HW4_2082014_경제_김은심

```
!pip install ISLP
```

```
Collecting lightning-utilities>=0.10.0 (from pytorch-lightning->ISLP)
  Downloading lightning_utilities-0.11.7-py3-none-any.whl.metadata (5.2 kB)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (fro
Collecting interface-meta>=1.2.0 (from formulaic>=0.2.2->lifelines->ISLP)
  Downloading interface_meta-1.3.0-py3-none-any.whl.metadata (6.7 kB)
Requirement already satisfied: wrapt>=1.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.1
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from p
Requirement already satisfied: python-utils>=3.8.1 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: yarl<2.0,>=1.12.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: idna>=2.0 in /usr/local/lib/python3.10/dist-packages (
Downloading ISLP-0.4.0-py3-none-any.whl (3.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 32.1 MB/s eta 0:00:00
Downloading lifelines-0.29.0-py3-none-any.whl (349 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 349.3/349.3 kB 21.7 MB/s eta 0:00:00
Downloading pygam-0.9.1-py3-none-any.whl (522 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 522.0/522.0 kB 26.3 MB/s eta 0:00:00
Downloading scipy-1.11.4-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 36.4/36.4 MB 21.0 MB/s eta 0:00:00
Downloading pytorch_lightning-2.4.0-py3-none-any.whl (815 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 815.2/815.2 kB 29.4 MB/s eta 0:00:00
Downloading torchmetrics-1.4.3-py3-none-any.whl (869 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 869.5/869.5 kB 35.7 MB/s eta 0:00:00
Downloading formulaic-1.0.2-py3-none-any.whl (94 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.5/94.5 kB 6.1 MB/s eta 0:00:00
Downloading lightning_utilities-0.11.7-py3-none-any.whl (26 kB)
Downloading interface_meta-1.3.0-py3-none-any.whl (14 kB)
Building wheels for collected packages: autograd-gamma
  Building wheel for autograd-gamma (setup.py) ... done
  Created wheel for autograd-gamma: filename=autograd_gamma-0.5.0-py3-none-any.whl si
  Stored in directory: /root/.cache/pip/wheels/25/cc/e0/ef2969164144c899fedb22b338f67
Successfully built autograd-gamma
Installing collected packages: scipy, lightning-utilities, interface-meta, autograd-g
  Attempting uninstall: scipy
    Found existing installation: scipy 1.13.1
    Uninstalling scipy-1.13.1:
      Successfully uninstalled scipy-1.13.1
Successfully installed ISLP-0.4.0 autograd-gamma-0.5.0 formulaic-1.0.2 interface-meta
```

```
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
                         summarize,
                         poly)
```

```
import ISLP
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
from matplotlib import pyplot as plt

import statsmodels.api as sm
```

```
Boston = load_data("Boston")
```

Exercise 1.1

Do the estimatsion procedure using torch.nn.Linear(1,1, bias=True). Answer the following questions.

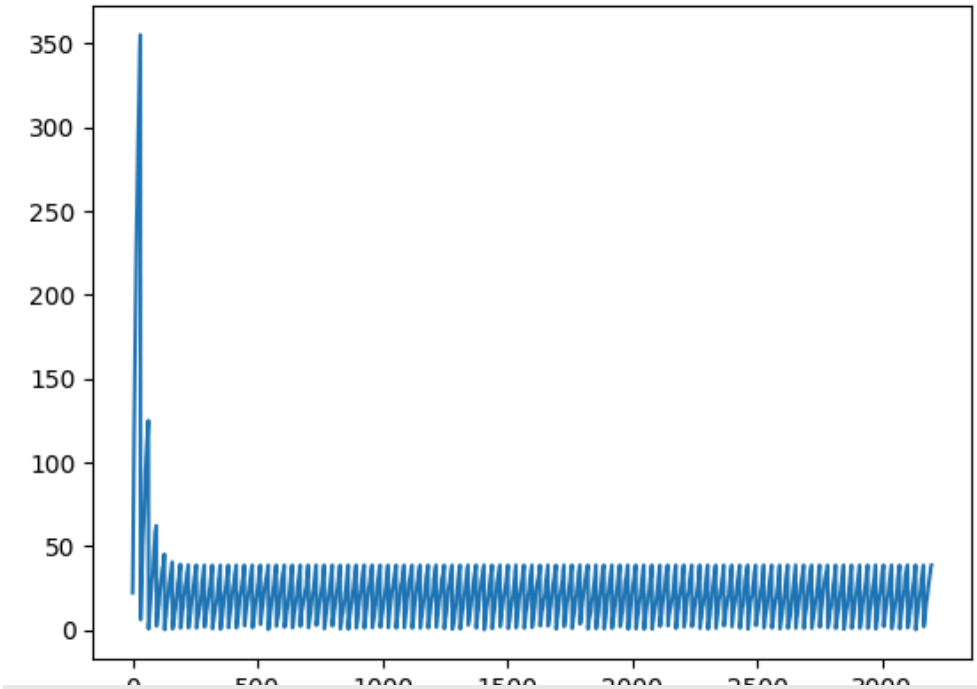
i. Calculate $\hat{Y}$ when lstats are given by 5, 10, 15, respectively.

ii. Calculate $R2$.

```
design = MS(['lstat'])
X = design.fit_transform(Boston)
y = Boston['medv']
import torch

xx = torch.tensor(np.array(X), dtype=torch.float32)
yy = torch.tensor(np.array(y).reshape([-1,1]), dtype=torch.float3



ux = torch.mean(xx[:, 1:], axis=0)
stdx = torch.std(xx[:, 1:], axis=0)
x_normal = (xx[:, 1:]-ux)/stdx # [n,1]

from torch.utils.data import Dataset, DataLoader, TensorDataset

dataset = TensorDataset(x_normal,yy)
trainloader = DataLoader(dataset=dataset, batch_size=16, shuffle=

forward = torch.nn.Linear(1, 1, bias=True) #beta: [2,1]
optimizer = torch.optim.SGD(forward.parameters(), lr=0.01)
history = []
n = len(trainloader.dataset)
epochs=100
for i in range(epochs):
```

```
  LOSS = 0
  for x, y in trainloader:
    yhat = forward(x)
    loss = torch.mean((y-yhat)**2)
    LOSS += loss*len(x)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    history.append(LOSS.item()/n)

plt.plot(history)
```

```
[<matplotlib.lines.Line2D at 0x7cbe300d4c40>]
```




```
x_1=torch.tensor([5,10,15.0],dtype=torch.float32)
ones=torch.ones([3])
xx_my=torch.stack([ones,(x_1-ux)/stdx],axis=1).reshape([-1,1])


forward(xx_my)
```

```
tensor([[15.7348],
        [29.6601],
        [15.7348],
        [24.9537],
        [15.7348],
        [20.2474]], grad_fn=<AddmmBackward0>)
```

```
yhat=forward(x_normal)
SS_reg=torch.mean((yhat-yy)**2)
SS_reg

yhat0=torch.mean(yy)
```

```
SS_total=torch.mean((yhat0-yy)**2)

SS_total
R_sqr=1-(SS_reg/SS_total)
print(R_sqr) #0.544
```

```
tensor(0.5440, grad_fn=<RsubBackward1>)
```

Exercise 2

Do the estimatsion procedure using torch.nn.Linear(2,1, bias=False).Answer the following questions.

i. Calculate $\hat{Y}$ when lstats are given by 5, 10, 15, respectively.

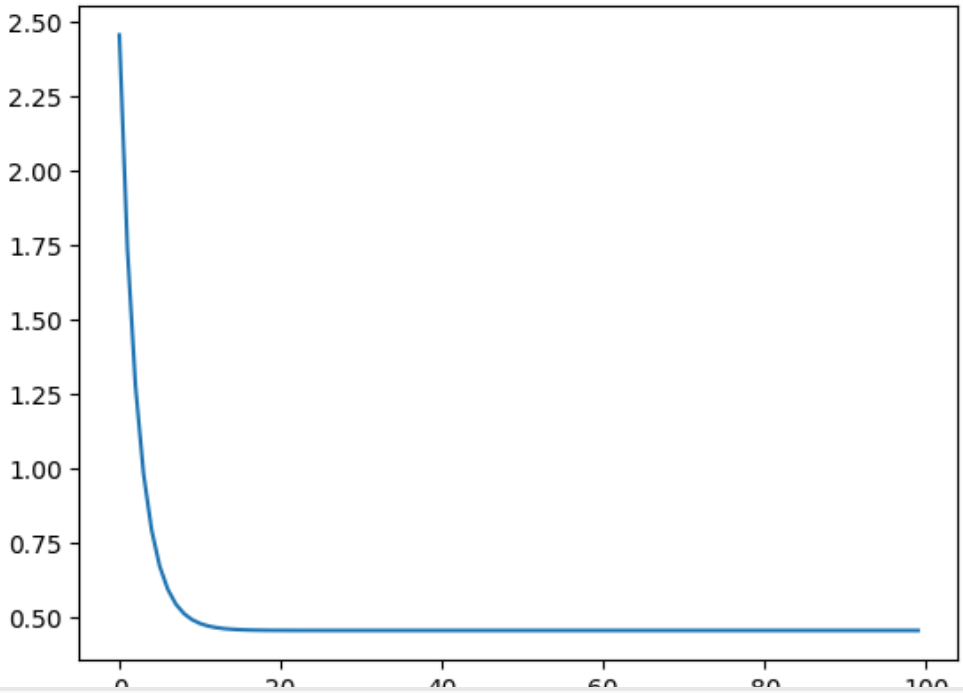
ii. Calculate $R2$.

iii. Calculate AIC.

```

design = MS(['lstat'])
X = design.fit_transform(Boston)
y = Boston['medv']
np.array(X)
xx=torch.tensor(np.array(X),dtype=torch.float32)

yy=torch.tensor(np.array(y).reshape([-1,1]),dtype=torch.float32)
#forward.weight.dtype


ux=torch.mean(xx[:,1])
stdx=torch.std(xx[:,1])
uy=torch.mean(yy)
stdy=torch.std(yy)
Y=(yy-uy)/stdy
X1=(xx[:,1]-ux)/stdx
XX=torch.stack([xx[:,0],X1],axis=1)
XX.shape
XX

xx.shape


forward=torch.nn.Linear(2,1,bias=False)#포함됨 #beta :[2,1]


#torch.nn.SDG
optimizer=torch.optim.SGD(forward.parameters(),lr=0.1)
#beta=torch.tensor([5.0,10.0,15.0],requires_grad=True)
history=[]
epochs=100
```

```
for i in range(epochs):
  Yhat=forward(XX) #560,1 Y[506]
  Loss=torch.mean((Y-Yhat)**2)
  history.append(Loss.item())

  optimizer.zero_grad()
  Loss.backward() #미분
  optimizer.step()
#i 예측
# lstats are given by 5, 10, 15

myinput=torch.tensor([[1.0,(5.0-ux)/stdx]],dtype=torch.float32)
myinput1=torch.tensor([[1.0,(10.0-ux)/stdx]],dtype=torch.float32)
myinput2=torch.tensor([[1.0,(15.0-ux)/stdx]],dtype=torch.float32)
print(forward(myinput) * stdy +uy)
print(forward(myinput1) * stdy +uy)
print(forward(myinput2) * stdy +uy)

#ii
#iii
plt.plot(history)
```

```
tensor([[29.8036]], grad_fn=<AddBackward0>)
tensor([[25.0533]], grad_fn=<AddBackward0>)
tensor([[20.3031]], grad_fn=<AddBackward0>)
[<matplotlib.lines.Line2D at 0x7cbe2dd12680>]
```




```
#R squared
yhat=forward(XX)
SS_reg=torch.mean((yhat-Y)**2)
SS_reg

yhat0=torch.mean(Y)
```

```
SS_total=torch.mean((yhat0-Y)**2)

SS_total
R_sqr=1-(SS_reg/SS_total)
print(R_sqr) #0.544

```

```
tensor(0.5441, grad_fn=<RsubBackward1>)
```

Exercise 3

Do the estimatsion procedure using torch.nn.Linear(1,1, bias=False) and an additional bias parameter defined by torch.tensor. Model itself should be the same as in Exercise 1.1 and 1.2. (Caution: However, the estimated parameter(s) should not be the exactly the same due to the randomness in the parameter initialization and batch learning.)

Answer the following questions.

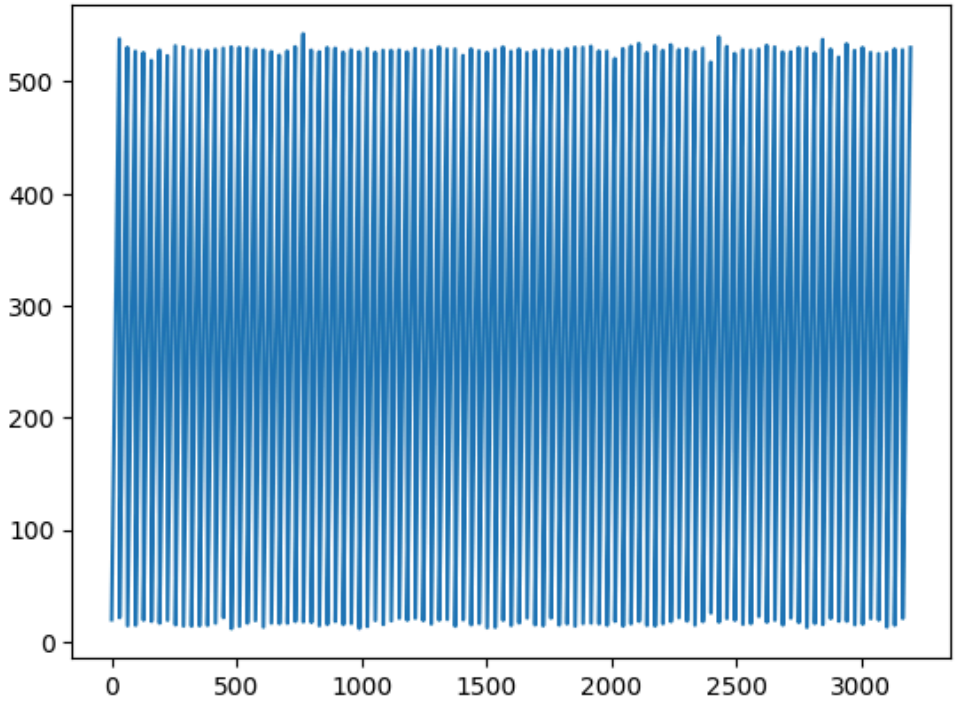
i. Calculate $\hat{Y}$ when lstats are given by 5, 10, 15, respectively.

ii. Calculate $R2$.

```
design = MS(['lstat'])
X = design.fit_transform(Boston)
y = Boston['medv']
X
xx = torch.tensor(np.array(X), dtype=torch.float32)
yy = torch.tensor(np.array(y).reshape([-1,1]), dtype=torch.float3



ux = torch.mean(xx[:, 1:], axis=0)
stdx = torch.std(xx[:, 1:], axis=0)
x_normal = (xx[:, 1:]-ux)/stdx # [n,1]

from torch.utils.data import Dataset, DataLoader, TensorDataset

dataset = TensorDataset(x_normal,yy)
trainloader = DataLoader(dataset=dataset, batch_size=16, shuffle=

forward = torch.nn.Linear(1, 1, bias=False) #beta: [1,1]
b0=torch.tensor(torch.randn([1]),requires_grad=True)

optimizer = torch.optim.SGD(forward.parameters([b0]+list(forward.
history = []

n = len(trainloader.dataset)
epochs=100
for i in range(epochs):
  LOSS = 0
```

```
  for x, y in trainloader:
    yhat = forward(x)+b0 ###
    loss = torch.mean((y-yhat)**2)
    LOSS += loss*len(x)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    history.append(LOSS.item()/n)


plt.plot(history)


yhat=forward(x_normal)+b0
SS_reg=torch.mean((yhat-yy)**2)
SS_reg

yhat0=torch.mean(yy)
SS_total=torch.mean((yhat0-yy)**2)

SS_total
1-(SS_reg/SS_total)
```

```
<ipython-input-27-46a3b57ee70c>:20: UserWarning: To copy construct from a tensor, it is r
  b0=torch.tensor(torch.randn([1]),requires_grad=True)
tensor(-5.1970, grad_fn=<RsubBackward1>)
```



2.1 exercise:

Using the standardizied Boston data answer the following questions.

(a) Using pytorch and customized layer, calculate the train MSE of the following model:

```
design=MS(['age','lstat'])
X=design.fit_transform(Boston)
y=Boston['medv']

import torch

xx = torch.tensor(np.array(X), dtype=torch.float32)#[n,3]
xx2=xx[:,1:2]**2 #lstat square
xx=torch.concat([xx,xx2],axis=1)#[n,4]: 1,lstat,age,lstat2


yy = torch.tensor(np.array(y).reshape([-1,1]), dtype=torch.float3


ux = torch.mean(xx[:, 1:], axis=0)
stdx = torch.std(xx[:, 1:], axis=0)
xx_pure = (xx[:, 1:]-ux)/stdx # [n,2]
xx_normal = torch.concat([xx[:, :1], xx_pure], axis=1)
print(xx_normal.shape)

from torch.utils.data import Dataset, DataLoader, TensorDataset
dataset = TensorDataset(xx_normal,yy)
trainloader = DataLoader(dataset=dataset, batch_size=16, shuffle=

forward=torch.nn.Linear(3,1,bias=False)# 설명변수 4개인 셈
optimizer = torch.optim.SGD(forward.parameters((forward.parameter

history = []

n = len(trainloader.dataset)
epochs=100
for i in range(epochs):
  LOSS = 0
  for x, y in trainloader:
    yhat = forward(x)
    loss = torch.mean((y-yhat)**2)
    LOSS += loss*len(x)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    history.append(LOSS.item()/n)


plt.plot(history)
```

```
torch.Size([506, 4])
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-37-072d38811368> in <cell line: 32>()
     33   LOSS = 0
     34   for x, y in trainloader:
---> 35     yhat = forward(x)
     36     loss = torch.mean((y-yhat)**2)
     37     LOSS += loss*len(x)

                                 2 frames
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/linear.py in forward(self,
input)
    115
    116     def forward(self, input: Tensor) -> Tensor:
--> 117         return F.linear(input, self.weight, self.bias)
    118
    119     def extra_repr(self) -> str:

RuntimeError: mat1 and mat2 shapes cannot be multiplied (16x4 and 3x1)
```

다음 단계: 오류 설명

Using the standardizied Boston data answer the following questions.

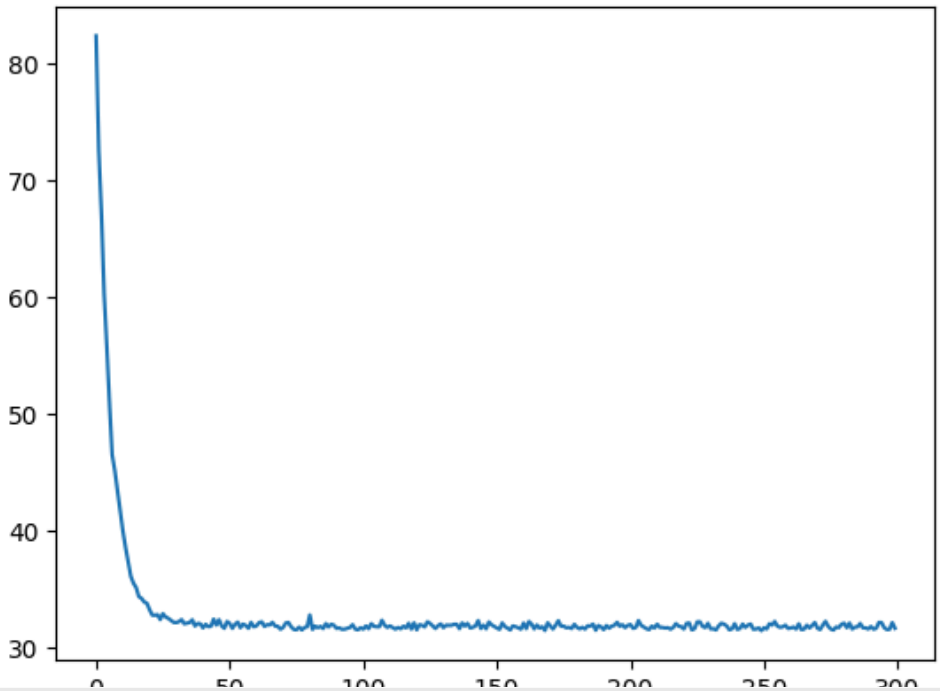
(a) Using pytorch and customized layer, calculate the train MSE of the following model:

medv ~ age + lstat

```
design=MS(['age','lstat'])
X=design.fit_transform(Boston)
Y=torch.tensor(Boston['medv'],dtype=torch.float32)
X1=torch.tensor(Boston["age"],dtype=torch.float32)
X2=torch.tensor(Boston["lstat"],dtype=torch.float32)
X=torch.stack([X1,X2],axis=1)
#normalization
ux=torch.mean(X,axis=0)#[2,](ul,u2)
sd=torch.std(X,axis=0)
x_norm=(X-ux)/sd

uy=torch.mean(Y)
y_norm=(Y-uy)/torch.std(Y)

#linear regression
class linear_regression(torch.nn.Module):
  def __init__(self, input_size, output_size): #설명변수 2개 ,반응변수

    super().__init__()
    self.bias = torch.nn.Parameter(torch.randn([output_size]))
    self.weight = torch.nn.Parameter(torch.randn([input_size,outp
  def forward(self,x):
    yhat = x @ self.weight + self.bias #W 부를 때, 셀프로 지칭
    return yhat

```

```
model = linear_regression(2,1) # model = torch.nn.Linear(2, 1, bi
print(model.weight)

dataset = TensorDataset(x_norm,y_norm) #두개의 데이터
trainloader = DataLoader(dataset=dataset, batch_size=16, shuffle=

def criterion(yhat, y):
  return torch.mean((yhat-y)**2)
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

iters = 300
history = [] #300개의 로스
#n=len(trainloader.dataset)
for epoch in range(iters):
  current_epoch_loss = 0
  for x,y in trainloader: #두번째 for 문 ,x 10개, y 10개, 3개일 때도 있

    yhat = model(x)
    loss = criterion(yhat, y)
    current_epoch_loss += loss.item()
    loss.backward() #최소화하기 위해서
    optimizer.step() #50번 일어남, decent gradient
    optimizer.zero_grad()
  history.append(current_epoch_loss)

import matplotlib.pyplot as plt
plt.plot(history)

```

```
Parameter containing:
tensor([[-0.9624],
        [-0.5362]], requires_grad=True)
[<matplotlib.lines.Line2D at 0x7cbe2dca9000>]
```



```
Yhat=model(x_norm)
#Y_normal.reshape(-1,1)
criterion(Yhat,y_norm)#MSE 구하는 방법

#R-squared
SS_reg=torch.mean((Yhat-y_norm)**2)
SS_reg

yhat0=torch.mean(y_norm)
SS_total=torch.mean((yhat0-y_norm)**2)

SS_total
```

```
tensor(0.9980)
```

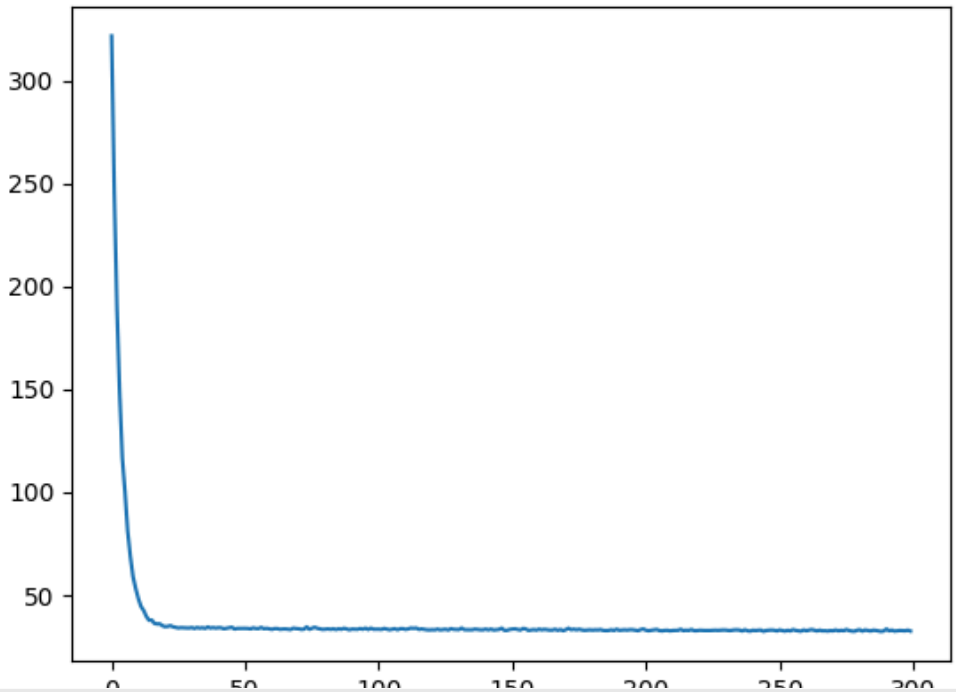
(b) Using pytorch and customized layer, calculate the train MSE of the following model:

medv ~ age + lstat + lstat**2

```
design=MS(['age','lstat'])
X=design.fit_transform(Boston)
Y=torch.tensor(Boston['medv'],dtype=torch.float32)
X1=torch.tensor(Boston["age"],dtype=torch.float32)
X2=torch.tensor(Boston["lstat"],dtype=torch.float32)
X3=torch.tensor(Boston["lstat"]**2,dtype=torch.float32)
X=torch.stack([X1,X2,X3],axis=1)
#normalization
ux=torch.mean(X,axis=0)#[3,](ul,u2)
sd=torch.std(X,axis=0)
x_norm=(X-ux)/sd

uy=torch.mean(Y)
y_norm=(Y-uy)/torch.std(Y)

model = linear_regression(3,1) # model = torch.nn.Linear(2, 1, bi
print(model.weight)

dataset = TensorDataset(x_norm,y_norm) #두개의 데이터
trainloader = DataLoader(dataset=dataset, batch_size=16, shuffle=

def criterion(yhat, y):
  return torch.mean((yhat-y)**2)
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

iters = 300
history = [] #300개의 로스
#n=len(trainloader.dataset)
for epoch in range(iters):
```

```
  current_epoch_loss = 0
  for x,y in trainloader: #두번째 for 문 ,x 10개, y 10개, 3개일 때도 있
    yhat = model(x)
    loss = criterion(yhat, y)
    current_epoch_loss += loss.item()
    loss.backward() #최소화하기 위해서
    optimizer.step() #50번 일어남, decent gradient
    optimizer.zero_grad()
  history.append(current_epoch_loss)

import matplotlib.pyplot as plt
plt.plot(history)
```

```
Parameter containing:
tensor([[-1.0802],
        [-0.4141],
        [-2.1306]], requires_grad=True)
[<matplotlib.lines.Line2D at 0x7cbe2dee80d0>]
```




```
Yhat=model(x_norm)
#Y_normal.reshape(-1,1)
print(criterion(Yhat,y_norm))#MSE 구하는 방법

#R-squared
SS_reg=torch.mean((Yhat-y_norm)**2)
SS_reg

yhat0=torch.mean(y_norm)
SS_total=torch.mean((yhat0-y_norm)**2)
```

```
11
12 SS_total ##b가 R square 더 크다, MSE는 작음
```

```
tensor(1.0110, grad_fn=<MeanBackward0>)
tensor(0.9980)
```

(d) Repeat (a) and (b) by making use of 70% of data as train set and the remaining 30% as test set.

```
 1 train_dataset=TensorDataset(x_norm[:400],yy[:400])
 2 test_dataset=TensorDataset(x_norm[400:],yy[400:])
 3 trainloader=DataLoader(dataset=train_dataset,batch_size=16,shuffl
 4 testloader=DataLoader(dataset=test_dataset,batch_size=16,shuffle=
```

```
 1 #repeat a
 2 design=MS(['age','lstat'])
 3 X=design.fit_transform(Boston)
 4 Y=torch.tensor(Boston['medv'],dtype=torch.float32)
 5 X1=torch.tensor(Boston["age"],dtype=torch.float32)
 6 X2=torch.tensor(Boston["lstat"],dtype=torch.float32)
 7 X=torch.stack([X1,X2],axis=1)
 8 #normalization
 9 ux=torch.mean(X,axis=0)#[2,](ul,u2)
10 sd=torch.std(X,axis=0)
11 x_norm=(X-ux)/sd
12
13 uy=torch.mean(Y)
14 y_norm=(Y-uy)/torch.std(Y)
15
16
17 model = linear_regression(2,1) # model = torch.nn.Linear(2, 1, bi
18 print(model.weight)
19 train_dataset=TensorDataset(x_norm[:400],yy[:400])
20
21 trainloader=DataLoader(dataset=train_dataset,batch_size=16,shuffl
22 def criterion(yhat, y):
23   return torch.mean((yhat-y)**2)
24 optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
25
26 iters = 300
27 history = [] #300개의 로스
28 #n=len(trainloader.dataset)
29 for epoch in range(iters):
30   current_epoch_loss = 0
31   for x,y in trainloader: #두번째 for 문 ,x 10개, y 10개, 3개일 때도 있
32
33     yhat = model(x)
34     loss = criterion(yhat, y)
35     current_epoch_loss += loss.item()
36     loss.backward() #최소화하기 위해서
```

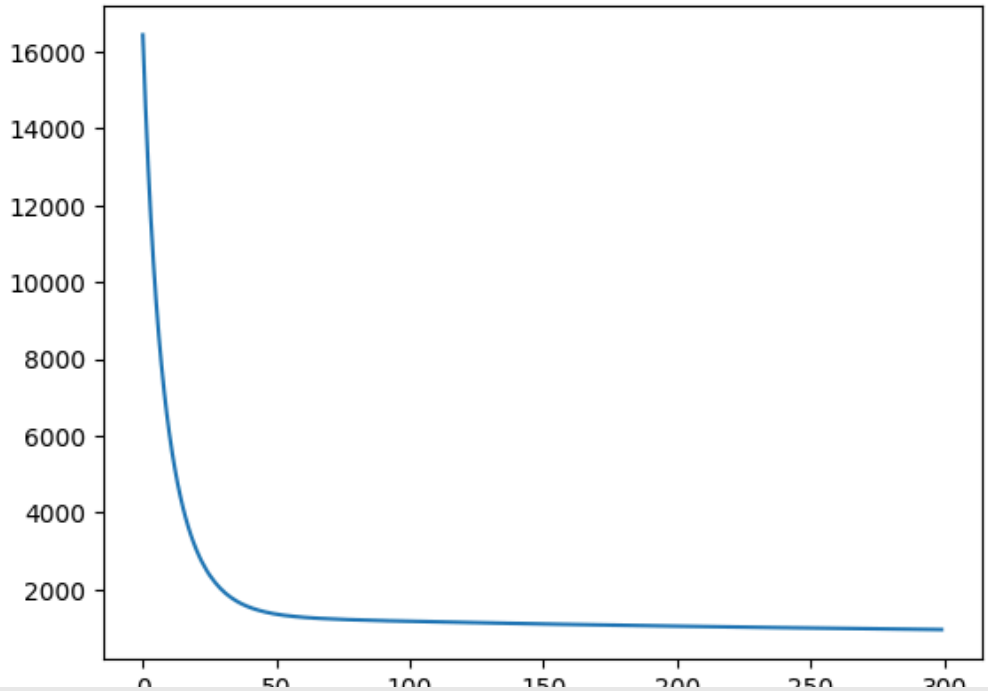
```
    optimizer.step() #50번 일어남, decent gradient
    optimizer.zero_grad()
  history.append(current_epoch_loss)

#test result
test_dataset=TensorDataset(x_norm[400:],yy[400:])
testloader=DataLoader(dataset=test_dataset,batch_size=16,shuffle=

#mse
Yhat=model(x_norm[400:])
print(criterion(Yhat,yy[400:])) #test data 에 테스트

```

```
Parameter containing:
tensor([[-0.5007],
        [ 0.9949]], requires_grad=True)
tensor(29.1841, grad_fn=<MeanBackward0>)
```

```
#B반복

design=MS(['age','lstat'])
X=design.fit_transform(Boston)
Y=torch.tensor(Boston['medv'],dtype=torch.float32)
X1=torch.tensor(Boston["age"],dtype=torch.float32)
X2=torch.tensor(Boston["lstat"],dtype=torch.float32)
X3=torch.tensor(Boston["lstat"]**2,dtype=torch.float32)
X=torch.stack([X1,X2,X3],axis=1)
#normalization
ux=torch.mean(X,axis=0)#[3,](ul,u2)
sd=torch.std(X,axis=0)
x_norm=(X-ux)/sd

uy=torch.mean(Y)
y_norm=(Y-uy)/torch.std(Y)

model = linear_regression(3,1) # model = torch.nn.Linear(2, 1, bi
print(model.weight)

train_dataset=TensorDataset(x_norm[:400],yy[:400])

trainloader=DataLoader(dataset=train_dataset,batch_size=16,shuffl

 #두개의 데이터
 #10개는 gpu명령, 메모리가 작아서 16으로 하는 게 좋음

def criterion(yhat, y):
  return torch.mean((yhat-y)**2)
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)

iters = 300
```

```
history = [] #300개의 로스
#n=len(trainloader.dataset)
for epoch in range(iters):
  current_epoch_loss = 0
  for x,y in trainloader: #두번째 for 문 ,x 10개, y 10개, 3개일 때도 있
    yhat = model(x)
    loss = criterion(yhat, y)
    current_epoch_loss += loss.item()
    loss.backward() #최소화하기 위해서
    optimizer.step() #50번 일어남, decent gradient
    optimizer.zero_grad()
  history.append(current_epoch_loss)

import matplotlib.pyplot as plt
plt.plot(history)
#mse
Yhat=model(x_norm[400:])
print(criterion(Yhat,yy[400:])) #test data 에 테스트

#이번 모델의 MSE가 작아짐

```

```
Parameter containing:
tensor([[ 0.5320],
        [ 2.5453],
        [-2.7427]], requires_grad=True)
tensor(26.3213, grad_fn=<MeanBackward0>)
```



3.1

(a) Using pytorch, calculate the train MSE of the following model:

Sales ~ Advertising + ShelveLoc

```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the Carseats dataset
Carseats = load_data('Carseats')

# Helper function for calculating MSE
def calculate_mse(y_true, y_pred):
    return torch.mean((y_true - y_pred) ** 2).item()

# Helper function for calculating R^2
def calculate_r2(y_true, y_pred):
    ss_total = torch.sum((y_true - torch.mean(y_true)) ** 2)
    ss_residual = torch.sum((y_true - y_pred) ** 2)
    return 1 - (ss_residual / ss_total).item()

# Convert the dataset to PyTorch tensors
def prepare_data():
    X = Carseats[['Advertising', 'ShelveLoc']]
    y = Carseats['Sales'].values.reshape(-1, 1)


    # One-hot encode ShelveLoc
    X = pd.get_dummies(X, columns=['ShelveLoc'], drop_first=True)

    # Normalize Advertising column
    scaler = StandardScaler()
    X['Advertising'] = scaler.fit_transform(X[['Advertising']])

    # Ensure the DataFrame contains only numerical values before c
    X = X.astype(np.float32)
    # One-hot encode ShelveLoc


    # Normalize Advertising column
    scaler = StandardScaler()
    X['Advertising'] = scaler.fit_transform(X[['Advertising']])

    X = torch.tensor(X.values)
    y = torch.tensor(y, dtype=torch.float32)

```

```
    return X, y

# Define the model for Sales ~ Advertising + ShelveLoc (Model a)
class ModelA(nn.Module):
    def __init__(self):
        super(ModelA, self).__init__()
        self.linear = nn.Linear(3, 1)

    def forward(self, x):
        return self.linear(x)

# Define the model for Sales ~ Advertising (Model b)
class ModelB(nn.Module):
    def __init__(self):
        super(ModelB, self).__init__()
        self.linear = nn.Linear(1, 1)

    def forward(self, x):
        return self.linear(x)

# Train the model
def train_model(model, X_train, y_train, num_epochs=1000, lr=0.01)
    criterion = nn.MSELoss()
    optimizer = optim.SGD(model.parameters(), lr=lr)

    for epoch in range(num_epochs):
        model.train()
        optimizer.zero_grad()
        y_pred = model(X_train)
        loss = criterion(y_pred, y_train)
        loss.backward()
        optimizer.step()

    return model

# Train and evaluate the model
def evaluate_model(model, X_train, y_train, X_test, y_test):
    # Calculate MSE on training data
    model.eval()
    with torch.no_grad():
        y_train_pred = model(X_train)
        train_mse = calculate_mse(y_train, y_train_pred)

        # Calculate MSE on test data
        y_test_pred = model(X_test)
        test_mse = calculate_mse(y_test, y_test_pred)

        # Calculate R² on training data
        train_r2 = calculate_r2(y_train, y_train_pred)

```

```
        # Calculate R² on test data
        test_r2 = calculate_r2(y_test, y_test_pred)

    return train_mse, test_mse, train_r2, test_r2

# Prepare data
X, y = prepare_data()

# Split into training (70%) and testing (30%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

# (a) Model: Sales ~ Advertising + ShelveLoc
model_a = ModelA()
trained_model_a = train_model(model_a, X_train, y_train)

# (b) Model: Sales ~ Advertising
X_train_advertising = X_train[:, 0].reshape(-1, 1)
X_test_advertising = X_test[:, 0].reshape(-1, 1)

model_b = ModelB()
trained_model_b = train_model(model_b, X_train_advertising, y_trai

# Evaluate Model (a) and (b)
train_mse_a, test_mse_a, train_r2_a, test_r2_a = evaluate_model(tr
train_mse_b, test_mse_b, train_r2_b, test_r2_b = evaluate_model(tr
print(train_mse_a, test_mse_a, train_r2_a, test_r2_a)

```

```
5.1499810218811035 4.812638282775879 0.3281272053718567 0.43245571851730347
```

```
코딩을 시작하거나 AI로 코드를 생성하세요.
```