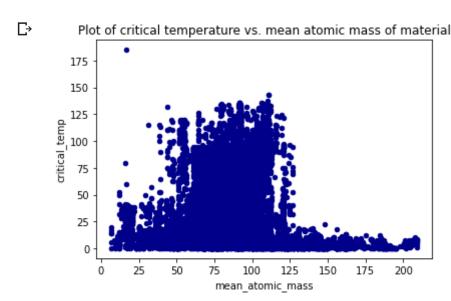
```
# Using google colab - this first step is for loading in the data from my personal Dri
# Login with google credentials
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
auth.authenticate user()
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
```

```
C→ (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 mass of
```



merge df.describe()

 \Box

	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()</pre>
```

С→

	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')
```

number of elements 🕆

1 0.194 00 D50 3 4 0 4 0 4 40 6 50 2 50 2 50 2 70 4 30 6 55 0 165 0 194 0 2 50 B 40 0 B 30 5 40 0 2 50 5 50 5 50 0 4 50 0 2 50 0 5 40 0 5 50 0 4 50 0 5 50 0 5 50 0 5 50 0 4 50 0 5 50 5 50 0 5 50 0 5 50 0 5 50 0 5 50 0 5 50 0 5 50 0 5 50 0 5 50 0 5

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

```
0.110.1B 02204501902201402705B 0D 016 0D 06502D 05 043009D 350160150D 1D 0550850D D3D 570
                                  mean atomic mass
                                                                                                                                 1 005902703903903903903903903903900700002703903902904900007070087057037037037039060002
                                                                                                        .04.022002703 1 00220350250250240 D90027009601002505900140350 D90D7002702270290390540390260280220
                                                                                         Be
                                                                                                         04404603603902 \frac{1}{2}02020303040030130100290048018010303601903601601400780510130030029043001
                                                                                           ΑI
                                                                                                          Si
                                                                                                          0 219 0 122 0 328 0 948 0 2 10 0 32 0 0 5 <u>1</u> 0 0 0 4 6 0 3 9 0 1 6 0 1 6 0 1 6 0 1 8 0 1 8 0 0 2 0 0 4 9 0 2 8 0 2 10 0 7 10 0 328 0 2 6 0 10 8 0 15 70 0 32 0 0 34 0 0 32 0 0
                                                                                                          02070 040 038 9 035 9 0 2040 0 10 10 20 0 4
                                                                                                                                                                                                            1 00 0 300 3 8 0 1 4 0 0 5 0 2 3 0 0 0 3 0 0 2 0 2 3 0 0 2 6 0 5 7 0 2 5 0 0 5 7 0 2 0 0 0 7 1 0 5 1 0 4 6 0 0 3 6 0 3 2 0 5
                                                                                                          0.4302770380400199024957003400110.18001120140840330020046019019083003700700705580360312031056
                                                                                                          0 655 0 553 0 03 9 0 15 10 0 270 0 13 0 8 95 0 0 3 9 0 3 18 1 2 0 0 9 15 0 15 0 0 16 0 0 60 0 250 8 60 0 10 0 20 0 0 4 4 0 0 10 0 0 3 10 0 0 10 0 0 10 0 0 10 0 0 10 0 0 10 0 0 10 0 0 10 0 0 10 0 0 10 0 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10 0 10
                                                                                                          016.01.00160XX009X)DX00X00180190DX0911 0005X01200XX008X0190XX008609X01800XX0280180190190180X
                                                                                           Or
                                                                                                                                                                                                                                                             1 00 D502 90 036 020 01320 08370 D73 013 0614e 03502 901 901 901 902 900 8
                                                                                                          0 104 0 D60 01 6 0 22 30 0 10 0 D10 0 20 01 6 0 104 D30 0 0
                                                                                         Μn
                                                                                                         029.02.036005802502900330360230084016001201 1 0038001901700290220017002702800705503820034029028018
                                                                                                          03#406#50060093059004#07#007#0D303300600D2025003
                                                                                                                                                                                                                                                                                      1000290390490320620390480059039057030049069
                                                                                                        00.837)210.02 0.03.801 40.030030021002.002002330080033611 90.2
                                                                                                                                                                                                                                                                                                 1 00280160120240190XE004E02802200190XE00XE0E
                                                                                         Zn
                                                                                                         00204300 B202400190 26040028028019030010010012002904801603
                                                                                                                                                                                                                                                                                                                          1 001 003 002 602220 06320 450 030 0260 0220 069
                                                                                                          0 $50 039 9 032 8 0 8 30 D 70 0 D 90 0 30 032 00 03 70 D 90 00 00 0 8 8 0 6 370 2 10 03 10 01 10 032 9 0 1
                                                                                                                                                                                                                                                                                                                                                 .00290190130043039023002001900
                                                                                                          O 599 O BATO CARO TERRO TERRO TO DE PROPERCIO DE PORTO DE
                                                                                                                                                                                                                                                                                                                                                   1 00370370370790540580038038036049020
                                                                                                          0480 D600 D10 B60 02 20 01 60 33000 BB 03 20 30 03 00 D30 010 02 20 03 B 01 B 02 B 02 B 01 B 0
                                                                                                          023301350122034403190314033902403290329033031031202403490319032403290319032203
                                                                                                        0.09900 D.003XD1X0D5407XX1 D00XXD76D76D76D666329050CXXD59046DD0D6204XD030D720
                                                                                                          054.10.0062089039050080050050050058005802902305008902907604903405408980
                                                                                          Sn
                                                                                                          04208500300590290300490390390390390390190190390500190300900905003
                                                                                                         0.09903 \pm 0.03803 \times 0.02803 \times 0.02803 \times 0.03803 \times 0.03
                                                                                                        006405700660D10044004903790250540166037502702602590639007390390065004004903450B600100632009355005
                                                                                                          O DDO 0780 02 0 03:00 XXX 0 XXX 0 4201 9 01 8 0 220 0 7XX 08 01 8 02 8 0 05 0 0 5 0 01 9 01 0 02 2 0 01 8 0 XXX 0 420 1 9 01 9 01 7 0 0
                                                                                                         086 OD 0120 D7007 20 970 15 008 010 OD 0 D2004 20 380 D 0 D600 970 D2008 B005 B008 20 20 20 14 01 D 0 D 0 17 0 10
                                                                                         Nd
                                                                                                         0.5900340401010049061501003407001680790290240164086034094056042004006605601031010076044064010201
                                                                                         Εu
                                                                                                          0920330074DD0043D060097003D069064078028029046DD0036099038042D00060640530048DD00680660630
                                                                                         Gd
                                                                                                         0333020.006003860B70050D600B50B500B3006002B002B0D5D2900D5DB6004B003B001B004B004B006BD38705B005B005B004B003B002
                                                                                                          00 DZD D9D 034 9 035 9 032 9 0 B 70 D9D 0 4 D0D 4D 03: 9 034 9 03 9 03 PD 109 0 DZD 02 9 03: 9 03 PD 03: 9 03 PD 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 03: 9 0
                                                                                         Tm
                                                                                                         0230840054091040053190058040056054028024030960B40440346018029003804901808202200580590
                                                                                                          04\,0452-05009701\,801\,00\,8600502\,602\,602\,601\,001\,001\,001\,902\,701\,903\,004405\,401\,000\,500\,700\,7813\,802\,802500\,1904\,901\,4
                                                                                          Hf
                                                                                                          0530530160460960340D3640D364003604D0160160390560200500460390690800600340590400360340
                                                                                                          042 063 0 210 03 003 20 02 70 240 12 0 25 0 02 0 210 0370 08 002 0 030 01 0 002 8 01 8 01 8 01 8 01 8 01 9 03 0 0 210 0 990 1 70
                                                                                                         05 V 14 00 5 B 0 7 B 0 D 60 1 B 0 2 4 0 4 4 0 2 0 0 4 B 0 5 D 0 D 40 D 2 0 0 4 B 0 7 B 0 D 2 0 0 3 D 0 3 D 0 5 D 0 4 D 0 3 D 0 1 D 0 D 0 5 D 0 4 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5 D 0 5
                                                                                                           110 1 D 0102 0 D60 05 00 922 15 0 1 D 0 10 0 D 0 D 0 0 00 05 00 480 D 10 0 40 00 2 4 D 40 08 8 0 6 20 D20 09 8 0 4 8 0 D 10 16 0 D 10 0 1 8 0 9 4 D 15 0 1 8
                                                                                                        . 0.650. 12. 0.0 16:50 D.30 016:80 8:50 D.30 018:9 018:10 018:9 018:0 018:0 D.30 019:0 D.30 019:0 0 D.30 017:0 D.30 017:0 D.30 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018:0 018
                                                                                                                 0.1 D.03660550330433072046044904504801301504207020107069020702605905 D.0300860807.056015.010.0460089
                                                        critical_temp
                                                                     is highTc - 0.380.020.025.0307.0109.0207.04 B. 0 B. 0 209.0209.033.0009.008.02 B. 0 430.0809.04.0108.0304.0208.0103.0005.0407.0302.0209.0103.0408.0102.0
                                                                                                                      mass
                                                                                                                                                                                                                                                           E & B B
                                                                                                          number of elements
                                                                                                                      mean atomic
## 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
\Gamma 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')
```

https://colab.research.google.com/drive/1fydc7wvad0oNKG-omdGT2Bee5dfgzK0E#scrollTo=Kkjr3PkEzb0u&printMode=true

```
model.cuda()
    X_torch = X_torch.to(device)
    y_torch = y_torch.to(device)
  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
\vdash 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)
```

С→

1	Pr	0.041064281556999704
2	1	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	Н	-0.008437590086405584
29	Li	-0.01163916808635421
30	Si	-0.012322585793922964
31	Re	-0.01402616639844002
22	Vh	U U1638E1EUEE433UU48

٥Z

ΙU

-U.U 10303 139334229U40

```
33
                       Но
                             -0.01982766565246884
     34
                       Tm
                             -0.02208966573194638
     35
                        Dγ
                            -0.024206409144849235
     36
                        Cr
                             -0.02469367830073721
     37
                        Pd
                            -0.025045077251339514
     38
                        W
                             -0.02597203207359606
     39
                             -0.028727885969391152
     40
                        Be
                            -0.029509879727915984
     41
                       Cd
                             -0.03191099950190225
     42
                        Hf
                            -0.036622955943893684
     43
                        ΤI
                               -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())
```

```
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')
```