In [4]:
```
# Importing all modules or library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split, cross_val_score
```

In [9]:
```
input_file = 'data_multivar_nb.txt' # Giving the file path to read our data

# Load data from input file
data = np.loadtxt(input_file, delimiter=',')

# Here the speration of the data is done {X, y}
X, y = data[:, :-1], data[:, -1]
```

In [10]:
```
# Have already import the GaussianNB in importing part
# From navie bayes, which is from sklearn
# Create Navies Bayes Classifier
classifier = GaussianNB()

# Train the Classifier
classifier.fit(X,y)
```

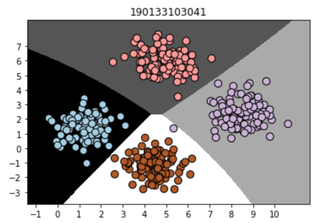
Out[10]:
```
GaussianNB()
```

In [11]:
```
# Predict the values for training data
y_pred = classifier.predict(X)

# Compute accuracy
accuracy = 100.0 * (y == y_pred).sum() / X.shape[0]
print("Accuracy of Naive Bayes classifier =", round(accuracy, 2), "%")
```

```
Accuracy of Naive Bayes classifier = 99.75 %
```

In [15]:
```
# Defining the Visualizer
def visualize_classifier(classifier, X, y):
    # Define the minimum and maximum values for X and Y
    # that will be used in the mesh grid
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0

    # Define the step size to use in plotting the mesh grid
    mesh_step_size = 0.01

    # Define the mesh grid of X and Y values
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size), np.arange(min_y, m
ax_y, mesh_step_size))

    # Run the classifier on the mesh grid
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.ravel()])

    # Reshape the output array
    output = output.reshape(x_vals.shape)

    # Create a plot
    plt.figure()

    # Choose a color scheme for the plot
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)

    # Overlay the training points on the plot
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1, cmap=plt.cm.Pa
ired)
        # Specify the boundaries of the plot
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())

    # Specify the ticks on the X and Y axes
    plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1), 1.0)))
    plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1), 1.0)))
    plt.title("190133103041")
    plt.show()
```

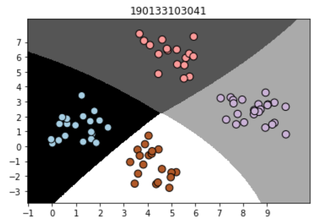
In [16]:
```
# Visualize the performance of the classifier
visualize_classifier(classifier, X, y)
```



In [17]:
```
#TRAIN DATA TEST DATA

# Cross validation

# Split data into training and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=3)

#Creating a classifier
classifier_new = GaussianNB()

#Train the classifier
classifier_new.fit(X_train, y_train)

# Predict the test data
y_test_pred = classifier_new.predict(X_test)
```

In [18]:
```
# compute accuracy of the classifier
accuracy = 100.0 * (y_test == y_test_pred).sum() / X_test.shape[0]
print("Accuracy of the new classifier =", round(accuracy, 2), "%")

# Visualize the performance of the classifier
visualize_classifier(classifier_new, X_test, y_test)

print('Predicted output :-', y_test_pred)
```

```
Accuracy of the new classifier = 100.0 %
```




```
Predicted output :- [0. 0. 1. 2. 3. 1. 0. 0. 2. 0. 1. 3. 3. 3. 3. 2. 1. 2. 2. 0. 2. 0. 3. 2.
 1. 2. 3. 3. 0. 0. 2. 1. 0. 2. 3. 0. 1. 0. 1. 0. 1. 3. 2. 1. 3. 2. 3. 3.
 0. 1. 2. 1. 3. 2. 3. 2. 3. 0. 3. 1. 0. 2. 0. 2. 2. 3. 3. 2. 0. 1. 2. 1. 1.
 0. 1. 0. 2. 2. 3. 2. 2.]
```

In [19]:
```
# Scoring functions

num_folds = 3

accuracy_values = cross_val_score(classifier, X, y, scoring='accuracy', cv=num_folds)
print("Accuracy: " + str(round(100*accuracy_values.mean(), 2)) + "%")

precision_values = cross_val_score(classifier, X, y, scoring='precision_weighted', cv=num_fo
lds)
print("Precision: " + str(round(100*precision_values.mean(), 2)) + "%")

recall_values = cross_val_score(classifier, X, y, scoring='recall_weighted', cv=num_folds)
print("Recall: " + str(round(100*recall_values.mean(), 2)) + "%")

f1_values = cross_val_score(classifier, X, y, scoring='f1_weighted', cv=num_folds)
print("F1: " + str(round(100*f1_values.mean(), 2)) + "%")
```

```
Accuracy: 99.75%
Precision: 99.76%
Recall: 99.75%
F1: 99.75%
```

In [ ]:
```
```