In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.impute import SimpleImputer
from scipy.stats import zscore
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.decomposition import PCA
from sklearn import metrics
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

In [2]:

```
veh= pd.read_csv("vehicle-1.csv") # Loading the dataset.
```

In [3]:

```
veh.shape # shape od the dataset.
```

Out[3]:

```
(846, 19)
```

In [4]:

```
veh.head() # to get a general idea of data.
```

Out[4]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_aspect_ratio	scatter_ratio	elongatednes
0	95	48.0	83.0	178.0	72.0	10	162.0	42.(
1	91	41.0	84.0	141.0	57.0	9	149.0	45.(
2	104	50.0	106.0	209.0	66.0	10	207.0	32.(
3	93	41.0	82.0	159.0	63.0	9	144.0	46.(
4	85	44.0	70.0	205.0	103.0	52	149.0	45.(

In [5]:

```
veh.info() #Identifying datatype of each columns.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 846 entries, 0 to 845
Data columns (total 19 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   compactness                  846 non-null    int64
 1   circularity                  841 non-null    float64
 2   distance_circularity         842 non-null    float64
 3   radius_ratio                 840 non-null    float64
 4   pr.axis_aspect_ratio         844 non-null    float64
 5   max.length_aspect_ratio      846 non-null    int64
 6   scatter_ratio                845 non-null    float64
 7   elongatedness                845 non-null    float64
 8   pr.axis_rectangularity       843 non-null    float64
 9   max.length_rectangularity    846 non-null    int64
 10  scaled_variance              843 non-null    float64
 11  scaled_variance.1            844 non-null    float64
 12  scaled_radius_of_gyration    844 non-null    float64
```

```
 13  scaled_radius_of_gyration.1  842 non-null    float64
 14  skewness_about               840 non-null    float64
 15  skewness_about.1             845 non-null    float64
 16  skewness_about.2             845 non-null    float64
 17  hollows_ratio                846 non-null    int64
 18  class                        846 non-null    object
dtypes: float64(14), int64(4), object(1)
memory usage: 125.7+ KB
```

In [6]:

```
veh.describe()
```

Out[6]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_aspect_ratio	scatter_ratio	elongate
count	846.000000	841.000000	842.000000	840.000000	844.000000	846.000000	845.000000	845.(
mean	93.678487	44.828775	82.110451	168.888095	61.678910	8.567376	168.901775	40.9
std	8.234474	6.152172	15.778292	33.520198	7.891463	4.601217	33.214848	7.8
min	73.000000	33.000000	40.000000	104.000000	47.000000	2.000000	112.000000	26.(
25%	87.000000	40.000000	70.000000	141.000000	57.000000	7.000000	147.000000	33.(
50%	93.000000	44.000000	80.000000	167.000000	61.000000	8.000000	157.000000	43.(
75%	100.000000	49.000000	98.000000	195.000000	65.000000	10.000000	198.000000	46.(
max	119.000000	59.000000	112.000000	333.000000	138.000000	55.000000	265.000000	61.(

In [7]:

```
veh.isnull().values.any() # To check if there are null values.
```

Out[7]:

```
True
```

In [8]:

```
labelencoder = LabelEncoder()
veh['class_dup'] = labelencoder.fit_transform(veh['class'])
veh
```

Out[8]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_aspect_ratio	scatter_ratio	elongatedn
0	95	48.0	83.0	178.0	72.0	10	162.0	4
1	91	41.0	84.0	141.0	57.0	9	149.0	4
2	104	50.0	106.0	209.0	66.0	10	207.0	3
3	93	41.0	82.0	159.0	63.0	9	144.0	4
4	85	44.0	70.0	205.0	103.0	52	149.0	4
...	...	...	...	...	...	...	...	
841	93	39.0	87.0	183.0	64.0	8	169.0	4
842	89	46.0	84.0	163.0	66.0	11	159.0	4
843	106	54.0	101.0	222.0	67.0	12	222.0	3
844	86	36.0	78.0	146.0	58.0	7	135.0	5
845	85	36.0	66.0	123.0	55.0	5	120.0	5

846 rows × 20 columns

> In the above step, The class was a categorical column. \ This can't be used in SVM algorithms. Thus the columns elements were given number. \ Such as bus is 0, Van is 2 and car is 1.

In [9]:

```
veh.mean()
```

Out[9]:

```
compactness                   93.678487
circularity                   44.828775
distance_circularity          82.110451
radius_ratio                 168.888095
pr.axis_aspect_ratio          61.678910
max.length_aspect_ratio        8.567376
scatter_ratio                168.901775
elongatedness                 40.933728
pr.axis_rectangularity        20.582444
max.length_rectangularity    147.998818
scaled_variance              188.631079
scaled_variance.1            439.494076
scaled_radius_of_gyration    174.709716
scaled_radius_of_gyration.1   72.447743
skewness_about                 6.364286
skewness_about.1              12.602367
skewness_about.2             188.919527
hollows_ratio                195.632388
class_dup                      0.977541
dtype: float64
```

In [10]:

```
veh.drop("class",axis=1,inplace=True)
```

In [11]:

```
meanFiller = lambda x: x.fillna(x.mean())
veh = veh.apply(meanFiller,axis=0)
```

> In the above step null values were replaced with means values of there respective columns \ and the class clomn was dropped.

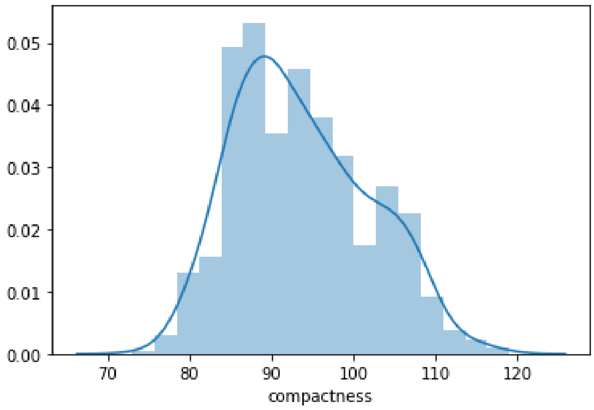
Univariate Analysis

In [12]:

```
sns.distplot(veh["compactness"])
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x240748e5588>
```

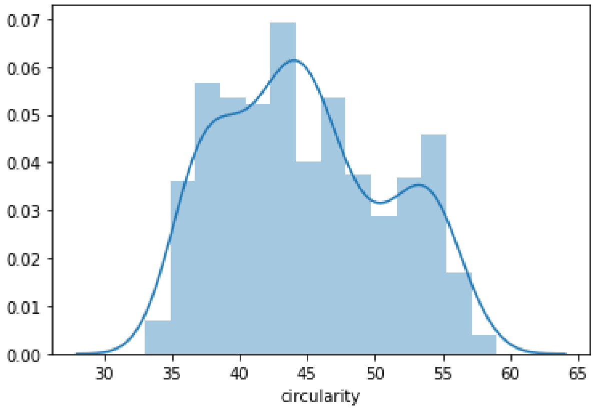


In [13]:

```
sns.distplot(veh["circularity"])
```

Out[13]:

<matplotlib.axes._subplots.AxesSubplot at 0x2407500d588>

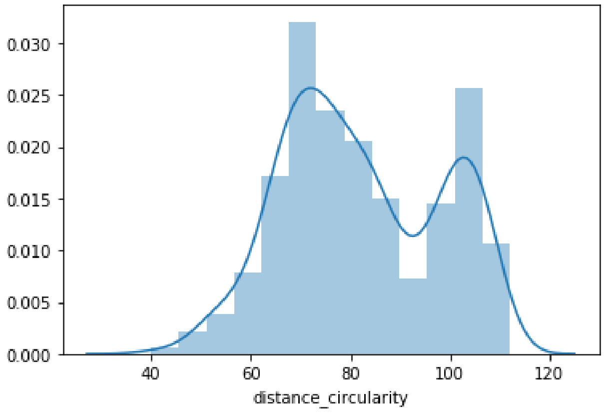


In [14]:

```
sns.distplot(veh["distance_circularity"])
```

Out[14]:

<matplotlib.axes._subplots.AxesSubplot at 0x240750fbf48>

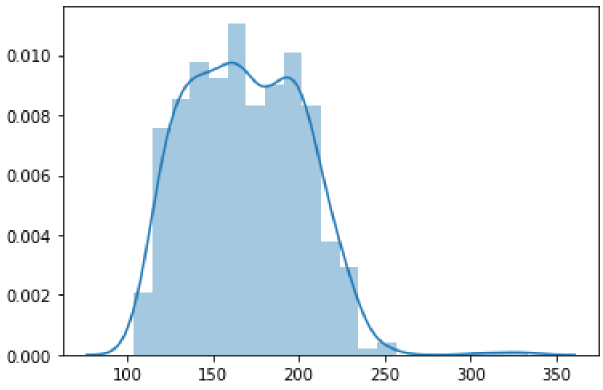


In [15]:

```
sns.distplot(veh["radius_ratio"])
```

Out[15]:

<matplotlib.axes._subplots.AxesSubplot at 0x240751b9c08>



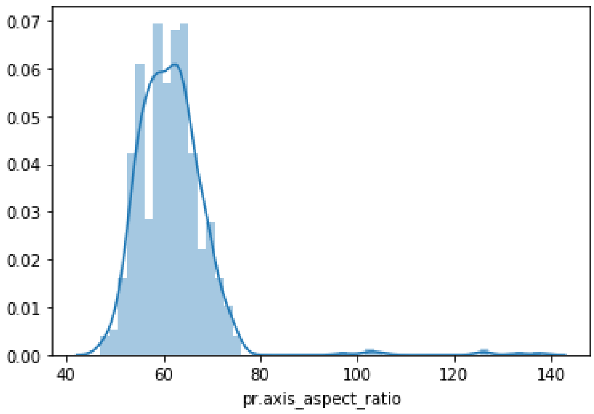
radius_ratio

In [16]:

```
sns.distplot(veh["pr.axis_aspect_ratio"])
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24075254048>
```

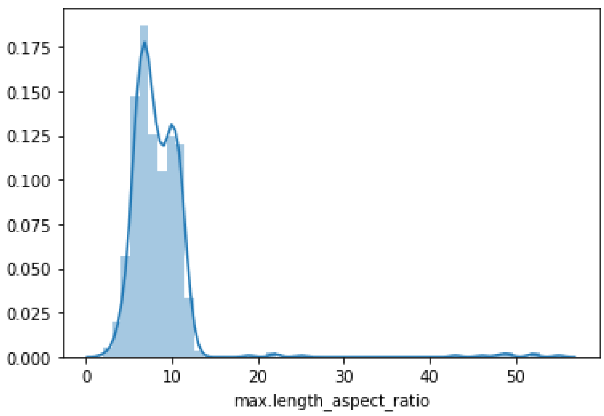


In [17]:

```
sns.distplot(veh["max.length_aspect_ratio"])
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x240753568c8>
```

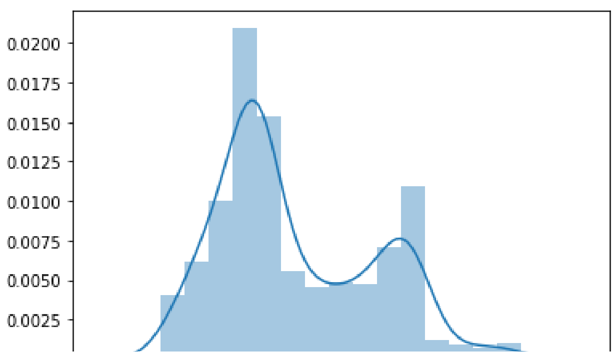


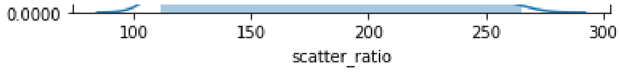
In [18]:

```
sns.distplot(veh["scatter_ratio"])
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2407533d808>
```



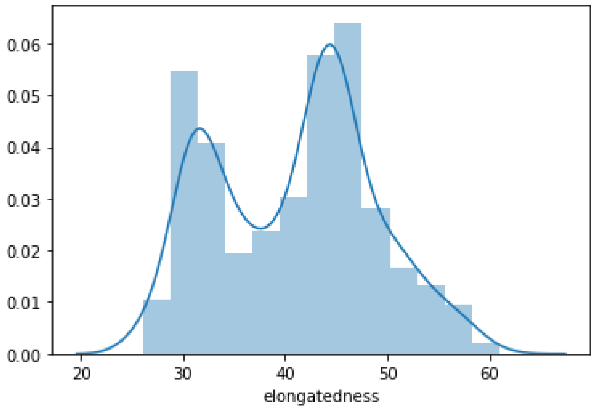


In [19]:

```
sns.distplot(veh["elongatedness"])
```

Out[19]:

<matplotlib.axes._subplots.AxesSubplot at 0x240754d54c8>

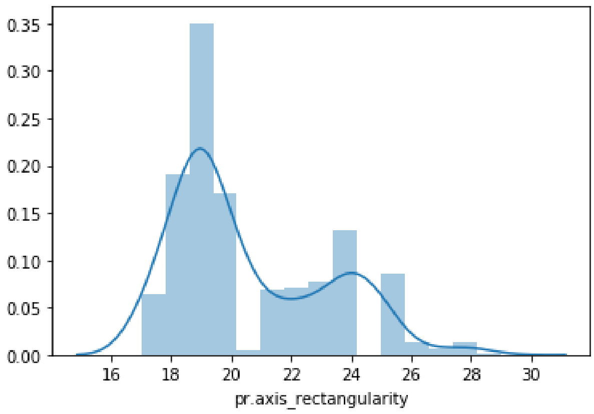


In [20]:

```
sns.distplot(veh["pr.axis_rectangularity"])
```

Out[20]:

<matplotlib.axes._subplots.AxesSubplot at 0x24075564d48>

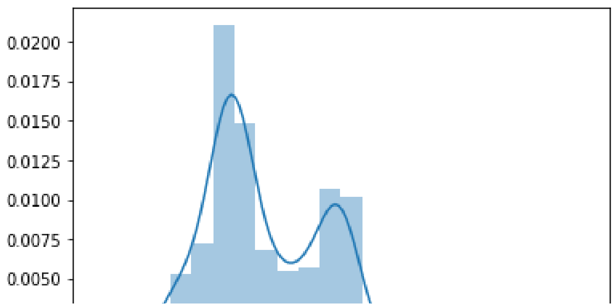


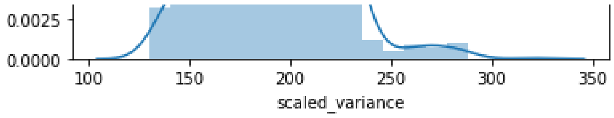
In [21]:

```
sns.distplot(veh["scaled_variance"])
```

Out[21]:

<matplotlib.axes._subplots.AxesSubplot at 0x240754de708>



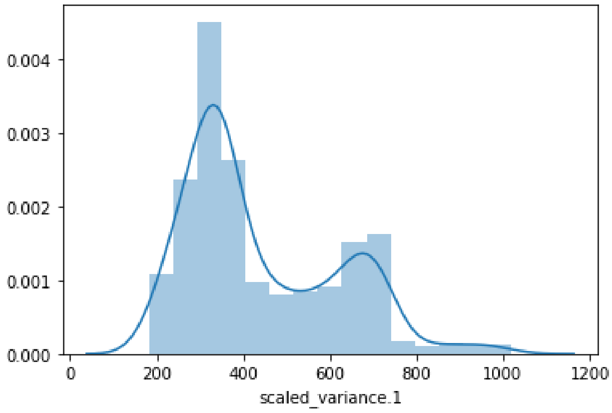


In [22]:

```
sns.distplot(veh["scaled_variance.1"])
```

Out[22]:

<matplotlib.axes._subplots.AxesSubplot at 0x240756a1a48>

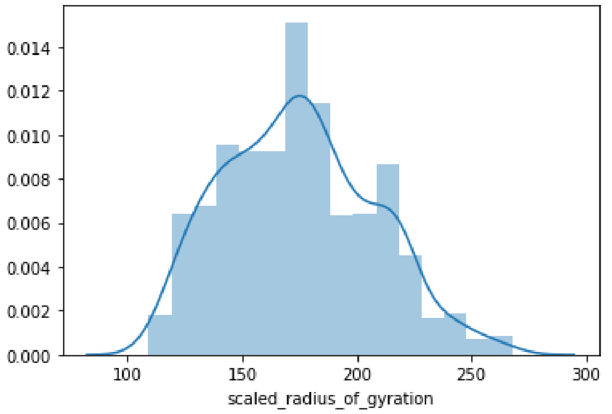


In [23]:

```
sns.distplot(veh["scaled_radius_of_gyration"])
```

Out[23]:

<matplotlib.axes._subplots.AxesSubplot at 0x2407572b048>

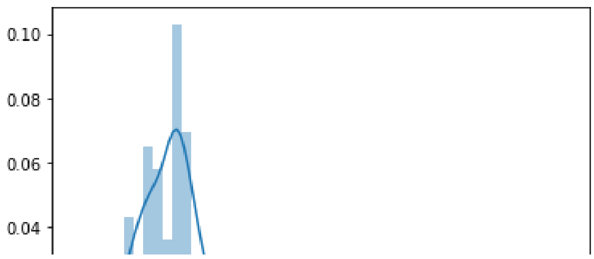


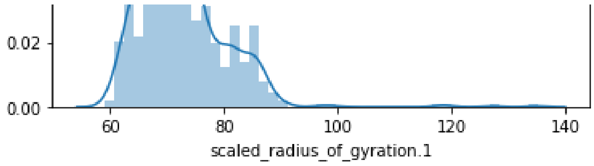
In [24]:

```
sns.distplot(veh["scaled_radius_of_gyration.1"])
```

Out[24]:

<matplotlib.axes._subplots.AxesSubplot at 0x240757c42c8>



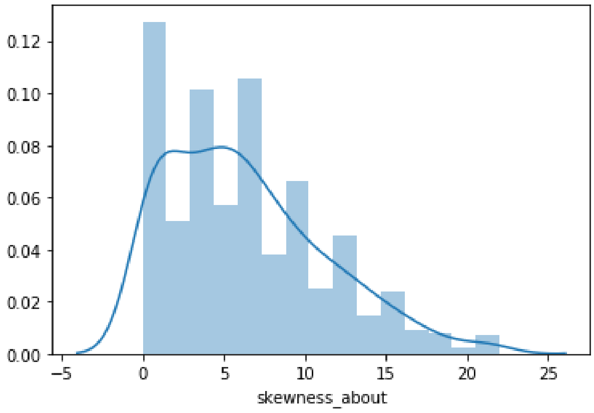


In [25]:

```
sns.distplot(veh["skewness_about"])
```

Out[25]:

<matplotlib.axes._subplots.AxesSubplot at 0x2407581a908>

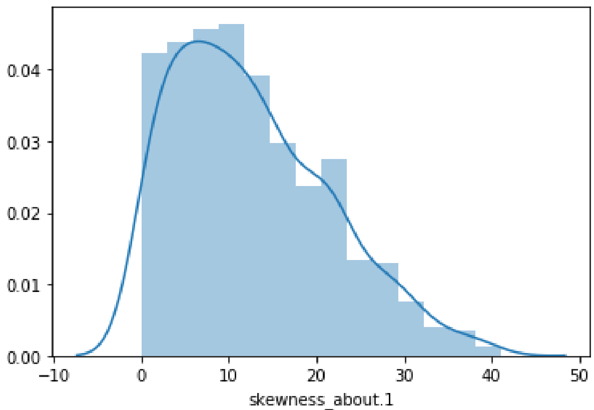


In [26]:

```
sns.distplot(veh["skewness_about.1"])
```

Out[26]:

<matplotlib.axes._subplots.AxesSubplot at 0x2407593d6c8>

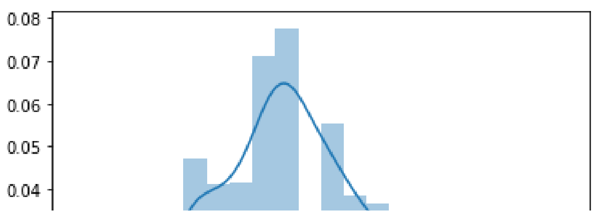


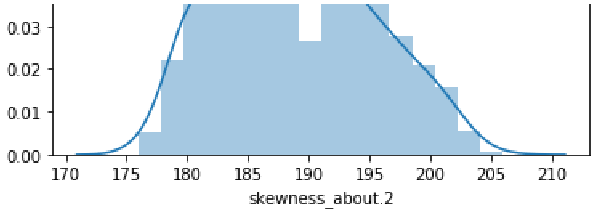
In [27]:

```
sns.distplot(veh["skewness_about.2"])
```

Out[27]:

<matplotlib.axes._subplots.AxesSubplot at 0x240769aa088>



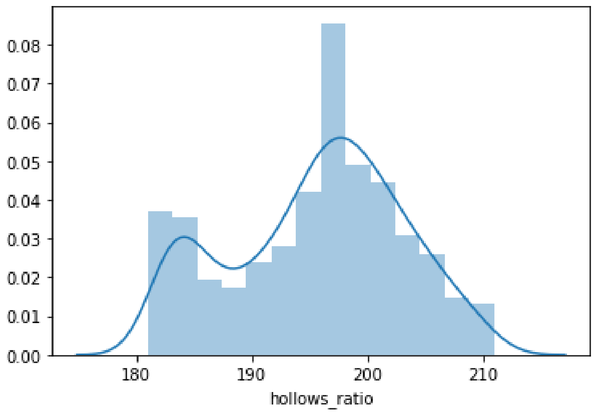


In [28]:

```
sns.distplot(veh["hollows_ratio"])
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24076a33688>
```



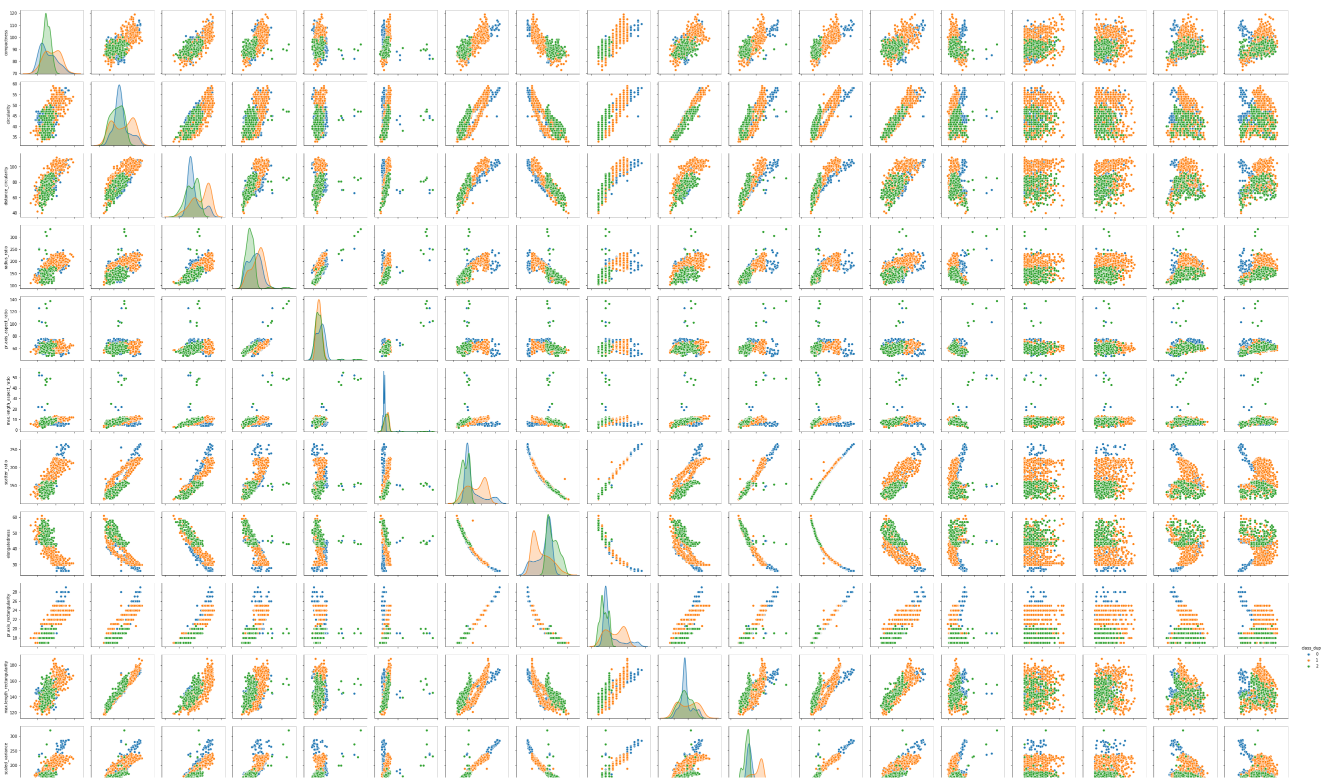
Bivariate Analysis

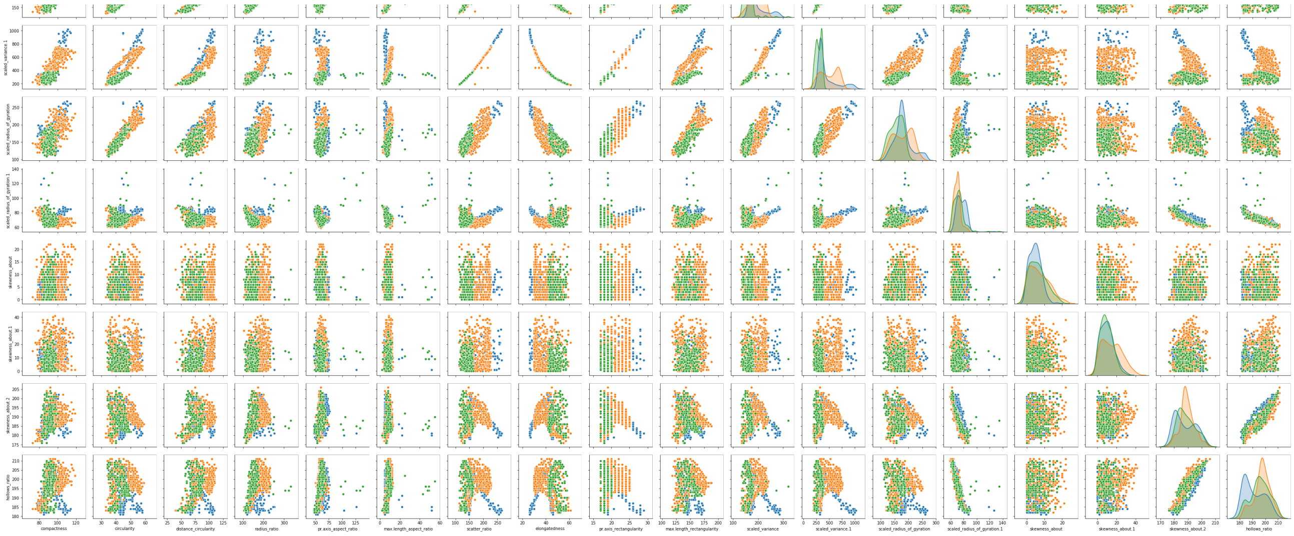
In [29]:

```
sns.pairplot(veh,hue='class_dup',diag_kind='kde')
```

Out[29]:

```
<seaborn.axisgrid.PairGrid at 0x24076a21888>
```





Correlation table

In [30]:

```
core=veh.corr()
veh.corr()
```

Out[30]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_aspect_ratio	s
compactness	1.000000	0.685421	0.789909	0.689840	0.091704	0.148249	
circularity	0.685421	1.000000	0.793016	0.620967	0.153362	0.251208	
distance_circularity	0.789909	0.793016	1.000000	0.767079	0.158397	0.264550	
radius_ratio	0.689840	0.620967	0.767079	1.000000	0.663559	0.450036	
pr.axis_aspect_ratio	0.091704	0.153362	0.158397	0.663559	1.000000	0.648704	
max.length_aspect_ratio	0.148249	0.251208	0.264550	0.450036	0.648704	1.000000	
scatter_ratio	0.812235	0.848207	0.904400	0.734228	0.103715	0.165967	
elongatedness	-0.788643	-0.821901	-0.911435	-0.789795	-0.183264	-0.180041	
pr.axis_rectangularity	0.813636	0.844972	0.893128	0.708285	0.079395	0.161592	
max.length_rectangularity	0.676143	0.961943	0.774669	0.569205	0.127128	0.305943	
scaled_variance	0.762770	0.796822	0.861980	0.794041	0.273414	0.318955	
scaled_variance.1	0.815901	0.838525	0.887328	0.720150	0.089620	0.143713	
scaled_radius_of_gyration	0.585156	0.926888	0.705953	0.536536	0.122111	0.189704	
scaled_radius_of_gyration.1	-0.250071	0.052642	-0.225852	-0.180819	0.152776	0.295574	
skewness_about	0.235687	0.144394	0.113813	0.048720	-0.058481	0.015439	
skewness_about.1	0.157387	-0.011851	0.265553	0.173832	-0.032134	0.043489	
skewness_about.2	0.298526	-0.105645	0.145563	0.382129	0.239849	-0.026180	
hollows_ratio	0.365552	0.045318	0.332095	0.471262	0.267724	0.143919	
class_dup	-0.033796	-0.159804	-0.064902	-0.182270	-0.098318	0.207619	

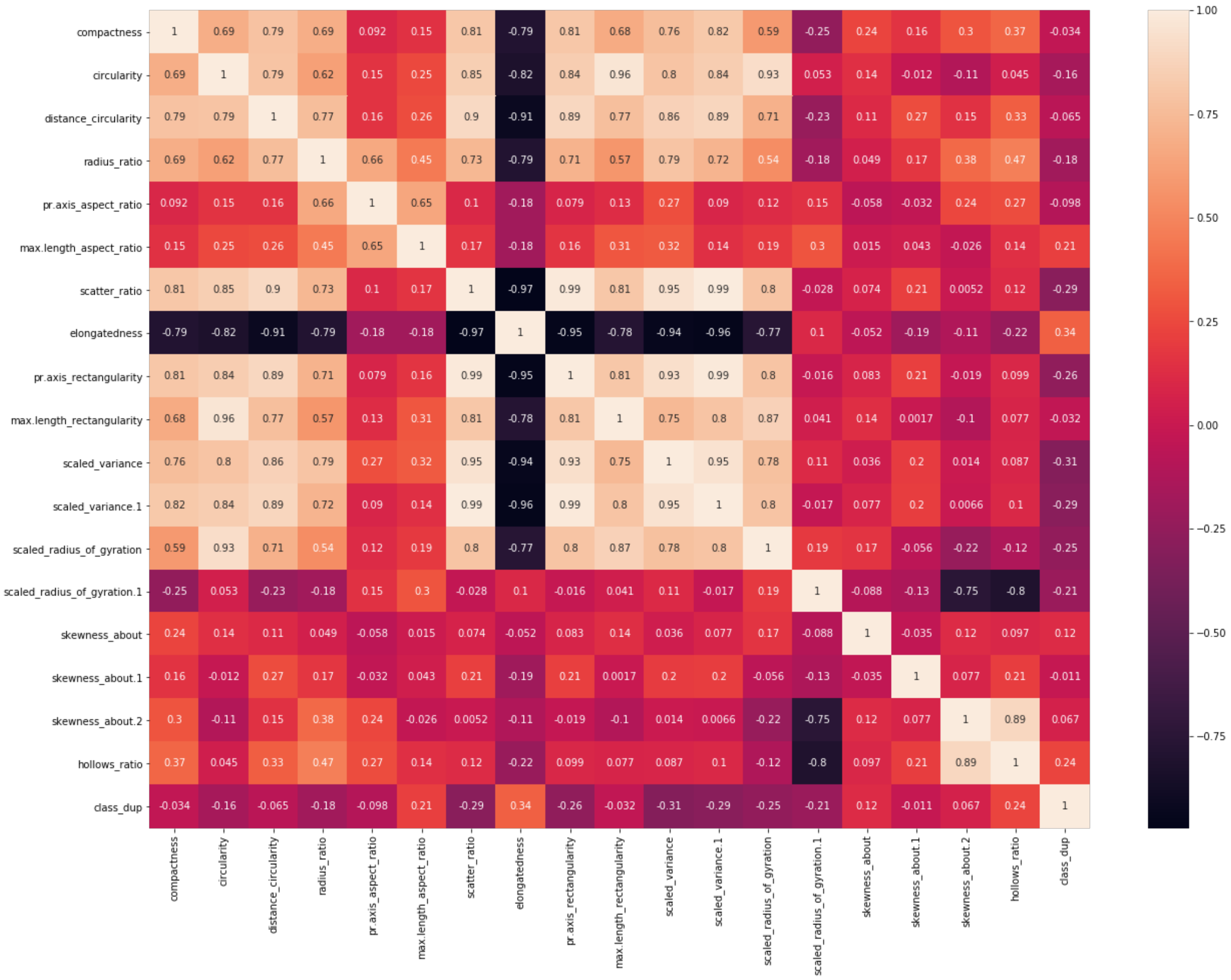
Correlation Heatmap

In [31]:

```
plt.figure(figsize = (21,15))
sns.heatmap(core,annot=True)
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x24004a91288>
```



> From the bivariate analysis, correlation table and correlation heatmap.\ The columns the is of least significance to target variable was found.

Preprocessing Data for analysis

In [32]:

```
veh.drop("compactness",axis=1,inplace=True)
veh.drop("distance_circularity",axis=1,inplace=True)
veh.drop("pr.axis_aspect_ratio",axis=1,inplace=True)
veh.drop("max.length_rectangularity",axis=1,inplace=True)
veh.drop("scatter_ratio",axis=1,inplace=True)
veh.drop("pr.axis_rectangularity",axis=1,inplace=True)
veh.drop("skewness_about.2",axis=1,inplace=True)
```

In [33]:

```
X= veh.drop(["class_dup"],axis=1,)
Y= veh[["class_dup"]]
```

In [34]:

```
XScaled  = X.apply(zscore)
```

Splitting the data into test,train

In [35]:

```
x_train, x_test, y_train, y_test = train_test_split(XScaled, Y, test_size=0.3, random_state=7)
```

SVM Analysis

In [36]:

```
clf =  SVC(C= .1, kernel='linear', gamma= 1)
```

In [37]:

```
clf.fit(x_train , y_train.values.ravel())
```

Out[37]:

```
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [38]:

```
y_pred = clf.predict(x_test)
```

In [39]:

```
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8582677165354331
```

In [40]:

```
clf =  SVC(C= .1, kernel='rbf', gamma= 1)
clf.fit(x_train , y_train.values.ravel())
```

Out[40]:

```
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf', max_iter=-1,
    probability=False, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

In [41]:

```
y_pred = clf.predict(x_test)
```

In [42]:

```
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.531496062992126
```

In [43]:

```
clf =  SVC(C= .1, kernel='poly', gamma= 1)
clf.fit(x_train , y_train.values.ravel())
```

Out[43]:

```
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [44]:

```
y_pred = clf.predict(x_test)
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9133858267716536
```

In [45]:

```
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
```

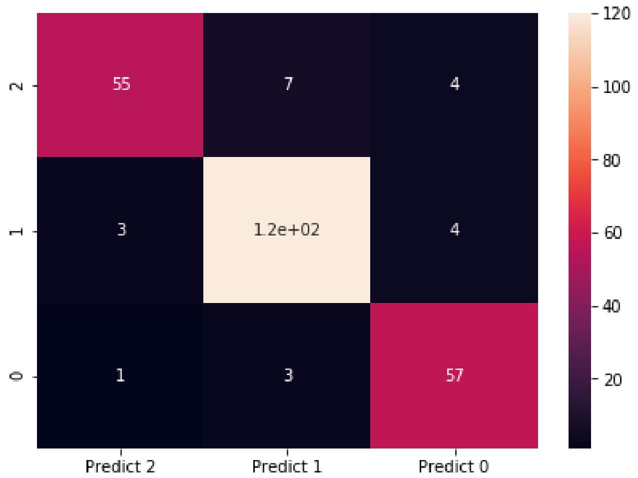
```
Confusion Matrix:
 [[ 55   7   4]
 [  3 120   4]
 [  1   3  57]]
```

In [46]:

```
knnm=metrics.confusion_matrix(y_test,y_pred)
knn_m = pd.DataFrame(knnm, index = [i for i in ["2","1","0"]],columns = [i for i in ["Predict
2","Predict 1","Predict 0"]])
plt.figure(figsize = (7,5))
sns.heatmap(knn_m, annot=True)
```

Out[46]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x2400405fa88&gt;
```



In [47]:

```
print("Classification Report")
print(metrics.classification_report(y_test,y_pred, labels=[2,1, 0]))
```

```
Classification Report
              precision    recall  f1-score   support

           2       0.88      0.93      0.90        61
           1       0.92      0.94      0.93       127
           0       0.93      0.83      0.88        66

    accuracy                           0.91       254
   macro avg       0.91      0.90      0.91       254
weighted avg       0.91      0.91      0.91       254
```

In [48]:

```
clf =  SVC(C= .1, kernel='sigmoid', gamma= 1)
clf.fit(x_train , y_train.values.ravel())
```

Out[48]:

```
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='sigmoid',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [49]:

```
y_pred = clf.predict(x_test)
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5236220472440944
```

> After trying the SVM algorithm in the different kernels. Following inference was made:-
>
> - The highest model accuracy is when the algorithm is running in a polynomial kernel.
> - The highest accuracy score was 91.
> - Confusion matrix and classification reports were found for the same.

K-fold and cross-validation

In [50]:

```
kfold = KFold(n_splits=10, random_state=7,shuffle=True)
clf= SVC(C= .1, kernel='poly', gamma= 1)
model=clf.fit(x_train , y_train.values.ravel())
results = cross_val_score(model, XScaled, Y.values.ravel() , cv=kfold)
print(results)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
[0.94117647 0.91764706 0.90588235 0.95294118 0.96470588 0.83529412
 0.92857143 0.85714286 0.86904762 0.91666667]
Accuracy: 90.891% (4.039%)
```

> After using K-fold validation.
>
> - The range of accuracy of the model was from 86% to 94% at a 95% confidence level.

Identifying Eigen vectors using PCA

In [51]:

```
covMatrix = np.cov(XScaled,rowvar=False)
print(covMatrix)
```

```
[[ 1.00118343  0.62170187  0.25150523 -0.82287347  0.7977645   0.83951746
   0.92798524  0.05270433  0.14456452 -0.01186527  0.04537164]
 [ 0.62170187  1.00118343  0.45056889 -0.79072934  0.79498064  0.72100219
   0.5371705  -0.1810333   0.04877731  0.17403799  0.47181974]
 [ 0.25150523  0.45056889  1.00118343 -0.18025396  0.31933203  0.14388275
   0.18992805  0.29592367  0.01545721  0.04354026  0.14408905]
 [-0.82287347 -0.79072934 -0.18025396  1.00118343 -0.93782312 -0.95620413
  -0.76693543  0.10360323 -0.05205875 -0.18591103 -0.21697531]
 [ 0.7977645   0.79498064  0.31933203 -0.93782312  1.00118343  0.94814159
   0.77989661  0.11243163  0.03604752  0.19549063  0.08669654]
 [ 0.83951746  0.72100219  0.14388275 -0.95620413  0.94814159  1.00118343
   0.79701241 -0.0166278   0.07706469  0.20181158  0.10388468]
 [ 0.92798524  0.5371705   0.18992805 -0.76693543  0.77989661  0.79701241
   1.00118343  0.19166642  0.16656805 -0.05603902 -0.1182971 ]
 [ 0.05270433 -0.1810333   0.29592367  0.10360323  0.11243163 -0.0166278
```

```
   0.19166642  1.00118343 -0.08840848 -0.12656621 -0.8035581 ]
 [ 0.14456452  0.04877731  0.01545721 -0.05205875  0.03604752  0.07706469
   0.16656805 -0.08840848  1.00118343 -0.03506456  0.09698477]
 [-0.01186527  0.17403799  0.04354026 -0.18591103  0.19549063  0.20181158
  -0.05603902 -0.12656621 -0.03506456  1.00118343  0.20533271]
 [ 0.04537164  0.47181974  0.14408905 -0.21697531  0.08669654  0.10388468
  -0.1182971  -0.8035581   0.09698477  0.20533271  1.00118343]]
```

In [52]:

```
pca = PCA(n_components=6)
pca.fit(XScaled)
```

Out[52]:

```
PCA(copy=True, iterated_power='auto', n_components=6, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

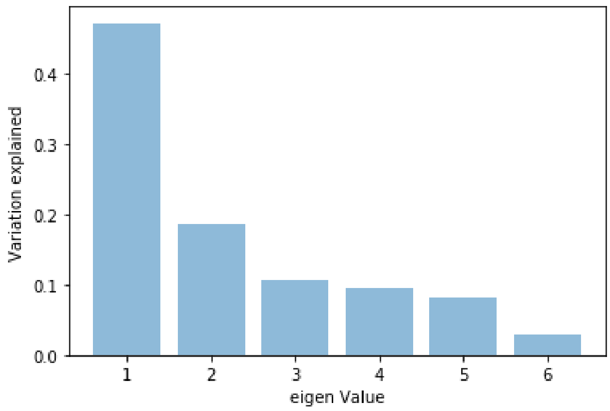
In [53]:

```
print(pca.explained_variance_)
```

```
[5.18662721 2.05550394 1.17384819 1.05528084 0.88494564 0.32133759]
```

In [54]:

```
plt.bar(list(range(1,7)),pca.explained_variance_ratio_,alpha=0.5, align='center')
plt.ylabel('Variation explained')
plt.xlabel('eigen Value')
plt.show()
```



In [55]:

```
pca3 = PCA(n_components=6)
pca3.fit(XScaled)
print(pca3.components_)
print(pca3.explained_variance_ratio_)
Xpca3 = pca3.transform(XScaled)
```

```
[[ 0.39611623  0.36809097  0.14874228 -0.42089468  0.42119261  0.41767613
   0.37539781 -0.00394397  0.05149428  0.07315524  0.086749  ]
 [ 0.1160752  -0.20374372  0.07430875  0.05260597  0.06271017  0.01894014
   0.22546146  0.6382345  -0.06066981 -0.23779087 -0.64345404]
 [-0.17512039  0.22052399  0.65318935  0.06155997  0.07617988 -0.09618641
  -0.22467436  0.27090577 -0.45675851  0.36913349  0.08027758]
 [ 0.04984377  0.14037895  0.56535966  0.14808997 -0.09600545 -0.18739837
   0.02838387  0.05231413  0.59254708 -0.44522476  0.19756629]
 [-0.07623596 -0.1169546  -0.01159183  0.03607123  0.02585673  0.0410813
  -0.02828491  0.16883607  0.63259572  0.72603927 -0.14178125]
 [-0.52481012  0.39545311 -0.22829585 -0.19894854  0.31027591  0.17083502
  -0.45312434  0.2015486   0.17776456 -0.24707486 -0.11156243]]
[0.47095422 0.18664312 0.10658733 0.09582122 0.08035451 0.02917798]
```

In [56]:

```
Xpca3
```

Out[56]:

```
array([[ 0.12968519, -0.34629014,  0.22002525,  0.17720568,  0.04526196,
        -0.76880923],
       [-1.38405839, -0.43238023, -0.01926176,  0.47611585,  0.51626564,
        -0.19116416],
       [ 2.80683385,  0.23658889, -0.92230398,  1.06053268,  0.48296298,
         0.47390012],
       ...,
       [ 3.3245831 , -0.28744424,  0.18466963,  0.26263644, -1.57503553,
         0.23190269],
       [-2.49331101, -1.01044899,  1.01014057, -1.37892084,  0.2449125 ,
        -0.53232961],
       [-3.77918478,  0.18938257,  0.55137633, -1.1286787 ,  0.16053356,
        -0.33219449]])
```

In [57]:

```
Xpca3.shape
```

Out[57]:

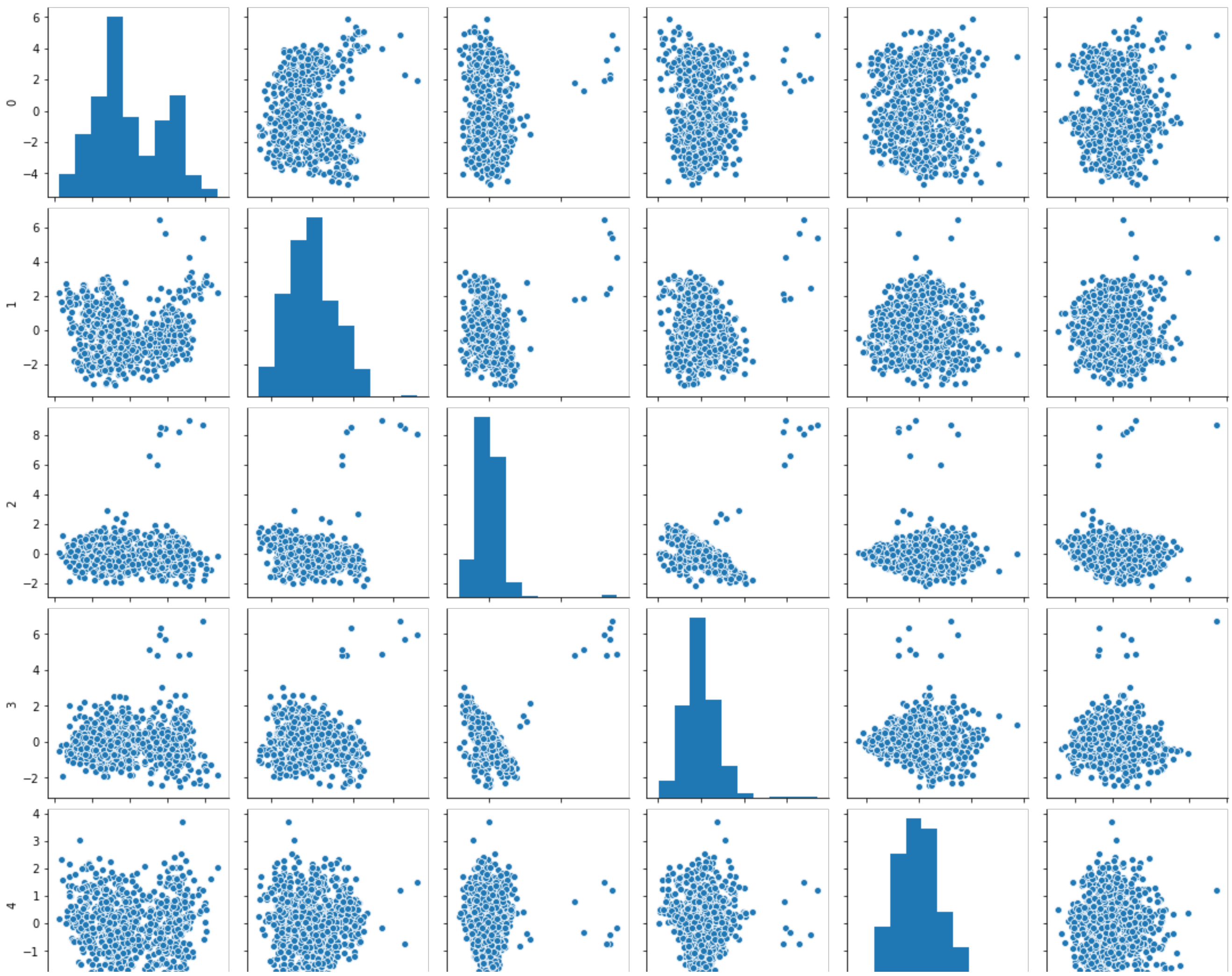
```
(846, 6)
```

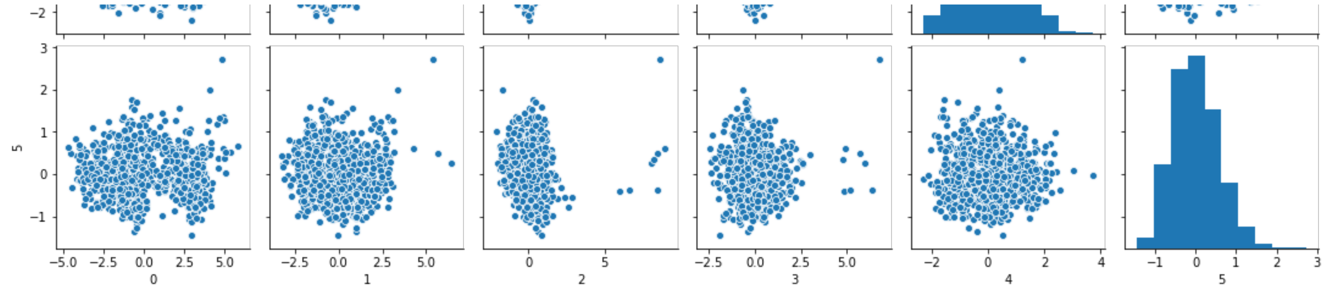
In [58]:

```
sns.pairplot(pd.DataFrame(Xpca3))
```

Out[58]:

```
<seaborn.axisgrid.PairGrid at 0x240050e1c48>
```





In [59]:

```
x_train, x_test, y_train, y_test = train_test_split(Xpca3, Y, test_size=0.3, random_state=7)
```

SVM Analysis using the eigen vectors

In [60]:

```
clf =  SVC(C= .1, kernel='linear', gamma= 1)
```

In [61]:

```
clf.fit(x_train , y_train.values.ravel())
```

Out[61]:

```
SVC(C=0.1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [62]:

```
y_pred = clf.predict(x_test)
```

In [63]:

```
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8110236220472441
```

In [64]:

```
clf =  SVC(C= .1, kernel='rbf', gamma= 1)
clf.fit(x_train , y_train.values.ravel())
y_pred = clf.predict(x_test)
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5511811023622047
```

In [65]:

```
clf =  SVC(C= .1, kernel='poly', gamma= 1)
clf.fit(x_train , y_train.values.ravel())
y_pred = clf.predict(x_test)
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.8346456692913385
```

In [66]:

```
print("Confusion Matrix:\n",confusion_matrix(y_test,y_pred))
```

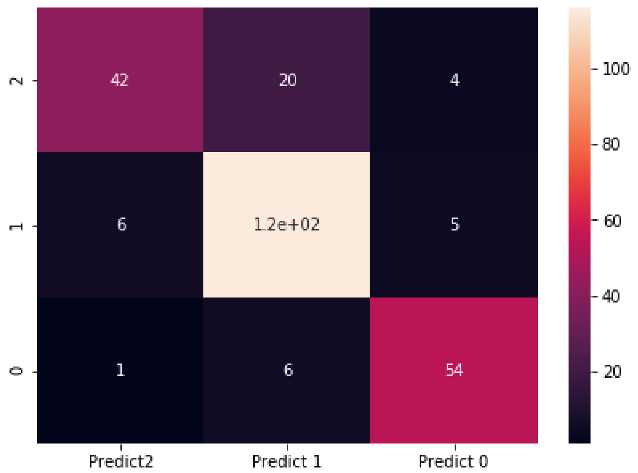
```
Confusion Matrix:
 [[ 42  20   4]
 [  6 116   5]
 [  1   6  54]]
```

In [67]:

```python
knnm=metrics.confusion_matrix(y_test,y_pred)
knn_m = pd.DataFrame(knnm, index = [i for i in ["2","1","0"]],columns = [i for i in
["Predict2","Predict 1","Predict 0"]])
plt.figure(figsize = (7,5))
sns.heatmap(knn_m, annot=True)
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2400c2d8848>
```



In [68]:

```python
print("Classification Report")
print(metrics.classification_report(y_test,y_pred, labels=[2,1, 0]))
```

```
Classification Report
              precision    recall  f1-score   support

           2       0.86      0.89      0.87        61
           1       0.82      0.91      0.86       127
           0       0.86      0.64      0.73        66

    accuracy                           0.83       254
   macro avg       0.84      0.81      0.82       254
weighted avg       0.84      0.83      0.83       254
```

In [69]:

```python
clf =  SVC(C= .1, kernel='sigmoid', gamma= 1)
clf.fit(x_train , y_train.values.ravel())
y_pred = clf.predict(x_test)
print("Accuracy:",accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.5118110236220472
```

> After running the SVM model with eigenvectors in different kernels. The following was inferred.
>
> - The maximum accuracy was again found in the polynomial kernel.
> - The accuracy of this model was 83.
> - A drop of around 6% was found but this expected as the columns are reduced

K-fold validadtion with PCA value

In [70]:

```
kfold = KFold(n_splits=10, random_state=7,shuffle=True)
clf= SVC(C= .1, kernel='poly', gamma= 1)
model=clf.fit(x_train , y_train.values.ravel())
results = cross_val_score(model, Xpca3, Y.values.ravel() , cv=kfold)
print(results)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0, results.std()*100.0))
```

```
[0.85882353 0.87058824 0.81176471 0.84705882 0.84705882 0.75294118
 0.80952381 0.82142857 0.82142857 0.89285714]
Accuracy: 83.335% (3.706%)
```

> After using K-fold validation with eigen vectors.
>
> - The range of accuracy of the model was from 80% to 87% at 95% confidence level.

> Even after only selecting six column that covered 95% of data using PCA.
>
> - The accuracy dropped from 91 to 83
> - This is result was within the expected region.