**Problem Definition :**

**Content**

The data has been taken by SDSS. The data of the space consists of 10,000 observations. Each observation has 17 feature columns. There is one column named “class” from where we need to predict the output. The output can be either a star, galaxy or quasar.

Link of the dataset :

<https://raw.githubusercontent.com/dsrscientist/dataset1/master/Skyserver.csv>

**Feature Description**

The table results from a query which joins two tables (actuaclly views): "PhotoObj" which contains photometric data and "SpecObj" which contains spectral data.

To ease our start with the data we can read the feature descriptions below:

***View "PhotoObj"***

* objid = Object Identifier
* ra = J2000 Right Ascension (r-band)
* dec = J2000 Declination (r-band)

Right ascension (shortened RA) is the cornered distance estimated toward the eastward along the divine equator from the Sun at the March equinox to the hour circle of the point over the earth being referred to. At the point when combined with declination (truncated dec), these galactic directions determine the course of a point on the heavenly circle (generally brought in English the skies or the sky) in the central organize framework.

* u = better of DeV/Exp magnitude fit
* g = better of DeV/Exp magnitude fit
* r = better of DeV/Exp magnitude fit
* i = better of DeV/Exp magnitude fit
* z = better of DeV/Exp magnitude fit

The Thuan-Gunn astronomic magnitude system. u, g, r, i, z represent the response of the 5 bands of the telescope.

u, g, r, I, z address the reaction of the 5 bands of the telescope.

* run = Run Number
* rereun = Rerun Number
* camcol = Camera column
* field = Field number

Run, rerun, camcol and field are highlights which illustrate a field inside a picture taken by the SDSS. A field is fundamentally a piece of the whole picture conform to 2048 by 1489 pixels. A field can be recognized by:

* run number, which identifies the specific scan,
* the camera column, or "camcol," a number from 1 to 6, identifying the scanline within the run, and
* the field number. The field number typically starts at 11 (after an initial rampup time), and can be as large as 800 for particularly long runs.
* An additional number, rerun, specifies how the image was processed.

***View "SpecObj"***

* specobjid = Object Identifier
* class = object class (galaxy, star or quasar object)

The class recognizes an item to be either a system, star or quasar. This will be the outcome variable which we will be attempting to predict.

* redshift = Final Redshift
* plate = plate number
* mjd = MJD of observation
* fiberid = fiber ID

In physical science, redshift happens when light or other electromagnetic radiation from an object is expanded in wavelength, or moved to the red ending point of the spectrum.

Each spectroscopic exposure utilizes a huge, slender, round metal plate that positions optical filaments through openings bored at the areas of the pictures in the telescope central plane. Each plate has it’s own serial number, which is called plate in perspectives like SpecObj in the CAS.

1. Exploratory Analysis :

# Importing All necessary Libraries

**import** **pandas** **as** **pd**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

%matplotlib inline

**Loading the data :**

df=pd.read\_csv('skyserver.csv')

df1=pd.read\_csv('skyserver.csv')

Redshift = df1["redshift"]

“read\_csv” is an important function of pandas which allows to read csv files and we can make various operations on the dataset. As my file is a CSV file that’s why I have used “read\_csv” function to load the data from the specific directory.The name of the dataset id df. I have created another dataset called df1 because after removing the skew from the dataset we will get Null values on the "redshift" column. We can explore it at the later step of our solution.

## **Description of Data :**

**df.info()**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10000 entries, 0 to 9999

Data columns (total 18 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 objid 10000 non-null float64

1 ra 10000 non-null float64

2 dec 10000 non-null float64

3 u 10000 non-null float64

4 g 10000 non-null float64

5 r 10000 non-null float64

6 i 10000 non-null float64

7 z 10000 non-null float64

8 run 10000 non-null int64

9 rerun 10000 non-null int64

10 camcol 10000 non-null int64

11 field 10000 non-null int64

12 specobjid 10000 non-null float64

13 class 10000 non-null object

14 redshift 10000 non-null float64

15 plate 10000 non-null int64

16 mjd 10000 non-null int64

17 fiberid 10000 non-null int64

dtypes: float64(10), int64(7), object(1)

memory usage: 1.4+ MB

Normaly to explore the data we can use various functions such as shape, columns, dtypes, info(), head(), tail(), describe(). Here, I have used df.info()

By using info() we can get a concise summary of a DataFrame. It includes the index dtype and column dtypes, non-null values and memory usage.

In our dataset we can see that there are 17 Numeric columns and one object type column.

**Observations:**

Numeric features

Numeric features = ['objid', 'ra', 'dec', 'u', 'g', 'r', 'i', 'z', 'run', 'rerun', 'camcol', 'field', 'specobjid','redshift', 'plate', 'mjd', 'fiberid']

Catagorical features

Catagorical features =['class']

# **Missing Values**

There are different ways to check the missing values in our dataset.

**df.isnull().values.any()**

False

Here , we can see that there are no null values present in our dataset.

But, If we want to see column wise we can use another method.

**df.isnull().sum()**

objid 0

ra 0

dec 0

u 0

g 0

r 0

i 0

z 0

run 0

rerun 0

camcol 0

field 0

specobjid 0

class 0

redshift 0

plate 0

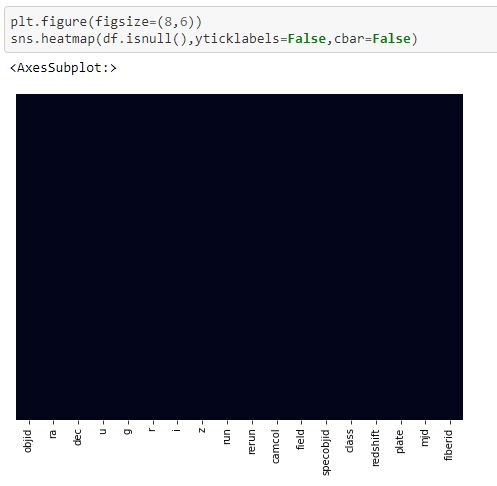
mjd 0

fiberid 0

dtype: int64

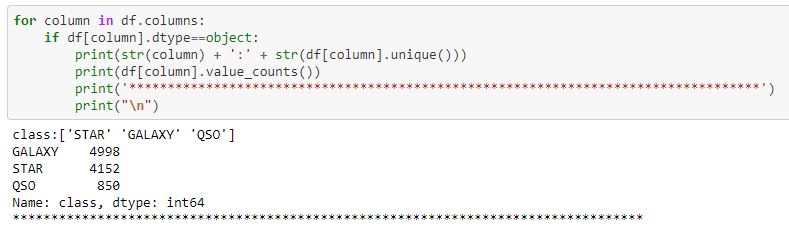
In every colulm there are no Null values in the datset.

If we want to visualizise the null values then there are another method.



By any method we will get the same results.

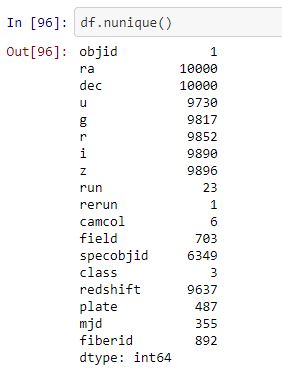
# Exploratory the catagorical columns :



To explore the catagorical columns and to count the number of values we can use the following code. We can get the following observations after using this code :

Class column is object type. Three values are present here. The values are GALAXY, STAR and QSO. GALAXY has been appeared for 4998 times; STAR 4152 times and QSO 850 times.

# **Checking Unique values :**



To explore the dataset it’s also necessary to explore the unique values. If we see our dataset we can see that in some columns there are more unique values and some columns contains less unique values. Columns with less unique values normally effect more to predict the outcome. But if there are constant value in this case there will have no use of that column. From the dataset we can see that ‘objid’ and ‘rerun’ has only one unique value. So, these two columns should be deleted. It will not effect our dataset. All the values of ‘ra’,’dec’ column are unique. These two columns can also be deleted. It will not effect our dataset. 3. u,g,r,i,z having a lot of unique values. But all the values are not unique.

### Change class into numeric type:

### 

### For analyzing the data with target i.e. class we have to change class into numeric type. Otherwise we will not be able to explore the correlationship of other columns with the target variable.

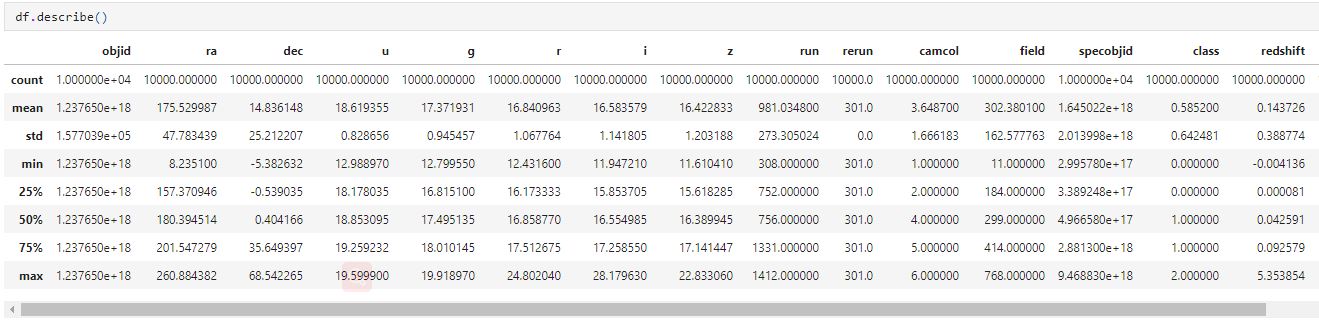
**Observations :**

0 stands for GALAXY

1 stands for STAR.

2 stands for QSO

# **Summary Statistics :**





By using describe() function we can explore the count, mean , median, standard deviation, minimum value, 25th, 50th and 75th percentile , maximum value.

We can find the following observations from the dataset :

1.Maximum 'objid', 'ra', 'dec', 'u', 'g', 'r', 'i', 'z', 'run', 'rerun', 'camcol','field', 'specobjid', 'class','redshift', 'plate', 'mjd', 'fiberid' are : 1.237650e+18,260.884382,68.542265,19.599900,19.918970,24.802040,28.179630,22.833060,1412.000000,301.0,6.000000,768.000000,9.468830e+18, 2.000000,5.353854,8410.000000,57481.000000,1000.000000

2. Minimum 'objid', 'ra', 'dec', 'u', 'g', 'r', 'i', 'z', 'run', 'rerun', 'camcol',

'field', 'specobjid', 'class', 'redshift', 'plate', 'mjd', 'fiberid' are 1.237650e+18, 8.235100, -5.382632, 12.988970, 12.799550, 12.431600, 11.947210, 11.610410, 308.000000, 301.0, 1.000000, 11.000000, 2.995780e+17, 0.000000, -0.004136, 266.000000, 51578.000000, 1.000000

3.dec,(i,z),run,field,redshift,plate,mjd,fiberid mean is greater than median therefore data is right skewed for these attributes. (i,z) columns have very less difference with median.

4. ra,(u,g,r),camcol,specobjid,class, median is greater than mean therefre data is left skewed for these attributes.(u,g,r) have very less difference with median.

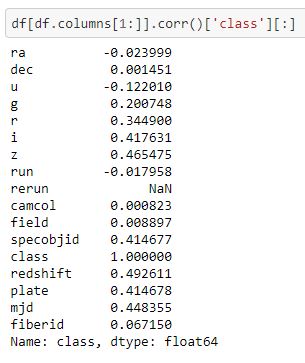
5. The difference between 75% and maximum is higher in run,field,specobjid,redshift,plate,mjd and fiberid columns therefore outliers are present in these columns.

# **Correlation:**

### Correlation only with target variable :

### 

We can see the correlation of every column with the target variable by using barplot. The peak points which are above 0 are positively correlated with the target variable. And the peak points which are under 0 are negatively correlated with the target variable. But the problem is we can no know the exaxt value of correlation by using barplot. To know the exact correlation we have another technique.



From the above figure we we be able to know the exact correlation of each column with the target variable.

The findings are mentioned below:

Findings:

1. Negative correlation : ra,u,run,objid

2. Positive correlation : dec,g, r, i, z,camcol, field, specobjid, redshift, plate, mjd, fiberid

3. No correlation : rerun . Because there is only one constant value present.

4. Strong Correlationship : g,r, i,u, z,specobjid, redshift, plate, mjd

5. Weak Correlation :dec, ra,run,camcol,objid field,fiberid. If these columns have outliers then these columns should be deleted.

**Correlation among all the variables:**



From the previous steps we have only known about the correlation of each column with the target variable. But by using heatmap we will be able to explore the correlation among all the numeric variables present in the dataset.

These are the observations from the above figure :

1. From the previous step we have already known about the correlation with target variable. Now, Let me check the other observations.

2. Good Correlationship with other variables except target variable :

1. (u,r,i,z) with g. The correlation is 85%,96%,91%,88%

2. (u,g,i,z) with r.

3. (u,g,r,z) with i.

4. (u,g,r,i) with z

5. (g,r,i,z) with u.

6. run and dec

7. field and ra

9. fiberid and camol

10.ra and field

11.specobjid and (plate,mjd)

12. plate and mjd

**Data Visualization :**

Our feature variable of interest is class

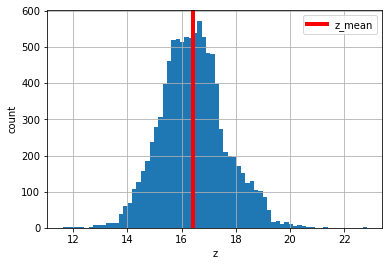
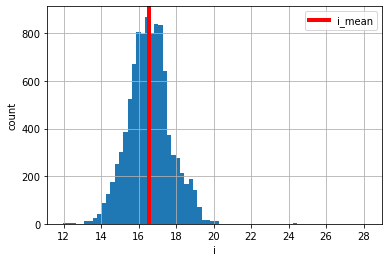
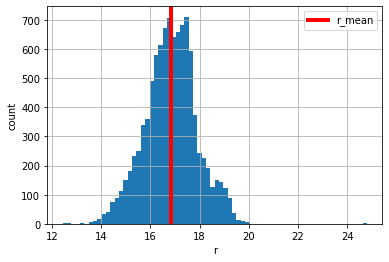
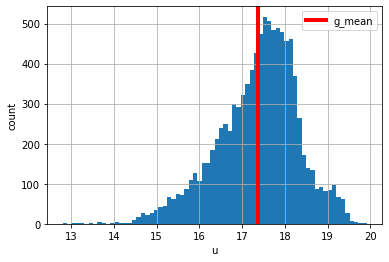
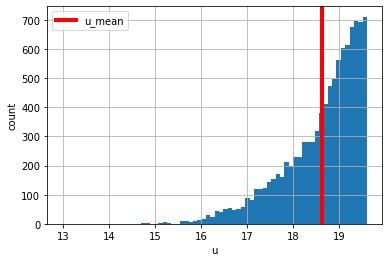
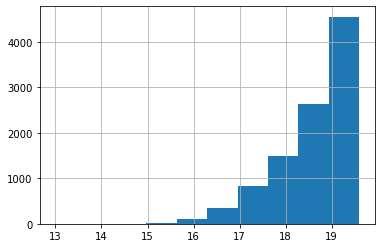
We already change it to integer valued as it contains three numbers 0,1 and 2

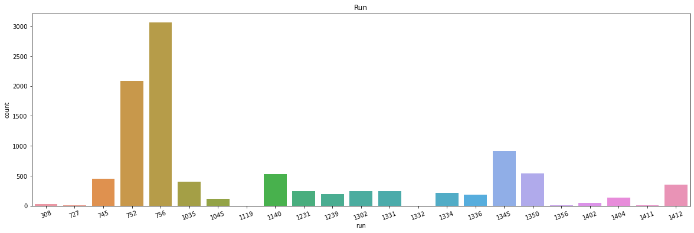
0 stands for GALAXY

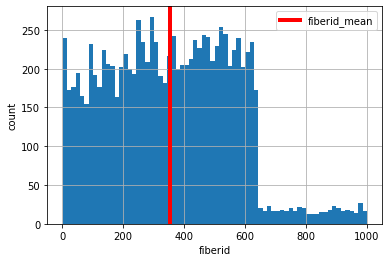
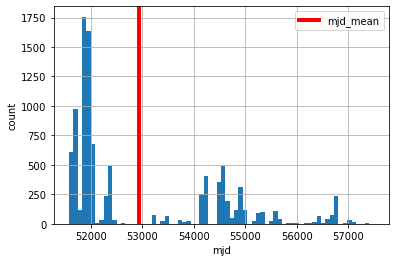
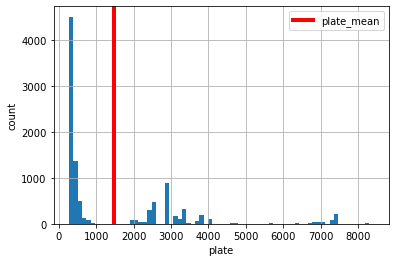
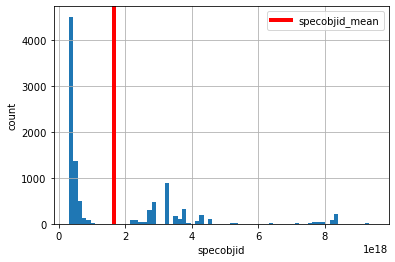
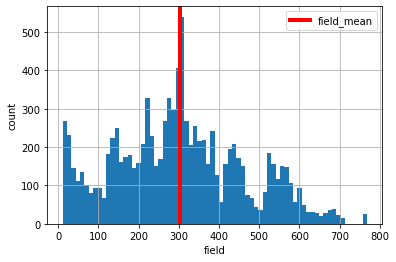
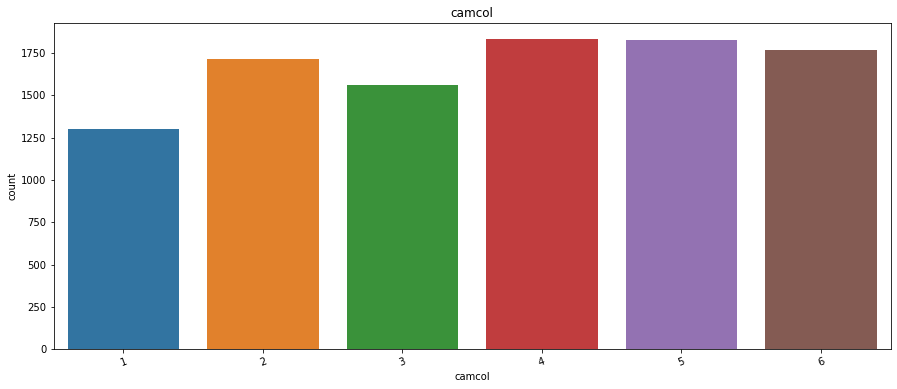
1 stands for STAR.

2 stands for QSO

**Univariate anslysis :**

****

****

****

In some columns the values are continuous type. For continuous type of data I have used histogram with the red coloured mean value to visualize every column perfectly. Some of the columns are catagorical and there are few specific values. In these cases I have used countplot.

Sample code for histogram :

i\_amount=df['Column Name'].hist(bins=70)

mean\_val=np.mean(df['Column Name'])

plt.axvline(mean\_val,linewidth=4,color='red',label='Column Name\_mean')

plt.xlabel('Column Name')

plt.ylabel('count')

plt.legend()

plt.show()

Sample code for countplot :

plt.subplots(figsize=(20,6))

sns.countplot(x='Column Name',data=df)

plt.title('Column Name')

plt.xticks(rotation=20)

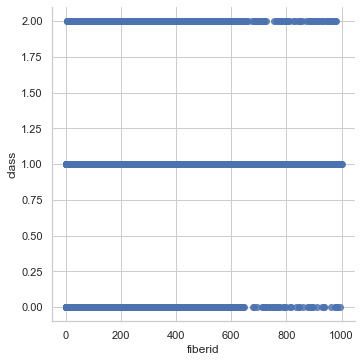
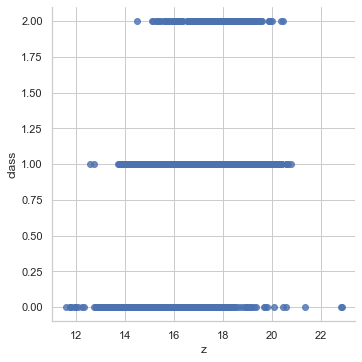
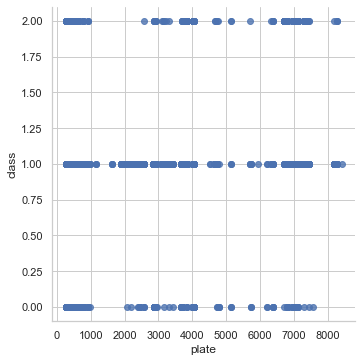
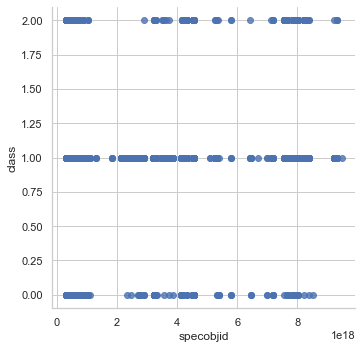
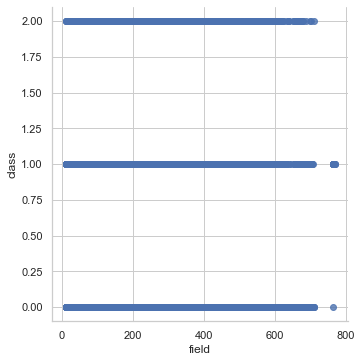
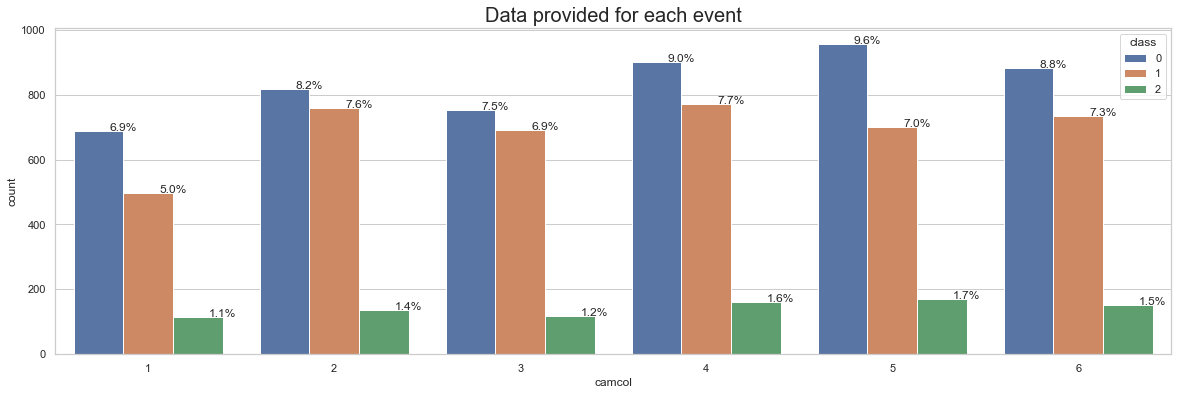
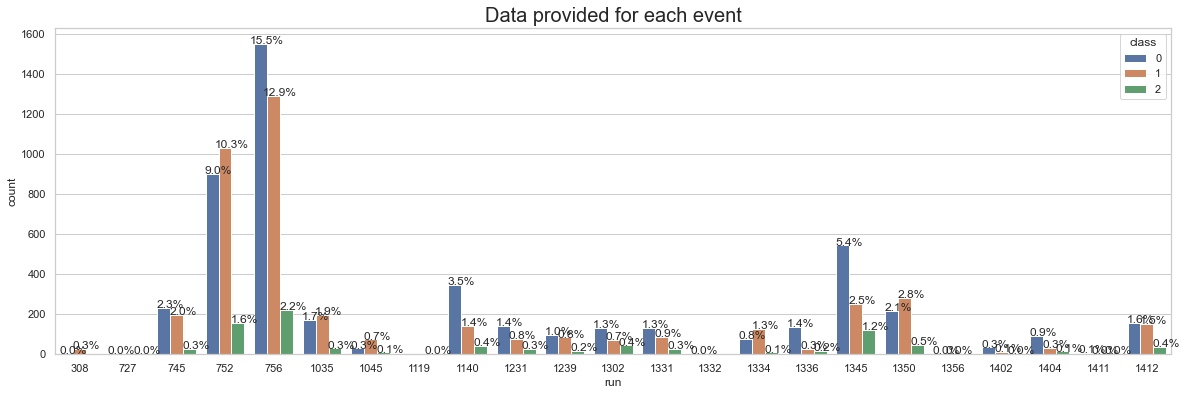
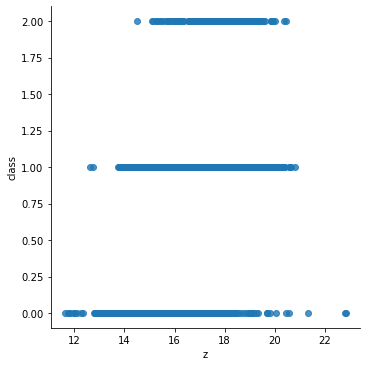
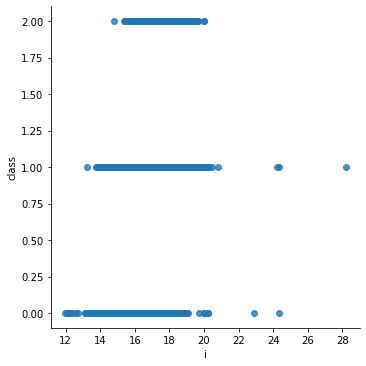
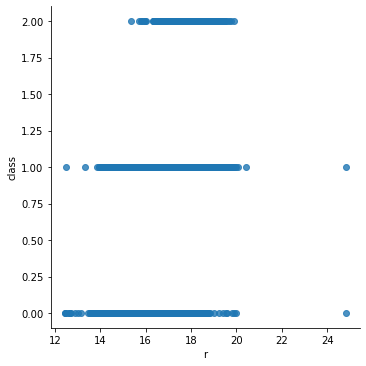
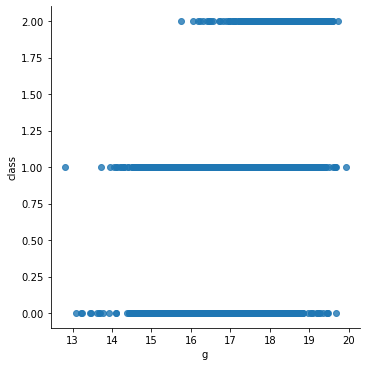
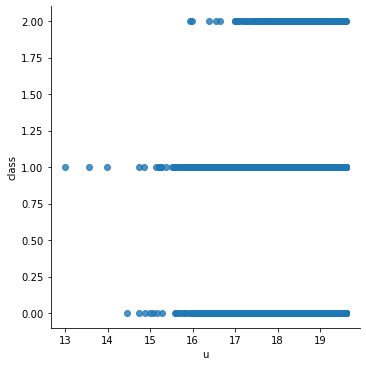
plt.show()

print(df.Column Name.value\_counts())

From the above figure we can explore the visualization of 'u', 'g', 'r', 'i', 'z', 'run', 'camcol', 'field', 'specobjid', 'plate', 'mjd', 'fiberid' columns.

In the g column most of the values are in between 17.2 and 18.2.Here , the mean value is 17.3.If we look at the ‘run’ column we can see that the value 756 has been appeared most of the times(3060). And then the value 752 which has been appeared 2086 times. By using countplot we can basically know about the frequency level.

**Bivariate Analysis :**

****

In bivariate analysis most of the times I have used scatterplot. Because most of the values are continuous types. We have used x as the column name and y for target variable named ‘class’. In some cases such as ‘run’ vs ‘class’ I have used countplot. Because both of the variables are catagorical. The sample codes are given below :

**For countplot :**

sns.set(style="whitegrid")

plt.figure(figsize=(20,6))

total = float(len(df))

ax = sns.countplot(x="column\_name", hue="Target\_variable", data=df)

plt.title('Data provided for each event', fontsize=20)

**for** p **in** ax.patches:

percentage = '**{:.1f}**%'.format(100 \* p.get\_height()/total)

x = p.get\_x() + p.get\_width()

y = p.get\_height()

ax.annotate(percentage, (x, y),ha='center')

plt.show()

**For Scatterplot :**

plt.figure(figsize=(8,4))

sns.lmplot(x='column\_name',y='Target\_variable',fit\_reg=**False**,data=df)

plt.show()

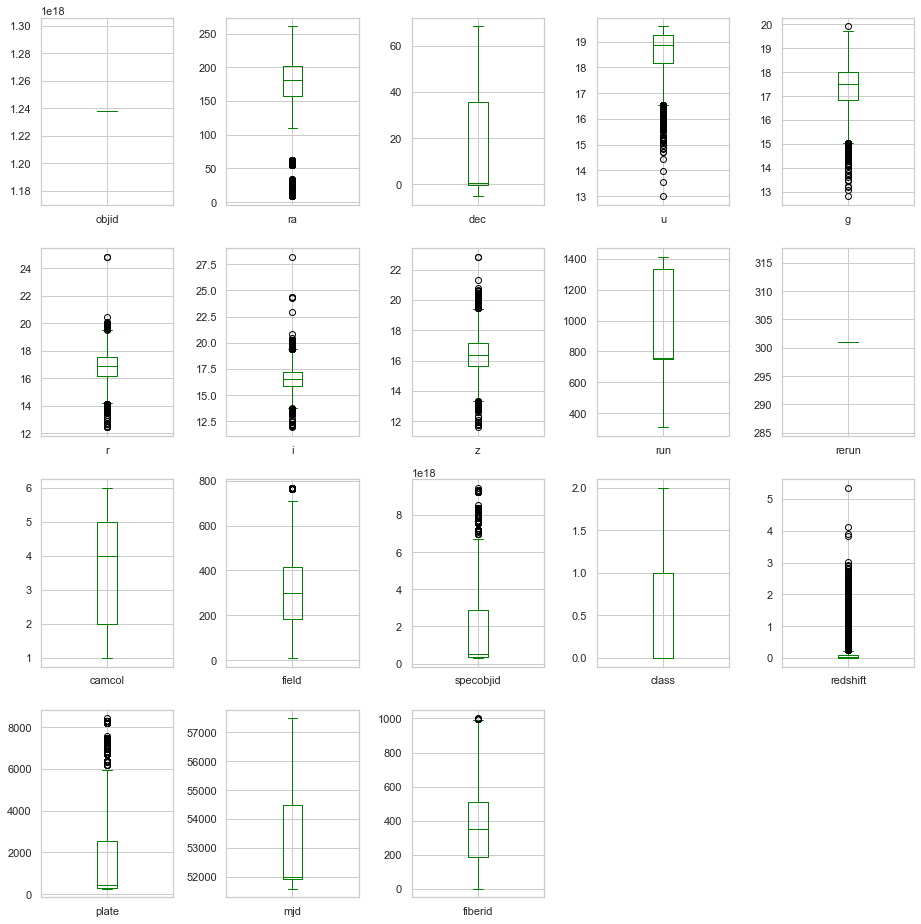
# In the relationship of “run vs class” we can see that the specific scan of 756 was 15.5% when the target value was Galaxy, 12.9% when the target value was Star and 2.2% when the target value was QSO.

# In the relationship of ‘z vs class’ most of the values are between 15 to 18. Here y axis represents the target variable.

# **Check Outliers :**

df.plot(kind='box',subplots=**True**,layout=(4,5),color='green',figsize=(13,13))

plt.tight\_layout()



After illustrating the above figure we can explore that ra,u,g,r,i,z,field,specobjid,redshift,plate,fiberid having outliers. The other columns don’t have outliers. So, we need to treatthe outliers.

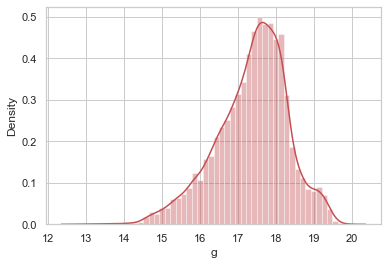
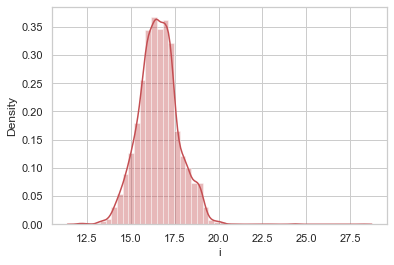
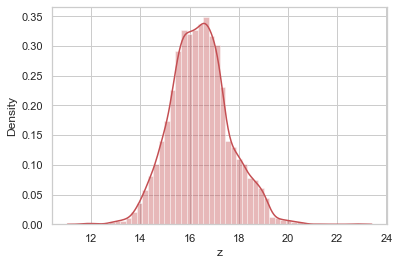
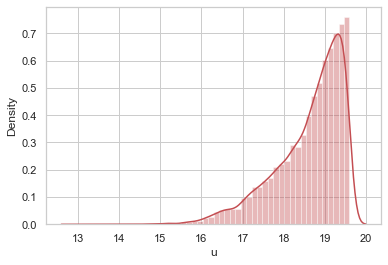
We can also check the outliers individually.

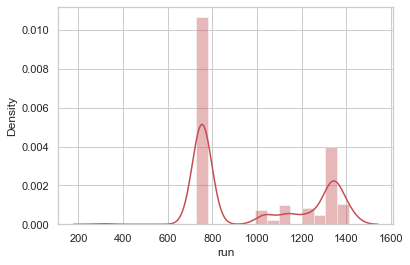
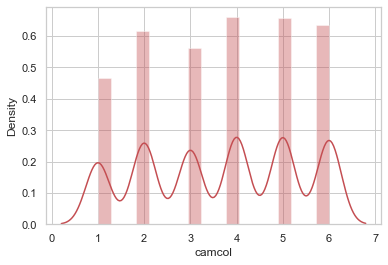
# Dropping columns :

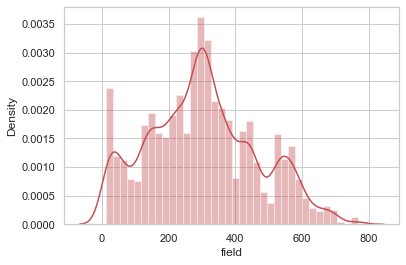
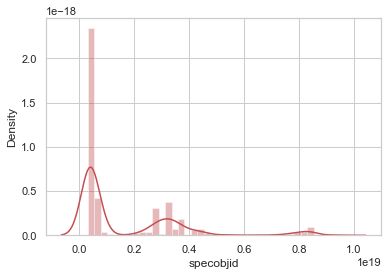
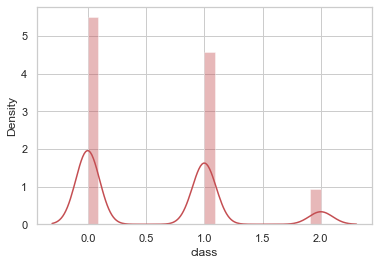
df.drop(['objid','rerun','ra','dec'], axis = 1,inplace=**True**)

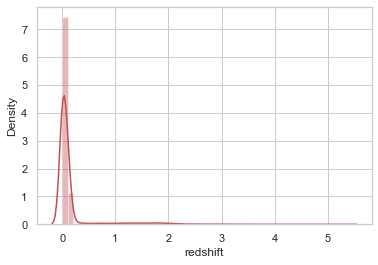
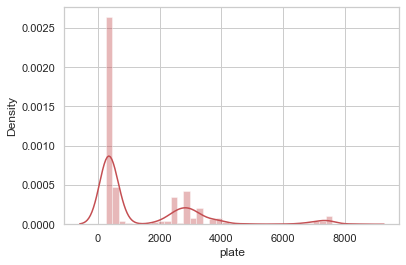
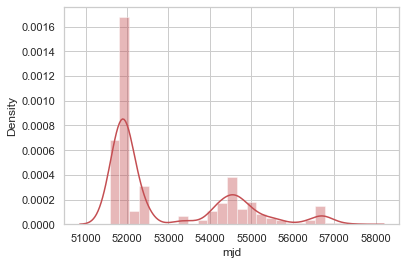
1. These objid,rerun,ra,dec four colums will be deleted. Because there are either one single value or all the values are unique.
2. fiberid and field having outliers. But there are very few outliers. So, we will keep these two columns.

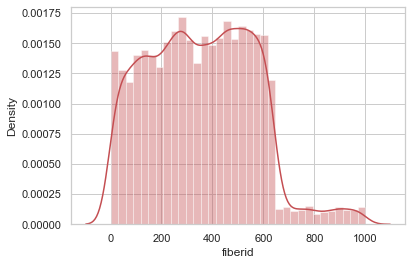
# Skewness:









In most of the columns skewness is present. Such as u. Left skewness is present here.In the Specobjid,class,redshift,plate,mjd columns right skewness is present.

**df.skew()**

u -1.219795

g -0.536293

r -0.021673

i 0.286442

z 0.214313

run 0.412555

camcol -0.100220

field 0.249795

specobjid 1.794627

class 0.641904

redshift 4.265729

plate 1.794609

mjd 1.039610

fiberid 0.308053

dtype: float64

If we want to know the exact value then skew() function is the best way to know the skewness of the variavles. Here, The standard value I have used is 0.65. If the value is not in between -0.56 and 0.56 that means skewness is present in those columns.

### Treating Skewness via root method and cube root method:

df.skew()

**for** col **in** df.skew().index:

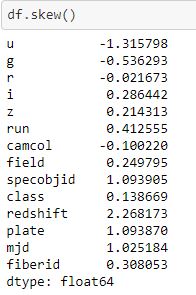
**if** col **in** df.describe().columns:

**if** df[col].skew()>0.55:

df[col]=np.sqrt(df[col])

**if** df[col].skew()<-0.55:

df[col]=np.cbrt(df[col])



After treating the skewness it has been reduced than before.

# **Removing Outliers :**

**IQR**[**¶**](https://render.githubusercontent.com/view/ipynb?color_mode=light&commit=0b6bdb66b0ab80e5723941a5e5938db19ad0655a&enc_url=68747470733a2f2f7261772e67697468756275736572636f6e74656e742e636f6d2f4d64526173686964756e6e6162692f4576616c756174696f6e2d50726f6a6563742f306236626462363662306162383065353732333934316135653539333864623139616430363535612f736b797365727665722e6970796e62&nwo=MdRashidunnabi%2FEvaluation-Project&path=skyserver.ipynb&repository_id=337394635&repository_type=Repository#IQR)

q1=df.quantile(0.25)

q3=df.quantile(0.75)

IQR=q3-q1

print(IQR)

df\_new1=df[~((df<(q1-1.5\*IQR))|(df>(q3+1.5\*IQR))).any(axis=1)]

print(df\_new1.shape)

u 5.112910e-02

g 1.195045e+00

r 1.339343e+00

i 1.404845e+00

z 1.523162e+00

run 5.790000e+02

camcol 3.000000e+00

field 2.300000e+02

specobjid 1.115267e+09

class 1.000000e+00

plate 3.323721e+01

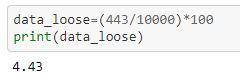
mjd 5.568089e+00

fiberid 3.232500e+02

dtype: float64

(9557, 13)

I have used IQR method to remove outliers. After removing outliers the new dataset is having 9557 rows and 13 columns.



After removing the outliers we can see that the we have losen only 4.43% data which is not huge. So, we can accept with the new dataset called “df\_new1”

# **Model Training :**

### Spilitting the data into input and output variable :

x=df\_new1.drop(columns=['class'],axis=1)

y=df\_new1['class']

We can split the data into x and y. x is having all the columns except the target variable. Y is having only the target column.

# **Scaling in input variables :**

**from** **sklearn.preprocessing** **import** StandardScaler

ss=StandardScaler()

x=ss.fit\_transform(x)

By using the following code we can scale our data. Because in some of the columns the range is too high.

### Spliting the data into training and testing set :

**from** **sklearn.model\_selection** **import** train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=.20,random\_state=42,stratify=y)

# **Importing all the model Library :**

*# Libraries for data modelling*

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.tree** **import** DecisionTreeClassifier

**from** **sklearn.neighbors** **import** KNeighborsClassifier

**from** **sklearn.naive\_bayes** **import** GaussianNB

**from** **sklearn.svm** **import** SVC

**from** **sklearn.neighbors** **import** KNeighborsClassifier

*#Importinf boosting models*

**from** **sklearn.ensemble** **import** RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, BaggingClassifier, ExtraTreesClassifier

*#Importing error metrics*

**from** **sklearn.metrics** **import** accuracy\_score,confusion\_matrix,classification\_report

**from** **sklearn.model\_selection** **import** GridSearchCV,cross\_val\_score

# **All algorithms are in one code :**

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("**\033**[1m"+ 'Accuracy score of',m,'is : ' + "**\033**[0m" ) *# Make the line bold*

print(accuracy\_score(y\_test,predm))

print(confusion\_matrix(y\_test,predm))

print(classification\_report(y\_test,predm))

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('**\n**')

By using all the algoriths one by one we can use one function to implement all the algorithms. If we summarize the result we get the following accuracy :

1. Accuracy score of LogisticRegression() is : 0.9314853556485355

2. Accuracy score of GaussianNB() is : 0.8875523012552301

3. Accuracy score of SVC() is : 0.9382845188284519

4. Accuracy score of DecisionTreeClassifier() is : 0.9884937238493724

5. Accuracy score of KNeighborsClassifier() is : 0.9095188284518828

6. Accuracy score of RandomForestClassifier() is : 0.9884937238493724

7. Accuracy score of AdaBoostClassifier() is : 0.9879707112970711

8. Accuracy score of GradientBoostingClassifier() is : 0.9879707112970711

9. Accuracy score of BaggingClassifier() is : 0.9879707112970711

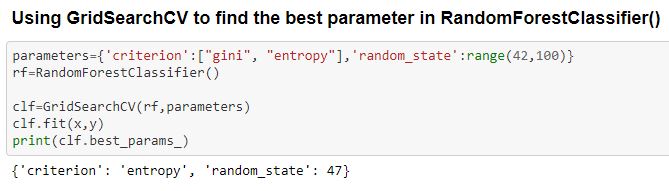
10. Accuracy score of ExtraTreesClassifier() is : 0.981694560669456

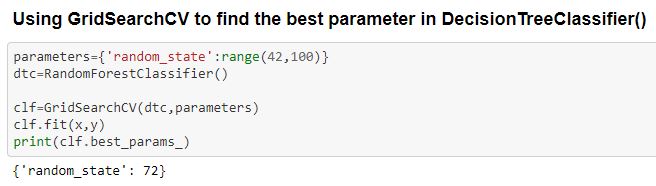
We have got good accuracy score by using the following algorithms :

DecisionTreeClassifier(),RandomForestClassifier(),AdaBoostClassifier(),GradientBoostingClassifier(),BaggingClassifier(),ExtraTreesClassifier()

But out of these algorithms RandomForestClassifier()and DecisionTreeClassifier() are giving the best result and exactly same. We will do the hyperparameter tuning to reduce the overfitting.

# **Using Best Parameter :**

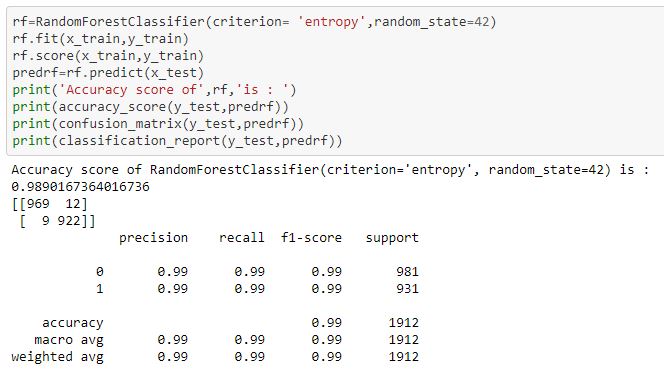


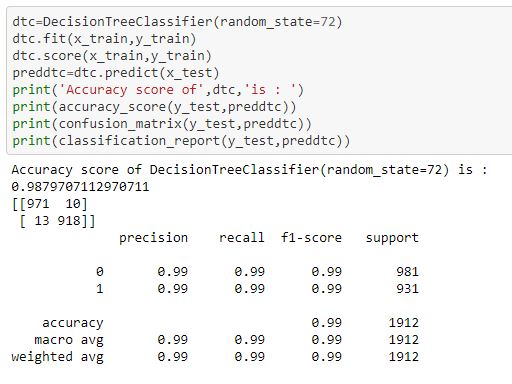


Here the best parametes for RandomForestClassifier() are 'criterion': 'entropy', 'random\_state': 47.

For DecisionTreeClassifier() the best random state is 72

# **Using Best Parameter :**

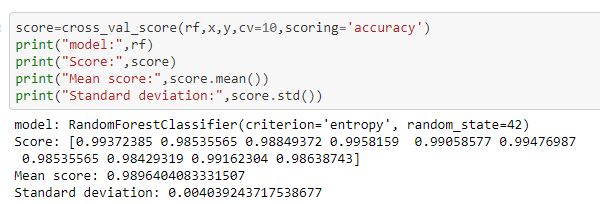


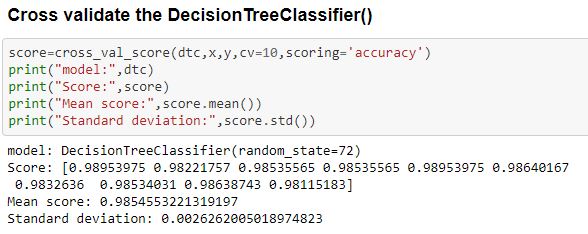


After using the best parametes we have got the accuracy for RandomForestClassifier(). The accuracy is 0.9890167364016736.

For DecisionTreeClassifier() the accuracy is : 0.9879707112970711. Although both of the accuracies are almost same but the accuracy for RandomForestClassifier() is better than DecisionTreeClassifier(). So, finallywe will use RandomForestClassifier() as our model.

# **Cross Validation score :**

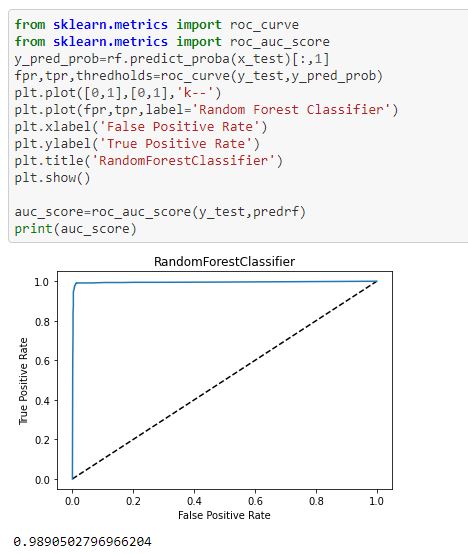




### Plotting Confusion matrix for RandomForestClassifier() :

### 

# Auc\_Roc Curve and finding auc score:



We can see that the auc\_score is almost 99. That means our our model is giving very good performance.

# Saving the model

**import** **joblib**

# Save the model as a pickle in a file

joblib.dump(rf,'skyserver.pkl')

['skyserver.pkl']

To save our model first we need to import joblib. Then we can save our model as pkl file.

You can get the whole sourcecode from the following link :

<https://github.com/MdRashidunnabi/Evaluation-Project/blob/main/skyserver.ipynb>

Name : Md Rashidunnabi

Email : [rashidunnabi11@gmail.com](mailto:rashidunnabi11@gmail.com)

Phone : +8801319527349

Whatsapp : +8801319527349