In [1]:

```
import pandas as pd

url = "SquareFeet_Data.csv"
data = pd.read_csv(url)
```

In [2]:

```
#convert sqft to m^2

data["SquareFeet"] = data["SquareFeet"] * 0.092903
data['price'] = data['price'] / 1000
```

In [3]:

```
x = data["SquareFeet"]
y = data["price"]
```

In [4]:

```
from sklearn.model_selection import train_test_split
import torch

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=
x_train = x_train.to_numpy()
x_test = x_test.to_numpy()
y_train = y_train.to_numpy()
y_test = y_test.to_numpy()

x_train = torch.from_numpy(x_train)
x_test = torch.from_numpy(x_test)
y_train = torch.from_numpy(y_train)
y_test = torch.from_numpy(y_test)
```

In [5]:

```
from torch import nn
```

In [6]:

```
class LinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear_layer = nn.Linear(in_features=1,out_features=1,dtype=torch.floa

    def forward(self,x):
        return self.linear_layer(x)
```

In [7]:

```
torch.manual_seed(87)
model = LinearRegressionModel()
model.state_dict()
```

Out[7]:

```
OrderedDict([('linear_layer.weight', tensor([[-0.9419]], dtype=torch.float64)),
             ('linear_layer.bias', tensor([0.2761], dtype=torch.float64))])
```

In [8]:

```
x_train.reshape(-1,1).shape,y_train.shape
```

Out[8]:

```
(torch.Size([26, 1]), torch.Size([26]))
```

In [9]:

```
cost_fn = nn.MSELoss() #https://docs.pytorch.org/docs/stable/generated/torch.nn.MSE
```

In [10]:

```
x_train = x_train.reshape(-1,1) #reshape x_train and y_train to tally shape
y_train = y_train.reshape(-1,1)
```

```
x_test = x_test.reshape(-1,1)
y_test = y_test.reshape(-1,1)
```

In [11]:
```
# y_pred = w*x +b
# def compute_cost(x,y,w,b):
#     y_pred = w*x + b
#     cost = (y - y_pred)**2
#     cost = cost.sum()/len(x)

#     return cost
```

In [12]:
```
y_pred = model(x_train)
cost = cost_fn(y_pred,y_train)
print(model.state_dict()) #before
print(cost) #before

optimizer = torch.optim.SGD(model.parameters(), lr=0.0000001) #https://docs.pytorch
optimizer.zero_grad()
cost.backward()
optimizer.step()

y_pred = model(x_train)
cost = cost_fn(y_pred,y_train)
print(model.state_dict()) #after
print(cost) #after
```

```
OrderedDict([('linear_layer.weight', tensor([[-0.9419]], dtype=torch.float64)), ('li
near_layer.bias', tensor([0.2761], dtype=torch.float64))])
tensor(194676.7224, dtype=torch.float64, grad_fn=<MseLossBackward0>)
OrderedDict([('linear_layer.weight', tensor([[-0.9339]], dtype=torch.float64)), ('li
near_layer.bias', tensor([0.2761], dtype=torch.float64))])
tensor(194036.3670, dtype=torch.float64, grad_fn=<MseLossBackward0>)
```

In [13]:
```
#Loop
epochs = 100

train_cost_hist = []
test_cost_hist = []

for epoch in range(epochs):
    model.train() # to indicate now is in train phase
    y_pred = model(x_train)
    train_cost = cost_fn(y_pred,y_train)
    train_cost_hist.append(train_cost.detach().numpy()) # .detach().numpy() is conv

    optimizer.zero_grad()
    train_cost.backward()
    optimizer.step()

    model.eval() # to indicate now is in test phase
    with torch.inference_mode():
        test_pred = model(x_test)
        test_cost = cost_fn(test_pred,y_test)
        test_cost_hist.append(test_cost.detach().numpy())
    if epoch%10==0:
        print(f"{epoch:5} - train_cost : {train_cost: .4e} : test_cost : {test_cost
```

```
    0 - train_cost :  1.9404e+05 : test_cost :  1.8902e+05
   10 - train_cost :  1.8776e+05 : test_cost :  1.8255e+05
   20 - train_cost :  1.8169e+05 : test_cost :  1.7631e+05
   30 - train_cost :  1.7584e+05 : test_cost :  1.7029e+05
   40 - train_cost :  1.7018e+05 : test_cost :  1.6450e+05
   50 - train_cost :  1.6473e+05 : test_cost :  1.5891e+05
   60 - train_cost :  1.5946e+05 : test_cost :  1.5352e+05
   70 - train_cost :  1.5437e+05 : test_cost :  1.4833e+05
   80 - train_cost :  1.4946e+05 : test_cost :  1.4333e+05
   90 - train_cost :  1.4472e+05 : test_cost :  1.3850e+05
```

In [14]:

```
model.state_dict()
```

Out[14]:

```
OrderedDict([('linear_layer.weight', tensor([[-0.2005]], dtype=torch.float64)),
             ('linear_layer.bias', tensor([0.2833], dtype=torch.float64))])
```

In [15]:

```
#Save
torch.save(obj = model.state_dict(), f = "pytorch_linear_regression_result.pth")
```

Train with GPU

In [16]:

```
# Load
model_new1 = LinearRegressionModel()
model_new1.state_dict() #before Load
```

Out[16]:

```
OrderedDict([('linear_layer.weight', tensor([[-0.2918]], dtype=torch.float64)),
             ('linear_layer.bias', tensor([-0.2794], dtype=torch.float64))])
```

In [17]:

```
#Load
model_new1.load_state_dict(torch.load(f = "pytorch_linear_regression_result.pth"))
```

Out[17]:

```
<All keys matched successfully>
```

In [18]:

```
#after Load
model_new1.state_dict()
```

Out[18]:

```
OrderedDict([('linear_layer.weight', tensor([[-0.2005]], dtype=torch.float64)),
             ('linear_layer.bias', tensor([0.2833], dtype=torch.float64))])
```

In [19]:

```
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

Out[19]:

```
'cuda'
```

In [20]:

```
model_new1 = model_new1.to(device)
model_new1.linear_layer.weight.device
```

Out[20]:

```
device(type='cuda', index=0)
```

In [21]:

```
x_train = x_train.to(device)
x_test = x_test.to(device)
```

```
y_train = y_train.to(device)
y_test = x_test.to(device)
```

In [22]:
```
cost_fn = nn.MSELoss()

y_pred = model_new1(x_train) #before
cost = cost_fn(y_pred, y_train) #before
print(model_new1.state_dict()) #before
print(cost) #before

optimizer = torch.optim.SGD(params = model_new1.parameters(),lr = 0.00000001)
optimizer.zero_grad()  #have set to zero, otherwise it will iterative
cost.backward()
optimizer.step()

y_pred = model_new1(x_train) #after
cost = cost_fn(y_pred, y_train) #after
print(model_new1.state_dict()) #after
print(cost) #after
```

```
OrderedDict([('linear_layer.weight', tensor([[-0.2005]], device='cuda:0', dtype=torc
h.float64)), ('linear_layer.bias', tensor([0.2833], device='cuda:0', dtype=torch.flo
at64))])
tensor(140144.0466, device='cuda:0', dtype=torch.float64,
       grad_fn=<MseLossBackward0>)
OrderedDict([('linear_layer.weight', tensor([[-0.1998]], device='cuda:0', dtype=torc
h.float64)), ('linear_layer.bias', tensor([0.2833], device='cuda:0', dtype=torch.flo
at64))])
tensor(140099.1198, device='cuda:0', dtype=torch.float64,
       grad_fn=<MseLossBackward0>)
```

In [23]:
```
#Loop
epochs = 10000

train_cost_hist = []
test_cost_hist = []

for epoch in range(epochs):
    model_new1.train() # to indicate now is in train phase
    y_pred = model_new1(x_train)
    train_cost = cost_fn(y_pred,y_train)
    train_cost_hist.append(train_cost.cpu().detach().numpy()) # .detach().numpy() i

    optimizer.zero_grad()
    train_cost.backward()
    optimizer.step()

    model_new1.eval() # to indicate now is in test phase
    with torch.inference_mode():
        test_pred = model_new1(x_test)
        test_cost = cost_fn(test_pred,y_test)
        test_cost_hist.append(test_cost.cpu().detach().numpy())

    if epoch%1000==0:
        print(f"{epoch:5} - train_cost : {train_cost: .4e} : test_cost : {test_cost
   0 - train_cost :  1.4010e+05 : test_cost :  1.4101e+04
1000 - train_cost :  1.0222e+05 : test_cost :  3.3357e+03
2000 - train_cost :  7.5571e+04 : test_cost :  4.3472e+01
3000 - train_cost :  5.6815e+04 : test_cost :  1.3197e+03
4000 - train_cost :  4.3617e+04 : test_cost :  5.2319e+03
5000 - train_cost :  3.4329e+04 : test_cost :  1.0513e+04
6000 - train_cost :  2.7793e+04 : test_cost :  1.6351e+04
7000 - train_cost :  2.3194e+04 : test_cost :  2.2238e+04
8000 - train_cost :  1.9958e+04 : test_cost :  2.7873e+04
9000 - train_cost :  1.7680e+04 : test_cost :  3.3091e+04
```