Mini Project - Palmer Penguins

By *Tiana Liang* and *Jazmin Alonso* from UCLA

Project Description

This prject is to determine a small set of measurements that are highly predictive of a penguin's species and to train several models to do the prediciton.

For training and evaluating the models, we will use the Palmer Penguins data set. The Palmer Penguins data set was collected by collected by Dr. Kristen Gorman and the Palmer Station, Antarctica LTER, a member of the Long Term Ecological Research Network. The data set contains measurements on three penguin species: Chinstrap, Gentoo, and Adelie.

Group Contributions

Tiana: Histogram, Feature selection, Complexity of C, Decision regions plot, Inspection of mistakes, Conclusion

Jazmin: Scatter plot, Cleaning data, Multinomial logistic regression and model

Both: Prepare data; Analyzed why prediction model made mistakes and in the plots as well; Explain the steps; Table; Decision tree.

Data Preparation

We need to import necessary libraries, and retrieve the 'Palmer Penguins' data.

In [1]:
```
# import necessary libraries
import pandas as pd
from matplotlib import pyplot as plt
from sklearn import tree, preprocessing
import numpy as np
```

In [2]:
```
# retrieve data
url = "https://philchodrow.github.io/PIC16A/datasets/palmer_penguins.csv"
"""
Original penguin data set on case we want to go back to it,
especially to look at gender.
Simplify the species names.
"""
penguins_orig = pd.read_csv(url)
# simplify the species names
penguins_orig["Species"] = penguins_orig["Species"].str.split().str.get(0)
```

In [3]:
```
"""
This penguins data set has all of the numerical variables
and only species(simpler names) and island for categorical variables,
since we only want to look at islands.
```

```
"""

penguins = penguins_orig[["Species","Island","Body Mass (g)",
                          "Culmen Length (mm)", "Culmen Depth (mm)",
                          "Flipper Length (mm)",
                     "Delta 15 N (o/oo)", "Delta 13 C (o/oo)"]]
penguins = penguins.dropna()
```

In [4]:
```
# Inspect the Palmer Penguins data set with selected columns above.
# We have 330 rows and 8 columns.
penguins
```

Out[4]:

	Species	Island	Body Mass (g)	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
1	Adelie	Torgersen	3800.0	39.5	17.4	186.0	8.94956	-24.69454
2	Adelie	Torgersen	3250.0	40.3	18.0	195.0	8.36821	-25.33302
4	Adelie	Torgersen	3450.0	36.7	19.3	193.0	8.76651	-25.32426
5	Adelie	Torgersen	3650.0	39.3	20.6	190.0	8.66496	-25.29805
6	Adelie	Torgersen	3625.0	38.9	17.8	181.0	9.18718	-25.21799
...	...	...	...	...	...	...	...	...
338	Gentoo	Biscoe	4925.0	47.2	13.7	214.0	7.99184	-26.20538
340	Gentoo	Biscoe	4850.0	46.8	14.3	215.0	8.41151	-26.13832
341	Gentoo	Biscoe	5750.0	50.4	15.7	222.0	8.30166	-26.04117
342	Gentoo	Biscoe	5200.0	45.2	14.8	212.0	8.24246	-26.11969
343	Gentoo	Biscoe	5400.0	49.9	16.1	213.0	8.36390	-26.15531

330 rows × 8 columns

Exploratory Data

Compute summary statistics and construct visualizations about the relationships between variables. 2 types of figures and 1 table is included.

Scatter Plot

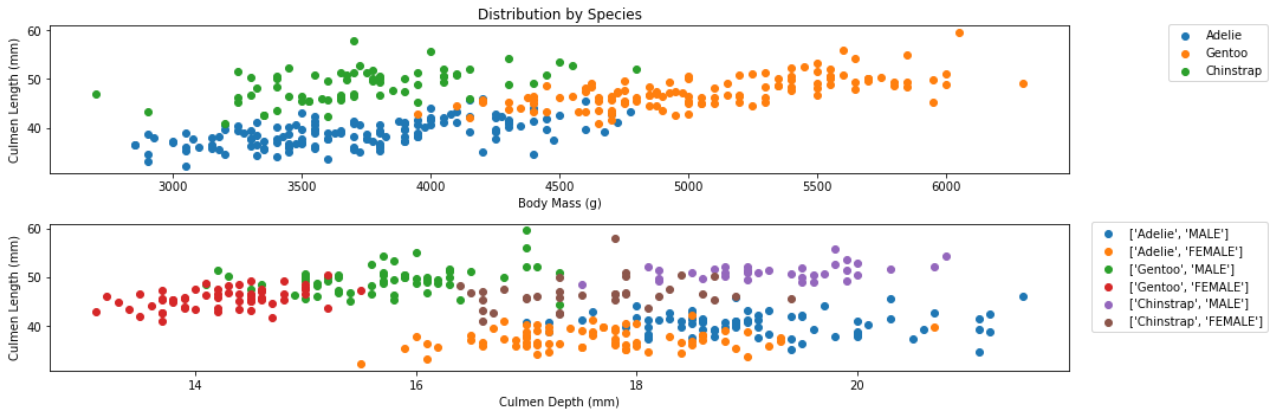
From two scatter plots, we will observe species and gender for label of the points.

The first scatter plot is to explore relationship between body mass and culmen length by species.

The second scatter plot is to explore relationship between culmen length and culmen depth by speceis and gender.

In [5]:
```
species = ["Adelie", "Gentoo","Chinstrap"]
gender = ["MALE","FEMALE"]
```

```
fig,ax = plt.subplots(2, figsize = (15,5))
ax[0].set(ylabel = "Culmen Length (mm)",
          xlabel = "Body Mass (g)")
ax[1].set(ylabel = "Culmen Length (mm)",
          xlabel = "Culmen Depth (mm)")


for i in species:
    y = penguins[penguins["Species"] == i]["Culmen Length (mm)"]
    x2 = penguins[penguins["Species"] == i]["Body Mass (g)"]
    ax[0].scatter(x2,y, label = i)
    #ax[1].scatter(x2,y, label = i)
    for j in gender:
        combin = np.where((penguins_orig["Species"] == i )
                          & (penguins_orig["Sex"] == j))
        for l in combin:
            x = penguins_orig.iloc[l]["Culmen Depth (mm)"]
            y = penguins_orig.iloc[l]["Culmen Length (mm)"]
            ax[1].scatter(x,y, label = [i,j])



ax[0].legend(bbox_to_anchor=(1.2, 1.05))
ax[1].legend(bbox_to_anchor=(1.2, 1.05))
ax[0].set(title = "Distribution by Species")


plt.tight_layout()
plt.show()
```



Scatter plot 1

- If we were to choose columns culmen length and body mass:
  - There would be a lot of prediction errors around 4000 g to 4800 g since there are a lot of overlap dots between species Adelie, Gentoo, and Chinstrap.
  - There is specifically more overlap when culmen length is also similar for Adelie and Gentoo, who we know have an island in common and have culmen length around mid 40s.
  - We also observe a strong positive relationship between culmen length and body mass in Gentoo penguins.

Scatter plot 2

- if we were to choose sex as our categorical variable, and observe factors culmen depth and culmen length there:
  - The Gentoo males show a lot of variations in culmen length and Chinstrap females show a lot of varaitions in culmen length. Adelie female penguins also have a lot of variations in culmen depth, with super high depth that it can be incorrectly labled as a Chinstrap male. This would cause more prediction errors. (We will explore this part deeper later with the histogram.)
  - From the plot, it seems that there is not a strong correlation between culmen length and culmen depth.

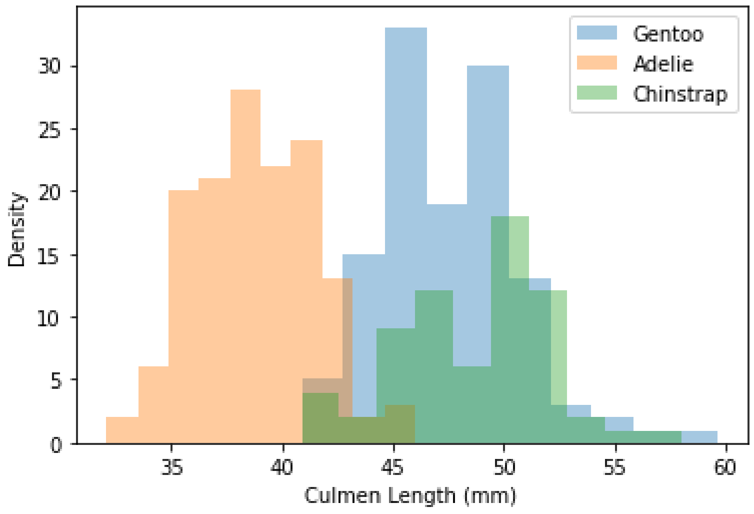
Histogram

Histogram 1: About Culmen Length

The histogram will explore the culmen length(mm) varied among different species.

In [6]:
```
fig, ax = plt.subplots(1)

species = ["Gentoo", "Adelie", "Chinstrap"]
for name in species:
    length = penguins[penguins["Species"] == name]["Culmen Length (mm)"]
    length = length[length.notnull()]
    ax.hist(length, label = name, alpha = 0.4)

ax.set(xlabel = "Culmen Length (mm)", ylabel = "Density")
ax.legend()
```

Out[6]:
```
<matplotlib.legend.Legend at 0x7fbcfb1c03d0>
```



Analysis of histogram 1

- Generally, Adelie penguins tend to have smaller culmen length, compared to Chinstrap penguins and Gentoo penguins.
- Adelie penguins

    - Adelie penguins have the maximum culmen length at around 45mm.
    - Adelie penguins have mode culmen length at around 38mm.
- Chinstrap penguins
    - Chinstrap penguins have the maximum culmen length at around 58mm, and minimum culmen length at 41mm.
    - Chinstrap penguins have mode culmen length at around 51mm.
- Gentoo penguins
    - Gentoo penguins have the maximum culmen length at around 59mm, and minimum culmen length at 41mm.
    - Gentoo penguins have mode culmen length at around 45mm.

Apply to modeling

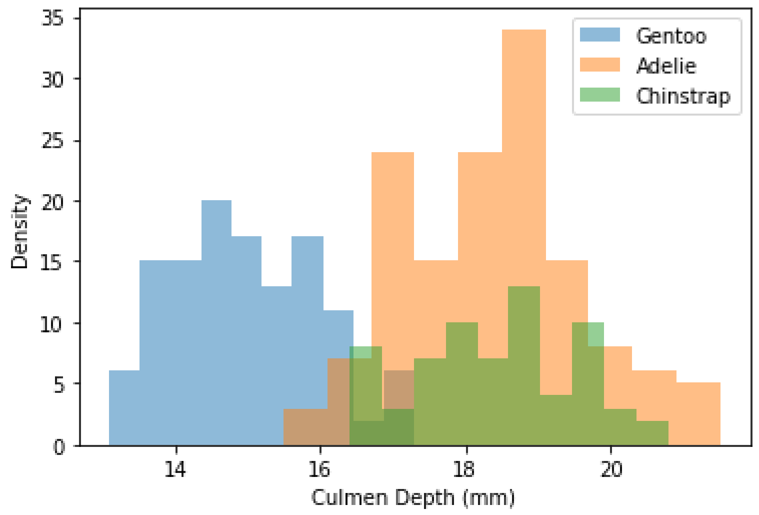
We can infer the possible reasons of mistakes from the overlapping area of different species' culmen length and some other statistics (such as mode) of culmen length, which will provide a more detailed and instinct knowledge about the culmen length relationship among these species.

Histogram 2: About Culmen Depth

The histogram will explore the culmen depth(mm) varied among different species.

In [7]:
```
fig, ax = plt.subplots(1)

species = ["Gentoo", "Adelie", "Chinstrap"]
for name in species:
    depth = penguins[penguins["Species"] == name]["Culmen Depth (mm)"]
    depth = depth[depth.notnull()]
    ax.hist(depth, label = name, alpha = 0.5)

ax.set(xlabel = "Culmen Depth (mm)", ylabel = "Density")
ax.legend()
```

Out[7]:
```
<matplotlib.legend.Legend at 0x7fbcfb287ee0>
```



Analysis of histogram 2

> - Generally, Gentoo penguins tend to have smaller culmen depth, compared to Chinstrap penguins and Adelie penguins.
> - Adelie penguins
>   - Adelie penguins have the maximum culmen depth at around 21.5mm, and minimum culmen depth at 15.5mm.
>   - Adelie penguins have mode culmen depth at around 19mm.
> - Chinstrap penguins
>   - Chinstrap penguins have the maximum culmen depth at around 20.5mm, and minimum culmen depth at 16.5mm.
>   - Chinstrap penguins have mode culmen depth at around 19mm.
> - Gentoo penguins
>   - Gentoo penguins have the maximum culmen depth at around 17.5mm.
>   - Gentoo penguins have mode culmen depth at around 14.5mm.
>
> **Apply to modeling**

Similar to the histogram about culmen length, we can infer the possible reasons of mistakes from the overlapping area of different species' culmen depth and some other statistics (such as mode) of it, which will provide a more detailed and instinct knowledge about the culmen depth relationship among these species.

Table

The table shows the mean of each feature, grouped by species and islands. The table allows us to have a more precise knowledge about the feature of each species on different islands, so that we can use it to:

- choose feature columns to build an effective model.
- know more about how the model "thinks" later.

In [8]:
```
penguins.groupby(["Species", "Island"]).mean()
```

Out[8]:

		Body Mass (g)	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
Species	**Island**						
Adelie	**Biscoe**	3709.659091	38.975000	18.370455	188.795455	8.823593	-25.918702
	Dream	3684.615385	38.401923	18.205769	190.096154	8.948276	-25.747446
	Torgersen	3720.000000	39.093333	18.426667	191.977778	8.792753	-25.757806
Chinstrap	**Dream**	3729.850746	48.788060	18.404478	195.671642	9.356155	-24.557869
Gentoo	**Biscoe**	5074.590164	47.506557	14.979508	217.147541	8.245338	-26.185298

Choose Feature Columns to Prepare for Modeling

Split data into train and test dataset (80% (training) and 20% (test) of the rows).

Then we check the score below to see which model performs best with different feature columns.

In [9]:
```
from sklearn.model_selection import train_test_split
np.random.seed(1111)
train, test = train_test_split(penguins, test_size = 0.2)
```

Clean Data

Encode Island names with built-in method.

In [10]:
```
def prep_penguin_data(data_df):
    df = data_df.copy()
    le = preprocessing.LabelEncoder()
    df['Island'] = le.fit_transform(df['Island'])
    df['Species'] = le.fit_transform(df['Species'])

    X = df.drop(['Species'], axis = 1)
    y = df['Species']

    return(X, y)

X_train, y_train = prep_penguin_data(train)
X_test,  y_test  = prep_penguin_data(test)
```

In [11]:
```
# inspect X_train
X_train
```

Out[11]:

	Island	Body Mass (g)	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
62	0	3600.0	37.6	17.0	185.0	8.58063	-26.21569
50	0	3500.0	39.6	17.7	186.0	8.46616	-26.12989
147	1	3475.0	36.6	18.4	184.0	8.68744	-25.83060
117	2	3775.0	37.3	20.5	199.0	9.49645	-26.36678
144	1	3000.0	37.3	16.8	192.0	9.06829	-25.85203
...	...	...	...	...	...	...	...
163	1	3775.0	51.7	20.3	194.0	8.68747	-24.38751
191	1	4500.0	53.5	19.9	205.0	10.02544	-24.90816
173	1	3400.0	48.5	17.5	191.0	9.42666	-24.26375
241	0	5000.0	45.1	14.5	215.0	7.63220	-25.46569
324	0	4725.0	47.3	13.8	216.0	8.25818	-26.23886

264 rows × 7 columns

Since we want to train a model to predict penguin species with certain features, we can see we have got `X_train` and `y_train` .

`X_train` is the training data without species names, wherein columns being perdictor variables later.

`y_train` is the training data with species names only, which will be target variable later.

`X_test` has the same columns as X_train, but with the test data.

`y_test` has the same column as y_train, but with the test data.

Select Columns Based on Score of Model

We systematically select features to achieve optimal predictive accuracy.

First, we see the score of the model with all columns.

In [12]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(max_iter = 4000)
LR.fit(X_train, y_train)
LR.score(X_train, y_train)
```

Out[12]:
```
1.0
```

In [13]:
```
from sklearn.model_selection import cross_val_score
cross_val_score(LR, X_train, y_train, cv = 5).mean()
```

Out[13]:
```
1.0
```

Our project is designed to select only three feature columns to determing the penguin species. So we need to systematically select features based on score to build an optimal model.

In [14]:
```
def check_column_score(cols):
    """
    Trains and evaluates a model via cross-validation
    on the columns of the data with selected indices
    """
    print("training with columns " + str(cols))

    LR = LogisticRegression(max_iter = 4000)
    return cross_val_score(LR, X_train[cols], y_train, cv = 5).mean()
```

In [15]:
```
combos = [['Island', "Body Mass (g)","Culmen Length (mm)"],
          ['Island', "Body Mass (g)","Culmen Depth (mm)"],
          ['Island', "Body Mass (g)","Flipper Length (mm)"],
          ['Island', "Body Mass (g)","Delta 15 N (o/oo)"],
          ['Island', "Body Mass (g)","Delta 13 C (o/oo)"],
          ['Island', "Culmen Length (mm)","Culmen Depth (mm)"],
          ['Island', "Culmen Length (mm)","Flipper Length (mm)"],
          ['Island', "Culmen Length (mm)","Delta 15 N (o/oo)"],
          ['Island', "Culmen Length (mm)","Delta 13 C (o/oo)"],
          ['Island',"Culmen Depth (mm)","Flipper Length (mm)"],
          ['Island',"Culmen Depth (mm)","Delta 15 N (o/oo)"],
          ['Island',"Culmen Depth (mm)","Delta 13 C (o/oo)"],
          ['Island',"Flipper Length (mm)","Delta 15 N (o/oo)"],
          ['Island',"Flipper Length (mm)","Delta 13 C (o/oo)"],
          ['Island', "Delta 15 N (o/oo)", "Delta 13 C (o/oo)"]]

for cols in combos:
    x = check_column_score(cols)
    print("CV score is " + str(np.round(x, 3)))
```

```
training with columns ['Island', 'Body Mass (g)', 'Culmen Length (mm)']
CV score is 0.966
training with columns ['Island', 'Body Mass (g)', 'Culmen Depth (mm)']
CV score is 0.799
training with columns ['Island', 'Body Mass (g)', 'Flipper Length (mm)']
CV score is 0.75
training with columns ['Island', 'Body Mass (g)', 'Delta 15 N (o/oo)']
CV score is 0.773
training with columns ['Island', 'Body Mass (g)', 'Delta 13 C (o/oo)']
CV score is 0.765
training with columns ['Island', 'Culmen Length (mm)', 'Culmen Depth (mm)']
CV score is 0.977
training with columns ['Island', 'Culmen Length (mm)', 'Flipper Length (mm)']
CV score is 0.955
training with columns ['Island', 'Culmen Length (mm)', 'Delta 15 N (o/oo)']
CV score is 0.951
training with columns ['Island', 'Culmen Length (mm)', 'Delta 13 C (o/oo)']
CV score is 0.966
training with columns ['Island', 'Culmen Depth (mm)', 'Flipper Length (mm)']
CV score is 0.837
training with columns ['Island', 'Culmen Depth (mm)', 'Delta 15 N (o/oo)']
CV score is 0.811
training with columns ['Island', 'Culmen Depth (mm)', 'Delta 13 C (o/oo)']
CV score is 0.913
training with columns ['Island', 'Flipper Length (mm)', 'Delta 15 N (o/oo)']
CV score is 0.845
training with columns ['Island', 'Flipper Length (mm)', 'Delta 13 C (o/oo)']
CV score is 0.936
training with columns ['Island', 'Delta 15 N (o/oo)', 'Delta 13 C (o/oo)']
CV score is 0.886
```

In [16]:

```
def test_column_score(cols):
    """
    Evaluates the model on the test set using the columns of the data
    with selected indices
    """
    print("testing with columns " + str(cols))
    LR = LogisticRegression(max_iter = 4000)
    LR.fit(X_train[cols], y_train)
    return LR.score(X_test[cols], y_test)
```

In [17]:

```
for cols in combos:
    y = test_column_score(cols)
    print("test score is " + str(np.round(y, 3)))
```

```
testing with columns ['Island', 'Body Mass (g)', 'Culmen Length (mm)']
test score is 0.985
testing with columns ['Island', 'Body Mass (g)', 'Culmen Depth (mm)']
test score is 0.773
testing with columns ['Island', 'Body Mass (g)', 'Flipper Length (mm)']
test score is 0.742
testing with columns ['Island', 'Body Mass (g)', 'Delta 15 N (o/oo)']
test score is 0.788
testing with columns ['Island', 'Body Mass (g)', 'Delta 13 C (o/oo)']
test score is 0.788
testing with columns ['Island', 'Culmen Length (mm)', 'Culmen Depth (mm)']
test score is 1.0
testing with columns ['Island', 'Culmen Length (mm)', 'Flipper Length (mm)']
test score is 1.0
testing with columns ['Island', 'Culmen Length (mm)', 'Delta 15 N (o/oo)']
test score is 1.0
testing with columns ['Island', 'Culmen Length (mm)', 'Delta 13 C (o/oo)']
test score is 0.985
testing with columns ['Island', 'Culmen Depth (mm)', 'Flipper Length (mm)']
```

```
test score is 0.788
testing with columns ['Island', 'Culmen Depth (mm)', 'Delta 15 N (o/oo)']
test score is 0.864
testing with columns ['Island', 'Culmen Depth (mm)', 'Delta 13 C (o/oo)']
test score is 0.985
testing with columns ['Island', 'Flipper Length (mm)', 'Delta 15 N (o/oo)']
test score is 0.909
testing with columns ['Island', 'Flipper Length (mm)', 'Delta 13 C (o/oo)']
test score is 0.985
testing with columns ['Island', 'Delta 15 N (o/oo)', 'Delta 13 C (o/oo)']
test score is 0.955
```

[***Discussion***]

Based on the score we derived above, we could see that the columns `Island` , `Culmen Length (mm)` , and `Culmen Depth (mm)` have the highest cv score of 0.977 and test score of 1.0. There is not a big difference between two scores, and they all demonstrate high accuracy of the prediction.

Note that some other "column combos" can also derive a fantastic model, such as `Island` , `Culmen Length (mm)` , and `Body mass (g)` with cv score of 0.966 and test score of 0.985 (high score and not overfitting).

We finally decide to go with `Island` , `Culmen Length (mm)` , and `Culmen Depth (mm)` as three feature columns for our model to work on.

Build Model - Logistic Regression

The first model we build is a Logistic Regression model. In the following parts, we will try to develop an optimal model and evaluate its performance. We will inspect mistakes, and if there is any, we will do analysis, which is very meaningful to machine learning.

Firstly, remove the columns we do not need when building our model.

In [18]:
```
def drop(data):
    """
    Remove columns we do not need for our model.
    """
    choice = ['Island', "Culmen Length (mm)","Culmen Depth (mm)"]
    data = data[choice]
    return data
```

In [19]:
```
X_train = drop(X_train)
X_test = drop(X_test)
```

Training and test data sets for X and y has been prepared respectively for the model !

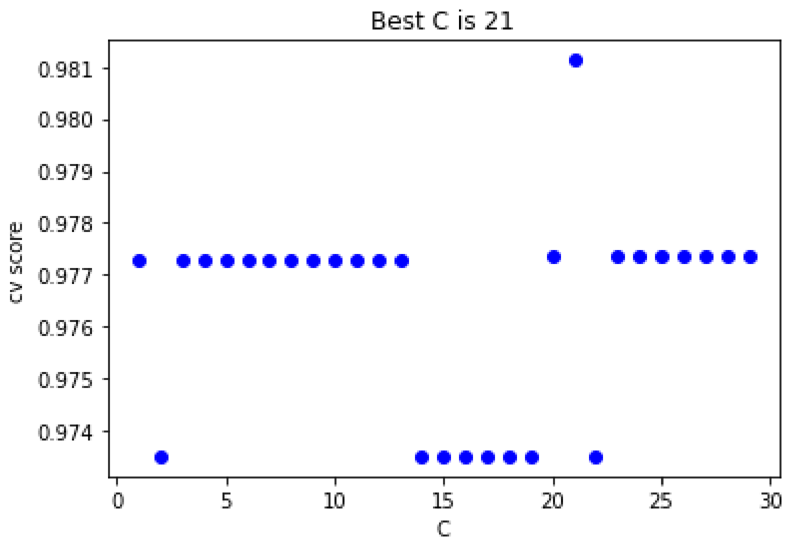
Logistic Regression Model

Here we check for the optimal `C` with plots because plot helps to see the result more distinctly. We choose `C` with best `cv_score` .

In [20]:
```
# ignore the warnings to present the plot(parts we want to see) more clearly.
import warnings
warnings.simplefilter('ignore')
```

In [21]:
```
fig,ax = plt.subplots(1)
best_cv = 0

for my_c in range(1,30):
    LR = LogisticRegression(C = my_c)
    cv_score = cross_val_score(LR, X_train, y_train, cv = 5).mean()
    ax.scatter (my_c,cv_score,color = "blue")
    if cv_score > best_cv:
        best_c = my_c
        best_cv = cv_score
ax.set(title = "Best C is " + str(best_c),
      xlabel = "C",
      ylabel = "cv score")
```

Out[21]:
```
[Text(0.5, 1.0, 'Best C is 21'), Text(0.5, 0, 'C'), Text(0, 0.5, 'cv score')]
```



Evaluate the model using the best_c we got above.

In [22]:
```
m = LogisticRegression(max_iter = 4000,C = best_c)
m.fit(X_train,y_train)
test_score = m.score(X_test,y_test)
train_score = m.score(X_train, y_train)
print("train score: "+ str(train_score))
print("test score: "+ str(test_score))
```

```
train score: 0.9924242424242424
test score: 1.0
```

We have observed that the train score has an improvement with optimal `C`, from around 0.977 to around 0.992.

So we see that finding optimal `C` is really helpful to improve the model performance.

Inspect the mistakes

Inspect a few instances in which the model gave the wrong answer on the test set via the confusion matrix.

The confusion matrix is a simple visualization of the model's predictions against truth. To create a confusion matrix, we first need to explicitly extract the predictions.

Firstly, we inspect the confusion matrix.

In [23]:
```
m = LogisticRegression(random_state = 0, solver = "liblinear")
```

In [24]:
```
m.fit(X_test, y_test)
y_test_pred = m.predict(X_test)
y_test_pred
```

Out[24]:
```
array([2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 0, 2, 0, 2, 0, 1, 2, 0, 0, 2, 0, 2,
       0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 1, 0, 0, 1, 2, 1, 0, 2, 0, 2, 0, 1,
       0, 2, 1, 0, 0, 1, 0, 0, 0, 2, 1, 0, 1, 1, 2, 2, 0, 0, 2, 1, 0, 0])
```

In [25]:
```
from sklearn.metrics import confusion_matrix
c = confusion_matrix(y_test, y_test_pred)
c
```

Out[25]:
```
array([[32,  0,  0],
       [ 1, 13,  0],
       [ 0,  0, 20]])
```

Analyze the mistakes

First, we explore which penguin is being misclassified.

In [26]:
```
mistakes = X_test[y_test != y_test_pred]
mistake_labels = y_test[y_test != y_test_pred]
mistake_preds = y_test_pred[y_test != y_test_pred]
```

In [27]:
```
m_plots = len(mistake_labels)
for i in range(m_plots):
    falseDF = penguins.loc[list(mistakes.index)][["Species","Island",
                                                  "Culmen Length (mm)",
                                                  "Culmen Depth (mm)"]]
    falseDF["Prediction"] = list(mistake_preds)
falseDF
```

Out[27]:

	Species	Island	Culmen Length (mm)	Culmen Depth (mm)	Prediction
209	Chinstrap	Dream	49.3	19.9	0

See the table presenting mean values of culmen length and culmen depth, grouped by species and islands.

We have created a table similar to the one below before, with more variables included. In this part, we filter necessary columns to make the table more readable.

In [28]:
```
def penguin_summary_table(group_cols,value_cols):
    """
    Computes the mean values of culmen length and culmen depth,
    grouped by species and islands.
    Uses the output table to inspect the possible reasons of prediction errors.

    parameters:
    group_cols: the group_by columns
    value_cols: the columns to compute mean value

    """
    x = penguins.groupby(group_cols)[value_cols].aggregate([np.mean]).round(2)
    return(x)
penguin_summary_table(["Species","Island"],
                      ["Culmen Length (mm)","Culmen Depth (mm)"])
```

Out[28]:

		Culmen Length (mm)	Culmen Depth (mm)
		mean	mean
Species	**Island**		
Adelie	**Biscoe**	38.98	18.37
	Dream	38.40	18.21
	Torgersen	39.09	18.43
Chinstrap	**Dream**	48.79	18.40
Gentoo	**Biscoe**	47.51	14.98

[***Discussion on mistakes***]

Our model misclassifies Chinstrap as Adelie. Through the plot we see there is a overlap between the culmen depth of Adelie penguins and Chinstrap penguins (from 16.5mm to 20.5mm, approximately). The misclassified poor penguin has the culmen depth of 19.9, which is far from the mode and mean of the culmen depths of Chinstrap penguins, so our model mistakenly bring it into the Adelie family.

Decision Regions

Now we do decision regions so we are able know how our training data should be predicting compared the testing data. The plots will be based by islands since that is our qualitative variable.

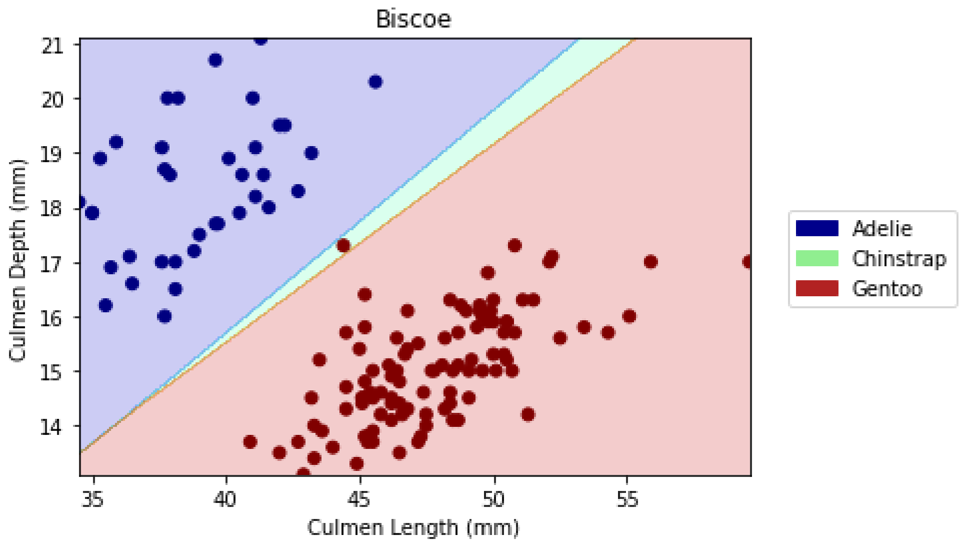
The shaded regions are the predictions of species and the points are the testing data points.

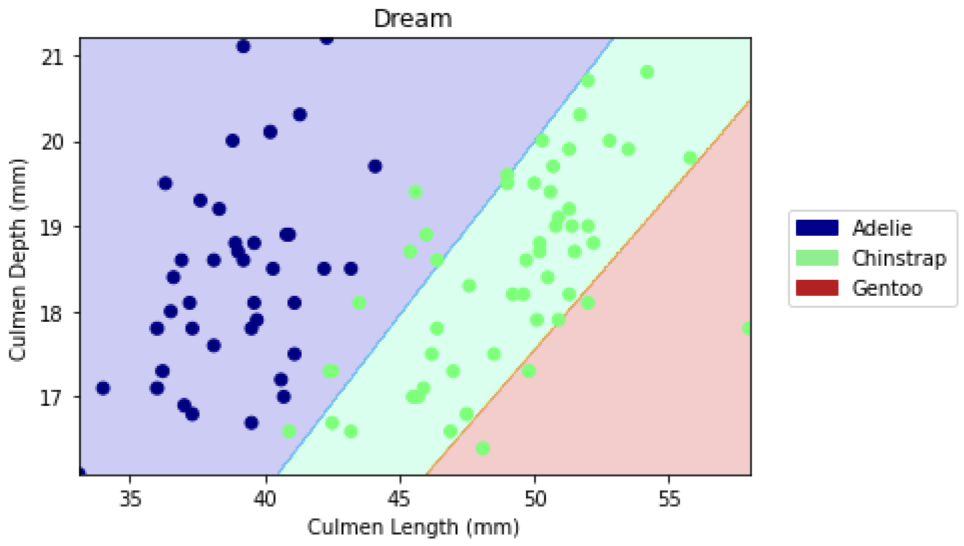
In [37]:

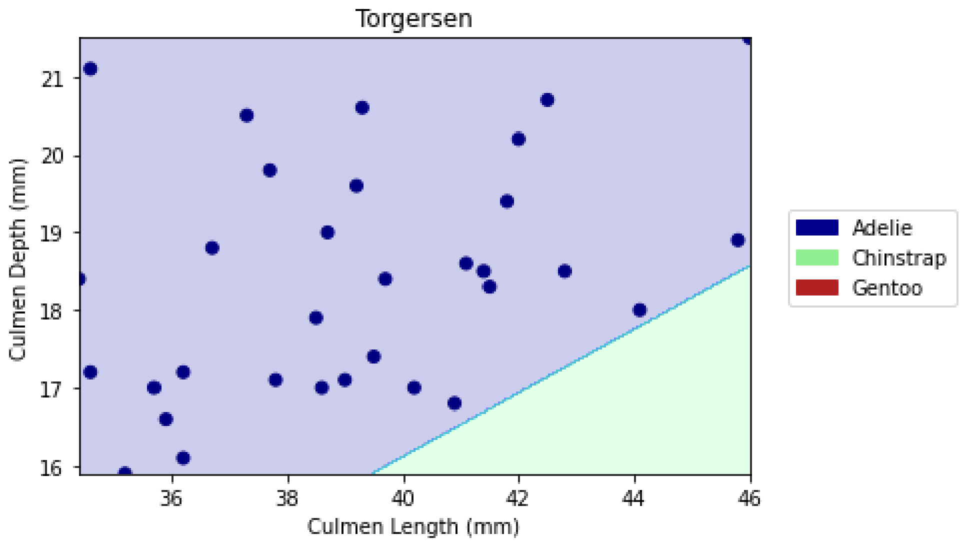
```
import matplotlib.patches as mpatches
def plot_regions(c, X, y, island):
    """
    Plots the decision regions that a model predicts as a grid that
    uses Culmen Length(horizontal axis) and Culmen Depth(vertical axis).
    May specify the sex of the penguin to
    visualize decisions of the model for this category.

    parameters:
    c: the model
    X: predictor variable
    y: target variable
    island: the island the penguins live on,
    0 represents Biscoe,
    1 represents Dream,
    2 represents Torgersen.
    """

    # for convenience, give names to the two
    # columns of the data
    x0 = X["Culmen Length (mm)"]
    x1 = X["Culmen Depth (mm)"]

    # create a grid
    grid_x = np.linspace(x0.min(),x0.max(),501)
    grid_y = np.linspace(x1.min(),x1.max(),501)
    xx, yy = np.meshgrid(grid_x, grid_y)

    # extract model predictions, using the
    # np.c_ attribute to join together the
    # two parts of the grid.
    # array.ravel() converts an multidimensional
    # array into a 1d array, and we use array.reshape()
    # to turn the resulting predictions p
    # back into 2d

    XX = xx.ravel()
    YY = yy.ravel()
    XY = np.c_[XX, YY]

    p = c.predict(np.insert(XY,0,island,axis = 1))
    p = p.reshape(xx.shape)

    # create the plot
    fig, ax = plt.subplots(1)


    # use contour plot to visualize the predictions
    ax.contourf(xx, yy, p, cmap = "jet", alpha = 0.2,vmin = 0, vmax = 2)

    # plot the data
    ax.scatter(x0, x1, c = y, cmap = "jet",vmin = 0, vmax = 2)

    # set the labels
    ax.set(xlabel = "Culmen Length (mm)",
           ylabel = "Culmen Depth (mm)")

    # set the titles
    if (island == 0):
        ax.set(title = "Biscoe")
    elif (island == 1):
        ax.set(title = "Dream")
    elif(island == 2):
        ax.set(title = "Torgersen")

    # set the legend
    colors = {
        "Adelie": "darkblue",
        "Chinstrap": "Lightgreen",
        "Gentoo":"firebrick"
    }
    patches = [mpatches.Patch(color = colors[key],
                              label = key) for key in colors]
    fig.legend(handles = patches, bbox_to_anchor = (1.15, 0.6))
```

In [30]:

```
for idx in range(3):
    plotX = X_train[X_train['Island'] == idx]
    plotY = y_train[X_train['Island'] == idx]
    plot_regions(m, plotX, plotY,idx)
```







Interpret plot & **Analyze mistakes**

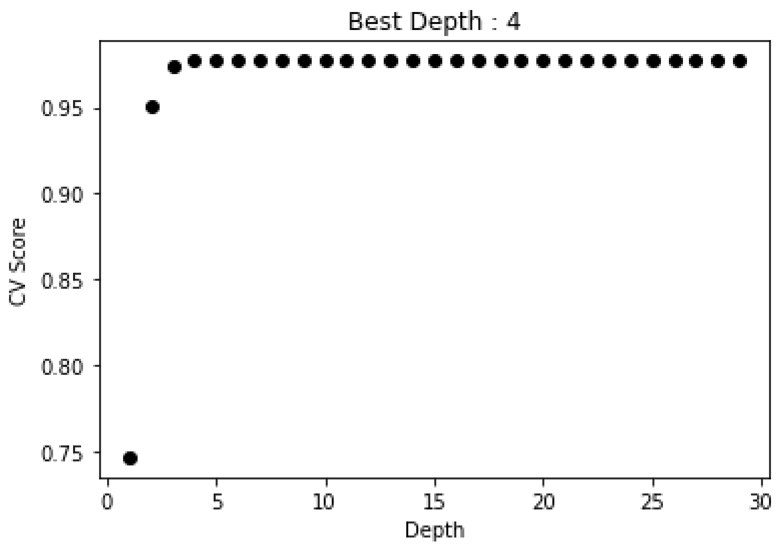
- The plot works on the training data, showing errors of our model more distinctly. We analyze the mistakes together with the table of mean values before.
- In island **Biscoe**, there is a shaded region of Chinstrap, which is wrongly predicted by model(predict Gentoo as Chinstrap).

    - This poor penguin is misclassfied because it has a culmen depth of 17mm, while the mean culmen depth of Gentoo penguins is around 14.9mm and the mean culmen depth of Chinstrap is around 18mm.
- In island **Dream**, thre is a shaded region of Gentoo, which is wrongly predicted by our model. Our model wrongly predicts some Chinstrap penguins as Adelie penguins and Gentoo penguins.
    - Chinstrap penguins that are misclassfied as Adelie penguins are usually with culmen lengths of 40mm-42mm. Mean culmen length of Adelie is around 40mm. Chinstrap and Adelie have similar mean culmen depths.(around 18mm)
    - Chinstrap penguins that are misclassified as Gentoo penguins are usually with culmen depths lower than 18mm. Chinstrap penguins and Gentoo penguins have similar culmen lengths. (47mm-48mm)
- In island **Torgersen**, we can see thare is no penguins classified mistakenly. Nice!

Build Model - Decision Tree

The second model we build is a Decision Tree model using the same feature columns. In the following parts, we will try to develop an optimal model and evaluate its performance. We will also use a decision region plot to see how model "thinks".

In [31]:
```
fig, ax = plt.subplots(1)

best_score = 0

for d in range(1,30):
    T = tree.DecisionTreeClassifier(max_depth = d)
    cv_score = cross_val_score(T, X_train, y_train, cv=5).mean()
    ax.scatter(d, cv_score, color = "black")
    if cv_score > best_score:
        best_depth = d
        best_score = cv_score

l = ax.set(title = "Best Depth : " + str(best_depth),
       xlabel = "Depth",
       ylabel = "CV Score")
```



Build the Decision Tree Classifier Model with best_depth.

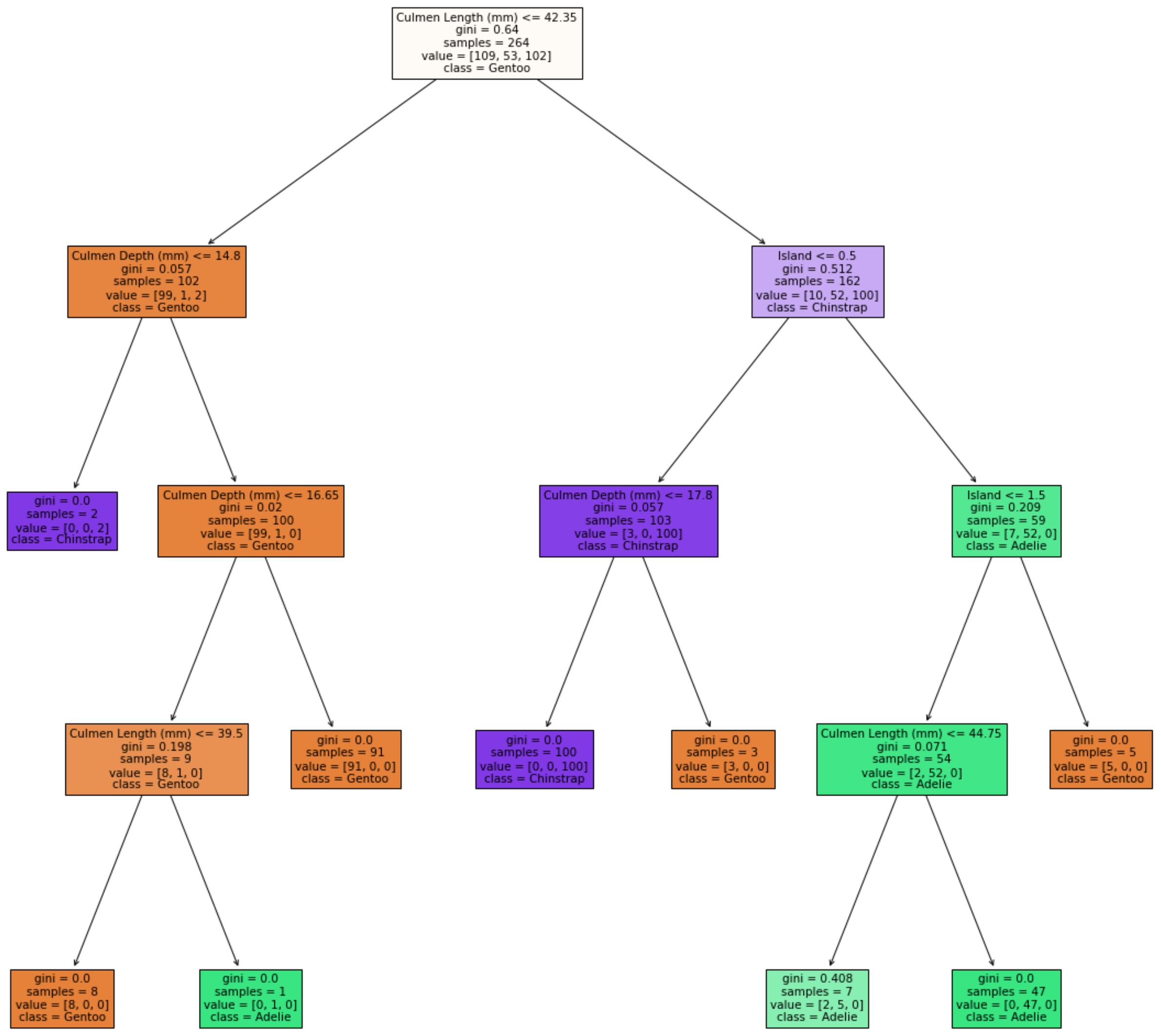
In [32]:
```
T = tree.DecisionTreeClassifier(max_depth = best_depth)
T.fit(X_train, y_train)
T.score(X_test, y_test), T.score(X_train, y_train)
```

Out[32]:
```
(1.0, 0.9924242424242424)
```

We got that the test score of this model is 1.0, and the train score of this model is around 0.992.

Next step: Visualize the logic of our model below.

In [33]:
```
fig, ax = plt.subplots(1, figsize = (20, 20))
p = tree.plot_tree(T, filled = True,
                   feature_names = X_test.columns,
                   class_names= species)
```



Inspect Decision Logic

First, our decision tree splits the penguins by checking whether the culmen length is less than or equal to 42.35mm.

- Then, if the culmen length is less than or equal to 42.35mm, the model checks whether the culmen depth is less than or equal to 14.8mm.
    - If it is then it is predicted as a Chinstrap. However, if it is less than 16.65 the culmen depth (mm), it then looks at the culmen length and if it is no more than 39.5 (mm) it goes down to be classified as either a Gentoo 88.9%(8/9) of the time or Adelie penguin about 11.1%(1/9) of the other time.
    - If it is just less than 16.65 the culmen depth and the culmen length is more than 39.5 (mm), it is predicted as Gentoo.
- Above is the analysis of the left branch of the tree. The right branch of the tree is with the same logic. The percentage of prediction possibility is derived based on value in the box. For example, in the orange box checking (culmen length <= 39.5), value of [8,1,0] means the possibility of classifying the penguin as Gentoo is 8/(8+1), that is around 88.9%.

Inspect mistakes

In order to look at the prediction mistakes in our decision tree classifier we will construct a confusion matrix.

In [34]:
```
from sklearn.metrics import confusion_matrix
#training data prediction mistakes
tree_train_pred = T.predict(X_train)
c = confusion_matrix(y_train,tree_train_pred)
c
```

Out[34]:
```
array([[107,   2,   0],
       [  0,  53,   0],
       [  0,   0, 102]])
```

In [35]:
```
#test data prediction mistakes
tree_test_pred = T.predict(X_test)
c = confusion_matrix(y_test,tree_test_pred)
c
```

Out[35]:
```
array([[32,  0,  0],
       [ 0, 14,  0],
       [ 0,  0, 20]])
```

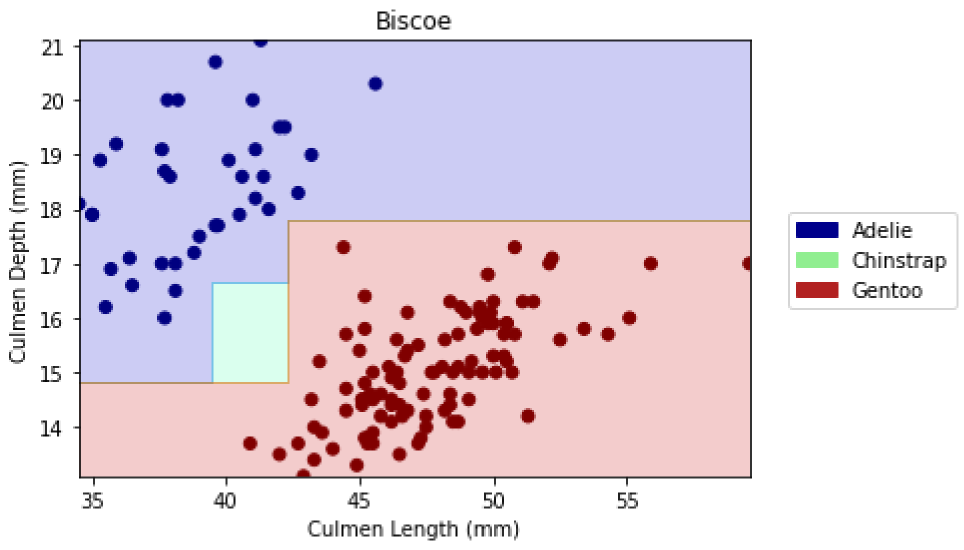
The confusion matrices has proved that there were no mistakes made in the model since there is no difference in the off diagonals which provide the misclassifications.

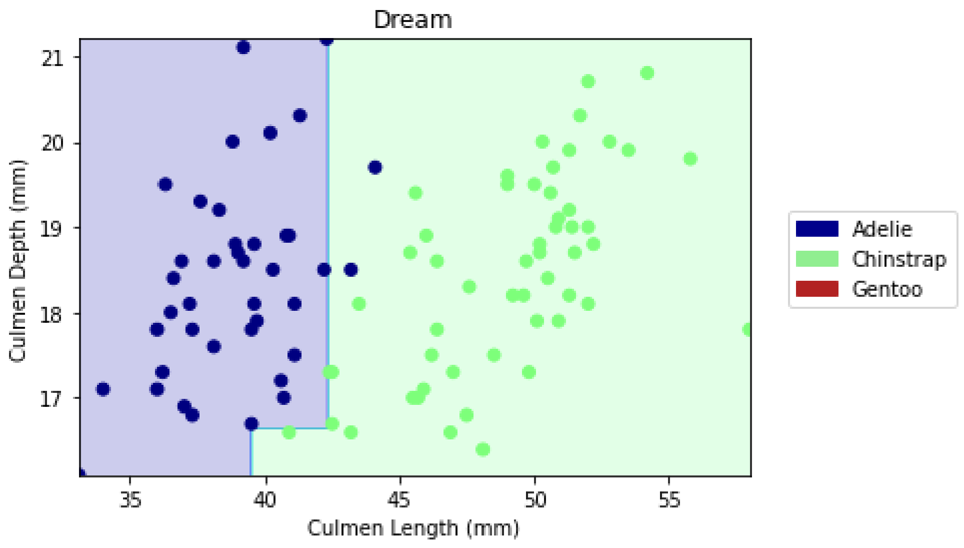
Decision Regions

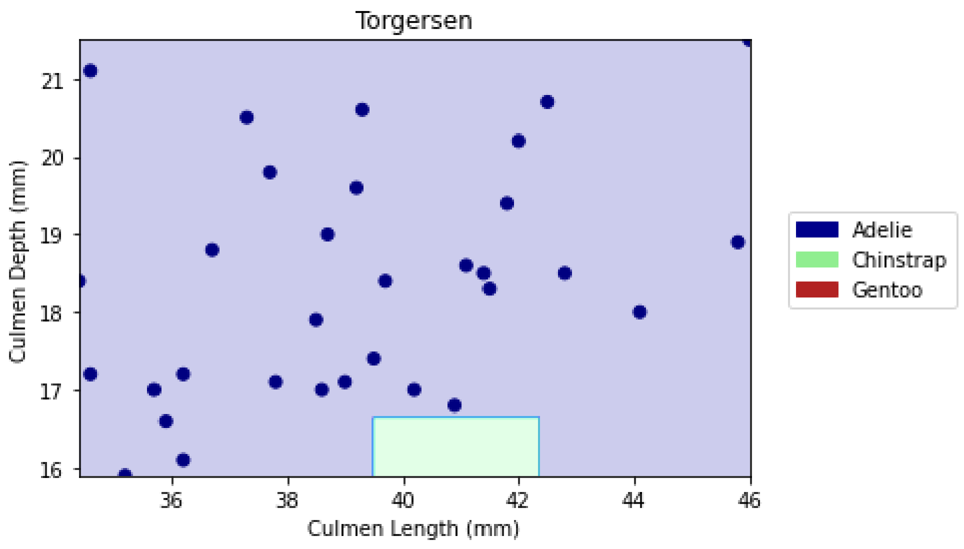
We further explore how Decision Tree Model "thinks" by plotting the decision region based on training data.

This part, we use the `plot_regions` function defined in the last part (Logistic Regression Modeling).

In [36]:
```
for idx in range(3):
    plotX = X_train[X_train['Island'] == idx]
    plotY = y_train[X_train['Island'] == idx]
    plot_regions(T, plotX, plotY,idx)
```







Seeing from the decision region plot, we know that on the Dream island, some Adelie penguins are misclassified as Chinstrap (blue dots in green area). These Adelie penguins have culmen length at around 43mm, which is way far from the mean value of Adelie's culmen lengths (around 38mm). Adelie penguins and Chinstrap penguins have similar mean value of culmen depth (around 18mm). So the "outlier" penguins are misclassified by our model.

Conclusion

Model Performance Analysis

Here we will go back to see and compare the performance of each model. The model performace(test dataset score) is listed below. We can see that both models achieve 1.0, the best score a model can achieve. From the excellent performance of these models, we could know the models we build are effective and the feature columns we choose to apply in the model are effective, too.

Model	Test Score
Logistic Regression	1.0
Decision Tree Classifier	1.0

Both models are good at classifying penguins.

In conclusion, we have built two powerful models in this project with three proper feature columns(island, culmen length, and culmen depth).