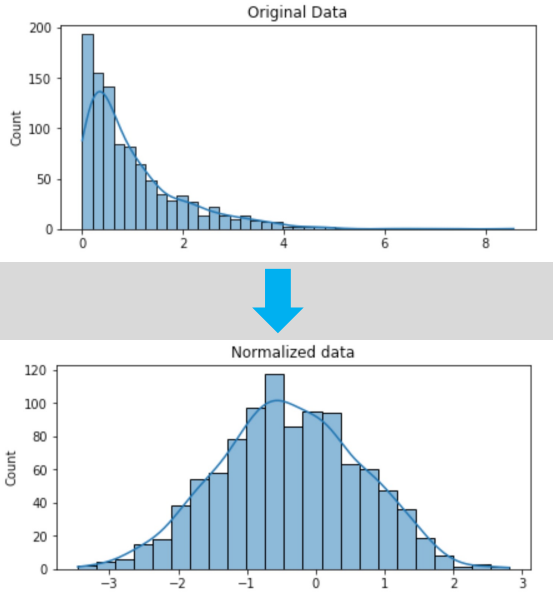
Scaling

Change the range of values
of a feature (affine transformations)



Normalization

Change the shape of the distribution
of a feature



Source: https://www.kaggle.com/code/alexisbcook/scaling-and-normalization

Scaling

- Transforming features to specific scale using an affine transformation
  - Such that a change of "1 unit" in any numeric feature is given the same importance
  - See **`sklearn.preprocessing`** for a large selection of scalers

- Popular scalers
  - Min-Max-Scaler: scale each feature to a given range.
    `MinMaxScaler``(feature_range=(0, 1), *, copy=True, clip=False)`

  - RobustScaler: Scale features using statistics that are robust to outliers.
    `RobustScaler``(*, with_centering=True, with_scaling=True, quantile_range=(25.0, 75.0), copy=True, unit_variance=False)`

- When to use scaling?
  - When using methods based on the distance between data points
  - Examples: SVM, kNN, k-means-Clustering, PCA, linear regression with L1 or L2 regularization, …

Normalization

- Change features so that they can be described by a normal (Gaussian) distribution
  - See `sklearn.preprocessing` for a large selection of normalizers
- Popular normalization algorithms
  - Yeo-Johnson transformation or Box-Cox transformation
    `PowerTransformer(method='yeo-johnson', *, standardize=True, copy=True)`
  - Quantile Transformer
    `QuantileTransformer(*, n_quantiles=1000, output_distribution='normal', …)`
- When to use normalization?
  - When using methods that assumes the data is normally distributed
  - Examples: Naive Bayes Classification, EM-Clustering, …

Standardization

- Transforming features to mean 0 and variance 1
  - Depending on the transformation: a special case of Scaling or Normalization
- Code available in **`sklearn.preprocessing`**
  - Standard-Scaler / “z-score-Normalization”
    `StandardScaler(*, copy=True, with_mean=True, with_std=True)`
- When to use standardization?
  - Often helpful, when using methods based on the distance between data points
  - Examples: SVM, kNN, k-means-Clustering, PCA, linear regression with L1 or L2 regularization, …

Summary: Scaling, Normalization, Standardization

- Critical for many ML models
- Examples
  - Linear Regression without L1/L2 regularization and data without Outliers: no transformation needed
  - Linear Regression with L1/L2 regularization and data without Outliers: Scaling a must
  - Linear Regression with L1/L2 regularization and data out Outliers: Normalization recommended
  - Tree-Based models usually do not benefit from Scaling/Normalization/Standarization
- Data Scientist has to understand the data and the model to decide what to use!
- Usually the last step in feature engineering
  - After creating new features, encoding categorical features, etc.
  - Often strong interaction with Outliers – usually, remove Outliers before Scaling/Standardization
- Nice comparison of the most popular Scalers, Normalizers, Standardizers: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

Feature Engineering

1. Feature Engineering, Domain Knowledge and ML Algorithms
2. Scaling, Normalization, Standardization
3. Identifying important Features
4. Encoding Categorical Features
5. Creating new Features

Identifying Important Features

- Many measures to quantify the importance of features have been proposed
  - Pearson Correlation Coefficient
  - Mutual Information
  - Chi-squared
  - …
- Many methods to chose the most important features have been proposed
  - Select k-best features
  - Sequential Feature Selection (forward or backward)
  - Recursive Elimination
- When to use?
  - May be helpful for EDA when little domain knowledge is available
  - May be helpful if a large number of features are available
  - Very frequently: domain knowledge is much more important
  - Code available in `sklearn.feature_selection`

Pearson Correlation Coefficient / Matrix

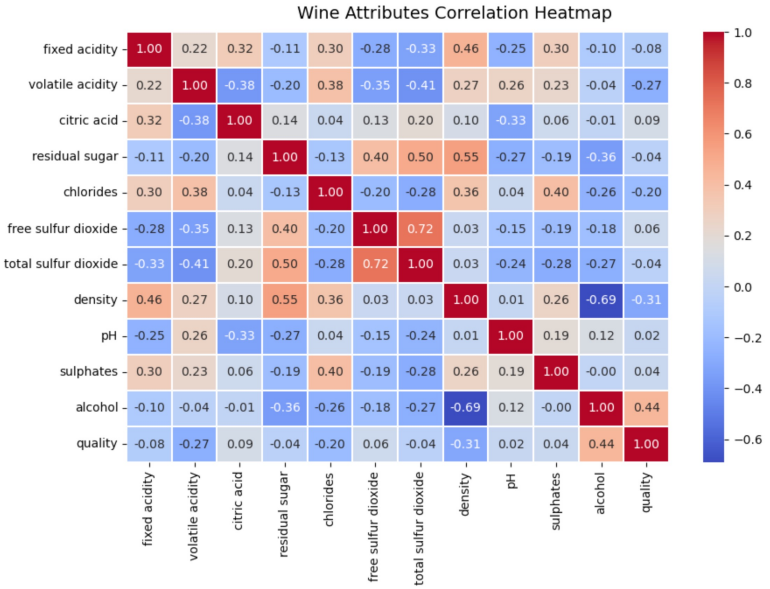
- Many “Correlation Coefficients” defined / in use
- Most commonly used: Pearson product-moment correlation coefficient (r or R)
  - measure of the strength and direction of the *linear* relationship between two variables
- Code available in pandas

  `DataFrame.corr(method='pearson', …)`
  Compute pairwise correlation of columns.

  Frequent usage in conjunction with seaborn/matplotlib:

```
sns.heatmap(df.corr(), annot=True)
plt.show()
```

- When/How to use?
  - Strong correlation between a feature and the target
    → valuable feature
  - Strong correlation between two features
    → may decrease quality of model (e.g. of linear regression)



Feature Engineering

1. Feature Engineering, Domain Knowledge and ML Algorithms
2. Scaling, Normalization, Standardization
3. Identifying important Features
4. Encoding Categorical Features
5. Creating new Features

Motivation and Overview

- Motivation
  - Categorical features take on a limit number of values
  - Examples
    - Maritial status: ”Single”, “Married”, “Divorced”
    - Many surveys offer answers of “Never”, “Rarely”, “Usually”, or ”Often”.
  - Many ML algorithms can only handle numeric input features
  - ➔ Need to transform categorical features into numeric features

- Approaches
  - Drop categorical features
  - Ordinal Encoding
  - One-Hot Encoding
  - Target-Encoding

Drop categorical features and Ordinal Encoding

- Drop categorical features
  - Only a good option if the dropped feature does not contain any useful information
- Ordinal Encoding
  - Assigns a different integer to each unique value of the feature
    - E.g., ”Single” = 1, “Married” = 2, “Divorced” = 3
      “Never” = 1, “Rarely” = 2, “Usually”=3, ”Often”=4.
  - Creates an ordering of the categories and a distance between them
    - E.g., Never < Rarely; Often is 4 * Never; Single < Divorced
  - Code

```
sklearn.preprocessing.OrdinalEncoder(*, categories='auto', …)
```

  - When to use
    - Can work well for ordinal features and tree-based methods (XGBoost etc.)
    - Sidenote: there are two version in `sklearn`: `OrdinalEncoder` is for features (2-dim), `LabelEncoder` for the target (classes, 1-dim) – basic functionality is the same.

One-Hot Encoding

- Create one new feature for each unique value of the feature to be encoded and set exactly one of them to ”1”, all others to “0”
  - E.g., to encode the maritial status:
  - Frequently drop one of/the first of the new columns (”base line”) to decrease correlation among the new features

Maritial Status
Single
Married
Single
Divorced
Married
…

Single	Married	Divorced
1	0	0
0	1	0
1	0	0
0	0	1
0	1	0
…	…	…

- does **not** assume an ordering of the categories
- Code
  - `sklearn.preprocessing.OneHotEncoder(*, categories='auto', …)`
  - `pandas.get_dummies()`
- When to use?
  - Low-Cardinality Categorical attributes (works well up to around 15 values)

Target-Encoding - Basics

- Basic Idea: Use the target to create the encoding (→ Supervised Feature Engineering)
- Most frequent approach: Encoding each feature value as the mean of the target within this group
  - Therefore, often called “mean encoding”
  - Works for numeric target (i.e., regression) and binary target (i.e., binary classification)
  - For multi-class target (multi-class classification), need to one-hot-encode the target class and apply binary-target-encoding multiple times
- Issues with this simple idea
  - Unknown feature values: need to be imputed somehow
  - Rare feature values: high risk of overfitting
  - Overfitting: training set means may not be close to real means

➔ Regularization needed!

Target-Encoding – Regularization with Prior Smoothing

- Idea: Smooth/Regularize the encoded mean value by moving it towards the mean of the whole population (the Prior).
  - Many mathematical formulations possible

- M-Estimate Encoder
  - Popular choice (as it has only one parameter m)

    $encoding = means\_in\_group * weight + total\_population\_means * (1 - weight)$ with

    $weight = n / (n + m)$ ($weight$ is called the m-Estimate)

    where n is the number of times the group occurs and m the smoothing factor (parameter)

- Code available in `scikit-learn-contrib`

```
category_encoders.m_estimate.MEstimateEncoder(…, m=1.0)
category_encoders.wrapper.PolynomialWrapper(feature_encoder: BaseEncoder)
```

- When to use?
  - High-Cardinality Categorical features
  - Domain-Motivated features

Exercise

Exercise 1

The Ames Housing Data Set – Part II

Identifying Important Features
Encoding Categorical Features
Scaling

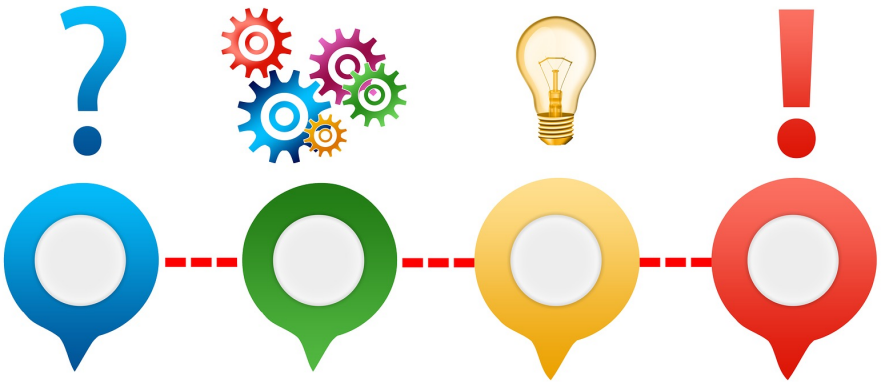


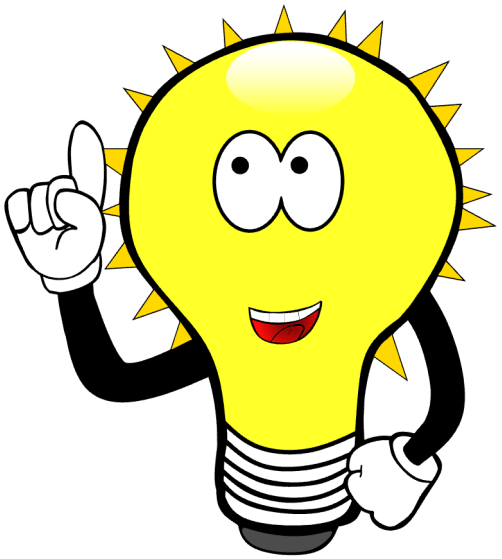
Photo by Gerd Altmann on Pixabay

Feature Engineering

1. Feature Engineering, Domain Knowledge and ML Algorithms
2. Scaling, Normalization, Standardization
3. Identifying important Features
4. Encoding Categorical Features
5. Creating new Features

Overview

- New features can improve the performance of the model significantly
  - Many method to create to new features have been proposed
- Here, we'll look at:
  - A. Transformations
    - A.a. Mathematical Transformations
    - A.b. Counts
    - A.c. Splitting Features
    - A.d. Combining Features
    - A.e. Group-by/Aggregation Transformations
  - B. Clustering
  - C. PCA
- How to come up with new features?
  - Understand the business domain and the existing features / raw data
  - Use data visualization
  - Creative Process: Brainstorming, Brainwriting, Teamdiscussion, ...

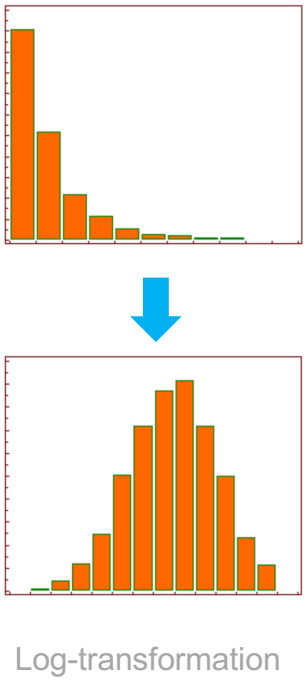


A.a. Mathematical Transformations

- Observation: Relationships among numerical features often expressed through complex mathematical formulas (identify such formulas during domain research)

- Frequent mathematical transformations
  - Ratios, if ratio is more important than absolute numbers
    - Example: ratio cement production ingredients instead of absolute values
  - Log-Transformation, if data is highly skewed (approx. follow a log-normal distribution)
    - Example: income vs. age may be exponential. Apply log-transform to income to get linear relationship.
    - Note: Data has to be positive. In case ”0” is included, use log1p()
    - Code: `numpy.log()` or `numpy.log1p()` `(=log(x+1))`
  - Duration, if exact time is less relevant
    - Examples: age at time of purchase instead of birthday; length of a process instead of start/stop time

- Why?
  - Many models have a hard time to learn / cannot learn complex formulas. Examples:
    - Linear regression can learn sum/difference, but nothing more complex
    - Ratios are hard for pretty much all models
  - Code: Can be easily implemented in Pandas



Log-transformation

Source: https://www.medcalc.org/manual/log-transformation.php

A.b. Counts

- Given: (Boolean) Features describing the presence or absence of something.
- Idea: Aggregating such features by creating a count.
- Examples:
  - Set of risk factors for a disease (is_smoking, is_overweight, …)
  - Cement production: given are the amounts of the components, some have value “0” → number of used components as a new feature
- Why?
  - Certain ML models (including tree-based model) can not learn counts
- Code available in pandas
  - `Dataframe.count(axis=1)` or `Dataframe.sum(axis=1)` für 1/0 or True/False columns
  - `Dataframe.gt(0).sum(axis=1)` for value-columns

A.c. Splitting Features

- Idea: Break down complex features (strings) into simpler, more useful parts.
- Frequent complex strings
  - Dates, times and timestamps, e.g. ISO 8601 “2023-02-14T08:22:06+00:00”
  - Phone numbers, e.g. “+49 (0) 8031 805 – 0” or “(08031) 805-0”
  - Street addresses, e.g. “Hochschulstraße 1, 83024 Rosenheim”
  - URLs, e.g. “https://www.th-rosenheim.de/”
  - ”Speaking” ID numbers or product codes, e.g. “TEE01-RD”: first three letters state product category, last two letters state colour.
- Why?
  - Parts can provide additional information, e.g. Phone-Area-Code and postal code provide location
- Code available in pandas
  - `Series.str` lets you apply string methods (e.g. `split`) directly to columns, e.g.
    ```
    df[[”my_date”, ”my_time”]] = (df[”my_timestamp"].str.split(”T”, 2, expand=True))
    ```
  - `DataFrame.map` is a general method to map values to new values using a dictionary

A.d. Combining Features

- Idea: Join simple features into a composed feature
- Why?
  - If we believe there is some interaction in the combination
- Example
  - Clothing retailer may believe that black clothes sell better in large and colorful clothes sell better in smaller sizes (as black makes you appear smaller and thinner). So she might join size and color into one feature.
- Code available in pandas

```
sales["color_and_size"] = sales["color"] + "_" + sales["size"]
```

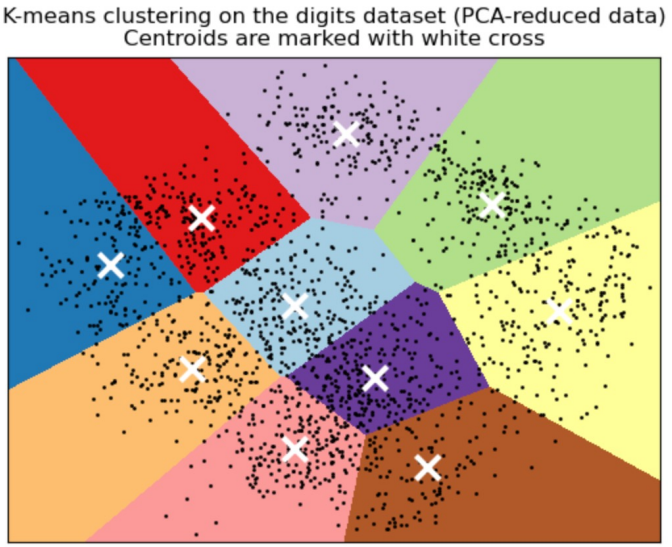
A.e. Group-by/Aggregation Transformations

- Idea: Create aggregations over groups in the data
  - Like SQL: select … from … group by…
- Why?
  - If we believe there is an interaction with subgroups
- Example
  - In a customer analysis, we believe the target has a relationship with the average income in an area (identified by postal code PLZ)
- Code available in pandas

```
customer["AvgIncome"] = (customer.groupby("PLZ")["Income"].transform("mean"))
```

B. Clustering

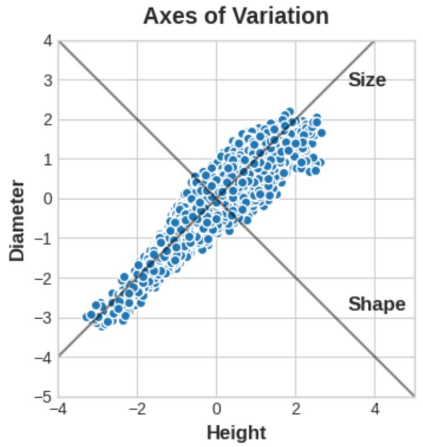
- Clustering
  - Assigning data points to groups based upon how similar the points are to each other (Unsupervised Learning)
  - Most popular Clustering Algorithms:
    k-means Clustering, EM-Clustering
  - Applied to a single feature:
    clustering = binning or discretization
  - Applied to multiple features:
    multi-dimensional binning (also called vector quantization)



- Idea
  - Clusters (cluster labels) break up complicated relationships across features into simpler chunks.
  - “divide and conquer” strategy: Model can learn the simpler chunks one-by-one instead having to learn the complicated whole all at once.

Source: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html#sphx-glr-auto-examples-cluster-plot-kmeans-digits-py

C. PCA



- PCA: partitioning of the variation in the data (Unsupervised Learning)
  - In a way similar to Clustering, which partitions according to similarity/distance
- PCA for Feature Engineering
  - Dimensionality reduction: Drop PCA components with near-zero variance.
  - Anomaly detection: Look for unusual variation in the low-variance components.
  - Noise reduction: PCA can sometimes collect the (informative) signal into a smaller number of features while leaving the noise alone.
  - Decorrelation: transform correlated features into uncorrelated components.
- PCA best practises
  - Note: only works with numeric features
  - PCA is sensitive to scale → usually standardize the data
  - Outliers can strongly influence the result → usually remove outliers beforehand

Source: https://www.kaggle.com/code/ryanholbrook/principal-component-analysis

Exercise

Exercise 2

The Ames Housing Data Set – Part III

Creating New Features

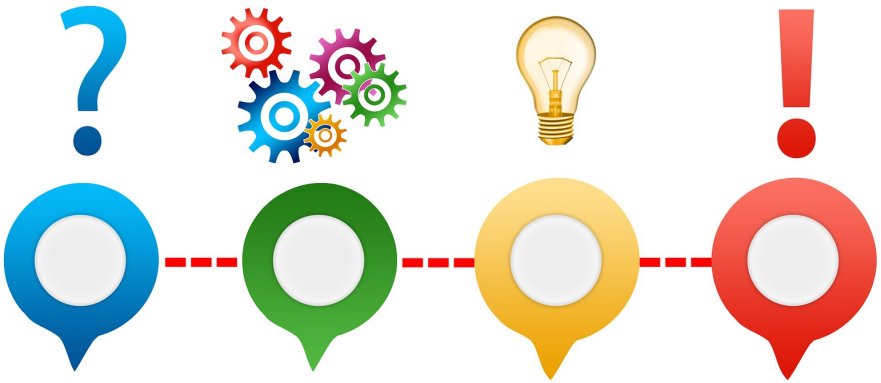
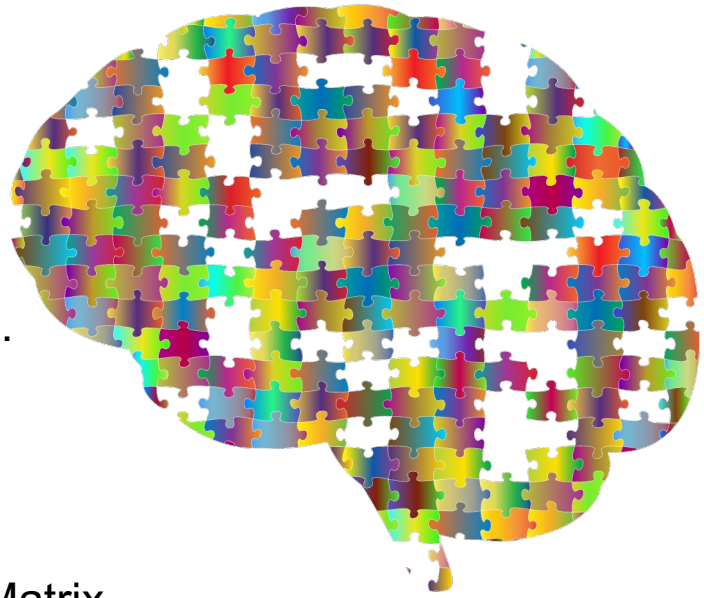


Photo by Gerd Altmann on Pixabay

Key Takeaways



- Feature engineering is the process of using domain knowledge to extract features from raw data in order to improve the performance of machine learning models.
- Scaling ←→ Normalization ←→ Standardization
  (these are Special-Purpose Mathematical Transformations)
- Identifying important Features: Pearson Correlation Coefficient Matrix
- Encoding Categorical Features: Ordinal Encoding, One-Hot Encoding, Target-Encoding
- Creating new Features
  - Transformations:
    Mathematical incl. ratios, Counts, Splitting/Combining Features, Group-by/Aggregation
  - Clustering
  - PCA

Image by Gordon Johnson on pixabay