In [1]:

**import** numpy **as** np  
**import** pandas **as** pd  
**from** matplotlib **import** pyplot **as** plt  
  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** pandas **import** read\_csv  
**from** sklearn.linear\_model **import** ElasticNet  
**from** sklearn.preprocessing **import** StandardScaler  
**from** sklearn.model\_selection **import** GridSearchCV

In [43]:

data **=** pd**.**read\_csv(r"C:\Users\zcolr\Downloads\phenotype.csv", sep **=** ',',index\_col**=**0)  
gen\_txt **=** pd**.**read\_csv(r"C:\Users\zcolr\Downloads\genotype\_full\_1\_2.txt",sep**=**'\t',index\_col**=**0)

In [44]:

data1 **=** pd**.**concat([gen\_txt, data['1\_CobaltChloride\_1']], axis**=**1)  
print("Phenotype shape:",data**.**shape)  
print("Genotype shape:",gen\_txt**.**shape)  
  
data1**.**head()

Phenotype shape: (4390, 20)  
Genotype shape: (4390, 28220)

Out[44]:

|  | **33070\_chrI\_33070\_A\_T** | **33147\_chrI\_33147\_G\_T** | **33152\_chrI\_33152\_T\_C** | **33200\_chrI\_33200\_C\_T** | **33293\_chrI\_33293\_A\_T** | **33328\_chrI\_33328\_C\_A** | **33348\_chrI\_33348\_G\_C** | **33403\_chrI\_33403\_C\_T** | **33502\_chrI\_33502\_A\_G** | **33548\_chrI\_33548\_A\_C** | **...** | **12049199\_chrXVI\_925939\_T\_C** | **12049441\_chrXVI\_926181\_C\_T** | **12050613\_chrXVI\_927353\_T\_G** | **12051167\_chrXVI\_927907\_A\_C** | **12051240\_chrXVI\_927980\_A\_G** | **12051367\_chrXVI\_928107\_C\_T** | **12052782\_chrXVI\_929522\_C\_T** | **12052988\_chrXVI\_929728\_A\_G** | **12053130\_chrXVI\_929870\_C\_T** | **1\_CobaltChloride\_1** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **01\_01** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | ... | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | -2.253831 |
| **01\_02** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | ... | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | -1.887746 |
| **01\_03** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | ... | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1.047185 |
| **01\_04** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | ... | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2.417437 |
| **01\_06** | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | ... | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | -1.041743 |

5 rows × 28221 columns

In [45]:

data1 **=** data1**.**dropna()  
print(data1**.**shape)  
  
*#Normalize*   
scale **=** StandardScaler()

(4168, 28221)

In [47]:

x\_ **=** data1**.**iloc[:,0:**-**1]**.**to\_numpy()  
y\_ **=** data1['1\_CobaltChloride\_1']**.**to\_numpy()  
  
*#x\_train = data1.to\_numpy()[0:,0]*  
*#y\_train = data1['1\_CobaltChloride\_1'].to\_numpy()[0]*  
  
*#x\_train\_std = np.array(x\_train).reshape(-1, 1).T*  
  
print("Inputs: \n",x\_,"\nTarget Value: \n",y\_)

Inputs:   
 [[1 1 1 ... 2 2 2]  
 [1 1 1 ... 2 2 2]  
 [2 2 2 ... 1 1 1]  
 ...  
 [1 1 1 ... 2 2 2]  
 [2 2 2 ... 1 1 1]  
 [2 2 2 ... 1 1 1]]   
Target Value:   
 [-2.25383071 -1.88774566 1.04718512 ... -0.41612464 -0.69793334  
 -0.60690851]

In [48]:

x\_**.**shape

Out[48]:

(4168, 28220)

In [6]:

*#scale.fit(x\_train)*  
*#x\_train\_std = scale.transform(x\_train)*

In [41]:

data**.**describe()

Out[41]:

|  | **1\_CobaltChloride\_1** | **1\_CopperSulfate\_1** | **1\_Diamide\_1** | **1\_E6-Berbamine\_1** | **1\_Ethanol\_1** | **1\_Formamide\_1** | **1\_Hydroxyurea\_1** | **1\_IndolaceticAcid\_1** | **1\_Lactate\_1** | **1\_Lactose\_1** | **1\_MagnesiumChloride\_1** | **1\_ManganeseSulfate\_1** | **1\_Menadione\_1** | **1\_Neomycin\_1** | **1\_Raffinose\_1** | **1\_Trehalose\_1** | **1\_Xylose\_1** | **1\_YNB\_1** | **1\_YPD\_1** | **1\_Zeocin\_1** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 4168.000000 | 4276.000000 | 4309.000000 | 4310.000000 | 4261.000000 | 4268.000000 | 4297.000000 | 4286.000000 | 3762.000000 | 3816.000000 | 4263.000000 | 4323.000000 | 4298.000000 | 4304.000000 | 4163.000000 | 4311.000000 | 4286.000000 | 4339.000000 | 4331.000000 | 4316.000000 |
| **mean** | -0.002167 | 0.007495 | -0.003565 | 0.010244 | -0.001460 | -0.016001 | -0.015837 | -0.018995 | -0.000989 | -0.007086 | -0.008544 | 0.000123 | 0.004164 | 0.003531 | -0.008845 | -0.008982 | -0.020917 | 16.387996 | 24.988156 | -0.000662 |
| **std** | 0.928656 | 0.716367 | 0.839731 | 0.967866 | 0.472114 | 0.787717 | 0.364769 | 0.615602 | 0.776357 | 0.770810 | 0.438869 | 0.948098 | 0.803748 | 0.833738 | 0.663019 | 0.735866 | 0.738140 | 1.069024 | 2.273753 | 0.967717 |
| **min** | -2.986634 | -4.359688 | -3.458580 | -2.115366 | -2.111576 | -4.352861 | -2.865521 | -3.222564 | -3.944102 | -3.584842 | -2.279338 | -2.339743 | -3.050449 | -3.814010 | -2.905491 | -3.619955 | -3.237204 | 11.811819 | 16.425624 | -2.106594 |
| **25%** | -0.570615 | -0.257038 | -0.531162 | -0.664872 | -0.299687 | -0.410830 | -0.221138 | -0.264527 | -0.465318 | -0.490573 | -0.280307 | -0.684072 | -0.523554 | -0.349418 | -0.424614 | -0.441370 | -0.493247 | 15.688399 | 23.477206 | -0.810701 |
| **50%** | -0.353169 | 0.137448 | 0.134708 | -0.402642 | 0.014423 | 0.093355 | -0.001265 | 0.093428 | 0.034626 | 0.018394 | -0.001556 | -0.257264 | 0.042384 | 0.204966 | 0.004687 | 0.017288 | -0.018307 | 16.432730 | 24.876704 | -0.198123 |
| **75%** | 0.366364 | 0.455080 | 0.577930 | 0.600145 | 0.306615 | 0.491365 | 0.209070 | 0.369638 | 0.513707 | 0.498850 | 0.283265 | 0.583188 | 0.566911 | 0.590661 | 0.420898 | 0.491016 | 0.462095 | 17.098054 | 26.356892 | 0.821270 |
| **max** | 4.128672 | 2.152255 | 2.353494 | 3.531346 | 2.061407 | 3.932560 | 2.195662 | 1.797898 | 3.118644 | 4.434836 | 2.168053 | 3.339181 | 3.244874 | 2.336118 | 2.819776 | 2.398808 | 3.546980 | 20.600210 | 34.536419 | 2.742622 |

In [49]:

print(x\_**.**shape)  
print(y\_**.**shape)

(4168, 28220)  
(4168,)

In [32]:

*#print(y\_train[0])*  
*#print(x\_train[0])*

In [33]:

**class** Layer\_Dense:  
 **def** \_\_init\_\_(self, n\_inputs, n\_nuerons):  
 self**.**weights **=** 0.1 **\*** np**.**random**.**randn(n\_inputs, n\_nuerons)  
 self**.**biases **=** np**.**zeros((1, n\_nuerons))  
 **def** forward(self, inputs):  
 self**.**output **=** np**.**dot(inputs, self**.**weights) **+** self**.**biases  
   
**class** Activation\_ReLU:  
 **def** forward(self, inputs):  
 self**.**output **=** np**.**maximum(0, inputs)  
  
**class** SoftMax\_Actv:  
 **def** forward(self, inputs):  
 exp\_val **=** np**.**exp(inputs **-** np**.**max(inputs, axis**=**1, keepdims**=True**))  
 prob **=** exp\_val **/** np**.**sum(exp\_val, axis**=**1, keepdims**=True**)  
 self**.**output **=** prob  
  
**class** Loss:  
 **def** calc(self, output, y):  
 sample\_loss **=** self**.**forward(output,y)  
 data\_loss **=** np**.**mean(sample\_loss)  
 **return** data\_loss  
  
**class** Loss\_Catergory\_E(Loss):  
 **def** forward(self, y\_pred, y\_true):  
 samples **=** len(y\_pred)  
 y\_pred\_clip **=** np**.**clip(y\_pred, 1e-7, 1**-**1e-7)  
   
 **if** len(y\_true**.**shape) **==** 1:  
 correct\_confidence **=** y\_pred\_clip[range(samples), y\_true]  
 **elif** len(y\_true**.**shape) **==** 2:  
 correct\_confidence **=** np**.**sum(y\_pred\_clip **\*** y\_pred, axis**=**1)  
   
 neg\_log\_like **=** **-**np**.**log(correct\_confidence)  
 **return** neg\_log\_like

In [34]:

*#data2 = pd.concat([gen\_txt, data['1\_CopperSulfate\_1']], axis=1)*

In [35]:

*#x\_test = gen\_txt.to\_numpy()[0:,0].reshape(-1, 1)*  
*#x\_test1 = np.array(x\_test)*  
  
*#x\_test\_std = scale.fit(x\_test1)*

In [36]:

*# define model*  
model **=** ElasticNet(alpha**=**1.0, l1\_ratio**=**0.5)  
  
*# fit model*  
model**.**fit(x\_ ,y\_)  
*# define genotpe test data*  
*# make a prediction*  
ypred **=** model**.**predict(x\_)  
*# summarize prediction*  
print('Predicted: \n',ypred)  
  
*#param\_grid = {'alpha': [0.1, 1, 10], 'l1\_ratio': [0.1, 0.5, 0.9]}*  
  
*# Initialize the Elastic Net model*  
*#model = ElasticNet()*  
  
*# Use GridSearchCV to find the best combination of alpha and l1\_ratio*  
*#grid\_search = GridSearchCV(model, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')*  
*#grid\_search.fit(X\_train, y\_train)*  
  
*# Print the best combination of alpha and l1\_ratio*  
*#print('Best parameters:', grid\_search.best\_params\_)*  
  
*# Train the final model using the best combination of alpha and l1\_ratio*  
*#best\_model = ElasticNet(alpha=grid\_search.best\_params\_['alpha'], l1\_ratio=grid\_search.best\_params\_['l1\_ratio'])*  
*#best\_model.fit(X\_train, y\_train)*

Predicted:   
 [-0.00216727 -0.00216727 -0.00216727 ... -0.00216727 -0.00216727  
 -0.00216727]

In [38]:

ypred**.**shape

Out[38]:

(4168,)

In [50]:

print(x\_train**.**shape)  
print(y\_train**.**shape)

(4168, 28220)  
(4168,)

In [73]:

*#scaler = StandardScaler()*  
*#inputs\_scaled = scaler.fit\_transform(x\_train)*  
  
*# split the data into training and testing sets*  
X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(x\_, y\_, test\_size**=**0.2, random\_state**=**42)  
  
*# initialize the Elastic Net regression model*  
model **=** ElasticNet(alpha**=**0.0, l1\_ratio**=**0.1)  
  
*# train the model on the training set*  
model**.**fit(X\_train, y\_train)  
  
 *# make predictions on the test set*  
y\_pred **=** model**.**predict(X\_test)  
  
*# print the model coefficients*  
print('Model coefficients:', model**.**coef\_)  
  
*# print the model intercept*  
print('Model intercept:', model**.**intercept\_)  
  
*# print the mean squared error of the model predictions*  
**from** sklearn.metrics **import** mean\_squared\_error  
mse **=** mean\_squared\_error(y\_test,y\_pred)  
print('Mean squared error:', mse)  
  
*#print('Predicted: \n', check)*  
  
print(y\_pred)

Model coefficients: [ 0. 0.00028867 0.00028866 ... -0.00055774 -0.00035392  
 -0.00058249]  
Model intercept: 0.43010762933082314  
Mean squared error: 0.48920861471841565  
[ 0.88511913 0.52353753 -0.76546957 -0.90974516 0.97049122 -0.39633694  
 -0.78089807 -0.22618172 0.58170134 1.03445857 0.89233933 -0.58145795  
 -0.2853995 1.08596052 0.31760162 -0.59598293 -0.0934545 0.10604977  
 0.18784766 -0.5398372 -0.50620311 -0.54249919 0.38608576 0.11574532  
 -0.56177423 -0.53741172 0.57059055 -0.21275049 -0.21204702 -0.56077902  
 -0.22487836 0.21592124 -0.00205216 -0.58453959 -0.10740228 0.7044551  
 0.11610875 -0.15075788 -0.54398632 -0.06573941 -0.37884932 0.82014464  
 -0.72409547 0.90674718 -0.10356042 -0.13242459 0.32656596 0.04355098  
 0.04089848 0.82714581 -0.42707332 -0.34374623 0.11557235 -0.63881715  
 -0.90008088 -0.15290858 0.71618092 -0.00934777 -0.50811469 0.47680716  
 -0.26836946 0.47490271 0.72638705 0.15027244 0.55553829 0.28166036  
 -0.73081035 -0.57880115 -0.14349054 0.36237979 -0.59688672 -0.22009336  
 0.64206557 -0.01333211 1.09541658 -0.34388794 0.33670305 0.44204586  
 0.50597375 -0.18340741 -0.05839625 -1.13930624 -0.02449817 0.69809408  
 0.44345547 0.03942533 0.58701112 -1.42762442 1.06561977 -0.29036404  
 0.37351686 -0.79770587 0.612909 -1.67219122 -1.17647432 0.31891989  
 0.66542429 0.45280512 0.68229051 -0.52828408 0.55418049 0.08946426  
 -0.63594173 -0.102749 0.07713424 -0.44173878 -0.52400552 0.98097827  
 0.6436546 1.37165645 0.07379337 0.05328529 0.38482576 0.47857629  
 0.32678597 0.3384187 -0.70455715 -0.20485815 0.09932498 -0.10375863  
 0.09712013 1.02832253 -0.26038776 1.32462329 0.15307062 0.03926414  
 0.95650313 -0.34755498 -0.55941051 0.14576633 1.09955101 0.71510843  
 -0.21901397 -0.12246817 0.08167592 -0.2683779 -1.16486607 0.50340097  
 1.00278637 0.4785242 0.1500676 -1.09619408 0.11213292 0.08259288  
 -0.7319909 0.3923007 0.34363289 0.48388255 -0.14975514 -0.52159382  
 0.64528692 0.17241211 0.12699322 0.13600356 -0.66588099 -0.07318169  
 -0.85904632 0.26414903 -0.1261498 -0.89805301 -0.48433871 0.34924975  
 0.4545461 -0.37977786 0.56243687 0.03173165 1.00963067 -0.16367116  
 0.74355101 1.00041271 -0.2329374 0.4115574 0.09102669 -0.0732397  
 0.08761633 -0.25830052 0.86490101 0.65251096 0.36120723 0.06734118  
 -0.01378773 0.05638205 -0.37537454 0.29452561 -0.15430895 -1.05332485  
 -0.02339334 -0.60226379 0.21123654 -0.07690402 0.05615342 0.87347227  
 0.16835666 0.19193696 -0.245841 -1.10226262 0.04095702 -0.30897531  
 0.98966276 -0.67676471 -0.85829752 -0.92081918 -0.46384305 0.15587772  
 0.40140733 -0.32655705 0.47002999 -0.27484648 0.7037919 1.20138211  
 0.15422913 0.53408255 0.59209376 1.09367609 0.63038062 1.18978447  
 -0.19046574 0.30250849 -0.21498834 0.39196333 -0.43058212 0.17984126  
 0.7610575 0.12570543 0.15126345 -0.23686374 -0.23117859 -0.00853489  
 0.17379907 -0.91904334 -0.50917556 0.8985167 -0.65430249 -0.38097028  
 1.05918353 -0.33714312 -0.77105633 -0.61369023 -0.68067639 -0.3505384  
 0.71828632 -0.07513182 -0.21211425 -0.87069348 0.52962092 0.22172498  
 0.17612102 -0.03708112 -0.64651019 1.23616611 0.89777144 0.0722904  
 -0.43245739 -0.48471207 -0.23104462 0.50230284 -0.07482178 0.29807405  
 -0.14322768 -0.53333018 -1.3723344 -0.96219095 1.34439396 0.50436957  
 -1.06145769 -0.55221131 -0.15673282 -0.61900267 -0.3215994 -0.22391798  
 -0.58144969 0.35859146 -0.79858973 -0.49553649 -0.52073707 -0.08945799  
 -0.49206585 0.50678348 0.06976874 -0.69071254 0.49123843 0.23205932  
 0.07159292 0.13894709 0.81467793 -0.73850659 -0.54882532 -0.44606773  
 0.91759305 -0.9699167 -0.20827729 -0.18658721 -0.52998478 -1.14742297  
 -0.3634262 0.27571829 0.47064974 0.10396083 0.15092107 0.41966661  
 -0.11229952 0.46278091 0.00883569 -0.66401893 0.48973938 0.39596651  
 -0.8387907 -0.03444697 0.07727073 -0.11672053 0.3067382 -1.18152215  
 -0.11252822 0.4841962 0.03928568 0.70759779 1.16550549 -0.65057131  
 0.49679299 0.08347086 0.04808667 -0.3878394 0.59626829 -0.25470253  
 -0.30421375 -1.34943152 -0.13403852 -0.53576982 -0.95212596 -0.81313176  
 -0.18652014 -1.11422713 1.47651277 0.27742035 -0.32150857 -1.41551133  
 -0.33146366 0.39926918 -0.44996839 0.0068409 -0.1067649 1.22378268  
 1.09508999 0.39284269 -0.95418262 -0.61604182 -0.15049319 1.39244967  
 -0.443103 1.8973064 0.50343102 0.58580494 -1.48856573 -0.03059518  
 -0.12604513 -0.67898534 1.15865435 0.35955987 0.08716428 -0.49091939  
 1.00695302 -0.97120992 -1.24982325 0.61967838 0.26622097 0.54258271  
 0.41672121 0.63087862 0.77231247 0.99532978 0.9466069 -0.84709958  
 0.57212112 -0.62532296 -0.40698399 -0.31729042 -1.23666619 -0.78406684  
 0.31481527 -0.00207914 -0.80332234 0.80316971 0.08978573 -0.78453229  
 -0.71985891 0.71455854 0.3210675 -0.19137582 -0.33690015 0.96367862  
 -0.51852326 -0.52026427 -0.57601281 -0.26598351 -1.04276851 -0.51467805  
 0.68857105 -0.11958841 -0.06153621 -1.79157283 -0.07680605 0.03984833  
 -0.39280713 0.35067102 -0.19522608 -1.3352473 0.20827816 -0.55529006  
 -0.50348469 0.79112535 -1.2289515 -0.54840663 -0.68700867 -1.13536129  
 -0.52422943 -0.4463555 0.77609647 -0.52014047 0.84330265 -0.4952522  
 -0.82969356 -0.72541355 0.33059383 -0.49810976 -0.08040826 0.33121971  
 0.0222207 -0.66106447 -1.14519547 -0.3804269 1.11987187 0.5204147  
 -0.08084834 0.04028017 -0.13291527 0.49513387 -0.21830397 -0.84919527  
 0.13493156 1.07157287 0.39791513 -0.16788081 0.11783005 0.30918324  
 -0.40727362 -1.13509351 0.14620316 0.37480519 -0.3672252 0.95684942  
 -0.75634458 -0.79758486 -0.31871688 -0.3321575 0.79435887 0.42313729  
 0.03327169 0.0189194 0.16601338 0.86099495 -0.45027603 0.12649825  
 1.13811374 0.92885923 -0.90495537 0.38381247 0.31327677 -0.44301448  
 0.40468775 0.17380735 0.25718159 -0.56533797 -0.89897166 0.20623864  
 -0.45843777 0.20510234 -0.04404304 0.32101627 -0.26504961 -0.04511172  
 -0.37714277 -0.7693971 -0.30808489 -0.93539794 -0.6601565 -0.73997162  
 0.14894737 -1.19079048 -0.56305154 0.22809663 0.28577114 1.14619002  
 -0.43688756 0.42383313 0.82909636 0.26143129 -1.41448219 -0.21192604  
 0.60192375 0.11744167 0.20518768 -0.73258087 0.40867901 0.66259948  
 -0.02503564 -0.05126886 -0.18427301 0.2669131 0.82769794 -0.00376931  
 -0.73562897 -0.6506214 -0.23874197 0.13558305 0.49930606 -0.53211322  
 -0.80859994 -1.19678498 0.46460611 -0.06581452 -1.33796858 0.69493197  
 -0.12225853 0.32377157 0.18569108 0.1333354 -0.59749012 -0.2310377  
 -0.49124198 0.4630876 -0.58339184 -0.15512527 -0.1604488 -1.45694227  
 0.22860235 -0.31854918 0.37638743 -0.18330805 -0.37025624 0.51675548  
 -0.3778179 -0.05140053 -0.46081135 1.06939031 0.68493678 -0.32519889  
 0.14768698 -0.31331882 -0.94566891 -0.18035389 -0.2827062 -0.46491759  
 1.18125133 -0.62510412 -0.5551392 0.76165478 -0.36365776 1.42612994  
 -0.76042798 -0.8455119 -0.45942404 1.62216514 -0.45626982 -0.50346571  
 0.54121998 -0.11505492 0.70278786 -0.33685138 0.26155895 0.77925367  
 0.09225238 0.22257855 0.42523701 0.6089501 0.28486479 -0.01554343  
 -0.00688096 0.03253688 0.77891532 0.24732628 -1.46235448 0.58746145  
 -0.93219282 -0.01991429 -0.17196538 -0.77808981 -0.31705203 -0.73395187  
 -0.37359729 0.07506578 -1.53700661 -0.12194244 -0.27802564 -1.20015015  
 -0.86765901 -0.17345323 0.27735171 -0.30073692 0.08461383 0.84549743  
 -0.07596932 -0.90437918 -0.86921838 0.18546973 0.64021896 0.01038798  
 0.05098262 -0.86111288 0.18424489 0.10542401 0.9838235 0.00819233  
 0.09049862 -0.25269511 -0.2940285 -0.29645353 0.67126243 0.68087996  
 -0.66384431 0.05923852 0.6892458 0.80189663 0.99737617 0.58313048  
 -0.55591065 1.12935639 -0.9327511 -0.09639488 0.27600723 0.60092039  
 -0.66253447 -0.72893684 0.25471875 0.48823219 0.14072765 1.02385218  
 -0.45256014 0.9596628 0.27395915 -0.36536899 0.33216264 -0.59040765  
 -0.19879265 0.13960615 0.13253309 -0.19725925 -0.36386239 -0.19378543  
 0.23892569 -1.21274413 -0.25011469 -0.25413038 0.28647772 0.33089733  
 -0.09744478 -0.82106407 -0.63100027 -1.07017301 0.76921497 0.10201263  
 -0.62893424 0.06447604 -0.44955076 0.37276818 -0.04086207 0.35004418  
 0.02741932 -0.03075997 0.89679716 0.04969427 -0.5064458 0.74530419  
 -0.24617208 -0.26724055 -1.62349348 -0.52272073 -0.64022765 -0.44454005  
 0.01424243 -0.41603168 0.38240911 -0.13199518 -0.39117796 0.28905534  
 0.84929576 0.15142059 -0.09175682 0.57287091 0.53152804 -1.24680284  
 -0.26750053 0.06308641 0.5633481 -0.2501332 0.04387515 0.91079814  
 -1.8759626 -0.21306614 -0.53754698 -0.21537534 0.13958758 -0.44978204  
 -0.39799899 1.31238359 0.47895267 -0.35990842 -1.22044926 -0.1664458  
 1.01675616 -0.3037876 -0.16012819 0.05894382 0.58382851 -0.13147617  
 0.15442684 -0.06332278 0.54997859 0.03072229 -0.50202384 -0.45927265  
 -0.15328929 -0.07199843 0.03860502 0.92166047 0.37464028 -0.05865004  
 -0.29607897 0.59267367 1.05922289 0.197237 0.71251986 0.56250082  
 0.03722506 0.42765583 -0.61769892 -0.79370759 -0.80685413 -0.42713176  
 -0.15887453 -0.48535367 0.49370803 -0.7263993 -0.49554376 -0.12262951  
 -0.46621143 0.77327619 0.03007785 0.70895392 0.81622894 0.46844449  
 0.42803968 -0.20294329 -0.33107371 -0.46380574 0.03954705 -0.21068321  
 -0.42885172 -0.40817744 -0.66857706 0.34979886 -0.04930348 -1.21251499  
 -0.31974983 -1.63435104 0.02982496 -0.78612371 -0.99664013 0.50462335  
 -0.21430836 -0.61413704 0.83461732 0.09470827 0.23630424 -0.7984954  
 0.42758849 -0.63981842 0.90661974 -0.35661436 1.25439599 0.12376225  
 0.15915715 -0.69001745 0.6369295 0.16250701 0.99917831 0.40765427  
 1.28721299 -0.62701552 -0.47128993 -0.97657842 0.26857172 -0.78446931  
 0.5550719 -0.47280434 0.3622701 0.11432673 0.47642349 -0.27861277  
 0.05269561 -1.39120062 -0.01381893 0.09113843 0.22867114 -0.39392093  
 0.22898142 -0.50936782 0.22852923 -0.90105367 -1.21139742 -0.38511435  
 -0.97538069 -0.08838545 -0.26000985 -0.3335768 -0.35583611 0.69156192  
 -1.20663586 0.06137924 -0.7459055 0.95927124 0.11693435 0.40957735  
 0.16045238 0.61642801 0.76854204 0.17932262 0.45277118 -0.82148312  
 -0.23579657 0.74565596 -0.43719931 -0.24071831 -0.38588072 -0.20029347]

In [68]:

y\_pred**.**shape

Out[68]:

(834,)

In [74]:

y\_pred[0]

Out[74]:

0.885119134257663

In [75]:

y\_test**.**shape

Out[75]:

(834,)

In [76]:

y\_test[0]

Out[76]:

1.162769196

In [79]:

(y\_pred[2] **-** y\_test[2])**\*\***2

Out[79]:

0.035351549786525834

In [ ]:

*# split the data into training and testing sets*  
X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(x\_train, y\_train, test\_size**=**0.2, random\_state**=**42)  
  
*# initialize the StandardScaler object*  
scaler **=** StandardScaler()  
  
*# fit and transform the training data*  
X\_train\_scaled **=** scaler**.**fit\_transform(X\_train)  
*#y\_train\_scaled = scaler.transform(y\_train.reshape(-1, 1)).ravel()*  
  
*# transform the test data using the scaler fitted on the training data*  
X\_test\_scaled **=** scaler**.**transform(X\_test)  
*#y\_test\_scaled = scaler.transform(y\_test.reshape(-1, 1)).ravel()*  
  
*# initialize the Elastic Net regression model*  
model **=** ElasticNet(alpha**=**0.1, l1\_ratio**=**0.5)  
  
*# train the model on the training set*  
model**.**fit(X\_train\_scaled, y\_train)  
  
*# make predictions on the test set*  
y\_pred **=** model**.**predict(X\_test\_scaled)  
  
*# print the model coefficients*  
print('Model coefficients:', model**.**coef\_)  
  
*# print the model intercept*  
print('Model intercept:', model**.**intercept\_)  
  
*# print the mean squared error of the model predictions*  
mse **=** mean\_squared\_error(y\_test, y\_pred)  
print('Mean squared error:', mse)

In [ ]: