**Space Server Dataset**

**Introduction**

**Predicting Stars, Galaxies and Quasars**

**1. Star -** is a type of astronomical object consisting of a luminous spheroid of plasma which is held together by its own gravity. It is a luminous fixed point in the night sky which has a large remote incandescent body just like Sun. The nearest star to Earth is the Sun.

2. **Galaxy** - is a gravitationally bound system of stars, stellar remnants, interstellar gas, dust, and dark matter. Galaxies are categorized according to their visual morphology as elliptical, spiral, or irregular. Many galaxies are thought to have super massive black holes at their active centers. It is a system of millions or billions of stars, together with gas and dust, held together by gravitational attraction.



**3. Quasars -** A quasar is an extremely luminous active galactic nucleus, in which a super massive black hole with mass ranging from millions to billions of times the mass of the Sun is surrounded by a gaseous accretion disk. It is a massive and extremely remote celestial object, emitting exceptionally large amounts of energy and typically having a star like image in a telescope. It has been suggested that quasars contain massive black holes and may represent a stage in the evolution of some galaxies.

**Problem Statement**

# The data consists of 10,000 observations of space taken by the SDSS. Every observation is described by 17 feature columns and 1 class column which identifies it to be either a star, galaxy or quasar. The data released by the SDSS is under public domain. It’s taken from the current data release **RD14**. The dataset offers plenty of information about space to explore.

# The class identifies an object to be galaxy, star or quasar. Also the class column is the perfect target for “classification practices”.

**Data Description**

Some relavant details of individual columns

1. objid - Object Identifier
2. ra - J2000 Right Ascension (r-band)
3. dec - J2000 Declination (r-band)
4. u - better of DeV/Exp magnitude fit
5. g - better of DeV/Exp magnitude fit
6. r - better of DeV/Exp magnitude fit
7. i - better of DeV/Exp magnitude fit
8. z - better of DeV/Exp magnitude fit
9. run - Run Number
10. rerun - Rerun Number
11. camcol - Camera column
12. field - Field number
13. specobjid - Object Identifier
14. class - object class (galaxy, star or quasar object)
15. redshift - Final Redshift
16. plate - plate number
17. mjd - MJD of observation
18. fiberid - fiber ID

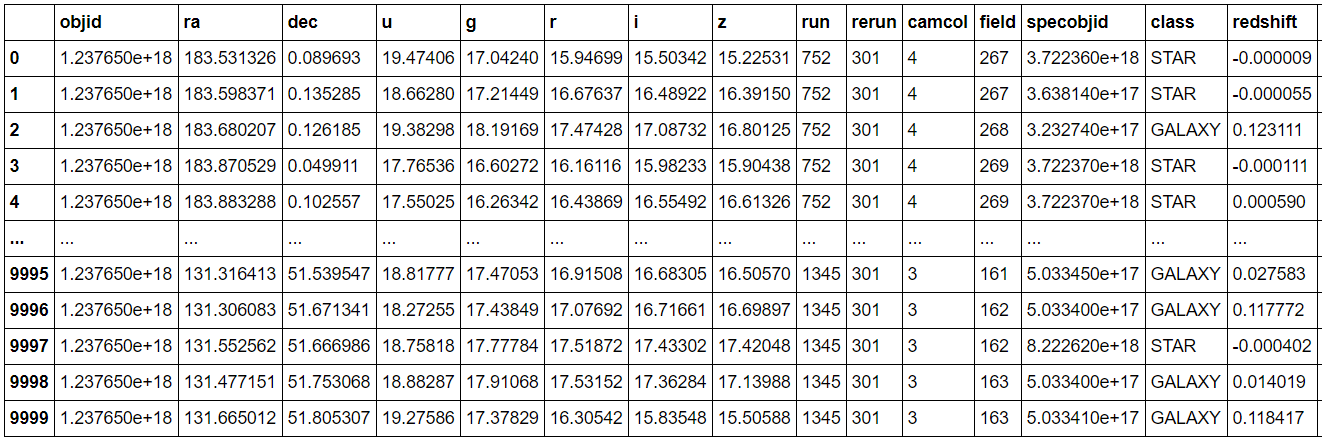
Now let’s load the dataset and do the analysis.

**Importing Libraries**

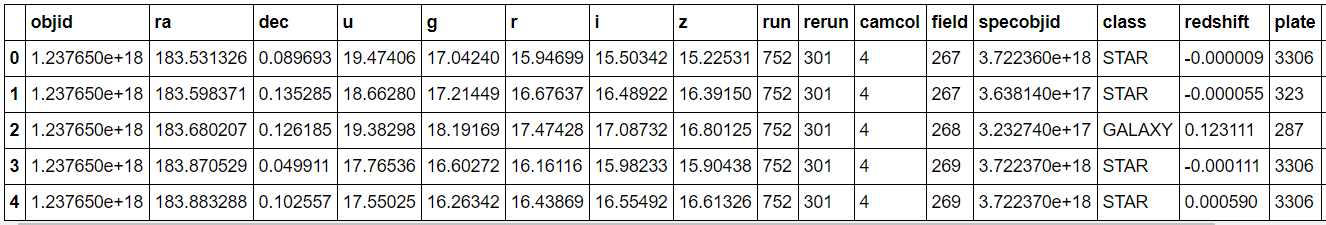
* **import** pandas as pd
* **import** numpy as np
* **import** seaborn as sns
* **import** matplotlib.pyplot as plt
* **import** warnings
* warnings.filterwarnings('ignore')

**Loading the dataset**

* ds=pd.read\_csv('Skyserver.csv')
* ds

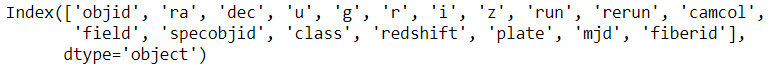


* df=pd.DataFrame(ds)
* df.head()

****

* #Checking the columns available in the dataset.

* df.columns

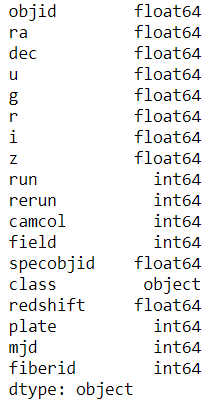
****

* #Checking the shape of dataset
* ds.shape

****

**Checking the data type of dataset**

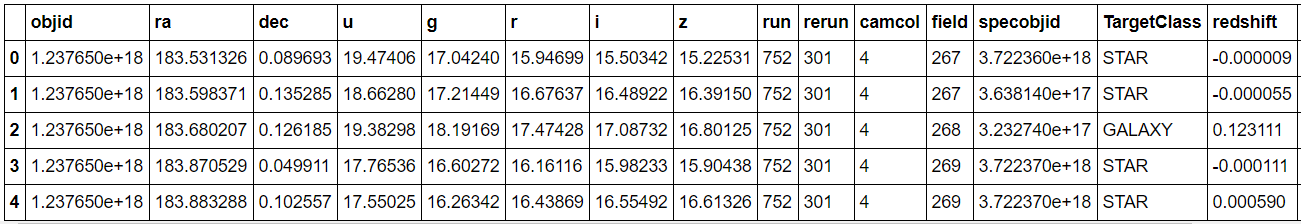
* #Checking the data types of dataset
* df.dtypes

****

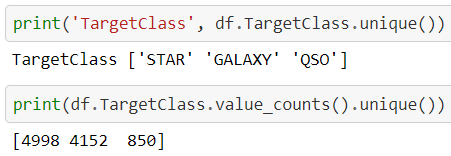
In the dataset class column is in object format, run, rerun, camcol, field, specobjid, plate, mjd, fibered columns are in int64 format and remaining columns are in float64 format.

For further analysis let’s change the name of the class column into TargetClass.

* #changing the "class" column name
* df.rename(columns={'class': 'TargetClass'}, inplace=True)
* df.head()

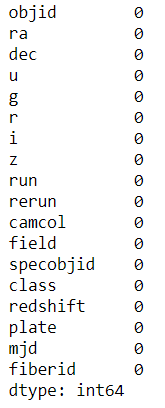


To check the unique values and value\_counts in the target column (TargetClass)



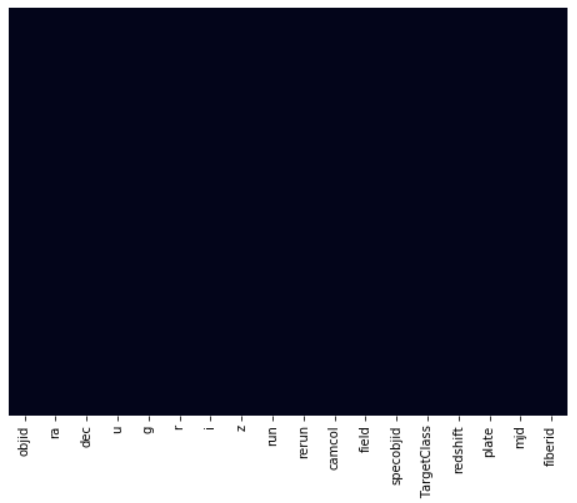
**Tracing Null values in data frame.**

* #Changing the null values in data frame.
* df=pd.DataFrame(ds)
* df.isnull().sum()



**Tracing Null values with the help of heat map.**

* #seeing missing value via visualization
* plt.figure(figsize=(8, 6))
* sns.heatmap(df.isnull(),yticklabels=False, cbar=False)

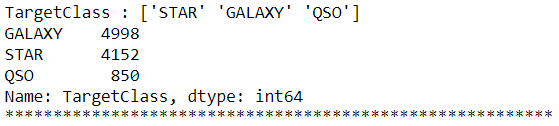


**Observations**

1. In the Skyserver dataset we have 10 float64 data type, 7 int64 data type and 1 obect type column.
2. There are no null values present in the dataset.

**Exploring the categorical values**

* #Printing the object datatypes and their unique values
* **for** column **in** df.columns:
* **if** df[column].dtype==object:
* **print**(str(column) + ' : ' +str(df[column].unique()))
* **print**(df[column].value\_counts())
* **print**('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
* **print**('\n')

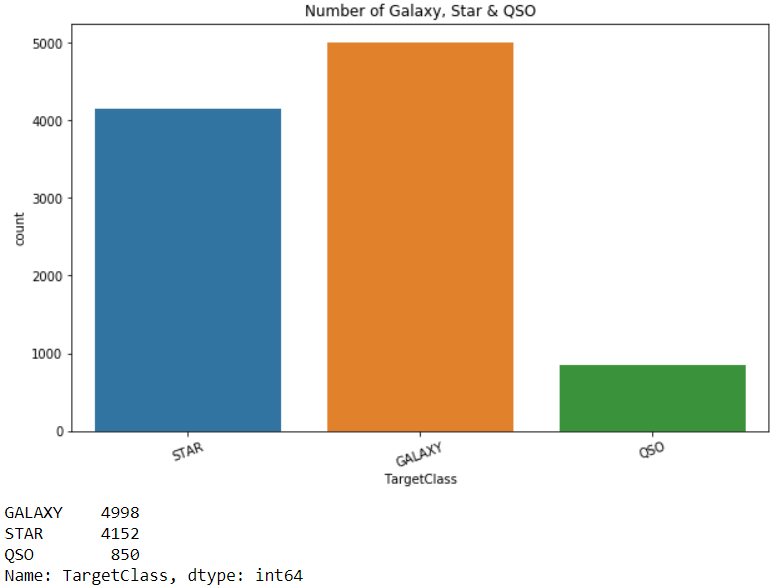


**Exploratory Data Analysis (EDA)**

Using EDA we can understand the relation between different columns

1**. Number of Galaxy, Star & QSO in TargetClass**

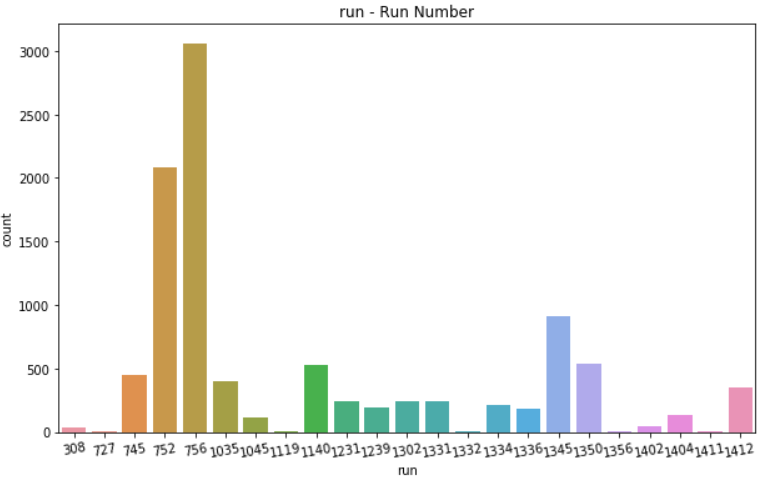
* plt.subplots(figsize=(10,6))
* sns.countplot(df['TargetClass'])
* plt.title("Number of Galaxy, Star & QSO")
* plt.xticks(rotation=20)
* plt.show()
* **print**(df['TargetClass'].value\_counts())



In the above graph we can see that there are total of 4998 count of Galaxy, 4152 count of Star and 850 count of Quasar are present.

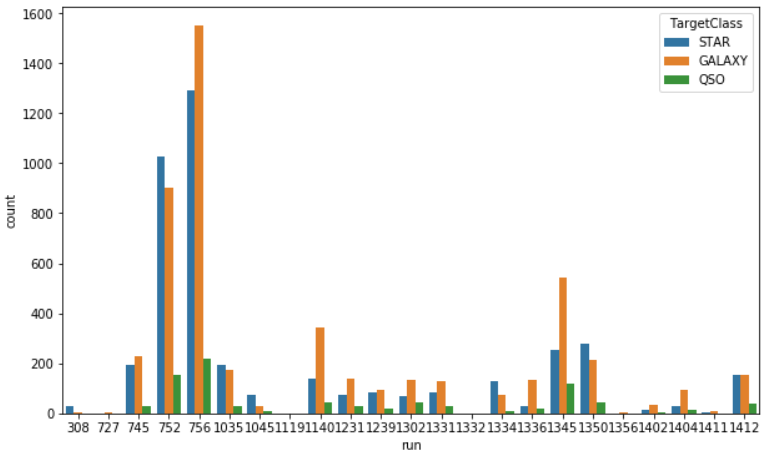
**2. Count plot of run column**

* #Count plot of run column
* plt.subplots(figsize=(10,6))
* sns.countplot(x="run", data=df)
* plt.title("run - Run Number")
* plt.xticks(rotation=10)
* plt.show()



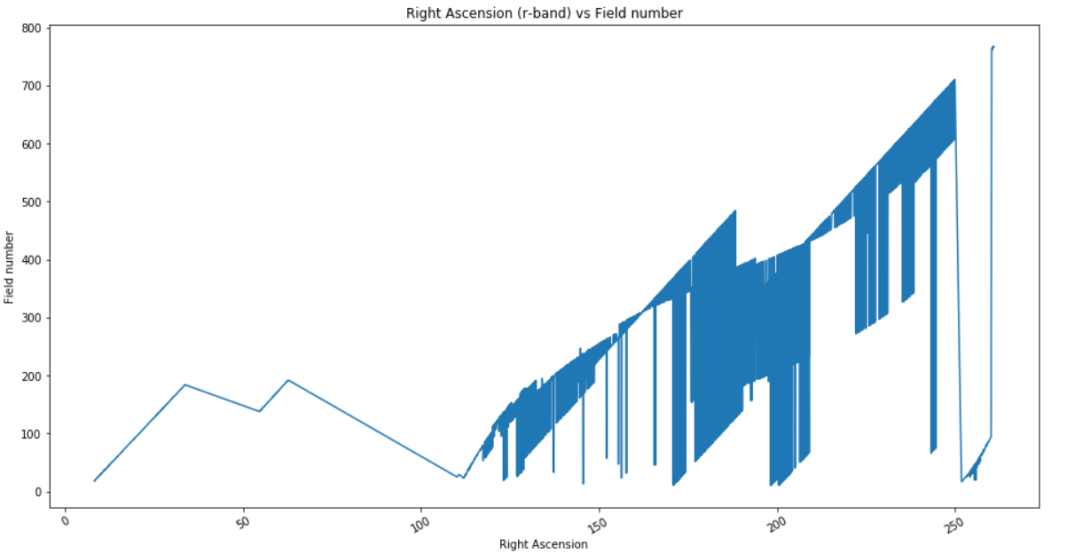
**3. Run vs TargetClass**

* #Count plot of run and TargetClass
* plt.figure(figsize=(10, 6))
* sns.countplot(x ='run', hue = "TargetClass", data = df)
* plt.show()



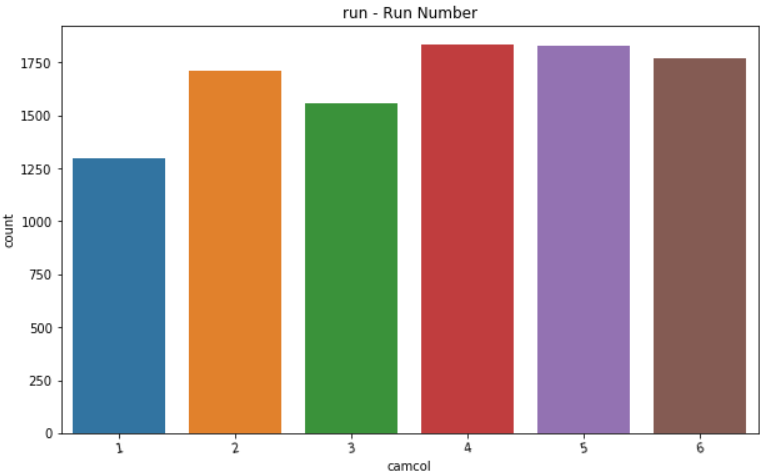
**4. Right Ascension (r-band) vs Field number**

* #Right Ascension (r-band) vs Field number
* plt.figure(figsize=(16, 8))
* sns.lineplot(x="ra", y="field", data=df)
* plt.ylabel('Field number')
* plt.xlabel('Right Ascension')
* plt.xticks(rotation=30)
* plt.title("Right Ascension (r-band) vs Field number")
* plt.show()



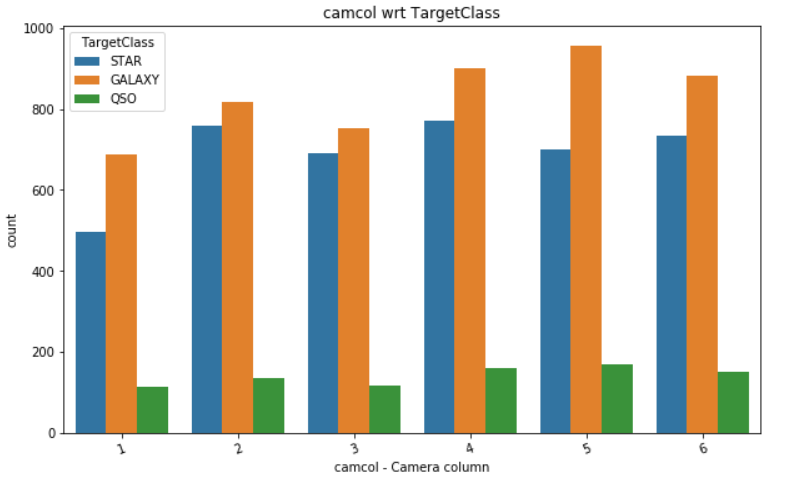
**5. Count plot of camcol**

* #Count plot of camcol
* plt.subplots(figsize=(10,6))
* sns.countplot(x="camcol", data=df)
* plt.title("run - Run Number")
* plt.xticks(rotation=10)
* plt.show()
* **print**(df.run.value\_counts())



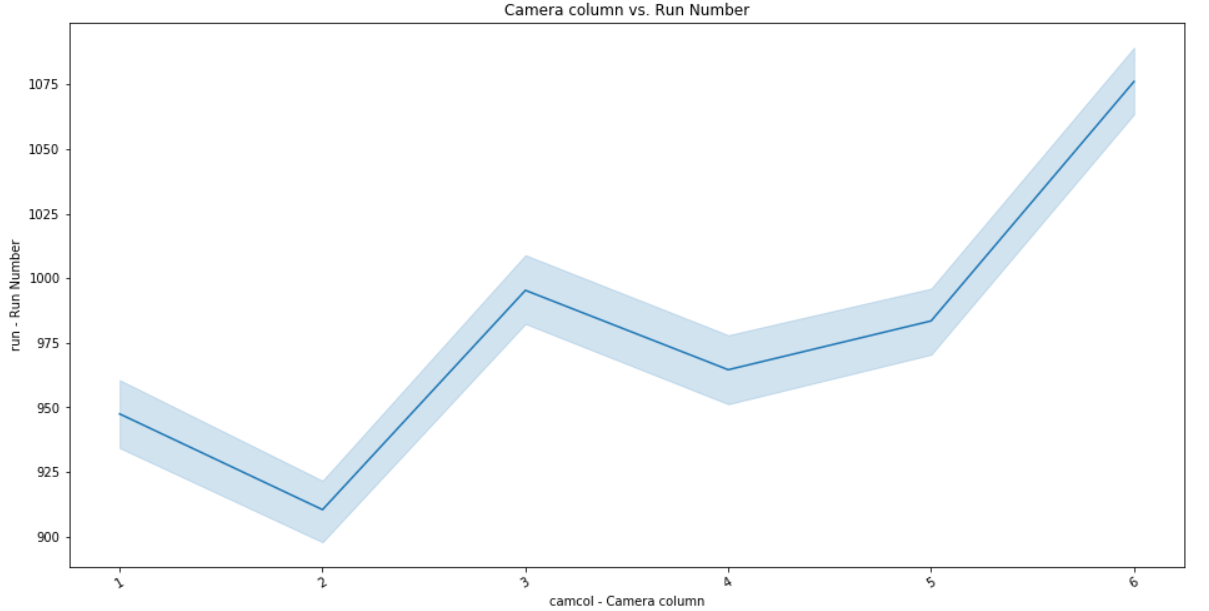
**6. Camcol vs TargetClass**

* #camcol vs TargetClass
* plt.figure(figsize=(10, 6))
* sns.countplot(x ='camcol', hue = "TargetClass", data = df)
* plt.xticks(rotation=20)
* plt.xlabel('camcol - Camera column')
* plt.title("camcol wrt TargetClass")
* plt.show()



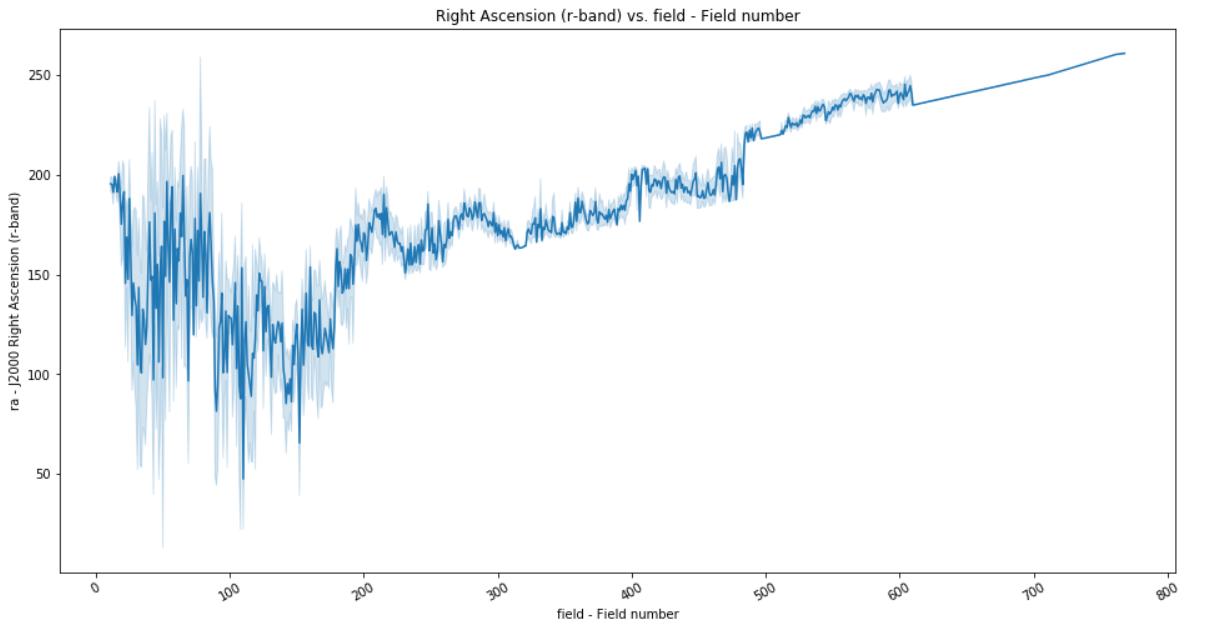
**7. Camera column vs. Run Number**

* #camcol vs Run Number
* plt.figure(figsize=(16, 8))
* sns.lineplot(x="camcol", y="run", data=df)
* plt.ylabel('run - Run Number')
* plt.xlabel('camcol - Camera column')
* plt.xticks(rotation=30)
* plt.title("Camera column vs. Run Number")
* plt.show()



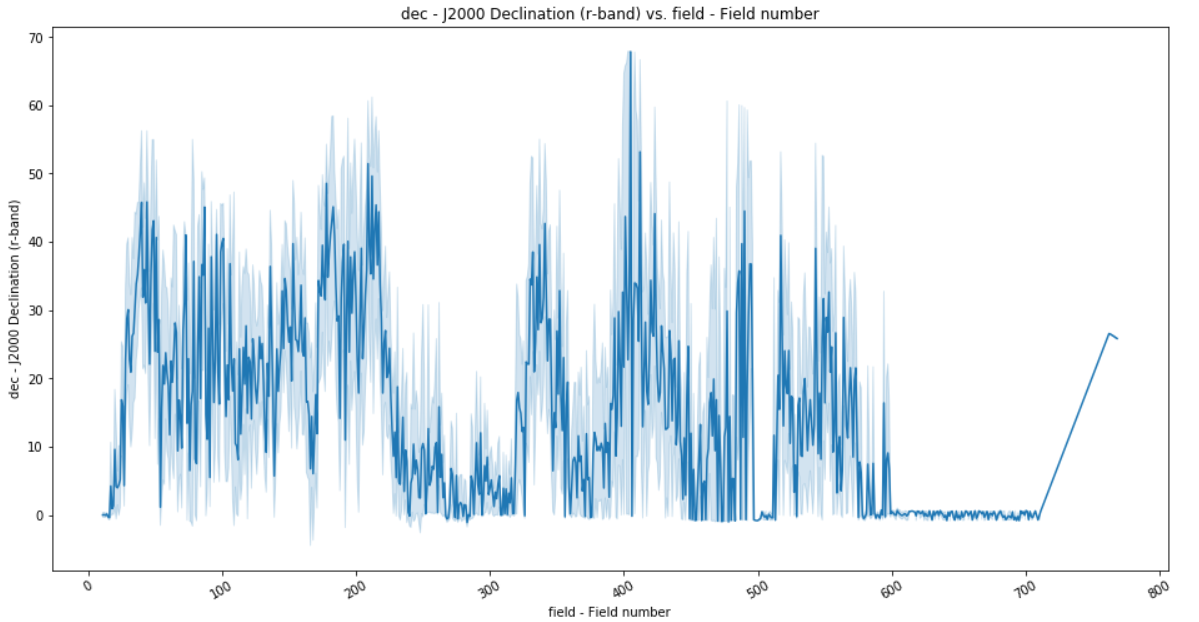
**8. Right Ascension (r-band) vs. field - Field number**

* #Right Ascension (r-band) vs. field - Field number
* plt.figure(figsize=(16, 8))
* sns.lineplot(x="field", y="ra", data=df)
* plt.ylabel('ra - J2000 Right Ascension (r-band)')
* plt.xlabel('field - Field number')
* plt.xticks(rotation=30)
* plt.title("Right Ascension (r-band) vs. field - Field number")
* plt.show()



9. **dec - J2000 Declination (r-band) vs. field - Field number**

* #dec - J2000 Declination (r-band) vs. field - Field number
* plt.figure(figsize=(16, 8))
* sns.lineplot(x="field", y="dec", data=df)
* plt.ylabel('dec - J2000 Declination (r-band)')
* plt.xlabel('field - Field number')
* plt.xticks(rotation=30)
* plt.title("dec - J2000 Declination (r-band) vs. field - Field number")
* plt.show()

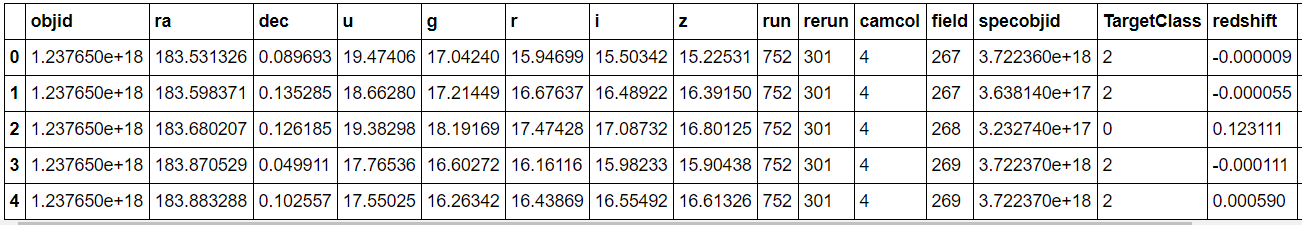


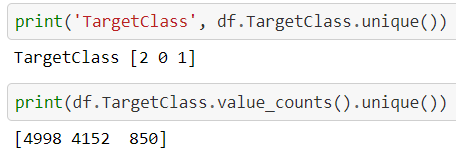
Now let’s replace the categorical values (alphabetic values) to numeric values using Label Encoder.

**Label Encoder** - refers to converts the labels into numeric form i.e the columns which are in alphabetical or categorical values are assigned with numbers so as to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated during the process.

Import label encoder for converting categorical data (object) into numeric(int 64) data.

* # Import label encoder
* # Now lets replace the categorical values (alphabetic values) to numeric values using Label Encoder
* **from** sklearn **import** preprocessing
* # label\_encoder object knows how to understand word labels.
* label\_encoder = preprocessing.LabelEncoder()
* # Lets Encode labels in columns
* df['TargetClass']= label\_encoder.fit\_transform(df['TargetClass'])
* df.head()





Observations

1. Dropping the 'objid' column from the dataset because it is same across all the columns and

also in the problem statement it is clearly mentioned as "specobjid = Object Identifier".

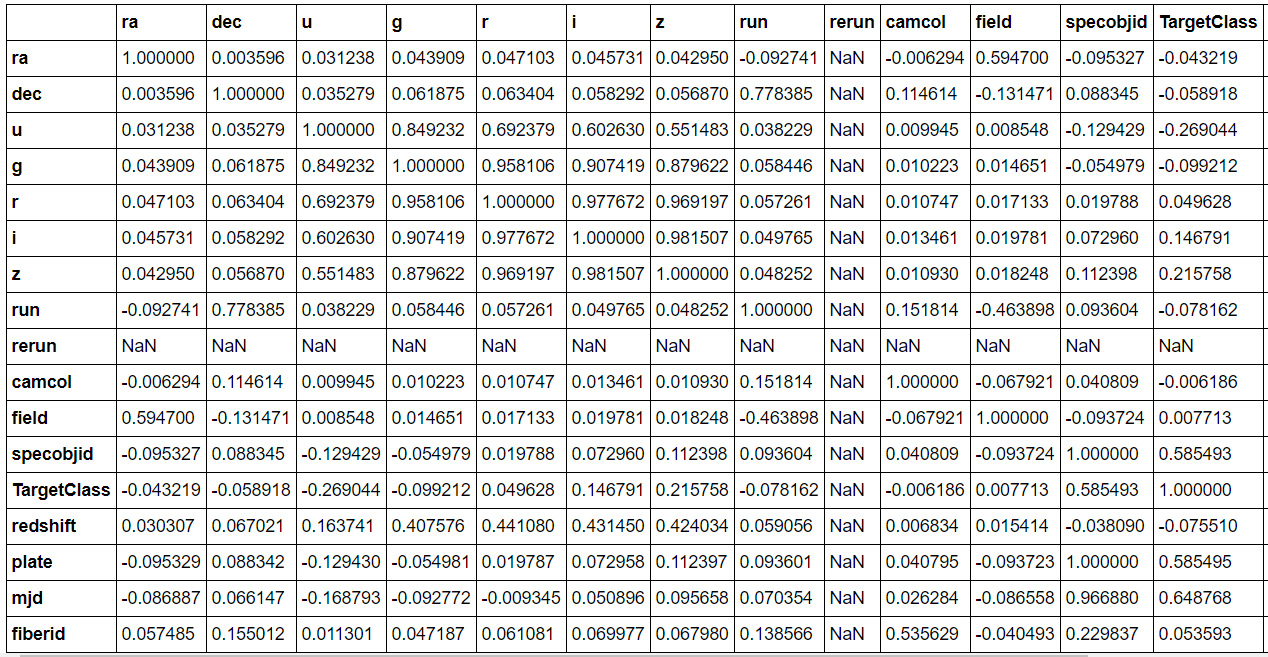
2. So the objid and specobjid both are similar, we can drop the objid column from dataset.

**Correlation Factor**

The statistical relationship between two variables is referred to as their correlation. The correlation factor represents the relation between columns in a given dataset. A correlation can be positive, meaning both variables are moving in the same direction or it can be negative, meaning that when one variable's value increasing, the other variable’s value is decreasing.

Now let’s check the correlation factor of the df dataset

* #The Correlation factor
* dfcor=df.corr()
* dfcor

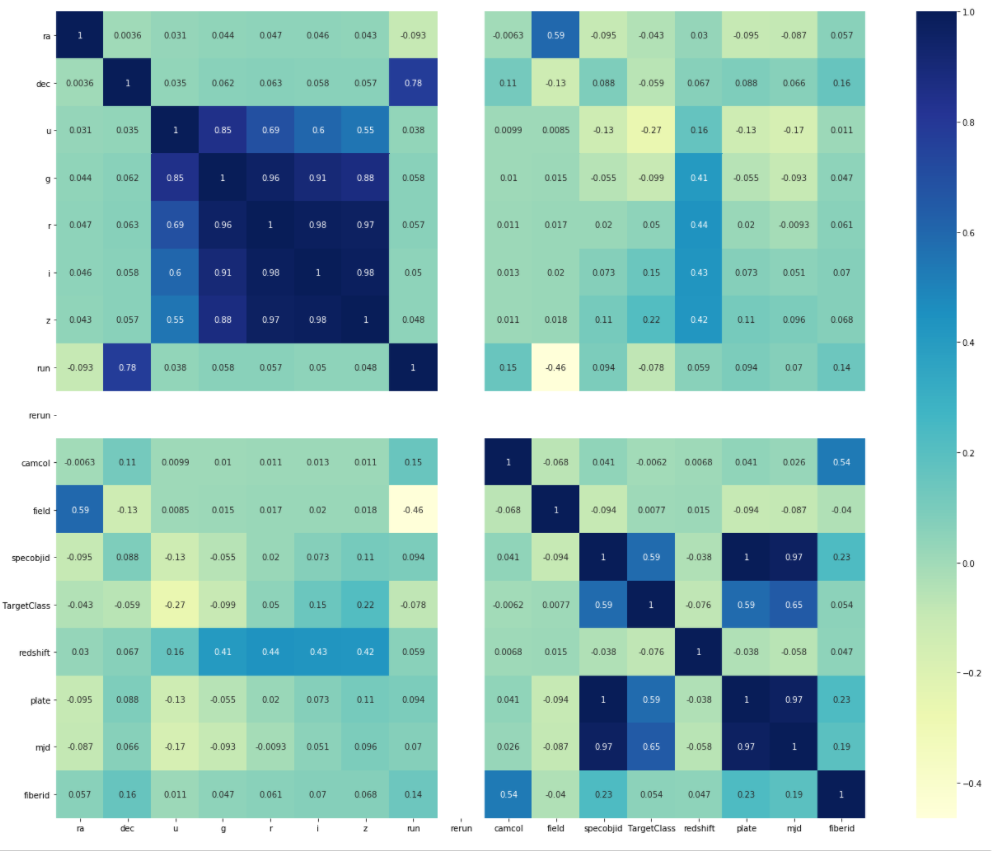


**Correlation Matrix**

A correlation matrix is a tabular data representing the ‘correlations’ between pairs of variables in a given dataset. It is also a very important pre-processing step in Machine Learning pipelines. The Correlation matrix is a data analysis representation that is used to summarize data to understand the relationship between various different variables of the given dataset.

**Correlation factor with visualization / Correlation matrix**

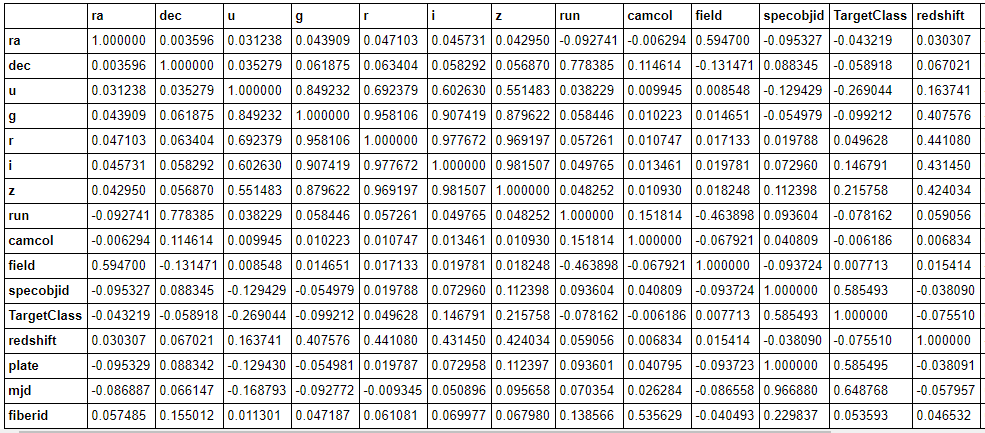
* #Now lets check the Correlation factor with visualization
* plt.figure(figsize=(22, 18))
* sns.heatmap(dfcor, cmap='YlGnBu', annot=True)



The rerun column is not contributing well in the dataset as it is same across the columns and it is not correlated to any of the other columns. Considering its nil contribution towards the model building, we can drop the "rerun" column for further analysis.

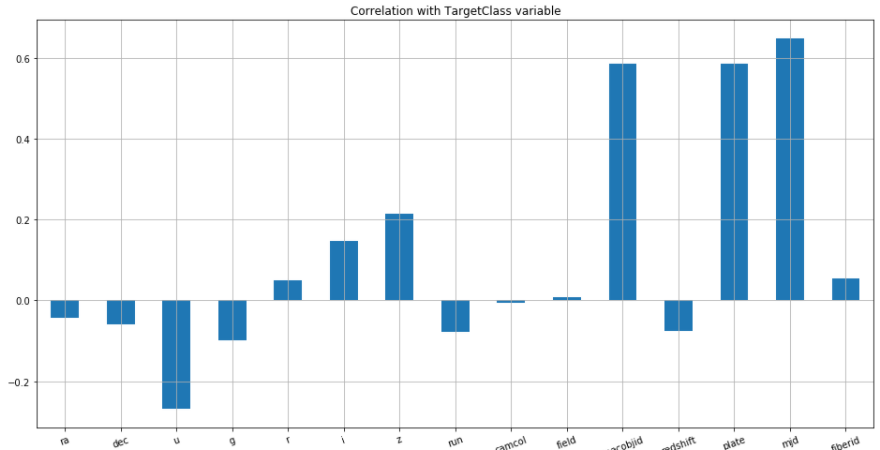
* # Lets drop rerun column
* df=df.drop(['rerun'], axis=1)

* #Now lets check the Correlation factor
* dfcor=df.corr()
* dfcor



**Checking Correlation with target (TargetClass) variable**

* #checking Correlation with TargetClass variable
* plt.figure(figsize=(16, 8))
* df.drop('TargetClass', axis=1).corrwith(df['TargetClass']).plot(kind='bar', grid=True)
* plt.xticks(rotation=20)
* plt.title("Correlation with Confirmed variable")



**Observation**

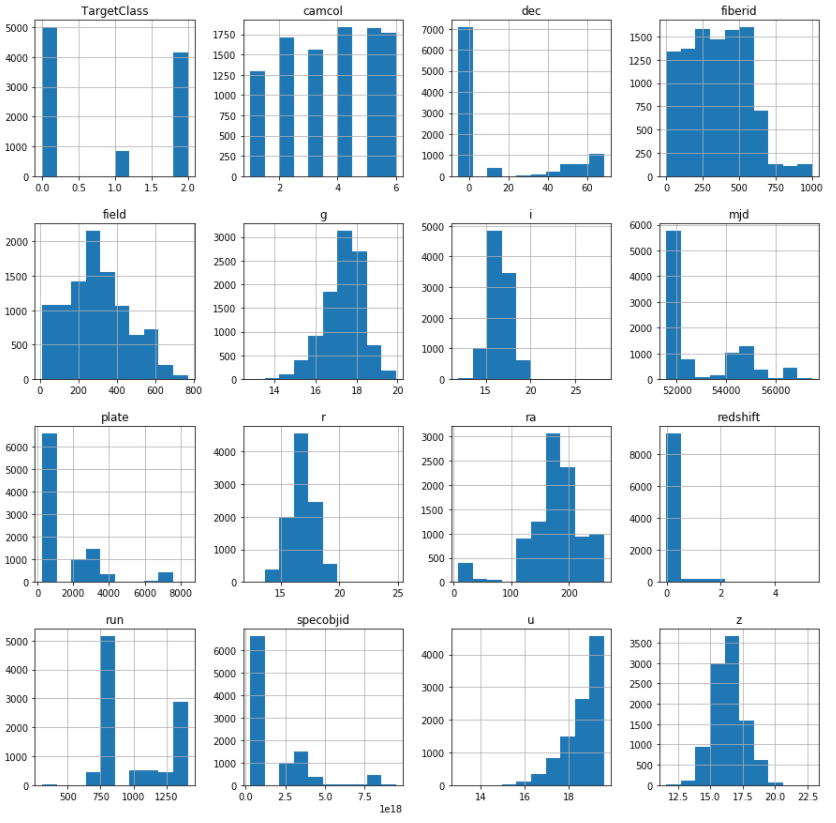
u, g, run & redshift are negetively correlated with target variable TargetClass

mjd, plate, spaceobjid is highly correlated with target variable among all input variables.

**Plotting Histogram**

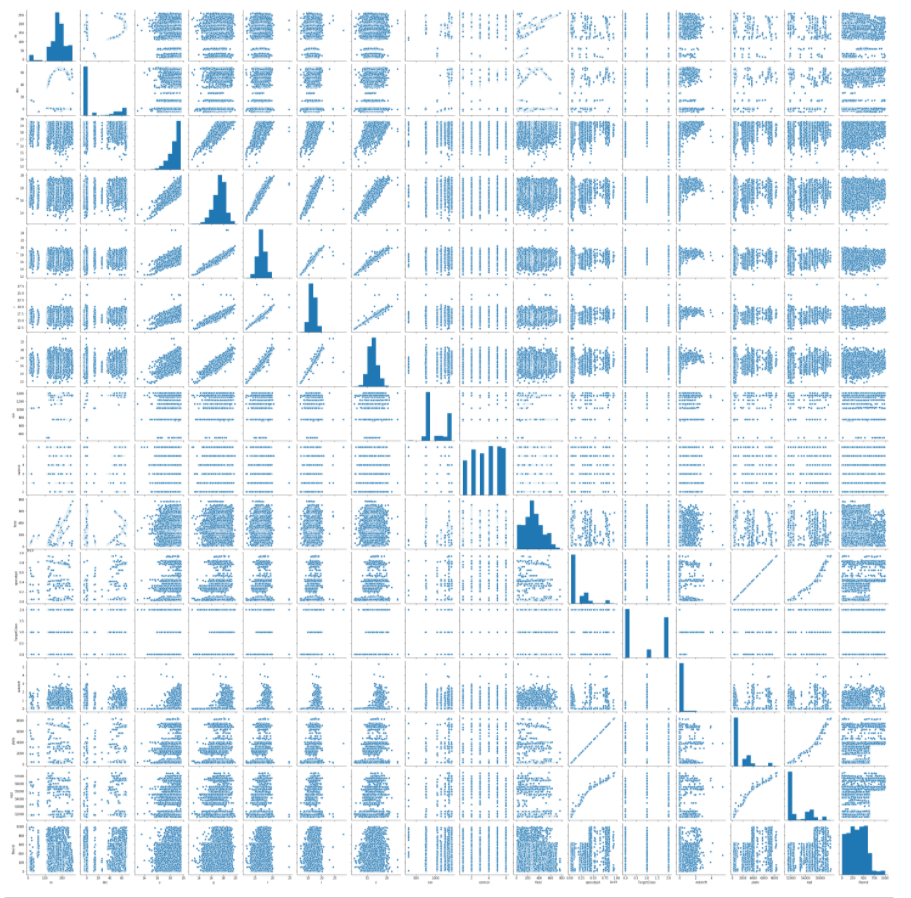
A histogram is used to represent data provided in a form of certain groups using various columns of the dataset. It is accurate method for the graphical representation of numerical data distribution in the given dataset.

* df.hist(figsize=(15,15))



**Pairplot**

A pair plot plots a pair wise relationship in a dataset. The pair plot function creates a grid of Axes such that each variable in data will by shared in the y-axis across a single row and in the x-axis across a single column. That creates plots as shown below for the given dataset. A pairs plot allows us to see both distribution of single variables and relationships between two variables.



**Outliers**

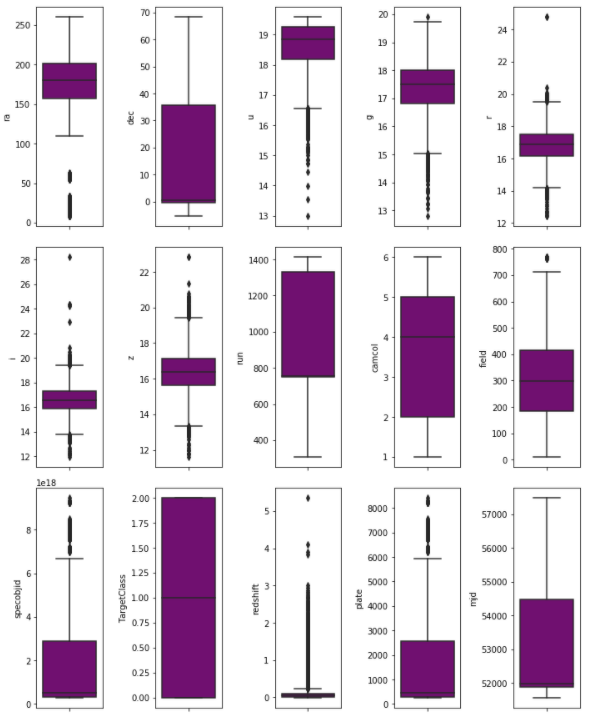
An outlier is a data point in a data set which is distant or far from all other observations available. It is a data point which lies outside the overall distribution which is available in the dataset. In statistics, an outlier is an observation point that is distant from other observations.

A box plot is a method or a process for graphically representing groups of numerical data through their quartiles. Outliers may also be plotted as an individual point. If there is an outlier it will plotted as point in box plot but other numerical data will be grouped together and displayed as boxes in the diagram. In most cases a threshold of 3 or -3 is used i.e if the Z-score value is higher than or less than 3 or -3 respectively, that particular data point will be identified as outlier.

**Plotting outliers in the Dataset**

* #boxplot to identify plotters
* collist=df.columns.values
* ncol=5
* nrow=5
* plt.figure(figsize=(10, 20))
* **for** i **in** range(0, len(collist)):
* plt.subplot(nrow, ncol, i+1)
* sns.boxplot(df[collist[i]], color='purple', orient='v')
* plt.tight\_layout()

**Representation of outliers in Box plot**

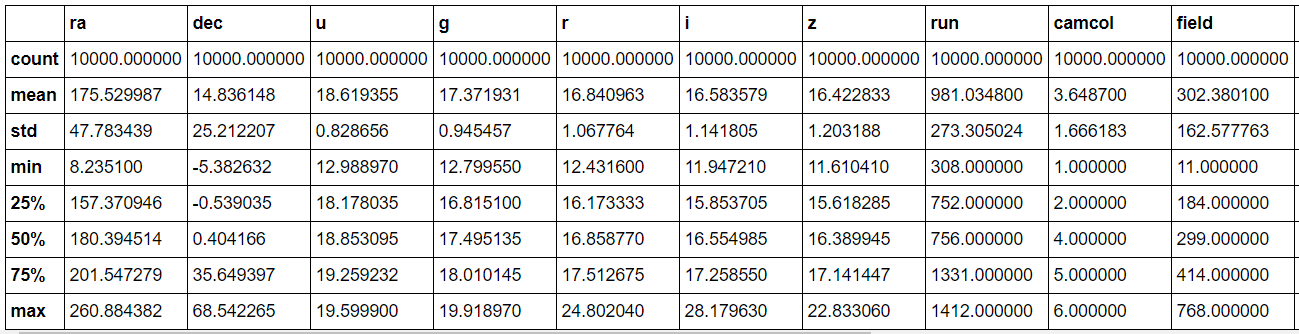


The above graph shows that outliers are present in “ra, u, g, r, i, z, field, specobjid, redshift, plate, mjd” columns which should be taken care using Z score method.

**Summary Statistics**

The describe() function computes a summary of statistics pertaining to the Data Frame columns. This function gives the mean, count, max, standard deviation and IQR values of the dataset in a simple understandable way.

* df.describe()



In the above columns we can see the value of Summary statistics.

1. The total count of all the columns is 10,000

2. In the ‘ra’ (J2000 Right Ascension) column maximum ‘ra’ obtained is 260.884382 & a minimum ‘ra’ obtained is 8.235100.

3. In the ‘dec’ (J2000 Declination) column maximum ‘dec’ obtained is 68.542265 & a minimum ‘dec’ obtained is -5.382632.

4. In the ‘u’ (better of DeV/Exp magnitude fit) column maximum ‘u’ obtained is 19.599900 & a minimum ‘u’ obtained is 12.988970.

5. In the ‘g’ (better of DeV/Exp magnitude fit) column maximum ‘g’ obtained is 19.918970 & a minimum ‘g’ obtained is 12.799550.

6. In the ‘r’ (better of DeV/Exp magnitude fit) column maximum ‘r’ obtained is 24.802040 & a minimum ‘r’ obtained is 12.431600.

7. In the ‘i’ (better of DeV/Exp magnitude fit) column maximum ‘i’ obtained is 28.179630 & a minimum ‘i’ obtained is 11.947210.

8. In the ‘z’ (better of DeV/Exp magnitude fit) column maximum ‘z’ obtained is 22.833060 & a minimum ‘g’ obtained is 11.610410.

9. In the ‘run’ column maximum ‘run’ obtained is 1412.00 & a minimum ‘run’ obtained is 308.00.

10. In the ‘camcol’ column maximum camera column is 6 & a minimum camera column is 1.

11. In the ‘field’ column maximum field number is 768 & a minimum field number is 11.

12. In the ‘redshift’ column maximum Final Red shift determined is 5.353854 & minimum Final Red shift determined is -0.004136.

13. In the plate column maximum plate number 8410.0 & minimum plate number is 266.

14. In the mjd (MJD of observation) column maximum mjd is 57481.00 & minimum mjd is 51578.00.

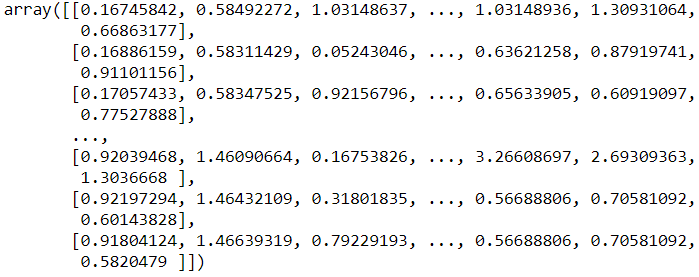
15. In the fiberid column maximum fiberid is determined as 1000 & minimum fiberid is determined as 1.

16. Describe table gives the min, max, std, count values of the different columns across the dataset & also it provides IQR values in 25%, 50% and 75% of the columns.

**Z – Score**

A Z-score is a numerical measurement that describes a value's relationship to the mean of a group of values in the dataset. Z-score is measured in terms of [standard deviations](https://www.investopedia.com/terms/s/standarddeviation.asp) from the mean.

* #Z score
* **from** scipy.stats **import** zscore
* z=np.abs(zscore(df))
* z
* threshold=3
* **print**(np.where(z>3))



* #Checking the shape of df dataset & df\_new dataset
* df\_new=df[(z<3).all(axis=1)]
* **print**(df.shape, '\t', df\_new.shape)



* #shape of df dataset
* df=df\_new
* **print**(df.shape)



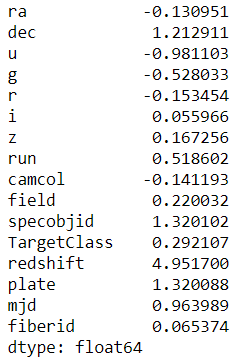
**To check distribution of Skewness**

**Skewness**

Skewness refers to distortion or asymmetry in a symmetrical bell curve, or [normal distribution](https://www.investopedia.com/terms/n/normaldistribution.asp) in a set of data. Besides positive and negative skew, distributions can also be said to have zero or undefined skew. The skewness value can be positive, zero, negative, or undefined.

**Check skewness in datset.**

* df.skew()

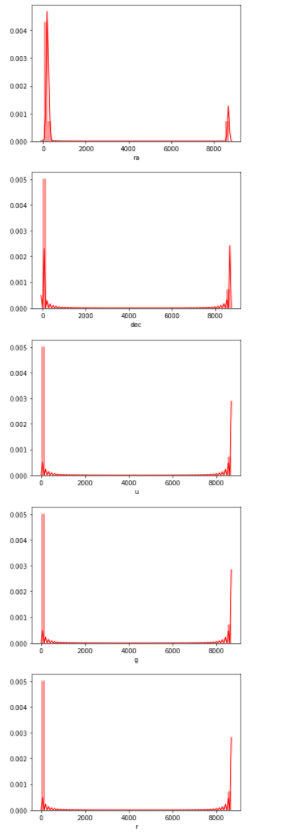


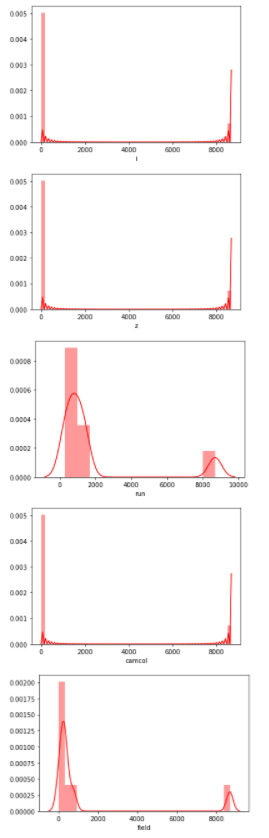
**Check skewness of dataset with visualization**

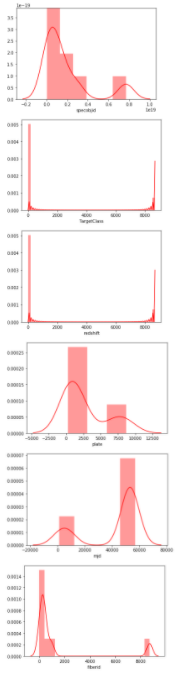
* #To check skewness of dataset with visualization

* **for** col **in** df.describe().columns:
* sns.distplot(df.describe()[col], color='r')
* plt.show()

**Checking skewness using distplot**







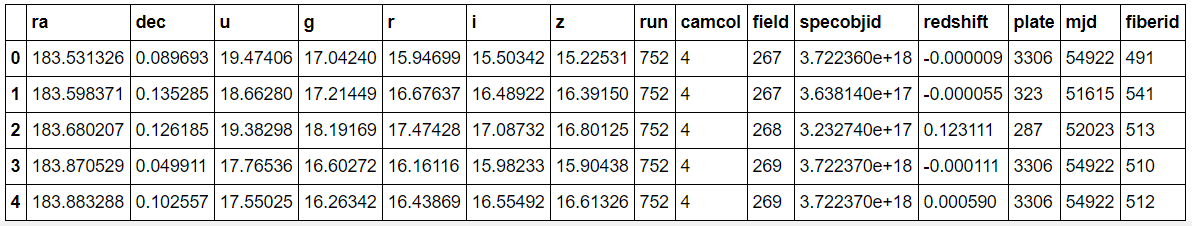
As per the graphical representation of the dataset the columns “run, field, spaceobjid, plate columns are right skewed. After treating skewness the data is further moved to model training.

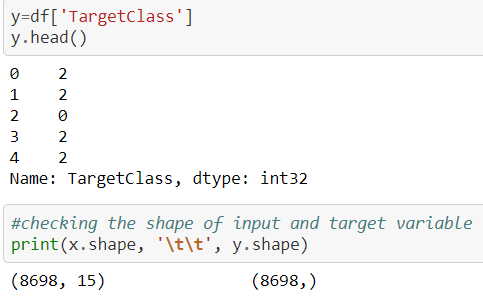
**Model Training**

Now separating the output variable “y” from the input variable df\_x

* #seprating into input and output variables
* df\_x=df.drop(columns=['TargetClass'])
* y=pd.DataFrame(df['Confirmed'])

df\_x.head()

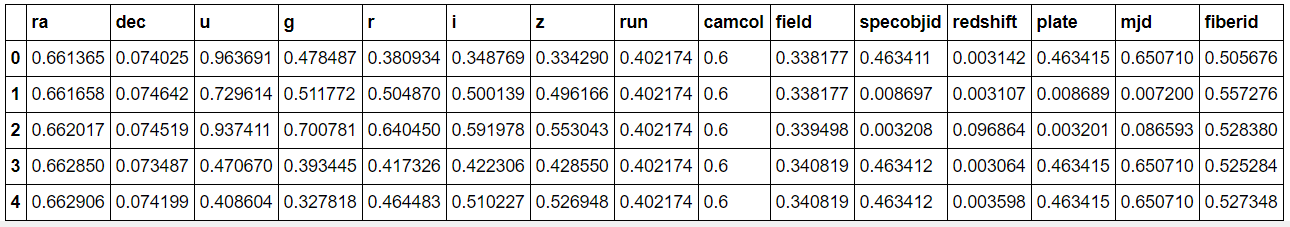




Using Normalization method for further processing of model training.

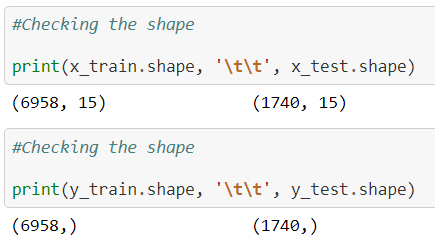
**Normalization** - is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.

* # Using normalization method
* x = (df\_x - np.min(df\_x))/(np.max(df\_x) - np.min(df\_x))
* x.head()



Splitting the data into training and testing data

* #breaking input and output into target variable
* **from** sklearn.model\_selection **import** train\_test\_split
* x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.20, random\_state=42)



**Using Classifier models for model building.**

Different type of Classifier models used in model building.

**1. Logistic regression** - is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression).

2. **Gaussian Naive Bayes** - algorithm is a special type of NB algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a gaussian distribution i.e, normal distribution. A Gaussian classifier is a generative approach in the sense that it attempts to model class posterior as well as input class-conditional distribution. Therefore, we can generate new samples in input space with a Gaussian classifier.

**3.** **SVC** - “Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification and regression challenges. It is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving a SVM model sets of labeled training data for each category, they're able to categorize new text.

**4.** **Decision Tree Classifier** - Decision tree learning is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree to go from observations about an item to conclusions about the item's target value.

**5. KNeighbors** **Classifier** - It is a method based on k-nearest neighbors. In the KNeighbors model target is predicted by local interpolation of the targets which associated to the nearest neighbors in the training set. KNN works by finding the distances between a query and all the examples in the data, selecting the specified number examples (K) closest to the query, then votes for the most frequent label (in the case of classification).

**6. Random Forest Classifier -** Random Forest uses multiple decision trees as base learning models in the dataset. Random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting in the dataset. The main concept of Random Forest is to combine multiple decision trees in determining the final result rather than relying on individual decision trees.

7. **AdaBoost Classifier** - AdaBoost is best used to boost the performance of decision trees on binary classification problems. AdaBoost was originally called AdaBoost. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers.

**8. Gradient boosting classifiers** - are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting.

**9. A Bagging classifier -** is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Bagging is used when the goal is to reduce the variance of a decision tree classifier.

**10. Extra Trees Classifier -** is an ensemble learning method fundamentally based on decision trees. Extra Trees Classifier, like Random Forest, randomizes certain decisions and subsets of data to minimize over-learning from the data and over fitting. This class implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Importing all the necessary libraries for model building.

* #Importing all the model library
* **from** sklearn.linear\_model **import** LogisticRegression
* **from** sklearn.naive\_bayes **import** GaussianNB
* **from** sklearn.svm **import** SVC
* **from** sklearn.tree **import** DecisionTreeClassifier
* **from** sklearn.neighbors **import** KNeighborsClassifier
* #Importing boosting models
* **from** sklearn.ensemble **import** RandomForestClassifier
* **from** sklearn.ensemble **import** AdaBoostClassifier
* **from** sklearn.ensemble **import** GradientBoostingClassifier
* **from** sklearn.ensemble **import** BaggingClassifier
* **from** sklearn.ensemble **import** ExtraTreesClassifier
* #Importing error metrics
* **from** sklearn.metrics **import** accuracy\_score, confusion\_matrix, classification\_report
* **from** sklearn.model\_selection **import** GridSearchCV, cross\_val\_score

**Using algorithms via for loop for models “Logistic Regression, GaussianNB, SVC, Decision Tree Classifier, KNeighbors Classifier, Random Forest Classifier, AdaBoost Classifier, Gradient Boosting Classifier, Bagging Classifier, Extra Trees Classifier.”**

* #All Algorithm by using for loop
* model=[LogisticRegression(), GaussianNB(), SVC(), DecisionTreeClassifier(), KNeighborsClassifier(), RandomForestClassifier(),
* AdaBoostClassifier(), GradientBoostingClassifier(), BaggingClassifier(), ExtraTreesClassifier()]
* **for** m **in** model:
* m.fit(x\_train, y\_train)
* m.score(x\_train, y\_train)
* predm=m.predict(x\_test)
* **print**('Accuracy score of' , m, 'is:')
* **print**(accuracy\_score(y\_test, predm))
* **print**(confusion\_matrix(y\_test, predm))
* **print**(classification\_report(y\_test, predm))
* **print**('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
* **print**('\n')

**Output**

* Accuracy score of LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,
* intercept\_scaling=1, l1\_ratio=None, max\_iter=100,
* multi\_class='auto', n\_jobs=None, penalty='l2',
* random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,
* warm\_start=False) **is**:
* 0.9396551724137931
* [[917   4  34]
* [ 19  63   0]
* [ 48   0 655]]
* precision    recall  f1-score   support
* 0       0.93      0.96      0.95       955
* 1       0.94      0.77      0.85        82
* 2       0.95      0.93      0.94       703
* accuracy                           0.94      1740
* macro avg       0.94      0.89      0.91      1740
* weighted avg       0.94      0.94      0.94      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
* Accuracy score of GaussianNB(priors=None, var\_smoothing=1e-09) **is**:
* 0.9264367816091954
* [[924  27   4]
* [  8  74   0]
* [ 74  15 614]]
* precision    recall  f1-score   support
* 0       0.92      0.97      0.94       955
* 1       0.64      0.90      0.75        82
* 2       0.99      0.87      0.93       703
* accuracy                           0.93      1740
* macro avg       0.85      0.91      0.87      1740
* weighted avg       0.94      0.93      0.93      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Accuracy score of SVC(C=1.0, break\_ties=False, cache\_size=200, class\_weight=None, coef0=0.0,
* decision\_function\_shape='ovr', degree=3, gamma='scale', kernel='rbf',
* max\_iter=-1, probability=False, random\_state=None, shrinking=True,
* tol=0.001, verbose=False) **is**:
* 0.9597701149425287
* [[932   4  19]
* [ 12  70   0]
* [ 35   0 668]]
* precision    recall  f1-score   support
* 0       0.95      0.98      0.96       955
* 1       0.95      0.85      0.90        82
* 2       0.97      0.95      0.96       703
* accuracy                           0.96      1740
* macro avg       0.96      0.93      0.94      1740
* weighted avg       0.96      0.96      0.96      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Accuracy score of DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',
* max\_depth=None, max\_features=None, max\_leaf\_nodes=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, presort='deprecated',
* random\_state=None, splitter='best') **is**:
* 0.9873563218390805
* [[941  12   2]
* [  7  75   0]
* [  1   0 702]]
* precision    recall  f1-score   support
* 0       0.99      0.99      0.99       955
* 1       0.86      0.91      0.89        82
* 2       1.00      1.00      1.00       703
* accuracy                           0.99      1740
* macro avg       0.95      0.97      0.96      1740
* weighted avg       0.99      0.99      0.99      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Accuracy score of KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',
* metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,
* weights='uniform') **is**:
* 0.9017241379310345
* [[911   3  41]
* [ 17  64   1]
* [109   0 594]]
* precision    recall  f1-score   support
* 0       0.88      0.95      0.91       955
* 1       0.96      0.78      0.86        82
* 2       0.93      0.84      0.89       703
* accuracy                           0.90      1740
* macro avg       0.92      0.86      0.89      1740
* weighted avg       0.90      0.90      0.90      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Accuracy score of RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,
* criterion='gini', max\_depth=None, max\_features='auto',
* max\_leaf\_nodes=None, max\_samples=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, n\_estimators=100,
* n\_jobs=None, oob\_score=False, random\_state=None,
* verbose=0, warm\_start=False) **is**:
* 0.9902298850574712
* [[949   2   4]
* [  9  73   0]
* [  2   0 701]]
* precision    recall  f1-score   support
* 0       0.99      0.99      0.99       955
* 1       0.97      0.89      0.93        82
* 2       0.99      1.00      1.00       703
* accuracy                           0.99      1740
* macro avg       0.99      0.96      0.97      1740
* weighted avg       0.99      0.99      0.99      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*


* Accuracy score of AdaBoostClassifier(algorithm='SAMME.R', base\_estimator=None, learning\_rate=1.0,
* n\_estimators=50, random\_state=None) **is**:
* 0.7729885057471264
* [[636 316   3]
* [  5  77   0]
* [ 71   0 632]]
* precision    recall  f1-score   support
* 0       0.89      0.67      0.76       955
* 1       0.20      0.94      0.32        82
* 2       1.00      0.90      0.94       703
* accuracy                           0.77      1740
* macro avg       0.69      0.83      0.68      1740
* weighted avg       0.90      0.77      0.82      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Accuracy score of GradientBoostingClassifier(ccp\_alpha=0.0, criterion='friedman\_mse', init=None,
* learning\_rate=0.1, loss='deviance', max\_depth=3,
* max\_features=None, max\_leaf\_nodes=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, n\_estimators=100,
* n\_iter\_no\_change=None, presort='deprecated',
* random\_state=None, subsample=1.0, tol=0.0001,
* validation\_fraction=0.1, verbose=0,
* warm\_start=False) **is**:
* 0.9862068965517241
* [[945   8   2]
* [ 11  71   0]
* [  3   0 700]]
* precision    recall  f1-score   support
* 0       0.99      0.99      0.99       955
* 1       0.90      0.87      0.88        82
* 2       1.00      1.00      1.00       703
* accuracy                           0.99      1740
* macro avg       0.96      0.95      0.96      1740
* weighted avg       0.99      0.99      0.99      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Accuracy score of BaggingClassifier(base\_estimator=None, bootstrap=True, bootstrap\_features=False,
* max\_features=1.0, max\_samples=1.0, n\_estimators=10,
* n\_jobs=None, oob\_score=False, random\_state=None, verbose=0,
* warm\_start=False) **is**:
* 0.9873563218390805
* [[944   7   4]
* [ 11  71   0]
* [  0   0 703]]
* precision    recall  f1-score   support
* 0       0.99      0.99      0.99       955
* 1       0.91      0.87      0.89        82
* 2       0.99      1.00      1.00       703
* accuracy                           0.99      1740
* macro avg       0.96      0.95      0.96      1740
* weighted avg       0.99      0.99      0.99      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Accuracy score of ExtraTreesClassifier(bootstrap=False, ccp\_alpha=0.0, class\_weight=None,
* criterion='gini', max\_depth=None, max\_features='auto',
* max\_leaf\_nodes=None, max\_samples=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, n\_estimators=100,
* n\_jobs=None, oob\_score=False, random\_state=None, verbose=0,
* warm\_start=False) **is**:
* 0.9844827586206897
* [[943   4   8]
* [ 10  72   0]
* [  5   0 698]]
* precision    recall  f1-score   support
* 0       0.98      0.99      0.99       955
* 1       0.95      0.88      0.91        82
* 2       0.99      0.99      0.99       703
* accuracy                           0.98      1740
* macro avg       0.97      0.95      0.96      1740
* weighted avg       0.98      0.98      0.98      1740
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Cross Validation**

Cross validation helps to find out the over fitting and under fitting of the model.In the cross validation the model is made to run on different subsets of the dataset which will get multiple measures of the model. If we take 5 folds, the data will be divided into 5 pieces where each part being 20% of full dataset. While running the Cross validation the 1st part (20%) of the 5 parts will be kept out as a hold out set for validation and everything else is used for training data. This way we will get the first estimate of the model quality of the dataset. In the similar way further iterations are made for the second 20% of the dataset is held as a hold out set and remaining 4 parts are used for training data during process. This way we will get the second estimate of the model quality of the dataset. These steps are repeated during the cross validation process to get the remaining estimate of the model quality.

* models=[LogisticRegression(), GaussianNB(), SVC(), DecisionTreeClassifier(), KNeighborsClassifier(), RandomForestClassifier(),
* AdaBoostClassifier(), GradientBoostingClassifier(), BaggingClassifier(), ExtraTreesClassifier()]
* **for** m **in** models:
* score=cross\_val\_score(m,x,y, cv=15, scoring='accuracy')
* **print**("Model:", m)
* **print**("Score:", score)
* **print**("Mean Score:", score.mean())
* **print**("Standard deviation:", score.std())
* **print**('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
* **print**('\n')

**Output of Cross Validation**

* Model: LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,
* intercept\_scaling=1, l1\_ratio=None, max\_iter=100,
* multi\_class='auto', n\_jobs=None, penalty='l2',
* random\_state=None, solver='lbfgs', tol=0.0001, verbose=0,
* warm\_start=False)
* Score: [0.96206897 0.93793103 0.93793103 0.95689655 0.93793103 0.92758621
* 0.92241379 0.93275862 0.93275862 0.92931034 0.90344828 0.95
* 0.95344828 0.95854922 0.94300518]
* Mean Score: 0.9390691441843845
* Standard deviation: 0.015110454254975572
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Model: GaussianNB(priors=None, var\_smoothing=1e-09)
* Score: [0.9637931  0.94137931 0.93448276 0.95517241 0.93793103 0.9
* 0.87413793 0.93448276 0.93448276 0.88448276 0.86896552 0.93793103
* 0.95862069 0.92055268 0.92227979]
* Mean Score: 0.9245796359160662
* Standard deviation: 0.028793623407978943
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Model: SVC(C=1.0, break\_ties=False, cache\_size=200, class\_weight=None, coef0=0.0,
* decision\_function\_shape='ovr', degree=3, gamma='scale', kernel='rbf',
* max\_iter=-1, probability=False, random\_state=None, shrinking=True,
* tol=0.001, verbose=False)
* Score: [0.98103448 0.95862069 0.96034483 0.97758621 0.97068966 0.94482759
* 0.95172414 0.95862069 0.94827586 0.94137931 0.9137931  0.9637931
* 0.96896552 0.95509499 0.94991364]
* Mean Score: 0.9563109205328251
* Standard deviation: 0.015925438325002917
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Model: DecisionTreeClassifier(ccp\_alpha=0.0, class\_weight=None, criterion='gini',
* max\_depth=None, max\_features=None, max\_leaf\_nodes=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, presort='deprecated',
* random\_state=None, splitter='best')
* Score: [0.98275862 0.98448276 0.97758621 0.98448276 0.98275862 0.97758621
* 0.97758621 0.99310345 0.9862069  0.97931034 0.98275862 0.98103448
* 0.98965517 0.9775475  0.98272884]
* Mean Score: 0.9826391122228179
* Standard deviation: 0.004430809714790078
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Model: KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',
* metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,
* weights='uniform')
* Score: [0.92931034 0.88275862 0.89137931 0.91034483 0.91724138 0.87413793
* 0.82758621 0.90172414 0.87068966 0.85       0.82413793 0.92068966
* 0.92241379 0.89637306 0.90673575]
* Mean Score: 0.8883681734262403
* Standard deviation: 0.032273992698778925
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Model: RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,
* criterion='gini', max\_depth=None, max\_features='auto',
* max\_leaf\_nodes=None, max\_samples=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, n\_estimators=100,
* n\_jobs=None, oob\_score=False, random\_state=None,
* verbose=0, warm\_start=False)
* Score: [0.99310345 0.9862069  0.98793103 0.98965517 0.99310345 0.98965517
* 0.98793103 0.99310345 0.98965517 0.98275862 0.98103448 0.9862069
* 0.99482759 0.98618307 0.98791019]
* Mean Score: 0.9886177118694538
* Standard deviation: 0.0037681691318515597
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* Model: AdaBoostClassifier(algorithm='SAMME.R', base\_estimator=None, learning\_rate=1.0,
* n\_estimators=50, random\_state=None)
* Score: [0.85689655 0.48793103 0.91896552 0.76896552 0.86551724 0.92413793
* 0.91206897 0.85862069 0.93275862 0.97413793 0.87586207 0.96724138
* 0.93275862 0.89464594 0.92400691]
* Mean Score: 0.8729676612470968
* Standard deviation: 0.11407850844249341
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

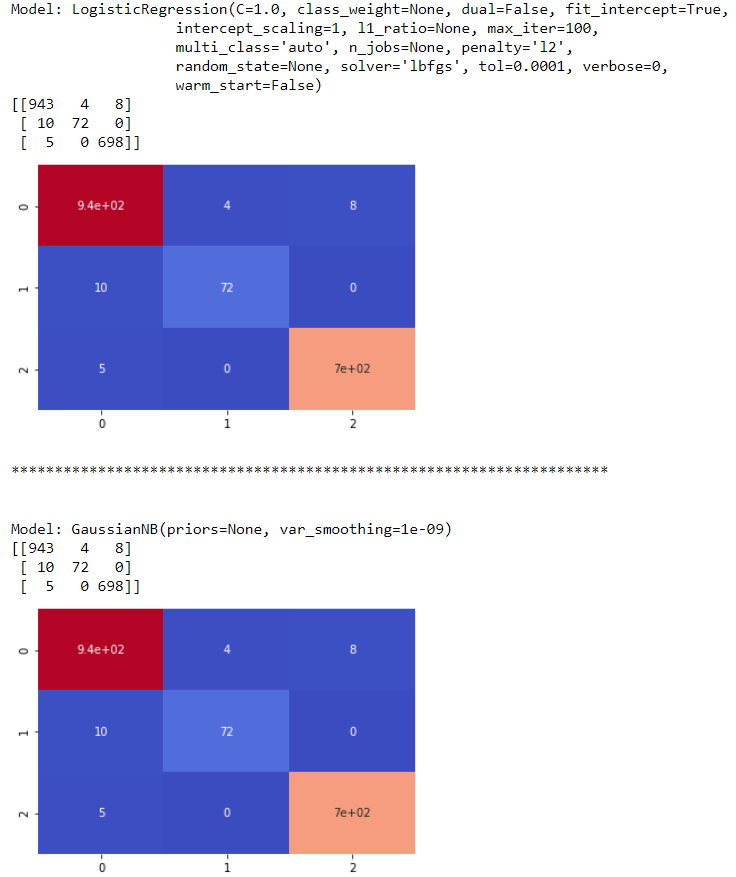
* Model: GradientBoostingClassifier(ccp\_alpha=0.0, criterion='friedman\_mse', init=None,
* learning\_rate=0.1, loss='deviance', max\_depth=3,
* max\_features=None, max\_leaf\_nodes=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, n\_estimators=100,
* n\_iter\_no\_change=None, presort='deprecated',
* random\_state=None, subsample=1.0, tol=0.0001,
* validation\_fraction=0.1, verbose=0,
* warm\_start=False)
* Score: [0.99482759 0.98793103 0.98965517 0.9862069  0.99655172 0.98793103
* 0.98965517 0.99310345 0.98793103 0.98793103 0.98275862 0.98793103
* 0.99310345 0.98272884 0.9775475 ]
* Mean Score: 0.9883862386595995
* Standard deviation: 0.00478073060888594
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

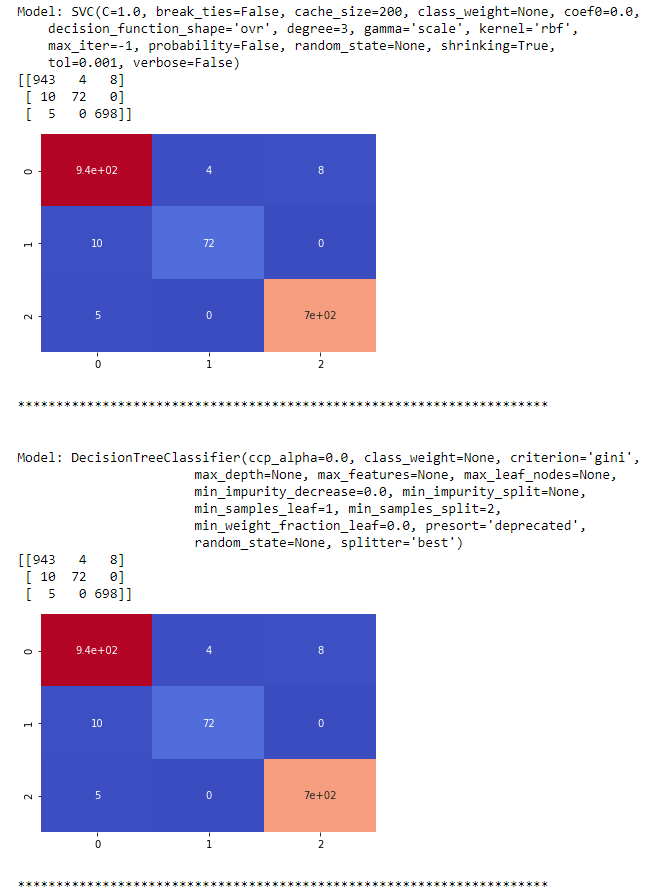
* Model: BaggingClassifier(base\_estimator=None, bootstrap=True, bootstrap\_features=False,
* max\_features=1.0, max\_samples=1.0, n\_estimators=10,
* n\_jobs=None, oob\_score=False, random\_state=None, verbose=0,
* warm\_start=False)
* Score: [0.98965517 0.98793103 0.98103448 0.99310345 0.99482759 0.98965517
* 0.98965517 0.98965517 0.98965517 0.98448276 0.98103448 0.9862069
* 0.99137931 0.98100173 0.98963731]
* Mean Score: 0.9879276596589441
* Standard deviation: 0.004179192547066833
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

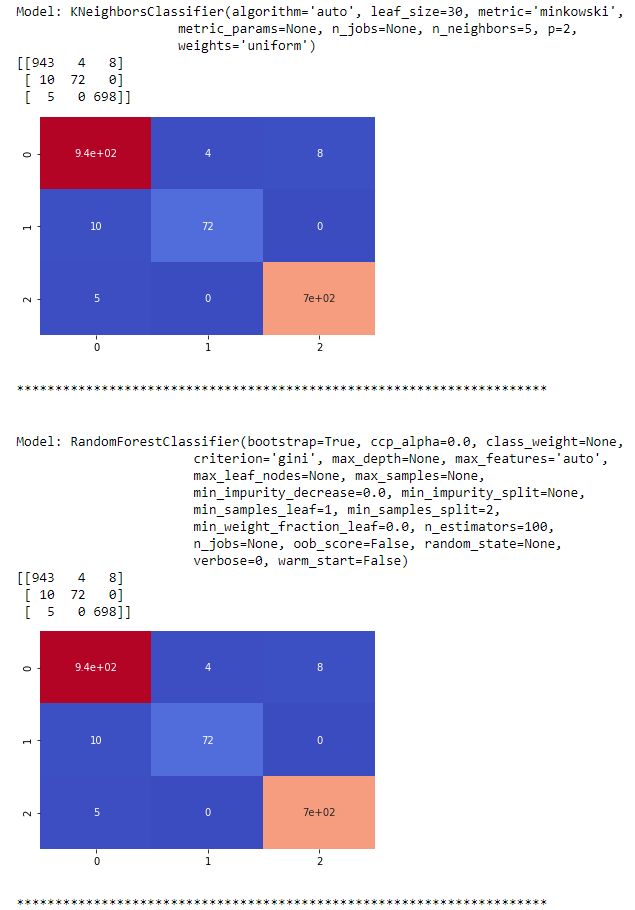
* Model: ExtraTreesClassifier(bootstrap=False, ccp\_alpha=0.0, class\_weight=None,
* criterion='gini', max\_depth=None, max\_features='auto',
* max\_leaf\_nodes=None, max\_samples=None,
* min\_impurity\_decrease=0.0, min\_impurity\_split=None,
* min\_samples\_leaf=1, min\_samples\_split=2,
* min\_weight\_fraction\_leaf=0.0, n\_estimators=100,
* n\_jobs=None, oob\_score=False, random\_state=None, verbose=0,
* warm\_start=False)
* Score: [0.98965517 0.96896552 0.97586207 0.9862069  0.9862069  0.97413793
* 0.97931034 0.97586207 0.97931034 0.96551724 0.95172414 0.97931034
* 0.9862069  0.9775475  0.98272884]
* Mean Score: 0.9772368133722432
* Standard deviation: 0.009340958405021905
* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

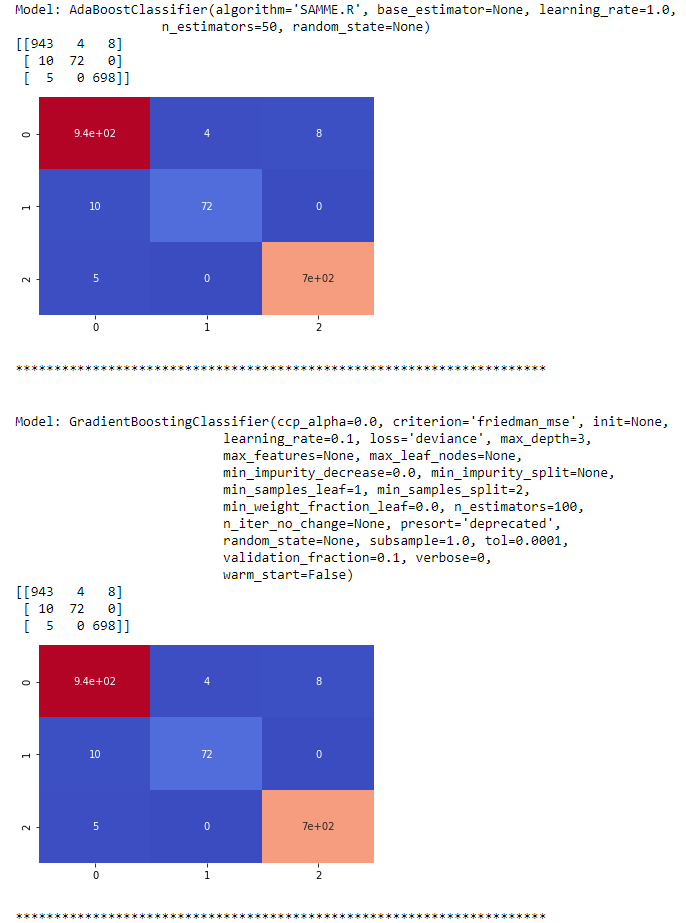
**Confusion Matrix for models**

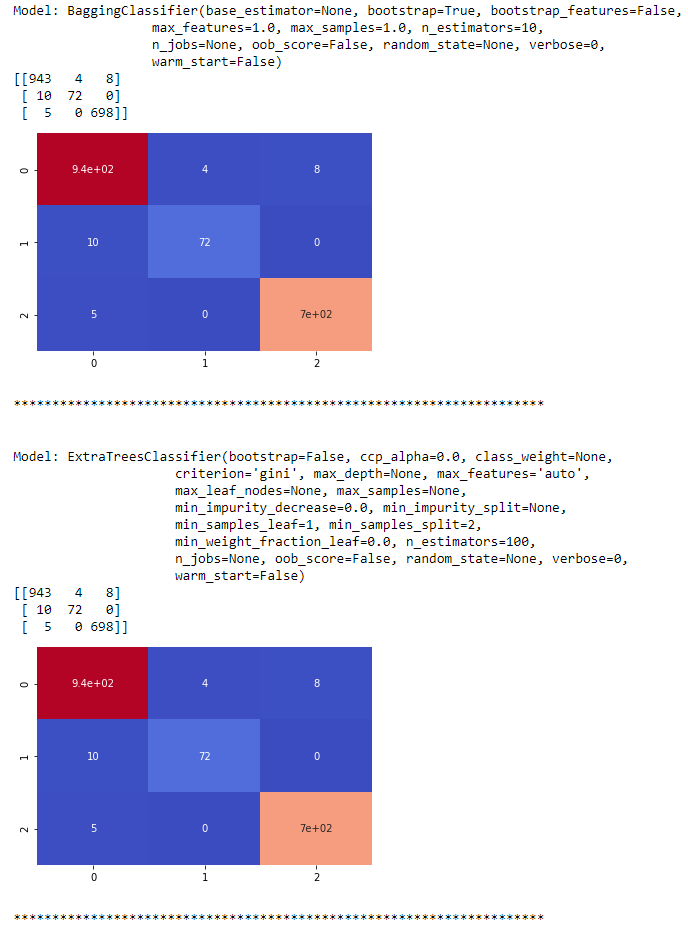
* #Plotting confusion matrix for models
* **for** m **in** model:
* **print**("Model:", m)
* cm=confusion\_matrix(y\_test, predm)
* sns.heatmap(cm, annot=True, cbar=False, cmap='coolwarm')
* **print**(confusion\_matrix(y\_test, predm))
* plt.show()
* **print**('\n')
* **print**('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')
* **print**('\n')

****

****

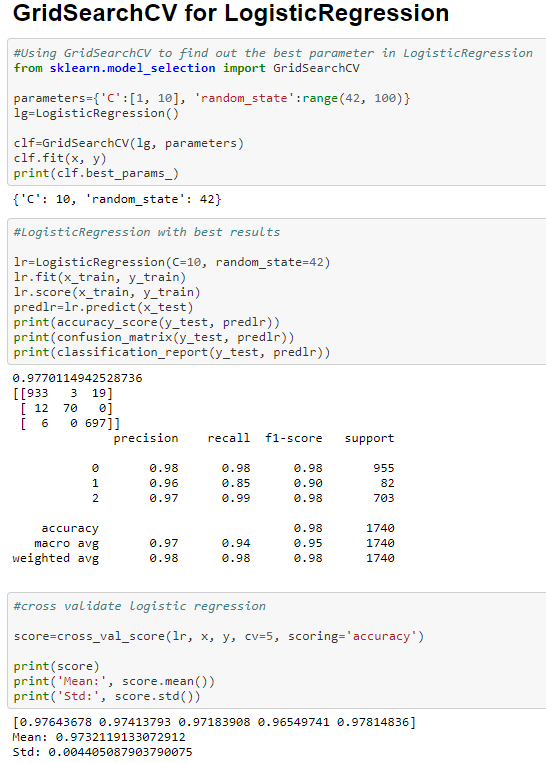
****

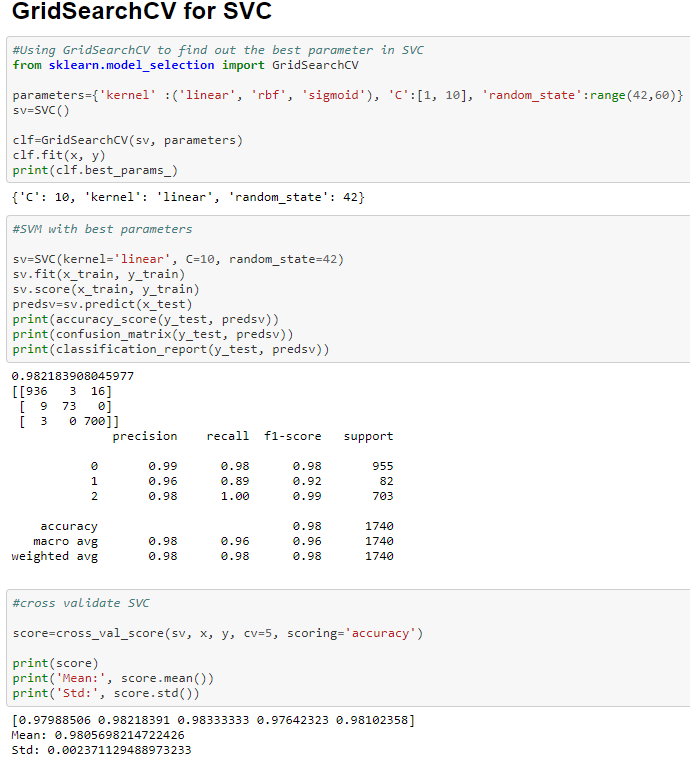
****

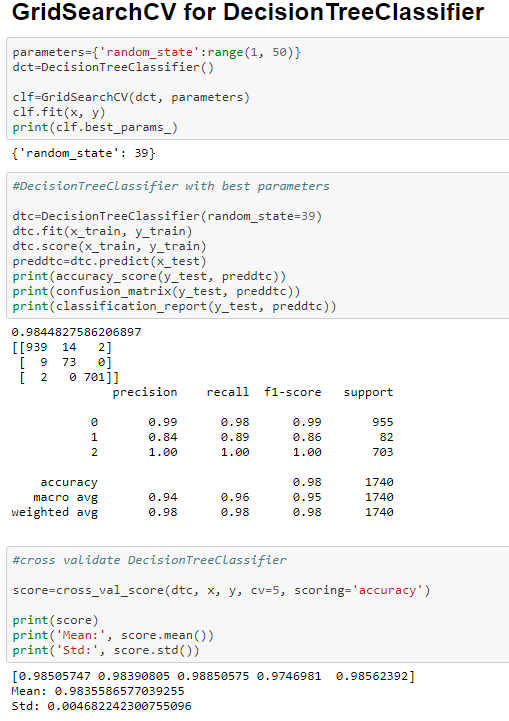
****

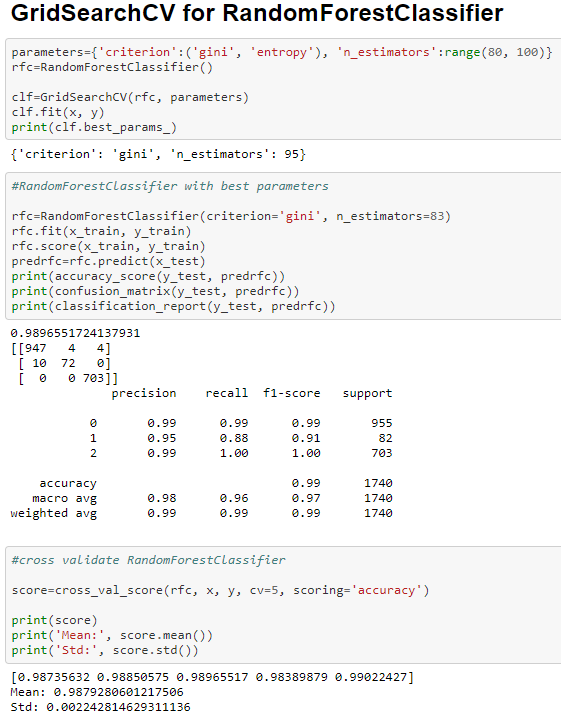
**Using GridsearchCV to find out the best parameter**

Grid search is an approach to hyper parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

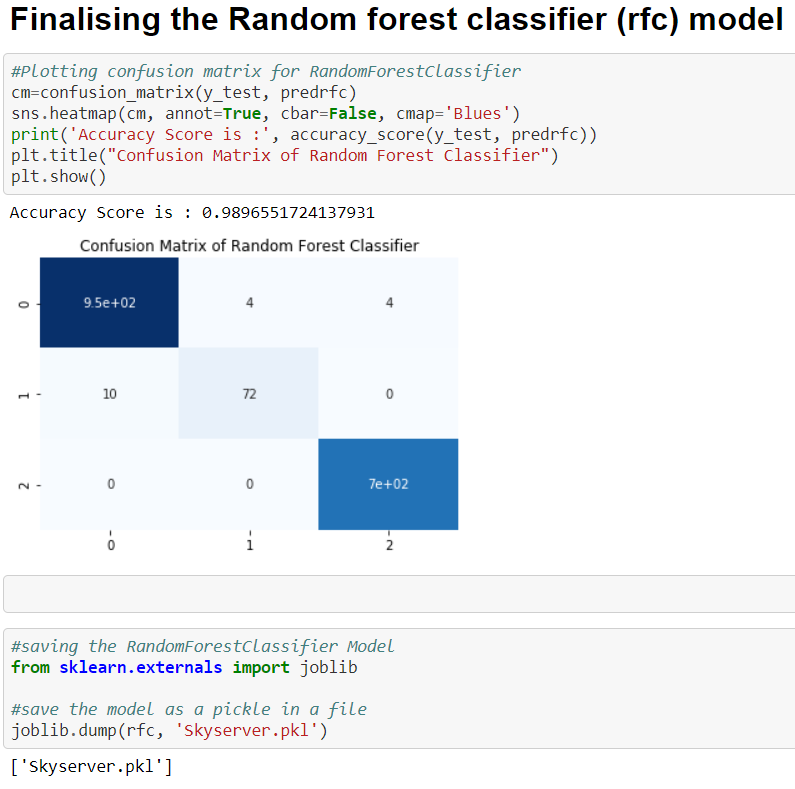
****

****

****

****

Random forest classifier is having the best scores for the model. No let’s finalize the Random forest model as final model.



**Observations**

1.Random Forest uses multiple decision trees as base learning models in the dataset.

**2.** Random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting in the dataset.

3. The main concept of Random Forest is to combine multiple decision trees in determining the final result rather than relying on individual decision trees.

4. Using Random Forest Classifier Accuracy Score is : 0.9896551 is achieved during the process.

5. The cross validation score of the Random Forest Classifier model is (CV =5) –

[0.98735632 0.98850575 0.98965517 0.98389879 0.99022427]

6. The score obtained during the cross validation proves that model is working accurately for the given dataset & shows that Random Forest Classifier fits perfectly.

NOTE - **NOTE** – The “Skyserver\_dataset” classification problem detailed code can be found in Github repository.

**Github Repository Link** –

https://github.com/Carneiro22/Evaluation-Projects/blob/main/Evaluation%20Project%208%20-%20Skyserver.ipynb