Discussion 4B Group 5 Final Project

Aashna Sibal, Emily Olds, and Sung Been Lee

Group Contributions Statement

All three of us wrote the data import and cleaning portion and worked on creating Tables 1 and 2 and Figure 4. Emily led Figure 1 and the random forests model. Sung Been led Figure 2 and the support vector machines model. Aashna led Figure 3 and the multinomial logistic regression model. We all worked on the comments and discussions for our respective sections and evenly split the rest of the explanations, writing, and markdown cells. We all reviewed each other’s work and made revisions where necessary.

Project Overview

Machine learning is a powerful field, made up of tools and methods that allow us to make accurate predictions using existing data.

In this project, we will be demonstrating how to use some of these machine learning techniques. Through Python, we will showcase a systematic way of accurately predicting the species of a penguin using variables such as its species, sex, location, and other physical characteristics. Firstly, we will clean and explore the data through standard data cleaning procedures and exploratory data analysis. Then, using an automated feature selection technique, we will determine two quantitative variables and one qualitative variable from the dataset that are the best predictors of a penguin's species. We will use these variables to build three different machine learning models that predict a penguin's species and reflect on their strengths, weaknesses, and overall performance.

Data Import and Cleaning

Before we explore and familiarize ourselves with data, we will need to import the dataset, split it, and clean it. Since we will be using a `pandas` data frame, we are going to import `pandas` and `numpy`. Then, we will download the data from a URL and assign it to a data frame called `penguins`. Finally, we will split the dataset into two: `train` and `test`. As the names suggest, `train` is the data we will use to train our model, and `test` is what we will assess the model's performance on.

In [1]:
```python
# importing relevant packages to save data in a data frame and to train-test split
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# importing the data from url
url = "https://www.philchodrow.com/PIC16A/content/IO_and_modules/IO/palmer_penguins.csv"
penguins = pd.read_csv(url)

# set seed for reproducibility
np.random.seed(1234)

# train-test split with 70% training data and 30% test data
train, test = train_test_split(penguins, test_size = .3)
```

Data Cleaning

Now that the data is in the workspace and properly split, we will need to clean it. This includes, but is not limited to, removing observations we do not want to use, removing rows with `NaN` values and shortening the full scientific species names to only their first word, for simplicity purposes. We also will remove a row where the penguins' sex is incorrectly marked as `.`. To build machine learning models, we also need all of our data to be encoded as integers. Therefore, we will change qualitative columns like `Sex`, `Island`, and `Species` into integers beginning at 0, with each integer representing a different value of that category. For instance, for `Sex`, `MALE` and `FEMALE` values will be encoded as `0`s and `1`s. Since we will have to clean both the training and test data, we will define the function `clean_data()` that we can then call for each dataset. If the `split` argument is `True` in the function, we will separate the data into two data frames depending which dataset is inputted (`X_test` or `X_train` and `y_test` or `y_train`). The `X` variables will be the predictor variables, which is every column of the data except `Species`. And `y` variables will be the target variables, with just `Species` column, as our goal is to predict the species.

In [2]:
```python
# importing preprocessing from sklearn to encode qualitative variables as quantitative
from sklearn import preprocessing

def clean_data(data_df, split = True):
    '''
    clean_data takes in a data frame and cleans it by making a copy of it, specifying the columns of interest,
    removing rows with NaN values, shortening the species name, changing qualitative variables to
    quantitative, and either separating the data into two data frames: one with target data and one with
    predictor data and returning these two data frames, or returning the cleaned, unseparated data frame as
    specified by the split argument.
    parameter data_df : a data frame to clean
    split : boolean value to specify whether the function should return two data frames (predictor, target) or
    a single data frame. Defaults to True.
    '''
    #dataframes are mutable, so the function begins by making a copy before cleaning
    df = data_df.copy()

    #here the function specifies which columns are of interest in our analysis.
    #we are not interested in the "comments" column, so it is not included in the dataframe.
    df = df[['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)']]

    #the function then drops rows with NaN values.
    #axis = 0 because the function should drop rows, not columns.
    df = df.dropna(axis = 0)

    #one of the sex entries had a "." by mistake, so here that row is dropped from the data frame.
    df = df[df['Sex']!='.']

    #the code below shortens the species names by only using the first word of the species.
    df['Species'] = df['Species'].str.split().str.get(0)

    #when split is true, we want to return X with predictor data, and y with the target data
    if split:
        #setting up the LabelEncoder function to transform qualitiative variables to quantitative.
        le = preprocessing.LabelEncoder()

        #transforming "Sex", "Island", and "Species", the three qualitative variables, to quantitative variables.
        df['Sex'] = le.fit_transform(df['Sex'])
        df['Species'] = le.fit_transform(df['Species'])
        df['Island'] = le.fit_transform(df['Island'])
        #splitting the dataframe into two: one with predictor data (X) and one with target data (y).
        X = df.drop(columns = ['Species'])
        y = df['Species'] #species is target data since it's what we want to predict
        return (X, y)
    else: #when split is not true
        return df #clean the whole data frame and leave it as one, with all the columns we are interested in
```

In [3]:
```
#using the function to clean the train and test sets
X_train, y_train = clean_data(train)
X_test, y_test = clean_data(test)
```

For the exploratory analysis only, we will use the entirety of the training dataset, one that is *not* split into predictor and target variables, and is clean. We will achieve this by calling the `clean_data()` function with the `split` argument set to `False`. The resulting data frame will be stored in the `clean_train` variable.

In [4]:
```
clean_train = clean_data(train, split = False)
clean_train
```

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)
316	Gentoo	Biscoe	49.4	15.8	216.0	4925.0	MALE	8.03624	-26.06594
232	Gentoo	Biscoe	45.5	13.7	214.0	4650.0	FEMALE	7.77672	-25.41680
300	Gentoo	Biscoe	49.1	14.5	212.0	4625.0	FEMALE	8.35802	-26.27660
199	Chinstrap	Dream	49.0	19.6	212.0	4300.0	MALE	9.34089	-24.45189
17	Adelie	Torgersen	42.5	20.7	197.0	4500.0	MALE	8.67538	-25.13993
...	...	...	...	...	...	...	...	...	...
204	Chinstrap	Dream	45.7	17.3	193.0	3600.0	FEMALE	9.41500	-24.80500
53	Adelie	Biscoe	42.0	19.5	200.0	4050.0	MALE	8.48095	-26.31460
294	Gentoo	Biscoe	46.4	15.0	216.0	4700.0	FEMALE	8.47938	-26.95470
211	Chinstrap	Dream	45.6	19.4	194.0	3525.0	FEMALE	9.46985	-24.65786
303	Gentoo	Biscoe	50.0	15.9	224.0	5350.0	MALE	8.20042	-26.39677

225 rows × 9 columns

Exploratory Analysis

As previously stated, we will be selecting three variables to use to construct our model——two of which are quantitative and one of which is qualitative. Before we use automated feature selection to decide which variables are the best suited to make the most accurate model, we will explore the data through tables and graphs. As seen in the last step, we saw that the variables we could potentially use are `Island`, `Culmen Length (mm)`, `Culmen Depth (mm)`, `Flipper Length (mm)`, `Body Mass (g)`, `Sex`, `Delta 15 N (o/oo)`, and `Delta 13 C (o/oo)`.

Table 1

First, we will make a summary table of `clean_train` that show the standard statistical summaries, including the count, mean, standard deviation, minimum and maximum, and 25th, 50th, and 75th percentile of each column (or variable) of the data frame listed above.

In [5]:
```
# generate descriptive statistics
clean_train.describe()
```

Out[5]:

	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
count	225.000000	225.000000	225.000000	225.000000	225.000000	225.000000
mean	44.236889	17.082667	201.560000	4220.000000	8.726639	-25.691214
std	5.342102	2.021075	13.794318	811.424425	0.546875	0.769289
min	32.100000	13.100000	172.000000	2700.000000	7.632200	-27.018540
25%	39.600000	15.300000	190.000000	3550.000000	8.299300	-26.275730
50%	45.100000	17.200000	198.000000	4050.000000	8.664960	-25.881560
75%	48.700000	18.600000	214.000000	4800.000000	9.153080	-25.112230
max	59.600000	21.500000	230.000000	6050.000000	10.025440	-23.890170

Table 2

Next, we will create a display table that groups our data by `Species`, `Island`, and `Sex`, and displays the mean of every column for every species-island-sex pair rounded to the nearest two decimal places.

```
In [6]: clean_train.groupby(['Species','Island','Sex'])[['Culmen Length (mm)',
                                                 'Culmen Depth (mm)',
                                                 'Flipper Length (mm)',
                                                 'Body Mass (g)',
                                                 'Delta 15 N (o/oo)',
                                                 'Delta 13 C (o/oo)']].mean().round(2)
```

Out[6]:

Species	Island	Sex	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
Adelie	Biscoe	FEMALE	38.12	17.77	185.31	3394.23	8.76	-25.88
		MALE	40.47	19.17	190.85	4034.62	8.88	-25.96
	Dream	FEMALE	36.68	17.58	188.00	3353.95	8.93	-25.83
		MALE	40.17	19.02	191.38	3945.31	9.05	-25.70
	Torgersen	FEMALE	38.08	17.58	189.12	3348.44	8.77	-25.71
		MALE	40.67	19.41	196.75	4070.31	8.95	-25.84
Chinstrap	Dream	FEMALE	46.45	17.53	191.96	3471.74	9.29	-24.61
		MALE	51.11	19.15	199.39	3938.04	9.39	-24.52
Gentoo	Biscoe	FEMALE	45.63	14.26	213.20	4711.67	8.17	-26.16
		MALE	49.63	15.80	220.93	5485.98	8.27	-26.16

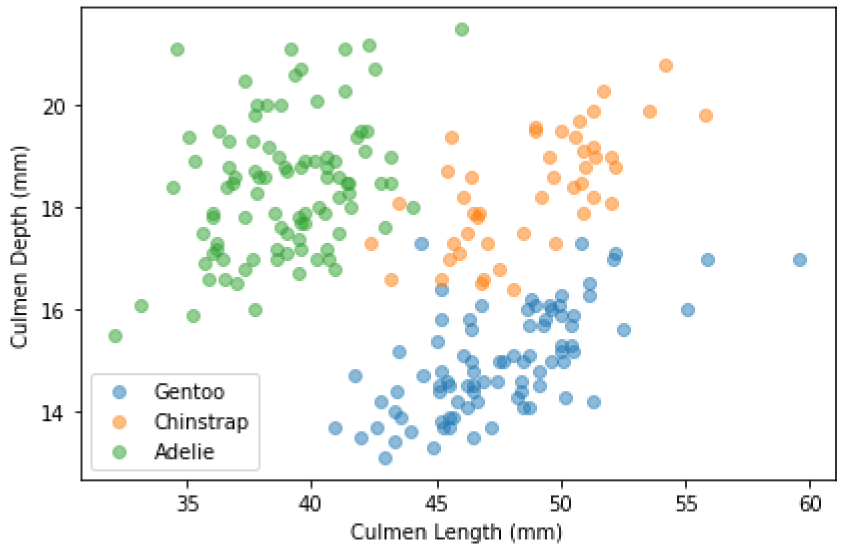
This shows some important distinctions:

1. Adelie penguins are the only species that is present on all three islands. Chinstraps are only on Dream, Gentoo only on Biscoe.
2. Adelie penguins appear to have lower culmen lengths, on average. We will explore this further.
3. Gentoo penguins appear to have the lowest culmen depths, on average.
4. Gentoo penguins appear to have much higher flipper lengths and body masses, on average.
5. The `Delta 15 N (o/oo)` and `Delta 13 C (o/oo)` columns do not show very dramatic distinctions. Overall, Chinstrap penguins have the highest average Delta 15 and 13 values, Gentoos being the lowest. Because the mean values are very similar, these are probably not the best variables to use.
6. There are clear differences among male and female penguins for every species, with the dergree of disparity depending on the column variable.

Figure 1

In [7]:
```
# importing package needed for plotting
from matplotlib import pyplot as plt

# initializing the figure and axes
fig, ax = plt.subplots(1)

# labelling x and y-axis according to the columns to be explored
ax.set(xlabel = "Culmen Length (mm)",
       ylabel = "Culmen Depth (mm)")

# creating an array containing distinct species names from clean_train
n_species = clean_train["Species"].unique()

# iterating through n_species
for k in range(len(n_species)):
    # extracting current species data from clean_train
    species_data = clean_train[clean_train["Species"] == n_species[k]]
    # plotting the current species' data on a scatterplot
    ax.scatter(species_data["Culmen Length (mm)"],
               species_data["Culmen Depth (mm)"],
               label = n_species[k], # labelling by the species name
               alpha = .5)           # adding transparency to data points

# making plot look tidier
plt.tight_layout()

# adding legend containing each species' color distinction
ax.legend()
```

Out[7]:
```
<matplotlib.legend.Legend at 0x7f8fb5203370>
```

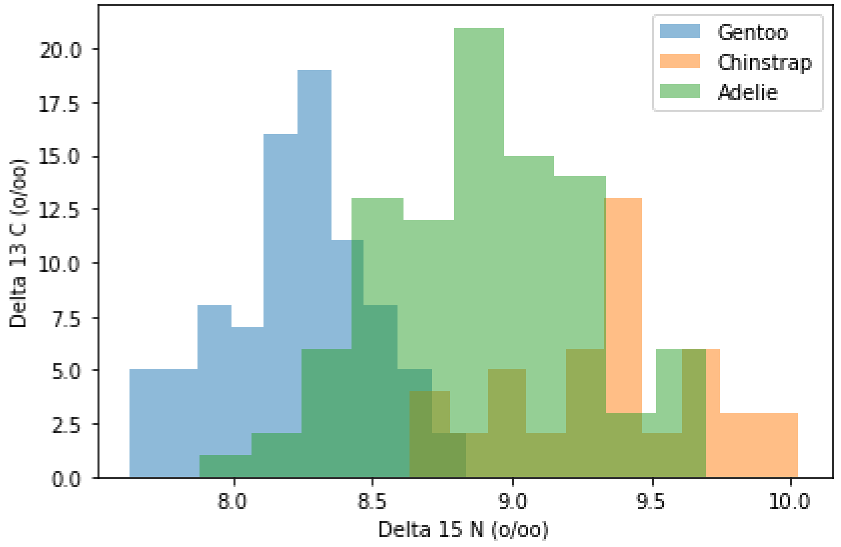


The plot above shows that, on average, Adelie penguins tend to have the lowest culmen lengths and the highest culmen depths, and Gentoo penguins tend to have the lowest culmen depths, according to our training dataset. Because we can visualize where the general differences between each species lie, we can conclude that these variables are probably good variables to use to build our models.

Figure 2

In [8]:
```
# initializing the figure and axes
fig, ax = plt.subplots(1)

# labelling x and y-axis according to the columns to be explored
ax.set(xlabel = "Delta 15 N (o/oo)",
       ylabel = "Delta 13 C (o/oo)")

# creating an array containing distinct species names from clean_train
n_species = clean_train["Species"].unique()

# iterating through n_species
for k in range(len(n_species)):
    # extracting current species data from clean_train
    species_data = clean_train[clean_train["Species"] == n_species[k]]
    # plotting the current species' data on a histogram
    ax.hist(species_data["Delta 15 N (o/oo)"],
            label = n_species[k], # labelling by the species name
            alpha = .5)           # adding transparency to data points

# making the plot look nicer and adding a legend for the labels
plt.tight_layout()
ax.legend()
```

Out[8]:
```
<matplotlib.legend.Legend at 0x7f8fb52f7250>
```

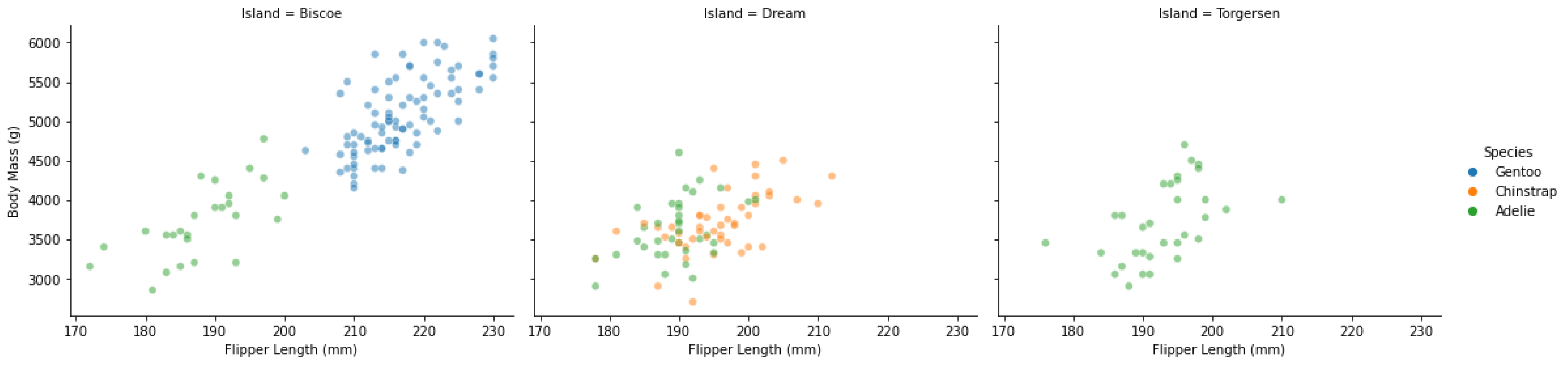


From the histogram, we see a many overlaps in data points between the species and do not see many distinctions. This reinforces our findings from Table 2 that the `Delta 15 N (o/oo)` and `Delta 13 C (o/oo)` columns are not best suited to use as our predictor variables.

Figure 3

In [9]:
```
# import to use seaborn's replot() method to plot multiple axes
import seaborn as sns

sns.relplot(data = clean_train,
            x = "Flipper Length (mm)", # Flipper Length as x-axis
            y = "Body Mass (g)",       # Body Mass as y-axis
            hue = "Species",           # colored by species
            alpha = .5,                # adding transparency to data points
            col = "Island",            # subplots grouped by Island
            aspect = 1.3,
            height = 4)
```

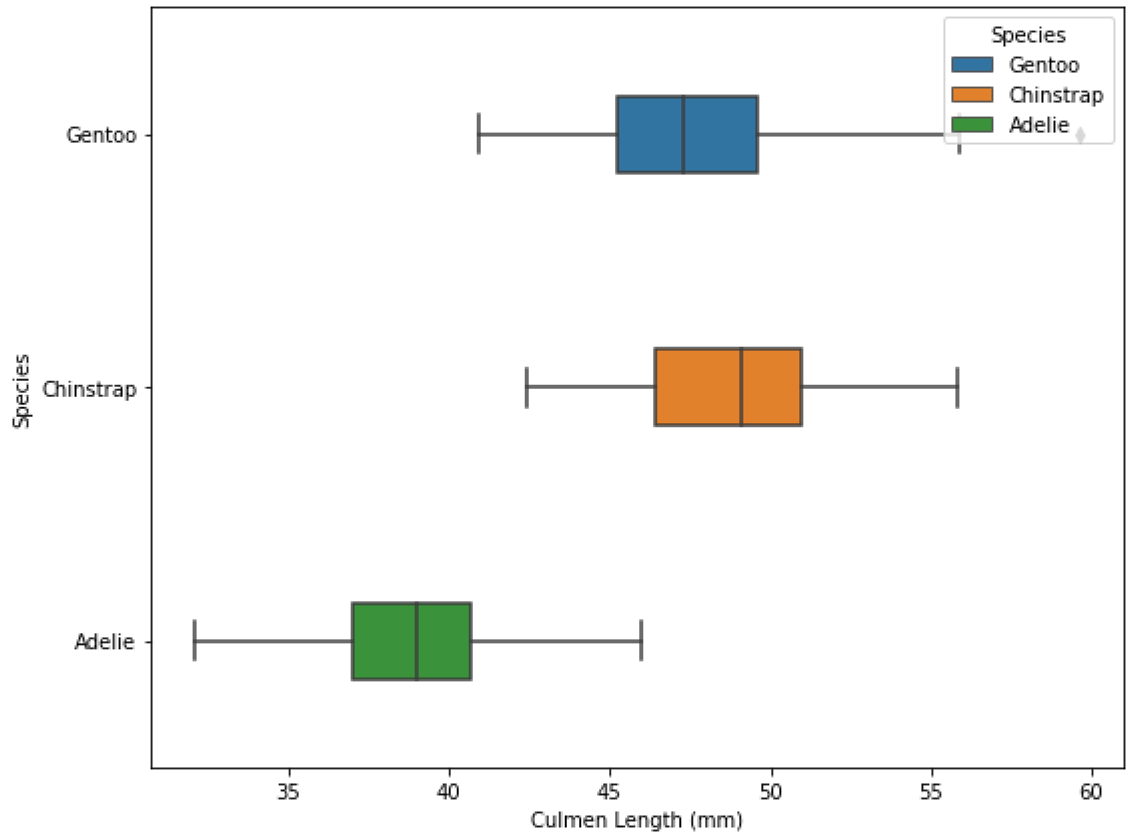
Out[9]:
```
<seaborn.axisgrid.FacetGrid at 0x7f8fb5676160>
```



The faceted grid above shows considerable differences among flipper lengths and body masses for the Gentoo and Adelie penguins. This could be thus be useful information when selecting features.

Figure 4

In [10]:
```
# initializing the figure and axes
fig, ax = plt.subplots(figsize = (8, 6))

#using seaborns to make a boxplot with data clean_train where Culmen Length is on the x-axis
#and species is on the y-axis
#box for each species and colors correspond to species too
sns.boxplot(data = clean_train,
            x = 'Culmen Length (mm)', # Culmen Length as x-axis
            y = 'Species',            # Species as y-axis
            hue = 'Species',          # colored by species
            width = .3,
            dodge = False)

# adding a legend and setting to appear on the upper right and labelling the title
plt.legend(loc = "upper right", title = "Species")
# making the plot look nicer
plt.tight_layout()
```



The boxplot above further explores the `Culmen Length (mm)` variable from Figure 1 and reinforces our conclusion that, on average, Adelie penguins have the lowest culmen lengths out of the three species.

Feature Selection

In order to select columns that generate the highest scores on the test set, we use a systematic method in which we define a list of all possible combinations of columns and subsequently check which combination returns the best score. We use Logistic Regression for this task as it is well-suited for predicting probabilities and class labels.

In [32]:
```
# import to use the LogisticRegression() method
from sklearn.linear_model import LogisticRegression
```

```python
def check_col_score(columns):
    """
    a function that receives a set of columns of a data frame and returns a score indicating how well the
    Logistic Regression model, trained by the training data, fits the testing data.
    parameter columns : a list containing column names
    """
    # set model using sklearn's LogisticRegression() method
    LR = LogisticRegression(max_iter = 1000)
    # fit the model with the training data, subsetted with the columns argument
    LR.fit(X_train[columns], y_train)
    # return a score indicating how well the testing data performs on that model
    return LR.score(X_test[columns], y_test)
```

In [33]:
```python
# represents a list form of X_train's columns
    # pandas attribute .columns : returns the column labels of X_train
    # pandas attribute .tolist() : converts those column labels into a list
cols = X_train.columns.tolist()

# initialize list to contain the lists of combinations of columns
combos = []

# create all possible combinations of columns; three nested for-loops, each representing one column of three in
# a single combination
for i in range(len(cols)):
    for j in range(len(cols)):
        for k in range(len(cols)):
            # when the three indices are not the same (i.e. different columns), append to combos
            if i != j and j != k and i != k:     # if i < j and j < k: ???
                combos.append([cols[i], cols[j], cols[k]])

# initialize best score to lowest possible value to ensure initial replacement
best_score = -np.inf

# iterate through the list of combinations
for combo in combos:
    # get test socre of current combination using check_col_score()
    test_score = check_col_score(combo)

    # check to see if the current test score is better than the current best score
    if test_score > best_score:
        # reassign best values
        best_score = test_score
        best_cols = combo
```

```
# show result
print("The best columns are " + str(best_cols) + " with a testing score of " + str(best_score))
```

```
The best columns are ['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex'] with a testing score of 0.9797979797979798
```

Based on the above results, we will use `Culmen Length (mm)`, `Culmen Depth (mm)`, and `Sex` as the features for all of our models, as this combination of features, according to our algorhithm, produces the highest score of 0.98 across all combinations of features.

Modeling

In [34]:

```
# setting X_train and X_test to include only the selected features
X_train = X_train[['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex']]
X_test = X_test[['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex']]
```

In [35]:

```
# import to use cross_val_score() method to perform cross validation
from sklearn.model_selection import cross_val_score

def get_best_parameters(X, y, model):
    """
    a function that receives predictor data, target data, and the type of model and returns a tuple containing
    the value that generated the highest cross validation score with the selected parameter and the cross
    validation score returned when using that value.

    parameter 1, X : predictor data
    parameter 2, y : target data
    model : a string representing the name of the model to perform cross validation on
    """
    # initialize best_score to lowest possible value to ensure replacement
    best_score = -np.inf

    # for the Random Forest model
    if model == "RandomForestClassifier":
        # largest max depth
        N = 30
        # intializing array with size 30 with zeros
        scores = np.zeros(N)
        # for every possible depth from 1 to N
        for d in range(1, N+1):
            # initialize model with current max depth
            RF = RandomForestClassifier(random_state = 42, max_depth = d)
            # assign cross validation score corresponding to current max depth
            scores[d-1] = cross_val_score(RF, X_train, y_train, cv = 5).mean()
            # if the current score is greater than the current best_score
            if scores[d-1] > best_score:
                # assign best_depth as current depth (d)
                best_depth = d
                # assign best_score as current score
                best_score = scores[d-1]
        # display results and return calculated values as a tuple
        print("The best depth is " + str(best_depth) + " and the best score is " + str(best_score) + ". ")
        return (best_depth, best_score)
    # models are either Support Vector Machines or Logistic Regression (multinomial)

    # largest max gamma or C
    g = 1
    # initialize scores to the number of iterations
    scores = np.zeros(g*10)
    # initialize iteration ID to avoid confusion with d
    index = 0

    # for the Support Vector Machines model
    if model == "svm":
        # iterating through gamma values ranging from 0.1 to 1 (d/10)
        for d in range(1, g*10 + 1): #do we need 1?
            # create a svm model using the current gamma
            SVM = svm.SVC(gamma = d/10)
            # assign cross validation score using this model
            scores[index] = cross_val_score(SVM, X_train, y_train, cv = 5).mean()
            # if the current score is greater than the current best_score
            if scores[index] > best_score:
                # assign best_gamma as the current gamma (d/10)
                best_gamma = d/10
                # assign best_score as the current score
                best_score = scores[index]
            # add 1 to current index value for next iteration
            index += 1
        # display results and return calculated values as a tuple
        print("The best gamma is " + str(best_gamma) + " and the best score is " + str(best_score) + ". ")
        return (best_gamma, best_score)
    # for the Logistic Regression (multinomial) model
    elif model == "LogisticRegression":
        # iterating through C values ranging from 0.1 to 1 (d/10)
        for d in range(1, g*10 + 1): #do not need 1
            # create a MLR model using the current C
            MLR = LogisticRegression(multi_class = "multinomial", solver = "lbfgs", max_iter = 1000, C = d/10)
            # assign cross validation score using this model
            scores[index] = cross_val_score(MLR, X_train, y_train, cv = 5).mean()
            # if the current score is greater than the current best_score
            if scores[index] > best_score:
                # assign best_C as the current C (d/10)
                best_C = d/10
                # assign best_score as the current score
                best_score = scores[index]
            # add 1 to current index value for next iteration
            index += 1
        # display results and return calculated values as a tuple
        print("The best C is " + str(best_C) + " and the best score is " + str(best_score) + ". ")
        return (best_C, best_score)
    else:
        # if the model argument is not any of the three models, raise error
        raise ValueError("""
                         Invalid parameter name. 'param' must be either 'max_depth' (RandomForestClassifier),
                         or 'gamma' (svm), or 'C' (LogisticRegression)
                         """)
```

In [36]:

```
# a function to plot confusion matrices
def conf_matrix(c, X, y):
    """
    A function that displays a plot of a confusion matrix for a certain model fit on a dataset.
    A confusion matrix is a 2D array that compares predicted category labels to actual labels.
    This matrix using seaborns also has an additional row and column, referring to all the data.
    A number is displayed in each box signifying the percent of predicted that matched the
    actual for that specific combination.

    parameter 1, c: the model that is going to be fit to the dataset
    parameter 2, X: the predictor data we are going to use to fit the model
    parameter 3, y: the target data we are going to use to the fit the model???
    """
    # fitting the model, generating predictions using our variables, and then getting the confusion
    # matrix
    confusion_matrix = pd.crosstab(y, c.fit(X, y).predict(X[['Culmen Length (mm)',
                                                              'Culmen Depth (mm)',
                                                              'Sex']]),
                                   rownames = ['Actual'],
                                   colnames = ['Predicted'],
                                   margins = True)
    # creating the confusion matrix plot using the array from above
    sns.heatmap(confusion_matrix/np.sum(confusion_matrix),
                annot = True,
                fmt = '.1%',
                xticklabels = ['Adelie','Chinstrap','Gentoo','All'],
                yticklabels = ['Adelie','Chinstrap','Gentoo','All'])

    # displaying the confusion matrix plot
    plt.show()
```

In [37]:
```
# a function to plot decision regions

def plot_regions(c, X, y):
    """
    A function that plots decision regions for a certain model and the data specified.
    Decision regions are the parts of data space that the model assigns to each label.

    parameter 1, c: the model that is going to be fit to the dataset
    parameter 2, X: the predictor data we are going to use to fit the model
    parameter 3, y: the target data we are going to use to the fit the model???

    """
    # fitting the model to the dataset
    c.fit(X.to_numpy(), y)

    # our x-axis variable
    x0 = X["Culmen Length (mm)"]
    # y-axis variable
    x1 = X["Culmen Depth (mm)"]

    # creating the variables that allow us to store the x and y axis variables as coordinate vectors
    grid_x = np.linspace(x0.min(), x0.max(), 501)
    grid_y = np.linspace(x1.min(), x1.max(), 501)

    # using np.meshgrid to transform grid_x and grid_y from coordinate vectors to coordinate matrices
    xx, yy = np.meshgrid(grid_x, grid_y)
    # using .ravel() to transform xx and yy into contiguous flattened arrays
    XX = xx.ravel()
    YY = yy.ravel()

    # creating dicts so each color corresponds to a certain species
    color_dict = {0: "blue", 1: "green", 2: "red"}
```

```python
species_dict = {0: "Adelie", 1: "Chinstrap", 2: "Gentoo"}

X = X.values

# creating two subplots, one for female and one for male
fig, ax = plt.subplots(1, 2, sharey = True)

# making a separate plot for each of the two sexes, which contains another for loop that plots each species'
# points on the correct plot according to Sex
for Sex in range(2):
    # creates a list of the names of the sexes to be used later for labeling purposes
    Sex_list = ["Female", "Male"]
    # creating an array of equal length to XX and YY to use Sex as a predictive variable
    ZZ = Sex*(np.ones(len(XX)))
    # using the model to make predictions and reshaping the array to match the other arrays
    p = c.predict(np.c_[XX, YY, ZZ])
    p = p.reshape(xx.shape)
    # graphs the predictions based on the model, which produces three regions on the plot of different colors
    ax[Sex].contourf(xx, yy, p, cmap = "jet", alpha = .2)
    # extracting the data for the Sex being plotted during that iteration of the for loop
    Xi = X[X[:,2] == Sex]
    yi = y[X[:,2] == Sex]

    # looping through each species and plots the scatter points of that species
    for Species in range(3):
        ax[Sex].scatter(Xi[yi == Species][:,0],
                        Xi[yi == Species][:,1],
                        c = color_dict[Species],
                        label = species_dict[Species])

    # adding the appropriate titles to each of the two plots and x axis label
    ax[Sex].set(xlabel = "Culmen Length (mm)", title = Sex_list[Sex])

# setting the axis labels and adding a legend
# only adding the y label to the plot on the left to reduce redundancy
# only adding the legend to the plot on the right to reduce redundancy
ax[0].set(xlabel = "Culmen Length (mm)", ylabel = "Culmen Depth (mm)", label = y)
ax[1].legend()

# making the plot look nicer
plt.tight_layout()
```

Model 1: Random Forests

Our first model is Random Forests. The Random Forest model works by taking a dataset and creating a large number of uncorrelated random decision trees to make a prediction. The model then aggregates the results from the trees to make more accurate predictions. Although we have already performed feature selection, the Random Forest model would generally include the feature selection process, making it suitable for datasets with a large number of predictive features. We expect to disregard overfitting and outlier effects as significant problems with the Random Forest model based on the model's use of bagging. Our Random Forest model will create trees that include `Culmen Length (mm)`, `Culmen Depth (mm)`, and `Sex` as predictive variables.

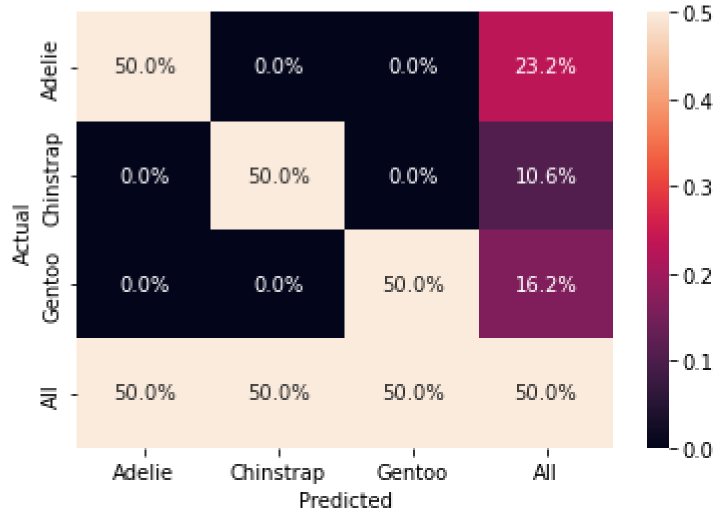
In [38]:
```
# getting the best_depth and score for our random forest model that is returned by our function
from sklearn.ensemble import RandomForestClassifier

# get best parameters
best_param, best_score = get_best_parameters(X_train, y_train, "RandomForestClassifier")
```
```
The best depth is 4 and the best score is 0.9866666666666667.
```

In [39]:
```
# creating Random Forest model using the best parameter value
best_RF = RandomForestClassifier(random_state = 42, max_depth = best_param)
```

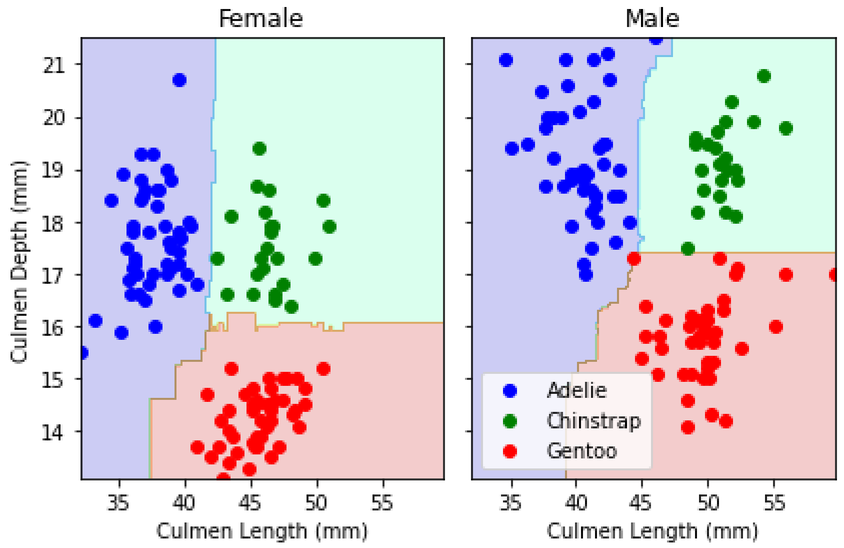
In [40]:
```
# evaluating model with test data
best_RF.fit(X_train, y_train).score(X_test[['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex']], y_test)
```
Out[40]:
```
0.9595959595959596
```

Below is a visualization of the confusion matrix for `best_RF`, using `seaborn`. It compares the actual species to the predicted species. It incldues a 4th row and column that refers to all of the penguins in the data.

In [41]:
```
# producing the confusion matrix plot for our best random forests model
conf_matrix(best_RF, X_test, y_test)
```



In [42]:
```
# plotting the decision regions using our random forests model and the training data
plot_regions(best_RF, X_train, y_train)
```



Mistakes Made by the Random Forest Model

The Random Forest model does not appear to make any mistakes based on the decision regions plot and the confusion matrix. This is likely because we are working with a relatively small data set with a large number of estimators in the model (100).

Model 2: Support Vector Machines

Our second model is Support Vector Machines. The Support Vector Machine model works by determining the best hyperplane to make predictions in the n-dimensional space, where n equals the number of predictive features. The model determines the best hyperplane both through how well it is able to separate the training data to produce the correct predictions and the maximum margin between the two closest data points across predictions. This maximized margin creates a buffer, where unseen data is less likely to be misclassified by the model should it fall slightly outside of the training data.

Our Support Vector Machines Model will create two 2D planes, one for female and one for male, using the predictive variables of `Culmen Length (mm)` and `Culmen Depth (mm)`. Overfitting is a potential problem with the Support Vector Machines model, but its effects are mitigated by determining optimal values of gamma (how closely the hyperplane tries to match the training data).

In [43]:
```
# import svm
from sklearn import svm

# get best parameters
best_param, best_score = get_best_parameters(X_train, y_train, "svm")
```

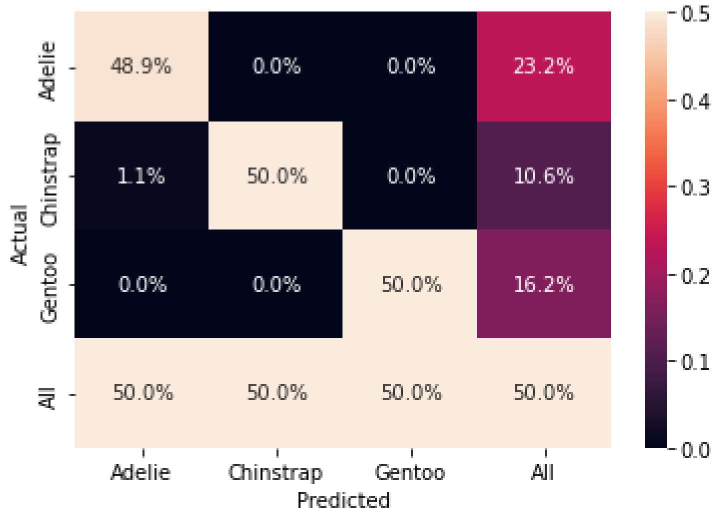
```
The best gamma is 0.1 and the best score is 0.9777777777777779.
```

In [44]:
```
# creating a new version of the SVM model that is refit using our best_gamma
best_SVM = svm.SVC(gamma = best_param)
```

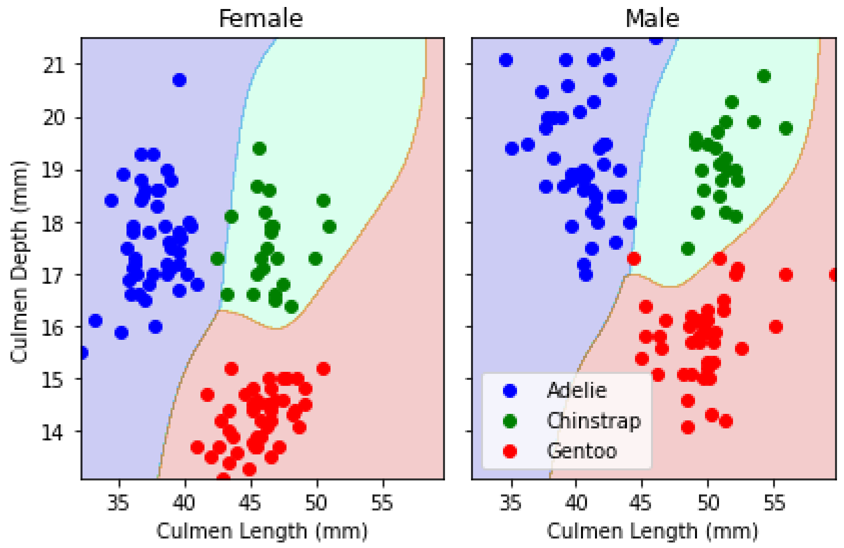
In [45]:
```
# evaluating model with test data
best_SVM.fit(X_train, y_train).score(X_test[['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex']], y_test)
```

Out[45]:
```
0.9494949494949495
```

In [46]:
```
# producing the confusion matrix plot for our best SVM model
conf_matrix(best_SVM, X_test, y_test)
```



In [47]:
```
# plotting the decision regions using our best SVM model and the training data
plot_regions(best_SVM, X_train, y_train)
```



Mistakes Made by the Support Vector Machines Model

While the Support Vector Machines model appears to generate reasonable decision regions, one of its shortcomings is that it noticeably overrepresents the decision regions for Gentoo penguins, as shown in both the male and female subplots. Since on average, the Gentoo penguins have shorter culmen depths than the other two species, we can assume that when plotted, a new, unseen penguin with a very large culmen length and culmen depth is better classified as a Chinstrap. According to the decision regions above, however, the Support Vector Machine will classify such a penguin as a Gentoo, which is likely false—especially with extraordinarily high culmen depth values.

Moreover, a female Chinstrap was incorrectly predicted to be an Adelie and a male Gentoo was incorrectly predicted to be a Chinstrap, but these discrepancies could be explained by the fact that the boundaries between species groups were very close, and the Support Vector Machines model is generally better suited for datasets that are fairly sparsed.

Model 3: Multinomial Logistic Regression

Our third and last model is multinomial logistic regression. Logistic regression is a model that is only limited to two-class classification problems. In these models, the target data is modeled by using a binomial probability distribution function. The model overall predicts the probability that an example belongs to class 1, where class labels are mapped to 1 for the positive class/outcome and 0 for the negative class/outcome.

Multinomial logistic regression is similar to logistic regression but with modifications that allow it to support multi-class classification problems. One of these modifications is changing the loss function used to train the model to cross-entropy loss and a change to the output from a single probability value (like in logistic regression) to one probability for each class label.

In [48]:
```
# import Logistic Regression
from sklearn.linear_model import LogisticRegression

# get best parameters
best_param, best_score = get_best_parameters(X_train, y_train, "LogisticRegression")
```

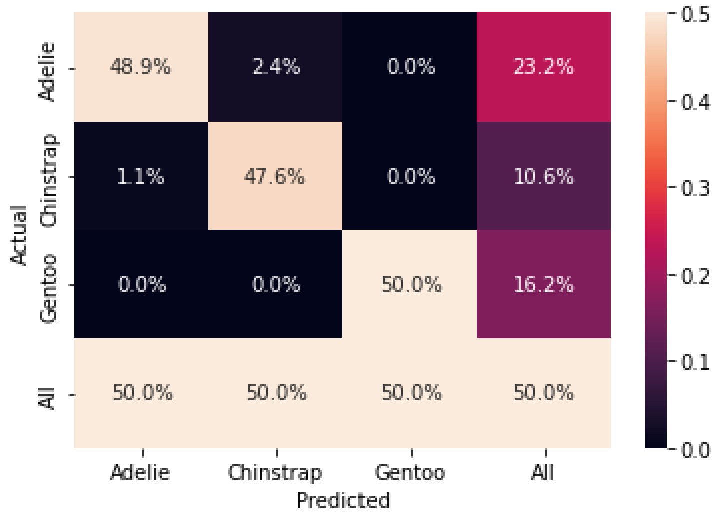
```
The best C is 0.8 and the best score is 0.9866666666666667.
```

In [49]:
```
# creating a new version of the MLR model that is refit using our best_gamma
best_MLR = LogisticRegression(multi_class = "multinomial", solver = "lbfgs", max_iter = 1000, C = best_param)
```

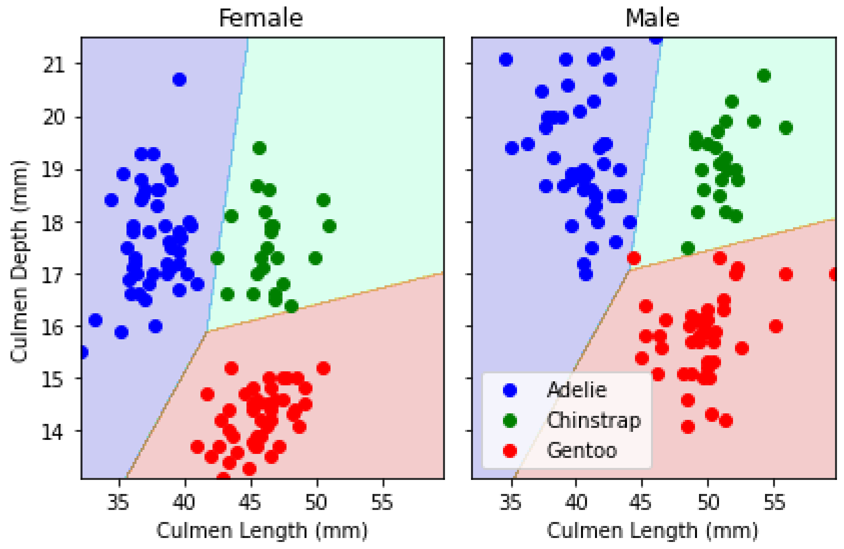
In [50]:
```
# evaluating model with test data
best_MLR.fit(X_train, y_train).score(X_test[['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex']], y_test)
```

Out[50]:
```
0.9797979797979798
```

In [51]:
```
# producing the confusion matrix plot for our best MLR model
conf_matrix(best_MLR, X_test, y_test)
```



In [52]:
```
# plotting the decision regions using our best MLR model and the training data
plot_regions(best_MLR, X_train, y_train)
```



Mistakes Made by the Multinomial Logistic Regression Model

From our confusion matrix for this model, we can see that the model makes some errors. A few Chinstrap penguins were incorrectly predicted to be Adelie penguins, and a few Adelie penguins were incorrectly predicted to be Chinstrap penguins. Upon further analysis by looking at our decision regions for this model, we can see that the mistakes lie on the boundaries of the decision regions. A possible reason for the model making this mistake is that the model runs under the assumption that the data does not have outliers. When the model is used with real data, there are outliers that get misclassified.

Discussion

As a reminder, with our systematic feature selection, the best columns were shown to be `['Culmen Length (mm)', 'Culmen Depth (mm)', 'Sex']`, with a testing score of `0.9797979797979798`.

Below are their scores on the test data:

- Random Forest : `0.9595959595959596`
- Support Vector Machines : `0.9494949494949495`
- Multinomial Logistic Regression: `0.9797979797979798`

After analyzing our models and their performances, we recommend using the Random Forests model fit with parameter `max_depth = 4`, and using features `Culmen Length (mm)`, `Culmen Depth (mm)`, and `Sex`. Through our feature selection method, and guided by our exploratory analysis, we chose to use these three variables since this was the combination that produced the highest testing score. Even though our Random Forest and Multinomial Logistic Regression model had identical scores, the Random Forest model generated the least errors, which is why we think that using the Random Forest model and the previously specified features produces the most accurate model. Random Forests was likely the best model because we were able to use a high number of estimators for a relatively small data set, and this did not take too long to run because we had pre-selected our features. Moreover, the Random Forest model accounts for outliers the most of the three models through the data aggregation that occurs when the decision trees are grouped together. The Multinomial Logistic Regression model assumes that there are no outliers in the data, while the Support Vector Machines model tries to maximize the margin between data boundaries to account for outliers. Since our data is both close together in terms of data boundaries, and has outliers, it makes intuitive sense that Random Forest is the model best equipped to deal with these conditions.

We do not recommend using the Support Vector Machines model because it had the lowest score and produced errors. The Support Vector Machines model is a better fit for data sets that are small, but have a high number of features because of its ability to work with a large amount of features with minimal work from the user.

Improvements: The Random Forest model with the specified parameters and features could be improved through the following recommendations:

- Determine the best parameters for other parameters besides depth
- Perform cross validations with more n-folds (increase cv in cross_val_score)
- Increase the number of estimators from 100 to 1,000+
- Add more different features into the data set - Random Forest is able to work with a large number of features, so it would improve the model to add more features that could potentially have higher predictive value than the combination of three that we selected during the Feature Selection process