Shaikat_303527_exercise_2

November 8, 2019

1 Pandas

On this part the data set is being loaded in dataframe named ' auto_imports' by using appropiate column names for the datas

Then cleaning and refining the data by replacing nan values with the mean of the specific columns

In [125]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from numpy.linalg import inv

pd.options.mode.chained_assignment = None
filename = r"E:\Documents\University of Hildesheim\Machine learning lab\lab2\imports-
column_names = ['symboling','normalized-losses','make','fuel-type','aspiration', 'nun
auto_imports = pd.read_csv(filename, delimiter=',',names=column_names,na_values=['?']

#replacing nan values in the columns
auto_imports.fillna(auto_imports.mean(), inplace=True)
auto_imports['num-of-doors'] = auto_imports['num-of-doors'].fillna('four')
auto_imports.head(5)
```

Out[125]:
```
   symboling  normalized-losses         make fuel-type aspiration  \
0          3              122.0  alfa-romero       gas        std
1          3              122.0  alfa-romero       gas        std
2          1              122.0  alfa-romero       gas        std
3          2              164.0         audi       gas        std
4          2              164.0         audi       gas        std

  num-of-doors   body-style drive-wheels engine-location  wheel-base  ...  \
0          two  convertible          rwd           front        88.6  ...
1          two  convertible          rwd           front        88.6  ...
2          two    hatchback          rwd           front        94.5  ...
3         four        sedan          fwd           front        99.8  ...
```

```
4          four         sedan          4wd           front        99.4  ...

   engine-size  fuel-system  bore  stroke compression-ratio horsepower  \
0          130         mpfi  3.47    2.68               9.0      111.0
1          130         mpfi  3.47    2.68               9.0      111.0
2          152         mpfi  2.68    3.47               9.0      154.0
3          109         mpfi  3.19    3.40              10.0      102.0
4          136         mpfi  3.19    3.40               8.0      115.0

   peak-rpm city-mpg  highway-mpg    price
0    5000.0       21           27  13495.0
1    5000.0       21           27  16500.0
2    5000.0       19           26  16500.0
3    5500.0       24           30  13950.0
4    5500.0       18           22  17450.0

[5 rows x 26 columns]
```

After analysing the dataframe(df) info its clear that there are 16 columns which are NUMERIC they are : symboling,normalized-losses,wheel-base,length,width,height,curb-weight,engine-size,bore,stroke,compression-ration,horse power,peak-rpm,city-mpg,highway-mpg,price

```
In [126]: auto_imports.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
symboling            205 non-null int64
normalized-losses    205 non-null float64
make                 205 non-null object
fuel-type            205 non-null object
aspiration           205 non-null object
num-of-doors         205 non-null object
body-style           205 non-null object
drive-wheels         205 non-null object
engine-location      205 non-null object
wheel-base           205 non-null float64
length               205 non-null float64
width                205 non-null float64
height               205 non-null float64
curb-weight          205 non-null int64
engine-type          205 non-null object
num-of-cylinders     205 non-null object
engine-size          205 non-null int64
fuel-system          205 non-null object
bore                 205 non-null float64
stroke               205 non-null float64
compression-ratio    205 non-null float64
```

```
horsepower           205 non-null float64
peak-rpm             205 non-null float64
city-mpg             205 non-null int64
highway-mpg          205 non-null int64
price                205 non-null float64
dtypes: float64(11), int64(5), object(10)
memory usage: 41.7+ KB
```

Pandas dataframe has paramater where skipna : boolean, default True exclude NA/null values when computing the result and numeric_only takes the columns which are numeric then the function mean std and median is used

```
In [127]: mean=auto_imports.mean(axis = 0, skipna = True,numeric_only=True)
          std_dev=auto_imports.std(axis = 0, skipna = True,numeric_only=True)
          median=auto_imports.median(axis = 0, skipna = True,numeric_only=True)
```

```
In [128]: print('The mean of each numeric columns are:')
          print(mean)
          print('****************************************')
          print('The standard deviation of each numeric columns are:')
          print(std_dev)
          print('****************************************')
          print('The median of each numeric columns are:')
          print(median)
```

```
The mean of each numeric columns are:
symboling                0.834146
normalized-losses      122.000000
wheel-base              98.756585
length                 174.049268
width                   65.907805
height                  53.724878
curb-weight           2555.565854
engine-size            126.907317
bore                     3.329751
stroke                   3.255423
compression-ratio       10.142537
horsepower             104.256158
peak-rpm              5125.369458
city-mpg                25.219512
highway-mpg             30.751220
price                13207.129353
dtype: float64
****************************************
The standard deviation of each numeric columns are:
symboling               1.245307
normalized-losses      31.681008
```

```
wheel-base              6.021776
length                 12.337289
width                   2.145204
height                  2.443522
curb-weight           520.680204
engine-size            41.642693
bore                    0.270844
stroke                  0.313597
compression-ratio       3.972040
horsepower             39.519211
peak-rpm              476.979093
city-mpg                6.542142
highway-mpg             6.886443
price                7868.768212
dtype: float64
****************************************
The median of each numeric columns are:
symboling                1.00
normalized-losses      122.00
wheel-base              97.00
length                 173.20
width                   65.50
height                  54.10
curb-weight           2414.00
engine-size            120.00
bore                     3.31
stroke                   3.29
compression-ratio        9.00
horsepower              95.00
peak-rpm              5200.00
city-mpg                24.00
highway-mpg             30.00
price                10595.00
dtype: float64
```

In this part the data is grouped by field "make"

```
In [129]: grp=auto_imports.groupby('make')
```

The data shows the average of price, highway-mpg,city-mpg which is grouped by 'make'

```
In [131]: mean_price=grp["price"].mean()
          mean_price.head(8)

Out[131]: make
          alfa-romero    15498.333333
          audi           17194.589908
```

```
bmw              26118.750000
chevrolet         6007.000000
dodge             7875.444444
honda             8184.692308
isuzu            11061.814677
jaguar           34600.000000
Name: price, dtype: float64
```

```
In [132]: mean_mpg=grp["highway-mpg"].mean()
          mean_mpg.head(8)
```

```
Out[132]: make
          alfa-romero    26.666667
          audi           24.142857
          bmw            25.375000
          chevrolet      46.333333
          dodge          34.111111
          honda          35.461538
          isuzu          36.000000
          jaguar         18.333333
          Name: highway-mpg, dtype: float64
```

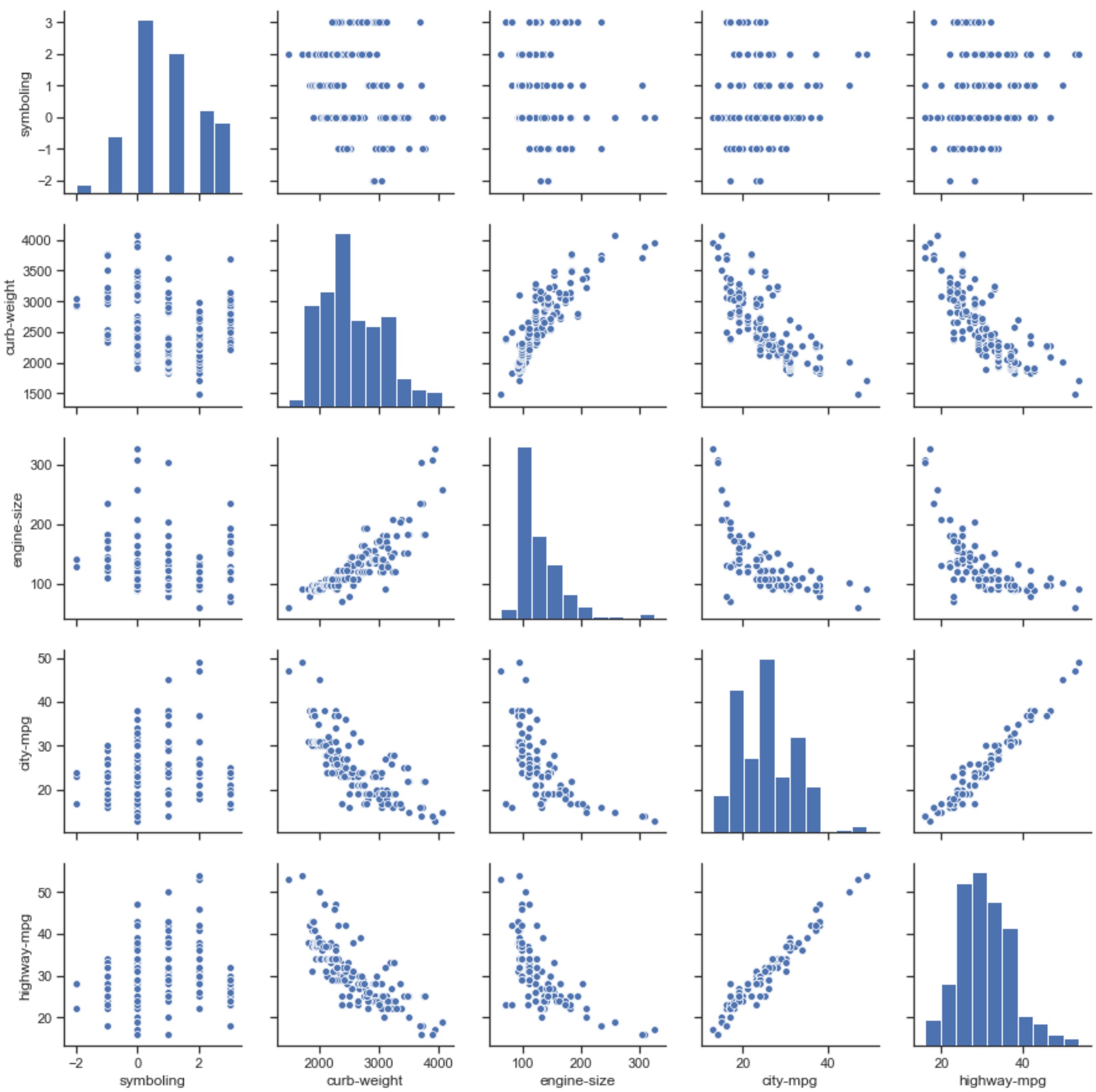
```
In [133]: mean_mpg=grp["city-mpg"].mean()
          mean_mpg.head(8)
```

```
Out[133]: make
          alfa-romero    20.333333
          audi           18.857143
          bmw            19.375000
          chevrolet      41.000000
          dodge          28.000000
          honda          30.384615
          isuzu          31.000000
          jaguar         14.333333
          Name: city-mpg, dtype: float64
```

There are several relations we can find in between the datas they are: 1.curb-weight vs engine-size : as the size of the engine increases the curbweight also increases 2.city-mpg and highway-mpg vs curb-weight : if curb weight increases the fuel consumption increases both in city and highway 3.engine-size vs city-mpg and highway-mpg : As the engine size increases the fuel consumption increases for both in city and highway

```
In [134]: sns.set(style="ticks", color_codes=True)

          g = sns.pairplot(auto_imports,vars=['symboling','curb-weight','engine-size','city-mpg
```

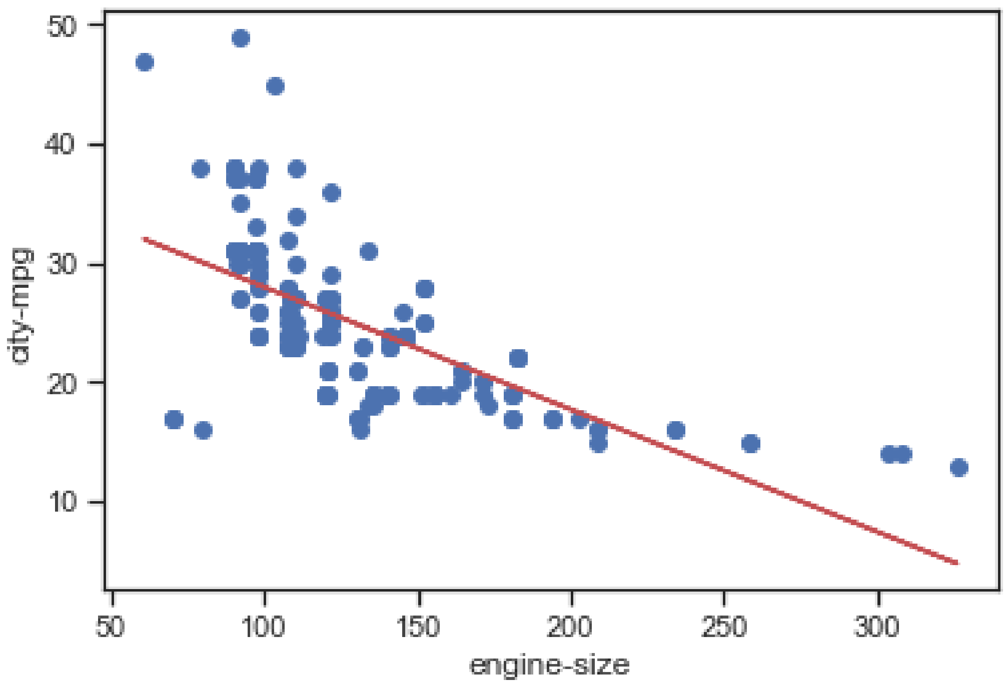



```
In [135]: X=auto_imports['engine-size']
          y=auto_imports['city-mpg']
          n=len(X)
```

```
In [136]: beta0 = (np.sum(y)*np.sum(X**2) -  np.sum(X)*np.sum(X*y))/ (n * np.sum(X**2) - np.su
          beta1 = (n*np.sum(X*y) - np.sum(X)*np.sum(y)) / (n*np.sum(X**2) - np.sum(X)**2)
          print("Beta 0 is ", beta0)
          print("Beta 1 is ", beta1)
          y_pred = beta0 + beta1 * X
          plt.scatter(X,y)
          plt.xlabel("engine-size")
          plt.ylabel("city-mpg")
          plt.plot( X,y_pred,'r')
```

```
Beta 0 is  38.25172970058016
Beta 1 is  -0.10269082828332302
```

Out[136]: [<matplotlib.lines.Line2D at 0x223c3311080>]



According to the fit the trend can illustrates as decreasing slope which means when the engine size increases the city-mpg also increases.On the other hand, the prediction is not good because the relative error is high for a lot of data points

2 Linear Regression via Normal Equations

In order to choose the columns which can be used to predict we need to figure out the correlation of the data with the price which our target.In this part we found the correlation of the datas between them.

In [137]: auto_imports.corr()

Out[137]:

	symboling	normalized-losses	wheel-base	length \
symboling	1.000000	0.465190	-0.531954	-0.357612
normalized-losses	0.465190	1.000000	-0.056518	0.019209
wheel-base	-0.531954	-0.056518	1.000000	0.874587
length	-0.357612	0.019209	0.874587	1.000000
width	-0.232919	0.084195	0.795144	0.841118
height	-0.541038	-0.370706	0.589435	0.491029

curb-weight	-0.227691	0.097785	0.776386	0.877728
engine-size	-0.105790	0.110997	0.569329	0.683360
bore	-0.130083	-0.029266	0.488760	0.606462
stroke	-0.008689	0.054929	0.160944	0.129522
compression-ratio	-0.178515	-0.114525	0.249786	0.158414
horsepower	0.071389	0.203434	0.351957	0.554434
peak-rpm	0.273679	0.237748	-0.360704	-0.287031
city-mpg	-0.035823	-0.218749	-0.470414	-0.670909
highway-mpg	0.034606	-0.178221	-0.544082	-0.704662
price	-0.082201	0.133999	0.583168	0.682986

	width	height	curb-weight	engine-size	bore \
symboling	-0.232919	-0.541038	-0.227691	-0.105790	-0.130083
normalized-losses	0.084195	-0.370706	0.097785	0.110997	-0.029266
wheel-base	0.795144	0.589435	0.776386	0.569329	0.488760
length	0.841118	0.491029	0.877728	0.683360	0.606462
width	1.000000	0.279210	0.867032	0.735433	0.559152
height	0.279210	1.000000	0.295572	0.067149	0.171101
curb-weight	0.867032	0.295572	1.000000	0.850594	0.648485
engine-size	0.735433	0.067149	0.850594	1.000000	0.583798
bore	0.559152	0.171101	0.648485	0.583798	1.000000
stroke	0.182939	-0.055351	0.168783	0.203094	-0.055909
compression-ratio	0.181129	0.261214	0.151362	0.028971	0.005201
horsepower	0.642195	-0.110137	0.750968	0.810713	0.575737
peak-rpm	-0.219859	-0.320602	-0.266283	-0.244599	-0.254761
city-mpg	-0.642704	-0.048640	-0.757414	-0.653658	-0.584508
highway-mpg	-0.677218	-0.107358	-0.797465	-0.677470	-0.586992
price	0.728699	0.134388	0.820825	0.861752	0.532300

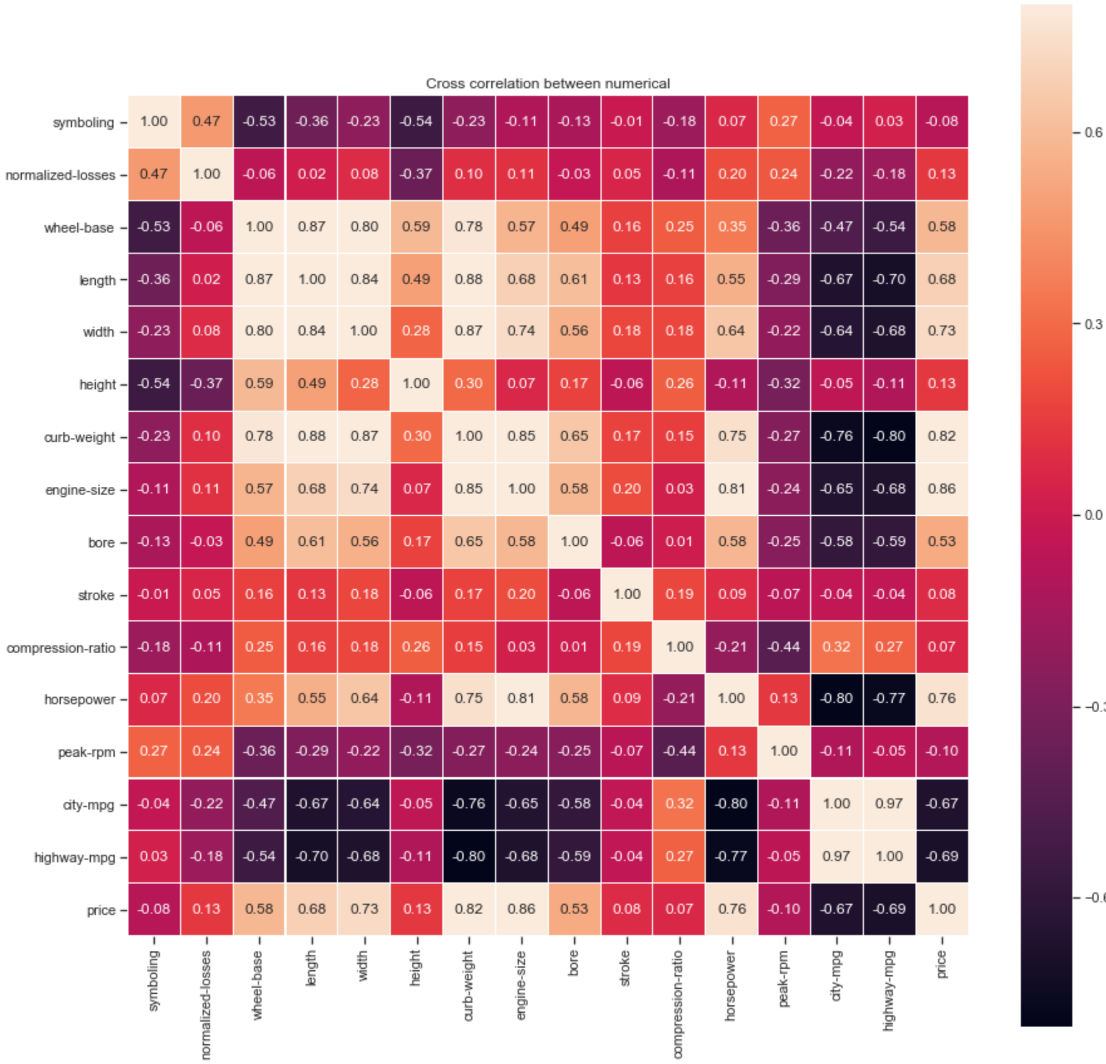
	stroke	compression-ratio	horsepower	peak-rpm \
symboling	-0.008689	-0.178515	0.071389	0.273679
normalized-losses	0.054929	-0.114525	0.203434	0.237748
wheel-base	0.160944	0.249786	0.351957	-0.360704
length	0.129522	0.158414	0.554434	-0.287031
width	0.182939	0.181129	0.642195	-0.219859
height	-0.055351	0.261214	-0.110137	-0.320602
curb-weight	0.168783	0.151362	0.750968	-0.266283
engine-size	0.203094	0.028971	0.810713	-0.244599
bore	-0.055909	0.005201	0.575737	-0.254761
stroke	1.000000	0.186105	0.088264	-0.066844
compression-ratio	0.186105	1.000000	-0.205740	-0.435936
horsepower	0.088264	-0.205740	1.000000	0.130971
peak-rpm	-0.066844	-0.435936	0.130971	1.000000
city-mpg	-0.042179	0.324701	-0.803162	-0.113723
highway-mpg	-0.043961	0.265201	-0.770903	-0.054257
price	0.082095	0.070990	0.757917	-0.100854

	city-mpg	highway-mpg	price

```
symboling          -0.035823     0.034606 -0.082201
normalized-losses -0.218749    -0.178221  0.133999
wheel-base         -0.470414    -0.544082  0.583168
length             -0.670909    -0.704662  0.682986
width              -0.642704    -0.677218  0.728699
height             -0.048640    -0.107358  0.134388
curb-weight        -0.757414    -0.797465  0.820825
engine-size        -0.653658    -0.677470  0.861752
bore               -0.584508    -0.586992  0.532300
stroke             -0.042179    -0.043961  0.082095
compression-ratio   0.324701     0.265201  0.070990
horsepower         -0.803162    -0.770903  0.757917
peak-rpm           -0.113723    -0.054257 -0.100854
city-mpg            1.000000     0.971337 -0.667449
highway-mpg         0.971337     1.000000 -0.690526
price              -0.667449    -0.690526  1.000000
```

Now if we plot the data in heatmap it shows something like this

```
In [138]: plt.figure(figsize=(15, 15))
          ax = sns.heatmap(auto_imports.corr(), vmax=.8, square=True, fmt='.2f', annot=True, li
          plt.title('Cross correlation between numerical')
          plt.show()
```



1 Above graph shows Wheel base , Length , Width are highly correlated.

2 Highway mpg and city mpg is also highly correlated.

3 Compression ratio and fuel type is also correlated

4 Engine size and horse power is also correlated Attributes which has stronger relationship with price - 1. Curb-Weight 2. Engine-Size 3. Horsepower 4. Mpg(City / Highway mpg) 5. Widthű

```
In [139]: from sklearn.model_selection import train_test_split
          Ydata = auto_imports['price']
          Xdata = auto_imports[['curb-weight','engine-size','horsepower','city-mpg','highway-mp
```

```
x_train, x_test, y_train, y_test =train_test_split(Xdata, Ydata,train_size=0.8, test_
```

The dataset Xdata; Ydata into Xtrain; Ytrain and Xtest; Ytest and assigned 80% of the data to a Xtrain, Ytrain set and remaining 20% to a Xtest; ytest set.

```
In [140]: x_train.shape

Out[140]: (164, 5)
```

In this part linear regression we calculated the value of a and b which later used for gaussian elimination to find the betas

```
In [141]: temp=np.transpose(x_train)
          a=temp.dot(x_train)
          b=temp.dot(y_train)
          a=a.values
          b= np.reshape(b.values, (a.shape[0], 1))


          def Gaussian_elimination(A,b):
              n =  len(A)
              for pivot_row in range(n-1):
                  for row in range(pivot_row+1, n):
                      multiplier = A[row][pivot_row]/A[pivot_row][pivot_row]
                      #the only one in this column since the rest are zero
                      A[row][pivot_row] = multiplier
                      for col in range(pivot_row + 1, n):
                          A[row][col] = A[row][col] - multiplier*A[pivot_row][col]
                      #Equation solution column
                      b[row] = b[row] - multiplier*b[pivot_row]
              x = np.zeros(n)
              k = n-1
              x[k] = b[k]/A[k,k]
              while k >= 0:
                  x[k] = (b[k] - np.dot(A[k,k+1:],x[k+1:]))/A[k,k]
                  k = k-1
              return x

          betas_gauss_elimination=Gaussian_elimination(a,b)
          print(betas_gauss_elimination)

[   1.80706542   97.14317341   26.1328034   -38.9751633  -177.96642241]
```

Then y prediction of gaussian elimination illustrated

```
In [142]: y_pred_gaus=np.dot(x_test,betas_gauss_elimination)
          print(np.sort(y_pred_gaus))
```

```
[-1395.03157134  4825.01007961  4888.25736944  5441.44970639
  5939.01685881  6088.56492951  6227.68847913  6846.61014898
  6935.15635474  7208.85867288  7315.51410461  7355.81785044
  7948.33065832  8041.75712664  8984.4454988   9698.51394028
 10048.54710593 10649.85695295 10917.2535033  11028.46310431
 11277.78114634 11628.90318905 11764.47789514 13579.39765214
 13794.43843756 14684.80106921 15064.2848082  15426.70431161
 15549.73581617 15642.15715064 15781.04019041 15844.28748024
 16234.46255603 17880.01630991 18189.15869158 20399.10915846
 21257.25107732 22123.97218388 26628.11068712 27683.25619336
 37660.09354555]
```

Another method is to solve SLE is QR decompostion a QR decomposition, also known as a QR factorization or QU factorization is a decomposition of a matrix A into a product A = QR of an orthogonal matrix Q and an upper triangular matrix R

```
In [160]: def qr(A):
              m, n = A.shape
              Q = np.eye(m)
              for i in range(n - (m == n)):
                  H = np.eye(m)
                  H[i:, i:] = make_householder(A[i:, i])
                  Q = np.dot(Q, H)
                  A = np.dot(H, A)
              return Q, A

          def make_householder(a):
              v = a / (a[0] + np.copysign(np.linalg.norm(a), a[0]))
              v[0] = 1
              H = np.eye(a.shape[0])
              H -= (2 / np.dot(v, v)) * np.dot(v[:, None], v[None, :])
              return H


          q,r = qr(a)

          p = np.dot(q.T, b)
          betas_qr=np.dot(np.linalg.inv(r), p)
          betas_qr=betas_qr.flatten()

          print('The betas after qr decompostion is :',betas_qr)
          print('q:\n', q.round(6))
          print('r:\n', r.round(6))
```

```
The betas after qr decompostion is : [    1.80708237    97.14693198    26.1256531    -39.15989068
q:
 [[-1.0e+00  0.0e+00  0.0e+00  0.0e+00 -0.0e+00]
```

```
 [-0.0e+00 -1.0e+00  7.0e-06 -2.0e-06  2.0e-06]
 [-0.0e+00 -7.0e-06 -1.0e+00 -1.0e-06  0.0e+00]
 [-0.0e+00  2.0e-06  1.0e-06 -1.0e+00 -8.3e-05]
 [-0.0e+00  2.0e-06  0.0e+00 -8.3e-05  1.0e+00]]
r:
 [[-1.13274884e+09 -5.67736910e+07 -4.64083680e+07 -1.02204150e+07
  -1.24448070e+07]
 [ 0.00000000e+00 -8.86546017e+04 -5.65870669e+04  1.52121976e+04
   1.71513492e+04]
 [ 0.00000000e+00 -0.00000000e+00 -7.72410956e+04  1.25795524e+04
   1.14824651e+04]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -1.35246141e+04
  -1.57766347e+04]
 [-0.00000000e+00 -0.00000000e+00 -0.00000000e+00  0.00000000e+00
   4.26601738e+02]]
```

The y prediction of qr decompostion is illustrated

In [144]:
```
y_pred_qr=np.dot(x_test,betas_qr)
print(np.sort(y_pred_qr))
```

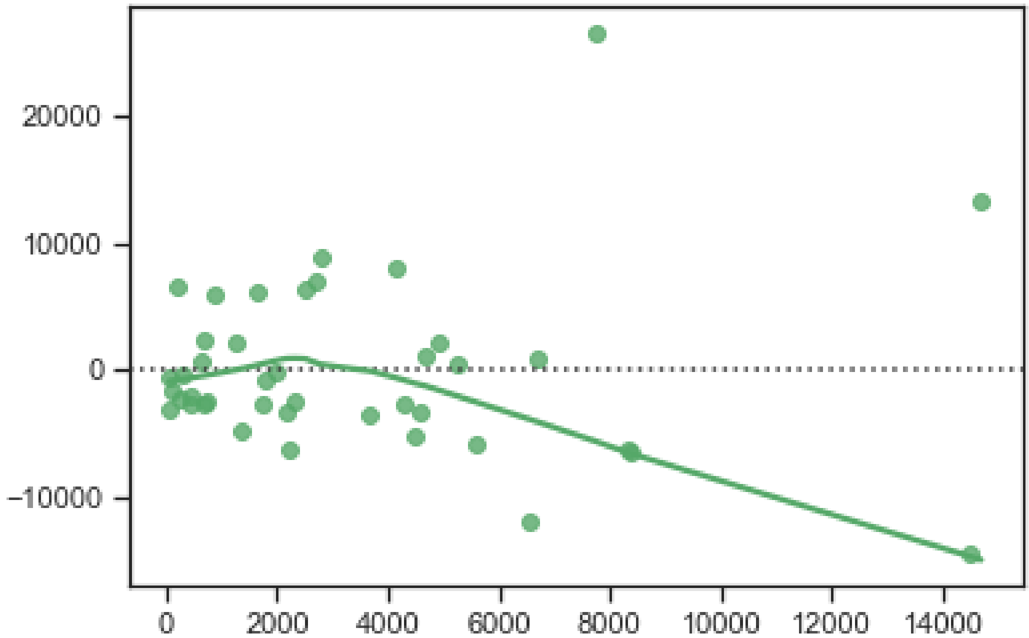
```
[-1395.33921648  4824.72637148  4887.97425445  5441.05392049
  5939.24215546  6088.79447618  6227.96655033  6846.69514137
  6935.24217752  7208.5813591   7315.44214113  7356.08743224
  7948.3941937   8041.91780518  8984.55912421  9698.79491031
 10048.66545752 10649.8975694  10917.25524516 11028.61902447
 11277.6934537  11629.43891004 11764.64386515 13579.8914923
 13794.93429437 14684.94753257 15064.43483035 15426.9537162
 15549.85543481 15642.15744664 15781.16197822 15844.40986118
 16234.71953575 17880.41009418 18189.54772114 20398.97484278
 21257.11630807 22124.04276702 26628.20956947 27683.34773311
 37659.95221804]
```

In [154]:
```
y_test= np.reshape(y_test, (y_pred_qr.shape[0], 1))
y_test=y_test.flatten()

epsilon_qr=abs(y_test-y_pred_qr)
epsilon_gauss=abs(y_test-y_pred_gaus)
```

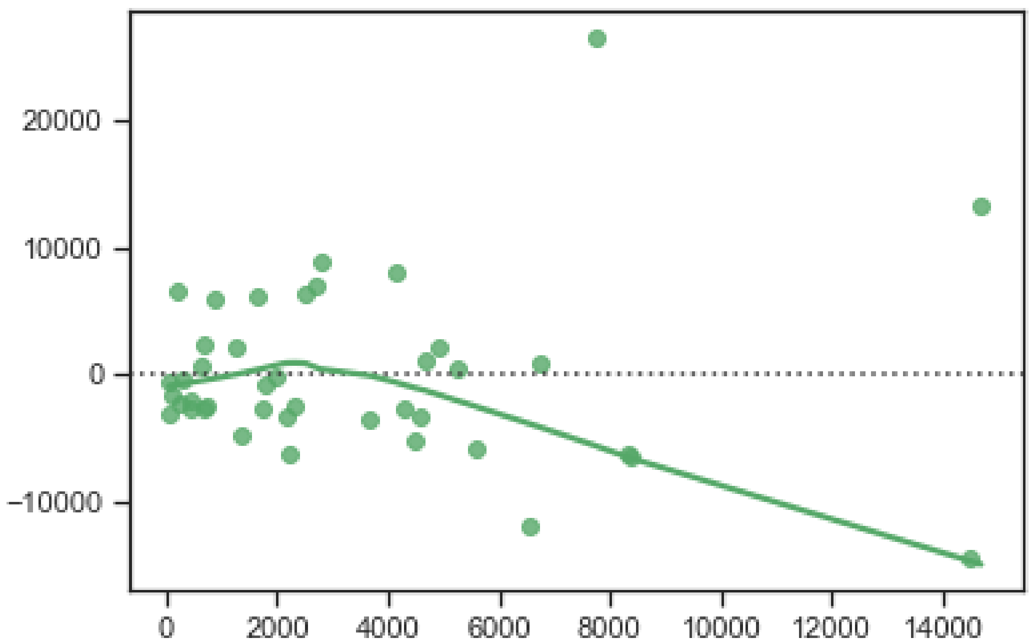
In [155]:
```
sns.residplot(epsilon_qr, y_test, lowess=True, color="g")
```

Out[155]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x223c87c2b38>
```




```
In [156]: sns.residplot(epsilon_gauss, y_test, lowess=True, color="g")

Out[156]: <matplotlib.axes._subplots.AxesSubplot at 0x223c8828438>
```



```
In [157]: print('average residual of qr decompostion model is :',np.mean(epsilon_qr))
          print('average residual of gaussian elimination model is :',np.mean(epsilon_gauss))
```

```
average residual of qr decompostion model is : 3372.0603842510873
average residual of gaussian elimination model is : 3372.0688538019604
```

```
In [158]: print("rmse value of qr decomposition is =",np.sqrt(np.mean((y_test-y_pred_qr)**2)))
          print("rmse value of Gaussian elimination is =",np.sqrt(np.mean((y_test-y_pred_gaus)
```

```
rmse value of qr decomposition is = 4846.2285090644245
rmse value of Gaussian elimination is = 4846.202719226408
```

```
In [ ]:
```