Name - Ria Soam

Roll No. - 101803258

Batch - COE 12

Q1: (Based on Step-by-Step Implementation of Ridge Regression using Gradient Descent Optimization) Generate a dataset with atleast seven highly correlated columns and a target variable. Implement Ridge Regression using Gradient Descent Optimization. Take different values of learning rate (such as 0.0001,0.001,0.01,0.1,1,10) and regularization parameter (10-15,10-10,10-5,10- 3 ,0,1,10,20). Choose the best parameters for which ridge regression cost function is minimum and R2_score is maximum.

In [1]:

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

In [2]:

```
X=np.array([i*np.pi/180 for i in range (60,300,4)])
np.random.seed(10)
y=np.sin(X)+np.random.normal(0, 0.15, len(X))
df= pd.DataFrame(np.column_stack([X,y]),columns=['X','y'])
for i in range (2,16):
  colname='X_%d'%i
  df[colname]=df['X']**i
print(df)
```

```
           X         y        X_2        X_3         X_4         X_5  \
0   1.047198  1.065763   1.096623   1.148381    1.202581    1.259340
1   1.117011  1.006086   1.247713   1.393709    1.556788    1.738948
2   1.186824  0.695374   1.408551   1.671702    1.984016    2.354677
3   1.256637  0.949799   1.579137   1.984402    2.493673    3.133642
4   1.326450  1.063496   1.759470   2.333850    3.095735    4.106339
5   1.396263  0.876795   1.949551   2.722087    3.800751    5.306850
6   1.466077  1.034349   2.149381   3.151156    4.619837    6.773034
7   1.535890  1.015673   2.358957   3.623098    5.564680    8.546734
8   1.605703  1.000035   2.578282   4.139955    6.647537   10.673970
9   1.675516  0.968332   2.807354   4.703767    7.881237   13.205140
10  1.745329  1.049762   3.046174   5.316577    9.279177   16.195219
11  1.815142  1.150751   3.294742   5.980426   10.855325   19.703961
12  1.884956  0.806297   3.553058   6.697356   12.624218   23.796091
13  1.954769  1.081425   3.821121   7.469408   14.600965   28.541510
14  2.024582  0.933089   4.098932   8.298624   16.801244   34.015494
15  2.094395  0.932796   4.386491   9.187045   19.241302   40.298889
16  2.164208  0.658547   4.683797  10.136713   21.937959   47.478311
17  2.234021  0.808281   4.990852  11.149670   24.908602   55.646350
18  2.303835  0.965825   5.307654  12.227957   28.171190   64.901763
19  2.373648  0.532688   5.634204  13.373615   31.744252   75.349674
20  2.443461  0.346128   5.970501  14.588687   35.646887   87.101777
21  2.513274  0.326279   6.316547  15.875214   39.898764  100.276530
22  2.583087  0.569830   6.672340  17.235237   44.520121  114.999358
23  2.652900  0.827217   7.037881  18.670797   49.531767  131.402848
24  2.722714  0.575290   7.413170  20.183938   54.955082  149.626952
25  2.792527  0.592913   7.798206  21.776699   60.812016  169.819185
26  2.862340  0.290510   8.192990  23.451123   67.125087  192.134820
27  2.932153  0.417611   8.597522  25.209251   73.917386  216.737094
28  3.001966  0.098486   9.011802  27.053125   81.212571  243.797402
29  3.071779  0.161737   9.435829  28.984787   89.034873  273.495495
30  3.141593 -0.040098   9.869604  31.006277   97.409091  306.019685
31  3.211406 -0.152153  10.313127  33.119637  106.360596  341.567038
32  3.281219 -0.119267  10.766398  35.326910  115.915328  380.343575
33  3.351032 -0.279333  11.229417  37.630136  126.099796  422.564473
34  3.420845 -0.079366  11.702183  40.031357  136.941082  468.454262
35  3.490659 -0.312768  12.184697  42.532615  148.466836  518.247023
36  3.560472 -0.346705  12.676959  45.135952  160.705278  572.186590
37  3.630285 -0.520116  13.178968  47.843408  173.685199  630.526745
38  3.700098 -0.341448  13.690725  50.657026  187.435960  693.531422
39  3.769911 -0.697581  14.212230  53.578846  201.987491  761.474902
```

40	3.839724	-0.543753	14.743483	56.610911	217.370294	834.642013
41	3.909538	-0.747289	15.284484	59.755262	233.615441	913.328331
42	3.979351	-0.884060	15.835232	63.013941	250.754571	997.840376
43	4.049164	-0.861411	16.395728	66.388989	268.819897	1088.495812
44	4.118977	-0.949726	16.965972	69.882448	287.844200	1185.623647
45	4.188790	-0.897930	17.545963	73.496360	307.860831	1289.564433
46	4.258603	-0.949665	18.135703	77.232765	328.903713	1400.670460
47	4.328417	-0.880358	18.735190	81.093705	351.007336	1519.305962
48	4.398230	-0.866284	19.344425	85.081223	374.206764	1645.847309
49	4.468043	-0.992409	19.963407	89.197360	398.537628	1780.683213
50	4.537856	-0.988694	20.592138	93.444156	424.036130	1924.214920
51	4.607669	-0.951158	21.230616	97.823655	450.739043	2076.856415
52	4.677482	-1.080373	21.878842	102.337896	478.683708	2239.034617
53	4.747296	-0.893167	22.536815	106.988923	507.908039	2411.189580
54	4.817109	-0.868188	23.204537	111.778776	538.450517	2593.774691
55	4.886922	-0.954271	23.882006	116.709497	570.350197	2787.256871
56	4.956735	-0.611090	24.569223	121.783127	603.646700	2992.116770
57	5.026548	-0.813438	25.266187	127.001709	638.380219	3208.848970
58	5.096361	-0.944025	25.972900	132.367284	674.591518	3437.962183
59	5.166175	-0.953121	26.689360	137.881893	712.321929	3679.979448

	X_6	X_7	X_8	X_9	X_10 \
0	1.318778	1.381021	1.446202	1.514459e+00	1.585938e+00
1	1.942424	2.169709	2.423588	2.707173e+00	3.023942e+00
2	2.794587	3.316683	3.936319	4.671717e+00	5.544505e+00
3	3.937850	4.948448	6.218404	7.814277e+00	9.819710e+00
4	5.446854	7.224981	9.583578	1.271214e+01	1.686202e+01
5	7.409760	10.345976	14.445708	2.017001e+01	2.816265e+01
6	9.929787	14.557828	21.342890	3.129031e+01	4.587399e+01
7	13.126841	20.161381	30.965658	4.755984e+01	7.304667e+01
8	17.139225	27.520503	44.189752	7.095561e+01	1.139336e+02
9	22.125424	37.071504	62.113901	1.040728e+02	1.743757e+02
10	28.265990	49.333460	86.103130	1.502783e+02	2.622851e+02
11	35.765495	64.919467	117.838079	2.138929e+02	3.882461e+02
12	44.854574	84.548881	159.370885	3.004070e+02	5.662539e+02
13	55.792053	109.060562	213.188180	4.167336e+02	8.146178e+02
14	68.867155	139.427197	282.281785	5.715026e+02	1.157054e+03
15	84.401795	176.770706	370.227700	7.754031e+02	1.624000e+03
16	102.752954	222.378793	481.274024	1.041577e+03	2.254190e+03
17	124.315140	277.722688	620.438441	1.386073e+03	3.096516e+03
18	149.522928	344.476097	793.615955	1.828360e+03	4.212239e+03
19	178.853587	424.535421	1007.697562	2.391919e+03	5.677573e+03
20	212.829792	520.041285	1270.700574	3.104907e+03	7.586720e+03
21	252.022409	633.401398	1591.911344	4.000910e+03	1.005538e+04
22	297.053380	767.314810	1982.041137	5.119785e+03	1.322485e+04
23	348.598676	924.797589	2453.395952	6.508615e+03	1.726671e+04
24	407.391343	1109.209963	3020.061087	8.222761e+03	2.238822e+04
25	474.224625	1324.284975	3698.101288	1.032705e+04	2.883856e+04
26	549.955176	1574.158684	4505.777325	1.289707e+04	3.691579e+04
27	635.506352	1863.401948	5463.779880	1.602064e+04	4.697497e+04
28	731.871587	2197.053849	6595.481644	1.979941e+04	5.943717e+04
29	840.117850	2580.656776	7927.208539	2.435064e+04	7.479979e+04
30	961.389194	3020.293228	9488.531016	2.980910e+04	9.364805e+04
31	1096.910373	3522.624361	11312.576388	3.632927e+04	1.166680e+05
32	1247.990562	4094.930337	13436.363201	4.408765e+04	1.446612e+05
33	1416.027142	4745.152497	15901.158638	5.328529e+04	1.785607e+05
34	1602.509577	5481.937409	18752.860008	6.415063e+04	2.194494e+05
35	1809.023379	6314.682842	22042.401362	7.694250e+04	2.685800e+05
36	2037.254145	7253.585676	25826.186336	9.195340e+04	3.273975e+05
37	2288.991686	8309.691827	30166.548301	1.095132e+05	3.975640e+05
38	2566.134238	9494.948197	35132.238969	1.299927e+05	4.809858e+05
39	2870.692750	10822.256704	40798.946588	1.538084e+05	5.798440e+05
40	3204.795266	12305.530435	47249.844905	1.814264e+05	6.966273e+05
41	3570.691384	13959.751952	54576.174090	2.133676e+05	8.341686e+05
42	3970.756793	15801.033802	62877.854835	2.502130e+05	9.956854e+05
43	4407.497909	17846.681265	72264.136881	2.926093e+05	1.184823e+06
44	4883.556576	20115.257383	82854.283209	3.412749e+05	1.405703e+06
45	5401.714865	22626.650317	94778.291216	3.970064e+05	1.662976e+06
46	5964.899950	25402.143057	108177.652151	4.606857e+05	1.961878e+06
47	6576.189063	28464.485542	123206.150163	5.332875e+05	2.308291e+06
48	7238.814543	31837.969225	140030.702310	6.158872e+05	2.708813e+06
49	7956.168961	35548.504117	158832.240898	7.096693e+05	3.170833e+06
50	8731.810327	39623.698366	179806.639557	8.159366e+05	3.702603e+06
51	9569.467389	44092.940389	203165.684482	9.361203e+05	4.313333e+06
52	10473.045003	48987.483628	229138.092260	1.071789e+06	5.013276e+06
53	11446.629600	54340.533937	257970.575782	1.224663e+06	5.813835e+06
54	12494.494722	60187.339672	289928.959703	1.396619e+06	6.727667e+06

```
54  12494.494722  60187.559672  289928.959703  1.396619e+06  6.727667e+06
55  13621.106657  66565.284502  325299.346982  1.589713e+06  7.768801e+06
56  14831.130142  73513.982988  364389.338029  1.806181e+06  8.952763e+06
57  16129.434161  81075.378985  407529.304011  2.048466e+06  1.029671e+07
58  17521.097818  89293.846882  455073.715919  2.319220e+06  1.181958e+07
59  19011.416302  98216.295741  507402.530978  2.621330e+06  1.354225e+07

            X_11          X_12          X_13          X_14          X_15
0   1.660790e+00  1.739176e+00  1.821260e+00  1.907219e+00  1.997235e+00
1   3.377775e+00  3.773011e+00  4.214494e+00  4.707635e+00  5.258479e+00
2   6.580351e+00  7.809718e+00  9.268760e+00  1.100039e+01  1.305552e+01
3   1.233981e+01  1.550666e+01  1.948625e+01  2.448714e+01  3.077145e+01
4   2.236663e+01  2.966822e+01  3.935342e+01  5.220035e+01  6.924117e+01
5   3.932248e+01  5.490454e+01  7.666120e+01  1.070392e+02  1.494550e+02
6   6.725479e+01  9.860066e+01  1.445561e+02  2.119303e+02  3.107061e+02
7   1.121916e+02  1.723140e+02  2.646553e+02  4.064813e+02  6.243104e+02
8   1.829436e+02  2.937530e+02  4.716801e+02  7.573781e+02  1.216124e+03
9   2.921693e+02  4.895344e+02  8.202227e+02  1.374296e+03  2.302656e+03
10  4.577739e+02  7.989662e+02  1.394459e+03  2.433790e+03  4.247765e+03
11  7.047219e+02  1.279171e+03  2.321877e+03  4.214537e+03  7.649985e+03
12  1.067364e+03  2.011933e+03  3.792404e+03  7.148513e+03  1.347463e+04
13  1.592389e+03  3.112753e+03  6.084713e+03  1.189421e+04  2.325042e+04
14  2.342550e+03  4.742685e+03  9.601954e+03  1.943994e+04  3.935776e+04
15  3.401299e+03  7.123663e+03  1.491976e+04  3.124788e+04  6.544541e+04
16  4.878537e+03  1.055817e+04  2.285008e+04  4.945233e+04  1.070251e+05
17  6.917684e+03  1.545425e+04  3.452513e+04  7.712989e+04  1.723098e+05
18  9.704302e+03  2.235711e+04  5.150707e+04  1.186638e+05  2.733817e+05
19  1.347656e+04  3.198861e+04  7.592968e+04  1.802303e+05  4.278033e+05
20  1.853785e+04  4.529652e+04  1.106803e+05  2.704429e+05  6.608168e+05
21  2.527193e+04  6.351529e+04  1.596313e+05  4.011973e+05  1.008319e+06
22  3.416095e+04  8.824071e+04  2.279335e+05  5.887720e+05  1.520850e+06
23  4.580686e+04  1.215210e+05  3.223832e+05  8.552506e+05  2.268895e+06
24  6.095672e+04  1.659677e+05  4.518825e+05  1.230347e+06  3.349882e+06
25  8.053244e+04  2.248890e+05  6.280085e+05  1.753731e+06  4.897340e+06
26  1.056655e+05  3.024507e+05  8.657167e+05  2.477976e+06  7.092809e+06
27  1.377378e+05  4.038683e+05  1.184204e+06  3.472267e+06  1.018122e+07
28  1.784284e+05  5.356360e+05  1.607961e+06  4.827046e+06  1.449063e+07
29  2.297684e+05  7.057980e+05  2.168056e+06  6.659789e+06  2.045740e+07
30  2.942040e+05  9.242692e+05  2.903677e+06  9.122171e+06  2.865815e+07
31  3.746684e+05  1.203212e+06  3.864003e+06  1.240888e+07  3.984996e+07
32  4.746652e+05  1.557480e+06  5.110434e+06  1.676845e+07  5.502097e+07
33  5.983628e+05  2.005133e+06  6.719265e+06  2.251647e+07  7.545342e+07
34  7.507024e+05  2.568037e+06  8.784857e+06  3.005164e+07  1.028020e+08
35  9.375210e+05  3.272566e+06  1.142341e+07  3.987522e+07  1.391908e+08
36  1.165690e+06  4.150404e+06  1.477740e+07  5.261451e+07  1.873325e+08
37  1.443270e+06  5.239483e+06  1.902082e+07  6.905098e+07  2.506747e+08
38  1.779695e+06  6.585045e+06  2.436531e+07  9.015404e+07  3.335788e+08
39  2.185960e+06  8.240877e+06  3.106737e+07  1.171212e+08  4.415367e+08
40  2.674857e+06  1.027071e+07  3.943671e+07  1.514261e+08  5.814344e+08
41  3.261214e+06  1.274984e+07  4.984597e+07  1.948747e+08  7.618699e+08
42  3.962181e+06  1.576691e+07  6.274206e+07  2.496727e+08  9.935351e+08
43  4.797543e+06  1.942604e+07  7.865921e+07  3.185040e+08  1.289675e+09
44  5.790060e+06  2.384912e+07  9.823400e+07  4.046236e+08  1.666635e+09
45  6.965859e+06  2.917852e+07  1.222227e+08  5.119653e+08  2.144515e+09
46  8.354859e+06  3.558003e+07  1.515212e+08  6.452689e+08  2.747944e+09
47  9.991243e+06  4.324626e+07  1.871878e+08  8.102269e+08  3.507000e+09
48  1.191398e+07  5.240044e+07  2.304692e+08  1.013656e+09  4.458293e+09
49  1.416742e+07  6.330062e+07  2.828299e+08  1.263696e+09  5.646249e+09
50  1.680188e+07  7.624451e+07  3.459866e+08  1.570037e+09  7.124604e+09
51  1.987441e+07  9.157471e+07  4.219460e+08  1.944187e+09  8.958172e+09
52  2.344951e+07  1.096847e+08  5.130481e+08  2.399774e+09  1.122490e+10
53  2.759999e+07  1.310253e+08  6.220160e+08  2.952894e+09  1.401826e+10
54  3.240790e+07  1.561124e+08  7.520104e+08  3.622516e+09  1.745005e+10
55  3.796552e+07  1.855345e+08  9.066928e+08  4.430937e+09  2.165364e+10
56  4.437647e+07  2.199624e+08  1.090295e+09  5.404306e+09  2.678771e+10
57  5.175692e+07  2.601586e+08  1.307700e+09  6.573217e+09  3.304059e+10
58  6.023687e+07  3.069889e+08  1.564526e+09  7.973391e+09  4.063528e+10
59  6.996162e+07  3.614339e+08  1.867231e+09  9.646441e+09  4.983520e+10
```

In [3]:

```python
X=df.drop(['y'],axis=1).values
Y=df.iloc[:,1].values
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X)
X_scaled=np.insert(X_scaled,0,values=1,axis=1)
```

```
X_scaled=np.insert(X_scaled,0,values=1,axis=1)
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size=0.3, random_state=42)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.2, random_state=42)
betas=[]
l_rate = [0.0001, 0.001, 0.01, 0.1, 1, 10]
r_para = [pow(10,-15), pow(10,-10), pow(10,-5), pow(10,-3), 0, 1, 10, 20]
for learning_rate in l_rate:
  for r_p in r_para:
    beta = np.zeros(X_train.shape[1])
    for j in range(X_train.shape[1]):
      parsum=0
      for i in range(X_train.shape[0]):
        sum=0
        sum+=beta[0]
        for k in range(X_train.shape[1]):
          if k==0:
            continue
          sum+=beta[k]*X_train[i][k]
        sum-=Y_train[i]
        sum*=X_train[i][j]
      parsum=sum
      one=(parsum*learning_rate)/X_train.shape[0]
      two=1-((learning_rate*r_p)/X_train.shape[0])
      beta[j]=(beta[j]*two)-one
    betas.append(beta)
```

In [4]:

```
r2 = []
from sklearn.metrics import r2_score
for beta in betas:
  Y_pred_val = X_val.dot(beta)
  r2.append(r2_score(Y_val, Y_pred_val))
```

In [5]:

```
max_index = r2.index(max(r2))
optimal_beta = betas[max_index]
Y_pred_final = X_test.dot(optimal_beta)
r2_final = r2_score(Y_test, Y_pred_final)
```

In [6]:

```
r2_final
```

Out[6]:

```
0.7623215188142889
```

Q2: (Based on using Inbuilt function of Linear, Ridge, and Lasso Regression) Load the Hitters dataset from the following link https://drive.google.com/file/d/1qzCKF6JKKMB0p7ul_ILy8tdmRk3vE_bG/view?usp=sharing (a) Pre-process the data (null values, noise, categorical to numerical encoding) (b) Separate input and output features and perform scaling (c) Fit a Linear, Ridge (use regularization parameter as 0.5748), and LASSO (use regularization parameter as 0.5748) regression function on the dataset. (d) Evaluate the performance of each trained model on test set. Which model performs the best and Why?

In [7]:

```
df=pd.read_csv('Hitters.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 322 entries, 0 to 321
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   AtBat      322 non-null    int64
 1   Hits       322 non-null    int64
 2   HmRun      322 non-null    int64
 3   Runs       322 non-null    int64
 4   RBI        322 non-null    int64
```

```
 5   Walks      322 non-null    int64
 6   Years      322 non-null    int64
 7   CAtBat     322 non-null    int64
 8   CHits      322 non-null    int64
 9   CHmRun     322 non-null    int64
 10  CRuns      322 non-null    int64
 11  CRBI       322 non-null    int64
 12  CWalks     322 non-null    int64
 13  League     322 non-null    object
 14  Division   322 non-null    object
 15  PutOuts    322 non-null    int64
 16  Assists    322 non-null    int64
 17  Errors     322 non-null    int64
 18  Salary     263 non-null    float64
 19  NewLeague  322 non-null    object
dtypes: float64(1), int64(16), object(3)
memory usage: 50.4+ KB
```

In [8]:

```
df.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75%	max
AtBat	322.0	380.928571	153.404981	16.0	255.25	379.5	512.00	687.0
Hits	322.0	101.024845	46.454741	1.0	64.00	96.0	137.00	238.0
HmRun	322.0	10.770186	8.709037	0.0	4.00	8.0	16.00	40.0
Runs	322.0	50.909938	26.024095	0.0	30.25	48.0	69.00	130.0
RBI	322.0	48.027950	26.166895	0.0	28.00	44.0	64.75	121.0
Walks	322.0	38.742236	21.639327	0.0	22.00	35.0	53.00	105.0
Years	322.0	7.444099	4.926087	1.0	4.00	6.0	11.00	24.0
CAtBat	322.0	2648.683230	2324.205870	19.0	816.75	1928.0	3924.25	14053.0
CHits	322.0	717.571429	654.472627	4.0	209.00	508.0	1059.25	4256.0
CHmRun	322.0	69.490683	86.266061	0.0	14.00	37.5	90.00	548.0
CRuns	322.0	358.795031	334.105886	1.0	100.25	247.0	526.25	2165.0
CRBI	322.0	330.118012	333.219617	0.0	88.75	220.5	426.25	1659.0
CWalks	322.0	260.239130	267.058085	0.0	67.25	170.5	339.25	1566.0
PutOuts	322.0	288.937888	280.704614	0.0	109.25	212.0	325.00	1378.0
Assists	322.0	106.913043	136.854876	0.0	7.00	39.5	166.00	492.0
Errors	322.0	8.040373	6.368359	0.0	3.00	6.0	11.00	32.0
Salary	263.0	535.925882	451.118681	67.5	190.00	425.0	750.00	2460.0

In [9]:

```
df[df.isnull().any(axis=1)].head(3)
```

Out[9]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks	League	Division	PutOuts	Assi
0	293	66	1	30	29	14	1	293	66	1	30	29	14	A	E	446	
15	183	39	3	20	15	11	3	201	42	3	20	16	11	A	W	118	
18	407	104	6	57	43	65	12	5233	1478	100	643	658	653	A	W	912	

In [10]:

```
df.isnull().sum().sum()
```

Out[10]:

59

In [11]:

```
#(a) Data Preprocessing
df=df.copy()
df.corr()
```

Out[11]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CW
AtBat	1.000000	0.967939	0.592198	0.913060	0.820539	0.669845	0.047372	0.235526	0.252717	0.236659	0.266534	0.244053	0.16
Hits	0.967939	1.000000	0.562158	0.922187	0.811073	0.641211	0.044767	0.227565	0.255815	0.202712	0.261787	0.232005	0.15
HmRun	0.592198	0.562158	1.000000	0.650988	0.855122	0.481014	0.116318	0.221882	0.220627	0.493227	0.262361	0.351979	0.23
Runs	0.913060	0.922187	0.650988	1.000000	0.798206	0.732213	0.004541	0.186497	0.204830	0.227913	0.250556	0.205976	0.18
RBI	0.820539	0.811073	0.855122	0.798206	1.000000	0.615997	0.146168	0.294688	0.308201	0.441771	0.323285	0.393184	0.25
Walks	0.669845	0.641211	0.481014	0.732213	0.615997	1.000000	0.136475	0.277175	0.280671	0.332473	0.338478	0.308631	0.42
Years	0.047372	0.044767	0.116318	0.004541	0.146168	0.136475	1.000000	0.920289	0.903631	0.726872	0.882877	0.868812	0.83
CAtBat	0.235526	0.227565	0.221882	0.186497	0.294688	0.277175	0.920289	1.000000	0.995063	0.798836	0.983345	0.949219	0.90
CHits	0.252717	0.255815	0.220627	0.204830	0.308201	0.280671	0.903631	0.995063	1.000000	0.783306	0.984609	0.945141	0.89
CHmRun	0.236659	0.202712	0.493227	0.227913	0.441771	0.332473	0.726872	0.798836	0.783306	1.000000	0.820243	0.929484	0.79
CRuns	0.266534	0.261787	0.262361	0.250556	0.323285	0.338478	0.882877	0.983345	0.984609	0.820243	1.000000	0.943769	0.92
CRBI	0.244053	0.232005	0.351979	0.205976	0.393184	0.308631	0.868812	0.949219	0.945141	0.929484	0.943769	1.000000	0.88
CWalks	0.166123	0.151818	0.233154	0.182168	0.250914	0.424507	0.838533	0.906501	0.890954	0.799983	0.927807	0.884726	1.00
PutOuts	0.317550	0.310673	0.282923	0.279347	0.343186	0.299515	-0.004684	0.062283	0.076547	0.112724	0.064180	0.110098	0.05
Assists	0.353824	0.320455	-0.106329	0.220567	0.106591	0.149656	-0.080638	0.002038	-0.002523	-0.158511	-0.022978	-0.079387	0.03
Errors	0.352117	0.310038	0.039318	0.240475	0.193370	0.129382	-0.162140	-0.066922	-0.062756	-0.138115	-0.084395	-0.100990	0.11
Salary	0.394771	0.438675	0.343028	0.419859	0.449457	0.443867	0.400657	0.526135	0.548910	0.524931	0.562678	0.566966	0.48

In [12]:

```
df['Year_lab'] = pd.cut(x=df['Years'], bins=[0, 3, 6, 10, 15, 19, 24])
df.groupby(['League','Division', 'Year_lab']).agg({'Salary':'mean'})
```

Out[12]:

League	Division	Year_lab	Salary
A	**E**	**(0, 3]**	112.500000
		(3, 6]	655.568182
		(6, 10]	852.738125
		(10, 15]	816.311353
		(15, 19]	665.416750
		(19, 24]	NaN
	W	**(0, 3]**	153.613636
		(3, 6]	401.360000
		(6, 10]	633.958375
		(10, 15]	835.250000

League	Division	Year_lab	Salary
		(15, 19]	479.000000
		(19, 24]	487.500000
N	E	(0, 3]	248.520813
		(3, 6]	501.191650
		(6, 10]	824.226143
		(10, 15]	894.322667
		(15, 19]	662.500000
		(19, 24]	NaN
	W	(0, 3]	191.766667
		(3, 6]	458.333333
		(6, 10]	563.229188
		(10, 15]	721.894000
		(15, 19]	760.833250
		(19, 24]	475.000000

In [13]:

```
df['Salary'] = df.groupby(['League', 'Division', 'Year_lab'])['Salary'].transform(lambda x: x.filln
a(x.mean()))
df.head()
```

Out[13]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	...	CRBI	CWalks	League	Division	PutOuts	Assists
0	293	66	1	30	29	14	1	293	66	1	...	29	14	A	E	446	33
1	315	81	7	24	38	39	14	3449	835	69	...	414	375	N	W	632	43
2	479	130	18	66	72	76	3	1624	457	63	...	266	263	A	W	880	82
3	496	141	20	65	78	37	11	5628	1575	225	...	838	354	N	E	200	11
4	321	87	10	39	42	30	2	396	101	12	...	46	33	N	E	805	40

5 rows × 21 columns

In [14]:

```
df.isnull().sum()
```

Out[14]:

```
AtBat        0
Hits         0
HmRun        0
Runs         0
RBI          0
Walks        0
Years        0
CAtBat       0
CHits        0
CHmRun       0
CRuns        0
CRBI         0
CWalks       0
League       0
Division     0
PutOuts      0
Assists      0
Errors       0
Salary       0
NewLeague    0
Year_lab     0
dtype: int64
```

In [15]:

In [15]:

```
df.shape
```

Out[15]:

```
(322, 21)
```

In [16]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['League'] = le.fit_transform(df['League'])
df['Division'] = le.fit_transform(df['Division'])
df['NewLeague'] = le.fit_transform(df['NewLeague'])
df['Year_lab'] = le.fit_transform(df['Year_lab'])
```

In [17]:

```
df.head()
```

Out[17]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	...	CRBI	CWalks	League	Division	PutOuts	Assists
0	293	66	1	30	29	14	1	293	66	1	...	29	14	0	0	446	33
1	315	81	7	24	38	39	14	3449	835	69	...	414	375	1	1	632	43
2	479	130	18	66	72	76	3	1624	457	63	...	266	263	0	1	880	82
3	496	141	20	65	78	37	11	5628	1575	225	...	838	354	1	0	200	11
4	321	87	10	39	42	30	2	396	101	12	...	46	33	1	0	805	40

5 rows × 21 columns

In [18]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 322 entries, 0 to 321
Data columns (total 21 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   AtBat      322 non-null    int64
 1   Hits       322 non-null    int64
 2   HmRun      322 non-null    int64
 3   Runs       322 non-null    int64
 4   RBI        322 non-null    int64
 5   Walks      322 non-null    int64
 6   Years      322 non-null    int64
 7   CAtBat     322 non-null    int64
 8   CHits      322 non-null    int64
 9   CHmRun     322 non-null    int64
 10  CRuns      322 non-null    int64
 11  CRBI       322 non-null    int64
 12  CWalks     322 non-null    int64
 13  League     322 non-null    int32
 14  Division   322 non-null    int32
 15  PutOuts    322 non-null    int64
 16  Assists    322 non-null    int64
 17  Errors     322 non-null    int64
 18  Salary     322 non-null    float64
 19  NewLeague  322 non-null    int32
 20  Year_lab   322 non-null    int32
dtypes: float64(1), int32(4), int64(16)
memory usage: 47.9 KB
```

In [19]:

```
from sklearn import preprocessing
df_X= df.drop(["Salary","League","Division","NewLeague"], axis=1)
```

```
scaled_cols5=preprocessing.normalize(df_X)
scaled_cols=pd.DataFrame(scaled_cols5, columns=df_X.columns)
scaled_cols.head()
```

Out[19]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWalks
0	0.472401	0.106411	0.001612	0.048369	0.046756	0.022572	0.001612	0.472401	0.106411	0.001612	0.048369	0.046756	0.022572
1	0.085657	0.022026	0.001903	0.006526	0.010333	0.010605	0.003807	0.937879	0.227060	0.018763	0.087289	0.112578	0.101973
2	0.237036	0.064331	0.008907	0.032660	0.035630	0.037609	0.001485	0.803645	0.226149	0.031176	0.110848	0.131631	0.130147
3	0.082624	0.023488	0.003332	0.010828	0.012993	0.006163	0.001832	0.937518	0.262365	0.037481	0.137929	0.139595	0.058970
4	0.331579	0.089867	0.010330	0.040285	0.043384	0.030989	0.002066	0.409050	0.104328	0.012395	0.049582	0.047516	0.034088

In [20]:

```
cat_df=pd.concat([df.loc[:,"League":"Division"],df.loc[:,"NewLeague":"Year_lab"]], axis=1)
cat_df.head()
```

Out[20]:

	League	Division	NewLeague	Year_lab
0	0	0	0	0
1	1	1	1	3
2	0	1	0	0
3	1	0	1	3
4	1	0	1	0

In [21]:

```
df= pd.concat([scaled_cols,cat_df,df["Salary"]], axis=1)
print(df)
print(df.shape)
```

```
        AtBat      Hits     HmRun      Runs       RBI     Walks     Years  \
0    0.472401  0.106411  0.001612  0.048369  0.046756  0.022572  0.001612
1    0.085657  0.022026  0.001903  0.006526  0.010333  0.010605  0.003807
2    0.237036  0.064331  0.008907  0.032660  0.035630  0.037609  0.001485
3    0.082624  0.023488  0.003332  0.010828  0.012993  0.006163  0.001832
4    0.331579  0.089867  0.010330  0.040285  0.043384  0.030989  0.002066
..        ...       ...       ...       ...       ...       ...       ...
317  0.169544  0.043324  0.002388  0.022174  0.016374  0.012622  0.001706
318  0.083222  0.023004  0.000846  0.012855  0.008457  0.015900  0.002030
319  0.256903  0.068147  0.001623  0.032992  0.023256  0.028124  0.003245
320  0.155442  0.039064  0.002441  0.023059  0.016277  0.021160  0.002170
321  0.120098  0.032356  0.001713  0.014655  0.008375  0.005900  0.002094

       CAtBat     CHits    CHmRun  ...    CWalks   PutOuts   Assists  \
0    0.472401  0.106411  0.001612  ...  0.022572  0.719082  0.053206
1    0.937879  0.227060  0.018763  ...  0.101973  0.171858  0.011693
2    0.803645  0.226149  0.031176  ...  0.130147  0.435473  0.040578
3    0.937518  0.262365  0.037481  ...  0.058970  0.033316  0.001832
4    0.409050  0.104328  0.012395  ...  0.034088  0.831529  0.041318
..        ...       ...       ...  ...       ...       ...       ...
317  0.922085  0.274954  0.010916  ...  0.047076  0.110869  0.003070
318  0.932185  0.255585  0.006597  ...  0.148006  0.052944  0.064446
319  0.919443  0.234188  0.003786  ...  0.078964  0.020011  0.061116
320  0.867543  0.232484  0.026314  ...  0.090064  0.356458  0.035537
321  0.934139  0.277311  0.005710  ...  0.047392  0.077655  0.000761

       Errors  Year_lab  League  Division  NewLeague  Year_lab  Salary
0    0.032246  0.000000       0         0          0         0   112.5
1    0.002719  0.000816       1         1          1         3   475.0
2    0.006928  0.000000       0         1          0         0   480.0
3    0.000500  0.000500       1         0          1         3   500.0
4    0.004132  0.000000       1         0          1         0    91.5
..        ...       ...     ...       ...        ...       ...     ...
```

```
317  0.001023  0.000341       1          0          1          1   700.0
318  0.003383  0.000507       0          0          0          3   875.0
319  0.003786  0.000541       0          1          0          1   385.0
320  0.003255  0.000543       0          0          0          2   960.0
321  0.000571  0.000571       0          1          0          3  1000.0

[322 rows x 22 columns]
(322, 22)
```

In [22]:

```
#(b) Seperating Input and Output Features and Performing Scaling
from sklearn.model_selection import train_test_split
X=df.drop("Salary", axis=1)
y=df["Salary"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=46)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

In [23]:

```
#(c) Linear, Ridge and Lasso Regression
from sklearn import metrics
from sklearn.metrics import mean_squared_error
#Linear Regression
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
model = linreg.fit(X_train,y_train)
y_pred = model.predict(X_test)
print(metrics.r2_score(y_test, y_pred))
```

```
0.37608958263225
```

In [24]:

```
#Ridge Regression
from sklearn.linear_model import Ridge
ridreg = Ridge(alpha=0.5748, normalize=True)
model = ridreg.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(metrics.r2_score(y_test, y_pred))
```

```
-0.23721799488853867
```

In [25]:

```
#Lasso Regression
from sklearn.linear_model import Lasso
lasreg = Lasso(alpha=0.5748, normalize=True)
model = lasreg.fit(X_train,y_train)
y_pred = model.predict(X_test)
print(metrics.r2_score(y_test, y_pred))
```

```
0.2634733775931074
```

Q3: Cross Validation for Ridge and Lasso Regression Explore Ridge Cross Validation (RidgeCV) and Lasso Cross Validation (LassoCV) function of Python. Implement both on Boston House Prediction Dataset (load_boston dataset from sklearn.datasets).

In [26]:

```
from sklearn.datasets import load_boston
boston_dataset = load_boston()
dataset = pd.DataFrame(boston_dataset.data, columns = boston_dataset.feature_names)
```

In [27]:

```
dataset.head()
```

Out[27]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [28]:

```
dataset['MEDV'] = boston_dataset.target
dataset.head()
```

Out[28]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

In [29]:

```
X = dataset.iloc[:, 0:13].values
y = dataset.iloc[:, 13].values.reshape(-1,1)
```

In [30]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 25)
from sklearn.model_selection import cross_val_score
```

In [31]:

```
#Ridge Regression Model
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

steps = [
    ('scalar', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Ridge(alpha=3.8, fit_intercept=True))
]

ridge_pipe = Pipeline(steps)
ridge_pipe.fit(X_train, y_train)
```

Out[31]:

```
Pipeline(steps=[('scalar', StandardScaler()), ('poly', PolynomialFeatures()),
                ('model', Ridge(alpha=3.8))])
```

In [32]:

```
cv_ridge = cross_val_score(estimator = ridge_pipe, X = X_train, y = y_train.ravel(), cv = 10)
print('CV: ', cv_ridge.mean())
```

CV: 0.7635629962138676

In [33]:

```
#Lasso Regression Model
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

steps = [
    ('scalar', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Lasso(alpha=0.012, fit_intercept=True, max_iter=3000))
]

lasso_pipe = Pipeline(steps)
lasso_pipe.fit(X_train, y_train)
```

Out[33]:

```
Pipeline(steps=[('scalar', StandardScaler()), ('poly', PolynomialFeatures()),
                ('model', Lasso(alpha=0.012, max_iter=3000))])
```

In [34]:

```
cv_lasso = cross_val_score(estimator = lasso_pipe, X = X_train, y = y_train, cv = 10)
print('CV: ', cv_lasso.mean())
```

CV: 0.7505443182491572

In [ ]:

```

```