

COMP5360 Spring 2022: Final Project

Optical-Spectroscopic Object Classification

Your Data

First Name: Roanna

Last Name: Rague

E-mail: u1343169@umail.utah.edu

UID: u1343169

First Name: Alexander

Last Name: Millar

E-mail: u0740821@umail.utah.edu

UID: u0740821

Project Description

Our project in summary is the classification of astronomical objects using Python. The ultimate goal of the project is to be able to pass in specific astronomical data and have our Python model return what type of astronomical object it is. The classifications of astronomical objects we chose to have is: Galaxy, Star and Quasar. When passing in the specific data points, our model should return to us what exactly the astronomical object is out of those three classifications. With this in mind, we also hope to try our model out when it is complete with other data to see if it can accurately return the correct classification of the astronomical object.

We are using data from https://www.sdss.org/dr17/data_access/bulk and <http://skyserver.sdss.org/dr17> to use for our data model. This data includes numerous data points, including right ascension angle, declination angle, ultraviolet filter, green filter, red filter, etc. With this data we can construct the classification system we need to determine the final object.

We plan to use dimensionality reduction, such as PCA/LDA, as well as cross-validated classification/clustering methods – such as SVM (Support Vector Machine) and K-means – for our analysis methods on this data and to build our model.

In summary, the question we are trying to answer – our hypothesis – is, what astronomical objects do we have based on the data we have been given?

Data Acquisition

Assessment of possible data sources

Data Description

Our data set contains numerous data points including right ascension angle, declination angle, ultraviolet filter, green filter, red filter, etc

Flexible Image Transport System (FITS)

Our first process for acquiring the data from https://www.sdss.org/dr17/data_access/bulk involved researching and locating .fits files (Flexible Image Transport System – defines digital file format useful for storage). However, with this process, we were unable to determine exactly what the .fits files contained. For example, we were unsure with our initial data processing if one .fits file represented one data object or multiple.

Source in our Repository: "comp5360_final\EDA\Python\Read.py"

```
In [ ]: # #conda install astropy
# import astropy
# from astropy.io import fits
# from astropy.table import Table

# with fits.open('spec-10000-57346-0001.fits') as hdul:
#     print(hdul.info())
#     print(hdul[1].header)
#     print(hdul[1].data)
```

Web Scraping

Our second process for acquiring data from https://www.sdss.org/dr17/data_access/bulk involved web scraping. Since we knew the data was stored here we were able to locate the specific astronomical objects here: <http://skyserver.sdss.org/dr17/VisualTools>.

Upon accessing this page, we noticed there were individual links to the "PhotoObj" (imaging) and the "SpecObj" (spectra)



When we followed these links, we saw that the data was constructed as HTML tables. HTML

So, we decided we would web scrape our data using these HTML tables, store the information and write it all out to a CSV using Python.

However, we did run into some issues:

Issue#1: We first checked to see if we could web scrape this website and when we noticed they had a specific API, we continued with our process. The first issue we noticed is that we could not ping the website too much, so we set up a timer to wait a second or two before querying the website again. One person queried spectral data; the other person queried image data. Once all our data had been queried, we would merge both CSVs together to have one large dataset of both the image and spectral data.

Issue#2: After querying a few of the HTML pages, we noticed if we programmatically accessed the webpage, it would not return the correct data. Even typing in the URL by hand resulted in incorrect data. For example, where there should have been a value a "0" was always returned. At this point, we determined we had to change our course for acquiring the data again.

Source in our Repository:

"comp5360_final\data_acquisition\Python\web_scraping_optical_spectra_and_imagir

```
In [261...]: # %% imports
# from bs4 import BeautifulSoup
# from selenium import webdriver
# import time
# from os import listdir
# from os import chdir
# import pandas as pd

# %% SWITCHES

# #folder containing repo root
# repo_path = r'C:\Users\Bob\Desktop\U\comp5360_final'
# #repo_path = r'/Users/roannarague/Documents/BMI_6106_Final'

# #
# web_driver_path = 'C:/Users/Bob/Downloads/chromedriver_win32/chromedriver.exe'
# #web_driver_path = '/Users/roannarague/Downloads/chromedriver_win32/chromedriver.exe'

# # set FALSE for imaging data
# #want_optical_spectra = True
# want_optical_spectra = False

# %% working directory
# chdir(repo_path)
# chdir('.\data_acquisition\Python')

# %% data temp/out dirs
# web_scraping_dir = '../../data/web_scraping'
# data_folder = 'optical_spectra_data' if want_optical_spectra else 'imaging_data'
# data_out_dir = f'{web_scraping_dir}/{data_folder}'
# temp_html_dir = f'{web_scraping_dir}/temp_html'

# %% out file
# data_type = 'optical_spectra' if want_optical_spectra else 'imaging_data'
# out_csv = f'{data_out_dir}/{data_type}_first_10000.csv'

# %% read in reference data
```

```

# df = pd.read_csv('../data/star_classification.csv')

# %% scrape table from each spec_obj_ID's page in 1 second intervals and write to file
# optical_spectra_url = 'http://skyserver.sdss.org/dr17/VisualTools/explore/displayresults?name='
# imaging_url = 'https://skyserver.sdss.org/dr17/VisualTools/explore/displayresults?name='
# spec_obj_ID_param = '&spec=' # common parameter
# obj_ID_param = '&id=' # imaging parameter

# abort = False
# no_data = []
# with webdriver.Chrome(web_driver_path) as driver:
#     for ids in df.loc[0:10000,['spec_obj_ID','obj_ID']].to_numpy():
#         specObjId = int(ids[0])
#         objId = int(ids[1])
#         query = ''
#         if want_optical_spectra:
#             query = f'{optical_spectra_url}{spec_obj_ID_param}{specObjId}'
#         else:
#             query = f'{imaging_url}{obj_ID_param}{objId}{spec_obj_ID_param}{specObjId}'
#         try:
#             driver.get(query)
#             table = driver.find_element_by_xpath('//table')
#             table_html = table.get_attribute('innerHTML')
#             with open(f'{temp_html_dir}/{specObjId}.html', 'w') as new_file:
#                 new_file.write(table_html)
#         except:
#             no_data.append(specObjId)
#             time.sleep(1)
#             if not abort:
#                 continue
#             break

#     with open(f'{data_out_dir}/no_data_spec_Ids_first_10000.txt', 'w') as nd_file:
#         nd_file.write(f'{no_data}')


# %% List all files in saved HTML directory
# pages = []
# for file in listdir(temp_html_dir):
#     with open(f'{temp_html_dir}/{file}', encoding='utf-8') as f:
#         pages.append(BeautifulSoup(f.read(), 'html.parser'))


# %% hacky way to visualize first page contents
# Len(pages)
# pg = pages[0]
# pg.text.replace('\n', ' ').replace(' ', ',')[2:3424].replace(' ', ': ').split(',') # replace commas with colons

# %% extract key/value pairs from each page into list of dicts
# spec_obj_dictionaries = []
# pg_idx = 0
# for page in pages:
#     try:
#         pdict = {}
#         for row in page.select('tr'):
#             field = row.select('tr > th > span')[0].text
#             value = row.select('tr > td')[0].text
#             if field == 'img':

```

```

#             break
#             pdict[field]=value
#             spec_obj_dictionaries.append(pdict)
#         except Exception as e:
#             print(f'Broke on page index {pg_idx} due to {str(e)}')
#             pg_idx += 1

# #%% check successful scrapes weren't all GALAXY
# sdf = pd.DataFrame(spec_obj_dictionaries)
# #print(sdf['class'].value_counts()) not all team members' scrape designations have cl
# #sdf['spec_obj_ID']...

# # TODO: RO PR -> df.iloc[0] =>
# # obj_ID          1237660961327743232.0
# # alpha           135.689107
# # delta           32.494632
# # u               23.87882
# # g               22.2753
# # r               20.39501
# # i               19.16573
# # z               18.79371
# # run_ID          3606
# # rerun_ID        301
# # cam_col         2
# # field_ID        79
# # spec_obj_ID     6543777369295181824.0
# # class           GALAXY
# # redshift        0.634794
# # plate            5812
# # MJD              56354
# # fiber_ID         171

# # 6543777369295181824.html written to disk has specObjID of 0
# #
# # fortunately, the file name has this ID and in memory list of soup,
# # but I am concerned about the values
# # (e.g. 'u'=23.87882' (above) vs sdf.iloc[0].u=='23.78062')
# # in cases where we get this specObjId

# #%% save to disk for team merge
# sdf.to_csv(out_csv)

# #%% check
# test = pd.read_csv(out_csv)

```

Web Querying

Finally our third process for acquiring data from https://www.sdss.org/dr17/data_access/bulk, involved using a special tool on the website:

<https://skyserver.sdss.org/dr17/CrossMatchTools/ObjectCrossID>. This tool allowed us to query the data directly on the website using SQL. We were able to check the correct search types we needed and write the data out as a final CSV file.



However, we also ran into some issues:

Issue#1: The data returned resulted in massive files, so we had to do 10 runs of this query to get all our data. Then we merged all the CSV files together

Issue#2: When we had our final dataset, we noticed there were only two classifications: stars and galaxies. But no quasars. For this database, these were the only two classifications. To fix this issue, we used: <https://skyserver.sdss.org/dr17/SearchTools/sql#> to write a direct SQL query for Quasars. This returned a CSV file of just quasars where we then matched the "spec_obj_id" with the "spec_obj_id" in our initial dataset (with just galaxies and stars) and re-classified any of them that had a matching "spec_obj_id" in the quasar CSV.



Source in our Repository:

"comp5360_final\data_acquisition\Python\web_query_file_builder.py"

In [259...]

```
# %% Imports

# import numpy as np
# import pandas as pd
# from os import chdir

# %% working directory

# folder containing repo root
# repo_path = r'C:\Users\Bob\Desktop\U\comp5360_final'
# repo_path = r'/Users/roannarague/Documents/BMI_6106_Final'

# chdir(repo_path)

# chdir('./data_acquisition/Python')

# %% read in reference data
# df = pd.read_csv('../..../data/star_classification.csv')

# %%% build
# out_dir = '../..../data/skyserver_querying'

# num_chunks = 10
# num_objs = df.shape[0]
# chunk_size = int(num_objs / num_chunks)

# for i in range(num_chunks):
#     ra_dec = []
#     ra_dec.append(' name ra dec\n')
#     idx = 0
#     start = chunk_size * i
#     end = start + chunk_size - 1
#     if num_chunks == i + 1:
#         end += 1
#     for ids in df.loc[start:end,['alpha', 'delta']].values:
#         idx += 1
```

```

#         xname = hex(idx)[2:]
#         ra_dec.append(f' {xname}:<4> {ids[0]:<19>} {ids[1]}\n')

#     with open(f'{out_dir}/ra_dec_{start}-{end}.txt', 'w') as f:
#         f.writelines(ra_dec)

```

Clean up concatenated Web Query data

Check data for issues

Source in our Repository:

"comp5360_final\data_acquisition\Python\clean_and_compare_ref_vs_final_combine

In [263...]

```

# %% Imports

# import numpy as np
# import pandas as pd
# from os import chdir

# %% working directory

# #folder containing repo root
# repo_path = r'C:\Users\Bob\Desktop\U\comp5360_final'
# #repo_path = r'/Users/roannarague/Documents/BMI_6106_Final'

# chdir(repo_path)

# chdir('./data_acquisition/Python')

# %% read in reference data
# df = pd.read_csv('../..../data/star_classification.csv')

# %% clean up join of queries

# def FreshValDf():
#     t = pd.read_csv('../..../data/skyserver_querying/final_combined.csv', skiprows=1)

#     cols = t.columns.to_numpy()
#     # ['name', 'objID', 'ra', 'dec', 'run', 'rerun', 'camcol', 'field', 'type', ...]

#     # drop 'name' per artifact of query
#     t = t.drop('name', axis=1)

#     cols = t.columns.to_numpy()
#     # ['objID', 'ra', 'dec', 'run', 'rerun', 'camcol', 'field', 'type', 'modelMag_u',
#     # 'modelMag_i', 'modelMag_z', 'specObjID', 'plate', 'mjd', 'fiberID']

```

```

#     Len(t.specObjID.unique())
#     #126598

#     t.shape
#     #(128429, 19)

#     t.shape[0] - len(t.specObjID.unique())
#     #1831

#     val_cnts = t.specObjID.value_counts()
#     val_cnts.reset_index().iLoc[0]
#     # index      specObjID
#     # specObjID      9

#     #remove header rows (from joining queries)
#     rows_to_drop_idx = t[t['specObjID']=='specObjID'].index
#     t = t.drop(rows_to_drop_idx)

#     #check ref and val data for matching spec_obj_IDs

#     # all duplicated specObjID rows have same values in all columns
#     t.duplicated().sum()
#     # 1823 (doesn't include first occurrence)
#     len(t) - t.duplicated().sum()
#     # 128420 - 1823 = 126597

#     t = t.drop(t.loc[t.duplicated()].index)

#     Len(t)
#     # 126597
#     len(t['specObjID'].unique())
#     # 126597
#     return t

# tdf = FreshValDf()

# %% subset ref and val data by matching spec_obj_IDs

# ref_spec_obj_ids = df['spec_obj_ID'].to_numpy()
# val_spec_obj_ids = tdf['specObjID'].astype(float).to_numpy()

# spec_obj_ids_from_val_in_ref = val_spec_obj_ids[np.in1d(val_spec_obj_ids, ref_spec_obj_ids)]
# len(spec_obj_ids_from_val_in_ref)
# #71155

# ref_spec_obj_ids = df['spec_obj_ID'].apply(lambda x: str(int(x))).to_numpy()
# val_spec_obj_ids = tdf['specObjID'].to_numpy()

# spec_obj_ids_from_val_in_ref = val_spec_obj_ids[np.in1d(val_spec_obj_ids, ref_spec_obj_ids)]
# len(spec_obj_ids_from_val_in_ref)
# #71155

# # subset queried data (common soi w/ref data)
# df_val_subset = tdf[tdf['specObjID'].isin(spec_obj_ids_from_val_in_ref)]
# df_val_subset.shape

# # subset ref data
# soi_float = spec_obj_ids_from_val_in_ref.astype(float)

```

```

# df_ref_subset = df[df['spec_obj_ID'].isin(soi_float)]
# df_ref_subset.shape

# %% Look at data types/values to decide what makes sense

# def prnt(df):
#     print()
#     idx = df.dtypes.index
#     dt = df.dtypes
#     val = df.iloc[0]
#     for i in range(len(val)):
#         typ = str(dt[i])
#         print(f'{typ:<10} {idx[i]:<15}{val[i]:>24}')
#     print()

# prnt(df_ref_subset)
# # float64    obj_ID          1.2376609613277432e+18
# # float64    alpha           135.6891066036
# # float64    delta           32.4946318397087
# # float64    u                23.87882
# # float64    g                22.2753
# # float64    r                20.39501
# # float64    i                19.16573
# # float64    z                18.79371
# # int64     run_ID           3606
# # int64     rerun_ID          301
# # int64     cam_col           2
# # int64     field_ID          79
# # float64   spec_obj_ID      6.543777369295182e+18
# # object    class             GALAXY
# # float64   redshift          0.6347936
# # int64    plate              5812
# # int64    MJD                56354
# # int64    fiber_ID           171

# prnt(df_val_subset)
# # object    objID            1237660961327743273
# # object    ra               135.6891066036
# # object    dec              32.4946318397087
# # object    run              3606
# # object    rerun             301
# # object    camcol            2
# # object    field              79
# # object    type              GALAXY
# # object    modelMag_u        23.87882
# # object    modelMag_g        22.2753
# # object    modelMag_r        20.39501
# # object    modelMag_i        19.16573
# # object    modelMag_z        18.79371
# # object    specObjID         6543777369295181824
# # object    plate              5812
# # object    mjd                56354
# # object    fiberID            171
# # object    z                  0.6347936

# modelMag ~ (de Vaucouleurs magnitude fit || Exponential fit magnitude), better of DeV
# %% match up
# def ConvertToMatchRef(d):

```

```

#     d['redshift'] = d['z']
#     d['class'] = d['type']
#     d['spec_obj_ID'] = d['specObjID']
#     d['obj_ID'] = d['objID']
#     d['alpha'] = d['ra'].astype(float)
#     d['delta'] = d['dec'].astype(float)
#     d['u'] = d['modelMag_u'].astype(float)
#     d['g'] = d['modelMag_g'].astype(float)
#     d['r'] = d['modelMag_r'].astype(float)
#     d['i'] = d['modelMag_i'].astype(float)
#     d['z'] = d['modelMag_z'].astype(float)

#
#     d = d.drop(['type'], axis=1)
#     d = d.drop(['ra'], axis=1)
#     d = d.drop(['dec'], axis=1)
#     d = d.drop(['modelMag_u'], axis=1)
#     d = d.drop(['modelMag_g'], axis=1)
#     d = d.drop(['modelMag_r'], axis=1)
#     d = d.drop(['modelMag_i'], axis=1)
#     d = d.drop(['modelMag_z'], axis=1)

#
#     d['run_ID'] = d['run'].astype(int)
#     d['rerun_ID'] = d['rerun'].astype(int)
#     d['cam_col'] = d['camcol'].astype(int)
#     d['field_ID'] = d['field'].astype(int)
#     d['plate'] = d['plate'].astype(int)
#     d['MJD'] = d['mjd'].astype(int)
#     d['fiber_ID'] = d['fiberID'].astype(int)

#
#     d = d.drop(['objID', 'specObjID', 'run',
#                 'rerun', 'camcol', 'field',
#                 'mjd', 'fiberID'], axis=1)
#     return d

# #%%
# df_val_subset = ConvertToMatchRef(df_val_subset)

#
# %% check
# print(df_val_subset)
# print(df_ref_subset)
# print(df_val_subset.shape)
# print(df_ref_subset.shape)

#
# %% write
# df_val_subset.to_csv('final_combined_clean_subset_matching_ref.csv')

#
# %% NO QUASARS?
# df_val_subset['class'].value_counts()

#
# #%%
# print(df['class'].value_counts())

#
# %% YUP, NO QUASARS!

# df_val = FreshValDf()
# df_val['type'].value_counts()

```

```

# df_val = df_val.drop(df_val[df_val.isna().any(axis=1)].index)
# ddd = ConvertToMatchRef(df_val)
# prnt(ddd)
# print(ddd['class'].value_counts())

# t = pd.read_csv('../data/skyserver_querying/final_combined.csv', skiprows=1)
# t['type'].value_counts()

```

Found that we had no Quasar data

Peer Review verified no Quasar data

Queried <https://skyserver.sdss.org/dr17/CrossMatchTools/ObjectCrossID> for only Quasars ("Quasars2.csv")

Matched spec_obj_ID to previously queried astronomical objects (Quasars mostly over-generalized as STAR)

Source in our Repository:

"comp5360_final\data_acquisition\Python\SetQuasars.py"

```

In [ ]:
# ## Part 1a: Read in initial final csv (the one with no Quasars)

# # imports and setup
# import pandas as pd
# import numpy as np
# import os

# #set working directory
# os.chdir('/Users/roannarague/Downloads')
# #os.chdir('/Users/Bob/Desktop/u/comp5360_final/data_acquisition/Python/')

# #read in the final_combined csv (with no quasars)
# # renamed final_combined_clean_subset_matching_ref.csv to shorten
# df_combined = pd.read_csv('/Users/roannarague/Downloads/final_combined_3.csv')
# #df_combined = pd.read_csv('./final_combined_3.csv')

# #print a few lines
# #df_combined.head()

# #check out all our spec ob IDs
# #print(df_combined['spec_obj_ID'])

# ## PART 1b: Read in our ALL Quasar CSV data

# #read in the final_combined csv (with no quasars)
# df_qso = pd.read_csv('/Users/roannarague/Downloads/Quasars2.csv')
# #df_qso = pd.read_csv('./Quasars2.csv')

# #print a few lines
# #df_qso.head()

# #print columns
# #print(df_qso.columns.tolist())

```

```

# #create new dataframe of just spec obj ids
# df_specs = pd.DataFrame().assign(specObjID=df_qso['specObjID'])

# #df_specs.head()

# #create a numpy array out of this new dataframe
# specID_array = df_specs.to_numpy()

# #print(specID_array)

# # PART 2: Iterate through our final_combined_3.csv to find specObjIDs that match our

# ## PART 2a: iterate through our df_combined dataframe and start comparing specObjIDs

# #iterate through our df_combined dataframe

# #create a variable to store our first quasar spec id we find so we can validate later
# firstQSOspecID = 0

# #print("===== SpecObjID ====== Class =====")
# for index, row in df_combined.iterrows():
#     #print("index: ", index, " --> ", row['spec_obj_ID'], " ", row['class'])
#     #print(row['spec_obj_ID'])
#     if row['spec_obj_ID'] in specID_array:
#         firstQSOspecID = row['spec_obj_ID']

#         #print our old row
#         #print("===== ROW =====")
#         #print("OLD Row: ", "index: ", index, "Row: ", row['spec_obj_ID'], " CLASS: "
#               #so now we have to change the class type from whatever it was to QSO == Quasar
#               df_combined.at[index,'class']='QSO'

#         #print our new row
#         #print("NEW Row: ", "index: ", index, "Row: ", row['spec_obj_ID'], " CLASS: "

# #DOUBLE CHECKING DOWN HERE to make sure that it actually was in the quasar numpy array
# #for i in specID_array:
#     #if i == firstQSOspecID:
#         #print("Successful Test #1!")
#         #print(i, " == ", firstQSOspecID)

# # PART 3: Write to a new final_combined_QSO.csv file

# ## PART 3a: Write a new csv file with Quasars in it

# #first let's take a look at our dataframe head again:
# #for index, row in df_combined.iterrows():
#     #if row['class'] == 'QSO':
#         #print("index: ", index, " --> ", row['spec_obj_ID'], " ", row['class'])

# #Write to a csv file
# df_combined.to_csv('/Users/roannarague/Downloads/final_combined_QSO_1.csv')
# #df_combined.to_csv('./final_combined_QSO_1.csv')

```

```

# # PART 4: re-read in our new combined csv file with quasars

# #read in the final_combined csv (with quasars)
# df_combined_QSO = pd.read_csv('/Users/roannarague/Downloads/final_combined_QSO_1.csv')
# #df_combined_QSO = pd.read_csv('./final_combined_QSO_1.csv')

# #print a few Lines
# #df_combined_QSO.head()
# print(df_combined_QSO.groupby('class').count())

# df_combined['class'].value_counts()

```

Remove index columns and exclude in output CSV

Query 100,000 Optical Spectroscopic Objects for Test

Lastly, we queried 100,000 objects with a fair degree of similarity--ensuring GALAXY/STAR/QUASAR-to our train objects for model test.

Source in our Repository: "comp5360_final\data\star_classification_test_readme.txt"

In [50]:

```

# source:
# https://skyserver.sdss.org/dr17/SearchTools/sql

# SQL:
# -- This query does a table JOIN between the imaging (PhotoObj) and spectra
# --(SpecObj) tables and includes the necessary columns in the SELECT to upload
# --the results to the SAS(Science Archive Server) for FITS file retrieval.
# SELECT TOP 100000
# p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
# p.run, p.rerun, p.camcol, p.field,
# s.specobjid, s.class, s.z as redshift,
# s.plate, s.mjd, s.fiberid
# FROM PhotoObj AS p
# JOIN SpecObj AS s ON s.bestobjid = p.objid
# WHERE
#   p.u BETWEEN 10.99 AND 32.8
#   AND g BETWEEN 10.49 AND 31.61
#   AND p.i BETWEEN 9.45 AND 32.15
#   AND p.z BETWEEN 9.6 AND 29.39
#   AND s.z BETWEEN -0.0099 AND 7.012
#   AND s.mjd BETWEEN 51607 AND 58933

# SELECT TOP 50000
# p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
# p.run, p.rerun, p.camcol, p.field,
# s.specobjid, s.class, s.z as redshift,
# s.plate, s.mjd, s.fiberid
# FROM PhotoObj AS p
# JOIN SpecObj AS s ON s.bestobjid = p.objid
# WHERE
#   p.u BETWEEN 10.99 AND 32.8
#   AND g BETWEEN 10.49 AND 31.61
#   AND p.i BETWEEN 9.45 AND 32.15

```

```
#    AND p.z BETWEEN 9.6 AND 29.39  
#    AND s.z BETWEEN -0.0099 AND 7.012  
#    AND s.mjd BETWEEN 51607 AND 58933
```

Source in our Repository: "comp5360_final\data\star_classification_test.csv"

Source in our Repository: "comp5360_final\data\star_classification_test_500000.csv"

Final notes: After our process of acquiring data, we finally retrieved a usable dataset of ~100,000 objects to use for building our Python model.

Ethical Data Concerns

Something to consider would be why some of the data was classified as a quasar and then as a galaxy/star? As mentioned above, the classification rules change over time. Get a percentage of how many astronomical objects were later classified as quasar instead of a star/galaxy.

Methods

For our model, we are going to use Python and we are going to utilize the following methods:

- Dimensionality reduction – PCA/LDA
 - Cross validated classification/clustering -- SVM and K-means
-

Dimensionality Reduction

Dimensionality reduction is important to visualize and understand high dimensional data. For example, we have up to 17 dimensions to potentially use to predict our 1 dimension of `class`. Additionally, dimensionality reduction can help with feature selection by clustering the reduced dimensions.

In [81]:

```
# imports and setup  
  
import numpy as np  
  
import pandas as pd  
pd.set_option('display.notebook_repr_html', False)  
  
from sklearn.decomposition import PCA  
from sklearn.manifold import TSNE  
from sklearn.preprocessing import scale
```

```

from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import homogeneity_score, v_measure_score, classification_report
from sklearn.feature_selection import mutual_info_classif
from sklearn.neighbors import LocalOutlierFactor
from sklearn.model_selection import train_test_split, cross_val_score, cross_validate
from sklearn import svm, metrics

from imblearn.over_sampling import SMOTE

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'], 'GALAXY, QSO, STAR')
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
plt.rcParams['figure.figsize'] = (10, 6)
plt.style.use('ggplot')

import seaborn as sns

import plotly.express as px

```

Load Data

In [82]: `df = pd.read_csv('data/star_classification.csv')`

In [83]: `df.columns.values`

Out[83]: `array(['obj_ID', 'alpha', 'delta', 'u', 'g', 'r', 'i', 'z', 'run_ID',
 'rerun_ID', 'cam_col', 'field_ID', 'spec_obj_ID', 'class',
 'redshift', 'plate', 'MJD', 'fiber_ID'], dtype=object)`

Variables

- `obj_ID`
 - unique identifier for the object in the image catalog used by CAS (Catalog Archive Server)
- `alpha`
 - right ascension angle (similar to longitude)
- `delta`
 - declination angle (similar to latitude)
- `u`
 - ultraviolet filter (photometric system)
- `g`
 - green filter (photometric system)
- `r`
 - red filter (photometric system)
- `i`
 - near infrared filter (photometric system)
- `z`

- infrared filter (photometric system)
- run_ID
 - identifier for the specific scan
- rerun_ID
 - identifier specifying how the image was processed
- cam_col
 - camera column (identifies the scanline within run)
- field_ID
 - identifier specifying each field
- spec_obj_ID
 - unique optical spectroscopic object identifier
- class
 - class of optical spectroscopic object (e.g. GALAXY, STAR, QUASAR)
- redshift
 - based on the increase in wavelength
- plate
 - identifier of plate in SDSS
- MJD
 - modified julian date (date of observation)
- fiber_ID
 - identifier of the fiber pointing light at the focal plane (for observation)

What Variables May Contain Information

In [84]: `df.describe()`

	obj_ID	alpha	delta	u	\
count	1.000000e+05	100000.000000	100000.000000	100000.000000	
mean	1.237665e+18	177.629117	24.135305	21.980468	
std	8.438560e+12	96.502241	19.644665	31.769291	
min	1.237646e+18	0.005528	-18.785328	-9999.000000	
25%	1.237659e+18	127.518222	5.146771	20.352353	
50%	1.237663e+18	180.900700	23.645922	22.179135	
75%	1.237668e+18	233.895005	39.901550	23.687440	
max	1.237681e+18	359.999810	83.000519	32.781390	
	g	r	i	z	\
count	100000.000000	100000.000000	100000.000000	100000.000000	
mean	20.531387	19.645762	19.084854	18.668810	
std	31.750292	1.854760	1.757895	31.728152	
min	-9999.000000	9.822070	9.469903	-9999.000000	
25%	18.965230	18.135828	17.732285	17.460677	
50%	21.099835	20.125290	19.405145	19.004595	
75%	22.123767	21.044785	20.396495	19.921120	
max	31.602240	29.571860	32.141470	29.383740	
	run_ID	rerun_ID	cam_col	field_ID	spec_obj_ID
count	100000.000000	100000.0	100000.000000	100000.000000	1.000000e+05
mean	4481.366060	301.0	3.511610	186.130520	5.783882e+18
std	1964.764593	0.0	1.586912	149.011073	3.324016e+18
min	109.000000	301.0	1.000000	11.000000	2.995191e+17

25%	3187.000000	301.0	2.000000	82.000000	2.844138e+18
50%	4188.000000	301.0	4.000000	146.000000	5.614883e+18
75%	5326.000000	301.0	5.000000	241.000000	8.332144e+18
max	8162.000000	301.0	6.000000	989.000000	1.412694e+19
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	0.576661	5137.009660	55588.647500	449.312740	
std	0.730707	2952.303351	1808.484233	272.498404	
min	-0.009971	266.000000	51608.000000	1.000000	
25%	0.054517	2526.000000	54234.000000	221.000000	
50%	0.424173	4987.000000	55868.500000	433.000000	
75%	0.704154	7400.250000	56777.000000	645.000000	
max	7.011245	12547.000000	58932.000000	1000.000000	

`rerun_ID` is constant (`std == 0`) and can thereby be removed

`obj_ID` & `spec_obj_ID`, aside from having high Standard Deviation (`std`) are assigned to specific parent objects and spectroscopic objects for identification. These values are arbitrary and do not contain observable information about the underlying object being observed. Additionally, they may have some relation to the class of the object and thereby have the potential to confound downstream analyses/predictions and should therefore be dropped.

In [85]:

```
df = df.drop('rerun_ID', axis=1)
df = df.drop('obj_ID', axis=1)
df = df.drop('spec_obj_ID', axis=1)
```

Univariate Outlier Detection

In [86]:

```
df.describe()
```

Out[86]:

count	100000.000000	100000.000000	100000.000000	100000.000000	\
mean	177.629117	24.135305	21.980468	20.531387	
std	96.502241	19.644665	31.769291	31.750292	
min	0.005528	-18.785328	-9999.000000	-9999.000000	
25%	127.518222	5.146771	20.352353	18.965230	
50%	180.900700	23.645922	22.179135	21.099835	
75%	233.895005	39.901550	23.687440	22.123767	
max	359.999810	83.000519	32.781390	31.602240	
count	100000.000000	100000.000000	100000.000000	100000.000000	\
mean	19.645762	19.084854	18.668810	4481.366060	
std	1.854760	1.757895	31.728152	1964.764593	
min	9.822070	9.469903	-9999.000000	109.000000	
25%	18.135828	17.732285	17.460677	3187.000000	
50%	20.125290	19.405145	19.004595	4188.000000	
75%	21.044785	20.396495	19.921120	5326.000000	
max	29.571860	32.141470	29.383740	8162.000000	
count	100000.000000	100000.000000	100000.000000	100000.000000	\
mean	3.511610	186.130520	0.576661	5137.009660	
std	1.586912	149.011073	0.730707	2952.303351	

```

min          1.000000    11.000000   -0.009971   266.000000
25%         2.000000    82.000000    0.054517  2526.000000
50%         4.000000   146.000000    0.424173  4987.000000
75%         5.000000   241.000000    0.704154  7400.250000
max          6.000000   989.000000    7.011245 12547.000000

```

	MJD	fiber_ID
count	100000.000000	100000.000000
mean	55588.647500	449.312740
std	1808.484233	272.498404
min	51608.000000	1.000000
25%	54234.000000	221.000000
50%	55868.500000	433.000000
75%	56777.000000	645.000000
max	58932.000000	1000.000000

u , g , and z have an outstanding min of -9999 , investigate how prevalent

In [87]:

```
df[((df.u===-9999) | (df.g===-9999) | (df.z===-9999))]
```

Out[87]:

	alpha	delta	u	g	r	i	z	\
79543	224.006526	-0.624304	-9999.0	-9999.0	18.1656	18.01675	-9999.0	
	run_ID	cam_col	field_ID	class	redshift	plate	MJD	fiber_ID
79543	752	2	537	STAR	0.000089	3314	54970	162

It appears there is a single outlier, which will heavily impact a variety of parametric methods (e.g. PCA).

What type of effect is the having on the relationship of the variables? Should we drop the outlier?

Correlation Matrix with and without Outlier

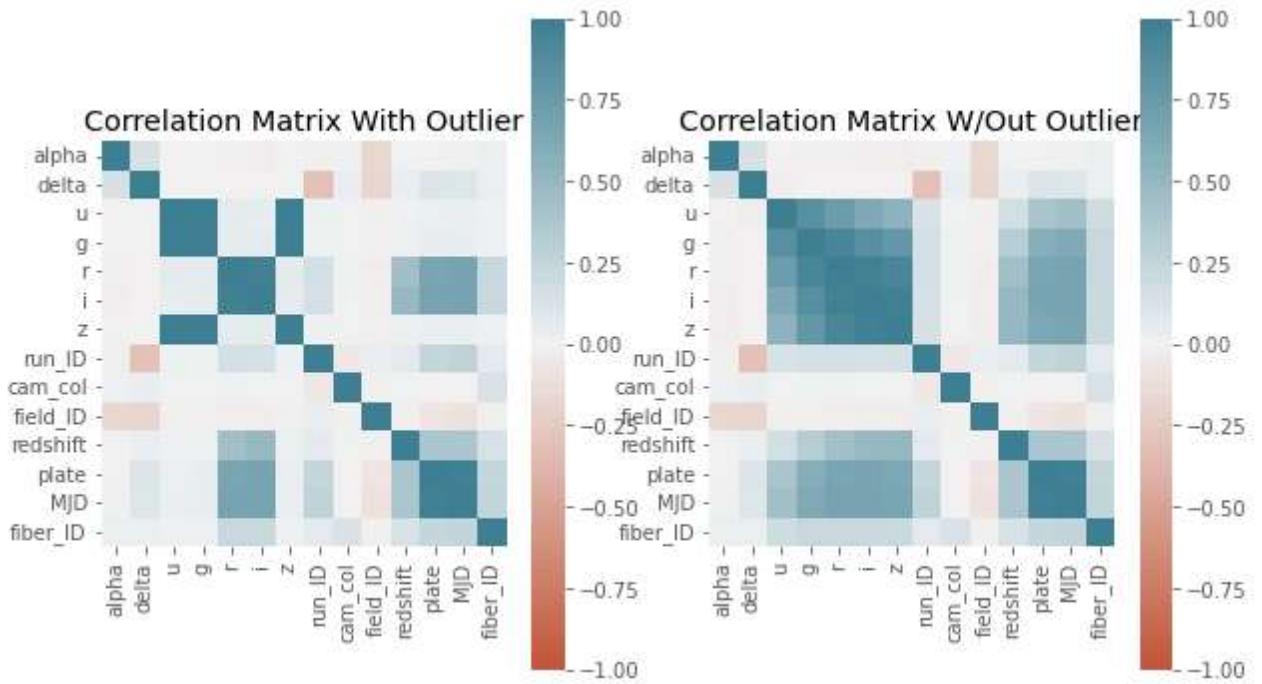
In [88]:

```

rb_diverging = sns.diverging_palette(20,220, n=200)
fig,axs = plt.subplots(1,2)

sns.heatmap(df.corr(),
            center=0,
            vmin=-1,
            vmax=1,
            square=True,
            cmap=rb_diverging,
            ax=axs[0]).set_title('Correlation Matrix With Outlier')
sns.heatmap(df.drop(79543).corr(),
            center=0,
            vmin=-1,
            vmax=1,
            square=True,
            cmap=rb_diverging,
            ax=axs[1]).set_title('Correlation Matrix W/Out Outlier')
print()

```



We can see that the outlier is having a significant impact on the relationship among variables and should be dropped

```
In [89]: df = df.drop(79543)
df.describe()
```

	alpha	delta	u	g	r	\
count	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000	
mean	177.628653	24.135552	22.080679	20.631583	19.645777	
std	96.502612	19.644608	2.251068	2.037384	1.854763	
min	0.005528	-18.785328	10.996230	10.498200	9.822070	
25%	127.517698	5.147477	20.352410	18.965240	18.135795	
50%	180.900527	23.646462	22.179140	21.099930	20.125310	
75%	233.895005	39.901582	23.687480	22.123775	21.044790	
max	359.999810	83.000519	32.781390	31.602240	29.571860	

	i	z	run_ID	cam_col	field_ID	\
count	99999.000000	99999.000000	99999.000000	99999.000000	99999.000000	
mean	19.084865	18.768988	4481.403354	3.511625	186.127011	
std	1.757900	1.765982	1964.739021	1.586913	149.007687	
min	9.469903	9.612333	109.000000	1.000000	11.000000	
25%	17.732280	17.460830	3187.000000	2.000000	82.000000	
50%	19.405150	19.004600	4188.000000	4.000000	146.000000	
75%	20.396510	19.921120	5326.000000	5.000000	241.000000	
max	32.141470	29.383740	8162.000000	6.000000	989.000000	

	redshift	plate	MJD	fiber_ID
count	99999.000000	99999.000000	99999.000000	99999.000000
mean	0.576667	5137.027890	55588.653687	449.315613
std	0.730709	2952.312485	1808.492217	272.498252
min	-0.009971	266.000000	51608.000000	1.000000
25%	0.054522	2526.000000	54234.000000	221.000000
50%	0.424176	4987.000000	55869.000000	433.000000

```
75%      0.704172  7400.500000  56777.000000  645.000000
max      7.011245  12547.000000  58932.000000  1000.000000
```

Split y (response) and X (predictors)

In [90]:

```
y = df['class']
X = df.drop(['class'], axis=1)
X.columns.values
```

Out[90]:

```
array(['alpha', 'delta', 'u', 'g', 'r', 'i', 'z', 'run_ID', 'cam_col',
       'field_ID', 'redshift', 'plate', 'MJD', 'fiber_ID'], dtype=object)
```

What Classes are present

In [91]:

```
print(y.value_counts().index.values)
```

```
['GALAXY' 'STAR' 'QSO']
```

The dataset contains 14 features (attributes) corresponding to 3 different types of Optical-Spectroscopic Object.

Features (attributes):

- alpha
- delta
- u
- g
- r
- i
- z
- run_ID
- cam_col
- field_ID
- redshift
- plate
- MJD
- fiber_ID

Classes:

- GALAXY
- STAR
- QSO (QUASAR)

Center/Scale the Data

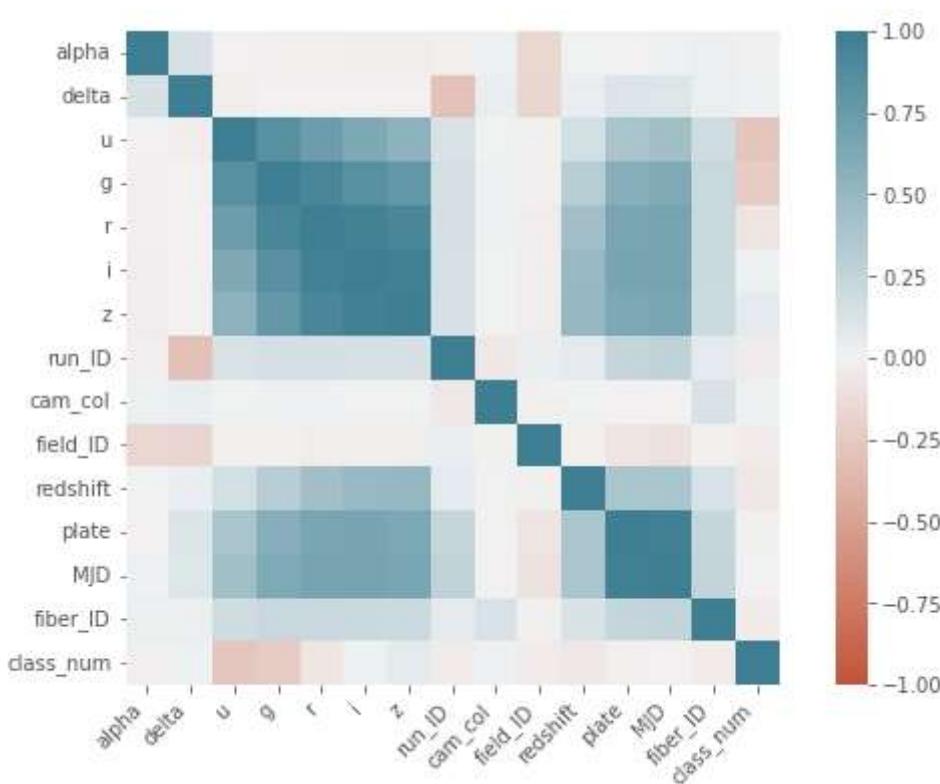
```
In [92]: X_df = pd.DataFrame(X).apply(scale)
X = scale(X)
```

Encode Class

```
In [93]: # hold on to le to allow easy decoding as needed
le = LabelEncoder().fit(y)
y_num = le.transform(y)
```

Check Correlation Matrix Prior to PCA

```
In [94]: X_df['class_num'] = y_num
ax = sns.heatmap(X_df.corr(), center=0, vmin=-1, vmax=1, square=True, cmap=rb_diverging
ax.set_xticklabels(ax.get_xticklabels(), horizontalalignment='right', rotation=45)
X_df = X_df.drop('class_num', axis=1)
print()
```



Remove correlated variables believed to represent same underlying trend, if any, before PCA per will amplify variance and dominate PC1

PCA

Principal Component Analysis (PCA) is an often-used tool in astronomy and stellar classification. It helps with the trial-and-error process, meaning it finds the most interesting linear coreelations of the variables. Can do either 2D or 3D plots.

Need to select correct amount of scaled components

In [95]:

```
pca_scaled_1 = PCA(n_components = 0.95)
pca_scaled_1.fit(X)
reduced_data = pca_scaled_1.transform(X)
print('Original Dimensions: ',X.shape)
print('Reduced Dimensions: ',reduced_data.shape)
```

Original Dimensions: (99999, 14)

Reduced Dimensions: (99999, 9)

So what this means is that 95% of variance is observed by 9 dimensions

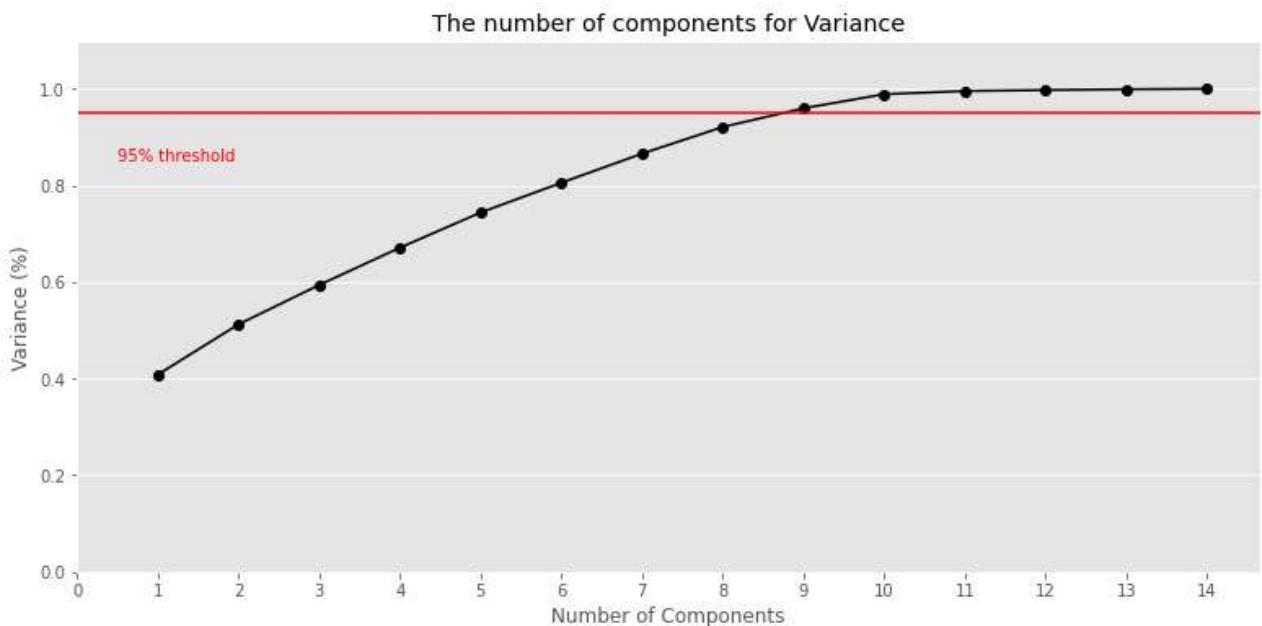
Now we can select the correct amount of components

So let's create a plot to visualize this data, basically I want to see the component cutoff point by creating a plot with our 95% threshold line, and wherever our 'component line' crosses is the amount of components we should choose

Cumulative Variance Plot

In [96]:

```
pca_2_plot = PCA().fit(X)
plt.rcParams["figure.figsize"] = (13,6)
fig, ax = plt.subplots()
plt.ylim(0.0,1.1)
plt.plot(np.arange(1, 15, step=1), np.cumsum(pca_2_plot.explained_variance_ratio_), marker='o')
plt.xlabel('Number of Components')
plt.xticks(np.arange(0, 15, step=1)) #change to 1 based
plt.ylabel('Variance (%)')
plt.title('The number of components for Variance')
plt.axhline(y=0.95, color='r', linestyle='-')
plt.text(0.5, 0.85, '95% threshold', color = 'red', fontsize=10)
ax.grid(axis='x')
plt.show()
```



Create our PCA on our scaled data

```
In [97]: pca = PCA(n_components=9, random_state=37)
X_pca_nine = pca.fit_transform(X)
```

Create a Scree Plot

The scree plot will show us how much variance each component has on the data. This way we can see which components we actually need for our final PCA. Whatever components don't capture most of the information then we can ignore them.

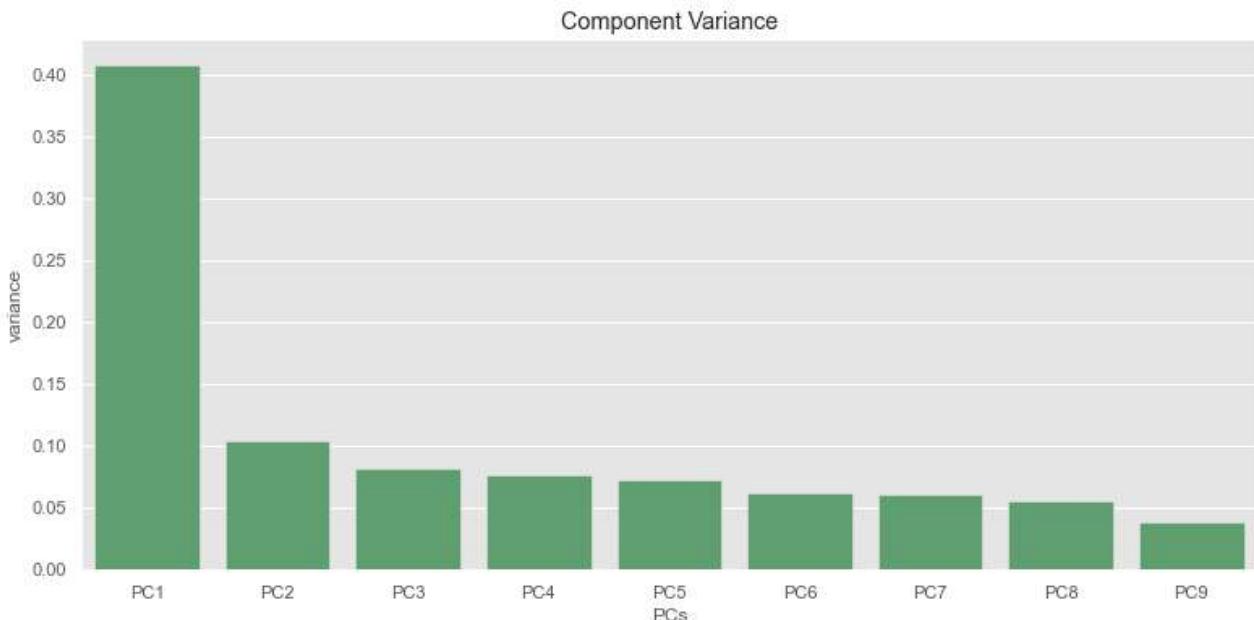
```
In [98]: #print our variance
print('----- VARIANCE -----')
print(pca.explained_variance_)

#now print the ratio
print('----- RATIO -----')
print(pca.explained_variance_ratio_)
```

```
----- VARIANCE -----
[5.71484359 1.45721281 1.14195913 1.08035967 1.02027689 0.86192915
 0.84450327 0.7722561  0.54279451]
----- RATIO -----
[0.40819903 0.10408559 0.08156769 0.07716778 0.07287619 0.06156575
 0.06032106 0.0551606  0.03877065]
```

SCREE PLOT

```
In [139... scree_plot = pd.DataFrame({'variance':pca.explained_variance_ratio_, 'PCs':['PC1','PC2',
sns.barplot(x='PCs',y="variance",data=scree_plot, color="g").set_title('Component Varia
```



Interpretation of Scree Plot:

This shows that it looks like the first 3 components contain the most valuable information of our data.

Take a look at the correlations between the original data and each component

```
In [99]: #create a new data frame with our PCA
pc_rem = pd.DataFrame(data = X_pca_nine,
```

```

columns = ['pc1', 'pc2', 'pc3', 'pc4', 'pc5',
           'pc6', 'pc7', 'pc8', 'pc9'])

#drop the components we established above that we do not need
pc_rem=pc_rem.drop(['pc4'],axis=1)
pc_rem=pc_rem.drop(['pc5'],axis=1)
pc_rem=pc_rem.drop(['pc6'],axis=1)
pc_rem=pc_rem.drop(['pc7'],axis=1)
pc_rem=pc_rem.drop(['pc8'],axis=1)
pc_rem=pc_rem.drop(['pc9'],axis=1)

pc_col = pd.concat([pc_rem, X_df], axis=1)
pc_col
corrMat = pd.DataFrame.cov(pc_col)
sns.set(rc={'figure.figsize':(16,6)})
sns.heatmap(corrMat, annot=True, fmt='.2f')
plt.figure(figsize=(28,28))
plt.show()

```



<Figure size 2016x2016 with 0 Axes>

Explain the Components observed

PC1 — The first component is negatively correlated with eight of the original variables. It decreases with MJD, plate, red-shift, z, i, r, g, u

PC2 — The second component increases with two of the original variables. It increases with field_ID, run_ID

PC3 — The third component increases with only one of the values, increasing on run_ID

Create a new dataframe for our new PCA

In [100]:

```

df_pca_au = pd.DataFrame()
df_pca_au['pca-1'] = X_pca_nine[:,0]
df_pca_au['pca-2'] = X_pca_nine[:,1]
df_pca_au['y_num'] = y_num

```

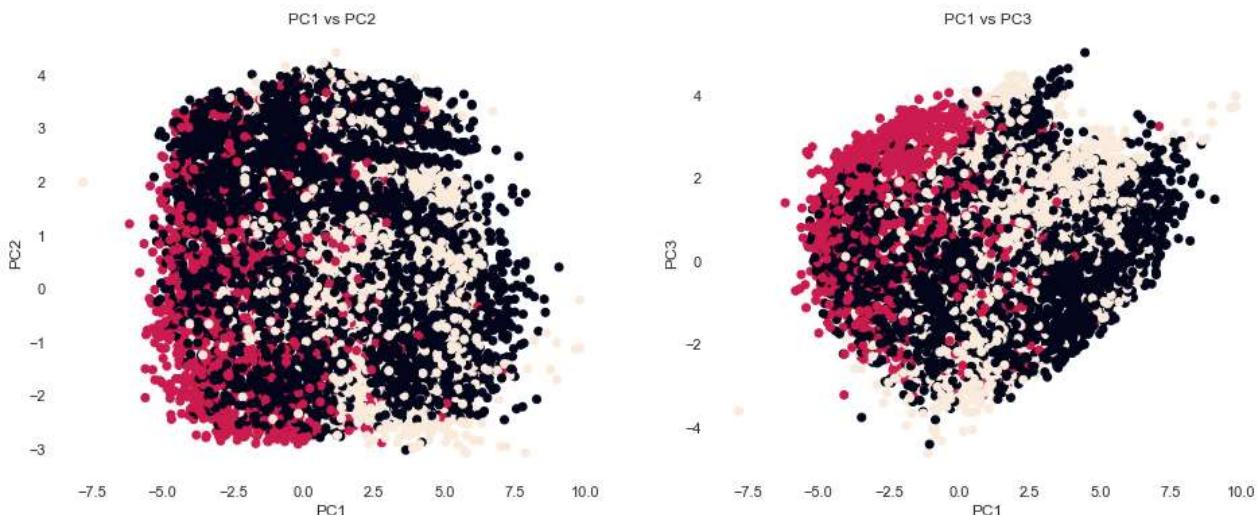
Scatter plot (PC1 vs PC2 or PC1 vs PC3)

In [139...]

```
fig = plt.figure()
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

ax1.scatter(X_pca_nine[:,0],X_pca_nine[:,1], c=y_num)
ax1.set_xlabel("PC1")
ax1.set_ylabel("PC2")
ax1.set_frame_on(False)
ax1.grid(True)
ax1.set_title('PC1 vs PC2')

ax2.scatter(X_pca_nine[:,0],X_pca_nine[:,2], c=y_num)
ax2.set_xlabel("PC1")
ax2.set_ylabel("PC3")
ax2.set_title('PC1 vs PC3')
ax2.grid(True)
ax2.set_frame_on(False)
```



We can see from the above Principal Component Scatter Plots that our classes are not forming a cohesive structure. Let's look further.

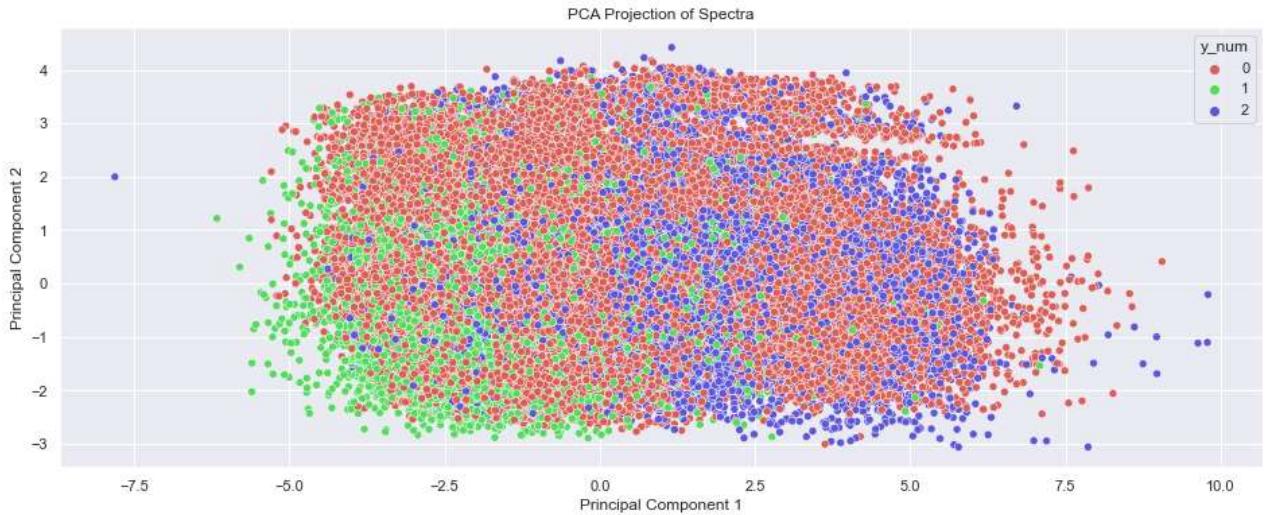
Plot our PCA as a scatter plot with Auxilary Data

In [141...]

```
sns.scatterplot(x='pca-1',
                 y='pca-2',
                 hue='y_num',
                 data=df_pca_au,
                 palette=sns.color_palette('hls',3)).set_title('PCA Projection of Spectr
print(le.classes_)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

['GALAXY' 'QSO' 'STAR']



Interpretation of Principal Component Scatter Plots:

We can see from the above Principal Component Scatter Plots that our classes are not forming a cohesive structure and something may that does not correspond to the underlying classes may be dominating the variance--particularly in PC1 and PC2

PCA BiPlot with Auxilary Data

Let's take a look at what the BiPlot would be with all the auxilary data

```
In [101...]: features = ['alpha', 'delta', 'u', 'g', 'r', 'i', 'z', 'run_id', 'cam-col',
              'field_id', 'red-shift', 'plate', 'MJD', 'fiber_id']
Loadings = pca.components_.T * np.sqrt(pca.explained_variance_)
fig = px.scatter(df_pca_au, x='pca-1', y='pca-2', title='Biplot of PC1 & PC2', opacity=0.5)
for i, feature in enumerate(features):
    fig.add_shape(
        type='line',
        x0=0, y0=0,
        x1=Loadings[i, 0]*4.5,
        y1=Loadings[i, 1]*4.5
    )
    fig.add_annotation(
        x=Loadings[i, 0]*5.5,
        y=Loadings[i, 1]*5.5,
        ax=0, ay=0,
        xanchor="center",
        yanchor="bottom",
        text=feature,
    )
fig.show()
```

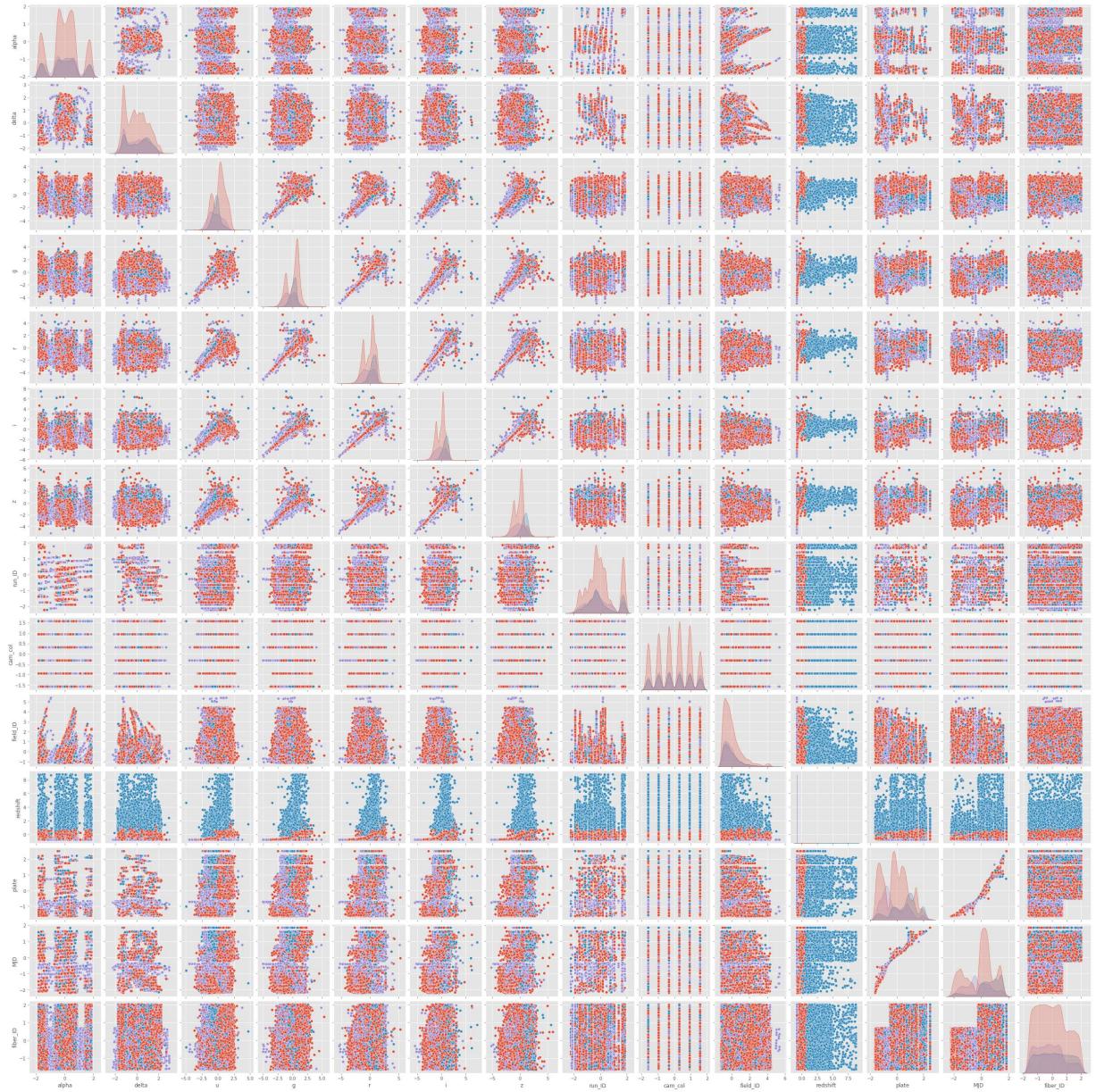
Interpretation of Bi-Plot with Auxilary Data

- *The orientation of our vectors with respect to our first Principal Component (PC1) shows that u, z, r, red_shift, fiber_id, MJD, plate, and i contributes more to PC1.*
- *The orientation of our vectors with respect to our second Principal Component (PC2) shows that run_id, field_id, cam_col, alpha and delta contributes more to PC2.*
- *The length of r, i, z, MJD, and plate, seem to represent that there might be more variability of these variables.*
- *The angles of r, i , z, MJD, plate, u, red_shift and fiber_id, seem to imply that there is a high positive correlation between them.*
- *The angles of alpha and delta appear to imply that there is a high positive correlation between them.*
- *The angles of field_id and g, u, i, r, z, MJD, plate, red_shift, and fiber_id imply there is NO correlation between them.*
- *And the angles between field_id, run_id AND cam_col, alpha and delta imply a strong negative correlation between the variables.*

Investigate Correlations and Potential PCA Variance Inflation

In [965...]

```
sns.pairplot(X_df, hue='cLass')
print()
```



Based on the Biplot and Pairplot, we can clearly see a lot of variables do not provide distinguishing, characteristic information about any of our classes.

We explore this in depth in our t-SNE section (further below), but for now, let's look at the places we see linear combinations that relate to our classes.

In [943...]

```
sns.pairplot(pd.merge(X_df.Loc[:,['u','g','r','i','z','redshift']],  
y,  
        on=X_df.index).drop('key_0', axis=1),  
hue='class')  
print()
```

Out[943...]



We can clearly see much cleaner separation of our classes among the relationships of these variables.

Let's see how this changes the results of our PCA

Without Auxilary Columns

```
In [102...]: 
pca_scaled_1 = PCA(n_components = 0.95, random_state=37)
pca_scaled_1.fit(X_df.Loc[:,['u','g','r','i','z','redshift']])
reduced_data = pca_scaled_1.transform(X_df.Loc[:,['u','g','r','i','z','redshift']])
print('Original Dimensions: ',X.shape)
print('Reduced Dimensions: ',reduced_data.shape)
```

Original Dimensions: (99999, 14)
Reduced Dimensions: (99999, 3)

So what this means is that 95% of variance is observed by 3 dimensions

Now we can select the correct amount of components

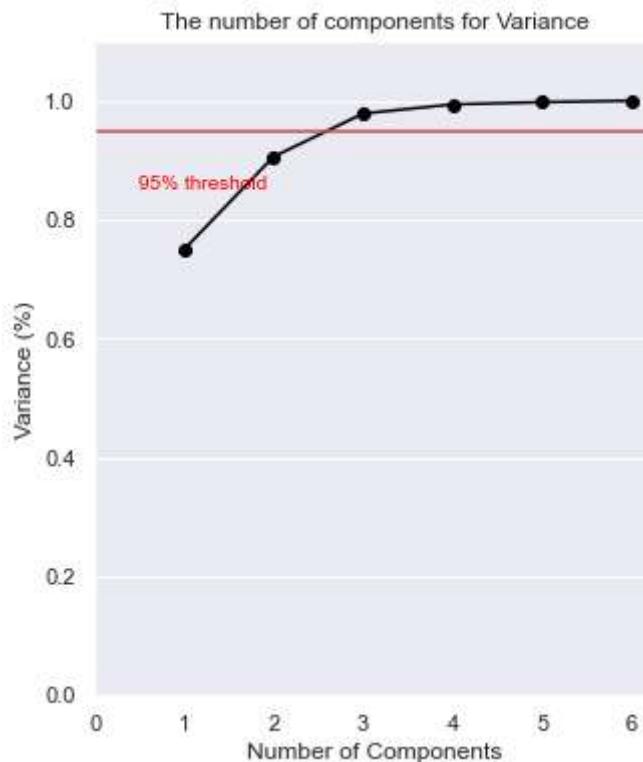
So let's create a plot to visualize this data, basically I want to see the component cutoff point by

creating a plot with our 95% threshold line, and wherever our 'component line' crosses is the amount of components we should choose

Cumulative Variance Plot

In [103...]

```
pca_3_plot = PCA().fit(X_df.Loc[:,['u','g','r','i','z','redshift']])
plt.rcParams["figure.figsize"] = (5,6)
fig, ax = plt.subplots()
plt.ylim(0.0,1.1)
plt.plot(np.arange(1, 7, step=1), np.cumsum(pca_3_plot.explained_variance_ratio_), marker='o')
plt.xlabel('Number of Components')
plt.xticks(np.arange(0, 7, step=1)) #change to 1 based
plt.ylabel('Variance (%)')
plt.title('The number of components for Variance')
plt.axhline(y=0.95, color='r', linestyle='--')
plt.text(0.5, 0.85, '95% threshold', color = 'red', fontsize=10)
ax.grid(axis='x')
plt.show()
```



After looking at our Scree Plot, it seems that actually 95% of variance is explained by 3 dimensions

Create our PCA on our scaled data without Auxilary Columns

In [104...]

```
pca_3 = PCA(n_components=3)
X_pca_three = pca_3.fit_transform(X_df.Loc[:,['u','g','r','i','z','redshift']])
```

Create a Scree Plot without Auxliary Columns

The scree plot will show us how much variance each component has on the data. This way we can see which components we actually need for our final PCA. Whatever components don't capture most of the information then we can ignore them.

In [142...]

```

#print our variance
print('----- VARIANCE -----')
print(pca_3.explained_variance_)

#now print the ratio
print('----- RATIO -----')
print(pca_3.explained_variance_ratio_)

```

```

----- VARIANCE -----
[4.49809045 0.9373612  0.43547947]
----- RATIO -----
[0.74967425 0.15622531 0.07257919]

```

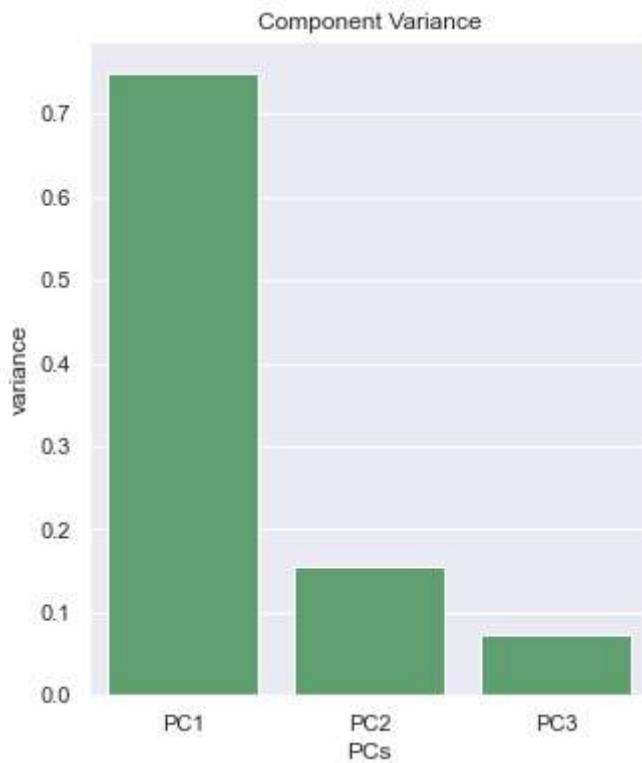
SCREE PLOT

In [142...]

```

scree_plot = pd.DataFrame({'variance':pca_3.explained_variance_ratio_, 'PCs':['PC1','PC2','PC3']})
sns.barplot(x='PCs',y="variance",data=scree_plot, color="g").set_title('Component Variance')

```



Interpretation of Scree Plot:

This shows that it looks like the first 2 components contain the most valuable information of our data.

Take a look at the correlations between the original data and each component

In [105...]

```

#create a new data frame with our PCA
pc_rem_3 = pd.DataFrame(data = X_pca_three, columns = ['pc1','pc2','pc3'])
#drop the components we established above that we do not need
pc_rem_3=pc_rem_3.drop(['pc3'],axis=1)

pc_col_3 = pd.concat([pc_rem_3, X_df], axis=1)
pc_col_3
corrMat = pd.DataFrame.cov(pc_col_3)
sns.set(rc={'figure.figsize':(16,6)})

```

```

sns.heatmap(corrMat, annot=True, fmt='.2f')
plt.figure(figsize=(28,28))
plt.show()

```



<Figure size 2016x2016 with 0 Axes>

Explain the Components observed without Auxilary columns

PC1 — The first component is negatively correlated with eight of the original variables. It decreases with MJD , plate , red-shift , z , i , r , g , u

PC2 — The second component increases with one of the original varaibles. It increases with red-shift

Create a new dataframe for our new PCA

In [106...]

```

df_pca_wout_au = pd.DataFrame()
df_pca_wout_au['pca-1'] = X_pca_three[:,0]
df_pca_wout_au['pca-2'] = X_pca_three[:,1]
df_pca_wout_au['pca-3'] = X_pca_three[:,2]
df_pca_wout_au['y_num'] = y_num

```

Scatter plot (PC1 vs PC2 or PC1 vs PC3)

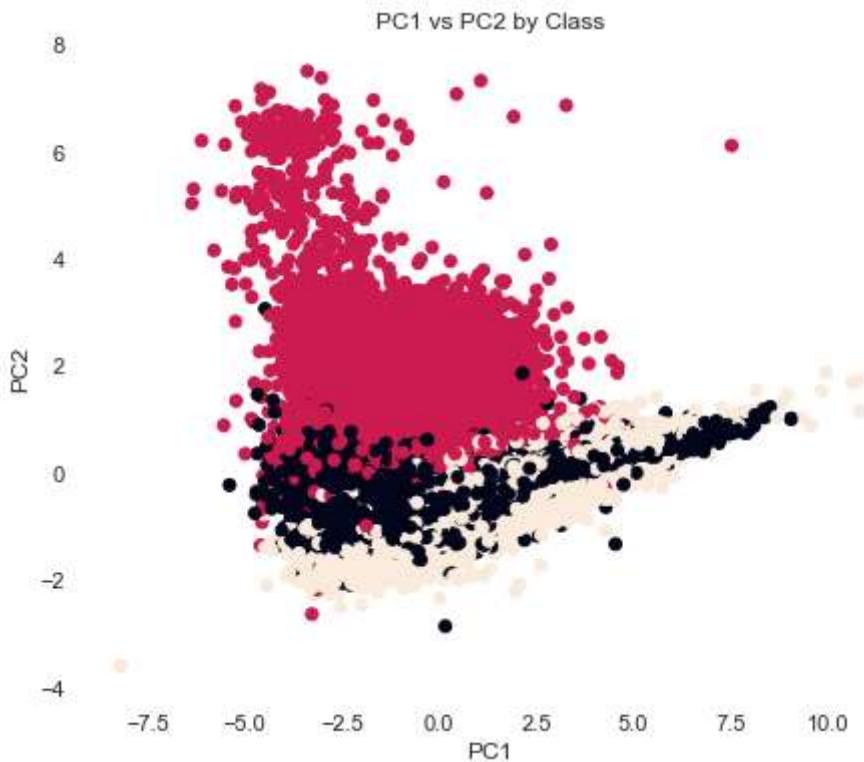
In [107...]

```

fig = plt.figure()
ax1 = fig.add_subplot(121)

ax1.scatter(X_pca_three[:,0],X_pca_three[:,1], c=y_num)
ax1.set_xlabel("PC1")
ax1.set_ylabel("PC2")
ax1.set_frame_on(False)
ax1.grid(True)
ax1.set_title('PC1 vs PC2 by Class')
print()

```



We can see from the above Principal Component Scatter Plots that our classes are now forming a cohesive structure. Let's look further.

Plot our PCA as a scatter plot without Auxilary Columns

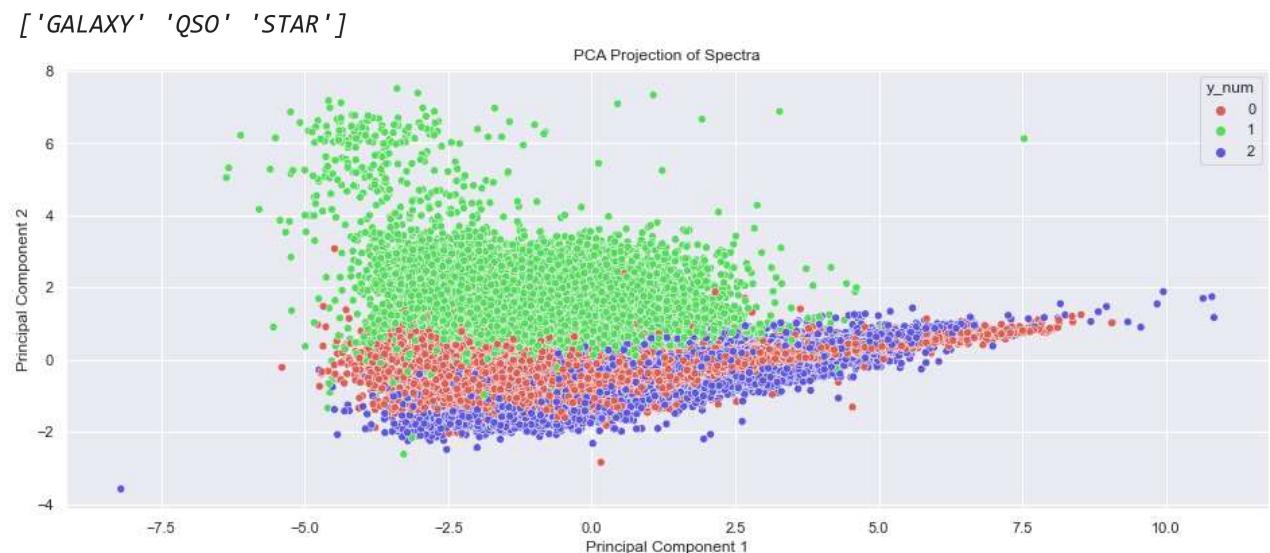
In [143...]

```

sns.scatterplot(x='pca-1',
                 y='pca-2',
                 hue='y_num',
                 data=df_pca_wout_au,
                 palette=sns.color_palette('hls',3)).set_title('PCA Projection of Spectra')
print(le.classes_)

plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()

```



Interpretation of Principal Component Scatter Plots without Auxilary data:

We can see from the above Principal Component Scatter Plots that our classes are now forming a cohesive structure and that the additional auxilary data was potentially dominating the variance.

PCA BiPlot without Auxilary Data

Now let's take a look at our BiPlot without the additional Auxilary Data

In [108...]

```
features = ['u', 'g', 'r', 'i', 'z', 'redshift']
Loadings = pca_3.components_.T * np.sqrt(pca_3.explained_variance_)
fig = px.scatter(df_pca_wout_au, x='pca-1', y='pca-2', title='Biplot of PC1 & PC2', opacity=0.5)
for i, feature in enumerate(features):
    fig.add_shape(
        type='Line',
        x0=0, y0=0,
        x1=Loadings[i, 0]*4.5,
        y1=Loadings[i, 1]*4.5
    )
    fig.add_annotation(
        x=Loadings[i, 0]*5.5,
        y=Loadings[i, 1]*5.5,
        ax=0, ay=0,
        xanchor="center",
        yanchor="bottom",
        text=feature,
    )
fig.show()
```

Interpretation of biplot without Auxiliary Data

- The orientation of our vectors with respect to our first Principal Component (PC1) shows that z , r , and i contributes more to PC1. The magnitude of g making it a significant contributor as well.
 - The orientation of our vectors with respect to our second Principal Component (PC2) shows that redshift, g , and u contribute more to PC2.
 - The length/magnitude of r , i , and z seem to represent that these variables contribute a significant amount of variance to Principle Component 1.
 - The angles of r , i , and z seem to imply that there is a high positive correlation between them.
 - The angles of g and u to imply that there is a high positive correlation between them.
-

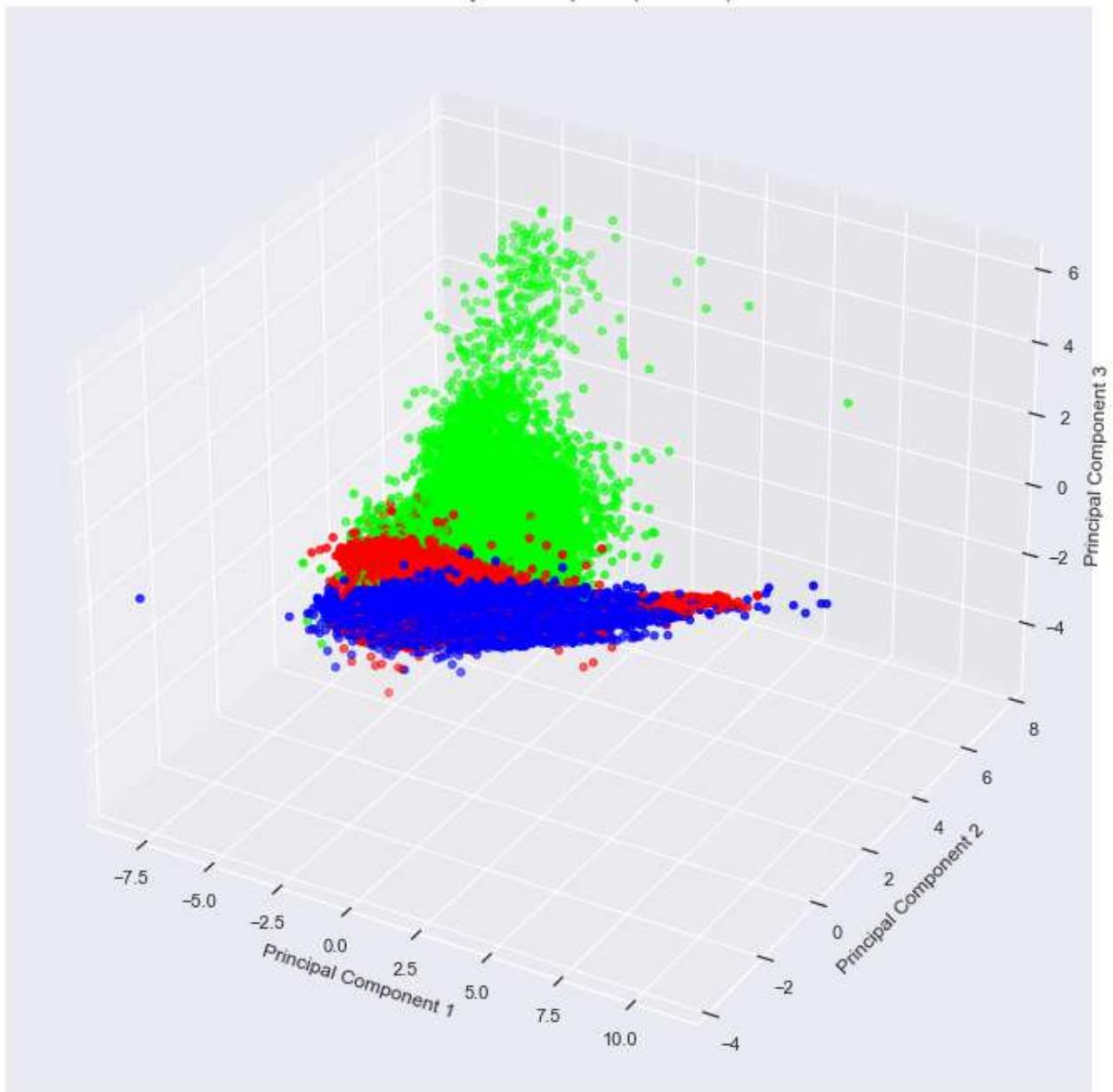
3D PCA Scatter Plot

In [145...]

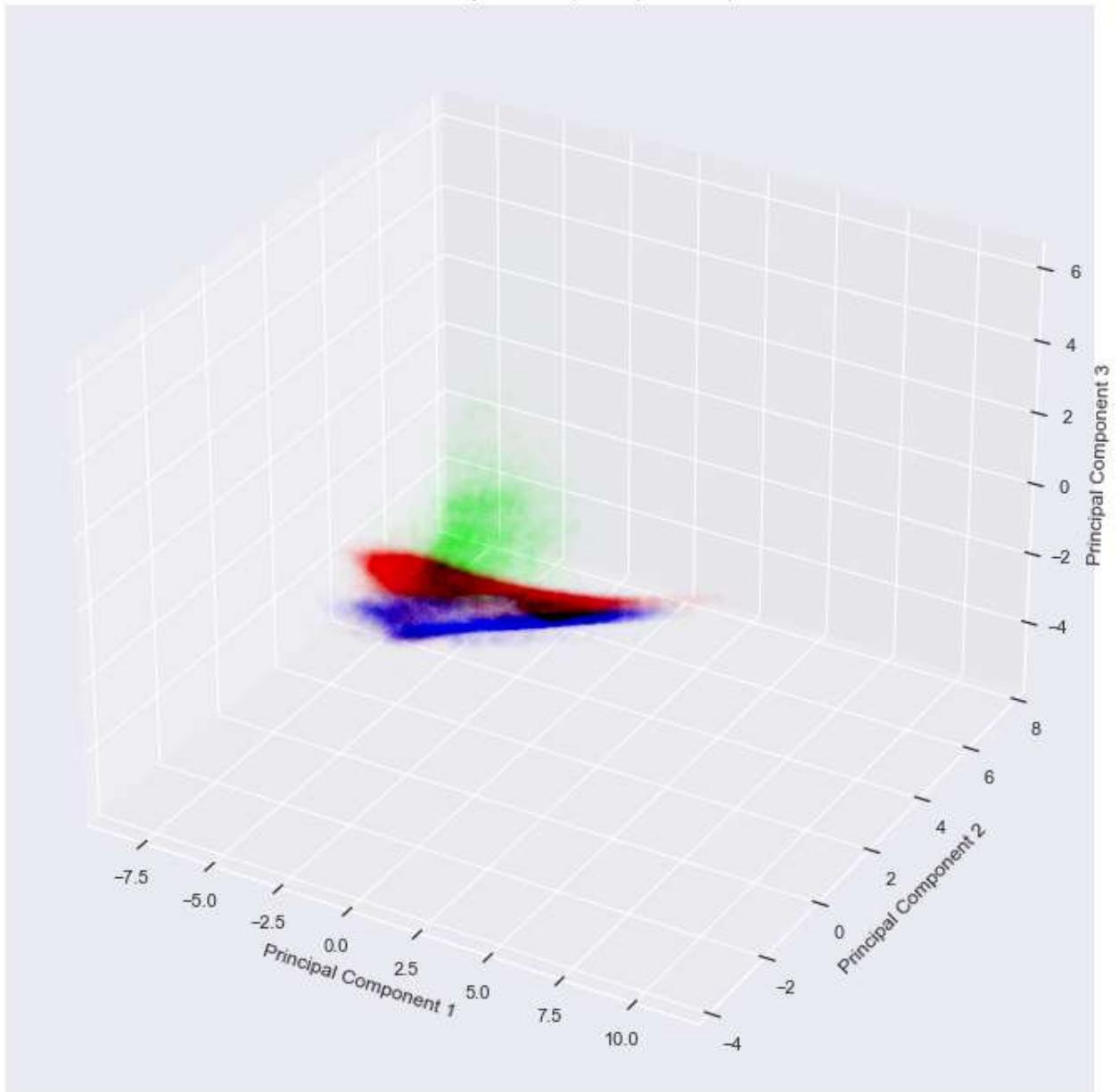
```
fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df_pca_wout_au['pca-1'],
           df_pca_wout_au['pca-2'],
           zs=df_pca_wout_au['pca-3'],
           c=y_num,
           cmap=cmap_bold)
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('3D PCA Projection of Spectra (w/out Aux)')
plt.show()

fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(df_pca_wout_au['pca-1'],
           df_pca_wout_au['pca-2'],
           zs=df_pca_wout_au['pca-3'],
           c=y_num,
           cmap=cmap_bold, alpha=0.002)
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('3D PCA Projection of Spectra (w/out Aux)')
plt.show()
```

3D PCA Projection of Spectra (w/out Aux)



3D PCA Projection of Spectra (w/out Aux)



We can clearly see that our classes are forming cohesive structures among the first three dimensions of our PCA with variables `u`, `g`, `r`, `i`, `z`, and `redshift`--particularly in comparison to our PCA that also included `alpha`, `delta`, `run_id`, `cam_col`, `field_id`, `pPlate`, `MJD`, `fiber_id`.

t-SNE

Calculate 2D t-SNE (with Auxiliary Columns)

In [425...]

```
tsne_2_aux = TSNE(n_components=2, verbose=1, random_state=37)
tsne_2d_aux = tsne_2_aux.fit_transform(X)
```



```
[t-SNE] Computed conditional probabilities for sample 58000 / 99999
[t-SNE] Computed conditional probabilities for sample 59000 / 99999
[t-SNE] Computed conditional probabilities for sample 60000 / 99999
[t-SNE] Computed conditional probabilities for sample 61000 / 99999
[t-SNE] Computed conditional probabilities for sample 62000 / 99999
[t-SNE] Computed conditional probabilities for sample 63000 / 99999
[t-SNE] Computed conditional probabilities for sample 64000 / 99999
[t-SNE] Computed conditional probabilities for sample 65000 / 99999
[t-SNE] Computed conditional probabilities for sample 66000 / 99999
[t-SNE] Computed conditional probabilities for sample 67000 / 99999
[t-SNE] Computed conditional probabilities for sample 68000 / 99999
[t-SNE] Computed conditional probabilities for sample 69000 / 99999
[t-SNE] Computed conditional probabilities for sample 70000 / 99999
[t-SNE] Computed conditional probabilities for sample 71000 / 99999
[t-SNE] Computed conditional probabilities for sample 72000 / 99999
[t-SNE] Computed conditional probabilities for sample 73000 / 99999
[t-SNE] Computed conditional probabilities for sample 74000 / 99999
[t-SNE] Computed conditional probabilities for sample 75000 / 99999
[t-SNE] Computed conditional probabilities for sample 76000 / 99999
[t-SNE] Computed conditional probabilities for sample 77000 / 99999
[t-SNE] Computed conditional probabilities for sample 78000 / 99999
[t-SNE] Computed conditional probabilities for sample 79000 / 99999
[t-SNE] Computed conditional probabilities for sample 80000 / 99999
[t-SNE] Computed conditional probabilities for sample 81000 / 99999
[t-SNE] Computed conditional probabilities for sample 82000 / 99999
[t-SNE] Computed conditional probabilities for sample 83000 / 99999
[t-SNE] Computed conditional probabilities for sample 84000 / 99999
[t-SNE] Computed conditional probabilities for sample 85000 / 99999
[t-SNE] Computed conditional probabilities for sample 86000 / 99999
[t-SNE] Computed conditional probabilities for sample 87000 / 99999
[t-SNE] Computed conditional probabilities for sample 88000 / 99999
[t-SNE] Computed conditional probabilities for sample 89000 / 99999
[t-SNE] Computed conditional probabilities for sample 90000 / 99999
[t-SNE] Computed conditional probabilities for sample 91000 / 99999
[t-SNE] Computed conditional probabilities for sample 92000 / 99999
[t-SNE] Computed conditional probabilities for sample 93000 / 99999
[t-SNE] Computed conditional probabilities for sample 94000 / 99999
[t-SNE] Computed conditional probabilities for sample 95000 / 99999
[t-SNE] Computed conditional probabilities for sample 96000 / 99999
[t-SNE] Computed conditional probabilities for sample 97000 / 99999
[t-SNE] Computed conditional probabilities for sample 98000 / 99999
[t-SNE] Computed conditional probabilities for sample 99000 / 99999
[t-SNE] Computed conditional probabilities for sample 99999 / 99999
[t-SNE] Mean sigma: 0.538113
[t-SNE] KL divergence after 250 iterations with early exaggeration: 103.694984
[t-SNE] KL divergence after 1000 iterations: 3.134697
```

Create 2D t-SNE Data Frame (with Auxiliary Columns)

In [816...]

```
df_tsne_2d_aux = pd.DataFrame()
df_tsne_2d_aux['tsne-2d-1'] = tsne_2d_aux[:,0]
df_tsne_2d_aux['tsne-2d-2'] = tsne_2d_aux[:,1]
df_tsne_2d_aux['y_num'] = y_num
```

Scatter Plot 2D t-SNE by Class (with Auxiliary Columns)

In [817...]

```
sns.scatterplot(x='tsne-2d-1',
```

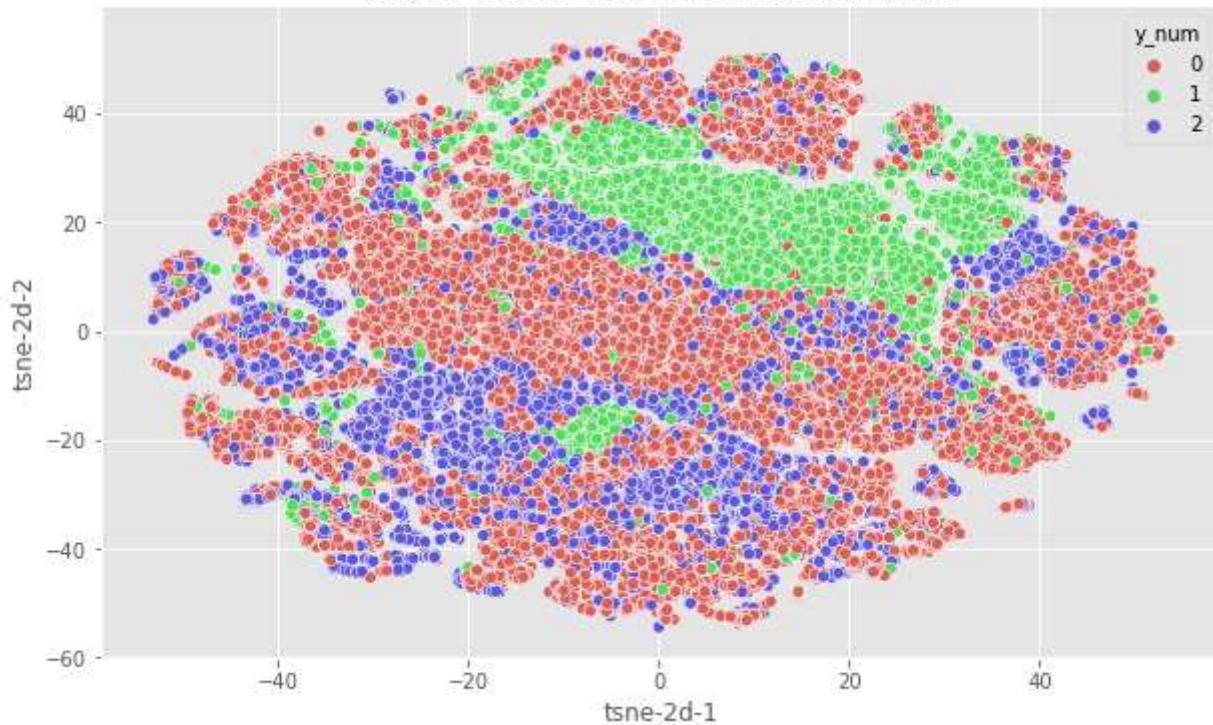
```

y='tsne-2d-2',
hue='y_num',
data=df_tsne_2d_aux,
palette=sns.color_palette('hls',3)).set_title('Scatter Plot of 2D t-SNE')
print(le.classes_)

```

['GALAXY' 'QSO' 'STAR']

Scatter Plot of 2D t-SNE by Class (w/Aux)



Correlation Matrix With/Without Auxiliary Columns

In [147]:

```

fig,axs = plt.subplots(1,2)
X_df['y_num'] = y_num

ax = sns.heatmap(X_df.corr(), center=0, vmin=-1, vmax=1, square=True, cmap=rb_diverging
ax.set_xticklabels(ax.get_xticklabels(), horizontalalignment='right', rotation=45)
ax.set_title('Correlation Matrix (w/Aux)')

ax = sns.heatmap(X_df.drop(['alpha',
                           'delta',
                           'cam_col',
                           'plate',
                           'field_ID',
                           'run_ID',
                           'MJD',
                           'fiber_ID'], axis=1).corr(),
                  center=0,
                  vmin=-1,
                  vmax=1,
                  square=True,
                  cmap=rb_diverging,
                  ax=axs[1])
ax.set_xticklabels(ax.get_xticklabels(), horizontalalignment='right', rotation=45)
ax.set_title('Correlation Matrix (w/o Aux)')

```

```
X_df = X_df.drop('y_num', axis=1)
print()
```

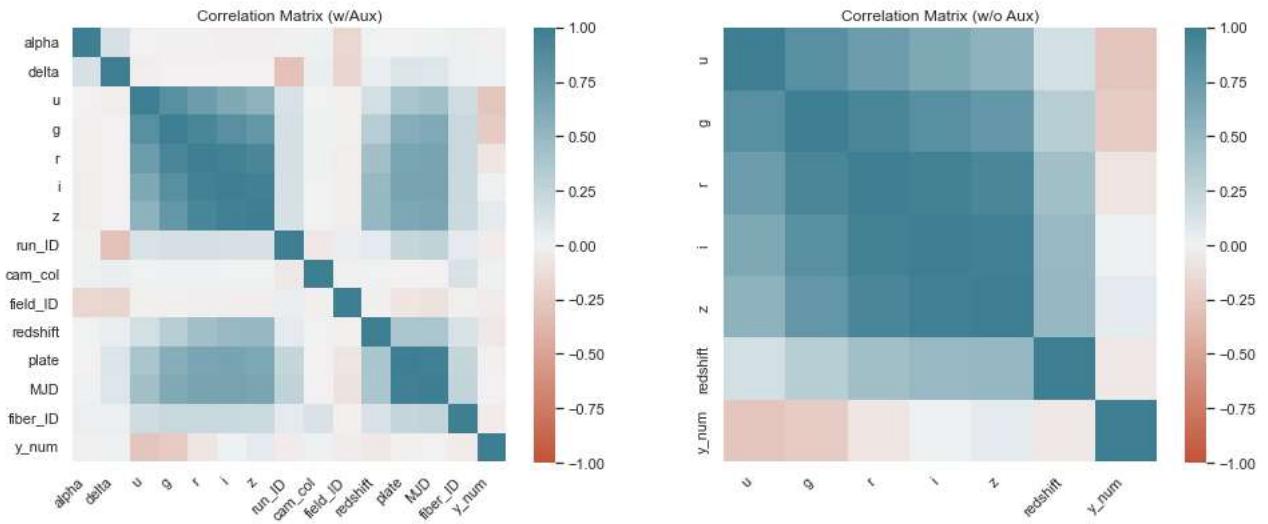


plate & MJD are moderately correlated with *u, g, r, i, z, and redshift*. The same can be said for *fiber_ID*, to a lesser degree. In addition, *alpha* and *delta* represent the location on the celestial sphere (sky), which may have a relationship with the classes of astronomical objects present, but is not an observable characteristic of the object itself. Similarly, *run_id* is the identifier of the run, which may have implications on classes observed in that run, but is not directly an attribute of the object--and likewise for *cam_col / field_ID*.

As we saw with the single outlier that shaped our correlation matrix, added variance that does not represent the underlying silhouette we are trying to capture can lead us astray--particularly when contained in a variable that has a significant correlation with a variable that does represent what we are trying to capture.

Let's examine what happens to our t-SNE dimensions when we remove these variables.

Calculate 2D t-SNE (without Auxiliary Columns)

In [429...]

```
tsne_2 = TSNE(n_components=2, verbose=1, random_state=37)
tsne_2d = tsne_2.fit_transform(
    X_df.drop(['alpha',
               'delta',
               'cam_col',
               'plate',
               'field_ID',
               'run_ID',
               'MJD',
               'fiber_ID'], axis=1).to_numpy())
```

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 99999 samples in 0.108s...
[t-SNE] Computed neighbors for 99999 samples in 4.149s...
[t-SNE] Computed conditional probabilities for sample 1000 / 99999


```
[t-SNE] Computed conditional probabilities for sample 62000 / 99999
[t-SNE] Computed conditional probabilities for sample 63000 / 99999
[t-SNE] Computed conditional probabilities for sample 64000 / 99999
[t-SNE] Computed conditional probabilities for sample 65000 / 99999
[t-SNE] Computed conditional probabilities for sample 66000 / 99999
[t-SNE] Computed conditional probabilities for sample 67000 / 99999
[t-SNE] Computed conditional probabilities for sample 68000 / 99999
[t-SNE] Computed conditional probabilities for sample 69000 / 99999
[t-SNE] Computed conditional probabilities for sample 70000 / 99999
[t-SNE] Computed conditional probabilities for sample 71000 / 99999
[t-SNE] Computed conditional probabilities for sample 72000 / 99999
[t-SNE] Computed conditional probabilities for sample 73000 / 99999
[t-SNE] Computed conditional probabilities for sample 74000 / 99999
[t-SNE] Computed conditional probabilities for sample 75000 / 99999
[t-SNE] Computed conditional probabilities for sample 76000 / 99999
[t-SNE] Computed conditional probabilities for sample 77000 / 99999
[t-SNE] Computed conditional probabilities for sample 78000 / 99999
[t-SNE] Computed conditional probabilities for sample 79000 / 99999
[t-SNE] Computed conditional probabilities for sample 80000 / 99999
[t-SNE] Computed conditional probabilities for sample 81000 / 99999
[t-SNE] Computed conditional probabilities for sample 82000 / 99999
[t-SNE] Computed conditional probabilities for sample 83000 / 99999
[t-SNE] Computed conditional probabilities for sample 84000 / 99999
[t-SNE] Computed conditional probabilities for sample 85000 / 99999
[t-SNE] Computed conditional probabilities for sample 86000 / 99999
[t-SNE] Computed conditional probabilities for sample 87000 / 99999
[t-SNE] Computed conditional probabilities for sample 88000 / 99999
[t-SNE] Computed conditional probabilities for sample 89000 / 99999
[t-SNE] Computed conditional probabilities for sample 90000 / 99999
[t-SNE] Computed conditional probabilities for sample 91000 / 99999
[t-SNE] Computed conditional probabilities for sample 92000 / 99999
[t-SNE] Computed conditional probabilities for sample 93000 / 99999
[t-SNE] Computed conditional probabilities for sample 94000 / 99999
[t-SNE] Computed conditional probabilities for sample 95000 / 99999
[t-SNE] Computed conditional probabilities for sample 96000 / 99999
[t-SNE] Computed conditional probabilities for sample 97000 / 99999
[t-SNE] Computed conditional probabilities for sample 98000 / 99999
[t-SNE] Computed conditional probabilities for sample 99000 / 99999
[t-SNE] Computed conditional probabilities for sample 99999 / 99999
[t-SNE] Mean sigma: 0.060146
[t-SNE] KL divergence after 250 iterations with early exaggeration: 97.411530
[t-SNE] KL divergence after 1000 iterations: 2.827295
```

Create 2D t-SNE Data Frame (without Auxiliary Columns)

In [81]...

```
df_tsne_2d = pd.DataFrame()
df_tsne_2d['tsne-2d-1'] = tsne_2d[:,0]
df_tsne_2d['tsne-2d-2'] = tsne_2d[:,1]
df_tsne_2d['y_num'] = y_num
```

Scatter Plot 2D t-SNE by Class (with/without Auxiliary Columns)

In [820]...

```
fig,axs = plt.subplots(1,2)

sns.scatterplot(x='tsne-2d-1',
                 y='tsne-2d-2',
                 hue='y_num',
```

```

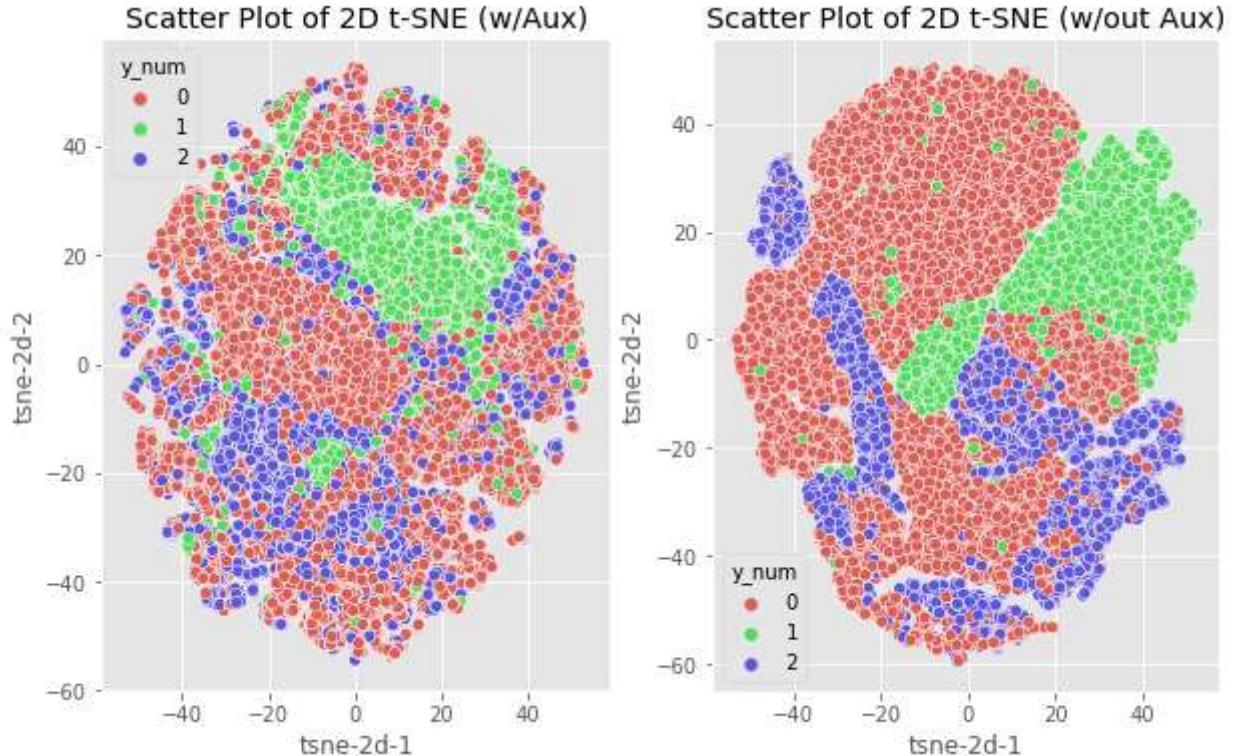
    data=df_tsne_2d_aux,
    palette=sns.color_palette('hls',3),
    ax= axs[0]).set_title('Scatter Plot of 2D t-SNE (w/Aux)')

sns.scatterplot(x='tsne-2d-1',
                 y='tsne-2d-2',
                 hue='y_num',
                 data=df_tsne_2d,
                 palette=sns.color_palette('hls',3),
                 ax= axs[1]).set_title('Scatter Plot of 2D t-SNE (w/out Aux)')

print(le.classes_)

```

['GALAXY' 'QSO' 'STAR']



Clearly the difference is a much more coherent Dimensionality Reduction, given the insight from coloring based on class

Let's validate that the variables we are considering removing/keeping.

In [147]:

```

X_df['class_num'] = y_num
X_df['class_num'] = X_df['class_num'].astype(int)

def KdePlot(column, log=False):
    for num_class in range(3):
        dat=X_df[X_df['class_num']==num_class][column]
        if log:
            dat=np.log(dat)
        sns.kdeplot(data=dat,
                     label = le.inverse_transform([num_class]))
    sns.kdeplot(data=X_df[column],
                 label = ['ALL'])
    plt.legend();

```

Variable vs Class Density Plots

Let's validate that the variables we are considering removing do not appear to categorize or distinguish any of our classes

Conversely, let's validate that the variables we are considering keeping do appear to categorize or distinguish one or more classes

In [147...]

```
y.value_counts()
```

Out[147...]

```
GALAXY    59445  
STAR      21593  
QSO       18961  
Name: class, dtype: int64
```

One thing to keep in mind, before we proceed, is that we have **Class Imbalance** among our classes (`GALAXY`, `STAR`, `QSO`).

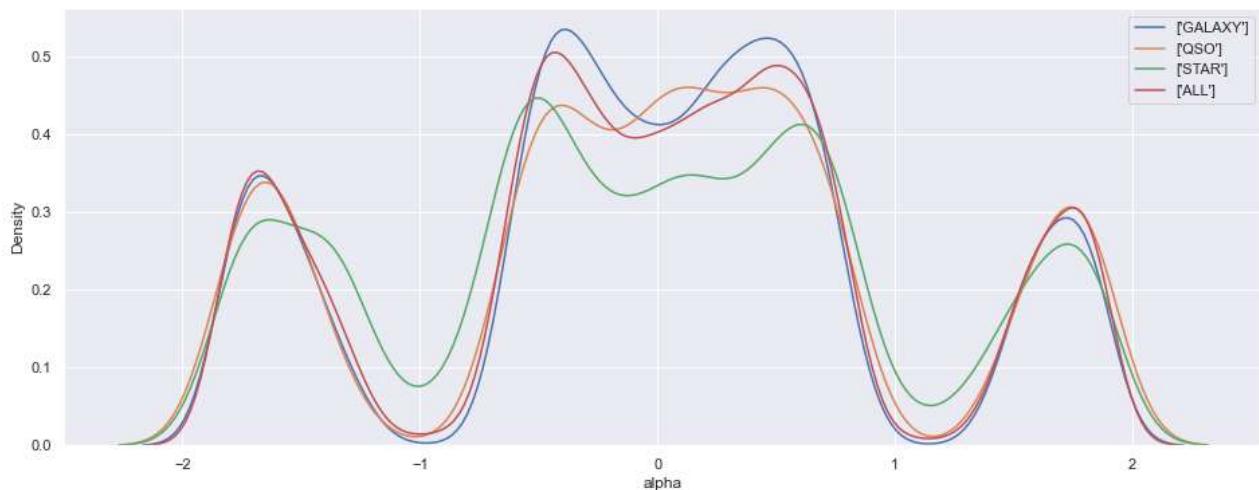
This means that we should interpret the below density plots with that in mind.

In our case, `GALAXY` has a significantly higher sample size and will therefore dominate the distribution of the `ALL` density plot.

Right Ascension (similar to longitude)

In [147...]

```
KdePlot('alpha')
```

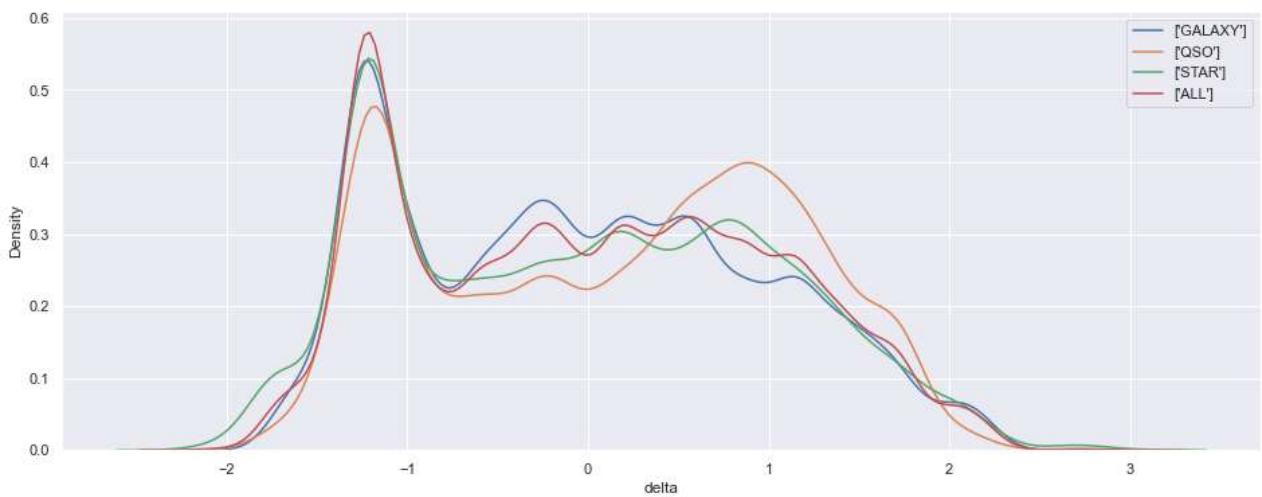


Though not highly significant, `STAR` distinctly stands out relative to the other classes

Declination (similar to latitude)

In [148...

KdePlot('delta')

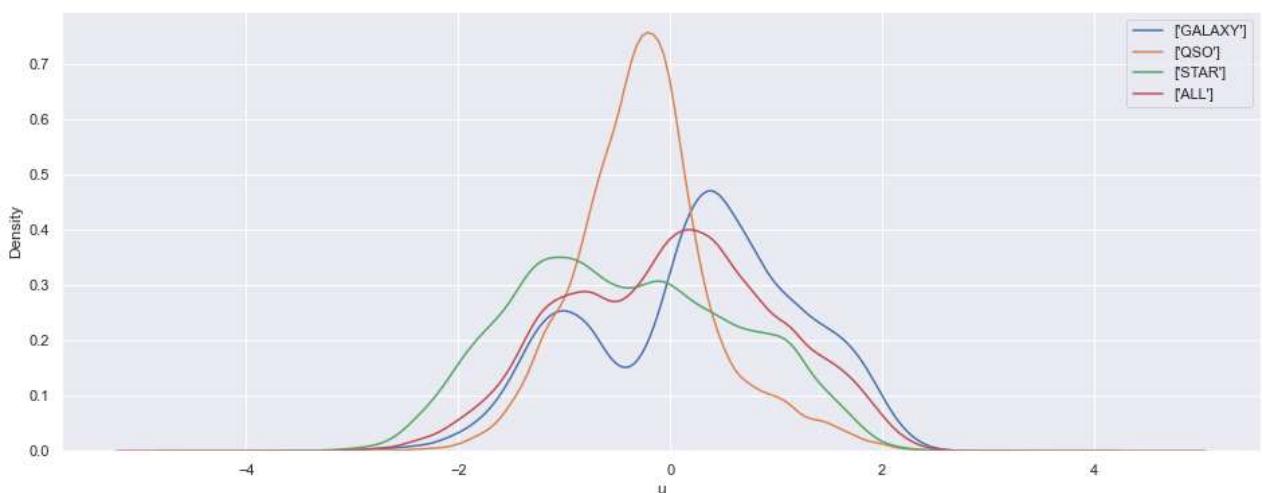


QSO has a distinct profile on the right half of the density graph

Ultraviolet Photometric Filter

In [148...

KdePlot('u')

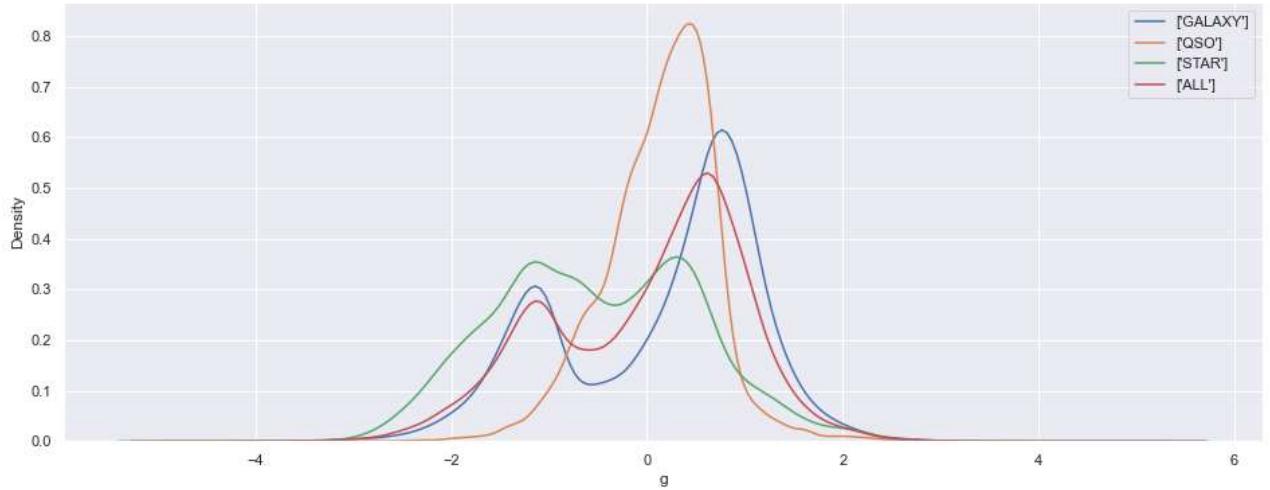


QSO has a distinct profile and GALAXY / STAR --though not significant--may be somewhat distinguishable as well

Green Photometric Filter

In [148...

KdePlot('g')

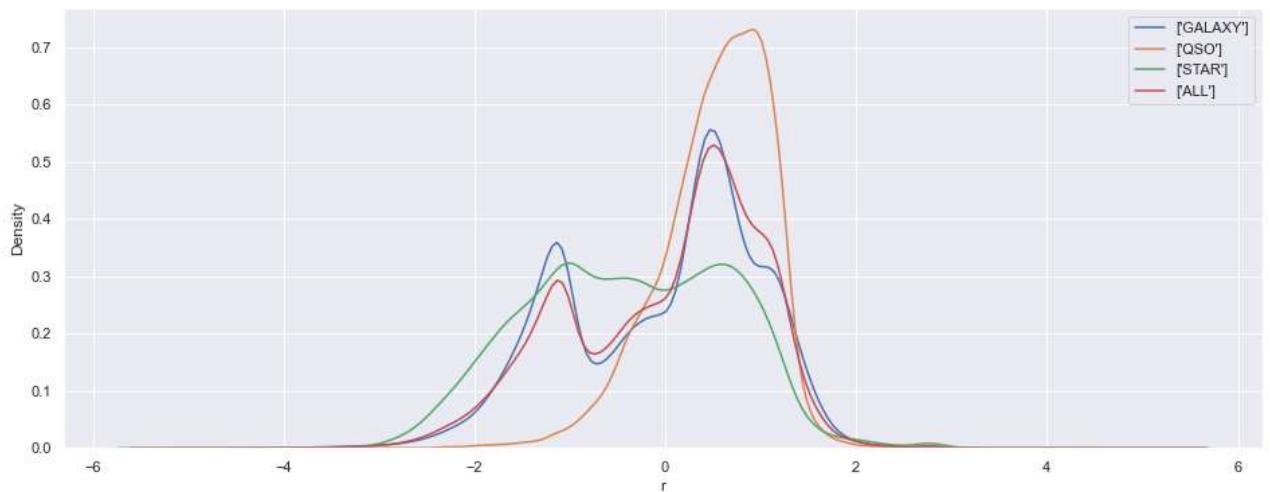


QSO has a distinct profile and GALAXY / STAR --though not significant--may be somewhat distinguishable as well

Red Photometric Filter

In [148...

`KdePlot('r')`

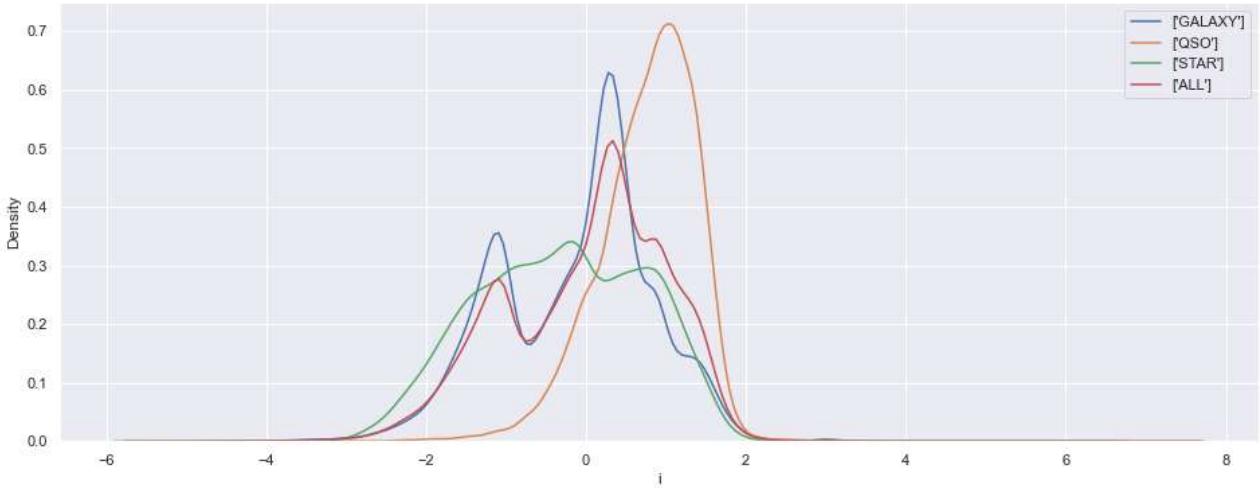


QSO again has a distinct profile and GALAXY / STAR are more distinct than in the last two plots for g and u

Near Infrared Photometric Filter

In [148...

`KdePlot('i')`

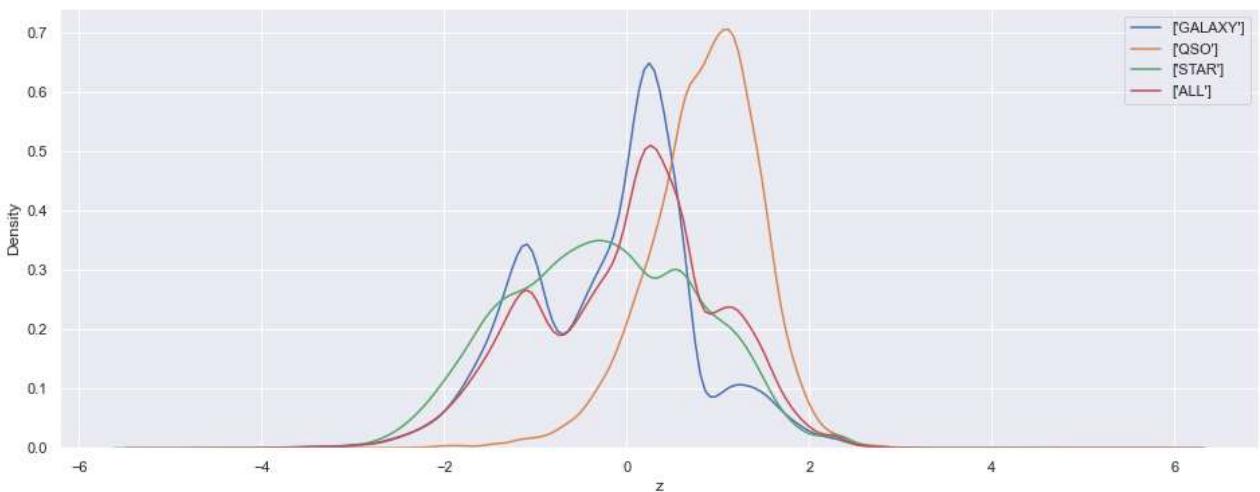


QSO continues to have the high unimodal peak; however, *STAR* is close to unimodal--in contrast to the last two plots for *g* and *r*

Infrared Photometric Filter

In [148...]

```
KdePlot('z')
```

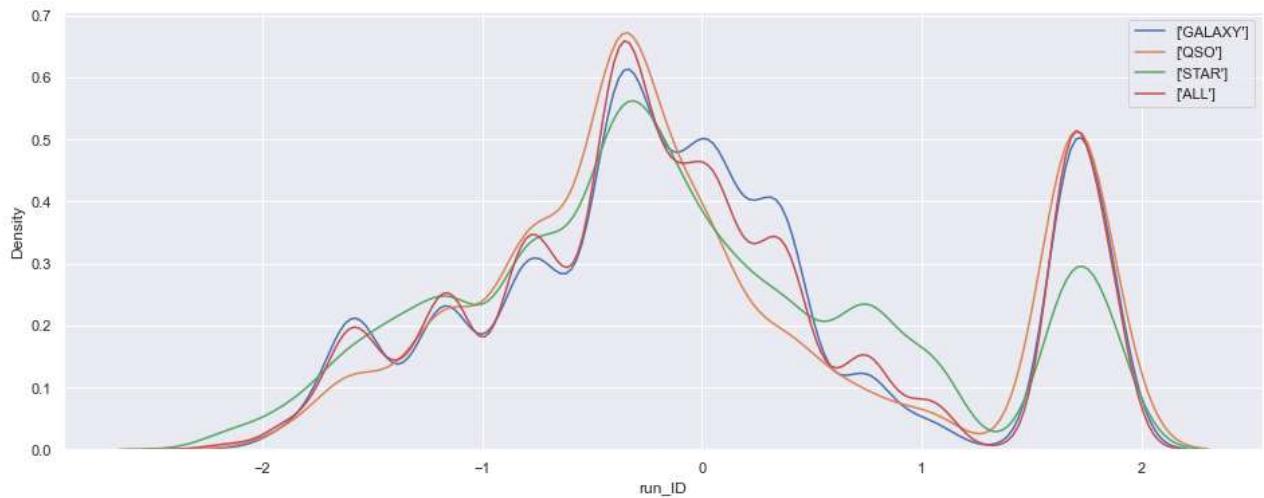


QSO continues to have the high unimodal peak and *STAR* is closer to unimodal than in the higher photometric frequencies

Run Identifier for the Specific Scan

In [148...]

```
KdePlot('run_ID')
```

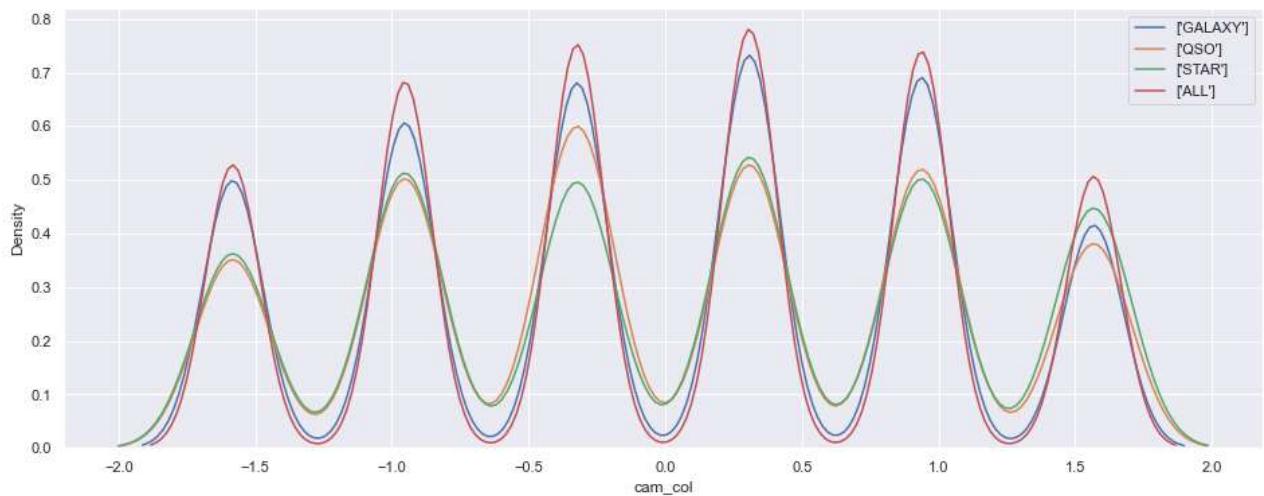


There is no significant distinction here and this variable should be removed

Camera Column (identifies the scanline within run)

In [148...]

```
KdePlot('cam_col')
```

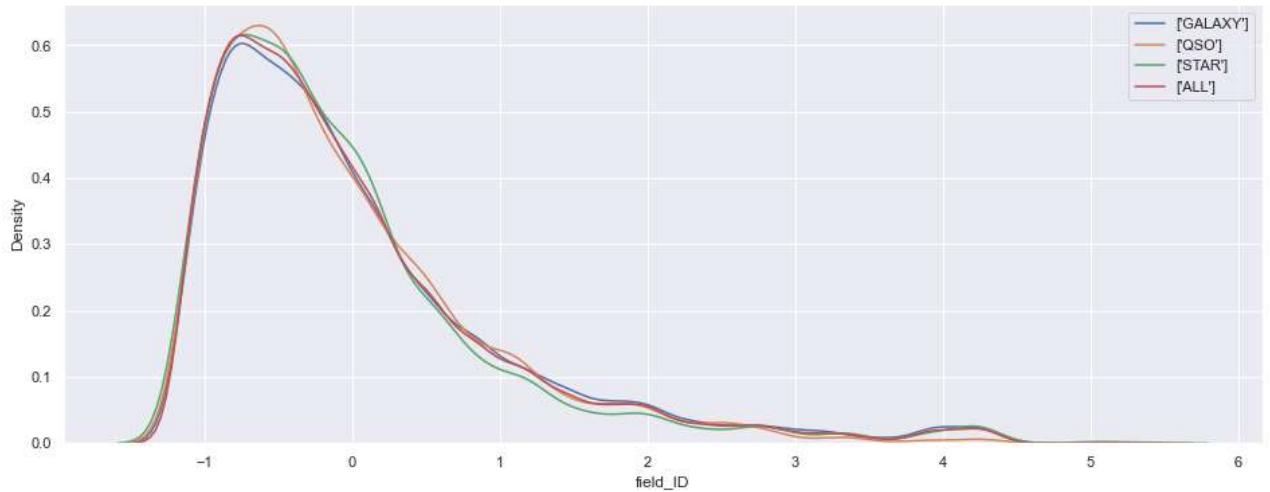


There is no significant distinction here and this variable should be removed

Field ID (identifier specifying each field)

In [148...]

```
KdePlot('field_ID')
```

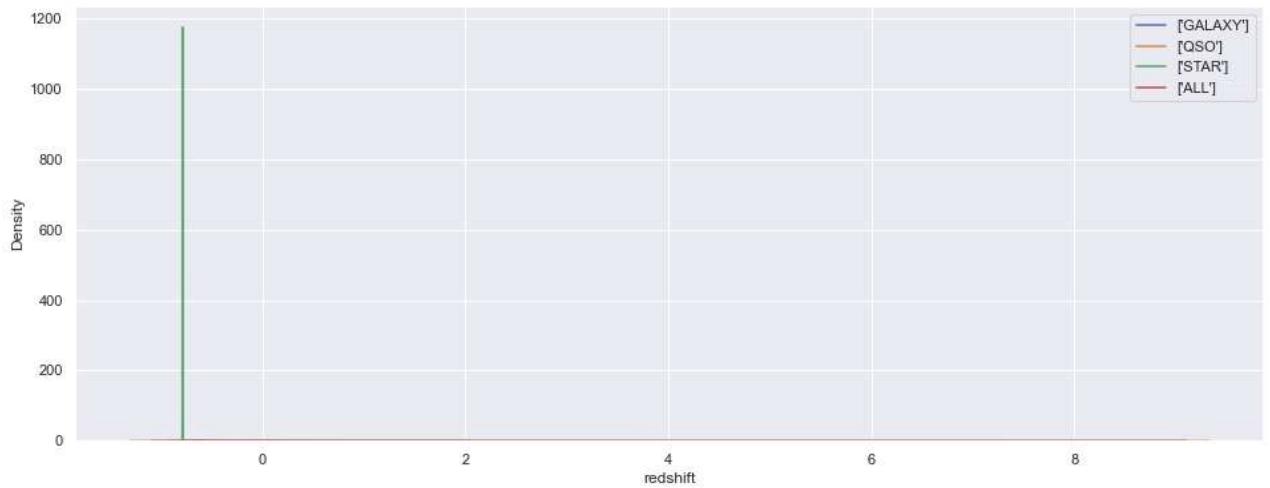


There is no significant distinction here and this variable should be removed

Redshift (based on the increase in wavelength)

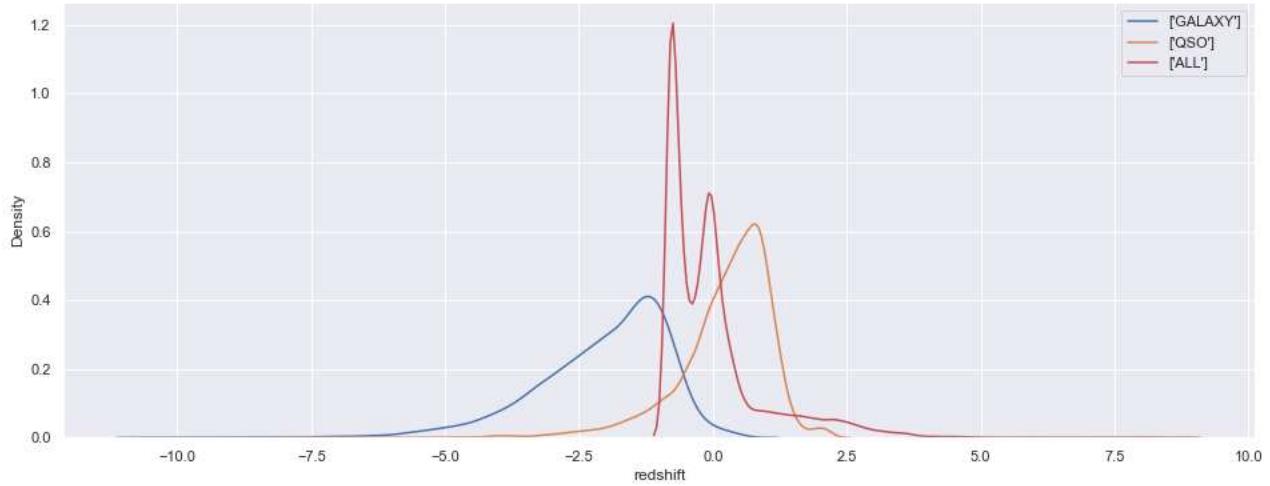
In [148...]

```
KdePlot('redshift')
```



In [149...]

```
KdePlot('redshift', Log=True)
```

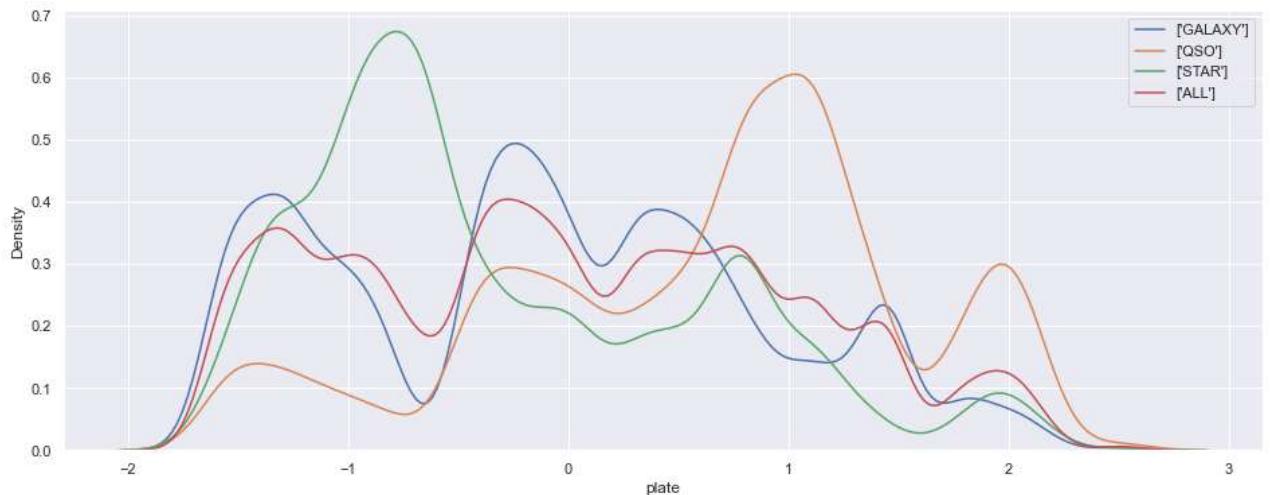


We can see that a warning was thrown and `STAR` does not appear on the log-transformed graph. This may be in part because we are dealing with the data in its already scaled form; however, **we can clearly see that `redshift` distinctly characterized all three classes.**

Plate (identifier of plate in SDSS)

In [149]:

```
KdePlot('plate')
```

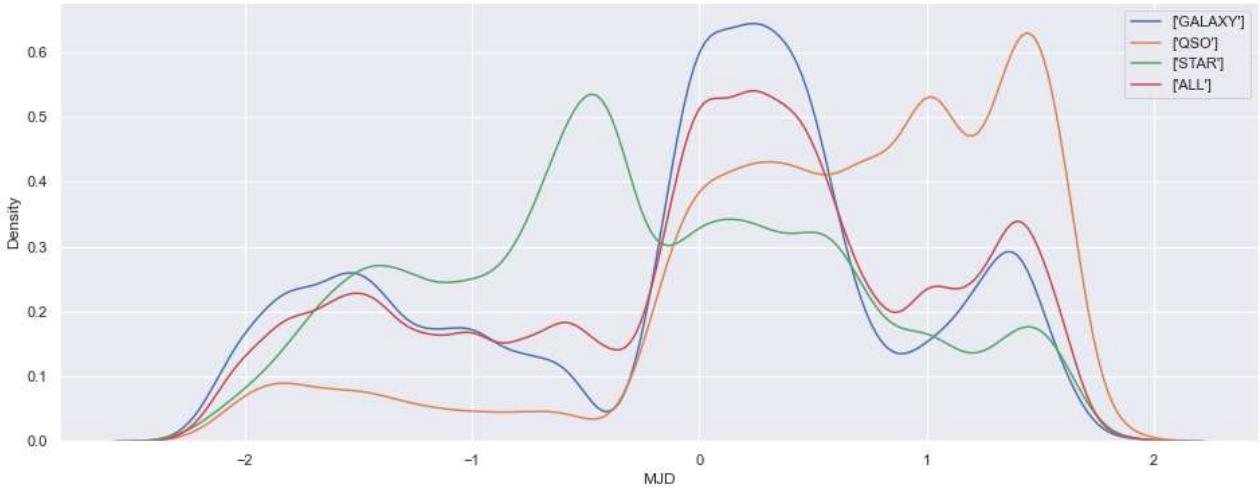


We can see that `QSO` has a distinct profile--particularly on the right of the graph--and `STAR / GALAXY` on the left half of the graph.

MJD (Modified Julian Date of observation)

In [149]:

```
KdePlot('MJD')
```

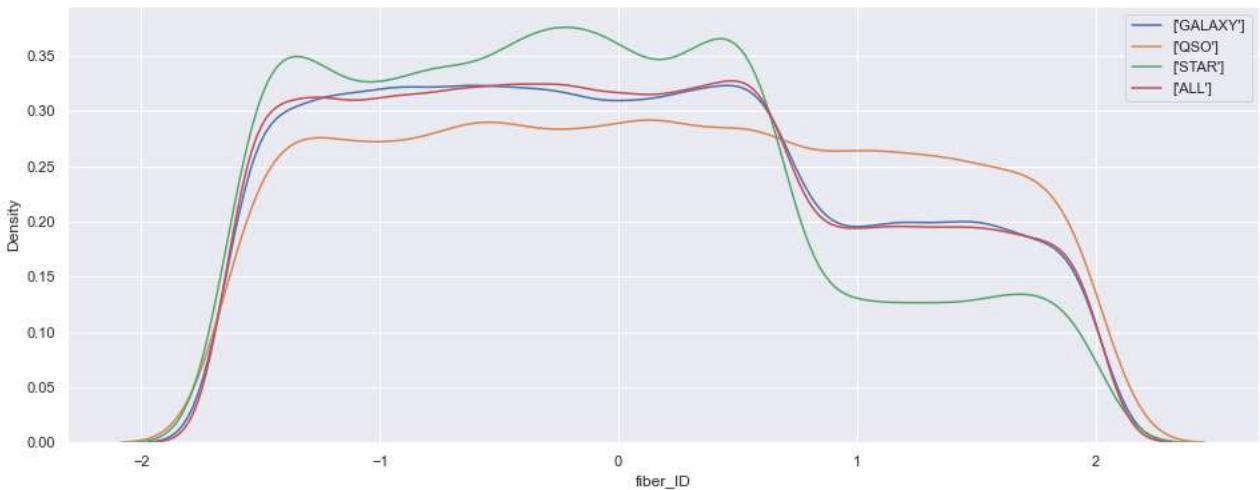


The distribution for MJD distinctly characterizes each of the classes.

- Fiber ID (identifies the fiber pointing the light at the focal plane for the observation)

In [149...]

```
KdePlot('fiber_ID')
```



There is no significant distinction here and this variable should be removed

Based on the above, we have validated the inclusion of all the variables we were not considering removing.

We will proceed with dimensionality reduction using these variables, as well as perform the same dimensionality reduction where we include the other outstanding variables from above and compare the results.

In [149...]

```
X_df_aux = X_df.drop(['fiber_ID', 'field_ID', 'cam_col', 'run_ID', 'class_num'], axis=1)
X_aux = X_df_aux.to_numpy()
X_df_no_aux = X_df.drop(['alpha', 'delta', 'cam_col', 'plate', 'field_ID',
                         'run_ID', 'MJD', 'fiber_ID', 'class_num'], axis=1)
X_no_aux = X_df_no_aux.to_numpy()
```

Calculate 3D t-SNE (with Auxiliary Columns)

In [775...]

```
tsne_3_aux = TSNE(n_components=3, verbose=1, random_state=37)
tsne_3d_aux = tsne_3_aux.fit_transform(X_aux)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 99999 samples in 0.214s...
[t-SNE] Computed neighbors for 99999 samples in 17.158s...
[t-SNE] Computed conditional probabilities for sample 1000 / 99999
[t-SNE] Computed conditional probabilities for sample 2000 / 99999
[t-SNE] Computed conditional probabilities for sample 3000 / 99999
[t-SNE] Computed conditional probabilities for sample 4000 / 99999
[t-SNE] Computed conditional probabilities for sample 5000 / 99999
[t-SNE] Computed conditional probabilities for sample 6000 / 99999
[t-SNE] Computed conditional probabilities for sample 7000 / 99999
[t-SNE] Computed conditional probabilities for sample 8000 / 99999
[t-SNE] Computed conditional probabilities for sample 9000 / 99999
[t-SNE] Computed conditional probabilities for sample 10000 / 99999
[t-SNE] Computed conditional probabilities for sample 11000 / 99999
[t-SNE] Computed conditional probabilities for sample 12000 / 99999
[t-SNE] Computed conditional probabilities for sample 13000 / 99999
[t-SNE] Computed conditional probabilities for sample 14000 / 99999
[t-SNE] Computed conditional probabilities for sample 15000 / 99999
[t-SNE] Computed conditional probabilities for sample 16000 / 99999
[t-SNE] Computed conditional probabilities for sample 17000 / 99999
[t-SNE] Computed conditional probabilities for sample 18000 / 99999
[t-SNE] Computed conditional probabilities for sample 19000 / 99999
[t-SNE] Computed conditional probabilities for sample 20000 / 99999
[t-SNE] Computed conditional probabilities for sample 21000 / 99999
[t-SNE] Computed conditional probabilities for sample 22000 / 99999
[t-SNE] Computed conditional probabilities for sample 23000 / 99999
[t-SNE] Computed conditional probabilities for sample 24000 / 99999
[t-SNE] Computed conditional probabilities for sample 25000 / 99999
[t-SNE] Computed conditional probabilities for sample 26000 / 99999
[t-SNE] Computed conditional probabilities for sample 27000 / 99999
[t-SNE] Computed conditional probabilities for sample 28000 / 99999
[t-SNE] Computed conditional probabilities for sample 29000 / 99999
[t-SNE] Computed conditional probabilities for sample 30000 / 99999
[t-SNE] Computed conditional probabilities for sample 31000 / 99999
[t-SNE] Computed conditional probabilities for sample 32000 / 99999
[t-SNE] Computed conditional probabilities for sample 33000 / 99999
[t-SNE] Computed conditional probabilities for sample 34000 / 99999
[t-SNE] Computed conditional probabilities for sample 35000 / 99999
[t-SNE] Computed conditional probabilities for sample 36000 / 99999
[t-SNE] Computed conditional probabilities for sample 37000 / 99999
[t-SNE] Computed conditional probabilities for sample 38000 / 99999
[t-SNE] Computed conditional probabilities for sample 39000 / 99999
[t-SNE] Computed conditional probabilities for sample 40000 / 99999
[t-SNE] Computed conditional probabilities for sample 41000 / 99999
[t-SNE] Computed conditional probabilities for sample 42000 / 99999
[t-SNE] Computed conditional probabilities for sample 43000 / 99999
[t-SNE] Computed conditional probabilities for sample 44000 / 99999
[t-SNE] Computed conditional probabilities for sample 45000 / 99999
[t-SNE] Computed conditional probabilities for sample 46000 / 99999
[t-SNE] Computed conditional probabilities for sample 47000 / 99999
[t-SNE] Computed conditional probabilities for sample 48000 / 99999
```

```
[t-SNE] Computed conditional probabilities for sample 49000 / 99999
[t-SNE] Computed conditional probabilities for sample 50000 / 99999
[t-SNE] Computed conditional probabilities for sample 51000 / 99999
[t-SNE] Computed conditional probabilities for sample 52000 / 99999
[t-SNE] Computed conditional probabilities for sample 53000 / 99999
[t-SNE] Computed conditional probabilities for sample 54000 / 99999
[t-SNE] Computed conditional probabilities for sample 55000 / 99999
[t-SNE] Computed conditional probabilities for sample 56000 / 99999
[t-SNE] Computed conditional probabilities for sample 57000 / 99999
[t-SNE] Computed conditional probabilities for sample 58000 / 99999
[t-SNE] Computed conditional probabilities for sample 59000 / 99999
[t-SNE] Computed conditional probabilities for sample 60000 / 99999
[t-SNE] Computed conditional probabilities for sample 61000 / 99999
[t-SNE] Computed conditional probabilities for sample 62000 / 99999
[t-SNE] Computed conditional probabilities for sample 63000 / 99999
[t-SNE] Computed conditional probabilities for sample 64000 / 99999
[t-SNE] Computed conditional probabilities for sample 65000 / 99999
[t-SNE] Computed conditional probabilities for sample 66000 / 99999
[t-SNE] Computed conditional probabilities for sample 67000 / 99999
[t-SNE] Computed conditional probabilities for sample 68000 / 99999
[t-SNE] Computed conditional probabilities for sample 69000 / 99999
[t-SNE] Computed conditional probabilities for sample 70000 / 99999
[t-SNE] Computed conditional probabilities for sample 71000 / 99999
[t-SNE] Computed conditional probabilities for sample 72000 / 99999
[t-SNE] Computed conditional probabilities for sample 73000 / 99999
[t-SNE] Computed conditional probabilities for sample 74000 / 99999
[t-SNE] Computed conditional probabilities for sample 75000 / 99999
[t-SNE] Computed conditional probabilities for sample 76000 / 99999
[t-SNE] Computed conditional probabilities for sample 77000 / 99999
[t-SNE] Computed conditional probabilities for sample 78000 / 99999
[t-SNE] Computed conditional probabilities for sample 79000 / 99999
[t-SNE] Computed conditional probabilities for sample 80000 / 99999
[t-SNE] Computed conditional probabilities for sample 81000 / 99999
[t-SNE] Computed conditional probabilities for sample 82000 / 99999
[t-SNE] Computed conditional probabilities for sample 83000 / 99999
[t-SNE] Computed conditional probabilities for sample 84000 / 99999
[t-SNE] Computed conditional probabilities for sample 85000 / 99999
[t-SNE] Computed conditional probabilities for sample 86000 / 99999
[t-SNE] Computed conditional probabilities for sample 87000 / 99999
[t-SNE] Computed conditional probabilities for sample 88000 / 99999
[t-SNE] Computed conditional probabilities for sample 89000 / 99999
[t-SNE] Computed conditional probabilities for sample 90000 / 99999
[t-SNE] Computed conditional probabilities for sample 91000 / 99999
[t-SNE] Computed conditional probabilities for sample 92000 / 99999
[t-SNE] Computed conditional probabilities for sample 93000 / 99999
[t-SNE] Computed conditional probabilities for sample 94000 / 99999
[t-SNE] Computed conditional probabilities for sample 95000 / 99999
[t-SNE] Computed conditional probabilities for sample 96000 / 99999
[t-SNE] Computed conditional probabilities for sample 97000 / 99999
[t-SNE] Computed conditional probabilities for sample 98000 / 99999
[t-SNE] Computed conditional probabilities for sample 99000 / 99999
[t-SNE] Computed conditional probabilities for sample 99999 / 99999
[t-SNE] Mean sigma: 0.237256
[t-SNE] KL divergence after 250 iterations with early exaggeration: 94.544510
[t-SNE] KL divergence after 1000 iterations: 2.222878
```

Create 3D t-SNE Data Frame (with Auxiliary Columns)

In [823...]

```
df_tsne_3d_aux = pd.DataFrame()
```

```
df_tsne_3d_aux['tsne-3d-1'] = tsne_3d_aux[:,0]
df_tsne_3d_aux['tsne-3d-2'] = tsne_3d_aux[:,1]
df_tsne_3d_aux['tsne-3d-3'] = tsne_3d_aux[:,2]
df_tsne_3d_aux['y_num'] = y_num
```

Calculate 3D t-SNE (without Auxiliary Columns)

In [440...]

```
tsne_3 = TSNE(n_components=3, verbose=1, random_state=37)
tsne_3d = tsne_3.fit_transform(X_no_aux)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 99999 samples in 0.107s...
[t-SNE] Computed neighbors for 99999 samples in 4.112s...
[t-SNE] Computed conditional probabilities for sample 1000 / 99999
[t-SNE] Computed conditional probabilities for sample 2000 / 99999
[t-SNE] Computed conditional probabilities for sample 3000 / 99999
[t-SNE] Computed conditional probabilities for sample 4000 / 99999
[t-SNE] Computed conditional probabilities for sample 5000 / 99999
[t-SNE] Computed conditional probabilities for sample 6000 / 99999
[t-SNE] Computed conditional probabilities for sample 7000 / 99999
[t-SNE] Computed conditional probabilities for sample 8000 / 99999
[t-SNE] Computed conditional probabilities for sample 9000 / 99999
[t-SNE] Computed conditional probabilities for sample 10000 / 99999
[t-SNE] Computed conditional probabilities for sample 11000 / 99999
[t-SNE] Computed conditional probabilities for sample 12000 / 99999
[t-SNE] Computed conditional probabilities for sample 13000 / 99999
[t-SNE] Computed conditional probabilities for sample 14000 / 99999
[t-SNE] Computed conditional probabilities for sample 15000 / 99999
[t-SNE] Computed conditional probabilities for sample 16000 / 99999
[t-SNE] Computed conditional probabilities for sample 17000 / 99999
[t-SNE] Computed conditional probabilities for sample 18000 / 99999
[t-SNE] Computed conditional probabilities for sample 19000 / 99999
[t-SNE] Computed conditional probabilities for sample 20000 / 99999
[t-SNE] Computed conditional probabilities for sample 21000 / 99999
[t-SNE] Computed conditional probabilities for sample 22000 / 99999
[t-SNE] Computed conditional probabilities for sample 23000 / 99999
[t-SNE] Computed conditional probabilities for sample 24000 / 99999
[t-SNE] Computed conditional probabilities for sample 25000 / 99999
[t-SNE] Computed conditional probabilities for sample 26000 / 99999
[t-SNE] Computed conditional probabilities for sample 27000 / 99999
[t-SNE] Computed conditional probabilities for sample 28000 / 99999
[t-SNE] Computed conditional probabilities for sample 29000 / 99999
[t-SNE] Computed conditional probabilities for sample 30000 / 99999
[t-SNE] Computed conditional probabilities for sample 31000 / 99999
[t-SNE] Computed conditional probabilities for sample 32000 / 99999
[t-SNE] Computed conditional probabilities for sample 33000 / 99999
[t-SNE] Computed conditional probabilities for sample 34000 / 99999
[t-SNE] Computed conditional probabilities for sample 35000 / 99999
[t-SNE] Computed conditional probabilities for sample 36000 / 99999
[t-SNE] Computed conditional probabilities for sample 37000 / 99999
[t-SNE] Computed conditional probabilities for sample 38000 / 99999
[t-SNE] Computed conditional probabilities for sample 39000 / 99999
[t-SNE] Computed conditional probabilities for sample 40000 / 99999
[t-SNE] Computed conditional probabilities for sample 41000 / 99999
[t-SNE] Computed conditional probabilities for sample 42000 / 99999
[t-SNE] Computed conditional probabilities for sample 43000 / 99999
```


Create 3D t-SNE Data Frame (without Auxiliary Columns)

In [826...]

```
df_tsne_3d = pd.DataFrame()
df_tsne_3d['tsne-3d-1'] = tsne_3d[:,0]
df_tsne_3d['tsne-3d-2'] = tsne_3d[:,1]
df_tsne_3d['tsne-3d-3'] = tsne_3d[:,2]
df_tsne_3d['y_num'] = y_num
```

Scatter Plot First 2D of 3D t-SNE by Class (with/without Auxiliary Columns)

In [149...]

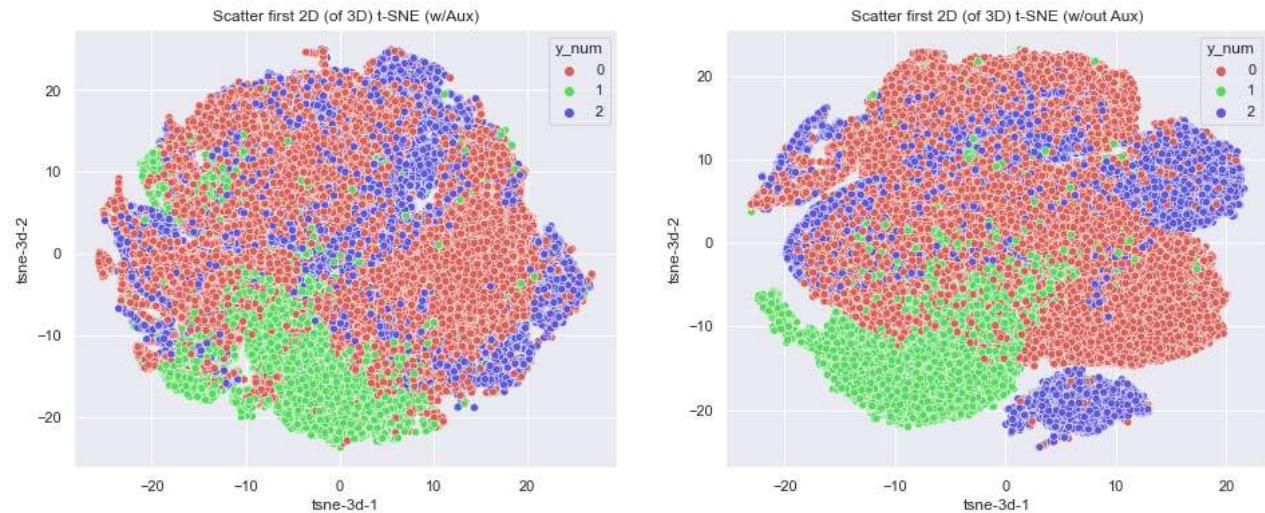
```
fig, axs = plt.subplots(1,2)

sns.scatterplot(x='tsne-3d-1',
                 y='tsne-3d-2',
                 hue='y_num',
                 data=df_tsne_3d_aux,
                 palette=sns.color_palette('hls',3),
                 ax=axs[0]).set_title('Scatter first 2D (of 3D) t-SNE (w/Aux)')

sns.scatterplot(x='tsne-3d-1',
                 y='tsne-3d-2',
                 hue='y_num',
                 data=df_tsne_3d,
                 palette=sns.color_palette('hls',3),
                 ax=axs[1]).set_title('Scatter first 2D (of 3D) t-SNE (w/out Aux)')

print(le.classes_)
```

['GALAXY' 'QSO' 'STAR']



Based on the first two dimensions of the 3D t-SNE's, we can clearly see a more cohesive structure in the plot on the right (without auxiliary columns).

Let's see if this trend carries over to all three dimensions of the 3D t-SNE's.

In [103...]

```
#  
# 3D scatter plot of 3D t-SNE (100% vs 10% alpha)  
#
```

```

def Plot3DtSNE_df(ttle='title', df1=None, azm=-60, elv=30):
    fig = plt.figure()
    ax = fig.add_subplot(121, projection='3d')
    ax.scatter(df1['tsne-3d-1'],
               df1['tsne-3d-2'],
               zs=df1['tsne-3d-3'],
               c=df1['y_num'],
               cmap=cmap_bold,
               label=le.inverse_transform,
               s=1
               )
    plt.title(ttle)
    plt.legend(le.classes_) # TODO
    ax.set_xlabel('Dimension 1')
    ax.set_ylabel('Dimension 2')
    ax.set_zlabel('Dimension 3')
    ax.azim = azm
    ax.elev = elv

    ax = fig.add_subplot(122, projection='3d')
    ax.scatter(df1['tsne-3d-1'],
               df1['tsne-3d-2'],
               zs=df1['tsne-3d-3'],
               c=df1['y_num'],
               cmap=cmap_bold,
               label=le.inverse_transform,
               s=1,
               alpha=0.1
               )
    plt.title(ttle)
    plt.legend(le.classes_) # TODO
    ax.set_xlabel('Dimension 1')
    ax.set_ylabel('Dimension 2')
    ax.set_zlabel('Dimension 3')
    ax.azim = azm
    ax.elev = elv
    return ax

```

In [103...]

cmap_bold

Out[103...]



under

bad

over

In [104...]

#	GALAXY	QSO	STAR
---	--------	-----	------

3D Scatter Plot w/Aux

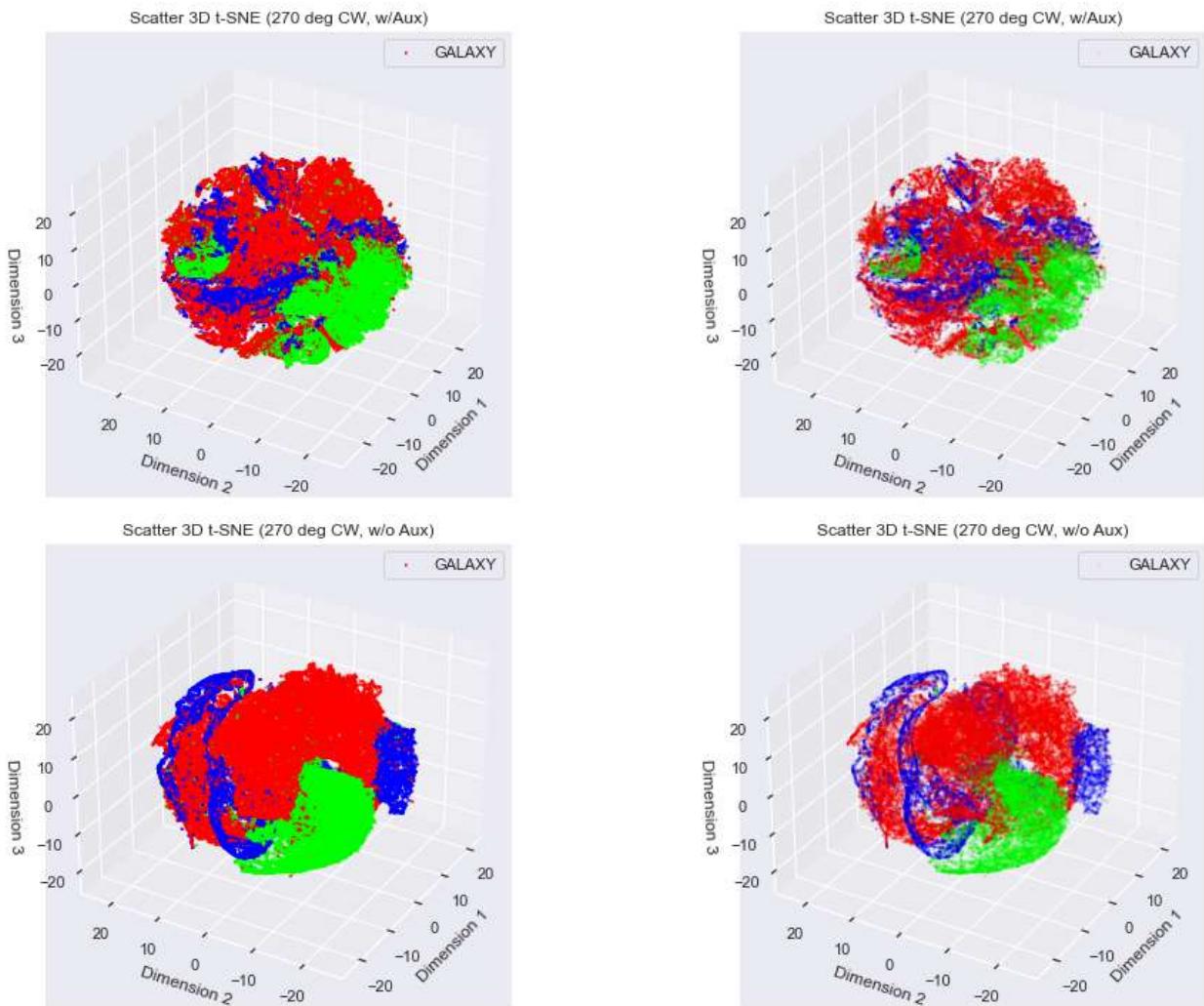
alpha, delta, u, g, r, i, z, redshift, plate, MJD

3D Scatter Plot w/o Aux

u, g, r, i, z, redshift

In [149...]

```
Plot3DtSNE_df('Scatter 3D t-SNE (270 deg CW, w/Aux)', df_tsne_3d_aux, 210)
Plot3DtSNE_df('Scatter 3D t-SNE (270 deg CW, w/o Aux)', df_tsne_3d, 210)
print()
```



We can see, particularly for the blue points (STAR), in the top two graphs, that the added variance/noise appears to prevent a cohesive structure from forming.

This is in contrast to the bottom two graphs, where you can clearly see a cohesive structure.

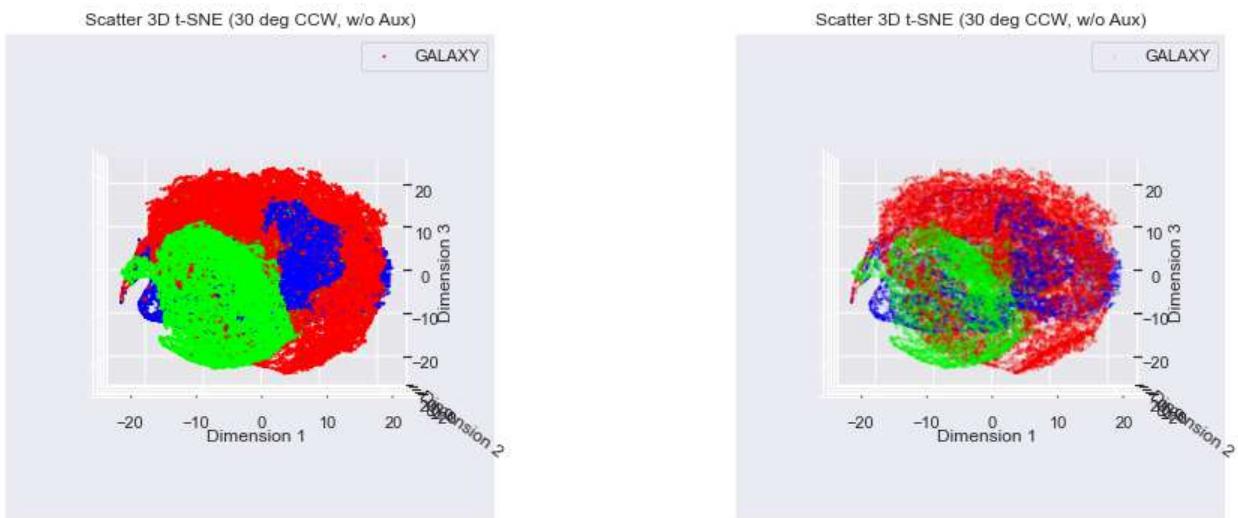
Below, we can follow the blue structure through the rotations of the 3D t-SNE.

3D Scatter Plot w/o Aux

$u, g, r, i, z, \text{redshift}$

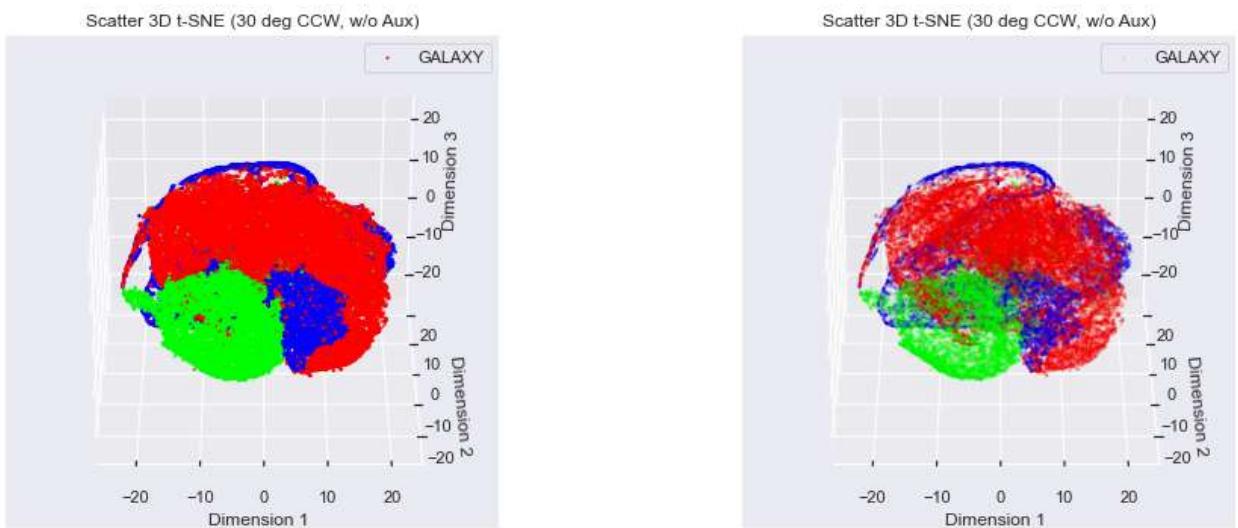
In [104...]

```
Plot3DtSNE_df('Scatter 3D t-SNE (30 deg CCW, w/o Aux)', df_tsne_3d, -90, 0)  
print()
```



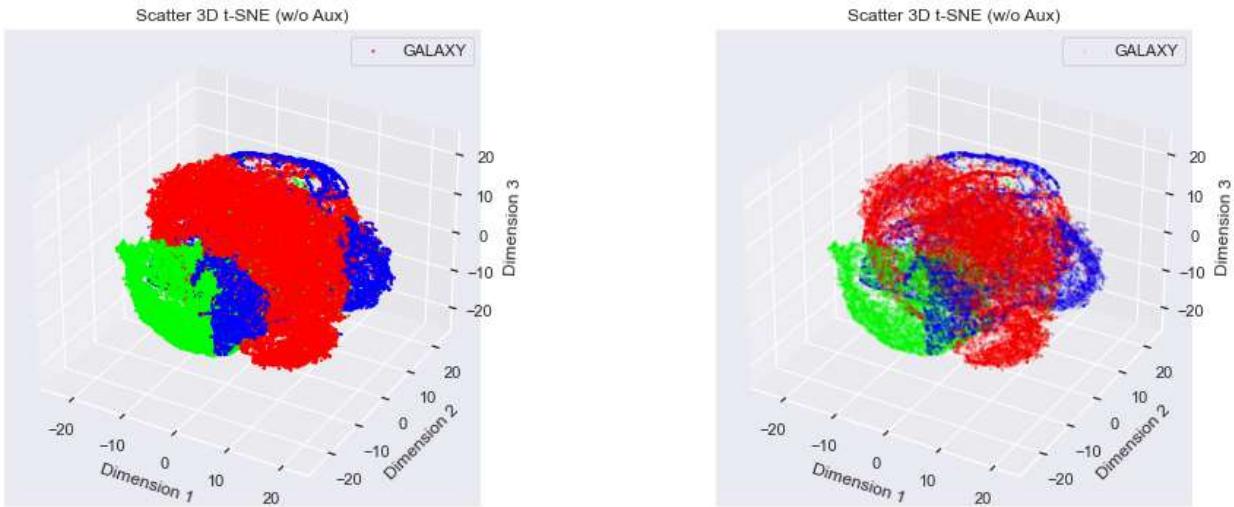
In [104...]

```
Plot3DtSNE_df('Scatter 3D t-SNE (30 deg CCW, w/o Aux)', df_tsne_3d, -90)  
print()
```

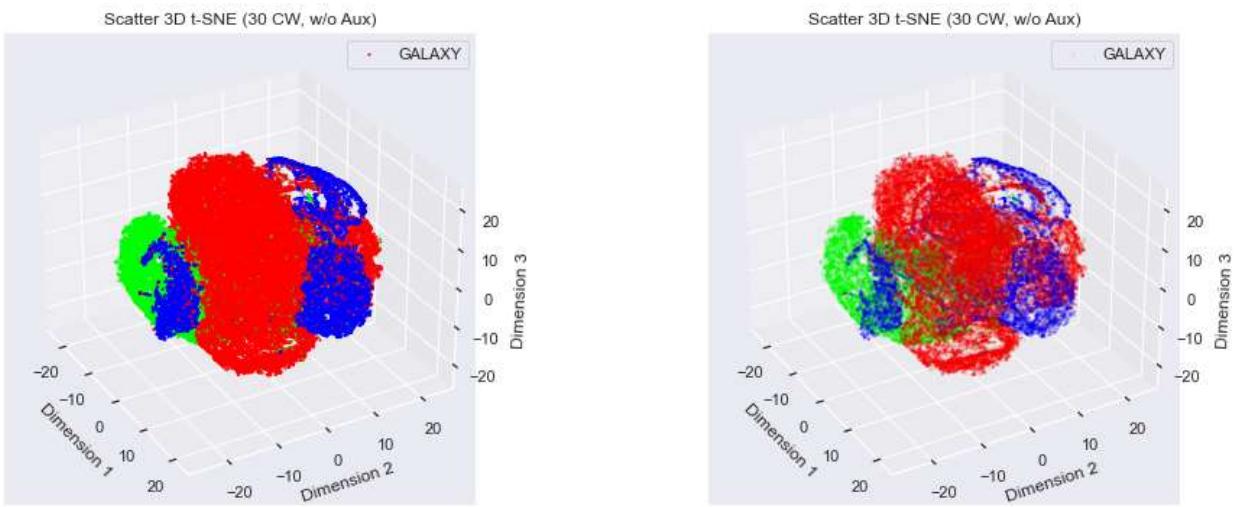


In [104...]

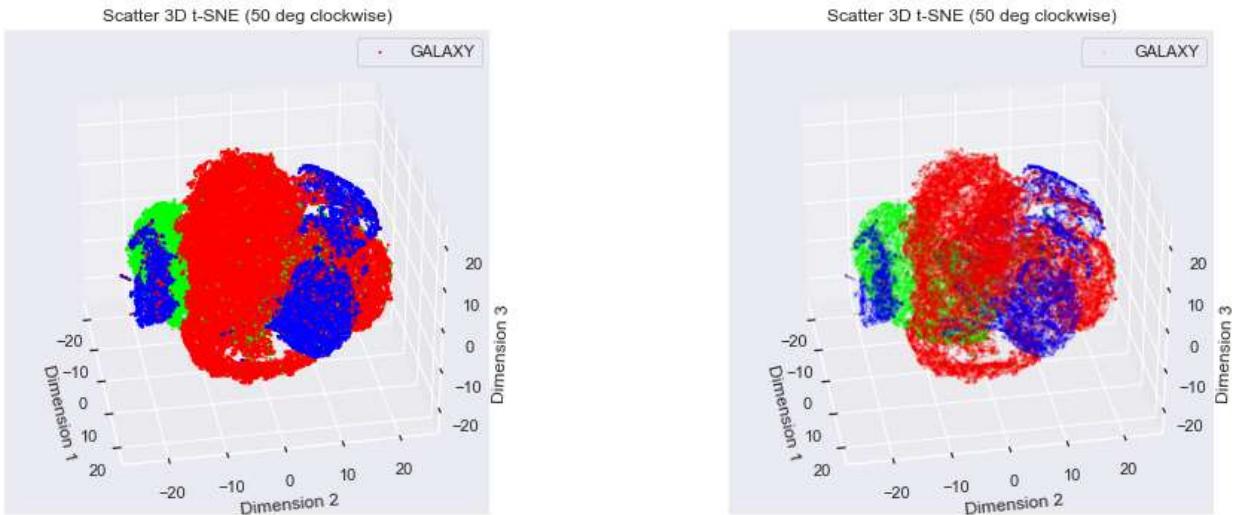
```
Plot3DtSNE_df('Scatter 3D t-SNE (w/o Aux)', df_tsne_3d)
print()
```



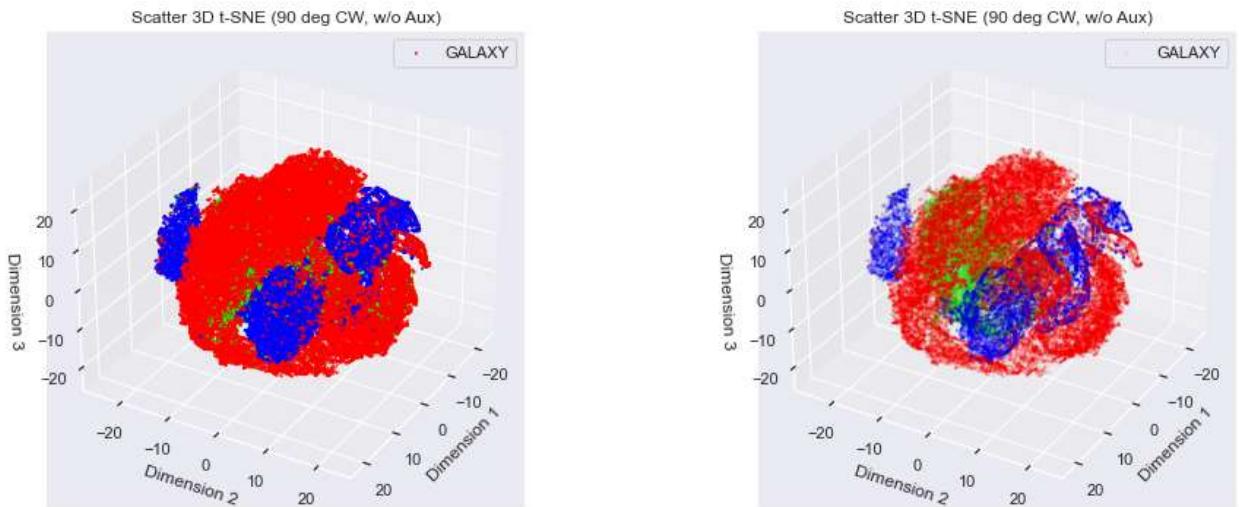
```
In [104...  
Plot3DtSNE_df('Scatter 3D t-SNE (30 CW, w/o Aux)', df_tsne_3d, -30)  
print()
```



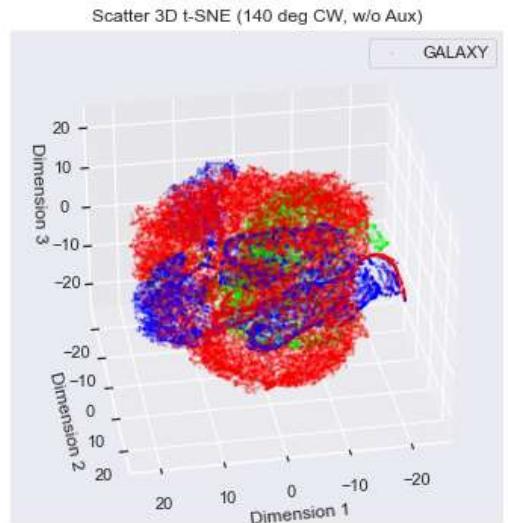
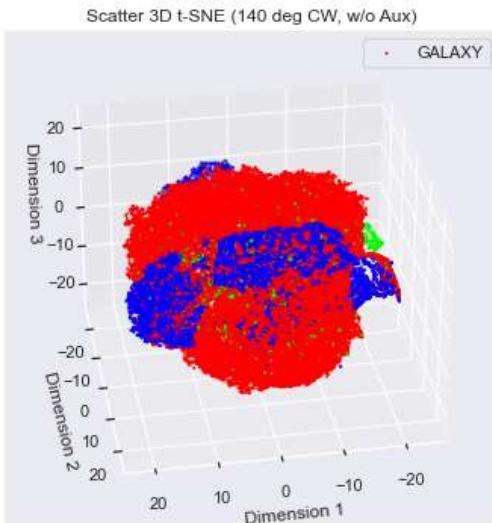
```
In [104...  
Plot3DtSNE_df('Scatter 3D t-SNE (50 deg clockwise)', df_tsne_3d, -10)  
print()
```



```
In [104...  
Plot3DtSNE_df('Scatter 3D t-SNE (90 deg CW, w/o Aux)', df_tsne_3d, 30)  
print()
```

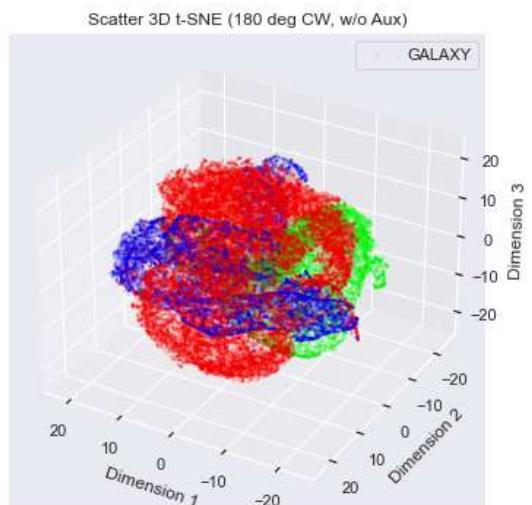
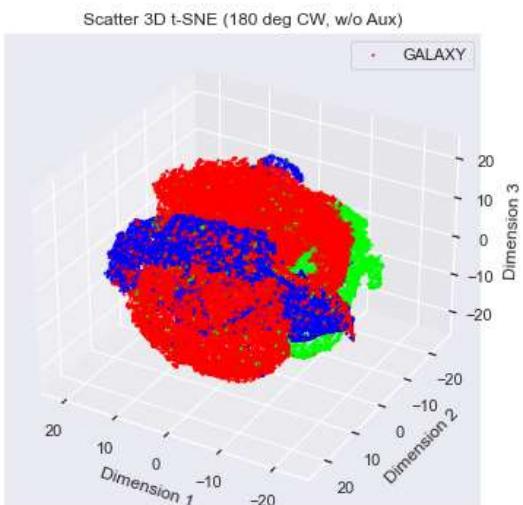


```
In [104...  
Plot3DtSNE_df('Scatter 3D t-SNE (140 deg CW, w/o Aux)', df_tsne_3d, 80)  
print()
```



In [104]:

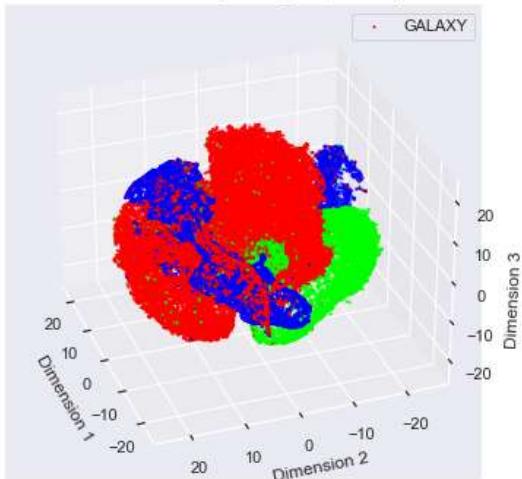
```
Plot3DtSNE_df('Scatter 3D t-SNE (180 deg CW, w/o Aux)', df_tsne_3d, 120)  
print()
```



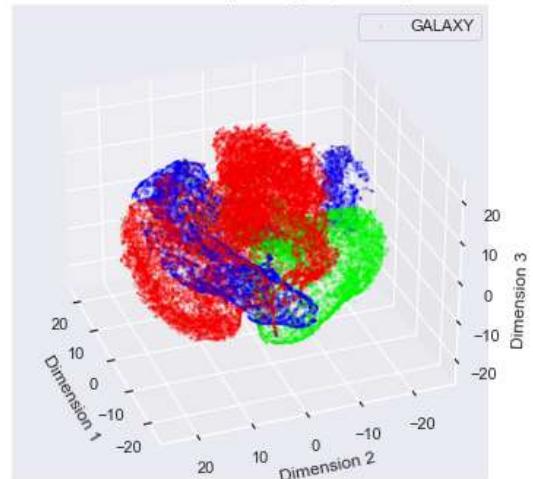
In [105]:

```
Plot3DtSNE_df('Scatter 3D t-SNE (220 deg CW, w/o Aux)', df_tsne_3d, 160)  
print()
```

Scatter 3D t-SNE (220 deg CW, w/o Aux)

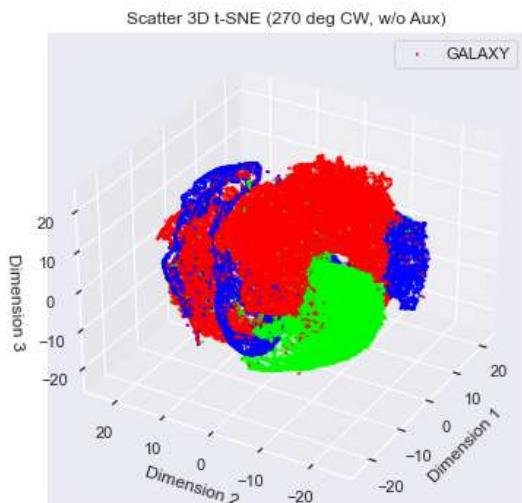


Scatter 3D t-SNE (220 deg CW, w/o Aux)

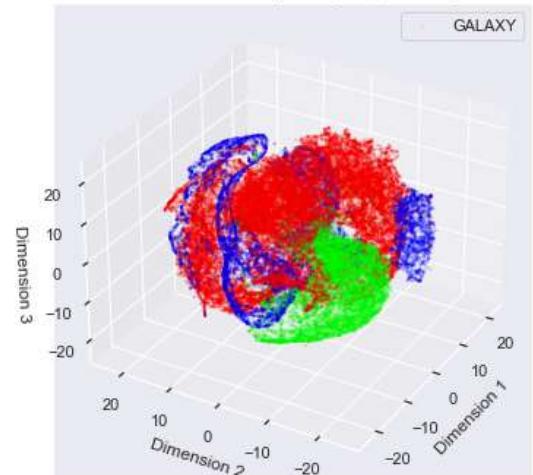


In [105...]

```
Plot3DtSNE_df('Scatter 3D t-SNE (270 deg CW, w/o Aux)', df_tsne_3d, 210)  
print()
```

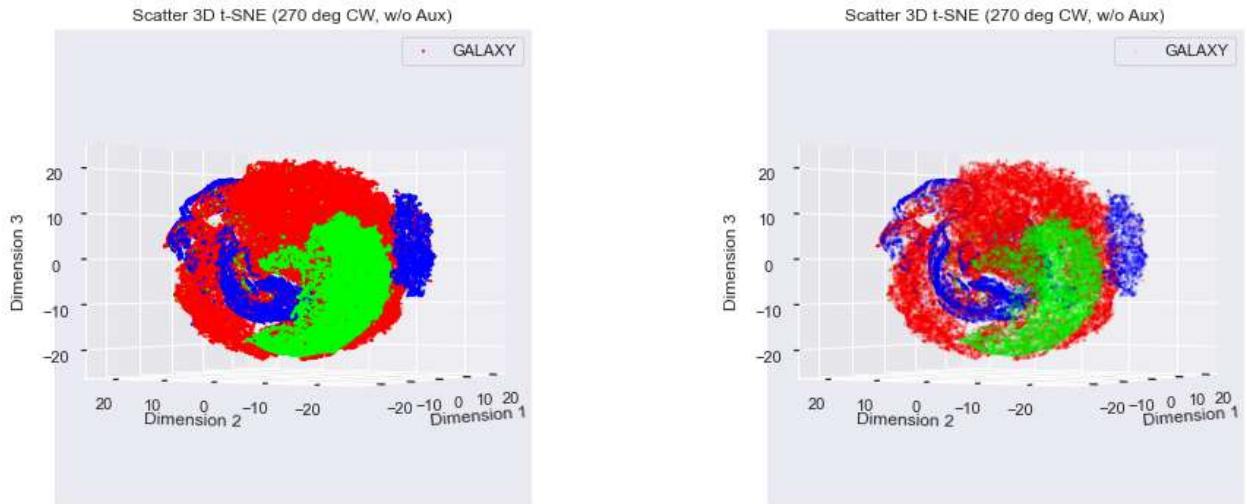


Scatter 3D t-SNE (270 deg CW, w/o Aux)



In [105...]

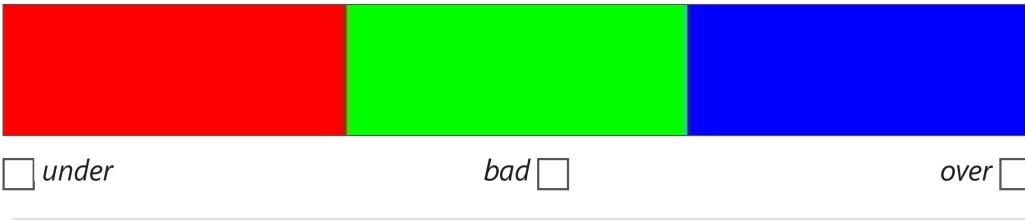
```
Plot3DtSNE_df('Scatter 3D t-SNE (270 deg CW, w/o Aux)', df_tsne_3d, 210, 0)  
print()
```



In [105...]

cmap_bold

Out[105...]



In [105...]

#	GALAXY	QSO	STAR
---	--------	-----	------

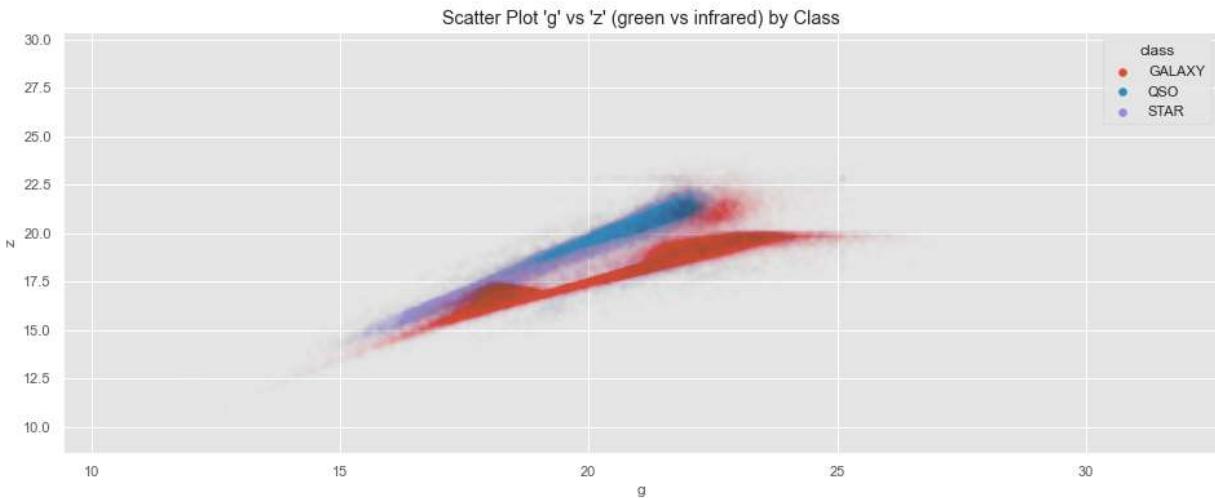
The `u`, `g`, `r`, `i`, `z`, `redshift` t-SNE clearly has a more cohesive structure than the t-SNE that included `alpha`, `delta`, `plate`, and `MJD` in addition the photometric/redshift variables.

This is particularly true for the entity corresponding to `STAR`, as well as `QSO`.

We can see in the `g` vs `z` scatter plot **below** that **'STAR'** and **'QSO'** have a particular relationship, in some dimensions, that would become harder to distinguish as more dimensions/variance is added through variables that do not directly correspond to the true underlying trend.

In [149...]

```
plt.style.use('ggplot')
sns.scatterplot(x='g',
                 y='z',
                 data=df,
                 hue=df['class'],
                 alpha=.002).set_title('Scatter Plot \'g\' vs \'z\' (green vs infrared)')
print()
```



Notice the ring of STAR around the QSO cluster (mentioned above)

t-SNE from First 3 Principle Components

Calculate 3D t-SNE from 3D PCA

In [145...]

```
tsne_3_pca = TSNE(n_components=3, verbose=1, random_state=37)
tsne_3d_pca = tsne_3_pca.fit_transform(df_pca_wout_au.drop('y_num', axis=1))
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 99999 samples in 0.062s...
[t-SNE] Computed neighbors for 99999 samples in 1.614s...
[t-SNE] Computed conditional probabilities for sample 1000 / 99999
[t-SNE] Computed conditional probabilities for sample 2000 / 99999
[t-SNE] Computed conditional probabilities for sample 3000 / 99999
[t-SNE] Computed conditional probabilities for sample 4000 / 99999
[t-SNE] Computed conditional probabilities for sample 5000 / 99999
[t-SNE] Computed conditional probabilities for sample 6000 / 99999
[t-SNE] Computed conditional probabilities for sample 7000 / 99999
[t-SNE] Computed conditional probabilities for sample 8000 / 99999
[t-SNE] Computed conditional probabilities for sample 9000 / 99999
[t-SNE] Computed conditional probabilities for sample 10000 / 99999
[t-SNE] Computed conditional probabilities for sample 11000 / 99999
[t-SNE] Computed conditional probabilities for sample 12000 / 99999
[t-SNE] Computed conditional probabilities for sample 13000 / 99999
[t-SNE] Computed conditional probabilities for sample 14000 / 99999
[t-SNE] Computed conditional probabilities for sample 15000 / 99999
[t-SNE] Computed conditional probabilities for sample 16000 / 99999
[t-SNE] Computed conditional probabilities for sample 17000 / 99999
[t-SNE] Computed conditional probabilities for sample 18000 / 99999
[t-SNE] Computed conditional probabilities for sample 19000 / 99999
[t-SNE] Computed conditional probabilities for sample 20000 / 99999
[t-SNE] Computed conditional probabilities for sample 21000 / 99999
[t-SNE] Computed conditional probabilities for sample 22000 / 99999
[t-SNE] Computed conditional probabilities for sample 23000 / 99999
[t-SNE] Computed conditional probabilities for sample 24000 / 99999
[t-SNE] Computed conditional probabilities for sample 25000 / 99999
[t-SNE] Computed conditional probabilities for sample 26000 / 99999
[t-SNE] Computed conditional probabilities for sample 27000 / 99999
```



```
[t-SNE] Computed conditional probabilities for sample 88000 / 99999
[t-SNE] Computed conditional probabilities for sample 89000 / 99999
[t-SNE] Computed conditional probabilities for sample 90000 / 99999
[t-SNE] Computed conditional probabilities for sample 91000 / 99999
[t-SNE] Computed conditional probabilities for sample 92000 / 99999
[t-SNE] Computed conditional probabilities for sample 93000 / 99999
[t-SNE] Computed conditional probabilities for sample 94000 / 99999
[t-SNE] Computed conditional probabilities for sample 95000 / 99999
[t-SNE] Computed conditional probabilities for sample 96000 / 99999
[t-SNE] Computed conditional probabilities for sample 97000 / 99999
[t-SNE] Computed conditional probabilities for sample 98000 / 99999
[t-SNE] Computed conditional probabilities for sample 99000 / 99999
[t-SNE] Computed conditional probabilities for sample 99999 / 99999
[t-SNE] Mean sigma: 0.043916
[t-SNE] KL divergence after 250 iterations with early exaggeration: 91.936081
[t-SNE] KL divergence after 1000 iterations: 2.008830
```

Create Data Frame for 3D t-SNE from 3D PCA

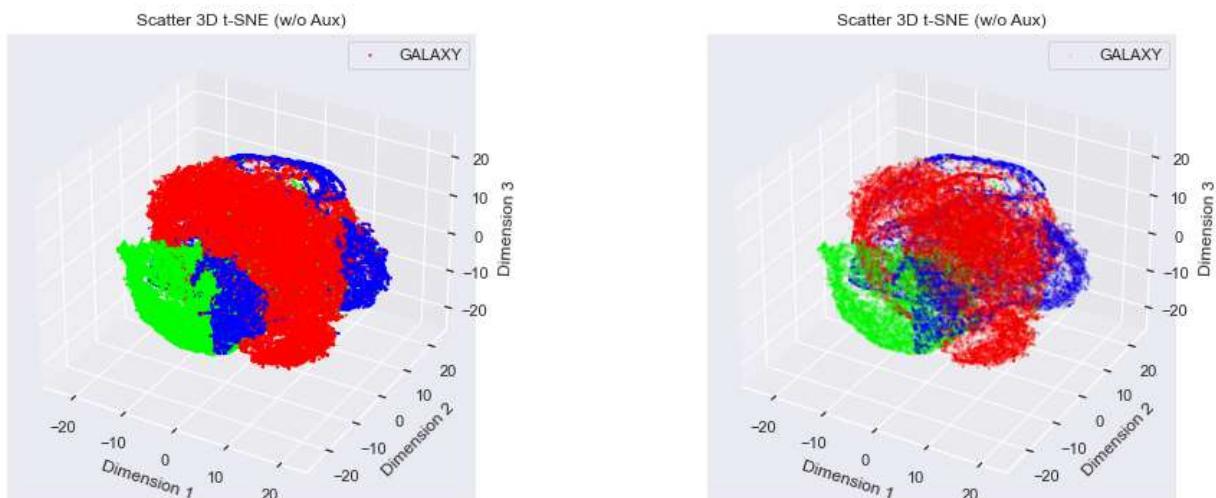
In [146...]

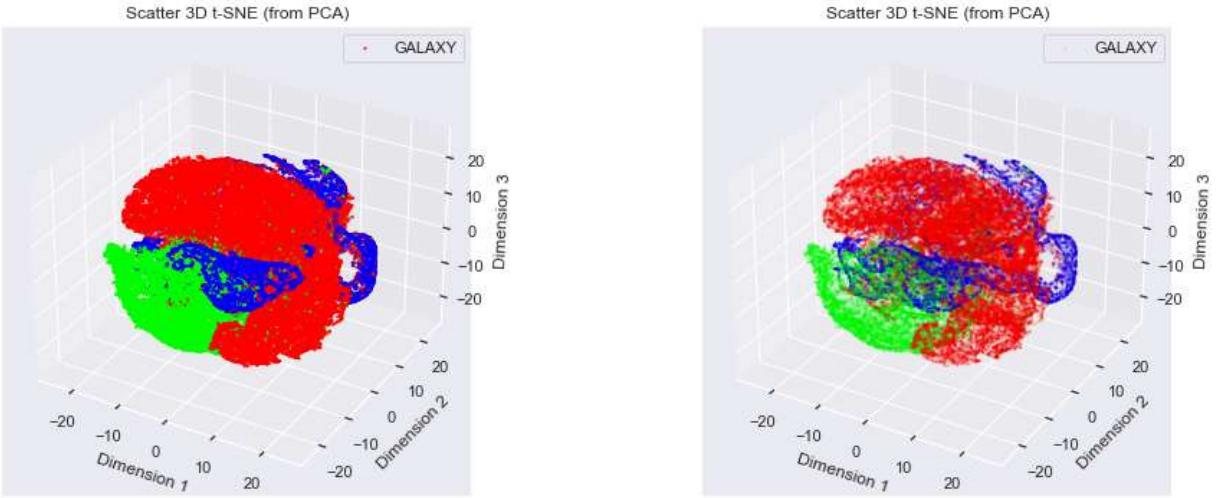
```
df_tsne_3d_pca = pd.DataFrame()
df_tsne_3d_pca['tsne-3d-1'] = tsne_3d_pca[:,0]
df_tsne_3d_pca['tsne-3d-2'] = tsne_3d_pca[:,1]
df_tsne_3d_pca['tsne-3d-3'] = tsne_3d_pca[:,2]
df_tsne_3d_pca['y_num'] = y_num
```

Compare 3D t-SNE (from 3D PCA) vs 3D t-SNE (from Scaled r,g,u,i,z,redshift)

In [146...]

```
Plot3DtSNE_df('Scatter 3D t-SNE (w/o Aux)', df_tsne_3d)
Plot3DtSNE_df('Scatter 3D t-SNE (from PCA)', df_tsne_3d_pca)
print()
```



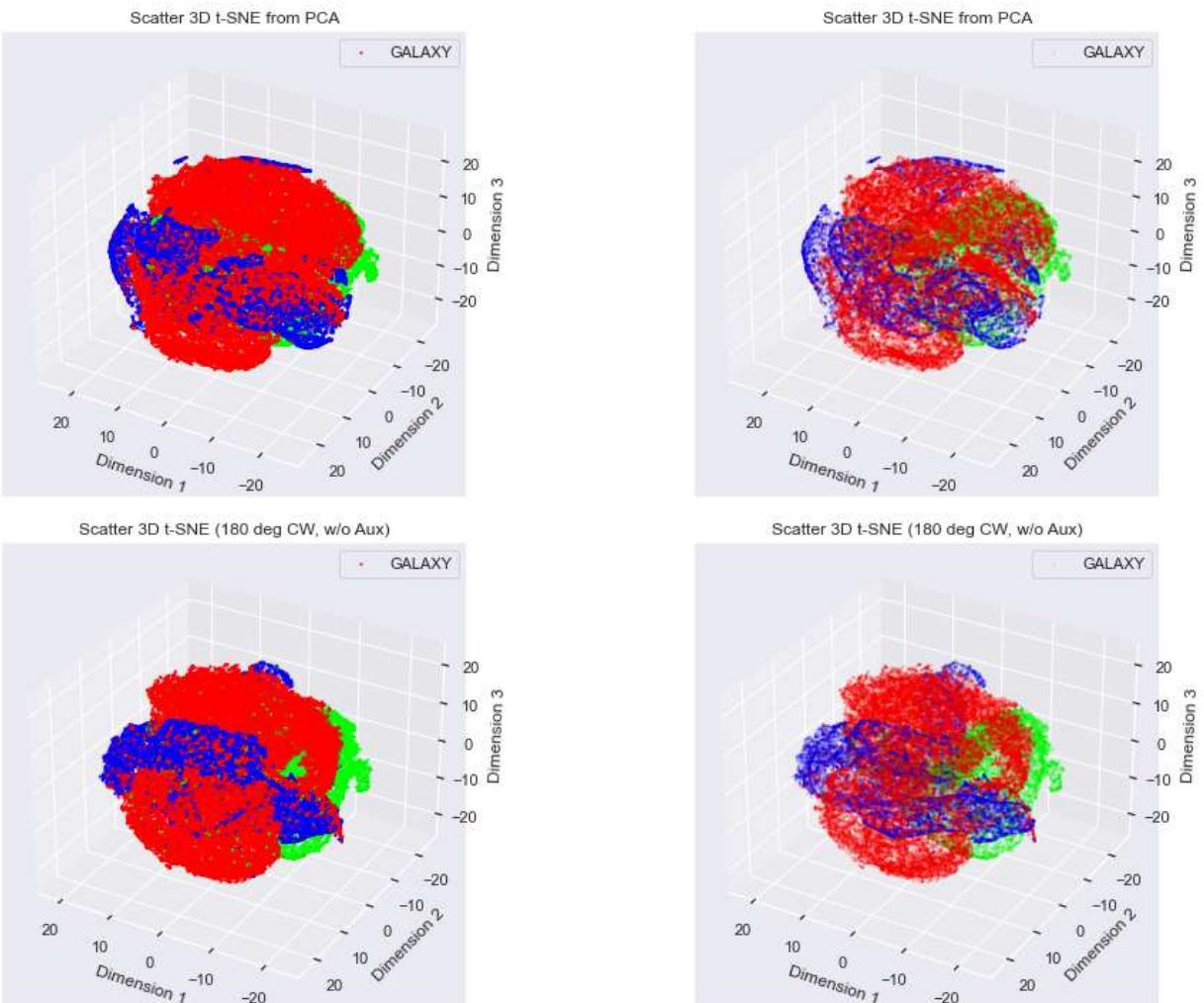


We can clearly see that there is potentially a more cohesive structure in the results of our 3D t-SNE from our first three principle components.

Let's look at the other side of our 3D scatter plot.

In [146]...

```
Plot3DtSNE_df('Scatter 3D t-SNE (180 deg CW, from PCA)', df_tsne_3d_pca, 120)
Plot3DtSNE_df('Scatter 3D t-SNE (180 deg CW, w/o Aux)', df_tsne_3d, 120)
print()
```



It is less clear whether our 3D t-SNE from $u, g, r, i, z, \text{redshift}$ has a more cohesive structure than our PCA based t-SNE, from this angle.

It is worth noting that we may be able to achieve clearer results if we did not constrain our PCA to three principal components and provide all of them to t-SNE. However, we have sufficiently validated the variables selected above.

Clustering

Cluster PCA

Cluster t-SNE

Cluster t-SNE on PCA

Compare with Base-Truth

In [121...]

```
def KMeans3K(X_, y_):
    kmeans_model = KMeans(init='k-means++', n_clusters=3, n_init=10, random_state=37)
    kmeans_model.fit(X_)
    y_pred = kmeans_model.predict(X_)
    print('V-Measure Score: ', metrics.v_measure_score(labels_true=y_, labels_pred=kmeans_model.labels_))
    print('Accuracy Score: ', metrics.accuracy_score(y_, y_pred))
    return kmeans_model
```

KMeans

Sorted by V-Measure Score

- $X_{\text{no_aux}}$ /3D PCA
- 3D t-SNE
- X_{aux}
- 2D t-SNE
- 3D t-SNE on PCA
- 2D t-SNE w/Aux
- 3D t-SNE w/Aux

X Scaled w/out Aux

In [150...]

```
KMeans3K(X_no_aux, y_num)
```

```
V-Measure Score: 0.30703665139691416
Accuracy Score: 0.44782447824478244
Out[150...]
```

Suspiciously the same as below, need to verify (is KMeans doing a PCA?)

3D PCA

```
In [150...]: KMeans3K(X_PCA_3d, y_num)
```

```
V-Measure Score: 0.30703665139691416
Accuracy Score: 0.44782447824478244
Out[150...]
```

```
In [150...]: print(X_no_aux[0,:])
print(X_PCA_3d[0,:])
```

```
[0.79879842 0.80678207 0.40395284 0.04600118 0.01399907 0.07954924]
[-0.88132996 -0.54926534 0.56192322 -0.22867185 0.1010563 -0.06401947]
```

It appears KMeans may be doing a PCA

3D t-SNE

```
In [122...]: KMeans3K(tsne_3d, y_num)
```

```
V-Measure Score: 0.16699460723575163
Accuracy Score: 0.26049260492604925
Out[122...]
```

X Scaled w/Aux

```
In [121...]: KMeans3K(X_aux, y_num)
```

```
V-Measure Score: 0.1482924637078717
Accuracy Score: 0.4248142481424814
Out[121...]
```

2D t-SNE

```
In [121...]: KMeans3K(tsne_2d, y_num)
```

```
V-Measure Score: 0.14884707165058883
Accuracy Score: 0.3103331033310333
Out[121...]
```

3D t-SNE on PCA

```
In [150...]: KMeans3K(df_tsne_3d_pca.drop('y_num', axis=1), y_num)
```

```
V-Measure Score: 0.09576087505968077
Accuracy Score: 0.3100431004310043
```

```
Out[150... KMeans(n_clusters=3, random_state=37)
```

2D t-SNE (w/Aux)

```
In [122... KMeans3K(tsne_2d_aux, y_num)
```

```
V-Measure Score: 0.05277012708157446
```

```
Accuracy Score: 0.25211252112521126
```

```
Out[122... KMeans(n_clusters=3, random_state=37)
```

3D t-SNE (w/Aux)

```
In [121... KMeans3K(tsne_3d_aux, y_num)
```

```
V-Measure Score: 0.005559777477021477
```

```
Accuracy Score: 0.3012030120301203
```

```
Out[121... KMeans(n_clusters=3, random_state=37)
```

Based on the above V-Measure Scores, it appears that the scaled values of $u, g, r, i, z, redshift$ and a 3D PCA from the variables perform the best, with a V-Measure Score of 0.307 (in both cases).

Given that a variety of parameters have been tried on KMeans with little difference and that our best V-Measure Score is low indicates that something else may be causing issue. Additionally, the fact that this is the case for KMeans clustering on our data and variations of our data in reduced dimensions indicates that we need to take other considerations, such as validating our feature selection, looking for multivariate outliers, mitigating class imbalance, and any other aspects that may be causing such poor results.

Univariate Feature Selection

```
In [127... X_df_all = df.drop('class', axis=1)
```

```
In [127... # encode any categorcal variables to discrete integers (excluding response variable)
for column_name in X_df_all.select_dtypes('object'):
    X_df_all[column_name], _ = X_df_all[column_name].factorize()
```

Calculate Mutual Information between variables (0 means variables are independent)

`sklearn.feature_selection.f_classif` is parametric and uses F-test to estimate linear dependency between two random variables.

`sklearn.feature_selection.mutual_info_classif` is nonparametric and can capture any kind of statistical dependency--though requiring larger sample sizes for accurate estimates.

[Univariate Feature Selection](#)

```
In [127...]
mutual_info_scores = mutual_info_classif(X_df_all, y, discrete_features = X_df_all.dtypes == 'category')
mutual_info_scores = pd.Series(mutual_info_scores, index=X_df_all.columns, name='MI')
mutual_info_scores = mutual_info_scores.sort_values(ascending=False)
mutual_info_scores
```

```
Out[127...]
redshift      0.801814
plate         0.275445
MJD           0.192275
z              0.145658
run_ID        0.143826
g              0.120600
i              0.109588
u              0.100328
r              0.075618
fiber_ID      0.051621
delta          0.044373
alpha          0.040456
field_ID      0.006539
cam_col        0.001044
Name: MI Scores, dtype: float64
```

Based on the above, it looks like removing `alpha` and `delta`, as well as adding `run_ID` may be worth trying.

With Auxiliary Columns == `alpha`, `delta`, `u`, `g`, `r`, `i`, `z`, `redshift`, `plate`, `MJD`

Without Auxiliary Columns == `u`, `g`, `r`, `i`, `z`, `redshift`

Proposed Columns == `u`, `g`, `r`, `i`, `z`, `redshift`, `plate`, `MJD`, `run_ID`

```
In [128...]
X_mi = X_df_all.loc[:,['u','g','r','i','z','run_ID','redshift','plate','MJD']].apply(KMeans3K)
```

```
In [128...]
KMeans3K(X_mi, y_num)
```

V-Measure Score: 0.15018334565259933

Accuracy Score: 0.4248142481424814

```
Out[128...]
KMeans(n_clusters=3, random_state=37)
```

This is better than with Auxiliary Columns (V-Score 0.148), but not with out Auxiliary Columns (V-Score 0.307)

```
In [129...]
mi_tsne_3 = TSNE(n_components=3, verbose=1, random_state=37)
mi_tsne_3d = mi_tsne_3.fit_transform(X_mi)
```

[t-SNE] Computing 91 nearest neighbors...

[t-SNE] Indexed 99999 samples in 0.157s...

[t-SNE] Computed neighbors for 99999 samples in 10.192s...

[t-SNE] Computed conditional probabilities for sample 1000 / 99999

[t-SNE] Computed conditional probabilities for sample 2000 / 99999

[t-SNE] Computed conditional probabilities for sample 3000 / 99999

[t-SNE] Computed conditional probabilities for sample 4000 / 99999

[t-SNE] Computed conditional probabilities for sample 5000 / 99999

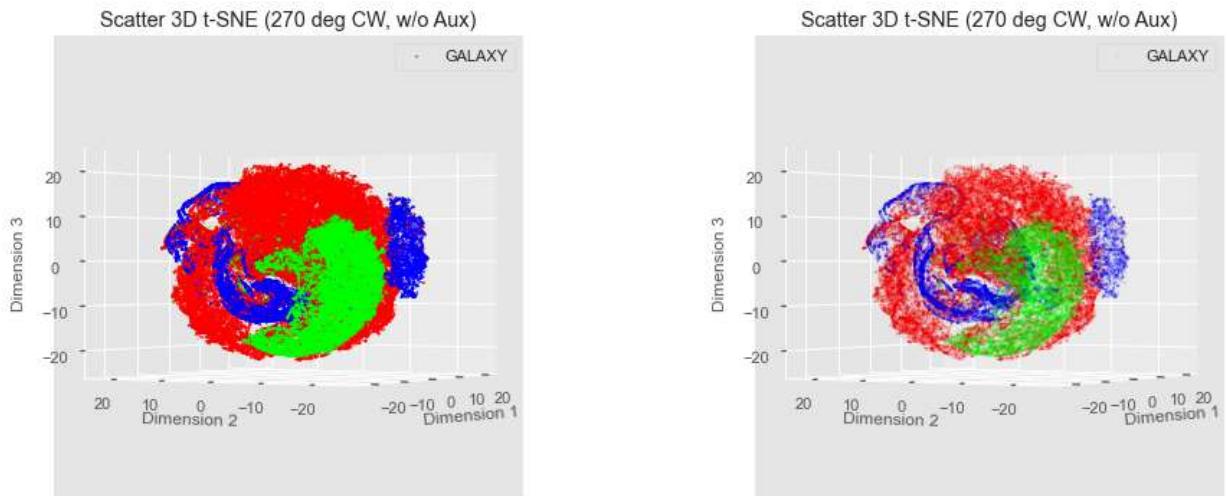
[t-SNE] Computed conditional probabilities for sample 6000 / 99999

[t-SNE] Computed conditional probabilities for sample 7000 / 99999


```
[t-SNE] Computed conditional probabilities for sample 68000 / 99999
[t-SNE] Computed conditional probabilities for sample 69000 / 99999
[t-SNE] Computed conditional probabilities for sample 70000 / 99999
[t-SNE] Computed conditional probabilities for sample 71000 / 99999
[t-SNE] Computed conditional probabilities for sample 72000 / 99999
[t-SNE] Computed conditional probabilities for sample 73000 / 99999
[t-SNE] Computed conditional probabilities for sample 74000 / 99999
[t-SNE] Computed conditional probabilities for sample 75000 / 99999
[t-SNE] Computed conditional probabilities for sample 76000 / 99999
[t-SNE] Computed conditional probabilities for sample 77000 / 99999
[t-SNE] Computed conditional probabilities for sample 78000 / 99999
[t-SNE] Computed conditional probabilities for sample 79000 / 99999
[t-SNE] Computed conditional probabilities for sample 80000 / 99999
[t-SNE] Computed conditional probabilities for sample 81000 / 99999
[t-SNE] Computed conditional probabilities for sample 82000 / 99999
[t-SNE] Computed conditional probabilities for sample 83000 / 99999
[t-SNE] Computed conditional probabilities for sample 84000 / 99999
[t-SNE] Computed conditional probabilities for sample 85000 / 99999
[t-SNE] Computed conditional probabilities for sample 86000 / 99999
[t-SNE] Computed conditional probabilities for sample 87000 / 99999
[t-SNE] Computed conditional probabilities for sample 88000 / 99999
[t-SNE] Computed conditional probabilities for sample 89000 / 99999
[t-SNE] Computed conditional probabilities for sample 90000 / 99999
[t-SNE] Computed conditional probabilities for sample 91000 / 99999
[t-SNE] Computed conditional probabilities for sample 92000 / 99999
[t-SNE] Computed conditional probabilities for sample 93000 / 99999
[t-SNE] Computed conditional probabilities for sample 94000 / 99999
[t-SNE] Computed conditional probabilities for sample 95000 / 99999
[t-SNE] Computed conditional probabilities for sample 96000 / 99999
[t-SNE] Computed conditional probabilities for sample 97000 / 99999
[t-SNE] Computed conditional probabilities for sample 98000 / 99999
[t-SNE] Computed conditional probabilities for sample 99000 / 99999
[t-SNE] Computed conditional probabilities for sample 99999 / 99999
[t-SNE] Mean sigma: 0.176875
[t-SNE] KL divergence after 250 iterations with early exaggeration: 96.042648
[t-SNE] KL divergence after 1000 iterations: 2.264761
```

In [129...]

```
Plot3DtSNE_df('Scatter 3D t-SNE (270 deg CW, w/o Aux)', df_tsne_3d, 210, 0)
print()
```



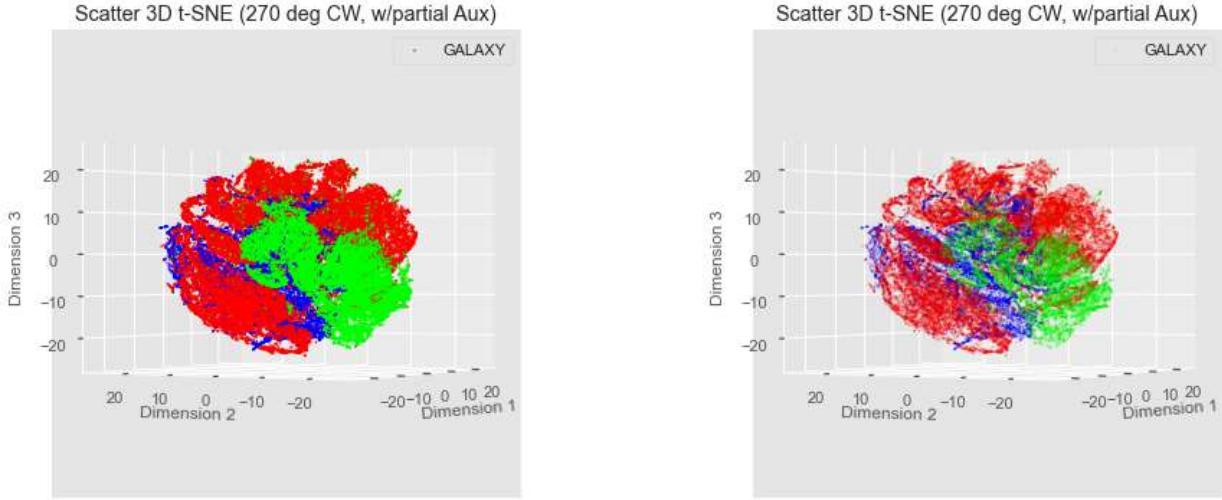
In [129...]

```
df_mi_tsne_3d = pd.DataFrame()
```

```

df_mi_tsne_3d['tsne-3d-1'] = mi_tsne_3d[:,0]
df_mi_tsne_3d['tsne-3d-2'] = mi_tsne_3d[:,1]
df_mi_tsne_3d['tsne-3d-3'] = mi_tsne_3d[:,2]
df_mi_tsne_3d['y_num'] = y_num
Plot3DTsNE_df('Scatter 3D t-SNE (270 deg CW, w/partial Aux)', df_mi_tsne_3d, 210, 0)
print()

```



Based on the above KMeans scores being lower and the 3D t-SNE being less cohesive, our selection of features has been validated and includes:

u, g, r, i, z, and redshift.

Multivariate Outlier Detection

In [22]:

```
df_features = df.loc[:,['u','g','r','i','z','redshift','class']]
```

In [24]:

```
clf = LocalOutlierFactor()
y_lof = clf.fit_predict(df_features.drop('class',axis=1).apply(scale))
```

In [25]:

```
Lof_score = pd.DataFrame()
Lof_score['score'] = clf.negative_outlier_factor_
```

In [26]:

```
Lof_score.describe()
```

Out[26]:

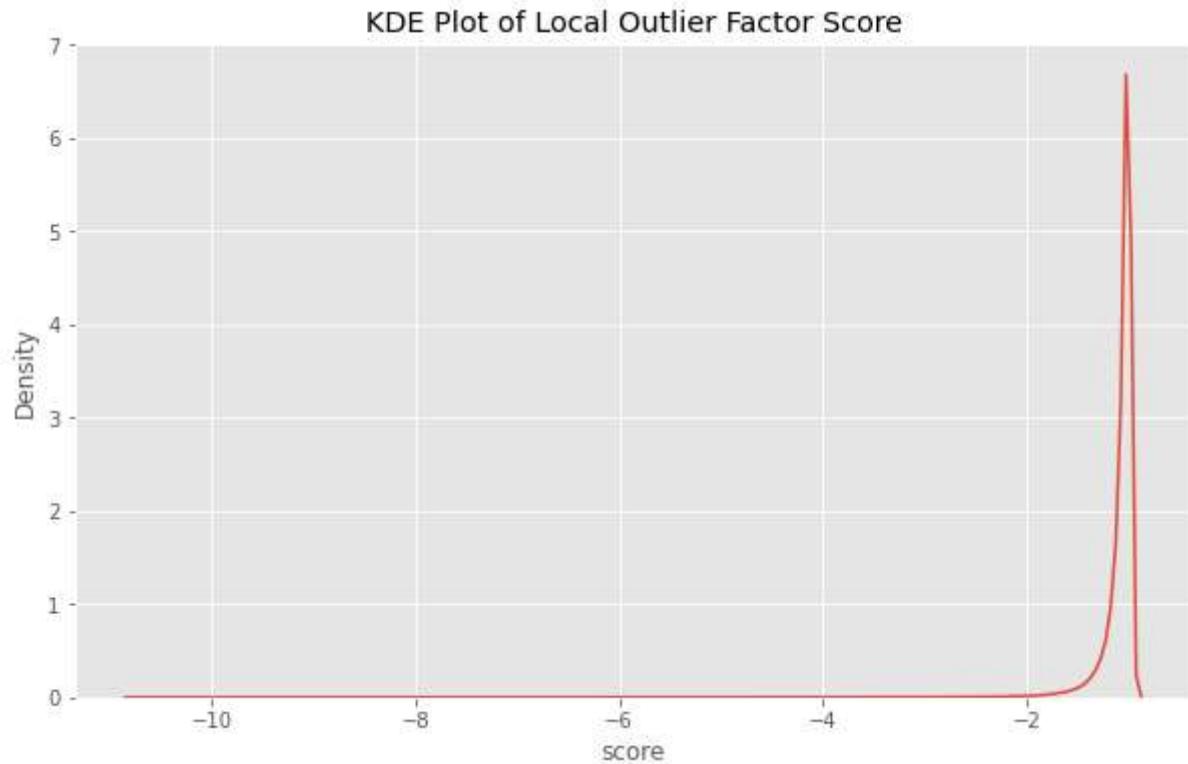
	score
count	99999.000000
mean	-1.090468
std	0.238155
min	-10.780989
25%	-1.098164
50%	-1.030006

```
75%      -0.999547  
max      -0.946417
```

KDE Plot Local Outlier Factor to Pick Cutoff

In [27]:

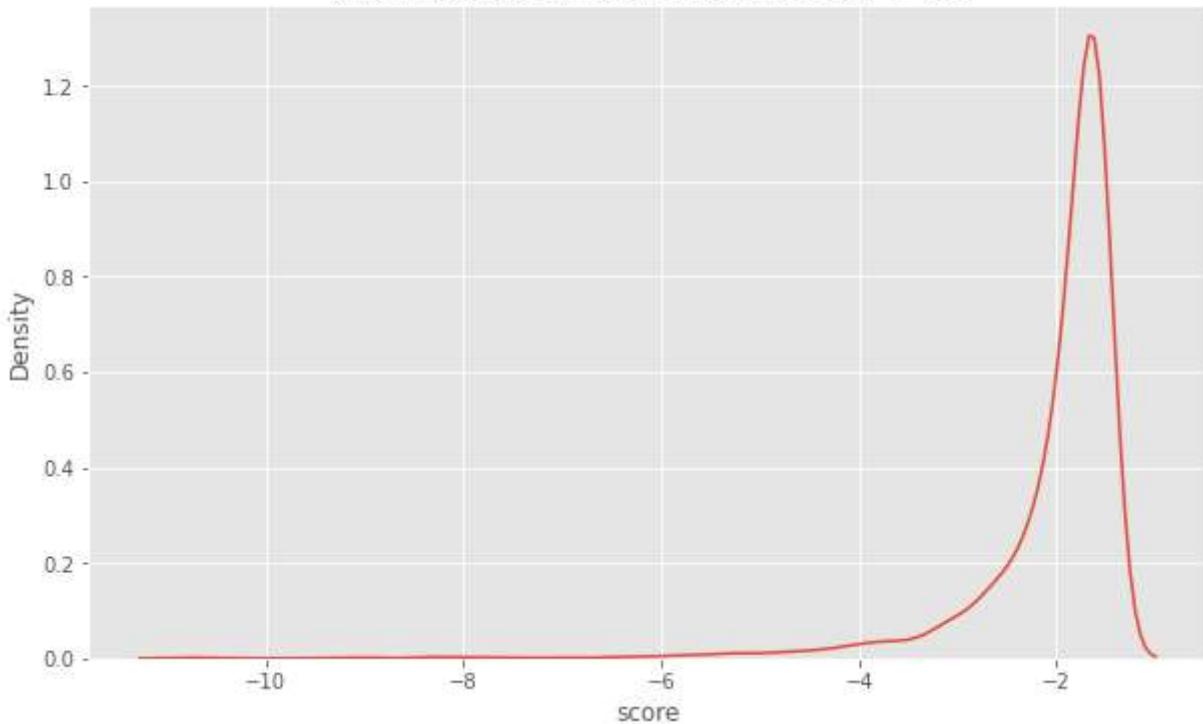
```
sns.kdeplot(Lof_score['score']).set_title('KDE Plot of Local Outlier Factor Score')  
print()
```



In [28]:

```
sns.kdeplot(Lof_score[Lof_score < -1.5]['score']).set_title('KDE Plot of Local Outlier Factor Score for Scores Below -1.5')  
print()
```

KDE Plot of Local Outlier Factor Score < -1.5



Based on the mean Local Outlier Factor Score of -1.1 and the KDE Plot of all observations, as well as observations with a score less than -1.5, it appears that -1.5 is a reasonable cutoff for multivariate outliers.

In [29]: `df_features['class'].value_counts()`

Out[29]:
GALAXY 59445
STAR 21593
QSO 18961
Name: class, dtype: int64

Drop Outliers

In [30]:
`Lof_mask = Lof_score['score'] < -1.5`
`outlier_indices = Lof_score[Lof_mask].index.tolist()`

In [32]:
`df_minus_outliers = df_features.drop(outlier_indices)`
`X_minus_outliers = df_minus_outliers.drop('class', axis=1)`
`y_minus_outliers = df_minus_outliers['class']`

In [33]: `y_minus_outliers.value_counts()`

Out[33]:
GALAXY 57824
STAR 20698
QSO 18464
Name: class, dtype: int64

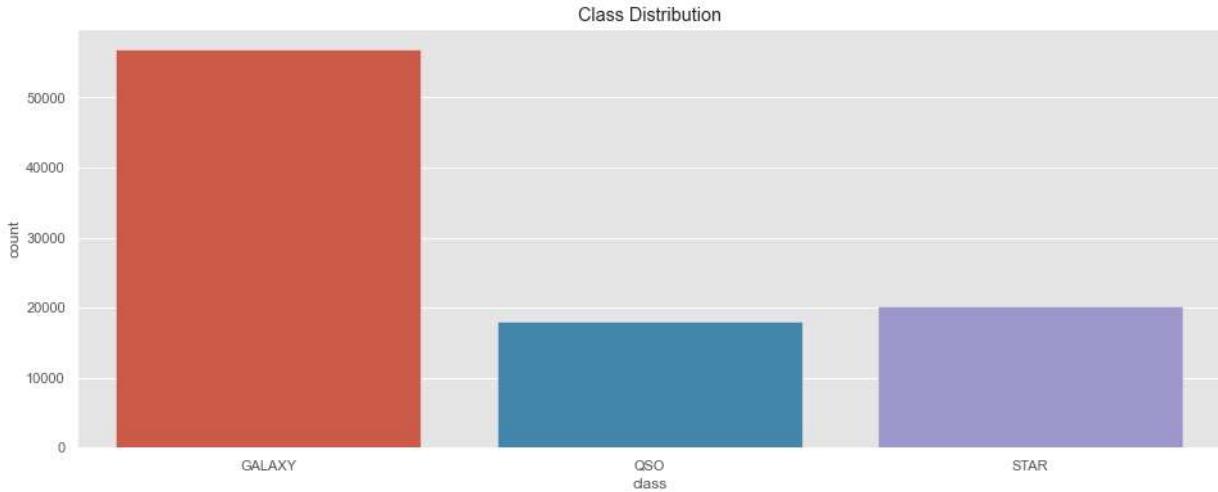
It appears that GALAXY accounted for a good portion of the outliers, which makes sense, as GALAXY is quite a broad range of observable objects--although so is STAR and this is reflected

by the proportion of STAR outlier vs QSO outliers.

Class Imbalance

In [137]:

```
sns.countplot(df_minus_outliers['class']).set_title('Class Distribution')
print()
```



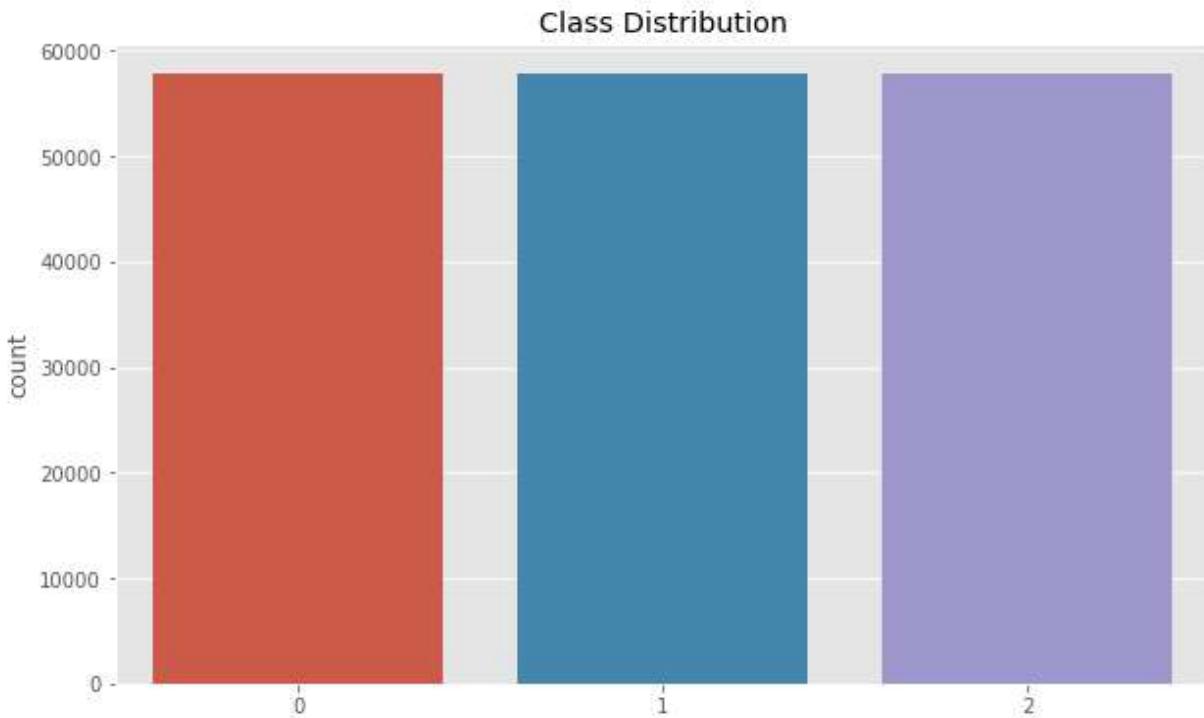
Mitigate Class Imbalance

In [37]:

```
sm = SMOTE(random_state=37)
x1, y1 = sm.fit_resample(x1, y1)
```

In [41]:

```
sns.countplot(x=y1).set_title('Class Distribution')
print()
```



Random Forest

Random Forest WITHOUT Aux Data

Create a function to run our random forest classifier

In [109]:

```
#Need this variable

X_df = pd.DataFrame(X).apply(scale)
X = scale(X)
X_svm = X
```

In [26]:

```
def rForestMethod(x_train, y_train, x_text, y_test):
    r_forest = RandomForestClassifier()
    r_forest_model = r_forest.fit(x_train,y_train)
    predicted = r_forest.predict(x_test)
    score = r_forest.score(x_test, y_test)
    rf_score_ = np.mean(score)

    print('Accuracy : %.3f' % (rf_score_))
    print('----- Classification Report -----')
    print(classification_report(y_test, predicted))
    return r_forest_model
```

Test / Train our data for our Random Forest classifier

In [37]:

```
#run on our scaled data WITHOUT Aux Data
x3 = X_df.Loc[:,['u','g','r','i','z','redshift']]
y3 = y_num
```

```
#Class imbalance
sm_3 = SMOTE(random_state=42)
x3, y3 = sm_3.fit_resample(x3, y3)

x3_train, x3_test, y3_train, y3_test = train_test_split(x3, y3, test_size = 0.30, rai
```

Run our Random Forest Model

In [38]:

```
rf_wout_aux = rForestMethod(x3_train, y3_train, x3_test, y3_test)
```

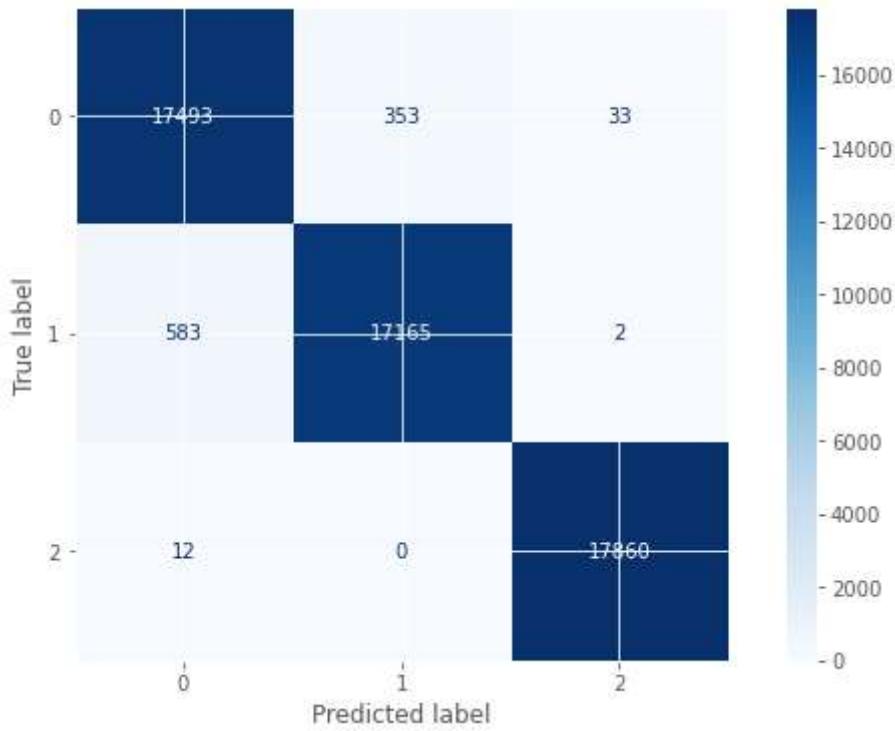
```
Accuracy : 0.982
-----
Classification Report -----
      precision    recall   f1-score   support
          0       0.97     0.98     0.97    17879
          1       0.98     0.97     0.97    17750
          2       1.00     1.00     1.00    17872
accuracy                           0.98    53501
macro avg       0.98     0.98     0.98    53501
weighted avg    0.98     0.98     0.98    53501
```

Confusion Matrix for Random Forest WITHOUT Aux Data

In [30]:

```
plot_confusion_matrix(rf_wout_aux, x3_test, y3_test, cmap=plt.cm.Blues)
plt.show()
```

```
/Users/roannarague/opt/anaconda3/lib/python3.8/site-packages/skLearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
warnings.warn(msg, category=FutureWarning)
```



SVM Classifier

SVM Classifier without Aux Data

Create our Test and Training datasets

In [20]:

```
def svmMethod(x_train, y_train, x_test, y_test):
    svm_clf = svm.SVC(kernel='rbf', C=1, random_state=0, probability=True)
    svm_model = svm_clf.fit(x_train, y_train)
    predicted = svm_clf.predict(x_test)
    score = svm_clf.score(x_test, y_test)
    svm_score_ = np.mean(score)

    print('Accuracy : %.3f' % (svm_score_))
    print('----- Classification Report -----')
    print(classification_report(y_test, predicted))
    return svm_model
```

In [23]:

```
#run on our scaled data WITHOUT Aux Data
x1 = X_df.Loc[:, ['u', 'g', 'r', 'i', 'z', 'redshift']]
y1 = y_num

#class imbalance
sm = SMOTE(random_state=42)
x1, y1 = sm.fit_resample(x1, y1)

x_train, x_test, y_train, y_test = train_test_split(x1, y1, test_size = 0.30, random_
```

Run our SVM and fit our test and training sets

In [24]:

```
svm_wout_aux = svmMethod(x_train, y_train, x_test, y_test)
```

```
Accuracy : 0.964
----- Classification Report -----
      precision    recall   f1-score   support
0       0.94     0.96     0.95    17879
1       0.98     0.94     0.96    17750
2       0.98     1.00     0.99    17872

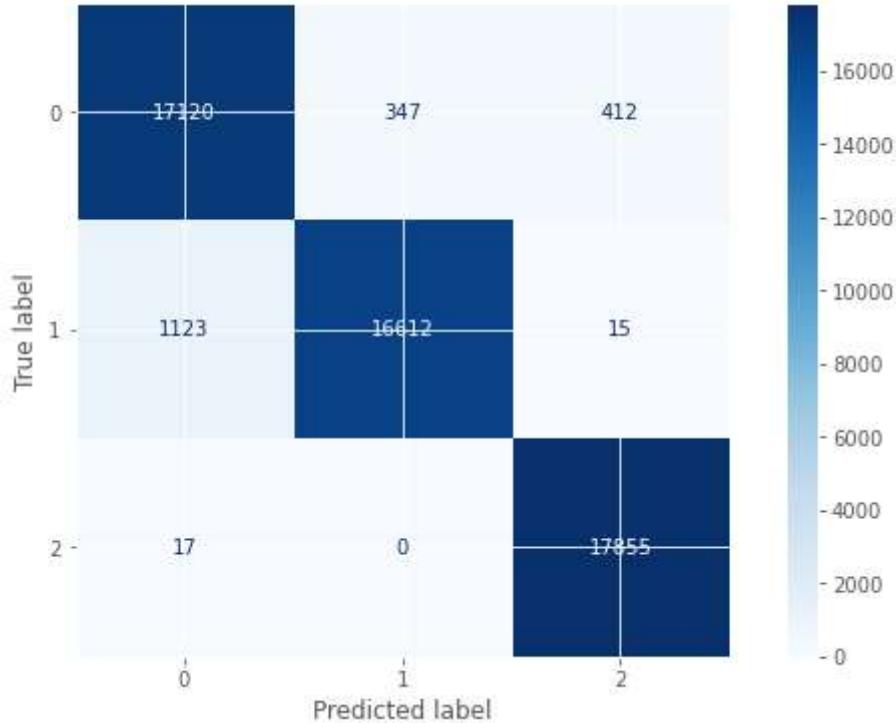
   accuracy          0.96    53501
macro avg       0.96     0.96     0.96    53501
weighted avg    0.96     0.96     0.96    53501
```

Confusion Matrix without Aux Data

In [25]:

```
plot_confusion_matrix(svm_wout_aux, x_test, y_test, cmap=plt.cm.Blues)
plt.show()
```

```
/Users/roannarague/opt/anaconda3/lib/python3.8/site-packages/skLearn/utils/deprecatio
n.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_
confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the cla
ss methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_es
timator.
warnings.warn(msg, category=FutureWarning)
```



SVM Classifier with Aux Data

Run our SVM on our scaled data with Aux Data

Test/Train split with AUX

In [50]:

```
#run on our scaled data WITH Aux Data
x2 = X_svm
y2 = y_num
```

```
#Class imbalance
sm2 = SMOTE(random_state=42)
x2, y2 = sm2.fit_resample(x2, y2)

x2_train, x2_test, y2_train, y2_test = train_test_split(x2, y2, test_size = 0.30, rai
```

Run our SVM and fit our test and training sets

In [51]:

```
svm_w_aux = svmMethod(x2_train, y2_train, x2_test, y2_test)
```

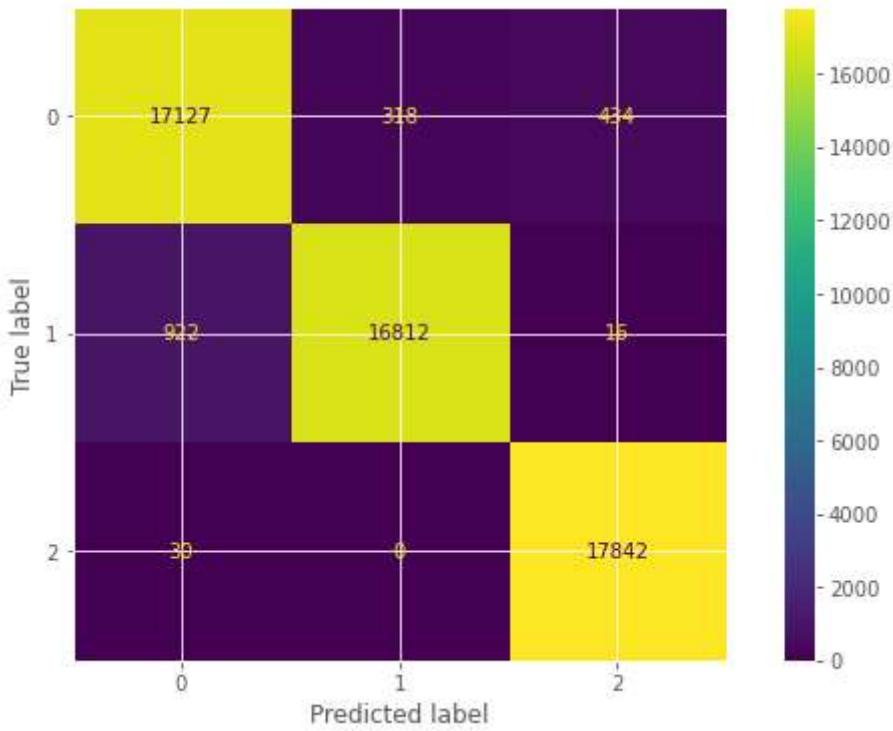
```
Accuracy : 0.968
----- Classification Report -----
      precision    recall   f1-score   support
          0       0.95     0.96     0.95    17879
          1       0.98     0.95     0.96    17750
          2       0.98     1.00     0.99    17872
   accuracy                           0.97    53501
  macro avg       0.97     0.97     0.97    53501
weighted avg       0.97     0.97     0.97    53501
```

Confusion Matrix with Aux Data

In [52]:

```
plot_confusion_matrix(svm_w_aux, x2_test, y2_test)
plt.show()
```

```
/Users/roannarague/opt/anaconda3/Lib/python3.8/site-packages/sklearn/utils/deprecatio
n.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_
confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the cla
ss methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_es
timator.
warnings.warn(msg, category=FutureWarning)
```



Preliminary Results

Exploratory clustering as an initial pass over our data, but we have not done qualifying clustering yet. We have not checked for outliers; we have not mitigated any class imbalance or dimensionality reduction to remove unimportant variables that might cause noise, and we have not cross validated yet.

In [3]:

```
import numpy as np

from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.metrics import silhouette_samples, silhouette_score

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
%matplotlib inline
plt.rcParams['figure.figsize'] = (10, 6)
plt.style.use('ggplot')
# Create color maps
from matplotlib.colors import ListedColormap
cmap = ListedColormap(['#e41a1c', "#984ea3", "#a65628", "#377eb8", "#fffff33", "#4daf4a", "#d9ead3", "#ffccbc", "#ff9800', "#ff5722", "#ff0000'])
```

How do you choose K? Ideas:

1. Visual comparison.
2. Looking for at which k the total intra-cluster distance tapers off.

3. Silhouette analysis

The first method is useful when the feature vectors are lower dimensional, but it's difficult to visualize data in higher dimensions. Let's consider the other two ideas.

Analyzing Change in Intra-Cluster Distance

We can run K-Means for different Ks and plot the intra-cluster distance.

In [285...]

```
import pandas as pd
```

In [390...]

```
df = pd.read_csv('./data_acquisition/Python/final_combined_QSO_1.csv')
df = df.Loc[:, 'plate':'fiber_ID']
df.head(1)
```

Out[390...]

	plate	z	redshift	class	spec_obj_ID	obj_ID	alpha	delta	u	g	r	i	run_ID	rerun_ID	cam_col	field_ID	MJD	fiber_ID	dtype
0	5812	18.79371	0.634794	GALAXY	6543777369295181824	1237660961327743273	135.689107	32.49											

Are there any NaN's

In [298...]

```
df.isna().sum()
```

Out[298...]

	plate	z	redshift	class	spec_obj_ID	obj_ID	alpha	delta	u	g	r	i	run_ID	rerun_ID	cam_col	field_ID	MJD	fiber_ID	dtype
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	int64

Check data types

In [389...]

```
df.dtypes
```

Out[389...]

	plate	z
	int64	float64

```
redshift      float64
alpha         float64
delta         float64
u             float64
g             float64
r             float64
i             float64
class_num     int32
class         object
dtype: object
```

Encode class

In [392...]

```
Le = LabelEncoder().fit(df['class'])
df['class_num'] = Le.transform(df['class'])
df.tail().loc[:,['class', 'class_num']]
```

Out[392...]

	class	class_num
71150	GALAXY	0
71151	QSO	1
71152	GALAXY	0
71153	GALAXY	0
71154	GALAXY	0

Look at other variables

In [139...]

```
df.dtypes
```

Out[139...]

```
plate          int64
z              float64
redshift       float64
class          object
spec_obj_ID    uint64
obj_ID         int64
alpha          float64
delta          float64
u              float64
g              float64
r              float64
i              float64
run_ID         int64
rerun_ID       int64
cam_col        int64
field_ID       int64
MJD            int64
fiber_ID       int64
class_num      int32
dtype: object
```

Modified Julian Date (MJD)

In [140...]:

```
df.MJD.min()
```

Out[140...]:

```
51608
```

In [141...]:

```
march_2000 = 51603 #https://core2.gsfc.nasa.gov/time/mjd2000.html
```

In [142...]:

```
(df.MJD.max() - df.MJD.min()) / 365.25
```

Out[142...]:

```
20.05201916495551
```

Roughly March 2000 through Mid-2020

Drop for now to simplifiy exploratory clustering

In [143...]:

```
df = df.drop(['cClass', 'spec_obj_ID', 'obj_ID', 'MJD'], axis=1)  
df.dtypes
```

Out[143...]:

plate	int64
z	float64
redshift	float64
alpha	float64
delta	float64
u	float64
g	float64
r	float64
i	float64
run_ID	int64
rerun_ID	int64
cam_col	int64
field_ID	int64
fiber_ID	int64
class_num	int32
dtype:	object

k-means

In [184...]:

```
# clustering for k = 1 to k = 10  
ks = range(1,11)  
scores = []  
  
fig, axs = plt.subplots(2,5, figsize=(20, 8))  
# fig, axs = plt.subplots(1,1)  
  
for k in ks:  
    model = KMeans(n_clusters=k)  
    y_pred = model.fit_predict(df.drop(['class_num'], axis=1).values)
```

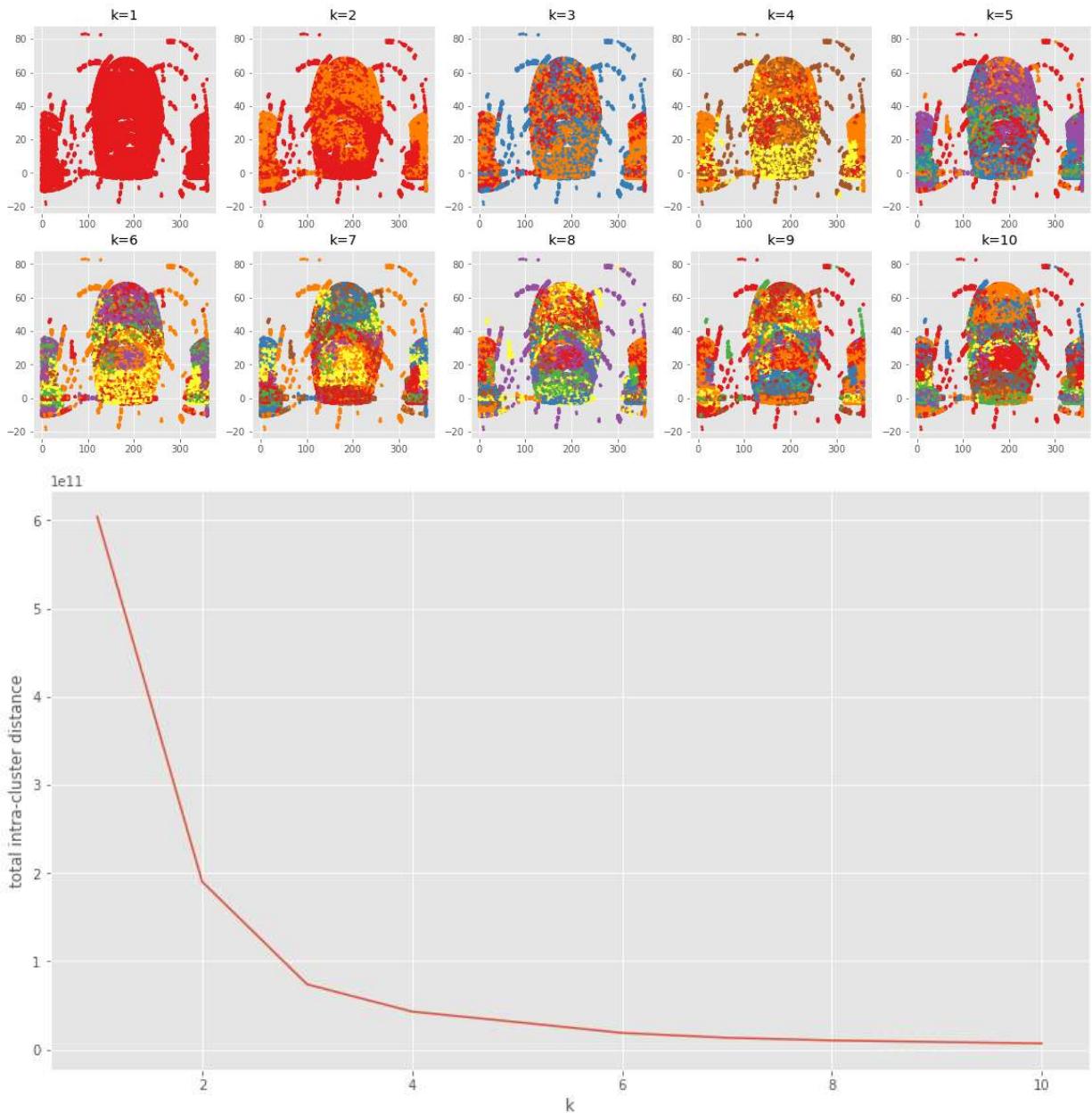
```

scores.append(-model.score(df.drop(['class_num'], axis=1)))

subPlot = axes[int((k-1)/5)][(k-1)%5]
subPlot.scatter(x='alpha', y='delta', c=y_pred, data=df, marker="o", cmap=cmap,
subPlot.set_title("k="+str(k))

fig = plt.figure(figsize=(14, 8))
plt.plot(ks, scores)
plt.yLabel('total intra-cluster distance')
plt.xLabel('k')
plt.show()

```



Base-Truth is $k=3$, elbow appears to be 3 or 4

Rerun with less columns

```
In [145...]: df.columns
```

```
Out[145...]: Index(['plate', 'z', 'redshift', 'alpha', 'delta', 'u', 'g', 'r', 'i',
       'run_ID', 'rerun_ID', 'cam_col', 'field_ID', 'fiber_ID', 'class_num'],
      dtype='object')
```

k-means

```
In [146...]:
```

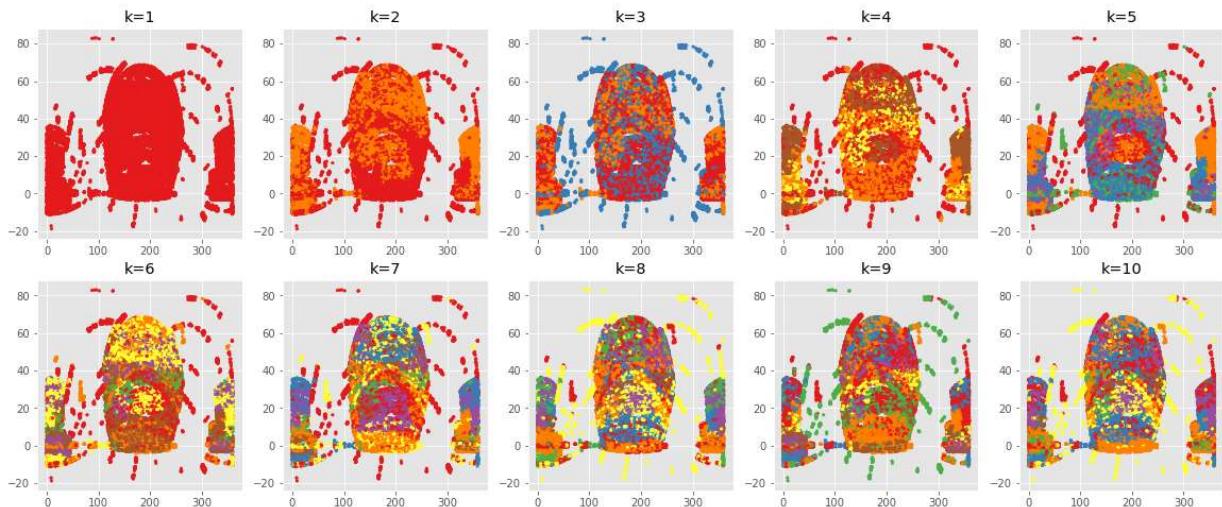
```
# clustering for k = 1 to k = 10
ks = range(1,11)
scores = []

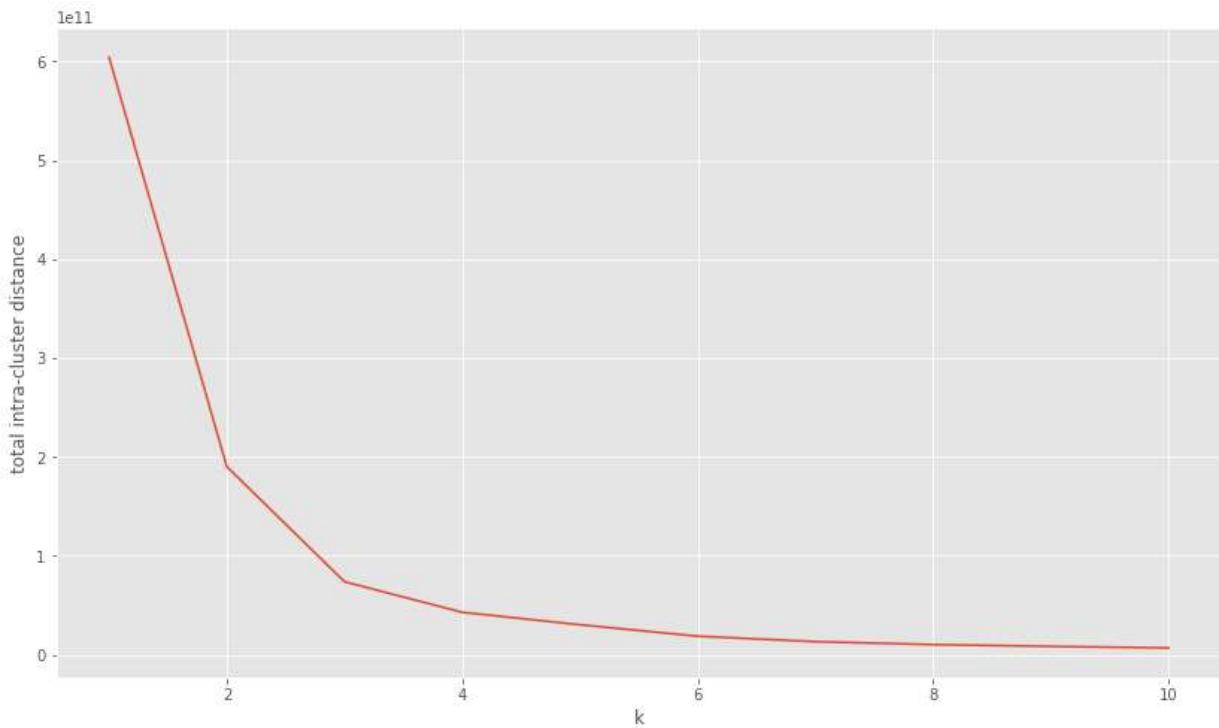
fig, axs = plt.subplots(2,5, figsize=(20, 8))
# fig, axs = plt.subplots(1,1)

for k in ks:
    model = KMeans(n_clusters=k)
    y_pred = model.fit_predict(df.drop(['run_ID', 'rerun_ID',
                                         'cam_col', 'field_ID',
                                         'fiber_ID', 'class_num'], axis=1).values)
    scores.append(-model.score(df.drop(['run_ID', 'rerun_ID',
                                         'cam_col', 'field_ID',
                                         'fiber_ID', 'class_num'], axis=1)))

    subplot = axs[int((k-1)/5)][(k-1)%5]
    subplot.scatter(x='alpha', y='delta', c=y_pred, data=df, marker="o", cmap=cmap,
                    subplot.set_title("k="+str(k))

fig = plt.figure(figsize=(14, 8))
plt.plot(ks, scores)
plt.ylabel('total intra-cluster distance')
plt.xlabel('k')
plt.show()
```





Elbow is more distinctly k=3 (Base-Truth)

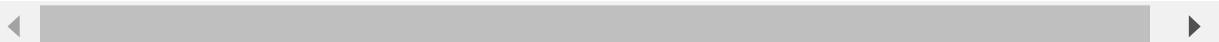
Removed variables resulted in ~ order of magnitude less intra-cluster distance at k=3 than before

In [147...]

```
df = df.drop(['run_ID', 'rerun_ID', 'cam_col', 'field_ID', 'fiber_ID'], axis=1)
df.head(1)
```

Out[147...]

	plate	z	redshift	alpha	delta	u	g	r	i	class_num
0	5812	18.79371	0.634794	135.689107	32.494632	23.87882	22.2753	20.39501	19.16573	(



Silhouette plot

In [202...]

```
d = df.drop('class_num', axis=1).values
range_n_clusters = [2, 3, 4, 5, 6, 7]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # Lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
```

```

# plots of individual clusters, to demarcate them clearly.
ax1.set_xlim([0, Len(d) + (n_clusters + 1) * 10])

# Initialize the clusterer with n_clusters value and a random generator
# seed of 10 for reproducibility.
clusterer = KMeans(n_clusters=n_clusters, random_state=10)
cluster_labels = clusterer.fit_predict(d)

# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the formed
# clusters
silhouette_avg = silhouette_score(d, cluster_labels)
print("For n_clusters =", n_clusters,
      "The silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(d, cluster_labels)

y_lower = 10
for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.nipy_spectral(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10 # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster Label")

# The vertical line for average silhouette coefficient of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([]) # Clear the yaxis labels / ticks
ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed
colors = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
ax2.scatter(d[:, 3], d[:, 4], marker='.', s=30, lw=0, alpha=0.7,
            c=colors, edgecolor='k')

# Labeling the clusters
centers = clusterer.cluster_centers_
# Draw white circles at cluster centers
ax2.scatter(centers[:, 3], centers[:, 4], marker='o',

```

```

c="white", alpha=1, s=200, edgecolor='k')

for i, c in enumerate(centers):
    ax2.scatter(c[3], c[4], marker='$_d$' % i, alpha=1,
                s=50, edgecolor='k')

ax2.set_title("The visualization of the clustered data.")
ax2.set_xlabel("Feature space for the 1st feature")
ax2.set_ylabel("Feature space for the 2nd feature")

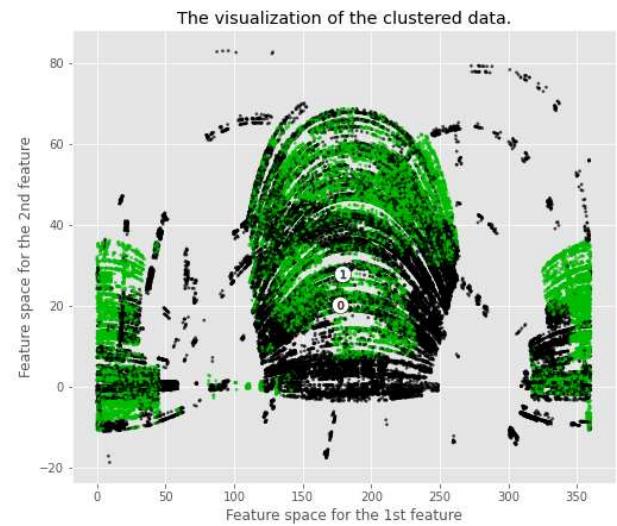
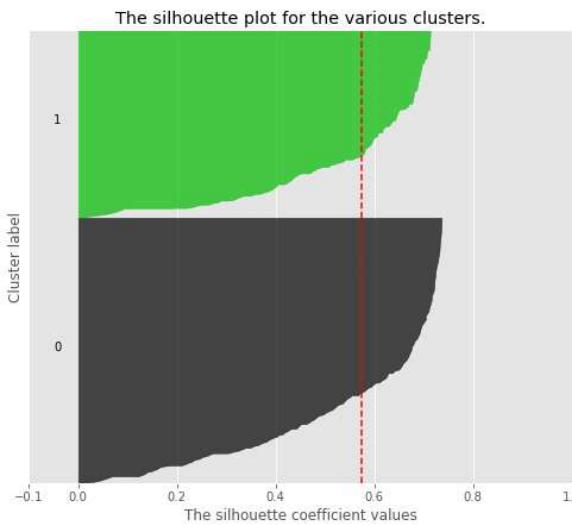
plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
             "with n_clusters = %d" % n_clusters),
             fontsize=14, fontweight='bold')

plt.show()

```

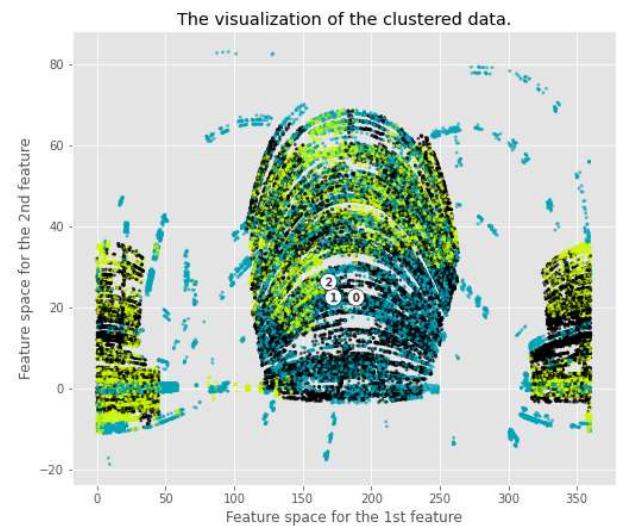
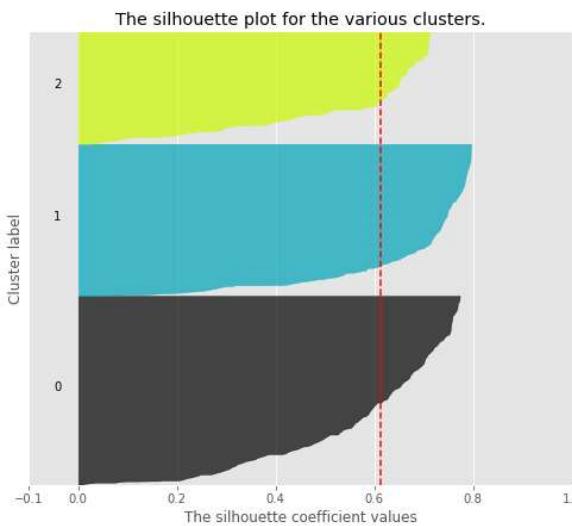
For n_clusters = 2 The silhouette_score is : 0.5738599509906098

Silhouette analysis for KMeans clustering on sample data with n_clusters = 2



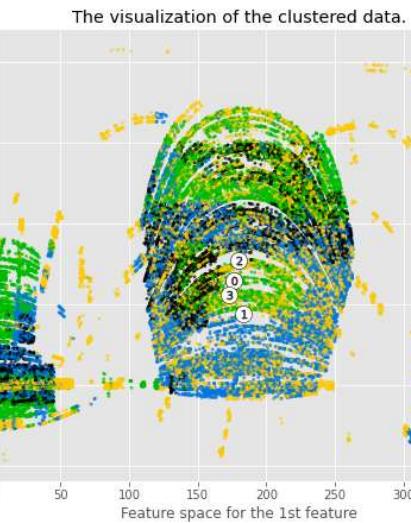
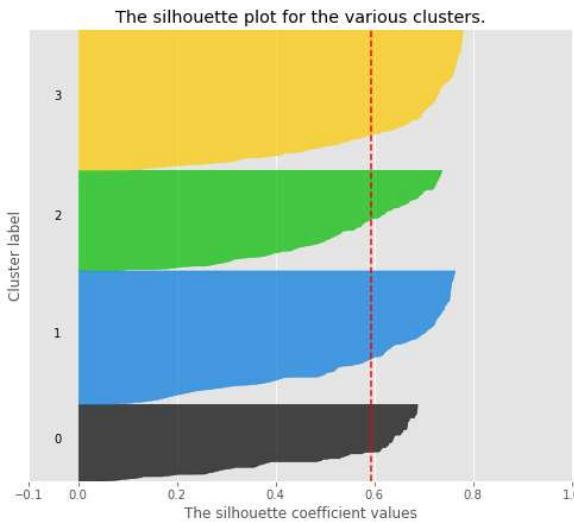
For n_clusters = 3 The silhouette_score is : 0.6119220136140294

Silhouette analysis for KMeans clustering on sample data with n_clusters = 3



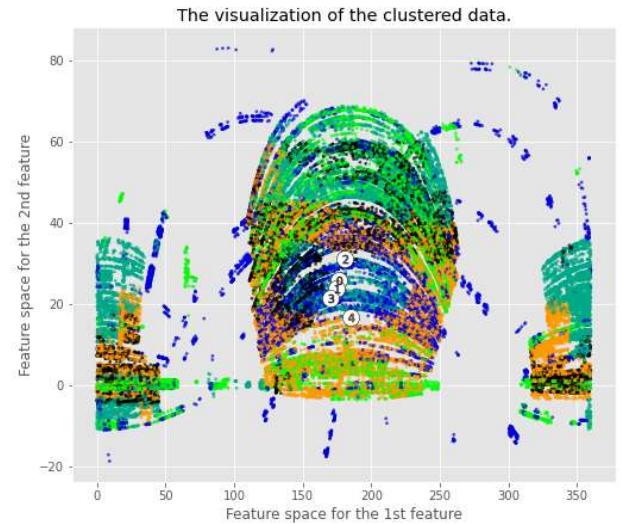
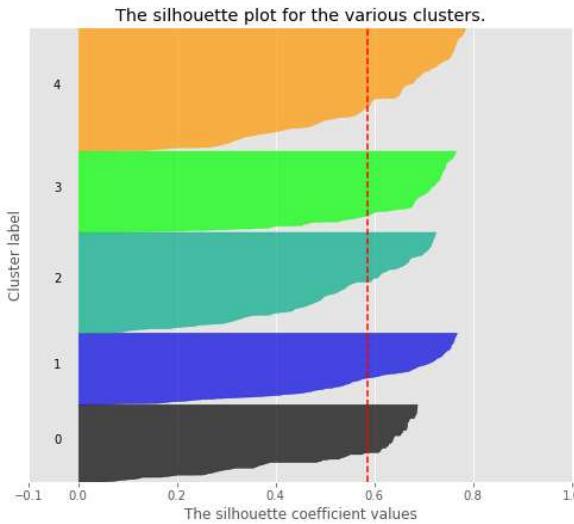
For n_clusters = 4 The silhouette_score is : 0.593575621783908

Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



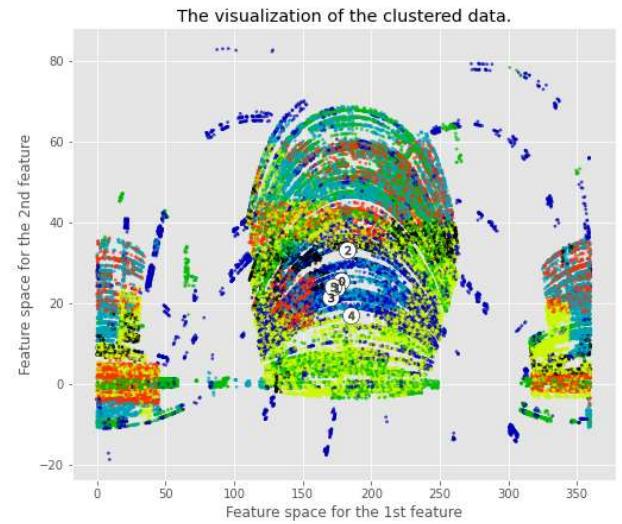
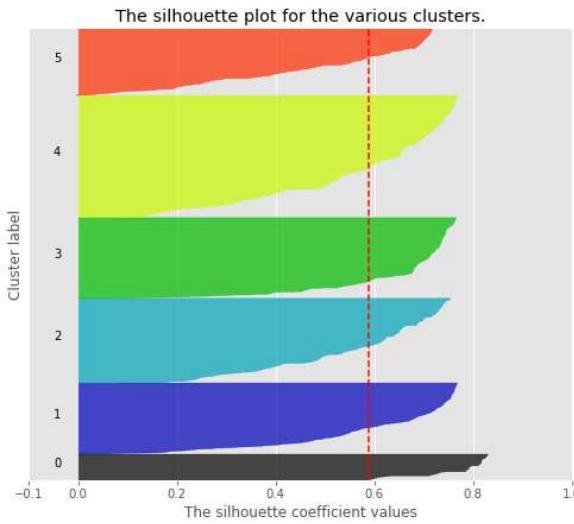
For n_clusters = 5 The silhouette_score is : 0.5857406457568644

Silhouette analysis for KMeans clustering on sample data with n_clusters = 5

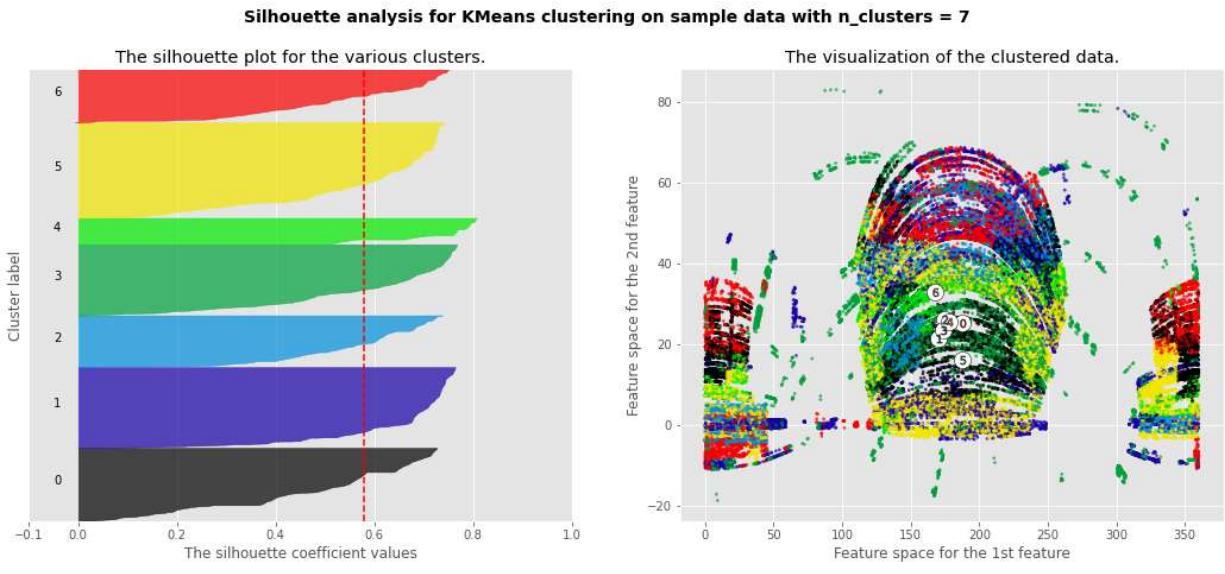


For n_clusters = 6 The silhouette_score is : 0.5890318221073857

Silhouette analysis for KMeans clustering on sample data with n_clusters = 6



For n_clusters = 7 The silhouette_score is : 0.5791386409248354



Peer Feedback

Our peer feedback answered the following questions:

Are the objectives interesting and suitable for a class project? -- **yes**

Is the scope of the project appropriate? If not, suggest improvements. -- **yes. It was mentioned that we should consider data classifications changing over time given that over time there have been higher resolution telescopes/cameras, better technology that could change the outcome classification of an astronomical object. If we were to create this model now how accurate would it be in the future?**

Is the split between optional and must-have features appropriate? Why? -- **yes**

Is the stakeholder analysis comprehensive? Are there specific ethical issues? -- **something to consider is why some data is classified as a quasar and then as a galaxy/star. As mentioned above, the classification rules change over time. Get a percentage of how many astronomical objects were later classified as quasar instead of a star/galaxy.**

Data Acquisition and Cleanup

Is data acquisition realistic? -- **with .fits files it was not realistic, with web-scraping we got closer with acquiring the data and finally with using SDSS's query/crossmatch/SQL tools we were able to realistically retrieve the data.**

Is the data acquisition trivial (e.g., downloading an existing dataset). Does the rest of the project compensate? -- **not it was not trivial, and the rest of the project does compensate.**

What is the plan for cleanup? Is cleanup that goes beyond what was taught in class necessary?

The cleanup plan involves combining CSVs, adjusting some of the classifications from galaxy/star to quasar if the "spec_obj_id" aligns with the quasar dataset. Removing duplicate header rows and validating we have balanced data.

Is there enough data to accomplish the stated objectives?

Yes. Instructor (Bei Wang) indicated that ~100,000 data points is a sufficient starting point

If needed, more data points can be acquired using the same methods we used to acquire our working dataset.

Are there other variables that are not being modeled that have a strong effect on the problem? -- at this moment in time, this is unknown. The hope is our model might be able to tell us which variables have a strong effect on the problem.

Analysis Methodology

Does the analysis methodology work for the data? E.g., for supervised learning - is the data labeled?

Yes, supervised learning with labeled data

Would other analysis methods be appropriate?

Yes, Bayesian Models could also be appropriate

--Instructor (Haoyu Chen) feedback was to optionally consider adding methods from Bayesian Models for Astrophysical Data by Hilbe, Souza, and Ishida Yes, t-SNE may be appropriate for Dimensionality Reduction/Visualization**

--Instructor (Bei Wang) feedback

"Visualization in Astrophysics: Developing New Methods, Discovering Our Universe, and Educating the Earth" - Bei Wang, et al

CNN could also be a potential option, as "Kim and Brunner [KB16] performed star-galaxy classification using CNNs"

Is the scale of the dataset sufficient for the analysis methodology?

Yes. Instructor (Bei Wang) indicated that ~100,000 data points is a sufficient starting point. More data points can be acquired as needed.

Completed Milestones

- Determining the proper method for retrieving our dataset

- Acquiring our dataset (retrieving 100,000 objects)
 - Validating our dataset had at least a portion of our three classification types: Galaxy, Star, Quasar
 - Getting a final, useable CSV file to perform and build our model with
 - Initial data analysis of the final dataset (clustering)
-
-

Upcoming Milestones

- Check for class imbalance and outliers
 - Perform the rest of the analysis methods on the data:
 - Classification (SVM, K-means)
 - Dimensionality Reduction (PCA/LDA)
-
-

Potential Modification to the Project

How do we check if our data is linear? So first we should do some kind check to verify whether our data is linear or nonlinear. If the data is nonlinear, we will have to switch our methods from LDA and PCA to t-SNE and UMAP. We originally planned to mitigate outliers before dimensionality reduction, we are now of the opinion it would be best to run dimensionality reduction with and without outliers and then compare the results of both. Along these same lines, we originally planned to center/scale the data before clustering, but now we are going to center/scale and then cluster and then not center/scale and compare the results.

Project Summary:

Overall, our project is on track, and we have followed our original milestones that we set at the beginning of the class. We initially ran into some roadblocks at the start when trying to acquire the appropriate data and acquiring in a way that made sense and was not so cumbersome. Once those roadblocks were surpassed, we are now able to view and understand our final dataset to perform our final data analysis and all the methods we listed to create a model to accurately predict an astronomical object based on the data we feed into it.

Assessment

Is our project on track: yes

Final Results Summary

As our final results, and in summary, our model was able to determine the correct classes of stellar objects by: After our initial steps of data acquisition that involved:

- *parsing through FITS files*
- *Web scraping*
- *An initial Web Query using the Cross Match Tools on the SDSS dr17 database*
- *Injecting the necessary Quasar and data*
- *Until finally running the final query using the Search Tools on the SDSS dr17 website to give us ~100,000 objects of data.*

We were then able to massage the data accordingly, checking for any outliers, and any class imbalance before performing the following methods on the final, cleaned data set:

- *PCA*
- *t-SNE*
- *Random Forest*
- *SVM*
- *and initially some K-Means clustering*

We determined that when comparing both models, one model containing auxiliary data and another model without that there were differences in the model's capability of determining the correct class for the stellar object.

Dimensionality Reduction

- *PCA: for our PCA dimensionality reduction method/model ->*
 - *The difference between data WITH our auxiliary resulted in a model with indiscriminate clusters, it was hard to tell and there was not cohesive structure forming within our PC1 and PC2 models. This implied that there was possibly some variables dominating the variance.*
 - *The difference between data WITHOUT our auxiliary resulted in a model with discriminant clusters, it was a lot easier to tell and there WAS a somewhat cohesive structure forming within our PC1 and PC2 models.*
- *t-SNE:*

- With t-SNE, we noticed the same behavior with some of the variables, for example --> plate & MJD are moderately correlated with u, g, r, i, z, and redshift. The same can be said for fiber_ID, to a lesser degree. In addition, alpha and delta represent the location on the celestial sphere (sky), which may have a relationship with the classes of astronomical objects present, but is not an observable characteristic of the object itself. Similarly, run_id is the identifier of the run, which may have implications on classes observed in that run, but is not directly an attribute of the object--and likewise for cam_col/field_ID.
- As we saw with the single outlier that shaped our correlation matrix, added variance that does not represent the underlying silhouette we are trying to capture can lead us astray--particularly when contained in a variable that has a significant correlation with a variable that does represent what we are trying to capture.

Let's examine what happens to our t-SNE dimensions when we remove these variables.

Classifiers / Clustering

- Random Forest:
 - WITHOUT the Auxiliary data, Random Forest had about a 98% accuracy score
 - WITH Auxiliary data we were unable to run
- SVM:
 - WITHOUT the Auxiliary data, SVM had about a 96.4% accuracy score
 - WITH the Auxiliary data, was about a 96.8%
- K-Means:
 - WITHOUT the Auxiliary data, K-means had about a 44% accuracy score
 - WITH the Auxiliary, K-means. had about a 42% accuracy score

FINAL NOTES In summary for the Classifiers and Clustering Random Forest did do a much better job at classifying the data with a 98% accuracy score compared to 96% on SVM WITHOUT Auxiliary data also compared with 44% by K-Means WITHOUT Auxiliary data.

As final, our model with the combination of proper data acquisition, data massaging/cleaning, dimensionality reduction using multiple methods and then clustering/classifying using multiple methods on the data to determine that certain data was causing some dominance in the variance and once we removed depending on which classifier/model we decided to use we had to about 98% accuracy in classifying our astronomical data.