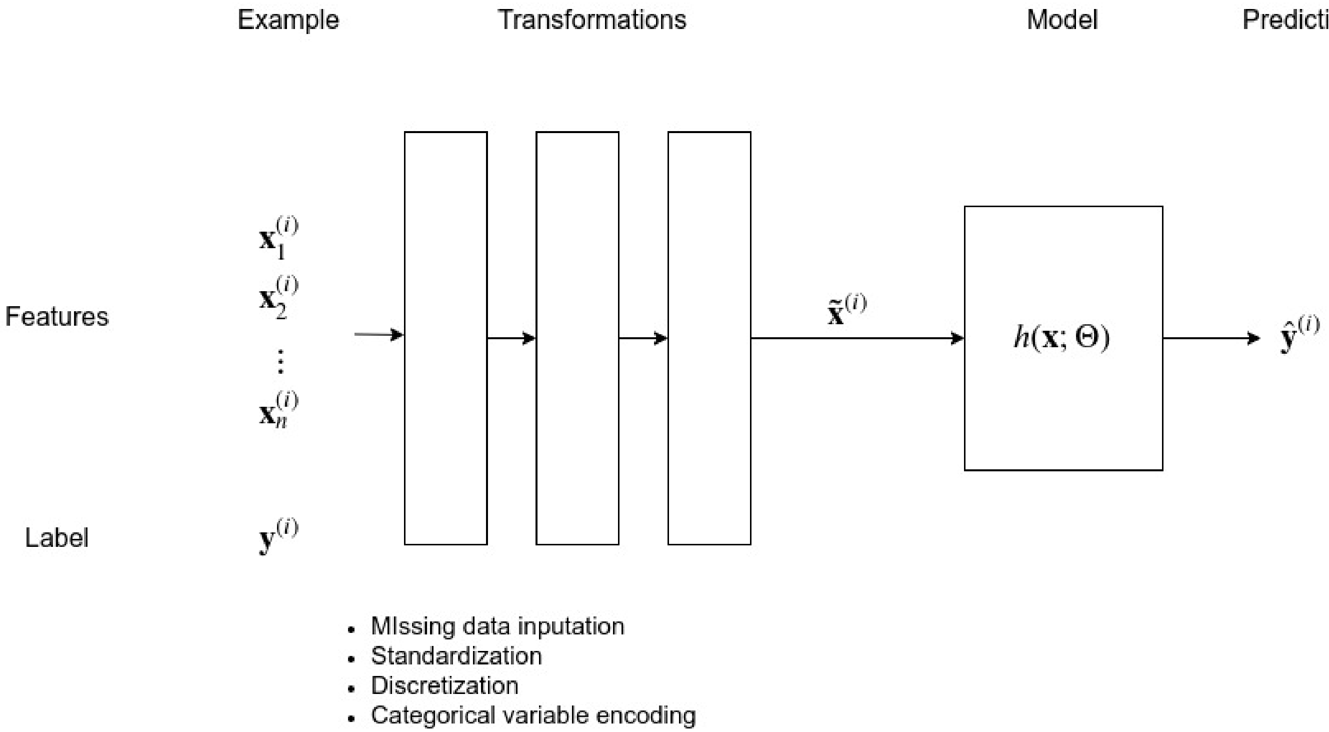
Prepare data: transformations

Feature engineering, or transformations

- takes an example: vector $\mathbf{x^{(i)}}$ with n features
- produces a new vector $\tilde{\mathbf{x}}^{\mathbf{(i)}}$, with n' features

We ultimately fit the model with the transformed *training* examples.

Feature Engineering



Transforming data (Recipe C.3) may be **the most important** step of the multi-step Recipe !

It is often the case that the "raw" features given to us don't suffice

- we may need to create "synthetic" features.
- This is called **feature engineering**.

In the "curvy" data case, adding the squared feature was key to a better prediction.

There will be other reasons for transforming the data (e.g., accomodating model assumptions).

Some of these transformations may *alter* the raw features rather than just augment them.

Transformations in detail will be the subject of a separate lecture but let's cover the basics.

Let's consider a second reason for transformation: filling in (imputing) missing data for a feature.

$\boxed{\#}$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$
1	1.0	10
2	2.0	20
$\vdots$	$\vdots$	$\vdots$
i	2.0	NaN
$\vdots$	$\vdots$	$\vdots$
m	...	

In the above: feature $\mathbf{x}_2$ is missing a value in example i: $x^\ip_2 = \text{NaN} $

We will spend more time later discussing the various ways to deal with missing data imputation.

For now: let's adopt the common strategy of replacing it with the median of the defined values:

$$\text{median}(\mathbf{x}_2) = \text{median}(\{\mathbf{x}_2^{(\mathbf{i})} | 1 \le i \le m, \mathbf{x}_2^{(\mathbf{i})} \ne \text{NaN}\})$$

This imputation is a kind of data transformation: replacing an undefined value.

”Fitting” transformations

Transformations often have their own parameters $\Theta_{\text{transform}}$ that is separate from the Θ parameters of the model.

In the case of imputation: the mean/median of a feature j for a missing value.

In that case: $\Theta_{\text{transform}}$ must contain $\text{median}(\mathbf{x}_j)$

The process of Transformations is similar to fitting a model and predicting.

The parameters in $\Theta_{\text{transform}}$

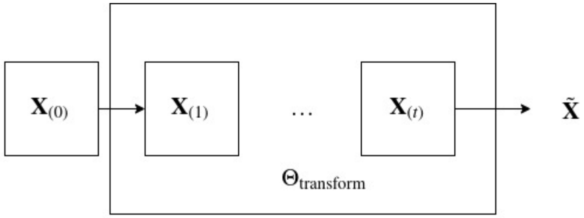
- are "fit" by examining all training data $\mathbf{X}$
- once fit, we can transform ("predict") *any* example (whether it be training/validation or test)

Note that the transformation of an example depends on $\Theta_{\text{transform}}$, which is fit *only on the training data*.

When transforming any example (including one *not* in $\mathbf{X}$) one uses $\Theta_{\text{transform}}$ from the transformation fitting

- You **do not** recalculate $\Theta_{\text{transform}}$ on test examples !
    - Just imagine that you are given each test example in isolation-- there is no summary statistic to compute!

Feature engineering: fit, then transform



Train

$$\text{fit} : \mathbf{X}_{(0)} \mapsto \Theta_{\text{transform}}$$

$$\text{transform}(\mathbf{x}^{(\mathbf{i})}; \Theta_{\text{transform}}) \mapsto \tilde{\mathbf{x}}^{(i)}$$

Test

$$\text{transform}(\underline{\mathbf{x}}; \Theta_{\text{transform}}) \mapsto \underline{\mathbf{x}}'$$

NO fitting at test time, re-use

$\Theta_{\text{transform}}$

- standardization: mean($\mathbf{X}$), std-dev($\mathbf{X}$)
- scaling: min($\mathbf{X}$), max($\mathbf{X}$)
- imputation: median($\mathbf{X}$)

To re-iterate:

- **No** fitting is applied to test examples only train !
- The $\Theta_{\text{transform}}$ obtained from training data is used in transforming *test* as well as *train* examples

There are several reasons not to re-fit on test examples

- it would be a kind of "cheating" to see all test examples (required to fit)
- you should assume that you only encounter one test example at a time, not as a group

Transformations are applied to both training and test examples

- training examples so that the model may be fit
- test examples in order to be able to predict
    - to the extent that transformations added features (e.g., $\mathbf{x}^2$) or changed features (imputation)
    - the test examples *must be transformed* the same way as training
        - otherwise they won't be similar to training examples, violating the fundamental assumption of ML

In [2]:
```
print("Done")
```

```
Done
```