Data cleaning

Data cleaning

1. Duplicate values
2. Missing values
3. Outliers
4. Values that don’t make sense

Duplicate values

- Understand why there are duplicates
- Two types of duplicate values:

client_id	age	sex
001	28	M
001	28	M

client_id	age	sex
001	28	M
001	32	M

- First case: we can usually drop one of the rows
- Second case: we need to understand the root of the issue and make assumptions (randomly drop one, get the average of the age, etc.)

Missing values

- It’s important to understand why they are missing
- Possible solutions
  a. Dropping rows with too many missing values
  b. Dropping columns with too many missing values
  c. Replacing missing values with
     - 0
     - the mode/mean/median value of the column
     - the result of a model based on other variables (excluding the target variable)

Outliers

- It’s important to understand why they are outliers
- Actual outliers
  a. Exclude them or
  b. Leave them but use models that are robust to outliers
- Measurement/imputation errors
  a. Exclude them
  b. Correct them manually
  c. Treat them as missing and use imputation methods

Values that don’t make sense

- Ex:

client_id	age	sex
001	999	M

- Treat them as measurement/imputation errors
  - Exclude them
  - Correct them manually
  - Treat them as missing and use imputation methods

Feature engineering

Feature engineering

1. Enriching your dataset
2. Calculating new features
3. Numeric transformations
    a. Quantization or binning
    b. Log transformation
    c. Min-max scaling/standardization
4. Encoding categorical variables
    a. One-hot, dummy, effect
5. Dimensionality reduction

Enriching your dataset

- Weather data
  a. When dealing with seasonal products (ice cream, hot chocolate, etc.)
- Economic data
  a. When dealing with products that are highly dependent on the economic activity (loans, car sales, etc.)
- Demographic
  a. When dealing with products that might be sensible to demographic differences (usually B2B products)

Calculating new features

- Examples:
  a. Recency (time since last purchase)
  b. Customer since (time since first purchase)
  c. Average basket (average spending in each purchase)
  d. Average price (average price of the products bought)
  e. Average number of products per purchase
  f. % of spending per product category
  g. Customer average rating

Numeric data transformation

- Binarization
- Quantization or binning
- Log transformation
- Power transformation
- Scaling

Numeric data transformation

Binarization

- It means turning a numeric variable into a binary one: for example, imagine we are using weather data to predict sales, and we have information on rain levels (in millimetres). Instead of using this information as a continuous variable, it might be more useful to take a binary variable instead, where 0 = no rain, 1 = any level of rain. This way, you lose information, since you don't know the amount of rain anymore, but in some cases this information can be irrelevant.

```
from sklearn.preprocessing import Binarizer
X = [[ 1., -1.,  2.],
     [ 2.,  0.,  0.],
     [ 0.,  1., -1.]]
transformer = Binarizer()
transformer.fit(X)  # fit does nothing.
transformer.transform(X)
```

Numeric data transformation

Quantization or binning

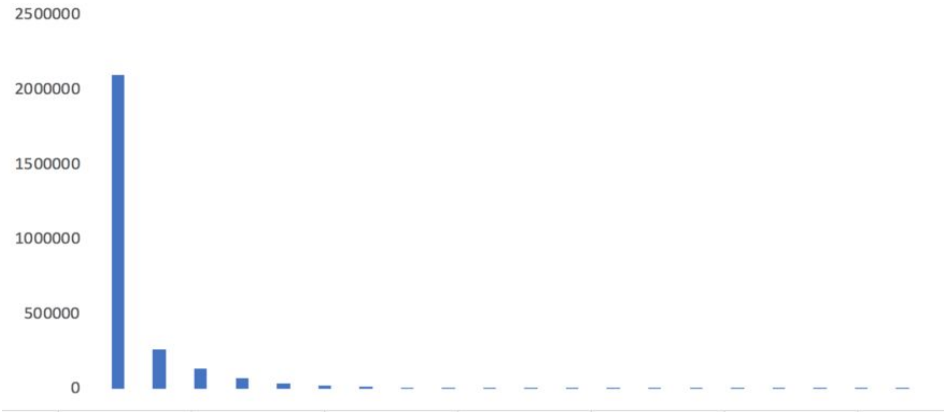
- In our previous example, instead of using a binary variable, we could also have transformed the original variable into a categorical one, where 0 = no rain, 1 = light rain, 2 = heavy rain, 3 = storm. The number of bins and their size may vary, and it might take you a few tries to get them “right”. It helps to take a look at the variable distribution (a histogram) to find reasonable cutoff points (such as the quartiles).

```
from sklearn.preprocessing import KBinsDiscretizer
X = [[-2, 1, -4,   -1],
     [-1, 2, -3, -0.5],
     [ 0, 3, -2,  0.5],
     [ 1, 4, -1,    2]]
est = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='uniform')
est.fit(X)
Xt = est.transform(X)
```

Numeric data transformation

Log transformation

- Imagine a variable with a histogram that looks like the picture below :

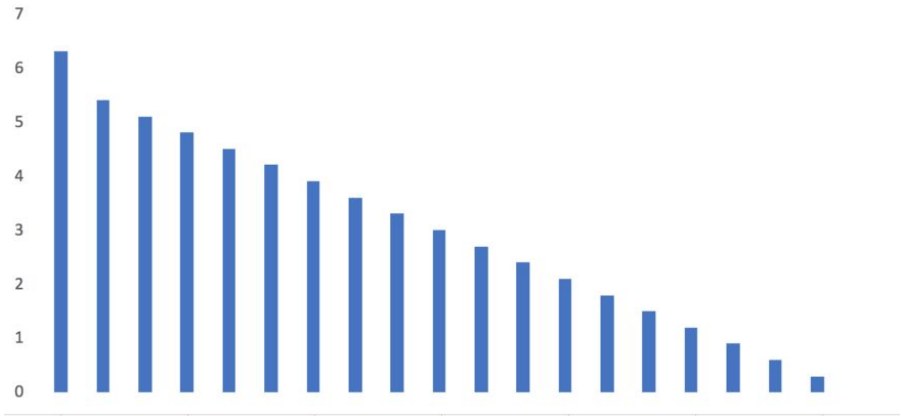


- This variable is highly concentrated in one tail (a heavy-tailed distribution), and this might create errors in some models. To fix this, we could apply a log transformation, meaning that, instead of using our variable X in our model, we could use a new variable: log(X).

Numeric data transformation

Log transformation

- Our new variable, log_X, has a much more “well-behaved” distribution:

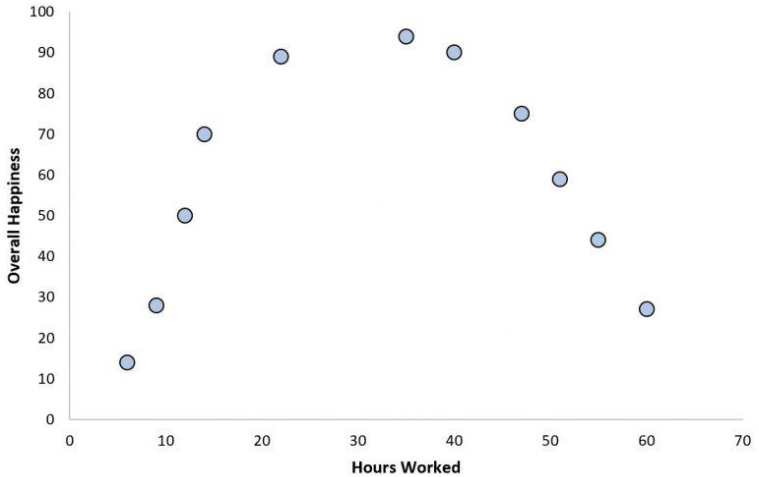


- This distribution usually yields better results for most models (think of a linear regression — it is easier to fit a straight line when our observations are not all concentrated around 0).

Numeric data transformation

Power transformation

- Power transformation is essentially taking your feature X and applying a power function to it, such as squaring or taking the square root.
- It can help your model account for non-linear interactions between your feature and your target variable. Ex.: hours worked vs. overall happiness:



Numeric data transformation

Scaling

- Scaling is specially useful when you use models that attribute different coefficients for your variables and you want to interpret these coefficients later. If your variables are not in the same scale, it will be very hard to compare coefficients between them.

```
from sklearn.preprocessing import StandardScaler
data = [[0, 0], [0, 0], [1, 1], [1, 1]]
scaler = StandardScaler()

print(scaler.fit(data))
print(scaler.transform(data))
print(scaler.transform([[2, 2]]))
```

Numeric data transformation

Scaling

- The most popular scaling method is called standardisation. Feature standardisation creates a new variable which has an average of 0 (it moves a variable's distribution to be around 0) and a variance of 1 (it scales the variable's variance). The way of doing it is using the formula below, where std_dev(x) represents the standard deviation of x:

$$x^* = \frac{x - mean(x)}{std\_dev(x)}$$

-
- Your variable will then assume negative and positive values, distributed around 0.

Numeric data transformation

Scaling

- If you don’t want to have negative values for your variable, you can use a min-max scaling, another popular scaling method, for which the formula is:

$$x^* = \frac{x - min(x)}{max(x) - \min(x)}$$

- This time, your new variable will be bounded between 0 and 1, making it easier to interpret its new values and any coefficients generated by your models.

Text data transformation

- Cleaning
- Stemming
- Bag-of-words
- TF-IDF

Text data transformation

Cleaning

- Text data contains a lot of noise, which should be dealt with before we try more advanced transformations.
- Deal with uppercase/lowercase letters, usually by turning all letters into lowercase, but be careful when dealing with cases that might differ in meaning (“penguins” and “Penguins” mean exactly the same thing, while “Friends” might mean the TV show and not the word friends).
- Get rid of stop words, which are words that don’t carry much meaning such as “is”, “are”, “in”, “for”, etc. There are public lists of stop words for many different languages, accessible directly from libraries in Python and R.
- Cleaning extra spaces, fixing mistyped words, etc.

Text data transformation

Stemming

- Stemming means transforming a word into its root. For example, the words “runs”, “runner”, “ran”, “run” would all become “run”.
- Sometimes it is a useful feature, but sometimes it can also change the meaning of words (“new” and “news” do not mean the same thing), so consider your specific case before applying this transformation.

Text data transformation

Bag-of-words

- It is a lot harder to deal with text data than it is to deal with numbers, so most of the methods used to deal with text are based on turning it into numbers.
- Count the number of times each word appears in a text, turn those counts into a vector and use these vectors to compare different texts.
- The problem with this approach is obvious: words can have different meanings, depending on the order they appear in, or on which words precede them. For instance, consider the two following sentences: “a man ate a hamburger” and “a hamburger ate a man”. Both have exactly the same word counts, and would be represented in the same way, while they clearly mean different things.
- To address this issue, a common solution is to create bags of n-grams, meaning that, instead of counting words we count sets of words that appear together (usually 2 or 3 words maximum). So in our example, we would count the number of occurrences for “a man”, “man ate”, “ate a”, “a hamburger”, etc.

Text data transformation

TF-IDF

- TF-IDF stands for Term Frequency — Inverse Document Frequency, which is a way of taking into account words occurrence in a text compared to other texts.
- It can be calculated in different ways, but a simple one is:

$$tfidf(w,d) = bow(w,d) * idf(w,d)$$

where:

$$idf(w,d) = (\#\ documents)/(\#\ documents\ containing\ word\ w)$$

$$bow(w,d) = \#\ occurences\ of\ word\ w\ in\ document\ d$$

Text data transformation

TF-IDF

- Let’s say we are analysing Trump’s tweets, and we want to know what are the words he uses the most. If we just look at simple word counts, we would get quite boring results: words such as “and” and “I” would come up because they are common. The problem is: they are common everywhere, not because Trump writes them particularly often, and that’s where TF-IDF can help us. It can be calculated in different ways, but a simple one is:
- By using TF-IDF, we give more weight to words that are not very common, helping us to see words that Trump uses a lot more when compared to other people.
- This time, we would probably get words such as “America”, “Congress” and “great”.

Categorical data transformation

- One-hot encoding
- Dummy encoding

Categorical data transformation

One-hot encoding

- Again, our goal is to turn a non-numerical variable into a numerical one. One-Hot Encoding creates n new variables, where n is the number of unique categories in our original variable.
- For example, if we have a Sex variable, we would have only 2 categories: “Man” and “Woman”. We would then replace the Sex variable by 2 new variables: Man and Woman for which the value is 1 if the observation belongs to the respective sex :

	Sex	Man	Woman
Alice	Woman	0	1
Bob	Man	1	0
Carl	Man	1	0
Dana	Woman	0	1

Categorical data transformation

One-hot encoding

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore')
X = [['Male', 1], ['Female', 3], ['Female', 2]]
enc.fit(X)

enc.categories_

enc.transform([['Female', 1], ['Male', 4]]).toarray()


enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])


enc.get_feature_names_out(['gender', 'group'])
```

Categorical data transformation

Dummy encoding

- Have you noticed that one of the variables above is redundant? We could keep only one of them without losing any information.
- If you drop the Woman column, you can still know the person's sex (if it's not a woman, than it's a man). Dummy Coding does exactly that :

	Sex	Man
Alice	Woman	0
Bob	Man	1
Carl	Man	1
Dana	Woman	0

Categorical data transformation

Dummy encoding

```
from sklearn.preprocessing import OneHotEncoder

enc = OneHotEncoder(handle_unknown='ignore', drop='first')
X = [['Male', 1], ['Female', 3], ['Female', 2]]
enc.fit(X)

enc.categories_

enc.transform([['Female', 1], ['Male', 4]]).toarray()

enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])

enc.get_feature_names_out(['gender', 'group'])
```

Dimensionality reduction

- Principal Component Analysis (PCA)
- Clustering (k-means, etc.)

Dimensionality reduction

Principal Component Analysis (PCA)

- PCA is a subject on its own. If you want to understand the details a bit better, check out this article
- In general terms, PCA is a method that reduces dimensions while keeping as much information as possible
- It takes the original features (which can also be called dimensions), and creates new features, based on projections from the original dimensions. You start with 30 dimensions and can end up with 2, 3, 4 or 18, which are called principal components
- You will have a trade-off between reducing dimensions and keeping information and variance
- Keeping at least 80% of the original variance is a common benchmark, but it really depends on your application.
- PCA can be useful when you have too many variables and not many observations

Dimensionality reduction

Principal Component Analysis (PCA)

```
import numpy as np
from sklearn.decomposition import PCA

X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
pca = PCA(n_components=2)
pca.fit(X)

print(pca.explained_variance_ratio_)
```

Dimensionality reduction

Clustering (k-means, etc.)

- Clustering is also a subject on its own. If you want to understand the details a bit better, check out this article
- There are multiple ways to make clusters, but essentially, they all split observations into different clusters (groups)
- These clusters will be created trying to keep similar observations in the same clusters, and different observations in different clusters
- The number of clusters can be determined by the user or automatically, depending on the algorithm
- It can also be useful when you have too many variables and not many observations

Dimensionality reduction

Clustering (k-means, etc.)

```
from sklearn.cluster import KMeans
import numpy as np

X = np.array([[1, 2], [1, 4], [1, 0],
              [10, 2], [10, 4], [10, 0]])
kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
kmeans.labels_

kmeans.predict([[0, 0], [12, 3]])

kmeans.cluster_centers_
```