Data Modelling and Analysis

Coursework: 2020/2021

Word Count: 2610

Name: Chi Wang

Student ID: 20299113

1. ANALYSIS AND PRE-PROCESSING

1. Explore the data

i. Provide a table for all the input features.

Feature	Type	Mean	Median	Mode	SD	IQR	Range	Missing
objid	Nominal	N/A	N/A	1.24e+18	N/A	N/A	N/A	0
dia	Ratio	2000.02763891039	349.1009346	27.54042634	22677.730429659	464.144166325	848144.24227366	6646
rerun	Nominal	N/A	N/A	301	N/A	N/A	N/A	30
ra	Ratio	175.544814643177	180.4418126	189.6527639	47.7723983180261	44.162726775	252.649281303	48
dec	Ratio	14.8267452163686	0.40286887	-0.393345053	25.208269356313	36.1039735355	73.924897909	49
u	Ratio	18.619020674865	18.85277	18.90212	0.828957666539591	1.08170999999999	6.61093	50
g	Ratio	17.3718525957404	17.49512	17.55623	0.945450735688265	1.19509	7.11942	51
r	Ratio	16.8407677727773	16.85862	15.99986	1.06771750344749	1.339165	12.37044	53
i	Ratio	16.5838594371126	16.55516	14.06613	1.14193103947858	1.404825	16.23242	50
z	Ratio	16.4230659535954	16.38993	13.9471	1.20347866067648	1.5222	11.22265	53
run	Ratio	980.929914017197	756	756	273.285055418003	579	1104	50
m_unt	Ratio	0.000233574184969019	0.000244096	1e-04	7.07436468736703e-05	0.000104917	0.0004041582	46
native	Nominal	N/A	N/A	1	N/A	N/A	N/A	50
flux	Ratio	183.517630957942	183.3246383	189.7145297	50.2938333923231	50.847381375	309.585733399	50
camcol	Ordinal	N/A	4	4	N/A	3	5	50
field	Nominal	N/A	N/A	301	N/A	N/A	N/A	50
specobjid	Nominal	N/A	N/A	2.88e+18	N/A	N/A	N/A	50
redshift	Ratio	0.143684261263117	0.04254421	0	0.388743044659762	0.0924784125	5.357990078	50
plate	Ratio	1461.86682663467	441	2558	1788.82020518795	2258	8144	50
mjd	Interval	52944.4652069586	51997	52000	1511.34139007686	2568	5903	50
fiberid	Nominal	N/A	N/A	155	N/A	N/A	N/A	32

Table1: Centrality, Dispersion and Missing Values

Centrality: The centrality of the input features represent the central tendency of the data distribution. Three metrics are used to illustrate it. (Mean/Median/Mode)

Dispersion: The dispersion show how is the spread of distribution. (Standard deviation / Range(max-min)/Interquartile Range)

Nominal: Category data, such as id, gender. As shown in Table1, *objid, rerun, native, field, specobjid* and *fiberid* are all Nominal data. The valid statistics is mode.

Ordinal: Data can be arranged by order. The valid statistics are mode, median and percentile.

Interval: Data can be arranged by order with meaningful difference, no true 0. As shown in Table1, mjd represent date, therefore it is interval.

Ratio: Rations in data are meaningful. All statistics are available.

ii. Analyse the class variable.

The class variable is Nominal data. Therefore, I chose bar plot to show the count of each object class. The appropriate statistics should be mode. As shown in Figure 1, the most frequently class is GALAXY.

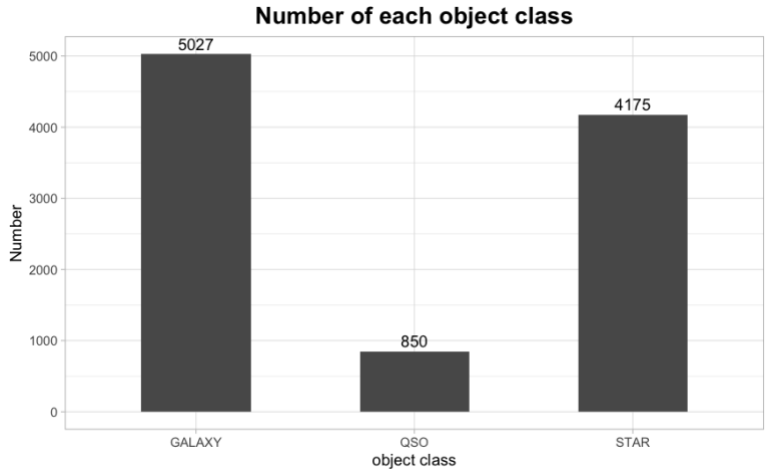


Figure 1: Bar plot of object class

iii. Produce histograms for each input attribute

Firstly, I used par function to set grid for graphs as 3 rows and 4 columns in a page. Then, I used sapply with hist function to draw all the graphs once. However, *objid* and *rerun* only have one value, that makes the their x axis of histogram kind of meaningless.

There are 6 types of shape of distribution - bell shaped, Bimodal, Skewed right, Skewed left, Uniform and Random.

Shapes	bell shaped	Bimodal	Skewed right	Skewed left	Uniform	Random
features	ra,g,r,i,z, flux, field	m_unt, native	dia, dec, specobjid, redshift, plate, mjd	u	objid, rerun	run, camcol, fiberid

Table2: Shape of histogram distribution

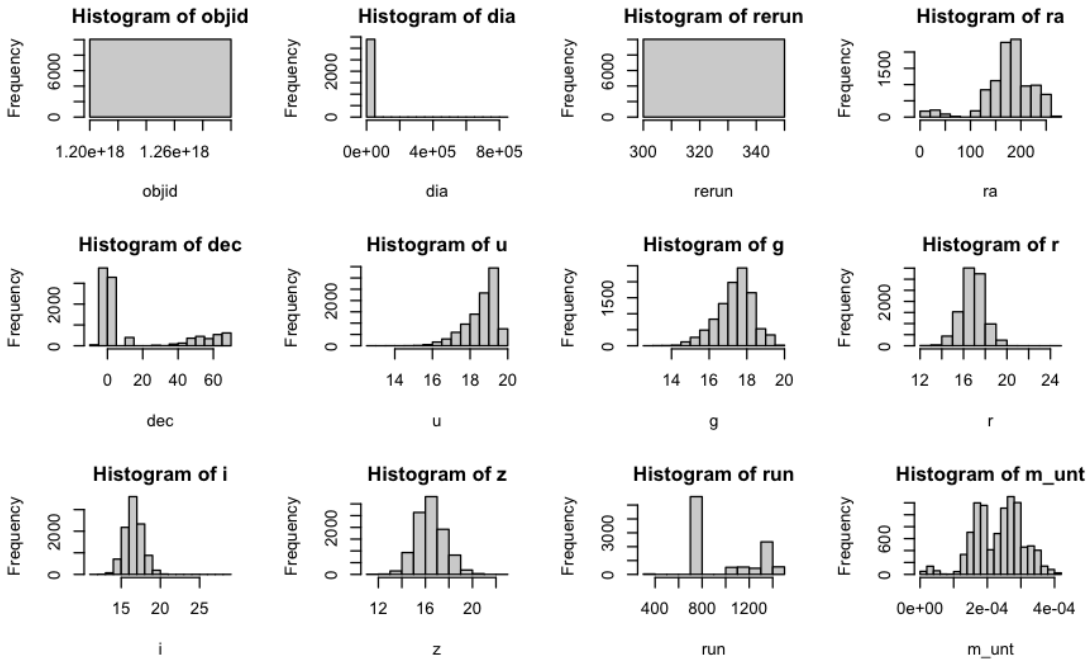


Figure 2: Bar plot of object class

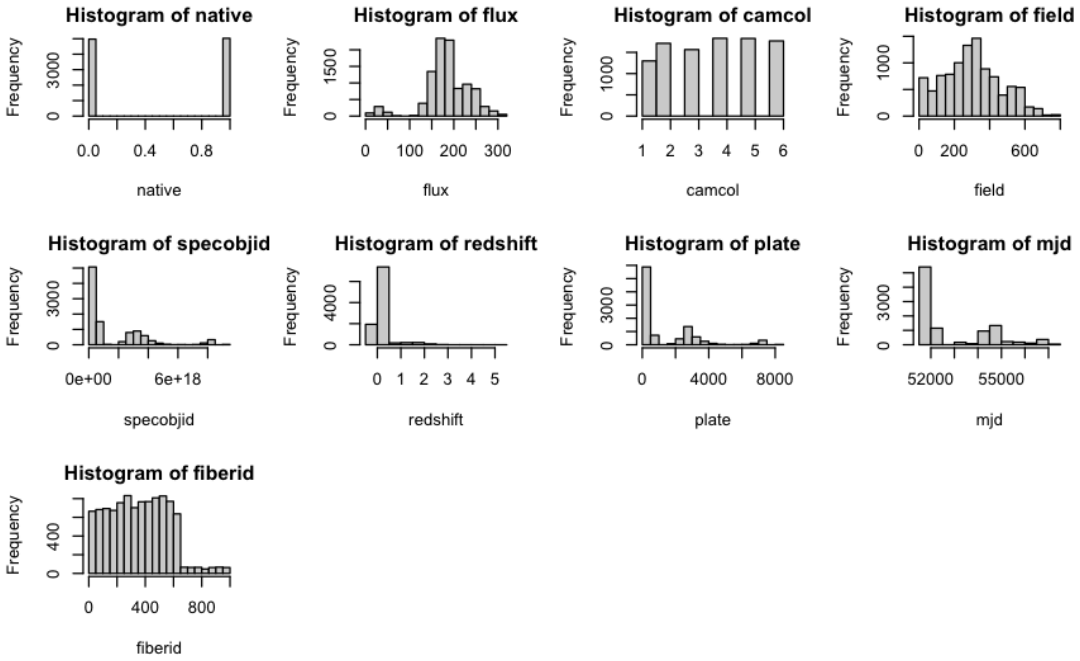


Figure 3: Bar plot of object class

2. Explore the relationships between the attributes

i. Calculate the correlation and produce a scatterplot for the variables: r and g.

As shown in Figure 4, we can see *r* and *g* are high positive correlation.

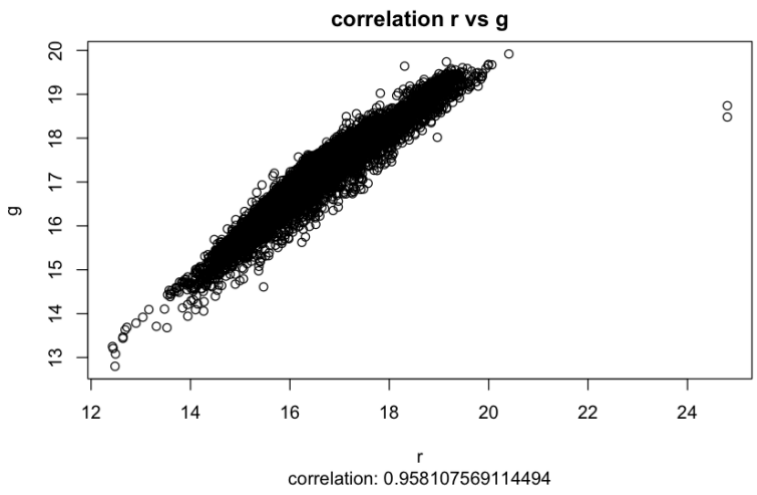


Figure 4: Scatterplot plot of *r* and *g*

ii. Calculate the correlation and produce a scatterplot for the variables: *mjd* and *r*.

As shown in Figure 5, we can see *mjd* and *r* are no correlation.

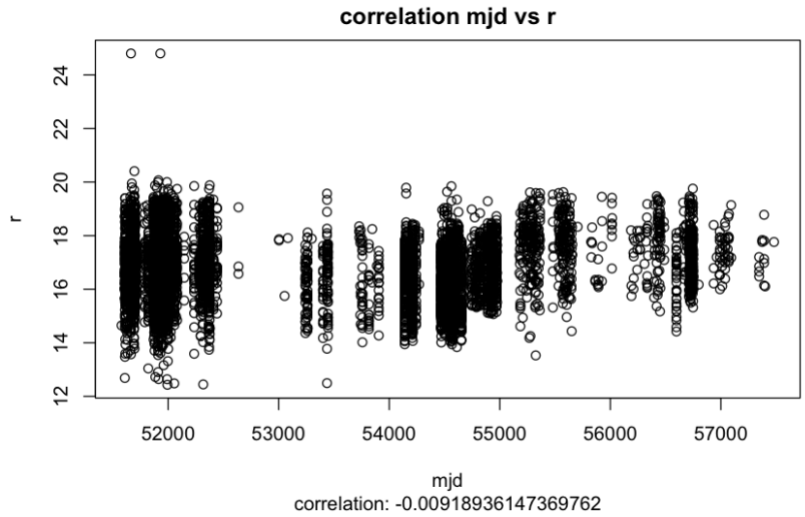


Figure 5: Scatterplot plot of *mjd* and *r*

iii. Produce scatterplots between the class variable and u, z, and redshift.

As shown in Figure 6, we can see that the range of *redshift* on object class are quite different(skewed). The range of u and z on object class are quite similar. Scatterplots seems not suitable for this scenario because the points all keep vertical corresponding to each class of class variable.

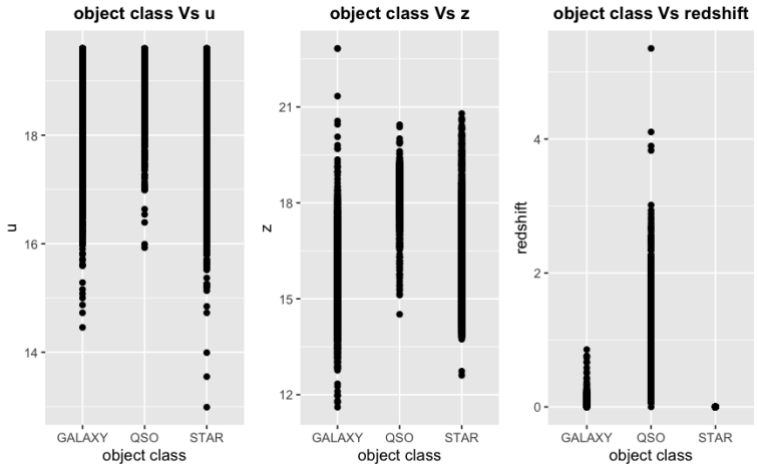


Figure 6: Scatterplot between the class variable and u, z, and redshift.

iv. Produce boxplots for all of the appropriate attributes.

I chose appropriate attributes for boxplots based on whether the standard deviation of that feature is meaningful. If the standard deviation is 0 or N/A, there is no need to draw a boxplot.

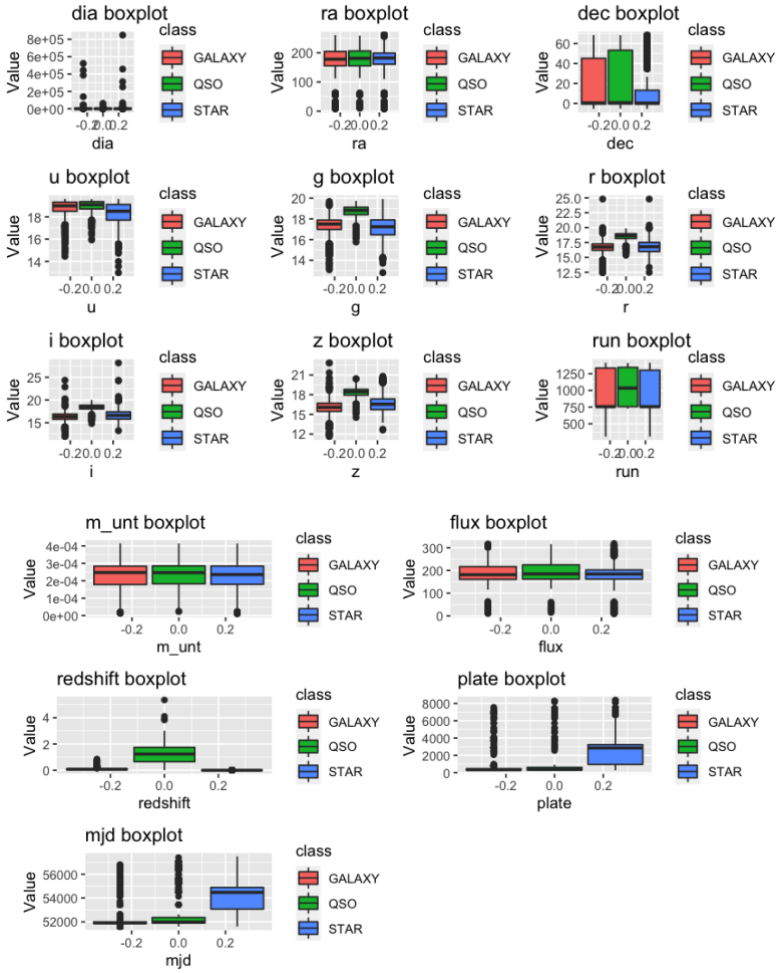


Figure 7: Boxplot for all of the appropriate attributes.

3. General Conclusions

Hold insignificant information:

(1). In figure 4, *r* and *g* have a highly positive correlation, that makes one of them can be treated as a redundant feature.

(2). In table 1, there are more than half of the *dia* are missing values. Too many missing values in an individual feature could lose the value of this feature. In addition, the missing of dia appear randomly on each class, see table 3. Therefore, *dia* holds insignificant information.

(3). *objid, rerun* are all same values, it could lead these two features to be insignificant.

(4). As shown in Figure 3, the histogram of *specobjid* and *palte* are quite similar, one of them can be treated as a redundant feature.

Class	dia(NA)
GALAXY	3388
QSO	576
STAR	2682

Table 3: Missing value of *dia* by class

Hold significant information:

(1). In Figure 6, *redshift* seems hold significant information because it represent differently on each class. It also can be seen in Figure 7.

(2). The features that represent differently on each class could potentially hold significant information. They are *plate*, *mjd*, *dia*, *g*, *r*, *i* and *z* , shown in Figure 7.

(3). Actually, it is hard to say which feature hold significant information from above processes. The above can be treated as a hypothesis. All the features except which hold insignificant information cold hold potential value.

4. Dealing with missing values in R.

Imputation	Definition	Influence
0	Filling with 0	Too rigid, could lose the veracity of the data
Group mean	Filling with the mean value of corresponding class	1. Mean value highly influence by outliers 2. Mean value is bad to reflect the centrality, especially in skewed data
Group median	Filling with the median value of corresponding class	1. Median value is not influenced by outliers so much. 2. Median value can reflect the centrality even in a skewed data

Table 4: Definition of 0/mean/median imputation

Let's use *dia* to illustrate the influence which has the most missing values. As shown in Table 5:

(1). Replacement with 0 and median definitely change the mean value of the origin data.

(2). Replacement with 0 and mean definitely change the median value of the origin data.

(3). Replacement with 0, mean and median have the similar influence on standard deviation of origin data.

(4). Replacement with mean and median have significant influence on IQR of the origin data.

Median imputation centralize the data.

(5). Replacement with mean and median make no change on the Range of the original data, because the outliers didn't change. Replacement with 0 make slightly change on the Range because 0 is smaller than the lowest outlier.

Data	Mean	Median	SD	IQR	Range	Missing
dia_origin	2000.0276	349.10093	22677.73	464.14417	848144.24	6646
dia0	677.68545	0	13233.285	234.42633	848171.78	0
dia_mean	1989.3876	1912.3464	13201.513	1578.9997	848144.24	0
dia_median	908.67572	348.64537	13222.484	1.3259056	848144.24	0

Table 5: Descriptive statistic of 0/mean/median imputation on *dia* feature

5. Attribute transformation

Transformation	Definition	Influence
mean centering(mc)	x - mean(x)	rescale mean value to 0
normalisation(nor)	(x - min(x)) / (max(x) - min(x))	rescale the range of the data to [0,1]
standardisation(sd)	(x - mean(x)) / sd(x)	rescale mean value to 0 and standard deviation to 1

Table 6: Mean centring, normalisation and standardisation transformation

Let's use *ra* to illustrate the influence in Table 7. We can see that after mean centring, the mean value of *ra* become 0. The range of *ra* is between 0 and 1 after normalisation. Normalisation make sure the scale of all the features are the same. Standardisation centred data around the mean value with unit standard deviation. After standardisation, the mean value the *ra* is 0 and standard deviation become 1.

Data	Mean	Median	SD	IQR	Range
ra_origin	175.54561	180.19688	47.658414	43.891653	252.64928
ra_mc	1.5E-15	4.6512745	47.658414	43.891653	252.64928
ra_nor	0.6622244	0.6806344	0.1886347	0.1737256	1
ra_sd	3.306E-17	0.0975961	1	0.9209634	5.3012524

Table 7: Mean centring, normalisation and standardisation transformation on *ra*

6. Attribute / instance selection

i . Attribute and instance deletion

Missing% - dia	Missing instance after drop "dia"	Duplicated instance after drop missing value
66.12%	57(0.567%)	2(0.02%)

Table 8: Missing and duplicated value

As shown in table 8:

(1). Drop the attribute with over 50% missing value - *dia*(66.12%). This is an uncommon situation. However, if an attribute has more than 50% missing value, that could be useless as it lack of so much information. Drop it should not have a huge influence on the data.

(2). As several missing value is in the same instance(end of the dataset), totally 57(0.567%). It's a quite small proportion of the data. It should also not influence the data so much.

(3). Only two instances are duplicated after drop all the missing value. Therefore, it should also not influence the data so much.

(4). Drop the feature which all the data are duplicated – *objid* and *rerun*. The feature with all duplicated data is useless.

ii. Use correlations to reduce the number of attributes.

(1). Drop the attribute with over 50% missing value - *dia*

(2). Drop all the missing instances.

(3). Pair compare the Pearson correlation between all the remaining features exclude ordinal attribute *camcol* and nominal attributes *objid*, *rerun*, *native*, *field*, *specobjid* and *fibered.*

(4). As shown in table 9, find the high correlation(> 95%), which are (*r*, *g*), (*r*, *i* ,*z*), (*ra*, *flux*), (*plate*, *mjd*). Therefore, pick only one in each high correlation set and drop the others. Finally, there are 17 remaining uncorrelated features.

Feature	ra	dec	u	g	r	i	z	run	m_unt	flux	redshift	plate	mjd
ra	1	0	0	0	0	0	0	0	0	1	0	0	0
dec	0	1	0	0	0	0	0	0	0	0	0	0	0
u	0	0	1	0	0	0	0	0	0	0	0	0	0
g	0	0	0	1	1	0	0	0	0	0	0	0	0
r	0	0	0	1	1	1	1	0	0	0	0	0	0
i	0	0	0	0	1	1	1	0	0	0	0	0	0
z	0	0	0	0	1	1	1	0	0	0	0	0	0
run	0	0	0	0	0	0	0	1	0	0	0	0	0
m_unt	0	0	0	0	0	0	0	0	1	0	0	0	0
flux	1	0	0	0	0	0	0	0	0	1	0	0	0
redshift	0	0	0	0	0	0	0	0	0	0	1	0	0
plate	0	0	0	0	0	0	0	0	0	0	0	1	1
mjd	0	0	0	0	0	0	0	0	0	0	0	1	1

Table 9: Pearson correlation results between the features. (1: > 95%)

7. Attribute transformation / reduction

Appropriate pre-processing:

(1). Drop the attribute with over 50% missing values – *dia*, and all same value – (*objid*, *rerun*), because they lack of valuable information.

(2). Drop all missing and duplicated instance. Because the proportion of missing and duplicated instance are quite small, shown in Table 8. If the proportion is not small, I will consider imputation methods.

i. Compare the effects of Principal Component Analysis

When PCA is using only for transformation, the purpose is to decorrelate the data. As shown in table 10, there are no correlation between each features after PCA. It should be a good method to decorrelate the data without dropping any attributes compare to Questions 6.ii.

As shown in table 11, when PCA is using dimensionality reduction, we can select the number of dimensions by looking cumulative proportion. The proportion of variance will decrease when the number of PC increase. That means we can drop several PCs from the end of all PCs without losing too much information. PCA is one of the way to help to alleviate the Curse of Dimensionality.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16	PC17	PC18
PC1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PC2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PC3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PC4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PC5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
PC6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
PC7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
PC8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
PC9	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
PC10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
PC11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
PC12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
PC13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
PC14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
PC15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
PC16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
PC17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
PC18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 10: Pearson correlation results between the PCA features. (1: > 0.1)

ii. How many PCs should be used to obtain a cumulative variance of at least 90%?

As shown in table 11, we need to 8 PCs. The coding part automatically return the required PCs, therefore table 11 is only for illustration.

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14	PC15	PC16	PC17	PC18
Standard deviation	2.1503002	1.8182871	1.6764296	1.4178683	1.1652859	0.9997203	0.9351635	0.7325536	0.6780264	0.657636	0.602729	0.3590325	0.2049844	0.1391997	0.1252582	0.1161659	0.0762017	0.0005279
Proportion of Variance	0.25688	0.18368	0.15613	0.11169	0.07544	0.05552	0.04859	0.02981	0.02554	0.02403	0.02018	0.00716	0.00233	0.00108	0.00087	0.00075	0.00032	0
Cumulative Proportion	0.25688	0.44055	0.59669	0.70837	0.78381	0.83934	0.88792	0.91773	0.94327	0.9673	0.98748	0.99465	0.99698	0.99806	0.99893	0.99968	1	1

Table 11: PCA Importance of components

2. CLUSTERING

1. Choose an appropriate dataset and use HCA, k-means, and PAM as clustering algorithms.

An appropriate dataset : The data with median imputation and normalisation.

Clustering algorithms are distance-based models, they use distance between data points to determine their similarity. Therefore, it is better to rescale the data to avoid the feature with larger magnitude to bias the result.

Internal Metrics	Definition(Compactness&&Separability)	Evaluation
average.between	average distance between clusters.	larger -> better
average.within	average distance within clusters.	smaller -> better
dunn	minimum separation / maximum diameter.	larger -> better

Table 12: Internal Metrics

External Metrics	Definition(TP: True positive, FN: False negative)	Evaluation
Accuracy	TP/N (Multi-class problem)	larger -> better
Precision	TP/(TP+FP)	larger -> better
Recall	TP/(TP+FN)	larger -> better
F1 Score	2 * (precision * recall) / (precision + recall)	larger -> better

Table 13: External Metrics

Internal Metrics: It's significant to ensure the chosen metrics can represent both compactness and separability in clustering scenario. Therefore, as shown in table12, I chose

"average.between", "average.within" and dunn index.

External Metrics: As shown in table 13, I simply chose the common external metrics to be consistent with the metrics of classification. Therefore, I could have a chance to make a comparison between clustering and classification. Additional, as shown in figure 1, the class variable is imbalance, F1 score can help to keep the fairness of the results.

Internal Metrics	hca	kmeans	pam
dunn	0.04136869	0.02700754	0.02799579
average.between	1.60410032	1.49974243	1.56606913
average.within	1.12564808	1.1798807	1.09807039

Table 17: Internal Metrics results of HCA, K-means, PAM

HCA							
	1	3	2	precision	recall	f1_score	accuracy
GALAXY	2518	49	2460	0.53791925	0.50089517	0.518747425	
QSO	361	90	399	0.1369863	0.10588235	0.119442601	
STAR	1802	518	1855	0.3935087	0.44431138	0.417369783	
Average				0.35613808	0.35036297	0.35185327	0.4439912
K-means							
GALAXY	1995	1356	1676	0.50480769	0.39685697	0.444370197	
QSO	297	299	254	0.11893397	0.35176471	0.177764566	
STAR	1660	859	1656	0.46179587	0.39664671	0.42674913	
Average				0.36184584	0.38175613	0.349627964	0.3929566
PAM							
GALAXY	2354	890	1783	0.5118504	0.46827133	0.489092042	
QSO	354	219	277	0.12132964	0.25764706	0.164971751	
STAR	1891	696	1588	0.43530702	0.38035928	0.40598236	
Average				0.35616235	0.36875923	0.353348718	0.4139475

Table 18: External Metrics results of HCA, K-means, PAM

As shown in table 17, hca has the largest "dunn index" and "average.between", that makes become our best model refer to internal metrics. It's interesting that pam get the lowest "average.within". It seems that pam are good at compacting cluster. As shown in table 18, pam has the best F1 score and hca has the best accuracy. Combining the conclusion from internal metrics, that makes hca become our best model in this scenario. However, without tuning parameter, I can't say which model is better generally.

2. Optimise each clustering method

HCA	Definition	Default
method	The distance measure to be used. This must be one of "euclidean", "maximum", "manhattan", "canberra", "binary" or "minkowski".	euclidean
cut_tree	The heights where the tree should be cut. Numeric scalar or vector with heights where the tree should be cut.	NULL

Table 14: HCA Parameter

K-means	Definition	Default
iter.max	The maximum number of iterations allowed.	10
nstart	The number of random sets.	1

Table 15: K-means Parameter

PAM	Definition	Default
metric	Distance: Character string specifying the metric to be used for calculating dissimilarities between observations.	c("euclidean", "manhattan")
nstart	This argument corresponds to the one of kmeans()	if(variant == "faster") 1 else NA

Table 16: PAM Parameter

Clustering algorithm's parameter: As we know the clustering algorithm is distance based algorithm. Therefore, the type of distance should be one of my choice.

Generally, I use grid search to find the best parameter for all the clustering algorithms. The criteria to decide the best parameter is to consider the majority vote from all the evaluation metrics.

- HCA: As shown in table 17, the cut_tree parameter does not influence our result as I increase the heights where the tree should be cut. One possible reason is that the deeper tree could be overfitting. The parameter with manhattan distance and cut_tree(10) show us the best result. It seems HCA with manhattan distance are suitable for this scenario.
- K-means: As shown in table 18, the parameter with 50 max iterations and 3 start random sets show us the best results. 50 max iterations is an optimal number between

10 and 100. As we have already know the data is in three groups, so it make sense that start with 3 random sets give us the best result.

- PAM: As shown in table 19, the parameter with manhattan distance and 1 start random set show us the best results. It can further prove the fitness of manhattan distance for this scenario. However, it is interesting the 1 start random set show us the best results. That show us a different conclusion compare with K-mean. One possible reason could be the different rationale between K-means and PAM. K-means use artificial centroids while PAM use the actual points in the dataset.

HCA							
(method, cut_tree)	**dunn**	**average.between**	**average.within**	**recall**	**precision**	**f1_score**	**accuracy**
(euclidean, 10)	0.04136869	1.60410032	1.125648077	0.350362965	0.35613808	0.35185327	0.44399125
(euclidean, 50)	0.04136869	1.60410032	1.125648077	0.350363	0.356138	0.351853	0.443991
(euclidean, 100)	0.04136869	1.60410032	1.125648077	0.350362965	0.35613808	0.35185327	0.44399125
(manhattan, 10)	0.04949764	1.511147983	1.251343148	0.523370237	0.56479545	0.52762256	0.69707521
(manhattan, 50)	0.04949764	1.511147983	1.251343148	0.523370237	0.56479545	0.52762256	0.69707521
(manhattan, 100)	0.04949764	1.511147983	1.251343148	0.523370237	0.56479545	0.52762256	0.69707521

Table 17: HCA results from parameter tuning

K-means							
(inter.max, nstart)	**dunn**	**average.between**	**average.within**	**recall**	**precision**	**f1_score**	**accuracy**
(10, 1)	0.143800077	1.572256941	1.098319141	0.360516	0.3547992	0.3549188	0.4302626
(10, 3)	0.028876561	1.555302265	1.083163721	0.3722851	0.3499672	0.339193	0.3798249
(10, 5)	0.028876561	1.555302265	1.083163721	0.3722851	0.3499672	0.339193	0.3798249
(50, 1)	0.049768471	1.59483254	1.112140564	0.3321052	0.3346685	0.3322763	0.4187226
(50, 3)	0.055336924	1.521809076	1.175663234	0.538	0.5821202	0.5393659	0.6446478
(50, 5)	0.143800077	1.572256941	1.098319141	0.360516	0.3547992	0.3549188	0.4302626
(100, 1)	0.049768471	1.59483254	1.112140564	0.3321052	0.3346685	0.3322763	0.4187226
(100, 3)	0.028876561	1.555302265	1.083163721	0.3722851	0.3499672	0.339193	0.3798249
(100, 5)	0.143800077	1.572256941	1.098319141	0.360516	0.3547992	0.3549188	0.4302626

Table 18: K-means results from parameter tuning

PAM							
(metric, nstart)	**dunn**	**average.between**	**average.within**	**recall**	**precision**	**f1_score**	**accuracy**
(euclidean, 1)	0.027995791	1.566069132	1.098070392	0.36875923	0.356162353	0.353348718	0.413947473
(euclidean, 3)	0.027995791	1.566069132	1.098070392	0.36875923	0.356162353	0.353348718	0.413947473
(euclidean, 5)	0.027995791	1.566069132	1.098070392	0.36875923	0.356162353	0.353348718	0.413947473
(manhattan, 1)	0.017570378	1.522288905	1.138259604	0.51420029	0.585321967	0.485388143	0.521985674
(manhattan, 3)	0.026252297	1.561251058	1.091769937	0.37750625	0.357409361	0.352790428	0.402506964
(manhattan, 5)	0.026252297	1.561251058	1.091769937	0.37750625	0.357409361	0.352790428	0.402506964

Table 19: PAM results from parameter tuning

3. Clustering algorithm on different dataset

As shown in table 20:

(1). ALL PCs Vs 12PCs: They have the similar performance on each metric. That further

proved that we can drop the PCs refer to cumulative proportion without losing much information.

(2). Mean centering(0/mean/median imputation): The one with median imputation show us the best result. It further proved the advantage of median imputation as shown in table 4. The one with 0 imputation do show us the worst performance, it also fits the conclusion in table 4.

(3). Attribute deletion Vs Others: The result on "attribute deletion" dataset show us a competitive results with the best. It seems those features(*dia*, *objid* and *rerun*) which we dropped contains potential value. However, I still believe that the attribute with all same value(*objid* and *rerun*) should not influence the result. Therefore, the potential valuable feature should be *dia*. It is a surprise that the attribute with more than 50% randomly missing value still contain valuable information. That future proved the alert: "Be really careful to drop attribute of data".

(4). PCs Vs All: The result from PCs show us the worst result. One possible reason could be that the linear transformation of PCA damage the distance information of the data.

K-means							
Dataset	**dunn**	**average.between**	**average.within**	**recall**	**precision**	**f1_score**	**accuracy**
ALL PCs	0.01187978	6.234987556	4.930647767	0.3924417	0.53075012	0.43725638	0.54342606
12 PCs	0.0135915	6.219302894	4.913700968	0.39193391	0.53065701	0.43687779	0.54272564
Deletion	0.07859079	3.68694E+18	3.02682E+17	0.57003853	0.60556628	0.57145649	0.76413489
Mean centering(0)	0.07859079	3.68833E+18	3.04588E+17	0.56897719	0.60506774	0.57035511	0.76303223
Mean centering(mean)	0.07859079	3.6797E+18	3.02364E+17	0.57057399	0.60594834	0.57199578	0.76502189
Mean centering(median)	0.07859079	3.68137E+18	3.01737E+17	0.57057399	0.60594834	0.57199578	0.76502189

Table 20: K-means results on datasets with different pre-processing method

3. CLASSIFICATION

1. Choose an appropriate dataset to obtain predictions using the different classifiers.

An appropriate dataset : The data with median imputation and mean centering. Actually, tree-based model or rule-based model are not biased by unscaled, I could choose the dataset without rescaling. However, to keep fair to all the models including distance based model(IBK), I chose the dataset with normalisation. As shown in table 4, median imputation seems to be the best on in this scenario. Additionally, the results from table 20 partly proved the advantage of median imputation which show us the best performance.

Evaluation Protocol		Description	Evaluation
Data splits	Training/testing set up	Training: A subset of data for training/tuning the algorithm. Testing : A previously unseen subset of data for performance evaluation.	Usually: (70% -30%), (80%-20%)
Evaluation metrics	Accuracy	Rate of correctly classified instances from true positives and true negatives.	Higer -> better
	Recall	Rate of positive results.	Higer -> better
	Precision	The number of correct positive results divided by the number of all positive results.	Higer -> better
	F1 score	A balance between Precision and Recall. Represent well on imbalance dataset.	Higer -> better
	Cohen's Kappa Coefficient	Compares an observed accuracy with an expected accuracy (random chance).	Higer -> better

Table 21: Evaluation Protocol

ZeroR								
Class	**Confusion Matrix**			**Precision**	**Recall**	**F-Measure**	**Kappa statistic**	**Accuracy**
	GALAXY	**QSO**	**STAR**					
GALAXY	1515	0	0	0.502	1	0.669		
QSO	255	0	0	?	0	?		
STAR	1246	0	0	?	0	?		
Average				?	0.502	?	0	50.23%

Table 22: ZeroR Classification Performance

OneR								
Class	**Confusion Matrix**			**Precision**	**Recall**	**F-Measure**	**Kappa statistic**	**Accuracy**
	GALAXY	**QSO**	**STAR**					
GALAXY	1502	12	1	0.989	0.991	0.99		
QSO	16	238	1	0.952	0.933	0.943		
STAR	0	0	1246	0.998	1	0.999		
Average				0.99	0.99	0.99	0.9825	99.01%

Table 23: OneR Classification Performance

NaïveBayes								
Class	**Confusion Matrix**			**Precision**	**Recall**	**F-Measure**	**Kappa statistic**	**Accuracy**
	GALAXY	**QSO**	**STAR**					
GALAXY	1465	42	8	0.9	0.967	0.933		
QSO	16	238	1	0.751	0.933	0.832		
STAR	146	37	1063	0.992	0.853	0.917		
Average				0.925	0.917	0.918	0.8554	91.71%

Table 24: NaïveBayes Classification Performance

IBK 5-NN								
Class	**Confusion Matrix**			**Precision**	**Recall**	**F-Measure**	**Kappa statistic**	**Accuracy**
	GALAXY	**QSO**	**STAR**					
GALAXY	1449	4	62	0.818	0.956	0.882		
QSO	66	165	24	0.917	0.647	0.759		
STAR	257	11	978	0.919	0.785	0.847		
Average				0.868	0.859	0.857	0.7463	85.94%

Table 25: IBK 5-NN Classification Performance

J48								
Class	Confusion Matrix			Precision	Recall	F-Measure	Kappa statistic	Accuracy
	GALAXY	QSO	STAR					
GALAXY	1508	3	4	0.997	0.995	0.996		
QSO	4	250	1	0.988	0.98	0.984		
STAR	0	0	1246	0.996	1	0.998		
Average				0.996	0.996	0.996	0.993	99.60%

Table 26: J48 Classification Performance

ZeroR Vs OneR Vs J48 Vs IBK 5-NN Vs NaïveBayes					
Algorithm	Precision	Recall	F-Measure	Kappa	Accuracy
ZeroR	?	0.502	?	0	50.23%
OneR	0.99	0.99	0.99	0.9825	99.01%
J48	0.996	0.996	0.996	0.993	99.60%
IBK 5-NN	0.868	0.859	0.857	0.7463	85.94%
NaïveBayes	0.925	0.917	0.918	0.8554	91.71%

Table 27: ZeroR, OneR, J48, IBK 5-NN, NaïveBayes Classification Performance

- ZeroR: As the baseline of the model performance, ZeroR use zero rules to predict the majority vote-“GALAXY”. As shown in table 22, it even worse than a random change prediction(Kappa statistic). However, the accuracy is 50.23%, because this is an imbalance class(See Figure 1).
- OneR: As a rule-based classification algorithm, OneR use a single attribute rule to minimize error. As shown in table 23, the performance is much better than the baseline, even close to the best.
- J48: As a rule-based classification algorithm, J48 use multiple attribute rules(decision tree). As shown in table 27, it is better than OneR. It shows us the best performance. That’s quite reasonable because of the multiple attribute rules.
- IBK 5-NN: As a distance-based classification algorithm, it gives us the worst performance of all chosen models except the baseline(see table 27). It seems that the distance information are not valuable than the information of rules in this classification scenario.
- NaïveBayes: As a probability-based classification algorithm, it assumes the attributes are independent of each other. However, it is difficult to ensure the attributes didn’t

influence with each other in the real world. Therefore, the performance are even worse than OneR(see table 27).

In summary, as shown in table 27, the J48 algorithm shows us the best results.

Rule-based algorithm > Probability-based algorithm > Distance-based algorithm.

2. IBk (k-NN) Optimization

Parameter	Description	Example
Train/Test split	The ratio of train/test of the data	70% - 30%
K neighbors	How many nearest points can vote for prediction.	5
Distance metric	Euclidean distances are root sum-of-squares of differences. manhattan distances are the sum of absolute differences.	Euclidean manhattan

Table 28: Description of IBK parameters

IBk - Optimization					
(test%, K-neighbors, distance)	Precision	Recall	F-Measure	Kappa statistic	Accuracy
(20%, 3, Euclidean)	0.874	0.869	0.867	0.7635	86.87%
(20%, 3, Manhattan)	0.873	0.869	0.866	0.7628	86.87%
(20%, 5, Euclidean)	0.867	0.86	0.858	0.7475	86.02%
(20%, 5, Manhattan)	0.875	0.868	0.866	0.7618	86.82%
(20%, 10, Euclidean)	0.865	0.852	0.848	0.73	85.17%
(20%, 10, Manhattan)	0.871	0.859	0.856	0.7443	85.92%
(30%, 3, Euclidean)	0.871	0.865	0.863	0.7579	86.54%
(30%, 3, Manhattan)	0.874	0.87	0.867	0.7655	86.97%
(30%, 5, Euclidean)	0.868	0.859	0.857	0.7463	85.94%
(30%, 5, Manhattan)	0.872	0.866	0.864	0.7596	86.64%
(30%, 10, Euclidean)	0.867	0.853	0.849	0.7322	85.28%
(30%, 10, Manhattan)	0.871	0.858	0.854	0.7427	85.81%

Table 29: IBK Optimization by tuning parameters

As shown in table 29, with the test data proportion(30%), 3 neighbors and Manhattan distance method, the IBK algorithm show us the best performance. The number of neighbors is the parameter which has a significant influence on the model performance.

3. Apply J48 to the datasets listed below using 10-fold cross-validation.

J48 - Data from different preprocessing					
Dataset	**Precision**	**Recall**	**F-Measure**	**Kappa**	**Accuracy**
ALL PCs	0.947	0.947	0.947	0.9074	94.73%
12 PCs	0.896	0.895	0.895	0.8159	89.54%
Deletion	0.987	0.987	0.987	0.9773	98.71%
normalised(0)	0.986	0.986	0.986	0.9755	98.61%
normalised(mean)	0.995	0.995	0.995	0.9918	99.53%
normalised(median)	0.991	0.991	0.991	0.9845	99.11%

Table 30: The performance of J48 on the data from different preprocessing

As show in table 30:
(1). All PCs Vs 12PCs: The performance on 12PCs are quite lower than all PCs. This is different from clustering scenario. One possible reason could be that rule-based algorithm need much more information to establish better rules. Therefore, dimension reduction of PCA is not suitable for this situation.
(2). Normalised data: The data with 0 imputation still perform bad of all imputation method. This finding keeps consistent along all this coursework. While, the mean imputation seems better than median this time. However, as shown in table 27, my chosen data is median imputation with mean centering. It shows the best performance of all J48 results. Therefore, it is difficult to state which is the best imputation method between mean and median. In practice, they are competitive. However, theoretically, I prefer median imputation for this dataset(see 4).
(3). Normalised data Vs all: The normalized dataset are those dataset which didn't drop any values. They are better than the deletion dataset(drop *dia*, *objid* and *rerun*). The deletion dataset are better than the one after PCA. The one after PCA dimension reduction perform worst. This further proved the value of each data in this dataset. The same conclusion with clustering. I assume the *dia* feature has valuable information although it has more than 50% missing value at the end of clustering question. In tree-based, it's easier to confirmed that hypothesis. The follow is the decision tree of J48 on the dataset which shows us the best performance(Figure 8).

As shown in Figure 8, another amazing finding is that feature *redshift* contribute a lot for decision tree. Which exactly confirmed the hypothesis in Part1 general conclusions.

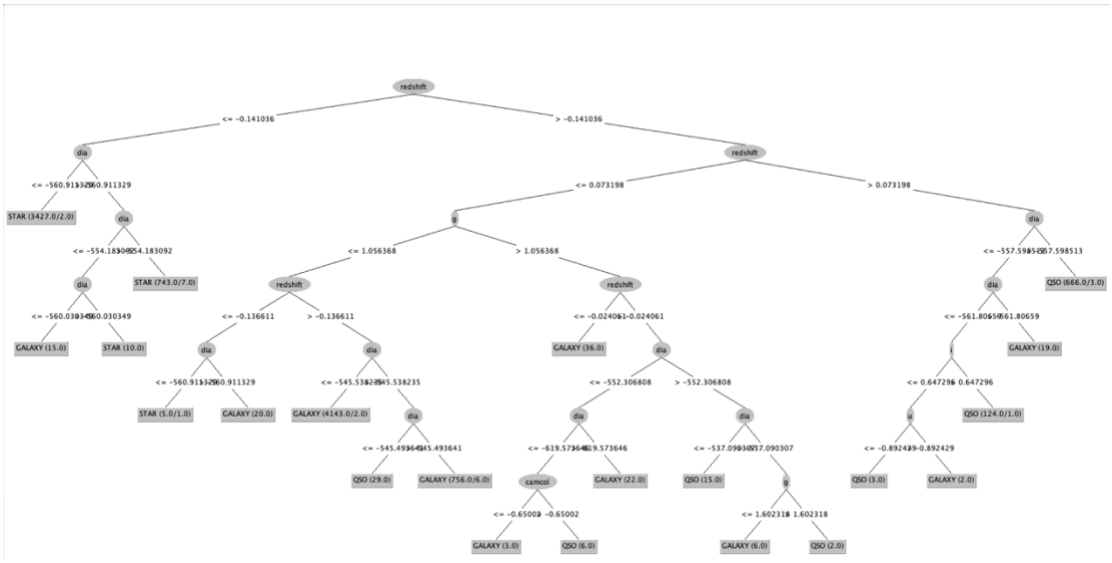


Figure 8: Decision tree of J48 on the dataset with median imputation and mean centering

Appendix:

R Code with results are in the following ~

Data Modelling and Analysis

Coursework: 2020/2021

20299113

1 1. ANALYSIS AND PRE-PROCESSING

1.1 1. Explore the data

1.1.1 1. Provide a table for all the input features of the dataset including measures of centrality,

dispersion, and how many missing values each attribute has.

```
# Read data
data = read.csv('cw_data.csv')
```

```
# Load required packages
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("ggplot2")
library("gridExtra")
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library("knitr")
library("cluster.datasets")
library("cluster")
library("e1071")
library("fpc")
library("xlsx")
```

```
# Get feature column
features = select(data, -class)
# Get Mean, Median, Mode, standard deviation, interquartile range ,range, Missing value number
statistics_res = sapply(features, function(x) c(mean(x, na.rm=TRUE),
                                                median(x, na.rm=TRUE),
                                                names(which.max(table(x))),
                                                sd(x, na.rm=TRUE),
                                                IQR(x, na.rm=TRUE),
                                                max(x, na.rm=TRUE)-min(x, na.rm=TRUE),
                                                sum(is.na(x))))
# Transform to r dataframe to store
statistics_res_df = as.data.frame(statistics_res,
                        row.names = c("Mean", "Median", "Mode", "SD", "IQR", "Range", "Missing"))
statistics_res_df = as.data.frame(t(statistics_res_df))
# Save to Excel for report
#write.xlsx(statistics_res_df, "statistics_res.xls", sheetName = "statistics_res", append = TRUE)
statistics_res_df
```

```
##                          Mean      Median         Mode                   SD
## objid                1.24e+18    1.24e+18     1.24e+18                    0
## dia           2000.02763891039 349.1009346  27.54042634      22677.730429659
## rerun                     301         301          301                    0
## ra            175.544814643177 180.4418126  189.6527639     47.7723983180261
## dec           14.8267452163686  0.40286887 -0.393345053      25.208269356313
## u              18.619020674865    18.85277     18.90212    0.828957666539591
## g             17.3718525957404    17.49512     17.55623    0.945450735688265
## r             16.8407677727773    16.85862     15.99986     1.06771750344749
## i             16.5838594371126    16.55516     14.06613     1.14193103947858
## z             16.4230659535954    16.38993      13.9471     1.20347866067648
## run           980.929914017197         756          756     273.285055418003
## m_unt     0.000233574184969019 0.000244096        1e-04 7.07436468736703e-05
## native       0.502699460107978           1            1    0.500017709373118
## flux          183.517630957942 183.3246383  189.7145297     50.2938333923231
## camcol        3.64827034593081           4            4      1.6660684047498
## field         302.393521295741         299          301     162.553590987784
## specobjid  1645881423715257088    4.97e+17     2.88e+18  2013805366298222848
## redshift     0.143684261263117  0.04254421            0    0.388743044659762
## plate         1461.86682663467         441         2558     1788.82020518795
## mjd           52944.4652069586       51997        52000     1511.34139007686
## fiberid       352.755289421158         351          155     206.516700091964
##                         IQR           Range Missing
## objid                     0               0       0
## dia           464.144166325 848144.24227366    6646
## rerun                     0               0      30
## ra             44.162726775   252.649281303      48
## dec           36.1039735355    73.924897909      49
```

```
## u         1.08170999999999         6.61093      50
## g                  1.19509         7.11942      51
## r                 1.339165        12.37044      53
## i                 1.404825        16.23242      50
## z                   1.5222        11.22265      53
## run                   579            1104      50
## m_unt           0.000104917   0.0004041582      46
## native                    1               1      50
## flux           50.847381375   309.585733399      50
## camcol                    3               5      50
## field                   229             757      50
## specobjid         2.541e+18       9.17e+18      50
## redshift       0.0924784125    5.357990078      50
## plate                  2258            8144      50
## mjd                    2568            5903      50
## fiberid                 324             999      32
```

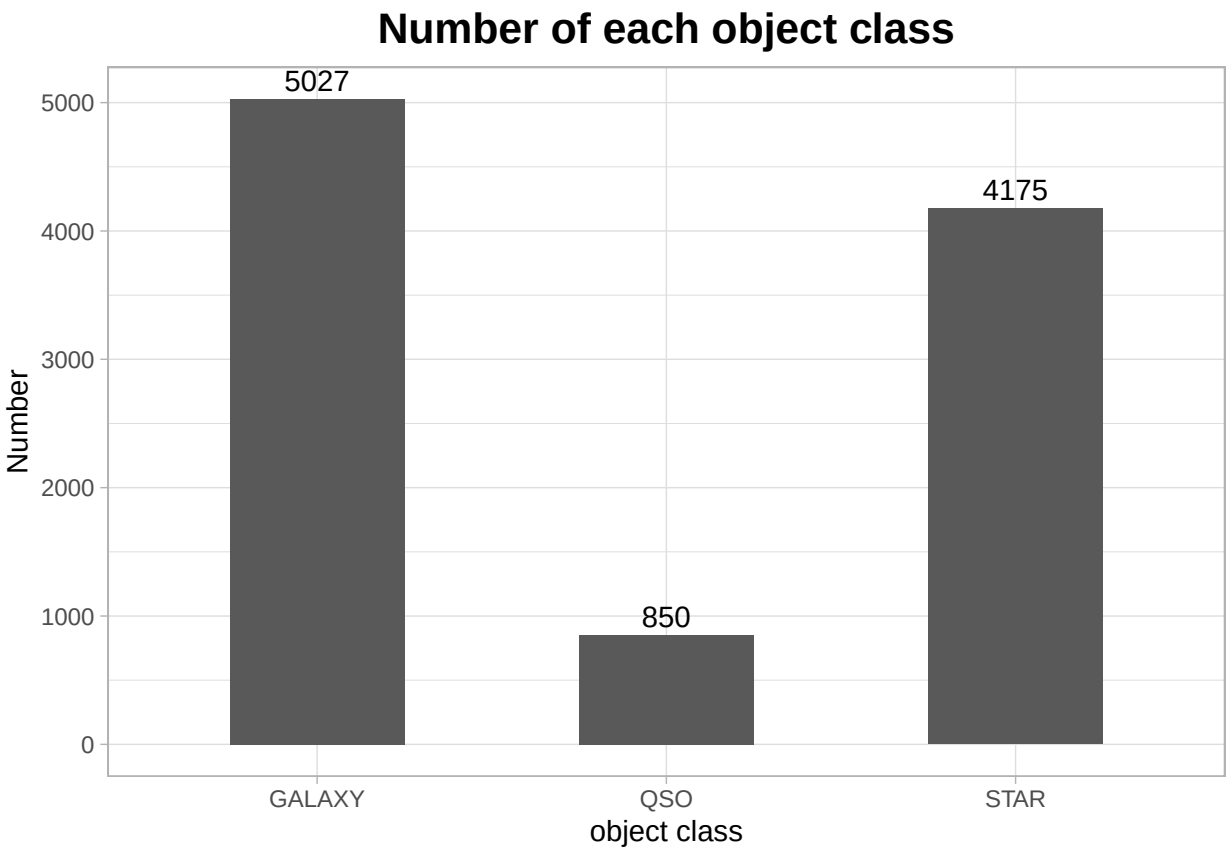
1.1.2 2. Analyse the class variable using appropriate statistics and visualisations

The class variable is Nominal data. Therefore, I chose bar plot to show the count of each object class. The appropriate statistics should be mode. The appropriate statistics should be mode. The most frequently class is GALAXY.

```
names(which.max(table(data$class)))
```

```
## [1] "GALAXY"
```

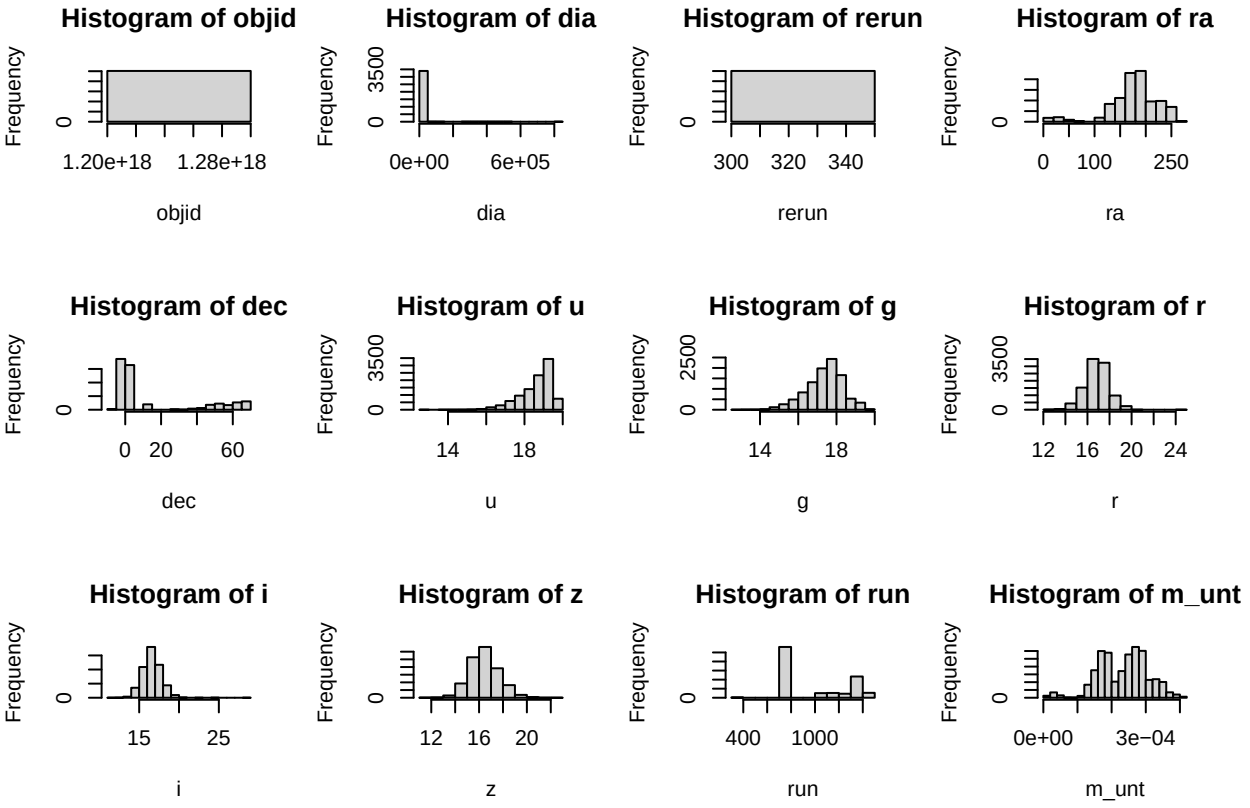
```
# Count the number of each class
count_class = group_by(data, class) %>% count
ggplot(data = count_class, aes(x = class, y = n))+
  geom_bar(stat = 'identity', width = 0.5)+
  theme_light()+
  ggtitle("Number of each object class")+
  theme(plot.title = element_text(face = "bold", hjust = 0.5, size = 16))+
  geom_text(aes(label = n, vjust = -0.4, hjust = 0.5))+
  labs(x = "object class", y = "Number")
```



1.1.3 3. Produce histograms for each input attribute.

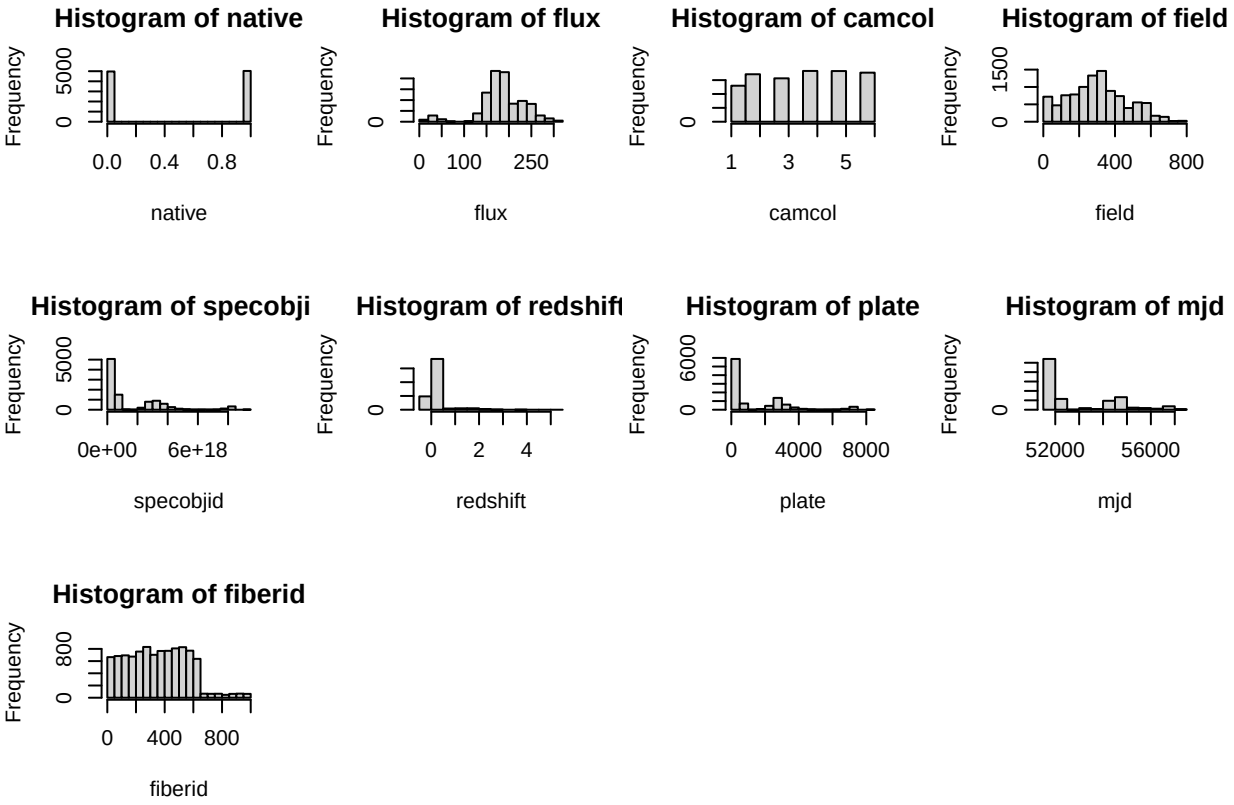
Firstly, I used par function to set grid for graphs as 3 rows and 4 columns in a page. Then, I used sapply with hist function to draw all the graphs once.

```
par(mfrow = c(3, 4))
sapply(names(features), function(columns)
  hist(features[[columns]], main = paste("Histogram of" , columns), xlab = columns))
```




```
##          objid                 dia                   rerun
## breaks   Numeric,2             Numeric,18            Integer,2
## counts   10052                 Integer,17            10022
## density  1e-17                 Numeric,17            0.02
## mids     1.25e+18              Numeric,17            325
## xname    "features[[columns]]" "features[[columns]]" "features[[columns]]"
## equidist TRUE                  TRUE                  TRUE
##          ra                    dec                   u
## breaks   Numeric,15            Numeric,17            Numeric,16
## counts   Integer,14            Integer,16            Integer,15
## density  Numeric,14            Numeric,16            Numeric,15
## mids     Numeric,14            Numeric,16            Numeric,15
## xname    "features[[columns]]" "features[[columns]]" "features[[columns]]"
## equidist TRUE                  TRUE                  TRUE
##          g                     r                     i
## breaks   Numeric,16            Integer,14            Integer,19
## counts   Integer,15            Integer,13            Integer,18
## density  Numeric,15            Numeric,13            Numeric,18
## mids     Numeric,15            Numeric,13            Numeric,18
## xname    "features[[columns]]" "features[[columns]]" "features[[columns]]"
## equidist TRUE                  TRUE                  TRUE
##          z                     run                   m_unt
## breaks   Integer,13            Integer,13            Numeric,22
## counts   Integer,12            Integer,12            Integer,21
## density  Numeric,12            Numeric,12            Numeric,21
## mids     Numeric,12            Numeric,12            Numeric,21
## xname    "features[[columns]]" "features[[columns]]" "features[[columns]]"
## equidist TRUE                  TRUE                  TRUE
```

```
##          native                flux                  camcol
## breaks   Numeric,21            Numeric,17            Numeric,11
## counts   Integer,20            Integer,16            Integer,10
## density  Numeric,20            Numeric,16            Numeric,10
## mids     Numeric,20            Numeric,16            Numeric,10
## xname    "features[[columns]]" "features[[columns]]" "features[[columns]]"
## equidist TRUE                  TRUE                  TRUE
##          field                 specobjid             redshift
## breaks   Numeric,17            Numeric,20            Numeric,13
## counts   Integer,16            Integer,19            Integer,12
## density  Numeric,16            Numeric,19            Numeric,12
## mids     Numeric,16            Numeric,19            Numeric,12
## xname    "features[[columns]]" "features[[columns]]" "features[[columns]]"
## equidist TRUE                  TRUE                  TRUE
##          plate                 mjd                   fiberid
## breaks   Numeric,18            Integer,13            Numeric,21
## counts   Integer,17            Integer,12            Integer,20
## density  Numeric,17            Numeric,12            Numeric,20
## mids     Numeric,17            Numeric,12            Numeric,20
## xname    "features[[columns]]" "features[[columns]]" "features[[columns]]"
## equidist TRUE                  TRUE                  TRUE
```

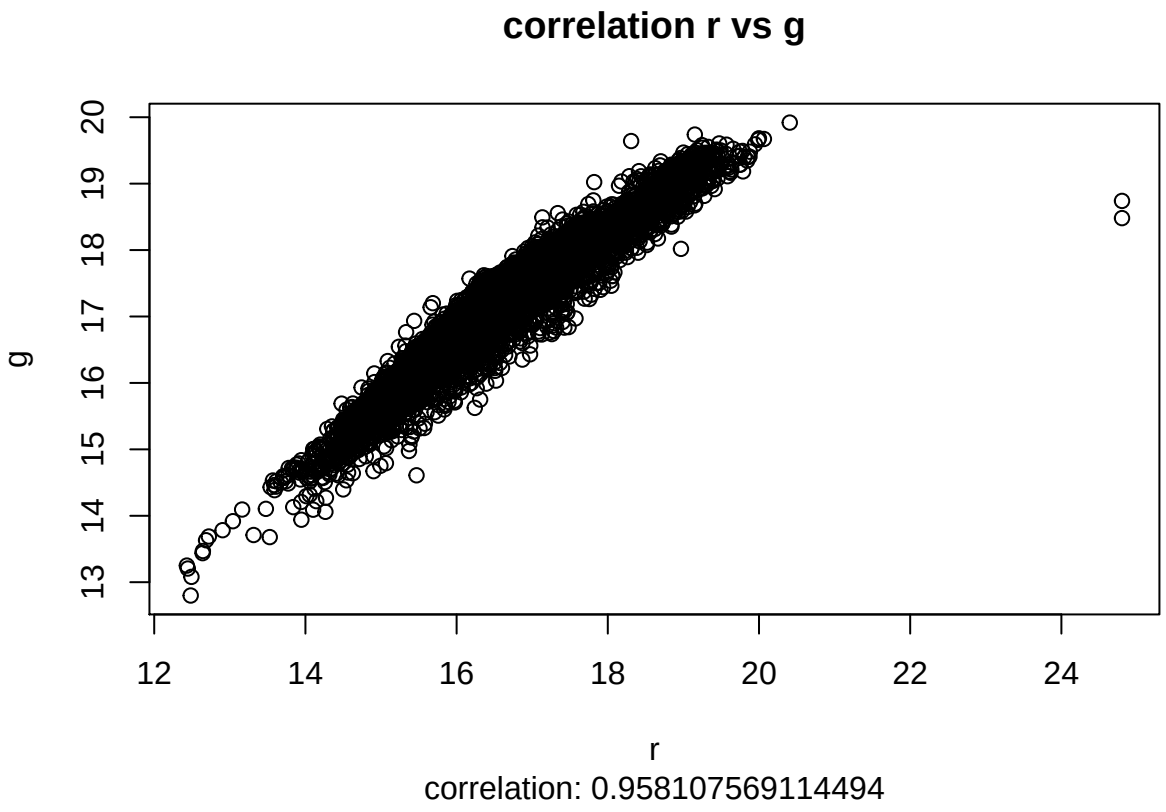


1.2 2. Explore the relationships between the attributes, and between the class and the attributes

1.2.1 1. Calculate the correlation and produce a scatterplot for the variables:r and g.

What does this correlation tell you about the relationships between these variables? High positive correlation

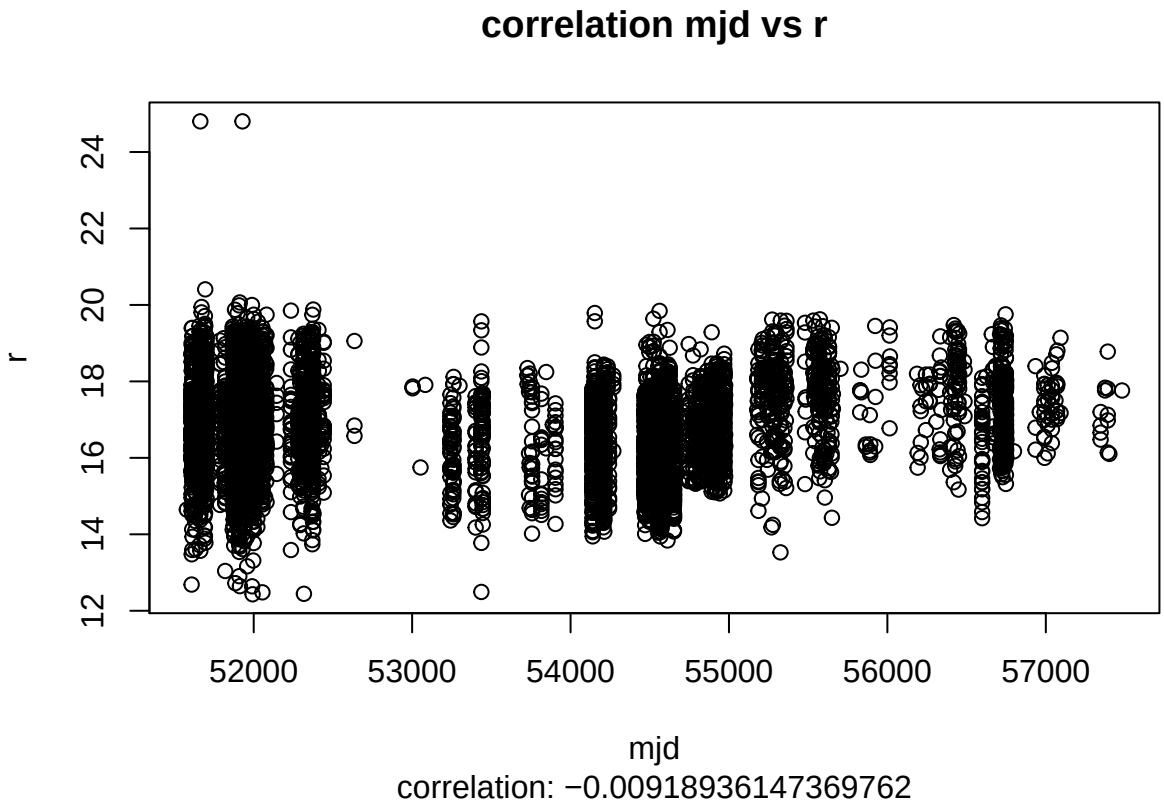
```
# Calculate pearson correlation, then plot
cor_value = cor(features$r, features$g, use="complete.obs")
plot(features$r, features$g, xlab="r", ylab="g",
     main="correlation r vs g", sub = paste("correlation:", cor_value))
```



1.2.2 2. Calculate the correlation and produce a scatterplot for the variables:mjd and r.

What does thiscorrelation tell you about the relationships between these variables? No correlation

```
# Calculate pearson correlation, then plot
cor_value = cor(features$mjd, features$r, use="complete.obs")
plot(features$mjd, features$r, xlab="mjd", ylab="r",
     main="correlation mjd vs r", sub = paste("correlation:", cor_value))
```



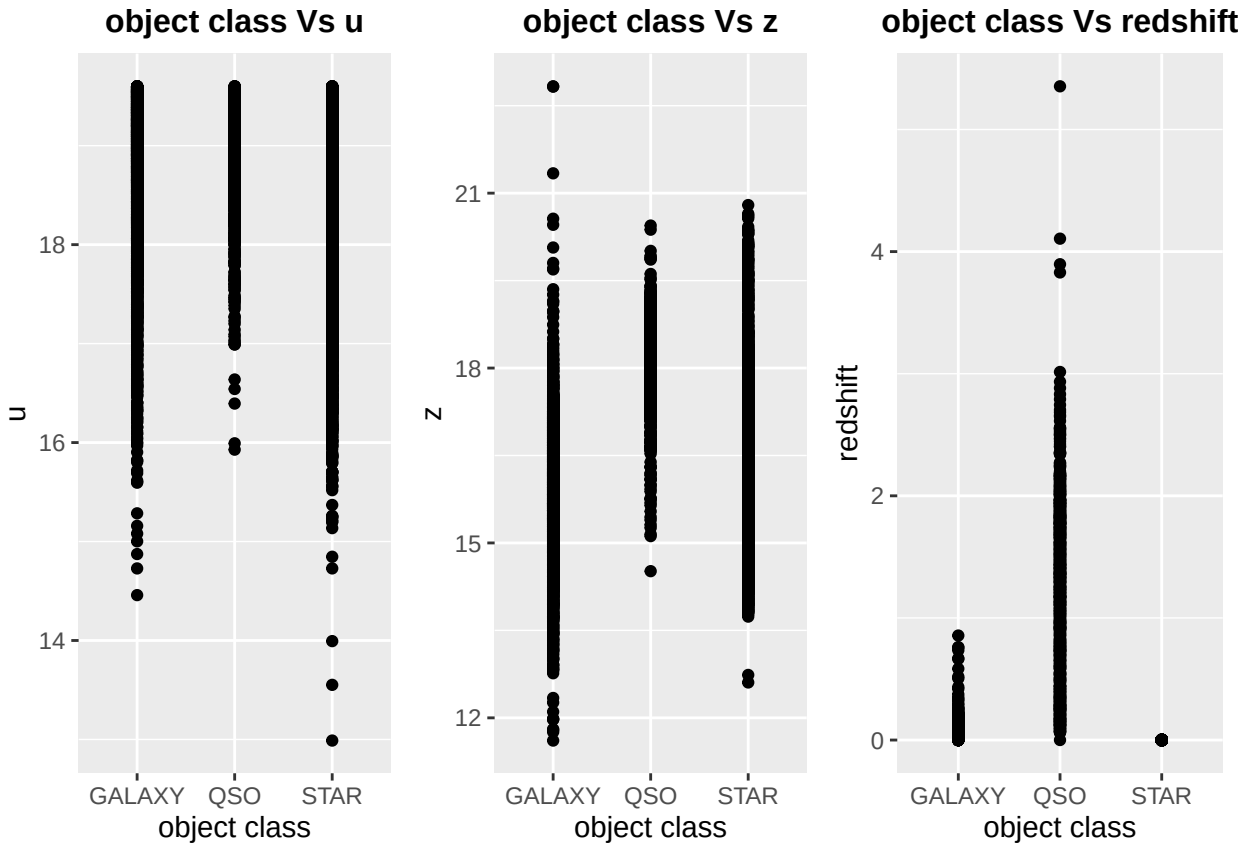
1.2.3 3. Produce scatter plots between the class variable and u,z, and redshift.

What do these three scatterplots tell you about the relationships between these variables and the class? No correlation. The range of redshift on object class are quite different(skewed)

```
p_u = ggplot(data, aes(x = class, y=u)) +
      geom_point() +
      theme(plot.title = element_text(face = "bold", hjust = 0.5, size = 12)) +
      ggtitle("object class Vs u") +
      labs(x = "object class", y = "u")

p_z = ggplot(data, aes(x = class, y=z)) +
      geom_point() +
      theme(plot.title = element_text(face = "bold", hjust = 0.5, size = 12)) +
      ggtitle("object class Vs z") +
      labs(x = "object class", y = "z")

p_redshift = ggplot(data, aes(x = class, y=redshift)) +
      geom_point() +
      theme(plot.title = element_text(face = "bold", hjust = 0.5, size = 12)) +
      ggtitle("object class Vs redshift") +
      labs(x = "object class", y = "redshift")
grid.arrange(p_u, p_z, p_redshift, nrow=1, ncol=3)
```

```
## Warning: Removed 50 rows containing missing values (geom_point).

## Warning: Removed 53 rows containing missing values (geom_point).
```

```
## Warning: Removed 50 rows containing missing values (geom_point).
```



1.2.4 4. Produce boxplots for all of the appropriate attributes.

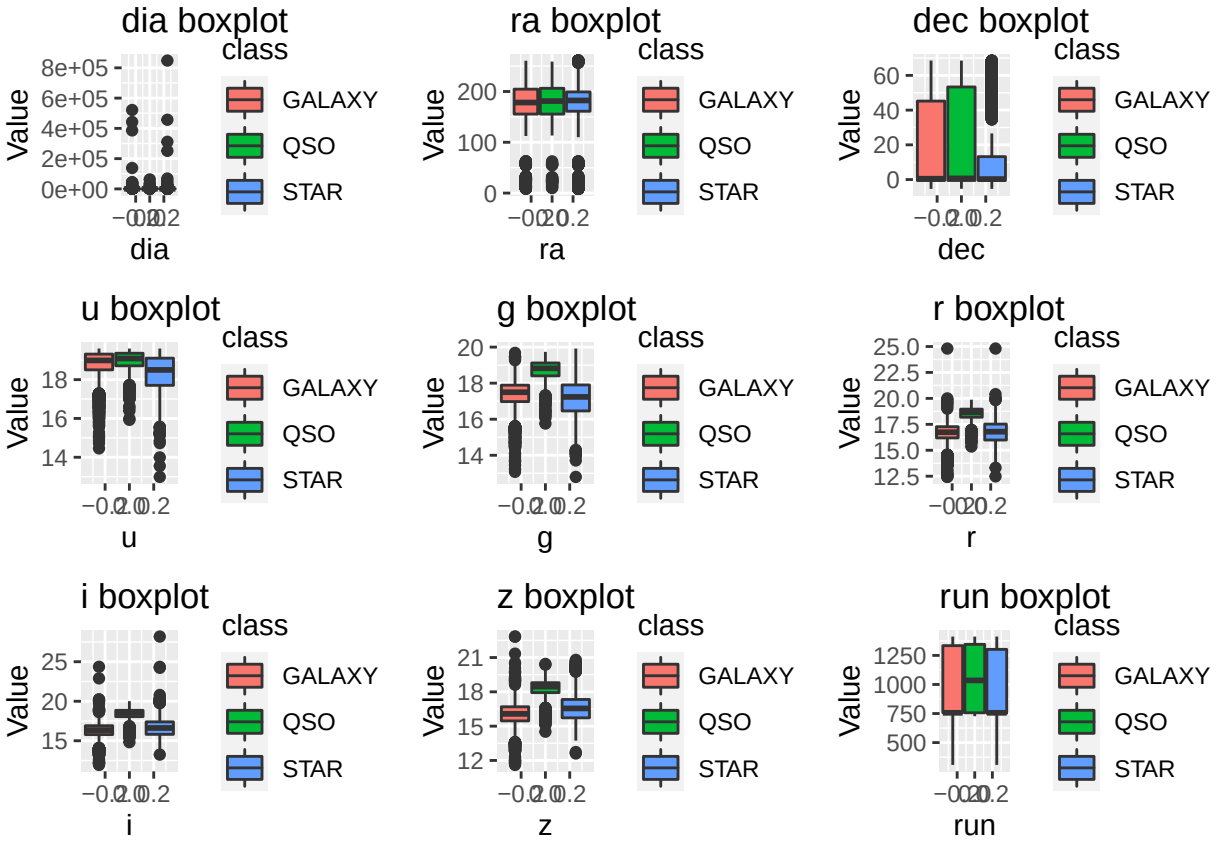
I chose appropriate attributes for boxplots based on whether the standard deviation of that feature is meaningful. If the standard deviation is 0 or N/A, there is no need to draw a boxplot.

```
appropriate_attr = c("dia", "ra", "dec", "u", "g", "r", "i",
                     "z", "run", "m_unt", "flux", "redshift", "plate", "mjd")

p_lst = list()
i = 1
for(attri in appropriate_attr){
  p_lst[[i]] = ggplot(data, aes_string(y = attri, fill = "class")) +
             geom_boxplot() +
             ggtitle(paste(attri, "boxplot"))+
             labs(x = attri, y = "Value")
  i = i + 1
}

grid.arrange(grobs = p_lst[1:9], ncol = 3)
```

```
## Warning: Removed 6646 rows containing non-finite values (stat_boxplot).
```

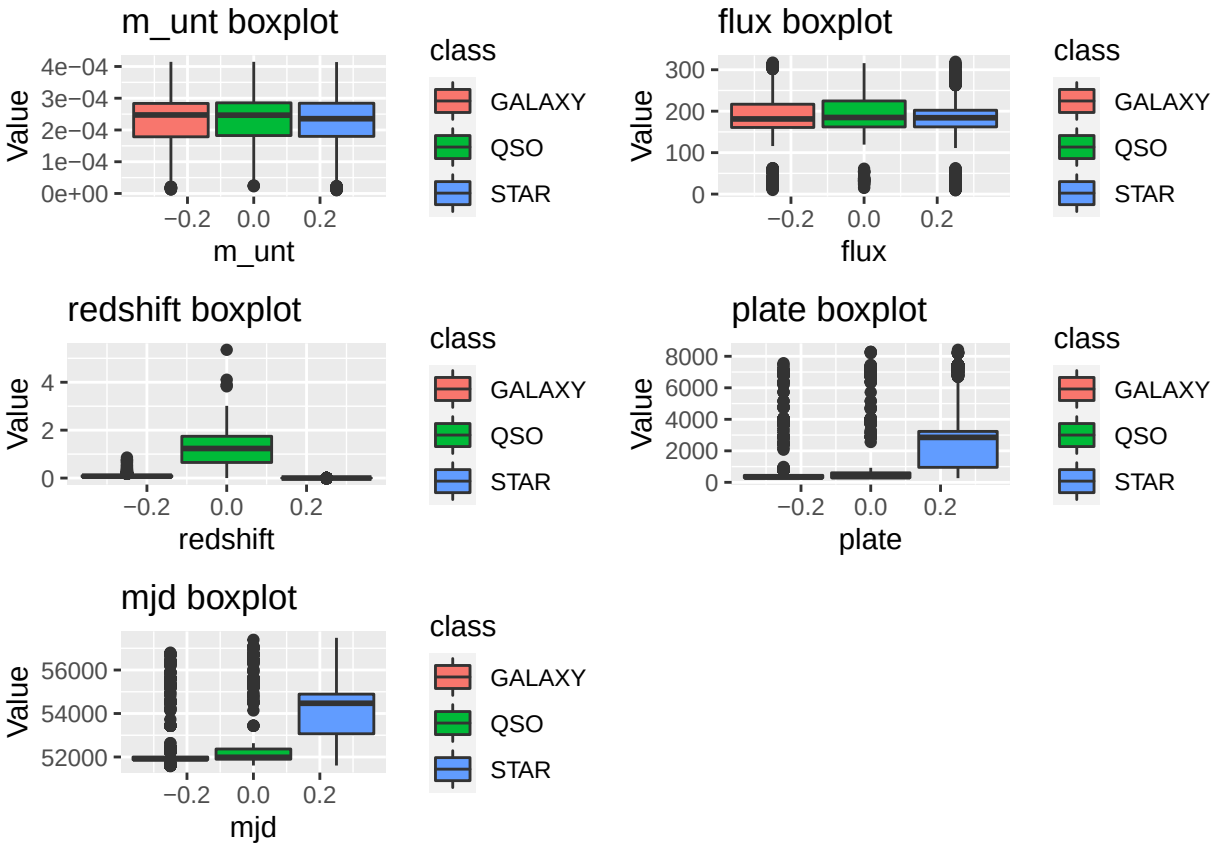
```
## Warning: Removed 48 rows containing non-finite values (stat_boxplot).
```

```
## Warning: Removed 49 rows containing non-finite values (stat_boxplot).

## Warning: Removed 50 rows containing non-finite values (stat_boxplot).

## Warning: Removed 51 rows containing non-finite values (stat_boxplot).

## Warning: Removed 53 rows containing non-finite values (stat_boxplot).

## Warning: Removed 50 rows containing non-finite values (stat_boxplot).

## Warning: Removed 53 rows containing non-finite values (stat_boxplot).

## Warning: Removed 50 rows containing non-finite values (stat_boxplot).
```




```
# The remaining box plot
grid.arrange(grobs = p_lst[10:14], ncol = 2)
```

```
## Warning: Removed 46 rows containing non-finite values (stat_boxplot).

## Warning: Removed 50 rows containing non-finite values (stat_boxplot).

## Warning: Removed 50 rows containing non-finite values (stat_boxplot).

## Warning: Removed 50 rows containing non-finite values (stat_boxplot).

## Warning: Removed 50 rows containing non-finite values (stat_boxplot).
```



1.3 3. General Conclusions

```
# Missing value by class of dia
aggregate(dia ~ class, data=data, function(x) {sum(is.na(x))}, na.action = NULL)
```

```
##    class  dia
## 1 GALAXY 3388
## 2    QSO  576
## 3   STAR 2682
```

1.4 4. Dealing with missing values in R

```
# Description: Missing data imputation
# Args:
#       df: R dataframe
#       criteria: Imputation criteria. group mean/ group media/ 0
# Return:
#       new_df: New R dataframe after data imputation
imputeData = function(df, criteria){
  if (criteria == "mean"){
    new_df = as.data.frame(mutate_at(group_by(df, class),
                                     vars(-group_cols()),
```

```
                                    funs(ifelse(is.na(.),
                                    mean(., na.rm = TRUE),.))))
  }else if (criteria == "median"){
    new_df = as.data.frame(mutate_at(group_by(df, class),
                                    vars(-group_cols()),
                                    funs(ifelse(is.na(.),
                                    median(., na.rm = TRUE),.))))
  }else if (criteria == 0){
    imputation = function(attri) replace(attri, is.na(attri), 0)
    new_df = replace(df, TRUE, lapply(df, imputation))
  }
  return(new_df)
}
```

```
# Replacement with 0, group mean and group median
data_0 = imputeData(data, 0)
data_mean = imputeData(data, "mean")
```

```
## Warning: `funs()` was deprecated in dplyr 0.8.0.
## Please use a list of either functions or lambdas:
##
##   # Simple named list:
##   list(mean = mean, median = median)
##
##   # Auto named with `tibble::lst()`:
##   tibble::lst(mean, median)
##
##   # Using lambdas
##   list(~ mean(., trim = .2), ~ median(., na.rm = TRUE))
```

```
data_median = imputeData(data, "median")
```

```
# Descriptive statistics to show the influence of different Imputation method.
dia_origin =data$dia
dia0 =data_0$dia
dia_mean =data_mean$dia
dia_median =data_median$dia
dia_data = data.frame(dia_origin, dia0, dia_mean, dia_median)
dia_data_metric = sapply(dia_data, function(x) c(mean(x, na.rm=TRUE),
                                                 median(x, na.rm=TRUE),
                                                 names(which.max(table(x))),
                                                 sd(x, na.rm=TRUE),
                                                 IQR(x, na.rm=TRUE),
                                                 max(x, na.rm=TRUE)-min(x, na.rm=TRUE),
                                                 sum(is.na(x))))
dia_data_df = as.data.frame(dia_data_metric,
                    row.names = c("Mean", "Median", "SD", "IQR", "Range", "Missing"))
dia_data_df = as.data.frame(t(dia_data_df))
dia_data_df
```

```
##                            V1              V2              V3              V4
## dia_origin 2000.02763891039      349.1009346      27.54042634  22677.730429659
```

```
## dia0       677.685449475605               0               0 13233.2851568431
## dia_mean   1989.38755752408 1912.34636563239 1912.34636563239 13201.5129640726
## dia_median 908.675720659727      348.6453718      348.6453718 13222.4841448636
##                          V5              V6   V7
## dia_origin    464.144166325 848144.24227366 6646
## dia0           234.42632895     848171.7827    0
## dia_mean   1578.99969403197 848144.24227366    0
## dia_median        1.3259056 848144.24227366    0
```

1.5 5. Attribute transformation

Using the three datasets generated in 1.4, explore the use of three transformation techniques (mean centering, normalisation and standardisation) to scale the attributes. Define, compare and contrast these approaches, and explain their effects on the data. mean centering, normalisation and standardisation

```
# mean centering
data_0_mc = mutate_if(data_0, is.numeric, function(x) scale(x, scale = FALSE))
data_mean_mc = mutate_if(data_mean, is.numeric, function(x) scale(x, scale = FALSE))
data_median_mc = mutate_if(data_median, is.numeric, function(x) scale(x, scale = FALSE))
```

```
# normalisation (x - min(x)) / (max(x) - min(x))
features_0_nor = as.data.frame(lapply(select(data_0, -class),
function(attri) (attri - min(attri, na.rm = TRUE)) / (max(attri, na.rm = TRUE) - min(attri, na.rm = TRU
data_0_nor = cbind(features_0_nor, select(data_0, class))

features_mean_nor = as.data.frame(lapply(select(data_mean, -class),
function(attri) (attri - min(attri, na.rm = TRUE)) / (max(attri, na.rm = TRUE) - min(attri, na.rm = TRU
data_mean_nor = cbind(features_mean_nor, select(data_mean, class))

features_median_nor = as.data.frame(lapply(select(data_median, -class),
function(attri) (attri - min(attri, na.rm = TRUE)) / (max(attri, na.rm = TRUE) - min(attri, na.rm = TRU
data_median_nor = cbind(features_median_nor, select(data_median, class))
```

```
# standardisation (x - mean(x)) / sd(x)
data_0_sd = mutate_if(data_0, is.numeric, scale)
data_mean_sd = mutate_if(data_mean, is.numeric, scale)
data_median_sd = mutate_if(data_median, is.numeric, scale)
```

```
# Descriptive statistics to show the influence of different scaling method.
ra_origin =data_mean$ra
ra_mc =data_mean_mc$ra
ra_nor =data_mean_nor$ra
ra_sd =data_mean_sd$ra
ra_data = data.frame(ra_origin, ra_mc, ra_nor, ra_sd)
ra_data_metric = sapply(ra_data, function(x) c(mean(x, na.rm=TRUE),
                                               median(x, na.rm=TRUE),
                                               sd(x, na.rm=TRUE),
                                               IQR(x, na.rm=TRUE),
                                               max(x, na.rm=TRUE)-min(x, na.rm=TRUE)))
ra_data_df = as.data.frame(ra_data_metric,
                           row.names = c("Mean", "Median", "SD", "IQR", "Range"))
ra_data_df = as.data.frame(t(ra_data_df))
ra_data_df
```

```
##                   Mean       Median         SD        IQR      Range
## ra_origin 1.755456e+02 180.19688180 47.6584139 43.8916530 252.649281
## ra_mc     1.500234e-15   4.65127454 47.6584139 43.8916530 252.649281
## ra_nor    6.622244e-01   0.68063436  0.1886347  0.1737256   1.000000
## ra_sd     3.306430e-17   0.09759608  1.0000000  0.9209634   5.301252
```

1.6 6. Attribute / instance selection

1.6.1 1. consider attribute and instance deletion strategies to deal with missing and duplicated values.

```
## Attribute deletion
# drop the attribute with over 50% NaN
missing_prop = colMeans(is.na(data))
missing_attri = missing_prop[missing_prop > 0.5]
missing_attri_name = names(missing_attri)
data_del = select(data, -missing_attri_name)
```

```
## Note: Using an external vector in selections is ambiguous.
## i Use ‘all_of(missing_attri_name)‘ instead of ‘missing_attri_name‘ to silence this message.
## i See <https://tidyselect.r-lib.org/reference/faq-external-vector.html>.
## This message is displayed once per session.
```

```
## instance deletion
# drop all missing value
data_del = na.omit(data_del)
# drop all duplicated instances
data_del = unique(data_del)
# drop all duplicated columns
col_duplicated = sapply(data_del, function(x) length(unique(x)))
col_duplicated_name = names(col_duplicated[col_duplicated == 1])
data_del = select(data_del, -all_of(col_duplicated_name))
```

1.6.2 2. Use correlations between attributes to reduce the number of attributes.

```
# Drop the attribute with >50% Missing value
data_cor = select(data, -c("dia"))
# drop all missing value
data_cor = na.omit(data_cor)
```

```
# Pearson correlation
# Ignore the ordinal, nominal and class attributes
cor_abs = abs(cor(select(data_cor, -c("camcol", "objid", "rerun",
                                      "native", "field", "specobjid",
                                      "fiberid", "class")))) > 0.95
cor_abs_df = as.data.frame(cor_abs)
cor_abs_df = replace(cor_abs_df, cor_abs_df == FALSE, 0)
cor_abs_df
```

```
##          ra dec u g r i z run m_unt flux redshift plate mjd
## ra        1   0 0 0 0 0 0   0     0    1        0     0   0
## dec       0   1 0 0 0 0 0   0     0    0        0     0   0
## u         0   0 1 0 0 0 0   0     0    0        0     0   0
## g         0   0 0 1 1 0 0   0     0    0        0     0   0
## r         0   0 0 1 1 1 1   0     0    0        0     0   0
## i         0   0 0 0 1 1 1   0     0    0        0     0   0
## z         0   0 0 0 1 1 1   0     0    0        0     0   0
## run       0   0 0 0 0 0 0   1     0    0        0     0   0
## m_unt     0   0 0 0 0 0 0   0     1    0        0     0   0
## flux      1   0 0 0 0 0 0   0     0    1        0     0   0
## redshift  0   0 0 0 0 0 0   0     0    0        1     0   0
## plate     0   0 0 0 0 0 0   0     0    0        0     1   1
## mjd       0   0 0 0 0 0 0   0     0    0        0     1   1
```

```
high_cor = c("r","z","flux","plate")
data_cor = select(data_cor, -all_of(high_cor))
# 17 features left
```

1.7 7. Attribute transformation / reduction

Use Principal Component Analysis. Explain your process, along with the results obtained. Appropriate pre-processing steps: no missing values, no objid/rerun- duplicated value.

```
# drop the attribute with over 50% NaN and all same value
data_pca = select(data, -c("objid", "dia", "rerun"))
# drop all missing value
data_pca = na.omit(data_pca)
# drop all duplicated value
data_pca = unique(data_pca)
```

1.7.1 1.Reduced to 12 dimensions (i.e: PC1-PC12).

```
data_stand = as.data.frame(scale(select(data_pca, -class)))
pca = prcomp(data_stand, scale=T)
pca_info = summary(pca)
data_pca12 = pca$x[,1:12]
data_pca12 = cbind(data_pca12, select(data_pca, class))
data_pca_all = cbind(pca$x, select(data_pca, class))
```

```
pca_info
```

```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     2.1503 1.8183 1.6764 1.4179 1.16529 0.99972 0.93516
## Proportion of Variance 0.2569 0.1837 0.1561 0.1117 0.07544 0.05552 0.04859
## Cumulative Proportion  0.2569 0.4405 0.5967 0.7084 0.78381 0.83934 0.88792
##                           PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.73255 0.67803 0.65764 0.60273 0.35903 0.20498 0.13920
## Proportion of Variance 0.02981 0.02554 0.02403 0.02018 0.00716 0.00233 0.00108
```

```
## Cumulative Proportion  0.91773 0.94327 0.96730 0.98748 0.99465 0.99698 0.99806
##                           PC15    PC16    PC17      PC18
## Standard deviation     0.12526 0.11617 0.07620 0.0005279
## Proportion of Variance 0.00087 0.00075 0.00032 0.0000000
## Cumulative Proportion  0.99893 0.99968 1.00000 1.0000000
```

```
# Show the pearson of the data after PCA
pca_cor = abs(cor(pca_info$x)) > 0.1
pca_cor_df = as.data.frame(pca_cor)
pca_cor_df = replace(pca_cor_df, pca_cor_df == FALSE, 0)
pca_cor_df
```

```
##      PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12 PC13 PC14 PC15 PC16
## PC1    1   0   0   0   0   0   0   0   0    0    0    0    0    0    0    0
## PC2    0   1   0   0   0   0   0   0   0    0    0    0    0    0    0    0
## PC3    0   0   1   0   0   0   0   0   0    0    0    0    0    0    0    0
## PC4    0   0   0   1   0   0   0   0   0    0    0    0    0    0    0    0
## PC5    0   0   0   0   1   0   0   0   0    0    0    0    0    0    0    0
## PC6    0   0   0   0   0   1   0   0   0    0    0    0    0    0    0    0
## PC7    0   0   0   0   0   0   1   0   0    0    0    0    0    0    0    0
## PC8    0   0   0   0   0   0   0   1   0    0    0    0    0    0    0    0
## PC9    0   0   0   0   0   0   0   0   1    0    0    0    0    0    0    0
## PC10   0   0   0   0   0   0   0   0   0    1    0    0    0    0    0    0
## PC11   0   0   0   0   0   0   0   0   0    0    1    0    0    0    0    0
## PC12   0   0   0   0   0   0   0   0   0    0    0    1    0    0    0    0
## PC13   0   0   0   0   0   0   0   0   0    0    0    0    1    0    0    0
## PC14   0   0   0   0   0   0   0   0   0    0    0    0    0    1    0    0
## PC15   0   0   0   0   0   0   0   0   0    0    0    0    0    0    1    0
## PC16   0   0   0   0   0   0   0   0   0    0    0    0    0    0    0    1
## PC17   0   0   0   0   0   0   0   0   0    0    0    0    0    0    0    0
## PC18   0   0   0   0   0   0   0   0   0    0    0    0    0    0    0    0
##      PC17 PC18
## PC1     0    0
## PC2     0    0
## PC3     0    0
## PC4     0    0
## PC5     0    0
## PC6     0    0
## PC7     0    0
## PC8     0    0
## PC9     0    0
## PC10    0    0
## PC11    0    0
## PC12    0    0
## PC13    0    0
## PC14    0    0
## PC15    0    0
## PC16    0    0
## PC17    1    0
## PC18    0    1
```

1.7.2 2. How many PCs should be used to obtain a cumulative variance of at least 90%?

```
# Description: Automatically chose dimensions by variance threshold
# Args:
#       data: The data without labels
#       thres: The setting variance threshold
# Return:
#       chosen_pca: The chosen dimensions pca
auto_pca = function(data, thres){
  data_stand = as.data.frame(scale(data))
  data_pca = prcomp(data_stand, scale=T)
  pca_info = summary(data_pca)
  cumulative_proportion = tail(pca_info$importance,1)
  whole_len = length(cumulative_proportion)
  len = sum(cumulative_proportion < thres) + 1
  len = ifelse(len > whole_len, whole_len, len)
  chosen_pca = data_pca$x[,1:len]
  return (chosen_pca)
}
```

```
# 90% cumulative variance
data_pca90p = auto_pca(select(data_pca, -class), 0.9)
data_pca90p = cbind(data_pca90p, select(data_pca, class))
```

###17 Dataset from Part 1

```
# imputation with 0/mean/Median
dim(data_0)
```

```
## [1] 10052    22
```

```
dim(data_mean)
```

```
## [1] 10052    22
```

```
dim(data_median)
```

```
## [1] 10052    22
```

```
# mean center scale
dim(data_0_mc)
```

```
## [1] 10052    22
```

```
dim(data_mean_mc)
```

```
## [1] 10052    22
```

```
dim(data_median_mc)
```

```
## [1] 10052    22
```

```
# normalisation
dim(data_0_nor)
```

```
## [1] 10052    22
```

```
dim(data_mean_nor)
```

```
## [1] 10052    22
```

```
dim(data_median_nor)
```

```
## [1] 10052    22
```

```
# standardisation
dim(data_0_sd)
```

```
## [1] 10052    22
```

```
dim(data_mean_sd)
```

```
## [1] 10052    22
```

```
dim(data_median_sd)
```

```
## [1] 10052    22
```

```
# deletion Missing and duplicated attributes
dim(data_del)
```

```
## [1] 9993   19
```

```
# Drop high correlation attributes and Missing value
dim(data_cor)
```

```
## [1] 9995   17
```

```
# PCA
dim(data_pca_all)
```

```
## [1] 9994   19
```

```
dim(data_pca12)
```

```
## [1] 9994   13
```

```
dim(data_pca90p)
```

```
## [1] 9994    9
```

```
## Write all the datasets to csv
# imputation with 0/mean/Median
write.csv(data_0, "data_0.csv", row.names = FALSE)
write.csv(data_mean, "data_mean.csv", row.names = FALSE)
write.csv(data_median, "data_median.csv", row.names = FALSE)
# mean center scale
write.csv(data_0_mc, "data_0_mc.csv", row.names = FALSE)
write.csv(data_mean_mc, "data_mean_mc.csv", row.names = FALSE)
write.csv(data_median_mc, "data_median_mc.csv", row.names = FALSE)
# normalisation
write.csv(data_0_nor, "data_0_nor.csv", row.names = FALSE)
write.csv(data_mean_nor, "data_mean_nor.csv", row.names = FALSE)
write.csv(data_median_nor, "data_median_nor.csv", row.names = FALSE)
# standardisation
write.csv(data_0_sd, "data_0_sd.csv", row.names = FALSE)
write.csv(data_mean_sd, "data_mean_sd.csv", row.names = FALSE)
write.csv(data_median_sd, "data_median_sd.csv", row.names = FALSE)
# deletion Missing and duplicated attributes
write.csv(data_del, "data_del.csv", row.names = FALSE)
# Drop high correlation attributes and Missing value
write.csv(data_cor, "data_cor.csv", row.names = FALSE)
# PCA
write.csv(data_pca_all, "data_pca_all.csv", row.names = FALSE)
write.csv(data_pca12, "data_pca12.csv", row.names = FALSE)
write.csv(data_pca90p, "data_pca90p.csv", row.names = FALSE)
```

2 2. CLUSTERING

```
# Description: calculate the results of different clustering algorithm
# Args:
#       data_all: R dataframe read from the dataset
#   distance_method: different different method.("euclidean" and "manhattan"). For hca and pam algorith
#   cut_tree: The heights where the tree should be cut. For hca algorithm.
#   random_sets: The number of chosen random sets. For kmeans and pam algorithm.
#   iterations: the maximum number of iterations allowed. For kmeans algorithm.
# Return:
#       results: The aggregate results after clustering
clusterCalculator = function(data_all, distance_method, cut_tree, random_sets, iterations){

    # Only use features
    data_clus = select(data_all, -class)
```

```
    results = data.frame(class = select(data_all, class), hca=0, kmeans = 0, pam = 0)
    ## 3 groups
    k = 3

    ## HCA
    hca_res = hclust(dist(data_clus, method = distance_method))
    results$hca = cutree(hca_res, k, h = cut_tree)
    ## K-means
    km_res = kmeans(data_clus, k, iter.max = iterations, nstart = random_sets)
    results$kmeans = km_res$cluster
    ## PAM
    pam_res = pam(data_clus, k, metric = distance_method, nstart = random_sets)
    results$pam = pam_res$clustering
    return(results)
}
```

```
# Description: Return max diag matrix
# Args:
#       matr: confusion matrix
# Return:
#       max_matr: aligned matrix
maxDiag = function(matr){
  diagMax = -Inf
  col_num = dim(matr)[2]
  # Try all the combinations
  for(i in 1: nrow(permutations(col_num))){
    new_matr = matr[,c(permutations(col_num)[i,])]
    dig_sum = sum(diag(new_matr))
    if(dig_sum > diagMax){
      diagMax = dig_sum
      max_matr = new_matr
    }
  }
return(max_matr)
}
```

```
## External Metrics
# Description: calculate External metrics from aligned matrix
# Args:
#       matr: confusion matrix
#   model_name: the name of models. str type
# Return:
#       results: External metrics from aligned matrix
metricCal = function(matr){
  diag_value = diag(matr)
  len = length(diag_value)
  accuracy = sum(diag_value) / sum(matr)

  recall = sapply(c(1:len), function(x) diag_value[x] / sum(matr[x,]))
  precision = sapply(c(1:len), function(x) diag_value[x] / sum(matr[,x]))
  f1_score = sapply(c(1:len), function(x)  2 * recall[x] * precision[x] / (recall[x] + precision[x]))
  results = cbind(matr, recall, precision, f1_score)
```

```
  results = rbind(results, Average = c(NA, NA, NA, mean(recall), mean(precision), mean(f1_score)))
  results = cbind(results, accuracy= c(NA, NA, NA, accuracy))
  return(results)
}
```

2.0.1 2. parameters tuning

```
### Dataset
## Self-chosen
#dataset = "data_median_nor.csv"

## PCA - i&ii
#dataset = "data_pca_all.csv"
#dataset = "data_pca12.csv"

## Deletion Missing and duplicated attributes - iii
#dataset = "data_del.csv"

## Drop high correlation attributes and Missing value -iii
#dataset = "data_cor.csv"

## Mean center scale - iV
#dataset = "data_0_mc.csv"
#dataset = "data_mean_mc.csv"
dataset = "data_median_mc.csv"


# Parameters
distance_method = "euclidean" # "euclidean" and "manhattan". For hca and pam algorithm.
cut_tree = 10 # 10,50,100. For hca algorithm.
random_sets = 3 # 1,3,5. For kmeans and pam algorithm.
iterations = 50 # 10,50,100. For kmeans algorithm.

# Export Excel name
inter_excel_name = "internal_metrics_df16.xls"
ext_excel_name = "all_ext_metric_df16.xls"

data_all = read.csv(dataset)
# Get column with all Nan
nan_col = sapply(data_all, function(x) all(is.na(x)))

cluster_results = clusterCalculator(data_all[,!nan_col],
                                    distance_method,
                                    cut_tree,
                                    random_sets,
                                    iterations)

## Internal Metrics
distance = dist(select(data_all, -class))
cluster_res = select(cluster_results, c("hca", "kmeans", "pam"))
internal_metrics = sapply(cluster_res, function(x) cluster.stats(distance,
                                                                 clustering = x,
```

```
                                                    silhouette = TRUE))
all_internal_metrics_df = as.data.frame(internal_metrics)
internal_metrics_df = all_internal_metrics_df[c("dunn","average.between","average.within"),]

## External Metrics
t_hca = table(cluster_results$class, cluster_results$hca)
t_kmeans = table(cluster_results$class, cluster_results$kmeans)
t_pam = table(cluster_results$class, cluster_results$pam)

aligned_matr_hca = maxDiag(t_hca)
aligned_matr_kmeans = maxDiag(t_kmeans)
aligned_matr_pam = maxDiag(t_pam)

hca_ext_metric = metricCal(aligned_matr_hca)
kmeans_ext_metric = metricCal(aligned_matr_kmeans)
pam_ext_metric = metricCal(aligned_matr_pam)
all_ext_metric = rbind(hca_ext_metric, kmeans_ext_metric, pam_ext_metric)
# results order: "hca", "kmeans", "pam"
all_ext_metric_df = as.data.frame(all_ext_metric)

# Write to csv for report
#write.xlsx(internal_metrics_df, file = inter_excel_name, sheetName = "internal_metrics_df", append = T
#write.xlsx(all_ext_metric_df, file = ext_excel_name, sheetName = "all_ext_metric_df", append = TRUE)
```