Stellar Classification Dataset - SDSS17

Machine learning models applied

- Logistic Regression
- Decision Tree
- K Nearest Neighbors
- Bagging (or Bootstrap aggregating)
- Random Forest
- Naive Bayes
- Support vector machines (SVMs)
- Adaptive boosting or AdaBoost
- Gradient boosting
- XGBoost
- Voting classifier

Objective

- In astronomy, stellar classification is the classification of stars based on their spectral characteristics.
- The objective of this project is to classify stars, galaxies, and quasars based on their spectral characteristics.

Dataset source & brief

- The dataset has been sourced from kaggle and it contains 100,000 observations of space taken by the SDSS (Sloan Digital Sky Survey).

- Every observation is described by 17 feature columns and 1 class column which identifies it to be either a star, galaxy or quasar.
- The columns are obj_ID = Object Identifier, the unique value that identifies the object in the image catalog used by the CAS, alpha = Right Ascension angle (at J2000 epoch), delta = Declination angle (at J2000 epoch), u = Ultraviolet filter in the photometric system, g = Green filter in the photometric system, r = Red filter in the photometric system, i = Near Infrared filter in the photometric system, z = Infrared filter in the photometric system, run_ID = Run Number used to identify the specific scan, rereun_ID = Rerun Number to specify how the image was processed, cam_col = Camera column to identify the scanline within the run, field_ID = Field number to identify each field, spec_obj_ID = Unique ID used for optical spectroscopic objects (this means that 2 different observations with the same spec_obj_ID must share the output class), class = object class (galaxy, star or quasar object), redshift = redshift value based on the increase in wavelength, plate = plate ID, identifies each plate in SDSS, MJD = Modified Julian Date, used to indicate when a given piece of SDSS data was taken & fiber_ID = fiber ID that identifies the fiber that pointed the light at the focal plane in each observation

Import the libraries

In [94]:

```
import os
import numpy as np
import pandas as pd

#visualization
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

#evaluation
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
from sklearn.model_selection import cross_val_score

import warnings
warnings.filterwarnings('ignore')
```

Load and read the dataset

In [32]:

```
df=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stats and ML\Dataset\star_classification.csv
df.head()
```

Out[32]:

	obj_ID	alpha	delta	u	g	r	i	z	run_ID
0	1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.79371	360
1	1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.61427	451
2	1.237661e+18	142.188790	35.582444	25.26307	22.66389	20.60976	19.34857	18.94827	360
3	1.237663e+18	338.741038	-0.402828	22.13682	23.77656	21.61162	20.50454	19.25010	419
4	1.237680e+18	345.282593	21.183866	19.43718	17.58028	16.49747	15.97711	15.54461	810

Basic info about the dataset

In [33]:

```
df.shape  #check shape
```

Out[33]:

```
(100000, 18)
```

- Dataset has 100000 rows and 18 columns

In [34]:

```
df.columns  #check column names
```

Out[34]:

```
Index(['obj_ID', 'alpha', 'delta', 'u', 'g', 'r', 'i', 'z', 'run_ID',
       'rerun_ID', 'cam_col', 'field_ID', 'spec_obj_ID', 'class', 'redshift',
       'plate', 'MJD', 'fiber_ID'],
      dtype='object')
```

In [35]:

```
df.duplicated().sum()  #check duplicates
```

Out[35]:

```
0
```

- No duplicates present

In [36]:

```
df.isnull().sum()   #check null values
```

Out[36]:

```
obj_ID         0
alpha          0
delta          0
u              0
g              0
r              0
i              0
z              0
run_ID         0
rerun_ID       0
cam_col        0
field_ID       0
spec_obj_ID    0
class          0
redshift       0
plate          0
MJD            0
fiber_ID       0
dtype: int64
```

- No null values present.

In [37]:

```
df.info()    #check info
```

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

- All are numerical column except class

In [38]:

```
df.describe().T.style.background_gradient(cmap='Blues')      #statistical summary
```

Out[38]:

	count	mean	std	
obj_ID	100000.000000	1237664721814903296.000000	8438559894562.676758	12376459
alpha	100000.000000	177.629117	96.502241	
delta	100000.000000	24.135305	19.644665	
u	100000.000000	21.980468	31.769291	
g	100000.000000	20.531387	31.750292	
r	100000.000000	19.645762	1.854760	
i	100000.000000	19.084854	1.757895	
z	100000.000000	18.668810	31.728152	
run_ID	100000.000000	4481.366060	1964.764593	
rerun_ID	100000.000000	301.000000	0.000000	
cam_col	100000.000000	3.511610	1.586912	
field_ID	100000.000000	186.130520	149.011073	
spec_obj_ID	100000.000000	5783882297552056320.000000	3324016169583855104.000000	2995190
redshift	100000.000000	0.576661	0.730707	
plate	100000.000000	5137.009660	2952.303351	
MJD	100000.000000	55588.647500	1808.484233	
fiber_ID	100000.000000	449.312740	272.498404	

Exploratory Data Analysis

In [9]:

```
from dataprep.eda import create_report
report = create_report(df, title='Data Report')
report
```

Data Report | Overview | Variables ≡ | Interactions | Correlations | Missing Values

Overview

Dataset Statistics

Number of Variables	18
Number of Rows	100000
Missing Cells	0
Missing Cells (%)	0.0%
Duplicate Rows	0
Duplicate Rows (%)	0.0%
Total Size in Memory	18.9 MB
Average Row Size in Memory	198.0 B
Variable Types	Numerical: 15 Categorical: 3

Dataset Insights

`u` is skewed	Skewed
`g` is skewed	Skewed
`z` is skewed	Skewed
`redshift` is skewed	Skewed
`rerun_ID` has constant value "301"	Constant
`rerun_ID` has constant length 3	Constant Length
`cam_col` has constant length 1	Constant Length
`delta` has 12061 (12.06%) negatives	Negatives
`redshift` has 13724 (13.72%) negatives	Negatives

Variables

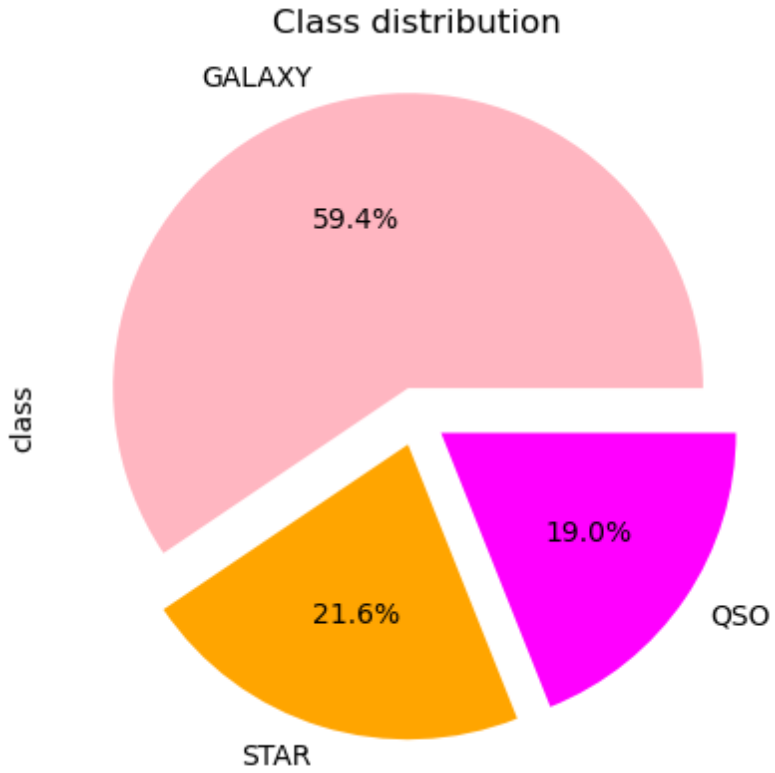
Sort by Feature order ☐ Reverse order

Understanding Class

- Stars - A star is an astronomical object comprising a luminous spheroid of plasma held together by self-gravity. The nearest star to Earth is the Sun.
- Galaxy- A galaxy is a huge collection of gas, dust, and billions of stars and their solar systems, all held together by gravity.
- Quasar - A quasar is an extremely luminous active galactic nucleus. It is sometimes known as a quasi-stellar object, abbreviated QSO. Compact area in the center of a massive galaxy that is around a

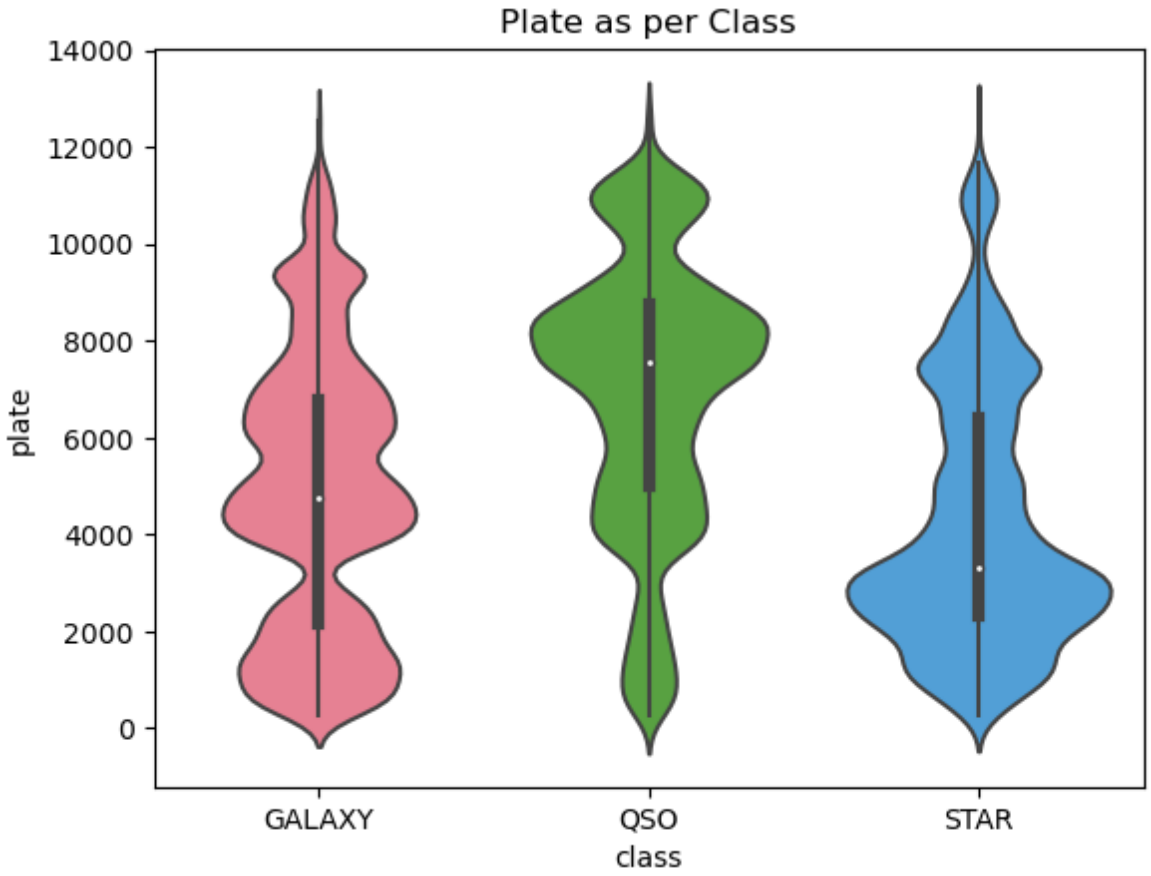
In [10]:

```
df['class'].value_counts().plot(kind='pie',colors=('lightpink','orange','fuchsia'),
                                explode=[0.1,0.1,0.1],autopct='%0.1f%%')
plt.title('Class distribution')
plt.show()
```



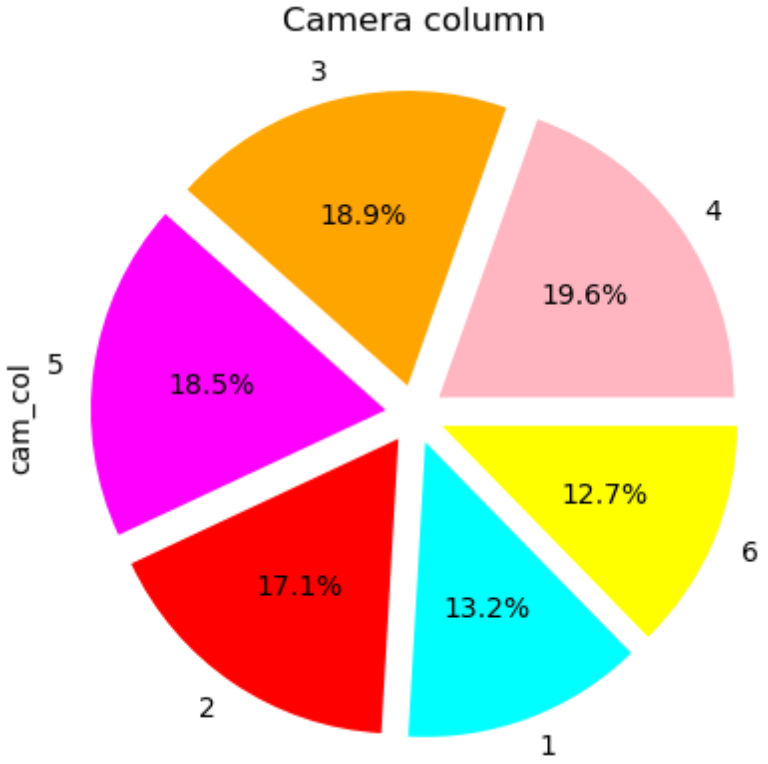
In [14]:

```
sns.violinplot(data=df, x=df["class"], y=df["plate"],palette ="husl")
plt.title('Plate as per Class')
plt.show()
```



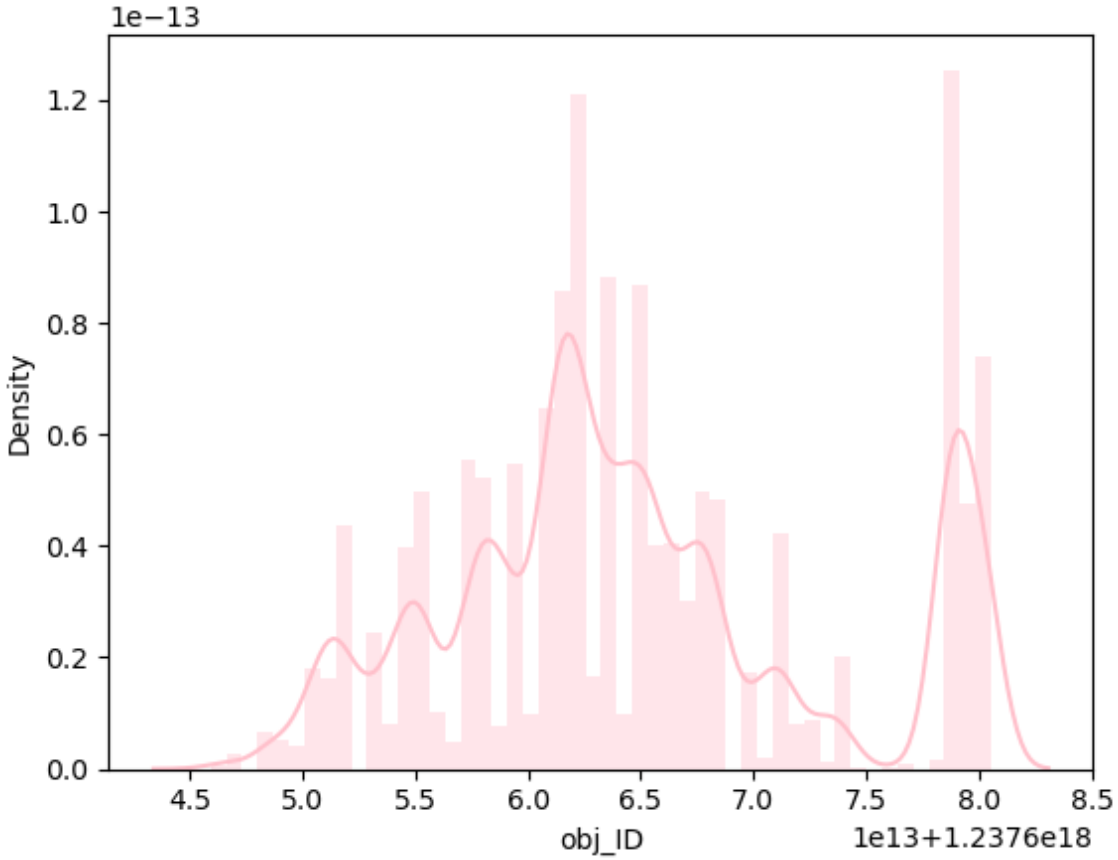
In [140]:

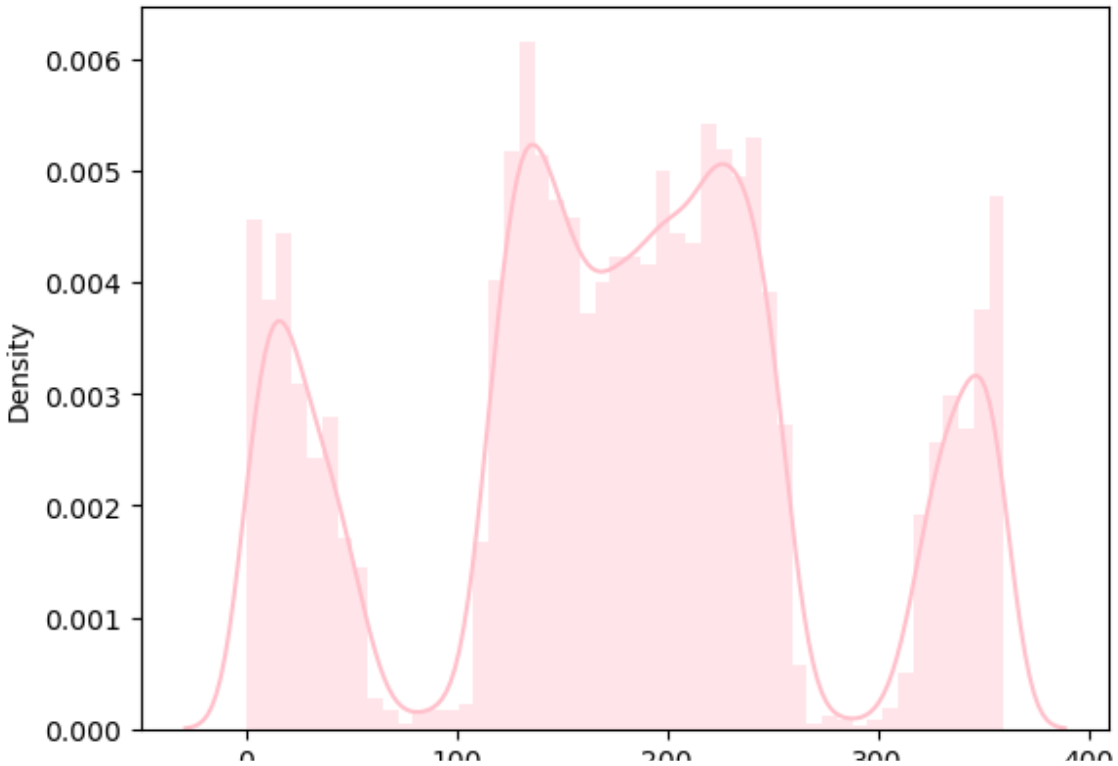
```
df['cam_col'].value_counts().plot(kind='pie',colors=('lightpink','orange',
                                                     'fuchsia','red','aqua','yellow'),
                                  explode=[0.1,0.1,0.1,0.1,0.1,0.1],autopct='%0.1f%%')
plt.title('Camera column')
plt.show()
```



In [19]:

```
def distplots(col):
    sns.distplot(df[col],color='pink')
    plt.show()

for i in list(df.columns)[0:]:
    distplots(i)
```



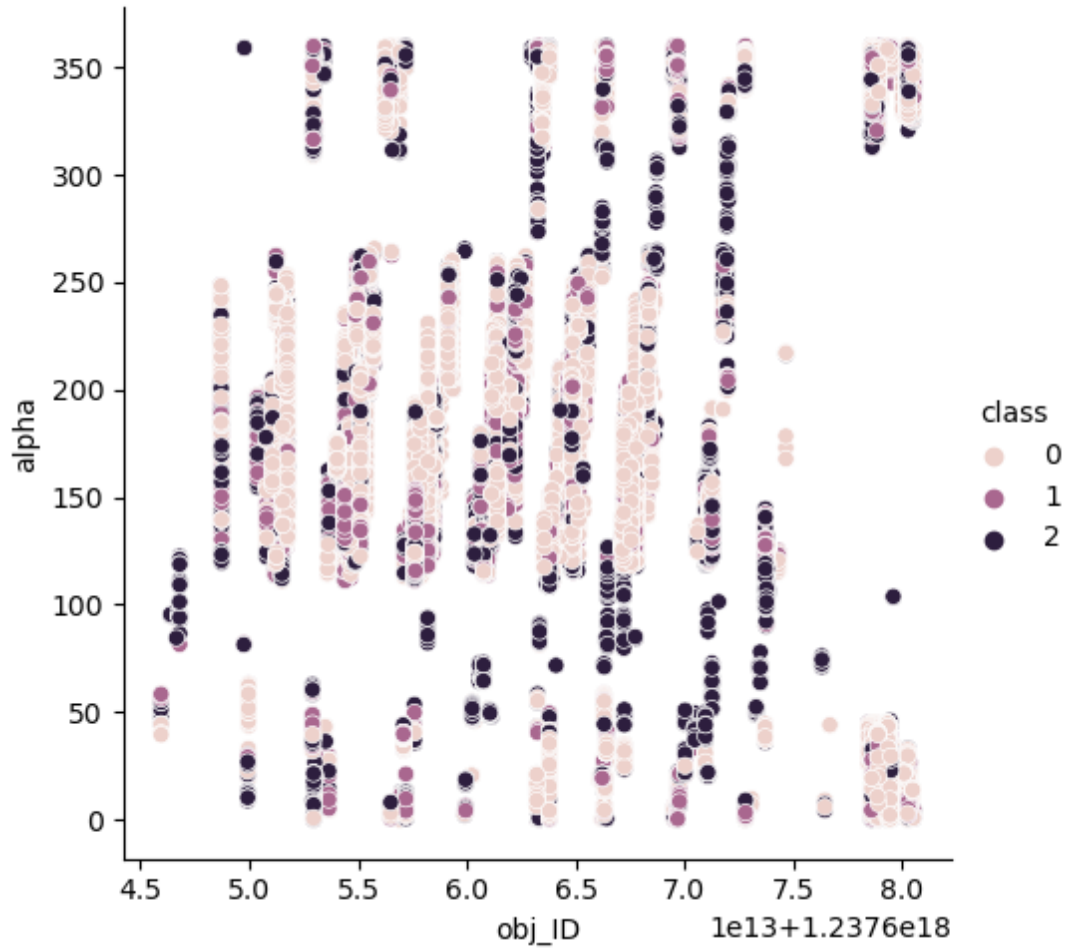


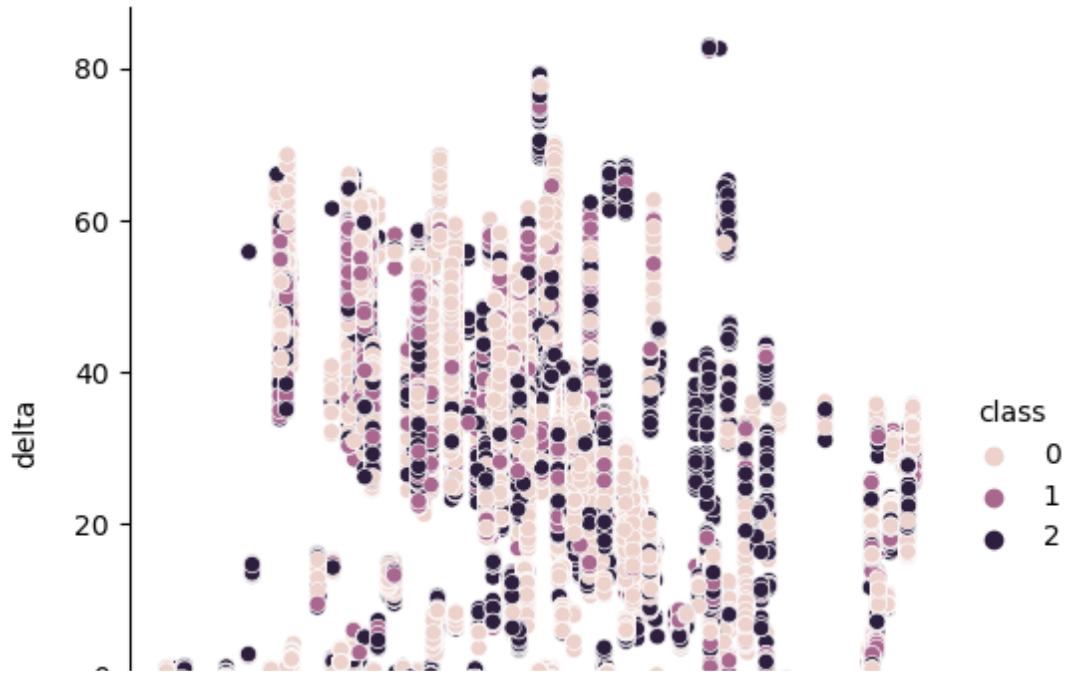
In [136]:

```
df.hist(bins=10, figsize=(16,16),color='pink', edgecolor='purple')
plt.title("Distribution of all variables")
plt.show()
```

In [137]:

```
plots=[]
for i in ['alpha', 'delta', 'u', 'g', 'r', 'i', 'z', 'run_ID','rerun_ID',
          'cam_col', 'field_ID', 'spec_obj_ID', 'redshift','plate', 'MJD', 'fiber_ID']:
    g=sns.relplot(data=df,x='obj_ID', y=i, hue='class')
    plots.append(g)
```



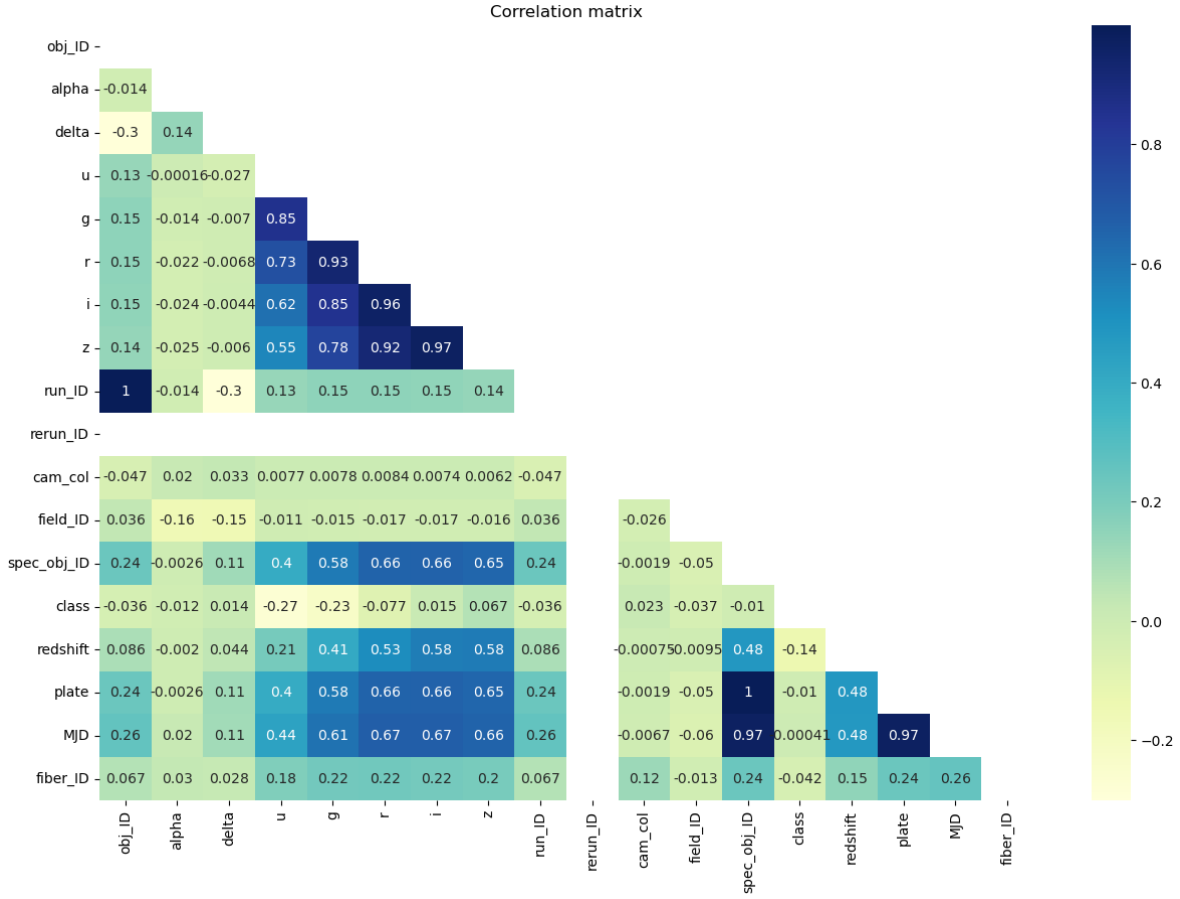


Correlation matrix

- It is a square matrix showing the correlation coefficients between two variables. Correlation coefficients measure how strong and in which direction two variables are linked in a straight line. Positive numbers indicate positive correlations, while negative numbers indicate negative correlations. The closer the number is to 1 (or -1), the stronger the correlation. A number of 0 means there is no correlation between the two variables.

In [139]:

```
mask = np.zeros_like(df.corr(), dtype=float)
mask[np.triu_indices_from(mask)]=True
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu', annot_kws={'size':10}, mask=mask)
plt.title('Correlation matrix')
plt.show()
```



Feature Transformation

1. Segregation of Numerical and Categorical Variables/Columns

In [39]:

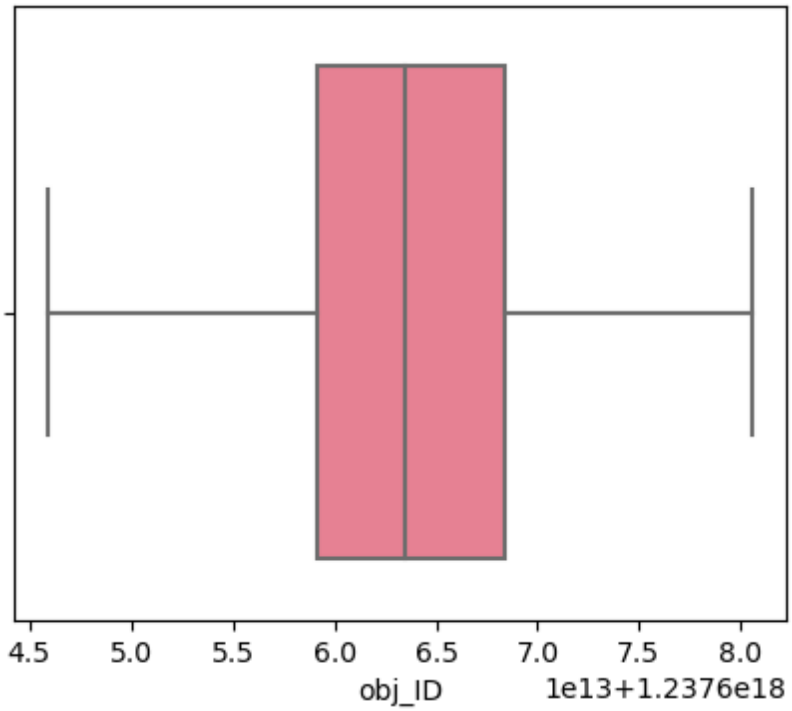
```
categorical_col = df.select_dtypes(include = ['object']).columns
numerical_col = df.select_dtypes(exclude = ['object']).columns
```

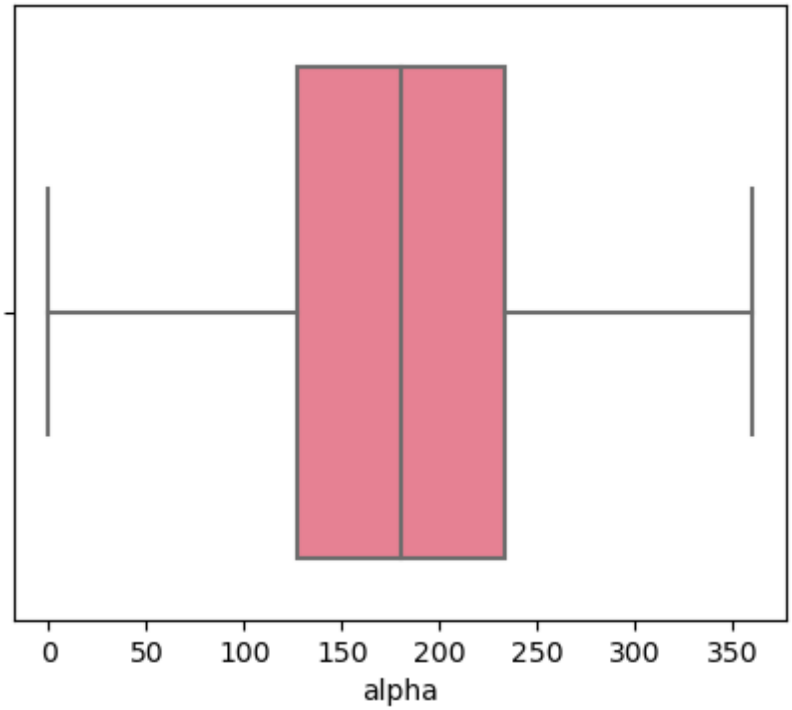
2a. Checking for Outliers

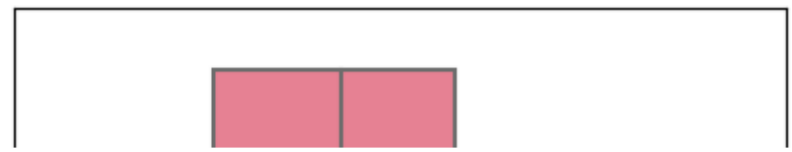
- Outlier is an observation in a given dataset that lies far from the rest of the observations.

In [40]:

```
def boxplots(col):
    plt.figure(figsize=(5,4))
    sns.boxplot(df,x=col,palette='husl')
    plt.show()

for i in list(df.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```







2b.Handling Outliers

In [41]:

```
def outlier(col):
    q3=df[col].quantile(0.75)
    q1=df[col].quantile(0.25)
    IQR=q3-q1
    Lower=q1-1.5*IQR
    Upper=q3+1.5*IQR
    df[col].clip(Lower,Upper,inplace=True)
```

In [42]:

```
for i in numerical_col:
    outlier(i)
```

3. Encoding

- We will use Label encoder for class variable as its target variable to convert it from categorical variable to numerical varaiable

In [43]:

```
df['class']=df['class'].astype('category')
df['class']=df['class'].cat.codes
```

Feature Construction

1. Split the data into Independent (x) & Dependent(y) variables

In [44]:

```
x= df.drop(['class'],axis=1)
y= df[['class']]
```

In [45]:

```
x.head(2)
```

Out[45]:

	obj_ID	alpha	delta	u	g	r	i	z	run_ID
0	1.237661e+18	135.689107	32.494632	23.87882	22.27530	20.39501	19.16573	18.79371	360
1	1.237665e+18	144.826101	31.274185	24.77759	22.83188	22.58444	21.16812	21.61427	451

In [46]:

```
y.head(2)
```

Out[46]:

	class
0	0
1	0

4a. Check target variable data balance (Part of Feature transformation)

In [47]:

```
y.value_counts()    # target variable data is imbalance
```

Out[47]:

```
class
0        59445
2        21594
1        18961
dtype: int64
```

4b. Handle imbalance data (Part of Feature transformation)

In [48]:

```
from imblearn.over_sampling import RandomOverSampler
ros=RandomOverSampler()
x_sam,y_sam=ros.fit_resample(x,y)
print(x_sam.shape,y_sam.shape,x.shape,y.shape)
```

```
(178335, 17) (178335, 1) (100000, 17) (100000, 1)
```

In [49]:

```
y_sam.value_counts()
```

Out[49]:

```
class
0        59445
1        59445
2        59445
dtype: int64
```

5. Feature Scaling (Part of Feature transformation)

- Feature scaling is a method used to normalize the range of independent variables or features of data. We can only do with independent variable not with dependent variable.

In [50]:

```
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler((-1,1))
x = pd.DataFrame(sc.fit_transform(x_sam),columns=x_sam.columns)
y = y_sam
```

In [51]:

```
x
```

Out[51]:

	obj_ID	alpha	delta	u	g	r	i	z	
0	-0.131593	-0.246190	0.007605	0.278692	0.273988	0.138315	0.019019	0.020893	-0.
1	0.094992	-0.195428	-0.016376	0.413436	0.362095	0.514640	0.394813	0.594074	0.
2	-0.131593	-0.210080	0.068278	0.486220	0.335502	0.175227	0.053333	0.052302	-0.
3	0.013970	0.881894	-0.638800	0.017529	0.511639	0.347429	0.270277	0.113638	0.
4	0.985006	0.918236	-0.214641	-0.387204	-0.469239	-0.531606	-0.579399	-0.639375	0.
...	...	...	...	...	...	...	...	...	
178330	-0.226088	0.396388	0.087917	-0.082039	-0.110743	-0.049964	0.009864	0.065509	-0.
178331	-0.325552	-0.300707	0.111379	-0.274270	-0.262246	-0.227902	-0.194705	-0.150501	-0.
178332	-0.108404	0.084897	0.304798	0.334719	0.364508	0.199280	0.122678	0.046323	-0.
178333	-0.051998	0.358048	-0.036206	0.585594	0.329654	0.206697	-0.130770	-0.336961	-0.
178334	0.295688	0.165260	-0.298667	-0.341089	-0.346473	-0.314351	-0.290529	-0.265639	0.

178335 rows × 17 columns

Pricipal Component Analysis

- Principal component analysis is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

In [52]:

```
from sklearn.decomposition import PCA
pca = PCA(0.95)
x_pca = pca.fit_transform(x)
```

In [53]:

```
print(pd.DataFrame(x).shape)  # Shape before PCA
print(x_pca.shape)            # Shape after PCA
```

```
(178335, 17)
(178335, 9)
```

In [54]:

```
# Convert to dataframe
component_names = [f"PC{i+1}" for i in range(x_pca.shape[1])]
x_pca = pd.DataFrame(x_pca, columns=component_names)
x_pca.head()
```

Out[54]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC
0	-0.115512	0.176361	0.747886	-0.283111	0.161904	0.538015	-0.224443	-0.283535	0.08917
1	-1.477867	0.227483	-0.559148	-0.347408	0.427592	0.602889	0.017622	-0.206647	0.00220
2	0.012936	0.206359	0.698125	-0.134717	0.176606	0.174621	-0.155161	-0.521317	0.25697
3	-1.264214	0.064551	0.021554	-0.574818	-0.121789	-0.659895	0.510591	0.044077	-0.69167
4	0.351286	-1.383841	-0.423039	-0.829652	-0.779717	-0.792544	-0.064936	0.936283	-0.08283

PCA Explained variance

In [56]:

```
explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[56]:

```
array([0.331027  , 0.13263991, 0.1112485 , 0.0950974 , 0.08524991,
       0.06998124, 0.06357797, 0.06034522, 0.03093128])
```

PCA Cumulative variance

In [57]:

```
c_variance=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
c_variance
```

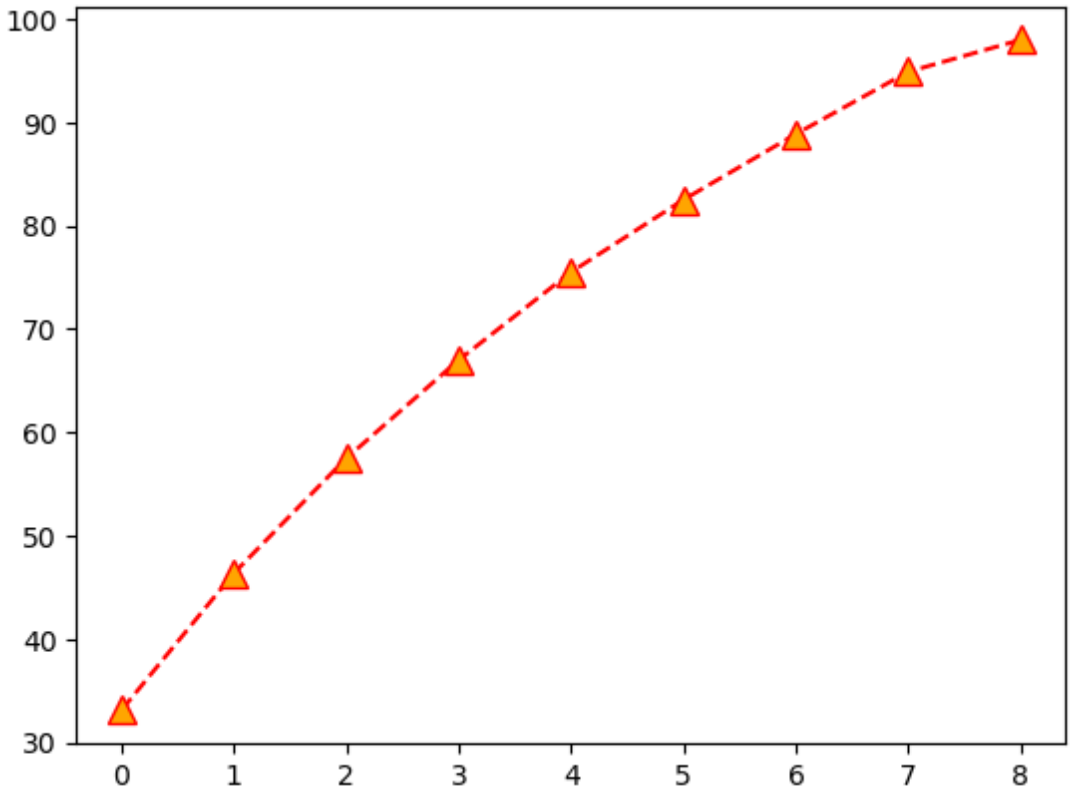
Out[57]:

```
array([33.1 , 46.36, 57.48, 66.99, 75.51, 82.51, 88.87, 94.9 , 97.99])
```

PCA visualization

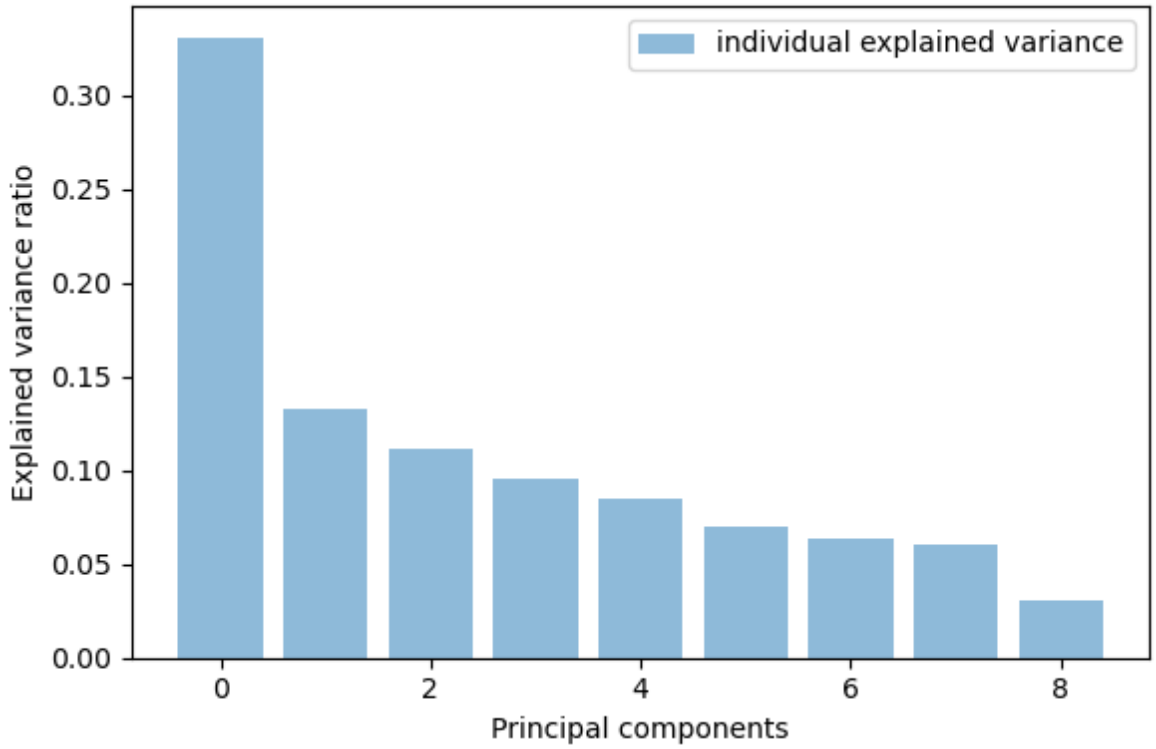
In [58]:

```
plt.plot(c_variance,color='red', linestyle='dashed', marker='^',
         markerfacecolor='orange', markersize=10)
plt.show()
```



In [132]:

```
plt.figure(figsize=(6,4))
plt.bar(range(9), explained_variance, alpha=0.5, align='center',
            label='individual explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
```



2. Split the data into train and test- Feature Construction

In [60]:

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x_pca,y,test_size=0.20,random_state=101,stra
```

Model Building

Model No. 1 - Logistic Regression

- It is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class.

In [61]:

```
# Model building
from sklearn.linear_model import LogisticRegression
logit = LogisticRegression(random_state=100)
log=logit.fit(x_train, y_train)
# Predict
y_pred_train_log = logit.predict(x_train)
y_pred_test_log = logit.predict(x_test)
# Evaluate
from sklearn.metrics import confusion_matrix,classification_report, accuracy_score
accuracy_log_test=accuracy_score(y_test,y_pred_test_log)
accuracy_log_train=accuracy_score(y_train,y_pred_train_log)
print('Logistic regression Train accuracy:', accuracy_score(y_train, y_pred_train_log))
print('------'*10)
print('Logistic regression Test accuracy:', accuracy_score(y_test, y_pred_test_log))
```

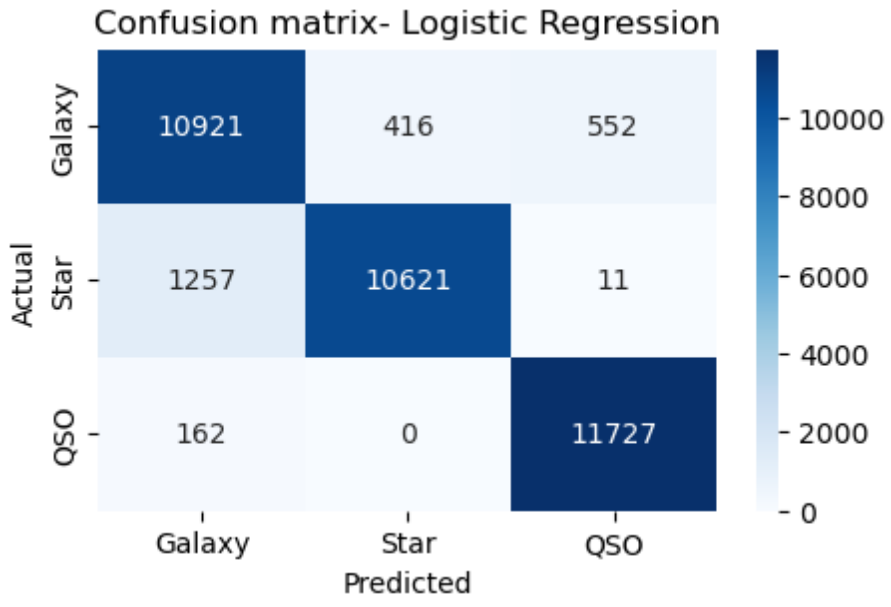
```
Logistic regression Train accuracy: 0.9314983037541705
------------------------------------------------------------
Logistic regression Test accuracy: 0.9327669834861356
```

Confusion Matrix

- A confusion matrix presents a table layout of the different outcomes of the prediction and results of a classification problem and helps visualize its outcomes. It plots a table of all the predicted and actual values of a classifier.

In [114]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_log),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Logistic Regression")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



2. Model No. 2 - Decision Tree

- A decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

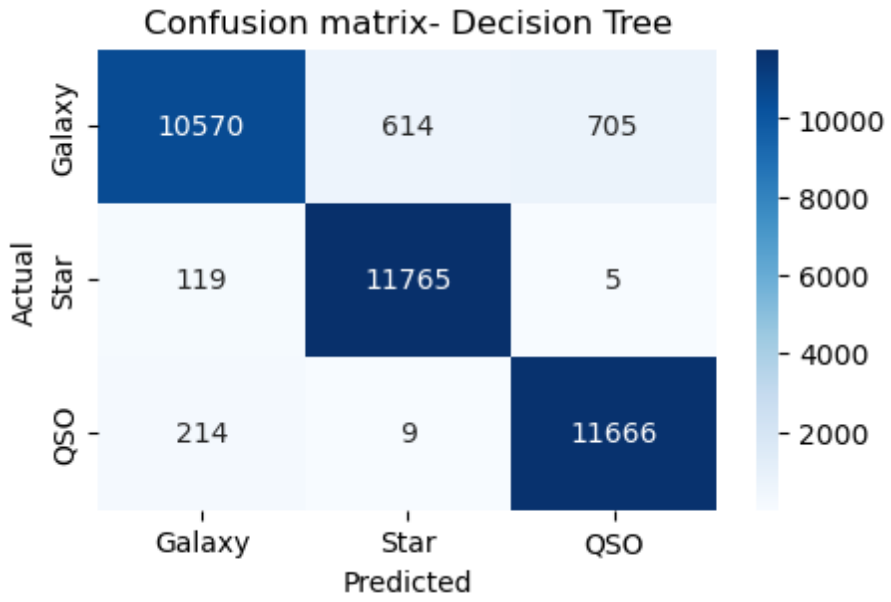
In [62]:

```
# Model building
from sklearn.tree import DecisionTreeClassifier,plot_tree
dtree= DecisionTreeClassifier()
dtree.fit(x_train,y_train)
#Predict
y_pred_train_dtree=dtree.predict(x_train)
y_pred_test_dtree=dtree.predict(x_test)
#Evaluate
accuracy_dtree_test=accuracy_score(y_test,y_pred_test_dtree)
accuracy_dtree_train=accuracy_score(y_train,y_pred_train_dtree)
print('Decision Tree - Train accuracy:', accuracy_score(y_train, y_pred_train_dtree))
print('-------'*10)
print('Decision Tree - Test accuracy:', accuracy_score(y_test, y_pred_test_dtree))
```

```
Decision Tree - Train accuracy: 1.0
----------------------------------------------------------------------
Decision Tree - Test accuracy: 0.9532901561667648
```

In [115]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_dtree),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Decision Tree")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Decision Tree Visualization

In [63]:

```
# Using Post prunning method to handle overfitting probelm
def dtree_model(model):
    model_preds=model.predict(x_test)
    print(classification_report(y_test,model_preds))
    print('\n')
    plt.figure(figsize=(15,12),dpi=150)
    plot_tree(model,filled=True,feature_names=x.columns)
plt.show()
```

In [64]:

```
# max depth at 5
prunned_dtree=DecisionTreeClassifier(max_depth=5)
prunned_dtree.fit(x_train,y_train)
```

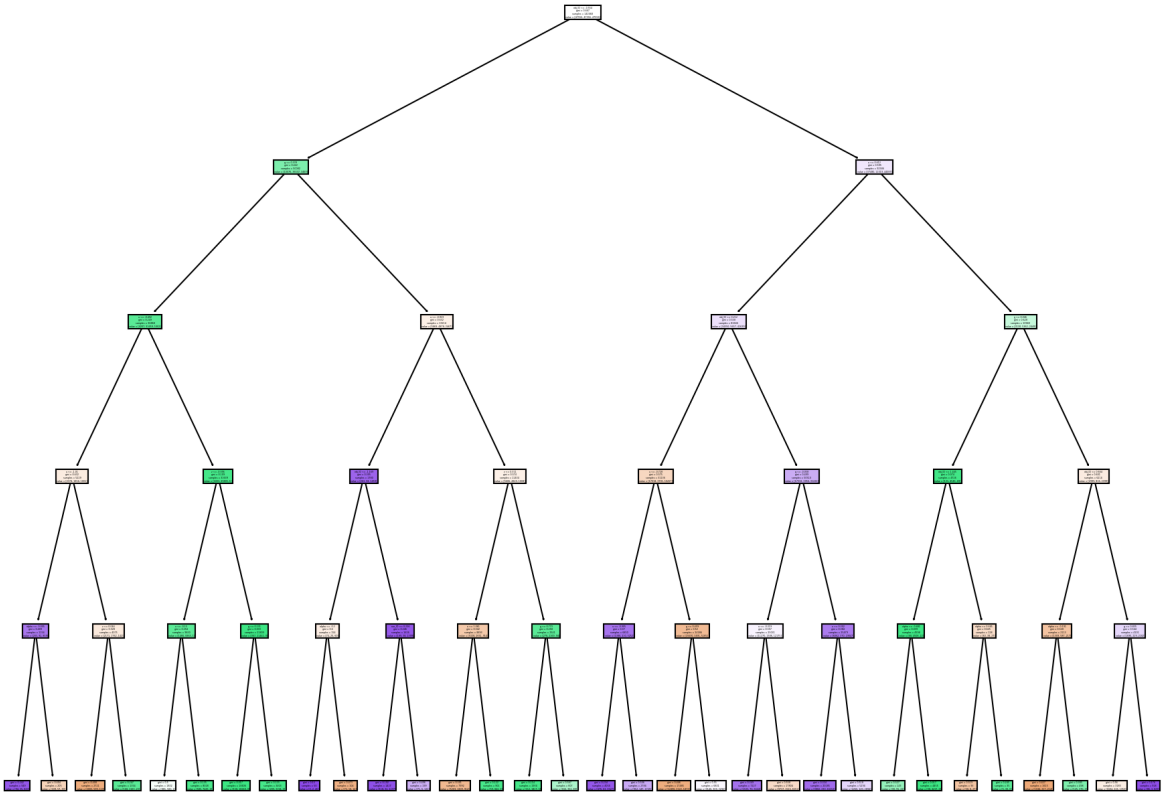
Out[64]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5)
```

In [65]:

```
dtree_model(prunned_dtree)
```

```
              precision    recall  f1-score   support

           0       0.63      0.68      0.65     11889
           1       0.94      0.82      0.87     11889
           2       0.72      0.75      0.74     11889

    accuracy                           0.75     35667
   macro avg       0.76      0.75      0.76     35667
weighted avg       0.76      0.75      0.76     35667
```



In [66]:

```
# Predict
y_pred_prunned_train=prunned_dtree.predict(x_train)
y_pred_prunned_test=prunned_dtree.predict(x_test)
# Evaluate
print('Decision Tree post prunning- Train accuracy:',accuracy_score(y_train,y_pred_prunned_
print('-------'*10)
print('Decision Tree post prunning- Test accuracy:', accuracy_score(y_test,y_pred_prunned_t
```

```
Decision Tree post prunning- Train accuracy: 0.7550536910869992
----------------------------------------------------------------------
Decision Tree post prunning- Test accuracy: 0.7511144755656489
```

Model No. 3 - K Nearest Neighbors (KNN)

- The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point.

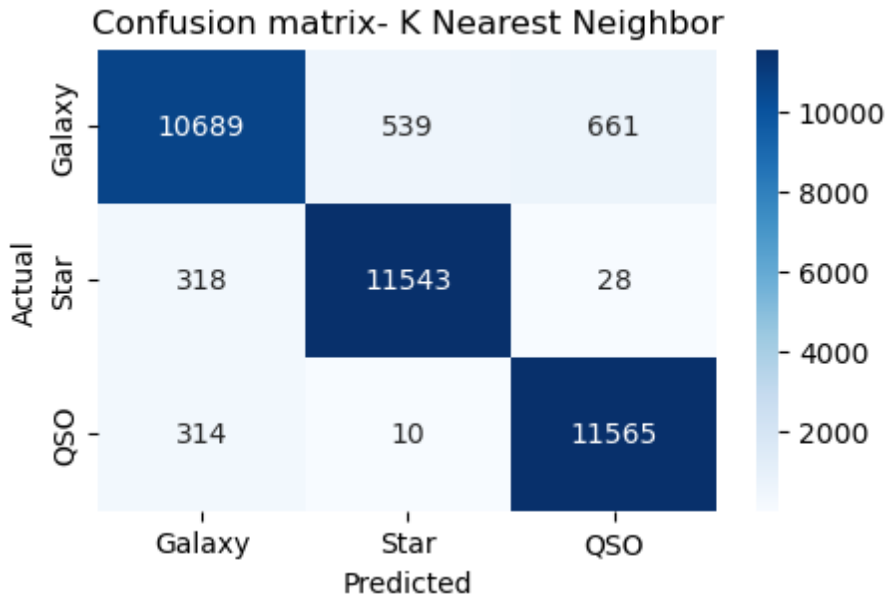
In [133]:

```
# Model building with K point as 3
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)
# Predict
y_pred_train_knn = knn.predict(x_train)
y_pred_test_knn = knn.predict(x_test)
#Evaluate
accuracy_knn_test=accuracy_score(y_test,y_pred_test_knn)
accuracy_knn_train=accuracy_score(y_train,y_pred_train_knn)
print('K nearest neighbor - Train accuracy:', accuracy_score(y_train, y_pred_train_knn))
print('-------'*10)
print('K nearest neighbor - Test accuracy:', accuracy_score(y_test, y_pred_test_knn))
```

```
K nearest neighbor - Train accuracy: 0.974310987747778
----------------------------------------------------------------------
K nearest neighbor - Test accuracy: 0.9475705834524911
```

In [116]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_knn),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- K Nearest Neighbor")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



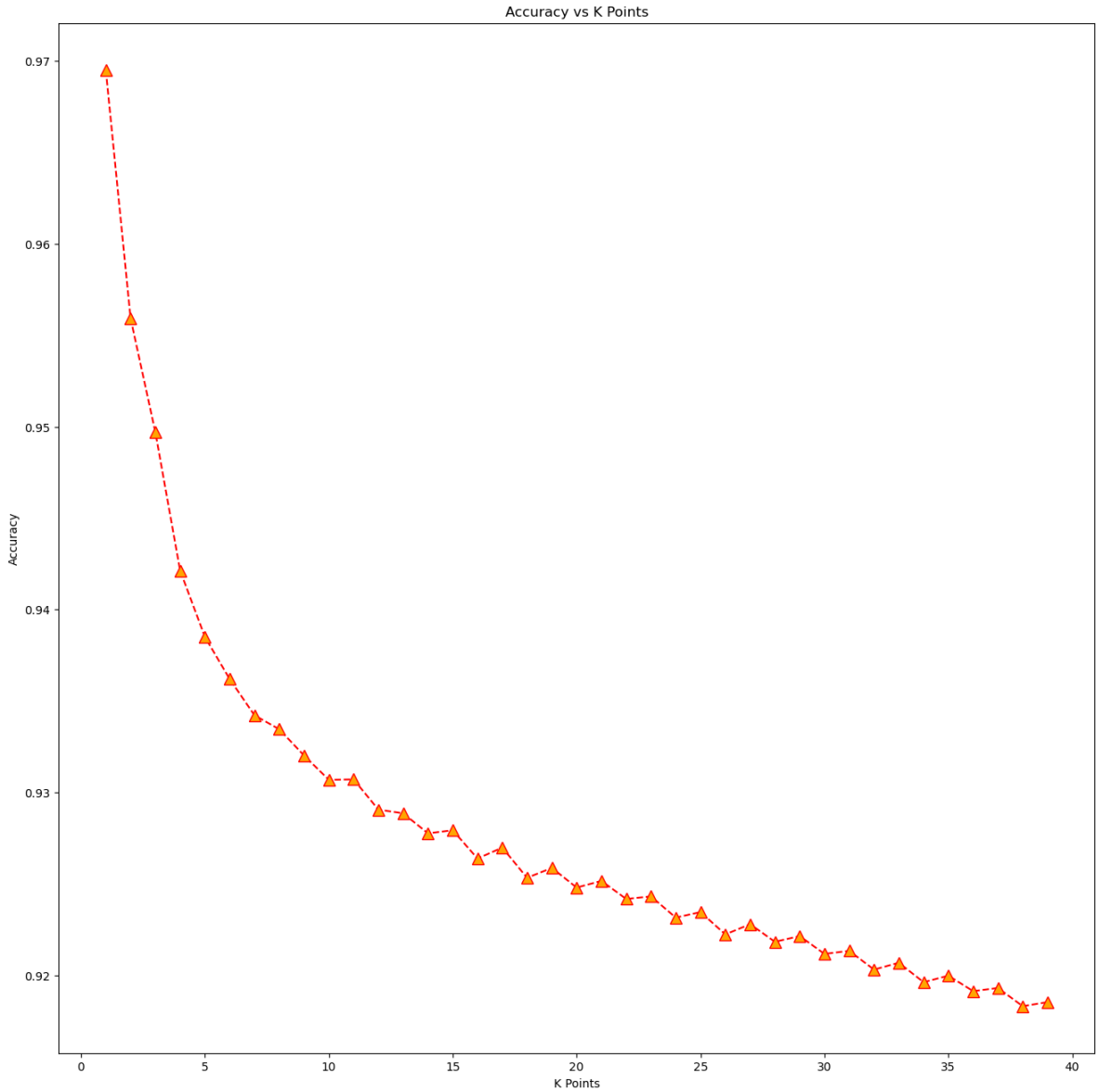
In [134]:

```
accuracy=[]
for i in range(1,40):
    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,x,y,cv=10)
    accuracy.append(score.mean())
```

Plotting the graph

In [135]:

```
plt.figure(figsize=(16,16))
plt.plot(range(1,40), accuracy, color='red', linestyle='dashed', marker='^',
         markerfacecolor='orange', markersize=10)
plt.title('Accuracy vs K Points')
plt.xlabel('K Points')
plt.ylabel('Accuracy')
plt.show()
```



Model No. 4 - Bagging model

- Bagging (or Bootstrap aggregating) is a type of ensemble learning in which multiple base models are trained independently in parallel on different subsets of the training data.

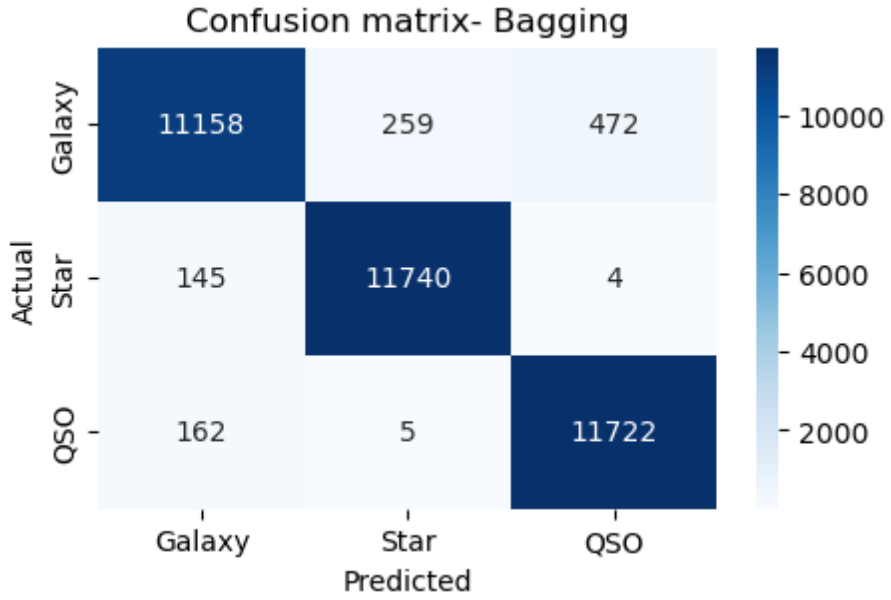
In [68]:

```
# Model building
from sklearn.ensemble import BaggingClassifier
bagging=BaggingClassifier()
bagging.fit(x_train,y_train)# Predict
#Predict
y_pred_train_bag=bagging.predict(x_train)
y_pred_test_bag=bagging.predict(x_test)
# Evaluate
accuracy_bag_test=accuracy_score(y_test,y_pred_test_bag)
accuracy_bag_train=accuracy_score(y_train,y_pred_train_bag)
print('Bagging - Train accuracy:', accuracy_score(y_train, y_pred_train_bag))
print('-------'*10)
print('Bagging - Test accuracy:', accuracy_score(y_test, y_pred_test_bag))
```

```
Bagging - Train accuracy: 0.9983808562536799
----------------------------------------------------------------------
Bagging - Test accuracy: 0.9706451341576247
```

In [117]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_bag),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Bagging")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 5 - Random Forest

- A random forest is a machine learning technique that utilizes ensemble learning - which is a technique that combines many classifiers to provide solutions to complex problems. It establishes the outcome based on the predictions of the decision trees and employs the bagging method to generate the required prediction. It eradicates the biggest limitation of decision tree - overfitting of dataset and increases precision. The main

difference between the decision tree algorithm and the random forest algorithm is that establishing root nodes and segregating nodes is done randomly in the latter

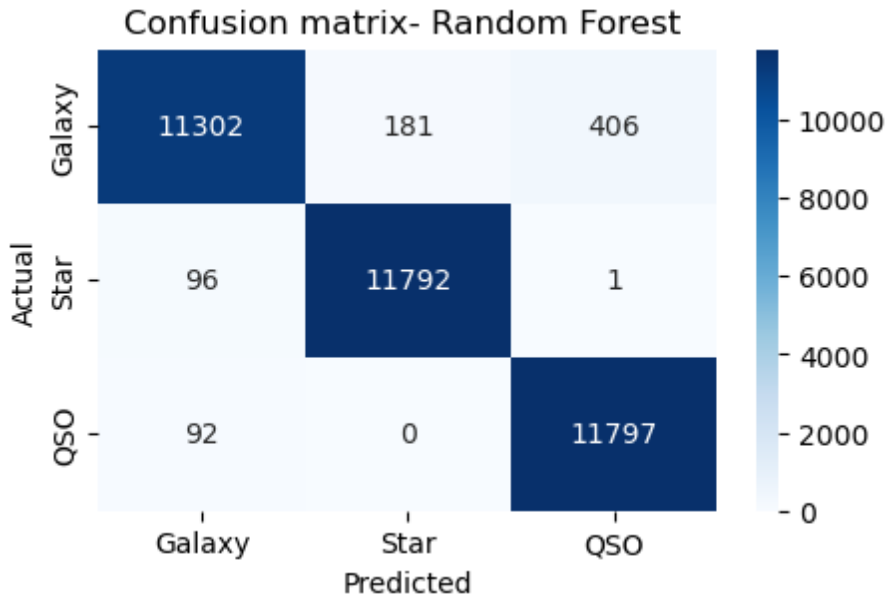
In [69]:

```
# Model building
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=200,oob_score=False)
rf.fit(x_train,y_train)
# Predict
y_pred_train_rf=rf.predict(x_train)
y_pred_test_rf=rf.predict(x_test)
# Evaluate
accuracy_rf_test=accuracy_score(y_test,y_pred_test_rf)
accuracy_rf_train=accuracy_score(y_train,y_pred_train_rf)
print('Random Forest - Train accuracy:', accuracy_score(y_train, y_pred_train_rf))
print('-------'*10)
print('Random Forest - Test accuracy:', accuracy_score(y_test, y_pred_test_rf))
```

```
Random Forest - Train accuracy: 1.0
----------------------------------------------------------------------
Random Forest - Test accuracy: 0.9782431939888412
```

In [118]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_rf),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Random Forest ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 6 - Naives Bayes

- The Naive Bayes algorithm is comprised of Naive and Bayes. It is called Naive because it assumes that the occurrence of a certain feature is independent of the occurrence of other features and it is called Bayes

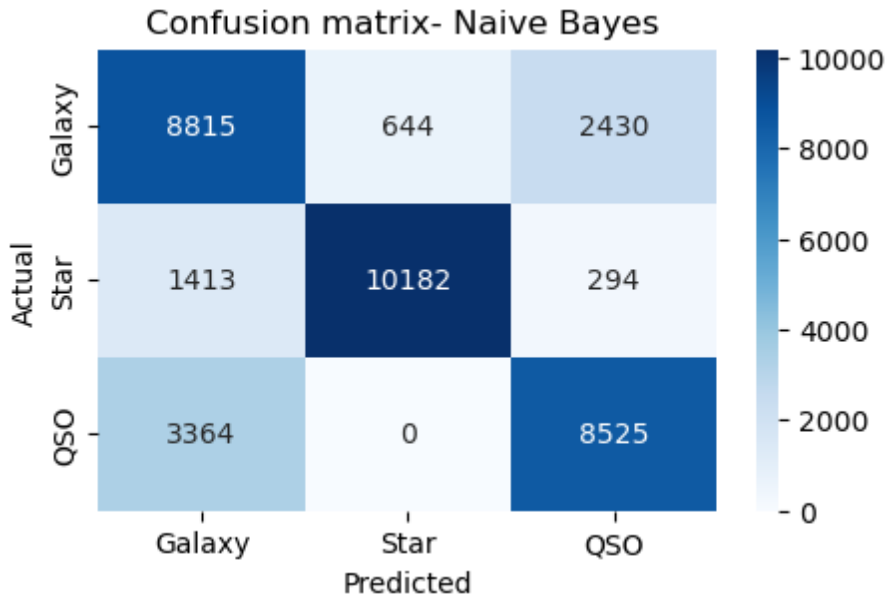
In [70]:

```
# Model building
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)
# Predict the model
y_pred_train_nb = nb.predict(x_train)
y_pred_test_nb = nb.predict(x_test)
# Evaluate
accuracy_nb_test=accuracy_score(y_test,y_pred_test_nb)
accuracy_nb_train=accuracy_score(y_train,y_pred_train_nb)
print('Naive Bayes -Train accuracy:', accuracy_score(y_train, y_pred_train_nb))
print('-------'*10)
print('Naive Bayes -Test accuracy:', accuracy_score(y_test, y_pred_test_nb))
```

```
Naive Bayes -Train accuracy: 0.7696540219250287
----------------------------------------------------------------------
Naive Bayes -Test accuracy: 0.7716376482462781
```

In [119]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_nb),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Naive Bayes ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 7 - Support Vector Machine Model

- Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression.They are extremely popular because of their ability to handle multiple continuous and categorical variables.The objective of the support vector machine algorithm is to find a maximum marginal hyperplane.

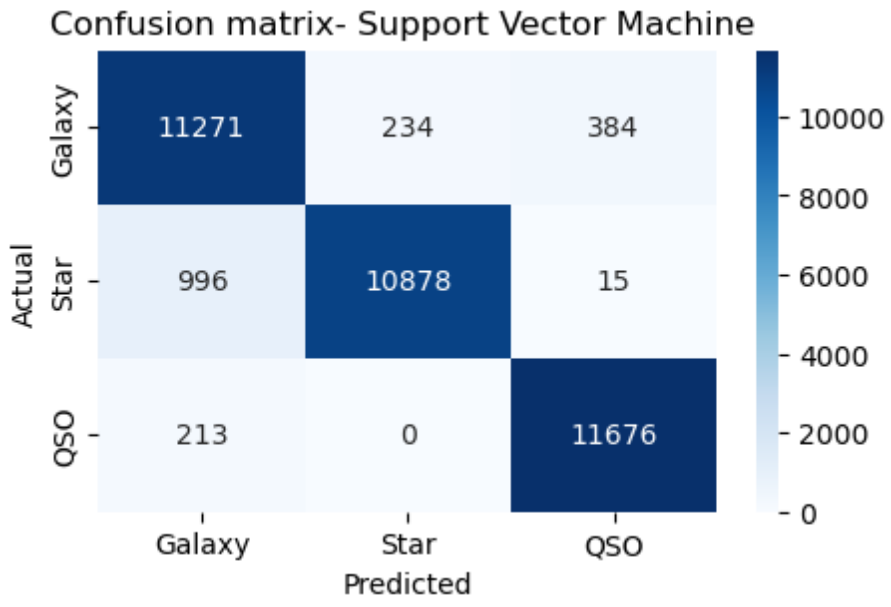
In [71]:

```
# Radial Basis Function Kernel (RBF) - (Defaut SVM) Model building
from sklearn.svm import SVC
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(x_train, y_train)
#Predict
y_pred_train_rbf = svm_rbf.predict(x_train)
y_pred_test_rbf = svm_rbf.predict(x_test)
#Evaluate
accuracy_rbf_test=accuracy_score(y_test,y_pred_test_rbf)
accuracy_rbf_train=accuracy_score(y_train,y_pred_train_rbf)
print('Rbf - SVM - Train accuracy:', accuracy_score(y_train, y_pred_train_rbf))
print('-------'*10)
print('Rbf - SVM - Test accuracy:', accuracy_score(y_test, y_pred_test_rbf))
```

```
Rbf - SVM - Train accuracy: 0.9473603050438781
----------------------------------------------------------------------
Rbf - SVM - Test accuracy: 0.9483556228446464
```

In [120]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_test_rbf),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Support Vector Machine ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 8 - Adaptive boosting or AdaBoost

- Yoav Freund and Robert Schapire are credited with the creation of the AdaBoost algorithm. This method operates iteratively, identifying misclassified data points and adjusting their weights to minimize the training error. The model continues optimize in a sequential fashion until it yields the strongest predictor.

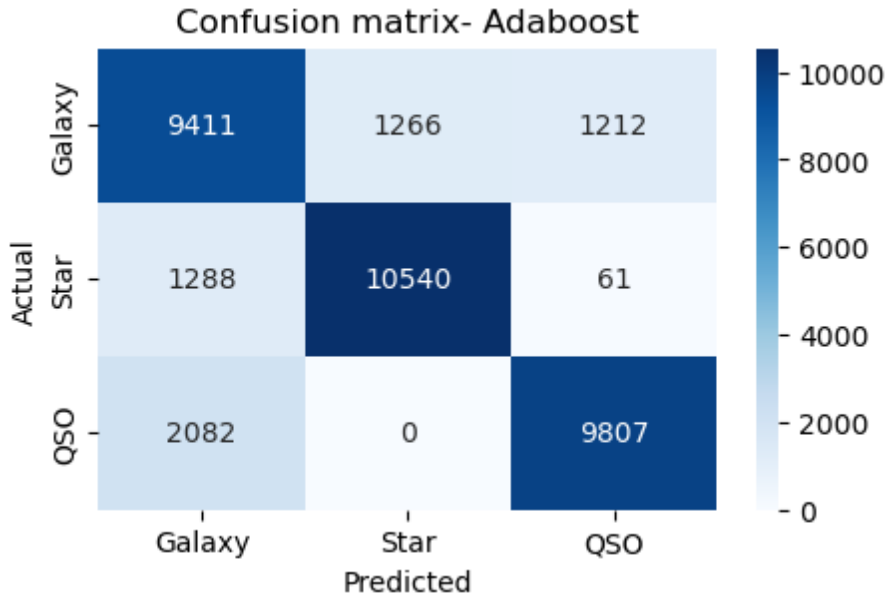
In [72]:

```
# Model building
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
ad=ada.fit(x_train, y_train)
# Predict
y_pred_ad = ada.predict(x_test)
y_pred_ad_train = ada.predict(x_train)
# Evaluate
from sklearn.metrics import confusion_matrix,classification_report, accuracy_score
accuracy_ad_test=accuracy_score(y_test,y_pred_ad)
accuracy_ad_train=accuracy_score(y_train,y_pred_ad_train)
print('AdaBoost Train accuracy:', accuracy_score(y_train, y_pred_ad_train))
print('-------'*5)
print('AdaBoost Test accuracy:', accuracy_score(y_test, y_pred_ad))
```

```
AdaBoost Train accuracy: 0.8314548462163905
-----------------------------------
AdaBoost Test accuracy: 0.8343286511341016
```

In [121]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_ad),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Adaboost")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 9 - Gradient Boosting

- Jerome H. Friedman developed gradient boosting, which works by sequentially adding predictors to an ensemble with each one. However, instead of changing weights of data points like AdaBoost, the gradient boosting trains on the residual errors of the previous predictor. The name, gradient boosting, is used since it combines the gradient descent algorithm and boosting method.

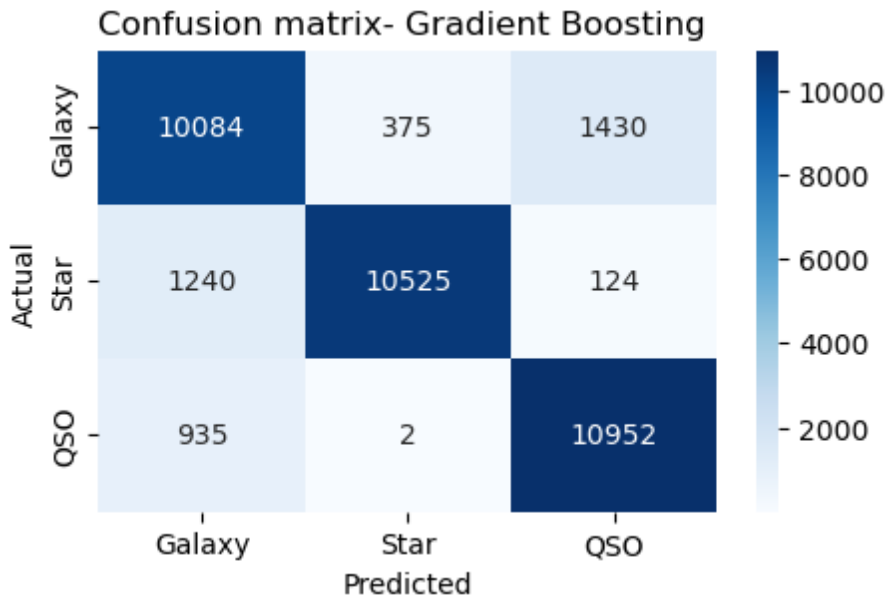
In [73]:

```
# Model building
from sklearn.ensemble import GradientBoostingClassifier
gdb = GradientBoostingClassifier()
gd=gdb.fit(x_train, y_train)
# Predict
y_pred_gd = gdb.predict(x_test)
y_pred_gd_train = gdb.predict(x_train)
# Evaluate
accuracy_gd_test=accuracy_score(y_test,y_pred_gd)
accuracy_gd_train=accuracy_score(y_train,y_pred_gd_train)
print('GradientBoosting Train accuracy:', accuracy_score(y_train, y_pred_gd_train))
print('-------'*5)
print('GradientBoosting Test accuracy:', accuracy_score(y_test, y_pred_gd))
```

```
GradientBoosting Train accuracy: 0.8886295455182662
-----------------------------------
GradientBoosting Test accuracy: 0.8848795805646676
```

In [122]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_gd),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Gradient Boosting ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 10 - Extreme Gradient Boosting or XGBoost

- XGBoost is an implementation of gradient boosting that's designed for computational speed and scale. It leverages multiple cores on the CPU, allowing for learning to occur in parallel during training

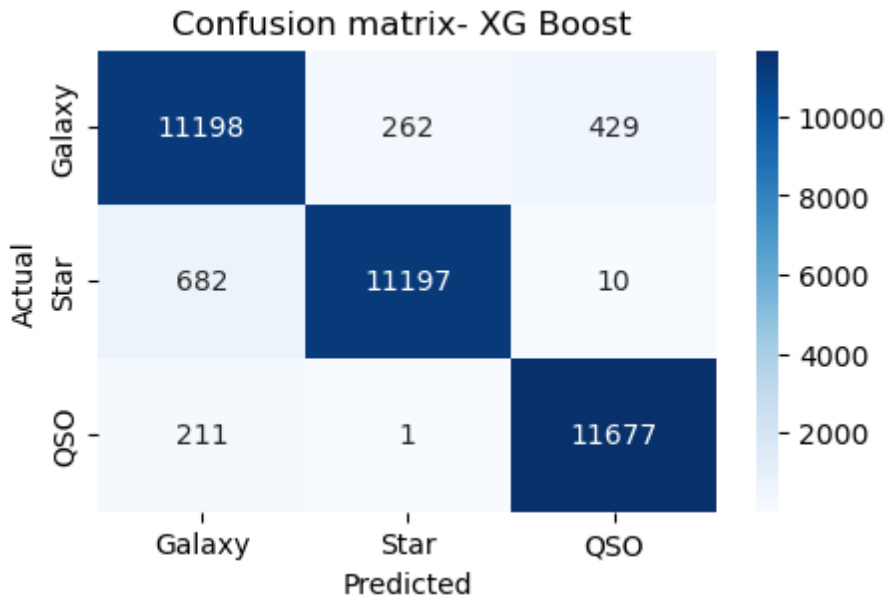
In [74]:

```
# Model building
from xgboost import XGBClassifier
xgb = XGBClassifier()
xg=xgb.fit(x_train, y_train)
# Predict
y_pred_xg = xgb.predict(x_test)
y_pred_xg_train = xgb.predict(x_train)
# Evaluate
accuracy_xg_test=accuracy_score(y_test,y_pred_xg)
accuracy_xg_train=accuracy_score(y_train,y_pred_xg_train)
print('XGBoost Train accuracy:', accuracy_score(y_train, y_pred_xg_train))
print('-------'*5)
print('XGBoost Test accuracy:', accuracy_score(y_test, y_pred_xg))
```

```
XGBoost Train accuracy: 0.9692152409790563
-----------------------------------
XGBoost Test accuracy: 0.9552807917683013
```

In [123]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,y_pred_xg),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- XG Boost ")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Model No. 11 Voting ensemble

- Voting is an ensemble method that combines the performances of multiple models to make predictions.

In [92]:

```
from sklearn.ensemble import VotingClassifier
```

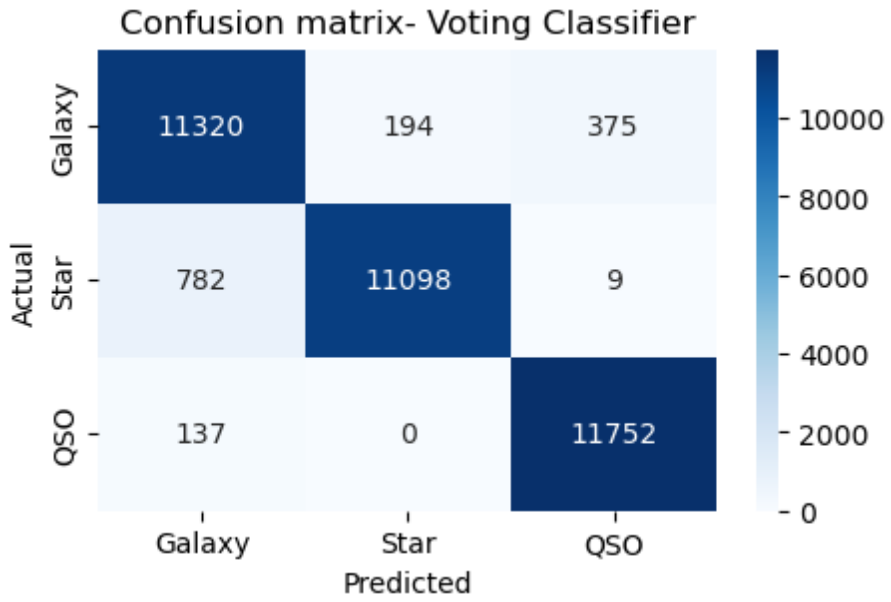
In [93]:

```
evc=VotingClassifier(estimators = [('Logistic',logit),('Decision_tree', dtree),('KNN',knn),
                                   ('Randomforest',rf),('NaiveBayes',nb), ('SVM_RBF' ,svm_rb
                                   ('Gradient_boosting',gd),('Extra_Gradient_booting',xg)],v
evc_model=evc.fit(x_train,y_train)
evc_pred=evc.predict(x_test)
evc_pred_train=evc.predict(x_train)

accuracy_evc=accuracy_score(y_test,evc_pred)
accuracy_evc_train=accuracy_score(y_train,evc_pred_train)

print('Voting ensemble train accuracy:', accuracy_score(y_train, evc_pred_train))
print('---------------'*5)
print('Voting ensemble train accuracy:', accuracy_score(y_test, evc_pred))
```

```
Voting ensemble train accuracy: 0.9713180250651863
--------------------------------------------------------------------------------
Voting ensemble train accuracy: 0.9580284296408444
```

In [124]:

```
Labels = ['Galaxy','Star','QSO']
plt.figure(figsize=(5,3))
sns.heatmap(confusion_matrix(y_test,evc_pred),xticklabels=Labels,
            yticklabels=Labels,cmap='Blues',annot=True, fmt='g')
plt.title("Confusion matrix- Voting Classifier")
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```



Combining all models in Tabular format for better understanding

In [125]:

```
Models=['Logistic','Decision_tree','KNN','Bagging','Random_forest','Naive_bayes','SVM',
        'Adaboost','GradientBoosting','XGboost','Voting_Classifier']
Trainacc=[accuracy_log_train,accuracy_dtree_train,accuracy_knn_train,accuracy_bag_train,acc
          accuracy_rbf_train,accuracy_ad_train,accuracy_gd_train, accuracy_xg_train,accuracy
Testacc=[accuracy_log_test,accuracy_dtree_test,accuracy_knn_test,accuracy_bag_test,accuracy_
         accuracy_rbf_test, accuracy_ad_test,accuracy_gd_test, accuracy_xg_test,accuracy_ev
```

In [126]:

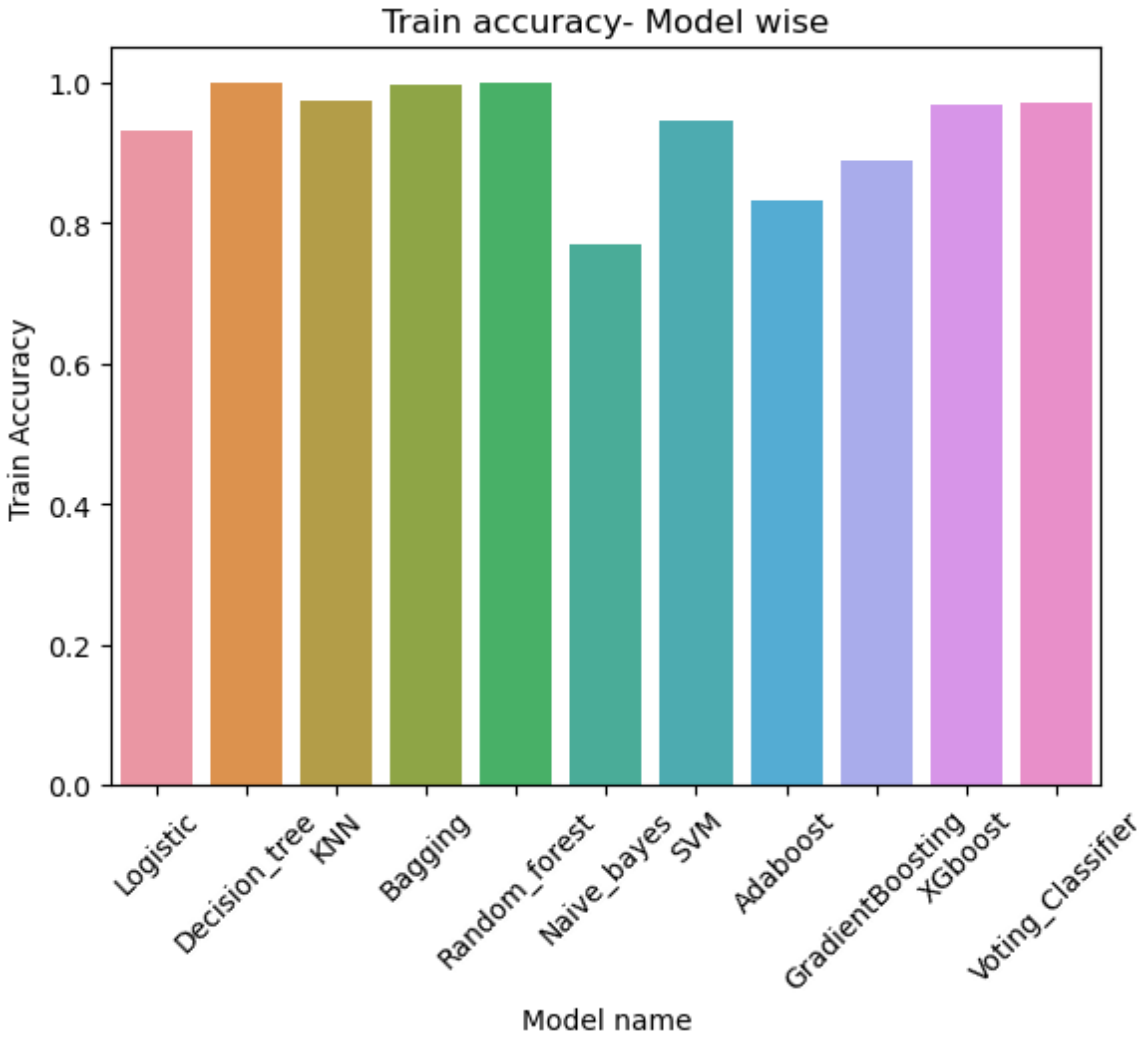
```
Combined_accuracy=pd.DataFrame({'Model name':Models,'Train Accuracy':Trainacc,
                                'Test Accuracy':Testacc})
print(Combined_accuracy)
```

```
           Model name  Train Accuracy  Test Accuracy
0            Logistic        0.931498       0.932767
1       Decision_tree        1.000000       0.953290
2                 KNN        0.974311       0.947571
3             Bagging        0.998381       0.970645
4       Random_forest        1.000000       0.978243
5         Naive_bayes        0.769654       0.771638
6                 SVM        0.947360       0.948356
7            Adaboost        0.831455       0.834329
8    GradientBoosting        0.888630       0.884880
9             XGboost        0.969215       0.955281
10  Voting_Classifier        0.971318       0.958028
```

Accuracy visualization

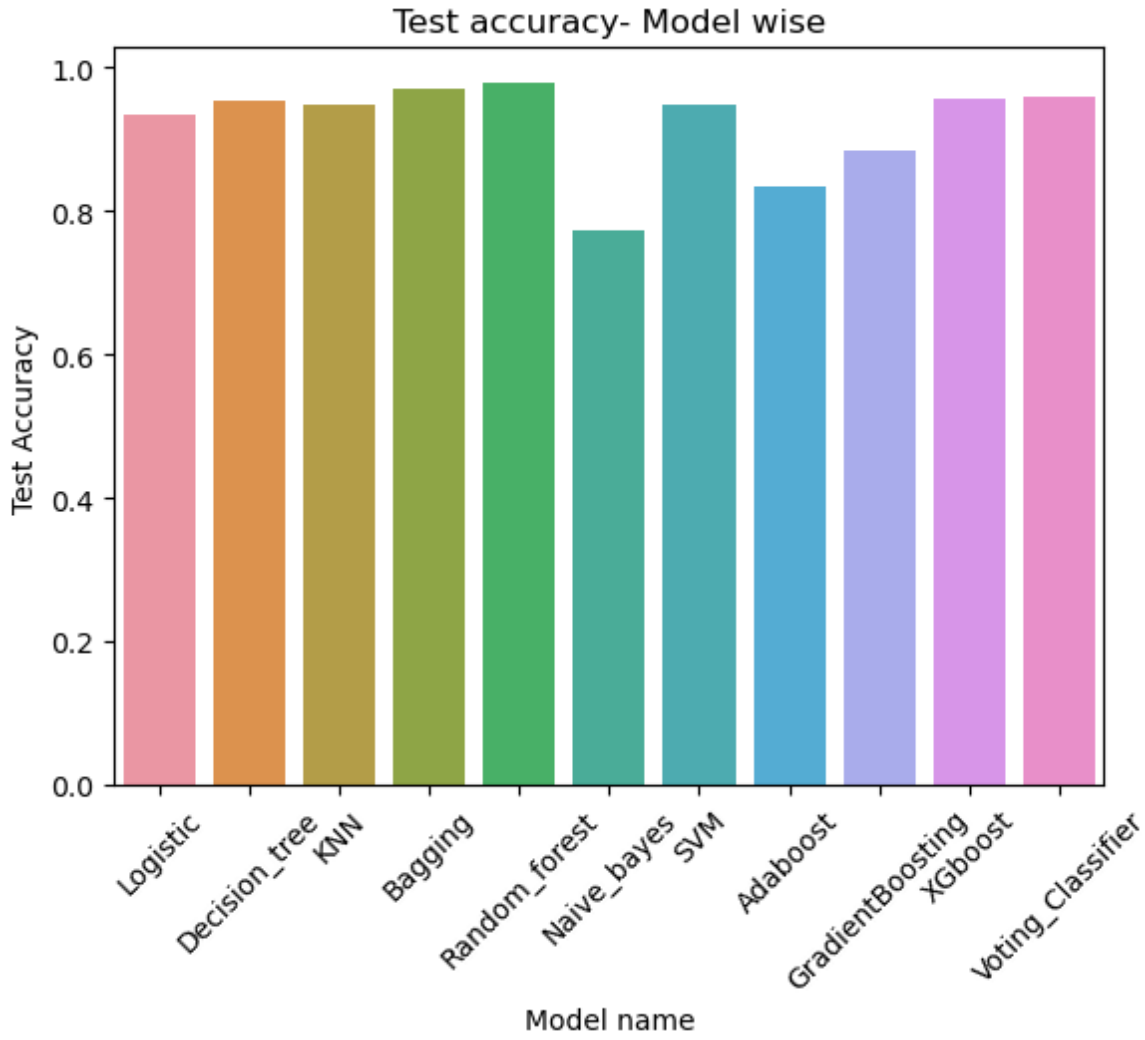
In [127]:

```
sns.barplot(x='Model name',y='Train Accuracy',data=Combined_accuracy)
plt.xticks(rotation=45)
plt.title('Train accuracy- Model wise')
plt.show()
```



In [128]:

```
sns.barplot(x='Model name',y='Test Accuracy',data=Combined_accuracy)
plt.xticks(rotation=45)
plt.title('Test accuracy- Model wise')
plt.show()
```



Cross validation of highest accuracy model- Random Forest Classifier

In [113]:

```
train_accuracy_rf = cross_val_score(rf,x_train, y_train, cv=10)
crossval_train_rf=train_accuracy_rf.mean()
test_accuracy_rf = cross_val_score(rf,x_test, y_test, cv=10)
crossval_test_rf=test_accuracy_rf.mean()
print('Random forest after Cross validation Train accuracy:', crossval_train_rf)
print('-------'*10)
print('Random forest  after Cross validation Test accuracy:', crossval_test_rf)
```

```
Random forest after Cross validation Train accuracy: 0.9742338990082896
----------------------------------------------------------------------
Random forest  after Cross validation Test accuracy: 0.9388229739144627
```

Conclusion

- I used various Supervised machine learning models to solve the classification problem performed on 100,000 observations of space taken by the SDSS (Sloan Digital Sky Survey).
- The dataset has 59,445 galaxies, 21,593 stars and 18,961 quasar objects.
- Dimension reduction was done with the help of Principal Component Analysis.
- For Evaluation I used accuracy score & confusion matrix.
- Random Forest classifier with Train accuracy at 100% and Test accuracy at 97% gives the highest accuracy amongst all the models but it has overfitting issue.
- So after cross validation to deal with overfitting issue of Random Forest model train accuracy is 97% and Test accuracy is 93% making it the best model to solve the classification problem for this dataset.
- Decision Tree follows next in accuracy but post prunning its gives accuracy of 75% for both Train & Test data.
- Gaussian Naive Bayes Classifier performs the worst with 76% accuracy for Train data & 77% for Test data making it almost at par with Decision Tree accuracy.
- Voting ensemble method yielded Train accuracy of 97% & test accuracy of 95%.
- Concluding the same, we can say that rest all models performed more than 80% for both Train & Test data and also there was no high variance problem in any of the model.

In [ ]: