House prices (NumPy, OOP)

1) Implement dataset normalization to get X and Y features in range from -1..1.

$$X_i = 2 \cdot \left(\frac{X_i - min(X_i)}{max(X_i) - min(X_i)} - 0.5 \right)$$

Need also function to convert Y back to real values.
Solution: To convert values back – we need to remember min and max values of initial dataset, otherwise we don't know according to what values we have -1 and 1 boundaries.

```
def normalize(values: np.ndarray) -> Tuple[np.ndarray, np.ndarray, np.ndarray]:
    max_values = np.max(values, axis=0)
    min_values = np.min(values, axis=0)

    return 2.0 * ((values - min_values) / (max_values - min_values) - 0.5), min_values, max_values



def denormalize(values: np.ndarray, min_values: np.ndarray, max_values: np.ndarray) -> np.ndarray:
    return (values / 2.0 + 0.5) * (max_values - min_values) + min_values
```

2) Implement model with new functions

- Use code from 4. (?) task
- Add classes LossMSE (Mean square error loss function), LayerSigmoid

```
class SigmoidLayer:
    def __init__(self):
        self.x = None
        self.output = None

    def forward(self, x: Variable) -> Variable:
        self.x = x
        self.output = Variable(
            1.0 / (1.0 + np.exp(-x.value))
        )
        return self.output

    def backward(self):
        self.x.grad = -1.0 / (1.0 + np.exp(-self.x.value)) ** 2 * self.output.grad
```

```
class MSELoss:
    def __init__(self):
        self.y: Optional[Variable] = None
        self.y_prim: Optional[Variable] = None

    def forward(self, y: Variable, y_prim: Variable) -> float:
        self.y = y
        self.y_prim = y_prim
        return np.mean((y.value - y_prim.value) ** 2)

    def backward(self):
        self.y_prim.grad = 2.0 * (self.y_prim.value - self.y.value)
```

- Replace ReLU with Sigmoid in Model

We will use same Sigmoid formulas:

$$\frac{1}{1+e^{-x}}$$

$$\frac{\partial}{\partial x}\frac{1}{1+e^{-x}} = \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$$

- Train with LossMSE

And same MSE cost function:

$$L_{MSE} = \frac{1}{N} \cdot \sum_{i=0}^{N} \left(h_\theta(x_i) - y_i\right)^2$$

$$\frac{\partial}{\partial L_{MSE}} = \frac{2}{N} \sum_{i=0}^{N} \left(b + W \cdot x_i - y_i\right)$$

- Fine tune Hyper parameters so you can get lowest error in 300 epochs

Resulting model layers are:

```
self.layers = [
    LinearLayer(in_features=8, out_features=4),
    SigmoidLayer(),
    LinearLayer(in_features=4, out_features=4),
    SigmoidLayer(),
    LinearLayer(in_features=4, out_features=1)
]
```