Machine Learning

Lecture 02

Data Pre-processing

Why data pre-processing?

From: Data Science Dojo #WeeklyJokes

Why data pre-processing?

- Data pre-processing (data munging / data wrangling): Getting the data ready for use in analytical models
- Why do we need to clean and prepare data?

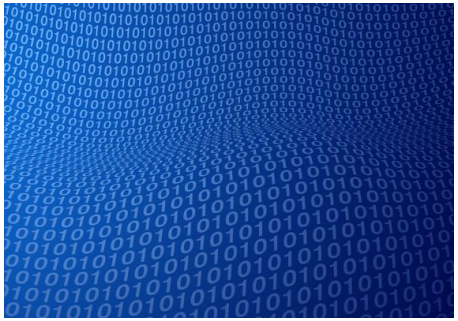
  In real world applications, data can be (and usually are) inconsistent, incomplete, and noisy due to:
  - Data entry, transmission or collection problems
  - Discrepancy in naming conventions
  - Duplicate records
  - Contradictions in data
  - Missing values
  - and many more . . .

How important is it?

- Data pre-processing is a major part of any Data Analytics project
  - Typically, more than 70-80% of data analytics projects are spent on getting the data ready for analysis

Data workflow

- Working on data to create a model input



Raw data

Data cleaning and preparation

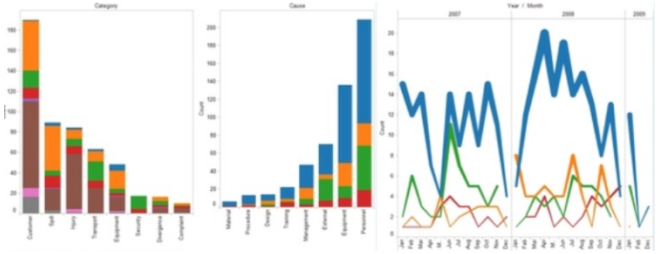
Feature engineering

Model building

Prelude to Data Analysis

1. **Experimental setup**
2. **Integrating data**
3. **Cleaning and exploring the data**
   - Data cleaning

   - Exploratory Data Analysis (EDA)

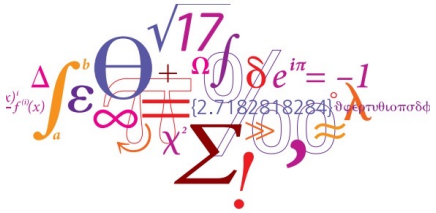


Prelude to Data Analysis – cont'd

4. Preparing the data

- Missing value treatment
- Outlier detection and treatment
- Smoothing out noisy data
- Variable transformation
- Data reduction
  - Sampling, Dimension reduction, Feature selection
- Dealing with imbalanced datasets

5. Feature Engineering

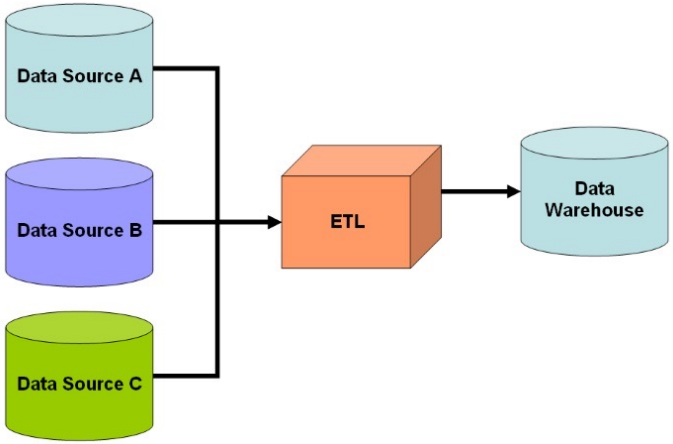


Extracting best features for more accurate predictions

- **Experimental setup for collecting data**
  - Ask the right questions and create a use-case
  - Know what you are measuring
  - Create your sample (watch out for hidden bias)
  - Polling? Carefully design your questionnaire (right & relevant questions)
  - Collect data (interview, online surveys, social media, sensors, clicks, etc.)
  - Aggregating/overlaying data becomes critical if data is coming from multiple external sources as well

Data Integration

- Combining data from multiple sources into a coherent store

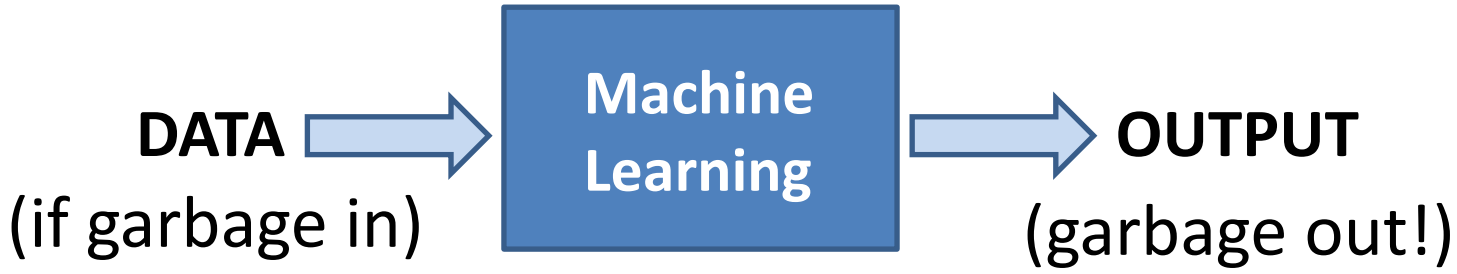


  - Schema integration: cust_id, CUSTID and Cust# ?
  - Identifying the variable: William Smith = Bill Smith ?
  - Detecting and resolving data conflicts before merging
    - Metric vs British units: kg vs lbs, km vs mile, etc.
    - Conflicts in the format: 13.07.1962 vs 07/13/62
    - How the data is stored:

Name	First name	Last name
Dale, Bob	Mary	Brown
Smith, John	Jerry	Hammon

Data Cleaning

- Data quality: Why is it so important?
  - Missing/incomplete/inconsistent/noisy data
- For the problem we're trying to solve, the data used has to be **accurate, consistent, up-to-date,** and most importatnly **relevant** to the problem at hand.
- Data quality/integrity is, and will always be a critical part of data management. No matter what technologies are in play, if the data is bad, then the information coming out cannot be trusted.



What could go wrong?

- Potential causes for low data quality (unknown, unrecorded or irrelevant entries):
  - Human error (typographical)
  - Inapplicable cases
  - Something changed in the middle of data collection
  - Measurement error due to malfunctioning equipment
  - Changes in experimental design
  - Collation of different datasets
  - Duplicate values
  - Metadata problems (multiple definitions, changing definitions over time)
  - Text-field formatting – a single ", or { sign left unclosed
  - Physically impossible data (like a heart rate of 900 bpm)
  - If the data has a "comments" column, be prepared for disasters!
  - Same entry with different names: Facebook, FB, Face book, …

Before you go any further:

- Check
  - the **presence** of data (missing/incomplete values?)
  - the **type** of data (Nominal, ordinal, interval, ratio, etc? Text field has a text? Number field has a number?, “o” for “0” or vice versa very common)
  - the **length** of data (phone #, zipcode, especially text fields could be way too long)
  - the **range** of data (e.g. any age larger than 120, or a month larger than 12?)
  - the **format** of data (date and time formats are the most confusing ones, email, zipcode etc.)
    - 01-01-2016, 01/01/16 (Month-Day or Day-Month?), 11:40:00 (AM? PM?) , does email look like an email?

Data Screening

- Before you do any analytical work, do a preliminary study on your data
- How to screen data?
  - Inspect raw data
  - Summary statistics
    - Mean, median, mode, max, min, range, variance (standard deviation) etc.
  - Visualize
    - Visualize what?
      - Examples across all features (rarely)
      - Features across all examples (a lot more common)

Exploratory Data Analysis (EDA)

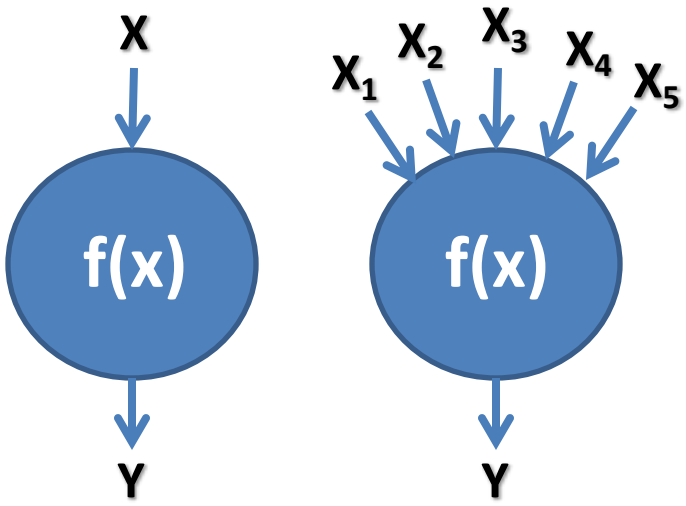
Exploratory Data Analysis (EDA)

- Why EDA?
  - Understand the behavior of your numbers
  - Detect errors early in the analysis
  - Find violations of statistical assumptions and assess assumptions for confirmatory analysis
  - What does the distribution look like? Symmetric, too tall and narrow, too short and wide spread, right- or left-skewed etc? Is the normality assumption violated? These are important as most of our analyses will assume a reference distribution to infer conclusions about the population parameters
  - Anomalous patterns? **Outliers**?

EDA – cont'd

- EDA provides hints on relations among the variables and might reveal patterns in the data, thus helping us generate hypotheses
- EDA serves as a sanity check before we dive into the mechanics of Machine Learning

- Machine Learning process:



List of names used in the literature:

$\mathbf{X_i}$: input, covariate, explanatory variable, independent variable, predictor, feature, attribute

Y : output, dependent variable, target variable, response variable

Missing values

- Question: Why the missing value?
  - Understanding the potential cause for missing data is an essential step before trying to fix it
- There could be several reasons for missing data among which are:
  - Problems in the data collection process
  - Problems in the data extraction process
- Why should we treat missing values?
  - Most models cannot handle missing values in data
  - Potential bias: If examples with missing attributes differ in some way from completely observed examples, then our analysis may yield biased results leading to inaccurate predictions.
  - Loss of power: Throwing away the samples with missing values reduces the precision of our results accordingly.

Missing values

- A missing value could be:
  - **MCAR** (**M**issing **C**ompletely **A**t **R**andom)
    - The fact that a certain value is missing has nothing to do with its hypothetical value and with the values of other variables (completely unsystematic)
  - **MAR** (**M**issing **A**t **R**andom)
    - Propensity for a data point to be missing isn't related to the missing data, but is related to some of the observed data

Gender	Age
Female	?

Female respondents choose not to reveal their ages?

  - **MNAR** (**M**issing **N**ot **A**t **R**andom)
    - Missing value could depend on the hypothetical value

…	Salary
…	?

Respondents whose salaries below 1500 TL may choose not to provide information

Diagnose missing value mechanism

- Is it MCAR?
- **Little's test for MCAR** (Little 1988): Maximum likelihood chi-square test for missing completely at random.

  H_0: The data is MCAR

  H_A: The data is not MCAR
- If the p value for Little's MCAR test is not significant, then the data may be assumed to be MCAR and missingness is ignorable.
- But it's not definitive and it provides only one piece of information.
- No official implementation of this test on Python (see R)
- Are there other tests we can use?

Diagnose missing value mechanism

- **If non-missing variable is numerical:**
  - **t-test comparisons**: Series of independent t-tests to compare missing data subgroups. Create dummy variables whether a variable is missing. Then we will know it's MCAR if there's not a statistically significant difference between sample means for C_1 corresponding to binary missingness in C_{2dummy} (repeating for all features).

Does the missingness in C_2 depend on the values of C_1?

2-sample t-test between:
Sample-1 [C_1 where $C_{2dummy} = 1$]

Sample-2 [C_1 where $C_{2dummy} = 0$]

If not significantly different => MCAR

C_1 (temp)	C_2 (noise)	C_{2dummy}
35	NaN	1
19	89	0
36	NaN	1
22	100	0
…	…	0
18	85	0
38	NaN	1

Diagnose missing value mechanism

- **If non-missing variable is categorical:**
  - Once you separate the categorical variables into "complete" and "incomplete" cases, we can analyze the association between each variable's complete and incomplete cases using the Chi-square independence test and the effect size from the Cramer's V.

Does the missingness in C_2 depend on the values in C_1?

χ^2 test between categorical variables

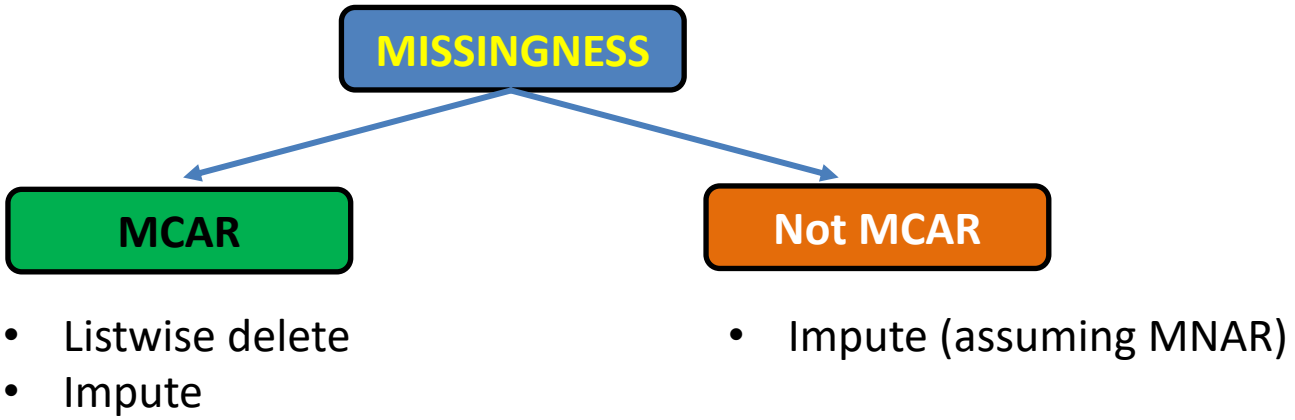
C_{2dummy} \ C_1	Cool	Warm	Hot
0	34	51	43
1	52	77	63

If association not significant => MCAR

C_1 (temp)	C_2 (noise)	C_{2dummy}
Hot	NaN	1
Cool	89	0
Hot	NaN	1
Warm	100	0
…	…	0
Cool	85	0
Hot	NaN	1

Diagnose missing value mechanism

- What if it's not MCAR?
  - No testing for MAR vs MNAR. The only way to distinguish between them is to go back and measure some of that missing data if possible. If the measured quantities differ by very much, we have evidence that it's MNAR. Sometimes using domain expertise could help resolve the mystery. For all other cases we can only assume (hope) that it's MAR and go on with imputing the missing entries.



Handling missing data

- What can you do with missing data?
  - Don't impute missing values without looking closely into the potential causes (non-randomness?)
- **1. Ignore the missing values**
  - This only works if you can assume that your data is MCAR, but this is often rarely the case
  - Certain models of Machine Learning can directly handle missing values:
    - Tree based models
      - Decision trees and Random Forests
    - kNN (k-Nearest Neighbors)
      - Python implementation of kNN (in Scikit-learn), however, cannot!

Handling missing data

- **2. Remove samples/rows** (list-wise deletion)
  - May introduce substantial bias if not MCAR
  - Increases sampling error (with a less sample size) reducing the statistical power
  - Suitable when there are only a few missing entries (< 5%)

Credit	Payback	Income	Approve
Excellent	3 yrs	High	Yes
Fair	?	Low	No
Fair	3 yrs	High	Yes
Poor	5 yrs	High	No
Excellent	3 yrs	Low	No
Fair	5 yrs	High	Yes
Poor	3 yrs	High	No
Poor	?	Low	Yes
Fair	5 yrs	High	Yes

Credit	Payback	Income	Approve
Excellent	3 yrs	High	Yes
Fair	3 yrs	High	Yes
Poor	5 yrs	High	No
Excellent	3 yrs	Low	No
Fair	5 yrs	High	Yes
Poor	3 yrs	High	No
Fair	5 yrs	High	Yes

Handling missing data – cont'd

- **3. Remove features** (columns)
  - When there are too many missing entries (say more than 60%)

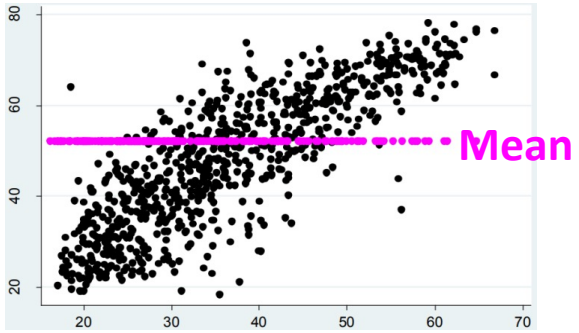
Credit	Payback	Income	Approve
Excellent	3 yrs	High	Yes
Fair	?	Low	No
Fair	3 yrs	High	Yes
Poor	?	High	No
Excellent	?	Low	No
Fair	?	High	Yes
Poor	3 yrs	High	No
Poor	?	Low	Yes
Fair	?	High	Yes

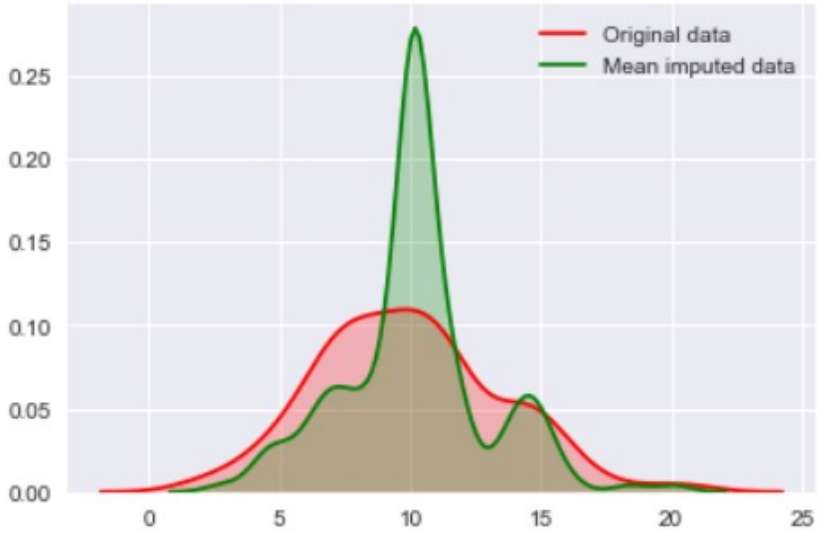
Credit	Payback	Income	Approve
Excellent	3 yrs	High	Yes
Fair	3 yrs	High	Yes
Poor	3 yrs	High	No

Credit	Income	Approve
Excellent	High	Yes
Fair	Low	No
Fair	High	Yes
Poor	High	No
Excellent	Low	No
Fair	High	Yes
Poor	High	No
Poor	Low	Yes
Fair	High	Yes

Handling missing data – cont'd

- **4. Imputation with a single value**
  - **Mean** or **median** for numerical variables: Quick and simple, but it underestimates the variance, distorts the shape of the distribution (larger concentration around the mean), and depresses the observed correlation as all missing data will have a single constant value.





Distribution of the original data (with no missing values)
Distribution of the imputed data (random missing data is created from the original data and then imputed by the feature mean)

Handling missing data – cont'd

- **4. Imputation with a single value** (cont'd)
  - **Mean** or **median** imputation can be done in a selective manner based on the nature and values of other attributes.
  - Example:

City	Salary	Experience
A	60000	Mid
B	NaN	Entry
C	75000	Mid
A	47500	Entry
B	85000	High
…	…	…

Instead of using the overall mean value of '**Salary**' to impute the missing value, use the mean of '**Entry**' level salaries of people employed in '**City B**'.

Handling missing data – cont'd

- **3. Imputation with a single value** (cont'd)
  - A user-selected value
    - Requires domain expertise
  - Distribution-based values
    - Using the probability distribution of data
  - **Model (prediction) based values**
    - Using a predictive method
      - Linear Regression (single-value imputation)
      - Tree-based regression (single-value imputation)
      - Nearest neighbors (single-value imputation)
      - Multiple imputation (a randomness is injected into the repeated computations)

Handling missing data – cont'd

- **Prediction-based Imputation** (Linear Regression)
- Suppose we have a data file:

  We can take x_2 as the regressor (independent variable) and form a multiple regression problem by using the non-missing covariates:

x_1	x_2	x_3	x_4	y
65	21	3	30	11
77	?	7	54	14
…	…	…	…	…
99	?	11	87	21
118	89	14	101	25
μ_i :	…	**50**	…	70

$$x_2 = \beta_0 + \beta_1 x_1 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 y$$

Having computed all β_i, we can fill the missing values of x_2 by:

$$(x_2)_1 = \beta_0 + \beta_1 77 + \beta_3 7 + \beta_4 54 + \beta_5 14$$

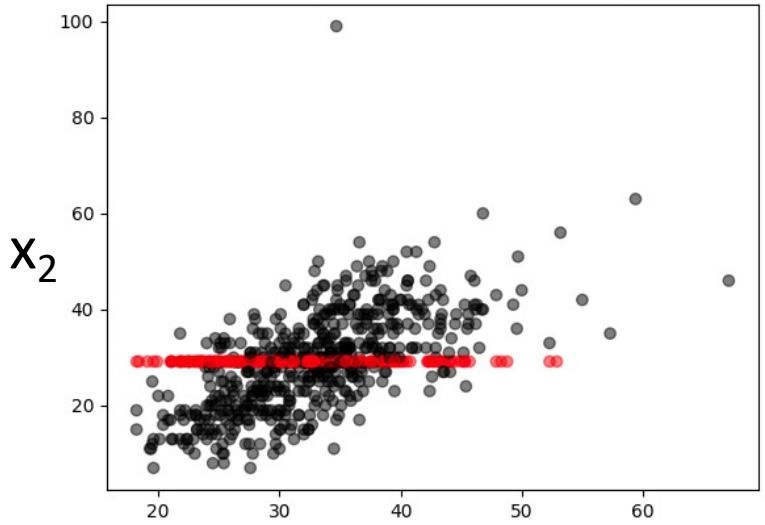
$$(x_2)_2 = \beta_0 + \beta_1 99 + \beta_3 11 + \beta_4 87 + \beta_5 21$$

Disadvantages:

- Assumes that variable with missing data is highly correlated with the other covariates.
- The predicted values may not fall in the valid ranges

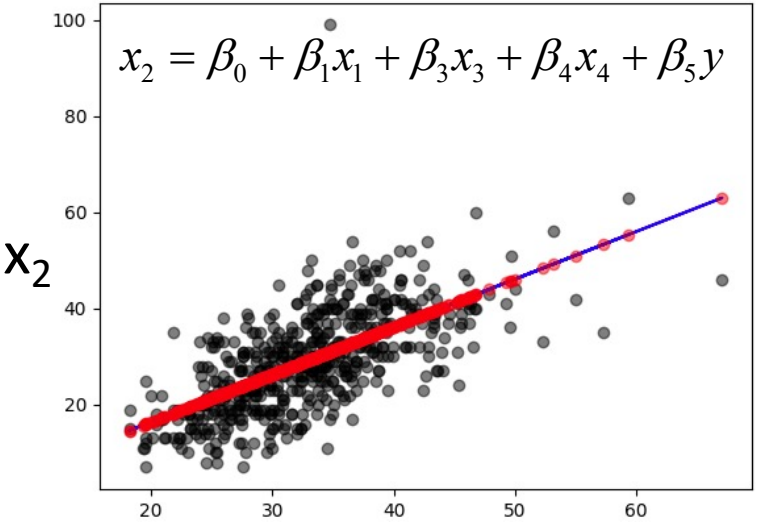
Handling missing data – cont'd

Mean imputation

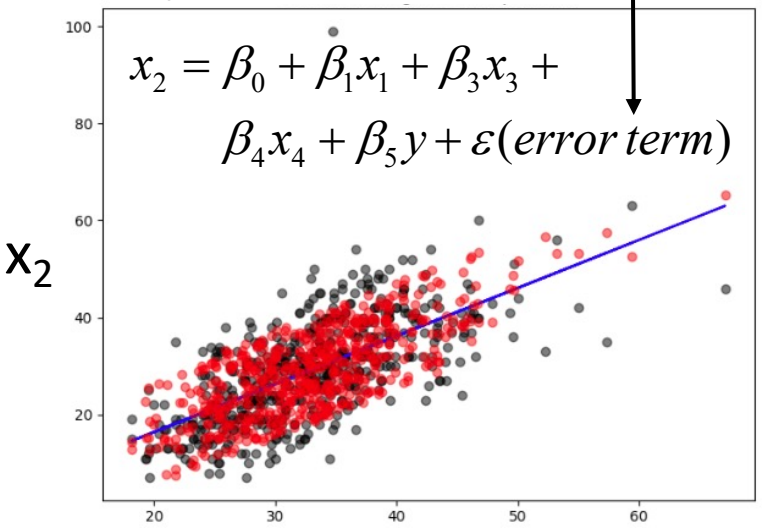


This error (residual) term is normally distributed with a mean of zero and a variance equal to the variance of the predictor used for imputing

Regression imputation



Stochastic Regression imputation



Handling missing data – cont'd

- **Prediction-based Imputation** (**k**-Nearest **N**eighbors)
- We have a data set **D** of size **N**
- Assume data in instance '**i**' has a missing value for attribute $\mathbf{C_1}$

#	C_1	C_2	C_3	C_4	Class
1	21	6.5	52	41	Yes
2	17	11	87	15	No
…	…	…	…	…	…
i	?	7	54	39	Yes
…	…	…	…	…	…
N	40	14	99	82	No

Pseudocode for the imputation

- for j = 1 , N (where j ≠ i) :
  - Compute distance $\mathbf{d_j}$ between the data instances '**i**' and '**j**'

$$d(i,j) = d_j = \sqrt{\sum_{m \in D^*} (x_i^m - x_j^m)^2}$$

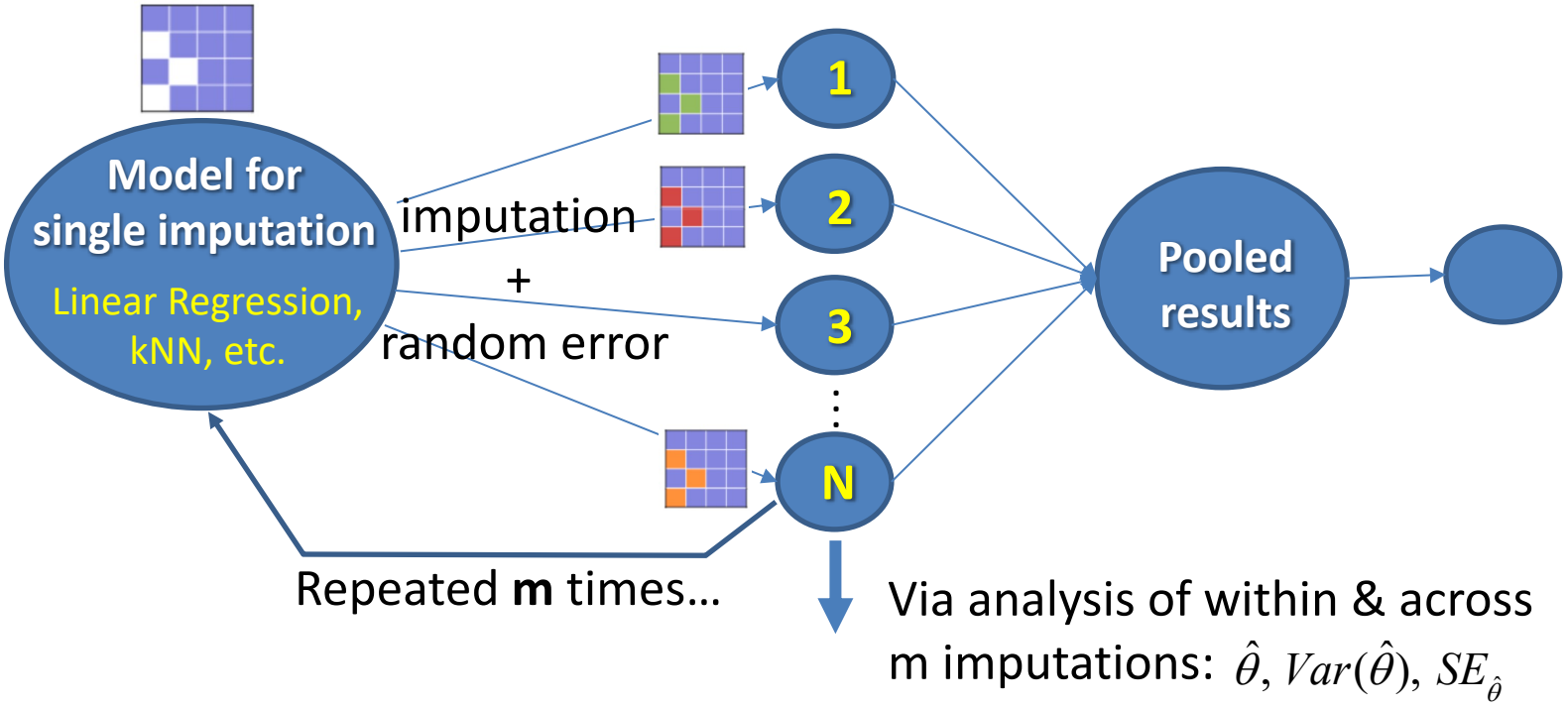
D*is the set of attributes with non-missing values

where sum is over all columns with no missing values

  - Save the distance $\mathbf{d_j}$ in a similarity array **S**
- Sort tne array **S** in descending order
- Pick the top **k** data instances from **S**
- Impute the missing value of '**i**' by the mean (or mode in case of categorical attribute) of the top **k** known values of attribute $\mathbf{C_1}$

Handling missing data – cont'd

- **Multiple Imputation**
  - Single imputation usually underestimates the SE
  - So the missing value is imputed several times (15 times if 15% of data is missing, say) for good estimates of SE
  - **MICE** (Multiple Imputation Chained Equations)



Handling missing data – cont'd

- **Multiple Imputation (procedure)**
  - Step 1: For every feature with missing entries, conduct a mean-imputation (consider these as place-holders)
  - Step 2: Take **one feature**, (say var1) and set the mean-imputed values back to missing.
  - Step 3: Regress the observed values in var1 on the other variables (var1 as the dependent, all others as the independent variables)
  - Step 4: The missing values for var1 are then replaced with predictions from the regression model. Then var1 is subsequently used as an independent variable to impute the missing entries in other features.
  - Step 5: Steps 2-4 are repeated for each variable that has missing data. Cycling through all variables constitutes one iteration
  - Step 6: Steps 2-4 are repeated for a number of iterations (say 10) until coefficients in the regression models converge in the sense of becoming stable.

Handling missing data – cont'd

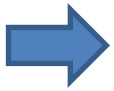
- **Notes on imputation**
  - Imputation gives a power boost to your analysis, and it works best when many variables are missing in small proportions such that a complete case analysis might render 60-30% completeness, but each variable is perhaps only missing 10% of its values. The power boost is much less impressive when one important variable is missing in 70% of cases, because the uncertainty in estimates will yield highly varying imputed datasets.
  - When performing multiple imputation, should we include all predictor variables even if only 1 or 2 variables have missing values? That depends strongly on your specific data. For data consisting of few variables, it is often a good approach to use all variables. With larger data, you should usually do a variable selection, mainly due to computational reasons and to exclude noisy predictors.

Ref: stats.stackexchange.com/questions/212161/opinion-on-when-to-impute-data

Handling missing data – cont'd

- **What if the variable is categorical?**
  - Replace with the most frequent (mode) value
  - Use a classification model to predict the missing values (Logistic Regression, kNN)
  - Using missing value as a category

Residency	Approve
Owner	Yes
NaN	No
Tenant	Yes
Tenant	No
NaN	No
Owner	No
…	…

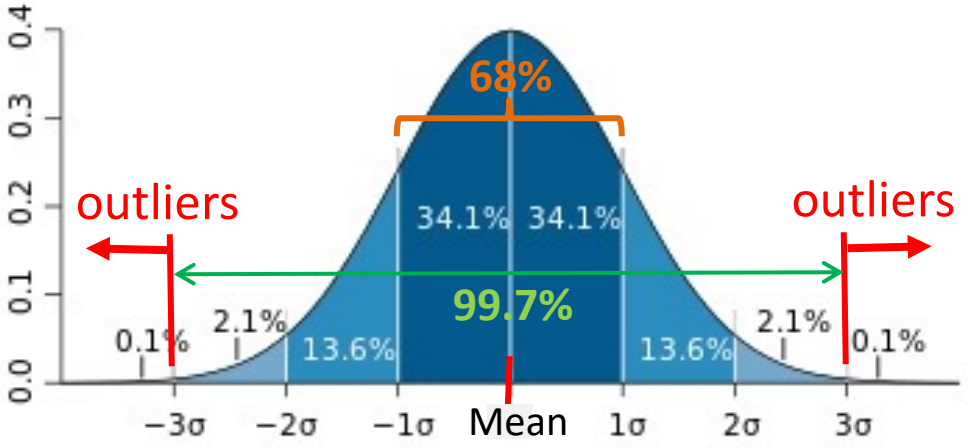


Residency	…
Owner	…
U	…
Tenant	…
Tenant	…
U	…
Owner	…
…	…

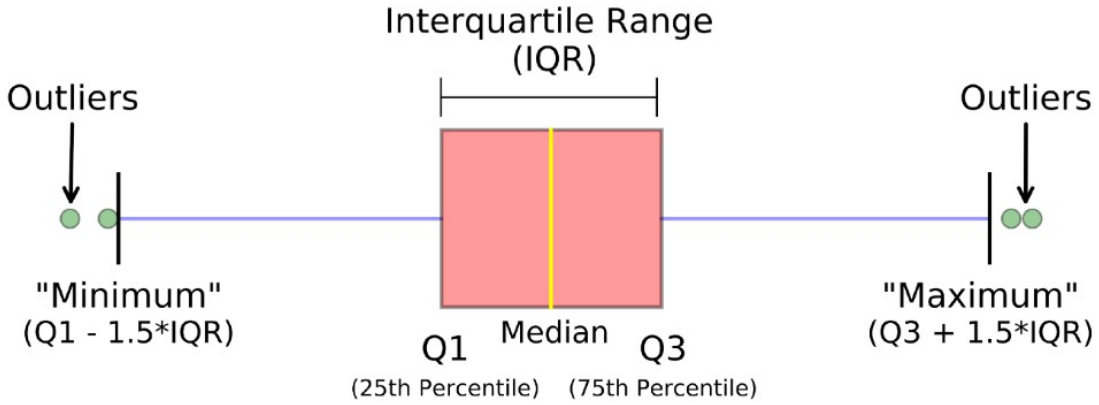
Create another level for missing values (an indicator of missingness)

Detecting outliers

- Capping: Values beyond ±**3σ** from the mean



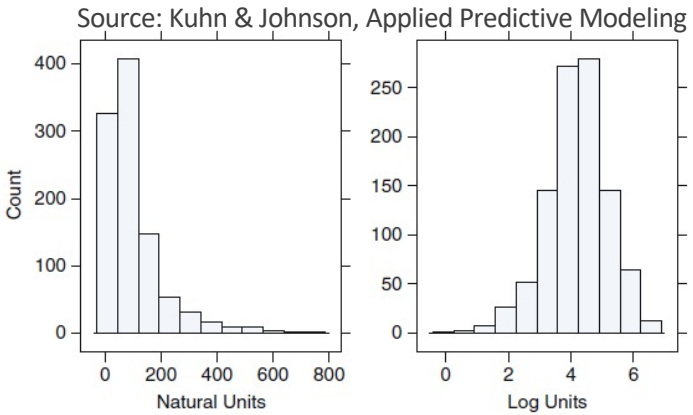
- ±1.5IQR in a boxplot:



- Many other ML algorithms used for outlier detection:
  - Isolation forest, Elliptic envelope, One class SVM, Local outlier factor, clustering techniques (DBSCAN, Gaussian Mixture Modeling), etc.

Variable transformation

- We transform features for a number of reasons:
- Removing skew: If features are normally distributed, taking the "log" of feature values may produce a normal distribution



- Scaling: No dominant effects in the presence of features with highly varying magnitudes

Min-Max Scaling: Mapping the data to a fixed range - usually 0 to 1

$$x_{norm} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Standard normal distribution: Convert to N(0,1)

$$x = \frac{x - \overline{x}}{\sigma_x}$$

- Changing numerical variables to categorical ones or vice versa (see next slides)

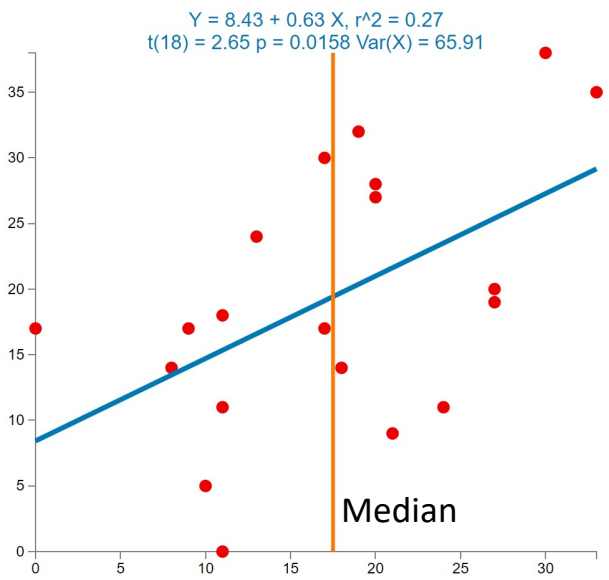
Variable transformation – cont'd

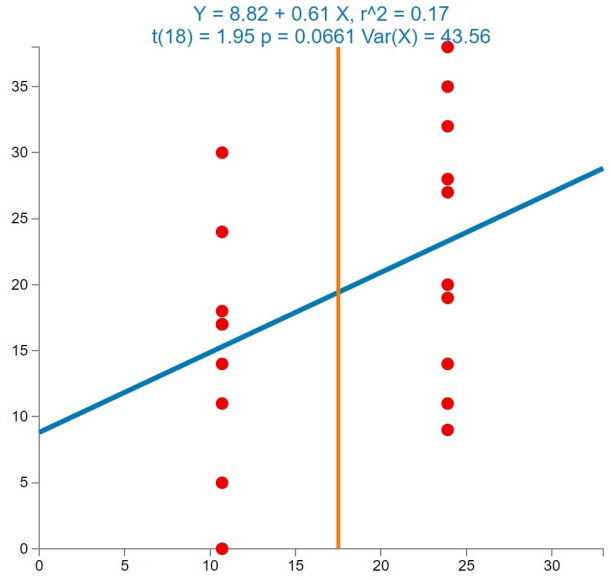
- **Decomposing attributes**
  - Changing numerical variables to categorical ones via binning (grouping):

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

small medium large

Warning: Not recommended due to loss of resolution





Ref: http://psych.colorado.edu/~mcclella/MedianSplit/

Variable transformation – cont'd

- **Decomposing attributes**
  - Decomposing a Date-Time: From the date & time data you could create features like **day_of_week** or **part_of_day** (morning, midday, afternoon, night) which could increase the predictive power of the model

Date	Time	Day	DayTime	IsHoliday?	IsWeekend?
01.01.2020	10:00	Wed	Morning	1	0
02.01.2020	13:30	Thu	Afternoon	0	0
03.01.2020	12:00	Fri	Midday	0	0
04.01.2020	21:00	Sat	Night	0	1
05.01.2020	23:00	Sun	Night	0	1
06.01.2020	07:30	Mon	Evening	0	0

  - Decomposing categorical attributes (encoding categorical variables as numerical variables – next)

Handling categorical variables

One-hot encoding / Ordinal encoding

#	Color	Size	Price
0	green	M	10.50
1	red	L	12.50
2	blue	XL	13.25

Nominal (One-hot encoding)

Green ➔ (1,0,0)
Red ➔ (0,1,0)
Blue ➔ (0,0,1)

Dummy Variables

Ordinal encoding

M ➔ 1
L ➔ 2
XL ➔ 3

#	is_green	is_red	is_blue	Size	Price
0	1	0	0	1	10.50
1	0	1	0	2	12.50
2	0	0	1	3	13.25

Handling categorical variables – cont'd

Frequency encoding (Count encoding)

#	Temperature	Temp_encoded
1	Hot	0.3
2	Warm	0.5
3	Warm	0.5
4	Warm	0.5
5	Hot	0.3
6	Cold	0.2
7	Warm	0.5
8	Cold	0.2
9	Hot	0.3
10	Warm	0.5

Every category level in the data set are represented by their respective frequencies:

$$\frac{\text{Number of instances labeled as "Hot"}}{\text{Number of total instances}}$$

If you were to use the **number of instances** labeled as "Hot", then it would be called "**Count encoding**"

Handling categorical variables – cont'd

Target (mean) encoding

#	Temp	Target	Temp encoded
1	Hot	1	1
2	Warm	0	0.6
3	Warm	1	0.6
4	Warm	1	0.6
5	Hot	1	1
6	Cold	0	0
7	Warm	0	0.6
8	Cold	0	0
9	Hot	1	1
10	Warm	1	0.6

Step 1	Sum of Target
Cold	0+0=0
Warm	0+1+1+0+1=3
Hot	1+1+1=3

Step 2	Count on Target
Cold	2
Warm	5
Hot	3

Much like label encoding, except here labels are correlated directly with the target

Step1 & Step2

Step 3	Mean
Cold	0/2
Warm	3/5
Hot	3/3

Disadvantages:

- May lead to over-fitting
- Difficult to implement with the cross validation procedure
- If 2 categories show the same mean of target, they get replaced by the same number => potential loss of value

Handling categorical variables – cont'd

- **What about categorical variables with high cardinality** (variables with many unique labels?
- Example**:** In predicting house prices, one of the variables could be the town (area) in a city. If there are 50 towns in our data set, one-hot encoding will end up with 50 separate columns, one for each town (a very sparse and overly complex data set).

#	…	town1	town2	town3	…	town49	town50	…
1	…	0	1	0	…	0	0	…
2	…	0	0	0	…	1	0	…

- One method that is worth trying would be to use a numerical value for each town. Say, total number of houses for sale, social welfare, or any other statistical information that's in line with the business objective of the problem.

Handling categorical variables – cont'd

- **What about categorical variables with cyclic nature?**
- Example**:** Hours, days and months are features of such nature.
- These can certainly be one-hot encoded, but in some cases, the model may not have enough data to learn well.
- One option is to treat these variables as numerical. In this approach, hour 12 is as close to 11 as it is to 1, but direct use of numerical values don't preserve this fact.
- Solution: Normalize all hours (min-max is between 0 and 2π) and apply a sine/cosine transformation. This ensures that 0 and 23 are close to each other (due to continuity).
- One advantage of this approach is that, using categorical features essentially means that you don't consider distance between two categories as relevant, but this is not the case for cyclic variables.
- Another approach might be to group days/hours/months and use them as categorical variables. In any case, most of these are case-specific and may perform differently for different cases.

Encoding via Scikit-learn

- Sklearn has 4 types of encoders:

```
# Uses an arbitrary number for each category (1D data, target)
from sklearn.preprocessing import LabelEncoder

# Like LabelEncoder but for 2D data (for features)
from sklearn.preprocessing import OrdinalEncoder

# Creates a binary feature for each category
from sklearn.preprocessing import OneHotEncoder

# Turns multi-class labels to binary labels
from sklearn.preprocessing import LabelBinarizer
```

- Note: Use manual mapping for ordinal features

- For frequency and target encoding, you can use

```
!conda install -c conda-forge category_encoders
import category_encoders as ce
# count encoding example
cat_features = ['category', 'currency', 'country']
count_enc = ce.CountEncoder()
count_encoded = count_enc.fit_transform(df[cat_features])
df = df.join(count_encoded.add_suffix("_count"))
```

Data reduction

- Data: the more the better?
- Do we need all of it?
- Curse of dimensionality
- Data reduction can be done in a number of ways:
  - **Sampling**: Using only a subset of data
  - **Feature selection**: Using only a subset of attributes
  - **Dimensionality reduction**: Combining existing attributes into a new data frame with a reduced number of attributes.

Sampling

- **Systematic random sampling**
  - Randomly select a starting number and an interval
  - Example: Sample of 10 in a population of 100

    Start #: 14, interval: 8 => {14,22,30,38,46,54,62,70,78,86}
- **Simple random sampling**
  - The entire process of sampling is done in a single step with each subject drawn independently (like lottery)
- **Cluster sampling**
  - Select individual subjects from each group of clusters by either systematic or simple random sampling.
  - Example: Geographical clusters. Divide the entire population of Turkey into cities for a study of the academic performance of students in Turkey.
- **Stratified sampling**
  - Data set is divided into subgroups, then subjects are randomly selected from subgroups in a proportional way.

Feature selection

- If you have 1,000 dimensions and 10,000,000 instances, how much memory would you need?

  $10^3 \times 10^7 \times 8 = 80$ GB

- If you try to build a model with all possible combinations of "k" attributes, you end up with 2^k-1 models. For features A, B, C and D we have 15 models:

  {A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD, ABCD}

- Is every feature useful? Any redundancy?
- Remove irrelevant and redundant features and select the important ones:

Feature selection – cont'd

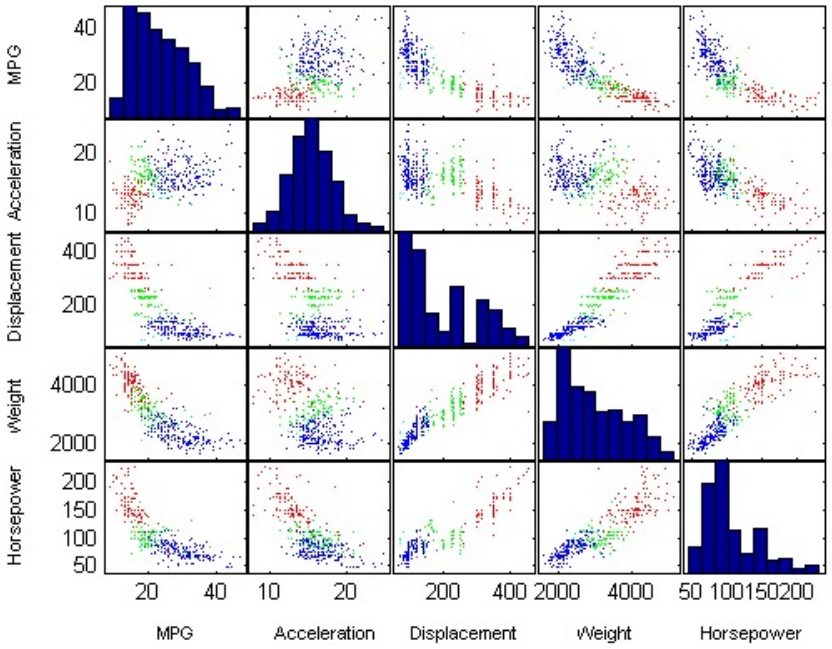
- **Feature ranking** (feature importance)
  - Process of ranking the attributes by their value to predictive ability of a model
  - Domain expertise might provide good insight
- Potential benefits of feature selection
  - Reduced overfitting: With less redundant data, model generalizes better without fitting to noise
  - Improved accuracy: With less amount of misleading data comes better accuracy
  - Reduced training time: Training phase speeds up with less amount of relevant data
  - Parsimony: Remember, simpler the better

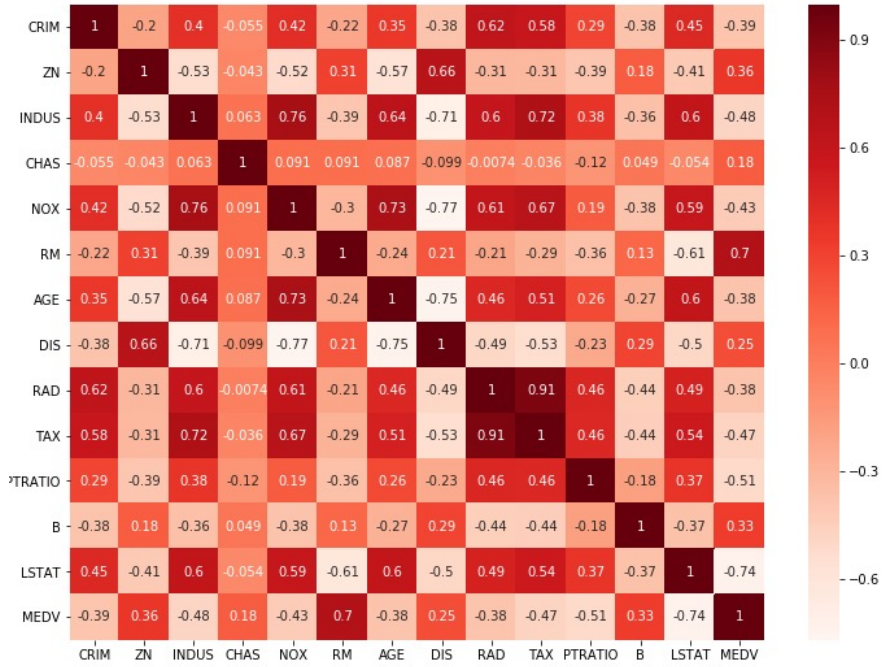
Feature selection: methods

- **Filter methods**
  - We filter and take only a subset of the relevant features.
    Pros: No training is involved. Simple and fast.
    Cons: They may fail if there is not enough data though.
  - Apply a statistical measure to assign a scoring to each feature and decide whether it should be kept or removed.
  - These statistical measures/tests include:
    Pearson's/Spearman's correlation, Chi-square test, Cramer's V coefficients, Information gain, etc.
  - A feature may be important if it is highly correlated with the dependent variable (target). Selected thresholds are used to filter features.
  - This could be done via eyeball test over scatter plots or correlation heatmaps/matrices (see next slide).

Feature selection: methods

- Scatterplots and heatmaps for correlations:





Feature selection: methods

- **Wrapper methods**
  - These methods use a Machine Learning model.
    Pros: They are more accurate than Filter Methods.
    Cons: Involves training. Computationally expensive and may not be practical for large data sets.
  - Wrapper methods search through the subset of feature space, train a model, evaluate it and iterate...
  - They employ simple greedy search heuristics for feature selection. There are 3 basic types of wrapper methods:
    - Forward selection
    - Backward elimination
    - Recursive Feature Elimination (RFE)

Feature selection: methods

- **Wrapper methods** (cont'd)
  - **Backward elimination** - start with the full set, gradually remove the "weakest" features

    start: $\mathbf{X} = [\, \mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3, \mathbf{X}_4 \,]$

    $\mathbf{X} = [\, \mathbf{X}_1, \mathbf{X}_3, \mathbf{X}_4 \,]$

    stop: $\mathbf{X} = [\, \mathbf{X}_1, \mathbf{X}_3 \,]$
  - **Forward selection** - start with an empty set, gradually add the "strongest" features

    start: $\mathbf{X} = [\, \mathbf{X}_1 \,]$ => $\mathbf{X} = [\, \mathbf{X}_1, \mathbf{X}_2 \,]$ => $\mathbf{X} = [\, \mathbf{X}_1, \mathbf{X}_3 \,]$

    => $\mathbf{X} = [\, \mathbf{X}_1, \mathbf{X}_3, \mathbf{X}_4 \,]$: stop
  - **Recursive Feature Elimination** (RFE): It works by recursively removing attributes and building a model on the remaining attributes. It uses accuracy metric to rank the features according to their importance.

Feature selection: methods

- **Embedded methods**
  - These are iterative methods that extract those features contributing the most to the training process iteration.
  - Regularization methods which introduce a penalty for complexity are the most commonly used techniques.
  - Lasso regularization, for example, sets the coefficients of irrelevant features to zero and eliminates them.
- **Summary**:
  - Wrapper and Embedded methods are more accurate but are computationally expensive. They are practical with features 20 or less.
  - Filter methods are simple, fast , but usually less accurate.

Imbalanced data sets

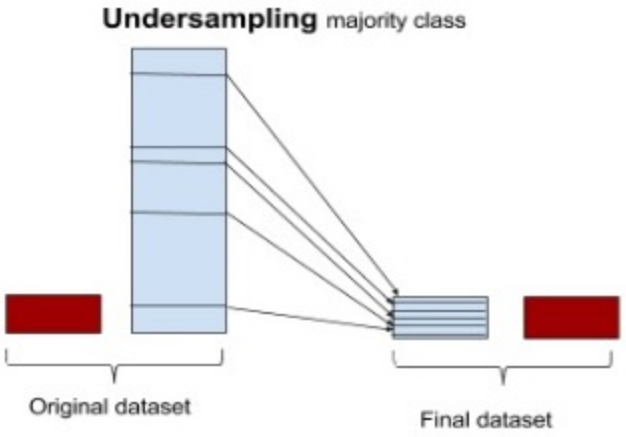
- Imbalanced classes often considers “imbalanced” to mean a minority class of 10% to 20%
- In reality, datasets can get far more imbalanced than this (minority class is greatly outnumbered by the majority):
  - 2% of credit card accounts falls victim to fraud
  - Medical screening for positives (HIV, malicious tumors, etc)
  - Conversion rates of online ads
  - Defect rates in manufacturing (0.1%)
- Suppose you have a data set with 5% fraud and 95% non-fraud transactions.
- Potential problems with imbalanced data sets:
  - Sample is biased towards non-fraud which result in biased predictions and misleading accuracies as models will learn non-frauds better
  - Any model that will call every decision a “non-fraud” will get a classification accuracy of 95%... Crazy!

Methods to deal with Imbalanced data sets

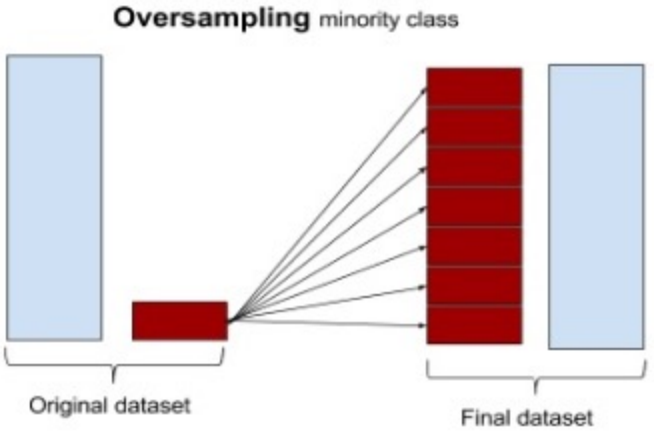
- Ignore the problem and try algorithms that are better able to deal with imbalanced data (like Decision trees, Random forests) using a stratified data split.
- Adjust class weights (by resampling) accordingly: Under- or Over-sampling, SMOTE, AdaSYN
- Adjust the decision thresholds
- Use cost-sensitive algorithms (such as Logistic Regression, SVM, Ensemble methods – Boosting) using a class balance parameter
- Penalize algorithms (cost-sensitive training)
- Use other performance metrics for measuring accuracy
  - Use the Confusion matrix (precision/recall) to reduce false positives and false negatives
  - F1 score (weighted average of precision/recall) or AUC

Methods to deal with Imbalanced data sets

- **Undersampling** (delete observations from the over-represented class – throws away useful info)
  - Undersampling can make the independent variables look like they have a higher variance than they do



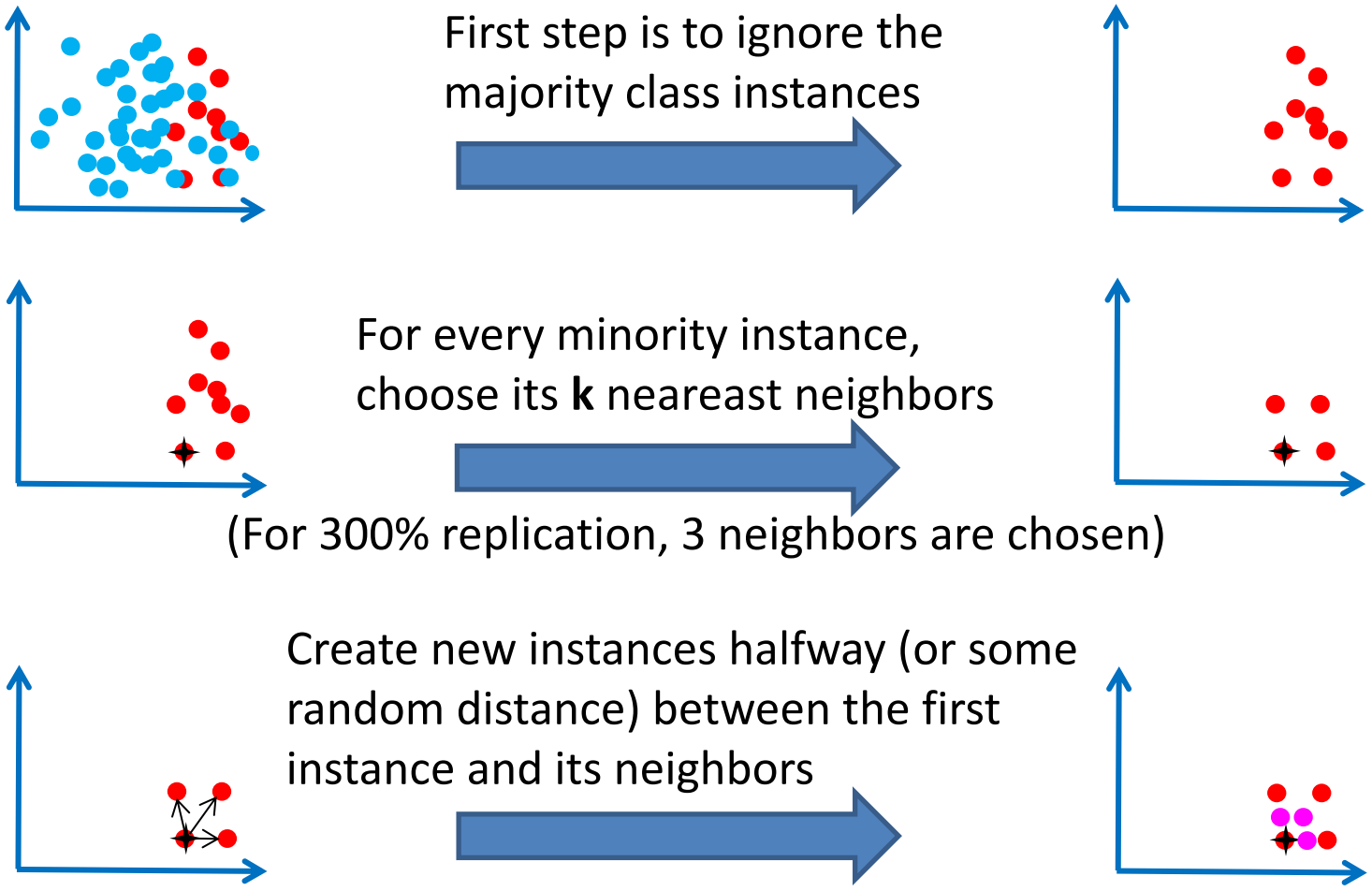
- **Oversampling** (add copies of observations from the under-represented class – increases training time)
  - Duplicate data makes variables appear to have lower variance than they do



Ref: https://www.svds.com/learning-imbalanced-classes/

Methods to deal with Imbalanced data sets

- Synthetic data generation through sampling randomly from observations in the under-represented class (e.g. **S**ynthetic **M**inority **O**ver-sampling **TE**chnique: **SMOTE**)



Feature Engineering

"**Good features allow a simple model to beat a complex model**" *Peter Norvig*

- Better features => Flexibility and simpler models
- Feature Engineering is the art and science of generating additional relevant features from the existing ones in the data to increase predictive power of the algorithm
- Feature generation/construction:
  - Process of manually constructing new attributes from raw data for a higher predictive power
- In essence, “feature engineering” is not needed. Given enough data, a complex machine learning algorithm should be able to learn the features directly from data.

Feature generation/construction

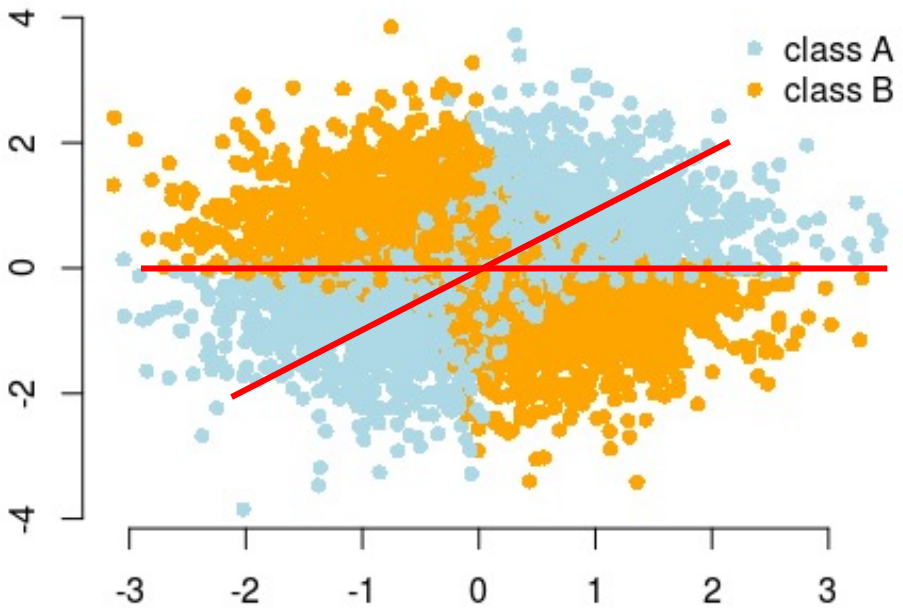
- **Derive unique identifiers to obtain hierarchy information**
- Features like Customer ID, Product ID etc. are removed from the data file in most cases. Sometimes a careful investigation of these features can be converted into forms that have predictive power. Here is an example:

Product ID	Count	…	Product category
S1	301		S
S2	533		S
S3	70		S
M1	63		M
M2	90		M
Q1	77		Q
Q2	100		Q
Q3	100		Q
Q4	125		Q
Q5	178		Q

Feature generation/construction

- **XOR example**:

  Imagine a set of data with two features (x and y) and a binary class (A and B) to predict. A decision boundary based on features x and y cannot separate the two classes.



  But if you generate an additional feature z=xy, then the problem becomes trivial as z>0 gives you nearly perfect decision criterion for classification.

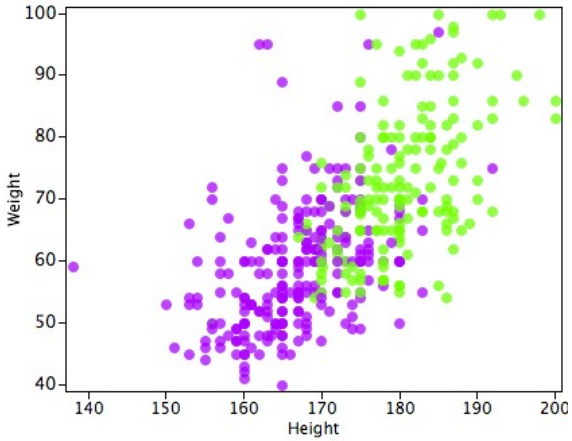
Feature generation/construction

- **Example:** Is there a comfortable driving distance between the two cities?

CITY1 LAT.	CITY1 LNG.	CITY2 LAT.	CITY2 LNG.	DRIVABLE?
123.24	46.71	121.33	47.34	Yes
123.24	56.91	121.33	55.23	Yes
123.24	46.71	121.33	55.34	No
123.24	46.71	130.99	47.34	No

DISTANCE (KM.)	DRIVABLE?
22.5	Yes
45	Yes
1134	No
3913	No

- **Example:** Construct a new feature from correlated features

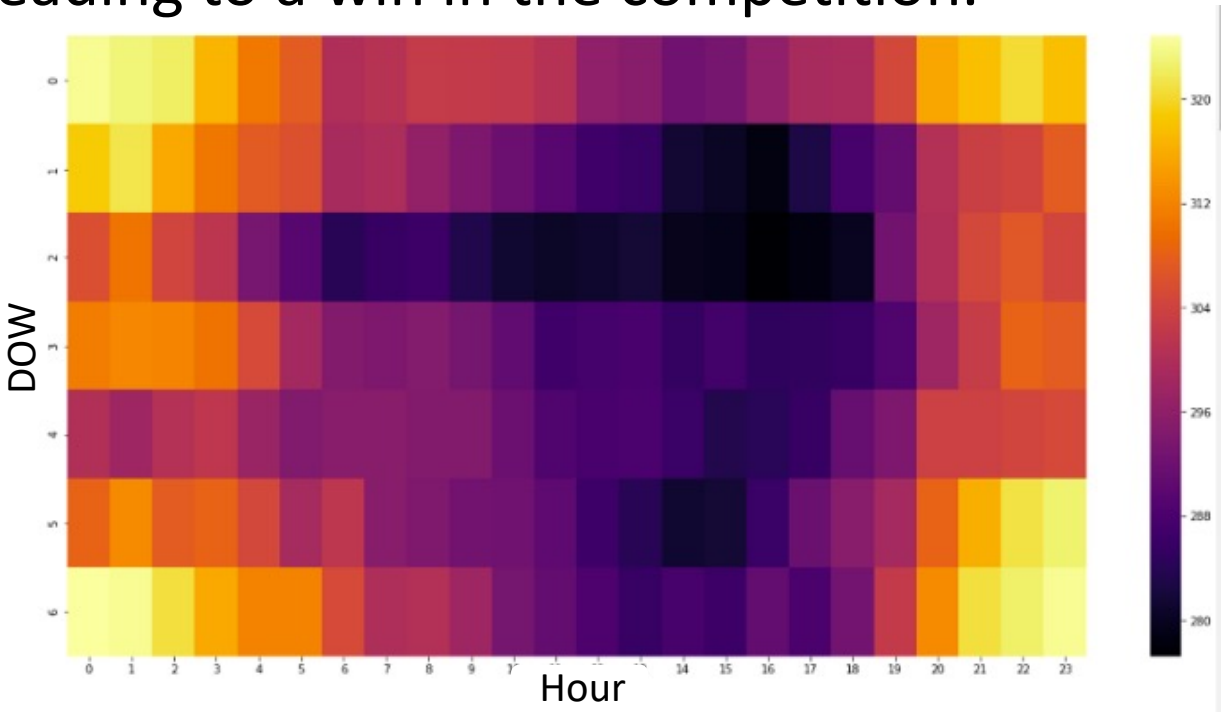


Weight (kg)	Height (m)
W_1	H_1
...	...
...	...

BMI index
W_1 / H_1^2
...
...

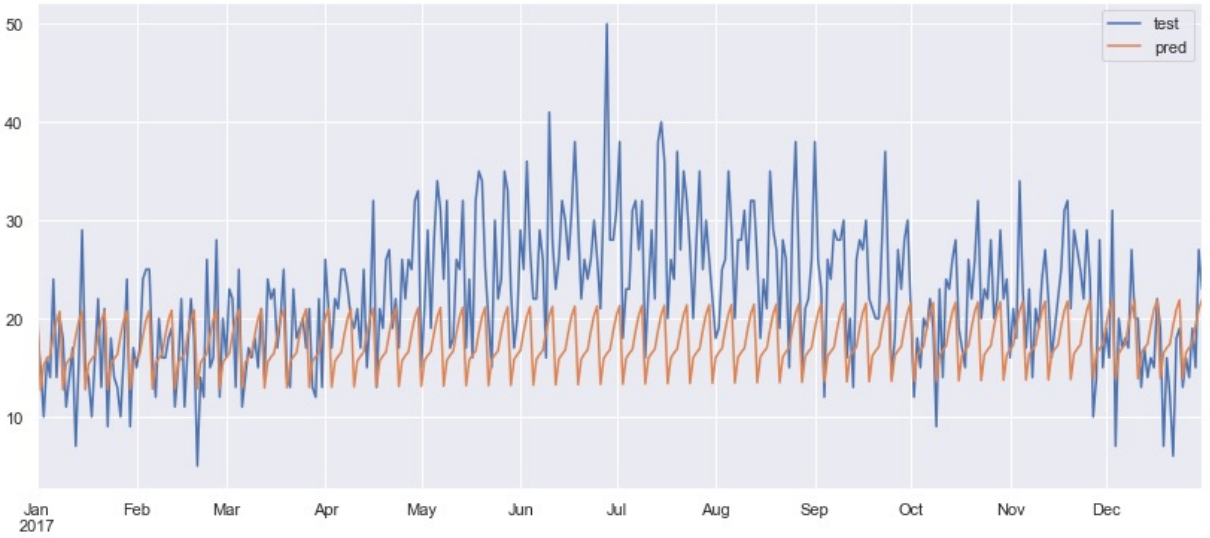
Feature generation/construction

- **Example:** Kaggle competition about Energy prediction
- Transforming the data to image, new feature from the heatmap formed via groupby data of Day of Week (**DOW**) and **Hour** of the day is created.
- The engineered feature (weekday/weekend) improved the score leading to a win in the competition.

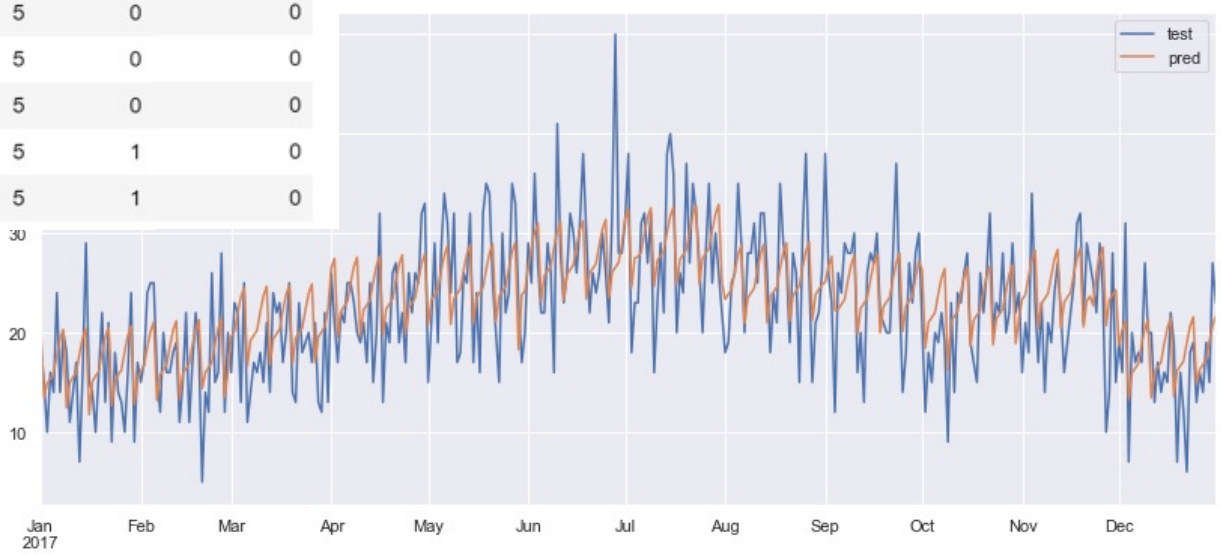


Feature generation/construction

date	sales
2017-12-27	14
2017-12-28	19
2017-12-29	15
2017-12-30	27
2017-12-31	23

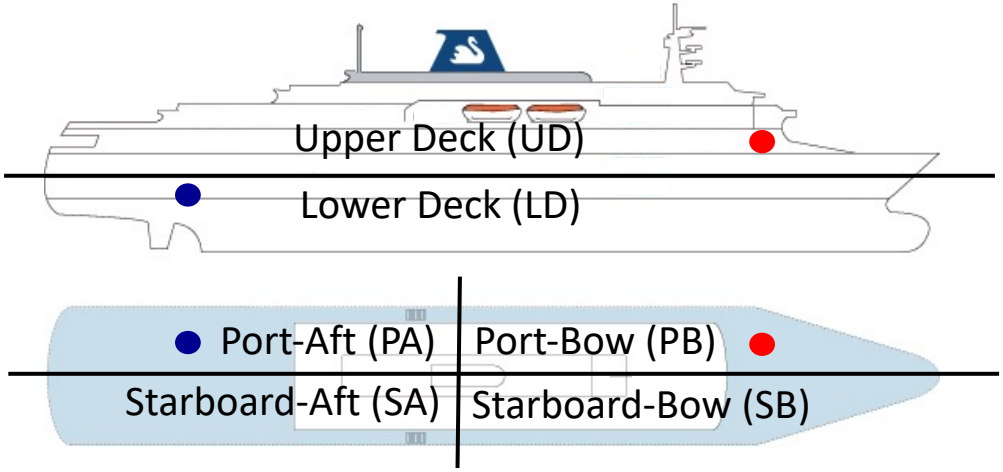


date	sales	month	weekday	year	weekend	holiday_bool
2017-12-27	14	12	2	5	0	0
2017-12-28	19	12	3	5	0	0
2017-12-29	15	12	4	5	0	0
2017-12-30	27	12	5	5	1	0
2017-12-31	23	12	6	5	1	0



Feature generation/construction

- **Example:** Imagine you're working on a problem similar to one that predicts Titanic survivors. Further assume that a set of data gives you the cabin numbers of all passengers.
- Cabin numbers alone may not mean much and don't provide any additional information. If we can, however, classify the cabin numbers with respect to where they are in the ship, we may get some extra prediction power.



#	…	…	UD	LD	PB	SB	PA	SA
●	…	…	1	0	1	0	0	0
●	…	…	0	1	0	0	1	0