Hitters example from ISL lab

```
##################################################
### imports

##basic
import numpy as np
import pandas as pd
import math

## graphics
import matplotlib.pyplot as plt

## sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LassoCV

## keras, will get from tensoflow
import tensorflow as tf

import random

## loss functions
def rmsef(y,yp):
   return(np.sqrt(np.mean((y-np.squeeze(yp))**2)))
def madf(y,yp):
   return(np.mean(np.abs(y-np.squeeze(yp))))


##################################################
### read in data

## same train/test split as in ISLR lab
hdtr = pd.read_csv("https://bitbucket.org/remcc/rob-data-sets/downloads/Gitters_trai
hdte = pd.read_csv("https://bitbucket.org/remcc/rob-data-sets/downloads/Gitters_test
print(hdtr.shape)
print(hdte.shape)
print(hdtr.columns) # names of variables
ntr = hdtr.shape[0] #sample size = number of rows
p = hdtr.shape[1]-1 #number of features = number of columns -1 , last column is y=sa

## get X and y as simple numpy arrays
# train
hdtrnp = hdtr.to_numpy()
Xtr = hdtrnp[:,:p] #first p columns
```

```
ytr = hdtrnp[:,-1] #last column

# test
hdtenp = hdte.to_numpy()
Xte = hdtenp[:,:p] #first p columns
yte = hdtenp[:,-1] #last column

## check it is already scaled, should scale using just train, but this is the way IS
X = np.vstack([Xtr,Xte])
print('feature means (should be 0):')
print(np.mean(X,axis=0))
print('feature sd (should be 1):')
print(np.std(X,axis=0))
```

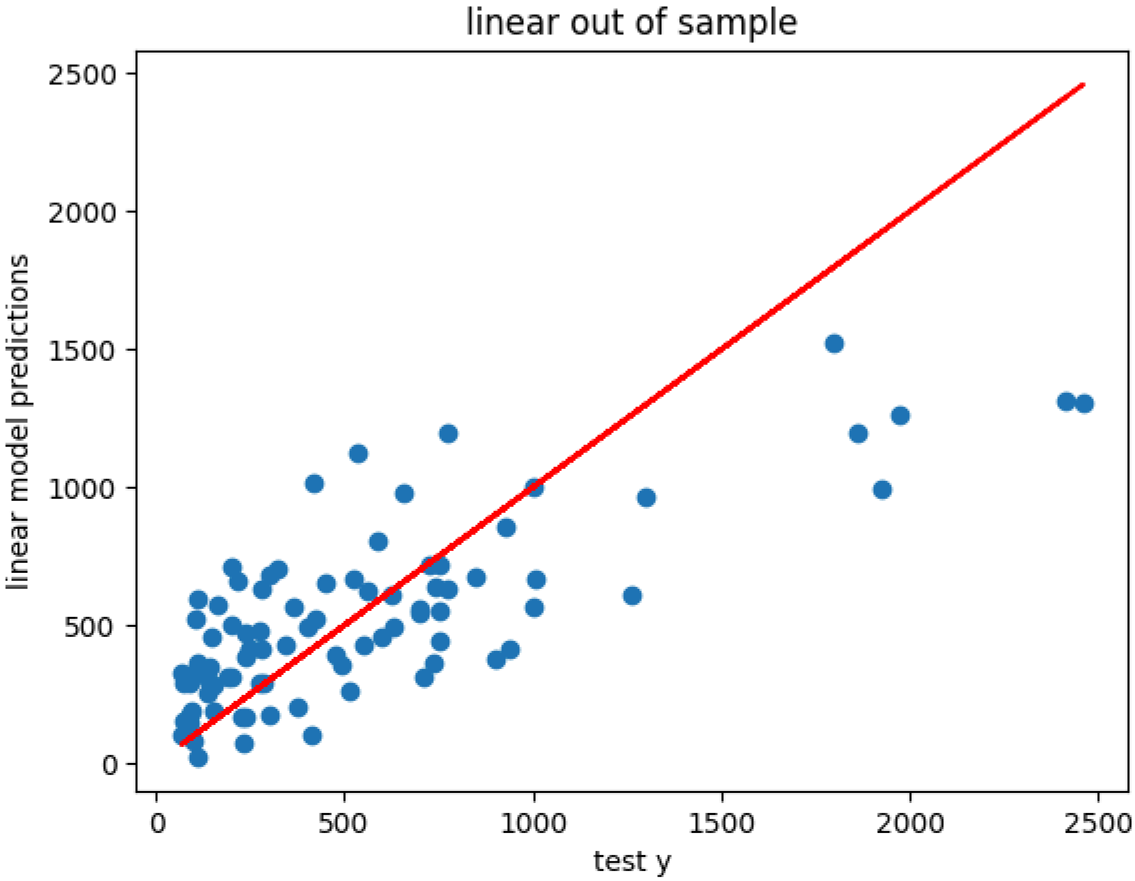
```
(176, 21)
(87, 21)
Index(['AtBat', 'Hits', 'HmRun', 'Runs', 'RBI', 'Walks', 'Years', 'CAtBat',
       'CHits', 'CHmRun', 'CRuns', 'CRBI', 'CWalks', 'LeagueA', 'LeagueN',
       'DivisionW', 'PutOuts', 'Assists', 'Errors', 'NewLeagueN', 'y'],
      dtype='object')
feature means (should be 0):
[ 1.83207906e-16  2.70168341e-16 -4.05252511e-17 -1.84052182e-16
 -1.55346796e-16  2.02626255e-17  1.71388041e-16  9.11818150e-17
  6.24764288e-17 -2.56659924e-16  6.75420851e-17  1.41838379e-16
  4.39023553e-17  1.53320533e-15 -1.53320533e-15 -2.02626255e-15
  1.74765145e-16 -1.35084170e-16  8.76358555e-16  1.55009085e-15]
feature sd (should be 1):
[0.99809705 0.99809705 0.99809705 0.99809705 0.99809705 0.99809705
 0.99809705 0.99809705 0.99809705 0.99809705 0.99809705 0.99809705
 0.99809705 0.99809705 0.99809705 0.99809705 0.99809705 0.99809705
 0.99809705 0.99809705]
```

```
##################################################
## fit linear on train, predict on test

lmmod = LinearRegression()
lmmod.fit(Xtr,ytr)
ypredlin = lmmod.predict(Xte)
yhatlin = lmmod.predict(Xtr)

print(f'out of sample test rmse from linear model is {rmsef(yte,ypredlin):0.4f}')
print(f'out of sample test mad from linear model is {madf(yte,ypredlin):0.4f}')

## check
print(f'check: rmse from sklearn mse fun: {math.sqrt(mean_squared_error(yte,ypredlir
```

```
out of sample test rmse from linear model is 341.0237
out of sample test mad from linear model is 254.6687
check: rmse from sklearn mse fun: 341.0237
```

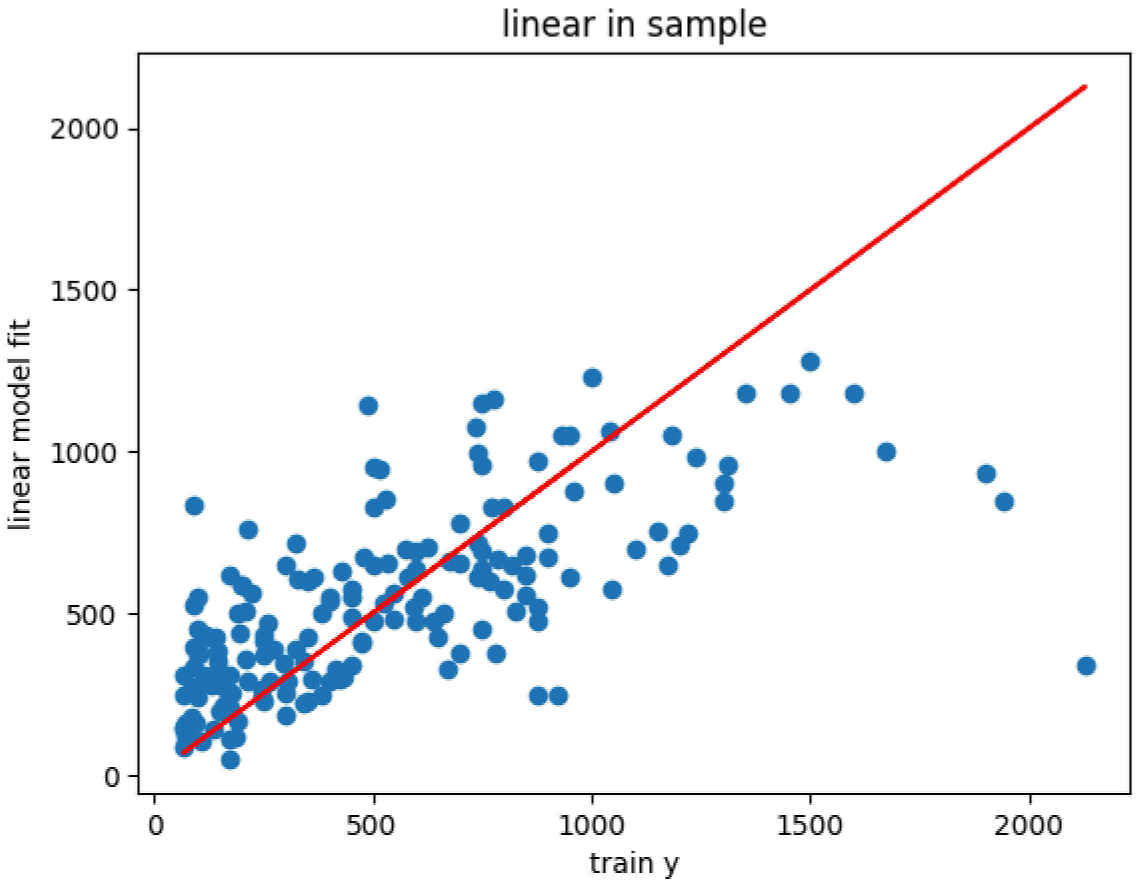
```
##plot out of sample prediction
plt.scatter(yte,ypredlin)
plt.plot(yte,yte,c='r')
plt.xlabel('test y'); plt.ylabel('linear model predictions')
plt.title('linear out of sample')
```

```
Text(0.5, 1.0, 'linear out of sample')
```



```
##plot in sample prediction, you can see why they use MAD as well as MSE
plt.scatter(ytr,yhatlin)
plt.plot(ytr,ytr,c='r')
plt.xlabel('train y'); plt.ylabel('linear model fit')
plt.title('linear in sample')
```

```
Text(0.5, 1.0, 'linear in sample')
```

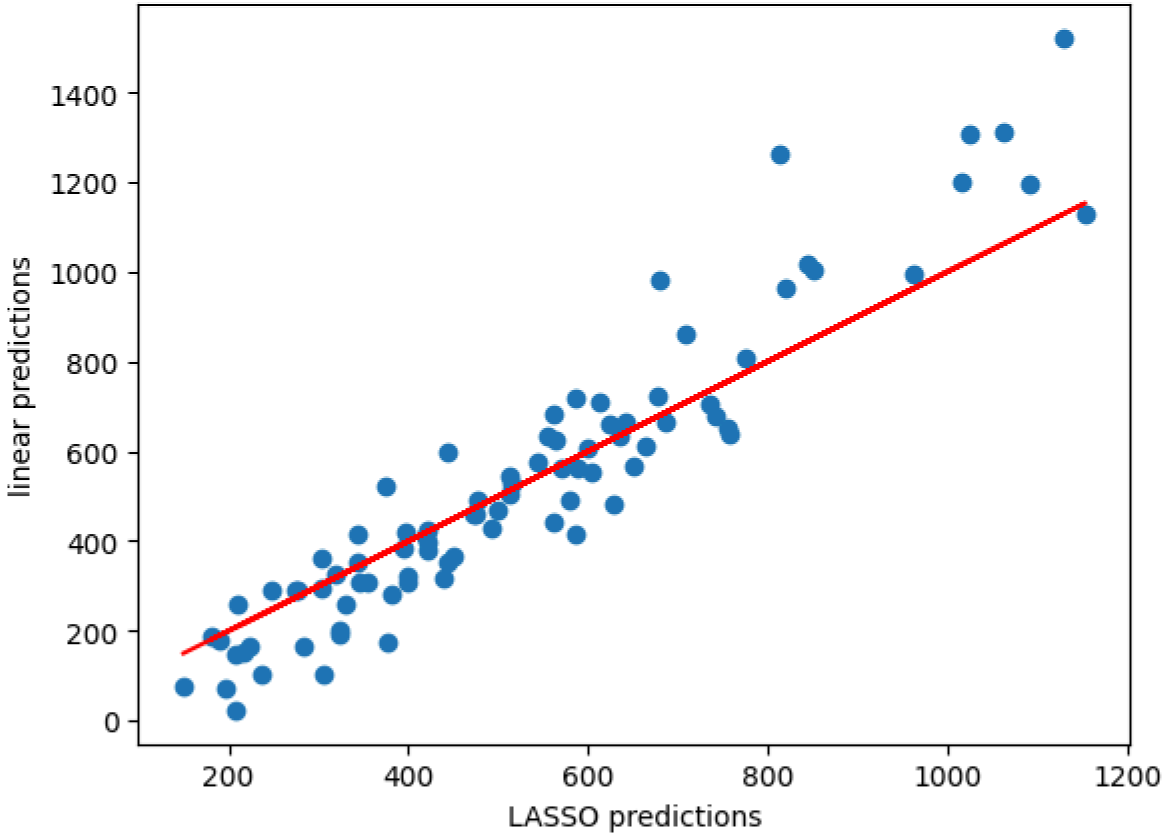


```
################################################
## fit LASSO on train, predict on test

## LassoCV seems to have mse hard coded in,
##    unlike glmnet which allows us to choose cv loss with type.measure parameter.
lcv = LassoCV(cv=10)
lcv.fit(Xtr,ytr)

#best alpha and coefficients
print("best alpha: ",lcv.alpha_)

#coefficents
print("coeficients at best alpha: ",lcv.coef_)
print("number of 0 coefficents: ",np.sum(lcv.coef_ == 0))

#predicted values
ypredL = lcv.predict(Xte)
yhatL = lcv.predict(Xtr)

print(f'out of sample test rmse from linear model is {rmsef(yte,ypredlin):0.4f}')
print(f'out of sample test mad from linear model is {madf(yte,ypredlin):0.4f}')
print(f'out of sample test rmse from LASSO is {rmsef(yte,ypredL):0.4f}')
print(f'out of sample test mad from LASSO is {madf(yte,ypredL):0.4f}')
```

```
best alpha:  25.238631113423445
coeficients at best alpha:  [  0.          16.34663779  30.45102074   2.16080661
  81.06971356   0.           0.         153.25603511   0.
   0.           1.25121039   0.          -0.           0.
 -40.48935331  18.97158766   0.          -0.           0.         ]
number of 0 coefficents:  12
out of sample test rmse from linear model is 341.0237
out of sample test mad from linear model is 254.6687
out of sample test rmse from LASSO is 370.4170
out of sample test mad from LASSO is 258.0145
```

```
## compare LASSO to linear
plt.scatter(ypredL,ypredlin)
plt.xlabel('LASSO predictions'); plt.ylabel('linear predictions')
plt.plot(ypredL,ypredL,c='r')
```

```
[<matplotlib.lines.Line2D at 0x7fcb5138d7d0>]
```



```
################################################
### single layer, 50 units, relu, dropout
seed=34
random.seed(seed)
np.random.seed(seed)
tf.random.set_seed(seed) ## ? just need this one ??

## make model
nunit =  50
```

```
nx = Xtr.shape[1] # number of x's
nn1 = tf.keras.models.Sequential()

## add one hidden layer and dropout, one linear output
nn1.add(tf.keras.Input(shape=(nx,)))
nn1.add(tf.keras.layers.Dense(units=nunit,activation='relu',))
nn1.add(tf.keras.layers.Dropout(.4))
nn1.add(tf.keras.layers.Dense(units=1))

#compile model
nn1.compile(loss='mse',optimizer='rmsprop',metrics=['mean_absolute_error'])

# fit
nepoch = 1500
nhist = nn1.fit(Xtr,ytr,epochs=nepoch,verbose=1,batch_size=32,validation_data=(Xte,y
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 89544.0781 - mean_absolute_error
Epoch 1492/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 95169.0859 - mean_absolute_error
Epoch 1493/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 90298.7891 - mean_absolute_error
Epoch 1494/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 87383.8828 - mean_absolute_error
Epoch 1495/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 80113.6797 - mean_absolute_error
Epoch 1496/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 89148.2656 - mean_absolute_error
Epoch 1497/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 88347.4453 - mean_absolute_error
Epoch 1498/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 91368.4688 - mean_absolute_error
Epoch 1499/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - loss: 93495.6953 - mean_absolute_error
Epoch 1500/1500
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - loss: 100980.7656 - mean_absolute_erro
```

```
# summary
nn1.summary()
```

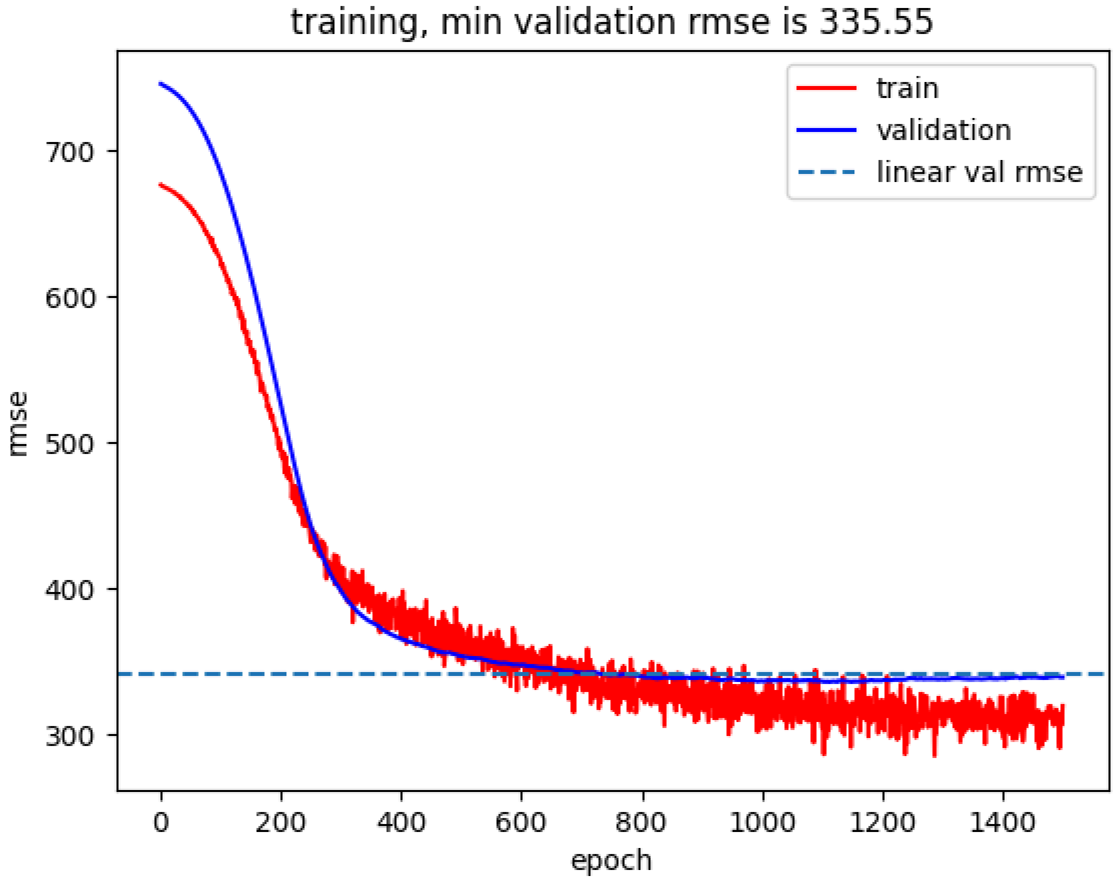
Model: "sequential"

Layer (type)	Output Shape	P
dense (Dense)	(None, 50)	
dropout (Dropout)	(None, 50)	
dense_1 (Dense)	(None, 1)	

```
Total params: 2,204 (8.61 KB)
Trainable params: 1,101 (4.30 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 1,103 (4.31 KB)
```

```
##################################################
### plot training by epoch
yhatnn = nn1.predict(Xtr)
yprednn = nn1.predict(Xte)

trL = np.sqrt(nhist.history['loss'])
teL = np.sqrt(nhist.history['val_loss'])
epind = range(1,len(trL)+1)
plt.plot(epind,trL,c='red',label='train')
plt.plot(epind,teL,c='blue',label='validation')
plt.xlabel('epoch'); plt.ylabel('rmse')
plt.axhline(rmsef(yte,ypredlin),linestyle='--',label='linear val rmse')
plt.legend()
minrmse = teL.min()
plt.title(f'training, min validation rmse is {minrmse:0.2f}')
```

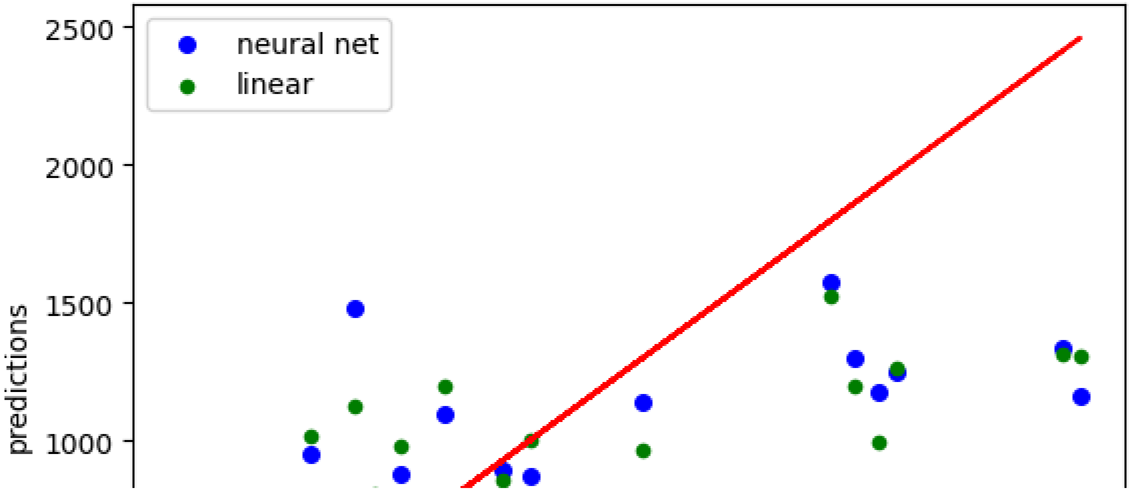
```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
Text(0.5, 1.0, 'training, min validation rmse is 335.55')
```



Generate a slider using jupyter widgets Close

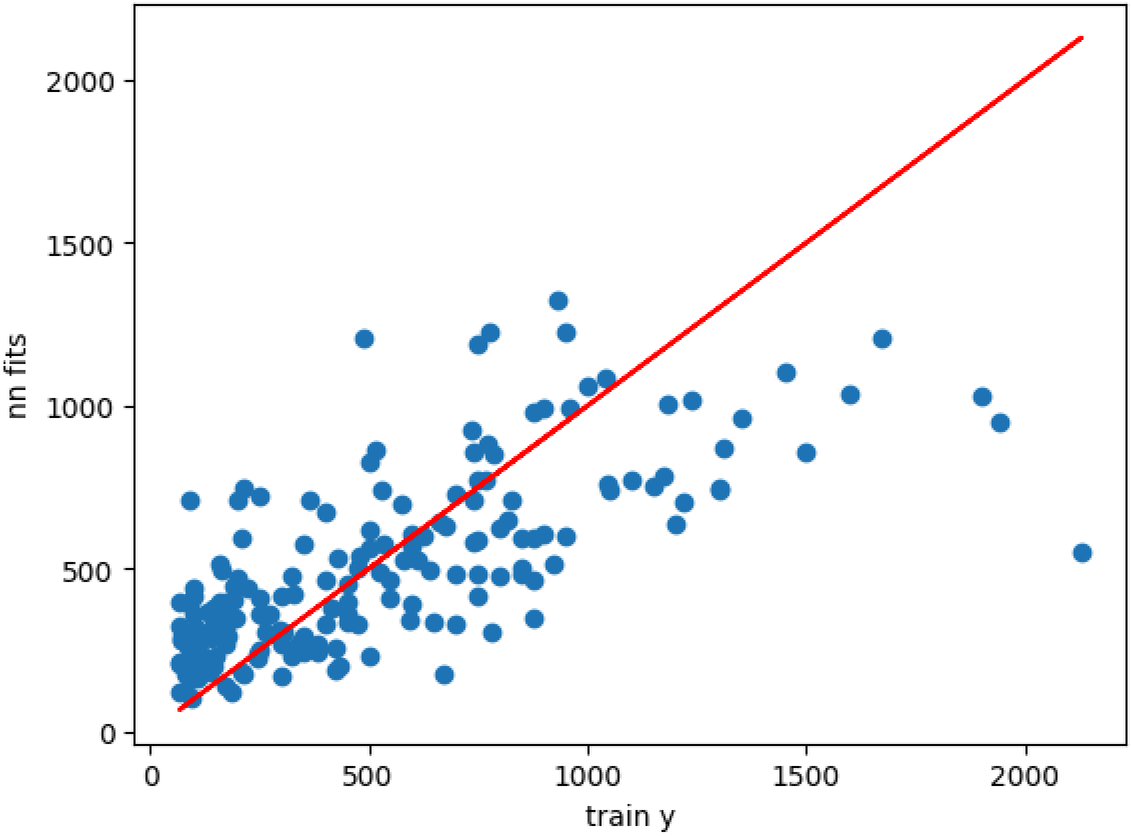
```
##################################################
### plot and rmse/mad

# out-of-sample (test)
plt.scatter(yte,yprednn,c='blue',s=30,label='neural net')
plt.scatter(yte,ypredlin,c='green',s=20,label='linear')
plt.plot(yte,yte,c='r')
plt.xlabel('test y'); plt.ylabel('predictions')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fcb4a7adad0>
```



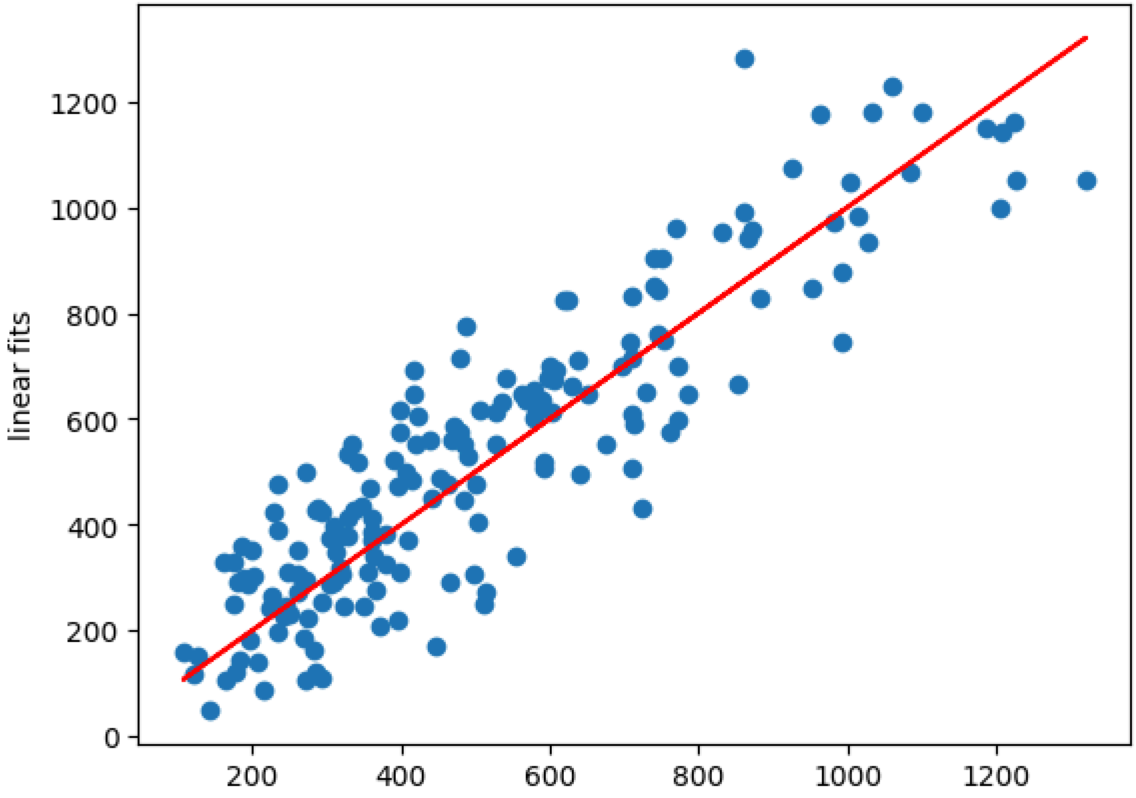
```
# in-sample (train)
plt.scatter(ytr,yhatnn)
plt.plot(ytr,ytr,c='r')
plt.xlabel('train y'); plt.ylabel('nn fits')
```

```
Text(0,0.5,'nn fits')
```



```
plt.scatter(yhatnn,yhatlin)
plt.plot(yhatnn,yhatnn,c='r')
plt.xlabel('nn fits'); plt.ylabel('linear fits')
```

```
Text(0, 0.5, 'linear fits')
```



```
print(f'out of sample test rmse from linear model is {rmsef(yte,ypredlin):0.4f}')
print(f'out of sample test rmse from LASSO is {rmsef(yte,ypredL):0.4f}')
print(f'out of sample test rmse from neural net is {rmsef(yte,ypednn):0.4f}')
print('\n\n')
```

```
out of sample test rmse from linear model is 341.0237
out of sample test rmse from LASSO is 370.4170
out of sample test rmse from neural net is 338.8322
```

```
print(f'out of sample test mad from linear model is {madf(yte,ypredlin):0.4f}')
print(f'out of sample test mad from LASSO is {madf(yte,ypredL):0.4f}')
print(f'out of sample test mad from neural net is {madf(yte,ypednn):0.4f}')
```

```
out of sample test mad from linear model is 254.6687
out of sample test mad from LASSO is 258.0145
out of sample test mad from neural net is 247.8864
```

Start coding or generate with AI.