STAR CLASSIFICATION

USING XGBOOST

ALL ABOUT THE DATASET

The “Stellar Classification Dataset – SDSS17” by FEDESORIANO is made from the “Sloan Digital Sky Survey” project and contains spectroscopic observations of celestial objects in the night sky. The dataset consists of 100,000 observations, each described by 17 feature columns and 1 class column that identifies it as a star, galaxy, or quasar. Quasars, short for “quasi-stellar radio sources,” are incredibly bright and distant astronomical objects found at the centers of galaxies. Quasars are not stars but the active cores of distant galaxies powered by supermassive black holes.

Importing the libraries

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
from matplotlib import pyplot as plt
import seaborn as sns
import math
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
import plotly.figure_factory as ff
from sklearn.preprocessing import label_binarize
```

Drive mount and loading the dataset

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')#Mount the drive

     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ]  file_path = '/content/drive/My Drive/Advanced Applied machine learning/star_classification.csv'
     # Read the CSV file into a DataFrame
     df = pd.read_csv(file_path)
```

Exploratory Data Analysis(EDA)

```
df.sample(8)
```

	obj_ID	alpha	delta	u	g	r	i	z	run_ID	rerun_ID	cam_col	field_ID	spec_obj_ID	class	redshift	pla
41875	1.237659e+18	239.567384	39.402618	24.63467	20.72572	19.03342	18.33362	18.20663	3180	301	3	211	5.845886e+18	GALAXY	0.226434	51
95209	1.237661e+18	183.891225	46.268753	19.50289	18.49177	18.19354	18.07250	18.00694	3698	301	1	164	8.359954e+18	STAR	0.000340	74
98722	1.237661e+18	145.450266	39.118061	20.89660	19.96199	19.89736	19.89189	19.99044	3530	301	1	189	3.630054e+18	STAR	0.000629	32
56514	1.237658e+18	135.050350	4.275687	23.76919	23.75406	21.24608	20.00530	19.56291	3015	301	2	140	4.294446e+18	GALAXY	0.571624	38
8688	1.237659e+18	208.823630	51.942133	22.99855	21.18974	20.97836	20.93164	20.58851	3180	301	2	47	7.588840e+18	STAR	0.000350	67
95224	1.237679e+18	345.298478	22.973691	21.82521	20.35423	20.24570	20.15662	19.60457	7708	301	2	115	7.421013e+18	QSO	2.375080	65
46816	1.237679e+18	25.524589	11.618876	21.10785	21.15398	20.81400	20.74949	20.78499	7773	301	3	477	1.245362e+19	QSO	1.223312	110
33466	1.237662e+18	168.682439	42.223693	19.85040	19.68569	19.37961	19.36574	19.36136	3840	301	6	95	9.420612e+18	QSO	1.170880	83

Running sample function on the dataframe to view the random set ove values from the dataframe

Isnull()

Data Integrity: In a dataset, null values signify information that is absent or unclear.
Making ensuring that there are no unexpected null values in your data contributes to
data integrity and guarantees the accuracy of any analyses and operations done on it.

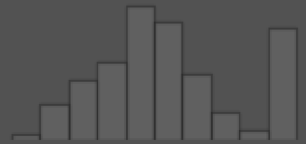
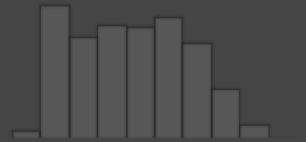
Error Prevention: Null values can lead to mistakes in computations, modeling, and data processing.
These errors can be avoided by properly handling null data, such as by omitting them from analyses
or imputing meaningful values to them.

Preventing Bias: Improper handling of null values might result in the introduction of bias into models or
analysis. For instance, the findings may be skewed toward particular groups or dataset characteristics
if records with null values are excluded. Through the process of identifying null values and managing them
correctly, you might be able to mitigate the biases

```
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
```

THIS SHOWS THAT THERE ARE NO PRERSISTANT NULL VALUES IN THE DATAFRAME.

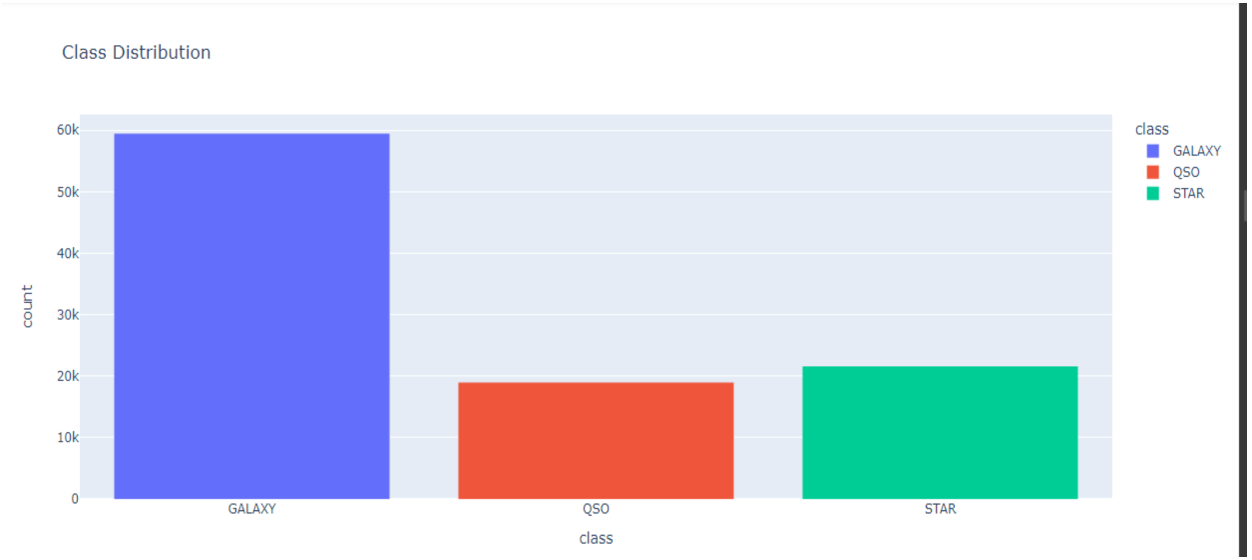
Data Frame Summary
df
Dimensions: 100,000 x 18
Duplicates: 0

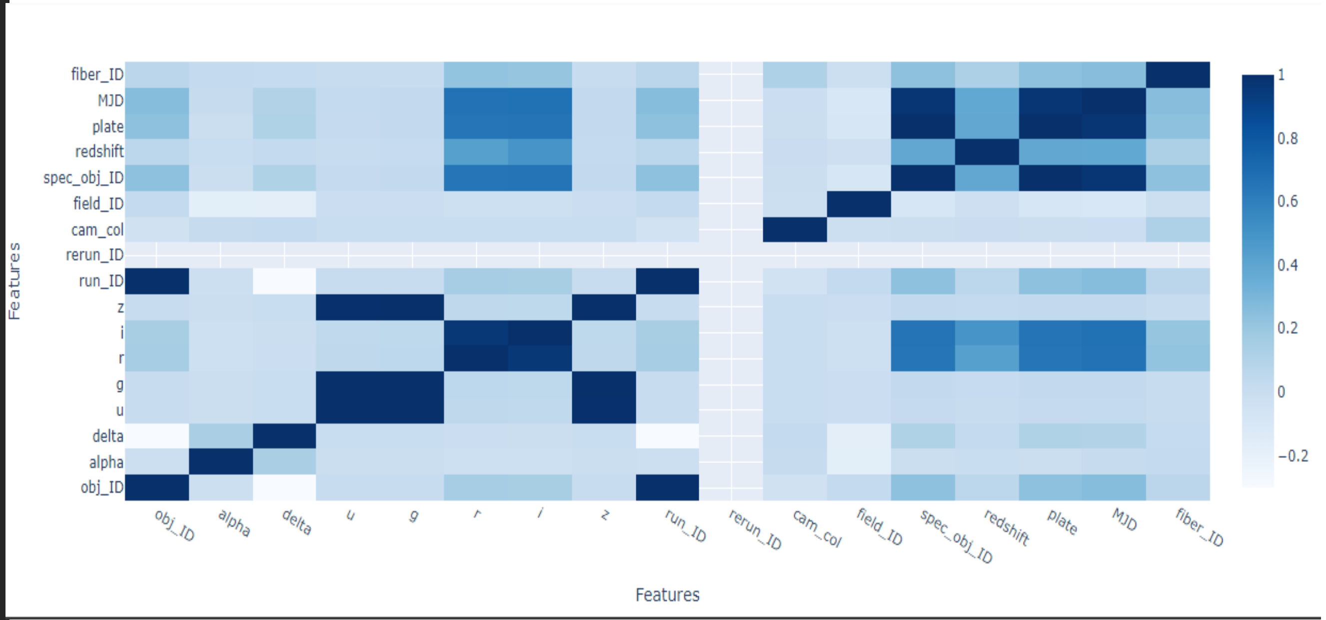
No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	obj_ID [float64]	Mean (sd) : 1237664721814903296.0 (8438559894562.6) min < med < max: 1237645942904389888.0 < 1237663463144292864.0 < 1237680531356386304.0 IQR (CV) : 9189091327744.0 (146667.8)	78,053 distinct values		0 (0.0%)
2	alpha [float64]	Mean (sd) : 177.6 (96.5) min < med < max: 0.0 < 180.9 < 360.0 IQR (CV) : 106.4 (1.8)	99,999 distinct values		0 (0.0%)
3	delta [float64]	Mean (sd) : 24.1 (19.6) min < med < max: -18.8 < 23.6 < 83.0 IQR (CV) : 34.8 (1.2)	99,999 distinct values		0 (0.0%)
4	u [float64]	Mean (sd) : 22.0 (31.8) min < med < max: -9999.0 < 22.2 < 32.8 IQR (CV) : 3.3 (0.7)	93,748 distinct values		0 (0.0%)

Summary of the dataframe to clearly view the statistical and graphical representation of each feature

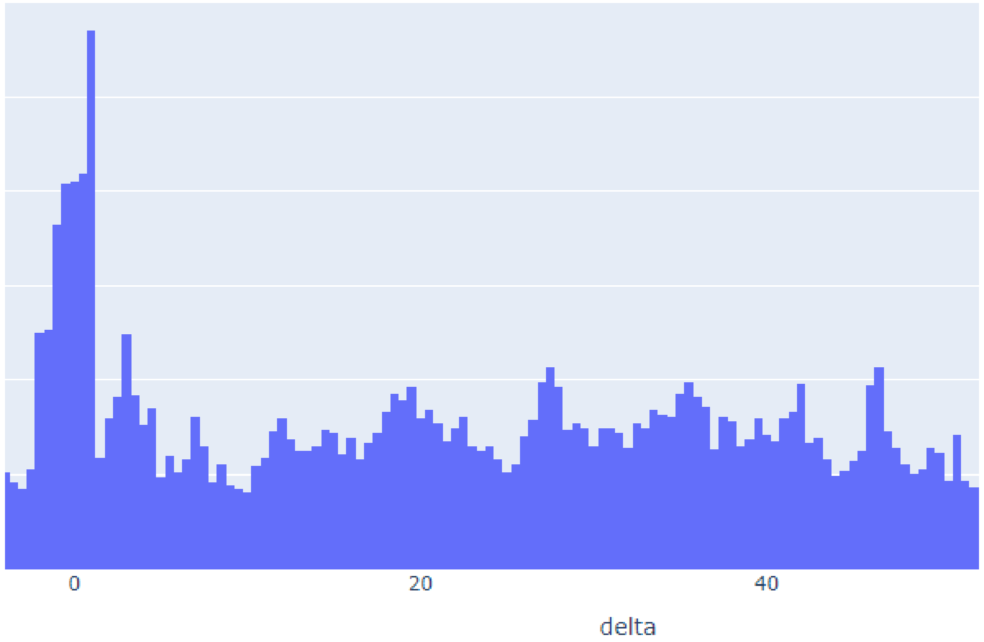
EDA

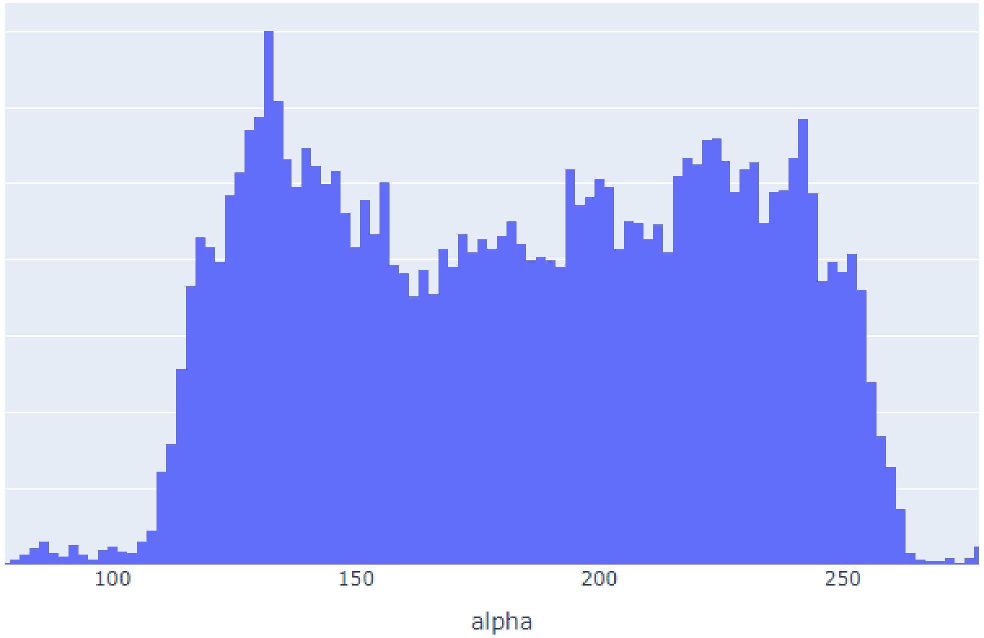
Bar graph on the target class and a correlation map of all the features in the dataframe.





Histogram of alpha and delta attributes of the dataframe





Mapping the class names to numerical values to avoid the disparities.

```
[ ]  # mapping from class names to numerical values
     class_mapping = {'GALAXY': 0, 'QSO': 1, 'STAR': 2}
     # changing the class names to numerical values in the 'class' column
     df['class'] = df['class'].replace(class_mapping)
```

Splitting the data into training and testing data

```
# For this dataset, 'class' is the target and already encoded

# Splitting the dataset into features and target variable
X = df.drop(['class'], axis=1)
y = df['class']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Training the XGBoost for multi class classification and Evaluating its performance using F1 score

Instantiate XGBoost

Specify Evaluation Set

Choose Evaluation Metric

Train the Model

Make Predictions

Calculate F1 Score

```
[ ] from xgboost import XGBClassifier
    from sklearn.metrics import f1_score

    # Instantiate the XGBoost classifier for multi-class classification
    model = XGBClassifier(booster='gbtree', objective='multi:softmax', num_class=3, random_state=2)

    # Specify the evaluation set
    eval_set = [(X_test, y_test)]

    # Use a suitable evaluation metric for multi-class classification
    eval_metric = 'mlogloss'  # or 'merror'
    model.fit(X_train, y_train, eval_metric=eval_metric, eval_set=eval_set)

    # Make predictions for test data
    y_pred = model.predict(X_test)

    # Calculate the F1 score with average parameter for multi-class
    f1 = f1_score(y_test, y_pred, average='weighted')  # 'weighted' accounts for label imbalance

    # Print the F1 score
    print("F1 Score: %.2f" % f1)
```

mLogLoss allows for fair comparison between different models. Models with lower mLogLoss values are generally considered to have better performance

Hyper parameter Tuning and fitting the model on the train data and performing random search

```
xgb_param_grid = {
    'n_estimators': np.arange(50, 400, 50),
    'max_depth': np.arange(3, 15),
    'learning_rate': np.linspace(0.01, 0.3, 10),
    'subsample': np.linspace(0.6, 1.0, 5),
    'min_child_weight': np.arange(1, 10, 2),
    'gamma': np.linspace(0, 0.5, 5),
    'colsample_bytree': np.linspace(0.6, 1.0, 5),
    'reg_alpha': np.linspace(0, 1, 5)
}

# Initialize the XGBoost classifier with additional settings to handle warnings, etc.
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss')

# Perform Randomized Search with the expanded grid
xgb_random_search = RandomizedSearchCV(xgb, xgb_param_grid, n_iter=10, scoring='f1_macro', cv=5, verbose=2, random_state=42, n_jobs=-1)
xgb_random_search.fit(X_train, y_train)
```

Extracting the best model and getting the predictions

```python
    # Extract the best model
    best_model = xgb_random_search.best_estimator_

    # Make predictions with the best model
    y_pred = best_model.predict(X_test)

    # Calculate F1 score
    f1 = f1_score(y_test, y_pred, average='macro')

    print('Best parameters:', xgb_random_search.best_params_)
    print('Best F1 Score:', f1)

    return best_model


best_model = tune_and_train_xgboost_model(X_train, y_train, X_test, y_test)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters: {'subsample': 1.0, 'reg_alpha': 0.5, 'n_estimators': 200, 'min_child_weight': 7, 'max_depth': 12, 'learning_rate': 0.07444444444444444, 'gamma':
Best F1 Score: 0.9738236926855697
```

Training the model again with best parameters to get best outputs with best f1 score

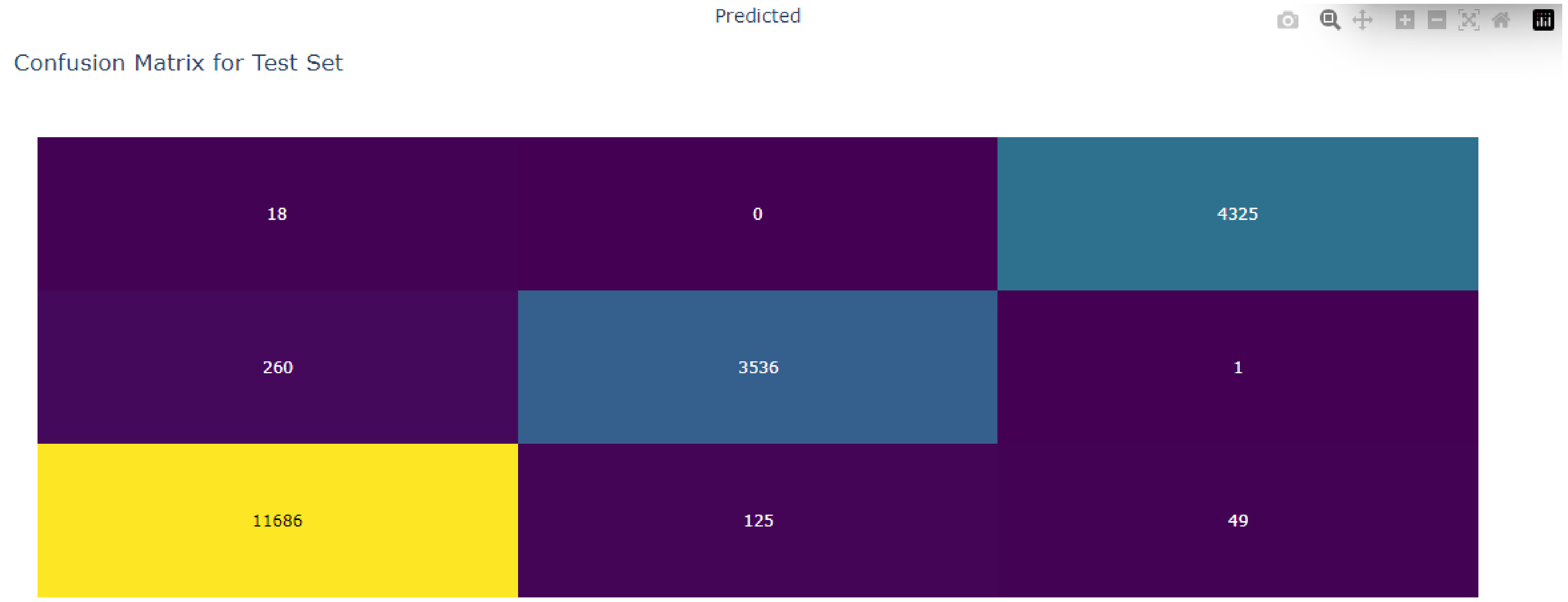
```
# Step 1: Get the best model (already trained with the best parameters)
best_model = tune_and_train_xgboost_model(X_train, y_train, X_test, y_test)

# If you need to retrain the model or want to explicitly show the training with the best parameters, you can do the following:
# Extract the best parameters from the model (Not necessary if using the model as is)
best_params = best_model.get_params()

# Initialize a new XGBoost model with these best parameters
new_best_xgb_model = XGBClassifier(**best_params)

# Train this new model (Optional if you are using the model returned by the function)
new_best_xgb_model.fit(X_train, y_train)

# Now, `new_best_xgb_model` is your trained XGBoost model with the best parameters found
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best parameters: {'subsample': 1.0, 'reg_alpha': 0.5, 'n_estimators': 200, 'min_child_weight': 7, 'max_depth': 12, 'learning_rate': 0.07444444444444444, 'gamma':
Best F1 Score: 0.9738236926855697
```

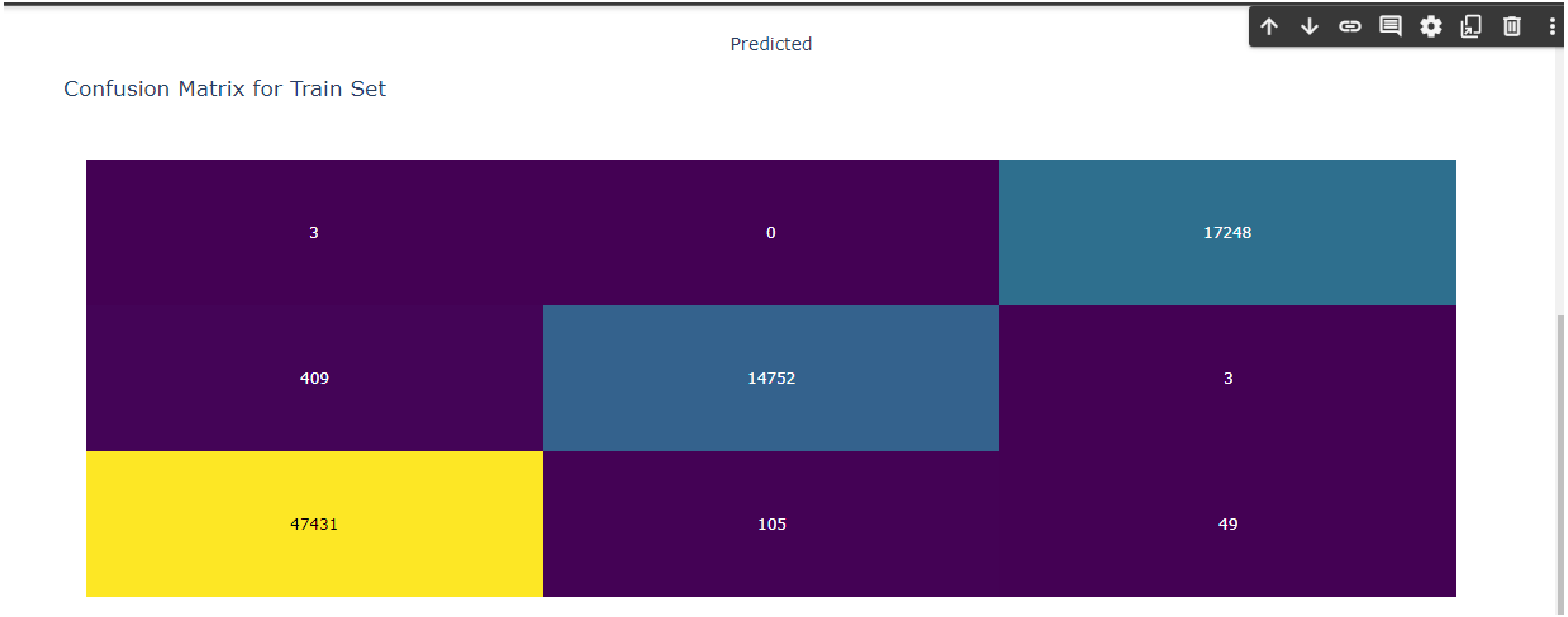
	precision	recall	f1-score	support
0	0.98	0.99	0.98	11860
1	0.97	0.93	0.95	3797
2	0.99	1.00	0.99	4343
accuracy			0.98	20000
macro avg	0.98	0.97	0.97	20000
eighted avg	0.98	0.98	0.98	20000

Classification Report

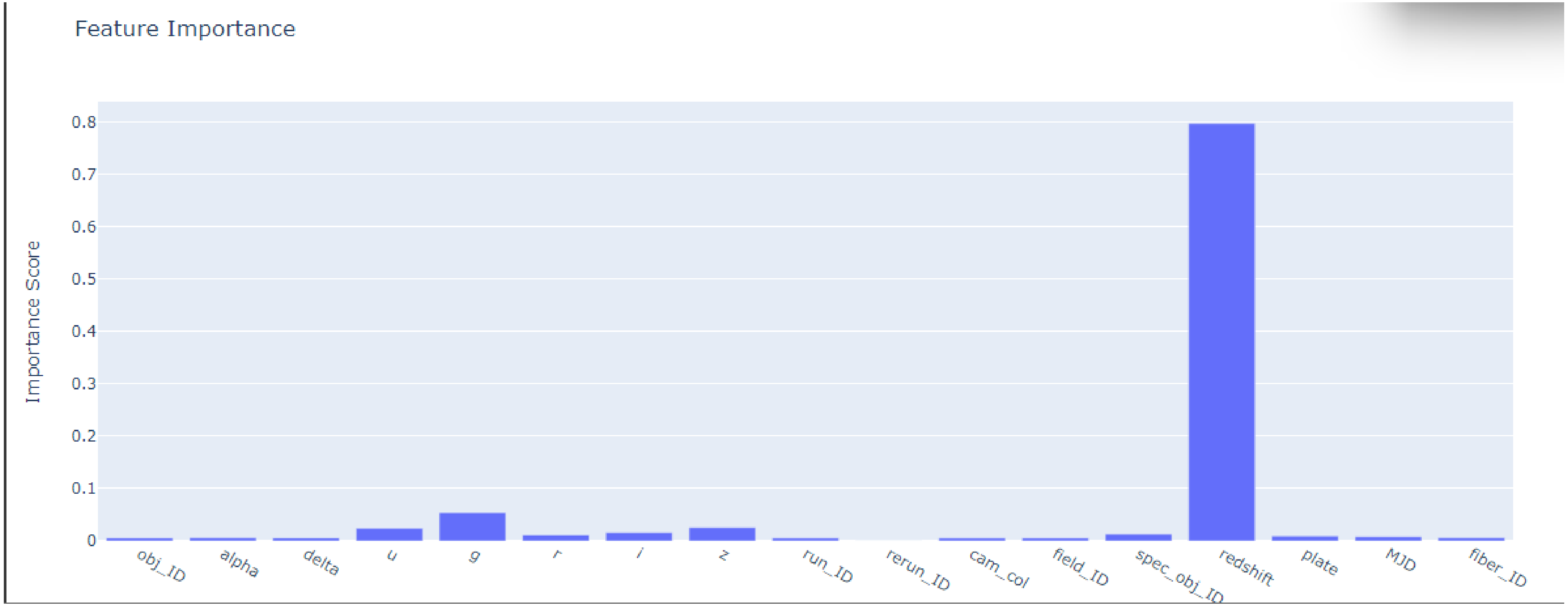
Confusion matrix of test data



Confusion matrix of train data



Visualizing the feature importance



It is evident that redshift has high importance.

Plotting the MULTI-CLASS ROC Curve.