Summary

In the era of big data and reliable algorithms, data classification is instrumental in the field of astronomical research. In this project, we explore supervised classification methods such as Random Forest, Decision Tree and Random Forest. We applied these methods to classify the dataset as Star, Galaxy or Quasar along with comparing different classification models to find the superior one.

Background

In the branches of astronomy, the classification of celestial objects is essential. The ability to collect data samples of stars, galaxies, and quasars is its main advantage. Even though quasars are crucial to a variety of astronomical studies and research, their sample sizes are still in the relative minority group. Significant increases in the sample sizes of quasars and other astronomical objects can only be produced through improved data classification methods, which will then make it possible to advance study. The use of machine learning models in the task of classification has grown more significant and common as contemporary telescopes capture a growing amount of astronomical data due to their precision and speed.

The classification models of random forest and support vector model (SVM), along with decision tree and Boosting Trees, were used with supervised machine learning models to either exclusively identify and distinguish quasars from other objects or to classify all three of the objects: stars, galaxies, and quasars. Additionally, the fact that numerous supervised classification models were trained using data from kaggle.com shows that the information and variables the dataset supplies are comprehensive, trustworthy, and closely related to the objective of identifying celestial objects.

Data Description

The data used in this study are from the Sloan Digital Sky Survey (SDSS), which is a famous astronomical survey that has spent more than 20 years attempting to create incredibly accurate and detailed images and a map of the universe. The data represents 100,000 observations of space collected by the SDSS (Sloan Digital Sky Survey). Every observation is defined by 17 feature columns and a class column that categorizes it as a star, galaxy, or quasar.

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

2.alpha = Right Ascension angle (at J2000 epoch)

3.delta = Declination angle (at J2000 epoch)

4.u = Ultraviolet filter in the photometric system

5.g = Green filter in the photometric system

6.r = Red filter in the photometric system

7.i = Near Infrared filter in the photometric system

8.z = Infrared filter in the photometric system

9.run_ID = Run Number used to identify the specific scan

10.rereun_ID = Rerun Number to specify how the image was processed

11.cam_col = Camera column to identify the scanline within the run

12.field_ID = Field number to identify each field

13.spec_obj_ID = Unique ID used for optical spectroscopic objects (this means that 2 different observations with the same spec_obj_ID must share the output class)

14.class = object class (galaxy, star or quasar object)

15.redshift = redshift value based on the increase in wavelength

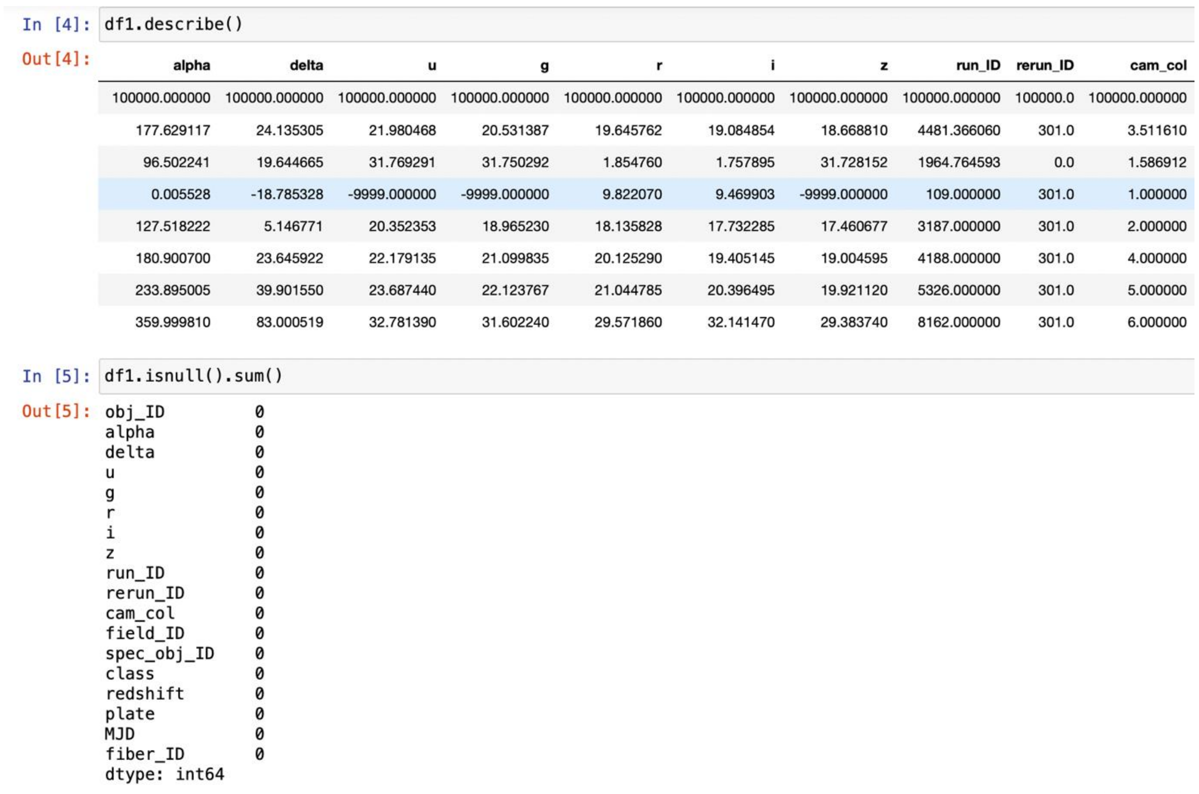
16.plate = plate ID, identifies each plate in SDSS

17.MJD = Modified Julian Date, used to indicate when a given piece of SDSS data was taken

18.fiber_ID = fiber ID that identifies the fiber that pointed the light at the focal plane in each observation

Exploratory Data Analysis

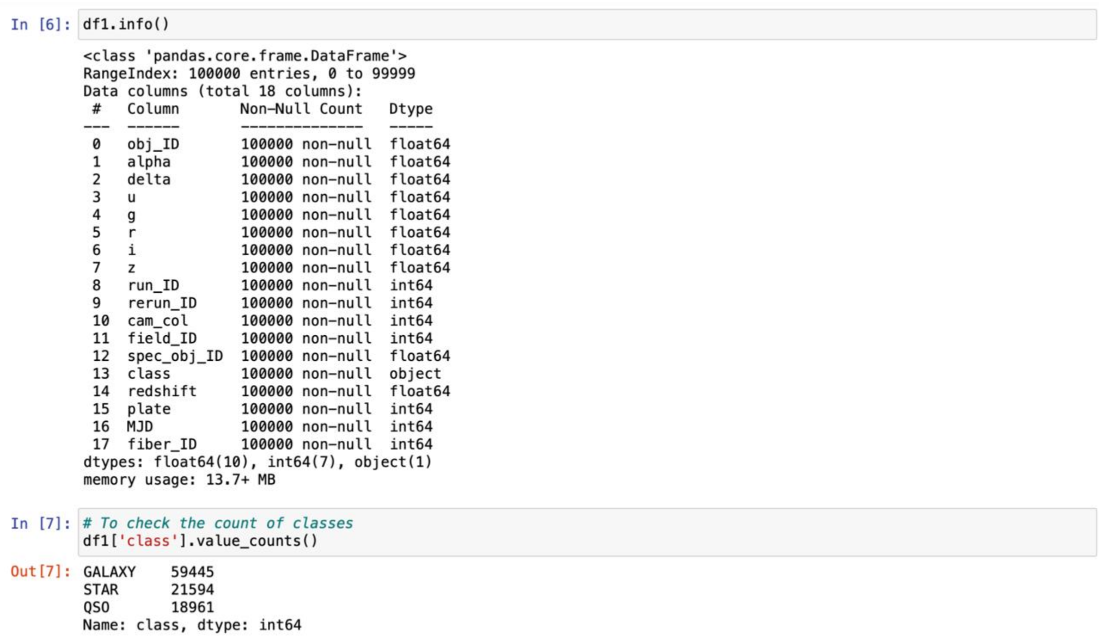
The dataset consists of 180000 records and a few null values with 18 attributes. The data set statistics have been obtained by using a summary function in R. Furthermore, it has been identified that the dataset has three classes- Galaxy, QSO and Star having 59445, 18961 and 21594 records.



```
In [4]: df1.describe()
Out[4]:
```

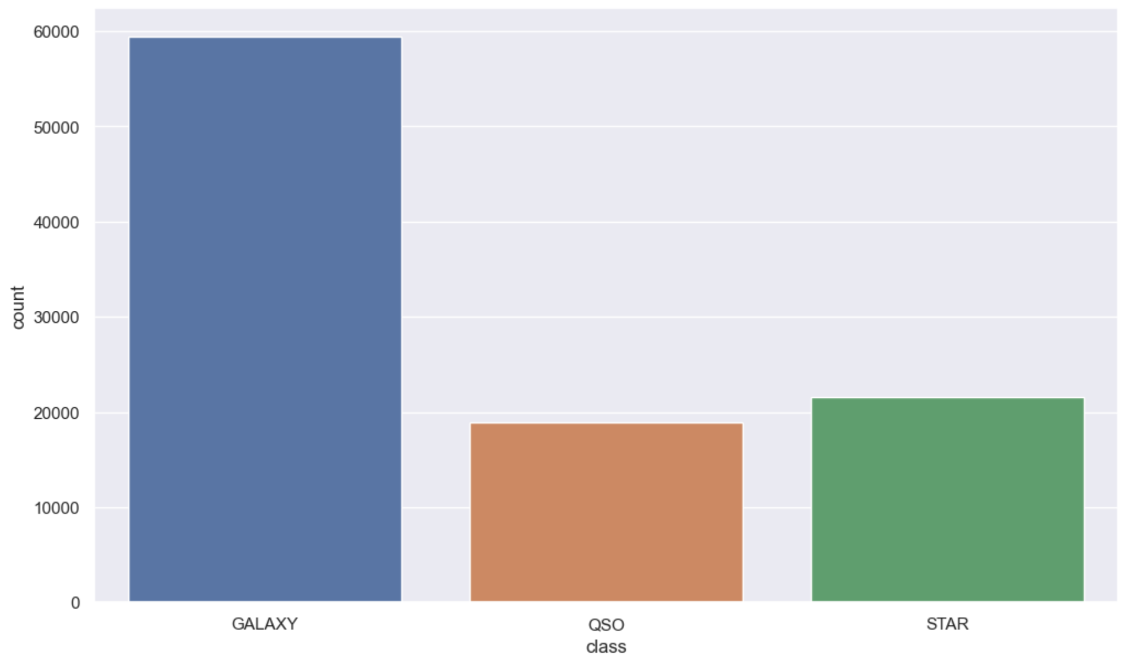
alpha	delta	u	g	r	i	z	run_ID	rerun_ID	cam_col
100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.0	100000.000000
177.629117	24.135305	21.980468	20.531387	19.645762	19.084854	18.668810	4481.366060	301.0	3.511610
96.502241	19.644665	31.769291	31.750292	1.854760	1.757895	31.728152	1964.764593	0.0	1.586912
0.005528	-18.785328	-9999.000000	-9999.000000	9.822070	9.469903	-9999.000000	109.000000	301.0	1.000000
127.518222	5.146771	20.352353	18.965230	18.135828	17.732285	17.460677	3187.000000	301.0	2.000000
180.900700	23.645922	22.179135	21.099835	20.125290	19.405145	19.004595	4188.000000	301.0	4.000000
233.895005	39.901550	23.687440	22.123767	21.044785	20.396495	19.921120	5326.000000	301.0	5.000000
359.999810	83.000519	32.781390	31.602240	29.571860	32.141470	29.383740	8162.000000	301.0	6.000000

```
In [5]: df1.isnull().sum()

Out[5]: 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
        class          0
        redshift       0
        plate          0
        MJD            0
        fiber_ID       0
        dtype: int64
```



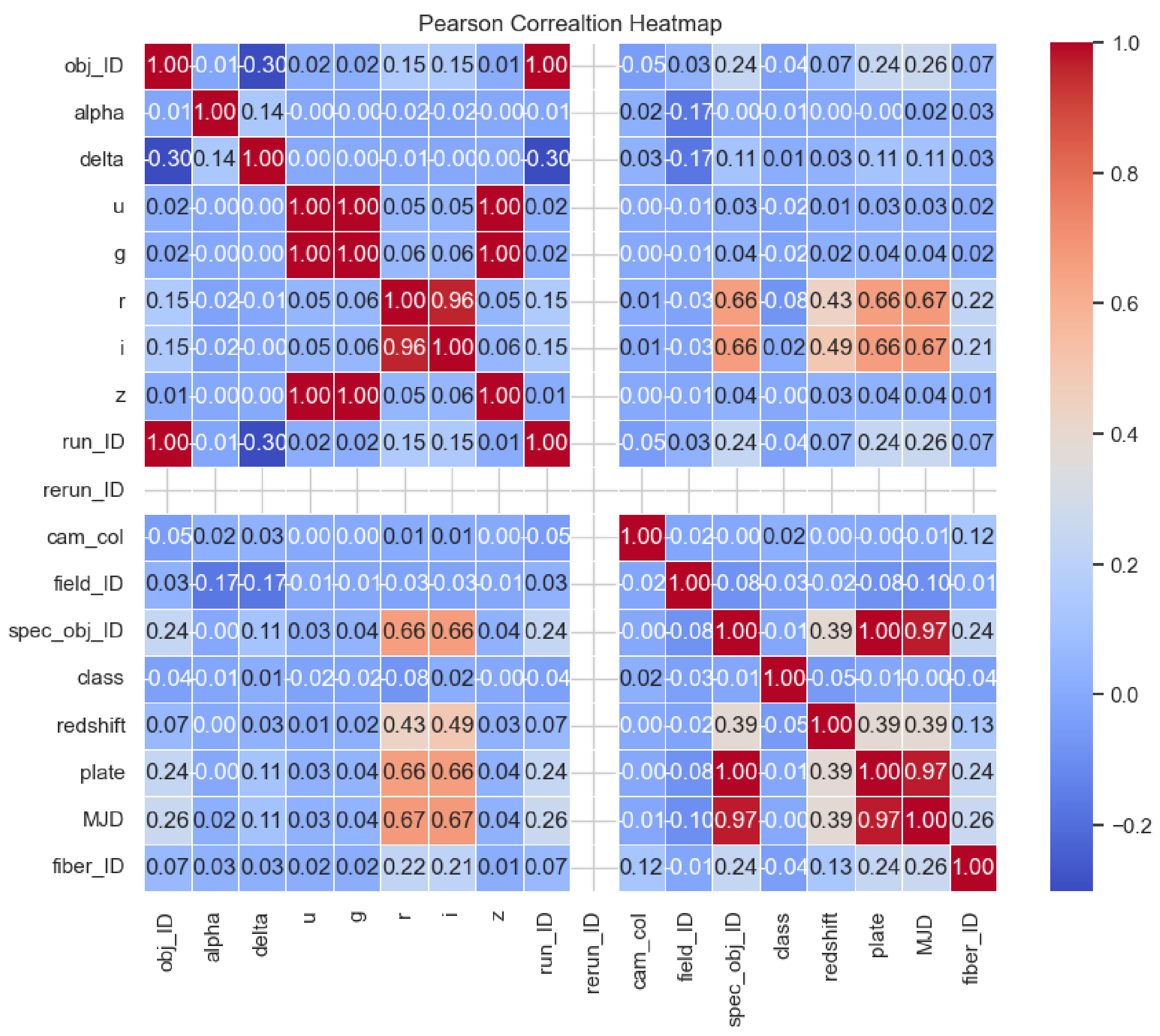
```
In [6]: df1.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 100000 entries, 0 to 99999
        Data columns (total 18 columns):
         #   Column       Non-Null Count   Dtype
        ---  ------       --------------   -----
         0   obj_ID       100000 non-null  float64
         1   alpha        100000 non-null  float64
         2   delta        100000 non-null  float64
         3   u            100000 non-null  float64
         4   g            100000 non-null  float64
         5   r            100000 non-null  float64
         6   i            100000 non-null  float64
         7   z            100000 non-null  float64
         8   run_ID       100000 non-null  int64
         9   rerun_ID     100000 non-null  int64
         10  cam_col      100000 non-null  int64
         11  field_ID     100000 non-null  int64
         12  spec_obj_ID  100000 non-null  float64
         13  class        100000 non-null  object
         14  redshift     100000 non-null  float64
         15  plate        100000 non-null  int64
         16  MJD          100000 non-null  int64
         17  fiber_ID     100000 non-null  int64
        dtypes: float64(10), int64(7), object(1)
        memory usage: 13.7+ MB

In [7]: # To check the count of classes
        df1['class'].value_counts()

Out[7]: GALAXY    59445
        STAR      21594
        QSO       18961
        Name: class, dtype: int64
```

The count plot was used to determine whether there was a reasonable amount of balance between the data entries for the three classes. This chart shows the relationship between different kinds of numbers (numeric variables) and what they categorize (categoric variables). Each of the categories in the categorical variable is represented by a bar. The bar on the chart has a number next to it. This number represents the numeric value of the bar. This bar plot represents distribution of the three different classes- Galaxy, QSO and Star. Additionally, it was also concluded that data imbalance is not a problem.



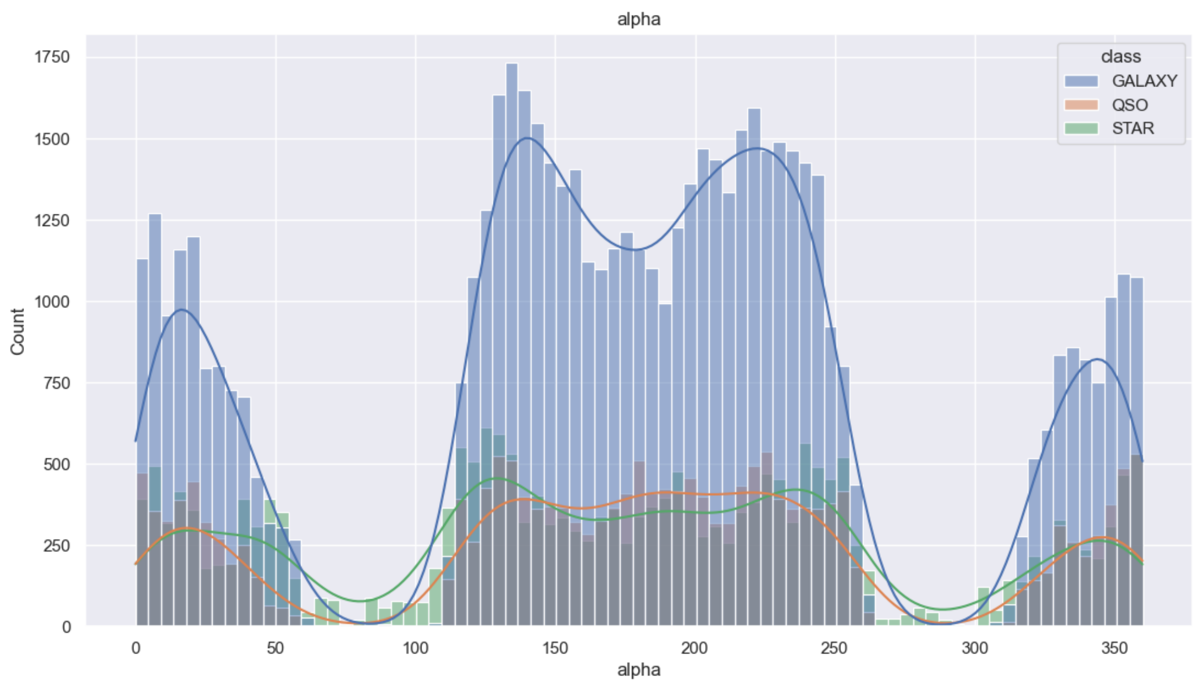
Check for correlation:-

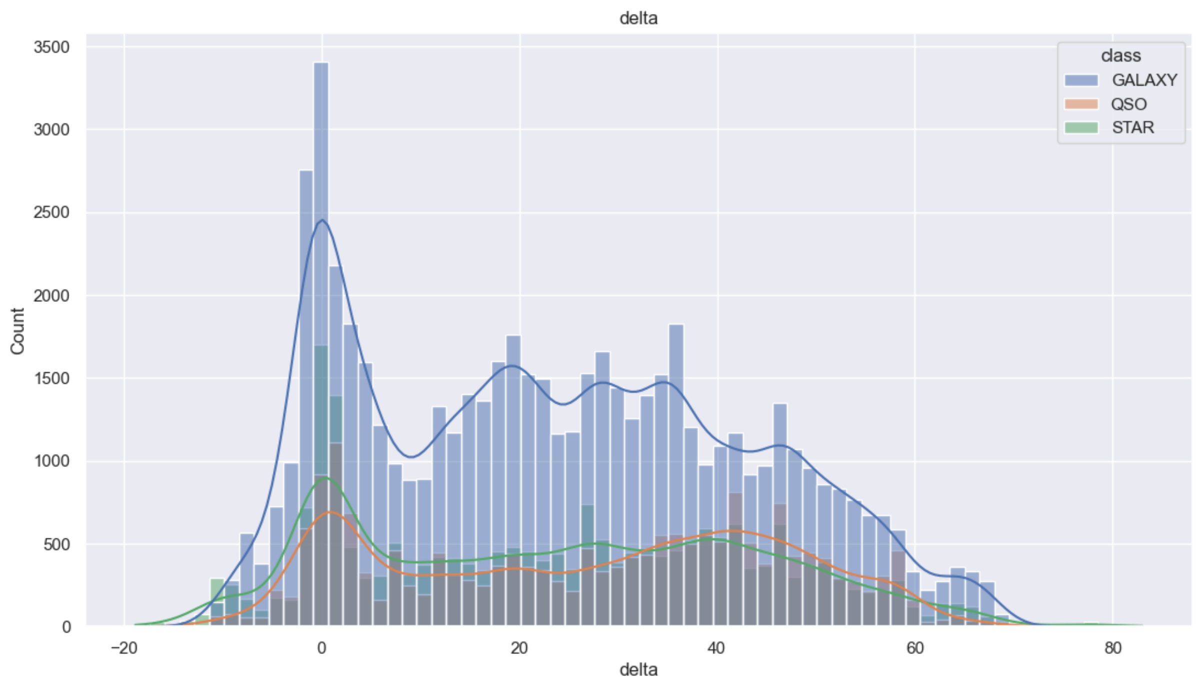
Initially we need to check for collinearity between the variables in the dataset which could lead to a 'Multicollinearity' problem. Multicollinearity tells a linear dependence between the independent variables. The existence of multicollinearity generates a high variance of the estimated coefficients and hence, the coefficient estimates corresponding to those interrelated independent variables will not be accurate in the classification model. They can become very sensitive to small changes in the model. Collinearity tells us how two variables are correlated with each other. In other words, if one variable is highly correlated with another, then most of the variance of one variable is explained by another (shares similar information). So, we may not include the highly correlated variable in the model. One way to extract correlation between variables is using Pearson's correlation coefficient. This correlation coefficient does not prove the 'Causality' (cause and effect). Causality tells the actual relation between variables unlike correlation which is just a possibility.

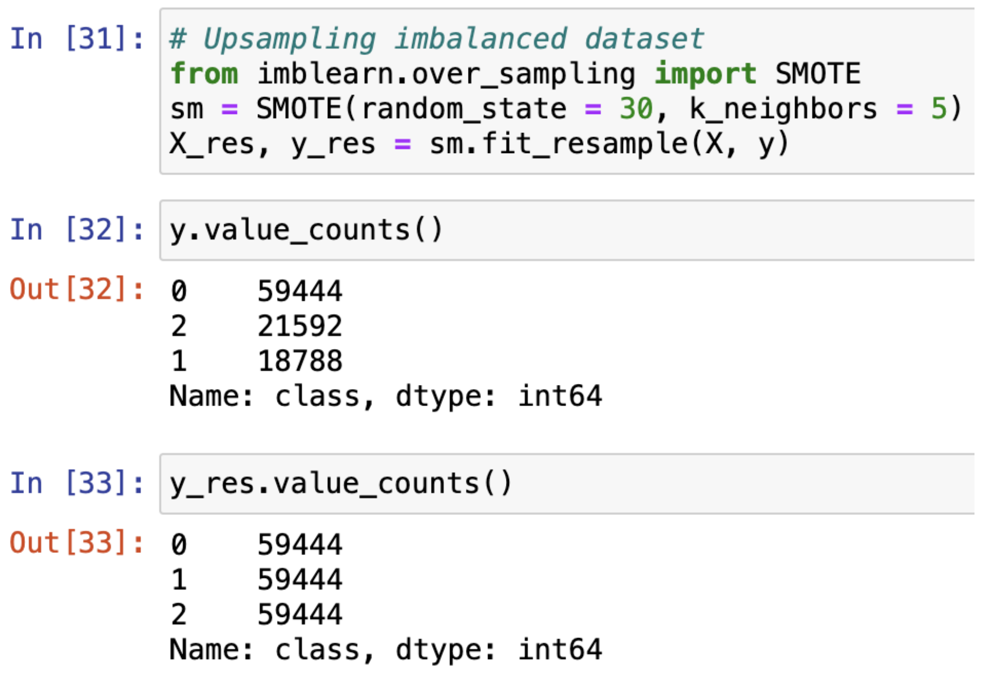


Feature Distribution Analysis:

As alpha, delta, u, g, r, i, z, redshift, plate, MJD are astronomical quantities, therefore, we'll keep them as our primary features. Rest of the features are irrelevant or have no significant difference in distribution by class.


alpha
Count
1750
1500
1250
1000
750
500
250
0
0
50
100
150
200
250
300
350
alpha
class
GALAXY
QSO
STAR


delta
Count
3500
3000
2500
2000
1500
1000
500
0
−20
0
20
40
60
80
delta
class
GALAXY
QSO
STAR



```
In [31]: # Upsampling imbalanced dataset
         from imblearn.over_sampling import SMOTE
         sm = SMOTE(random_state = 30, k_neighbors = 5)
         X_res, y_res = sm.fit_resample(X, y)

In [32]: y.value_counts()

Out[32]: 0    59444
         2    21592
         1    18788
         Name: class, dtype: int64

In [33]: y_res.value_counts()

Out[33]: 0    59444
         1    59444
         2    59444
         Name: class, dtype: int64
```

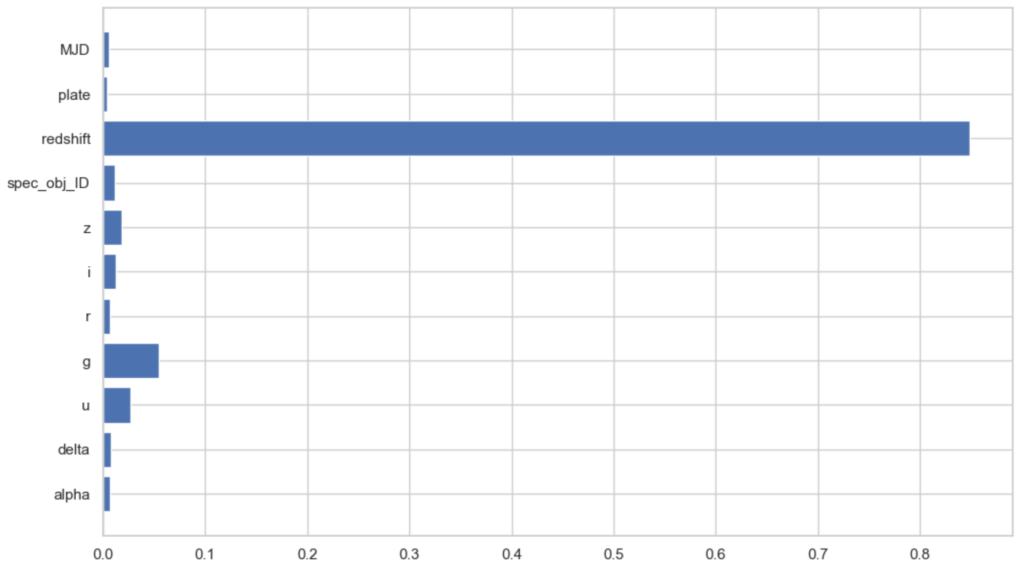
Models and Performance Evaluation

The first model used is Decision Tree. A decision tree model consists of a number of nodes, or conditions, which are joined by branches, or the results of the conditions of the nodes. Each node has two or more branches leading to additional nodes or leaf nodes that begin at the root node (the resulting predictions/classification). The feature variables and their values were used to create the conditions for the nodes in the decision tree, and the Gini index was used to determine where each node should be located. The Gini index measures the heterogeneity/impurity of the data as a result of the separation of the data by the condition, and the smallest value was chosen for each node. Additionally produced were the tree visualization and its confusion matrix.

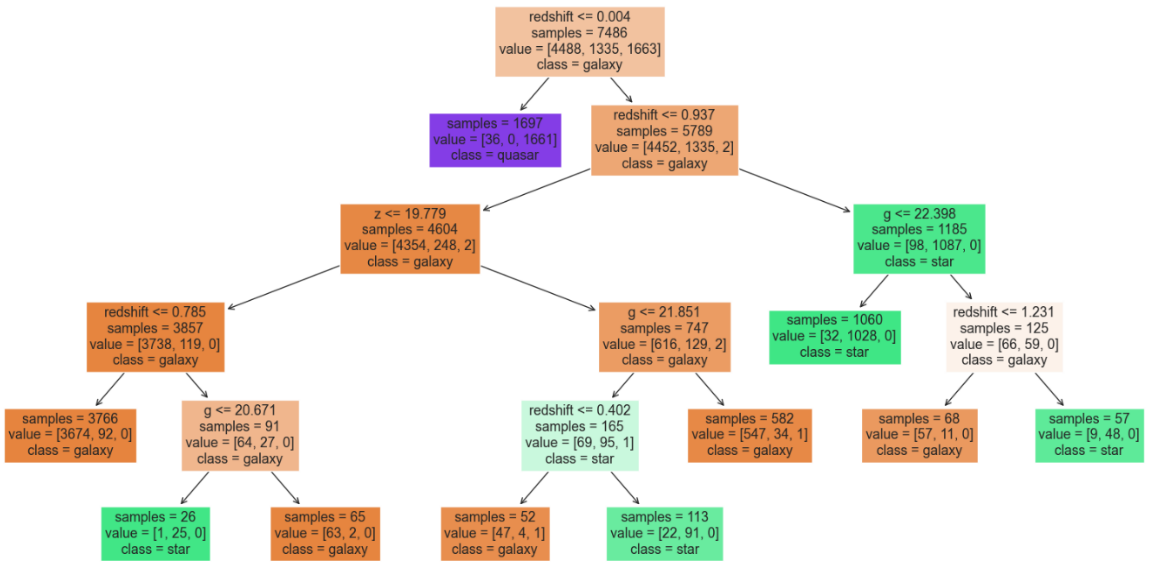
The second Model used is Random Forest. This is an algorithm for collective tree- based learning and is the random forest classifier. A collection of decision trees was drawn from a randomly chosen portion of the training set make up the random forest classifier. In order to determine the final class of the test object, it combined the votes from many decision trees. Additionally, the algorithm was conceived as a confusion matrix to help in classification.

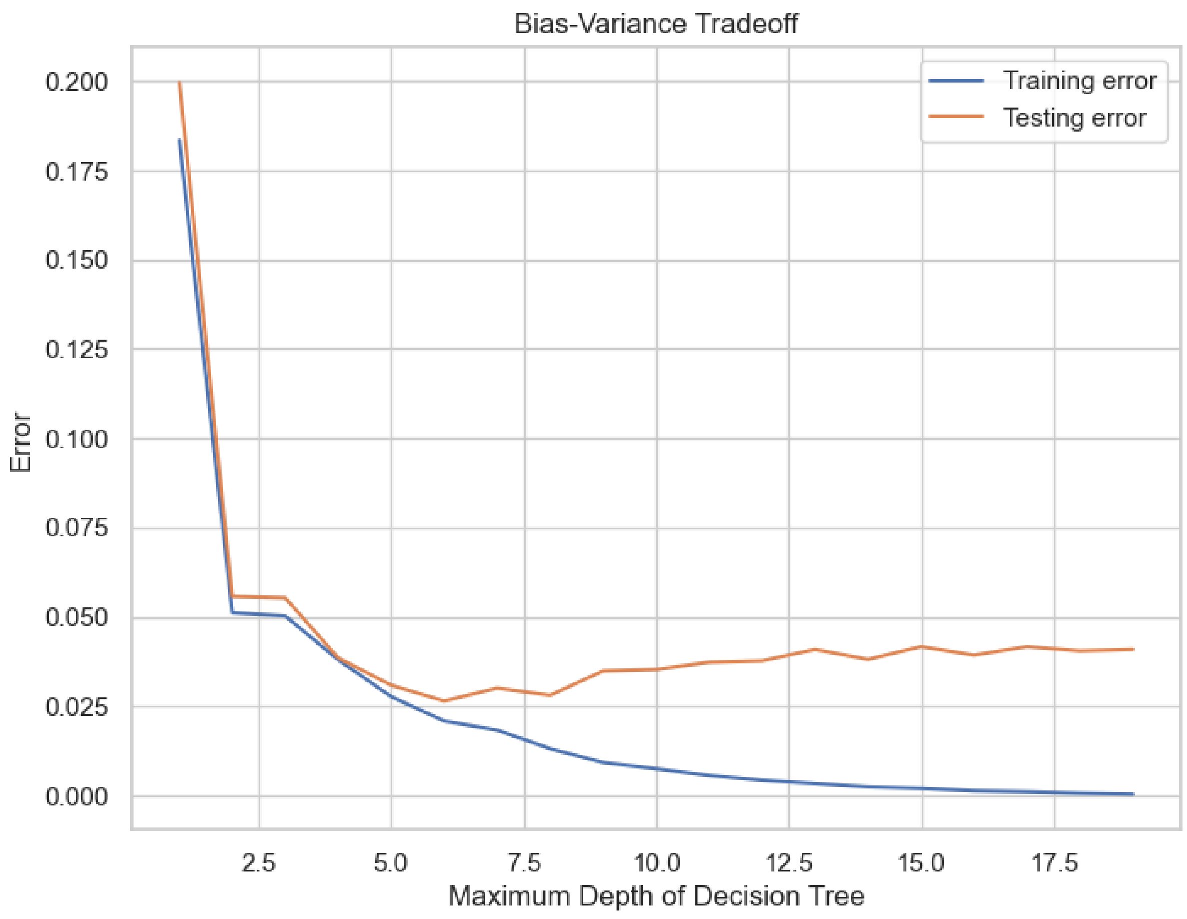
The last model used is Support Vector Machine. SVM is particularly effective in high-dimensional spaces and is used to find a hyperplane that best separates the data into different classes in feature space

1. Decision Tree



```
In [40]: # Report the best hyperparameters chosen
         grid_tree.best_params_

Out[40]: {'max_depth': 6, 'max_leaf_nodes': 10, 'min_samples_split': 2}
```

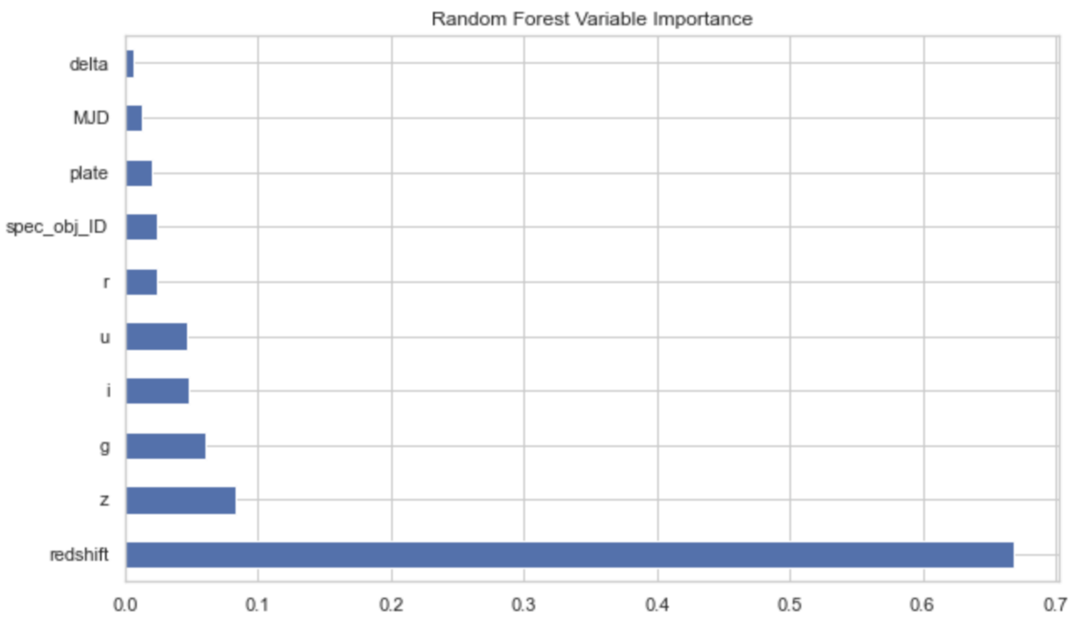


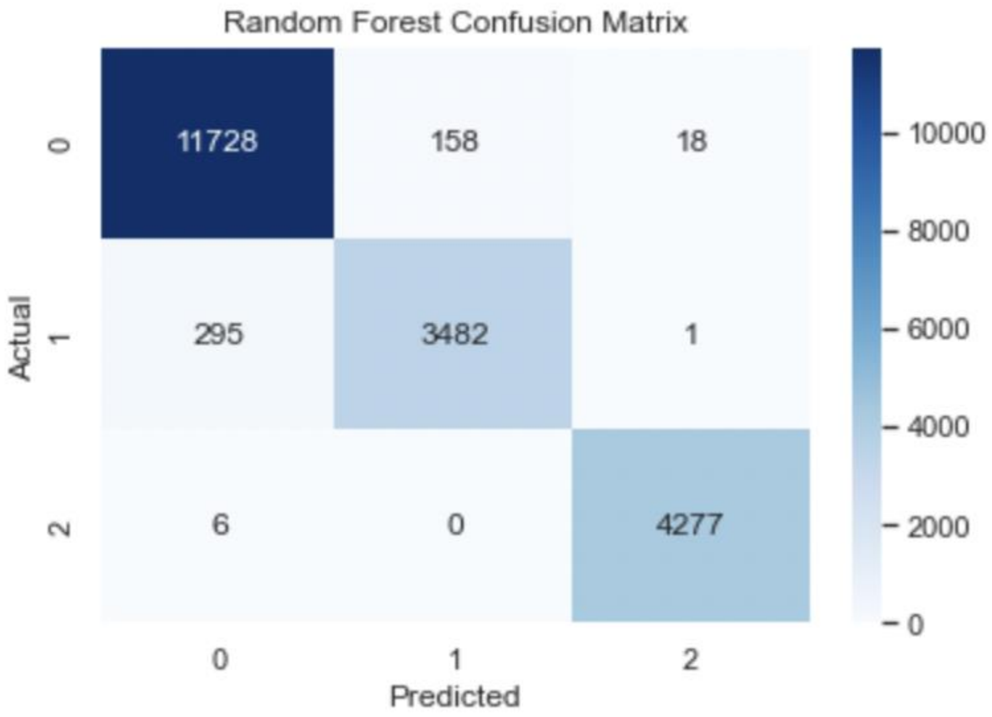


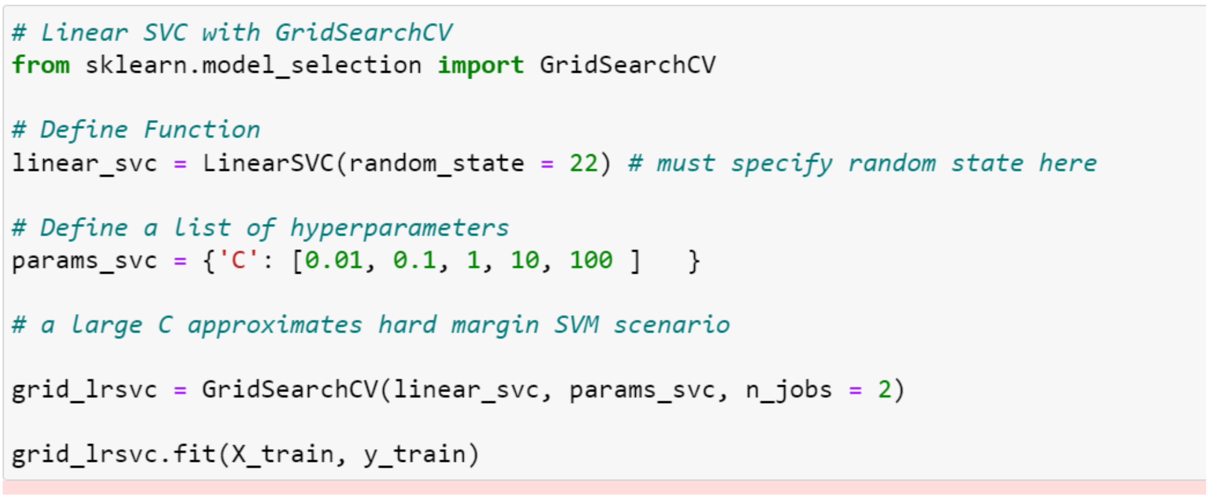
2. Random Forest

```
print("The training accuracy is: {:.2f}%".format(raf.score(X_train, y_train) * 100))
print("The test accuracy is: {:.2f}%".format(raf.score(X_test, y_test) * 100))
```

```
The training accuracy is: 99.90%
The test accuracy is: 97.61%
```

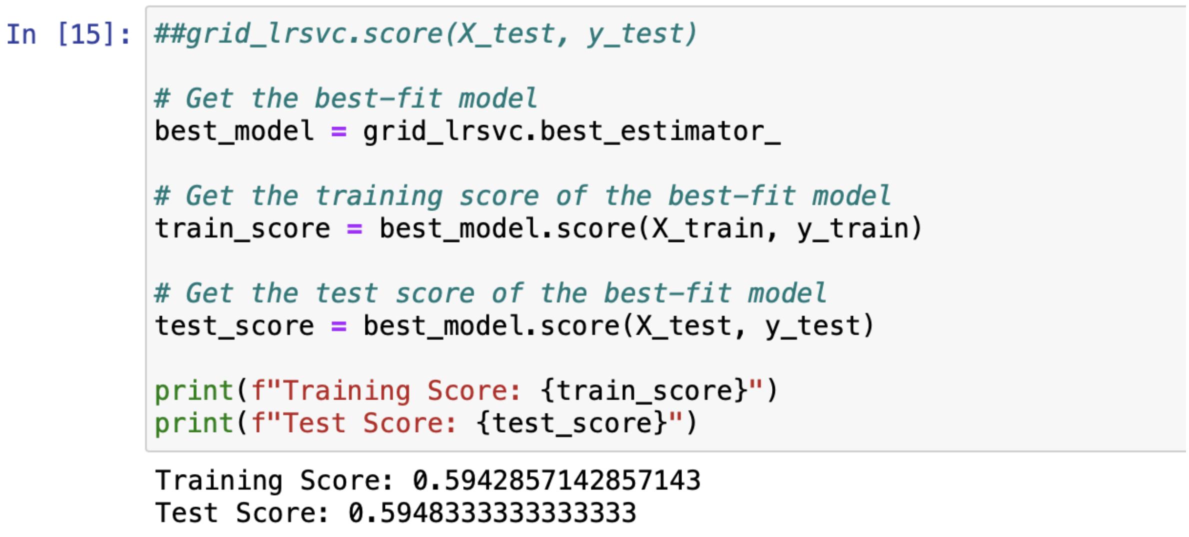

Random Forest Variable Importance
delta
MJD
plate
spec_obj_ID
r
u
i
g
z
redshift
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7


Random Forest Confusion Matrix
11728
158
18
295
3482
1
6
0
4277
0
1
2
Actual
0
1
2
Predicted
10000
8000
6000
4000
2000
0

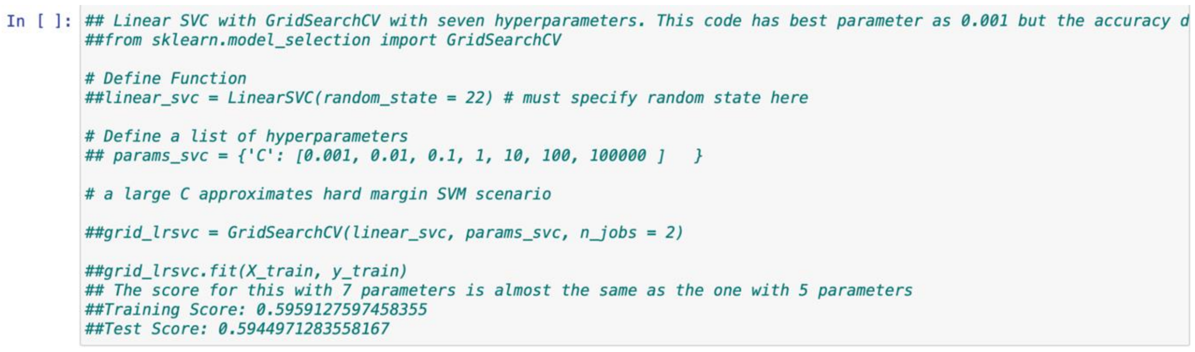


```
# Linear SVC with GridSearchCV
from sklearn.model_selection import GridSearchCV

# Define Function
linear_svc = LinearSVC(random_state = 22) # must specify random state here

# Define a list of hyperparameters
params_svc = {'C': [0.01, 0.1, 1, 10, 100 ]    }

# a large C approximates hard margin SVM scenario

grid_lrsvc = GridSearchCV(linear_svc, params_svc, n_jobs = 2)

grid_lrsvc.fit(X_train, y_train)
```

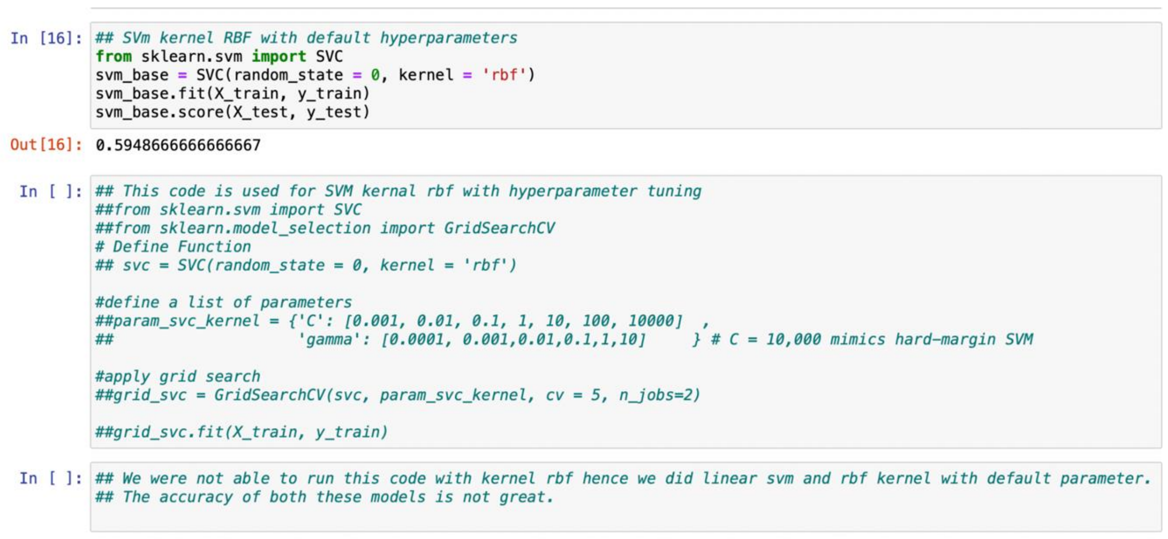
3. Support Vector Machine



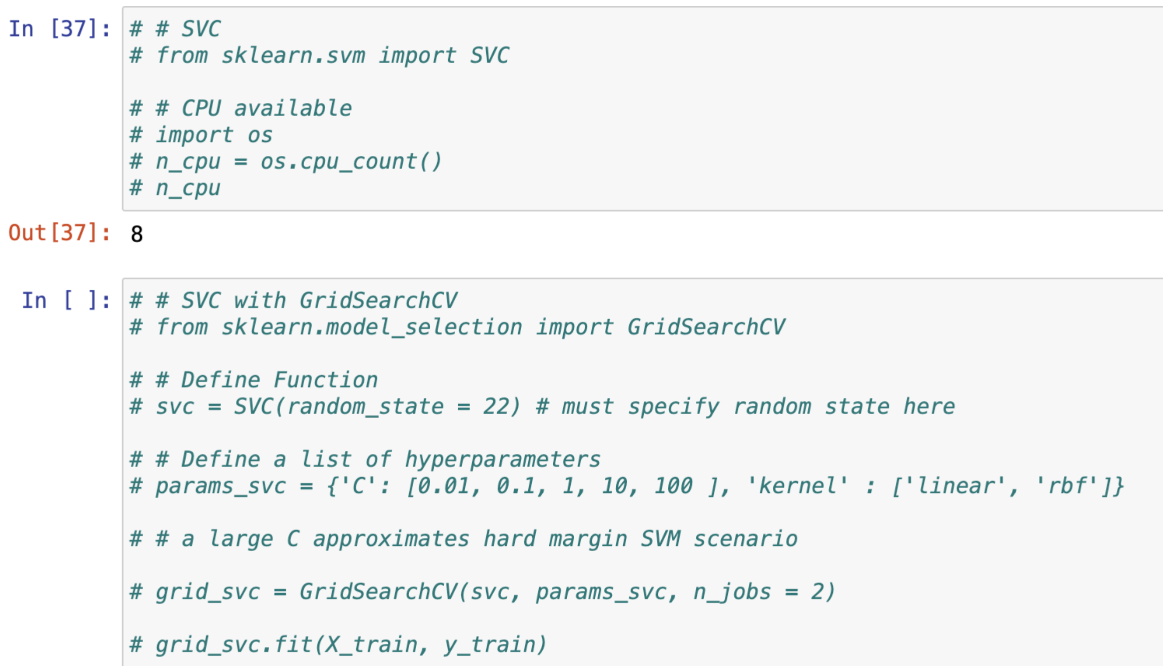
```
In [15]: ##grid_lrsvc.score(X_test, y_test)

         # Get the best-fit model
         best_model = grid_lrsvc.best_estimator_

         # Get the training score of the best-fit model
         train_score = best_model.score(X_train, y_train)

         # Get the test score of the best-fit model
         test_score = best_model.score(X_test, y_test)

         print(f"Training Score: {train_score}")
         print(f"Test Score: {test_score}")

         Training Score: 0.5942857142857143
         Test Score: 0.5948333333333333
```



```
In [ ]: ## Linear SVC with GridSearchCV with seven hyperparameters. This code has best parameter as 0.001 but the accuracy d
        ##from sklearn.model_selection import GridSearchCV

        # Define Function
        ##linear_svc = LinearSVC(random_state = 22) # must specify random state here

        # Define a list of hyperparameters
        ## params_svc = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 100000 ]    }

        # a large C approximates hard margin SVM scenario

        ##grid_lrsvc = GridSearchCV(linear_svc, params_svc, n_jobs = 2)

        ##grid_lrsvc.fit(X_train, y_train)
        ## The score for this with 7 parameters is almost the same as the one with 5 parameters
        ##Training Score: 0.5959127597458355
        ##Test Score: 0.5944971283558167
```



```
In [16]: ## SVm kernel RBF with default hyperparameters
         from sklearn.svm import SVC
         svm_base = SVC(random_state = 0, kernel = 'rbf')
         svm_base.fit(X_train, y_train)
         svm_base.score(X_test, y_test)

Out[16]: 0.5948666666666667

In [ ]:  ## This code is used for SVM kernal rbf with hyperparameter tuning
         ##from sklearn.svm import SVC
         ##from sklearn.model_selection import GridSearchCV
         # Define Function
         ## svc = SVC(random_state = 0, kernel = 'rbf')

         #define a list of parameters
         ##param_svc_kernel = {'C': [0.001, 0.01, 0.1, 1, 10, 100, 10000]  ,
         ##                    'gamma': [0.0001, 0.001,0.01,0.1,1,10]     } # C = 10,000 mimics hard-margin SVM

         #apply grid search
         ##grid_svc = GridSearchCV(svc, param_svc_kernel, cv = 5, n_jobs=2)

         ##grid_svc.fit(X_train, y_train)

In [ ]:  ## We were not able to run this code with kernel rbf hence we did linear svm and rbf kernel with default parameter.
         ## The accuracy of both these models is not great.
```



```
In [37]: # # SVC
         # from sklearn.svm import SVC

         # # CPU available
         # import os
         # n_cpu = os.cpu_count()
         # n_cpu

Out[37]: 8

In [ ]:  # # SVC with GridSearchCV
         # from sklearn.model_selection import GridSearchCV

         # # Define Function
         # svc = SVC(random_state = 22) # must specify random state here

         # # Define a list of hyperparameters
         # params_svc = {'C': [0.01, 0.1, 1, 10, 100 ], 'kernel' : ['linear', 'rbf']}

         # # a large C approximates hard margin SVM scenario

         # grid_svc = GridSearchCV(svc, params_svc, n_jobs = 2)

         # grid_svc.fit(X_train, y_train)
```

Conclusion

The Three models were fitted to the training data and evaluated using testing sample for a range of performance criteria. When it comes to classifying stars, galaxies and quasars the random forest model developed in this project has a best accuracy, can be used as a competent and reliable classification tool.

Accuracy	Decision Tree	Random Forest	SVM
Training	99.11	99.44	59.48
Testing	96	97.61	59.42