In [1]:

```
## Madhavi Ghanta
## DS 680 Project 2
```

In [2]:

```
#Load required libraries

import pandas as pd
import numpy as np
import plotly.express as px
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import opendatasets as od

from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection  import chi2, SelectKBest
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score,confusion_matrix,
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

In [3]:

```
## Load the dataset
heart_df = pd.read_csv("C:/Users/mghan/Documents/MSDS/DSC680/Project 2_Heart/heart.csv
heart_df.head(5)
```

Out[3]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina
0	40	M	ATA	140	289	0	Normal	172	N
1	49	F	NAP	160	180	0	Normal	156	N
2	37	M	ATA	130	283	0	ST	98	N
3	48	F	ASY	138	214	0	Normal	108	Y
4	54	M	NAP	150	195	0	Normal	122	N

In [4]:

```
## Data Processing
# Check for null columns
heart_df.isna().sum()
```

Out[4]:

```
Age               0
Sex               0
ChestPainType     0
RestingBP         0
Cholesterol       0
FastingBS         0
RestingECG        0
MaxHR             0
ExerciseAngina    0
Oldpeak           0
ST_Slope          0
HeartDisease      0
dtype: int64
```

In [5]:

```
# Chest Pain types

# Value 1: typical angina
# Value 2: atypical angina
# Value 3: non-anginal pain
# Value 4: asymptomatic
```

In [6]:

```
# Check for duplicates
heart_df[heart_df.duplicated()==True]
```

Out[6]:

Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina

In [7]:

```
print('Dataframe before dropping duplicates :', heart_df.shape)
heart_df = heart_df.drop_duplicates()
print('Dataframe before dropping duplicates :', heart_df.shape)
```

```
Dataframe before dropping duplicates : (918, 12)
Dataframe before dropping duplicates : (918, 12)
```

In [8]:

```
# There are no duplicates to drop.
```

In [9]:

```
# Renaming columns
```

In [10]:

```
heart_df.columns = ['Age','Sex','ChestPainType','RestingBP','Cholesterol','FastingBloc
'ST_Slope','HeartDiseaseOutcome']
heart_df.head(5)
```

Out[10]:

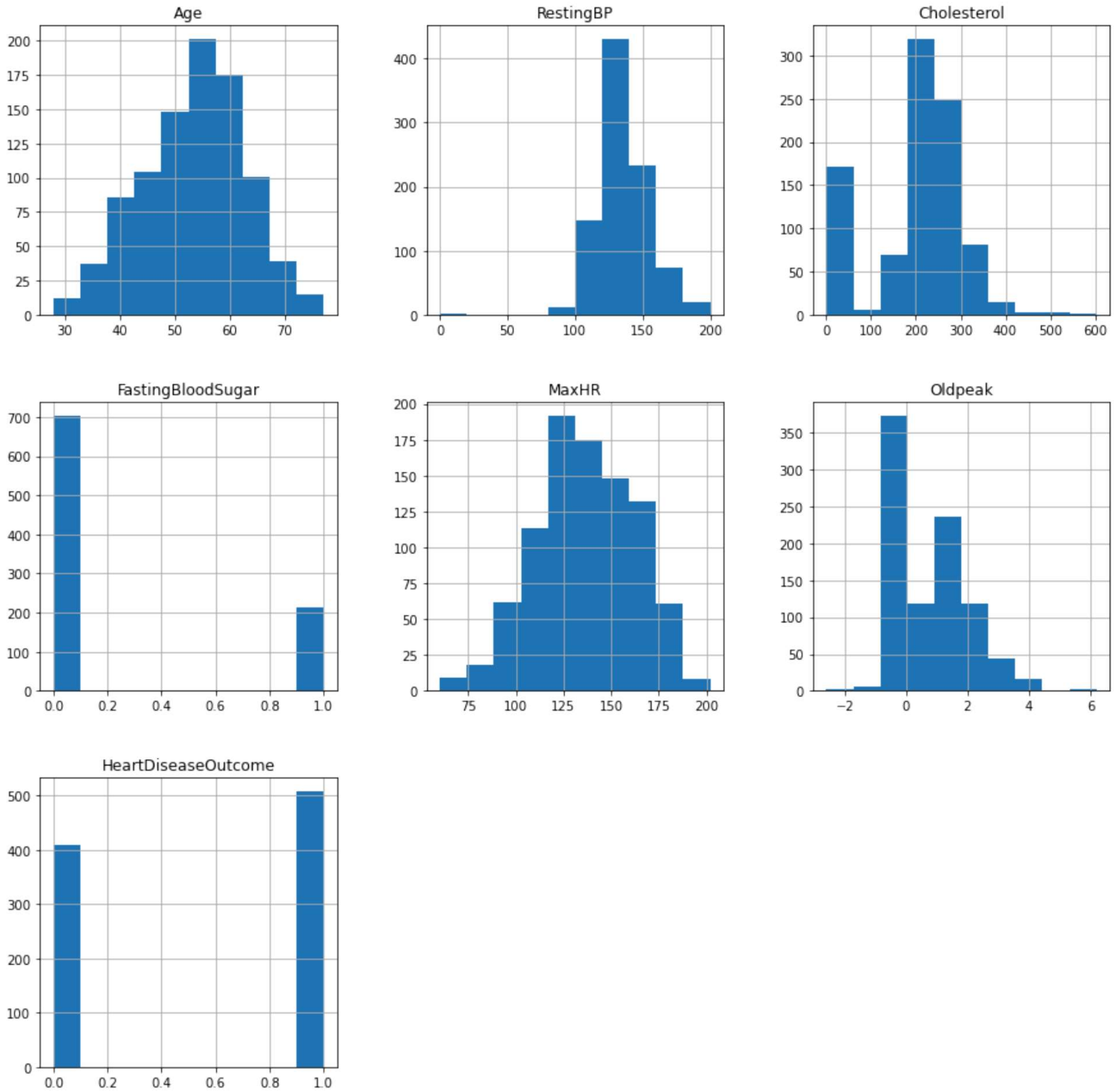
	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBloodSugar	RestingECG	MaxHR	Exercis
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	

In [11]:

```
# Data Visualizations
```

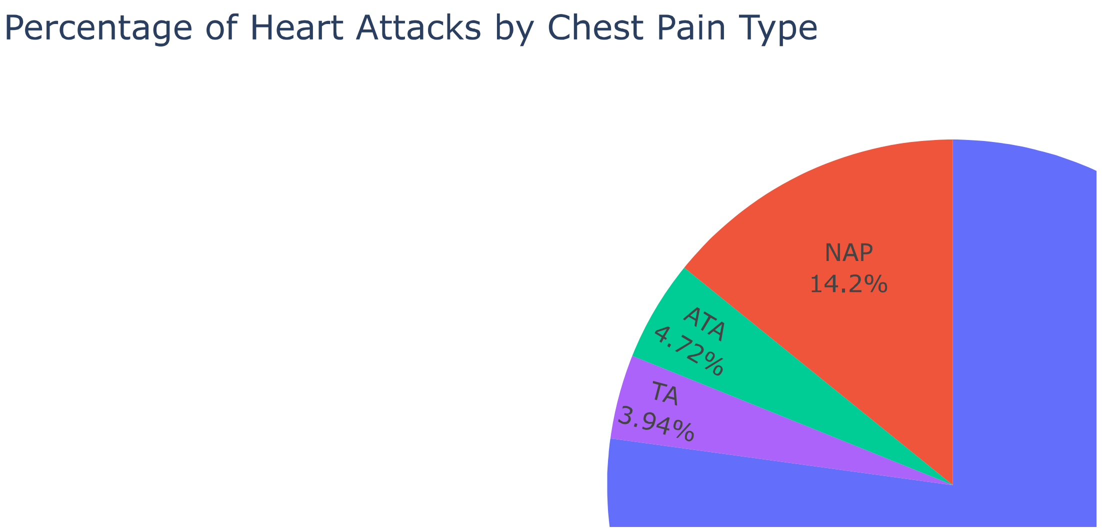
In [13]:

```
heart_df.hist(figsize = (15,15))
plt.show()
```



In [12]:

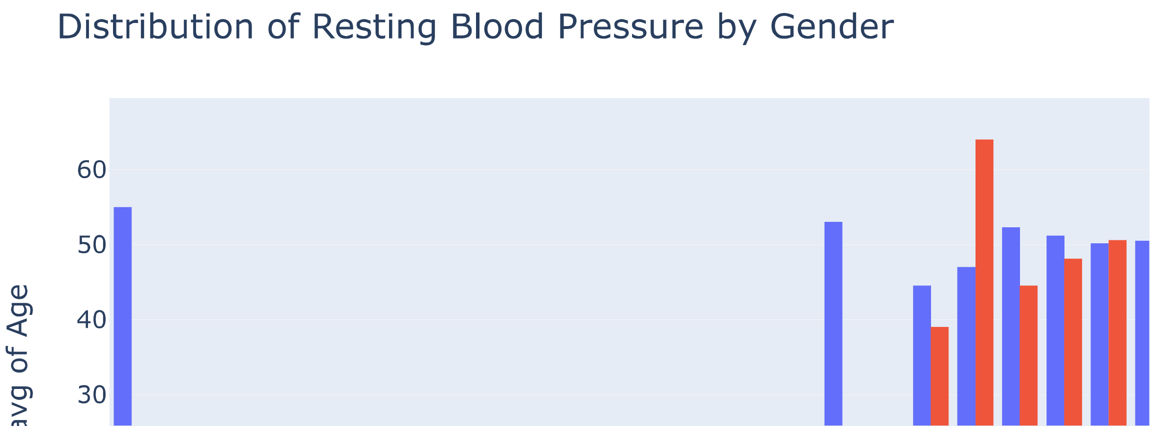
```
fig = px.pie(heart_df[heart_df.HeartDiseaseOutcome==1],  names='ChestPainType', title=
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.update_layout(title = "Percentage of Heart Attacks by Chest Pain Type")
fig.show("notebook")
```



Percentage of Heart Attacks by Chest Pain Type

```
In [13]: # Sex: displays the gender of the individual using the following format : 1 = male 0 =
```

```
In [14]: fig = px.histogram(heart_df, y="Age", x="RestingBP",color='Sex', barmode='group',histf
         fig.update_layout(title = "Distribution of Resting Blood Pressure by Gender")
         fig.show("notebook")
```



In [31]:
```
## xx = heart_df['HeartDiseaseOutcome'].value_counts().reset_index()
# def formatter(x, pos):
#     return str(x)

# ax = sns.barplot(x="HeartDiseaseOutcome",y="count",data=xx)
# ax.set_title('Distribution of HeartAttack Outcomes')
# ax.yaxis.set_major_formatter(formatter)
# ax.yaxis.set_minor_formatter(NullFormatter())
# for i in ax.containers:
#    ax.bar_label(i,)
```

In [15]:
```
# Model Building
```

In [16]:
```
X = heart_df.drop(['HeartDiseaseOutcome'],axis=1)
Y = heart_df['HeartDiseaseOutcome']
X.shape, Y.shape
```

Out[16]:
```
((918, 11), (918,))
```

In [17]:
```
X.dtypes
```

Out[17]:
```
Age                     int64
Sex                    object
ChestPainType          object
RestingBP               int64
Cholesterol             int64
FastingBloodSugar       int64
RestingECG             object
MaxHR                   int64
ExerciseAngina         object
Oldpeak               float64
ST_Slope               object
dtype: object
```

In [18]:

```
# Encode categorical variables (e.g., 'gender', 'category', 'state', etc.)
categorical_columns = ['Sex', 'ChestPainType','RestingECG', 'ExerciseAngina','ST_Slope
for col in categorical_columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
```

In [19]:

```
# Split dataset into Train and Test Sets
```

In [20]:

```
scaler = StandardScaler()
x = scaler.fit_transform(X)
X.shape, x.shape
```

Out[20]:

```
((918, 11), (918, 11))
```

In [21]:

```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(x,Y, test_size=0.2, random_state=4
```

In [22]:

```
X_train.shape, X_test.shape, Y_train.shape, Y_test.shape
```

Out[22]:

```
((734, 11), (184, 11), (734,), (184,))
```

In [23]:

```
## Models
# Random Forest

# Use the RandomForestClassifier to fit balanced data
rfc = RandomForestClassifier()
rfc_model = rfc.fit(X_train,Y_train)

#Predict y data with classifier:
y_pred_rfc = rfc_model.predict(X_test)

# Evaluate the model
print(classification_report(Y_test, y_pred_rfc))
print(confusion_matrix(Y_test, y_pred_rfc))
print(f'ROC-AUC score : {roc_auc_score(Y_test, y_pred_rfc)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_rfc)}')
```

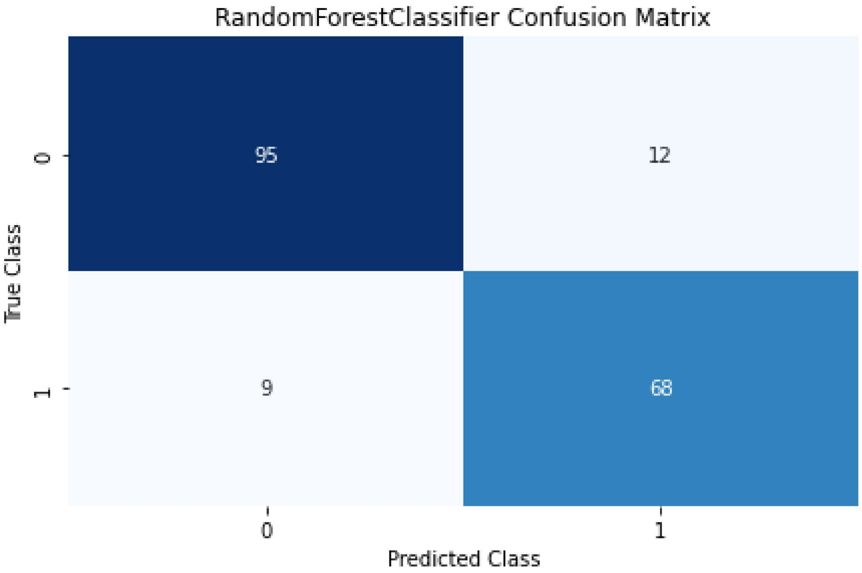
```
              precision    recall  f1-score   support

           0       0.83      0.87      0.85        77
           1       0.90      0.87      0.89       107

    accuracy                           0.87       184
   macro avg       0.87      0.87      0.87       184
weighted avg       0.87      0.87      0.87       184

[[67 10]
 [14 93]]
ROC-AUC score : 0.8696443743172715
Accuracy score : 0.8695652173913043
```

In [28]:

```
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_rfc, labels=[1,0])

print(matrix)

# Create pandas dataframe
```

```
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("RandomForestClassifier Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[95 12]
 [ 9 68]]
```



In [24]:

```
# Train a logistic regression model
logistic_model = LogisticRegression(solver='liblinear', random_state=42)
logistic_model.fit(X_train,Y_train)

# Make predictions on the test set
y_pred_lr = logistic_model.predict(X_test)

# Evaluate the model
print(classification_report(Y_test, y_pred_lr))
print(confusion_matrix(Y_test, y_pred_lr))
print(f'ROC-AUC score : {roc_auc_score(Y_test, y_pred_lr)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_lr)}')
```

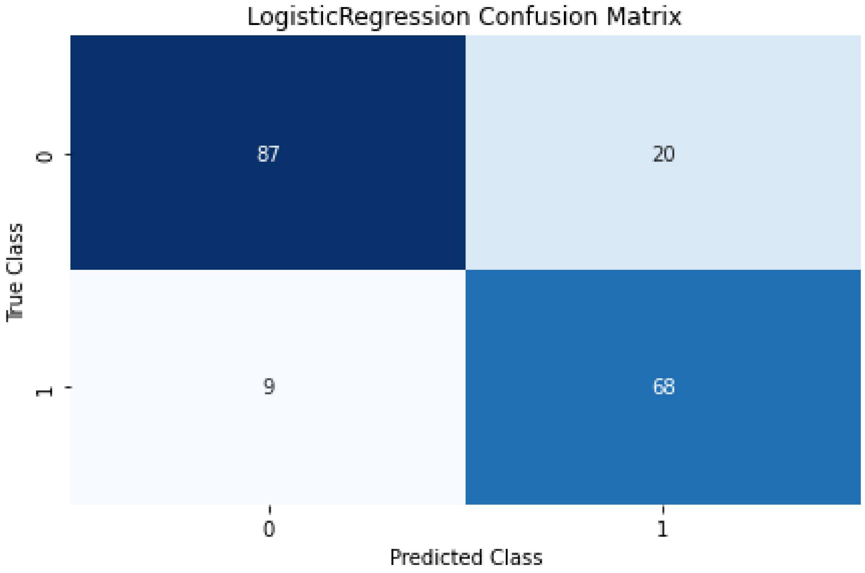
```
              precision    recall  f1-score   support

           0       0.77      0.88      0.82        77
           1       0.91      0.81      0.86       107

    accuracy                           0.84       184
   macro avg       0.84      0.85      0.84       184
weighted avg       0.85      0.84      0.84       184

[[68  9]
 [20 87]]
ROC-AUC score : 0.8481004976332078
Accuracy score : 0.842391304347826
```

In [25]:

```
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_lr, labels=[1,0])

print(matrix)
```

```
# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("LogisticRegression Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[87 20]
 [ 9 68]]
```



In [26]:

```
svc_model = SVC()
svc_model.fit(X_train, Y_train)

y_pred_svc = svc_model.predict(X_test)

# Evaluate the model
print(classification_report(Y_test, y_pred_svc))
print(confusion_matrix(Y_test, y_pred_svc))
print(f'ROC-AUC score : {roc_auc_score(Y_test, y_pred_svc)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_svc)}')
```

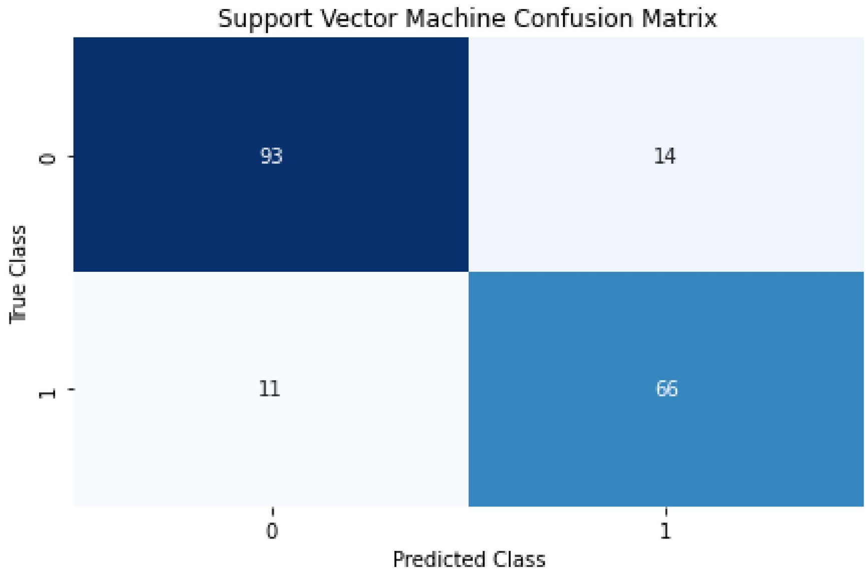
```
              precision    recall  f1-score   support

           0       0.82      0.86      0.84        77
           1       0.89      0.87      0.88       107

    accuracy                           0.86       184
   macro avg       0.86      0.86      0.86       184
weighted avg       0.87      0.86      0.86       184

[[66 11]
 [14 93]]
ROC-AUC score : 0.863150867823765
Accuracy score : 0.8641304347826086
```

In [27]:

```
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_svc, labels=[1,0])

print(matrix)
```

```python
# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("Support Vector Machine Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[93 14]
 [11 66]]
```



In [28]:

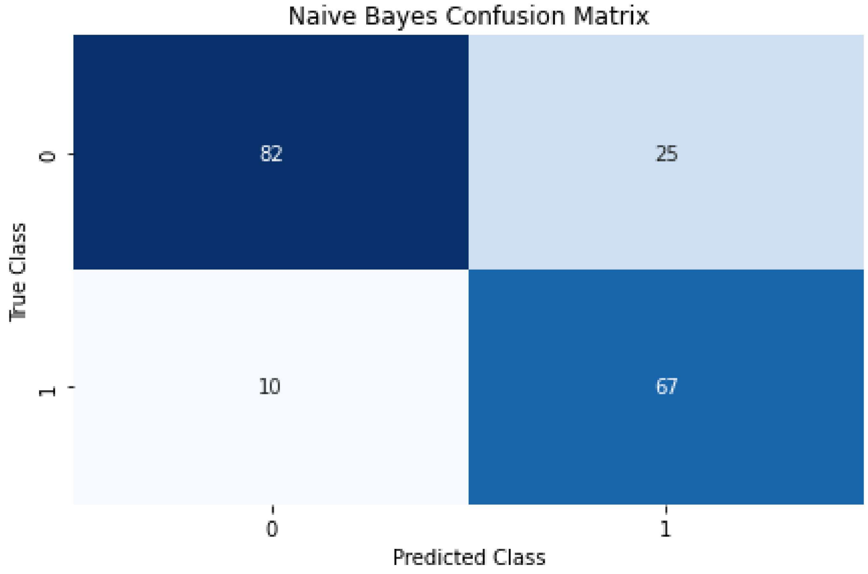
```python
# Initialize and train the Multinomial Naive Bayes classifier

# Ensure non-negative values in the feature vectors
x_train = np.maximum(0, X_train)
x_test = np.maximum(0, X_test)


nb_model = MultinomialNB()
nb_model.fit(x_train, Y_train)

# Make predictions on the test data
y_pred_nb = nb_model.predict(x_test)

# Evaluate the model
print(classification_report(Y_test, y_pred_nb))
print(confusion_matrix(Y_test, y_pred_nb))
print(f'ROC-AUC score : {roc_auc_score(Y_test, y_pred_nb)}')
print(f'Accuracy score : {accuracy_score(Y_test, y_pred_nb)}')
```

```
              precision    recall  f1-score   support

           0       0.73      0.87      0.79        77
           1       0.89      0.77      0.82       107

    accuracy                           0.81       184
   macro avg       0.81      0.82      0.81       184
weighted avg       0.82      0.81      0.81       184

[[67 10]
 [25 82]]
ROC-AUC score : 0.8182425051583929
Accuracy score : 0.8097826086956522
```

In [29]:

```
#Build the confusion matrix
matrix = confusion_matrix(Y_test, y_pred_nb, labels=[1,0])

print(matrix)

# Create pandas dataframe
df = pd.DataFrame(matrix)

# Create a heatmap
sns.heatmap(df, annot=True, cbar=None, cmap="Blues",fmt='.0f')
plt.title("Naive Bayes Confusion Matrix"), plt.tight_layout()
plt.ylabel("True Class"), plt.xlabel("Predicted Class")
plt.show()
```

```
[[82 25]
 [10 67]]
```



In [ ]: