Predicting Healthy Life Expectancy

The goal of this problem is to explore how to properly analyze, visualize, split, clean and format data and perform linear regression, polynomial regression and regularization.

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
import sys

if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

Uploading the file from device

In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving happiness_data.csv to happiness_data.csv
```

In [ ]:
```
import io

df = pd.read_csv(io.BytesIO(uploaded['happiness_data.csv']))
df
```

Out[ ]:

	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
...	...	...	...	...	...	...	...	...	...
1944	Zimbabwe	2016	3.735	7.984	0.768	54.40	0.733	-0.095	0.724
1945	Zimbabwe	2017	3.638	8.016	0.754	55.00	0.753	-0.098	0.751
1946	Zimbabwe	2018	3.616	8.049	0.775	55.60	0.763	-0.068	0.844
1947	Zimbabwe	2019	2.694	7.950	0.759	56.20	0.632	-0.064	0.831
1948	Zimbabwe	2020	3.160	7.829	0.717	56.80	0.643	-0.009	0.789

1949 rows × 11 columns

How much data is present? There are 1949 data points with 11 attributes

What attributes/features are continuous valued?

continued values 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 Which attributes are categorical?

contry name

code related to this is below

Printing the first 5 columns and getting the number of rows and columns

In [ ]:
```
display(df.head())
print(df.shape)
```

	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	Po
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	

(1949, 11)

Dropping columns year and life ladder

In [ ]:
```
df = df.drop(['year', 'Life Ladder'], axis=1)
df.head()
```

Out[ ]:

	Country name	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negativ affec
0	Afghanistan	7.370	0.451	50.80	0.718	0.168	0.882	0.518	0.25
1	Afghanistan	7.540	0.552	51.20	0.679	0.190	0.850	0.584	0.23
2	Afghanistan	7.647	0.539	51.60	0.600	0.121	0.707	0.618	0.27
3	Afghanistan	7.620	0.521	51.92	0.496	0.162	0.731	0.611	0.26
4	Afghanistan	7.705	0.521	52.24	0.531	0.236	0.776	0.710	0.26

Printing the descriptive statistics for each numeric column

In [ ]:
```
describe = (df.describe().T)
describe
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Log GDP per capita	1913.0	9.368453	1.154084	6.635	8.46400	9.4600	10.353	11.648
Social support	1936.0	0.812552	0.118482	0.290	0.74975	0.8355	0.905	0.987
Healthy life expectancy at birth	1894.0	63.359374	7.510245	32.300	58.68500	65.2000	68.590	77.100
Freedom to make life choices	1917.0	0.742558	0.142093	0.258	0.64700	0.7630	0.856	0.985
Generosity	1860.0	0.000103	0.162215	-0.335	-0.11300	-0.0255	0.091	0.698
Perceptions of corruption	1839.0	0.747125	0.186789	0.035	0.69000	0.8020	0.872	0.983
Positive affect	1927.0	0.710003	0.107100	0.322	0.62550	0.7220	0.799	0.944
Negative affect	1933.0	0.268544	0.085168	0.083	0.20600	0.2580	0.320	0.705

From the above statistics, we can see the mean, standard deviation, minimum and maximum values, and percentiles for each numeric data attribute.

Printing a summary of the dataframe

In [ ]:
```
(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1949 entries, 0 to 1948
Data columns (total 9 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Country name                      1949 non-null   object 
 1   Log GDP per capita                1913 non-null   float64
 2   Social support                    1936 non-null   float64
 3   Healthy life expectancy at birth  1894 non-null   float64
 4   Freedom to make life choices      1917 non-null   float64
 5   Generosity                        1860 non-null   float64
 6   Perceptions of corruption         1839 non-null   float64
 7   Positive affect                   1927 non-null   float64
 8   Negative affect                   1933 non-null   float64
dtypes: float64(8), object(1)
memory usage: 137.2+ KB
```

From the above information, we can see the data type of each column, the number of non-null values, and memory usage.

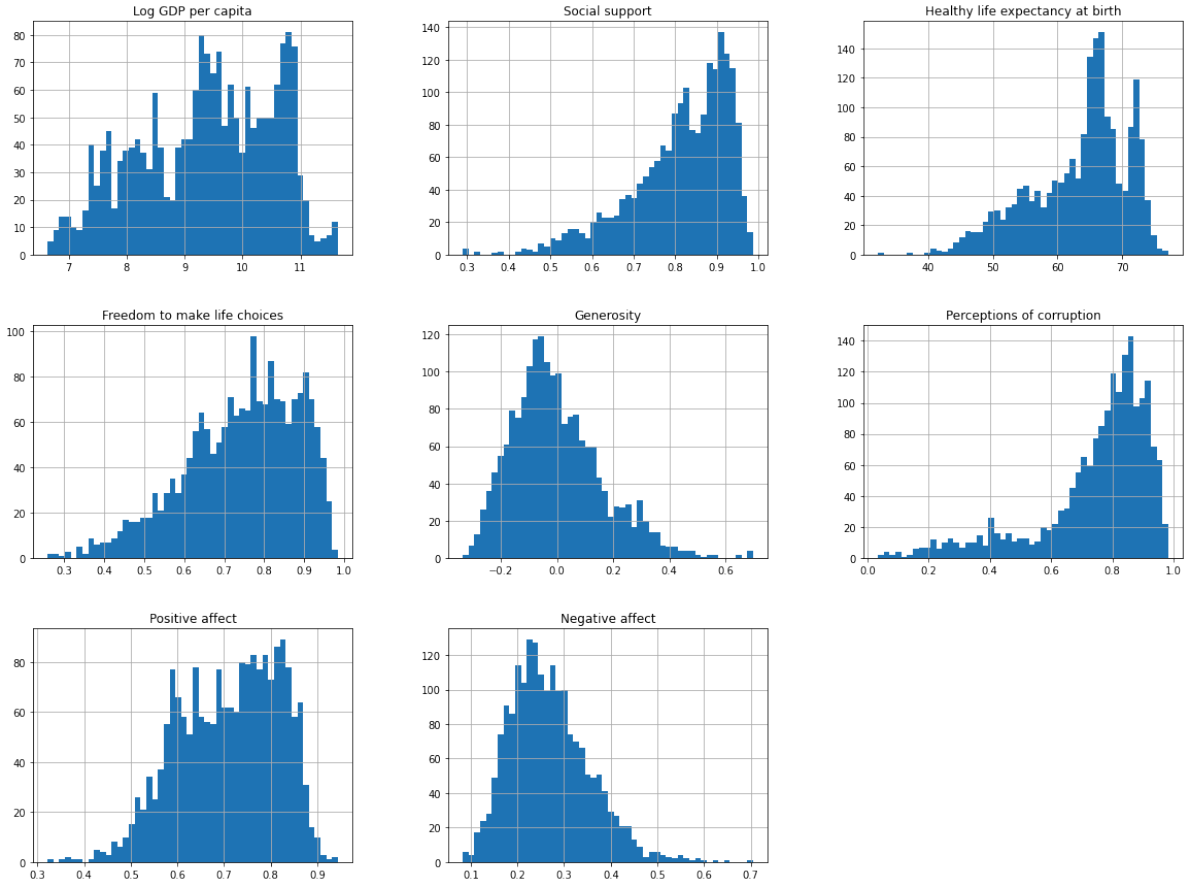
Printing the histograms to see the distribution of each data attribute

Display the statistical values for each of the attributes, along with visualizations (e.g., histogram)
below is the statistical analysis of all of the attributes. Its histogram is ploted to uderstand the it distribution and check if the data is skewed or not.

Explain noticeable traits for key attributes. Generosity attribute is fairly normal, Freedom to make chioces and perceptions of corruptions seem to be highly left sided skewed and hence makes the data skwed.

Are there any attributes that might require special treatment? If so, what special treatment might they require? Some attributes like year are dropped as they do not provide any special value. Atribute country needs to be taken into consideration but doing one hot encoding as that column has good weight on the dataset. Here below you an see all of them are also scaled into standaedscaler form for the betterment of learning of the model.

In [ ]:
```
df.hist(bins=50, figsize=(20,15))
plt.show()
```



Calculating the interquartile range for each numeric column

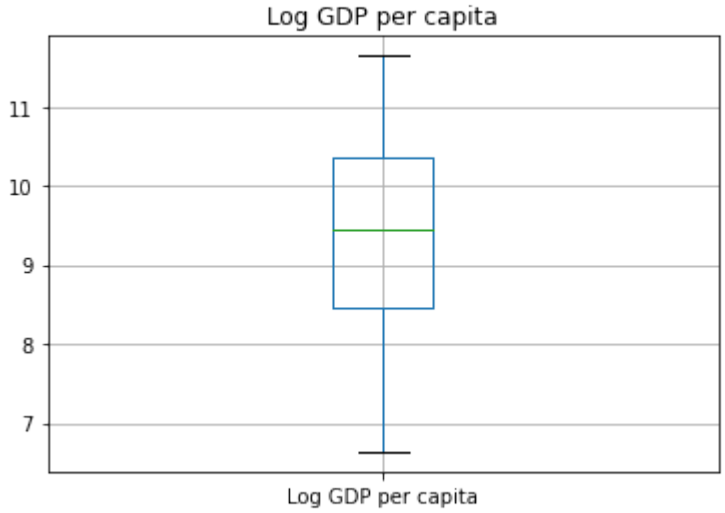
In [ ]:
```
describe = df.describe(percentiles=[.25, .75])
IQR = describe.loc['75%'] - describe.loc['25%']
IQR
```

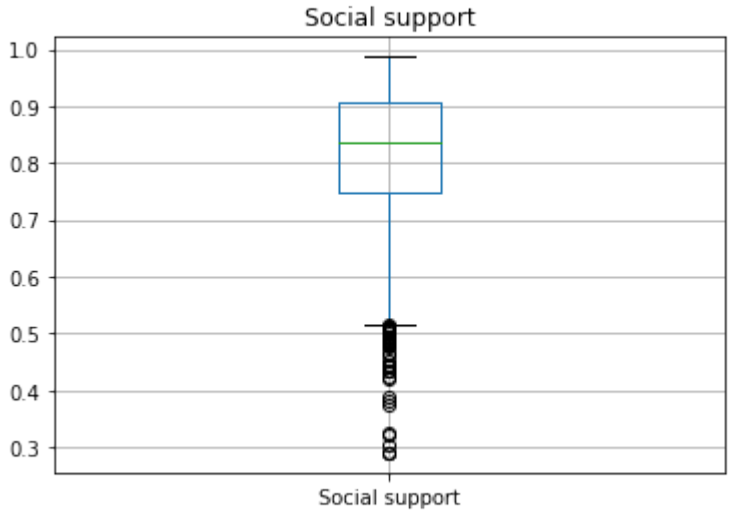
Out[ ]:
```
Log GDP per capita                  1.88900
Social support                      0.15525
Healthy life expectancy at birth    9.90500
Freedom to make life choices        0.20900
Generosity                          0.20400
Perceptions of corruption           0.18200
Positive affect                     0.17350
Negative affect                     0.11400
dtype: float64
```

Box plots for each column

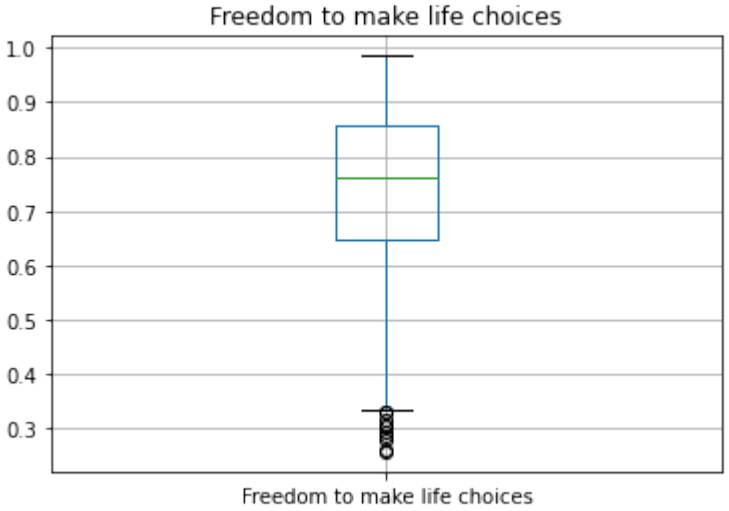
In [ ]:

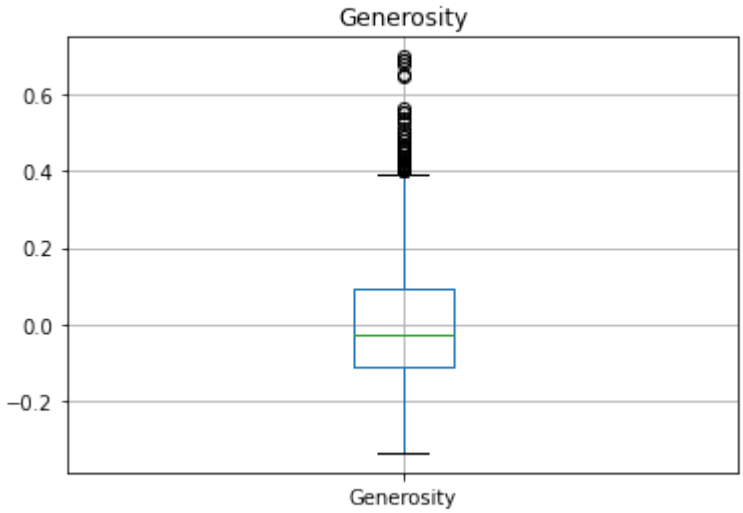
```
for column in df.columns:
    if df[column].dtype in ['float64', 'int64']:
        plt.figure()
        df.boxplot([column])
        plt.title(column)
        plt.show()
```

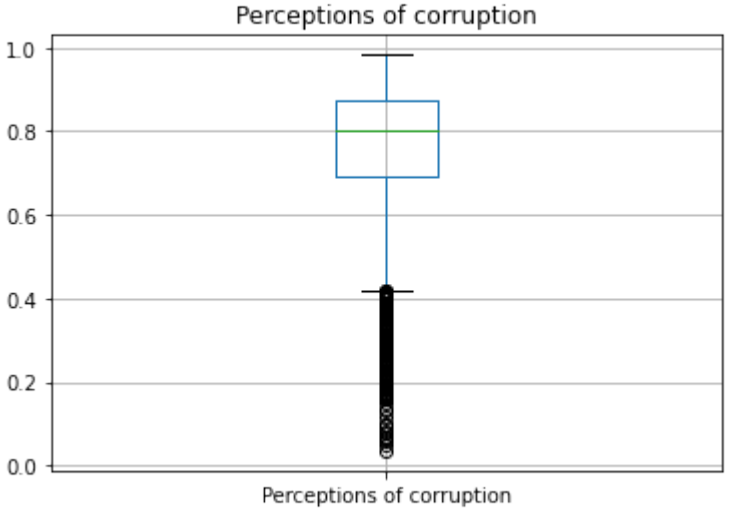

Log GDP per capita
11
10
9
8
7
Log GDP per capita

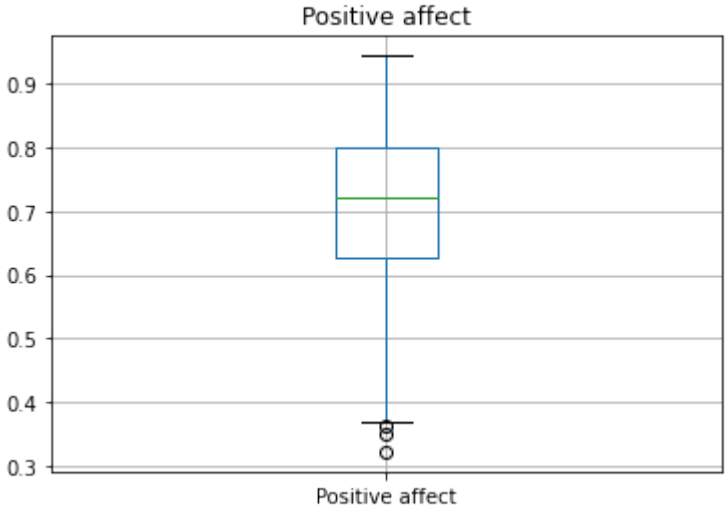

Social support
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
Social support

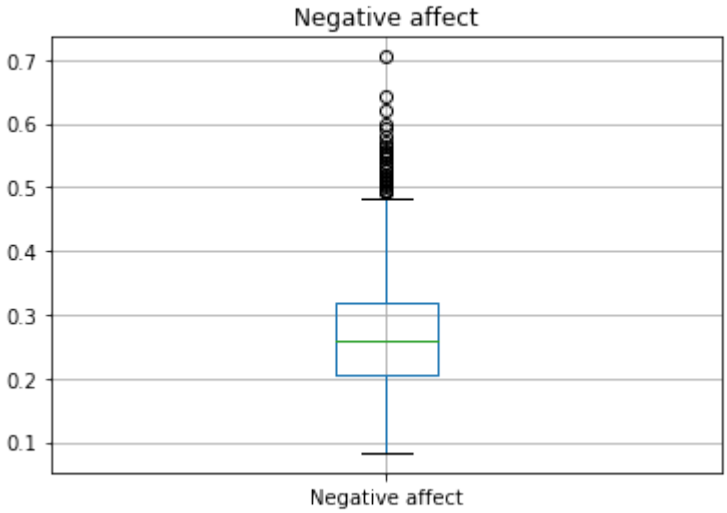

Healthy life expectancy at birth
70
60
50
40
Healthy life expectancy at birth


Freedom to make life choices
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
Freedom to make life choices


Generosity
0.6
0.4
0.2
0.0
−0.2
Generosity


Perceptions of corruption
1.0
0.8
0.6
0.4
0.2
0.0
Perceptions of corruption

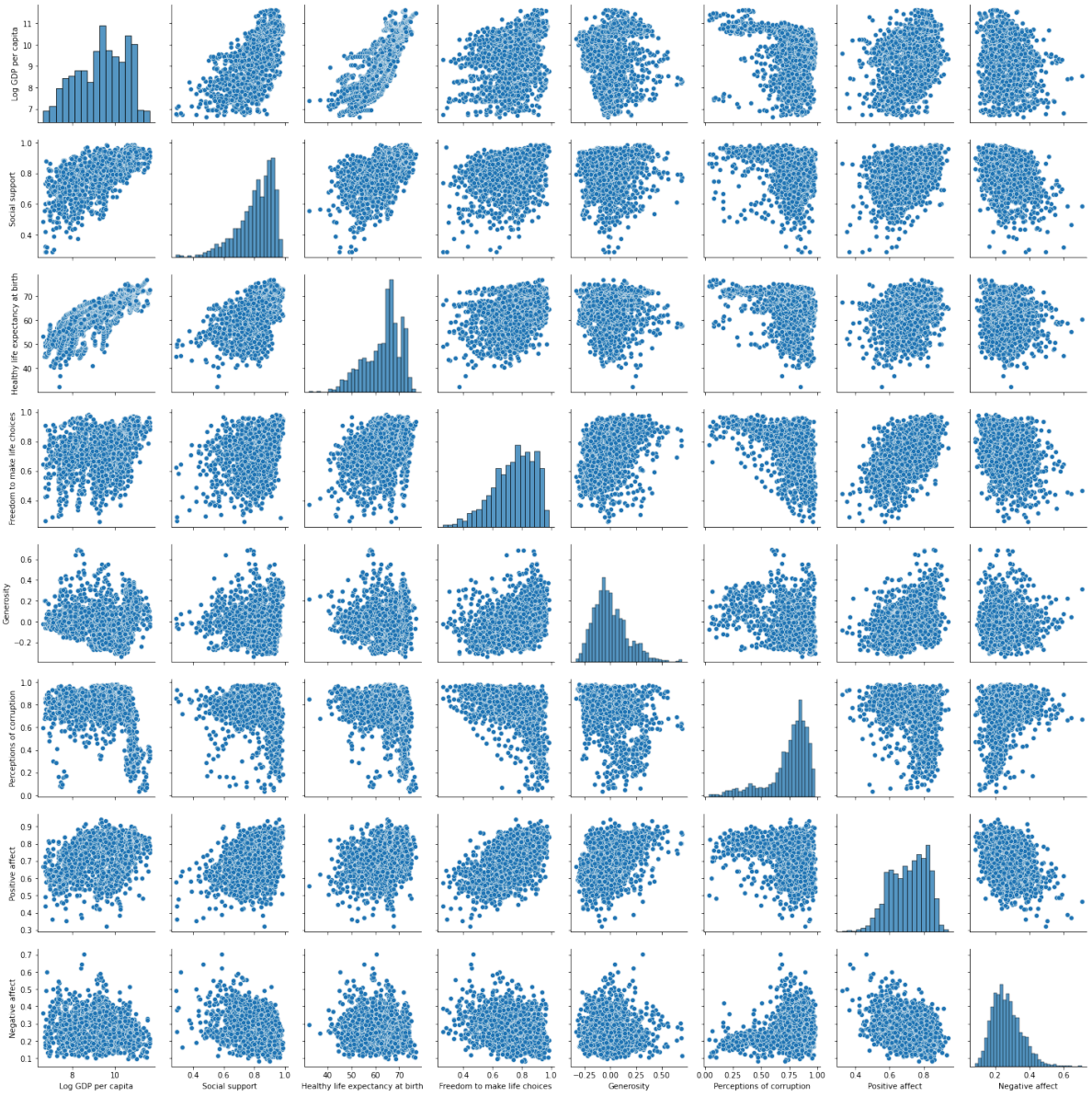




Using pairplot to visualize the relationship of data attributes with one another.

```
In [ ]: sns.pairplot(df)
```

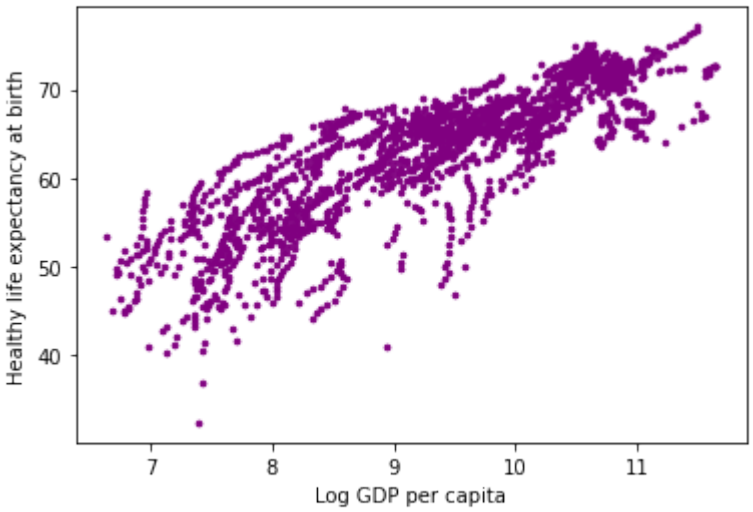
```
Out[ ]: <seaborn.axisgrid.PairGrid at 0x7f2007e5e130>
```

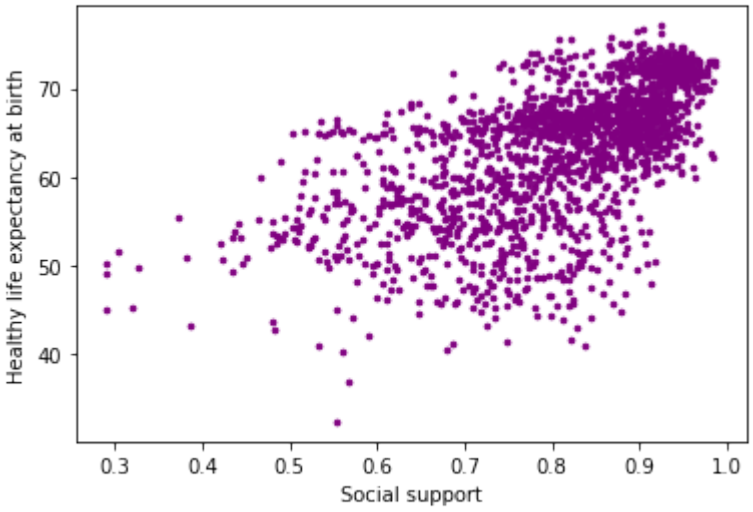


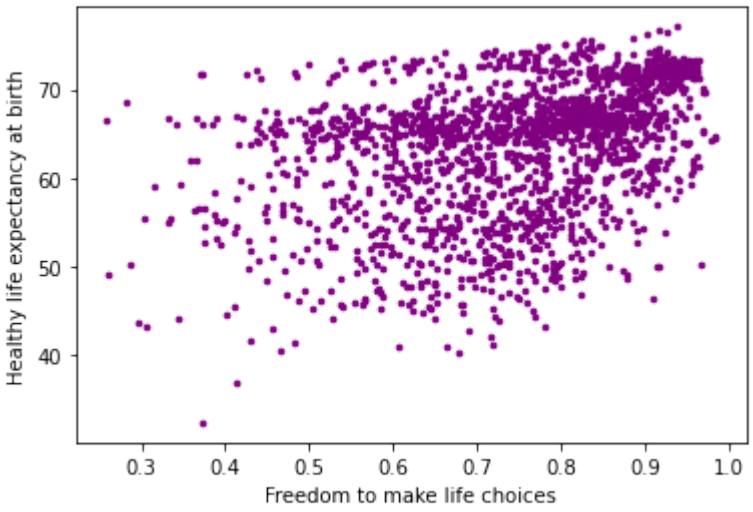
From the above visualizations we can say that, Log GDP per capita has a positive correlation with Social Support, Healthy Life Expectancy, Freedom to make life choices and Positive affect. As GDP is increasing, these variables are also increasing. However, as GDP increases, Generosity, Perception of corruption and Negative affect decrease which means they are negativly correlated. The diagonal is showing the distribution of each variable and which helps us understand the skweness of the data.

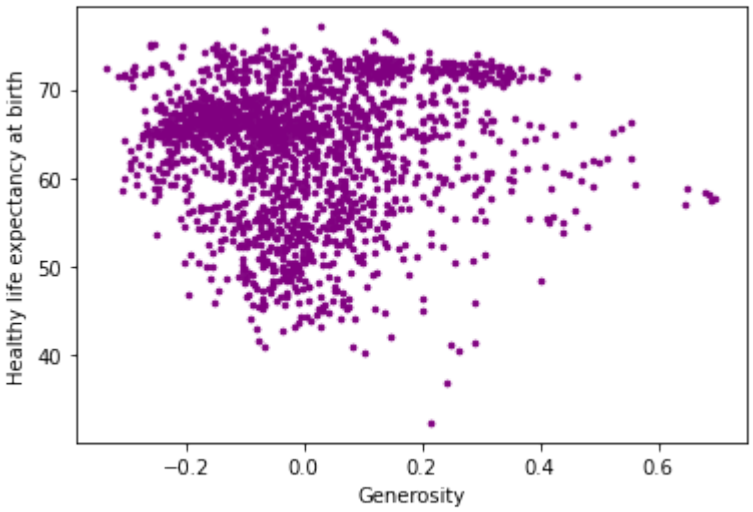
Creating scatter plots to show relationship between each variable with label 'Healthy life expectancy at birth'

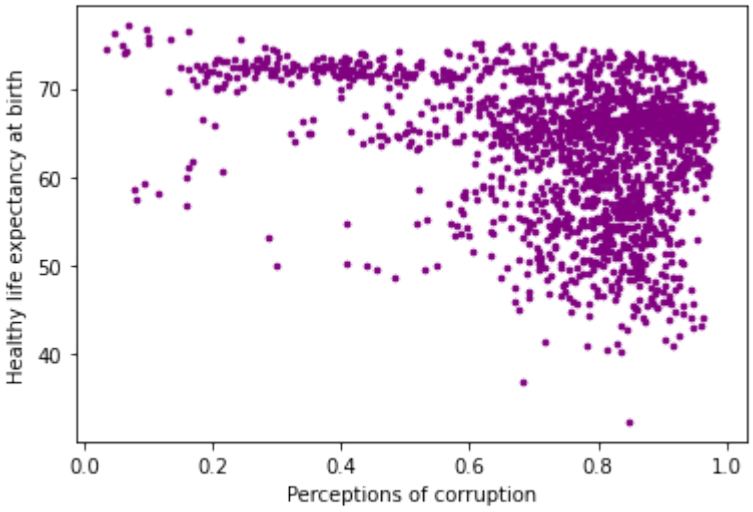
In [ ]:
```
variables = ['Log GDP per capita', 'Social support', 'Freedom to make
life choices',
             'Generosity', 'Perceptions of corruption', 'Positive affe
ct', 'Negative affect']
for variable in variables:
    df.plot.scatter(x=variable, y='Healthy life expectancy at birth',
s=7, color = "purple")
```

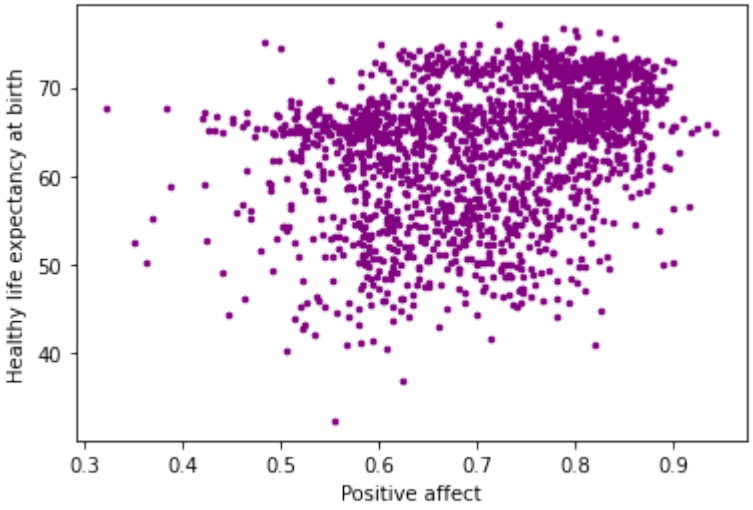

Healthy life expectancy at birth
70
60
50
40
7
8
9
10
11
Log GDP per capita

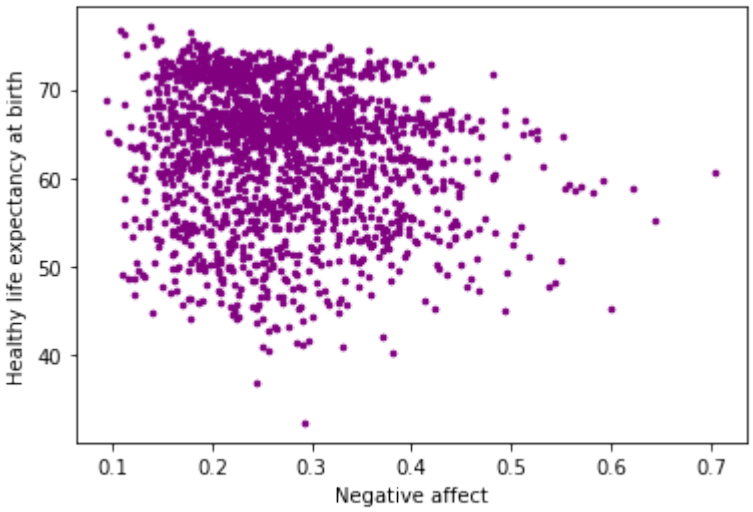

Healthy life expectancy at birth
70
60
50
40
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Social support


Healthy life expectancy at birth
70
60
50
40
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
Freedom to make life choices


Healthy life expectancy at birth
70
60
50
40
−0.2
0.0
0.2
0.4
0.6
Generosity


Healthy life expectancy at birth
70
60
50
40
0.0
0.2
0.4
0.6
0.8
1.0
Perceptions of corruption


Healthy life expectancy at birth
70
60
50
40
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Positive affect



Calculating the Pearson's correlation coefficient r between each data attribute

In [ ]:
```
corr = df.corr(method='pearson')
corr
```

Out[ ]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Ne
Log GDP per capita	1.000000	0.692602	0.848049	0.367932	-0.000915	-0.345511	0.302282	-0.
Social support	0.692602	1.000000	0.616037	0.410402	0.067000	-0.219040	0.432152	-0.
Healthy life expectancy at birth	0.848049	0.616037	1.000000	0.388681	0.020737	-0.322461	0.318247	-0.
Freedom to make life choices	0.367932	0.410402	0.388681	1.000000	0.329300	-0.487883	0.606114	-0.
Generosity	-0.000915	0.067000	0.020737	0.329300	1.000000	-0.290706	0.358006	-0.
Perceptions of corruption	-0.345511	-0.219040	-0.322461	-0.487883	-0.290706	1.000000	-0.296517	0.
Positive affect	0.302282	0.432152	0.318247	0.606114	0.358006	-0.296517	1.000000	-0.
Negative affect	-0.210781	-0.395865	-0.139477	-0.267661	-0.092542	0.264225	-0.374439	1.

Plotting the correlation coefficient values using a heatmap

In [ ]:
```
corr.style.background_gradient(cmap='bwr').set_precision(2)
```

```
<ipython-input-14-df65042b4cca>:1: FutureWarning: this method is depre
cated in favour of `Styler.format(precision=..)`
  corr.style.background_gradient(cmap='bwr').set_precision(2)
```

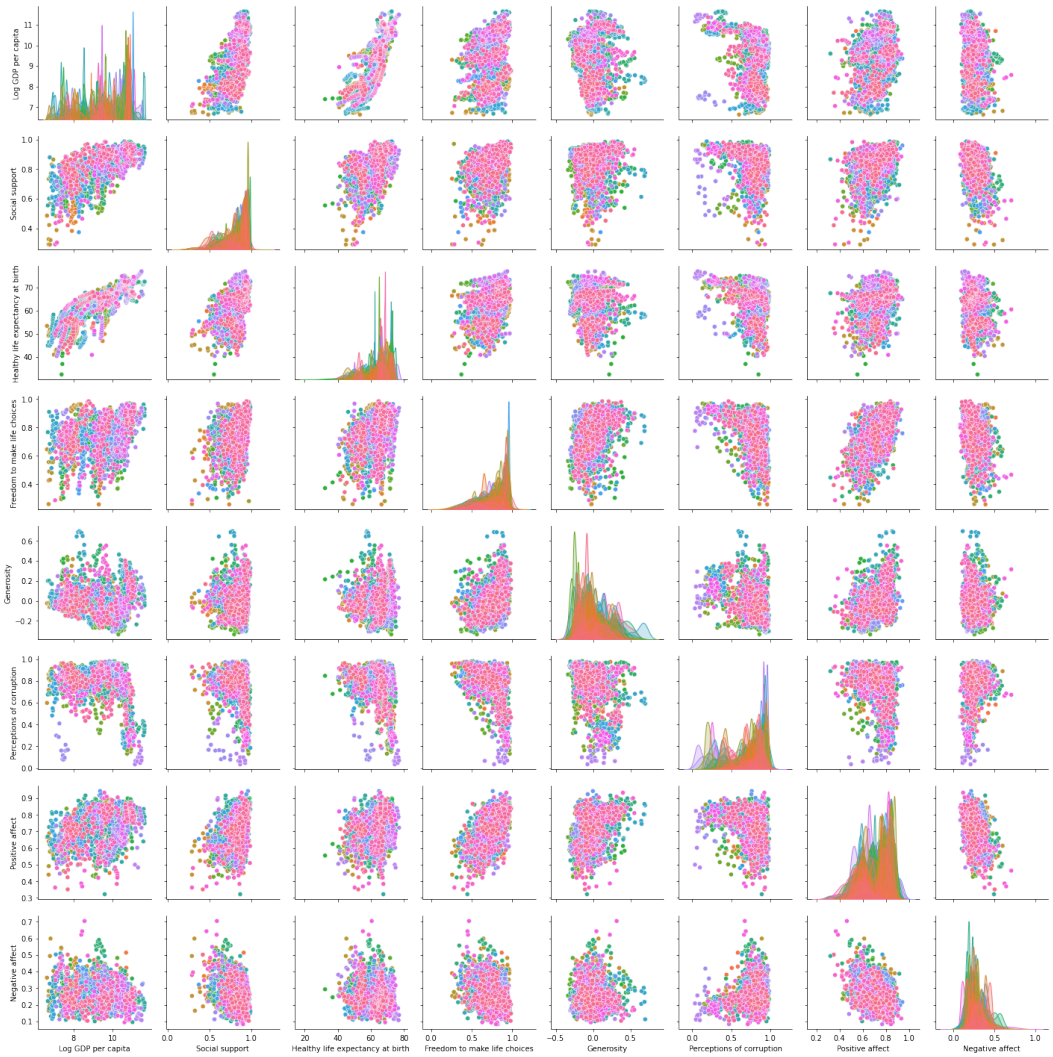
Out[ ]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negative affect
Log GDP per capita	1.00	0.69	0.85	0.37	-0.00	-0.35	0.30	-0.21
Social support	0.69	1.00	0.62	0.41	0.07	-0.22	0.43	-0.40
Healthy life expectancy at birth	0.85	0.62	1.00	0.39	0.02	-0.32	0.32	-0.14
Freedom to make life choices	0.37	0.41	0.39	1.00	0.33	-0.49	0.61	-0.27
Generosity	-0.00	0.07	0.02	0.33	1.00	-0.29	0.36	-0.09
Perceptions of corruption	-0.35	-0.22	-0.32	-0.49	-0.29	1.00	-0.30	0.26
Positive affect	0.30	0.43	0.32	0.61	0.36	-0.30	1.00	-0.37
Negative affect	-0.21	-0.40	-0.14	-0.27	-0.09	0.26	-0.37	1.00

Using pairplot to visualize with respect to Countries.

In [ ]:
```
sns.pairplot(df, hue = 'Country name', dropna = "TRUE")
```

Out[ ]: <seaborn.axisgrid.PairGrid at 0x7f200245b040>



Country name

- Afghanistan
- Albania
- Algeria
- Angola
- Argentina
- Armenia
- Australia
- Austria
- Azerbaijan
- Bahrain
- Bangladesh
- Belarus
- Belgium
- Belize
- Benin
- Bhutan
- Bolivia
- Bosnia and Herzegovina
- Botswana
- Brazil
- Bulgaria
- Burkina Faso
- Burundi
- Cambodia
- Cameroon
- Canada
- Central African Republic
- Chad
- Chile
- China
- Colombia
- Comoros
- Congo (Brazzaville)
- Congo (Kinshasa)
- Costa Rica
- Croatia
- Cuba
- Cyprus
- Czech Republic
- Denmark
- Djibouti
- Dominican Republic
- Ecuador
- Egypt
- El Salvador
- Estonia
- Ethiopia
- Finland
- France
- Gabon
- Gambia
- Georgia
- Germany
- Ghana
- Greece
- Guatemala
- Guinea
- Guyana
- Haiti
- Honduras
- Hong Kong S.A.R. of China
- Hungary
- Iceland
- India
- Indonesia
- Iran
- Iraq
- Ireland
- Israel
- Italy
- Ivory Coast
- Jamaica
- Japan
- Jordan
- Kazakhstan
- Kenya
- Kosovo
- Kuwait
- Kyrgyzstan
- Laos
- Latvia
- Lebanon
- Lesotho
- Liberia
- Libya
- Lithuania
- Luxembourg
- Madagascar
- Malawi
- Malaysia
- Maldives
- Mali
- Malta
- Mauritania
- Mauritius
- Mexico
- Moldova
- Mongolia
- Montenegro
- Morocco
- Mozambique
- Myanmar
- Namibia
- Nepal
- Netherlands
- New Zealand
- Nicaragua
- Niger
- Nigeria
- North Cyprus
- North Macedonia
- Norway
- Oman
- Pakistan
- Palestinian Territories
- Panama
- Paraguay
- Peru
- Philippines
- Poland
- Portugal
- Qatar
- Romania
- Russia
- Rwanda
- Saudi Arabia
- Senegal
- Serbia
- Sierra Leone
- Singapore
- Slovakia
- Slovenia
- Somalia
- Somaliland region
- South Africa
- South Korea
- South Sudan
- Spain
- Sri Lanka
- Sudan
- Suriname
- Swaziland
- Sweden
- Switzerland
- Syria
- Taiwan Province of China
- Tajikistan
- Tanzania
- Thailand
- Togo
- Trinidad and Tobago
- Tunisia
- Turkey
- Turkmenistan
- Uganda
- Ukraine
- United Arab Emirates
- United Kingdom
- United States
- Uruguay
- Uzbekistan
- Venezuela
- Vietnam
- Yemen
- Zambia
- Zimbabwe

Checking the null values and then treating them with the median of the data as the data is skewed here.

Checking the null values

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Country name                          0
Log GDP per capita                   36
Social support                       13
Healthy life expectancy at birth     55
Freedom to make life choices         32
Generosity                           89
Perceptions of corruption           110
Positive affect                      22
Negative affect                      16
dtype: int64
```

Replacing null values with median

In [ ]:
```
df['Log GDP per capita'].fillna(value=df['Log GDP per capita'].median
(), inplace=True)
df['Social support'].fillna(value=df['Social support'].median(), inpla
ce=True)
df['Healthy life expectancy at birth'].fillna(value=df['Healthy life e
xpectancy at birth'].median(), inplace=True)
df['Freedom to make life choices'].fillna(value=df['Freedom to make li
fe choices'].median(), inplace=True)
df['Generosity'].fillna(value=df['Generosity'].median(), inplace=True)
df['Perceptions of corruption'].fillna(value=df['Perceptions of corrup
tion'].median(), inplace=True)
df['Positive affect'].fillna(value=df['Positive affect'].median(), inp
lace=True)
df['Negative affect'].fillna(value=df['Negative affect'].median(), inp
lace=True)
```

One hot encoding

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Country name                          0
Log GDP per capita                    0
Social support                        0
Healthy life expectancy at birth      0
Freedom to make life choices          0
Generosity                            0
Perceptions of corruption             0
Positive affect                       0
Negative affect                       0
dtype: int64
```

There are no null values present in the data anymore.

Converting categorical values into numerical

In [ ]:
```
y = pd.get_dummies(df["Country name"], prefix='Country')
print(y.head())
```

```
   Country_Afghanistan  Country_Albania  Country_Algeria  Country_Ango
la  \
0                    1                0                0
0
1                    1                0                0
0
2                    1                0                0
0
3                    1                0                0
0
4                    1                0                0
0

   Country_Argentina  Country_Armenia  Country_Australia  Country_Aust
ria  \
0                  0                0                  0
0
1                  0                0                  0
0
2                  0                0                  0
0
3                  0                0                  0
0
4                  0                0                  0
0

   Country_Azerbaijan  Country_Bahrain  ...  Country_United Arab Emira
tes  \
0                   0                0  ...
0
1                   0                0  ...
0
2                   0                0  ...
0
3                   0                0  ...
0
4                   0                0  ...
0

   Country_United Kingdom  Country_United States  Country_Uruguay  \
0                       0                      0                0
1                       0                      0                0
2                       0                      0                0
3                       0                      0                0
4                       0                      0                0

   Country_Uzbekistan  Country_Venezuela  Country_Vietnam  Country_Yem
en  \
0                   0                  0                0
0
1                   0                  0                0
0
2                   0                  0                0
0
3                   0                  0                0
0
4                   0                  0                0
```

```
0

   Country_Zambia  Country_Zimbabwe
0               0                 0
1               0                 0
2               0                 0
3               0                 0
4               0                 0

[5 rows x 166 columns]
```

In [ ]:
```
df.info() # no null values left
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1949 entries, 0 to 1948
Data columns (total 9 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Country name                      1949 non-null   object 
 1   Log GDP per capita                1949 non-null   float64
 2   Social support                    1949 non-null   float64
 3   Healthy life expectancy at birth  1949 non-null   float64
 4   Freedom to make life choices      1949 non-null   float64
 5   Generosity                        1949 non-null   float64
 6   Perceptions of corruption         1949 non-null   float64
 7   Positive affect                   1949 non-null   float64
 8   Negative affect                   1949 non-null   float64
dtypes: float64(8), object(1)
memory usage: 137.2+ KB
```

Concatenating dataframes

In [ ]:
```
df1 = pd.concat([df, y], axis=1)
```

In [ ]:
```
df1.head
```

Out[ ]:
```
<bound method NDFrame.head of      Country name  Log GDP per capita  S
ocial support  \
0     Afghanistan               7.370           0.451
1     Afghanistan               7.540           0.552
2     Afghanistan               7.647           0.539
3     Afghanistan               7.620           0.521
4     Afghanistan               7.705           0.521
...           ...                 ...             ...
1944     Zimbabwe               7.984           0.768
1945     Zimbabwe               8.016           0.754
1946     Zimbabwe               8.049           0.775
1947     Zimbabwe               7.950           0.759
1948     Zimbabwe               7.829           0.717

      Healthy life expectancy at birth  Freedom to make life choices
\
0                                50.80                         0.718
1                                51.20                         0.679
2                                51.60                         0.600
3                                51.92                         0.496
4                                52.24                         0.531
...                                ...                           ...
1944                             54.40                         0.733
1945                             55.00                         0.753
1946                             55.60                         0.763
1947                             56.20                         0.632
1948                             56.80                         0.643

      Generosity  Perceptions of corruption  Positive affect  Negative
affect  \
0          0.168                      0.882            0.518
0.258
1          0.190                      0.850            0.584
0.237
2          0.121                      0.707            0.618
0.275
3          0.162                      0.731            0.611
0.267
4          0.236                      0.776            0.710
0.268
...          ...                        ...              ...
...
1944      -0.095                      0.724            0.738
0.209
1945      -0.098                      0.751            0.806
0.224
1946      -0.068                      0.844            0.710
0.212
1947      -0.064                      0.831            0.716
0.235
1948      -0.009                      0.789            0.703
0.346

      Country_Afghanistan  ...  Country_United Arab Emirates  \
0                       1  ...                             0
1                       1  ...                             0
2                       1  ...                             0
```

```
3                     1  ...                         0
4                     1  ...                         0
...                 ...  ...                       ...
1944                  0  ...                         0
1945                  0  ...                         0
1946                  0  ...                         0
1947                  0  ...                         0
1948                  0  ...                         0

      Country_United Kingdom  Country_United States  Country_Uruguay
\
0                           0                      0                0
1                           0                      0                0
2                           0                      0                0
3                           0                      0                0
4                           0                      0                0
...                       ...                    ...              ...
1944                        0                      0                0
1945                        0                      0                0
1946                        0                      0                0
1947                        0                      0                0
1948                        0                      0                0

      Country_Uzbekistan  Country_Venezuela  Country_Vietnam  Country_
Yemen  \
0                       0                  0                0
0
1                       0                  0                0
0
2                       0                  0                0
0
3                       0                  0                0
0
4                       0                  0                0
0
...                   ...                ...              ...
...
1944                    0                  0                0
0
1945                    0                  0                0
0
1946                    0                  0                0
0
1947                    0                  0                0
0
1948                    0                  0                0
0

      Country_Zambia  Country_Zimbabwe
0                  0                 0
1                  0                 0
2                  0                 0
3                  0                 0
4                  0                 0
...              ...               ...
1944               0                 1
1945               0                 1
```

```
1946                0                 1
1947                0                 1
1948                0                 1

[1949 rows x 175 columns]>
```

In [ ]:
```
df1 = df1.drop(["Country name"], axis = 1)
```

In [ ]:
```
df1.head(10)
```

Out[ ]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negative affect	Country_A
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	
5	7.725	0.484	52.56	0.578	0.061	0.823	0.621	0.273	
6	7.718	0.526	52.88	0.509	0.104	0.871	0.532	0.375	
7	7.702	0.529	53.20	0.389	0.080	0.881	0.554	0.339	
8	7.697	0.559	53.00	0.523	0.042	0.793	0.565	0.348	
9	7.697	0.491	52.80	0.427	-0.121	0.954	0.496	0.371	

10 rows × 174 columns

Stratification

Splitting the data into test set 20 % and verifying the set is sample representative of the population original dataset

This is done using stratified sampling method. Reference for code and concept is the text book here

Here a dummy column hasd been made to check the distribution of the data to be same in original column from original dataset and the stratified test dataset.

Then a table will also tell the error between if we would have used train test split and strtified sampling against the original column.

In [ ]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(df1, test_size=0.2, random_stat
e=42)
```

In [ ]:
```
test_set.describe().T
```

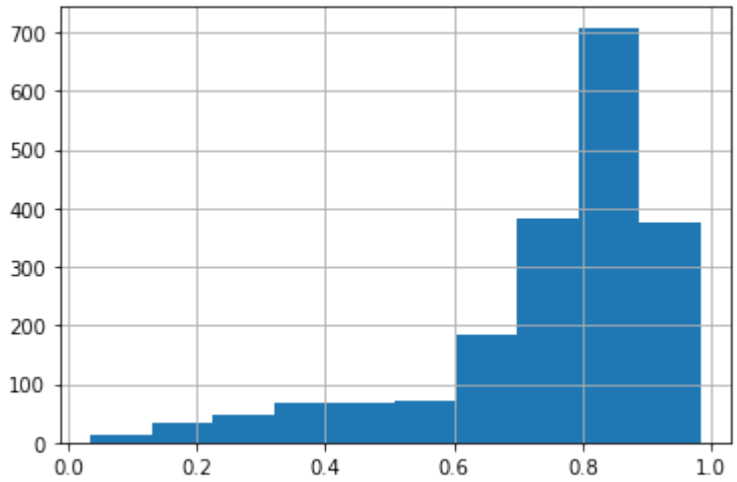
Out[ ]:

	count	mean	std	min	25%	50%	75%	max
Log GDP per capita	390.0	9.392251	1.115580	6.723	8.50875	9.4615	10.39200	11.520
Social support	390.0	0.812183	0.117070	0.326	0.74250	0.8355	0.90650	0.975
Healthy life expectancy at birth	390.0	63.552403	7.478433	40.808	59.68000	65.2000	68.47500	75.000
Freedom to make life choices	390.0	0.745744	0.139500	0.258	0.66100	0.7630	0.85450	0.980
Generosity	390.0	-0.001995	0.154058	-0.296	-0.10475	-0.0255	0.08725	0.650
...	...	...	...	...	...	...	...	...
Country_Venezuela	390.0	0.015385	0.123235	0.000	0.00000	0.0000	0.00000	1.000
Country_Vietnam	390.0	0.000000	0.000000	0.000	0.00000	0.0000	0.00000	0.000
Country_Yemen	390.0	0.005128	0.071519	0.000	0.00000	0.0000	0.00000	1.000
Country_Zambia	390.0	0.007692	0.087480	0.000	0.00000	0.0000	0.00000	1.000
Country_Zimbabwe	390.0	0.012821	0.112644	0.000	0.00000	0.0000	0.00000	1.000

174 rows × 8 columns

In [ ]:
```
df1["Perceptions of corruption"].hist()
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ff2cf4c40>
```



In [ ]:
```
df1["perp_cat"] = pd.cut(df1["Perceptions of corruption"],
                         bins=[0, 0.3, 0.45, 0.6, 0.75, 1],
                         labels=[1, 2, 3, 4,5])
```

In [ ]:
```
from sklearn.model_selection import StratifiedShuffleSplit

split1 = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_stat
e=42)
for train_index, test_index in split1.split(df1, df1["perp_cat"]):
    strat_train_set = df1.loc[train_index]
    strat_test_set = df1.loc[test_index]
train_index
```

Out[ ]:
```
array([1519,  768,   49, ...,   56,  515, 1160])
```

In [ ]:
```
strat_test_set["perp_cat"].value_counts() / len(strat_test_set)
```

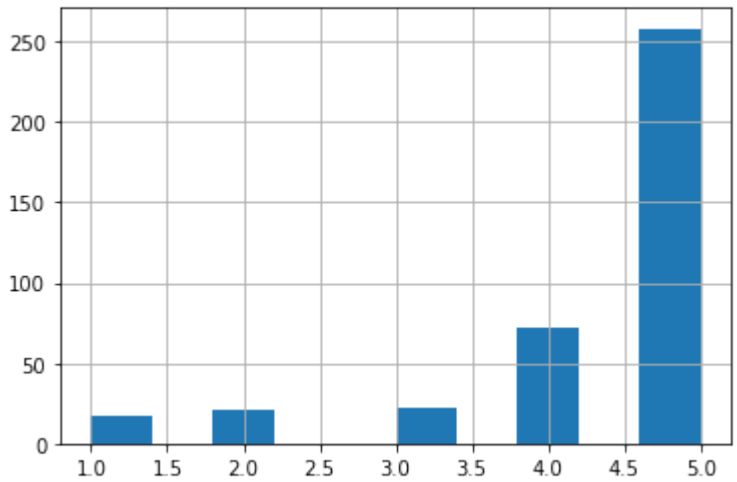
Out[ ]:
```
5    0.661538
4    0.184615
3    0.056410
2    0.053846
1    0.043590
Name: perp_cat, dtype: float64
```

In [ ]:
```
df1["perp_cat"].value_counts() / len(df1)
```

Out[ ]:
```
5    0.660852
4    0.185223
3    0.056439
2    0.053361
1    0.044125
Name: perp_cat, dtype: float64
```

In [ ]:
```
strat_test_set["perp_cat"].hist()
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1ff2c539a0>
```



In [ ]:
```
def perp_cat_proportions(data):
    return data["perp_cat"].value_counts() / len(data)

train_set, test_set = train_test_split(df1, test_size=0.2, random_stat
e=42)

compare_props = pd.DataFrame({
    "Overall": perp_cat_proportions(df1),
    "Stratified": perp_cat_proportions(strat_test_set),
    "Random": perp_cat_proportions(test_set),
}).sort_index()
compare_props["Rand. %error"] = 100 * compare_props["Random"] / compar
e_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] / c
ompare_props["Overall"] - 100
```

In [ ]:
```
compare_props
```

Out[ ]:

	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.044125	0.043590	0.035897	-18.646392	-1.213476
2	0.053361	0.053846	0.074359	39.351578	0.909763
3	0.056439	0.056410	0.058974	4.491841	-0.051282
4	0.185223	0.184615	0.184615	-0.328148	-0.328148
5	0.660852	0.661538	0.646154	-2.224080	0.103918

Preparing for k fold method

In [ ]:
```
datako = strat_train_set.copy()
data = datako.drop(["perp_cat"], axis = 1)
data.head()
```

Out[ ]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negative affect	Counti
1519	7.417	0.650	50.8	0.716	0.095	0.856	0.552	0.466	
768	9.449	0.572	65.1	0.780	0.176	0.699	0.645	0.520	
49	10.032	0.900	68.8	0.846	-0.211	0.855	0.820	0.321	
1473	10.779	0.867	64.8	0.560	-0.120	0.802	0.715	0.225	
458	10.878	0.960	71.5	0.941	0.222	0.191	0.829	0.218	

5 rows × 174 columns

Scaling the features with standardscalar

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scaled_features = StandardScaler().fit_transform(data.values)

scaled_features_data = pd.DataFrame(scaled_features, index=data.index,
columns=data.columns)
scaled_features_data.head()
```

Out[ ]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negative affect
1519	-1.695973	-1.395090	-1.687549	-0.196812	0.581962	0.580990	-1.517637	2.328542
768	0.067212	-2.062037	0.229016	0.258257	1.087240	-0.286605	-0.638183	2.964019
49	0.573086	0.742562	0.724911	0.727547	-1.326865	0.575464	1.016703	0.622169
1473	1.221264	0.460392	0.188809	-1.306044	-0.759207	0.282582	0.023772	-0.507568
458	1.307167	1.255598	1.086780	1.403041	1.374188	-3.093853	1.101812	-0.589944

5 rows × 174 columns

In [ ]:
```
df2 = scaled_features_data.copy()
df2.head()
```

Out[ ]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negative affect
1519	-1.695973	-1.395090	-1.687549	-0.196812	0.581962	0.580990	-1.517637	2.328542
768	0.067212	-2.062037	0.229016	0.258257	1.087240	-0.286605	-0.638183	2.964019
49	0.573086	0.742562	0.724911	0.727547	-1.326865	0.575464	1.016703	0.622169
1473	1.221264	0.460392	0.188809	-1.306044	-0.759207	0.282582	0.023772	-0.507568
458	1.307167	1.255598	1.086780	1.403041	1.374188	-3.093853	1.101812	-0.589944

5 rows × 174 columns

In [ ]:
```
y_labels = df2.iloc[:, 2]
y_labelsnp = y_labels.values
y_labelsnp.shape
```

Out[ ]:
```
(1559,)
```

In [ ]:
```
x_train = df2.drop(['Healthy life expectancy at birth'], axis=1)
x_trainnp = x_train.values
x_trainnp.shape
```

Out[ ]:
```
(1559, 173)
```

Regression Models

In [ ]:
```
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Ridge, Lasso, ElasticNet, SGDRegresso
r
from sklearn.model_selection import train_test_split
from copy import deepcopy
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
```

In [ ]:
```
#function for regression models

def fit_and_evaluate_model(model, x_trainnp, y_labelsnp, degree = 2):
    kfd = KFold(n_splits=4)
    kfd.get_n_splits(x_trainnp)
    train_losses = []
    val_losses = []
    train_losses_mae = []
    val_losses_mae = []
    train_losses_root = []
    val_losses_root = []
    train_losses_r2 = []
    val_losses_r2 = []

    if degree == 2:
        poly = PolynomialFeatures(degree=degree)
        X = poly.fit_transform(x_trainnp)
        x_trainnp = X

    for train_index, val_index in kfd.split(x_trainnp):
        X_train, X_val = x_trainnp[train_index], x_trainnp[val_index]
        y_train, y_val = y_labelsnp[train_index], y_labelsnp[val_inde
x]

        model.fit(X_train, y_train)
        y_train_pred = model.predict(X_train)
        y_val_pred = model.predict(X_val)

#calculating mean squared error (MSE)
        train_loss = mean_squared_error(y_train, y_train_pred)
        val_loss = mean_squared_error(y_val, y_val_pred)

        train_losses.append(train_loss)
        val_losses.append(val_loss)

#calculating mean absolute error (MAE)

        train_loss_mae = mean_absolute_error(y_train, y_train_pred)
        val_loss_mae = mean_absolute_error(y_val, y_val_pred)

        train_losses_mae.append(train_loss_mae)
        val_losses_mae.append(val_loss_mae)

#calculating root mean absolute error (RMSE)

        train_loss_root = np.sqrt(mean_squared_error(y_train, y_train_
pred))
        val_loss_root = np.sqrt(mean_squared_error(y_val, y_val_pred))

        train_losses_root.append(train_loss_root)
        val_losses_root.append(val_loss_root)


#calculating R squared score

        train_loss_r2 = r2_score(y_train, y_train_pred)
        val_loss_r2 = r2_score(y_val, y_val_pred)
```

```
        train_losses_r2.append(train_loss_r2)
        val_losses_r2.append(val_loss_r2)

    avg_train_loss = np.mean(train_losses)
    avg_val_loss = np.mean(val_losses)

    avg_train_loss_mae = np.mean(train_losses_mae)
    avg_val_loss_mae = np.mean(val_losses_mae)

    avg_train_loss_root = np.mean(train_losses_root)
    avg_val_loss_root = np.mean(val_losses_root)

    avg_train_loss_r2 = np.mean(train_losses_r2)
    avg_val_loss_r2 = np.mean(val_losses_r2)

    print ("avg_train_loss: ", avg_train_loss)
    print ("avg_val_loss: ", avg_val_loss)

    print ("avg_train_loss_mae: ", avg_train_loss_mae)
    print ("avg_val_loss_mae: ", avg_val_loss_mae)

    print ("avg_train_loss_root: ", avg_train_loss_root)
    print ("avg_val_loss_root: ", avg_val_loss_root)

    print ("avg_train_loss_r_squared: ", avg_train_loss_r2)
    print ("avg_val_loss_r_squared: ", avg_val_loss_r2)

    plt.plot(train_losses, label='Training Loss',marker='o')
    plt.plot(val_losses, label='Validation Loss',marker='x')
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()

    return avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_l
oss_mae, avg_train_loss_root, avg_val_loss_root, avg_train_loss_r2, av
g_val_loss_r2
```

In [ ]:

```
#function to iterate over alpha
def alpha_iterate(x_trainnp, y_labelsnp, degree =1):
    alphas = np.linspace(0.05, 1,100)
    kf = KFold(n_splits=4)
    kf.get_n_splits(x_trainnp)
    train_losses = []
    val_losses = []

    for alpha in alphas:
        alpha_train_losses = []
        alpha_val_losses = []


        if degree == 2:
            poly = PolynomialFeatures(degree=degree)
            X = poly.fit_transform(x_trainnp)
            x_trainnp = X



        for train_index, val_index in kf.split(x_trainnp):
            X_train, X_val = x_trainnp[train_index], x_trainnp[val_ind
ex]
            y_train, y_val = y_labelsnp[train_index], y_labelsnp[val_i
ndex]

            #model = Ridge(alpha = alpha).fit(X_train, y_train)
            #model = Lasso(alpha = alpha).fit(X_train, y_train)
            #model = ElasticNet(alpha = alpha, l1_ratio = 0.5).fit(X_t
rain, y_train)
            #model = SGDRegressor(max_iter=1000, tol=1e-3, penalty="l
2", eta0=0.1, random_state=42, alpha = alpha).fit(X_train, y_train)
            #model = SGDRegressor(max_iter=1000, tol=1e-3, penalty="l
1", eta0=0.1, random_state=42, alpha = alpha).fit(X_train, y_train)
            model = SGDRegressor(max_iter=1000, tol=1e-3, penalty="ela
sticnet", eta0=0.1, random_state=42, alpha = alpha,l1_ratio = 0.5).fit
(X_train, y_train)


            y_train_pred = model.predict(X_train)
            y_val_pred = model.predict(X_val)

            train_loss = mean_squared_error(y_train, y_train_pred)
            val_loss = mean_squared_error(y_val, y_val_pred)

            alpha_train_losses.append(train_loss)
            alpha_val_losses.append(val_loss)

        avg_train_loss = np.mean(alpha_train_losses)
        avg_val_loss = np.mean(alpha_val_losses)

        train_losses.append(avg_train_loss)
        val_losses.append(avg_val_loss)

    plt.plot(alphas, train_losses, label='train')
    plt.plot(alphas, val_losses, label='validation')
    plt.xscale('log')
```

```
    plt.xlabel('alpha')
    plt.ylabel('Loss')
    plt.legend()
    plt.show()
```

In [ ]:
```
#function for learning curves
def plot_learning_curves_kfold(model, X, y):
    kf = KFold(n_splits=4)
    kf.get_n_splits(x_trainnp)

    for train_index, val_index in kf.split(x_trainnp):
        X_train, X_val = x_trainnp[train_index], x_trainnp[val_index]
        y_train, y_val = y_labelsnp[train_index], y_labelsnp[val_inde
x]

        train_errors, val_errors = [], []
        for m in range(1, len(X_train) + 1):
            model.fit(X_train[:m], y_train[:m])
            y_train_predict = model.predict(X_train[:m])
            y_val_predict = model.predict(X_val)
            train_errors.append(mean_squared_error(y_train[:m], y_trai
n_predict))
            val_errors.append(mean_squared_error(y_val, y_val_predic
t))

        plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="tra
in")
        plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
        plt.legend(loc="upper right", fontsize=14)
        plt.xlabel("Training set size", fontsize=14)
        plt.ylabel("RMSE", fontsize=14)
        plt.axis([0, 100, 0, 3])
        plt.show()
```

In [ ]:
```
#function to plot learning curves
def plot_learning_curves(model, X, y):
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=
0.2, random_state=10)
    train_errors, val_errors = [], []
    for m in range(1, len(X_train) + 1):
        model.fit(X_train[:m], y_train[:m])
        y_train_predict = model.predict(X_train[:m])
        y_val_predict = model.predict(X_val)
        train_errors.append(mean_squared_error(y_train[:m], y_train_pr
edict))
        val_errors.append(mean_squared_error(y_val, y_val_predict))

    plt.plot(np.sqrt(train_errors), "r-+", linewidth=2, label="train")
    plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="val")
    plt.legend(loc="upper right", fontsize=14)   # not shown in the bo
ok
    plt.xlabel("Training set size", fontsize=14) # not shown
    plt.ylabel("RMSE", fontsize=14)              # not shown
```

In [ ]:
```
#function to get performance metrics
def get_metrics(model, x_trainnp, y_labelsnp, degree =1):
  X_train, X_val, y_train, y_val = train_test_split(x_trainnp, y_label
snp, test_size=0.2, random_state=42)
  if degree == 2:
        poly = PolynomialFeatures(degree=degree, include_bias = False)
        X_train_poly = poly.fit_transform(X_train)
        X_val_poly_scaled = poly.fit_transform(X_val)

  minimum_val_error = float("inf")
  best_epoch = None
  best_model = None
  for epoch in range(1000):
    model.fit(X_train, y_train)  # continues where it left off
    y_val_predict = model.predict(X_val)
    val_error = mean_squared_error(y_val, y_val_predict)
    if val_error < minimum_val_error:
        minimum_val_error = val_error
        best_epoch = epoch
        best_model = deepcopy(model)
```

In [ ]:
```
def plot_results():
  n_epochs = 20
  train_errors, val_errors = [], []
  for epoch in range(n_epochs):
    model.fit(X_train, y_train)
    y_train_predict = model.predict(X_train)
    y_val_predict = model.predict(X_val)
    train_errors.append(mean_squared_error(y_train, y_train_predict))
    val_errors.append(mean_squared_error(y_val, y_val_predict))


    best_epoch = np.argmin(val_errors)
    best_val_rmse = np.sqrt(val_errors[best_epoch])


  best_val_rmse -= 0.03  # just to make the graph look better
  plt.plot([0, n_epochs], [best_val_rmse, best_val_rmse], "k:", linewi
dth=2)
  plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="Validation s
et")
  plt.plot(np.sqrt(train_errors), "r--", linewidth=2, label="Training
set")
  plt.legend(loc="upper right", fontsize=14)
  plt.xlabel("Epoch", fontsize=14)
  plt.ylabel("RMSE", fontsize=14)
  plt.show()
```

In [ ]:
```
results_df = pd.DataFrame(columns=['Regression', 'Avg Training Loss',
'Avg Validation Loss', 'Avg Training Loss MAE', 'Avg Val Loss MAE','Av
g Training Loss RMSE', 'Avg Val Loss RMSE', 'Avg Training Loss R squar
ed', 'Avg Val Loss R squared' ])
```

We have run the asked models with variations of loss functions with the :

1) Kfold
2) Alpha(penalty term) to determine the optimum value of alpha 3) batch size and epoch

In the end a comparision between performances of all the models has been done and prediction has been made on the identified best model.

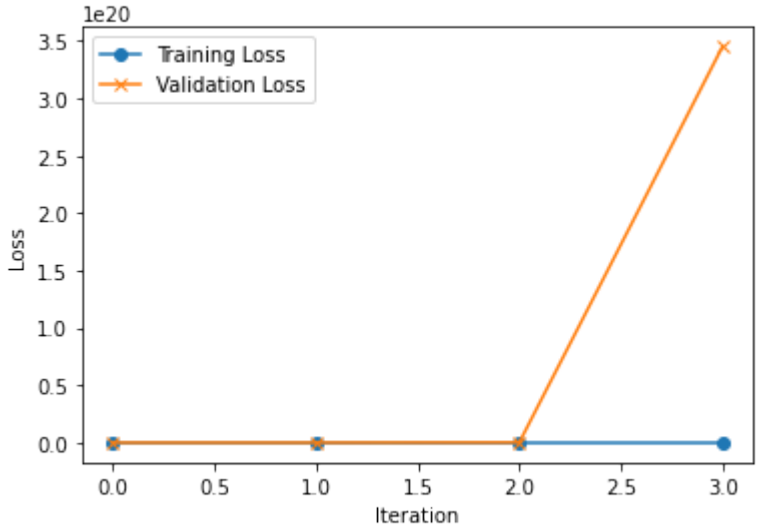
In some places like the polynomial SGD regularizaion, alpha iterations could not be plotted as google collab crashed.

Rference for code is the couse text book here

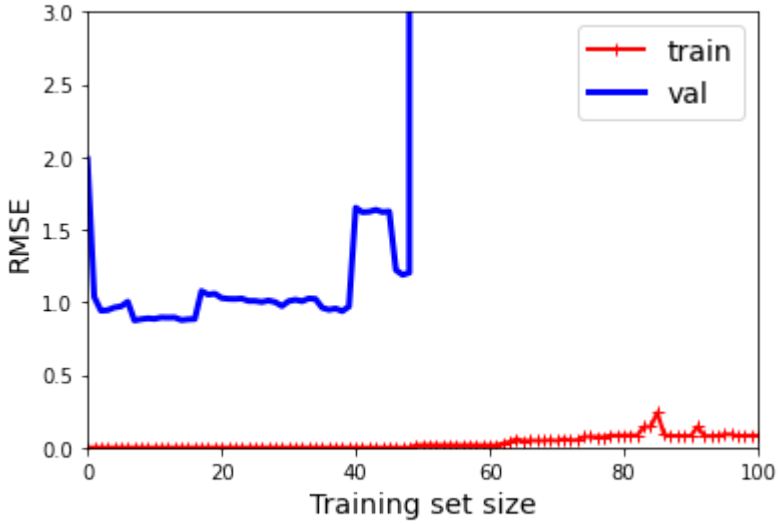
Linear Regression

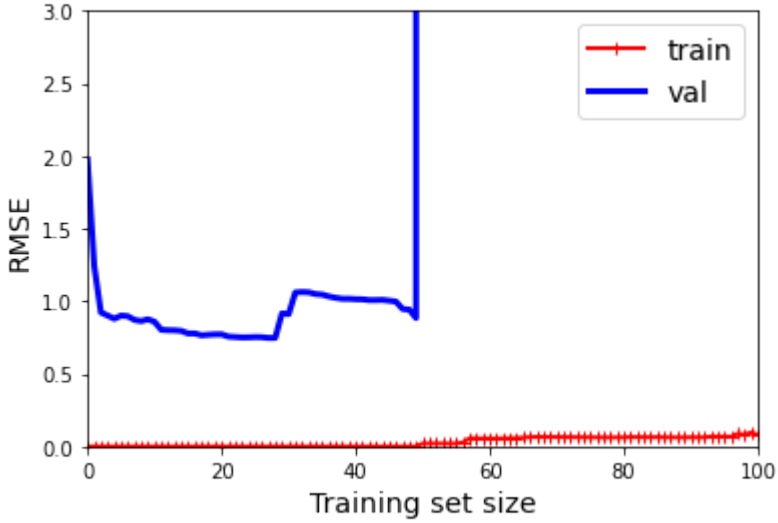
In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(LinearRegression(), x_trainnp, y_labelsnp)
results_df = results_df.append({'Regression': 'LR', 'Avg Training Los
s': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Training
Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss_mae,
'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMSE': av
g_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r2, 'Av
g Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

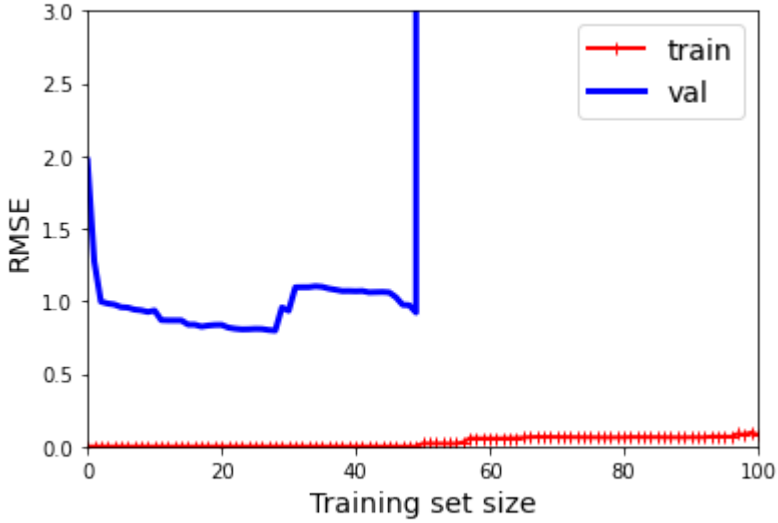
```
avg_train_loss:  0.0002837711165011684
avg_val_loss:  8.632475615176778e+19
avg_train_loss_mae:  0.007185570471666293
avg_val_loss_mae:  754456869.7964728
avg_train_loss_root:  0.016798006019620438
avg_val_loss_root:  4757067885.400153
avg_train_loss_r_squared:  0.999716298126192
avg_val_loss_r_squared:  -9.178719087946877e+19
```

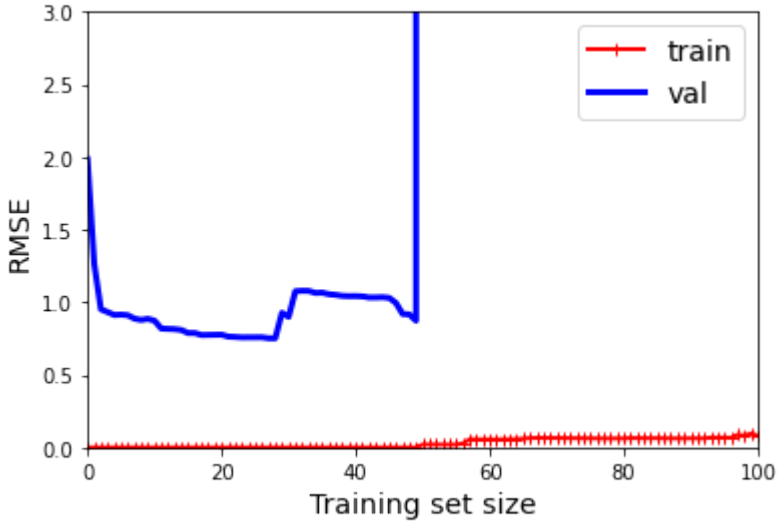


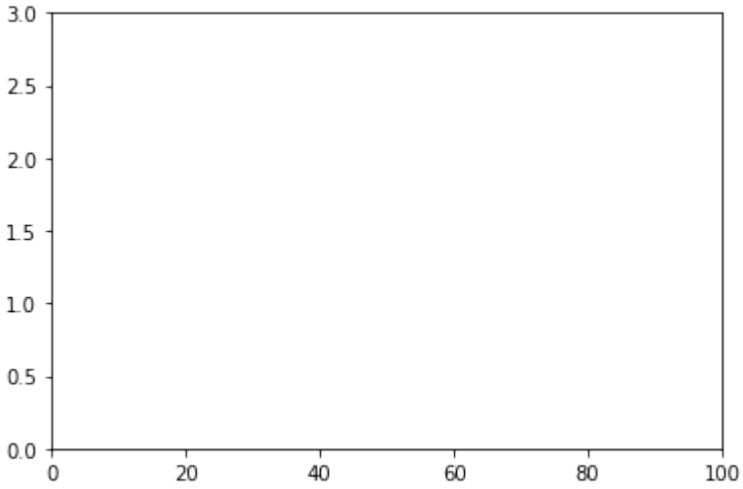
In [ ]:
```
#plotting learning curves
lin_reg = LinearRegression()
plot_learning_curves_kfold(lin_reg, x_trainnp, y_labelsnp)
plt.axis([0, 100, 0, 3])
plt.show()
```


train
val
RMSE
Training set size
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100


train
val
RMSE
Training set size
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100


train
val
RMSE
Training set size
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20
40
60
80
100



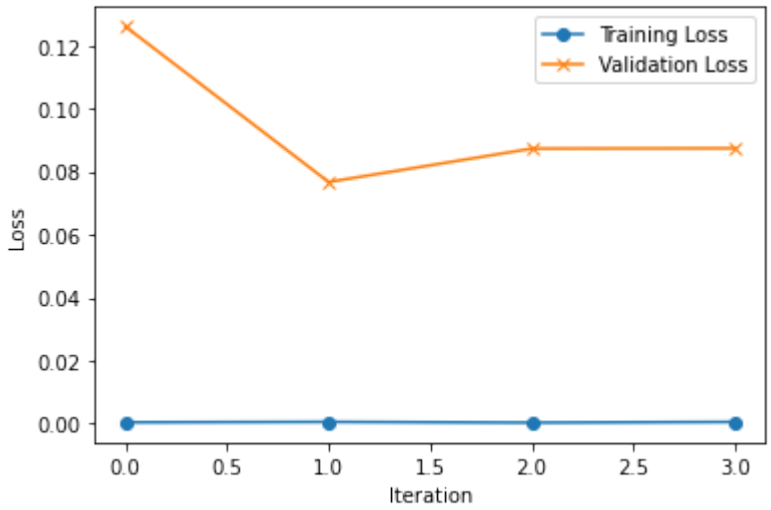


above for each fold the loss with increasing training size has been plot. It can be seen that the model is overfitted as training loss is constantly zero and vlidation loss shoots up higher

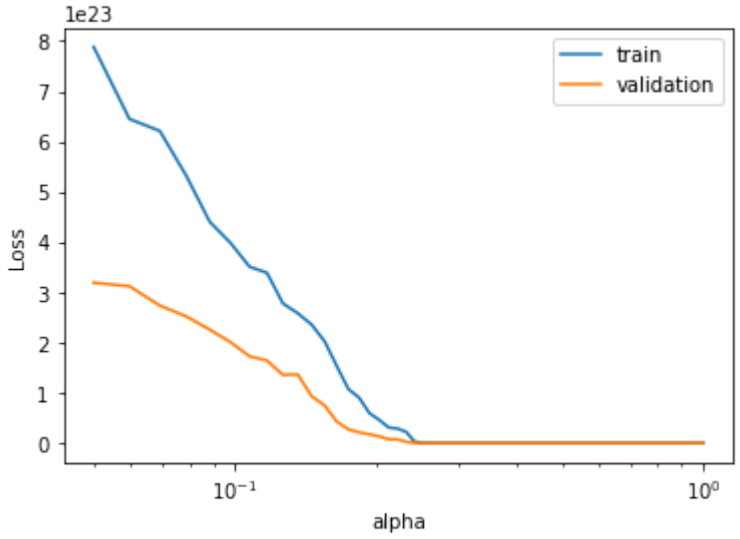
Linear Regression Model with Ridge Regularization

In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, avg_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_r2 = fit_and_evaluate_model(Ridge(alpha=0.1), x_trainnp, y_labelsnp)
results_df = results_df.append({'Regression': 'LR Ridge', 'Avg Training Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  0.00045595348970808326
avg_val_loss:  0.09436537031640028
avg_train_loss_mae:  0.010243459617789653
avg_val_loss_mae:  0.17025210094904847
avg_train_loss_root:  0.021220082114040084
avg_val_loss_root:  0.30577598013708973
avg_train_loss_r_squared:  0.9995459765374222
avg_val_loss_r_squared:  0.9058396865528563
```

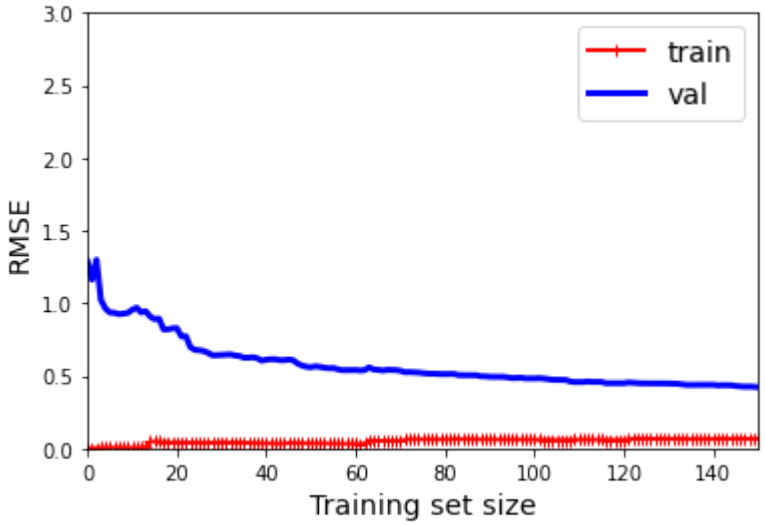


In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



here we can see and find the optimum value of alpha as the model is coverging, It can be in the elbow which is formed where train and validation loss meet.

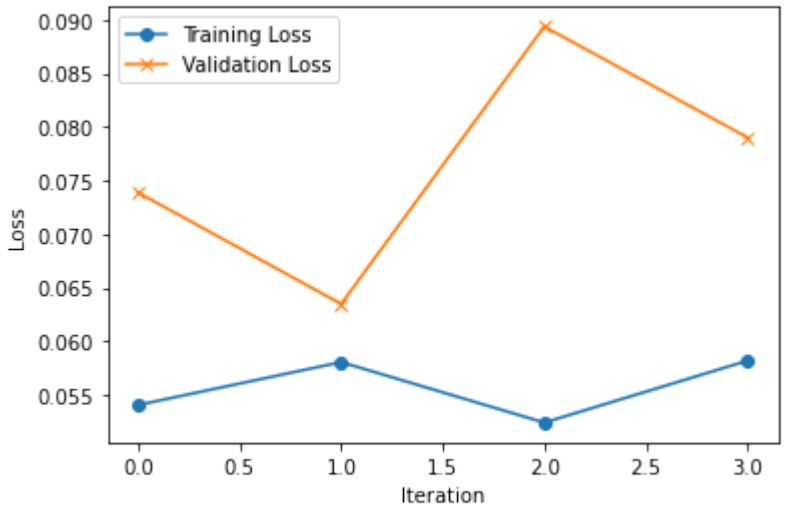
In [ ]:
```
lin_reg = Ridge(alpha = 1)
plot_learning_curves(lin_reg, x_trainnp, y_labelsnp)
plt.axis([0, 150, 0, 3])
plt.show()
```



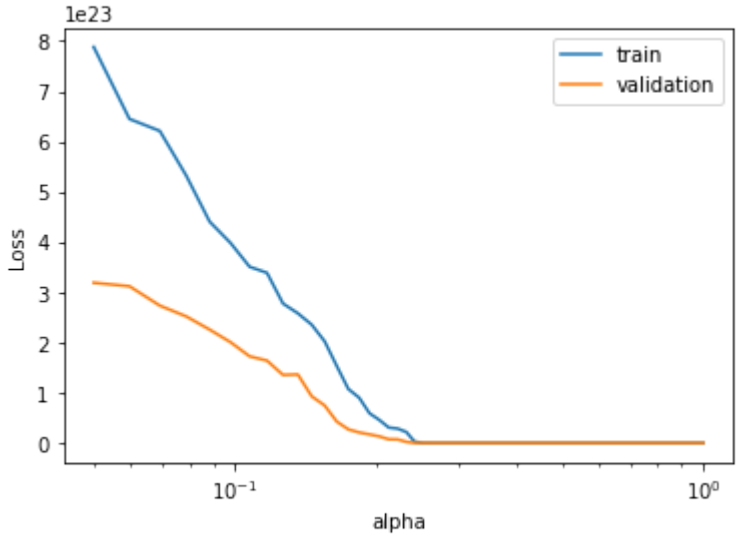
Linear Regression Model with Lasso Regularization

In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(Lasso(alpha=0.1), x_trainnp, y_labelsnp)
results_df = results_df.append({'Regression': 'LR Lasso', 'Avg Trainin
g Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Tra
ining Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss_
mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMS
E': avg_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r
2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  0.055654378618253425
avg_val_loss:  0.07643411120563436
avg_train_loss_mae:  0.16500996393346978
avg_val_loss_mae:  0.19166206932675112
avg_train_loss_root:  0.2358512884603449
avg_val_loss_root:  0.2759467268590462
avg_train_loss_r_squared:  0.9443550359534377
avg_val_loss_r_squared:  0.9234021863827608
```

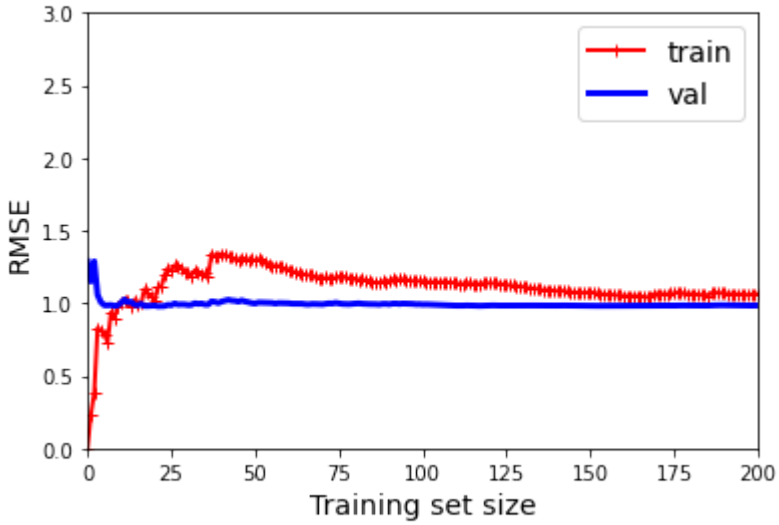


In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



In [ ]:
```
lin_reg = Lasso(alpha = 10, tol=0.000001)
plot_learning_curves(lin_reg, x_trainnp, y_labelsnp)
plt.axis([0, 200, 0, 3])
plt.show()
```

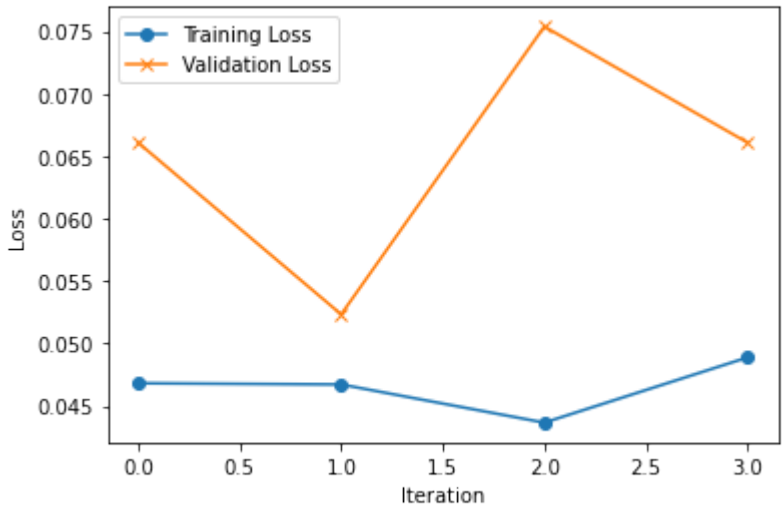
```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinat
e_descent.py:647: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations, check the scale of th
e features or consider increasing regularisation. Duality gap: 0.000e+
00, tolerance: 0.000e+00
  model = cd_fast.enet_coordinate_descent(
```



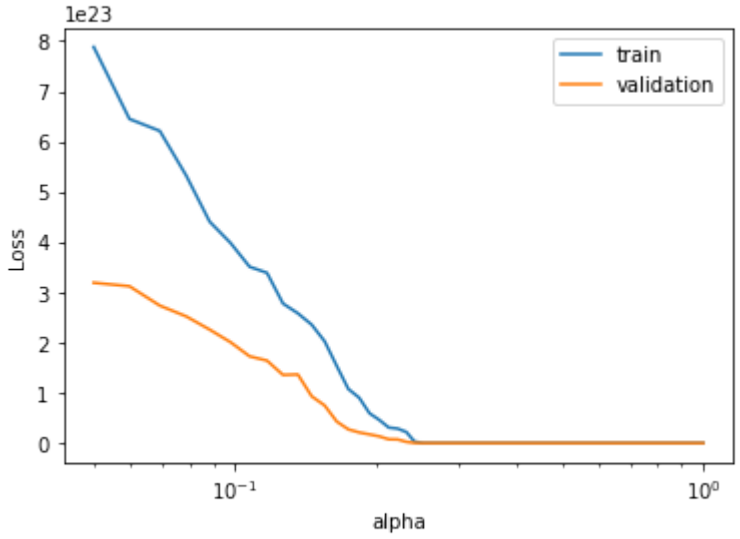
Linear Regression Model with Elasticnet Regularization

In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(ElasticNet(alpha=0.1, l1_ratio=0.5), x_tra
innp, y_labelsnp)
results_df = results_df.append({'Regression': 'LR Elastic Net', 'Avg T
raining Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'A
vg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val
_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Los
s RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_train_l
oss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  0.04653185450283695
avg_val_loss:  0.06497978160321585
avg_train_loss_mae:  0.1441425101004093
avg_val_loss_mae:  0.1690535864624909
avg_train_loss_root:  0.21566885261537255
avg_val_loss_root:  0.2543826552226378
avg_train_loss_r_squared:  0.9534528233025902
avg_val_loss_r_squared:  0.9349876549366097
```

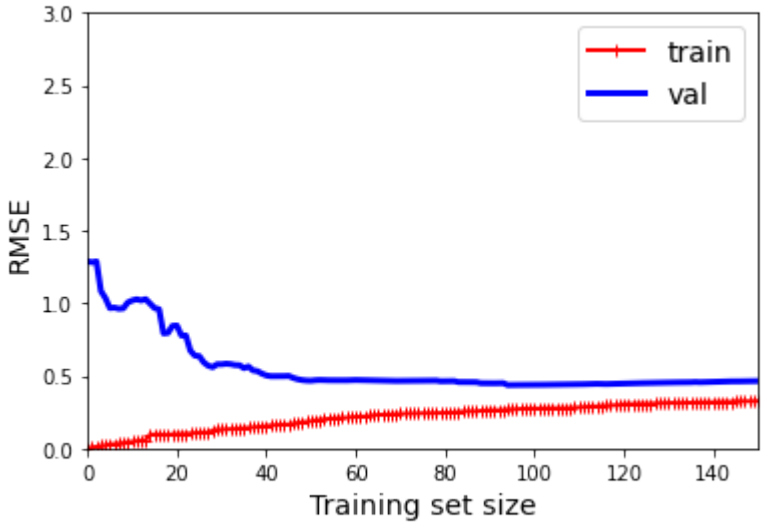


In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



In [ ]:
```
lin_reg = ElasticNet(alpha=0.1, l1_ratio=0.5)
plot_learning_curves(lin_reg, x_trainnp, y_labelsnp)
plt.axis([0, 150, 0, 3])
plt.show()
```

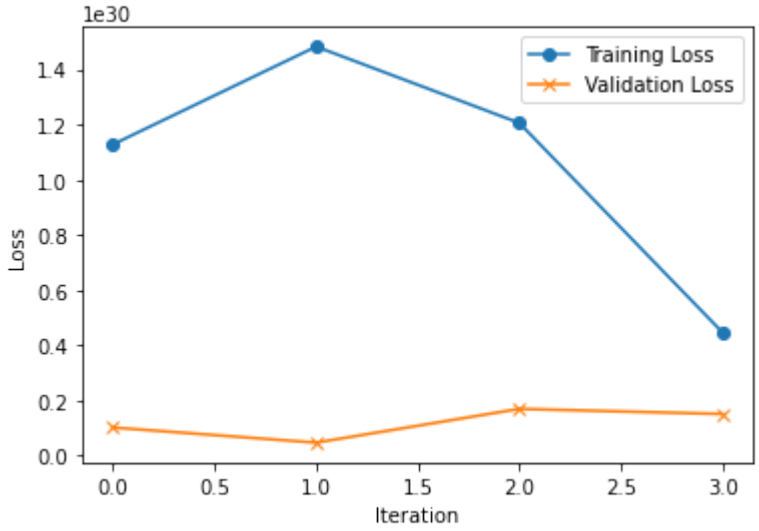
```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinat
e_descent.py:647: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations, check the scale of th
e features or consider increasing regularisation. Duality gap: 0.000e+
00, tolerance: 0.000e+00
  model = cd_fast.enet_coordinate_descent(
```



Stochastic Gradient Descent

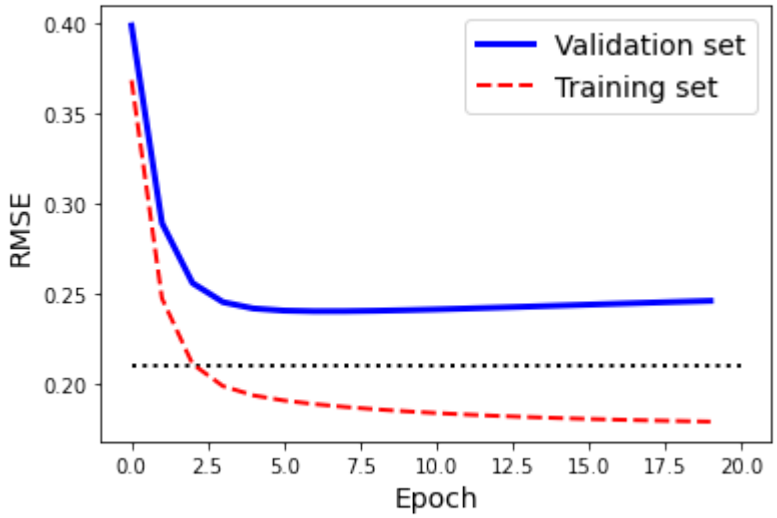
In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(SGDRegressor(max_iter=1000, tol=1e-3, pena
lty=None, eta0=0.1, random_state=42), x_trainnp, y_labelsnp)
results_df = results_df.append({'Regression': 'SGD', 'Avg Training Los
s': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Training
Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss_mae,
'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMSE': av
g_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r2, 'Av
g Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  1.0658089710012361e+30
avg_val_loss:  1.1766201048621915e+29
avg_train_loss_mae:  217731899632827.12
avg_val_loss_mae:  158336511278287.25
avg_train_loss_root:  1011560128681193.2
avg_val_loss_root:  334594821174788.0
avg_train_loss_r_squared:  -1.0678246000211526e+30
avg_val_loss_r_squared:  -1.1667418244007247e+29
```



In [ ]:
```
X_train, X_val, y_train, y_val = train_test_split(x_trainnp, y_labelsn
p, test_size=0.2, random_state=10)

sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,
                       penalty=None , learning_rate="constant", eta0=
0.0005, random_state=42)
minimum_val_error = float("inf")
best_epoch = None
best_model = None
for epoch in range(1000):
    sgd_reg.fit(X_train, y_train)
    y_val_predict = sgd_reg.predict(X_val)
    val_error = mean_squared_error(y_val, y_val_predict)
    if val_error < minimum_val_error:
        minimum_val_error = val_error
        best_epoch = epoch
        best_model = deepcopy(sgd_reg)
```

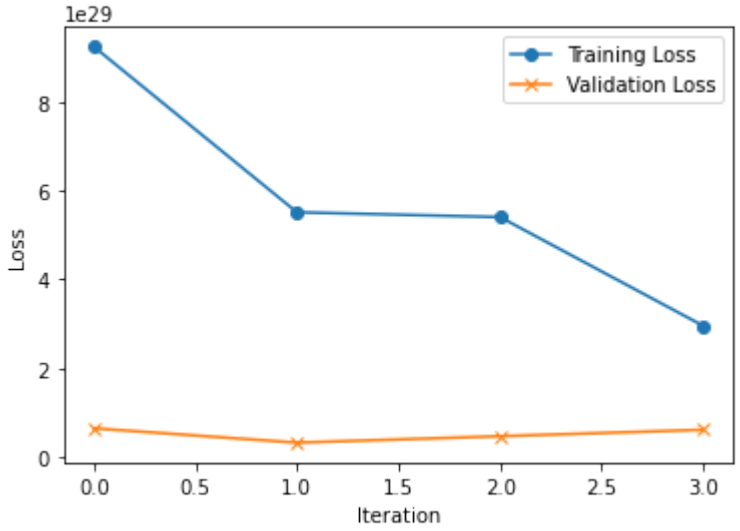
In [ ]:
```
sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,
                       penalty=None , learning_rate="constant", eta0=
0.0005, random_state=42)

n_epochs = 20
train_errors, val_errors = [], []
for epoch in range(n_epochs):
    sgd_reg.fit(X_train, y_train)
    y_train_predict = sgd_reg.predict(X_train)
    y_val_predict = sgd_reg.predict(X_val)
    train_errors.append(mean_squared_error(y_train, y_train_predict))
    val_errors.append(mean_squared_error(y_val, y_val_predict))

best_epoch = np.argmin(val_errors)
best_val_rmse = np.sqrt(val_errors[best_epoch])


best_val_rmse -= 0.03  #
plt.plot([0, n_epochs], [best_val_rmse, best_val_rmse], "k:", linewidt
h=2)
plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="Validation se
t")
plt.plot(np.sqrt(train_errors), "r--", linewidth=2, label="Training se
t")
plt.legend(loc="upper right", fontsize=14)
plt.xlabel("Epoch", fontsize=14)
plt.ylabel("RMSE", fontsize=14)

plt.show()
```



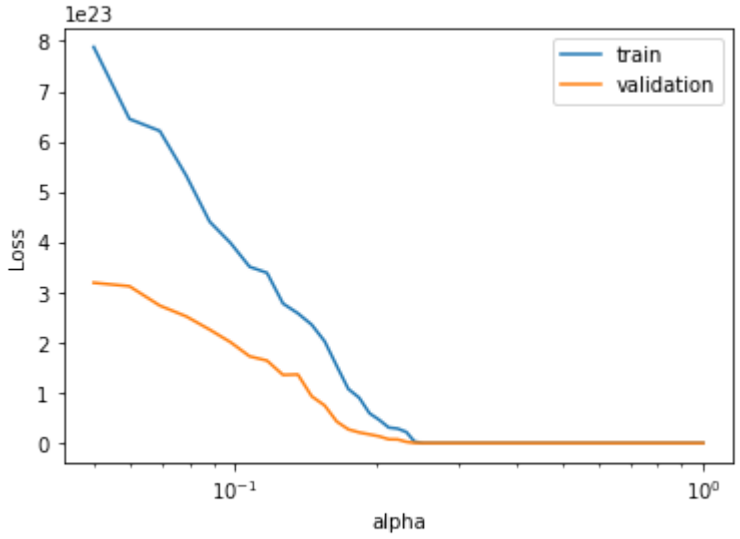
SGD Regression with Ridge Regularization

In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, avg_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_r2 = fit_and_evaluate_model(SGDRegressor(max_iter=1000, tol=1e-3, penalty="l2", eta0=0.1, random_state=42, alpha = 0.1), x_trainnp, y_labelsnp)
results_df = results_df.append({'Regression': 'SGD Ridge', 'Avg Training Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  5.780266860251954e+29
avg_val_loss:  5.155549667563392e+28
avg_train_loss_mae:  188276952825192.56
avg_val_loss_mae:  143947915517379.62
avg_train_loss_root:  745761615268002.9
avg_val_loss_root:  225144956546658.3
avg_train_loss_r_squared:  -5.82254691984482e+29
avg_val_loss_r_squared:  -5.149092887040303e+28
```

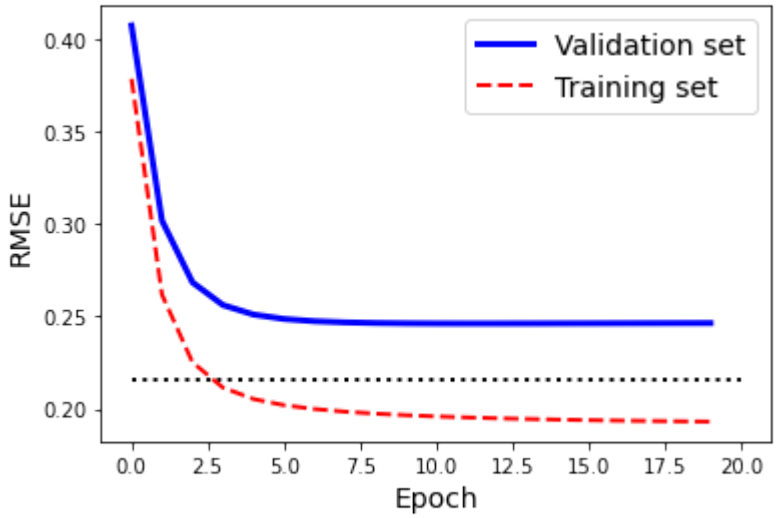


In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



In [ ]:
```
X_train, X_val, y_train, y_val = train_test_split(x_trainnp, y_labelsn
p, test_size=0.2, random_state=10)

sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,
                       penalty='l2',alpha = 0.1 , learning_rate="cons
tant", eta0=0.0005, random_state=42)
minimum_val_error = float("inf")
best_epoch = None
best_model = None
for epoch in range(1000):
    sgd_reg.fit(X_train, y_train)
    y_val_predict = sgd_reg.predict(X_val)
    val_error = mean_squared_error(y_val, y_val_predict)
    if val_error < minimum_val_error:
        minimum_val_error = val_error
        best_epoch = epoch
        best_model = deepcopy(sgd_reg)
```

In [ ]:
```
sgd_reg = SGDRegressor(max_iter=1, tol=-np.infty, warm_start=True,
                       penalty='l2',alpha = 0.1 , learning_rate="cons
tant", eta0=0.0005, random_state=42)

n_epochs = 20
train_errors, val_errors = [], []
for epoch in range(n_epochs):
    sgd_reg.fit(X_train, y_train)
    y_train_predict = sgd_reg.predict(X_train)
    y_val_predict = sgd_reg.predict(X_val)
    train_errors.append(mean_squared_error(y_train, y_train_predict))
    val_errors.append(mean_squared_error(y_val, y_val_predict))

best_epoch = np.argmin(val_errors)
best_val_rmse = np.sqrt(val_errors[best_epoch])


best_val_rmse -= 0.03  # just to make the graph look better
plt.plot([0, n_epochs], [best_val_rmse, best_val_rmse], "k:", linewidt
h=2)
plt.plot(np.sqrt(val_errors), "b-", linewidth=3, label="Validation se
t")
plt.plot(np.sqrt(train_errors), "r--", linewidth=2, label="Training se
t")
plt.legend(loc="upper right", fontsize=14)
plt.xlabel("Epoch", fontsize=14)
plt.ylabel("RMSE", fontsize=14)
#save_fig("early_stopping_plot")
plt.show()
```

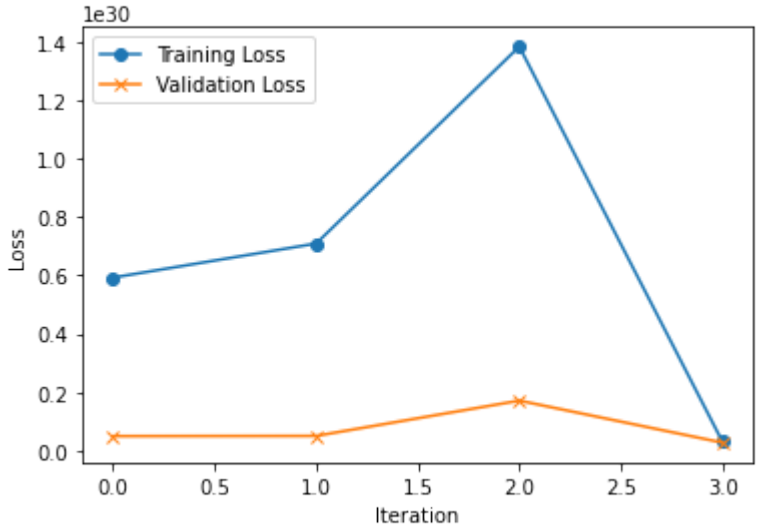


From the graph here it coul dbe seen what the best eteration and epoch coul be near the elbow

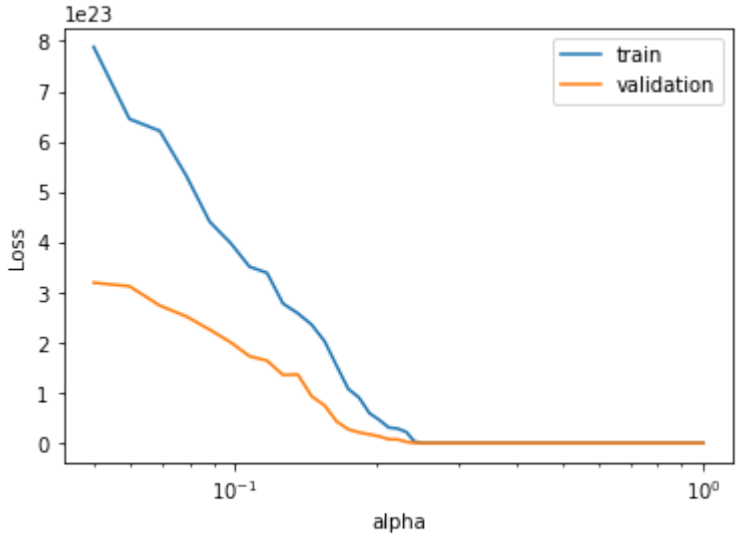
SGD Regression with Lasso Regularization

In [ ]:
```
sgd_lasso = SGDRegressor(max_iter=1000, tol=1e-3, penalty="l1", eta0=
0.1, random_state=42, alpha = 0.1)
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(sgd_lasso, x_trainnp, y_labelsnp)
results_df = results_df.append({'Regression': 'SGD Lasso', 'Avg Traini
ng Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Tr
aining Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss
_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMS
E': avg_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r
2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  6.790604960613917e+29
avg_val_loss:  7.46509457847614e+28
avg_train_loss_mae:  177538883597901.53
avg_val_loss_mae:  130349625949385.06
avg_train_loss_root:  740604129482829.9
avg_val_loss_root:  256597259938301.66
avg_train_loss_r_squared:  -6.863374875101751e+29
avg_val_loss_r_squared:  -7.265520089014521e+28
```



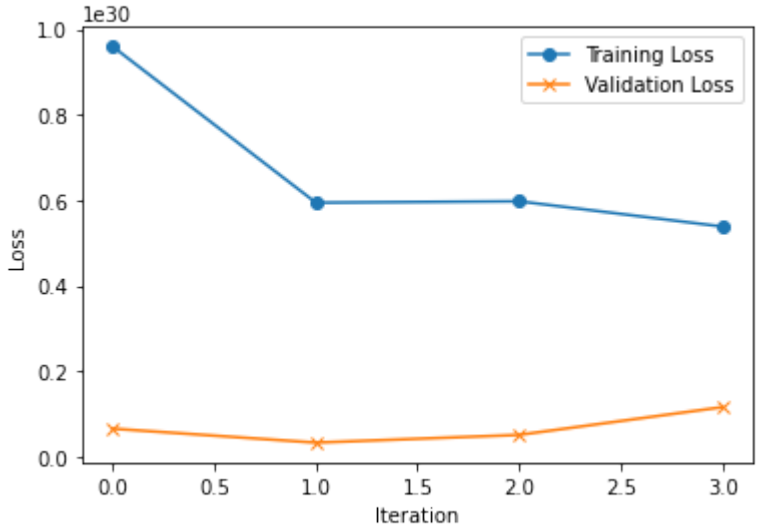
In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



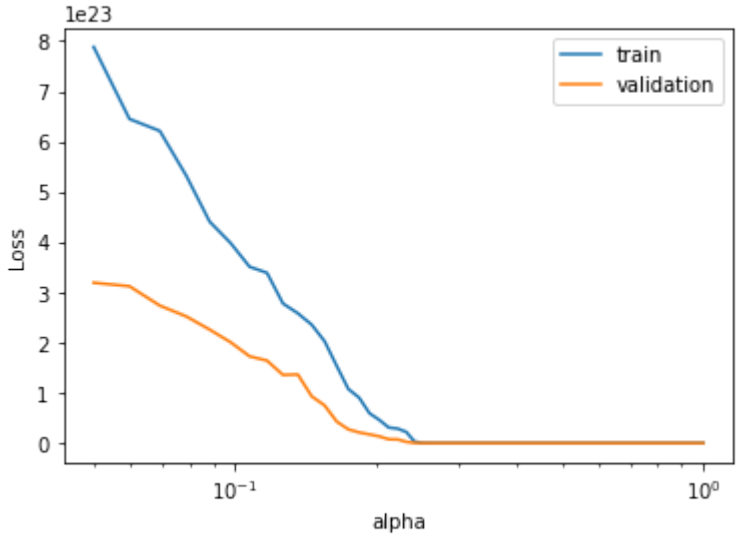
SGD Regression with Elastic Net Regularization

In [ ]:
```
sgd_elastic_net = SGDRegressor(max_iter=1000, tol=1e-3, penalty="elast
icnet", eta0=0.1, random_state=42, alpha = 0.1)
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(sgd_elastic_net, x_trainnp, y_labelsnp)
results_df = results_df.append({'Regression': 'SGD Lasso', 'Avg Traini
ng Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Tr
aining Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss
_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMS
E': avg_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r
2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  6.725058400220633e+29
avg_val_loss:  6.638394898580423e+28
avg_train_loss_mae:  193375608943347.4
avg_val_loss_mae:  145432892860346.56
avg_train_loss_root:  814350530088545.6
avg_val_loss_root:  251013985491559.66
avg_train_loss_r_squared:  -6.758779069927839e+29
avg_val_loss_r_squared:  -6.710818062503446e+28
```



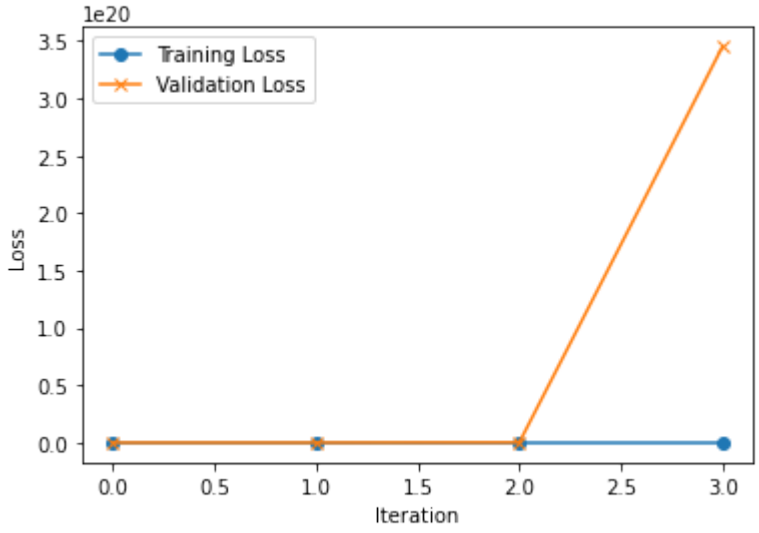
In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



Polynomial Regression with Linear Regression

In [ ]:
```
poly_linear = LinearRegression()
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(poly_linear, x_trainnp, y_labelsnp, degree
=2)
results_df = results_df.append({'Regression': 'Polynomial Regression
(LR)', 'Avg Training Loss': avg_train_loss, 'Avg Validation Loss': avg
_val_loss, 'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss
MAE': avg_val_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root,
'Avg Val Loss RMSE': avg_val_loss_root, 'Avg Training Loss R squared':
avg_train_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_
index=True)
```

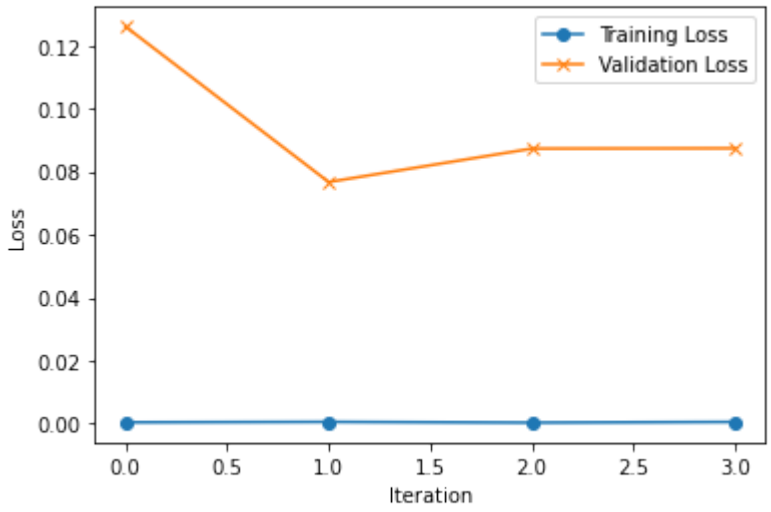
```
avg_train_loss:  0.0002837711165011684
avg_val_loss:  8.632475615176778e+19
avg_train_loss_mae:  0.007185570471666293
avg_val_loss_mae:  754456869.7964728
avg_train_loss_root:  0.016798006019620438
avg_val_loss_root:  4757067885.400153
avg_train_loss_r_squared:  0.999716298126192
avg_val_loss_r_squared:  -9.178719087946877e+19
```



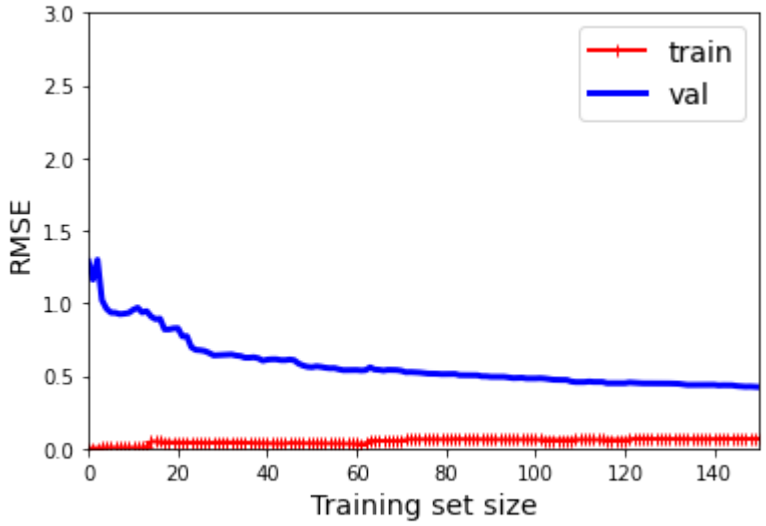
Polynomial Regression with Linear Regression and Ridge Regularization

In [ ]:
```
ridge = Ridge(alpha=0.1)
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(ridge, x_trainnp, y_labelsnp, degree=2)
results_df = results_df.append({'Regression': 'Poly Ridge Regression',
'Avg Training Loss': avg_train_loss, 'Avg Validation Loss': avg_val_lo
ss, 'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': a
vg_val_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg V
al Loss RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_t
rain_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index
=True)
```

```
avg_train_loss:  0.00045595348970808326
avg_val_loss:  0.09436537031640028
avg_train_loss_mae:  0.010243459617789653
avg_val_loss_mae:  0.17025210094904847
avg_train_loss_root:  0.021220082114040084
avg_val_loss_root:  0.30577598013708973
avg_train_loss_r_squared:  0.9995459765374222
avg_val_loss_r_squared:  0.9058396865528563
```



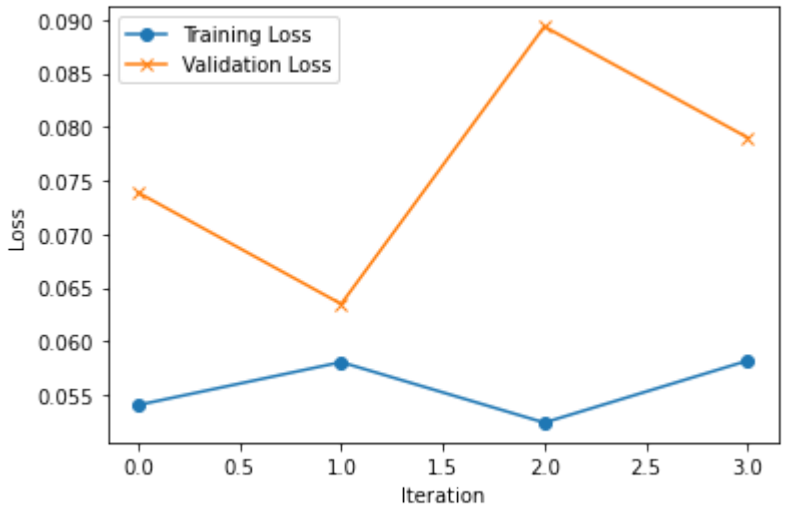
In [ ]:
```
ridge = Ridge(alpha = 1)
plot_learning_curves(ridge, x_trainnp, y_labelsnp)
plt.axis([0, 150, 0, 3])
plt.show()
```



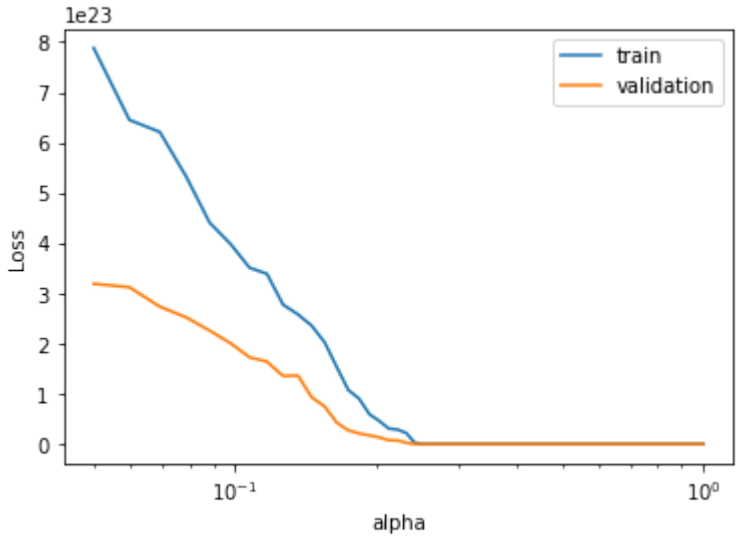
Polynomial Regression with Linear Regression and Lasso Regularization

In [ ]:
```
lasso = Lasso(alpha=0.1)
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(lasso, x_trainnp, y_labelsnp, degree=2)
results_df = results_df.append({'Regression': 'Poly Lasso Regression',
'Avg Training Loss': avg_train_loss, 'Avg Validation Loss': avg_val_lo
ss, 'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': a
vg_val_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg V
al Loss RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_t
rain_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index
=True)
```

```
avg_train_loss:  0.055654378618253425
avg_val_loss:  0.07643411120563436
avg_train_loss_mae:  0.16500996393346978
avg_val_loss_mae:  0.19166206932675112
avg_train_loss_root:  0.2358512884603449
avg_val_loss_root:  0.2759467268590462
avg_train_loss_r_squared:  0.9443550359534377
avg_val_loss_r_squared:  0.9234021863827608
```

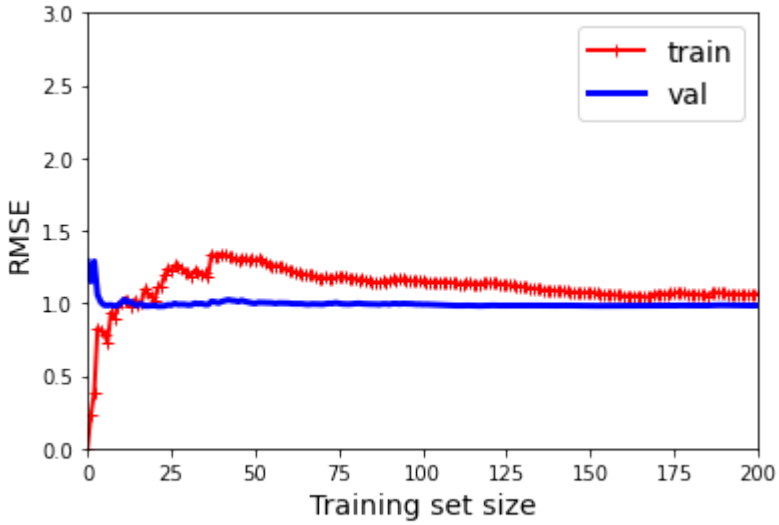


In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



In [ ]:
```
lasso = Lasso(alpha = 10, tol=0.000001)
plot_learning_curves(lasso, x_trainnp, y_labelsnp)
plt.axis([0, 200, 0, 3])
plt.show()
```

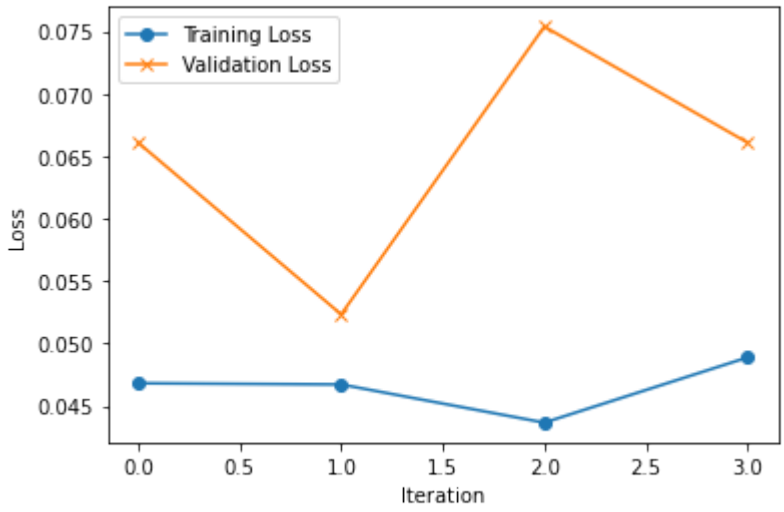
```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinat
e_descent.py:647: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations, check the scale of th
e features or consider increasing regularisation. Duality gap: 0.000e+
00, tolerance: 0.000e+00
  model = cd_fast.enet_coordinate_descent(
```



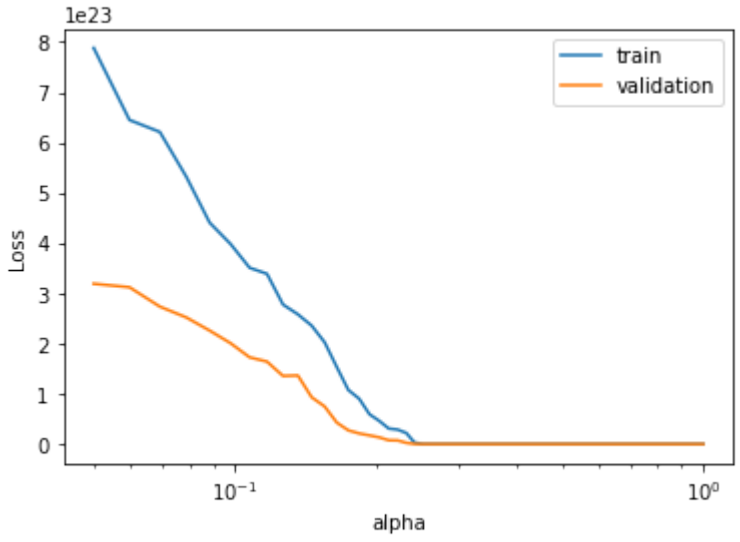
Polynomial Regression with Linear Regression and Elastic Net

In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(ElasticNet(alpha=0.1, l1_ratio=0.5), x_tra
innp, y_labelsnp, degree=2)
results_df = results_df.append({'Regression': 'Poly Elastic Net', 'Avg
Training Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss,
'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_v
al_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val L
oss RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_train
_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=Tru
e)
```

```
avg_train_loss:  0.04653185450283695
avg_val_loss:  0.06497978160321585
avg_train_loss_mae:  0.1441425101004093
avg_val_loss_mae:  0.1690535864624909
avg_train_loss_root:  0.21566885261537255
avg_val_loss_root:  0.2543826552226378
avg_train_loss_r_squared:  0.9534528233025902
avg_val_loss_r_squared:  0.9349876549366097
```

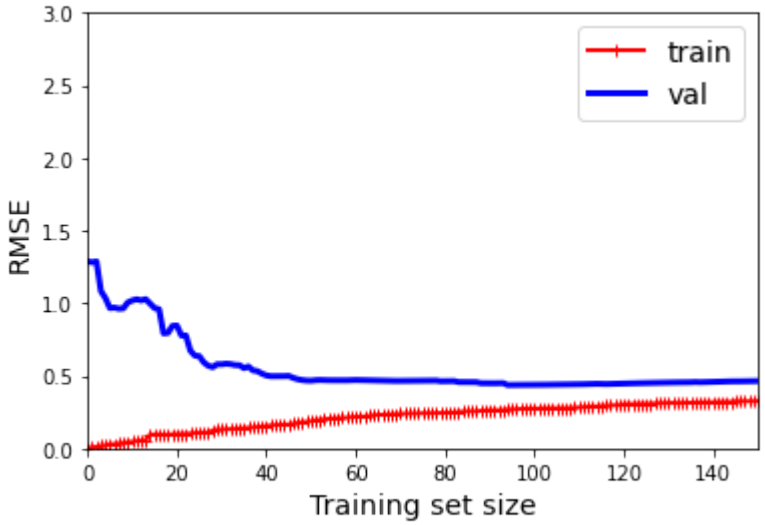


In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



In [ ]:
```
elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
plot_learning_curves(elastic_net, x_trainnp, y_labelsnp)
plt.axis([0, 150, 0, 3])
plt.show()
```

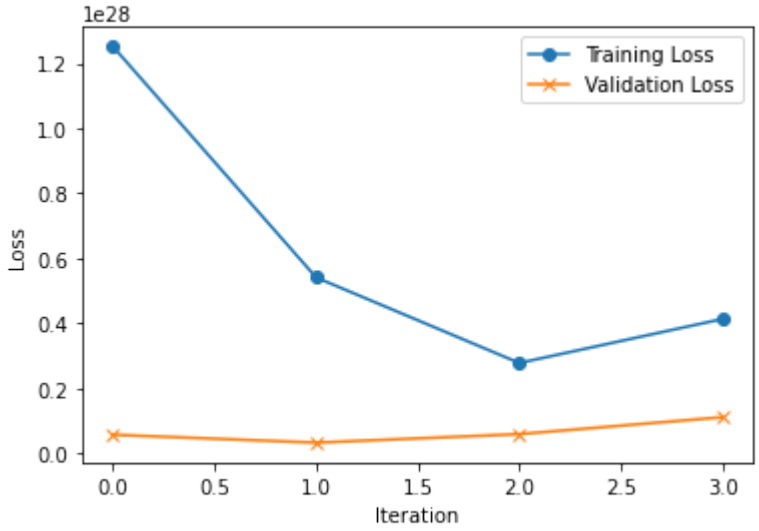
```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinat
e_descent.py:647: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations, check the scale of th
e features or consider increasing regularisation. Duality gap: 0.000e+
00, tolerance: 0.000e+00
  model = cd_fast.enet_coordinate_descent(
```



Polynomial Regession with SGD

In [ ]:
```
SGD = SGDRegressor(alpha=0.5)
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(SGD, x_trainnp, y_labelsnp, degree=2)
results_df = results_df.append({'Regression': 'SGD', 'Avg Training Los
s': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Training
Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss_mae,
'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMSE': av
g_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r2, 'Av
g Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

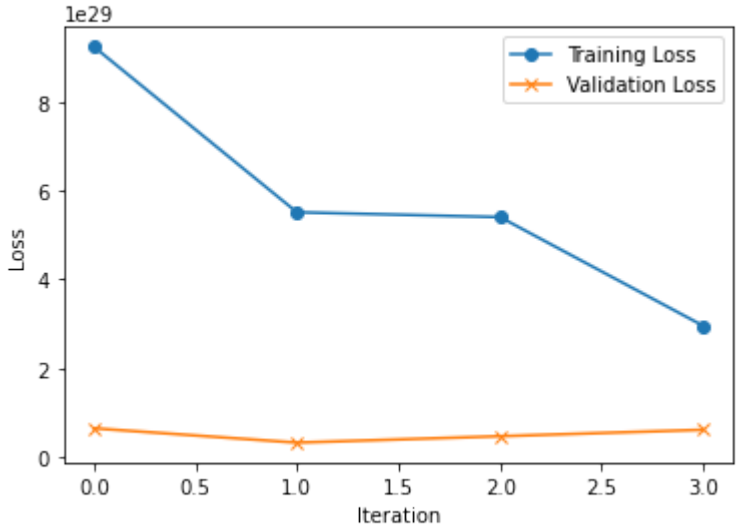
```
avg_train_loss:  6.204469352129446e+27
avg_val_loss:  6.42386713919952e+26
avg_train_loss_mae:  19553656621392.336
avg_val_loss_mae:  14962413710754.371
avg_train_loss_root:  75568373804188.55
avg_val_loss_root:  24745182252720.484
avg_train_loss_r_squared:  -6.248189617839002e+27
avg_val_loss_r_squared:  -6.488427960842539e+26
```



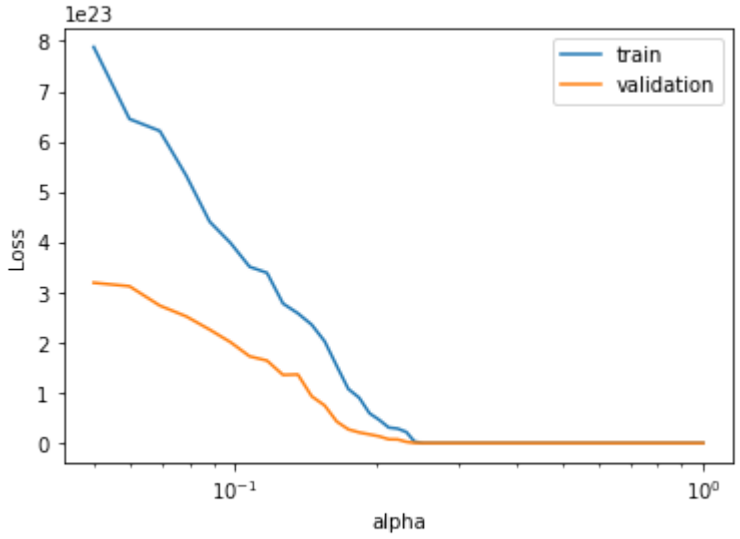
Polynomial Regression with SGD and Ridge Regularization

In [ ]:
```
SGD = SGDRegressor(alpha=0.5)
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, av
g_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_
r2 = fit_and_evaluate_model(SGDRegressor(max_iter=1000, tol=1e-3, pena
lty="l2", eta0=0.1, random_state=42, alpha = 0.1), x_trainnp, y_labels
np, degree=2)
results_df = results_df.append({'Regression': 'SGD Ridge Regression',
'Avg Training Loss': avg_train_loss, 'Avg Validation Loss': avg_val_lo
ss, 'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': a
vg_val_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg V
al Loss RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_t
rain_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index
=True)
```

```
avg_train_loss:  5.780266860251954e+29
avg_val_loss:  5.155549667563392e+28
avg_train_loss_mae:  188276952825192.56
avg_val_loss_mae:  143947915517379.62
avg_train_loss_root:  745761615268002.9
avg_val_loss_root:  225144956546658.3
avg_train_loss_r_squared:  -5.82254691984482e+29
avg_val_loss_r_squared:  -5.149092887040303e+28
```



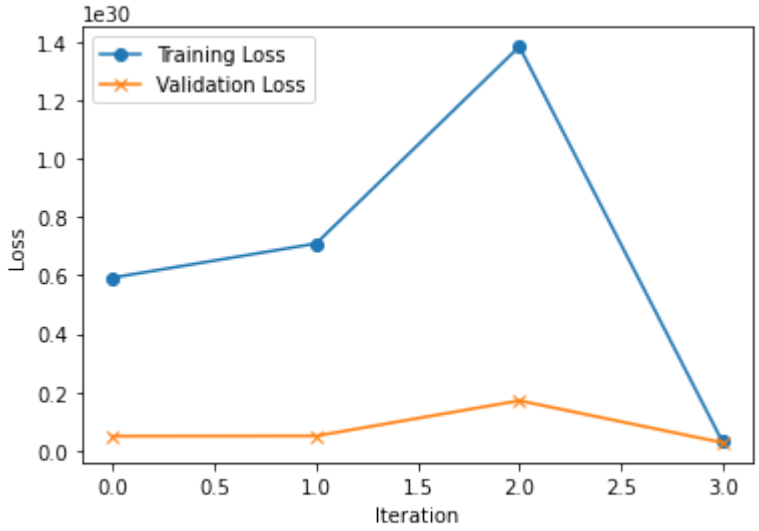
In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



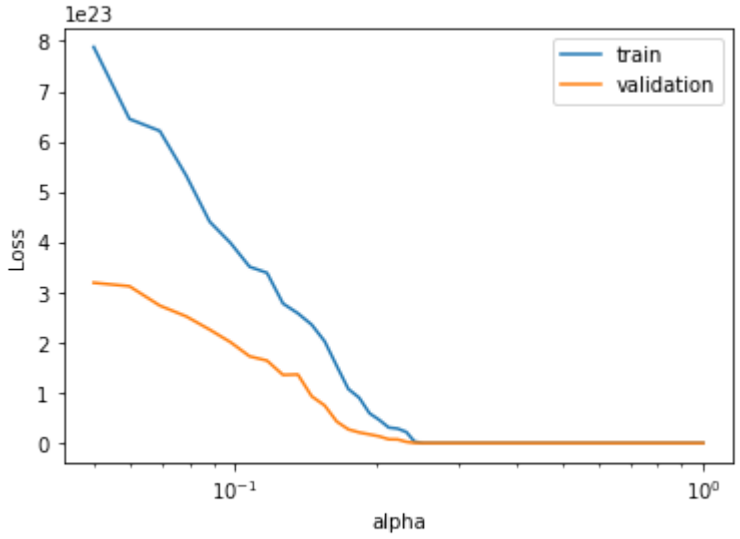
Polynomial Regression with SGD and Lasso Regularization

In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, avg_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_r2 = fit_and_evaluate_model(SGDRegressor(max_iter=1000, tol=1e-3, penalty="l1", eta0=0.1, random_state=42, alpha = 0.1), x_trainnp, y_labelsnp, degree=2)
results_df = results_df.append({'Regression': 'Poly SGD Lasso Regression', 'Avg Training Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  6.790604960613917e+29
avg_val_loss:  7.46509457847614e+28
avg_train_loss_mae:  177538883597901.53
avg_val_loss_mae:  130349625949385.06
avg_train_loss_root:  740604129482829.9
avg_val_loss_root:  256597259938301.66
avg_train_loss_r_squared:  -6.863374875101751e+29
avg_val_loss_r_squared:  -7.265520089014521e+28
```

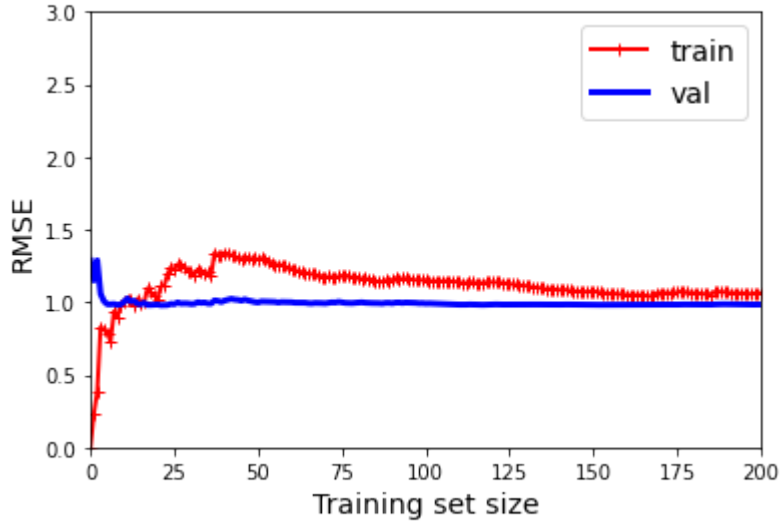


In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



In [ ]:
```
lasso = Lasso(alpha = 10, tol=0.000001)
plot_learning_curves(lasso, x_trainnp, y_labelsnp)
plt.axis([0, 200, 0, 3])
plt.show()
```

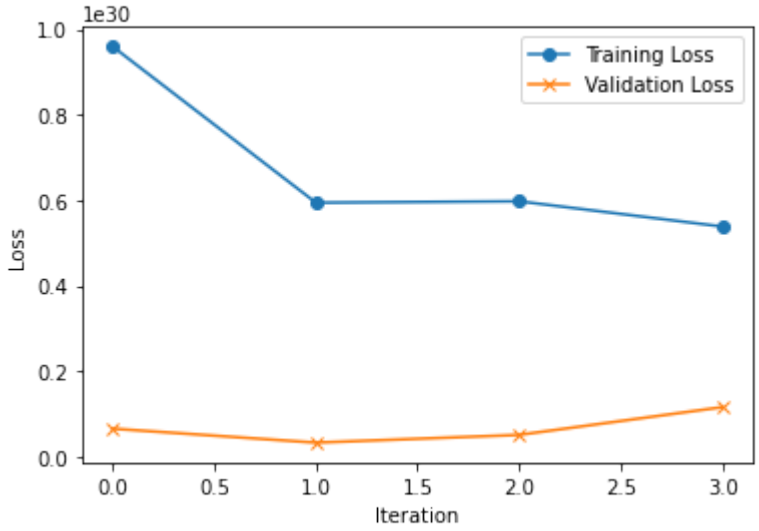
```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_coordinat
e_descent.py:647: ConvergenceWarning: Objective did not converge. You
might want to increase the number of iterations, check the scale of th
e features or consider increasing regularisation. Duality gap: 0.000e+
00, tolerance: 0.000e+00
  model = cd_fast.enet_coordinate_descent(
```



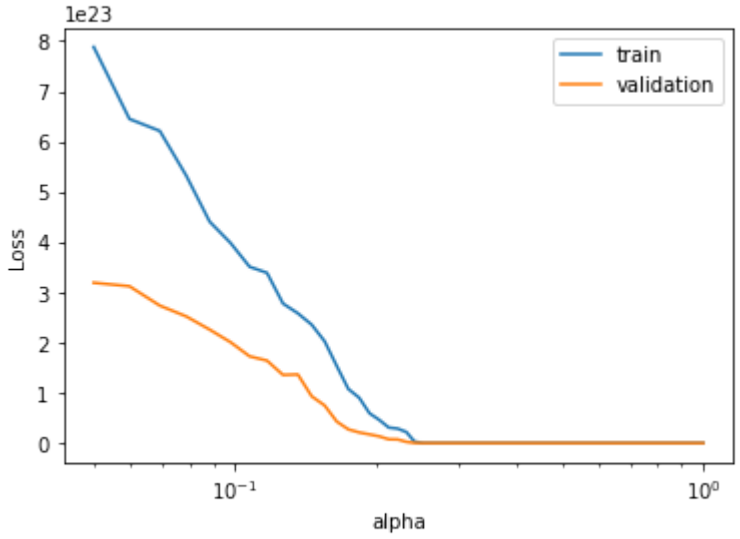
Polynomial Regression with SGD and Elastic Net

In [ ]:
```
avg_train_loss, avg_val_loss, avg_train_loss_mae, avg_val_loss_mae, avg_train_loss_root, avg_val_loss_root, avg_train_loss_r2, avg_val_loss_r2 = fit_and_evaluate_model(SGDRegressor(max_iter=1000, tol=1e-3, penalty="elasticnet", eta0=0.1, random_state=42, alpha = 0.1), x_trainnp, y_labelsnp, degree=2)
results_df = results_df.append({'Regression': 'Poly SGD Elastic Net', 'Avg Training Loss': avg_train_loss, 'Avg Validation Loss': avg_val_loss, 'Avg Training Loss MAE': avg_train_loss_mae, 'Avg Val Loss MAE': avg_val_loss_mae, 'Avg Training Loss RMSE': avg_train_loss_root, 'Avg Val Loss RMSE': avg_val_loss_root, 'Avg Training Loss R squared': avg_train_loss_r2, 'Avg Val Loss R squared': avg_val_loss_r2}, ignore_index=True)
```

```
avg_train_loss:  6.725058400220633e+29
avg_val_loss:  6.638394898580423e+28
avg_train_loss_mae:  193375608943347.4
avg_val_loss_mae:  145432892860346.56
avg_train_loss_root:  814350530088545.6
avg_val_loss_root:  257613985491559.66
avg_train_loss_r_squared:  -6.758779069927839e+29
avg_val_loss_r_squared:  -6.710818062503446e+28
```



In [ ]:
```
#iterating alpha over several vlaues
alpha_iterate(x_trainnp, y_labelsnp)
```



Storing the losses in a dataframe

In [ ]:
```
results_df
```

Out[ ]:

	Regression	Avg Training Loss	Avg Validation Loss	Avg Training Loss MAE	Avg Val Loss MAE	Avg Training Loss RMSE	Avg Val R
0	LR	2.837711e-04	8.632476e+19	7.185570e-03	7.544569e+08	1.679801e-02	4.757068
1	LR Ridge	4.559535e-04	9.436537e-02	1.024346e-02	1.702521e-01	2.122008e-02	3.057760
2	LR Lasso	5.565438e-02	7.643411e-02	1.650100e-01	1.916621e-01	2.358513e-01	2.759467
3	LR Elastic Net	4.653185e-02	6.497978e-02	1.441425e-01	1.690536e-01	2.156689e-01	2.543827
4	SGD	1.065809e+30	1.176620e+29	2.177319e+14	1.583365e+14	1.011560e+15	3.345948
5	SGD Ridge	5.780267e+29	5.155550e+28	1.882770e+14	1.439479e+14	7.457616e+14	2.251450
6	SGD Lasso	6.790605e+29	7.465095e+28	1.775389e+14	1.303496e+14	7.406041e+14	2.565973
7	SGD Lasso	6.725058e+29	6.638395e+28	1.933756e+14	1.454329e+14	8.143505e+14	2.510140
8	Polynomial Regression (LR)	2.837711e-04	8.632476e+19	7.185570e-03	7.544569e+08	1.679801e-02	4.757068
9	Poly Ridge Regression	4.559535e-04	9.436537e-02	1.024346e-02	1.702521e-01	2.122008e-02	3.057760
10	Poly Lasso Regression	5.565438e-02	7.643411e-02	1.650100e-01	1.916621e-01	2.358513e-01	2.759467
11	Poly Elastic Net	4.653185e-02	6.497978e-02	1.441425e-01	1.690536e-01	2.156689e-01	2.543827
12	SGD	6.204469e+27	6.423867e+26	1.955366e+13	1.496241e+13	7.556837e+13	2.474518
13	SGD Ridge Regression	5.780267e+29	5.155550e+28	1.882770e+14	1.439479e+14	7.457616e+14	2.251450
14	Poly SGD Lasso Regression	6.790605e+29	7.465095e+28	1.775389e+14	1.303496e+14	7.406041e+14	2.565973
15	Poly SGD Elastic Net	6.725058e+29	6.638395e+28	1.933756e+14	1.454329e+14	8.143505e+14	2.510140

Predictions:

In [ ]:
```
min_loss_rmse = results_df['Avg Val Loss RMSE'].min()
print ('Minimum los for RMSE', min_loss_rmse)
# we came to know the least error is for linear regression with elasti
c net
```

```
Minimum los for RMSE 0.2543826552226378
```

In [ ]:
```
#testing:
test_dataset = strat_test_set.copy()
test_dataset = test_dataset.drop(["perp_cat"], axis = 1)
```

In [ ]:
```
test_dataset.head()
```

Out[ ]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negative affect	Countr
1742	9.222	0.863	65.10	0.624	-0.135	0.732	0.725	0.249	
128	8.274	0.649	63.30	0.875	-0.089	0.688	0.560	0.235	
361	9.384	0.893	65.70	0.816	-0.050	0.815	0.831	0.265	
1807	9.428	0.885	65.20	0.784	0.126	0.946	0.688	0.285	
1051	9.881	0.871	65.12	0.844	0.089	0.799	0.775	0.162	

5 rows × 174 columns

In [ ]:
```
y_labels = test_dataset.iloc[:, 2]
y_labelsnp_test = y_labels.values
y_labelsnp_test.shape
```

Out[ ]:
```
(390,)
```

In [ ]:
```
x_test = test_dataset.drop(['Healthy life expectancy at birth'], axis=
1)
x_testnp = x_test.values
x_testnp.shape
```

Out[ ]:
```
(390, 173)
```

In [ ]:
```
y_train = strat_train_set.iloc[:,2]
```

In [ ]:
```
X_trainn = strat_train_set.drop(["perp_cat"],  axis = 1)
X_trainn = X_trainn.drop(["Healthy life expectancy at birth"], axis =
1)
```

In [ ]:
```
strat_train_set.head()
```

Out[ ]:

	Log GDP per capita	Social support	Healthy life expectancy at birth	Freedom to make life choices	Generosity	Perceptions of corruption	Positive affect	Negative affect	Counti
1519	7.417	0.650	50.8	0.716	0.095	0.856	0.552	0.466	
768	9.449	0.572	65.1	0.780	0.176	0.699	0.645	0.520	
49	10.032	0.900	68.8	0.846	-0.211	0.855	0.820	0.321	
1473	10.779	0.867	64.8	0.560	-0.120	0.802	0.715	0.225	
458	10.878	0.960	71.5	0.941	0.222	0.191	0.829	0.218	

5 rows × 175 columns

In [ ]:
```
#predictions:
model = SGDRegressor(max_iter=1000, tol=1e-3, penalty="elasticnet", et
a0=0.1, random_state=42, alpha = 0.1)
model.fit(X_trainn,y_train)
y_pred = model.predict(x_testnp)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarnin
g: X does not have valid feature names, but SGDRegressor was fitted wi
th feature names
  warnings.warn(
```

In [ ]:
```
print(model.score(x_testnp, y_labelsnp_test))
```

```
0.33403441788270694

/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarnin
g: X does not have valid feature names, but SGDRegressor was fitted wi
th feature names
  warnings.warn(
```

In [ ]:
```
y_pred[3]
```

Out[ ]:
```
59.29868242616972
```

In [ ]:
```
y_labelsnp_test[3]
```

Out[ ]:
```
65.2
```

In [ ]:
```
rmse = mean_squared_error(y_labelsnp_test, y_pred, squared= False)
rmse
```

Out[ ]:
```
5.867317081575426
```

In [ ]:
```

```