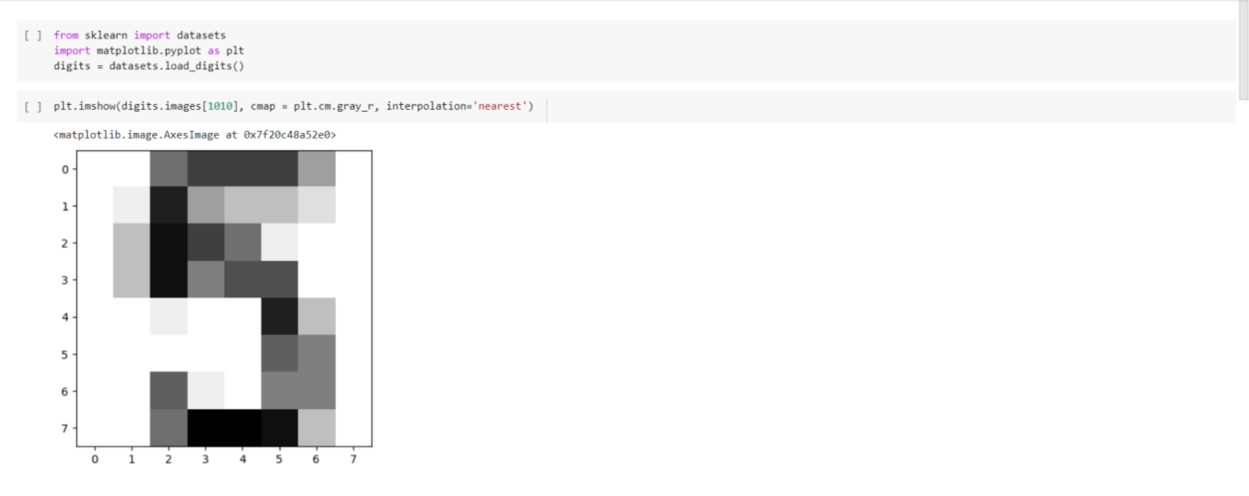
Link github: https://github.com/NguyenHuynhThaoNhu/MKTG5883.N22_19521970

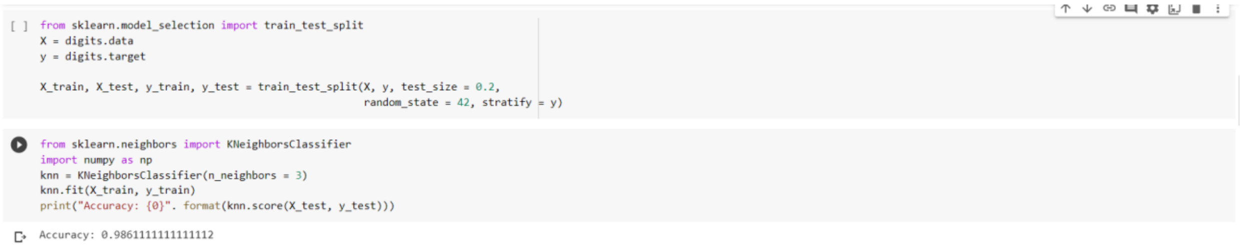
I. Classification



```
from sklearn import datasets
import matplotlib.pyplot as plt
digits = datasets.load_digits()
```

```
plt.imshow(digits.images[1010], cmap = plt.cm.gray_r, interpolation='nearest')
```

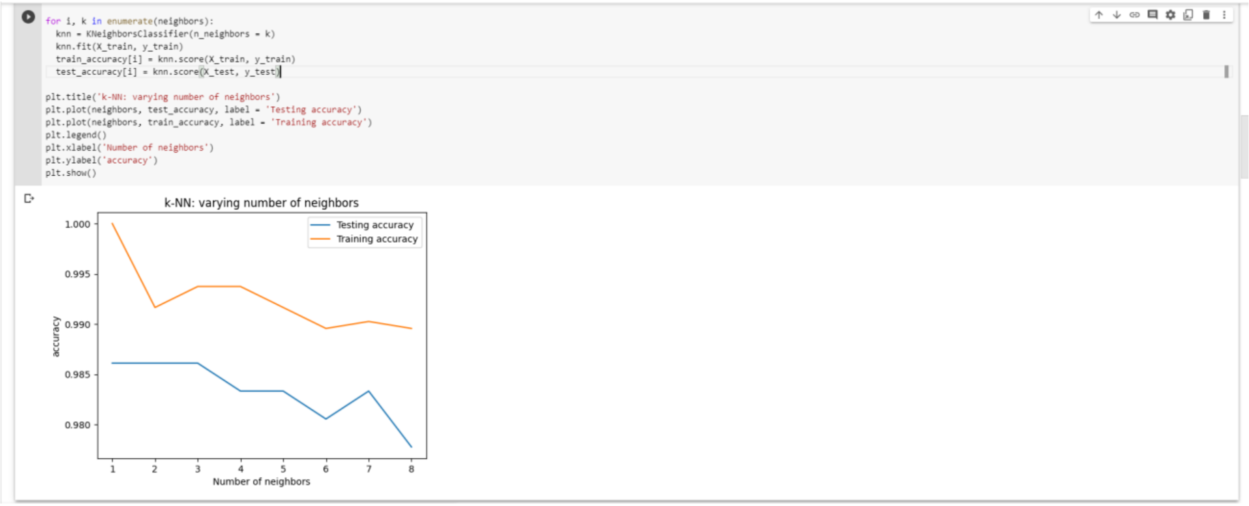
```
<matplotlib.image.AxesImage at 0x7f20c48a52e0>
```



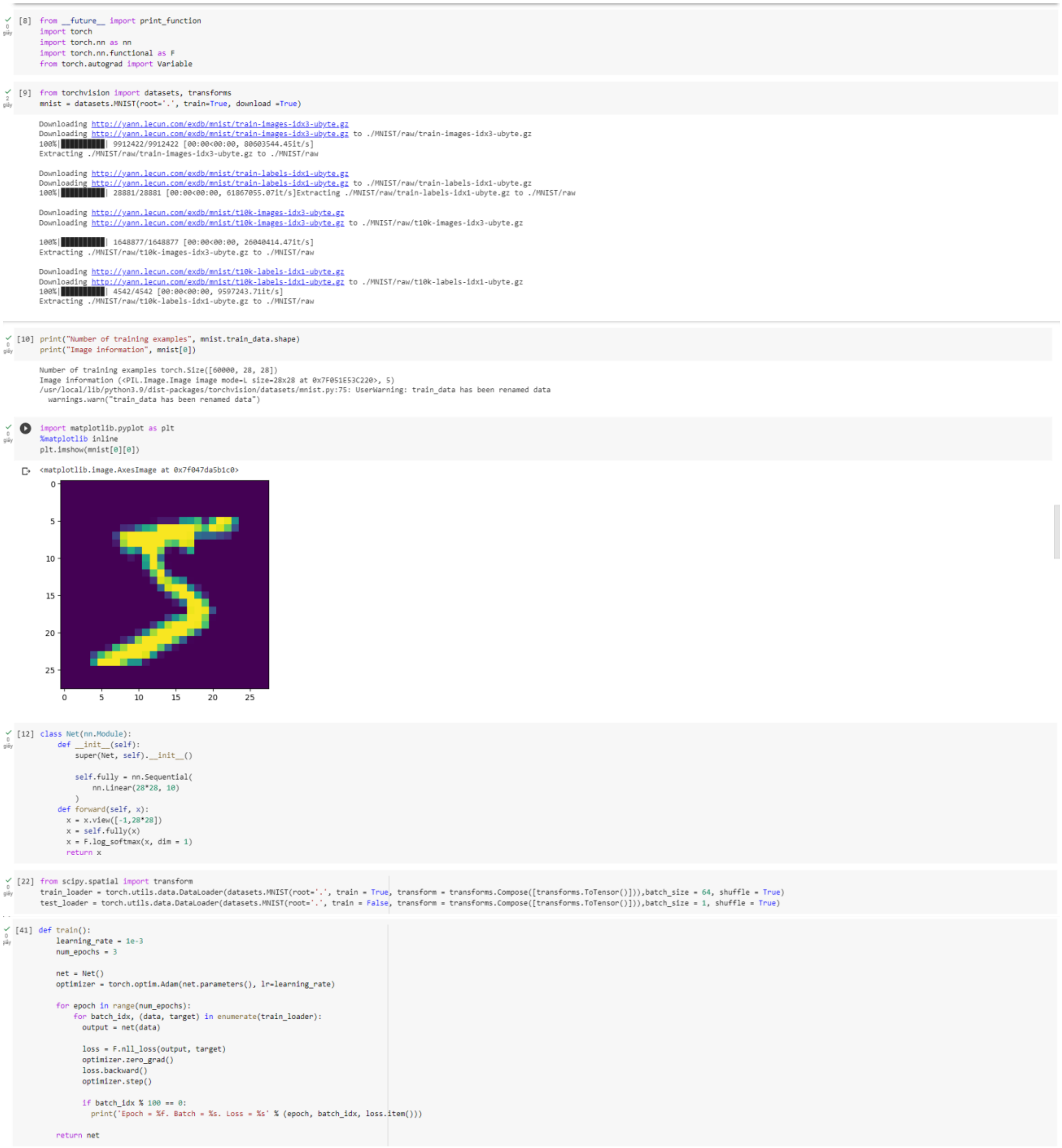
```
from sklearn.model_selection import train_test_split
X = digits.data
y = digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
                                                    random_state = 42, stratify = y)
```

```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
print("Accuracy: {0}". format(knn.score(X_test, y_test)))
```

```
Accuracy: 0.9861111111111112
```



```
for i, k in enumerate(neighbors):
  knn = KNeighborsClassifier(n_neighbors = k)
  knn.fit(X_train, y_train)
  train_accuracy[i] = knn.score(X_train, y_train)
  test_accuracy[i] = knn.score(X_test, y_test)

plt.title('k-NN: varying number of neighbors')
plt.plot(neighbors, test_accuracy, label = 'Testing accuracy')
plt.plot(neighbors, train_accuracy, label = 'Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('accuracy')
plt.show()
```



```
[8]  from __future__ import print_function
     import torch
     import torch.nn as nn
     import torch.nn.functional as F
     from torch.autograd import Variable
```

```
[9]  from torchvision import datasets, transforms
     mnist = datasets.MNIST(root='.', train=True, download =True)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to ./MNIST/raw/train-images-idx3-ubyte.gz
100%|██████████| 9912422/9912422 [00:00<00:00, 80603544.45it/s]
Extracting ./MNIST/raw/train-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to ./MNIST/raw/train-labels-idx1-ubyte.gz
100%|██████████| 28881/28881 [00:00<00:00, 61867055.07it/s]Extracting ./MNIST/raw/train-labels-idx1-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to ./MNIST/raw/t10k-images-idx3-ubyte.gz

100%|██████████| 1648877/1648877 [00:00<00:00, 26040414.47it/s]
Extracting ./MNIST/raw/t10k-images-idx3-ubyte.gz to ./MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to ./MNIST/raw/t10k-labels-idx1-ubyte.gz
100%|██████████| 4542/4542 [00:00<00:00, 9597243.71it/s]
Extracting ./MNIST/raw/t10k-labels-idx1-ubyte.gz to ./MNIST/raw
```

```
[10] print("Number of training examples", mnist.train_data.shape)
     print("Image information", mnist[0])
```

```
Number of training examples torch.Size([60000, 28, 28])
Image information (<PIL.Image.Image image mode=L size=28x28 at 0x7F051E53C220>, 5)
/usr/local/lib/python3.9/dist-packages/torchvision/datasets/mnist.py:75: UserWarning: train_data has been renamed data
  warnings.warn("train_data has been renamed data")
```

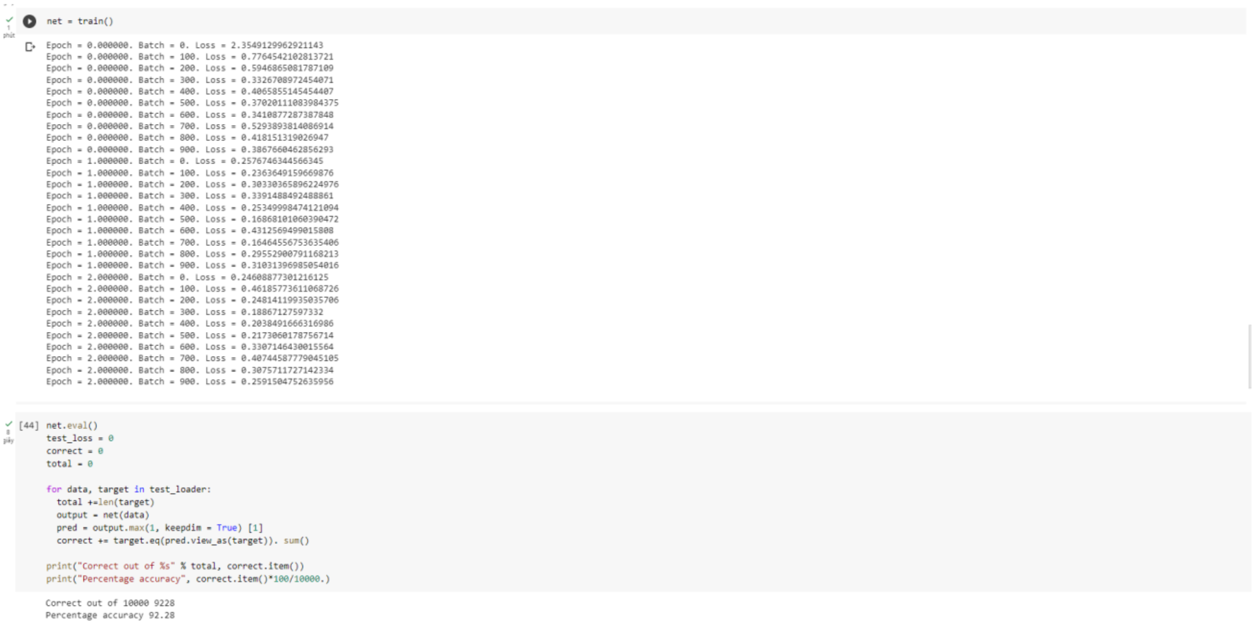
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(mnist[0][0])
```

```
<matplotlib.image.AxesImage at 0x7f047da5b1c0>
```

```
[12] class Net(nn.Module):
         def __init__(self):
             super(Net, self).__init__()

             self.fully = nn.Sequential(
                 nn.Linear(28*28, 10)
             )
         def forward(self, x):
           x = x.view([-1,28*28])
           x = self.fully(x)
           x = F.log_softmax(x, dim = 1)
           return x
```

```
[22] from scipy.spatial import transform
     train_loader = torch.utils.data.DataLoader(datasets.MNIST(root='.', train = True, transform = transforms.Compose([transforms.ToTensor()])),batch_size = 64, shuffle = True)
     test_loader = torch.utils.data.DataLoader(datasets.MNIST(root='.', train = False, transform = transforms.Compose([transforms.ToTensor()])),batch_size = 1, shuffle = True)
```

```
[41] def train():
         learning_rate = 1e-3
         num_epochs = 3

         net = Net()
         optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)

         for epoch in range(num_epochs):
             for batch_idx, (data, target) in enumerate(train_loader):
               output = net(data)

               loss = F.nll_loss(output, target)
               optimizer.zero_grad()
               loss.backward()
               optimizer.step()

               if batch_idx % 100 == 0:
                 print('Epoch = %f. Batch = %s. Loss = %s' % (epoch, batch_idx, loss.item()))

         return net
```



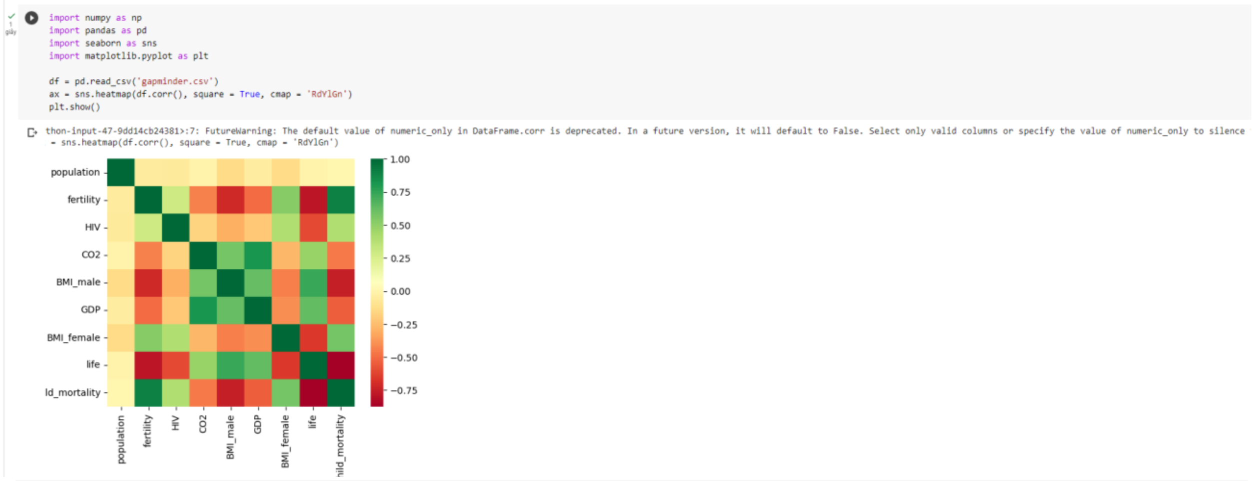
```
net = train()
```

```
Epoch = 0.000000. Batch = 0. Loss = 2.3549129962921143
Epoch = 0.000000. Batch = 100. Loss = 0.7764542102813721
Epoch = 0.000000. Batch = 200. Loss = 0.5946865081787109
Epoch = 0.000000. Batch = 300. Loss = 0.3326708972454071
Epoch = 0.000000. Batch = 400. Loss = 0.4065855145454407
Epoch = 0.000000. Batch = 500. Loss = 0.37020111083984375
Epoch = 0.000000. Batch = 600. Loss = 0.3410877287387848
Epoch = 0.000000. Batch = 700. Loss = 0.5293893814086914
Epoch = 0.000000. Batch = 800. Loss = 0.418151319026947
Epoch = 0.000000. Batch = 900. Loss = 0.3867660462856293
Epoch = 1.000000. Batch = 0. Loss = 0.2576746344566345
Epoch = 1.000000. Batch = 100. Loss = 0.2363649159669876
Epoch = 1.000000. Batch = 200. Loss = 0.30330365896224976
Epoch = 1.000000. Batch = 300. Loss = 0.3391488492488861
Epoch = 1.000000. Batch = 400. Loss = 0.25349998474121094
Epoch = 1.000000. Batch = 500. Loss = 0.16868101060390472
Epoch = 1.000000. Batch = 600. Loss = 0.4312569499015808
Epoch = 1.000000. Batch = 700. Loss = 0.16464556753635406
Epoch = 1.000000. Batch = 800. Loss = 0.29552900791168213
Epoch = 1.000000. Batch = 900. Loss = 0.31031396985054016
Epoch = 2.000000. Batch = 0. Loss = 0.24608877301216125
Epoch = 2.000000. Batch = 100. Loss = 0.46185773611068726
Epoch = 2.000000. Batch = 200. Loss = 0.24814119935035706
Epoch = 2.000000. Batch = 300. Loss = 0.18867127597332
Epoch = 2.000000. Batch = 400. Loss = 0.2038491666316986
Epoch = 2.000000. Batch = 500. Loss = 0.2173060178756714
Epoch = 2.000000. Batch = 600. Loss = 0.3307146430015564
Epoch = 2.000000. Batch = 700. Loss = 0.40744587779045105
Epoch = 2.000000. Batch = 800. Loss = 0.3075711727142334
Epoch = 2.000000. Batch = 900. Loss = 0.2591504752635956
```

```
[44] net.eval()
     test_loss = 0
     correct = 0
     total = 0

     for data, target in test_loader:
       total +=len(target)
       output = net(data)
       pred = output.max(1, keepdim = True) [1]
       correct += target.eq(pred.view_as(target)). sum()

     print("Correct out of %s" % total, correct.item())
     print("Percentage accuracy", correct.item()*100/10000.)
```

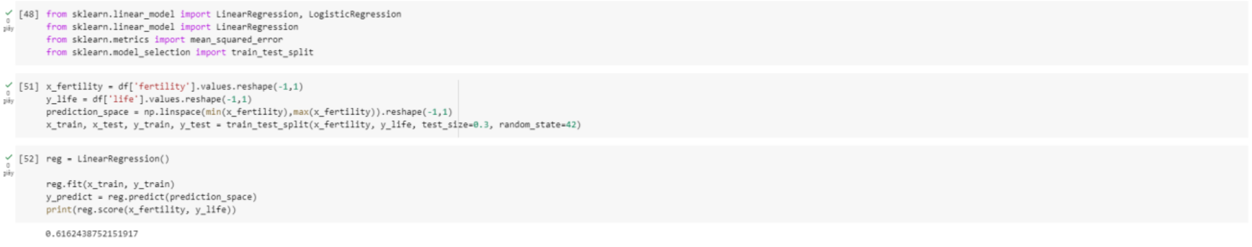
```
Correct out of 10000 9228
Percentage accuracy 92.28
```

II. Linear Regression



```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('gapminder.csv')
ax = sns.heatmap(df.corr(), square = True, cmap = 'RdYlGn')
plt.show()
```

```
thon-input-47-9dd14cb24381>:7: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence
  = sns.heatmap(df.corr(), square = True, cmap = 'RdYlGn')
```



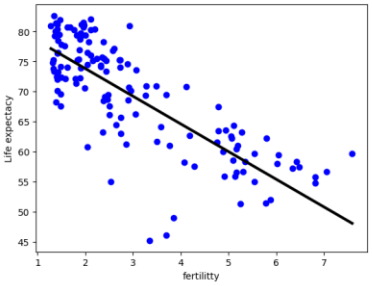
```
[48] from sklearn.linear_model import LinearRegression, LogisticRegression
     from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_squared_error
     from sklearn.model_selection import train_test_split
```

```
[51] x_fertility = df['fertility'].values.reshape(-1,1)
     y_life = df['life'].values.reshape(-1,1)
     prediction_space = np.linspace(min(x_fertility),max(x_fertility)).reshape(-1,1)
     x_train, x_test, y_train, y_test = train_test_split(x_fertility, y_life, test_size=0.3, random_state=42)
```

```
[52] reg = LinearRegression()

     reg.fit(x_train, y_train)
     y_predict = reg.predict(prediction_space)
     print(reg.score(x_fertility, y_life))
```

```
0.6162438752151917
```

```
plt.scatter(x_fertility, y_life, color = 'blue')
plt.plot(prediction_space, y_predict, color='black', linewidth = 3)
plt.ylabel('Life expectacy')
plt.xlabel('fertilitty')
plt.show()
```



[61]

```
features = pd.read_csv('gapminder.csv')
df = pd.read_csv('gapminder.csv')
del features['life']
del features['Region']

y_lide = df['life'].values.reshape(-1,1)

#Create training and test sets
x_train, x_test, y_train, y_test = train_test_split(features, y_life,test_size = 0.3, random_state=42)

#Create the regeression model: reg_all
reg_all = LinearRegression()

#Fitteh regression to the training data
reg_all.fit(x_train, y_train )

#Print accuracy
print(reg_all.score(features,y_life))
```

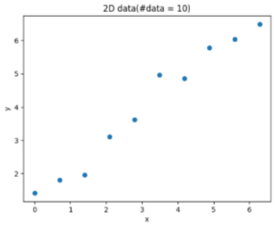
```
0.8914651485793176
```

Linear Regression using Pytorch

[62]

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

```
N = 10 #number of data points
m = .9
c = 1
x = np.linspace(0, 2 * np.pi, N)
y = m*x + c + np.random.normal(0,.3, x.shape)
plt.figure()
plt.plot(x,y,'o')
plt.xlabel('x')
plt.ylabel('y')
plt.title('2D data(#data = %d)' % N)
plt.show()
```



[77]

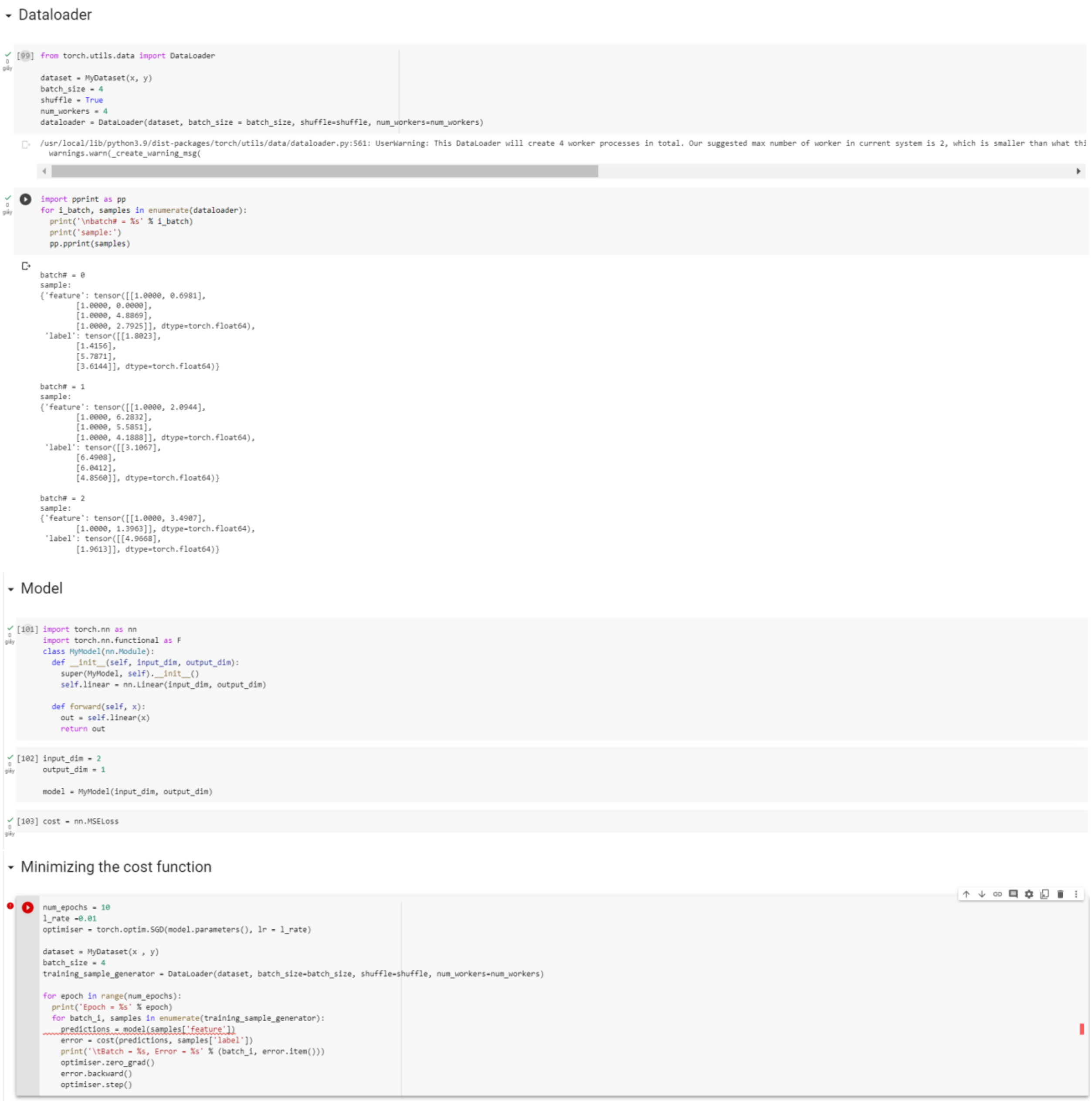
```
import torch
```

Dataset

```
from torch.utils.data import Dataset

class MyDataset(Dataset):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __len__(self):
        return len(self.x)

    def __getitem__(self, idx):
        sample = {
            'feature': torch.tensor([1, self.x[idx]]),
            'label': torch.tensor([self.y[idx]])
        }
        return sample
```

```
dataset = MyDataset(x, y)

for i in range(len(dataset)):
    sample = dataset[i]
    print(i, sample['feature'], sample['label'])
```

```
0 tensor([1., 0.], dtype=torch.float64) tensor([1.4156], dtype=torch.float64)
1 tensor([1.0000, 0.6981], dtype=torch.float64) tensor([1.8023], dtype=torch.float64)
2 tensor([1.0000, 1.3963], dtype=torch.float64) tensor([1.9613], dtype=torch.float64)
3 tensor([1.0000, 2.0944], dtype=torch.float64) tensor([3.1067], dtype=torch.float64)
4 tensor([1.0000, 2.7925], dtype=torch.float64) tensor([3.6144], dtype=torch.float64)
5 tensor([1.0000, 3.4907], dtype=torch.float64) tensor([4.9668], dtype=torch.float64)
6 tensor([1.0000, 4.1888], dtype=torch.float64) tensor([4.8560], dtype=torch.float64)
7 tensor([1.0000, 4.8869], dtype=torch.float64) tensor([5.7871], dtype=torch.float64)
8 tensor([1.0000, 5.5851], dtype=torch.float64) tensor([6.0412], dtype=torch.float64)
9 tensor([1.0000, 6.2832], dtype=torch.float64) tensor([6.4908], dtype=torch.float64)
```



Dataloader

```
[99] from torch.utils.data import DataLoader

     dataset = MyDataset(x, y)
     batch_size = 4
     shuffle = True
     num_workers = 4
     dataloader = DataLoader(dataset, batch_size = batch_size, shuffle=shuffle, num_workers=num_workers)
```

```
/usr/local/lib/python3.9/dist-packages/torch/utils/data/dataloader.py:561: UserWarning: This DataLoader will create 4 worker processes in total. Our suggested max number of worker in current system is 2, which is smaller than what thi
  warnings.warn(_create_warning_msg(
```

```
import pprint as pp
for i_batch, samples in enumerate(dataloader):
  print('\nbatch# = %s' % i_batch)
  print('sample:')
  pp.pprint(samples)
```

```
batch# = 0
sample:
{'feature': tensor([[1.0000, 0.6981],
        [1.0000, 0.0000],
        [1.0000, 4.8869],
        [1.0000, 2.7925]], dtype=torch.float64),
 'label': tensor([[1.8023],
        [1.4156],
        [5.7871],
        [3.6144]], dtype=torch.float64)}

batch# = 1
sample:
{'feature': tensor([[1.0000, 2.0944],
        [1.0000, 6.2832],
        [1.0000, 5.5851],
        [1.0000, 4.1888]], dtype=torch.float64),
 'label': tensor([[3.1067],
        [6.4908],
        [6.0412],
        [4.8560]], dtype=torch.float64)}

batch# = 2
sample:
{'feature': tensor([[1.0000, 3.4907],
        [1.0000, 1.3963]], dtype=torch.float64),
 'label': tensor([[4.9668],
        [1.9613]], dtype=torch.float64)}
```

Model

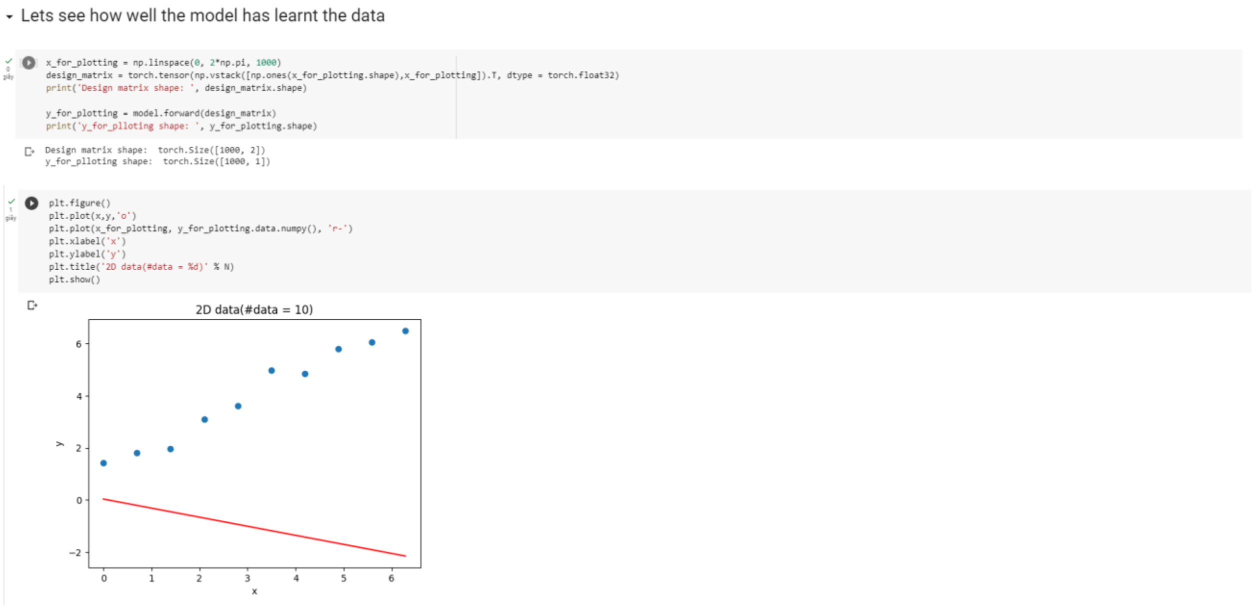
```
[101] import torch.nn as nn
      import torch.nn.functional as F
      class MyModel(nn.Module):
        def __init__(self, input_dim, output_dim):
          super(MyModel, self).__init__()
          self.linear = nn.Linear(input_dim, output_dim)

        def forward(self, x):
          out = self.linear(x)
          return out
```

```
[102] input_dim = 2
      output_dim = 1

      model = MyModel(input_dim, output_dim)
```

```
[103] cost = nn.MSELoss
```

Minimizing the cost function

```
num_epochs = 10
l_rate =0.01
optimiser = torch.optim.SGD(model.parameters(), lr = l_rate)

dataset = MyDataset(x , y)
batch_size = 4
training_sample_generator = DataLoader(dataset, batch_size=batch_size, shuffle=shuffle, num_workers=num_workers)

for epoch in range(num_epochs):
  print('Epoch = %s' % epoch)
  for batch_i, samples in enumerate(training_sample_generator):
    predictions = model(samples['feature'])
    error = cost(predictions, samples['label'])
    print('\tBatch = %s, Error = %s' % (batch_i, error.item()))
    optimiser.zero_grad()
    error.backward()
    optimiser.step()
```



Lets see how well the model has learnt the data

```
x_for_plotting = np.linspace(0, 2*np.pi, 1000)
design_matrix = torch.tensor(np.vstack([np.ones(x_for_plotting.shape),x_for_plotting]).T, dtype = torch.float32)
print('Design matrix shape: ', design_matrix.shape)

y_for_plotting = model.forward(design_matrix)
print('y_for_plloting shape: ', y_for_plotting.shape)
```

```
Design matrix shape:  torch.Size([1000, 2])
y_for_plloting shape:  torch.Size([1000, 1])
```

```
plt.figure()
plt.plot(x,y,'o')
plt.plot(x_for_plotting, y_for_plotting.data.numpy(), 'r-')
plt.xlabel('x')
plt.ylabel('y')
plt.title('2D data(#data = %d)' % N)
plt.show()
```