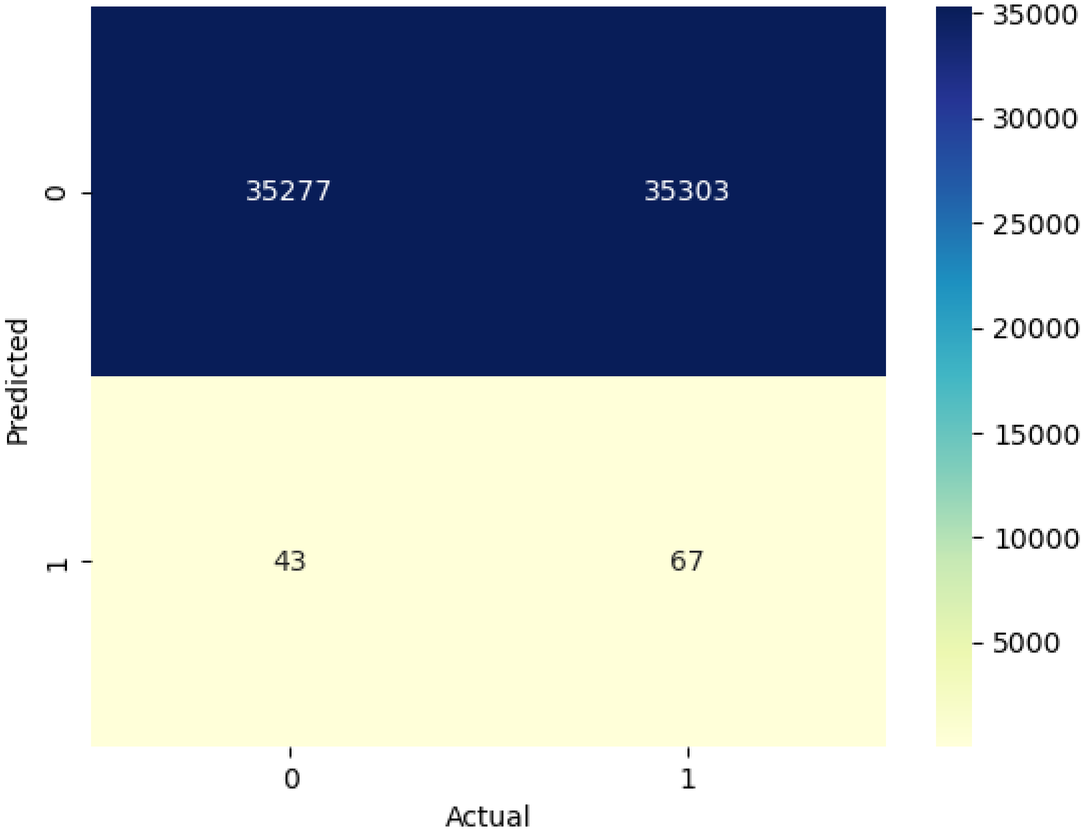
In [32]:

```python
from csv import reader
from random import seed
from random import randrange
from math import sqrt
from math import exp
from math import pi
import pandas as pd
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# Load a CSV file
def load_csv(filename):
	dataset = list()
	with open(filename, 'r') as file:
		csv_reader = reader(file)
		for row in csv_reader:
			if not row:
				continue
			dataset.append(row)
	return dataset

# Convert string column to float
def str_column_to_float(dataset, column):
	for row in dataset:
		row[column] = float(row[column].strip())

# Convert string column to integer
def str_column_to_int(dataset, column):
	class_values = [row[column] for row in dataset]
	unique = set(class_values)
	lookup = dict()
	for i, value in enumerate(unique):
		lookup[value] = i
	for row in dataset:
		row[column] = lookup[row[column]]
	return lookup

# Split a dataset into k folds
def cross_validation_split(dataset, n_folds):
	dataset_split = list()
	dataset_copy = list(dataset)
	fold_size = int(len(dataset) / n_folds)
	for _ in range(n_folds):
		fold = list()
		while len(fold) < fold_size:
			index = randrange(len(dataset_copy))
			fold.append(dataset_copy.pop(index))
		dataset_split.append(fold)
	return dataset_split

# Calculate accuracy percentage
def accuracy_metric(actual, predicted):
	correct = 0
	for i in range(len(actual)):
		if actual[i] == predicted[i]:
			correct += 1
	return correct / float(len(actual)) * 100.0

# Evaluate an algorithm using a cross validation split
def evaluate_algorithm(dataset, algorithm, n_folds, *args):
	folds = cross_validation_split(dataset, n_folds)
```

```
	scores = list()
	stats = [0,0,0,0]
	for fold in folds:
		train_set = list(folds)
		train_set.remove(fold)
		train_set = sum(train_set, [])
		test_set = list()
		for row in fold:
			row_copy = list(row)
			test_set.append(row_copy)
		predicted = algorithm(train_set, test_set, *args)
		actual = [row[0] for row in fold]
		stat = stat0(actual, predicted)
		for i in range(4):
			stats[i] += stat[i]
		accuracy = accuracy_metric(actual, predicted)
		scores.append(accuracy)
	plot(stats)
	return scores

# Split the dataset by class values, returns a dictionary
def separate_by_class(dataset):
	separated = dict()
	for i in range(len(dataset)):
		vector = dataset[i]
		class_value = vector[0]
		if (class_value not in separated):
			separated[class_value] = list()
		separated[class_value].append(vector)
	return separated

# Calculate the mean of a list of numbers
def mean(numbers):
	return sum(numbers)/float(len(numbers))

# Calculate the standard deviation of a list of numbers
def stdev(numbers):
	avg = mean(numbers)
	variance = sum([(x-avg)**2 for x in numbers]) / float(len(numbers)-1)
	return sqrt(variance)

# Calculate the mean, stdev and count for each column in a dataset
def summarize_dataset(dataset):
	summaries = [(mean(column), stdev(column), len(column)) for column in zip(*
	del(summaries[0])
	return summaries

# Split dataset by class then calculate statistics for each row
def summarize_by_class(dataset):
	separated = separate_by_class(dataset)
	summaries = dict()
	for class_value, rows in separated.items():
		summaries[class_value] = summarize_dataset(rows)
	return summaries

# Calculate the Gaussian probability distribution function for x
def calculate_probability(x, mean, stdev):
	exponent = exp(-((x-mean)**2 / (2 * stdev**2 )))
	return (1 / (sqrt(2 * pi) * stdev)) * exponent

# Calculate the probabilities of predicting each class for a given row
def calculate_class_probabilities(summaries, row):
	total_rows = sum([summaries[label][0][2] for label in summaries])
	probabilities = dict()
```

```
        for class_value, class_summaries in summaries.items():
                probabilities[class_value] = summaries[class_value][0][2]/float(tot
                for i in range(len(class_summaries)):
                        mean, stdev, _ = class_summaries[i]
                        probabilities[class_value] *= calculate_probability(row[i],
        return probabilities

# Predict the class for a given row
def predict(summaries, row):
        probabilities = calculate_class_probabilities(summaries, row)
        best_label, best_prob = None, -1
        for class_value, probability in probabilities.items():
                if best_label is None or probability > best_prob:
                        best_prob = probability
                        best_label = class_value
        return best_label

# Naive Bayes Algorithm
def naive_bayes(train, test):
        summarize = summarize_by_class(train)
        predictions = list()
        for row in test:
                output = predict(summarize, row)
                predictions.append(output)
        return(predictions)

# Have the stat for class 0
def stat0 (actual, predicted):
    tp = 0
    fp = 0
    tn = 0
    fn = 0
    result = list()
    for i in range(len(actual)):
        if actual[i] == 0 and predicted[i] == 0:
            tp += 1
        if actual[i] == 1 and predicted[i] == 0:
            fp += 1
        if actual[i] == 0 and predicted[i] == 1:
            fn += 1
        if actual[i] == 1 and predicted[i] == 1:
            tn += 1
    result.append(tp)
    result.append(fp)
    result.append(tn)
    result.append(fn)
    return(result)

# Plot the heatmap
def plot(stat):
    data = [[0,0],[0,0]]
    data[0][0] = stat[0]
    data[0][1] = stat[1]
    data[1][0] = stat[2]
    data[1][1] = stat[3]
    hm = sn.heatmap(data = data, annot=True, cmap="YlGnBu", fmt='g')
    plt.xlabel('Actual')
    plt.ylabel('Predicted')
    plt.show(hm)



# Test Naive Bayes
seed(1)
```

```python
filename = 'Data2.csv'
dataset = load_csv(filename)
for i in range(len(dataset[0])):
        str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, 0)
# evaluate algorithm
n_folds = 5
scores = evaluate_algorithm(dataset, naive_bayes, n_folds)
print('Scores: %s' % scores)
print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```




```
Scores: [50.29707172160136, 49.738293959541664, 49.41292969302589, 50.53755835337389, 49.83731786674211]
Mean Accuracy: 49.965%
```

In [ ]:

In [ ]: