A few coding practices we can start doing

1. Labeled axes in np.arrays (pysindy, xarray)

Improves general readability and debugging. Not sure if it is usable with pytorch.

- Instead of

```
x_dot = differentiation_method(x, axis=-2)
foo = [bar(x[...,i]) for i in range(x.shape[-1])]

```

- becomes:

```
x_dot = differentiation_method(x, axis=x.ax_time)
foo = [
    bar(x[..., feature])
    for feature in range(x.n_coord)
]

```

2. Use typing to create special types

If you have a big project, you might want to create new variable types for debugging purposes.

Type system enforces mathematical compatibility and flexibility

1. TypeErrors found by code hints and SCA, easier to propagate
2. Doesn't mean `isinstance` checks
3. From

```
constraint = np.zeros(4) # nonnegativity
```

To

```
nonneg_constraint = np.zeros(4)
```

(ref: Joel Spolsky)

To

```
InequalityConstraint = typing.NewType(
    "InequalityConstraint", np.ndarray
)
nonneg_const = InequalityConstraint(np.zeros(4))

```

- Different LHS

```
if isinstance(self.feat_lib, WeakPDELibrary):
    x_dot = self.feat_lib.udot_integral(x)
else:
    x_dot = self.feat_lib.differentiate(x)

```

becomes:

```
x_dot = self.feat_lib.calc_lhs(x)

```

- `if ensembling: do this` becomes `EnsembleOptimizer(inner_optimizer)`

Nathar Kutz suggested to create a new Kaggle

And use a comprehensive metrics wheel, which would describe models:

- accuracy metrics
- noise tolerance
- so on