

Feature engineering

Lecture 12

IFT 6758

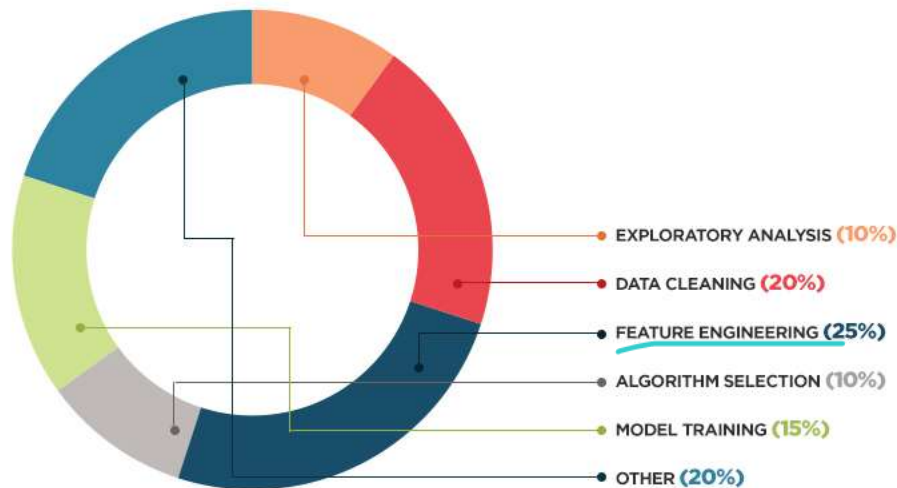
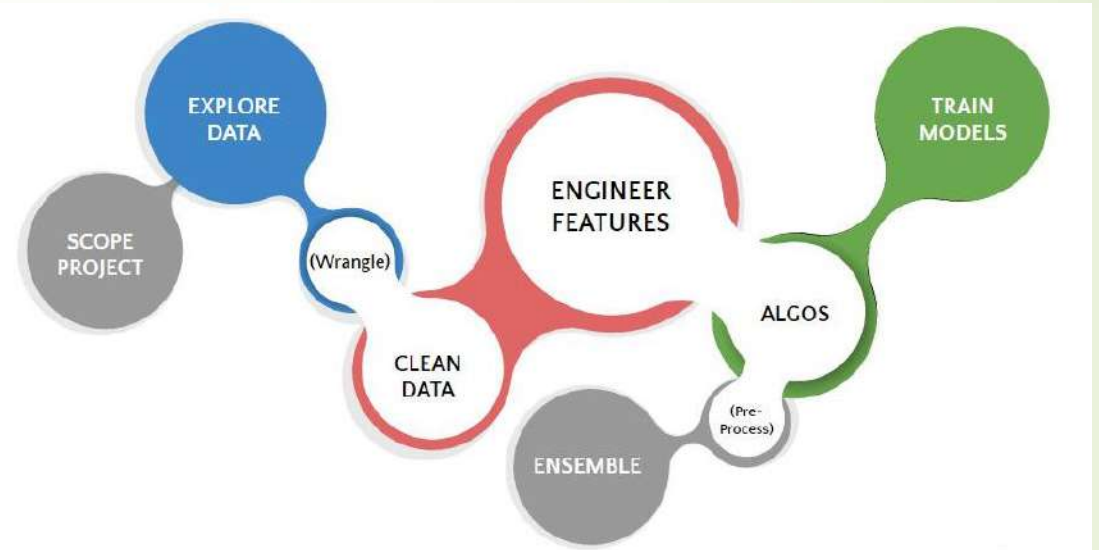




ML Workflow and Feature Engineering (FE)



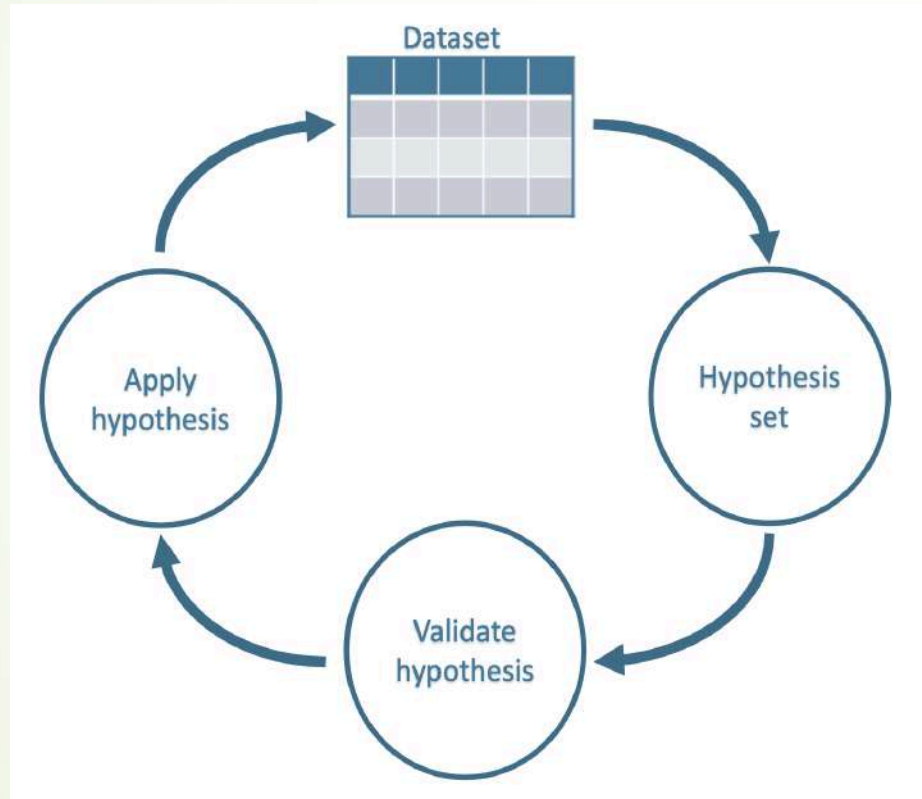
ML Workflow and Feature Engineering (FE)



Every unique attribute of the data is considered a *feature*

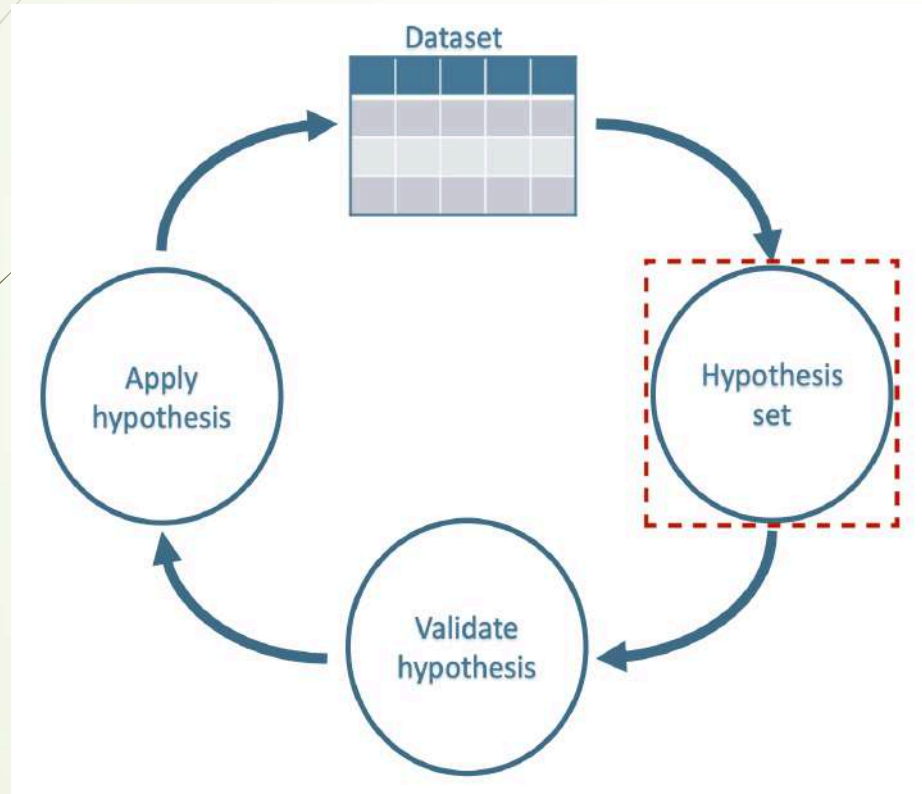
Feature eng. is diff. from data preprocessing/cleaning

Feature Engineering Cycle



- Data preprocessing (not FE)
 - Data collection
 - Creating target variable
 - Removing duplicates
 - Fixing mislabeled classes

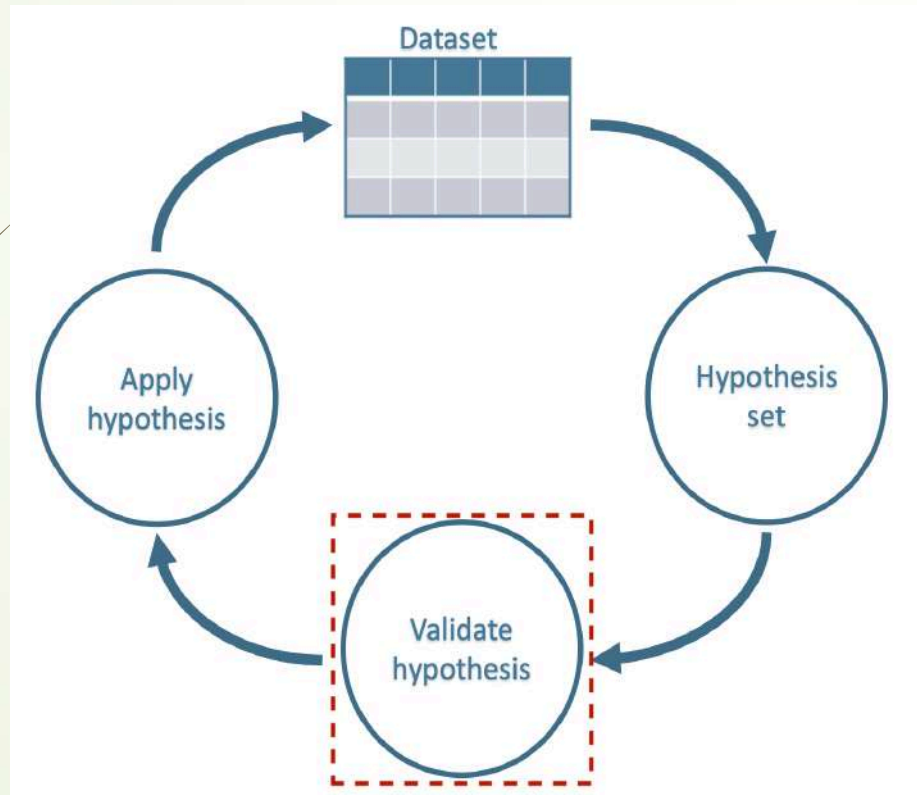
Feature Engineering Cycle



How?

- Domain knowledge
- Prior experience
- Exploratory Data Analysis (EDA)
- ML model feedback

Feature Engineering Cycle



How?

- Cross validation
- Measurement of desired metrics
- Avoid leakage

(Dropping valuable information)

Feature Engineering is hard but worthy

losing important info.



Why hard?

- Powerful **feature transformations** (like target encoding) **can introduce leakage** when applied wrong
- Usually **requires domain knowledge** about how features interact with each other
- **Time-consuming**, need to run thousand of experiments

Why worthy?

- Extract relevant features, **remove irrelevant or noisy features**
- **Simpler models** with better results



Key Elements of Feature Engineering

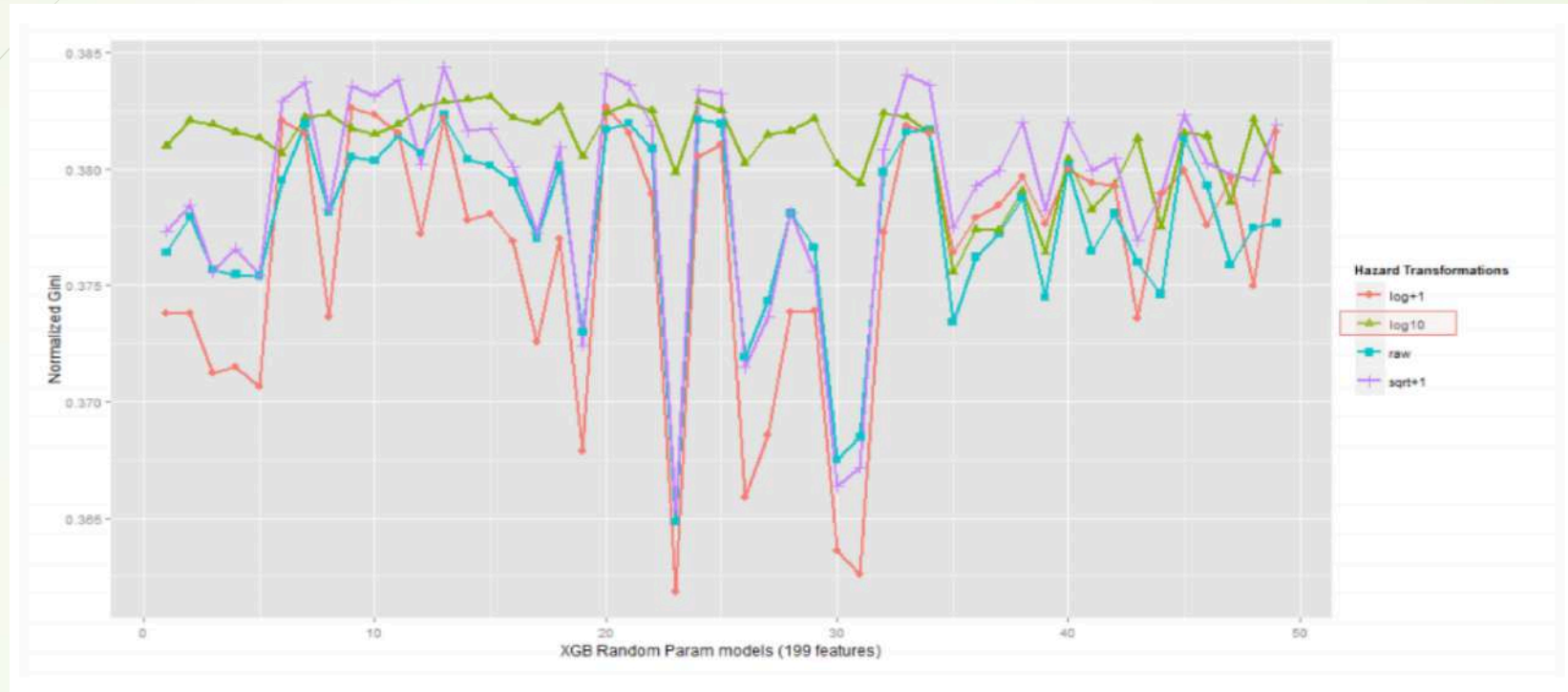
- Target transformation
 - Feature extraction
 - Feature encoding
- 

Target transformation

<https://towardsdatascience.com/catalog-of-variable-transformations-to-make-your-model-works-better-7b506bf80b97>

- Predictor/Response **Variable Transformation**
 - [Catalog for variable transformation](#)
- Use it when variable shows a **skewed distribution** make the residuals **closer to “normal distribution”** (bell curve)
- Can **improve model fit**
 - Example: $\log(x)$, $\log(x + 1)$, \sqrt{x} , $\sqrt{(x + 1)}$

Target transformation



Different transformations might lead to different results

Feature Extraction

- Common problem in preparing the data: **Missing Values**
- Why we have missing values?
 - Human errors
 - Interruptions in the data flow
 - Privacy concerns

Remedies?

- Simple solution: **drop** the row/column
- Preferable option: **Imputation**





Imputation

➤ Numerical imputation


- Assigning zero
- Assigning NA
- Default value or medians of the columns (Note that averages are sensitive to outlier values)

➤ Categorical imputation

- Replace with maximum occurred value
- If there is not a dominant value, imputing a category like "Other"



Some other techniques of feature extraction

- Outlier detection and removal (in details)
 - Binning
 - Log Transformation
 - Grouping
 - Splitting
 - Scaling
 - Normalization
 - Standardization
- 

Outliers

- Outliers creep in during data collections

Players	Scores
Player1	500
Player2	350
Player3	10
Player4	300
Player5	450

Possible reason?

- Mistake?
- Variance in the data?

Outliers

- **Outlier:** A data object that deviates significantly from the normal objects as if it were generated by a different mechanism
 - Example: Unusual credit card purchase, sports: Michael Jordon, Wayne Gretzky, ...
- **Outliers are different from the noise data.**
 - Noise is random error or variance in a measured variable
 - Noise should be removed before outlier detection
- **Outliers are interesting.** It violates the mechanism that generates the normal data
 - Outlier detection vs. novelty detection: early stage, outlier; but later merged into the model
- **Applications:**
 - Credit card fraud detection
 - Telecom fraud detection
 - Customer segmentation
 - Medical analysis

Types of outliers

- **Global, contextual and collective** outliers
- **Global outlier (or point anomaly):**
 - Object is O_g if it significantly deviates from the rest of the data set
 - **Example:** Intrusion detection in computer networks
 - **Issue:** Find an appropriate measurement of deviation
- **Contextual outlier (or conditional outlier):**
 - Object is O_c if it deviates significantly based on a selected context
 - **Example:** 80°F in Urbana: outlier? (depending on summer or winter?)
 - **Attributes** of data objects should be divided into two groups
 - Contextual attributes: defines the context, e.g., time & location
 - Behavioral attributes: characteristics of the object, used in outlier evaluation, e.g., temperature
 - Can be viewed as a **generalization of local outliers**—whose density significantly deviates from its local area
 - **Issue:** How to define or formulate meaningful context?

Types of outliers

➤ **Collective Outliers:**

- A subset of data objects collectively deviates significantly from the whole data set, even if the individual data objects may not be outliers

- **Applications:** E.g., intrusion detection:

- When a number of computers keep sending denial-of-service packages to each other

- **Detection** of collective outliers

- Consider not only behavior of individual objects, but also that of groups of objects
 - Need to have the background knowledge on the relationship among data objects, such as a distance or similarity measure on objects.

- A data set **may have multiple types** of outlier

- **One object may belong to more than one type** of outlier

Finding outliers

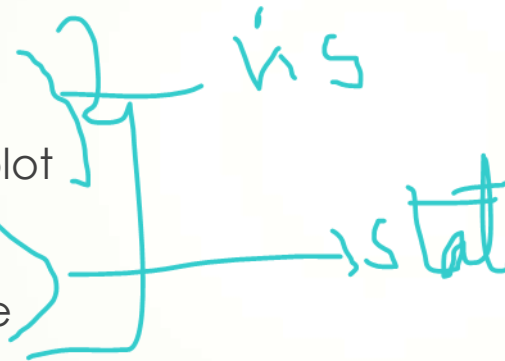
- **Visualization tools** (some mentioned in earlier lectures) or **statistical methodologies**

- Box plot

- Scatter plot

- Z-score

- IQR score

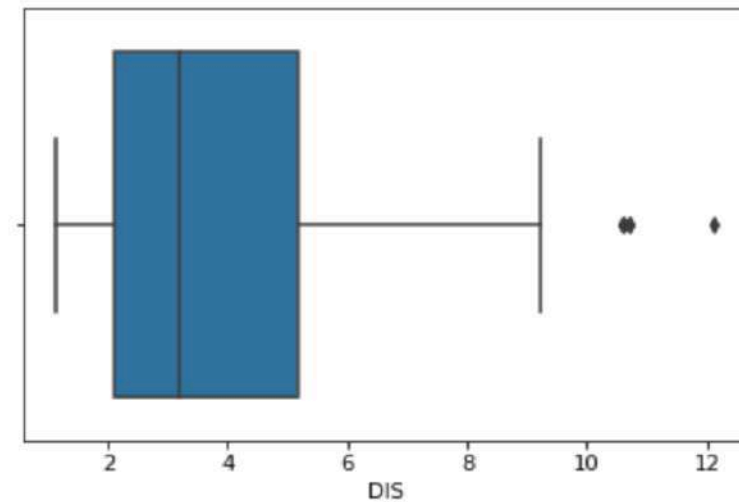


- **Advanced techniques** for anomaly detection (after mid-term)

Box plot

- Univariate plot.
- What about multivariate plot?
- Categorical variables along with continuous variables.

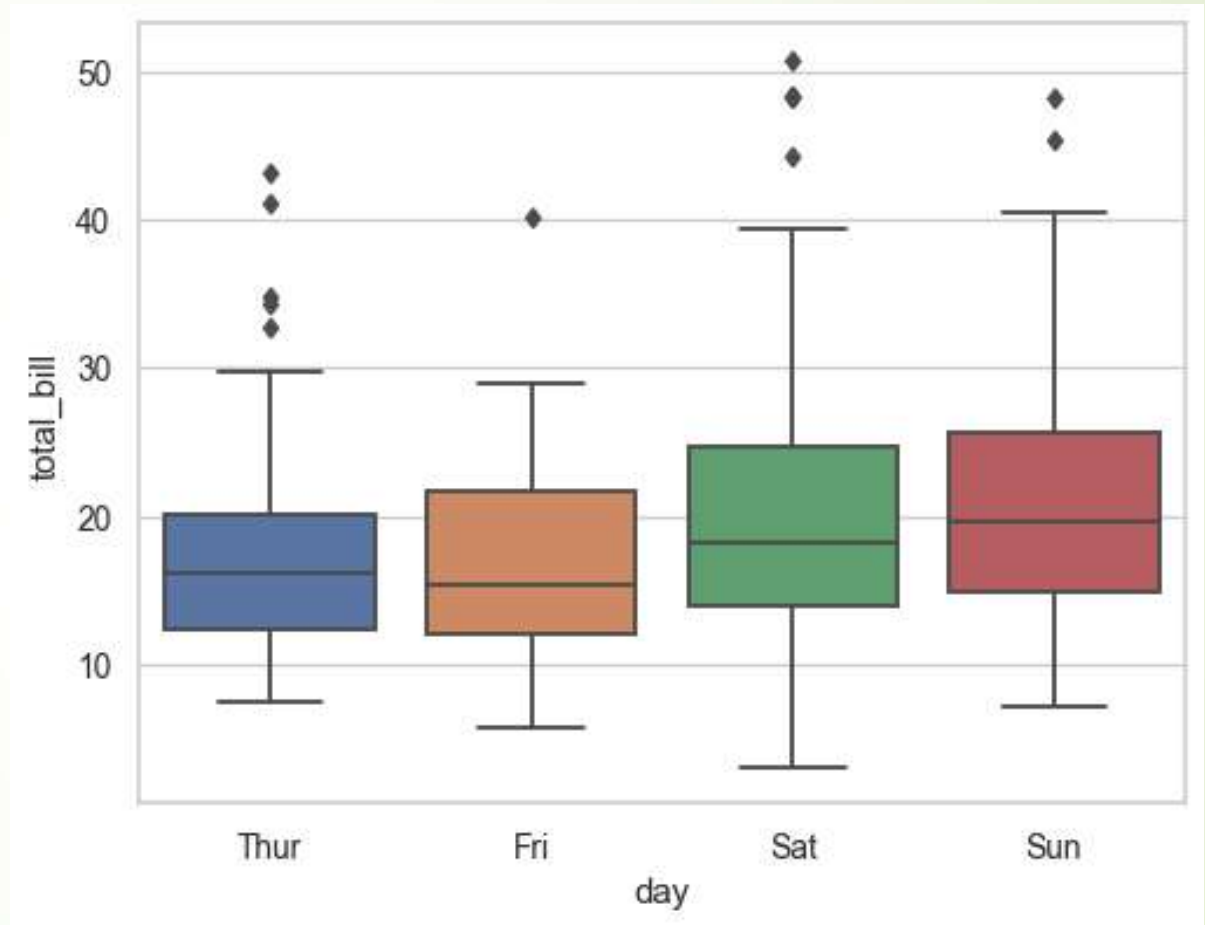
```
import seaborn as sns
sns.boxplot(x=boston_df['DIS'])
```



Boxplot — Distance to Employment Center

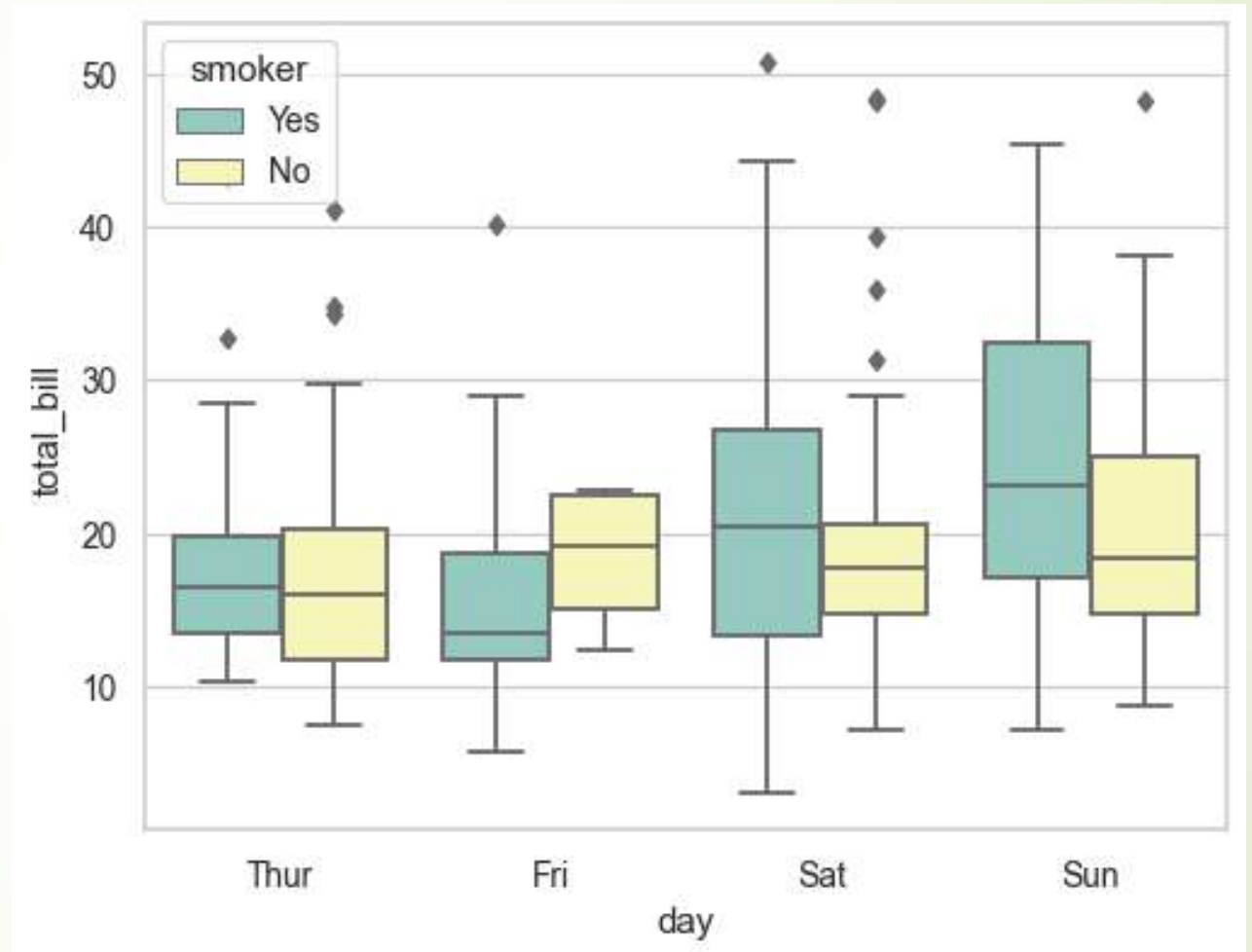
Box plot

- vertical boxplot grouped by a categorical variable
- `ax = sns.boxplot(x="day", y="total_bill", data=tips)`



Boxplot

- ▶ boxplot with nested grouping
by two categorical variables
- ▶ `ax = sns.boxplot(x="day",
y="total_bill", hue="smoker", ...
data=tips, palette="Set3")`



Scatter plot

- Metric to identify outliers: **standard deviation**
- If a value has a distance to the average higher than $x * \text{standard deviation}$, it can be assumed as an outlier. Then what x should be?
- Usually, a value between 2 and 4 seems practical.

```
#Dropping the outlier rows with standard deviation
factor = 3
upper_lim = data['column'].mean () + data['column'].std () * factor
lower_lim = data['column'].mean () - data['column'].std () * factor

data = data[(data['column'] < upper_lim) & (data['column'] >
lower_lim)]
```

Z- score

- Wikipedia definition:

- The Z-score is the signed number of standard deviations by which the value of an observation or data point is above the mean value of what is being observed or measured.*

- Re-scale and center the data**

- Outliers:** data points which are too far from zero.

- In most of the cases a threshold of 3 or -3 is used

```
from scipy import stats
import numpy as np

z = np.abs(stats.zscore(boston_df))
print(z)
```

```
[[0.41771335 0.28482986 1.2879095 ... 1.45900038 0.44105193 1.0755623 ]
 [0.41526932 0.48772236 0.59338101 ... 0.30309415 0.44105193 0.49243937]
 [0.41527165 0.48772236 0.59338101 ... 0.30309415 0.39642699 1.2087274 ]
 ...
 [0.41137448 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.98304761]
 [0.40568883 0.48772236 0.11573841 ... 1.17646583 0.4032249 0.86530163]
 [0.41292893 0.48772236 0.11573841 ... 1.17646583 0.44105193 0.66905833]]
```

Z-score

- Looking the code and the output in the previous slide, it is difficult to say which data point is an outlier. Let's try and define a threshold to identify an outlier.

```
threshold = 3  
print(np.where(z > 3))
```

```
(array([ 55,  56,  57, 102, 141, 142, 152, 154, 155, 160, 162, 163, 199,  
       200, 201, 202, 203, 204, 208, 209, 210, 211, 212, 216, 218, 219,  
       220, 221, 222, 225, 234, 236, 256, 257, 262, 269, 273, 274, 276,  
       277, 282, 283, 283, 284, 347, 351, 352, 353, 353, 354, 355, 356,  
       357, 358, 363, 364, 364, 365, 367, 369, 370, 372, 373, 374, 374,  
       380, 398, 404, 405, 406, 410, 410, 411, 412, 412, 414, 414, 415,  
       416, 418, 418, 419, 423, 424, 425, 426, 427, 427, 429, 431, 436,  
       437, 438, 445, 450, 454, 455, 456, 457, 466], dtype=int64), array([ 1,  1,  1, 11, 12,  3,  3,  
       3,  3,  3,  3,  1,  1,  1,  1,  1,  
       1,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  5,  3,  3,  1,  5,  
       5,  3,  3,  3,  3,  3,  3,  3,  1,  3,  1,  1,  7,  7,  1,  7,  7,  7,  
       3,  3,  3,  3,  3,  5,  5,  5,  3,  3,  3, 12,  5, 12,  0,  0,  0,  
       0,  5,  0, 11, 11, 11, 12,  0, 12, 11, 11,  0, 11, 11, 11, 11, 11,  
       11,  0, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11],  
      dtype=int64))
```

List of row indices

List of column indices

Data points where Z-scores is greater than 3



IQR score





IQR score

good approach

- Wikipedia definition:
 - The interquartile range (IQR), also called the **midspread** or **middle 50%**, or technically **H-spread**, is a measure of statistical dispersion, being equal to **the difference between 75th and 25th percentiles**, or **between upper and lower quartiles**; **$IQR = Q3 - Q1$** .
- Remark: It is a measure of the dispersion similar to standard deviation or variance but is much **more robust against outliers.**

IQR score

- Let's find out we can box plot uses IQR and how we can use it to find the list of outliers as we did using Z-score calculation. First we will calculate IQR

```
Q1 = boston_df_o1.quantile(0.25)
Q3 = boston_df_o1.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
CRIM      3.565378
ZN        12.500000
INDUS     12.910000
CHAS      0.000000
NOX       0.175000
RM        0.738000
AGE       49.050000
DIS       3.088250
RAD       20.000000
TAX      387.000000
PTRATIO   2.800000
B         20.847500
LSTAT     10.005000
dtype: float64
```

IQR for each column

IQR score

- The data point where we have False means these values are valid whereas True indicates presence of an outlier.

```
print(boston_df_o1 < (Q1 - 1.5 * IQR)) |(boston_df_o1 > (Q3 + 1.5 * IQR))
```

IQR score

- The data point where we have False means these values are valid whereas True indicates presence of an outlier.

4	False	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	False	False	False	False	False	False
11	False	False	False	False	False	False	False	False	False	False	False	False	False
12	False	False	False	False	False	False	False	False	False	False	False	False	False
13	False	False	False	False	False	False	False	False	False	False	False	False	False
14	False	False	False	False	False	False	False	False	False	False	False	False	False
15	False	False	False	False	False	False	False	False	False	False	False	False	False
16	False	False	False	False	False	False	False	False	False	False	False	False	False
17	False	False	False	False	False	False	False	False	False	False	False	False	False
18	False	False	False	False	False	False	False	False	False	False	False	<u>True</u>	False
19	False	False	False	False	False	False	False	False	False	False	False	False	False
20	False	False	False	False	False	False	False	False	False	False	False	False	False
21	False	False	False	False	False	False	False	False	False	False	False	False	False

Detecting outlier with IQR

Percentile

- ▶ Another mathematical method to detect outliers is to use percentiles.
 - ▶ A **percentile** (or a **centile**) **is a measure** used in statistics indicating the value **below which a given percentage of observations** in a group of observations falls.
- ▶ You can assume a certain percent of the value from the top or the bottom as an outlier.
- ▶ A common mistake is using the percentiles according to the range of the data, e.g., if your data ranges from 0 to 100, **your top 5% is not the values between 96 and 100. Top 5% means the values that are out of the 95th percentile of data.**

An Outlier Dilemma: Drop or Cap

➤ Correcting

#Capping the outlier rows with Percentiles

```
upper_lim = data['column'].quantile(.95)
```

```
lower_lim = data['column'].quantile(.05)
```

```
data.loc[(df[column] > upper_lim),column] = upper_lim
```

```
data.loc[(df[column] < lower_lim),column] = lower_lim
```

➤ Removing

➤ Z-score:

```
boston_df_o = boston_df_o[(z < 3).all(axis=1)]
```

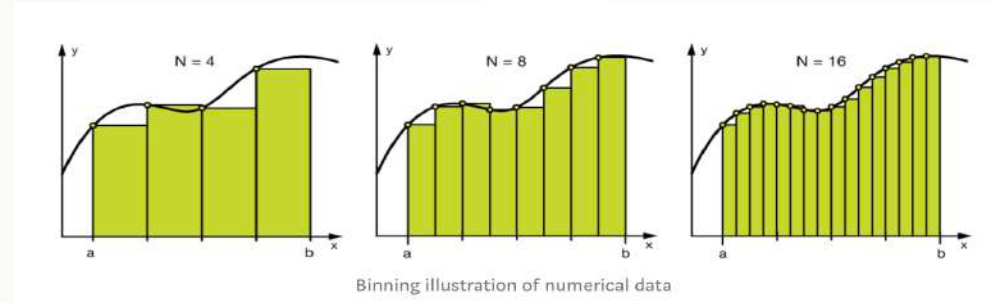
➤ IQR score:

```
boston_df_out = boston_df_o1[~((boston_df_o1 < (Q1 - 1.5 * IQR)) |  
(boston_df_o1 > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
boston_df_out.shape
```

Binning using bin

- Binning can be applied on both categorical and numerical data:



- Example:

#Numerical Binning Example

Value	Bin
0-30	-> Low
31-70	-> Mid
71-100	-> High

#Categorical Binning Example

Value	Bin
Spain	-> Europe
Italy	-> Europe
Chile	-> South America
Brazil	-> South America



Binning

Through bins, we highlight more relevant features.

- The main motivation of binning is to make the model more **robust** and prevent overfitting, however, it has a cost to the performance.
- The **trade-off between performance and overfitting** is the key point of the binning process.
- **Numerical binning:** binning might be redundant due to its effect on model performance.
- **Categorical binning:** the labels with low frequencies probably affect the robustness of statistical models negatively. Thus, assigning a general category to these less frequent values helps to keep the robustness of the model.

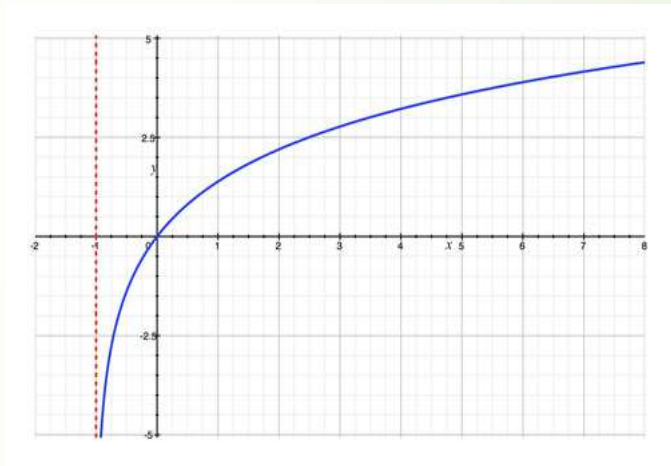
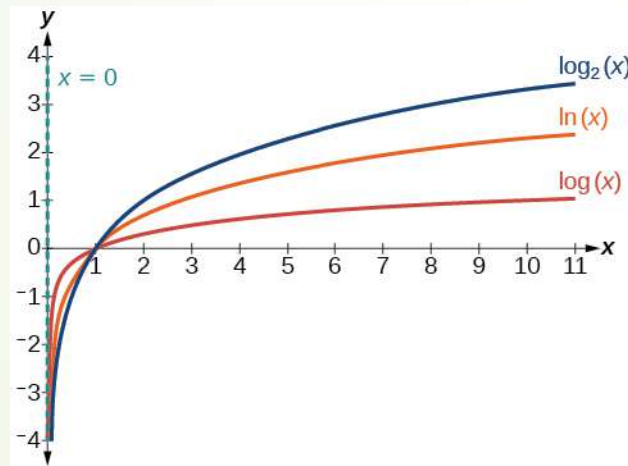


Log transformation

- It helps to handle skewed data and after transformation, **the distribution becomes closer approximate to normal.**
- In most of the cases the **magnitude order of the data changes within the range of the data.** It also decreases the effect of the outliers due to the normalization of magnitude differences and the model become more robust.

Log transformation

- **The data** you apply log transform must have only positive values, otherwise you receive an error. Also, you **can add 1** to your data before transform it. Thus, you **ensure the output of the transformation to be positive**



Log transformation

- **The data** you apply log transform **must have only positive values**, otherwise you receive an error. Also, you **can add 1** to your data before transform it. Thus, you **ensure the output of the transformation to be positive**

```
#Log Transform Example
data = pd.DataFrame({'value':[2,45, -23, 85, 28, 2, 35, -12]})
```

```
data['log+1'] = (data['value']+1).transform(np.log)
```

```
#Negative Values Handling
```

```
#Note that the values are different
```

```
data['log'] = (data['value']-data['value'].min()+1)
               .transform(np.log)
```

	value	log(x+1)	log(x-min(x)+1)
0	2	1.09861	3.25810
1	45	3.82864	4.23411
2	-23	nan	0.00000
3	85	4.45435	4.69135
4	28	3.36730	3.95124
5	2	1.09861	3.25810
6	35	3.58352	4.07754
7	-12	nan	2.48491

to handle neg value



Grouping

- Tidy data where each row represents an instance and each column represent a feature.

Tidy datasets are easy to manipulate, model and visualise, and have a specific structure: each variable is a column, each observation is a row, and each type of observational unit is a table.

— Hadley Wickham

- Datasets such as transactions rarely fit the definition of tidy data -> we use grouping.
- The key point of group by operations is to **decide the aggregation functions of the features.**

Grouping

- Aggregating categorical columns:
 - Highest frequency: the **max** operation for categorical columns

```
data.groupby('id').agg(lambda x: x.value_counts().index[0])
```

- Make a **Pivot table**: This would be a good option if you aim to go beyond binary flag columns and merge multiple features into aggregated features, which are more informative.

User	City	Visit Days
1	Roma	1
2	Madrid	2
1	Madrid	1
3	Istanbul	1
2	Istanbul	4
1	Istanbul	3
1	Roma	3



User	Istanbul	Madrid	Roma
1	3	1	4
2	4	2	0
3	1	0	0

Pivot table example: Sum of Visit Days grouped by Users

- Apply **one-hot encoding**



Grouping

- ▶ Numerical columns are mostly grouped using:
 - ▶ Sum
 - ▶ Mean

```
#sum_cols: List of columns to sum
#mean_cols: List of columns to average

grouped = data.groupby('column_to_group')

sums = grouped[sum_cols].sum().add_suffix('_sum')
avgs = grouped[mean_cols].mean().add_suffix('_avg')

new_df = pd.concat([sums, avgs], axis=1)
```

Splitting

- Most of the time the dataset contains string columns that violates tidy data principles.
- **Split function** is a good option, however, there is no one way of splitting features

```
data.name
0 Luther N. Gonzalez
1 Charles M. Young
2 Terry Lawson
3 Kristen White
4 Thomas Logsdon

#Extracting first names
data.name.str.split(" ").map(lambda x: x[0])
0 Luther
1 Charles
2 Terry
3 Kristen
4 Thomas

#Extracting last names
data.name.str.split(" ").map(lambda x: x[-1])
0 Gonzalez
1 Young
2 Lawson
3 White
4 Logsdon
```




Scaling

- In most cases, the numerical features of the dataset **do not have a certain range** and they differ from each other.
- E.g.: In real life, it is nonsense to expect **age** and **income** columns to have the same range. But from the machine learning point of view, how these two columns can be compared?
- It is important for algorithms that work based on distance: such as **kNN** or **k-Means**.
- Basically, there are two common ways of scaling: Normalization, and Standardization.

Normalization

- Normalization (or min-max normalization) **scale all values in a fixed range between 0 and 1.**

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- This transformation **does not change the distribution of the feature.**
- But due to the **decreased standard deviations, the effects of the outliers increases.** So before normalization, **it is recommended to handle the outliers.**

Normalization

➤ Example:

```
data = pd.DataFrame({'value': [2, 45, -23, 85, 28, 2, 35, -12]})  
  
data['normalized'] = (data['value'] - data['value'].min()) /  
                    (data['value'].max() - data['value'].min())
```

	value	normalized
0	2	0.23
1	45	0.63
2	-23	0.00
3	85	1.00
4	28	0.47
5	2	0.23
6	35	0.54
7	-12	0.10

Standardization

- Standardization (or **z-score normalization**) scales the values while **taking into account standard deviation**.
- In the following formula of standardization, the **mean** is shown as μ and the **standard deviation** is shown as σ

$$z = \frac{x - \mu}{\sigma}$$

- If the standard deviation of features is different, their range also would differ from each other. **This reduces the effect of the outliers in the features.**

Standardization

➤ Example:

```
data = pd.DataFrame({'value': [2, 45, -23, 85, 28, 2, 35, -12]})  
data['standardized'] = (data['value'] - data['value'].mean()) /  
data['value'].std()
```

	value	standardized
0	2	-0.52
1	45	0.70
2	-23	-1.23
3	85	1.84
4	28	0.22
5	2	-0.52
6	35	0.42
7	-12	-0.92



Feature encoding

- Turn **categorical features into numeric features** to provide more fine-grained information
- Help explicitly **capture non-linear relationships** and interactions between the values of features
- **Most of machine learning tools only accept numbers** as their input e.g., xgboost, gbm, glmnet, libsvm, liblinear, etc.

Feature encoding

use only on cardinal numbers


➤ Labeled encoding

- Interpret the categories as ordered integers (can be wrong)
- Python Scikit-Learn: LabelEncoder
- Effective in tree-based methods

• Labeled Encoding

A	0
B	1
C	2

Feature 1	Encoded Feature 1
A	0
A	0
A	0
A	0
B	1
B	1
B	1
C	2
C	2



Feature encoding

► **Labeled encoding**

- Interpret the categories as ordered integers (can be wrong)
- Python Scikit-Learn: `LabelEncoder`
- Effective in tree-based methods

► **One Hot Encoding**

- Transform categories into individual binary (0 or 1) features
- Python Scikit-Learn: `DictVectorizer`, `OneHotEncoder`
- Effective for K-means, Linear, NNs, etc.

One-hot encoding

- **One of the most common encoding methods** in machine learning.
- This method spreads the values in a column to multiple flag columns and assigns 0 or 1 to them. These binary values express the relationship between grouped and encoded column.

User	City
1	Roma
2	Madrid
1	Madrid
3	Istanbul
2	Istanbul
1	Istanbul
1	Roma

→

User	Istanbul	Madrid
1	0	0
2	0	1
1	0	1
3	1	0
2	1	0
1	1	0
1	0	0

One hot encoding example on City column

```
encoded_columns = pd.get_dummies(data['column'])  
data = data.join(encoded_columns).drop('column', axis=1)
```

Frequency encoding

- Encoding of categorical levels of feature to **values between 0 and 1 based on their relative frequency**

A	0.44 (4 out of 9)
B	0.33 (3 out of 9)
C	0.22 (2 out of 9)

Feature	Encoded Feature
A	0.44
A	0.44
A	0.44
A	0.44
B	0.33
B	0.33
B	0.33
C	0.22
C	0.22

Target mean encoding

- Instead of **dummy encoding** of categorical variables and **increasing the number of features** we can encode each level as the mean of the response.

A	0.75 (3 out of 4)
B	0.66 (2 out of 3)
C	1.00 (2 out of 2)

Feature	Outcome	MeanEncode
A	1	0.75
A	0	0.75
A	1	0.75
A	1	0.75
B	1	0.66
B	1	0.66
B	0	0.66
C	1	1.00
C	1	1.00

Target mean encoding

- Calculate **weighted average of the overall mean** of the training set and the mean of the level:

$$\lambda(n) * \text{mean}(\text{level}) + (1 - \lambda(n)) * \text{mean}(\text{dataset})$$

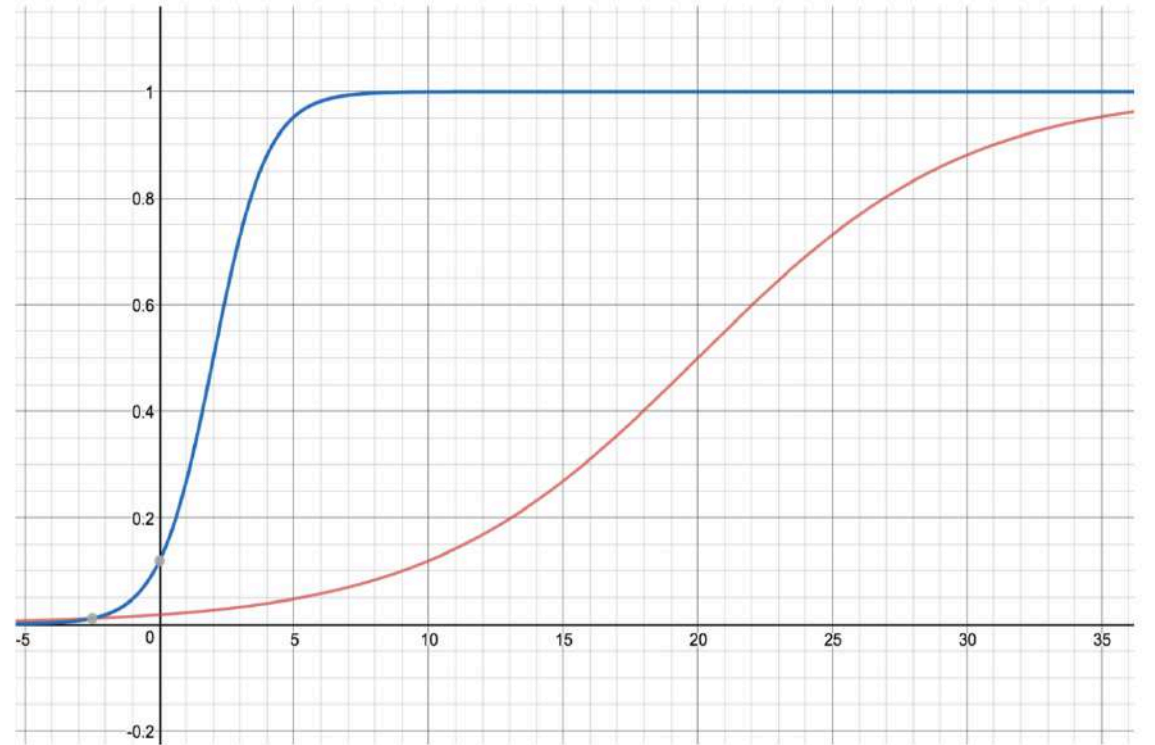
- **The weights are based on the frequency of the levels** i.e. if a category only appears a **few times** in the dataset then its encoded value will be close to the **overall mean** instead of the mean of that level.

Target mean encoding

Smoothing

$$\frac{1}{1 + \exp\left(\frac{-(x-k)}{f}\right)}$$

x = frequency
k = inflection
point
f = steepness



Target mean encoding

Smoothing

$$\lambda = \frac{1}{1 + \exp\left(-\frac{(x-2)}{0.25}\right)}$$

	x	level	dataset	λ	
A	4	0.75	0.77	0.99	$0.99*0.75 + 0.01*0.77 = 0.7502$
B	3	0.66	0.77	0.98	$0.98*0.66 + 0.02*0.77 = 0.6622$
C	2	1.00	0.77	0.5	$0.5*1.0 + 0.5*0.77 = 0.885$

$$\lambda = \frac{1}{1 + \exp\left(-\frac{(x-3)}{0.25}\right)}$$

	x	level	dataset	λ	
A	4	0.75	0.77	0.98	$0.98*0.75 + 0.01*0.77 = 0.7427$
B	3	0.66	0.77	0.5	$0.5*0.66 + 0.5*0.77 = 0.715$
C	2	1.00	0.77	0.017	$0.017*1.0 + 0.983*0.77 = 0.773$

Feature	Outcome
A	1
A	0
A	1
A	1
B	1
B	1
B	0
C	1
C	1

Target mean encoding

► Leave-one-out (LOO) schema

Feature	Outcome
A	1
A	0
A	1
A	1
B	1
B	1
B	0
C	1
C	1

LOOencode
0.66



Target mean encoding

Leave-one-out (LOO) schema

Feature	Outcome		LOOencode
A	1		0.66
A	0		1.00
A	1		
A	1		
B	1		
B	1		
B	0		
C	1		
C	1		

Target mean encoding

► Leave-one-out (LOO) schema

Feature	Outcome
A	1
A	0
A	1
A	1
B	1
B	1
B	0
C	1
C	1

LOOencode
0.66
1.00
0.66

Target mean encoding

► Leave-one-out (LOO) schema

Feature	Outcome
A	1
A	0
A	1
A	1
B	1
B	1
B	0
C	1
C	1

LOOencode
0.66
1.00
0.66
0.66

Target mean encoding

Leave-one-out (LOO) schema

Feature	Outcome
A	1
A	0
A	1
A	1
B	1
B	1
B	0
C	1
C	1

LOOencode
0.66
1.00
0.66
0.66
0.50
0.50
1.00
1.00
1.00

Weight of Evidence and Information Value

- Weight of evidence:

$$WoE = \ln\left(\frac{\% \text{ non-events}}{\% \text{ events}}\right)$$

- To avoid division by zero:

$$WoE_{adj} = \ln\left(\frac{\text{Number of non-events in a group} + 0.5 / \text{Number of non-events}}{\text{Number of events in a group} + 0.5 / \text{Number of events}}\right)$$

- Information Value:

$$IV = \sum (\% \text{ non-events} - \% \text{ events}) * WoE$$

Weight of Evidence and Information Value

► IV = 0.221

	Non-events	Events	% of non-events	% of events	WoE	IV
A	1	3	50	42	$\ln\left(\frac{(1+0.5)/2}{(3+0.5)/7}\right) = 0.4$	$(0.5 - 0.42) * 0.4 = 0.032$
B	1	2	50	29	$\ln\left(\frac{(1+0.5)/2}{(2+0.5)/7}\right) = 0.74$	$(0.5 - 0.29) * 0.4 = 0.084$
C	0	2	0	29	$\ln\left(\frac{(0+0.5)/2}{(2+0.5)/7}\right) = -0.35$	$(0 - 0.29) * -0.35 = 0.105$

Feature	Outcome	WoE
A	1	0.4
A	0	0.4
A	1	0.4
A	1	0.4
B	1	0.74
B	1	0.74
B	0	0.74
C	1	-0.35
C	1	-0.35

Weight of Evidence and Information Value

Information Value	Variable Predictiveness
Less than 0.02	Not useful for prediction
0.02 to 0.1	Weak predictive Power
0.1 to 0.3	Medium predictive Power
0.3 to 0.5	Strong predictive Power
>0.5	Suspicious Predictive Power



Concluding remarks

More of feature engineering:

- Already seen:

- **Feature Extraction:**

- **Numerical data:** Dimensionality reduction techniques – SVD and PCA Clustering and using cluster IDs or/and distances to cluster centers as new features

- Soon to be seen:

- Feature selection

- Future scope:

- Feature extraction: text, time series, image, geographic location, relational data

- Anomaly detection



Resources

- [Fundamental techniques](#)
 - [Detect and remove outliers](#)
 - [SlideShare lecture](#)
- 