Import Libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

import os

import matplotlib.pyplot as plt
plt.style.use("ggplot")

import seaborn as sb
sb.set(rc = {'figure.figsize':(12,7)})

import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import LabelEncoder


from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler, SMOTE
oversampler = SMOTE(random_state=1)

from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier

from sklearn.metrics import accuracy_score
from sklearn.model_selection import StratifiedKFold as skf
skfold = skf(n_splits = 4, shuffle = True, random_state = 70)
from sklearn.model_selection import cross_val_score

from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
```

Read CSV and load data

```
star =
pd.read_csv('../input/stellar-classification-dataset-sdss17/star_class
ification.csv')
```

Exploratory data analysis

```
star.head()

star.tail()

#A rough overview of data
star.info()

#Checking if any values are null......
star.isnull().sum()

a, b, c = star["class"].value_counts() / len(star)
print(f"Total percentage of Galaxies : {round(a*100, 1)}%")
print(f"Total percentage of Stars : {round(b*100, 1)}%")
print(f"Total percentage of QSO : {round(c*100, 1)}%")

sb.countplot(x = star["class"], palette="Set3")

star.describe()
```

Visualisation of data

```
#We can train out models faster if we choose those features which can
distinguish well between out classes.....
#As alpha, delta, u, g, r, i, z, redshift, plate, MJD are astronomical
quantities, therefore, we'll keep them as our primary feature...

for i in ['alpha', 'delta','redshift', 'plate', 'MJD']:
    plt.figure(figsize=(13,7))
    sb.histplot(data=star, x=i, kde=True, hue="class")
    plt.title(i)
    plt.show()

# Now i'll use KDE to visualise the photometric filters which are u,
g, r, i, z

le = LabelEncoder()
star["class"] = le.fit_transform(star["class"])
star["class"] = star["class"].astype(int)

def plot(column):
    for i in range(3):
        sb.kdeplot(data=star[star["class"] == i][column], label =
le.inverse_transform([i]), fill = True)
    sb.kdeplot(data=star[column],label = ["All"], fill = True)
```

```
    plt.legend();

def log_plot(column):
    for i in range(3):
        sb.kdeplot(data=np.log(star[star["class"] == i][column]),
label = le.inverse_transform([i]), fill = True)
    sb.kdeplot(data=np.log(star[column]),label = ["All"], fill = True)
    plt.legend();

plot('u')

#I'll apply log due to extreme values
log_plot('u')

plot('g')

#I'll apply log due to extreme values
log_plot('g')

plot('r')

plot('i')

plot('z')

#I'll apply log due to extreme values
log_plot('z')
```

Data Preprocessing

Cleaning of data

```
#Sometimes due to outliers, we can't do good analysis of our data
therefore, we decided to remove them as our current dataset contains
aloooooooooooot of outliers

def rem_outliers():
    s1 = star.shape

    for i in star.select_dtypes(include = 'number').columns:
        qt1 = star[i].quantile(0.25)
        qt3 = star[i].quantile(0.75)
        iqr =  qt3 - qt1
        lower = qt1-(1.5*iqr)
        upper = qt3+(1.5*iqr)
        min_in = star[star[i]<lower].index
        max_in = star[star[i]>upper].index
        star.drop(min_in, inplace = True)
        star.drop(max_in, inplace = True)

    s2 = star.shape
```

```
    outliers = s1[0] - s2[0]
    return outliers

print("Number of outliers deleted are : ", rem_outliers())
```

Data Modeling

```
#First of all we'll make a copy to for further training and testing
purpose
star = star.copy()

# I'ill drop the rest like run_id, rerun_id, field_id as they don't
make any significant difference in distribution by class
star.drop(['run_ID','rerun_ID','cam_col','field_ID','spec_obj_ID','fib
er_ID','obj_ID'],axis=1, inplace = True)

#Our target column is class therefore, we'll do test on it after
training on the rest of columns
X = star.drop('class',axis=1)
y = star['class']

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size =
0.7, shuffle=True, random_state=43)

#Making a standardize dataset
sc = StandardScaler()
sc.fit(X_train)

X_train = pd.DataFrame(sc.transform(X_train), columns =
X_train.columns, index = X_train.index)
X_test = pd.DataFrame(sc.transform(X_test), columns = X_test.columns,
index = X_test.index)

#Training the dataset
X_train_smote, y_train_smote = oversampler.fit_resample(X_train,
y_train)

models = {
    '        Linear Support Vector Machine : ': LinearSVC(),
    '        Decision Tree : ': DecisionTreeClassifier(),
    '        Random Forest Classifier : ': RandomForestClassifier(),
}

for model_name, model in models.items():
    model = model.fit(X_train_smote, y_train_smote)
    print(model_name + " Trained")
```

Validation of Models

```python
#Checking the validy of SVM, DecisionTree and random forest classifier model.....

for model_name, model in models.items():
    print(model_name + " {:.2f}%".format(model.score(X_test, y_test) * 100))

#Validating CatBoostClassifier model here using cross validation.....

model = CatBoostClassifier(verbose  = 0)
score = cross_val_score(model, X, y, cv = skfold)
print('CatBoostClassifier : ' + " {:.2f}%".format(np.mean(score) * 100))

#Validating LGBC model here using cross validation.....

model = LGBMClassifier()
score = cross_val_score(model , X , y ,cv=skfold)
print('Light GradientBoostingClassifier : ' + " {:.2f}%".format(np.mean(score) * 100))
```