**Loan Eligibility Prediction**

**INTRODUCTION:**

In today’s financial landscape, determining a customer's loan eligibility is a crucial process for banking and financial institutions. Manual evaluation of loan applications is time-consuming and often prone to inconsistencies. To streamline this process, we developed a machine learning-based system that predicts loan eligibility using various applicant parameters such as income, loan amount, credit history, and employment status. This project utilizes Python and the scikit-learn library to build and evaluate different classification models including Logistic Regression, Decision Tree, Random Forest, and Support Vector Machine. Data preprocessing, visualization, model training, evaluation, and performance comparison have been carried out to identify the most accurate model. The objective is to assist financial institutions in making informed and faster decisions regarding loan approval while minimizing human bias and errors.

**CODING:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sb

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn import metrics

from sklearn.svm import SVC

from imblearn.over\_sampling import RandomOverSampler

import warnings

warnings.filterwarnings('ignore')

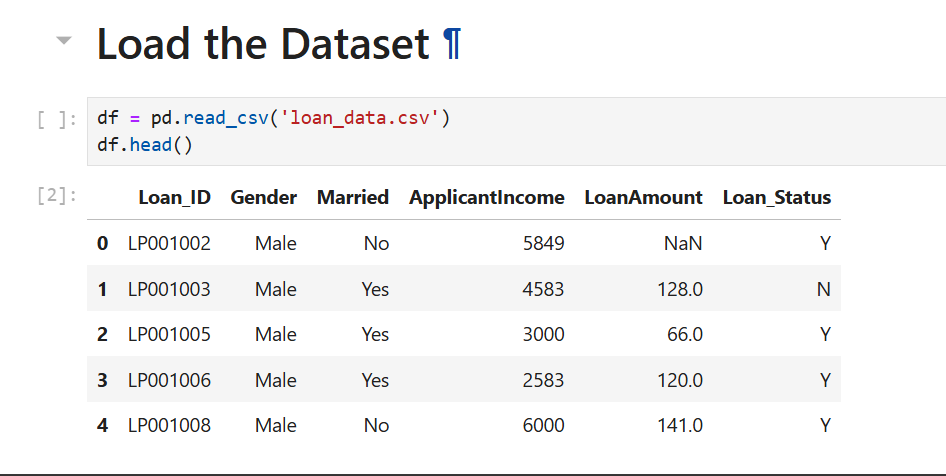
* numpy: A package for numerical computing in Python, useful for handling large, multi-dimensional arrays and matrices, and performing mathematical operations.
* pandas: A library for data manipulation and analysis, especially for structured data like tables (DataFrames).
* matplotlib.pyplot: A plotting library used to create static, interactive, and animated visualizations in Python.
* seaborn: A data visualization library built on top of matplotlib that makes it easier to generate complex visualizations with less code.
* sklearn.model\_selection.train\_test\_split: A utility from Scikit-learn to split datasets into training and test sets for model training and evaluation.
* sklearn.preprocessing.LabelEncoder: A utility for encoding categorical labels (e.g., transforming labels like "yes"/"no" to 1/0).
* sklearn.preprocessing.StandardScaler: Used to standardize features by removing the mean and scaling to unit variance. This is often important when using algorithms like Support Vector Machines (SVM).
* sklearn.metrics: Contains various functions to assess the performance of a model (e.g., accuracy, precision, recall, etc.).
* sklearn.svm.SVC: The Support Vector Machine (SVM) classifier, which is useful for classification tasks.
* imblearn.over\_sampling.RandomOverSampler: A technique for handling imbalanced datasets. It oversamples the minority class by duplicating examples to balance the dataset
* warnings.filterwarnings ('ignore'): Suppresses any warnings that might pop up during the execution of the code. This is useful when you are not interested in seeing minor warnings.

**LOAD THE DATASET:**

df = pd.read\_csv('loan\_data.csv')

df.head()

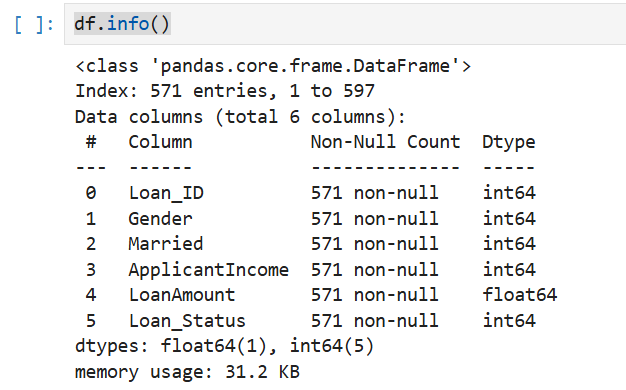
* df = pd.read\_csv('loan\_data.csv'): This line reads the CSV file loan\_data.csv into a pandas DataFrame, which is a tabular data structure (similar to a table or a spreadsheet).
* pd.read\_csv() is a function provided by pandas to read CSV files. The df variable stores the resulting DataFrame.
* df.head(): This method returns the first five rows of the DataFrame by default. It gives you a quick overview of the structure of the data (the columns and their values).



**DATASET INFORMATION:**

df.info()

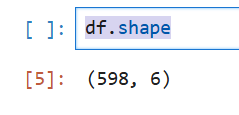
* The df.info() command provides a concise summary of the DataFrame df. It’s used to understand the **structure** and **basic properties** of the dataset.



**GETTING THE DIMENSIONS OF THE DATAFRAME:**

df.shape

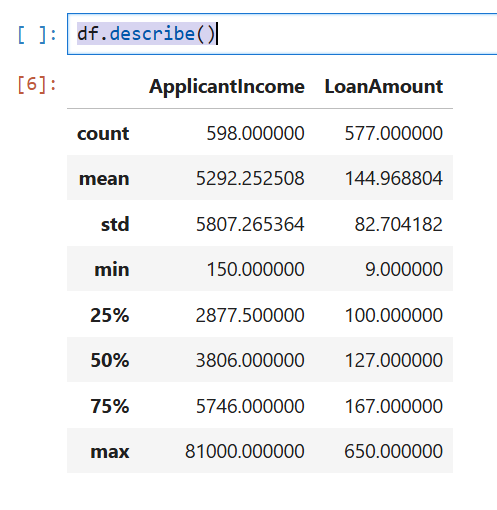
* df.shape returns a tuple indicating the number of rows and columns in the DataFrame.



**DESCRIPTIVE STATISTICS OF NUMERICAL COLUMNS:**

df.describe()

* df.describe() provides statistical summaries (like count, mean, std, min, max, and percentiles) for all numerical columns in the DataFrame.

****

**EXPLORATORY DATA ANALYSIS:**

1. **PIE CHART:**

temp = df['Loan\_Status'].value\_counts()

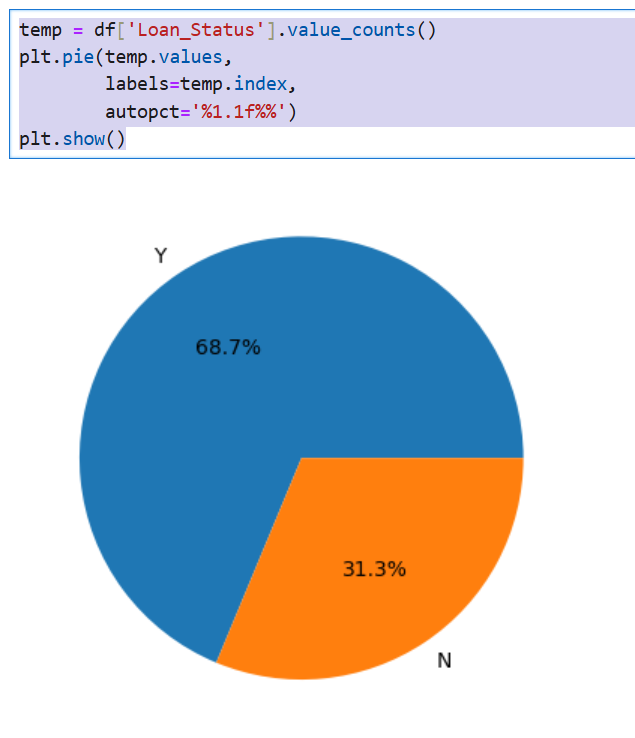
plt.pie(temp.values,

labels=temp.index,

autopct='%1.1f%%')

plt.show()

* This code generates a pie chart to visualize the distribution of loan statuses in the dataset.
* It first uses value\_counts() to count the occurrences of each unique value in the 'Loan\_Status' column, then plots these counts as a pie chart using matplotlib.
* The labels parameter assigns the category names to the chart slices, and autopct='%1.1f%%' displays the percentage of each category on the chart, formatted to one decimal place.
* Finally, plt.show() displays the pie chart.



1. **BAR PLOT:**

plt.subplots(figsize=(15, 5))

for i, col in enumerate(['Gender', 'Married']):

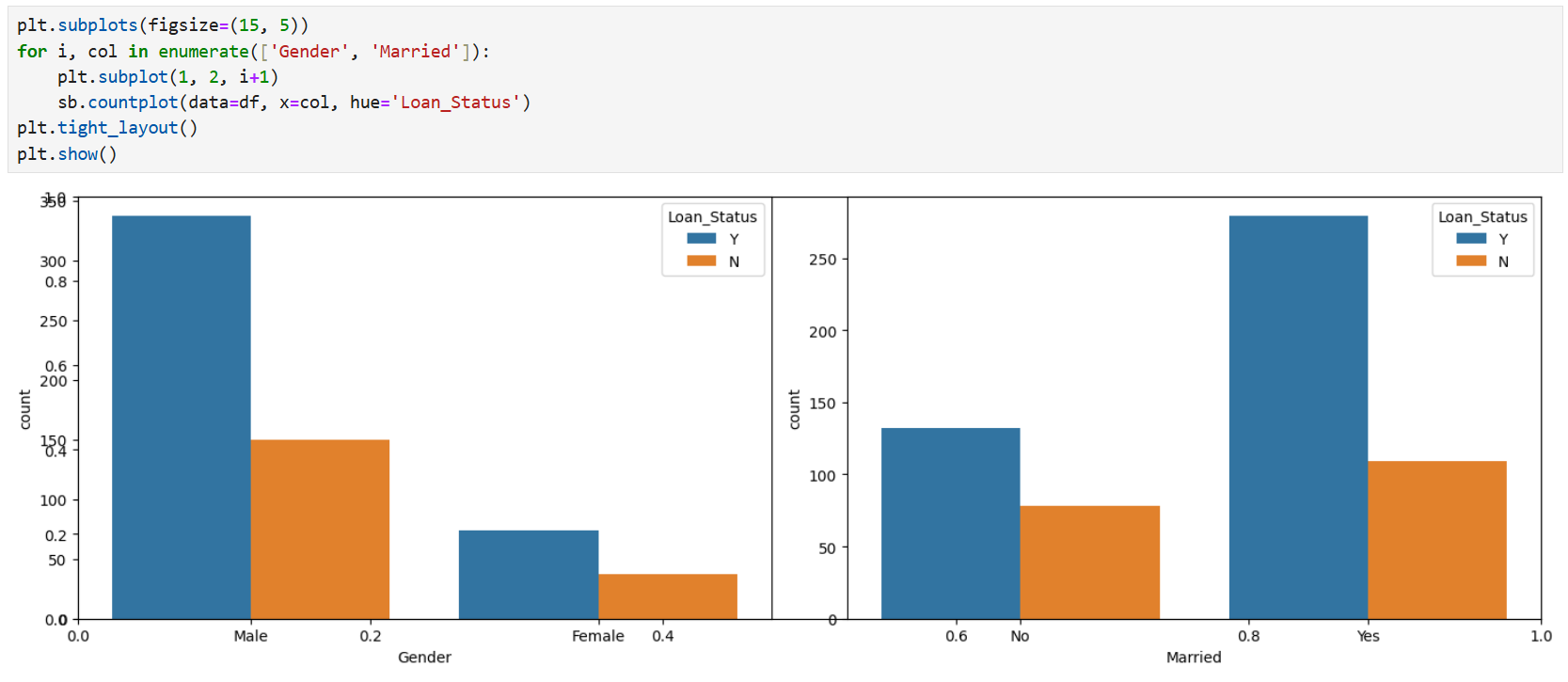
plt.subplot(1, 2, i+1)

sb.countplot(data=df, x=col, hue='Loan\_Status')

plt.tight\_layout()

plt.show()

* This code creates side-by-side bar plots to compare loan approval status based on the applicant's **Gender** and **Married** status.
* A figure of size 15×5 inches is created using plt.subplots().
* Then, a for loop iterates over the columns 'Gender' and 'Married', and for each, a subplot is generated using plt.subplot(1, 2, i+1) to place them side-by-side (1 row, 2 columns).
* The seaborn.countplot() function is used to plot the frequency of each category, further divided by Loan\_Status using the hue parameter.
* plt.tight\_layout() ensures proper spacing between subplots, and plt.show() displays the final figure.



1. **HISTOGRAM:**

plt.subplots(figsize=(15, 5))

for i, col in enumerate(['ApplicantIncome', 'LoanAmount']):

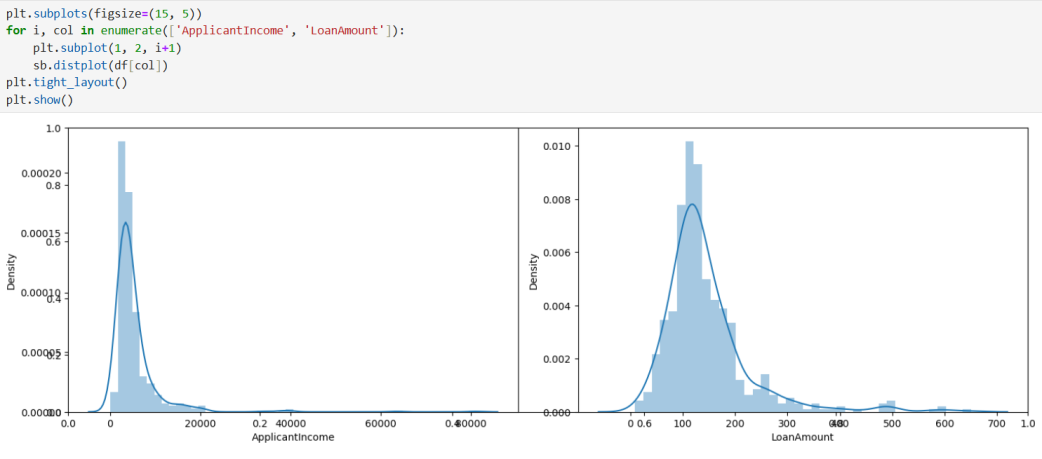
plt.subplot(1, 2, i+1)

sb.distplot(df[col])

plt.tight\_layout()

plt.show()

* This code generates side-by-side distribution plots to visualize the distribution of **ApplicantIncome** and **LoanAmount** in the dataset.
* The plt.subplots(figsize=(15, 5)) creates a figure with a size of 15×5 inches.
* The for loop iterates over the columns 'ApplicantIncome' and 'LoanAmount', and for each column, it creates a subplot using plt.subplot(1, 2, i+1) to place the plots side by side (1 row, 2 columns). seaborn.distplot() is used to plot the distribution (histogram + KDE) of each feature.
* plt.tight\_layout() ensures there’s no overlap between subplots, and plt.show() displays the plots.



1. **BOX PLOT:**

plt.subplots(figsize=(15, 5))

for i, col in enumerate(['ApplicantIncome', 'LoanAmount']):

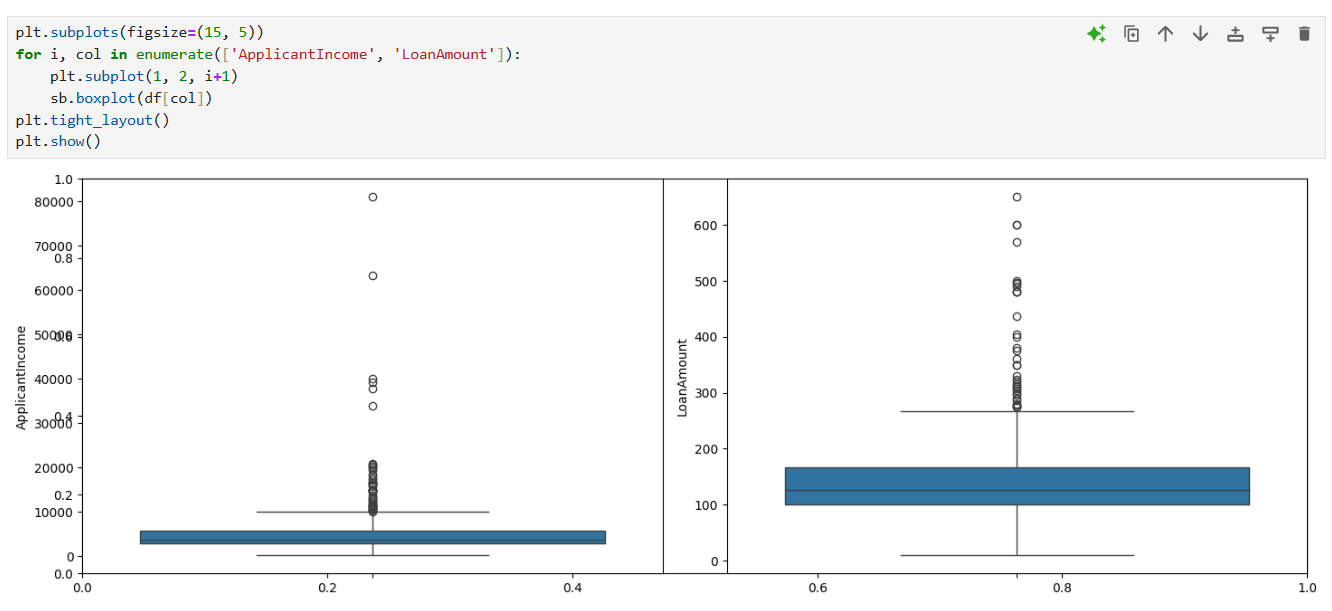
plt.subplot(1, 2, i+1)

sb.boxplot(df[col])

plt.tight\_layout()

plt.show()

* This code creates side-by-side **box plots** to visualize the spread and detect outliers in the **ApplicantIncome** and **LoanAmount** columns.
* The plt.subplots(figsize=(15, 5)) defines the figure size.
* Then, a for loop iterates over the two numerical columns, creating a subplot for each using plt.subplot(1, 2, i+1).
* The seaborn.boxplot() function is used to draw the box plots, which show the median, quartiles, and any outliers in the data.
* plt.tight\_layout() adjusts the spacing to prevent overlap, and plt.show() displays the figure.



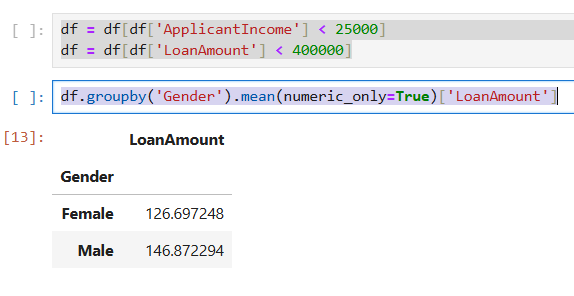
**AVERAGE LOAN AMOUNT BY GENDER (AFTER FILTERING):**

df = df[df['ApplicantIncome'] < 25000]

df = df[df['LoanAmount'] < 400000]

df.groupby('Gender').mean(numeric\_only=True)['LoanAmount']

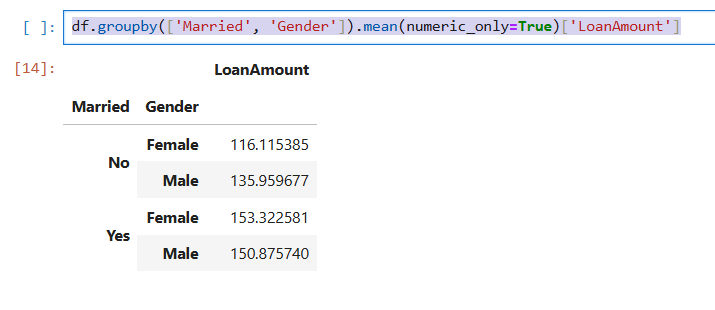
* The given code filters the DataFrame df to include only rows where the ApplicantIncome is less than 25,000 and the LoanAmount is less than 400,000, then groups the filtered data by the Gender column and calculates the mean LoanAmount for each gender using numeric columns only.



**AVERAGE LOAN AMOUNT BY MARITAL STATUS AND GENDER:**

df.groupby(['Married', 'Gender']).mean(numeric\_only=True)['LoanAmount']

* This code groups the DataFrame df by both Married and Gender columns, then calculates the average LoanAmount for each combination of marital status and gender, considering only numeric columns.

****

**HEATMAP OF STRONG FEATURE CORRELATIONS AFTER LABEL ENCODING:**

def encode\_labels(data):

for col in data.columns:

if data[col].dtype == 'object':

le = LabelEncoder()

data[col] = le.fit\_transform(data[col])

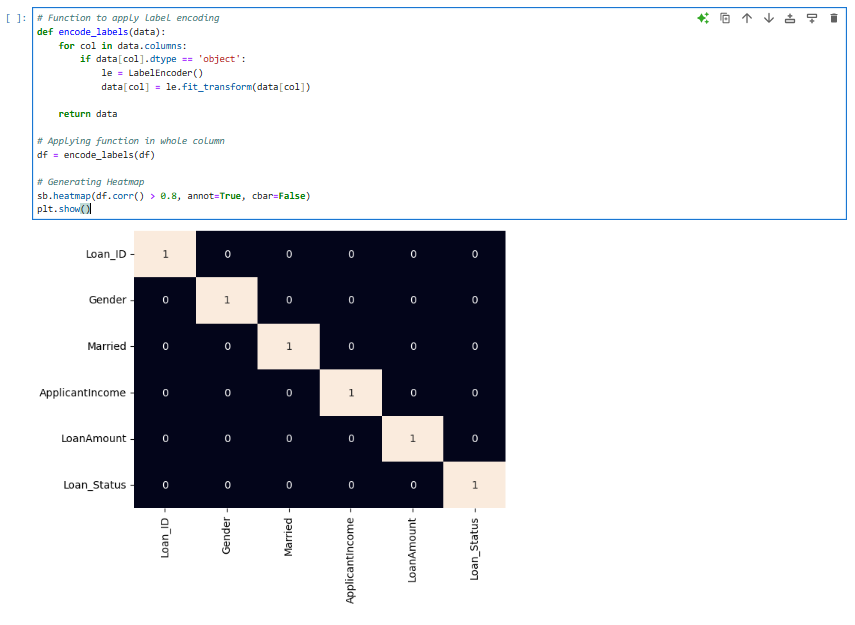
return data

df = encode\_labels(df)

sb.heatmap(df.corr() > 0.8, annot=True, cbar=False)

plt.show()

* The code defines a function to label encode all categorical columns in the DataFrame so that correlation can be computed, then it generates a heatmap showing where the correlation between features is greater than 0.8, highlighting strong relationships between variables.



# DATA PREPROCESSING:

# features = df.drop('Loan\_Status', axis=1)

# target = df['Loan\_Status'].values

# X\_train, X\_val, Y\_train, Y\_val = train\_test\_split(features, target, test\_size=0.2,random\_state=10)

# ros = RandomOverSampler(sampling\_strategy='minority',

# random\_state=0)

# X, Y = ros.fit\_resample(X\_train, Y\_train)

# X\_train.shape, X.shape

* Splits the dataset into features (X) and target (Y), where Loan\_Status is the target variable.
* Splits the data into training and validation sets using train\_test\_split, with 20% of the data used for validation.
* Handles class imbalance in the training data using RandomOverSampler, which duplicates samples of the minority class to balance the dataset.
* Finally, it checks the shapes of X\_train (original) and X (resampled).

# MODEL DEPLOYMENT:

# from sklearn.metrics import roc\_auc\_score

# model = SVC(kernel='rbf')

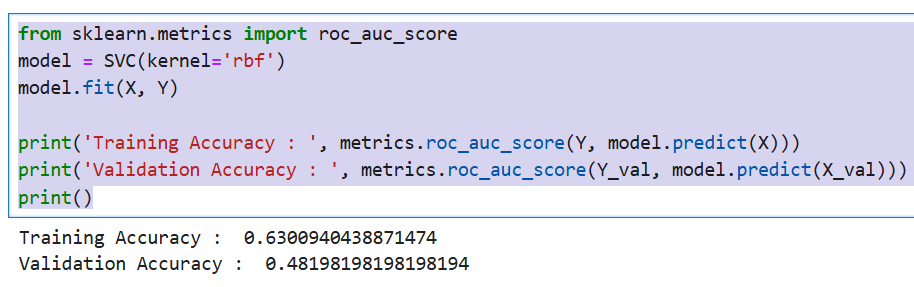
# model.fit(X, Y)

# print('Training Accuracy : ', metrics.roc\_auc\_score(Y, model.predict(X)))

# print('Validation Accuracy : ', metrics.roc\_auc\_score(Y\_val, model.predict(X\_val)))

# print()

* Trains an SVM classifier with an RBF kernel on the balanced training data and evaluates it using the ROC AUC score for both the training and validation sets.
* This metric measures how well the model distinguishes between the classes, especially useful for imbalanced datasets.



**MODEL EVALUATION AND PERFORMANCE VISUALIZATION:**

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix

training\_roc\_auc = roc\_auc\_score(Y, model.predict(X))

validation\_roc\_auc = roc\_auc\_score(Y\_val, model.predict(X\_val))

print('Training ROC AUC Score:', training\_roc\_auc)

print('Validation ROC AUC Score:', validation\_roc\_auc)

print()

plt.figure(figsize=(6, 6))

sb.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

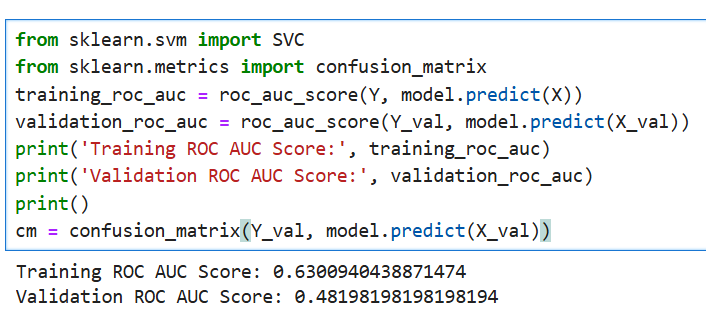
plt.title('Confusion Matrix')

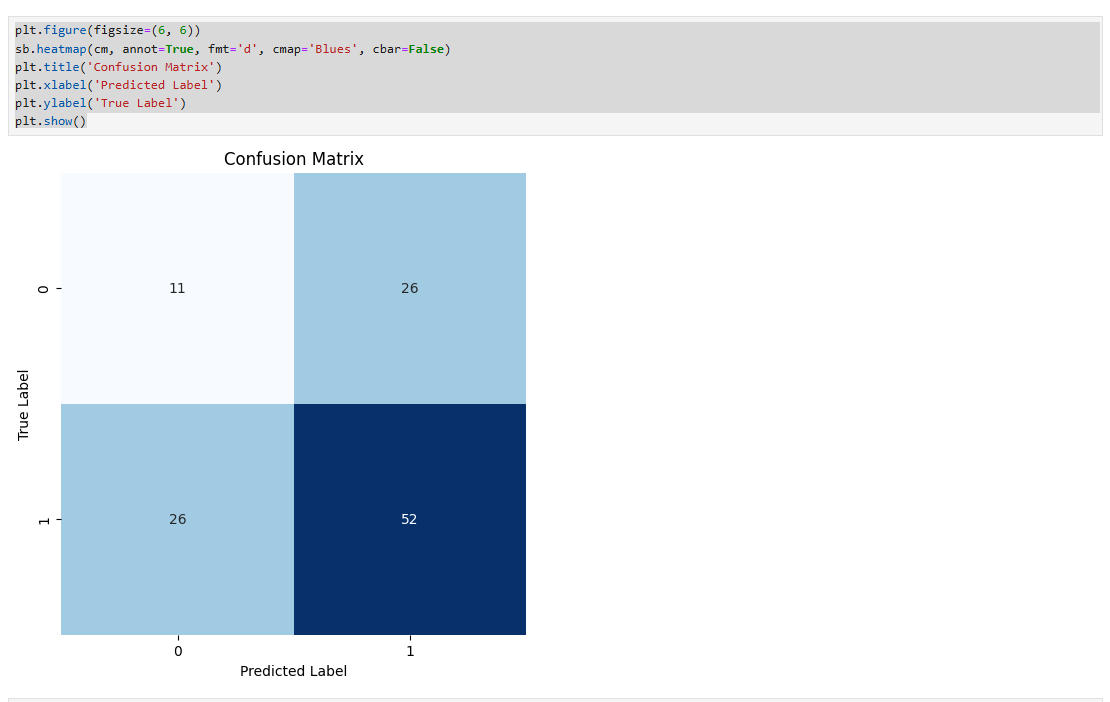
plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()cm = confusion\_matrix(Y\_val, model.predict(X\_val))

* This code evaluates a machine learning model (assumed to be an SVC) by calculating the ROC AUC score for both the training and validation sets using roc\_auc\_score.
* It then computes the confusion matrix for the validation set using confusion\_matrix, and visualizes it as a heatmap with seaborn.heatmap.
* The heatmap displays the performance of the model, showing how well the predicted labels match the true labels.
* The ROC AUC scores provide an overall measure of the model's ability to discriminate between classes, with higher values indicating better performance.
* The confusion matrix offers a detailed breakdown of true positives, true negatives, false positives, and false negatives, helping to assess the model's accuracy, precision, recall, and other metrics.



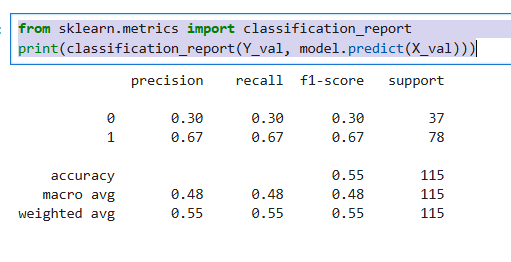


**CLASSIFICATION REPORT FOR MODEL EVALUATION:**

from sklearn.metrics import classification\_report

print(classification\_report(Y\_val, model.predict(X\_val)))

* This section of the code prints the classification report, which provides key performance metrics such as precision, recall, F1-score, and support for each class in the validation set.
* It helps assess the model's effectiveness in distinguishing between classes and highlights areas for improvement in its predictions.



**CROSS VALIDATION:**

from sklearn.model\_selection import cross\_val\_score

from sklearn.pipeline import make\_pipeline

pipeline = make\_pipeline(StandardScaler(), SVC(kernel='rbf'))

scores = cross\_val\_score(pipeline, features, target, cv=5, scoring='roc\_auc')

print("Cross-Validation ROC AUC Scores:", scores)

print("Mean ROC AUC Score:", scores.mean())

* This code evaluates the model's stability and generalization ability using 5-fold cross-validation with ROC AUC as the scoring metric.
* It first creates a pipeline that standardizes the features using StandardScaler and applies an SVC with an RBF kernel.
* Then, cross\_val\_score splits the dataset into 5 parts, trains the model on 4 parts, and tests it on the remaining part in each iteration.
* It returns the ROC AUC scores for each fold, and the mean score gives an overall estimate of the model’s performance across different data splits, helping to ensure the model is not overfitting or underperforming on unseen data.

