course-work

April 29, 2023

1 Course work

0. Import the libraries that you will need

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

1. Get the data - in the cell below run: **Note** you only need to run this command the first time you do the exercise. If you save and go away and come back, then can skip straight to step 2.

```
!python get-my-data.py
```

2. Read in the csv:

```
df = pd.read_csv('coursework-data.csv')
```

3. Perform some exploratory data analysis to clean up the dataset. The code needed for this part is found in the first set of exercises that you did.
   - Remove outliers
   - If any pairs of variables are highly correlated, remove one of the pair - highly correlated in this case > 0.99
4. Fit a baseline model, linear regression to map the control parameters (all parameters *except* `gllbsc_gap`) to the dependent parameter `gllbsc_gap`. Summarise its performance.

To set up the data use:

```
x = df.loc[:, df.columns != "gllbsc_gap"].values
y = df.loc[:, df.columns == "gllbsc_gap"].values
```

The rest of the code you need for this found in the second set of exercises that you did.

- From looking at the linear regression model, which features have the greatest influence on the band gap?

5. Develop a gradient boosted regressor to the same data. Summarise its performance.

1.1 Important notes

1.1.1 Submitting the coursework

When you are finished with the coursework - use `File > Save and Export Notebook As > pdf` to download a pdf of the completed notebook. Submit this pdf *via* the portal on QMplus.

The deadline for submission is Friday 28th April at 16:00.

1.1.2 Text explanations

Please please please add text to explain what you are doing in the code. Adding text boxes is easy, just add a new cell as normal then change the type to `Markdown` with the dropdown menu at the top of the cells. Adding text will make sure that markers can give you proper grades even if you make a small slip in your code. If you have no text explanation and still have a small slip, you will likely get no marks!

1.1.3 Datasets

All of your datasets are generated randomly. So do not expect the same answers as your friends. If you compare answers and find that you have something very different, do not worry.

1.1.4 Warnings from the code

Don't worry if the code throws some warnings sometimes. If it keeps running then it is fine. Warnings usually just alert you to future planned changes in the code you are using.

1.1.5 Long run times

There is a certain part of the exercise where a grid search is required. It could take quite a long time with this code. I have tested it and it took about 15 minutes for a 10-fold cross validation on a 5x5 gridsearch. Dont worry if it seems to be running for a long time, that's okay.

[1]:
```
#data manipulation
import pandas as pd
import numpy as np

#data visualization
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import rcParams
```

[2]:
```
#Getting the data
!python get-my-data.py
```

```
File already exists no more to see here.
```

[3]:
```
#Reading in the csv file
df = pd.read_csv('coursework-data.csv')
```

[4]:
```
#Getting the shape of the character in terms of (examples, characteristics)
df.shape
```

[4]:
```
(749, 16)
```

[6]:
```
#Using describe to get a summary of the data
df.describe()
```

[6]:

```
              GS mean          GS dev  HOMO_energy      Weight dev    Eneg dev  \
count  749.000000      749.000000   749.000000      749.000000  749.000000
mean    20.129941      145.794327    -0.318680      168.245383    0.880011
std      8.758476     3653.481944     0.041349     3652.708662    0.202963
min      8.332000        0.000000    -0.338381        0.000000    0.000000
25%     14.012447        6.064600    -0.338381       18.529332    0.759008
50%     17.530000        9.299354    -0.338381       32.680587    0.899592
75%     23.485833       16.631250    -0.320380       48.093618    1.036694
max     80.211667   100000.000000    -0.144272   100000.000000    1.325000

             Spg dev  gllbsc_gap  NValence mean  LUMO values  CovRad dev  \
count     749.000000  749.000000     749.000000   749.000000  749.000000
mean      217.080871    5.308134       6.900038   481.708041   41.414244
std      3650.926077    2.317242       2.628995   223.128903   12.558078
min         0.000000    0.144493       2.666667    54.800000    0.000000
25%        72.592593    3.652201       4.923077   290.248000   32.395062
50%        89.306122    5.346850       6.300000   445.613333   40.592593
75%        99.750000    6.903045       8.000000   656.466667   50.000000
max    100000.000000   11.560321      20.000000  1220.600000   89.000000

        MeltT mean  Number dev  Periodic nature  Mendeleev dev  NdValence dev  \
count   749.000000  749.000000       749.000000     749.000000     749.000000
mean    481.708041   14.144738        -0.318680      22.488916       2.220428
std     223.128903    7.934567         0.041349      12.110542       1.677656
min      54.800000    0.000000        -0.338381       0.000000       0.000000
25%     290.248000    7.836735        -0.338381      11.555556       0.555556
50%     445.613333   13.500000        -0.338381      24.612245       2.370370
75%     656.466667   19.222222        -0.320380      32.520000       3.750000
max    1220.600000   36.122449        -0.144272      44.571429       5.000000

         MeltT dev
count   749.000000
mean    493.400798
std     285.198279
min       0.000000
25%     255.460408
50%     439.656198
75%     708.573750
max    1372.535000
```

[7]:

```
#Use info to get full list of characteristics and their data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 749 entries, 0 to 748
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
```

```
---  ------           --------------  -----
 0   GS mean          749 non-null    float64
 1   GS dev           749 non-null    float64
 2   HOMO_energy      749 non-null    float64
 3   Weight dev       749 non-null    float64
 4   Eneg dev         749 non-null    float64
 5   Spg dev          749 non-null    float64
 6   gllbsc_gap       749 non-null    float64
 7   NValence mean    749 non-null    float64
 8   LUMO values      749 non-null    float64
 9   CovRad dev       749 non-null    float64
 10  MeltT mean       749 non-null    float64
 11  Number dev       749 non-null    float64
 12  Periodic nature  749 non-null    float64
 13  Mendeleev dev    749 non-null    float64
 14  NdValence dev    749 non-null    float64
 15  MeltT dev        749 non-null    float64
dtypes: float64(16)
memory usage: 93.8 KB
```

[8]:
```
##Starting to clean the dataset
#Checking for duplicated rows
df.duplicated().sum()
```

[8]:
```
0
```

[9]:
```
df.nunique()
```

[9]:
```
GS mean            748
GS dev             748
HOMO_energy         30
Weight dev         748
Eneg dev           665
Spg dev            407
gllbsc_gap         749
NValence mean      184
LUMO values        748
CovRad dev         613
MeltT mean         748
Number dev         574
Periodic nature     30
Mendeleev dev      624
NdValence dev      188
MeltT dev          748
dtype: int64
```

[10]:
```
#Working to find and remove the Outliers
df.head()
```

[10]:

	GS mean	GS dev	HOMO_energy	Weight dev	Eneg dev	Spg dev
0	20.010909	12.915702	-0.160771	18.519174	1.090909	105.619835
1	10.555455	2.109752	-0.338381	51.543246	0.643967	72.198347
2	11.680714	3.679592	-0.338381	67.325910	0.791837	88.571429
3	22.894091	17.092314	-0.273634	48.630545	1.029421	102.545455
4	30.719167	21.614167	-0.338381	36.772863	1.002500	92.750000

	gllbsc_gap	NValence mean	LUMO values	CovRad dev	MeltT mean	Number dev
0	3.178134	4.909091	601.531818	50.876033	601.531818	8.528926
1	6.064334	9.272727	731.734545	29.884298	731.734545	20.628099
2	5.143263	9.714286	979.142857	42.448980	979.142857	26.530612
3	3.833288	7.090909	555.856364	53.057851	555.856364	19.834711
4	3.881077	7.000000	337.477500	52.500000	337.477500	14.750000

	Periodic nature	Mendeleev dev	NdValence dev	MeltT dev
0	-0.160771	35.371901	1.652893	596.434711
1	-0.338381	11.107438	2.644628	991.732893
2	-0.338381	15.918367	1.224490	1320.489796
3	-0.273634	29.305785	2.826446	546.606942
4	-0.338381	31.250000	3.750000	283.151250

[11]:
```
df.tail()
```

[11]:

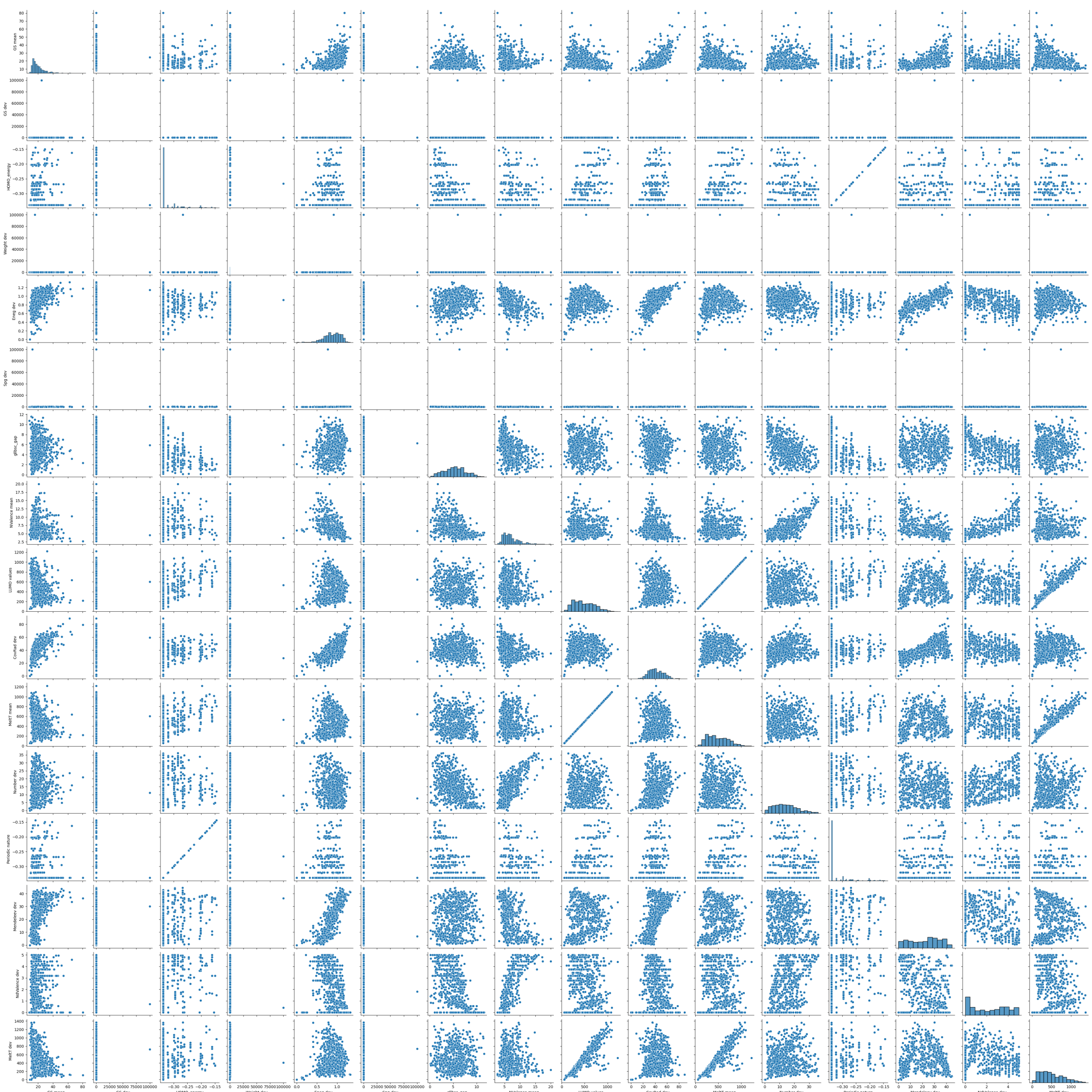
	GS mean	GS dev	HOMO_energy	Weight dev	Eneg dev	Spg dev
744	23.689231	18.415740	-0.338381	45.616776	0.962840	90.792899
745	17.307813	12.787344	-0.338381	34.121112	0.814062	82.031250
746	30.929063	24.495703	-0.197497	53.511675	0.973750	89.906250
747	14.552500	8.815417	-0.338381	18.375224	0.875972	92.458333
748	19.864250	15.972725	-0.338381	41.178064	0.988800	96.600000

	gllbsc_gap	NValence mean	LUMO values	CovRad dev	MeltT mean
744	7.162802	6.461538	432.184615	52.35503	432.184615
745	6.588943	6.250000	589.000000	36.75000	589.000000
746	3.423648	5.250000	825.300000	63.18750	825.300000
747	8.772669	4.666667	662.211667	30.87500	662.211667
748	4.819665	6.900000	811.659500	49.53000	811.659500

	Number dev	Periodic nature	Mendeleev dev	NdValence dev	MeltT dev
744	18.366864	-0.338381	27.479290	2.60355	464.473373
745	13.875000	-0.338381	17.250000	2.18750	667.750000
746	21.500000	-0.197497	34.875000	2.18750	577.875000
747	7.500000	-0.338381	13.222222	0.00000	708.646944
748	16.170000	-0.338381	24.930000	0.76500	1050.770250

[12]:
```
#Graphical examination - Using seaborn to plot scatter graphs of the different
↪variables against each other
sns.pairplot(df)
```

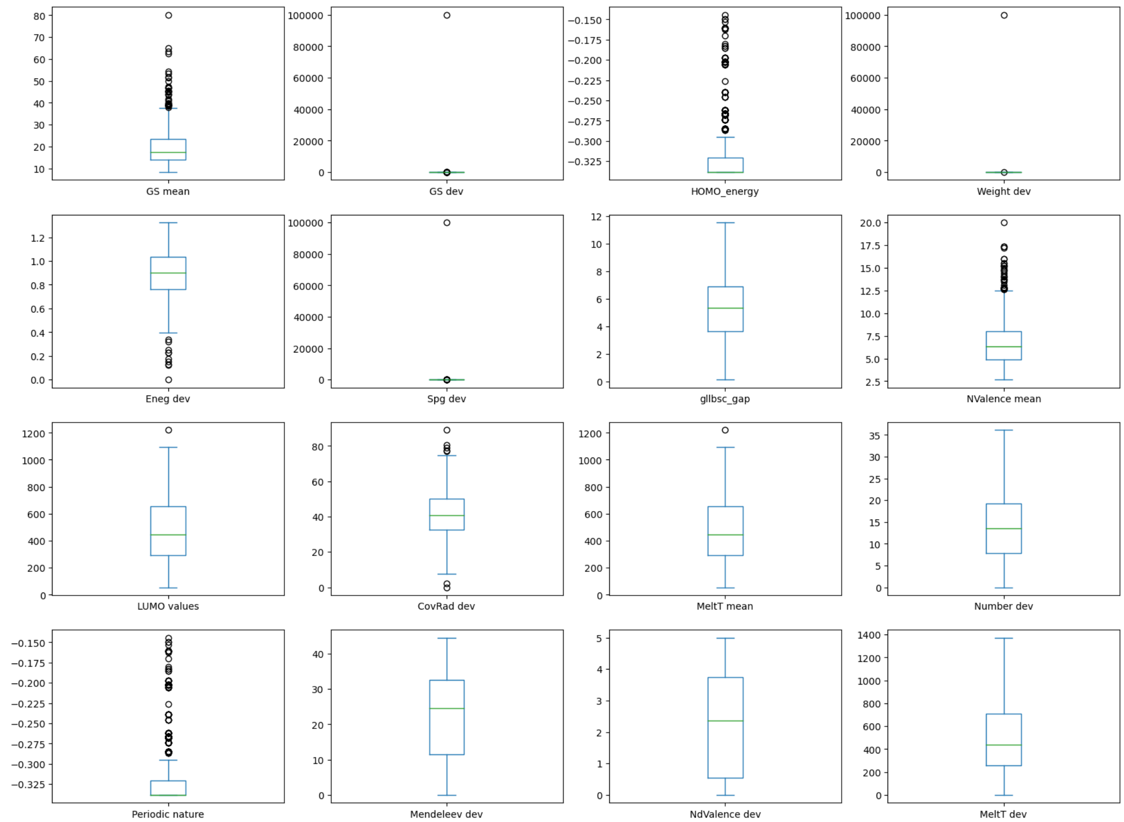
[12]:
```
<seaborn.axisgrid.PairGrid at 0x7fb09a916170>
```



[13]:
```
df.columns
```

[13]:
```
Index(['GS mean', 'GS dev', 'HOMO_energy', 'Weight dev', 'Eneg dev', 'Spg dev',
       'gllbsc_gap', 'NValence mean', 'LUMO values', 'CovRad dev',
       'MeltT mean', 'Number dev', 'Periodic nature', 'Mendeleev dev',
       'NdValence dev', 'MeltT dev'],
      dtype='object')
```

[14]:
```python
#Plotting box plots to look for outliers in the columns

columns = ['GS mean', 'GS dev', 'HOMO_energy', 'Weight dev', 'Eneg dev', 'Spg dev',
       'gllbsc_gap', 'NValence mean', 'LUMO values', 'CovRad dev',
       'MeltT mean', 'Number dev', 'Periodic nature', 'Mendeleev dev',
       'NdValence dev', 'MeltT dev']
df[columns].plot(kind='box',subplots=True,layout=(4,4),figsize=(20,15))
plt.show()
```



[15]:
```python
#Inspecting the individual distributions by finding the Skewness and Kurtosis of each column
column_names = list(df.columns)
print(column_names)

skewness = df[columns].skew()
print(skewness)
```

```
['GS mean', 'GS dev', 'HOMO_energy', 'Weight dev', 'Eneg dev', 'Spg dev',
'gllbsc_gap', 'NValence mean', 'LUMO values', 'CovRad dev', 'MeltT mean',
'Number dev', 'Periodic nature', 'Mendeleev dev', 'NdValence dev', 'MeltT dev']
```

```
GS mean             1.978349
GS dev             27.367619
HOMO_energy         2.313798
Weight dev         27.366534
Eneg dev           -0.677817
Spg dev            27.366438
gllbsc_gap          0.097489
NValence mean       1.334078
LUMO values         0.430720
CovRad dev          0.217036
MeltT mean          0.430720
Number dev          0.495327
Periodic nature     2.313798
Mendeleev dev      -0.177724
NdValence dev       0.075290
MeltT dev           0.628195
dtype: float64
```

[16]:
```
kurtosis = df[columns].kurt()
print(skewness)
```

```
GS mean             1.978349
GS dev             27.367619
HOMO_energy         2.313798
Weight dev         27.366534
Eneg dev           -0.677817
Spg dev            27.366438
gllbsc_gap          0.097489
NValence mean       1.334078
LUMO values         0.430720
CovRad dev          0.217036
MeltT mean          0.430720
Number dev          0.495327
Periodic nature     2.313798
Mendeleev dev      -0.177724
NdValence dev       0.075290
MeltT dev           0.628195
dtype: float64
```

[39]:
```
#Based on the values of Skewness and Kurtosis, it seems like the following␣
↪columns have outliers: GS dev, Weight dev, and Spg dev
#We know this because they are the only columms with high values, of 27, for␣
↪Skewness and Kurtosis
```

[17]:
```
#Repeating the describe function to find the mean of each column
df.describe()
```

[17]:
```
            GS mean          GS dev  HOMO_energy     Weight dev    Eneg dev  \
count  749.000000      749.000000   749.000000     749.000000  749.000000
mean    20.129941      145.794327    -0.318680     168.245383    0.880011
std      8.758476     3653.481944     0.041349    3652.708662    0.202963
min      8.332000        0.000000    -0.338381       0.000000    0.000000
25%     14.012447        6.064600    -0.338381      18.529332    0.759008
50%     17.530000        9.299354    -0.338381      32.680587    0.899592
75%     23.485833       16.631250    -0.320380      48.093618    1.036694
max     80.211667   100000.000000    -0.144272  100000.000000    1.325000

             Spg dev  gllbsc_gap  NValence mean  LUMO values  CovRad dev  \
count     749.000000  749.000000     749.000000   749.000000  749.000000
mean      217.080871    5.308134       6.900038   481.708041   41.414244
std      3650.926077    2.317242       2.628995   223.128903   12.558078
min         0.000000    0.144493       2.666667    54.800000    0.000000
25%        72.592593    3.652201       4.923077   290.248000   32.395062
50%        89.306122    5.346850       6.300000   445.613333   40.592593
75%        99.750000    6.903045       8.000000   656.466667   50.000000
max    100000.000000   11.560321      20.000000  1220.600000   89.000000

        MeltT mean  Number dev  Periodic nature  Mendeleev dev  NdValence dev  \
count   749.000000  749.000000       749.000000     749.000000     749.000000
mean    481.708041   14.144738        -0.318680      22.488916       2.220428
std     223.128903    7.934567         0.041349      12.110542       1.677656
min      54.800000    0.000000        -0.338381       0.000000       0.000000
25%     290.248000    7.836735        -0.338381      11.555556       0.555556
50%     445.613333   13.500000        -0.338381      24.612245       2.370370
75%     656.466667   19.222222        -0.320380      32.520000       3.750000
max    1220.600000   36.122449        -0.144272      44.571429       5.000000

         MeltT dev
count   749.000000
mean    493.400798
std     285.198279
min       0.000000
25%     255.460408
50%     439.656198
75%     708.573750
max    1372.535000
```

[18]:
```python
#Using to mean, I can find out which specific row has an outlier in each␣
↪outlier column mentioned above
df[df['GS dev'] > 145]
```

[18]:
```
       GS mean    GS dev  HOMO_energy  Weight dev  Eneg dev    Spg dev  \
320  24.650556  100000.0    -0.338381   25.497611   1.14449  96.761905
```

```
     gllbsc_gap  NValence mean  LUMO values  CovRad dev  MeltT mean  \
320    5.880432       4.571429   602.081905   59.482993  602.081905

     Number dev  Periodic nature  Mendeleev dev  NdValence dev   MeltT dev
320   11.102041        -0.338381      29.986395       0.725624  726.627664
```

[19]:
```
#Dropping the row with the outlier from GS dev
df = df.drop(320)
```

[20]:
```
df[df['Weight dev'] > 168]
```

[20]:
```
      GS mean    GS dev  HOMO_energy  Weight dev  Eneg dev   Spg dev  \
67  15.888457  1.166008    -0.266297    100000.0   0.90963  24.54321

    gllbsc_gap  NValence mean  LUMO values  CovRad dev  MeltT mean  \
67    5.962006       3.703704   533.268148   27.654321  533.268148

    Number dev  Periodic nature  Mendeleev dev  NdValence dev   MeltT dev
67    9.399177        -0.266297      36.345679       0.411523  408.404719
```

[21]:
```
df = df.drop(67)
```

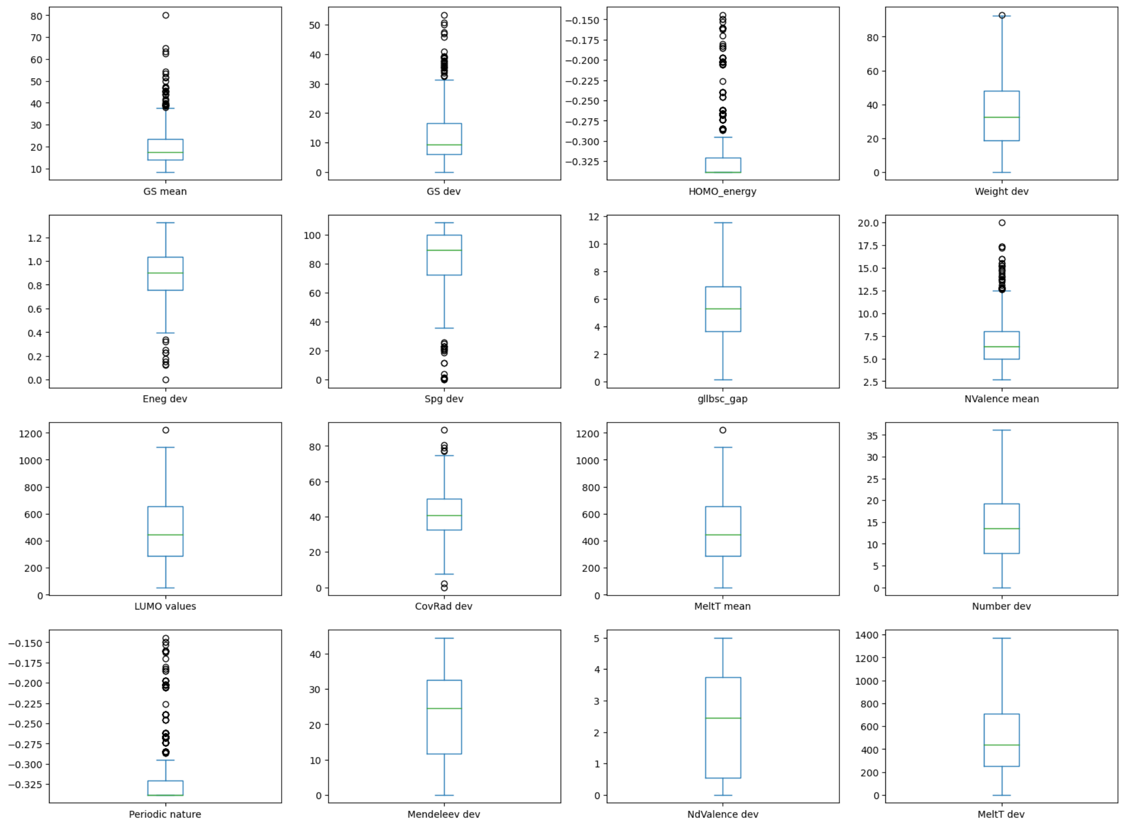
[22]:
```
df[df['Spg dev'] > 217]
```

[22]:
```
    GS mean  GS dev  HOMO_energy  Weight dev  Eneg dev   Spg dev  gllbsc_gap  \
16   12.515   6.229    -0.338381   18.529332    0.7692  100000.0    6.291999

    NValence mean  LUMO values  CovRad dev  MeltT mean  Number dev  \
16            5.8      645.288       22.24     645.288         7.6

    Periodic nature  Mendeleev dev  NdValence dev  MeltT dev
16        -0.338381           6.72            1.8   736.6272
```

[24]:
```
df = df.drop(16)
```

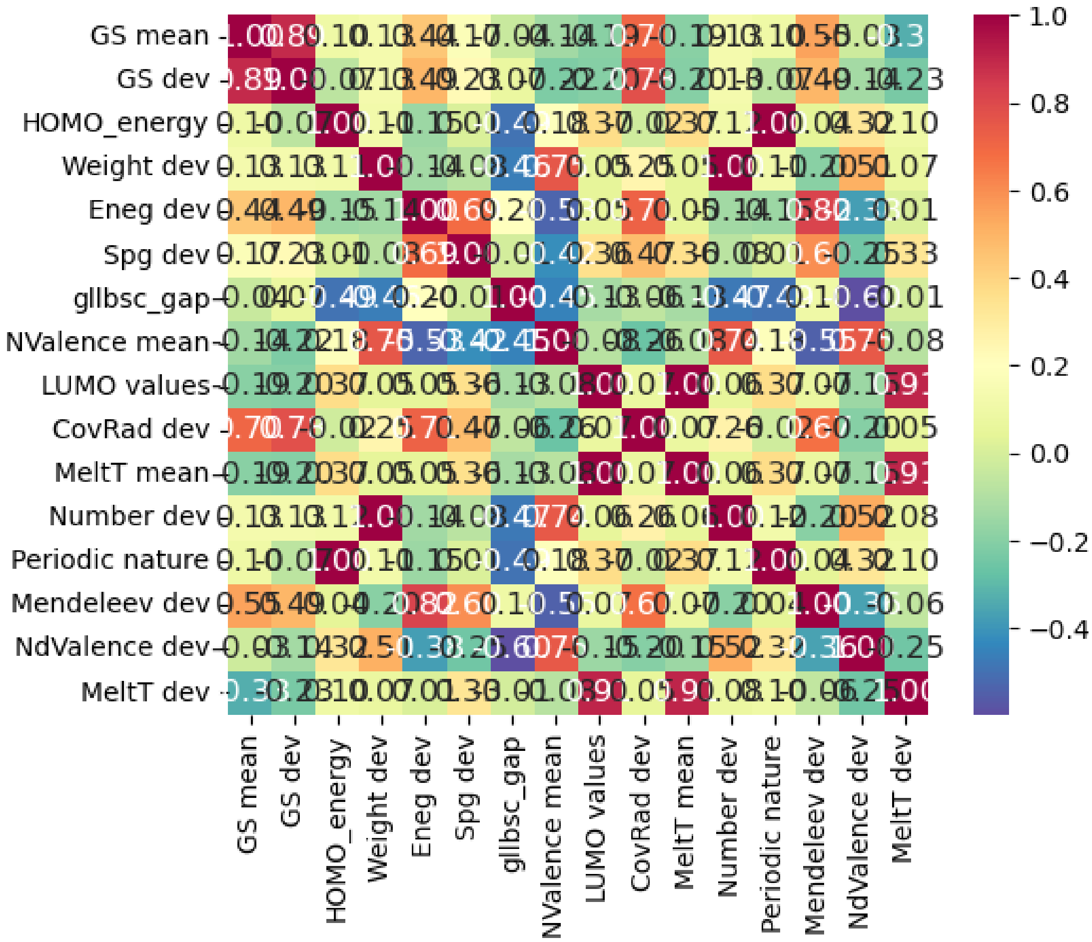
[26]:
```
#Saving the dataset after dropping the outliers
df.to_pickle('coursework-cleaned-data.pickle')
```

[27]:
```
#Plotting box plots again to make sure all outliers have been removed

columns = ['GS mean', 'GS dev', 'HOMO_energy', 'Weight dev', 'Eneg dev', 'Spg␣
 ↪dev',
       'gllbsc_gap', 'NValence mean', 'LUMO values', 'CovRad dev',
       'MeltT mean', 'Number dev', 'Periodic nature', 'Mendeleev dev',
       'NdValence dev', 'MeltT dev']
df[columns].plot(kind='box',subplots=True,layout=(4,4),figsize=(20,15))
plt.show()
```



[28]:
```
#Exploring Correlations in the data
```

[32]:
```
#Now I will obtain Pearsons correlations and make a heatmap. Doing so will
↪allow me to look for any redundant columns

corrmat = df.corr()
corrmat

hm = sns.heatmap(corrmat,
                 cbar=True,
                 annot=True,
                 square=True,
                 fmt='.2f',
                 annot_kws={'size': 12},
                 yticklabels=df.columns,
                 xticklabels=df.columns,
                 cmap="Spectral_r")
plt.show()
```



[34]:
```
#Linear Regression
```

[35]:
```
#First, I will set up x and y and then save the data
x = df.loc[:, df.columns != "gllbsc_gap"].values
y = df.loc[:, df.columns == "gllbsc_gap"].values
df.to_pickle('coursework-regression-train.pickle')
```

[36]:
```
#I will now look at the shape of the dataset
df.values.shape
```

[36]: (746, 16)

[37]:
```
#The number of rows has decreased from 749 to 746 because I removed three rows␣
↪due to the fact that they had outliers
```

[38]:
```
#Scaling the data
#I will now standardize the data using the StandardScaler function
```

```
from sklearn.preprocessing import StandardScaler

scaler_x = StandardScaler()
x = scaler_x.fit_transform(x)
scaler_y = StandardScaler()
y = scaler_y.fit_transform(y.reshape(-1, 1))
```

[39]:
```
#Here, I will use the train_test_split tool from scikit-learn to make an 80:20␣
↪training:test split as shown in the previous exercises.

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0,␣
↪train_size=0.8)
```

[40]:
```
#Setting up a linear regression and fitting the model

#This code fits the data using the LinearRegression function from scikit-learn
from sklearn.linear_model import LinearRegression
# with sklearn
regr = LinearRegression()
regr.fit(x_train, y_train)

print('Intercept: \n', regr.intercept_)
print('Coefficients: \n', regr.coef_)

#Using the results I can determine which feature appears to have the greatest␣
↪influence on the band gap
```

```
Intercept:
 [0.00338702]
Coefficients:
 [[-0.10255972  0.35950801 -0.12733959 -0.43379754  0.40560351 -0.2227987
   0.15962037  0.03401533 -0.43188967  0.03401533  0.22291752 -0.12733959
  -0.06211499 -0.5043008  -0.01283062]]
```

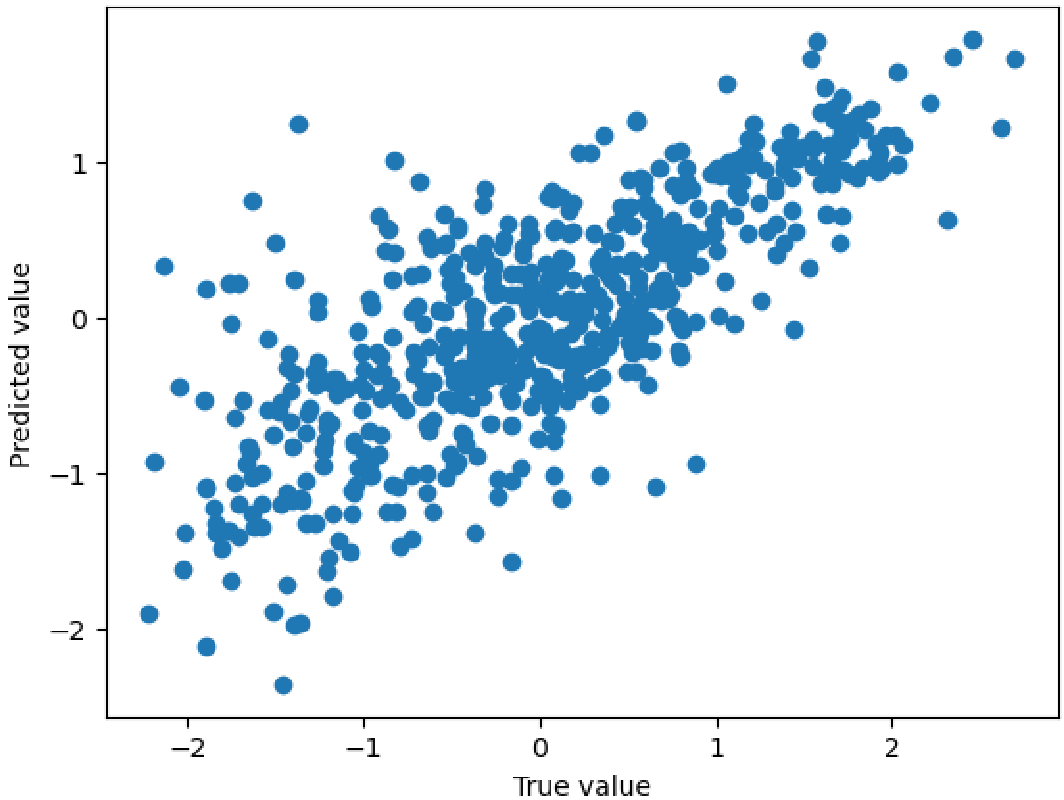
[41]:
```
#Based on this I believe that the feature with the greatest influence on the␣
↪band gap is feature (#) which is BLANK
```

[49]:
```
# Fit a baseline linear regression model
lr = LinearRegression()
lr.fit(x_train, y_train)
y_pred_lr = lr.predict(x_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
print("Baseline Linear Regression MSE: ", mse_lr)
```

```
Baseline Linear Regression MSE:  0.4836761609819223
```

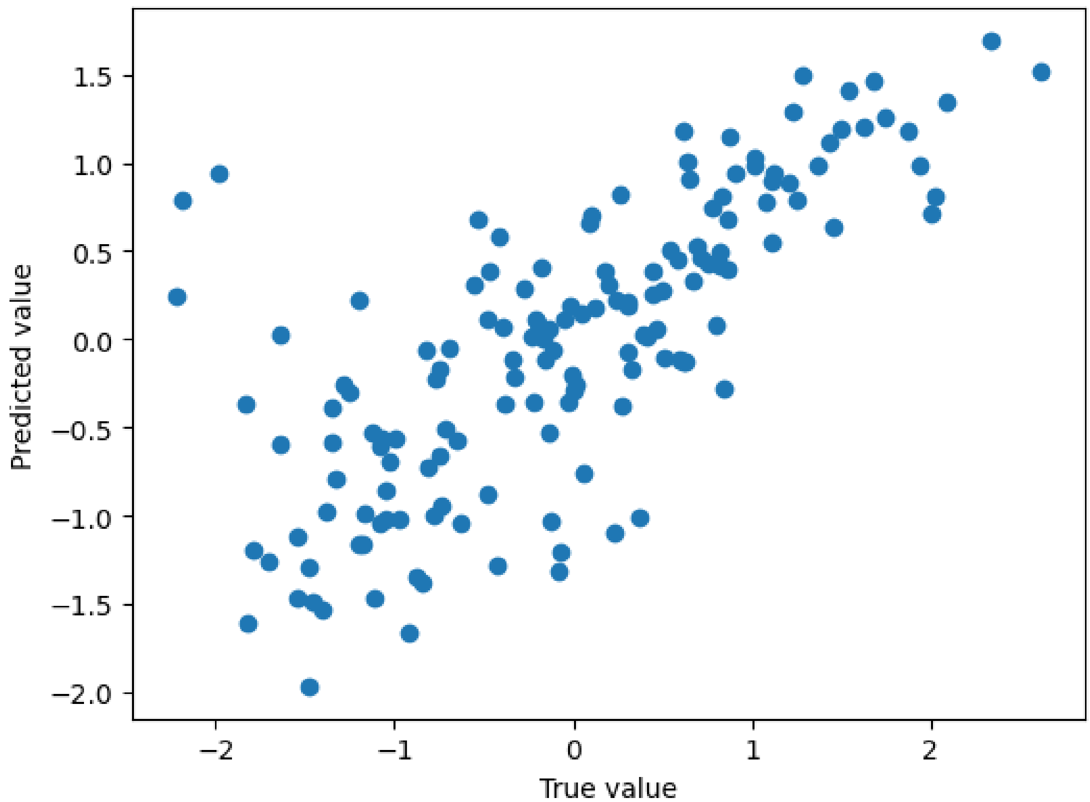
[43]:
```
#Analysing the performance of my model

#With the below code I am first making predictions for the training set. Then I
↪am creating a scatter plot graph to plot the predictions I made.

predictions = regr.predict(x_train)
plt.scatter(y_train, predictions)
plt.xlabel('True value')
plt.ylabel('Predicted value')
```

[43]: Text(0, 0.5, 'Predicted value')



[44]:
```
#From there, I can plot the test set

predictions = regr.predict(x_test)
plt.scatter(y_test, predictions)
plt.xlabel('True value')
plt.ylabel('Predicted value')
```

[44]: Text(0, 0.5, 'Predicted value')



[45]:
```
#Next, I will summarize the performance of my model using statistics values
#These include mean squared error, root mean squared error and r-squared

from sklearn.metrics import mean_squared_error, r2_score

print('Mean squared error:', mean_squared_error(predictions, y_test))
print('Root mean squared error:', mean_squared_error(predictions, y_test,␣
 ↪squared=False))
print('r-squared:', r2_score(y_test, predictions))
```

```
Mean squared error: 0.4836761609819223
Root mean squared error: 0.6954683033625058
r-squared: 0.5519131120239659
```

[51]:
```
#Gradient Boosted Regressor
```

[52]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

[53]:
```
regr = GradientBoostingRegressor()
regr.fit(x_train, y_train)
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/ensemble/_gb.py:570:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[53]:

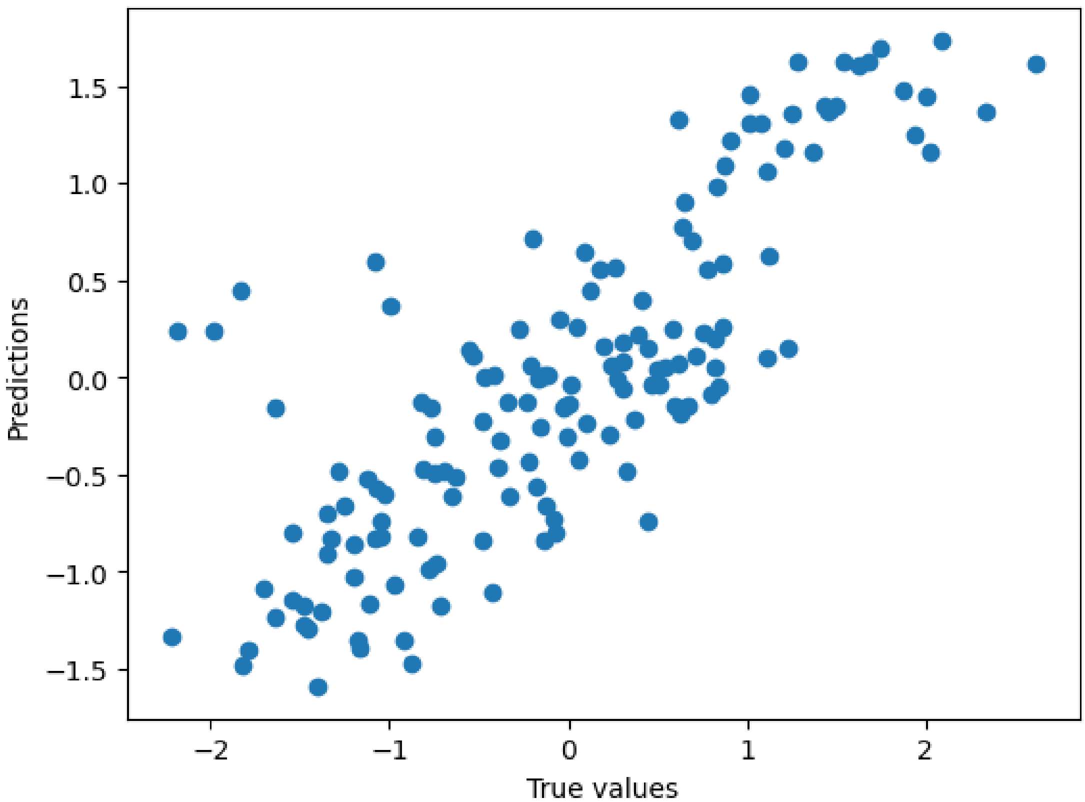
```
GradientBoostingRegressor()
```

[54]:

```python
predictions = regr.predict(x_test)
plt.scatter(y_test, predictions)
plt.xlabel('True values')
plt.ylabel('Predictions')
```

[54]:

```
Text(0, 0.5, 'Predictions')
```



[57]:

```python
#Testing my model's performance

from sklearn.metrics import mean_squared_error, r2_score

print('Mean squared error:', mean_squared_error(predictions, y_test))
```

```
print('Root mean squared error:', mean_squared_error(predictions, y_test,␣
    ↪squared=False))
print('r-squared:', r2_score(y_test, predictions))
```

```
Mean squared error: 0.36589473580019977
Root mean squared error: 0.6048923340564003
r-squared: 0.6610280871426852
```

[ ]:

```

```