# Loading the dataset  
# Importing pandas   
import pandas as pd  
# Reading the dataset  
lake = pd.read\_csv('E:\\Lake Level\\Lake.csv')

# Seeting the seed to make the codes reproducible  
seed = 7

lake.describe()

ET SH AT ST P SM \  
count 240.000000 240.000000 240.000000 240.000000 240.000000 240.000000   
mean 0.000023 0.011697 26.616167 25.978083 26.410833 190.322167   
std 0.000012 0.004884 2.038722 2.471990 44.105702 43.431268   
min 0.000001 0.003926 21.370000 19.630000 0.000000 98.570000   
25% 0.000013 0.007485 25.197500 24.440000 0.040000 158.040000   
50% 0.000023 0.012002 26.470000 25.960000 1.250000 182.270000   
75% 0.000033 0.015124 28.185000 28.070000 39.112501 222.010000   
max 0.000046 0.030330 31.080000 30.990000 194.190002 292.340000   
  
 LL\_R LL\_G   
count 240.000000 240.000000   
mean 281.188333 279.216667   
std 0.455144 0.691097   
min 280.200000 277.800000   
25% 280.830000 278.720000   
50% 281.120000 279.215000   
75% 281.470000 279.710000   
max 282.280000 280.780000

# Correlation coefficients  
lake.corr()

ET SH AT ST P SM LL\_R \  
ET 1.000000 0.480772 -0.027288 0.176526 0.564678 0.891857 -0.087631   
SH 0.480772 1.000000 0.188390 0.511782 0.493766 0.420082 -0.168198   
AT -0.027288 0.188390 1.000000 0.849136 -0.165267 -0.276043 -0.458832   
ST 0.176526 0.511782 0.849136 1.000000 0.116543 -0.093402 -0.503949   
P 0.564678 0.493766 -0.165267 0.116543 1.000000 0.535514 -0.285750   
SM 0.891857 0.420082 -0.276043 -0.093402 0.535514 1.000000 0.133670   
LL\_R -0.087631 -0.168198 -0.458832 -0.503949 -0.285750 0.133670 1.000000   
LL\_G -0.485342 -0.434356 -0.269355 -0.433467 -0.611487 -0.323463 0.664079   
  
 LL\_G   
ET -0.485342   
SH -0.434356   
AT -0.269355   
ST -0.433467   
P -0.611487   
SM -0.323463   
LL\_R 0.664079   
LL\_G 1.000000

# Finding the maxima and minima

# Reading the dataset  
#df = pd.read\_csv('E:\\Lake Level\\Lake.csv',sep =',', header = 0,index\_col = 0,parse\_dates=True)  
df = pd.read\_csv('E:\\Lake Level\\Lake.csv')

# Getting the datetime format  
from datetime import datetime  
df['Date'] = df['Date'].apply(lambda x: datetime.strptime(x, '%d/%m/%Y'))

# # LL\_R

# The maxima LL\_R  
df.loc[df.groupby(df['Date'].dt.year)['LL\_R'].idxmax().values]

Date ET SH AT ST P SM LL\_R LL\_G  
9 1993-01-10 0.000037 0.015654 27.00 27.04 0.29 248.63 281.64 278.63  
22 1994-01-11 0.000035 0.009943 24.78 24.51 0.06 241.21 282.00 279.95  
34 1995-01-11 0.000031 0.010449 24.83 25.06 0.06 237.58 282.04 280.06  
46 1996-01-11 0.000003 0.010118 24.56 25.54 0.06 100.11 282.12 280.33  
58 1997-01-11 0.000012 0.024205 26.33 27.30 0.06 191.71 281.77 279.92  
70 1998-01-11 0.000032 0.010698 25.83 25.06 0.06 227.42 282.28 280.55  
83 1999-01-12 0.000023 0.008354 24.36 22.92 0.00 209.66 282.20 280.77  
84 2000-01-01 0.000020 0.005909 24.25 22.69 0.04 191.24 282.07 280.67  
105 2001-01-10 0.000031 0.011635 26.17 25.58 0.30 211.50 282.00 280.07  
108 2002-01-01 0.000005 0.005055 21.37 21.08 0.04 143.40 281.63 280.17  
130 2003-01-11 0.000026 0.010396 25.42 24.57 0.06 209.32 281.50 280.30  
132 2004-01-01 0.000009 0.006893 24.66 22.63 0.04 165.83 281.52 280.09  
144 2005-01-01 0.000010 0.005843 22.96 20.95 0.04 158.17 281.25 279.25  
165 2006-01-10 0.000033 0.013189 26.39 25.64 4.73 232.42 281.37 279.84  
168 2007-01-01 0.000005 0.004010 22.20 19.63 0.04 160.09 281.45 279.50  
190 2008-01-11 0.000031 0.007294 25.49 22.93 0.06 206.16 281.98 279.50  
202 2009-01-11 0.000027 0.009111 25.81 24.92 0.06 215.08 281.57 279.00  
214 2010-01-11 0.000033 0.009001 26.47 25.13 0.03 223.99 282.15 280.10  
226 2011-01-11 0.000029 0.006705 25.13 23.64 0.00 209.92 281.75 279.35  
238 2012-01-11 0.000031 0.008772 27.32 24.79 0.03 222.22 282.26 280.50

# The minima LL\_R  
df.loc[df.groupby(df['Date'].dt.year)['LL\_R'].idxmin().values]

Date ET SH AT ST P SM LL\_R \  
5 1993-01-06 0.000036 0.016907 27.44 28.86 20.280001 221.94 280.43   
17 1994-01-06 0.000034 0.013636 27.49 28.57 14.410000 209.00 280.45   
29 1995-01-06 0.000036 0.014899 28.39 29.27 10.780000 222.82 280.74   
40 1996-01-05 0.000007 0.013905 28.99 30.99 2.710000 145.97 280.85   
53 1997-01-06 0.000026 0.013842 28.02 29.63 27.010000 189.88 280.83   
65 1998-01-06 0.000029 0.013156 28.83 29.69 29.280001 171.57 280.81   
77 1999-01-06 0.000028 0.014874 27.44 28.17 8.479999 194.76 280.94   
89 2000-01-06 0.000018 0.011911 28.84 30.05 20.680000 184.42 280.91   
101 2001-01-06 0.000026 0.012145 28.16 28.38 39.209999 156.23 280.62   
113 2002-01-06 0.000021 0.010931 29.47 29.49 14.260000 142.37 280.67   
125 2003-01-06 0.000025 0.013258 27.88 28.66 81.410004 167.74 280.87   
137 2004-01-06 0.000025 0.013391 27.35 27.78 23.920000 181.46 280.51   
149 2005-01-06 0.000030 0.014638 27.21 27.28 72.519997 192.69 280.39   
161 2006-01-06 0.000025 0.013106 27.95 27.85 37.919998 172.73 280.46   
173 2007-01-06 0.000027 0.013615 27.80 28.55 22.290001 180.91 280.70   
180 2008-01-01 0.000009 0.004930 24.18 22.40 0.040000 168.98 280.20   
198 2009-01-07 0.000035 0.014839 26.22 26.68 88.690002 215.81 280.58   
209 2010-01-06 0.000032 0.013962 28.20 28.45 51.410000 196.96 280.46   
222 2011-01-07 0.000037 0.014306 27.49 27.44 70.370003 206.04 280.44   
233 2012-01-06 0.000035 0.013685 28.83 29.12 38.740002 192.13 280.46   
  
 LL\_G   
5 278.39   
17 278.16   
29 278.74   
40 279.08   
53 279.21   
65 278.79   
77 279.30   
89 279.41   
101 278.82   
113 279.06   
125 278.25   
137 278.96   
149 278.95   
161 278.58   
173 278.29   
180 279.76   
198 278.35   
209 278.06   
222 278.64   
233 278.47

### The LL\_G

# The maxima LL\_G  
df.loc[df.groupby(df['Date'].dt.year)['LL\_G'].idxmax().values]

Date ET SH AT ST P SM LL\_R LL\_G  
0 1993-01-01 0.000015 0.006582 21.98 23.02 0.04 184.21 281.54 279.71  
23 1994-01-12 0.000025 0.006703 22.05 21.34 0.00 210.80 281.87 280.24  
35 1995-01-12 0.000023 0.008384 24.01 23.19 0.00 210.80 281.99 280.36  
47 1996-01-12 0.000003 0.008101 24.18 23.71 0.00 98.57 282.00 280.49  
48 1997-01-01 0.000001 0.022651 24.40 24.30 0.04 104.01 281.66 280.34  
71 1998-01-12 0.000023 0.007412 24.20 23.29 0.00 201.03 282.06 280.78  
83 1999-01-12 0.000023 0.008354 24.36 22.92 0.00 209.66 282.20 280.77  
84 2000-01-01 0.000020 0.005909 24.25 22.69 0.04 191.24 282.07 280.67  
107 2001-01-12 0.000008 0.007490 24.09 22.63 0.00 157.65 281.86 280.42  
108 2002-01-01 0.000005 0.005055 21.37 21.08 0.04 143.40 281.63 280.17  
130 2003-01-11 0.000026 0.010396 25.42 24.57 0.06 209.32 281.50 280.30  
132 2004-01-01 0.000009 0.006893 24.66 22.63 0.04 165.83 281.52 280.09  
155 2005-01-12 0.000009 0.005897 25.97 23.59 0.00 173.70 280.78 280.00  
165 2006-01-10 0.000033 0.013189 26.39 25.64 4.73 232.42 281.37 279.84  
179 2007-01-12 0.000019 0.005419 24.31 22.63 0.00 187.09 281.07 279.82  
180 2008-01-01 0.000009 0.004930 24.18 22.40 0.04 168.98 280.20 279.76  
192 2009-01-01 0.000011 0.005147 25.72 22.61 0.04 166.24 281.42 279.68  
215 2010-01-12 0.000020 0.005071 24.04 22.33 0.00 194.80 281.99 280.46  
216 2011-01-01 0.000010 0.004176 23.26 20.92 0.04 175.56 281.61 280.27  
239 2012-01-12 0.000023 0.005589 25.50 23.06 0.00 193.34 282.01 280.61

# The minima LL\_G  
df.loc[df.groupby(df['Date'].dt.year)['LL\_G'].idxmin().values]

Date ET SH AT ST P SM LL\_R \  
8 1993-01-09 0.000041 0.018766 26.70 26.48 34.259998 269.18 281.36   
18 1994-01-07 0.000037 0.015331 25.57 26.67 194.190002 239.22 280.53   
32 1995-01-09 0.000040 0.018415 26.39 26.36 44.910000 280.21 281.58   
43 1996-01-08 0.000004 0.017514 25.47 27.58 87.190002 117.41 281.37   
54 1997-01-07 0.000022 0.016092 27.14 28.42 73.489998 201.22 281.11   
67 1998-01-08 0.000043 0.018123 25.96 26.44 125.750000 266.64 281.45   
78 1999-01-07 0.000036 0.016365 25.93 26.68 182.419998 240.91 280.99   
90 2000-01-07 0.000019 0.012411 28.08 29.39 109.610001 186.31 281.07   
103 2001-01-08 0.000035 0.015905 25.48 25.13 59.799999 219.74 281.31   
115 2002-01-08 0.000032 0.015594 26.18 26.36 81.809998 214.22 280.83   
127 2003-01-08 0.000033 0.016528 24.41 25.03 121.230003 242.57 280.95   
139 2004-01-08 0.000029 0.016105 24.74 24.90 145.080002 232.31 280.82   
150 2005-01-07 0.000035 0.015307 25.39 25.86 63.320000 215.51 280.71   
163 2006-01-08 0.000038 0.015893 24.65 25.10 112.360001 247.41 280.96   
174 2007-01-07 0.000029 0.015233 25.40 26.71 147.779999 199.95 280.83   
187 2008-01-08 0.000035 0.016457 24.27 24.56 76.459999 261.31 281.02   
199 2009-01-08 0.000039 0.016034 25.79 25.99 107.809998 242.27 280.66   
211 2010-01-08 0.000043 0.016230 25.39 25.07 123.250000 269.44 280.85   
224 2011-01-09 0.000041 0.014614 26.46 25.86 53.639999 253.40 281.26   
234 2012-01-07 0.000039 0.015486 26.16 26.75 118.800003 230.38 280.62   
  
 LL\_G   
8 277.87   
18 277.87   
32 278.31   
43 278.58   
54 279.05   
67 278.53   
78 279.13   
90 279.25   
103 278.67   
115 278.78   
127 278.10   
139 278.73   
150 277.80   
163 278.33   
174 278.11   
187 278.20   
199 278.22   
211 277.87   
224 278.51   
234 278.35

################# P

# The maxima P  
df.loc[df.groupby(df['Date'].dt.year)['P'].idxmax().values]

Date ET SH AT ST P SM LL\_R \  
7 1993-01-08 0.000044 0.018319 25.80 26.56 122.290001 266.61 280.86   
18 1994-01-07 0.000037 0.015331 25.57 26.67 194.190002 239.22 280.53   
30 1995-01-07 0.000039 0.017276 26.73 27.58 116.320000 252.64 280.74   
43 1996-01-08 0.000004 0.017514 25.47 27.58 87.190002 117.41 281.37   
54 1997-01-07 0.000022 0.016092 27.14 28.42 73.489998 201.22 281.11   
67 1998-01-08 0.000043 0.018123 25.96 26.44 125.750000 266.64 281.45   
78 1999-01-07 0.000036 0.016365 25.93 26.68 182.419998 240.91 280.99   
90 2000-01-07 0.000019 0.012411 28.08 29.39 109.610001 186.31 281.07   
102 2001-01-07 0.000031 0.014932 25.78 26.91 62.869999 178.55 280.83   
115 2002-01-08 0.000032 0.015594 26.18 26.36 81.809998 214.22 280.83   
127 2003-01-08 0.000033 0.016528 24.41 25.03 121.230003 242.57 280.95   
139 2004-01-08 0.000029 0.016105 24.74 24.90 145.080002 232.31 280.82   
151 2005-01-08 0.000037 0.016146 24.46 24.90 140.750000 242.57 280.55   
163 2006-01-08 0.000038 0.015893 24.65 25.10 112.360001 247.41 280.96   
175 2007-01-08 0.000034 0.016181 24.07 24.66 189.419998 248.29 281.01   
186 2008-01-07 0.000033 0.015423 25.31 26.24 151.389999 220.35 280.68   
199 2009-01-08 0.000039 0.016034 25.79 25.99 107.809998 242.27 280.66   
211 2010-01-08 0.000043 0.016230 25.39 25.07 123.250000 269.44 280.85   
223 2011-01-08 0.000043 0.015482 25.73 26.18 114.279999 239.49 280.70   
235 2012-01-08 0.000041 0.016032 25.16 25.28 172.220001 263.16 281.12   
  
 LL\_G   
7 278.01   
18 277.87   
30 278.53   
43 278.58   
54 279.05   
67 278.53   
78 279.13   
90 279.25   
102 278.74   
115 278.78   
127 278.10   
139 278.73   
151 277.80   
163 278.33   
175 278.14   
186 278.25   
199 278.22   
211 277.87   
223 278.52   
235 278.37

# The minima P  
df.loc[df.groupby(df['Date'].dt.year)['P'].idxmin().values]

Date ET SH AT ST P SM LL\_R LL\_G  
1 1993-01-02 0.000011 0.006498 24.86 24.25 0.0 170.82 281.16 279.51  
13 1994-01-02 0.000009 0.005117 25.03 24.13 0.0 168.61 281.11 279.18  
25 1995-01-02 0.000014 0.006186 24.24 23.06 0.0 178.62 281.38 280.00  
37 1996-01-02 0.000014 0.007709 26.25 24.80 0.0 180.43 281.42 279.93  
49 1997-01-02 0.000002 0.022470 24.18 23.16 0.0 112.03 281.40 280.11  
61 1998-01-02 0.000018 0.006483 26.66 24.69 0.0 151.48 281.32 279.70  
73 1999-01-02 0.000018 0.008510 27.58 25.97 0.0 173.69 281.54 280.32  
85 2000-01-02 0.000015 0.005507 24.90 23.27 0.0 175.95 281.70 280.41  
97 2001-01-02 0.000009 0.005111 25.01 22.69 0.0 122.10 281.37 279.67  
109 2002-01-02 0.000005 0.005935 25.36 22.10 0.0 132.41 281.37 279.75  
121 2003-01-02 0.000010 0.006281 26.29 22.66 0.0 147.77 281.12 279.20  
133 2004-01-02 0.000006 0.005574 25.04 23.04 0.0 151.78 281.42 278.88  
145 2005-01-02 0.000006 0.007716 28.58 24.58 0.0 145.57 281.09 279.09  
157 2006-01-02 0.000007 0.005947 28.58 25.46 0.0 148.64 281.11 279.56  
169 2007-01-02 0.000004 0.004967 25.90 23.02 0.0 147.57 281.30 279.29  
181 2008-01-02 0.000005 0.003926 25.05 21.99 0.0 154.70 281.18 279.55  
203 2009-01-12 0.000024 0.004962 24.62 22.53 0.0 184.49 281.45 279.22  
205 2010-01-02 0.000008 0.005636 28.03 24.80 0.0 154.66 281.08 279.03  
217 2011-01-02 0.000009 0.005736 27.39 23.91 0.0 163.51 281.34 279.97  
239 2012-01-12 0.000023 0.005589 25.50 23.06 0.0 193.34 282.01 280.61

############# AT

# The maxima AT  
df.loc[df.groupby(df['Date'].dt.year)['AT'].idxmax().values]

Date ET SH AT ST P SM LL\_R LL\_G  
3 1993-01-04 0.000026 0.013593 29.18 29.93 0.26 170.93 280.75 278.95  
15 1994-01-04 0.000027 0.012009 29.97 29.62 0.33 168.54 280.79 278.67  
27 1995-01-04 0.000026 0.012817 29.41 29.78 1.11 180.16 280.90 279.35  
40 1996-01-05 0.000007 0.013905 28.99 30.99 2.71 145.97 280.85 279.08  
51 1997-01-04 0.000014 0.030330 29.75 30.39 1.50 152.83 281.09 279.66  
63 1998-01-04 0.000023 0.012474 30.95 30.01 0.17 138.14 281.03 279.21  
75 1999-01-04 0.000019 0.011751 29.52 28.33 0.16 161.60 281.15 279.75  
88 2000-01-05 0.000018 0.010479 29.64 29.79 0.35 177.20 281.02 279.64  
100 2001-01-05 0.000020 0.010483 30.20 29.68 6.17 140.77 280.65 279.07  
111 2002-01-04 0.000013 0.008457 30.39 28.48 0.28 131.68 280.82 279.48  
123 2003-01-04 0.000016 0.010359 29.60 28.06 8.68 147.91 280.96 278.68  
136 2004-01-05 0.000019 0.012244 29.60 29.01 4.50 151.75 280.62 279.16  
147 2005-01-04 0.000012 0.011170 29.62 28.74 0.59 135.57 280.56 278.53  
159 2006-01-04 0.000012 0.008199 29.16 27.69 0.15 142.46 281.12 279.02  
172 2007-01-05 0.000025 0.012808 29.43 29.23 4.62 160.65 280.93 278.52  
182 2008-01-03 0.000011 0.007437 28.75 25.26 0.15 149.00 280.97 279.31  
195 2009-01-04 0.000020 0.011158 29.76 28.69 6.76 155.70 280.84 278.98  
207 2010-01-04 0.000017 0.009313 30.65 28.69 0.00 156.28 280.74 278.56  
220 2011-01-05 0.000021 0.010963 30.00 29.35 5.65 158.53 280.74 279.06  
231 2012-01-04 0.000015 0.008324 31.08 28.64 0.02 143.79 280.74 278.89

# The minima AT  
df.loc[df.groupby(df['Date'].dt.year)['AT'].idxmin().values]

Date ET SH AT ST P SM LL\_R LL\_G  
0 1993-01-01 0.000015 0.006582 21.98 23.02 0.04 184.21 281.54 279.71  
23 1994-01-12 0.000025 0.006703 22.05 21.34 0.00 210.80 281.87 280.24  
24 1995-01-01 0.000018 0.006002 22.49 21.84 0.04 193.07 281.65 280.15  
36 1996-01-01 0.000018 0.007157 24.04 22.10 0.04 193.52 281.84 280.19  
59 1997-01-12 0.000009 0.020715 23.43 24.42 0.00 184.64 281.62 279.95  
60 1998-01-01 0.000021 0.006014 23.41 22.64 0.04 167.12 281.46 279.86  
72 1999-01-01 0.000016 0.008082 24.06 21.95 0.04 183.54 281.76 280.56  
95 2000-01-12 0.000013 0.006053 23.57 23.23 0.00 143.71 281.57 279.98  
96 2001-01-01 0.000010 0.004782 22.62 21.87 0.04 131.88 281.46 279.84  
108 2002-01-01 0.000005 0.005055 21.37 21.08 0.04 143.40 281.63 280.17  
131 2003-01-12 0.000017 0.007205 23.37 21.97 0.00 182.82 281.34 280.24  
143 2004-01-12 0.000017 0.007509 23.82 21.96 0.00 174.40 281.24 279.48  
144 2005-01-01 0.000010 0.005843 22.96 20.95 0.04 158.17 281.25 279.25  
167 2006-01-12 0.000010 0.004711 22.19 20.88 0.00 176.98 281.33 279.67  
168 2007-01-01 0.000005 0.004010 22.20 19.63 0.04 160.09 281.45 279.50  
180 2008-01-01 0.000009 0.004930 24.18 22.40 0.04 168.98 280.20 279.76  
203 2009-01-12 0.000024 0.004962 24.62 22.53 0.00 184.49 281.45 279.22  
215 2010-01-12 0.000020 0.005071 24.04 22.33 0.00 194.80 281.99 280.46  
216 2011-01-01 0.000010 0.004176 23.26 20.92 0.04 175.56 281.61 280.27  
228 2012-01-01 0.000007 0.004121 24.60 21.49 1.58 163.59 281.35 279.39

# Input and output data preparation  
# Importing numpy  
#import numpy as np

# The predictors

X = lake.drop(['Date','LL\_R','LL\_G'], axis = 1)  
print(X[:5])

ET SH AT ST P SM  
0 0.000015 0.006582 21.98 23.02 0.04 184.21  
1 0.000011 0.006498 24.86 24.25 0.00 170.82  
2 0.000015 0.009289 28.36 28.43 0.15 163.86  
3 0.000026 0.013593 29.18 29.93 0.26 170.93  
4 0.000032 0.016427 29.03 30.77 11.40 189.82

X.shape

(240, 6)

# The first ouput feature  
y1 = lake['LL\_R']

y1.shape

(240,)

print(y1[:5])

0 281.54  
1 281.16  
2 280.93  
3 280.75  
4 280.59  
Name: LL\_R, dtype: float64

# The second output feature  
y2 = lake['LL\_G']

y2.shape

(240,)

print(y2[:5])

0 279.71  
1 279.51  
2 279.26  
3 278.95  
4 278.65  
Name: LL\_G, dtype: float64

# Standardizing / Scaling the features  
# Importing the StandardScaler function  
from sklearn.preprocessing import StandardScaler

# Initializing and fitting the scaler  
scaler = StandardScaler()  
scaler.fit(X)

StandardScaler()

# Scaling X  
X\_scaled = scaler.transform(X)

# VIF  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
vif = pd.DataFrame()  
vif['vif'] = [variance\_inflation\_factor(X\_scaled,i) for i in range(X\_scaled.shape[1])]  
vif['Features'] = X.columns  
vif.round(2)

vif Features  
0 7.78 ET  
1 2.53 SH  
2 5.64 AT  
3 7.85 ST  
4 1.82 P  
5 8.21 SM

# Splitting the feature data into training and testing datasets  
# Importing train\_test\_splt function  
from sklearn.model\_selection import train\_test\_split  
# Splitting the feature's data  
  
X\_train, X\_test, y1\_train, y1\_test = train\_test\_split(X\_scaled, y1, test\_size = 0.25, random\_state = 30)

print(X\_train.shape)

(180, 6)

# MULTIPLE LINEAR REGRESSION  
# Importing linearRegression module  
from sklearn.linear\_model import LinearRegression

# Remote sensing lake level data (LL\_R) as output feature

#!pip install scikit-learn==0.24.2.

# Instantiation using LinearRegression  
model = LinearRegression(fit\_intercept = True, copy\_X = True, positive = False, n\_jobs = None, normalize = False)

#import sklearn

#print(sklearn.\_\_version\_\_)

# Fitting the linear model  
lake\_lm1 = model.fit(X\_train, y1\_train)

# Outputting the regression results  
import sklearn.metrics

# The model intercept  
print('The intercept is:', lake\_lm1.intercept\_)

The intercept is: 281.2102979783335

# The model coefficients  
print('The model coefficients are:', lake\_lm1.coef\_)

The model coefficients are: [-0.13852157 0.05846207 -0.13894661 -0.07876897 -0.24037903 0.23041285]

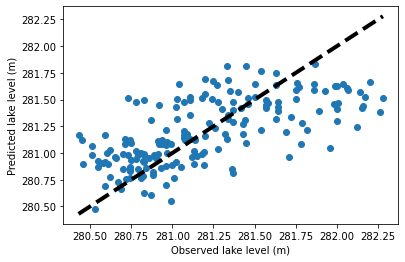
# The coefficient of determination  
  
print('The coefficient of determination is:', lake\_lm1.score(X\_train, y1\_train))

The coefficient of determination is: 0.33533470046182934

# Training Model evaluation  
from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error, explained\_variance\_score  
import numpy as np

# Finding metrics on X\_train  
  
ytrain\_predlm1 = lake\_lm1.predict(X\_train)

# Plotting the scatter plot of the correlation test  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_train, ytrain\_predlm1)  
ax.plot([y1\_train.min(),y1\_train.max()], [y1\_train.min(), y1\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values  
from numpy import cov  
covtrainlm1 = cov(y1\_train, ytrain\_predlm1)  
print(covtrainlm1)

[[0.21501729 0.09432887]  
 [0.09432887 0.09432887]]

# Computing the pearson correlation between the observed and predicted values  
from scipy.stats import pearsonr  
cortrainlm1 = pearsonr(y1\_train, ytrain\_predlm1)  
print(cortrainlm1)

(0.6623470836627627, 4.26684275586793e-24)

# lm1 MSE  
print('The lm1 MSE is:%.2f'% mean\_squared\_error(y1\_train, ytrain\_predlm1))

The lm1 MSE is:0.14

# lm1 RMSE  
print('The lm1 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y1\_train, ytrain\_predlm1)))

The lm1 RMSE is: 0.38

#lm1 MAE  
print('lm1 MAE is:', mean\_absolute\_error(y1\_train, ytrain\_predlm1))

lm1 MAE is: 0.29508547381578326

#lm1 ESV  
print('lm1 EVS is:', explained\_variance\_score(y1\_train, ytrain\_predlm1))

lm1 EVS is: 0.33533470046182934

# Prediction  
pred\_lm1 = lake\_lm1.predict(X\_test)  
  
print('The predicted lake level data is:', pred\_lm1[:5])

The predicted lake level data is: [281.52491252 280.75850639 281.34454981 280.65884578 281.86473911]

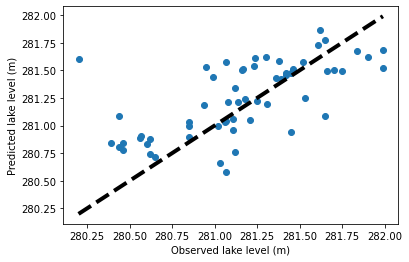
# To save the predicted data  
import numpy   
numpy.savetxt('E:/Lake Level/LM/lmpredLLR.csv', pred\_lm1, delimiter = ',')

# The prediction r\_sq  
from sklearn.metrics import r2\_score

# The Prediction coefficient of determination  
print('The coefficient of determination of the prediction is:',r2\_score(y1\_test,pred\_lm1) )

The coefficient of determination of the prediction is: 0.3803122681875505

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_lm1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values  
from numpy import cov  
covtestlm1 = cov(y1\_test, pred\_lm1)  
print(covtestlm1)

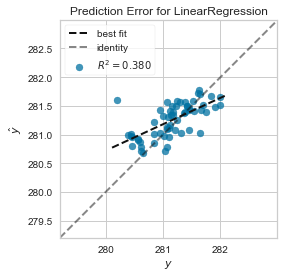
[[0.1852037 0.09448399]  
 [0.09448399 0.11132215]]

# Computing the pearson correlation between the observed and predicted values  
from scipy.stats import pearsonr  
cortestlm1 = pearsonr(y1\_test, pred\_lm1)  
print(cortestlm1)

(0.658025328736949, 1.1093139575506476e-08)

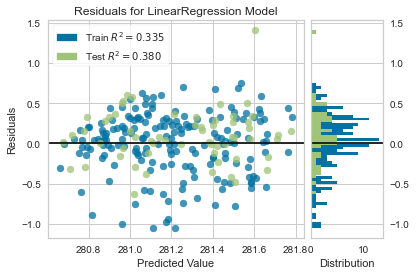
# Plotting the prediction error and residuals  
from sklearn.preprocessing import StandardScaler

# Plotting the prediction error  
from yellowbrick.regressor import PredictionError  
visualizer = PredictionError(lake\_lm1)  
visualizer.fit(X\_train, y1\_train)  
visualizer.score(X\_test, y1\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Prediction Error for LinearRegression'}, xlabel='$y$', ylabel='$\\hat{y}$'>

# Plotting the residuals  
from yellowbrick.regressor import ResidualsPlot  
visualizer = ResidualsPlot(lake\_lm1)  
visualizer.fit(X\_train, y1\_train)  
visualizer.score(X\_test, y1\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>

# Model evaluation  
from sklearn import metrics  
import numpy as np

# The mean squared error, MSE  
print('The RS Data LM MSE is:', metrics.mean\_squared\_error(y1\_test, pred\_lm1))

The RS Data LM MSE is: 0.11285565344093959

# The root mean squared error, RMSE  
print('The RS Data LM RMSE is:', np.sqrt(metrics.mean\_squared\_error(y1\_test, pred\_lm1)))

The RS Data LM RMSE is: 0.3359399551124272

# The mean absolute error, MAE  
print('The RS Data LM MAE is:', metrics.mean\_absolute\_error(y1\_test, pred\_lm1))

The RS Data LM MAE is: 0.24893065509151693

# The lm1 Explained variance score  
from sklearn.metrics import explained\_variance\_score  
  
print('lm1 EVS is:', explained\_variance\_score(y1\_test, pred\_lm1))

lm1 EVS is: 0.4317974771399863

# The k-fold cross-validation  
from sklearn.model\_selection import cross\_val\_score

# On the training dataset  
score\_train = cross\_val\_score(lake\_lm1, X\_train, y1\_train, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score\_train

array([-0.18999899, -0.0956805 , -0.07674422, -0.15468979, -0.16257589,  
 -0.07955097, -0.2536322 , -0.17740024, -0.15428534, -0.18194307])

# The absolute mean score on the training dataset  
from numpy import absolute  
  
print(absolute(np.mean(score\_train)))

0.1526501199982468

# On the testing dataset  
score\_test = cross\_val\_score(lake\_lm1, X\_test, y1\_test, scoring ='neg\_mean\_squared\_error', cv = 10)  
score\_test

array([-0.17163243, -0.06033246, -0.05589898, -0.05185005, -0.05639866,  
 -0.34649628, -0.13915611, -0.07949392, -0.08599598, -0.07633582])

# The absolute mean score on the testing dataset  
print(absolute(np.mean(score\_test)))

0.11235906804499292

# Feature Importance

# Importing library/module  
from matplotlib import pyplot

# Defining the model  
model = LinearRegression(fit\_intercept = True, copy\_X = True, positive = False, n\_jobs = None, normalize = False)

# Fitting the model  
lm1 = model.fit(X, y1)

# Getting the importance  
importance = lm1.coef\_

# Summarizing the feature importance  
for i, y1 in enumerate(importance):  
 print('X%0d,Score:%.2f'%(i,y1))

X0,Score:-14528.55  
X1,Score:14.02  
X2,Score:-0.06  
X3,Score:-0.03  
X4,Score:-0.00  
X5,Score:0.01

# Plotting the feature inportance  
pyplot.bar([X for X in range(len(importance))], importance)  
pyplot.show()



# Ground truth lake level data (LL\_G) as ouput feature

# Splitting the feature data into training and testing datasets  
# Importing train\_test\_splt function  
from sklearn.model\_selection import train\_test\_split  
# Splitting the feature's data  
  
X\_train, X\_test, y2\_train, y2\_test = train\_test\_split(X\_scaled, y2, test\_size = 0.25, random\_state = 30)

# Instantiating and fitting the linear model  
lake\_lm2 = LinearRegression(fit\_intercept = True, copy\_X = True, positive = False, n\_jobs = None, normalize = False).fit(X\_train, y2\_train)

# The model intercept  
print('The intercept is:', lake\_lm2.intercept\_)

The intercept is: 279.22738339573084

# The model coefficients  
print('The coefficients are:', lake\_lm2.coef\_)

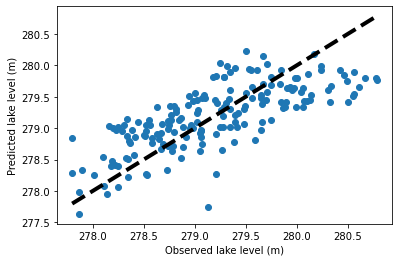
The coefficients are: [-0.1605134 0.06105699 -0.05889774 -0.12036017 -0.46145285 0.17876163]

# The coefficient of determination  
  
print('The coefficient of determination is:', lake\_lm2.score(X\_train, y2\_train))

The coefficient of determination is: 0.5159034226917156

# Predicting on X\_train  
  
ytrain\_predlm2 = lake\_lm2.predict(X\_train)

# Plotting the scatter plot of the correlation test  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_train, ytrain\_predlm2)  
ax.plot([y2\_train.min(),y2\_train.max()], [y2\_train.min(), y2\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values  
from numpy import cov  
covtrainlm2 = cov(y2\_train, ytrain\_predlm2)  
print(covtrainlm2)

[[0.49448696 0.26666971]  
 [0.26666971 0.26666971]]

# Computing the pearson correlation between the observed and predicted values  
from scipy.stats import pearsonr  
cortrainlm2 = pearsonr(y2\_train, ytrain\_predlm2)  
print(cortrainlm2)

(0.7343606877171274, 8.986703716768093e-32)

# lm2 MSE  
print('The lm2 MSE is:%.2f'% mean\_squared\_error(y2\_train, ytrain\_predlm2))

The lm2 MSE is:0.24

# lm2 RMSE  
print('The lm2 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y2\_train, ytrain\_predlm2)))

The lm2 RMSE is: 0.49

#lm2 MAE  
print('lm2 MAE is:', mean\_absolute\_error(y2\_train, ytrain\_predlm2))

lm2 MAE is: 0.4016509439683416

#lm2  
print('lm2 EVS is:', explained\_variance\_score(y2\_train, ytrain\_predlm2))

lm2 EVS is: 0.5159034226917156

# Prediction  
pred\_lm2 = lake\_lm2.predict(X\_test)

print('The predicted lake level data is:', pred\_lm2[:10])

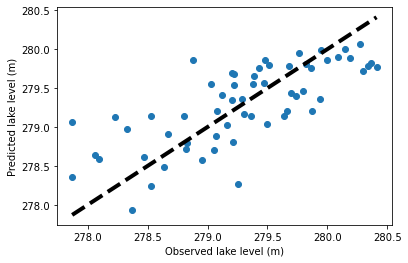
The predicted lake level data is: [279.80509043 278.48757324 279.76620135 278.9773065 280.11570299  
 279.67213438 279.18134699 279.39731305 278.53129628 279.31429559]

# To save the predited data on C  
numpy.savetxt("E:/Lake Level/LM/lmpredLLG.csv", pred\_lm2, delimiter = ',')

# The Prediction coefficient of determination  
print('The coefficient of determination of the prediction is:',r2\_score(y2\_test,pred\_lm2) )

The coefficient of determination of the prediction is: 0.511186062807089

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_test, pred\_lm2)  
ax.plot([y2\_test.min(),y2\_test.max()], [y2\_test.min(), y2\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values  
from numpy import cov  
covtestlm2 = cov(y2\_test, pred\_lm2)  
print(covtestlm2)

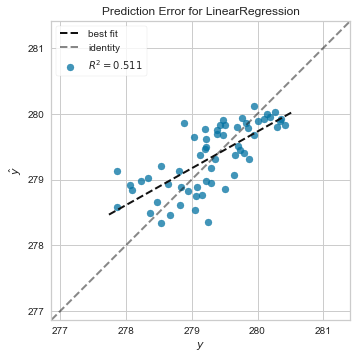
[[0.42726607 0.25112988]  
 [0.25112988 0.27090627]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestlm2 = pearsonr(y2\_test, pred\_lm2)  
print(cortestlm2)

(0.738141563025427, 1.7012110847799302e-11)

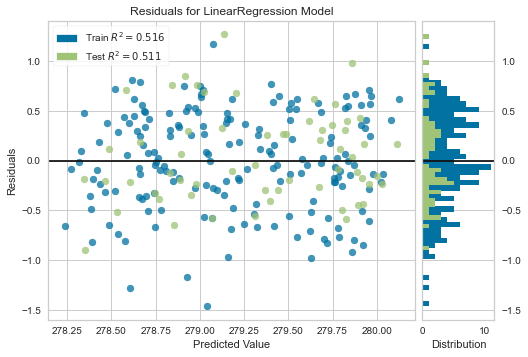
# Plotting the prediction error and residuals

# Plotting the prediction errors  
from yellowbrick.regressor import PredictionError  
visualizer = PredictionError(lake\_lm2)  
visualizer.fit(X\_train, y2\_train)  
visualizer.score(X\_test, y2\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Prediction Error for LinearRegression'}, xlabel='$y$', ylabel='$\\hat{y}$'>

# Plotting the residuals  
from yellowbrick.regressor import ResidualsPlot  
visualizer = ResidualsPlot(lake\_lm2)  
visualizer.fit(X\_train, y2\_train)  
visualizer.score(X\_test, y2\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Residuals for LinearRegression Model'}, xlabel='Predicted Value', ylabel='Residuals'>

# Model evaluation  
from sklearn import metrics

# The mean squared error, MSE  
print('The GT Data LM MSE is:', metrics.mean\_squared\_error(y2\_test, pred\_lm2))

The GT Data LM MSE is: 0.20537271806375104

# The root mean squared error, RMSE  
print('The GT Data LM RMSE is:', np.sqrt(metrics.mean\_squared\_error(y2\_test, pred\_lm2)))

The GT Data LM RMSE is: 0.45318066823701897

# The mean absolute error, MAE  
print('The GT Data LM MAE is:', metrics.mean\_absolute\_error(y2\_test, pred\_lm2))

The GT Data LM MAE is: 0.3754120924538853

# The lm2 Explained variance score  
print('lm2 EVS is:', explained\_variance\_score(y2\_test, pred\_lm2))

lm2 EVS is: 0.5155597257825832

# The k-fold cross-validation

# On the training dataset  
score\_train = cross\_val\_score(lake\_lm2, X\_train, y2\_train, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score\_train

array([-0.20948844, -0.20920257, -0.23921725, -0.33410696, -0.36246004,  
 -0.26025418, -0.1891152 , -0.28579809, -0.22844637, -0.3240588 ])

# The absolute mean score on the training dataset  
print(absolute(np.mean(score\_train)))

0.26421478976445323

# On the testing dataset  
score\_test = cross\_val\_score(lake\_lm2, X\_test, y2\_test, scoring ='neg\_mean\_squared\_error', cv = 10)  
score\_test

array([-0.17960741, -0.19366776, -0.25614177, -0.1531712 , -0.0980852 ,  
 -0.14555479, -0.18508221, -0.07965864, -0.33279951, -0.36760442])

# The absolute mean score on the testing dataset  
print(absolute(np.mean(score\_test)))

0.19913729121530105

# Feature Importance

# Model definition  
model = LinearRegression(fit\_intercept = True, copy\_X = True, positive = False, n\_jobs = None, normalize = False)

# Fitting the model  
lm2 = model.fit(X, y2)

# Getting the importance  
importance = lm2.coef\_

# Summarizing the importance  
for i, y2 in enumerate(importance):  
 print('X:%0d, Score:%.2f'%(i,y2))

X:0, Score:-21590.41  
X:1, Score:2.02  
X:2, Score:-0.08  
X:3, Score:-0.02  
X:4, Score:-0.01  
X:5, Score:0.00

# Plotting the importance  
pyplot.bar([X for X in range(len(importance))], importance)  
pyplot.show()



#

############################################################################

# SUPPORT VECTOR REGRESSION MODELS  
from sklearn.svm import SVR

# Remote sensing lake level data as output feature  
import warnings;warnings.simplefilter('ignore')

# Create the svr regressor and fitting the model   
lake\_svr1 = SVR(C = 100).fit(X, y1)

# Printing the model  
print(lake\_svr1)

SVR(C=100)

# The coefficient of determination  
  
#print('The svr1 coefficient of determination on the datset is: %.2f'% lake\_svr1.score(X, y1))

The svr1 coefficient of determination on the datset is: 0.46

# Model with best parameter values

# kernel='rbf'The svr1 R2 on the datset is: 0.29  
# kernel='linear': : 0.31  
# kernel='poly': 0.22  
# kernel='sigmoid': -212.50

# epsilon=0.0001: 0.30  
# epsilon=0.001: : 0.30  
# epsilon=0.01: 0.29  
# epsilon=0.1: 0.29  
# epsilon= 1: -0.03

#C  
# C = 1: 0.29  
# C = 30: 0.43  
# C = 60: 0.44  
# C = 90: 0.45  
# C = 100: 0.45

# Training model

# Create the svr regressor and fitting the model with the best parameter values  
lake\_svr1 = SVR(kernel = 'rbf', epsilon = 0.1, C = 90).fit(X\_train, y1\_train)

# Printing the model  
print(lake\_svr1)

SVR(C=90)

# Training Model evaluation  
from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error, explained\_variance\_score

# kernel='rbf': The svr1 MSE is:0.15  
# kernel='linear':0.14  
# kernel='poly': 0.16  
# kernel='sigmoid':44.04

# epsilon=0.0001: 0.15  
# epsilon=0.001: 0.15  
# epsilon=0.01: 0.15  
# epsilon=0.1: 0.15  
# epsilon= 1: 0.21

#C  
# C = 1: 0.15  
# C = 30: 0.12  
# C = 60: 0.12  
# C = 90: 0.11  
# C = 100: 0.11  
# C = 120: 0.11  
# C = 150: 0.11  
# C = 180: 0.11

# Predicting on X  
  
y1\_predsvr1 = lake\_svr1.predict(X)

# svr1 MSE  
print('The svr1 MSE is:%.2f'% mean\_squared\_error(y1, y1\_predsvr1))

The svr1 MSE is:0.11

# svr1 RMSE  
print('The svr1 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y1, y1\_predsvr1)))

The svr1 RMSE is: 0.33

# kernel='rbf': The svr1 RMSE is: 0.38  
# kernel='linear': 0.38  
# kernel='poly': 0.40  
# kernel='sigmoid': 6.64

# epsilon=0.0001: 0.38  
# epsilon=0.001: 0.38  
# epsilon=0.01: 0.38  
# epsilon=0.1: 0.38  
# epsilon= 1: 0.46

#C  
# C = 1: 0.38  
# C = 30: 0.34  
# C = 60: 0.34  
# C = 90: 0.34  
# C = 100: 0.34

#svr1 MAE  
print('svr1 MAE is:', mean\_absolute\_error(y1, y1\_predsvr1))

svr1 MAE is: 0.2393468635035492

# kernel='rbf': svr1 MAE is: 0.28727940395504087  
# kernel='linear': 0.2872204529444763  
# kernel='poly': 0.31688947551197894  
# kernel='sigmoid': 5.216388421951588

# epsilon=0.0001: 0.2862995263130664  
# epsilon=0.001: 0.28627654611490716  
# epsilon=0.01: 0.2864764744850383  
# epsilon=0.1: 0.28727940395504087  
# epsilon= 1: 0.3845410837455399

#C  
# C = 1: 0.28727940395504087  
# C = 30: 0.2501249625100658  
# C = 60: 0.2456727282123005  
# C = 90: 0.24233600545218484  
# C = 100: 0.242050890067461926  
# C = 120: 0.24114567024881903  
# C = 150: 0.24000868880481907  
# C = 180: 0.2393468635035492

#svr1 evs  
#print('svr1 EVS is:', explained\_variance\_score(y1, y1\_predsvr1))

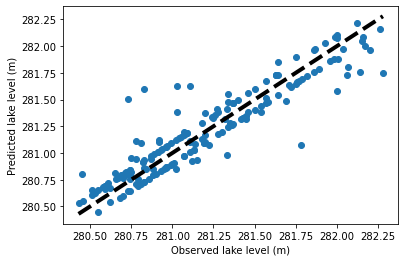
# kernel='rbf': svr1 EVS is: 0.30867308307824015  
# kernel='linear': 0.3155049712964756  
# kernel='poly': 0.22575004042882973  
# kernel='sigmoid': -212.18242135964184

# epsilon=0.0001: 0.3073320163851938  
# epsilon=0.001: 0.3072859729561842  
# epsilon=0.01: 0.30624137560173026  
# epsilon=0.1: 0.30867308307824015  
# epsilon= 1: -0.01446956727010651

#C  
# C = 1: 0.30867308307824015  
# C = 30: 0.43547182040642496  
# C = 60: 0.44468974146071627  
# C = 90: 0.454396481408593  
# C = 100: 0.45559235242831153  
# C = 120: 0.46089218245858377  
# C = 150: 0.46482071332405883  
# C = 180: 0.4681352414894896

# Predicting on X\_train  
  
ytrain\_predsvr1 = lake\_svr1.predict(X\_train)

# # Plotting the scatter plot of the correlation test/rbf  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_train, ytrain\_predsvr1)  
ax.plot([y1\_train.min(),y1\_train.max()], [y1\_train.min(), y1\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtrainsvr1 = cov(y1\_train, ytrain\_predsvr1)  
print(covtrainsvr1)

[[0.21501729 0.1819498 ]  
 [0.1819498 0.18000347]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortrainsvr1 = pearsonr(y1\_train, ytrain\_predsvr1)  
print(cortrainsvr1)

(0.924856719808464, 1.1893786584637652e-76)

# The coefficient of determination  
  
print('The svr1 coefficient of determination on the training dataset is: %.2f'% lake\_svr1.score(X\_train, y1\_train))

The svr1 coefficient of determination on the training dataset is: 0.88

# The svr1 coefficient of determination on the training dataset is: 0.88

# svr1 MSE  
print('The svr1 MSE is:%.2f'% mean\_squared\_error(y1\_train, ytrain\_predsvr1))

The svr1 MSE is:0.03

# svr1 RMSE  
print('The svr1 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y1\_train, ytrain\_predsvr1)))

The svr1 RMSE is: 0.16

#svr1 MAE  
print('svr1 MAE is:', mean\_absolute\_error(y1\_train, ytrain\_predsvr1))

svr1 MAE is: 0.12116343176912052

#svr1 evs  
print('svr1 EVS is:', explained\_variance\_score(y1\_train, ytrain\_predsvr1))

svr1 EVS is: 0.8810667780769954

# To save the predicted data on the drive  
import numpy  
numpy.savetxt('E:/Lake Level/SVR/svrLLRtrainbest.csv', ytrain\_predsvr1, delimiter = ',')

# Prediction on the test dataset  
pred\_svr1 = lake\_svr1.predict(X\_test)

print(pred\_svr1[:5])

[281.25104801 281.00939767 281.24874777 281.19667909 282.00410417]

# To save the predicted data on the drive  
numpy.savetxt('E:/Lake Level/SVR/svrLLRtestbest.csv', pred\_svr1, delimiter = ',')

# The prediction r\_sq  
from sklearn.metrics import r2\_score  
# The Prediction coefficient of determination  
print('The SVR1 coefficient of determination of the prediction is:',r2\_score(y1\_test, pred\_svr1) )

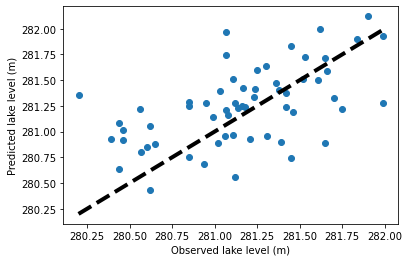
The SVR1 coefficient of determination of the prediction is: 0.3317527671954451

# kernel='rbf': The SVR1 coefficient of determination of the prediction is: 0.4943227806230167  
# kernel='linear': 0.39855455316675503  
# kernel='poly': 0.23720266995281614  
# kernel='sigmoid': -54.70072444238841

# epsilon=0.0001: 0.5109983688632853  
# epsilon=0.001: 0.511178050789183  
# epsilon=0.01: 0.5092831205293169  
# epsilon=0.1: 0.4941931598640047  
# epsilon= 1: -0.22220898625750274

#C  
# C = 1: 0.4941931598640047  
# C = 30: 0.3954988090266349  
# C = 60: 0.36169948002954755  
# C = 90: 0.3317527671954451  
# C = 100: 0.3197972887013947

# Plotting the predicted against the observed data/rbf  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_svr1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



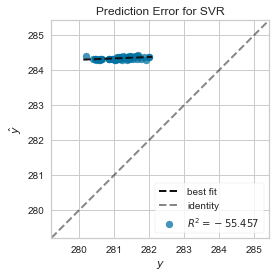
# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtestsvr1 = cov(y1\_test, pred\_svr1)  
print(covtestsvr1)

[[0.1852037 0.09059144]  
 [0.09059144 0.14402264]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestsvr1 = pearsonr(y1\_test, pred\_svr1)  
print(cortestsvr1)

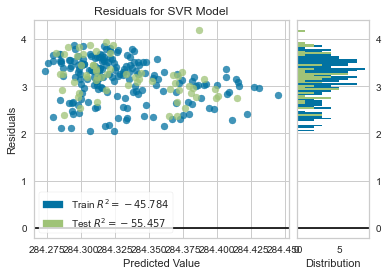
(0.5546857005396336, 4.248646591145792e-06)

# Plotting the prediction errors/rbf  
from yellowbrick.regressor import PredictionError  
visualizer = PredictionError(lake\_svr1)  
visualizer.fit(X\_train, y1\_train)  
visualizer.score(X\_test, y1\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Prediction Error for SVR'}, xlabel='$y$', ylabel='$\\hat{y}$'>

# Plotting the residuals/rbf  
from yellowbrick.regressor import ResidualsPlot  
visualizer = ResidualsPlot(lake\_svr1)  
visualizer.fit(X\_train, y1\_train)  
visualizer.score(X\_test, y1\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Residuals for SVR Model'}, xlabel='Predicted Value', ylabel='Residuals'>

# Model evaluation

# The MSE  
   
print('The RS Data SVR1 MSE is:',metrics.mean\_squared\_error(y1\_test, pred\_svr1) )

The RS Data SVR1 MSE is: 0.1216991627342438

# kernel='rbf': The RS Data SVR1 MSE is: 0.0920924041146885  
# kernel='linear': 0.10953342373411194  
# kernel='poly': 0.13891834016738805  
# kernel='sigmoid': 10.14404728603209

# epsilon=0.0001: 0.08905549647434631  
# epsilon=0.001: 0.08902277334603874  
# epsilon=0.01: 0.08936787230753783  
# epsilon=0.1: 0.09211601025485691  
# epsilon= 1: 0.22258500000000767

#C  
# C = 1: 0.09211601025485691  
# C = 30: 0.1100899266047964  
# C = 60: 0.11624535806488873  
# C = 90: 0.1216991627342438  
# C = 100: 0.12387645827904817

# The RMSE  
  
print('The RS Data svr1 RMSE is:',np.sqrt(metrics.mean\_squared\_error(y1\_test, pred\_svr1)))

The RS Data svr1 RMSE is: 0.3488540708293997

# kernel='rbf': The RS Data svr1 RMSE is: 0.3034673032052852  
# kernel='linear': : 0.330958341387722  
# kernel='poly': 0.37271750719195906  
# kernel='sigmoid': 3.184972101295722

# epsilon=0.0001: 0.2984216756107812  
# epsilon=0.001:0.298366843576894  
# epsilon=0.01: 0.29894459738810775  
# epsilon=0.1: 0.3035061947553244  
# epsilon= 1: 0.47178914781924314

#C  
# C = 1: 0.3035061947553244  
# C = 30: 0.33179802079698484  
# C = 60: 0.3409477350927686  
# C = 90: 0.3488540708293997  
# C = 100: 0.3519608760630195

# The MAE  
  
print('The RS Data svr1 MAE is: %.2f'% metrics.mean\_absolute\_error(y1\_test, pred\_svr1))

The RS Data svr1 MAE is: 0.27

# kernel='rbf': The RS Data svr1 MAE is: 0.23  
# kernel='linear': 0.24  
# kernel='poly': 0.29  
# kernel='sigmoid': 2.61

# Epsilon  
# epsilon=0.0001: 0.23  
# epsilon=0.001: 0.23  
# epsilon=0.01: 0.23  
# epsilon=0.1: 0.23  
# epsilon= 1: 0.38

#C  
# C = 1: 0.23  
# C = 30: 0.25  
# C = 60: 0.26  
# C = 90: 0.27  
# C = 100: 0.27

# The svr1 Explained variance score  
from sklearn.metrics import explained\_variance\_score  
# y1 EVS  
print('svr1 EVS is:', explained\_variance\_score(y1\_test, pred\_svr1))

svr1 EVS is: 0.3600150834584971

# kernel='rbf': svr1 EVS is: 0.5118141602562055  
# kernel='linear': : 0.4165193786669812  
# kernel='poly': 0.23754022188475876  
# kernel='sigmoid': -49.69570451472918

# Epsilon  
# epsilon=0.0001: 0.5192840653142299  
# epsilon=0.001: 0.5193931649931952  
# epsilon=0.01: 0.517648097999043  
# epsilon=0.1: 0.5117159412110379  
# epsilon= 1: 3.3306690738754696e-16

#C  
# C = 1: 0.5117159412110379  
# C = 30: 0.41840483311824084  
# C = 60: 0.3915994764068782  
# C = 90: 0.3600150834584971  
# C = 100: 0.34732301902324625

# The k-fold cross-validation  
from sklearn.model\_selection import cross\_val\_score   
from numpy import absolute

# On the whole dataset  
#score = cross\_val\_score(lake\_svr1, X, y1, scoring = 'neg\_mean\_squared\_error', cv = 10)  
#score

# The absolute mean score on the training dataset  
#print(absolute(np.mean(score)))

# kernel='rbf':0.16397520084995906  
# kernel='linear': 0.3155049712964756  
# kernel='poly': 0.1730794882491565  
# kernel='sigmoid':35.01227331499174

# Epsilon  
# epsilon=0.0001: 0.1606687501256016  
# epsilon=0.001: 0.1607726373440683  
# epsilon=0.01: 0.16125619535649127  
# epsilon=0.1: 0.16397520084995906  
# epsilon= 1: 0.21741823244213565

#C  
# C = 1: 0.16397520084995906  
# C = 30: 0.14148431917053442  
# C = 60: 0.1416287596323649  
# C = 90: 0.14357295211315863  
# C = 100: 0.1441800870171332  
# C = 120: 0.14469116353289535  
# C = 150: 0.1441740099368459  
# C = 180: 0.144556637673928

# On the training dataset  
score\_train = cross\_val\_score(lake\_svr1, X\_train, y1\_train, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score\_train

array([-0.12445773, -0.07446356, -0.18308787, -0.16373802, -0.08378953,  
 -0.17890533, -0.31451666, -0.19024435, -0.04028723, -0.14348043])

# The absolute mean score on the training dataset  
print(absolute(np.mean(score\_train)))

0.1496970704621687

# On the testing dataset  
score\_test = cross\_val\_score(lake\_svr1, X\_test, y1\_test, scoring ='neg\_mean\_squared\_error', cv = 10)  
score\_test

array([-0.27951271, -0.20133513, -0.23670822, -0.10888955, -0.91438922,  
 -0.33277175, -0.12435296, -0.21369239, -0.23864116, -0.12478305])

# The absolute mean score on the testing dataset  
print(absolute(np.mean(score\_test)))

0.2775076148227219

# kernel='rbf': 0.10955348983437949  
# kernel='linear': 0.10766877844119409  
# kernel='poly': 0.1610812802042855  
# kernel='sigmoid':1.1448867358304273

# Epsilon  
# epsilon=0.0001: 0.1148444410244986  
# epsilon=0.001: 0.1148217881770672  
# epsilon=0.01: 0.1142746793961578  
# epsilon=0.1: 0.10925383677267424  
# epsilon= 1: 0.18869749999999574

#C  
# C = 1: 0.10925383677267424  
# C = 30: 0.19237953393711327  
# C = 60: 0.2312731586159265  
# C = 90: 0.2775076148227219  
# C = 100: 0.2939641920945568

# Feature importance

# Importing libraries.modules  
from sklearn.inspection import permutation\_importance  
import matplotlib.pyplot as plt  
import numpy as np  
%matplotlib inline

# Creating and fitting the model  
from sklearn.svm import SVR  
svr1 = SVR(kernel='rbf', C=90, epsilon = 0.1).fit(X, y1)

# Performing the importance  
f\_importance = permutation\_importance(svr1, X, y1)

# Getting the importance  
importance = f\_importance.importances\_mean

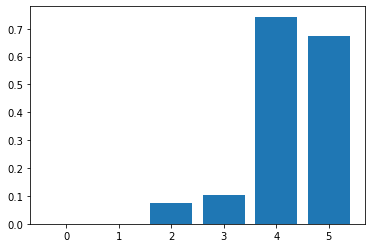
# Summrizing the feature importance scores  
for i, y1 in enumerate(importance):  
 print('X:%0d, Score:%.5f'%(i,y1))

X:0, Score:0.00000  
X:1, Score:0.00000  
X:2, Score:0.07479  
X:3, Score:0.10162  
X:4, Score:0.74329  
X:5, Score:0.67317

# Creating an array of the features  
f\_names=['ET','SH','AT','ST','P','SM']  
features = np.array(f\_names)

# Plotting the importances  
plt.bar([X for X in range(len(importance))], importance)

<BarContainer object of 6 artists>



# SVR: Ground truth lake level data as output feature

# SUPPORT VECTOR REGRESSION MODELS  
from sklearn.svm import SVR

# Create the svr regressor and fitting the model using the whole dataset  
#lake\_svr2 = SVR(C = 100).fit(X, y2)

# Printing the model  
#print(lake\_svr2)

# The coefficient of determination  
  
print('The svr2 coefficient of determination on the dataset is: %.2f'% lake\_svr2.score(X, y2))

The svr2 coefficient of determination on the dataset is: 0.54

# kernel='rbf': The svr2 R2 on the dataset is: 0.50  
# kernel='linear': 0.49  
# kernel='poly': 0.42  
# kernel='sigmoid': -93.62

# Epsilon  
# epsilon=0.0001: 0.50  
# epsilon=0.001: 0.50  
# epsilon=0.01: 0.50  
# epsilon=0.1: 0.50  
# epsilon= 1: 0.42

#C  
# C = 1: 0.50  
# C = 30: 0.53  
# C = 60: 0.54  
# C = 90: 0.54  
# C = 100: 0.54

# Training Model evaluation  
from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error, explained\_variance\_score

# Predicting on X  
  
y2\_predsvr2 = lake\_svr2.predict(X)

# svr2 MSE  
print('The svr2 MSE is:%.2f'% mean\_squared\_error(y2, y2\_predsvr2))

The svr2 MSE is:0.22

# kernel='rbf':The svr2 MSE is:0.14  
# kernel='linear'::0.24  
# kernel='poly':0.27  
# kernel='sigmoid':45.01

# Epsilon  
# epsilon=0.0001: 0.24  
# epsilon=0.001: 0.24  
# epsilon=0.01: 0.24  
# epsilon=0.1: 0.24  
# epsilon= 1: 0.28

#C  
# C = 1: 0.24  
# C = 30: 0.23  
# C = 60: 0.23  
# C = 90: 0.22  
# C = 100: 0.22

# svr2 RMSE  
print('The svr2 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y2, y2\_predsvr2)))

The svr2 RMSE is: 0.47

# kernel='rbf':The svr2 RMSE is: 0.49  
# kernel='linear': 0.49  
# kernel='poly': 0.52  
# kernel='sigmoid': 6.71

# Epsilon  
# epsilon=0.0001: 0.49  
# epsilon=0.001:0.49  
# epsilon=0.01: 0.49  
# epsilon=0.1: 0.49  
# epsilon= 1: 0.53

#C  
# C = 1: 0.49  
# C = 30: 0.48  
# C = 60: 0.48  
# C = 90: 0.47  
# C = 100: 0.47

#svr2 MAE  
print('svr2 MAE is:', mean\_absolute\_error(y2, y2\_predsvr2))

svr2 MAE is: 0.36613515018713916

# kernel='rbf': svr2 MAE is: 0.39164994281203486  
# kernel='linear': 0.3964212056056856  
# kernel='poly': 0.42177377501722096  
# kernel='sigmoid': 5.13959768733391

# Epsilon  
# epsilon=0.0001: 0.390343498454182  
# epsilon=0.001: 0.3902871023122029  
# epsilon=0.01: 0.3906135522774517  
# epsilon=0.1: 0.39164994281203486  
# epsilon= 1: 0.43015755462643857

#C  
# C = 1: 0.39164994281203486  
# C = 30: 0.37640779802555  
# C = 60: 0.37640779802555  
# C = 90: 0.36722515452544824  
# C = 100: 0.36613515018713916

#svr2 EVS  
print('svr2 EVS is:', explained\_variance\_score(y2, y2\_predsvr2))

svr2 EVS is: 0.5433089936980056

# kernel='rbf':svr2 EVS is: 0.5008127523596394  
# kernel='linear': 0.493300758862116  
# kernel='poly': 0.425021392526331  
# kernel='sigmoid': -93.62454951755745

# Epsilon  
# epsilon=0.0001: 0.5003928064554193  
# epsilon=0.001: 0.5004551372749682  
# epsilon=0.01: 0.5005294711163736  
# epsilon=0.1: 0.5008127523596394  
# epsilon= 1: 0.4467904351189198

#C  
# C = 1: 0.5008127523596394  
# C = 30: 0.5270414718499628  
# C = 60: 0.5270414718499628  
# C = 90: 0.5421174333017775  
# C = 100: 0.5433089936980056

# Create the svr regressor and fitting the training model  
lake\_svr2 = SVR(kernel = 'rbf', epsilon = 0.01, C = 90).fit(X\_train, y2\_train)

# Printing the model  
print(lake\_svr2)

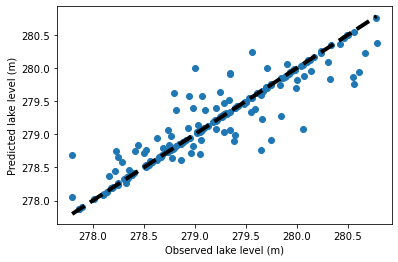
SVR(C=90, epsilon=0.01)

# The coefficient of determination  
  
print('The svr2 coefficient of determination on the training dataset is: %.2f'% lake\_svr2.score(X\_train, y2\_train))

The svr2 coefficient of determination on the training dataset is: 0.88

# Predicting on X\_train  
  
ytrain\_predsvr2 = lake\_svr2.predict(X\_train)

# # Plotting the scatter plot of the correlation test/rbf  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_train, ytrain\_predsvr2)  
ax.plot([y2\_train.min(),y2\_train.max()], [y2\_train.min(), y2\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtrainsvr2 = cov(y2\_train, ytrain\_predsvr2)  
print(covtrainsvr2)

[[0.49448696 0.41952764]  
 [0.41952764 0.41582125]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortrainsvr2 = pearsonr(y2\_train, ytrain\_predsvr2)  
print(cortrainsvr2)

(0.9251876188796462, 8.151304104269273e-77)

# svr2 MSE  
print('The svr2 MSE is:%.2f'% mean\_squared\_error(y2\_train, ytrain\_predsvr2))

The svr2 MSE is:0.06

# svr2 RMSE  
print('The svr2 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y2\_train, ytrain\_predsvr2)))

The svr2 RMSE is: 0.25

#svr2 MAE  
print('svr2 MAE is:', mean\_absolute\_error(y2\_train, ytrain\_predsvr2))

svr2 MAE is: 0.1403016721923946

#svr2 EVS  
print('svr2 EVS is:', explained\_variance\_score(y2\_train, ytrain\_predsvr2))

svr2 EVS is: 0.8772127924380826

# To save the predicted data on the drive  
numpy.savetxt('E:/Lake Level/SVR/svrLLGtrainbest.csv', ytrain\_predsvr2, delimiter = ',')

# Prediction on testing data  
pred\_svr2 = lake\_svr2.predict(X\_test)

print(pred\_svr2[:10])

[280.29781763 278.4143932 279.49460526 279.91993203 280.86973659  
 280.33568163 278.66049746 278.95828414 279.19948746 278.8474106 ]

# To save the preicted data on the drive  
numpy.savetxt('E:/Lake Level/SVR/svrLLGtestbest.csv', pred\_svr2, delimiter = ',')

# The prediction r\_sq  
from sklearn.metrics import r2\_score  
# The Prediction coefficient of determination  
print('The SVR2 coefficient of determination of the prediction is:',r2\_score(y2\_test, pred\_svr2))

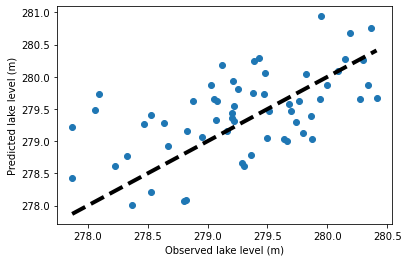
The SVR2 coefficient of determination of the prediction is: 0.30655191152342165

# kernel='rbf': The SVR2 r2 of the prediction is: 0.6249255051397227  
# kernel='linear':0.5459848974249283  
# kernel='poly': 0.44934145083851385  
# kernel='sigmoid'-23.86186644586103

# Epsilon  
# epsilon=0.0001: 0.6288150127984721  
# epsilon=0.001: 0.6287091136654188  
# epsilon=0.01: 0.6274158726336185  
# epsilon=0.1: 0.6250286595969288  
# epsilon= 1: 0.38599170942859473

#C  
# C = 1: 0.6250286595969288  
# C = 30: 0.48883939623435224  
# C = 60: 0.4140441320683296  
# C = 90: 0.3490276363384549  
# C = 100: 0.33012042012580667

# Plotting the predicted against the observed data/rbf  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_test, pred\_svr2)  
ax.plot([y2\_test.min(),y2\_test.max()], [y2\_test.min(), y2\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



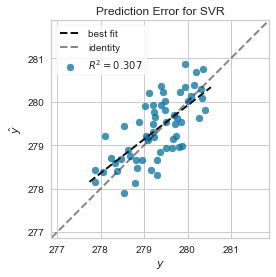
# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtestsvr2 = cov(y2\_test, pred\_svr2)  
print(covtestsvr2)

[[0.42726607 0.23374311]  
 [0.23374311 0.40972793]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestsvr2 = pearsonr(y2\_test, pred\_svr2)  
print(cortestsvr2)

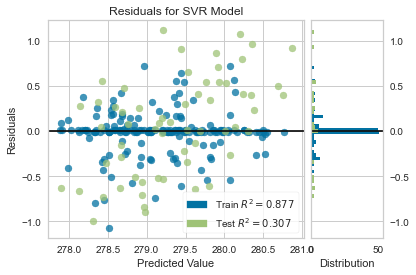
(0.5586526108991863, 3.506224189742129e-06)

# Plotting the prediction errors/rbf  
from yellowbrick.regressor import PredictionError  
visualizer = PredictionError(lake\_svr2)  
visualizer.fit(X\_train, y2\_train)  
visualizer.score(X\_test, y2\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Prediction Error for SVR'}, xlabel='$y$', ylabel='$\\hat{y}$'>

# Plotting the residuals/rbf  
from yellowbrick.regressor import ResidualsPlot  
visualizer = ResidualsPlot(lake\_svr2)  
visualizer.fit(X\_train, y2\_train)  
visualizer.score(X\_test, y2\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Residuals for SVR Model'}, xlabel='Predicted Value', ylabel='Residuals'>

# Model evaluation

# The MSE  
   
print('The G Data svr2 MSE is:', metrics.mean\_squared\_error(y2\_test, pred\_svr2))

The G Data svr2 MSE is: 0.2913487278705456

# kernel='rbf':The G Data svr2 MSE is: 0.1575856632243355  
# kernel='linear': 0.19075216265987321  
# kernel='poly': 0.2313564208413822  
# kernel='sigmoid': 10.4455881872889

# Epsilon  
# epsilon=0.0001: 0.15595150613709213  
# epsilon=0.001: 0.15599599912540713  
# epsilon=0.01: 0.15653934784278953  
# epsilon=0.1: 0.15754232339777807  
# epsilon= 1: 0.25797249618633766

#C  
# C = 1: 0.15754232339777807  
# C = 30: 0.2147615576702128  
# C = 60: 0.24618641185560022  
# C = 90: 0.2735027656480147  
# C = 100: 0.28144653747847737

# The RMSE  
   
print('The G Data svr2 RMSE is:',np.sqrt(metrics.mean\_squared\_error(y2\_test, pred\_svr2)))

The G Data svr2 RMSE is: 0.539767290478541

# kernel='rbf':The G Data svr2 RMSE is: 0.3969706075068222  
# kernel='linear':0.4367518318906896  
# kernel='poly': 0.4809952399363035  
# kernel='sigmoid': 3.231963518867269

# Epsilon  
# epsilon=0.0001: 0.3949069588359923  
# epsilon=0.001: 0.3949632883261521  
# epsilon=0.01: 0.39565053752369594  
# epsilon=0.1: 0.39691601554709033  
# epsilon= 1: 0.5079099292062892

#C  
# C = 1: 0.39691601554709033  
# C = 30: 0.463423734470099  
# C = 60: 0.4961717564065897  
# C = 90: 0.5229749187561624  
# C = 100: 0.5305153508414977

# The MAE  
  
print('The G Data svr2 MAE is: %.2f'% metrics.mean\_absolute\_error(y2\_test, pred\_svr2))

The G Data svr2 MAE is: 0.45

# kernel='rbf':The G Data svr2 MAE is: 0.32  
# kernel='linear': 0.36  
# kernel='poly': 0.40  
# kernel='sigmoid': 2.70

# Epsilon  
# epsilon=0.0001: 0.31  
# epsilon=0.001: 0.31  
# epsilon=0.01: 0.32  
# epsilon=0.1: 0.32  
# epsilon= 1: 0.41

#C  
# C = 1: 0.32  
# C = 30: 0.39  
# C = 60: 0.41  
# C = 90: 0.43  
# C = 100: 0.44

# The svr2 Explained variance score  
from sklearn.metrics import explained\_variance\_score  
# y2 EVS  
print('svr2 EVS is:', explained\_variance\_score(y2\_test, pred\_svr2))

svr2 EVS is: 0.318731370452389

# kernel='rbf':svr2 EVS is: 0.6249264665211749  
# kernel='linear': 0.5471531450612157  
# kernel='poly': 0.459519310110444  
# kernel='sigmoid': -21.908708408075636

# Epsilon  
# epsilon=0.0001: 0.6301512018880294  
# epsilon=0.001: 0.6300363164292188  
# epsilon=0.01: 0.6285595384837214  
# epsilon=0.1: 0.6250300534584043  
# epsilon= 1: 0.40443084207880664

#C  
# C = 1: 0.6250300534584043  
# C = 30: 0.4949581130515428  
# C = 60: 0.4222264332608343  
# C = 90: 0.35999329907480315  
# C = 100: 0.3426825959198754

# The k-fold cross-validation  
from sklearn.model\_selection import cross\_val\_score

# On the whole dataset  
score = cross\_val\_score(lake\_svr2, X, y2, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score

array([-0.43702854, -0.09325348, -0.24306141, -0.60746885, -0.142507 ,  
 -0.18162508, -0.20414984, -0.13968969, -0.35722244, -0.35942753])

# The absolue mean score on the training dataset  
print(absolute(np.mean(score)))

0.2765433851251574

# kernel='rbf': 0.2802652794264413  
# kernel='linear': 0.3057712048203037  
# kernel='poly': 0.3206065848233636  
# kernel='sigmoid': 36.283361113987375

# Epsilon  
# epsilon=0.0001: 0.2802652794264413  
# epsilon=0.001: 0.27441994923003354  
# epsilon=0.01: 0.2737282648611411  
# epsilon=0.1: 0.2802652794264413  
# epsilon= 1: 0.2906078335294981

#C  
# C = 1: 0.22031067617575473  
# C = 30: 0.2821870339070361  
# C = 60: 0.27849889037195036  
# C = 90: 0.2762883534895512  
# C = 100: 0.2765433851251574

# On the training dataset  
score\_train = cross\_val\_score(lake\_svr2, X\_train, y2\_train, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score\_train

array([-0.41713873, -0.26714501, -0.33615911, -0.55121139, -0.29320398,  
 -1.16690154, -0.61156276, -0.40721471, -0.27997189, -0.37128967])

# The absolue mean score on the training dataset  
print(absolute(np.mean(score\_train)))

0.47017987910380715

# On the testing dataset  
score\_test = cross\_val\_score(lake\_svr2, X\_test, y2\_test, scoring ='neg\_mean\_squared\_error', cv = 10)  
score\_test

array([-1.02142373, -0.55560992, -0.4719546 , -0.41650052, -0.48333512,  
 -0.7191149 , -0.379447 , -0.36476588, -0.91965221, -0.26679785])

# The absolute mean score on the testing dataset  
print(absolute(np.mean(score\_test)))

0.5598601735916302

# kernel='rbf': 0.2024515127511612  
# kernel='linear': 0.20294760537611722  
# kernel='poly':0.22804439383203973  
# kernel='sigmoid':0.7930785511107233

# Epsilon  
# epsilon=0.0001: 0.21567398200603577  
# epsilon=0.001: 0.2152940726485296  
# epsilon=0.01: 0.21171297425180602  
# epsilon=0.1: 0.2030121558572117  
# epsilon= 1: 0.32034490712891406

#C  
# C = 1: 0.2030121558572117  
# C = 30: 0.44375371046648393  
# C = 60: 0.47743627688241197  
# C = 90: 0.48626988142162164  
# C = 100: 0.49289490821769516

# Feature importance

# Importing libraries/modules  
from sklearn.inspection import permutation\_importance  
import numpy as np  
import matplotlib.pyplot as plt  
%matplotlib inline

# Creating and fitting the model  
from sklearn.svm import SVR  
svr2 = SVR(kernel = 'rbf', C = 90, epsilon = 0.01).fit(X, y2)

# Getting the importance  
results = permutation\_importance(svr2, X, y2)

importance = results.importances\_mean

# Summarizing the importance  
for i,y2 in enumerate(importance):  
 print('X:%0d, Score: %.5f'%(i,y2))

X:0, Score: 0.00000  
X:1, Score: 0.00000  
X:2, Score: 0.04950  
X:3, Score: 0.03778  
X:4, Score: 0.95848  
X:5, Score: 0.10312

# Barplot

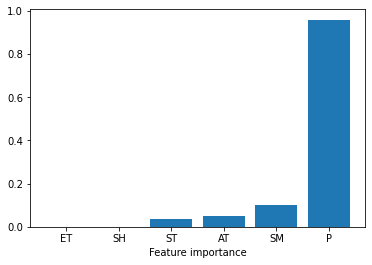
# Setting the features for and sorting them by index  
features = ['ET','SH','AT','ST','P','SM']

features = np.array(features)

sorted\_index=results.importances\_mean.argsort()  
sorted\_index

array([0, 1, 3, 2, 5, 4], dtype=int64)

# Plotting the permutation values in horizontal bar  
plt.bar(features[sorted\_index], results.importances\_mean[sorted\_index])  
plt.xlabel('Feature importance')  
plt.show()



########################################################################

# DECISION TREE

from sklearn.tree import DecisionTreeRegressor

# Remote sensing lake level data as output feature

# Creating the dt regressor and fitting the model  
  
lake\_dt1 = DecisionTreeRegressor(max\_features = 1).fit(X, y1)

#print(lake\_dt1)

# The cofficient of determination  
print('The dt1 coefficient of determination on the dataset is:', lake\_dt1.score(X, y1))

The dt1 coefficient of determination on the dataset is: 1.0

# max\_depth = None: 1.0  
# max\_depth = 3: 0.48329226121195845  
# max\_depth = 6: 0.8329611267451625  
# max\_depth = 9: 0.9798222404602556  
# max\_depth = 12: 0.9996584551373113  
# max\_depth = 15: 0.9999959604392349

# min\_samples\_leaf=1 (default): 1.0  
# min\_samples\_leaf= 2: 0.9430092034659435  
# min\_samples\_leaf=3: 0.904714234738876   
# min\_samples\_leaf=4: 0.8570508080564225  
# min\_samples\_leaf=5: 0.7924548234439408  
# min\_samples\_leaf=10: 0.6295195919991818  
# min\_samples\_leaf=15: 0.5305476912323674  
# min\_samples\_leaf=20: 0.47096276733540254

# min\_samples\_split = 2 (default): 1.0  
# min\_samples\_split = 4: 0.9647063575953847  
# min\_samples\_split = 6: 0.9243866263608275  
# min\_samples\_split = 8: 0.897559682105805  
# min\_samples\_split = 10: 0.8432417074389789

# max\_leaf\_nodes = None: 1.0  
# max\_leaf\_nodes = 50: 0.9209603521862545  
# max\_leaf\_nodes = 100: 0.9844616275390322  
# max\_leaf\_nodes = 150: 0.9979140718099252

# Model evaluation  
from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error, explained\_variance\_score

# Predicting on X\_train  
  
y1\_preddt1 = lake\_dt1.predict(X)

# dt1 MSE  
print('The dt1 MSE is: %.5f'% mean\_squared\_error(y1, y1\_preddt1))

The dt1 MSE is: 0.00043

# max\_depth = None: 0.00   
# max\_depth = 3: 0.10659  
# max\_depth = 6: 0.03446  
# max\_depth = 9: 0.00416  
# max\_depth = 12: 0.00007  
# max\_depth = 15: 0.00000

# min\_samples\_leaf=1: 0.00  
# min\_samples\_leaf=2: 0.01176  
# min\_samples\_leaf=3: 0.01966  
# min\_samples\_leaf=4: 0.02949  
# min\_samples\_leaf= 5: 0.04282  
# min\_samples\_leaf=10: 0.07643  
# min\_samples\_leaf=15:  
# min\_samples\_leaf=20:

# min\_samples\_split = 2: 0.0  
# min\_samples\_split = 4: 0.00728  
# min\_samples\_split = 6: 0.01560  
# min\_samples\_split = 8: 0.02113  
# min\_samples\_split = 10: 0.03234

# max\_leaf\_nodes = None: 0.00  
# max\_leaf\_nodes = 50: 0.01631  
# max\_leaf\_nodes = 100: 0.00321  
# max\_leaf\_nodes = 150: 0.00043

# dt1 RMSE  
print('The dt1 RMSE is: %.5f'% np.sqrt(mean\_squared\_error(y1, y1\_preddt1)))

The dt1 RMSE is: 0.02074

# max\_depth = None: 0.00   
# max\_depth = 3 : 0.32649  
# max\_depth = 6: 0.18563  
# max\_depth = 9: 0.06452  
# max\_depth = 12: 0.00839  
# max\_depth = 15: 0.00091

# min\_samples\_leaf=1: 0.00  
## min\_samples\_leaf=2: 0.10843  
# min\_samples\_leaf=3: 0.14020  
# min\_samples\_leaf=4: 0.17172  
# min\_samples\_leaf= 5: 0.20692  
# min\_samples\_leaf= 10: 0.27646

# min\_samples\_split = 2: 0.00  
# min\_samples\_split = 4: 0.08533  
# min\_samples\_split = 6: 0.12489  
# min\_samples\_split = 8: 0.14537  
# min\_samples\_split = 10: 0.17983

# max\_leaf\_nodes = None: 0.00  
# max\_leaf\_nodes = 50: 0.12769  
# max\_leaf\_nodes = 100: 0.05662  
# max\_leaf\_nodes = 150: 0.02074

#dt1 MAE  
print('dt1 MAE is:', mean\_absolute\_error(y1, y1\_preddt1))

dt1 MAE is: 0.0136624999999988

# max\_depth = None: 0.0   
# max\_depth = 3: 0.25234331251268977  
# max\_depth = 6: 0.13465806253477883  
# max\_depth = 9: 0.03332963980464072  
# max\_depth = 12: 0.0017583333333322552  
# max\_depth = 15: 0.00008333333333325754

# min\_samples\_leaf=1: 0.0  
# min\_samples\_leaf=2: 0.07366666666666623  
# min\_samples\_leaf=3: 0.10373194444444328  
# min\_samples\_leaf=4: 0.1277535714285712  
# min\_samples\_leaf= 5: 0.1529363425925922  
# min\_samples\_leaf= 10: 0.2108198329448321

# min\_samples\_split = 2: 0.0  
# min\_samples\_split = 4: 0.04736111111111043  
# min\_samples\_split = 6: 0.08455138888889024  
# min\_samples\_split = 8: 0.10545833333333358  
# min\_samples\_split = 10: 0.12916851851851827

# max\_leaf\_nodes = None: 0.00  
# max\_leaf\_nodes = 50: 0.10229958228046337  
# max\_leaf\_nodes = 100: 0.04296111111111145  
# max\_leaf\_nodes = 150: 0.0136624999999988

#evs1  
print('dt1 EVS is:', explained\_variance\_score(y1, y1\_preddt1))

dt1 EVS is: 0.9844616275390322

# max\_depth = None: 1.0   
# max\_depth = 3 : 0.48329226121195845  
# max\_depth = 6: 0.8329611267451626  
# max\_depth = 9: 0.9798222404602556  
# max\_depth = 12: 0.9996584551373113  
# max\_depth = 15: 0.9999959604392349

# min\_samples\_leaf=1: 1.0  
# min\_samples\_leaf=2: 0.9430092034659435  
# min\_samples\_leaf=3: 0.904714234738876  
# min\_samples\_leaf=4: 0.8570508080564225  
# min\_samples\_leaf= 5: 0.7924548234439408  
# min\_samples\_leaf= 10: 0.6295195919991818

# min\_samples\_split = 2: 1.0  
# min\_samples\_split = 4: 0.9647063575953847  
# min\_samples\_split = 6: 0.9243866263608275  
# min\_samples\_split = 8: 0.8975596821058052  
# min\_samples\_split = 10: 0.8432417074389789

# max\_leaf\_nodes = None: 1.0  
# max\_leaf\_nodes = 50: 0.9209603521862545  
# max\_leaf\_nodes = 100: 0.9844616275390322  
# max\_leaf\_nodes = 150: 0.9999298484375494

# Creating the dt regressor and fitting the training model  
  
lake\_dt1 = DecisionTreeRegressor(max\_depth=9, min\_samples\_leaf = 2, min\_samples\_split = 4, max\_leaf\_nodes = 100, max\_features=2).fit(X\_train, y1\_train)

print(lake\_dt1)

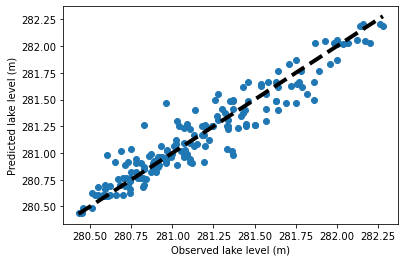
DecisionTreeRegressor(max\_depth=9, max\_features=2, max\_leaf\_nodes=100,  
 min\_samples\_leaf=2, min\_samples\_split=4)

# The cofficient of determination  
print('The dt1 coefficient of determination on the training dataset is:', lake\_dt1.score(X\_train, y1\_train))

The dt1 coefficient of determination on the training dataset is: 0.8761305714576609

# Predicting on X\_train  
  
ytrain\_preddt1 = lake\_dt1.predict(X\_train)

# Plotting the scatter plot of the correlation test  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_train, ytrain\_preddt1)  
ax.plot([y1\_train.min(),y1\_train.max()], [y1\_train.min(), y1\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtraindt1 = cov(y1\_train, ytrain\_preddt1)  
print(covtraindt1)

[[0.21501729 0.19527339]  
 [0.19527339 0.19527339]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortraindt1 = pearsonr(y1\_train, ytrain\_preddt1)  
print(cortraindt1)

(0.9529823031297124, 3.163744181630932e-94)

# dt1 MSE  
print('The dt1 MSE is: %.5f'% mean\_squared\_error(y1\_train, ytrain\_preddt1))

The dt1 MSE is: 0.01998

# dt1 RMSE  
print('The dt1 RMSE is: %.5f'% np.sqrt(mean\_squared\_error(y1\_train, ytrain\_preddt1)))

The dt1 RMSE is: 0.14134

#dt1 MAE  
print('dt1 MAE is:', mean\_absolute\_error(y1\_train, ytrain\_preddt1))

dt1 MAE is: 0.09952083333333203

#evs1  
print('dt1 EVS is:', explained\_variance\_score(y1\_train, ytrain\_preddt1))

dt1 EVS is: 0.9065745380342781

# To save the predicted valuies  
numpy.savetxt('E:/Lake Level/DT/dtLLRtrainbest.csv', ytrain\_preddt1, delimiter = ',')

# Prediction  
pred\_dt1 = lake\_dt1.predict(X\_test)

print(pred\_dt1[:5])

[281.63 281.23 281.37666667 281.27 281.21333333]

# To save the predicted data on the drive  
numpy.savetxt('E:/Lake Level/DT/dtLLRtestbest.csv', pred\_dt1, delimiter = ',')

# The prediction r\_sq  
from sklearn.metrics import r2\_score  
# The Prediction coefficient of determination  
print('The dt1 coefficient of determination of the prediction is:',r2\_score(y1\_test, pred\_dt1) )

The dt1 coefficient of determination of the prediction is: -0.033927919647474214

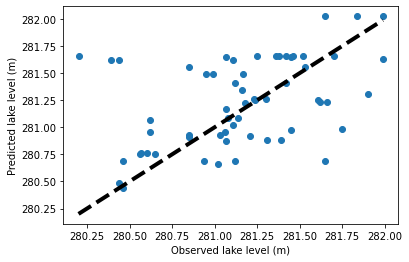
# max\_depth = None: 0.13403641218991158  
# max\_depth = 3: 0.08579018479854539  
# max\_depth = 6: 0.3271610140637162  
# max\_depth = 9: 0.3063342564560262  
# max\_depth = 12: 0.19127502150252407  
# max\_depth = 15: 0.2612623358217169

# min\_samples\_leaf=1: 0.13403641218991158  
# min\_samples\_leaf=2: 0.2671242073610123 /   
# min\_samples\_leaf=3: 0.13871012540627115 /   
# min\_samples\_leaf=4: 0.23355211159758982  
# min\_samples\_leaf= 5: 0.12341790859312107  
# min\_samples\_leaf= 10: 0.07125814831653787

# min\_samples\_split = 2: 0.13403641218991158  
# min\_samples\_split = 4: 0.2507110178933274  
# min\_samples\_split = 6: 0.3790980669576901  
# min\_samples\_split = 8: 0.2813226242006639  
# min\_samples\_split = 10: 0.33851143031508846

# max\_leaf\_nodes = None: 0.13403641218991158  
# max\_leaf\_nodes = 50: 0.3587007947284808  
# max\_leaf\_nodes = 100: 0.24405407584757288  
# max\_leaf\_nodes = 150: 0.18027729735971587

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_dt1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



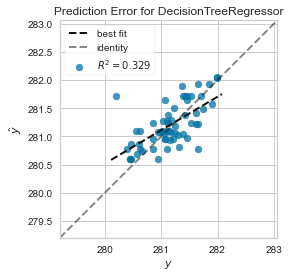
# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtestdt1 = cov(y1\_test, pred\_dt1)  
print(covtestdt1)

[[0.1852037 0.07735123]  
 [0.07735123 0.16206226]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestdt1 = pearsonr(y1\_test, pred\_dt1)  
print(cortestdt1)

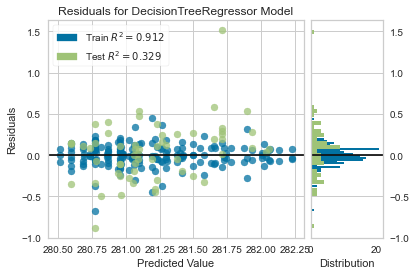
(0.44647939571686013, 0.00034919429763929656)

# Plotting the prediction errors  
from yellowbrick.regressor import PredictionError  
visualizer = PredictionError(lake\_dt1)  
visualizer.fit(X\_train, y1\_train)  
visualizer.score(X\_test, y1\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Prediction Error for DecisionTreeRegressor'}, xlabel='$y$', ylabel='$\\hat{y}$'>

# Plotting the residuals  
from yellowbrick.regressor import ResidualsPlot  
visualizer = ResidualsPlot(lake\_dt1)  
visualizer.fit(X\_train, y1\_train)  
visualizer.score(X\_test, y1\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Residuals for DecisionTreeRegressor Model'}, xlabel='Predicted Value', ylabel='Residuals'>

# Model evaluation

# The MSE   
print('The RS Data dt1 MSE is: %.2f'% mean\_squared\_error(y1\_test, pred\_dt1) )

The RS Data dt1 MSE is: 0.12

# max\_depth = None: 0.16   
# max\_depth = 3: 0.17  
# max\_depth = 6: 0.12  
# max\_depth = 9: 0.13  
# max\_depth = 12: 0.15  
# max\_depth = 15: 0.13

# min\_samples\_leaf=1: 0.16  
# min\_samples\_leaf=2: 0.13  
# min\_samples\_leaf=3: 0.16   
# min\_samples\_leaf=4: 0.14  
# min\_samples\_leaf= 5: 0.16  
# min\_samples\_leaf= 5: 0.17

# min\_samples\_split = 2: 0.16  
# min\_samples\_split = 4: 0.14  
# min\_samples\_split = 6: 0.11  
# min\_samples\_split = 8: 0.13  
# min\_samples\_split = 10: 0.12

# max\_leaf\_nodes = None: 0.16  
# max\_leaf\_nodes = 50: 0.12  
# max\_leaf\_nodes = 100: 0.14  
# max\_leaf\_nodes = 150: 0.15

# The RMSE  
   
print('The RS Data dt1 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y1\_test, pred\_dt1)))

The RS Data dt1 RMSE is: 0.35

# max\_depth = None: 0.40   
# max\_depth = 3: 0.41  
# max\_depth = 6: 0.35  
# max\_depth = 9: 0.36  
# max\_depth = 12: 0.38  
# max\_depth = 15: 0.37

# min\_samples\_leaf=1: 0.40  
# min\_samples\_leaf=2: 0.37  
# min\_samples\_leaf=3: 0.40  
# min\_samples\_leaf=4: 0.37  
# min\_samples\_leaf= 5:0.40  
# min\_samples\_leaf= 10: 0.41

# min\_samples\_split = 2: 0.40  
# min\_samples\_split = 4: 0.37  
# min\_samples\_split = 6: 0.34  
# min\_samples\_split = 8: 0.36  
# min\_samples\_split = 10: 0.35

# max\_leaf\_nodes = None: 0.40  
# max\_leaf\_nodes = 50: 0.34  
# max\_leaf\_nodes = 100: 0.37  
# max\_leaf\_nodes = 150: 0.39

# The MAE  
  
print('The RS Data dt1 MAE is: %.2f'% mean\_absolute\_error(y1\_test, pred\_dt1))

The RS Data dt1 MAE is: 0.26

# max\_depth = None: 0.32   
# max\_depth = 3: 0.33  
# max\_depth = 6: 0.26  
# max\_depth = 9: 0.29  
# max\_depth = 12: 0.31  
# max\_depth = 15: 0.29

# min\_samples\_leaf=1: 0.32  
# min\_samples\_leaf=2: 0.27  
# min\_samples\_leaf=3: 0.29   
# min\_samples\_leaf=4: 0.27  
# min\_samples\_leaf= 5: 0.30  
# min\_samples\_leaf= 10: 0.31

# min\_samples\_split = 2: 0.32  
# min\_samples\_split = 4: 0.30  
# min\_samples\_split = 6: 0.27  
# min\_samples\_split = 8: 0.28  
# min\_samples\_split = 10: 0.27

# max\_leaf\_nodes = None: 0.32  
# max\_leaf\_nodes = 50: 0.27  
# max\_leaf\_nodes = 100: 0.29  
# max\_leaf\_nodes = 150: 0.30

# The dt1 Explained variance score  
from sklearn.metrics import explained\_variance\_score  
# y1 EVS  
print('dt1 EVS is:', explained\_variance\_score(y1\_test, pred\_dt1))

dt1 EVS is: 0.3422975218121542

# max\_depth = None: 0.13   
# max\_depth = 3: 0.10728513387855687  
# max\_depth = 6: 0.33625785231320515  
# max\_depth = 9: 0.30687497552455234  
# max\_depth = 12: 0.19241363037280024  
# max\_depth = 15: 0.2613117546094891

# min\_samples\_leaf=1: 0.13  
# min\_samples\_leaf=2: 0.2690432061716208/  
# min\_samples\_leaf=3: 0.1626308320884975   
# min\_samples\_leaf=4: 0.24665279768668968  
# min\_samples\_leaf= 5: 0.16773278552017645  
# min\_samples\_leaf= 10: 0.07689600883978398

# min\_samples\_split = 2: 0.13  
# min\_samples\_split = 4: 0.2507110178933275  
# min\_samples\_split = 6: 0.3822484354064424  
# min\_samples\_split = 8: 0.2884021094065531  
# min\_samples\_split = 10: 0.34590893094032205

# max\_leaf\_nodes = None: 0.13  
# max\_leaf\_nodes = 50: 0.3684345099315941  
# max\_leaf\_nodes = 100: 0.24751880778056312  
# max\_leaf\_nodes = 150: 0.18175715363645906

# The k-fold cross-validation  
from sklearn.model\_selection import cross\_val\_score  
from numpy import absolute

# On the dataset  
score = cross\_val\_score(lake\_dt1, X, y1, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score

array([-0.23772685, -0.17080567, -0.21072234, -0.30362039, -0.12479167,  
 -0.0922281 , -0.26928021, -0.18811878, -0.18405729, -0.11760891])

# The absolute mean score on the dataset  
print(absolute(np.mean(score)))

0.18989602014125018

# max\_depth = None: 0.20281166666666603   
# max\_depth = 3: 0.1750549778339912  
# max\_depth = 6: 0.17528048814939132  
# max\_depth = 9: 0.1898180905048286  
# max\_depth = 12: 0.1910865173044198  
# max\_depth = 15: 0.1914647916666657

# min\_samples\_leaf=1: 0.18628791666666591  
# min\_samples\_leaf=2: 0.1702941435185178  
# min\_samples\_leaf=3: 0.17237226157407384  
# min\_samples\_leaf=4: 0.15610009805838856  
# min\_samples\_leaf= 5: 0.15276173465300089  
# min\_samples\_leaf= 10: 0.1624101633015565

# min\_samples\_split = 2: 0.19142791666666642  
# min\_samples\_split = 4: 0.19466947916666397  
# min\_samples\_split = 6: 0.18843273923611034  
# min\_samples\_split = 8: 0.1869927737929873  
# min\_samples\_split = 10: 0.1721469145519048

# max\_leaf\_nodes = None: 0.1895029166666655  
# max\_leaf\_nodes = 50: 0.18176951021067553  
# max\_leaf\_nodes = 100: 0.1959876133351147  
# max\_leaf\_nodes = 150: 0.18989602014125018

# On the training dataset  
score\_train = cross\_val\_score(lake\_dt1, X\_train, y1\_train, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score\_train

array([-0.20790836, -0.09935316, -0.09396572, -0.21436117, -0.17398012,  
 -0.15841952, -0.3258976 , -0.26474783, -0.08286806, -0.10906315])

# The absolute mean score on the training dataset  
print(absolute(np.mean(score\_train)))

0.17305646925208468

# On the testing dataset  
score\_test = cross\_val\_score(lake\_dt1, X\_test, y1\_test, scoring ='neg\_mean\_squared\_error', cv = 10)  
score\_test

array([-0.10508009, -0.09105217, -0.04203009, -0.17174676, -0.37640833,  
 -0.3838375 , -0.11411898, -0.25979074, -0.03420093, -0.34465833])

# The absolute mean score on the testing dataset  
mean\_score\_test = absolute(np.mean(score\_test))  
mean\_score\_test

0.19229239259258665

# max\_depth = None: 0.24   
# max\_depth = 3: 0.22124618701610355  
# max\_depth = 6: 0.24987545168352696  
# max\_depth = 9: 0.2509700810279606  
# max\_depth = 12: 0.23818166666666074  
# max\_depth = 15: 0.24319833333332985

# min\_samples\_leaf=1: 0.24  
# min\_samples\_leaf=2: 0.1827279629629563 / 0.19674143518517825  
# min\_samples\_leaf=3: 0.17950382962962275  
# min\_samples\_leaf=4: 0.1461044662414917  
# min\_samples\_leaf= 5: 0.15872615489995118  
# min\_samples\_leaf= 10: 0.17100676982977359

# min\_samples\_split = 2: 0.24  
# min\_samples\_split = 4: 0.254532268518512  
# min\_samples\_split = 6: 0.21591862499999617  
# min\_samples\_split = 8: 0.19863594377361823  
# min\_samples\_split = 10: 0.20319987047954421

# max\_leaf\_nodes = None: 0.24  
# max\_leaf\_nodes = 50: 0.2503391666666615  
# max\_leaf\_nodes = 100: 0.23180999999999594  
# max\_leaf\_nodes = 150: 0.25268499999999566

# Feature Importance  
from sklearn.tree import DecisionTreeRegressor  
#import numpy as np

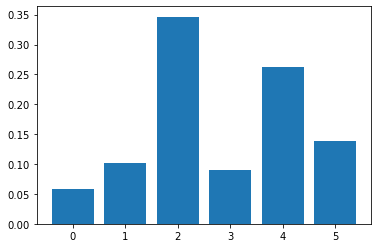
# Setting and fitting the model  
dt1 = DecisionTreeRegressor(max\_depth = 9, min\_samples\_leaf = 2, min\_samples\_split = 4, max\_leaf\_nodes = 100, max\_features = 2).fit(X, y1)

# Getting the feature importance  
f\_importance = dt1.feature\_importances\_

# Summary of the importances  
for i, y1 in enumerate(f\_importance):  
 print('X:%0d, Score:%.5f'%(i,y1))

X:0, Score:0.05906  
X:1, Score:0.10240  
X:2, Score:0.34651  
X:3, Score:0.09021  
X:4, Score:0.26250  
X:5, Score:0.13931

# PLotting the importances  
from matplotlib import pyplot  
pyplot.bar([X for X in range(len(f\_importance))], f\_importance)  
pyplot.show()



###########################################################################

# DT: Ground truth lake level data as output feature  
seed = 7

# Creating the svr regressor and fitting the model  
lake\_dt2 = DecisionTreeRegressor(max\_leaf\_nodes = 130).fit(X, y2)

print(lake\_dt2)

# The coefficient of determination  
  
print('The dt2 coefficient of determination on the model is: %.2f'% lake\_dt2.score(X, y2))

The dt2 coefficient of determination on the model is: 1.00

# max\_depth = None: 1.00   
# max\_depth = 3: 0.61  
# max\_depth = 6: 0.87  
# max\_depth = 9: 0.97  
# max\_depth = 12: 1.00  
# max\_depth = 15: 1.00

# min\_samples\_leaf=1 (default): 1.00  
# min\_samples\_leaf=2: 0.96  
# min\_samples\_leaf=3: 0.91  
# min\_samples\_leaf=4: 0.86  
# min\_samples\_leaf= 5: 0.82  
# min\_samples\_leaf= 10: 0.71

# min\_samples\_split = 2 (default): 1.00  
# min\_samples\_split = 4: 0.98  
# min\_samples\_split = 6: 0.94  
# min\_samples\_split = 8: 0.92  
# min\_samples\_split = 10: 0.90

# max\_leaf\_nodes = None: 1.00  
# max\_leaf\_nodes = 50: 0.92  
# max\_leaf\_nodes = 70: 0.96  
# max\_leaf\_nodes = 90: 0.98  
# max\_leaf\_nodes = 110: 0.99  
# max\_leaf\_nodes = 130: 1.00

# Training Model evaluation

# Predicting on X\_train  
  
y2\_preddt2 = lake\_dt2.predict(X)

# dt2 MSE  
print('The dt2 MSE is: %.4f'% mean\_squared\_error(y2, y2\_preddt2))

The dt2 MSE is: 0.0020

# max\_depth = None: 0.00   
# max\_depth = 3: 0.1859  
# max\_depth = 6: 0.0626  
# max\_depth = 9: 0.0120  
# max\_depth = 12: 0.0003  
# max\_depth = 15: 0.0000

# min\_samples\_leaf=1: 0.00  
# min\_samples\_leaf=2: 0.0180  
# min\_samples\_leaf=3: 0.0416  
# min\_samples\_leaf=4: 0.0655  
# min\_samples\_leaf= 5: 0.0837  
# min\_samples\_leaf= 10: 0.1397

# min\_samples\_split = 2: 0.00  
# min\_samples\_split = 4: 0.0082  
# min\_samples\_split = 6: 0.0265  
# min\_samples\_split = 8: 0.0390  
# min\_samples\_split = 10: 0.0472

# max\_leaf\_nodes = None: 0.00  
# max\_leaf\_nodes = 50: 0.0387  
# max\_leaf\_nodes = 70: 0.0196  
# max\_leaf\_nodes = 90: 0.0094  
# max\_leaf\_nodes = 110: 0.0044  
# max\_leaf\_nodes = 130: 0.0020

# dt2 RMSE  
print('The dt2 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y2, y2\_preddt2)))

The dt2 RMSE is: 0.05

# max\_depth = None: 0.00   
# max\_depth = 3: 0.43  
# max\_depth = 6: 0.25  
# max\_depth = 9: 0.11  
# max\_depth = 12: 0.02  
# max\_depth = 15: 0.00

# min\_samples\_leaf=1: 0.00  
# min\_samples\_leaf=2: 0.13  
# min\_samples\_leaf=3: 0.20  
# min\_samples\_leaf=4: 0.26  
# min\_samples\_leaf= 5: 0.29  
# min\_samples\_leaf= 10: 0.37

# min\_samples\_split = 2: 0.00  
# min\_samples\_split = 4: 0.09  
# min\_samples\_split = 6: 0.16  
# min\_samples\_split = 8: 0.20  
# min\_samples\_split = 10: 0.22

# max\_leaf\_nodes = None: 0.00  
# max\_leaf\_nodes = 50: 0.20  
# max\_leaf\_nodes = 70: 0.14  
# max\_leaf\_nodes = 90: 0.10  
# max\_leaf\_nodes = 110: 0.07  
# max\_leaf\_nodes = 130: 0.05

#dt2 MAE  
print('dt2 MAE is:', mean\_absolute\_error(y2, y2\_preddt2))

dt2 MAE is: 0.03238055555555472

# max\_depth = None: 0.0   
# max\_depth = 3: 0.34664381498156255  
# max\_depth = 6: 0.18093073346839314  
# max\_depth = 9: 0.053377861952862087  
# max\_depth = 12: 0.004441666666665848  
# max\_depth = 15: 0.0

# min\_samples\_leaf=1: 0.0  
# min\_samples\_leaf=2: 0.09613888888888837  
# min\_samples\_leaf=3: 0.1566458333333325  
# min\_samples\_leaf=4: 0.1965244047619059  
# min\_samples\_leaf= 5: 0.22429179894179943  
# min\_samples\_leaf= 10: 0.2979612705148315

# min\_samples\_split = 2: 0.0  
# min\_samples\_split = 4: 0.0577777777777752  
# min\_samples\_split = 6: 0.1083638888888854  
# min\_samples\_split = 8: 0.14190793650793576  
# min\_samples\_split = 10: 0.16343670634920643

# max\_leaf\_nodes = None: 0.0  
# max\_leaf\_nodes = 50: 0.1549091570466558  
# max\_leaf\_nodes = 70: 0.11035980639730643  
# max\_leaf\_nodes = 90: 0.07470462962962851  
# max\_leaf\_nodes = 110: 0.050599206349206345  
# max\_leaf\_nodes = 130: 0.03238055555555472

#dt2 EVS  
print('dt2 EVS is:', explained\_variance\_score(y2, y2\_preddt2))

dt2 EVS is: 0.9957390471729627

# max\_depth = None: 1.0   
# max\_depth = 3: 0.6091711411900074  
# max\_depth = 6: 0.8684719065077946  
# max\_depth = 9: 0.9747839412833526  
# max\_depth = 12: 0.9993409702543786  
# max\_depth = 15: 1.0

# min\_samples\_leaf=1: 1.0  
# min\_samples\_leaf=2: 0.9620575709650213  
# min\_samples\_leaf=3: 0.9125972695607885  
# min\_samples\_leaf=4: 0.8622019234652044  
# min\_samples\_leaf= 5: 0.8241055486047175  
# min\_samples\_leaf= 10: 0.7063479042206806

# min\_samples\_split = 2: 1.0  
# min\_samples\_split = 4: 0.9827411502512486  
# min\_samples\_split = 6: 0.9442250760986117  
# min\_samples\_split = 8: 0.9180394987753806  
# min\_samples\_split = 10: 0.881917997020645

# max\_leaf\_nodes = None: 1.0  
# max\_leaf\_nodes = 50: 0.9187091113093037  
# max\_leaf\_nodes = 70: 0.9588728025218467  
# max\_leaf\_nodes = 90: 0.9801421850677748  
# max\_leaf\_nodes = 110: 0.9907013436940691  
# max\_leaf\_nodes = 130: 0.9957390471729627

# Creating the dt regressor and fitting the training model  
lake\_dt2 = DecisionTreeRegressor(max\_depth = 9, min\_samples\_leaf= 2, min\_samples\_split = 4, max\_leaf\_nodes = 100, max\_features = 2).fit(X\_train, y2\_train)

print(lake\_dt2)

DecisionTreeRegressor(max\_depth=9, max\_features=2, max\_leaf\_nodes=100,  
 min\_samples\_leaf=2, min\_samples\_split=4)

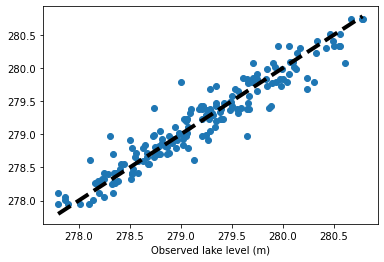
# The coefficient of determination  
  
print('The dt2 coefficient of determination on the training model is: %.2f'% lake\_dt2.score(X\_train, y2\_train))

The dt2 coefficient of determination on the training model is: 0.90

# Predicting on X\_train  
  
ytrain\_preddt2 = lake\_dt2.predict(X\_train)

# Plotting the scatter plot of the correlation test  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_train, ytrain\_preddt2)  
ax.plot([y2\_train.min(),y2\_train.max()], [y2\_train.min(), y2\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')

Text(0.5, 0, 'Observed lake level (m)')



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtraindt2 = cov(y2\_train, ytrain\_preddt2)  
print(covtraindt2)

[[0.49448696 0.44903646]  
 [0.44903646 0.44903646]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortraindt2 = pearsonr(y2\_train, ytrain\_preddt2)  
print(cortraindt2)

(0.9529352226075093, 3.4512406697794e-94)

# dt2 MSE  
print('The dt2 MSE is: %.4f'% mean\_squared\_error(y2\_train, ytrain\_preddt2))

The dt2 MSE is: 0.0469

# dt2 RMSE  
print('The dt2 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y2\_train, ytrain\_preddt2)))

The dt2 RMSE is: 0.22

#dt2 MAE  
print('dt2 MAE is:', mean\_absolute\_error(y2\_train, ytrain\_preddt2))

dt2 MAE is: 0.16825363108206343

#dt2 EVS  
print('dt2 EVS is:', explained\_variance\_score(y2\_train, ytrain\_preddt2))

dt2 EVS is: 0.9045945781996795

# Saving the predicted data  
numpy.savetxt("E:/Lake Level/DT/dtLLGtrainbest.csv", ytrain\_preddt2, delimiter = ',')

# Prediction on testing data  
pred\_dt2 = lake\_dt2.predict(X\_test)

print(pred\_dt2[:5])

[280.07 279.66333333 279.76666667 279.275 279.79 ]

# To save the predicted data on the drive  
numpy.savetxt('E:/Lake Level/DT/dtLLGtestbest.csv', pred\_dt2, delimiter = ',')

# The Prediction coefficient of determination  
print('The dt2 coefficient of determination of the prediction is: %.2f'% r2\_score(y2\_test, pred\_dt2))

The dt2 coefficient of determination of the prediction is: 0.38

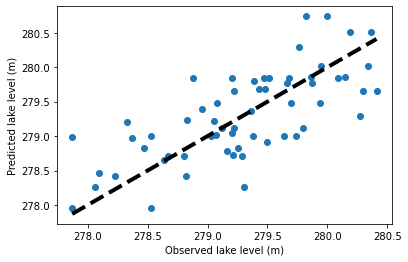
# max\_depth = None: 0.26   
# max\_depth = 3: 0.47  
# max\_depth = 6: 0.46  
# max\_depth = 9: 0.32  
# max\_depth = 12: 0.19  
# max\_depth = 15: 0.18

# min\_samples\_leaf=1: 0.26  
# min\_samples\_leaf=2: 0.27  
# min\_samples\_leaf=3: 0.34  
# min\_samples\_leaf=4: 0.35  
# min\_samples\_leaf= 5: 0.42  
# min\_samples\_leaf= 10: 0.53

# min\_samples\_split = 2: 0.26  
# min\_samples\_split = 4: 0.18  
# min\_samples\_split = 6: 0.30  
# min\_samples\_split = 8: 0.31  
# min\_samples\_split = 10: 0.33

# max\_leaf\_nodes = None: 0.26  
# max\_leaf\_nodes = 50: 0.31  
# max\_leaf\_nodes = 70: 0.25  
# max\_leaf\_nodes = 90: 0.22

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_test, pred\_dt2)  
ax.plot([y2\_test.min(),y2\_test.max()], [y2\_test.min(), y2\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtestdt2 = cov(y2\_test, pred\_dt2)  
print(covtestdt2)

[[0.42726607 0.30099354]  
 [0.30099354 0.41649476]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestdt2 = pearsonr(y2\_test, pred\_dt2)  
print(cortestdt2)

(0.7135151469805541, 1.5736998008404213e-10)

# Plotting the prediction errors  
from yellowbrick.regressor import PredictionError  
visualizer = PredictionError(lake\_dt2)  
visualizer.fit(X\_train, y2\_train)  
visualizer.score(X\_test, y2\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Prediction Error for DecisionTreeRegressor'}, xlabel='$y$', ylabel='$\\hat{y}$'>

# Plotting the residuals  
from yellowbrick.regressor import ResidualsPlot  
visualizer = ResidualsPlot(lake\_dt2)  
visualizer.fit(X\_train, y2\_train)  
visualizer.score(X\_test, y2\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Residuals for DecisionTreeRegressor Model'}, xlabel='Predicted Value', ylabel='Residuals'>

# The MSE  
   
print('The G Data dt2 MSE is: %.2f'% mean\_squared\_error(y2\_test, pred\_dt2))

The G Data dt2 MSE is: 0.26

# max\_depth = None: 0.31  
# max\_depth = 3: 0.22  
# max\_depth = 6: 0.23  
# max\_depth = 9: 0.28  
# max\_depth = 12: 0.34  
# max\_depth = 15: 0.34

# min\_samples\_leaf=1: 0.31  
# min\_samples\_leaf=2: 0.30  
# min\_samples\_leaf=3: 0.28  
# min\_samples\_leaf=4: 0.27  
# min\_samples\_leaf= 5: 0.25  
# min\_samples\_leaf= 5: 0.20

# min\_samples\_split = 2: 0.31  
# min\_samples\_split = 4: 0.34  
# min\_samples\_split = 6: 0.29  
# min\_samples\_split = 8: 0.29  
# min\_samples\_split = 10: 0.28

# max\_leaf\_nodes = None: 0.31  
# max\_leaf\_nodes = 50: 0.29  
# max\_leaf\_nodes = 70: 0.32  
# max\_leaf\_nodes = 90: 0.33

# The RMSE  
print('The G Data dt2 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y2\_test, pred\_dt2)))

The G Data dt2 RMSE is: 0.51

# max\_depth = None: 0.56   
# max\_depth = 3: 0.47  
# max\_depth = 6: 0.48  
# max\_depth = 9: 0.53  
# max\_depth = 12: 0.58  
# max\_depth = 15: 0.59

# min\_samples\_leaf=1: 0.56  
# min\_samples\_leaf=2: 0.55  
# min\_samples\_leaf=3: 0.53  
# min\_samples\_leaf=4: 0.52  
# min\_samples\_leaf= 5: 0.50  
# min\_samples\_leaf= 10: 0.44

# min\_samples\_split = 2: 0.56  
# min\_samples\_split = 4: 0.59  
# min\_samples\_split = 6: 0.54  
# min\_samples\_split = 8: 0.54  
# min\_samples\_split = 10: 0.53

# max\_leaf\_nodes = None: 0.56  
# max\_leaf\_nodes = 50: 0.54  
# max\_leaf\_nodes = 70: 0.56  
# max\_leaf\_nodes = 90: 0.57

# The MAE  
  
print('The G Data dt2 MAE is: %.2f'% mean\_absolute\_error(y2\_test, pred\_dt2))

The G Data dt2 MAE is: 0.41

# max\_depth = None: 0.46   
# max\_depth = 3: 0.38  
# max\_depth = 6: 0.40  
# max\_depth = 9: 0.44  
# max\_depth = 12: 0.49  
# max\_depth = 15: 0.49

# min\_samples\_leaf=1: 0.46  
# min\_samples\_leaf=2: 0.45  
# min\_samples\_leaf=3: 0.42  
# min\_samples\_leaf=4: 0.42  
# min\_samples\_leaf= 5: 0.40  
# min\_samples\_leaf= 10: 0.37

# min\_samples\_split = 2: 0.46  
# min\_samples\_split = 4: 0.48  
# min\_samples\_split = 6: 0.44  
# min\_samples\_split = 8: 0.43  
# min\_samples\_split = 10: 0.42

# max\_leaf\_nodes = None: 0.46  
# max\_leaf\_nodes = 50: 0.43  
# max\_leaf\_nodes = 70: 0.46  
# max\_leaf\_nodes = 90: 0.47

# The dt1 Explained variance score  
# y2 EVS  
print('dt2 EVS is: %.2f'% explained\_variance\_score(y2\_test, pred\_dt2))

dt2 EVS is: 0.39

# max\_depth = None: 0.27   
# max\_depth = 3: 0.48  
# max\_depth = 6: 0.48  
# max\_depth = 9: 0.33  
# max\_depth = 12: 0.21  
# max\_depth = 15: 0.19

# min\_samples\_leaf=1: 0.27  
# min\_samples\_leaf=2: 0.28  
# min\_samples\_leaf=3: 0.35  
# min\_samples\_leaf=4: 0.36  
# min\_samples\_leaf= 5: 0.43  
# min\_samples\_leaf= 10: 0.53

# min\_samples\_split = 2: 0.27  
# min\_samples\_split = 4: 0.20  
# min\_samples\_split = 6: 0.30  
# min\_samples\_split = 8: 0.32  
# min\_samples\_split = 10: 0.34

# max\_leaf\_nodes = None: 0.27  
# max\_leaf\_nodes = 50: 0.32  
# max\_leaf\_nodes = 70: 0.26  
# max\_leaf\_nodes = 90: 0.24

# The k-fold cross-validation  
seed = 7  
from sklearn.model\_selection import cross\_val\_score

# max\_depth = None: 0.4525220833333329  
# max\_depth = 3: 0.35086545797452706  
# max\_depth = 6: 0.3977228319595142  
# max\_depth = 9: 0.44379549503440624  
# max\_depth = 12: 0.46309719248327824  
# max\_depth = 15: 0.43269144166666634

# min\_samples\_leaf=1: 0.4645933333333341  
# min\_samples\_leaf=2: 0.4409124768518547  
# min\_samples\_leaf=3: 0.3832731878472202  
# min\_samples\_leaf=4: 0.3853931588624317  
# min\_samples\_leaf= 5: 0.3840010411705046  
# min\_samples\_leaf= 10: 0.3167520721146747

# min\_samples\_split = 2: 0.4437699999999989  
# min\_samples\_split = 4: 0.4390325925925936  
# min\_samples\_split = 6: 0.4205006568286981  
# min\_samples\_split = 8: 0.414960664786467  
# min\_samples\_split = 10: 0.4026734112088704

# max\_leaf\_nodes = None: 0.44257624999999995  
# max\_leaf\_nodes = 50: 0.42080640032097927  
# max\_leaf\_nodes = 70: 0.44367835188068216  
# max\_leaf\_nodes = 90: 0.4449153520161505  
# max\_leaf\_nodes = 110: 0.4556885603009263  
# max\_leaf\_nodes = 130: 0.44322269131944436

# On the model  
score = cross\_val\_score(lake\_dt2, X, y2, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score

array([-0.50430856, -0.18339688, -1.20142954, -0.54253584, -0.57520289,  
 -0.30719463, -0.28931678, -0.21593287, -0.27452338, -0.33838554])

# The absolute mean score on the model   
print(absolute(np.mean(score)))

0.44322269131944436

# On the training model  
score\_train = cross\_val\_score(lake\_dt2, X\_train, y2\_train, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score\_train

array([-0.30022012, -0.30288343, -0.50354676, -0.27063004, -0.24391382,  
 -0.3580482 , -0.45772123, -0.19409423, -0.33049784, -0.31001836])

# The absolute mean score on the training dataset  
print(absolute(np.mean(score\_train)))

0.3271574036905379

# On the testing dataset  
score\_test = cross\_val\_score(lake\_dt2, X\_test, y2\_test, scoring ='neg\_mean\_squared\_error', cv = 10)  
score\_test

array([-0.17824676, -0.24915093, -0.37554306, -0.383825 , -0.46753426,  
 -0.14919583, -0.26199769, -0.37982778, -0.26831713, -0.18370417])

# The mean score on the testing dataset  
mean\_score\_test = np.mean(score\_test)

# Positive  
mean\_score\_test = absolute(mean\_score\_test)  
mean\_score\_test

0.28973425925926677

# max\_depth = None: 0.32917500000000366  
# max\_depth = 3: 0.24825340997021633  
# max\_depth = 6: 0.3262334293438272  
# max\_depth = 9: 0.34495953703703963  
# max\_depth = 12:0.3281183333333394  
# max\_depth = 15: 0.34399833333333985

# min\_samples\_leaf=1: 0.32917500000000366  
# min\_samples\_leaf=2: 0.3523446296296348  
# min\_samples\_leaf=3: 0.29627703796296634  
# min\_samples\_leaf=4: 0.27661032328043145  
# min\_samples\_leaf= 5: 0.2614815496945132  
# min\_samples\_leaf= 10: 0.19355182656953573

# min\_samples\_split = 2: 0.32917500000000366  
# min\_samples\_split = 4: 0.3361530092592613  
# min\_samples\_split = 6: 0.29996402407407463  
# min\_samples\_split = 8: 0.27876262356387066  
# min\_samples\_split = 10: 0.271312616415767

# max\_leaf\_nodes = None: 0.32917500000000366  
# max\_leaf\_nodes = 50: 0.356204537037039  
# max\_leaf\_nodes = 70: 0.34463333333333745  
# max\_leaf\_nodes = 90: 0.3329333333333383

# Feature Importance

# Setting and fitting the model  
dt2 = DecisionTreeRegressor(max\_depth = 9, min\_samples\_leaf = 2, min\_samples\_split = 4, max\_leaf\_nodes = 100, max\_features = 2).fit(X,y2)

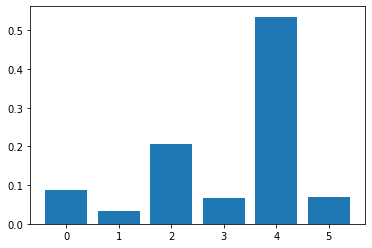
# Getting the importances  
f\_importance = dt2.feature\_importances\_

# Outputing the importance values  
for i, y2 in enumerate(f\_importance):  
 print('X:%0d, Score:%.5f'%(i,y2))

X:0, Score:0.08854  
X:1, Score:0.03417  
X:2, Score:0.20596  
X:3, Score:0.06757  
X:4, Score:0.53527  
X:5, Score:0.06848

# Plotting the importances  
from matplotlib import pyplot  
pyplot.bar([X for X in range(len(f\_importance))], f\_importance)

<BarContainer object of 6 artists>



##########################################################################

# RANDOM FOREST MODEL  
from sklearn.ensemble import RandomForestRegressor

# Remote sensing lake level data as output feature

# Instantiation  
model = RandomForestRegressor(max\_leaf\_nodes = 175)

# Fitting the model  
lake\_rf1 = model.fit(X, y1)

print(lake\_rf1)

# Training Model evaluation  
from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error, explained\_variance\_score

# The coefficient of determination  
  
print('The rf1 coefficient of determination on the model is:', lake\_rf1.score(X, y1))

The rf1 coefficient of determination on the model is: 0.9315605696453895

# n\_estimators=5: 0.8817481603167016   
# n\_estimators=50: 0.9230926745258579  
# n\_estimators=100 (default): 0.9327932440366054  
# n\_estimators=150: 0.9352877834256829  
# n\_estimators=200: 0.9363412896297804

# max\_depth = None: 0.9338223003278843  
# max\_depth = 3: 0.570997862185677  
# max\_depth = 6: 0.8463219141478038  
# max\_depth = 9: 0.9236292902446975  
# max\_depth = 12: 0.930914193545224  
# max\_depth = 15: 0.9335960395197589

# min\_samples\_leaf=1 (default): 0.9322520075270523  
# min\_samples\_leaf=2: 0.8979616542254918  
# min\_samples\_leaf=3: 0.8364840665928865  
# min\_samples\_leaf=4: 0.7973202943894222  
# min\_samples\_leaf= 5: 0.7594396326049195

# min\_samples\_split = 2 (default): 0.929286541126102  
# min\_samples\_split = 4: 0.9107974116618418  
# min\_samples\_split = 6: 0.8797244481861373  
# min\_samples\_split = 8: 0.8572522548075303  
# min\_samples\_split = 10: 0.8222805812487262  
# min\_samples\_split = 12: 0.8009258122090865

# max\_leaf\_nodes = None: 0.9319126872504707  
# max\_leaf\_nodes = 2: 0.2937312993379332  
# max\_leaf\_nodes = 25: 0.8391194680854236  
# max\_leaf\_nodes = 50: 0.9113468089876312  
# max\_leaf\_nodes = 75: 0.9282959805960277  
# max\_leaf\_nodes = 100: 0.934044178300523  
# max\_leaf\_nodes = 125: 0.9361870279627948

# Predicting on X  
  
y1\_predrf1 = lake\_rf1.predict(X)

# rf1 MSE  
print('The rf1 MSE is: %.2f'% mean\_squared\_error(y1, y1\_predrf1))

The rf1 MSE is: 0.01

# n\_estimators=5: 0.03  
# n\_estimators=50: 0.02  
# n\_estimators=100: 0.01  
# n\_estimators=150: 0.01  
# n\_estimators=200: 0.01

# max\_depth = None: 0.1  
# max\_depth = 3: 0.09  
# max\_depth = 6: 0.03  
# max\_depth = 9: 0.02  
# max\_depth = 12: 0.01  
# max\_depth = 15: 0.01

# min\_samples\_leaf=1: 0.01  
# min\_samples\_leaf=2: 0.02  
# min\_samples\_leaf=3: 0.03  
# min\_samples\_leaf=4: 0.04  
# min\_samples\_leaf= 5:0.05

# min\_samples\_split = 2: 0.01  
# min\_samples\_split = 4: 0.02  
# min\_samples\_split = 6: 0.02  
# min\_samples\_split = 8: 0.03  
# min\_samples\_split = 10: 0.04  
# min\_samples\_split = 12: 0.04

# max\_leaf\_nodes = None: 0.01  
# max\_leaf\_nodes = 2: 0.14   
# max\_leaf\_nodes = 25: 0.03   
# max\_leaf\_nodes = 50: 0.02   
# max\_leaf\_nodes = 75: 0.01   
# max\_leaf\_nodes = 100: 0.01

# rf1 RMSE  
import numpy as np  
print('The rf1 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y1, y1\_predrf1)))

The rf1 RMSE is: 0.12

# n\_estimators=5: 0.16  
# n\_estimators=50: 0.13  
# n\_estimators=100: 0.12  
# n\_estimators=150: 0.12  
# n\_estimators=200: 0.12

# max\_depth = None: 0.12  
# max\_depth = 3: 0.30  
# max\_depth = 6: 0.18  
# max\_depth = 9: 0.13  
# max\_depth = 12: 0.12  
# max\_depth = 15: 0.12

# min\_samples\_leaf=1: 0.12  
# min\_samples\_leaf=2: 0.15  
# min\_samples\_leaf=3: 0.18  
# min\_samples\_leaf=4: 0.20  
# min\_samples\_leaf= 5:0.22

# min\_samples\_split = 2: 0.12  
# min\_samples\_split = 4: 0.14  
# min\_samples\_split = 6: 0.16  
# min\_samples\_split = 8: 0.17  
# min\_samples\_split = 10: 0.19  
# min\_samples\_split = 12: 0.20

# max\_leaf\_nodes = None: 0.12  
# max\_leaf\_nodes = 2: 0.38   
# max\_leaf\_nodes = 25: 0.18  
# max\_leaf\_nodes = 50: 0.14  
# max\_leaf\_nodes = 75: 0.12  
# max\_leaf\_nodes = 100: 0.12

#rf1 MAE  
print('rf1 MAE is:', mean\_absolute\_error(y1, y1\_predrf1))

rf1 MAE is: 0.09133485879167083

# n\_estimators=5: 0.11061666666666643  
# n\_estimators=50: 0.09453916666666515  
# n\_estimators=100: 0.09040624999997889  
# n\_estimators=150: 0.08891722222220863  
# n\_estimators=200: 0.08896458333334252

# max\_depth = None: 0.0894187499999885  
# max\_depth = 3: 0.2344848999876909  
# max\_depth = 6: 0.1430118099902252  
# max\_depth = 9: 0.09882078561614734  
# max\_depth = 12: 0.09161566813435845  
# max\_depth = 15: 0.09093576059702974

# min\_samples\_leaf=1: 0.09026874999999326   
# min\_samples\_leaf=2: 0.10873822106482246   
# min\_samples\_leaf=3: 0.1366008419497182  
# min\_samples\_leaf=4: 0.15602882944996257  
# min\_samples\_leaf= 5: 0.17086420600408136

# min\_samples\_split = 2: 0.09122499999996923  
# min\_samples\_split = 4: 0.10409304291425461  
# min\_samples\_split = 6: 0.1190442885150001  
# min\_samples\_split = 8: 0.13221900643494705  
# min\_samples\_split = 10: 0.14725754257626963  
# min\_samples\_split = 12: 0.15693980142460404

# max\_leaf\_nodes = None: 0.09163624999997731  
# max\_leaf\_nodes = 2: 0.30218448943443427  
# max\_leaf\_nodes = 25: 0.1495650870862638  
# max\_leaf\_nodes = 50: 0.10946501408577684  
# max\_leaf\_nodes = 75: 0.09609355492056319  
# max\_leaf\_nodes = 100: 0.09133485879167083

#rf1 evs  
print('rf1 EVS is:', explained\_variance\_score(y1, y1\_predrf1))

rf1 EVS is: 0.9345063276775131

# n\_estimators=5: 0.8817513933118338  
# n\_estimators=50: 0.9236357106630939  
# n\_estimators=100: 0.9329695268740255  
# n\_estimators=150: 0.935461098788956  
# n\_estimators=200: 0.9367720840641142

# max\_depth = None: 0.9339774777327685  
# max\_depth = 3: 0.571097768373038  
# max\_depth = 6: 0.8465412291249891  
# max\_depth = 9: 0.9237384452895795  
# max\_depth = 12: 0.9312517321042814  
# max\_depth = 15: 0.9340040872365292

# min\_samples\_leaf=1: 0.9323811369419556  
# min\_samples\_leaf=2: 0.898106202077794  
# min\_samples\_leaf=3: 0.83692922880156572  
# min\_samples\_leaf=4: 0.7975481702978278  
# min\_samples\_leaf= 5: 0.7596433142461536

# min\_samples\_split = 2: 0.9296340295291945  
# min\_samples\_split = 4: 0.9111393514793884  
# min\_samples\_split = 6: 0.8797872307963757  
# min\_samples\_split = 8: 0.8573341670961596  
# min\_samples\_split = 10: 0.822425191864775  
# min\_samples\_split = 12: 0.8009919750314262

# max\_leaf\_nodes = None: 0.9325340827243099  
# max\_leaf\_nodes = 2: 0.2973873831518625  
# max\_leaf\_nodes = 25: 0.8392278932137353  
# max\_leaf\_nodes = 50: 0.911569843073005  
# max\_leaf\_nodes = 75: 0.92867737012289  
# max\_leaf\_nodes = 100: 0.9345063276775131

# Fitting the training model

# Instantiation  
model = RandomForestRegressor(n\_estimators = 150, max\_depth = 12, min\_samples\_leaf = 2, min\_samples\_split = 2, max\_features = 2, max\_leaf\_nodes = 75)

# Fitting the training model  
lake\_rf1 = model.fit(X\_train, y1\_train)

print(lake\_rf1)

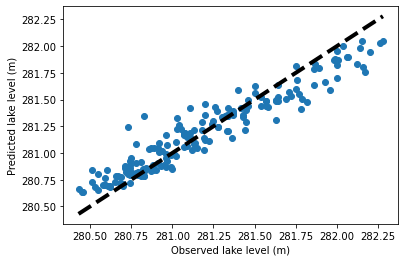
RandomForestRegressor(max\_depth=12, max\_features=2, max\_leaf\_nodes=75,  
 min\_samples\_leaf=2, n\_estimators=150)

# The coefficient of determination  
  
print('The rf1 coefficient of determination on the training model is:', lake\_rf1.score(X\_train, y1\_train))

The rf1 coefficient of determination on the training model is: 0.8766769514756587

# Predicting on X\_train  
  
ytrain\_predrf1 = lake\_rf1.predict(X\_train)

# Plotting the scatter plot of the correlation test  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_train, ytrain\_predrf1)  
ax.plot([y1\_train.min(),y1\_train.max()], [y1\_train.min(), y1\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtrainrf1 = cov(y1\_train, ytrain\_predrf1)  
print(covtrainrf1)

[[0.21501729 0.16353975]  
 [0.16353975 0.13638344]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortrainrf1 = pearsonr(y1\_train, ytrain\_predrf1)  
print(cortrainrf1)

(0.9550053787638286, 6.907858685542796e-96)

# rf1 MSE  
print('The rf1 MSE is: %.2f'% mean\_squared\_error(y1\_train, ytrain\_predrf1))

The rf1 MSE is: 0.03

# rf1 RMSE  
import numpy as np  
print('The rf1 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y1\_train, ytrain\_predrf1)))

The rf1 RMSE is: 0.16

#rf1 MAE  
print('rf1 MAE is:', mean\_absolute\_error(y1\_train, ytrain\_predrf1))

rf1 MAE is: 0.12452905595638855

#rf1  
print('rf1 EVS is:', explained\_variance\_score(y1\_train, ytrain\_predrf1))

rf1 EVS is: 0.8766824355332881

# Saving the predicted values  
numpy.savetxt('E:/Lake Level/RF/rfLLRtrainbest.csv', ytrain\_predrf1, delimiter = ',')

# Prediction  
  
pred\_rf1 = lake\_rf1.predict(X\_test)

print(pred\_rf1[:5])

[281.4683846 281.15332775 281.29278579 280.81804553 281.42227325]

# To save the predicted data on C  
numpy.savetxt('E:/Lake Level/RF/rfLLRtrainbest.csv', pred\_rf1, delimiter = ',')

# The prediction r\_sq  
# The Prediction coefficient of determination  
print('The rf1 coefficient of determination of the prediction is:',r2\_score(y1\_test, pred\_rf1) )

The rf1 coefficient of determination of the prediction is: 0.4203114069524415

# n\_estimators=5: 0.3101455703606575  
# n\_estimators=50: 0.41495619710833676  
# n\_estimators=100: 0.42571026344332763  
# n\_estimators=150: 0.3957787520871038  
# n\_estimators=200: 0.4163941769872056

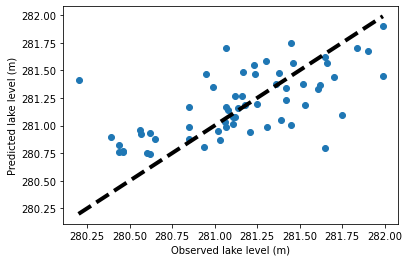
# max\_depth = None: 0.4161040189218286  
# max\_depth = 3: 0.3170828523779732  
# max\_depth = 6: 0.4167703615288163  
# max\_depth = 9: 0.4350579412050445  
# max\_depth = 12: 0.4066050767314152  
# max\_depth = 15: 0.3991537919765187

# min\_samples\_leaf=1: 0.4089912624838169  
# min\_samples\_leaf=2: 0.38470512592510775  
# min\_samples\_leaf=3: 0.4166717793761596  
# min\_samples\_leaf=4: 0.3656648237647264  
# min\_samples\_leaf= 5:0.3380908597465174

# min\_samples\_split = 2: 0.4233631885855983  
# min\_samples\_split = 4: 0.3866211223468714  
# min\_samples\_split = 6: 0.4146539114453599  
# min\_samples\_split = 8: 0.39459067881598087  
# min\_samples\_split = 10: 0.41435459476868997  
# min\_samples\_split = 12: 0.3857278415556752

# max\_leaf\_nodes = None: 0.4193109944143689  
# max\_leaf\_nodes = 2: 0.2558994315398737  
# max\_leaf\_nodes = 25: 0.4105229565381896  
# max\_leaf\_nodes = 50: 0.4001678156833808  
# max\_leaf\_nodes = 75: 0.4177119431187465  
# max\_leaf\_nodes = 100: 0.41138037650390247

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_rf1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



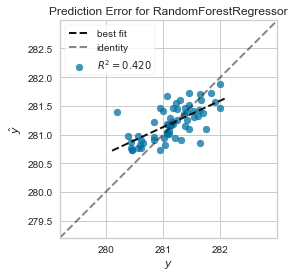
# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtestrf1 = cov(y1\_test, pred\_rf1)  
print(covtestrf1)

[[0.1852037 0.08673858]  
 [0.08673858 0.09185775]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestrf1 = pearsonr(y1\_test, pred\_rf1)  
print(cortestrf1)

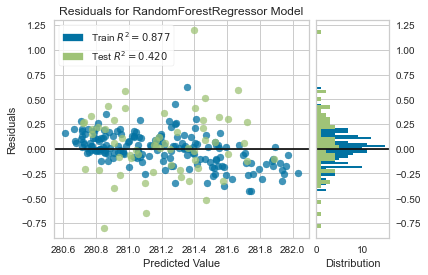
(0.6650121010400969, 6.819717973176086e-09)

# Plotting the prediction errors  
from yellowbrick.regressor import PredictionError  
visualizer = PredictionError(lake\_rf1)  
visualizer.fit(X\_train, y1\_train)  
visualizer.score(X\_test, y1\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Prediction Error for RandomForestRegressor'}, xlabel='$y$', ylabel='$\\hat{y}$'>

# Plotting the residuals  
from yellowbrick.regressor import ResidualsPlot  
visualizer = ResidualsPlot(lake\_rf1)  
visualizer.fit(X\_train, y1\_train)  
visualizer.score(X\_test, y1\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Residuals for RandomForestRegressor Model'}, xlabel='Predicted Value', ylabel='Residuals'>

# Model evaluation

# The MSE  
   
print('The RS Data rf1 MSE is:',metrics.mean\_squared\_error(y1\_test, pred\_rf1))

The RS Data rf1 MSE is: 0.10557113139758113

# n\_estimators=5: 0.12563420000000491  
# n\_estimators=50: 0.10654640600000413  
# n\_estimators=100: 0.10458790799999866  
# n\_estimators=150: 0.1100389442222292  
# n\_estimators=200: 0.10628452545834806

# max\_depth = None: 0.10633736816668038  
# max\_depth = 3: 0.1243708032035597  
# max\_depth = 6: 0.10621601586863312  
# max\_depth = 9: 0.1028855372287258  
# max\_depth = 12: 0.10806728675771238  
# max\_depth = 15: 0.10942429217643973

# min\_samples\_leaf=1: 0.10763272183332519   
# min\_samples\_leaf=2: 0.1120556394903727   
# min\_samples\_leaf=3: 0.10623396935179011  
# min\_samples\_leaf=4: 0.11552320167001759  
# min\_samples\_leaf=5:0.12054488850917829

# min\_samples\_split = 2: 0.10501535016666724  
# min\_samples\_split = 4: 0.1117067040232525  
# min\_samples\_split = 6: 0.10660145734969165  
# min\_samples\_split = 8: 0.11025531252914426  
# min\_samples\_split = 10: 0.1066559679965824  
# min\_samples\_split = 12: 0.11186938561628942

# max\_leaf\_nodes = None: 0.10575332349998806  
# max\_leaf\_nodes = 2: 0.13551334255679237  
# max\_leaf\_nodes = 25: 0.10735377434977206  
# max\_leaf\_nodes = 50: 0.10923962124918446  
# max\_leaf\_nodes = 75: 0.10604453788037485  
# max\_leaf\_nodes = 100: 0.10719762362169354

# The RMSE  
   
print('The RS Data rf1 RMSE is:',np.sqrt(metrics.mean\_squared\_error(y1\_test, pred\_rf1)))

The RS Data rf1 RMSE is: 0.324917114657848

# n\_estimators=5: 0.35444915009067934   
# n\_estimators=50: 0.32641446965476906  
# n\_estimators=100: 0.32340053803294555  
# n\_estimators=150: 0.33172118446404536  
# n\_estimators=200: 0.32601307559413634

# max\_depth = None: 0.32609410937132915  
# max\_depth = 3: 0.3526624493812174  
# max\_depth = 6: 0.32590798681320027  
# max\_depth = 9: 0.3207577547444891  
# max\_depth = 12: 0.32873589210445575  
# max\_depth = 15: 0.3307934282546129

# min\_samples\_leaf=1: 0.32807426268045653  
# min\_samples\_leaf=2: 0.33474712768054143  
# min\_samples\_leaf=3: 0.3259355294407011  
# min\_samples\_leaf=4: 0.339887042515624  
# min\_samples\_leaf=5: 0.3471957495551728

# min\_samples\_split = 2: 0.32406071987617885  
# min\_samples\_split = 4: 0.3342255286827332  
# min\_samples\_split = 6: 0.3264987861381595  
# min\_samples\_split = 8: 0.33204715407475527  
# min\_samples\_split = 10: 0.32658225303372257  
# min\_samples\_split = 12: 0.33446881112637306

# max\_leaf\_nodes = None: 0.3251973608441312  
# max\_leaf\_nodes = 2: 0.3681213693291825  
# max\_leaf\_nodes = 25: 0.33051417707745073  
# max\_leaf\_nodes = 50: 0.32564480324484657  
# max\_leaf\_nodes = 75: 0.3270520375270383  
# max\_leaf\_nodes = 100: 0.32741048184457006

# The MAE  
  
print('The RS Data rf1 MAE is:', metrics.mean\_absolute\_error(y1\_test, pred\_rf1))

The RS Data rf1 MAE is: 0.24335251667468186

# n\_estimators=5: 0.2799666666666714  
# n\_estimators=50: 0.24999666666667697  
# n\_estimators=100: 0.24605000000000435  
# n\_estimators=150: 0.2531033333333369  
# n\_estimators=200: 0.2515591666666751

# max\_depth = None: 0.2535650000000013  
# max\_depth = 3: 0.2805354620030945  
# max\_depth = 6: 0.25395746911294453  
# max\_depth = 9: 0.2475251059606279  
# max\_depth = 12: 0.25337358730159376  
# max\_depth = 15: 0.2506811904761804

# min\_samples\_leaf=1: 0.25470166666664984  
# min\_samples\_leaf=2: 0.2562516728595625  
# min\_samples\_leaf=3: 0.2506362740823903  
# min\_samples\_leaf=4: 0.26031879996501744  
# min\_samples\_leaf=5: 0.2687892923878072

# min\_samples\_split = 2: 0.2489616666666649  
# min\_samples\_split = 4: 0.2537006034151176  
# min\_samples\_split = 6: 0.2509846465941545  
# min\_samples\_split = 8: 0.25429711366433216  
# min\_samples\_split = 10: 0.25374541886501256  
# min\_samples\_split = 12: 0.2589452363641612

# max\_leaf\_nodes = None: 0.2488549999999833  
# max\_leaf\_nodes = 2: 0.29323559661515997  
# max\_leaf\_nodes = 25: 0.25183866500283897  
# max\_leaf\_nodes = 50: 0.2513755565277184  
# max\_leaf\_nodes = 75: 0.25534518308540916  
# max\_leaf\_nodes = 100: 0.2494688333333253

# The rf1 Explained variance score  
from sklearn.metrics import explained\_variance\_score  
  
print('rf1 EVS is:', explained\_variance\_score(y1\_test, pred\_rf1))

rf1 EVS is: 0.4324231771308473

# n\_estimators=5: 0.3152089888503916  
# n\_estimators=50: 0.4198531524381858  
# n\_estimators=100: 0.42718771686879975  
# n\_estimators=150: 0.40040622249646307  
# n\_estimators=200: 0.4203998172914686

# max\_depth = None: 0.41923396127726653  
# max\_depth = 3: 0.329299091155541  
# max\_depth = 6: 0.42440288107918267  
# max\_depth = 9: 0.4377017706081825  
# max\_depth = 12: 0.4115298222553102  
# max\_depth = 15: 0.40334226521058125

# min\_samples\_leaf=1: 0.4131745738659691  
# min\_samples\_leaf=2: 0.3890763798570914  
# min\_samples\_leaf=3: 0.4196486065301077  
# min\_samples\_leaf=4: 0.3720365039161555  
# min\_samples\_leaf=5: 0.34345508997777363

# min\_samples\_split = 2: 0.42835405786971426  
# min\_samples\_split = 4: 0.39143420232930815  
# min\_samples\_split = 6: 0.4191516584301389  
# min\_samples\_split = 8: 0.4055426936380394  
# min\_samples\_split = 10: 0.421915758324131  
# min\_samples\_split = 12: 0.39464399749079804

# max\_leaf\_nodes = None: 0.4232720204551912  
# max\_leaf\_nodes = 2: 0.2813804445429813  
# max\_leaf\_nodes = 25: 0.4152776831726118  
# max\_leaf\_nodes = 50: 0.40645240421673834  
# max\_leaf\_nodes = 75: 0.4209932918512145  
# max\_leaf\_nodes = 100: 0.41502479742950393

# The k-fold cross-validation  
from sklearn.model\_selection import cross\_val\_score  
from numpy import absolute

# On the whole model  
score = cross\_val\_score(lake\_rf1, X, y1, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score

array([-0.10744274, -0.09482344, -0.12023139, -0.16525508, -0.09047302,  
 -0.04782529, -0.15848074, -0.15556075, -0.10240444, -0.12846364])

# The absolute mean score on the model  
print(absolute(np.mean(score)))

0.11709605274812748

# n\_estimators=5: 0.13104471666666698  
# n\_estimators=50: 0.1185037345000032  
# n\_estimators=100: 0.11578166412499932  
# n\_estimators=150: 0.11512316251851369  
# n\_estimators=200: 0.11685892417707426

# max\_depth = None: 0.11848137775000749  
# max\_depth = 3: 0.14244422427128006  
# max\_depth = 6: 0.11646728450654172  
# max\_depth = 9: 0.11620326978709312  
# max\_depth = 12: 0.11670719550530088  
# max\_depth = 15: 0.11604560146039236

# min\_samples\_leaf=1: 0.11407063225000431  
# min\_samples\_leaf=2: 0.10677368195993017  
# min\_samples\_leaf=3: 0.12242419462831469  
# min\_samples\_leaf=4: 0.12244751775375018  
# min\_samples\_leaf=5: 0.12944139250847556

# min\_samples\_split = 2: 0.11743723287499963  
# min\_samples\_split = 4: 0.11774482965106044  
# min\_samples\_split = 6: 0.11537564980583423  
# min\_samples\_split = 8: 0.11795174831013358  
# min\_samples\_split = 10: 0.12052479395100842  
# min\_samples\_split = 12: 0.1230489130341568

# max\_leaf\_nodes = None: 0.11650783579167327  
# max\_leaf\_nodes = 2: 0.16955752535265228  
# max\_leaf\_nodes = 25: 0.1173672022055023  
# max\_leaf\_nodes = 50: 0.11646743327546065  
# max\_leaf\_nodes = 75: 0.11504335430088446  
# max\_leaf\_nodes = 100: 0.11709605274812748

# On the training dataset  
score\_train = cross\_val\_score(lake\_rf1, X\_train, y1\_train, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score\_train

array([-0.12169985, -0.07083377, -0.06572458, -0.14584904, -0.07212566,  
 -0.12502955, -0.16992393, -0.13475428, -0.05885668, -0.11079078])

# The absolute mean score on the training dataset  
print(absolute(np.mean(score\_train)))

0.10755881103891471

# On the testing dataset  
score\_test = cross\_val\_score(lake\_rf1, X\_test, y1\_test, scoring ='neg\_mean\_squared\_error', cv = 10)  
score\_test

array([-0.09456883, -0.02405407, -0.07191878, -0.16509531, -0.11238726,  
 -0.29727277, -0.15829698, -0.14335555, -0.05430117, -0.11573715])

# The absolute mean score on the testing dataset  
print(absolute(np.mean(score\_test)))

0.12369878821444288

# n\_estimators=5: 0.1578649333333308   
# n\_estimators=50: 0.15078209933331493  
# n\_estimators=100: 0.14889716866666486  
# n\_estimators=150: 0.15155564725925982  
# n\_estimators=200: 0.14527509208332937

# max\_depth = None: 0.14585897100000814  
# max\_depth = 3: 0.1379223508901645  
# max\_depth = 6: 0.15083003820606994  
# max\_depth = 9: 0.1490635623600697  
# max\_depth = 12: 0.14249242878038265  
# max\_depth = 15: 0.14752358758332457

# min\_samples\_leaf=1: 0.14721756933332367  
# min\_samples\_leaf=2: 0.13308944302208792  
# min\_samples\_leaf=3: 0.12717543296812794  
# min\_samples\_leaf=4: 0.12681556945513658  
# min\_samples\_leaf=5: 0.12689980109420534

# min\_samples\_split = 2: 0.14458388533333646  
# min\_samples\_split = 4: 0.14867394619309554  
# min\_samples\_split = 6: 0.14209751916703575  
# min\_samples\_split = 8: 0.1417091729653371  
# min\_samples\_split = 10: 0.1468787567048182  
# min\_samples\_split = 12: 0.1502116875090235

# max\_leaf\_nodes = None: 0.1481241775000037  
# max\_leaf\_nodes = 2: 0.14297036100540644  
# max\_leaf\_nodes = 25: 0.14363437176557603  
# max\_leaf\_nodes = 50: 0.14260273166666237  
# max\_leaf\_nodes = 75: 0.14928588349999503  
# max\_leaf\_nodes = 100: 0.14813674933334023

# RF Feature ranking  
import pandas as pd  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import RandomForestRegressor

# Instantiation  
model = RandomForestRegressor(n\_estimators = 150, max\_depth = 12, min\_samples\_leaf = 2, min\_samples\_split = 2, max\_features = 2, max\_leaf\_nodes = 75)

# Fitting the model  
lake\_rf1 = model.fit(X, y1)

f\_list = list(X.columns)  
f\_importance = pd.Series(lake\_rf1.feature\_importances\_, index = f\_list).sort\_values(ascending = False)  
print(f\_importance)

P 0.204014  
ST 0.194979  
AT 0.183888  
SM 0.155531  
SH 0.139356  
ET 0.122232  
dtype: float64

###########################################################################

# RF: Ground truth lake level data as output feature

# Instantiation  
model = RandomForestRegressor(max\_leaf\_nodes = 100)

# Fitting the model  
lake\_rf2 = model.fit(X, y2)

print(lake\_rf2)

# Training Model evaluation  
from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error, explained\_variance\_score

# The coefficient of determination  
  
print('The rf2 coefficient of determination of the model is:', lake\_rf2.score(X, y2))

The rf2 coefficient of determination of the model is: 0.9430820822878868

# n\_estimators=5: 0.8896287579164749  
# n\_estimators=50: 0.9443116564617662  
# n\_estimators=100 (default): 0.9469936669076754  
# n\_estimators=150: 0.9475660050817919  
# n\_estimators=200: 0.9455163117782586

# max\_depth = None: 0.9452540686768097  
# max\_depth = 3: 0.6936667316077897  
# max\_depth = 6: 0.8788135367801024  
# max\_depth = 9: 0.9367300961602074  
# max\_depth = 12: 0.9432675147283196  
# max\_depth = 15: 0.9444926035910809

# min\_samples\_leaf=1 (default): 0.9435171944900618  
# min\_samples\_leaf=2: 0.9207181018786981  
# min\_samples\_leaf=3: 0.8806441652850843  
# min\_samples\_leaf=4: 0.8455526952889904  
# min\_samples\_leaf=5: 0.8120980070552839

# min\_samples\_split = 2: 0.9482516941723562  
# min\_samples\_split = 4: 0.9281172125800863  
# min\_samples\_split = 6: 0.9091914311039473  
# min\_samples\_split = 8: 0.8873644337305665  
# min\_samples\_split = 10: 0.8613698234302274  
# min\_samples\_split = 12: 0.8416621772658663

# max\_leaf\_nodes = None: 0.943066934832025  
# max\_leaf\_nodes = 2: 0.4580675938330918  
# max\_leaf\_nodes = 25: 0.8559480125530132  
# max\_leaf\_nodes = 50: 0.9240003360028418  
# max\_leaf\_nodes = 75: 0.9422087718299069  
# max\_leaf\_nodes = 100: 0.9430820822878868  
# max\_leaf\_nodes = 125: 0.9401490992215795

# Predicting on X\_train  
  
y2\_predrf2 = lake\_rf2.predict(X)

# rf2 MSE  
print('The rf2 MSE is: %.2f'% mean\_squared\_error(y2, y2\_predrf2))

The rf2 MSE is: 0.03

# n\_estimators=5: 0.05   
# n\_estimators=50: 0.03  
# n\_estimators=100: 0.03  
# n\_estimators=150: 0.02  
# n\_estimators=200: 0.03

# max\_depth = None: 0.03  
# max\_depth = 3: 0.15  
# max\_depth = 6: 0.06  
# max\_depth = 9: 0.03  
# max\_depth = 12: 0.03  
# max\_depth = 15: 0.03

# min\_samples\_leaf=1: 0.03  
# min\_samples\_leaf=2: 0.04  
# min\_samples\_leaf=3: 0.06  
# min\_samples\_leaf=4: 0.07  
# min\_samples\_leaf=5: 0.09

# min\_samples\_split = 2: 0.02  
# min\_samples\_split = 4: 0.03  
# min\_samples\_split = 6: 0.04  
# min\_samples\_split = 8: 0.06  
# min\_samples\_split = 10: 0.07  
# min\_samples\_split = 12: 0.08

# max\_leaf\_nodes = None: 0.03  
# max\_leaf\_nodes = 2: 0.26  
# max\_leaf\_nodes = 25: 0.07   
# max\_leaf\_nodes = 50: 0.04  
# max\_leaf\_nodes = 75: 0.03  
# max\_leaf\_nodes = 100: 0.03

# rf2 RMSE  
print('The rf2 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y2, y2\_predrf2)))

The rf2 RMSE is: 0.16

# n\_estimators=5: 0.23  
# n\_estimators=50: 0.17  
# n\_estimators=100: 0.17  
# n\_estimators=150: 0.14142135623730950488016887242097  
# n\_estimators=200: 0.17

# max\_depth = None: 0.17  
# max\_depth = 3: 0.38  
# max\_depth = 6: 0.24  
# max\_depth = 9: 0.17  
# max\_depth = 12: 0.17  
# max\_depth = 15: 0.17

# min\_samples\_leaf=1: 0.17  
# min\_samples\_leaf=2: 0.20  
# min\_samples\_leaf=3: 0.24  
# min\_samples\_leaf=4: 0.26  
# min\_samples\_leaf=5: 0.30

# min\_samples\_split = 2: 0.14  
# min\_samples\_split = 4: 0.17  
# min\_samples\_split = 6: 0.20  
# min\_samples\_split = 8: 0.22  
# min\_samples\_split = 10: 0.26  
# min\_samples\_split = 12: 0.28

# max\_leaf\_nodes = None: 0.17  
# max\_leaf\_nodes = 2: 0.51  
# max\_leaf\_nodes = 25: 0.26  
# max\_leaf\_nodes = 50: 0.20  
# max\_leaf\_nodes = 75: 0.17  
# max\_leaf\_nodes = 100: 0.17

#rf2 MAE  
print('rf2 MAE is:', mean\_absolute\_error(y2, y2\_predrf2))

rf2 MAE is: 0.13299122797109983

# n\_estimators=5: 0.15989166666666535  
# n\_estimators=50: 0.13210000000001423  
# n\_estimators=100: 0.12871708333332896  
# n\_estimators=150: 0.1288072222222387  
# n\_estimators=200: 0.13190000000003507

# max\_depth = None: 0.13138749999999605  
# max\_depth = 3: 0.31415276754530386  
# max\_depth = 6: 0.19807753450657753  
# max\_depth = 9: 0.14140263874679065  
# max\_depth = 12: 0.13086950068120468  
# max\_depth = 15: 0.13260403472221755

# min\_samples\_leaf=1: 0.13096749999999582  
# min\_samples\_leaf=2: 0.15619877525252074  
# min\_samples\_leaf=3: 0.1903491713124782  
# min\_samples\_leaf=4: 0.21728626172973028  
# min\_samples\_leaf=5: 0.23965006432354155

# min\_samples\_split = 2: 0.12817624999999566  
# min\_samples\_split = 4: 0.14525540725708788  
# min\_samples\_split = 6: 0.16531808085358152  
# min\_samples\_split = 8: 0.18565126172844207  
# min\_samples\_split = 10: 0.2059516889969134  
# min\_samples\_split = 12: 0.22295754362190792

# max\_leaf\_nodes = None: 0.13117833333333498  
# max\_leaf\_nodes = 2: 0.4105200662290874  
# max\_leaf\_nodes = 25: 0.21800690045872742  
# max\_leaf\_nodes = 50: 0.15789794942560273  
# max\_leaf\_nodes = 75: 0.13533841050269946  
# max\_leaf\_nodes = 100: 0.13299122797109983

#rf2 EVS  
print('rf2 EVS is:', explained\_variance\_score(y2, y2\_predrf2))

rf2 EVS is: 0.9430920047670591

# n\_estimators=5: 0.891147317246527  
# n\_estimators=50: 0.9444020424329902  
# n\_estimators=100: 0.9470159143809593  
# n\_estimators=150: 0.9476404942444734  
# n\_estimators=200: 0.9456455565860489

# max\_depth = None: 0.9454535613931279  
# max\_depth = 3: 0.693667510520694  
# max\_depth = 6: 0.8788953897145431  
# max\_depth = 9: 0.9367513162293519  
# max\_depth = 12: 0.9457821075747503  
# max\_depth = 15: 0.9445074480919645

# min\_samples\_leaf=1: 0.9435640346981623  
# min\_samples\_leaf=2: 0.9207996533498628  
# min\_samples\_leaf=3: 0.880764033728949  
# min\_samples\_leaf=4: 0.8456072069104468  
# min\_samples\_leaf=5: 0.8121593115000655

# min\_samples\_split = 2: 0.9482535634828463  
# min\_samples\_split = 4: 0.928122080423508  
# min\_samples\_split = 6: 0.9091937205297856  
# min\_samples\_split = 8: 0.887492911467145  
# min\_samples\_split = 10: 0.8614550635094312  
# min\_samples\_split = 12: 0.8417010539410441

# max\_leaf\_nodes = None: 0.9430699961571104  
# max\_leaf\_nodes = 2: 0.4581299099734246  
# max\_leaf\_nodes = 25: 0.8560548592392404  
# max\_leaf\_nodes = 50: 0.9240986283282163  
# max\_leaf\_nodes = 75: 0.9422149100459809  
# max\_leaf\_nodes = 100: 0.9430920047670591

# Instantiation  
model = RandomForestRegressor(n\_estimators = 150, max\_depth = 12, min\_samples\_leaf = 2, min\_samples\_split = 2, max\_features = 2, max\_leaf\_nodes = 75)

# Fitting the training model  
lake\_rf2 = model.fit(X\_train, y2\_train)

print(lake\_rf2)

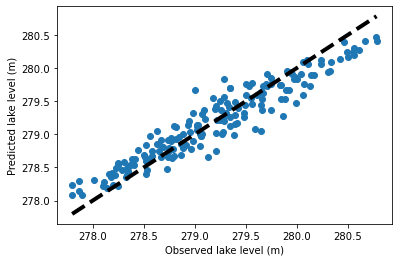
RandomForestRegressor(max\_depth=12, max\_features=2, max\_leaf\_nodes=75,  
 min\_samples\_leaf=2, n\_estimators=150)

# The coefficient of determination  
  
print('The rf2 coefficient of determination of the training model is:', lake\_rf2.score(X\_train, y2\_train))

The rf2 coefficient of determination of the training model is: 0.8876351407769257

# Predicting on X\_train  
  
ytrain\_predrf2 = lake\_rf2.predict(X\_train)

# Plotting the scatter plot of the correlation test  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_train, ytrain\_predrf2)  
ax.plot([y2\_train.min(),y2\_train.max()], [y2\_train.min(), y2\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtrainrf2 = cov(y2\_train, ytrain\_predrf2)  
print(covtrainrf2)

[[0.49448696 0.39075447]  
 [0.39075447 0.34198035]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortrainrf2 = pearsonr(y2\_train, ytrain\_predrf2)  
print(cortrainrf2)

(0.950223907328464, 4.471002422632074e-92)

# rf2 MSE  
print('The rf2 MSE is: %.2f'% mean\_squared\_error(y2\_train, ytrain\_predrf2))

The rf2 MSE is: 0.06

# rf2 RMSE  
print('The rf2 RMSE is: %.2f'% np.sqrt(mean\_squared\_error(y2\_train, ytrain\_predrf2)))

The rf2 RMSE is: 0.24

#rf2 MAE  
print('rf2 MAE is:', mean\_absolute\_error(y2\_train, ytrain\_predrf2))

rf2 MAE is: 0.1939661273983344

#rf2 EVS  
print('rf2 EVS is:', explained\_variance\_score(y2\_train, ytrain\_predrf2))

rf2 EVS is: 0.8876680028650344

# Prediction  
  
pred\_rf2 = lake\_rf2.predict(X\_test)

print(pred\_rf2[:5])

[279.91335206 278.45822714 279.71363119 279.11715889 279.75547841]

# To save the predicted data on C  
numpy.savetxt('E:/Lake Level/RF/rfLLGtestbest.csv', pred\_rf2, delimiter = ',')

# The Prediction coefficient of determination  
print('The rf2 coefficient of determination of the prediction is:',r2\_score(y2\_test, pred\_rf2) )

The rf2 coefficient of determination of the prediction is: 0.6194792323180314

# n\_estimators=5: 0.42571060954556006  
# n\_estimators=50: 0.6115059115507759  
# n\_estimators=100: 0.5942868356484635  
# n\_estimators=150: 0.6204719162082831  
# n\_estimators=200: 0.6154710792947657

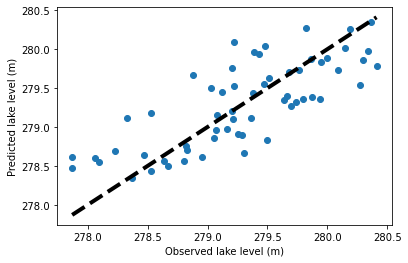
# max\_depth = None: 0.599509365120575  
# max\_depth = 3: 0.5650969767140663  
# max\_depth = 6: 0.6227023274524864  
# max\_depth = 9: 0.6192569479902732  
# max\_depth = 12: 0.6075356684560344  
# max\_depth = 15: 0.5947090137811049

# min\_samples\_leaf=1: 0.5994655243008152  
# min\_samples\_leaf=2: 0.6219026984085025  
# min\_samples\_leaf=3: 0.6131419328301301  
# min\_samples\_leaf=4: 0.619495519433096  
# min\_samples\_leaf=5: 0.5932442096952423

# min\_samples\_split = 2: 0.600352192454174  
# min\_samples\_split = 4: 0.6151313748850452  
# min\_samples\_split = 6: 0.6062514643878303   
# min\_samples\_split = 8: 0.6035539720320362  
# min\_samples\_split = 10: 0.624029019878042  
# min\_samples\_split = 12: 0.6242678065366978

# max\_leaf\_nodes = None: 0.6106309885496479  
# max\_leaf\_nodes = 2: 0.44653855017385424  
# max\_leaf\_nodes = 25: 0.6110069575741177  
# max\_leaf\_nodes = 50: 0.6235922732561365  
# max\_leaf\_nodes = 75: 0.5908109955418792  
# max\_leaf\_nodes = 100: 0.6064143502021754

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_test, pred\_rf2)  
ax.plot([y2\_test.min(),y2\_test.max()], [y2\_test.min(), y2\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtestrf2 = cov(y2\_test, pred\_rf2)  
print(covtestrf2)

[[0.42726607 0.27784125]  
 [0.27784125 0.30050317]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestrf2 = pearsonr(y2\_test, pred\_rf2)  
print(cortestrf2)

(0.7753950019919903, 3.5064679935657697e-13)

# Plotting the prediction errors  
from yellowbrick.regressor import PredictionError  
visualizer = PredictionError(lake\_rf2)  
visualizer.fit(X\_train, y2\_train)  
visualizer.score(X\_test, y2\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Prediction Error for RandomForestRegressor'}, xlabel='$y$', ylabel='$\\hat{y}$'>

# Plotting the residuals  
from yellowbrick.regressor import ResidualsPlot  
visualizer = ResidualsPlot(lake\_rf2)  
visualizer.fit(X\_train, y2\_train)  
visualizer.score(X\_test, y2\_test)  
visualizer.poof()



<AxesSubplot:title={'center':'Residuals for RandomForestRegressor Model'}, xlabel='Predicted Value', ylabel='Residuals'>

# Model evaluation

# The MSE  
   
print('The G Data rf2 MSE is:',metrics.mean\_squared\_error(y2\_test, pred\_rf2) )

The G Data rf2 MSE is: 0.15987388736771965

# n\_estimators=5: 0.24128479999999802  
# n\_estimators=50: 0.16322383799999704  
# n\_estimators=100: 0.17045834616666647  
# n\_estimators=150: 0.1594568162222244  
# n\_estimators=200: 0.161557892708342

# max\_depth = None: 0.16826412666667648  
# max\_depth = 3: 0.18272231863782937  
# max\_depth = 6: 0.15851972015198446  
# max\_depth = 9: 0.1599672790104311  
# max\_depth = 12: 0.16489191567475275  
# max\_depth = 15: 0.1702809701468551

# min\_samples\_leaf=1: 0.16828254616667662  
# min\_samples\_leaf=2: 0.15885568027445715  
# min\_samples\_leaf=3: 0.16253647188502784  
# min\_samples\_leaf=4: 0.15986704441821323  
# min\_samples\_leaf=5: 0.17089640021882077

# min\_samples\_split = 2: 0.16791001700001335  
# min\_samples\_split = 4: 0.16170061780812783  
# min\_samples\_split = 6: 0.16543146755731597  
# min\_samples\_split = 8: 0.1665648054082108  
# min\_samples\_split = 10: 0.15796231699970198  
# min\_samples\_split = 12: 0.15786199198563403

# max\_leaf\_nodes = None: 0.1635914325000026  
# max\_leaf\_nodes = 2: 0.23253404546327724  
# max\_leaf\_nodes = 25: 0.16343347100466027  
# max\_leaf\_nodes = 50: 0.15814581389703058  
# max\_leaf\_nodes = 75: 0.1719187029116962  
# max\_leaf\_nodes = 100: 0.1653630319013726

# The RMSE  
   
print('The G Data rf2 RMSE is:',np.sqrt(metrics.mean\_squared\_error(y2\_test, pred\_rf2)))

The G Data rf2 RMSE is: 0.39984232813412796

# n\_estimators=5: 0.49120749179954293  
# n\_estimators=50: 0.40400970037858874  
# n\_estimators=100: 0.41286601478768686  
# n\_estimators=150: 0.3993204430306873  
# n\_estimators=200: 0.40194264853128236

# max\_depth = None: 0.4102001056395238  
# max\_depth = 3: 0.4274603123540586  
# max\_depth = 6: 0.3981453505341792  
# max\_depth = 9: 0.39995909667168605  
# max\_depth = 12: 0.4060688558295905  
# max\_depth = 15: 0.41265114824371335

# min\_samples\_leaf=1: 0.41022255687209186  
# min\_samples\_leaf=2: 0.3985670336021999  
# min\_samples\_leaf=3: 0.4031581226826862  
# min\_samples\_leaf=4: 0.39983377098265876  
# min\_samples\_leaf=5: 0.4133961782827954

# min\_samples\_split = 2: 0.40976824791583516  
# min\_samples\_split = 4: 0.40212015344686197  
# min\_samples\_split = 6: 0.4067326733338692  
# min\_samples\_split = 8: 0.4081235173427412  
# min\_samples\_split = 10: 0.3974447345225522  
# min\_samples\_split = 12: 0.3973185019422504

# max\_leaf\_nodes = None: 0.4044643772942218  
# max\_leaf\_nodes = 2: 0.48221784025819414  
# max\_leaf\_nodes = 25: 0.40426905768888655  
# max\_leaf\_nodes = 50: 0.3976755133234011  
# max\_leaf\_nodes = 75: 0.41463080313900486  
# max\_leaf\_nodes = 100: 0.4066485360865973

# The MAE  
  
print('The G Data rf2 MAE is:', metrics.mean\_absolute\_error(y2\_test, pred\_rf2))

The G Data rf2 MAE is: 0.3239236930816152

# n\_estimators=5: 0.40853333333333147  
# n\_estimators=50: 0.3231566666666557  
# n\_estimators=100: 0.3405083333333541  
# n\_estimators=150: 0.3244233333333568  
# n\_estimators=200: 0.3237808333333694

# max\_depth = None: 0.3339366666667142  
# max\_depth = 3: 0.3557698294694404  
# max\_depth = 6: 0.32152943481457613  
# max\_depth = 9: 0.32087654521322123  
# max\_depth = 12: 0.33342383085437366  
# max\_depth = 15: 0.3412846111111321

# min\_samples\_leaf=1: 0.3302183333333716  
# min\_samples\_leaf=2: 0.3209687362313398  
# min\_samples\_leaf=3: 0.3247014696507108  
# min\_samples\_leaf=4: 0.32352839619149826  
# min\_samples\_leaf=5: 0.33518532627309316

# min\_samples\_split = 2: 0.3381266666666922  
# min\_samples\_split = 4: 0.3241626104497414  
# min\_samples\_split = 6: 0.3241518961964014  
# min\_samples\_split = 8: 0.3288345232267801  
# min\_samples\_split = 10: 0.3201432061630461  
# min\_samples\_split = 12: 0.32223273468254376

# max\_leaf\_nodes = None: 0.33366166666669034  
# max\_leaf\_nodes = 2: 0.40263096488091604  
# max\_leaf\_nodes = 25: 0.3287015631431814  
# max\_leaf\_nodes = 50: 0.3215166580294768  
# max\_leaf\_nodes = 75: 0.33822416338291816  
# max\_leaf\_nodes = 100: 0.325036011904776

# The rf2 Explained variance score  
  
print('rf2 EVS is:', explained\_variance\_score(y2\_test, pred\_rf2))

rf2 EVS is: 0.6203605115015087

# n\_estimators=5: 0.4272381246182342  
# n\_estimators=50: 0.6116812643836838  
# n\_estimators=100: 0.5942962138994493  
# n\_estimators=150: 0.6204820274301063  
# n\_estimators=200: 0.6155925590713514

# max\_depth = None: 0.5995148883199414  
# max\_depth = 3: 0.5651881298741972  
# max\_depth = 6: 0.6228496729046284  
# max\_depth = 9: 0.6192685714746242  
# max\_depth = 12: 0.6075361670582856  
# max\_depth = 15: 0.5948934684449861

# min\_samples\_leaf=1: 0.5995144746069229  
# min\_samples\_leaf=2: 0.6219033450706433  
# min\_samples\_leaf=3: 0.613267508481077  
# min\_samples\_leaf=4: 0.6196784548992518  
# min\_samples\_leaf=5: 0.5934860998214666

# min\_samples\_split = 2: 0.6022579664069325  
# min\_samples\_split = 4: 0.6155802867414845  
# min\_samples\_split = 6: 0.6063290442277518  
# min\_samples\_split = 8: 0.6035745779129927  
# min\_samples\_split = 10: 0.6240486571540123  
# min\_samples\_split = 12: 0.6247905645413829

# max\_leaf\_nodes = None: 0.6107192147697107  
# max\_leaf\_nodes = 2: 0.45008206270898  
# max\_leaf\_nodes = 25: 0.6110158892324298  
# max\_leaf\_nodes = 50: 0.6242059104479463  
# max\_leaf\_nodes = 75: 0.5909135236527968  
# max\_leaf\_nodes = 100: 0.6067377923394454

# Cross-validation  
from sklearn.model\_selection import cross\_val\_score  
from numpy import absolute

# On the model   
score = cross\_val\_score(lake\_rf2, X, y2, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score

array([-0.30468726, -0.06554447, -0.41788525, -0.50800286, -0.14004723,  
 -0.2033526 , -0.11840612, -0.12228402, -0.24328624, -0.20366138])

# The absolute mean score  
print(absolute(np.mean(score)))

0.2327157423511957

# n\_estimators=5: 0.2674777333333336   
# n\_estimators=50: 0.24086659516666242  
# n\_estimators=100: 0.22657470145833286  
# n\_estimators=150: 0.22984240424073749  
# n\_estimators=200: 0.22649567885414976

# max\_depth = None: 0.23369159570833906  
# max\_depth = 3: 0.26051332913266406  
# max\_depth = 6: 0.22977668707858184  
# max\_depth = 9: 0.23076658075339215  
# max\_depth = 12: 0.23584818168163127  
# max\_depth = 15: 0.23603850746635402

# min\_samples\_leaf=1: 0.2272398441666585  
# min\_samples\_leaf=2: 0.22965664320483414  
# min\_samples\_leaf=3: 0.23282570464986668  
# min\_samples\_leaf=4: 0.24302306346177333  
# min\_samples\_leaf=5: 0.2390702686828306

# min\_samples\_split = 2: 0.2311502454583397  
# min\_samples\_split = 4: 0.23776290748071477  
# min\_samples\_split = 6: 0.23714449101534596  
# min\_samples\_split = 8: 0.22208159969796873  
# min\_samples\_split = 10: 0.23344126358953826  
# min\_samples\_split = 12: 0.23555440083773732

# max\_leaf\_nodes = None: 0.232950075541667  
# max\_leaf\_nodes = 2: 0.3233728308037539  
# max\_leaf\_nodes = 25: 0.22804155432216616  
# max\_leaf\_nodes = 50: 0.23245547653490012  
# max\_leaf\_nodes = 75: 0.2360154430467111  
# max\_leaf\_nodes = 100: 0.2327157423511957

# On the training model  
score\_train = cross\_val\_score(lake\_rf2, X\_train, y2\_train, scoring = 'neg\_mean\_squared\_error', cv = 10)  
score\_train

array([-0.19736792, -0.14546322, -0.27567946, -0.25729464, -0.24860914,  
 -0.22142344, -0.16113019, -0.21380119, -0.15268681, -0.21758034])

# The absolute mean score on the training dataset  
print(absolute(np.mean(score\_train)))

0.20910363686489403

# On the testing dataset  
score\_test = cross\_val\_score(lake\_rf2, X\_test, y2\_test, scoring ='neg\_mean\_squared\_error', cv = 10)  
score\_test

array([-0.15678635, -0.26430085, -0.20042867, -0.16188341, -0.20967407,  
 -0.13326073, -0.2212627 , -0.17517729, -0.26357182, -0.13554077])

# The mean score on the testing dataset  
print(absolute(np.mean(score\_test)))

0.1921886660646745

# n\_estimators=5: 0.2559665333333369  
# n\_estimators=50: 0.2160986019999779  
# n\_estimators=100: 0.21331101666667096  
# n\_estimators=150: 0.21782862814814297  
# n\_estimators=200: 0.21533470437497257

# max\_depth = None: 0.20751561933334126  
# max\_depth = 3: 0.2148007247665628  
# max\_depth = 6: 0.2146223446041467  
# max\_depth = 9: 0.22022433083408877  
# max\_depth = 12: 0.2201754515429033  
# max\_depth = 15: 0.22376015583334646

# min\_samples\_leaf=1: 0.21134408183334746  
# min\_samples\_leaf=2: 0.20659695663632163  
# min\_samples\_leaf=3: 0.20035301895251195  
# min\_samples\_leaf=4: 0.19242615732935536  
# min\_samples\_leaf=5: 0.1954292744787311

# min\_samples\_split = 2: 0.21617311316668036  
# min\_samples\_split = 4: 0.2123402393112228  
# min\_samples\_split = 6: 0.21495300082832638  
# min\_samples\_split = 8: 0.20377708721079668  
# min\_samples\_split = 10: 0.2116725141899351  
# min\_samples\_split = 12: 0.19943883916078736

# max\_leaf\_nodes = None: 0.2116190046666701  
# max\_leaf\_nodes = 2: 0.2153581416663287  
# max\_leaf\_nodes = 25: 0.22592773370498379  
# max\_leaf\_nodes = 50: 0.22093479533334692  
# max\_leaf\_nodes = 75: 0.22400711800001  
# max\_leaf\_nodes = 100: 0.21663335200002193

# RF Feature ranking  
import pandas as pd  
from sklearn.pipeline import Pipeline  
from sklearn.ensemble import RandomForestRegressor

# Instantiation  
model = RandomForestRegressor(n\_estimators = 150, max\_depth = 12, min\_samples\_leaf = 2, min\_samples\_split = 2, max\_features = 2, max\_leaf\_nodes = 75)

lake\_rf2 = model.fit(X, y2)

# RF2 Feature ranking  
f\_list = list(X.columns)  
f\_importance = pd.Series(lake\_rf2.feature\_importances\_, index = f\_list).sort\_values(ascending = False)  
print(f\_importance)

P 0.285690  
SH 0.218307  
ST 0.137405  
ET 0.135904  
AT 0.122614  
SM 0.100079  
dtype: float64

############################################################################

# ARTIFICIAL NEURAL NETWORK  
# Importing libraries  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.wrappers.scikit\_learn import KerasRegressor  
from sklearn.model\_selection import KFold  
from sklearn.pipeline import Pipeline  
import warnings;warnings.simplefilter('ignore')

seed = 7

# Definition of metrics

# The coefficient of determination  
def r\_square(y1\_train, predtrain\_nn1):  
 from keras import backend as K  
 SS\_res = K.sum(K.square(y1\_train - predtrain\_nn1))  
 SS\_tot = K.sum(K.square(y1\_train - K.mean(y1\_train)))  
 return (1 - SS\_res/(SS\_tot + K.epsilon()))

print(r\_square)

<function r\_square at 0x000001EFD67F11E0>

# The mean absolute error  
def mae(y1\_train, predtrain\_nn1):  
 from keras import backend  
 return backend.mean(backend.absolute(y1\_train - predtrain\_nn1), axis = -1)

# The mean squared error  
def mse(y1\_train, predtrain\_nn1):  
 from keras import backend  
 return backend.mean(backend.square(y1\_train - predtrain\_nn1), axis = -1)

print(mse)

<function mse at 0x000001EFDECC27B8>

# The root mean square error  
def rmse(y1\_train, predtrain\_nn1):  
 from keras import backend  
 return backend.sqrt(backend.mean(backend.square(y1\_train - predtrain\_nn1), axis = -1))

print(rmse)

<function rmse at 0x0000020979E989D8>

# Remote sensing lake level as output

# Relu

# Model definition  
model = Sequential()  
model.add(Dense(128, input\_dim = 6, activation = 'relu', kernel\_initializer = 'normal'))  
model.add(Dense(64, activation = 'relu'))  
# Ouput layer  
model.add(Dense(1, activation = 'linear'))  
# Compiling the model  
model.compile(loss = 'mean\_absolute\_error', optimizer='adam', metrics=['mae', 'mse', rmse, r\_square])  
#model.compile(loss = 'mean\_absolute\_error', optimizer='adam')  
model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
dense\_1 (Dense) (None, 128) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_2 (Dense) (None, 64) 8256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_3 (Dense) (None, 1) 65   
=================================================================  
Total params: 9,217  
Trainable params: 9,217  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Fitting the model  
history1 = model.fit(X\_train, y1\_train, validation\_split = 0.25, epochs = 100, batch\_size = 10, verbose = 0)

hist = pd.DataFrame(history1.history)  
hist['epochs'] = history1.epoch

hist.head()

val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
0 280.223189 280.223189 78525.270833 280.223189   
1 278.840888 278.840888 77752.802951 278.840888   
2 276.259199 276.259199 76321.048611 276.259193   
3 271.609433 271.609433 73778.490451 271.609433   
4 263.933153 263.933153 69683.107639 263.933139   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error \  
0 -618745.881944 280.832099 280.832099 78867.015625   
1 -612706.288194 279.721094 279.721094 78244.554398   
2 -601486.062500 277.838616 277.838616 77196.215856   
3 -581597.645833 274.329948 274.329948 75264.350116   
4 -549529.934028 268.359497 268.359497 72040.645833   
  
 rmse r\_square epochs   
0 280.832101 -426224.789352 0   
1 279.721096 -419366.302083 1   
2 277.838612 -530000.118056 2   
3 274.329941 -657495.885417 3   
4 268.359493 -380601.369213 4

hist.tail()

val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
95 1.113645 1.113645 1.946241 1.113645   
96 1.023851 1.023851 1.744490 1.023851   
97 1.527359 1.527359 4.357307 1.527359   
98 2.012644 2.012644 5.251811 2.012644   
99 0.821499 0.821499 0.957353 0.821499   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error rmse \  
95 -13.454569 1.208261 1.208261 2.831450 1.208261   
96 -13.341105 0.798871 0.798871 1.234389 0.798871   
97 -36.823327 1.219518 1.219518 2.484378 1.219518   
98 -44.110157 1.438601 1.438601 3.145895 1.438601   
99 -7.213455 1.356153 1.356153 3.669086 1.356153   
  
 r\_square epochs   
95 -15.384341 95   
96 -5.999598 96   
97 -11.270808 97   
98 -16.591206 98   
99 -22.783783 99

#history1.history

# Finding the mean value of the metrics  
import numpy as np

# Mean val\_loss  
avg\_val\_loss = np.mean(history1.history['val\_loss'])  
avg\_val\_loss

37.53076197604338

# Mean val mae  
avg\_val\_mae = np.mean(history1.history['val\_mean\_absolute\_error'])  
avg\_val\_mae

37.53076197604338

# Mean val mse  
avg\_val\_mse = np.mean(history1.history['val\_mean\_squared\_error'])  
avg\_val\_mse

6676.9086224637085

# Mean val rmse  
avg\_val\_rmse = np.mean(history1.history['val\_rmse'])  
avg\_val\_rmse

37.5307614984115

# 0.86408407755090893122458229808173 from the calculator

# Mean val r\_square  
avg\_val\_rsq = np.mean(history1.history['val\_r\_square'])  
avg\_val\_rsq

-53207.59811586049

# Mean loss  
avg\_loss = np.mean(history1.history['loss'])  
avg\_loss

39.209825106594295

# mean mae  
avg\_mae = np.mean(history1.history['mean\_absolute\_error'])  
avg\_mae

39.209825106594295

# Mean mse  
avg\_mse = np.mean(history1.history['mean\_squared\_error'])  
avg\_mse

7170.893647646286

# Mean rmse  
avg\_rmse = np.mean(history1.history['rmse'])  
avg\_rmse

39.209824883805375

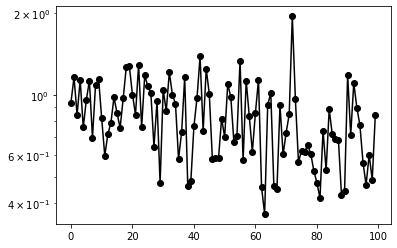
# 0.83390341333486925594814814687701 from the calculator

# mean rsq  
avg\_rsq = np.mean(history1.history['r\_square'])  
avg\_rsq

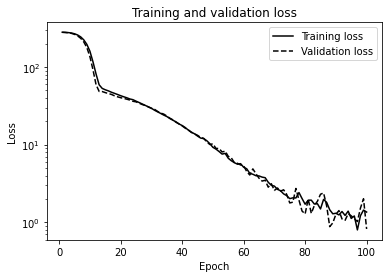
-44495.60819525847

# Plotting the training and testing accuracy and loss at each epoch  
from matplotlib import pyplot as plt  
#import matplotlib.pyplot as plt

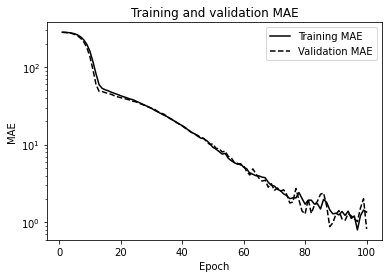
model\_ = model.fit(X\_train, y1\_train, validation\_split = 0.25, epochs = 100, batch\_size = 10, verbose = 0)  
plt.plot(list(model\_.history.values())[0], 'k-o')  
plt.yscale('log')  
plt.show()



# Relu  
loss1 = history1.history['loss']  
val\_loss1 = history1.history['val\_loss']  
epochs = range(1, len(loss1)+1)  
plt.plot(epochs, loss1, 'k-', label = 'Training loss')  
plt.plot(epochs, val\_loss1, 'k--', label = 'Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.yscale('log')  
plt.legend(loc='best')  
plt.show()

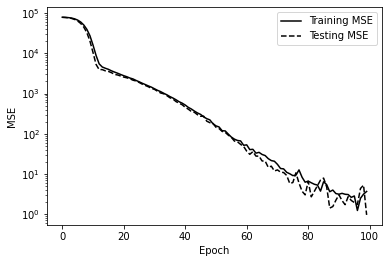


# relu  
acc1 = history1.history['mean\_absolute\_error']  
val\_acc1 = history1.history['val\_mean\_absolute\_error']  
plt.plot(epochs, acc1, 'k-', label = 'Training MAE')  
plt.plot(epochs, val\_acc1, 'k--', label = 'Validation MAE')  
plt.title('Training and validation MAE')  
plt.xlabel('Epoch')  
plt.ylabel('MAE')  
plt.yscale('log')  
plt.legend(loc = 'best')  
plt.show()



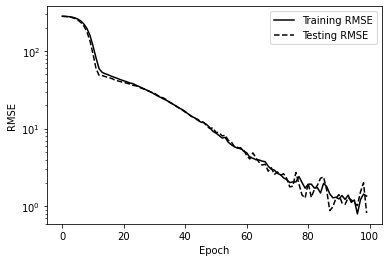
# Plotting the mse  
import matplotlib.pyplot as plt  
plt.plot(history1.history['mean\_squared\_error'], 'k-')  
plt.plot(history1.history['val\_mean\_squared\_error'], 'k--')  
plt.ylabel('MSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MSE', 'Testing MSE'], loc = 'best')

<matplotlib.legend.Legend at 0x2097d01f320>



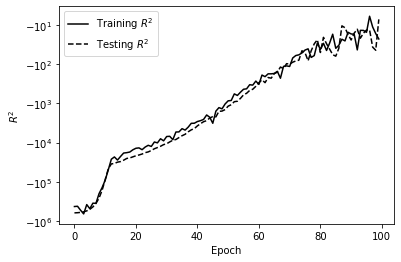
# Plotting the rmse training curve  
plt.plot(history1.history['rmse'], 'k-')  
plt.plot(history1.history['val\_rmse'], 'k--')  
plt.ylabel('RMSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training RMSE', 'Testing RMSE'], loc = 'best')

<matplotlib.legend.Legend at 0x2097e70f860>



# Plotting the r\_squared  
import pylab  
plt.plot(history1.history['r\_square'], 'k-')  
plt.plot(history1.history['val\_r\_square'], 'k--')  
plt.ylabel('$R^2$')  
pylab.yscale('symlog', linthreshy = 1)  
plt.xlabel('Epoch')  
plt.legend(['Training $R^2$', 'Testing $R^2$'], loc = 'best')

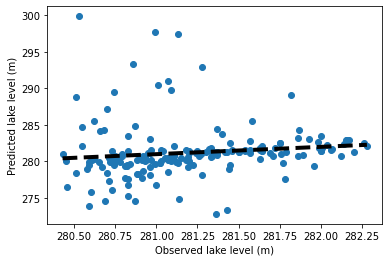
<matplotlib.legend.Legend at 0x2097f88af98>



# Prediction on the training dataset  
predtrain\_nn1 = model.predict(X\_train)

## Plotting the scatter plot of the test correlation dataset  
#Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_train, predtrain\_nn1)  
ax.plot([y1\_train.min(),y1\_train.max()], [y1\_train.min(), y1\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')

Text(0, 0.5, 'Predicted lake level (m)')



# Computing the covariance between the observed and predicted values   
from numpy import cov  
import numpy as np  
#covtrain\_nn1 = cov(y1\_train, predtrain\_nn1)  
#print(covtrain\_nn1)

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
#cortrain\_nn1 = pearsonr(y1\_train, predtrain\_nn1)  
#print(cortrain\_nn1)

# To save the predicted data on the drive  
import numpy  
numpy.savetxt('E:/Lake Level/DL/reluTrainpredLLR.csv', predtrain\_nn1, delimiter = ',')

# Model evaluation

# The train coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn1 coefficient of determination is:', r2\_score(y1\_train, predtrain\_nn1))

The nn1 coefficient of determination is: -4.429029897610583

# The training MSE  
   
print('The RS Data nn train MSE is:',mean\_squared\_error(y1\_train, predtrain\_nn1))

The RS Data nn train MSE is: 1.1608501025393143

# The training RMSE  
import numpy as np  
   
print('The RS Data nn train RMSE is:',np.sqrt(mean\_squared\_error(y1\_train, predtrain\_nn1)))

The RS Data nn train RMSE is: 1.0774275393451358

# The training MAE  
   
print('The RS Data nn train MAE is:',mean\_absolute\_error(y1\_train, predtrain\_nn1))

The RS Data nn train MAE is: 0.8803556993272593

# The training explained variance score  
   
print('The RS Data nn train evs is:',explained\_variance\_score(y1\_train, predtrain\_nn1))

The RS Data nn train evs is: -2.149570152500808

# Prediction on testing data  
pred\_nn1 = model.predict(X\_test).flatten()  
  
print(pred\_nn1[:5])

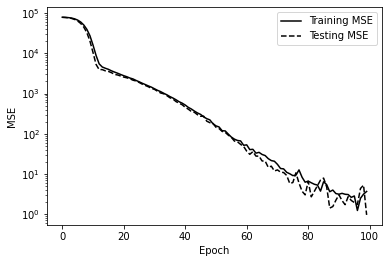
[281.14578 291.58548 280.82947 280.75262 281.54504]

# To save the predicted data on the drive  
#numpy.savetxt('E:/Lake Level/DL/tanhdlpredLLR.csv', pred\_nn1, delimiter = ',')

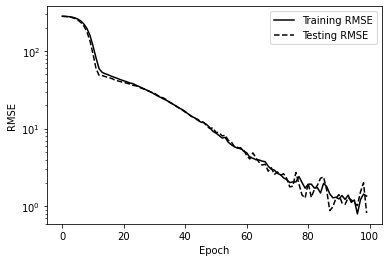
import numpy as np  
y1\_test = np.array(y1\_test)  
print(y1\_test[:5])

[281.99 281.12 281.12 281.03 281.62]

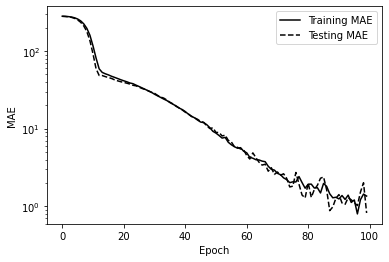
# Plotting the mse  
import matplotlib.pyplot as plt  
plt.plot(history1.history['mean\_squared\_error'], 'k-')  
plt.plot(history1.history['val\_mean\_squared\_error'], 'k--')  
plt.ylabel('MSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MSE', 'Testing MSE'], loc = 'best')  
plt.show()



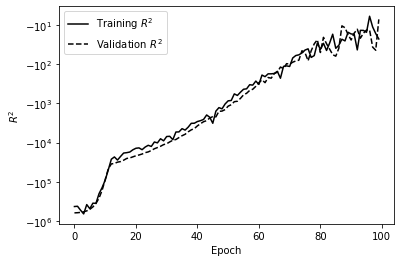
# Plotting the rmse training curve  
plt.plot(history1.history['rmse'], 'k-')  
plt.plot(history1.history['val\_rmse'], 'k--')  
plt.ylabel('RMSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training RMSE', 'Testing RMSE'], loc = 'best')  
plt.show()



# Plotting the mae  
plt.plot(history1.history['mean\_absolute\_error'], 'k-')  
plt.plot(history1.history['val\_mean\_absolute\_error'], 'k--')  
plt.yscale('log')  
plt.ylabel('MAE')  
plt.xlabel('Epoch')  
plt.legend(['Training MAE', 'Testing MAE'], loc = 'best')  
plt.show()

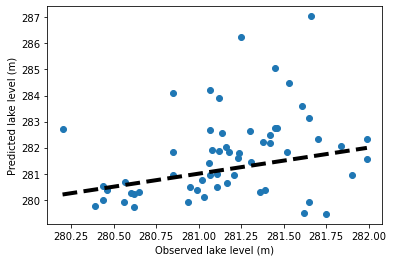


# Plotting the r\_squared  
import pylab  
plt.plot(history1.history['r\_square'], 'k-')  
plt.plot(history1.history['val\_r\_square'], 'k--')  
pylab.yscale('symlog')  
plt.ylabel('$R^2$')  
plt.xlabel('Epoch')  
plt.legend(['Training $R^2$', 'Validation $R^2$'], loc = 'best')  
plt.show()



# relu   
# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_nn1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')

Text(0, 0.5, 'Predicted lake level (m)')



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtestnn1 = cov(y1\_test, pred\_nn1)  
print(covtestnn1)

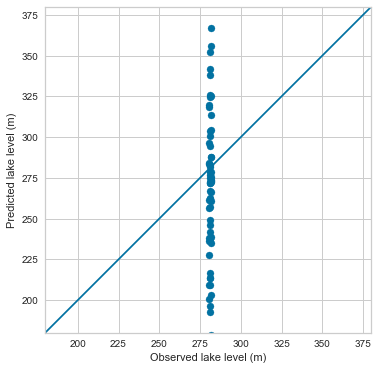
[[0.1852037 0.1737918 ]  
 [0.1737918 6.04620299]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestnn1 = pearsonr(y1\_test, pred\_nn1)  
print(cortestnn1)

(0.16423400635221244, 0.20986932024357804)

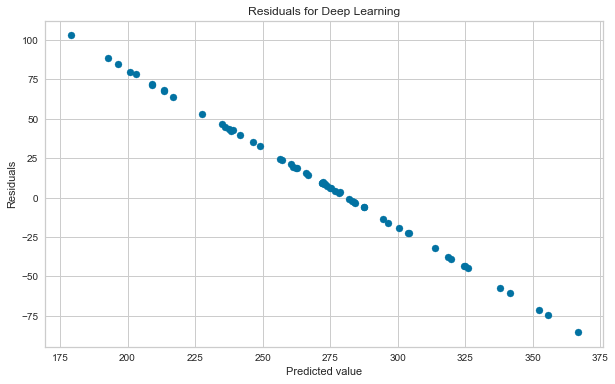
# Relu   
plt.axes(aspect='equal')  
plt.scatter(y1\_test, pred\_nn1)  
lims = [180, 380]  
plt.xlim(lims)  
plt.ylim(lims)  
plt.xlabel('Observed lake level (m)')  
plt.ylabel('Predicted lake level (m)')  
plt.plot(lims, lims)

[<matplotlib.lines.Line2D at 0x1c489150dd8>]



# relu  
# Plotting the prediction errors  
from pandas import DataFrame, Series  
error = y1\_test - pred\_nn1  
plt.scatter(pred\_nn1, error)  
plt.xlabel('Predicted value')  
plt.ylabel('Residuals')  
plt.title('Residuals for Deep Learning')

Text(0.5, 1.0, 'Residuals for Deep Learning')



# Testing model evalutations Relu

# The test coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn1 coefficient of determination is:', r2\_score(y1\_test, pred\_nn1))

The nn1 coefficient of determination is: -7.921402422904675

# The The MSE  
   
print('The RS Data nn MSE is:',mean\_squared\_error(y1\_test, pred\_nn1))

The RS Data nn MSE is: 1.6247387972353944

# The RMSE  
   
print('The RS data nn1 rmse is:',np.sqrt(mean\_squared\_error(y1\_test, pred\_nn1)))

The RS data nn1 rmse is: 1.2746524221274576

# The MAE  
  
print('The RS data nn1 mae is:',mean\_absolute\_error(y1\_test, pred\_nn1))

The RS data nn1 mae is: 1.0192775065104172

# The nn1 Explained variance score  
print('nn1 evs is:', explained\_variance\_score(y1\_test, pred\_nn1))

nn1 evs is: -3.9369596440789643

# k-fold cross-validation  
from sklearn.model\_selection import RepeatedKFold, cross\_val\_score  
from sklearn.metrics import mean\_squared\_error  
from numpy import absolute

# Training score

y1score\_train = model.evaluate(X\_train, y1\_train, verbose = 2)

print(absolute(np.mean(y1score\_train)))

0.19415369722578274

# Testing score

y1score\_test = model.evaluate(X\_test, y1\_test, verbose = 2)

print(absolute(np.mean(y1score\_test)))

0.6573094717661538

# Sigmoid

# Model definition  
model = Sequential()  
model.add(Dense(128, input\_dim = 6, activation = 'sigmoid', kernel\_initializer = 'normal'))  
model.add(Dense(64, activation = 'sigmoid'))  
# Ouput layer  
model.add(Dense(1, activation = 'linear'))  
model.compile(loss='mean\_absolute\_error', optimizer='adam', metrics=['mae', 'mse', rmse, r\_square])  
model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
dense\_4 (Dense) (None, 128) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_5 (Dense) (None, 64) 8256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_6 (Dense) (None, 1) 65   
=================================================================  
Total params: 9,217  
Trainable params: 9,217  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Fitting the model  
history1 = model.fit(X\_train, y1\_train, validation\_split = 0.25, epochs = 100, verbose = 0)

hist = pd.DataFrame(history1.history)  
hist['epochs'] = history1.epoch

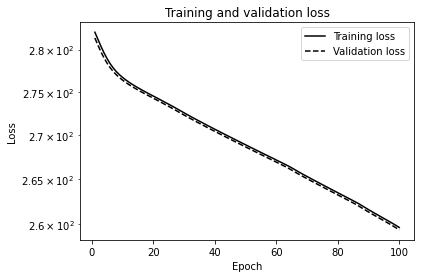
hist.head()

val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
0 281.373566 281.373566 79171.247569 281.373566   
1 280.512914 280.512914 78687.670313 280.512914   
2 279.709088 279.709088 78237.347049 279.709088   
3 278.991267 278.991267 77836.300174 278.991267   
4 278.368843 278.368843 77489.376562 278.368821   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error \  
0 -478947.706944 282.061398 282.061398 79558.879745   
1 -476022.544444 281.212299 281.212299 79080.626852   
2 -473298.497222 280.366885 280.366885 78605.857697   
3 -470872.395833 279.589737 279.589737 78170.671181   
4 -468773.636111 278.903176 278.903176 77787.259317   
  
 rmse r\_square epochs   
0 282.061398 -374880.579398 0   
1 281.212284 -400089.929630 1   
2 280.366885 -359475.466435 2   
3 279.589737 -377729.544444 3   
4 278.903176 -364030.122917 4

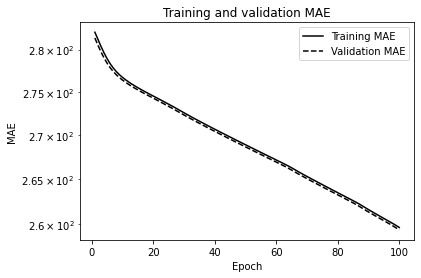
hist.tail()

val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
95 260.218297 260.218297 67713.721181 260.218297   
96 260.024196 260.024196 67612.759028 260.024196   
97 259.824162 259.824162 67508.760069 259.824162   
98 259.613347 259.613347 67399.257812 259.613325   
99 259.396681 259.396681 67286.810243 259.396681   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error \  
95 -409635.636111 260.470651 260.470651 67845.183681   
96 -409024.850000 260.278111 260.278111 67744.926331   
97 -408395.727778 260.082499 260.082499 67643.137789   
98 -407733.293056 259.879161 259.879161 67537.410359   
99 -407053.049306 259.665539 259.665539 67426.420081   
  
 rmse r\_square epochs   
95 260.470651 -335612.698611 95   
96 260.278111 -329539.655093 96   
97 260.082492 -314161.922685 97   
98 259.879161 -309445.176389 98   
99 259.665524 -388003.487037 99

# Plotting the raining and validation loss  
loss1 = history1.history['loss']  
val\_loss1 = history1.history['val\_loss']  
epochs = range(1, len(loss1)+1)  
plt.plot(epochs, loss1, 'k-', label = 'Training loss')  
plt.plot(epochs, val\_loss1, 'k--', label = 'Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.yscale('log')  
plt.legend(loc='best')  
plt.show()



# Plotting the raining and validation MAE  
acc1 = history1.history['mean\_absolute\_error']  
val\_acc1 = history1.history['val\_mean\_absolute\_error']  
plt.plot(epochs, acc1, 'k-', label = 'Training MAE')  
plt.plot(epochs, val\_acc1, 'k--', label = 'Validation MAE')  
plt.title('Training and validation MAE')  
plt.xlabel('Epoch')  
plt.ylabel('MAE')  
plt.yscale('log')  
plt.legend(loc = 'best')  
plt.show()



# Prediction on the training dataset  
predtrain\_nn1 = model.predict(X\_train)

# Model evaluation

# The train coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn1 coefficient of determination is:', r2\_score(y1\_train, predtrain\_nn1))

The nn1 coefficient of determination is: -314907.76914186805

# The training MSE  
   
print('The RS Data nn train MSE is:',mean\_squared\_error(y1\_train, predtrain\_nn1))

The RS Data nn train MSE is: 67334.65901702944

# The training RMSE  
import numpy as np  
   
print('The RS Data nn train RMSE is:',np.sqrt(mean\_squared\_error(y1\_train, predtrain\_nn1)))

The RS Data nn train RMSE is: 259.4892271695098

# The training MAE  
   
print('The RS Data nn train MAE is:',mean\_absolute\_error(y1\_train, predtrain\_nn1))

The RS Data nn train MAE is: 259.48881460147436

# The training explained variance score  
   
print('The RS Data nn train evs is:',explained\_variance\_score(y1\_train, predtrain\_nn1))

The RS Data nn train evs is: -0.001360945529618185

# Prediction on testing data  
pred\_nn1 = model.predict(X\_test).flatten()  
  
print(pred\_nn1[:30])

[21.709673 21.712801 21.710083 21.710678 21.70855 21.708542 21.711634  
 21.711098 21.712374 21.710941 21.712273 21.71123 21.711527 21.711712  
 21.709845 21.71133 21.710354 21.709894 21.71033 21.711369 21.710732  
 21.707466 21.712585 21.710827 21.711588 21.71272 21.709436 21.71019  
 21.71117 21.70973 ]

import numpy as np  
y1\_test = np.array(y1\_test)  
print(y1\_test[:5])

[281.99 281.12 281.12 281.03 281.62]

# The coefficient of determination

# The test coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn1 coefficient of determination is:', r2\_score(y1\_test, pred\_nn1))

The nn1 coefficient of determination is: -369601.5726301128

# Model evaluation

# The The MSE  
   
print('The RS Data nn MSE is:',mean\_squared\_error(y1\_test, pred\_nn1))

The RS Data nn MSE is: 67310.90145294003

# The RMSE  
   
print('The RS data nn1 rmse is:',np.sqrt(mean\_squared\_error(y1\_test, pred\_nn1)))

The RS data nn1 rmse is: 259.4434455771431

# The MAE  
  
print('The RS data nn1 mae is:',mean\_absolute\_error(y1\_test, pred\_nn1))

The RS data nn1 mae is: 259.4430939750671

# The nn1 Explained variance score  
print('nn1 evs is:', explained\_variance\_score(y1\_test, pred\_nn1))

nn1 evs is: -0.0017824384415707772

# k-fold cross-validation  
from sklearn.model\_selection import RepeatedKFold, cross\_val\_score  
from sklearn.metrics import mean\_squared\_error

# Training score  
from numpy import absolute

y1score\_train = model.evaluate(X\_train, y1\_train, verbose = 2)

print(absolute(np.mean(y1score\_train)))

52857.072233072926

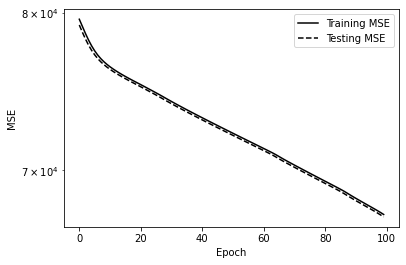
# Testing score

y1score\_test = model.evaluate(X\_test, y1\_test, verbose = 0)

print(absolute(np.mean(y1score\_test)))

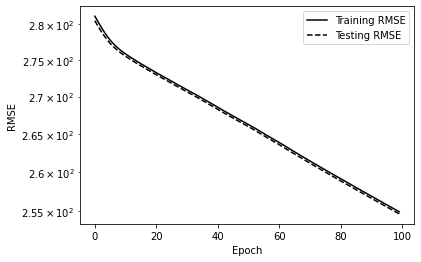
60513.64237263998

# Plotting the mse  
import matplotlib.pyplot as plt  
plt.plot(history1.history['mean\_squared\_error'], 'k-')  
plt.plot(history1.history['val\_mean\_squared\_error'], 'k--')  
plt.ylabel('MSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MSE', 'Testing MSE'], loc = 'best')  
plt.show()

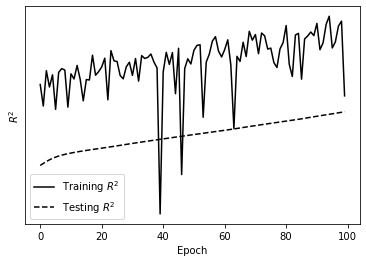


# Plotting the rmse training curve  
plt.plot(history1.history['rmse'], 'k-')  
plt.plot(history1.history['val\_rmse'], 'k--')  
plt.ylabel('RMSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training RMSE', 'Testing RMSE'], loc = 'best')

<matplotlib.legend.Legend at 0x1ca78679fd0>

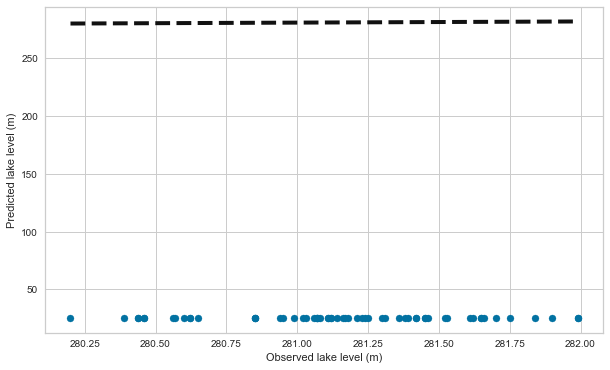


# Plotting the r\_squared  
import pylab  
pylab.yscale('symlog')  
plt.plot(history1.history['r\_square'], 'k-')  
plt.plot(history1.history['val\_r\_square'], 'k--')  
plt.ylabel('$R^2$')  
plt.xlabel('Epoch')  
plt.legend(['Training $R^2$', 'Testing $R^2$'], loc = 'best')  
plt.show()



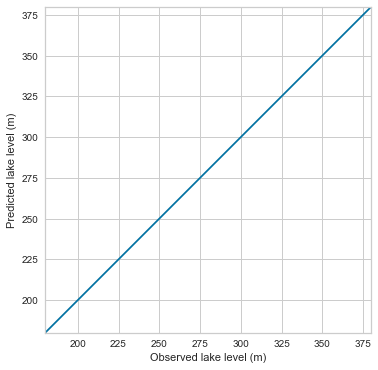
# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_nn1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')

Text(0, 0.5, 'Predicted lake level (m)')



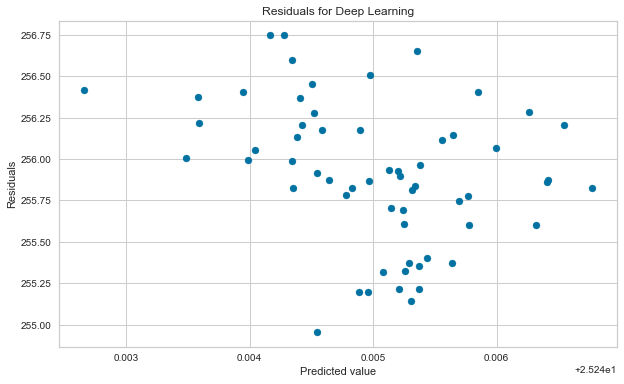
plt.axes(aspect='equal')  
plt.scatter(y1\_test, pred\_nn1)  
lims = [180, 380]  
plt.xlim(lims)  
plt.ylim(lims)  
plt.xlabel('Observed lake level (m)')  
plt.ylabel('Predicted lake level (m)')  
plt.plot(lims, lims)

[<matplotlib.lines.Line2D at 0x1c4872dce10>]



# Plotting the prediction errors  
from pandas import DataFrame, Series  
error = y1\_test - pred\_nn1  
plt.scatter(pred\_nn1, error)  
plt.xlabel('Predicted value')  
plt.ylabel('Residuals')  
plt.title('Residuals for Deep Learning')

Text(0.5, 1.0, 'Residuals for Deep Learning')



# Tanh

# Model definition  
model = Sequential()  
model.add(Dense(128, input\_dim = 6, activation = 'tanh', kernel\_initializer = 'normal'))  
model.add(Dense(64, activation = 'tanh'))  
# Ouput layer  
model.add(Dense(1, activation = 'linear'))  
model.compile(loss='mean\_absolute\_error', optimizer='adam', metrics=['mae', 'mse', rmse, r\_square])  
model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
dense\_7 (Dense) (None, 128) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_8 (Dense) (None, 64) 8256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_9 (Dense) (None, 1) 65   
=================================================================  
Total params: 9,217  
Trainable params: 9,217  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Fitting the model  
history1 = model.fit(X\_train, y1\_train, validation\_split = 0.25, epochs = 100, verbose = 0)

hist = pd.DataFrame(history1.history)  
hist['epochs'] = history1.epoch

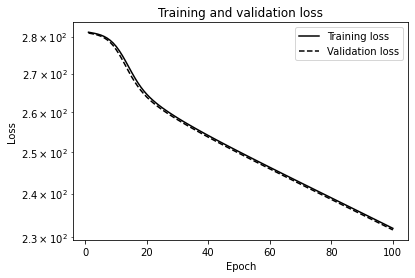
hist.head()

val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
0 280.979057 280.979057 78949.464236 280.979057   
1 280.831188 280.831188 78866.391319 280.831188   
2 280.646212 280.646212 78762.588542 280.646212   
3 280.421711 280.421711 78636.711806 280.421711   
4 280.124710 280.124710 78470.368056 280.124710   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error \  
0 -477617.991667 281.206300 281.206300 79077.280556   
1 -477116.765278 281.052894 281.052894 78991.053125   
2 -476492.244444 280.891465 280.891465 78900.402373   
3 -475733.827778 280.682896 280.682896 78783.402257   
4 -474730.236111 280.426476 280.426476 78639.674016   
  
 rmse r\_square epochs   
0 281.206300 -375449.253241 0   
1 281.052880 -370679.930093 1   
2 280.891457 -358320.029861 2   
3 280.682889 -372447.025926 3   
4 280.426469 -379898.656944 4

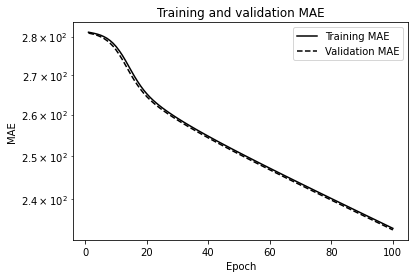
hist.tail()

val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
95 232.960115 232.960115 54270.580469 232.960115   
96 232.611077 232.611077 54108.083160 232.611077   
97 232.262330 232.262330 53945.958073 232.262330   
98 231.913970 231.913970 53784.261024 231.913970   
99 231.565908 231.565908 53622.933333 231.565908   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error \  
95 -328311.502083 233.318577 233.318577 54437.792679   
96 -327328.472222 232.969178 232.969178 54274.875550   
97 -326347.698611 232.620238 232.620238 54112.409491   
98 -325369.529861 232.271603 232.271603 53950.320544   
99 -324393.568056 231.923368 231.923368 53788.677141   
  
 rmse r\_square epochs   
95 233.318569 -293392.362963 95   
96 232.969178 -263125.021759 96   
97 232.620231 -280065.084954 97   
98 232.271600 -251430.316782 98   
99 231.923357 -257901.290509 99

# Plotting the raining and validation loss  
loss1 = history1.history['loss']  
val\_loss1 = history1.history['val\_loss']  
epochs = range(1, len(loss1)+1)  
plt.plot(epochs, loss1, 'k-', label = 'Training loss')  
plt.plot(epochs, val\_loss1, 'k--', label = 'Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.yscale('log')  
plt.legend(loc='best')  
plt.show()



# Plotting the raining and validation MAE  
acc1 = history1.history['mean\_absolute\_error']  
val\_acc1 = history1.history['val\_mean\_absolute\_error']  
plt.plot(epochs, acc1, 'k-', label = 'Training MAE')  
plt.plot(epochs, val\_acc1, 'k--', label = 'Validation MAE')  
plt.title('Training and validation MAE')  
plt.xlabel('Epoch')  
plt.ylabel('MAE')  
plt.yscale('log')  
plt.legend(loc = 'best')  
plt.show()



# Prediction on the training dataset  
predtrain\_nn1 = model.predict(X\_train)

# Model evaluation

# The train coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn1 coefficient of determination is:', r2\_score(y1\_train, predtrain\_nn1))

The nn1 coefficient of determination is: -250981.00700753246

# The training MSE  
   
print('The RS Data nn train MSE is:',mean\_squared\_error(y1\_train, predtrain\_nn1))

The RS Data nn train MSE is: 53665.66293886991

# The training RMSE  
import numpy as np  
   
print('The RS Data nn train RMSE is:',np.sqrt(mean\_squared\_error(y1\_train, predtrain\_nn1)))

The RS Data nn train RMSE is: 231.65850500007528

# The training MAE  
   
print('The RS Data nn train MAE is:',mean\_absolute\_error(y1\_train, predtrain\_nn1))

The RS Data nn train MAE is: 231.65804281616207

# The training explained variance score  
   
print('The RS Data nn train evs is:',explained\_variance\_score(y1\_train, predtrain\_nn1))

The RS Data nn train evs is: -0.0014718036854275418

# Prediction on testing data  
pred\_nn1 = model.predict(X\_test).flatten()  
  
print(pred\_nn1[:5])

[49.54089 49.5428 49.542095 49.540287 49.537983]

import numpy as np  
y1\_test = np.array(y1\_test)  
print(y1\_test[:5])

[281.99 281.12 281.12 281.03 281.62]

# The test coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn1 coefficient of determination is:', r2\_score(y1\_test, pred\_nn1))

The nn1 coefficient of determination is: -294559.1674806951

# Model evaluation

# The The MSE  
   
print('The RS Data nn MSE is:',mean\_squared\_error(y1\_test, pred\_nn1))

The RS Data nn MSE is: 53644.4058388548

# The RMSE  
   
print('The RS data nn1 rmse is:',np.sqrt(mean\_squared\_error(y1\_test, pred\_nn1)))

The RS data nn1 rmse is: 231.61262020635837

# The MAE  
  
print('The RS data nn1 mae is:',mean\_absolute\_error(y1\_test, pred\_nn1))

The RS data nn1 mae is: 231.6122261123657

# The nn1 Explained variance score  
print('nn1 evs is:', explained\_variance\_score(y1\_test, pred\_nn1))

nn1 evs is: -0.0024004185218529095

# k-fold cross-validation  
from sklearn.model\_selection import RepeatedKFold, cross\_val\_score  
from sklearn.metrics import mean\_squared\_error

# Training score  
from numpy import absolute

y1score\_train = model.evaluate(X\_train, y1\_train, verbose = 2)

print(absolute(np.mean(y1score\_train)))

42111.96876607259

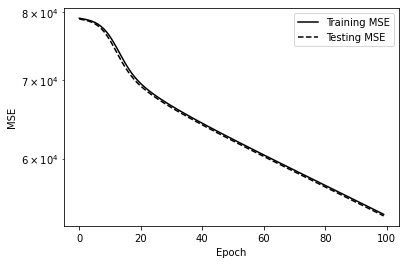
# Testing score

y1score\_test = model.evaluate(X\_test, y1\_test, verbose = 2)

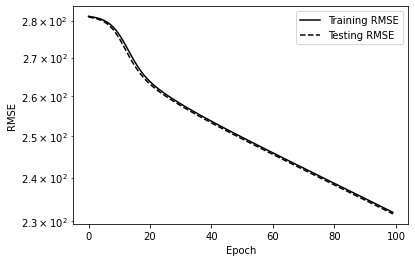
print(absolute(np.mean(y1score\_test)))

48212.25480326335

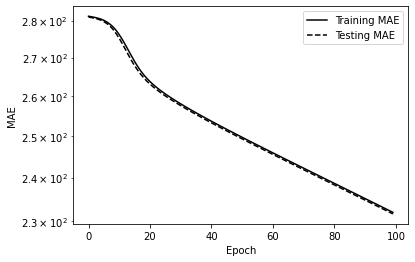
# Plotting the mse  
import matplotlib.pyplot as plt  
plt.plot(history1.history['mean\_squared\_error'], 'k-')  
plt.plot(history1.history['val\_mean\_squared\_error'], 'k--')  
plt.ylabel('MSE')  
plt.xlabel('Epoch')  
plt.yscale('log')  
plt.legend(['Training MSE', 'Testing MSE'], loc = 'best')  
plt.show()



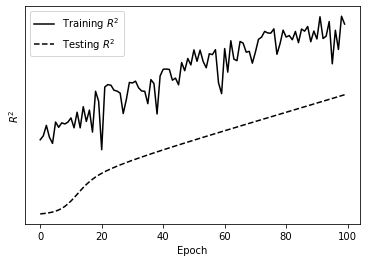
# Plotting the rmse training curve  
plt.plot(history1.history['rmse'], 'k-')  
plt.plot(history1.history['val\_rmse'], 'k--')  
plt.ylabel('RMSE')  
plt.xlabel('Epoch')  
plt.yscale('log')  
plt.legend(['Training RMSE', 'Testing RMSE'], loc = 'best')  
plt.show()



# Plotting the mae  
plt.plot(history1.history['mean\_absolute\_error'], 'k-')  
plt.plot(history1.history['val\_mean\_absolute\_error'], 'k--')  
plt.ylabel('MAE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MAE', 'Testing MAE'], loc = 'best')  
plt.show()

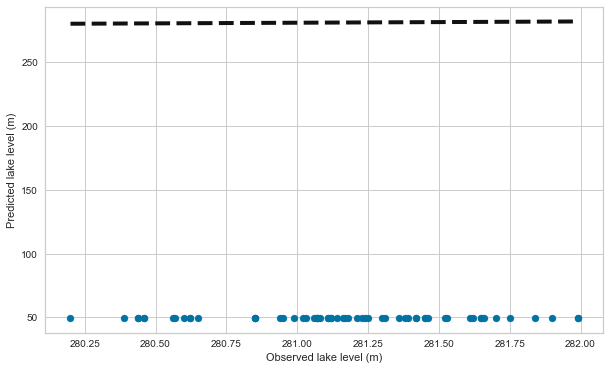


# Plotting the r\_squared  
import pylab  
pylab.yscale('symlog')  
plt.plot(history1.history['r\_square'], 'k-')  
plt.plot(history1.history['val\_r\_square'], 'k--')  
plt.ylabel('$R^2$')  
plt.xlabel('Epoch')  
plt.legend(['Training $R^2$', 'Testing $R^2$'], loc = 'best')  
plt.show()



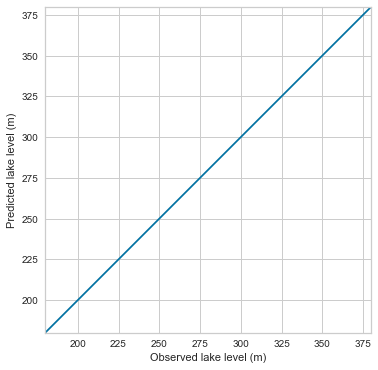
# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_nn1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')

Text(0, 0.5, 'Predicted lake level (m)')



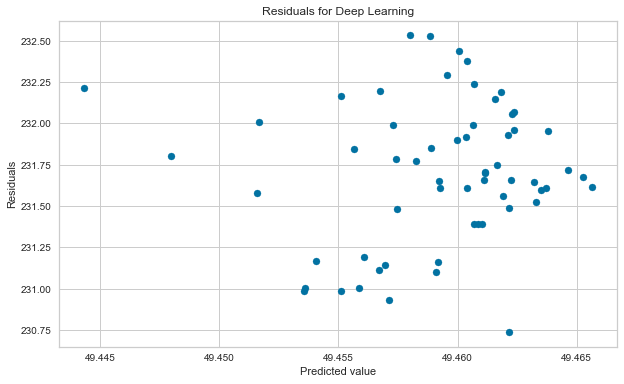
plt.axes(aspect='equal')  
plt.scatter(y1\_test, pred\_nn1)  
lims = [180, 380]  
plt.xlim(lims)  
plt.ylim(lims)  
plt.xlabel('Observed lake level (m)')  
plt.ylabel('Predicted lake level (m)')  
plt.plot(lims, lims)

[<matplotlib.lines.Line2D at 0x1c488896c88>]



# Plotting the prediction errors  
from pandas import DataFrame, Series  
error = y1\_test - pred\_nn1  
plt.scatter(pred\_nn1, error)  
plt.xlabel('Predicted value')  
plt.ylabel('Residuals')  
plt.title('Residuals for Deep Learning')

Text(0.5, 1.0, 'Residuals for Deep Learning')



##########################################################################################################################

# The coefficient of determination  
  
#print('The nn1 coefficient of determination on the training datset is:', model1.score(X\_train\_scaled, y1\_train))

# The test coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn1 coefficient of determination is:', r2\_score(y1\_test, pred\_nn1))

The nn1 coefficient of determination is: -340229.5185838571

# relu -217036.68542486447

# sigmoid -245838.1132805952

# tanh -340229.5185838571

# Model evaluation

# The The MSE  
   
print('The RS Data nn MSE is:',mean\_squared\_error(y1\_test, pred\_nn1))

The RS Data nn MSE is: 61961.75190208848

# Relu mse 44771.474964457826

# Sigmoid mse 44771.474964457826

# Tanh mse 61961.75190208848

# The RMSE  
   
print('The RS data nn1 rmse is:',np.sqrt(mean\_squared\_error(y1\_test, pred\_nn1)))

The RS data nn1 rmse is: 248.92117608208522

# Relu rmse 211.59271009289952

# Sigmoid rmse 211.59271009289952

# tanh rmse 248.92117608208522

# The MAE  
  
print('The RS data nn1 mae is:',mean\_absolute\_error(y1\_test, pred\_nn1))

The RS data nn1 mae is: 248.91164503860472

# relu mae 192.0717692228953

# sigmoid mae 192.0717692228953

# tanh mae 248.91164503860472

# The nn1 Explained variance score  
print('nn1 evs is:', explained\_variance\_score(y1\_test, pred\_nn1))

nn1 evs is: -25.053948768254386

# relu evs -45181.09856544394

# sigmoid evs -45181.09856544394

# tanh evs -25.053948768254386

# k-fold cross-validation  
from sklearn.model\_selection import RepeatedKFold, cross\_val\_score  
from sklearn.metrics import mean\_squared\_error

# Training score  
from numpy import absolute

y1score\_train = model.evaluate(X\_train, y1\_train, verbose = 2)

print(absolute(np.mean(y1score\_train)))

36080.01031155056

# relu 31604.847605048282

# sigmoid 45817.016223415805

# tanh 36080.01031155056

# Testing score

y1score\_test = model.evaluate(X\_test, y1\_test, verbose = 0)

print(absolute(np.mean(y1score\_test)))

43154.55840983073

# relu 40305.086005249024

# sigmoid 54720.42006673177

# tanh 43154.55840983073

############################################################################################################################

######CV METHOD 2 #########################  
seed = 7

# Deep Learning modeling  
# Importing libraries  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.wrappers.scikit\_learn import KerasRegressor  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import KFold  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import Pipeline  
import warnings;warnings.simplefilter('ignore')

# On the training dataset  
# Model definition  
def model():  
 # Model creation  
 model = Sequential()  
 # model.add(Dense(128, input\_dim = 6, kernel\_initializer = 'normal', activation = 'relu'))  
 model.add(Dense(6, input\_dim = 6, kernel\_initializer = 'normal', activation = 'relu'))  
 # Ouput layer  
 model.add(Dense(1, activation = 'linear', kernel\_initializer = 'normal'))  
 # Model compilation  
 model.compile(loss='mean\_absolute\_error', optimizer='adam')  
 return model  
 #model.summary()  
# Evaluate the model with standardized dataset  
estimators = []  
estimators.append(('standardize',StandardScaler()))  
estimators.append(('mlp', KerasRegressor(build\_fn = model, epochs = 100, batch\_size = 10, verbose = 0 )))  
pipeline = Pipeline(estimators)  
kfold = KFold(n\_splits = 10)  
results = cross\_val\_score(pipeline, X\_train, y1\_train, cv = kfold)  
print('Standardized: %.2f, %.2f, %.2f MAE'% (results.mean(), results.var(), results.std()))

Standardized: -225.83, 170.09, 13.04 MAE

# Relu: -49901.68, 25303034.31, 5030.21 || -222.72, 96.40, 9.82 MAE

# Sigmoid: -54608.81, 34084950.68, 5838.23 MSE || -222.53, 262.42, 16.20 MAE

# Tanh: -54461.47, 22412398.30, 4734.17 MSE || -225.83, 170.09, 13.04 MAE

# ON THE TESTING DATASET

# On the testing dataset  
# Model definition  
def model():  
 # Model creation  
 model = Sequential()  
 #model.add(Dense(128, input\_dim = 6, kernel\_initializer = 'normal', activation = 'relu'))  
 model.add(Dense(6, input\_dim = 6, kernel\_initializer = 'normal', activation = 'relu'))  
 # Ouput layer  
 model.add(Dense(1, activation = 'linear', kernel\_initializer = 'normal'))  
 # Model compilation  
 model.compile(loss='mean\_absolute\_error', optimizer='adam')  
 return model  
 #model.summary()  
# Evaluate the model with standardized dataset  
estimators = []  
estimators.append(('standardize',StandardScaler()))  
estimators.append(('mlp', KerasRegressor(build\_fn = model, epochs = 100, batch\_size = 10, verbose = 0 )))  
pipeline = Pipeline(estimators)  
kfold = KFold(n\_splits = 10)  
results = cross\_val\_score(pipeline, X\_test, y1\_test, cv = kfold)  
print('Standardized: %.2f, %.2f, %.2f MAE'% (results.mean(), results.var(), results.std()))

Standardized: -273.15, 1.85, 1.36 MAE

# Relu: -74062.03, 506650.23, 711.79 MSE || -272.54, 2.78, 1.67 MAE With batch\_size = 5

# Sigmoid: -44831.77, 322405.71, 567.81 MSE || -208.14, 1.06, 1.03 MAE

# Tanh: -41122.68, 14395.20, 119.98 MSE || -198.35, 0.11, 0.33 MAE

# Ground truth lake level as output

# Metrics definitions  
# The coefficient of determination  
def r\_square(y2\_train, predtrain\_nn2):  
 from keras import backend as K  
 SS\_res = K.sum(K.square(y2\_train - predtrain\_nn2))  
 SS\_tot = K.sum(K.square(y2\_train - K.mean(y2\_train)))  
 return (1 - SS\_res/(SS\_tot + K.epsilon()))  
# The mean absolute error  
def mae(y2\_train, predtrain\_nn2):  
 from keras import backend  
 return backend.mean(backend.absolute(y2\_train - predtrain\_nn2), axis = -1)  
# The mean squared error  
def mse(y2\_train, predtrain\_nn2):  
 from keras import backend  
 return backend.mean(backend.square(y2\_train - predtrain\_nn2), axis = -1)  
# The root mean square error  
def rmse(y2\_train, predtrain\_nn2):  
 from keras import backend  
 return backend.sqrt(backend.mean(backend.square(y2\_train - predtrain\_nn2), axis = -1))

# ReLu Activation Function

# Model definition  
model = Sequential()  
model.add(Dense(128, input\_dim = 6, activation = 'relu'))  
model.add(Dense(64, activation = 'relu'))  
# Ouput layer  
model.add(Dense(1, activation = 'linear'))  
model.compile(loss='mean\_absolute\_error', optimizer='adam', metrics=['mae', 'mse', rmse, r\_square])  
model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
dense\_4 (Dense) (None, 128) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_5 (Dense) (None, 64) 8256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_6 (Dense) (None, 1) 65   
=================================================================  
Total params: 9,217  
Trainable params: 9,217  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Fitting the model  
history2 = model.fit(X\_train, y2\_train, validation\_split = 0.25, epochs = 100, batch\_size = 10, verbose = 0)

hist = pd.DataFrame(history2.history)  
hist['epochs'] = history2.epoch

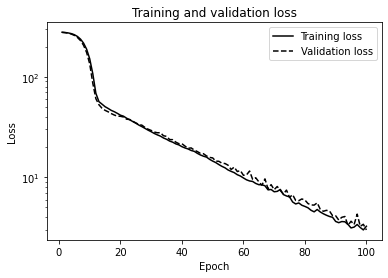
hist.head()

val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
0 277.541724 277.541724 77029.921875 277.541718   
1 275.587911 275.587911 75949.967014 275.587911   
2 272.404060 272.404060 74207.644097 272.404060   
3 267.182895 267.182895 71398.049479 267.182895   
4 259.023261 259.023261 67125.238715 259.023248   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error \  
0 -239005.102431 278.375961 278.375961 77493.865741   
1 -235703.385417 276.701317 276.701317 76565.099537   
2 -230369.140625 274.055012 274.055012 75109.880498   
3 -221777.425347 269.790935 269.790935 72797.656829   
4 -208708.449653 263.100337 263.100337 69245.361111   
  
 rmse r\_square epochs   
0 278.375958 -201482.950231 0   
1 276.701317 -177564.046875 1   
2 274.055010 -317207.839699 2   
3 269.790931 -182839.292245 3   
4 263.100333 -178121.888889 4

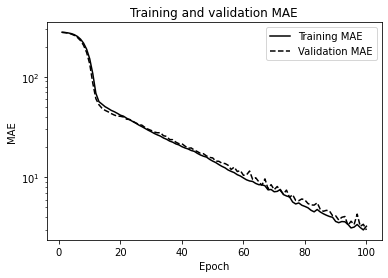
hist.tail()

val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
95 3.385235 3.385235 21.411135 3.385235   
96 4.292761 4.292761 31.228913 4.292761   
97 3.248440 3.248440 18.031299 3.248440   
98 3.401204 3.401204 19.766648 3.401204   
99 2.990760 2.990760 16.205198 2.990760   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error rmse \  
95 -69.356941 3.190503 3.190503 20.951289 3.190503   
96 -103.587511 3.385249 3.385249 22.872526 3.385249   
97 -55.399370 3.159253 3.159253 21.337602 3.159253   
98 -63.232630 2.989530 2.989530 18.514620 2.989530   
99 -51.983429 3.252233 3.252233 19.225769 3.252233   
  
 r\_square epochs   
95 -53.169526 95   
96 -51.906961 96   
97 -54.336813 97   
98 -46.050917 98   
99 -42.324428 99

# Relu  
# Plotting the training and validation loss  
loss2 = history2.history['loss']  
val\_loss2 = history2.history['val\_loss']  
epochs = range(1, len(loss2)+1)  
plt.plot(epochs, loss2, 'k-', label = 'Training loss')  
plt.plot(epochs, val\_loss2, 'k--', label = 'Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epoch')  
plt.yscale('log')  
plt.ylabel('Loss')  
plt.legend(loc='best')  
plt.show()

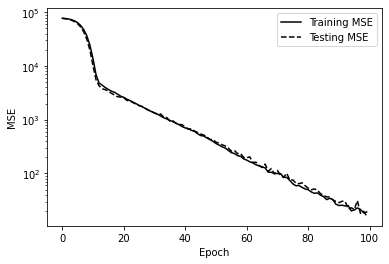


# relu  
# Plotting the training and validation MAE  
acc2 = history2.history['mean\_absolute\_error']  
val\_acc2 = history2.history['val\_mean\_absolute\_error']  
plt.plot(epochs, acc2, 'k-', label = 'Training MAE')  
plt.plot(epochs, val\_acc2, 'k--', label = 'Validation MAE')  
plt.title('Training and validation MAE')  
plt.xlabel('Epoch')  
plt.yscale('log')  
plt.ylabel('MAE')  
plt.legend(loc = 'best')  
plt.show()

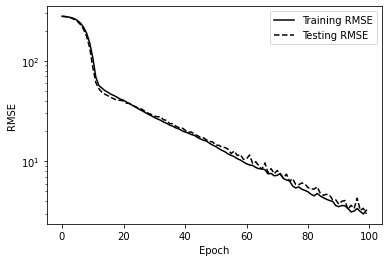


# Plotting the mse  
import matplotlib.pyplot as plt  
plt.plot(history2.history['mean\_squared\_error'], 'k-')  
plt.plot(history2.history['val\_mean\_squared\_error'], 'k--')  
plt.ylabel('MSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MSE', 'Testing MSE'], loc = 'best')

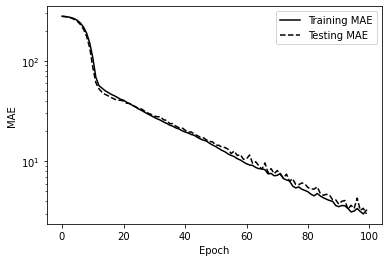
<matplotlib.legend.Legend at 0x1e4c22b34a8>



# Plotting the rmse training curve  
plt.plot(history2.history['rmse'], 'k-')  
plt.plot(history2.history['val\_rmse'], 'k--')  
plt.ylabel('RMSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training RMSE', 'Testing RMSE'], loc = 'best')  
plt.show()

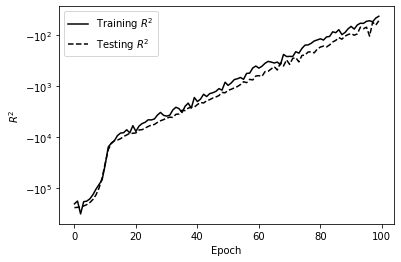


# Plotting the mae  
plt.plot(history2.history['mean\_absolute\_error'], 'k-')  
plt.plot(history2.history['val\_mean\_absolute\_error'], 'k--')  
plt.ylabel('MAE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MAE', 'Testing MAE'], loc = 'best')  
plt.show()



# Plotting the r\_squared  
import pylab  
pylab.yscale('symlog')  
plt.plot(history2.history['r\_square'], 'k-')  
plt.plot(history2.history['val\_r\_square'], 'k--')  
plt.ylabel('$R^2$')  
plt.xlabel('Epoch')  
plt.legend(['Training $R^2$', 'Testing $R^2$'], loc = 'best')

<matplotlib.legend.Legend at 0x1e4c635da58>

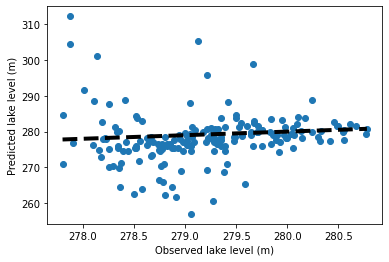


#Prediction on the training data

predtrain\_nn2 = model.predict(X\_train)

## Plotting the scatter plot of the test correlation dataset  
#Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_train, predtrain\_nn2)  
ax.plot([y2\_train.min(),y2\_train.max()], [y2\_train.min(), y2\_train.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')

Text(0, 0.5, 'Predicted lake level (m)')



# Computing the covariance between the observed and predicted values   
from numpy import cov  
#covtestnn2 = cov(y2\_train, predtrain\_nn2)  
#print(covtestnn2)

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
#cortestnn2 = pearsonr(y2\_train, predtrain\_nn2)  
#print(cortestnn2)

# To save the predicted data on the drive  
numpy.savetxt('E:/Lake Level/DL/ReluTrainpredLLG.csv', predtrain\_nn2, delimiter = ',')

# Saving observed LL\_R Train  
# To save the predicted data on the drive  
numpy.savetxt('E:/Lake Level/RS Output/y1\_trainLLR.csv', y1\_train, delimiter = ',')

# Saving observed LL\_G Train  
# To save the predicted data on the drive  
numpy.savetxt('E:/Lake Level/GT Output/y2\_trainLLG.csv', y2\_train, delimiter = ',')

# Model evaluation

# The coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn2 train r\_sq is:', r2\_score(y2\_train, predtrain\_nn2))

The nn2 train r\_sq is: -30.08952657045501

# The nn2 train mse  
print('The nn2 train mse is:', mean\_squared\_error(y2\_train, predtrain\_nn2))

The nn2 train mse is: 15.287957942480476

# The nn2 train rmse  
print('The nn2 train rmse is:', np.sqrt(mean\_squared\_error(y2\_train, predtrain\_nn2)))

The nn2 train rmse is: 3.9099818340345873

# The nn2 train mae  
print('The nn2 train mae is:', mean\_absolute\_error(y2\_train, predtrain\_nn2))

The nn2 train mae is: 2.7270263943142354

# The nn2 train evs  
print('The nn2 train evs is:', explained\_variance\_score(y2\_train, predtrain\_nn2))

The nn2 train evs is: -29.492710594337197

# Prediction on testing data  
pred\_nn2 = model.predict(X\_test).flatten()  
  
print(pred\_nn2[:5])

[277.7389 296.59088 279.0667 279.98514 276.87555]

# To save the predicted data on the drive  
#numpy.savetxt('E:/Lake Level/DL/tanhdlpredLLG.csv', pred\_nn2, delimiter = ',')

# To save the observed data on the drive  
numpy.savetxt('E:/Lake Level/y\_testLLG.csv', y2\_test, delimiter = ',')

# Model evaluation

# The coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn2 coefficient of determination is:', r2\_score(y2\_test, pred\_nn2))

The nn2 coefficient of determination is: -53.428249546429385

# The The MSE  
   
print('The G Data nn2 MSE is:',mean\_squared\_error(y2\_test, pred\_nn2))

The G Data nn2 MSE is: 22.867755393788794

# The RMSE  
   
print('The G data nn2 rmse is:',np.sqrt(mean\_squared\_error(y2\_test, pred\_nn2)))

The G data nn2 rmse is: 4.782024194186892

# The MAE  
  
print('The G data nn2 mae is:',mean\_absolute\_error(y2\_test, pred\_nn2))

The G data nn2 mae is: 3.4024349161783882

# The nn2 Explained variance score  
print('nn2 evs is:', explained\_variance\_score(y2\_test, pred\_nn2))

nn2 evs is: -53.29114394621583

# CV

# Training dataset

y2score\_train = model.evaluate(X\_train, y2\_train, verbose = 2)

print(absolute(np.mean(y2score\_train)))

2.2062233765920007

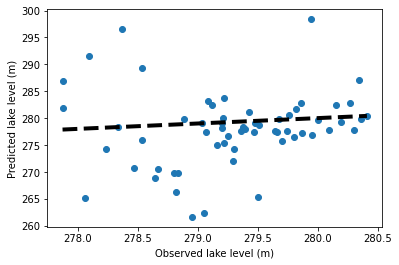
# Testing dataset

y2score\_test = model.evaluate(X\_test, y2\_test, verbose = 2)

print(absolute(np.mean(y2score\_test)))

3.772355578740438

# relu  
# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y2\_test, pred\_nn2)  
ax.plot([y2\_test.min(),y2\_test.max()], [y2\_test.min(), y2\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')  
plt.show()



# Computing the covariance between the observed and predicted values   
from numpy import cov  
covtestnn2 = cov(y2\_test, pred\_nn2)  
print(covtestnn2)

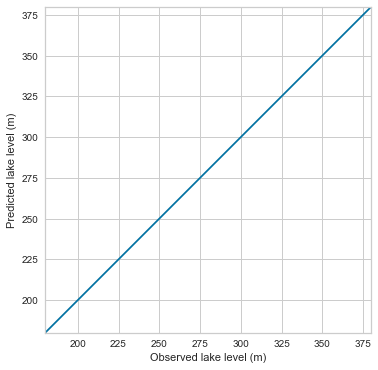
[[ 0.42726607 0.62209026]  
 [ 0.62209026 48.44649216]]

# Computing the pearson correlation between the observed and predicted values   
from scipy.stats import pearsonr  
cortestnn2 = pearsonr(y2\_test, pred\_nn2)  
print(cortestnn2)

(0.13673290113340464, 0.2975270542257127)

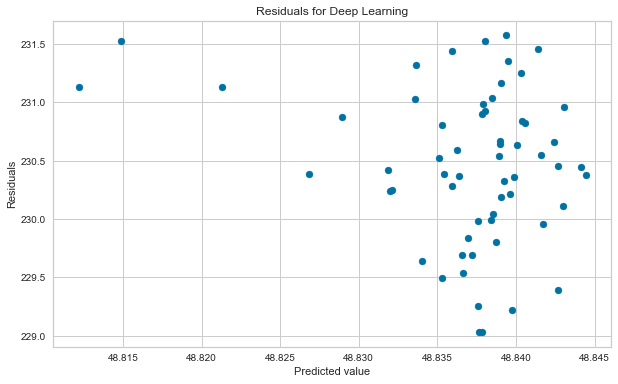
# relu  
plt.axes(aspect='equal')  
plt.scatter(y2\_test, pred\_nn2)  
lims = [180, 380]  
plt.xlim(lims)  
plt.ylim(lims)  
plt.xlabel('Observed lake level (m)')  
plt.ylabel('Predicted lake level (m)')  
plt.plot(lims, lims)

[<matplotlib.lines.Line2D at 0x1c492fc8400>]



# relu  
# Plotting the prediction errors  
from pandas import DataFrame, Series  
error = y2\_test - pred\_nn2  
plt.scatter(pred\_nn2, error)  
plt.xlabel('Predicted value')  
plt.ylabel('Residuals')  
plt.title('Residuals for Deep Learning')

Text(0.5, 1.0, 'Residuals for Deep Learning')



# Sigmoid Activation Function

# Model definition  
model = Sequential()  
model.add(Dense(128, input\_dim = 6, activation = 'sigmoid', kernel\_initializer = 'normal'))  
model.add(Dense(64, activation = 'sigmoid'))  
# Ouput layer  
model.add(Dense(1, activation = 'linear'))  
model.compile(loss='mean\_absolute\_error', optimizer='adam', metrics=['mae', 'mse', rmse, r\_square])  
model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
dense\_34 (Dense) (None, 128) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_35 (Dense) (None, 64) 8256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_36 (Dense) (None, 1) 65   
=================================================================  
Total params: 9,217  
Trainable params: 9,217  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

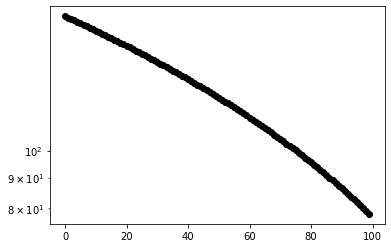
# Fitting the model  
history2 = model.fit(X\_train, y2\_train, validation\_split = 0.25, epochs = 100, batch\_size = 10, verbose = 0)

hist.head()

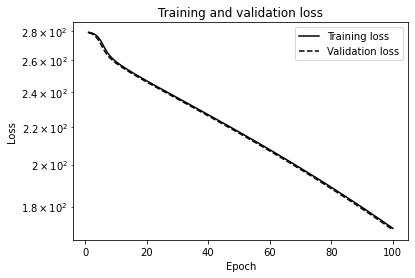
val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
0 228.571248 228.571248 52245.219184 228.571248   
1 227.601049 227.601049 51802.638455 227.601049   
2 226.632774 226.632774 51362.816840 226.632774   
3 225.666623 225.666623 50925.823785 225.666623   
4 224.702454 224.702454 50491.588542 224.702454   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error \  
0 -162072.750868 229.250806 229.250806 52556.521846   
1 -160699.722222 228.279110 228.279110 52111.989583   
2 -159335.240451 227.309729 227.309729 51670.279948   
3 -157979.544271 226.342461 226.342461 51231.485243   
4 -156632.409722 225.377172 225.377172 50795.394676   
  
 rmse r\_square epochs   
0 229.250806 -124102.951389 0   
1 228.279114 -120891.826678 1   
2 227.309732 -123597.996238 2   
3 226.342456 -122915.943866 3   
4 225.377170 -130191.203704 4

# Plotting the training and testing accuracy and loss at each epoch  
from matplotlib import pyplot as plt

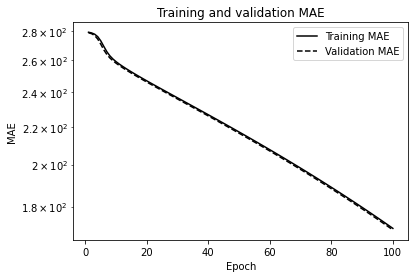
model\_ = model.fit(X\_train, y2\_train, validation\_split = 0.25, epochs = 100, batch\_size = 10, verbose = 0)  
plt.plot(list(model\_.history.values())[0], 'k-o')  
plt.yscale('log')  
plt.show()



# Plotting the training and validation loss  
loss2 = history2.history['loss']  
val\_loss2 = history2.history['val\_loss']  
epochs = range(1, len(loss2)+1)  
plt.plot(epochs, loss2, 'k-', label = 'Training loss')  
plt.plot(epochs, val\_loss2, 'k--', label = 'Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.yscale('log')  
plt.legend(loc='best')  
plt.show()

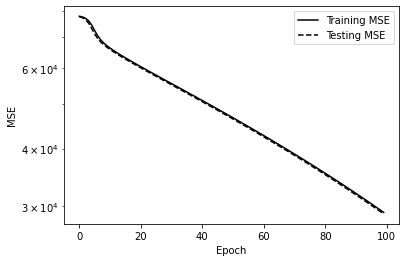


# Plotting the training and validation MAE  
acc2 = history2.history['mean\_absolute\_error']  
val\_acc2 = history2.history['val\_mean\_absolute\_error']  
plt.plot(epochs, acc2, 'k-', label = 'Training MAE')  
plt.plot(epochs, val\_acc2, 'k--', label = 'Validation MAE')  
plt.title('Training and validation MAE')  
plt.xlabel('Epoch')  
plt.ylabel('MAE')  
plt.yscale('log')  
plt.legend(loc = 'best')  
plt.show()



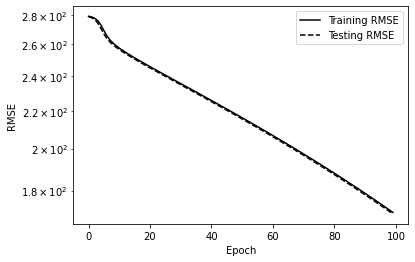
# Plotting the mse  
import matplotlib.pyplot as plt  
plt.plot(history2.history['mean\_squared\_error'], 'k-')  
plt.plot(history2.history['val\_mean\_squared\_error'], 'k--')  
plt.ylabel('MSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MSE', 'Testing MSE'], loc = 'best')

<matplotlib.legend.Legend at 0x1e4c837c2b0>



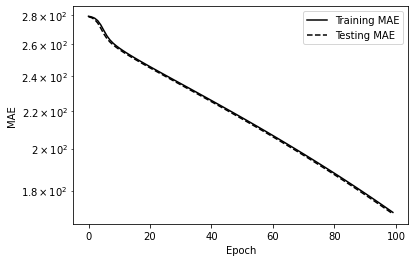
# Plotting the rmse training and testing curve  
plt.plot(history2.history['rmse'], 'k-')  
plt.plot(history2.history['val\_rmse'], 'k--')  
plt.ylabel('RMSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training RMSE', 'Testing RMSE'], loc = 'best')

<matplotlib.legend.Legend at 0x1e4c8428898>



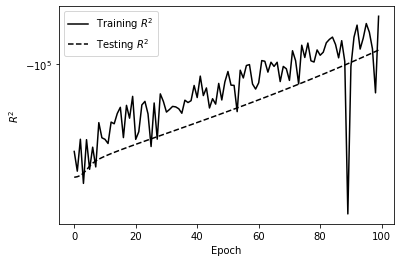
# Plotting the mae  
plt.plot(history2.history['mean\_absolute\_error'], 'k-')  
plt.plot(history2.history['val\_mean\_absolute\_error'], 'k--')  
plt.ylabel('MAE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MAE', 'Testing MAE'], loc = 'best')

<matplotlib.legend.Legend at 0x1e4c951ce48>



# Plotting the r\_squared  
plt.plot(history2.history['r\_square'], 'k-')  
plt.plot(history2.history['val\_r\_square'], 'k--')  
plt.ylabel('$R^2$')  
plt.yscale('symlog')  
plt.xlabel('Epoch')  
plt.legend(['Training $R^2$', 'Testing $R^2$'], loc = 'best')

<matplotlib.legend.Legend at 0x1e4c96de0b8>



#Prediction on the training data

predtrain\_nn2 = model.predict(X\_train)

# The coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn2 train r\_sq is:', r2\_score(y2\_train, predtrain\_nn2))

The nn2 train r\_sq is: -46910.24430248077

# The nn2 train mse  
print('The nn2 train mse is:', mean\_squared\_error(y2\_train, predtrain\_nn2))

The nn2 train mse is: 23068.12644124663

# The nn2 train rmse  
print('The nn2 train ymse is:', np.sqrt(mean\_squared\_error(y2\_train, predtrain\_nn2)))

The nn2 train ymse is: 151.8819490303131

# The nn2 train mae  
print('The nn2 train mae is:', mean\_absolute\_error(y2\_train, predtrain\_nn2))

The nn2 train mae is: 151.88033010355633

# The nn2 train evs  
print('The nn2 train evs is:', explained\_variance\_score(y2\_train, predtrain\_nn2))

The nn2 train evs is: -5.9115956644983925e-05

# Prediction on testing data  
pred\_nn2 = model.predict(X\_test).flatten()  
  
print(pred\_nn2[:5])

[127.3119 127.312004 127.311935 127.311935 127.3118 ]

# To save the predicted data on the drive  
numpy.savetxt('E:/Lake Level/DL/sigdlpredLLG.csv', pred\_nn2, delimiter = ',')

# The coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn2 coefficient of determination is:', r2\_score(y2\_test, pred\_nn2))

The nn2 coefficient of determination is: -54974.55038819995

# Model evaluation

# The The MSE  
   
print('The G Data nn2 MSE is:',mean\_squared\_error(y2\_test, pred\_nn2))

The G Data nn2 MSE is: 23097.701090751685

# The RMSE  
   
print('The G data nn2 rmse is:',np.sqrt(mean\_squared\_error(y2\_test, pred\_nn2)))

The G data nn2 rmse is: 151.9792784913512

# The MAE  
  
print('The G data nn2 mae is:',mean\_absolute\_error(y2\_test, pred\_nn2))

The G data nn2 mae is: 151.9778961283366

# The nn2 Explained variance score  
print('nn2 evs is:', explained\_variance\_score(y2\_test, pred\_nn2))

nn2 evs is: -8.136238112044225e-05

# k-fold cross-validation  
from sklearn.model\_selection import RepeatedKFold, cross\_val\_score  
from sklearn.metrics import mean\_squared\_error

# Training score  
from numpy import absolute

y2score\_train = model.evaluate(X\_train, y2\_train, verbose = 2)

print(absolute(np.mean(y2score\_train)))

5591.053165893554

# Testing score

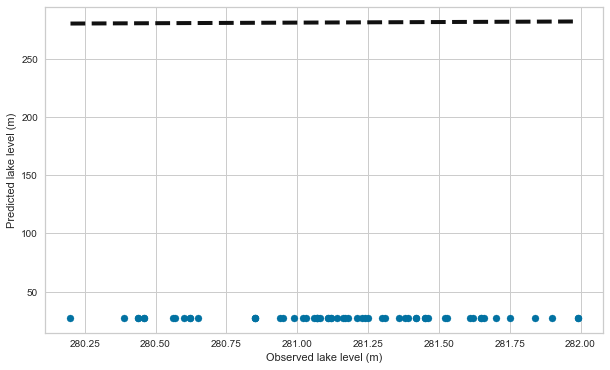
y2score\_test = model.evaluate(X\_test, y2\_test, verbose = 2)

print(absolute(np.mean(y2score\_test)))

6813.566235961914

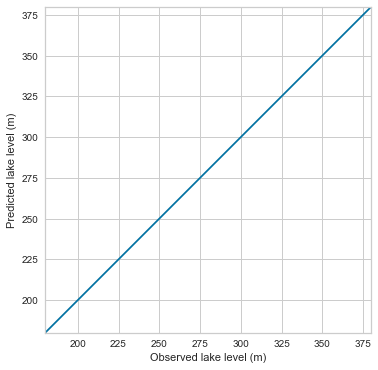
# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_nn1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')

Text(0, 0.5, 'Predicted lake level (m)')



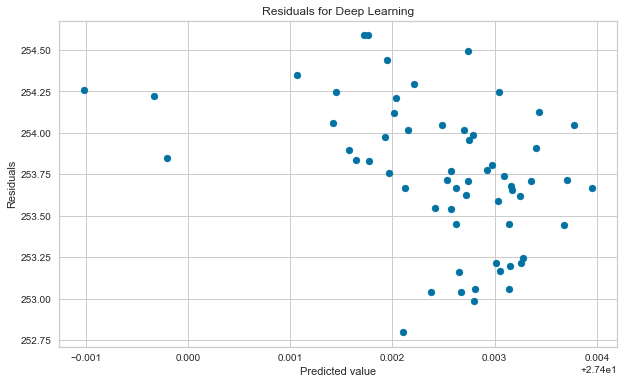
plt.axes(aspect='equal')  
plt.scatter(y1\_test, pred\_nn1)  
lims = [180, 380]  
plt.xlim(lims)  
plt.ylim(lims)  
plt.xlabel('Observed lake level (m)')  
plt.ylabel('Predicted lake level (m)')  
plt.plot(lims, lims)

[<matplotlib.lines.Line2D at 0x1c48fe6c630>]



# Plotting the prediction errors  
from pandas import DataFrame, Series  
error = y1\_test - pred\_nn1  
plt.scatter(pred\_nn1, error)  
plt.xlabel('Predicted value')  
plt.ylabel('Residuals')  
plt.title('Residuals for Deep Learning')

Text(0.5, 1.0, 'Residuals for Deep Learning')



# Tanh Activation Function

# Model definition  
model = Sequential()  
model.add(Dense(128, input\_dim = 6, activation = 'tanh', kernel\_initializer = 'normal'))  
model.add(Dense(64, activation = 'tanh'))  
# Ouput layer  
model.add(Dense(1, activation = 'linear'))  
model.compile(loss='mean\_absolute\_error', optimizer='adam', metrics=['mae', 'mse', rmse, r\_square])  
model.summary()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Layer (type) Output Shape Param #   
=================================================================  
dense\_40 (Dense) (None, 128) 896   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_41 (Dense) (None, 64) 8256   
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
dense\_42 (Dense) (None, 1) 65   
=================================================================  
Total params: 9,217  
Trainable params: 9,217  
Non-trainable params: 0  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

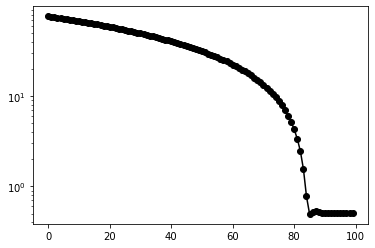
# Fitting the model  
history2 = model.fit(X\_train, y2\_train, validation\_split = 0.25, epochs = 100, batch\_size = 10, verbose = 0)

hist.head()

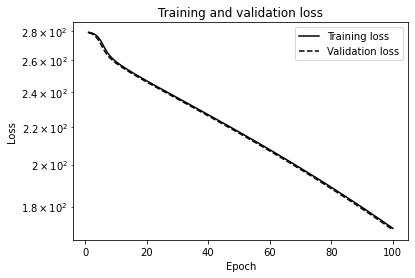
val\_loss val\_mean\_absolute\_error val\_mean\_squared\_error val\_rmse \  
0 277.541724 277.541724 77029.921875 277.541718   
1 275.587911 275.587911 75949.967014 275.587911   
2 272.404060 272.404060 74207.644097 272.404060   
3 267.182895 267.182895 71398.049479 267.182895   
4 259.023261 259.023261 67125.238715 259.023248   
  
 val\_r\_square loss mean\_absolute\_error mean\_squared\_error \  
0 -239005.102431 278.375961 278.375961 77493.865741   
1 -235703.385417 276.701317 276.701317 76565.099537   
2 -230369.140625 274.055012 274.055012 75109.880498   
3 -221777.425347 269.790935 269.790935 72797.656829   
4 -208708.449653 263.100337 263.100337 69245.361111   
  
 rmse r\_square epochs   
0 278.375958 -201482.950231 0   
1 276.701317 -177564.046875 1   
2 274.055010 -317207.839699 2   
3 269.790931 -182839.292245 3   
4 263.100333 -178121.888889 4

# Plotting the training and testing accuracy and loss at each epoch  
from matplotlib import pyplot as plt

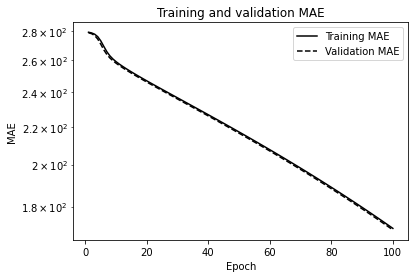
model\_ = model.fit(X\_train, y2\_train, validation\_split = 0.25, epochs = 100, batch\_size = 10, verbose = 0)  
plt.plot(list(model\_.history.values())[0], 'k-o')  
plt.yscale('log')  
plt.show()



# # Tanh Training and validation loss plot  
loss2 = history2.history['loss']  
val\_loss2 = history2.history['val\_loss']  
epochs = range(1, len(loss2)+1)  
plt.plot(epochs, loss2, 'k-', label = 'Training loss')  
plt.plot(epochs, val\_loss2, 'k--', label = 'Validation loss')  
plt.title('Training and validation loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.yscale('log')  
plt.legend(loc='best')  
plt.show()

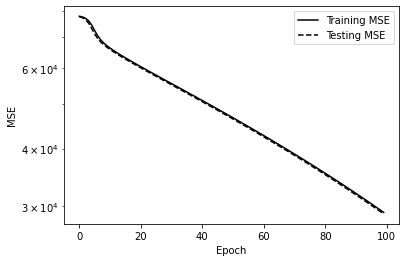


# Tanh Training and validation MAE  
acc2 = history2.history['mean\_absolute\_error']  
val\_acc2 = history2.history['val\_mean\_absolute\_error']  
plt.plot(epochs, acc2, 'k-', label = 'Training MAE')  
plt.plot(epochs, val\_acc2, 'k--', label = 'Validation MAE')  
plt.title('Training and validation MAE')  
plt.xlabel('Epoch')  
plt.ylabel('MAE')  
plt.yscale('log')  
plt.legend(loc = 'best')  
plt.show()



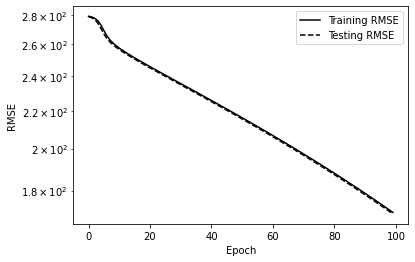
# Tanh: Plotting the mse  
import matplotlib.pyplot as plt  
plt.plot(history2.history['mean\_squared\_error'], 'k-')  
plt.plot(history2.history['val\_mean\_squared\_error'], 'k--')  
plt.ylabel('MSE')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training MSE', 'Testing MSE'], loc = 'best')

<matplotlib.legend.Legend at 0x1e4c99e5dd8>



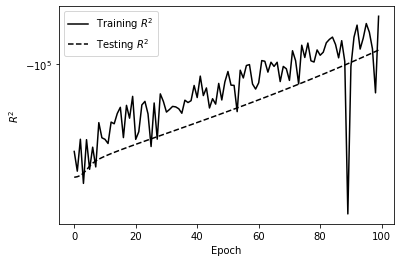
# Tanh: Plotting the rmse training and testing curves  
plt.plot(history2.history['rmse'], 'k-')  
plt.plot(history2.history['val\_rmse'], 'k--')  
plt.ylabel('RMSE')  
plt.yscale('log')  
plt.yscale('log')  
plt.xlabel('Epoch')  
plt.legend(['Training RMSE', 'Testing RMSE'], loc = 'best')

<matplotlib.legend.Legend at 0x1e4c98d3470>



# Tanh: Plotting the r\_squared  
plt.plot(history2.history['r\_square'], 'k-')  
plt.plot(history2.history['val\_r\_square'], 'k--')  
plt.ylabel('$R^2$')  
plt.yscale('symlog')  
plt.xlabel('Epoch')  
plt.legend(['Training $R^2$', 'Testing $R^2$'], loc = 'best')

<matplotlib.legend.Legend at 0x1e4c97d5ac8>



# Prediction on the training dataset  
predtrain\_nn2 = model.predict(X\_train)

# Model evaluation

# The coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn2 train r\_sq is:', r2\_score(y2\_train, predtrain\_nn2))

The nn2 train r\_sq is: -58684.82201542347

# The nn2 train mse  
print('The nn2 train mse is:', mean\_squared\_error(y2\_train, predtrain\_nn2))

The nn2 train mse is: 28858.15506899043

# The nn2 train rmse  
print('The nn2 train ymse is:', np.sqrt(mean\_squared\_error(y2\_train, predtrain\_nn2)))

The nn2 train ymse is: 169.87688209109098

# The nn2 train mae  
print('The nn2 train mae is:', mean\_absolute\_error(y2\_train, predtrain\_nn2))

The nn2 train mae is: 169.8754344482422

# The nn2 train evs  
print('The nn2 train evs is:', explained\_variance\_score(y2\_train, predtrain\_nn2))

The nn2 train evs is: -0.00020376562629542683

# Prediction on testing data  
pred\_nn2 = model.predict(X\_test).flatten()  
  
print(pred\_nn2[:5])

[109.31671 109.31703 109.31697 109.31608 109.316574]

# The testing coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The tanh nn2 coefficient of determination is:', r2\_score(y2\_test, pred\_nn2))

The tanh nn2 coefficient of determination is: -68763.93722253655

# Model evaluation  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score

# The The MSE  
   
print('The G Data nn2 MSE is:',mean\_squared\_error(y2\_test, pred\_nn2))

The G Data nn2 MSE is: 28891.242639225522

# The RMSE  
   
print('The G data nn2 rmse is:',np.sqrt(mean\_squared\_error(y2\_test, pred\_nn2)))

The G data nn2 rmse is: 169.97424110501427

# The MAE  
  
print('The G data nn2 mae is:',mean\_absolute\_error(y2\_test, pred\_nn2))

The G data nn2 mae is: 169.97300492350263

# The nn2 Explained variance score  
print('nn2 evs is:', explained\_variance\_score(y2\_test, pred\_nn2))

nn2 evs is: -0.0002178492325561887

# CV

# Training dataset

y2score\_train = model.evaluate(X\_train, y2\_train, verbose = 2)

print(absolute(np.mean(y2score\_train)))

7006.641703491211

# Testing dataset

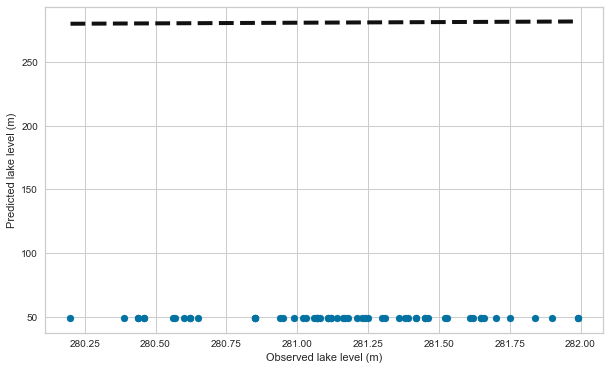
y2score\_test = model.evaluate(X\_test, y2\_test, verbose = 2)

print(absolute(np.mean(y2score\_test)))

8534.761350708006

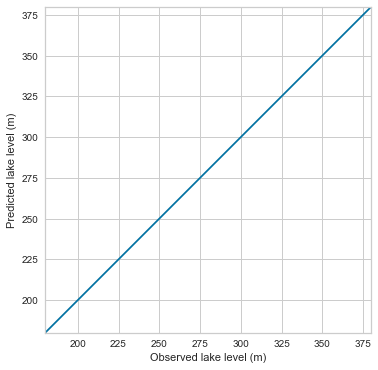
# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaaries and characteristics  
fig, ax = plt.subplots()  
ax.scatter(y1\_test, pred\_nn1)  
ax.plot([y1\_test.min(),y1\_test.max()], [y1\_test.min(), y1\_test.max()], 'k--',lw =4) # Line of best fit  
# Labelling  
ax.set\_xlabel('Observed lake level (m)')  
ax.set\_ylabel('Predicted lake level (m)')

Text(0, 0.5, 'Predicted lake level (m)')



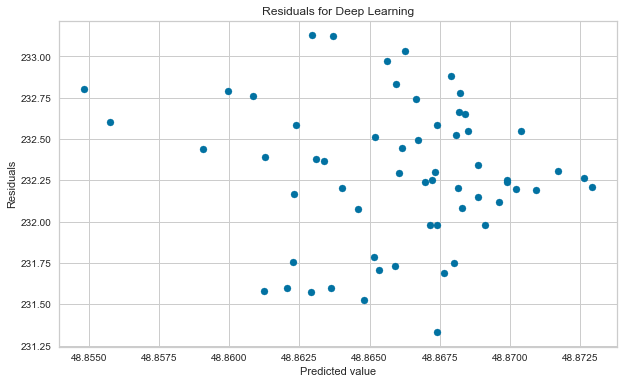
plt.axes(aspect='equal')  
plt.scatter(y1\_test, pred\_nn1)  
lims = [180, 380]  
plt.xlim(lims)  
plt.ylim(lims)  
plt.xlabel('Observed lake level (m)')  
plt.ylabel('Predicted lake level (m)')  
plt.plot(lims, lims)

[<matplotlib.lines.Line2D at 0x1c4914eaa58>]



# Plotting the prediction errors  
from pandas import DataFrame, Series  
error = y1\_test - pred\_nn1  
plt.scatter(pred\_nn1, error)  
plt.xlabel('Predicted value')  
plt.ylabel('Residuals')  
plt.title('Residuals for Deep Learning')

Text(0.5, 1.0, 'Residuals for Deep Learning')



# The coefficient of determination  
from sklearn.metrics import r2\_score, mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score  
  
print('The nn2 coefficient of determination is:', r2\_score(y2\_test, pred\_nn2))

The nn2 coefficient of determination is: -460.74491303215

# relu The nn2 coefficient of determination is: -4785.57791197407

# sigmoid -146713.14435645883

# tanh -126664.7005494471

# Model evaluation  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, explained\_variance\_score

# The The MSE  
   
print('The G Data nn2 MSE is:',mean\_squared\_error(y2\_test, pred\_nn2))

The G Data nn2 MSE is: 193.9998036596454

# relu mse 2011.0566438658514

# sigmoid 61641.21010525161

# tanh 53217.95723885586

# The RMSE  
   
print('The G data nn2 rmse is:',np.sqrt(mean\_squared\_error(y2\_test, pred\_nn2)))

The G data nn2 rmse is: 13.928381228974363

# relu rmse 44.84480620836544

# sigmoid rmse 248.27647916234758

# tanh rmse 230.69017586116635

# The MAE  
  
print('The G data nn2 mae is:',mean\_absolute\_error(y2\_test, pred\_nn2))

The G data nn2 mae is: 10.88163858032227

#relu mae 34.39941069539388

# sigmoi mae 248.27563156509402

# tanh mae 230.6892627182007

# The nn2 Explained variance score  
print('nn2 evs is:', explained\_variance\_score(y2\_test, pred\_nn2))

nn2 evs is: -445.9632452662075

# relu evs -4411.001509318695

# sigmoid evs -0.0017404303838475244

# tanh evs -0.0027619433542900307

# CV

# Training dataset

y2score\_train = model.evaluate(X\_train, y2\_train, verbose = 0)

print(absolute(np.mean(y2score\_train)))

6.429788046942815

# relu [1745.7116509331597, 32.59369862874349]

# sigmoid [61592.756336805556, 248.17788865831164]

# tanh [53173.053385416664, 230.59175957573785]

# Testing dataset

y2score\_test = model.evaluate(X\_test, y2\_test, verbose = 0)

print(absolute(np.mean(y2score\_test)))

13.860934575398778

# relu [2011.0568359375, 34.39941253662109]

# sigmoid [61641.20911458333, 248.27562662760417]

# tanh [53217.959635416664, 230.68925882975262]

######## CV METHOD 2 #####  
seed = 7

# Deep Learning modeling  
# Importing libraries  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.wrappers.scikit\_learn import KerasRegressor  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import KFold  
from sklearn.preprocessing import StandardScaler  
from sklearn.pipeline import Pipeline  
import warnings;warnings.simplefilter('ignore')

# On the training dataset   
# Model definition  
def model():  
 # Model creation  
 model = Sequential()  
 model.add(Dense(128, input\_dim = 6, kernel\_initializer = 'normal', activation = 'relu'))  
 model.add(Dense(64, kernel\_initializer = 'normal', activation = 'relu'))  
 # Ouput layer  
 model.add(Dense(1, activation = 'linear', kernel\_initializer = 'normal'))  
 # Model compilation  
 model.compile(loss='mean\_squared\_error', optimizer='adam')  
 return model  
# Evaluate the model with standardized dataset  
estimators = []  
estimators.append(('standardize',StandardScaler()))  
estimators.append(('mlp', KerasRegressor(build\_fn = model, epochs = 100, batch\_size = 5, verbose = 0 )))  
pipeline = Pipeline(estimators)  
kfold = KFold(n\_splits = 10)  
results = cross\_val\_score(pipeline, X\_train, y2\_train, cv = kfold)  
print('Standardized: %.2f, %.2f, %.2f MSE'% (results.mean(), results.var(), results.std()))

Standardized: -5.52, 13.63, 3.69 MSE

# Relu: -6.86, 41.53, 6.44 MSE || -1.11, 0.16, 0.40 MAE

# Sigmoid -9449.92, 128959.59, 359.11 MSE || -60.64, 4.22, 2.05 MAE

# Tanh -7987.39, 1723.65, 41.52 MSE || -52.28, 0.04, 0.20 MAE

# On the testing dataset

# On the testing dataset   
# Model definition  
def model():  
 # Model creation  
 model = Sequential()  
 model.add(Dense(128, input\_dim = 6, kernel\_initializer = 'normal', activation = 'relu'))  
 model.add(Dense(64, kernel\_initializer = 'normal', activation = 'relu'))  
 # Ouput layer  
 model.add(Dense(1, activation = 'linear', kernel\_initializer = 'normal'))  
 # Model compilation  
 model.compile(loss='mean\_squared\_error', optimizer='adam')  
 return model  
# Evaluate the model with standardized dataset  
estimators = []  
estimators.append(('standardize',StandardScaler()))  
estimators.append(('mlp', KerasRegressor(build\_fn = model, epochs = 100, batch\_size = 5, verbose = 0 )))  
pipeline = Pipeline(estimators)  
kfold = KFold(n\_splits = 10)  
results = cross\_val\_score(pipeline, X\_test, y2\_test, cv = kfold)  
print('Standardized: %.2f, %.2f, %.2f MSE'% (results.mean(), results.var(), results.std()))

Standardized: -745.50, 101749.75, 318.98 MSE

# Relu -779.87, 101896.86, 319.21 MSE || -13.52, 13.47, 3.67 MAE

# Sigmoid -44238.34, 476850.88, 690.54 MSE || -205.79, 0.80, 0.89 MAE

# Tanh -40370.29, 25540.20, 159.81 MSE || -196.67, 0.11, 0.33 MAE

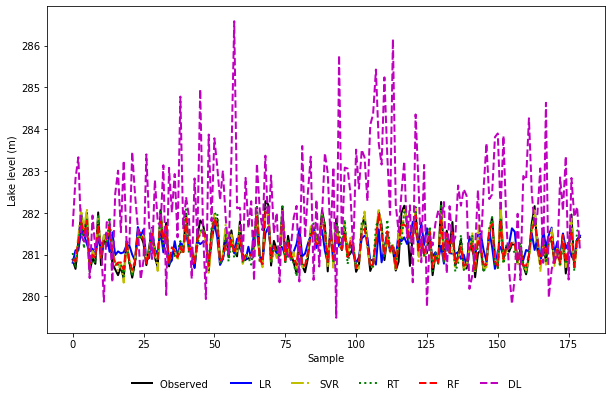
#############################################################################

# Plotting the predicted and observed data

### Remote sensing data

# Training data

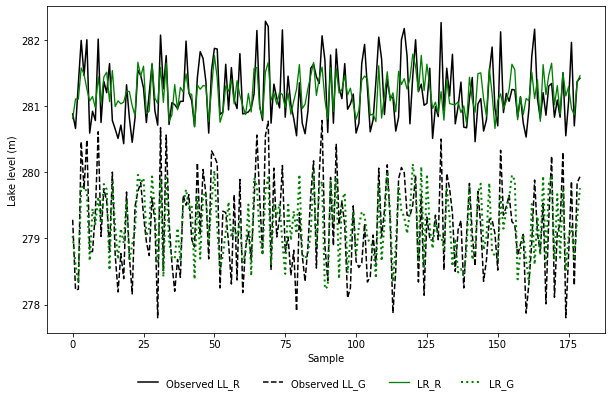
# Plotting the predicted against the observed testing data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
#fig.set\_size\_inches(18.5, 10.5)  
x\_ax = range(len(X\_train))  
plt.plot(x\_ax, y1\_train, label = 'Observed ',color = 'k', linestyle = '-', lw = 2)  
plt.plot(x\_ax, ytrain\_predlm1, label = 'LR', color = 'b', linestyle = '-', lw = 2)  
plt.plot(x\_ax, ytrain\_predsvr1, label = 'SVR',color = 'y', linestyle = '-.', lw = 2)  
plt.plot(x\_ax, ytrain\_preddt1, label = 'RT',color = 'g', linestyle = ':', lw = 2)  
plt.plot(x\_ax, ytrain\_predrf1, label = 'RF', color = 'r', linestyle = '--', lw = 2)  
plt.plot(x\_ax, predtrain\_nn1, label = 'DL', color = 'm', linestyle = '--', lw = 2)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 6, frameon=False)  
plt.show()



# Plotting LL\_R and LL\_G together

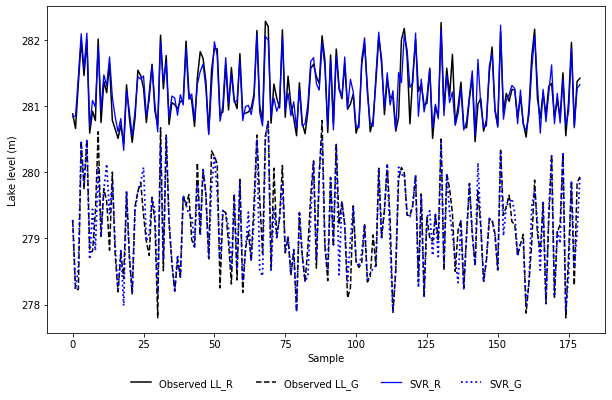
### Plotting predicted LR versus observed LL\_R and LL\_G training data

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
x\_ax = range(len(X\_train))  
plt.plot(x\_ax, y1\_train, label = 'Observed LL\_R ', color = 'k', linestyle = '-', lw = 1.5)  
plt.plot(x\_ax, y2\_train, label = 'Observed LL\_G ', color = 'k', linestyle = '--', lw = 1.5)  
plt.plot(x\_ax, ytrain\_predlm1, label = 'LR\_R', color = 'g', linestyle = '-', lw = 1.3)  
plt.plot(x\_ax, ytrain\_predlm2, label = 'LR\_G',color = 'g', linestyle = ':', lw = 2)  
#plt.plot(x\_ax, pred\_svr1, label = 'SVR\_R',color = 'b', linestyle = '-', lw = 1.3)  
#plt.plot(x\_ax, pred\_svr2, label = 'SVR\_G',color = 'b', linestyle = ':', lw = 1.8)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 4, frameon=False)  
plt.show()



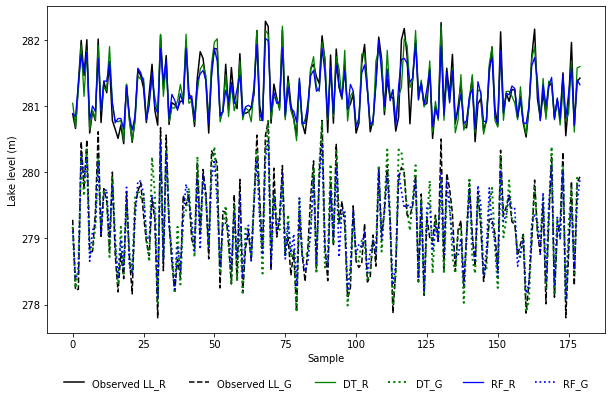
### Plotting predicted from SVR versus observed LL\_R and LL\_G training data

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
x\_ax = range(len(X\_train))  
plt.plot(x\_ax, y1\_train, label = 'Observed LL\_R ', color = 'k', linestyle = '-', lw = 1.5)  
plt.plot(x\_ax, y2\_train, label = 'Observed LL\_G ', color = 'k', linestyle = '--', lw = 1.5)  
plt.plot(x\_ax, ytrain\_predsvr1, label = 'SVR\_R',color = 'b', linestyle = '-', lw = 1.3)  
plt.plot(x\_ax, ytrain\_predsvr2, label = 'SVR\_G',color = 'b', linestyle = ':', lw = 1.8)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 6, frameon=False)  
plt.show()



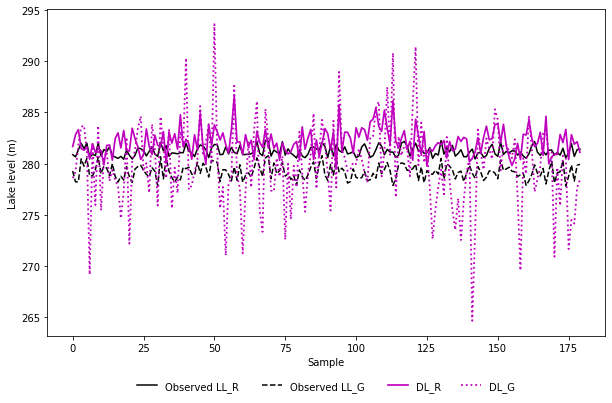
### Plotting predicted from RT and RF versus observed LL\_R and LL\_G training data

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
x\_ax = range(len(X\_train))  
plt.plot(x\_ax, y1\_train, label = 'Observed LL\_R ', color = 'k', linestyle = '-', lw = 1.5)  
plt.plot(x\_ax, y2\_train, label = 'Observed LL\_G ', color = 'k', linestyle = '--', lw = 1.5)  
plt.plot(x\_ax, ytrain\_preddt1, label = 'DT\_R', color = 'g', linestyle = '-', lw = 1.3)  
plt.plot(x\_ax, ytrain\_preddt2, label = 'DT\_G',color = 'g', linestyle = ':', lw = 2)  
plt.plot(x\_ax, ytrain\_predrf1, label = 'RF\_R',color = 'b', linestyle = '-', lw = 1.3)  
plt.plot(x\_ax, ytrain\_predrf2, label = 'RF\_G',color = 'b', linestyle = ':', lw = 1.7)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 6, frameon=False)  
plt.show()



### Plotting predicted from DL versus observed LL\_R and LL\_G training data

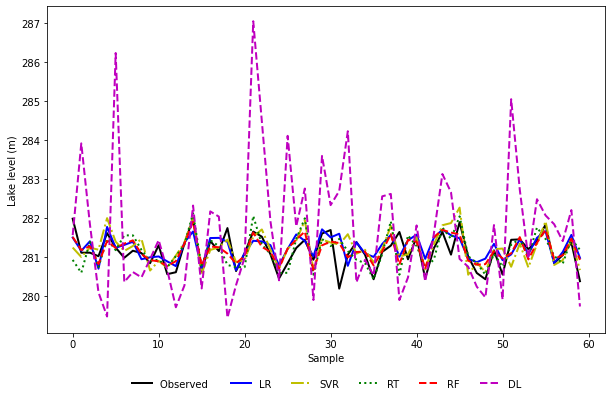
# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
x\_ax = range(len(X\_train))  
plt.plot(x\_ax, y1\_train, label = 'Observed LL\_R ', color = 'k', linestyle = '-', lw = 1.5)  
plt.plot(x\_ax, y2\_train, label = 'Observed LL\_G ', color = 'k', linestyle = '--', lw = 1.5)  
plt.plot(x\_ax, predtrain\_nn1, label = 'DL\_R', color = 'm', linestyle = '-', lw = 1.7)  
plt.plot(x\_ax, predtrain\_nn2, label = 'DL\_G',color = 'm', linestyle = ':', lw = 1.8)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 4, frameon=False)  
plt.show()



######################################################################

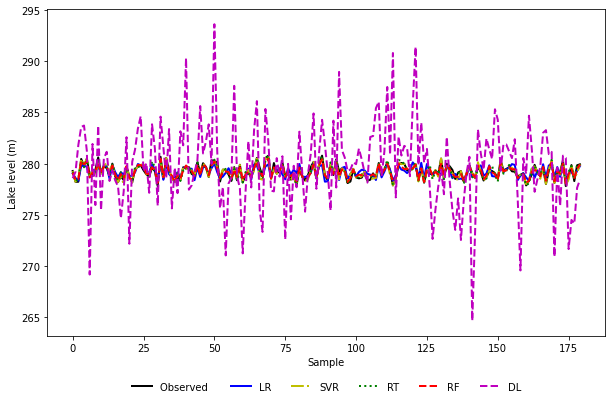
# Testing data

# Plotting the predicted against the observed testing data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
#fig.set\_size\_inches(18.5, 10.5)  
x\_ax = range(len(X\_test))  
plt.plot(x\_ax, y1\_test, label = 'Observed ',color = 'k', linestyle = '-', lw = 2)  
plt.plot(x\_ax, pred\_lm1, label = 'LR', color = 'b', linestyle = '-', lw = 2)  
plt.plot(x\_ax, pred\_svr1, label = 'SVR',color = 'y', linestyle = '-.', lw = 2)  
plt.plot(x\_ax, pred\_dt1, label = 'RT',color = 'g', linestyle = ':', lw = 2)  
plt.plot(x\_ax, pred\_rf1, label = 'RF', color = 'r', linestyle = '--', lw = 2)  
plt.plot(x\_ax, pred\_nn1, label = 'DL', color = 'm', linestyle = '--', lw = 2)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 6, frameon=False)  
plt.show()

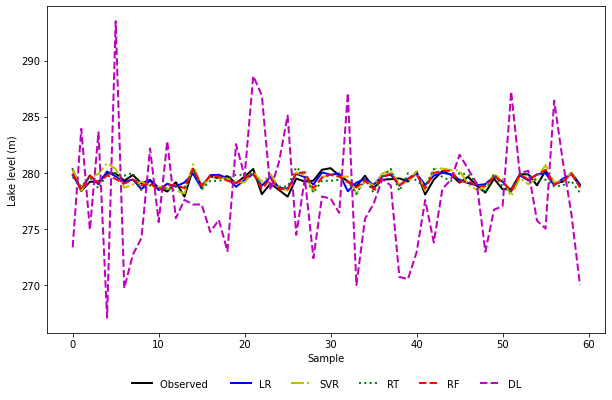


# Ground truth data

# Plotting the predicted against the observed data on the training  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
x\_ax = range(len(X\_train))  
plt.plot(x\_ax, y2\_train, label = 'Observed ', color = 'k', linestyle = '-', lw = 2)  
plt.plot(x\_ax, ytrain\_predlm2, label = 'LR', color = 'b', linestyle = '-', lw = 2)  
plt.plot(x\_ax, ytrain\_predsvr2, label = 'SVR', color = 'y', linestyle = '-.', lw = 2)  
plt.plot(x\_ax, ytrain\_preddt2, label = 'RT', color = 'g', linestyle = ':', lw = 2)  
plt.plot(x\_ax, ytrain\_predrf2, label = 'RF', color = 'r', linestyle = '--', lw = 2)  
plt.plot(x\_ax, predtrain\_nn2, label = 'DL', color = 'm', linestyle = '--', lw = 2)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol=6, frameon=False)  
plt.show()



# Plotting the predicted against the observed data on the testing data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
#fig.set\_size\_inches(18.5, 10.5)  
x\_ax = range(len(X\_test))  
plt.plot(x\_ax, y2\_test, label = 'Observed ', color = 'k', linestyle = '-', lw = 2)  
plt.plot(x\_ax, pred\_lm2, label = 'LR', color = 'b', linestyle = '-', lw = 2)  
plt.plot(x\_ax, pred\_svr2, label = 'SVR', color = 'y', linestyle = '-.', lw = 2)  
plt.plot(x\_ax, pred\_dt2, label = 'RT', color = 'g', linestyle = ':', lw = 2)  
plt.plot(x\_ax, pred\_rf2, label = 'RF', color = 'r', linestyle = '--', lw = 2)  
plt.plot(x\_ax, pred\_nn2, label = 'DL', color = 'm', linestyle = '--', lw = 2)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol=6, frameon=False)  
plt.show()

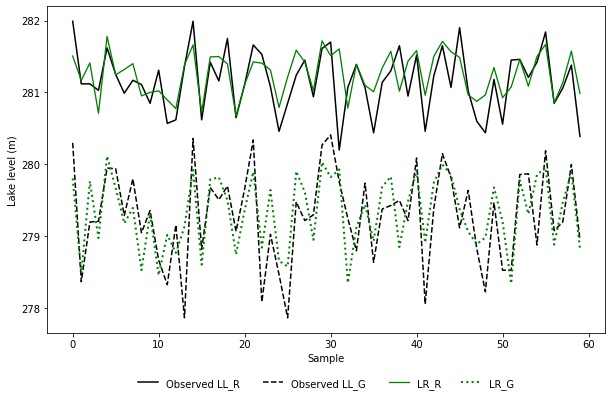


##########################################################################

# Plotting LL\_R and LL\_G models together

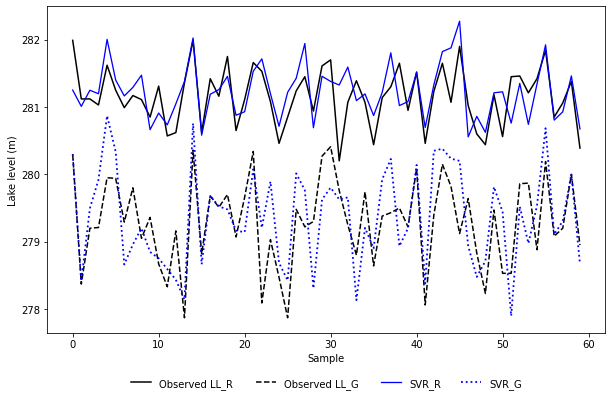
# Plotting predicted from LR versus observed LL\_R and LL\_G testing data

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
x\_ax = range(len(X\_test))  
plt.plot(x\_ax, y1\_test, label = 'Observed LL\_R ', color = 'k', linestyle = '-', lw = 1.5)  
plt.plot(x\_ax, y2\_test, label = 'Observed LL\_G ', color = 'k', linestyle = '--', lw = 1.5)  
plt.plot(x\_ax, pred\_lm1, label = 'LR\_R', color = 'g', linestyle = '-', lw = 1.3)  
plt.plot(x\_ax, pred\_lm2, label = 'LR\_G',color = 'g', linestyle = ':', lw = 2)  
#plt.plot(x\_ax, pred\_svr1, label = 'SVR\_R',color = 'b', linestyle = '-', lw = 1.3)  
#plt.plot(x\_ax, pred\_svr2, label = 'SVR\_G',color = 'b', linestyle = ':', lw = 1.8)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 4, frameon=False)  
plt.show()



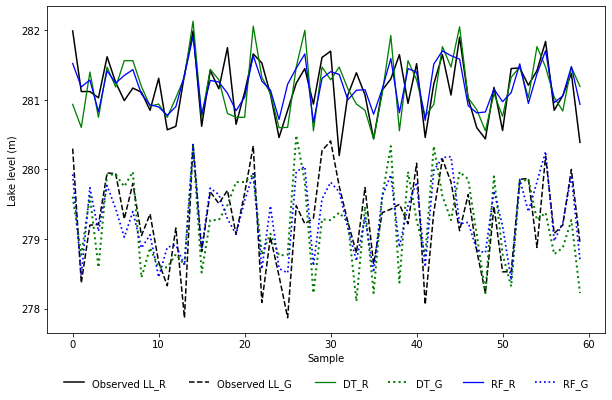
# Plotting predicted from SVR versus observed LL\_R and LL\_G

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
#fig.set\_size\_inches(18.5, 10.5)  
x\_ax = range(len(X\_test))  
plt.plot(x\_ax, y1\_test, label = 'Observed LL\_R ', color = 'k', linestyle = '-', lw = 1.5)  
plt.plot(x\_ax, y2\_test, label = 'Observed LL\_G ', color = 'k', linestyle = '--', lw = 1.5)  
#plt.plot(x\_ax, pred\_lm1, label = 'LR\_R', color = 'g', linestyle = '-', lw = 1.3)  
#plt.plot(x\_ax, pred\_lm2, label = 'LR\_G',color = 'g', linestyle = ':', lw = 2)  
plt.plot(x\_ax, pred\_svr1, label = 'SVR\_R',color = 'b', linestyle = '-', lw = 1.3)  
plt.plot(x\_ax, pred\_svr2, label = 'SVR\_G',color = 'b', linestyle = ':', lw = 1.8)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 6, frameon=False)  
plt.show()



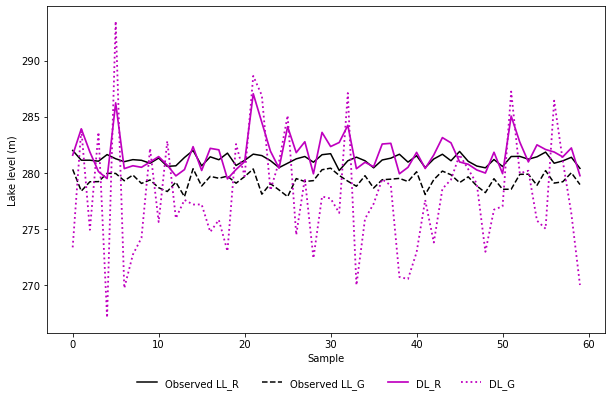
# Plotting predicted from RT and RF versus observed LL\_R and LL\_G

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
#fig.set\_size\_inches(18.5, 10.5)  
x\_ax = range(len(X\_test))  
plt.plot(x\_ax, y1\_test, label = 'Observed LL\_R ', color = 'k', linestyle = '-', lw = 1.5)  
plt.plot(x\_ax, y2\_test, label = 'Observed LL\_G ', color = 'k', linestyle = '--', lw = 1.5)  
plt.plot(x\_ax, pred\_dt1, label = 'DT\_R', color = 'g', linestyle = '-', lw = 1.3)  
plt.plot(x\_ax, pred\_dt2, label = 'DT\_G',color = 'g', linestyle = ':', lw = 2)  
plt.plot(x\_ax, pred\_rf1, label = 'RF\_R',color = 'b', linestyle = '-', lw = 1.3)  
plt.plot(x\_ax, pred\_rf2, label = 'RF\_G',color = 'b', linestyle = ':', lw = 1.7)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 6, frameon=False)  
plt.show()



# Plotting predicted from DL versus observed LL\_R and LL\_G

# Plotting the predicted against the observed data  
import matplotlib.pyplot as plt  
# Setting the boundaries and characteristics  
plt.rcParams['figure.figsize'] = (10,6)  
#fig.set\_size\_inches(18.5, 10.5)  
x\_ax = range(len(X\_test))  
plt.plot(x\_ax, y1\_test, label = 'Observed LL\_R ', color = 'k', linestyle = '-', lw = 1.5)  
plt.plot(x\_ax, y2\_test, label = 'Observed LL\_G ', color = 'k', linestyle = '--', lw = 1.5)  
plt.plot(x\_ax, pred\_nn1, label = 'DL\_R', color = 'm', linestyle = '-', lw = 1.7)  
plt.plot(x\_ax, pred\_nn2, label = 'DL\_G',color = 'm', linestyle = ':', lw = 1.8)  
plt.ylabel('Lake level (m)')  
plt.xlabel('Sample')  
plt.legend(bbox\_to\_anchor =(0.5, -0.2), loc='lower center', ncol= 4, frameon=False)  
plt.show()



#########################################################################

# COMPARING THE DIFFERENT ALGORITHMS

# Importing the libraries  
from sklearn import model\_selection  
import matplotlib.pyplot as plt  
from sklearn.linear\_model import LinearRegression  
from sklearn.svm import SVR  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.ensemble import RandomForestRegressor  
from keras.models import Sequential  
from keras.wrappers.scikit\_learn import KerasRegressor

# Preparing the configuration for cross-validation test  
seed = 7

# Models preparation  
models = []  
models.append(('LR', LinearRegression(fit\_intercept=True, copy\_X = True,positive= False, n\_jobs = None, normalize=False)))  
models.append(('SVR', SVR(kernel = 'rbf', epsilon = 0.1, C = 90)))  
models.append(('DT', DecisionTreeRegressor(max\_depth = 9, max\_features = 2, min\_samples\_leaf = 2, min\_samples\_split = 4, max\_leaf\_nodes= 100)))  
models.append(('RF', RandomForestRegressor(n\_estimators = 150,max\_depth = 12, max\_features = 2, min\_samples\_leaf = 2, min\_samples\_split = 2, max\_leaf\_nodes = 75)))  
#models.append(('DL', Sequential()))  
#models.append(('DL', KerasRegressor()))

# Remote sensing lake level as output

# Model evaluation on the whole dataset

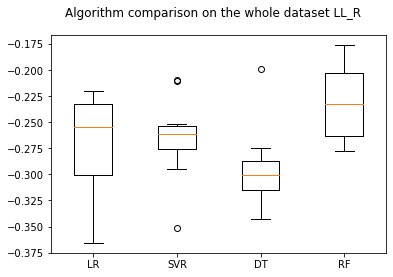
#MAE

# Model evaluation  
import warnings;warnings.simplefilter('ignore')  
results = []  
names = []  
scoring = 'neg\_mean\_absolute\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_scaled, y1, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.272661 (0.050310)  
SVR: -0.265164 (0.038473)  
DT: -0.295865 (0.037572)  
RF: -0.231822 (0.033552)

# MAE  
#LR: -0.272661 (0.050310)  
#SVR: -0.265164 (0.038473)  
#DT: -0.295865 (0.037572)  
#RF: -0.231822 (0.033552)

# MAE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison on the whole dataset LL\_R')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



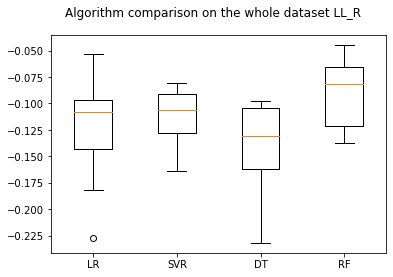
# MSE

# Model evaluation  
import warnings;warnings.simplefilter('ignore')  
results = []  
names = []  
scoring = 'neg\_mean\_squared\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_scaled, y1, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.122567 (0.048739)  
SVR: -0.112706 (0.026785)  
DT: -0.142132 (0.043186)  
RF: -0.091061 (0.031080)

# MSE  
#LR: -0.122567 (0.048739)  
#SVR: -0.112706 (0.026785)  
#DT: -0.142132 (0.043186)  
#RF: -0.091061 (0.031080)

#MSE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison on the whole dataset LL\_R')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



# Model evaluation on the training dataset

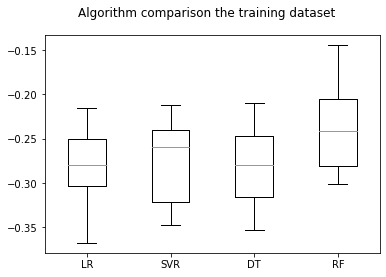
#MAE

# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_absolute\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_train, y1\_train, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.282777 (0.044139)  
SVR: -0.274544 (0.045620)  
DT: -0.281412 (0.047215)  
RF: -0.236088 (0.049894)

# MAE  
#LR: -0.282777 (0.044139)  
#SVR: -0.274544 (0.045620)  
#DT: -0.281412 (0.047215)  
#RF: -0.236088 (0.049894)

#MAE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison the training dataset')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



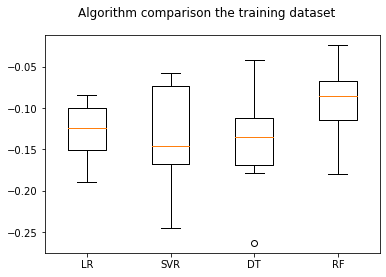
# MSE

# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_squared\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_train, y1\_train, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.128991 (0.034917)  
SVR: -0.135525 (0.062077)  
DT: -0.140976 (0.055236)  
RF: -0.093707 (0.040675)

#MSE  
#LR: -0.128991 (0.034917)  
#SVR: -0.135525 (0.062077)  
#DT: -0.140976 (0.055236)  
#RF: -0.093707 (0.040675)

#MSE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison the training dataset')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



# Model evaluation on the testing dataset

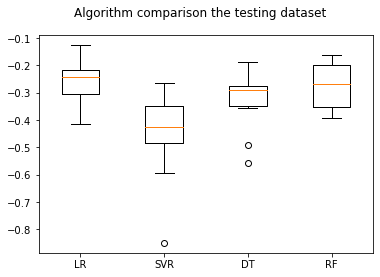
# MAE

# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_absolute\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_test, y1\_test, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.258272 (0.077461)  
SVR: -0.447112 (0.164668)  
DT: -0.328694 (0.108079)  
RF: -0.274755 (0.080303)

#MAE  
#LR: -0.258272 (0.077461)  
#SVR: -0.447112 (0.164668)  
#DT: -0.328694 (0.108079)  
#RF: -0.274755 (0.080303)

#MAE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison the testing dataset')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



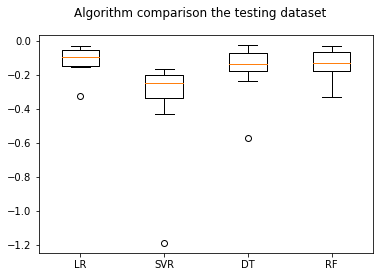
# MSE

# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_squared\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_test, y1\_test, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.114577 (0.082847)  
SVR: -0.350943 (0.289987)  
DT: -0.166557 (0.147400)  
RF: -0.133852 (0.088318)

#MSE  
#LR: -0.114577 (0.082847)  
#SVR: -0.350943 (0.289987)  
#DT: -0.166557 (0.147400)  
#RF: -0.133852 (0.088318)

# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison the testing dataset')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



# Ground truth lake level as output

# Models preparation  
models = []  
models.append(('LR', LinearRegression(fit\_intercept=True, copy\_X = True,positive= False, n\_jobs = None, normalize=False)))  
models.append(('SVR', SVR(kernel = 'rbf', epsilon = 0.01, C = 90)))  
models.append(('DT', DecisionTreeRegressor(max\_depth = 9, max\_features = 2, min\_samples\_leaf = 2, min\_samples\_split = 4, max\_leaf\_nodes= 100)))  
models.append(('RF', RandomForestRegressor(n\_estimators = 150,max\_depth = 12, max\_features = 2, min\_samples\_leaf = 2, min\_samples\_split = 2, max\_leaf\_nodes = 75)))

# Model evaluation on the whole dataset

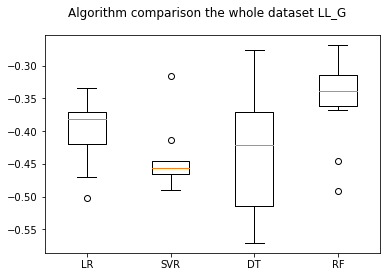
# MAE

# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_absolute\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_scaled, y2, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.400055 (0.048719)  
SVR: -0.443806 (0.047040)  
DT: -0.428603 (0.098739)  
RF: -0.352267 (0.064447)

#MAE  
#LR: -0.400055 (0.048719)  
#SVR: -0.443806 (0.047040)  
#DT: -0.428603 (0.098739)  
#RF: -0.352267 (0.064447)

#MAE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison the whole dataset LL\_G')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()

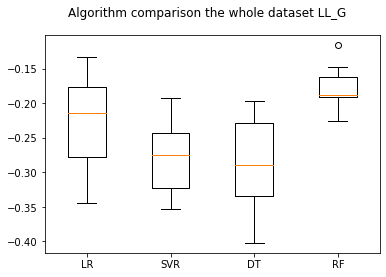


# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_squared\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_scaled, y2, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.229009 (0.070117)  
SVR: -0.277095 (0.053987)  
DT: -0.285518 (0.064404)  
RF: -0.176809 (0.029234)

# MSE  
#LR: -0.229009 (0.070117)  
#SVR: -0.277095 (0.053987)  
#DT: -0.285518 (0.064404)  
#RF: -0.176809 (0.029234)

#MSE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison the whole dataset LL\_G')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



# Model evaluation on the training dataset

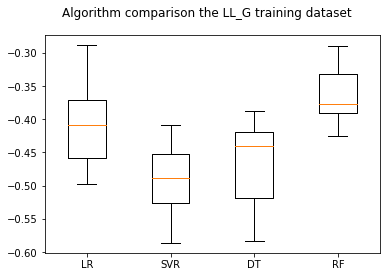
# MAE

# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_absolute\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10,shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_train, y2\_train, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.406291 (0.061784)  
SVR: -0.489284 (0.053984)  
DT: -0.469709 (0.064093)  
RF: -0.364124 (0.039333)

# MAE  
#LR: -0.406291 (0.061784)  
#SVR: -0.489284 (0.053984)  
#DT: -0.469709 (0.064093)  
#RF: -0.364124 (0.039333)

#MAE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison the LL\_G training dataset')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



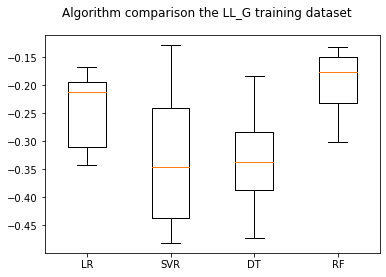
# MSE

# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_squared\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_train, y2\_train, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.244439 (0.063873)  
SVR: -0.335596 (0.113586)  
DT: -0.329653 (0.083844)  
RF: -0.196378 (0.055235)

#MSE  
#LR: -0.244439 (0.063873)  
#SVR: -0.335596 (0.113586)  
#DT: -0.329653 (0.083844)  
#RF: -0.196378 (0.055235)

#MSE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison the LL\_G training dataset')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



# Model evaluation on the testing dataset

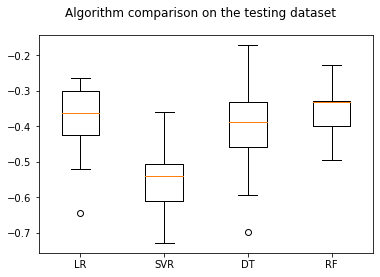
# MAE

# Model evaluation  
results = []  
names = []  
scoring = 'neg\_mean\_absolute\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_test, y2\_test, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.386028 (0.113028)  
SVR: -0.562000 (0.103447)  
DT: -0.406611 (0.147265)  
RF: -0.353457 (0.077874)

#MAE  
#LR: -0.386028 (0.113028)  
#SVR: -0.562000 (0.103447)  
#DT: -0.406611 (0.147265)  
#RF: -0.353457 (0.077874)

#MAE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison on the testing dataset')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()



#MSE

results = []  
names = []  
scoring = 'neg\_mean\_squared\_error'  
for name, model in models:  
 kfold = model\_selection.KFold(n\_splits = 10, shuffle = True)  
 cv\_results = model\_selection.cross\_val\_score(model, X\_test, y2\_test, cv = kfold, scoring = scoring)  
 results.append(cv\_results)  
 names.append(name)  
 msg = '%s: %f (%f)' % (name, cv\_results.mean(), cv\_results.std())  
 print(msg)

LR: -0.220572 (0.096398)  
SVR: -0.589415 (0.206164)  
DT: -0.348947 (0.196455)  
RF: -0.195860 (0.085949)

#MSE  
#LR: -0.220572 (0.096398)  
#SVR: -0.589415 (0.206164)  
#DT: -0.348947 (0.196455)  
#RF: -0.195860 (0.085949)

#MSE  
# Plotting the algorithm comparison in a boxplot  
fig = plt.figure()  
fig.suptitle('Algorithm comparison on the testing dataset')  
ax = fig.add\_subplot(111)  
plt.boxplot(results)  
ax.set\_xticklabels(names)  
plt.show()

