# Astronomical object classification

Group 12: Le Uyen Nhu Dinh, Sheikh Hasan Elahi, Isma Sohail.

We found the template did not match the way we explore the data, so we keep 'Data exploration and visualization, Data cleaning, Data exploration and visualization after cleaning' all in one section.

## Imports

In [56]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import zscore

from sklearn.model_selection import train_test_split, GridSearchCV, cross_va
from sklearn.preprocessing import LabelEncoder,StandardScaler, FunctionTrans
from sklearn.compose import ColumnTransformer, make_column_selector
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatri

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier, AdaBc
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

## Reading the data

In [57]:
```
df = pd.read_csv('train.csv')
df.head(10)
```

Out[57]:

| | obj_ID | alpha | delta | u | g | r | |
|---|---|---|---|---|---|---|---|
| **0** | 1.237661e+18 | 135.689107 | 32.494632 | 23.87882 | 22.27530 | 20.39501 | 19.16573 |
| **1** | 1.237665e+18 | 144.826101 | 31.274185 | 24.77759 | 22.83188 | 22.58444 | 21.16812 |
| **2** | 1.237661e+18 | 142.188790 | 35.582444 | 25.26307 | 22.66389 | 20.60976 | 19.34857 |
| **3** | 1.237663e+18 | 338.741038 | -0.402828 | 22.13682 | 23.77656 | 21.61162 | 20.50454 |
| **4** | 1.237680e+18 | 345.282593 | 21.183866 | 19.43718 | 17.58028 | 16.49747 | 15.97711 |
| **5** | 1.237680e+18 | 340.995121 | 20.589476 | 23.48827 | 23.33776 | 21.32195 | 20.25615 |
| **6** | 1.237679e+18 | 23.234926 | 11.418188 | 21.46973 | 21.17624 | 20.92829 | 20.60826 |
| **7** | 1.237679e+18 | 5.433176 | 12.065186 | 22.24979 | 22.02172 | 20.34126 | 19.48794 |
| **8** | 1.237671e+18 | 39.149691 | 28.102842 | 21.74669 | 20.03493 | 19.17553 | 18.81823 |
| **9** | 1.237680e+18 | 328.092076 | 18.220310 | 25.77163 | 22.52042 | 20.63884 | 19.78071 |

**Comment**: The training dataset has 80000 samples with 'class' as target variables and 17 other features.

## EDA

In [58]:
```
# Statiscal summary:
df.describe()
```

Out[58]:

| | obj_ID | alpha | delta | u | g |
|---|---|---|---|---|---|
| **count** | 8.000000e+04 | 80000.000000 | 80000.000000 | 79638.000000 | 80000.000000 |
| **mean** | 1.237665e+18 | 177.579220 | 24.132590 | 21.961115 | 20.507677 |
| **std** | 8.424878e+12 | 96.409584 | 19.650113 | 35.581856 | 35.483302 |
| **min** | 1.237646e+18 | 0.005528 | -18.785328 | -9999.000000 | -9999.000000 |
| **25%** | 1.237659e+18 | 127.643892 | 5.170723 | 20.353990 | 18.963188 |
| **50%** | 1.237663e+18 | 180.761747 | 23.603480 | 22.187965 | 21.101015 |
| **75%** | 1.237668e+18 | 233.815698 | 39.904905 | 23.698457 | 22.125007 |
| **max** | 1.237681e+18 | 359.999615 | 83.000519 | 32.781390 | 31.602240 |

In [59]:
```
# Check samples where one of 'u', 'g', 'z' has value smaller than 0
df[(df['u'] < 0) | (df['g'] < 0) | (df['z'] < 0)]
```

Out[59]:

| | obj_ID | alpha | delta | u | g | r | |
|---|---|---|---|---|---|---|---|
| **63607** | 1.237649e+18 | 224.006526 | -0.624304 | -9999.0 | -9999.0 | 18.1656 | 18.01675 |

**Comment**: It is noticable that 'u', 'g', and 'z' has minimum value at -9999, which so very far from the mean and maximum value. There is absolutely one 1

samples where these three features have -9999. This must be an outlier.

In [60]:
```
# Check missing values
df.isna().any()
```

Out[60]:
```
obj_ID         False
alpha          False
delta          False
u               True
g              False
r              False
i              False
z              False
run_ID         False
rerun_ID       False
cam_col        False
field_ID       False
spec_obj_ID    False
class          False
redshift       False
plate          False
MJD            False
fiber_ID       False
dtype: bool
```

In [61]:
```
# Observe rows having missing values at feature 'u'
pd.DataFrame(df[df.isna().any(axis = 1)])
```

Out[61]:

| | obj_ID | alpha | delta | u | g | r | |
|---|---|---|---|---|---|---|---|
| **397** | 1.237668e+18 | 243.674064 | 9.251630 | NaN | 17.41451 | 17.47570 | 17.40724 |
| **401** | 1.237679e+18 | 2.891353 | 1.480128 | NaN | 21.74609 | 21.85504 | 21.92592 |
| **513** | 1.237651e+18 | 184.833910 | -2.004270 | NaN | 21.64660 | 21.53089 | 21.46142 |
| **603** | 1.237668e+18 | 220.769515 | 14.164362 | NaN | 21.59484 | 21.31533 | 21.57566 |
| **642** | 1.237658e+18 | 179.468486 | 54.324141 | NaN | 20.20231 | 20.06267 | 19.97901 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **79524** | 1.237668e+18 | 136.425633 | 17.655753 | NaN | 22.65696 | 21.96614 | 21.13199 |
| **79540** | 1.237649e+18 | 150.229991 | -0.929703 | NaN | 21.12765 | 20.96447 | 21.03944 |
| **79541** | 1.237658e+18 | 149.508362 | 50.758880 | NaN | 17.34075 | 17.36986 | 17.44620 |
| **79748** | 1.237658e+18 | 141.716794 | 47.433948 | NaN | 18.61372 | 18.60112 | 18.60454 |
| **79791** | 1.237663e+18 | 321.869739 | 74.898733 | NaN | 19.05700 | 18.09605 | 17.62846 |

362 rows × 18 columns

**Comment**: Among 17 features, only 'u' has missing values and this feature contains 362 missing samples out of 80000 total. So this number is not too significant and they can be both removed or imputed.

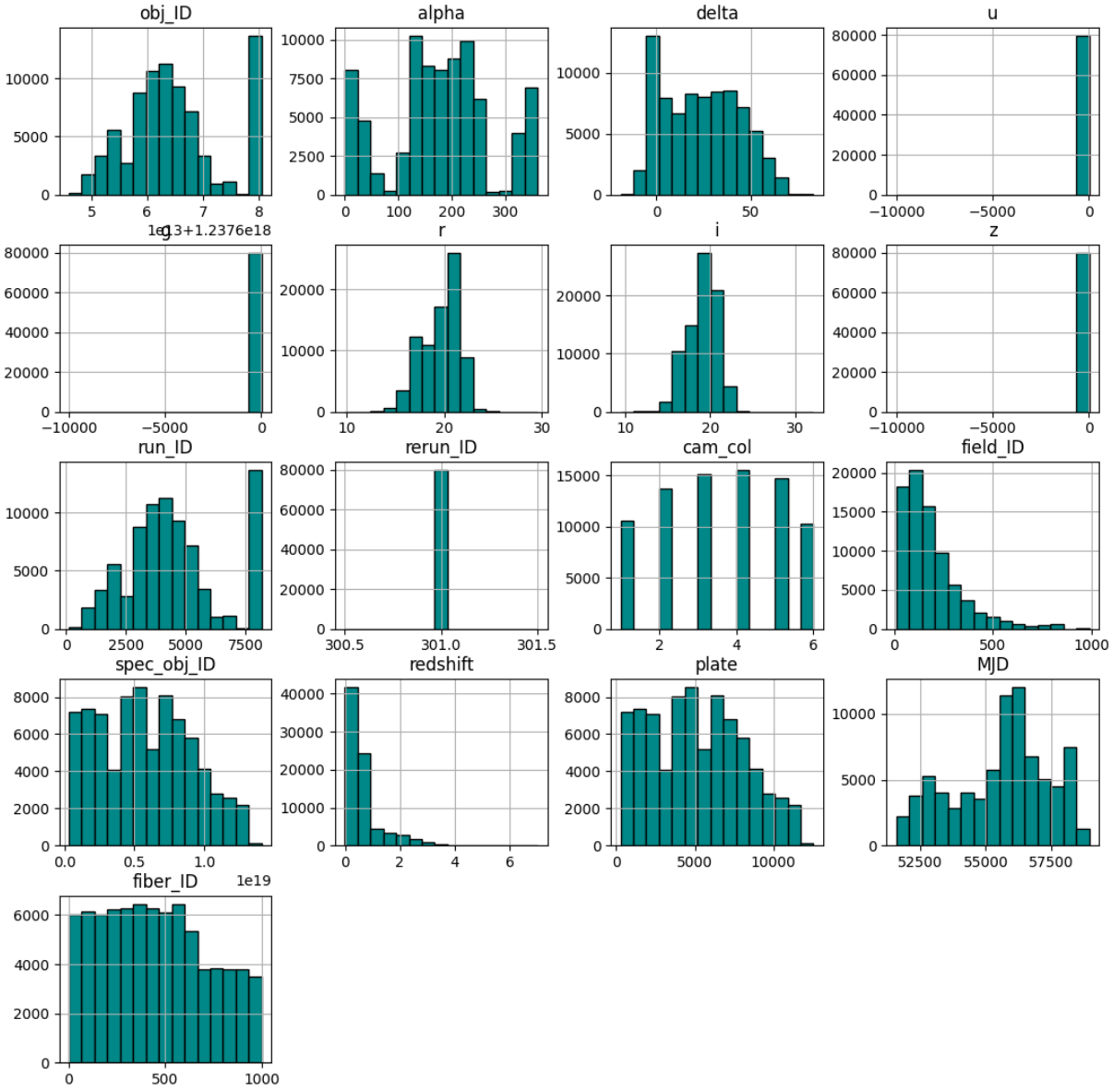
However, if just putting the mean/median values of the whole columns would bring wrong information to the learning. So based on which class the sample belongs to, the imputed values will be the mean of 'u' values to that particular class.

In [62]:
```
# Check for duplicated samples:
df.duplicated().any()
```

Out[62]: False

**Comment**: There is no duplicated data.

In [63]:
```
# Histograms for 17 features
df.drop(['class'],axis=1).hist(figsize=(13,13),
                               bins = 15, color = 'darkcyan',
                               edgecolor='black')
plt.show()
```



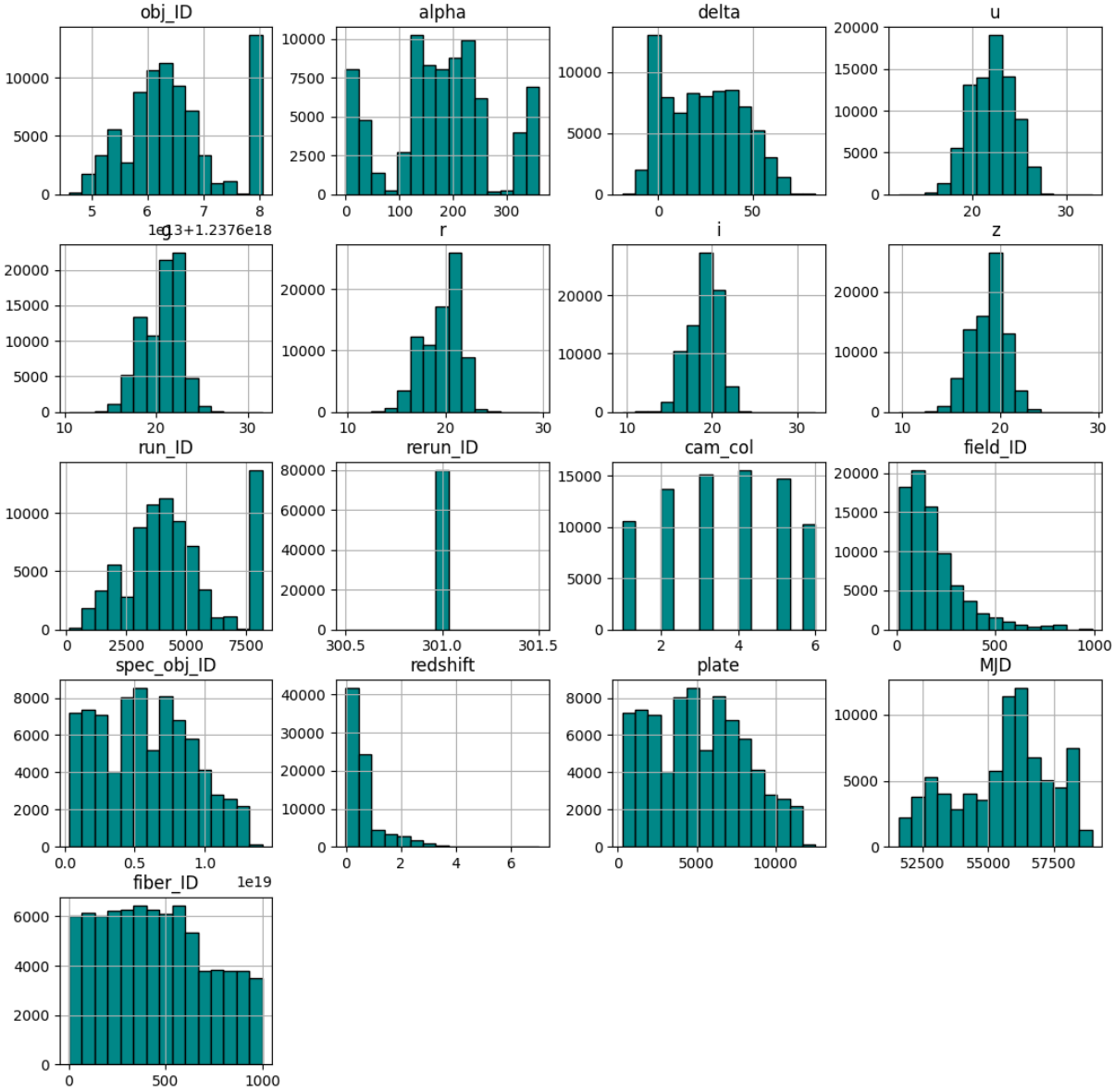
**Comment**: Each feature has distinguish distribution.

- Feature 'u', 'g', 'z' are very suspicious that as observed from dataset, their values lies between 17-25. But in the histogram none of the datapoints has value above 0. The plot could be distorted by the existence of the value -9999 in these features.

-> Very importance: Removing '-9999' sample

- Feature 'rerun_ID' seems to have only 1 value: 301. This need to be checked.
- Each feature has different range to the data need to be scaled for less computation.

In [64]:
```
# Removing -9999 values
vars = ['u', 'g', 'z']
df = df[~df[vars].isin([-9999]).any(axis=1)]

df.drop(['class'],axis=1).hist(figsize=(13,13),
                               bins = 15, color = 'darkcyan',
                               edgecolor='black')
plt.show()
```




**Comment**: Most of the features' distributions are skewed and unlikely to be Guassian.

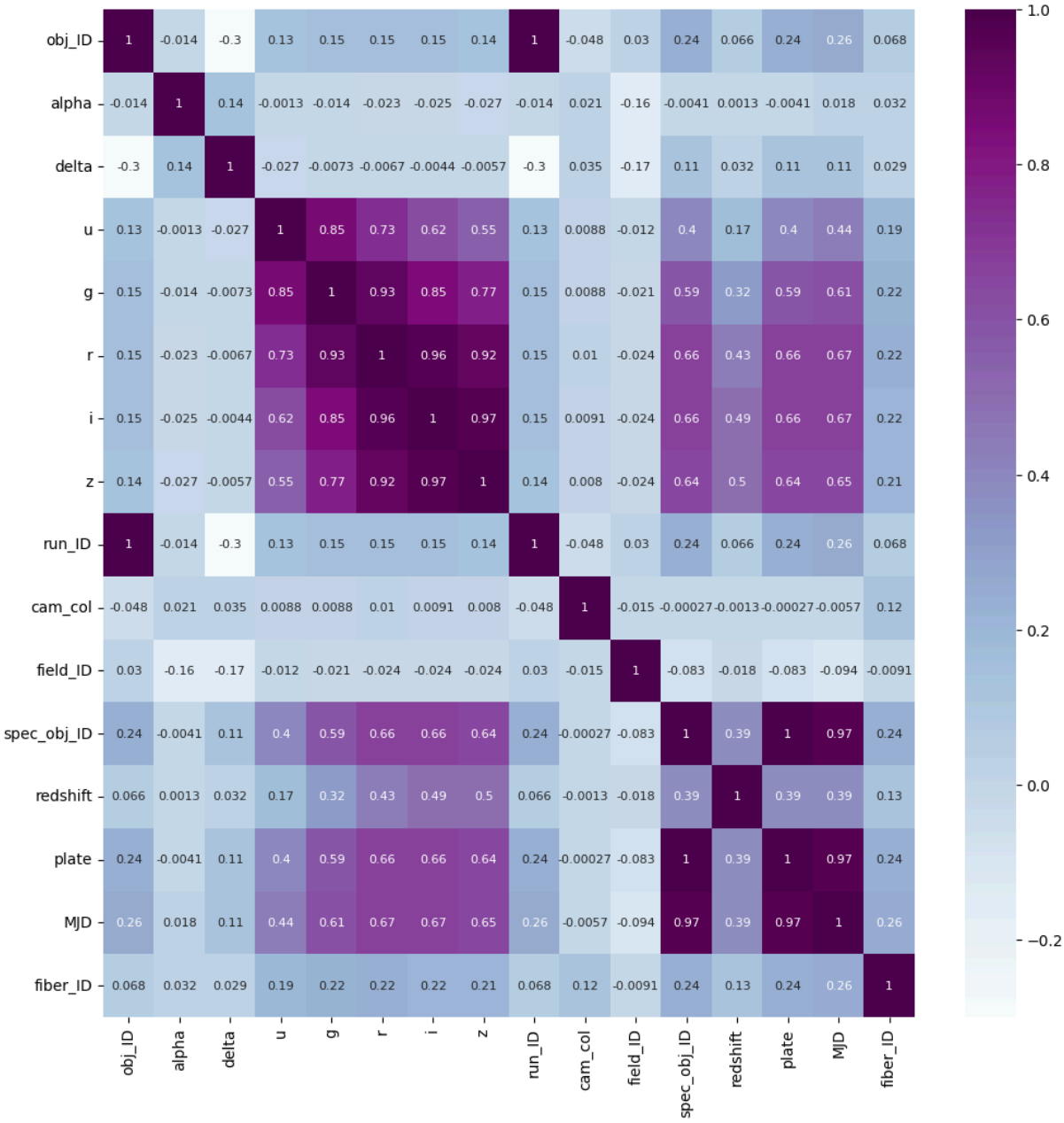
In [65]:
```
# Check number of unique values in 'rerun_ID'
df['rerun_ID'].nunique()
```

Out[65]: 1

**Comment**: This means all sample in training set has 301 as their 'rerun_ID' so this feature is not meaningful for the classification. The same happens in test set so 'rerun_ID' must be completely removed from train and test data.

In [66]:
```
# Dataframe for current 16 features
features_df = df.drop(['class','rerun_ID'], axis =1)

# Correlation matrix:
plt.figure(figsize=(12, 12))
sns.heatmap(features_df.corr(), annot=True,
```

```
            annot_kws={'size': 8}, cmap="BuPu")
plt.show()
```




**Comment**: The heatmap shows some pairs of feature have very high correlation (somes at 1).

- For those pairs have correlation of 1, the correlation of the features with the third features are exactly the same. Examples: ('obj_ID' vs 'run_ID') and ('spec_obj_ID' vs'plate').

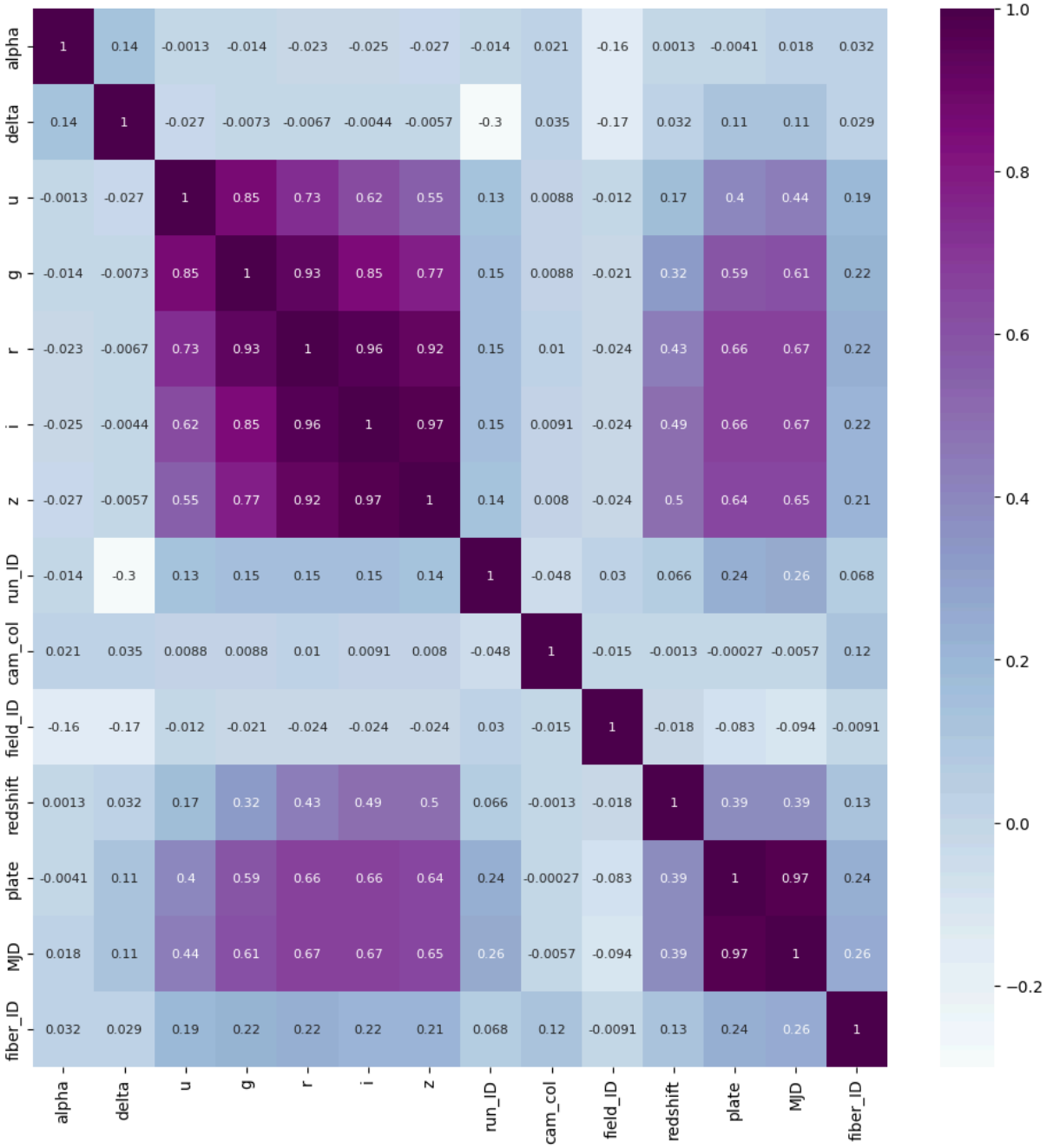
!! This adds noise, increases training time so one of the features can be dropped. Because:

- 'obj_ID'= Object Identifier, the unique value that identifies the object in the image catalog used by the CAS

- 'run_ID' = Run Number used to identify the specific scan
- 'spec_obj_ID' = Unique ID used for optical spectroscopic objects
- 'plate' = plate ID, identifies each plate in SDSS

They are not real features, just metadata so we can just keep one of each pair.

In [67]:
```
# Correlation matrix after removing 'obj_ID' and 'spec_obj_ID'
features_df = features_df.drop(['obj_ID','spec_obj_ID'], axis =1)

plt.figure(figsize=(12, 12))
sns.heatmap(features_df.corr(), annot=True,
            annot_kws={'size': 8}, cmap="BuPu")
plt.show()
```



**Comment**: Feature 'u', 'g', 'r', 'i', 'z' are highly correlated, why?

u = Ultraviolet filter in the photometric system

g = Green filter in the photometric system

r = Red filter in the photometric system

i = Near Infrared filter in the photometric system

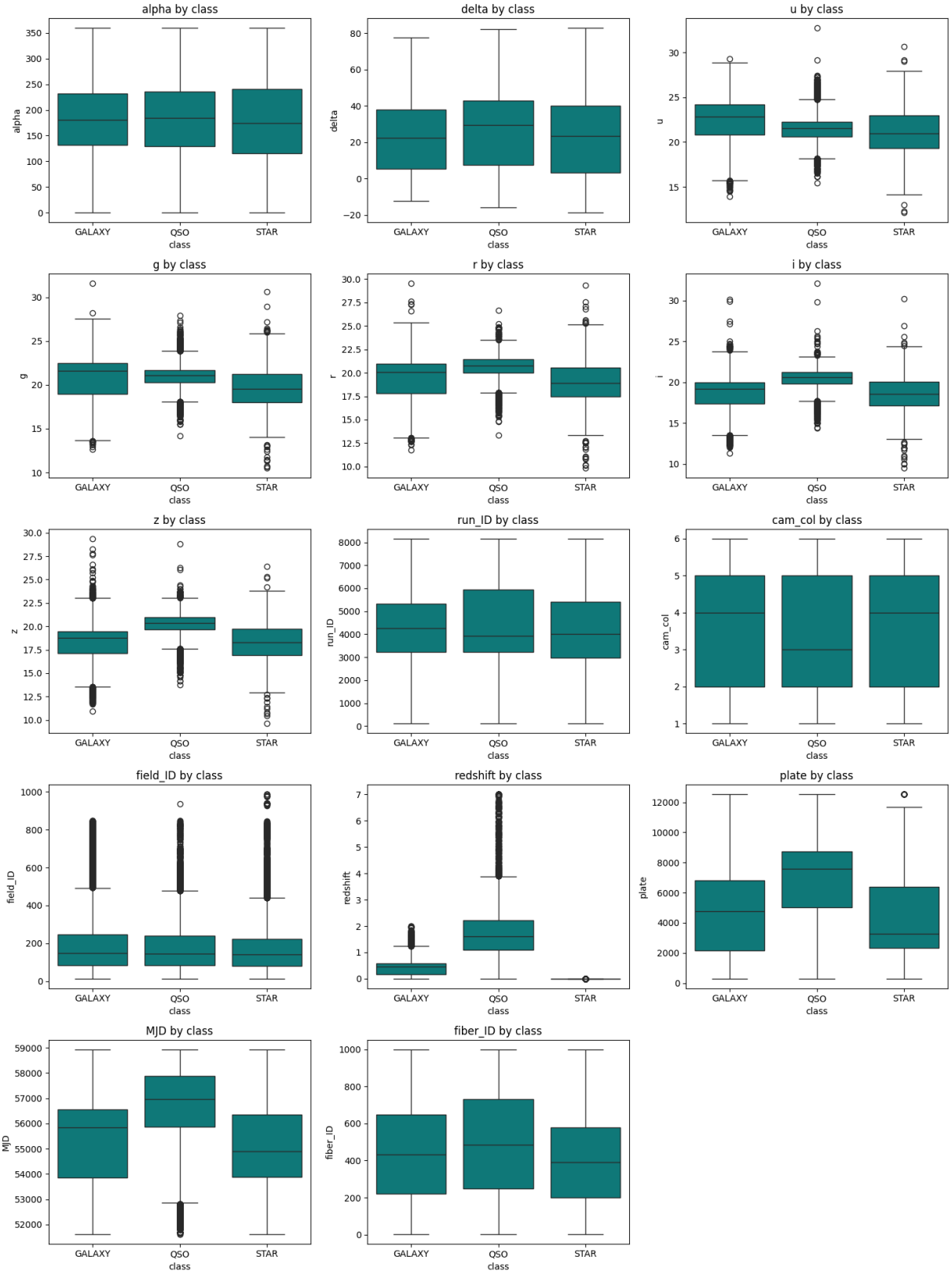
z = Infrared filter in the photometric system

These filters are all measuring light intensity from the same object, just at different wavelengths.

In [68]:
```
combined_df = df[features_df.columns.tolist() + ['class']].copy()

sns.pairplot(combined_df, hue='class')
```

Out[68]:
```
<seaborn.axisgrid.PairGrid at 0x2c6c05acf40>
```

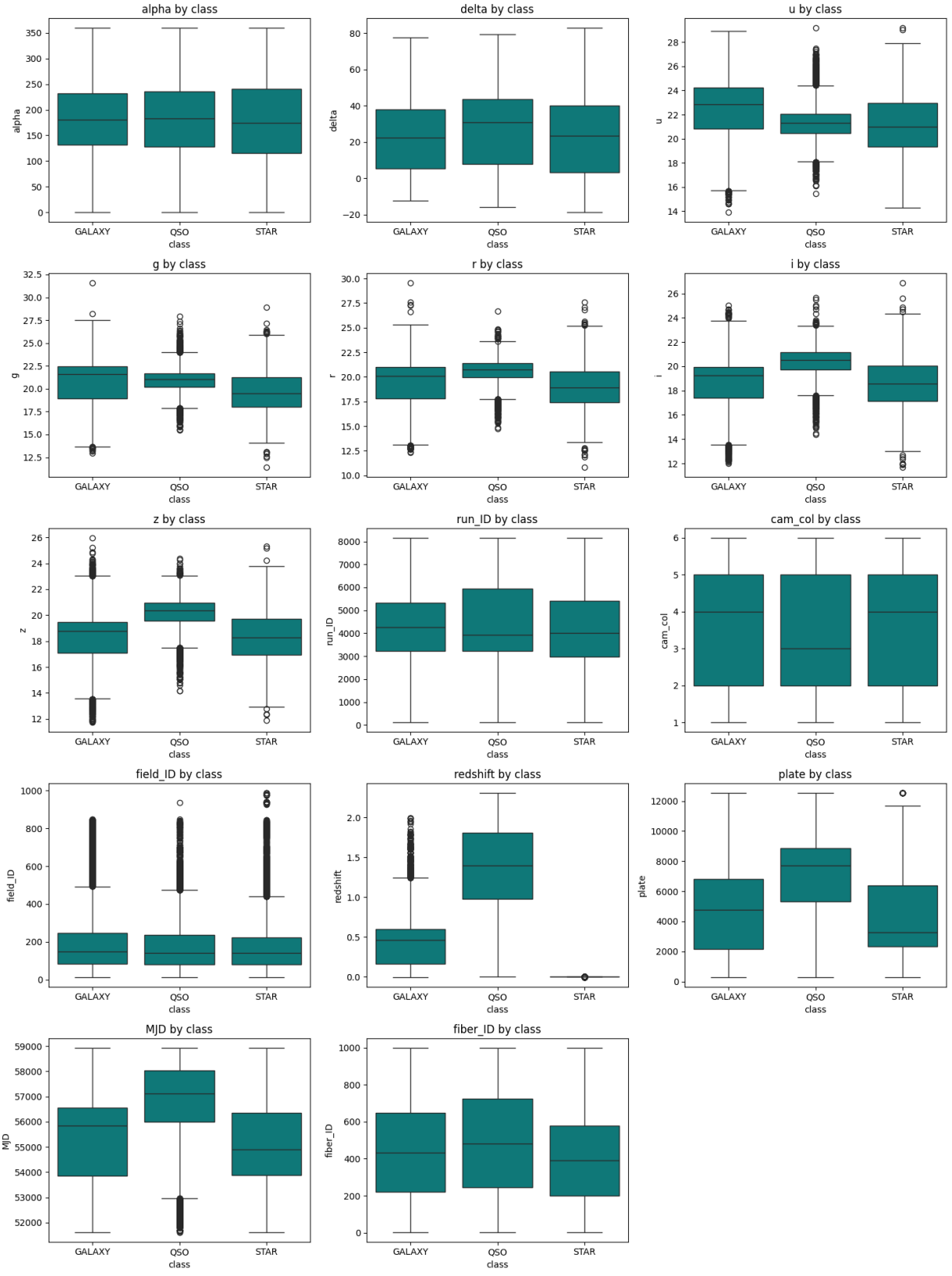

In [ ]:
```
# Boxplots classified by 'class' and to see outliers:
fig, axes = plt.subplots(5, 3, figsize=(15, 20))
axes = axes.flatten()

# Loop through features
for i, col in enumerate(features_df.columns):
    sns.boxplot(data=df, x='class', y=col, ax=axes[i],
                color = 'darkcyan')
    axes[i].set_title(f'{col} by class')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```



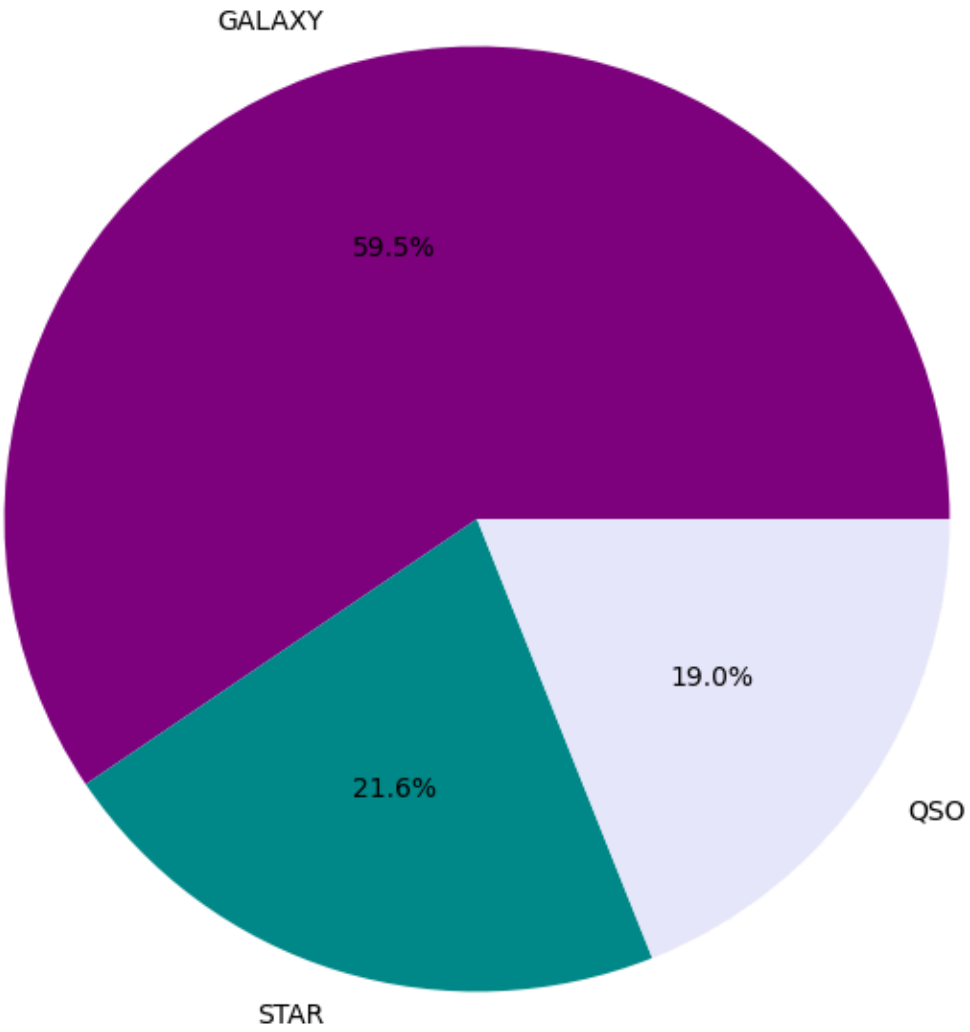

Because we saw that most of features are skewed / not Guassian so it seems that z-score is not applicable to remove the outliers. Then we try the Interquartile Range (IQR) method, but only on some specific columns like 'u', 'z', 'i', 'z', 'MJD', 'redshift'.

In [18]:
```
# Removing outliers:
def remove_outliers_iqr(df, columns):
    filtered_df = df.copy()
    for column in columns:
        Q1 = filtered_df[column].quantile(0.25)
        Q3 = filtered_df[column].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 2.5 * IQR
        upper_bound = Q3 + 2.5 * IQR
        filtered_df = filtered_df[(filtered_df[column] >= lower_bound) & (fi
    return filtered_df

cleaned_df = remove_outliers_iqr(df, ['u', 'z', 'i', 'z', 'MJD', 'redshift']
```

In [19]:
```
fig, axes = plt.subplots(5, 3, figsize=(15, 20))
axes = axes.flatten()

# Loop through features
for i, col in enumerate(features_df.columns):
    sns.boxplot(data=cleaned_df, x='class', y=col, ax=axes[i],
                color = 'darkcyan')
    axes[i].set_title(f'{col} by class')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

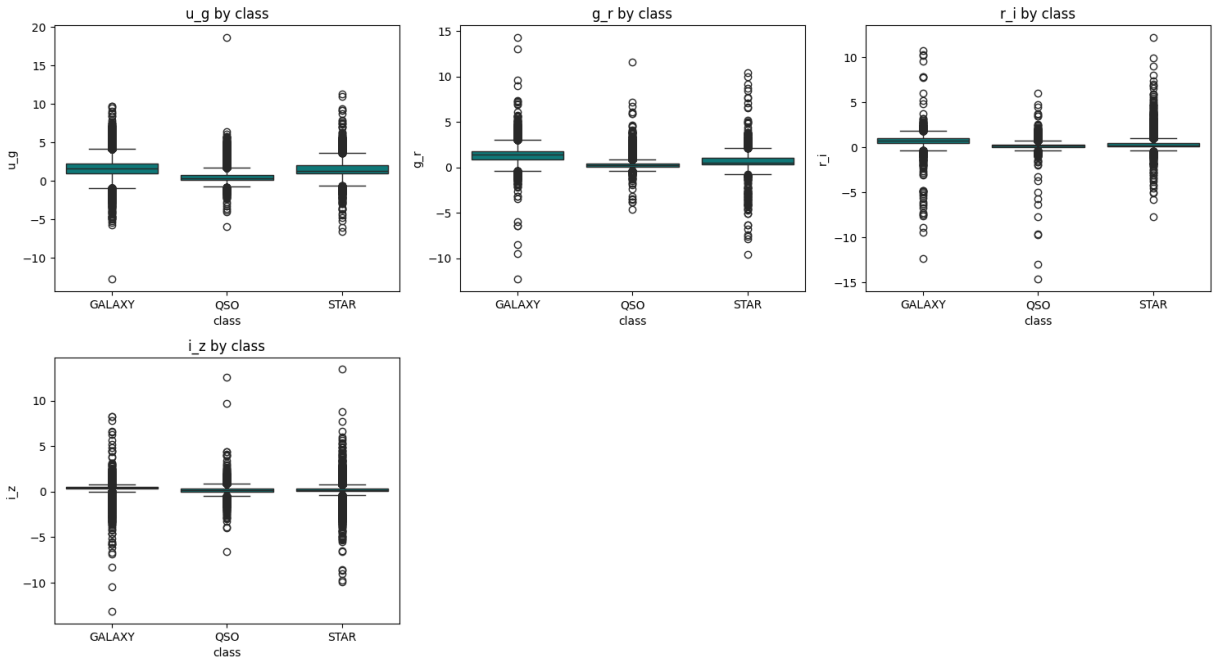



In [20]:
```
classes = df['class'].value_counts()
plt.figure(figsize=(8,8))
plt.pie(classes, labels= classes.index, autopct= '%1.1f%%',
        colors=['purple', 'darkcyan', 'lavender'])
plt.show()
```




In [ ]:
```
# Testing : feature engineering
df['u_g'] = df['u'] - df['g']
df['g_r'] = df['g'] - df['r']
df['r_i'] = df['r'] - df['i']
df['i_z'] = df['i'] - df['z']

# Boxplots classified by 'class' :
fig, axes = plt.subplots(2, 2, figsize=(15, 20))
axes = axes.flatten()

# Loop through generated features:
for i, col in enumerate(['u_g', 'g_r', 'r_i', 'i_z']):
    sns.boxplot(data=df, x='class', y=col, ax=axes[i],
                color = 'darkcyan')
    axes[i].set_title(f'{col} by class')

for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])
```

```
plt.tight_layout()
plt.show()
```




**Summary**: To sum up what we are going to preprocess data

*For training only*:

- Remove samples with '-9999' values at features : only training
- Impute nan values in 'u' with the mean of each class.
- Remove outliers with 2.5*IQR

*Both training and test*:

- Scaling
- Encode the 'class'
- Remove 'rerun_ID', 'obj_ID', 'spec_obj_ID' columns.

# DATA PREPROCESSING

In [ ]:
```
# All preprocessing steps in functions:
def remove_minus_9999(df, features):
    return df[~(df[features] == -9999).any(axis=1)].copy()

def impute_u_by_class(df, target_col='class'):
    means = df.groupby(target_col)['u'].mean()
    for cls, mean_val in means.items():
        mask = (df[target_col] == cls) & (df['u'].isnull())
        df.loc[mask, 'u'] = mean_val
    return df

def remove_outliers_iqr(df, features, threshold=2.5):
    for col in features:
```

```
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower = Q1 - threshold * IQR
        upper = Q3 + threshold * IQR
        df = df[(df[col] >= lower) & (df[col] <= upper)]
    return df.copy()

def feature_engineer(df):
    df = df.copy()
    df['u_g'] = df['u'] - df['g']
    df['g_r'] = df['g'] - df['r']
    df['r_i'] = df['r'] - df['i']
    df['i_z'] = df['i'] - df['z']
    return df

def drop_columns(df, cols):
    return df.drop(columns=cols, errors='ignore')

DROP_COLS = ['rerun_ID', 'obj_ID', 'spec_obj_ID']
FILTER_COLS = ['u', 'g', 'r', 'i', 'z']
```

In [ ]:
```
# Refine training data
final_df = pd.read_csv('train.csv')

le = LabelEncoder()
final_df['class'] = le.fit_transform(final_df['class'])

final_df = remove_minus_9999(final_df, FILTER_COLS)
final_df = impute_u_by_class(final_df, target_col='class')
final_df = remove_outliers_iqr(final_df, features=FILTER_COLS + ['MJD', 'rec
```

In [20]:
```
# Split data
X = final_df.drop(columns='class')
y = final_df['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, str
```

In [ ]:
```
# Complete preprocessing pipeline
scale_transform = ColumnTransformer([
    ('scale', StandardScaler(), make_column_selector(dtype_include='number')
])

preprocess_pipe = Pipeline([
    ('feature_eng', FunctionTransformer(feature_engineer)),
    ('drop_cols', FunctionTransformer(lambda df: drop_columns(df, DROP_COLS)
    ('scale', scale_transform)
])
```

## EVALUATING DIFFERENT MODELS/ HYPERPARAMTERS

In [ ]:
```
# Pipeline 1: Regularized Logistic Regression + PCA
pipe1 = Pipeline([
    ('preprocessing', preprocess_pipe),
    ('lr', LogisticRegression(max_iter=1000, solver='saga', multi_class='mul
```

```
])

# Gridsearch: regularization strength + penalty type
params_lr = {
    'lr__C': [0.1, 1.0, 10],
    'lr__penalty': ['l1', 'l2']
}

# GridSearch with macro F1 scoring
grid_lr = GridSearchCV(pipe1, param_grid=params_lr, cv=5, scoring='f1_macro'
grid_lr.fit(X_train, y_train)


# Results
print("Best Logistic Regression F1 Macro:", grid_lr.best_score_)
print("Best Parameters:", grid_lr.best_params_)

# Evaluate:
y_pred1 = grid_lr.predict(X_test)
f1_macro1 = f1_score(y_test, y_pred1, average='macro')
print(f"Macro F1-score: {f1_macro1:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred1, digits=4))
```

```
c:\Users\nguye\AppData\Local\Programs\Python\Python310\lib\site-packages\skl
earn\linear_model\_logistic.py:1247: FutureWarning: 'multi_class' was deprec
ated in version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Best Logistic Regression F1 Macro: 0.947057817487227
Best Parameters: {'lr__C': 10, 'lr__penalty': 'l1'}
Macro F1-score: 0.9446

 Classification Report:
              precision    recall  f1-score   support

           0     0.9614    0.9709    0.9661      9515
           1     0.9310    0.8451    0.8860      2395
           2     0.9642    0.9997    0.9816      3449

    accuracy                         0.9577     15359
   macro avg     0.9522    0.9386    0.9446     15359
weighted avg     0.9573    0.9577    0.9571     15359
```

```
c:\Users\nguye\AppData\Local\Programs\Python\Python310\lib\site-packages\skl
earn\linear_model\_sag.py:349: ConvergenceWarning: The max_iter was reached
which means the coef_ did not converge
  warnings.warn(
```

**Pipeline 1 score 0.955 on Kaggle.**

In [31]:
```
# Pipeline 2: PCA + Kernel SVM
```

```
pipe2 = Pipeline([
    ('preprocessing', preprocess_pipe),
    ('pca', PCA(n_components=0.95)),
    ('svc', SVC(kernel='rbf', probability=True, random_state=42))
])

params_svc = {
    'svc__C': [5, 10],
    'svc__gamma': ['scale', 'auto']
}

grid_svc = GridSearchCV(pipe2, params_svc, cv=5, scoring='f1_macro', n_jobs=
grid_svc.fit(X_train, y_train)

# Results
print("Best SVM F1 Macro:", grid_svc.best_score_)
print("Best Parameters:", grid_svc.best_params_)

# Evaluate:
y_pred2 = grid_svc.predict(X_test)
f1_macro2 = f1_score(y_test, y_pred2, average='macro')
print(f"Macro F1-score: {f1_macro2:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred2, digits=4))
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
Best SVM F1 Macro: 0.9587121804674691
Best Parameters: {'svc__C': 10, 'svc__gamma': 'scale'}
Macro F1-score: 0.9579

 Classification Report:
              precision    recall  f1-score   support

           0     0.9713    0.9754    0.9734      9515
           1     0.9537    0.8948    0.9233      2395
           2     0.9623    0.9925    0.9772      3449

    accuracy                         0.9667     15359
   macro avg     0.9625    0.9542    0.9579     15359
weighted avg     0.9666    0.9667    0.9664     15359
```

**Pipeline 2 score 0.959 on Kaggle.**

In [34]:
```
# Pipeline 3: Majority Voting between regularized Logistic Regression and Ke

pipe_lr = Pipeline([
    ('pre', preprocess_pipe),
    ('clf', LogisticRegression(max_iter=1000, solver='saga', C=10, penalty='
])

pipe_svm = Pipeline([
    ('pre', preprocess_pipe),
    ('clf', SVC(kernel='rbf', C=10, gamma='scale', probability=True, random_
])
```

```
vote1 = VotingClassifier(
    estimators=[('lr', pipe_lr), ('svm', pipe_svm)],
    voting='soft',
    n_jobs=-1
)

vote1.fit(X_train, y_train)
y_pred3 = vote1.predict(X_test)
f1_macro3 = f1_score(y_test, y_pred3, average='macro')
print(f"Macro F1-score: {f1_macro3:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred3, digits=4))
```

```
Macro F1-score: 0.9576

 Classification Report:
              precision    recall  f1-score   support

           0     0.9688    0.9786    0.9736      9515
           1     0.9577    0.8781    0.9161      2395
           2     0.9688    0.9977    0.9830      3449

    accuracy                         0.9672     15359
   macro avg     0.9651    0.9514    0.9576     15359
weighted avg     0.9670    0.9672    0.9668     15359
```

**Pipeline 3 score 0.967 on Kaggle.**

In [37]:
```
# Pipeline 4: Voting with more classifiers
logreg = LogisticRegression(max_iter=1000, solver='saga', C=10, penalty='l1'
dt = DecisionTreeClassifier(max_depth=8, min_samples_split=4, min_samples_le
rf = RandomForestClassifier(n_estimators=150, max_depth=16, min_samples_spli
svm = SVC(kernel='rbf', C=10, gamma='scale', probability=True, random_state=

vote2 = VotingClassifier(
    estimators=[('lr', logreg), ('dt', dt), ('rf', rf), ('svm', svm)],
    voting='hard',
    n_jobs=-1
)

vote2_pipe = Pipeline([
    ('pre', preprocess_pipe),
    ('model', vote2)
])

vote2_pipe.fit(X_train, y_train)
y_pred4 = vote2_pipe.predict(X_test)
f1_macro4 = f1_score(y_test, y_pred4, average='macro')
print(f"Macro F1-score: {f1_macro4:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred4, digits=4))
```

```
Macro F1-score: 0.9659

 Classification Report:
              precision    recall  f1-score   support

           0     0.9729    0.9863    0.9796      9515
           1     0.9578    0.8914    0.9234      2395
           2     0.9897    0.9997    0.9947      3449

    accuracy                         0.9745     15359
   macro avg     0.9735    0.9592    0.9659     15359
weighted avg     0.9743    0.9745    0.9742     15359
```

**Pipeline 4 score 0.974 on Kaggle.**

In [38]:
```
# Pipeline 5: Gradient Boosting
gb_pipe = Pipeline([
    ('pre', preprocess_pipe),
    ('gb', GradientBoostingClassifier(random_state=42, n_estimators = 200, m
])

gb_pipe.fit(X_train, y_train)

y_pred5 = gb_pipe.predict(X_test)
f1_macro5 = f1_score(y_test, y_pred5, average='macro')
print(f"Macro F1-score: {f1_macro5:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred5, digits=4))
```

```
Macro F1-score: 0.9696

 Classification Report:
              precision    recall  f1-score   support

           0     0.9785    0.9853    0.9819      9515
           1     0.9444    0.9148    0.9294      2395
           2     0.9962    0.9988    0.9975      3449

    accuracy                         0.9773     15359
   macro avg     0.9730    0.9663    0.9696     15359
weighted avg     0.9772    0.9773    0.9772     15359
```

**Pipeline 5 score 0.975 on Kaggle.** I have done gridsearch on GradientBoosting parameters which took 26 minutes to finish so I just give the best hypeparameters here.

In [42]:
```
# Pipeline 5: AdaBoosting
pipe6 = Pipeline([
    ('pre', preprocess_pipe),
    ('ada', AdaBoostClassifier(DecisionTreeClassifier(max_depth = 9,min_samp
])
```

```
pipe6.fit(X_train, y_train)

y_pred6 = pipe6.predict(X_test)
f1_macro6 = f1_score(y_test, y_pred6, average='macro')
print(f"Macro F1-score: {f1_macro6:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred6, digits=4))
```

```
c:\Users\nguye\AppData\Local\Programs\Python\Python310\lib\site-packages\skl
earn\ensemble\_weight_boosting.py:527: FutureWarning: The SAMME.R algorithm
(the default) is deprecated and will be removed in 1.6. Use the SAMME algori
thm to circumvent this warning.
  warnings.warn(
Macro F1-score: 0.9638

 Classification Report:
              precision    recall  f1-score   support

           0     0.9733    0.9834    0.9783      9515
           1     0.9431    0.8931    0.9174      2395
           2     0.9917    0.9997    0.9957      3449

    accuracy                         0.9730     15359
   macro avg     0.9693    0.9587    0.9638     15359
weighted avg     0.9727    0.9730    0.9727     15359
```

**Pipeline 6 score 0.976 on Kaggle.**

In [45]:
```
# Pipeline 7: Voting with Gradient Boosting and AdaBoost
gb = GradientBoostingClassifier(
    random_state=42,
    n_estimators=200,
    max_depth=5
)

ada = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=10, min_samples_leaf=5),
    random_state=42
)

# === Voting Classifier (Hard or Soft)
vote_clf = VotingClassifier(
    estimators=[('gb', gb), ('ada', ada)],
    voting='soft',
    n_jobs=-1
)

pipe7 = Pipeline([
    ('pre', preprocess_pipe),
    ('vote', vote_clf)
])

pipe7.fit(X_train, y_train)
```

```
y_pred7 = pipe7.predict(X_test)
f1_macro7 = f1_score(y_test, y_pred7, average='macro')
print(f"Macro F1-score: {f1_macro7:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred7, digits=4))
```

```
Macro F1-score: 0.9698

 Classification Report:
              precision    recall  f1-score   support

           0     0.9780    0.9858    0.9819      9515
           1     0.9492    0.9123    0.9304      2395
           2     0.9948    0.9997    0.9973      3449

    accuracy                         0.9775     15359
   macro avg     0.9740    0.9659    0.9698     15359
weighted avg     0.9773    0.9775    0.9773     15359
```

**Pipeline 7 score 0.976 on Kaggle.**

In [ ]:

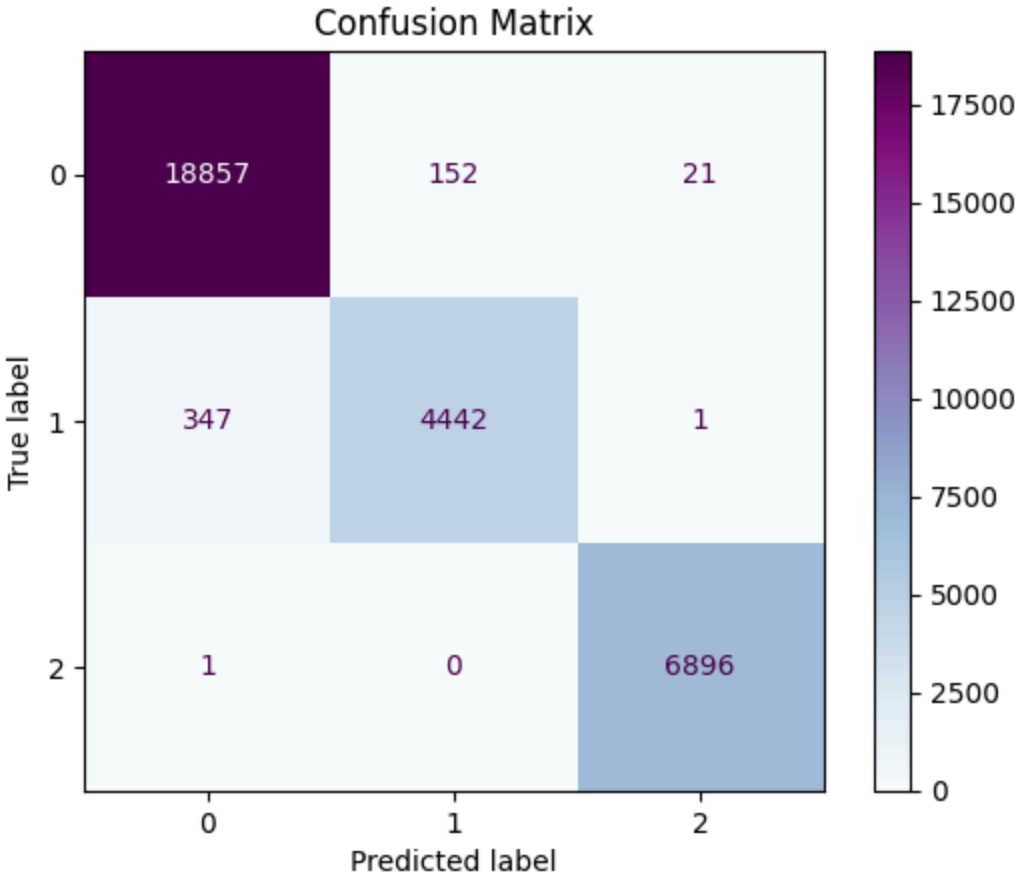
```
# Pipeline 8: Voting with several classifiers
vote8 = VotingClassifier(
    estimators=[('gb', gb), ('ada', ada), ('dt', dt), ('rf', rf), ('svm', sv
    voting='soft',
    n_jobs=-1
)

pipe8 = Pipeline([
    ('pre', preprocess_pipe),
    ('vote', vote8)
])

pipe8.fit(X_train, y_train)

y_pred8 = pipe8.predict(X_test)
f1_macro8 = f1_score(y_test, y_pred8, average='macro')
print(f"Macro F1-score: {f1_macro8:.4f}")

print("\n Classification Report:")
print(classification_report(y_test, y_pred8, digits=4))
```

```
Macro F1-score: 0.9697

 Classification Report:
              precision    recall  f1-score   support

           0     0.9771    0.9867    0.9819      9515
           1     0.9523    0.9086    0.9299      2395
           2     0.9948    0.9997    0.9973      3449

    accuracy                         0.9774     15359
   macro avg     0.9747    0.9650    0.9697     15359
weighted avg     0.9772    0.9774    0.9772     15359
```

**Pipeline 8 score 0.977 on Kaggle.**

**Comment**: The highest scored model is using Stacking Classfier, which we are aware that it is not on the list of DAT200's ensemble classifier. But it gave the best prediction and basically it trains the data on multiple base models and use the their prediction as another data for a final model to make the final prediction. It scored 0.978 on Kaggle, just slightly 0.1% higher than the pipeline 8. So we will continue with pipeline 8 as our best model in this assignment.

In [54]:
```
# Split the data again with 60:40 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, str

y_pred = pipe8.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='BuPu', values_format='d')
plt.title("Confusion Matrix")
plt.show()
```




**Comment**: Very accurate prediction on class 2 - STAR but noticable missclassifications between class 0 and 1 (347 + 152 samples)

In [50]:
```
df_test = pd.read_csv('test.csv')
test_prediction = pipe8.predict(df_test)
y_test_kaggle = pd.DataFrame(test_prediction, columns=["class"])
y_test_kaggle.index.name = "ID"
y_test_kaggle[['class']].to_csv("kaggle.csv")
```

In [ ]:
```

```