Transformations

It is often the case that the "raw" example data is not yet in a form amenable to successful modeling.

We have already encountered two such cases

- The "curvy" dataset in Linear Regression
    - The raw data was not amenable to a linear model until we added a polynomial feature
- The Categorical features in our Classification task
    - Non-numeric data needed to be encoded as numbers

As motivation, consider a Binary Classification task

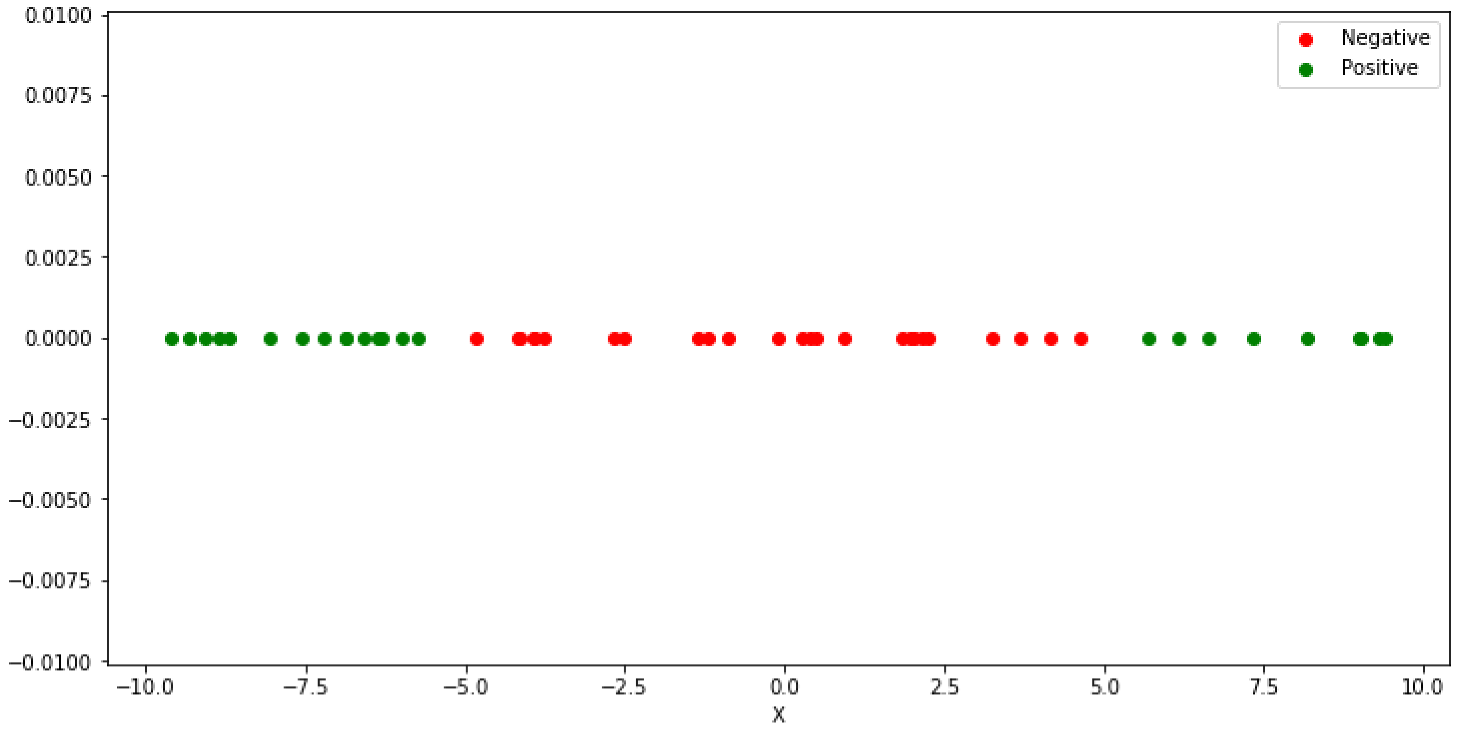
- examples have a single numeric feature
- Classes Positive and Negative
    - encoded as 1 and 0

It should be clear from the plot that the classes are not linearly separable.

In [16]:
```
th = transform_helper.Transformation_Helper()
fig_raw, ax_raw, fig_trans, ax_trans = th.LinearSeparate_1d_example(max_val=10,
num_examples=50, visible=False)

fig_raw
```

Out[16]:

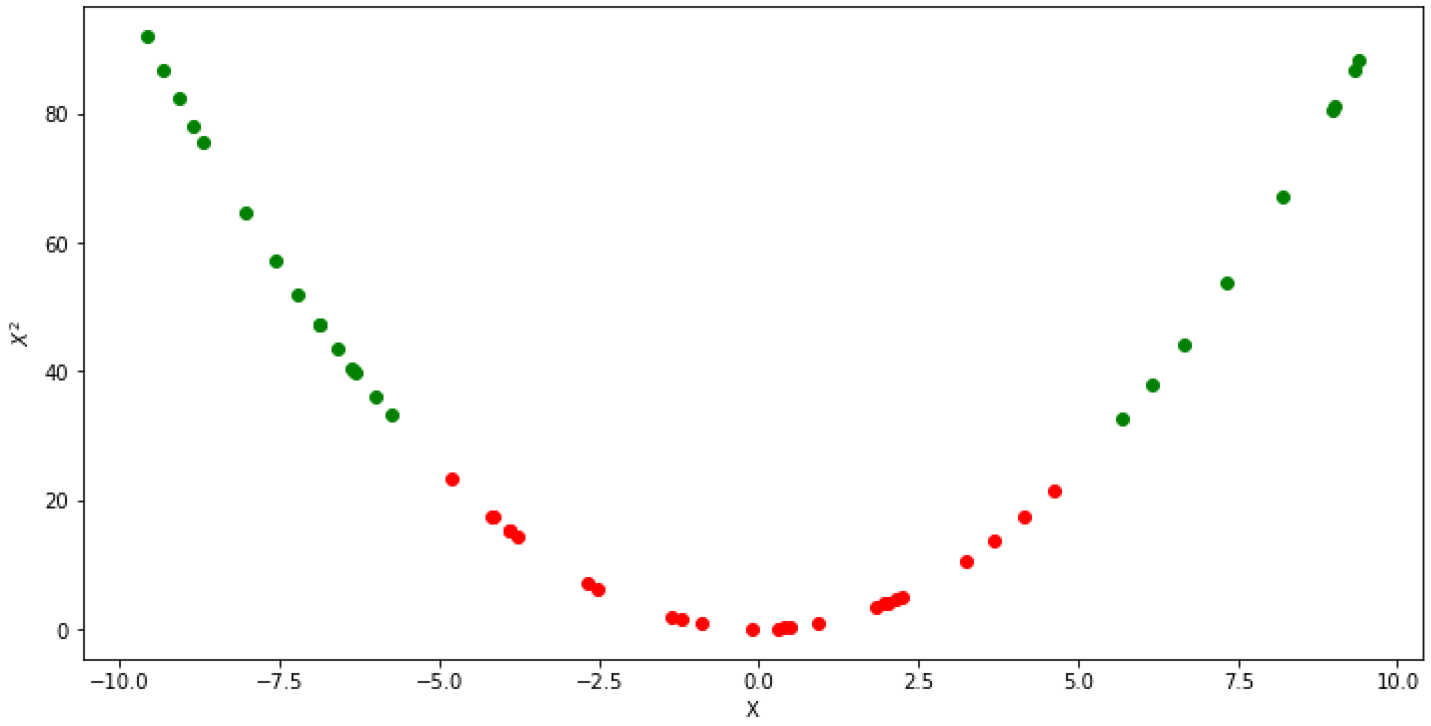


But consider the simple transformation of added a new dimension

- $\mathbf{x}_2 = \mathbf{x}_1^2$

```
In [17]: fig_trans
```

Out[17]:



This simple transformation has made the data linearly separable

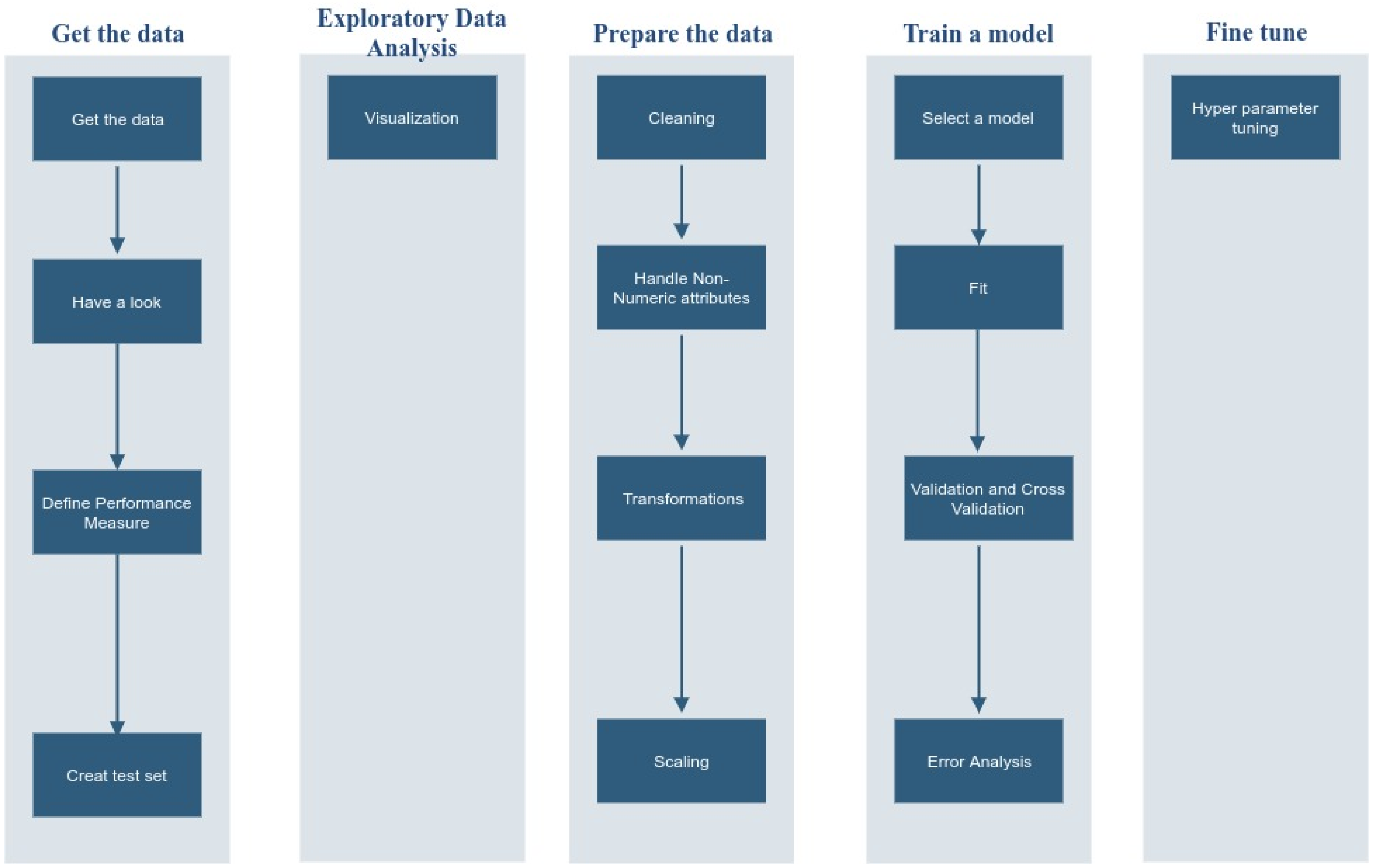
- and thus amenable to several classification models

In [ ]:

```
In this module we focus on the process of transforming raw examples into a form that
facilitates successful modeling.

We will use the term *Transformation* broadly to encompass this process, sometimes including
- Cleaning
- Numericalization (Handle Non-Numeric attributes)
- Scaling
```

Process


Get the data
Get the data
Have a look
Define Performance Measure
Creat test set
Exploratory Data Analysis
Visualization
Prepare the data
Cleaning
Handle Non-Numeric attributes
Transformations
Scaling
Train a model
Select a model
Fit
Validation and Cross Validation
Error Analysis
Fine tune
Hyper parameter tuning

Our focus

- Is not so much on the *how* (relatively straight forward)
    - We will enumerate transformations built-into `sklearn`
- But on the *when* and *why*
    - Transformations are one of the key skills of a successful Data Scientist

The mechanics of transformations

Let's briefly review the *how*:

- Mechanics of transformations (Transformations_Mechanics.ipynb)
- Coding transformations in `sklearn` (Transformations_Pipelines.ipynb)

The why’s of transformations

There are many different types of transformations, and the same transformation may be used for several reasons.

This makes it hard to provide a clean taxonomy.

We will instead motivate most transformations with example use cases.

Making the data fit your model’s assumptions

We have thus far been dealing with models that assume a linear relationship between targets and features.

Sometimes, either the target (e.g., Logistic Regression) or the features ("curvy data" for Linear Regression) need to be transformed to induce linearity.

Let's visit the notebook section Inducing linearity (Transformations.ipynb#Linearity-inducing-transformations:-Making-data-fit-your-model)

Missing features

Sometimes, your examples have all the "information" you need, but in the wrong form.

Creating new "synthetic" features from raw features is one way of making this information available to the model.

Let's visit the notebook section Missing numeric features (Transformations.ipynb#Transformation-to-add-a-%22missing%22numeric-feature)

Missing ”group” indicator feature

identical up to an additive constant

There is a more subtle case of a missing feature

- examples that naturally partition into sub-groups

The sub-groups might be examples that come from similar geographies or points in time.

The key is that

- The relationship between target and features is *almost identical* between groups
- with the exception of a constant shift

Adding indicator/dummy variables as new features is the way to address this.

Let's visit the notebook section [Missing indicator (Transformations.ipynb#Transformation-to-add-a-%22missing%22-indicator)](Transformations.ipynb#Transformation-to-add-a-%22missing%22-indicator)

Cross features

We witnessed the power of a simple indicator feature to isolate differences between groups of examples.

It is possible to

- Create multiple indicator features
- Create indicator features that are the *product* of other indicators

This is called a *cross feature* and is a powerful way to isolate complicated sub-groups of examples.

Let's visit the notebook section [Cross features (Transformations.ipynb#Cross-features)](Transformations.ipynb#Cross-features)

Feature Scaling

There is a class of transformations that alter the *scale* (magnitude) of features.

In the Recipe for ML, Scaling is treated as separate from the othe transformations.

Perhaps this is because scaling is sometimes performed

- *not* strictly because of the relationship between target and features
- but because of the mathematics of the *loss function*

Let's visit the notebook section [Scale sensitive loss functions (/Transformations.ipynb#Feature-scaling)](/Transformations.ipynb#Feature-scaling)

Normalization

We will use the term "normalization" (non-standard terminology ?) to refer to a transformation that re-scales examples by different amounts

- as opposed to traditional "scaling" where all examples are scaled equally

Let's visit the notebook section Target normalization (Transformations.ipynb#Target-??-normalization)

Other transformations

The continuation of the linked notebook describes more transformations.

We encourage you to review these.

- Categorical variable transformation (Transformations.ipynb#Categorical-transformation)
- Transformations to induce normality (Transformations.ipynb#Normality-inducing-transformations)
- Other transformation (Transformations.ipynb#Other-transformations)

```
In [18]: print("Done")
```

```
Done
```