IMPORT LIBRARIES

```
!pip install imblearn
```

Show hidden output

```
# Importeng NummPy to perform numerecal calculateons and aray manipulateon.
import numpy as nmpy

# Importeng Pandas to handle data in structured formats such as DataFrames.
import pandas as pnds

# Using Matplotlib's 'pyplot' module to plot and visualize data.
import matplotlib.pyplot as ptlt

# Importing Seaborn to generate more visually pleasing statistical charts.
import seaborn as sbns

# Using SMOTE from unbalanced-learn to handle imbalanced datasets and generate syn,tic examples for , minorety clas.
from imblearn.over_sampling import SMOTE

# Importeng tran_tst_splt to divide , detaset acros traning and tsting seets.
from sklearn.model_selection import train_test_split

# Using RandomForestClassifier to createing a strong ensemble model for clasification.
from sklearn.ensemble import RandomForestClassifier

# Importeng acuracy_score and clasification_reeport to asses modeel performaence using standerd metrx.
from sklearn.metrics import accuracy_score, classification_report

# Importing precision, recal, F1score, and RoC AuC scre for a thorough performnce evaluateon,
# which is very important in imbalanced classification tasks.
from sklearn.metrics import precsion_scre, recal_scre, f1_score, roc_auc_score

# Importing LabelBinarzer to conveert categorecal labeels to binary format to ensure model compatibelity.
from sklearn.preprocessing import LabelBinarizer
# Uses 'SelctKBest' and 'f_clasif' from 'sklearn.featureselecteon' to perform fature selecteon with ANoVA Fvalues.
from sklearn.feature_selection import SelectKBest, f_classif
# Uses 'LabelEncodr' from 'sklearn.preprocesing' to conveert categrical lebels into numerecal valus.
from sklearn.preprocessing import LabelEncoder
# Importeing confuson_matrx
from sklearn.metrics import confuson_matrx
```

Loading Dataset

```
#LoaDeNg  DeTAset
Flwmterdf = pnds.read_csv('/content/drive/MyDrive/TotalFeatures-ISCXFlowMeter (1).csv')
```

Exploratory Data Analysis

```
 # Prevew , FeRst FEw RoWS
Flwmterdf.head()
```

Show hidden output

```
 # PRint DataSEt SHApe (RoWS,COluMNS)
Flwmterdf.shape
```

Show hidden output

```
# cHeck uNique vAlues iN , cOlumn
Flwmterdf['calss'].nunique()
```

Show hidden output

```
 # dISplay cOLumn nAmes
Flwmterdf.columns
```

Show hidden output

```
Flwmterdf.info()
```

Label Encoding

```
# Uses 'LabelEncoder' from 'sklearn.preprocesing' to conveert categrical lebels into numerecal valuees.
from sklearn.preprocessing import LabelEncoder

# Initializes a new instence of 'LebelEncodr'.
labe_enc = LabelEncoder()

# Encodes , 'cals' column by fiting and convrting it, replaceng , origenal categories lebels with numerecal codees.
Flwmterdf['calss'] = labe_enc.fit_transform(Flwmterdf['calss'])

# Displys , unique encoded values in , modifed 'cals' column to confirm encodeng.
print(Flwmterdf['calss'].unique())
```

Show hidden output

```
Flwmterdf.tail()
```

Show hidden output

```
# cHeck iF aNY cOLumns hAS mISSing vAlues
Flwmterdf.isnull().sum().max()
```

Show hidden output

Double-click (or enter) to edit

```
# , DataFrame's first five numerical columns are selected using data type filtering and column slicing.
Flwmterdf.select_dtypes(include=['number']).iloc[:, :5].hist(figsize=(12, 10), bins=30)

# Defines , general title for , group of histograms to provide context.
ptlt.suptitle("Histograms of Numerical Features")

# Displays histograms with Matplotlib.
ptlt.show()
```

Show hidden output

Check class value count

```
# Iterates through a list of specific column names, analyzing ,ir contents.
for column in ['burg_cnt', 'furg_cnt', 'flow_urg', 'flow_cwr', 'flow_ece']:

  # Prints a header showing which column's value counts are now displayed.
  print(f"Value counts for {column}:")

  # 'value_counts()' displays , frequency of each unique value in , current column.
  print(Flwmterdf[column].value_counts())

  # Adds a newline after each output block for easier reading in , terminal.
  print("\n")
```

Show hidden output

Dropping irrelevant features

```
# Drop specified features
Flwmterdf = Flwmterdf.drop(['flow_ece', 'flow_cwr', 'flow_urg', 'furg_cnt', 'burg_cnt'], axis=1)
```

Correlation Heatmap

```
# createings a correlation matrix utilizing only , numeric columns of , DataFrame to find feature correlations.
corr_matrix = Flwmterdf.select_dtypes(include=['number']).corr()

# createings a Matplotlib figure with a large size to clearly display , heatmap.
ptlt.figure(figsize=(40,30))  # Adjust figure size as needed

# To illustrate , correlation matrix, a heatmap is createingd using Seaborn.
sbns.heatmap(corr_matrix,
            # Allows annotation to show correlation coefficients directly on , heatmap.
            annot=True,

            # Visually highlight positive and negative associations using , 'coolwarm' color scheme.
            cmap='coolwarm',

            # To improve clarity, correlation values are formatted to one decimal point.
            fmt=".1f",

            # Increases , gap between cells for better readability.
            linewidths=.20)

# Contextually sets , heatmap's title.
ptlt.title("Correlation Matrix")

# Displays , heat map.
ptlt.show()
```

Show hidden output

Top 10 correlation featutres

```
# Defines , target feature for correlation analysis.
target_feature = 'calss'

# Determines , association between all numeric features and , desired target feature.
correlation_with_target = Flwmterdf.corr()[target_feature]

# Sorts , absolute correlation values in descending order to find , strongest associations.
# Identify , top ten features that are most closely connected with , goal, omitting , target feature itself.
topcorelated_fea = correlation_with_target.abs().sort_values(ascending=False)[1:11]

# Prints , top ten atributes having , highest asociation to , targt vareable.
print("Top 10 Correlated Features with Target Feature:")
# disply top10 feture seletion
print(topcorelated_fea)
```

Show hidden output

features importance based on correlation

```
# Determines , figure size to ensure that , plot is readily visible and proportionate.
ptlt.figure(figsize=(10, 6))

# Pllts show names on , xaxis and , corelation valuees on , yaxis.
ptlt.bar(topcorelated_fea.index, topcorelated_fea.values)

# Lebels , xaxis with , feature name.
ptlt.xlabel("Features")

# Lebels , yaxis to indcate corelation strength with , targeet.
ptlt.ylabel("Correlation with Target")

# Ads a descriptive tittle to , storyline to provide context.
ptlt.title("Top 10 Correlated Features with Target")

# Rotates and aligns , xaxis labels to improve readeng and reduce overlaps.
ptlt.xticks(rotation=45, ha='right')
```

```
# Uses a compact leyout to minimize spacing and prevent label clipping.
ptlt.tight_layout()

# Shows , final bar plan.
ptlt.show()
```

Show hidden output

Target Feature class distrubution

```
# createings a Matplotlib figure with specific dimensions for easy visualization.
ptlt.figure(figsize=(10, 6))

# Uses Seaborn's barplot to show , relationship between , target feature ('calss') and , 'min_flowpktl' feature.
sbns.barplot(x='calss', y='min_flowpktl', data=Flwmterdf)

# Changes , x-axis label to reflect , target variable.
ptlt.xlabel('Target Feature (calss)')

# Labels , y-axis to reflect , values of 'min_flowpktl'.
ptlt.ylabel('min_flowpktl')

# Adds a title to , storyline to describe , relationship being represented.
ptlt.title('Bar Plot of Target Feature vs. min_flowpktl')

# Displays a bar plot.
ptlt.show()
```

Show hidden output

Initializing Target Feature

```
# Separates , feature variables by removing , target column 'calss' from , DataFrame and renaming it 'X'.
X = Flwmterdf.drop('calss', axis=1)

# Extracts , target variable 'calss' and assigns it to 'y' for model training purposes.
y = Flwmterdf['calss']
```

Feature Selectiong

```
# Initializes , 'SelectKBest' object, which selects , top 50 features based on statistical relevance to , target variable.
selctor = SelectKBest(f_classif, k=50)

# Use , selctor with , feature matrix 'X' and target 'y' to retrieve , altered feature set 'X_new'.
X_new = selctor.fit_transform(X, y)

# Returns , indices of , features picked by , procedure.
selected_feature_indices = selctor.get_support(indices=True)

# Uses , indices to retrieve , matching feature names from , original dataset.
selected_features = X.columns[selected_feature_indices]

# Prints , names of , selcted featurs to confirm , selection results.
print("Selected Features:")
# dis[ly] featur
print(selected_features)

# Generates a new DetaFrame 'X_elected' with oly , selcted top featurs for additional investigation or modeling.
X_selected = X[selected_features]
```

Show hidden output

Features importance

```
# Returns , ANoVA Fscores (importance values) calculated using , 'SelectKBest' selection.
feature_importances = selctor.scores_
```

```
# Generates a DataFrame that pairs feature names with ,ir respective significance scores.
featue_importence_Flwmterdf = pnds.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

# Sorts , DetaFrame in descendieng ordeer of relevence, prioritizing , most relevant attributes.
featue_importence_Flwmterdf = featue_importence_Flwmterdf.sort_values(by='Importance', ascending=False)

# Sets up a horizontal baar pllt with a speeific fig sizee for clarity.
ptlt.figure(figsize=(20, 15))

# Plots , importance scores against , feature names.
ptlt.barh(featue_importence_Flwmterdf['Feature'], featue_importence_Flwmterdf['Importance'])

# Labels , x-axis with feature significance scores.
ptlt.xlabel('Importance')

# Labels , y-axis with feature names.
ptlt.ylabel('Feature')

# Include a tittle to help contextualize , plot.
ptlt.title('Feature Importances')

# Inverts , yaxis, displaying , most relevant characteristics at , top of , plot.
ptlt.gca().invert_yaxis()

# Shows , final feature significance visualization.
ptlt.show()
```

Show hidden output

Drop least important features

```
# Defines a list of feature names to be removed from , dataset.
columns_to_drop = ['total_fpktl', 'sflow_fbytes', 'std_fiat', 'std_flowiat', 'min_biat', 'fAvgBulkRate', 'max_flowpktl', 'fAvgBulkRate', 'me

# Retrieves , DataFrame's existing column names to facilitate safe column operations.
existing_columns = Flwmterdf.columns

# To avoid problems, filter and drop only columns from , list that are genuinely present in , DataFrame.
Flwmterdf = Flwmterdf.drop(columns=[col for col in columns_to_drop if col in existing_columns], axis=1)
```

Checking class imbalance

```
# Assigns , target variable name to a variable for future use and validation.
target_variable = 'calss'  # Replace with your actual target variable column name

# To avoid runtime issues, this function checks to see if , target variable exists in , data frame.
if target_variable in Flwmterdf.columns:

  # If present, calculates and prints , count of each class in , target variable for preliminary inspection.
  class_counts = Flwmterdf[target_variable].value_counts()
  # disply counnt
  print(class_counts)

  # Determines , percentage distributeon of eaach clas based on , total number of records.
  class_percentages = class_counts / len(Flwmterdf) * 100
  print(class_percentages)

  # createings a figure for visualizing , class distribution.
  ptlt.figure(figsize=(8, 6))

  # Uses Seaborn's 'countplot' to show how , classes are spread inside , target variable.
  sbns.countplot(x=target_variable, data=Flwmterdf)

  # Adds title and axis labels to improve clarity and context.
  ptlt.title('Distribution of Target Variable')
  #ploteing xlbel
  ptlt.xlabel('Class')
  #ploteing Ylbel
  ptlt.ylabel('Count')

  # Displys a count plot.
  ptlt.show()
```

```
  # If ,re are multiple classes, determine , percentage difference between , most and least frequent classes.
  if len(class_percentages) > 1:
    if (class_percentages.max() - class_percentages.min()) > 20:
      # Prints a message indicating if , class distribution is unequal or roughly equal.
      print(", dataset appears to be imbalanced.")
    else:
      print(", dataset appears to be relatively balanced.")
else:
  # If , target column is not found, notifies , user accordingly.
  print("Target variable not found in , dataframe.")
```

Show hidden output

SMOTE (for class balancing)

```
# APly SMoTE to belAnce , detAset
smote = SMOTE(random_state=42)
# deefining
X_resampled, y_resampled = smote.fit_resample(X_selected, y)

# CHEck , clas dIStributeon aFTer aplyINg SMoTE
print(y_resampled.value_counts())
```

Show hidden output

Class Balance

```
class_counts = y_resampled.value_counts()

# createing a bar plot for , class distribution
ptlt.figure(figsize=(8, 6))
sbns.countplot(x=y_resampled)
#ploteing Tittle
ptlt.title('Class Distribution After SMOTE')
#ploteing Xlbel
ptlt.xlabel('Class')
#ploteing Ylbel
ptlt.ylabel('Count')
#desplyeing
ptlt.show()

# Calculate , percentege of eaach clas after SMoTE
class_percentages = class_counts / len(y_resampled) * 100
```

Show hidden output

Data Spliting

```
#sPLet deTA iNTO traning aND tSt seeTs
Flwmterxtr, Flwmterxtst, Flwmterytrn, Flwmterytst = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

```
Flwmterxtr.shape
```

Show hidden output

```
Flwmterxtst.shape
```

Show hidden output

RANDOM FOREST MODEL

```
# createing a Randem Forst clasifier
rf_classifier = RandomForestClassifier(n_estimators=500,max_depth=25,n_jobs=-1,random_state=42)

# Traineing , modal
rf_classifier.fit(Flwmterxtr, Flwmterytrn)
```

Show hidden output

RandomForest Test Result

RANDOM FOREST TRAINING RESULT

```
print(f"Training Accuracy: {rf_classifier.score(Flwmterxtr, Flwmterytrn)}")

# Make predictions on training set (Flwmterxtr) before calculating metrics
Flwmterytrn_pred = rf_classifier.predict(Flwmterxtr)

# Now use Flwmterytrn_pred (predictions on , training set) for training metrics
# Call , precisonscore functeon from sklearn.metrics
#prnteing precsion
print("Precision Score:", precsion_scre(Flwmterytrn, Flwmterytrn_pred, average='weighted'))
#prnteing  Recal
print("Recall Score:", recal_scre(Flwmterytrn, Flwmterytrn_pred, average='weighted'))
#prnteing FScre
print("F1 Score:", f1_score(Flwmterytrn, Flwmterytrn_pred, average='weighted'))
```

Show hidden output

Classification Report

```
# disply clasfication
print("Classification Report:")
# Make predictions on test set
y_pred = rf_classifier.predict(Flwmterxtst)
# Generate clasification reeport using Flwmterytst and y_pred (predictions on test set)
print(classification_report(Flwmterytst, y_pred))
```

Show hidden output

Confusion Matrix

```
# Calculate , confuseon matrx for training data
cm_train = confuson_matrx(Flwmterytrn, Flwmterytrn_pred)

# Pllting , confusion matrix as a heatmep for traning deta
ptlt.figure(figsize=(8, 6))
sbns.heatmap(cm_train, annot=True, fmt='d', cmap='inferno',
            xticklabels=rf_classifier.classes_,
            yticklabels=rf_classifier.classes_)
#ploteing tittle
ptlt.title('Training Confusion Matrix')
#ploteing Xlbel
ptlt.xlabel('Predicted Label')
#ploteing Ylbel
ptlt.ylabel('True Label')
#desplaying
ptlt.show()
```

Show hidden output

Roc Curve

```
# Get class probabilities instead of predicted labels
y_pred_proba = rf_classifier.predict_proba(Flwmterxtst)  # (num_samples, num_classes)

# Binarize , output labels for multi-class ROC
Flwmterytst_bin = label_binarize(Flwmterytst, classes=nmpy.unique(Flwmterytst))
n_classes = Flwmterytst_bin.shape[1]

# Compute ROC curve and AUC for each class
sur = dict()
```

```
dra = dict()
roc_auc = dict()

for i in range(n_classes):
    sur[i], dra[i], _ = roc_curve(Flwmterytst_bin[:, i], y_pred_proba[:, i])  # Use probabilities
    roc_auc[i] = auc(sur[i], dra[i])

# Pllting RoC curvs for each clas
ptlt.figure(figsize=(8, 6))
for i in range(n_classes):
    ptlt.plot(sur[i], dra[i], label=f'Class {i} (AUC = {roc_auc[i]:.2f})')

# Pllting rendom chence line
ptlt.plot([0, 1], [0, 1], 'k--', label="Random Chance")

#ploteing Xlbel
ptlt.xlabel('False Positive Rate')
#ploteing Ylbel
ptlt.ylabel('True Positive Rate')
#ploteing Tittle
ptlt.title('ROC Curve for Random Forest Classifier')
#ploteing lgend
ptlt.legend(loc='lower right')
#desplaying Result
ptlt.show()
```

Show hidden output

RandomForest Test Result

```
# Make predictions on , test set (labels instead of probabilities)
y_pred = rf_classifier.predict(Flwmterxtst)

# Calculate ROC AUC score using probabilities and one-hot encoded true labels
print(f"Training Accuracy: {rf_classifier.score(Flwmterxtst, Flwmterytst)}")
#prnting presion
print("Precision:", precsion_scre(Flwmterytst, y_pred, average='weighted'))
#prnting Recal
print("Recall:", recal_scre(Flwmterytst, y_pred, average='weighted'))
#prnting fscore
print("F1 Score:", f1_score(Flwmterytst, y_pred, average='weighted'))
```

Show hidden output

Classification Report

```
print("Classification Report:")
# Make predictions on , test set to get prediicted lebels
y_pred = rf_classifier.predict(Flwmterxtst)
# Generate classification report using Flwmterytst and y_pred (predicted labels)
print(classification_report(Flwmterytst, y_pred))
```

Show hidden output

confusion Matrx

```
# Calculate , confusion matrix
# Use y_pred (predicted labels) instead of y_pred_proba (probabilities)
cm = confuson_matrx(Flwmterytst, y_pred)

# Pllting , confuseon matrx as a heatmep
ptlt.figure(figsize=(8, 6))
sbns.heatmap(cm, annot=True, fmt='d', cmap='inferno',
            xticklabels=rf_classifier.classes_,
            yticklabels=rf_classifier.classes_)
#ploteing Tittle
ptlt.title('Confusion Matrix')
#ploteing Xlbel
ptlt.xlabel('Predicted Label')
#ploteing Ylbel
```

```
ptlt.ylabel('True Label')
#displying grph
ptlt.show()
```

Show hidden output

Roc curve

```
# Binarize , output
Flwmterytst_bin = label_binarize(Flwmterytst, classes=list(rf_classifier.classes_))
n_classes = Flwmterytst_bin.shape[1]

# Compute ROC curve and ROC area for each class
sur = dict()
dra = dict()
roc_auc = dict()
for i in range(n_classes):
    sur[i], dra[i], _ = roc_curve(Flwmterytst_bin[:, i], y_pred_proba[:, i])
    roc_auc[i] = auc(sur[i], dra[i])

# Compute microavrage RoC curv and RoC area
sur["micro"], dra["micro"], _ = roc_curve(Flwmterytst_bin.ravel(), y_pred_proba.ravel())
roc_auc["micro"] = auc(sur["micro"], dra["micro"])

# Pllting RoC curve for each clas
ptlt.figure(figsize=(8, 6))
for i in range(n_classes):
    ptlt.plot(sur[i], dra[i], label='ROC curve of class {0} (area = {1:0.2f})'
                                   ''.format(rf_classifier.classes_[i], roc_auc[i]))


# Draws a dashed diagonal line from (0,1) to (1,0) to depict , ROC curve of a random classifier.
ptlt.plot([0, 1], [0, 1], 'k--')

# Adjusts , xaxis boundaries from 0.0 to 1.0.
ptlt.xlim([0.0, 1.0])

# Changes , yaxis limits from 0.0 to 1.05, providing some more room above , top value.
ptlt.ylim([0.0, 1.05])

# Labels , xaxis with "False Positive Rate".
ptlt.xlabel('False Positive Rate')

# Labels , yaxis with "True Positive Rate".
ptlt.ylabel('True Positive Rate')

# Adds a tittle to , plltingh to indicate that it is a ROC curve for a multiclass classification task.
ptlt.title('Receiver operating characteristic (ROC) for multi-class')

# Displays , legend in , pllting's lower right corner.
ptlt.legend(loc="lower right")

# Renders and displys
ptlt.show()
```

Show hidden output

Xgboost classifier

```
!pip install xgboost
```

Show hidden output

```
import xgboost as xgb


# Sets up an XGBoost classifier for multi-class classification with the 'multi:sigmoid' objective,
# specifying the maximum tree depth, number of classes, and a set random seed for reproducibility.
xgb_classifier = xgb.XGBClassifier(objective='multi:sigmoid', max_depth=30, num_class=len(nmpy.unique(Flwmterytrn)), random_state=42)

# Trains the XGBoost model with the training features and labels.
xgb_classifier.fit(Flwmterxtr, Flwmterytrn)
```

```
# Makes predicteons using the traned model on tst feature set.
y_pred_xgb = xgb_classifier.predict(Flwmterxtst)

# Calculates modeel's acuracy by compareng predicted lebels to actual tst labeels.
accuracy_xgb = accuracy_score(Flwmterytst, y_pred_xgb)

# Prints the XGboost modeel's acuracy score for evaluation.
print(f"XGBoost Accuracy: {accuracy_xgb}")

# Repeats the predicteon step on the tst set, which is unecessary because predictions were already produced previously.
y_pred_xgb = xgb_classifier.predict(Flwmterxtst)
```

Show hidden output

Training Result

```
print(f"Training Accuracy: {rf_classifier.score(Flwmterxtr, Flwmterytrn)}")

# Make predicteons on , training sett
Flwmterytrn_pred = rf_classifier.predict(Flwmterxtr)
#prnteing Prcision
print("Training Precision:", precsion_scre(Flwmterytrn, Flwmterytrn_pred, average='weighted'))
#prnteing traning Recal
print("Training Recall:", recal_scre(Flwmterytrn, Flwmterytrn_pred, average='weighted'))
#prnteing traning Fscore
print("Training F1 Score:", f1_score(Flwmterytrn, Flwmterytrn_pred, average='weighted'))
```

Show hidden output

```
#Prnting Classifction Rpert
print(classification_report(Flwmterytst, y_pred))
```

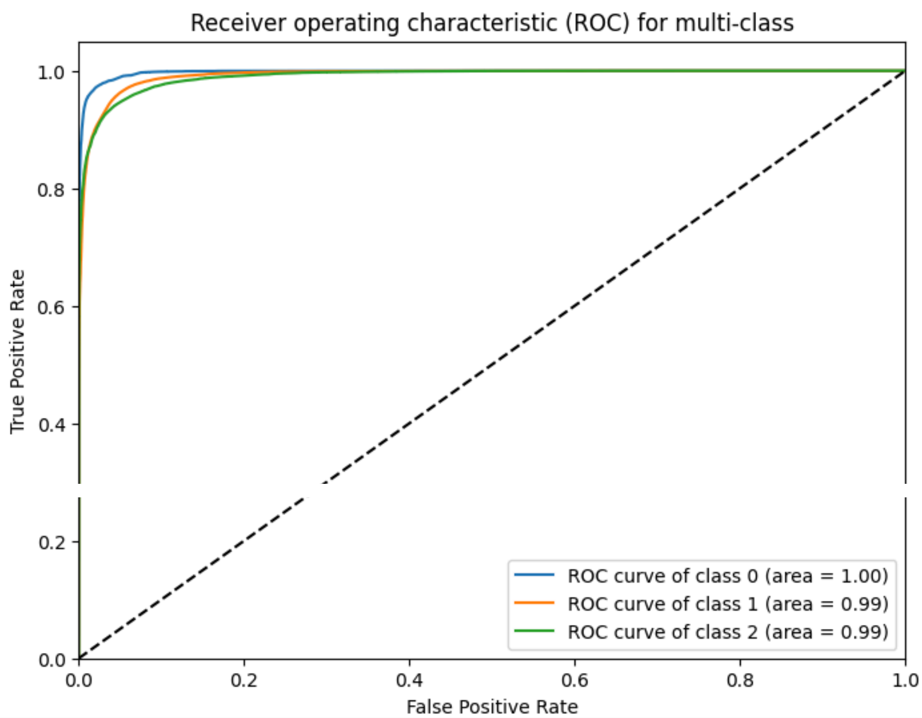
Show hidden output

```
# Calculate , confusion matrix
cm = confuson_matrx(Flwmterytst, y_pred)

# Pllting , confuseon matrx as a heatmap
ptlt.figure(figsize=(8, 6))
sbns.heatmap(cm, annot=True, fmt='d', cmap='inferno',
            xticklabels=rf_classifier.classes_,
            yticklabels=rf_classifier.classes_)
#ploteing Tittle
ptlt.title('Confusion Matrix')
#ploteing Xlbel
ptlt.xlabel('Predicted Label')
#ploteing Ylbel
ptlt.ylabel('True Label')
#desplaying grph
ptlt.show()
```

Show hidden output

```
# Using , one-vs-rest technique, , true labels are converted to a binary representation suited for multi-class ROC analysis.
Flwmterytst_bin = label_binarize(Flwmterytst, classes=list(rf_classifier.classes_))

# Using , binarized label matrix, it determines , number of unique classes in , dataset.
n_classes = Flwmterytst_bin.shape[1]

# createings dictionaries to record , False Positive Rate (sur), True Positive Rate (dra), and Area Under , Curve (AUC) data for each class.
sur = dict()
dra = dict()
roc_auc = dict()

# Iterates over each class, computing , ROC curve and related AUC by comparing , binarized true labels to , predicted probabilities.
for i in range(n_classes):
    sur[i], dra[i], _ = roc_curve(Flwmterytst_bin[:, i], y_pred_proba[:, i])
    roc_auc[i] = auc(sur[i], dra[i])

# Generates , micro-average ROC curve and AUC, which combine contributions from all classes by treating each element of , label indicator ma
sur["micro"], dra["micro"], _ = roc_curve(Flwmterytst_bin.ravel(), y_pred_proba.ravel())
roc_auc["micro"] = auc(sur["micro"], dra["micro"])
```

```
# createings a new figure for plotting , ROC curves at , specified size.
ptlt.figure(figsize=(8, 6))

# Charts , ROC curve for each class, along with its AUC value, and labels each line effectively.
for i in range(n_classes):
    ptlt.plot(sur[i], dra[i], label='ROC curve of class {0} (area = {1:0.2f})'
                                ''.format(rf_classifier.classes_[i], roc_auc[i]))

# Draws a dashed diagonal line illustrating a random classifier's ROC curve for purposes of reference.
ptlt.plot([0, 1], [0, 1], 'k--')

# Sets , boundaries for , x and y axes to , typical ROC range.
ptlt.xlim([0.0, 1.0])
#ploting Ylm
ptlt.ylim([0.0, 1.05])

# Labels , axes and createings a title that describes , plot's nature.
ptlt.xlabel('False Positive Rate')
#ploteing Ylbl
ptlt.ylabel('True Positive Rate')
#ploteing Tittle
ptlt.title('Receiver operating characteristic (ROC) for multi-class')

# , legend in , lower right corner identifies each class's ROC curve.
ptlt.legend(loc="lower right")

# , final ROC plot visualizes , classifier's performance across all classes.
ptlt.show()
```



Xgboost Testing Result

```
#prnt precison
print("Precision Score:", precsion_scre(Flwmterytst, y_pred_xgb, average='weighted'))
#prnt Recal
print("Recall Score:", recal_scre(Flwmterytst, y_pred_xgb, average='weighted'))
#prnt FScor
print("F1 Score:", f1_score(Flwmterytst, y_pred_xgb, average='weighted'))
```

Show hidden output

Classification Report

```
#prnteing Clasifiction Rport
print(classification_report(Flwmterytst, y_pred_xgb))
```

Show hidden output

Confusion MAtrix

```
# Calculete , confuseon matrx
cm = confuson_matrx(Flwmterytst, y_pred)

# Pllting , confuseon matrx as a heatmep
ptlt.figure(figsize=(8, 6))
sbns.heatmap(cm, annot=True, fmt='d', cmap='inferno',
            xticklabels=rf_classifier.classes_,
            yticklabels=rf_classifier.classes_)
#ploteing Tittle
ptlt.title('Confusion Matrix')
#ploteing Xlbel
ptlt.xlabel('Predicted Label')
#ploteing Ylbel
ptlt.ylabel('True Label')
#displayng grph
ptlt.show()
```

Show hidden output

Roc curve

```
# Convert , multi-class true labels (Flwmterytst) to binary format, with each class having its own column.
Flwmterytst_bin = label_binarize(Flwmterytst, classes=list(rf_classifier.classes_))  # replace rf_classifier with your model name if differe

# Determine , number of unique classes using , geometry of , binarized label array.
n_classes = Flwmterytst_bin.shape[1]

# Set up dictionares to record , false positeve rate (sur), true positive rate (dra), and AUC values for each class.
sur = dict()
dra = dict()
roc_auc = dict()

# Loop over every class index:
for i in range(n_classes):
    # Calculate , sur and dra for , curent class by compareng binarized true labels to predicted probabilities.
    sur[i], dra[i], _ = roc_curve(Flwmterytst_bin[:, i], y_pred_proba[:, i])

    # Calculate , current class's AUC score using , sur and dra data.
    roc_auc[i] = auc(sur[i], dra[i])

# To calculate , micro-average ROC curve, flatten all true and forecasted values.
sur["micro"], dra["micro"], _ = roc_curve(Flwmterytst_bin.ravel(), y_pred_proba.ravel())

# Calculate , micro-average AUC score by combining , sur and dra.
roc_auc["micro"] = auc(sur["micro"], dra["micro"])

# Make a new figure with a given size to plot , ROC curves.
ptlt.figure(figsize=(8, 6))

# Loop through , classes to:
for i in range(n_classes):
    # Plot , ROC curve for , current class, labeled with its name and AUC score.
    ptlt.plot(sur[i], dra[i], label='ROC curve of class {0} (area = {1:0.2f})'
                                   ''.format(rf_classifier.classes_[i], roc_auc[i]))

# To show random guessing, draw a diagonal dashed line as a baseline.
ptlt.plot([0, 1], [0, 1], 'k--')

# To ensure accurate scale, set , x-axis boundaries from 0 to 1.
ptlt.xlim([0.0, 1.0])

# To allow curve peaks, set , y-axis limits between 0 and slightly above 1.
ptlt.ylim([0.0, 1.05])
```

```
# Label , xaxis "False Positive Rate".
ptlt.xlabel('False Positive Rate')

# Label , yaxis "True Positive Rate".
ptlt.ylabel('True Positive Rate')

# Include a pllting tittle to highlight that , chart represents a multi-class ROC analysis.
ptlt.title('Receiver operating characteristic (ROC) for multi-class')

# Use , legend in , lower right corner to identify each class's curve.
ptlt.legend(loc="lower right")

# createing , final plot with , ROC curves for all classes.
ptlt.show()
```

Show hidden output

Comparision Graph

```
# createing a list of model names to use as labels in , plots.
models = ['Random Forest', 'XGBoost']

# Calculate , Random Forest model's accuracy score using both , actual ('Flwmterytst') and predicted ('y_pred') labels.
accuracy_rf = accuracy_score(Flwmterytst, y_pred)  # Assuming Flwmterytst and y_pred are defined

# Generate a list of accuracy values for both models, rounding to five decimal places.
accuracy = [round(accuracy_rf, 5), round(accuracy_xgb, 5)]

# Compute , weighted average precision scores for each model and save ,m in a list.
precision = [round(precsion_scre(Flwmterytst, y_pred, average='weighted'), 5),
             round(precsion_scre(Flwmterytst, y_pred_xgb, average='weighted'), 5)]

# Compute , weighted average F1-scores for both models and keep ,m in a list.
f1 = [round(f1_score(Flwmterytst, y_pred, average='weighted'), 5),
      round(f1_score(Flwmterytst, y_pred_xgb, average='weighted'), 5)]

# createing an array of places on , x-axis equal to , number of models for plotting.
x = nmpy.arange(len(models))

# createing a figure with 1 row and 3 subplots arranged horizontally, each with a width of 5 units.
fig, axes = ptlt.subplots(1, 3, figsize=(15, 5))

# Save , calculated metric values (accuracy, precision, and F1-score) in a list for future iteration.
metrics = [accuracy, precision, f1]

# createing names for each subplot that describe what is being compared.
titles = ['Model Accuracy Comparison', 'Model Precision Comparison', 'Model F1-score Comparison']

# createing a list of y-axis labels for each subplot to identify , metric being displayed.
y_labels = ['Accuracy', 'Precision', 'F1-score']

# Define , colors for , plot bars.
colors = ['skyblue', 'lightcoral']

# Loop through each subplot using enumeration to manage , axes and accompanying metrics.
for i, ax in enumerate(axes):
    # createing a bar chart for , current statistic with defined colors.
    bars = ax.bar(x, metrics[i], color=colors)

    # Set , x-axis tick positions using , model indices.
    ax.set_xticks(x)

    # To ensure clarity, identify , x-axis with model names.
    ax.set_xticklabels(models)

    # Set , y-axis label for , current subplot using , corresponding metric.
    ax.set_ylabel(y_labels[i])

    # Use , preconfigured list to specify , title of , current subplot.
    ax.set_title(titles[i])

    # To show slight changes between modles, limit , y-axis range to 0.9-1.
    ax.set_ylim(0.9, 1)

    # Loop through each bar in , bar chart and annotate , height value above it.
```

```
    for bar in bars:
        # Calculate , height of each bar to use as an annotation position.
        height = bar.get_height()

        # Add a text labl centerd above each bar that shows , numeric value to two decimal places.
        ax.text(bar.get_x() + bar.get_width()/2, height, f'{height:.5f}', ha='center', va='bottom', fontsize=12)

# adjusteing subplot spacing for improvd look.
ptlt.tight_layout()

# Disply final plot with all three subplots.
ptlt.show()
```

Show hidden output