



MACHINE LEARNING: DEVELOPING A SUPERVISED MULTI-CLASS CLASSIFICATION ALGORITHM



Course: *Computer Science*

Student Name: *Amrit Kooner*

Student ID: *20134667*

Words: 7.5K

(Without titles, text boxes, tables, figures, etc.)

Contents

1 - Report Introduction.....	3
1.1 - Dataset identification	4
1.1.1 - Dataset Attributes.....	4
1.1.2 – Where we found the dataset	7
1.1.3 – Where the data originated from	7
1.2 - Supervised learning task identification.....	8
1.3 - Team Identification	8
2 - Exploratory Data Analysis	9
2.1 - Question(s) identification	9
2.2 - Splitting the dataset.....	11
2.3 - Exploratory Data Analysis process and results	13
2.3.1 - Univariate Analysis.....	13
2.3.1.1 - Target Variable Analysis	13
2.3.1.2 - Individual Feature Analysis.....	15
2.3.1.3 - Individual Feature to Target Variable Analysis.....	25
2.3.1.4 – Redshift to Target Variable Analysis.....	34
2.3.2 - Multivariate Analysis.....	35
2.3.2.1 - Checking for missing values	35
2.3.2.2 - Feature to feature analysis.....	37
2.4 - EDA conclusions	38
3 - Experimental Design.....	40
3.1 - Identification of your chosen supervised learning algorithm(s).....	40
3.2 - Identification of appropriate evaluation techniques.....	41
3.3 Data cleaning and Pre-processing transformations.....	42
3.4 - Limitations and Options.....	60
4 - Predictive Modelling / Model Development	61
4.1 - The predictive modelling processes	61
4.1.1 – Modeling With Training Set (SEEN DATA)	61
4.1.2 – Modeling With Testing Set (UNSEEN DATA)	62
4.2 - Evaluation results on “seen” data.....	63
4.2 - Evaluation results on “unseen” data	66
4.3 – Comparing results on “unseen” and “seen” data	69
5 - Evaluation and further modelling improvements	70

5.1 - Initial evaluation comparison	70
5.2 - Further modelling improvements attempted.....	73
5.2.1 – Standardization (Data Scaling)	73
5.2.2 – Dimensionality Reduction (Data Transformation)	75
5.2.3 – Balancing Dataset (Data Transformation)	76
5.3 - Final Evaluation results.....	78
5.3.1 – Evaluation results on “seen” data.....	78
5.3.2 – Evaluation results on “unseen” data.....	80
5.3.3 – Comparing results on “unseen” and “seen” data.	82
6 - Conclusion	83
6.1 - Summary of results	83
6.2 - Suggested further improvements to the model development process	84
6.2.1 – What went well with the model development process.....	84
6.2.2 – How the model could be improved	84
6.3 - Reflection on Research Team	85
6.4 - Reflection on Individual Learning	85
7 – References	86
8 – Bibliography.....	88

1 - Report Introduction

This report is about evaluating and developing a supervised machine learning algorithm to form a prediction based on a chosen dataset model. I will be working within a team of four where we each research and develop a separate chosen algorithm model appropriate to our dataset, which we then later test and compare with one another to evaluate its differences in accuracy and other metrics for further improvements. I will be using Python to develop my algorithm with imported libraries such as Scikit-learn for machine learning for algorithm modeling, Pandas for dataset manipulation, Seaborn and Matplotlib for data visualization with graphs, and more.

1.1 - Dataset identification

Our chosen dataset contains 17 features and 100,000 observations that represent real-world data for the concept of stellar classification where a spectral class is determined based on its characteristics. So each observation within the dataset corresponds to a type of spectral class which is determined based on its features. Out of the 17 features, there is 1 class feature that identifies as one of three possible spectral objects an observation could classify as, being star, galaxy, or quasar. The data is continuous as most of the features are measurements. They also author stated that the dataset aims to classify stars, galaxies, and quasars based on their spectral characteristics, and the classification scheme of galaxies, quasars, and stars is one of the most fundamental concepts and findings in astronomy, Fedesoriano. (January 2022). *Stellar Classification Dataset - SDSS17*.

1.1.1 - Dataset Attributes

The following table shows every feature within our database, and they represent.

No.	Attribute	Description
1.	<i>obj_id</i>	Object Identifier, is the unique value that identifies the object in the image catalog used by the CAS. It functions as a key to retrieve information about the object and its associated metadata, such as its position, colour, size, or other characteristics.
2.	<i>alpha</i>	Right Ascension angle (at J2000 epoch) that is symbolized as Alpha. It is used to classify stellar objects according to the position of their spectral lines on the object's spectrum. It is based on the principle that the properties of a stellar object's atmosphere can be inferred from the strength and width of its spectral lines. The J2000 epoch is a specific point in time that was used as a reference for the positions of celestial objects such as stars, quasars, and galaxies.
3.	<i>delta</i>	Declination angle (at J2000 epoch) that is symbolized as Delta). It is used to classify stellar objects according to the position of an object in the sky in the context of astronomy. It is the angle between a celestial object and the equator, measured either north or south of the celestial equator. The J2000 epoch is a specific point in time that was used as a reference for the positions of celestial objects such as stars, quasars, and galaxies.
4.	<i>u</i>	Ultraviolet filter in the photometric system. Used for measuring the intensity and colour of light emitted by stars, galaxies, quasars, and other stellar objects. The ultraviolet filter allows the measurement of light in the ultraviolet range of the electromagnetic spectrum.
5.	<i>g</i>	Green filter in the photometric system. It is used to measure the intensity of light in the green range of the electromagnetic spectrum, which are emitted from stellar objects such as stars and other stellar objects. It measured temperature, luminosity, chemical composition and evolutionary states.
6.	<i>r</i>	Red filter in the photometric system. It is used to measure the intensity of light in the red range of the electromagnetic spectrum, which are emitted from stellar objects such as stars and other stellar objects. It measured temperature, luminosity, chemical composition and evolutionary states.

7.	<i>i</i>	Near Infrared filter in the photometric system. It is used to measure the intensity and colour of light emitted by stars, galaxies, quasars and other stellar objects, specifically it measures the light in the near-infrared range of the electromagnetic spectrum. It provides essential information about the characteristics of stars, galaxies, quasars, such as their chemical composition, brightness and temperature.
8.	<i>z</i>	Infrared filter in the photometric system. Infrared filter in the photometric system. It is used to measure the intensity and colour of light emitted by stars, galaxies, quasars and other stellar objects, specifically it measures the light in the infrared range of the electromagnetic spectrum. It provides essential information about the characteristics of stars, galaxies, quasars, such as their chemical composition, brightness and temperature.
9.	<i>run_id</i>	Run Number used to identify the specific scan. It is used to classify a particular sky scan or set of sky scans. Additionally used to store, retrieve, and compare the data from that observation to other observations and datasets. Keeps track of and catalogues the data from different surveys which is helpful when comparing the results of different studies.
10.	<i>rerun_ID</i>	Rerun Number to specify how the image was processed. In stellar classification, it is a unique identifier for different versions of processed data. It is useful for comparing and verifying results because it is used to track specific processing methods such as algorithms, software, and updated techniques.
11.	<i>cam_col</i>	Camera column to identify the scanline within the run. <i>cam_col</i> refers to the camera column. It is a unique identifier used to identify a specific scanline within a run of observations. It can be used to match the data collected from a specific scanline with stellar objects. Its essential for better understanding the characteristics and distribution of objects in the field and for comparing the data from different scanlines to study various stellar objects.
12.	<i>field_id</i>	Field number to identify each field. It is used to identify and locate each field in the process of stellar classification. A field is a particular area of the sky that has undertaken survey or observation. This information is important for understanding the properties and distribution of stellar objects in the field, and for comparing these stellar objects.
13.	<i>spec_obj_id</i>	A unique ID used for optical spectroscopic objects (this means that 2 different observations with the same <i>spec_obj_ID</i> must share the output class). Its is useful for tracking the properties of stellar objects over time, as it allows them to be compared at different times and locations to better understand how the object has changed or evolved overtime.
14.	<i>class</i>	Object class (galaxy, star, or quasar object). These are types of stellar objects can that be identified, classified and further measured using these other attributes. We can measure aspects such as brightness, distance, current position, temperature, chemical composition, rotation, mass, radius and more.
15.	<i>redshift</i>	Redshift value based on the increase in wavelength. The redshift value is based on the increase in wavelength of light from an object as it moves away

		from an observer. The redshift value can be used to calculate the distance and velocity of the object and thus how far it is from the observer.
16.	<i>plate</i>	Identifies each plate in SDSS, it is similar to the <code>obj_id</code> . The plate number is used to identify and monitor each plate during the survey, as well as to connect the plate to other data sources like the SDSS database, which has details about the objects on the plate, including their locations, magnitudes, and other characteristics.
17.	<i>MJD</i>	Modified Julian Date, used to indicate when a given piece of SDSS data was taken. It is used to specify the day and time an image or spectrum was captured to better understanding the circumstances of data collection, including the weather, air conditions, and other elements that may have an impact on the data's accuracy and quality.
18.	<i>fiber_id</i>	Identifies the fiber that pointed the light at the focal plane in each observation. Used to gather light from celestial objects and direct it to a spectroscope for analysis. Each fibre is given a special fibre ID that can be used to identify the fibre and the observation it was used for. The <code>fibre_ID</code> is important for following analysing observations and comparing to others.

1.1.2 – Where we found the dataset

Dataset: [Stellar Classification Dataset - SDSS17](#)

We found this dataset on Kaggle while researching datasets for classification, *Fedesoriano. (January 2022). Stellar Classification Dataset - SDSS17*. We decided to use this particular one as we all found the topic of Astronomy interesting, and we immediately had a clear prediction for our algorithms set in mind. We were first going to use another similar dataset which was also focused on Astrology and stellar classification with the spectral class being star types. However, it only had 240 observations and our teacher suggested that we should find and use another one with a minimum of at least 1,000 observations to prevent overfitting and underfitting problems when training our models. Our current dataset has 100,000 observations.

Our solution to this problem will provide an impact as it will aid in the research of astronomy and stellar classification which is a heavily researched topic by organizations such as NASA and European Space Agency (ESA). Stellar classification research is important as it helps us to further understand space, by identifying and classifying stellar objects such as stars, galaxies, quasars, planets, moons, and more for analysis to see how they are formed and operate based on their many characteristics.

1.1.3 – Where the data originated from

Original Source: [SDSS Data Release 17](#)

The author of the dataset has stated the data originated from the Sloan Digital Sky Survey (SDSS) website, which is a large survey involved in the field of astrology that aims to map out the universe so its properties can be further studied such as stars, planets, moons, and more. The data was collected from a collaboration of multiple organizations from various countries such as the UK, USA, and Japan. These organizations consist of multiple space agencies such as The Fermi National Accelerator Laboratory (FNAL) and The Institute for Advanced Study (IAS), as well as various Universities around the world. The data was collected for the continuance of research in astronomy to better understand everything in the universe beyond Earth's atmosphere. Their findings are public that can be accessed from their official website through online databases. Release date 17 refers to data collected from 2014 to 2021, so the dataset is contained with data found within this time frame, "Data Release 17. (2021). SDSS (Sloan Digital Sky Survey)"

1.2 - Supervised learning task identification

The predictive problem our team chose to solve centered on our dataset is to predict the spectral class of an observation based on its features. In classification problems, the goal is to predict a categorical label. So, our problem is recognized as a classification type problem as the predicted output will be one of three possible spectral classes, being either a star, galaxy, or quasar object. We can tell it's not a regression-type problem as in regression, the goal is to only predict a continuous numeric value. Our problem is also recognized as a multiclass-classification problem as the predicted output would be one of three possible spectral objects. Since the problem was identified as a classification type problem, classification algorithms will be used to solve it, which are a type of supervised algorithms as the models have to be trained with input data which is why we are using a dataset, unlike unsupervised type algorithms that don't need to be trained with input data of any kind.

The target variable that will establish the ground truth of the predictive problem will be the spectral *class* feature within the dataset as it will be the predicted output, with the features being fed into the model so it can be trained to form the prediction. The target variable can also be described as the label.

To summarise, the research team aims to build 4 different classification models around our dataset to solve a supervised multiclass-classification predictive problem, which predicts the target variable of an observation. The target variable will be the spectral class within our dataset which will also establish the ground truth. There are 3 spectral classes the predicted output may result in, being either a star, galaxy, or quasar object.

1.3 - Team Identification

The following table details the members of my research team with their corresponding classification algorithms that they will further research, develop and evaluate.

Forename	Surname	Student ID	Model(s) developed
Amrit	Kooner	20134667	Logistic Regression
Reece	Lewis	19116884	K-Nearest Neighbours (KNN)
Manraj	Nandhra	20129117	Naïve Bayes
Mohammed	Khizer	20121143	Support Vector Machine (SVM)

2 - Exploratory Data Analysis

2.1 - Question(s) identification

It is essential to explore your dataset throughout the EDA process before its development and analyse it in terms of Univariate and Multivariate analysis using appropriate graphs to identify patterns and problems. This process can help us better understand our dataset's structure and identify key characteristics, such as common data patterns within clusters or groups, and potential issues, such as missing values or outliers that should be fixed before the model is developed. Otherwise model will not predict well as it's trained with broken data. Univariate is the analysis of a single variable, like to see if the variable has no correlation with the target class. While Multivariate is the analysis of more than one variable, like to see if variables are correlated with each other. Depending on the discoveries, they will be further dealt with in the data cleaning and pre-processing phase.

The following table outlines the EDA questions my research team planned to explore within our dataset. It states the question with its relevance to our study, my initial assumptions, and whether the question will be answered during or concluding the EDA process.

No.	Question	Relevance to Study	Initial Assumption	When will it be answered?
1.	Are there any missing numbers within the dataset?	We want to ensure our dataset has no missing values as the model cannot be efficiently trained or tested with a dataset with important missing data. This will affect the overall performance of the model.	It's a large dataset, so I assume it will have a few missing values. However, the data did originate from a credible and professional source.	This question will be answered during the EDA process.
2.	Is the dataset balanced, is there any stellar object that contains more or fewer observations another the others?	We want to ensure our dataset is balanced so the model is not biased towards a class with more data than the other classes. This will affect the overall performance of the model, as one class is greatly more accurate than the others.	It's a large dataset, so I assume the dataset is not balanced.	This question will be answered during the EDA process.
3.	Do any of the features contain outliers if so how would they affect the model?	We want to ensure our dataset has no detrimental outliers are occurrences of unusual data such as extremely high or low values that can affect the overall performance of the model as the model is being tested and trained with unusual data that should not be in the dataset. But not all outliers should be removed, some contain important information.	It's a large dataset, so I assume it will have many outliers.	This question will be answered during the EDA process.

4.	Do any of the features have no correlation to the target variable 'class', if so should they be dropped?	We want to ensure there are no features that have zero correlation to the target variable as it is harder for the model to differentiate and correctly predict each class because the data for each observation within the feature is too similar or the same. This will affect the overall performance of the model.	There are 17 input features which is a lot, so I assume there some are features that have no correlation to the target class.	This question will be answered during the EDA process.
5.	Are any of the features too strongly correlated with each other causing multicollinearity problems, if so should they be dropped?	We want to ensure our dataset doesn't have any highly correlated features that affect many other features, as this can cause overfitting and multicollinearity issues as they make it difficult for the model to interpret the data of individual features as they rely on each other. This will affect the overall performance of the model.	There are 17 input features which is a lot, so I assume there some are too strongly correlated with each other causing multicollinearity problems.	This question will be answered during the EDA process.
6.	How is redshift affected by the distance of celestial bodies?	This will help us better understand the relationship between redshift with the target variable being stellar objects (star, galaxy, quasar) within our dataset and learn general real-world terms.	Not sure.	This question will be answered during the EDA process.
7	Are the majority of observations in the high-end or low-end of redshift?	This will help us better understand the relationship between redshift with the target variable being stellar objects (star, galaxy, quasar) within our dataset and learn general real-world terms.	Not sure.	This question will be answered during the EDA process.

2.2 - Splitting the dataset

Data leakage is the result of a model being trained or tested with data it has already seen. Data leakage can occur for many reasons such as the dataset not having enough data to train the model so a sufficient split cannot be made, using data from the future to train the model, or a dataset not properly split, so each dataset does not contain unique data. So data leakage can be prevented by having enough data, don't train the model with future data, and perform sufficient dataset splits to ensure they have unique data. Data leakage can cause overfitting problems, where the model performs well on the SEEN training set but poorly on new the UNSEEN testing set. SEEN refers to data the training set has seen, while USEEN refers to data the training set has not seen being the data in the testing set. This happens since the model cannot generalize new UNSEEN data as it was trained with data with initial UNSEEN data from the testing set. The model should form predictions based on the testing data by first learning from the training data.

We need to split the dataset to form a separate SEEN training set for training the model and an UNSEEN testing set for testing the model. This prevents data leakage as it ensures the two datasets do not have the same data. The model cannot be tested with data that is also used to train the model, and the model cannot be trained with data that is also in the testing set, otherwise, it will return biased results as it will form predictions based on features it has already seen, and cause overfitting.

```
# separated the features from the dataset so i can preform a train/test split
X = df.drop(['class'], axis=1) # FEATURES (NO CLASS FEATURE)
Y = df['class'] # CLASS LABEL/TARGET VARIABLE (CLASS FEATURE ONLY)

# i will split the to preform EDA on the training set
# 80/20 split between training and testing split
# 42 random state
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 42)
```

Figure 1 - Splitting dataset

To split our dataset I used the `train_test_split` function that the scikit-learn library provides. Only the training data will be analysed throughout the EDA process since it's used to train the model. The dataset was split before the EDA process to further minimize data leakage, which ensures the information in the testing set is not used to inform the analysis of the training set.

The proportion we split our dataset was 20% for testing and 80% for training. The larger majority is towards training as the model requires plenty for the learning process, so it can return more accurate predictions. The random state is 42 which is the amount of possible random splits for each set when the model is running, otherwise, it will be random each time, this is so we can get more consistent predictions. Using a random state will also prevent potential data leakage because if there was no random state, so the split was chosen randomly each time, some of the training data could end up in the test set and vice versa, which would cause overfitting. It is set to 42 as we have 100,000 observations. To further prevent data leakage and overfitting our team chose a dataset with 100,000 observations that were advised by our Lecturer, so a sufficient split can be made and to ensure enough data was left over to train the model after the data cleaning and pre-processing phase.

```
1 # shows the dataset size for each train/test split
2
3 # training set has 80,000 observations for features and class label
4 # testing set has 20,000 observations for features and class label
5 print(f"X_train -\nShape: {X_train.shape}\n\nX_test -\nShape: {X_test.shape}\n")
6 print(f"Y_train -\nShape: {Y_train.shape}\n\nY_test -\nShape: {Y_test.shape}\n")

X_train -
Shape: (80000, 17)

X_test -
Shape: (20000, 17)

Y_train -
Shape: (80000,)

Y_test -
Shape: (20000,)
```

Figure 2 - Training set and testing set sizes after split

The new shape of the training and testing set after splitting the dataset. The training set has 80,000 observations while the testing set has 20,000 observations. Both retain 17 features not including that class target variable which is in Y_train and Y_test. X_train and X_test contains only input features.

```
# merged the training features and class label to preform EDA on training set
df_train = pd.merge(X_train, Y_train, left_index=True, right_index=True)
```

Figure 3 - Merge training set to begin EDA process with it

The train set was merged back together for the EDA process as it will be easier to use than referring back to X_train and Y_train.

2.3 - Exploratory Data Analysis process and results

2.3.1 - Univariate Analysis

Univariate analysis was performed to analyse data within single variables. Questions 2, 3, 4, 6, and 7 were explored here.

2.3.1.1 - Target Variable Analysis

- **EDA question (2): Is the dataset balanced, is there any stellar object that contains more or fewer observations another the others?**

For the univariate analysis of the target variable, pie charts, and countplots was used to view the data for each class to determine if the dataset is balanced or not. This was one of our EDA questions (Question 2) we set out to explore, as if the model was trained with unbalanced data it will be more biased toward the class with more data. Affecting performance and overall accuracy as the majority class will be way more accurate than the minority class. Each class should have around the same quality of data to prevent these issues.

Pie charts and countplots were specifically used for this analysis as they work well at visualizing the data for each class to give a percentage with a pie chart and the total with a countplot, so they can be easily compared to check if it's balanced or not. If the dataset is indeed unbalanced, it will be made balanced in the data cleaning and pre-processing phase.

```
▼ Target Univariate Analysis
[249] 1  """
2  via the bar and pie graphs, we can see that the galaxy class has more than
3  double the observations than the star and quasar class. this means the dataset is unbalanced,
4  so it will be balanced in the data processing and cleaning phase using techniques like oversampling
5  and undersampling.
6  """
7  # this was one of our EDA questions.
8
9
10 # PIE GRAPH SHOWING DISTANCING OF THE STELLAR CLASSES
11 df_train['class'].value_counts().plot.pie(autopct='%0.2f')
```

Figure 5 - Pie chart code

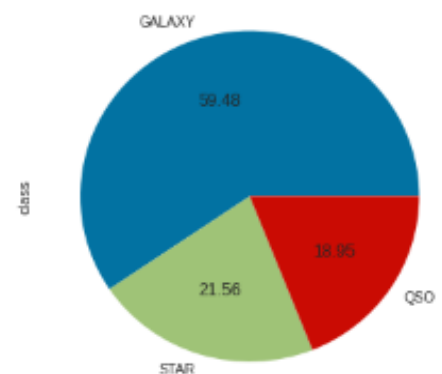


Figure 5 - Pie chart to check if dataset is balanced or not

Can see in figure 5 the pie chart shows that the data is indeed not balanced. The GALAXY class has far more data than the STAR and QSO class with 59.48%. STAR has 21.56% of the data while QSO has the rest being 18.95%.

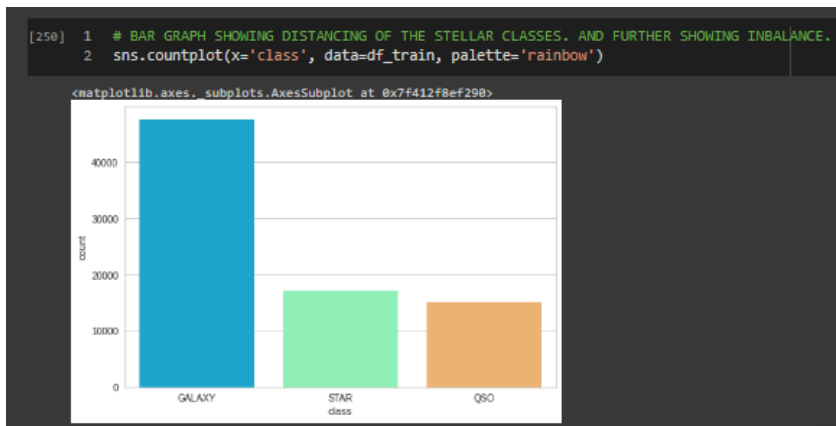


Figure 7 Countplot to see further see if the dataset is balanced or not with code

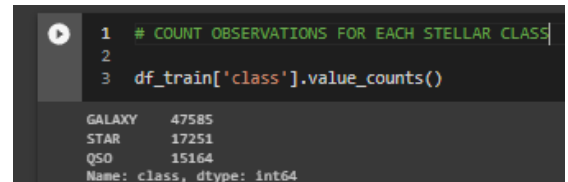


Figure 7 - Code to see exact number of observations for each class

From analysing the count plot it shows the GALAXY class has more than double the observations than the STAR and QSO class, with the biggest to smallest being GALAXY (47585) > STAR (17251) > QSO (15164).

It actually makes sense that the dataset is unbalanced as the data is a collection random stellar objects found throughout 2014 – 2021. Figure (4) and figure (7) shows that more galaxies were found than stars and quasars within this timeframe, and more galaxies were found than stars and quasars combined.

2.3.1.2 - Individual Feature Analysis

- **EDA Question (3): Do any of the features contain outliers if so how would they affect the model?**

For the univariate analysis of the rest of the individual features, boxplots were used to view the distribution of data to identify any detrimental outliers. This was one of our EDA questions (Question 3) we set out to explore, as occurrences of unusual data such as extremely high or low values that can affect the overall performance of the model, as the model is being tested and trained with unusual data that is different from the majority of data within that feature. Outliers could be the result of errors in a real-world occurrence that is rare or unusual. Outliers should be removed to prevent these issues, but not all outliers should be removed as some contain important information but look like an outlier.

Boxplots were specifically used for this analysis as they work well at visualizing the distributed data of the feature, so outliers can be easily spotted. Boxplots identifies outliers as it uses the interquartile range (IQR) of the middle 50% of the data, and any data points that fall outside of the IQR are considered outliers. If the dataset does contain detrimental outliers, they will be removed in the data cleaning and pre-processing phase.

- `obj_ID`

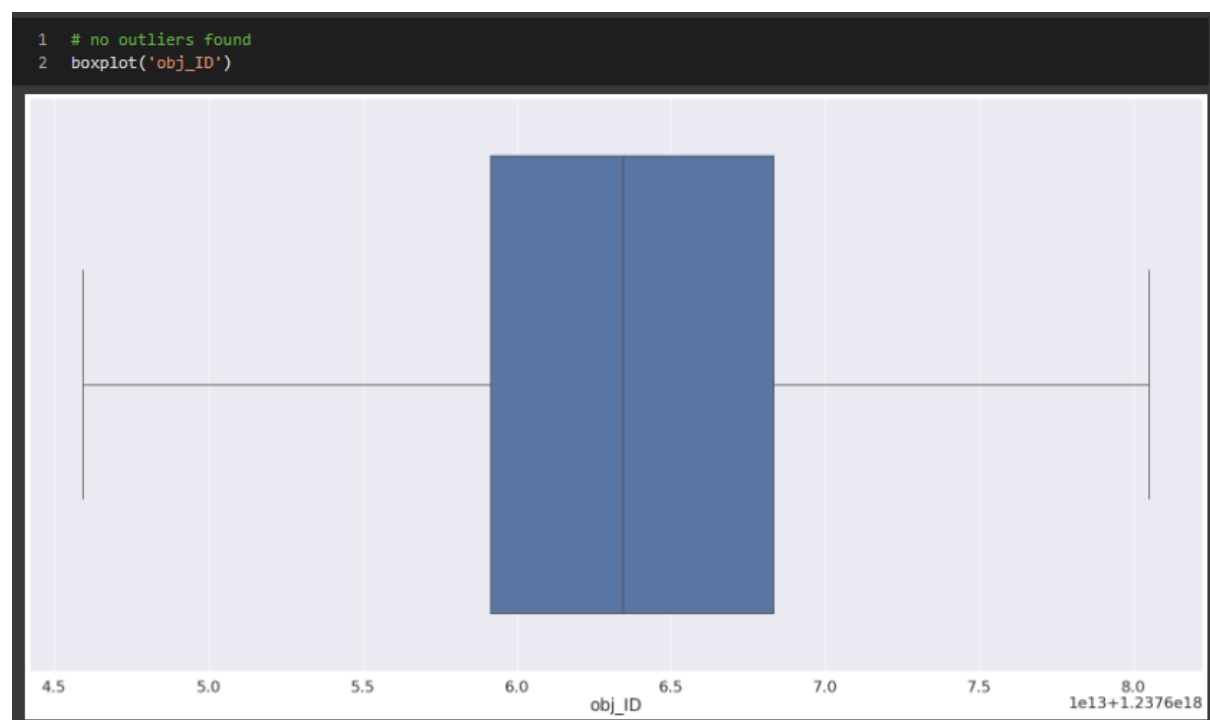


Figure 8 checking outliers on `obj_ID`

This feature has no outliers as there are no points outside either end of IQR's 50% range.

- alpha

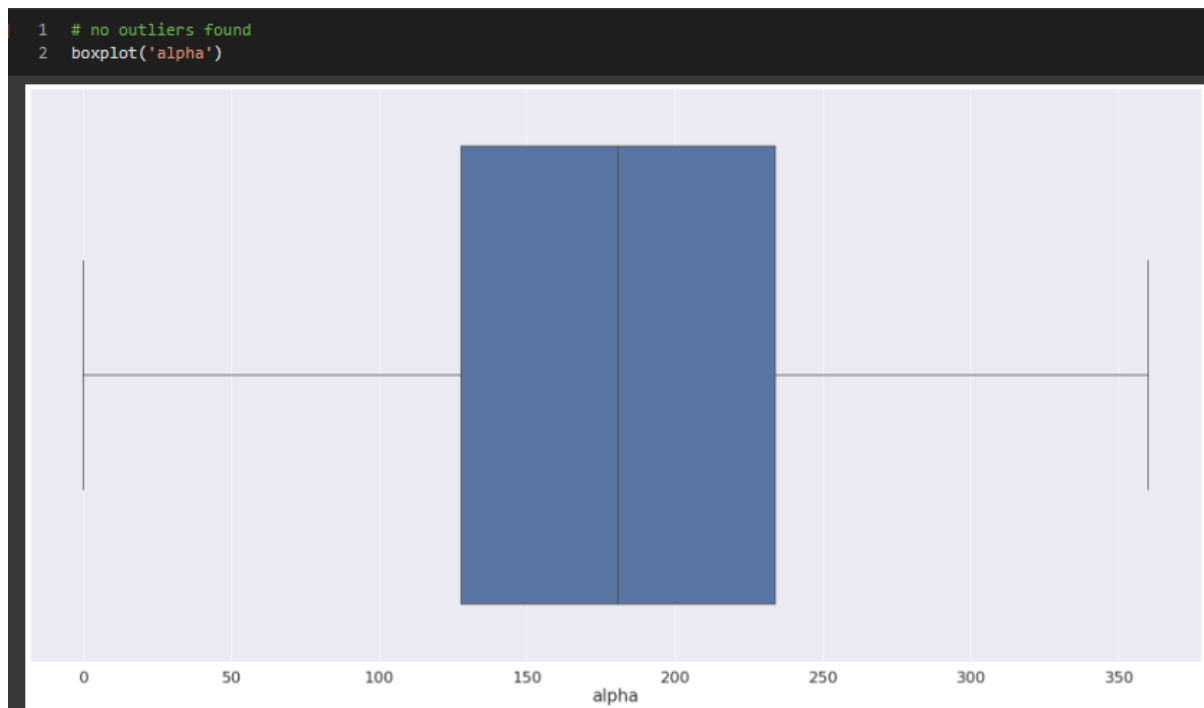


Figure 9 checking outliers on alpha

This feature has no outliers as there are no points outside either end of IQR's 50% range.

- delta

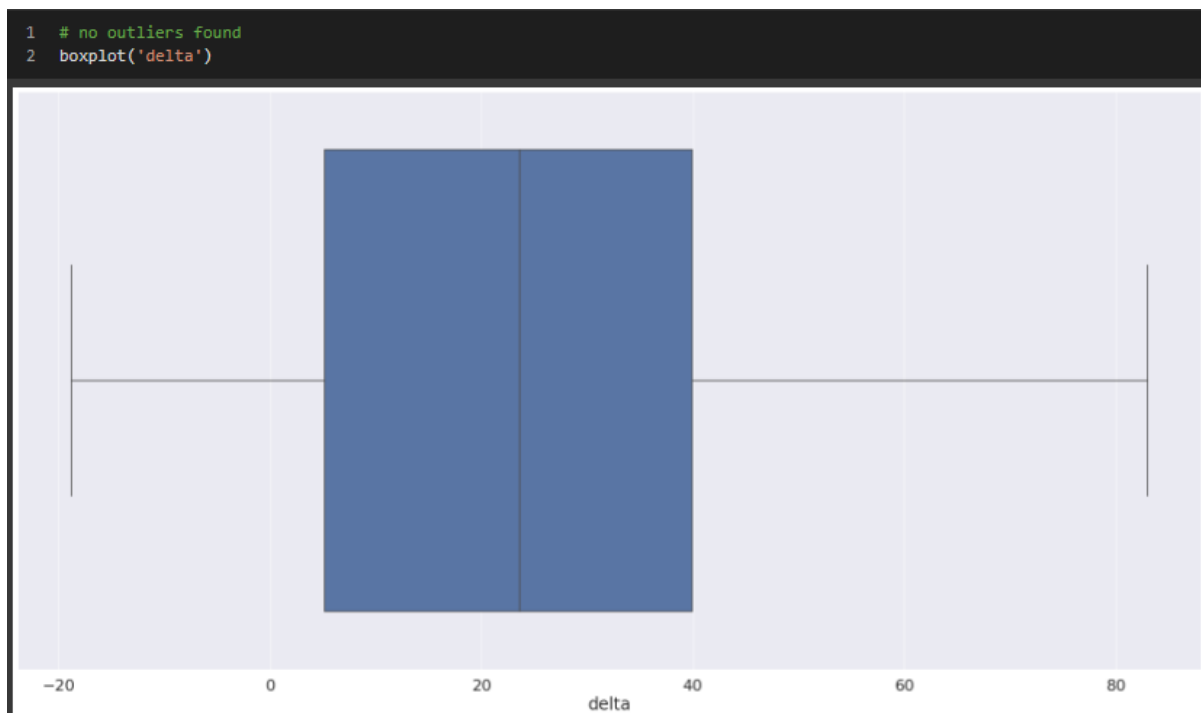


Figure 10 checking outliers on delta

This feature has no outliers as there are no points outside either end of IQR's 50% range.

- u

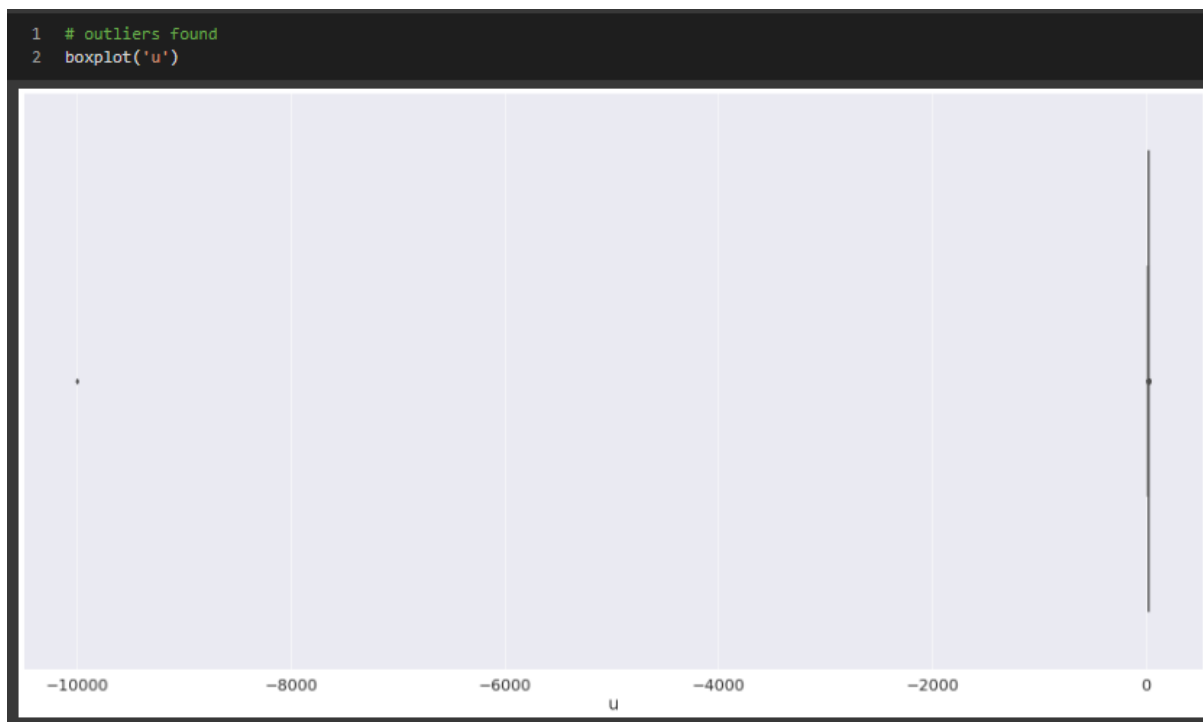


Figure 11 checking outliers on u

Contains many outliers towards the right of IQR's 50% range, and also some on the right.

- g

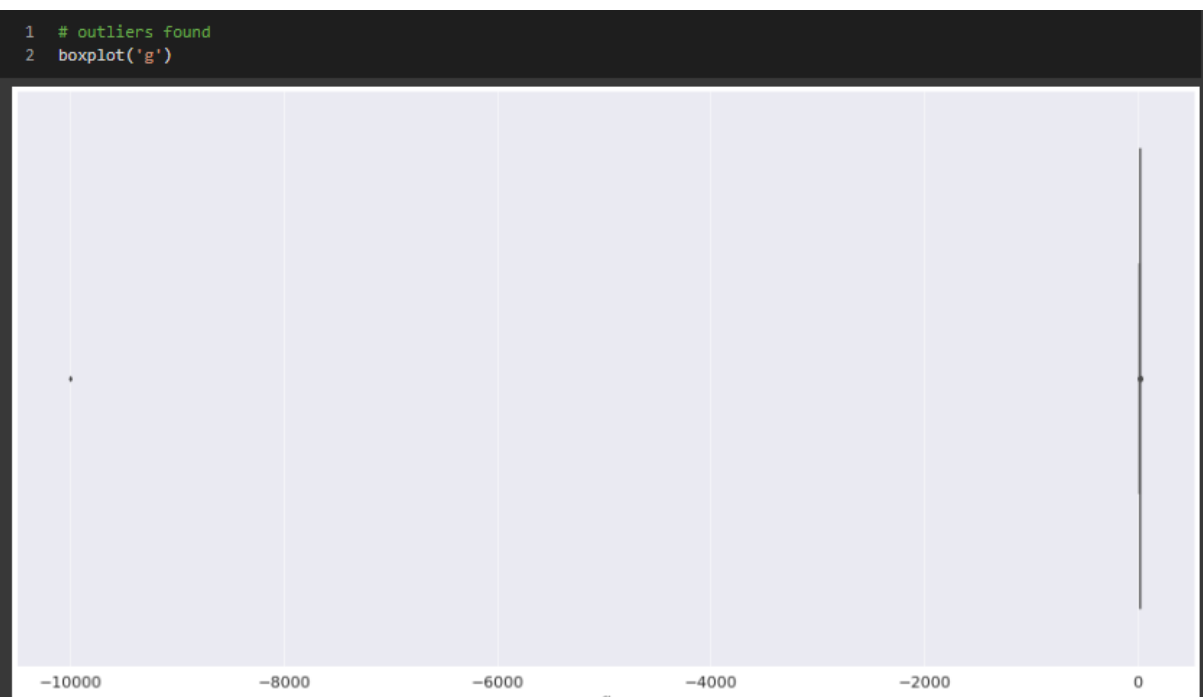


Figure 12 checking outliers on g

Contains many outliers towards the right of IQR's 50% range, and also some on the right.

- r

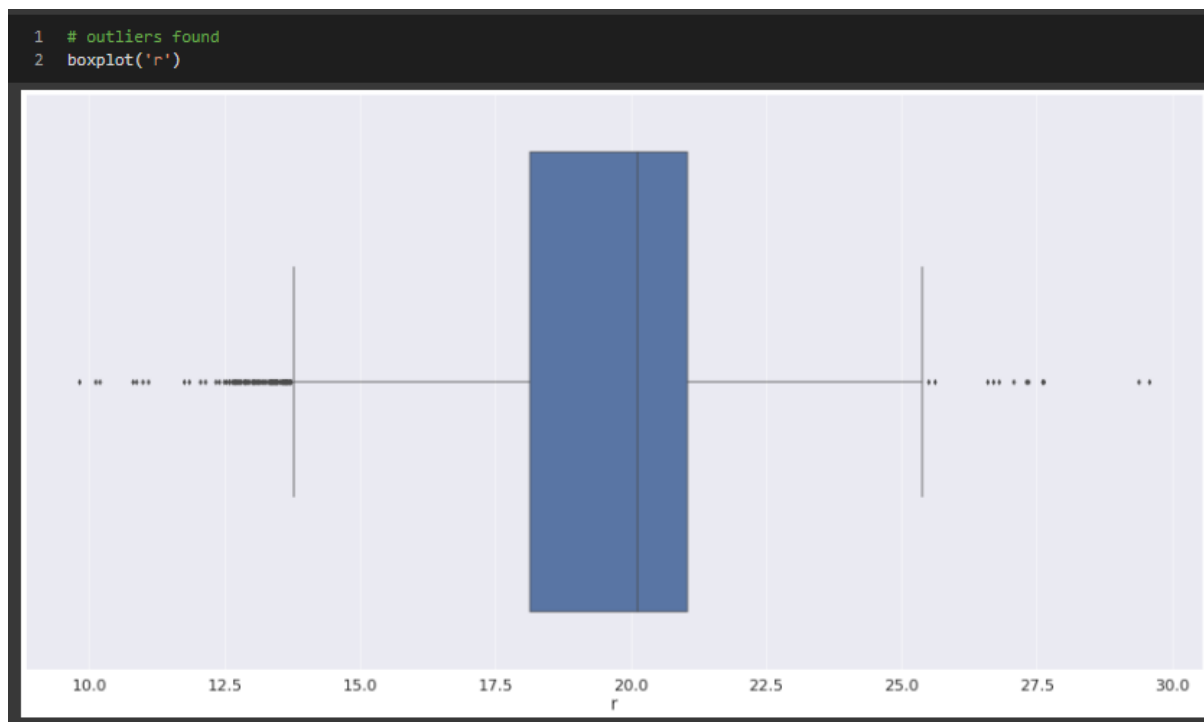


Figure 13 checking outliers on r

Contains many outliers on the right and left side of IQR's 50% range.

- i

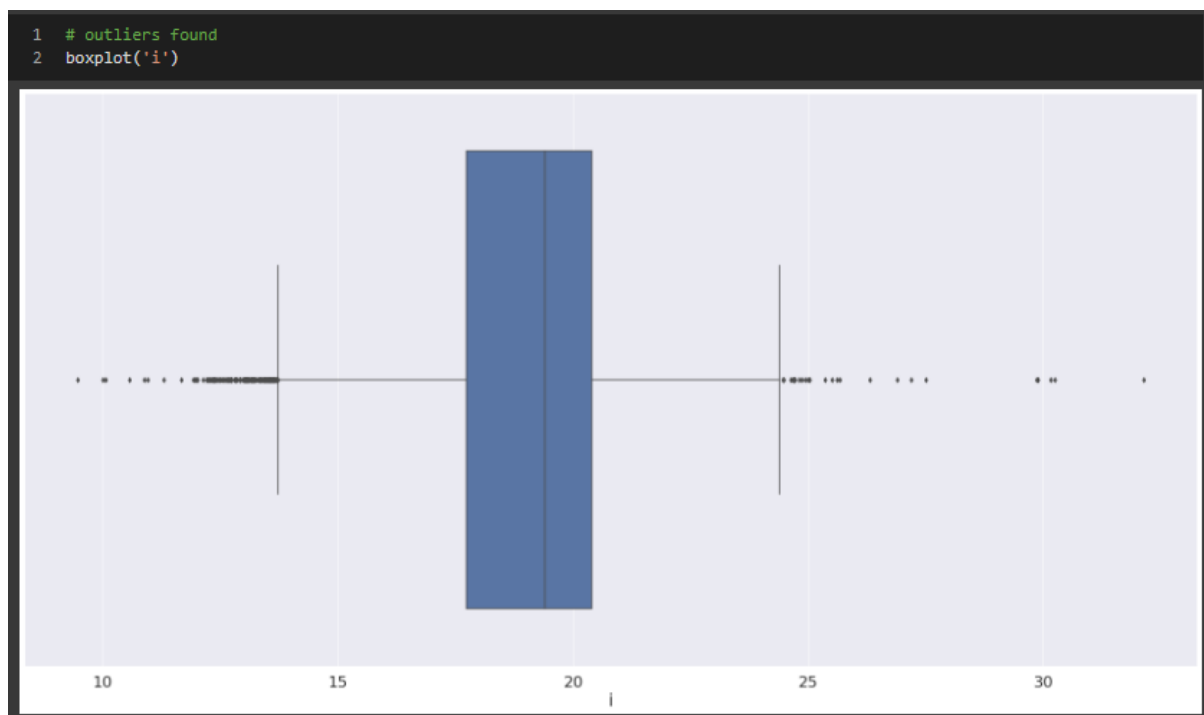


Figure 14 checking outliers on i

Contains many outliers on the right and left side of IQR's 50% range.

- z

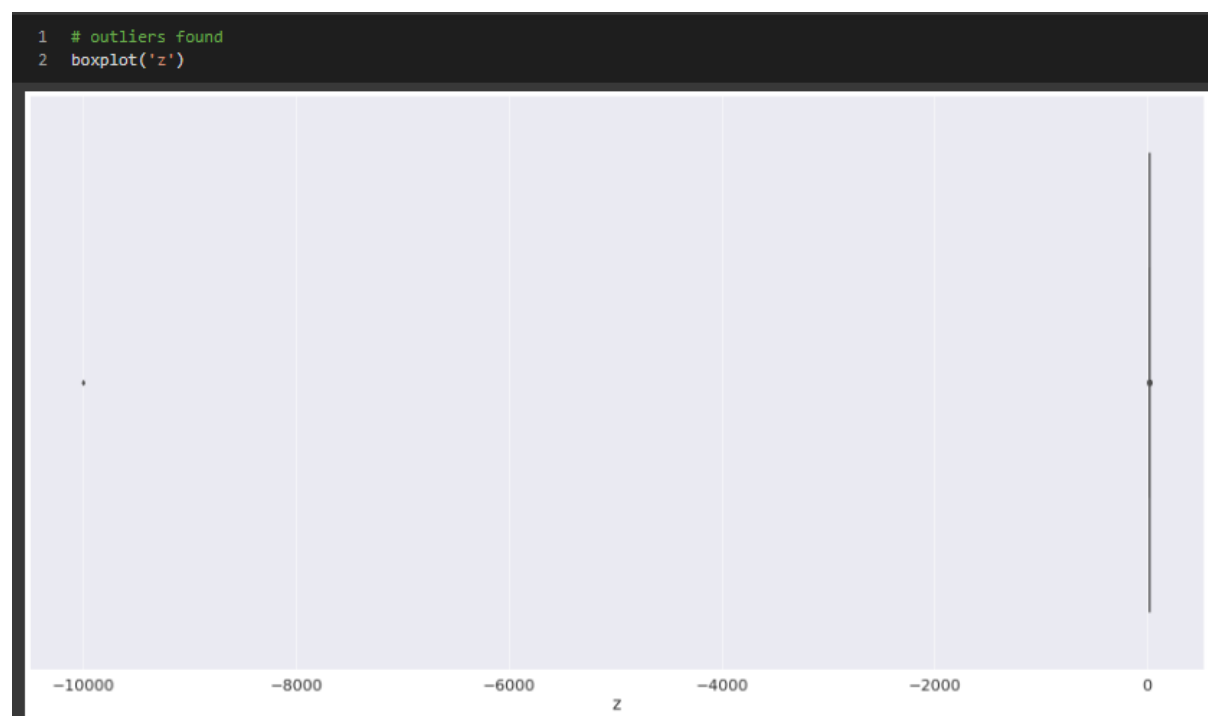


Figure 15 checking outliers on z

Contains many outliers towards the right of IQR's 50% range, and also some on the right.

- run_ID

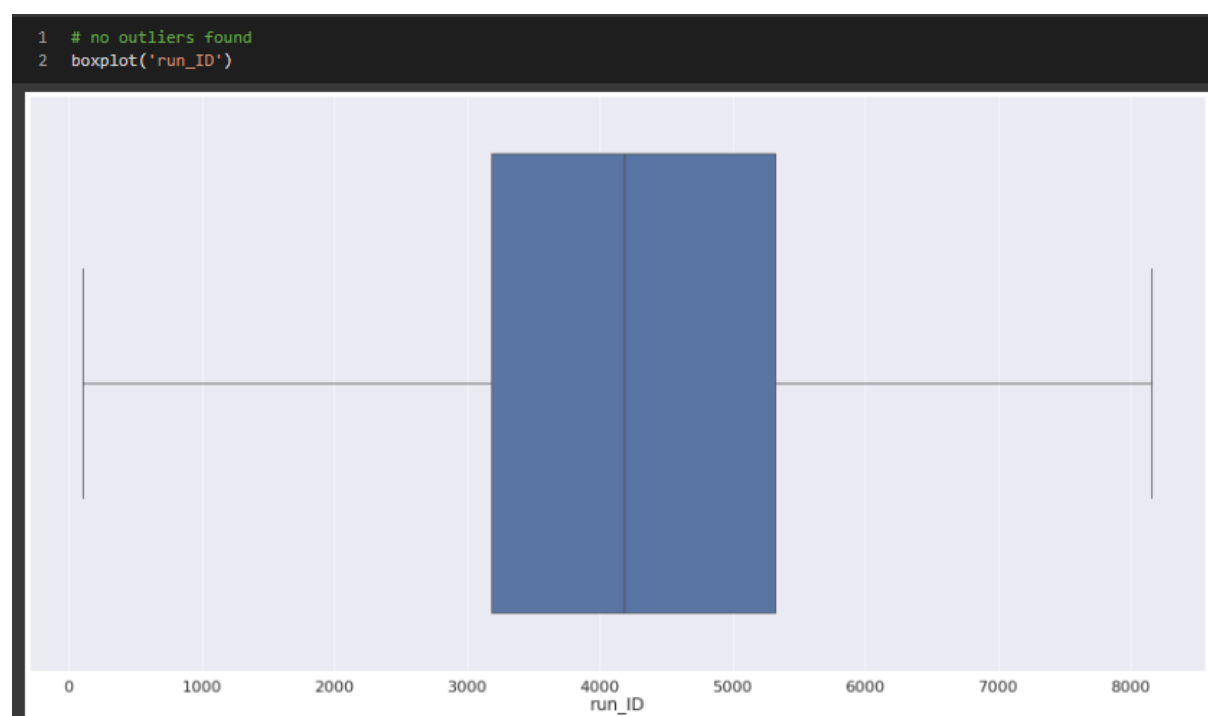


Figure 16 checking outliers on run_id

This feature has no outliers as there are no points outside either end of IQR's 50% range.

- rerun_ID

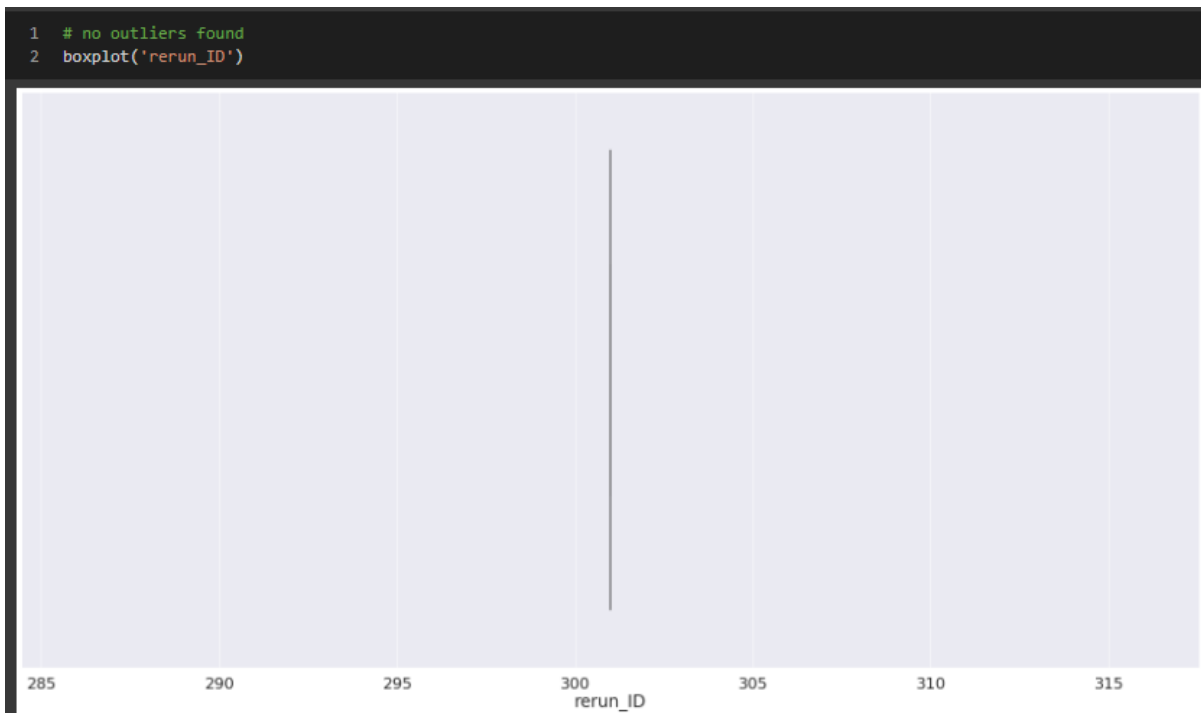


Figure 17 checking outliers on rerun_id

Very odd-looking graph, that's very different from the others but it has no outliers.

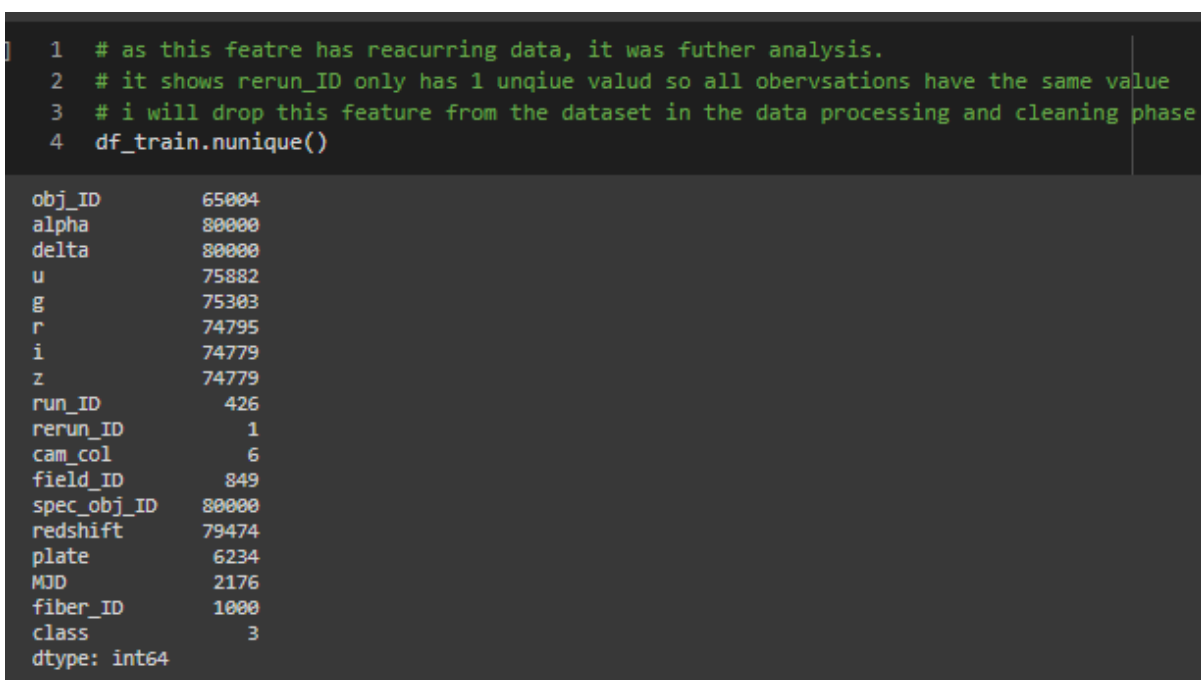


Figure 18 checking rerun unqiue values total

After further research we found that this feature has recurring values. Every features has the same value, this feature should also be removed as it will not help train the model.

- cam_col

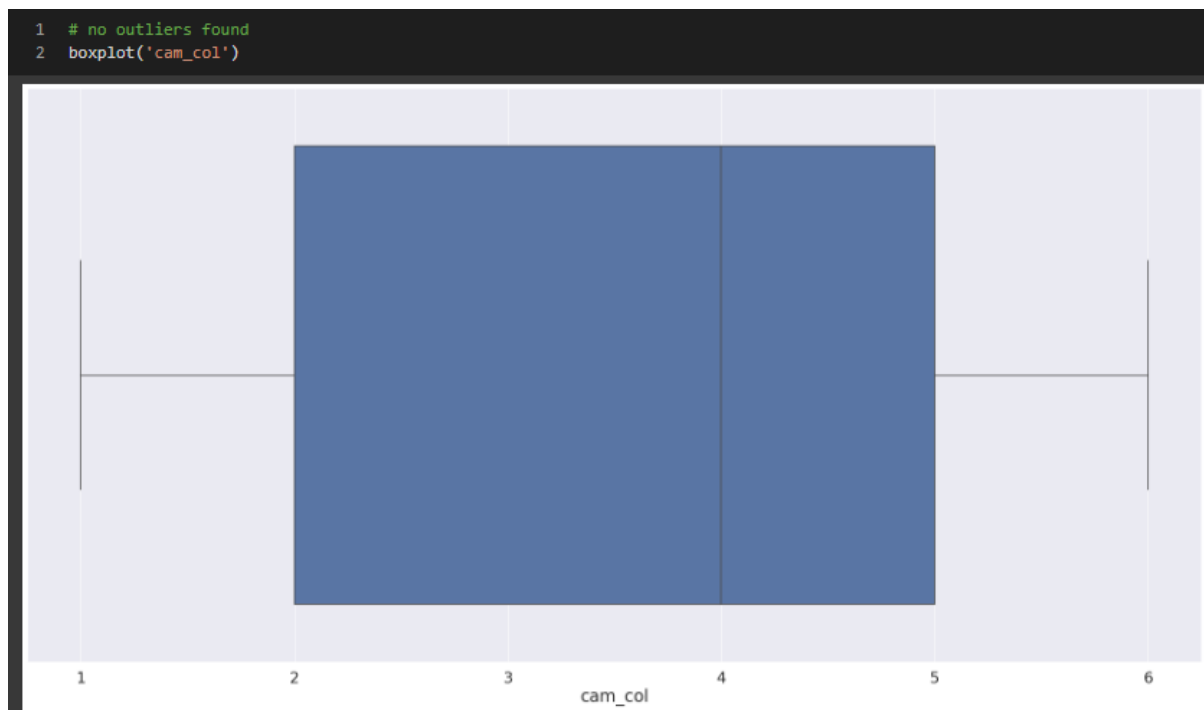


Figure 19 checking outliers on cam_col

This feature has no outliers as there are no points outside either end of IQR's 50% range.

- field_ID

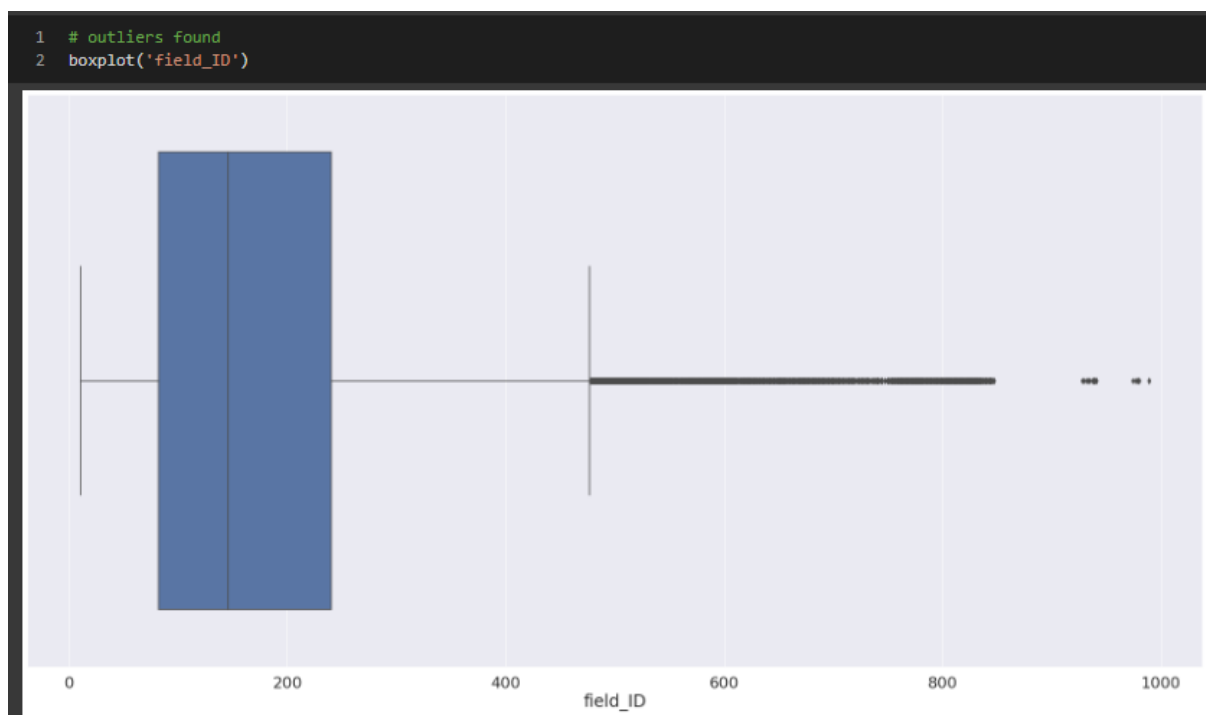


Figure 20 checking outliers on field_id

Contains many values on the right side of IQR's 50% range.

- spec_obj_ID

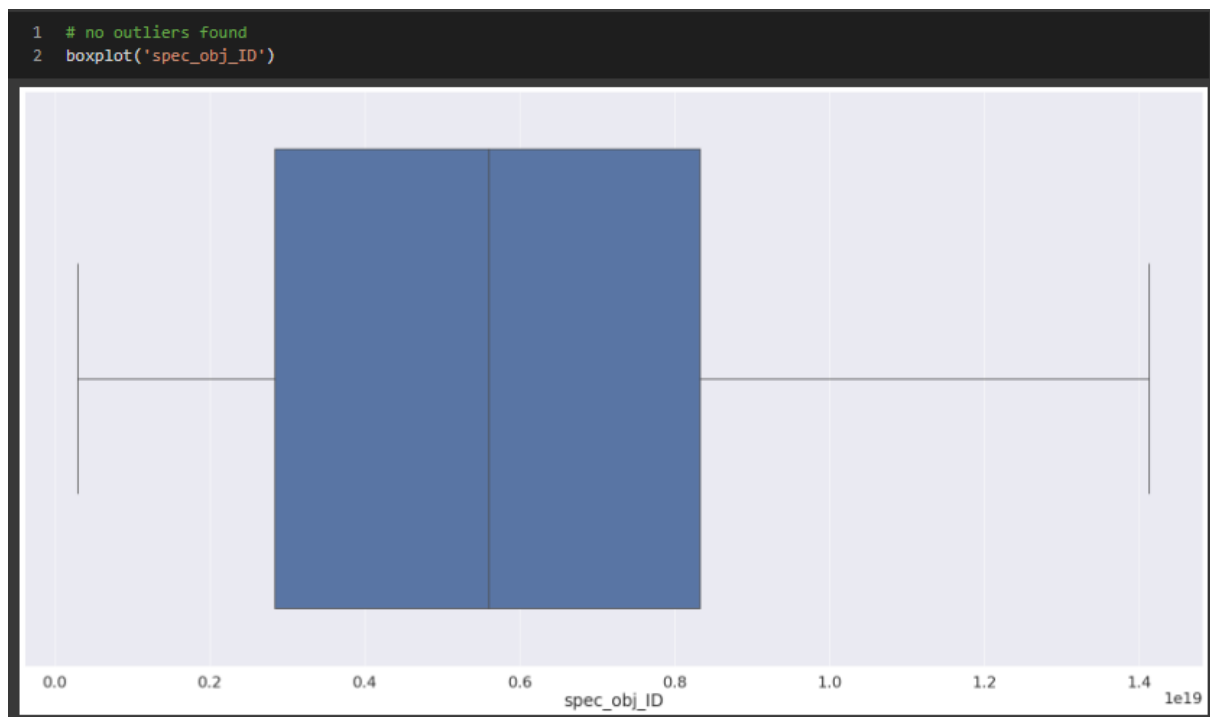


Figure 21 checking outliers on spec_obj_id

This feature has no outliers as there are no points outside either end of IQR's 50% range.

- redshift

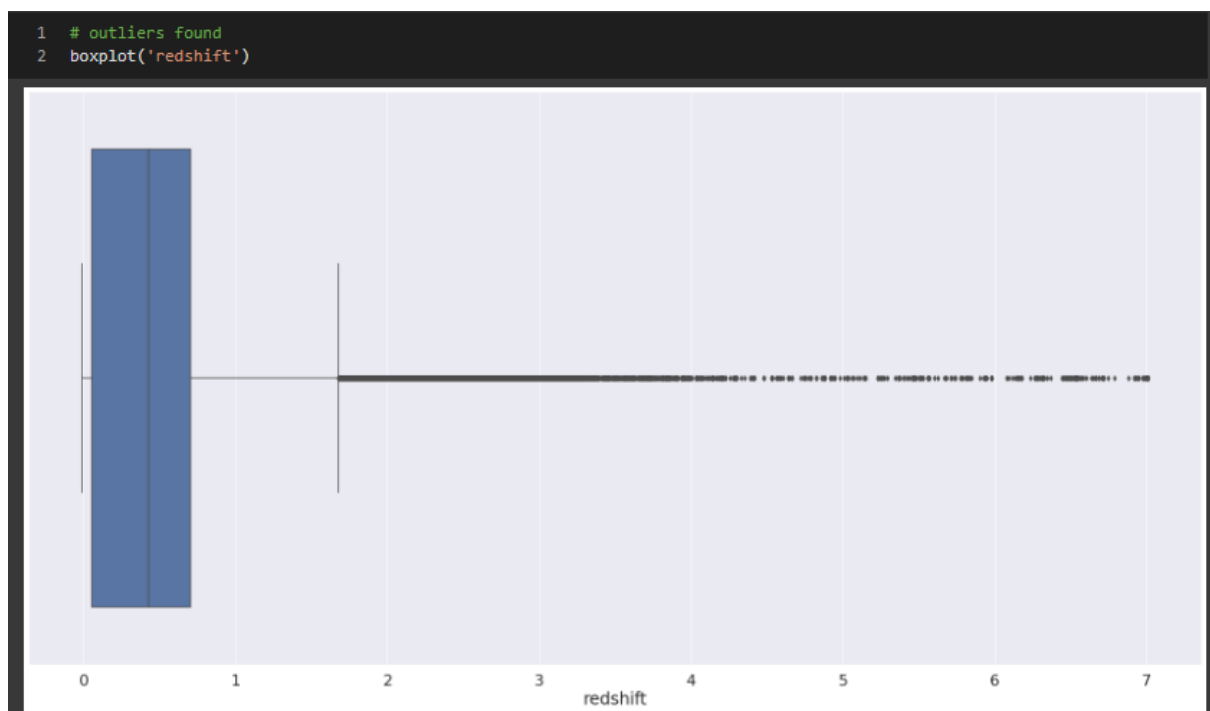


Figure 22 checking outliers on redshift

Contains many values on the right side of IQR's 50% range.

- plate

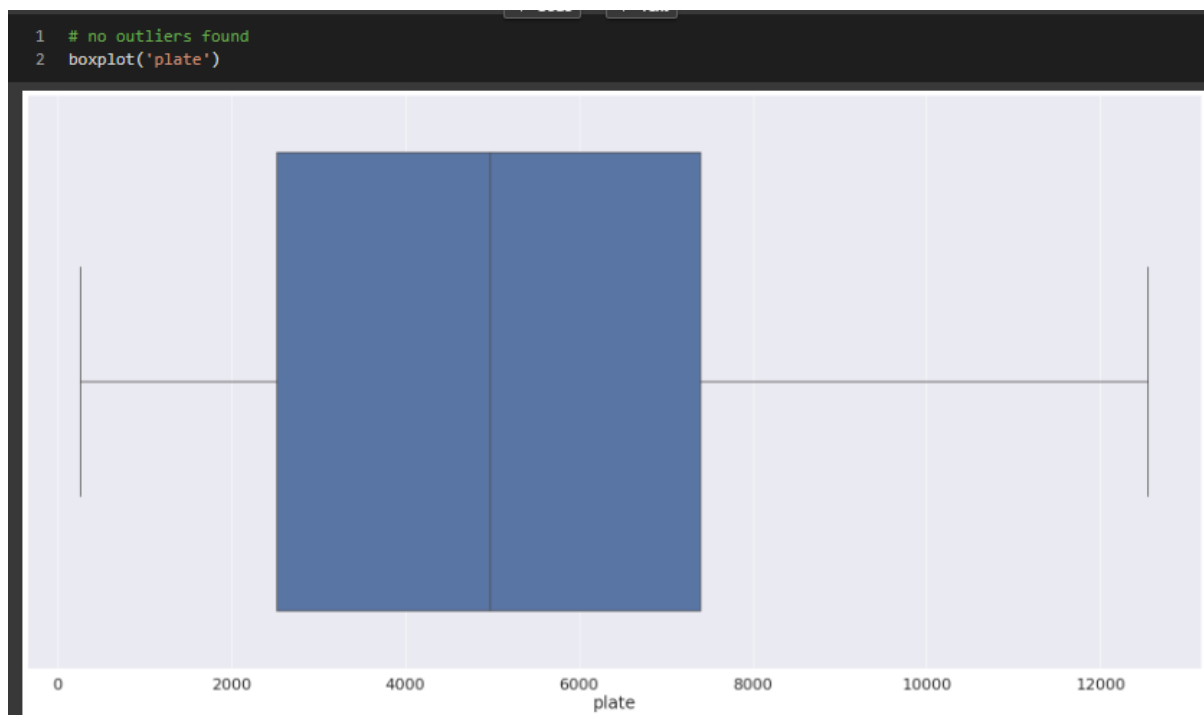


Figure 23 checking outliers on plate

This feature has no outliers as there are no points outside either end of IQR's 50% range.

- MJD

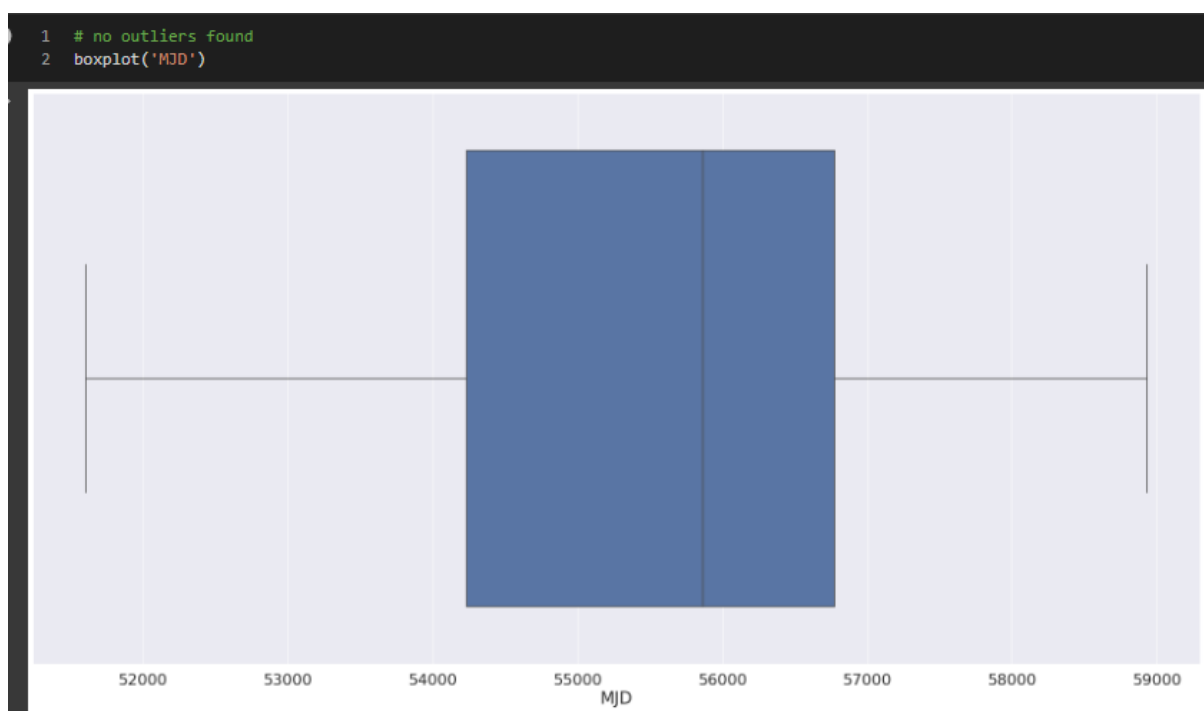


Figure 24 checking outliers on MJD

This feature has no outliers as there are no points outside either end of IQR's 50% range.

- fiber_id

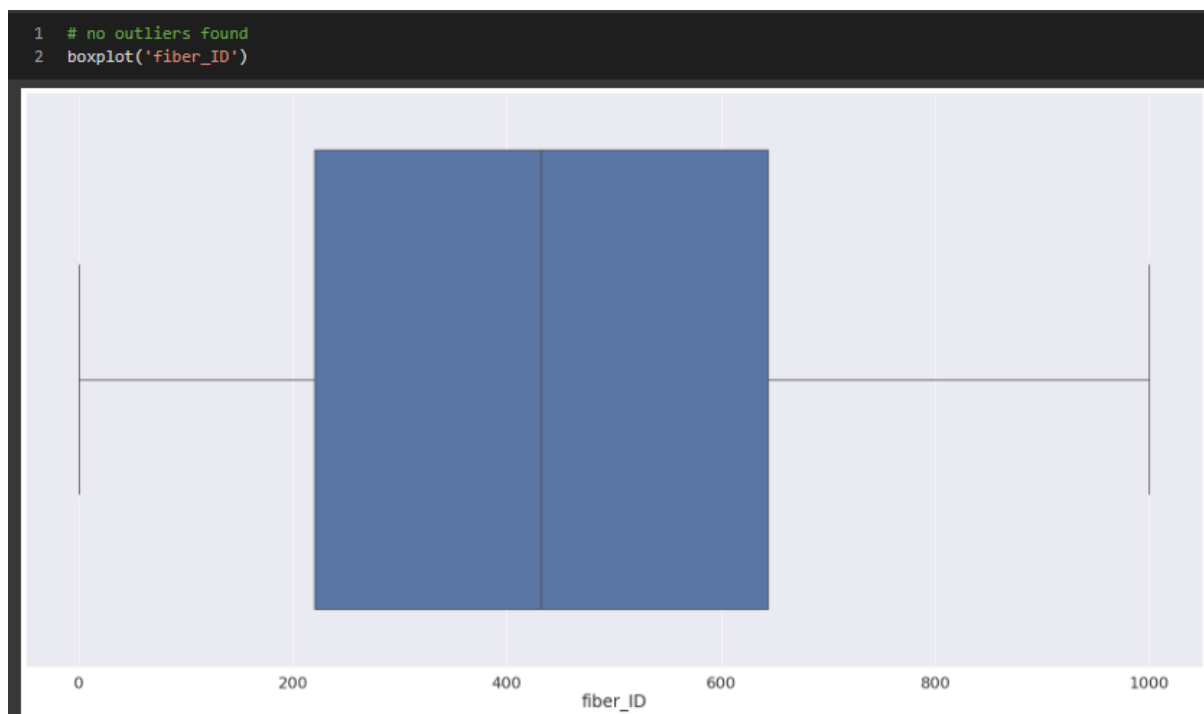


Figure 25 checking outliers on fiber_id

This feature has no outliers as there are no points outside either end of IQR's 50% range.

Concluded that a total of 8 features contain outliers. The outliers on the u, g and z features have extremely high extremely high valued outliers towards the left and possible towards the right, we can tell as IQR's 50% range is not on the graph. Field_ID and redshift was many outliers greater than the IQR's 50% range, while l and r have outliers from both ends. We also found an occurrence of recurring values within rerun_ID.

2.3.1.3 - Individual Feature to Target Variable Analysis

- **EDA question (4): Do any of the features have no correlation to the target variable 'class', if so should they be dropped?**

For the univariate analysis of the features, kdeplots were used to see if any features have no correlation to the target class. This was one of our EDA questions (Question 4) we set out to explore, as these features make it harder for the model to differentiate and correctly predict each class because the data for each observation within the feature is too similar or the same, thus affecting performance. It having no correlation means that there is no relationship between the feature and the target variable, as it does not contain any useful data. Features that have no correlation to the target class can also cause multicollinearity issues, which means the model is measuring similar data for each class, which can lead to unreliable results. So, if the data is all the same throughout each class for a feature then this data is useless. All features should have a correlation to the target class to prevent these issues.

Kdeplots were specifically used for this analysis, as they work well at showing the density of data for each class. This then can be easily compared to see if the density of data is too similar or the same for each class, if they are then the feature has no correlation to the target class. If the dataset does contain features that have no correlation to the target class, they will be removed in the data cleaning and pre-processing phase.

- obj_ID



```
1 # Corlation is too similar with this feature for each class
2 # this will be no use in the model be cause of multicollinearity.
3 # This feature will get dropped in the data processing and cleaning phase.
4 # this was one of our EDA questions we set out to undertake.
5
6 kdeplot('obj_ID')
```

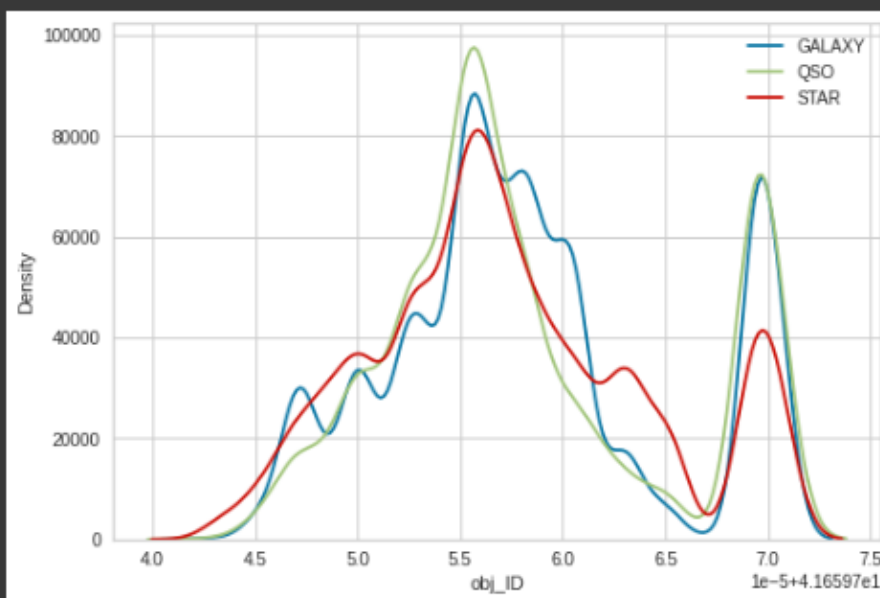


Figure 26 checking if obj_id has no correlation to target class

Each class is considered too similar to each other, especially galaxy and QSO. So this feature had no correlation to the target class.

- alpha



Figure 27 checking if alpha has no correlation to target class

Each class is considered too similar to each other. So this feature had no correlation to the target class.

- delta

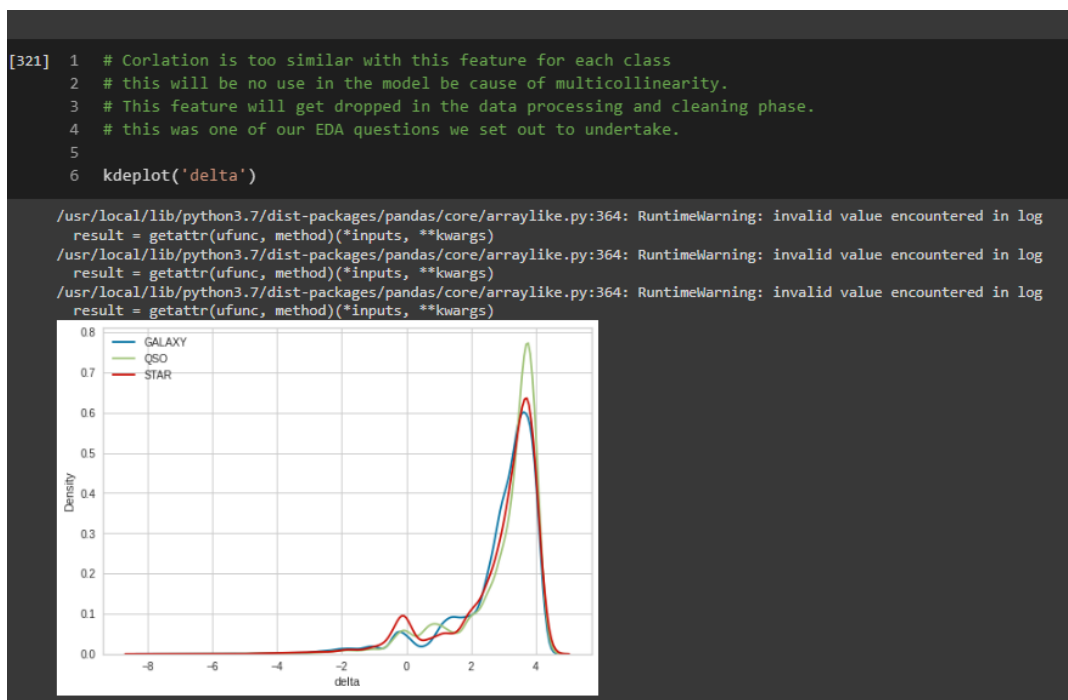


Figure 28 checking if delta has no correlation to target class

Each class is considered too similar to each other. So this feature had no correlation to the target class.

- u

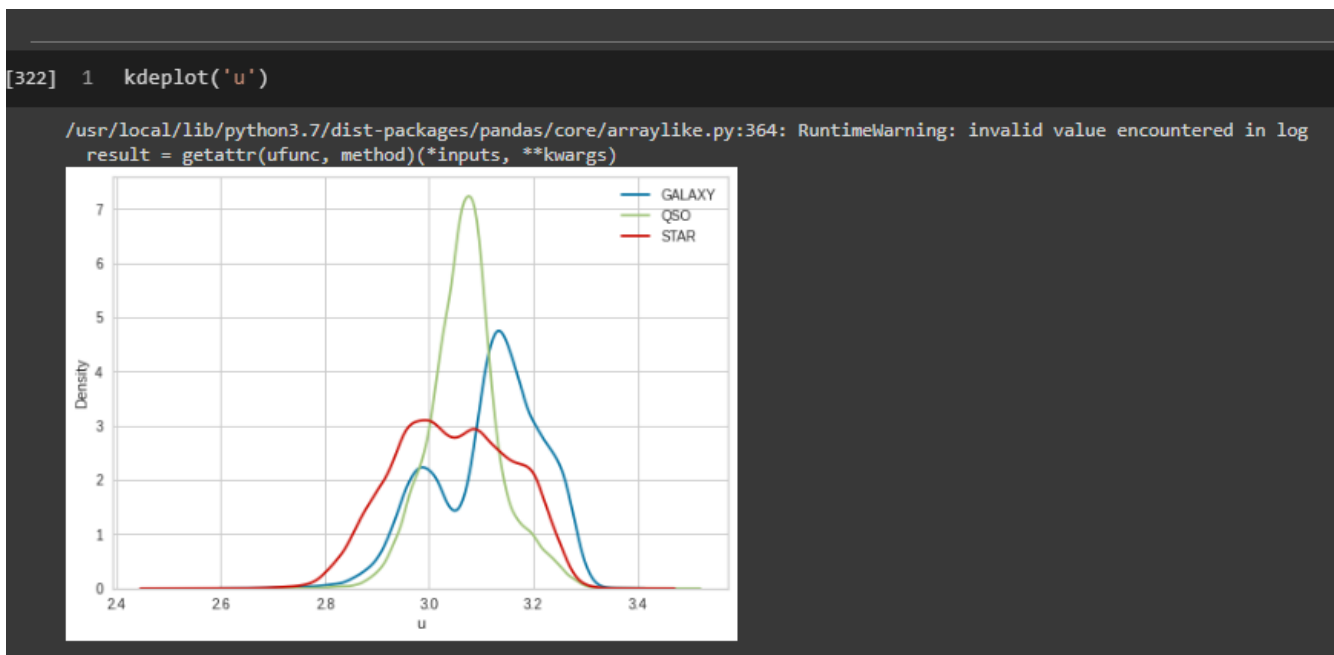


Figure 29 checking if u has no correlation to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- g

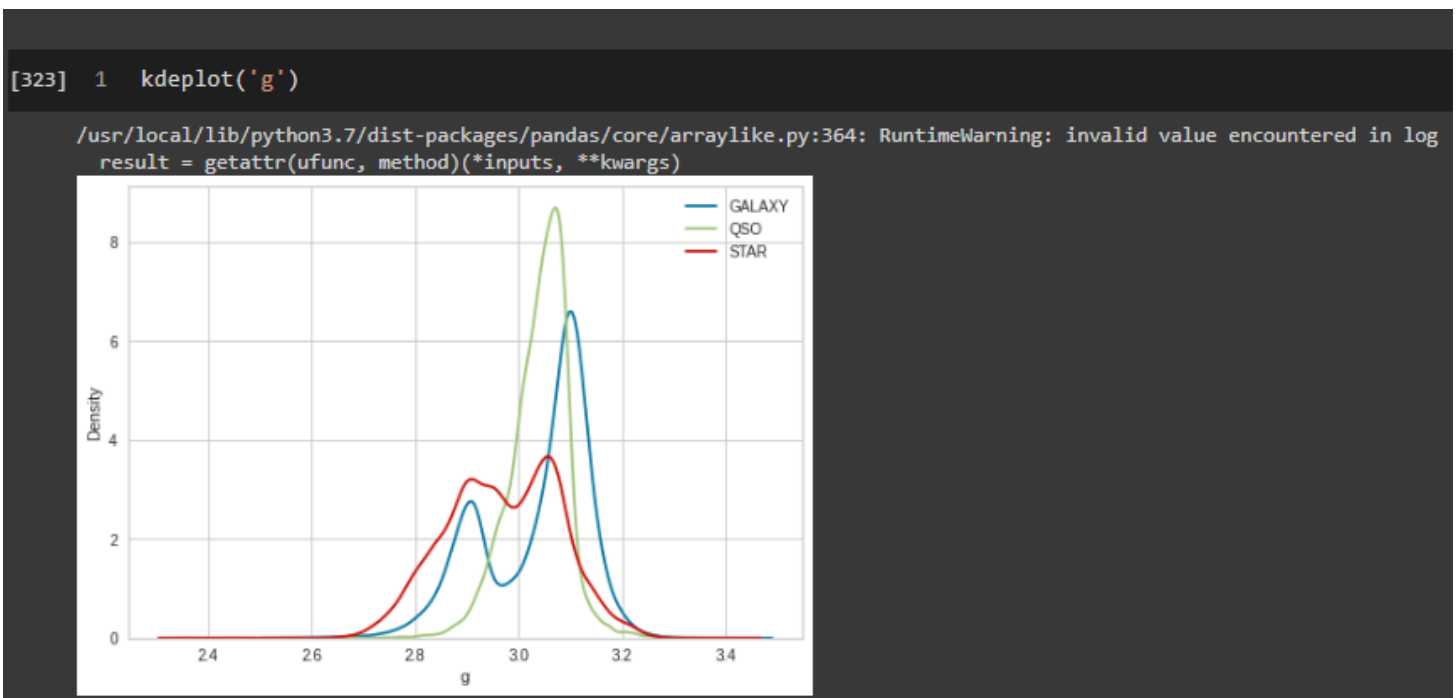


Figure 30 checking if g has no correlation to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- r

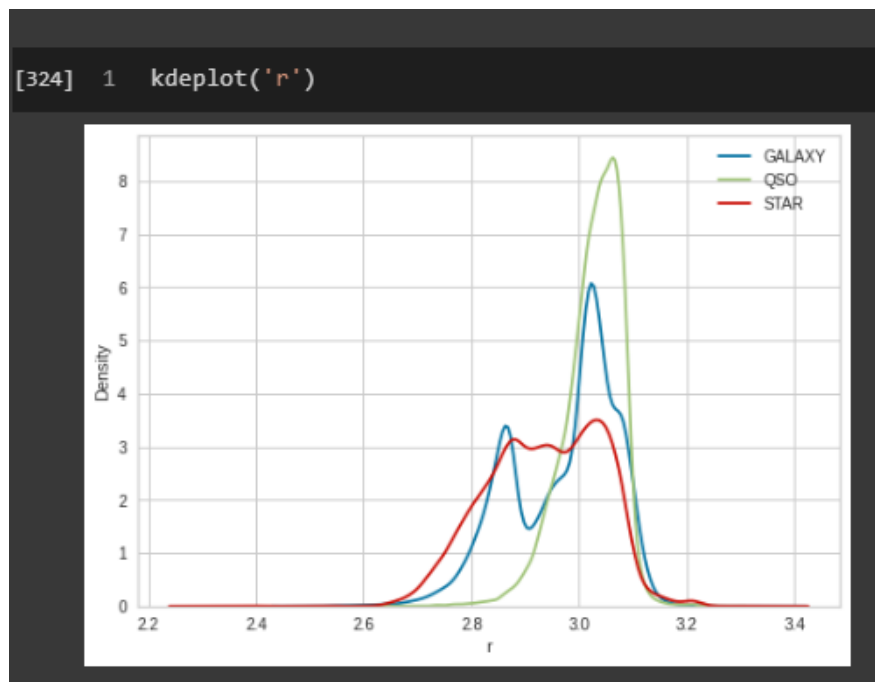


Figure 31 checking if r has no correlation to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- i

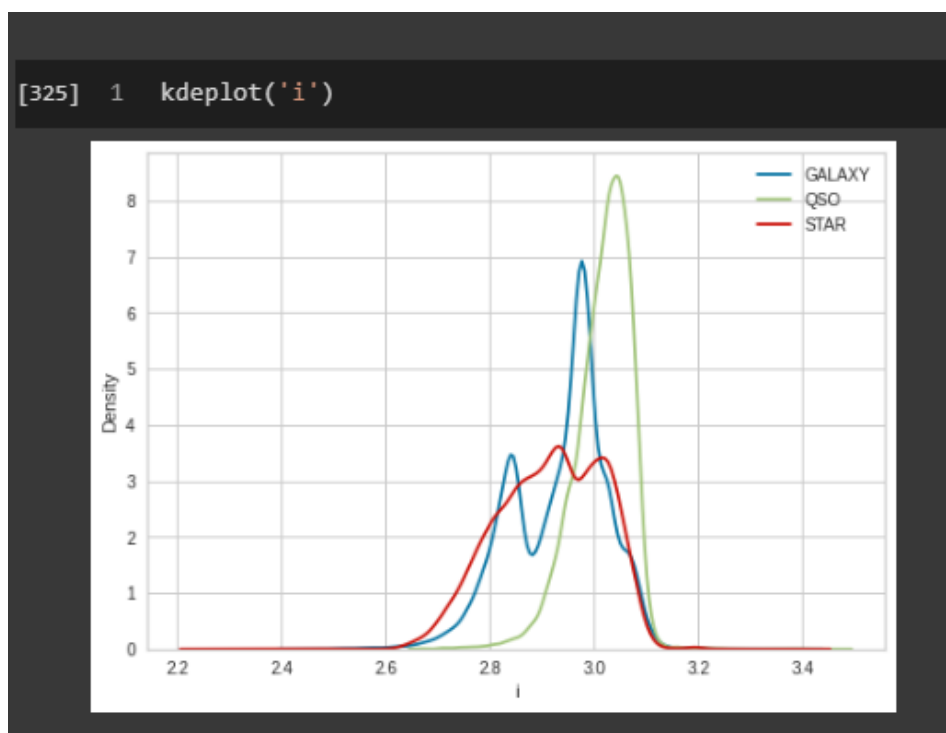


Figure 32 checking if i has no correlation to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- z

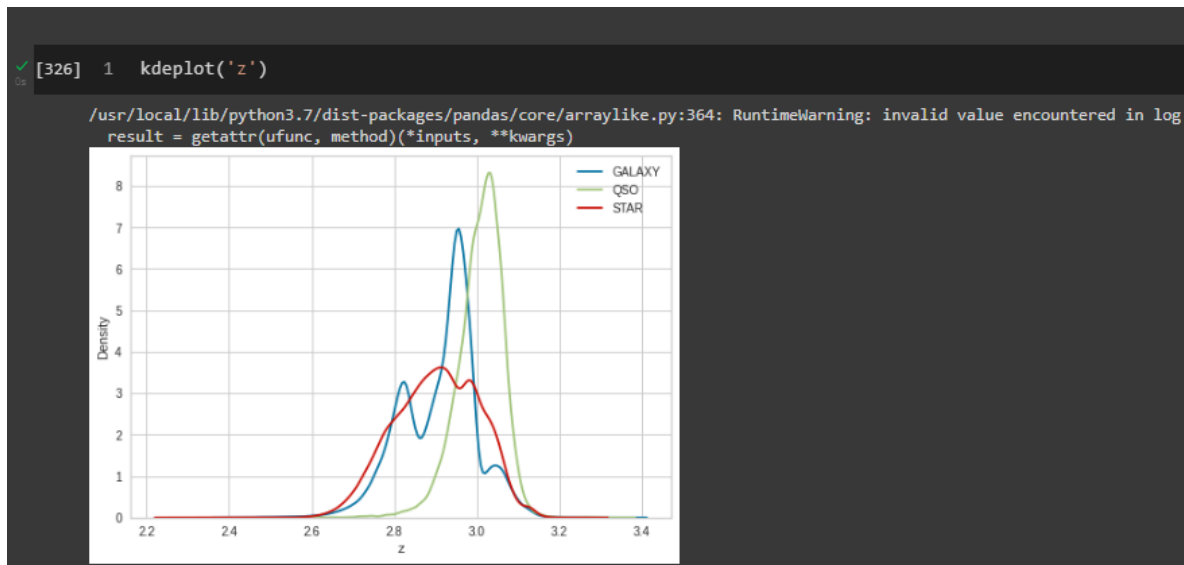


Figure 33 checking if *z* has no correlation to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- run_ID

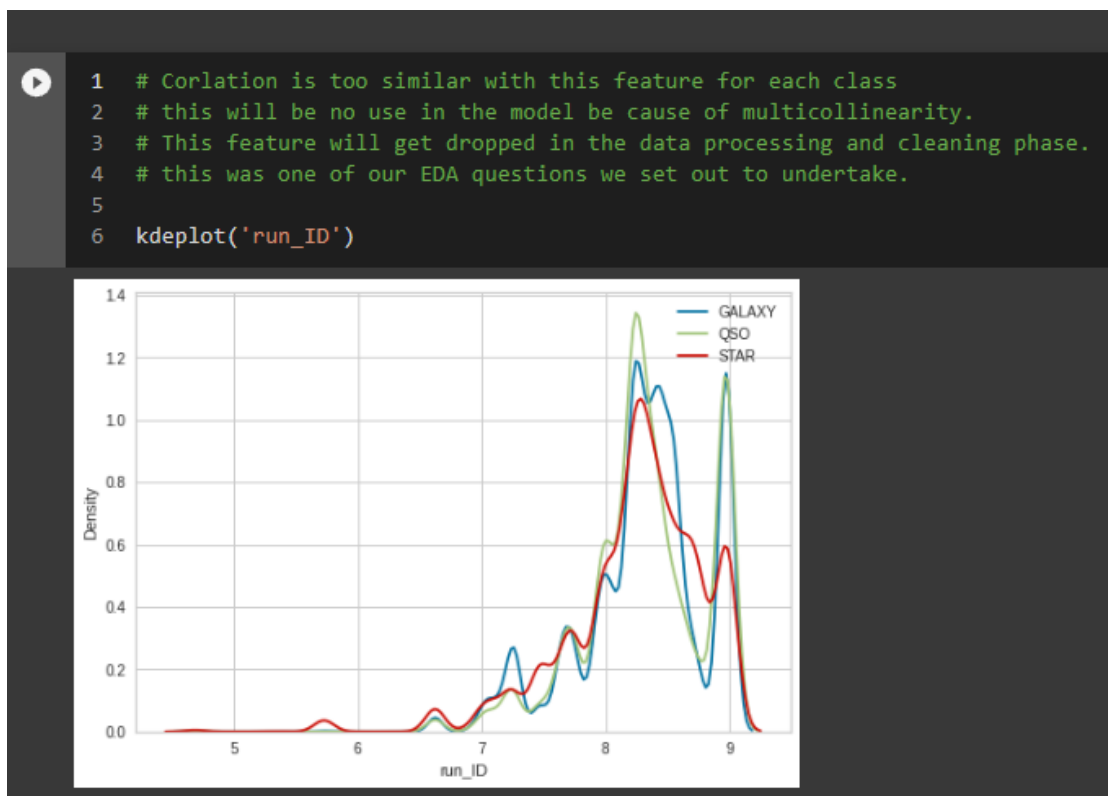


Figure 34 checking if *run_id* has no correlation to target class

Each class is considered too similar to each other. So this feature had no correlation to the target class.

- rerun_ID

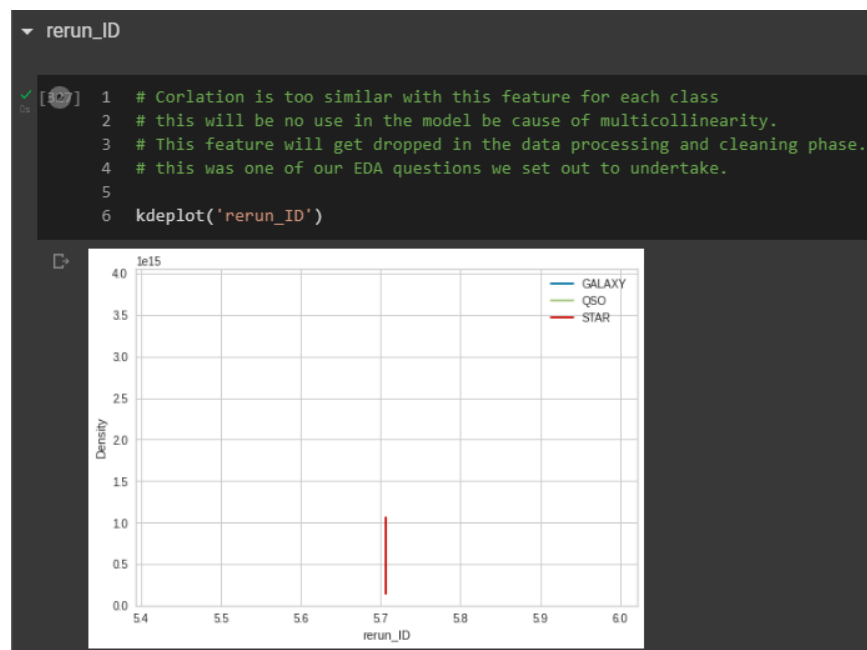


Figure 35 checking if rerun_id has no correation to target class

Odd looking graph that is different from the rest. But we know now this is because the feature has recurring values.

- cam_col

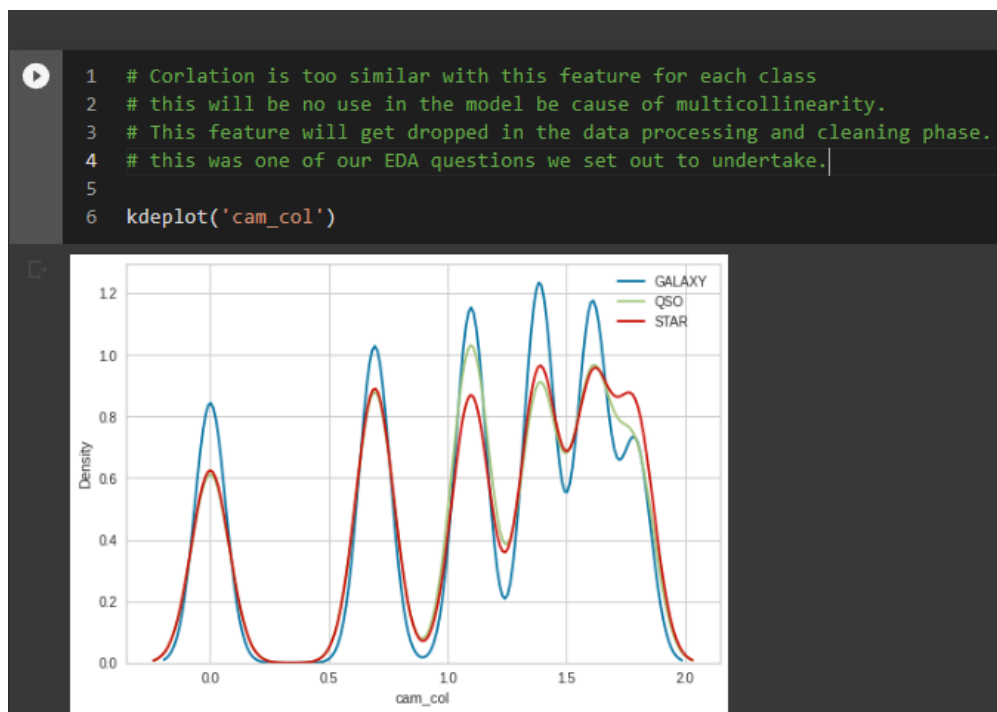


Figure 36 checking if cam_col has no correation to target class

Each class is considered too similar to each other. So this feature had no correlation to the target class.

- field_ID

```
[328] 1 # Corlation is too similar with this feature for each class  
2 # this will be no use in the model be cause of multicollinearity.  
3 # This feature will get dropped in the data processing and cleaning phase.  
4 # this was one of our EDA questions we set out to undertake.  
5  
6 kdeplot('field_ID')
```

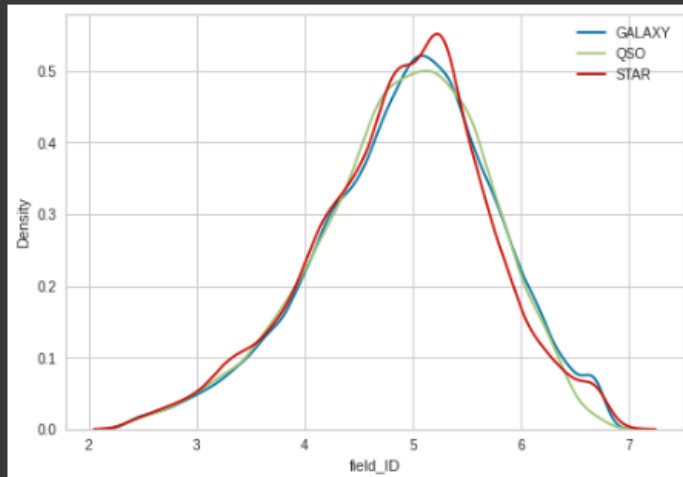


Figure 37 checking if field_id has no correaion to target class

Each class is considered too similar to each other. So this feature had no correlation to the target class.

- spec_obj_ID

```
✓ [387] 1 kdeplot('spec_obj_ID')
```

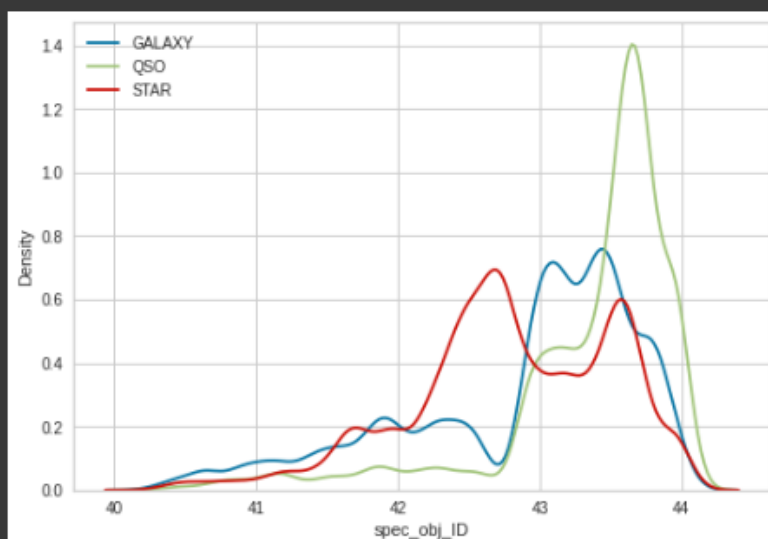


Figure 38 checking if spec_obj_id has no correaion to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- redshift

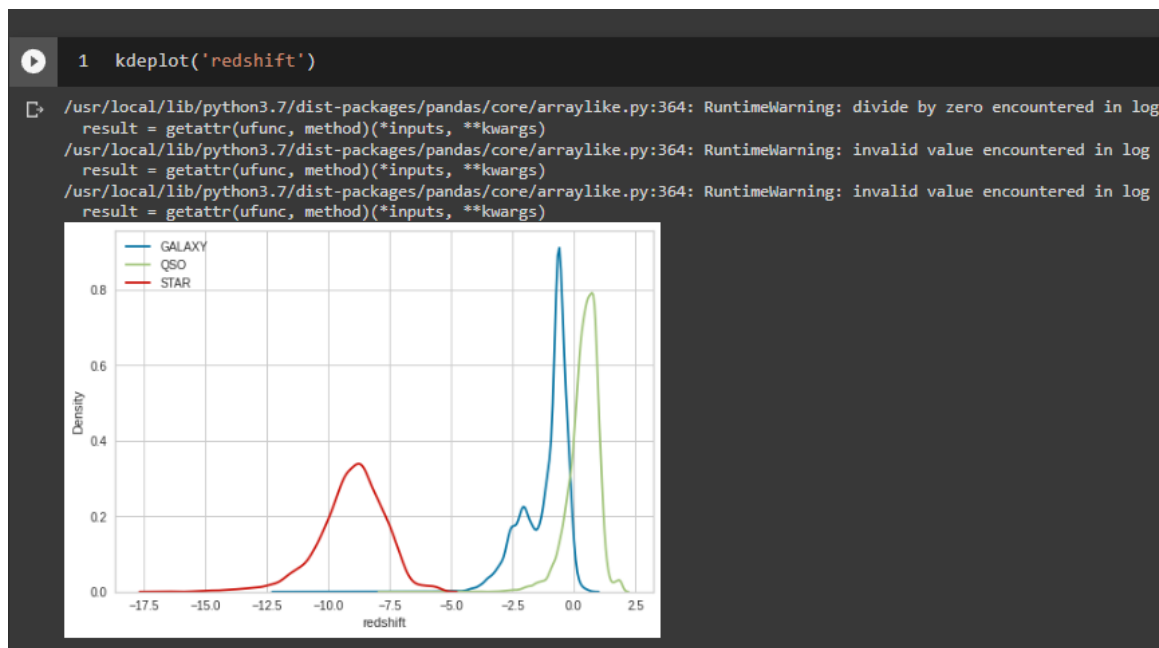


Figure 39 checking if redshift has no correlation to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- plate

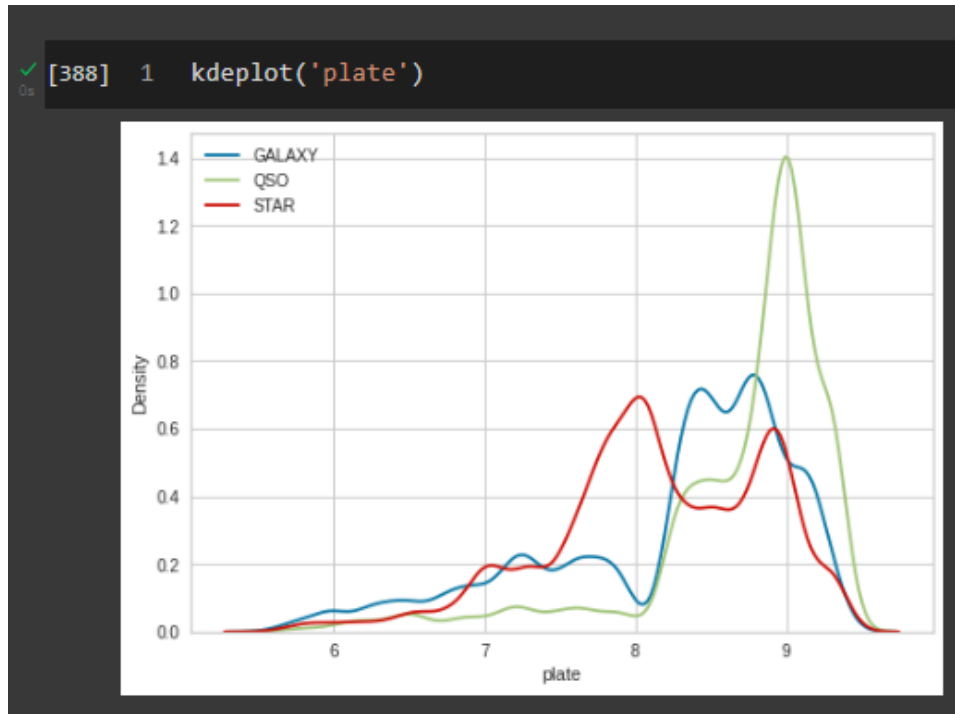


Figure 40 checking if plate has no correlation to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- MJD

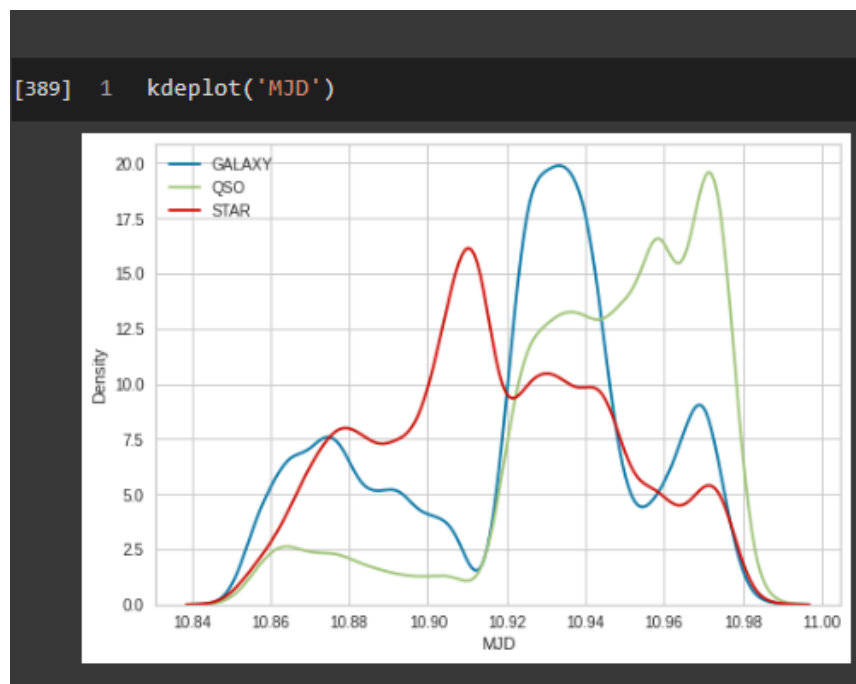


Figure 41 checking if MJD has no correlation to target class

The data distribution is morally different for each class, so they have correlation to the target class.

- fiber_ID

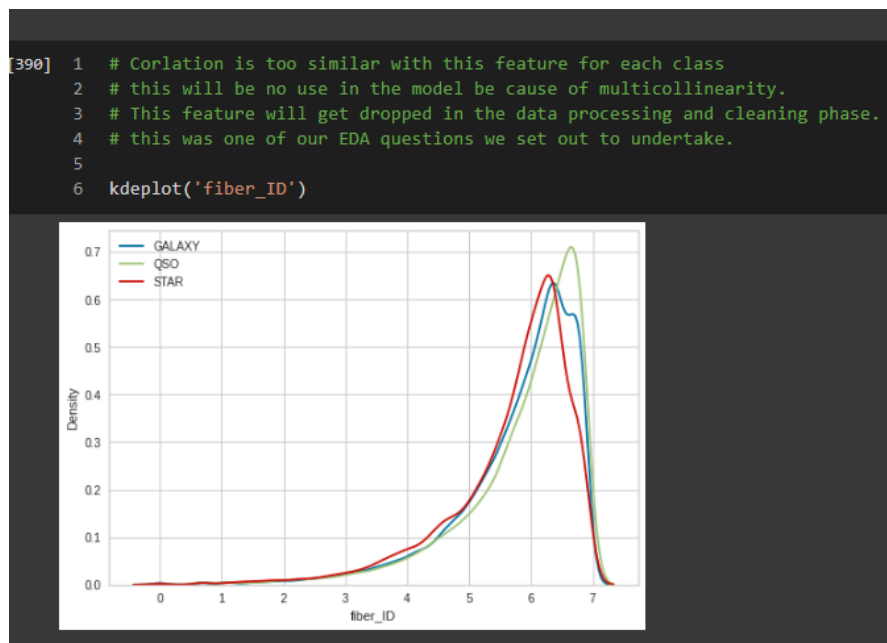


Figure 42 checking if fiber_id has no correlation to target class

Each class is considered too similar to each other. So this feature had no correlation to the target class.

Concluded that a total of 8 features had no correlation to the target class that will be later removed.

2.3.1.4 – Redshift to Target Variable Analysis

- **EDA question (6): *How is redshift affected by the distance of celestial bodies?***
- **EDA question (7): *Are the majority of observations in the high end or low end of redshift?***

The redshift feature was analysed for univariate analysis using boxplots to better understand its relationship with the target variable in real-world terms. Question 6 was set out to explore to see how the redshift value is affected by the distance of galaxies, stars, and quasars. To better understand how the redshift value correlates with stellar objects. Question 7 was set out to explore to see if the majority of observations are the high-end or low-end within the redshift wavelength spectrum. If they are high end it will mean the stellar objects moving away from us at a fast pace, while low-end means the stellar objects are relatively close to us and are moving away from us at a slower pace.

Boxplots were specifically used for this analysis, as they work well at showing the distribution of data for each target variable so these questions can be determined.



Figure 43 checking redshift relationship with target varibales

Figure 25 shows that there are more observations at the right side of the middle 50% of the data for galaxy and quasar, so majority of them are at the high end of the redshift wavelength spectrum, so the galaxies and quasars are moving away from us at a fast pace. But for stars, the majority of observations was to the left so at the low end the redshift wavelength spectrum, meaning they are close to us and are moving away from us at a slower pace. This concludes question 7.

It also shows when looking the measurements, the father away the observation from us is the larger the redshift, so the closer the obsession is from us the lower the redshift. This concludes question 6.

2.3.2 - Multivariate Analysis

Multivariate_analysis was performed to analyse data compared with two or more variables. Questions 1 and 5 were explored here.

2.3.2.1 - Checking for missing values

- EDA question (1): *Are there any missing numbers within the dataset?*

For the multivariate of all the features, the msno graph was used to see if any observations within features have missing values. This was one of our EDA questions (Question 1) we set out to explore, as the model cannot be efficiently trained or tested with observations that have important missing data. Affecting performance and overall accuracy as the model is being tested and trained with insufficient data. To prevent these issues, all observations within features should be removed along with their missing values.

The msno graph was specifically used for this analysis, as it shows patterns of missing values for each feature so they can be easily identified. If the dataset indeed has observations with missing values, these observations will be removed in the data cleaning and pre-processing phase.

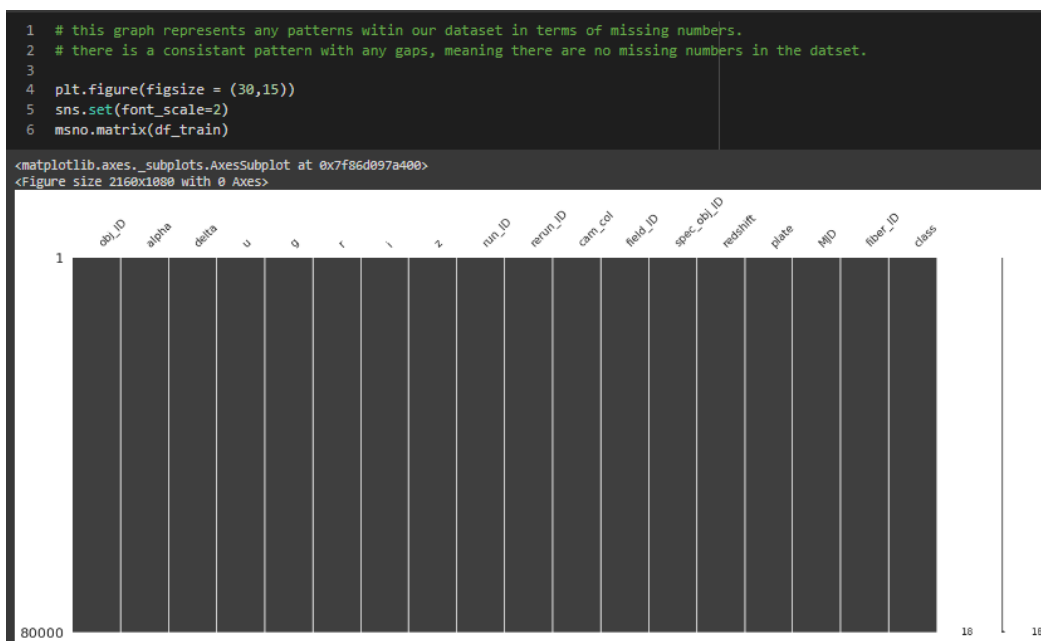


Figure 44 checking for missing values in dataet using mnso graph

Graph shows there is no NULL values in the training set as all the bars are grey for each feature.

```
▼ Checking for missing values

[ ] 1 # Through analysing the dataset, no missing values was found.
    2 # so no further action needs to take place here later on.
    3 # this was one of our EDA questions we set out to solve.
    4
    5 df_train.isnull().sum()

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

Figure 45 total NULL values in each dataset

Can see there are 0 missing NULL values for every feature on the training set, using the `isnull().sum()` function that displays the total of missing NULL values.

```
1 # there are a total of 326 values being 0, this is bad as they will be no use to train the model
2 (df_train == 0).sum().sum()

326
```

Figure 46 total of 0's in dataset

Shows that there is 323 values that our zero in the training set.

Concluded that our dataset does not have any NULL missing values, but it does contain values that are 0 which we don't want in the dataset as they doesn't make much sense. Because we had some occurrences of redshift being zero that which would means the object is not moving away from or towards the observer, but objects should always be moving as space is expanding.

2.3.2.2 - Feature to feature analysis

- **EDA question (5): Are any of the features too strongly correlated with each other causing multicollinearity problems, if so should they be dropped?**

For the multivariate analysis of the features compared to other features, a heatmap was used to view the correlation between every feature to identify if they are highly positively correlated or not. This was one of our EDA questions (Question 4) we set out to explore, as high positive correlated features can cause overfitting and multicollinearity issues as they make it difficult for the model to interpret the data of individual features as they rely on each other, thus affecting model performance. Not all highly correlated features are bad, only ones that are highly correlated with many other features. There should not be many features that are highly correlated with many others to prevent these issues.

Heatmaps were specifically used for this analysis as they work well at visualizing the correlation between every feature with a scale that measures correlation. If the dataset is indeed features that are highly correlated with many others, they will be removed in the data cleaning and pre-processing phase.

```
1 """
2 now that i have analysed the correlated for each feature to the class objects
3 i will analysis the correlated for each feature to other features
4 for this i used a heatmap and the correlation function
5 i will also look for any recurring values for features.
6 """
7 # this where two of our EDA questions we set out to analyse. (question 4 and 5)
8
9 plt.figure(figsize = (30,15))
10 sns.set(font_scale=2)
11 sns.heatmap(df_train.corr(), annot = True, fmt = ".2f", linewidths = .5, annot_kws={"size": 20})
12 plt.show()
13
14 # can see that rerun_ID has no correlation to any other features as well, as before i found it has recurring data.
15 # so it will be dropped in the data processing and cleaning phase.
```

Heatmap code

Figure 48 – Heatmap code



Figure 48 – Heatmap graph showing correlation between each feature

Concluded that many features are highly correlation with each other such as run_ID with obj_ID, u with z, r with i, spec_obj_ID with plate, and others. However they are only correlated within their own subgroups and don't effect many other features so they will not get removed. You can also see the rerun_ID that has recurring values, it has no correlation at all to anything.

2.4 EDA conclusions

In conclusion, all 7 of our EDA questions were successfully investigated through univariate and multivariate analysis. These discovered identifying issues with the dataset will be further analysed and fixed in the data process and cleaning phase.

No features were found that had missing values that were researched for EDA question (1) using a mns0 graph. This was investigated because the model cannot be efficiently trained or tested with a dataset with important missing data. This insight will lead me to apply future techniques to fix these missing value issues.

It was identified that our dataset is indeed unbalanced which was researched for EDA question (2) using a pie chart and a countplots. This was investigated because the model cannot be biased towards a class with more data than the other classes. This will affect the overall performance of the model, as one class is greatly more accurate than the others. This insight will lead me to apply future techniques to fix the unbalanced dataset and make it balanced.

8 features were found that contained outliers that were researched for EDA question (3) using boxplots. This was investigated because outliers can affect the overall performance of the model as the model is being tested and trained with unusual data that should not be in the dataset This insight will lead me to apply future techniques to remove these outliers.

8 features were found to have no correlation to the target class that was researched for EDA question (4) using kdeplots. This was investigated because these features make it harder for the model to differentiate and correctly predict each class as the data for each observation within the feature is too similar or the same. This insight will lead me to remove these features.

8 unique highly positive correlated features were found that were researched for EDA question (5) using a heatmap. This was investigated because highly positively correlated features can cause multicollinearity and overfitting issues. However, features didn't affect many others, and some were only within their subgroup. This insight will lead me to not remove any features, as again the features are not that heavily influenced by others such as Z with U and G, obj_ID with run_ID, and more.

For EDA question (6) it was found that for galaxy and quasar, there are more observations at the high end of the redshift wavelength spectrum, while for star there are more observations at the low end. For EDA question (7) it was found that the farther away the observation from us is the larger the redshift, so the closer the observation is from us the lower the redshift. EDA question (6) and (7) was investigated using a boxplot to get better insight and understanding of redshift and its relationship with the target class and draw out real-world conclusions.

Also, 1 recurring value was found throughout the research of EDA questions (3) and (5). This will be removed as the data is the same for every observation so it will not help to train the model.

Features found with no correlation to the target class –

- obj_ID
- alpha
- delta
- reun_ID
- rerun_ID
- cam_col
- field_ID
- fiber_ID

Features found containing outliers –

- obj_ID
- u
- g
- r
- i
- z
- field_ID
- redshift

Features found with recurring values –

- rereun_ID

features found that are highly positively correlated with others –

- run_ID with obj_ID
- u with z
- g with z
- r with i
- g with u
- spec_obj_ID with MJD
- spec_obj_ID with plate
- plate with MJD

3 - Experimental Design

3.1 - Identification of your chosen supervised learning algorithm(s)

The logistic regression algorithm is used to solve classification problems and it works similarly to linear regression that's used to solve regression problems, but it instead predicts a class instead of an estimated value. Logistic regression works by computing the sum of the input features that calculates the logistic result that determines the predicted output, each class will have its own unique mathematical range and so the prediction will be whatever range the logistic result falls on. There are many types of logistic regression, and I will use the multinomial type as my dataset is being used for a supervised multi-class classification problem. The regular type is binary logistic regression which only supports two class objects.

The Multinomial Logistic regression algorithm is appropriate for my problem as it is used to solve the same type of problem, which is a supervised multiclass classification problem. It's stated that logistic regression handles large datasets efficiently which is perfect as our dataset has 100,000 observations. Multinomial logistic regression can also handle numerical continuous data like within our dataset.

Strengths –

- Easier to implement and understand.
- Can easily extend to multiple classes (multinomial type for multi-classification).
- Handles large datasets efficiently.

Weaknesses –

- Doesn't work if the number of observations is lesser than the number of features.
- Non-linear problems cannot be solved using logistic regression.
- Doesn't work well with correlated features.

3.2 - Identification of appropriate evaluation techniques

Many evaluation metrics will be used to give the team an overview of our model's performance, and for comparison to see whose algorithm is the overall most efficient and whenever they need further improvements. These specific evaluation metrics were chosen as they are popular and common metrics used in machine learning, and they can be easily comparable as they are percentages. We will use many evaluation metrics as they provide accuracies of our models in different ways.

Accuracy -

The accuracy metric provides the percentage/decimal of correctly classified predictions to the total number of predictions. So, if the model's accuracy was 85%, then 85% of the predictions made were correct and 15% were incorrect. It gives us a good insight into how correctly our model predicts results. This metric was specifically chosen because it was commonly used, and easy to use and understand. The `accuracy_score` function from the `skit-learn` library will be used for this.

$$\text{Accuracy} = (\text{True Positives} + \text{True Negatives}) / \text{Total number of predictions}$$

Precision -

The precision metric provides the percentage/decimal of how accurate positive predictions the model made. This metric was specifically chosen because it helps to identify the proportion of true positive predictions made by the model out of all positive predictions. It is another commonly used metric to view the performance of classification models. The `precision_score` function from the `skit-learn` library will be used for this.

$$\text{Precision} = (\text{True Positives} / (\text{True Positives} + \text{False Positives}))$$

Recall -

The recall metric provides the percentage/decimal of correctly classified positive instances out of the total number of actual positive instances. This metric was specifically chosen because it helps to identify all the true positive cases among all the positive instances in the dataset, even if it leads to more false positives. It is another commonly used metric to view the performance of classification models. The `recall_score` function from the `skit-learn` library will be used for this.

$$\text{Recall} = (\text{True Positives} / (\text{True Positives} + \text{False Negatives}))$$

F1 score -

The F1 score metric provides the percentage/decimal of correctly classified positive instances true positives out of all positive instances true positives and false negatives considering both precision and recall. This metric was specifically chosen because it should be used along with recall and precision, and it helps to balance the trade-off between precision and recall. It is another commonly used metric to view the performance of classification models. The `f1_score` function from the `skit-learn` library will be used for this.

$$\text{F1 Score} = 2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$$

3.3 Data cleaning and Pre-processing transformations

3.3.1 – Splitting New Dataset

```

1 # i split a fresh dataset for the pre-processing, cleaning and modeling of my algorithm.
2 # the new split is the exact same as the EDA split.
3
4 # separated the features from the dataset so i can preform a train/test split
5 X = df.drop(['class'], axis=1) # FEATURES WITHOT LABEL
6 Y = df['class'] # LABEL WITHOUT FEATURES
7
8 # 80/20 split between training and testing split
9 # 42 random state
10 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 42)

```

Figure 49 seperating input features to target variable

Splitting a new dataset for this phase to avoid data leakage. The same split was done here as in the EDA. 80/20 - training/testing split with 42 random state.

```

1 # shows the dataset size for each train/test split
2
3 # training set has 80,000 observations for features and class label
4 # testing set has 20,000 observations for features and class label
5 print(f"X_train -\nShape: {X_train.shape}\n\nX_test -\nShape: {X_test.shape}\n")
6
7 print(f"Y_train -\nShape: {Y_train.shape}\n\nY_test -\nShape: {Y_test.shape}\n")

```

X_train -
Shape: (80000, 17)

X_test -
Shape: (20000, 17)

Y_train -
Shape: (80000,)

Y_test -
Shape: (20000,)

Figure 50 new test and train set shape

The new dataset shape after split for the training set is 80,0000 and for testing its 20,0000.

```

1 # merged the training and testing features to their coresponding class label for the data cleaning and pre-processin, so both will be effected
2 df_train = pd.merge(X_train, Y_train, left_index=True, right_index=True)
3
4 df_test = pd.merge(X_test, Y_test, left_index=True, right_index=True)

```

Figure 51 merging training set and testing set

I merged the features and target vairble back together so it's a whole dataset again, as pre-processing and data cleaning will need to effect both of them. This was done for both training and testing set.

3.3.2 – Dropping Features (Feature Selection)

For feature selection, unwanted features will be dropped from the dataset, being ones with recurring values and no correlation to the target class. It was found throughout the univariate and multivariate analysis of EDA question (4) and question (5) using kdeplots and a heatmap, that some features had no correlation to the target class, and obj_ID had recurring values.

These will be dropped as they provide no use when training the model as data is recurring, so all the data is the same throughout each observation, and some features have no correlation to the target class which makes it harder for the model to differentiate and correctly predict each class because the data for each observation within the feature is too similar or the same. This can cause multicollinearity and overfitting issues.

This was done using the drop function that panda's library provides. This should be for this for both the training and testing set as they should have the same features so the model can be efficiently trained, and they require to have the same amount of features for them to fit into the model. This particular method was chosen as it's simple and most commonly used for this process.

```
1 # all the highly correlated features and features with recurring values found throughout the EDA univariate and multivariate analysis where dropped
2 # this was done for both the training and testing set as they both need the same features for modeling
3
4 # a total of 8 features where dropped
5 df_train = df_train.drop(['obj_ID', 'alpha', 'delta', 'run_ID', 'rerun_ID', 'cam_col', 'field_ID', 'fiber_ID'], axis = 1)
6 df_test = df_test.drop(['obj_ID', 'alpha', 'delta', 'run_ID', 'rerun_ID', 'cam_col', 'field_ID', 'fiber_ID'], axis = 1)
```

Figure 52 dropping unwanted features

Process of removing all unwanted features what was determined in the EDA.

```
1 # shows the new size of total observations and features within the testing and training set after correlated features where dropped
2 print(f"df_train -\nShape: {df_train.shape}\n")
3
4 print(f"df_test -\nShape: {df_test.shape}\n")

df_train -
Shape: (80000, 10)

df_test -
Shape: (20000, 10)
```

Figure 53 new test and train set shape

New training and test set size with the wanted features, only 10 features remain as 8 were removed.

```
1 # shows how the new dataset looks after correlated features where dropped
2 df_train
```

	u	g	r	i	z	spec_obj_ID	redshift	plate	MJD	class
75220	22.32247	21.40113	20.51302	19.62691	19.34791	5.136368e+18	0.659869	4562	55570	GALAXY
48955	25.85486	24.22573	21.48514	20.38426	19.38826	8.709046e+18	0.813819	7735	58136	GALAXY
44966	22.62750	21.41766	19.54710	18.87770	18.39861	5.986547e+18	0.404277	5317	56000	GALAXY
13568	20.25937	18.53387	17.58141	17.16960	16.85448	2.200026e+18	0.108840	1954	53357	GALAXY
92727	22.03892	21.46501	21.32925	21.26120	21.17124	4.340534e+18	-0.001039	3855	55268	STAR
...
6265	20.82854	20.32382	20.25718	20.39426	20.36549	1.208439e+19	2.412806	10733	58244	QSO
54886	18.93209	16.96611	16.05940	15.64639	15.30356	1.140565e+18	0.081128	1013	52707	GALAXY
76820	23.95684	21.37612	19.75149	18.49137	17.77083	3.672834e+18	0.000253	3262	54884	STAR
860	17.79224	16.47265	15.84970	15.59104	15.46414	3.016365e+18	0.000000	2679	54368	GALAXY
15795	24.93142	21.08619	19.28560	18.66209	18.36053	4.471072e+18	0.352529	3971	55322	GALAXY

80000 rows x 10 columns

Figure 54 new dataset after removing features

New training set looks like this without the unwanted features. Testing set is the same but contains less observations.

3.3.3 - Removing Outliers (Data Cleaning)

For the data cleaning process, outliers within the dataset will be detected and removed. It was found throughout the univariate analysis in EDA question (3) using boxplots, that many features contained outliers.

Outliers need to be removed because they are occurrences of unusual data such as extremely high or low values that can affect the overall performance of the model, as the model is being tested and trained with unusual data that is different from the majority of data within that feature.

The IQR technique will be used for this process with a function that takes the dataset as a parameter, and a for loop that iterates through every feature within the dataset. IQR involves identifying and removing values that fall outside the region where the majority of the data is found. This is done by using the first and third quartiles (Q1 and Q3) of the data set to compute the interquartile range (IQR) ($Q3 - Q1$). The dataset is divided into four equal portions using the 25th and 75th percentiles. A value is considered an outlier if it is less than ($Q1 - 1.5 * IQR$) or greater than ($Q3 + 1.5 * IQR$), as per the standard rule that values that are more than 1.5 times the IQR outside the quartiles are outliers. It was programmed so it would remove the whole observation containing an outlier, so the dataset will contain no missing values.

This should be for both the training and testing set, as they should have the same distribution of data without any unusual occurrences that will harm the performance. This particular technique was used because it was easy to calculate and understand. The dataset is very large, so it potentially has a lot of outliers. This technique is not affected by outliers or extreme values in the data, so it can accurately identify outliers without being influenced by the occurrence of outliers or extreme values.

```

1  # IQR method used to detect remove outliers as it was discorved that many feartures had them throughout the EDA univariate analysis
2  # this was done for both the training and testing set as they both cannot have outliers
3
4  # this varibale is used for the later before_outlier_removal() function to show feature before removing outliers
5  dataset_before_removal = df_train.copy(deep=True)
6
7  def outlier_removal(dataset):
8
9      # dataset before removal
10     shape_before = dataset.shape
11
12     for feature in dataset.select_dtypes(include = 'number').columns:
13         # 0.25 and 0.75 quantile used to later calculate lower and upper bound
14         # 0.25 and 0.75 used for IQR method
15         QT1 = dataset[feature].quantile(0.25)
16         QT3 = dataset[feature].quantile(0.75)
17
18         IQR = abs(QT3 - QT1)
19
20         # to calculate upper and lower bound
21         lower = QT1 - (1.5 * IQR)
22         upper = QT3 + (1.5 * IQR)
23
24         # anything over the upper and lower bound will be detected as an outlier
25         min_in = dataset[dataset[feature] < lower].index
26         max_in = dataset[dataset[feature] > upper].index
27
28         # detected outliers outside lower and higher bound are dropped from the dataset
29         dataset.drop(min_in, inplace = True)
30         dataset.drop(max_in, inplace = True)
31
32     # dataset after removal
33     shape_after = dataset.shape
34
35     # caluclated different before and after outlier removal to get the sum of outliers removed
36     outlier_count = abs(shape_before[0] - shape_after[0])
37
38     return outlier_count

```

Figure 55 IQR code to removal outliers

Code for the IQR method of detecting and removing outliers. Its in a function with a for loop that loops through every feature in the dataset that's given as a parameter.

```

1 # shows numbers of outliers removed within the training set
2 print("Number of outliers deleted in training set: ", outlier_removal(df_train))
3
4 # shows numbers of outliers removed within the training set
5 print("Number of outliers deleted in testing set: ", outlier_removal(df_test))

Number of outliers deleted in training set: 7487
Number of outliers deleted in testing set: 1922

```

Figure 56 total outliers removed for each dataset

Outliers were removed from both the training set and testing set. 7487 outliers were removed from the training set, and 1922 outliers from the testing set.

- outlier removal comparison (r)

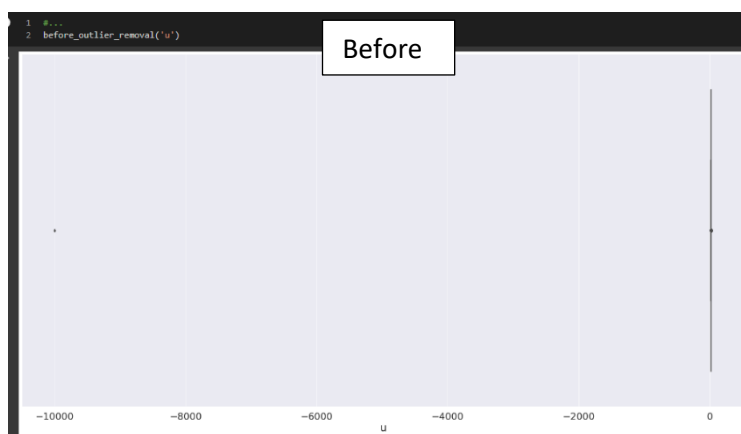


Figure 58 u before outlier removal

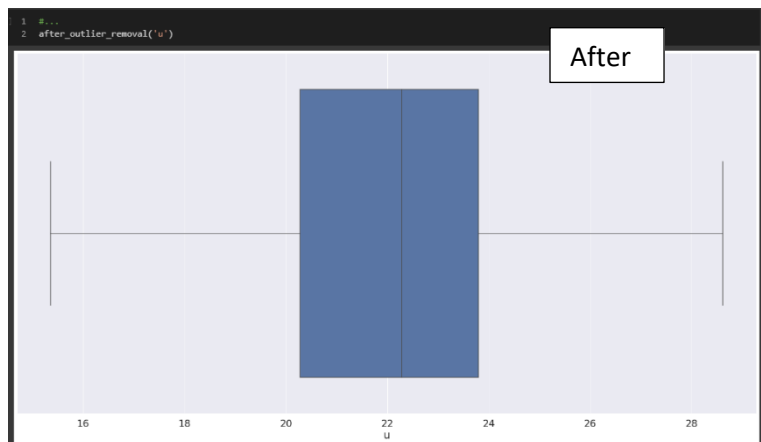


Figure 58 u after outlier removal

Can see that every outlier was removed from the u feature as only the middle 50% is showing.

- outlier removal comparison (g)

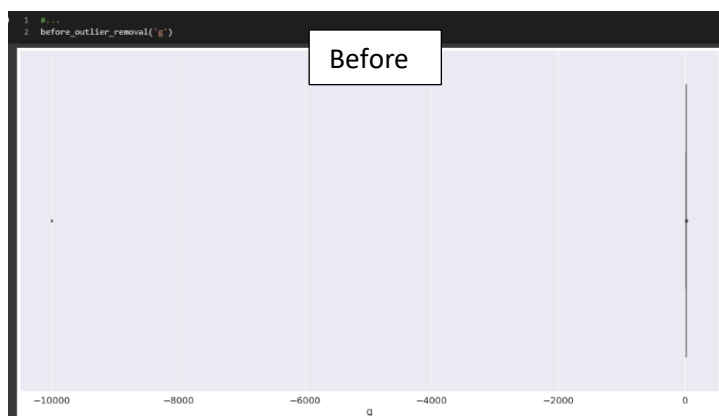


Figure 60 g before outlier removal

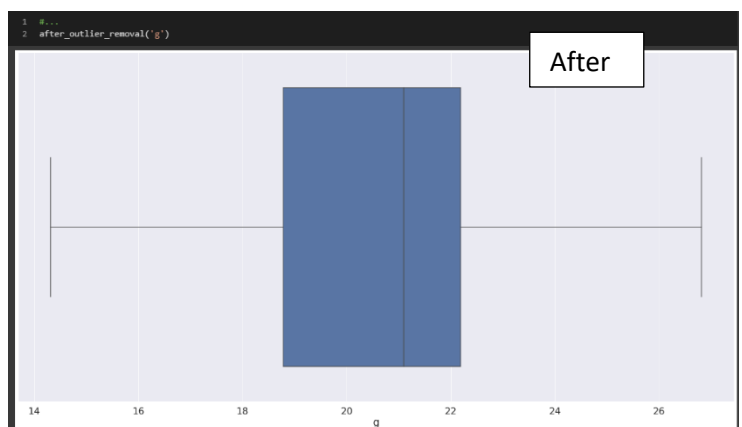


Figure 60 g after outlier removal

Can see that every outlier was removed from the g feature as only the middle 50% is showing.

- outlier removal comparison (f)

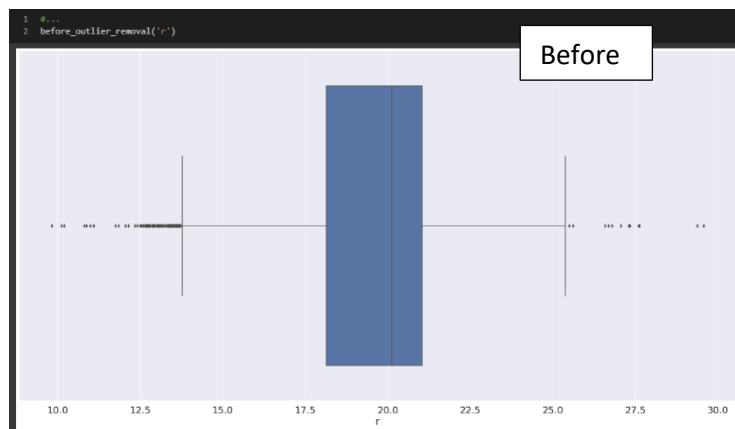


Figure 62 f before outlier removal

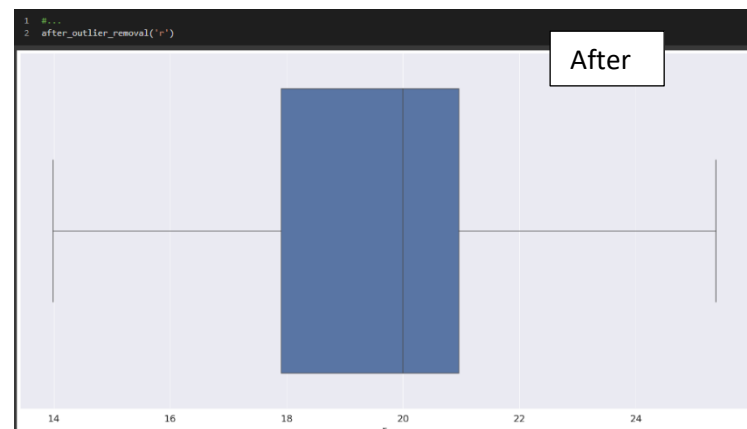


Figure 62 r after outlier removal

Can see that every outlier was removed from the r feature as only the middle 50% is showing.

- outlier removal comparison (i)

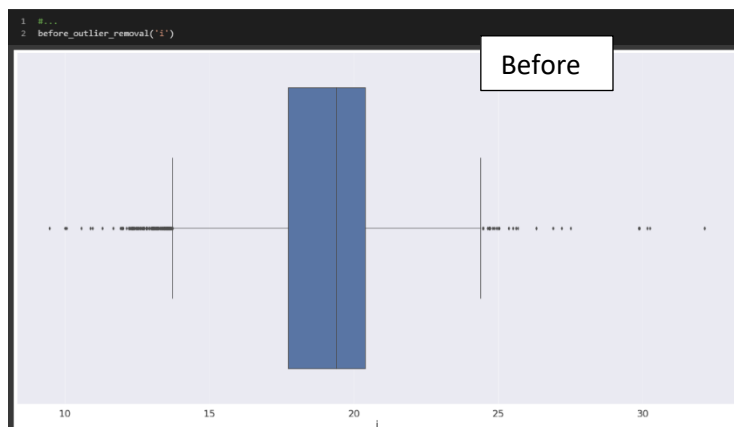


Figure 64 i before outlier removal

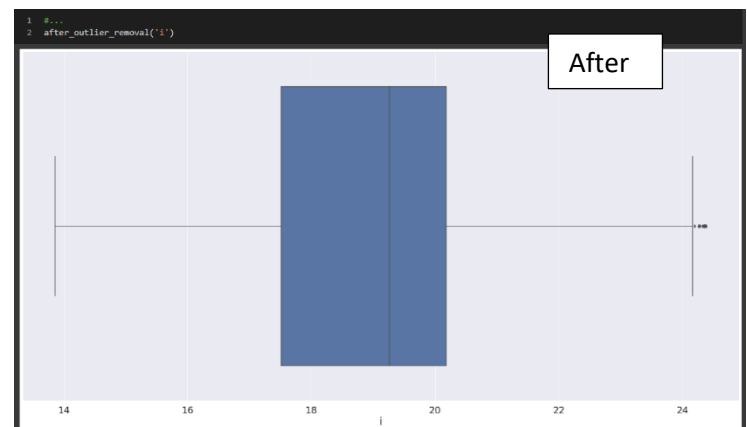


Figure 64 i after outlier removal

A few outliers still remained in feature i after the removal process. This mean means that these values are not considered outliers according to this method.

- outlier removal comparison (z)

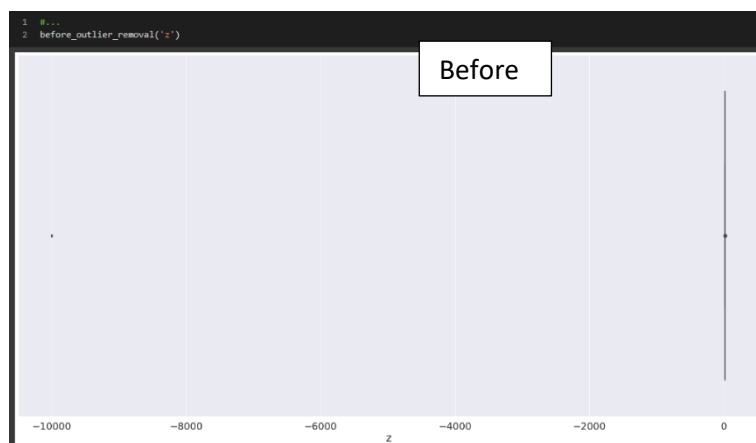


Figure 66 z before outlier removal

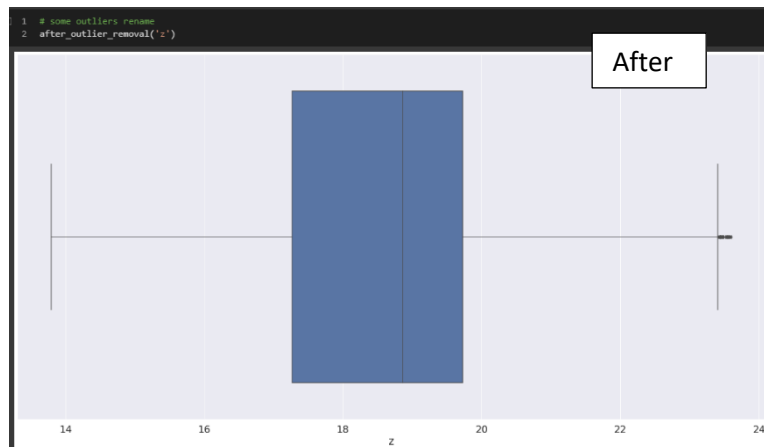


Figure 66 z after outlier removal

A few outliers still remained in feature z after the removal process. This mean means that these values are not considered outliers according to this method.

- outlier removal comparison (spec_obj_id)

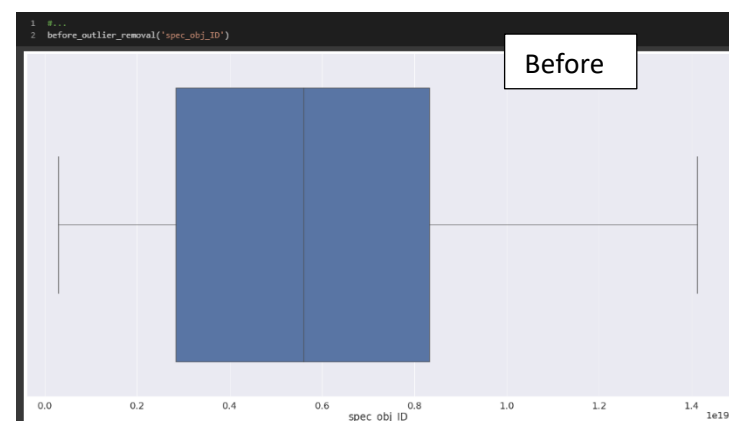


Figure 68 spec_obj before outlier removal

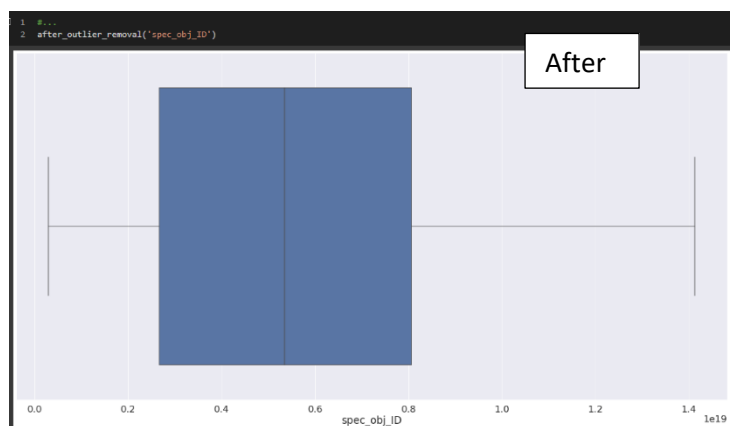


Figure 68 spec_obj after outlier removal

Feature spec_obj_id was unaffected as it originally didn't have any outliers.

- outlier removal comparison (redshift)

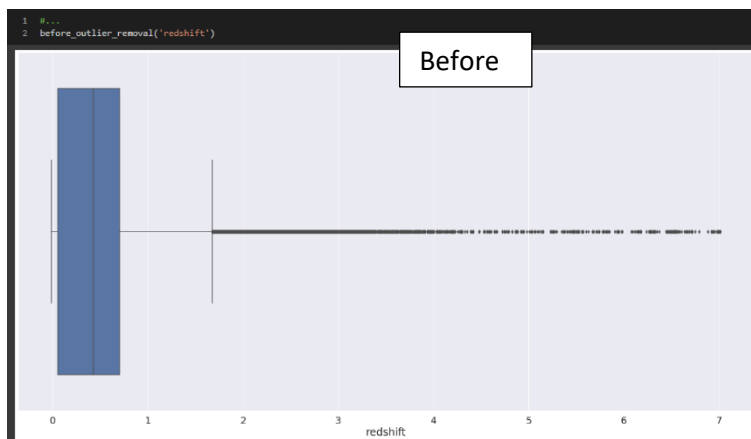


Figure 70 redshift before outlier removal

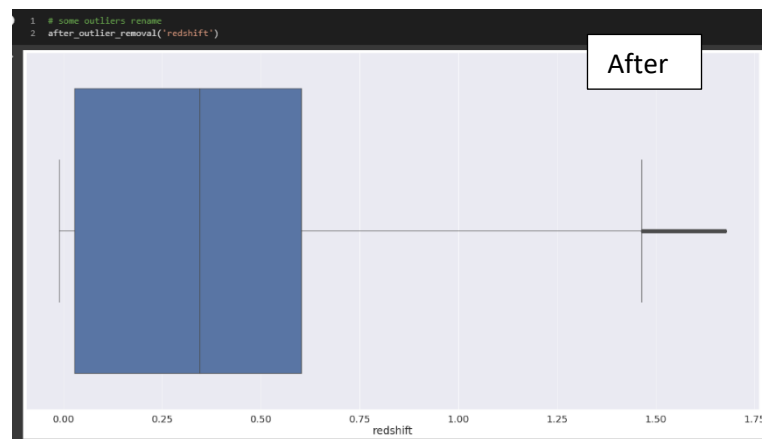


Figure 70 redshift after outlier removal

Many outliers still remained in feature redshift the removal process. This mean means that these values are not considered outliers according to this method.

- outlier removal comparison (plate)

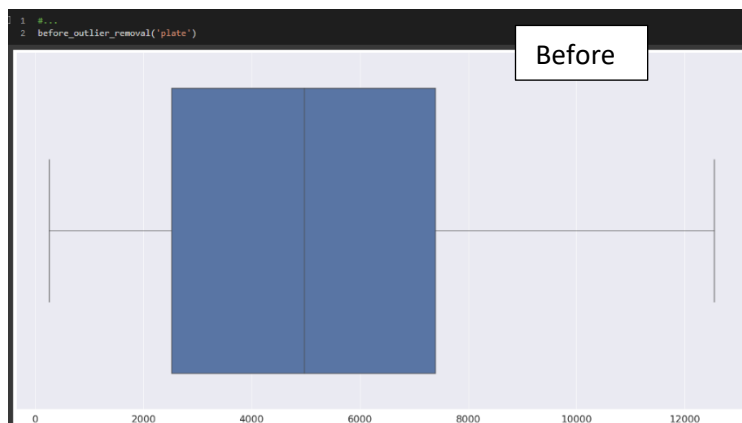


Figure 72 plate before outlier removal

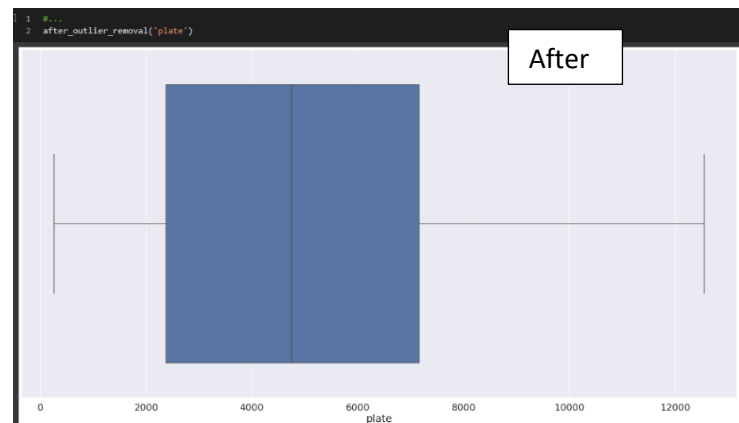


Figure 72 plate after outlier removal

Feature plate was unaffected as it originally didn't have any outliers.

- outlier removal comparison (MJD)

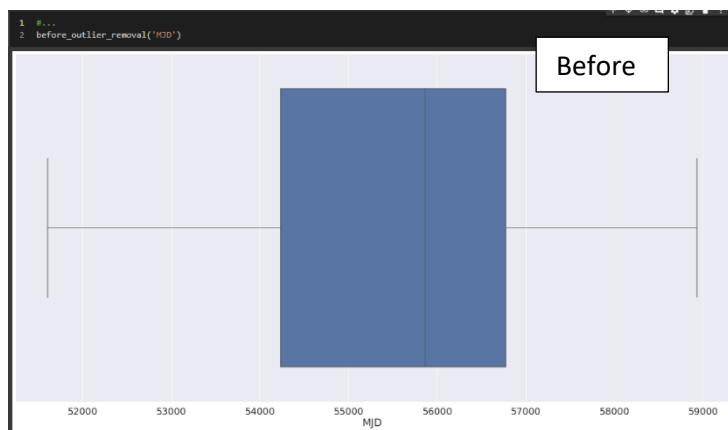


Figure 74 MJD before outlier removal

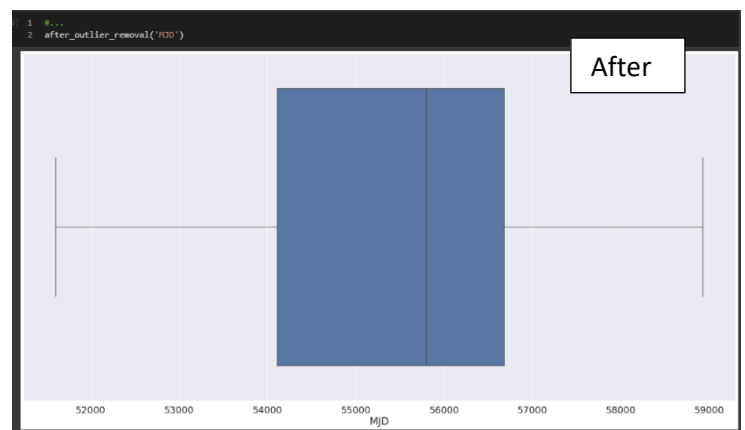


Figure 74 MJD after outlier removal

Feature MJD was unaffected as it originally didn't have any outliers.

3.3.4 - Dealing with Missing Values (Data Cleaning)

For the data cleaning process, it was initially checked if our dataset had any missing NULL values throughout the mutilative analysis in EDA question (1) using a mnso graph. And the dataset didn't have any initial missing values which were good, as the model cannot be efficiently trained or tested with a dataset with important missing data. Though it did have 326 occurrences in the training set and 86 occurrences in the testing set of the values being 0's that should be removed, as the research team thinks 0s in our dataset don't make much sense. Because we had some occurrences of redshift being zero that would mean the object is not moving away from or towards the observer, but objects should always be moving as space is expanding.

However, the removal of outliers can cause missing values, so it will be again checked using a mnso graph, on both the training and testing set. There shouldn't be any missing values after removing outliers, as our IQR function automatically removes observations with an outlier.

Techniques like imputation were not originally used here as they affect the variability and distribution of the data. And there were still overall enough observations for them to be dropped to not affect the training and testing process.



Figure 75 mnso graph showing no missing values

Graph shows there is no NULL values in the training set as all the bars are grey for each feature.

```

1 # no missing values were found after the IQR method on the training set as it removes the whole observation containing an outlier
2
3 # view the number of missing values for each feature within the training set
4 df_train.isnull().sum()

```

```

u          0
g          0
r          0
i          0
z          0
spec_obj_ID 0
redshift    0
plate       0
MOD         0
class       0
dtype: int64

```

Figure 76 shows all features have 0 null values in training set

Can see there are 0 missing NULL values for every feature on the training set, using the `isnull().sum()` function that displays the total of missing NULL values.

```

1 df_train.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 72513 entries, 75220 to 15795
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  -
0   u                72513 non-null  float64
1   g                72513 non-null  float64
2   r                72513 non-null  float64
3   i                72513 non-null  float64
4   z                72513 non-null  float64
5   spec_obj_ID      72513 non-null  float64
6   redshift         72513 non-null  float64
7   plate            72513 non-null  int64
8   MJD              72513 non-null  int64
9   class            72513 non-null  object
dtypes: float64(7), int64(2), object(1)
memory usage: 8.1+ MB

```

Figure 77 shows all datatypes for each feature

Shows that every feature in the training set is non-null, meaning they cannot be null. So no null values will be found.

```

1 # counts the amount of values that are 0 in the dataset, these must also be
2 (df_train == 0).sum().sum()

323

```

Figure 78 shows total number of 0s in training set

Shows that there is 323 values that are zero in the training set.

```

1 ## this removes all observations that have 0 in it
2 df_train = df_train[(df_train != 0).all(1)]
3
4 # can see now there are no values that are 0
5 (df_train == 0).sum().sum()

0

```

Figure 79 process of removing all 0's as we don't want them in training set

We want to remove the occurrences of 0's in our dataset as 0's, as the research team think 0s in our dataset doesn't make much sense. Because we had some occurrences of redshift being zero that which would mean the object is not moving away from or towards the observer, but objects should always be moving as space is expanding.

This code removes the whole observation containing the value 0 within the training set.



Figure 80 msno graph showing no missing values

Graph shows there are no NULL values in the testing set as all the bars are grey for each feature.

```

1 # no missing values were found after the IQR method on the testing set as it removes the whole observation containing an outlier
2
3 # view the number of missing values for each feature within the testing set
4 df_test.isnull().sum()

```

```

u          0
g          0
r          0
i          0
z          0
spec_obj_ID 0
redshift    0
plate       0
MJD         0
class       0
dtype: int64

```

Figure 81 shows all features have 0 null values in testing set

Can see there are 0 missing NULL values for every feature on the testing set, using the `isnull().sum()` function that displays the total of missing NULL values.

```

1 # shows all features are int/float datatype
2 # wont have any blank spaces as all datatypes are numbers
3 df_test.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 72513 entries, 75220 to 15795
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  -
0   u                72513 non-null  float64
1   g                72513 non-null  float64
2   r                72513 non-null  float64
3   i                72513 non-null  float64
4   z                72513 non-null  float64
5   spec_obj_ID      72513 non-null  float64
6   redshift         72513 non-null  float64
7   plate            72513 non-null  int64
8   MJD              72513 non-null  int64
9   class            72513 non-null  object
dtypes: float64(7), int64(2), object(1)
memory usage: 8.1+ MB

```

Figure 82 shows all datatypes for each feature

Shows that every feature in the testing set is non-null.

```

1 # counts the amount of values that are 0 in the dataset, these must also be removed
2 (df_test == 0).sum().sum()

86

```

Figure 83 shows total number of 0s in testing set

Shows that there is 86 values that our zero in the testing set.

```

1 # this removes all observations that have 0 in it
2 df_test = df_test[(df_test != 0).all(1)]
3
4 # can see now there are no values that are 0
5 (df_test == 0).sum().sum()

0

```

Figure 84 process of removing all 0's as we don't want them in testing set

0's were also removed from the testing set. This removes the whole observation containing the value 0.

3.3.5 – Target Encoding (Data Encoding)

For data encoding, we decided to encode the target variable that was initially categorical as objects, being GALAXY, STAR, and QSO. Through research, we found that encoding the target variable is necessary for continuous datasets like ours, which will help the model better understand the unique relationship of the class to the data. So it will improve the performance of the model.

This was done using the LabelEncoder function that scikit-learn library provides. It works by converting every object to a unique integer value. So, our classes [GALAXY, STAR, QSO] will become [0, 1, 2] or similar. There will now be three index values as there are three classes.

This should be for this for both the training and testing set for consistency which is necessary for the model to be properly trained and tested. This particular method was chosen because it's commonly used in multi-class classification problems. Converting class labels to integers can save memory when working with large datasets which are good as our dataset is very large with over 100,000 observations before pre-processing and cleaning, so they may make out models run faster.

```
1 encoder = LabelEncoder()
2
3 df_train['class'] = encoder.fit_transform(df_train['class'])
```

Figure 85 encoding class column in training set

Used the LabelEncoder encoder variable and fitted it with the class column on the training set to encode the target class.

```
1 df_test['class'] = encoder.fit_transform(df_test['class'])
```

Figure 86 encoding class column in testing set

Also used to encoder variable and fitted it with the testing set class column to encode it.

```
] 1 df_train['class']
75220    0
48955    0
44966    0
13568    0
92727    2
..
37194    0
82386    1
54886    0
76820    2
15795    0
Name: class, Length: 72190, dtype: int64
```

Figure 87 shows the class column is now encoded

```
1 df_train['class'].unique()
array([0, 2, 1])
```

Figure 88 all classes after data encoding

Can see the data is now encodes in 0s, 1s and 2s. [GALAXY, QSO, STAR] is now [0,2,1].

3.3.6 - Normalization (Data Scaling)

For the data scaling process, I decided to normalize the dataset because it has numerical continuous values with ranges very large, making them very hard to read and analyse. Most of the values in the dataset are measurements and normalization can help to bring it to a common scale and make it easier to compare the contribution of the features to the model. It's beneficial to normalize logistic regression for multi-class classification problems as it's sensitive to features with a large range or range, so by normalizing the dataset, the model can avoid being overly sensitive to the features like these.

This was done using the MinMaxScaler function the scikit-learn library provides. It works by scaling the data so that it falls between 0 and 1. This is often done by subtracting the minimum value from each value and then dividing it by the range of the data. This ensures that all the values of the feature will fall between 0 and 1, with 0 being the minimum value and 1 being the maximum value.

This should be done for both the testing and training set, as they need to be scaled down to the same range for the results to be more comparable, consistent, and not biased. Not scaling both would mean the values in each set are completely different.

I decided to use this technique over standardization that's also used for scaling because it works better with very large values and scales down the values to a smaller range. MinMaxScaler only depends on the minimum and maximum values and so is not affected by large values as much as StandardScaler which uses mean and variance. MinMaxScaler is commonly used for numerical continuous like mine.

```

1 # seperate target class and features from the training and testing set to perform normalization just onto the features
2
3 # training set
4 X_train = df_train.drop(["class"], axis = 1)
5 Y_train = df_train["class"]
6
7 # testing set
8 X_test = df_test.drop(["class"], axis = 1)
9 Y_test = df_test["class"]

```

Figure 89 separating target class to input features for both training and testing set

For this process I had to separate the features and target variable from each training and testing set.

```

1 # decided to normalize the data using MinMaxScaler to remove unwanted noise from dataset like duplicated observations and redundancies.
2 # this was done for both the testing and training set as both need to be normalised for them to be fit into the model
3
4 # initialize scaler variable for MinMaxScaler function
5 scaler = MinMaxScaler()
6 # fit with training to estimate the minimum and maximum observable values for when training and testing set are transformed
7 scaler.fit(X_train)
8
9 # normalizing training set
10 X_train = scaler.transform(X_train)
11 X_train = pd.DataFrame(X_train)
12
13 # normalizing testing set
14 X_test = scaler.transform(X_test)
15 X_test = pd.DataFrame(X_test)

```

Figure 90 normalization code

X_train was fit with the MinMaxScaler and then X_train and X_test was transformed using the scaler, so the data is scaled the same.


```

1 # shows how the new dataset looks after normalization
2 X_train

```

	0	1	2	3	4	5	6	7	8
0	0.525182	0.566664	0.572816	0.547707	0.566357	0.349798	0.397045	0.349809	0.540961
1	0.791571	0.792652	0.658188	0.619662	0.570472	0.608176	0.488298	0.608175	0.891316
2	0.548185	0.567986	0.487989	0.476525	0.469560	0.411284	0.245544	0.411286	0.599672
3	0.369597	0.337263	0.315361	0.314238	0.312110	0.137441	0.070425	0.137448	0.238804
4	0.503799	0.571775	0.644498	0.702980	0.752277	0.292243	0.005294	0.292240	0.499727
...
72185	0.404545	0.350163	0.318037	0.311756	0.299273	0.091933	0.082446	0.091931	0.166166
72186	0.494857	0.471646	0.517781	0.707873	0.682057	0.384420	0.560741	0.384415	0.564992
72187	0.269502	0.211831	0.181697	0.169518	0.153968	0.060820	0.053998	0.060826	0.150055
72188	0.648435	0.564663	0.505938	0.439819	0.405548	0.243955	0.006060	0.243954	0.447297
72189	0.721932	0.541466	0.465024	0.456039	0.465677	0.301684	0.214870	0.301686	0.507100

72190 rows × 9 columns

Figure 91 dataset after normalization

How the dataset now looks after its normalized. Data is not scaled from 0 to 1. Data is much easier to read also.

3.3.7 - Balancing Dataset (Data Transformation)

For the data transformation process, the dataset was balanced so every target class had the same amount of observations. It was found throughout the urinative analysis in EDA question (3) using pie charts and count plots, that the dataset is unbalanced.

The dataset should be balanced because unbalanced data will be more biased towards the class with more data, affecting performance and overall accuracy as the majority class will be way more accurate than the minority class. Each stellar class should have the same amount of corresponding observations.

This was done using the RandomUnderSampling function that sckit-learn library provides. It works by randomly eliminating some data from the over-represented class, which in my case is the galaxy and quasar class. This will be done until every class has the same amount of observations as the minority class being the star class.

This should be for both the training and testing set, as they should have the same distribution of observations for each class, so the model can be evenly trained and tested to not give biased results. This particular technique was used because it can help reduce overfitting since it reduces the amount of data that the model has to fit.

```

1 # throughout the EDA univariate analysis, it was found that dataset is indeed unbalanced.
2 # under sampling was used for this so every class will have the same amount as the class with the least sum of observations
3 # which was the QSO class after data scaling and cleaning.
4 # this was done for both the training and testing set to make the model less bias to a class with more observations
5
6 # rus variable to initalize RandomUnderSampler function
7 rus = RandomUnderSampler()
8
9 # process of balancing training set
10 X_train, Y_train = rus.fit_resample(X_train, Y_train)

```

Figure 92 process of under sampling data on training set

Used RandomUnderSampler on the training set by giving X_train and Y_train as parameters.

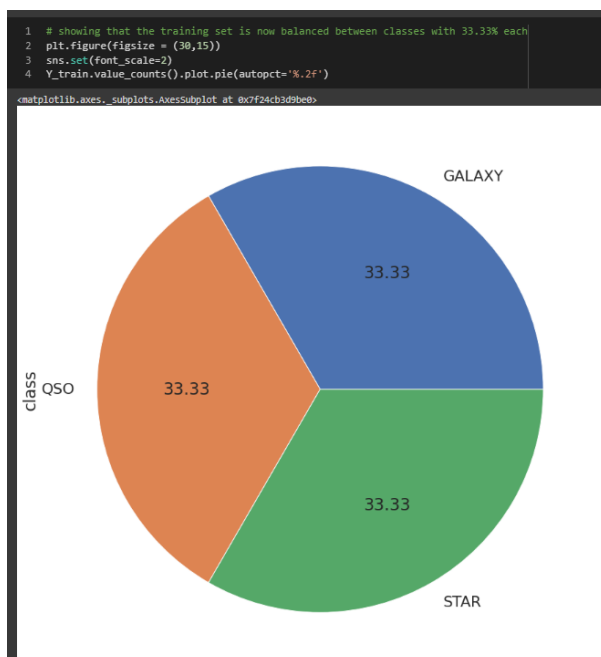


Figure 93 graph showing data is now even after balancing

Graph shows all classes on the training set are now event balanced by 33.33% each after under sampling.

```

1 # showing the balanced sum of observations for each class object for training set
2 # each class has 7995 observations after under sampling
3 Y_train.value_counts()

GALAXY    7995
QSO        7995
STAR       7995
Name: class, dtype: int64

```

Figure 94 new class total on training set after under sampling

Each class will now have 7996 corresponding observations after under sampling on the training set.

```

1 # process of balancing testing set
2
3 X_test, Y_test = rus.fit_resample(X_test, Y_test)

```

Figure 95 process of under sampling data on testing set

Used RandomUnderSampler on the training set by giving X_test and Y_test as parameters.

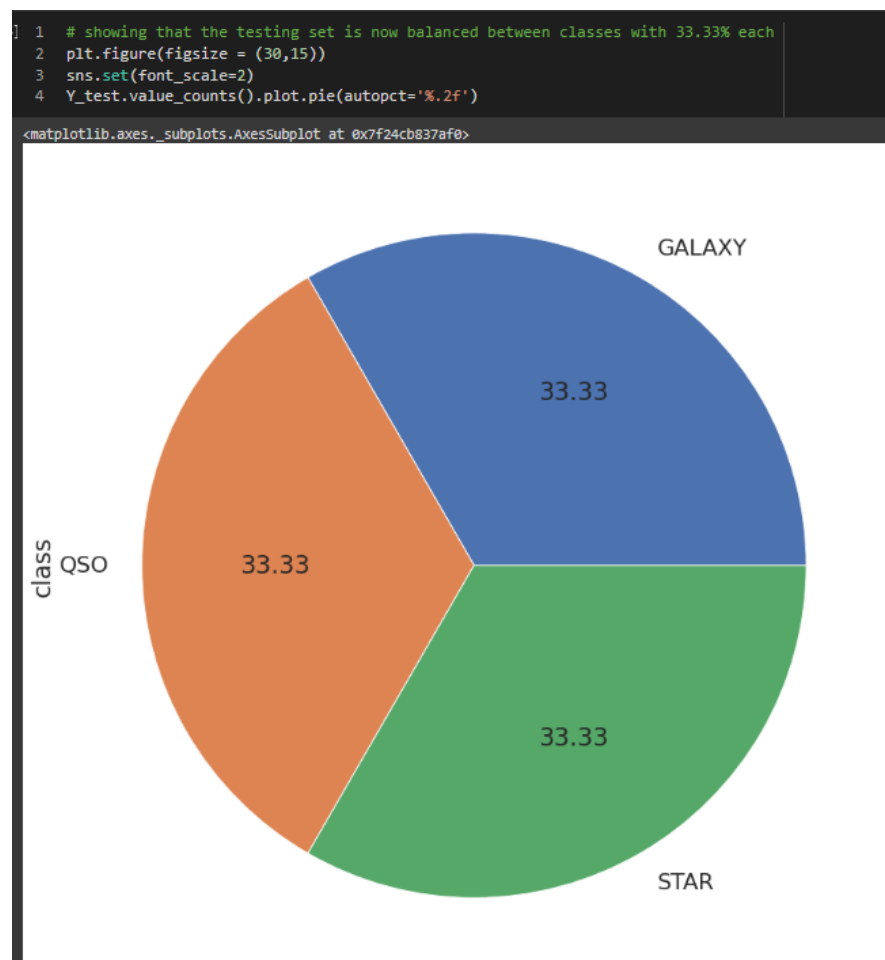


Figure 96 graph showing data is now even after balancing

Graph shows all classes on the testing set are now event balanced by 33.33% each after under sampling.

```
1 # showing the balanced sum of observations for each class object for testing set
2 # each class has 1977 observations after under sampling
3 Y_test.value_counts()

GALAXY    1977
QSO       1977
STAR      1977
Name: class, dtype: int64
```

Figure 97 new class total on testing set after under sampling

Each class will now have 7996 corresponding observations after under sampling on the testing set.

3.4 - Limitations and Options

The limitations of IQR for the removal of outliers is that it may not be able to detect and remove every single outlier which happened in our case for 3 of the dataset's features, being redshift, z, i. It also can be sensitive to the occurrence of skewed data and not remove all of its outliers, which happened in our case with the redshift feature not getting all its outliers removed as its positively skewed. Because of this, our model may have bad performance as unwanted outliers remain.

The limitations of normalization using MinMaxScaler is that its sensitive to outliers in the data and could potentially alter the rest of the data when trying to scale them. This may be an issue as a few outliers in our dataset remain. It's not always appropriate for all types of data that is already in a meaningful range, such as with basic measurements like temperature in Celsius. Many features within our dataset are used for measurements, such as alpha, delta, and redshift. So, this may harm our dataset and make its performance bad.

The limitations of balancing using RandomUnderSampling is that it could discard potentially useful data, as every class has the same amount of observations as the minority class. So, our models may perform worse as many observations are lost that could have been used the further train and test the model. Our model is also at risk of overfitting as the model has fewer data to train with.

The limitations of our overall dataset are that it is large so it may contain a lot of noise, and irrelevant and inconsistent data. Also, datasets with a large amount of data and features can lead to the model being overfitted. The dataset now has 29916 observations after data pre-processing and cleaning which is still large. So, redundant data still may remain, and the model is overfitted. Our initial dataset had over 100,000 observations.

4 - Predictive Modelling / Model Development

4.1 - The predictive modeling processes

4.1.1 – Modeling With Training Set (SEEN DATA)

```
[1001] 1 # initialized the Logistic Regression model so the training and testing sets can be later fit
2 # parameter of "multi_class" was set to "ovr" as its the problem type is multi-classification
3 LR = LogisticRegression(multi_class='ovr')
4
5 # model fit with be tested with the training set (SEEN)
6 LR.fit(X_train, Y_train)
7
8 print(f"X_train -\nShape: {X_train.shape}")
9 print(f"Y_train -\nShape: {Y_train.shape}")

X_train -
Shape: (23985, 9)
Y_train -
Shape: (23985,)
```

Figure 98 code for logistic regression and fitting training and testing data for the model SEEN

Now that the training and testing sets are cleaned and pre-processed, it's time to develop and then evaluate my classification model which will be Logistic Regression. To form the base of the model I used the `LogisticRegression()` function the scikit-learn library provides, and the parameter of 'ovr' was given into the function so the model identifies that it is being used for a multi-class classification problem. The training set was divided into X features being `X_train` and Y targets being `Y_train` as they need to be separate so they can fit into the model.

The model was then fit with two parameters, being `X_train` which contains all feature data, and `Y_train` which contains all corresponding target variable data. This was done using the `fit()` function with the model's LR variable. It was trained with 23985 observations so 7995 observations for each class type ($23985 / 3 = 7995$). Only the training set was used here as its purpose is to train the model.

```
1 # returns predicted class for each observation as an array
2 Y_pred = LR.predict(X_train)
3
4 # shows array of predicted classes
5 print(Y_pred)
6 # length of predicted classes
7 print("\nY_pred length:", len(Y_pred))
8
9 # length of X_train and Y_train, to check and show they are the same length as the Y_pred array
10 # 23985 predictions were made
11 print("\nX_train length:", len(X_train))

['GALAXY' 'GALAXY' 'GALAXY' ... 'STAR' 'STAR' 'STAR']

Y_pred length: 23985
X_train length: 23985
```

Figure 99 – Testing model with SEEN

The `predict()` function was used with the model's LR variable to test it using `X_train` as a parameter being the training set, as I first want to see if the model can generalize and predict observations it has already SEEN during the training process. This was done before using UNSEEN testing data as thoracically, UNSEEN should predict just as well as the SEEN data, otherwise the model may have overfitting issues.

The output of the `predict()` function is an array of results that can be later used to visualize the model's accuracy. A total of 23985 predictions were made as there are 23958 observations in the `X_train` training set which was printed out.

4.1.2 – Modeling With Testing Set (UNSEEN DATA)

Additionally, I personally created a new model to fit with the testing data, as I want to compare the results from both SEEN and UNSEEN models to see if the models are overfitted or under fitted, or not.

```

1 # initilized the Logistic Regression model so the training and testing sets can be later fit
2 # a paramater of "multi_class" was set to "ovr" as its the problem type is multi-classification
3 LR = LogisticRegression(multi_class='ovr')
4
5 # model fit with be tested with the testing set (UNSEEN)
6 LR.fit(X_train, Y_train)
7
8 print(f"X_train -\nShape: {X_train.shape}")
9 print(f"Y_train -\nShape: {Y_train.shape}")

```

X_train -
Shape: (23985, 9)
Y_train -
Shape: (23985,)

Figure 100 code for logstic rgression and fitting training and testing data for the model UNSEEN

The same process was done here as the previous model that was fit with the SEEN data.

```

1 # returns predicted class for each observation as an array
2 Y_pred = LR.predict(X_test)
3
4 # shows array of predicted classes
5 print(Y_pred)
6 # length of predicted classes
7 print("\nY_pred length:", len(Y_pred))
8
9 # length of X_test and Y_test, to check and show they are the same length as the Y_pred array
10 # 5931 predictions were made
11 print("\nX_train length:", len(X_test))

```

['STAR' 'GALAXY' 'GALAXY' ... 'STAR' 'STAR' 'STAR']
Y_pred length: 5931
X_train length: 5931

Figure 101 Testing model with SEEN

The same process was done here as the previous model that was fit with the SEEN data.

4.2 - Evaluation results on “seen” data

```

1  # used various evaluation metrics to view the performance of my model
2  # used accuracy, recall, precision, F1 score and cross validation
3  # tested with the training set (SEEN)
4
5  # accuracy
6  print('- Accuracy : %.2f' % (accuracy_score(Y_train, Y_pred)))
7
8  # recall
9  print('\n- Recall : %.2f' % (recall_score(Y_train, Y_pred, average='micro')))
10
11 # precision
12 print('\n- Precision : %.2f' % (precision_score(Y_train, Y_pred, average='micro')))
13
14 # f1 score
15 print('\n- F1 : %.2f' % (f1_score(Y_train, Y_pred, average='micro')))
```

Figure 102 – Code that prints evaluation metrics (accuracy, recall, precision, F1 score)

This shows the code is used to get every evaluation metric result, being accuracy, recall, precision, and f1 score using functions provided in the scikit-learn library.

The following table shows the evaluation metric results of my model tested with SEEN data.

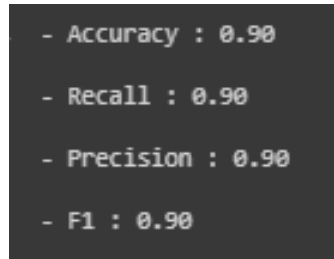
SEEN DATA		
Metric	Multinomial Logistic Regression	Evidence
Average Accuracy	90%	
Average Precision	90%	
Average Recall	90%	
Average F1 Score	90%	

Figure 103 – Metrics Results

My overall result for each evaluation metric was 90% from the model tested with SEEN data which is fairly good so I am happy with it. For accuracy, the model was predicted correctly 90% of the time. For precision, 90% of the positive predictions made by the model were correct. For recall, the model identified 90% of the positive instances. For F1 score, it's 90% as it had a good balance between high precision and high recall which were both 90%.


```

1 # classification report further shows precision, recall, f1-score metric scores with more detail such as macro and weighted average
2 print(classification_report(Y_train, Y_pred))

```

	precision	recall	f1-score	support
GALAXY	0.90	0.80	0.84	7995
QSO	0.90	0.92	0.91	7995
STAR	0.90	0.98	0.94	7995
accuracy			0.90	23985
macro avg	0.90	0.90	0.90	23985
weighted avg	0.90	0.90	0.90	23985

Figure 104 classification report results

This is just like figure 89, but it gives more information like there are 7995 observations for each class. It also shows the individual metric accuracies for each class. Star had the highest F1 score of 94%, and why it was averaged down to 90% as the F1 score for galaxy was 84% and star was 91%.

The macro accuracy is also 90% which calculates the metric independently for each class and then takes the average. And the weighted accuracy is 90% as well which accounts for the class imbalance and calculates the metric considering the class distribution, which does nothing as the training set is balanced.

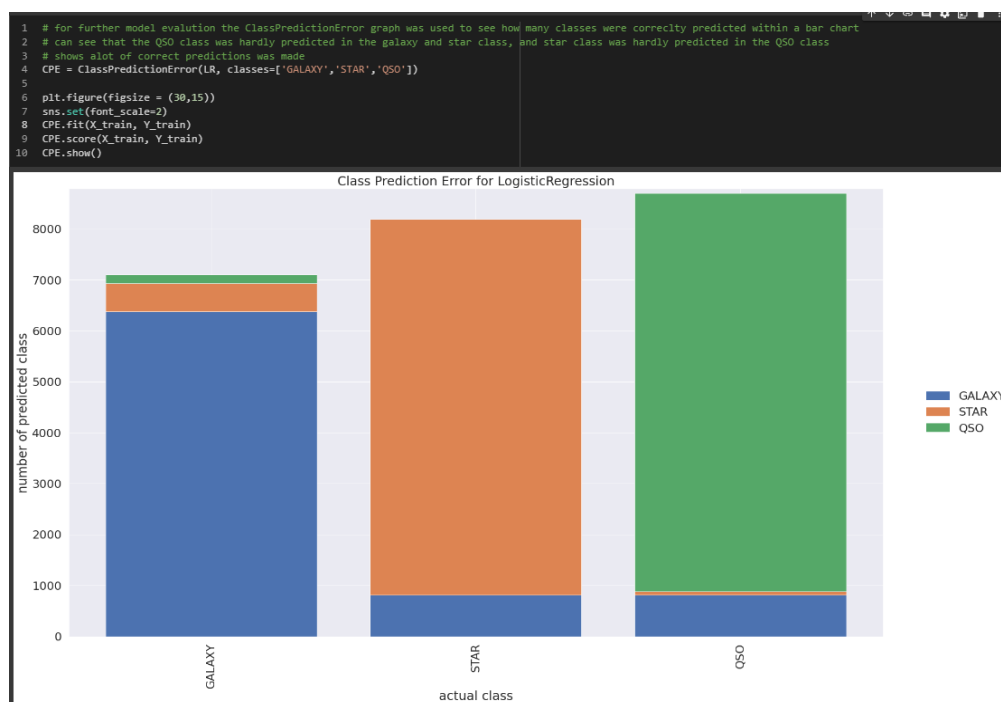


Figure 105 class prediction

The Class Prediction Error graph was used from the scikit-learn library to further analyse model predictions made from training with SEEN data. It shows the predictions made for each class. The QSO class was hardly predicted to be the galaxy and star class, and the star class was hardly predicted to be the QSO class. So the model was good at predicting each correct class

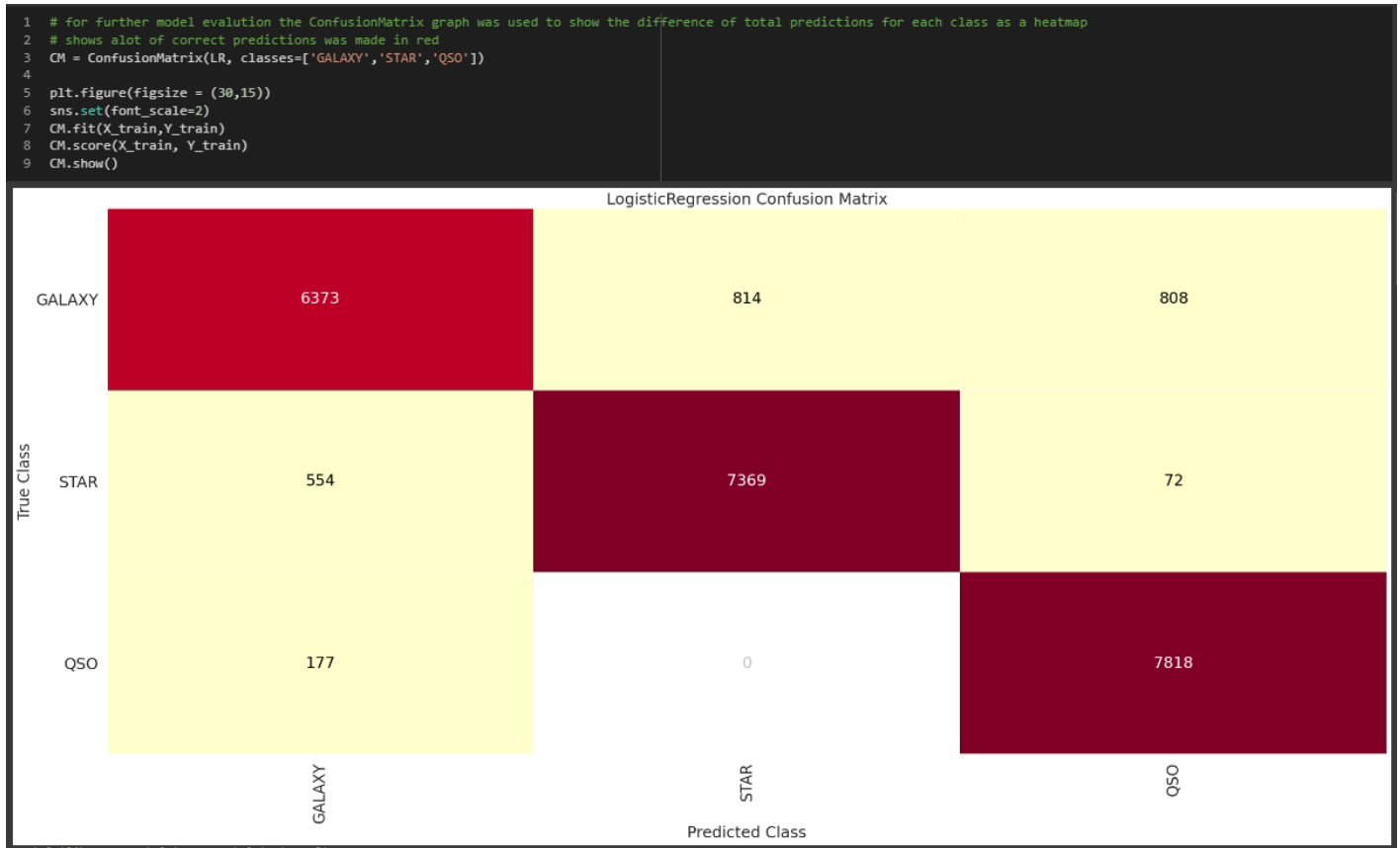
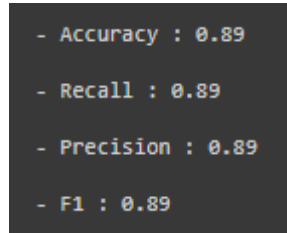


Figure 106 confusion matrix

A Confusion Matrix graph was used from the scikit-learn library to further analyse model predictions made from training with SEEN data. It shows the predictions made for each class. The star class was predicted 0 times to be the QSO class, and a lot of incorrect predictions were made in the galaxy class as 814 star classes and 808 QSO classes were predicted to be it. QSO was by far the most accurate here.

4.2 - Evaluation results on “unseen” data

The following table shows the evaluation metric results of my model tested with UNSEEN data.

UNSEEN DATA		
Metric	Multinomial Logistic Regression	Evidence
Average Accuracy	89%	 <p>Figure 107 metric results</p>
Average Precision	89%	
Average Recall	89%	
Average F1 Score	89%	

The result for the model testing with UNSEEN data was 89% for every evaluation metric. I would like to have the results to be 90% or above but it is very close and it's still a good result, so I'm happy with it overall.

```

1 # classification report further shows precision, recall, f1-score metric scores with more detail such as macro and weighted average
2 print(classification_report(Y_test, Y_pred))

```

	precision	recall	f1-score	support
GALAXY	0.87	0.88	0.84	1977
QSO	0.90	0.91	0.90	1977
STAR	0.90	0.97	0.94	1977
accuracy			0.89	5931
macro avg	0.89	0.89	0.89	5931
weighted avg	0.89	0.89	0.89	5931

Figure 108 classification report results

Star and QSO had the highest precision score of 90%, Star also had the highest recall score of 97%, and an f1 score of 94%. The star class is the highest throughout each emulation matrix, this means that the model is performing well in predicting it.

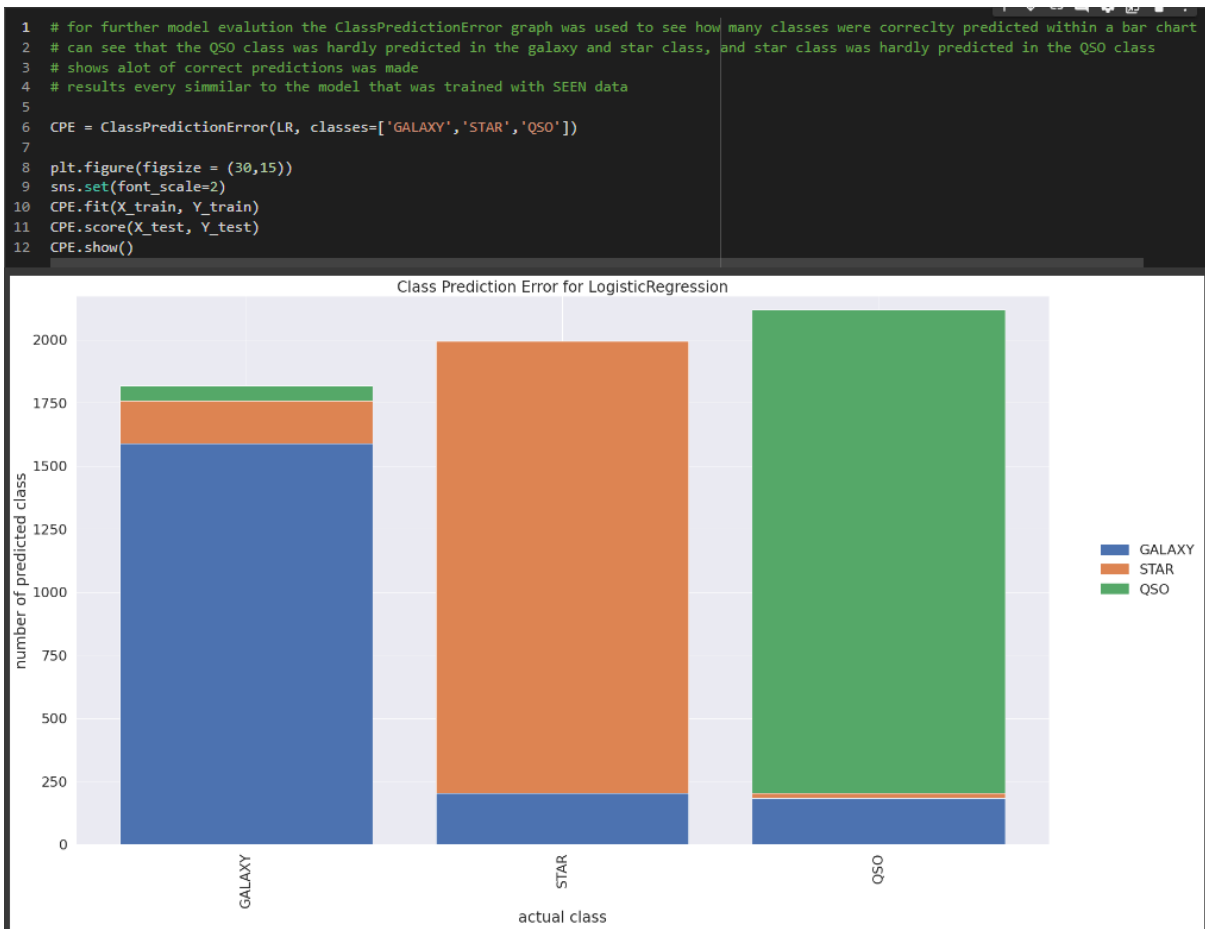


Figure 109 class prediction

The QSO class was hardly predicted to be the galaxy and star class, and the star class was hardly predicted to be the QSO class. So the model was good at predicting each correct class. This was very similar results to the model trained with seen data, as each minority predicted class was the same for each class.



Figure 110 confusion matrix

The star class was predicted 0 times to be the QSO class, and 61 GALAXY class where predicted to be QSO which is not a lot. Also, only 18 QSO classes were predicted to be STAR. QSO was by far the most accurate here. This was very similar results to the model trained with seen data, as they both had 0 stars classes to be predicted as the QSO class.

4.3 – Comparing results on “unseen” and “seen” data

SEEN				
1 # classification report further shows precision				
2 print(classification_report(Y_train, Y_pred))				
	precision	recall	f1-score	support
GALAXY	0.90	0.80	0.84	7995
QSO	0.90	0.92	0.91	7995
STAR	0.90	0.98	0.94	7995
accuracy			0.90	23985
macro avg	0.90	0.90	0.90	23985
weighted avg	0.90	0.90	0.90	23985

Figure 112 initial model trained with seen data

UNSEEN				
1 # classification report further shows precision				
2 print(classification_report(Y_test, Y_pred))				
	precision	recall	f1-score	support
GALAXY	0.87	0.80	0.84	1977
QSO	0.90	0.91	0.90	1977
STAR	0.90	0.97	0.94	1977
accuracy			0.89	5931
macro avg	0.89	0.89	0.89	5931
weighted avg	0.89	0.89	0.89	5931

Figure 111 initial model trained with unseen data

When comparing the accuracy of the two models they are both very similar with only a 1% difference. The model trained with UNSEEN has 90% accuracy which is more accurate than the model trained with SEEN data which got 89% accuracy. As there isn't a big difference between the two accuracies, think there are no overfitting problems. Because models that are overfitting typically have a high accuracy on the training set but a lower accuracy on the test set. I am happy with my overall results for both models.

5 - Evaluation and further modeling improvements

5.1 - Initial evaluation comparison

The following table shows all our evaluation metric results for each of our models with UNSEEN data.

UNSEEN DATA				
Metric	Multinomial Logistic Regression	Naïve Bayes	SVM	KNN
Accuracy	89%	90%	96%	91%
Precision	89%	90%	96%	91%
Recall	89%	90%	96%	91%
F1 Score	89%	90%	96%	90%

Each of our classification reports was compared to see who got better scores in each evaluation metric for each class.

```
1 # classification report further shows prec
2 print(classification_report(Y_test, Y_pred))
```

	precision	recall	f1-score	support
GALAXY	0.87	0.80	0.84	1977
QSO	0.90	0.91	0.90	1977
STAR	0.90	0.97	0.94	1977
accuracy			0.89	5931
macro avg	0.89	0.89	0.89	5931
weighted avg	0.89	0.89	0.89	5931

Figure - 114 Logistic Regression classification report

```
print(classification_report(Y_test, Y_pred))
```

	precision	recall	f1-score	support
0	0.90	0.79	0.84	3797
1	0.81	0.92	0.86	3797
2	0.99	0.99	0.99	3797
accuracy			0.90	11391
macro avg	0.90	0.90	0.90	11391
weighted avg	0.90	0.90	0.90	11391

Figure - 113 Naïve Bayes classification report

```
3 print(classification_report(Y_test, predicted))
```

	precision	recall	f1-score	support
GALAXY	0.98	0.95	0.97	11785
QSO	0.84	0.91	0.87	1977
STAR	0.96	1.00	0.98	4316
accuracy			0.96	18078
macro avg	0.93	0.95	0.94	18078
weighted avg	0.96	0.96	0.96	18078

Figure - 116 SVM classification report

```
2 knn_test_classifier_report = classification_report(
3 print(knn_test_classifier_report)
```

	precision	recall	f1-score	support
GALAXY	0.90	0.82	0.85	10904
QSO	0.94	0.93	0.94	10875
STAR	0.88	0.97	0.92	11102
accuracy			0.91	32881
macro avg	0.91	0.91	0.90	32881
weighted avg	0.91	0.91	0.90	32881

Figure - 115 KNN classification report

Without including the accuracy metric, SVM has the best results for predicting the GALAXY class, while KNN had the best results for predicting the QSO class, and navies base had the best results for predicting the STAR class with 99% for each metric.

Each of our confusion matrixes was compared to see all the total of individual predictions made for each class.

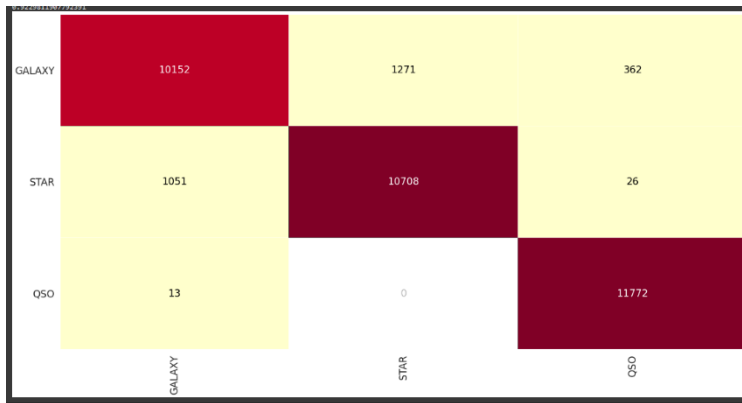


Figure 120 – Logistic regression confusion matrix

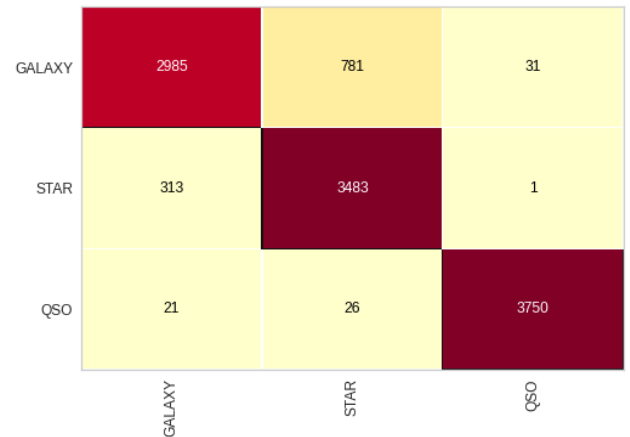


Figure 120 – Naïve Bayes confusion matrix

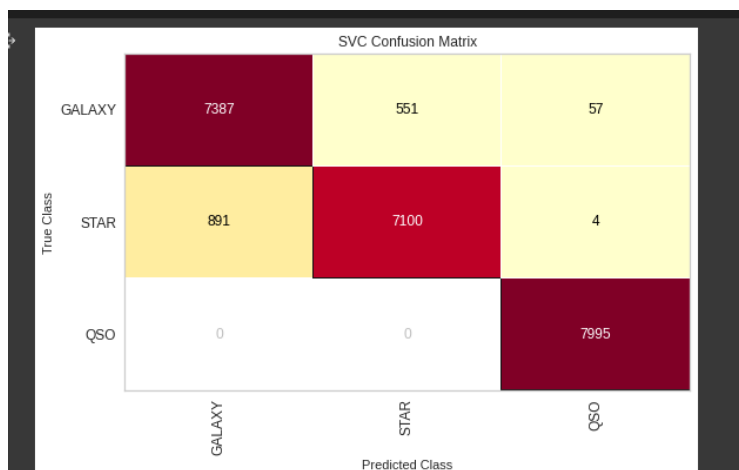


Figure 118 – SVM confusion matrix

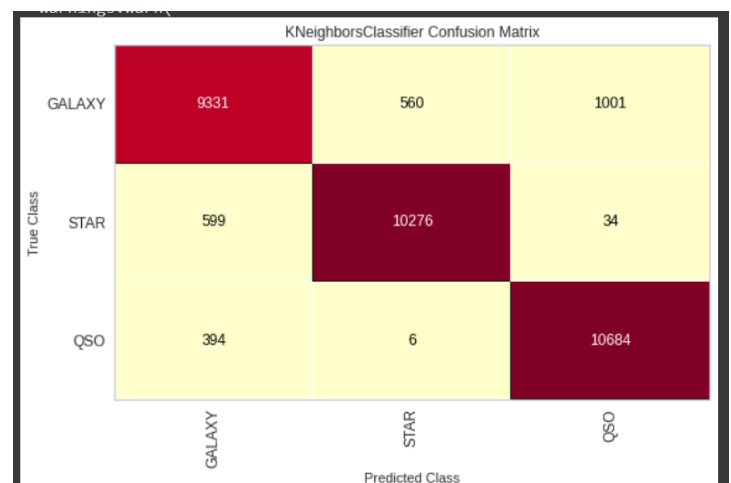


Figure 118 – KNN confusion matrix

SVM is more accurate at predicting the QSO class as GLAXY and STAR classes were never predicted to be it, so it got 100% accuracy. Logistic regression was also very accurate at predicting the QSO class, as it was never predicted as STAR and in only 13 cases it was predicted as GALAXY. KNN and Naïve Bayes had their predictions more spread out.

Overall, the SVM algorithm (Zains) was far superior to all the other algorithms with UNSEEN data and used relatively the same techniques for data pre-processing and cleaning. The SMV algorithm had 96% accuracy for each evaluation metric used. For accuracy, his model was predicted correctly 96% of the time. For precision, 90% of the positive predictions made by his model were correct. For recall, the model identified 90% of the positive instances. For F1 score, it's 96% as it had a good balance between high precision and high recall which there both 96%. The second-best algorithm was the KNN algorithm (Reeces) which got 91% for each evaluation metric except for F1 score which got 90%. The third best algorithm was Naïve Bayes (Manrajs) which got 90% for each evaluation metric. The fourth best algorithm was Multinomial Logistic Regression (mine) which got 89% for each evaluation metric, Naïve Bayes and logistic regression all had a difference of 1% for their metric results so they were very similar. SMV had an error rate of 4% in predictions, KNN had an error rate of 9%, while naive Bayes and logistic regression had an error rate of 10%. Algorithm for best to worst based on metric results: SVM, KNN, Naïve Bayes, Logistic Regression.

We think SVM was superior to the other algorithms as it's known to be more effective on larger datasets like ours because it can handle high-dimensional data like in our case and it's less likely to overfit compared to other algorithms.

All the metric results were all the same for every algorithm except KNN with was 1% off on F1 score. This means the model is making the same number of correct predictions and incorrect predictions for each class. We think this is because our dataset was balanced, as there is the same number of observations for each class, and the model is making predictions with the same rate of success and failure for each class.

5.2 - Further modeling improvements attempted

5.2.1 – Standardization (Data Scaling)

Initially, for data scaling, the research team chose a scaling technique that we thought was more suitable for classification algorithms, either standardization with StandardScaler or Normalization with MinMaxScaler was used. Both these scaling techniques can be applied to the dataset for our algorithms; however, they each have their benefits and limitations over each others.

To improve our dataset we chose the opposite technique, Me and Manraj initially used Normalization but now will try Standardization for our algorithms, and Reece and Zain initially used Standardization but now will try Normalization for their algorithms. To see which scaling technique is overall better for each of our algorithms. This will be done on both the training and testing set.

Some limitations of normalization were that it's sensitive to outliers in the data and could potentially alter the rest of the data when trying to scale them, unlike standardization. Outliers still contain in our dataset after removing them with IQR, so standardization will be used to counter these issues for those who initially used normalization.

Some limitations of standardization are that it's very time-consuming and costly to implement for our dataset, unlike normalization. Our dataset is very large with over 100,000 observations before data pre-processing and cleaning, so normalization will be used to counter these issues for those who initially used standardization.

```

1 # seperate target class and features from the training and testing set to perform standardized just onto the features
2
3 # training set
4 X_train = df_train.drop(["class"], axis = 1)
5 Y_train = df_train["class"]
6
7 # testing set
8 X_test = df_test.drop(["class"], axis = 1)
9 Y_test = df_test["class"]

```

Figure 121 separating target class to input features for both training and testing set

For this process I had to separate the features and target variable from each training and testing set.

```

1 # decided to normalize the data using StandardScaler to remove unwanted noise from dataset like duplicated observations and redundancies.
2 # this was done for both the testing and training set as both need to be standardized for them to be fit into the model
3
4 # initialize scaler variable for MinMaxScaler function
5 scaler = StandardScaler()
6 # fit with training to estimate the minimum and maximum observable values for when training and testing set are transformed
7 scaler.fit(X_train)
8
9 # standardizing testing set
10 X_train = scaler.transform(X_train)
11 X_train = pd.DataFrame(X_train)
12
13 # standardizing testing set
14 X_test = scaler.transform(X_test)
15 X_test = pd.DataFrame(X_test)

```

Figure 122 standardization code for both testing and training sets

X_train was fit with the StandardScaler and then X_train and X_test was transformed using the scaler, so the data is scaled the same.

1	X_train								
	0	1	2	3	4	5	6	7	8
0	0.086176	0.379449	0.519133	0.395764	0.426687	-0.139962	0.683822	-0.139929	0.045328
1	1.625973	1.736814	1.042921	0.836985	0.450274	0.938320	1.080348	0.938299	1.459319
2	0.219141	0.387392	-0.001315	-0.040714	-0.128222	0.116634	0.025498	0.116630	0.282279
3	-0.813146	-0.998417	-1.060448	-1.035828	-1.030837	-1.026189	-0.735454	-1.026163	-1.174142
4	-0.037426	0.410146	0.958926	1.347877	1.492507	-0.380155	-1.018467	-0.380178	-0.121088
...
72508	-0.089111	-0.191264	0.181471	1.377880	1.089959	0.004523	1.395138	0.004491	0.142313
72509	-1.391718	-1.751805	-1.880523	-1.923228	-1.937420	-1.345949	-0.806832	-1.345928	-1.532324
72510	0.798611	0.367430	0.108813	-0.265785	-0.495188	-0.581676	-1.015140	-0.581687	-0.332691
72511	-1.888588	-1.988938	-1.993511	-1.955474	-1.843554	-0.779807	-1.015791	-0.779798	-0.617032
72512	1.223438	0.228104	-0.142213	-0.166326	-0.150482	-0.340757	-0.107789	-0.340759	-0.091332

72513 rows x 9 columns

Figure 123 new dataset after standardization

How the dataset now looks after its standardized. Data is not scaled from 0 to standard deviation of 1. Data is much easier to read also.

5.2.2 – Dimensionality Reduction (Data Transformation)

Initially, for data transformation, we did not perform any dimensionality reduction techniques as we thought we did enough initial pre-processing, and we did not properly research dimensionality reduction to see what it does. After researching it, we found that it's suitable with high-dimensional datasets like ours that have extremely high values as shown in (figure 80). Also, it is generally better on datasets that have many features and with data that is hard to read like ours.

To improve our dataset we all will use PCA for dimensionality reduction on our dataset using PCA as it's the most commonly used technique for dimensionality reduction. PCA is used to reduce the dimensionality of features while preserving as much of the original variation in the data as possible. This will be done on both the training and testing.

Dimensionality reduction was done after scaling the dataset is first required to be normalized for PCA to work.

spec_obj_ID
6.543777e+18
1.176014e+19
5.152200e+18
1.030107e+19
6.891865e+18
5.658977e+18
1.246262e+19
6.961443e+18
7.459285e+18
2.751763e+18

Figure 124 high-dimensional data

```

1 # this was done after standardscaler as the dataset first needs to be scaled for PCA to work
2 # this was done for both the testing and training set as both need to have the same number of features for them to be fit into the model
3
4 # initialize pca variable for PCA function
5 pca = PCA(0.95)
6 # fit with training to estimate the minimum and maximum observable values for when training and testing set are transformed
7 pca.fit(X_train)
8
9 # PCA on training set
10 X_train = pca.transform(X_train)
11 X_train = pd.DataFrame(X_train)
12
13 # PCA on testing set
14 X_test = pca.transform(X_test)
15 X_test = pd.DataFrame(X_test)

```

Figure 125 PCA code

X_train was fit with PCA and then X_train and X_test was transformed with the scaler, so the data is transformed the same. I transferred both sets back into a data frame as the process turned them into an array. The choice of 0.95 for PCA is often used as a rule of thumb for retaining a high proportion of the variance in the original data.

	0	1	2	3
0	-0.740875	-0.498094	0.651841	0.026759
1	-3.314443	-0.248999	-0.336745	-1.139048
2	-0.317055	0.031234	-0.230588	-0.308523
3	2.984214	-0.140001	0.061045	0.011032
4	-0.946214	-1.335706	-0.323415	1.908880
...
72508	-1.270717	-0.230064	1.544759	0.534917
72509	4.710046	0.446703	0.222007	-0.403341
72510	0.641200	-1.091129	-1.082188	-0.297995
72511	4.318517	1.627870	-0.066251	-0.069659
72512	0.029717	-0.859089	-0.527885	-0.792120

Figure 126 new dataset after PCA

How the dataset now looks after dimensionality reduction. Only 4 features remain.

5.2.3 – Balancing Dataset (Data Transformation)

Initially, for data transformation, the research team chose to balance our dataset using under-sampling, except Reece who used over-sampling as his algorithm worked better with more training and testing data. Under-sampling and oversampling are both good techniques with their own benefits and limitations over each other.

To improve our dataset Me, Manraj and Zain decided to now use oversampling instead of undersampling, as with undersampling it potentially discards useful data, as every class has the same amount of observations as the minority class. The model's performance may be better using oversampling as the model will have more training and testing data to use, and this data could be important. This is because every class will have the same amount of observations as the majority class, being the galaxy class. Will see which balancing technique is overall better for each of our algorithms. Reece will continue using oversampling as well.

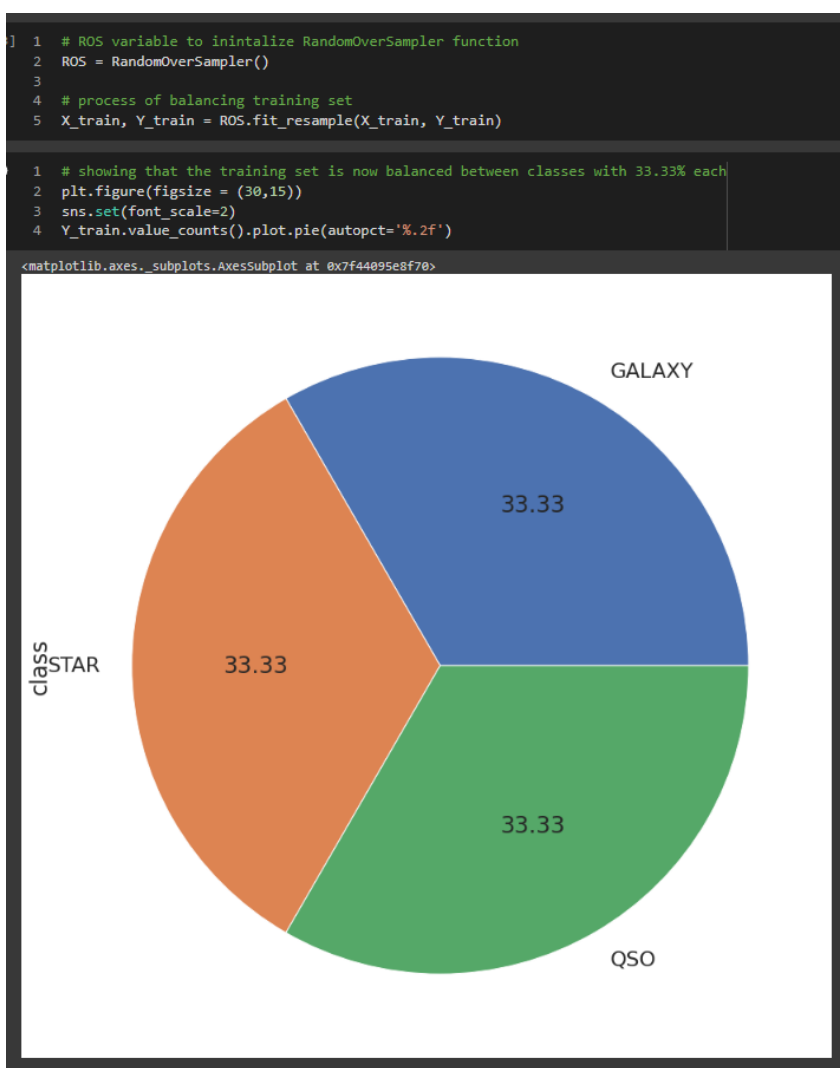


Figure 127 balacned training set

Balancing process of using oversampling with RandomOverSampling function on the training set. 33% for each class, evenly divided.

```

1 # showing the balanced sum of observations for each class object for training set
2 # each class has 47332 observations after over sampling
3
4 Y_train.value_counts()

```

GALAXY	47332
STAR	47332
QSO	47332

Name: class, dtype: int64

Figure 128 new class totals in training set

Each class has 47332 corresponding observations after the over sampling process in the training set.

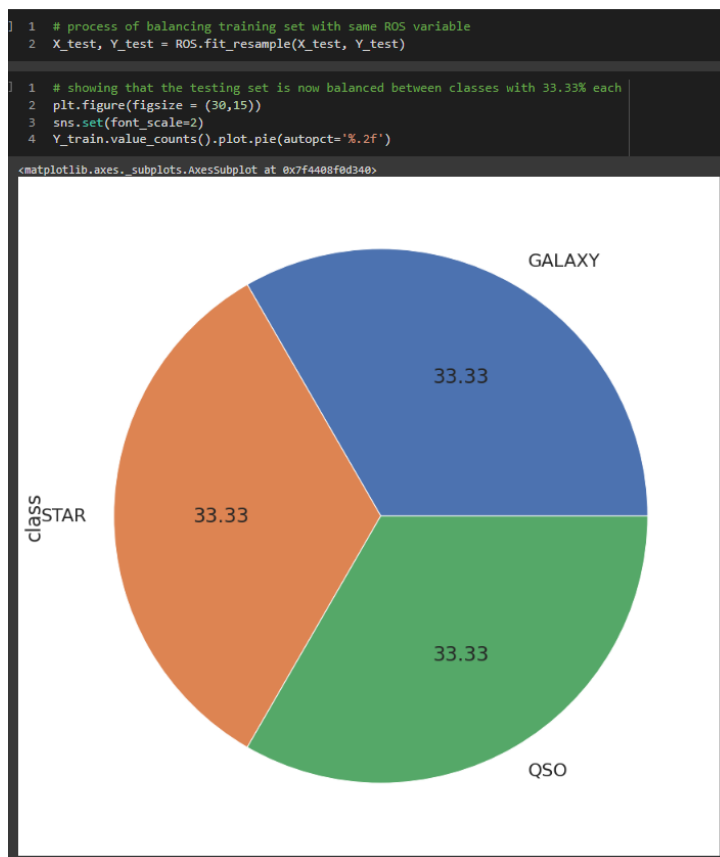


Figure 129 balanced testing set

Balancing process of using oversampling with RandomOverSampling function on the testing set. 33% for each class, evenly divided.

```

1 # showing the balanced sum of observations for each class object for testing set
2 # each class has 11785 observations after over sampling
3
4 Y_test.value_counts()

```

GALAXY	11785
STAR	11785
QSO	11785

Name: class, dtype: int64

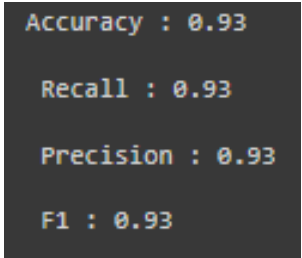
Figure 130 new class totals in testing set

Each class has 11785 corresponding observations after the over sampling process in the testing set.

5.3 - Final Evaluation results

5.3.1 – Evaluation results on “seen” data

The following table shows the evaluation metric results of my improved model tested with SEEN data.

SEEN DATA		
Metric	Multinomial Logistic Regression	Evidence
Average Accuracy	93%	 <p>Figure 131 metric results</p>
Average Precision	93%	
Average Recall	93%	
Average F1 Score	93%	

The final accuracy of my model trained with SEEN data was 93% for every used metric. My older model has 90% so my improved model has increased it by 3% which is a large increase.

```

1 # 0: GALAXY
2 # 1: QSO
3 # 2: STAR
4 print(classification_report(Y_train, Y_pred))

```

	precision	recall	f1-score	support
0	0.92	0.86	0.89	47332
1	0.90	0.92	0.91	47332
2	0.97	1.00	0.98	47332
accuracy			0.93	141996
macro avg	0.93	0.93	0.93	141996
weighted avg	0.93	0.93	0.93	141996

Figure 132 – NEW model results

On the new model, the galaxy has the highest precision of 92%, the star had the highest recall and f1 score of 94% and 98%. All the same as my old model. Overall the results for each evaluation metric were higher on the new model than on the old model.

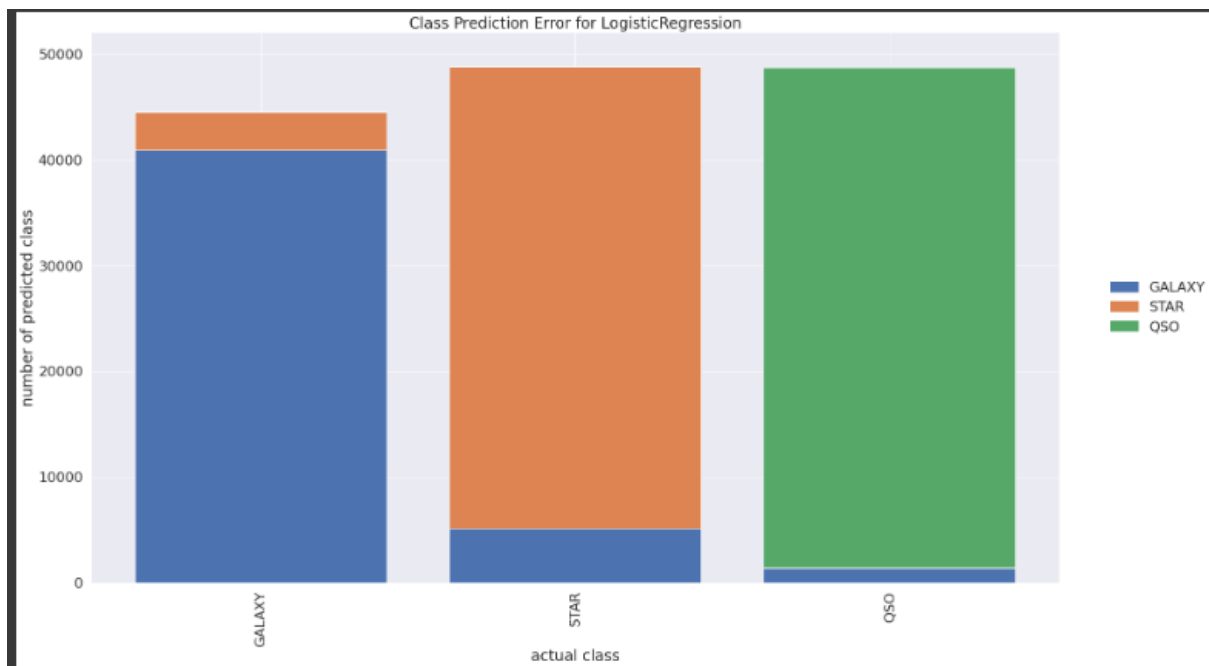


Figure 133 class prediction

The QSO class was hardly predicted to be the galaxy and star class, and the star class was hardly predicted to be the QSO class. So the model was good at predicting each correct class. This was overall the same as my old model.

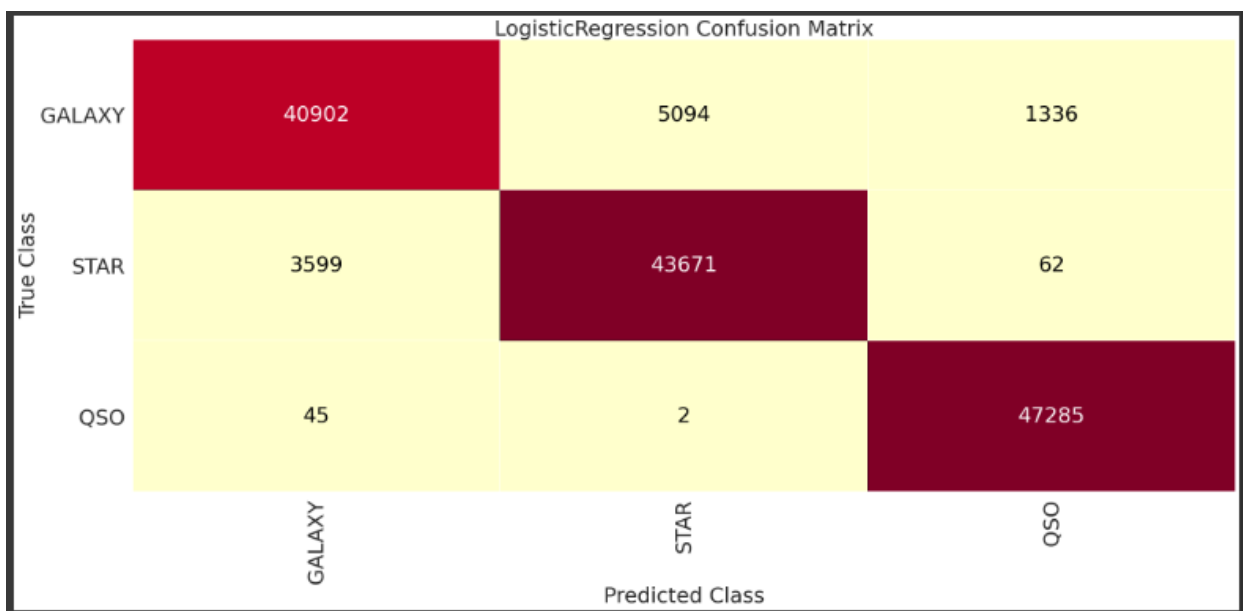
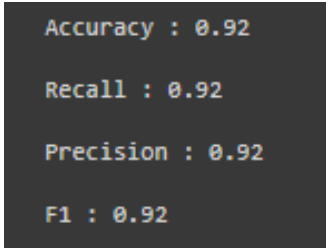


Figure 134 confusion matrices

The star and galaxy class were hardly predicted to be the QSO class as only 2 stars and 45 galaxies were predicted. The model was very accurate at predicting the QSO class, more than the rest, the same as my old model.

5.3.2 – Evaluation results on “unseen” data

The following table shows the evaluation metric results of my improved model tested with UNSEEN data.

UNSEEN DATA		
Metric	Multinomial Logistic Regression	Evidence
Average Accuracy	92%	 <p>Figure 135 metric results</p>
Average Precision	92%	
Average Recall	92%	
Average F1 Score	92%	

The final accuracy of my model trained with UNSEEN data was 92% for every used metric. My older model has 89% so my improved model has increased it by 3% which is a large increase. So a similar increase to my SEEN model.

```
# 0: Galaxy
# 1: Star
# 2: Quasar (QSO)
```

Figure class name to
connected each index

	precision	recall	f1-score	support
0	0.88	0.81	0.84	1977
1	0.91	0.90	0.91	1977
2	0.90	0.97	0.94	1977
accuracy			0.90	5931
macro avg	0.90	0.90	0.90	5931
weighted avg	0.90	0.90	0.90	5931

Figure 136 new model results

On our new mode, the star class had the highest precision of 91%, while the QSO class has the highest recall and f1 scores of 97% and 94%. Overall the results for each evaluation metric were higher on the new model than on the old model.

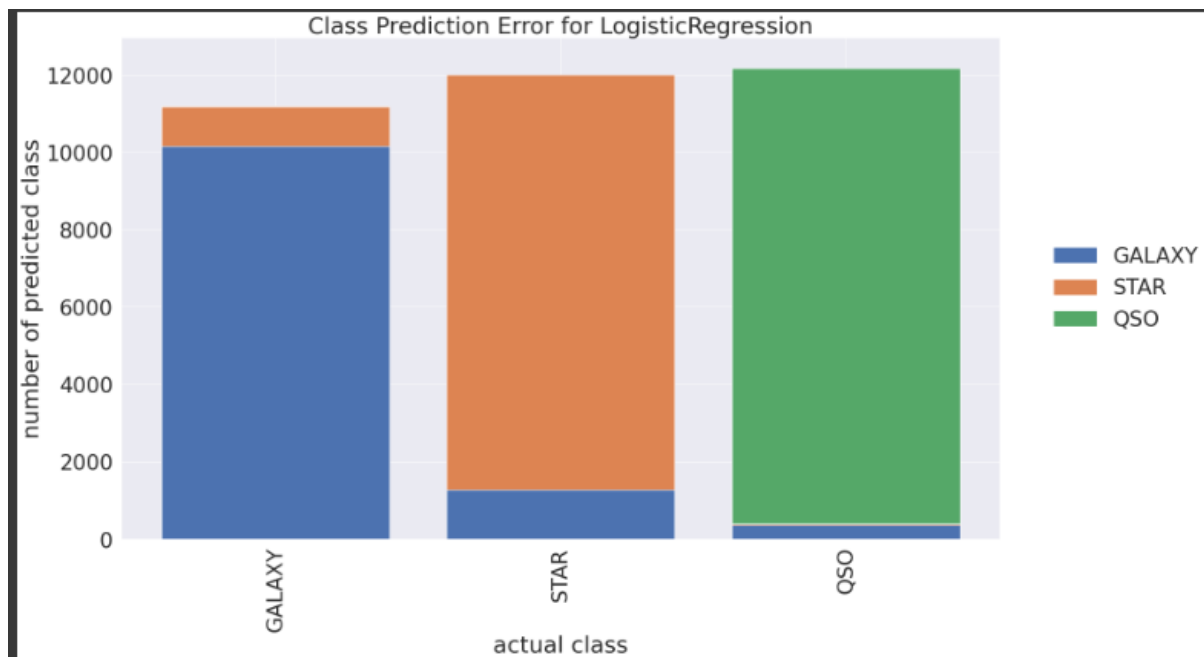


Figure 137 class prediction

The QSO class was hardly predicted to be the galaxy and star class, and the star class was hardly predicted to be the QSO class. So the model was good at predicting each correct class. This was overall the same as my old model and the improved model tested with SEEN data.

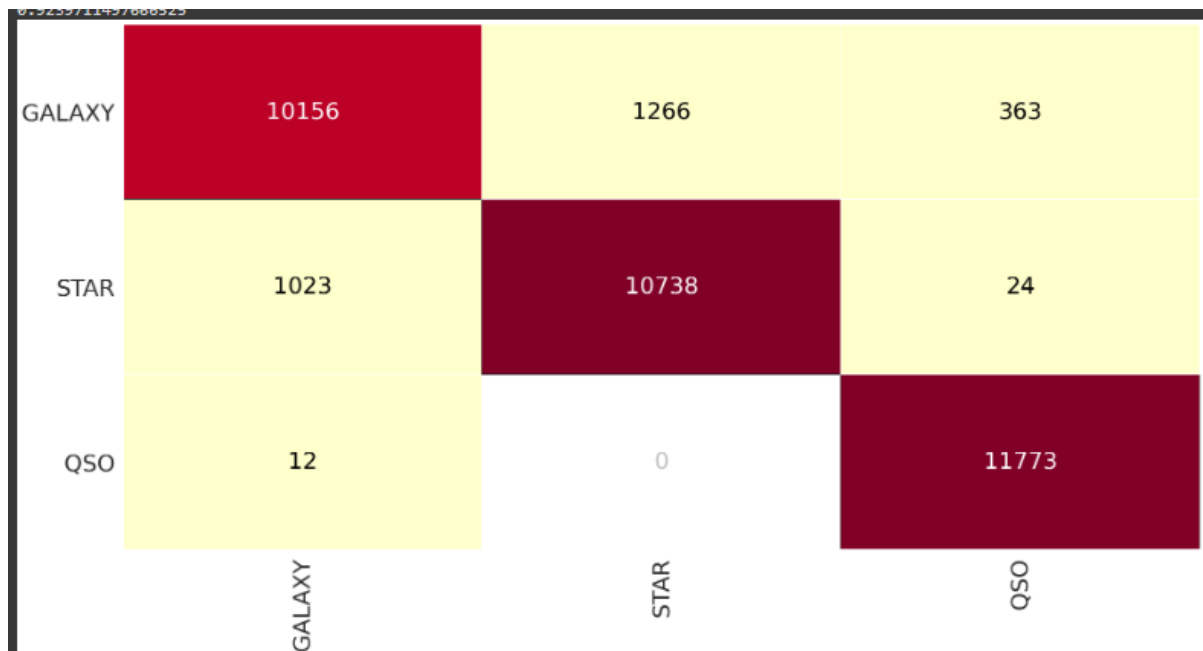


Figure 138 confusion matrix

The star was never predicted to be the QSO class and only 12 galaxies were predicted to be QSO. Again, the model was very accurate at predicting the QSO class, more than the rest

5.3.3 – Comparing results on “unseen” and “seen” data.

```
# 0: Galaxy
# 1: Star
# 2: Quasar (QSO)
```

Figure 139 encoded class

SEEN

```
1 # 0: GALAXY
2 # 1: QSO
3 # 2: STAR
4 print(classification_report(Y_train, Y_pred))
```

	precision	recall	f1-score	support
0	0.92	0.86	0.89	47332
1	0.90	0.92	0.91	47332
2	0.97	1.00	0.98	47332
accuracy			0.93	141996
macro avg	0.93	0.93	0.93	141996
weighted avg	0.93	0.93	0.93	141996

UNSEEN

```
4 print(classification_report(Y_test, Y_pred))
```

	precision	recall	f1-score	support
0	0.91	0.86	0.88	11785
1	0.89	0.91	0.90	11785
2	0.97	1.00	0.98	11785
accuracy			0.92	35355
macro avg	0.92	0.92	0.92	35355
weighted avg	0.92	0.92	0.92	35355

Figure 140 improved model trained with unseen

Figure 141 improved model trained with seen

When comparing the accuracy of the two models they are both very similar with only a 1% difference. The model trained with UNSEEN has 93% accuracy which is more accurate than the model trained with SEEN data that got 92% accuracy. As there isn't a big difference between the two accuracies, think there are no overfitting problems. Because models that are overfitting typically have a high accuracy on the training set but a lower accuracy on the test set. I am happy with my overall results for both models and large improvements were made to my older models, they are both 3% more accurate.

6 - Conclusion

6.1 - Summary of results

```
# 0: Galaxy
# 1: Star
# 2: Quasar (QSO)
```

Figure 142 encoded class

SEEN				
	precision	recall	f1-score	support
0	0.92	0.86	0.89	47332
1	0.90	0.92	0.91	47332
2	0.97	1.00	0.98	47332
accuracy			0.93	141996
macro avg	0.93	0.93	0.93	141996
weighted avg	0.93	0.93	0.93	141996

Figure 144 improve SEEN results

UNSEEN				
	precision	recall	f1-score	support
0	0.91	0.86	0.88	11785
1	0.89	0.91	0.90	11785
2	0.97	1.00	0.98	11785
accuracy			0.92	35355
macro avg	0.92	0.92	0.92	35355
weighted avg	0.92	0.92	0.92	35355

Figure 143 improved UNSEEN results

The accuracy of my Logistic Regression algorithm has improved by 3% for both the UNSEEN tested model and SEEN tested models. My initial accuracy for SEEN was 90% and UNSEEN was 89%, and now after improvements SEEN is 93% and UNSEEN is 92%. Also, the precision, recall, and f1 score has improved.

The improvements made to the model to achieve this was doing standardization instead of normalization, oversampling the dataset instead of under-undersampling it, and using dimensionality reduction with PCA. However, dimensionality reduction didn't improve my model, it was only from using standardization and oversampling.

Standardization was used as its less affected by outliers than normalization which our dataset still had after removing them with IQR, oversampling was used so the model can have more data to train and test with, and dimensionality reduction was used as our dataset has some high dimensional data like the spec_obj_ID and obj_ID features.

I have deemed that my new improved model is not overfitted accuracy difference is only 1% because models that are overfitting typically have a high accuracy on the training set but a lower accuracy on the test set.

6.2 - Suggested further improvements to the model development process

6.2.1 – What went well with the model development process

The parts that went well with the model development process were that the unwanted features were quickly determined within the EDA univariate analysis using kdeplots (2), which were then removed in feature selection for the data pre-processing and cleaning phase (3). These features had no correlation to the target class. Collaboration with the research went smoothly, as no one was behind in developing their models, and we all helped and encouraged each other throughout the module on Discord. Model development (4) went fine as it was simple to understand and execute. We all fully developed our algorithms and they all had high accuracies.

6.2.2 – How the model could be improved

Based on the evaluation, the models could be further improved by choosing a different astronomy dataset that has the same predictive problem but fewer features and observations. This is because I think our dataset was kind of too much for us to handle considering it has 18 features and 100,000 observations that contained many outliers and extremely high values. We managed in the end, but it was still tedious like doing the EDA process on 18 features. Outliers remained when we tried to remove them so the model could be improved by using a different outlier removal technique like Z-score, Isolation Forest, or Mahalanobis Distance. We should do more research on what method to use before implementing it.

6.3 - Reflection on Research Team

The positives drawn from working within a research team of 4 were that it improved my team-working skills, as well as my communication and collaboration skills when the team analysed, developed, and compared the classification algorithms and dataset. I enjoyed working in a research team, as it was fun learning and working together, and also because we could help each other to improve our models and coursework.

I provided the different pre-processing and cleaning methods we should use such as data scaling with normalization and standardization, and the outlier's method with IQR. Depending on our algorithm we used normalization or standardization. Reece was essential in fighting for the best dataset and researching its background information to make sure it was a suitable one for us to use. Manraj provided insight on how we should improve our models by using techniques such as dimensionality reduction using PCA as our dataset has some high dimensional data. Zain came up with our EDA questions and what graphs we should use for them. Also, he chose what evaluation metrics we should use when testing the performance of our models.

The team mitigated potential issues by meeting and discussing them in person within lab sessions, as well as at home through Discord by messaging and sometimes voice calling each other. This was done heavily throughout sections 1, 2, and 5 of the module as they required the most teamwork. We had a lot of issues in section 1 with choosing a suitable dataset.

6.4 - Reflection on Individual Learning

I learned many new topics throughout this module involving Machine Learning with developing a supervised classification algorithm being logistic regression using libraries such as pandas, NumPy, and scikit-learn. As well as comparing it with other developed classification algorithms with graph visualization using libraries such as matplotlib, seaborn and yellowbrick.

The aspect of the module I enjoyed the most was data visualization with graphs using the seaborn and matplotlib libraries. I enjoyed the *Exploratory Data Analysis* section (2) of the coursework where most of the data visualization with graphs was heavily used for univariate and multivariate analysis. Also working in a research team was fun and different, as most modules are independent. What I found the most difficult was the data pre-processing and cleaning phase as it was hard to understand the correct order to do required techniques, like should we balance the dataset first or last? This was something we debated. What I found the easiest was modeling the algorithms as it was a short process that was easy to understand and execute.

In the future, I will further improve my knowledge and skills within AI/ML/DL by implementing and comparing the other classification algorithms types such as decision trees and random forests. As well as to dive into regression algorithms such as linear and ridge regression, and also unsupervised algorithms such as clustering with K-means clustering, and association algorithms with Apriori. I also want to learn more about AI/ML libraries like *TensorFlow* and the *Caffe* library for Deep Learning. I will learn these new topics by watching and reading online courses from *YouTube*, *Coursera*, and *Harvard CS50AI*.

7 – References

- Fedesoriano. (2021). Stellar Classification Dataset - SDSS17. Retrieved [2022] from <https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17>
- Sourabh Gupta. (2021). Machine Learning Concepts. Retrieved [2022] from <https://ml-concepts.com/2021/10/07/univariate-analysis-for-outliers-in-machine-learnin/#:~:text=a.,of%20values%20in%20a%20dataset>
- SDSS. (2022). Data Access for SDSS DR17 Overview. Retrieved [2022] from https://www.sdss4.org/dr17/data_access/
- Thomas Plapinger. (2017). Visualizing your Exploratory Data Analysis. Retrieved [2022] from <https://towardsdatascience.com/visualizing-your-exploratory-data-analysis-d2d6c2e3b30e#:~:text=While%20fairly%20simple%20easy%20to,between%20two%20or%20more%20variables>
- Michael Waskom. (2021). Exploratory Data Analysis (Univariate, Bivariate, and Multivariate Analysis). Retrieved [2022] from <https://seaborn.pydata.org/>
- Shashank Gupta. (2017). Seaborn: statistical data visualization. Retrieved [2022] from <https://www.enjoyalgorithms.com/blog/univariate-bivariate-multivariate-analysis>
- John Hunter. (2021). Lines, bars and markers. Retrieved [2022] from <https://matplotlib.org/stable/gallery/index.html>
- sauravprateek. (2019). KDE Plot Visualization with Pandas and Seaborn. Retrieved [2022] from <https://www.geeksforgeeks.org/kde-plot-visualization-with-pandas-and-seaborn/>
- user237745. (2020). How can one interpret a heat map plot. Retrieved [2022] from <https://stats.stackexchange.com/questions/392517/how-can-one-interpret-a-heat-map-plot>
- Harikrishnan N B. (2019). Confusion Matrix, Accuracy, Precision, Recall, F1 Score. Retrieved [2022] from <https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd>
- STAT 200. (2022). 3.2 - Identifying Outliers: IQR Method. Retrieved [2022] from <https://online.stat.psu.edu/stat200/lesson/3/3.2#:~:text=We%20can%20use%20the%20IQR,add%20this%20value%20to%20Q3>
- Dinesh Yadav. (2019). Categorical encoding using Label-Encoding and One-Hot-Encoder. Retrieved [2023] from <https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd>
- Aniruddha Bhandari. (2020). Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization. Retrieved [2023] from <https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/>

- Guillaume Lemaitre. (2021). How to use sampling_strategy in imbalanced-learn Retrieved. [2023] from https://imbalanced-learn.org/stable/auto_examples/api/plot_sampling_strategy_usage.html
- Jason Brownlee. (2021). Multinomial Logistic Regression With Python. [2023] from <https://machinelearningmastery.com/multinomial-logistic-regression-with-python/>
- Sarang Narkhede. (2018). Understanding Confusion Matrix. [2022] from <https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>
- Ashwini Kumar Pal. (2018). Understanding Dimension Reduction with Principal Component Analysis (PCA). [2023] from [https://blog.paperspace.com/dimension-reduction-with-principal-component-analysis/#:~:text=Principal%20Component%20Analysis\(PCA\)%20is,a%20set%20of%20orthogonal%20axes](https://blog.paperspace.com/dimension-reduction-with-principal-component-analysis/#:~:text=Principal%20Component%20Analysis(PCA)%20is,a%20set%20of%20orthogonal%20axes)

8 – Bibliography

- Margherita Grandini. (2020). Metrics for Multi-Class Classification: an Overview. Retrieved [2022] from <https://arxiv.org/abs/2008.05756>
- Victoria Cox. (2017). Translating Statistics to Make Decisions. Retrieved [2022] from https://link.springer.com/chapter/10.1007/978-1-4842-2256-0_3
- Myeongsu Kang. (2018). Machine Learning: Data Pre-processing. [2022] from <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119515326.ch5>
- M. IKONOMAKIS. (2005). Classification Using Machine Learning Techniques. [2023] from <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=b0485fba23aabda526358f31cb5a382b66a08270>
- Felix Abramovich. (2021). Multiclass Classification by Sparse Multinomial Logistic Regression. [2023] from <https://ieeexplore.ieee.org/abstract/document/9410597>