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()
```

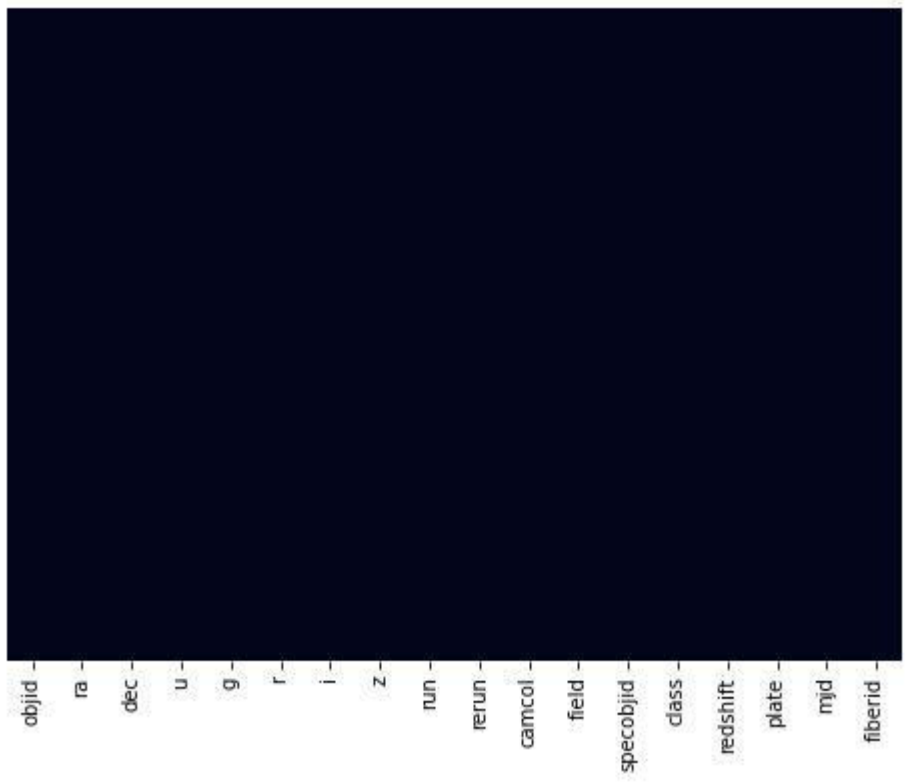
```
objid        0
ra           0
dec          0
u            0
g            0
r            0
i            0
z            0
run          0
rerun        0
camcol       0
field        0
specobjid    0
class        0
redshift     0
plate        0
mjd          0
fiberid      0
dtype: int64
```

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

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

```
plt.figure(figsize=(8,6))
sns.heatmap(df.isnull(),yticklabels=False,cbar=False)
```

```
<AxesSubplot:>
```



By any method we will get the same results.

Exploratory the catagorical columns :

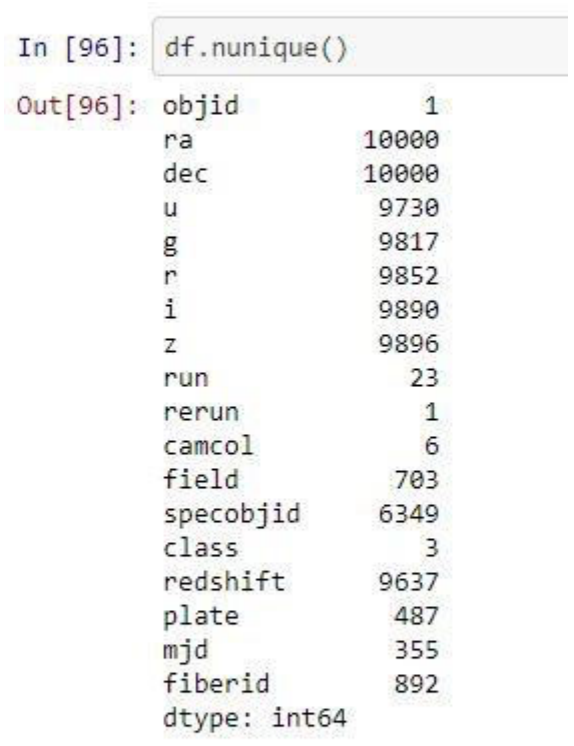
```
for column in df.columns:
    if df[column].dtype==object:
        print(str(column) + ':' + str(df[column].unique()))
        print(df[column].value_counts())
        print('*****************************************************************************************')
        print("\n")
```

```
class:['STAR' 'GALAXY' 'QSO']
GALAXY    4998
STAR      4152
QSO        850
Name: class, dtype: int64
***********************************************************************************
```

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 4
998 times; STAR  4152 times and QSO  850 times.
```

Checking Unique values :



```
In [96]: df.nunique()

Out[96]: objid            1
         ra           10000
         dec          10000
         u             9730
         g             9817
         r             9852
         i             9890
         z             9896
         run             23
         rerun            1
         camcol           6
         field          703
         specobjid     6349
         class            3
         redshift      9637
         plate          487
         mjd            355
         fiberid        892
         dtype: int64
```

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:

```
df['class']=df['class'].map({'GALAXY':0,'STAR':1,'QSO':2})
df['class'].dtypes
```

```
dtype('int64')
```

```
df['class'].unique()
```

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.

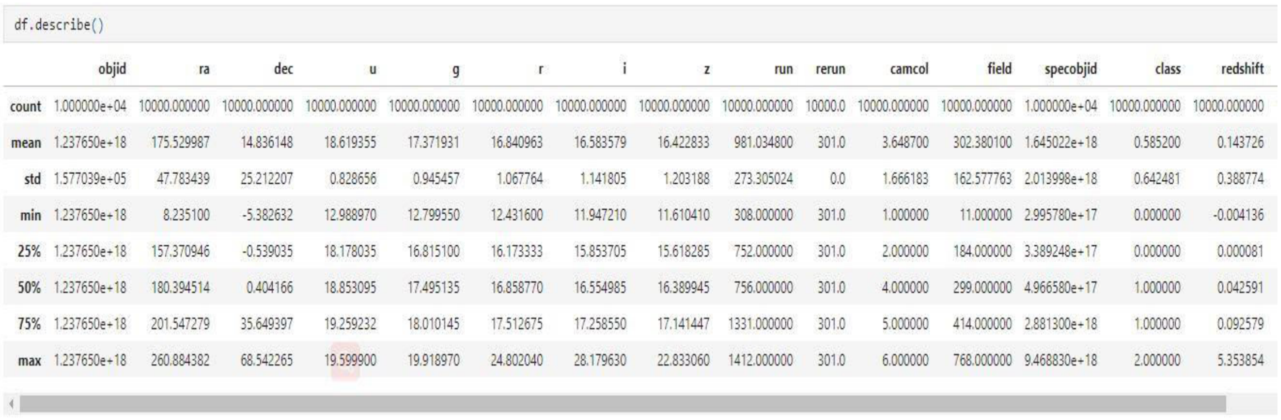
Observations :

0 stands for GALAXY

1 stands for STAR.

2 stands for QSO

Summary Statistics :



```
df.describe()
```

	objid	ra	dec	u	g	r	i	z	run	rerun	camcol	field	specobjid	class	redshift
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0	10000.000000	10000.000000	1.000000e+04	10000.000000	10000.000000
mean	1.237650e+18	175.529987	14.836148	18.619355	17.371931	16.840963	16.583579	16.422833	981.034800	301.0	3.648700	302.380100	1.645022e+18	0.585200	0.143726
std	1.577039e+05	47.783439	25.212207	0.828656	0.945457	1.067764	1.141805	1.203188	273.305024	0.0	1.666183	162.577763	2.013998e+18	0.642481	0.388774
min	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
25%	1.237650e+18	157.370946	-0.539035	18.178035	16.815100	16.173333	15.853705	15.618285	752.000000	301.0	2.000000	184.000000	3.389248e+17	0.000000	0.000081
50%	1.237650e+18	180.394514	0.404166	18.853095	17.495135	16.858770	16.554985	16.389945	756.000000	301.0	4.000000	299.000000	4.966580e+17	1.000000	0.042591
75%	1.237650e+18	201.547279	35.649397	19.259232	18.010145	17.512675	17.258550	17.141447	1331.000000	301.0	5.000000	414.000000	2.881300e+18	1.000000	0.092579
max	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

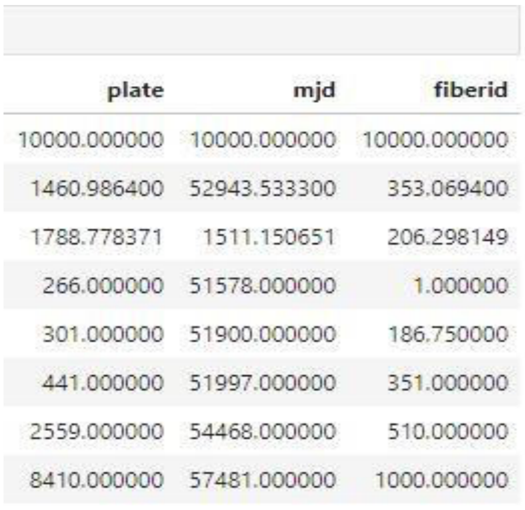


plate	mjd	fiberid
10000.000000	10000.000000	10000.000000
1460.986400	52943.533300	353.069400
1788.778371	1511.150651	206.298149
266.000000	51578.000000	1.000000
301.000000	51900.000000	186.750000
441.000000	51997.000000	351.000000
2559.000000	54468.000000	510.000000
8410.000000	57481.000000	1000.000000

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.

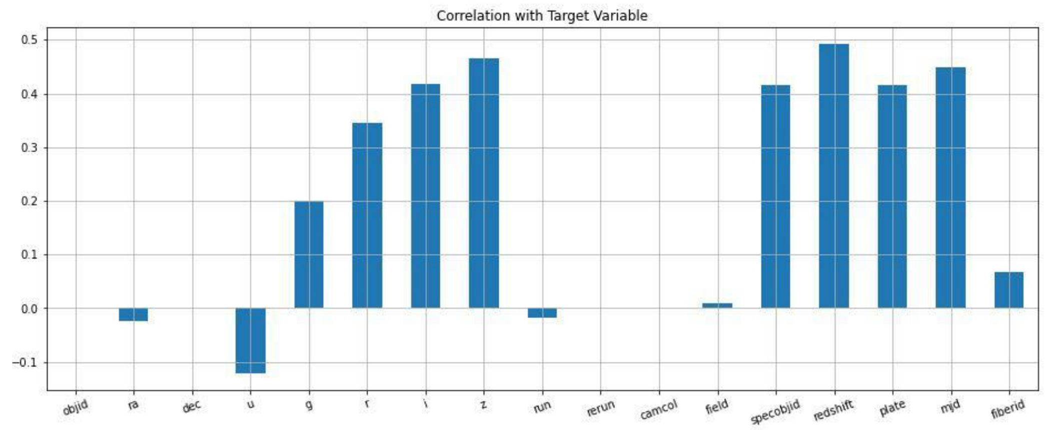
5. The difference between 75% and maximum is higher in run,field,specobjid,redshift,plate,mjd and fiberid columns therefore outliers are present in these columns.

Correlation:

Correlation only with target variable :

```
plt.figure(figsize=(16,6))
df.drop('class',axis=1).corrwith(df['class']).plot(kind='bar',grid=True)
plt.xticks(rotation=20)
plt.title("Correlation with Target Variable")
```

```
Text(0.5, 1.0, 'Correlation 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.

```
df[df.columns[1:]].corr()['class'][:]
```

```
ra          -0.023999
dec          0.001451
u           -0.122010
g            0.200748
r            0.344900
i            0.417631
z            0.465475
run         -0.017958
rerun             NaN
camcol       0.000823
field        0.008897
specobjid    0.414677
class        1.000000
redshift     0.492611
plate        0.414678
mjd          0.448355
fiberid      0.067150
Name: class, dtype: float64
```

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

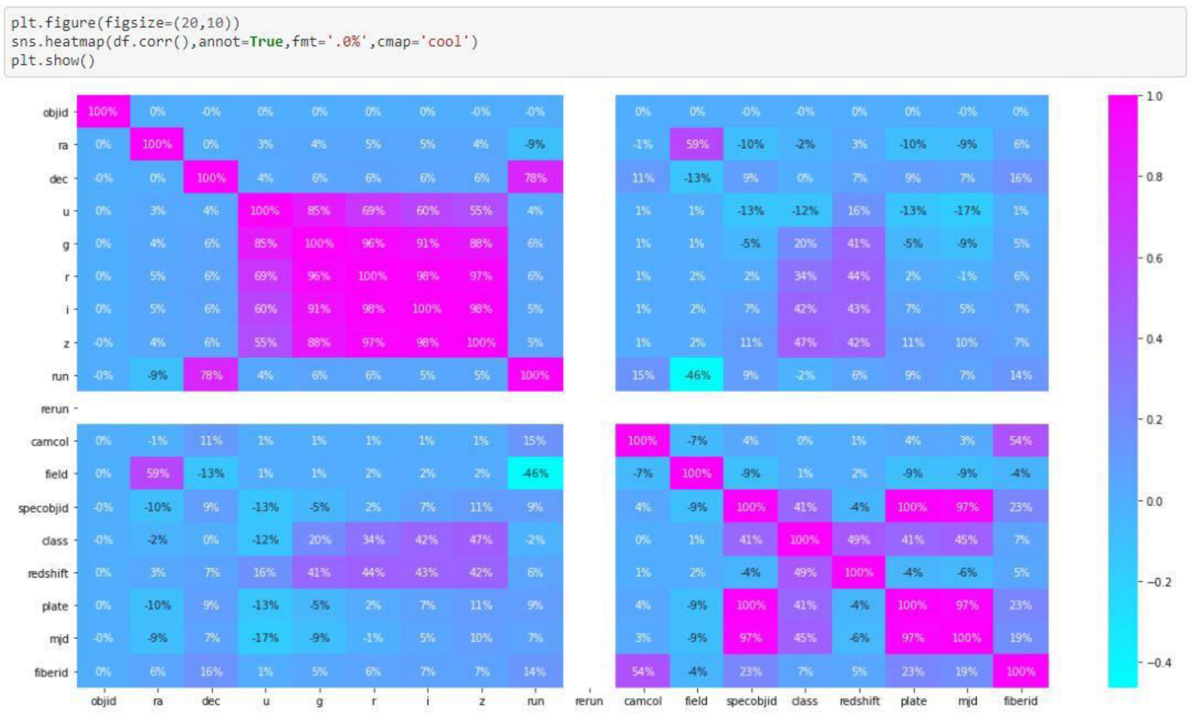
2. Positive correlation : dec,g, r, i, z,camcol, field, specobjid, redshift, plate, mjd, fi berid

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 o utliers then these columns should be deleted.

Correlation among all the variables:



```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(),annot=True,fmt='.0%',cmap='cool')
plt.show()
```

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 targe
t 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

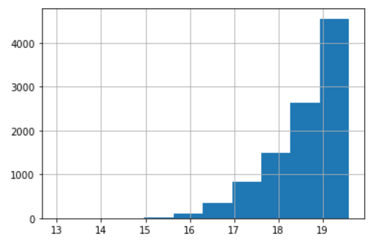
We already change it to integer valued as it contains three numbers 0,1 and 2

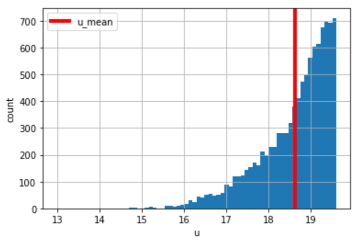
0 stands for GALAXY

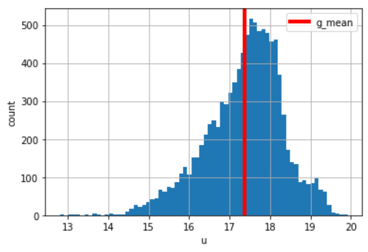
1 stands for STAR.

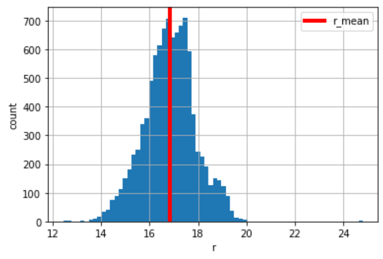
2 stands for QSO

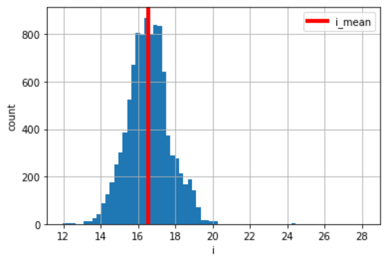
Univariate anslysis :

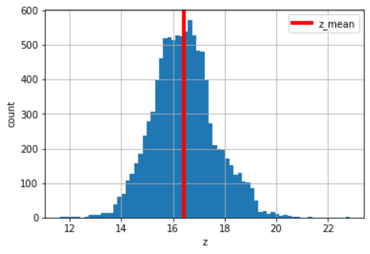


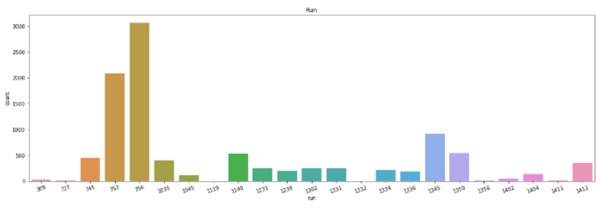


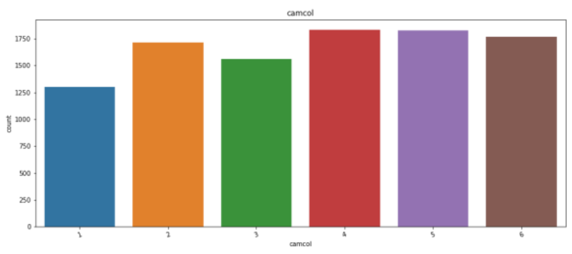


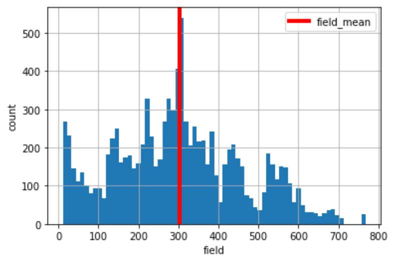


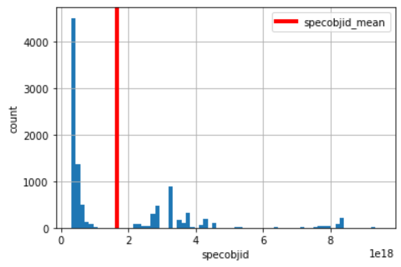


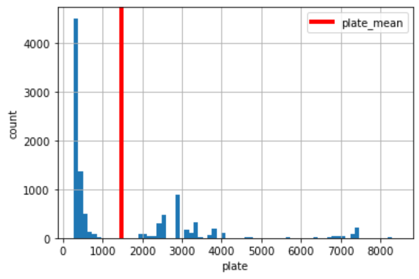


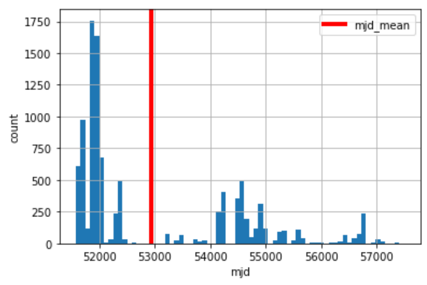


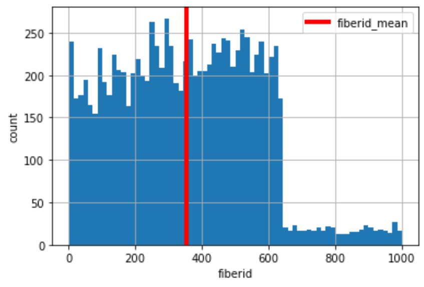












In some columns the values are continuous type. For continuous type of data I have used histogram with the red coloured mean value to visualize every column perfectly. Some of the columns are catagorical and there are few specific values. In these cases I have used countplot.

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()
```

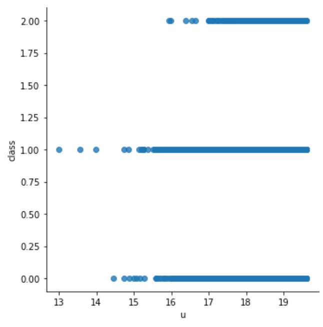
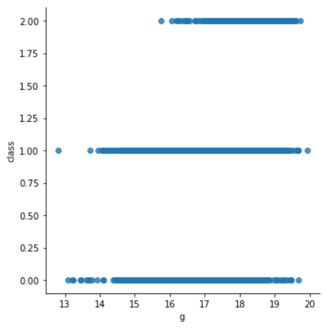
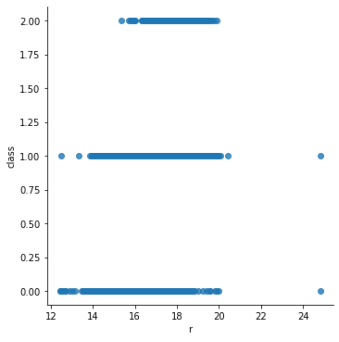
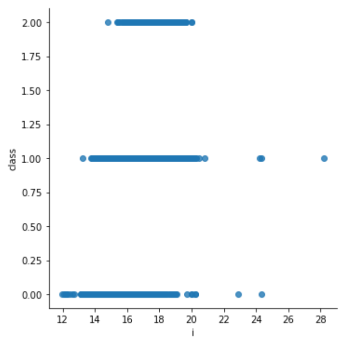
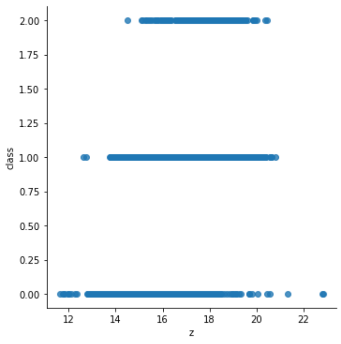
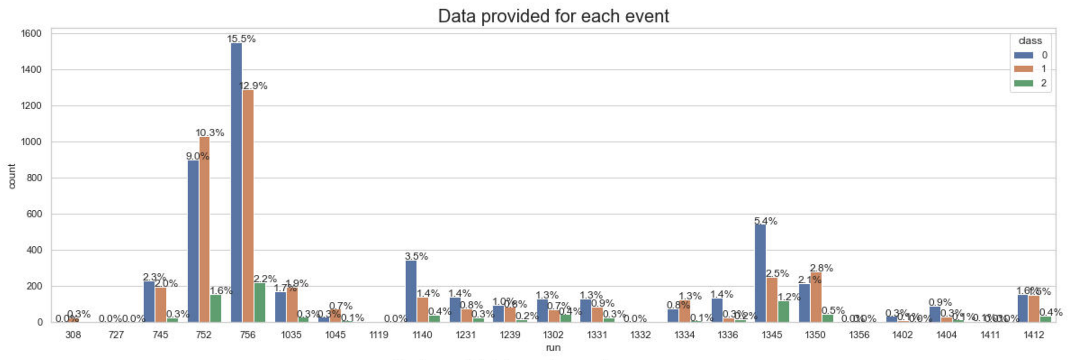
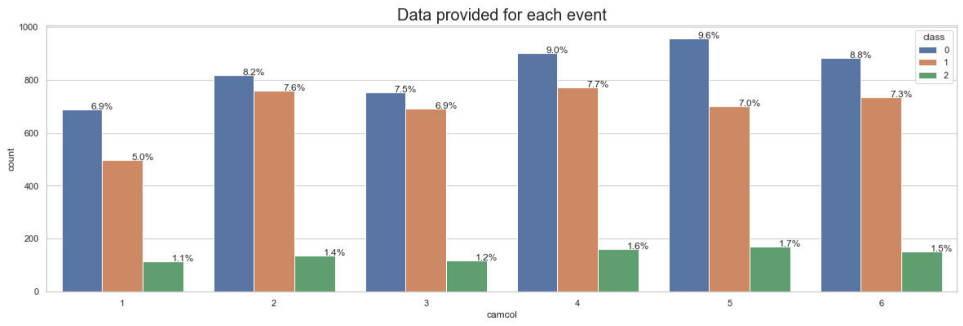
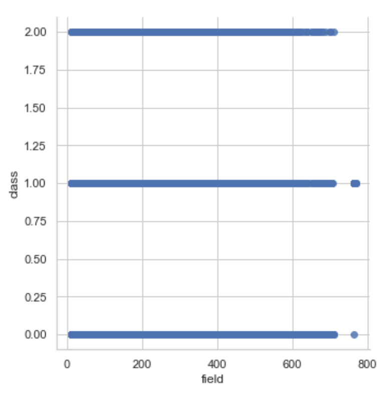
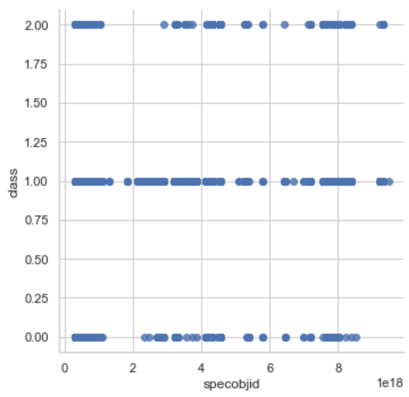
Sample code for countplot :

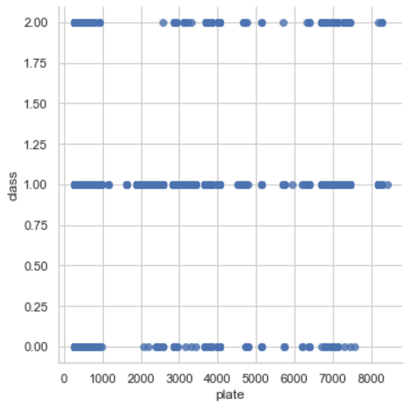
```
plt.subplots(figsize=(20,6))
sns.countplot(x='Column Name',data=df)

plt.title('Column Name')

plt.xticks(rotation=20)
plt.show()

print(df.Column Name.value_counts())
```

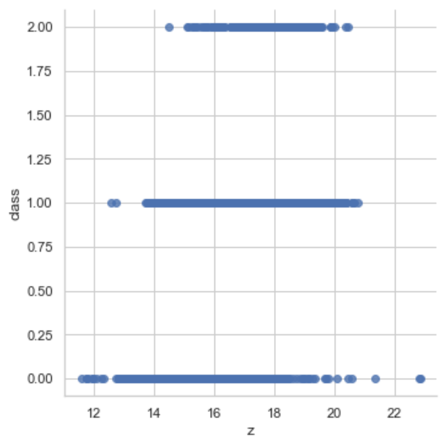
From the above figure we can explore the visualization of 'u', 'g', 'r', 'i', 'z', 'run', 'camcol', 'fie ld', 'specobjid', 'plate', 'mjd', 'fiberid' columns.

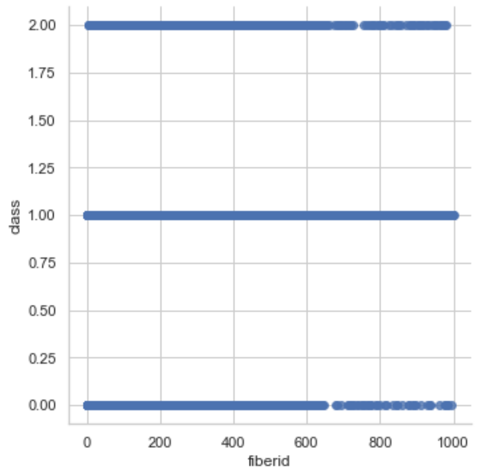
In the g column most of the values are in between 17.2 and 18.2.Here , the mean value is 17. 3.If we look at the ‘run’ column we can see that the value 756 has been appeared most of th e times(3060). And then the value 752 which has been appeared 2086 times. By using count plot we can basically know about the frequency level.

Bivariate Analysis :


class
u

class
g

class
r

class
i

class
z

Data provided for each event
count
run
class
0
1
2

Data provided for each event
count
camcol
class
0
1
2
6.9%
5.0%
1.1%
8.2%
7.6%
1.4%
7.5%
6.9%
1.2%
9.0%
7.7%
1.6%
9.6%
7.0%
1.7%
8.8%
7.3%
1.5%

class
field

class
specobjid
1e18







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 :

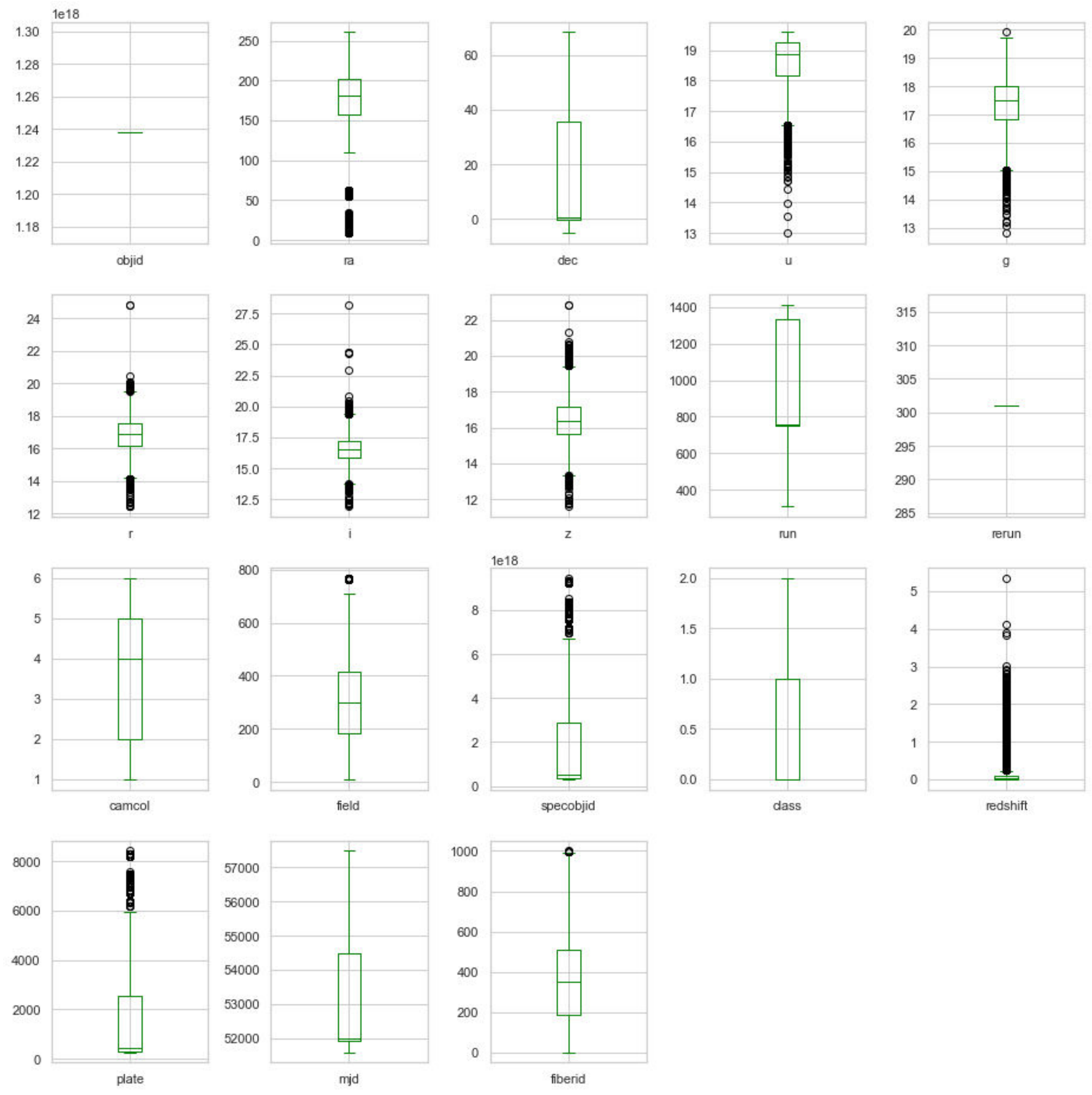
```
plt.figure(figsize=(8,4))
sns.lmplot(x='column_name',y='Target_variable',fit_reg=False,data=df)
plt.show()
```

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.

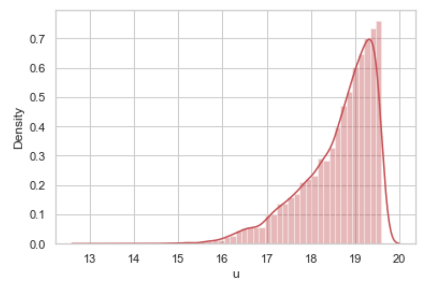
We can also check the outliers individually.

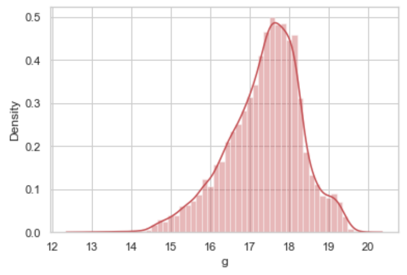
Dropping columns :

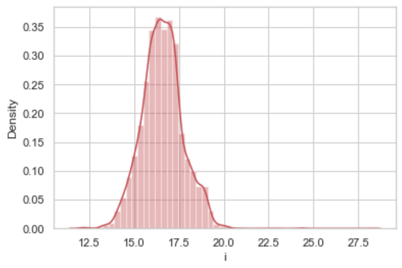
```
df.drop(['objid','rerun','ra','dec'], axis = 1,inplace=True)
```

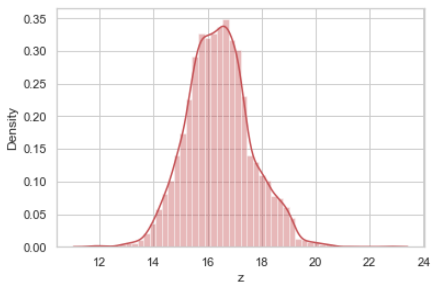
1. These objid,rerun,ra,dec four colums will be deleted. Because there are either one single value or all the values are unique.
2. fiberid and field having outliers. But there are very few outliers. So, we will keep these two columns.

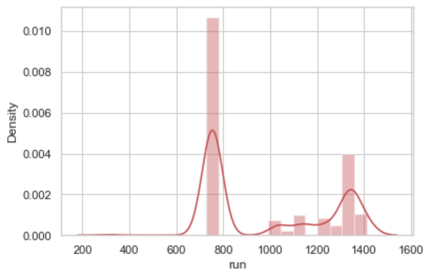
Skewness:

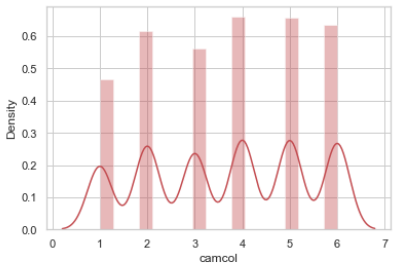


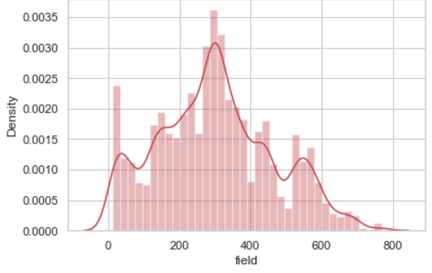


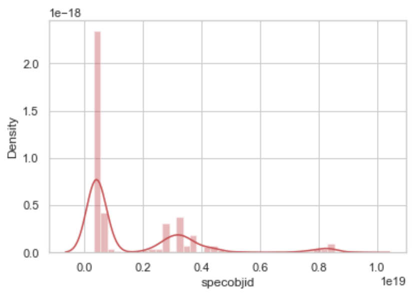


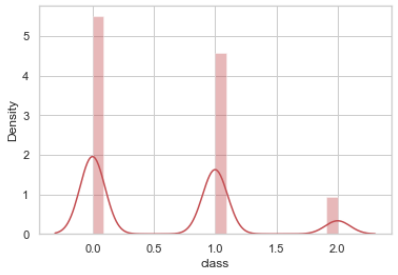


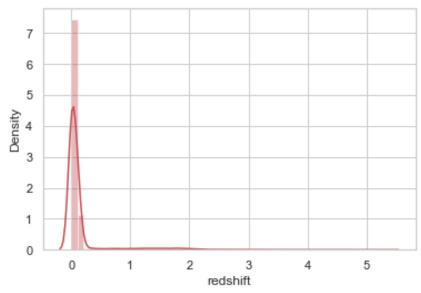


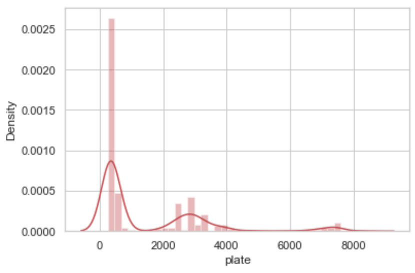


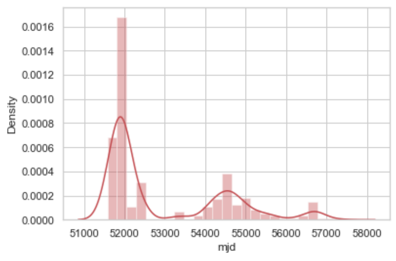


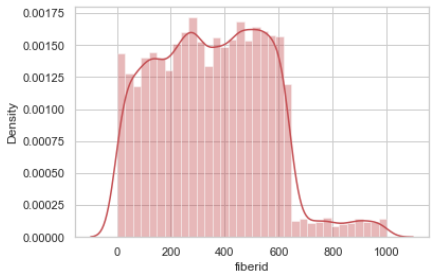












In most of the columns skewness is present. Such as u. Left skewness is present here.In the Specobjid,class,redshift,plate,mjd columns right skewness is present.

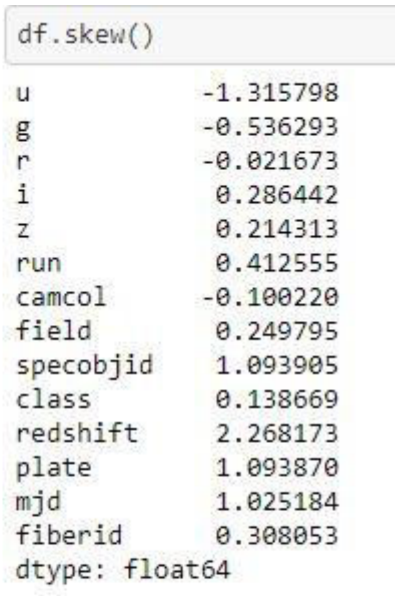
```
df.skew()
```

```
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])
```



```
df.skew()

u            -1.315798
g            -0.536293
r            -0.021673
i             0.286442
z             0.214313
run           0.412555
camcol       -0.100220
field         0.249795
specobjid     1.093905
class         0.138669
redshift      2.268173
plate         1.093870
mjd           1.025184
fiberid       0.308053
dtype: float64
```

After treating the skewness it has been reduced than before.

Removing Outliers :

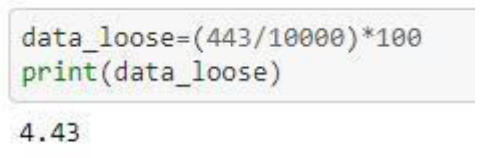
IQR¶

```
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.



```
data_loose=(443/10000)*100
print(data_loose)
```

```
4.43
```

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,classificat
ion_report
from sklearn.model_selection import GridSearchCV,cross_val_score
```

All algorithms are in one code :

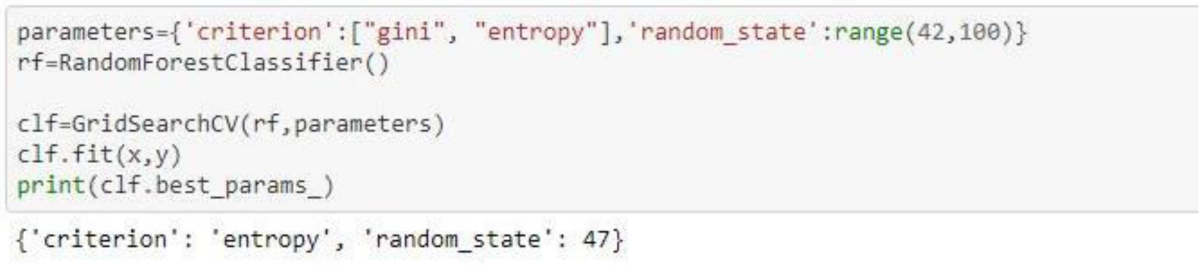
```
model=[LogisticRegression(),GaussianNB(),SVC(),DecisionTreeClassifier(),
KNeighborsClassifier(),RandomForestClassifier(),AdaBoostClassifier(),Gr
adientBoostingClassifier(),BaggingClassifier(),ExtraTreesClassifier()]

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" )  # Mak
e 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 : 0.931485355648535
5

2. Accuracy score of GaussianNB() is : 0.8875523012552301

3. Accuracy score of SVC() is : 0.9382845188284519

4. Accuracy score of DecisionTreeClassifier() is : 0.98849372384
93724

5. Accuracy score of KNeighborsClassifier() is : 0.9095188284518
828

6. Accuracy score of RandomForestClassifier() is : 0.98849372384
93724

7. Accuracy score of AdaBoostClassifier() is : 0.987970711297071
1

8. Accuracy score of GradientBoostingClassifier() is : 0.9879707
112970711

9. Accuracy score of BaggingClassifier() is : 0.9879707112970711

10. Accuracy score of ExtraTreesClassifier() is : 0.981694560669
456
```

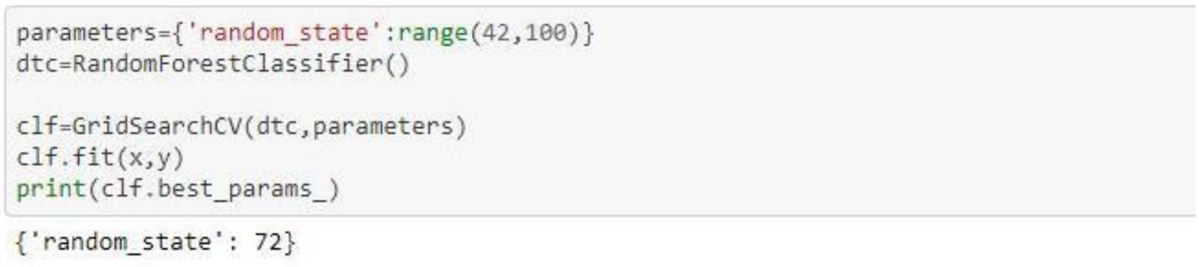
We have got good accuracy score by using the following algorithms :
DecisionTreeClassifier(),RandomForestClassifier(),AdaBoostClassifier(),GradientBoostingClassifier(),BaggingClassifier(),ExtraTreesClassifier()
But out of these algorithms RandomForestClassifier()and DecisionTreeClassifier() are giving the best result and exactly same. We will do the hyperparameter tuning to reduce the overfitting.

Using Best Parameter :



```
parameters={'criterion':["gini", "entropy"],'random_state':range(42,100)}
rf=RandomForestClassifier()

clf=GridSearchCV(rf,parameters)
clf.fit(x,y)
print(clf.best_params_)
```

```
{'criterion': 'entropy', 'random_state': 47}
```

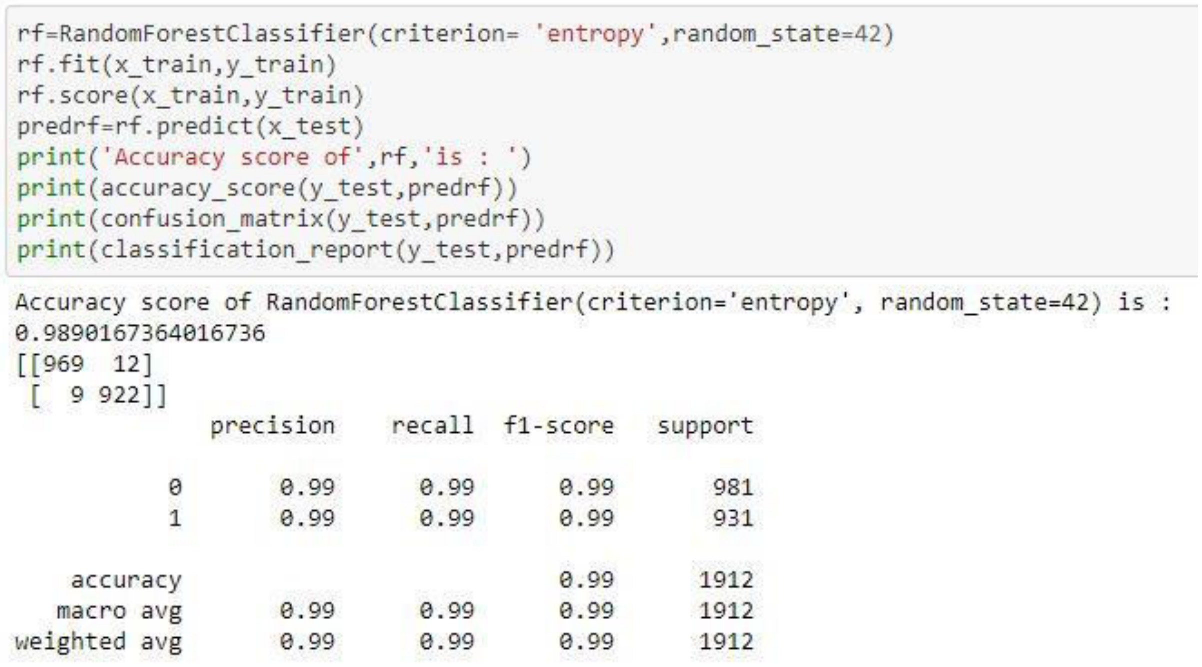


```
parameters={'random_state':range(42,100)}
dtc=RandomForestClassifier()

clf=GridSearchCV(dtc,parameters)
clf.fit(x,y)
print(clf.best_params_)
```

```
{'random_state': 72}
```

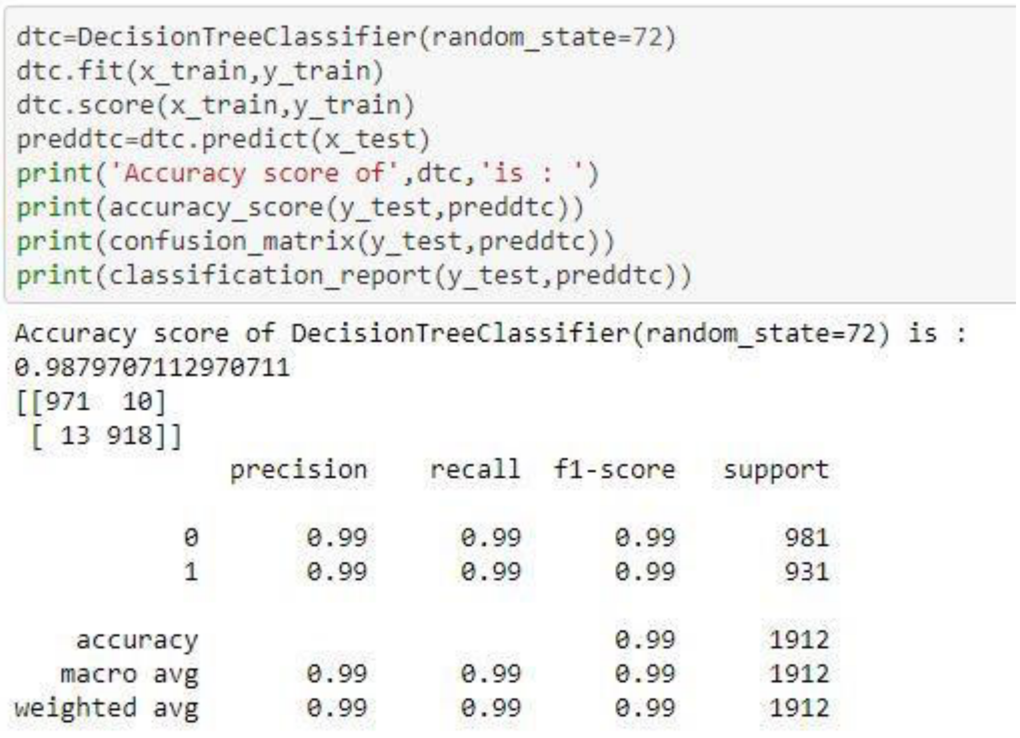
Here the best parametes for RandomForestClassifier() are 'criterion': 'entropy', 'random_st ate': 47.
For DecisionTreeClassifier() the best random state is 72

Using Best Parameter :



```
rf=RandomForestClassifier(criterion= 'entropy',random_state=42)
rf.fit(x_train,y_train)
rf.score(x_train,y_train)
predrf=rf.predict(x_test)
print('Accuracy score of',rf,'is : ')
print(accuracy_score(y_test,predrf))
print(confusion_matrix(y_test,predrf))
print(classification_report(y_test,predrf))
```

```
Accuracy score of RandomForestClassifier(criterion='entropy', random_state=42) is :
0.9890167364016736
[[969  12]
 [  9 922]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       981
           1       0.99      0.99      0.99       931

    accuracy                           0.99      1912
   macro avg       0.99      0.99      0.99      1912
weighted avg       0.99      0.99      0.99      1912
```



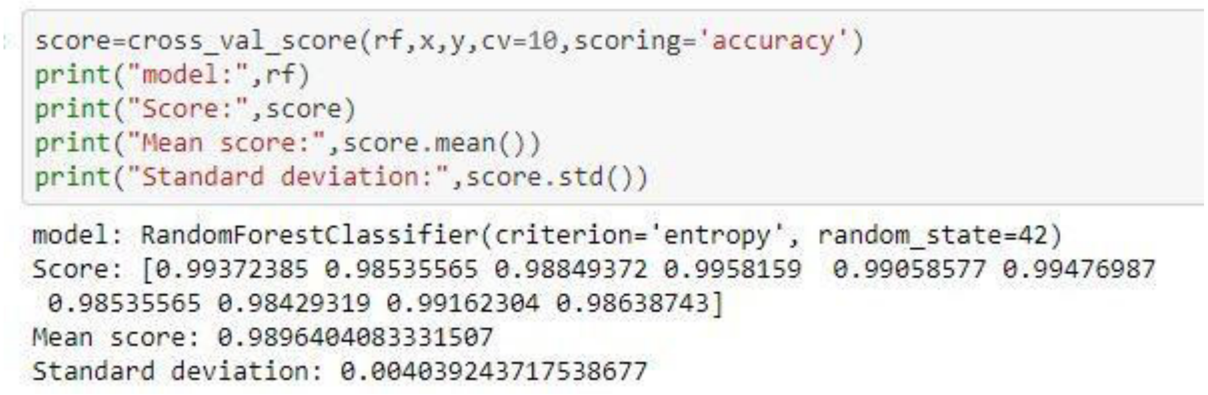
```
dtc=DecisionTreeClassifier(random_state=72)
dtc.fit(x_train,y_train)
dtc.score(x_train,y_train)
preddtc=dtc.predict(x_test)
print('Accuracy score of',dtc,'is : ')
print(accuracy_score(y_test,preddtc))
print(confusion_matrix(y_test,preddtc))
print(classification_report(y_test,preddtc))
```

```
Accuracy score of DecisionTreeClassifier(random_state=72) is :
0.9879707112970711
[[971  10]
 [ 13 918]]
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       981
           1       0.99      0.99      0.99       931

    accuracy                           0.99      1912
   macro avg       0.99      0.99      0.99      1912
weighted avg       0.99      0.99      0.99      1912
```

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

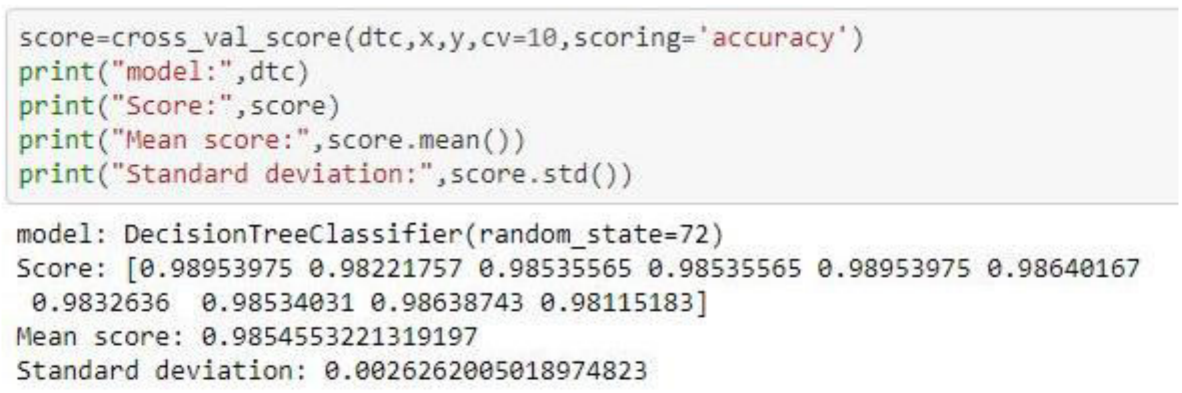
For DecisionTreeClassifier() the accuracy is : 0.9879707112970711. Although both of the acc uracies are almost same but the accuracy for RandomForestClassifier() is better than Decisio nTreeClassifier(). So, finallywe will use RandomForestClassifier() as our model.

Cross Validation score :



```
score=cross_val_score(rf,x,y,cv=10,scoring='accuracy')
print("model:",rf)
print("Score:",score)
print("Mean score:",score.mean())
print("Standard deviation:",score.std())
```

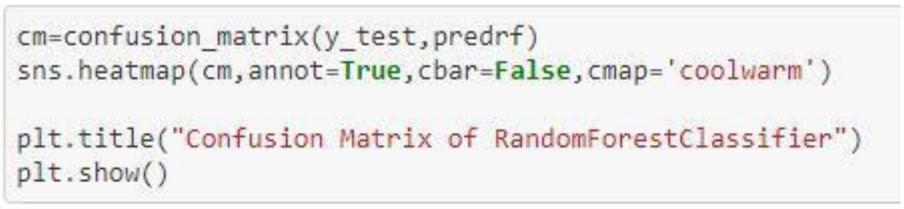
```
model: RandomForestClassifier(criterion='entropy', random_state=42)
Score: [0.99372385 0.98535565 0.98849372 0.9958159  0.99058577 0.99476987
 0.98535565 0.98429319 0.99162304 0.98638743]
Mean score: 0.9896404083331507
Standard deviation: 0.004039243717538677
```



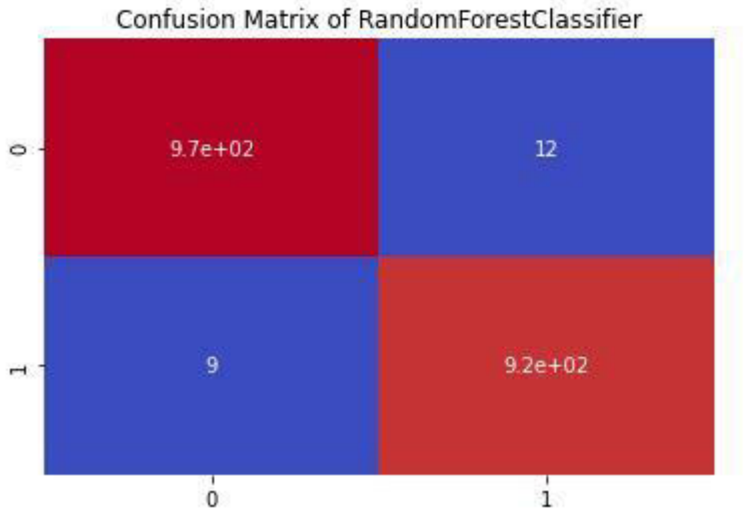
```
score=cross_val_score(dtc,x,y,cv=10,scoring='accuracy')
print("model:",dtc)
print("Score:",score)
print("Mean score:",score.mean())
print("Standard deviation:",score.std())
```

```
model: DecisionTreeClassifier(random_state=72)
Score: [0.98953975 0.98221757 0.98535565 0.98535565 0.98953975 0.98640167
 0.9832636  0.98534031 0.98638743 0.98115183]
Mean score: 0.9854553221319197
Standard deviation: 0.0026262005018974823
```

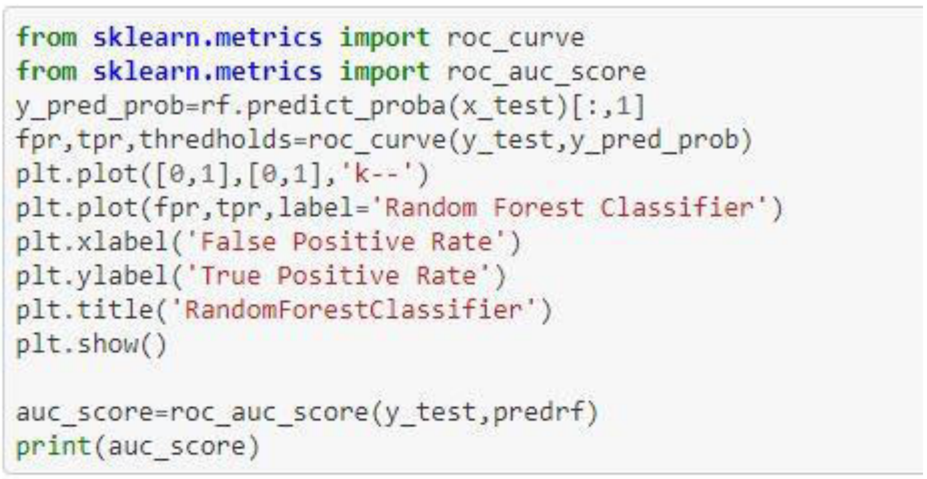
Plotting Confusion matrix for RandomForestClassifier() :



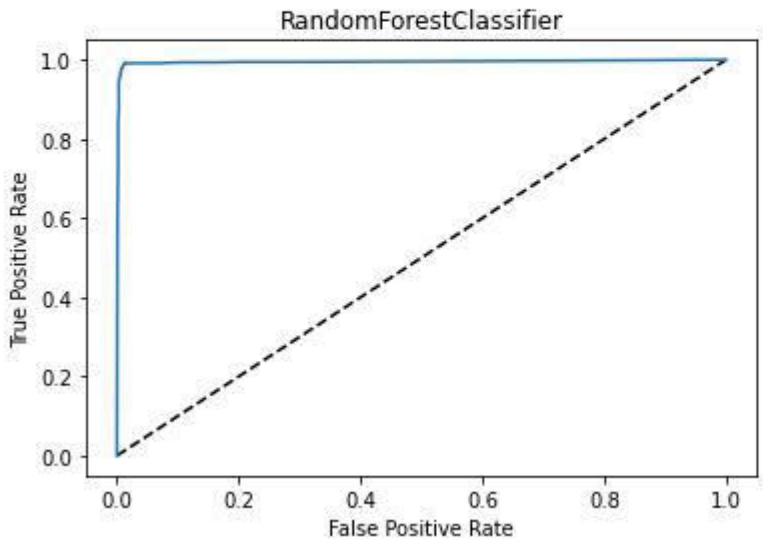
```
cm=confusion_matrix(y_test,predrf)
sns.heatmap(cm,annot=True,cbar=False,cmap='coolwarm')

plt.title("Confusion Matrix of RandomForestClassifier")
plt.show()
```



Auc_Roc Curve and finding auc score:



```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
y_pred_prob=rf.predict_proba(x_test)[:,1]
fpr,tpr,thredholds=roc_curve(y_test,y_pred_prob)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label='Random Forest Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('RandomForestClassifier')
plt.show()

auc_score=roc_auc_score(y_test,predrf)
print(auc_score)
```



```
0.9890502796966204
```

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

Phone : +8801319527349

Whatsapp : +8801319527349