# Data Science Africa

Lacuna Workshop
on
ML Data Preparation

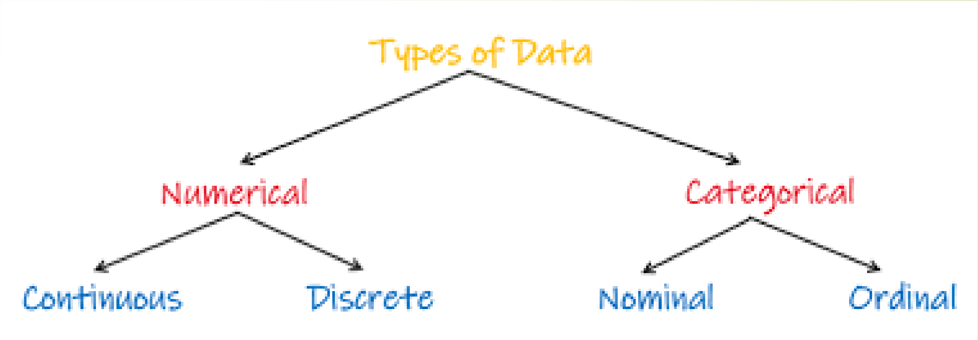
Isah Charles Saidu (Ph.D)

# Agenda

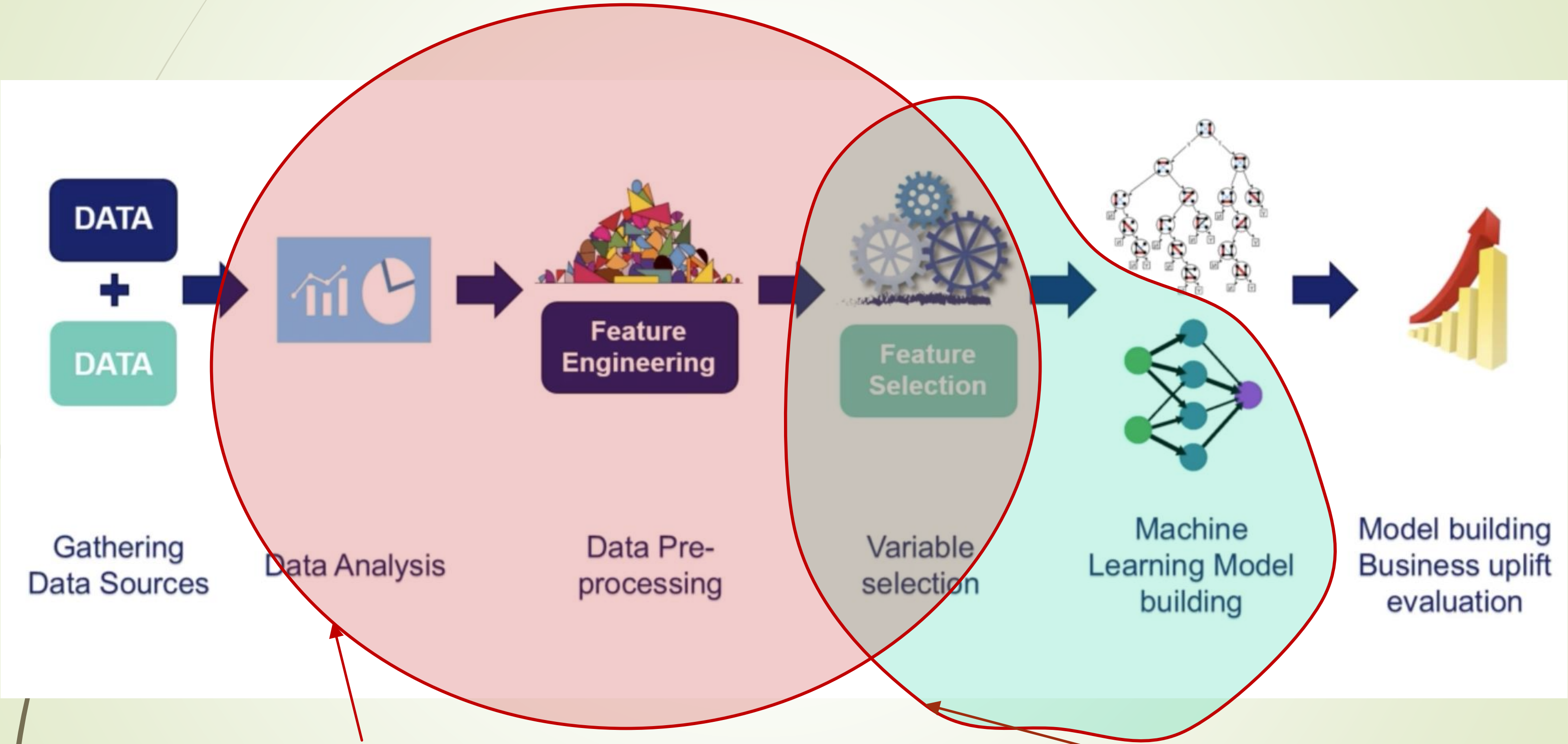
- Data Preparation Techniques
- Data Exploration
- Feature Engineering
  - Feature Encoding
  - Feature Transformation
  - Discretization and concept hierarchy generation
- Summary

# Forms and Types of Data

- Attribute-value data:
- Data types
  - numeric, categorical
- Other kinds of data
  - distributed data
  - text, web, metadata
  - images, audio/video
  - weather



# Machine learning Pipelines




Workshop focus (ML Ready Data)

*Feature selection is automatic for Deep learning models*

# Why Data Preprocessing?



- Real world is mostly dirty
  - incomplete: missing attribute values, lack of certain attributes of interest, or containing only aggregate data
    - e.g., occupation=""
  - noisy: containing errors or outliers
    - e.g., Salary="-10"
  - inconsistent: containing discrepancies in codes or names
    - e.g., Gender = "M" , "Male", Age="42" Birthday="03/07/1997"
    - e.g., Was rating "1,2,3", now rating "A, B, C"
    - e.g., discrepancy between duplicate records

# Why Data Preprocessing?

- Raw real world data lacks insights:
  - Quality decisions must be based on quality data: quality data should be void of missing data, duplicates or uninformative/misleading statistics
- Pattern recognition and Automated learning
  - Machine learning models feed off well data.
  - Data must be properly cleaned and encoded.
  - For successful pattern recognition, data must be representative of the underling task

# ML Ready Data

- Typical activities:
  1. Data Collection
  2. Data Preparation
     - Exploratory Analysis
     - Feature engineering
     - Feature Encoding

The above constitutes almost (90%) of the work in entire ML pipeline.

# Data Preparation tools

- Major Programming Languages
  - Python
  - R Language
- Major Software
  - Excel
  - Power BI
  - Tableau
  - Data Cleaner

# Useful Python Libraries for

- Pandas
- Polars
- Numpy
- Scipy
- Matplotlib
- Seaborn
- Imblearn
- Scikit-learn
- Feature_engine
- Lots more….

# Data Preparation

- Data Cleaning:
  - Importance
    - "Most Data come in raw, unstructured and unformatted with lost of inconsistencies, noise and redundancies."
  - Data cleaning tasks
    - Fill in missing values
    - Identify outliers and smooth out noisy data
    - Correct inconsistent data
    - Resolve redundancy caused by data integration

# Missing Data

- Data is not always available
  - E.g., many tuples have no recorded values for several attribute, such as customer income in sales data
- Missing data may be due to
  - equipment malfunction
  - inconsistent with other recorded data and thus deleted
  - data not entered due to misunderstanding
  - certain data may not be considered important at the time of entry
  - not register history or changes of the data

# How to Handle Missing Data?

- Ignore the tuple or delete the tuple
  - In pandas: df.dropna()
- Fill in missing values manually: tedious + can be infeasible
- Fill in it automatically with
  - a global constant : e.g., "unknown", a new class
  - the entire attribute mean, median or mode (May not a good idea)
- Missing data can be inferred
  - Based on closely corelated attribute (grouped corrected featured)
  - Interpolation: the most probable value: inference-based on models such as Bayesian formula, decision tree, linear or GP regression, etc.

# Noisy Data

- Noise: Random error in a measured variable.
- Sources of Nosie:
  - faulty data collection instruments
  - data entry problems
  - data transmission problems
  - technology limitation
  - inconsistency in naming convention

# How to Handle Noisy Data

- **Binning method**:
  - first sort data and partition into (equi-depth) bins
  - then one can smooth by bin means, median, boundaries, etc.
  - Technique is also used for discretization (discussed later)
- **Clustering**
  - detect and remove outliers
- **Semi-automated method**: combined computer and human inspection
  - detect suspicious values and check manually (may be tedious for large datasets)
- **Regression**
  - smooth by fitting the data into regression functions

# How to Handle Noisy Data?

- **Binning Method**:
  - Steps:
    - Sorted data. For example price variable with data: 4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34
    - Partition into (equi-depth) bins: Using the **price** example
      - Bin 1: 4, 8, 9, 15
      - Bin 2: 21, 21, 24, 25
      - Bin 3: 26, 28, 29, 34
    - Perform data smoothing via any of the following methods:
      - Means, Boundaries
    - Smoothing by bin means:
      - Bin 1: 9, 9, 9, 9
      - Bin 2: 23, 23, 23, 23
      - Bin 3: 29, 29, 29, 29
    - Smoothing by bin boundaries:
      - Bin 1: 4, 4, 4, 15
      - Bin 2: 21, 21, 25, 25
      - Bin 3: 26, 26, 26, 34

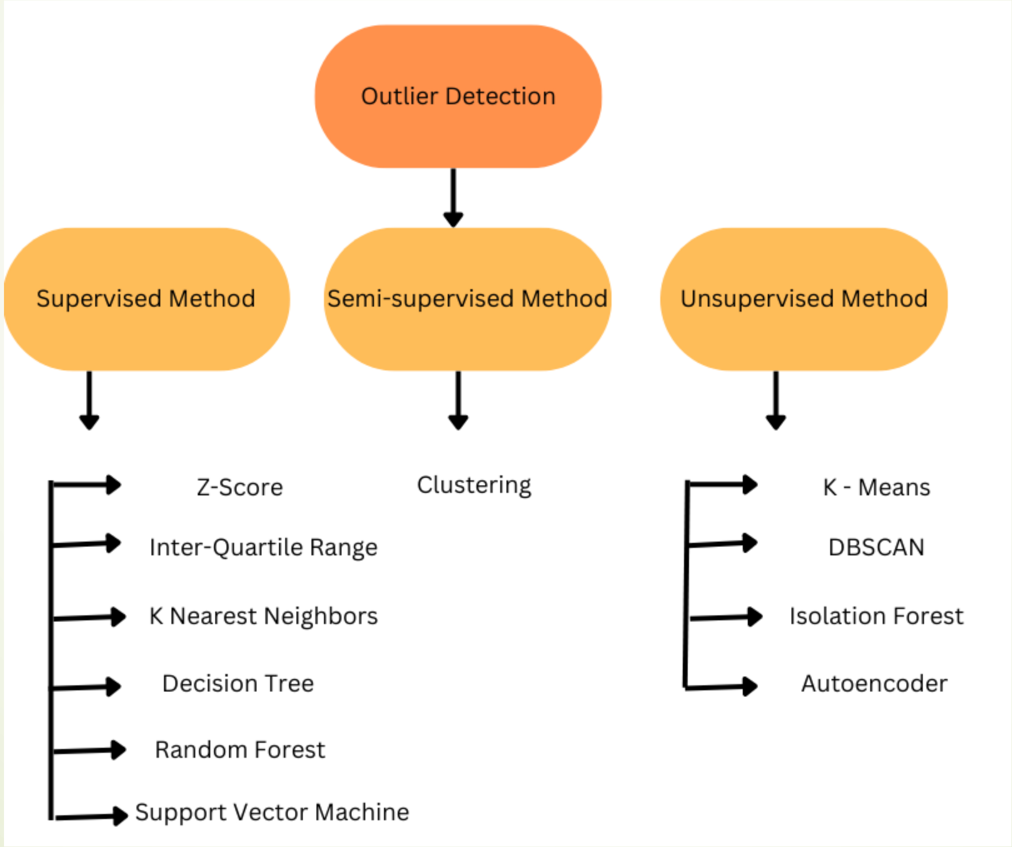
Identify the min and max per bin and replace each number in bin with the closest boundary

# How to Handle Noisy Data?

- Binning Method:
  - In Python using Pandas
    - Exploit the **pd.cut** function and the **groupby** function
    - For example:
      - df.groupby(pd.cut(df['A'], bins=2)).transform('mean')
      - df.groupby(pd.cut(df['A'], bins=2)).transform(median')
  - In Excel
    - Exploit Data Analytics suite of functions

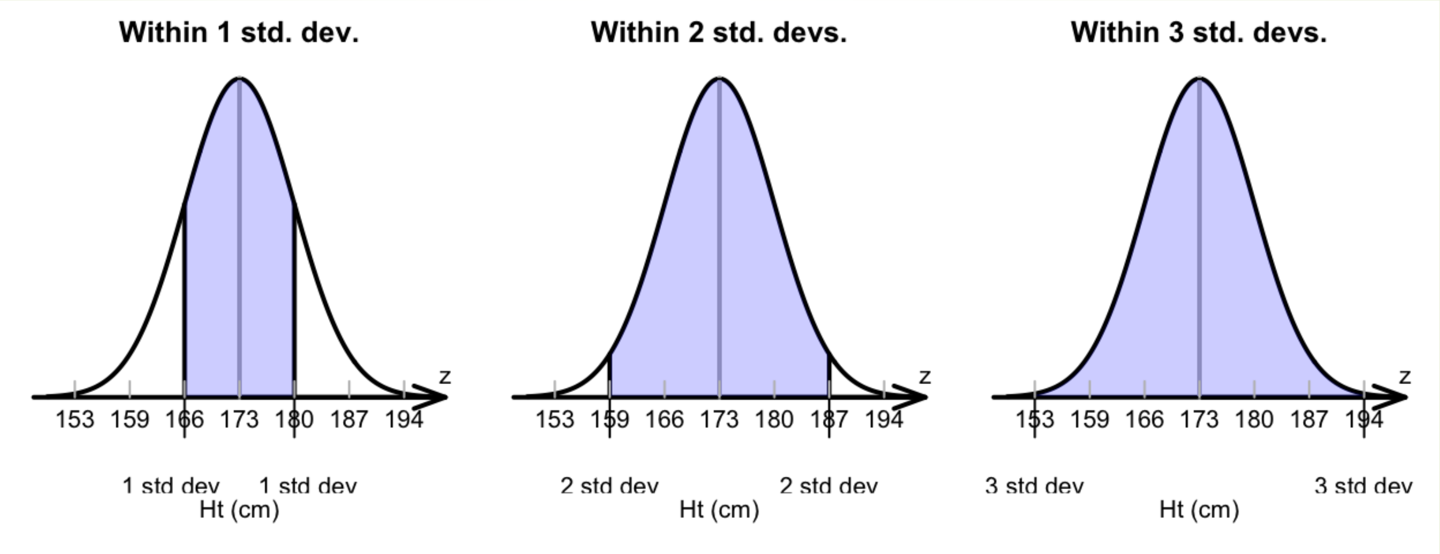
# How to Handle Noisy Data?

- Outlier removal methods:



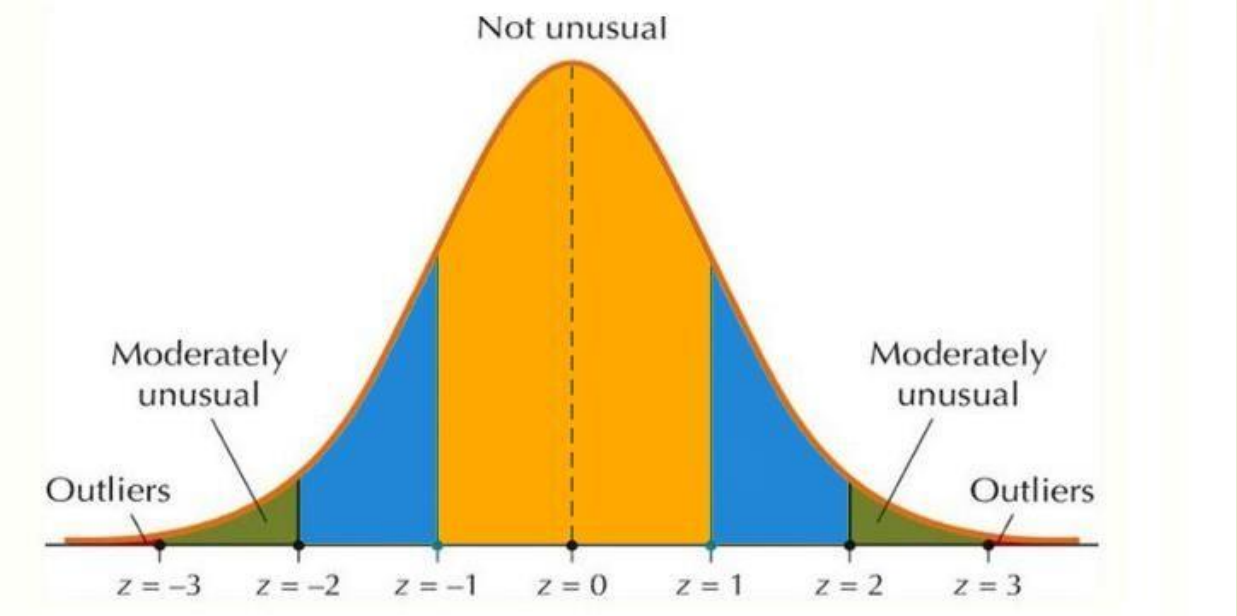
# How to Handle Noisy Data?

- Outlier removal methods:
  - Standard Deviation Approach
    - In Python: Filter the data based on standard deviation threshold



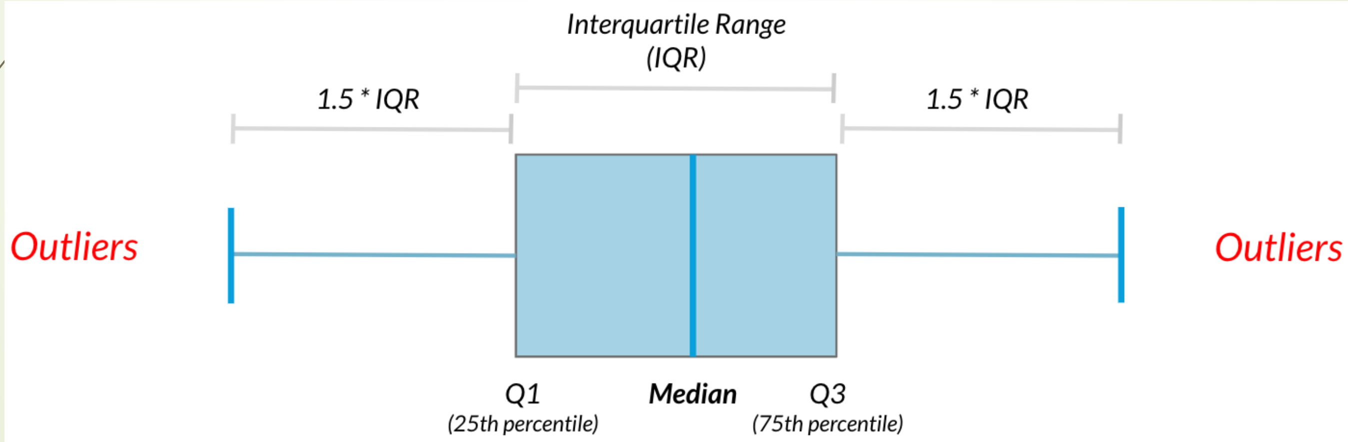
# How to Handle Noisy Data?

- Outlier removal methods:
  - Z – Score method:
    - $z = \frac{x - \mu}{\sigma}$



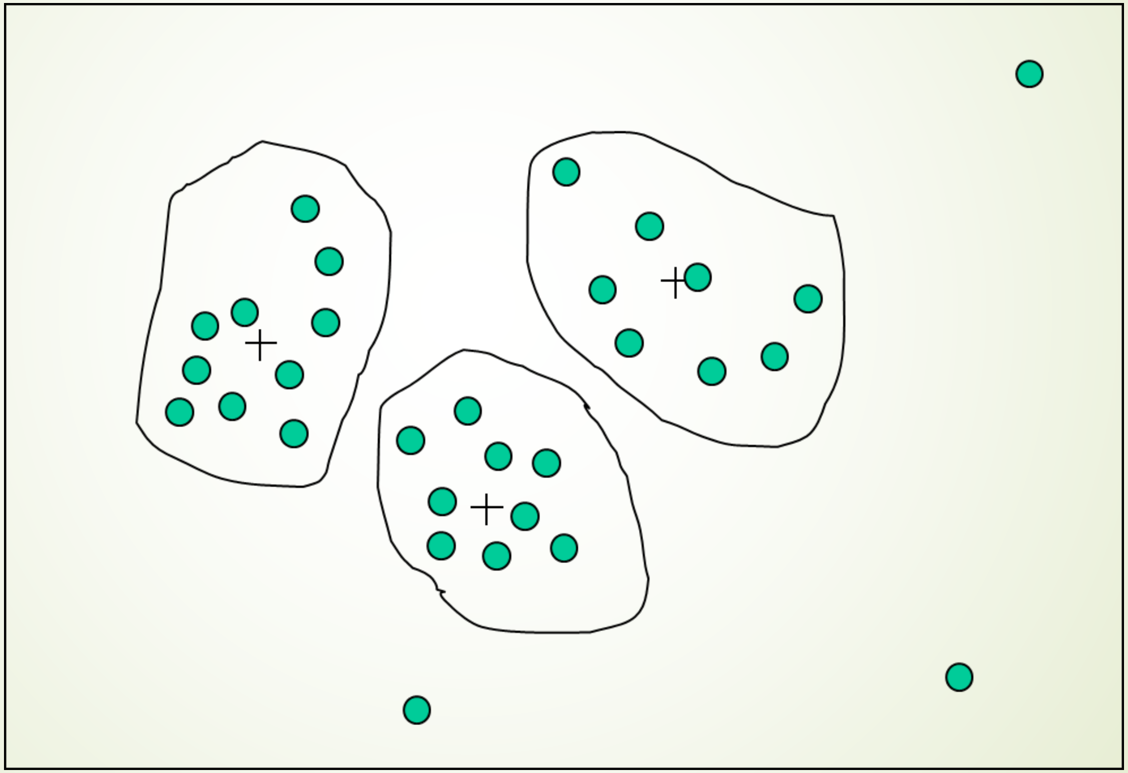
# How to Handle Noisy Data?

- Outlier removal methods:
  - Interquartile range:
    - Calculate the first and third quartiles (Q1 and Q3) of a dataset and then identify any data points that fall beyond the range of Q1 - 1.5 * IQR to Q3 + 1.5 * IQR. IQR = Q3 - Q1. Data points that fall outside of this range are considered outliers.



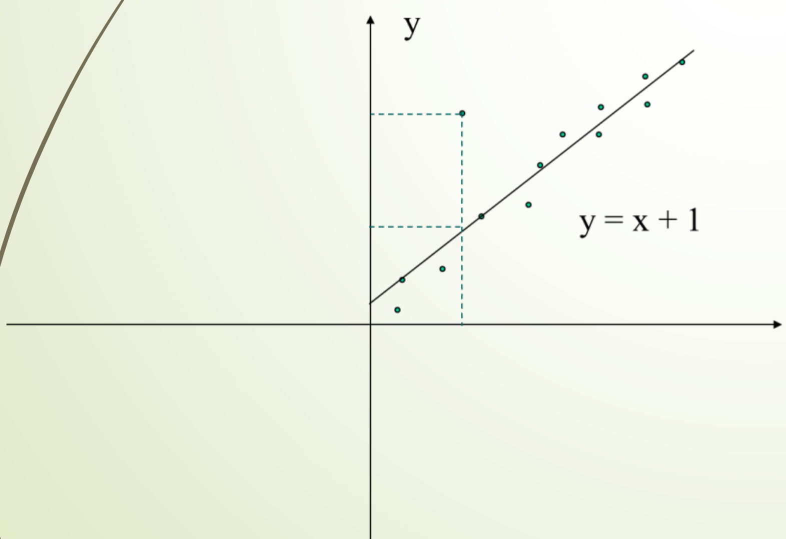
# How to Handle Noisy Data?

- Outlier removal methods:
  - Cluster Analysis:



# How to Handle Noisy Data?

- Outlier removal methods:
  - Regression:
    - Fit the entire dataset
    - Isolate the ones with error greater than a threshold
    - Tricky and difficult when true data manifold is non-linear




•Linear regression (best line to fit two variables)

•Multiple linear regression (more than two variables, fit to a multidimensional surface (hyperplane)

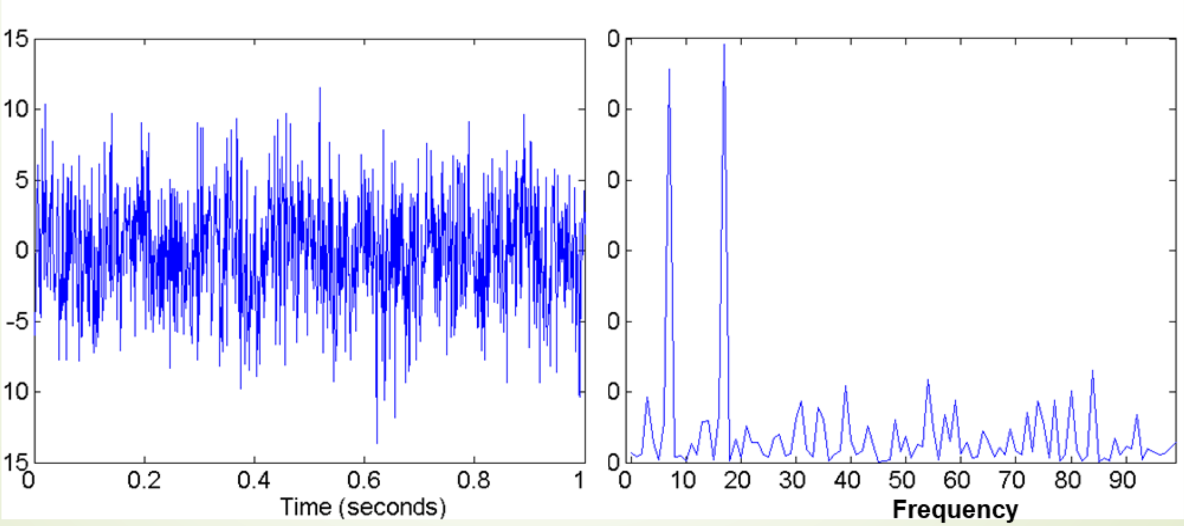
# How to Handle Noisy Data?

- Outlier removal methods:
  - Feature_engine Python library- https://feature-engine.trainindata.com/en/latest/api_doc/outliers/index.html

```
>>> import pandas as pd
>>> from feature_engine.outliers import OutlierTrimmer
>>> X = pd.DataFrame(dict(x = [0.49671,
>>>                         -0.1382,
>>>                          0.64768,
>>>                          1.52302,
>>>                         -0.2341,
>>>                         -17.2341,
>>>                          1.57921,
>>>                          0.76743,
>>>                         -0.4694,
>>>                          0.54256]))
>>> ot = OutlierTrimmer(capping_method='gaussian', tail='left', fold=3)
>>> ot.fit(X)
>>> ot.transform(X)
          x
0   0.49671
1  -0.13820
2   0.64768
3   1.52302
4  -0.23410
5 -17.23410
6   1.57921
7   0.76743
8  -0.46940
9   0.54256
```

# How to Handle Noisy Data?

- Map to a new space:
  - Typical useful for image, audio and video data
  - Common techniques involves
    - Convolution filters (fixed or learnable parameters)
    - Fourier Transforms
    - Wavelet Transforms

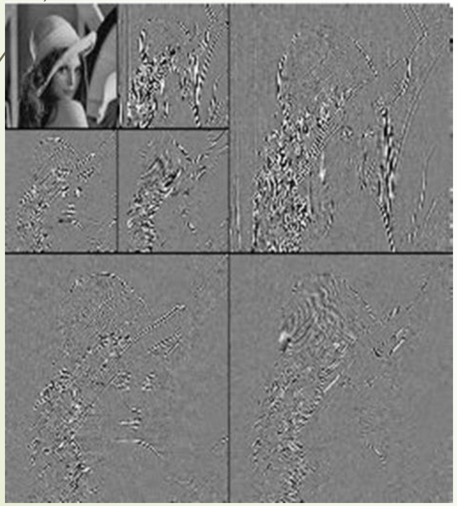



Fourier transform from time to frequency domain

# How to Handle Noisy Data?



- Map to a new space:
  - Typical useful for image, audio and video data
  - Common techniques involves
    - Convolution filters
    - Fourier Transforms
    - Wavelet Transforms



- Decomposes a signal into different frequency subbands
  - Applicable to n-dimensional signals
- Data are transformed to preserve relative distance between objects at different levels of resolution
- Allow natural clusters to become more distinguishable.
- Technique also used for image compression

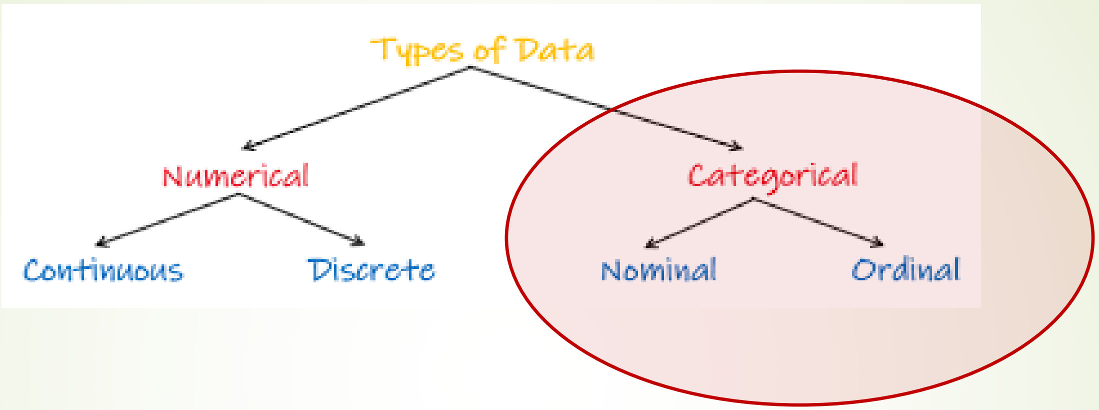
# Inconsistent Data

- This problem is not so common with image/audio data
- Common solution:
  - Manual correction using external references
  - Semi-automatic using various tools
    - To detect violation of known functional dependencies and data constraints
    - To correct redundant data
  - Using regx with search and replace: especially for text based data (easy in pandas and polar)

# Feature Engineering

- Typical activities:
  - Feature Encoding
  - Feature Transformation

# Feature Encoding




- **Categorical Data**: Categorical variables represent types of data which may be divided into groups. Examples: race, sex, age group, and educational level.
- **Types:**
  - Ordinal – Inherent order in the categories. Example: education level, income range, age, grades, etc.
  - Nominal – No inherent order in the categories. Example: male/female, nationalities, race, state of origin, etc.

# Encoding Techniques

- **Methods of Encoding**:
  - **One-Hot Encoding**: Represents categorical features with N columns where N is the number of categories. Typically used nominal categorical features (categories without intrinsic order)
  - **Dummy Encoding**: Also used for norminal categorical feature but with N-1 columns. Where N is the number of categories
  - **Ordinal Encoding**: Represents categorical features with unique integer. Typically used for ordinal categorical features (categories with intrinsic order)
  - **Binary Encoding**: Represents each categorical feature with a rank in binary format. For ordinal categories where a category's value is its rank in binary format
  - **Count Encoding**: Technique replaces the categories of categorical features by their counts, which are estimated from the training set.
  - **Frequency Encoding**: It is a way to utilize the frequency of the categories as labels. In the cases where the frequency is related somewhat with the target variable. For example ***Country*** varible, if Nigeria appears in 10% of the observations and Kenya in 1%, Nigeria will be replaced by 0.1 and Kenya with 0.01.
  - **Target encoding:** converting each category of a categorical feature into its corresponding expected value based on another target variable.

# One-Hot Encoding

- One-Hot Encoding: Most common encoding scheme used for nominal categorical data
    - Transforms the categorical variable into a set of binary variables [0/1], one column for each category
  - Drawbacks:
    - Sparsity, Curse of dimensionality and information loss (when used for ordinary categories)

**One-Hot Encoding**

| Places |
|---|
| New York |
| Boston |
| Chicago |
| California |
| New Jersey |

| New York | Boston | Chicago | California | New Jersey |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 |

Python:

Use OneHotEncoder(..) from sklearn

Use pandas.get_dummies() methods to create one_encoded columns:

Example:

```
df_encoded = pd.get_dummies(df, columns=['categorical_column', ])
```

Original dataframe

Columns to encode



img src:https://medium.com/aiskunks/

# Dummy Encoding

- **Dummy Encoding**: Similar to one-hot encoding but with a slight improvement
  - Dummy encoding uses **N-1** features to represent N labels/categories.

Dummy Encoding

| Places |
|---|
| New York |
| Boston |
| Chicago |
| California |
| New Jersey |

| New York | Boston | Chicago | California | New Jersey |
|---|---|---|---|---|
| | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 0 |
| | 0 | 1 | 0 | 0 |
| | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 |

```
# Use get_dummies() function for dummy encoding
dummy_df = pd.get_dummies(df['Color'], drop_first=True, prefix='Color')
```

Equivalent to drop='first' in OneHotEncoder

```
>>> drop_enc = OneHotEncoder(drop='first').fit(X)
```



img src:https://medium.com/aiskunks/

# Label Encoding

- **Label Encoding**: Encodes categorical features as unique integers and its typically used for nominal data: eg. Color (red, blue, green).
  - Drawbacks:
    - Can be misleading for machine learning algorithm if used for ordinal categories
  - If you need to impose order in integer categories then use **Ordinal Encoder (Discussed next)**

**Label Encoding**

| Places |
|---|
| New York |
| Boston |
| Chicago |
| California |
| New Jersey |

| Places | Map |
|---|---|
| New York | 1 |
| Boston | 2 |
| Chicago | 3 |
| California | 4 |
| New Jersey | 5 |

| Places | Encoded |
|---|---|
| New York | 1 |
| Boston | 2 |
| New York | 1 |
| California | 4 |
| Boston | 2 |

Example: color encoding

```
from sklearn.preprocessing import LabelEncoder

# Create a sample dataframe with categorical data
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red', 'green']})

print(f"Before Encoding the Data:\n\n{df}\n")

# Create a LabelEncoder object
le = LabelEncoder()

# Fit and transform the categorical data
df['color_label'] = le.fit_transform(df['color'])
```



img src:https://medium.com/aiskunks/

# Ordinal Encoding

- **Ordinal Encoding**: Encodes categorial features as ordinal integers based on the order of the categories.
  - For example, temperature variable with values ‘Low’, ‘Medium’ and ‘High’, can be assigned the values 1, 2, and 3, respectively.

**Ordinal Encoding**

| Grades |
|---|
| A |
| B |
| C |
| D |
| [illegible] |

| Grades | Encoded |
|---|---|
| A | 4 |
| B | 3 |
| C | 2 |
| D | 1 |
| [illegible] | 0 |

Note: case sensitivity

```
# Ordinal Encoding:
# create a sample dataframe with a categorical variable
df = pd.DataFrame({'quality': ['low', 'medium', 'high', 'medium']})
print(f"Before Encoding the Data:\n\n{df}\n")

# specify the order of the categories
quality_map = {'low': 0, 'medium': 1, 'high': 2}

# perform ordinal encoding on the 'quality' column
df['quality_map'] = df['quality'].map(quality_map)
```

In sklearn: Use OrdinalEncoder class
Note: categories array param that handles order via item index

```
from sklearn.preprocessing import OrdinalEncoder
enc = OrdinalEncoder(categories=[['first','second','third','forth']])
X = [['third'], ['second'], ['first']]
enc.fit(X)
print(enc.transform([['second'], ['first'], ['third'],['forth']]))
```

img src:https://medium.com/aiskunks/

# Binary Encoding

- **Binary Encoding**: Similar to binary encoding but instead of creating a separate column for each category, the categories are represented as binary digits.
  - For example, consider a variable with categories 'A', 'B', 'C' and 'D', each unique category can be represented as 0001, 0010, 0100 and 1000, respectively.

**Binary Encoding**

| Places |
|---|
| New York |
| Boston |
| Chicago |
| California |

| Feature | Map |
|---|---|
| New York | 0 |
| Boston | 1 |
| Chicago | 2 |
| California | 3 |

| Feature | Encoded |
|---|---|
| New York | 0 |
| Boston | 1 |
| Chicago | 10 |
| California | 11 |

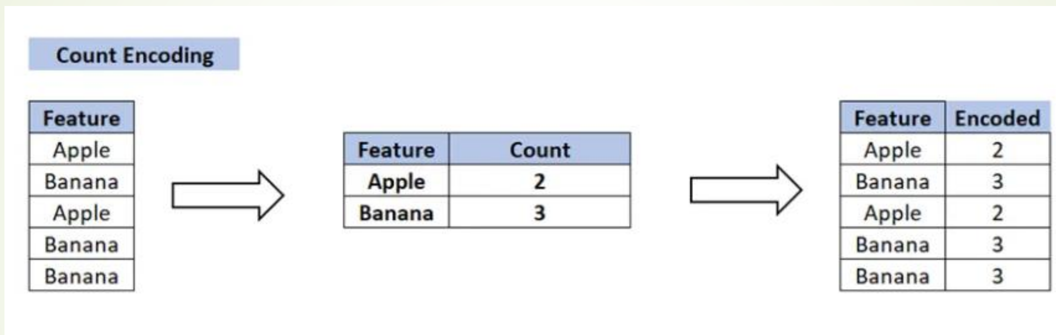
```python
import pandas as pd

# create a sample dataframe with a categorical variable
df = pd.DataFrame({'animal': ['cat', 'dog', 'bird', 'cat']})
print(f"Before Encoding the Data:\n\n{df}\n")

# perform binary encoding on the 'animal' column
animal_map = {'cat': 0, 'dog': 1, 'bird': 2}
df['animal'] = df['animal'].map(animal_map)
df['animal'] = df['animal'].apply(lambda x: format(x, 'b'))

# print the resulting dataframe
print(f"After Encoding the Data:\n\n{df}\n")
```



img src:https://medium.com/aiskunks/

# Count Encoding

- **Count Encoding**: Counts the number of occurrence of a category
  - For example, consider the variable with categories ‘A’, ‘B’ and ‘C’ and category ‘A’ appears 10 times in the dataset, it will be assigned a value of 10.
  - This technique is often used for natural language tasks.



Count Encoding

| Feature |
|---|
| Apple |
| Banana |
| Apple |
| Banana |
| Banana |

| Feature | Count |
|---|---|
| Apple | 2 |
| Banana | 3 |

| Feature | Encoded |
|---|---|
| Apple | 2 |
| Banana | 3 |
| Apple | 2 |
| Banana | 3 |
| Banana | 3 |

pandas

```
# Count Encoding:
# create a sample dataframe with a categorical variable
df = pd.DataFrame({'fruit': ['apple', 'banana', 'apple', 'banana']})
print(f"Before Encoding the Data:\n\n{df}\n")

# perform count encoding on the 'fruit' column
counts = df['fruit'].value_counts()
df['fruit'] = df['fruit'].map(counts)

# print the resulting dataframe
print(f"After Encoding the Data:\n\n{df}\n")
```

from feature_engine

```
encoder = CountFrequencyEncoder(
encoding_method='count',
variables=['cabin', 'sex', 'embarked'],
)

encoder.fit(X_train)
```



img src:https://medium.com/aiskunks/

# Target Encoding

- **Target Encoding**: A more advanced encoding technique used for dealing with high cardinality categorical features, i.e., features with many unique categories.
  - The average target value for each category is calculated and result used to replace the categorical feature.
  - This has the advantage of considering the relationship between the target and the categorical feature, but it can also lead to overfitting if not used with caution.

**Target Encoding**

| Feature | Target |
|---|---|
| Apple | 0 |
| Banana | 1 |
| Apple | 0 |
| Banana | 0 |
| Banana | 1 |

| Feature | Average(Target) |
|---|---|
| Apple | 0 |
| Banana | 2/ 3 = 0.66 |

| Feature | Encoded |
|---|---|
| Apple | 0 |
| Banana | 0.66 |
| Apple | 0 |
| Banana | 0.66 |
| Banana | 0.66 |

```
# Create a sample dataframe with categorical data and target
df = pd.DataFrame({'color': ['red', 'green', 'blue', 'red', 'green'],
                   'target': [0, 1, 0, 1, 0]})
print(f"Before Encoding the Data:\n\n{df}\n")

# Calculate the mean target value for each category
target_mean = df.groupby('color')['target'].mean()

# Replace the categorical data with the mean target value
df['color_label'] = df['color'].map(target_mean)


print(f"After Encoding the Data:\n\n{df}")
```

from sklearn

```
from sklearn.preprocessing import TargetEncoder
X = np.array([["dog"] * 20 + ["cat"] * 30 + ["snake"] * 38], dtype=object).T
y = [90.3] * 5 + [80.1] * 15 + [20.4] * 5 + [20.1] * 25 + [21.2] * 8 + [49] * 30
enc_auto = TargetEncoder(smooth="auto")
X_trans = enc_auto.fit_transform(X, y)
```

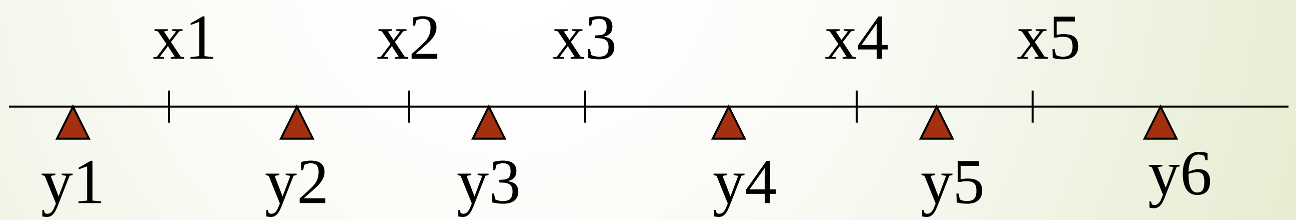
img src:https://medium.com/aiskunks/

# More Encodings

- **More Encodings Available**: See the following
  - sklearn.preprocessing https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing
  - Feature_engine https://feature-engine.trainindata.com/en/latest/

# Discretization/Quantization

- Three types of attributes:
  - Nominal — values from an unordered set
  - Ordinal — values from an ordered set
  - Continuous — real numbers
- Discretization/Quantization:
  - divide the range of a continuous attribute into intervals




  - Some classification algorithms only accept categorical attributes.
  - Reduce data size by discretization
  - Prepare for further analysis

# Discretization Techniques

- Equal-width (distance) binning
  - It divides the range into *N* intervals of equal size: uniform grid
  - if *A* and *B* are the lowest and highest values of the attribute, the width of intervals will be: $W = (B-A)/N$.
  - The most straightforward
  - But outliers may dominate presentation
  - Skewed data is not handled well.
- Equal-depth (frequency) partitioning:
  - Quantile based
  - It divides the range into *N* intervals, each containing approximately same number of samples
  - Good data scaling
  - Managing categorical attributes can be tricky.

# Discretization and Concept Hierarchy

- Discretization
  - reduce the number of values for a given continuous attribute by dividing the range of the attribute into intervals. Interval labels can then be used to replace actual data values.
- Concept Hierarchies
  - reduce the data by collecting and replacing low level concepts (such as numeric values for the attribute age) by higher level concepts (such as young, middle-aged, or senior).

# Discretization

- For implementation and more techniques see:
  - https://feature-engine.trainindata.com/en/latest/api_doc/discretisation/index.html

# Redundant Data Analysis

- Redundant data occur often when integrating multiple DBs
  - The same attribute may have different names in different databases
  - One attribute may be a "derived" attribute in another table, e.g., annual revenue
- Redundant data may be able to be detected by correlational analysis and covariance analysis
- Careful integration can help reduce/avoid redundancies and inconsistencies and improve mining speed and quality

# $\chi^2$ Correlation Analysis (Nominal Data)

- Chi-square: Hypothesis test used to determine whether there is a relationship between two random categorical and nominal variables.
- Textual categorical data, need to be converted to **contingency table** containing normal values.
- Can be use as a test of independence
- Can also be used to evaluate confusion matrix

$$\chi^2 = \sum_{k=1}^{n} \frac{(O_k - E_k)^2}{E_k}$$

$$where\ O_k\ are\ the\ Observed\ and\ E_k\ are\ the\ expected\ values$$

- The larger the chi-square value, the most like the variables are related
- The cells that contribute the most to the chi-square value are those whose actual count is different from the expected count
- Note: Correlation does not imply causality

# Chi-Square Calculation: An Example



| | Play chess | Not play chess | Sum (row) |
|---|---|---|---|
| Like science fiction | 250(90) | 200(360) | 450 |
| Not like science fiction | 50(210) | 1000(840) | 1050 |
| Sum(col.) | 300 | 1200 | 1500 |

- Chi-square calculations: Table above is the contingency table. The numbers in parenthesis are expected counts calculated based on the data distribution in the two categories (like_science_fiction and play_chess).
- Expected values can be computed using numpy.outer between the row sum and column sum divide by total

- $\chi^2 = \frac{(250-90)^2}{90} + \frac{(50-210)^2}{210} + \frac{(200-360)^2}{360} + \frac{(1000-840)^2}{840} = 507.94$
- It shows that like_science_fiction and play_chess are correlated in the group

Using scipy library

```
[1]: import numpy as np
     from scipy.stats import chisquare
     f_obs = np.array([250, 50, 200, 1000])
     f_exp = np.array([90, 210, 360, 840])
     chisquare(f_obs=f_obs, f_exp=f_exp, ddof=2)

[1]: Power_divergenceResult(statistic=507.93650793650795, pvalue=1.7830898208664246e-112)
```

# Correlation Analysis (Continous Data)



- Correlation coefficient (also called Pearson's product moment coefficient)

$$r_{A,B} = \frac{\sum_{i=1}^{n}(a_i - \bar{A})(b_i - \bar{B})}{(n-1)\sigma_A \sigma_B} = \frac{\sum_{i=1}^{n}(a_i b_i) - n\bar{A}\bar{B}}{(n-1)\sigma_A \sigma_B}$$

where n is the number of tuples, and are the respective means of A and B, $\sigma_A$ and $\sigma_B$ are the respective standard deviation of A and B, and $\Sigma(a_i b_i)$ is the sum of the AB cross-product.

- If $r_{A,B} > 0$, A and B are positively correlated (A's values increase as B's). The higher, the stronger correlation.
- $r_{A,B} = 0$: independent; $r_{AB} < 0$: negatively correlated

# Correlation Analysis (Continous Data)



- In Python:
  - np.corrcoef(x,y)  # in numpy
  - scipy.stats.pearsonr(x, y)  #pearson's corr in scipy (assumption: data follows normal distribution
  - Dataframe.corr(numeric_only=True) #pandas
  - Note: Transpose data for tupple correlation
- Other Correlation coefficients
  - **Spearman**: scipy.stats.spearmanr(x, y)  # Spearman's rho: Measures monotonic relationbship and can handle ordinal data
  - **Kendal:** scipy.stats.kendalltau(x, y)  # Kendall's tau Spearman's rho: Measures monotonic relationbship and can handle ordinal data

# Data Transformation

- Smoothing: remove noise from data (binning, clustering, regression)
- Aggregation: summarization, data cube construction
- Generalization: concept hierarchy climbing
- Normalization: scaled to fall within a small, specified range
  - min-max normalization
  - z-score normalization
  - normalization by decimal scaling
- Attribute/feature construction
  - New attributes constructed from the given ones

# Data Transformation: Normalization

Particularly useful for classification (NNs, distance measurements, nn classification, gradient descent based models, etc)

- min-max normalization

$$v' = \frac{v - \min(A)}{\max(A) - \min(A)} \left(max_{new}(A) - min_{new}(A)\right) + min_{new}(A)$$

*where* $max_{new}$ *and* $min_{new}$ *are the* new max and min for feature A

- z-score normalization

$$v' = \frac{v - \mu_A}{\sigma_A}$$

*where* $\mu_A$ *and* $\sigma_A$ *are* mean and std of feature A

- normalization by decimal scaling

$$v' = \frac{v}{10^j}$$

Where $j$ is the smallest integer such that Max($|v'|$)<1

# Feature Selection

- Selecting a subset of relevant features for use in model construction
- Typically useful for non-deep learning models
- Key Idea:
  - Select a minimum set of features such that the probability distribution of different classes given the values for those features is as close as possible to the original distribution given the values of all features
  - Nice side-effect: reduces # of attributes in the discovered patterns (which are now easier to understand)
- Methods:
  - Heuristics search
  - Principal component analysis PCA

# Principal Component Analysis (PCA)

- Idea:
  - Find c<=k orthogonal vectors that can be summarize N datapoints with K features.
  - The original data set is reduced (projected) to one consisting of N data vectors on c principal components (reduced dimensions)
  - Each data vector is a linear combination of the c principal component vectors
  - Works for ordered and unordered attributes
  - Used when the number of dimensions is large

# Need to know before data preparation

- Understand the Data: Gain a comprehensive understanding of the dataset, its history, including its structure, variables, and meaning. This involves reading any documentation available and exploring the data visually.
- Define a clear objective: Clearly define the goals of your analysis or modeling task.
- Determine which features are relevant to your objectives and which ones can be excluded.

# Summary

- Data preparation constitutes the bulk part of building usable ML models
- Data preparation includes
  - Data cleaning
  - Feature Encoding
  - Feature Transformation and discretization
  - Data feature selection
- Understanding the history and semantics of data is key to preparing ML informative data.

# Additional Resources

- Data Mining by Charu C. Aggarwal
- Pandas https://pandas.pydata.org/docs/
- Feature engines https://feature-engine.trainindata.com/en/latest/