

Applied Machine Learning

Homework 1 Fix

part 5 - Added learning rates to SGD and plotted the loss vs epochs graphs

part 6 - added ridge, lasso, etc for polynomial regressor.

In [1]:

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import numpy as np
4 from sklearn.pipeline import Pipeline
5 from sklearn.linear_model import LinearRegression
6 from sklearn.preprocessing import OneHotEncoder
7 from sklearn.compose import ColumnTransformer
8 from sklearn.model_selection import train_test_split
9 from sklearn.model_selection import StratifiedShuffleSplit
10 from sklearn.model_selection import KFold
11 from sklearn.linear_model import Ridge
12 from sklearn.linear_model import Lasso
13 from sklearn.linear_model import ElasticNet
14 from sklearn.linear_model import SGDRegressor
15 from sklearn.preprocessing import PolynomialFeatures
```

Question 1

In [2]:

```

1 # Loading the dataset as a dataframe
2
3 happiness_df = pd.read_csv('happiness_data.csv')
4 happiness_df.head(20)

```

Out[2]:

	Country name	year	Life Ladder	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption
0	Afghanistan	2008	3.724	7.370	0.451	50.80	0.718	0.168	0.882
1	Afghanistan	2009	4.402	7.540	0.552	51.20	0.679	0.190	0.850
2	Afghanistan	2010	4.758	7.647	0.539	51.60	0.600	0.121	0.707
3	Afghanistan	2011	3.832	7.620	0.521	51.92	0.496	0.162	0.731
4	Afghanistan	2012	3.783	7.705	0.521	52.24	0.531	0.236	0.776
5	Afghanistan	2013	3.572	7.725	0.484	52.56	0.578	0.061	0.823
6	Afghanistan	2014	3.131	7.718	0.526	52.88	0.509	0.104	0.871
7	Afghanistan	2015	3.983	7.702	0.529	53.20	0.389	0.080	0.881
8	Afghanistan	2016	4.220	7.697	0.559	53.00	0.523	0.042	0.793
9	Afghanistan	2017	2.662	7.697	0.491	52.80	0.427	-0.121	0.954
10	Afghanistan	2018	2.694	7.692	0.508	52.60	0.374	-0.094	0.928
11	Afghanistan	2019	2.375	7.697	0.420	52.40	0.394	-0.108	0.924
12	Albania	2007	4.634	9.142	0.821	65.80	0.529	-0.009	0.875
13	Albania	2009	5.485	9.262	0.833	66.20	0.525	-0.158	0.864
14	Albania	2010	5.269	9.303	0.733	66.40	0.569	-0.172	0.726
15	Albania	2011	5.867	9.331	0.759	66.68	0.487	-0.205	0.877
16	Albania	2012	5.510	9.347	0.785	66.96	0.602	-0.169	0.848
17	Albania	2013	4.551	9.359	0.759	67.24	0.632	-0.127	0.863
18	Albania	2014	4.814	9.378	0.626	67.52	0.735	-0.025	0.883
19	Albania	2015	4.607	9.403	0.639	67.80	0.704	-0.081	0.885

Part 1

Summarize the data. How much data is present? What attributes/features are continuous valued? Which attributes are categorical?

```
In [3]: 1 # Extracting the number of rows and columns of the dataframe
2
3 print(len(happiness_df))
4 print(len(happiness_df.columns))
```

1949

11

```
In [4]: 1 happiness_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1949 entries, 0 to 1948
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Country name    1949 non-null   object  
 1   year             1949 non-null   int64  
 2   Life Ladder      1949 non-null   float64 
 3   Log GDP per capita  1913 non-null   float64 
 4   Social support   1936 non-null   float64 
 5   Healthy life expectancy at birth 1894 non-null   float64 
 6   Freedom to make life choices  1917 non-null   float64 
 7   Generosity       1860 non-null   float64 
 8   Perceptions of corruption  1839 non-null   float64 
 9   Positive affect   1927 non-null   float64 
 10  Negative affect   1933 non-null   float64 
dtypes: float64(9), int64(1), object(1)
memory usage: 167.6+ KB
```

Answer:- The dataframe has a size of 1949 rows and 11 columns i.e. there are 1949 data entries and 11 attributes/features. There are 9 continuous valued attributes which are ['Life Ladder', 'Log GDP per capita', 'Social support', 'Healthy life expectancy at birth', 'Freedom to make life choices', 'Generosity', 'Perceptions of corruption', 'Positive affect', 'Negative affect'] and 2 categorical attributes which are ['Country name', 'year']. Continuous values are those which can assume an infinite number of values within a given range. Categorical values are those which can assume values only from a given set. Depending on the context and dataset, 'year' can be treated as categorical or continuous.

Part 2

Display the statistical values for each of the attributes, along with visualizations (e.g. histogram) of the distributions for each attribute. Explain noticeable traits for key attributes. Are there any attributes that might require special treatment? If so, what special treatment might they require?

In [5]: 1 happiness_df.describe()

Out[5]:

	year	Life Ladder	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity
count	1949.000000	1949.000000	1913.000000	1936.000000	1894.000000	1917.000000	1860.0000
mean	2013.216008	5.466705	9.368453	0.812552	63.359374	0.742558	0.0001
std	4.166828	1.115711	1.154084	0.118482	7.510245	0.142093	0.1622
min	2005.000000	2.375000	6.635000	0.290000	32.300000	0.258000	-0.3350
25%	2010.000000	4.640000	8.464000	0.749750	58.685000	0.647000	-0.1130
50%	2013.000000	5.386000	9.460000	0.835500	65.200000	0.763000	-0.0255
75%	2017.000000	6.283000	10.353000	0.905000	68.590000	0.856000	0.0910
max	2020.000000	8.019000	11.648000	0.987000	77.100000	0.985000	0.6980

◀ ▶

In [6]: 1 happiness_df["Country name"].value_counts()

Out[6]:

Zimbabwe	15
South Africa	15
Tanzania	15
Denmark	15
Tajikistan	15
.	.
Maldives	1
Suriname	1
Cuba	1
Oman	1
Guyana	1

Name: Country name, Length: 166, dtype: int64

In [7]:

```

1 # Calculating the mean of each attribute for each unique country
2
3 happiness_df[["Country name", "Life Ladder", "Log GDP per capita", "Social support", "Healthy life expectancy at birth", "Freedom to make life choices", "Generosity", "Perceptions of corruption", "Positive affect"]].groupby("Country name").mean()
4
5

```

Out[7]:

Country name	Life Ladder	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect
Afghanistan	3.594667	7.650833	0.508417	52.266667	0.518167	0.070083	0.843333	0.54
Albania	5.019385	9.384385	0.716231	67.546154	0.662923	-0.082692	0.869385	0.65
Algeria	5.389875	9.328875	0.803571	65.290000	0.519167	-0.144333	0.691000	0.60
Angola	4.420250	8.990000	0.738250	53.550000	0.456250	-0.088250	0.866750	0.61
Argentina	6.310133	10.033800	0.904400	67.900000	0.768200	-0.159867	0.842067	0.83
...
Venezuela	6.019867	9.480083	0.910133	65.958667	0.666400	-0.164182	0.795200	0.80
Vietnam	5.305857	8.655286	0.829231	66.928571	0.882455	0.002615	0.784700	0.64
Yemen	3.912250	8.102700	0.739833	54.866667	0.622417	-0.126200	0.824667	0.53
Zambia	4.551714	8.066857	0.737714	51.707143	0.756786	0.013357	0.828857	0.72
Zimbabwe	3.882600	7.850333	0.799400	50.233333	0.587933	-0.083000	0.844200	0.71

166 rows × 9 columns



In [8]:

```

1 # Calculating the median of each attribute for each unique country
2
3 happiness_df[["Country name", "Life Ladder", "Log GDP per capita", "Social support", "Healthy life expectancy at birth", "Freedom to make life choices", "Generosity", "Perceptions of corruption", "Positive affect"]].groupby("Country name").median()
4
5

```

Out[8]:

Country name	Life Ladder	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect
Afghanistan	3.7535	7.6970	0.5210	52.48	0.5160	0.0920	0.8605	0.5595
Albania	4.9950	9.3780	0.7100	67.52	0.7040	-0.0810	0.8770	0.6690
Algeria	5.3290	9.3360	0.8070	65.32	0.5565	-0.1695	0.6950	0.6040
Angola	4.1485	8.9985	0.7380	53.55	0.4330	-0.1200	0.8700	0.6185
Argentina	6.4240	10.0510	0.9020	67.84	0.7470	-0.1570	0.8510	0.8400
...
Venezuela	6.2580	9.5495	0.9110	65.82	0.6420	-0.1690	0.8130	0.8190
Vietnam	5.2960	8.6470	0.8320	66.90	0.8890	-0.0060	0.7885	0.6650
Yemen	4.0145	8.2250	0.7415	54.30	0.6410	-0.1490	0.8290	0.5250
Zambia	4.7770	8.1240	0.7525	52.69	0.7645	0.0055	0.8105	0.7145
Zimbabwe	3.7350	7.9500	0.7990	50.96	0.6330	-0.0880	0.8310	0.7150

166 rows × 9 columns

In [9]:

```
1 happiness_df.skew()
```

C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\4122175689.py:1: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with 'numeric_only =None') is deprecated; in a future version this will raise TypeError. Select only valid columns before calling the reduction.

```
happiness_df.skew()
```

Out[9]:

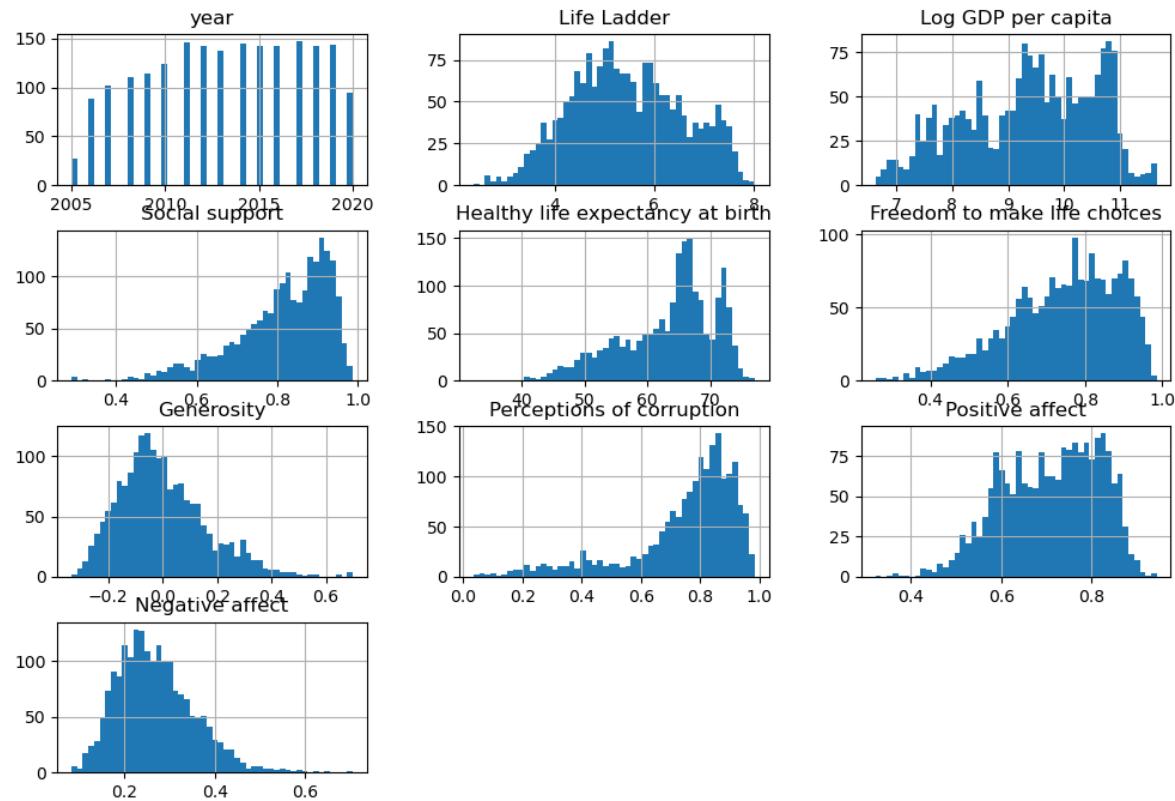
year	-0.128238
Life Ladder	0.068483
Log GDP per capita	-0.308453
Social support	-1.110682
Healthy life expectancy at birth	-0.744426
Freedom to make life choices	-0.623019
Generosity	0.807140
Perceptions of corruption	-1.496045
Positive affect	-0.364197
Negative affect	0.737166
dtype: float64	

In [10]: # Displaying the histogram for each attributes

```

1
2
3 happiness_df.hist(bins=50, figsize=(12, 8))
4 plt.show()

```



In [11]: # Checking for null values

```

1
2
3 print(happiness_df["year"].isnull().values.any())
4 print(happiness_df["Life Ladder"].isnull().values.any())
5 print(happiness_df["Log GDP per capita"].isnull().values.any())
6 print(happiness_df["Social support"].isnull().values.any())
7 print(happiness_df["Healthy life expectancy at birth"].isnull().values.any())
8 print(happiness_df["Freedom to make life choices"].isnull().values.any())
9 print(happiness_df["Generosity"].isnull().values.any())
10 print(happiness_df["Perceptions of corruption"].isnull().values.any())
11 print(happiness_df["Positive affect"].isnull().values.any())
12 print(happiness_df["Negative affect"].isnull().values.any())

```

```

False
False
True
True
True
True
True
True
True
True

```

Answer:- If we closely analyze the histograms and skewness values, these are the inferences:-

Log GDP per Capita: Its skewness value is -0.3, hence we can say the values are symmetrically distributed. The histogram confirms that.

Social support: The histogram is left skewed with a staggered bell shaped curve.

Healthy life expectancy at birth: The histogram is slightly left skewed since it has a skewness value of -0.7.

Freedom to make life choices: The histogram is slightly left skewed, but overall it can be considered symmetrically distributed.

Generosity: The histogram is right skewed with a staggered bell shaped curve.

Perceptions of corruption: The histogram is highly left skewed.

Positive affect: The values are symmetrically distributed.

Negative affect: The histogram is slightly right skewed.

All attributes except year and Life Ladder have null values in them. These attributes need special treatment and the null values need to be tackled. They can be either imputed or dropped. We will drop them.

Part 3

Analyze and discuss the relationships between the data attributes, and between the data attributes and label. This involves computing the Pearson Correlation Coefficient (PCC) and generating scatter plots.

```
In [12]: 1 # Finding the Pearson Correlation Coefficient (PCC) between the attributes
          2
          3 happiness_df.drop(columns=['Country name', 'year']).corr(method = 'pearson')
```

Out[12]:

	Life Ladder	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	P
Life Ladder	1.000000	0.790166	0.707806	0.744506	0.528063	0.190632	-0.427245	0.100000
Log GDP per capita	0.790166	1.000000	0.692602	0.848049	0.367932	-0.000915	-0.345511	0.100000
Social support	0.707806	0.692602	1.000000	0.616037	0.410402	0.067000	-0.219040	0.100000
Healthy life expectancy at birth	0.744506	0.848049	0.616037	1.000000	0.388681	0.020737	-0.322461	0.100000
Freedom to make life choices	0.528063	0.367932	0.410402	0.388681	1.000000	0.329300	-0.487883	0.100000
Generosity	0.190632	-0.000915	0.067000	0.020737	0.329300	1.000000	-0.290706	0.100000
Perceptions of corruption	-0.427245	-0.345511	-0.219040	-0.322461	-0.487883	-0.290706	1.000000	-0.100000
Positive affect	0.532273	0.302282	0.432152	0.318247	0.606114	0.358006	-0.296517	1.000000
Negative affect	-0.297488	-0.210781	-0.395865	-0.139477	-0.267661	-0.092542	0.264225	-0.100000

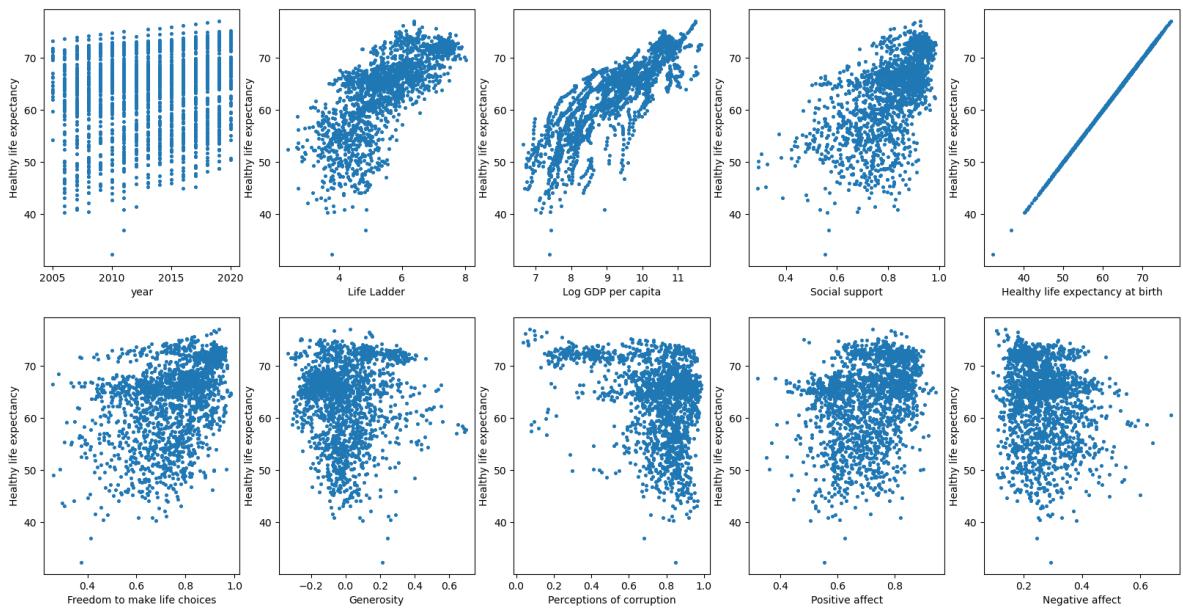


In [13]:

```

1 # Scatter plots of attributes versus output label
2
3 col = ['year','Life Ladder','Log GDP per capita','Social support','Healthy life expectancy','Generosity','Perceptions of corruption','Positive affect','Negative affect']
4
5 fig, ax = plt.subplots(2,int(len(col)/2) , figsize=(20,10))
6 j = 0
7 k = 0
8 for i in range(len(col)):
9     if(i == (len(col )/2)):
10         j = 0
11         k = 1
12     # scatter plot
13     ax[k,j].scatter(x=happiness_df[col [i]],y=happiness_df['Healthy life expectancy'])
14     ax[k,j].set_xlabel(col [i])
15     ax[k,j].set_ylabel('Healthy life expectancy')
16     j = j + 1
17

```



Answer:- From the correlation chart, we can say that attributes such as Life ladder, Log GDP per capita, Social support, Freedom to make life choices, Generosity and Positive affect have a positive correlation with the output label Healthy life expectancy at birth. Attributes like Perceptions of corruption, Negative affect have a negative correlation with the output label. Furthermore we can dissect Life ladder, Log GDP per capita, Social support have a high positive correlation and Freedom to make life choices, Generosity and Positive affect have low positive correlation.

Part 4

Select 20% of the data for testing. Describe how you did that and verify that your test portion of the data is representative of the entire dataset.

In [14]:

```

1 # Removing unwanted columns and NULL values
2
3 happiness_df_trimmed = happiness_df.drop(columns=['Country name', 'year',
4 happiness_df_trimmed = happiness_df_trimmed.dropna()
5 happiness_df_trimmed.head()

```

Out[14]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negative affect
0	7.370	0.451	50.80	0.718	0.168	0.882	0.518	0.258
1	7.540	0.552	51.20	0.679	0.190	0.850	0.584	0.237
2	7.647	0.539	51.60	0.600	0.121	0.707	0.618	0.275
3	7.620	0.521	51.92	0.496	0.162	0.731	0.611	0.267
4	7.705	0.521	52.24	0.531	0.236	0.776	0.710	0.268

In [15]:

```

1 # Converting 'Log GDP per capita' to a categorical type attribute 'GDP_Cat'
2
3 happiness_df_trimmed['GDP_Cat'] = pd.cut(happiness_df_trimmed['Log GDP per
4

```

In [16]:

```

1 # Splitting the data into training and testing using randomized method and stratified sampling
2
3 x = happiness_df_trimmed.drop('Healthy life expectancy at birth', axis=1)
4 y = happiness_df_trimmed['Healthy life expectancy at birth']
5
6 # We are using the newly created 'GDP_Cat' as our stratified sampling bins
7 y_strat_bin = happiness_df_trimmed['GDP_Cat']
8
9 # Randomized
10 x_train_1, x_test_1, y_train_1, y_test_1 = train_test_split(x, y, test_size=0.2, random_state=42, stratify=y_strat_bin)
11
12 # Stratified
13 x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(x, y, test_size=0.2, random_state=42, stratify=y)
14
15 print(x_train_1.shape, y_train_1.shape, x_test_1.shape, y_test_1.shape)

```

(1366, 8) (1366,) (342, 8) (342,)

In [17]:

```

1 # Distribution ratio of 'GDP_Cat' in the overall dataset
2 dist_overall = happiness_df_trimmed["GDP_Cat"].value_counts() / len(happiness_df_trimmed)
3
4 # Distribution ratio of 'GDP_Cat' in the randomized test dataset
5 dist_randomized = x_test_1["GDP_Cat"].value_counts() / len(x_test_1)
6
7 # Distribution ratio of 'GDP_Cat' in the stratified test dataset
8 dist_stratified = x_test_2["GDP_Cat"].value_counts() / len(x_test_2)
9
10 compare = pd.DataFrame({"Overall":dist_overall,"Randomized":dist_randomized,"Stratified":dist_stratified})
11 compare

```

Out[17]:

	Overall	Randomized	Stratified
1	0.170960	0.143275	0.172515
2	0.497658	0.505848	0.497076
3	0.331382	0.350877	0.330409

Answer:- Before train test split, we dropped the unwanted columns and also tackled the NULL values in the dataset. Then we created our test dataset using 2 methods: randomized and stratified. For randomized, we simply used the `test_train_split` function. For stratified, we first create a categorical attribute which can be used as stratas. We choose 'Log GDP per capita' attribute since it has the highest Pearson Correlation Coefficient with our target variable. We divide the 'Log GDP per capita' column into 3 labels 1,2,3 based on ranges. Then we compare the label distributions of the overall dataset, randomized test dataset and stratified test dataset. We see that stratified method produces better results than randomized methods. It produces a test set which is representative of the entire dataset.

Part 5 FIX!

Added learning rates to SGD and plotted the loss vs epochs graphs

Train a Linear Regression model using the training data with four-fold cross-validation using appropriate evaluation metric. Do this with closed form solution (using the Normal Equation or SVD) and with SGD. Perform Ridge, Lasso and Elastic Net regularization – try a few values of penalty term and describe its impact. Explore the impact of other hyperparameters, like batch size and learning rate (no need for grid search). Describe your findings. Display the training and validation loss as a function of training iterations.

In [18]:

```

1 # Converting training and test sets into numpy arrays
2
3 x_train_2, x_test_2, y_train_2, y_test_2 = np.array(x_train_2), np.array(y_train_2),
4 x_train_1, x_test_1, y_train_1, y_test_1 = np.array(x_train_1), np.array(y_train_1),
5
6 # Splitting the data into training and testing sets
7
8 # Training the model using SGD
9
10 # Evaluating the model
11
12 # Plotting the loss vs epochs graphs
13
14 # Displaying the results
15
16 # Saving the model
17
18 # Testing the model
19
20 # Displaying the results
21
22 # Saving the model
23
24 # Testing the model
25
26 # Displaying the results
27
28 # Saving the model
29
30 # Testing the model
31
32 # Displaying the results
33
34 # Saving the model
35
36 # Testing the model
37
38 # Displaying the results
39
40 # Saving the model
41
42 # Testing the model
43
44 # Displaying the results
45
46 # Saving the model
47
48 # Testing the model
49
50 # Displaying the results
51
52 # Saving the model
53
54 # Testing the model
55
56 # Displaying the results
57
58 # Saving the model
59
60 # Testing the model
61
62 # Displaying the results
63
64 # Saving the model
65
66 # Testing the model
67
68 # Displaying the results
69
70 # Saving the model
71
72 # Testing the model
73
74 # Displaying the results
75
76 # Saving the model
77
78 # Testing the model
79
80 # Displaying the results
81
82 # Saving the model
83
84 # Testing the model
85
86 # Displaying the results
87
88 # Saving the model
89
90 # Testing the model
91
92 # Displaying the results
93
94 # Saving the model
95
96 # Testing the model
97
98 # Displaying the results
99
100 # Saving the model
101
102 # Testing the model
103
104 # Displaying the results
105
106 # Saving the model
107
108 # Testing the model
109
110 # Displaying the results
111
112 # Saving the model
113
114 # Testing the model
115
116 # Displaying the results
117
118 # Saving the model
119
120 # Testing the model
121
122 # Displaying the results
123
124 # Saving the model
125
126 # Testing the model
127
128 # Displaying the results
129
130 # Saving the model
131
132 # Testing the model
133
134 # Displaying the results
135
136 # Saving the model
137
138 # Testing the model
139
140 # Displaying the results
141
142 # Saving the model
143
144 # Testing the model
145
146 # Displaying the results
147
148 # Saving the model
149
150 # Testing the model
151
152 # Displaying the results
153
154 # Saving the model
155
156 # Testing the model
157
158 # Displaying the results
159
160 # Saving the model
161
162 # Testing the model
163
164 # Displaying the results
165
166 # Saving the model
167
168 # Testing the model
169
170 # Displaying the results
171
172 # Saving the model
173
174 # Testing the model
175
176 # Displaying the results
177
178 # Saving the model
179
180 # Testing the model
181
182 # Displaying the results
183
184 # Saving the model
185
186 # Testing the model
187
188 # Displaying the results
189
190 # Saving the model
191
192 # Testing the model
193
194 # Displaying the results
195
196 # Saving the model
197
198 # Testing the model
199
200 # Displaying the results
201
202 # Saving the model
203
204 # Testing the model
205
206 # Displaying the results
207
208 # Saving the model
209
210 # Testing the model
211
212 # Displaying the results
213
214 # Saving the model
215
216 # Testing the model
217
218 # Displaying the results
219
220 # Saving the model
221
222 # Testing the model
223
224 # Displaying the results
225
226 # Saving the model
227
228 # Testing the model
229
230 # Displaying the results
231
232 # Saving the model
233
234 # Testing the model
235
236 # Displaying the results
237
238 # Saving the model
239
240 # Testing the model
241
242 # Displaying the results
243
244 # Saving the model
245
246 # Testing the model
247
248 # Displaying the results
249
250 # Saving the model
251
252 # Testing the model
253
254 # Displaying the results
255
256 # Saving the model
257
258 # Testing the model
259
260 # Displaying the results
261
262 # Saving the model
263
264 # Testing the model
265
266 # Displaying the results
267
268 # Saving the model
269
270 # Testing the model
271
272 # Displaying the results
273
274 # Saving the model
275
276 # Testing the model
277
278 # Displaying the results
279
280 # Saving the model
281
282 # Testing the model
283
284 # Displaying the results
285
286 # Saving the model
287
288 # Testing the model
289
290 # Displaying the results
291
292 # Saving the model
293
294 # Testing the model
295
296 # Displaying the results
297
298 # Saving the model
299
300 # Testing the model
301
302 # Displaying the results
303
304 # Saving the model
305
306 # Testing the model
307
308 # Displaying the results
309
310 # Saving the model
311
312 # Testing the model
313
314 # Displaying the results
315
316 # Saving the model
317
318 # Testing the model
319
320 # Displaying the results
321
322 # Saving the model
323
324 # Testing the model
325
326 # Displaying the results
327
328 # Saving the model
329
330 # Testing the model
331
332 # Displaying the results
333
334 # Saving the model
335
336 # Testing the model
337
338 # Displaying the results
339
340 # Saving the model
341
342 # Testing the model
343
344 # Displaying the results
345
346 # Saving the model
347
348 # Testing the model
349
350 # Displaying the results
351
352 # Saving the model
353
354 # Testing the model
355
356 # Displaying the results
357
358 # Saving the model
359
360 # Testing the model
361
362 # Displaying the results
363
364 # Saving the model
365
366 # Testing the model
367
368 # Displaying the results
369
370 # Saving the model
371
372 # Testing the model
373
374 # Displaying the results
375
376 # Saving the model
377
378 # Testing the model
379
380 # Displaying the results
381
382 # Saving the model
383
384 # Testing the model
385
386 # Displaying the results
387
388 # Saving the model
389
390 # Testing the model
391
392 # Displaying the results
393
394 # Saving the model
395
396 # Testing the model
397
398 # Displaying the results
399
400 # Saving the model
401
402 # Testing the model
403
404 # Displaying the results
405
406 # Saving the model
407
408 # Testing the model
409
410 # Displaying the results
411
412 # Saving the model
413
414 # Testing the model
415
416 # Displaying the results
417
418 # Saving the model
419
420 # Testing the model
421
422 # Displaying the results
423
424 # Saving the model
425
426 # Testing the model
427
428 # Displaying the results
429
430 # Saving the model
431
432 # Testing the model
433
434 # Displaying the results
435
436 # Saving the model
437
438 # Testing the model
439
440 # Displaying the results
441
442 # Saving the model
443
444 # Testing the model
445
446 # Displaying the results
447
448 # Saving the model
449
450 # Testing the model
451
452 # Displaying the results
453
454 # Saving the model
455
456 # Testing the model
457
458 # Displaying the results
459
460 # Saving the model
461
462 # Testing the model
463
464 # Displaying the results
465
466 # Saving the model
467
468 # Testing the model
469
470 # Displaying the results
471
472 # Saving the model
473
474 # Testing the model
475
476 # Displaying the results
477
478 # Saving the model
479
480 # Testing the model
481
482 # Displaying the results
483
484 # Saving the model
485
486 # Testing the model
487
488 # Displaying the results
489
490 # Saving the model
491
492 # Testing the model
493
494 # Displaying the results
495
496 # Saving the model
497
498 # Testing the model
499
500 # Displaying the results
501
502 # Saving the model
503
504 # Testing the model
505
506 # Displaying the results
507
508 # Saving the model
509
510 # Testing the model
511
512 # Displaying the results
513
514 # Saving the model
515
516 # Testing the model
517
518 # Displaying the results
519
520 # Saving the model
521
522 # Testing the model
523
524 # Displaying the results
525
526 # Saving the model
527
528 # Testing the model
529
530 # Displaying the results
531
532 # Saving the model
533
534 # Testing the model
535
536 # Displaying the results
537
538 # Saving the model
539
540 # Testing the model
541
542 # Displaying the results
543
544 # Saving the model
545
546 # Testing the model
547
548 # Displaying the results
549
550 # Saving the model
551
552 # Testing the model
553
554 # Displaying the results
555
556 # Saving the model
557
558 # Testing the model
559
560 # Displaying the results
561
562 # Saving the model
563
564 # Testing the model
565
566 # Displaying the results
567
568 # Saving the model
569
570 # Testing the model
571
572 # Displaying the results
573
574 # Saving the model
575
576 # Testing the model
577
578 # Displaying the results
579
580 # Saving the model
581
582 # Testing the model
583
584 # Displaying the results
585
586 # Saving the model
587
588 # Testing the model
589
590 # Displaying the results
591
592 # Saving the model
593
594 # Testing the model
595
596 # Displaying the results
597
598 # Saving the model
599
600 # Testing the model
601
602 # Displaying the results
603
604 # Saving the model
605
606 # Testing the model
607
608 # Displaying the results
609
610 # Saving the model
611
612 # Testing the model
613
614 # Displaying the results
615
616 # Saving the model
617
618 # Testing the model
619
620 # Displaying the results
621
622 # Saving the model
623
624 # Testing the model
625
626 # Displaying the results
627
628 # Saving the model
629
630 # Testing the model
631
632 # Displaying the results
633
634 # Saving the model
635
636 # Testing the model
637
638 # Displaying the results
639
640 # Saving the model
641
642 # Testing the model
643
644 # Displaying the results
645
646 # Saving the model
647
648 # Testing the model
649
650 # Displaying the results
651
652 # Saving the model
653
654 # Testing the model
655
656 # Displaying the results
657
658 # Saving the model
659
660 # Testing the model
661
662 # Displaying the results
663
664 # Saving the model
665
666 # Testing the model
667
668 # Displaying the results
669
670 # Saving the model
671
672 # Testing the model
673
674 # Displaying the results
675
676 # Saving the model
677
678 # Testing the model
679
680 # Displaying the results
681
682 # Saving the model
683
684 # Testing the model
685
686 # Displaying the results
687
688 # Saving the model
689
690 # Testing the model
691
692 # Displaying the results
693
694 # Saving the model
695
696 # Testing the model
697
698 # Displaying the results
699
700 # Saving the model
701
702 # Testing the model
703
704 # Displaying the results
705
706 # Saving the model
707
708 # Testing the model
709
710 # Displaying the results
711
712 # Saving the model
713
714 # Testing the model
715
716 # Displaying the results
717
718 # Saving the model
719
720 # Testing the model
721
722 # Displaying the results
723
724 # Saving the model
725
726 # Testing the model
727
728 # Displaying the results
729
730 # Saving the model
731
732 # Testing the model
733
734 # Displaying the results
735
736 # Saving the model
737
738 # Testing the model
739
740 # Displaying the results
741
742 # Saving the model
743
744 # Testing the model
745
746 # Displaying the results
747
748 # Saving the model
749
750 # Testing the model
751
752 # Displaying the results
753
754 # Saving the model
755
756 # Testing the model
757
758 # Displaying the results
759
760 # Saving the model
761
762 # Testing the model
763
764 # Displaying the results
765
766 # Saving the model
767
768 # Testing the model
769
770 # Displaying the results
771
772 # Saving the model
773
774 # Testing the model
775
776 # Displaying the results
777
778 # Saving the model
779
780 # Testing the model
781
782 # Displaying the results
783
784 # Saving the model
785
786 # Testing the model
787
788 # Displaying the results
789
790 # Saving the model
791
792 # Testing the model
793
794 # Displaying the results
795
796 # Saving the model
797
798 # Testing the model
799
800 # Displaying the results
801
802 # Saving the model
803
804 # Testing the model
805
806 # Displaying the results
807
808 # Saving the model
809
810 # Testing the model
811
812 # Displaying the results
813
814 # Saving the model
815
816 # Testing the model
817
818 # Displaying the results
819
820 # Saving the model
821
822 # Testing the model
823
824 # Displaying the results
825
826 # Saving the model
827
828 # Testing the model
829
830 # Displaying the results
831
832 # Saving the model
833
834 # Testing the model
835
836 # Displaying the results
837
838 # Saving the model
839
840 # Testing the model
841
842 # Displaying the results
843
844 # Saving the model
845
846 # Testing the model
847
848 # Displaying the results
849
850 # Saving the model
851
852 # Testing the model
853
854 # Displaying the results
855
856 # Saving the model
857
858 # Testing the model
859
860 # Displaying the results
861
862 # Saving the model
863
864 # Testing the model
865
866 # Displaying the results
867
868 # Saving the model
869
870 # Testing the model
871
872 # Displaying the results
873
874 # Saving the model
875
876 # Testing the model
877
878 # Displaying the results
879
880 # Saving the model
881
882 # Testing the model
883
884 # Displaying the results
885
886 # Saving the model
887
888 # Testing the model
889
890 # Displaying the results
891
892 # Saving the model
893
894 # Testing the model
895
896 # Displaying the results
897
898 # Saving the model
899
900 # Testing the model
901
902 # Displaying the results
903
904 # Saving the model
905
906 # Testing the model
907
908 # Displaying the results
909
910 # Saving the model
911
912 # Testing the model
913
914 # Displaying the results
915
916 # Saving the model
917
918 # Testing the model
919
920 # Displaying the results
921
922 # Saving the model
923
924 # Testing the model
925
926 # Displaying the results
927
928 # Saving the model
929
930 # Testing the model
931
932 # Displaying the results
933
934 # Saving the model
935
936 # Testing the model
937
938 # Displaying the results
939
940 # Saving the model
941
942 # Testing the model
943
944 # Displaying the results
945
946 # Saving the model
947
948 # Testing the model
949
950 # Displaying the results
951
952 # Saving the model
953
954 # Testing the model
955
956 # Displaying the results
957
958 # Saving the model
959
960 # Testing the model
961
962 # Displaying the results
963
964 # Saving the model
965
966 # Testing the model
967
968 # Displaying the results
969
970 # Saving the model
971
972 # Testing the model
973
974 # Displaying the results
975
976 # Saving the model
977
978 # Testing the model
979
980 # Displaying the results
981
982 # Saving the model
983
984 # Testing the model
985
986 # Displaying the results
987
988 # Saving the model
989
990 # Testing the model
991
992 # Displaying the results
993
994 # Saving the model
995
996 # Testing the model
997
998 # Displaying the results
999
1000 # Saving the model
1001
1002 # Testing the model
1003
1004 # Displaying the results
1005
1006 # Saving the model
1007
1008 # Testing the model
1009
1010 # Displaying the results
1011
1012 # Saving the model
1013
1014 # Testing the model
1015
1016 # Displaying the results
1017
1018 # Saving the model
1019
1020 # Testing the model
1021
1022 # Displaying the results
1023
1024 # Saving the model
1025
1026 # Testing the model
1027
1028 # Displaying the results
1029
1030 # Saving the model
1031
1032 # Testing the model
1033
1034 # Displaying the results
1035
1036 # Saving the model
1037
1038 # Testing the model
1039
1040 # Displaying the results
1041
1042 # Saving the model
1043
1044 # Testing the model
1045
1046 # Displaying the results
1047
1048 # Saving the model
1049
1050 # Testing the model
1051
1052 # Displaying the results
1053
1054 # Saving the model
1055
1056 # Testing the model
1057
1058 # Displaying the results
1059
1060 # Saving the model
1061
1062 # Testing the model
1063
1064 # Displaying the results
1065
1066 # Saving the model
1067
1068 # Testing the model
1069
1070 # Displaying the results
1071
1072 # Saving the model
1073
1074 # Testing the model
1075
1076 # Displaying the results
1077
1078 # Saving the model
1079
1080 # Testing the model
1081
1082 # Displaying the results
1083
1084 # Saving the model
1085
1086 # Testing the model
1087
1088 # Displaying the results
1089
1090 # Saving the model
1091
1092 # Testing the model
1093
1094 # Displaying the results
1095
1096 # Saving the model
1097
1098 # Testing the model
1099
1100 # Displaying the results
1101
1102 # Saving the model
1103
1104 # Testing the model
1105
1106 # Displaying the results
1107
1108 # Saving the model
1109
1110 # Testing the model
1111
1112 # Displaying the results
1113
1114 # Saving the model
1115
1116 # Testing the model
1117
1118 # Displaying the results
1119
1120 # Saving the model
1121
1122 # Testing the model
1123
1124 # Displaying the results
1125
1126 # Saving the model
1127
1128 # Testing the model
1129
1130 # Displaying the results
1131
1132 # Saving the model
1133
1134 # Testing the model
1135
1136 # Displaying the results
1137
1138 # Saving the model
1139
1140 # Testing the model
1141
1142 # Displaying the results
1143
1144 # Saving the model
1145
1146 # Testing the model
1147
1148 # Displaying the results
1149
1150 # Saving the model
1151
1152 # Testing the model
1153
1154 # Displaying the results
1155
1156 # Saving the model
1157
1158 # Testing the model
1159
1160 # Displaying the results
1161
1162 # Saving the model
1163
1164 # Testing the model
1165
1166 # Displaying the results
1167
1168 # Saving the model
1169
1170 # Testing the model
1171
1172 # Displaying the results
1173
1174 # Saving the model
1175
1176 # Testing the model
1177
1178 # Displaying the results
1179
1180 # Saving the model
1181
1182 # Testing the model
1183
1184 # Displaying the results
1185
1186 # Saving the model
1187
1188 # Testing the model
1189
1190 # Displaying the results
1191
1192 # Saving the model
1193
1194 # Testing the model
1195
1196 # Displaying the results
1197
1198 # Saving the model
1199
1200 # Testing the model
1201
1202 # Displaying the results
1203
1204 # Saving the model
1205
1206 # Testing the model
1207
1208 # Displaying the results
1209
1210 # Saving the model
1211
1212 # Testing the model
1213
1214 # Displaying the results
1215
1216 # Saving the model
1217
1218 # Testing the model
1219
1220 # Displaying the results
1221
1222 # Saving the model
1223
1224 # Testing the model
1225
1226 # Displaying the results
1227
1228 # Saving the model
1229
1230 # Testing the model
1231
1232 # Displaying the results
1233
1234 # Saving the model
1235
1236 # Testing the model
1237
1238 # Displaying the results
1239
1240 # Saving the model
1241
1242 # Testing the model
1243
1244 # Displaying the results
1245
1246 # Saving the model
1247
1248 # Testing the model
1249
1250 # Displaying the results
1251
1252 # Saving the model
1253
1254 # Testing the model
1255
1256 # Displaying the results
1257
1258 # Saving the model
1259
1260 # Testing the model
1261
1262 # Displaying the results
1263
1264 # Saving the model
1265
1266 # Testing the model
1267
1268 # Displaying the results
1269
1270 # Saving the model
1271
1272 # Testing the model
1273
1274 # Displaying the results
1275
1276 # Saving the model
1277
1278 # Testing the model
1279
1280 # Displaying the results
1281
1282 # Saving the model
1283
1284 # Testing the model
1285
1286 # Displaying the results
1287
1288 # Saving the model
1289
1290 # Testing the model
1291
1292 # Displaying the results
1293
1294 # Saving the model
1295
1296 # Testing the model
1297
1298 # Displaying the results
1299
1300 # Saving the model
1301
1302 # Testing the model
1303
1304 # Displaying the results
1305
1306 # Saving the model
1307
1308 # Testing the model
1309
1310 # Displaying the results
1311
1312 # Saving the model
1313
1314 # Testing the model
1315
1316 # Displaying the results
1317
1318 # Saving the model
1319
1320 # Testing the model
1321
1322 # Displaying the results
1323
1324 # Saving the model
1325
1326 # Testing the model
1327
1328 # Displaying the results
1329
1330 # Saving the model
1331
1332 # Testing the model
1333
1334 # Displaying the results
1335
1336 # Saving the model
1337
1338 # Testing the model
1339
1340 # Displaying the results
1341
1342 # Saving the model
1343
1344 # Testing the model
1345
1346 # Displaying the results
1347
1348 # Saving the model
1349
1350 # Testing the model
1351
1352 # Displaying the results
1353
1354 # Saving the model
1355
1356 # Testing the model
1357
1358 # Displaying the results
1359
1360 # Saving the model
1361
1362 # Testing the model
1363
1364 # Displaying the results
1365
1366 # Saving the model
1367
1368 # Testing the model
1369
1370 # Displaying the results
1371
1372 # Saving the model
1373
1374 # Testing the model
1375
1376 # Displaying the results
1377
1378 # Saving the model
1379
1380 # Testing the model
1381
1382 # Displaying the results
1383
1384 # Saving the model
1385
1386 # Testing the model
1387
1388 # Displaying the results
1389
1390 # Saving the model
1391
1392 # Testing the model
1393
1394 # Displaying the results
1395
1396 # Saving the model
1397
1398 # Testing the model
1399
1400 # Displaying the results
1401
1402 # Saving the model
1403
1404 # Testing the model
1405
1406 # Displaying the results
1407
1408 # Saving the model
1409
1410 # Testing the model
1411
1412 # Displaying the results
1413
1414 # Saving the model
1415
1416 # Testing the model
1417
1418 # Displaying the results
1419
1420 # Saving the model
1421
1422 # Testing the model
1423
1424 # Displaying the results
1425
1426 # Saving the model
1427
1428 # Testing the model
1429
1430 # Displaying the results
1431
1432 # Saving the model
1433
1434 # Testing the model
1435
1436 # Displaying the results
1437
1438 # Saving the model
1439
1440 # Testing the model
1441
1442 # Displaying the results
1443
1444 # Saving the model
1445
1446 # Testing the model
1447
1448 # Displaying the results
1449
1450 # Saving the model
1451
1452 # Testing the model
1453
1454 # Displaying the results
1455
1456 # Saving the model
1457
1458 # Testing the model
1459
1460 # Displaying the results
1461
1462 # Saving the model
1463
1464 # Testing the model
1465
1466 # Displaying the results
1467
1468 # Saving the model
1469
1470 # Testing the model
1471
1472 # Displaying the results
1473
1474 # Saving the model
1475
1476 # Testing the model
1477
1478 # Displaying the results
1479
1480 # Saving the model
1481
1482 # Testing the model
1483
1484 # Displaying the results
1485
1486 # Saving the model
1487
1488 # Testing the model
1489
1490 # Displaying the results
1491
1492 # Saving the model
1493
1494 # Testing the model
1495
1496 # Displaying the results
1497
1498 # Saving the model
1499
1500 # Testing the model
1501
1502 # Displaying the results
1503
1504 # Saving the model
1505
1506 # Testing the model
1507
1508 # Displaying the results
1509
1510 # Saving the model
1511
1512 # Testing the model
1513
1514 # Displaying the results
1515
1516 # Saving the model
1517
1518 # Testing the model
1519
1520 # Displaying the results
1521
1522 # Saving the model
1523
1524 # Testing the model
1525
1526 # Displaying the results
1527
1528 # Saving the model
1529
1530 # Testing the model
1531
1532 # Displaying the results
1533
1534 # Saving the model
1535
1536 # Testing the model
1537
1538 # Displaying the results
1539
1540 # Saving the model
1541
1542 # Testing the model
1543
1544 # Displaying the results
1545
1546 # Saving the model
1547
1548 # Testing the model
1549
1550 # Displaying the results
1551
1552 # Saving the model
1553
1554 # Testing the model
1555
1556 # Displaying the results
1557
1558 # Saving the model
1559
1560 # Testing
```

In [19]:

```
1 # Linear regression model using a closed-form solution (Normal equation) i
2 # We are using RMSE evaluation metric.
3
4 def normal(x,y):
5     phi = np.hstack([x, np.ones((x.shape[0],1))])
6     phi_t_phi = np.matmul(phi.T, phi)
7     reg = 1*np.eye(phi_t_phi.shape[0])
8     inv_term = np.linalg.inv(reg+phi_t_phi)
9     w = np.matmul(inv_term,np.matmul(phi.T,y))
10    return w
11
12 def mse(y_pred,y_true):
13     return np.sum((y_true.reshape((y_true.shape[0],))-y_pred.reshape((y_tu
```

In [20]:

```

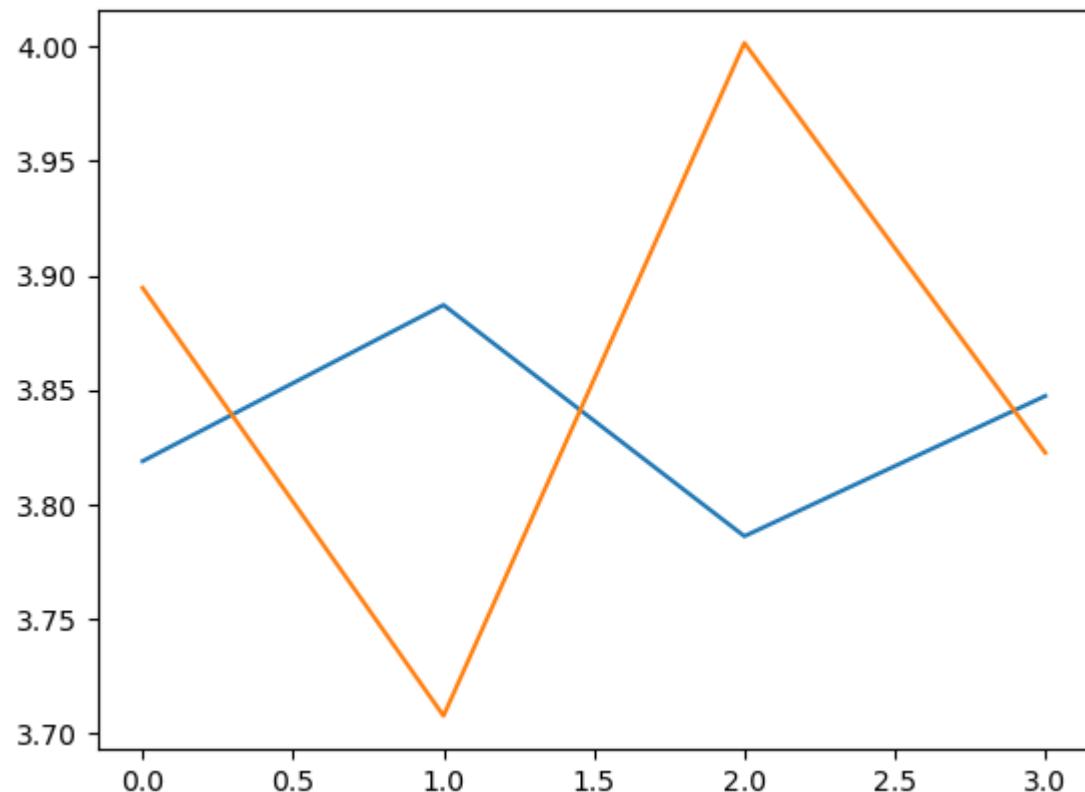
1 kf = KFold(n_splits = 4)
2
3 train_plot = []
4 val_plot = []
5
6 for cnt,var in enumerate(kf.split(x_train_2,y_train_2)) :
7     print(cnt)
8     x = x_train_2[var[0],:]
9     y = y_train_2[var[0]]
10    x_val = x_train_2[var[1],:]
11    y_val = y_train_2[var[1]]
12    w = normal(x,y)
13    phi = np.hstack([x, np.ones((x.shape[0],1))])
14    y_pred_train = np.matmul(phi,w.reshape(9,1))
15    phi_val = np.hstack([x_val, np.ones((x_val.shape[0],1))])
16    y_pred_val = np.matmul(phi_val,w.reshape(9,1),)
17    print("Train error:",np.sqrt(mse(y_pred_train,y)))
18    print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
19    train_plot.append(np.sqrt(mse(y_pred_train,y)))
20    val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
21
22 plt.plot([0,1,2,3], train_plot, label = "Training Error")
23 plt.plot([0,1,2,3], val_plot, label = "Val Error")
24 test_stack = np.hstack([x_test_2, np.ones((x_test_2.shape[0],1))])
25 y_pred_test = np.matmul(test_stack,w.reshape(9,1),)
26 metric_dict = {}
27 metric_dict['Linear regression'] = np.sqrt(mse(y_pred_test,y_test_2))

```

```

0
Train error: 3.818776080593649
Validation error: 3.8945433333753683
1
Train error: 3.8870500147623175
Validation error: 3.707428285591268
2
Train error: 3.7859531773863266
Validation error: 4.001585070020308
3
Train error: 3.8472813016461296
Validation error: 3.8224446443526774

```



In [21]:

```

1 # SGD with different learning rate
2
3 learning_rates = [0.000001, 0.00001, 0.0001, 0.001, 0.01]
4 n_epochs = 300
5
6 train_losses = np.zeros((len(learning_rates), n_epochs))
7 val_losses = np.zeros((len(learning_rates), n_epochs))
8 test_rmse_scores = []
9
10 for i, learning_rate in enumerate(learning_rates):
11     sgd = SGDRegressor(max_iter=1, warm_start=True, eta0=learning_rate, l
12     for epoch in range(n_epochs):
13         sgd.partial_fit(x_train_2, y_train_2)
14         y_pred_train = sgd.predict(x_train_2)
15         y_pred_val = sgd.predict(x_val)
16         train_losses[i, epoch] = np.sqrt(mse(y_pred_train, y_train_2))
17         val_losses[i, epoch] = np.sqrt(mse(y_pred_val, y_val))
18
19     # Plot train and val losses for each epoch and each learning rate sepa
20     fig, axs = plt.subplots(figsize=(10, 6))
21     fig.suptitle(f"Learning Rate: {learning_rate}")
22     axs.plot(range(1, n_epochs + 1), train_losses[i], label="Train Loss")
23     axs.plot(range(1, n_epochs + 1), val_losses[i], label="Val Loss")
24     axs.set_xlabel("Epoch")
25     axs.set_ylabel("RMSE")
26     axs.legend()
27
28     # Calculate RMSE score on test set
29     y_pred_test = sgd.predict(x_test_2)
30     test_rmse_score = np.sqrt(mse(y_pred_test, y_test_2))
31     metric_dict['SGD with learning rate ='+str(learning_rate)] = test_rmse_
32     print(f"RMSE score between y_pred_val and y_test for learning_rate={le
33     test_rmse_scores.append(test_rmse_score)
34
35 plt.show()
36

```

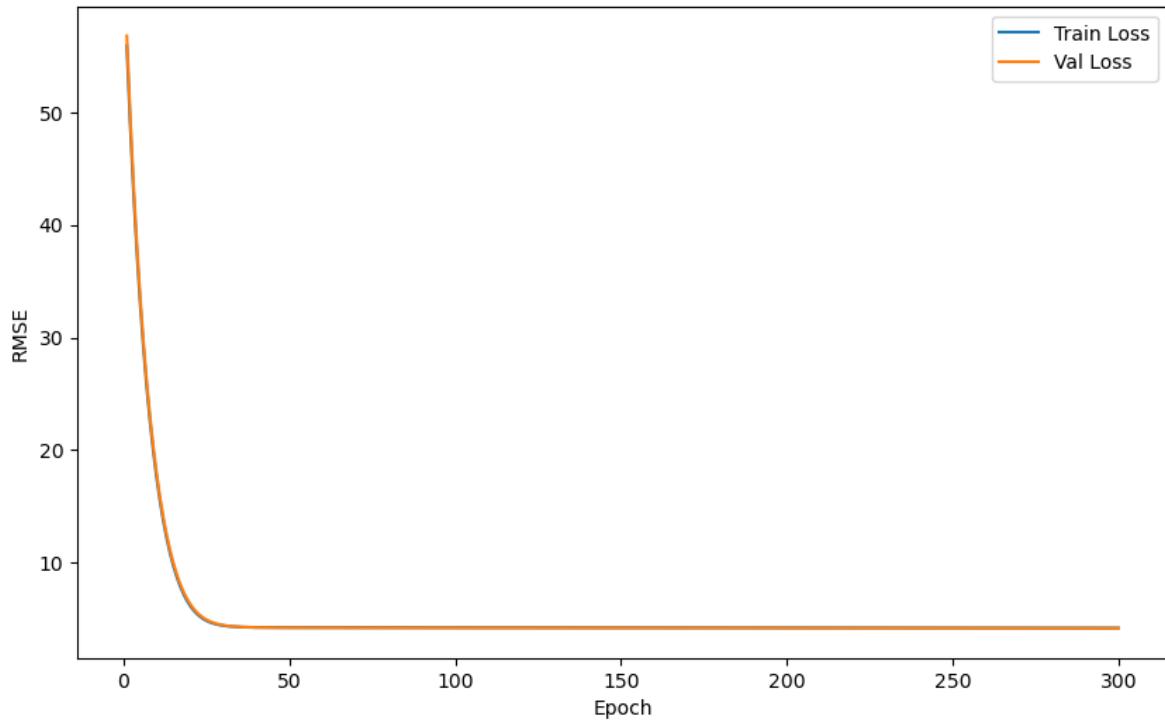
◀ ▶

```

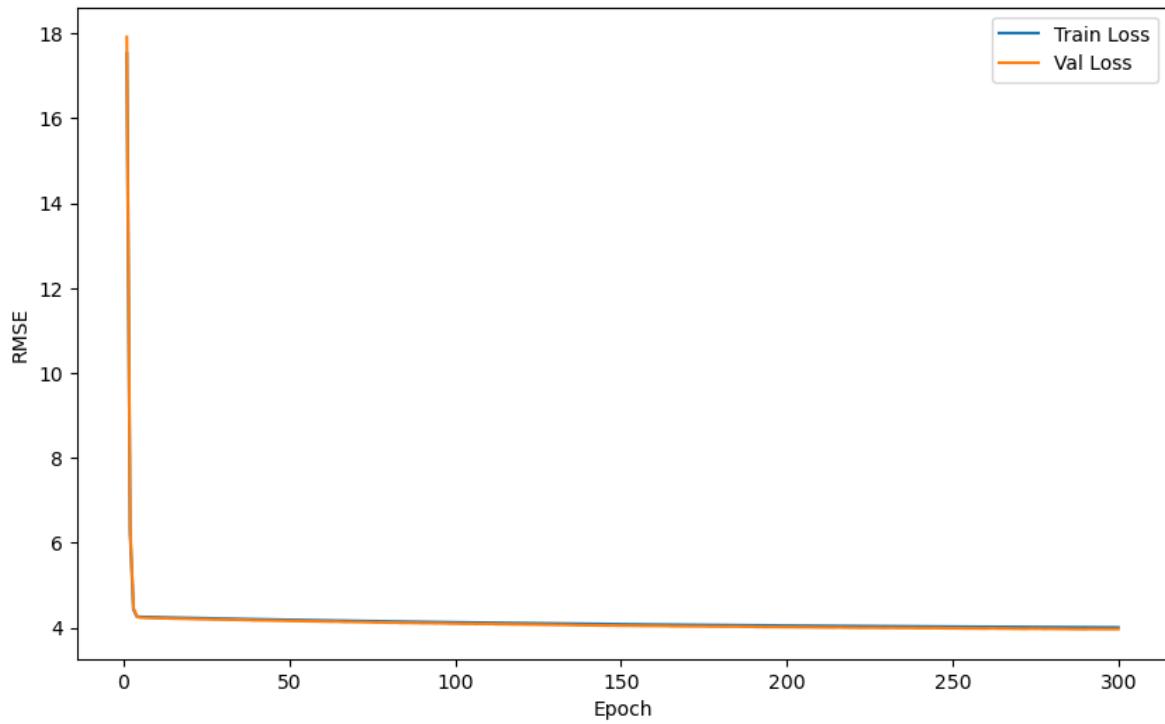
RMSE score between y_pred_val and y_test for learning_rate=1e-06: 3.969667490
79844
RMSE score between y_pred_val and y_test for learning_rate=1e-05: 3.767096303
4078917
RMSE score between y_pred_val and y_test for learning_rate=0.0001: 3.68929718
2286318
RMSE score between y_pred_val and y_test for learning_rate=0.001: 3.932298364
6063134
RMSE score between y_pred_val and y_test for learning_rate=0.01: 4.3745030436
89119

```

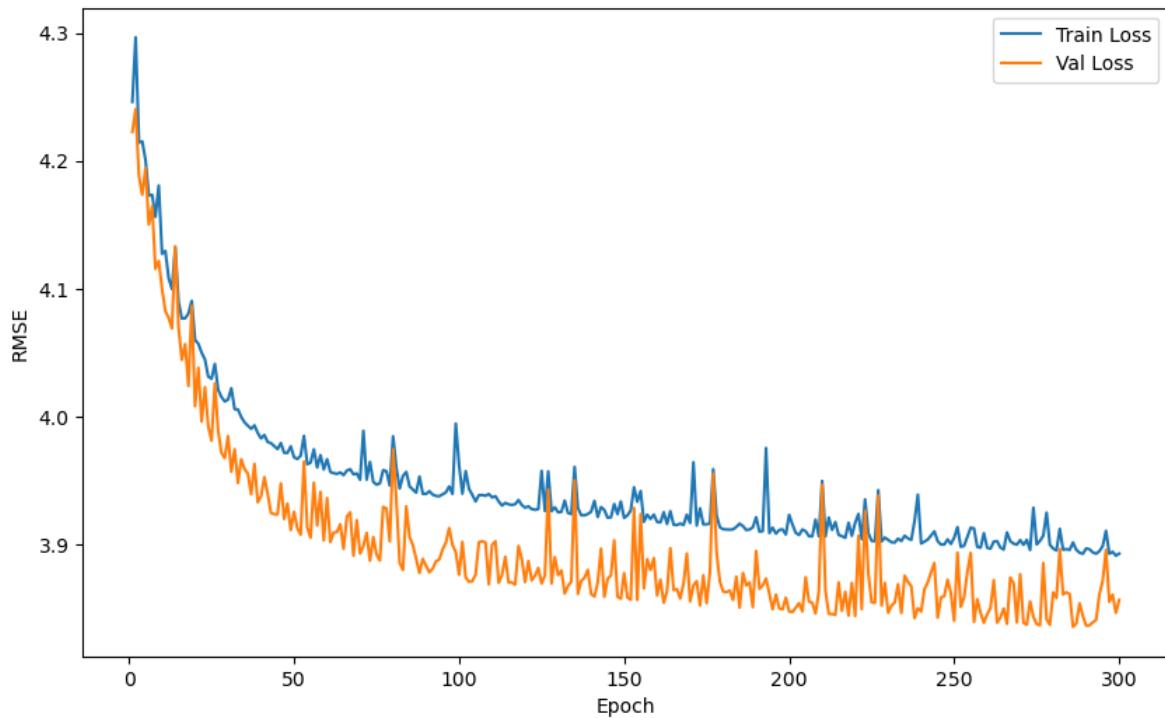
Learning Rate: 1e-06



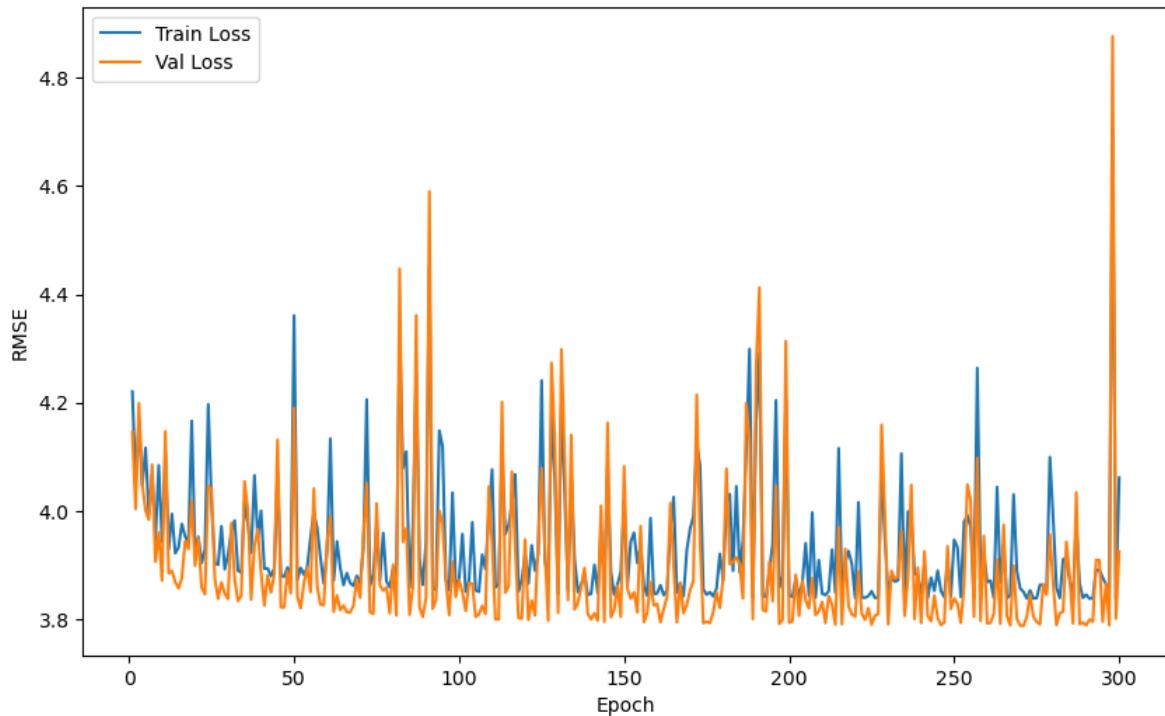
Learning Rate: 1e-05



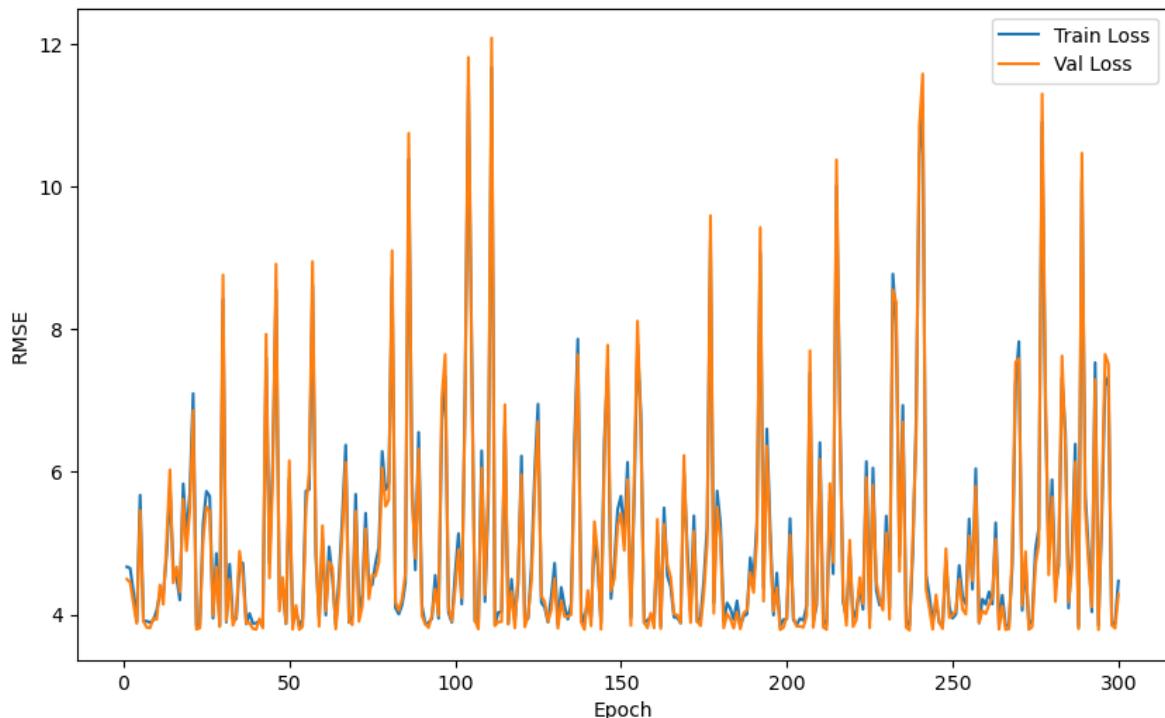
Learning Rate: 0.0001



Learning Rate: 0.001



Learning Rate: 0.01

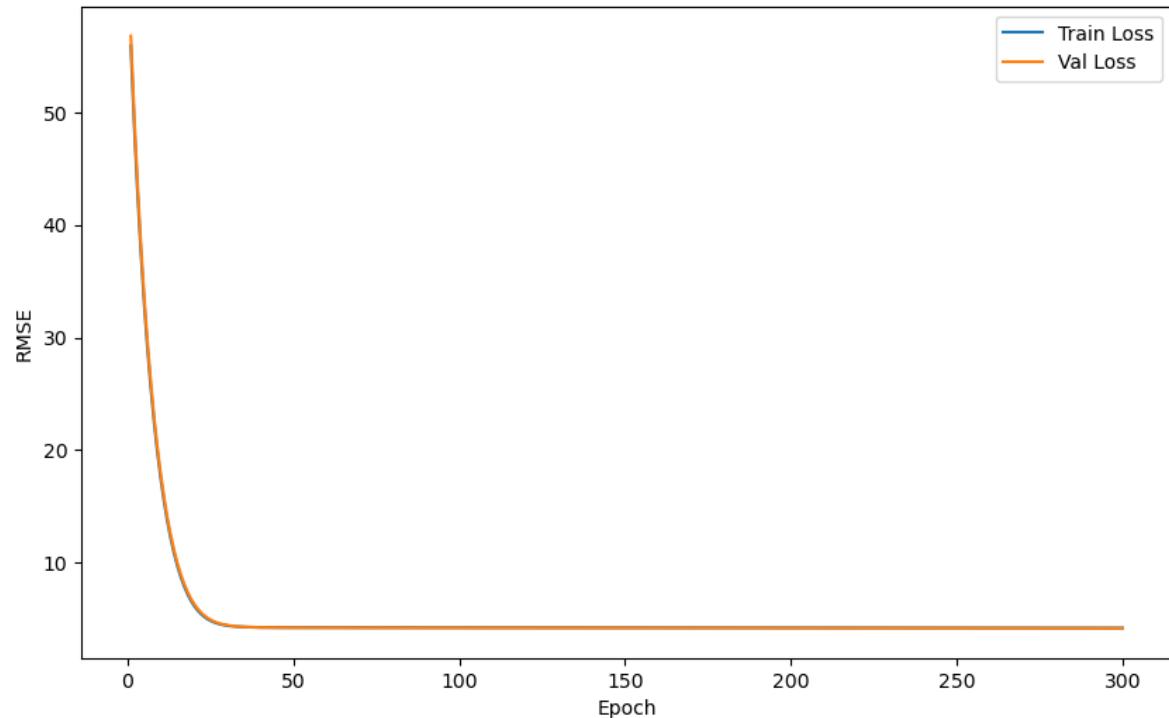


In [22]:

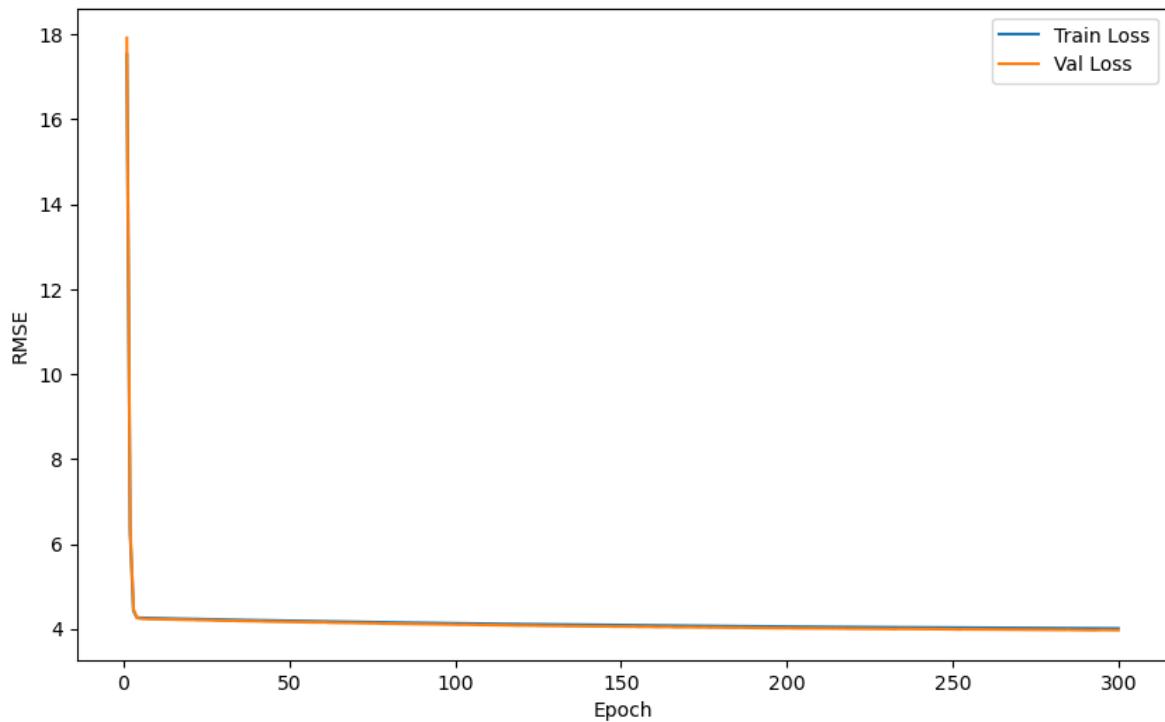
```
1 # SGD with different penalty terms
2
3 penalty_factors = [None, 'l1', 'l2']
4 learning_rates = [0.000001, 0.00001, 0.0001, 0.001, 0.01]
5 n_epochs = 300
6
7 for penalty in penalty_factors:
8     for i, learning_rate in enumerate(learning_rates):
9         sgd = SGDRegressor(max_iter=1, warm_start=True, eta0=learning_rate)
10        train_losses = np.zeros(n_epochs)
11        val_losses = np.zeros(n_epochs)
12        for epoch in range(n_epochs):
13            sgd.partial_fit(x_train_2, y_train_2)
14            y_pred_train = sgd.predict(x_train_2)
15            y_pred_val = sgd.predict(x_val)
16            train_losses[epoch] = np.sqrt(mse(y_pred_train, y_train_2))
17            val_losses[epoch] = np.sqrt(mse(y_pred_val, y_val))
18
19        # Plot train and val losses for each epoch and each learning rate
20        fig, axs = plt.subplots(figsize=(10, 6))
21        fig.suptitle(f"Learning Rate: {learning_rate}, Penalty: {penalty}")
22        axs.plot(range(1, n_epochs + 1), train_losses, label="Train Loss")
23        axs.plot(range(1, n_epochs + 1), val_losses, label="Val Loss")
24        axs.set_xlabel("Epoch")
25        axs.set_ylabel("RMSE")
26        axs.legend()
27
28        # Calculate RMSE score on test set
29        y_pred_test = sgd.predict(x_test_2)
30        test_rmse_score = np.sqrt(mse(y_pred_test, y_test_2))
31        metric_dict['SGD with learning rate =' + str(learning_rate) + ' and pen' +
32        print(f"RMSE score between y_pred_val and y_test for learning_rate {learning_rate} is {test_rmse_score}")
33        test_rmse_scores.append(test_rmse_score)
34
35 plt.show()
```

```
RMSE score between y_pred_val and y_test for learning_rate=1e-06 and penalty_factor=None: 3.9696651820128945
RMSE score between y_pred_val and y_test for learning_rate=1e-05 and penalty_factor=None: 3.767312827919267
RMSE score between y_pred_val and y_test for learning_rate=0.0001 and penalty_factor=None: 3.699389148877477
RMSE score between y_pred_val and y_test for learning_rate=0.001 and penalty_factor=None: 3.7830674990488733
RMSE score between y_pred_val and y_test for learning_rate=0.01 and penalty_factor=None: 3.836960346031612
RMSE score between y_pred_val and y_test for learning_rate=1e-06 and penalty_factor=l1: 3.9696603920934166
RMSE score between y_pred_val and y_test for learning_rate=1e-05 and penalty_factor=l1: 3.768378646001259
RMSE score between y_pred_val and y_test for learning_rate=0.0001 and penalty_factor=l1: 3.7035761649960635
RMSE score between y_pred_val and y_test for learning_rate=0.001 and penalty_factor=l1: 3.7274527732688942
RMSE score between y_pred_val and y_test for learning_rate=0.01 and penalty_factor=l1: 3.727742795406936
RMSE score between y_pred_val and y_test for learning_rate=1e-06 and penalty_factor=l2: 3.96968115196651
RMSE score between y_pred_val and y_test for learning_rate=1e-05 and penalty_factor=l2: 3.768136952869328
RMSE score between y_pred_val and y_test for learning_rate=0.0001 and penalty_factor=l2: 3.689837070793864
RMSE score between y_pred_val and y_test for learning_rate=0.001 and penalty_factor=l2: 3.797161489890803
RMSE score between y_pred_val and y_test for learning_rate=0.01 and penalty_factor=l2: 3.809220893603164
```

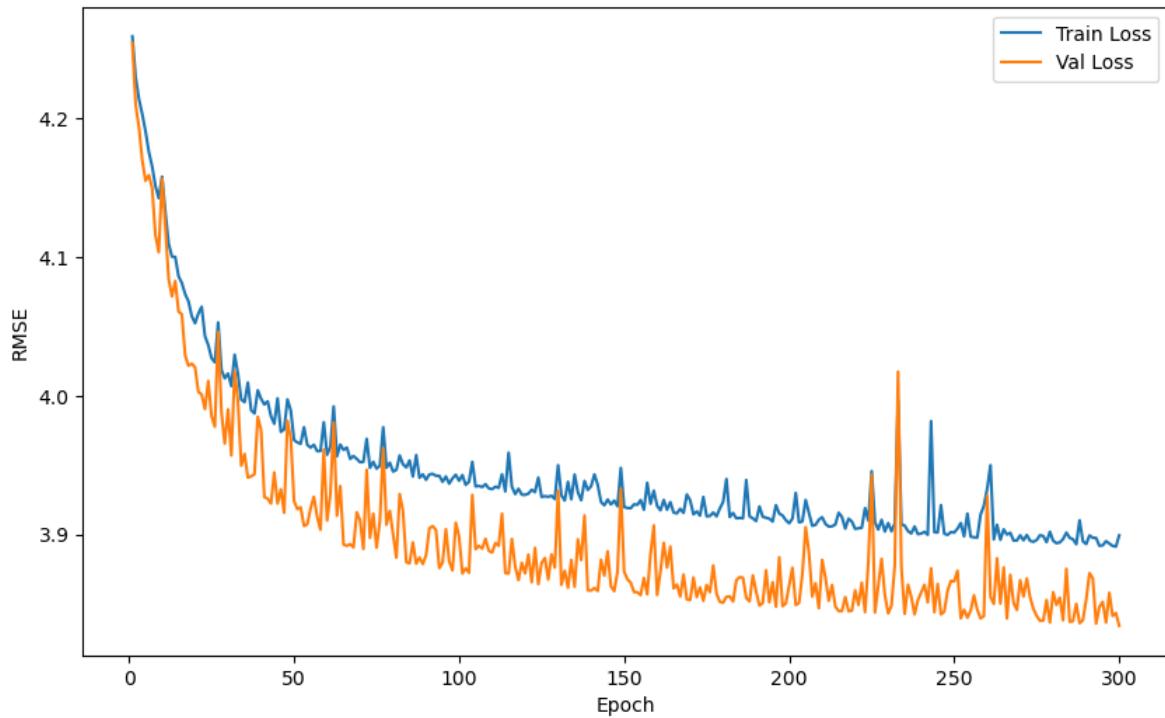
Learning Rate: 1e-06, Penalty: None



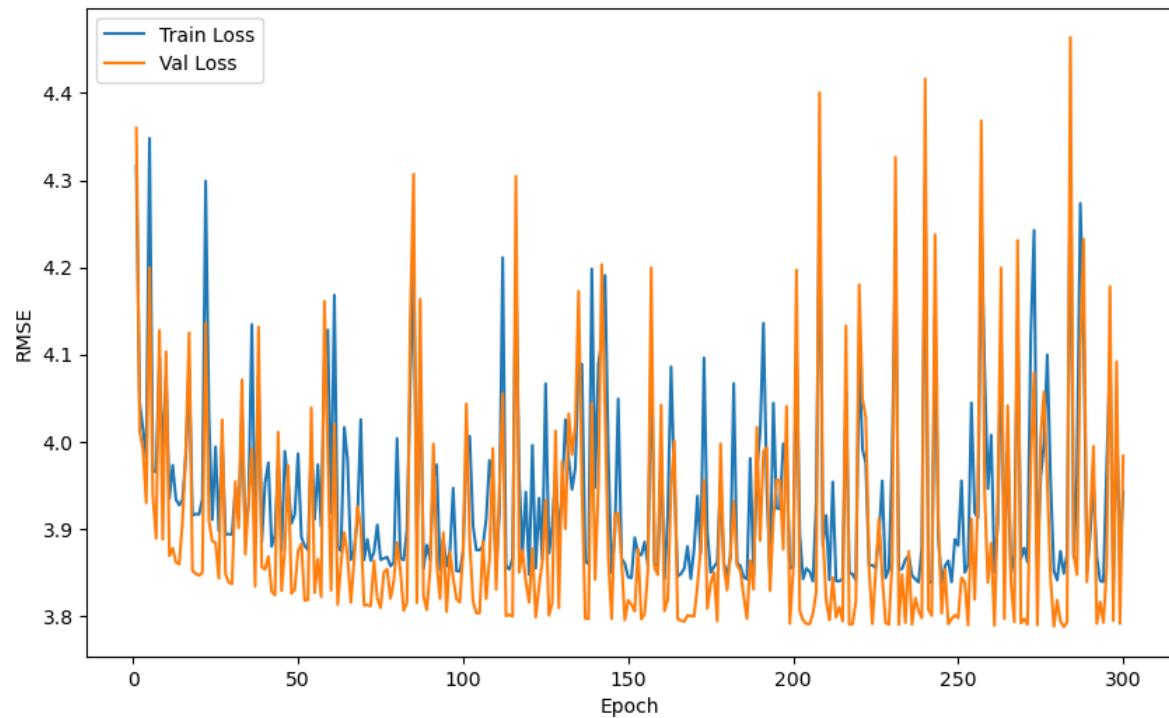
Learning Rate: 1e-05, Penalty: None



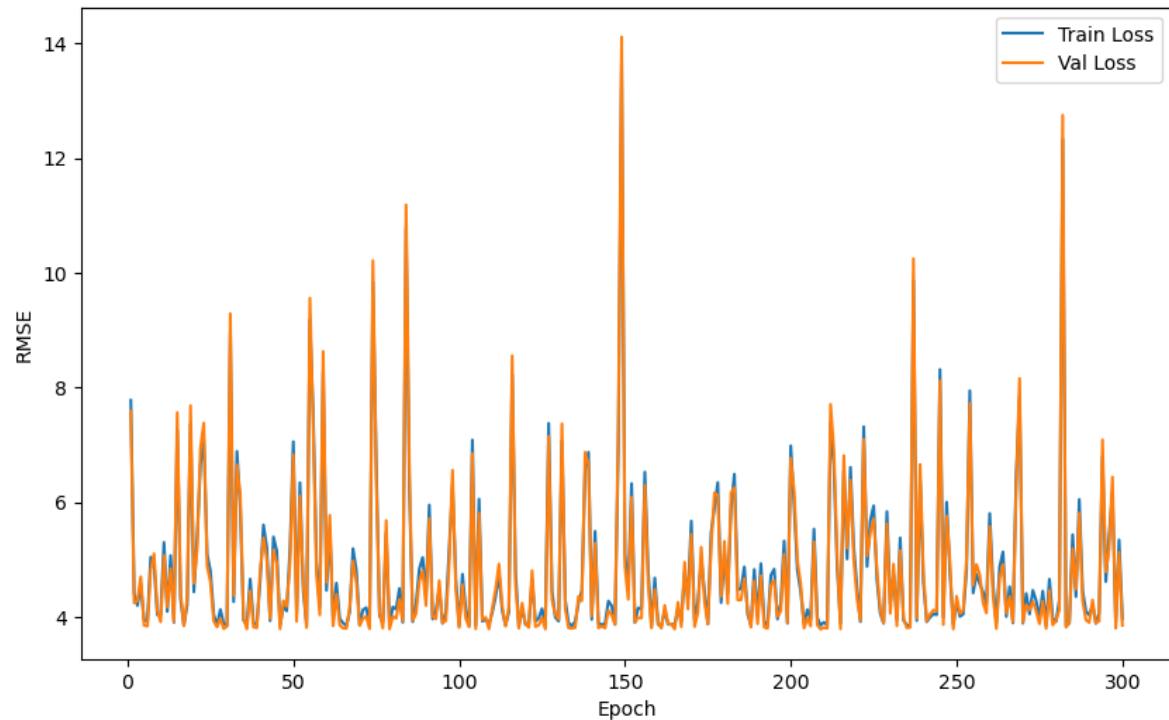
Learning Rate: 0.0001, Penalty: None



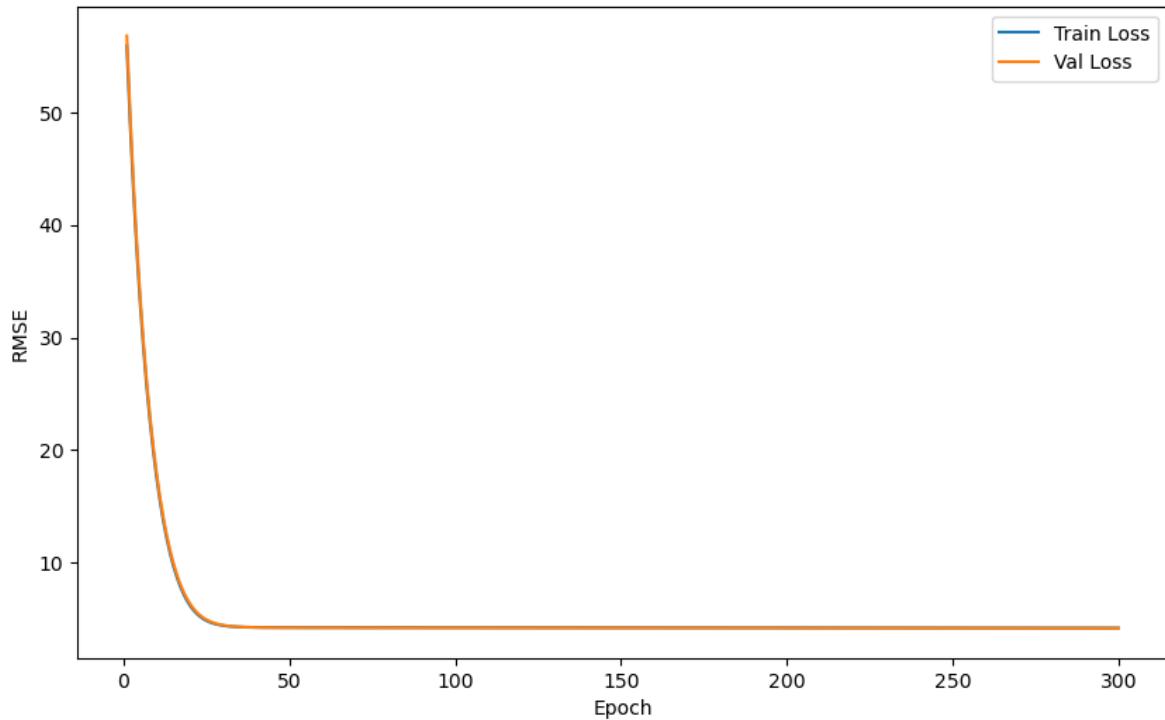
Learning Rate: 0.001, Penalty: None



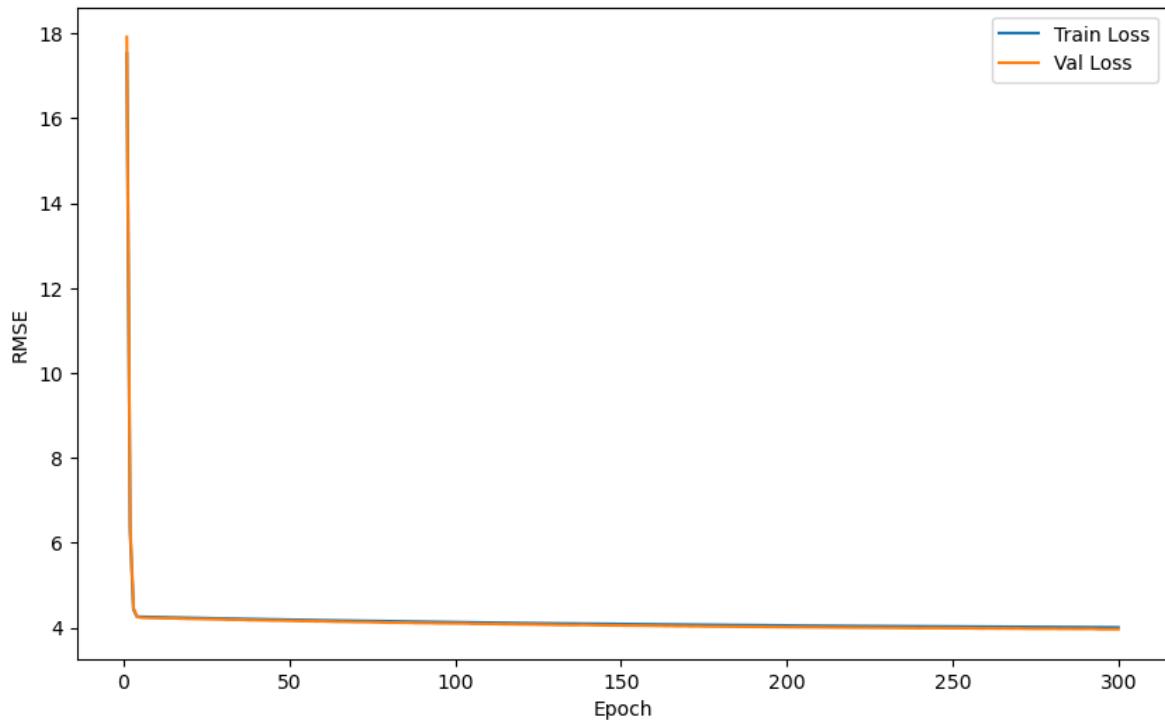
Learning Rate: 0.01, Penalty: None



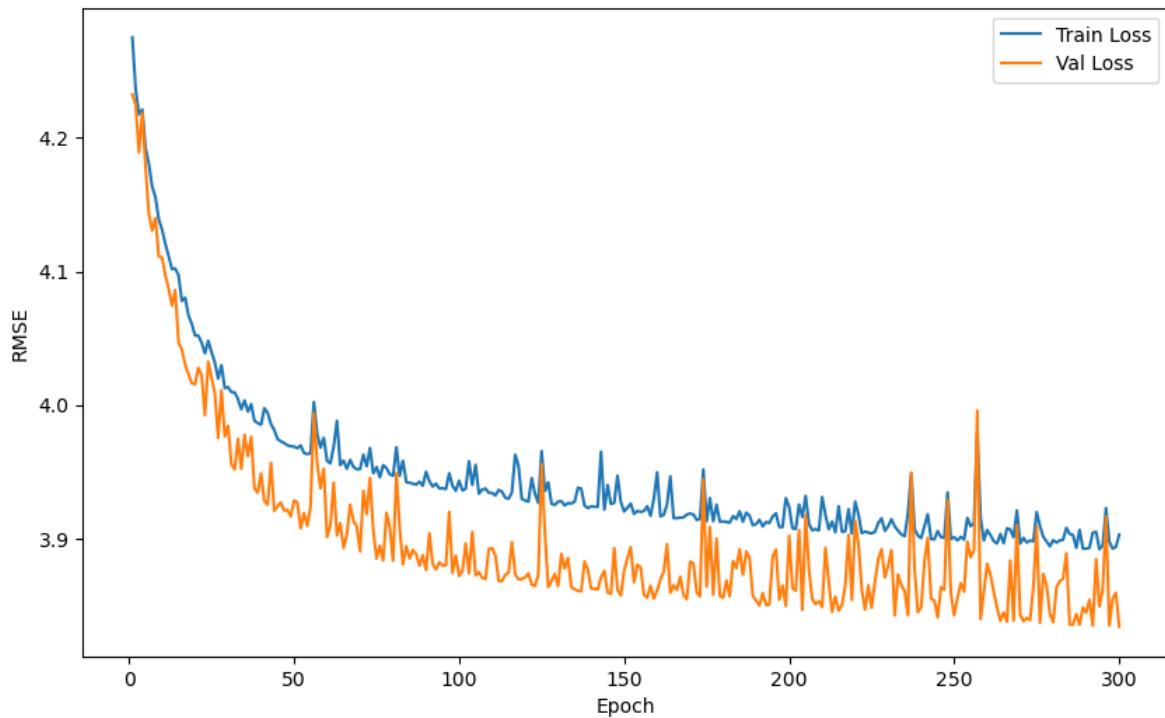
Learning Rate: 1e-06, Penalty: l1



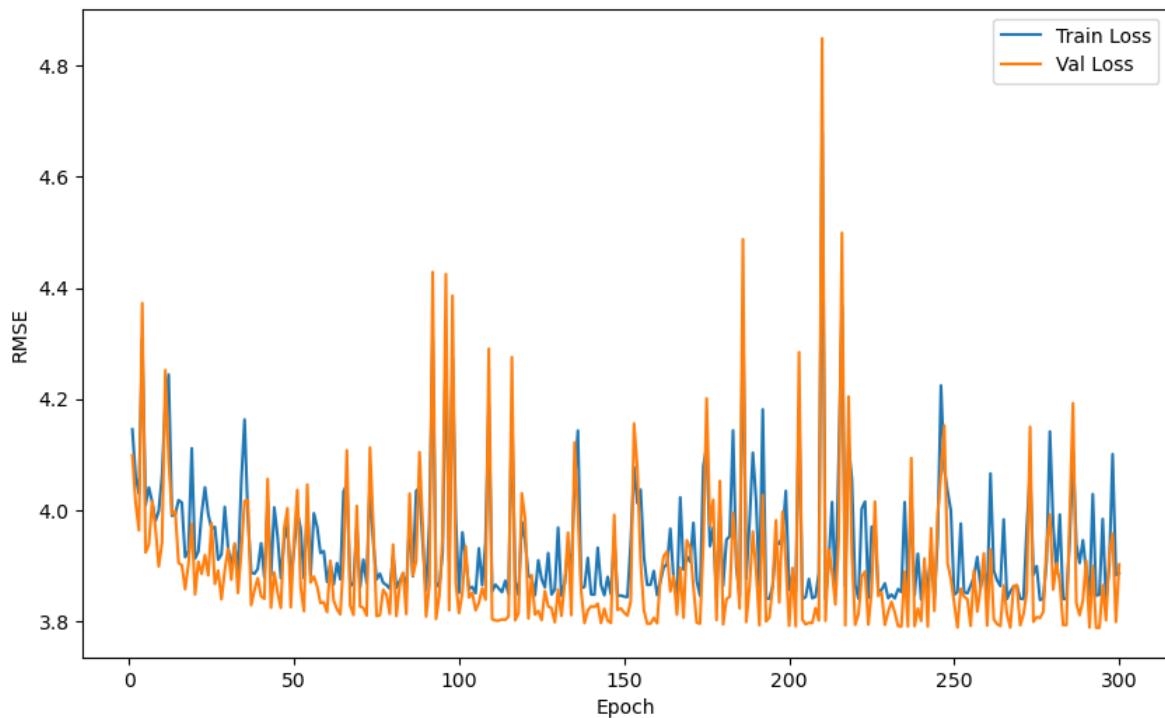
Learning Rate: 1e-05, Penalty: l1



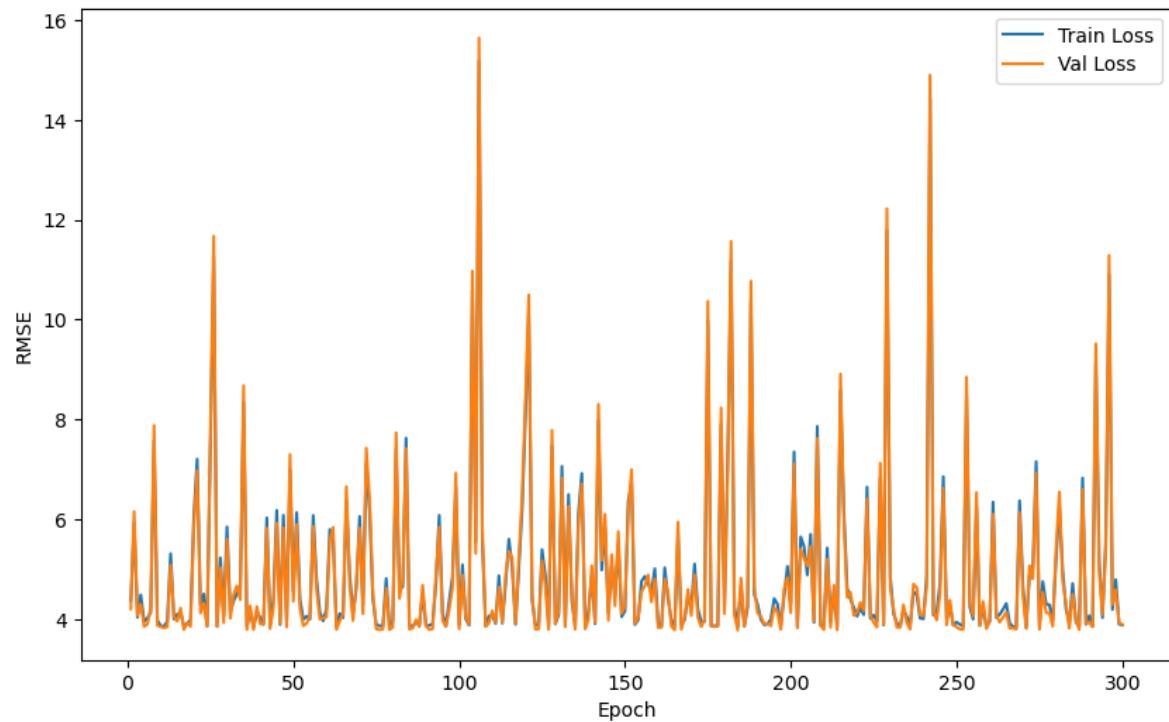
Learning Rate: 0.0001, Penalty: l1



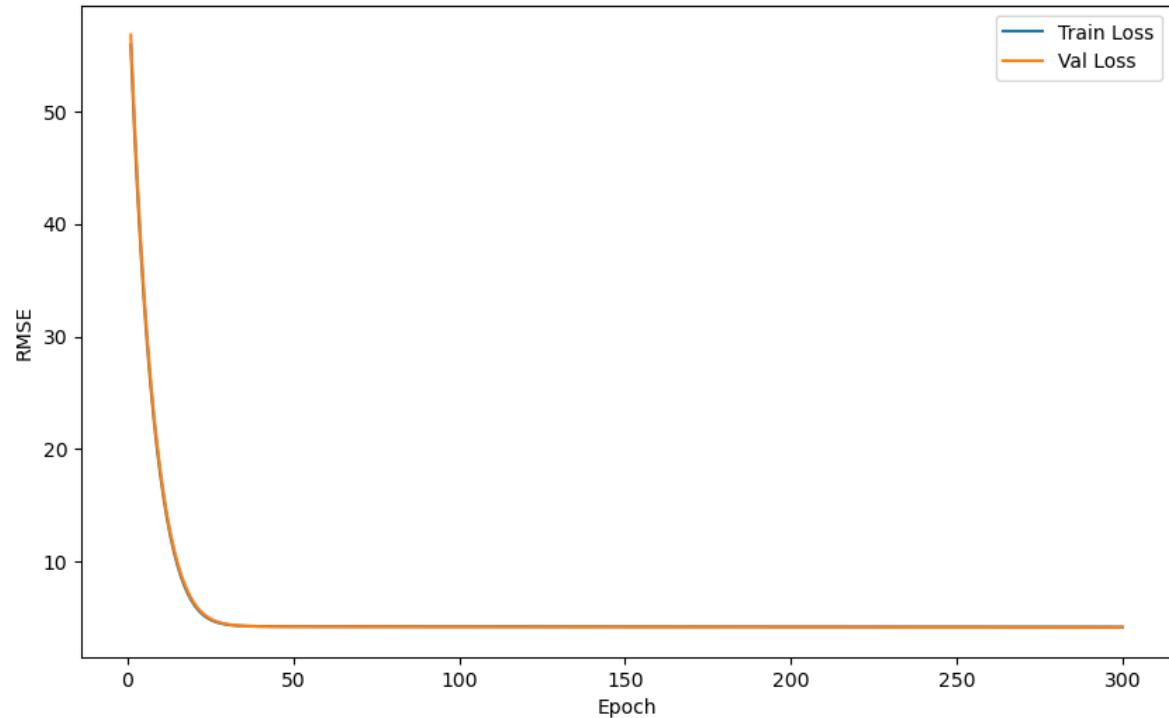
Learning Rate: 0.001, Penalty: l1



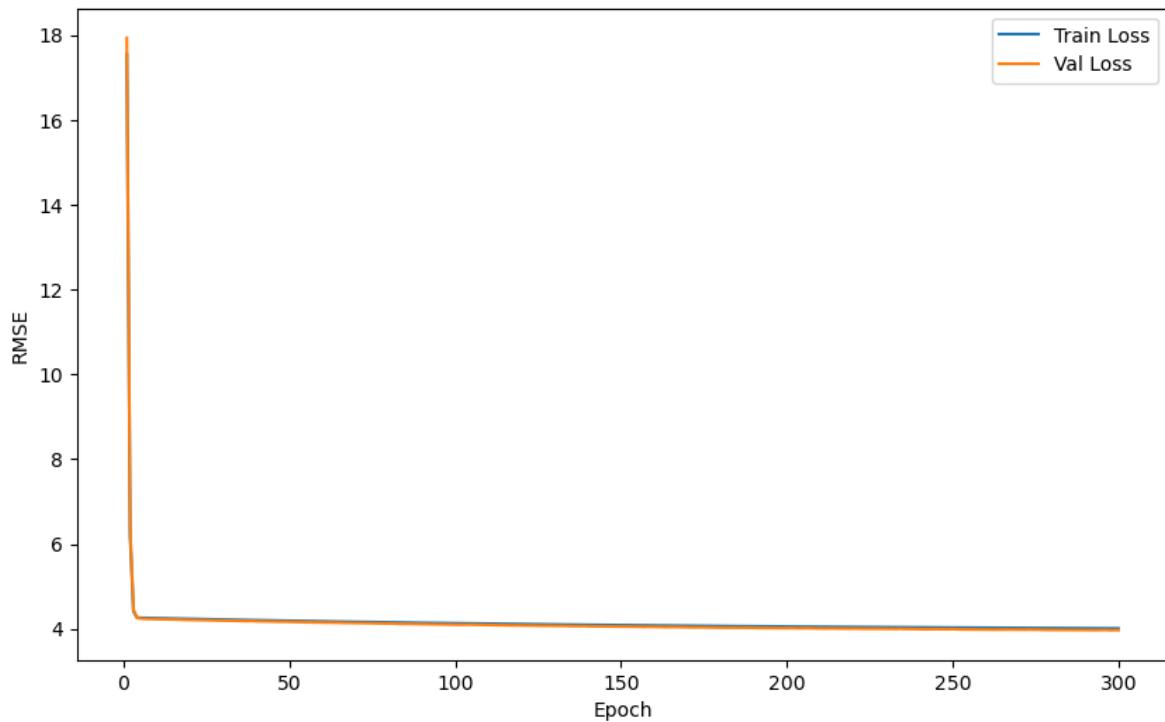
Learning Rate: 0.01, Penalty: l1



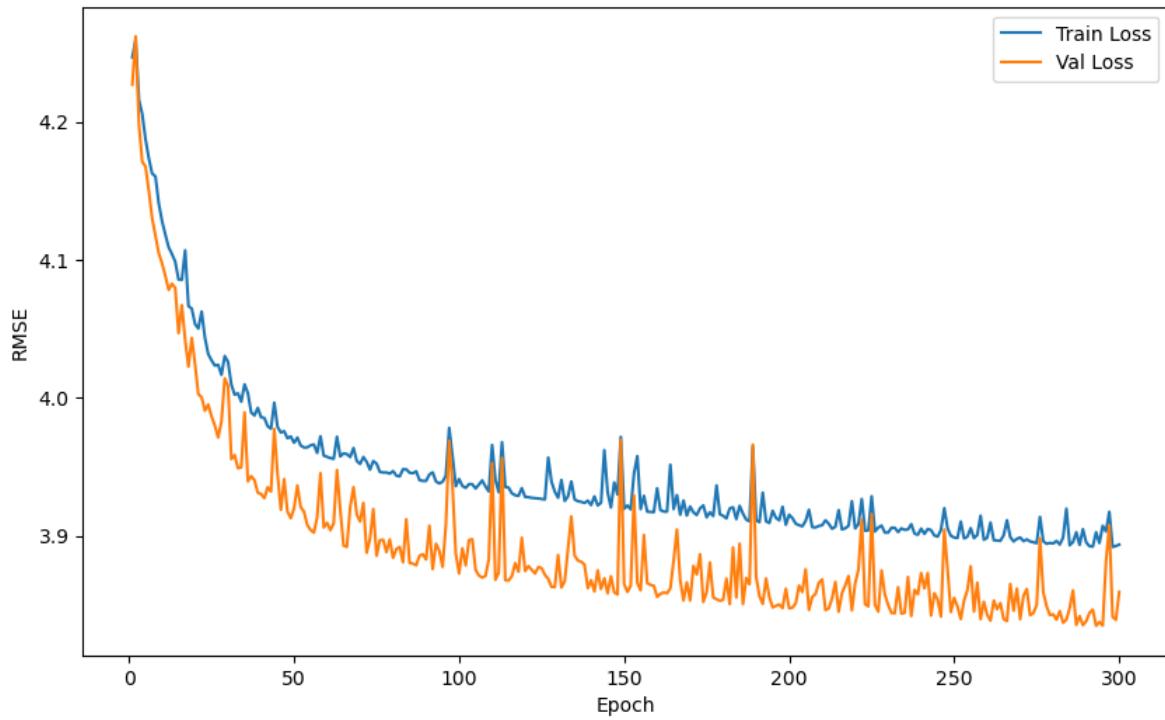
Learning Rate: 1e-06, Penalty: l2



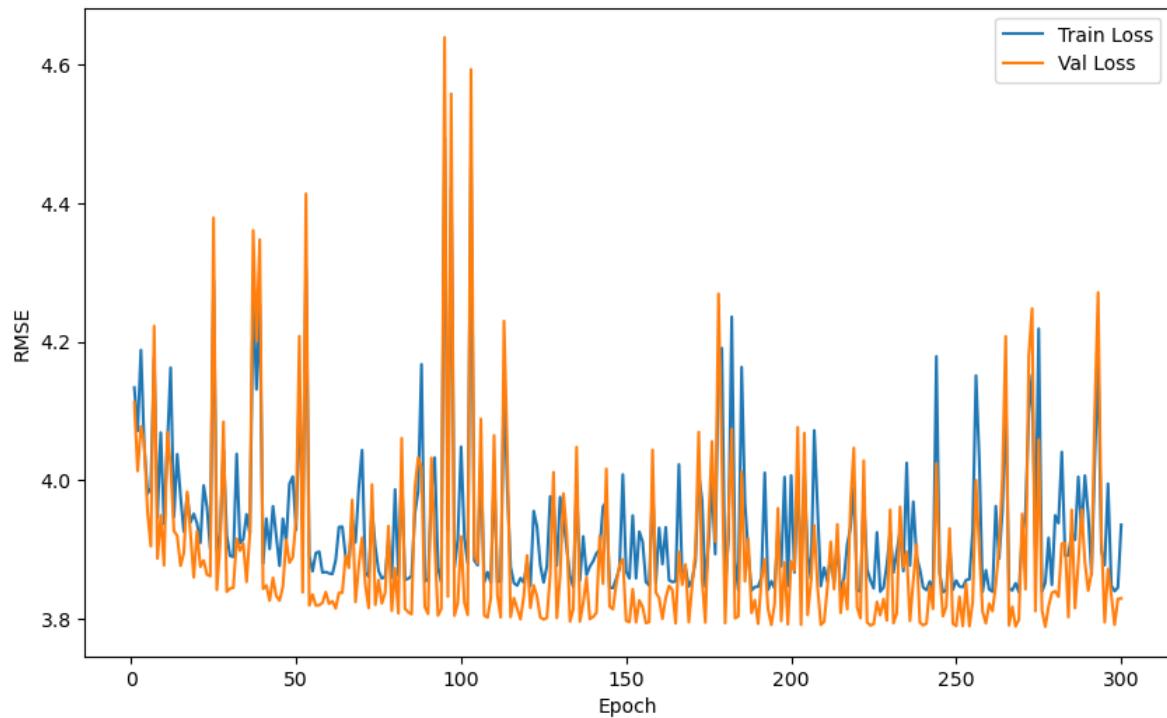
Learning Rate: 1e-05, Penalty: L2



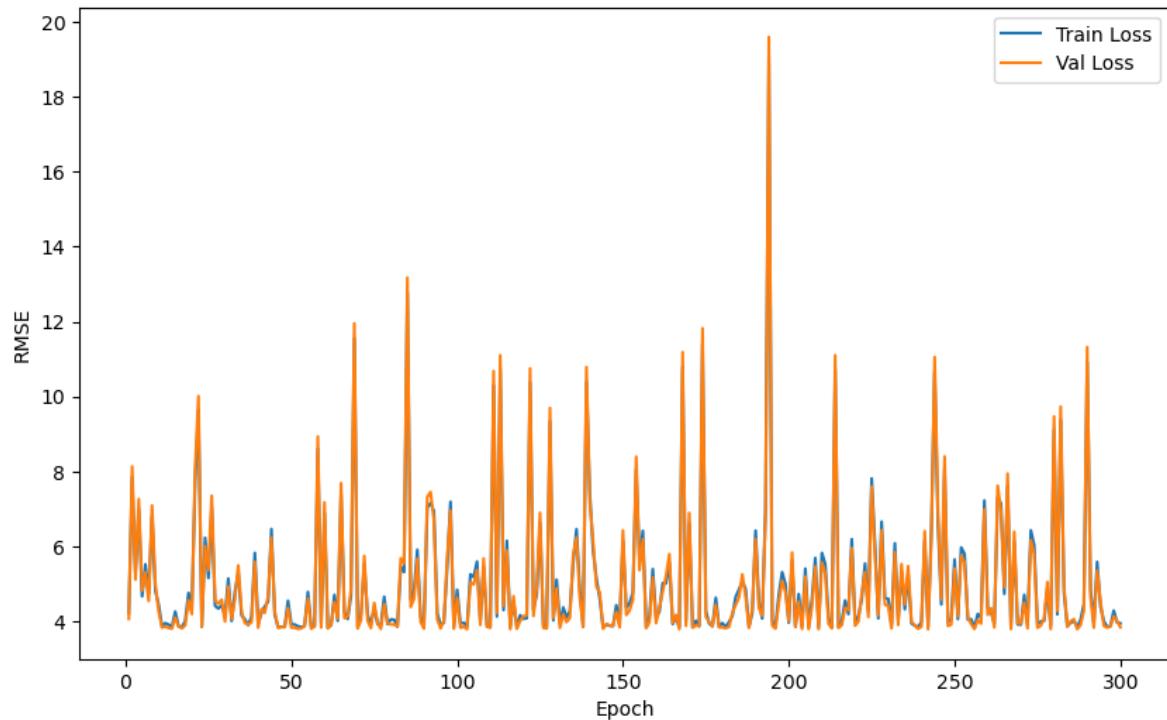
Learning Rate: 0.0001, Penalty: L2



Learning Rate: 0.001, Penalty: l2



Learning Rate: 0.01, Penalty: l2

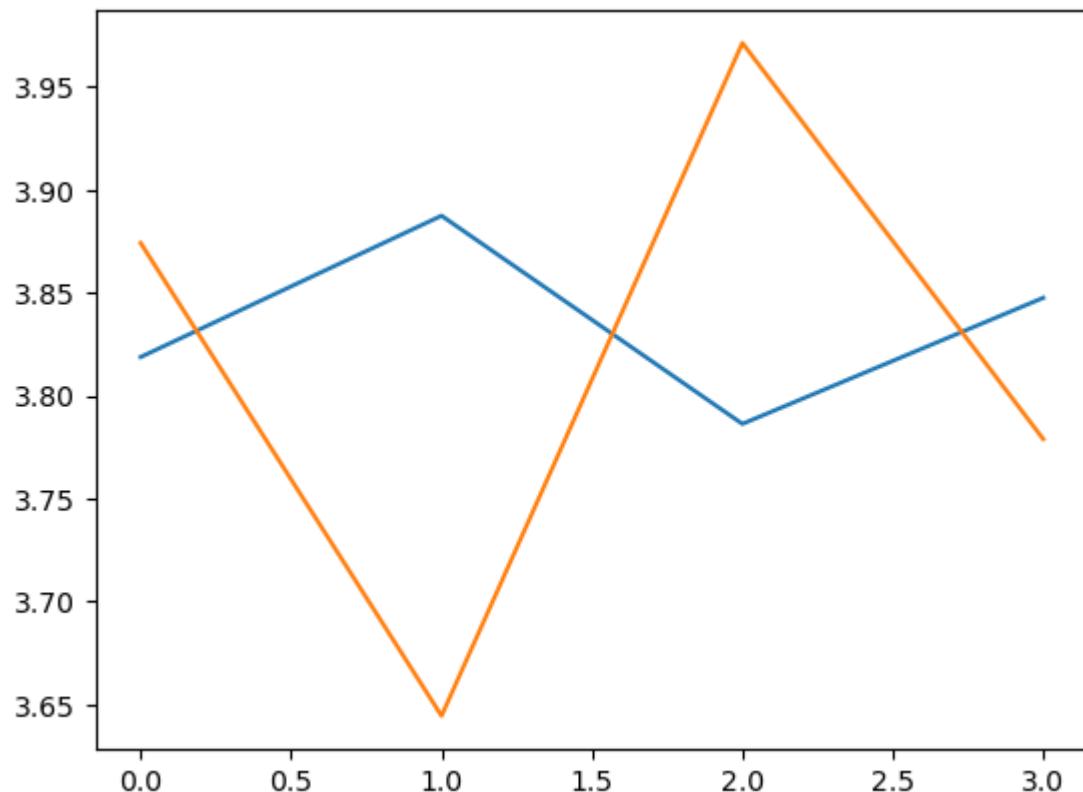


In [23]:

```
1 # Ridge
2
3 train_plot = []
4 val_plot = []
5 ridge = Ridge()
6 for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):
7     print(cnt)
8     x = x_train_2[var[0],:]
9     y = y_train_2[var[0]]
10    x_val = x_train_2[var[1],:]
11    y_val = y_train_2[var[1]]
12    ridge.fit(x, y)
13    y_pred_train = ridge.predict(x)
14    ridge.fit(x_val, y_val)
15    y_pred_val = ridge.predict(x_val)
16    print("Train error:",np.sqrt(mse(y_pred_train,y)))
17    print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
18    train_plot.append(np.sqrt(mse(y_pred_train,y)))
19    val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
20
21 plt.plot([0,1,2,3], train_plot, label = "Training Error")
22 plt.plot([0,1,2,3], val_plot, label = "Val Error")
```

```
0
Train error: 3.8186868297055447
Validation error: 3.874078616824582
1
Train error: 3.88731131963597
Validation error: 3.6444479315091987
2
Train error: 3.7862233026159204
Validation error: 3.9711948361115503
3
Train error: 3.8474595920132773
Validation error: 3.778825605436278
```

Out[23]: [`<matplotlib.lines.Line2D at 0x1aafb022c70>`]



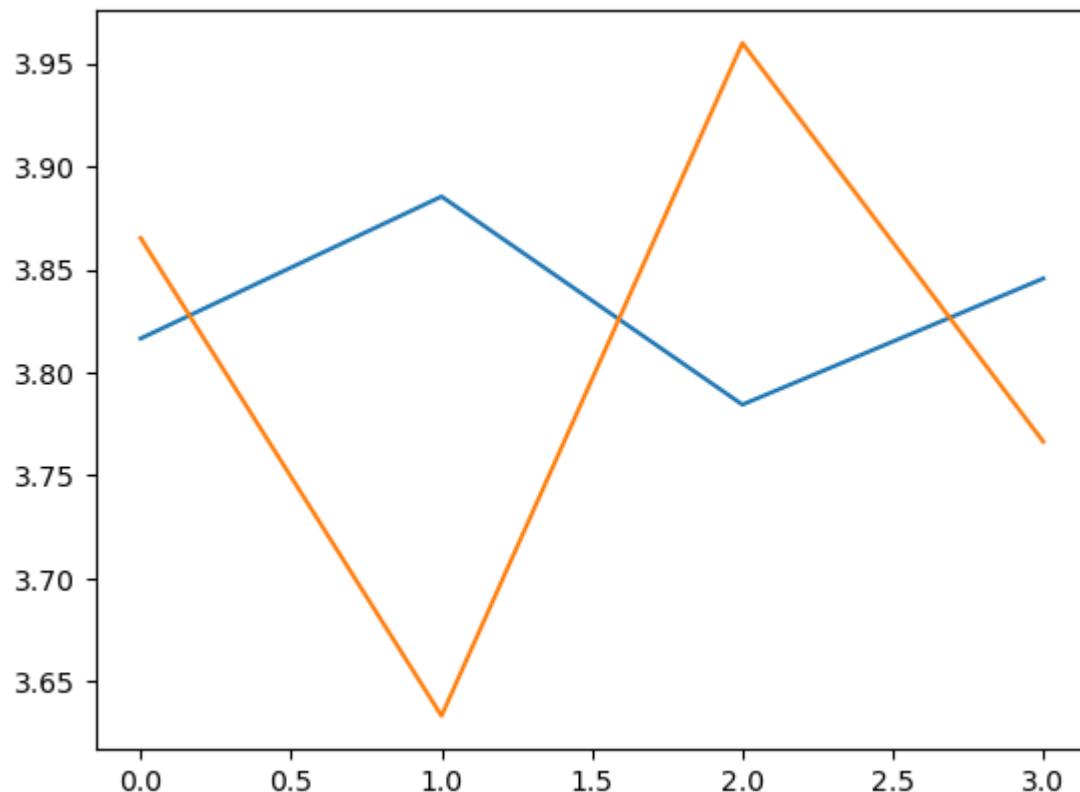
In [24]:

```

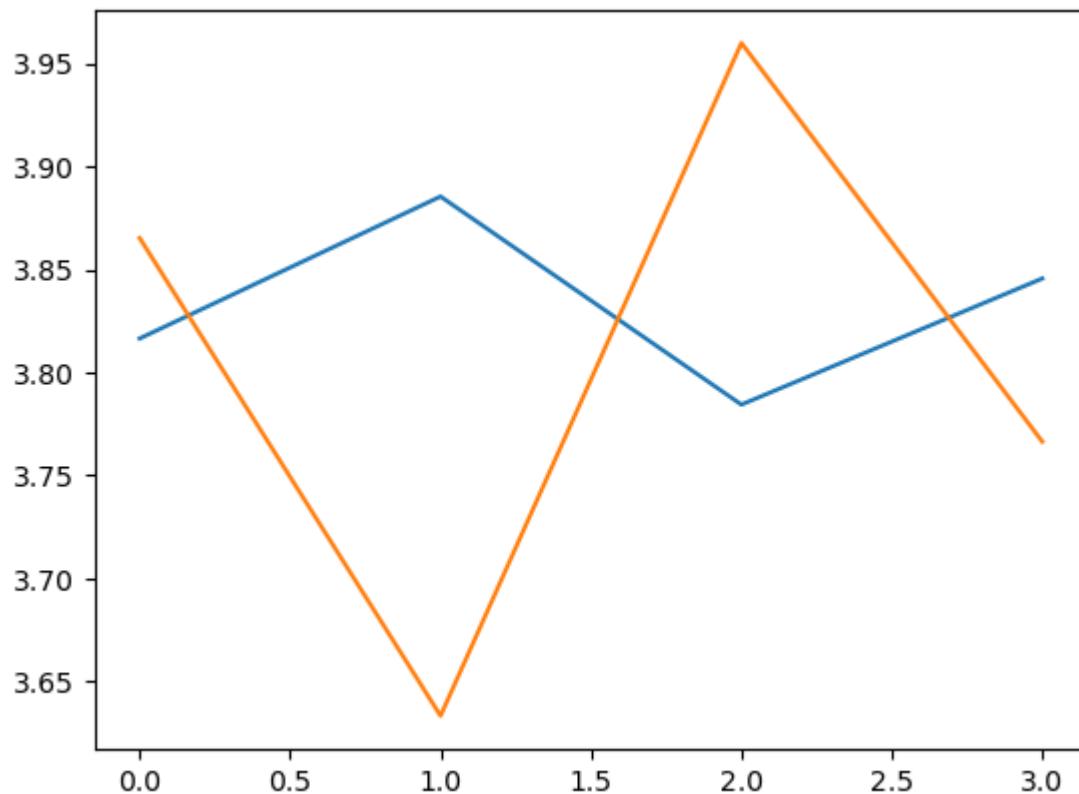
1 # Ridge with different Learning rates
2
3 ridge_alpha = [0,0.001,1,10,100]
4
5 for r in ridge_alpha:
6     print("Alpha value:", r)
7     train_plot = []
8     val_plot = []
9     ridge_obj = Ridge(alpha=r)
10    for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):
11        print(cnt)
12        x = x_train_2[var[0],:]
13        y = y_train_2[var[0]]
14        x_val = x_train_2[var[1],:]
15        y_val = y_train_2[var[1]]
16        ridge_obj.fit(x, y)
17        y_pred_train = ridge_obj.predict(x)
18        ridge_obj.fit(x_val, y_val)
19        y_pred_val = ridge_obj.predict(x_val)
20        print("Train error:",np.sqrt(mse(y_pred_train,y)))
21        print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
22        train_plot.append(np.sqrt(mse(y_pred_train,y)))
23        val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
24        y_pred_test = ridge_obj.predict(x_test_2)
25        metric_dict['Ridge lr=' + str(r)] = np.sqrt(mse(y_pred_test,y_test_2))
26    plt.plot([0,1,2,3], train_plot, label = "Training Error")
27    plt.plot([0,1,2,3], val_plot, label = "Val Error")
28    plt.show()
29    print('\n')

```

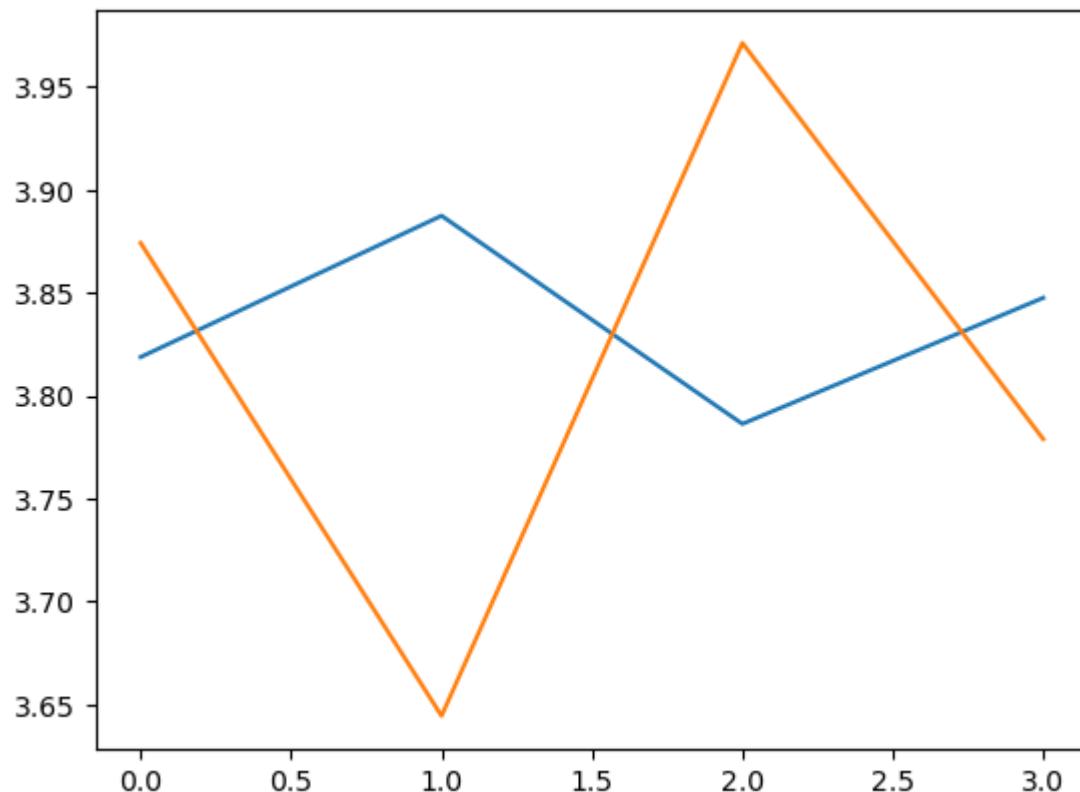
Alpha value: 0
0
Train error: 3.8164612972453944
Validation error: 3.865214725903877
1
Train error: 3.885364521098337
Validation error: 3.6333954981013745
2
Train error: 3.784381403215582
Validation error: 3.9598258620368276
3
Train error: 3.8455747398564712
Validation error: 3.766418213916159



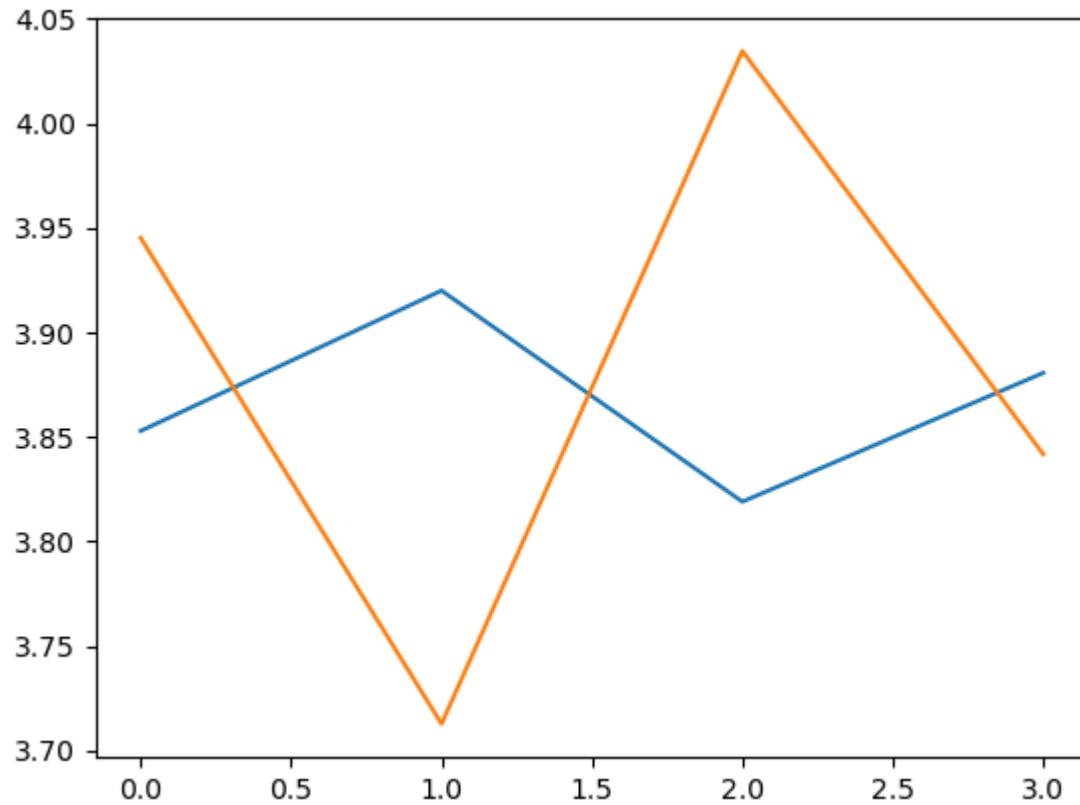
```
Alpha value: 0.001
0
Train error: 3.816461300573554
Validation error: 3.8652147455542765
1
Train error: 3.8853645239437653
Validation error: 3.6333955266443887
2
Train error: 3.7843814058771197
Validation error: 3.9598258914814157
3
Train error: 3.845574742575016
Validation error: 3.7664182493842664
```



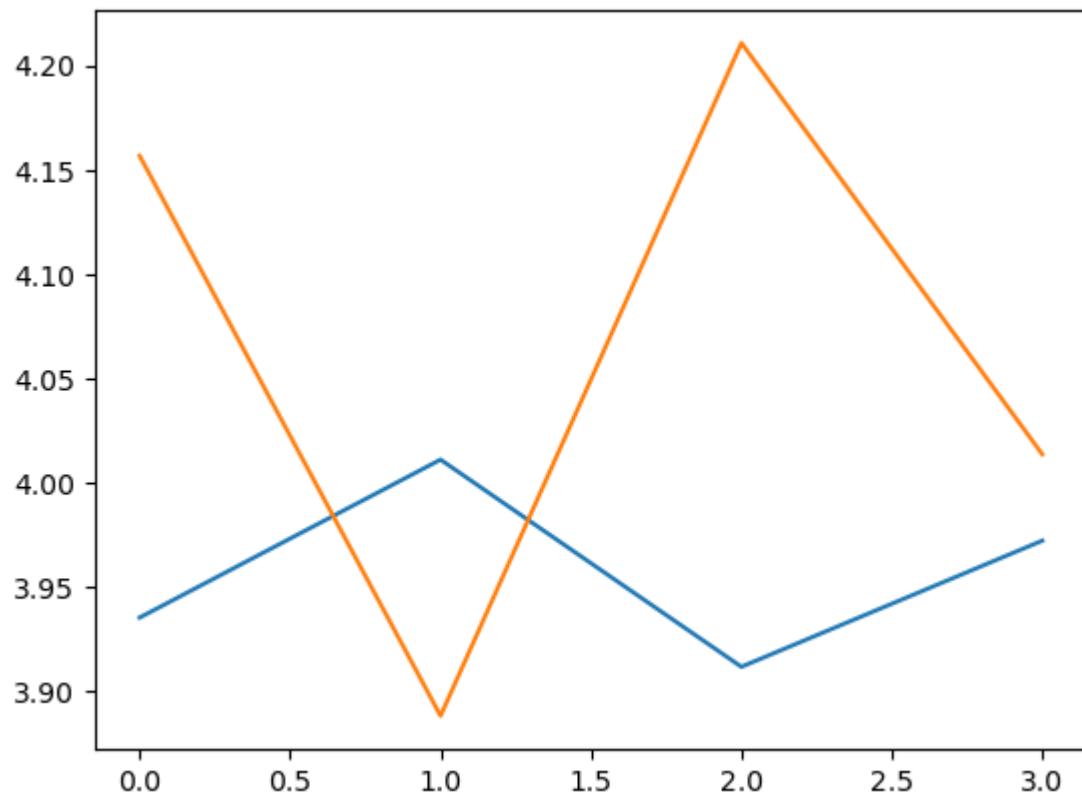
```
Alpha value: 1
0
Train error: 3.8186868297055447
Validation error: 3.874078616824582
1
Train error: 3.88731131963597
Validation error: 3.6444479315091987
2
Train error: 3.7862233026159204
Validation error: 3.9711948361115503
3
Train error: 3.8474595920132773
Validation error: 3.778825605436278
```



```
Alpha value: 10
0
Train error: 3.852839502605182
Validation error: 3.9452247522360193
1
Train error: 3.9199970199332586
Validation error: 3.7127295207848348
2
Train error: 3.8189913036504723
Validation error: 4.03460095127003
3
Train error: 3.880620352043506
Validation error: 3.8416972195632932
```



```
Alpha value: 100
0
Train error: 3.935456740365958
Validation error: 4.156639463680036
1
Train error: 4.011135347831783
Validation error: 3.888408358736641
2
Train error: 3.9118792669831643
Validation error: 4.210704490794341
3
Train error: 3.9723089888724608
Validation error: 4.013699803476162
```

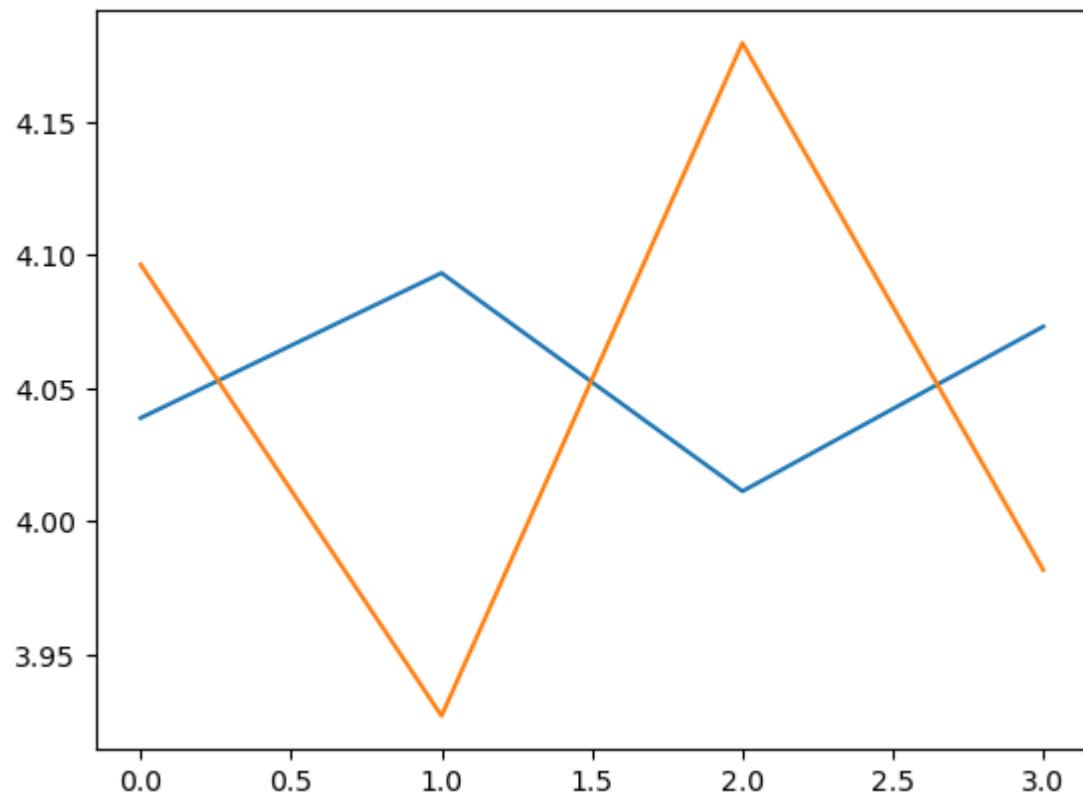


In [25]:

```
1 # Lasso
2
3 train_plot = []
4 val_plot = []
5 lasso = Lasso()
6 for cnt,var in enumerate(kf.split(x_train_2,y_train_2)) :
7     print(cnt)
8     x = x_train_2[var[0],:]
9     y = y_train_2[var[0]]
10    x_val = x_train_2[var[1],:]
11    y_val = y_train_2[var[1]]
12    lasso.fit(x, y)
13    y_pred_train = lasso.predict(x)
14    lasso.fit(x_val, y_val)
15    y_pred_val = lasso.predict(x_val)
16    print("Train error:",np.sqrt(mse(y_pred_train,y)))
17    print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
18    train_plot.append(np.sqrt(mse(y_pred_train,y)))
19    val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
20
21 plt.plot([0,1,2,3], train_plot, label = "Training Error")
22 plt.plot([0,1,2,3], val_plot, label = "Val Error")
```

```
0
Train error: 4.038797374934554
Validation error: 4.096396701079226
1
Train error: 4.093208809782641
Validation error: 3.926992058461979
2
Train error: 4.0112880311792205
Validation error: 4.1795496645221055
3
Train error: 4.073134414611913
Validation error: 3.98176966428072
```

Out[25]: [`<matplotlib.lines.Line2D at 0x1aaf91117c0>`]



In [26]:

```

1 # Lasso with different Learning rates
2
3 lasso_alpha = [0,0.001,1,10,100]
4
5 for r in lasso_alpha:
6     print("Alpha value:", r)
7     train_plot = []
8     val_plot = []
9     lasso_obj = Lasso(alpha=r)
10    for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):
11        print(cnt)
12        x = x_train_2[var[0],:]
13        y = y_train_2[var[0]]
14        x_val = x_train_2[var[1],:]
15        y_val = y_train_2[var[1]]
16        lasso_obj.fit(x, y)
17        y_pred_train = lasso_obj.predict(x)
18        lasso_obj.fit(x_val, y_val)
19        y_pred_val = lasso_obj.predict(x_val)
20        print("Train error:",np.sqrt(mse(y_pred_train,y)))
21        print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
22        train_plot.append(np.sqrt(mse(y_pred_train,y)))
23        val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
24        y_pred_test = lasso_obj.predict(x_test_2)
25        metric_dict['Lasso lr=' + str(r)] = np.sqrt(mse(y_pred_test,y_test_2))
26    plt.plot([0,1,2,3], train_plot, label = "Training Error")
27    plt.plot([0,1,2,3], val_plot, label = "Val Error")
28    plt.show()
29    print('\n')

```

Alpha value: 0
0
Train error: 3.8164612972453944
Validation error: 3.865214725903877
1
Train error: 3.885364521098337
Validation error: 3.6333954981013736
2
Train error: 3.7843814032155816
Validation error: 3.959825862036827
3
Train error: 3.8455747398564717
Validation error: 3.7664182139161584

```
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\3184651632.py:16: UserWarning: With alpha=0, this algorithm does not converge well. You are advised to use the LinearRegression estimator
    lasso_obj.fit(x, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: UserWarning: Coordinate descent with no regularization may lead to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 7.457e+03, tolerance: 5.971e+00
Linear regression models with null weight for the l1 regularization term are more efficiently fitted using one of the solvers implemented in sklearn.linear_model.Ridge/RidgeCV instead.
    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\3184651632.py:18: UserWarning: With alpha=0, this algorithm does not converge well. You are advised to use the LinearRegression estimator
    lasso_obj.fit(x_val, y_val)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: UserWarning: Coordinate descent with no regularization may lead to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.555e+03, tolerance: 2.052e+00
Linear regression models with null weight for the l1 regularization term are more efficiently fitted using one of the solvers implemented in sklearn.linear_model.Ridge/RidgeCV instead.
    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\3184651632.py:16: UserWarning: With alpha=0, this algorithm does not converge well. You are advised to use the LinearRegression estimator
    lasso_obj.fit(x, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: UserWarning: Coordinate descent with no regularization may lead to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 7.729e+03, tolerance: 6.179e+00
Linear regression models with null weight for the l1 regularization term are more efficiently fitted using one of the solvers implemented in sklearn.linear_model.Ridge/RidgeCV instead.
    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\3184651632.py:18: UserWarning: With alpha=0, this algorithm does not converge well. You are advised to use the LinearRegression estimator
    lasso_obj.fit(x_val, y_val)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: UserWarning: Coordinate descent with no regularization may lead to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: UserWarning: Coordinate descent with no regularization may lead to unexpected results and is discouraged.
```

```
escent.py:647: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 2.257e+03, tolerance: 1.870e+00
Linear regression models with null weight for the l1 regularization term are
more efficiently fitted using one of the solvers implemented in sklearn.linea
r_model.Ridge/RidgeCV instead.

    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\3184651632.py:16: UserWarning:
  With alpha=0, this algorithm does not converge well. You are advised to u
se the LinearRegression estimator
    lasso_obj.fit(x, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: UserWarning: Coordinate descent with no regularization may lea
d to unexpected results and is discouraged.

    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 7.340e+03, tolerance: 5.880e+00
Linear regression models with null weight for the l1 regularization term are
more efficiently fitted using one of the solvers implemented in sklearn.linea
r_model.Ridge/RidgeCV instead.

    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\3184651632.py:18: UserWarning:
  With alpha=0, this algorithm does not converge well. You are advised to u
se the LinearRegression estimator
    lasso_obj.fit(x_val, y_val)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: UserWarning: Coordinate descent with no regularization may lea
d to unexpected results and is discouraged.

    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 2.673e+03, tolerance: 2.163e+00
Linear regression models with null weight for the l1 regularization term are
more efficiently fitted using one of the solvers implemented in sklearn.linea
r_model.Ridge/RidgeCV instead.

    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\3184651632.py:16: UserWarning:
  With alpha=0, this algorithm does not converge well. You are advised to u
se the LinearRegression estimator
    lasso_obj.fit(x, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: UserWarning: Coordinate descent with no regularization may lea
d to unexpected results and is discouraged.

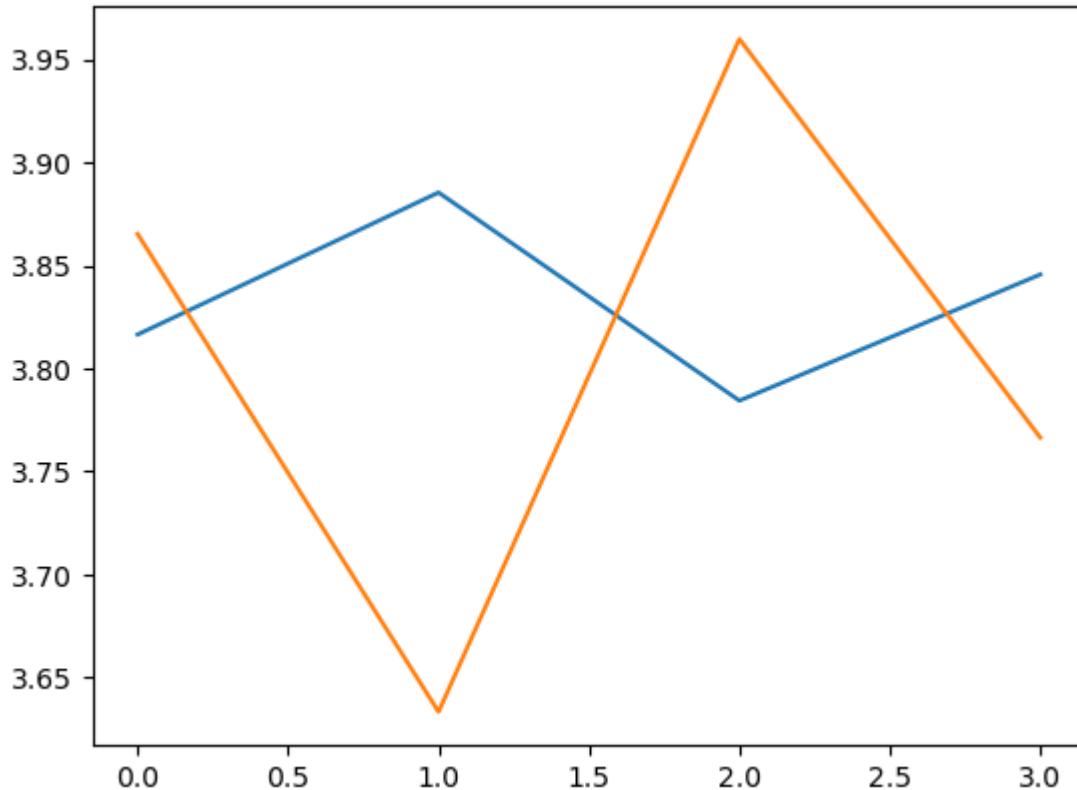
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 7.579e+03, tolerance: 6.105e+00
Linear regression models with null weight for the l1 regularization term are
more efficiently fitted using one of the solvers implemented in sklearn.linea
r_model.Ridge/RidgeCV instead.

    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\3184651632.py:18: UserWarning:
  With alpha=0, this algorithm does not converge well. You are advised to u
```

```

se the LinearRegression estimator
lasso_obj.fit(x_val, y_val)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: UserWarning: Coordinate descent with no regularization may lead to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.419e+03, tolerance: 1.902e+00
Linear regression models with null weight for the l1 regularization term are more efficiently fitted using one of the solvers implemented in sklearn.linear_model.Ridge/RidgeCV instead.
    model = cd_fast.enet_coordinate_descent()

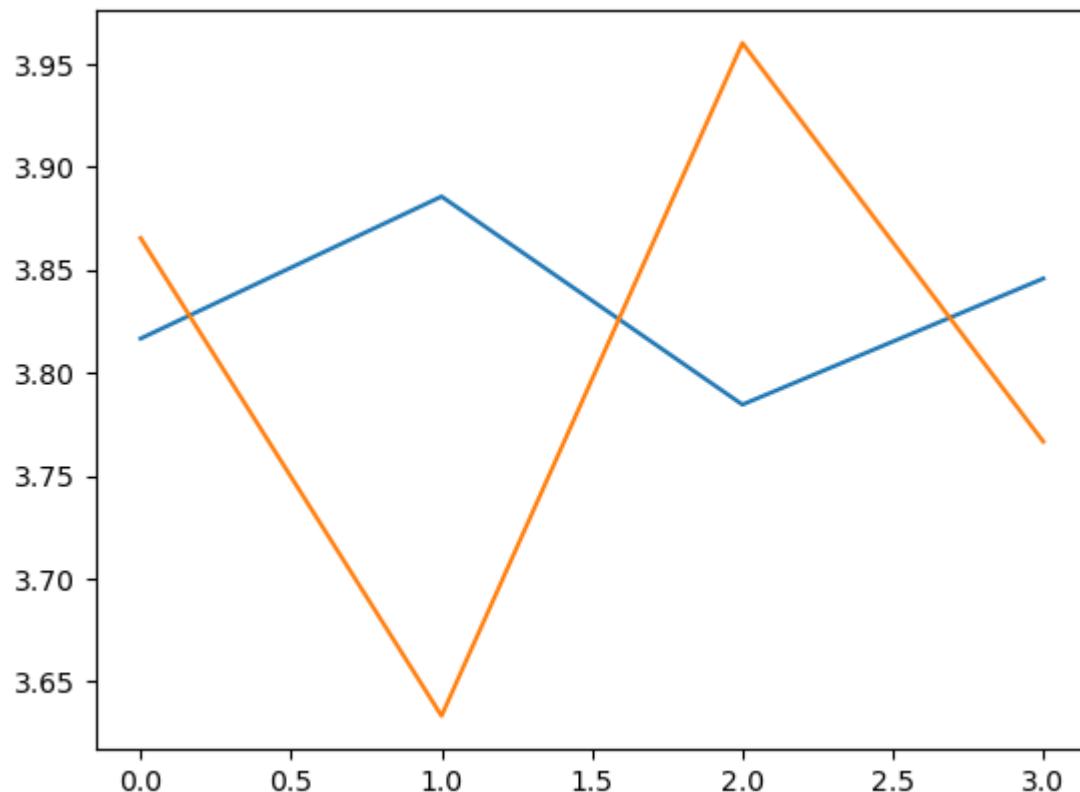
```



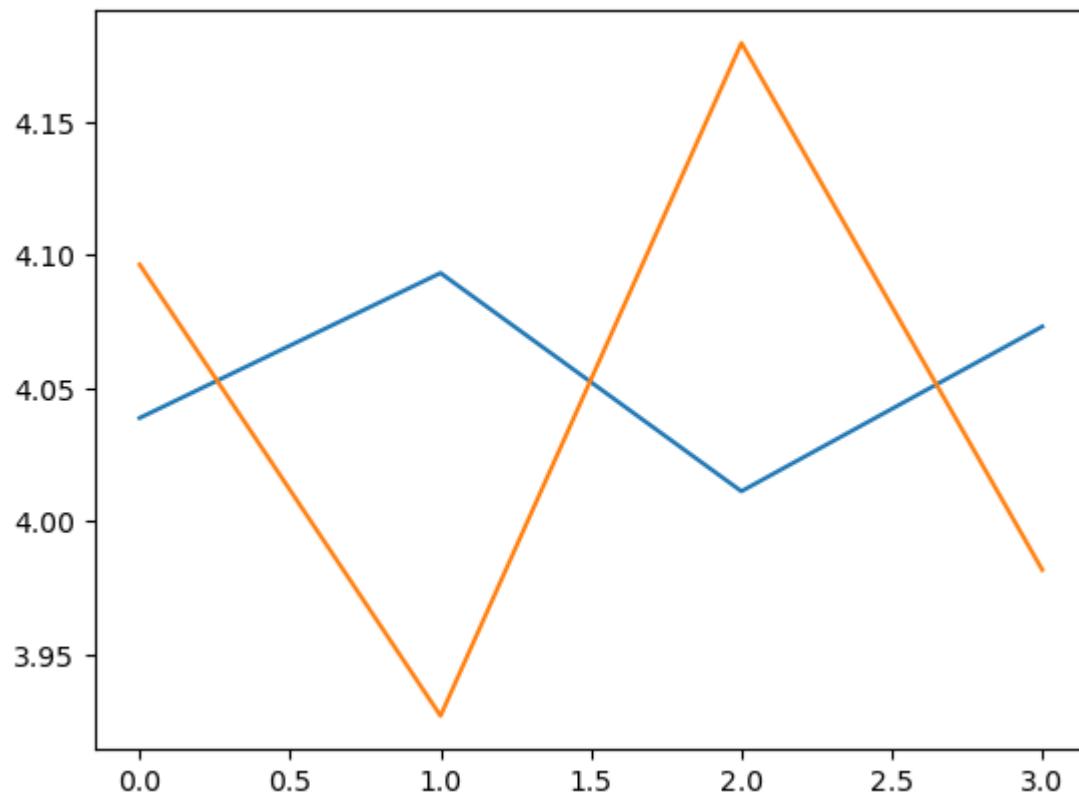
```

Alpha value: 0.001
0
Train error: 3.8165428504187138
Validation error: 3.8652885815082554
1
Train error: 3.8854637607861346
Validation error: 3.6334863391034093
2
Train error: 3.784485964434128
Validation error: 3.959932777446898
3
Train error: 3.845656624340425
Validation error: 3.7665211802799496

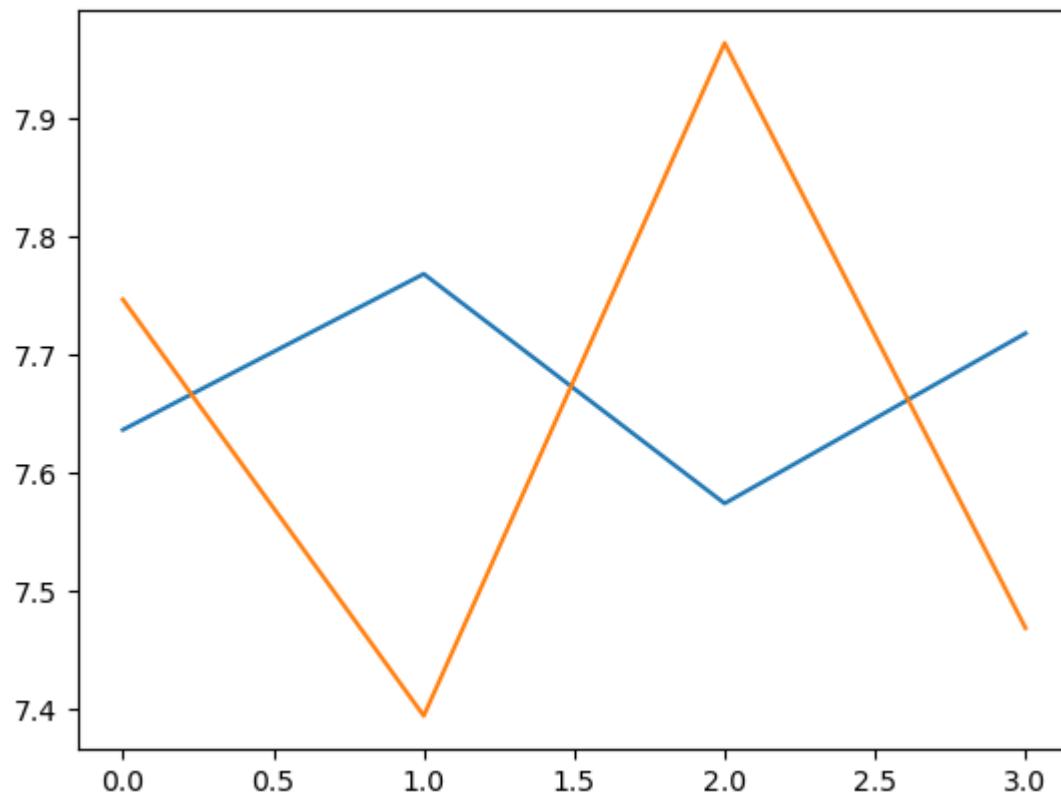
```



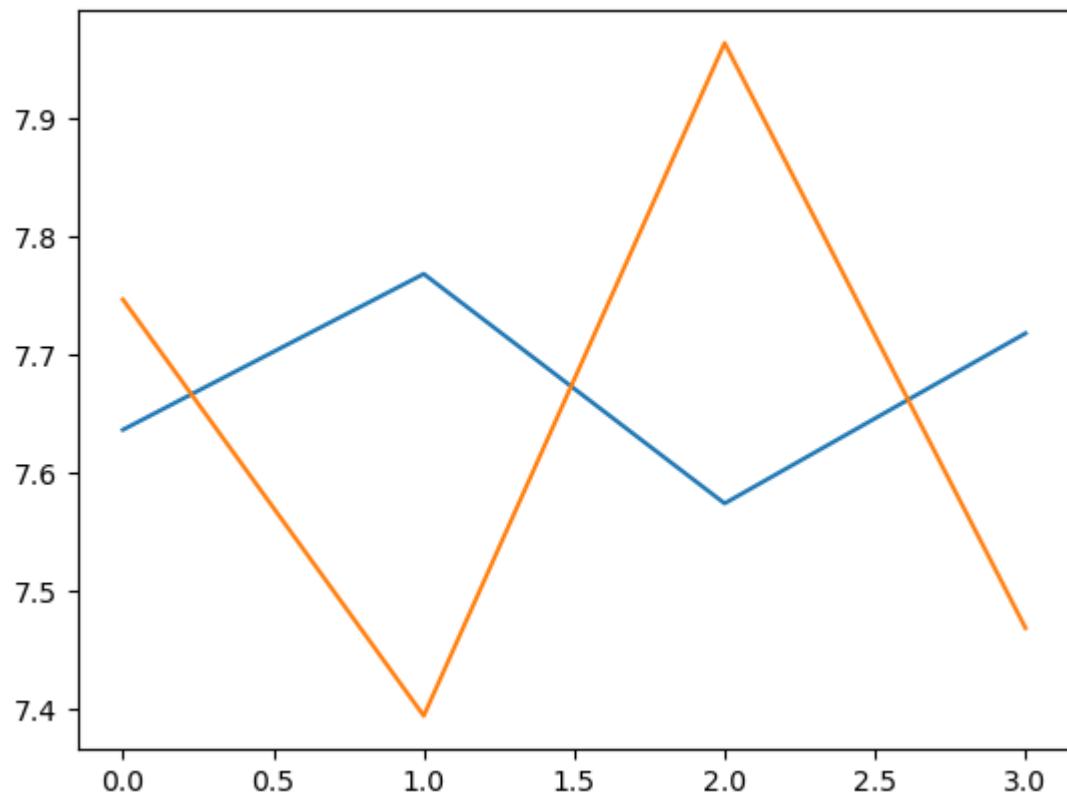
```
Alpha value: 1
0
Train error: 4.038797374934554
Validation error: 4.096396701079226
1
Train error: 4.093208809782641
Validation error: 3.926992058461979
2
Train error: 4.0112880311792205
Validation error: 4.1795496645221055
3
Train error: 4.073134414611913
Validation error: 3.98176966428072
```



```
Alpha value: 10
0
Train error: 7.63616284494717
Validation error: 7.746478390300121
1
Train error: 7.768110993028431
Validation error: 7.393978527563255
2
Train error: 7.573816670038543
Validation error: 7.963619835739025
3
Train error: 7.717837191561589
Validation error: 7.468266103185494
```



```
Alpha value: 100
0
Train error: 7.63616284494717
Validation error: 7.746478390300121
1
Train error: 7.768110993028431
Validation error: 7.393978527563255
2
Train error: 7.573816670038543
Validation error: 7.963619835739025
3
Train error: 7.717837191561589
Validation error: 7.468266103185494
```

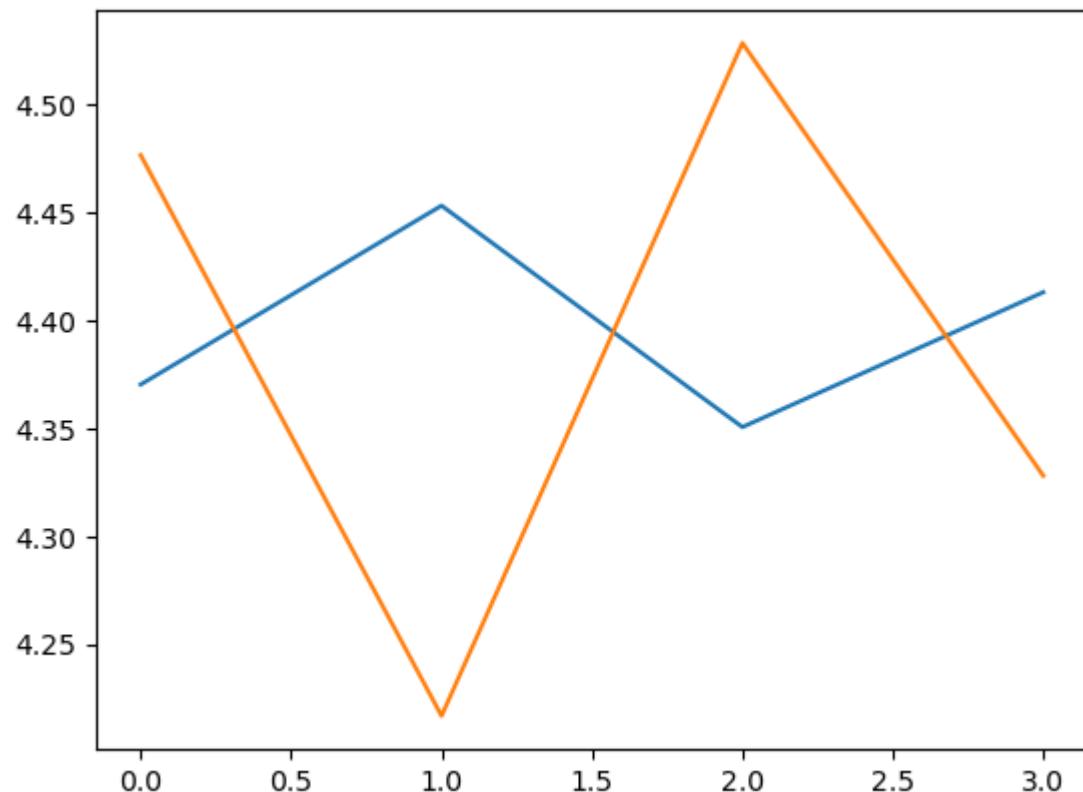


In [27]:

```
1 # Elastic Net
2
3 train_plot = []
4 val_plot = []
5 el_net = ElasticNet()
6 for cnt,var in enumerate(kf.split(x_train_2,y_train_2)) :
7     print(cnt)
8     x = x_train_2[var[0],:]
9     y = y_train_2[var[0]]
10    x_val = x_train_2[var[1],:]
11    y_val = y_train_2[var[1]]
12    el_net.fit(x, y)
13    y_pred_train = el_net.predict(x)
14    el_net.fit(x_val, y_val)
15    y_pred_val = el_net.predict(x_val)
16    print("Train error:",np.sqrt(mse(y_pred_train,y)))
17    print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
18    train_plot.append(np.sqrt(mse(y_pred_train,y)))
19    val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
20
21 plt.plot([0,1,2,3], train_plot, label = "Training Error")
22 plt.plot([0,1,2,3], val_plot, label = "Val Error")
```

```
0
Train error: 4.3702365712793885
Validation error: 4.4765321353181315
1
Train error: 4.453116189790851
Validation error: 4.216748083891603
2
Train error: 4.350523676313172
Validation error: 4.528462199812707
3
Train error: 4.413012157576303
Validation error: 4.327948718273263
```

Out[27]: [`<matplotlib.lines.Line2D at 0x1aaafca34550>`]



In [28]:

```

1 # ElasticNet with different learning rates
2
3 en_alpha = [0,0.001,1,10,100]
4
5 for r in en_alpha:
6     print("Alpha value:", r)
7     train_plot = []
8     val_plot = []
9     en_obj = ElasticNet(alpha=r)
10    for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):
11        print(cnt)
12        x = x_train_2[var[0],:]
13        y = y_train_2[var[0]]
14        x_val = x_train_2[var[1],:]
15        y_val = y_train_2[var[1]]
16        en_obj.fit(x, y)
17        y_pred_train = en_obj.predict(x)
18        en_obj.fit(x_val, y_val)
19        y_pred_val = en_obj.predict(x_val)
20        print("Train error:",np.sqrt(mse(y_pred_train,y)))
21        print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
22        train_plot.append(np.sqrt(mse(y_pred_train,y)))
23        val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
24        y_pred_test = en_obj.predict(x_test_2)
25        metric_dict['ElasticNet lr=' + str(r)] = np.sqrt(mse(y_pred_test,y))
26    plt.plot([0,1,2,3], train_plot, label = "Training Error")
27    plt.plot([0,1,2,3], val_plot, label = "Val Error")
28    plt.show()
29    print('\n')

```

Alpha value: 0
0
Train error: 3.8164612972453944
Validation error: 3.865214725903877
1
Train error: 3.885364521098337
Validation error: 3.6333954981013736
2
Train error: 3.7843814032155816
Validation error: 3.959825862036827
3
Train error: 3.8455747398564717
Validation error: 3.7664182139161584

```
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\9477209.py:16: UserWarning:  
With alpha=0, this algorithm does not converge well. You are advised to use t  
he LinearRegression estimator  
    en_obj.fit(x, y)  
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d  
escent.py:647: UserWarning: Coordinate descent with no regularization may lea  
d to unexpected results and is discouraged.  
    model = cd_fast.enet_coordinate_descent()  
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d  
escent.py:647: ConvergenceWarning: Objective did not converge. You might want  
to increase the number of iterations, check the scale of the features or cons  
ider increasing regularisation. Duality gap: 7.457e+03, tolerance: 5.971e+00  
Linear regression models with null weight for the l1 regularization term are  
more efficiently fitted using one of the solvers implemented in sklearn.linea  
r_model.Ridge/RidgeCV instead.  
    model = cd_fast.enet_coordinate_descent()  
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\9477209.py:18: UserWarning:  
With alpha=0, this algorithm does not converge well. You are advised to use t  
he LinearRegression estimator  
    en_obj.fit(x_val, y_val)  
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d  
escent.py:647: UserWarning: Coordinate descent with no regularization may lea  
d to unexpected results and is discouraged.  
    model = cd_fast.enet_coordinate_descent()  
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d  
escent.py:647: ConvergenceWarning: Objective did not converge. You might want  
to increase the number of iterations, check the scale of the features or cons  
ider increasing regularisation. Duality gap: 2.555e+03, tolerance: 2.052e+00  
Linear regression models with null weight for the l1 regularization term are  
more efficiently fitted using one of the solvers implemented in sklearn.linea  
r_model.Ridge/RidgeCV instead.  
    model = cd_fast.enet_coordinate_descent()  
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\9477209.py:16: UserWarning:  
With alpha=0, this algorithm does not converge well. You are advised to use t  
he LinearRegression estimator  
    en_obj.fit(x, y)  
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d  
escent.py:647: UserWarning: Coordinate descent with no regularization may lea  
d to unexpected results and is discouraged.  
    model = cd_fast.enet_coordinate_descent()  
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d  
escent.py:647: ConvergenceWarning: Objective did not converge. You might want  
to increase the number of iterations, check the scale of the features or cons  
ider increasing regularisation. Duality gap: 7.729e+03, tolerance: 6.179e+00  
Linear regression models with null weight for the l1 regularization term are  
more efficiently fitted using one of the solvers implemented in sklearn.linea  
r_model.Ridge/RidgeCV instead.  
    model = cd_fast.enet_coordinate_descent()  
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\9477209.py:18: UserWarning:  
With alpha=0, this algorithm does not converge well. You are advised to use t  
he LinearRegression estimator  
    en_obj.fit(x_val, y_val)  
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d  
escent.py:647: UserWarning: Coordinate descent with no regularization may lea  
d to unexpected results and is discouraged.  
    model = cd_fast.enet_coordinate_descent()  
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
```

```
escent.py:647: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 2.257e+03, tolerance: 1.870e+00
Linear regression models with null weight for the l1 regularization term are
more efficiently fitted using one of the solvers implemented in sklearn.linear_
r_model.Ridge/RidgeCV instead.

    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\9477209.py:16: UserWarning:
With alpha=0, this algorithm does not converge well. You are advised to use t
he LinearRegression estimator
    en_obj.fit(x, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: UserWarning: Coordinate descent with no regularization may lea
d to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 7.340e+03, tolerance: 5.880e+00
Linear regression models with null weight for the l1 regularization term are
more efficiently fitted using one of the solvers implemented in sklearn.linear_
r_model.Ridge/RidgeCV instead.

    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\9477209.py:18: UserWarning:
With alpha=0, this algorithm does not converge well. You are advised to use t
he LinearRegression estimator
    en_obj.fit(x_val, y_val)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: UserWarning: Coordinate descent with no regularization may lea
d to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 2.673e+03, tolerance: 2.163e+00
Linear regression models with null weight for the l1 regularization term are
more efficiently fitted using one of the solvers implemented in sklearn.linear_
r_model.Ridge/RidgeCV instead.

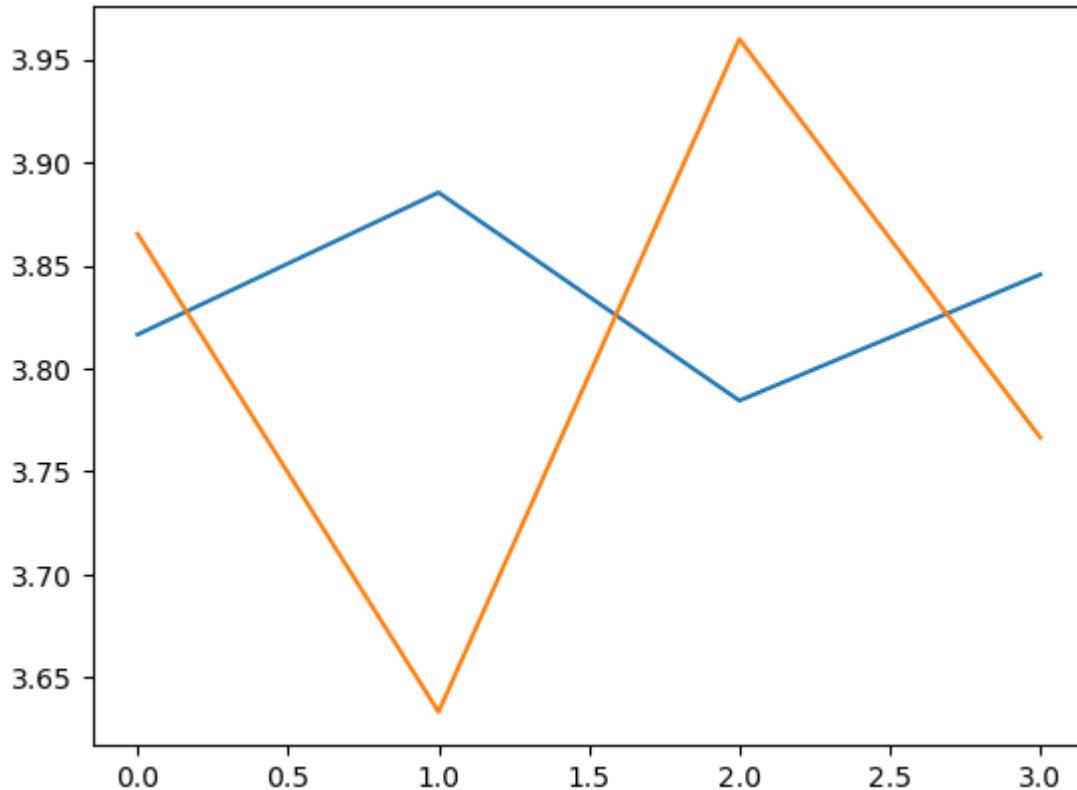
    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\9477209.py:16: UserWarning:
With alpha=0, this algorithm does not converge well. You are advised to use t
he LinearRegression estimator
    en_obj.fit(x, y)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: UserWarning: Coordinate descent with no regularization may lea
d to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_d
escent.py:647: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or consider
increasing regularisation. Duality gap: 7.579e+03, tolerance: 6.105e+00
Linear regression models with null weight for the l1 regularization term are
more efficiently fitted using one of the solvers implemented in sklearn.linear_
r_model.Ridge/RidgeCV instead.

    model = cd_fast.enet_coordinate_descent(
C:\Users\jeets\AppData\Local\Temp\ipykernel_29384\9477209.py:18: UserWarning:
With alpha=0, this algorithm does not converge well. You are advised to use t
```

```

he LinearRegression estimator
    en_obj.fit(x_val, y_val)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: UserWarning: Coordinate descent with no regularization may lead to unexpected results and is discouraged.
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.419e+03, tolerance: 1.902e+00
Linear regression models with null weight for the l1 regularization term are more efficiently fitted using one of the solvers implemented in sklearn.linear_model.Ridge/RidgeCV instead.
    model = cd_fast.enet_coordinate_descent()

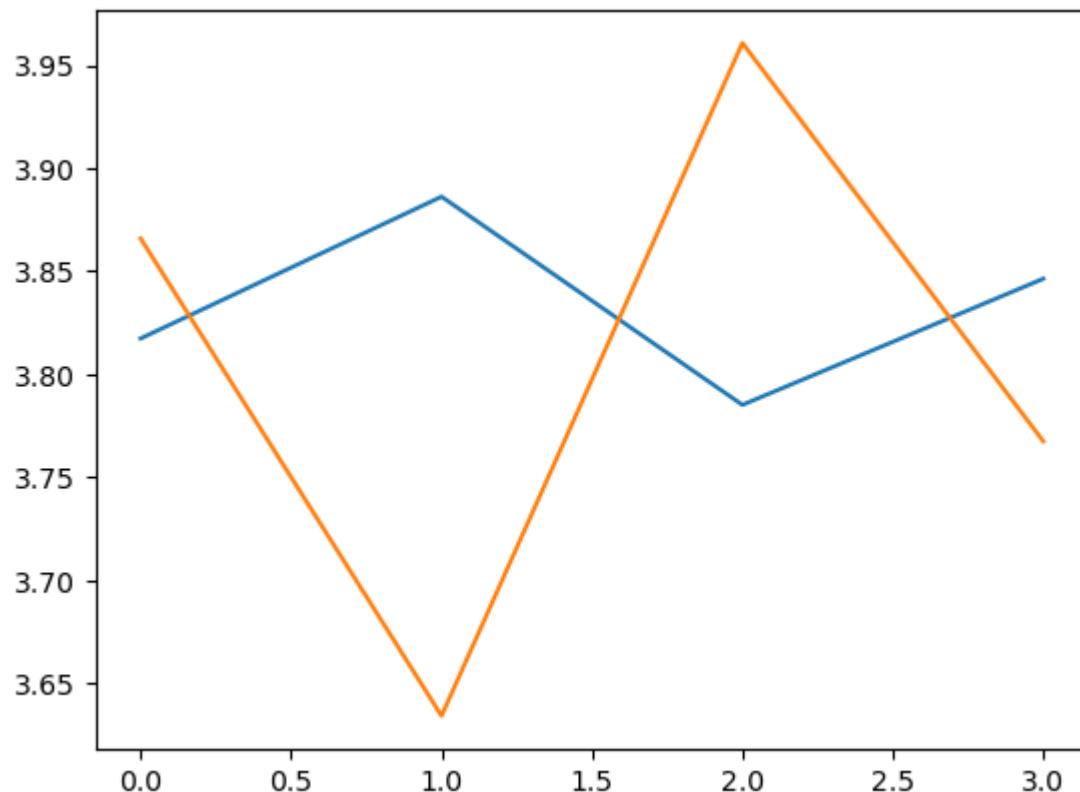
```



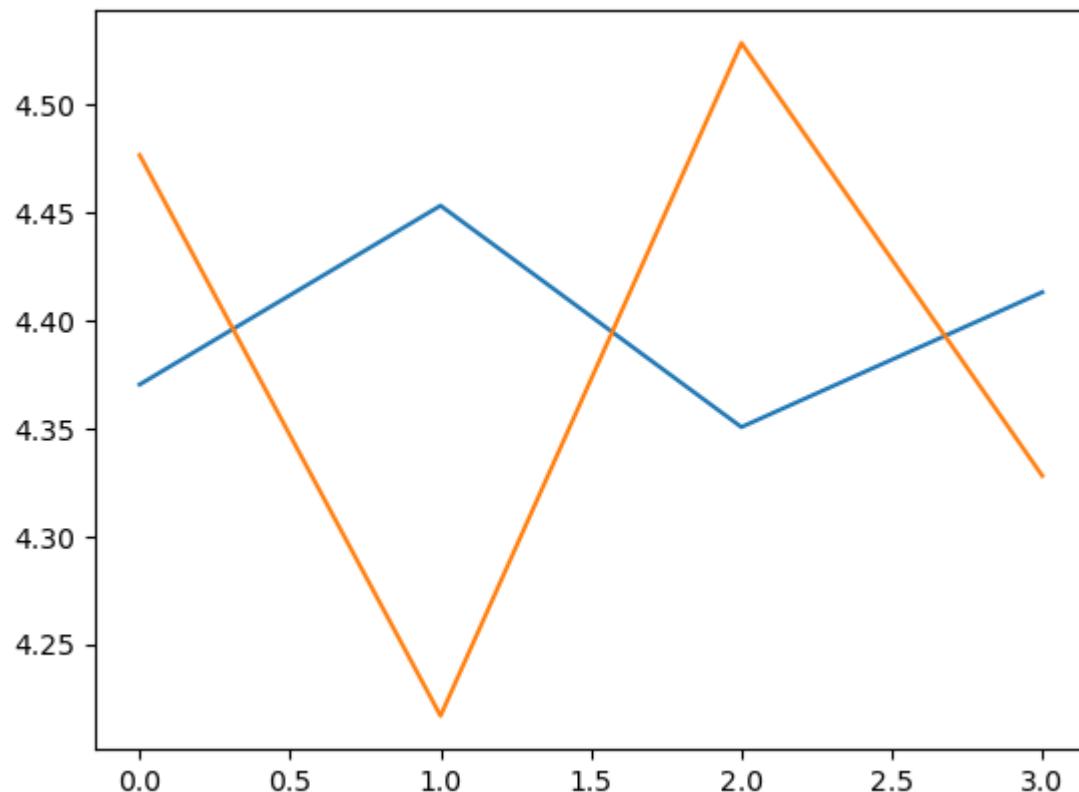
```

Alpha value: 0.001
0
Train error: 3.8173723278108436
Validation error: 3.865869484582345
1
Train error: 3.8861806530765493
Validation error: 3.6342771518819412
2
Train error: 3.785155090783941
Validation error: 3.96073348210214
3
Train error: 3.846346434889784
Validation error: 3.767491162795419

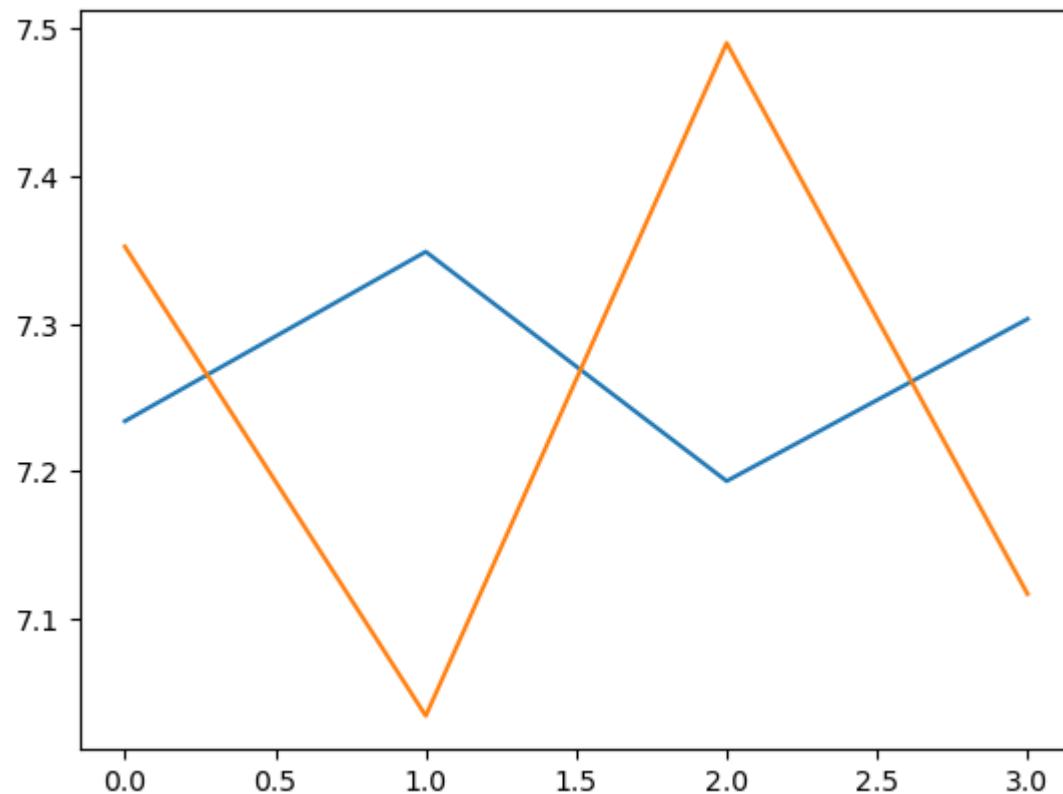
```



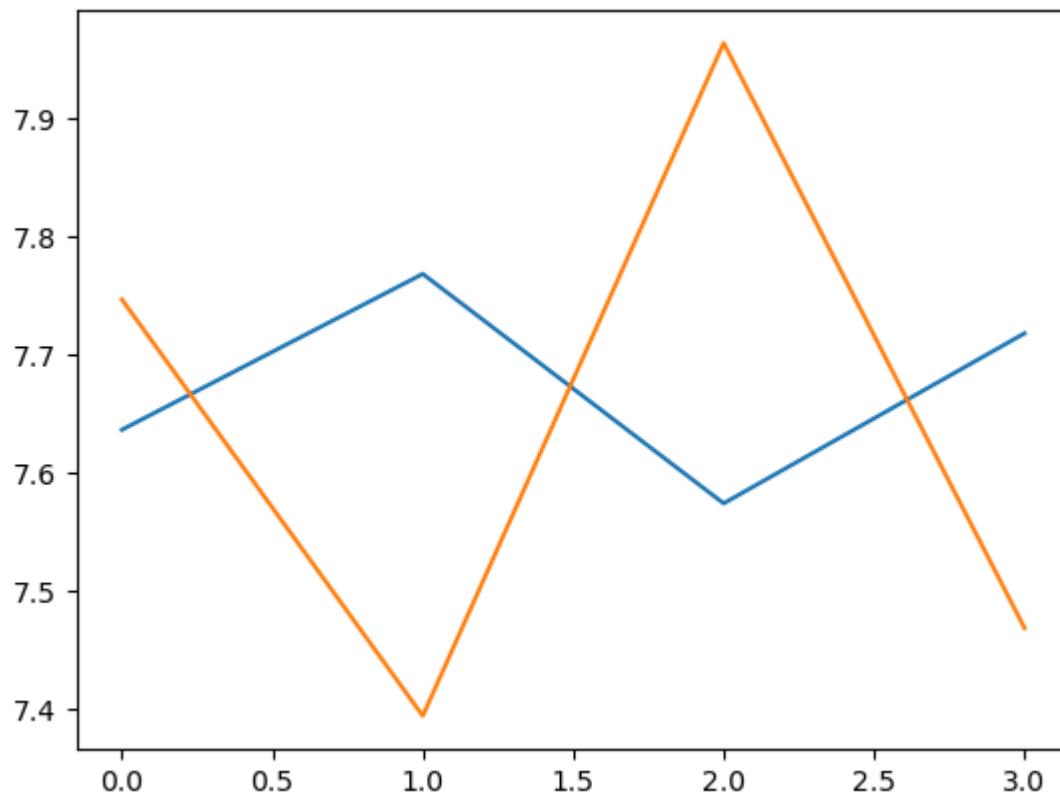
```
Alpha value: 1
0
Train error: 4.3702365712793885
Validation error: 4.4765321353181315
1
Train error: 4.453116189790851
Validation error: 4.216748083891603
2
Train error: 4.350523676313172
Validation error: 4.528462199812707
3
Train error: 4.413012157576303
Validation error: 4.327948718273263
```



```
Alpha value: 10
0
Train error: 7.234094340632255
Validation error: 7.35235457830052
1
Train error: 7.348853346128042
Validation error: 7.0345224480322
2
Train error: 7.193514583679743
Validation error: 7.4902096613872375
3
Train error: 7.303280342238653
Validation error: 7.117028796466902
```



```
Alpha value: 100
0
Train error: 7.63616284494717
Validation error: 7.746478390300121
1
Train error: 7.768110993028431
Validation error: 7.393978527563255
2
Train error: 7.573816670038543
Validation error: 7.963619835739025
3
Train error: 7.717837191561589
Validation error: 7.468266103185494
```



Answer:- The above results show the performance of different linear regression models with different hyperparameters on the dataset. From the results, we can see that the choice of hyperparameters such as learning rate and penalty factor has a significant impact on the performance of the models.

For example, we can see that the Ridge regression model with a learning rate of 1 performs better than the same model with a learning rate of 0, 0.001, or 10. Similarly, we can see that the Lasso and ElasticNet models perform better with a learning rate of 0.001 than 0 or 1.

In the case of SGD, we can see that the performance of the model is influenced by the learning rate and penalty factor. Among the SGD models, we can see that the SGD model with a learning rate of 0.0001 and no penalty performs the best. If we want to trade-off accuracy for best-fit model, we can select the model with lowest learning rate.

Based on the above results, we can conclude that the SGD model with a learning rate of 0.0001 and no penalty factor is the best performing model among the linear regression models.

Part 6 FIX!

added ridge, lasso, etc for polynomial regressor.

Repeat the previous step with polynomial regression. Using validation loss, explore if your model overfits/underfits the data.

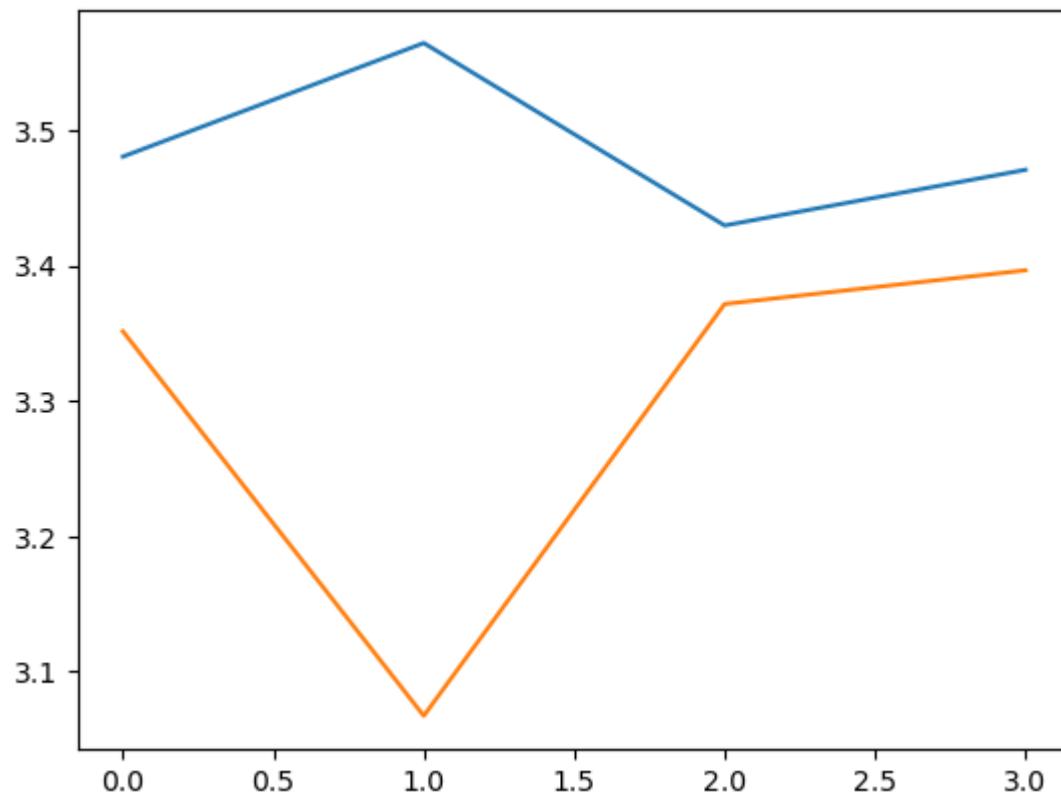
In [29]:

```

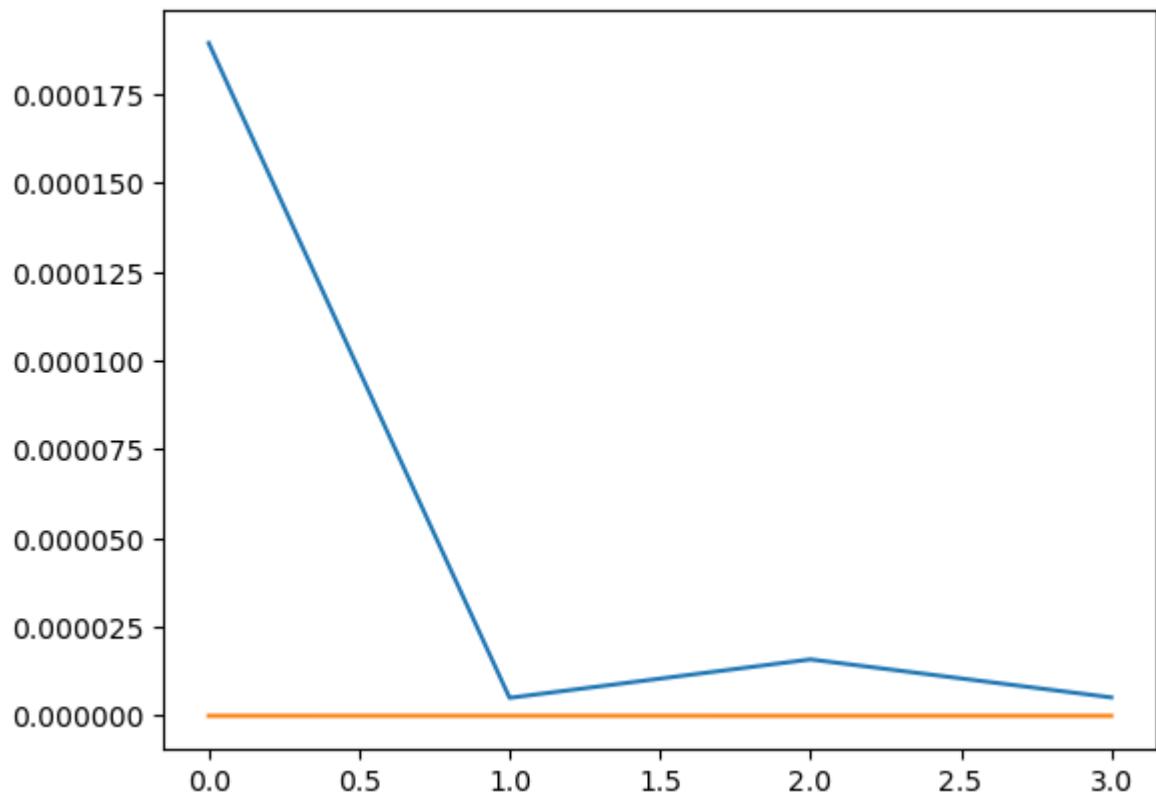
1 # Polynomial regression using various degrees
2
3 poly_degree = [2,5,10]
4
5 for r in poly_degree:
6     print("Degree:", r)
7     train_plot = []
8     val_plot = []
9     poly_features = PolynomialFeatures(degree=r, include_bias=False)
10    lin_reg = LinearRegression()
11    for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):
12        print(cnt)
13        x = x_train_2[var[0],:]
14        y = y_train_2[var[0]]
15        x_val = x_train_2[var[1],:]
16        y_val = y_train_2[var[1]]
17
18        X_poly_1 = poly_features.fit_transform(x)
19        lin_reg.fit(X_poly_1, y)
20        y_pred_train=lin_reg.predict(X_poly_1)
21
22        X_poly_2 = poly_features.fit_transform(x_val)
23        lin_reg.fit(X_poly_2, y_val)
24        y_pred_val=lin_reg.predict(X_poly_2)
25
26        print("Train error:",np.sqrt(mse(y_pred_train,y)))
27        print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
28        train_plot.append(np.sqrt(mse(y_pred_train,y)))
29        val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
30
31        X_test_poly = poly_features.fit_transform(x_test_2)
32        y_pred_test = lin_reg.predict(X_test_poly)
33        metric_dict['Polynomial degree= '+str(r)] = np.sqrt(mse(y_pred_te:
34        plt.plot([0,1,2,3], train_plot, label = "Training Error")
35        plt.plot([0,1,2,3], val_plot, label = "Val Error")
36        plt.show()
37        print('\n')

```

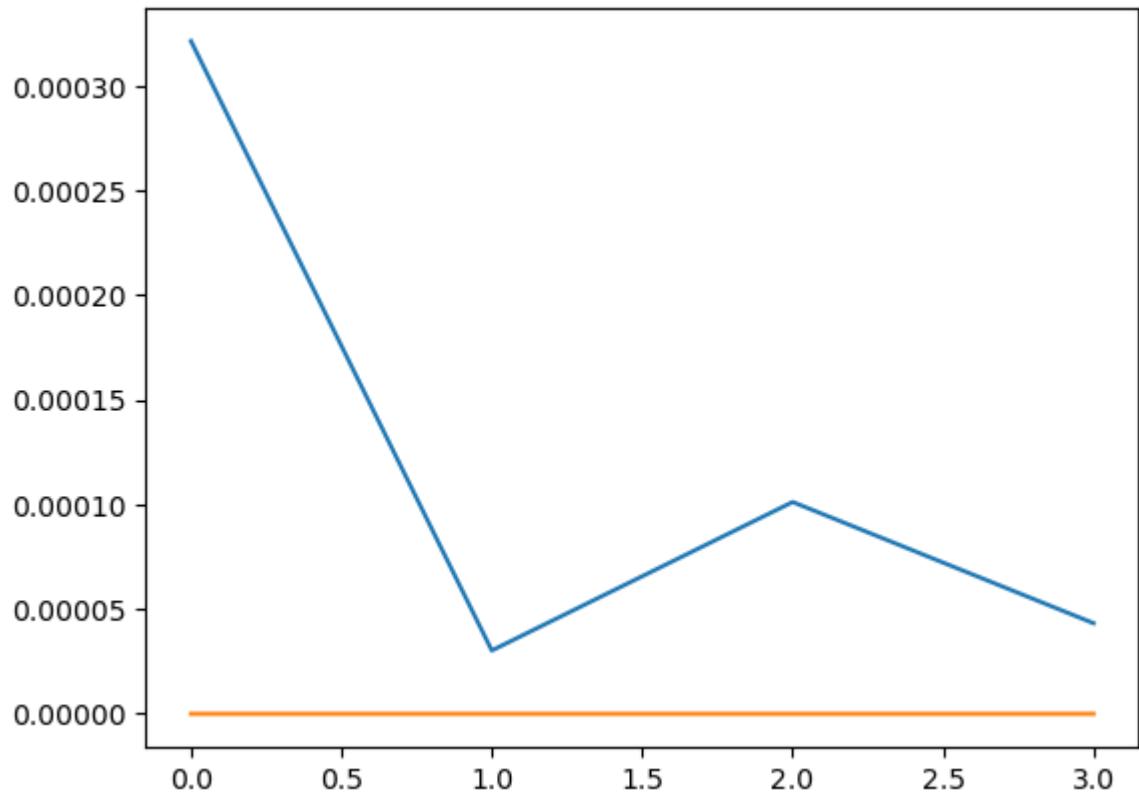
Degree: 2
0
Train error: 3.480439443923598
Validation error: 3.351374064617879
1
Train error: 3.5643830729882917
Validation error: 3.067284862232806
2
Train error: 3.4295680768999164
Validation error: 3.371557380615077
3
Train error: 3.470715748512232
Validation error: 3.3964585668919063



```
Degree: 5
0
Train error: 0.00018939667881141212
Validation error: 2.451125219907198e-09
1
Train error: 5.062712854654552e-06
Validation error: 7.071578381275772e-09
2
Train error: 1.5878902551751587e-05
Validation error: 1.5786709859717238e-09
3
Train error: 5.1821998857735465e-06
Validation error: 1.7168214995055538e-09
```



```
Degree: 10
0
Train error: 0.0003214144263393508
Validation error: 4.681995426008565e-09
1
Train error: 3.024337400238046e-05
Validation error: 2.6741039169683448e-09
2
Train error: 0.00010118785198348438
Validation error: 1.8587230792280205e-09
3
Train error: 4.328954718870466e-05
Validation error: 1.8945759387736444e-09
```



In [30]:

```

1 # Polynomial regression using ridge
2
3 poly_degree = [2,5,10]
4
5 for r in poly_degree:
6     print("Degree:", r)
7     train_plot = []
8     val_plot = []
9     poly_features = PolynomialFeatures(degree=r, include_bias=False)
10    ridge = Ridge()
11    for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):
12        print(cnt)
13        x = x_train_2[var[0],:]
14        y = y_train_2[var[0]]
15        x_val = x_train_2[var[1],:]
16        y_val = y_train_2[var[1]]
17
18        X_poly_1 = poly_features.fit_transform(x)
19        ridge.fit(X_poly_1, y)
20        y_pred_train=ridge.predict(X_poly_1)
21
22        X_poly_2 = poly_features.fit_transform(x_val)
23        ridge.fit(X_poly_2, y_val)
24        y_pred_val=ridge.predict(X_poly_2)
25
26        print("Train error:",np.sqrt(mse(y_pred_train,y)))
27        print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
28        train_plot.append(np.sqrt(mse(y_pred_train,y)))
29        val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
30
31        X_test_poly = poly_features.fit_transform(x_test_2)
32        y_pred_test = ridge.predict(X_test_poly)
33        metric_dict['Polynomial degree for ridge= '+str(r)] = np.sqrt(mse)
34        plt.plot([0,1,2,3], train_plot, label = "Training Error")
35        plt.plot([0,1,2,3], val_plot, label = "Val Error")
36        plt.show()
37        print('\n')

```

Degree: 2

0

Train error: 3.6602909085079114

Validation error: 3.742760400310355

1

Train error: 3.737412793841303

Validation error: 3.5212350260206375

2

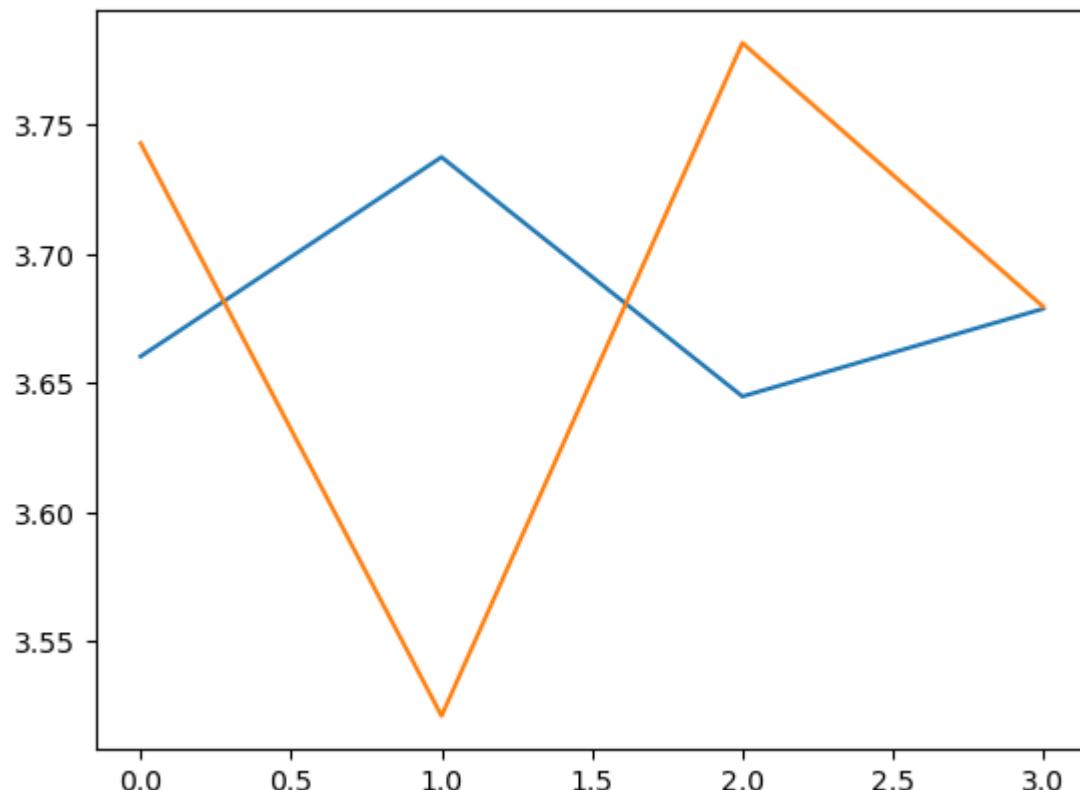
Train error: 3.6448151048627975

Validation error: 3.781549575381845

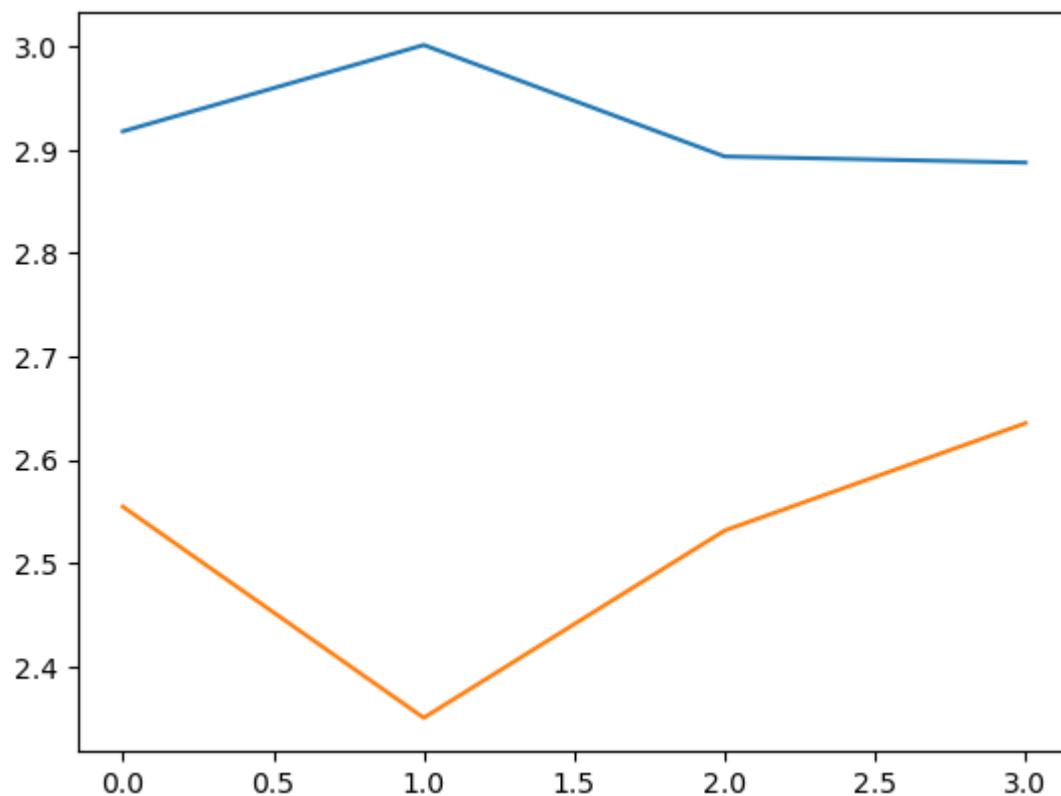
3

Train error: 3.6787414895501316

Validation error: 3.679502835184422



```
Degree: 5
0
Train error: 2.9179844038692258
Validation error: 2.554432896876155
1
Train error: 3.0015732246547286
Validation error: 2.349993268841313
2
Train error: 2.8936080554117076
Validation error: 2.5313295440635097
3
Train error: 2.887727509173558
Validation error: 2.6352916811729252
```



```
Degree: 10
```

```
0
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:19
6: UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
```

```
    warnings.warn(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:19
6: UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
```

```
    warnings.warn(
```

```
Train error: 2.8335826086736424
```

```
Validation error: 2.4871295827107733
```

```
1
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:19
6: UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
```

```
    warnings.warn(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:19
6: UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
```

```
    warnings.warn(
```

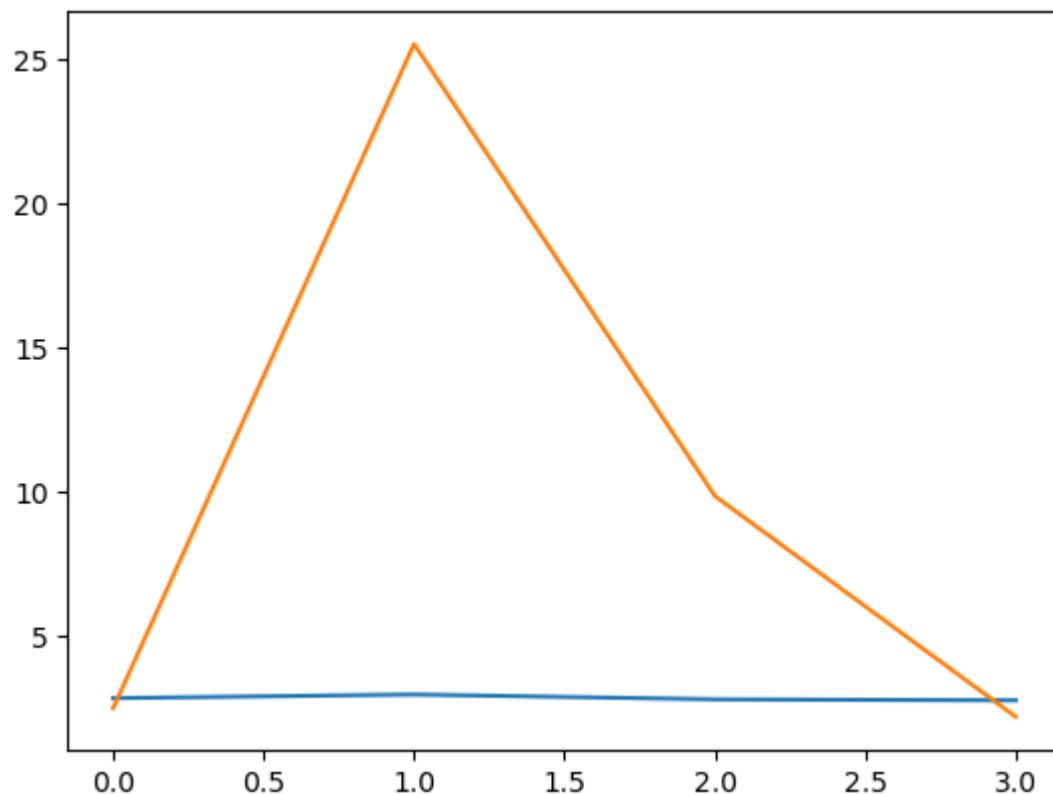
```
Train error: 2.9603305917893676
```

```
Validation error: 25.545416103975324
```

```
2
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:19
6: UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
    warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:19
6: UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
    warnings.warn(
Train error: 2.7915379552828097
Validation error: 9.852811014560062
3

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:19
6: UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
    warnings.warn(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_ridge.py:19
6: UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
    warnings.warn(
Train error: 2.7579822730904318
Validation error: 2.185656583260377
```



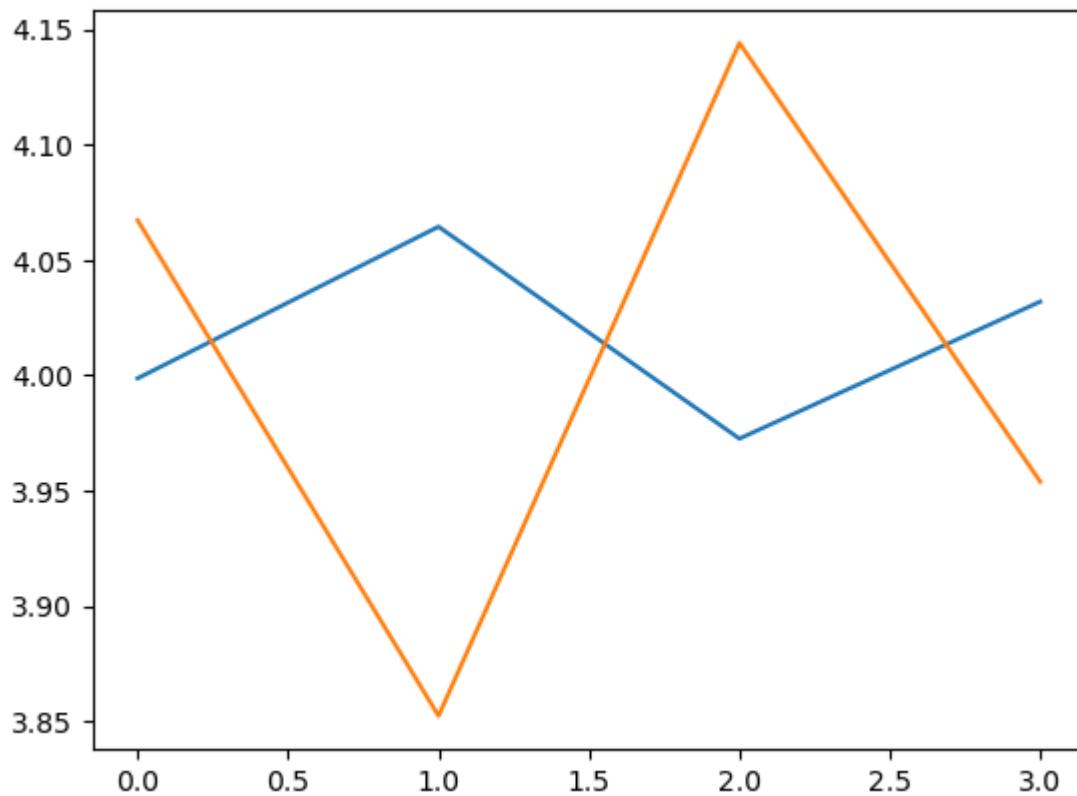
In [31]:

```

1 # Polynomial regression using Lasso
2
3 poly_degree = [2,5,10]
4
5 for r in poly_degree:
6     print("Degree:", r)
7     train_plot = []
8     val_plot = []
9     poly_features = PolynomialFeatures(degree=r, include_bias=False)
10    lasso = Lasso()
11    for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):
12        print(cnt)
13        x = x_train_2[var[0],:]
14        y = y_train_2[var[0]]
15        x_val = x_train_2[var[1],:]
16        y_val = y_train_2[var[1]]
17
18        X_poly_1 = poly_features.fit_transform(x)
19        lasso.fit(X_poly_1, y)
20        y_pred_train=lasso.predict(X_poly_1)
21
22        X_poly_2 = poly_features.fit_transform(x_val)
23        lasso.fit(X_poly_2, y_val)
24        y_pred_val=lasso.predict(X_poly_2)
25
26        print("Train error:",np.sqrt(mse(y_pred_train,y)))
27        print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
28        train_plot.append(np.sqrt(mse(y_pred_train,y)))
29        val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
30
31        X_test_poly = poly_features.fit_transform(x_test_2)
32        y_pred_test = lasso.predict(X_test_poly)
33        metric_dict['Polynomial degree for Lasso = '+str(r)] = np.sqrt(mse(y_pred_test,y))
34    plt.plot([0,1,2,3], train_plot, label = "Training Error")
35    plt.plot([0,1,2,3], val_plot, label = "Val Error")
36    plt.show()
37    print('\n')

```

Degree: 2
0
Train error: 3.99869187802109
Validation error: 4.067223344251254
1
Train error: 4.064428555541588
Validation error: 3.8523874684629775
2
Train error: 3.9725429633156844
Validation error: 4.144051958874811
3
Train error: 4.031961214029916
Validation error: 3.9538730532276265



Degree: 5

0

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.806e+03, tolerance: 5.971e+00
    model = cd_fast.enet_coordinate_descent()
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.097e+03, tolerance: 2.052e+00
    model = cd_fast.enet_coordinate_descent()
```

Train error: 3.592533436869899

Validation error: 3.553567654493042

1

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 7.053e+03, tolerance: 6.179e+00
    model = cd_fast.enet_coordinate_descent()
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.949e+03, tolerance: 1.870e+00
    model = cd_fast.enet_coordinate_descent()
```

```
Train error: 3.6794849903060785
Validation error: 3.3245845983410924
2
```

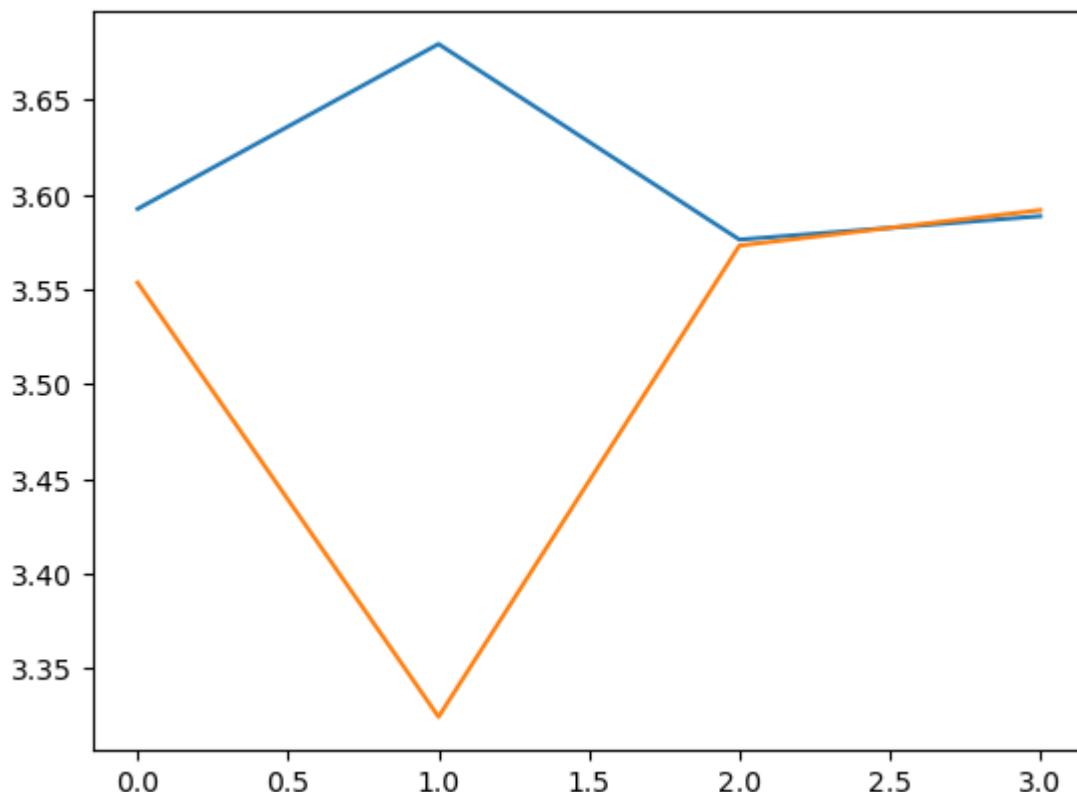
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.706e+03, tolerance: 5.880e+00
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.298e+03, tolerance: 2.163e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.5763128036350005
Validation error: 3.5731862311005425
3
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.756e+03, tolerance: 6.105e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.588692530944484
Validation error: 3.5918641573989802
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.274e+03, tolerance: 1.902e+00
    model = cd_fast.enet_coordinate_descent(
```



```
Degree: 10
```

```
0
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.282e+03, tolerance: 5.971e+00
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.499e+03, tolerance: 2.052e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.098252087785686
```

```
Validation error: 2.7967890257848333
```

```
1
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.611e+03, tolerance: 6.179e+00
    model = cd_fast.enet_coordinate_descent(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.287e+03, tolerance: 1.870e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.203328764270571
```

```
Validation error: 2.5856117987979634
```

```
2
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.273e+03, tolerance: 5.880e+00
    model = cd_fast.enet_coordinate_descent(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.480e+03, tolerance: 2.163e+00
    model = cd_fast.enet_coordinate_descent(
```

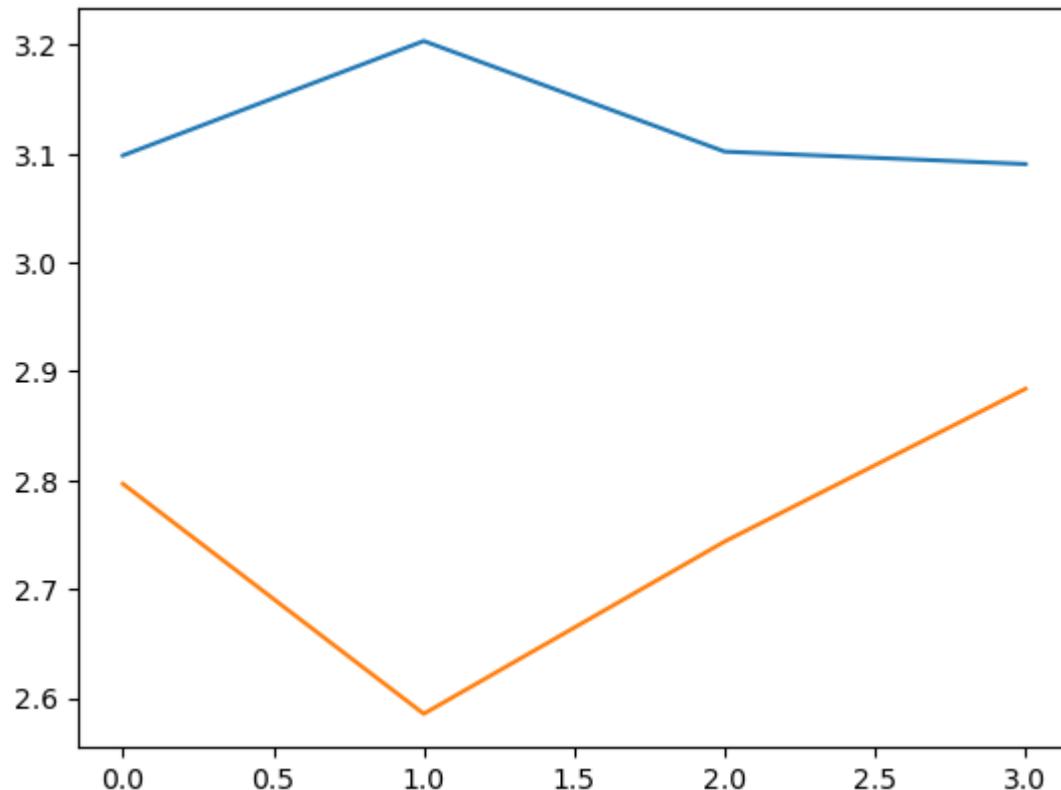
```
Train error: 3.1018034940110537
```

```
Validation error: 2.743790973073377
```

```
3
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.233e+03, tolerance: 6.105e+00
    model = cd_fast.enet_coordinate_descent()
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.616e+03, tolerance: 1.902e+00
    model = cd_fast.enet_coordinate_descent()

Train error: 3.0903722088814733
Validation error: 2.8840436612412015
```



In [32]: *Polynomial regression using SGD*

```

gd_pen = [None, 'l1', 'l2']
learning_rates = [0.000001, 0.00001, 0.0001, 0.001, 0.01]
n_epochs = 300
pol_deg = [2,5]
for penalty in sgd_pen:
    for lr in learning_rates:
        for pol in pol_deg:
            print("Polynomial degree =:", pol)
            print("Penalty:", penalty)
            print("Learning Rate:", lr)
            train_loss = []
            val_loss = []
            sgd_obj = SGDRegressor(penalty=penalty, learning_rate='constant', et
            poly_features = PolynomialFeatures(degree=pol, include_bias=False)
            X_train_poly = poly_features.fit_transform(x_train_2)
            X_val_poly = poly_features.transform(x_val)

            for epoch in range(n_epochs):
                sgd_obj.partial_fit(X_train_poly, y_train_2)
                y_pred_train = sgd_obj.predict(X_train_poly)
                y_pred_val = sgd_obj.predict(X_val_poly)
                train_loss.append(np.sqrt(mse(y_pred_train,y_train_2)))
                val_loss.append(np.sqrt(mse(y_pred_val,y_val)))

            plt.plot(range(n_epochs), train_loss, label = "Training Loss")
            plt.plot(range(n_epochs), val_loss, label = "Validation Loss")
            plt.legend()
            plt.show()

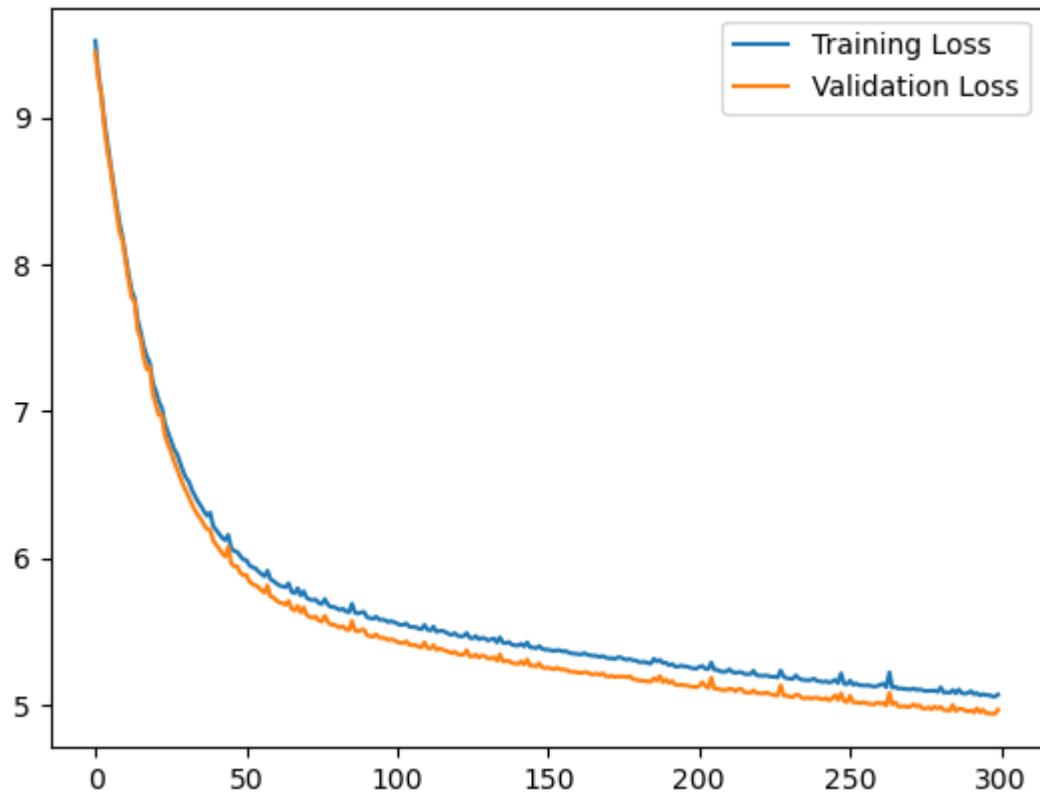
            # Calculate and print RMSE score between y_pred_val and y_test
            y_pred_test = sgd_obj.predict(poly_features.transform(x_test_2))
            rmse = np.sqrt(mse(y_pred_test, y_test_2))
            metric_dict['Polynomial degree for SGD= '+str(pol)+' Learning_rate='
            print("RMSE score on test data:", rmse)

```

Polynomial degree =: 2

Penalty: None

Learning Rate: 1e-06

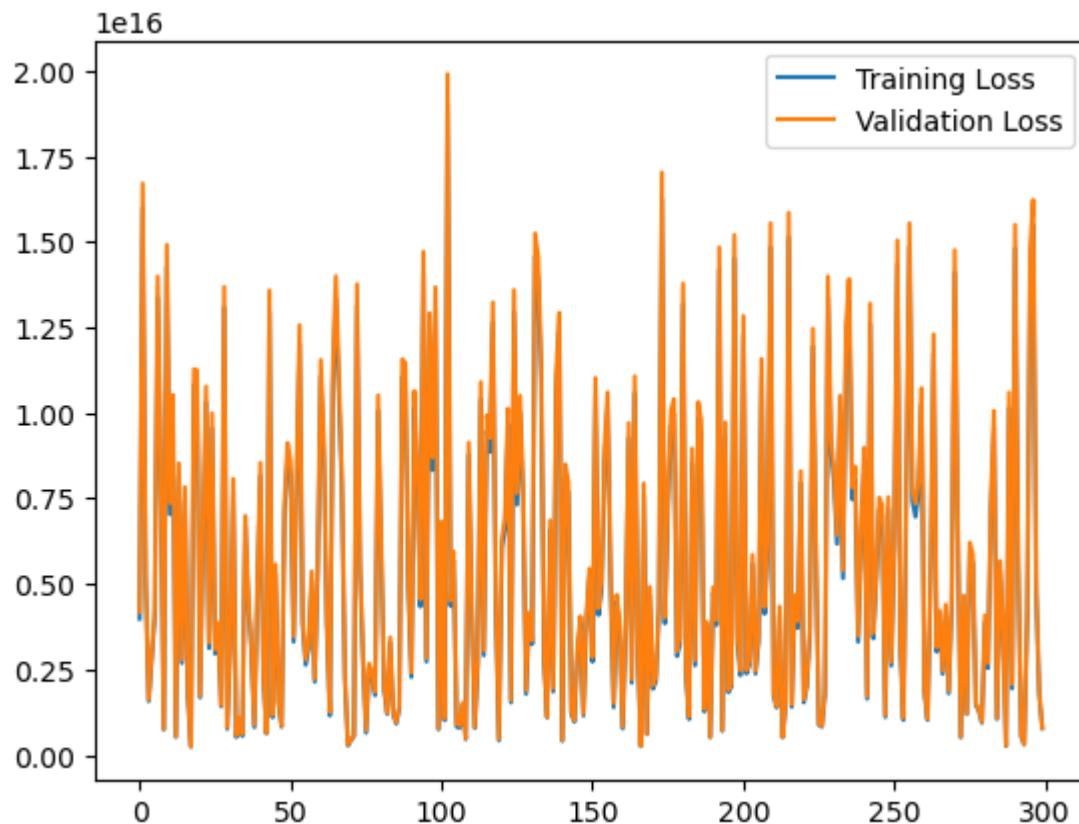


RMSE score on test data: 5.002836638073316

Polynomial degree =: 5

Penalty: None

Learning Rate: 1e-06

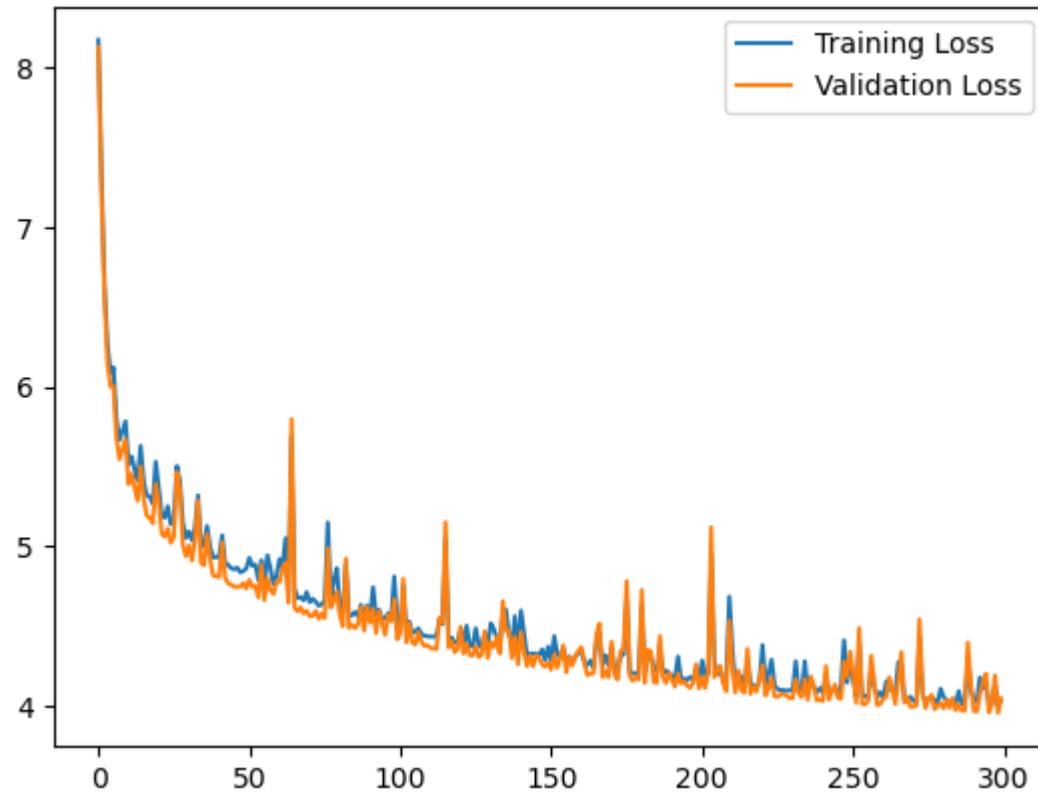


RMSE score on test data: 770328126591323.4

Polynomial degree =: 2

Penalty: None

Learning Rate: 1e-05

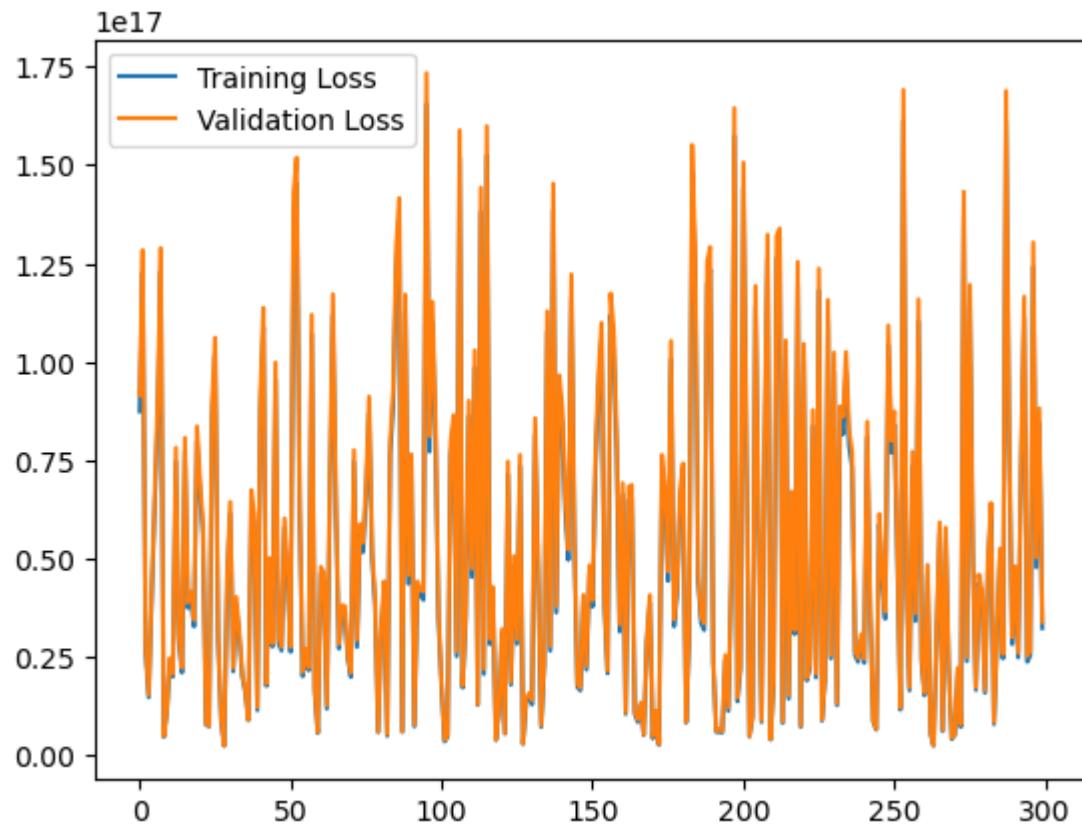


RMSE score on test data: 3.9412323193397887

Polynomial degree =: 5

Penalty: None

Learning Rate: 1e-05

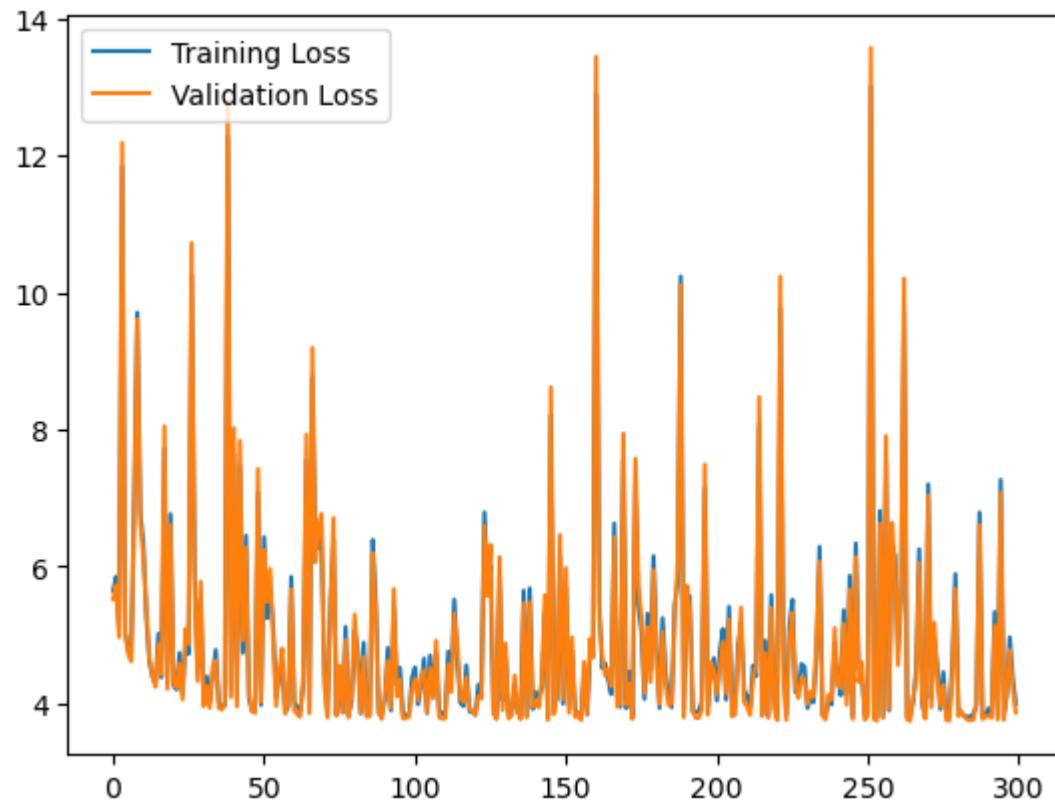


RMSE score on test data: 3.1996231306365056e+16

Polynomial degree =: 2

Penalty: None

Learning Rate: 0.0001

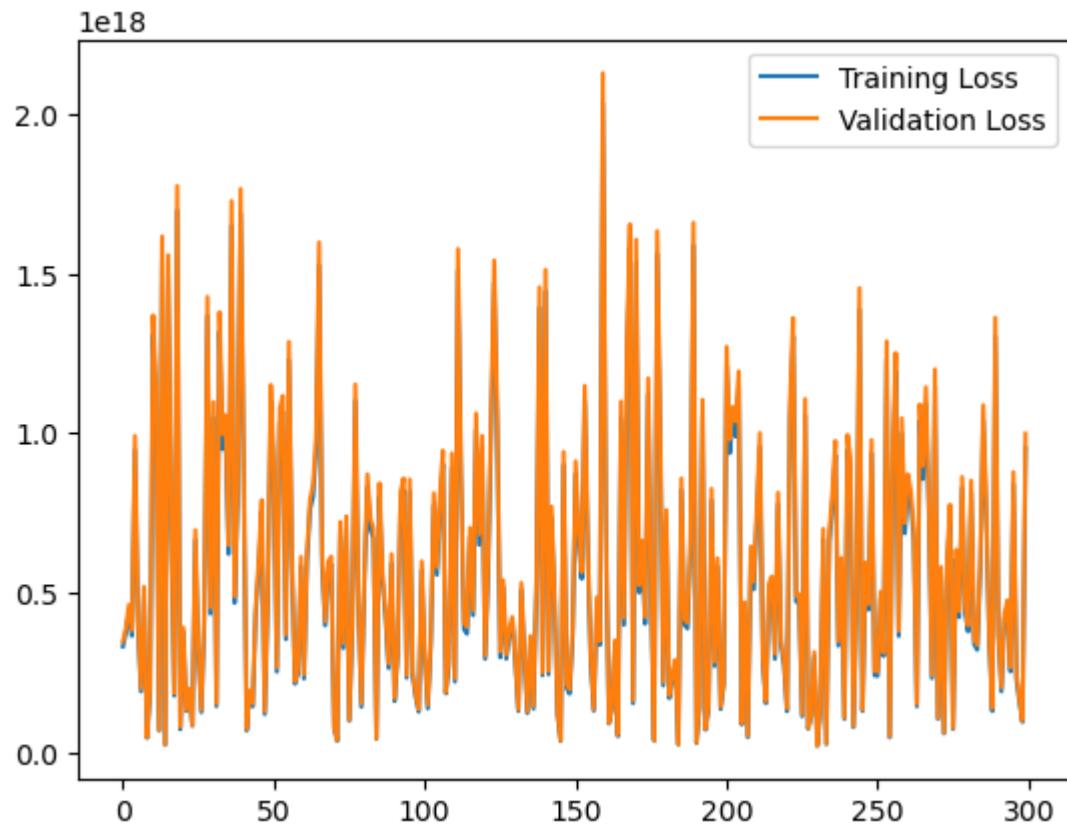


RMSE score on test data: 3.8693893503500916

Polynomial degree =: 5

Penalty: None

Learning Rate: 0.0001



RMSE score on test data: 9.397934997545254e+17

Polynomial degree =: 2

Penalty: None

Learning Rate: 0.001

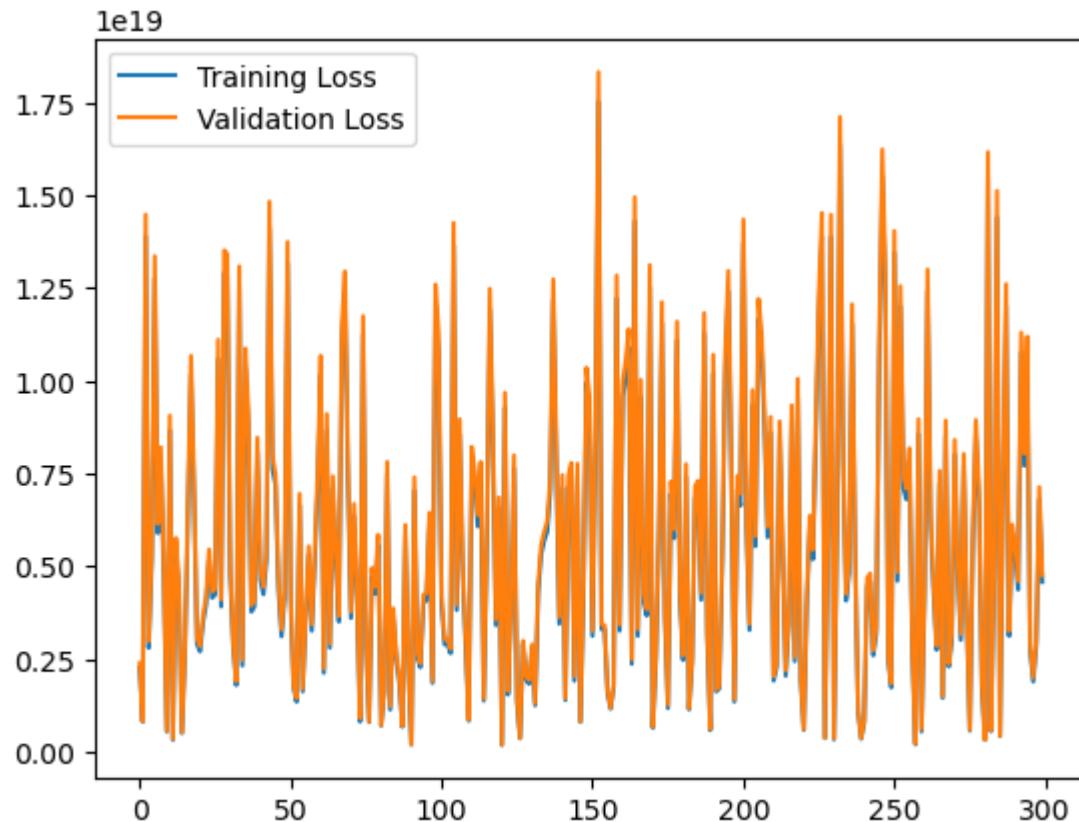


RMSE score on test data: 1411997102619.439

Polynomial degree =: 5

Penalty: None

Learning Rate: 0.001

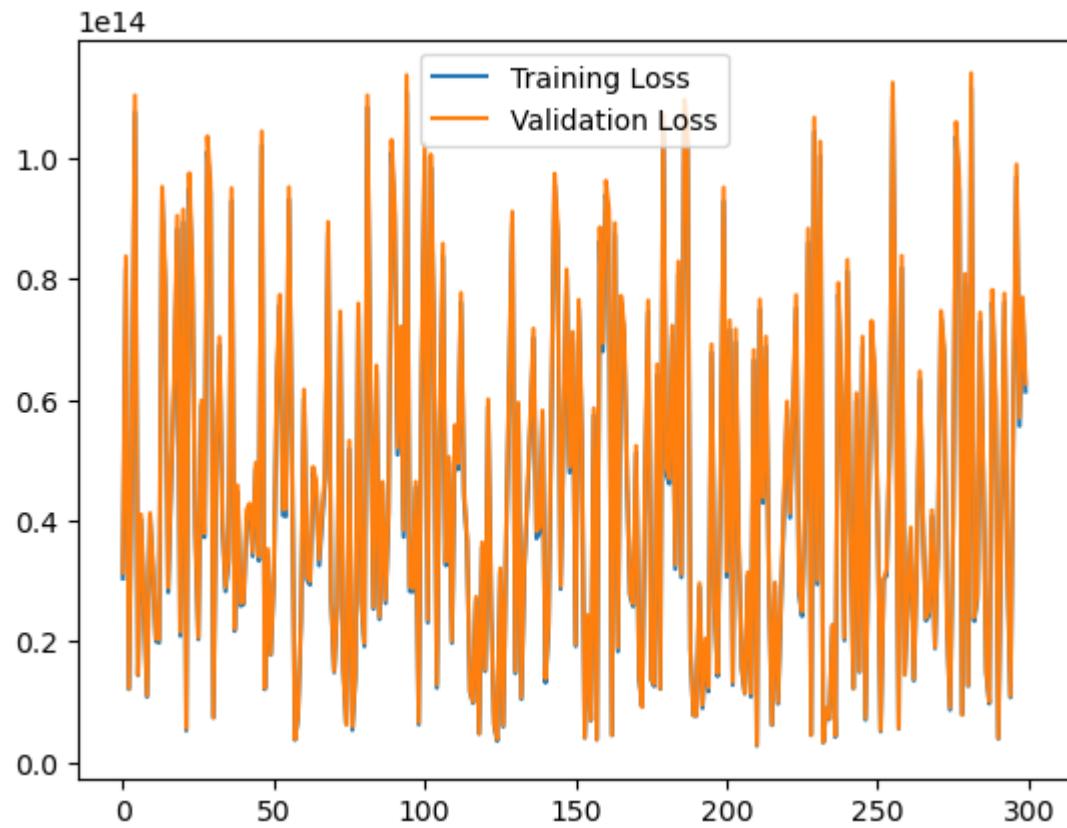


RMSE score on test data: 4.524759715010909e+18

Polynomial degree =: 2

Penalty: None

Learning Rate: 0.01

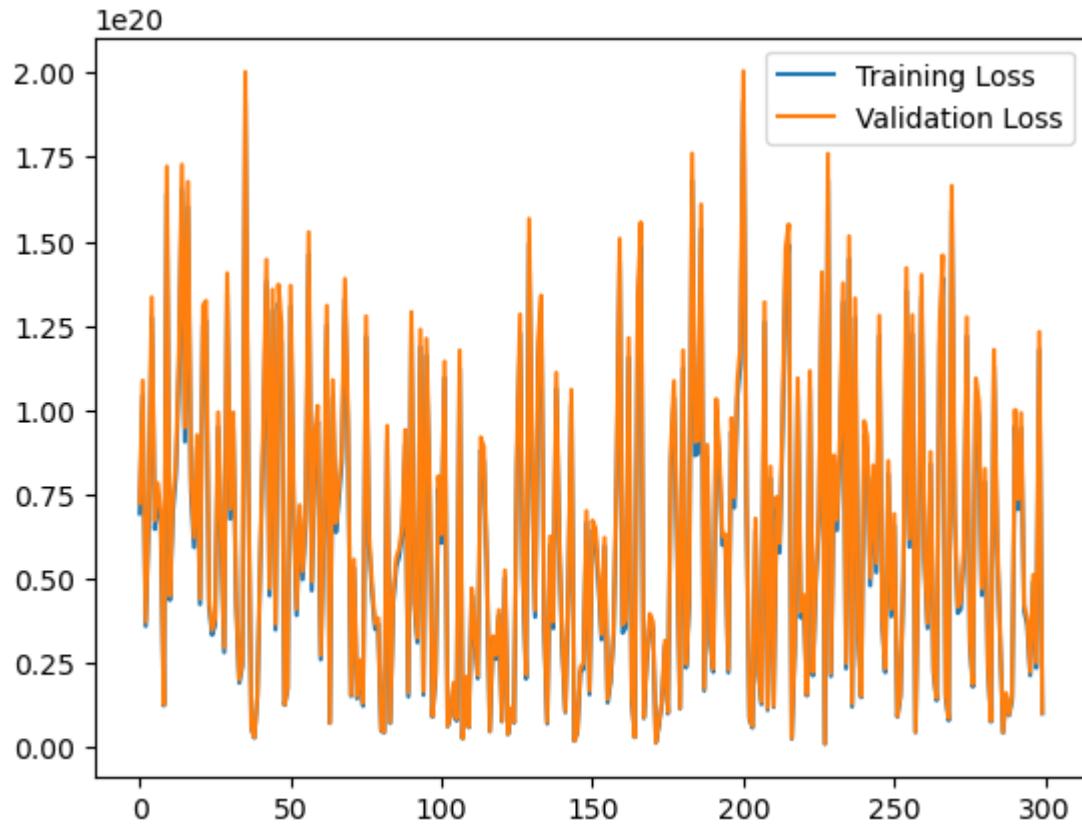


RMSE score on test data: 60805219622965.67

Polynomial degree =: 5

Penalty: None

Learning Rate: 0.01

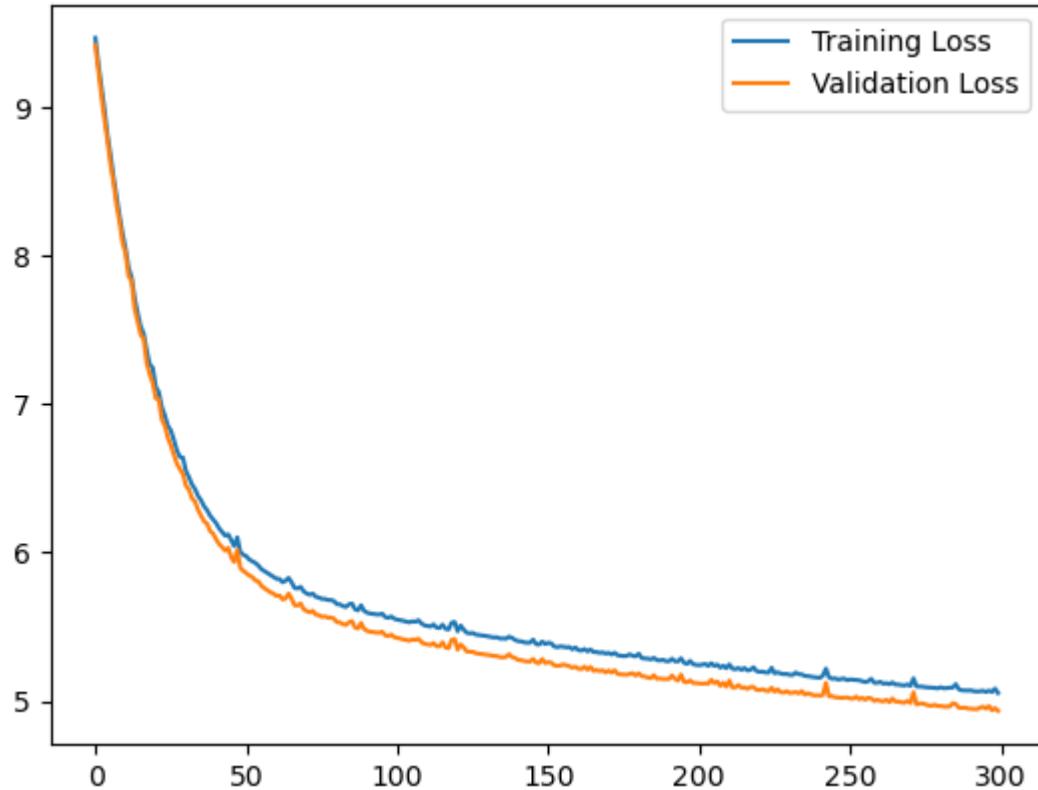


RMSE score on test data: 1.0193444668765428e+19

Polynomial degree =: 2

Penalty: 11

Learning Rate: 1e-06

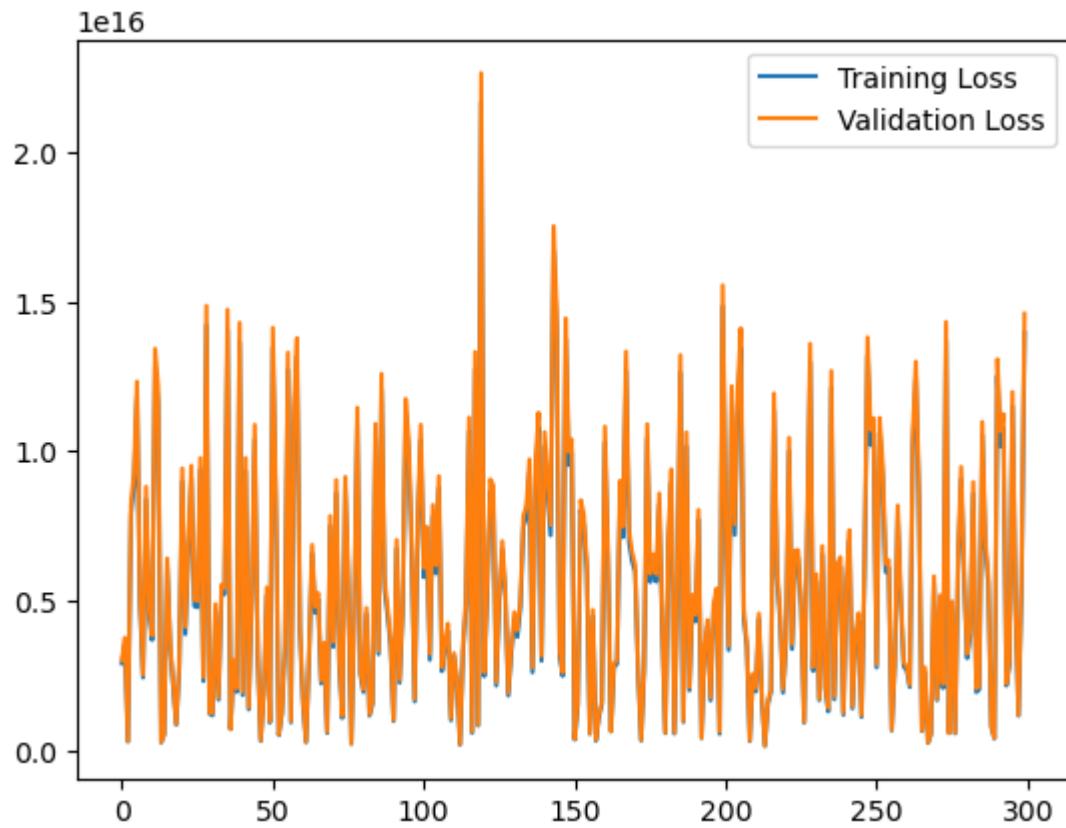


RMSE score on test data: 4.96685500381611

Polynomial degree =: 5

Penalty: 11

Learning Rate: 1e-06

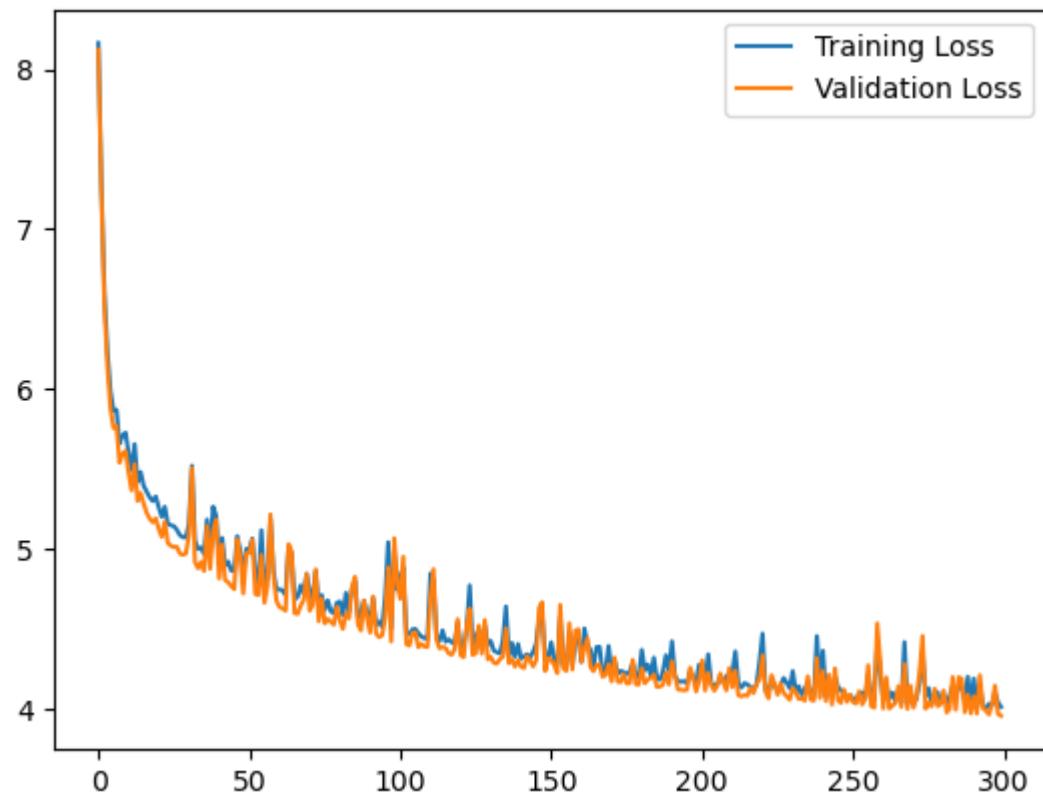


RMSE score on test data: 1.3761579679256478e+16

Polynomial degree =: 2

Penalty: 11

Learning Rate: 1e-05

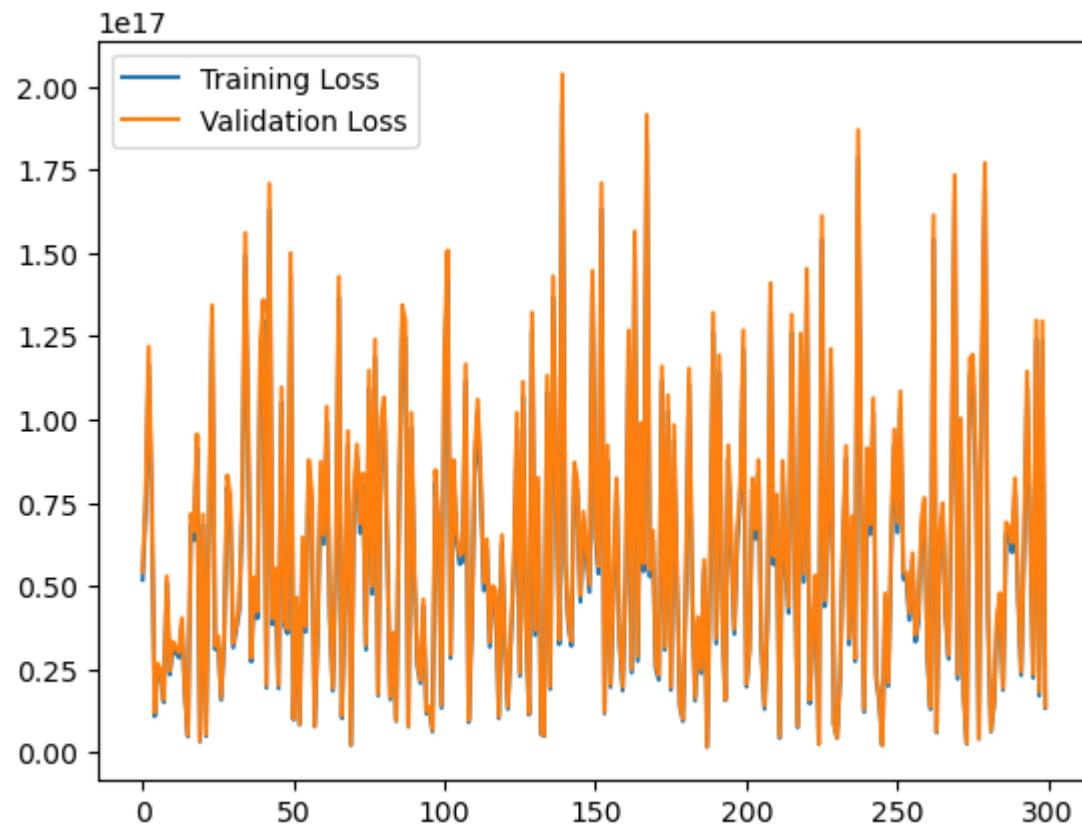


RMSE score on test data: 3.8999474542505728

Polynomial degree =: 5

Penalty: 11

Learning Rate: 1e-05

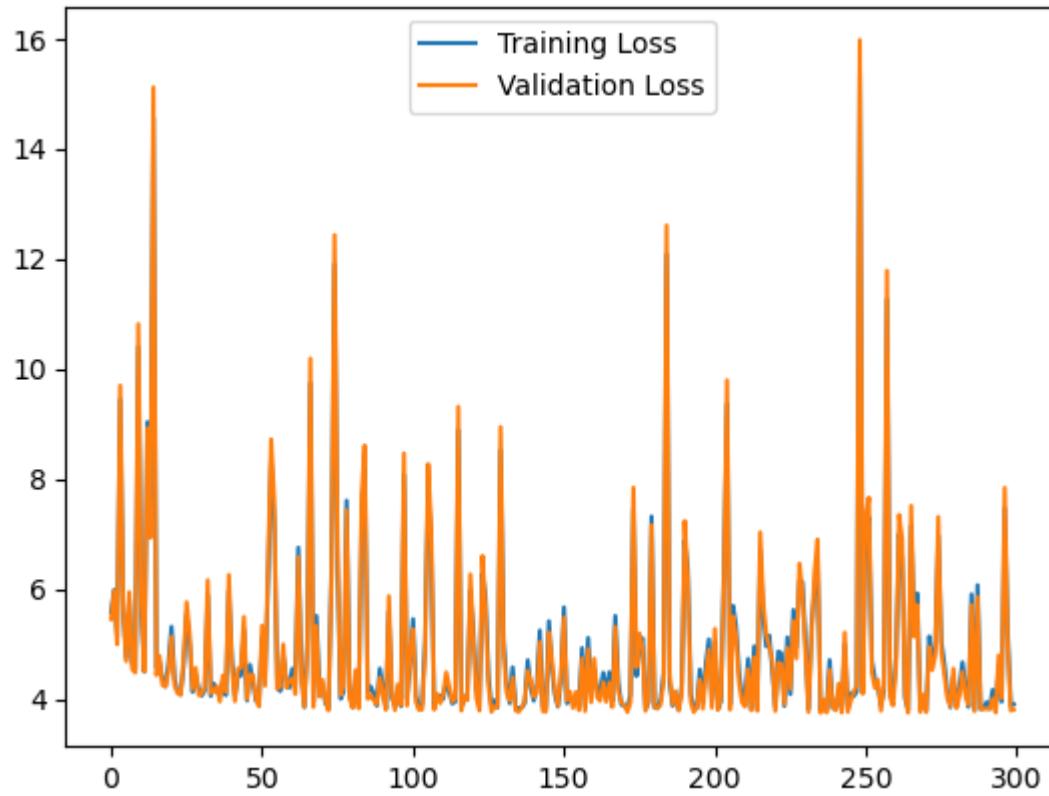


RMSE score on test data: 1.2971620554358054e+16

Polynomial degree =: 2

Penalty: 11

Learning Rate: 0.0001

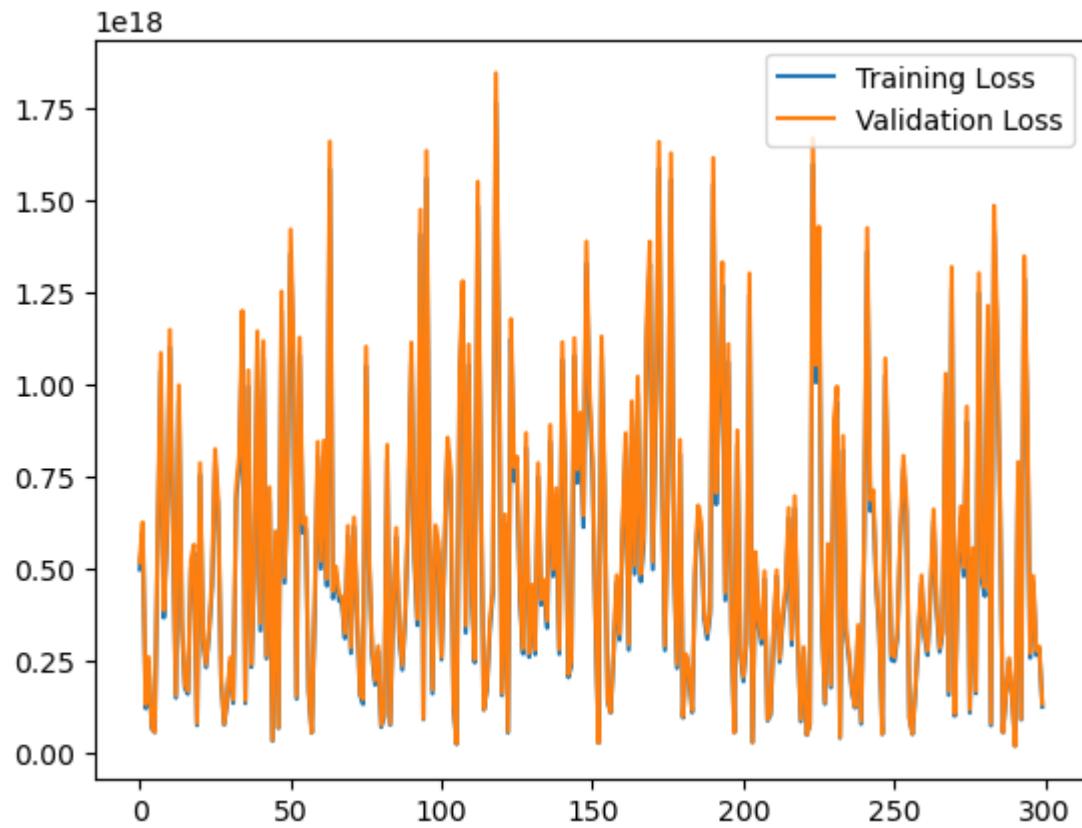


RMSE score on test data: 3.770076874012981

Polynomial degree =: 5

Penalty: 11

Learning Rate: 0.0001

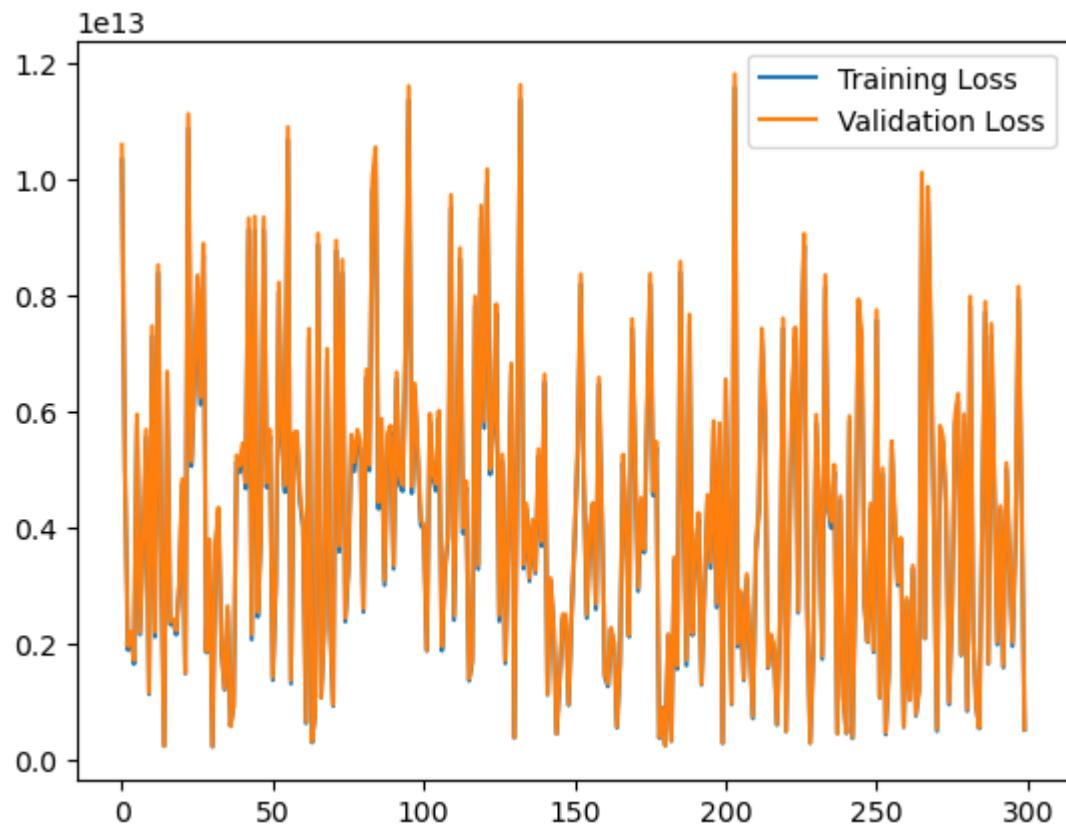


RMSE score on test data: 1.2534510663731338e+17

Polynomial degree =: 2

Penalty: 11

Learning Rate: 0.001

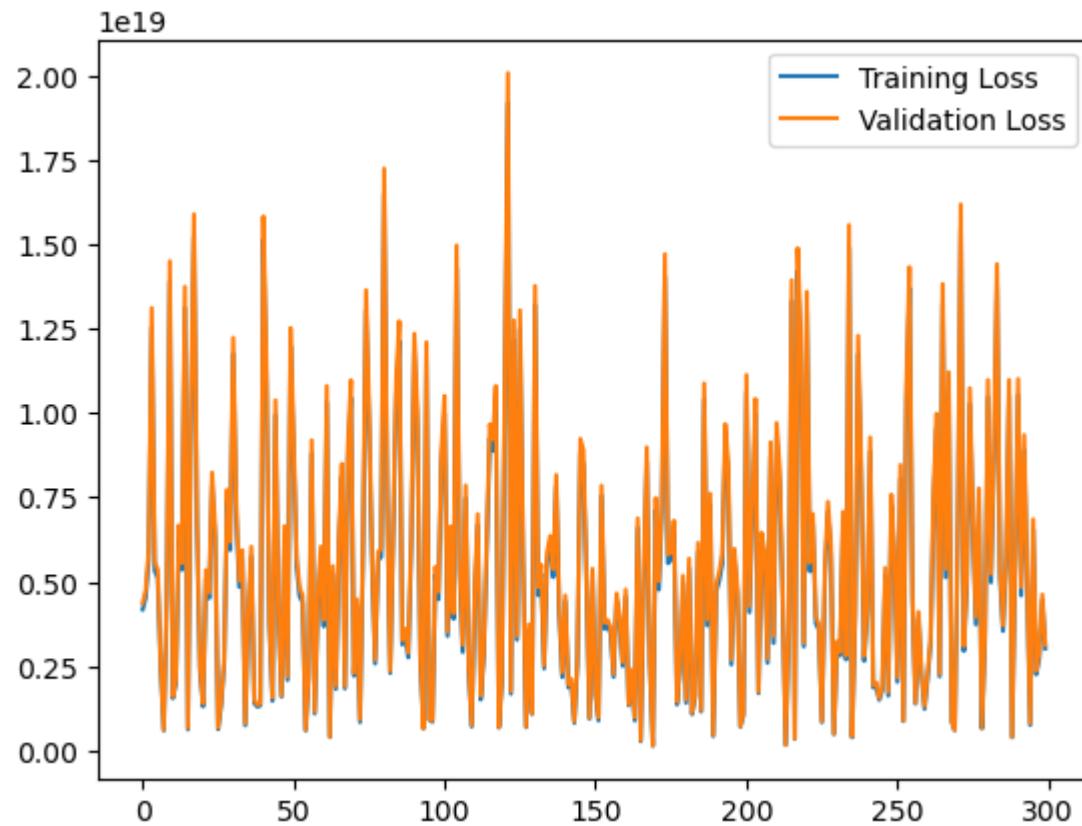


RMSE score on test data: 493769917474.58276

Polynomial degree =: 5

Penalty: 11

Learning Rate: 0.001

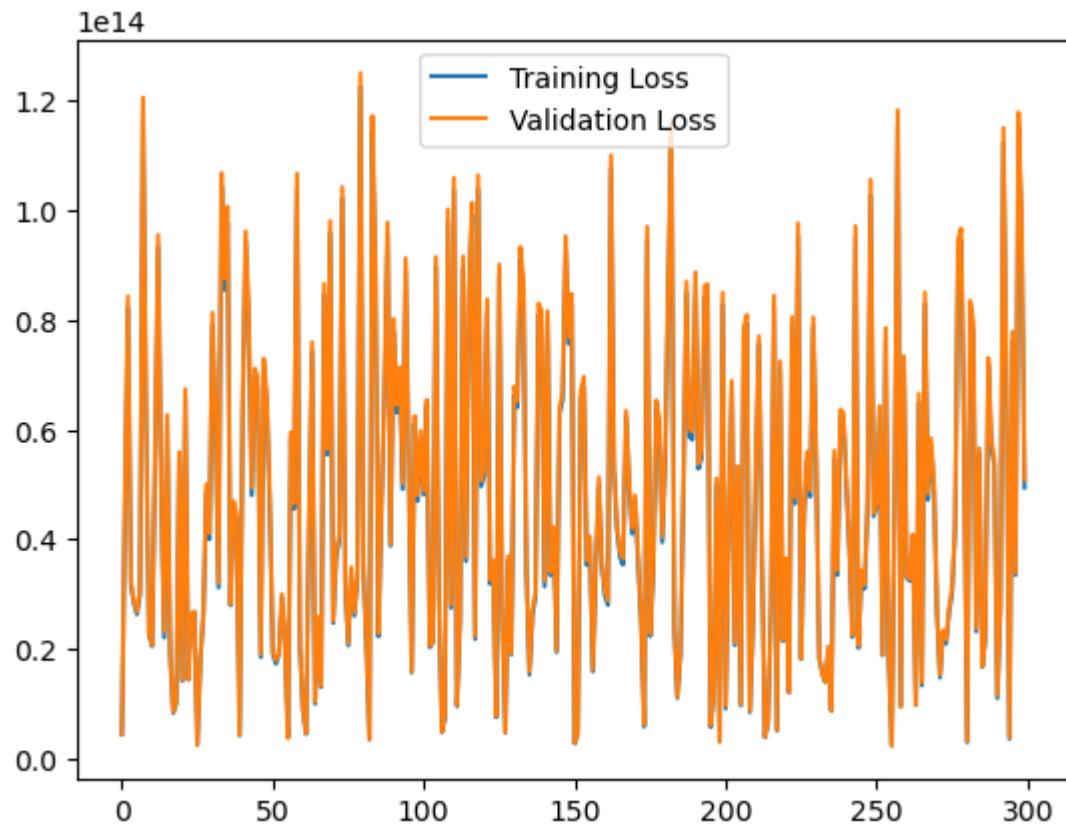


RMSE score on test data: 3.0020947762949115e+18

Polynomial degree =: 2

Penalty: 11

Learning Rate: 0.01

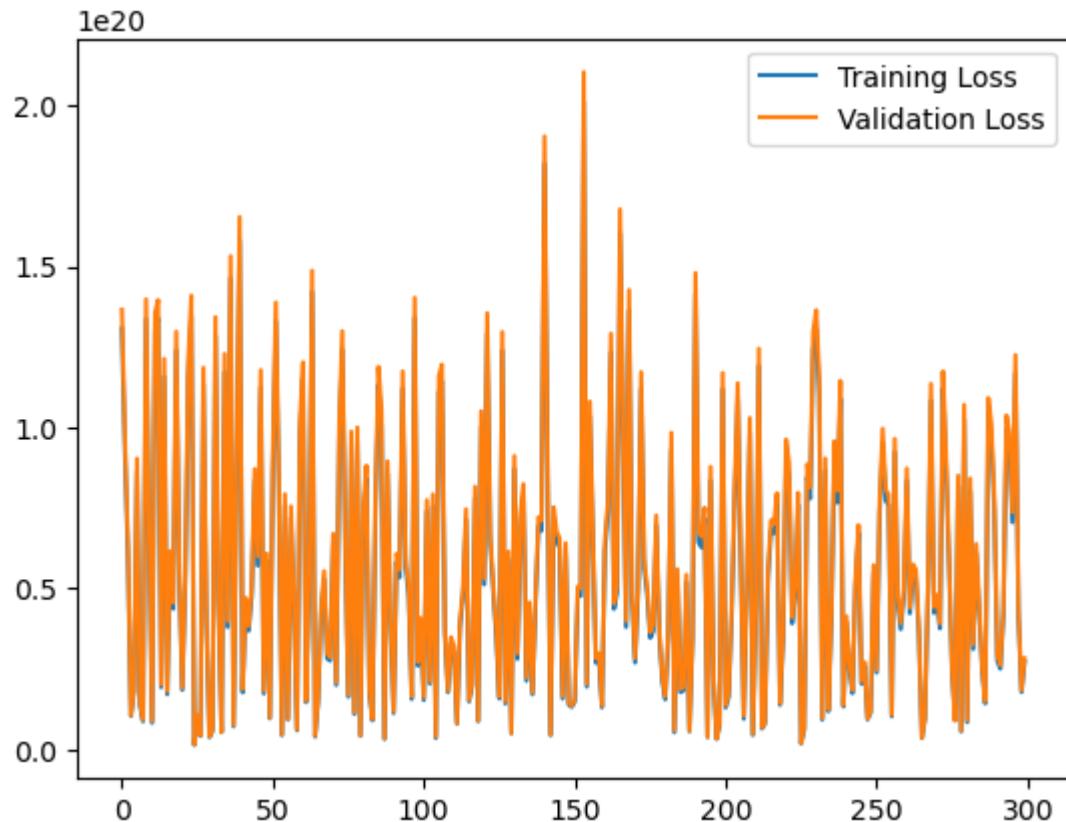


RMSE score on test data: 48963725981110.8

Polynomial degree =: 5

Penalty: 11

Learning Rate: 0.01

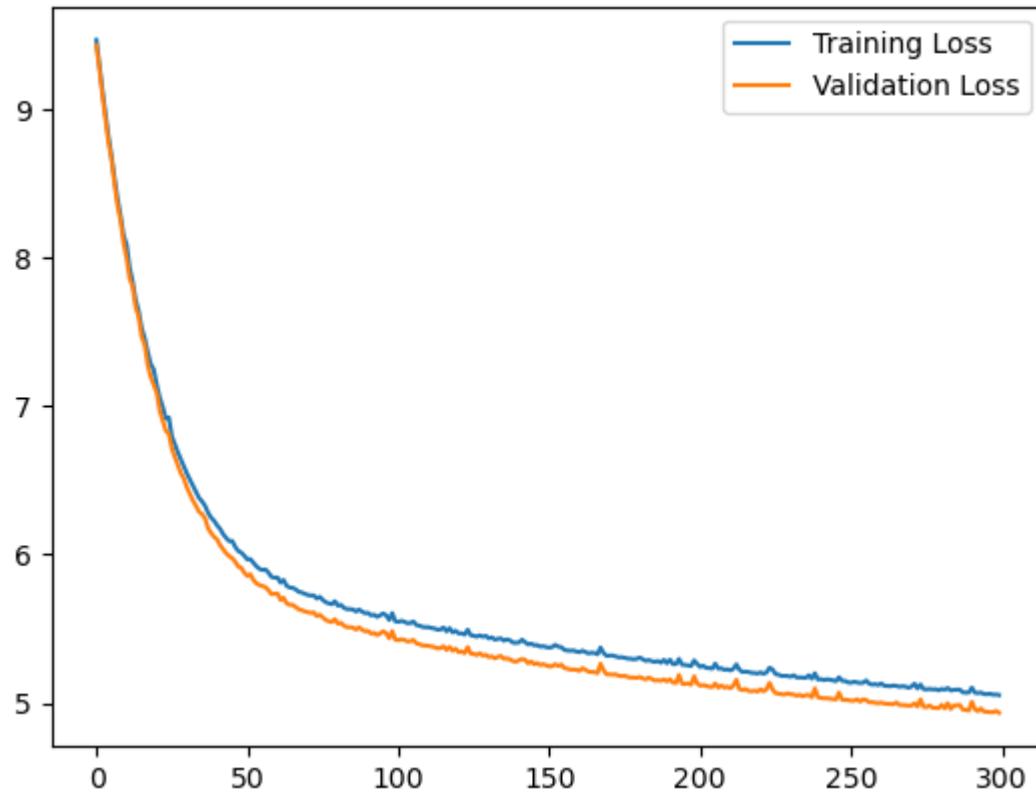


RMSE score on test data: 2.7161616109313794e+19

Polynomial degree =: 2

Penalty: 12

Learning Rate: 1e-06

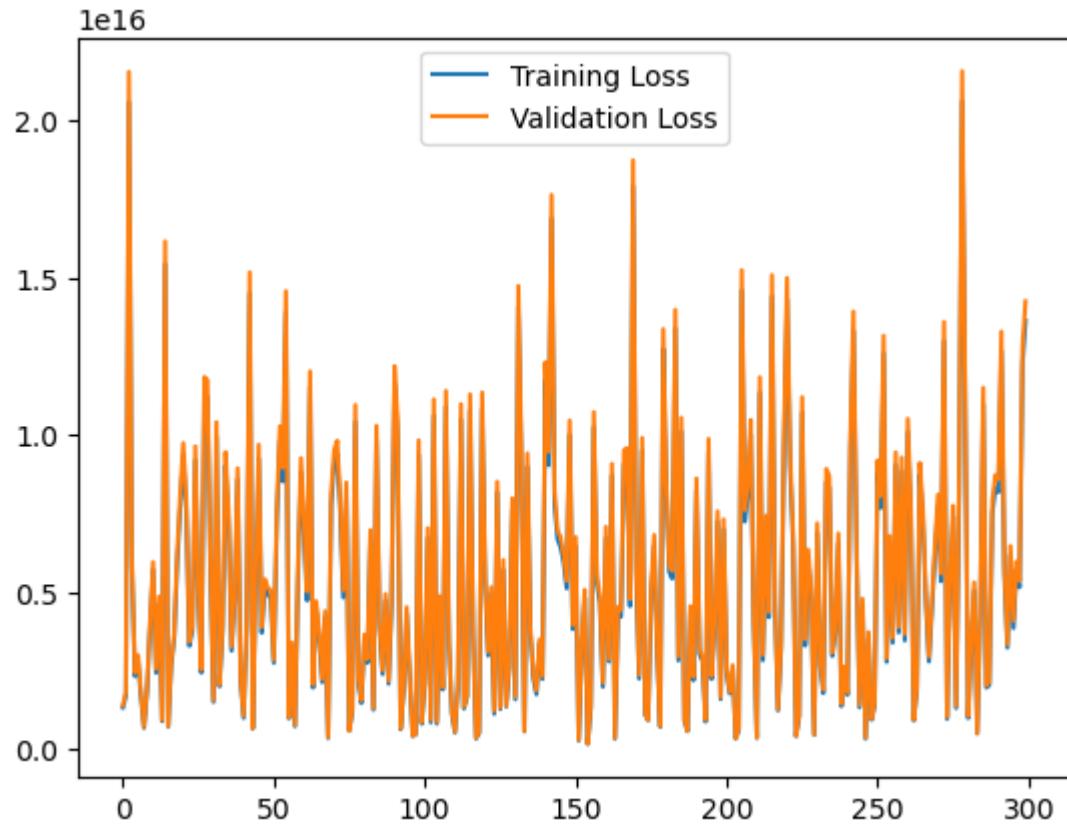


RMSE score on test data: 4.967026929673772

Polynomial degree =: 5

Penalty: 12

Learning Rate: 1e-06

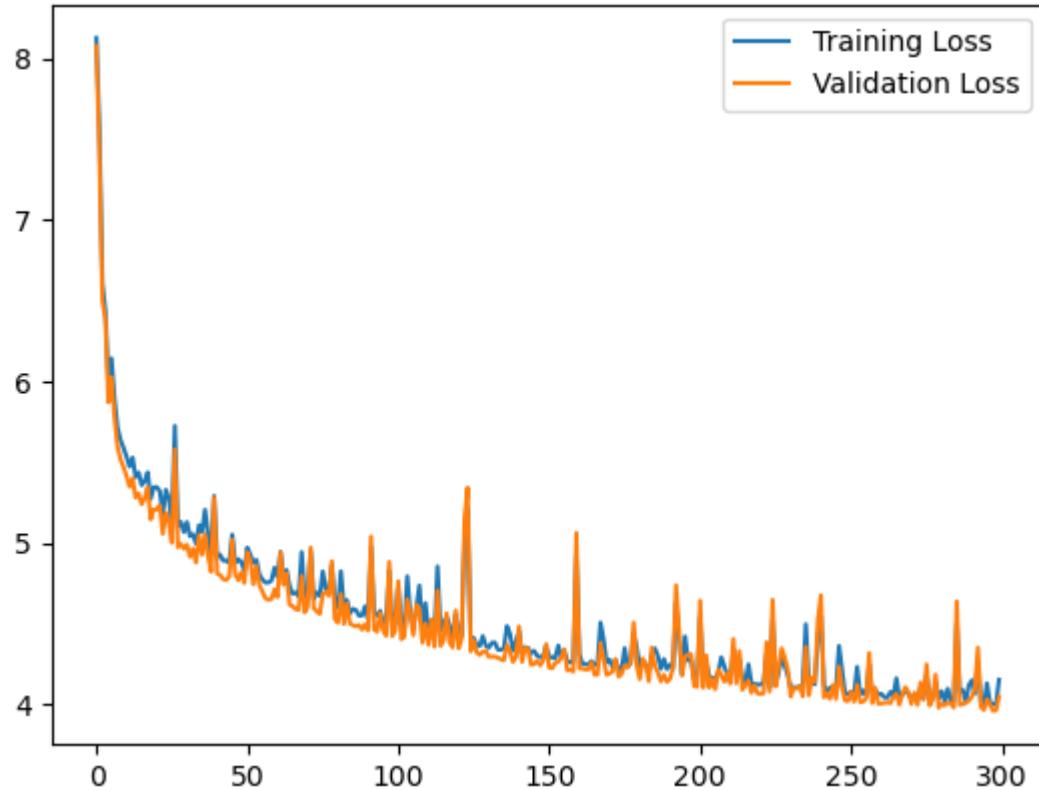


RMSE score on test data: $1.344293853433627e+16$

Polynomial degree =: 2

Penalty: 12

Learning Rate: $1e-05$

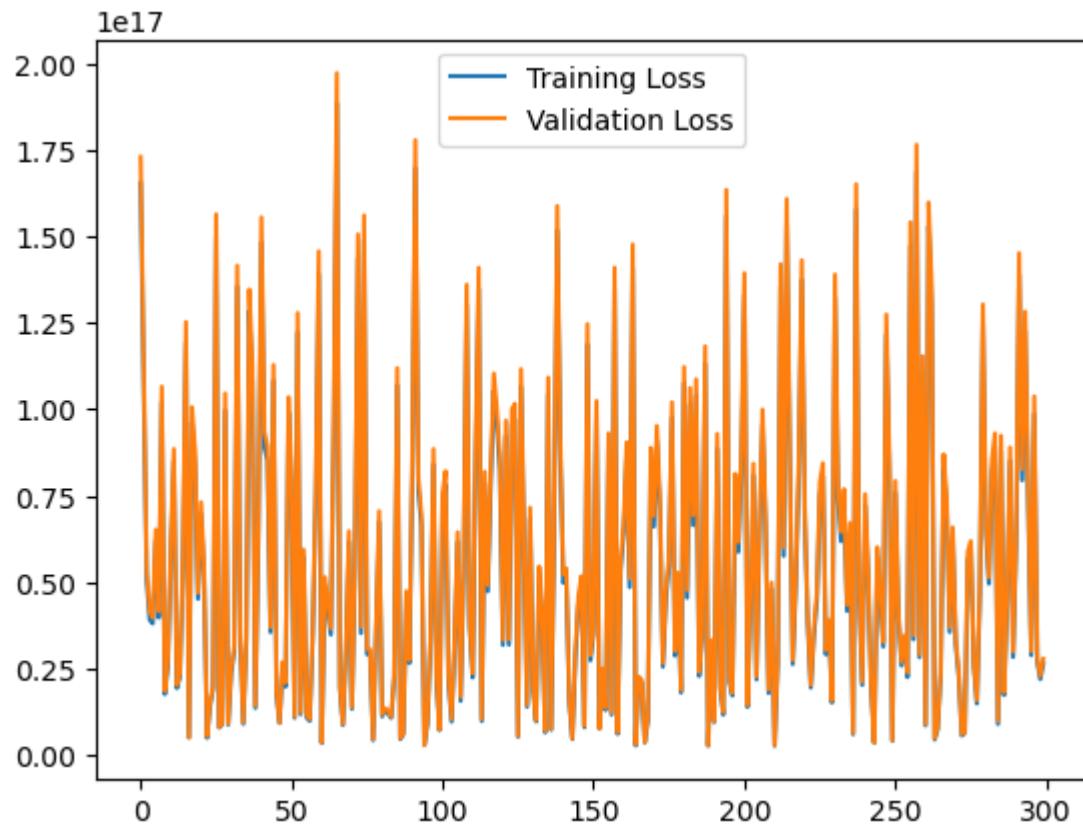


RMSE score on test data: 4.02519366708091

Polynomial degree =: 5

Penalty: 12

Learning Rate: 1e-05

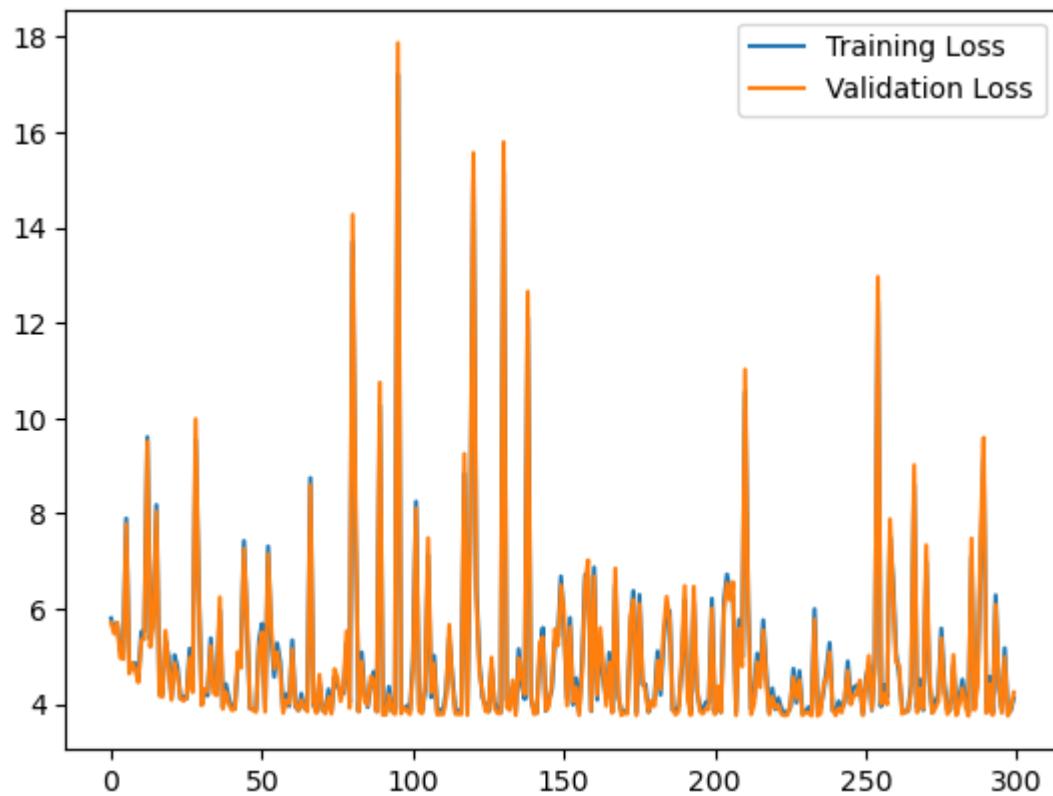


RMSE score on test data: 2.630551590131597e+16

Polynomial degree =: 2

Penalty: 12

Learning Rate: 0.0001

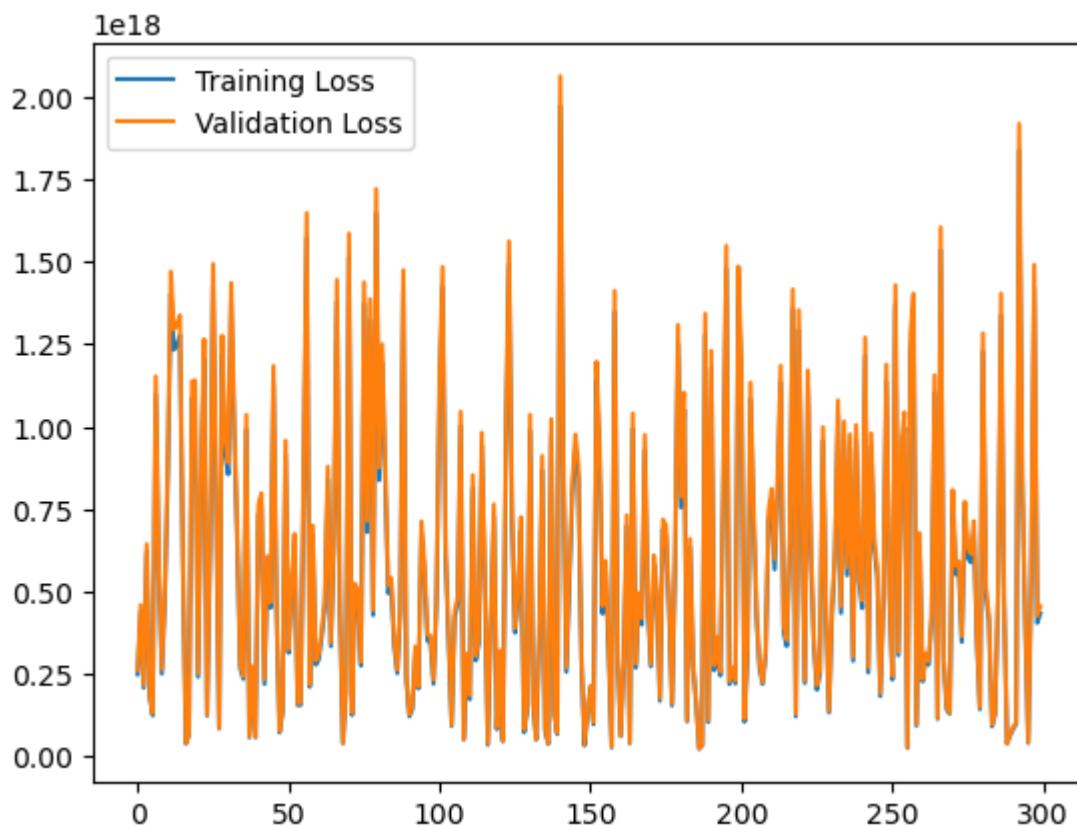


RMSE score on test data: 3.9898329295262225

Polynomial degree =: 5

Penalty: 12

Learning Rate: 0.0001

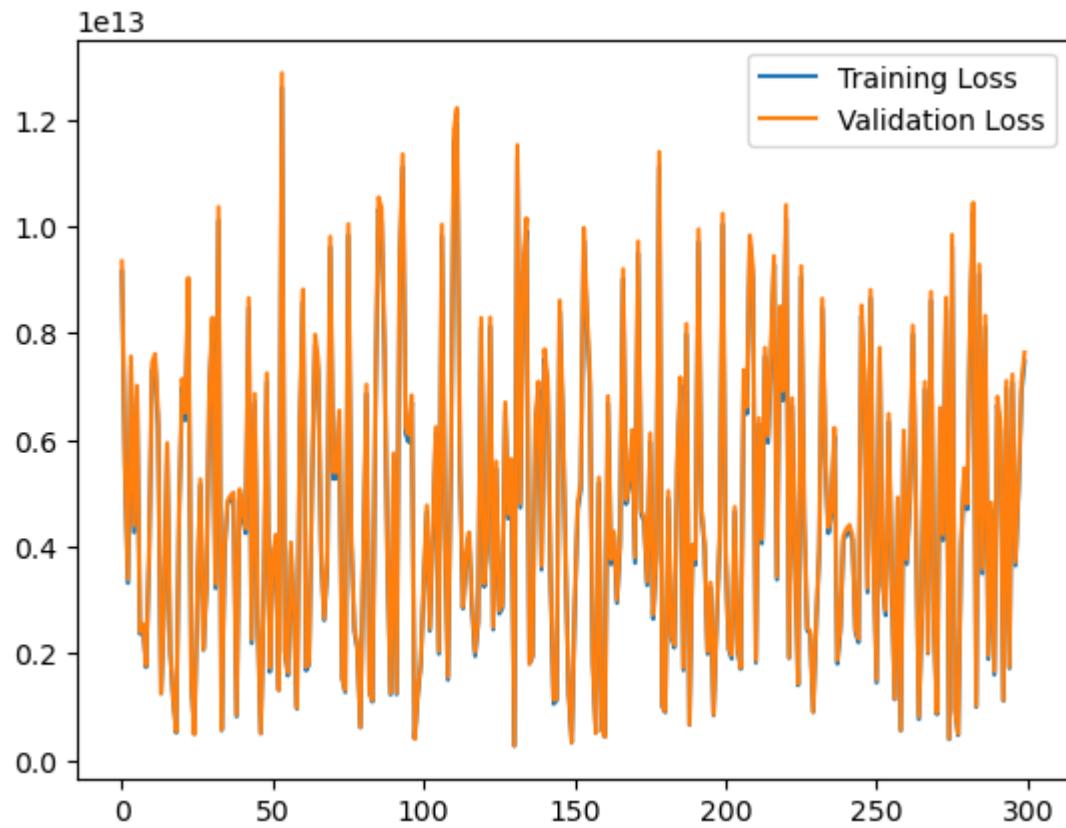


RMSE score on test data: 4.283540782282905e+17

Polynomial degree =: 2

Penalty: 12

Learning Rate: 0.001

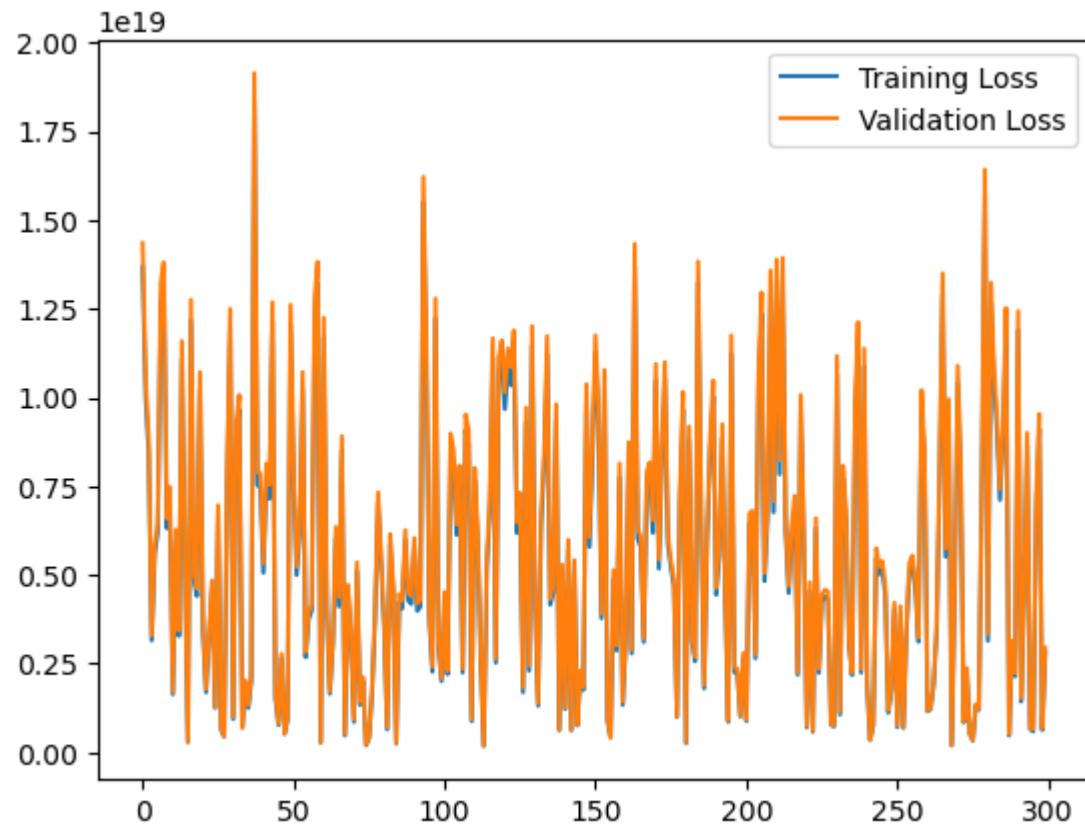


RMSE score on test data: 7430565722525.234

Polynomial degree =: 5

Penalty: 12

Learning Rate: 0.001

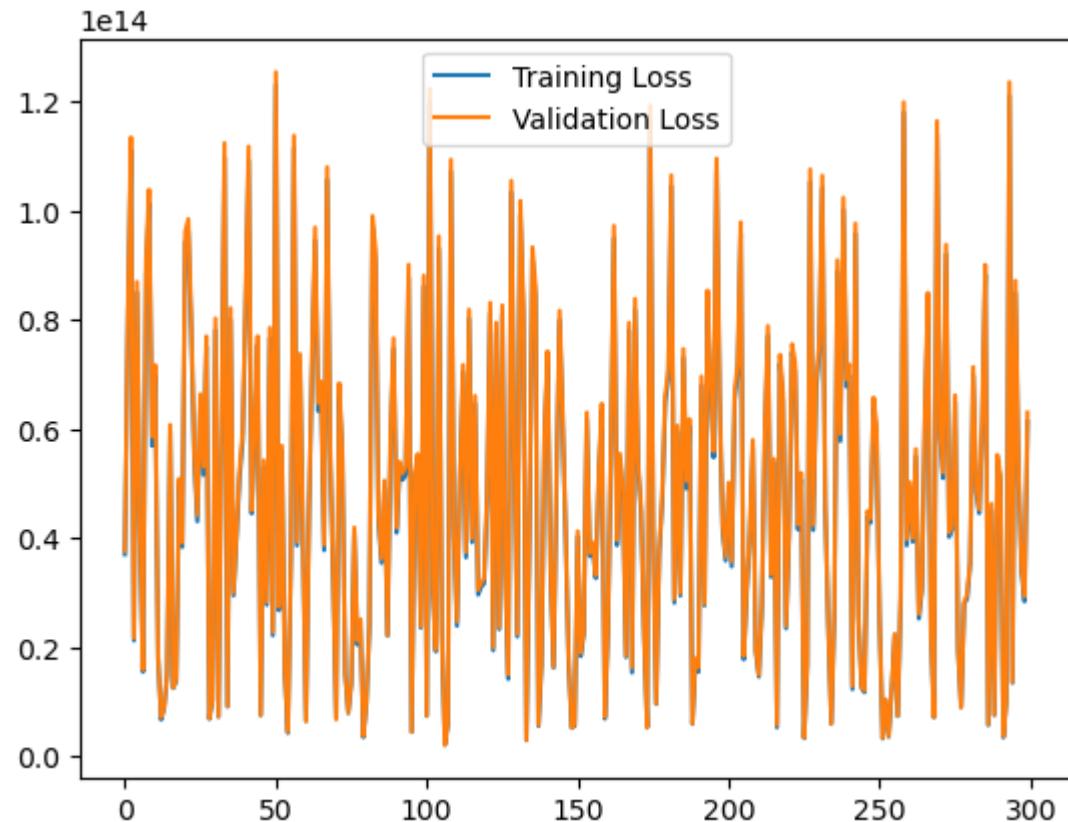


RMSE score on test data: $2.7507114710881167 \times 10^{18}$

Polynomial degree =: 2

Penalty: 12

Learning Rate: 0.01

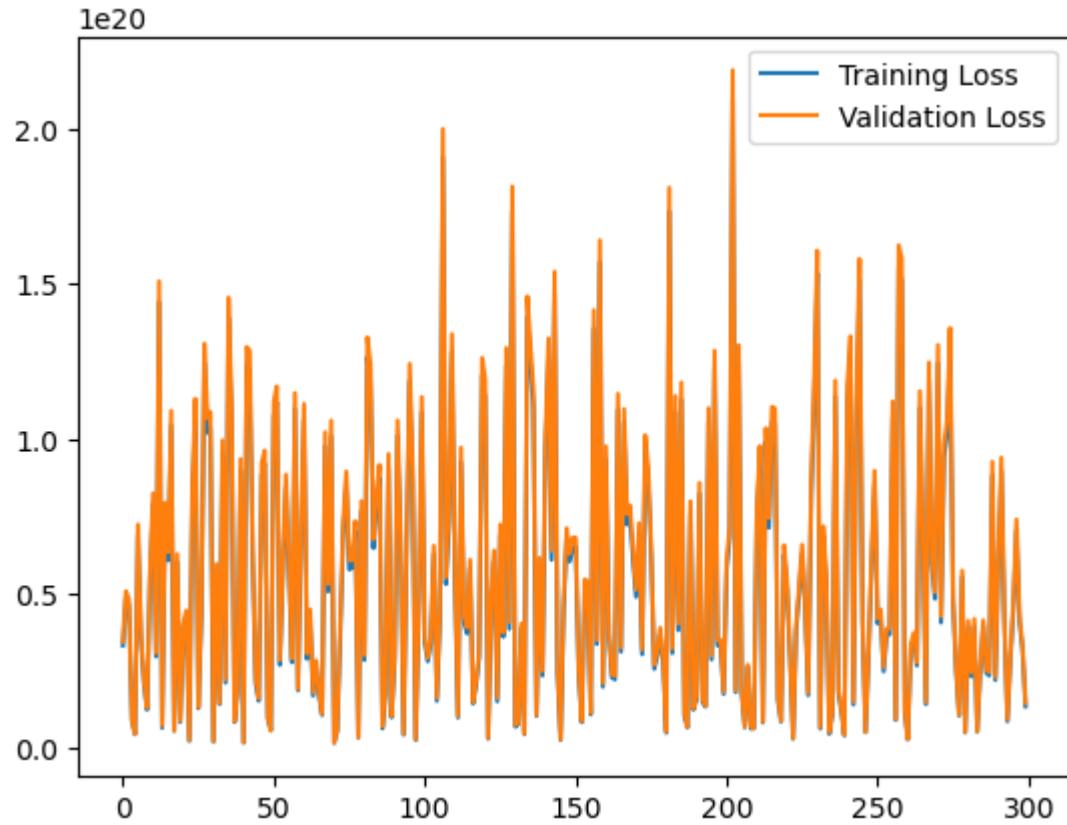


RMSE score on test data: 60868520316317.58

Polynomial degree =: 5

Penalty: 12

Learning Rate: 0.01



RMSE score on test data: 1.286544185438726e+19

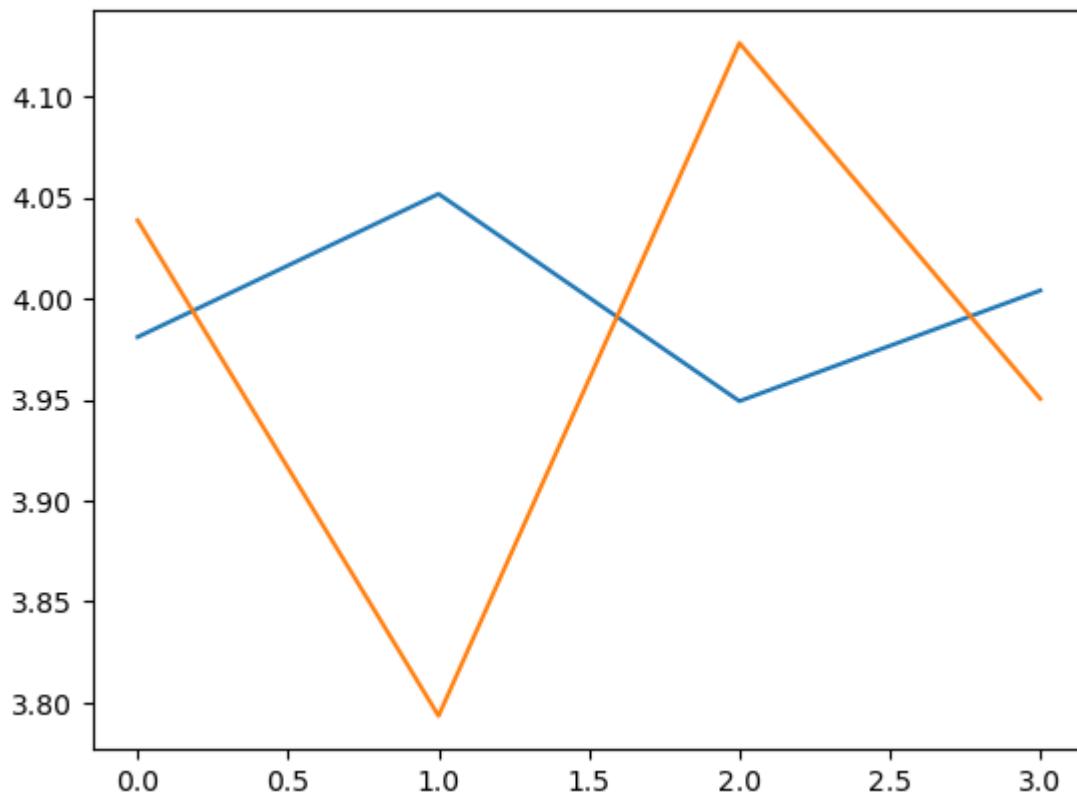
In [33]:

```

1 # Polynomial regression using elastic net
2
3 poly_degree = [2,5,10]
4
5 for r in poly_degree:
6     print("Degree:", r)
7     train_plot = []
8     val_plot = []
9     poly_features = PolynomialFeatures(degree=r, include_bias=False)
10    el_net = ElasticNet()
11    for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):
12        print(cnt)
13        x = x_train_2[var[0],:]
14        y = y_train_2[var[0]]
15        x_val = x_train_2[var[1],:]
16        y_val = y_train_2[var[1]]
17
18        X_poly_1 = poly_features.fit_transform(x)
19        el_net.fit(X_poly_1, y)
20        y_pred_train=el_net.predict(X_poly_1)
21
22        X_poly_2 = poly_features.fit_transform(x_val)
23        el_net.fit(X_poly_2, y_val)
24        y_pred_val=el_net.predict(X_poly_2)
25
26        print("Train error:",np.sqrt(mse(y_pred_train,y)))
27        print("Validation error:",np.sqrt(mse(y_pred_val,y_val)))
28        train_plot.append(np.sqrt(mse(y_pred_train,y)))
29        val_plot.append(np.sqrt(mse(y_pred_val,y_val)))
30
31        X_test_poly = poly_features.fit_transform(x_test_2)
32        y_pred_test = el_net.predict(X_test_poly)
33        metric_dict['Polynomial degree for elastic net= '+str(r)] = np.sqrt(mse(y_pred_test,y))
34    plt.plot([0,1,2,3], train_plot, label = "Training Error")
35    plt.plot([0,1,2,3], val_plot, label = "Val Error")
36    plt.show()
37    print('\n')

```

Degree: 2
0
Train error: 3.9809792086515667
Validation error: 4.038732445923404
1
Train error: 4.0518261122264905
Validation error: 3.793469635660633
2
Train error: 3.949243056023887
Validation error: 4.126506536817681
3
Train error: 4.003997486058499
Validation error: 3.9503997285724317



Degree: 5

0

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.673e+03, tolerance: 5.971e+00
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.213e+03, tolerance: 2.052e+00
    model = cd_fast.enet_coordinate_descent(
```

Train error: 3.5554164465245663

Validation error: 3.5015473359588265

1

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.990e+03, tolerance: 6.179e+00
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.851e+03, tolerance: 1.870e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.64850845473893
Validation error: 3.2528212536503998
2
```

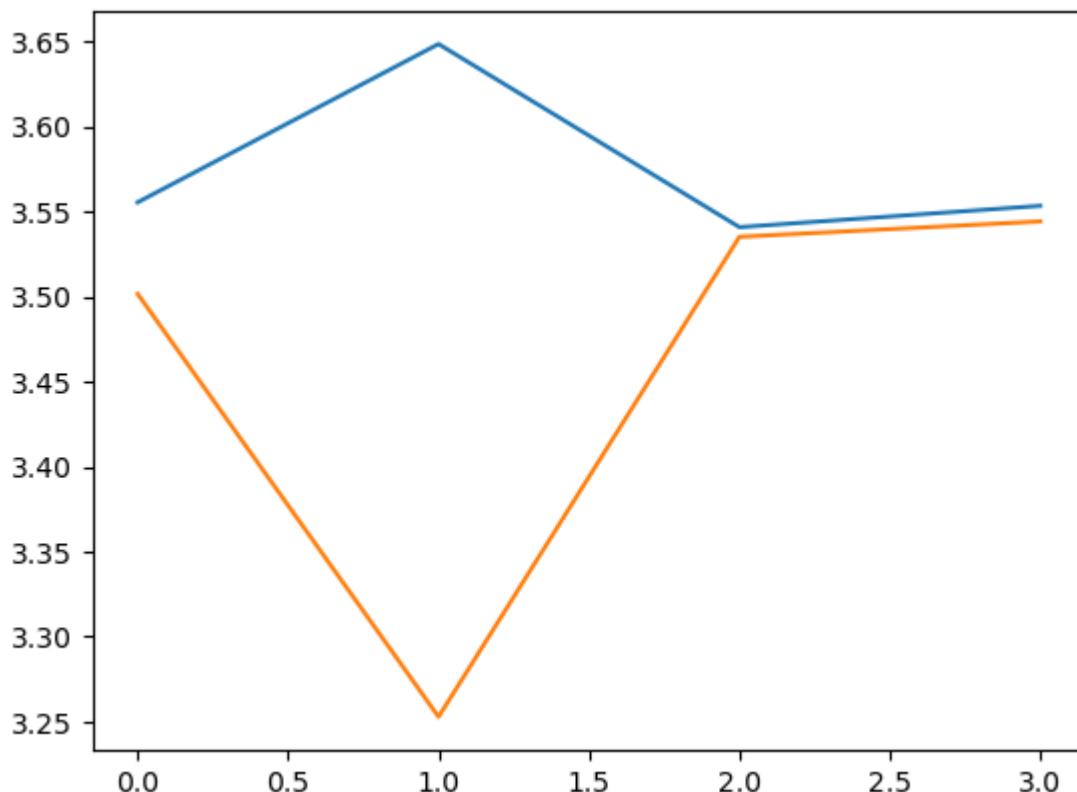
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.639e+03, tolerance: 5.880e+00
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.118e+03, tolerance: 2.163e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.5408035796202677
Validation error: 3.5351264253986843
3
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 6.552e+03, tolerance: 6.105e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.5533565411661088
Validation error: 3.5441836067983408
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 2.243e+03, tolerance: 1.902e+00
    model = cd_fast.enet_coordinate_descent(
```



```
Degree: 10
```

```
0
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.136e+03, tolerance: 5.971e+00
    model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.436e+03, tolerance: 2.052e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.062281414024047
```

```
Validation error: 2.727345524226253
```

```
1
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.464e+03, tolerance: 6.179e+00
    model = cd_fast.enet_coordinate_descent(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.232e+03, tolerance: 1.870e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.1646464802205694
```

```
Validation error: 2.5195633553373007
```

```
2
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.142e+03, tolerance: 5.880e+00
    model = cd_fast.enet_coordinate_descent(
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.406e+03, tolerance: 2.163e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.061226395047278
```

```
Validation error: 2.6725597341931024
```

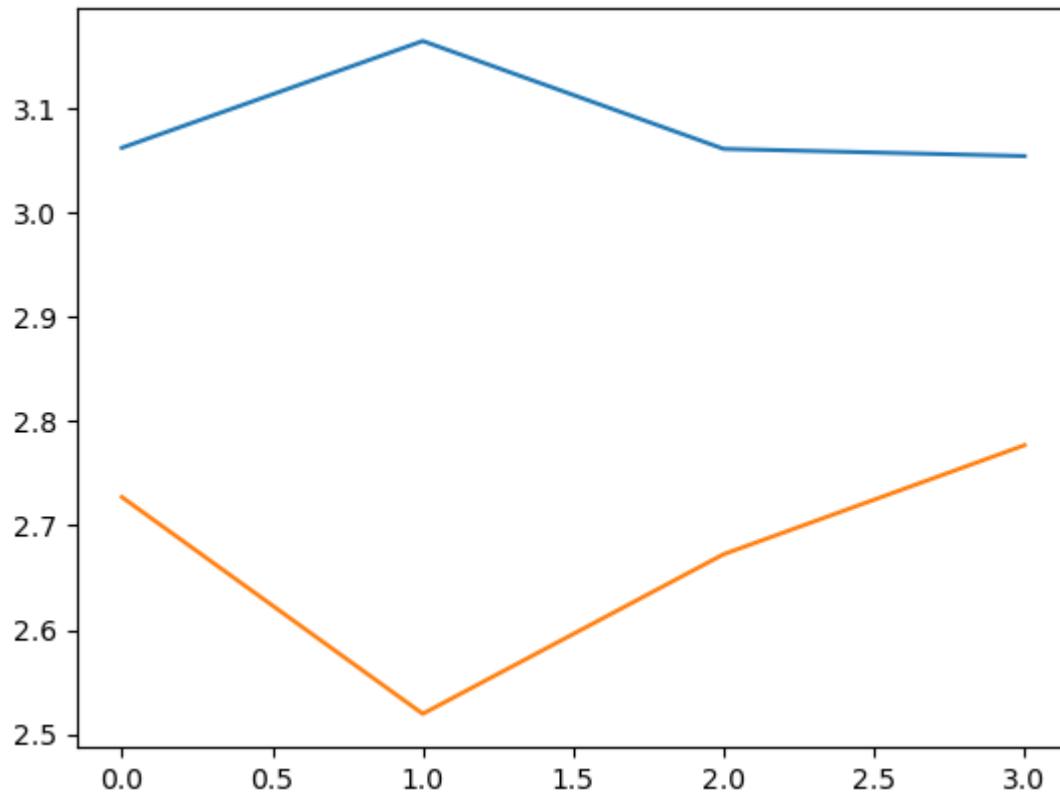
```
3
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 5.103e+03, tolerance: 6.105e+00
    model = cd_fast.enet_coordinate_descent(
```

```
Train error: 3.0543335150222743
```

```
Validation error: 2.7771198215282
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.py:647: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations, check the scale of the features or consider increasing regularisation. Duality gap: 1.528e+03, tolerance: 1.902e+00
    model = cd_fast.enet_coordinate_descent()
```



Answer:- The results show that as the degree of the polynomial increases, the training error decreases, but the validation error may start to increase, indicating overfitting.

In the case of polynomial regression using ridge and lasso, regularization has been applied to prevent overfitting. The results show that both ridge and lasso help to reduce the validation error compared to polynomial regression without regularization.

It is worth noting that for some of the results, the validation error is much larger than the train error, which could indicate that the model is overfitting the training data. In such cases, it may be necessary to further tune the hyperparameters of the model or collect more data to improve the performance.



Part 7

Make predictions of the labels on the test data, using the trained model with chosen hyperparameters. Summarize performance using the appropriate evaluation metric. Discuss the results. Include thoughts about what further can be explored to increase performance.

```
In [34]: 1 # Making predictions on the test dataset. We have used all of our previous  
2 # calculated the RMSE scores.  
3  
4 ridge = Ridge()  
5 for cnt,var in enumerate(kf.split(x_train_2,y_train_2)):  
6     x = x_train_2[var[0],:]  
7     y = y_train_2[var[0]]  
8     x_val = x_train_2[var[1],:]  
9     y_val = y_train_2[var[1]]  
10    ridge.fit(x, y)  
11    y_pred_train = ridge.predict(x)  
12    ridge.fit(x_val, y_val)  
13    y_pred_val = ridge.predict(x_val)
```

```
In [35]: 1 y_pred_test = ridge.predict(x_test_2)  
2 print("Error:",np.sqrt(mse(y_pred_test,y_test_2)))
```

Error: 3.6959852293570363

In [36]:

```
1 # Now we will look at the RMSE scores for all the models on our test data
2
3 metric_dict
```

```
Out[36]: {'Linear regression': 3.693685608175782,
'SGD with learning rate =1e-06': 3.96966749079844,
'SGD with learning rate =1e-05': 3.7670963034078917,
'SGD with learning rate =0.0001': 3.689297182286318,
'SGD with learning rate =0.001': 3.9322983646063134,
'SGD with learning rate =0.01': 4.374503043689119,
'SGD with learning rate =1e-06 and penalty_factor =None': 3.969665182012894
5,
'SGD with learning rate =1e-05 and penalty_factor =None': 3.767312827919267,
'SGD with learning rate =0.0001 and penalty_factor =None': 3.69938914887747
7,
'SGD with learning rate =0.001 and penalty_factor =None': 3.783067499048873
3,
'SGD with learning rate =0.01 and penalty_factor =None': 3.836960346031612,
'SGD with learning rate =1e-06 and penalty_factor =11': 3.9696603920934166,
'SGD with learning rate =1e-05 and penalty_factor =11': 3.768378646001259,
'SGD with learning rate =0.0001 and penalty_factor =11': 3.7035761649960635,
'SGD with learning rate =0.001 and penalty_factor =11': 3.7274527732688942,
'SGD with learning rate =0.01 and penalty_factor =11': 3.727742795406936,
'SGD with learning rate =1e-06 and penalty_factor =12': 3.96968115196651,
'SGD with learning rate =1e-05 and penalty_factor =12': 3.768136952869328,
'SGD with learning rate =0.0001 and penalty_factor =12': 3.689837070793864,
'SGD with learning rate =0.001 and penalty_factor =12': 3.797161489890803,
'SGD with learning rate =0.01 and penalty_factor =12': 3.809220893603164,
'Ridge lr=0': 3.704811795189289,
'Ridge lr=0.001': 3.70477088748351,
'Ridge lr=1': 3.6959852293570363,
'Ridge lr=10': 3.752850778603473,
'Ridge lr=100': 4.027764736515409,
'Lasso lr=0': 3.7048117951892854,
'Lasso lr=0.001': 3.7025921660801306,
'Lasso lr=1': 3.9493545094080553,
'Lasso lr=10': 7.815572369551083,
'Lasso lr=100': 7.815572369551083,
'ElasticNet lr=0': 3.7048117951892854,
'ElasticNet lr=0.001': 3.698810421197111,
'ElasticNet lr=1': 4.417471685851984,
'ElasticNet lr=10': 7.446647805600527,
'ElasticNet lr=100': 7.815572369551083,
'Polynomial degree= 2': 3.859292154675337,
'Polynomial degree= 5': 47.12077004638479,
'Polynomial degree= 10': 44.19804017615544,
'Polynomial degree for ridge= 2': 3.622974516903029,
'Polynomial degree for ridge= 5': 4.11366726420557,
'Polynomial degree for ridge= 10': 9.058475712119527,
'Polynomial degree for Lasso = 2': 3.8232737973216584,
'Polynomial degree for Lasso = 5': 3.5183674748612876,
'Polynomial degree for Lasso = 10': 3.889818627513967,
'Polynomial degree for SGD= 2 Learning_rate= 1e-06 and penalty_term= None':
5.002836638073316,
'Polynomial degree for SGD= 5 Learning_rate= 1e-06 and penalty_term= None':
770328126591323.4,
'Polynomial degree for SGD= 2 Learning_rate= 1e-05 and penalty_term= None':
3.9412323193397887,
'Polynomial degree for SGD= 5 Learning_rate= 1e-05 and penalty_term= None':
3.1996231306365056e+16,
'Polynomial degree for SGD= 2 Learning_rate= 0.0001 and penalty_term= None':
```

```
3.8693893503500916,
'Polynomial degree for SGD= 5 Learning_rate= 0.0001 and penalty_term= None': 9.397934997545254e+17,
'Polynomial degree for SGD= 2 Learning_rate= 0.001 and penalty_term= None': 1411997102619.439,
'Polynomial degree for SGD= 5 Learning_rate= 0.001 and penalty_term= None': 4.524759715010909e+18,
'Polynomial degree for SGD= 2 Learning_rate= 0.01 and penalty_term= None': 6 0805219622965.67,
'Polynomial degree for SGD= 5 Learning_rate= 0.01 and penalty_term= None': 1.0193444668765428e+19,
'Polynomial degree for SGD= 2 Learning_rate= 1e-06 and penalty_term= 11': 4. 966885500381611,
'Polynomial degree for SGD= 5 Learning_rate= 1e-06 and penalty_term= 11': 1. 3761579679256478e+16,
'Polynomial degree for SGD= 2 Learning_rate= 1e-05 and penalty_term= 11': 3. 8999474542505728,
'Polynomial degree for SGD= 5 Learning_rate= 1e-05 and penalty_term= 11': 1. 2971620554358054e+16,
'Polynomial degree for SGD= 2 Learning_rate= 0.0001 and penalty_term= 11': 3. 770076874012981,
'Polynomial degree for SGD= 5 Learning_rate= 0.0001 and penalty_term= 11': 1.2534510663731338e+17,
'Polynomial degree for SGD= 2 Learning_rate= 0.001 and penalty_term= 11': 49 3769917474.58276,
'Polynomial degree for SGD= 5 Learning_rate= 0.001 and penalty_term= 11': 3. 0020947762949115e+18,
'Polynomial degree for SGD= 2 Learning_rate= 0.01 and penalty_term= 11': 489 63725981110.8,
'Polynomial degree for SGD= 5 Learning_rate= 0.01 and penalty_term= 11': 2.7 161616109313794e+19,
'Polynomial degree for SGD= 2 Learning_rate= 1e-06 and penalty_term= 12': 4. 967026929673772,
'Polynomial degree for SGD= 5 Learning_rate= 1e-06 and penalty_term= 12': 1. 344293853433627e+16,
'Polynomial degree for SGD= 2 Learning_rate= 1e-05 and penalty_term= 12': 4. 02519366708091,
'Polynomial degree for SGD= 5 Learning_rate= 1e-05 and penalty_term= 12': 2. 630551590131597e+16,
'Polynomial degree for SGD= 2 Learning_rate= 0.0001 and penalty_term= 12': 3. 9898329295262225,
'Polynomial degree for SGD= 5 Learning_rate= 0.0001 and penalty_term= 12': 4.283540782282905e+17,
'Polynomial degree for SGD= 2 Learning_rate= 0.001 and penalty_term= 12': 74 30565722525.234,
'Polynomial degree for SGD= 5 Learning_rate= 0.001 and penalty_term= 12': 2. 7507114710881167e+18,
'Polynomial degree for SGD= 2 Learning_rate= 0.01 and penalty_term= 12': 608 68520316317.58,
'Polynomial degree for SGD= 5 Learning_rate= 0.01 and penalty_term= 12': 1.2 86544185438726e+19,
'Polynomial degree for elastic net= 2': 3.818430473841465,
'Polynomial degree for elastic net= 5': 3.5010190728760713,
'Polynomial degree for elastic net= 10': 4.004234525143996}
```

Answer: Polynomial degree for Elastic net= 5 works best on our test data, since they have the lowest RMSE scores.

We can try and add more data points so that we have a more robust understanding of the dataset. More amount of data always leads to better training and better results. We can preprocess the data further and try removing skewness or imputing null values. We can do in-depth EDA to find which features affect the target variable more and just feed those features into our model. These are some ways by which we can improve the efficiency of our models.

In []:

1