1. Group Contributions Statement

Group Members:

- Kit Larsen
- Olivia Gonzalez
- Jayson Hall

Contributions:

All three of us wrote the data acquisition and preparation. Jayson led the summary tables, Figure 2, and the KNN model. Kito led Figure , the support vector machine mode, and drawing the boundary regionsl.Olivian led Figure and the random forest modelelWe mostly wrote the descriptions of our own figures and models, with editing and suggestions from the othersion. We all checked each other's work and made revisi ns to code and writing.

NOTE

Our formatting is a little weird because none of us were able to export this document as a PDF directly through Jupyter (kept getting errors). We are instead using Microsoft Print to PDF, which cuts off some of the code blocks that are too far to the right, so we are trying right now to squish everything to the left as much as possible, which may lead to some weird spacing. The graphs and markdown cells should be fine, except for the page breaks.

2. Data Import and Cleaning

In [1]:
```
#Import Libraries
import pandas as pd
from matplotlib import pyplot as plt
from sklearn import tree, preprocessing
import numpy as np
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
import matplotlib.image as mpimg
from matplotlib.colors import ListedColormap
from itertools import product
from scipy.stats import randint
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.datasets import make_classification, load_iris
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz
```

```
from sklearn.metrics import accuracy_score, classification_report
#Sorry if duplicates, we just copy-pasted all of our
#libraries into one block when merging
```

In [2]:
```
#read penguins data
penguins = pd.read_csv("palmer_penguins.csv")
```

In [3]:
```
#set random seed
np.random.seed(3354354524)
```

In [4]:
```
#pick the columns that we can use for prediction (e.g. remove individual identifier
importantcols = ["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)"]
#group which (predictor) categories are qualitative
#and quantitative for future use
qual = ["Island", "Sex"]
quant = ["Culmen Length (mm)", "Culmen Depth (mm)",
         "Flipper Length (mm)",
                "Body Mass (g)", "Delta 15 N (o/oo)", "Delta 13 C (o/oo)"]
#subset original dataset so we have the columns we care about
#also remove the penguin with unrecorded sex (".")
#so that we don't have to search for it later
penguins = penguins.loc[:, importantcols]
penguins = penguins.loc[penguins["Sex"] != "."]
penguins
```

Out[4]:

	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)
0	Adelie Penguin (Pygoscelis adeliae)	Torgersen	39.1	18.7	181.0	3750.0	MALE	NaN	NaN
1	Adelie Penguin (Pygoscelis adeliae)	Torgersen	39.5	17.4	186.0	3800.0	FEMALE	8.94956	-24.69454
2	Adelie Penguin (Pygoscelis adeliae)	Torgersen	40.3	18.0	195.0	3250.0	FEMALE	8.36821	-25.33302
3	Adelie Penguin (Pygoscelis adeliae)	Torgersen	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Adelie Penguin (Pygoscelis adeliae)	Torgersen	36.7	19.3	193.0	3450.0	FEMALE	8.76651	-25.32426
...	...	...	...	...	...	...	...	...	...
339	Gentoo penguin (Pygoscelis papua)	Biscoe	NaN	NaN	NaN	NaN	NaN	NaN	NaN
340	Gentoo penguin (Pygoscelis papua)	Biscoe	46.8	14.3	215.0	4850.0	FEMALE	8.41151	-26.13832
341	Gentoo penguin (Pygoscelis papua)	Biscoe	50.4	15.7	222.0	5750.0	MALE	8.30166	-26.04117
342	Gentoo penguin (Pygoscelis papua)	Biscoe	45.2	14.8	212.0	5200.0	FEMALE	8.24246	-26.11969
343	Gentoo penguin (Pygoscelis papua)	Biscoe	49.9	16.1	213.0	5400.0	MALE	8.36390	-26.15531

343 rows × 9 columns

In [5]:
```
#split into target variable (y, species) and predictor variables (X, everything els
X = penguins[["Island", "Culmen Length (mm)", "Culmen Depth (mm)",
              "Flipper Length (mm)",
                "Body Mass (g)", "Sex",
              "Delta 15 N (o/oo)", "Delta 13 C (o/oo)"]]
y = penguins["Species"]
```

In [6]:
```
#split for training, 80% training data, 20% test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=69)
```

In [7]:
```
#function that cleans data (drops rows with NaN)
'''
def clean_data(df):
    new_df = df.copy()
    new_df = new_df.dropna()
    return new_df
'''
clean_data = lambda df: df.dropna()
```

In [8]:
```
#rejoin and clean the training & testing data
#remake the X and y parts
traindata = pd.concat((X_train, y_train), axis = 1)
testdata = pd.concat((X_test, y_test), axis = 1)
traindata = clean_data(traindata)
testdata = clean_data(testdata)
X_train = traindata.drop(["Species"], axis = 1)
y_train = traindata["Species"]
X_test = testdata.drop(["Species"], axis = 1)
y_test = testdata["Species"]
```

3. Exploratory Analysis

In [9]:
```
# show the means of the various quantitative predictors by species
sum_table = traindata.groupby(["Species"])[quant].mean()
sum_table.round(2)
#note that the only quantitative statistic that can tell
#adelie and chinstrap penguins apart is culmen length
#and possibly Delta 15 N, so one of those two statistics must be used
```

Out[9]:

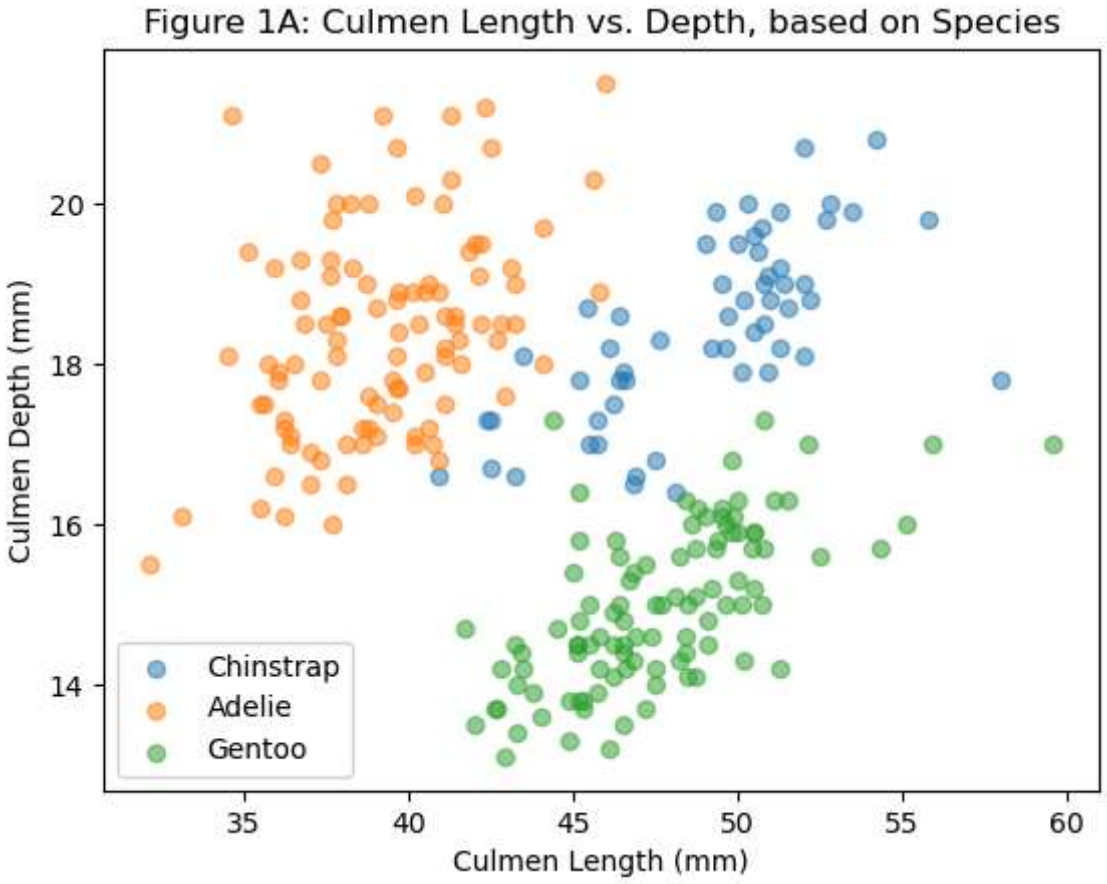
Species	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
Adelie Penguin (Pygoscelis adeliae)	39.23	18.39	191.10	3732.18	8.90	-25.83
Chinstrap penguin (Pygoscelis antarctica)	48.99	18.44	195.84	3740.62	9.38	-24.56
Gentoo penguin (Pygoscelis papua)	47.59	14.99	217.14	5081.86	8.25	-26.22

Discussion for Summary Table 1

The above summary table shows the overall means of the quantitative predictor columns by species. One thing of note is that the only quantitative statistics that can tell Adelie and Chinstrap penguins apart well (according to the numbers) are Culmen Length and Delta 15 N. This summary table helped us narrow down which categories to graph for a better visual perspective.

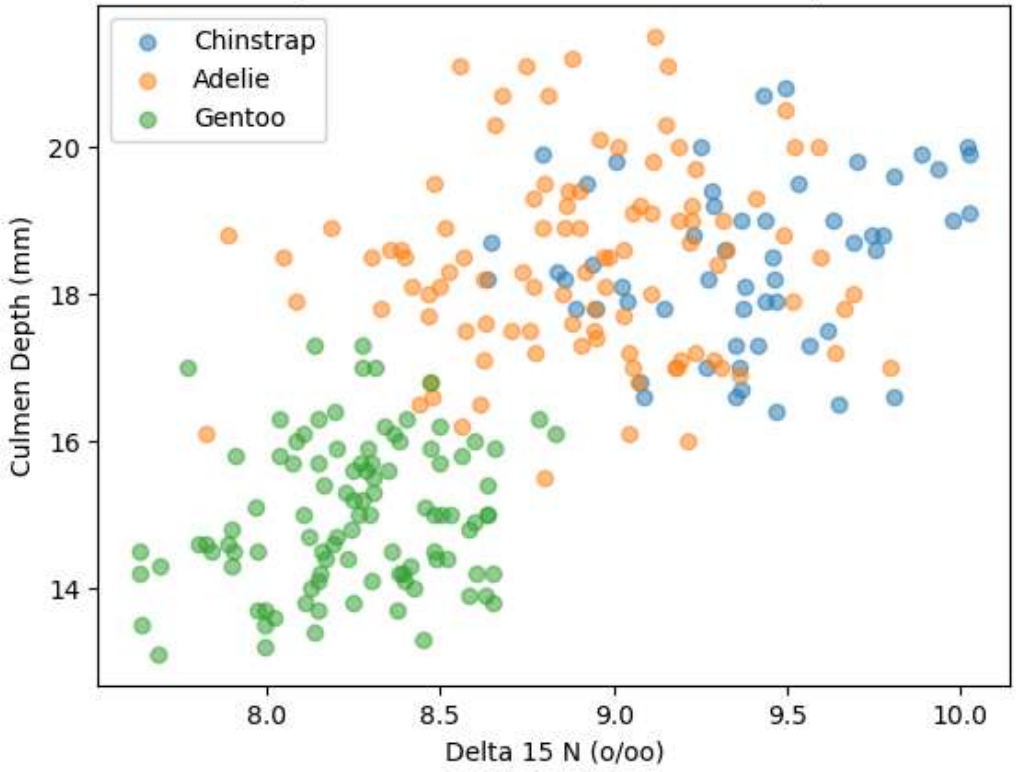
Note also that the Culmen Depth, Flipper Length, and Body Mass seem to be very effective at distinguishing Gentoo penguins from the other penguins, so that should be investigated as well.

In [10]:
```
#plot culmen length vs culmen depth by species, little overlap
fig, ax = plt.subplots(1)
ax.set(xlabel = "Culmen Length (mm)",
       ylabel = "Culmen Depth (mm)")

penguin_species = set(traindata["Species"])
for p in penguin_species:
    temp = traindata.loc[(traindata["Species"] == p)]
    if(len(temp) == 0):
        continue
    ax.scatter(temp["Culmen Length (mm)"], temp["Culmen Depth (mm)"],
               label = p.split()[0], alpha = 0.5)

ax.set(title = "Figure 1A: Culmen Length vs. Depth, based on Species")
ax.legend()
print()
```



In [11]:
```
#plot nitrogen concentration vs culmen depth by species to see if
#nitrogen concentration is superior to culmen length
fig, ax = plt.subplots(1)
ax.set(xlabel = "Delta 15 N (o/oo)",
       ylabel = "Culmen Depth (mm)")

penguin_species = set(traindata["Species"])
for p in penguin_species:
    temp = traindata.loc[(traindata["Species"] == p)]
    if(len(temp) == 0):
        continue
    ax.scatter(temp["Delta 15 N (o/oo)"], temp["Culmen Depth (mm)"],
               label = p.split()[0], alpha = 0.5)

ax.set(title = "Figure 1B: Blood Nitrogen Concentration vs. Culmen Depth, based on
ax.legend()
print()
#less clear for nitrogen concentration
```

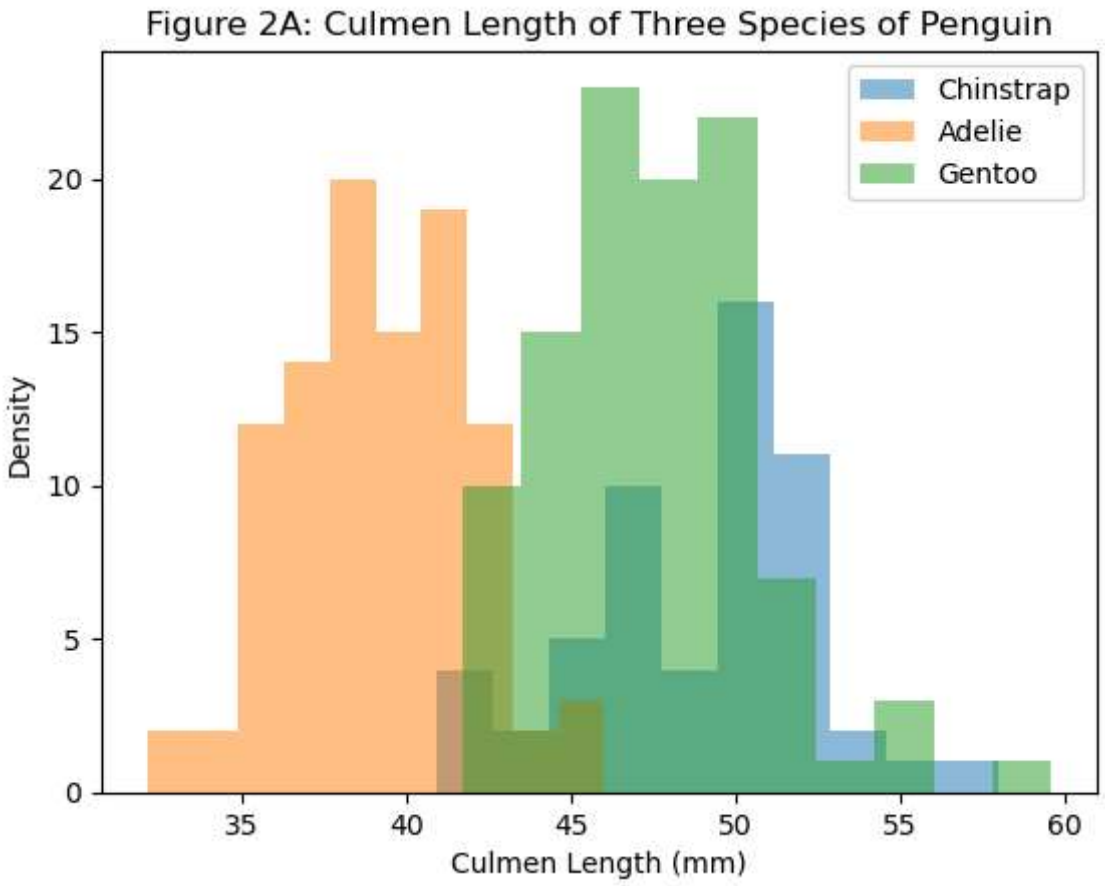
Figure 1B: Blood Nitrogen Concentration vs. Culmen Depth, based on Species



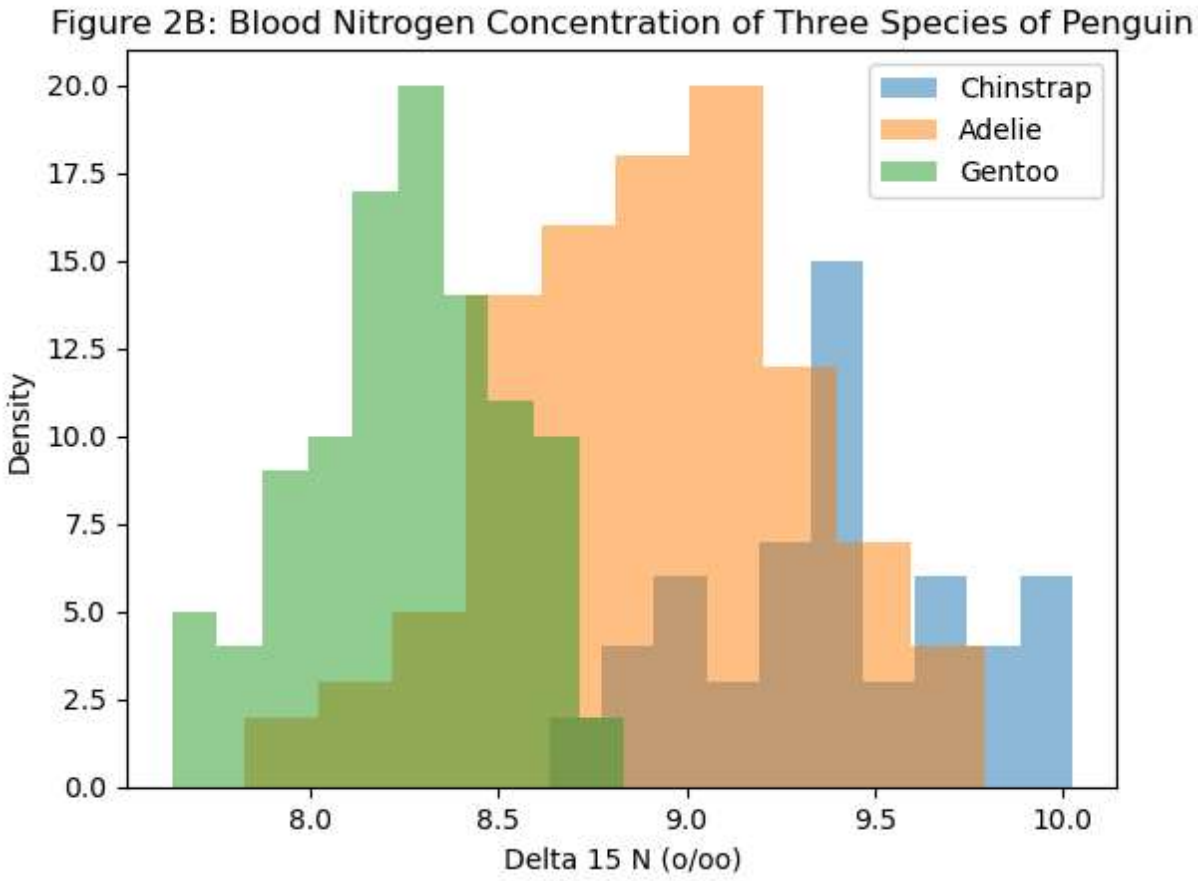
Discussion For Figure 1A and 1B

According to the scatterplots above (which use culmen depth arbitrarily as a standard), It appears that the blood nitrogen concentration of the Adelie penguins are more spread out, whereas the culmen length is more concentrated. Reduced variation would decrease misclassification, so culmen length is likely better for distinguishing between the Adelie and the Chinstrap.

In [12]:
```
#histogram to show overlap between Culmen Length of the species
fig, ax = plt.subplots(1)
ax.set(xlabel = "Culmen Length (mm)",
       ylabel = "Density")

penguin_species = set(traindata["Species"])

for p in penguin_species:
    temp = traindata.loc[traindata["Species"] == p]
    ax.hist(temp["Culmen Length (mm)"],
            label = p.split()[0], alpha = 0.5)

ax.set(title = "Figure 2A: Culmen Length of Three Species of Penguin")
ax.legend()
print()
```



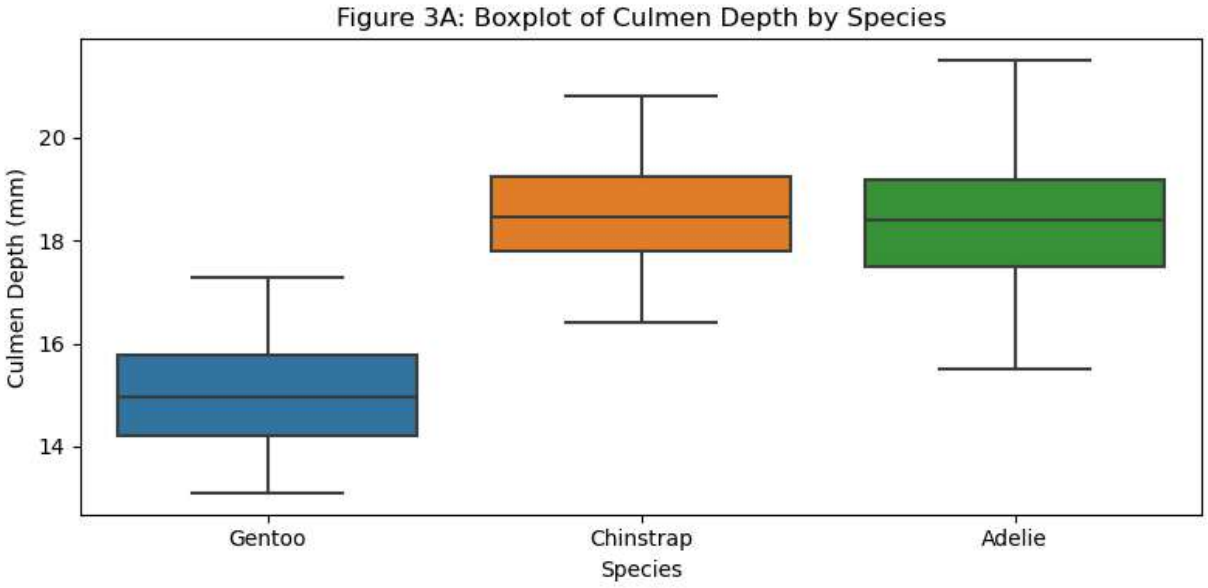
In [13]:
```
#histogram to show overlap between
#blood nitrogen concentration of the species
fig, ax = plt.subplots(1)
ax.set(xlabel = "Delta 15 N (o/oo)",
       ylabel = "Density")

penguin_species = set(traindata["Species"])

for p in penguin_species:
    temp = traindata.loc[traindata["Species"] == p]
    ax.hist(temp["Delta 15 N (o/oo)"], label = p.split()[0], alpha = 0.5)

ax.set(title = "Figure 2B: Blood Nitrogen Concentration of Three Species of Penguin
ax.legend()
print()
```



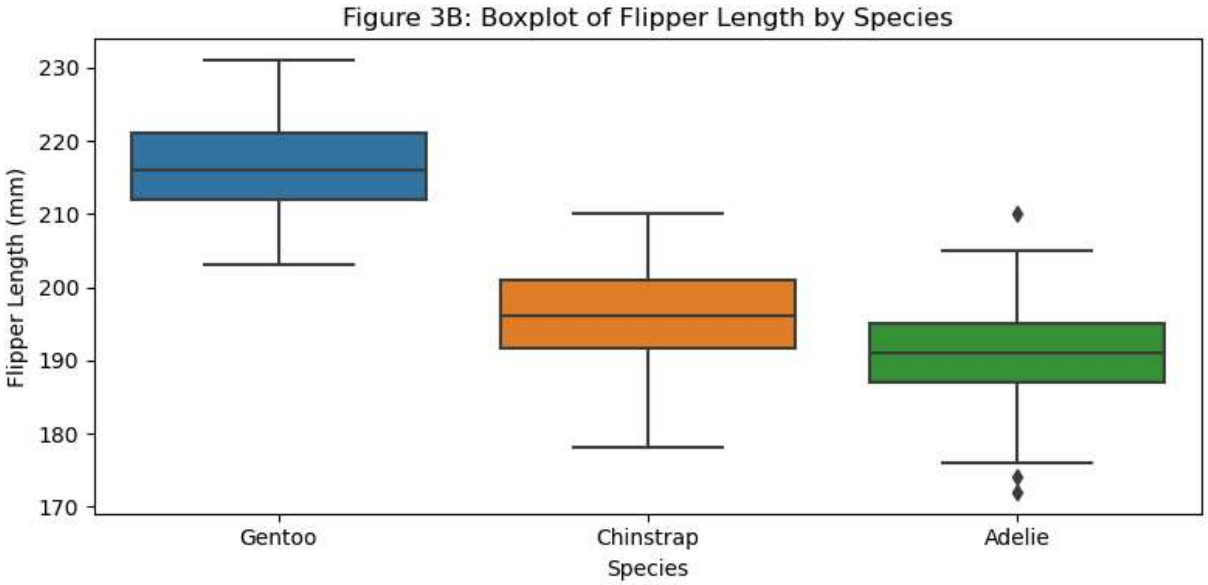
Discussion For Figure 2A, 2B

Likewise, according to the histograms, there is reduced overlap between Adelie and Chinstrap in the "Culmen Length" histogram compared to the "Blood Nitrogen Concentration" histogram. Therefore, Culmen Length would be a superior quantitative metric for classifying Adelie and Chinstrap.

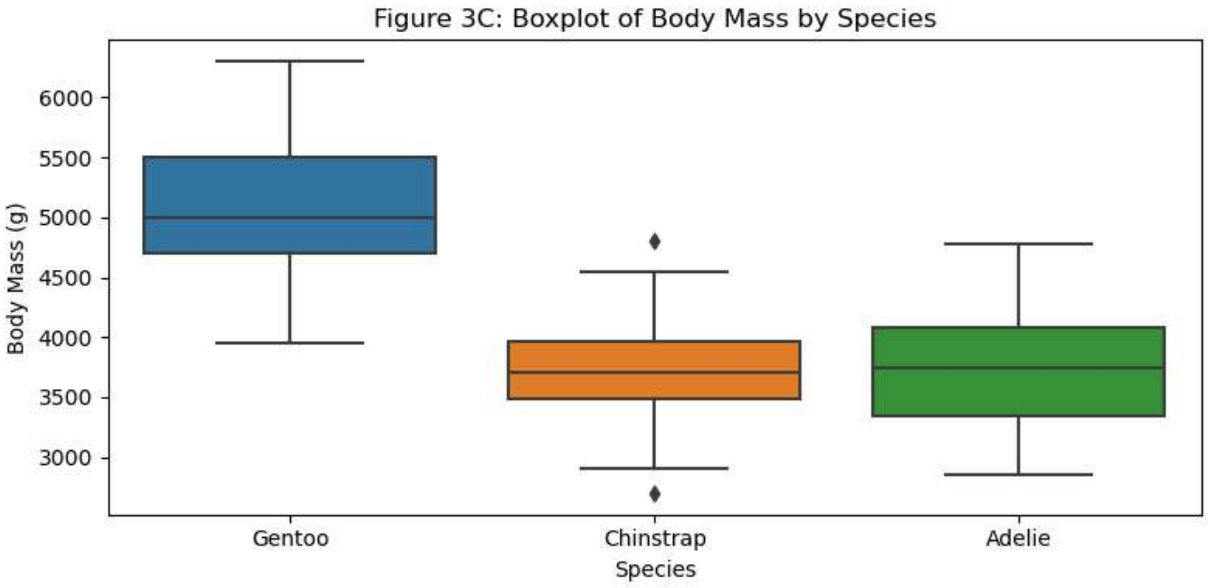
In [14]:
```
# making a boxplot
data_for_graph = traindata.copy()
#just shortening the name so the graph looks cleaner
data_for_graph["Species"] = data_for_graph["Species"].str.split().str.get(0)
#sns.set(style="whitegrid")
plt.figure(figsize=(8, 4))
sns.boxplot(x='Species', y='Culmen Depth (mm)', data=data_for_graph)
plt.title('Figure 3A: Boxplot of Culmen Depth by Species')
plt.xlabel('Species')
plt.ylabel('Culmen Depth (mm)')
plt.tight_layout()
plt.show()
```



In [15]:
```
#sns.set(style="whitegrid")
plt.figure(figsize=(8, 4))
sns.boxplot(x='Species', y='Flipper Length (mm)', data=data_for_graph)
plt.title('Figure 3B: Boxplot of Flipper Length by Species')
plt.xlabel('Species')
plt.ylabel('Flipper Length (mm)')
plt.tight_layout()
plt.show()
```



In [16]:
```
#sns.set(style="whitegrid")
plt.figure(figsize=(8, 4))
sns.boxplot(x='Species', y='Body Mass (g)', data=data_for_graph)
plt.title('Figure 3C: Boxplot of Body Mass by Species')
plt.xlabel('Species')
plt.ylabel('Body Mass (g)')
plt.tight_layout()
plt.show()
```



By looking at these boxplots, one can see that the Gentoo penguin is consistently rated as having the smallest culmen depth, longest flippers, and largest body mass. This is confirmed with the median being significantly lower than the two other species of penguins. The boxplots do show that there would be some overlap between the extremities of the Gentoo and of the other penguins, which means that there is some potential for misclassification. Body mass appears to have the most overlap, followed by Culmen Depth and Flipper Length, which appear similar.

In [17]:
```
#now further categorize with sex and island
sum_table = traindata.groupby(["Species", "Sex",
                               "Island"])[quant].mean()
sum_table.round(2)
#note that chinstrap and gentoo penguin can be
#distinguished by which island they live on
# but sex does not seem to help much
```

Out[17]:

Species	Sex	Island	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
Adelie Penguin (Pygoscelis adeliae)	FEMALE	Biscoe	37.92	17.77	187.38	3346.88	8.76	-25.93
		Dream	36.81	17.46	188.67	3286.11	9.00	-25.65
		Torgersen	37.92	17.49	188.62	3409.62	8.76	-25.79
	MALE	Biscoe	40.87	19.15	191.76	4123.53	8.93	-26.04
		Dream	40.29	18.92	192.26	4042.11	8.98	-25.71
		Torgersen	41.12	19.25	196.78	4056.94	8.94	-25.88
Chinstrap penguin (Pygoscelis antarctica)	FEMALE	Dream	46.43	17.49	191.27	3499.04	9.26	-24.56
	MALE	Dream	51.20	19.26	199.80	3950.00	9.50	-24.57
Gentoo penguin (Pygoscelis papua)	FEMALE	Biscoe	45.74	14.25	212.65	4684.13	8.19	-26.22
	MALE	Biscoe	49.52	15.76	221.80	5495.50	8.30	-26.22

Discussion for Summary Table 2

Between the two qualitative features, Island and Sex, Island seems more useful because Chinstrap penguins live exclusively on Dream Island and Gentoo Penguins live exclusively on Biscoe Island. Although there is differences between male and female penguins, there does not appear to be enough to overcome the utility of Island in classification.

4. Feature Selection

In [18]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score

def findOptimalKNN(df, qual, quant, target):
    '''
    Purpose: Find the optimal combination of one (1) qualitative
    and two (2) quantitative predictor variables
    for a K-nearest neighbors model, including
    the optimal number of neighbors (between 1 and 15).

    Arguments:
    df: the pandas dataframe in question used to build the model (training data)
    including both predictors and target
    note that the qualitative columns should have been transformed already into int

    qual: a list of the qualitative predictor columns
```

```
    (should already be in 'int' format, just used to keep track of combos)
    quant: a list of the quantitative predictor columns
    target: the name of the target column as a string

    Returns: A list containing the highest cross-value score
    found along with its corresponding predictor variables
    as a list and the corresponding number of neighbors.
    '''
    temp = df.copy() #make a copy of the data
    y = temp[target] # create a y column (of answers)
    maxScore = 0 # declare variables to store the best answers
    maxCats = []
    maxNeighbors = 0
    for qual1 in qual: # loop through qualitative columns
        copyQuant = quant.copy() # make a new copy for each
        #new qualitative column so that we can go through
        # each of the quantitative combinations again
        for quant1 in quant: # pick quantitative #1
            if(quant1 in copyQuant):
                copyQuant.remove(quant1) # remove this column from
                #this temporary list so we do not use it again
            for quant2 in copyQuant: # exhaust each possible pairing
                #with another quantitative column
                print("Testing:", qual1, quant1, quant2)
                # show which ones we are testing
                X = temp[[qual1, quant1, quant2]]
                # subset X to just the columns of interest
                best_score = 0
                # store best score of the parameter nearest neighbor
                #(estimators for forest)
                best_neighbors = 0
                for i in range(1, 15): #arbitrarily chose 15 so
                    #it does not take forever to run
                    knn = KNeighborsClassifier(n_neighbors = i) # make model
                    knn.fit(X, y) # fit model to modified data
                    raw_scores = cross_val_score(knn, X, y, cv=5)
                    # calculate cv scores
                    score = raw_scores.mean() # take the mean
                    if score > best_score: # if the average of the cv scores
                        #is the highest for the grouping of columns
                        best_score = score # save it
                        best_neighbors = i
                if best_score > maxScore:
                    # if this is the best score out of all the groupings of columns
                    maxScore = best_score
                    # save it, and the categories, as well as the # of neighbors
                    maxCats = [qual1, quant1, quant2]
                    maxNeighbors = best_neighbors
    return [maxScore, maxCats, maxNeighbors]
# return all the information of the best score
```

SAME CODE AS ABOVE, JUST IN MARKDOWN BECAUSE AFOREMENTIONED PDF PROBLEMS

from sklearn.neighbors import KNeighborsClassifier from sklearn.model_selection import cross_val_score

def findOptimalKNN(df, qual, quant, target): ''' Purpose: Find the optimal combination of one (1) qualitative and two (2) quantitative predictor variables for a K-nearest neighbors model, including the optimal number of neighbors (between 1 and 15).

```
Arguments:
df: the pandas dataframe in question used to build the model
(training data), including both predictors and target
    note that the qualitative columns should have been transformed
already into integers
qual: a list of the qualitative predictor columns (should be in
'int' format, just used to keep track of combos)
quant: a list of the quantitative predictor columns
target: the name of the target column as a string

Returns: A list containing the highest cross-value score found
along with its corresponding predictor variables
as a list and the corresponding number of neighbors.
'''
temp = df.copy() #make a copy of the data
y = temp[target] # create a y column (of answers)
maxScore = 0 # declare variables to store the best answers
maxCats = []
maxNeighbors = 0
for qual1 in qual: # loop through qualitative columns
    copyQuant = quant.copy() # make a new copy for each new
qualitative column so that we can go through
    # each of the quantitative combinations again
    for quant1 in quant: # pick quantitative #1
        if(quant1 in copyQuant):
            copyQuant.remove(quant1) # remove this column from this
temporary list so we do not use it again
        for quant2 in copyQuant: # exhaust each possible pairing
with another quantitative column
            print("Testing:", qual1, quant1, quant2) # show which
ones we are testing
            X = temp[[qual1, quant1, quant2]] # subset X to just
the columns of interest
            best_score = 0 # store best score of the parameter
nearest neighbor (estimators for forest)
            best_neighbors = 0
            for i in range(1, 15): #arbitrarily chose 15 so it does
not take forever to run
                knn = KNeighborsClassifier(n_neighbors = i) # make
model
                knn.fit(X, y) # fit model to modified data
                raw_scores = cross_val_score(knn, X, y, cv=5) #
calculate cv scores
                score = raw_scores.mean() # take the mean
```

```
                if score > best_score: # if the average of the cv
    scores is the highest for the grouping of columns
                    best_score = score # save it
                    best_neighbors = i
            if best_score > maxScore: # if this is the best score
    out of all the groupings of columns
                maxScore = best_score # save it, and the
    categories, as well as the # of neighbors
                maxCats = [qual1, quant1, quant2]
                maxNeighbors = best_neighbors
    return [maxScore, maxCats, maxNeighbors] # return all the
    information of the best score
```

In [19]:

```
def findOptimalForest(df, qual, quant, target):
    '''
    Same as above code, but with number of estimators
    rather than number of neighbors as the parameter checked.
    '''
    temp = df.copy()
    y = temp[target]
    maxScore = 0
    maxCats = []
    maxDepth = 0
    for qual1 in qual:
        copyQuant = quant.copy()
        for quant1 in quant:
            if(quant1 in copyQuant):
                copyQuant.remove(quant1)
            for quant2 in copyQuant:
                X = temp[[qual1, quant1, quant2]]
                best_score = 0
                best_neighbors = 0
                for i in range(1, 15):
                    m = RandomForestClassifier(n_estimators = i)
                    m.fit(X, y)
                    raw_scores = cross_val_score(m, X, y, cv=5)
                    score = raw_scores.mean()
                    if score > best_score:
                        best_score = score
                        best_depth = i
                if best_score > maxScore:
                    maxScore = best_score
                    maxCats = [qual1, quant1, quant2]
                    maxDepth = best_depth
    return [maxScore, maxCats, maxDepth]
```

In [20]:

```
from sklearn import svm
def findOptimalSVM(df, qual, quant, target):
    '''
    Same as above code, except without a number of neighbors / estimators parameter
    of this function returns one less value in the list, and also runs much quicker
    '''
    temp = df.copy()
    y = temp[target]
```

```
    maxScore = 0
    maxCats = []
    for qual1 in qual:
        copyQuant = quant.copy()
        for quant1 in quant:
            if(quant1 in copyQuant):
                copyQuant.remove(quant1)
            for quant2 in copyQuant:
                X = temp[[qual1, quant1, quant2]]
                best_score = 0
                best_neighbors = 0
                svm_model = svm.SVC()
                svm_model.fit(X, y)
                raw_scores = cross_val_score(svm_model, X, y, cv=5)
                score = raw_scores.mean()
                if score > best_score:
                     best_score = score
                if best_score > maxScore:
                    maxScore = best_score
                    maxCats = [qual1, quant1, quant2]
    return [maxScore, maxCats]
```

In [21]:

```
transformed_traindata = traindata.copy()
le_island = LabelEncoder()
transformed_traindata['Island'] = le_island.fit_transform(transformed_traindata['Is

le_sex = LabelEncoder()
transformed_traindata['Sex'] = le_sex.fit_transform(transformed_traindata['Island']

le_species = LabelEncoder()
transformed_traindata['Species'] = le_species.fit_transform(transformed_traindata['

optimalKNN = findOptimalKNN(transformed_traindata, qual, quant, "Species")
print(optimalKNN)
```

```
Testing: Island Culmen Length (mm) Culmen Depth (mm)
Testing: Island Culmen Length (mm) Flipper Length (mm)
Testing: Island Culmen Length (mm) Body Mass (g)
Testing: Island Culmen Length (mm) Delta 15 N (o/oo)
Testing: Island Culmen Length (mm) Delta 13 C (o/oo)
Testing: Island Culmen Depth (mm) Flipper Length (mm)
Testing: Island Culmen Depth (mm) Body Mass (g)
Testing: Island Culmen Depth (mm) Delta 15 N (o/oo)
Testing: Island Culmen Depth (mm) Delta 13 C (o/oo)
Testing: Island Flipper Length (mm) Body Mass (g)
Testing: Island Flipper Length (mm) Delta 15 N (o/oo)
Testing: Island Flipper Length (mm) Delta 13 C (o/oo)
Testing: Island Body Mass (g) Delta 15 N (o/oo)
Testing: Island Body Mass (g) Delta 13 C (o/oo)
Testing: Island Delta 15 N (o/oo) Delta 13 C (o/oo)
Testing: Sex Culmen Length (mm) Culmen Depth (mm)
Testing: Sex Culmen Length (mm) Flipper Length (mm)
Testing: Sex Culmen Length (mm) Body Mass (g)
Testing: Sex Culmen Length (mm) Delta 15 N (o/oo)
Testing: Sex Culmen Length (mm) Delta 13 C (o/oo)
Testing: Sex Culmen Depth (mm) Flipper Length (mm)
Testing: Sex Culmen Depth (mm) Body Mass (g)
Testing: Sex Culmen Depth (mm) Delta 15 N (o/oo)
Testing: Sex Culmen Depth (mm) Delta 13 C (o/oo)
Testing: Sex Flipper Length (mm) Body Mass (g)
Testing: Sex Flipper Length (mm) Delta 15 N (o/oo)
Testing: Sex Flipper Length (mm) Delta 13 C (o/oo)
Testing: Sex Body Mass (g) Delta 15 N (o/oo)
Testing: Sex Body Mass (g) Delta 13 C (o/oo)
Testing: Sex Delta 15 N (o/oo) Delta 13 C (o/oo)
[0.976923076923077, ['Island', 'Culmen Length (mm)', 'Culmen Depth (mm)'], 2]
```

In [22]:
```
optimalForest = findOptimalForest(transformed_traindata, qual, quant, "Species")
print(optimalForest)
```

```
[0.9884615384615385, ['Sex', 'Culmen Length (mm)', 'Culmen Depth (mm)'], 13]
```

In [23]:
```
optimalSVM = findOptimalSVM(transformed_traindata, qual, quant, "Species")
print(optimalSVM)
```

```
[0.8571644042232279, ['Island', 'Culmen Length (mm)', 'Culmen Depth (mm)']]
```

Discussion of Feature Selection

We chose Island for the qualitative feature, and Culmen Length & Culmen Depth for the quantitative feature. This is because, after checking all possible combinations of one qualitative and two quantitative features, Island, Culmen Length, and Culmen Depth had the best cross-validation score for each of the three models.

Similar analysis allowed us to include that the best number of neighbors for a KNN model was 2, and the best number of estimators for the random forest model was 9.

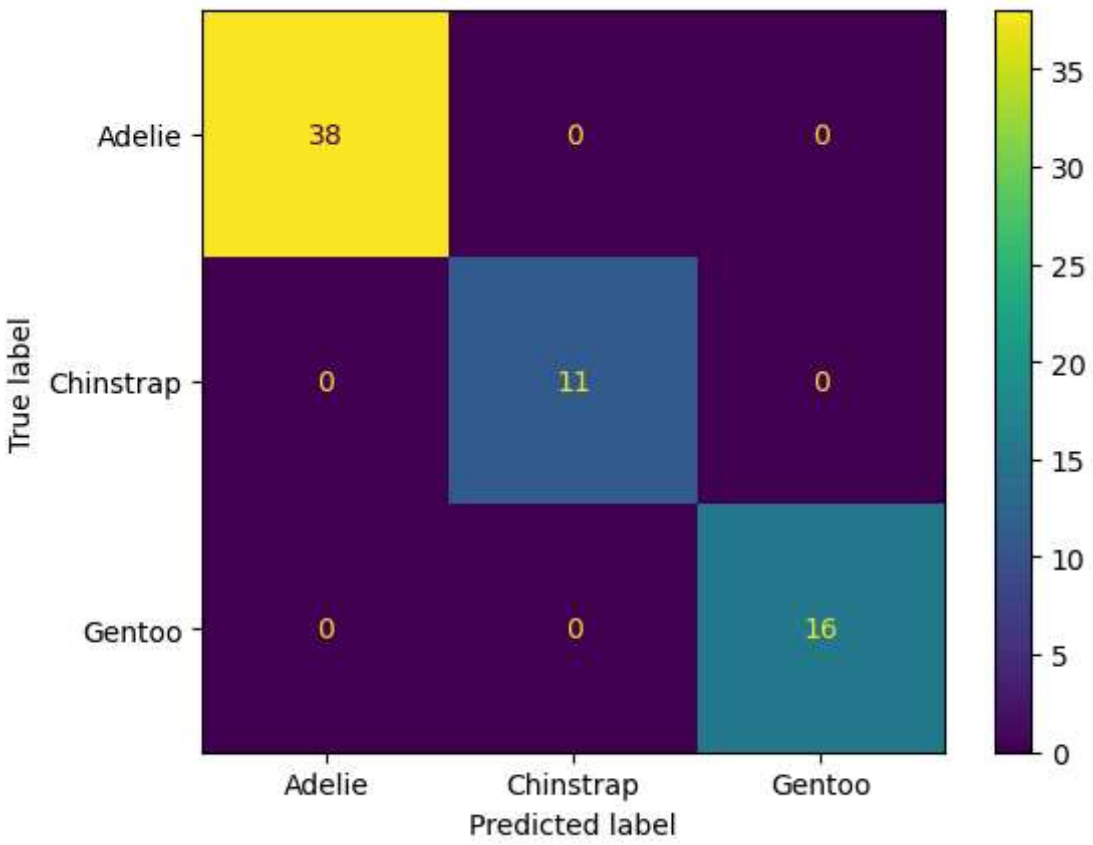
5. Modeling

KNN Model

In [24]:
```
#Using the test data to assess the KNN model
#trained on the parameters given by findOptimalKNN().
print("KNN Model")
print("---------")
knn = KNeighborsClassifier(n_neighbors = optimalKNN[2])
myX = transformed_traindata[optimalKNN[1]]
myY = transformed_traindata["Species"]

transformed_testdata = testdata.copy()
transformed_testdata['Island'] = le_island.fit_transform(transformed_testdata['Isla
transformed_testdata['Sex'] = le_sex.fit_transform(transformed_testdata['Island'])
transformed_testdata['Species'] = le_species.fit_transform(transformed_testdata['Sp

knn.fit(myX, myY)
myX_test = transformed_testdata[optimalKNN[1]]
myY_test = transformed_testdata["Species"]
predicted_labels_test = knn.predict(myX_test)
print("CV Score", optimalKNN[0])

from sklearn.metrics import accuracy_score
print("Accuracy Score:", accuracy_score(myY_test, predicted_labels_test))
labels = ["Adelie", "Chinstrap", "Gentoo"]
from sklearn.metrics import ConfusionMatrixDisplay
predicted_labels_test = knn.predict(myX_test)
ConfusionMatrixDisplay.from_predictions(myY_test,
                    predicted_labels_test, display_labels=labels)
```

```
KNN Model
---------
CV Score 0.976923076923077
Accuracy Score: 1.0
```

Out[24]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15ce635a090>
```

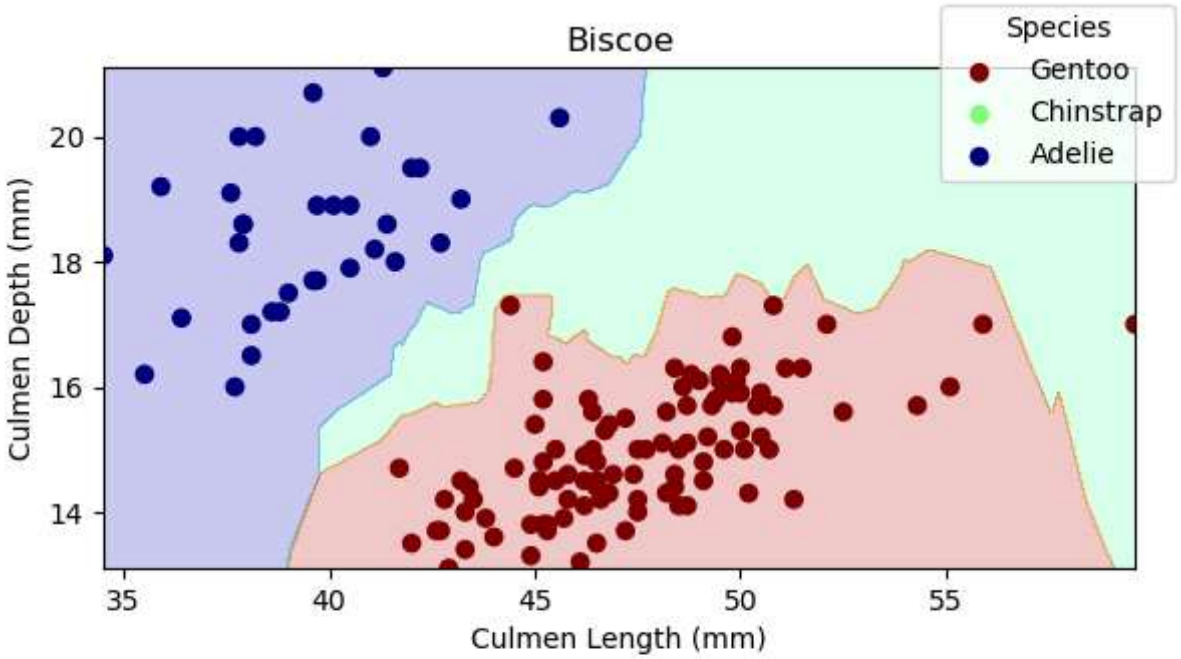


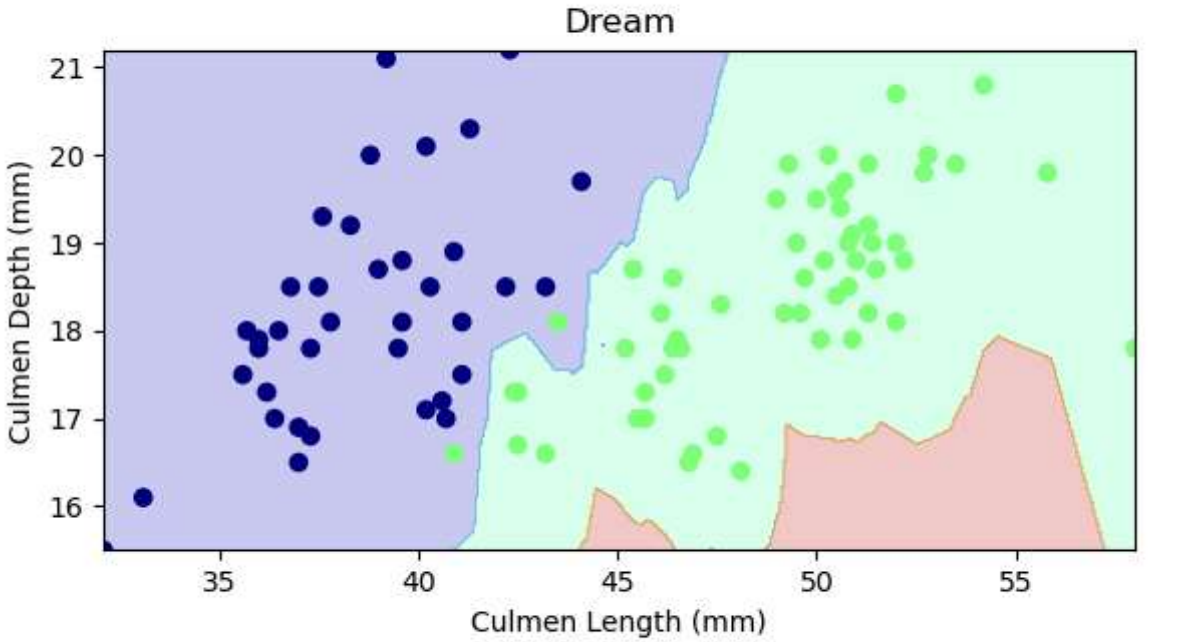
In [25]:
```python
#Plotting decision boundaries, taken from lecture
def plot_regions(c, df, col1, col2, qualcol, qual_classes):
    """
a simple function to plot decision regions for a classification model c
if you use this function in your projects, you will need to modify how
the prediction works in some important ways, AND you will need to ensure
that the plots produced have a legend describing how the colors correspond
to species. Additionally, you will need to include a better docstring than
this one. =)

My DOCSTRING

Purpose: Plot decision regions for a classification model (c)
on an encoded dataset (df) using two quantitative
columns (col1, col2) and one qualitative column (qualcol).

Arguments:
c: trained model
df: encoded pandas data frame
col1: quantitative column name 1 as a string
col2: quantitative column name 2 as a string
qualcol: qualitative column name as a string
qual_classes: the .classes_ attribute of the label encoder
used for the qualitative column

Returns:
none, but prints a separate graph for each aspect of the qualitative column
    """
    # for convenience, give names to the two
```

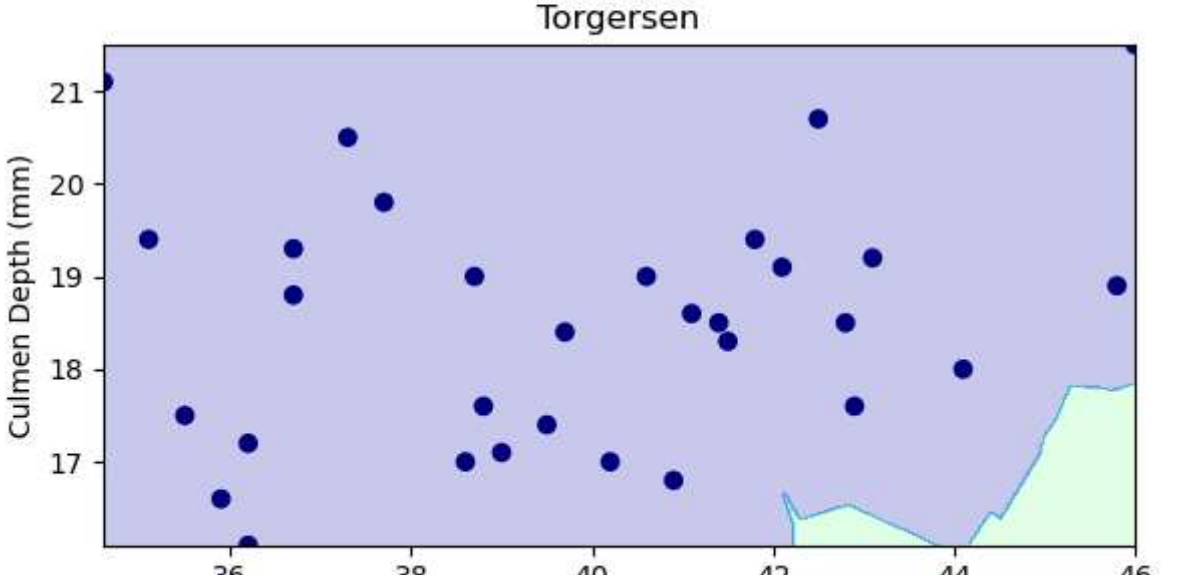
```
    # columns of the data
    fig, ax = plt.subplots(3, 1, figsize = (6, 10), label = "Inline Label")
    species_colors = ['red', 'green', 'blue']
    for idx, qual in enumerate(qual_classes): #note: uses encoded data (str --> int
        temp = df.loc[df[qualcol] == idx]
        tempX = temp.drop(["Species"], axis = 1)
        tempY = temp["Species"]
        x0 = tempX[col1]
        x1 = tempX[col2]
        # 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 = pd.DataFrame({
            qualcol : idx,
        col1 : XX,
        col2 : YY,
        })
        p = c.predict(XY)
        p = p.reshape(xx.shape)
        # use contour plot to visualize the predictions
        ax[idx].contourf(xx, yy, p, cmap = "jet", alpha = 0.2, vmin = 0, vmax = 2)
        # plot the data
        ax[idx].scatter(x0, x1, c = tempY, cmap = "jet", vmin = 0, vmax = 2)
        ax[idx].set(xlabel = col1,
        ylabel = col2, title = qual)
    fig.legend(["Gentoo", "Chinstrap", "Adelie"], title="Species")
    fig.tight_layout()
```

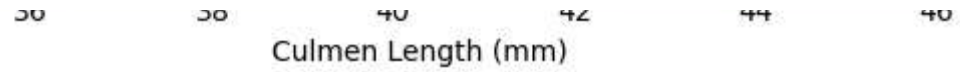
In [26]:

```
#plotting decision boundary region for KNN
plot_regions(knn, transformed_traindata, "Culmen Length (mm)",
             "Culmen Depth (mm)", "Island", le_island.classes_)
```


Biscoe
Species
Gentoo
Chinstrap
Adelie
Culmen Depth (mm)
Culmen Length (mm)
20
18
16
14
35
40
45
50
55


Dream
Culmen Depth (mm)
Culmen Length (mm)
21
20
19
18
17
16
35
40
45
50
55


Torgersen
Culmen Depth (mm)
21
20
19
18
17



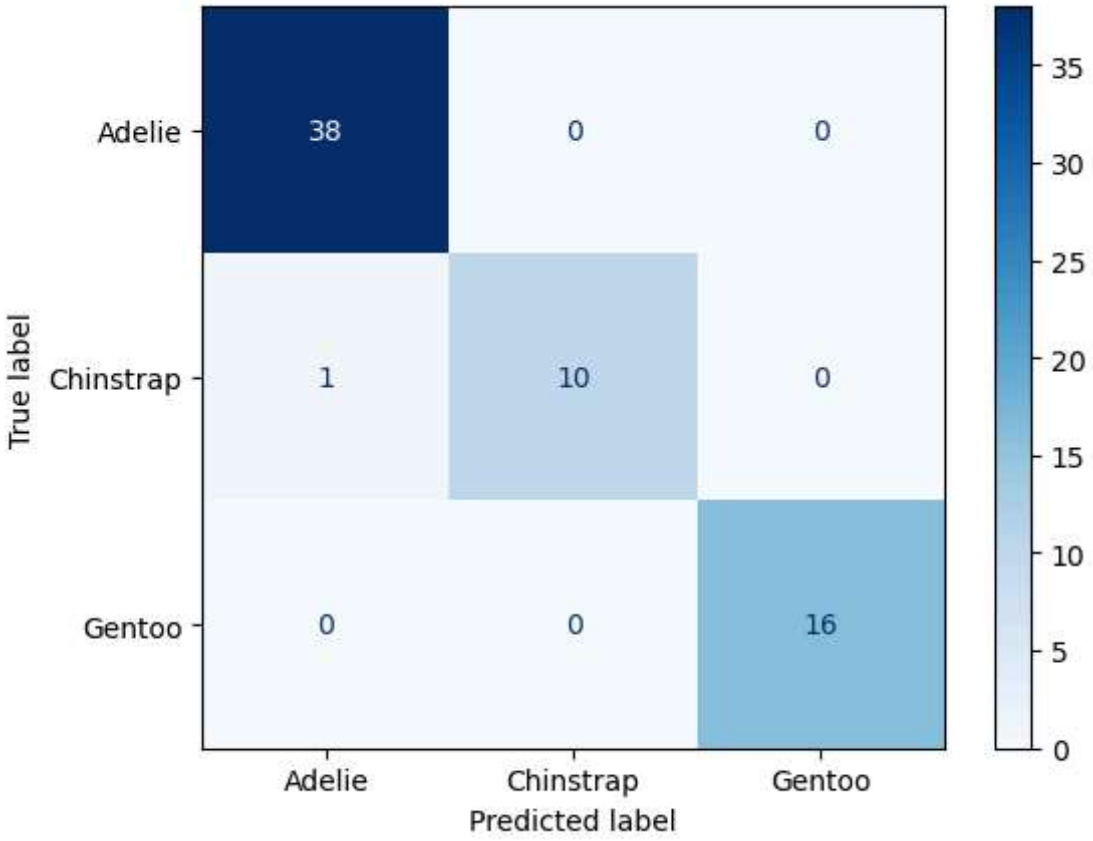
Support Vector Machine

In [27]:
```
# Create a pipeline with scaling and SVM (support vector machines)
svm_model = make_pipeline(StandardScaler(), SVC(kernel='linear'))

X_train = transformed_traindata[optimalSVM[1]]
y_train = transformed_traindata["Species"]
X_test = transformed_testdata[optimalSVM[1]]
y_test = transformed_testdata["Species"]

# Perform cross-validation on the training set
cv_scores = cross_val_score(svm_model, X_train, y_train, cv=5)

print("Cross-validation scores:", cv_scores)
# prints scores for each of the 5 training (5 folds)
print("Mean cross-validation score:", cv_scores.mean())
# mean of all training scores

# Train the model on the entire training set
svm_model.fit(X_train, y_train)

# Evaluate the model on the test set
test_score = svm_model.score(X_test, y_test)

print("Test set score:", test_score) # score of testing set
```

```
Cross-validation scores: [0.94230769 0.98076923 0.98076923 1.         0.98039216]
Mean cross-validation score: 0.9768476621417796
Test set score: 0.9846153846153847
```

In [28]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Make predictions on the test set
y_pred = svm_model.predict(X_test)

# Making the confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", cm)

labels = ["Adelie", "Chinstrap", "Gentoo"]

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```

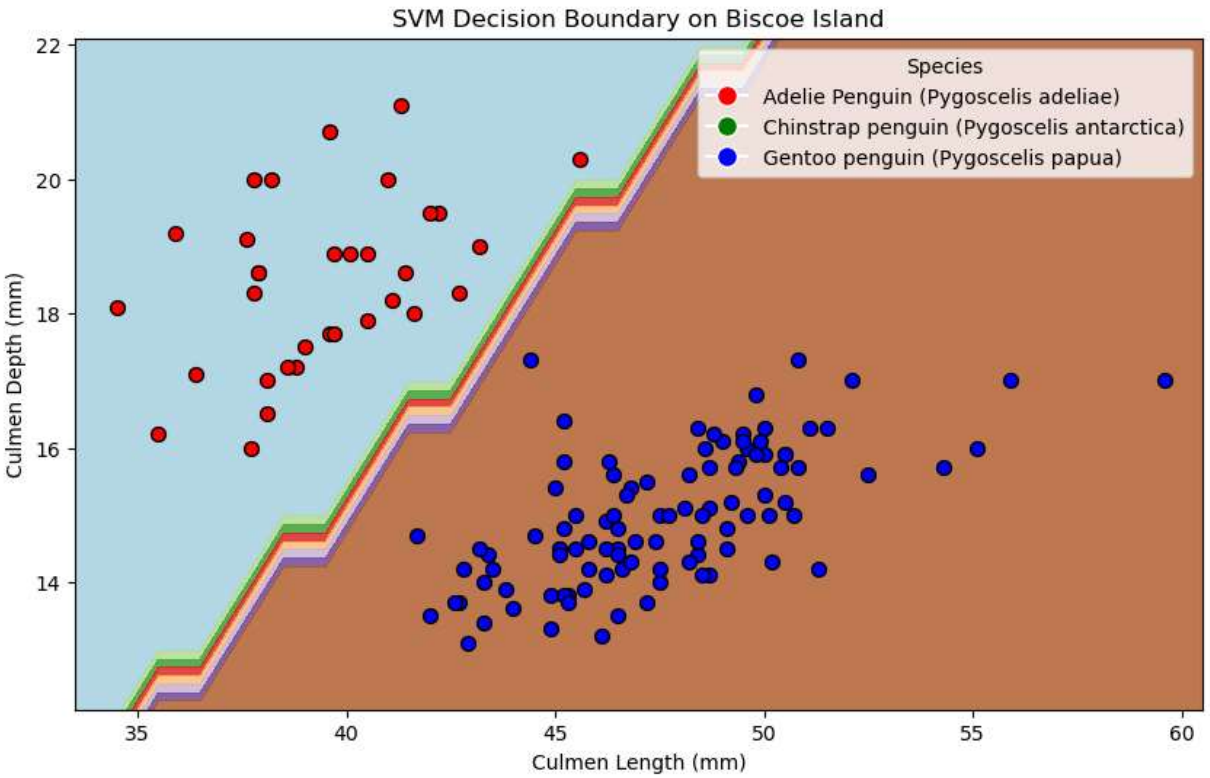
```
Confusion matrix:
 [[38  0  0]
 [ 1 10  0]
 [ 0  0 16]]
```

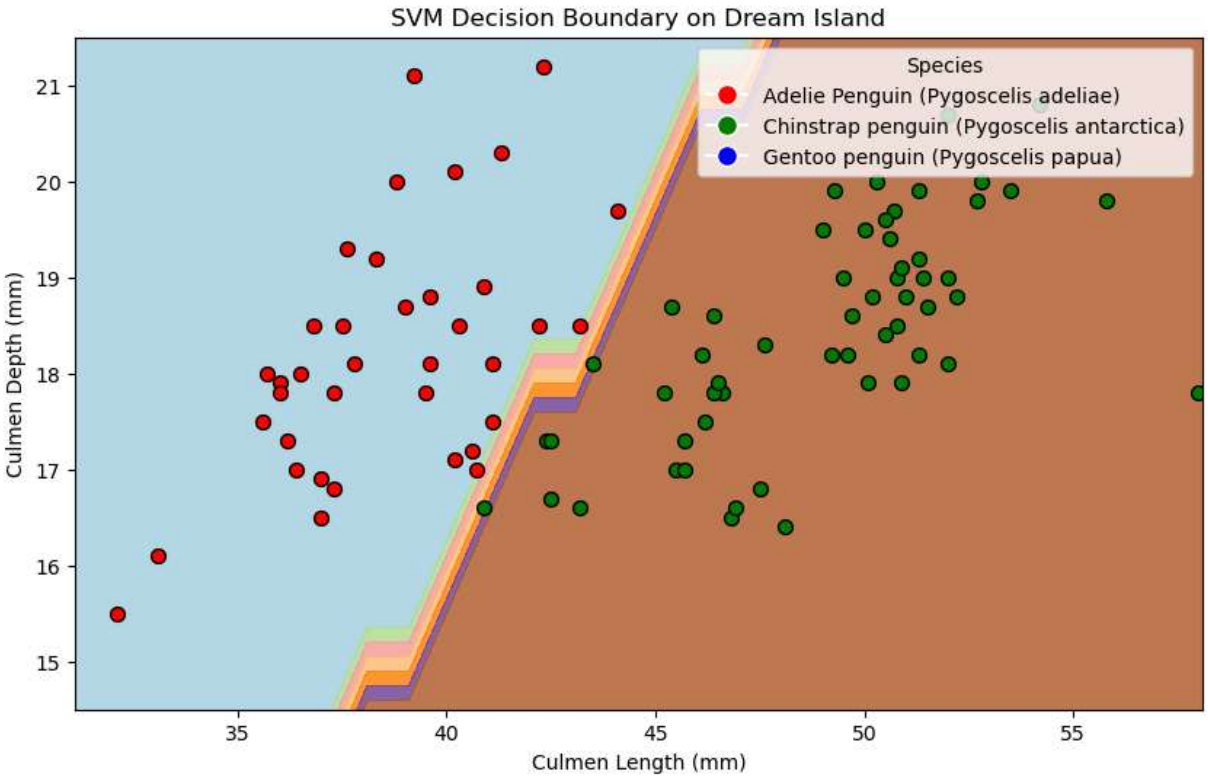


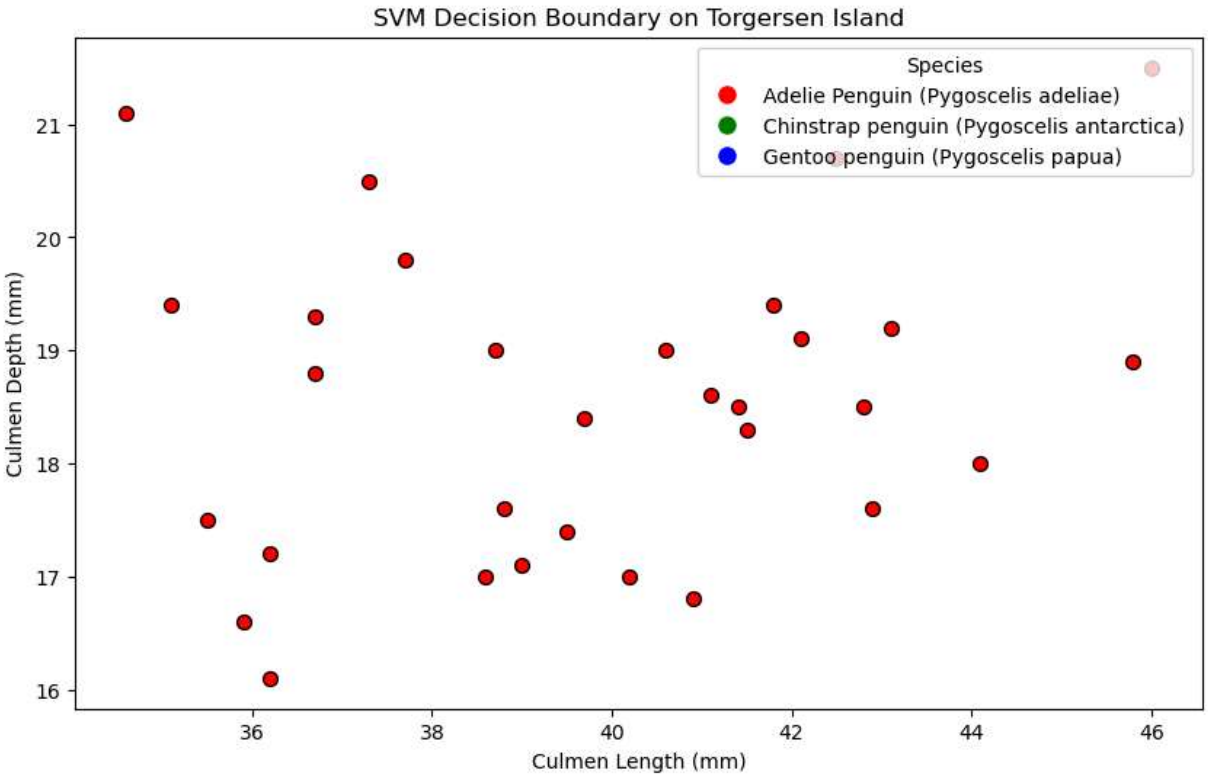
In [29]:

```
# Visualize the decision boundaries with two features for each island
h = 1.0  # increased step size for faster computation

# We will visualize using two features: 'Culmen Length (mm)' and 'Culmen Depth (mm)'
X_vis = transformed_traindata[['Culmen Length (mm)',
                               'Culmen Depth (mm)']].values
island = transformed_traindata['Island'].values
y_vis = transformed_traindata['Species'].values

island_labels = le_island.classes_
species_colors = ['red', 'green', 'blue']
species_labels = le_species.classes_

# Create a plot for each island
for i, island_label in enumerate(island_labels):

    # Filter data for the current island
    X_island = X_vis[island == i]
    y_island = y_vis[island == i]

    plt.figure(figsize=(10, 6))

    if len(np.unique(y_island)) > 1:
        # Train the SVM model on these two
        #features for the current island
        svm_model_vis = SVC(kernel='linear')
        svm_model_vis.fit(X_island, y_island)

        # Plot the decision boundary
```

```
        x_min, x_max = X_island[:, 0].min() - 1, X_island[:, 0].max() + 1
        y_min, y_max = X_island[:, 1].min() - 1, X_island[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                             np.arange(y_min, y_max, h))

        Z = svm_model_vis.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)

        plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

    # Plot the training points with colors for species
    for j, color in enumerate(species_colors):
        idx = (y_island == j)
        plt.scatter(X_island[idx, 0], X_island[idx, 1],
                    color=color, edgecolor='k', s=50, label=species_labels[j])



    # Label the axes
    plt.xlabel('Culmen Length (mm)')
    plt.ylabel('Culmen Depth (mm)')
    plt.title(f'SVM Decision Boundary on {island_label} Island')

    # Create a legend for species
    handles_species = [plt.Line2D([0], [0], marker='o', color='w',
                       markerfacecolor=species_colors[k], markersize=10,
                       label=species_labels[k]) for k in range(len(species_labels))
    plt.legend(handles_species, species_labels, title="Species", loc='upper right')

    plt.show()
```







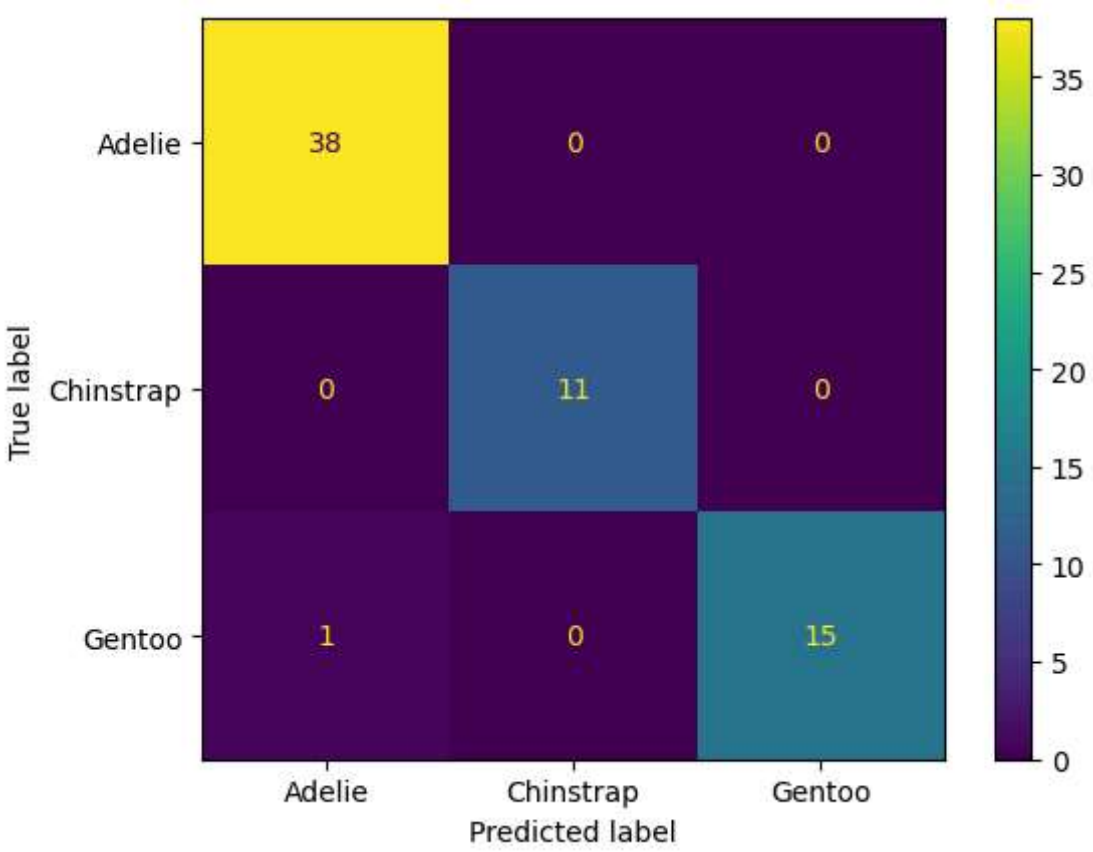
Random Forest

In [30]:
```
m = RandomForestClassifier(n_estimators = 9)
myX = transformed_traindata[["Island", "Culmen Length (mm)", "Culmen Depth (mm)"]]
myY = transformed_traindata["Species"]

transformed_testdata = testdata.copy()
transformed_testdata['Island'] = le_island.fit_transform(transformed_testdata['Isla
```

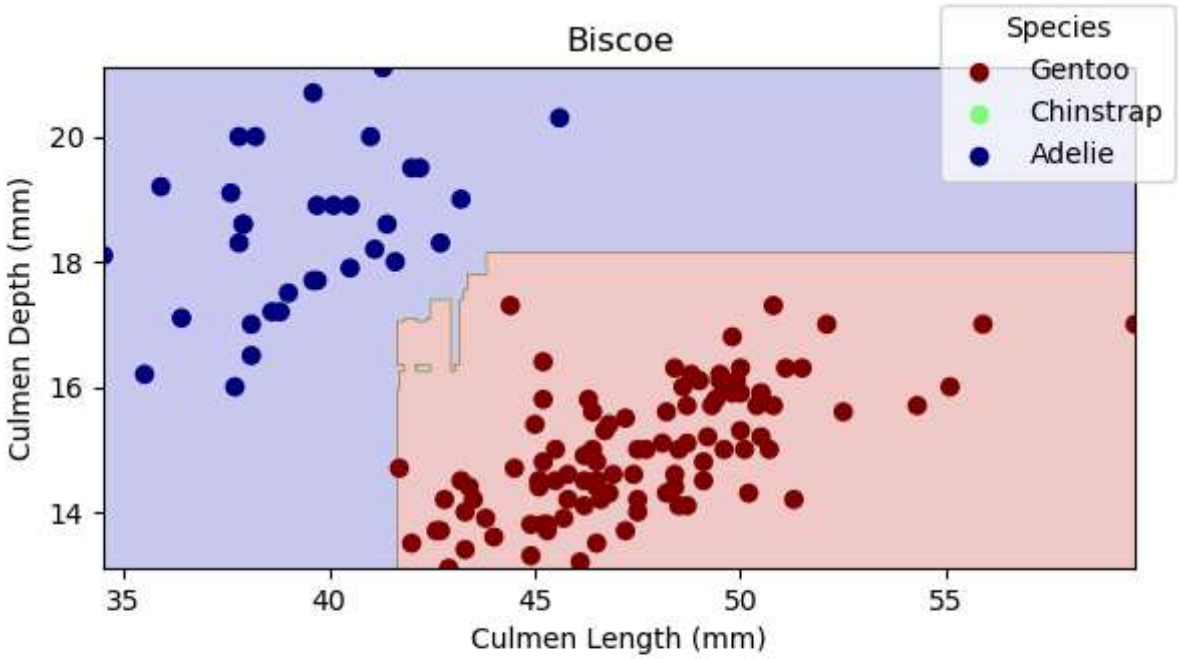
```
transformed_testdata['Sex'] = le_sex.fit_transform(transformed_testdata['Island'])
transformed_testdata['Species'] = le_species.fit_transform(transformed_testdata['Sp

m.fit(myX, myY)
myX_test = transformed_testdata[["Island",
                                 "Culmen Length (mm)", "Culmen Depth (mm)"]]
myY_test = transformed_testdata["Species"]
predicted_labels_test = m.predict(myX_test)
print("CV Score", optimalForest[0])

from sklearn.metrics import accuracy_score
print("Accuracy Score:", accuracy_score(myY_test, predicted_labels_test))
labels = ["Adelie", "Chinstrap", "Gentoo"]
from sklearn.metrics import ConfusionMatrixDisplay
predicted_labels_test = m.predict(myX_test)
ConfusionMatrixDisplay.from_predictions(myY_test,
                    predicted_labels_test, display_labels = labels)
```

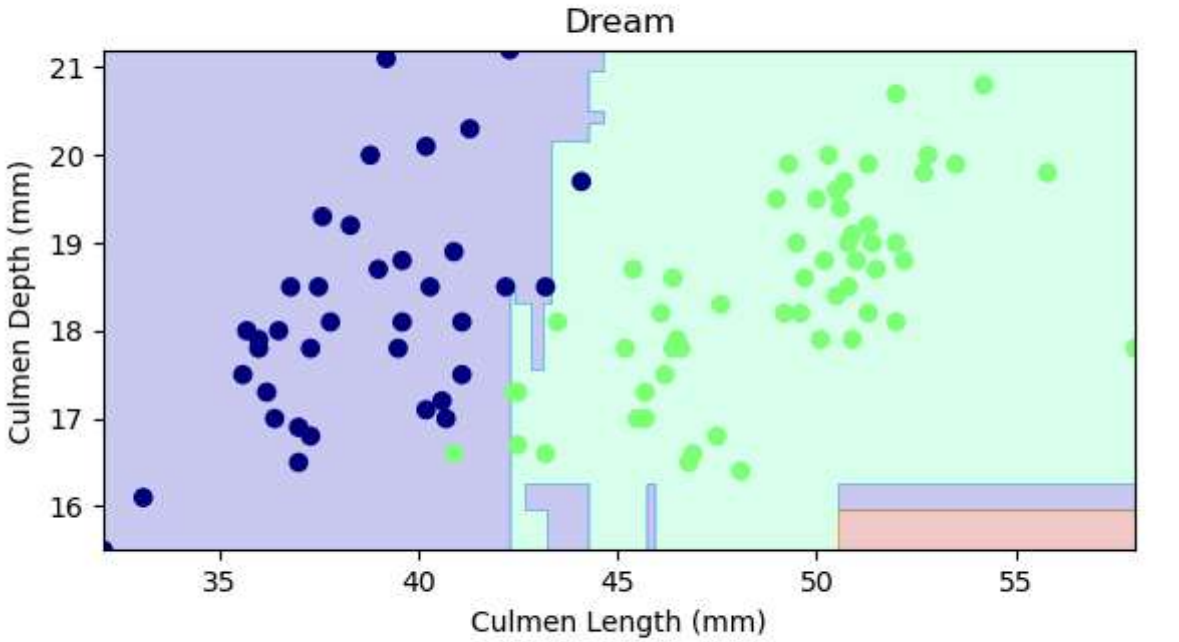
```
CV Score 0.9884615384615385
Accuracy Score: 0.9846153846153847
```

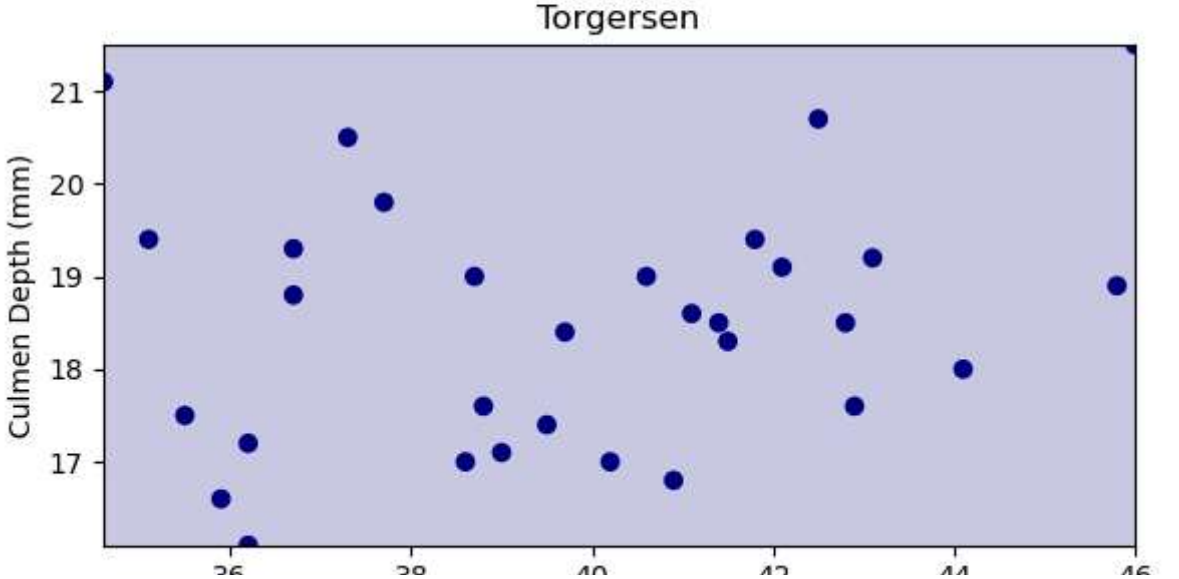
Out[30]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15ce8028590>

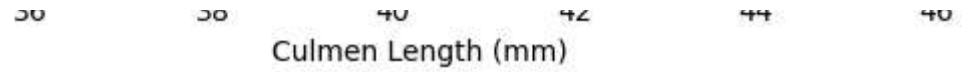


In [31]:
```
plot_regions(m, transformed_traindata, "Culmen Length (mm)",
             "Culmen Depth (mm)", "Island", le_island.classes_)
```


Biscoe
Species
Gentoo
Chinstrap
Adelie
Culmen Depth (mm)
Culmen Length (mm)
20
18
16
14
35
40
45
50
55


Dream
Culmen Depth (mm)
Culmen Length (mm)
21
20
19
18
17
16
35
40
45
50
55


Torgersen
Culmen Depth (mm)
21
20
19
18
17



Discussion

KNN: We found that the KNN model was generally quite good. It had a high accuracy score when testing on both cross-validation scores and the actual test data score. The confusion matrix revealed that the KNN model perfectly classified the test region. The boundary regions showed us that the KNN model got confused once by thinking that a Gentoo penguin was a Chinstrap (despite there being no Chinstrap on the island; we are unsure how that happened), and two times where a Chinstrap penguin had similar features to an Adelie penguin. This makes sense because those Chinstrap penguins were closer to the Adelie penguins on the graph than the other Chinstrap penguins.

SVM: We found that the SVM model was also good. It had a high accuracy score when testing on both cross-validation scores and the actual test data score. The confusion matrix revealed that the SVM model misclassified a Chinstrap penguin as an Adelie penguin. The boundary regions showed us that the SVM model may have gotten confused because some of the Chinstrap penguins were right on the decision boundary between the Chinstrap and Adelie regions for the support vectors.

Random Forest: We found that the Random Forest model was also good. It had a high accuracy score when testing on both cross-validation scores and the actual test data score. The confusion matrix revealed that the Random Forest model had one mistake on the confusion matrix, which shows that it is very accurate. The boundary regions showed us that the Random Forest model may not have had perfect scores on cross validation scores for the training data because some of the Chinstrap penguins were really close to the Adelie penguins.

Note: In general, it seems that there were a few anomalic Chinstrap Penguins misclassified as Adelie penguins that served as things that reduced the accuracy of our models. However, our models overall were quite accurate, especially on the actual test data, which is quite comforting.

In [32]:
```
pip install -U notebook-as-pdf
```

```
Requirement already satisfied: notebook-as-pdf in c:\users\hallj\anaconda3\lib\site-
packages (0.5.0)
Requirement already satisfied: nbconvert in c:\users\hallj\anaconda3\lib\site-packag
es (from notebook-as-pdf) (7.10.0)
Requirement already satisfied: pyppeteer in c:\users\hallj\anaconda3\lib\site-packag
es (from notebook-as-pdf) (2.0.0)
Requirement already satisfied: PyPDF2 in c:\users\hallj\anaconda3\lib\site-packages
(from notebook-as-pdf) (3.0.1)
Requirement already satisfied: beautifulsoup4 in c:\users\hallj\anaconda3\lib\site-p
ackages (from nbconvert->notebook-as-pdf) (4.12.2)
Requirement already satisfied: bleach!=5.0.0 in c:\users\hallj\anaconda3\lib\site-pa
ckages (from nbconvert->notebook-as-pdf) (4.1.0)
Requirement already satisfied: defusedxml in c:\users\hallj\anaconda3\lib\site-packa
ges (from nbconvert->notebook-as-pdf) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in c:\users\hallj\anaconda3\lib\site-pack
ages (from nbconvert->notebook-as-pdf) (3.1.3)
Requirement already satisfied: jupyter-core>=4.7 in c:\users\hallj\anaconda3\lib\sit
e-packages (from nbconvert->notebook-as-pdf) (5.5.0)
Requirement already satisfied: jupyterlab-pygments in c:\users\hallj\anaconda3\lib\s
ite-packages (from nbconvert->notebook-as-pdf) (0.1.2)
Requirement already satisfied: markupsafe>=2.0 in c:\users\hallj\anaconda3\lib\site-
packages (from nbconvert->notebook-as-pdf) (2.1.3)
Requirement already satisfied: mistune<4,>=2.0.3 in c:\users\hallj\anaconda3\lib\sit
e-packages (from nbconvert->notebook-as-pdf) (2.0.4)
Requirement already satisfied: nbclient>=0.5.0 in c:\users\hallj\anaconda3\lib\site-
packages (from nbconvert->notebook-as-pdf) (0.8.0)
Requirement already satisfied: nbformat>=5.7 in c:\users\hallj\anaconda3\lib\site-pa
ckages (from nbconvert->notebook-as-pdf) (5.9.2)
Requirement already satisfied: packaging in c:\users\hallj\anaconda3\lib\site-packag
es (from nbconvert->notebook-as-pdf) (23.1)
Requirement already satisfied: pandocfilters>=1.4.1 in c:\users\hallj\anaconda3\lib
\site-packages (from nbconvert->notebook-as-pdf) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in c:\users\hallj\anaconda3\lib\site-
packages (from nbconvert->notebook-as-pdf) (2.15.1)
Requirement already satisfied: tinycss2 in c:\users\hallj\anaconda3\lib\site-package
s (from nbconvert->notebook-as-pdf) (1.2.1)
Requirement already satisfied: traitlets>=5.1 in c:\users\hallj\anaconda3\lib\site-p
ackages (from nbconvert->notebook-as-pdf) (5.7.1)
Requirement already satisfied: appdirs<2.0.0,>=1.4.3 in c:\users\hallj\anaconda3\lib
\site-packages (from pyppeteer->notebook-as-pdf) (1.4.4)
Requirement already satisfied: certifi>=2023 in c:\users\hallj\anaconda3\lib\site-pa
ckages (from pyppeteer->notebook-as-pdf) (2024.2.2)
Requirement already satisfied: importlib-metadata>=1.4 in c:\users\hallj\anaconda3\l
ib\site-packages (from pyppeteer->notebook-as-pdf) (7.0.1)
Requirement already satisfied: pyee<12.0.0,>=11.0.0 in c:\users\hallj\anaconda3\lib
\site-packages (from pyppeteer->notebook-as-pdf) (11.1.0)
Requirement already satisfied: tqdm<5.0.0,>=4.42.1 in c:\users\hallj\anaconda3\lib\s
ite-packages (from pyppeteer->notebook-as-pdf) (4.65.0)
Requirement already satisfied: urllib3<2.0.0,>=1.25.8 in c:\users\hallj\anaconda3\li
b\site-packages (from pyppeteer->notebook-as-pdf) (1.26.18)
Requirement already satisfied: websockets<11.0,>=10.0 in c:\users\hallj\anaconda3\li
b\site-packages (from pyppeteer->notebook-as-pdf) (10.4)
Requirement already satisfied: six>=1.9.0 in c:\users\hallj\anaconda3\lib\site-packa
ges (from bleach!=5.0.0->nbconvert->notebook-as-pdf) (1.16.0)
Requirement already satisfied: webencodings in c:\users\hallj\anaconda3\lib\site-pac
kages (from bleach!=5.0.0->nbconvert->notebook-as-pdf) (0.5.1)
```

```
Requirement already satisfied: zipp>=0.5 in c:\users\hallj\anaconda3\lib\site-packag
es (from importlib-metadata>=1.4->pyppeteer->notebook-as-pdf) (3.17.0)
Requirement already satisfied: platformdirs>=2.5 in c:\users\hallj\anaconda3\lib\sit
e-packages (from jupyter-core>=4.7->nbconvert->notebook-as-pdf) (3.10.0)
Requirement already satisfied: pywin32>=300 in c:\users\hallj\anaconda3\lib\site-pac
kages (from jupyter-core>=4.7->nbconvert->notebook-as-pdf) (305.1)
Requirement already satisfied: jupyter-client>=6.1.12 in c:\users\hallj\anaconda3\li
b\site-packages (from nbclient>=0.5.0->nbconvert->notebook-as-pdf) (8.6.0)
Requirement already satisfied: fastjsonschema in c:\users\hallj\anaconda3\lib\site-p
ackages (from nbformat>=5.7->nbconvert->notebook-as-pdf) (2.16.2)
Requirement already satisfied: jsonschema>=2.6 in c:\users\hallj\anaconda3\lib\site-
packages (from nbformat>=5.7->nbconvert->notebook-as-pdf) (4.19.2)
Requirement already satisfied: typing-extensions in c:\users\hallj\anaconda3\lib\sit
e-packages (from pyee<12.0.0,>=11.0.0->pyppeteer->notebook-as-pdf) (4.9.0)
Requirement already satisfied: colorama in c:\users\hallj\anaconda3\lib\site-package
s (from tqdm<5.0.0,>=4.42.1->pyppeteer->notebook-as-pdf) (0.4.6)
Requirement already satisfied: soupsieve>1.2 in c:\users\hallj\anaconda3\lib\site-pa
ckages (from beautifulsoup4->nbconvert->notebook-as-pdf) (2.5)
Requirement already satisfied: attrs>=22.2.0 in c:\users\hallj\anaconda3\lib\site-pa
ckages (from jsonschema>=2.6->nbformat>=5.7->nbconvert->notebook-as-pdf) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in c:\users\hall
j\anaconda3\lib\site-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert->noteb
ook-as-pdf) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in c:\users\hallj\anaconda3\lib\s
ite-packages (from jsonschema>=2.6->nbformat>=5.7->nbconvert->notebook-as-pdf) (0.3
0.2)
Requirement already satisfied: rpds-py>=0.7.1 in c:\users\hallj\anaconda3\lib\site-p
ackages (from jsonschema>=2.6->nbformat>=5.7->nbconvert->notebook-as-pdf) (0.10.6)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hallj\anaconda3\li
b\site-packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert->notebook-a
s-pdf) (2.8.2)
Requirement already satisfied: pyzmq>=23.0 in c:\users\hallj\anaconda3\lib\site-pack
ages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert->notebook-as-pdf) (25.
1.2)
Requirement already satisfied: tornado>=6.2 in c:\users\hallj\anaconda3\lib\site-pac
kages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert->notebook-as-pdf) (6.
3.3)
Note: you may need to restart the kernel to use updated packages.
```

In [33]:
```
pyppeteer-install
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[33], line 1
----> 1 pyppeteer-install

NameError: name 'pyppeteer' is not defined
```

In [ ]: