```
# Using google colab - this first step is for loading in the data from my personal Dri

# Login with google credentials

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Handle errors from too many requests

import logging
logging.getLogger('googleapiclient.discovery_cache').setLevel(logging.ERROR)

# The ID for my personal Drive folder is 1BVUuroPvozFxMjMIYrGOFtI4r6erSBCx
# I am now listing the ID numbers for the files in this folder to find the data files

#file_list = drive.ListFile({'q': "'1BVUuroPvozFxMjMIYrGOFtI4r6erSBCx' in parents and
#for file1 in file_list:
#  print('title: %s, id: %s' % (file1['title'], file1['id']))

# Data ID: 1F2KojI0d-ZnN8ssQFUWSyZA8I0mAgMEf

# Now that I have the ID files, load the files

data_downloaded = drive.CreateFile({'id': '1dwQLnIskShTXwSeMONhu__bYFf_f8-t6'})
data_downloaded.GetContentFile('sc_train.csv')

data_downloaded = drive.CreateFile({'id': '1IcNFIYUDKz1UxFL8W_JNjz9TzjAlAOVa'})
data_downloaded.GetContentFile('sc_unique_m.csv')
```

```
# Load the data into pandas

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import io

train = pd.read_csv('sc_train.csv',low_memory=False, lineterminator='\n')
unique = pd.read_csv('sc_unique_m.csv',low_memory=False, lineterminator='\n')
```

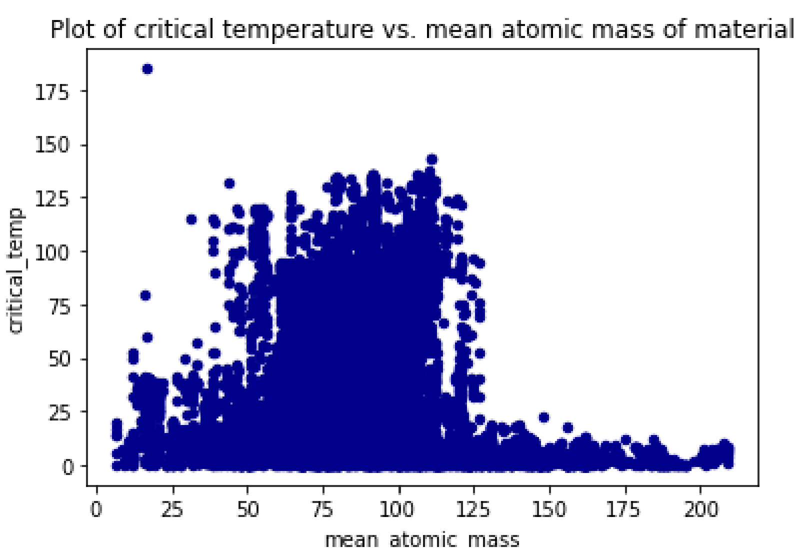
```
unique.shape
```

```
(21263, 88)
```

```
import torch
from torch.nn import functional as F

class LinearRegression(torch.nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearRegression, self).__init__()
        self.linear = torch.nn.Linear(input_dim, output_dim)

    def forward(self, x):
        outputs = self.linear(x)
        outputs = outputs.view(-1, 1)
        return outputs
```

```
# merge the two dataframes, drop material string
merge_df = pd.concat([train, unique], axis=1, sort=False)
merge_df = merge_df.drop(['material\r'], axis=1)
# Create feature identifying high-temp superconductors
merge_df['is_highTc'] = merge_df['critical_temp'] > 73

high_Tc_df = merge_df[merge_df['is_highTc']]
```

```
ax1 = merge_df.plot.scatter(x='mean_atomic_mass',
                      y='critical_temp',
                      c='DarkBlue',
                      title = 'Plot of critical temperature vs. mean atomic mass of ma
```




```
merge_df.describe()
```

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_ma
count	21263.000000	21263.000000	21263.000000	21263.000
mean	4.115224	87.557631	72.988310	71.290
std	1.439295	29.676497	33.490406	31.030
min	1.000000	6.941000	6.423452	5.320
25%	3.000000	72.458076	52.143839	58.041
50%	4.000000	84.922750	60.696571	66.361
75%	5.000000	100.404410	86.103540	78.116
max	9.000000	208.980400	208.980400	208.980

8 rows × 169 columns

```
high_Tc_df.describe()
```

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_ma
count	4371.000000	4371.000000	4371.000000	4371.000
mean	5.179822	86.365804	62.356138	64.431
std	0.936504	13.869704	15.820267	10.019
min	2.000000	16.157213	11.360293	5.685
25%	5.000000	76.444563	51.354450	59.356
50%	5.000000	86.671183	56.484598	64.570
75%	6.000000	95.450680	73.710253	70.446
max	9.000000	126.791862	152.464120	96.250

8 rows × 169 columns

```
# drop outlier
merge_df = merge_df[merge_df['critical_temp'] < 180]
merge_df.describe()
```

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_ma
count	21262.000000	21262.000000	21262.000000	21262.000
mean	4.115323	87.560971	72.991209	71.293
std	1.439255	29.673198	33.488527	31.027
min	1.000000	6.941000	6.423452	5.320
25%	3.000000	72.458076	52.144276	58.041
50%	4.000000	84.922750	60.697414	66.361
75%	5.000000	100.404410	86.103540	78.116
max	9.000000	208.980400	208.980400	208.980

8 rows × 169 columns

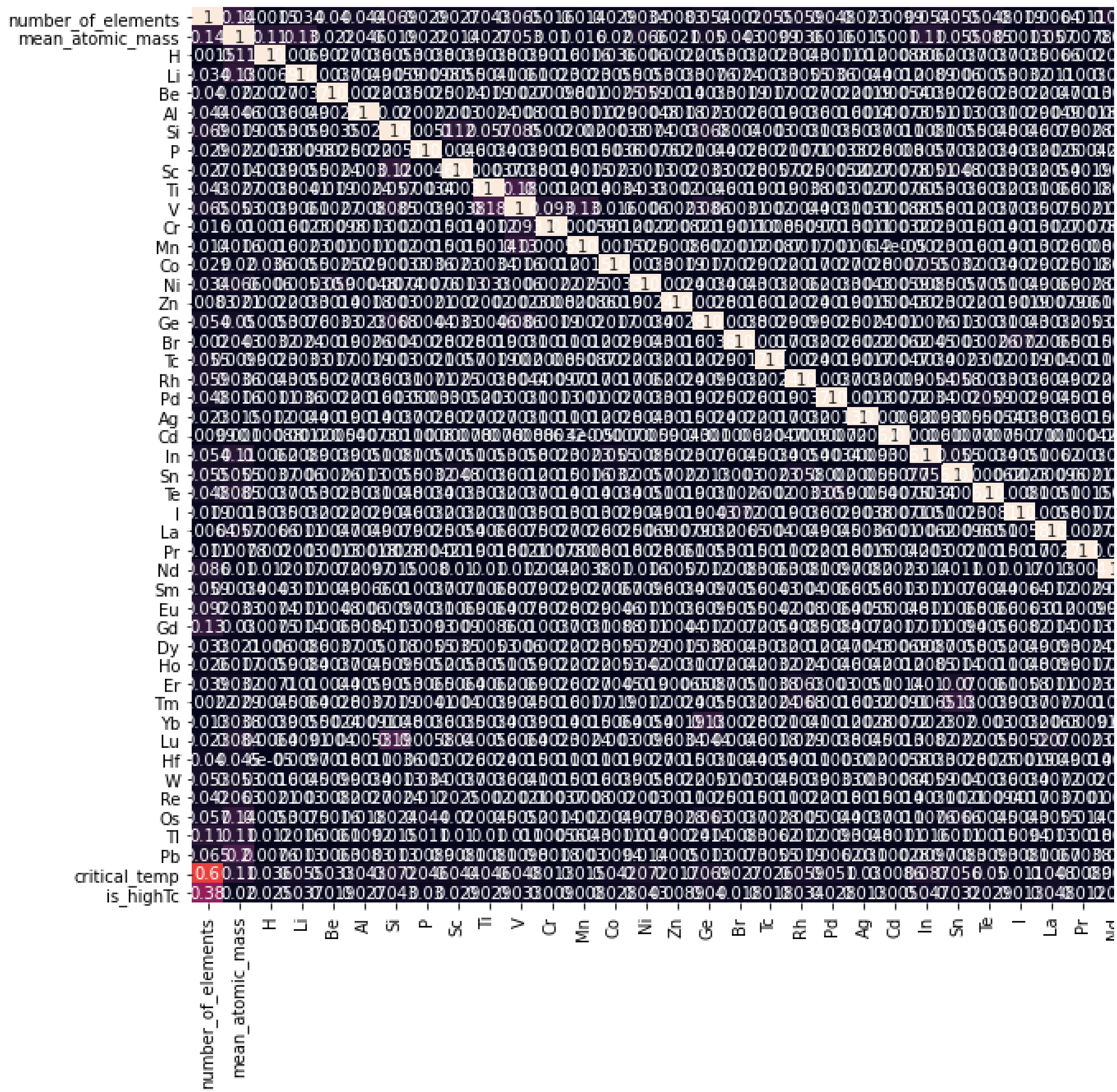
```
#normalize
merge_df = (merge_df-merge_df.min())/(merge_df.max()-merge_df.min())
# fix any NA values created by division by zero
merge_df = merge_df.fillna(0)


#drop cols with one value
for col in merge_df.columns:
    if len(merge_df[col].unique()) == 1:
        merge_df.drop(col,inplace=True,axis=1)


#print(merge_df.nunique())

## 9 columns with only 1 value


# Create correlation matrix

features = list(merge_df.columns.values.tolist())
corrMat = merge_df[features].corr().abs()

# Select upper triangle of correlation matrix
upper = corrMat.where(np.triu(np.ones(corrMat.shape), k=1).astype(np.bool))

# Find index of feature columns with correlation greater than 0.95
to_drop = [column for column in upper.columns if any(upper[column] > 0.2)]

# make sure I don't drop my target variables
if 'critical_temp' in to_drop: to_drop.remove('critical_temp')
if 'is_highTc' in to_drop: to_drop.remove('is_highTc')
```

```
print(len(to_drop)) # 55
#to_drop

merge_df = merge_df.drop(merge_df[to_drop], axis=1)
```

```
114
```

```
import seaborn as sn

features = list(merge_df.columns.values.tolist())
corrMat = merge_df[features].corr().abs()

plt.figure(figsize=(20,10))
sn.heatmap(corrMat, annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f866a5c74e0>
```




```
## train test split

train_df = merge_df.sample(frac=0.8, random_state=np.random.seed())
test_df = merge_df.drop(train_df.index)

# set up train and test data
X_train = train_df.drop(['critical_temp', 'is_highTc'], axis=1).to_numpy()
X_test = test_df.drop(['critical_temp', 'is_highTc'], axis=1).to_numpy()

X_train_high = train_df[train_df['is_highTc'] == 1].drop(['critical_temp', 'is_highTc'
X_test_high = test_df[test_df['is_highTc'] == 1].drop(['critical_temp', 'is_highTc'],
```

```
# set up target variable
y_train = train_df['critical_temp'].to_numpy()
y_test = test_df['critical_temp'].to_numpy()

# Set up alternative target - is high_T SC or not
y_high_temp_train = train_df['is_highTc'].to_numpy()
y_high_temp_test = test_df['is_highTc'].to_numpy()
#convert to Torch

X_torch = torch.from_numpy(X_train).float()
X_torch_high = torch.from_numpy(X_train_high).float()
y_torch = torch.from_numpy(y_train).float()
y_torch_highTC = torch.from_numpy(y_high_temp_train).float()
type(X_torch)

print(sum(sum(torch.isnan(X_torch))))

## No nans
```

```
tensor(0)
```

```
input_dim = X_train.shape[1]
output_dim = 1
lr_rate = 1e-3


y_train*185
```

```
array([  8.02071912,  32.34243316,  10.09065223, ...,  20.69905942,
       104.53135036, 115.13975755])
```

```
def MAPELoss(output, target):
  return 100*torch.mean(torch.abs((target - output) / (target + 0.001)))

def rmse(y, y_hat):

  #combined rmse value
  mse=torch.mean((y-y_hat)**2)
  rmse = torch.sqrt(mse)

  return rmse


# Linear model on all data
epochs = 2000
model = LinearRegression(input_dim, output_dim)

use_cuda = torch.cuda.is_available()

if use_cuda:
    print("Using GPU!")
    device = torch.device('cuda:0' if use_cuda else 'cpu')
```

```
    model.cuda()
    X_torch = X_torch.to(device)
    y_torch = y_torch.to(device)
else:
  print("Using CPU!")

criterion = torch.nn.MSELoss(reduction='mean')
optimizer = torch.optim.SGD(model.parameters(), lr=lr_rate)

for epoch in range(epochs):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(X_torch)
    #print(X_torch)
    #print(y_pred)
    #print(y_torch)
    # Compute and print loss
    loss = criterion(y_pred, y_torch)
    if epoch % 100 == 0:
      print(epoch, loss.item(), rmse(y_pred, y_torch))


    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
Using GPU!
0 0.12962457537651062 tensor(0.3600, device='cuda:0', grad_fn=<SqrtBackward>)
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/loss.py:431: UserWarning:
  return F.mse_loss(input, target, reduction=self.reduction)
100 0.10028598457574844 tensor(0.3167, device='cuda:0', grad_fn=<SqrtBackward>)
200 0.08297090232372284 tensor(0.2880, device='cuda:0', grad_fn=<SqrtBackward>)
300 0.07275111973285675 tensor(0.2697, device='cuda:0', grad_fn=<SqrtBackward>)
400 0.06671841442584991 tensor(0.2583, device='cuda:0', grad_fn=<SqrtBackward>)
500 0.06315656751394272 tensor(0.2513, device='cuda:0', grad_fn=<SqrtBackward>)
600 0.061052847653627396 tensor(0.2471, device='cuda:0', grad_fn=<SqrtBackward>)
700 0.05980958789587021 tensor(0.2446, device='cuda:0', grad_fn=<SqrtBackward>)
800 0.059074122458696365 tensor(0.2431, device='cuda:0', grad_fn=<SqrtBackward>)
900 0.058638330549001694 tensor(0.2422, device='cuda:0', grad_fn=<SqrtBackward>)
1000 0.05837938189506531 tensor(0.2416, device='cuda:0', grad_fn=<SqrtBackward>)
1100 0.058224812150001526 tensor(0.2413, device='cuda:0', grad_fn=<SqrtBackward>)
1200 0.05813184753060341 tensor(0.2411, device='cuda:0', grad_fn=<SqrtBackward>)
1300 0.05807524919509888 tensor(0.2410, device='cuda:0', grad_fn=<SqrtBackward>)
1400 0.05804012715816498 tensor(0.2409, device='cuda:0', grad_fn=<SqrtBackward>)
1500 0.05801767855882645 tensor(0.2409, device='cuda:0', grad_fn=<SqrtBackward>)
1600 0.058002740144729614 tensor(0.2408, device='cuda:0', grad_fn=<SqrtBackward>)
1700 0.057992227375507355 tensor(0.2408, device='cuda:0', grad_fn=<SqrtBackward>)
1800 0.05798434466123581 tensor(0.2408, device='cuda:0', grad_fn=<SqrtBackward>)
1900 0.057978030294179916 tensor(0.2408, device='cuda:0', grad_fn=<SqrtBackward>)
```

```
print(MAPELoss(y_pred, y_torch))
```

```
tensor(712.1074, device='cuda:0', grad_fn=<MulBackward0>)
```

```
X_test_torch = torch.from_numpy(X_test).float()
y_test_torch = torch.from_numpy(y_test.reshape(len(y_test))).float()

print("Test set!")

if use_cuda:
    print("Using GPU!")
    device = torch.device('cuda:0' if use_cuda else 'cpu')
    model.cuda()
    X_test_torch = X_test_torch.to(device)
    y_test_torch = y_test_torch.to(device)
else:
  print("Using CPU!")

test_preds = model(X_test_torch)
testLoss = criterion(test_preds, y_test_torch)
print(loss.item(), rmse(y_pred, y_torch))
```

```
Test set!
Using GPU!
0.05797269940376282 tensor(0.2408, device='cuda:0', grad_fn=<SqrtBackward>)
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/loss.py:431: UserWarning:
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
test_preds
```

```
tensor([[0.2370],
        [0.2637],
        [0.2358],
        ...,
        [0.2437],
        [0.2259],
        [0.2312]], device='cuda:0', grad_fn=<ViewBackward>)
```

```
def getImportanceTable(weights, features):

  weights_array = weights.tolist()
  weights_array = [item for sublist in weights_array for item in sublist]
  #weights_array = weights_array[1:]
  weights_sum = sum(list(map(abs, weights_array)))

  weights_array[:] = [x / weights_sum for x in weights_array]

  importances = list(zip(features, weights_array))
  importances = sorted(importances, key=lambda x : x[1], reverse=True)
  #importances = sorted(importances, key=lambda x : abs(x[1]), reverse=True)
```

```
    num = np.array(importances)
    reshaped = num.reshape(len(features),2)
    importanceTable = pd.DataFrame(reshaped, columns=['Feature','Importance'])

    return importanceTable


s = model.linear.weight.data.cpu().numpy()
importanceTable = getImportanceTable(s, train_df.drop(['critical_temp', 'is_highTc'],

importanceTable
```

1	Pr	0.041064281556999704
2	I	0.04019999882155142
3	mean_atomic_mass	0.038317976918262206
4	Rh	0.03582992634589807
5	Sn	0.03577510701287472
6	Gd	0.030568255718466584
7	Co	0.02860816039184059
8	number_of_elements	0.02743063717712784
9	Sc	0.026538998326407555
10	Te	0.025713763012875597
11	Mn	0.023902602251964684
12	Ti	0.02322779022861058
13	Ni	0.023224652125966035
14	Lu	0.022497923246778587
15	Al	0.018621918792401465
16	Er	0.01846746202285182
17	Tc	0.018235979292213298
18	Ge	0.016889773956645492
19	Pb	0.010648071731272909
20	Br	0.00977737998346116
21	Sm	0.00817147255599938
22	In	0.002433625644220945
23	La	0.001390502117752586
24	Os	-0.00159275015563155
25	Zn	-0.003428835537777344
26	Eu	-0.0038849001185545726
27	Nd	-0.007904398600452703
28	H	-0.008437590086405584
29	Li	-0.01163916808635421
30	Si	-0.012322585793922964
31	Re	-0.01402616639844002
32	Yb	0.016385159554229048

32	Yb	-0.016385159554229048
33	Ho	-0.01982766565246884
34	Tm	-0.02208966573194638
35	Dy	-0.024206409144849235
36	Cr	-0.02469367830073721
37	Pd	-0.025045077251339514
38	W	-0.02597203207359606
39	V	-0.028727885969391152
40	Be	-0.029509879727915984
41	Cd	-0.03191099950190225
42	Hf	-0.036622955943893684
43	Tl	-0.039921043987972
44	Ag	-0.04215055345931816

```
# linear model on High-Tc data

epochs = 1000
model = LinearRegression(input_dim, output_dim)

use_cuda = torch.cuda.is_available()

if use_cuda:
    print("Using GPU!")
    device = torch.device('cuda:0' if use_cuda else 'cpu')
    model.cuda()
    X_torch_high = X_torch_high.to(device)
    y_torch_highTC = y_torch_highTC.to(device)
else:
  print("Using CPU!")

criterion = torch.nn.MSELoss(reduction='mean')
optimizer = torch.optim.SGD(model.parameters(), lr=lr_rate)

for epoch in range(epochs):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(X_torch_high)
    #print(X_torch)
    #print(y_pred)
    #print(y_torch)
    # Compute and print loss
    loss = criterion(y_pred, y_torch_highTC)
    if epoch % 100 == 0: print(epoch, loss.item())

    # Zero gradients, perform a backward pass, and update the weights.
```

```
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

X_test_torch = torch.from_numpy(X_test_high).float()
y_test_torch_highTC = torch.from_numpy(y_high_temp_test).float()

print("Test set!")

if use_cuda:
    print("Using GPU!")
    device = torch.device('cuda:0' if use_cuda else 'cpu')
    model.cuda()
    X_test_torch = X_test_torch.to(device)
    y_test_torch_highTC = y_test_torch_highTC.to(device)
else:
  print("Using CPU!")

test_preds = model(X_test_torch)
testLoss = criterion(test_preds, y_test_torch_highTC)
print(testLoss.item())
```

```
Using GPU!
0 0.25091254711151123
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/loss.py:431: UserWarning:
  return F.mse_loss(input, target, reduction=self.reduction)
100 0.21296431124210358
200 0.19156567752361298
300 0.17949886620044708
400 0.17269398272037506
500 0.16885614395141602
600 0.16669131815433502
700 0.16546986997127533
800 0.1647803634405136
900 0.16439080238342285
Test set!
Using GPU!
0.16357533633708954
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/loss.py:431: UserWarning:
  return F.mse_loss(input, target, reduction=self.reduction)
```

```
s = model.linear.weight.data.cpu().numpy()
importanceTable = getImportanceTable(s, train_df.drop(['critical_temp', 'is_highTc'],

importanceTable
```

```
X_torch_high.shape
```

```
torch.Size([3499, 45])
```

```
import torch
```

```
from torch.nn import functional as F

class LogisticRegression(torch.nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearRegression, self).__init__()
        self.linear = torch.nn.Linear(input_dim, output_dim)

    def forward(self, x):
        outputs = F.softmax(self.linear(x))
        outputs = outputs.view(-1, 1)
        return outputs


# Logistic model - doesn't work

use_cuda = torch.cuda.is_available()
use_cuda = False

device = torch.device('cpu')

y_high_temp_train = train_df['is_highTc'].to_numpy()
y_high_temp_test = test_df['is_highTc'].to_numpy()
y_train_torch = torch.from_numpy(y_high_temp_train.reshape(len(y_train))).long()
y_test_torch = torch.from_numpy(y_high_temp_test.reshape(len(y_test))).long()

X_torch_high = torch.from_numpy(X_train).float()

epochs = 100
model = LinearRegression(input_dim, output_dim)



if use_cuda:
    print("Using GPU!")
    device = torch.device('cuda:0' if use_cuda else 'cpu')
    model.cuda()
    X_torch_high = X_torch_high.to(device)
    y_train_torch = y_train_torch.to(device)
else:
  print("Using CPU!")

criterion = torch.nn.CrossEntropyLoss(reduction='mean')
optimizer = torch.optim.SGD(model.parameters(), lr=lr_rate)

for epoch in range(epochs):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(X_torch)
    #print(X_torch)
    #print(y_pred)
    #print(y_torch)
    # Compute and print loss
    loss = criterion(y_pred, y_train_torch)
```

```
    if epoch % 100 == 0:
      print(epoch, loss.item(), rmse(y_pred, y_train_torch))

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

X_test_torch = torch.from_numpy(X_test_high).float()

print("Test set!")

if use_cuda:
    print("Using GPU!")
    device = torch.device('cuda:0' if use_cuda else 'cpu')
    model.cuda()
    X_test_torch = X_test_torch.to(device)
    y_test_torch = y_test_torch.to(device)
else:
  print("Using CPU!")

test_preds = model(X_test_torch)
testLoss = criterion(test_preds, y_test_torch)
print(testLoss.item())
print(epoch, testLoss.item(), rmse(test_preds, y_test_torch))
```

```
y_torch_highTC
```

```
tensor([0., 0., 0.,  ..., 0., 1., 1.], device='cuda:0')
```