TECHNEX INTERNSHIPS

PROJECT TITLE:

EDA AND CLASSIFICATION IN ASTRONOMY

SUBMITTED BY: THIRSHA SREE H

DATE: 10-09-2022

BATCH CODE:22DAT-029 – DATA SCIENCE (JULY 2022)

SUBMITTED TO:

Mr. Mayur Dev Sewak

General Manager, Operations

Eisystems Services

&

Ms. Mallika Srivastava

Trainer, Data Science & Analytics Domain

Eisystems Services

CONTENT TABLE

LIST OF FIGURES:

LIST OF TABLES:

A table is an arrangement of information or data, typically in rows and columns, or possibly in a more complex structure.

ABSTRACT OF THE PROJECT

To perform Exploratory Data Analysis on Sloan Digital Sky Survey (SDSS) Data Release 17 (DR17). This catalogue contains photometric data for all objects viewed through a telescope and spectroscopic data for a small part of these. Train ML classification models on the labelled Photometric data which is spectroscopically classified with labels. The EDA helps to choose the right models for the dataset. All the information gained from conducting EDA to help you choose a data model. I will use the trained models which have the highest accuracy on the training set , on new unclassified data. Various machine learning algorithms will be used to predict three classes - stars , galaxies , quasars. We will consider KNN , SVM , random forest algorithms for classification . Scaling of data will be done to get better results. The results will help astronomers and astrophysicists carry out further studies.

PROBLEM STATEMENT

It becomes increasingly important to have more accurate and detailed object recognition of Galaxy, Star or Quasar towards a more comprehensive understanding of images. In this context, object detection—a task that involves accurately identifying the type and location of objects contained in images—is important in addition to image classification. Image classification, which is defined as determining the class of the image, was one of the main issues. The challenge of image localization, when a single object is present in the image and the system must forecast its class and location in the image (a bounding box around the object), is a little bit complex.

1. So, I performed EDA on SDSS Dataset to to understand which features of the Astronomy data contribute most to the Classification.
2. Do feature Engineering for the suited ML Models and tuning them.
3. Predict Class column (Dependent Variable) based on Independent Variables. Task is to identify a Star, Galaxy or Quasar from a new set of data points.

OBJECTIVES OF THE PROJECT

The aim is to investigate the appropriateness of the application of certain ML models . To the best of our knowledge , SVM and KNN has been previously tried for this classification. But I have improved the performance by doing exploratory analysis on the dataset and removing outliers.

Various machine learning algorithms is used to predict three classes - Stars , Galaxies , Quasars. We will primarily consider KNN , SVM , random forest algorithms for classification and other models will be used if the EDA gives such inferences. Scaling of data will be done to get better results. The results will help astronomers and astrophysicists carry out further studies.

MOTIVATION:

It is difficult for astronomers to study quasars by relying on telescopic observations with template manual matching alone since quasars are difficult to distinguish from stars due to their great distance from Earth. Hence ,I present methods which will help them to distinguish celestial objects.

INTRODUCTION

- The classification scheme of Galaxies, Quasars, and Stars is one of the most fundamental work in Astronomy. **Quasars** are the brightest and most distant objects in the universe. The biggest **groups of stars** are called **galaxies**.
- The dataset is obtained from the SDSS server using their in built SQL querying.
- The Dataset has 500,000 observations (rows) of space. Every observation is described by 18 feature columns in which 1 column i.e. class column is a dependent variable or target variable. And we have to determine the outcome of the class column, i.e. whether it is a star, Galaxy or Quasar.
- The SDSS gives a three-dimensional picture of the universe through a volume one hundred times larger than that explored to date. Data Release 17 (DR17) is the final data release of the fourth phase of the Sloan Digital Sky Survey (SDSS-IV). DR17 contains SDSS observations through January 2021. The SDSS is the first large-area survey to use electronic light detectors, so the images it produces will be substantially more sensitive and accurate than earlier surveys, which relied on photographic plates.
- The results of the SDSS are electronically available to the scientific community and the general public, both as images and as precise catalogues of all objects discovered. By the end of the survey, the total quantity of information produced is about 15 terabytes.
- The data consists of 5,00,000 observations of space taken by the SDSS. Every observation is described by 17 feature columns and 1 class column which identifies it to be either a star, galaxy or quasar.

- The dataset offers plenty of information about space to explore. Also, the class column is the perfect target for classification practices.
- The classification models under consideration are :

- k-Nearest Neighbours.
- Decision Trees.
- Naive Bayes.
- Random Forest.
- Gradient Boosting.
- SVM.

PROCESS OF PROJECT DEVELOPMENT:

I did Univariate and Bivariate analysis on the dataset and describe the featureset. Then, I introduced ML algorithms to describe how models were optimised using training dataset. In this work, I applied supervised, unsupervised and semi supervised machine learning algorithms to classify SDSS data.I have divided spectroscopically-classified dataset into a training dataset and a testing dataset. The training set is used to train the machine learning classifier and fit the machine learning model. The test dataset is used to derive a variety of performance metrics which assess how the machine learning model would perform on unseen data , which will classify as stars , quasars and galaxies.

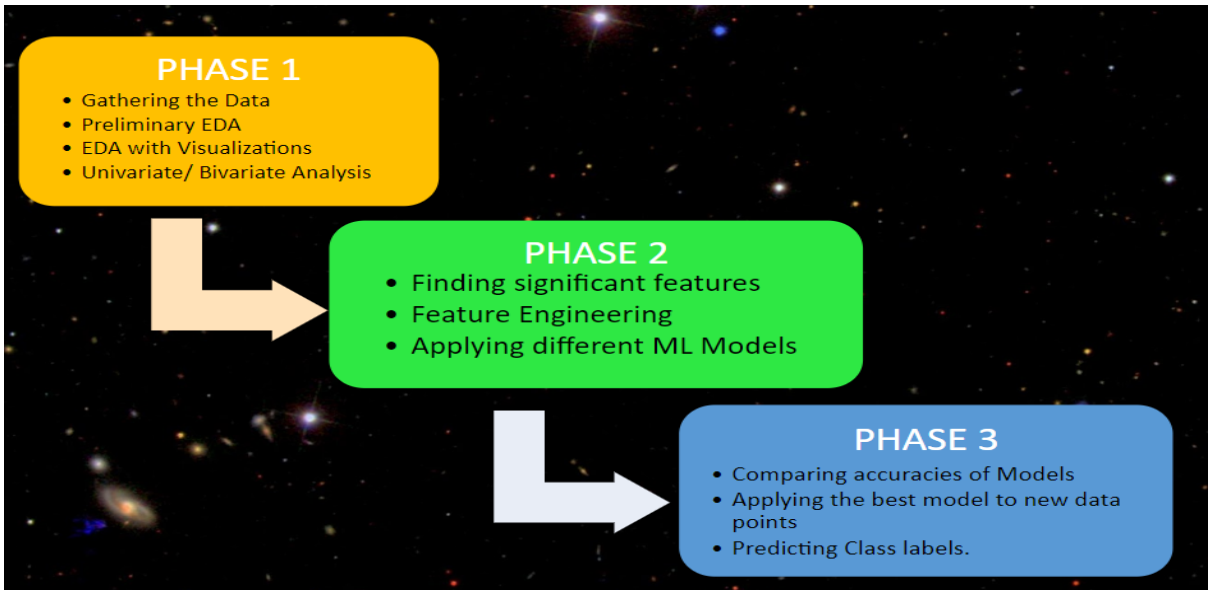
The above paper uses Data Release III, but we are going to use the latest and final addition to the SDSS Catalogues which is the DR17. This will be a superset of all the previous data releases giving a much more consistent and cleaner set of data points. Data Release 17 (DR17) is the final data release of the fourth phase of the Sloan Digital Sky Survey (SDSS-IV). DR17 contains SDSS observations through January 2021.

SDSS data releases are cumulative, so DR17 includes all the sky coverage of prior releases.

We will be using more fine tuned ML classification models and find the one with the maximum accuracy.

- Perform EDA on SDSS Dataset to to understand which features of the Astronomy data contribute most to the Classification.
- Do feature Engineering for the suited ML Models and tuning them.
- Predict Class column (Dependent Variable) based on Independent Variables. Task is to identify a Star, Galaxy or Quasar from a new set of data points.

PROJECT PLAN



METHODOLOGY:

With the given dataset I have used EDA approach with machine learning algorithm to make images more accurate with time constraint. First we will Analysis EDA with step and techniques, then will go to machine learning algorithm and interpretation

Exploratory Data Analysis (EDA) is an approach to analysing data sets to summarize their main characteristics, often with visual methods. Exploratory data analysis can help detect obvious errors, identify outliers in datasets, understand relationships, unearth important factors, find patterns within data, and provide new insights. Following are the different steps involved in EDA :

1. Data Collection
2.Data Cleaning
3.Data Preprocessing
4. Data Visualisation

Data collection is the process of gathering information in an established systematic way that enables one to test hypothesis and evaluate outcomes easily.

After getting data we need to check the data-type of features.

There are following types of features:

- numeric
- categorical
- ordinal

- datetime
- coordinates

Data Cleaning is the process of ensuring that your data is correct and useable by identifying any errors in the data, or missing data by correcting or deleting them. Refer to this link for data cleaning.

Once the data is clean we can go further for data preprocessing.

Data Preprocessing is a data mining technique that involves transforming raw data into an understandable format. It includes normalisation and standardisation, transformation, feature extraction and selection, etc. The product of data preprocessing is the final training dataset.

Data Visualisation is the graphical representation of information and data. It uses statistical graphics, plots, information graphics and other tools to communicate information clearly and efficiently.

Here we will focus on commonly used Seaborn visualisation. Seaborn is a Python data visualisation library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

Exploratory Data Analysis Techniques

1)UNIVARIATE ANALYSIS

- Box Plot
- Frequency Polygon
- Histplot
- Problem Density Function
- Violin Plot
- KDE Plot

2) MULTIVARIATE ANALYSIS:

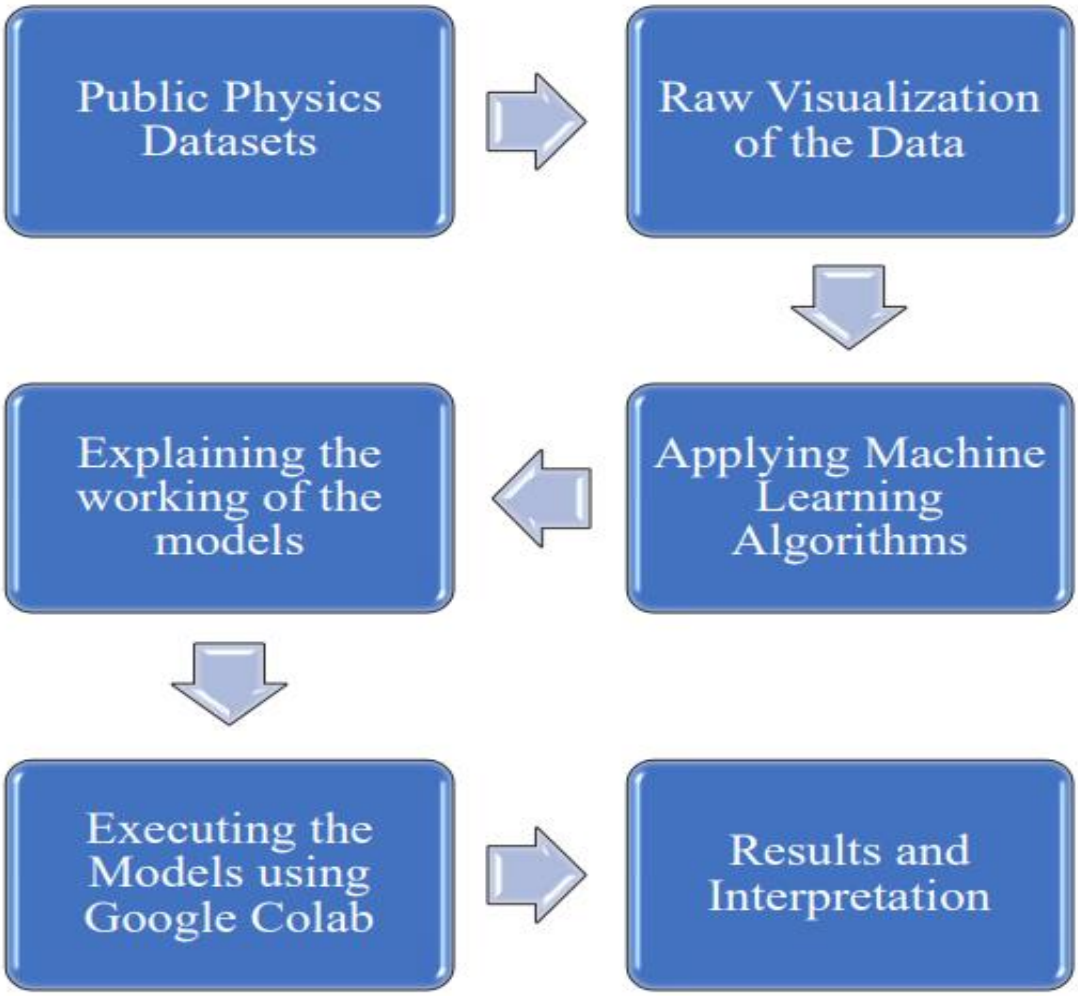
- Line Plot
- Sub Plot
- Scatter Plot
- Pair Plot
- Ra vs Dec : Equatorial Coordinates Plot

ML Algorithm:

- Logistic Regression
- Knn Neighbors

- Naïve Bayes
- Random Forest
- Support Vector Machines
- Decision tree

Neural Network : Multi layer Perception Classifier(MLPC)



SYSTEM REQUIREMENTS:

It has always been the pursuit of astronomers to study, analyze and understand the universe with its countless celestial objects of endless variety. However, due to the celestial objects differing distances, light- emitting intensities and positions in space, they often appear as near identical dots when recorded using photometric data from telescopes , making it difficult to differentiate from each other. Therefore , the classification of celestial objects is very fundamental in astronomy for astronomers to be able to collect meaningful samples to uncover complex and mysterious past , present and future of the universe

1.Hardware Specifications:

1. Supported Processor Architectures
    a. Intel (or compatible) 32 and 64 bit.
    b. PowerPC 32 and 64 bit.
    c. ARM 32 and 64 bit.
    d. MIPS 64 bit.
2. 1GB RAM (2G RAM recommended);
3. Up to 10GB available hard disk space;
4. Screen resolution of 1280x1024 works but 1920x1080 or higher is recommended.

2.Software Requirements:

1. SDSS dataset
2. Python 3.7
3. Jupyter Notebook
4. Machine Learning Libraries

ALGORITHM:

KNN Neigbhor Algorithm:

Step 1 − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 − For each point in the test data do the following −

- **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- **3.2** − Now, based on the distance value, sort them in ascending order.
- **3.3** − Next, it will choose the top K rows from the sorted array.
- **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 − End

Random Forest Algorithm:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

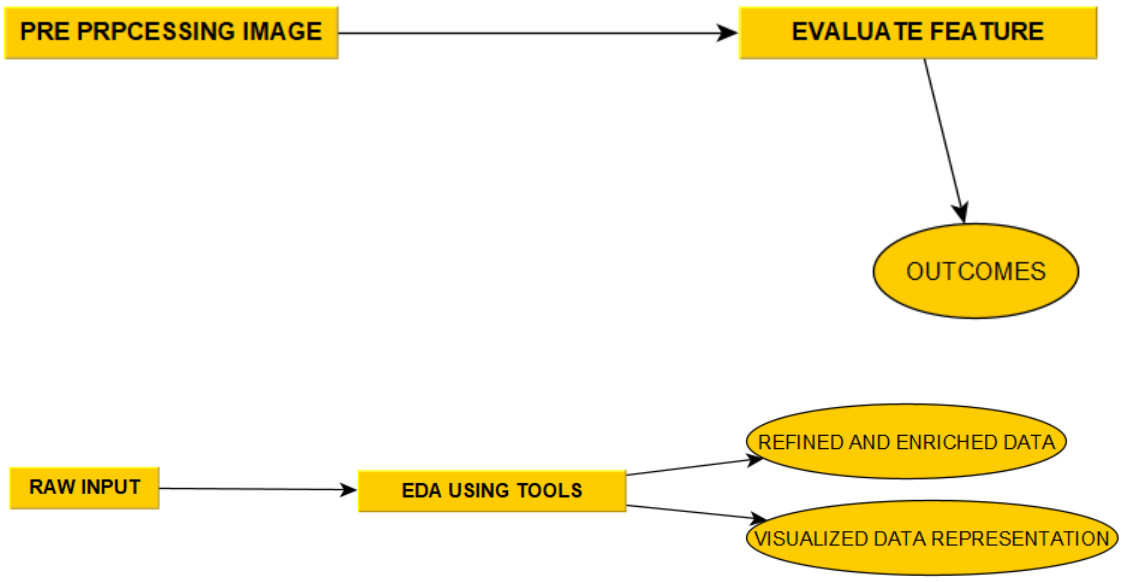
Step-4: Repeat Step 1 & 2.

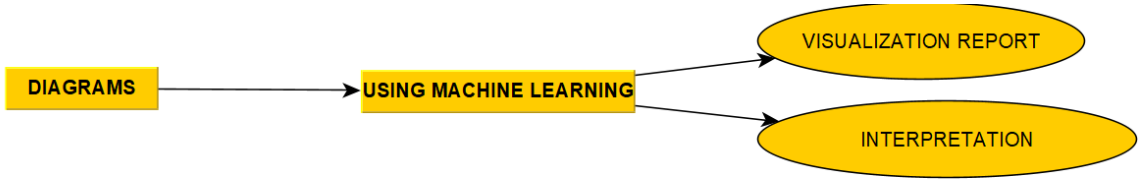
Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes

DATA FLOW DIAGRAM :

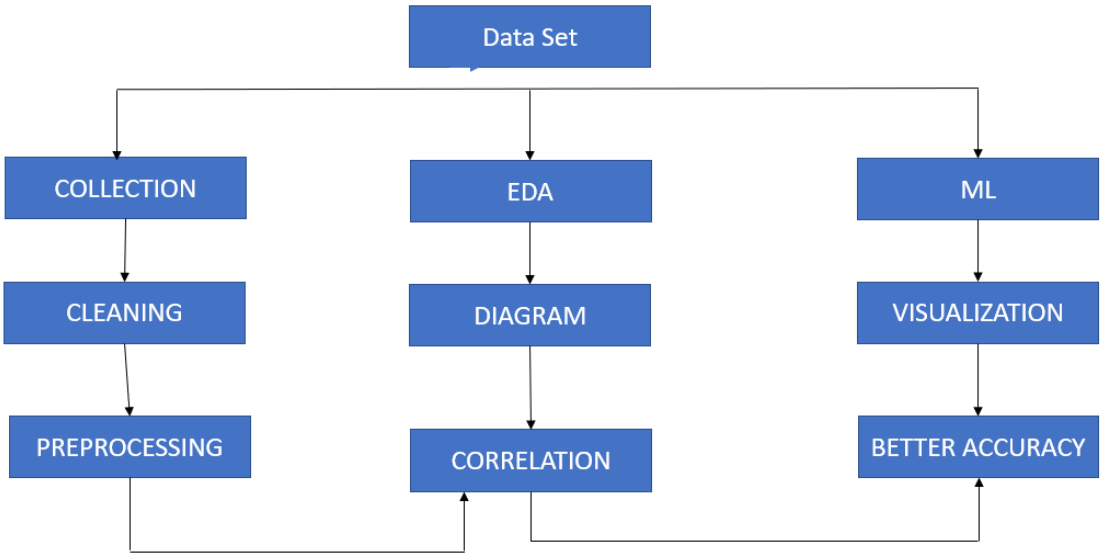
0th Level:

1st Level:





2nd LEVEL:



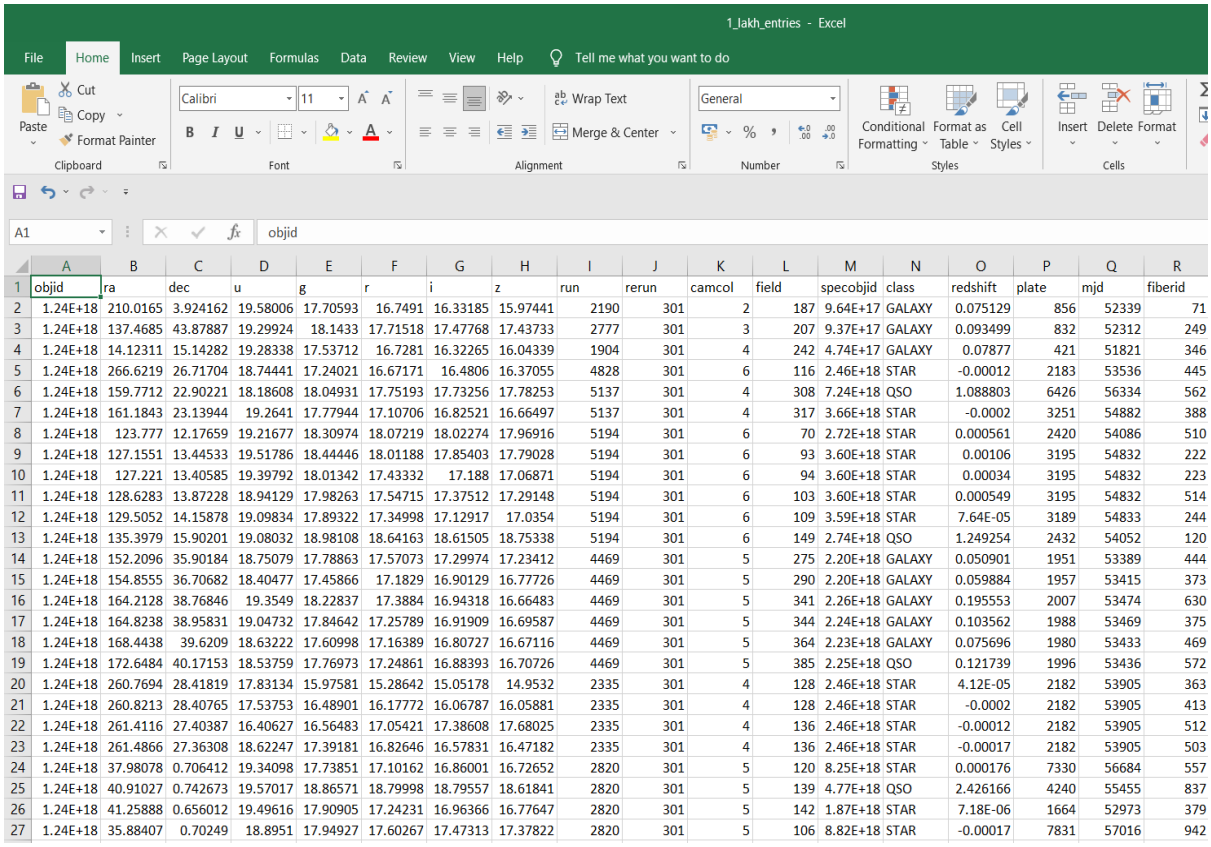
INPUT DATASETS:

DATASET :

The DataSets used this study serve as the raw material on which the work is done. Its from publicly available sources and in fact the creators of the data encourages the scientific as well as the non-scientific community to access, explore and apply their intuitions using Machine learning and statistical tools.

Sloan Digital Sky Survey

1_lakh_entries.csv:



1_lakh_entries - Excel

File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do

Paste Cut Copy Format Painter | Calibri 11 | Wrap Text | General | Conditional Formatting | Format as Table | Cell Styles | Insert Delete Format

Clipboard Font Alignment Number Styles Cells

A1 objid

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	objid	ra	dec	u	g	r	i	z	run	rerun	camcol	field	specobjid	class	redshift	plate	mjd	fiberid
2	1.24E+18	210.0165	3.924162	19.58006	17.70593	16.7491	16.33185	15.97441	2190	301	2	187	9.64E+17	GALAXY	0.075129	856	52339	71
3	1.24E+18	137.4685	43.87887	19.29924	18.1433	17.71518	17.47768	17.43733	2777	301	3	207	9.37E+17	GALAXY	0.093499	832	52312	249
4	1.24E+18	14.12311	15.14282	19.28338	17.53712	16.7281	16.32265	16.04339	1904	301	4	242	4.74E+17	GALAXY	0.07877	421	51821	346
5	1.24E+18	266.6219	26.71704	18.74441	17.24021	16.67171	16.4806	16.37055	4828	301	6	116	2.46E+18	STAR	-0.00012	2183	53536	445
6	1.24E+18	159.7712	22.90221	18.18608	18.04931	17.75193	17.73256	17.78253	5137	301	4	308	7.24E+18	QSO	1.088803	6426	56334	562
7	1.24E+18	161.1843	23.13944	19.2641	17.77944	17.10706	16.82521	16.66497	5137	301	4	317	3.66E+18	STAR	-0.0002	3251	54882	388
8	1.24E+18	123.777	12.17659	19.21677	18.30974	18.07219	18.02274	17.96916	5194	301	6	70	2.72E+18	STAR	0.000561	2420	54086	510
9	1.24E+18	127.1551	13.44533	19.51786	18.44446	18.01188	17.85403	17.79028	5194	301	6	93	3.60E+18	STAR	0.00106	3195	54832	222
10	1.24E+18	127.221	13.40585	19.39792	18.01342	17.43332	17.188	17.06871	5194	301	6	94	3.60E+18	STAR	0.00034	3195	54832	223
11	1.24E+18	128.6283	13.87228	18.94129	17.98263	17.54715	17.37512	17.29148	5194	301	6	103	3.60E+18	STAR	0.000549	3195	54832	514
12	1.24E+18	129.5052	14.15878	19.09834	17.89322	17.34998	17.12917	17.0354	5194	301	6	109	3.59E+18	STAR	7.64E-05	3189	54833	244
13	1.24E+18	135.3979	15.90201	19.08032	18.98108	18.64163	18.61505	18.75338	5194	301	6	149	2.74E+18	QSO	1.249254	2432	54052	120
14	1.24E+18	152.2096	35.90184	18.75079	17.78863	17.57073	17.29974	17.23412	4469	301	5	275	2.20E+18	GALAXY	0.050901	1951	53389	444
15	1.24E+18	154.8555	36.70682	18.40477	17.45866	17.1829	16.90129	16.77726	4469	301	5	290	2.20E+18	GALAXY	0.059884	1957	53415	373
16	1.24E+18	164.2128	38.76846	19.3549	18.22837	17.3884	16.94318	16.66483	4469	301	5	341	2.26E+18	GALAXY	0.195553	2007	53474	630
17	1.24E+18	164.8238	38.95831	19.04732	17.84642	17.25789	16.91909	16.69587	4469	301	5	344	2.24E+18	GALAXY	0.103562	1988	53469	375
18	1.24E+18	168.4438	39.6209	18.63222	17.60998	17.16389	16.80727	16.67116	4469	301	5	364	2.23E+18	GALAXY	0.075696	1980	53433	469
19	1.24E+18	172.6484	40.17153	18.53759	17.76973	17.24861	16.88393	16.70726	4469	301	5	385	2.25E+18	QSO	0.121739	1996	53436	572
20	1.24E+18	260.7694	28.41819	17.83134	15.97581	15.28642	15.05178	14.9532	2335	301	4	128	2.46E+18	STAR	4.12E-05	2182	53905	363
21	1.24E+18	260.8213	28.40765	17.53753	16.48901	16.17772	16.06787	16.05881	2335	301	4	128	2.46E+18	STAR	-0.0002	2182	53905	413
22	1.24E+18	261.4116	27.40387	16.40627	16.56483	17.05421	17.38608	17.68025	2335	301	4	136	2.46E+18	STAR	-0.00012	2182	53905	512
23	1.24E+18	261.4866	27.36308	18.62247	17.39181	16.82646	16.57831	16.47182	2335	301	4	136	2.46E+18	STAR	-0.00017	2182	53905	503
24	1.24E+18	37.98078	0.706412	19.34098	17.73851	17.10162	16.86001	16.72652	2820	301	5	120	8.25E+18	STAR	0.000176	7330	56684	557
25	1.24E+18	40.91027	0.742673	19.57017	18.86571	18.79998	18.79557	18.61841	2820	301	5	139	4.77E+18	QSO	2.426166	4240	55455	837
26	1.24E+18	41.25888	0.656012	19.49616	17.90905	17.24231	16.96366	16.77647	2820	301	5	142	1.87E+18	STAR	7.18E-06	1664	52973	379
27	1.24E+18	35.88407	0.70249	18.8951	17.94927	17.60267	17.47313	17.37822	2820	301	5	106	8.82E+18	STAR	-0.00017	7831	57016	942

CODE AND OUTPUT:

EDA AND CLASSIFICATION IN ASTRONOMY

1. Configuring the Environment

Importing required libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm.notebook import tqdm_notebook
import timeit
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import chi2, mutual_info_classif, SelectKBest, f_classif
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier, GradientBoostingClassifier

import warnings
warnings.filterwarnings('ignore')

#!pip install plotly
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
```

2.DATA PREPARATION

2.1 LOADING THE DATA

2.1 LOADING THE DATA

In [2]:
```
%%time
df = pd.read_csv('1_lakh_entries.csv', skiprows = 0)
df
```

```
Wall time: 218 ms
```

Out[2]:

	objid	ra	dec	u	g	r	i	z	run	rerun	camcol	field	specobjid	class	redshift	p
0	1.237650e+18	210.016478	3.924162	19.58006	17.70593	16.74910	16.33185	15.97441	2190	301	2	187	9.637900e+17	GALAXY	0.075129	
1	1.237660e+18	137.468475	43.878875	19.29924	18.14330	17.71518	17.47768	17.43733	2777	301	3	207	9.368170e+17	GALAXY	0.093499	
2	1.237650e+18	14.123114	15.142815	19.28338	17.53712	16.72810	16.32265	16.04339	1904	301	4	242	4.740990e+17	GALAXY	0.078770	
3	1.237670e+18	266.621865	26.717041	18.74441	17.24021	16.67171	16.48060	16.37055	4828	301	6	116	2.457960e+18	STAR	-0.000124	2
4	1.237670e+18	159.771218	22.902207	18.18608	18.04931	17.75193	17.73256	17.78253	5137	301	4	308	7.235190e+18	QSO	1.088803	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	1.237650e+18	235.711610	0.032597	19.58198	17.75248	16.82558	16.40437	16.04969	752	301	4	616	3.546800e+17	GALAXY	0.082767	
99996	1.237650e+18	237.082722	0.185409	19.38675	17.96262	17.14279	16.68619	16.36536	752	301	4	625	3.851970e+17	GALAXY	0.113793	
99997	1.237650e+18	240.853253	0.092452	19.18663	17.25995	16.27921	15.79979	15.42603	752	301	4	650	3.874260e+17	GALAXY	0.054825	
99998	1.237650e+18	241.534281	0.013727	19.00692	16.97655	15.93292	15.45320	15.01987	752	301	4	654	3.885430e+17	GALAXY	0.057930	
99999	1.237650e+18	241.737495	0.027826	19.43477	17.37000	16.39084	15.90887	15.56186	752	301	4	656	3.885440e+17	GALAXY	0.074140	

100000 rows × 18 columns

2.2 Dataset Overview

The table results from an SQL query which joins two tables:

"PhotoObj" which contains photometric data.
"SpecObj" which contains spectral data.
16 variables (double) and 1 additional variable (char) 'class'.
A class object can be predicted from the other 16 variables.

Variables description:
objid = Object Identifier **ra** = J2000 Right Ascension (r-band)
dec = J2000 Declination (r-band)
u = better of deV/Exp magnitude fit (u-band)
g = better of deV/Exp magnitude fit (g-band)
r = better of deV/Exp magnitude fit (r-band)
i = better of deV/Exp magnitude fit (i-band)
z = better of deV/Exp magnitude fit (z-band)
run = Run Number
rerun = Rerun Number
camcol = Camera column
field = Field number
specobjid = Object Identifier
class = object class (galaxy, star or quasar object)
redshift = Final Redshift
plate = plate number
mjd = MJD(Modified Julian Date) of observation
fiberid = fiberID

2.3 Basic information

```
In [3]: df.shape
Out[3]: (100000, 18)
```

```
In [4]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 18 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   objid      100000 non-null  float64
 1   ra         100000 non-null  float64
 2   dec        100000 non-null  float64
 3   u          100000 non-null  float64
 4   g          100000 non-null  float64
 5   r          100000 non-null  float64
 6   i          100000 non-null  float64
 7   z          100000 non-null  float64
 8   run        100000 non-null  int64
 9   rerun      100000 non-null  int64
 10  camcol     100000 non-null  int64
 11  field      100000 non-null  int64
 12  specobjid  100000 non-null  float64
 13  class      100000 non-null  object
 14  redshift   100000 non-null  float64
 15  plate      100000 non-null  int64
 16  mjd        100000 non-null  int64
 17  fiberid    100000 non-null  int64
dtypes: float64(10), int64(7), object(1)
memory usage: 13.7+ MB
```

Dropping Features that are not significant

'objid' and 'specobjid' are only identifiers for getting to the rows back when they were put away in the original databank. Along these lines we won't need them for classification as they are not identified with the result.

Significantly more: The features 'run', 'rerun', 'camcol' and 'field' are values which describe portions of the camera right when mentioning the objective fact, for example 'run' speaks to the comparing check which caught the oject.

We'll drop these features.

In [15]:
```
df.drop(['run', 'rerun', 'camcol', 'field', 'objid', 'specobjid', 'fiberid'], axis = 1, inplace= True)
df.head(5)
```

Out[15]:

	ra	dec	u	g	r	i	z	class	redshift	plate	mjd
0	210.016478	3.924162	19.58006	17.70593	16.74910	16.33185	15.97441	GALAXY	0.075129	856	52339
1	137.468475	43.878875	19.29924	18.14330	17.71518	17.47768	17.43733	GALAXY	0.093499	832	52312
2	14.123114	15.142815	19.28338	17.53712	16.72810	16.32265	16.04339	GALAXY	0.078770	421	51821
3	266.621865	26.717041	18.74441	17.24021	16.67171	16.48060	16.37055	STAR	-0.000124	2183	53536
4	159.771218	22.902207	18.18608	18.04931	17.75193	17.73256	17.78253	QSO	1.088803	6426	56334

3. High Level Statistics

3.1 Describing the Data

In [5]:
```
df.describe()
```

Out[5]:

	ra	dec	u	g	r	i	z	redshift	plate	mjd
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	177.517993	25.048529	18.637645	17.407114	16.881728	16.625299	16.466702	0.171041	2597.153180	53918.746860
std	78.062282	20.544956	0.832263	0.985319	1.132708	1.207941	1.280648	0.439244	2225.962745	1553.442988
min	0.011306	-19.495456	10.611810	9.668339	9.005167	8.848403	8.947795	-0.004268	266.000000	51608.000000
25%	136.418824	6.732728	18.211495	16.852083	16.196345	15.864370	15.619682	0.000001	1186.000000	52734.000000
50%	180.391592	23.962150	18.874240	17.515725	16.890090	16.598420	16.427275	0.045948	2087.000000	53726.000000
75%	224.650548	40.344539	19.273580	18.056393	17.585740	17.345235	17.235225	0.095516	2911.000000	54589.000000
max	359.999615	84.490494	19.599990	19.996050	31.990100	32.141470	29.383740	7.011245	12547.000000	58932.000000

From the above table we can tell that are no missing values at all. This means: no imputing.

In [6]:
```
df.shape
```

Out[6]:
```
(100000, 11)
```

Now we have only 10 features and 100000 data points since 6 features which were not significant to the class label were dropped.

3.2 Find the duplicates

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
0
```

Observations

There are no duplicates

5.3 Correlation among all the variables

3.4 Unique Values

In [9]:
```
# We can find the number of unique values in the particular column using unique() function in python.
df['class'].unique()
```

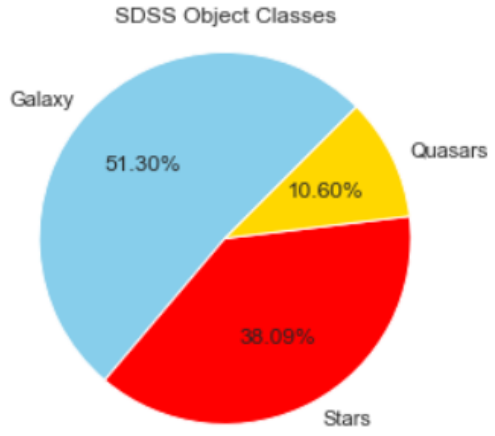
Out[9]:
```
array(['GALAXY', 'STAR', 'QSO'], dtype=object)
```

In [10]:
```
df['class'].value_counts()
```

Out[10]:
```
GALAXY    51304
STAR      38091
QSO       10605
Name: class, dtype: int64
```

In [12]:

```
def pieChart(df):
    '''Plot a pie chart for label count.'''
    label_counts = df['class'].value_counts()
    colors = ['skyblue', 'red', 'gold']
    fig1, ax1 = plt.subplots()
    ax1.pie(label_counts, labels=['Galaxy', 'Stars', 'Quasars'],
            autopct='%1.2f%%', startangle=45, colors=colors)
    ax1.axis('equal')
    plt.title('SDSS Object Classes')
    plt.show()
pieChart(df)
```



3.4 Observations

1. The column 'class' is the target variable and its a categorical variable. The unique values of the class label are 'GALAXY', 'QSO' and 'STAR'
2. The dataset is quite imbalanced with number of galaxies dominating.

3.5 Find null values

In [13]:

```
df.isnull().sum()
```

Out[13]:

```
ra          0
dec         0
u           0
g           0
r           0
i           0
z           0
class       0
redshift    0
plate       0
mjd         0
dtype: int64
```

There are no null values.

4. Univariate Analysis

Distribution plots are used to visually assess how the data points are distributed with respect to its frequency. Usually the data points are grouped into bins and the height of the bars representing each group increases with increase in the number of data points lie within that group. (histogram)

Probability Density Function (PDF) is the probability that the variable takes a value x. (smoothed version of the histogram)

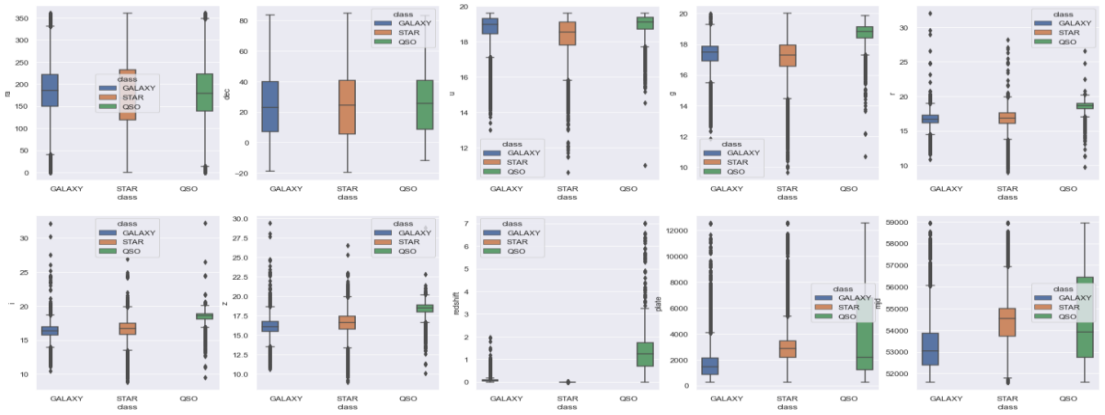
4.1 Boxplot

In [14]:
```
df.columns
```

Out[14]:
```
Index(['ra', 'dec', 'u', 'g', 'r', 'i', 'z', 'class', 'redshift', 'plate',
       'mjd'],
      dtype='object')
```

In [15]:
```
fig, ((ax1, ax2, ax3, ax4, ax5), (ax6, ax7, ax8, ax9, ax10)) = plt.subplots(nrows = 2, ncols = 5,
                                                                             figsize = (25,12))


sns.boxplot(ax = ax1, x = 'class', y = 'ra', hue = 'class', data = df)
sns.boxplot(ax = ax2, x = 'class', y = 'dec', hue = 'class', data = df)
sns.boxplot(ax = ax3, x = 'class', y = 'u', hue = 'class', data = df)
sns.boxplot(ax = ax4, x = 'class', y = 'g', hue = 'class', data = df)
sns.boxplot(ax = ax5, x = 'class', y = 'r', hue = 'class', data = df)
sns.boxplot(ax = ax6, x = 'class', y = 'i', hue = 'class', data = df)
sns.boxplot(ax = ax7, x = 'class', y = 'z', hue = 'class', data = df)
sns.boxplot(ax = ax8, x = 'class', y = 'redshift', hue = 'class', data = df)
sns.boxplot(ax = ax9, x = 'class', y = 'plate', hue = 'class', data = df)
sns.boxplot(ax = ax10, x = 'class', y = 'mjd', hue = 'class', data = df)
```

Out[15]: <AxesSubplot:xlabel='class', ylabel='mjd'>

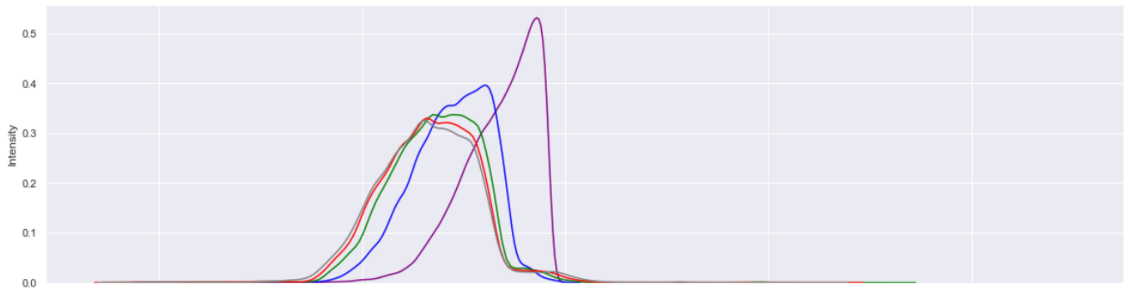


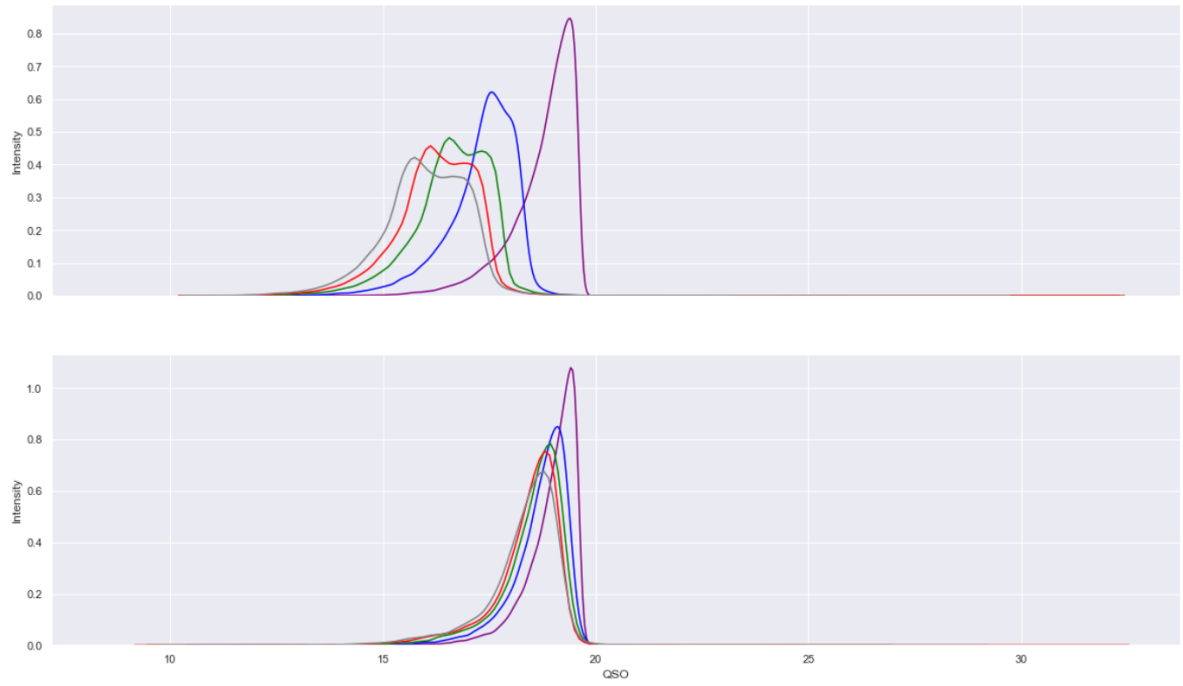
Observations

As we can see from the boxplot there are a large number of outliers.

4.2 Frequency polygon

In [16]:
```
f, axes = plt.subplots(3, 1, figsize=(20, 18), sharex=True)
c = ['STAR', 'GALAXY', 'QSO']

for ax_id in range(3):
    sns.distplot(df.loc[df['class']==c[ax_id],'u'], hist=False, color='purple', ax=axes[ax_id], label='u')
    sns.distplot(df.loc[df['class']==c[ax_id],'g'], hist=False, color='blue', ax=axes[ax_id], label='g')
    sns.distplot(df.loc[df['class']==c[ax_id],'r'], hist=False, color='green', ax=axes[ax_id], label='r')
    sns.distplot(df.loc[df['class']==c[ax_id],'i'], hist=False, color='red', ax=axes[ax_id], label='i')
    sns.distplot(df.loc[df['class']==c[ax_id],'z'], hist=False, color='grey', ax=axes[ax_id], label='z')
    axes[ax_id].set(xlabel=c[ax_id], ylabel='Intensity')
```

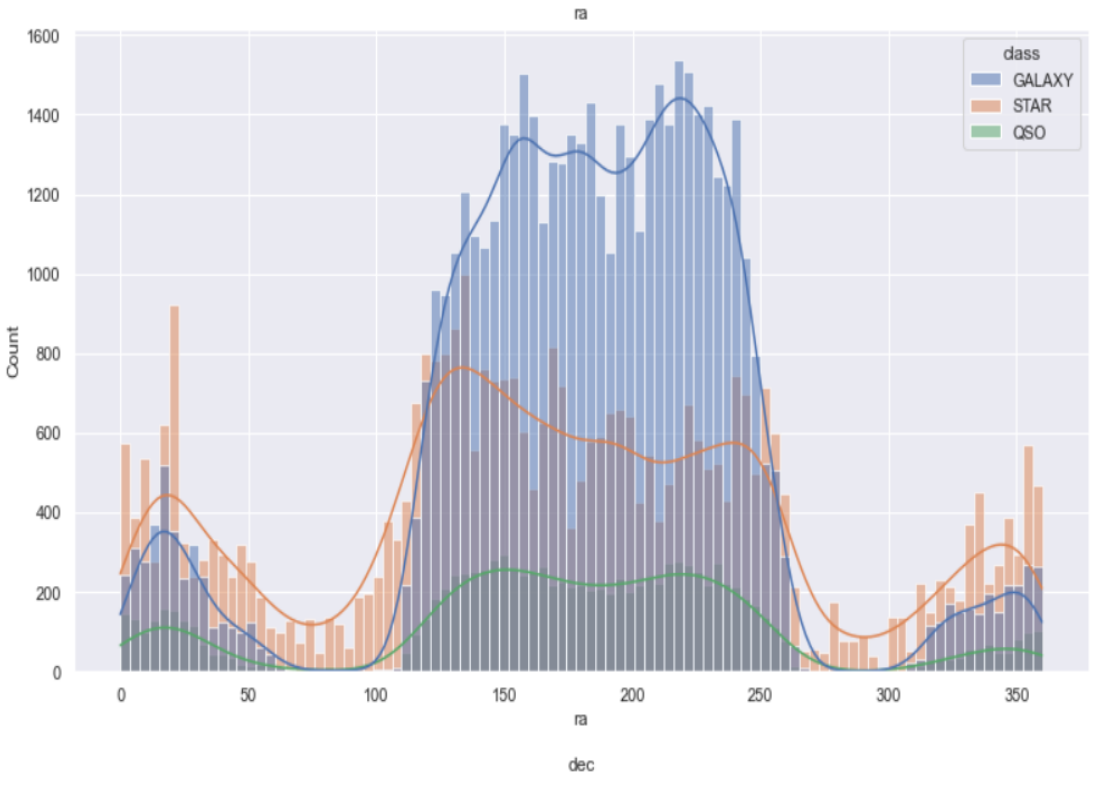




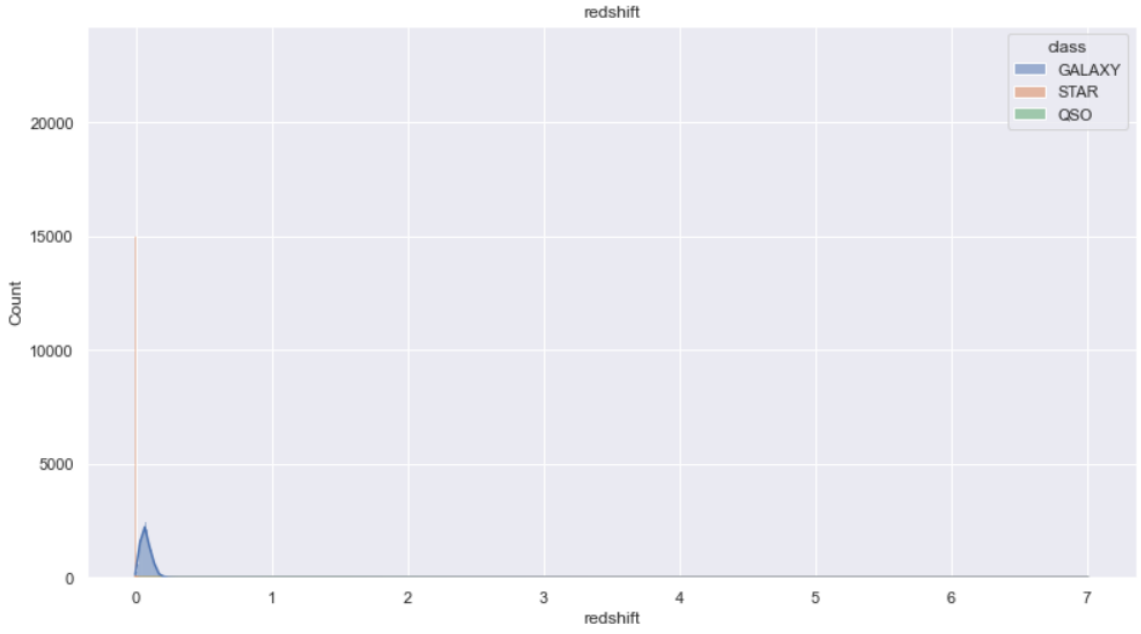
4.3 Histplot

In [17]:

```
import seaborn as sb
for i in ['ra', 'dec', 'redshift', 'plate', 'mjd']:
    plt.figure(figsize=(13,7))
    sb.histplot(data=df, x=i, kde=True, hue="class")
    plt.title(i)
    plt.show()
```




dec
class
GALAXY
STAR
QSO
Count
dec


redshift
class
GALAXY
STAR
QSO
Count
redshift

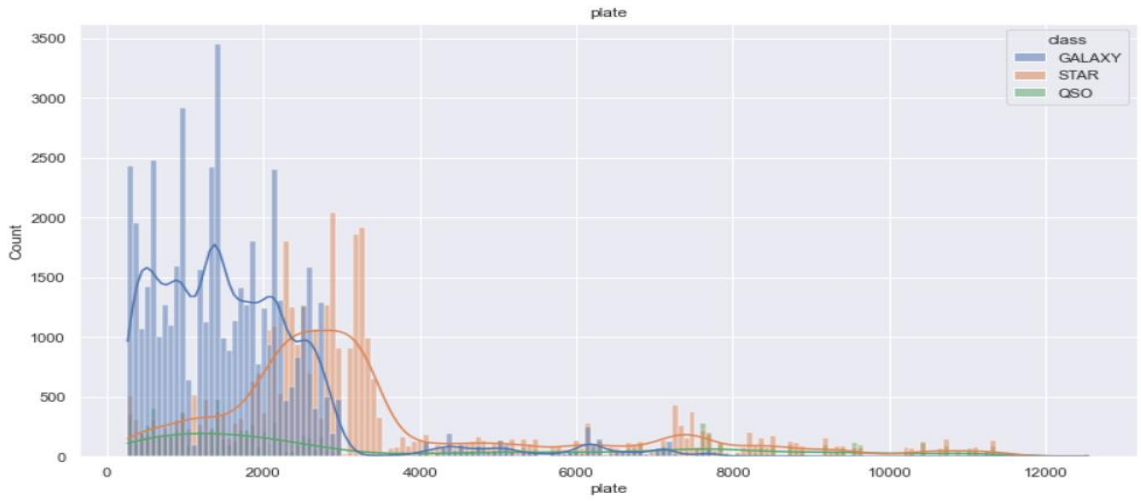
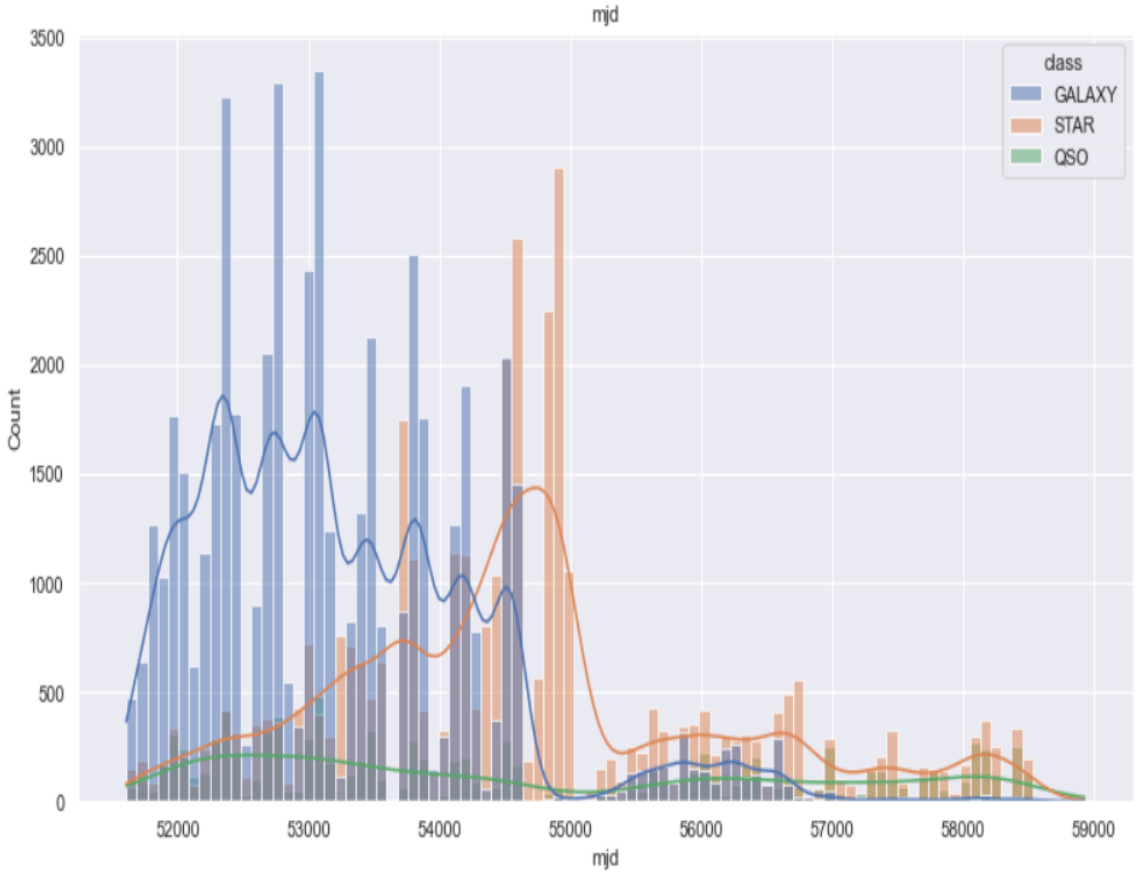


plate
class
GALAXY
STAR
QSO
Count
plate



4.4 Probability Density Function (PDF)

In [18]:
```
df.columns
```

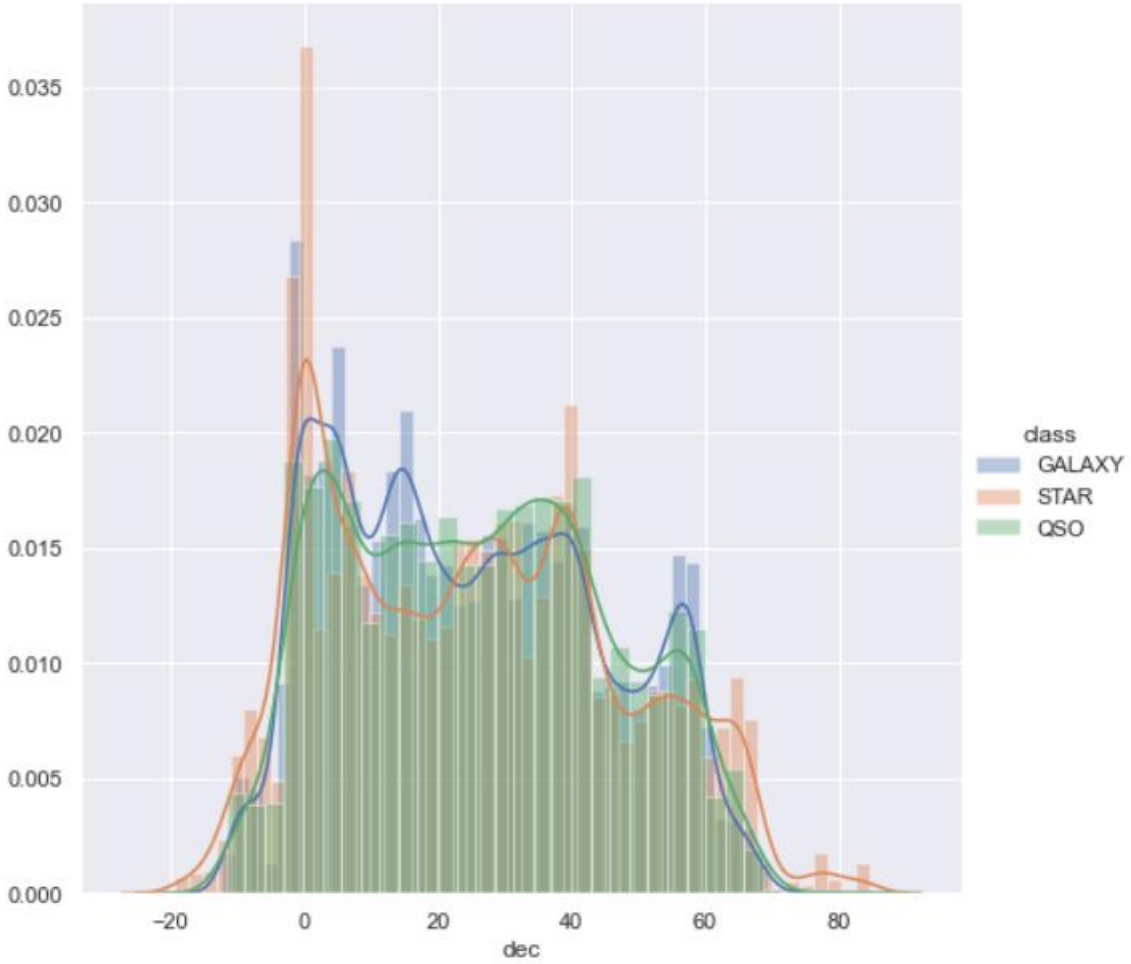
Out[18]:
```
Index(['ra', 'dec', 'u', 'g', 'r', 'i', 'z', 'class', 'redshift', 'plate',
       'mjd'],
      dtype='object')
```

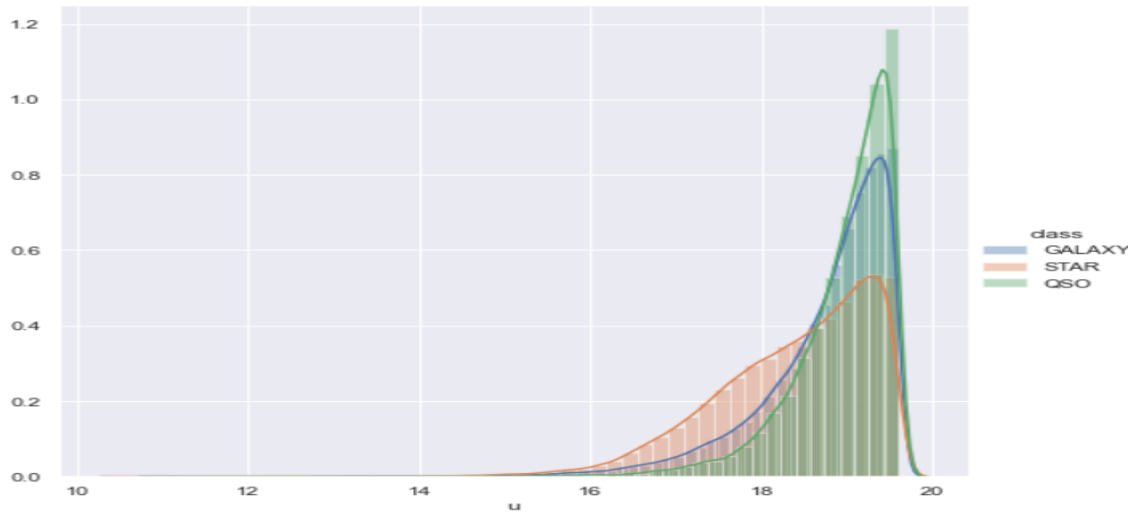
In [19]:
```
col = list(df.columns)
col.pop(7)
col
```

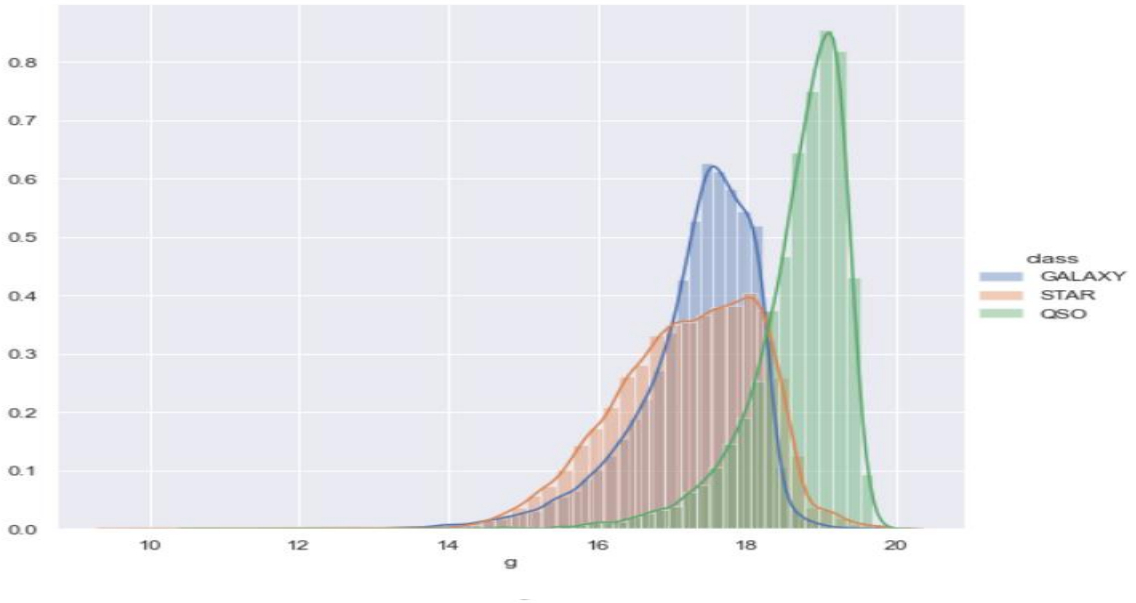
Out[19]:
```
['ra', 'dec', 'u', 'g', 'r', 'i', 'z', 'redshift', 'plate', 'mjd']
```

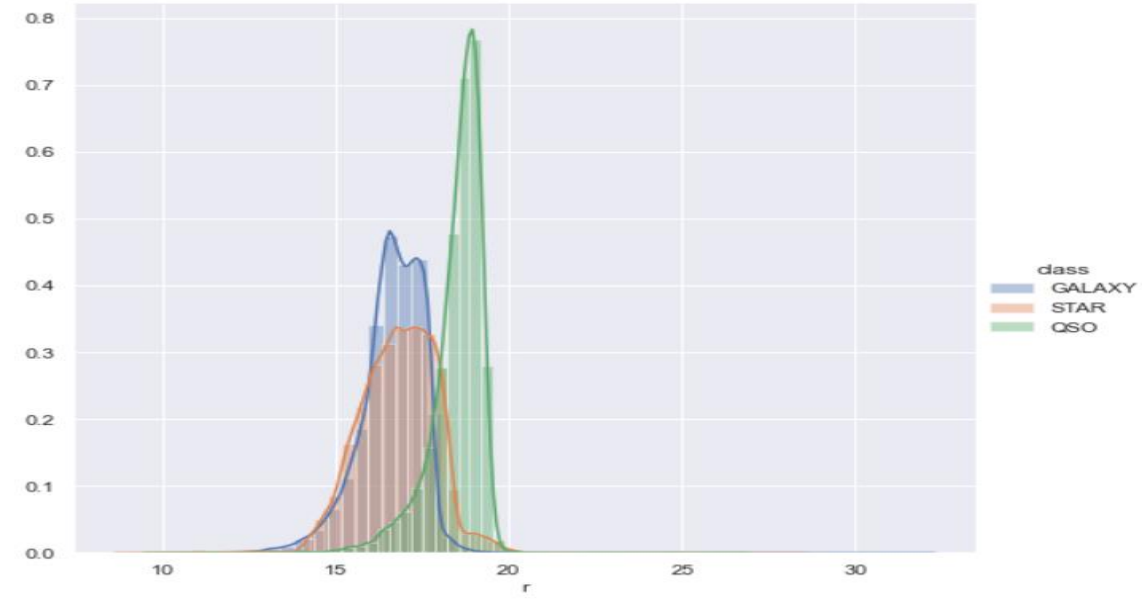
In [20]:
```
%%time
for idx, feature in enumerate(col):
    fg = sns.FacetGrid(df, hue='class', height=7)
    fg.map(sns.distplot, feature).add_legend()
    plt.show()
```

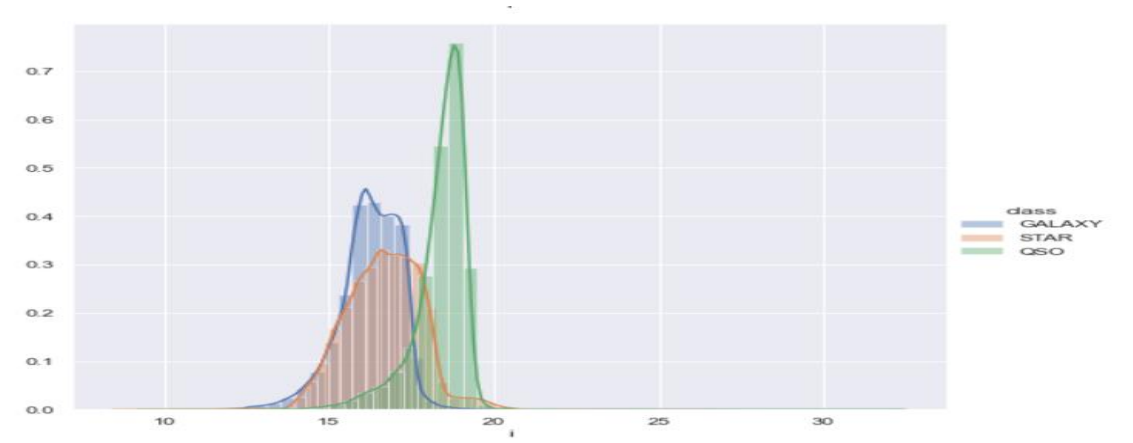

0.008
0.007
0.006
0.005
0.004
0.003
0.002
0.001
0.000
0
100
200
300
400
ra
class
GALAXY
STAR
QSO

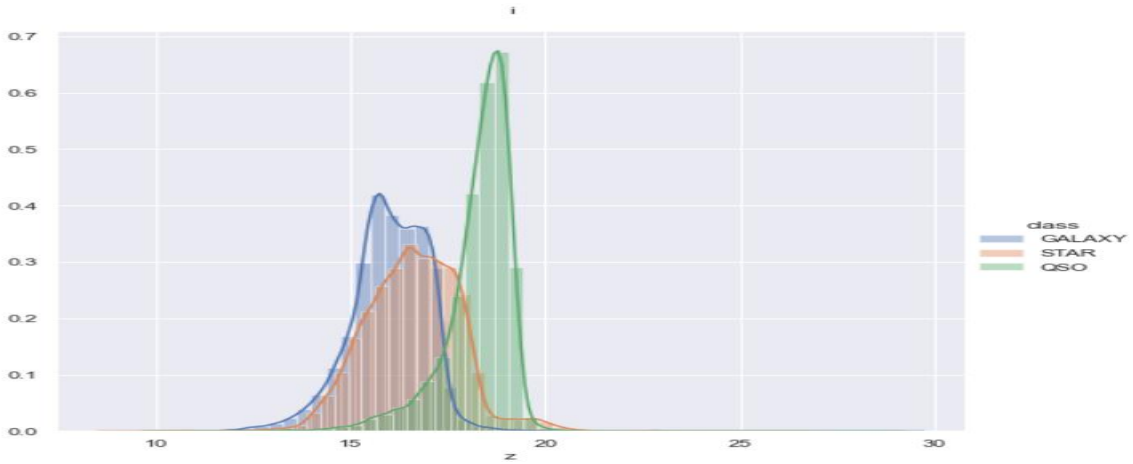

0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
−20
0
20
40
60
80
dec
class
GALAXY
STAR
QSO

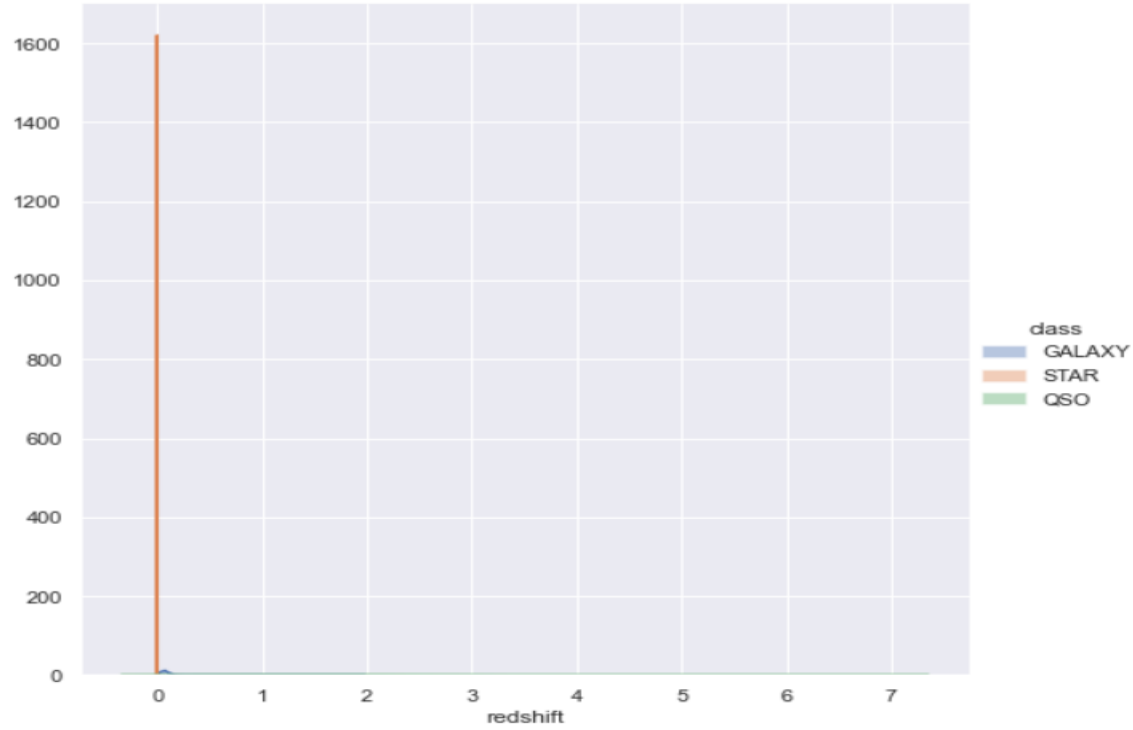

1.2
1.0
0.8
0.6
0.4
0.2
0.0
10
12
14
16
18
20
u
dass
GALAXY
STAR
QSO

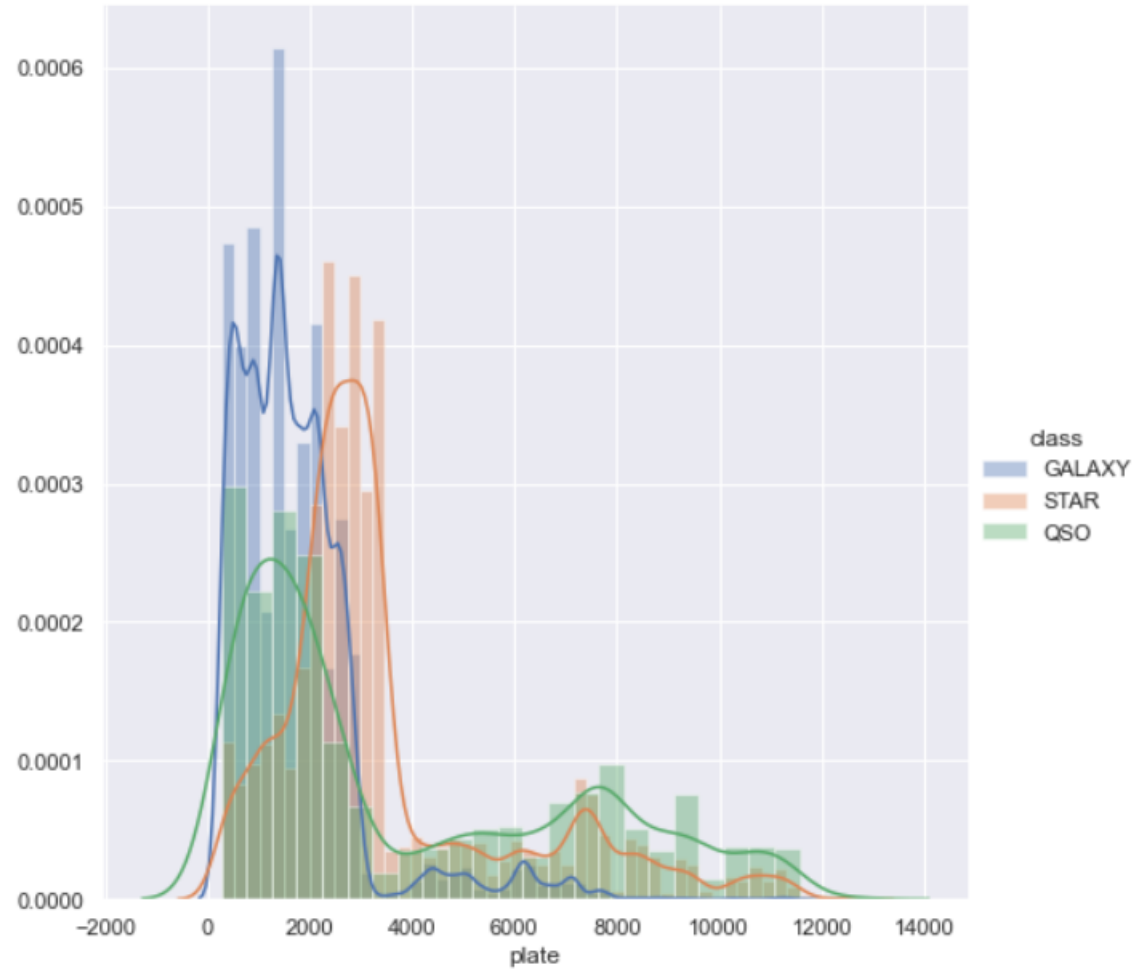

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
10
12
14
16
18
20
g
dass
GALAXY
STAR
QSO

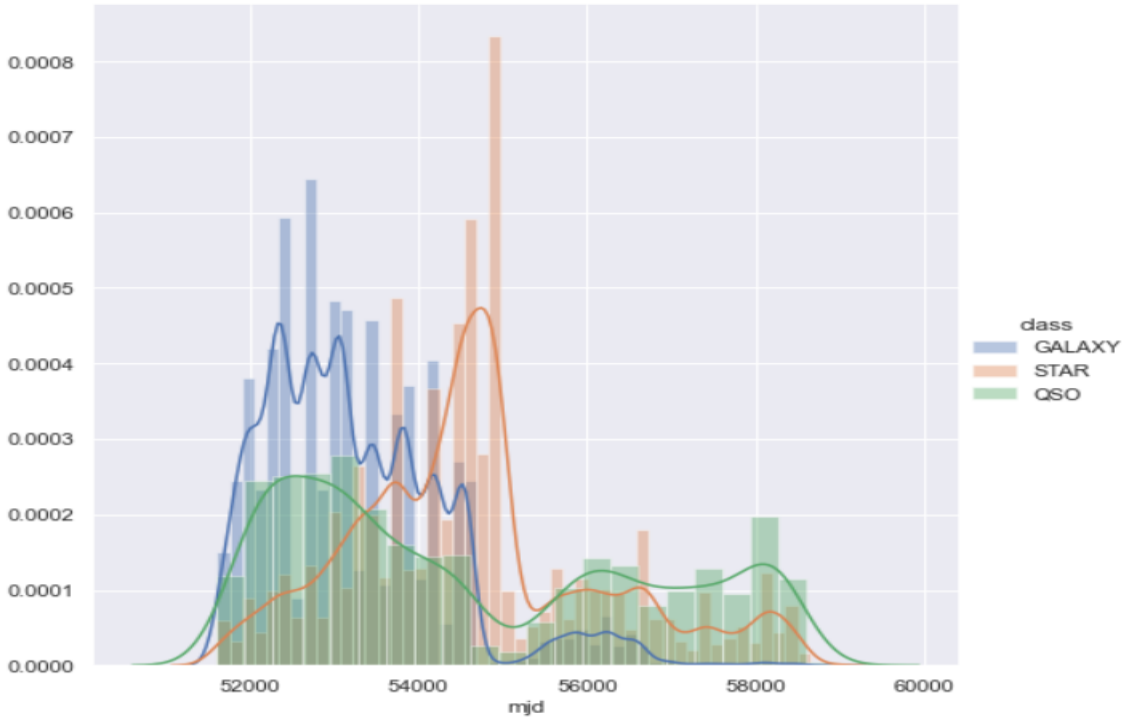

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
10
15
20
25
30
r
dass
GALAXY
STAR
QSO


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
10
15
20
25
30
i
dass
GALAXY
STAR
QSO


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
10
15
20
25
30
z
dass
GALAXY
STAR
QSO


1600
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5
6
7
redshift
dass
GALAXY
STAR
QSO

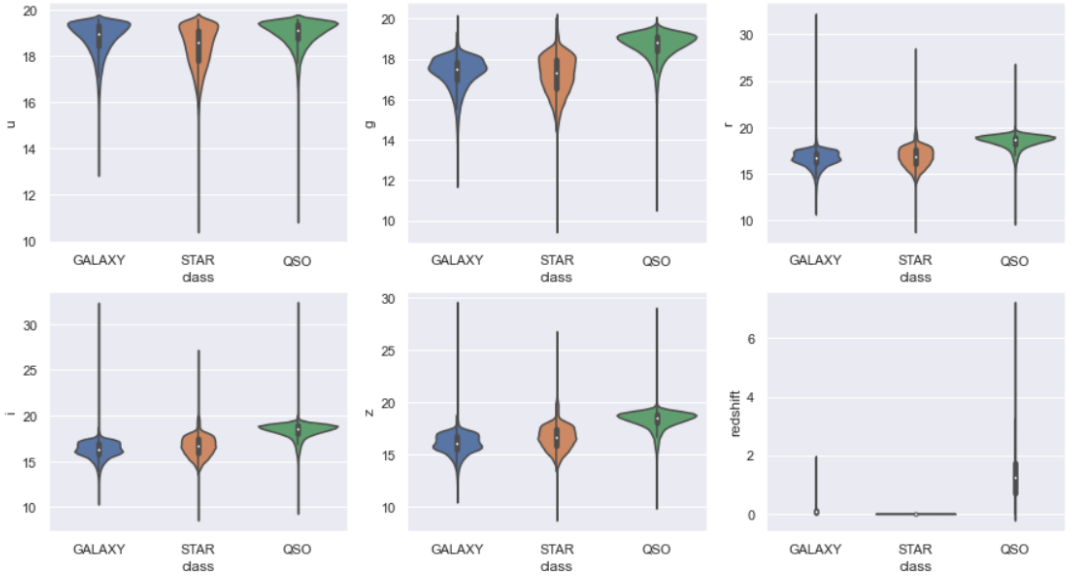





```
Wall time: 34.7 s
```

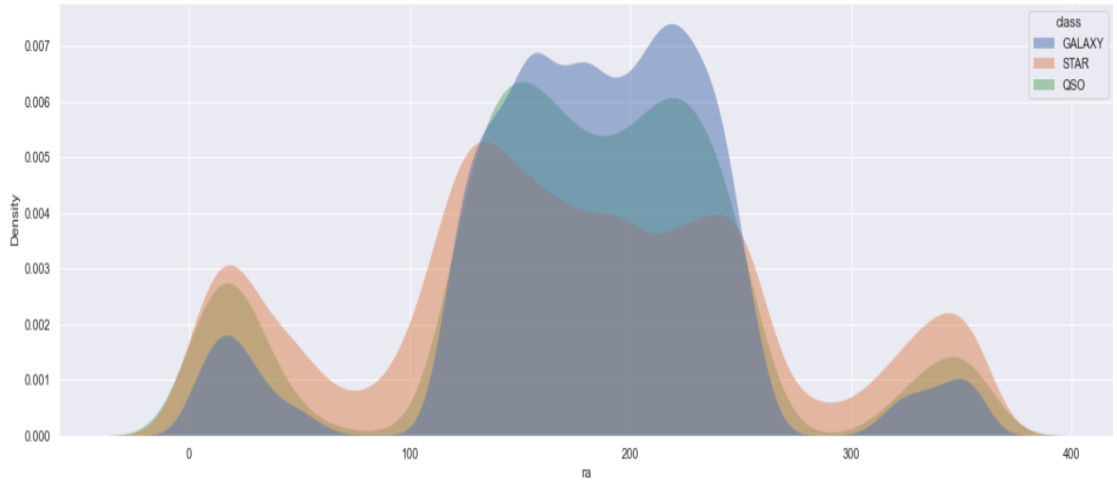
4.5 Violinplot

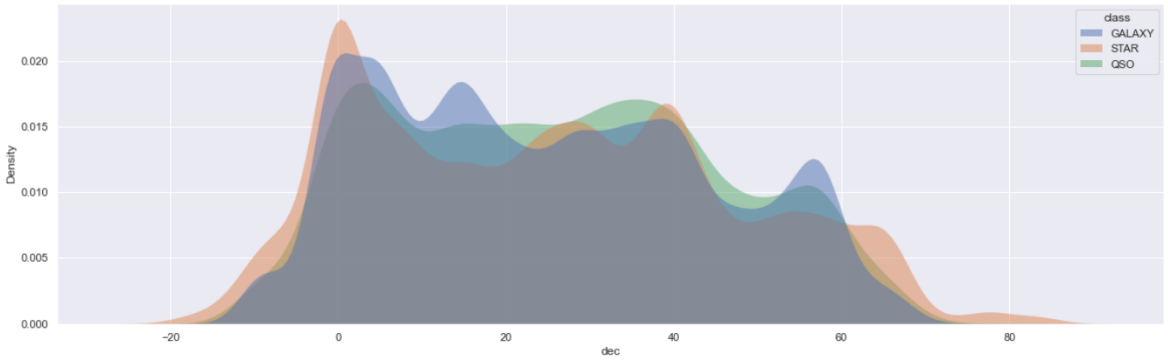
In [21]:
```
fig, axs = plt.subplots(2, 3, figsize=(15, 8))

sns.violinplot(x="class", y="u", data=df, ax=axs[0, 0])
sns.violinplot(x="class", y="g", data=df, ax=axs[0, 1])
sns.violinplot(x="class", y="r", data=df, ax=axs[0, 2])
sns.violinplot(x="class", y="i", data=df, ax=axs[1, 0])
sns.violinplot(x="class", y="z", data=df, ax=axs[1, 1])
sns.violinplot(x="class", y="redshift", data=df, ax=axs[1, 2])
plt.show()
```

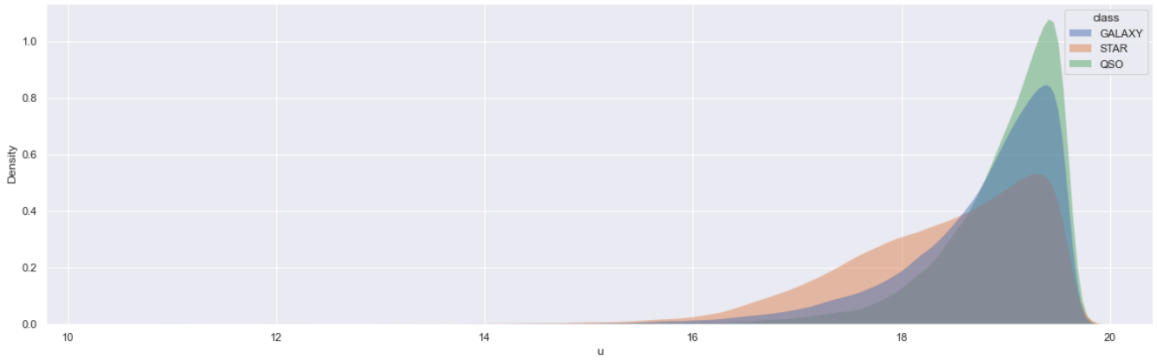


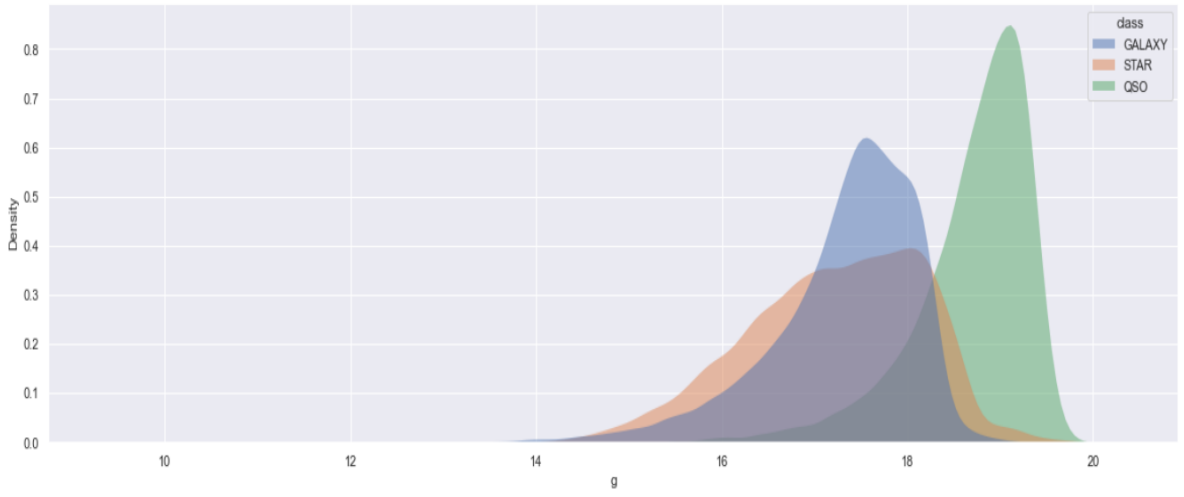
4.6 KDE Plot

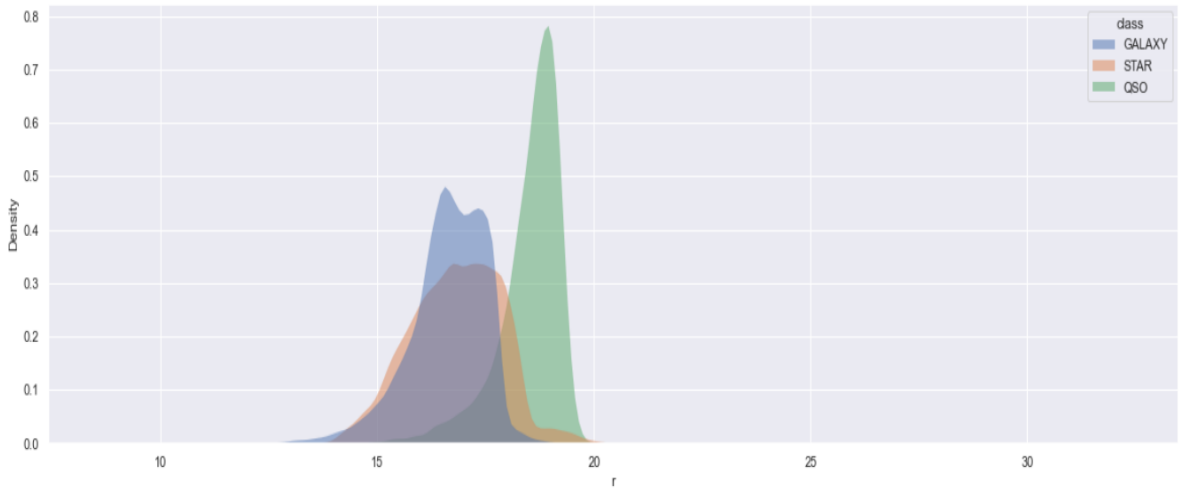
In [22]:
```
fig, axes = plt.subplots(nrows=10, ncols=1,figsize=(20, 70))
ax = sns.kdeplot(data=df, x='ra', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[0])
ax = sns.kdeplot(data=df, x='dec', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[1])
ax = sns.kdeplot(data=df, x='u', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[2])
ax = sns.kdeplot(data=df, x='g', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[3])
ax = sns.kdeplot(data=df, x='r', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[4])
ax = sns.kdeplot(data=df, x='i', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[5])
ax = sns.kdeplot(data=df, x='z', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[6])
ax = sns.kdeplot(data=df, x='redshift', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[7])
ax = sns.kdeplot(data=df, x='plate', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[8])
ax = sns.kdeplot(data=df, x='mjd', hue="class", fill=True, common_norm=False, alpha=.5, linewidth=0,ax = axes[9])
```

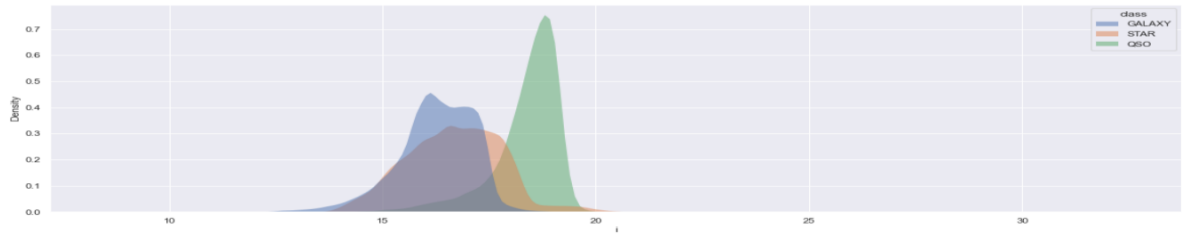


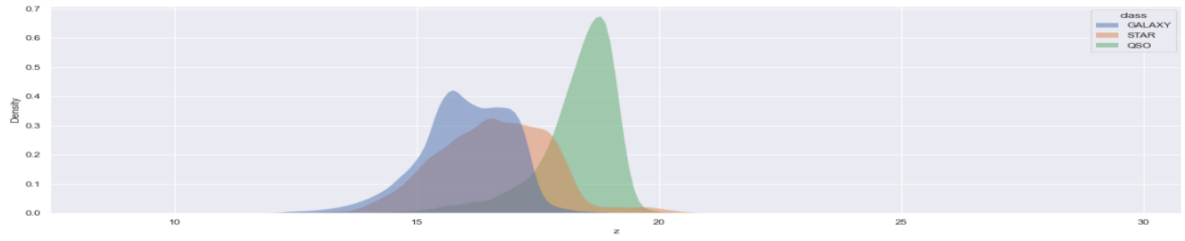

class
GALAXY
STAR
QSO
Density
0.020
0.015
0.010
0.005
0.000
−20
0
20
40
60
80
dec

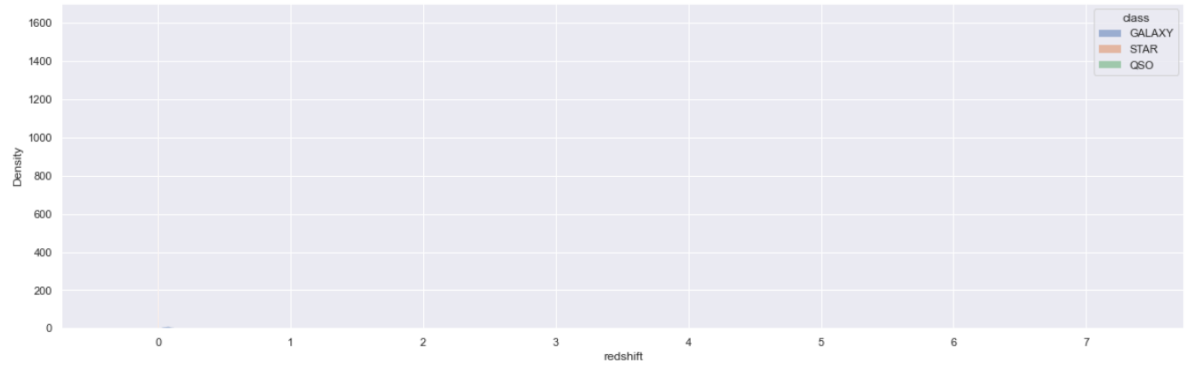

class
GALAXY
STAR
QSO
Density
1.0
0.8
0.6
0.4
0.2
0.0
10
12
14
16
18
20
u

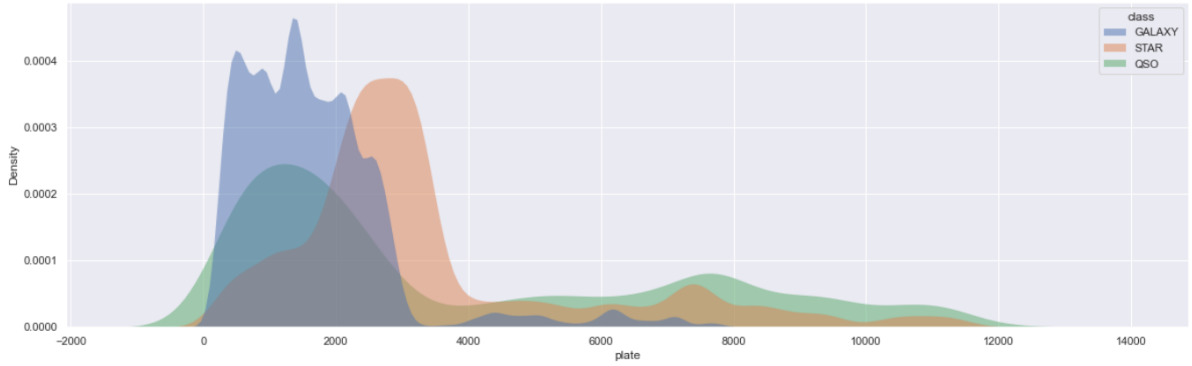

class
GALAXY
STAR
QSO
Density
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
10
12
14
16
18
20
g

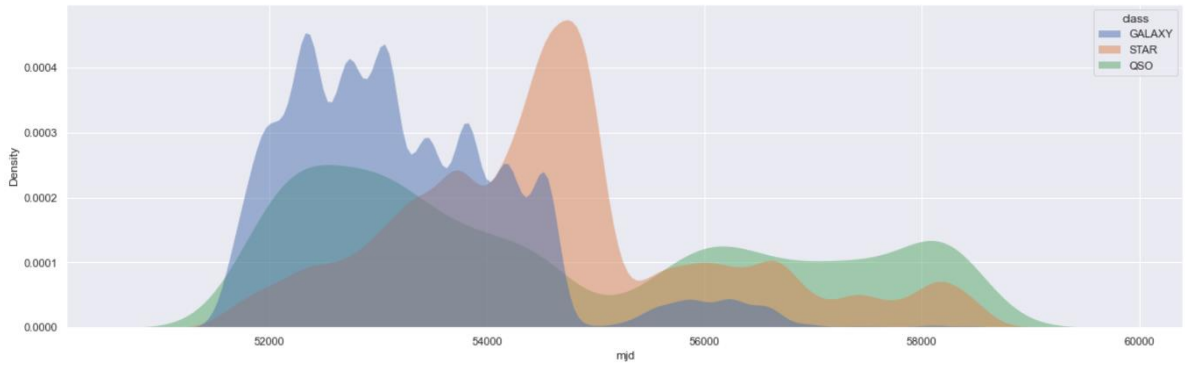

class
GALAXY
STAR
QSO
Density
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
10
15
20
25
30
r


class
GALAXY
STAR
QSO
Density
i

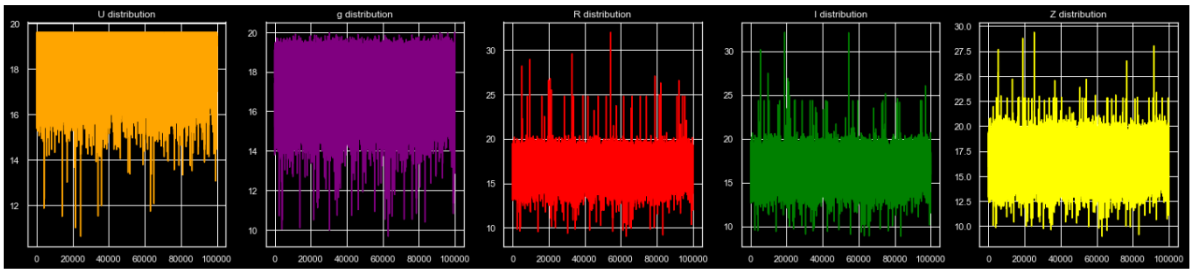

class
GALAXY
STAR
QSO
Density
z


class
GALAXY
STAR
QSO
Density
redshift


class
GALAXY
STAR
QSO
Density
plate


class
GALAXY
STAR
QSO
Density
mjd

```
fig,ax = plt.subplots(1,5,figsize = (25,5))
ax[0].plot(df['u'],color = 'orange');
ax[0].set_title('U distribution');
ax[1].plot(df['g'],color = 'purple');
ax[1].set_title('g distribution');
ax[2].plot(df['r'],color = 'red');
ax[2].set_title('R distribution');
ax[3].plot(df['i'],color = 'green');
ax[3].set_title('I distribution');
ax[4].plot(df['z'],color = 'yellow');
ax[4].set_title('Z distribution');
```



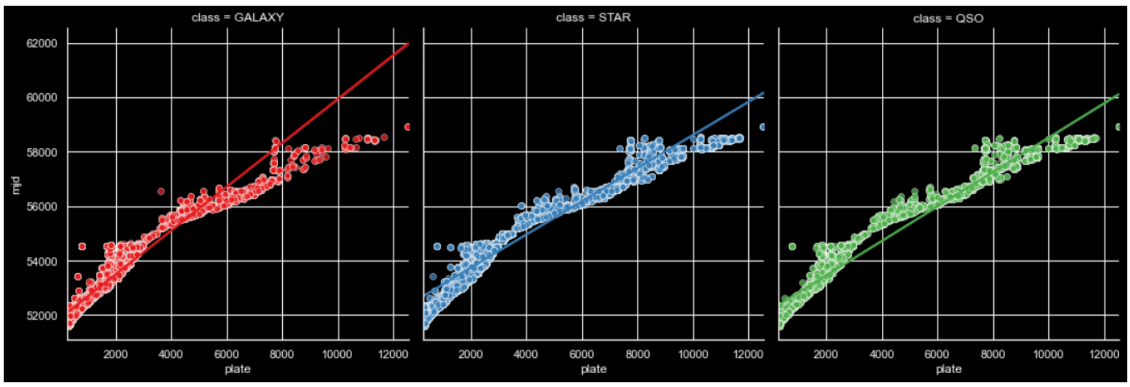
5 . Multivariate Analysis

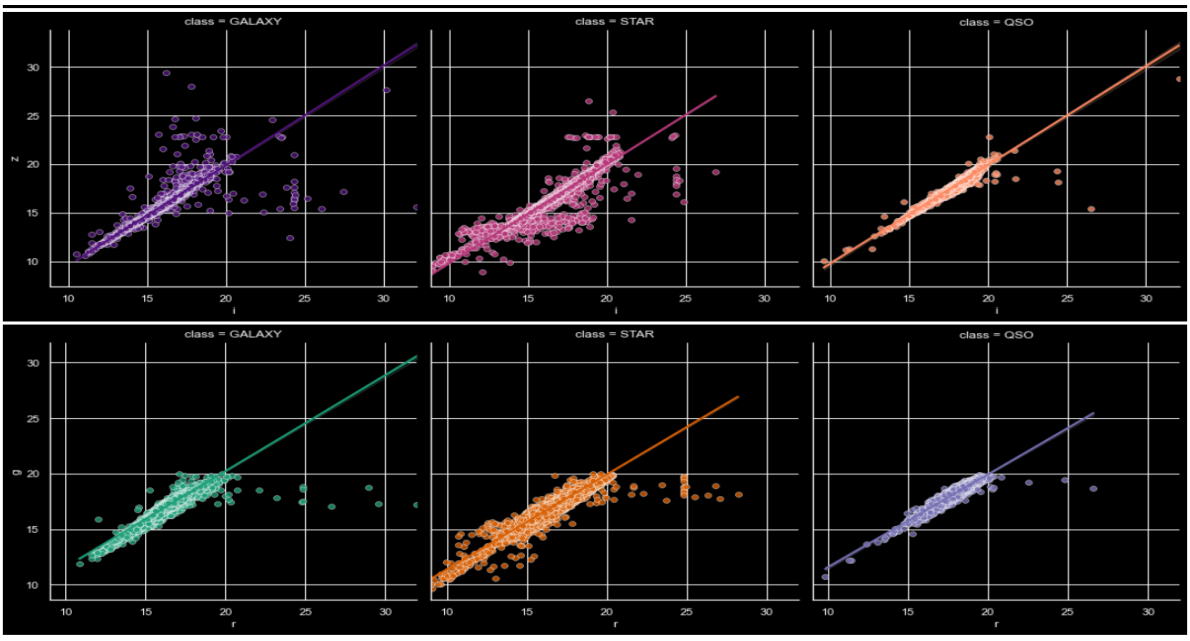
5.1 Lineplot

In [25]:
```
%%time
sns.lmplot(x = 'plate', y='mjd', data = df, hue='class', col = 'class', palette='Set1', scatter_kws= {'edgecolor':'white', 'alpha':0.8, 'linewidths':
sns.lmplot(x = 'i', y='z', data = df, hue='class', col = 'class', palette='magma', scatter_kws= {'edgecolor':'white', 'alpha':0.8, 'linewidths': 0.5})
sns.lmplot(x = 'r', y='g', data = df, hue='class', col = 'class', palette='Dark2', scatter_kws= {'edgecolor':'white', 'alpha':0.8, 'linewidths': 0.5})
```

```
Wall time: 48.4 s
```

Out[25]:
```
<seaborn.axisgrid.FacetGrid at 0x288001ea370>
```

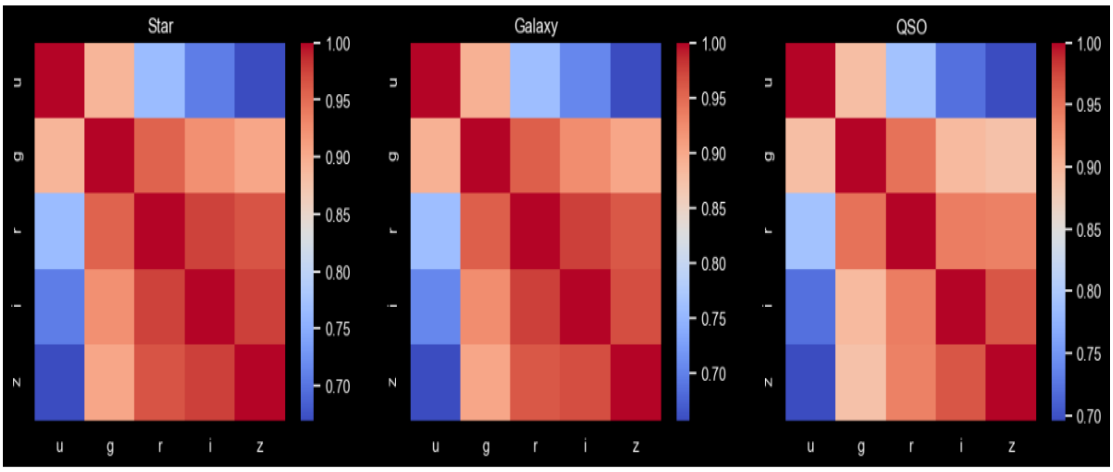




5.3 Correlation between different variables

In [26]:

```
fig, axes = plt.subplots(nrows=1, ncols=3,figsize=(16, 4))
fig.set_dpi(100)
ax = sns.heatmap(df[df['class']=='STAR'][['u', 'g', 'r', 'i', 'z']].corr(), ax = axes[0], cmap='coolwarm')
ax.set_title('Star')
ax = sns.heatmap(df[df['class']=='GALAXY'][['u', 'g', 'r', 'i', 'z']].corr(), ax = axes[1], cmap='coolwarm')
ax.set_title('Galaxy')
ax = sns.heatmap(df[df['class']=='QSO'][['u', 'g', 'r', 'i', 'z']].corr(), ax = axes[2], cmap='coolwarm')
ax = ax.set_title('QSO')
```



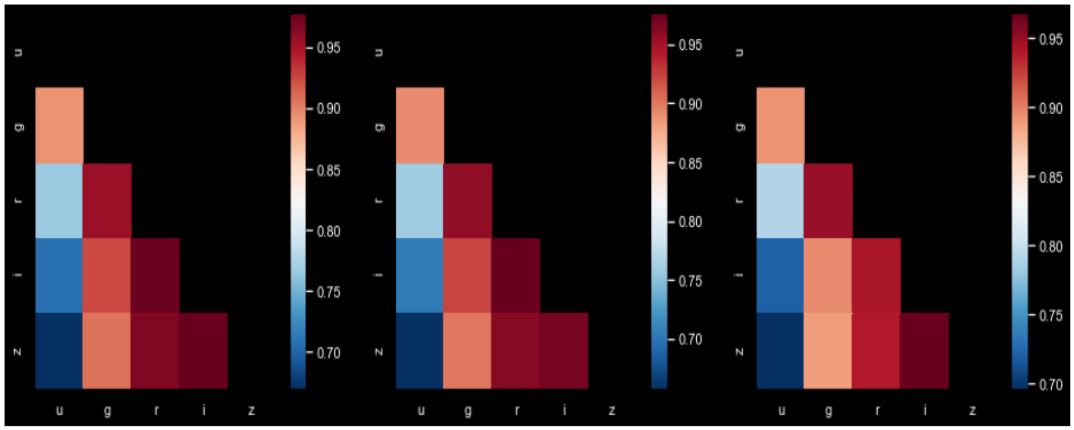
5.4 Subplot

5.4 Subplot

In [27]:

```
f, axes = plt.subplots(1, 3, figsize=(16, 5))

star_corr = df.loc[df['class']=='STAR', ['u','g','r','i','z']].corr()
galaxy_corr =df.loc[df['class']=='GALAXY', ['u','g','r','i','z']].corr()
qso_corr = df.loc[df['class']=='QSO', ['u','g','r','i','z']].corr()

msk = np.zeros_like(star_corr)
msk[np.triu_indices_from(msk)] = True

sns.heatmap(star_corr, cmap='RdBu_r', mask=msk, ax=axes[0])
sns.heatmap(galaxy_corr, cmap='RdBu_r', mask=msk, ax=axes[1])
sns.heatmap(qso_corr, cmap='RdBu_r', mask=msk, ax=axes[2])
```

Out[27]: <AxesSubplot:>



5.4 3-D Scatterplot

In [40]:

```
lbl = LabelEncoder()
cls_enc = lbl.fit_transform(df['class'])

g = go.Scatter3d(
    x=df['ra'], y=df['dec'], z=df['redshift'],
    mode='markers',
    marker=dict(
        color=cls_enc,
        opacity=0.5,
    )
)

g_data = [g]

layout = go.Layout(margin=dict(
    l=0, r=0, b=0, t=0
))

figure = go.Figure(data=g_data, layout=layout)

iplot(figure, filename='3d-repr-redshift')
```

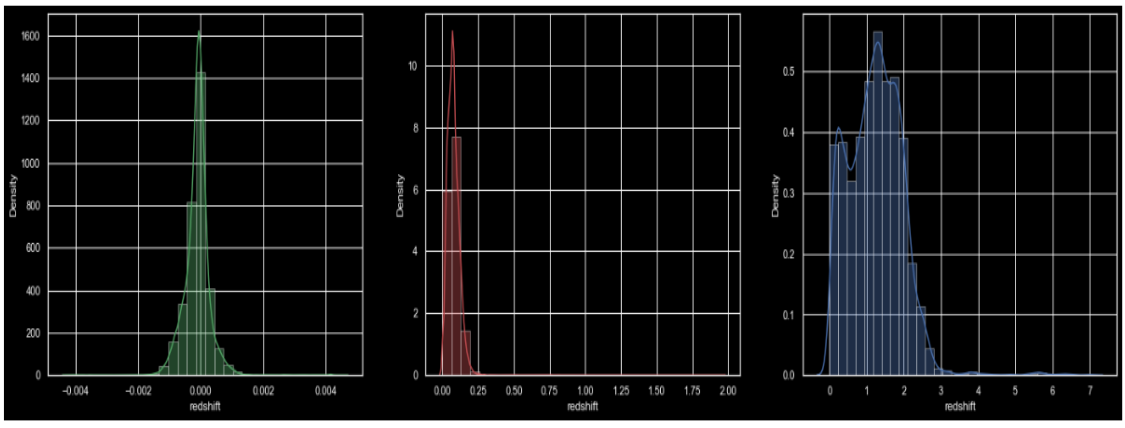
5.5 Subplot

In [29]:

```
fig, (ax1, ax2, ax3) = plt.subplots(ncols = 3, figsize = (24, 6))
sns.distplot(df[df['class'] == 'STAR'].redshift, ax = ax1, bins = 30, color = 'g')
sns.distplot(df[df['class'] == 'GALAXY'].redshift, ax = ax2, bins = 30, color = 'r')
sns.distplot(df[df['class'] == 'QSO'].redshift, ax = ax3, bins = 30, color = 'b')
```

Out[29]:

```
<AxesSubplot:xlabel='redshift', ylabel='Density'>
```



In [41]:

```
def distribution(df, axes, feature, row):
    '''Plot the distribution of a space object w.r.t. a given feature.'''
    labels = np.unique(df['class'])
    colors = ['skyblue', 'gold', 'red']
    for i in range(len(labels)):
        label = labels[i]
        ax = sns.distplot(df.loc[df['class']==label, feature],
                          kde=False, bins=30, ax=axes[row, i], color=colors[i])
        ax.set_title(label)
        if (i == 0):
            ax.set(ylabel='Count')
```

Removing Outliers

In [4]:
```
#Sometimes due to outliers, we can't do good analysis of our data therefore, we decided to remove them

def rem_outliers():
    s1 = df.shape

    for i in df.select_dtypes(include = 'number').columns:
        qt1 = df[i].quantile(0.25)
        qt3 = df[i].quantile(0.75)
        iqr =  qt3 - qt1
        lower = qt1-(1.5*iqr)
        upper = qt3+(1.5*iqr)
        min_in = df[df[i]<lower].index
        max_in = df[df[i]>upper].index
        df.drop(min_in, inplace = True)
        df.drop(max_in, inplace = True)

    s2 = df.shape
    outliers = s1[0] - s2[0]
    return outliers
```

In [5]:
```
print("Number of outliers deleted are : ", rem_outliers())
```

```
Number of outliers deleted are :  22790
```

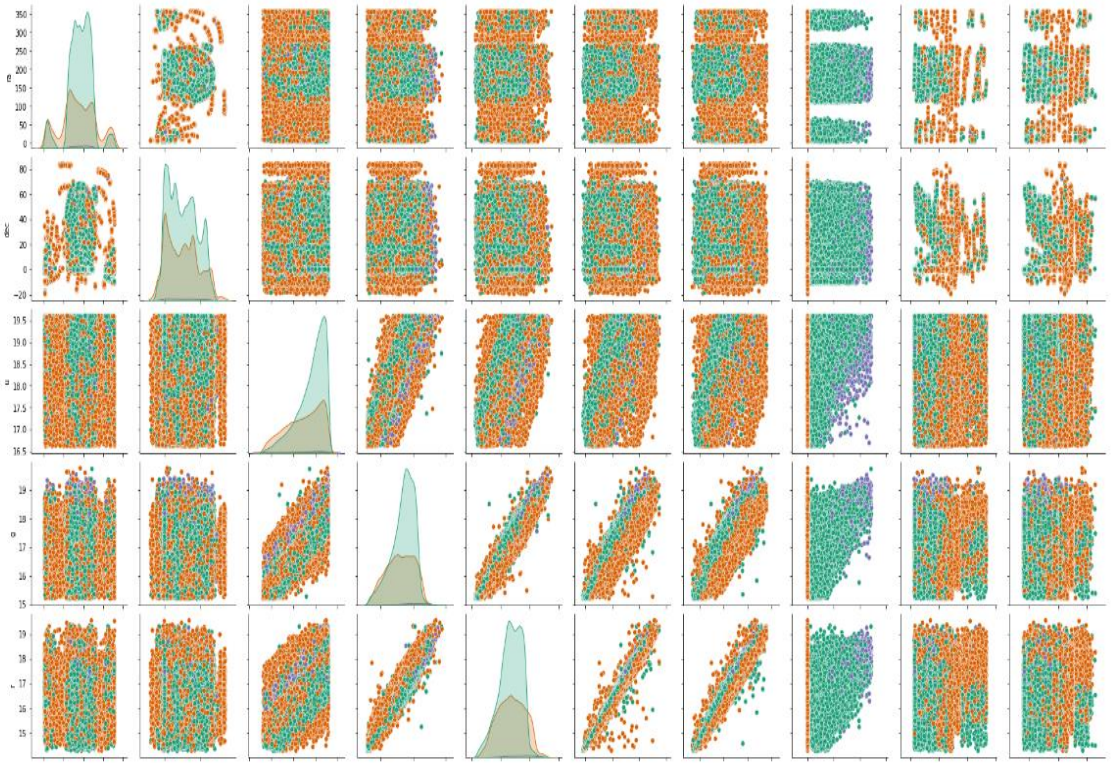
In [6]:
```
df.shape
```

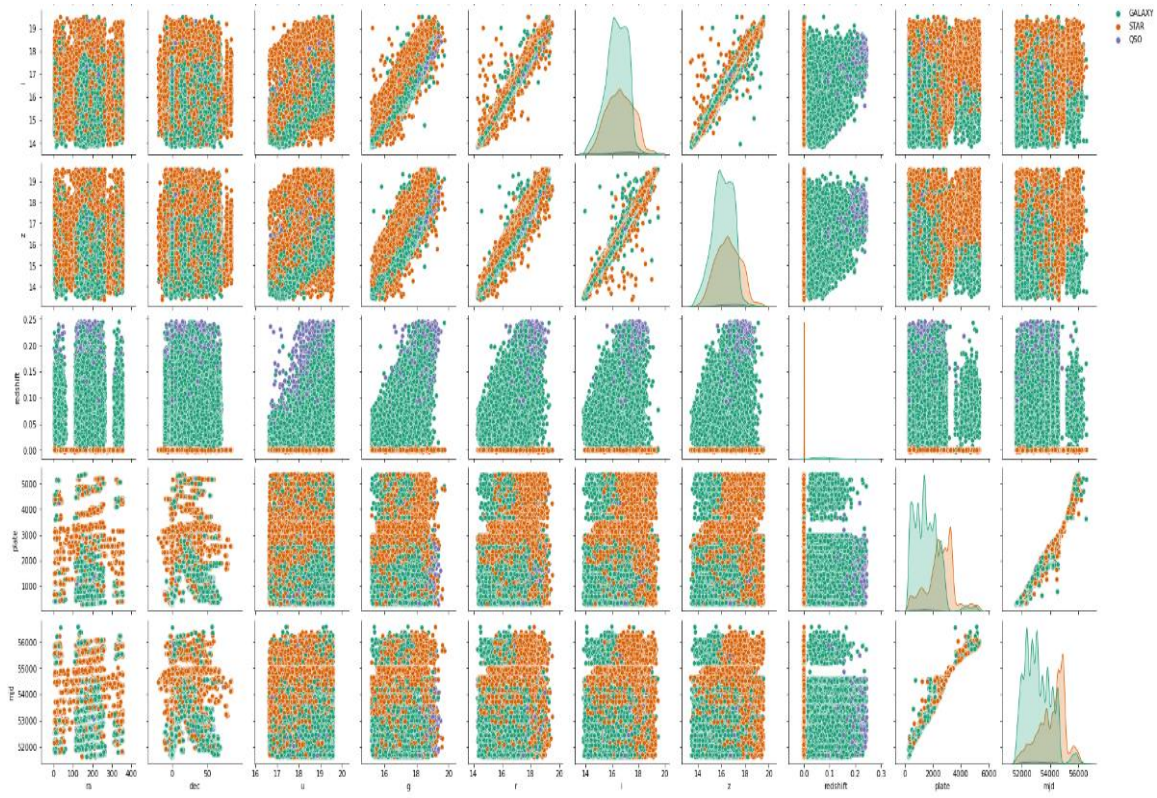
Out[6]:
```
(77210, 11)
```

5.4 Pairplot

In [18]:
```
sns.pairplot(df,palette = 'Dark2',hue = 'class')
```

Out[18]:
```
<seaborn.axisgrid.PairGrid at 0x146153f90>
```





5.5 Correlation Matrix

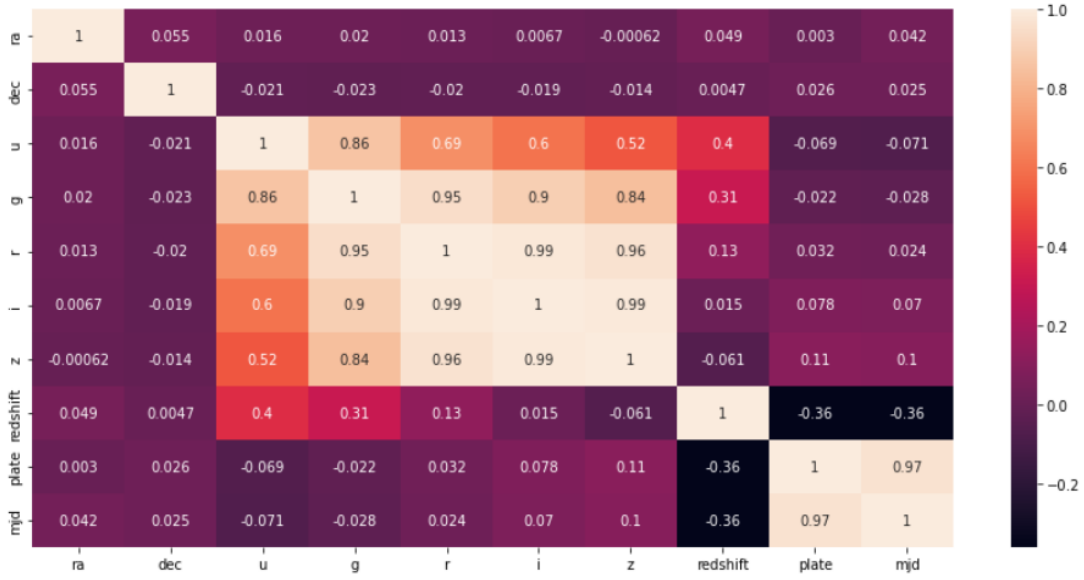
[16]:
```
plt.figure(figsize=(15,7))
sns.heatmap(df.corr(), annot=True)
df.corr()
```

5.5 Correlation Matrix

In [16]:
```
plt.figure(figsize=(15,7))
sns.heatmap(df.corr(), annot=True)
df.corr()
```

Out[16]:

	ra	dec	u	g	r	i	z	redshift	plate	mjd
ra	1.000000	0.054652	0.016463	0.019506	0.013206	0.006705	-0.000617	0.049057	0.003047	0.041765
dec	0.054652	1.000000	-0.020849	-0.023028	-0.019869	-0.019009	-0.014103	0.004714	0.025852	0.025079
u	0.016463	-0.020849	1.000000	0.859561	0.685888	0.595883	0.516801	0.397221	-0.068822	-0.070772
g	0.019506	-0.023028	0.859561	1.000000	0.949588	0.895153	0.843922	0.308837	-0.022102	-0.027656
r	0.013206	-0.019869	0.685888	0.949588	1.000000	0.985407	0.963497	0.129829	0.031882	0.024378
i	0.006705	-0.019009	0.595883	0.895153	0.985407	1.000000	0.990258	0.014931	0.078392	0.069568
z	-0.000617	-0.014103	0.516801	0.843922	0.963497	0.990258	1.000000	-0.061164	0.110117	0.100618
redshift	0.049057	0.004714	0.397221	0.308837	0.129829	0.014931	-0.061164	1.000000	-0.361682	-0.357039
plate	0.003047	0.025852	-0.068822	-0.022102	0.031882	0.078392	0.110117	-0.361682	1.000000	0.967004
mjd	0.041765	0.025079	-0.070772	-0.027656	0.024378	0.069568	0.100618	-0.357039	0.967004	1.000000



5.6 Pandas Profiling

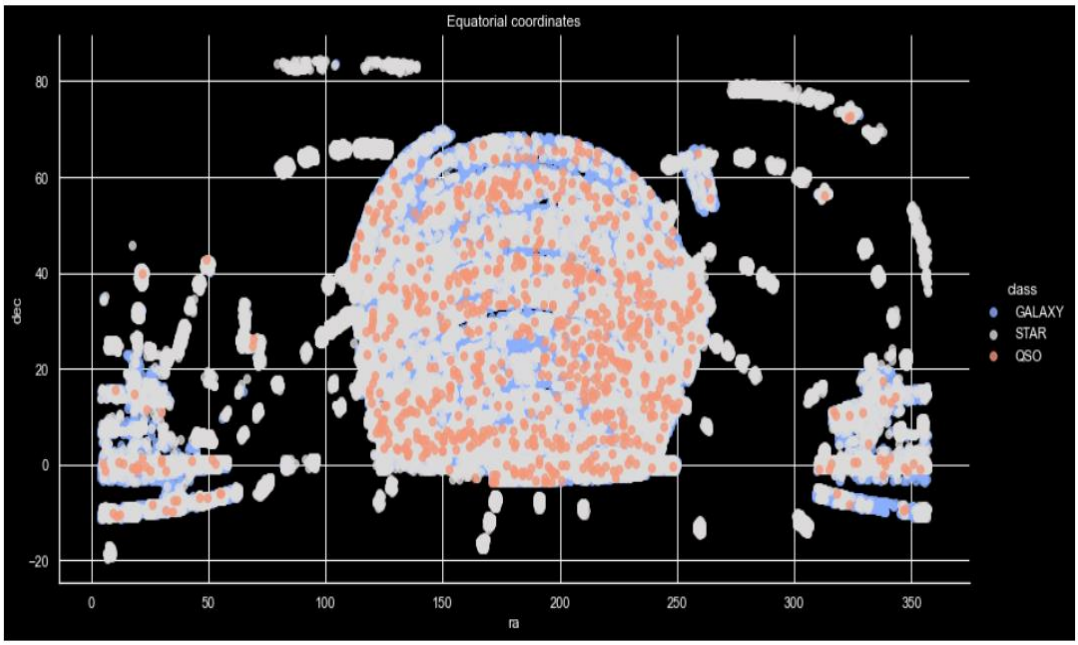
In [17]:
```
import pandas_profiling
profile = pandas_profiling.ProfileReport(df, title='SDSS dataset Report')
profile.to_file(output_file='SDSS_with_pandas_profiling.html')
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

5.7 Ra Vs Dec : Equatorial Coordinates Plot

In [38]:
```
sns.lmplot(x='ra', y='dec', data= df, hue='class', fit_reg=False, palette='coolwarm', size=6, aspect=2)
plt.title('Equatorial coordinates')
```

Out[38]:
```
Text(0.5, 1.0, 'Equatorial coordinates')
```



6. ML Algorithms

Encoding class labels For some cases, we cannot simply provide categorical values (just strings). Instead, we can convert them to numerical values. For example, since we have 3 classes, we able to assign to each class some values, so that:

0 is for GALAXY

1 is for QSO

2 is for STAR

In [16]:
```
from sklearn.model_selection import train_test_split, cross_val_predict, cross_val_score
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.metrics import confusion_matrix, precision_score
import time
```

In [17]:
```
# Some algorithms don't support categorical classes so we'll have to replace them with numbers
d=pd.DataFrame(df)


class_num=pd.DataFrame(LabelEncoder().fit_transform(d['class']), columns=['class'])
d.drop(['class'], axis=1, inplace=True)
names=list(d)
```

In [18]:
```
#Data are normalized for better conditioning of the problem

scaler = MinMaxScaler()
d=pd.DataFrame(scaler.fit_transform(d), columns=names)


d=pd.concat([d, class_num], axis=1)

d.head(3)
```

Out[18]:

	ra	dec	u	g	r	i	z	redshift	plate	mjd	class
0	0.583367	0.225219	0.997783	0.778255	0.336913	0.321274	0.343836	0.011317	0.048042	0.099809	0
1	0.381838	0.609451	0.966539	0.820604	0.378944	0.370465	0.415422	0.013936	0.046087	0.096122	0
2	0.039201	0.333105	0.964775	0.761909	0.336000	0.320879	0.347211	0.011836	0.012621	0.029082	0

In [19]:
```
#A cross validation will be performed to ensure the reliability of the results.

#In addition, an isolated training will serve to measure the times and extract a matrix of confusion than will give us a general idea.

x=d.drop('class',axis=1);
y=d['class']

x_train, x_test, y_train, y_test = train_test_split(d.drop('class',axis=1), d['class'], test_size=0.4)
```

In [20]:
```
x_train
```

Out[20]:

	ra	dec	u	g	r	i	z	redshift	plate	mjd
1915	0.696110	0.442079	0.803974	0.610743	0.268988	0.253857	0.271178	0.010575	0.094211	0.212862
22431	0.317856	0.645875	0.979347	0.913337	0.423381	0.424965	0.476320	0.138814	0.130364	0.233342
40890	0.591940	0.274641	0.907972	0.707011	0.318233	0.303795	0.331695	0.000608	0.125641	0.298198
36949	0.680111	0.301879	0.985078	0.868108	0.404197	0.400506	0.449883	0.000600	0.309502	0.512425
27090	0.672230	0.539150	0.908377	0.769102	0.372227	0.373380	0.425057	0.000602	0.261542	0.464227
...	...	...	...	...	...	...	...	...	...	...
52376	0.606070	0.317455	0.953895	0.835129	0.388015	0.378510	0.426351	0.020815	0.117580	0.258875
19508	0.536677	0.750332	0.589594	0.512951	0.246654	0.246052	0.275726	0.000550	0.177510	0.404560
78143	0.353561	0.221655	0.807960	0.698748	0.325942	0.322106	0.358882	0.000617	0.074831	0.141180
81490	0.408687	0.345725	0.792799	0.692519	0.328517	0.326968	0.367168	0.000641	0.758733	0.843528
9087	0.424925	0.380120	0.991804	0.831666	0.377498	0.368896	0.408933	0.000704	0.237196	0.439924

60000 rows × 10 columns

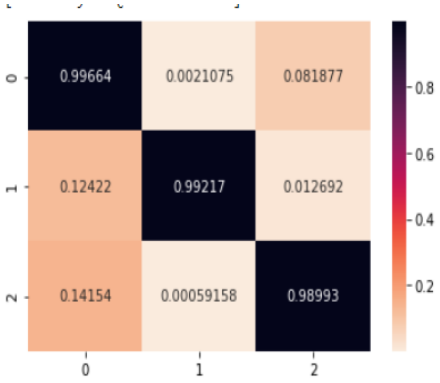
In [21]:
```
y_test
```

Out[21]:
```
30666    2
25983    0
55880    2
36800    2
38470    0
        ..
20954    0
62270    0
92296    0
67952    0
62992    0
Name: class, Length: 40000, dtype: int32
```

6.1 LOGISTIC REGRESSION

In [22]:
```
from sklearn import linear_model, datasets


lr = linear_model.LogisticRegression()

training_start = time.perf_counter()
lr.fit(x_train, y_train)#Training
training_end = time.perf_counter()

predict_start = time.perf_counter()
preds = lr.predict(x_test)#Prediction
predict_end = time.perf_counter()
acc_lreg = (preds == y_test).sum().astype(float) / len(preds)*100

print("The first iteration of the Logistic Regression gives an accuracy of the %3.2f %%" % (acc_lreg))

from numpy import linalg as LA
mc=confusion_matrix(y_test, preds)
mc_norm = mc / np.linalg.norm(mc, axis=1, keepdims=True)
sns.heatmap(pd.DataFrame(mc_norm), cmap=sns.cm.rocket_r, annot=True, fmt='.5g',);

print("[0=Galaxy 1=Quasar 2=Star]")
```

```
The first iteration of the Logistic Regression gives an accuracy of the 89.94 %
[0=Galaxy 1=Quasar 2=Star]
```



In [11]:
```
lr_train_t=training_end-training_start;
lr_predict_t=predict_end-predict_start;

scores = cross_val_score(lr, x, y, cv=10, scoring = "accuracy")
score_lr=scores.mean()
print("The 10 cross validations of Logistic Regression have had an average success rate of  %3.2f %%" %(score_lr*100))
std_lr=scores.std()
print("..and a standar deviation of %8.5f" %(std_lr))
```

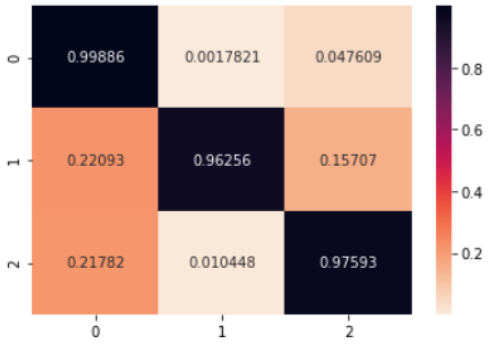
```
The 10 cross validations of Logistic Regression have had an average success rate of  90.84 %
..and a standar deviation of  0.00471
```

6.2 KNN Neighbors

In [10]:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

training_start = time.perf_counter()
knn.fit(x_train, y_train)
training_end = time.perf_counter()

predict_start = time.perf_counter()
preds = knn.predict(x_test)
predict_end = time.perf_counter()
acc_knn = (preds == y_test).sum().astype(float) / len(preds)*100

print("The first iteration of the K-Nearest Neighbours gives an accuracy of the %3.2f %%" % (acc_knn))



mc=confusion_matrix(y_test, preds)
mc_norm = mc / np.linalg.norm(mc, axis=1, keepdims=True)
sns.heatmap(pd.DataFrame(mc_norm), cmap=sns.cm.rocket_r, annot=True, fmt='.5g')


print("[0=Galaxy 1=Quasar 2=Star]")
```

```
The first iteration of the K-Nearest Neighbours gives an accuracy of the 87.41 %
[0=Galaxy 1=Quasar 2=Star]
```



In [ ]:

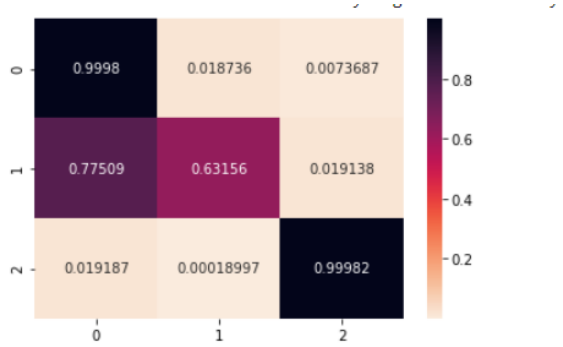
```
knn_train_t=training_end-training_start;
knn_predict_t=predict_end-predict_start;

scores = cross_val_score(knn, x, y, cv=10, scoring = "accuracy")
score_knn=scores.mean()
print("The 10 cross validations of K- Nearest Neighbours have had an average success rate of %3.2f %%" %(score_knn*100))
std_knn=scores.std()
print("..and a standar deviation of %8.5f" %(std_knn))
```

6.3 NAIVE BAYES

In [13]:

```
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

training_start = time.perf_counter()
gnb.fit(x_train, y_train)
training_end = time.perf_counter()

predict_start=time.perf_counter()
preds = gnb.predict(x_test)
predict_end = time.perf_counter()
acc_gnb = (preds == y_test).sum().astype(float) / len(preds)*100


print("The first iteration of the naive Bayes gives an accuracy of the %3.2f %%" % (acc_gnb))


mc=confusion_matrix(y_test, preds)
mc_norm = mc / np.linalg.norm(mc, axis=1, keepdims=True)
sns.heatmap(pd.DataFrame(mc_norm), cmap=sns.cm.rocket_r, annot=True, fmt='.5g');
```

```
The first iteration of the naive Bayes gives an accuracy of the 97.15 %
```



n [14]:

```
gnb_train_t=training_end-training_start;
gnb_predict_t=predict_end-predict_start;

scores = cross_val_score(gnb, x, y, cv=10, scoring = "accuracy")
score_gnb=scores.mean()
print("The 10 cross validations of naive Bayes have had an average success rate of %3.2f %%" %(score_gnb*100))
std_gnb=scores.std()
print("..and a standar deviation of %8.6f" %(std_gnb))
```

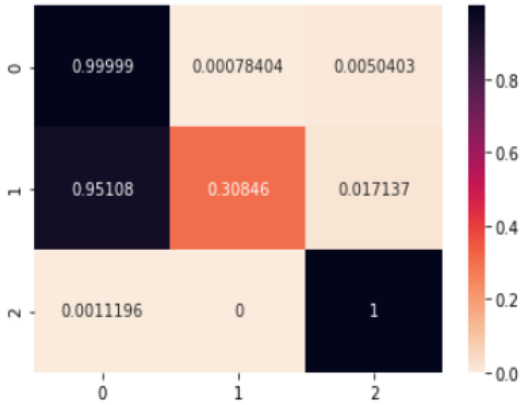
```
The 10 cross validations of naive Bayes have had an average success rate of 97.00 %
..and a standar deviation of 0.001918
```

6.4 RANDOM FOREST

In [15]:

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators=10)

training_start = time.perf_counter()
rfc.fit(x_train, y_train)
training_end = time.perf_counter()

predict_start=time.perf_counter()
preds = rfc.predict(x_test)
predict_end = time.perf_counter()

acc_rfc = (preds == y_test).sum().astype(float) / len(preds)*100


print("The first iteration of the Random Forest gives an accuracy of the %3.2f %%" % (acc_rfc))


mc=confusion_matrix(y_test, preds)
mc_norm = mc / np.linalg.norm(mc, axis=1, keepdims=True)
sns.heatmap(pd.DataFrame(mc_norm), cmap=sns.cm.rocket_r, annot=True, fmt='.5g');
```

```
The first iteration of the Random Forest gives an accuracy of the 98.82 %
```



In [16]:

```
rfc_train_t=training_end-training_start;
rfc_predict_t=predict_end-predict_start;

scores = cross_val_score(rfc, x, y, cv=10, scoring = "accuracy")
score_rfc=scores.mean()
print("The 10 cross validations of Random Forest have had an average success rate of %3.2f" %(score_rfc*100))
std_rfc=scores.std()
print("..and a standar deviation of %8.6f" %(std_rfc))
```

```
The 10 cross validations of Random Forest have had an average success rate of 98.96
..and a standar deviation of 0.000890
```

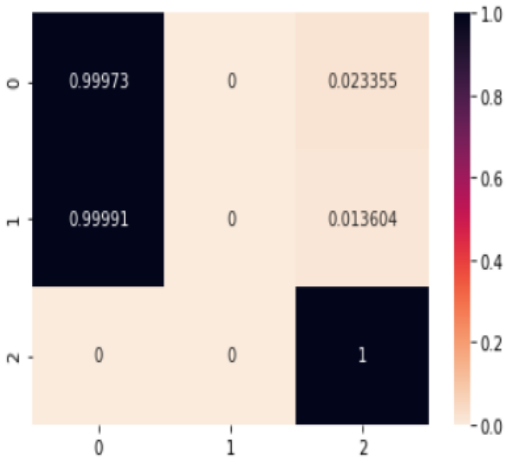
6.5 SUPPORT VECTOR MACHINES

In [17]:

```
from sklearn.svm import SVC

svm = SVC(kernel='sigmoid', gamma='auto')

training_start = time.perf_counter()
svm.fit(x_train, y_train)
training_end = time.perf_counter()

predict_start = time.perf_counter()
preds = svm.predict(x_test)
predict_end = time.perf_counter()

acc_svm = (preds == y_test).sum().astype(float) / len(preds)*100


print("The first iteration of the SVM gives an accuracy of the %3.2f %%" % (acc_svm))


mc=confusion_matrix(y_test, preds)
mc_norm = mc / np.linalg.norm(mc, axis=1, keepdims=True)
sns.heatmap(pd.DataFrame(mc_norm), cmap=sns.cm.rocket_r, annot=True, fmt='.5g');
```

```
The first iteration of the SVM gives an accuracy of the 97.56 %
```

The first iteration of the SVM gives an accuracy of the 97.56 %



[18]:

```
svm_train_t=training_end-training_start;
svm_predict_t=predict_end-predict_start;

scores = cross_val_score(svm, x, y, cv=10, scoring = "accuracy")
score_svm=scores.mean()
print("The 10 cross validations of SVM have had an average success rate of %3.2f %%" %(score_svm*100))
std_svm=scores.std()
print("..and a standar deviation of %8.6f" %(std_svm))
```

```
The 10 cross validations of SVM have had an average success rate of 97.53 %
..and a standar deviation of 0.001306
```

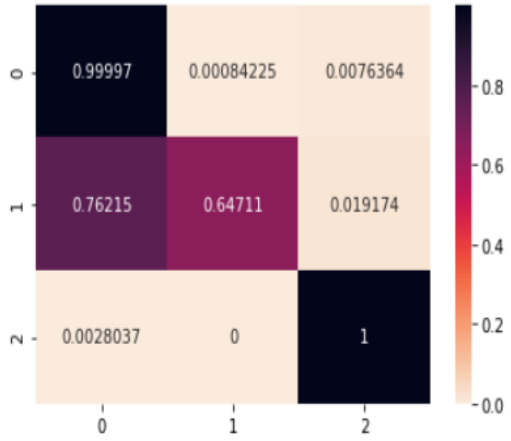
6.6 NEURAL NETWORKS

Multi-layer Perceptron classifier (MLPC)

In [19]:

```
from sklearn.neural_network import MLPClassifier

nnc = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(4, 3), random_state=1)

training_start = time.perf_counter()
nnc.fit(x_train, y_train)
training_end = time.perf_counter()

predict_start = time.perf_counter()
preds=nnc.predict(x_test)
predict_end=time.perf_counter()

acc_nnc = (preds == y_test).sum().astype(float) / len(preds)*100
print("The first iteration of the Neural Networks gives an accuracy of the %3.2f %%" % (acc_nnc))
mc=confusion_matrix(y_test, preds)
mc_norm = mc / np.linalg.norm(mc, axis=1, keepdims=True)
sns.heatmap(pd.DataFrame(mc_norm), cmap=sns.cm.rocket_r, annot=True, fmt='.5g');
```

```
The first iteration of the Neural Networks gives an accuracy of the 98.81 %
```



In [20]:

```
nnc_train_t=training_end-training_start;
nnc_predict_t=predict_end-predict_start;

scores = cross_val_score(nnc, x, y, cv=10, scoring = "accuracy")
score_nnc=scores.mean()
print("The 10 cross validations of Neural Networks have had an average success rate of %3.2f %%" %(score_nnc*100))
std_nnc=scores.std()
print("..and a standar deviation of %8.6f" %(std_nnc))
```

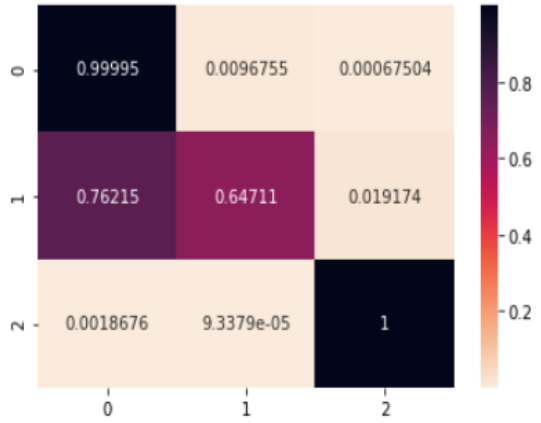
```
The 10 cross validations of Neural Networks have had an average success rate of 98.60 %
..and a standar deviation of 0.002011
```

6.7 DECISION TREE

In [23]:

```
from sklearn.tree import DecisionTreeClassifier

nnc = DecisionTreeClassifier(criterion='entropy', random_state=0)

training_start = time.perf_counter()
nnc.fit(x_train, y_train)
training_end = time.perf_counter()

predict_start = time.perf_counter()
preds=nnc.predict(x_test)
predict_end=time.perf_counter()

acc_nnc = (preds == y_test).sum().astype(float) / len(preds)*100
print("The first iteration of the Decision tree gives an accuracy of the %3.2f %%" % (acc_nnc))
mc=confusion_matrix(y_test, preds)
mc_norm = mc / np.linalg.norm(mc, axis=1, keepdims=True)
sns.heatmap(pd.DataFrame(mc_norm), cmap=sns.cm.rocket_r, annot=True, fmt='.5g');
```

```
The first iteration of the Decision tree gives an accuracy of the 98.73 %
```



In [24]:

```
nnc_train_t=training_end-training_start;
nnc_predict_t=predict_end-predict_start;

scores = cross_val_score(nnc, x, y, cv=10, scoring = "accuracy")
score_nnc=scores.mean()
print("The 10 cross validations of Decision tree have had an average success rate of %3.2f %%" %(score_nnc*100))
std_nnc=scores.std()
print("..and a standar deviation of %8.6f" %(std_nnc))
```

```
The 10 cross validations of Decision tree have had an average success rate of 98.74 %
..and a standar deviation of 0.000517
```

EDA Section

1. Stars have the lowest average redshift, followed by Galaxies and then Quasars.

2. u,g,r,i,z correlation looks in accordance with expected physical behaviour - Hotter objects emit more of every wavelength.

3. all wavelength radiations are strongly correlated except u as u has low correlation.

4. Redshift gives us the distance between us and the object. Redshift value based on the increase in wavelength.

5. There is not much difference in distribution according to class, but I can see that there are some characteristics to distinguish star class.

6. From the violin plot it can be said that, value of obj_ID doesn't contribute in classifying Star or Galaxy.

7. From the violin plot it can be concluded that values of alpha doesn't contribute in classifying Star and Galaxy. So, classification of star and galaxy doesn't depend on the right ascension angle

8. In the 'redshift' feature, if the value is negative(blueshift), the observation is more likely to be a Star. If the value is positive(redshift), the observation is more likely to be Galaxy.

ML Section

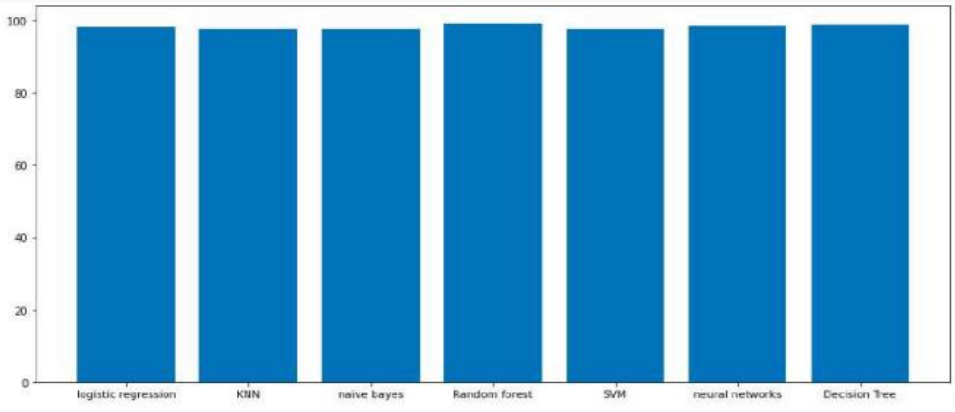
1. Random Forest scores the best accuracy and the minimum deviation, so we can consider it as the best classifier for this problem.

2. Galaxies are easier to separate than stars and quasars could have done.

3. Although quasars and stars shows parallel quantities, they have some distinctive statistical distributions to filters to help to decide which one it is.

COMPARISON OF MODEL ACCURACIES

```
In [96]: data = {'logistic regression':score_lr,'KNN':score_knn,'naive bayes':score_gnb,
                 'Random forest':score_rfc,'SVM':score_svm,'neural networks':score_nnc,'Decision Tree':score_nnd}
         val1 = list(data.keys())
         val2 = list(data.values())
         per = [element * 100 for element in val2]

         for index, value in enumerate(per):
             plt.text(value, index,
                      str(value))

         fig = plt.figure(figsize = (15,7))
         plt.bar(val1,per)


         plt.show()
```




```
In [93]: per

Out[93]: [98.16085999222899,
          97.63113586323013,
          97.43297500323791,
          99.08820101023184,
          97.53011267970471,
          98.40046626084705,
          98.79160730475327]
```

CONCLUSION:

The approaches used in this project can be used to solve the star,quasar and galaxy classification problem in particular and other problems in astronomy in general. These classifiers can be used to classify multi- wavelength astronomical data sources and pre-select quasar candidates for large surveys. The project is firmly focussed on scientific correctness and algorithmic relevance. Different ML algorithms have been applied and should be interpreted in that light , not as a suite of trial and error approaches to pick the better ones.

The main goal of the model to represent a colourful and cheesy version of Astrophysics and Cosmology in Tv shows and Science books but in reality its mostly about staring at data tables, graphs and equations for majority of your time. This study has partially been successful at providing a glimpse in to the tools and methodology that researchers apply on real world problems. This is a humble attempt at representing a cross section of the real scientific analysis that happens all around the world even as this is written.

- The application of EDA on Astronomy DataSet from SDSS has given insights into how photometric parameters contribute to class labels such as STAR< GALAXY, QUASAR
- EDA with Statistical summaries as well as extensive visualizations provided the guidelines to transform the data as input for ML models.
- A comparison of the accuracy and performance of the different ML models were done.

REFERENCES:

1. The data released by the SDSS is under public domain. It's taken from the current data release RD14. More information about the license: http://www.sdss.org/science/image-gallery/
2. It was acquired by querying the CasJobs database which contains all the data published by the SDSS. The exact query can be found at: http://skyserver.sdss.org/CasJobs/
3. Data Release 2 of S-PLUS: Accurate template-fitting based photometry covering 1000 deg2 in 12 optical filters.
4. miniJPAS survey: star-galaxy classification using machine learning.
5. https://engineering.papercup.com/posts/kernel-methods/

6. http://skyserver.sdss.org/dr2/en/proj/advanced/color/sdssfilters.asp
7. https://machinelearningmastery.com/support-vector-machines-for-machine-learning/