

```

1 import numpy as np
2
3
4 class Layer:
5     def __init__(self, n_inputs, n_neurons):
6         self.size = (n_inputs, n_neurons)
7         self.weights = np.random.randn(n_inputs, n_neurons)
8         self.bias = np.zeros((1, n_neurons))
9
10    def __repr__(self):
11        return "Weights:\n{}\nBias:\n{}\n".format(self.weights, self.bias)
12
13
14    def forward(self, inputs):
15        self.output = np.dot(inputs, self.weights) + self.bias
16        return self.output
17
18    def lin_rect_act(self):
19        self.output_act = np.maximum(0, self.output)
20        return self.output_act
21
22    def sigmoid_act(self):
23        self.output_act = 1/(1+np.power(np.e, self.output))
24        return self.output_act
25
26    def mutate(self, intensity):
27        new = Layer(self.size[0], self.size[1])
28        new.weights = self.weights+.001*intensity*np.random.randn(self.size[0], self.size[1])
29        new.bias = self.bias+.001*intensity*np.random.randn(1, self.size[1])
30        return new
31
32
33 class NN:
34    def __init__(self, num_layers, sizes):
35        self.nl = num_layers
36        self.sizes = sizes
37        if num_layers != len(sizes)-1:
38            raise ValueError

```

```

38         raise ValueError
39     self.layers = [None]*num_layers
40     for i in range(num_layers):
41         self.layers[i] = Layer(sizes[i], sizes[i+1])
42
43     def __repr__(self):
44         return "Sizes:\n{}\nLayers:\n{}".format(self.sizes, self.layers)
45
46
47     def f_pass(self, inputs, how="lin_rect"):
48         if type(inputs) == pd.DataFrame:
49             self.output = inputs.values#np.array([inputs.values.tolist()]).T.tolist()
50         else:
51             self.output = inputs
52         for i in range(self.nl):
53             self.output = self.layers[i].forward(self.output)
54             if i < self.nl-1 and how == "sigmoid":
55                 self.output = self.layers[i].sigmoid_act()
56             elif i < self.nl-1 and how == "lin_rect":
57                 self.output = self.layers[i].lin_rect_act()
58         return self.output
59
60     def mutate(self, intensity):
61         new = NN(self.nl, self.sizes)
62         for i, layer in enumerate(self.layers):
63             new.layers[i] = layer.mutate(intensity)
64         return new
65

```

```

1 class Generation:
2     def __init__(self, population, sizes=None, initial=False):
3         if initial:
4             NNs = [None] * population
5             for i in range(population):
6                 NNs[i] = NN(len(sizes)-1, sizes)
7             self.pop = NNs
8             self.sizes = sizes
9         else:
10             self.pop = population

```

```

11
12
13 def f_pass(self, data, label=None, how="lin_rect"):
14     size = len(self.pop)
15     y_ = [None] * size
16     errors = [None] * size
17     for i in range(size):
18         y_[i] = self.pop[i].f_pass(data, how=how)
19         if label is not None:
20             errors[i] = np.sqrt((label - y_[i].T[0])**2).mean()
21     if label is not None:
22         self.errors = pd.Series(errors).sort_values()
23     return y_
24
25 def f_pass_sep_inputs(self, data, label=None, how="lin_rect"):
26     size = len(self.pop)
27     y_ = [None] * size
28     errors = [None] * size
29     for i in range(size):
30         y_[i] = self.pop[i].f_pass(data[i], how=how)
31     return y_
32
33 def next_gen(self):
34     survivors = 5
35     best = self.errors.index.tolist()[0:survivors]
36     next_gen = [None]*survivors*201
37     for i, index in enumerate(best):
38         next_gen[i] = self.pop[index]
39         for ten in range(100):
40             if i<2:
41                 for q in range(2):
42                     next_gen[i*200+ten*2+q+5] = self.pop[index].mutate(ten)
43             else:
44                 next_gen[i*200+ten*2+5] = self.pop[index].mutate(ten)
45                 next_gen[i*200+ten*2+6] = NN(len(self.sizes)-1, self.sizes)
46     g = Generation(next_gen)
47     g.sizes = self.sizes
48     return g

```

```

1 # def rand_train(shape, x, starting=1000, gens=10):
2 #     gen = Generation(starting, shape, True)
3 #     errs = pd.Series(dtype=float)
4 #     for i in range(gens):
5 #         print(str(100*i//gens)+"% Done")
6 #         gen.f_pass(x, y)
7 #         errs.loc[i] = gen.errors.iloc[0]
8 #         gen = gen.next_gen()
9 #         clear_output()
10 #     gen.f_pass(x, y)
11 #     ind = gen.errors.index[0]
12 #     return pd.Series(gen.pop[ind].f_pass(x).T[0]), errs

```

```

1 from IPython.display import clear_output
2 def rand_train(shape, x, y, starting=1000, gens=10, how="lin_rect"):
3     gen = Generation(starting, shape, True)
4     errs = pd.Series(dtype=float)
5     for i in range(gens):
6         print(str(100*i//gens)+"% Done")
7         gen.f_pass(x, y, how=how)
8         errs.loc[i] = gen.errors.iloc[0]
9         gen = gen.next_gen()
10        clear_output()
11    gen.f_pass(x, y)
12    ind = gen.errors.index[0]
13    return pd.Series(gen.pop[ind].f_pass(x).T[0]), errs, gen.pop[ind]
14

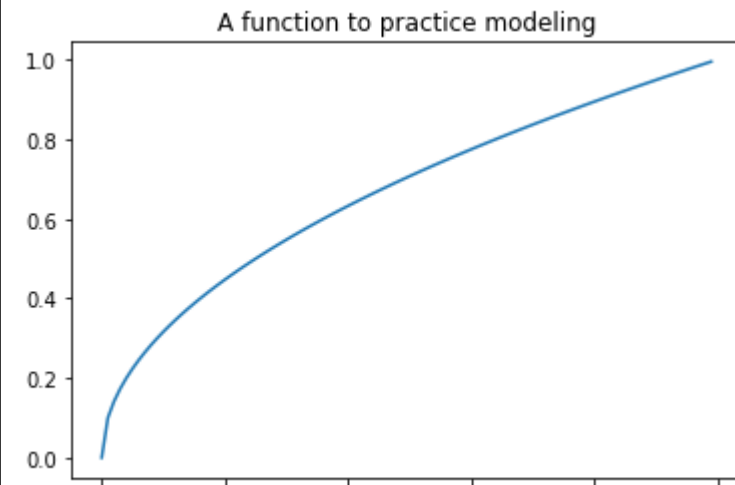
```

```

1 import pandas as pd
2
3 x = np.array(range(100))
4 y = np.sqrt(x)/10
5 Data = pd.Series(y, index=x)
6 Data.plot.line(title="A function to practice modeling")

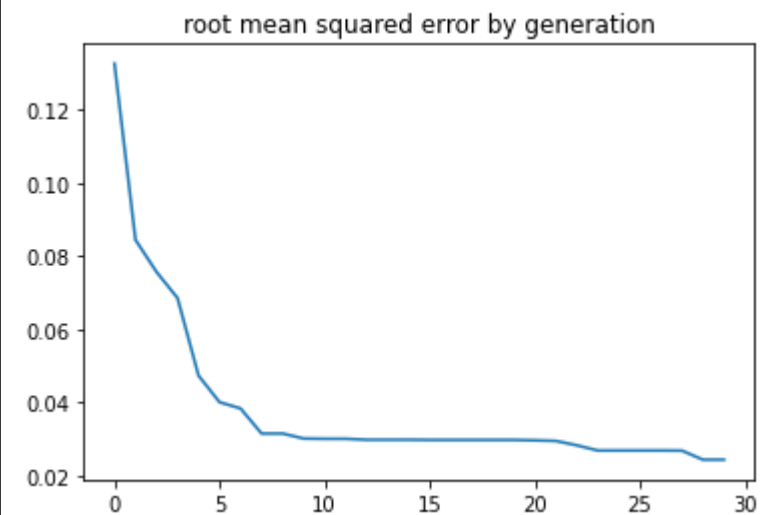
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9c6384abd0>
```



```
1 reg = rand_train([1, 10, 10, 10, 1], np.array([Data.index]).T, Data.values, gens=30, how="lin_rect")  
2 reg[1].plot.line(title = "root mean squared error by generation")
```

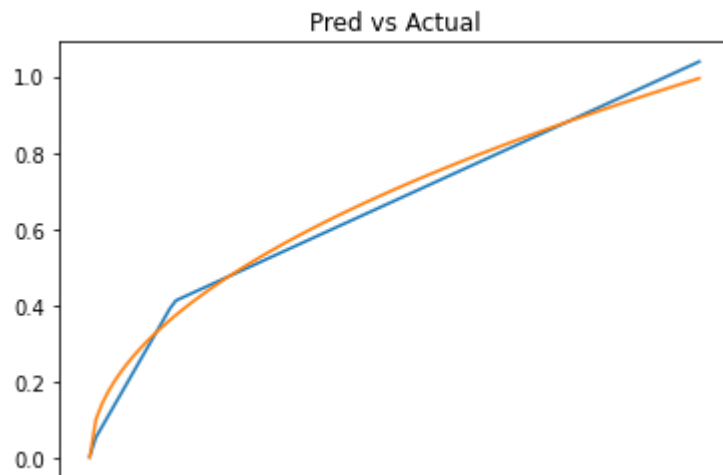
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f409ae04450>
```



```
1 reg[0].plot.line()  
2 Data.plot.line(title = "Pred vs Actual")
```



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f409ad33850>
```



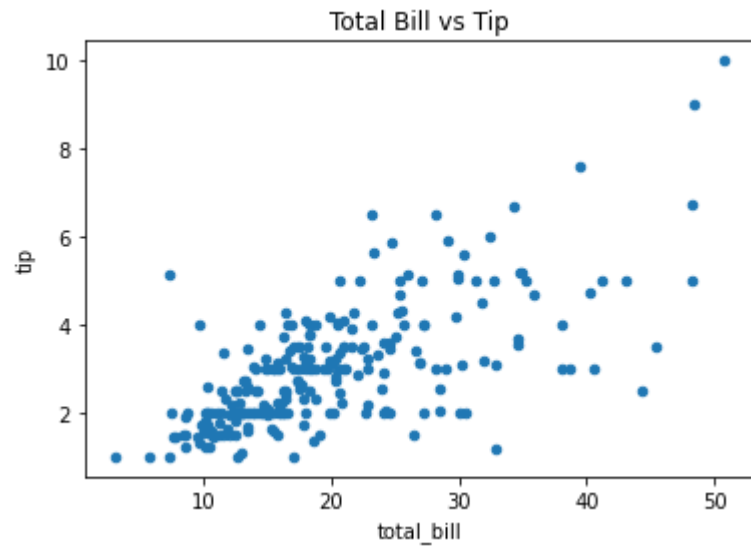
```
1 tips = pd.read_csv("https://dlsun.github.io/pods/data/"+ "tips.csv")
2 tips
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

244 rows × 7 columns

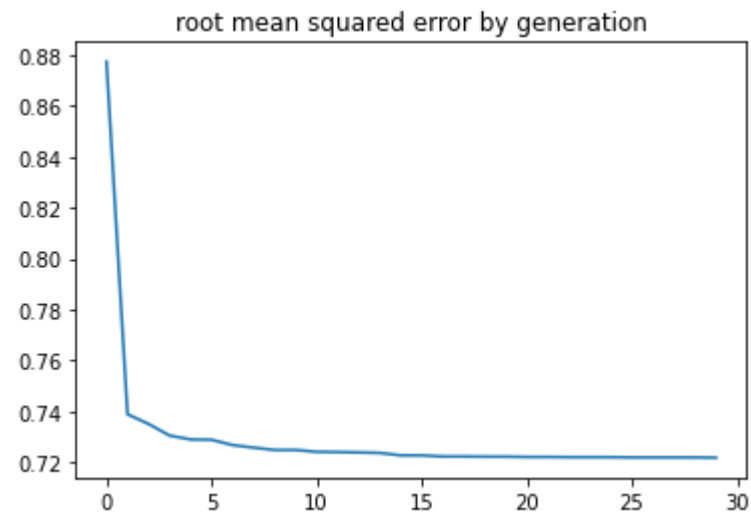
```
1 tips.plot.scatter(x="total_bill", y='tip', title = "Total Bill vs Tip")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f409c9bd850>
```



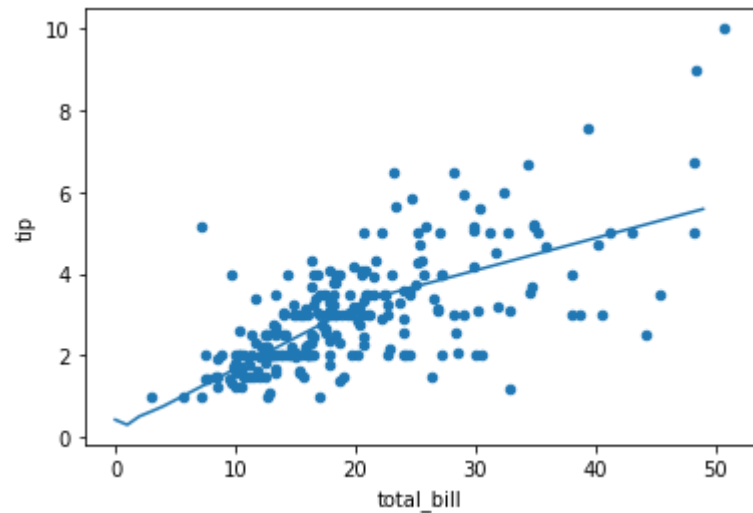
```
1 reg = rand_train([1, 10, 10, 10, 1], tips[["total_bill"]], tips["tip"], gens=30, how="lin_rect")
2 reg[1].plot.line(title = "root mean squared error by generation")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f409bde5850>
```



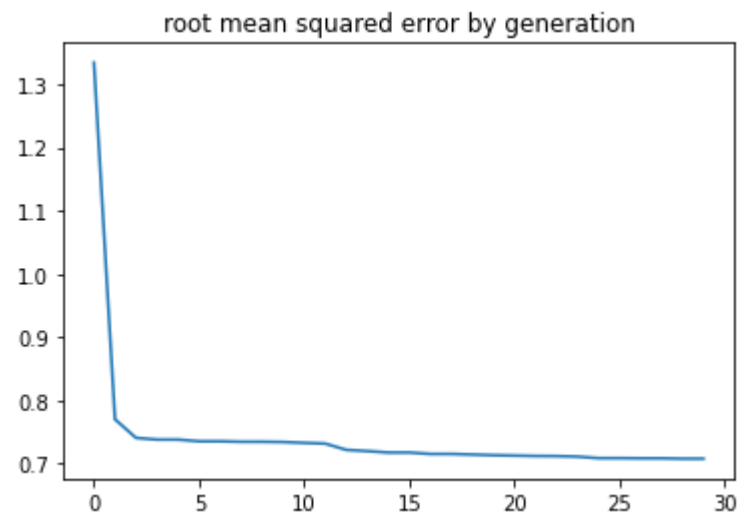
```
1 pred = pd.Series(reg[2].f_pass(pd.DataFrame(range(50))).T[0])
2 tips.plot.scatter(x="total_bill", y='tip')
3 pred.plot.line()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f409bdf41d0>



```
1 reg = rand_train([2, 10, 20, 10, 1], tips[["total_bill", "size"]], tips["tip"], gens=30, how="lin_rect")
2 reg[1].plot.line(title = "root mean squared error by generation")
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f409bcb6750>



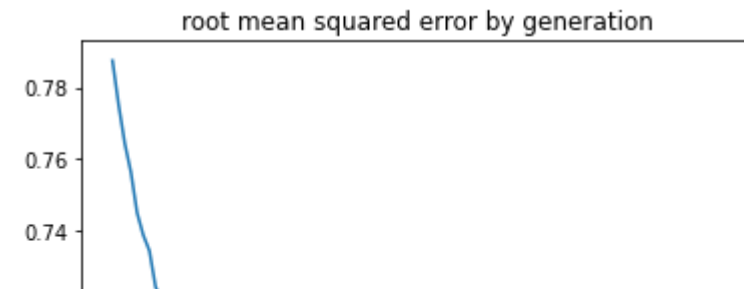

```
1 new = tips[["total_bill", "size"]].merge(pd.get_dummies(tips["sex"]), left_index=True, right_index=True)
2 new
```

	total_bill	size	Female	Male
0	16.99	2	1	0
1	10.34	3	0	1
2	21.01	3	0	1
3	23.68	2	0	1
4	24.59	4	1	0
...
239	29.03	3	0	1
240	27.18	2	1	0
241	22.67	2	0	1
242	17.82	2	0	1
243	18.78	2	1	0

244 rows × 4 columns

```
1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 X_train_st = scaler.fit_transform(new)
4 a = rand_train(x=pd.DataFrame(X_train_st), y=tips["tip"], shape=[4, 30, 20, 10, 1], gens=100, how="sigmoid")
5 a[1].plot.line(title = "root mean squared error by generation")
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f409c425710>



```
1 print("Root Mean Squared Error on Training data for my method: " + str(a[1][99]))+
2 "\nmy predictions were on average 67.7 cents away from the correct tip")
```

```
Root Mean Squared Error on Training data for my method: 0.6771494368216777
my predictions were on average 67.7 cents away from the correct tip
```

```
1 from sklearn.neighbors import KNeighborsRegressor
2
3 # Standardize the training and test data
4 scaler = StandardScaler()
5 X_train_st = scaler.fit_transform(new)
6 y_train = tips["tip"]
7
8 # Fit k-nearest neighbors
9 model = KNeighborsRegressor(n_neighbors=8)
10 model.fit(X=X_train_st, y=y_train)
11 "Root Mean Squared Error on Training data for KnearestNeighbors: " + str(np.sqrt((model.predict(X=X_train_st)-tips["tip"]
```

```
'Root Mean Squared Error on Training data for KnearestNeighbors: 0.7015727459016
398'
```

```
1 from sklearn.linear_model import LinearRegression
2
3 # Standardize the training and test data
4 scaler = StandardScaler()
5 X_train_st = scaler.fit_transform(new)
6 y_train = tips["tip"]
7
8 # Fit k-nearest neighbors
9 model = LinearRegression()
```

```
9 model = LinearRegression()  
10 model.fit(X=X_train_st, y=y_train)  
11 "Root Mean Squared Error on Training data for LinearRegression: " + str(np.sqrt((model.predict(X=X_train_st)-tips["tip"])
```

```
'Root Mean Squared Error on Training data for LinearRegression: 0.73871539121287  
94'
```