Group Project

*By

- Jonathan Martinez
- Ross Perry
- Tachin Ruangkriengsin

Group Contributions Statement

- For the Data Acquisition, Data Preparation, Splitting the data into train and test, Exploratory analysis (table), and the Discussion, all of up works equally in tamdem in a group call.

For the Exploratory analysis, we each split the the creating of figures one for each of us and that in the model sections:

- Jonathan Martinez: Worked on the implementation of Random Forest model
- Ross Perry: Worked on the implementation of Decision Tree model
- Tachin Ruangkriengsin: Worked on the implementation of Multinomial Logistic Regression model

We meet up with others in a group call to go over all the parts in the projects again at the end.

Data Acquisition

In this section, we will acquire our data of penguins and read it into Python

In [2]:
```
# We do our standard imports
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
url = "https://philchodrow.github.io/PIC16A/datasets/palmer_penguins.csv"
penguins = pd.read_csv(url)
```

Let's see what our data of penguins looks like

In [4]:
```
penguins.head()
```

Out[4]:

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Cu Le
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	

Exploratory Analysis

In this section, we will explore our data using summary tables and plots. First of all, we will prepare our data.

Data preparation

In [3]:
```
#Changing the species name to one word
penguins["Species"] = penguins["Species"].str.split().str.get(0)

#Gathering the columns we think are relavant for model prediction
cols = ["Species", "Island", "Culmen Length (mm)", "Culmen Depth (mm)",
        "Flipper Length (mm)", "Body Mass (g)", "Sex", "Delta 15 N (o/oo)", "Delt
penguins = penguins[cols]

#Dropping the na values, we still have plenty of data to work with afterwords
penguins = penguins.dropna()
penguins
```

Out[3]:

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

325 rows × 9 columns

Splitting the data into train and test

In [4]:
```
#Converting the string data into numbers
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

le = preprocessing.LabelEncoder()

#Dropping species from the predictor dataframe
X=penguins.drop(["Species"],axis=1)
#Transforming sex variable into numerical values
X["Sex"] = le.fit_transform(X["Sex"])
#Transforming island variable into numerical values
X["Island"] = le.fit_transform(X["Island"])
y=penguins[["Species"]].copy()
#Transforming species into numerical values
y.Species = le.fit_transform(y.Species)
```

In [16]:
```
?train_test_split
```

In [20]:
```
#Creating the train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s

#Combining the train data into one dataframe to more easily use for exploratory d
train_data = pd.concat([pd.DataFrame(y_train), X_train], join = "outer", axis = 1
train_data.head()
```

Out[20]:

	Species	Island	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Sex	Delta 15 N (o/oo)	Delta 13 C (o/oo)
55	0	0	41.4	18.6	191.0	3700.0	2	8.35396	-26.27853
145	0	1	39.0	18.7	185.0	3650.0	2	9.22033	-26.03442
185	1	1	51.0	18.8	203.0	4100.0	2	9.23196	-24.17282
175	1	1	50.6	19.4	193.0	3800.0	2	9.28153	-24.97134
190	1	1	46.9	16.6	192.0	2700.0	1	9.80589	-24.73735

Now that we have split our data into training and test data, we inspect the train data and look for predictors that might have good distinctions between species groups. It is important to split the data first and only look at training data as looking at the testing data would result in overfitting the model

Doing exploratory analysis

In [8]:
```
#Creates summary tables based on the train_data groups and value columns
def penguin_summary_table(group_cols, value_cols):
    return train_data.groupby(group_cols)[value_cols].mean()

#Creating a big summary table to illustrate which predictors have seemingly clear
big_table =  penguin_summary_table(["Species", "Island", "Sex"],
                                   ["Flipper Length (mm)", "Body Mass (g)",
                                    "Delta 15 N (o/oo)", "Delta 13 C (o/oo)",
                                   "Culmen Length (mm)", "Culmen Depth (mm)"])
big_table
```

Out[8]:

Species	Island	Sex	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)	Culmen Length (mm)	Culmen Depth (mm)
0	0	1	187.100000	3392.500000	8.739583	-25.897827	37.225000	17.790000
		2	190.000000	4062.500000	8.909285	-25.904406	40.828571	19.000000
	1	1	187.764706	3332.352941	8.949915	-25.674036	36.929412	17.635294
		2	193.642857	4108.928571	8.953301	-25.785406	39.785714	19.085714
	2	1	189.222222	3375.000000	8.611121	-25.928396	37.105556	17.577778
		2	195.454545	4188.636364	8.942111	-25.713245	41.009091	19.363636
1	1	1	191.100000	3586.250000	9.213340	-24.581554	46.865000	17.620000
		2	200.320000	3995.000000	9.441868	-24.547006	51.080000	19.336000
2	0	0	217.000000	4875.000000	8.041110	-26.184440	44.500000	15.700000
		1	213.000000	4680.813953	8.153465	-26.120760	45.630233	14.216279
		2	222.159091	5513.636364	8.304926	-26.167285	49.563636	15.763636

What is immediately apparent from this table is that island is going to be a useful variable to predict species. For example, if the island of any given penguin is 2, that penguin can only be of species 0. The other two islands also only have 2 options instead of three. Even if we were only to include island number in the model, we would likely still find that our predictions were well above chance. The question now becomes: given the fact that we now only need to determine what of penguin we find on islands 0 and 1, what other predictors allow us to do this? To find out, we must now turn to the biometric data.

In [9]:
```
smaller_table =  penguin_summary_table(["Species"],
                                       ["Flipper Length (mm)", "Body Mass (g)",
                                        "Delta 15 N (o/oo)", "Delta 13 C (o/oo)",
                                       "Culmen Length (mm)", "Culmen Depth (mm)"])
smaller_table
```

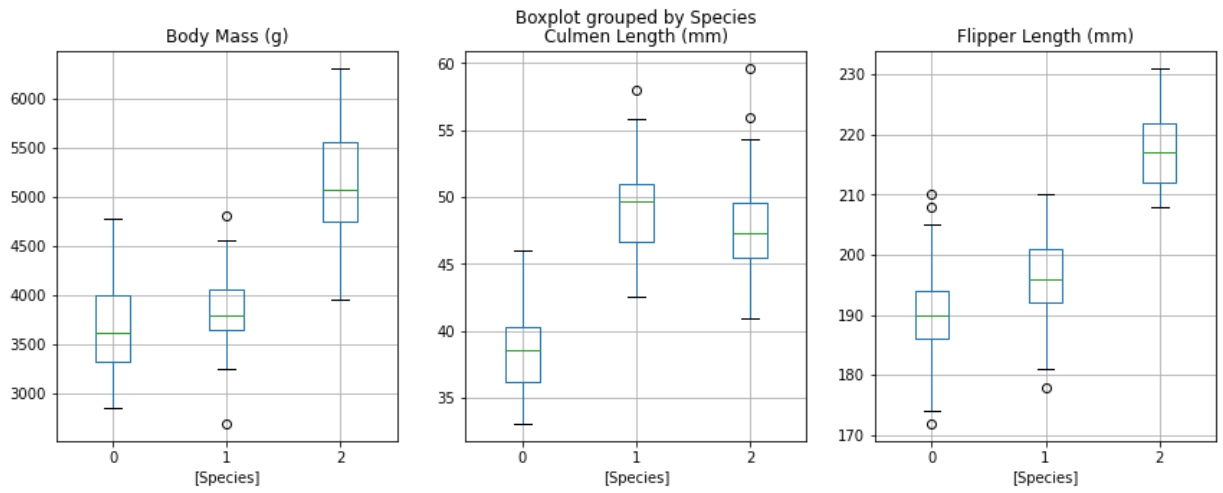
Out[9]:

Species	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)	Culmen Length (mm)	Culmen Depth (mm)
0	190.010638	3677.925532	8.833828	-25.825844	38.509574	18.278723
1	196.222222	3813.333333	9.340300	-24.562361	49.206667	18.573333
2	217.625000	5099.431818	8.227919	-26.144746	47.584091	15.006818

We can more clearly see the means between each species, which we will futher illustrate with plots

In [11]:
```
fig, ax = plt.subplots(1,3, figsize = (14,5))
train_data[["Species", "Body Mass (g)"]].boxplot(by = "Species", ax = ax[0])
train_data[["Species", "Culmen Length (mm)"]].boxplot(by = "Species", ax = ax[1])
train_data[["Species", "Flipper Length (mm)"]].boxplot(by = "Species", ax = ax[2]
```

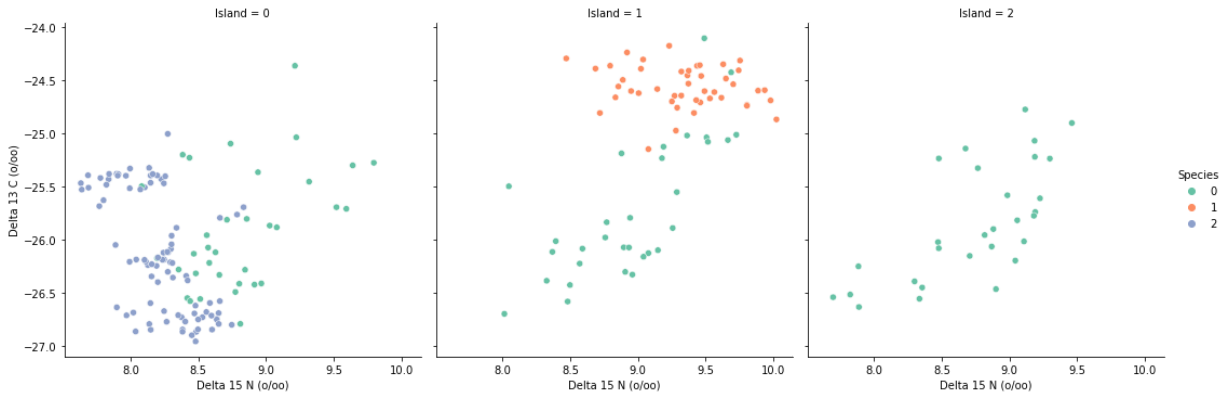
Out[11]:
```
<AxesSubplot:title={'center':'Flipper Length (mm)'}, xlabel='[Species]'>
```



These two boxplots clearly illustrate the distinctions between species.

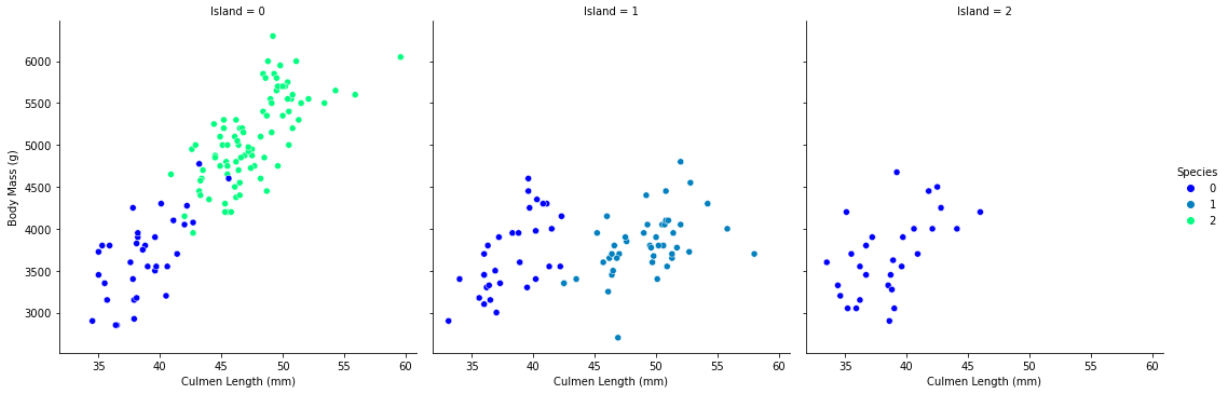
Additionally, we can see a similar species distinction as in the body mass plot, so we there might be strong correleation between both predictors. In this case we can opt to only use body mass, instead of using both.

In [66]:
```
fgrid_delta = sns.relplot(x = "Delta 15 N (o/oo)", y = "Delta 13 C (o/oo)", hue =
                          col="Island", sizes=(100, 1000), palette= "Set2")
```



The columns labeled "Delta" refer to the bloodwork of the penguins, which are very distinct from the other biometrics measured from the penguins in the dataset. They are graphed in a scatterplot by island above, and one can see that on island 0, Delta 15 is a better predictor, whereas on island 1, Delta 13 is a better predictor. For comparison, we also created a similar figure based on the biometric measurement data:

In [63]:
```
fgrid_metrics = sns.relplot(x = "Culmen Length (mm)", y = "Body Mass (g)", hue =
                          col="Island", sizes=(100, 1000), palette= "winter")
```



It's clear to see that these measurements are distinctly less interspersed based solely on visual inspection of these plots. For this reason, we selected Culmen Length and Body Mass as predictor variables along with Island.

In [13]:
```
#Dropping the predictors we didnt use from the train and test data
cols = ["Island", "Culmen Length (mm)", "Body Mass (g)"]
X_train = X_train[cols]
X_test = X_test[cols]
X_train.head()
```

Out[13]:

	Island	Culmen Length (mm)	Body Mass (g)
309	0	52.1	5550.0
94	1	36.2	3300.0
257	0	44.4	5250.0
231	0	49.0	5550.0
255	0	48.4	5400.0

Modeling

In this section, we will build several machine models to predict the specis of penguins given the Island, Body Mass, and Culmen Length. Our models include:

- Decision Trees
- Multinomial Logistic Regression
- Random Forest

In [7]:
```
# First, we will import necessary modules to be used in our models
from sklearn.model_selection import cross_val_score
from sklearn import tree
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.metrics import confusion_matrix
```

First model: Decision Trees

Decision Tree Classifiers create what amounts to a flowchart that leads a piece of data to a possible end state. Each note asks a binary question about the data and classifies it accordingly, ultimately leading to a prediction of which type of data a given piece is.

In [16]:
```
#initializing variables
N = 20
scores = np.zeros(N)
best_score = 0
current_av=0
#iterates over possible depths
for d in range(1,N):
    current_high=0
    #cross validates 50 times and adds scores together
    for i in range(50):
        T = tree.DecisionTreeClassifier(max_depth=d)
        current_av+=cross_val_score(T,X_train,y_train["Species"],cv=5).mean()
    #divides the sum by 50, generating an average score, compares against current
    current_av=current_av/50
    scores[d-1] = current_av
    if scores[d-1] > best_score:
        best_score = scores[d-1]
        best_depth = d
best_depth, best_score
```

Out[16]:
```
(3, 0.9840769468599022)
```

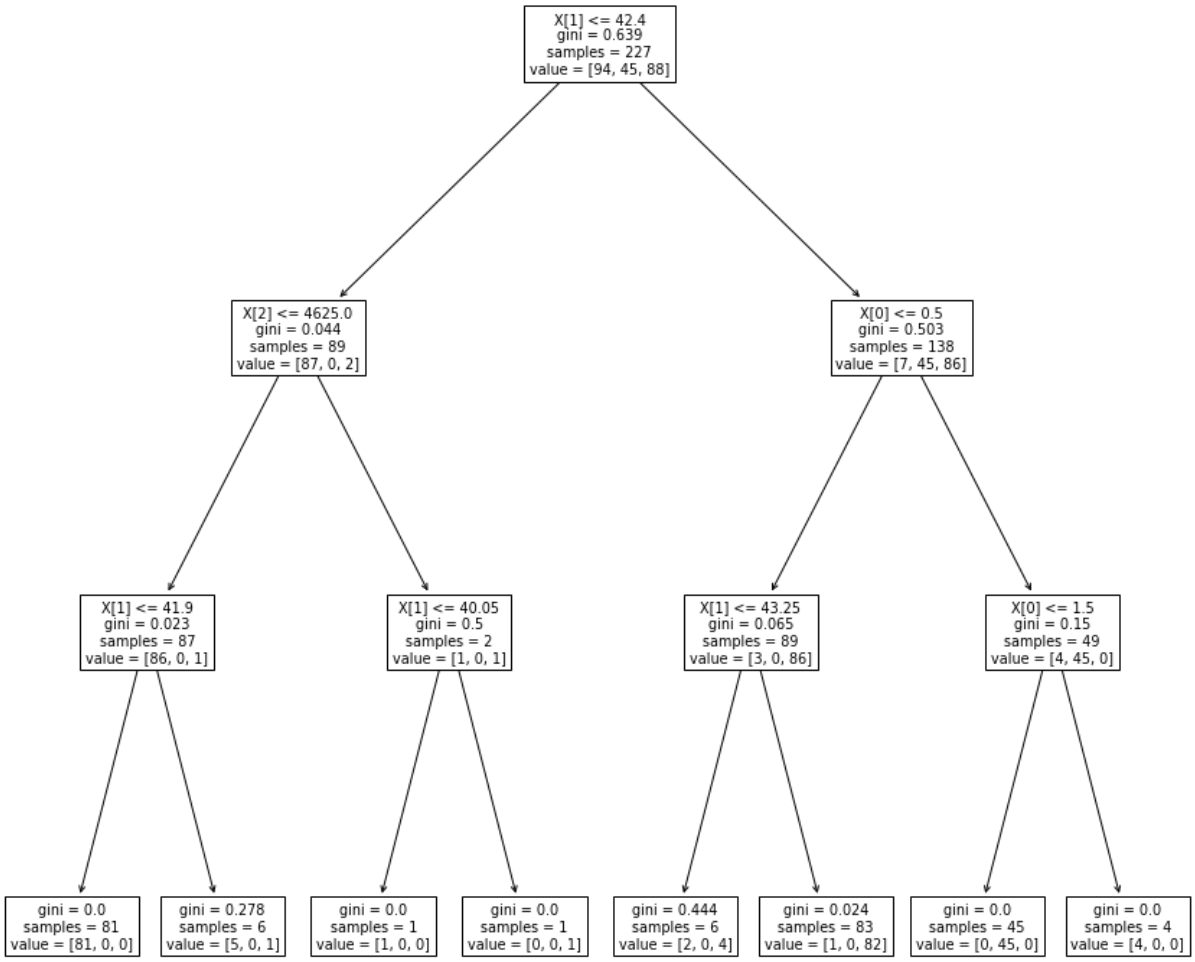
We struggled to find an ideal maximum depth with this model because it tended to vary wildly when each model was only tested once, so we began to average several models. Surprisingly, with 5 and 10 models the optimal depth seemed to vary as well, so we averaged even more scores until one consistently came out on top. We eventually were able to consistently determine optimal depth at 3

In [22]:
```
#fitting the model
dt_model=tree.DecisionTreeClassifier(max_depth=3)
dt_model.fit(X_train, y_train)
print(dt_model.score(X_train, y_train))
print(dt_model.score(X_test,y_test))
```

```
0.986784140969163
0.9795918367346939
```

Here we test the model and find that its accuracy is higher for the training data, but still acceptably high for the test data.

In [18]:
```
#visualizes the output
plt.subplots(figsize = (15, 15))
tree.plot_tree(dt_model)
plt.show()
```

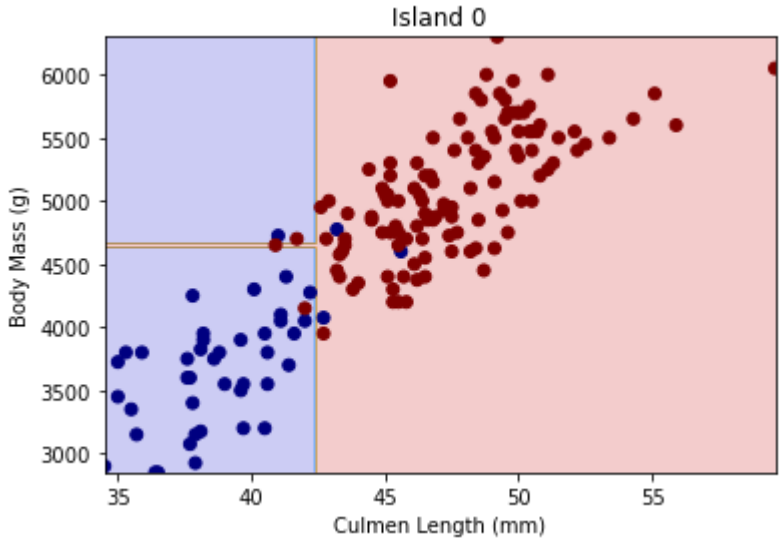


In [23]:
```
#constructing the confusion matrix
from sklearn.metrics import confusion_matrix
y_test_pred = dt_model.predict(X_test)
confusion_matrix(y_test,y_test_pred)
```

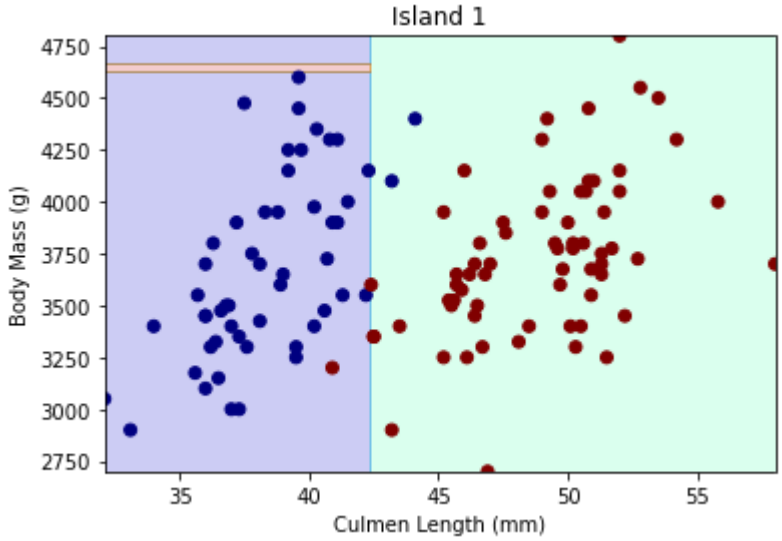
Out[23]:
```
array([[42,  0,  0],
       [ 2, 18,  0],
       [ 0,  0, 36]], dtype=int64)
```

In [129]:
```
def plot_regions(c, X, y, Island):
    """
    Arguments: An sklearn model c, test predictor datarame X,
    test values vector y, int Island
    This function makes and plot decision regions for given data X and y
    returns: Plots the data
    """

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

    # 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()

    #Need a third column representing the island number of the same size as XY
    XYZ = np.c_[np.zeros((251001,1))+Island, XX, YY]

    p = c.predict(XYZ)
    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)

    # plot the data
    ax.scatter(x0, x1,c = y[X["Island"] == Island]["Species"], cmap = "jet")

    ax.set(xlabel = "Culmen Length (mm)",
           ylabel = "Body Mass (g)",
           title= "Island " + str(Island))
```

In [130]:
```
plot_regions(dt_model, X, y, Island = 0)
```

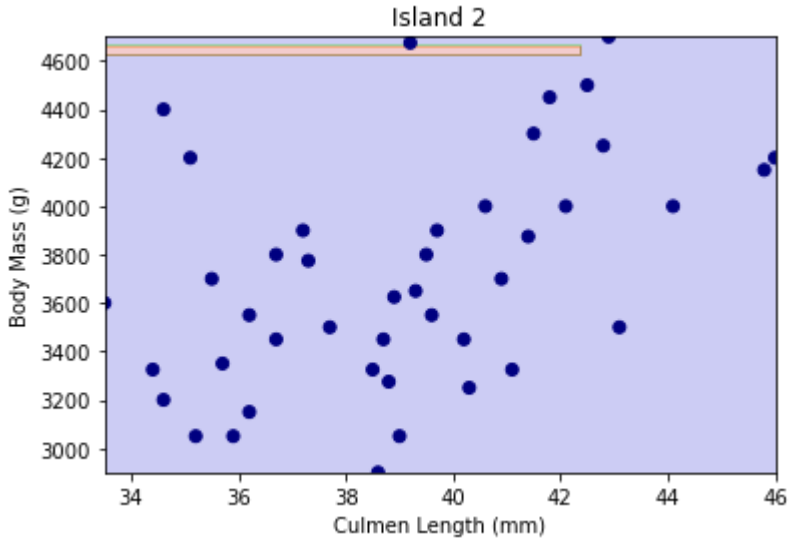


In [131]:
```
plot_regions(dt_model, X, y,Island = 1)
```



For some reason the decision region color is mismatched (which we couldn't figure out how to fix), but ignoring that we can still clearly see the disctions between regions

In [132]:
```
plot_regions(dt_model, X, y,Island = 2)
```



Here we can see the decision regions are very binary, which makes sense given that it is a decision tree classsifier

Second model: Multinomial Logistic Regression

Modified version of logistic regression that predicts a multinomial probability (i.e. more than two classes) for each input.

An important hyperparameter to tune for multinomial logistic regression is the penalty term, which by default, the LogisticRegression class uses the L2 penalty with a weighting of coefficients set to 1.0. Here, we will tune our model by changing the weighting of the coefficients in the penalty, which can be set via the “C” argument.

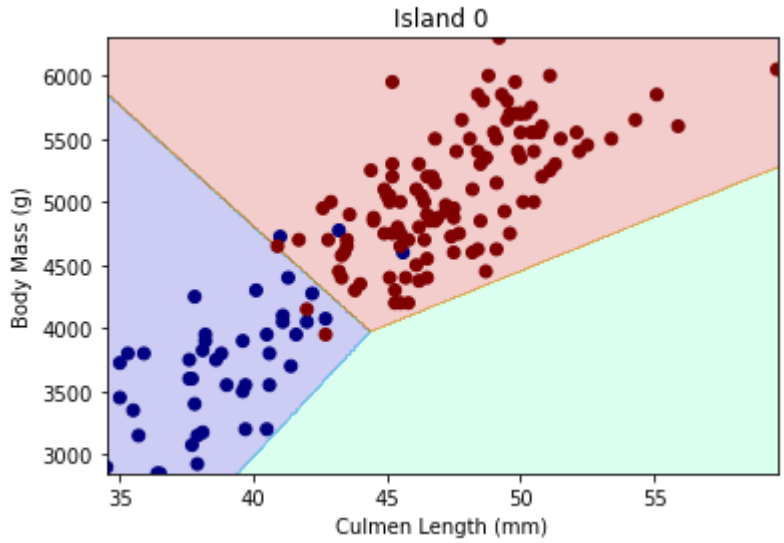
In [25]:
```
# We will use cross_validation method to estimate the model best complexity
# Here, we will vary the penalty weighting in the multinomial logistic regressior
N = 10
scores = np.zeros(N)
best_score = -np.inf
for d in range(1,N):
    # We will increment our weight by 0.1 from 0 to 1 to see the best penalty
    model_LR =  LogisticRegression(multi_class='multinomial', solver='lbfgs', per
    scores[d-1] = cross_val_score(model_LR,X_train,y_train["Species"],cv=5).mean(
    if scores[d-1] > best_score:
        best_score = scores[d-1]
        best_penalty_weight = 0.1*d
best_penalty_weight, best_score
```

Out[25]: (0.1, 0.9690821256038648)

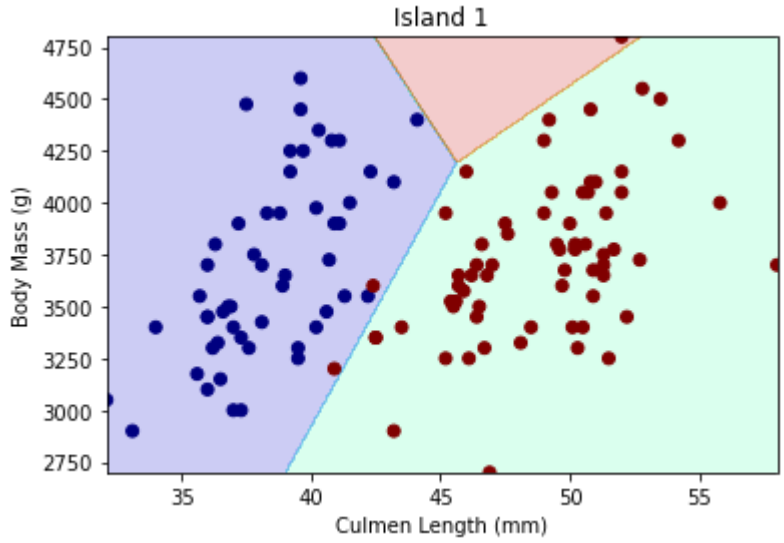
In [26]:
```
model_LR = LogisticRegression(multi_class='multinomial', solver='lbfgs', penalty=
model_LR.fit(X_train, y_train["Species"])
print(model_LR.score(X_train,y_train))
print(model_LR.score(X_test,y_test))
```

```
0.973568281938326
0.9489795918367347
```

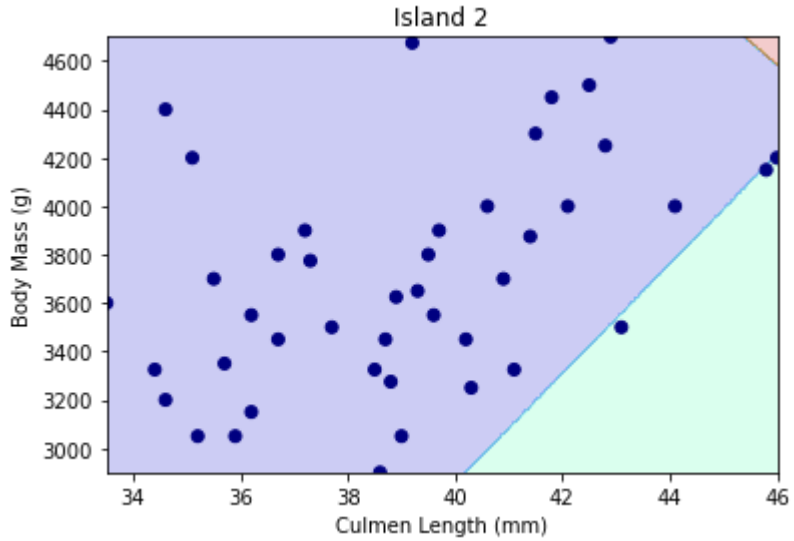
In [27]:
```
plot_regions(model_LR, X, y,
             Island = 0,
             title = "Island 0")
```



In [28]:
```
plot_regions(model_LR, X, y,
             Island = 1,
             title = "Island 1")
```



In [29]:
```
plot_regions(model_LR, X, y,
             Island = 2,
             title = "Island 2")
```



Third Model: Random Forest

Random forest work in a similar way to Decision trees. In fact, they are made of multiple decision trees, hence a forest. Each decision tree works only on a subset of the predictor data which lowers the correlation between them. It then aggregates and averages the decision trees to create a prediction.

In [30]:
```
#Doing cross validation to estimate best model complexity
#In the case of random forest, it would be the total number of trees

from sklearn.model_selection import cross_val_score
N = 20
scores = np.zeros(N)
best_score = -np.inf
for d in range(1,N):
    #We are going in increments of 10 because the number of trees can vary over a
    T = RFC(n_estimators=d*10, random_state = 0)
    scores[d-1] = cross_val_score(T,X_train,y_train["Species"],cv=5).mean()
    if scores[d-1] > best_score:
        best_score = scores[d-1]
        best_depth = d*10
best_depth, best_score
```

Out[30]:
```
(20, 0.9734299516908212)
```

In [24]:
```
# From the above we found 130 to be the best number of trees
model_RFC = RFC(n_estimators=130, random_state=0)
model_RFC.fit(X_train, y_train["Species"])
print(model_RFC.score(X_train,y_train["Species"]))
print(model_RFC.score(X_test,y_test))
```

```
1.0
0.9795918367346939
```

Actually, do to the randomness between the people running the cross validation score, we found that the best n estimators for random forest to vary

Next, we will construct the confusion matrix to see what kind of mistakes our model predict

In [25]:
```
y_test_pred = model_RFC.predict(X_test)
confusion_matrix(y_test, y_test_pred)
```

Out[25]:
```
array([[42,  0,  0],
       [ 2, 18,  0],
       [ 0,  0, 36]], dtype=int64)
```

In [26]:
```
mistakes = X_test[y_test["Species"] != y_test_pred].copy()
mistakes['True'] = y_test[y_test["Species"]!= y_test_pred]
mistakes['Prediction'] = y_test_pred[y_test["Species"] != y_test_pred]
mistakes
```

Out[26]:

	Island	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Sex	Delta 15 N (o/oo)	Delta 13 C (o/oo)	True	Prediction
174	1	43.2	16.6	187.0	2900.0	1	9.35416	-25.01185	1	0
182	1	40.9	16.6	187.0	3200.0	1	9.08458	-24.54903	1	0

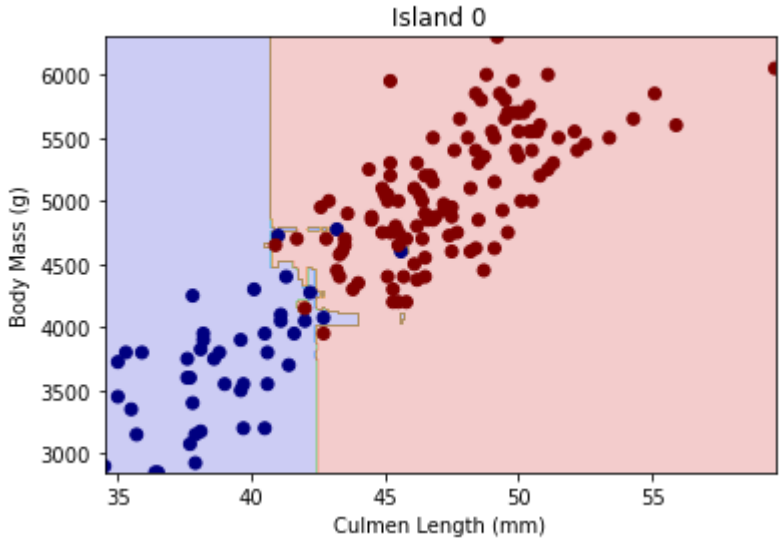
There was mistakes at row 260 and 236. We can take a look at row 260.

In [27]:
```
y_test.loc[182], X_test.loc[182]
```

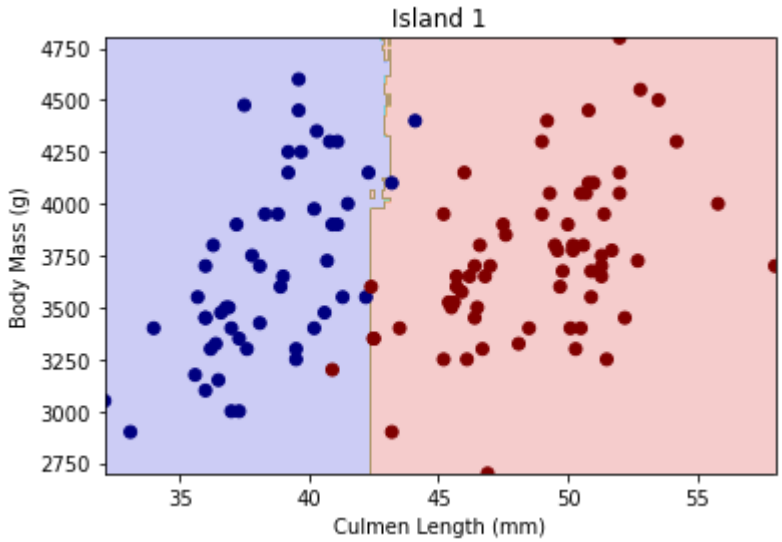
Out[27]:
```
(Species    1
 Name: 182, dtype: int32,
 Island                    1.00000
 Culmen Length (mm)       40.90000
 Culmen Depth (mm)        16.60000
 Flipper Length (mm)     187.00000
 Body Mass (g)          3200.00000
 Sex                       1.00000
 Delta 15 N (o/oo)         9.08458
 Delta 13 C (o/oo)       -24.54903
 Name: 182, dtype: float64)
```

It predicted a 0 instead of a 2 for the species. Based on our exploratory anlysis, we think this is because the island narrowed it down to species 0 or 1, but the body mass was notably lighter for species 2 and was more inline with species 0. Thus our model predicted species 0

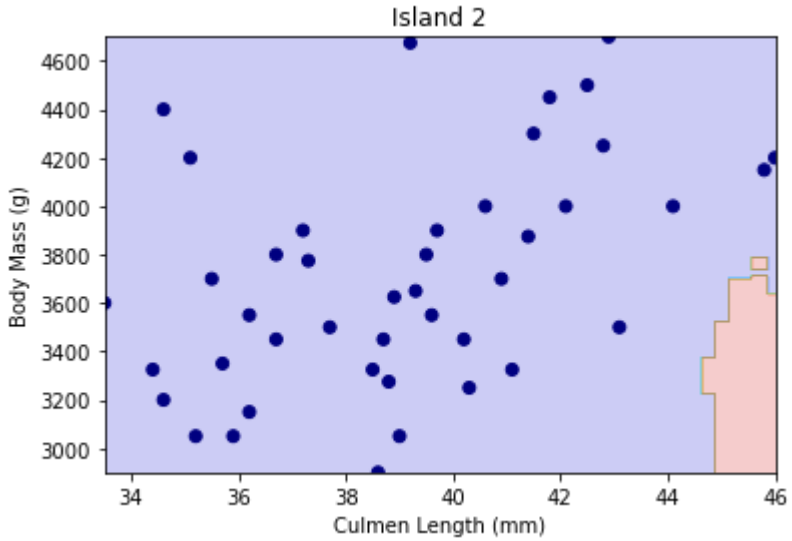
In [36]:
```
#island 0
plot_regions(model_RFC, X, y,
             Island = 0,
             title = "Island 0")
```



In [37]:
```
#island 1
plot_regions(model_RFC, X, y,
             Island = 1,
             title = "Island 1")
```



In [38]:
```
#island 2
plot_regions(model_RFC, X, y,
             Island = 2,
             title = "Island 2")
```



We can see that island 0 and 1 are exclusive to two penguin species each while island 2 only has one penguin species. We can also see that the decision regions are more fluid than the those from the decision tree which makes sense given that we used a random forest

Discussion

Reflect on your findings, as outlined in the project description.

The random forest classifier had 97.95% accuracy on the testing data while the decision tree classifier had a 97.95% accuracy and the multinomial logistic regression model had a 95.9% accuracy. Compare to each other we found that model x had the highest accuracy on the testing data. Based on these results we recommend using any of the models as they were pretty close in their accuracy when using the predictors "Island", "Culmen Length", and "Body Mass".

The data used to construct the model was very clean and had clear distinctions between species that made it easy to choose appropriate columns. In most real world datasets, which can contain hundreds of predictiors, it is not always clear which predictors to use at first glance, and as a

result, more effort must be put in exploratory analysis. To contrast, this dataset contained a small handfull of really clear data.

We believe that the accuracy of the random forest classifier and the decision tree classifier were both adequitely accurate against the testing data and believe that the variables that we chose were satisfactory predictors.