Unsupervised Learning Project

"Corgie" Model Vehicles Classification

by Sandesh Balyan

Table of Contents

Description

The data contains features extracted from the silhouette of vehicles in different angles. Four "Corgie" model vehicles were used for the experiment: a double decker bus, Cheverolet van, Saab 9000 and an Opel Manta 400 cars. This particular combination of vehicles was chosen with the expectation that the bus, van and either one of the cars would be readily distinguishable, but it would be more difficult to distinguish between the cars.

Objective

Apply dimensionality reduction technique – PCA and train a model using principle components instead of training the model using just the raw data.

We will build an SVM model by eliminating features manually and then using PCA and compare the accuracy scores for these models

1. Exploratory Data Analysis and Preprocessing

1.1 Loading Data

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import itertools
import time
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [2]:

```
data=pd.read_csv('vehicle-1.csv')
data.head()
```

Out[2]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_as
0	95	48.0	83.0	178.0	72.0	
1	91	41.0	84.0	141.0	57.0	
2	104	50.0	106.0	209.0	66.0	
3	93	41.0	82.0	159.0	63.0	
4	85	44.0	70.0	205.0	103.0	

In [3]:

```
data.shape
```

Out[3]:

```
(846, 19)
```

There are total 846 rows and 19 attributes(including dependent variable) in the dataset

In [4]:

```
data.info()
```

```
<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
```

1. All the data types are of numeric types and no action needed to handle any of them
2. There are few null value in some variables

1.1.1 Handling null in dataset

In [5]:

```
data.isnull().sum()
```

Out[5]:

```
compactness                  0
circularity                  5
distance_circularity         4
radius_ratio                 6
pr.axis_aspect_ratio         2
max.length_aspect_ratio      0
scatter_ratio                1
elongatedness                1
pr.axis_rectangularity       3
max.length_rectangularity    0
scaled_variance              3
scaled_variance.1            2
scaled_radius_of_gyration    2
scaled_radius_of_gyration.1  4
skewness_about               6
skewness_about.1             1
skewness_about.2             1
hollows_ratio                0
class                        0
dtype: int64
```

Dropping null values

In [6]:

```
tempData = data.copy()
data = data.dropna()
```

1.1.2 Checking distribution of observations in each class

In [7]:

```
data.groupby('class').count()
```

Out[7]:

class	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.lengt
bus	205	205	205	205	205	
car	413	413	413	413	413	
van	195	195	195	195	195	

1. Highest number of observations are for car, probably because it contains two different categories of cars (Saab 9000 and Opel Manta 400)
2. buses and vans have nearly the same share.

1.1.3 Checking duplicates in dataset

In [8]:

```
data.duplicated().value_counts()
```

Out[8]:

```
False    813
dtype: int64
```

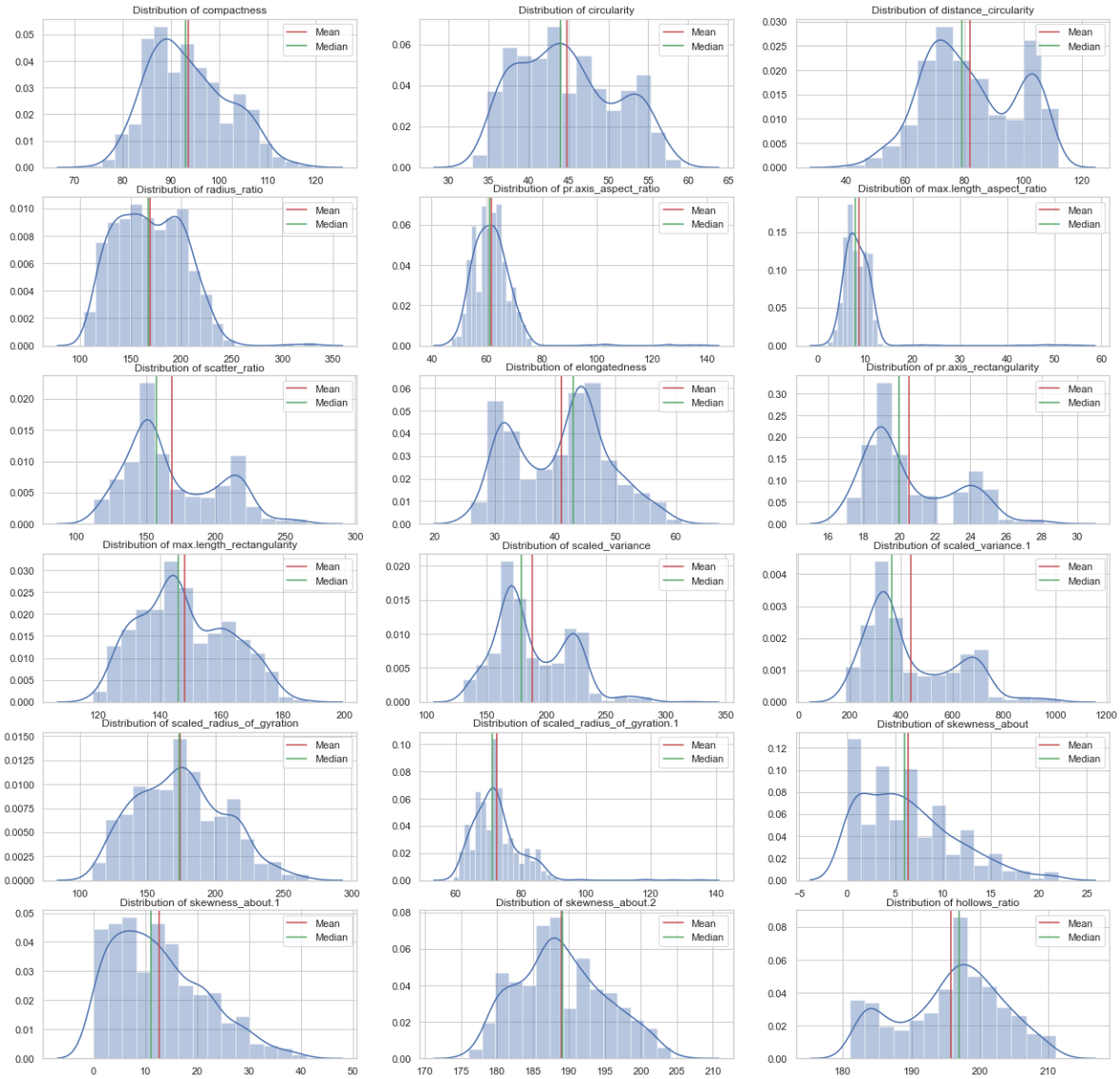
There are no duplicates in this data

1.2 Exploratory Data Analysis

1.2.1 Distribution of numerical variables

In [9]:

```
ncol = 3
nrow = len(data.columns[:-1]) / ncol

fig = plt.figure(1,figsize=(21,21))
for i,col in enumerate(data.columns[:-1]):
    sns.set(style = 'whitegrid')
    ax=plt.subplot(nrow,ncol,i+1)
    sns.distplot(data[col])
    ax.set_title('Distribution of '+col)
    ax.set_xlabel(None)
    ax.axvline(data[col].mean(),c='r',label='Mean')
    ax.axvline(data[col].median(),c='g',label='Median')
    ax.legend(loc="upper right")
```



Insights:

1. Median and mean values are same or nearby in majority of the attributes
2. Most of the variables depict atleast 2 peaks indicating presence of atleast 2 clusters i.e. atleast 2 normalised curves
3. In some cases example skewness_about, skewness_about1, radious ratio, pr axis aspect ratio, max length aspect ratio, data is right skewed and indicated presence of outliers.

1.2.2 Five Points Summary and Outliers

In [10]:

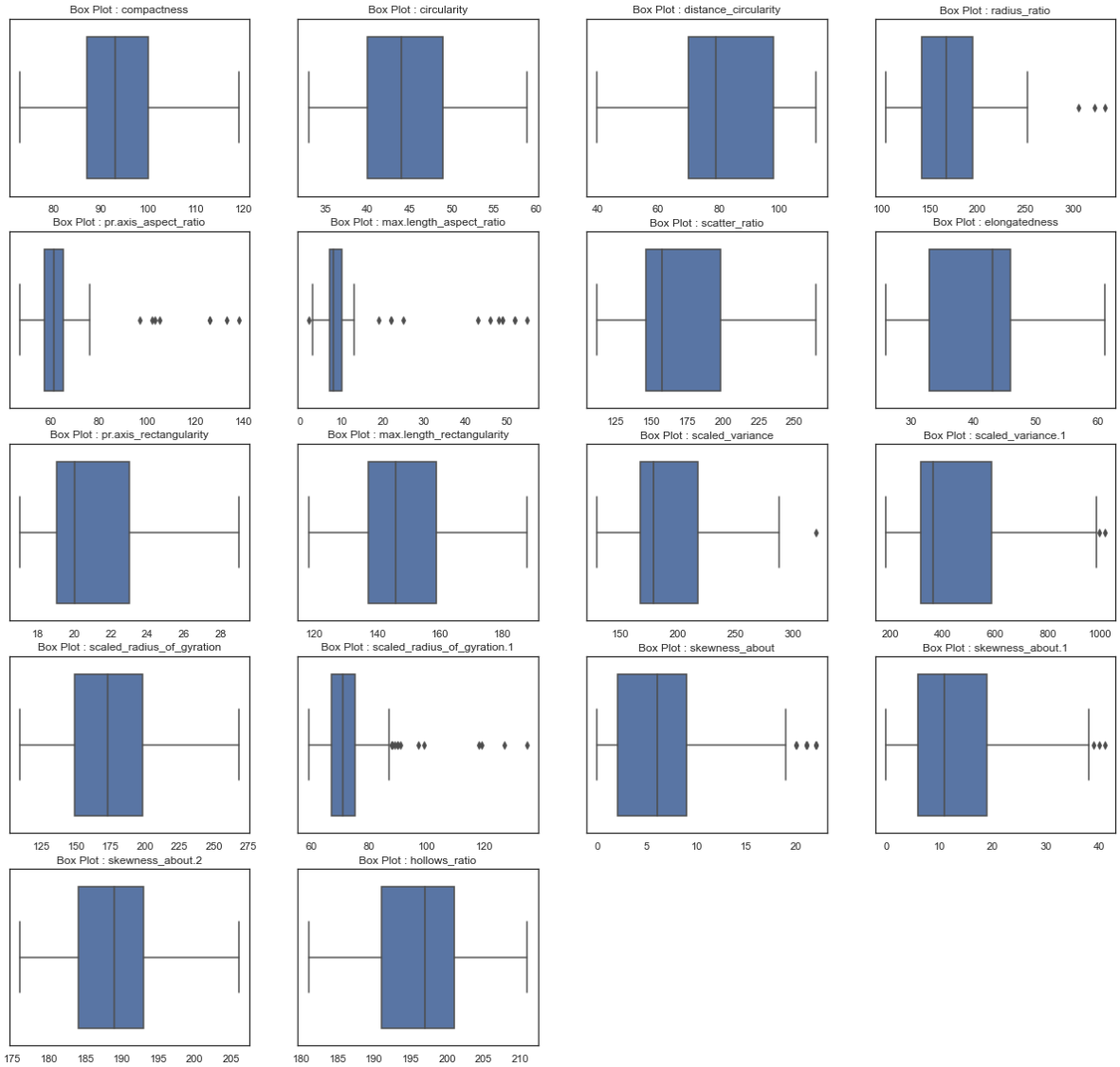
```
data.describe().transpose()
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
compactness	813.0	93.656827	8.233751	73.0	87.0	93.0	100.0	119.0
circularity	813.0	44.803198	6.146659	33.0	40.0	44.0	49.0	59.0
distance_circularity	813.0	82.043050	15.783070	40.0	70.0	79.0	98.0	112.0
radius_ratio	813.0	169.098401	33.615402	104.0	141.0	167.0	195.0	333.0
pr.axis_aspect_ratio	813.0	61.774908	7.973000	47.0	57.0	61.0	65.0	138.0
max.length_aspect_ratio	813.0	8.599016	4.677174	2.0	7.0	8.0	10.0	55.0
scatter_ratio	813.0	168.563346	33.082186	112.0	146.0	157.0	198.0	265.0
elongatedness	813.0	40.988930	7.803380	26.0	33.0	43.0	46.0	61.0
pr.axis_rectangularity	813.0	20.558426	2.573184	17.0	19.0	20.0	23.0	29.0
max.length_rectangularity	813.0	147.891759	14.504648	118.0	137.0	146.0	159.0	188.0
scaled_variance	813.0	188.377614	31.165873	130.0	167.0	179.0	217.0	320.0
scaled_variance.1	813.0	438.382534	175.270368	184.0	318.0	364.0	586.0	1018.0
scaled_radius_of_gyration	813.0	174.252153	32.332161	109.0	149.0	173.0	198.0	268.0
scaled_radius_of_gyration.1	813.0	72.399754	7.475994	59.0	67.0	71.0	75.0	135.0
skewness_about	813.0	6.351784	4.921476	0.0	2.0	6.0	9.0	22.0
skewness_about.1	813.0	12.687577	8.926951	0.0	6.0	11.0	19.0	41.0
skewness_about.2	813.0	188.979090	6.153681	176.0	184.0	189.0	193.0	206.0
hollows_ratio	813.0	195.729397	7.398781	181.0	191.0	197.0	201.0	211.0

In [11]:

```
fig = plt.figure(1,figsize=(21,20))
ncol = 4
nrow = np.ceil(len(data.columns[:-1]) / ncol)

for i,col in enumerate(data.columns[:-1]):
    sns.set(style='white')
    ax=plt.subplot(nrow,ncol,i+1)
    sns.boxplot(data[col])
    ax.set_title('Box Plot : '+col)
    ax.set_xlabel(None)
```



Insights:

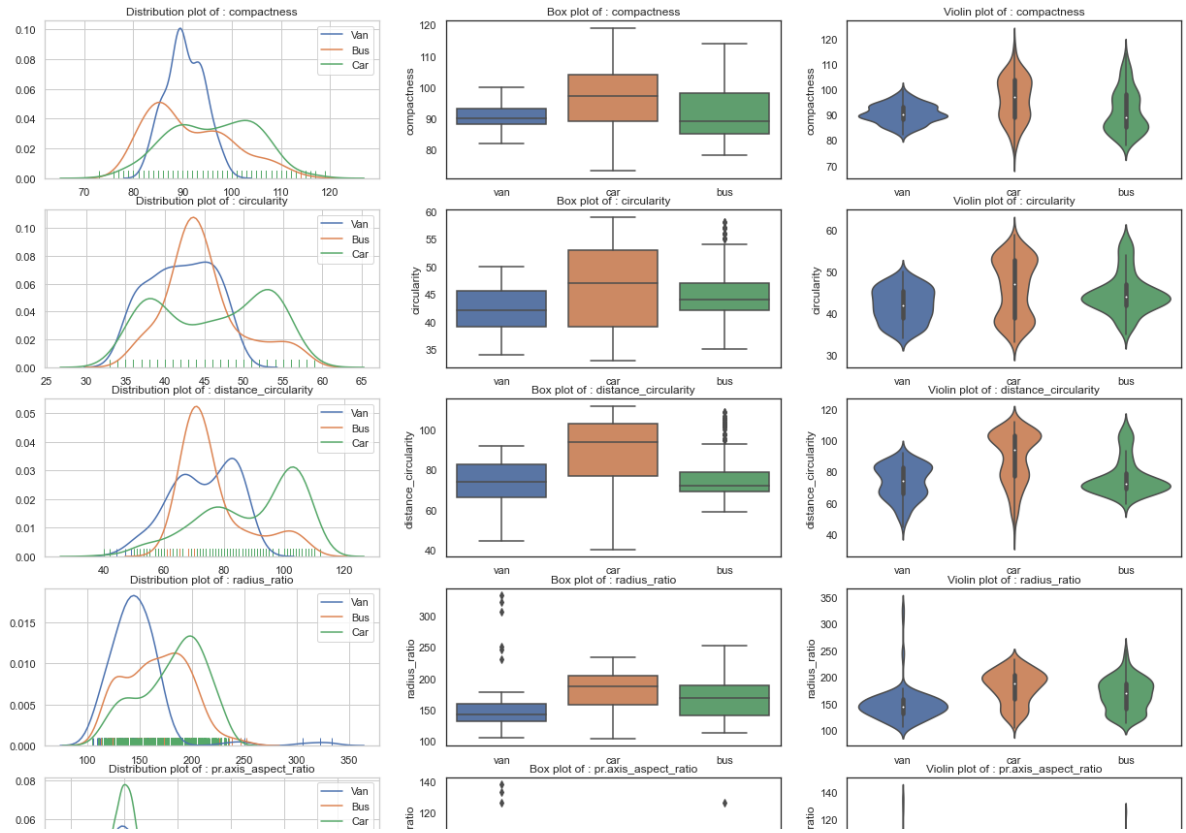
1. There are outliers in some of the variables viz. 'scaled_radius_of_gyration_1', 'scaled_variance', 'scaled_variance1', 'skewness_about', 'skewness_about1', radious_raio, pr_axis_aspect_ratio, max_length_aspect_ratio

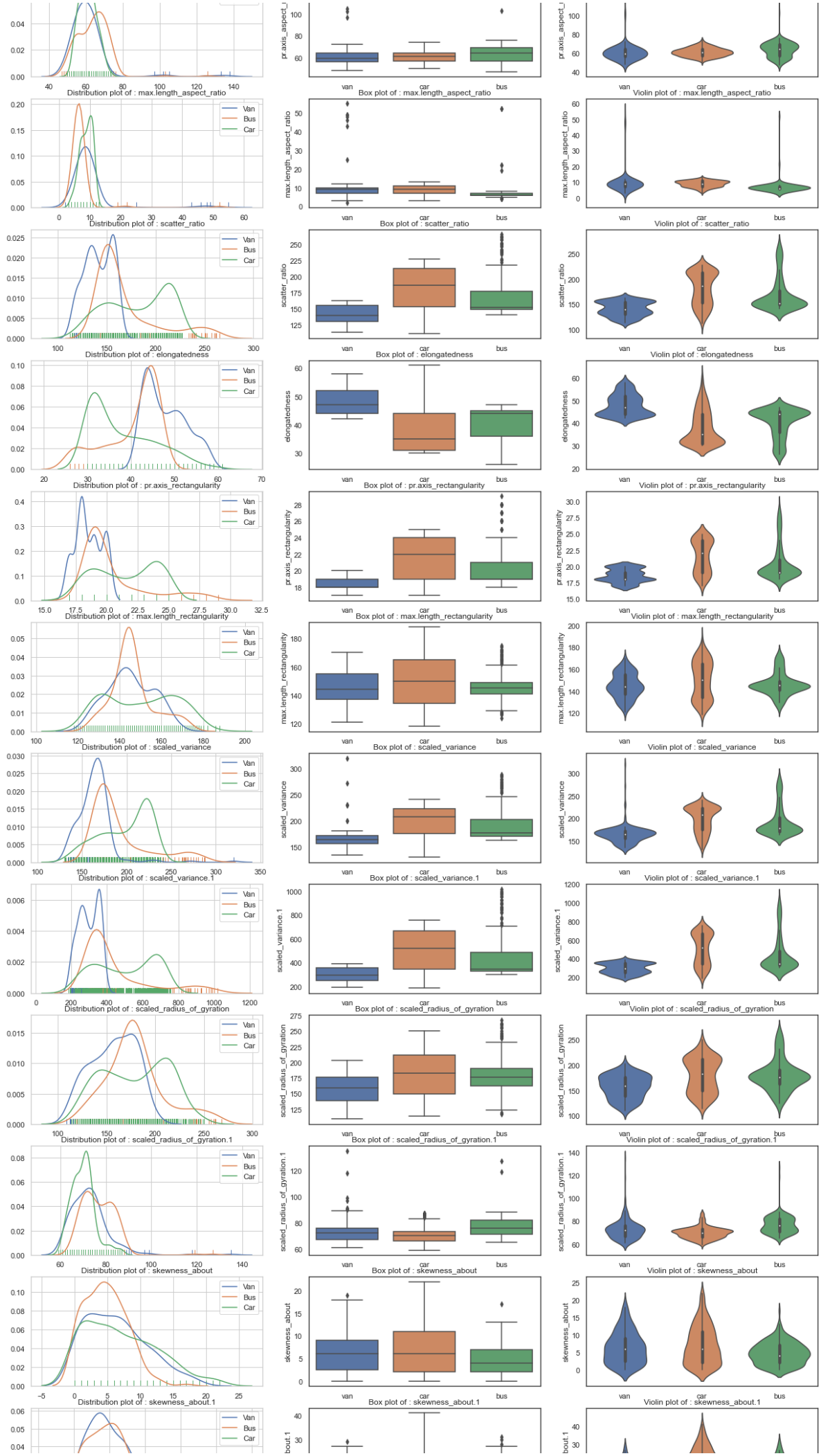
2. Mean and medians are same for almost all the variables

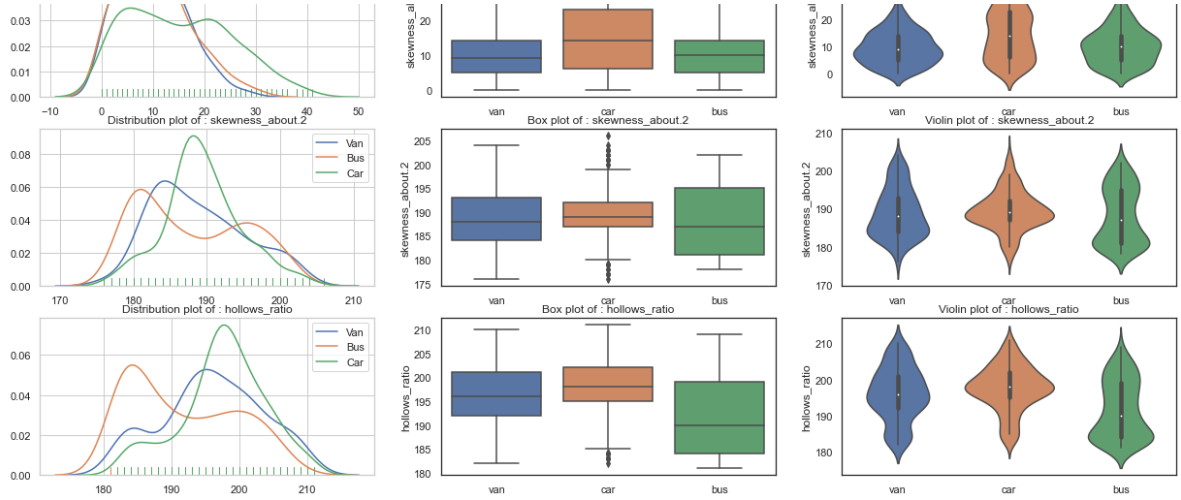
1.2.3 Numeric columns distributed over class label

In [12]:

```
fig = plt.figure(1,(21,64))
c=0

data_van = data.loc[data['class']=='van']
data_car = data.loc[data['class']=='car']
data_bus = data.loc[data['class']=='bus']

for i,col in enumerate(data.columns[:-1]):
    c += 1
    sns.set(style='whitegrid')
    ax1 = plt.subplot(18,3,c)
    sns.distplot(data_van[col],hist=False, rug=True,label='Van')
    sns.distplot(data_bus[col],hist=False, rug=True,label='Bus')
    sns.distplot(data_car[col],hist=False, rug=True,label='Car')
    ax1.set_title('Distribution plot of : ' + col)
    ax1.set_xlabel(None)
    ax1.legend(loc="upper right")

    c+=1
    sns.set(style='white')
    ax2 = plt.subplot(18,3,c)
    sns.boxplot(y=data[col], x=data['class'])
    ax2.set_title('Box plot of : ' + col)
    ax2.set_xlabel(None)

    c+=1
    sns.set(style='white')
    ax3 = plt.subplot(18,3,c)
    sns.set(style='white')
    sns.violinplot(y=data[col], x=data['class'])
    ax3.set_title('Violin plot of : ' + col)
    ax3.set_xlabel(None)
```




Distribution plot of : max.length_aspect_ratio
Box plot of : max.length_aspect_ratio
Violin plot of : max.length_aspect_ratio
Distribution plot of : scatter_ratio
Box plot of : scatter_ratio
Violin plot of : scatter_ratio
Distribution plot of : elongatedness
Box plot of : elongatedness
Violin plot of : elongatedness
Distribution plot of : pr.axis_rectangularity
Box plot of : pr.axis_rectangularity
Violin plot of : pr.axis_rectangularity
Distribution plot of : max.length_rectangularity
Box plot of : max.length_rectangularity
Violin plot of : max.length_rectangularity
Distribution plot of : scaled_variance
Box plot of : scaled_variance
Violin plot of : scaled_variance
Distribution plot of : scaled_variance.1
Box plot of : scaled_variance.1
Violin plot of : scaled_variance.1
Distribution plot of : scaled_radius_of_gyration
Box plot of : scaled_radius_of_gyration
Violin plot of : scaled_radius_of_gyration
Distribution plot of : scaled_radius_of_gyration.1
Box plot of : scaled_radius_of_gyration.1
Violin plot of : scaled_radius_of_gyration.1
Distribution plot of : skewness_about
Box plot of : skewness_about
Violin plot of : skewness_about
Distribution plot of : skewness_about.1
Box plot of : skewness_about.1
Violin plot of : skewness_about.1
Van
Bus
Car
van
car
bus



Inisghts:

These plots could give us some idea about some segregation between clusters based upon min, median and max of each feature over class label. Some feature might be able to distinguish between all the 3 clusters but in some cases there could be an overlap

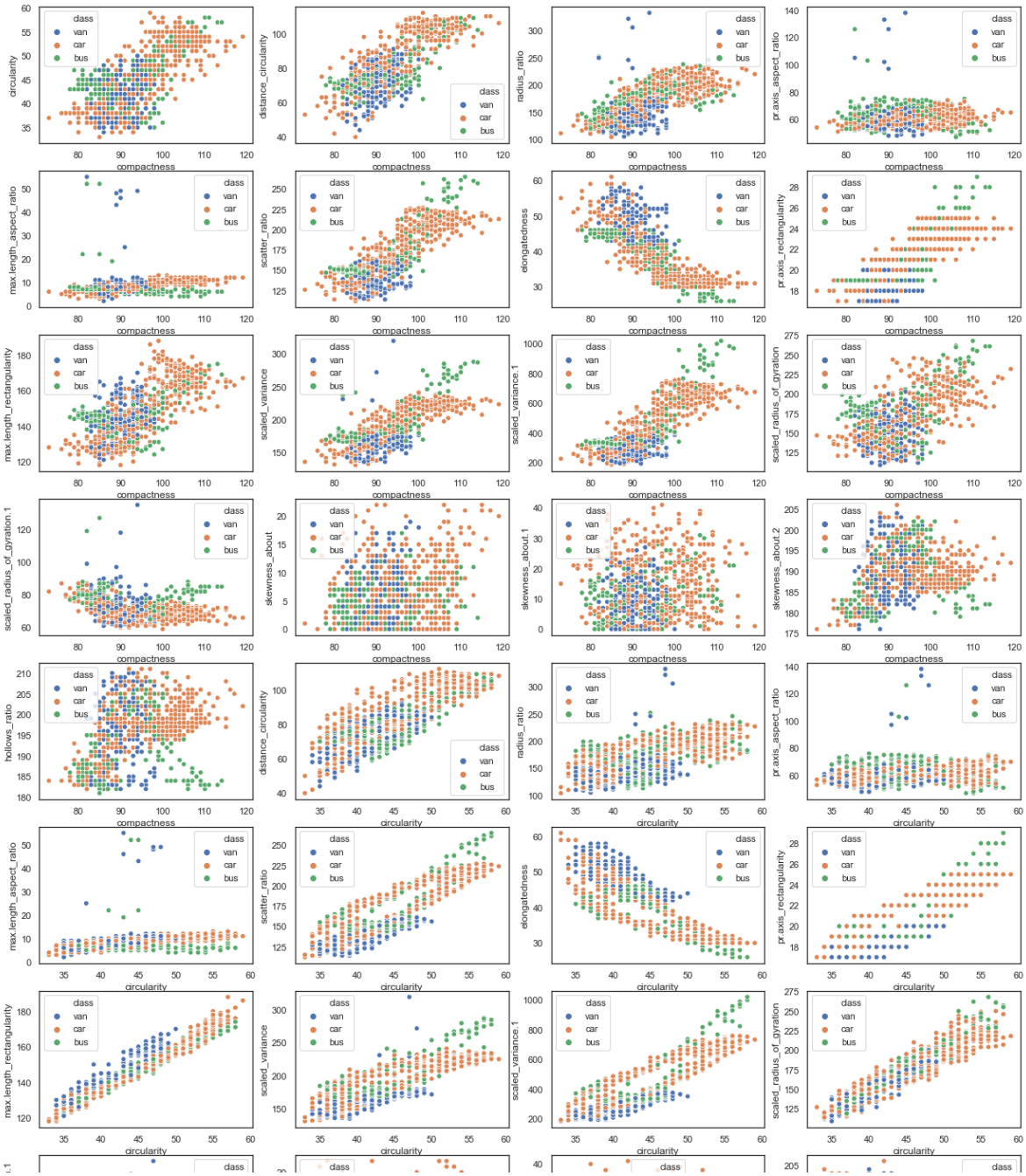
1. Compactness: Overall spread and median is highest for car and minimum for van for this variable. Median value is almost same for van and bus
2. Circularity: Median is highest for car and minimum for van. IQR is also max for car and smallest for bus. There are some outliers for this variable in bus
3. Distance circularity: Overall spread and IQR is highest for car and smallest for bus. Some outliers are present for bus in this variable. Median is again distiniguishable for car than other 2 clusters
4. Radius ratio: IQR and median are smallest for van and largest for car. bus has maximum value for this variable. Some outliers in van for this variable
5. Pr axis aspect ratio: median and IQR are almost the same for all clusters. cannot be distinguished. there are outliers in van and bus for this variable
6. max length aspect ratio:Data is overlapping for all clusters and cannot be distinguished. there are some outliers
7. scatter ratio: median and IQR is max for car followed by bus folloed by van. Clusters can be distinguished using this variable
8. elongatedness: Median is highest for van and lowest for car. data is overlapping beteween car and bus
9. pr axis rectangularity: median value and IQR is highest for car and lowest for van. data overlaps between car and bus.
10. max length rectangularity: Data overlaps for all 3 clusters and cannot be distinguished
11. scaled_variance: Overlap of data between car and bus. median is highest for car and lowest for van. Vans can be identified separately
12. scaled variance 1: Overlap of data between car and bus. median is highest for car and lowest for van. Vans can be identified separately
13. scaled radious of gyration: Data fairly overlaps and clusetrs cannot be distinguished
14. scaled radius of gyration 1: Data fairly overlaps and clusters cannot be distinguished
15. skewness about : Data fairly overlaps and clusters cannot be distinguished
16. skewness about 1 : Data fairly overlaps and clusters cannot be distinguished
17. skewness about 2 : Data fairly overlaps and clusters cannot be distinguished
18. Hollow ratios: Data fairly overlaps and clusters cannot be distinguished

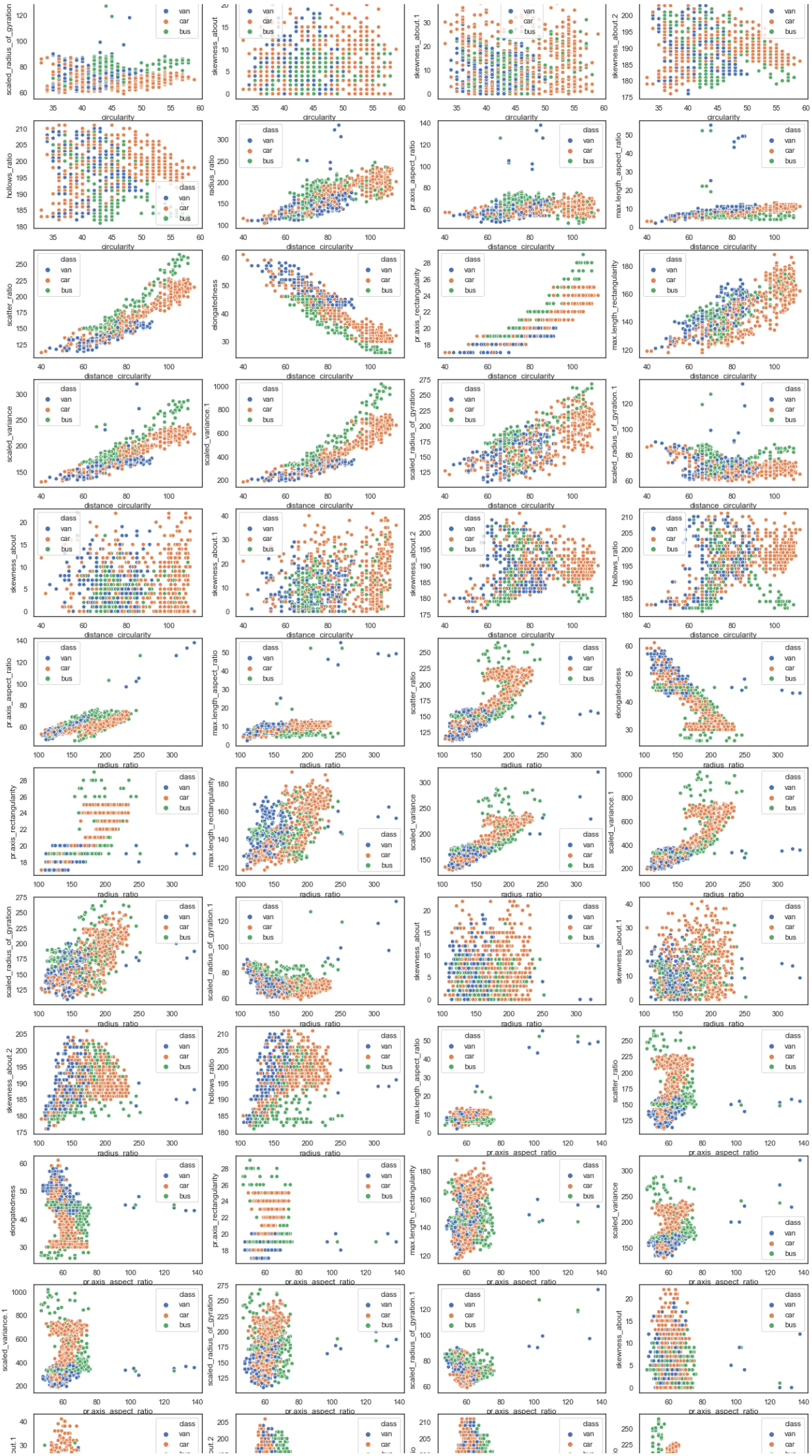
1.2.4 Scatterplots between pair of numeric columns and hue = class

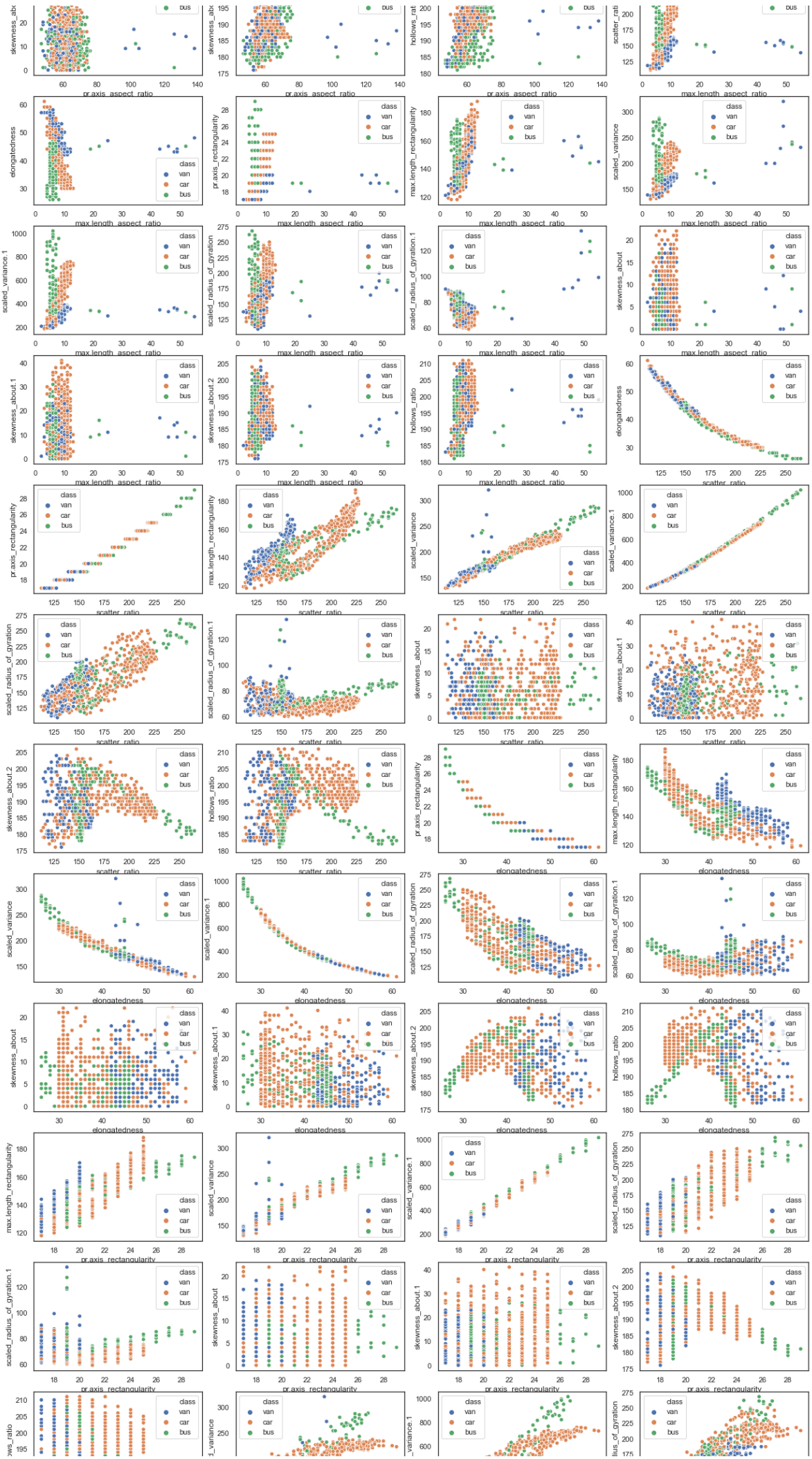
With scatter plots between variables we can try to understand which combination of variables clearly distinguish between clsuters

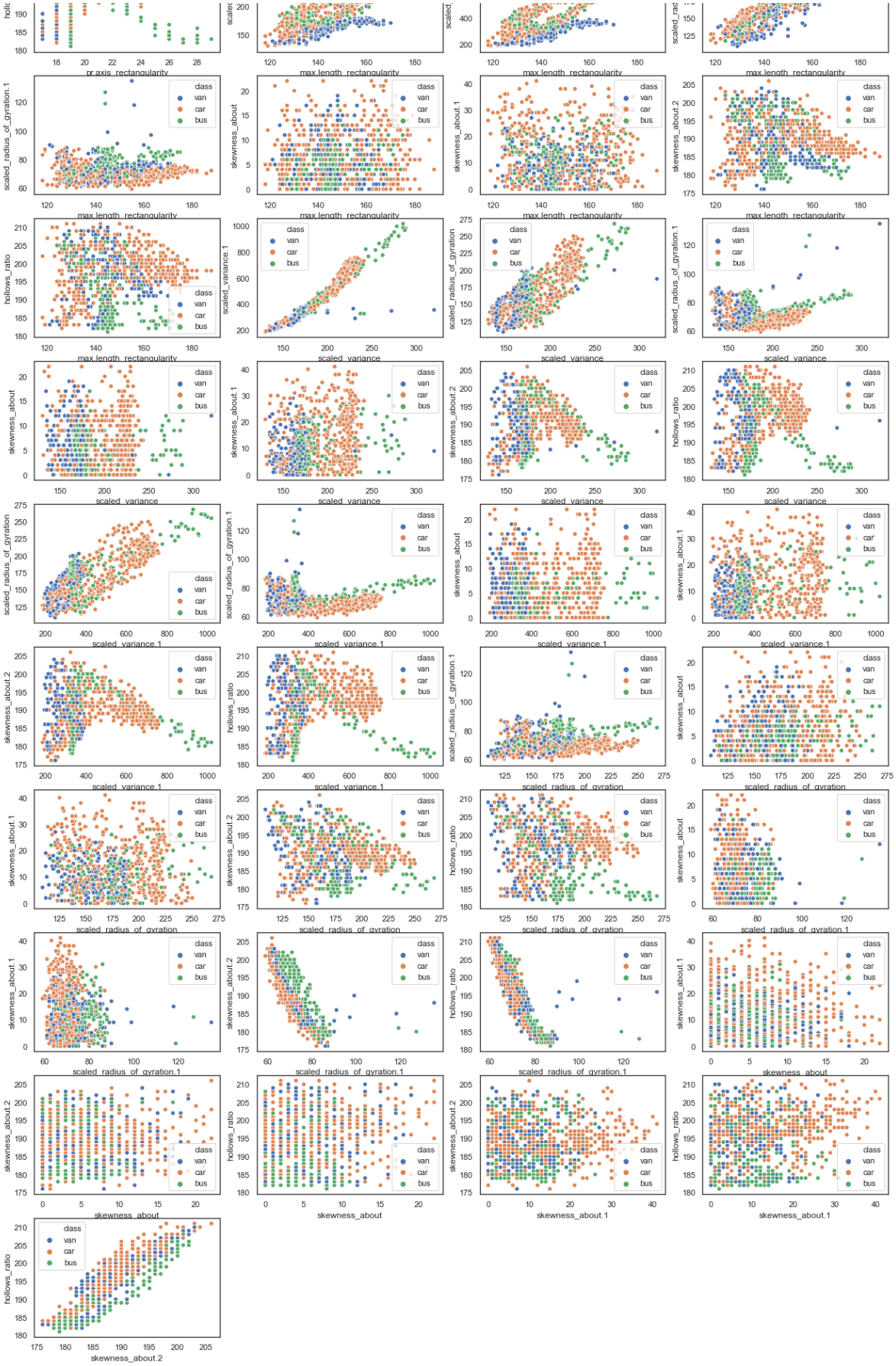
In [13]:

```
#col = df_scaled.columns[:-1]
col = list(data.columns[:-1])
combos = itertools.combinations(col,2)
counter=153
#for pair in combos:
#    counter+=1
#print(counter)

fig = plt.figure(1,figsize=(21,140))
ncol = 4
nrow = np.ceil(counter / ncol)

for i,pair in enumerate(combos):
    sns.set(style='white')
    ax=plt.subplot(nrow,ncol,i+1)
    sns.scatterplot(x = pair[0], y =pair[1],data=data,hue=data['class'])
```




scaled_radius_of_gyration
skewness_about
skewness_about.1
skewness_about.2
hollows_ratio
radius_ratio
pr.axis_aspect_ratio
max.length_aspect_ratio
scatter_ratio
elongatedness
pr.axis_rectangularity
max.length_rectangularity
scaled_variance
scaled_variance.1
scaled_radius_of_gyration.1
circularity
distance_circularity
radius_ratio
class
van
car
bus


class
van
car
bus
pr.axis_aspect_ratio
max.length_aspect_ratio
elongatedness
pr.axis_rectangularity
max.length_rectangularity
scaled_variance
scaled_variance.1
scaled_radius_of_gyration
scaled_radius_of_gyration.1
skewness_about
skewness_about.1
skewness_about.2
hollows_ratio
scatter_ratio



Insights:

1. There is no combination of 2 variables which clearly distinguishes between clusters clearly.
2. There is multicollinearity in features i.e many variables are positively related to each other ex. hollows ratio and skewness about 2, scaled variance and scaled variance 1 and many other.
3. We will take care of multocolinearity in later steps

1.3 Data Preprocessing

In this section we will detect the outliers and impute the outliers with central values. This could change the distribution of the variable, however since there are not lot of outliers in the dataset, impact will be too less.

1.3.1 Handling Outliers

1.3.1.1 Detecting and Quantifying outliers

In [14]:

```
def handle_outlier(x):
    IQR = data[x].quantile(0.75) - data[x].quantile(0.25)
    min_range = data[x].quantile(0.25) - 1.5*IQR
    max_range = data[x].quantile(0.75) + 1.5*IQR
    outliers = data[(data[x] < min_range) | (data[x] > max_range)]
    return outliers
```

In [15]:

```
df_outlier = pd.DataFrame()
list_dict = []
for feature in data.columns[:-1]:
    df = handle_outlier(feature)
    x = {'Feature name': feature,
         'Total count': data[feature].count(),
          'Outlier Count': len(df),
          '% Outliers': (len(df)/len(data)*100),
          'min_outlier': df[feature].min(),
          'max_outlier': df[feature].max()
         }
    list_dict.append(x)

df_outlier = df_outlier.append(list_dict, True)
print('Outlier analysis on numerical variables')
df_outlier
```

```
Outlier analysis on numerical variables
```

Out[15]:

	Feature name	Total count	Outlier Count	% Outliers	min_outlier	max_outlier
0	compactness	813	0	0.000000	NaN	NaN
1	circularity	813	0	0.000000	NaN	NaN
2	distance_circularity	813	0	0.000000	NaN	NaN
3	radius_ratio	813	3	0.369004	306.0	333.0
4	pr.axis_aspect_ratio	813	8	0.984010	97.0	138.0
5	max.length_aspect_ratio	813	13	1.599016	2.0	55.0
6	scatter_ratio	813	0	0.000000	NaN	NaN
7	elongatedness	813	0	0.000000	NaN	NaN
8	pr.axis_rectangularity	813	0	0.000000	NaN	NaN
9	max.length_rectangularity	813	0	0.000000	NaN	NaN
10	scaled_variance	813	1	0.123001	320.0	320.0
11	scaled_variance.1	813	2	0.246002	998.0	1018.0
12	scaled_radius_of_gyration	813	0	0.000000	NaN	NaN
13	scaled_radius_of_gyration.1	813	15	1.845018	88.0	135.0
14	skewness_about	813	12	1.476015	20.0	22.0
15	skewness_about.1	813	3	0.369004	39.0	41.0
16	skewness_about.2	813	0	0.000000	NaN	NaN
17	hollows_ratio	813	0	0.000000	NaN	NaN

Pending Insights

1. There are very less number of outliers overall
2. Maximum outlier in any column are 1.8% which is very low

1.3.1.2 Imputing Outliers

In below steps we will impute the outliers with median values

In [16]:

```
data_copy = data.copy()
for feature in data.columns[:-1]:
    df = handle_outlier(feature)
    lst_replace = list(np.array(df[feature]))
    data[feature] = data[feature].replace(lst_replace,data[feature].median())
```

In [17]:

```
# Rechecking presence of outliers in all variables
df_outlier = pd.DataFrame()
list_dict = []
for feature in data.columns[:-1]:
    df = handle_outlier(feature)
    x = {'Feature name': feature,
         'Total count': data[feature].count(),
          'Outlier Count': len(df),
          '% Outliers': (len(df)/len(data)*100),
          'min_outlier': df[feature].min(),
          'max_outlier': df[feature].max()
         }
    list_dict.append(x)

df_outlier = df_outlier.append(list_dict, True)
print('Outlier analysis on numerical variables')
df_outlier
```

```
Outlier analysis on numerical variables
```

Out[17]:

	Feature name	Total count	Outlier Count	% Outliers	min_outlier	max_outlier
0	compactness	813	0	0.0	NaN	NaN
1	circularity	813	0	0.0	NaN	NaN
2	distance_circularity	813	0	0.0	NaN	NaN
3	radius_ratio	813	0	0.0	NaN	NaN
4	pr.axis_aspect_ratio	813	0	0.0	NaN	NaN
5	max.length_aspect_ratio	813	0	0.0	NaN	NaN
6	scatter_ratio	813	0	0.0	NaN	NaN
7	elongatedness	813	0	0.0	NaN	NaN
8	pr.axis_rectangularity	813	0	0.0	NaN	NaN
9	max.length_rectangularity	813	0	0.0	NaN	NaN
10	scaled_variance	813	0	0.0	NaN	NaN
11	scaled_variance.1	813	0	0.0	NaN	NaN
12	scaled_radius_of_gyration	813	0	0.0	NaN	NaN
13	scaled_radius_of_gyration.1	813	0	0.0	NaN	NaN
14	skewness_about	813	0	0.0	NaN	NaN
15	skewness_about.1	813	0	0.0	NaN	NaN
16	skewness_about.2	813	0	0.0	NaN	NaN
17	hollows_ratio	813	0	0.0	NaN	NaN

1.3.2 Data Scaling

As all the variables have values on different scales it will becomes important to scale the data. We will use StandardScaler to scale

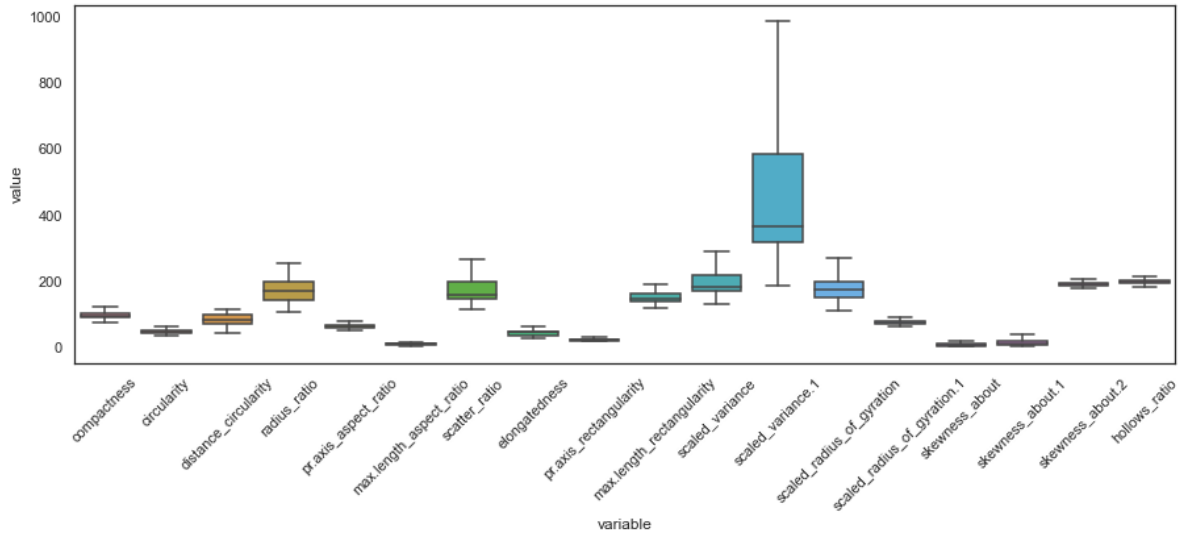
Box Plot - Before Scaling

In [18]:

```
fig = plt.figure(1, (15, 5))
ax = plt.subplot(1,1,1)
sns.boxplot(x="variable",y="value", data=pd.melt(data[data.columns[:-1]]))
plt.xticks(rotation=45)
```

Out[18]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17]),
 <a list of 18 Text major ticklabel objects>)
```



All the variables are on very different scales and ranges of values in each variable differs considerably. Hence it will be wise to scale the data and then plot the boxplot to visualise

Scaling Data using StandardScaler

In [19]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
df_scaled =data.copy()

col_names = data.columns[:-1]

col_to_scale = data[col_names]

col_to_scale = sc.fit_transform(col_to_scale.values)

df_scaled[col_names] = col_to_scale

df_scaled.head()
```

Out[19]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_as
0	0.163231	0.520408	0.060669	0.293086	1.910784	
1	-0.322874	-0.619123	0.124067	-0.852337	-0.750782	
2	1.256966	0.845988	1.518823	1.252765	0.846157	
3	-0.079822	-0.619123	-0.002729	-0.295104	0.313844	
4	-1.052030	-0.130753	-0.763506	1.128936	-0.041031	

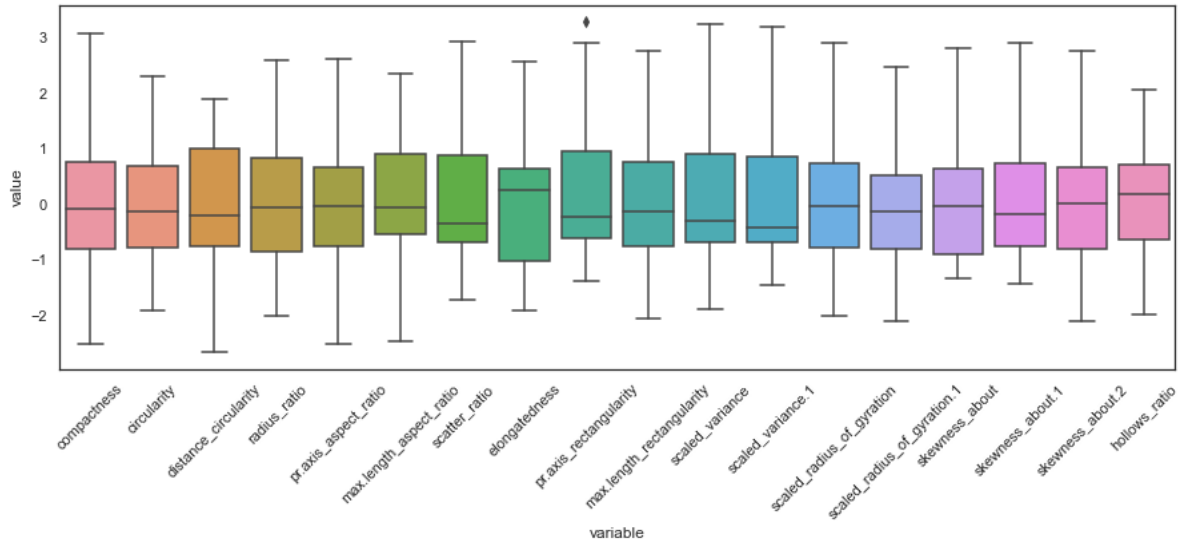
Box Plot - After Scaling

In [20]:

```
fig = plt.figure(1, (15, 5))
ax = plt.subplot(1,1,1)
sns.boxplot(x="variable",y="value", data=pd.melt(df_scaled[df_scaled.columns[:-1]]))
plt.xticks(rotation=45)
```

Out[20]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17]),
 <a list of 18 Text major ticklabel objects>)
```



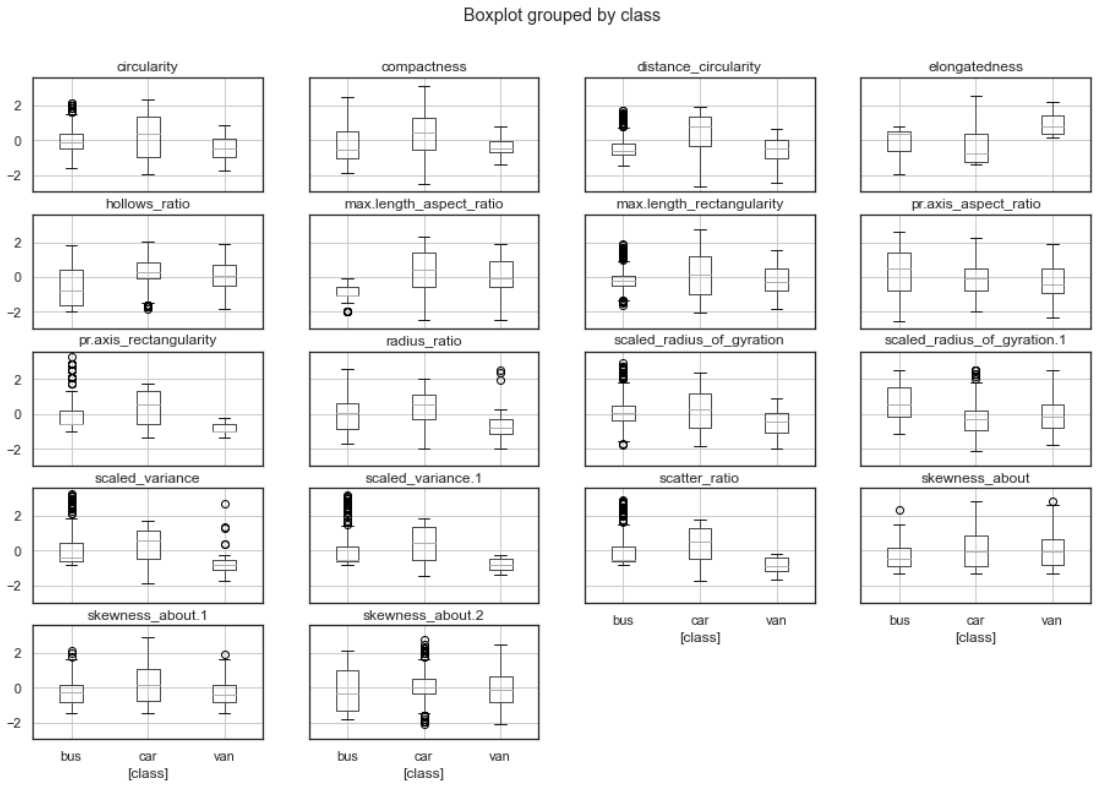
Box Plot - Hue over variable 'Class'

In [21]:

```
df_scaled.boxplot(by='class', layout = (5,4),figsize=(15,10))
```

Out[21]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001DA720E8390
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA71F43A20
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA71F77BA8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA71FABD30
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DA71FDFEB8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA72016B38
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA72046F98
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA72083400
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DA72083470
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA722A7CF8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA722E4198
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA723125F8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DA72343A58
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA72375EB8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA723B2358
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA723E47B8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DA72413C18
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA724520B8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA72482518
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DA724B1C88
>]],
      dtype=object)
```



There is no variable which can clearly distinguish between clusters and there is fair amount of overlap in data in variables

2. Understanding and Finalisation of Attributes

In this section we will try to understand relationship between different attributes. If there are attributes which are highly correltaed to each other, we shall make a decision to keep only one of them.

2.1 Pair Plot to check relationship between numeric variables

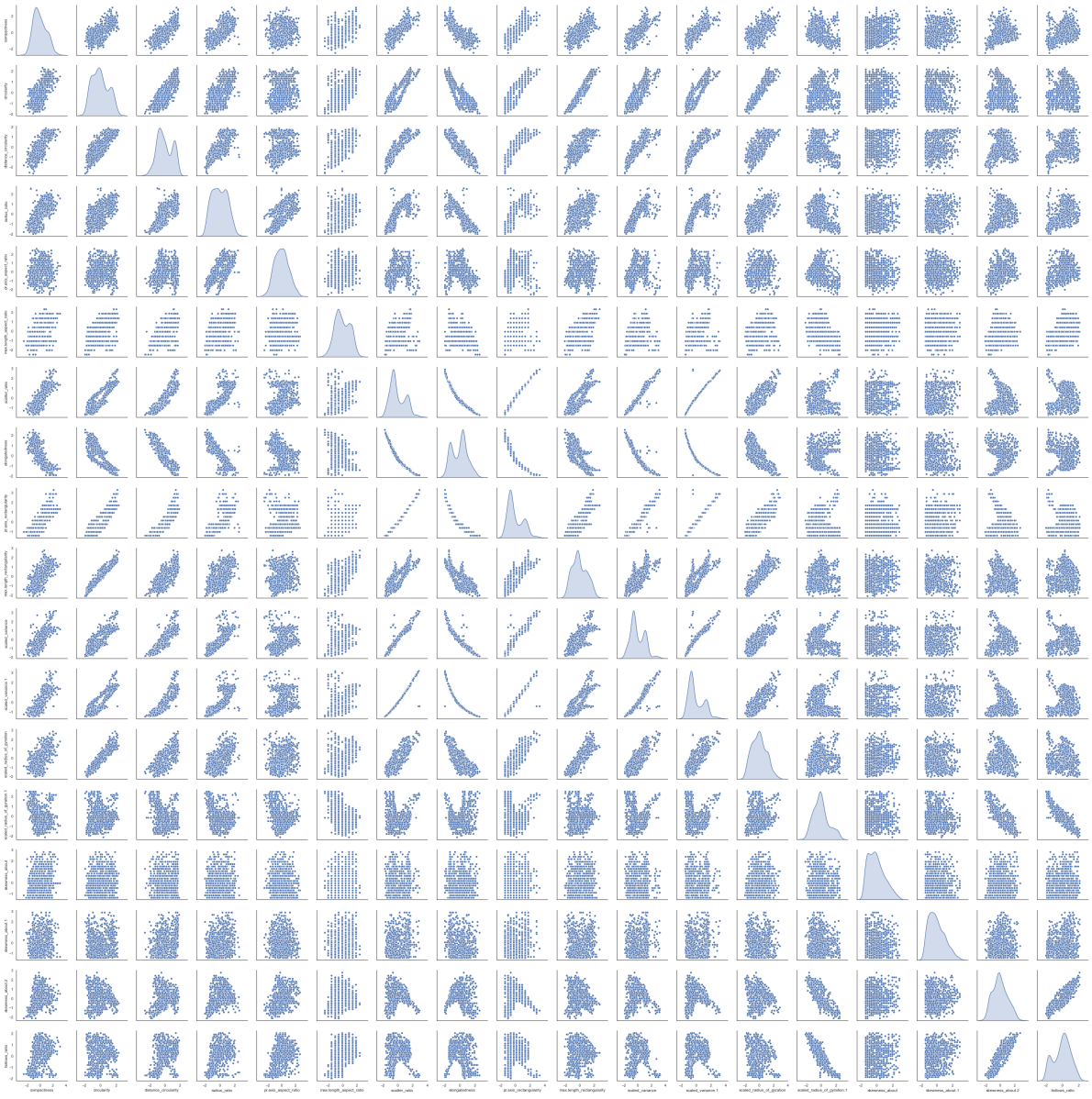
In [22]:

```
#bivariate analysis
fig = plt.figure(1,figsize=(21,20))
sns.pairplot(df_scaled,diag_kind='kde')
```

Out[22]:

```
<seaborn.axisgrid.PairGrid at 0x1da735cd860>
```

```
<Figure size 1512x1440 with 0 Axes>
```



We can see that some variables have direct positive or negative correlation among them. We will Analyse these combinations separately in later sections

2.2 Correlation Matrix

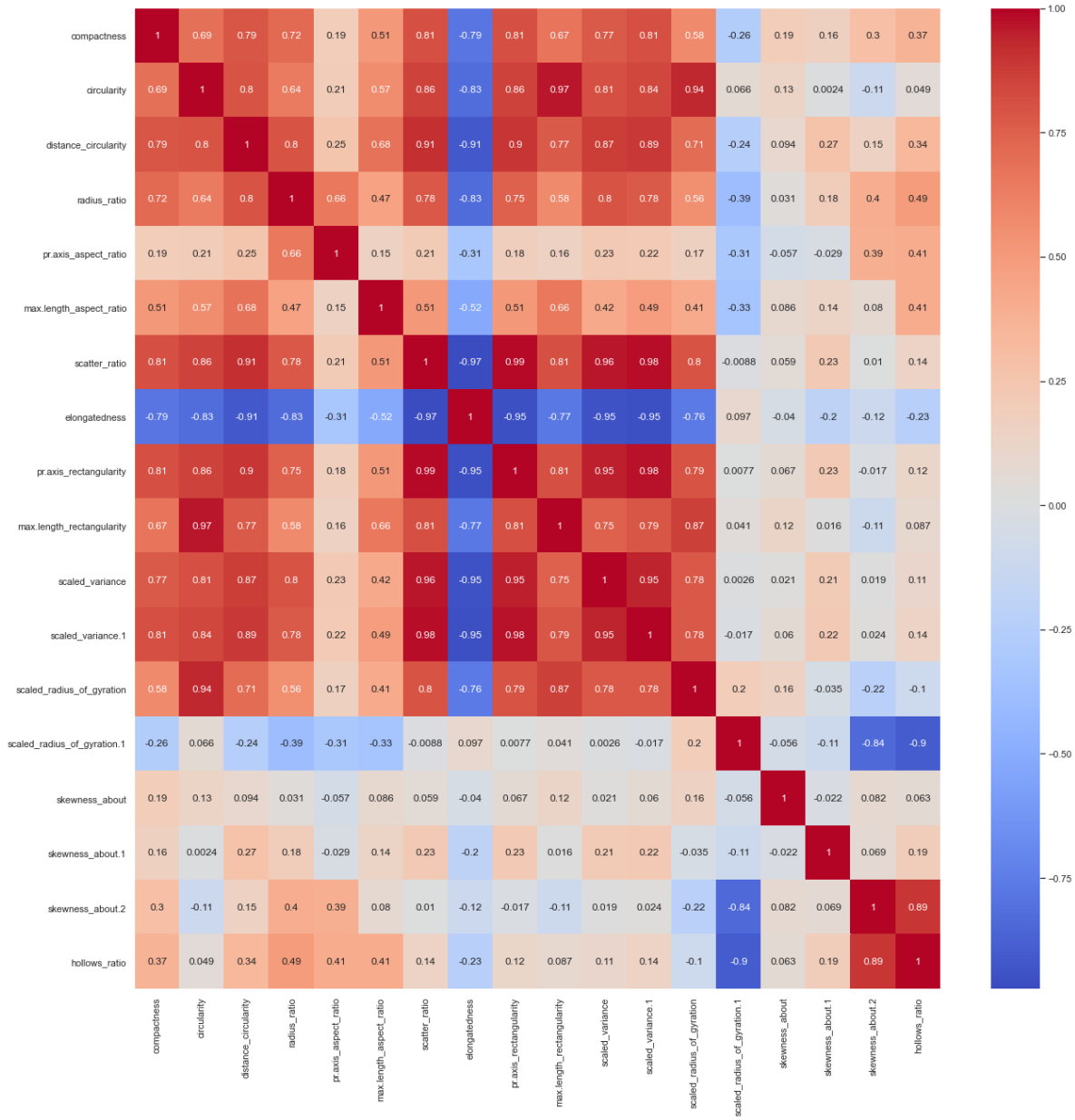
Correlation matrix will quantify degree of correlation between numeric variables

In [23]:

```
corrMatrix = df_scaled.corr()
fig = plt.figure(1,figsize=(21,21))
sns.heatmap(corrMatrix,annot=True,cmap='coolwarm')
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1da6eb96a90>
```



Comments:

1. From pairplot and correlation heatmap - there are multicollinear features in the dataset.
2. There are many features which are highly correlated with each other
3. We will create pairs of all the variables and compare the correlation coefficient of these combinations with a threshold
4. We will define threshold to be 0.7 i.e any combination of the features with correlation coefficient value greater than the threshold, one of those features shall be dropped

 # Variable pairs with correlation coefficient > 0.7

In [24]:

```
col = df_scaled.columns[:-1]
col = list(col)
combos = itertools.combinations(col,2)
lstCorrPair = []
# Lets assume a correlation value threshold beyond which we will assume that attributes ar
th = 0.8
for pair in combos:
    if (corrMatrix[pair[0]][pair[1]] > th) | (corrMatrix[pair[0]][pair[1]] < -th) :
        lstCorrPair.append(pair+(corrMatrix[pair[0]][pair[1]],))

print('There are total {} combinations which have correlation coefficient > 0.7. Following

df_corr_features = pd.DataFrame(lstCorrPair,columns=['Column1','Column2','Correlation Coe
df_corr_features
```

```
There are total 32 combinations which have correlation coefficient > 0.7. Fo
llowing is the list of the same
```

Out[24]:

	Column1	Column2	Correlation Coefficient
0	compactness	scatter_ratio	0.814026
1	compactness	pr.axis_rectangularity	0.814227
2	compactness	scaled_variance.1	0.812207
3	circularity	scatter_ratio	0.858149
4	circularity	elongatedness	-0.825108
5	circularity	pr.axis_rectangularity	0.856137
6	circularity	max.length_rectangularity	0.965366
7	circularity	scaled_variance	0.813063
8	circularity	scaled_variance.1	0.842380
9	circularity	scaled_radius_of_gyration	0.935594
10	distance_circularity	scatter_ratio	0.909023
11	distance_circularity	elongatedness	-0.912713
12	distance_circularity	pr.axis_rectangularity	0.897261
13	distance_circularity	scaled_variance	0.874259
14	distance_circularity	scaled_variance.1	0.889835
15	radius_ratio	elongatedness	-0.832791
16	scatter_ratio	elongatedness	-0.973413
17	scatter_ratio	pr.axis_rectangularity	0.991992
18	scatter_ratio	max.length_rectangularity	0.808154
19	scatter_ratio	scaled_variance	0.962947

	Column1	Column2	Correlation Coefficient
20	scatter_ratio	scaled_variance.1	0.983250
21	elongatedness	pr.axis_rectangularity	-0.950345
22	elongatedness	scaled_variance	-0.949732
23	elongatedness	scaled_variance.1	-0.951765
24	pr.axis_rectangularity	max.length_rectangularity	0.811979
25	pr.axis_rectangularity	scaled_variance	0.949475
26	pr.axis_rectangularity	scaled_variance.1	0.976775
27	max.length_rectangularity	scaled_radius_of_gyration	0.865240
28	scaled_variance	scaled_variance.1	0.945675
29	scaled_radius_of_gyration.1	skewness_about.2	-0.835244
30	scaled_radius_of_gyration.1	hollows_ratio	-0.900159
31	skewness_about.2	hollows_ratio	0.894057

2.2.1 Dropping features with high multicollinearity

In [25]:

```
col_to_drop = list(df_corr_features['Column2'].unique())
print("Dropped Features: {}".format(len(col_to_drop)))
col_to_drop
```

```
Dropped Features: 9
```

Out[25]:

```
['scatter_ratio',
 'pr.axis_rectangularity',
 'scaled_variance.1',
 'elongatedness',
 'max.length_rectangularity',
 'scaled_variance',
 'scaled_radius_of_gyration',
 'skewness_about.2',
 'hollows_ratio']
```

 # Relevant Features List

In [26]:

```
col = list(data.columns[:-1])
sel_features = []


for c in col_to_drop:
    if c in col:
        col.remove(c)
print("Relevant Features: {}".format(len(col)))
col
```

```
Relevant Features: 9
```

Out[26]:

```
['compactness',
 'circularity',
 'distance_circularity',
 'radius_ratio',
 'pr.axis_aspect_ratio',
 'max.length_aspect_ratio',
 'scaled_radius_of_gyration.1',
 'skewness_about',
 'skewness_about.1']
```

 # Dataset with Relevant Features

In [27]:

```
# prepare dataset with only relevant features
data_relevant_feature = df_scaled.drop(col_to_drop,axis=1)
data_relevant_feature
```

Out[27]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_
0	0.163231	0.520408	0.060669	0.293086	1.910784	
1	-0.322874	-0.619123	0.124067	-0.852337	-0.750782	
2	1.256966	0.845988	1.518823	1.252765	0.846157	
3	-0.079822	-0.619123	-0.002729	-0.295104	0.313844	
4	-1.052030	-0.130753	-0.763506	1.128936	-0.041031	
...	...	...	...	...	...	
841	-0.079822	-0.944703	0.314261	0.447873	0.491282	
842	-0.565926	0.194828	0.124067	-0.171275	0.846157	
843	1.500018	1.497149	1.201833	1.655211	1.023595	
844	-0.930504	-1.433074	-0.256321	-0.697550	-0.573344	
845	-1.052030	-1.433074	-1.017098	-1.409570	-1.105657	

813 rows × 10 columns

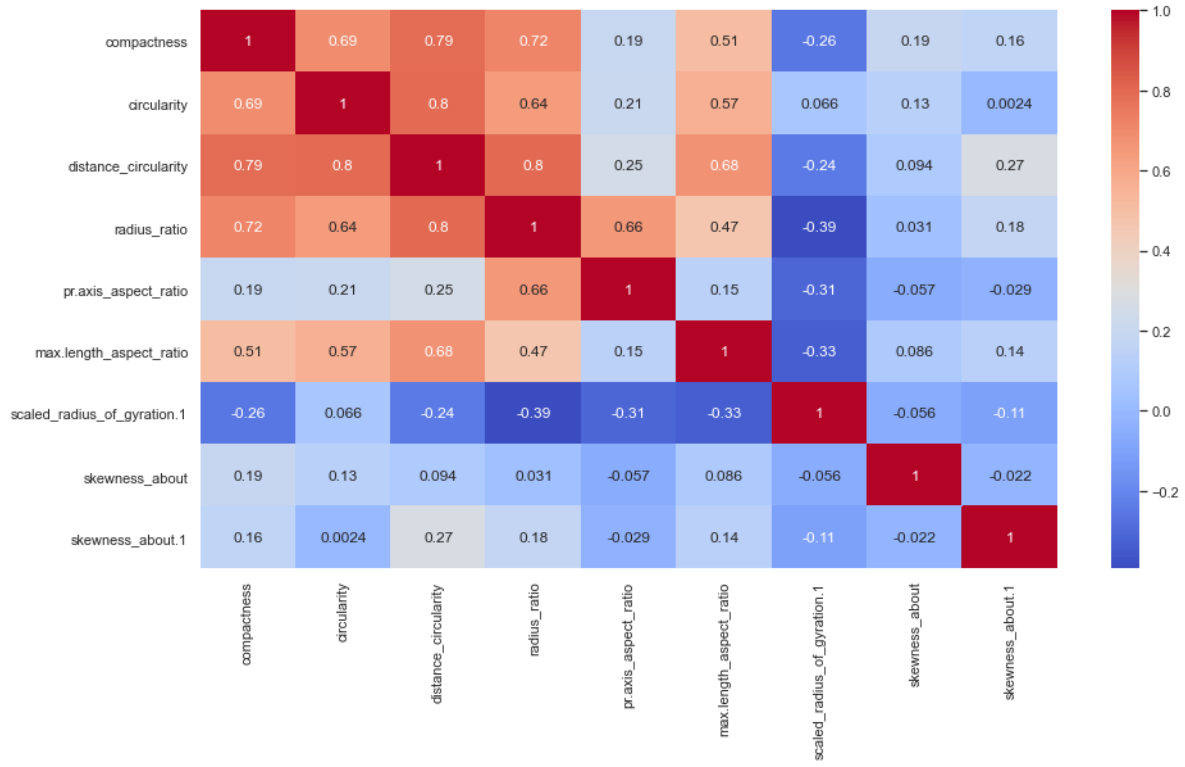
2.2.2 Rechecking correlation among relevant features

In [28]:

```
fig = plt.figure(1,figsize=(15,8))
sns.heatmap(data_relevant_feature.corr(),annot=True,cmap='coolwarm')
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1da7e630390>
```



Insights

1. There are no features which have correlation coefficient greater than 0.7 for any combination
2. Feature elingatedness is negatively correlated with compactness and circularity
3. Feature skewness_about.2 is negatively correlated with 'scaled_radius_of_gyration.1'

3. Splitting dataset in train and test dataset

In [29]:

```
from sklearn.model_selection import train_test_split
X = df_scaled.drop('class',axis=1)
y = df_scaled['class']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=1)
```

4. Building basic support vector machine classifier

4.1 SVM classifier with original scaled dataset

In [30]:

```
df = pd.DataFrame()
def build_comp_df(classifier,testaccuracy,trainaccuracy, precision, recall, f1score):
    global df
    compDict = {'Algorithm':classifier,
                'Clf Accuracy Test': testaccuracy,
                 'Clf Accuracy Train': trainaccuracy,
                'Clf Precision': precision,
                'Clf Recall': recall,
                'Clf f1 Score': f1score}
    df = df.append(compDict,ignore_index=True)
    return df
```

In [31]:

```
from sklearn.svm import SVC
base_svc = SVC()
base_svc.fit(X_train,y_train)
print("Training Accuracy : {}".format(base_svc.score(X_train,y_train)))
print("Testing Accuracy : {}".format(base_svc.score(X_test,y_test)))

testaccuracy = accuracy_score(y_test,base_svc.predict(X_test))
trainaccuracy = accuracy_score(y_train,base_svc.predict(X_train))
precision = precision_score(y_test,base_svc.predict(X_test),average='macro')
recall = recall_score(y_test,base_svc.predict(X_test),average='macro')
f1score = f1_score(y_test,base_svc.predict(X_test),average='macro')

df1 = build_comp_df('Basic SVC',testaccuracy,trainaccuracy,precision,recall,f1score)
```

```
Training Accuracy : 0.9737274220032841
Testing Accuracy : 0.9558823529411765
```

Insights:

1. Test and train accuracies are quite high and same. indicating that there is no overfitting.
2. Precision and recall scores are also very high

4.2 SVM classifier after feature selection

Selected Features:

'compactness', 'circularity', 'distance_circularity', 'radius_ratio', 'pr.axis_aspect_ratio', 'max.length_aspect_ratio', 'scaled_radius_of_gyration.1', 'skewness_about', 'skewness_about.1'

We will rebuild basic SVM using the seelcted features only to see the impact of 50 % reduction of number of variables i.e only 9 variables instead of original 18

In [32]:

```
X_rel = data_relevant_feature.drop('class',axis=1)
y_rel = data_relevant_feature['class']
X_train1,X_test1,y_train1,y_test1 = train_test_split(X_rel,y_rel,test_size=0.25,random_sta
base_svc1 = SVC()
base_svc1.fit(X_train1,y_train1)

print("Training Accuracy : {}".format(base_svc1.score(X_train1,y_train1)))
print("Testing Accuracy : {}".format(base_svc1.score(X_test1,y_test1)))

testaccuracy = accuracy_score(y_test,base_svc1.predict(X_test1))
trainaccuracy = accuracy_score(y_train,base_svc1.predict(X_train1))
precision = precision_score(y_test,base_svc1.predict(X_test1),average='macro')
recall = recall_score(y_test,base_svc1.predict(X_test1),average='macro')
f1score = f1_score(y_test,base_svc1.predict(X_test1),average='macro')

df1 = build_comp_df('SVC - Selected Features(9)',testaccuracy,trainaccuracy,precision,reca
```

```
Training Accuracy : 0.9359605911330049
Testing Accuracy : 0.8970588235294118
```

Insights

1. it is visible that after dropping following features there is aprrox 10% drop in accuracy on test set 'scatter_ratio', 'pr.axis_rectangularity', 'scaled_variance.1', 'elongatedness', 'max.length_rectangularity', 'scaled_variance', 'scaled_radius_of_gyration', 'skewness_about.2', 'hollows_ratio'
2. We have reduced the dimensions from 18 to 9 i.e 50% reduction and have managed to get a good accuracy score (just 6% less than the accuracy of model with 18 dimensions

5. SVM cross validation

This will be trained only on original scaled dataset and not on the selected features dataset

5.1 SVM with GridSearchCV

In [33]:

```
classifier = SVC()

#defining hyperparameter C and lambda
paramC = [0.1,1,10,100,1000]
paramGamma = [1,0.1,0.01,0.001,0.0001]
paramKernel = ['rbf']
hyperparameter =  {'C': paramC,
              'gamma': paramGamma,
              'kernel': paramKernel}
cvSVC = GridSearchCV(classifier, hyperparameter,n_jobs=-1)

cvSVC.fit(X_train,y_train)
```

Out[33]:

```
GridSearchCV(cv=None, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=T
rue,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100, 1000],
                         'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
                         'kernel': ['rbf']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [34]:

```
cvSVC.best_params_
```

Out[34]:

```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

After Grid search validation optimal value of hyperparameters

In [35]:

```
print("Training Accuracy : {}".format(accuracy_score(y_train,cvSVC.predict(X_train))))
print("Testing Accuracy : {}".format(accuracy_score(y_test,cvSVC.predict(X_test))))

testaccuracy = accuracy_score(y_test,cvSVC.predict(X_test))
trainaccuracy = accuracy_score(y_train,cvSVC.predict(X_train))
precision = precision_score(y_test,cvSVC.predict(X_test),average='macro')
recall = recall_score(y_test,cvSVC.predict(X_test),average='macro')
f1score = f1_score(y_test,cvSVC.predict(X_test),average='macro')

df1 = build_comp_df('SVC- Grid Search',testaccuracy,trainaccuracy,precision,recall,f1scor
```

```
Training Accuracy : 0.9967159277504105
Testing Accuracy : 0.9705882352941176
```

Insights:

1. There is a slight improvement in train and test accuracies
2. precision and recall scores have also improved marginally

5.2 SVM with K-fold Cross validation

In [36]:

```
from sklearn.model_selection import cross_val_score
kfSVC = SVC()
trainScores = cross_val_score(kfSVC, X_train,y_train, cv=10)
testScore = cross_val_score(kfSVC, X_test,y_test,cv=10)
print("Train Scores: ")
print(trainScores)
print("-"*60)
print("Test Scores: ")
print(testScore)
```

```
Train Scores:
[0.96721311 0.95081967 0.91803279 0.98360656 1.         0.98360656
 0.95081967 0.91803279 0.96721311 0.91666667]
------------------------------------------------------------
Test Scores:
[0.95238095 0.95238095 0.9047619  0.85714286 0.95       1.
 0.9        1.         0.85       0.85       ]
```

In [37]:

```
print("K-Fold cross validation mean score on Training dataset : {}".format(trainScores.mea
print("K-Fold cross validation mean score on Test dataset : {}".format(testScore.mean()))

testaccuracy = testScore.mean()
trainaccuracy = trainScores.mean()
precision = 'NA'
recall = 'NA'
f1score = 'NA'

df1 = build_comp_df('SVC- K-Fold',testaccuracy,trainaccuracy,precision,recall,f1score)
```

```
K-Fold cross validation mean score on Training dataset : 0.9556010928961747
K-Fold cross validation mean score on Test dataset : 0.9216666666666665
```

Insights:

There is a slight drop in train and test accuracies compared to other algorithms. This can be more reliable as K Fold over all reduces the chances of overfitting

6. Principal Component Analysis

6.1 Building PCA

In [38]:

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(X)
explained_variance = pca.explained_variance_
explained_variance
```

Out[38]:

```
array([9.82115454e+00, 3.30822663e+00, 1.17963517e+00, 1.15077799e+00,
       8.76739839e-01, 6.60591503e-01, 3.21967390e-01, 2.30083911e-01,
       1.30554075e-01, 8.06345168e-02, 7.00562768e-02, 6.23135927e-02,
       3.82321443e-02, 2.89721413e-02, 2.65237362e-02, 1.99450705e-02,
       1.27515951e-02, 3.00736189e-03])
```

6.1.1 Principal components

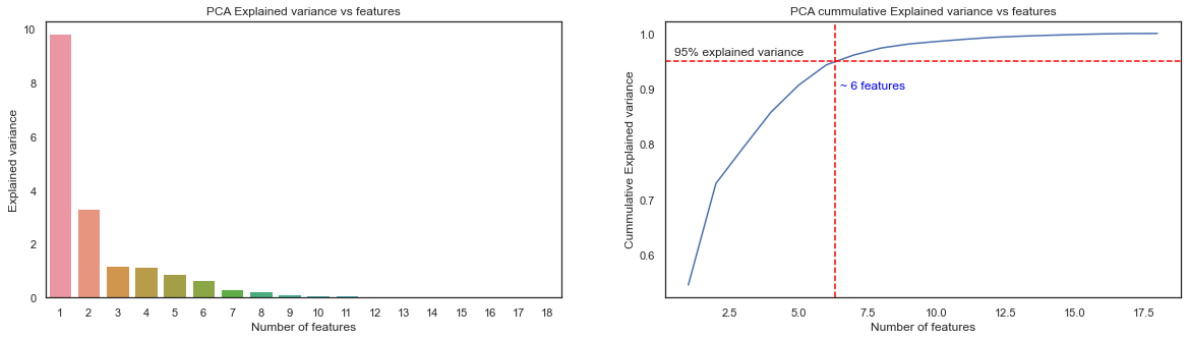
In below section we will plot explained variance vs feature and choose the number of features which explains 95% variance in the data

In [39]:

```
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(20,5))
x = np.arange(1,19)
y1 = explained_variance
sns.barplot(x,y1,ax=ax1)
ax1.set_xlabel('Number of features')
ax1.set_ylabel('Explained variance')
ax1.set_title('PCA Explained variance vs features')

sns.set(style='whitegrid')
sns.lineplot(x,np.cumsum(pca.explained_variance_ratio_),ax=ax2)
ax2.axhline(0.95, ls='--',c='red')
ax2.text(0.5,0.96,'95% explained variance')
ax2.axvline(6.3, ls='--',c='red')
ax2.text(6.5,0.9,'~ 6 features',c='blue')

ax2.set_xlabel('Number of features')
ax2.set_ylabel('Cummulative Explained variance')
ax2.set_title('PCA cummulative Explained variance vs features')
```

Out[39]:

```
Text(0.5, 1.0, 'PCA cummulative Explained variance vs features')
```



In [40]:

```
pca.explained_variance_
```

Out[40]:

```
array([9.82115454e+00, 3.30822663e+00, 1.17963517e+00, 1.15077799e+00,
       8.76739839e-01, 6.60591503e-01, 3.21967390e-01, 2.30083911e-01,
       1.30554075e-01, 8.06345168e-02, 7.00562768e-02, 6.23135927e-02,
       3.82321443e-02, 2.89721413e-02, 2.65237362e-02, 1.99450705e-02,
       1.27515951e-02, 3.00736189e-03])
```

In [41]:

```
df_comp = pd.DataFrame(pca.components_,columns=X.columns,index = list(np.arange(1,19)))
df_comp
```

Out[41]:

	compactness	circularity	distance_circularity	radius_ratio	pr.axis_aspect_ratio	max.length_a
1	0.271365	0.287377	0.301829	0.270661	0.101396	
2	-0.085112	0.142304	-0.043793	-0.191957	-0.249230	
3	-0.041988	-0.201979	0.058442	0.071763	-0.035272	
4	0.140206	-0.045569	0.108618	-0.249836	-0.612708	
5	0.106457	-0.110072	-0.081297	0.153919	0.197544	
6	0.289598	-0.068136	-0.032055	-0.118959	-0.561050	
7	0.216453	-0.377659	0.153908	0.157163	0.099892	
8	-0.741058	-0.069573	0.277252	0.119962	-0.195239	
9	0.384502	0.037239	0.081570	0.262076	-0.025132	
10	-0.154098	0.174007	-0.197755	0.044642	0.046907	
11	-0.095322	-0.013500	-0.561459	0.513134	-0.244865	
12	-0.013767	-0.080402	-0.605321	-0.124962	0.050422	
13	-0.062746	0.010870	0.161180	0.386951	-0.165283	
14	0.113352	0.005960	-0.167047	-0.090908	0.025129	
15	-0.035873	-0.209630	0.001337	-0.463172	0.233951	
16	-0.084821	-0.035755	0.038463	0.101474	-0.008052	
17	0.010211	-0.775211	0.028019	0.113782	-0.030207	
18	-0.015543	-0.073660	-0.003697	0.005314	0.007492	

Insights:

Important: 6 Fetures explains 95% of the variance in data. this is stark 66% reduction in number of variables

We will rebuild PCA with 6 principal components and then train SVM classifier again

6.2 SVM with Principal Components only

In [42]:

```
newPca = PCA(n_components=6)
X_new = newPca.fit_transform(X)

X_train2, X_test2, y_train2, y_test2 = train_test_split(X_new, y, test_size=0.25, random_s
```

In [43]:

```
pcaSvm = SVC()
pcaSvm.fit(X_train2,y_train2)
```

Out[43]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [44]:

```
print("Accuracy Training dataset post PCA : {}".format(accuracy_score(y_train2,pcaSvm.pre
print("Accuracy Test dataset post PCA : {}".format(accuracy_score(y_test2,pcaSvm.predict(

testaccuracy = accuracy_score(y_test2,pcaSvm.predict(X_test2))
trainaccuracy = accuracy_score(y_train2,pcaSvm.predict(X_train2))
precision = precision_score(y_test2,pcaSvm.predict(X_test2),average='macro')
recall = recall_score(y_test2,pcaSvm.predict(X_test2),average='macro')
f1score = f1_score(y_test2,pcaSvm.predict(X_test2),average='macro')

df1 = build_comp_df('SVC- PCA (6)',testaccuracy,trainaccuracy,precision,recall,f1score)
```

```
Accuracy Training dataset post PCA : 0.9326765188834154
Accuracy Test dataset post PCA : 0.8823529411764706
```

7. Model Comparison

In [45]:

```
df1
```

Out[45]:

	Algorithm	Clf Accuracy Test	Clf Accuracy Train	Clf Precision	Clf Recall	Clf f1 Score
0	Basic SVC	0.955882	0.973727	0.952072	0.955373	0.953525
1	SVC - Selected Features(9)	0.897059	0.935961	0.882024	0.902049	0.890991
2	SVC- Grid Search	0.970588	0.996716	0.969082	0.968745	0.968832
3	SVC- K-Fold	0.921667	0.955601	NA	NA	NA
4	SVC- PCA (6)	0.882353	0.932677	0.86159	0.898031	0.875296

Conclusions:

1. Best accuracy precision and recall scores are when SVC is built using Grid search CV and C = 100 and gamma =0.01
2. There is just 8% reduction in accuracy on test set when we train the model with 6 principal components compared with SVC with Grid search. On a larger dataset this would mean a huge reduction in time and space complexity in run time.
3. There is just 4% reduction in accuracy when using PCA and training model using 6 principal components compared with SVM build using K Fold CV.