1. Ridge Regression :

A. The ridge regression problem is : $\min \beta \; ||X\beta - y||^2 + \lambda ||\beta||^2$
By solving it, the closed form solution is

$$\beta = (\lambda I + X^T X)^{-1} X^T y$$

Augmented data of

Matrix X' = $\begin{pmatrix} X \\ \sqrt{\lambda} I \end{pmatrix}$ and response vector y' = $\begin{pmatrix} y \\ 0_k \end{pmatrix}$

Where I is k x k identity matrix and 0_k is k x 1 vector. Calculating $X^T X$

$$X'^T X' = (X^T \sqrt{\lambda} I) . \begin{pmatrix} X \\ \sqrt{\lambda} I \end{pmatrix} = X^T X + \lambda I$$

And $X'^T y$ can be calculated as

$$X'^T y = (X^T \sqrt{\lambda} I) . \begin{pmatrix} y \\ 0_k \end{pmatrix} = X^T y$$

We can calculate the least squares regression closed solution

$$\hat{\beta} = (X'^T X')^{-1} X'^T y'$$
$$\Rightarrow \hat{\beta} = (X^T X + \lambda I)^{-1} X^T y$$

Which is the closed form solution to ridge regression with parameter λ. We proved that the ridge regression estimates can be obtained by ordinary least squares regression on an augmented data set.

2a) Preprocessing the data.

```
def preprocess_data(trainx, valx, testx):
    train_mean = np.mean(trainx, axis=0)
    train_std = np.std(trainx, axis=0)
    trainx = (trainx - train_mean) / train_std
    valx = (valx - train_mean) / train_std
    testx = (testx - train_mean) / train_std
    return trainx, valx, testx
```

Initially, I scaled the features to have similar values by standardizing it (calculated the mean and standard deviation of each feature on the training data, then standardized the training, validation, and test data using those statistics.) This can also be achieved using standard scalar but I preferred this method. I chose to standardize because it prevents large value features from dominating. It makes all the features equally important and can improve the performance of the model.

Later on I did some feature engineering to add some features and removed the outliers to improve the accuracy of the model.

```
In [3]: df["hours"]=(df['date'].str.split(':').str[0].str.split(" ").str[1]).astype(str).astype(int)
        df['hour*lights'] = df.hours * df.lights
        df['hour_avg'] = list(map(dict(df.groupby('hours')["Appliances"].mean()).get, df.hours))
        df_val["hours"]=(df_val['date'].str.split(':').str[0].str.split(" ").str[1]).astype(str).astype(int)
        df_val['hour*lights'] = df_val.hours * df_val.lights
        df_val['hour_avg'] = list(map(dict(df_val.groupby('hours')["Appliances"].mean()).get, df_val.hours))
        df_test["hours"]=(df_test['date'].str.split(':').str[0].str.split(" ").str[1]).astype(str).astype(int)
        df_test['hour*lights'] = df_test.hours * df_test.lights
        df_test['hour_avg'] = list(map(dict(df_test.groupby('hours')["Appliances"].mean()).get, df_test.hours))

In [6]: feature_set = ['low_consum','high_consum','hours','t6','rh_6','lights','hour*lights',
                       'tdewpoint','visibility','press_mm_hg','windspeed']

In [37]: corr_matrix_train = df.corr(method='pearson')
         matrix_train=corr_matrix_train["Appliances"].sort_values(ascending=False)
         print((matrix_train))
         corr_matrix_val = df_val.corr(method='pearson')
         matrix_val=corr_matrix_val["Appliances"].sort_values(ascending=False)
         print((matrix_val))
         corr_matrix_test = df_test.corr(method='pearson')
         matrix_test=corr_matrix_test["Appliances"].sort_values(ascending=False)
         print((matrix_test))
```

```
: X_train=df.drop(['Appliances', 'date'], axis = 1).values
  Y_train = df[['Appliances']].values
  X_val=df_val.drop(['Appliances', 'date'], axis = 1).values
  Y_val = df_val['Appliances'].values
  X_test=df_test.drop(['Appliances', 'date'], axis = 1).values
  Y_test = df_test['Appliances'].values
  train_mean = np.mean(X_train, axis=0)
  train_std = np.std(X_train, axis=0)
  X_train = (X_train - train_mean) / train_std
  X_val = (X_val - train_mean) / train_std
  X_test = (X_test - train_mean) / train_std
```

2e) Comparing the RMSE and R^2 of 2c and 2d

Model	Train RMSE	Train R^2	Val RMSE	Val R^2	Test RMSE	Test R^2
Train only (2c)	94.164	0.25269	91.610	0.12024	88.04	0.0611
Train + val (2d)	95.021	0.2390	85.0001	0.24262	82.696	0.1718

Result :

The model trained on training + validation data(2d) performs better on test data rather than only trained on training data(2c). This is because 2d is trained with more data. But the train metrics decrease when trained with both the data. We can also see a slight increase in RMSE in train metrics. Coming to the Validation metrics, we can see a slight decrease in RMSE and a increase in R^2, which is also expected because the model is trained with more data. Overall, we can conclude that 2d gives a better performance than 2c.

2c ->

```
In [208]: model7 = LinearRegression()
data4 = {}
model7.fit(X_train, Y_train)
data4["train-rmse"] =sqrt(mean_squared_error(Y_train, model7.predict(X_train)))
data4["train-r2"] = metrics.r2_score(Y_train, model7.predict(X_train))
data4["val-rmse"] =sqrt(mean_squared_error(Y_val, model7.predict(X_val)))
data4["val-r2"] = metrics.r2_score(Y_val, model7.predict(X_val))
data4["test-rmse"] =sqrt(mean_squared_error(Y_test, model7.predict(X_test)))
data4["test-r2"] = metrics.r2_score(Y_test, model7.predict(X_test))
data4

Out[208]: {'train-rmse': 94.16443818232597,
 'train-r2': 0.2526942965780995,
 'val-rmse': 91.61037186132472,
 'val-r2': 0.12024864421105896,
 'test-rmse': 88.04916726513582,
 'test-r2': 0.06119046785291704}
```

2d->

```
In [212]: comb_datax = np.concatenate((X_train, X_val))
comb_datay = np.concatenate((Y_train, Y_val))
model = LinearRegression()
data3 = {}
model.fit(comb_datax, comb_datay)
data3["train-rmse"] =sqrt(mean_squared_error(Y_train, model.predict(X_train)))
data3["train-r2"] = metrics.r2_score(Y_train, model.predict(X_train))
data3["val-rmse"] =sqrt(mean_squared_error(Y_val, model.predict(X_val)))
data3["val-r2"] = metrics.r2_score(Y_val, model.predict(X_val))
data3["test-rmse"] =sqrt(mean_squared_error(Y_test, model.predict(X_test)))
data3["test-r2"] = metrics.r2_score(Y_test, model.predict(X_test))
data3

Out[212]: {'train-rmse': 95.02148918904848,
 'train-r2': 0.2390289718836781,
 'val-rmse': 85.00017194193263,
 'val-r2': 0.2426262282210745,
 'test-rmse': 82.69632325248155,
 'test-r2': 0.17186834357698244}
```

2h) Comparing RMSE and R^2 for different values of λ

λ	Train RMSE	Train R^2	Val RMSE	Val R^2	Test RMSE	Test R^2
1010	95.358	0.23362	88.71337	0.1750096656	82.95	0.16677
1015	95.363	0.23354	88.71336	0.175009877	82.947	0.16682
1020	95.368	0.2334	88.71337	0.1750096655	82.9450	0.16687

These are the values I tested λ on.

From the table we can see that there is a decrease in Train R^2 whereas Val R^2 and Test R^2 increased for increase in λ. I chose the λ because of sharp increase in Val R^2 at 1015 and a slight decrease in later value. This data is for Ridge regression.

In [217]:
```
model8 = Ridge(alpha=1010)
model8.fit(X_train, Y_train)
data5 = {}
data5["train-rmse"] =sqrt(mean_squared_error(Y_train, model8.predict(X_train)))
data5["train-r2"] = metrics.r2_score(Y_train, model8.predict(X_train))
data5["val-rmse"] =sqrt(mean_squared_error(Y_val, model8.predict(X_val)))
data5["val-r2"] = metrics.r2_score(Y_val, model8.predict(X_val))
data5["test-rmse"] =sqrt(mean_squared_error(Y_test, model8.predict(X_test)))
data5["test-r2"] = metrics.r2_score(Y_test, model8.predict(X_test))
data5
```

Out[217]:
```
{'train-rmse': 95.36813338271575,
 'train-r2': 0.23346670707893213,
 'val-rmse': 88.7133754131302,
 'val-r2': 0.17500966552596675,
 'test-rmse': 82.94507166042648,
 'test-r2': 0.1668788533804081}
```

In [219]:
```
model8 = Ridge(alpha=1015)
model8.fit(X_train, Y_train)
data5 = {}
data5["train-rmse"] =sqrt(mean_squared_error(Y_train, model8.predict(X_train)))
data5["train-r2"] = metrics.r2_score(Y_train, model8.predict(X_train))
data5["val-rmse"] =sqrt(mean_squared_error(Y_val, model8.predict(X_val)))
data5["val-r2"] = metrics.r2_score(Y_val, model8.predict(X_val))
data5["test-rmse"] =sqrt(mean_squared_error(Y_test, model8.predict(X_test)))
data5["test-r2"] = metrics.r2_score(Y_test, model8.predict(X_test))
data5
```

Out[219]:
```
{'train-rmse': 95.36335574564467,
 'train-r2': 0.23354350686548375,
 'val-rmse': 88.71336400061998,
 'val-r2': 0.1750098777873078,
 'test-rmse': 82.9475971322968,
 'test-r2': 0.16682811965966005}
```

In [220]:
```
model8 = Ridge(alpha=1020)
model8.fit(X_train, Y_train)
data5 = {}
data5["train-rmse"] =sqrt(mean_squared_error(Y_train, model8.predict(X_train)))
data5["train-r2"] = metrics.r2_score(Y_train, model8.predict(X_train))
data5["val-rmse"] =sqrt(mean_squared_error(Y_val, model8.predict(X_val)))
data5["val-r2"] = metrics.r2_score(Y_val, model8.predict(X_val))
data5["test-rmse"] =sqrt(mean_squared_error(Y_test, model8.predict(X_test)))
data5["test-r2"] = metrics.r2_score(Y_test, model8.predict(X_test))
data5
```

Out[220]:
```
{'train-rmse': 95.36813338271575,
 'train-r2': 0.23346670707893213,
 'val-rmse': 88.7133754131302,
 'val-r2': 0.17500966552596675,
 'test-rmse': 82.94507166042648,
 'test-r2': 0.1668788533804081}
```

For Lasso, I chose 7 as the R^2 values kept decreasing as the value of λ increased later on.

λ	Train RMSE	Train R^2	Val RMSE	Val R^2	Test RMSE	Test R^2
1	94.95	0.240117	89.7719	0.15520	85.25414	0.119847
10	98.677	0.1793	89.670	0.1571134	83.67966	0.15205
7	98.050	0.189735	89.0653	0.168450	84.0079	0.14539

In [227]:
```
model9 = Lasso(alpha = 1)
model9.fit(X_train, Y_train)
data6 = {}
data6["train-rmse"] =sqrt(mean_squared_error(Y_train, model9.predict(X_train)))
data6["train-r2"] = metrics.r2_score(Y_train, model9.predict(X_train))
data6["val-rmse"] =sqrt(mean_squared_error(Y_val, model9.predict(X_val)))
data6["val-r2"] = metrics.r2_score(Y_val, model9.predict(X_val))
data6["test-rmse"] =sqrt(mean_squared_error(Y_test, model9.predict(X_test)))
data6["test-r2"] = metrics.r2_score(Y_test, model9.predict(X_test))
data6
```

Out[227]:
```
{'train-rmse': 98.0507919899548,
 'train-r2': 0.18973576513962997,
 'val-rmse': 89.06535228802454,
 'val-r2': 0.16845025854750784,
 'test-rmse': 84.00794179055323,
 'test-r2': 0.14539058353012047}
```

In [228]:
```
model9 = Lasso(alpha = 10)
model9.fit(X_train, Y_train)
data6 = {}
data6["train-rmse"] =sqrt(mean_squared_error(Y_train, model9.predict(X_train)))
data6["train-r2"] = metrics.r2_score(Y_train, model9.predict(X_train))
data6["val-rmse"] =sqrt(mean_squared_error(Y_val, model9.predict(X_val)))
data6["val-r2"] = metrics.r2_score(Y_val, model9.predict(X_val))
data6["test-rmse"] =sqrt(mean_squared_error(Y_test, model9.predict(X_test)))
data6["test-r2"] = metrics.r2_score(Y_test, model9.predict(X_test))
data6
```

Out[228]:
```
{'train-rmse': 98.67700162145417,
 'train-r2': 0.17935307431419267,
 'val-rmse': 89.6704241982925,
 'val-r2': 0.15711349376385408,
 'test-rmse': 83.67966564890754,
 'test-r2': 0.1520566136751752}
```

In [229]:
```
model9 = Lasso(alpha = 7)
model9.fit(X_train, Y_train)
data6 = {}
data6["train-rmse"] =sqrt(mean_squared_error(Y_train, model9.predict(X_train)))
data6["train-r2"] = metrics.r2_score(Y_train, model9.predict(X_train))
data6["val-rmse"] =sqrt(mean_squared_error(Y_val, model9.predict(X_val)))
data6["val-r2"] = metrics.r2_score(Y_val, model9.predict(X_val))
data6["test-rmse"] =sqrt(mean_squared_error(Y_test, model9.predict(X_test)))
data6["test-r2"] = metrics.r2_score(Y_test, model9.predict(X_test))
data6
```

Out[229]:
```
{'train-rmse': 98.0507919899548,
 'train-r2': 0.18973576513962997,
 'val-rmse': 89.06535228802454,
 'val-r2': 0.16845025854750784,
 'test-rmse': 84.00794179055323,
 'test-r2': 0.14539058353012047}
```

2j) Comparing RMSE and R^2 for combinedly trained data with optimal parameters

Model	λ	Train RMSE	Train R^2	Val RMSE	Val R^2	Test RMSE	Test R^2
Ridge	1015	95.599	0.22973	85.405	0.23538	81.5823	0.194029
Lasso	7	98.3208	0.18526	88.7759	0.17384	84.2093	0.14128

Here, we used the optimal parameter λ from the previous question 2h and applied it in 2i. The values are produced above.

```
In [231]: comb_datax = np.concatenate((X_train, X_val))
          comb_datay = np.concatenate((Y_train, Y_val))
          model = Ridge(alpha = 1015)
          model.fit(comb_datax, comb_datay)
          data1= {}
          data1["train-rmse"] =sqrt(mean_squared_error(Y_train, model.predict(X_train)))
          data1["train-r2"] = metrics.r2_score(Y_train, model.predict(X_train))
          data1["val-rmse"] =sqrt(mean_squared_error(Y_val, model.predict(X_val)))
          data1["val-r2"] = metrics.r2_score(Y_val, model.predict(X_val))
          data1["test-rmse"] =sqrt(mean_squared_error(Y_test, model.predict(X_test)))
          data1["test-r2"] = metrics.r2_score(Y_test, model.predict(X_test))
          data1

Out[231]: {'train-rmse': 95.59971064405147,
           'train-r2': 0.22973952493994287,
           'val-rmse': 85.40540965779748,
           'val-r2': 0.2353874657286551,
           'test-rmse': 81.58233146326182,
           'test-r2': 0.1940293822818027 8}
```

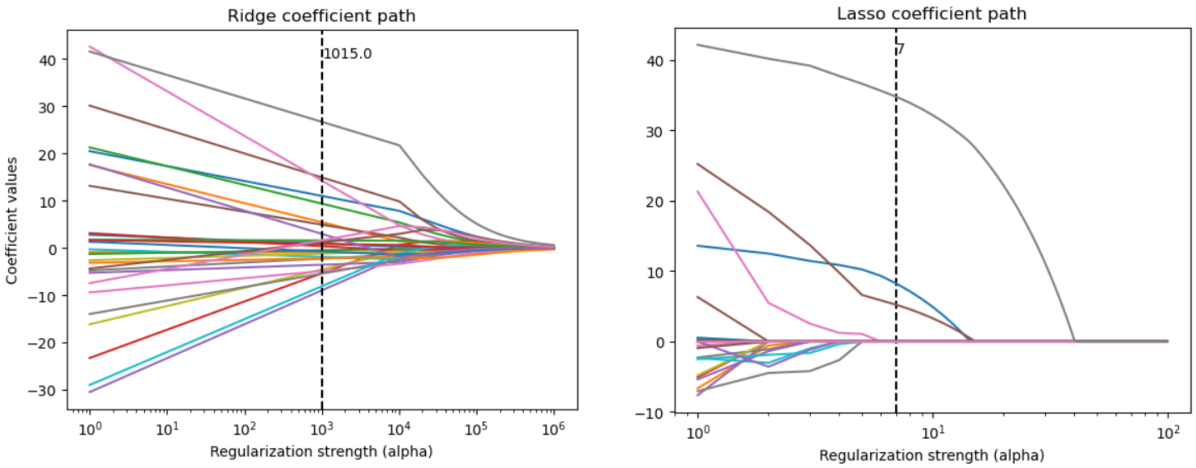
```
In [230]: comb_datax = np.concatenate((X_train, X_val))
          comb_datay = np.concatenate((Y_train, Y_val))
          model = Lasso(alpha = 7)
          model.fit(comb_datax, comb_datay)
          data1= {}
          data1["train-rmse"] =sqrt(mean_squared_error(Y_train, model.predict(X_train)))
          data1["train-r2"] = metrics.r2_score(Y_train, model.predict(X_train))
          data1["val-rmse"] =sqrt(mean_squared_error(Y_val, model.predict(X_val)))
          data1["val-r2"] = metrics.r2_score(Y_val, model.predict(X_val))
          data1["test-rmse"] =sqrt(mean_squared_error(Y_test, model.predict(X_test)))
          data1["test-r2"] = metrics.r2_score(Y_test, model.predict(X_test))
          data1

Out[230]: {'train-rmse': 98.32087385322708,
           'train-r2': 0.1852658559201099,
           'val-rmse': 88.7759830225751,
           'val-r2': 0.173844815285358,
           'test-rmse': 84.20939921311948,
           'test-r2': 0.14128683233875394}
```

For Ridge : There is a slight increase in RMSE of Train whereas it decreased in case of both Validation and Test (from 2h). Regarding R^2 for train it has decreased slightly whereas for Validation it went from 0.175009 to 0.23538 and for the test it increased from 0.16682 to 0.194029.

For Lasso : There is a slight increase in RMSE of Train whereas it decreased in case of both Validation and Test for the Lasso too (from 2h). Regarding R^2 for train it has decreased slightly whereas for Validation it went from 0.168450 to 0.17384 and for the test it decreased from 0.14539 to 0.14128.

So we can infer from the above information that when the model is trained with more data, it presents us with more accuracy in all the above three cases.

2k) Given coefficient path plots for both ridge and lasso :

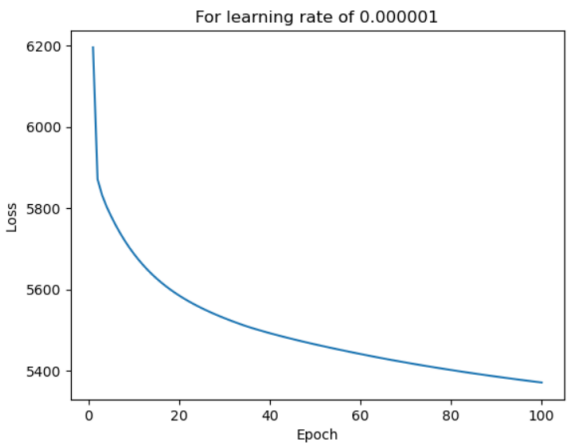


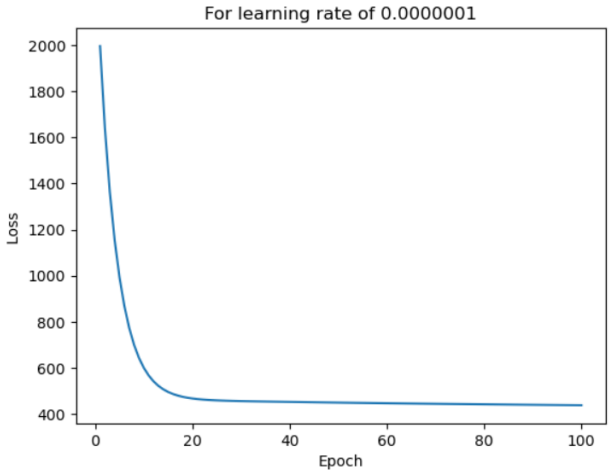
We can see a dotted line which represents the optimal value for both lasso and ridge.

2l) Three key observations that we can pick from the above plots :

1. The coefficients vary smoothly along the regularization path. As lambda increases, the coefficient values move towards zero. This shows the regularization is working as expected.
2. The lasso model sets more coefficients exactly to zero compared to ridge.
3. The optimal lambda is smaller for lasso than ridge, corresponding to less overall regularization strength.

3f) From the optimization parameter of ridge(1015), some of the good learning rates are 0.00001 and 0.000001.





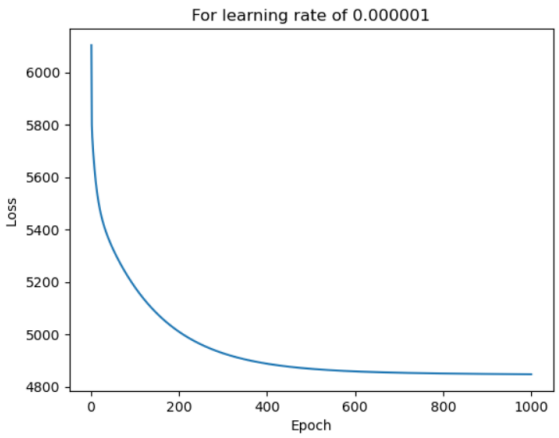
From the chosen learning rates, by observation I chose the learning rate of 0.0000001 to be optimal as Objective decreases smoothly over epochs and it does not diverge or become unstable.

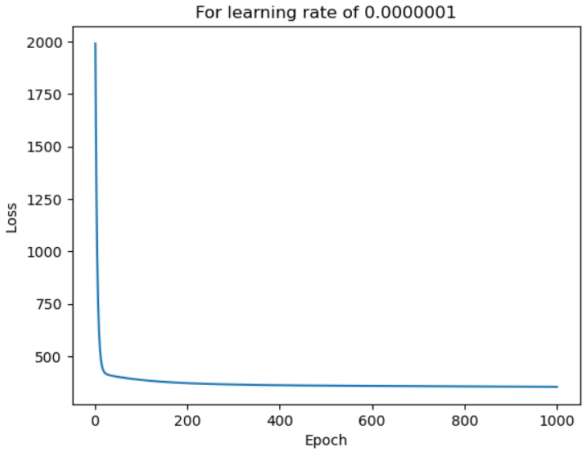
The RMSE values for Train, Validation and Test data are 105.81903171862118, 96.59776672570594, 90.0388759165188.

The R^2 values for Train, Validation and Test data are 0.05626073596552117, 0.021851429325226768, 0.01828118710364468

```
In [69]: elas_ridge1 = ElasticNet(1015, 0.5 ,0.0000001, 100, 100)
         dict2 = elas_ridge1.train(X_train, Y_train)
         plt.plot(dict2.keys(), dict2.values(), label = 'LR = 0.0000001')
         plt.xlabel('Epoch')
         plt.ylabel('Loss ')
         plt.title("For learning rate of 0.0000001")
```

From the optimization parameter of lasso (7), some of the good learning rates are 0.00001 and 0.000001.





From the chosen learning rates, by observation I chose the learning rate of 0.000001 to be optimal as Objective decreases smoothly over epochs and it does not diverge or become unstable. The RMSE values for train, validation and test data are 98.41135822347093, 89.74285386580931, 83.65010817548402 and The R^2 Values for Train, Validation and Test data are 0.18376557168723273, 0.15575129109868457, 0.15265553188314407.

```
In [55]: import matplotlib.pyplot as plt
         elas_lasso = ElasticNet(9, 0.5 ,0.000001, 100, 1000)
         dic = elas_lasso.train(X_train, Y_train)
         plt.plot(dic.keys(), dic.values())
         plt.xlabel('Epoch')
         plt.ylabel('Loss ')
         plt.title("For learning rate of 0.000001")
```

3g) For Ridge λ:

Alphas	Train R^2	Validation R^2	Test R^2	Train RMSE	Validation RMSE	Test RMSE
0.0	0.0562607521910835	0.021851438829579184	0.018281196451523707	105.8190308089561	96.59776625640136	90.03887548784589
0.11	0.0562607485854 0353	0.021851436717501 56	0.018281194374218046	105.8190310111 0387	96.59776636069 122	90.038875583106 5
0.22	0.0562607449797 2311	0.021851434605423 492	0.018281192296911 83	105.8190312132 5166	96.59776646498 112	90.038875678367 13
0.33	0.0562607413740 42356	0.021851432493345 09	0.018281190219605 28	105.8190314153 9946	96.59776656927 103	90.038875773627 78
0.44	0.0562607377683 6171	0.021851430381266 357	0.018281188142298066	105.8190316175 4727	96.59776667356 097	90.038875868888 47
0.55	0.0562607341626 8074	0.021851428269187 4	0.018281186064990 63	105.8190318196 9508	96.59776677785 091	90.038875964149 16
0.66	0.0562607305569 9943	0.021851426157107 89	0.018281183987682748	105.8190320218 4292	96.59776688214 087	90.038876059409 87
0.77	0.0562607269513 17785	0.021851424045027 934	0.018281181910374 31	105.8190322239 9079	96.59776698643 086	90.038876154670 6
0.88	0.0562607233456 3592	0.021851421932947 868	0.018281179833065653	105.8190324261 3865	96.59776709072 085	90.038876249931 36
1.0	0.0562607197399 5406	0.021851419820867 468	0.018281177755756328	105.8190326282 8652	96.59776719501 086	90.038876345192 14

```
]: alpha = np.linspace(0,1,10)
for i in alpha :
    elas_ridge = ElasticNet(1015, i ,0.0000001, 100, 100)
    elas_ridge.train(X_train, Y_train)
    b = elas_ridge.predict(X_train)
    b_val = elas_ridge.predict(X_val)
    b_test = elas_ridge.predict(X_test)
    r2_train = r2_score(Y_train, b)
    r2_val = r2_score(Y_val, b_val)
    r2_test = r2_score(Y_test, b_test)
    print ("The r2 values for Train, Validation and Test datas are ", r2_train, r2_val, r2_test)
    rmse = sqrt(mean_squared_error(Y_train,b))
    rmse_val = sqrt(mean_squared_error(Y_val,b_val))
    rmse_test = sqrt(mean_squared_error(Y_test,b_test))
    print ("The rmse values for Train, Validation and Test datas are ", rmse, rmse_val, rmse_test)
```

For Lasso λ :

Alphas	Train R^2	Validation R^2	Test R^2	Train RMSE	Validation RMSE	Test RMSE
0.0	0.1271943449820313	0.07895453603048042	0.07199037408780844	101.76455712717384	93.73573899687815	87.54125092216813
0.11	0.1271943445181503	0.07895453553103415	0.07199037359594362	101.76455715421662	93.73573902229275	87.5412509453675
0.22	0.1271943440542786 5	0.07895453503158778	0.07199037310407885	101.76455718125939	93.73573904770733	87.54125096856686
0.33	0.1271943435904025	0.07895453453214141	0.07199037261221397	101.76455720830214	93.735739073121 9	87.54125099176622
0.44	0.1271943431265 26	0.07895453403269515	0.07199037212034898	101.76455723534492	93.73573909853 65	87.54125101496558
0.55	0.1271943426626 4973	0.07895453353324888	0.07199037162848 42	101.76455726238 77	93.73573912395 06	87.54125103816494
0.66	0.1271943421987 7335	0.07895453303380251	0.07199037113661944	101.76455728943047	93.73573914936 66	87.5412510613643
0.77	0.1271943417348 9708	0.07895453253435636	0.07199037064475466	101.76455731647324	93.73573917478022	87.54125108456367
0.88	0.1271943412710 2092	0.07895453203490999	0.07199037015288999	101.76455734351 6	93.73573920019 48	87.54125110776302
1.0	0.1271943408071 4453	0.07895453153546383	0.07199036966102512	101.76455737055879	93.73573922560938	87.54125113096238

In [71]:
```
alphas = np.linspace(0,1,10)
for i in alphas :
    elas_lasso2 = ElasticNet(9, i ,0.000001, 100, 100)
    elas_lasso2.train(X_train, Y_train)
    c = elas_lasso2.predict(X_train)
    c_val = elas_lasso2.predict(X_val)
    c_test = elas_lasso2.predict(X_test)
    r2_train1 = r2_score(Y_train, c)
    r2_val1 = r2_score(Y_val, c_val)
    r2_test1 = r2_score(Y_test, c_test)
    print ("The r2 values for Train, Validation and Test datas are ", r2_train1, r2_val1, r2_test1)
    rmse1 = sqrt(mean_squared_error(Y_train,c))
    rmse_val1 = sqrt(mean_squared_error(Y_val,c_val))
    rmse_test1= sqrt(mean_squared_error(Y_test,c_test))
    print ("The rmse values for Train, Validation and Test datas are ", rmse1, rmse_val1, rmse_test1)
```

I have tested for both values of λ For Ridge and lasso, we can see when alpha value is near zero, it shows some possible lasso model behavior and when it increases towards 1, it tends towards ridge model.

3h) From my observations in 2h, 2j, when compared different values of λ to find the optimal parameter based on validation data performance. In 3g, we test models in a range of alphas(0,1) . From my observation, when we used optimal parameter and different λ values, we got better RMSE and R^2 compared to SGD variants. We can infer from the results that when we train the model using training and validation data with an optimal parameter, it showed good results.
From both lasso and ridge models, Ridge displayed a better performance than lasso when trained with both the data of train and validation.

Differences between SGD variants of Ridge and Lasso :

- SGD variants use stochastic gradient descent to optimize the regression coefficients, while standard implementations use closed-form solutions (for Ridge) or coordinate descent (for LASSO).
- For LASSO, the SGD variant typically uses a constant learning rate. So standard LASSO can converge faster and more precisely.
- SGD estimators have more hyperparameters to tune (learning rate, batch size, iterations) compared to the closed-form solutions.

3j)

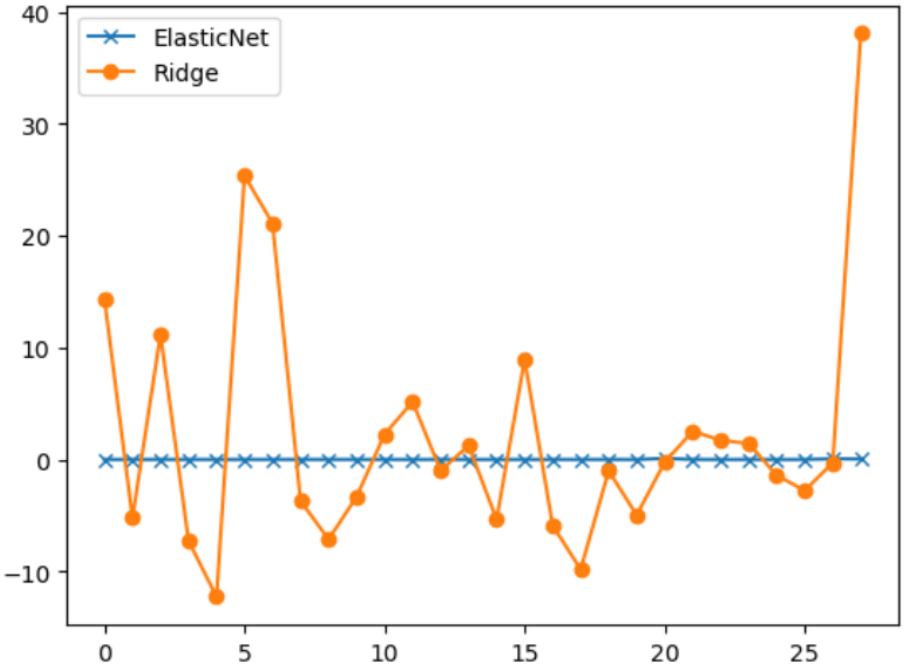
For final coefficients for the best Test R^2 in elastic net model are given below :

```
array([[0.00475841],
       [0.00322165],
       [0.00627088],
       [0.00322873],
       [0.00530439],
       [0.00336598],
       [0.00605122],
       [0.00309992],
       [0.00553556],
       [0.00271043],
       [0.00712199],
       [0.0012842 ],
       [0.00836365],
       [0.00290355],
       [0.00428036],
       [0.00326167],
       [0.00492464],
       [0.00270387],
       [0.0054718 ],
       [0.00113735],
       [0.1109303 ],
       [0.0091522 ],
       [0.00088932],
       [0.00540959],
       [0.00014211],
       [0.00476814],
       [0.07411704],
       [0.0504452 ]])
```

The coefficients of Ridge without SGD that performed well on validation data are :

```
array([ 14.36479005,  -5.22738175,  11.18651141,  -7.250327  ,
       -12.27338396,  25.36479653,  21.11861461,  -3.69370588,
        -7.13401571,  -3.33222394,   2.25977857,   5.16176822,
        -0.98252213,   1.23666284,  -5.35249052,   8.95058434,
        -5.89212568,  -9.82002434,  -1.00787877,  -4.96571115,
        -0.234283  ,   2.53411894,   1.72552405,   1.42319065,
        -1.41260902,  -2.76949532,  -0.29383492,  38.08538165])
```

The comparison between the coefficients can be plotted as :



From the above we can say the coefficients different in most of the time but they are equal some times.