Data Analysis and Morphological Classification of Galaxies from SDSS

Abstract:

According to Hubble Sequence, galaxies are divided into four morphological groups: spiral, barred spiral, elliptical, and irregular. For the study of cosmos- Ground-based telescopes aren't appropriate, because the Earth's atmosphere absorbs a lot of the infrared and ultraviolet light that passes through it. A space telescope or space observatory is an instrument located in outer space to observe distant planets, galaxies and other astronomical objects. Space telescopes avoid many of the problems such as light pollution and distortion of electromagnetic radiation (scintillation); ultraviolet frequencies, X-rays and gamma rays are blocked by the Earth's atmosphere, so they can only be observed from space. The main object of this project is to: create a huge dataset and then classify it.- to determine which all are stars, planets or galaxies. Then, we'll go further. we will try to classify each galaxy according to their types/shapes/morphologies and see the distributions

MOTIVATION:

Chandra is one of the Great Observatories in India, along with the Hubble Space Telescope in NASA, Compton Gamma Ray Observatory (1991–2000), and the Spitzer Space Telescope. The telescope is named after the Nobel Prize-winning Indian-American astrophysicist Subrahmanyan Chandrasekhar.

When I came to know about this, it gave me food for my imagination about outer galaxies.

Space telescopes tend to avoid real life problems, as - Light pollution, Scinrillation (Distortion of electro-magnetic radiation) etc. Moreover - Ultra violent frequencies, X-Rays and gamma rays are blocked by the Earth's atmosphere, so they can only be observed from space.

Being a huge Christopher Nolan fan (obviously am talking about the sci-fi's) my object to develop this project is to: Create a huge dataset and then classify it, to determine which all are stars or planets or galaxies.

Introduction:

Astronomy dates back as far as the ancient Indians, the Egyptians, the Mayans, the Inkas and the Mesopotemians. Astronomy was used as a means of keeping track of time and predicting future events which was achieved through a combination of religion, astrology and the meticulous study of the positions and motions of various celestial bodies. It is generally believed that priests were the first professional astronomers, the pioneers of the field. The real renaissance of astronomy began in the 1500s when Nicholaus Copernicus, a Polish university-trained Catholic priest, mathematician and astronomer, proposed a heliocentric model of our Universe in which the Sun, rather than the Earth, is in the center of the Solar System.

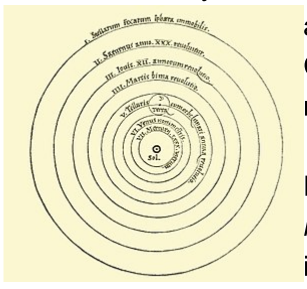
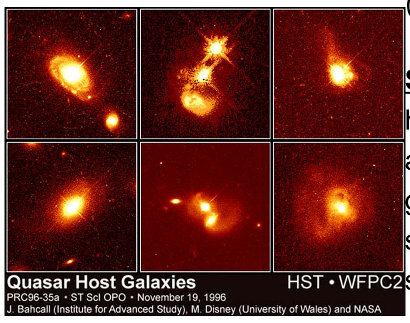


Figure in the left graphically illustrates this model. Just before he died in 1543, he published a book entitled *De revolutionibus orbium coelestium (On the Revolutions of the Celestial Spheres)* which became one of the most important contributions towards the scientific revolution.

Terminologies:

Galaxy: A galaxy is defined as a populous system of stars, dust, dark matter and gases that are all bound together by gravity. There are numerous sizes that a galaxy can possess in terms of the number of stars that live within it, ranging anywhere from 10 million to 10 trillion stars. There are two common general shapes that a galaxy can take, either spiral or elliptical. Many variations within each also exist as well as less-common shapes such as toothpicks or rings.

Messier 64 (M64) *- A spiral galaxy, the result of a collision between two galaxies. Due to the spectacular dark band of dust surrounding the galaxy's bright nuclues, its been nicknamed by some as the Evil Eye galaxy (credit: hubblesite.org).*


Quasar Host Galaxies
HST • WFPC2
PRC96-35a • ST ScI OPO • November 19, 1996
J. Bahcall (Institute for Advanced Study), M. Disney (University of Wales) and NASA

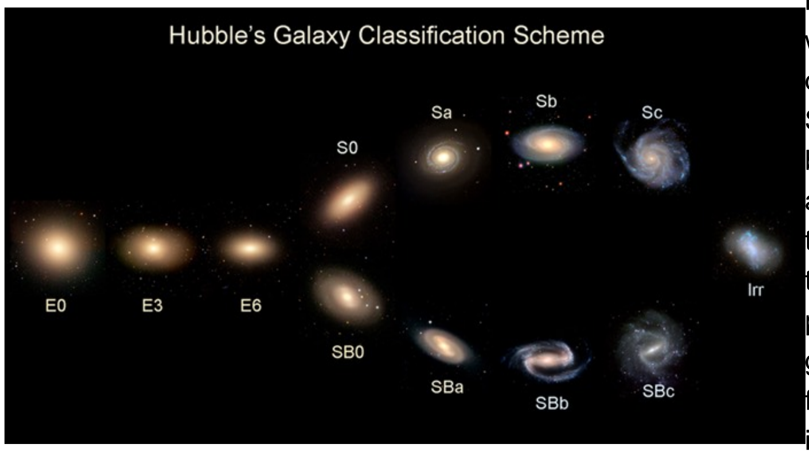
Star: A star, our Sun being a perfect example, is essentially a sphere of immensely hot gas, mainly that of hydrogen, partly helium and with minute traces of other various gases. Within its core, an incredible amount of energy is generated through the process of fusion in which smaller atoms smash together at great speeds to form larger atoms. Likewise with galaxies, astronomers also have a process for classifying stars. They are grouped into spectral types. By spectral, we refer to the temperature and brightness of the surfaces of the stars.

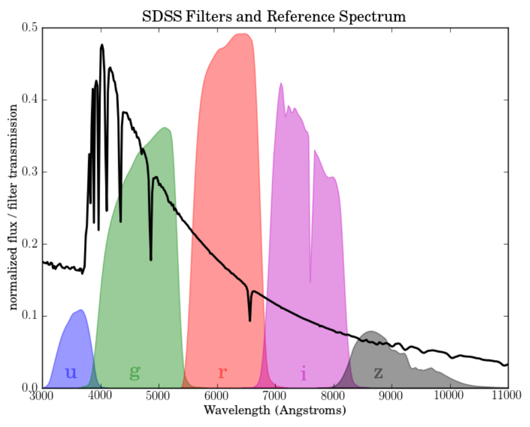
Quasar: a massive and extremely remote celestial object, emitting exceptionally large amounts of energy, and typically having a starlike image in a telescope. It has been suggested that quasars contain massive black holes and may represent a stage in the evolution of some galaxies.*(Above figure contains some of the clearest light views of quasar taken till date.) (credit: nasa.gov)*

Morphology: ***Galaxy morphology*** *is a visual grouping system used by astronomers for galaxy classification*. Almost a hundred years have now passed since it was first discovered that galaxies were independent systems subjected to morphology and mergers.

The most famous system for morphological classification, known as the ***Hubble Tuning Fork System****, was proposed by American astronomer* ***Edwin Powell Hubble*** *in the year 1936.*

In 1936, when Hubble released his book Realm of the Nebulae, that research in the area became abundant and the study of galaxy





morphology became a well established sub-field of optical astronomy. Since then, there have been numerous studies and proposed revisions to the **Fork Sequence** that have been published. Hubble is generally regarded, in the field, as one of the most important observational cosmologists of the 20th century who played an important role in the establishment of the field of extragalactic astronomy.

Redshift: A shift toward longer wavelengths of the spectral lines emitted by a celestial object that is caused by the object moving away from the earth. It is used by astronomers to compare the distances among two faraway celestial objects.

Objectives:

The datasets, which will be used in this project are not balanced. SDSS left almost 70% of it's entries unclassified. On the other hand most of the galaxies in the GalaxyZoo project are labelled as Uncertain. This is because GalaxyZoo uses human classifiers and in many cases the volunteers do not agree on a single conclusion. A 80% voting threshold is used to determine the morphology of each galaxy. Although such a high threshold is desirable, the outcome has significantly increased uncertainty instead of helping astronomers and cosmologists in their quest to unfold the facts about our universe. Combined datasets contain some probabilistic scores of each morphological classes according to each entry. This scores will help a lot in our research work.

Main aim of this project is to perform an intelligent data analysis technique to resolve this uncertainities. Three datasets with 1000,10000 and 100000 are used here for performance comparision purposes.

Requirement Analysis and Specification:

Data Excavation Sites:

Datasets used this project are mined from the following servers;

1. ***GalaxyZoo1:***
   Galaxy Zoo is a crowdsourced astronomy project which invites people to assist in the morphological classification of large numbers of galaxies. It is an example of citizen science as it enlists the help of members of the public to help in scientific research.There have been 15 versions as of July 2017, many of which are outlined in this article. Galaxy Zoo is part of the Zooniverse, a group of citizen science projects. An outcome of the project is to better determine the different aspects of objects and to separate them into classifications.

   A subscript from their official page says the following:

   > "*Galaxy Zoo is a scientific project that has invited members of the public to help classify a million galaxies. Those involved are directly contributing to scientific research, while getting an opportunity to view the beautiful and varied galaxies that inhabit our universe.Why do we need people to do this, rather than just using a computer? The simple answer is that the human brain is much better at recognising patterns than a computer. Galaxies are complicated objects that vary in appearance enormously, and yet in some ways they can be very similar. We could write a computer program to classify these galaxies, and many researchers have, but so far none have really done a good enough job. We have not been able to make computers 'see past' the complexity, to reliably identify the similarities that appear obvious to our eyes and brain. For now, and probably for some time yet, people do the best job of classifying galaxies.*"
   >
   > > "*Getting all these galaxy classifications is just the first stage of our project. What we really want to do is some science, to try and understand what kind of galaxies there are, how they formed, and the processes that have changed them into the systems we see today. The journey from raw classifications to accepted scientific results is often long and arduous. In this blog we aim to explain the steps involved, and the scientific ideas we are working on.*"

2. ***GalaxyZoo2:*** The original Galaxy Zoo consisted of a data set made up of ≈900,000 galaxies imaged by the Sloan Digital Sky Survey. With so many galaxies, it had been assumed that it would take years for visitors to the site to work through them all, but within 24 hours of launch, the website was receiving almost 70,000 classifications an hour. In the end, more than 50 million classifications were received by the project during its first year, contributed by more than 150,000 people. This was started in July 2007 and retired in 2009. **GalaxyZoo2** consisted of some 250,000 of the brightest galaxies from the Sloan Digital Sky Survey. Galaxy Zoo 2 allowed for a much more detailed classification, by shape and by the intensity or dimness of the galactic core, and with a special section for oddities like mergers or ring galaxies. The sample also contained fewer optical oddities. The project closed with some 60 million classifications

3. ***Sloan Digital SkyServer***(*10th,12th and 14th release*)**:** The Sloan Digital Sky Survey or SDSS is a major multi-spectral imaging and spectroscopic redshift survey using a dedicated 2.5-m wide-angle optical telescope at Apache Point Observatory in New Mexico, United States. The project was named after the Alfred P. Sloan Foundation, which contributed significant funding.

Data collection began in 2000; the final imaging data release (DR9) covers over 35% of the sky, **with photometric observations of around nearly 1 billion objects,** *while the survey continues to acquire spectra, having so far taken spectra of over 4 million objects.* The main galaxy sample has a median redshift of z = 0.1; there are redshifts for luminous red galaxies as far as z = 0.7, and for quasars as far as z = 5; and the imaging survey has been involved in the detection of quasars beyond a redshift z = 6.

Data release 8 (DR8)**, released in January 2011*, includes all photometric observations taken with the SDSS imaging camera, covering 14,555 square degrees on the sky *(just over 35% of the full sky)*. ***Data release 9 (DR9), released to the public on **31 July 2012**, includes the first results from the ***Baryon Oscillation Spectroscopic Survey (BOSS)*** spectrograph, including over 800,000 new spectra. Over 500,000 of the new spectra are of objects in the Universe 7 billion years ago *(roughly half the age of the universe)*. **Data release 10 (DR10), released to the public on 31 July 2013, includes all data from previous releases, plus the first results from the *APO Galactic Evolution Experiment (APOGEE)*** spectrograph, including over 57,000 high-resolution infrared spectra of stars in the Milky Way. ***DR10 also includes over 670,000 new BOSS spectra of galaxies and quasars in the distant universe.*** *The publicly available images from the survey were made between 1998 and 2009.*

Query used for mining the Data

```
getData function(n){
    select top n
    p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
    p.run, p.rerun, p.camcol, p.field,
     p.mCr4_u,p.mCr4_g,p.mCr4_r,p.mCr4_i,p.mCr4_z,
     p.mCr4PSF_u,p.mCr4PSF_g,p.mCr4PSF_r,p.mCr4PSF_i,p.mCr4PSF_z,
     p.petroR50_u,p.petroR50_g,p.petroR50_r,p.petroR50_i,p.petroR50_z,
     p.petroR90_u,p.petroR90_g,p.petroR90_r,p.petroR90_i,p.petroR90_z,p.type,
     s.specobjid, s.class,s.subClass, s.z as redshift,
     s.plate, s.mjd, s.fiberid,s.sourceType,
     z.nvote,z.p_el,z.p_cw,z.p_acw,z.p_edge,z.p_dk,z.p_mg
     FROM PhotoObjAll AS p
     JOIN SpecObjAll AS s ON s.bestObjID = p.objID JOIN zooNoSpec
     as z on p.objID = z.objID
}
```

Data Description: First Dataset contains 1000 entries from GalaxyZoo Second and Third Dataset contains 10000 and 100000 combined entries respectively from all the servers and multiple relations.

Tools and Languages used:

Languages Used:

1. **Python**:
   Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace.
2. **SQL**:
   SQL is a domain-specific language used in programming and designed for managing data held in a relational database management system, or for stream processing in a relational data stream management system.
3. **Markdown**:
   Markdown is a lightweight markup language with plain text formatting syntax. Its design allows it to be converted to many output formats, but the original tool by the same name only supports HTML.

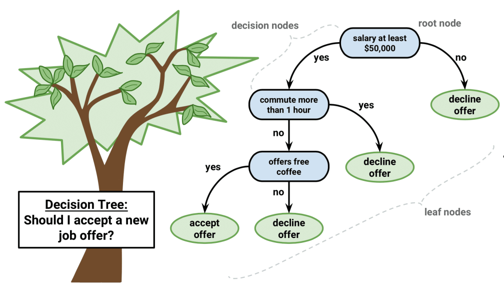
Tools Used: :

1. **Spyder3 IDE**:
   Spyder is a powerful scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts.
2. **Space Engine**:
   SpaceEngine is a proprietary 3D astronomy program and game engine developed by Russian astronomer and programmer Vladimir Romanyuk. It creates a three-dimensional planetarium representing the entire universe from a combination of real astronomical data and scientifically-accurate procedural generation algorithms.
3. **Stellarium**:
   Stellarium is an open-source free-software planetarium, licensed under the terms of the GNU General Public License version 2, available for Linux, Windows, and macOS. A port Stellarium called Stellarium Mobile is available for Android, iOS, and Symbian as a paid version, being developed by Noctua Software.
4. **SDSS SkyServer SQL Search Tool**:
   Search tool for the public database of SDSS.
5. **Hubble Voyage**:
   Exploration tools especially made for students, educators and hobbyists. Used for checking observations manually.
6. **Astrometry**: Exploration tool for constellations.

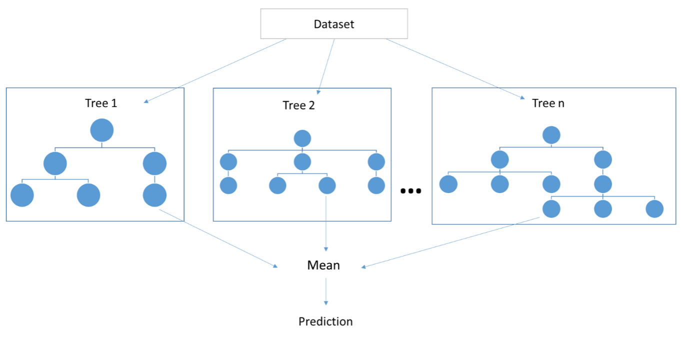
Data can be easily mined from the skyserver by using their own sql based search tool which is available at *https://skyserver.sdss.org/dr12/en/tools/search/sql.aspx (https://skyserver.sdss.org/dr12/en/tools/search/sql.aspx)* . Observations can be checked manually using another tool from the skyserver popularly known as the **voyage:** *http://cas.sdss.org/dr15/en/tools/chart/navi.aspx (http://cas.sdss.org/dr15/en/tools/chart/navi.aspx)* and is widely used by the educators and students from all over the world! or ;
astrometry: *http://nova.astrometry.net/user_images (http://nova.astrometry.net/user_images)*

Algorithms Used:

1. **Decision Tree Algorithm:**
   DescriptionIn computer science, Decision tree learning uses a decision tree to go from observations about an item to conclusions about the item's target value. It is one of the predictive modeling approaches used in statistics, data mining and machine learning.

2.


Random Forest Algorithm: Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

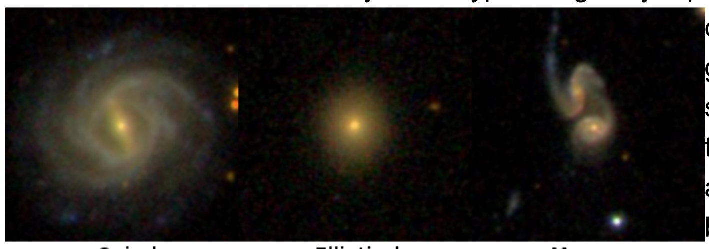


Importing Libraries:

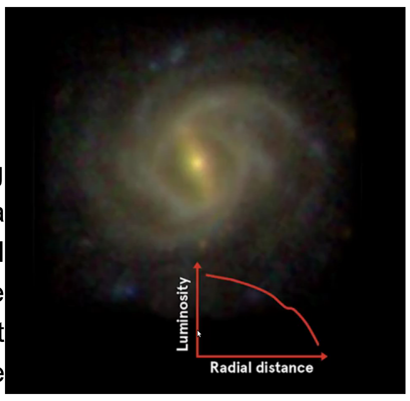
In [210]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict
from sklearn.tree import DecisionTreeClassifier
import itertools

import seaborn as sns
import plotly.offline as py
color = sns.color_palette()
import plotly.graph_objs as go
py.init_notebook_mode(connected=True)
import plotly.tools as tls

from IPython.display import Math

import warnings
warnings.filterwarnings('ignore')

SMALL_SIZE = 10
MEDIUM_SIZE = 12

plt.rc('font', size=SMALL_SIZE)
plt.rc('axes', titlesize=MEDIUM_SIZE)
plt.rc('axes', labelsize=MEDIUM_SIZE)
plt.rcParams['figure.dpi']=150
```

Dataset 1:

This dataset is limited to only three types of galaxy: spirals, ellipticals and mergers. It is a sample of galaxies where at least 20 human classifiers agreed to the same galaxy type. Examples of spiral and elliptical galaxies were selected where there was a unanimous classification. Due to low sample numbers, merger examples ar included which holds an 80% threshold, so that the classifier can be trained in a better way. Features used in these datasets are derived from fitting images according to known galaxy profiles and are mostly based on flux magnitudes.(SDSS used 'u','g','r','i','z' photometric system).



Feature Description:



Color Index: Spectroscopic wavebands (u,g,r,i,z for this project).
Adaptive moments: Adaptive moments are the second moments of the object intensity, measured using a particular scheme designed to have near-optimal signal-to-noise ratio. Moments are measured using a radial weight function interactively adapted to the shape (ellipticity) and size of the object. This elliptical weight function has a signal-to-noise advantage over axially symmetric weight functions. In principle there is an optimal (in terms of signal-to-noise) radial shape for the weight function, which is related to the light profile of the object itself. Adaptive moments also describe the shape of a galaxy. They are used in image analysis to detect similar objects at different sizes and orientations. We use the fourth moment here for each band.
Concentration: Concentration is similar to the luminosity profile of the galaxy, which measures what proportion of a galaxy's total light is emitted within what radius. A simplified way to represent this is to take the ratio of the radii containing 50% and 90% of the Petrosian flux. The Petrosian method allows us to compare the radial profiles of galaxies at different distances. If you are interested, you can read more here on the need for Petrosian approach. For these experiments, we will define concentration as:

In [2]:
```python
Math(r'\mbox{conc} = \frac{\mbox{petro}_{R50}}{\mbox{petro}_{R90}}')
```

Out[2]:
$$\mbox{conc} = \frac{\mbox{petro}_{R50}}{\mbox{petro}_{R90}}$$

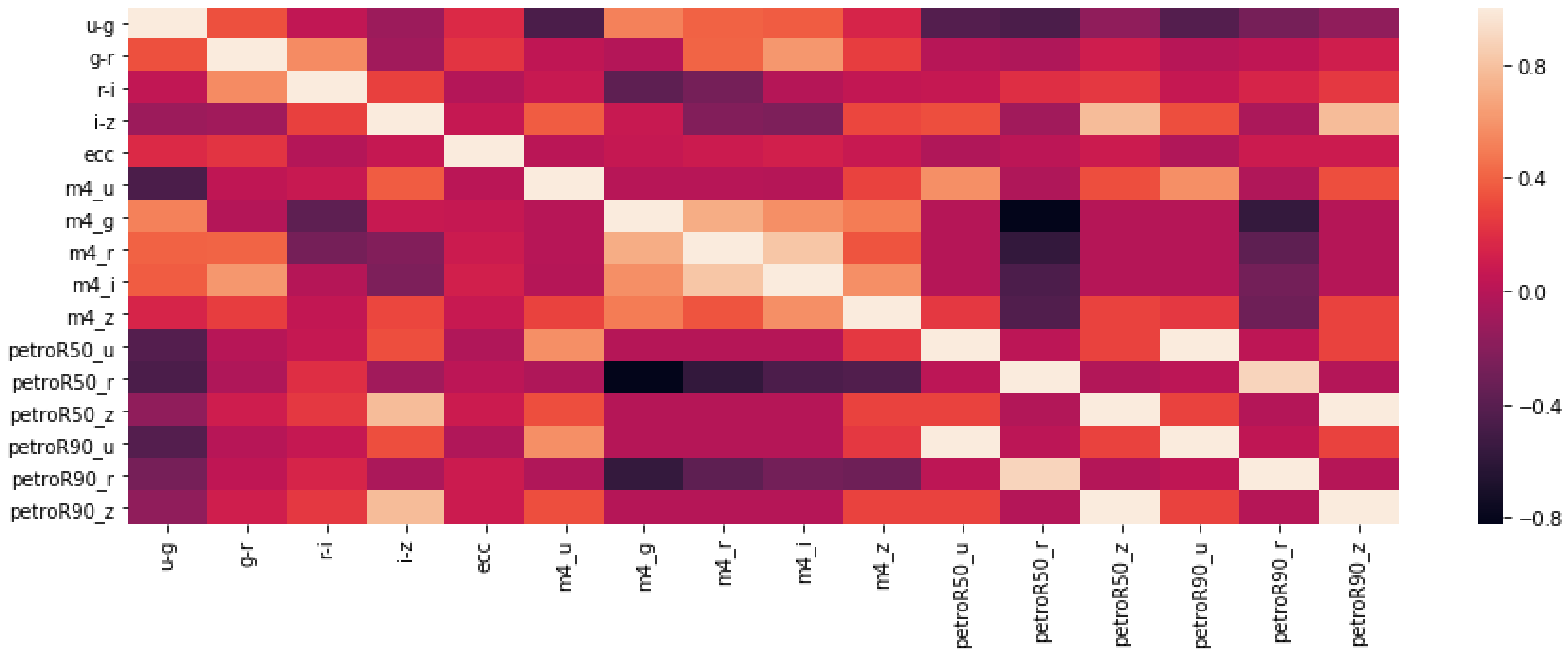
We will use the concentration from the u, r and z bands.

In [3]:
```
data1 = pd.DataFrame(np.load('D:\\\ThirdFinalProject\\\Data_Mining\\\GalaxyMorphologicalExcavation.npy'))
fraction_training = 0.7
msk = np.random.rand(len(data1)) < fraction_training
data1_train = data1[msk]
data1_test = data1[~msk]


print('Number data galaxies:', len(data1))
print('Train fraction:', fraction_training)
print('Number of galaxies in training set:', len(data1_train))
print('Number of galaxies in testing set:', len(data1_test))
```

```
Number data galaxies: 780
Train fraction: 0.7
Number of galaxies in training set: 513
Number of galaxies in testing set: 267
```

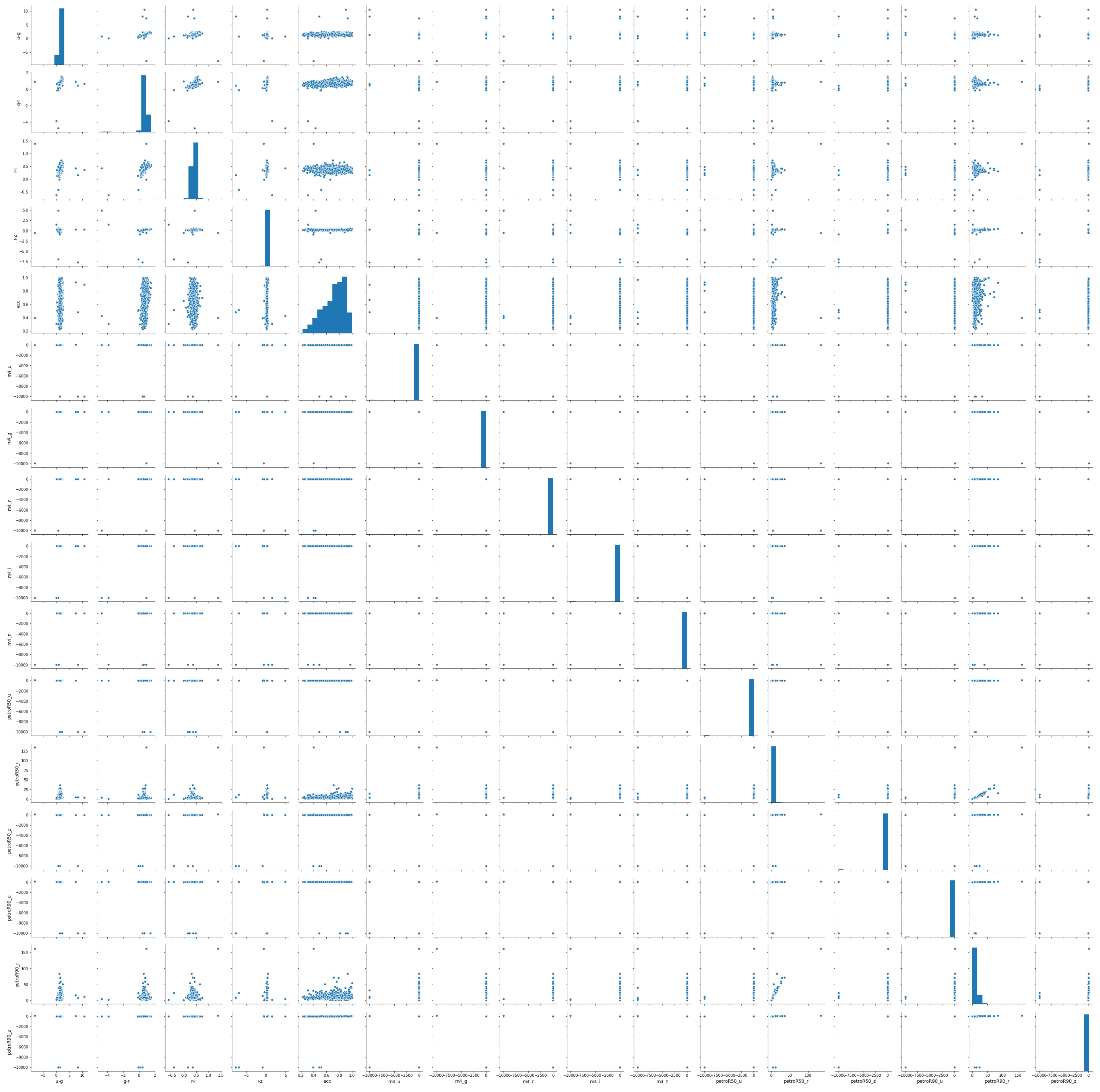
Data Correlation:

In [4]:
```
fig , ax = plt.subplots(figsize=(15,5))
sns.heatmap(data1.corr())
plt.show()
```



Data Distribution with repect to Redshift:

In [5]:
```
sns.pairplot(data1)
plt.show()
```



Feature Selection

In [6]:
```python

def generate_features_targets(data):

  targets = data['class']

  features = np.empty(shape=(len(data), 13))
  features[:, 0] = data['u-g']
  features[:, 1] = data['g-r']
  features[:, 2] = data['r-i']
  features[:, 3] = data['i-z']
  features[:, 4] = data['ecc']
  features[:, 5] = data['m4_u']
  features[:, 6] = data['m4_g']
  features[:, 7] = data['m4_r']
  features[:, 8] = data['m4_i']
  features[:, 9] = data['m4_z']


  # concentration in u filter
  features[:, 10] = data['petroR50_u']/data['petroR90_u']
  # concentration in r filter
  features[:, 11] = data['petroR50_r']/data['petroR90_r']
  # concentration in z filter
  features[:, 12] = data['petroR50_z']/data['petroR90_z']

  return features, targets


#  data = np.load('galaxy_catalogue.npy')

data1_features, data1_targets = generate_features_targets(data1)

print("Features shape:", data1_features.shape)
print("Targets shape:", data1_targets.shape)
```

```
Features shape: (780, 13)
Targets shape: (780,)
```

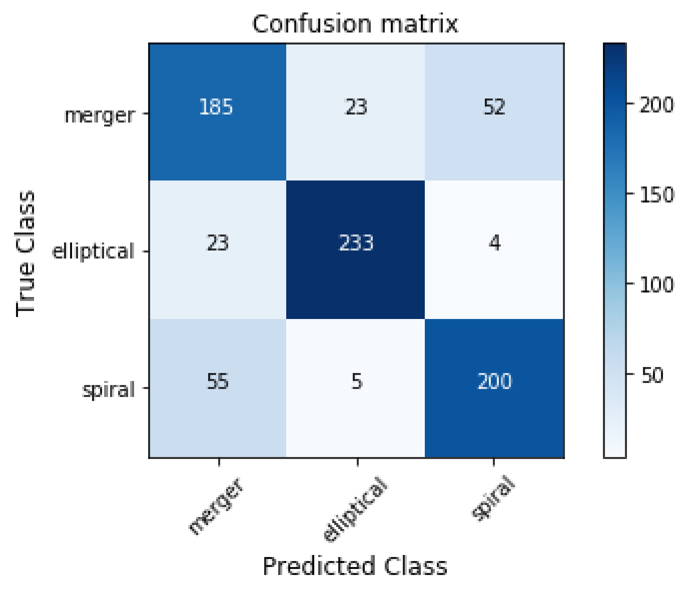
Morphological Classification of Galaxies using Decision Tree

In [7]:
```python
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
data1_train_features, data1_train_targets = generate_features_targets(data1_train)
data1_test_features, data1_test_targets = generate_features_targets(data1_test)
dtc.fit(data1_train_features, data1_train_targets)
data1_predictions = dtc.predict(data1_test_features)
```

In [8]:
```python
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, "{}".format(cm[i, j]),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True Class')
    plt.xlabel('Predicted Class')

def calculate_accuracy(predicted_classes, actual_classes):
  return sum(predicted_classes == actual_classes)/len(actual_classes)
```

In [9]:
```
data1_predicted = cross_val_predict(dtc, data1_features, data1_targets, cv=10)

# calculate the model score using your function
data1_model_score = calculate_accuracy(data1_predicted, data1_targets)
print("Our accuracy score:", data1_model_score)

# calculate the models confusion matrix using sklearns confusion_matrix function
data1_class_labels = list(set(data1_targets))
data1_model_cm_dtc = confusion_matrix(y_true=data1_targets, y_pred=data1_predicted, labels=data1_class_labels)

#classification report
from sklearn.metrics import classification_report
class_names = data1['class'].unique()
print("classification report of Decision Tree Classifier:\n")
print(classification_report(data1_targets, data1_predicted,target_names=class_names))

# Plot the confusion matrix using the provided functions.
plt.figure()
plot_confusion_matrix(data1_model_cm_dtc, classes=data1_class_labels, normalize=False)
plt.show()
```

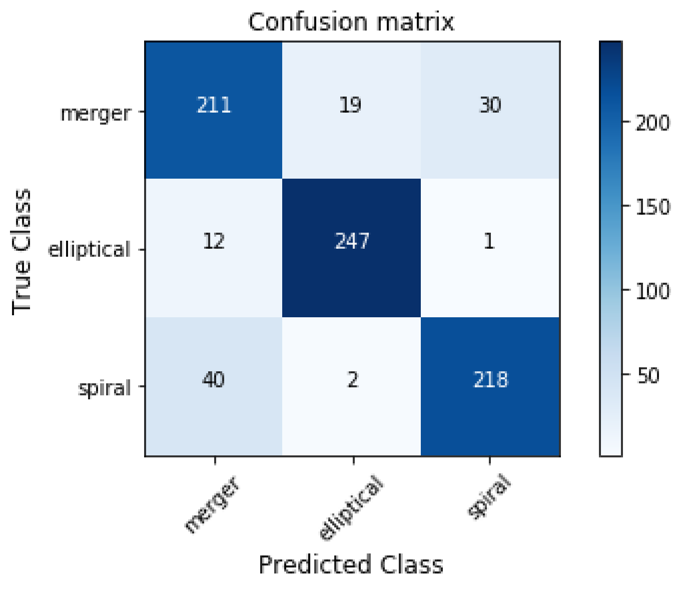
```
Our accuracy score: 0.7923076923076923
classification report of Decision Tree Classifier:

             precision    recall  f1-score   support

     merger       0.89      0.90      0.89       260
 elliptical       0.70      0.71      0.71       260
     spiral       0.78      0.77      0.78       260

avg / total       0.79      0.79      0.79       780

Confusion matrix, without normalization
[[185  23  52]
 [ 23 233   4]
 [ 55   5 200]]
```



Morphological Classification of Galaxies using Random Forest

In [156]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.ensemble import RandomForestClassifier


# complete this function to get predictions from a random forest classifier
def rf_predict_actual(data, n_estimators):
  # generate the features and targets
  features, targets = generate_features_targets(data)
  # instantiate a random forest classifier
  rfc = RandomForestClassifier(n_estimators=n_estimators)
  # get predictions using 10-fold cross validation with cross_val_predict
  predicted = cross_val_predict(rfc, features, targets, cv=10)
  # return the predictions and their actual classes
  return predicted, targets


# get the predicted and actual classes
data1_number_estimators = 50              # Number of trees
data1_predicted, data1_actual = rf_predict_actual(data1, data1_number_estimators)

# calculate the model score using your function
data1_accuracy = calculate_accuracy(data1_predicted, data1_actual)
print("Accuracy score:", data1_accuracy)

# calculate the models confusion matrix using sklearns confusion_matrix function
data1_class_labels = list(set(data1_actual))
data1_model_cm_rfc = confusion_matrix(y_true=data1_actual, y_pred=data1_predicted, labels=data1_class_labels)

# classification report of Random Forest Classifier
data1_class_names = data1['class'].unique()
print("classification report of Decision Tree Classifier:\n")
print(classification_report(data1_targets, data1_predicted,target_names=data1_class_names))

# plot the confusion matrix using the provided functions.
plt.figure()
plot_confusion_matrix(data1_model_cm_rfc, classes=data1_class_labels, normalize=False)
plt.show()
```

```
Accuracy score: 0.8666666666666667
classification report of Decision Tree Classifier:

             precision    recall  f1-score   support

     merger       0.92      0.95      0.94       260
 elliptical       0.80      0.81      0.81       260
     spiral       0.88      0.84      0.86       260

avg / total       0.87      0.87      0.87       780

Confusion matrix, without normalization
[[211  19  30]
 [ 12 247   1]
 [ 40   2 218]]
```



Dataset 2:

```
In [76]: df=pd.DataFrame(pd.read_csv('D:\\\ThirdFinalProject\\\Data_Mining\\\data.csv'))
         print("Dataset 2 contains {} entries and {} features".format(df.shape[0],df.shape[1]))
         print(df.info())
```

```
Dataset 2 contains 10000 entries and 48 features
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 48 columns):
objid          10000 non-null float64
ra             10000 non-null float64
dec            10000 non-null float64
u              10000 non-null float64
g              10000 non-null float64
r              10000 non-null float64
i              10000 non-null float64
z              10000 non-null float64
run            10000 non-null int64
rerun          10000 non-null int64
camcol         10000 non-null int64
field          10000 non-null int64
mCr4_u         10000 non-null float64
mCr4_g         10000 non-null float64
mCr4_r         10000 non-null float64
mCr4_i         10000 non-null float64
mCr4_z         10000 non-null float64
mCr4PSF_u      10000 non-null float64
mCr4PSF_g      10000 non-null float64
mCr4PSF_r      10000 non-null float64
mCr4PSF_i      10000 non-null float64
mCr4PSF_z      10000 non-null float64
petroR50_u     10000 non-null float64
petroR50_g     10000 non-null float64
petroR50_r     10000 non-null float64
petroR50_i     10000 non-null float64
petroR50_z     10000 non-null float64
petroR90_u     10000 non-null float64
petroR90_g     10000 non-null float64
petroR90_r     10000 non-null float64
petroR90_i     10000 non-null float64
petroR90_z     10000 non-null float64
type           10000 non-null int64
specobjid      10000 non-null float64
class          10000 non-null object
subClass       2453 non-null object
redshift       10000 non-null float64
plate          10000 non-null int64
mjd            10000 non-null int64
fiberid        10000 non-null int64
sourceType     10000 non-null object
nvote          10000 non-null int64
p_el           10000 non-null float64
p_cw           10000 non-null float64
p_acw          10000 non-null float64
p_edge         10000 non-null float64
p_dk           10000 non-null float64
p_mg           10000 non-null float64
dtypes: float64(36), int64(9), object(3)
memory usage: 3.7+ MB
None
```

Feature Description:

Objid, Specobjid and Fibreid: Primary ID for all SDSS objects and Unique ID for SPECOBJ table and SpecObjAll table respectively.

RA, Dec: Right Ascension and Declination.

u,g,r,i,z: SDSS color spectums or Filters.

Run: Imaging run.

Rerun: Rerun number.

Camcol: Camera column.

Field: Field number.

mcr4: Adaptive fourth moment of object.

mcr4PSF: Adaptive fourth moment of object for Petrusian Flux.

PetroR50: Radius containing 50% of Petrosian flux

PetroR90: Radius containing 90% of Petrosian flux

Type: Morphological type classification of the object.

Class: Spectroscopic classification of the object.

Subclass: Spectroscopic subclass of the object.

Redshift: Redshift of the object In physics, redshift happens when light or other electromagnetic radiation from an object is increased in wavelength, or shifted to the red end of the spectrum.

Each spectroscopic exposure employs a large, thin, circular metal plate that positions optical fibers via holes drilled at the locations of the images in the telescope focal plane. These fibers then feed into the spectrographs. Each plate has a unique serial number, which is called plate in views such as SpecObj in the CAS.

Plate: Plate number.

mjd: Modified Julian Date, used to indicate the date that a given piece of SDSS data (image or spectrum) was taken.

sourcetype: Spectroscopic type.

nvote: Total number of votes.

p_el, p_cw, p_acw, p_edge, p_dk, p_mg: Percentage of vote for each class.

Data Imputation:

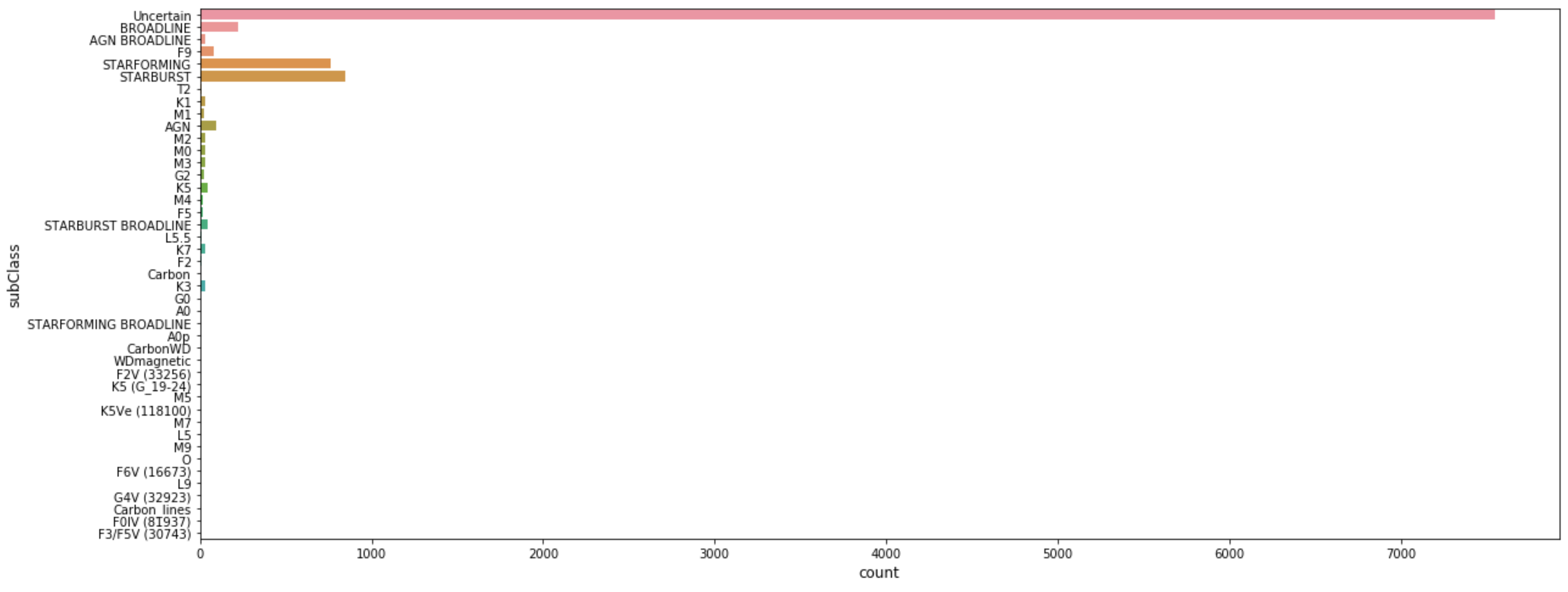
In [77]:
```
print('Only subclass feature has',df['subClass'].isna().sum()*100/10000,'% empty cells')
```

```
Only subclass feature has 75.47 % empty cells
```

and class distribution of this feature is like;

In [78]:
```
df['subClass'] = df['subClass'].replace(np.NaN,'Uncertain')
```

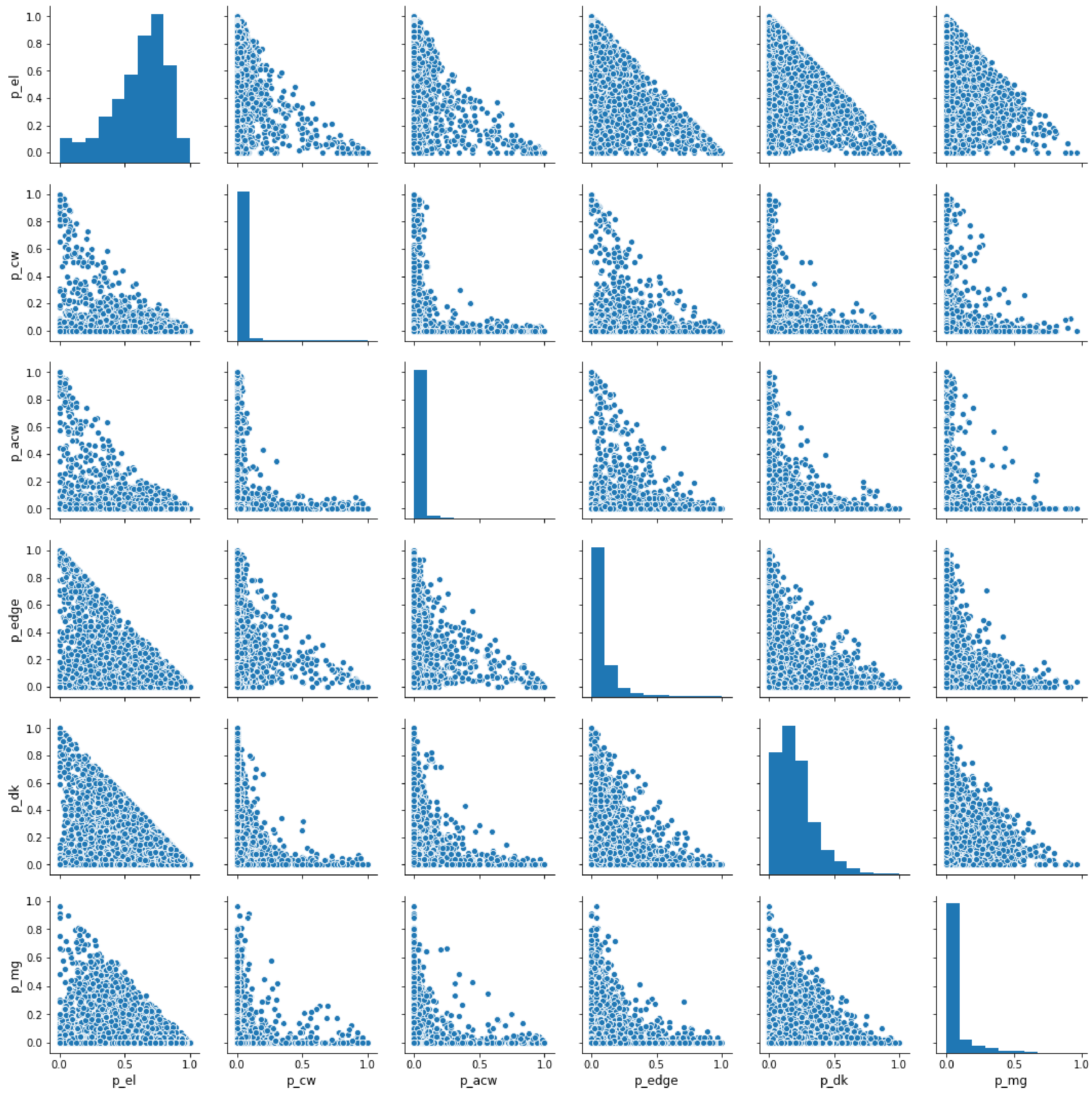
In [79]:
```
a4_dims = (20, 8)
fig, ax = plt.subplots(figsize=a4_dims)
sns.countplot(y= "subClass", data = df,palette="Paired")
plt.show()
```



Due to the heavy dominance of missing vallues, it is pretty clear that 'subClass' feature can not be used as a target untill and unless more data has been recorded.

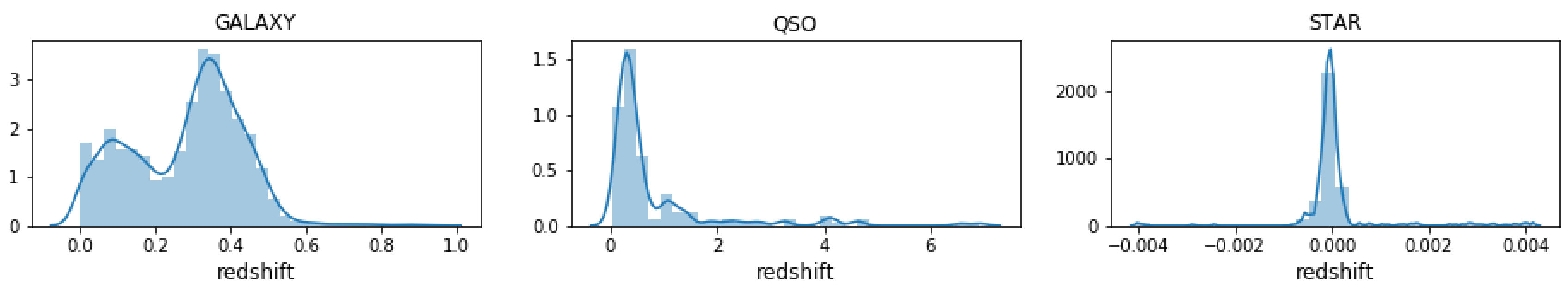
again, p_el, p_cw, p_acw, p_edge, p_dk, p_mg does not have a significance if left separated. Instead of keeping them separated, a new combined feature 'morphClass' is created with a thresholf of 50%.

```
In [80]: prob_col = ['p_el', 'p_cw', 'p_acw', 'p_edge', 'p_dk',
                 'p_mg']
         #fig, axes = plt.subplots(figsize=(4, 4))
         sns.pairplot(df[prob_col])
         #df[prob_col]
         def tune_wrt_class(col,data):
             c = len(col)
             for i in col:
                 data.loc[data[i] >= 0.5, i ] = c**3
                 data.loc[data[i] < 0.5, i ] = 0
                 c-=1
         tune_wrt_class(prob_col,df)

         #print(df[prob_col])

         temp = pd.DataFrame(df[prob_col])

         df['morphClass'] = temp.sum(axis = 1, skipna = True)


         df['morphClass'] = df['morphClass'].map({224:'el_dk', 217:'el_mg', 0:'uncertain', 1:'mg', 8:'dk', 27:'edge', 64:
```

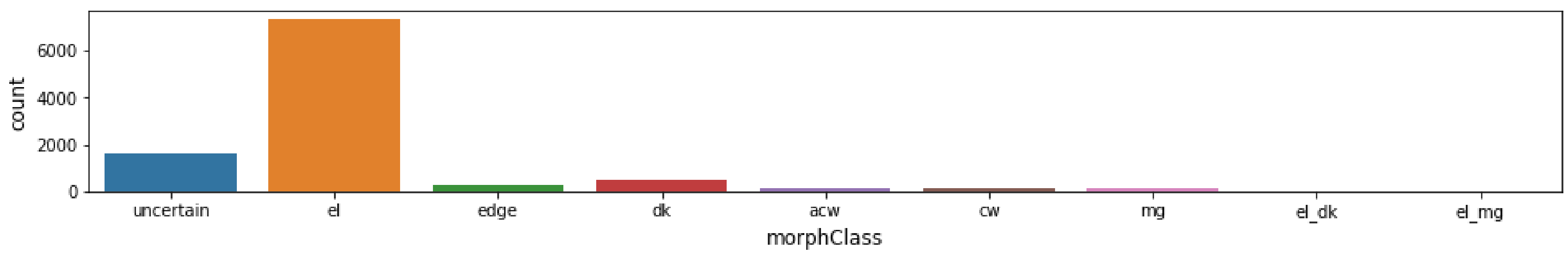


Data Visualizaion:

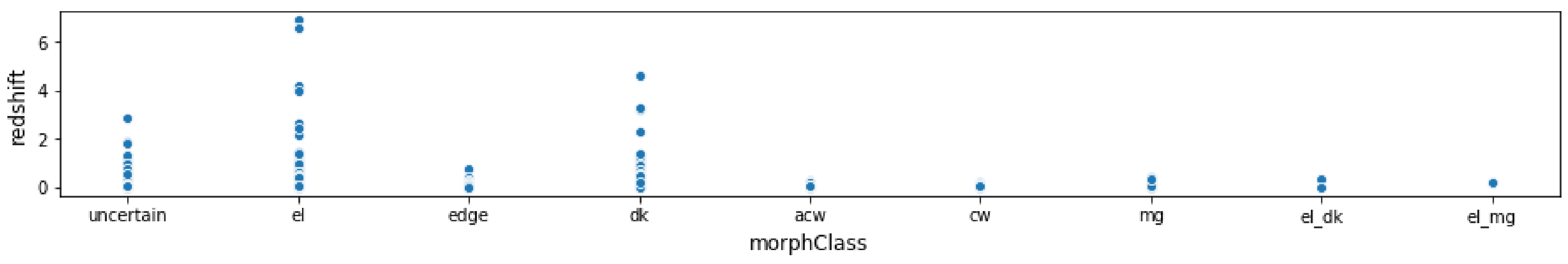
```
In [81]: #df['subClass'].unique()
         col = 'class'
         fig, axes = plt.subplots(nrows=1, ncols=3,figsize=(16, 2))

         j = 2
         colval = df[col].unique()
         for i in [i for i in df[col].unique()]:
             ax = sns.distplot(df[df[col]==colval[j] ].redshift, bins = 30, ax = axes[j])
             ax.set_title(colval[j])
             j-=1
```



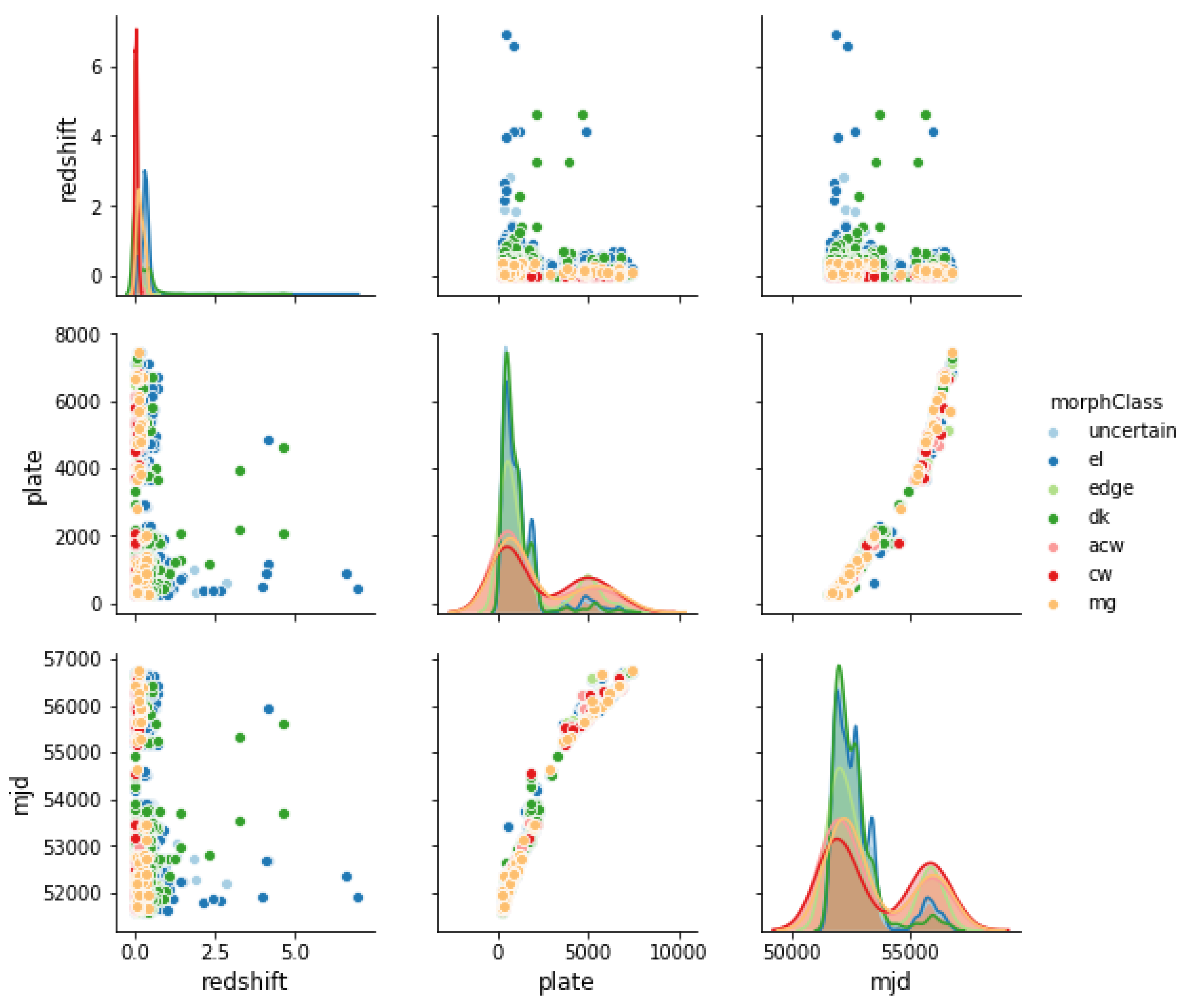
```
In [82]: fig, axes = plt.subplots(nrows=1, ncols=1,figsize=(16, 2))
         sns.countplot(df['morphClass'])
         #sns.scatterplot(df['morphClass'],df['redshift'])
         plt.show()
```



```
In [83]: fig, axes = plt.subplots(nrows=1, ncols=1,figsize=(16, 2))
         sns.scatterplot(df['morphClass'],df['redshift'])
         plt.show()
```

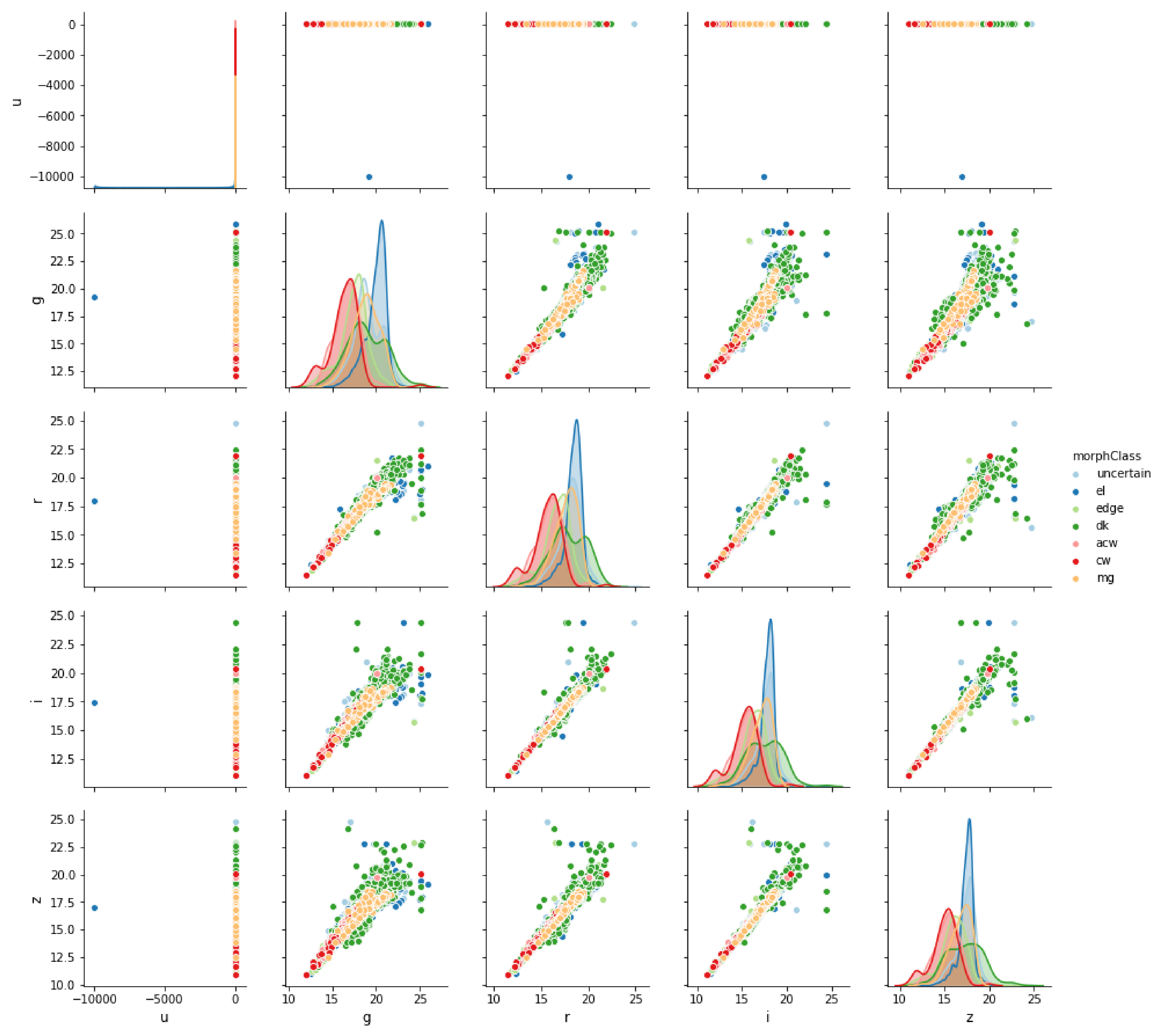


```
In [126]: plt.figure(figsize=(18,2))
          n = sns.pairplot(df[["redshift","plate","mjd","morphClass"]],hue="morphClass",palette='Paired')
          plt.show()

          <Figure size 1296x144 with 0 Axes>
```



In [123]:
```
plt.figure(figsize=(18,18))
n1 = sns.pairplot(df[["u","g","r","i","z","morphClass",]],hue="morphClass",palette="Paired")#,hue_order=["QSO","
plt.show()
```

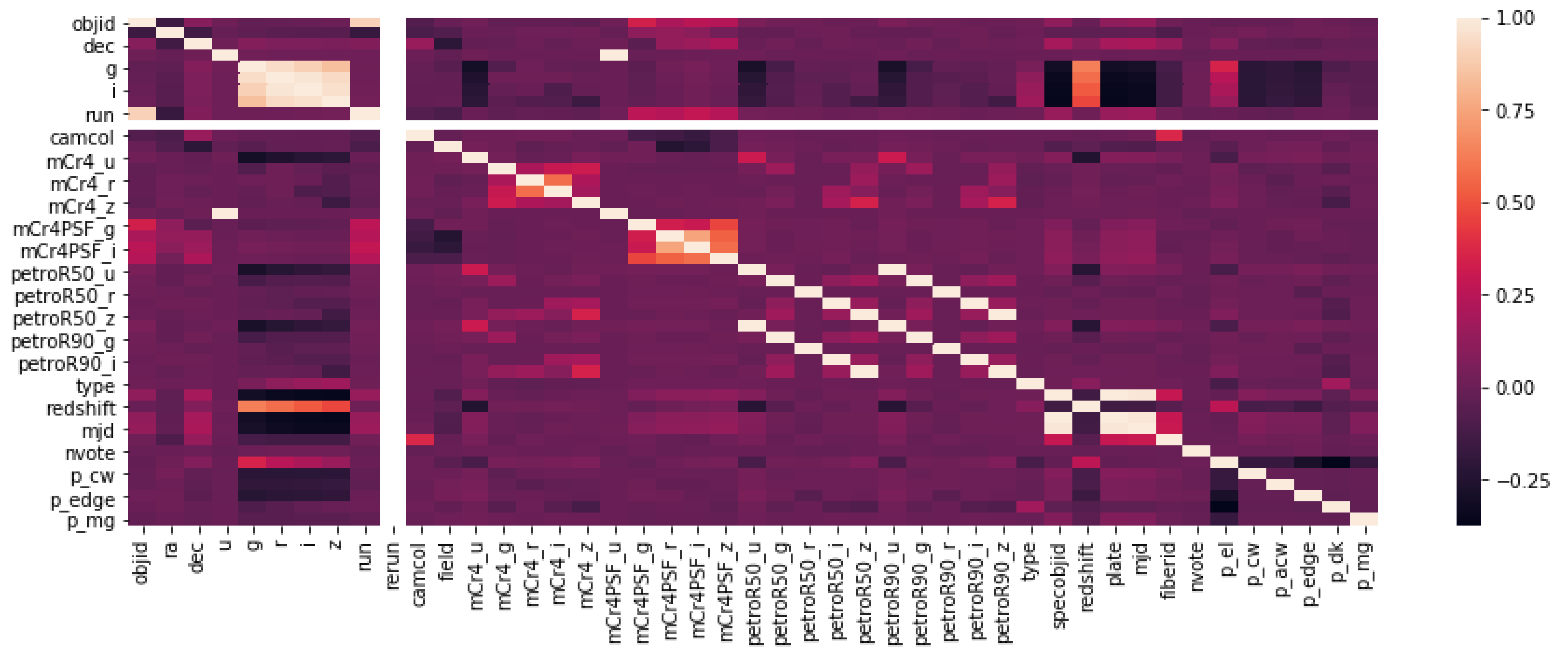
```
<Figure size 1296x1296 with 0 Axes>
```



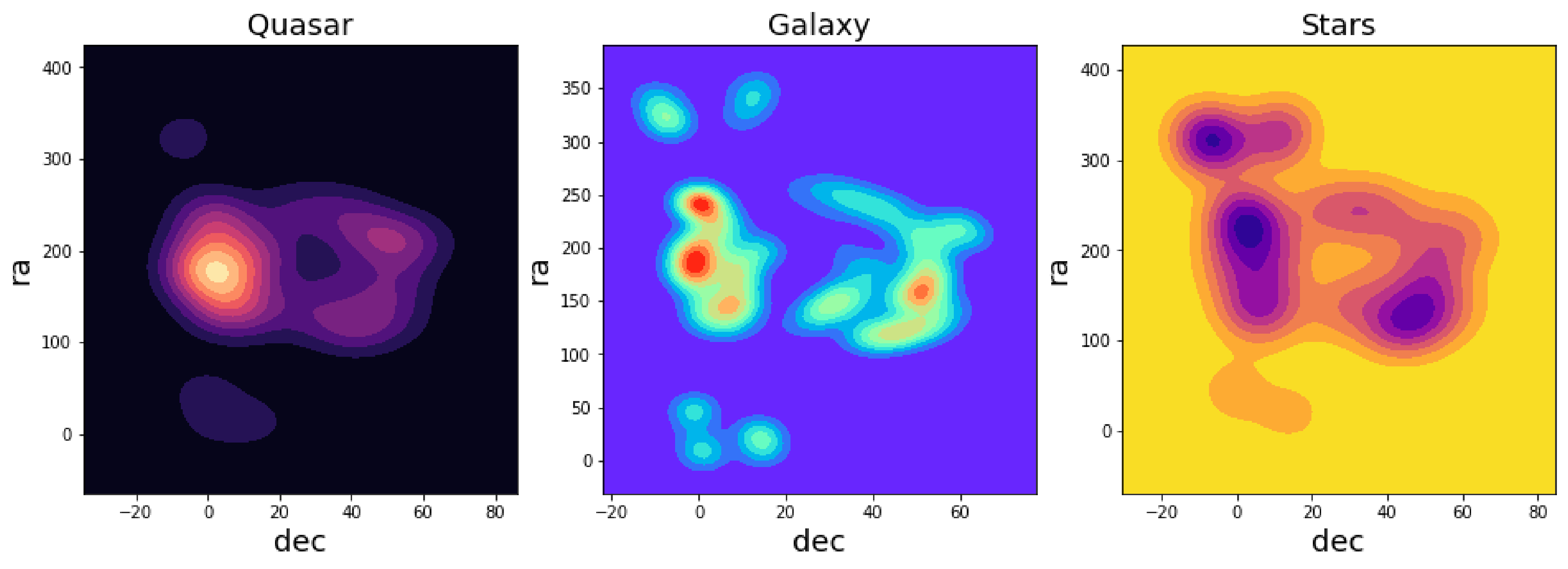
Data Correlation

In [85]:
```
fig , ax = plt.subplots(figsize=(15,5))
sns.heatmap(df.corr())
```

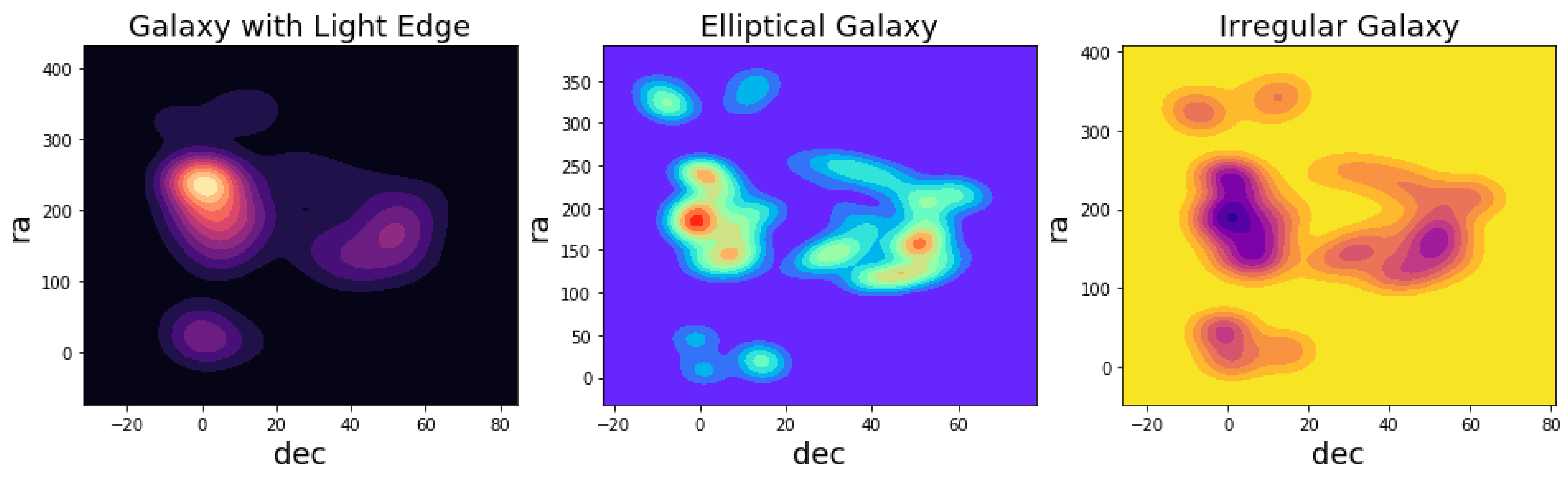
Out[85]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1cebf1aec18>
```



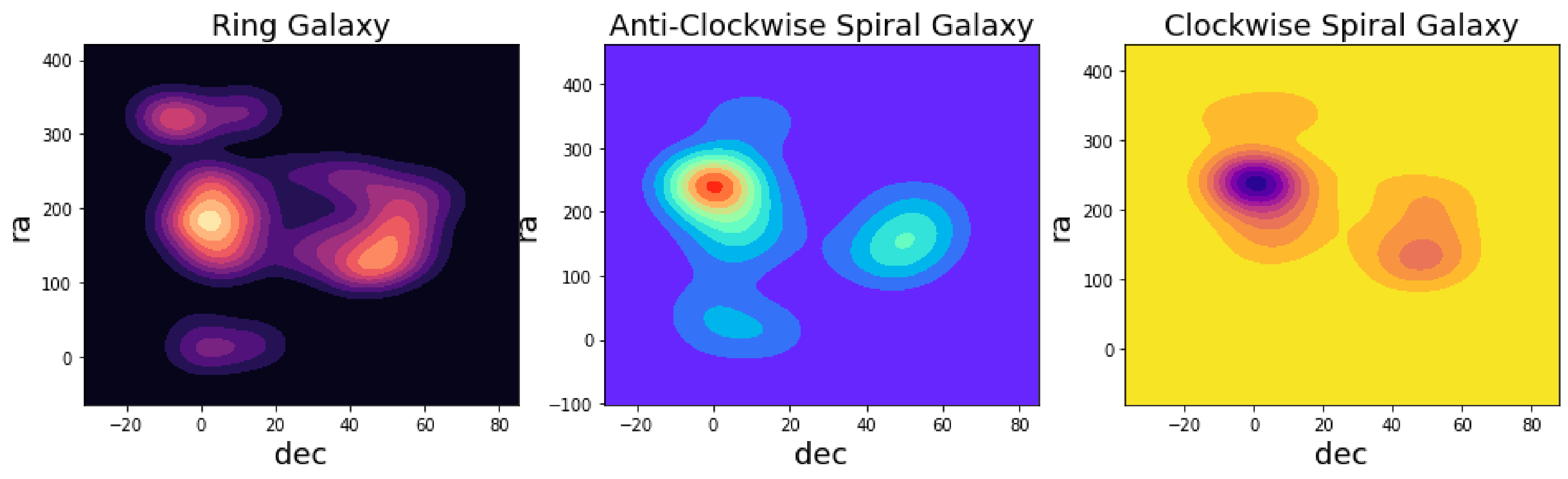
In [86]:
```
f,ax = plt.subplots(1,3,figsize=(16,5))
#k = sns.scatterplot("dec","ra",hue="class",data=Data,cmap="magma")
Stars = df.loc[df["class"] == "STAR"]
Galaxy = df.loc[df["class"] == "GALAXY"]
Qso = df.loc[df["class"] == "QSO"]
k = sns.kdeplot(Qso.dec,Qso.ra,shade=True,shade_lowest=True,cmap="magma",ax=ax[0])
k = sns.kdeplot(Galaxy.dec,Galaxy.ra,shade=True,shade_lowest=True,cmap="rainbow",ax=ax[1])
k = sns.kdeplot(Stars.dec,Stars.ra,shade=True,shade_lowest=True,cmap="plasma_r",ax=ax[2])
ax[0].set_xlabel("dec", fontsize=18)
ax[1].set_xlabel("dec", fontsize=18)
ax[2].set_xlabel("dec", fontsize=18)
ax[0].set_ylabel("ra", fontsize=18)
ax[1].set_ylabel("ra", fontsize=18)
ax[2].set_ylabel("ra", fontsize=18)
ax[0].set_title("Quasar", fontsize=18)
ax[1].set_title("Galaxy", fontsize=18)
ax[2].set_title("Stars", fontsize=18)
#k = sns.kdeplot(Data["dec"],Data["ra"],shade=True,shade_lowest=False,cmap="magma",ax=ax)
plt.show()
```



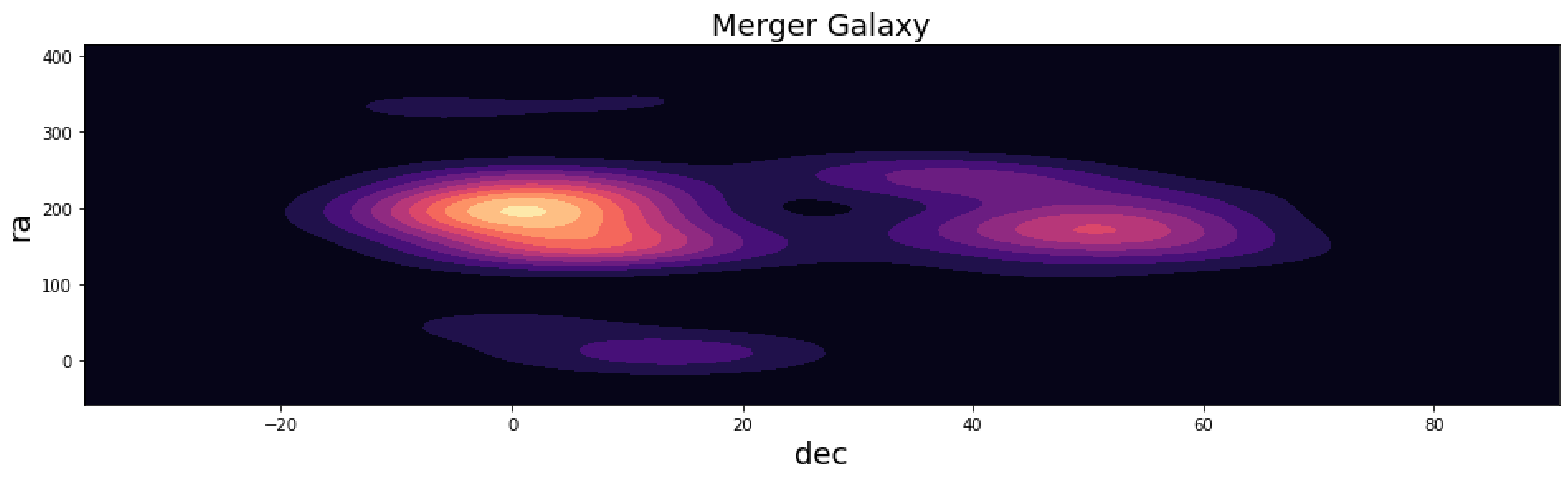
In [127]:
```
f,ax = plt.subplots(1,3,figsize=(16,4))
#k = sns.scatterplot("dec","ra",hue="class",data=Data,cmap="magma")
undefined = df.loc[df["morphClass"] == "uncertain"]
el = df.loc[df["morphClass"] == "el"]
edge = df.loc[df["morphClass"] == "edge"]


k = sns.kdeplot(edge.dec,edge.ra,shade=True,shade_lowest=True,cmap="magma",ax=ax[0])
k = sns.kdeplot(el.dec,el.ra,shade=True,shade_lowest=True,cmap="rainbow",ax=ax[1])
k = sns.kdeplot(undefined.dec,undefined.ra,shade=True,shade_lowest=True,cmap="plasma_r",ax=ax[2])

ax[0].set_xlabel("dec", fontsize=18)
ax[1].set_xlabel("dec", fontsize=18)
ax[2].set_xlabel("dec", fontsize=18)
ax[0].set_ylabel("ra", fontsize=18)
ax[1].set_ylabel("ra", fontsize=18)
ax[2].set_ylabel("ra", fontsize=18)
ax[0].set_title("Galaxy with Light Edge", fontsize=18)
ax[1].set_title("Elliptical Galaxy", fontsize=18)
ax[2].set_title("Irregular Galaxy", fontsize=18)
#k = sns.kdeplot(Data["dec"],Data["ra"],shade=True,shade_lowest=False,cmap="magma",ax=ax)
plt.show()
```



In [88]:
```
f,ax = plt.subplots(1,3,figsize=(16,4))
#k = sns.scatterplot("dec","ra",hue="class",data=Data,cmap="magma")
dk = df.loc[df["morphClass"] == "dk"]
acw = df.loc[df["morphClass"] == "acw"]
cw = df.loc[df["morphClass"] == "cw"]


k = sns.kdeplot(dk.dec,dk.ra,shade=True,shade_lowest=True,cmap="magma",ax=ax[0])
k = sns.kdeplot(acw.dec,acw.ra,shade=True,shade_lowest=True,cmap="rainbow",ax=ax[1])
k = sns.kdeplot(cw.dec,cw.ra,shade=True,shade_lowest=True,cmap="plasma_r",ax=ax[2])

ax[0].set_xlabel("dec", fontsize=18)
ax[1].set_xlabel("dec", fontsize=18)
ax[2].set_xlabel("dec", fontsize=18)
ax[0].set_ylabel("ra", fontsize=18)
ax[1].set_ylabel("ra", fontsize=18)
ax[2].set_ylabel("ra", fontsize=18)
ax[0].set_title("Ring Galaxy", fontsize=18)
ax[1].set_title("Anti-Clockwise Spiral Galaxy", fontsize=18)
ax[2].set_title("Clockwise Spiral Galaxy", fontsize=18)
#k = sns.kdeplot(Data["dec"],Data["ra"],shade=True,shade_lowest=False,cmap="magma",ax=ax)
plt.show()
```



In [89]:
```
f,ax = plt.subplots(1,1,figsize=(16,4))
#k = sns.scatterplot("dec","ra",hue="class",data=Data,cmap="magma")
mg = df.loc[df["morphClass"] == "mg"]


k = sns.kdeplot(mg.dec,mg.ra,shade=True,shade_lowest=True,cmap="magma")

ax.set_xlabel("dec", fontsize=18)
ax.set_ylabel("ra", fontsize=18)
ax.set_title("Merger Galaxy", fontsize=18)
plt.show()
```



Feature Elimination:

Dataset-2 contains many unnecessary features which are of no use to us. Those unnecessary features are removed. Elliptical Merger Galaxy and Elliptical Disk galaxy is a rare among the rarest phenomena. Removign this record to balance the dataset some more.

In [92]:
```
df.drop(["objid", "rerun","fiberid","specobjid","nvote"],axis=1, inplace=True)
df.drop(prob_col,axis=1,inplace=True)
df = df[df['morphClass'] != 'el_dk']
df = df[df['morphClass'] != 'el_mg']
```

Feature Extraction and Data Splitting:

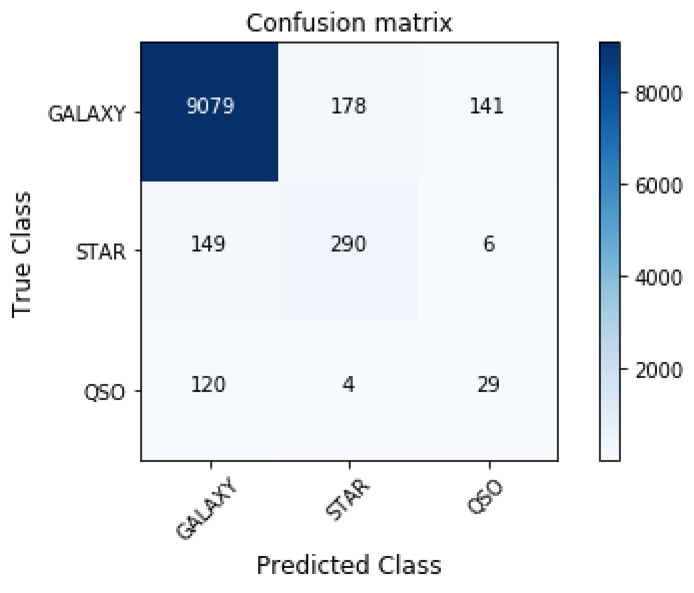
```
In [143]: df['Con_U'] = df['petroR50_u']/df['petroR90_u']
df['Con_G'] = df['petroR50_g']/df['petroR90_g']
df['Con_R'] = df['petroR50_r']/df['petroR90_r']
df['Con_I'] = df['petroR50_i']/df['petroR90_i']
df['Con_Z'] = df['petroR50_z']/df['petroR90_z']


target_col = ['class','morphClass']
feature_col = ['u','g','r','i','z','mCr4_u', 'mCr4_g', 'mCr4_r', 'mCr4_i', 'mCr4_z','Con_U','Con_G','Con_R','Con_
sdss = df[feature_col+target_col]

#sdss_feature = df[feature_col]
#sdss_target_class = df['class']
#sdss_target_morphClass = df['morphClass']

#for i in sdss_target_col:
#    sdss[i].fillna("Uncertain", inplace = True)

fraction_training = 0.7
msk = np.random.rand(len(sdss)) < fraction_training
sdss_train = sdss[msk]
sdss_test = sdss[~msk]


print('Number data galaxies:', len(sdss))
print('Train fraction:', fraction_training)
print('Number of galaxies in training set:', len(sdss_train))
print('Number of galaxies in testing set:', len(sdss_test))

#feature and target separation
sdss_feature = sdss[feature_col]
sdss_target_class = sdss['class']
sdss_target_morphClass = sdss['morphClass']
sdss_train_feature = sdss_train[feature_col]
sdss_train_target_class = sdss_train['class']
sdss_train_target_morphClass = sdss_train['morphClass']
sdss_test_feature = sdss_test[feature_col]
sdss_test_target_class = sdss_test['class']
sdss_test_target_morphClass = sdss_test['morphClass']

sdss_classtarget = sdss[feature_col]
sdss_classtarget['class'] = sdss['class']
sdss_morphtarget = sdss[feature_col]
sdss_morphtarget['morphClass'] = sdss['morphClass']
```

```
Number data galaxies: 9996
Train fraction: 0.7
Number of galaxies in training set: 7064
Number of galaxies in testing set: 2932
```

Spectroscopic Classification using Decision Tree Classifier:

In [151]:
```
dtc = DecisionTreeClassifier()
dtc.fit(sdss_train_feature, sdss_train_target_class)
sdss_predictions = dtc.predict(sdss_test_feature)

sdss_predicted = cross_val_predict(dtc, sdss_feature, sdss_target_class, cv=10)

# calculate the model score using your function
sdss_model_score = calculate_accuracy(sdss_predicted, sdss_target_class)
print("Our accuracy score:", sdss_model_score)

# calculate the models confusion matrix using sklearns confusion_matrix function
sdss_class_labels = list(set(sdss_target_class))
sdss_model_cm_dtc = confusion_matrix(y_true=sdss_target_class, y_pred=sdss_predicted, labels=sdss_class_labels)

#classification report
#from sklearn.metrics import classification_report
class_names = sdss['class'].unique()
print("classification report of Decision Tree Classifier:\n")
print(classification_report(sdss_target_class, sdss_predicted,target_names=class_names))

# Plot the confusion matrix using the provided functions.
plt.figure()
plot_confusion_matrix(sdss_model_cm_dtc, classes=sdss_class_labels, normalize=False)
plt.show()
```

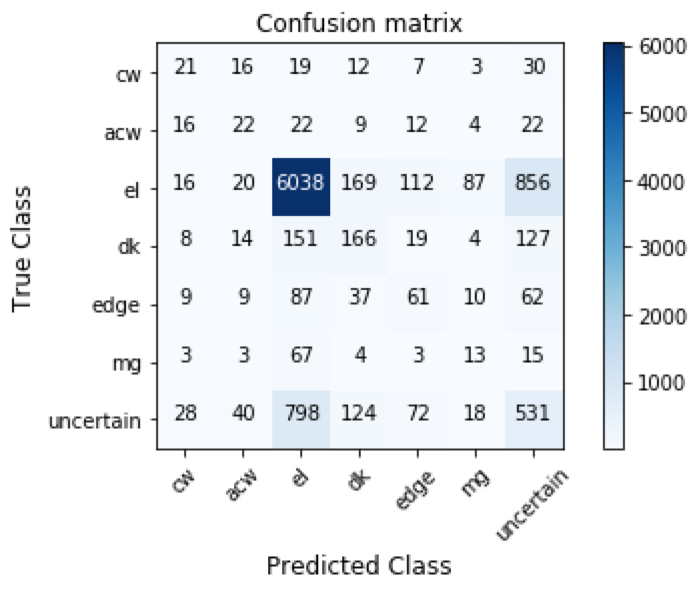
```
Our accuracy score: 0.9401760704281713
classification report of Decision Tree Classifier:

             precision    recall  f1-score   support

     GALAXY       0.97      0.97      0.97      9398
        QSO       0.16      0.19      0.18       153
       STAR       0.61      0.65      0.63       445

avg / total       0.94      0.94      0.94      9996

Confusion matrix, without normalization
[[9079  178  141]
 [ 149  290    6]
 [ 120    4   29]]
```



Morphological Classification using Decision Tree Classifier:

In [153]:
```
dtc = DecisionTreeClassifier()
dtc.fit(sdss_train_feature, sdss_train_target_morphClass)
sdss_predictions = dtc.predict(sdss_test_feature)

sdss_predicted = cross_val_predict(dtc, sdss_feature, sdss_target_morphClass, cv=10)

# calculate the model score using your function
sdss_model_score = calculate_accuracy(sdss_predicted, sdss_target_morphClass)
print("Our accuracy score:", sdss_model_score)

# calculate the models confusion matrix using sklearns confusion_matrix function
sdss_class_labels = list(set(sdss_target_morphClass))
sdss_model_cm_dtc = confusion_matrix(y_true=sdss_target_morphClass, y_pred=sdss_predicted, labels=sdss_class_labe

#classification report
#from sklearn.metrics import classification_report
class_names = sdss['morphClass'].unique()
print("classification report of Decision Tree Classifier:\n")
print(classification_report(sdss_target_morphClass, sdss_predicted,target_names=class_names))

# Plot the confusion matrix using the provided functions.
plt.figure()
plot_confusion_matrix(sdss_model_cm_dtc, classes=sdss_class_labels, normalize=False)
plt.show()
```

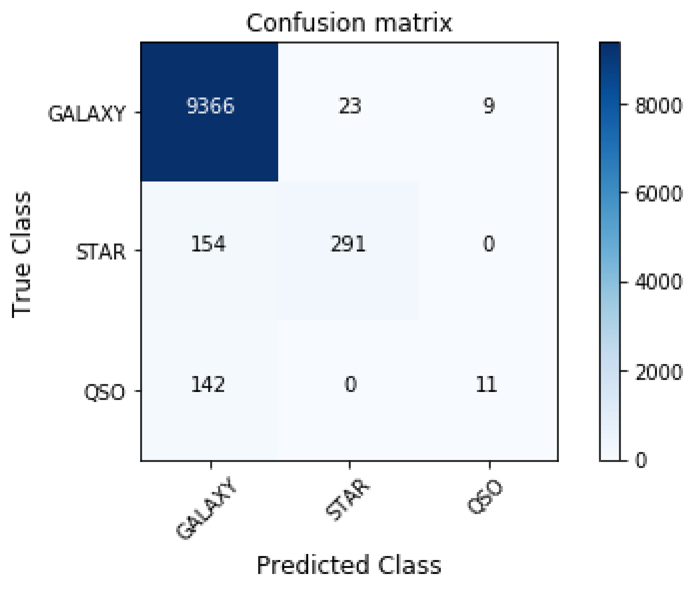
```
Our accuracy score: 0.6854741896758704
classification report of Decision Tree Classifier:

             precision    recall  f1-score   support

  uncertain       0.18      0.21      0.19       107
         el       0.21      0.19      0.20       108
       edge       0.32      0.34      0.33       489
         dk       0.21      0.22      0.22       275
        acw       0.84      0.83      0.83      7298
         cw       0.09      0.12      0.11       108
         mg       0.32      0.33      0.33      1611

avg / total       0.69      0.69      0.69      9996

Confusion matrix, without normalization
[[  21   16   19   12    7    3   30]
 [  16   22   22    9   12    4   22]
 [  16   20 6038  169  112   87  856]
 [   8   14  151  166   19    4  127]
 [   9    9   87   37   61   10   62]
 [   3    3   67    4    3   13   15]
 [  28   40  798  124   72   18  531]]
```



Spectroscopic Classification using Random Forest Classifier

In [181]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.ensemble import RandomForestClassifier


# complete this function to get predictions from a random forest classifier
def rf_predict_actual(data, n_estimators):
  # generate the features and targets
    features = sdss[feature_col]
    targets = sdss['class']
  # instantiate a random forest classifier
    rfc = RandomForestClassifier(n_estimators=n_estimators)
  # get predictions using 10-fold cross validation with cross_val_predict
    predicted = cross_val_predict(rfc, features, targets, cv=10)
  # return the predictions and their actual classes
    return predicted, targets


# get the predicted and actual classes
sdss_number_estimators = 50              # Number of trees
sdss_predicted, sdss_actual = rf_predict_actual(sdss_classtarget, sdss_number_estimators)

# calculate the model score using your function
sdss_accuracy = calculate_accuracy(sdss_predicted, sdss_actual)
print("Accuracy score:", sdss_accuracy)

# calculate the models confusion matrix using sklearns confusion_matrix function
sdss_class_labels = list(set(sdss_actual))
sdss_model_cm_rfc = confusion_matrix(y_true=sdss_actual, y_pred=sdss_predicted, labels=sdss_class_labels)

# classification report of Random Forest Classifier
sdss_class_names = sdss_classtarget['class'].unique()
print("classification report of Decision Tree Classifier:\n")
print(classification_report(sdss_target_class, sdss_predicted,target_names=sdss_class_names))

# plot the confusion matrix using the provided functions.
plt.figure()
plot_confusion_matrix(sdss_model_cm_rfc, classes=sdss_class_labels, normalize=False)
plt.show()
```

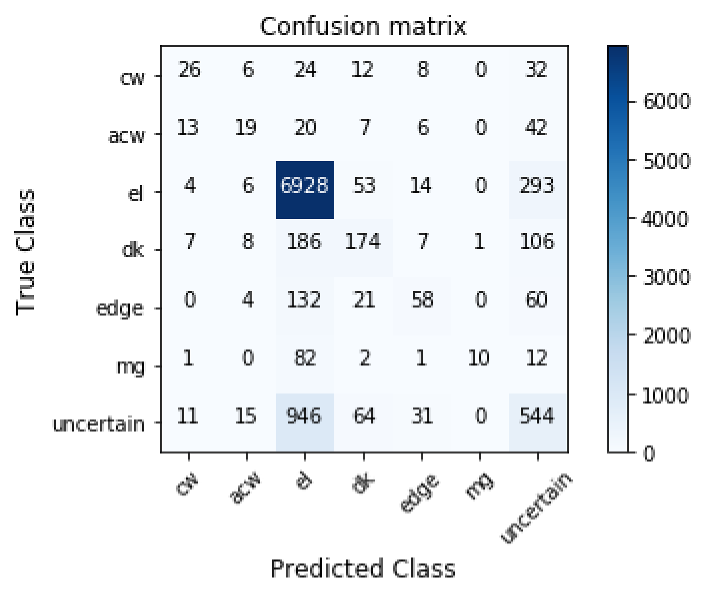
```
Accuracy score: 0.9671868747499
classification report of Decision Tree Classifier:

             precision    recall  f1-score   support

     GALAXY       0.97      1.00      0.98      9398
        QSO       0.55      0.07      0.13       153
       STAR       0.93      0.65      0.77       445

avg / total       0.96      0.97      0.96      9996

Confusion matrix, without normalization
[[9366   23    9]
 [ 154  291    0]
 [ 142    0   11]]
```



Morphological Classification using Random Forest Classifier:

In [183]:
```python
from sklearn.model_selection import cross_val_predict
from sklearn.ensemble import RandomForestClassifier


# complete this function to get predictions from a random forest classifier
def rf_predict_actual(data, n_estimators):
  # generate the features and targets
    features = sdss[feature_col]
    targets = sdss['morphClass']
  # instantiate a random forest classifier
    rfc = RandomForestClassifier(n_estimators=n_estimators)
  # get predictions using 10-fold cross validation with cross_val_predict
    predicted = cross_val_predict(rfc, features, targets, cv=10)
  # return the predictions and their actual classes
    return predicted, targets


# get the predicted and actual classes
sdss_number_estimators = 50              # Number of trees
sdss_predicted, sdss_actual = rf_predict_actual(sdss_morphtarget, sdss_number_estimators)

# calculate the model score using your function
sdss_accuracy = calculate_accuracy(sdss_predicted, sdss_actual)
print("Accuracy score:", sdss_accuracy)

# calculate the models confusion matrix using sklearns confusion_matrix function
sdss_class_labels = list(set(sdss_actual))
sdss_model_cm_rfc = confusion_matrix(y_true=sdss_actual, y_pred=sdss_predicted, labels=sdss_class_labels)

# classification report of Random Forest Classifier
sdss_class_names = sdss_morphtarget['morphClass'].unique()
print("classification report of Decision Tree Classifier:\n")
print(classification_report(sdss_target_morphClass, sdss_predicted,target_names=sdss_class_names))

# plot the confusion matrix using the provided functions.
plt.figure()
plot_confusion_matrix(sdss_model_cm_rfc, classes=sdss_class_labels, normalize=False)
plt.show()
```

```
Accuracy score: 0.7762104841936774
classification report of Decision Tree Classifier:

             precision    recall  f1-score   support

  uncertain       0.33      0.18      0.23       107
         el       0.42      0.24      0.31       108
       edge       0.52      0.36      0.42       489
         dk       0.46      0.21      0.29       275
        acw       0.83      0.95      0.89      7298
         cw       0.91      0.09      0.17       108
         mg       0.50      0.34      0.40      1611

avg / total       0.74      0.78      0.75      9996

Confusion matrix, without normalization
[[  26    6   24   12    8    0   32]
 [  13   19   20    7    6    0   42]
 [   4    6 6928   53   14    0  293]
 [   7    8  186  174    7    1  106]
 [   0    4  132   21   58    0   60]
 [   1    0   82    2    1   10   12]
 [  11   15  946   64   31    0  544]]
```



Dataset 3:

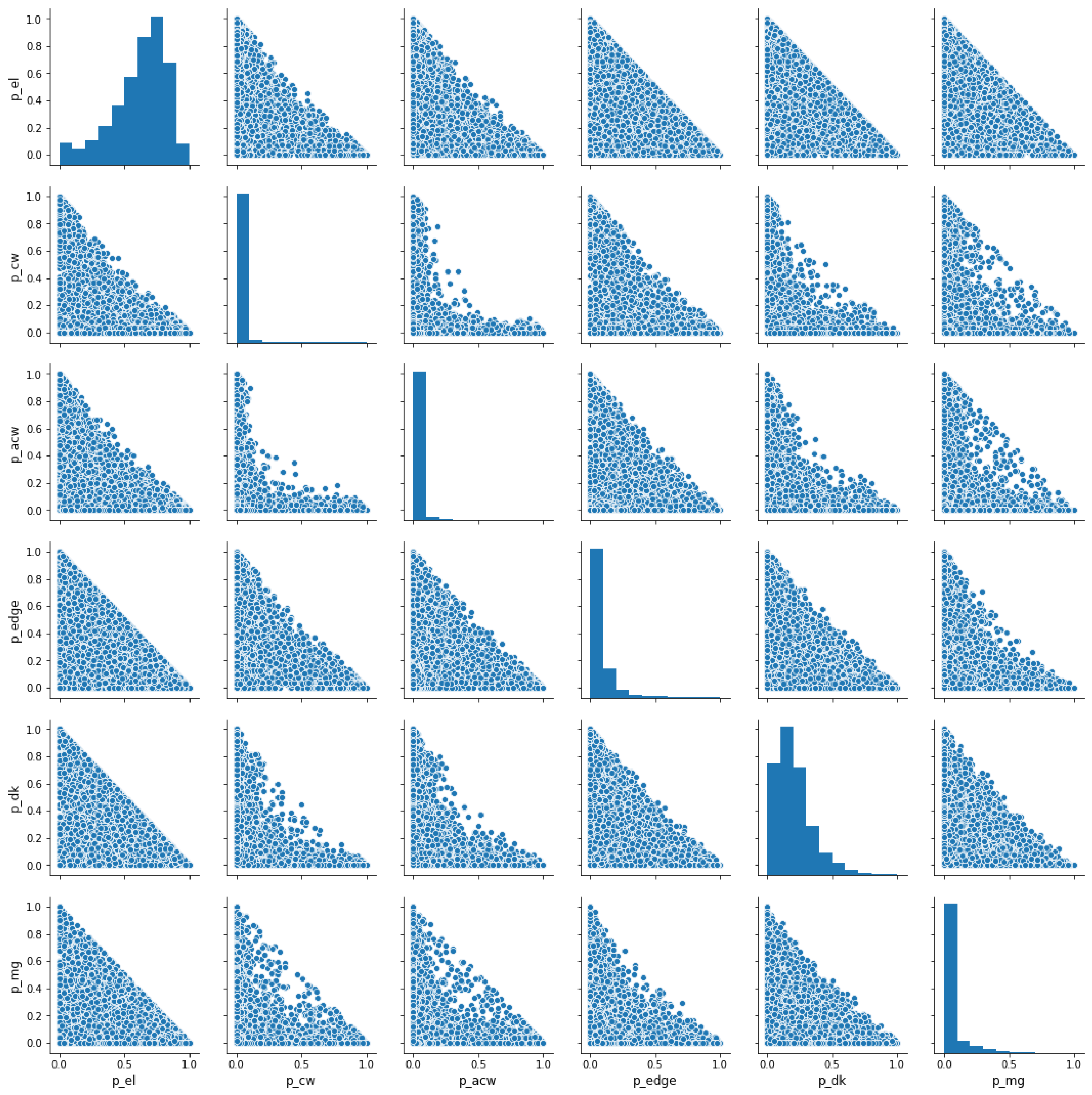
Now the final dataset will be used to check whether the accuracies are improving or not.

In [193]:
```
df=pd.DataFrame(pd.read_csv('D:///ThirdFinalProject///Data_Mining/thirddata.csv'))
print("Dataset 2 contains {} entries and {} features".format(df.shape[0],df.shape[1]))
print(df.info())
```

```
Dataset 2 contains 100000 entries and 46 features
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 46 columns):
ra            100000 non-null float64
dec           100000 non-null float64
u             100000 non-null float64
g             100000 non-null float64
r             100000 non-null float64
i             100000 non-null float64
z             100000 non-null float64
run           100000 non-null int64
rerun         100000 non-null int64
camcol        100000 non-null int64
field         100000 non-null int64
mCr4_u        100000 non-null float64
mCr4_g        100000 non-null float64
mCr4_r        100000 non-null float64
mCr4_i        100000 non-null float64
mCr4_z        100000 non-null float64
mCr4PSF_u     100000 non-null float64
mCr4PSF_g     100000 non-null float64
mCr4PSF_r     100000 non-null float64
mCr4PSF_i     100000 non-null float64
mCr4PSF_z     100000 non-null float64
petroR50_u    100000 non-null float64
petroR50_g    100000 non-null float64
petroR50_r    100000 non-null float64
petroR50_i    100000 non-null float64
petroR50_z    100000 non-null float64
petroR90_u    100000 non-null float64
petroR90_g    100000 non-null float64
petroR90_r    100000 non-null float64
petroR90_i    100000 non-null float64
petroR90_z    100000 non-null float64
type          100000 non-null int64
class         100000 non-null object
subClass      22966 non-null object
redshift      100000 non-null float64
plate         100000 non-null int64
mjd           100000 non-null int64
fiberid       100000 non-null int64
sourceType    100000 non-null object
nvote         100000 non-null int64
p_el          100000 non-null float64
p_cw          100000 non-null float64
p_acw         100000 non-null float64
p_edge        100000 non-null float64
p_dk          100000 non-null float64
p_mg          100000 non-null float64
dtypes: float64(34), int64(9), object(3)
memory usage: 35.1+ MB
None
```

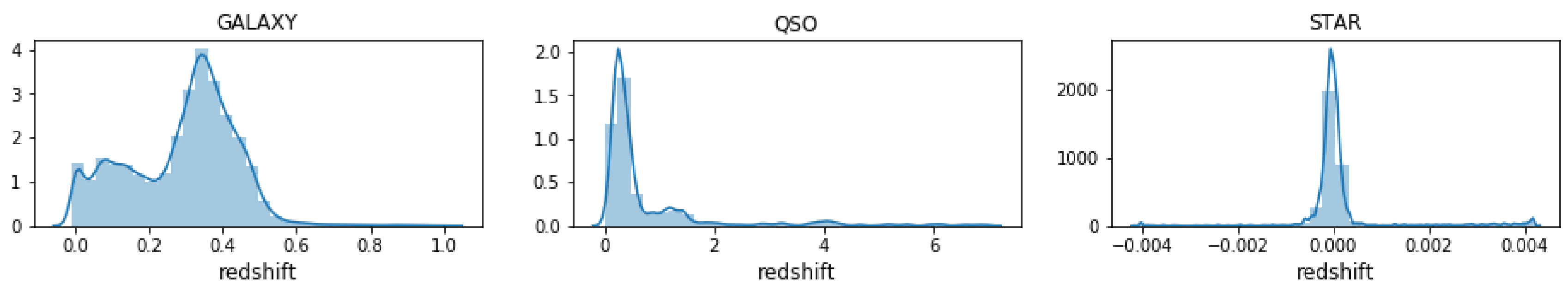
Creating morphClass feature:

In [195]:
```
prob_col = ['p_el', 'p_cw', 'p_acw', 'p_edge', 'p_dk',
         'p_mg']
#fig, axes = plt.subplots(figsize=(4, 4))
sns.pairplot(df[prob_col])
#df[prob_col]
def tune_wrt_class(col,data):
    c = len(col)
    for i in col:
        data.loc[data[i] >= 0.5, i ] = c**3
        data.loc[data[i] < 0.5, i ] = 0
        c-=1
tune_wrt_class(prob_col,df)

#print(df[prob_col])

temp = pd.DataFrame(df[prob_col])

df['morphClass'] = temp.sum(axis = 1, skipna = True)


df['morphClass'] = df['morphClass'].map({224:'el_dk', 217:'el_mg', 0:'uncertain', 1:'mg', 8:'dk', 27:'edge', 64:
```

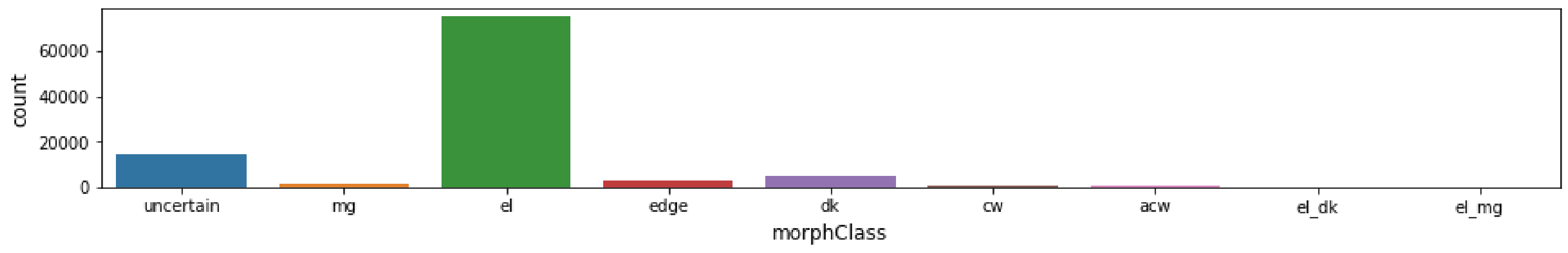


Some Visualizations:

In [196]:
```
#df['subClass'].unique()
col = 'class'
fig, axes = plt.subplots(nrows=1, ncols=3,figsize=(16, 2))

j = 2
colval = df[col].unique()
for i in [i for i in df[col].unique()]:
    ax = sns.distplot(df[df[col]==colval[j] ].redshift, bins = 30, ax = axes[j])
    ax.set_title(colval[j])
    j-=1
```

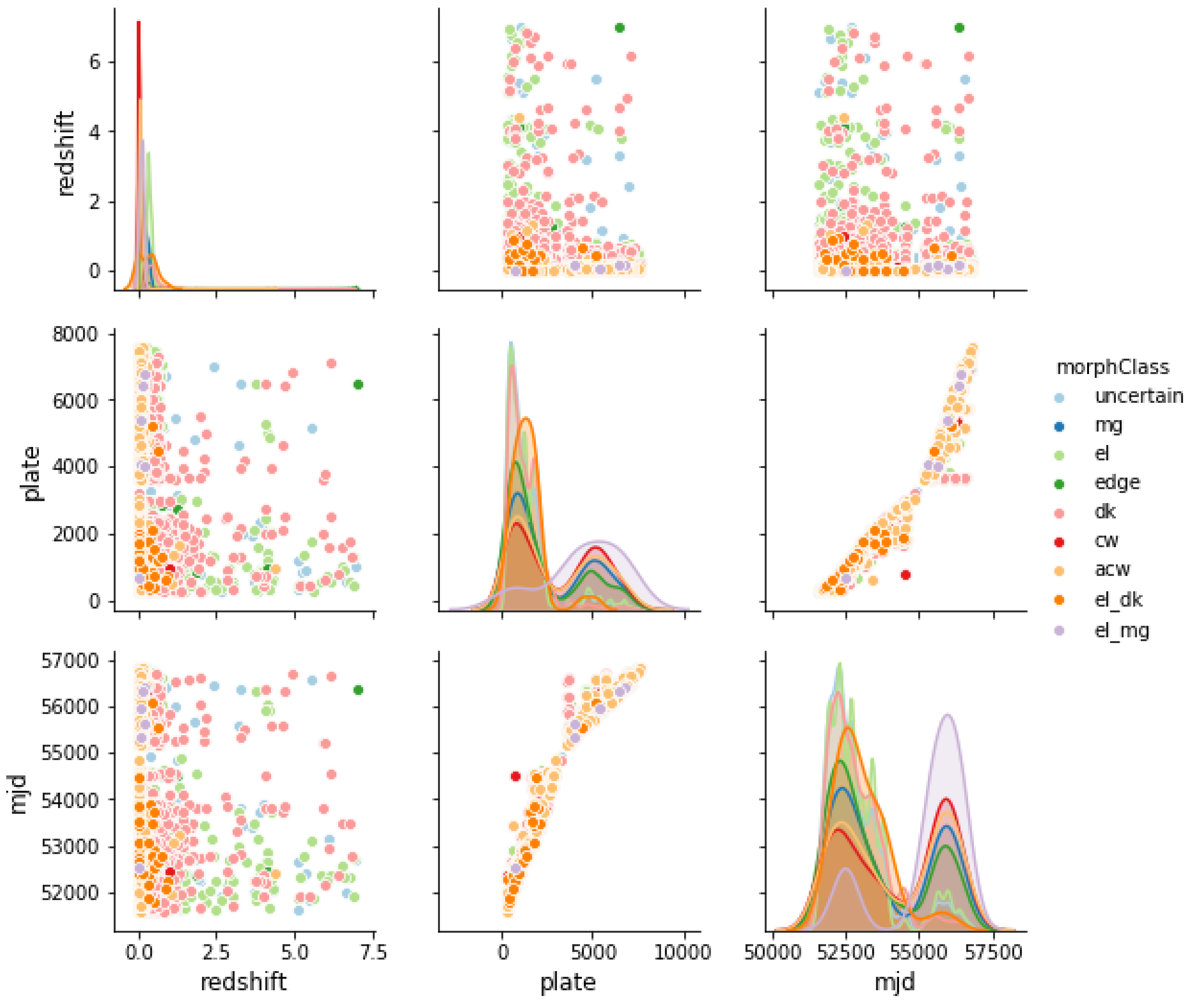


In [197]:
```
fig, axes = plt.subplots(nrows=1, ncols=1,figsize=(16, 2))
sns.countplot(df['morphClass'])
#sns.scatterplot(df['morphClass'],df['redshift'])
plt.show()
```



In [198]:
```
plt.figure(figsize=(18,2))
n = sns.pairplot(df[["redshift","plate","mjd","morphClass"]],hue="morphClass",palette='Paired')
plt.show()
```

```
<Figure size 1296x144 with 0 Axes>
```



Feature Elimination:

In [201]:
```
df.drop(["rerun","fiberid","nvote"],axis=1, inplace=True)
df.drop(prob_col,axis=1,inplace=True)
df = df[df['morphClass'] != 'el_dk']
df = df[df['morphClass'] != 'el_mg']
```

Feature Extraction and Data Splitting:

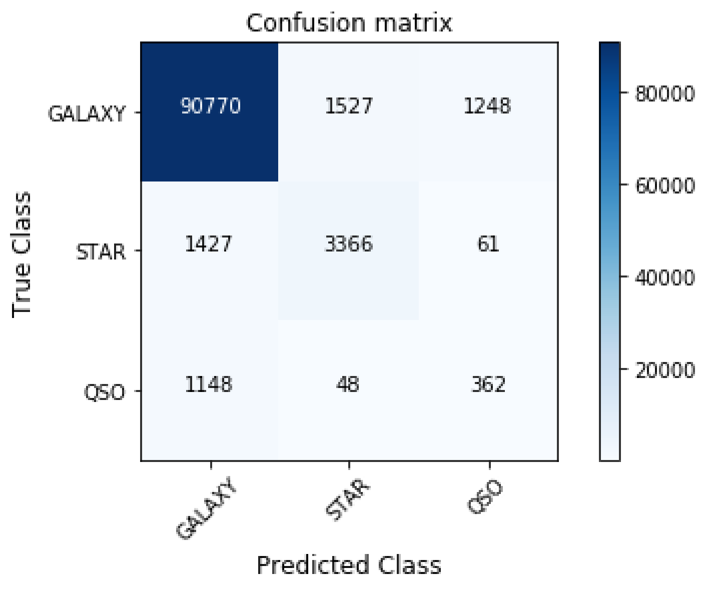
In [202]:
```
df['Con_U'] = df['petroR50_u']/df['petroR90_u']
df['Con_G'] = df['petroR50_g']/df['petroR90_g']
df['Con_R'] = df['petroR50_r']/df['petroR90_r']
df['Con_I'] = df['petroR50_i']/df['petroR90_i']
df['Con_Z'] = df['petroR50_z']/df['petroR90_z']


target_col = ['class','morphClass']
feature_col = ['u','g','r','i','z','mCr4_u', 'mCr4_g', 'mCr4_r', 'mCr4_i', 'mCr4_z','Con_U','Con_G','Con_R','Con_
sdss = df[feature_col+target_col]

#sdss_feature = df[feature_col]
#sdss_target_class = df['class']
#sdss_target_morphClass = df['morphClass']

#for i in sdss_target_col:
#    sdss[i].fillna("Uncertain", inplace = True)

fraction_training = 0.7
msk = np.random.rand(len(sdss)) < fraction_training
sdss_train = sdss[msk]
sdss_test = sdss[~msk]


print('Number data galaxies:', len(sdss))
print('Train fraction:', fraction_training)
print('Number of galaxies in training set:', len(sdss_train))
print('Number of galaxies in testing set:', len(sdss_test))

#feature and target separation
sdss_feature = sdss[feature_col]
sdss_target_class = sdss['class']
sdss_target_morphClass = sdss['morphClass']
sdss_train_feature = sdss_train[feature_col]
sdss_train_target_class = sdss_train['class']
sdss_train_target_morphClass = sdss_train['morphClass']
sdss_test_feature = sdss_test[feature_col]
sdss_test_target_class = sdss_test['class']
sdss_test_target_morphClass = sdss_test['morphClass']

sdss_classtarget = sdss[feature_col]
sdss_classtarget['class'] = sdss['class']
sdss_morphtarget = sdss[feature_col]
sdss_morphtarget['morphClass'] = sdss['morphClass']
```

```
Number data galaxies: 99957
Train fraction: 0.7
Number of galaxies in training set: 70067
Number of galaxies in testing set: 29890
```

Spectroscopic Classification using Decision Tree

In [203]:
```
dtc = DecisionTreeClassifier()
dtc.fit(sdss_train_feature, sdss_train_target_class)
sdss_predictions = dtc.predict(sdss_test_feature)

sdss_predicted = cross_val_predict(dtc, sdss_feature, sdss_target_class, cv=10)

# calculate the model score using your function
sdss_model_score = calculate_accuracy(sdss_predicted, sdss_target_class)
print("Our accuracy score:", sdss_model_score)

# calculate the models confusion matrix using sklearns confusion_matrix function
sdss_class_labels = list(set(sdss_target_class))
sdss_model_cm_dtc = confusion_matrix(y_true=sdss_target_class, y_pred=sdss_predicted, labels=sdss_class_labels)

#classification report
#from sklearn.metrics import classification_report
class_names = sdss['class'].unique()
print("classification report of Decision Tree Classifier:\n")
print(classification_report(sdss_target_class, sdss_predicted,target_names=class_names))

# Plot the confusion matrix using the provided functions.
plt.figure()
plot_confusion_matrix(sdss_model_cm_dtc, classes=sdss_class_labels, normalize=False)
plt.show()
```

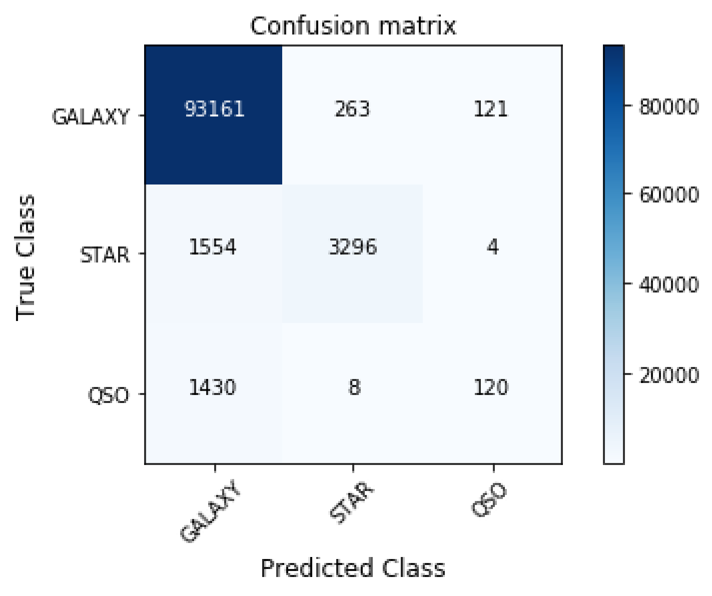
```
Our accuracy score: 0.9453865162019669
classification report of Decision Tree Classifier:

             precision    recall  f1-score   support

     GALAXY       0.97      0.97      0.97     93545
        QSO       0.22      0.23      0.22      1558
       STAR       0.68      0.69      0.69      4854

avg / total       0.95      0.95      0.95     99957

Confusion matrix, without normalization
[[90770  1527  1248]
 [ 1427  3366    61]
 [ 1148    48   362]]
```



Morphologic Classification using Decision Tree

In [209]:
```
from sklearn.model_selection import cross_val_predict
from sklearn.ensemble import RandomForestClassifier


# complete this function to get predictions from a random forest classifier
def rf_predict_actual(data, n_estimators):
  # generate the features and targets
    features = sdss[feature_col]
    targets = sdss['class']
  # instantiate a random forest classifier
    rfc = RandomForestClassifier(n_estimators=n_estimators)
  # get predictions using 10-fold cross validation with cross_val_predict
    predicted = cross_val_predict(rfc, features, targets, cv=10)
  # return the predictions and their actual classes
    return predicted, targets


# get the predicted and actual classes
sdss_number_estimators = 50              # Number of trees
sdss_predicted, sdss_actual = rf_predict_actual(sdss_classtarget, sdss_number_estimators)

# calculate the model score using your function
sdss_accuracy = calculate_accuracy(sdss_predicted, sdss_actual)
print("Accuracy score:", sdss_accuracy)

# calculate the models confusion matrix using sklearns confusion_matrix function
sdss_class_labels = list(set(sdss_actual))
sdss_model_cm_rfc = confusion_matrix(y_true=sdss_actual, y_pred=sdss_predicted, labels=sdss_class_labels)

# classification report of Random Forest Classifier
sdss_class_names = sdss_classtarget['class'].unique()
print("classification report of Decision Tree Classifier:\n")
print(classification_report(sdss_target_class, sdss_predicted,target_names=sdss_class_names))

# plot the confusion matrix using the provided functions.
plt.figure()
plot_confusion_matrix(sdss_model_cm_rfc, classes=sdss_class_labels, normalize=False)
plt.show()
```

```
Accuracy score: 0.9661854597476915
classification report of Decision Tree Classifier:

             precision    recall  f1-score   support

     GALAXY       0.97      1.00      0.98     93545
        QSO       0.49      0.08      0.13      1558
       STAR       0.92      0.68      0.78      4854

avg / total       0.96      0.97      0.96     99957

Confusion matrix, without normalization
[[93161   263   121]
 [ 1554  3296     4]
 [ 1430     8   120]]
```



Conclusion:

- Accuracy improvement is possible using more data.
- Accuracy improvement rate gets stuck due to data saturation.
- Customized algorithm with better memory handling mechanism increases accuracy.
- Data Saturation can be avoided by applying data folding methods or Big Data.

Scope:

- Cloud computing can be used here.
- Project can be used for better handing of the SDSS server.
- Fetched and Optimized dataset can be used for educational purposes. (not suitable for commercial projects)

- Project can be modified to use in other servers also.

Credit:

1. GalaxyZoo [https://blog.galaxyzoo.org/] (https://blog.galaxyzoo.org/])
2. Sloan Digital Sky Survey [http://www.sdss3.org/] (http://www.sdss3.org/])
3. Astronomy and Big Data: A Data Clustering Approach to Identifying Uncertain Galaxy Morphology [https://www.springer.com/in/book/9783319065984] (https://www.springer.com/in/book/9783319065984])
4. Encyclopaedia of Astronomy and Astrophysics. [http://www.astro.caltech.edu/~george/ay21/eaa/eaa-classif.pdf] (http://www.astro.caltech.edu/~george/ay21/eaa/eaa-classif.pdf])