stellar-classification-model

October 28, 2024

0.1 Introduction

The vastness of the universe has always intrigued humanity, and advancements in technology have allowed us to observe celestial bodies with unprecedented detail.

The Sloan Digital Sky Survey (SDSS) has provided a wealth of data about stars, galaxies, and quasars through its extensive imaging and spectroscopic observations.

This project aims to leverage the Stellar Classification Dataset - SDSS17 to build a robust machine learning model that classifies astronomical objects based on their features.

By utilizing classification models, we can automate the identification and categorization of celestial objects, which traditionally relied on manual analysis by astronomers.

0.2 Importance of the Project

The ability to accurately classify astronomical objects is essential for a variety of reasons:

```
1.Scientific Research: Understanding the composition and distribution of different types of cel
```

```
2.Efficiency: Automating the classification process allows astronomers to analyze large datase
```

```
3.Educational Value: This project serves as an excellent opportunity to apply machine learning
```

0.3 Classes

Quasars:

```
A quasar is an extremely luminous active galactic nucleus (AGN). It is pronounced / kwe z r/ KW
```

Star:

```
A star's life begins with the gravitational collapse of a gaseous nebula of material composed p
if it is sufficiently massive-a black hole.
```

Galaxies:

```
A galaxy is a system of stars, stellar remnants, interstellar gas, dust, dark matter, bound tog
supergiants with one hundred trillion stars,[4] each orbiting its galaxy's center of mass. Mos
```

0.4 Dataset Features

The Stellar Classification Dataset - SDSS17 comprises various features that provide critical information about each astronomical object. Here's a breakdown of the key features and their significance:

obj_ID: A unique identifier for each object in the dataset, allowing for precise tracking and referencing throughout the analysis.

alpha (Right Ascension) & delta (Declination): These coordinates define the object's position in the sky, similar to latitude and longitude on Earth. They are essential for locating celestial objects and conducting spatial analysis.

u, g, r, i, z: These photometric measurements represent the intensity of light detected in different wavelengths (ultraviolet to infrared). Analyzing these values helps in determining the object's temperature, chemical composition, and distance from Earth.

run_ID & rereun_ID: These identifiers specify the scan session and processing rerun of the data, ensuring that each observation is correctly matched with its imaging session.

cam_col (Camera Column): Indicates the specific scanline within a run, useful for identifying the exact instrument settings used during observation.

field_ID: Identifies the specific field of view during the observation, helping researchers understand the context of the data collected.

spec_obj_ID: A unique identifier for objects with spectroscopic measurements, linking photometric and spectroscopic data for more detailed analysis.

class: The categorical classification of the object (e.g., galaxy, star, quasar), which is the target variable for our machine learning models.

redshift: A measure of how much the light has shifted toward longer wavelengths due to the object's motion away from Earth, which is crucial for estimating distances and velocities.

plate, MJD, fiber_ID: These features provide information about the specific observations and instruments used, ensuring the data's reliability and facilitating future comparisons.

0.5 Project Benefits

This machine learning project offers multiple benefits:

```
1.Enhanced Classification Accuracy: By using classification models, we can achieve high accura

2.Insights into Cosmic Phenomena: The results can lead to discoveries regarding the distributi

3.Scalability: As new data from future surveys becomes available, the model can be easily retr
```

By harnessing the power of machine learning in astronomical research, this project not only contributes to the field of astronomy but also serves as a prime example of how data science techniques can be applied to solve complex scientific challenges.

[1]:
```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages
(0.12.4)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: numpy>=1.17.3 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.14.0)
Requirement already satisfied: scikit-learn>=1.0.2 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.5.0)
Requirement already satisfied: joblib>=1.1.1 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\users\hp\appdata\local\programs\python\python312\lib\site-packages (from
imbalanced-learn) (3.5.0)
```

0.6 Libraries:

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.io as pio
import plotly.colors as plc
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import matplotlib.cm as cm

from plotly.subplots import make_subplots
import plotly.graph_objects as go
from sklearn.model_selection import
 ↪train_test_split,GridSearchCV,RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler,
 ↪MinMaxScaler,RobustScaler,LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
↪GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import f1_score, classification_report,␣
↪confusion_matrix,accuracy_score,f1_score,precision_score,recall_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
from sklearn.ensemble import IsolationForest
from imblearn.over_sampling import SMOTE
from collections import Counter
```

[3]:
```python
df = pd.read_csv('star_classification.csv')
```

[4]:
```python
df.head()
```

[4]:
```
         obj_ID       alpha      delta         u         g         r  \
0  1.237661e+18  135.689107  32.494632  23.87882  22.27530  20.39501
1  1.237665e+18  144.826101  31.274185  24.77759  22.83188  22.58444
2  1.237661e+18  142.188790  35.582444  25.26307  22.66389  20.60976
3  1.237663e+18  338.741038  -0.402828  22.13682  23.77656  21.61162
4  1.237680e+18  345.282593  21.183866  19.43718  17.58028  16.49747

          i         z  run_ID  rerun_ID  cam_col  field_ID   spec_obj_ID  \
0  19.16573  18.79371    3606       301        2        79  6.543777e+18
1  21.16812  21.61427    4518       301        5       119  1.176014e+19
2  19.34857  18.94827    3606       301        2       120  5.152200e+18
3  20.50454  19.25010    4192       301        3       214  1.030107e+19
4  15.97711  15.54461    8102       301        3       137  6.891865e+18

    class  redshift  plate    MJD  fiber_ID
0  GALAXY  0.634794   5812  56354       171
1  GALAXY  0.779136  10445  58158       427
2  GALAXY  0.644195   4576  55592       299
3  GALAXY  0.932346   9149  58039       775
4  GALAXY  0.116123   6121  56187       842
```

[5]:
```python
rows,columns = df.shape
print(f"Number of rows/examples: {rows}")
print(f"Number of columns/features: {columns}")
```

```
Number of rows/examples: 100000
Number of columns/features: 18
```

[6]:
```python
df.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
```

[7]:
```
df.describe().T
```

[7]:
```
                count          mean           std           min           25%  \
obj_ID       100000.0  1.237665e+18  8.438560e+12  1.237646e+18  1.237659e+18
alpha        100000.0  1.776291e+02  9.650224e+01  5.527828e-03  1.275182e+02
delta        100000.0  2.413530e+01  1.964467e+01 -1.878533e+01  5.146771e+00
u            100000.0  2.198047e+01  3.176929e+01 -9.999000e+03  2.035235e+01
g            100000.0  2.053139e+01  3.175029e+01 -9.999000e+03  1.896523e+01
r            100000.0  1.964576e+01  1.854760e+00  9.822070e+00  1.813583e+01
i            100000.0  1.908485e+01  1.757895e+00  9.469903e+00  1.773228e+01
z            100000.0  1.866881e+01  3.172815e+01 -9.999000e+03  1.746068e+01
run_ID       100000.0  4.481366e+03  1.964765e+03  1.090000e+02  3.187000e+03
rerun_ID     100000.0  3.010000e+02  0.000000e+00  3.010000e+02  3.010000e+02
cam_col      100000.0  3.511610e+00  1.586912e+00  1.000000e+00  2.000000e+00
field_ID     100000.0  1.861305e+02  1.490111e+02  1.100000e+01  8.200000e+01
spec_obj_ID  100000.0  5.783882e+18  3.324016e+18  2.995191e+17  2.844138e+18
redshift     100000.0  5.766608e-01  7.307073e-01 -9.970667e-03  5.451684e-02
plate        100000.0  5.137010e+03  2.952303e+03  2.660000e+02  2.526000e+03
MJD          100000.0  5.558865e+04  1.808484e+03  5.160800e+04  5.423400e+04
fiber_ID     100000.0  4.493127e+02  2.724984e+02  1.000000e+00  2.210000e+02

                       50%           75%           max
```

```
obj_ID       1.237663e+18  1.237668e+18  1.237681e+18
alpha        1.809007e+02  2.338950e+02  3.599998e+02
delta        2.364592e+01  3.990155e+01  8.300052e+01
u            2.217914e+01  2.368744e+01  3.278139e+01
g            2.109983e+01  2.212377e+01  3.160224e+01
r            2.012529e+01  2.104478e+01  2.957186e+01
i            1.940514e+01  2.039650e+01  3.214147e+01
z            1.900460e+01  1.992112e+01  2.938374e+01
run_ID       4.188000e+03  5.326000e+03  8.162000e+03
rerun_ID     3.010000e+02  3.010000e+02  3.010000e+02
cam_col      4.000000e+00  5.000000e+00  6.000000e+00
field_ID     1.460000e+02  2.410000e+02  9.890000e+02
spec_obj_ID  5.614883e+18  8.332144e+18  1.412694e+19
redshift     4.241733e-01  7.041543e-01  7.011245e+00
plate        4.987000e+03  7.400250e+03  1.254700e+04
MJD          5.586850e+04  5.677700e+04  5.893200e+04
fiber_ID     4.330000e+02  6.450000e+02  1.000000e+03
```

[8]:
```
df.isnull().sum()
```

[8]:
```
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
```

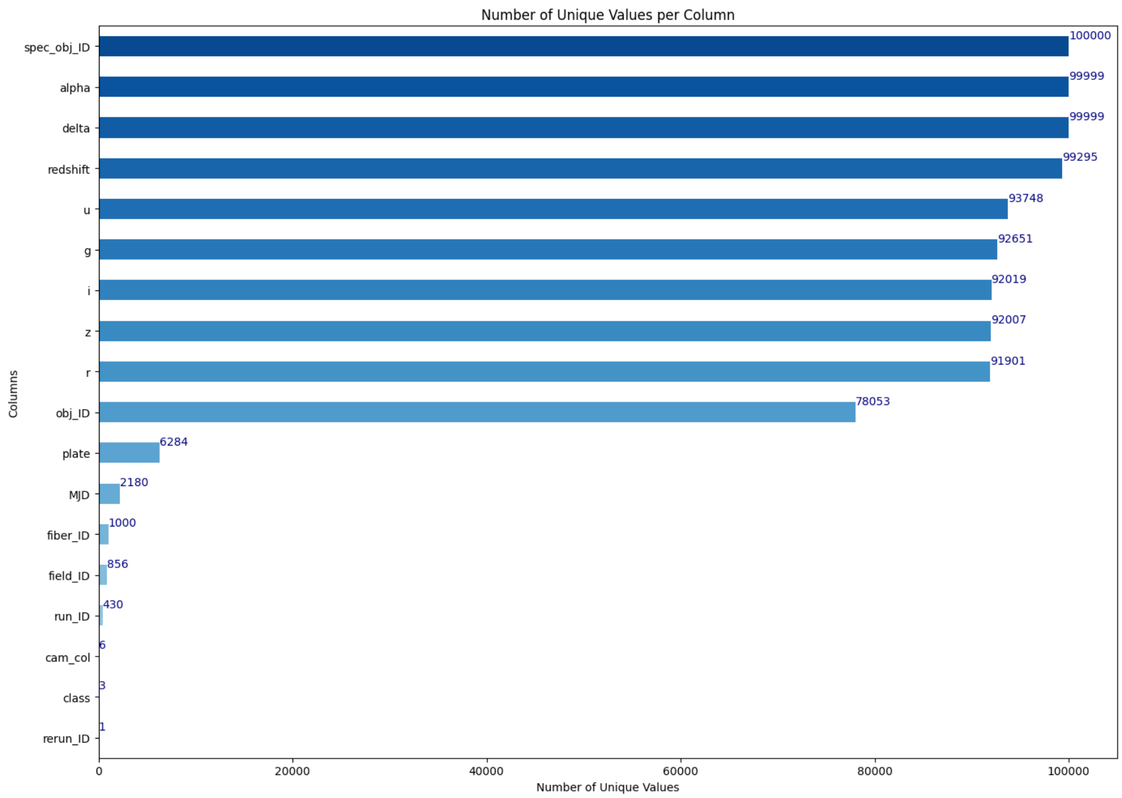
[9]:
```
df.duplicated().sum()
```

[9]:
```
0
```

0.7 EDA :

0.7.1 Unique Values :

[10]:
```
unique_values = df.nunique().sort_values()  # calculate unique values for each␣
 ↪column
rows = len(unique_values)
unique_values = pd.DataFrame(unique_values)  # convert to DataFrame
unique_values = unique_values.rename(columns={0: 'Unique Values'})  # rename␣
 ↪column



unique_values["Total Rows"] = rows

num_bars = len(unique_values)
colors = cm.Blues(np.linspace(0.3, 0.9, num_bars))
plt.figure(figsize=(14, 10))
ax = unique_values['Unique Values'].plot(kind="barh", color=colors)
plt.title("Number of Unique Values per Column")
plt.ylabel("Columns")
plt.xlabel("Number of Unique Values")

for p in ax.patches:
    ax.annotate(
        str(p.get_width()),
        (p.get_width() + 5, p.get_y() + p.get_height() / 2.),
        va='center',
        xytext=(0, 10),
        textcoords='offset points',
        color='navy',
        fontsize=10
    )


plt.tight_layout()
plt.show()
```



[11]:
```
#pip install nbformat
```

[12]:
```
dark_palette = ['#1e3a5f', '#AA336A', '#D8BFD8']  # Darker shades of blue

fig = px.histogram(df, x='class', color='class',
                   color_discrete_sequence=dark_palette,
                   title='Distribution of Classes')

# Update layout to customize axes and grid
fig.update_layout(
    xaxis_title='Class',
    yaxis_title='Count',
    title_font=dict(color='#13274F'),
    xaxis=dict(tickfont=dict(color='#13274F')),
    yaxis=dict(tickfont=dict(color='#13274F')),
    margin=dict(l=40, r=40, t=40, b=40),
)

# Show the figure
fig.show()
```

[13]:
```
class_counts = df['class'].value_counts()

# Create a pie chart using Plotly
fig = px.pie(
    names=class_counts.index,
    values=class_counts.values,
    title='Class Distribution of Celestial Objects',
    color_discrete_sequence=dark_palette  # Darker shades of blue
)

# Update layout to customize the text color
fig.update_traces(textinfo='percent+label', textfont=dict(color='white'))  #␣
↪Customize text color
fig.update_layout(title_font=dict(color='#13274F'), margin=dict(l=40, r=40,␣
↪t=40, b=40))

# Show the figure
fig.show()
```

[14]:
```
df.drop(df[df['z']<0].index,inplace=True)
features = ['u', 'g', 'r', 'i', 'z']
n_features = len(features)
palette=['#1e3a5f', '#AA336A', '#76ABDF']
fig = make_subplots(rows=n_features, cols=1, subplot_titles=features)


for i, feature in enumerate(features):
    for class_name in df['class'].unique():
        class_data = df[df['class'] == class_name][feature]
        fig.add_trace(
            go.Histogram(
                x=class_data,
                name=class_name,
                opacity=0.7,
                histnorm='probability density',
                marker_color=palette[df['class'].unique().tolist().
↪index(class_name)],
                legendgroup=class_name,
                showlegend=(i == 0)
            ),
            row=i + 1, col=1
        )

fig.update_layout(
    title='Distributions of u, g, r, i, z Colored by Class',
    xaxis_title='Magnitude',
    yaxis_title='Density',
```

```
    height=800,
    showlegend=True
)


fig.show()
```

[15]:
```
average_values = df.groupby('class')[['u', 'g', 'r', 'i', 'z']].mean().
 ↪reset_index()

# Melt the dataframe to have a long format suitable for plotly
average_values_melted = average_values.melt(id_vars='class',
                                             value_vars=['u', 'g', 'r', 'i',␣
 ↪'z'],
                                             var_name='Band',␣
 ↪value_name='Average')

# Create the bar plot using Plotly
fig = px.bar(average_values_melted,
             x='class',
             y='Average',
             color='Band',
             barmode='group',
             title='Average of u, g, r, i, z for Each Class',
             color_discrete_sequence=px.colors.sequential.Purples_r + px.colors.
 ↪sequential.Blues_r)

# Show the figure
fig.show()
```

[16]:
```
fig = px.histogram(df,
                   x='alpha',
                   color='class',
                   nbins=50,
                   title='Distribution of Alpha ( ) for Each Class',
                   labels={'alpha': 'Alpha ( )', 'class': 'Class'},
                   color_discrete_sequence=dark_palette,
                   opacity=0.7)

# Update layout for better visualization
fig.update_layout(barmode='overlay',
                  xaxis_title='Alpha ( )',
                  yaxis_title='Frequency')

# Show the figure
fig.show()
```

[17]:
```python
palette=['#2E5090','#9F2B68','#D8BFD8']
fig = px.histogram(df,
                   x='delta',
                   color='class',
                   nbins=50,
                   title='Distribution of Delta ( ) for Each Class',
                   labels={'delta': 'Delta ( )', 'class': 'Class'},
                   color_discrete_sequence=palette,
                   opacity=0.7)

# Update layout for better visualization
fig.update_layout(barmode='overlay',
                  xaxis_title='Delta ( )',
                  yaxis_title='Frequency')

# Show the figure
fig.show()
```

[18]:
```python
fig = px.scatter(df,
                 x='alpha',
                 y='delta',
                 color='class',
                 title='Scatter Plot of Alpha ( ) vs Delta ( ) for Each Class',
                 labels={'alpha': 'Alpha ( )', 'delta': 'Delta ( )', 'class': 'Class'},
                 color_discrete_sequence=palette,
                 opacity=0.7)

# Update layout for better visualization


# Show the figure
fig.show()
```

0.8 Multicolinearity :

[19]:
```python
df_encoded=df.copy()
df_encoded['class'] = df_encoded['class'].map({'GALAXY': 0, 'QSO': 1, 'STAR': 2})
```

[20]:
```python
df.drop(['run_ID','rerun_ID','cam_col','field_ID','spec_obj_ID','obj_ID'],axis=1,inplace=True)
df_encoded.drop(['run_ID','rerun_ID','cam_col','field_ID','spec_obj_ID','obj_ID'],axis=1,inplace=True)
```

[21]:

```python
# Calculate the Pearson and Spearman correlation matrices
pearson_corr = df_encoded.corr(method='pearson')
spearman_corr = df_encoded.corr(method='spearman')

# Function to create a heatmap with annotations
def create_heatmap(corr_matrix, title):
    # Convert the correlation matrix to a format that can be used for heatmap
    heatmap = go.Heatmap(
        z=corr_matrix.values,
        x=corr_matrix.columns,
        y=corr_matrix.index,
        colorscale='Blues',
        zmin=-1,  # Set min and max values to -1 and 1 for correlation heatmaps
        zmax=1,
        colorbar=dict(title="Correlation")
    )

    # Create annotations to show the correlation coefficients
    annotations = []
    for i in range(len(corr_matrix)):
        for j in range(len(corr_matrix.columns)):
            annotations.append(
                dict(
                    x=corr_matrix.columns[j],
                    y=corr_matrix.index[i],
                    text=str(round(corr_matrix.iat[i, j], 2)),  # Round to 2␣
↪decimal places
                    showarrow=False,
                    font=dict(color="navy")
                )
            )

    # Create the figure
    fig = go.Figure(data=[heatmap])
    fig.update_layout(
        title=title,
        annotations=annotations,
        xaxis_title="Features",
        yaxis_title="Features",
        xaxis=dict(tickfont=dict(color='navy')),
        yaxis=dict(tickfont=dict(color='navy')),
    )

    return fig

# Create the heatmaps for Pearson and Spearman
pearson_fig = create_heatmap(pearson_corr, 'Pearson Correlation Heatmap')
```

```
spearman_fig = create_heatmap(spearman_corr, 'Spearman Correlation Heatmap')

# Display the heatmaps
pearson_fig.show()
spearman_fig.show()
```

0.9 Feature Engineering :

The differences between the magnitudes (or bands) represent the relative brightness of the astronomical objects in various parts of the electromagnetic spectrum. Here's a breakdown of what each of these differences might indicate:

```
u-g: This difference measures how much more luminous an object is in the ultraviolet band ( u)

    Importance: A higher value suggests that the object emits significantly more light in the u

g-r: This difference measures the relative brightness in the green band compared to the red ban

    Importance: A positive value indicates that the object is brighter in the green band than i

r-i: This difference assesses how much brighter an object is in the red band compared to the in

    Importance: This feature can provide insights into the object's composition and surface tem

i-z: This difference looks at the brightness in the infrared band compared to the near-infrared

    Importance: Understanding the characteristics of the light emitted in these bands can help
```

[22]:
```
df['u_g'] = df['u'] - df['g']
df['g_r'] = df['g'] - df['r']
df['r_i'] = df['r'] - df['i']
df['i_z'] = df['i'] - df['z']
df.drop(['u','g','r','i','z'],axis=1,inplace=True)
df_encoded['u_g'] = df_encoded['u'] - df_encoded['g']
df_encoded['g_r'] = df_encoded['g'] - df_encoded['r']
df_encoded['r_i'] = df_encoded['r'] - df_encoded['i']
df_encoded['i_z'] = df_encoded['i'] - df_encoded['z']
df_encoded.drop(['u','g','r','i','z'],axis=1,inplace=True)
```

[23]:
```
pearson_corr = df_encoded.corr(method='pearson')
spearman_corr = df_encoded.corr(method='spearman')

def create_heatmap(corr_matrix, title):
    heatmap = go.Heatmap(
        z=corr_matrix.values,
        x=corr_matrix.columns,
        y=corr_matrix.index,
        colorscale='Blues',
```

```
        zmin=-1,
        zmax=1,
        colorbar=dict(title="Correlation")
    )

    annotations = []
    for i in range(len(corr_matrix)):
        for j in range(len(corr_matrix.columns)):
            annotations.append(
                dict(
                    x=corr_matrix.columns[j],
                    y=corr_matrix.index[i],
                    text=str(round(corr_matrix.iat[i, j], 2)),
                    showarrow=False,
                    font=dict(color="navy")
                )
            )

    fig = go.Figure(data=[heatmap])
    fig.update_layout(
        title=title,
        annotations=annotations,
        xaxis_title="Features",
        yaxis_title="Features",
        xaxis=dict(tickfont=dict(color='navy')),
        yaxis=dict(tickfont=dict(color='navy')),
    )

    return fig

pearson_fig = create_heatmap(pearson_corr, 'Pearson Correlation Heatmap')
spearman_fig = create_heatmap(spearman_corr, 'Spearman Correlation Heatmap')

pearson_fig.show()
spearman_fig.show()
```

[24]:
```
import plotly.subplots as sp

num_features = df.shape[1] - 1
num_cols = 3
num_rows = (num_features // num_cols) + 1

fig = sp.make_subplots(rows=num_rows, cols=num_cols, subplot_titles=df.columns[:
 ↪-1].tolist())

# Create boxplots for each feature
for i, column in enumerate(df.columns[:-1]):  # Exclude the target variable
```

```
    row = i // num_cols + 1  # Calculate row index
    col = i % num_cols + 1  # Calculate column index
    boxplot = px.box(df, y=column, title=f'Boxplot of {column}',␣
↪color_discrete_sequence=px.colors.sequential.Blues_r)

    # Add the boxplot trace to the subplot
    for trace in boxplot.data:
        fig.add_trace(trace, row=row, col=col)

# Update layout
fig.update_layout(title_text='Boxplots of Stellar Classification Features',␣
↪height=800)
fig.show()
```

0.10 Splitting Dataset :

[25]:
```
x=df.drop('class',axis=1)
y=df['class']
```

0.11 Modelling:

[26]:
```
final_best_models = pd.DataFrame(columns=["Model", "Scaling Method", "Train␣
↪Accuracy", "Test Accuracy",
                                          "Train Precision", "Test Precision",␣
↪"Train Recall", "Test Recall",
                                          "Train F1-Score", "Test F1-Score"])

def Model_Evaluation_Pipline(x, y, model_name, scaler_name=None,␣
↪param_grid=None):
    global final_best_models
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42, stratify=y)

    if scaler_name == 'StandardScaler':
        scaler = StandardScaler()
    elif scaler_name == 'MinMaxScaler':
        scaler = MinMaxScaler()
    elif scaler_name == 'RobustScaler':
        scaler = RobustScaler()
    else:
        scaler = None

    steps = []
    if scaler:
        steps.append(('scaler', scaler))
    steps.append(('classifier', model_name))
```

```
    pipeline = Pipeline(steps)

    grid_search = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1,␣
↪scoring='f1_macro')
    grid_search.fit(x_train, y_train)

    best_model = grid_search.best_estimator_

    y_train_pred = best_model.predict(x_train)
    y_test_pred = best_model.predict(x_test)

    train_accuracy = accuracy_score(y_train, y_train_pred)
    test_accuracy = accuracy_score(y_test, y_test_pred)

    train_precision = precision_score(y_train, y_train_pred, average='macro')
    test_precision = precision_score(y_test, y_test_pred, average='macro')

    train_recall = recall_score(y_train, y_train_pred, average='macro')
    test_recall = recall_score(y_test, y_test_pred, average='macro')

    train_f1 = f1_score(y_train, y_train_pred, average='macro')
    test_f1 = f1_score(y_test, y_test_pred, average='macro')

    print("\nBest Model: ", model_name.__class__.__name__)
    print("Best Parameters: ", grid_search.best_params_)
    print("Classification Report for Training Set:")
    print(classification_report(y_train, y_train_pred))

    print("Classification Report for Test Set:")
    print(classification_report(y_test, y_test_pred))

    # Confusion Matrix for Training Set
    train_cm = confusion_matrix(y_train, y_train_pred)
    plt.figure(figsize=(8, 5))
    sns.heatmap(train_cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title('Confusion Matrix - Training Set')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # Confusion Matrix for Test Set
    test_cm = confusion_matrix(y_test, y_test_pred)
    plt.figure(figsize=(8, 5))
    sns.heatmap(test_cm, annot=True, fmt='d', cmap='Purples', cbar=False)
    plt.title('Confusion Matrix - Test Set')
    plt.xlabel('Predicted')
```

```
    plt.ylabel('Actual')
    plt.show()



    # Store results in the DataFrame
    results = {
        'Model': model_name.__class__.__name__,
        'Scaling Method': scaler_name if scaler else "None",
        'Train Accuracy': train_accuracy,
        'Test Accuracy': test_accuracy,
        'Train Precision': train_precision,
        'Test Precision': test_precision,
        'Train Recall': train_recall,
        'Test Recall': test_recall,
        'Train F1-Score': train_f1,
        'Test F1-Score': test_f1
    }

    final_best_models = pd.concat([final_best_models, pd.DataFrame([results])],␣
↪ignore_index=True)
```

0.12 KNN :

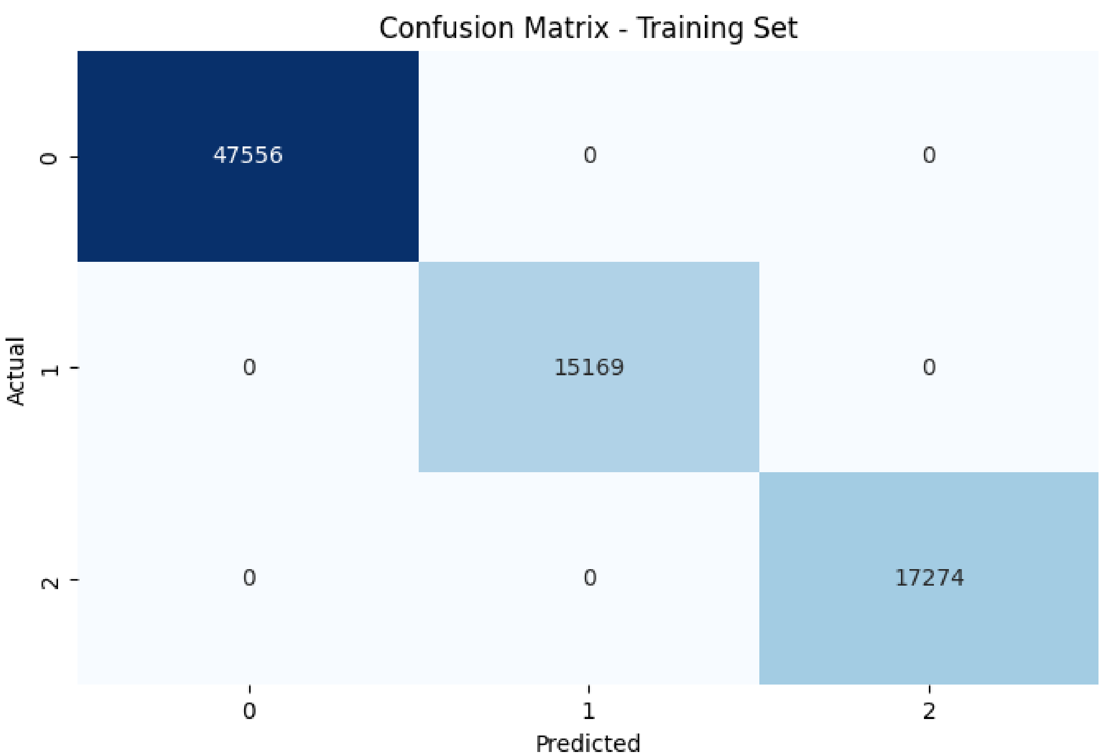
Standardization :

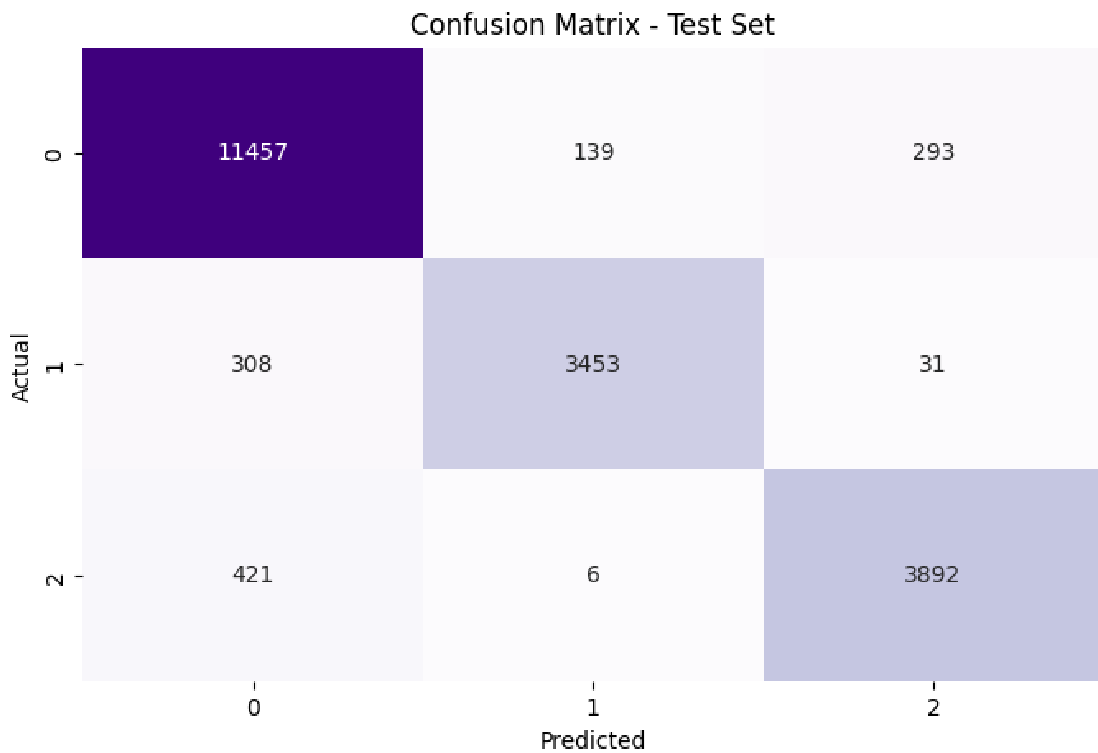
[27]:

```
Model_Evaluation_Pipline(
    x, y,
    KNeighborsClassifier(),
    scaler_name='StandardScaler',
    param_grid={
        'classifier__n_neighbors': [8,9],
        'classifier__weights': ['distance'],
    }
)
```

```
Best Model:  KNeighborsClassifier
Best Parameters:  {'classifier__n_neighbors': 8, 'classifier__weights':
'distance'}
Classification Report for Training Set:
              precision    recall  f1-score   support

      GALAXY       1.00      1.00      1.00     47556
         QSO       1.00      1.00      1.00     15169
        STAR       1.00      1.00      1.00     17274

    accuracy                           1.00     79999
```

```
   macro avg       1.00      1.00      1.00     79999
weighted avg       1.00      1.00      1.00     79999

Classification Report for Test Set:
              precision    recall  f1-score   support

      GALAXY       0.94      0.96      0.95     11889
         QSO       0.96      0.91      0.93      3792
        STAR       0.92      0.90      0.91      4319

    accuracy                           0.94     20000
   macro avg       0.94      0.93      0.93     20000
weighted avg       0.94      0.94      0.94     20000
```





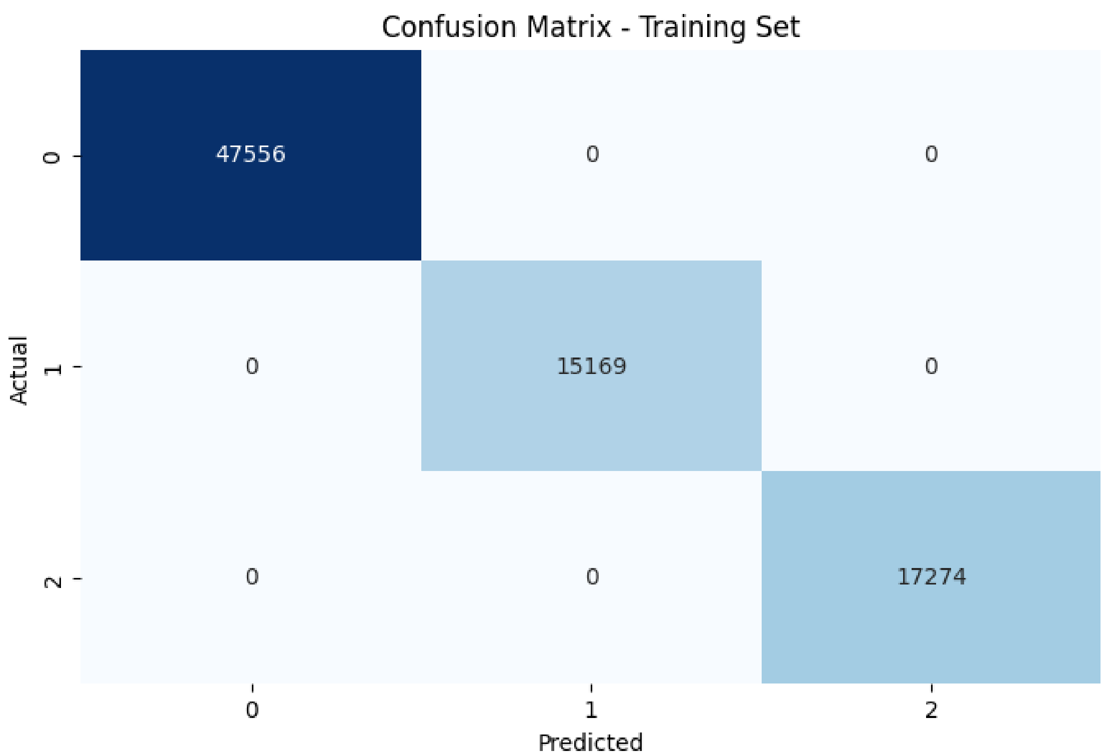

```
C:\Users\HP\AppData\Local\Temp\ipykernel_10072\2590765656.py:87: FutureWarning:

The behavior of DataFrame concatenation with empty or all-NA entries is
deprecated. In a future version, this will no longer exclude empty or all-NA
columns when determining the result dtypes. To retain the old behavior, exclude
the relevant entries before the concat operation.
```

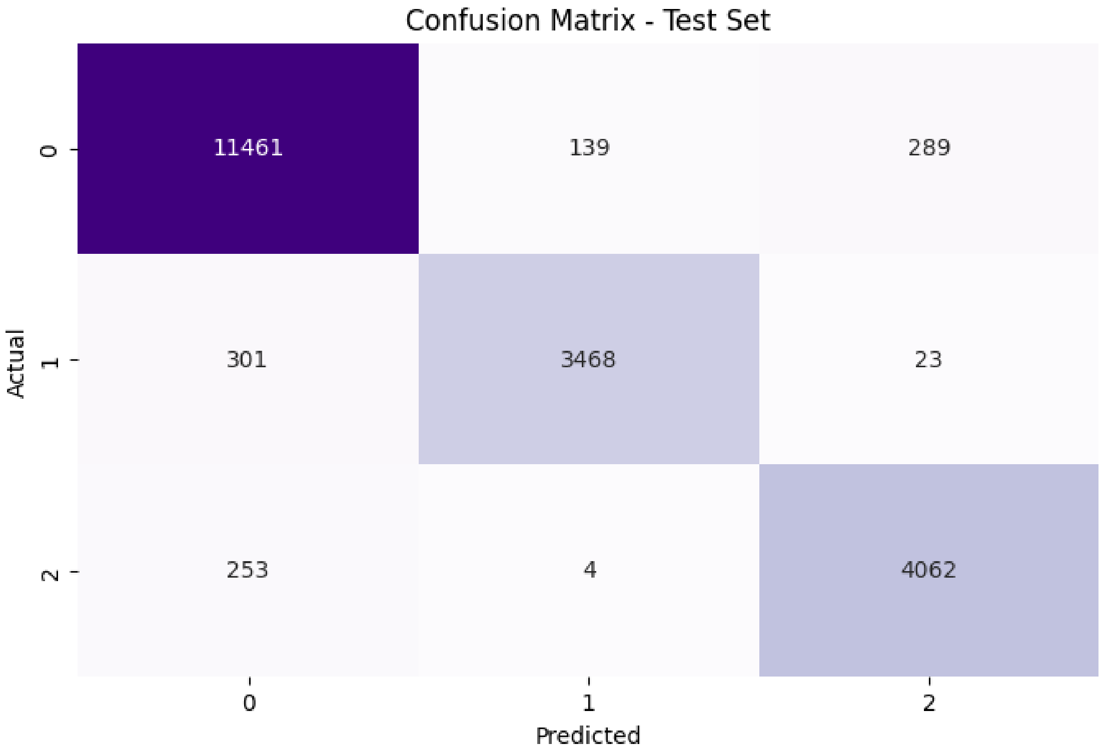
Robust :

[28]:
```
Model_Evaluation_Pipline(
    x, y,
    KNeighborsClassifier(),
    scaler_name='RobustScaler',
    param_grid={
        'classifier__n_neighbors': [8,9],
        'classifier__weights': ['distance'],
    }
)
```

```
Best Model:  KNeighborsClassifier
Best Parameters:  {'classifier__n_neighbors': 8, 'classifier__weights':
'distance'}
```

```
Classification Report for Training Set:
              precision    recall  f1-score   support

      GALAXY       1.00      1.00      1.00     47556
         QSO       1.00      1.00      1.00     15169
        STAR       1.00      1.00      1.00     17274

    accuracy                           1.00     79999
   macro avg       1.00      1.00      1.00     79999
weighted avg       1.00      1.00      1.00     79999

Classification Report for Test Set:
              precision    recall  f1-score   support

      GALAXY       0.95      0.96      0.96     11889
         QSO       0.96      0.91      0.94      3792
        STAR       0.93      0.94      0.93      4319

    accuracy                           0.95     20000
   macro avg       0.95      0.94      0.94     20000
weighted avg       0.95      0.95      0.95     20000
```





0.13 Random Forest :

Without Scaling :

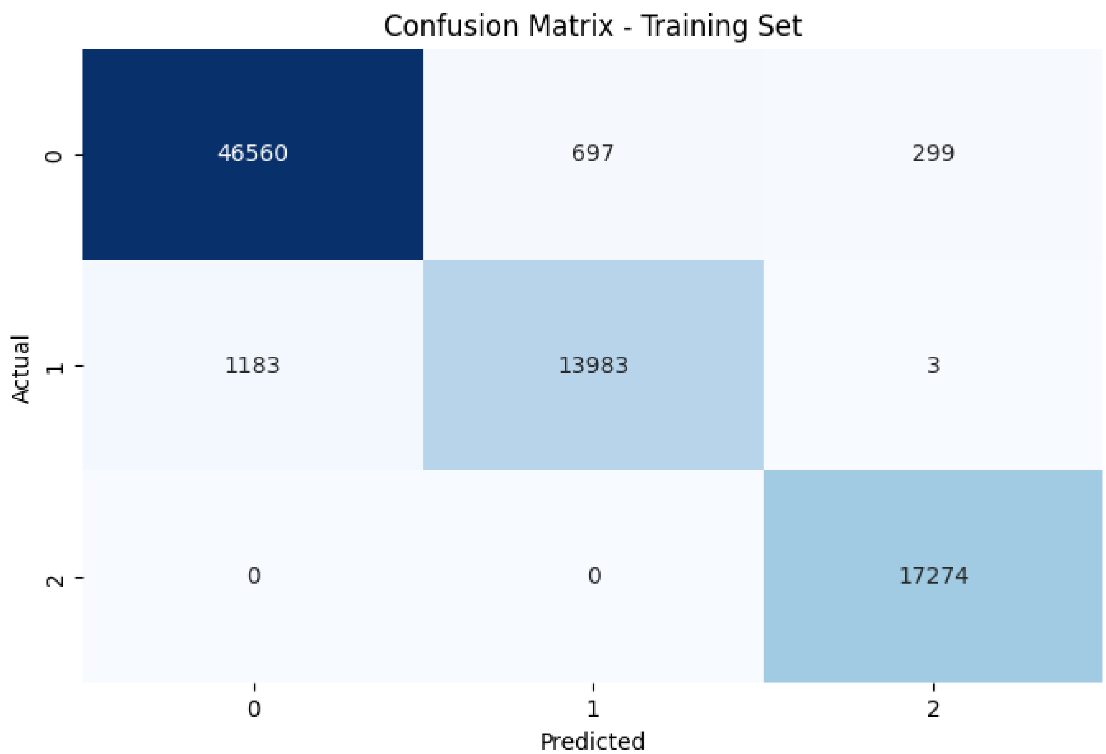
[29]:
```
Model_Evaluation_Pipline(
    x, y,
    RandomForestClassifier(),
    scaler_name='None',
    param_grid={
        'classifier__n_estimators': [50, 70],
        'classifier__max_depth': [4, 5,6],
        'classifier__min_samples_split': [2, 4]
    }
)
```

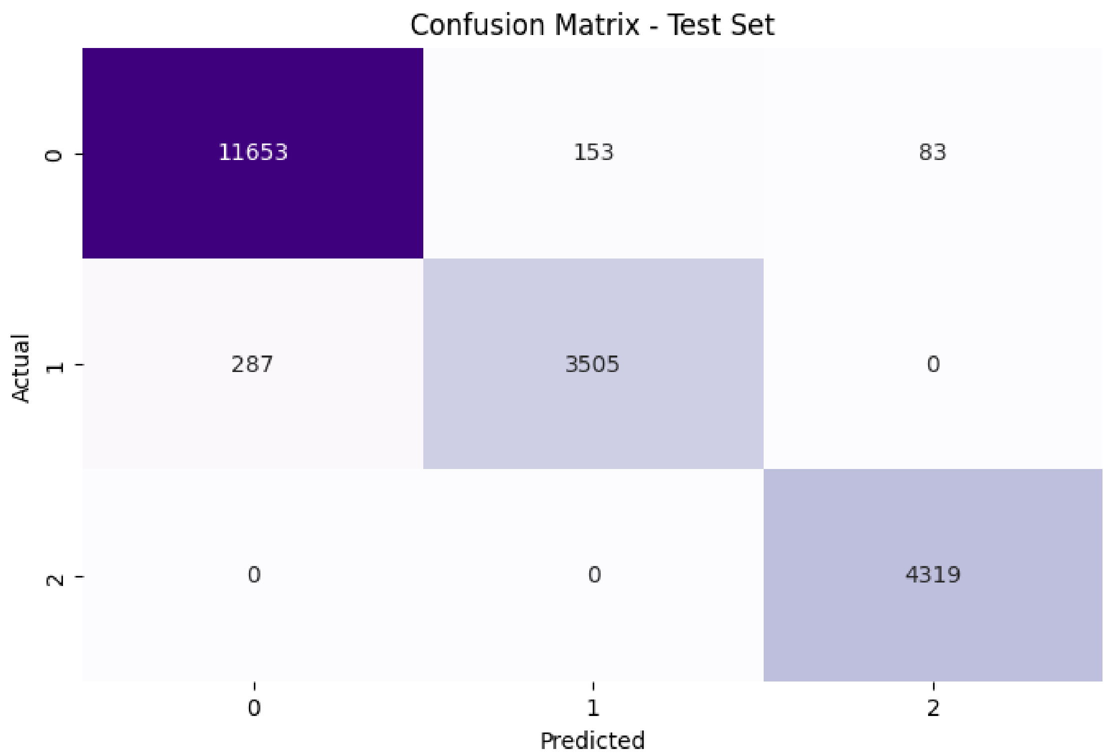
```
Best Model:  RandomForestClassifier
Best Parameters:  {'classifier__max_depth': 6, 'classifier__min_samples_split':
2, 'classifier__n_estimators': 50}
Classification Report for Training Set:
              precision    recall  f1-score   support

      GALAXY       0.98      0.98      0.98     47556
         QSO       0.95      0.92      0.94     15169
```

	precision	recall	f1-score	support
STAR	0.98	1.00	0.99	17274
accuracy			0.97	79999
macro avg	0.97	0.97	0.97	79999
weighted avg	0.97	0.97	0.97	79999

Classification Report for Test Set:

	precision	recall	f1-score	support
GALAXY	0.98	0.98	0.98	11889
QSO	0.96	0.92	0.94	3792
STAR	0.98	1.00	0.99	4319
accuracy			0.97	20000
macro avg	0.97	0.97	0.97	20000
weighted avg	0.97	0.97	0.97	20000



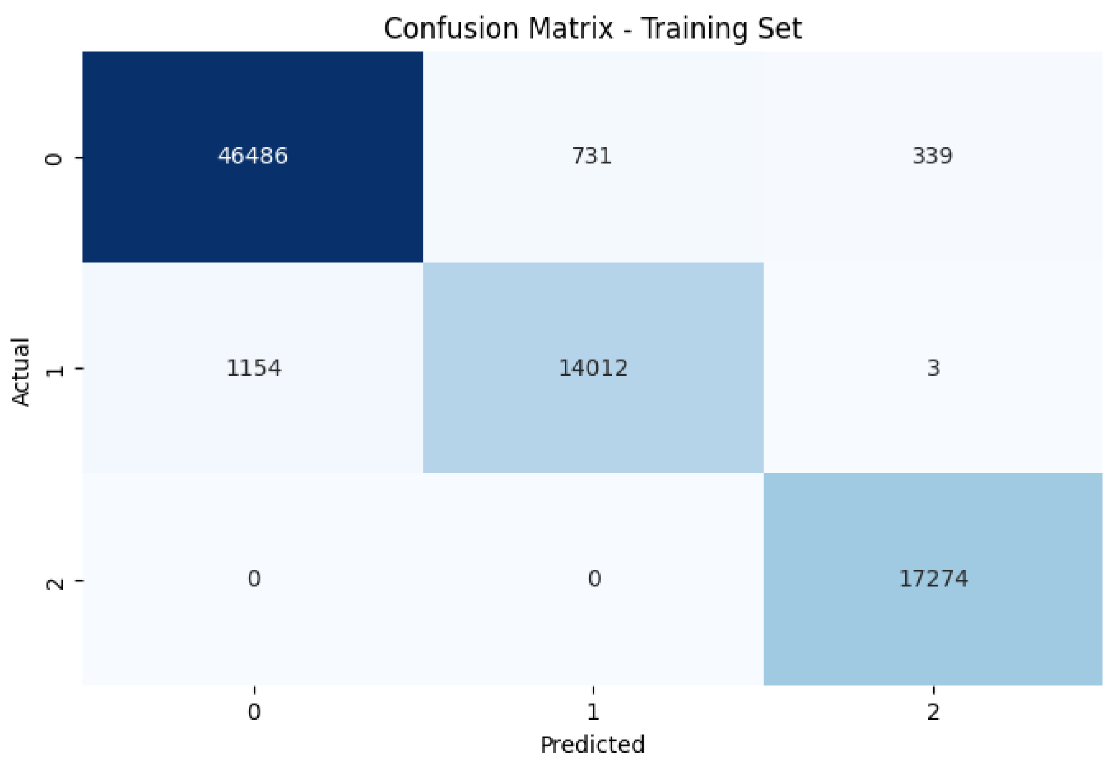


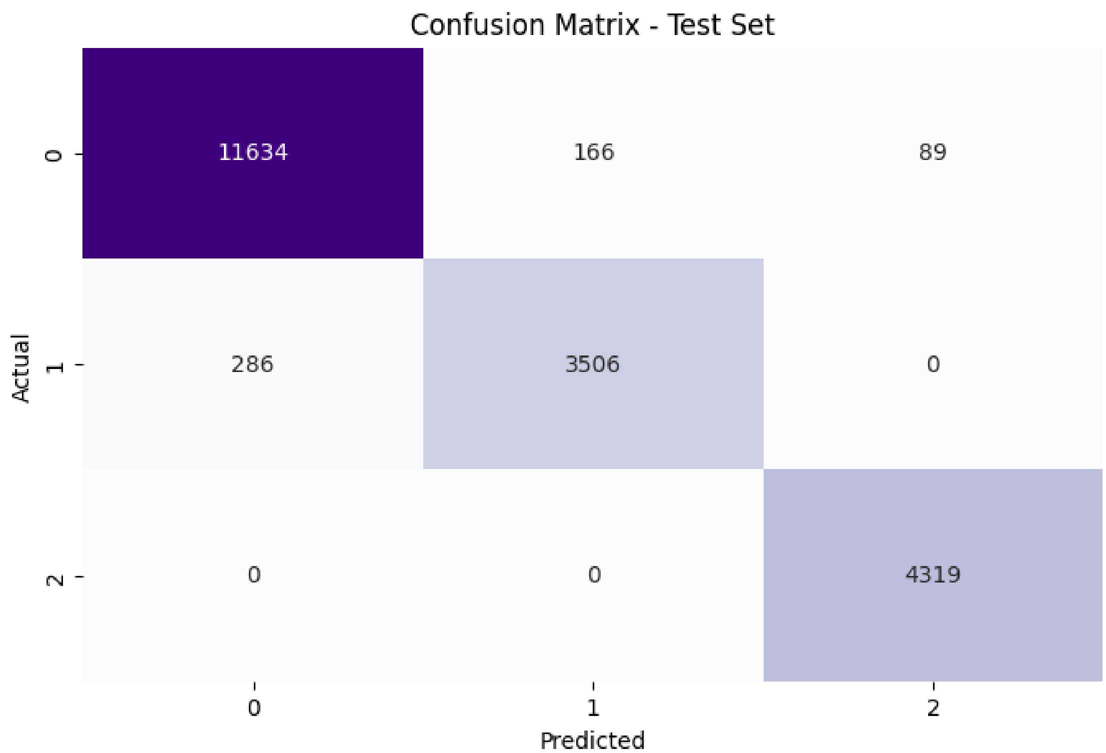
Standardization :

[30]:
```
Model_Evaluation_Pipline(
    x, y,
    RandomForestClassifier(),
    scaler_name='StandardScaler',
    param_grid={
        'classifier__n_estimators': [50, 60],
        'classifier__max_depth': [5,6],
        'classifier__min_samples_split': [2, 3]
    }
)
```

```
Best Model:  RandomForestClassifier
Best Parameters:  {'classifier__max_depth': 6, 'classifier__min_samples_split':
2, 'classifier__n_estimators': 50}
Classification Report for Training Set:
              precision    recall  f1-score   support

      GALAXY       0.98      0.98      0.98     47556
         QSO       0.95      0.92      0.94     15169
        STAR       0.98      1.00      0.99     17274
```

```
    accuracy                           0.97     79999
   macro avg       0.97      0.97      0.97     79999
weighted avg       0.97      0.97      0.97     79999

Classification Report for Test Set:
              precision    recall  f1-score   support

      GALAXY       0.98      0.98      0.98     11889
         QSO       0.95      0.92      0.94      3792
        STAR       0.98      1.00      0.99      4319

    accuracy                           0.97     20000
   macro avg       0.97      0.97      0.97     20000
weighted avg       0.97      0.97      0.97     20000
```



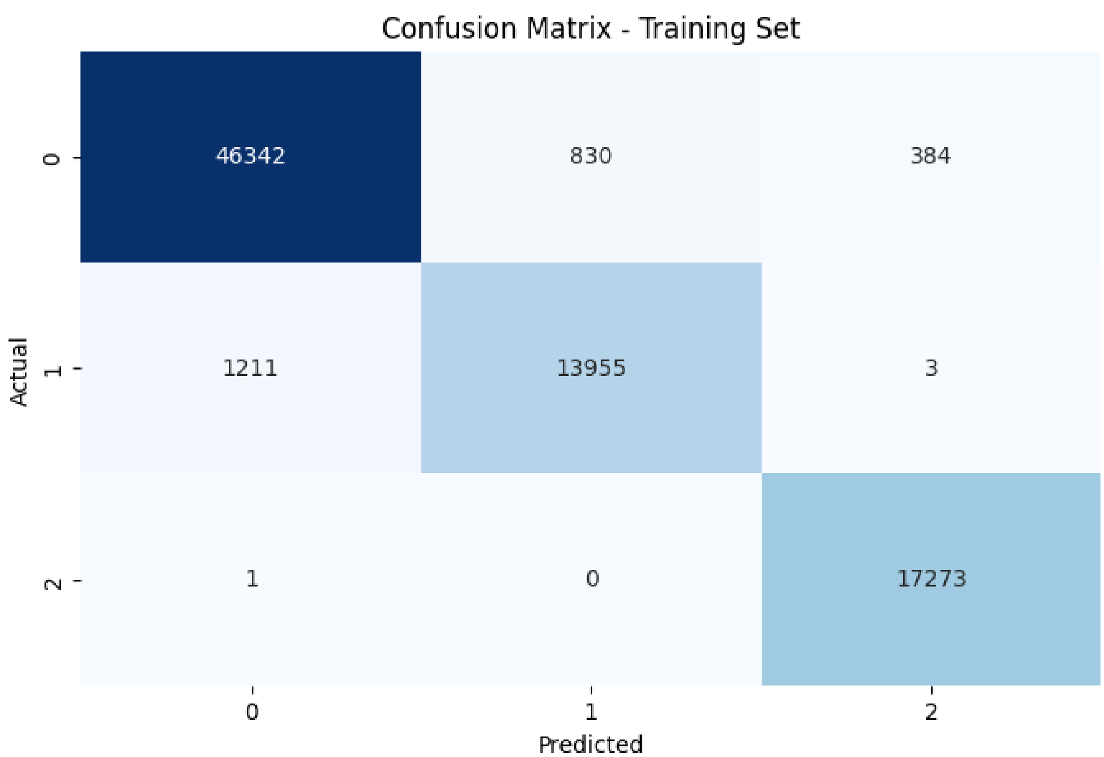


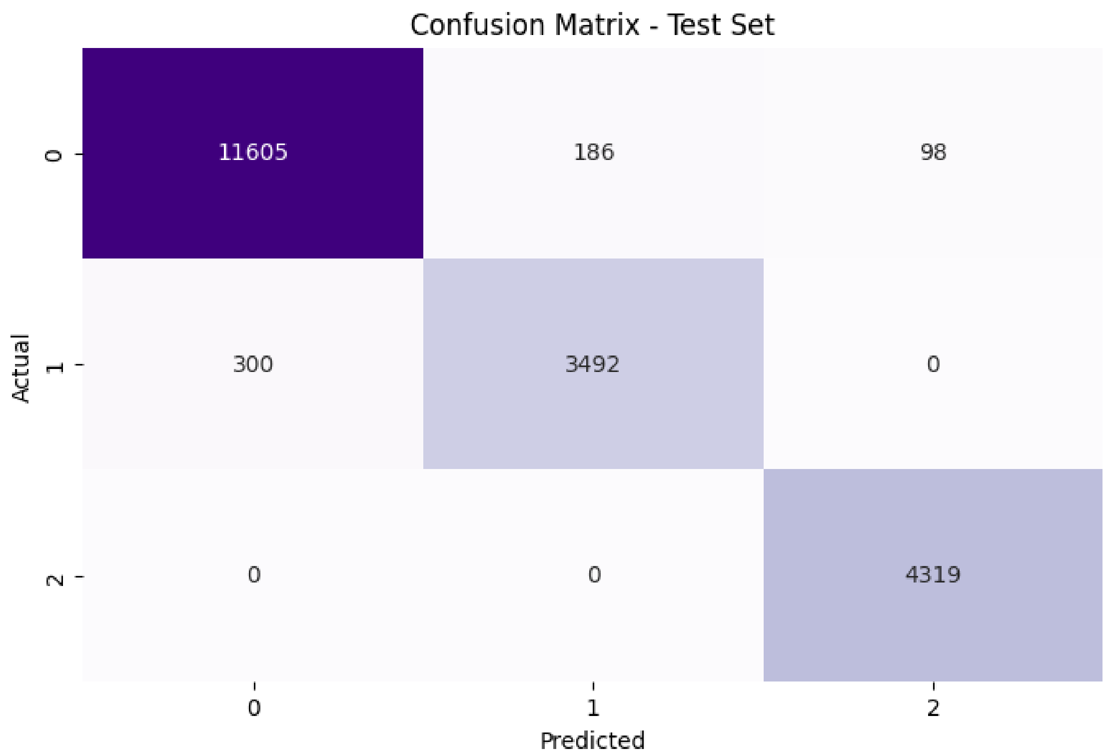
Robust :

[31]:
```
Model_Evaluation_Pipline(
    x, y,
    RandomForestClassifier(),
    scaler_name='RobustScaler',
    param_grid={
        'classifier__n_estimators': [50, 60],
        'classifier__max_depth': [5,6],
        'classifier__min_samples_split': [2, 3]
    }
)
```

```
Best Model:  RandomForestClassifier
Best Parameters:  {'classifier__max_depth': 6, 'classifier__min_samples_split':
2, 'classifier__n_estimators': 50}
Classification Report for Training Set:
              precision    recall  f1-score   support

      GALAXY       0.97      0.97      0.97     47556
         QSO       0.94      0.92      0.93     15169
        STAR       0.98      1.00      0.99     17274
```

```
    accuracy                           0.97     79999
   macro avg       0.97      0.96      0.97     79999
weighted avg       0.97      0.97      0.97     79999

Classification Report for Test Set:
              precision    recall  f1-score   support

      GALAXY       0.97      0.98      0.98     11889
         QSO       0.95      0.92      0.93      3792
        STAR       0.98      1.00      0.99      4319

    accuracy                           0.97     20000
   macro avg       0.97      0.97      0.97     20000
weighted avg       0.97      0.97      0.97     20000
```





0.14 Comparison Between Models :

[33]:
```
final_best_models
```

[33]:

	Model	Scaling Method	Train Accuracy	Test Accuracy
0	KNeighborsClassifier	StandardScaler	1.000000	0.94010
1	KNeighborsClassifier	RobustScaler	1.000000	0.94955
2	RandomForestClassifier	None	0.972725	0.97385
3	RandomForestClassifier	StandardScaler	0.972162	0.97295
4	RandomForestClassifier	RobustScaler	0.969637	0.97080

	Train Precision	Test Precision	Train Recall	Test Recall	Train F1-Score
0	1.000000	0.941009	1.000000	0.925133	1.000000
1	1.000000	0.947653	1.000000	0.939684	1.000000
2	0.970186	0.971761	0.966957	0.968155	0.968462
3	0.968927	0.970203	0.967075	0.967710	0.967906
4	0.965487	0.967348	0.964794	0.965666	0.965049

	Test F1-Score
0	0.932764
1	0.943461
2	0.969825

```
3        0.968841
4        0.966393
```

0.15 Applying SMOTE :

[34]:
```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(x, y)
```

0.16 Before and After SMOTE :

[35]:
```
# Define a darker color palette for consistency
dark_palette = ['#012169', '#224C98', '#6A0DAD', '#9B30FF']

# Original class distribution
original_counts = Counter(y)
original_labels = list(original_counts.keys())
original_values = list(original_counts.values())

# Resampled class distribution (after applying SMOTE)
resampled_counts = Counter(y_resampled)
resampled_labels = list(resampled_counts.keys())
resampled_values = list(resampled_counts.values())

# Create the pie chart for the original class distribution
fig_original = px.pie(
    names=original_labels,
    values=original_values,
    title='Class Distribution Before SMOTE',
    color_discrete_sequence=dark_palette
)
fig_original.update_traces(textinfo='percent+label',␣
 ↪textfont=dict(color='white'))
fig_original.update_layout(title_font=dict(color='#13274F'), margin=dict(l=40,␣
 ↪r=40, t=40, b=40))

# Create the pie chart for the resampled class distribution
fig_resampled = px.pie(
    names=resampled_labels,
    values=resampled_values,
    title='Class Distribution After SMOTE',
    color_discrete_sequence=dark_palette
)
fig_resampled.update_traces(textinfo='percent+label',␣
 ↪textfont=dict(color='white'))
fig_resampled.update_layout(title_font=dict(color='#13274F'), margin=dict(l=40,␣
 ↪r=40, t=40, b=40))
```

```
# Show the figures
fig_original.show()
fig_resampled.show()
```

0.17 Modelling with SMOTE :

[36]:
```
final_best_models_smote = pd.DataFrame(columns=["Model", "Scaling Method",␣
↪"Best Params",
                                                "Train Accuracy", "Test␣
↪Accuracy",
                                                "Train Precision", "Test␣
↪Precision",
                                                "Train Recall", "Test Recall",
                                                "Train F1-Score", "Test␣
↪F1-Score"])

def Model_Evaluation_Pipeline_Smote(x, y, model, scaler_name=None,␣
↪param_grid=None,
                                    use_randomized_search=False, n_iter=10):
    global final_best_models_smote

    # 1. Apply SMOTE
    smote = SMOTE(random_state=42)
    X_resampled, y_resampled = smote.fit_resample(x, y)

    # 2. Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X_resampled,␣
↪y_resampled, test_size=0.2,
                                                        random_state=42,␣
↪stratify=y_resampled)

    # 3. Set up the scaler
    if scaler_name == 'StandardScaler':
        scaler = StandardScaler()
    elif scaler_name == 'MinMaxScaler':
        scaler = MinMaxScaler()
    elif scaler_name == 'RobustScaler':
        scaler = RobustScaler()
    else:
        scaler = None

    # 4. Set up the Pipeline
    steps = []
    if scaler:
        steps.append(('scaler', scaler))
```

```
    steps.append(('classifier', model))
    pipeline = Pipeline(steps)

    # 5. Choose between GridSearchCV and RandomizedSearchCV
    if use_randomized_search:
        grid_search = RandomizedSearchCV(pipeline, param_grid, n_iter=n_iter,␣
↪cv=5,
                                         n_jobs=-1, scoring='f1_macro',␣
↪random_state=42)
    else:
        grid_search = GridSearchCV(pipeline, param_grid, cv=5, n_jobs=-1,␣
↪scoring='f1_macro')

    grid_search.fit(X_train, y_train)

    # 6. Extract the best model and evaluate
    best_model = grid_search.best_estimator_

    # Predict on training set
    y_train_pred = best_model.predict(X_train)
    train_accuracy = accuracy_score(y_train, y_train_pred)
    train_precision = precision_score(y_train, y_train_pred, average='macro')
    train_recall = recall_score(y_train, y_train_pred, average='macro')
    train_f1 = f1_score(y_train, y_train_pred, average='macro')

    # Predict on test set
    y_test_pred = best_model.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_precision = precision_score(y_test, y_test_pred, average='macro')
    test_recall = recall_score(y_test, y_test_pred, average='macro')
    test_f1 = f1_score(y_test, y_test_pred, average='macro')

    # 7. Display the reports
    print("\nBest Model: ", model.__class__.__name__)
    print("Best Parameters: ", grid_search.best_params_)

    print("\nClassification Report for Training Set:")
    print(classification_report(y_train, y_train_pred))

    print("\nClassification Report for Test Set:")
    print(classification_report(y_test, y_test_pred))

    # 8. Draw Confusion Matrices for both sets
    train_cm = confusion_matrix(y_train, y_train_pred)
    plt.figure(figsize=(8, 5))
    sns.heatmap(train_cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title('Confusion Matrix - Training Set')
```

```
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    test_cm = confusion_matrix(y_test, y_test_pred)
    plt.figure(figsize=(8, 5))
    sns.heatmap(test_cm, annot=True, fmt='d', cmap='Purples', cbar=False)
    plt.title('Confusion Matrix - Test Set')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # 9. Summarize results in the final_best_models_smote DataFrame
    results = {
        'Model': model.__class__.__name__,
        'Scaling Method': scaler_name if scaler else "None",
        'Best Params': grid_search.best_params_,
        'Train Accuracy': train_accuracy,
        'Test Accuracy': test_accuracy,
        'Train Precision': train_precision,
        'Test Precision': test_precision,
        'Train Recall': train_recall,
        'Test Recall': test_recall,
        'Train F1-Score': train_f1,
        'Test F1-Score': test_f1,
    }

    final_best_models_smote = pd.concat([final_best_models_smote, pd.
↪DataFrame([results])], ignore_index=True)
```

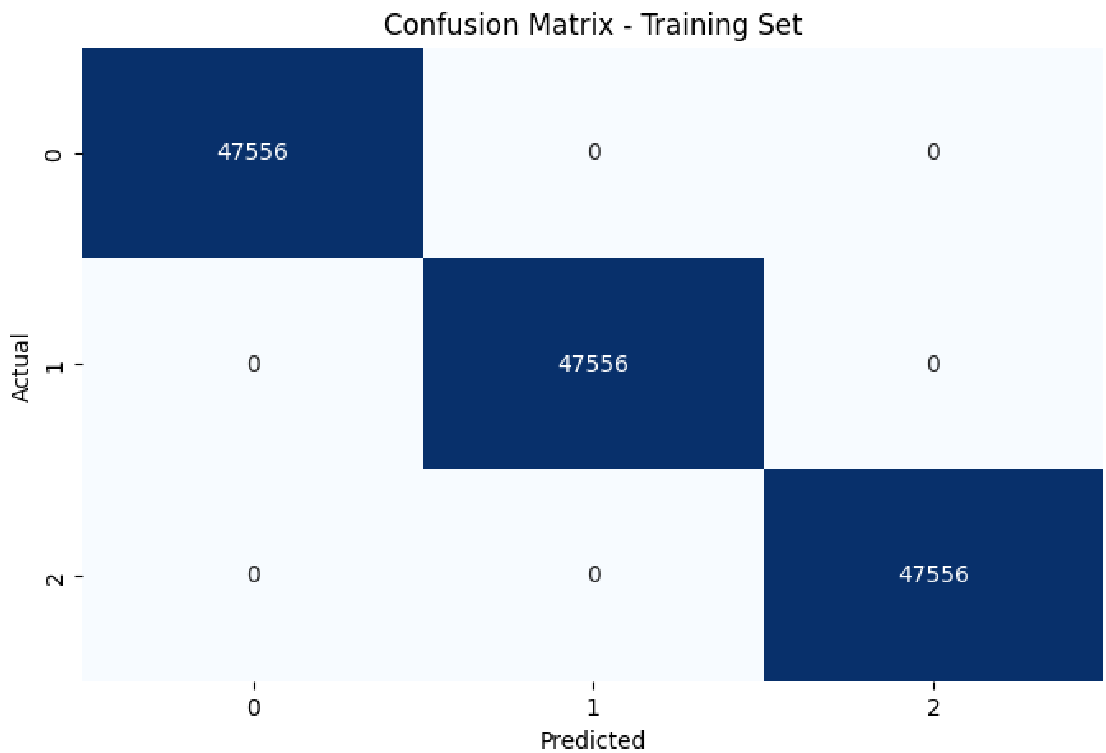
0.18 KNN With SMOTE:

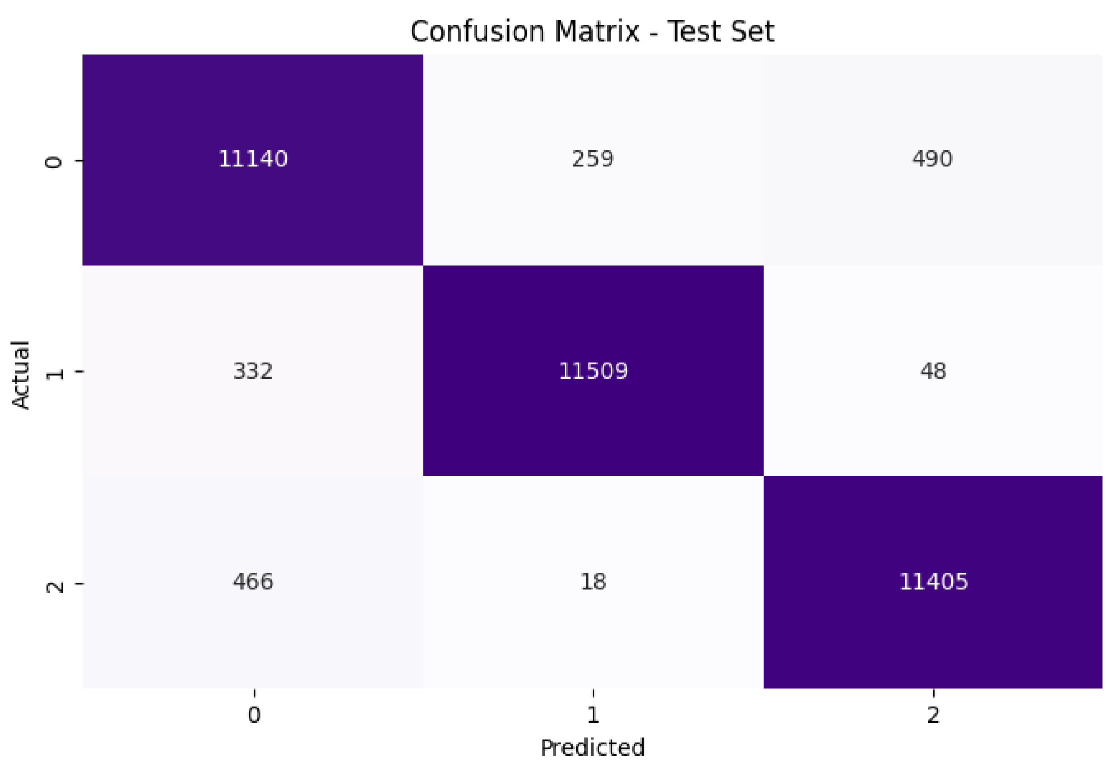
Standardization :

[38]:
```
n_iter=4
Model_Evaluation_Pipeline_Smote(
     X_resampled,
    y_resampled,
    KNeighborsClassifier(),
    scaler_name='StandardScaler',
    param_grid={
        'classifier__n_neighbors': [8,9],
        'classifier__weights': ['distance'],
    },use_randomized_search=True,  # Set this to True to use RandomizedSearchCV
    n_iter=n_iter
)
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-
```

```
packages\sklearn\model_selection\_search.py:320: UserWarning:

The total space of parameters 2 is smaller than n_iter=4. Running 2 iterations.
For exhaustive searches, use GridSearchCV.



Best Model:  KNeighborsClassifier
Best Parameters:  {'classifier__weights': 'distance', 'classifier__n_neighbors':
8}

Classification Report for Training Set:
              precision    recall  f1-score   support

      GALAXY       1.00      1.00      1.00     47556
         QSO       1.00      1.00      1.00     47556
        STAR       1.00      1.00      1.00     47556

    accuracy                           1.00    142668
   macro avg       1.00      1.00      1.00    142668
weighted avg       1.00      1.00      1.00    142668


Classification Report for Test Set:
              precision    recall  f1-score   support

      GALAXY       0.93      0.94      0.94     11889
         QSO       0.98      0.97      0.97     11889
        STAR       0.95      0.96      0.96     11889

    accuracy                           0.95     35667
   macro avg       0.95      0.95      0.95     35667
weighted avg       0.95      0.95      0.95     35667
```


Confusion Matrix - Training Set
47556
0
0
0
47556
0
0
0
47556
Actual
0
1
2
0
1
2
Predicted


Confusion Matrix - Test Set
11140
259
490
332
11509
48
466
18
11405
Actual
0
1
2
0
1
2
Predicted

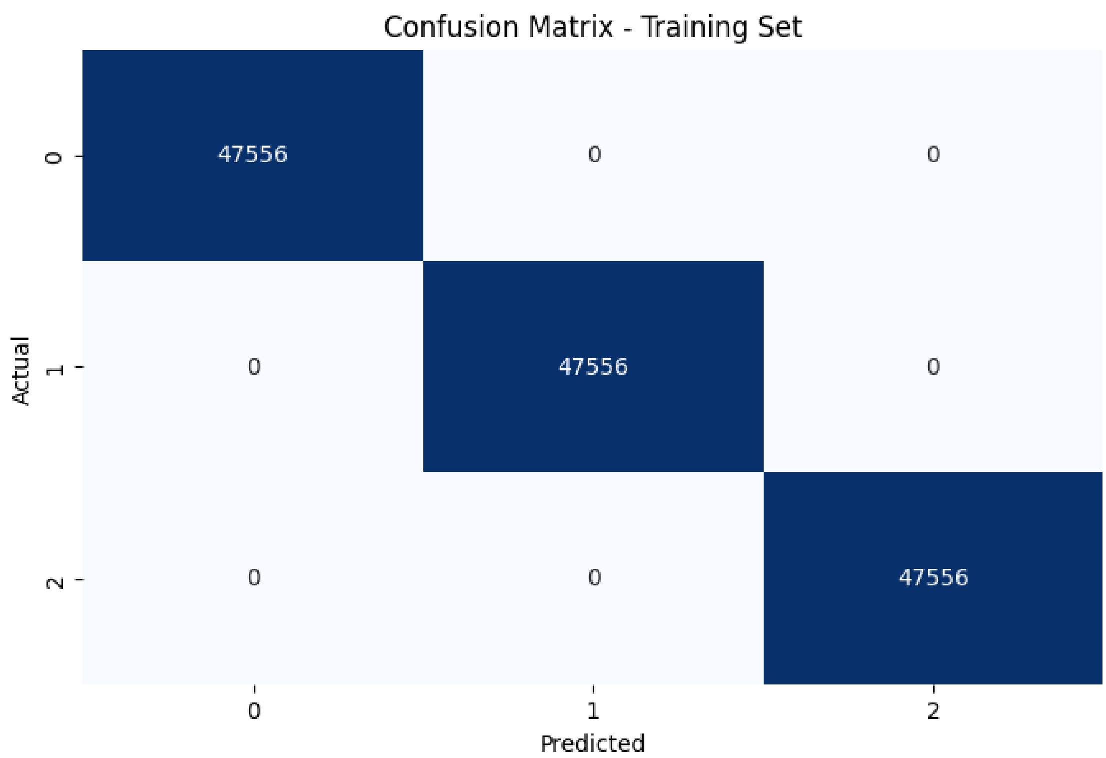
```
C:\Users\HP\AppData\Local\Temp\ipykernel_10072\1793332777.py:104: FutureWarning:

The behavior of DataFrame concatenation with empty or all-NA entries is
deprecated. In a future version, this will no longer exclude empty or all-NA
columns when determining the result dtypes. To retain the old behavior, exclude
the relevant entries before the concat operation.
```

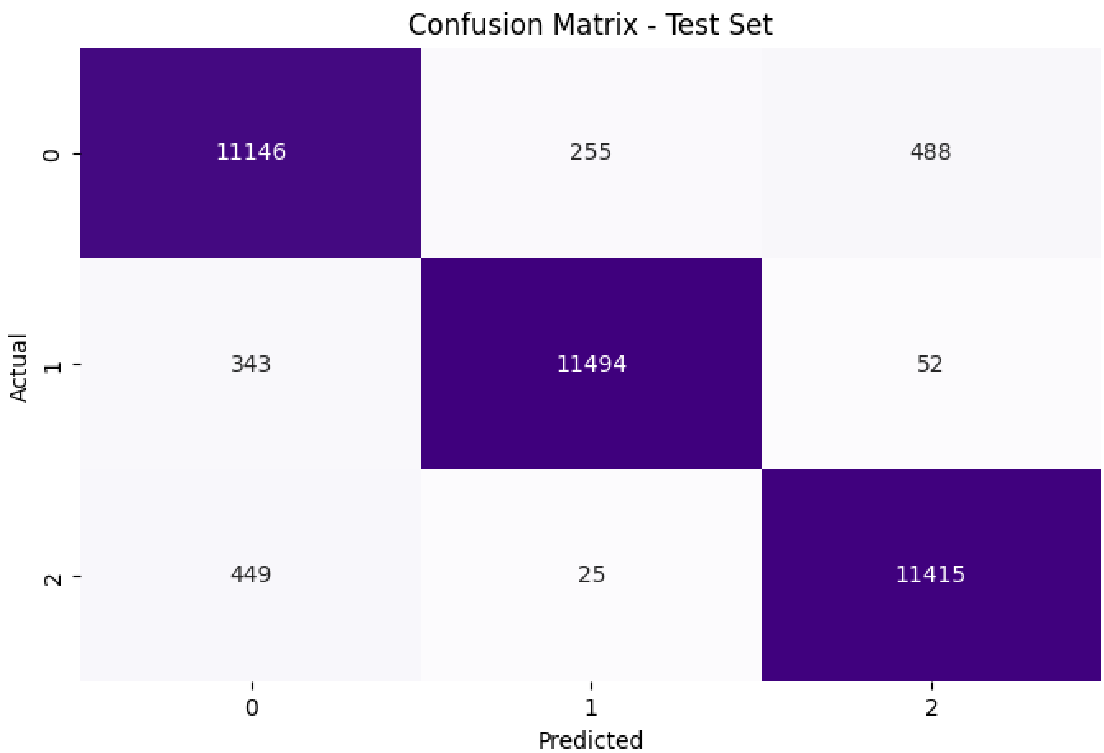
Robust :

[39]:
```python
Model_Evaluation_Pipeline_Smote(
     X_resampled,
    y_resampled,
    KNeighborsClassifier(),
    scaler_name='RobustScaler',
    param_grid={
        'classifier__n_neighbors': [8,9],
        'classifier__weights': ['distance'],
    },use_randomized_search=True,  # Set this to True to use RandomizedSearchCV
    n_iter=n_iter
)
```

```
C:\Users\HP\AppData\Local\Programs\Python\Python312\Lib\site-
packages\sklearn\model_selection\_search.py:320: UserWarning:

The total space of parameters 2 is smaller than n_iter=4. Running 2 iterations.
For exhaustive searches, use GridSearchCV.



Best Model:  KNeighborsClassifier
Best Parameters:  {'classifier__weights': 'distance', 'classifier__n_neighbors':
8}

Classification Report for Training Set:
              precision    recall  f1-score   support

      GALAXY       1.00      1.00      1.00     47556
         QSO       1.00      1.00      1.00     47556
        STAR       1.00      1.00      1.00     47556

    accuracy                           1.00    142668
   macro avg       1.00      1.00      1.00    142668
weighted avg       1.00      1.00      1.00    142668


Classification Report for Test Set:
```

	precision	recall	f1-score	support
GALAXY	0.93	0.94	0.94	11889
QSO	0.98	0.97	0.97	11889
STAR	0.95	0.96	0.96	11889
accuracy			0.95	35667
macro avg	0.95	0.95	0.95	35667
weighted avg	0.95	0.95	0.95	35667





0.19 Random Forest With SMOTE :

Without Scaling :

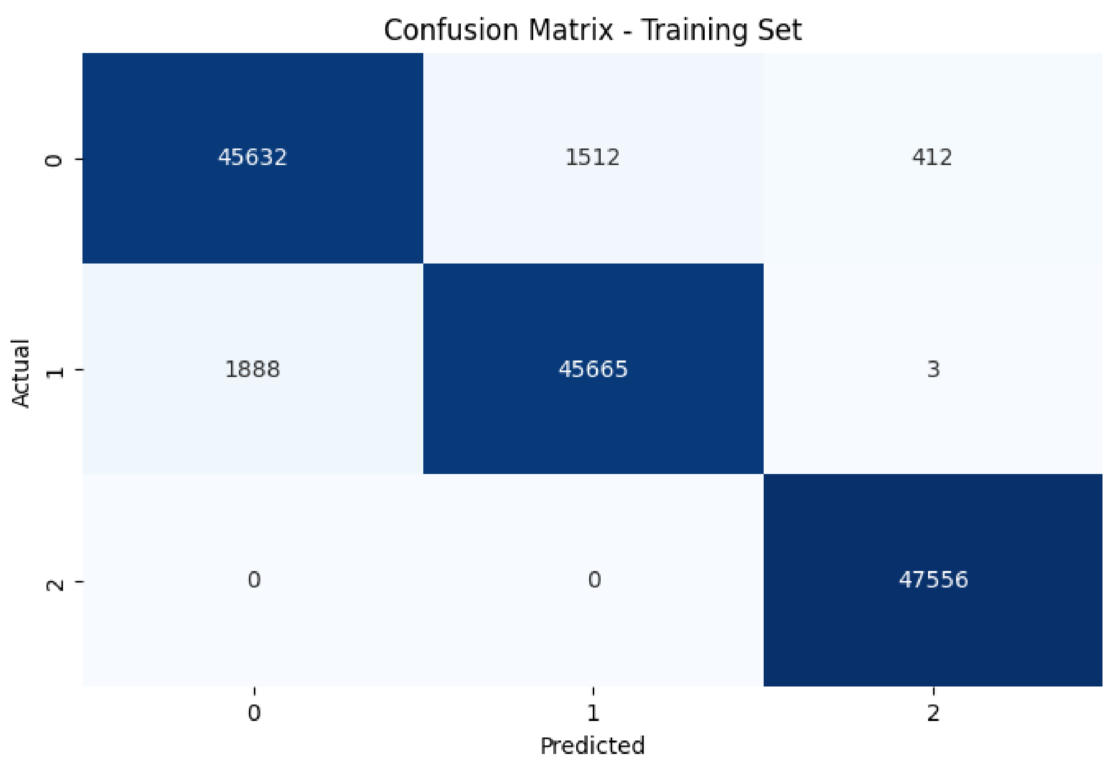
[40]:
```
n_iter=4
Model_Evaluation_Pipeline_Smote(
     X_resampled,
    y_resampled,
    RandomForestClassifier(),
    scaler_name='None',
    param_grid={
        'classifier__n_estimators': [50, 70],
        'classifier__max_depth': [4, 5,6],
        'classifier__min_samples_split': [2, 4]
    },use_randomized_search=True,  # Set this to True to use RandomizedSearchCV
    n_iter=n_iter
)
```

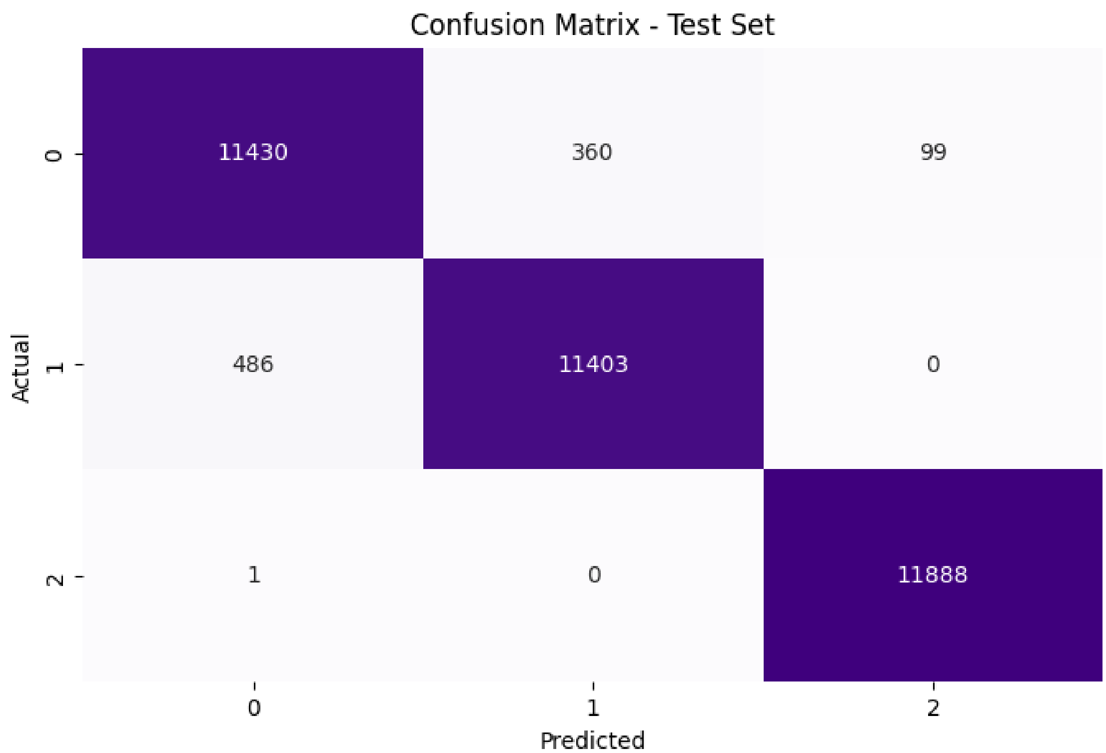
```
Best Model:  RandomForestClassifier
Best Parameters:  {'classifier__n_estimators': 70,
'classifier__min_samples_split': 2, 'classifier__max_depth': 6}

Classification Report for Training Set:
```

	precision	recall	f1-score	support
GALAXY	0.96	0.96	0.96	47556
QSO	0.97	0.96	0.96	47556
STAR	0.99	1.00	1.00	47556
accuracy			0.97	142668
macro avg	0.97	0.97	0.97	142668
weighted avg	0.97	0.97	0.97	142668

Classification Report for Test Set:

	precision	recall	f1-score	support
GALAXY	0.96	0.96	0.96	11889
QSO	0.97	0.96	0.96	11889
STAR	0.99	1.00	1.00	11889
accuracy			0.97	35667
macro avg	0.97	0.97	0.97	35667
weighted avg	0.97	0.97	0.97	35667





Standardization :

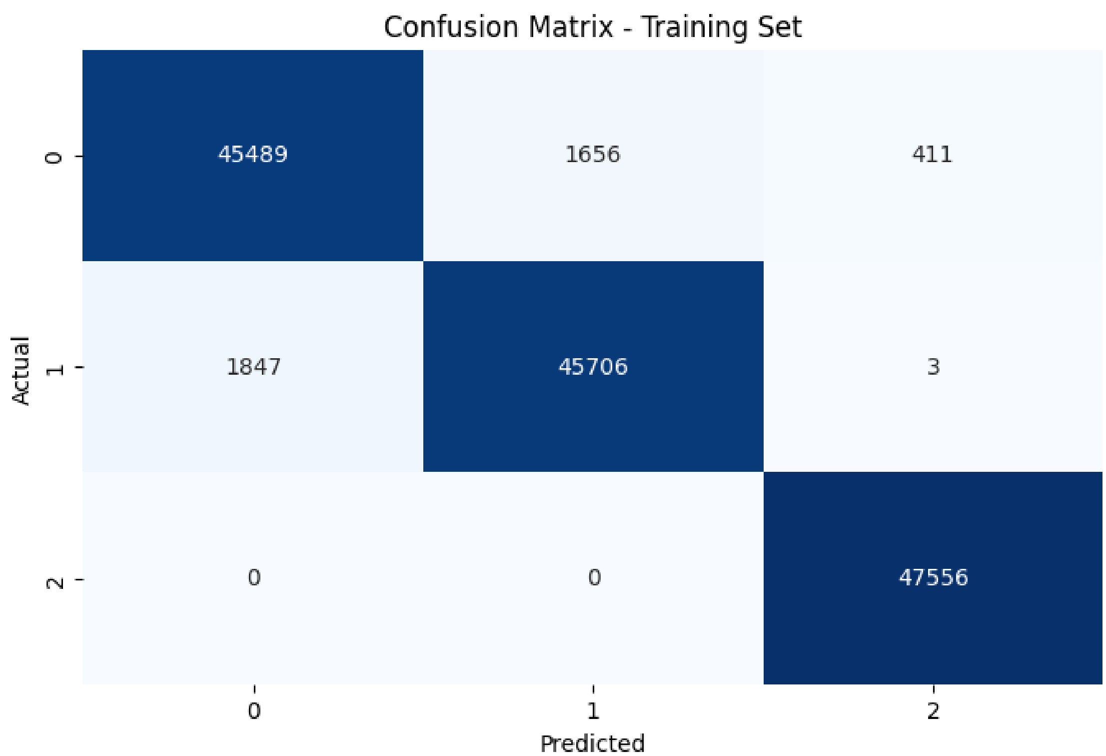
[41]:
```
n_iter=4
Model_Evaluation_Pipeline_Smote(
    X_resampled,
    y_resampled,
    RandomForestClassifier(),
    scaler_name='StandardScaler',
    param_grid={
        'classifier__n_estimators': [50, 60],
        'classifier__max_depth': [5,6],
        'classifier__min_samples_split': [2, 3]
    },use_randomized_search=True,  # Set this to True to use RandomizedSearchCV
    n_iter=n_iter
)
```

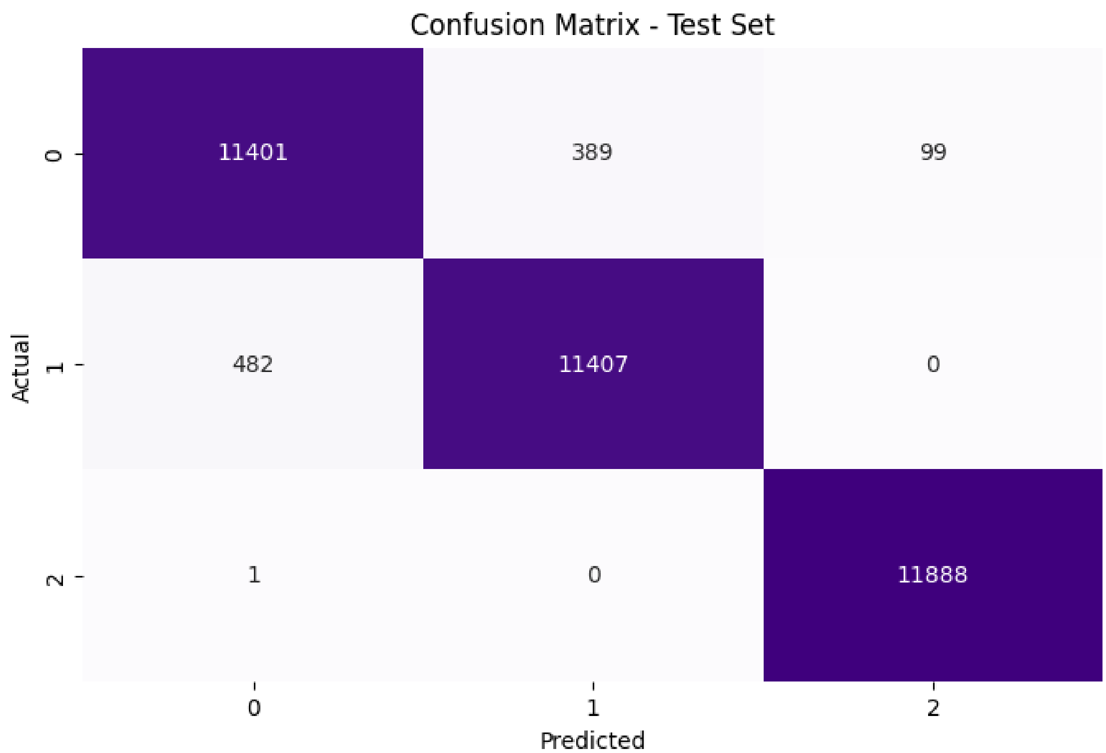
```
Best Model:  RandomForestClassifier
Best Parameters:  {'classifier__n_estimators': 60,
'classifier__min_samples_split': 3, 'classifier__max_depth': 6}

Classification Report for Training Set:
              precision    recall  f1-score   support
```

	precision	recall	f1-score	support
GALAXY	0.96	0.96	0.96	47556
QSO	0.97	0.96	0.96	47556
STAR	0.99	1.00	1.00	47556
accuracy			0.97	142668
macro avg	0.97	0.97	0.97	142668
weighted avg	0.97	0.97	0.97	142668

Classification Report for Test Set:

	precision	recall	f1-score	support
GALAXY	0.96	0.96	0.96	11889
QSO	0.97	0.96	0.96	11889
STAR	0.99	1.00	1.00	11889
accuracy			0.97	35667
macro avg	0.97	0.97	0.97	35667
weighted avg	0.97	0.97	0.97	35667





Robust :

[ ]:
```
n_iter=4
Model_Evaluation_Pipeline_Smote(
    X_resampled,
    y_resampled,
    RandomForestClassifier(),
    scaler_name='RobustScaler',
    param_grid={
        'classifier__n_estimators': [50, 60],
        'classifier__max_depth': [5,6],
        'classifier__min_samples_split': [2, 3]
    },use_randomized_search=True,  # Set this to True to use RandomizedSearchCV
    n_iter=n_iter
)
```

0.20 Comparison between Models :

[42]:
```
final_best_models_smote.drop("Best Params",axis=1)
```

[42]:
```
                    Model  Scaling Method  Train Accuracy  Test Accuracy  \
0    KNeighborsClassifier  StandardScaler        1.000000       0.954776
1    KNeighborsClassifier    RobustScaler        1.000000       0.954804
```

```
2  RandomForestClassifier            None        0.973260       0.973477
3  RandomForestClassifier  StandardScaler        0.972545       0.972776

   Train Precision  Test Precision  Train Recall  Test Recall  Train F1-Score  \
0         1.000000        0.954868      1.000000     0.954776        1.000000
1         1.000000        0.954902      1.000000     0.954804        1.000000
2         0.973190        0.973424      0.973260     0.973477        0.973213
3         0.972462        0.972707      0.972545     0.972776        0.972494

   Test F1-Score
0       0.954813
1       0.954842
2       0.973435
3       0.972731
```

0.21 Final Model :

[ ]:

0.22 Conclusion :

We studied the challenges and methods for categorizing stars using the Stellar Classification Dataset from the Sloan Digital Sky Survey (SDSS17). It is important to accurately classify celestial objects to improve our understanding of the universe.

We used machine learning techniques to automate the classification process and address issues like data imbalance and multicollinearity.

Our analysis provided valuable insights into feature importance and data quality, guiding our model development approach.

The best-performing models were XGBoost classifier and Random Forest, showing their effectiveness in handling the classification task. Moving forward, we will continue refining our model to ensure strong performance in identifying and categorizing stars.

This work not only contributes to astronomical research but also sets the stage for future advancements in the automated analysis of astronomical data.

0.23 Application of the Stellar Classification Project :

Astronomical Research: Enhances understanding of stellar evolution and galaxy formation.

Exoplanet Research: Helps identify host stars and analyze stellar spectra for exoplanet detection.

Educational Tools: Serves as teaching resources and supports interactive applications for public engagement.

Automated Surveys: Can be integrated into telescopes for real-time classification and streamline data processing in large surveys.

Citizen Science: Encourages public participation in star classification, fostering interest in astronomy.

AI in Astronomy: Sets the stage for applying AI techniques in astronomical research.

0.24 Thank You :)

[ ]:

```

```

[ ]:

```

```