PredictingPenguinSpecies_ML

October 14, 2025

1 Palmer Penguins: Predicting Penguin Species via Machine Learning

1.0.1 Introduction

Understanding and classifying the diverse species of penguins in the Antarctic is an essential ecological task. Penguins play a crucial role in their ecosystems, and identifying their species contributes to research in biodiversity, habitat preservation, and ecological balance. However, traditional classification methods often rely on extensive measurements and expert biological knowledge, which can be challenging to scale when faced with large populations.

This project focuses on leveraging machine learning techniques to streamline penguin species classification. By utilizing the Palmer Penguins dataset, which provides detailed attributes such as culmen length, culmen depth, flipper length, body mass, sex, and the island where the penguin was encountered, we aim to identify the most predictive features for species classification. The dataset, collected by Dr. Kristen Gorman and the Palmer Station LTER, is an open resource designed to facilitate ecological research.

Our goal is to develop machine learning models that can accurately classify penguin species using a minimal set of features—one qualitative and two quantitative attributes. This approach ensures efficiency while maintaining high predictive power. Through rigorous feature selection, exploratory analysis, and the application of at least three machine learning models, we aim to achieve both simplicity and accuracy in classification. Additionally, we will create decision region visualizations to provide intuitive insights into the models' predictions and explore potential limitations in their accuracy.

By identifying key features and deploying effective models, this project contributes to a more scalable and automated method of penguin classification, which is crucial for ecological monitoring and conservation efforts.

There're three kind of species:

1.1 Data Import and Cleaning

Step 1: import necessary libraries

[1]:
```
# libraries
import pandas as pd
import numpy as np
import random
import urllib

# visualization
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap, BoundaryNorm
import seaborn as sns

# data preprocessing
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score,␣
↪GridSearchCV, RandomizedSearchCV

# machine learning models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import tree

# evaluation metrics
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    ConfusionMatrixDisplay,
```

```
    accuracy_score
)
```

Step 2: load the data

[2]:
```
np.random.seed(1234)
# From github url, import Palmer Penguins dataset as a pd.dataframe
url = "https://philchodrow.github.io/PIC16A/datasets/palmer_penguins.csv"
penguins = pd.read_csv(url)

# print the first 5 rows of the dataframe
penguins.head(n=5)
```

[2]:
```
  studyName  Sample Number                                Species  Region  \
0   PAL0708              1  Adelie Penguin (Pygoscelis adeliae)  Anvers
1   PAL0708              2  Adelie Penguin (Pygoscelis adeliae)  Anvers
2   PAL0708              3  Adelie Penguin (Pygoscelis adeliae)  Anvers
3   PAL0708              4  Adelie Penguin (Pygoscelis adeliae)  Anvers
4   PAL0708              5  Adelie Penguin (Pygoscelis adeliae)  Anvers

      Island                Stage Individual ID Clutch Completion  Date Egg  \
0  Torgersen  Adult, 1 Egg Stage          N1A1                Yes  11/11/07
1  Torgersen  Adult, 1 Egg Stage          N1A2                Yes  11/11/07
2  Torgersen  Adult, 1 Egg Stage          N2A1                Yes  11/16/07
3  Torgersen  Adult, 1 Egg Stage          N2A2                Yes  11/16/07
4  Torgersen  Adult, 1 Egg Stage          N3A1                Yes  11/16/07

   Culmen Length (mm)  Culmen Depth (mm)  Flipper Length (mm)  Body Mass (g)  \
0                39.1               18.7                181.0         3750.0
1                39.5               17.4                186.0         3800.0
2                40.3               18.0                195.0         3250.0
3                 NaN                NaN                  NaN            NaN
4                36.7               19.3                193.0         3450.0

      Sex  Delta 15 N (o/oo)  Delta 13 C (o/oo)  \
0    MALE                NaN                NaN
1  FEMALE            8.94956          -24.69454
2  FEMALE            8.36821          -25.33302
3     NaN                NaN                NaN
4  FEMALE            8.76651          -25.32426

                         Comments
0  Not enough blood for isotopes.
1                             NaN
2                             NaN
3             Adult not sampled.
4                             NaN
```

Through the table, we noticed that the names of the species are too long. For convenience, we decided to use shorter versions.

[3]:
```
penguins["Species"] = penguins["Species"].str.split().str[0]
```

Step 3: preliminary examination of the dataset's structure.

Here, we explore the structure of the Palmer Penguins dataset by examining its size and the data types of each column. This step allows us to become familiar with the dataset, identify its characteristics, and pinpoint any potential issues before proceeding with further analysis.

[4]:
```
shape = penguins.shape
# find the shape of the penguins dataframe
print("Shape of the penguins dataframe is: ", shape, "\n")

datatypes = penguins.dtypes
# find the datatypes of each column within the dataframe penguins
print("The datatype of each of the columns of penguins dataframe is: \n",␣
 ↪datatypes)
```

```
Shape of the penguins dataframe is:  (344, 17)

The datatype of each of the columns of penguins dataframe is:
 studyName             object
Sample Number           int64
Species                object
Region                 object
Island                 object
Stage                  object
Individual ID          object
Clutch Completion      object
Date Egg               object
Culmen Length (mm)    float64
Culmen Depth (mm)     float64
Flipper Length (mm)   float64
Body Mass (g)         float64
Sex                    object
Delta 15 N (o/oo)     float64
Delta 13 C (o/oo)     float64
Comments               object
dtype: object
```

[5]:
```
penguins["Region"].unique() , penguins["Stage"].unique()
```

[5]:
```
(array(['Anvers'], dtype=object), array(['Adult, 1 Egg Stage'], dtype=object))
```

All the penguins in the dataset come from the same region, and all share the stage "Adult, 1 Egg Stage," which means these two variables lack variability and do not offer distinguishing power for analysis. Additionally, the Individual ID is likely a unique identifier for each penguin and does not represent any biological or behavioral significance, making it unsuitable for analysis. Including it

could introduce noise or potentially lead to overfitting in predictive models. The Clutch Completion column, indicating whether the clutch (set of eggs) was completed (e.g., "Yes"), might hold relevance in ecological contexts but is unlikely to contribute meaningfully to species classification or other predictive analyses unless the focus is specifically on reproductive success. Similarly, the Egg Date may not provide meaningful insights for species classification or biological analysis unless it aligns with a seasonal or temporal factor. If the date is constant or unrelated to the outcomes of interest, it is not a priority feature. Therefore, we will categorize the remaining columns into quantitative and qualitative variables for further exploration.

[6]:
```
quantitative_cols = ["Culmen Length (mm)", "Culmen Depth (mm)", "Flipper Length␣
 ↪(mm)", "Body Mass (g)", "Delta 15 N (o/oo)", "Delta 13 C (o/oo)"]
qualitative_cols = ["Species", "Island", "Sex"]
```

Step 4: data check

Check for the number of NAN/missing values within the dataset that needs to be removed during the cleaning process.

[7]:
```
# extract the column names from the dataframe as a list
p_columns = list(penguins.columns)

# initialize an empty dictionary to store the count of NaN values for each␣
 ↪column
nan_dict = {}

# iterate through each column in the dataframe
for i in range(len(p_columns)):
    # check for NaN values in the current column
    is_null = penguins[p_columns[i]].isna()
    # update the dictionary with the column name as key and count of NaN values␣
 ↪as value
    nan_dict.update({p_columns[i]: is_null.sum()})

# iterate through the dictionary to print the NaN counts for each column
for key, val in nan_dict.items():
    print("There are ", val, " NAN values within column [", key, "]")
```

```
There are  0  NAN values within column [ studyName ]
There are  0  NAN values within column [ Sample Number ]
There are  0  NAN values within column [ Species ]
There are  0  NAN values within column [ Region ]
There are  0  NAN values within column [ Island ]
There are  0  NAN values within column [ Stage ]
There are  0  NAN values within column [ Individual ID ]
There are  0  NAN values within column [ Clutch Completion ]
There are  0  NAN values within column [ Date Egg ]
There are  2  NAN values within column [ Culmen Length (mm) ]
There are  2  NAN values within column [ Culmen Depth (mm) ]
There are  2  NAN values within column [ Flipper Length (mm) ]
```

```
There are  2  NAN values within column [ Body Mass (g) ]
There are  10  NAN values within column [ Sex ]
There are  14  NAN values within column [ Delta 15 N (o/oo) ]
There are  13  NAN values within column [ Delta 13 C (o/oo) ]
There are  318  NAN values within column [ Comments ]
```

Step 5: define function used to get and clean data

[8]:
```python
def obtain_clean(dataframe):
    """
    clean the input dataframe by removing rows with NaN values and
    optionally filtering out rows where the 'Sex' column contains a '.'␣
 ↪character

    Parameters
    ----------
    dataframe: pandas.DataFrame
        the input dataframe to be cleaned

    Return
    ----------
    pandas.DataFrame
        a cleaned copy of the dataframe with NaN values removed
        and rows filtered where 'Sex' column equals '.'
    """
    # create a copy of the dataframe to avoid modifying the original
    data = dataframe.copy()

    # drop all rows with NaN values
    data = data.dropna()

    # check if the 'Sex' column exists in the dataframe
    if "Sex" in list(data.columns):
        # drop rows where the 'Sex' column has the value '.'
        data = data.drop(data[data["Sex"] == "."].index)

    # return the cleaned dataframe
    return data
```

Step 6: define additional functions

We will define a function needed to encode qualitative data into numerical values, and a function to split the data up into training and testing sets. Dividing the dataset ensures that the models are trained on one subset and tested on another, preventing overfitting.

[9]:
```python
Species = {'Adelie': 0, 'Chinstrap': 1, 'Gentoo': 2}
Islands = {'Biscoe': 0, 'Dream': 1, 'Torgersen': 2}
Sex = {'FEMALE': 0, 'MALE': 1}
```

[10]:
```
def convert_qual_to_int(df):
    """
    convert qualitative columns (e.g., 'Island', 'Sex', 'Species') in the␣
 ↪dataframe
    to integers using predefined mappings

    Parameters
    ----------
    df: pandas.DataFrame
        the dataframe containing the qualitative columns to be mapped

    Note
    ----------
    This function modifies the input dataframe directly
    """
    # map qualitative values in the 'Island' column to integers using the␣
 ↪'Islands' mapping
    df['Island'] = df['Island'].map(Islands)
    # map qualitative values in the 'Sex' column to integers using the 'Sex'␣
 ↪mapping
    df['Sex'] = df["Sex"].map(Sex)
    # map qualitative values in the 'Species' column to integers using the␣
 ↪'Species' mapping
    df['Species'] = df["Species"].map(Species)

def data_splitting(df, target_column='Species'):
    """
    split the dataframe into features (X) and target (y) after converting
    qualitative columns to integers

    Parameters
    ----------
    df: pandas.DataFrame
        the input dataframe to be split into features and target

    target_column: str, optional
        the column name to be used as the target variable
        defaults to 'Species'

    Return
    ----------
    tuple
        X: pandas.DataFrame
            dataframe containing all columns except the target column

        y: pandas.DataFrame
            dataframe containing only the target column
```

```
    """
    # create a copy of the dataframe to avoid modifying the original
    dataframe = df.copy()

    # convert qualitative columns to integers
    convert_qual_to_int(dataframe)

    # separate features (X) by dropping the target column
    X = dataframe.drop(target_column, axis=1)

    # extract the target column (y)
    y = dataframe[[target_column]]

    # return the features and target as a tuple
    return X, y
```

The dataset is first split into training and test sets, with 80% used for training and 20% for testing, ensuring consistency with a random state of 123. Then, the cleaned data is divided into features (X) and labels (y) for both the training and test sets using the data_splitting function.

[11]:
```
# select relevant qualitative and quantitative columns from the penguins dataset
penguins = penguins[qualitative_cols + quantitative_cols]

# split the dataset into training and test sets (80% training, 20% testing)
train, test = train_test_split(penguins, test_size = .2, random_state = 123)

# clean the entire dataset, training dataset, and test dataset separately
cleaned_penguin = obtain_clean(penguins)
cleaned_train = obtain_clean(train)
cleaned_test = obtain_clean(test)

# split the cleaned training and test datasets into features (X) and labels (y)
X_train, y_train = data_splitting(cleaned_train)
X_test, y_test = data_splitting(cleaned_test)

# split the entire cleaned dataset into features (X) and labels (y) for further␣
 ↪use
X, y = data_splitting(cleaned_penguin)
```

[12]:
```
print(X_train.count() + X_test.count())
```

```
Island                324
Sex                   324
Culmen Length (mm)    324
Culmen Depth (mm)     324
Flipper Length (mm)   324
Body Mass (g)         324
Delta 15 N (o/oo)     324
```

```
Delta 13 C (o/oo)      324
dtype: int64
```

This output confirms that all feature columns in the training (X_train) and test (X_test) datasets have consistent counts of 324. This indicates that no data was lost during the data preprocessing and splitting process. Each feature, such as "Island," "Sex," and the various measurements (e.g., "Culmen Length," "Body Mass"), is complete and ready for use in model training and evaluation.

1.2 Exploratory Analysis

In this part, we conducted exploratory data analysis (EDA) to gain an initial understanding of the dataset's structure and the relationships within the data. We began by summarizing the dataset, grouping the penguins by species and sex, and calculating the average values of key numerical features such as culmen length, culmen depth, flipper length, body mass, and isotopic measurements (Delta 15N and Delta 13C). Next, we grouped the penguins by island to analyze the distribution of species across the islands. To enhance our analysis and facilitate feature selection, we visualized the data using violin plots, scatterplots, boxplots, and histograms.

1.2.1 Table 1: Penguin Statistics Summary by Species and Sex

[13]:
```
# take the average of each numerical measurement by Species and by Sex.
stat_penguin = cleaned_penguin.drop(columns = "Island")
stat_penguin.groupby(["Species", "Sex"]).mean()
```

[13]:

Species	Sex	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Delta 15 N (o/oo)	Delta 13 C (o/oo)
Adelie	FEMALE	37.212676	17.619718	187.915493	3366.549296	8.793275	-25.794158
	MALE	40.427941	19.057353	192.838235	4053.676471	8.928437	-25.833813
Chinstrap	FEMALE	46.573529	17.588235	191.735294	3527.205882	9.250962	-24.565405
	MALE	51.069697	19.245455	199.727273	3938.636364	9.464535	-24.550104
Gentoo	FEMALE	45.563793	14.237931	212.706897	4679.741379	8.193405	-26.197205
	MALE	49.510000	15.725000	221.533333	5488.750000	8.303429	-26.170608

Notably, males across all species tend to exhibit larger physical features than females, highlighting a clear pattern of sexual dimorphism. For instance, male Gentoo penguins have significantly greater flipper lengths and body masses compared to females, showcasing their robust build. Similarly, the Chinstrap species has the largest culmen length averages, distinguishing it from the Adelie and Gentoo species. Beyond physical attributes, the isotopic measurements (Delta 15N and Delta 13C) show minimal variation between sexes within the same species but differ slightly across species, potentially reflecting differences in diet or habitat. The Gentoo species, in particular, stands out

with the highest body mass and flipper length, underscoring its physical dominance among the species.

1.2.2 Table 2: Penguin Count by Island

[43]:
```
# count the number of penguins on each island based on a specific column
# cleaned_penguin.groupby(["Island", "Species"]).size()
cleaned_penguin.groupby(["Island", "Species"]).size().reset_index(name="Penguin␣
↪Count")
```

[43]:

	Island	Species	Penguin Count
0	Biscoe	Adelie	44
1	Biscoe	Gentoo	118
2	Dream	Adelie	52
3	Dream	Chinstrap	67
4	Torgersen	Adelie	43

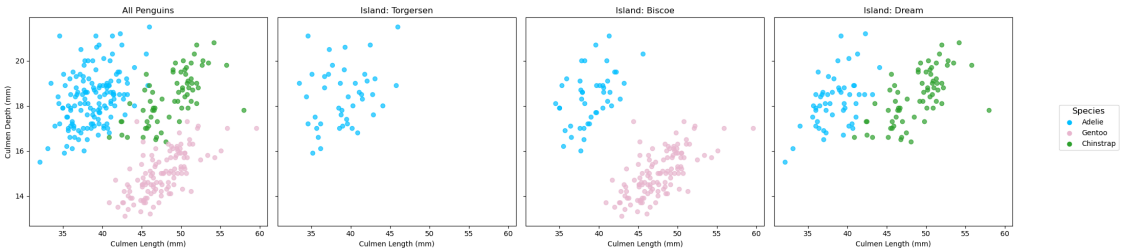
This table provides an overview of penguin populations categorized by species and island. It shows that Gentoo penguins dominate the population on Biscoe Island, with a total count of 124, followed by 44 Adelie penguins. Chinstrap penguins are exclusively found on Dream Island, where they represent a population of 68, alongside 56 Adelie penguins. Torgersen Island is exclusively home to 52 Adelie penguins. The distribution highlights that Gentoo penguins are concentrated on Biscoe Island, Chinstrap penguins are restricted to Dream Island, and Adelie penguins are the only species spread across all three islands, although their numbers vary.

1.2.3 Figure 1: Scatter Plot of Culmen Length vs. Culmen Depth by Species

The scatter plot visualizes the relationship between Culmen Length (mm) and Culmen Depth (mm) for different penguin species across various islands. Each subplot represents data either aggregated for all islands or filtered for individual islands.

[15]:
```
# extract relevant columns for scatter plots
penguins_scatter = cleaned_penguin[["Island", "Species", "Culmen Length (mm)",␣
↪"Culmen Depth (mm)"]]

# get unique islands for filtering plots
islands = penguins_scatter["Island"].unique()

# define a color map for species
color_map = {
    "Adelie": "#00BFFF",  # blue for adelie
    "Gentoo": "#e8b5ce",  # pink for gentoo
    "Chinstrap": "#2ca02c",  # green for chinstrap
}

# create subplots with shared x and y axes
fig, axes = plt.subplots(1, 4, figsize=(20, 5), sharey=True, sharex=True)
```

```
# plot all penguins on the first subplot
axes[0].scatter(
    penguins_scatter["Culmen Length (mm)"],
    penguins_scatter["Culmen Depth (mm)"],
    c=penguins_scatter["Species"].map(color_map),  # map species to colors
    alpha=0.7,  # set transparency for better visualization
)
axes[0].set_title("All Penguins")  # set title for the first subplot
axes[0].set_xlabel("Culmen Length (mm)")  # set x-axis label for the first␣
 ↪subplot

# add legend entries for species in the first plot
for species, color in color_map.items():
    axes[0].scatter([], [], color=color, label=species)

# iterate through each island and create a scatter plot for each
for i, subisland in enumerate(islands):
    ax = axes[i + 1]
    data = penguins_scatter[penguins_scatter["Island"] == subisland]
    ax.scatter(
        data["Culmen Length (mm)"],
        data["Culmen Depth (mm)"],
        c=data["Species"].map(color_map),  # map species to colors
        alpha=0.7,  # set transparency for better visualization
    )
    ax.set_title(f"Island: {subisland}")  # set title based on island name
    ax.set_xlabel("Culmen Length (mm)")  # set x-axis label for each island plot

# set the shared y-axis label
axes[0].set_ylabel("Culmen Depth (mm)")

# add a legend for species on the right of the figure
fig.legend(
    title="Species",
    loc="right",
    bbox_to_anchor=(1.1, 0.5),
    fontsize=10,
    title_fontsize=12,
)

# adjust layout for better spacing
plt.tight_layout()
plt.show()
```



The scatter plot effectively illustrates the distribution of penguins on each island, corresponding to the information presented in Table 2. However, this visualization is not ideal for detailed data analysis. While it highlights the distinct characteristics of Gentoo penguins (pink), which have longer culmen lengths and smaller depths, it also reveals significant overlap between Adelie (blue) and Chinstrap (green) penguins, particularly in the mid-range culmen lengths.

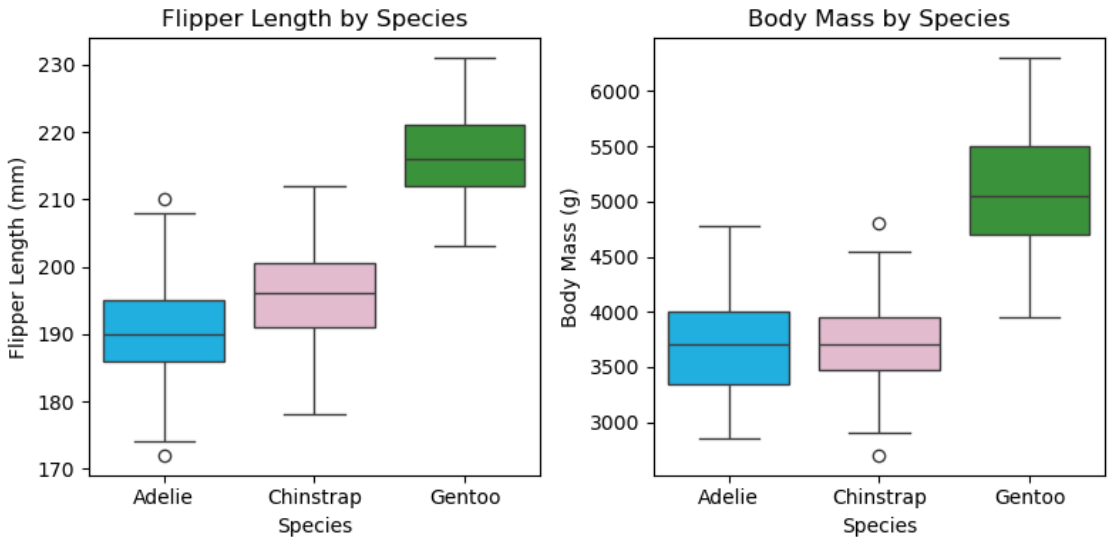
Let's explore another type of plot to see if we can gain clearer insights!

1.2.4 Figure 2: Box Plot of Flipper Length and Body Mass by Species

Then let's use a boxplot to see whether flipper length and body mass show clear distinctions between species.

[16]:
```
# selecting only the relevant columns from cleaned_penguin for plotting
penguin_boxplot = cleaned_penguin[["Species", "Flipper Length (mm)", "Body Mass␣
↪(g)"]]

# creating a figure with two subplots in a single row and setting figure size
fig, axes = plt.subplots(1, 2, figsize = (8, 4))

# creating a boxplot for flipper length grouped by species
sns.boxplot(
            data = penguin_boxplot, # specifying data source
            x = "Species", # x-axis variable (species)
            y = "Flipper Length (mm)", # y-axis variable (flipper length)
            ax = axes[0], # assigning plot to the first subplot
            palette = ["#00BFFF","#e8b5ce","#2ca02c"], # setting color palette
            hue = "Species" # adding species as a grouping hue
)
# setting the title, x-axis label, and y-axis label for the first subplot
axes[0].set_title("Flipper Length by Species")
axes[0].set_ylabel("Flipper Length (mm)")

# creating a boxplot for body mass grouped by species
sns.boxplot(
            data = penguin_boxplot, # specifying data source
            x = "Species", # x-axis variable (species)
            y = "Body Mass (g)", # y-axis variable (body mass)
```

```
            ax = axes[1], # assigning plot to the second subplot
            palette = ["#00BFFF","#e8b5ce","#2ca02c"], # setting color palette
            hue = "Species" # adding species as a grouping hue
)
# setting the title, x-axis label, and y-axis label for the second subplot
axes[1].set_title("Body Mass by Species")
axes[1].set_ylabel("Body Mass (g)")

# adjusting layout to prevent overlap of subplots
plt.tight_layout()

# displaying the plots
plt.show()
```



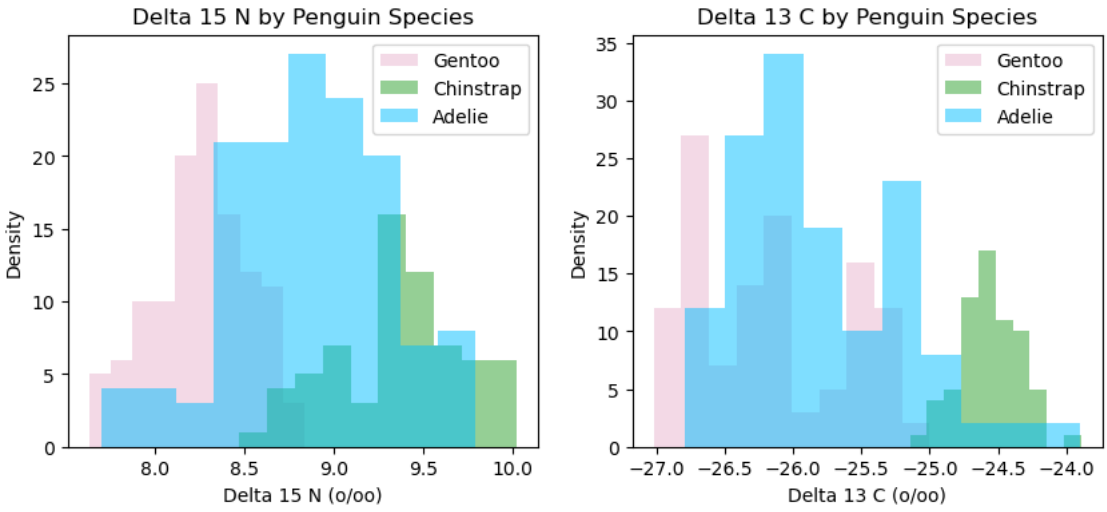
These two boxplots compare flipper length and body mass across three penguin species. In the left plot, Adelie penguins have the shortest median flipper length, around 190 mm, with low variability and a few outliers on the lower end. Chinstrap penguins show a slightly higher median flipper length of approximately 195–200 mm, with greater variability compared to Adelie penguins. Gentoo penguins, on the other hand, have the longest median flipper length, around 220 mm, with a narrow range and no apparent outliers. In the right plot, Adelie penguins have the lowest median body mass, about 3500 g, followed by Chinstrap penguins with a slightly higher median of 3750–4000 g, both showing moderate variability and a few lower outliers. Gentoo penguins exhibit the highest median body mass, approximately 5000 g, with more variability but no significant outliers. Overall, Gentoo penguins are distinct in both flipper length and body mass, being larger and more consistent in flipper length but more variable in body mass, while Adelie and Chinstrap penguins show overlapping characteristics that make them more similar to each other.

1.2.5 Figure 3: Histogram of Delta 15 N and Delta 13 C by Species

Then we would like to use a Histogram model to visualize the Delta 15 N and Delta 13 C data of each species.

[17]:
```
penguins_hist = cleaned_penguin[["Species", "Delta 15 N (o/oo)", "Delta 13 C (o/
 ↪oo)"]]

fig, ax = plt.subplots(1, 2, figsize = (10, 4))
ax[0].set(xlabel = "Delta 15 N (o/oo)",
       ylabel = "Density")
# create a figure, set the axis
species = set(penguins_hist['Species'])

# graph the species as a histogram
for s in species:
    subspecies = penguins_hist[penguins_hist['Species'] == s]
    ax[0].hist(subspecies['Delta 15 N (o/oo)'], color = color_map[s], label =␣
 ↪s, alpha = 0.5)
ax[0].set_title("Delta 15 N by Penguin Species")
ax[0].legend()

ax[1].set(xlabel = "Delta 13 C (o/oo)",
       ylabel = "Density")
# create a figure, set the axis
species = set(penguins['Species'])

# graph the species as a histogram
for s in species:
    subspecies = penguins_hist[penguins_hist['Species'] == s]
    ax[1].hist(subspecies['Delta 13 C (o/oo)'], color = color_map[s], label =␣
 ↪s, alpha = 0.5)

ax[1].set_title("Delta 13 C by Penguin Species")
ax[1].legend()
```

[17]:
```
<matplotlib.legend.Legend at 0x177d64320>
```



These histograms visualize the density distributions of isotopic measurements (Delta 15 N and Delta 13 C) the three species.

In the first plot ("Delta 15 N by Penguin Species"), Delta 15 N shows distinct patterns across species. Chinstrap penguins have higher values concentrated around 9.5–10, whereas Gentoo penguins are centered around 8–8.5. Adelie penguins display a broader distribution, overlapping both Chinstrap and Gentoo distributions, but their density peaks around 8.5–9.

In the second plot ("Delta 13 C by Penguin Species"), Delta 13 C values reveal another separation. Gentoo penguins have the most negative values, primarily between -27 and -26, while Adelie penguins dominate the middle range around -26 to -25. Chinstrap penguins have the least negative Delta 13 C values, clustering around -24.5 to -25.

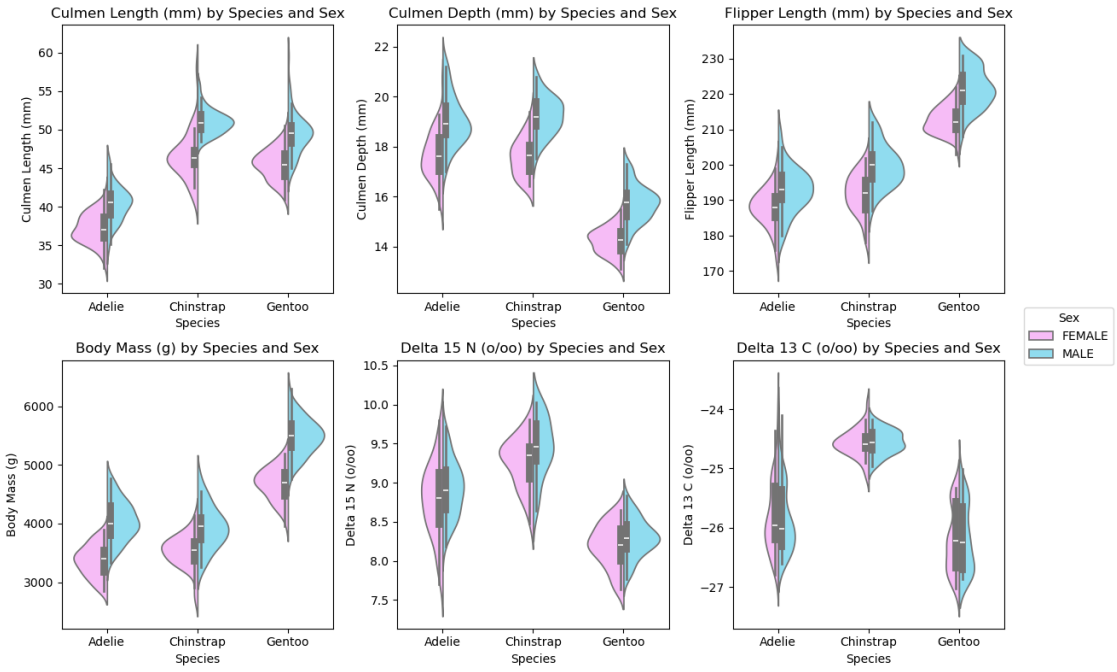
In summary, although Delta 15 N and Delta 13 C can exhibit trend differences within certain ranges, the overlap may make it difficult for classification models to achieve high accuracy relying solely on these two features.

1.2.6 Figure 4: Violin Plot of Quantitative Features Grouped by Species and Sex

Finally, to better visualize the relationship between numerical and categorical data, we utilized violin plots to aggregate and analyze the distributions effectively.

[18]:
```
# assign the cleaned penguin dataset
penguin_violinMap = cleaned_penguin

# create subplots based on the number of quantitative columns
fig, axes = plt.subplots(2, len(quantitative_cols) // 2, figsize=(4 *␣
 ↪(len(quantitative_cols) // 2), 8))
axes = axes.flatten()  # flatten axes for easier iteration

# initialize handles and labels for the legend
handles, labels = [], []
```

```
# iterate over quantitative columns to create violin plots
for i, var in enumerate(quantitative_cols):
    sns.violinplot(
        data = penguin_violinMap,  # use the cleaned dataset
        x = "Species",  # group by species
        y = var,  # quantitative variable for the y-axis
        hue = "Sex",  # split by sex
        palette = {"MALE": "#80e5ff", "FEMALE": "#ffb3ff"},  # color palette␣
 ↪for sexes
        split = True,  # split violin plots by hue
        ax = axes[i]  # assign subplot
    )
    axes[i].set_title(f"{var} by Species and Sex")  # set title for each plot
    axes[i].set_ylabel(var)  # set y-axis label
    axes[i].get_legend().set_visible(False)  # hide individual plot legends

    # capture legend handles and labels from the first plot
    if i == 0:
        handles, labels = axes[i].get_legend_handles_labels()

# add a single legend for all subplots
fig.legend(
          handles,  # legend handles
          labels,  # legend labels
          loc = 'center right',  # position legend to the right of the plots
          title = "Sex",  # set legend title
          bbox_to_anchor = (1.1, 0.5)  # position legend outside of the figure
)

# adjust layout for better spacing
plt.tight_layout()
plt.show()
```



In the violin plot, while some distinctions can be observed between males and females within the same species—such as males generally having larger body mass and longer flippers—the overall differences are not pronounced enough to serve as a clear distinguishing factor for species classification. This suggests that sex alone does not significantly contribute to differentiating penguin species.

1.3 Feature Selection and Modeling

The violin plot clearly demonstrates that sex has limited ability to differentiate between penguin species. Its distribution shows significant overlap across species, making it a low-information feature that is unsuitable for classification purposes. In contrast, island is shown to be a highly informative categorical feature, as highlighted in Table 2 and figure 1(scatter plot). Penguin species exhibit distinct geographic distributions: Gentoo and Adelie are found only on Biscoe Island, Chinstrap and Adelie are found only on Dream Island, and Adelie is the sole species on Torgersen Island. This geographic separation significantly reduces classification uncertainty, making island an essential categorical variable for the model.

The violin plots also provide strong support for the inclusion of culmen length and culmen depth as key quantitative features. These two features not only exhibit strong overall separation between species but also allow for direct differentiation of certain species. For example, Adelie penguins can be identified by their significantly shorter culmen lengths, while Gentoo penguins are characterized by their shallower culmen depths. Once Adelie and Gentoo are classified using these features, the remaining group—Chinstrap—can be easily identified, completing the classification. The combination of culmen length and culmen depth captures richer feature space information, enhancing the model's discriminative ability.

The box plots of flipper length and body mass highlight their potential as additional features. While

Gentoo penguins are clearly separable in both dimensions, Adelie and Chinstrap penguins show considerable overlap, reducing their effectiveness for distinguishing these two species. However, these features may still complement culmen length and culmen depth, particularly in separating Gentoo penguins from the other species.

The histograms of Delta 15 N and Delta 13 C reveal significant overlap between species, indicating limited discriminatory power for classification. These features are therefore less informative and were excluded from the feature set. Similarly, the violin plots reaffirm the limited utility of sex as a classification variable due to the lack of meaningful variation across species.

In conclusion, culmen length and culmen depth were selected as the primary quantitative variables for their strong differentiating capabilities, while island was chosen as the categorical variable due to its ability to reduce classification uncertainty. Although flipper length and body mass show overlap for some species, they may still contribute as supplementary features, particularly for separating Gentoo penguins. This carefully curated feature set ensures the model leverages the most significant interspecies differences for accurate classification.

To ensure robust predictions, we selected three models with complementary strengths: 1. Logistical Regression 2. Random Forest 3. Support Vector Machine

1.3.1 Logistic Regression

Logistic regression is a fundamental classification algorithm used to predict the probability of a binary outcome (e.g., classifying between two species). For multi-class classification (like in the Palmer Penguins dataset), logistic regression uses a multinomial extension. This model is popular due to its simplicity, interpretability, and effectiveness for linearly separable data. Below, we will steps to implement a logistic regression model and discuss evaluate its performance using accuracy score, confusion matrix, classification report, and decision regions.

[19]:
```
lg_quantitative_variable = ["Culmen Length (mm)", "Culmen Depth (mm)"]  #␣
↪quantitative features
lg_qualitative_variable = "Island"                          # qualitative feature
lg_features = [lg_qualitative_variable] + lg_quantitative_variable

lg_X_train, lg_X_test = X_train[lg_features], X_test[lg_features]
print(lg_X_train.shape, lg_X_test.shape, y_train.shape, y_test.shape)
```

```
(262, 3) (62, 3) (262, 1) (62, 1)
```

[20]:
```
# train logistic regression model
LR = LogisticRegression(max_iter = 10000)
LR.fit(lg_X_train, y_train)
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/utils/validation.py:1300:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[20]:
```
LogisticRegression(max_iter=10000)
```

[21]:
```
# evaluate the model
lg_y_pred = LR.predict(lg_X_test)
accuracy = accuracy_score(y_test, lg_y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

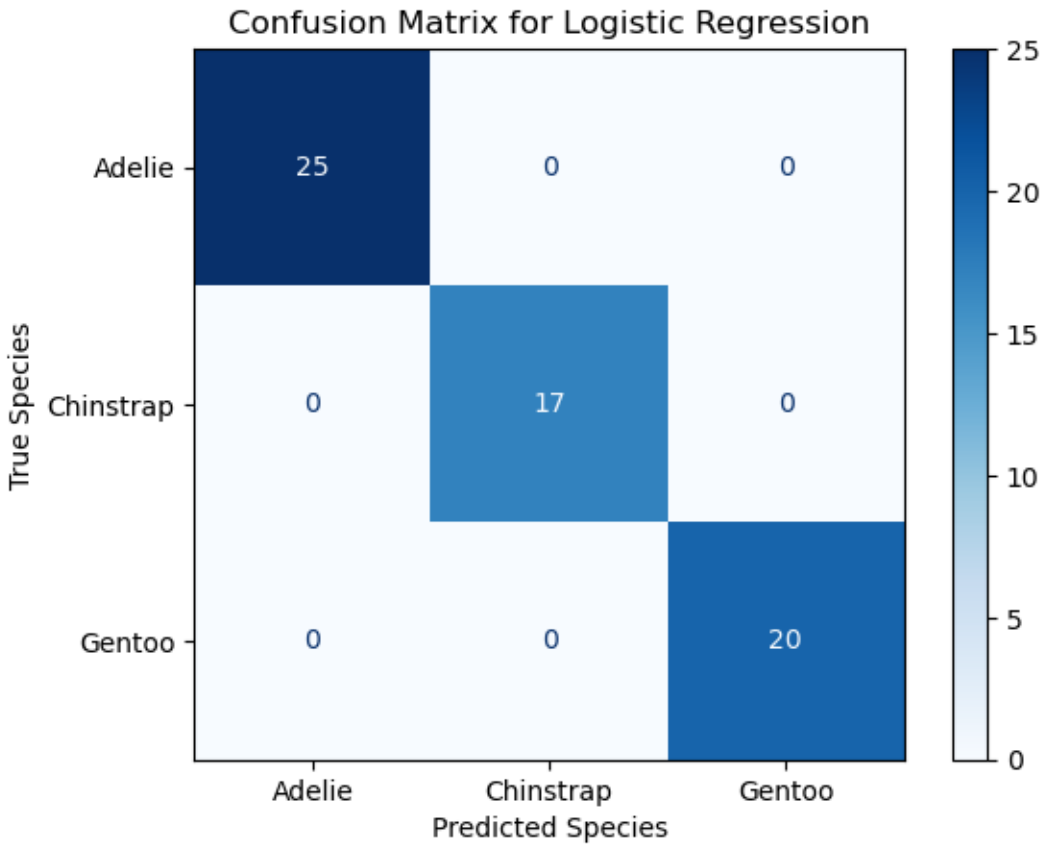
[22]:
```
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],  # regularization strengths
    'penalty': ['l1', 'l2'],       # regularization types
    'solver': ['liblinear', 'saga']  # solvers supporting 'l1' penalty
}

# set up the cross-validation with GridSearchCV
grid_search = GridSearchCV(estimator=LR, param_grid=param_grid, cv=5,␣
 ↪scoring='accuracy')

# fit the model to the training data
grid_search.fit(lg_X_train, y_train.squeeze())

# best hyperparameters and corresponding score
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_}")

# evaluate the best model on the test set
lg_best_model = grid_search.best_estimator_
lg_y_pred = lg_best_model.predict(lg_X_test)
accuracy = accuracy_score(y_test, lg_y_pred)
print(f"Test set accuracy: {accuracy}")
```

```
Best parameters: {'C': 10, 'penalty': 'l1', 'solver': 'saga'}
Best cross-validation score: 0.9846153846153847
Test set accuracy: 1.0
```

From the result analysis, it can be seen that the optimal hyperparameters are {C: 10, penalty: 'l1', solver: 'saga'}, with a cross-validation score of approximately 0.9846 and a test set accuracy of 1.0. This indicates that the model performs exceptionally well on both the training and test sets. However, the perfect accuracy on the test set might suggest that the data features are very easy to classify or that there is a data leakage issue (e.g., overlap between training and test samples). Additionally, although the cross-validation score is slightly lower, it is still close to 1, indicating that the model has strong generalization performance on the training set. We will continue to explore further.

[23]:
```
# define species names for better interpretability
# confusion matrix display with appropriate labels
conf_matrix = confusion_matrix(y_test, lg_y_pred)
disp = ConfusionMatrixDisplay(conf_matrix, display_labels =␣
 ↪cleaned_penguin["Species"].unique())
```

```
disp.plot(cmap = "Blues")
plt.title("Confusion Matrix for Logistic Regression")
plt.xlabel("Predicted Species")
plt.ylabel("True Species")
plt.show()

# displaying feature coefficients
coef_df = pd.DataFrame({
    "Feature": ["Island", "Culmen Length (mm)", "Culmen Depth (mm)"],
    "Coefficient": lg_best_model.coef_[0] # access the coefficients of the LR␣
 ↪model
})
# highlights the most influential values
coef_df.sort_values(by="Coefficient", ascending=False, inplace=True)
coef_df.reset_index(drop=True, inplace=True)

# display feature coefficients with species context
print("Logistic Regression Feature Coefficients:")
print(coef_df.to_string(index=False))

# classification report for detailed metrics
print("Classification Report:")
print(classification_report(y_test, lg_y_pred))
```




```
Logistic Regression Feature Coefficients:
           Feature  Coefficient
 Culmen Depth (mm)     3.318075
            Island     2.830524
Culmen Length (mm)    -1.433942
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        25
           1       1.00      1.00      1.00        17
           2       1.00      1.00      1.00        20

    accuracy                           1.00        62
   macro avg       1.00      1.00      1.00        62
weighted avg       1.00      1.00      1.00        62
```

The confusion matrix and classification report indicate that the logistic regression model achieved perfect accuracy, with all 62 samples correctly classified into their respective species: Adelie, Chinstrap, and Gentoo. The precision, recall, and F1-scores are all 1.00, reflecting a flawless performance in terms of predicting the true species labels. The feature coefficients reveal that culmen depth

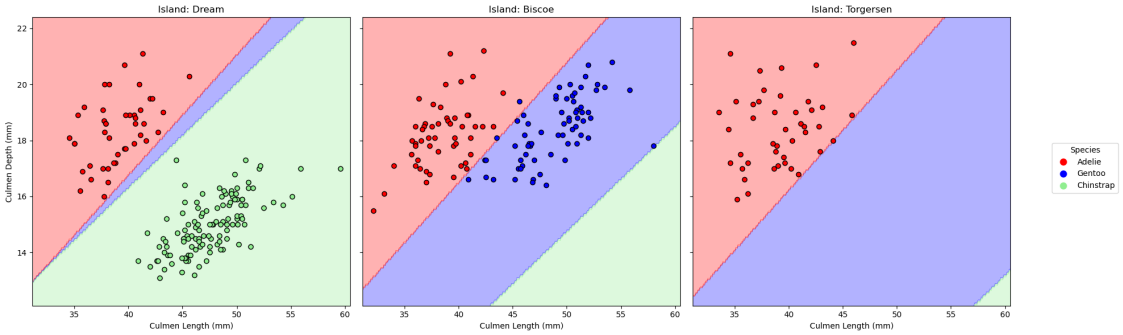
(3.318) had the most significant positive influence on classification, followed by island (2.831), while culmen length (-1.434) contributed negatively. Despite the seemingly perfect results, this outcome could indicate a lack of complexity in the dataset or potential overfitting in a controlled environment. Further validation on unseen data or cross-validation would be essential to ensure the model's generalizability, and testing with more complex models might provide insight into whether logistic regression's linear assumptions fully capture the data's underlying structure.

[24]:
```
def decision_regions_island(X, y, classifier, res=0.1):
    """
    Plots the decision regions of a classifier for all islands in a horizontal␣
↪layout.

    Parameters:
    -----------
        X (pd.DataFrame): Feature dataset (includes island as one of the␣
↪columns).
        y (pd.Series or np.array): Target labels.
        classifier: A trained sklearn-compatible classifier.
        res (float): Resolution of the grid for plotting.

    Returns:
    --------
        None
    """
    # Dictionaries for species and island names
    Species = {0: 'Adelie', 1: 'Gentoo', 2: 'Chinstrap'}
    Islands = {0: 'Dream', 1: 'Biscoe', 2: 'Torgersen'}

    # Define colors and colormap
    colors = ('red', 'blue', 'lightgreen')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # Create subplots for horizontal arrangement
    fig, ax = plt.subplots(1, 3, figsize=(18, 6), sharex=True, sharey=True)  #␣
↪3 horizontal subplots

    for i in range(3):
        # Set plot limits for the first two features
        x_min, x_max = X["Culmen Length (mm)"].min() - 1, X["Culmen Length␣
↪(mm)"].max() + 1
        y_min, y_max = X["Culmen Depth (mm)"].min() - 1, X["Culmen Depth (mm)"].
↪max() + 1

        # Create a grid for decision boundaries
        xx, yy = np.meshgrid(np.arange(x_min, x_max, res), np.arange(y_min,␣
↪y_max, res))
```

```
        # Create an array with constant island index
        island_array = np.full(xx.ravel().shape, i)

        # Predict grid points using the classifier
        grid_points = pd.DataFrame({
            "Island": island_array,
            "Culmen Length (mm)": xx.ravel(),
            "Culmen Depth (mm)": yy.ravel()
        })
        Z = classifier.predict(grid_points).reshape(xx.shape)

        # Plot the decision boundary
        ax[i].contourf(xx, yy, Z, alpha=0.3, cmap=cmap)
        ax[i].set_xlim(xx.min(), xx.max())
        ax[i].set_ylim(yy.min(), yy.max())

        # Filter training data for the current island
        island_mask = X["Island"] == i
        X_i = X[island_mask]
        y_i = y[island_mask]

        # Plot training points
        for idx, cl in enumerate(np.unique(y)):
            ax[i].scatter(
                x=X_i[(y_i == cl).squeeze()]["Culmen Length (mm)"],
                y=X_i[(y_i == cl).squeeze()]["Culmen Depth (mm)"],
                c=colors[idx],
                edgecolor="k",
                label=Species[cl] if i == 0 else ""  # Add legend only to the␣
↪first subplot
            )

        # Add labels, title, and legend for each subplot
        ax[i].set_title(f"Island: {Islands[i]}")
        ax[i].set_xlabel("Culmen Length (mm)")
        if i == 0:
            ax[i].set_ylabel("Culmen Depth (mm)")

    # Add a single legend for all subplots
    fig.legend(
        handles=[plt.Line2D([0], [0], marker='o', color='w', markerfacecolor=c,␣
↪markersize=10) for c in colors],
        labels=list(Species.values()),
        title="Species",
        loc='center right',
        bbox_to_anchor=(1.1, 0.5)
    )
```

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

[25]:
```
decision_regions_island(X, y, lg_best_model)
```



The visualization illustrates the decision boundaries for classifying penguin species (Adelie, Gentoo, and Chinstrap) based on culmen length and culmen depth across three islands: Dream, Biscoe, and Torgersen. Each subplot represents one island, with data points color-coded by species and shaded regions indicating the classification model's predictions. The model effectively separates Gentoo penguins (blue) from the other species, as they occupy distinct regions. However, notable overlap exists between Adelie (red) and Chinstrap (green), particularly on Torgersen and Biscoe islands, suggesting potential misclassifications due to similar culmen features. The decision boundaries imply the use of a linear classifier, such as logistic regression, which assumes linear separability between classes. While this works reasonably well here, it has limitations in capturing complex relationships. If the relationship between features and species is nonlinear, the model may struggle, leading to reduced accuracy. Additionally, an imbalance in species representation could cause the model to favor the majority class, further increasing misclassifications. Advanced models like random forests could better handle such complexities and adapt to intricate data patterns, improving classification performance, let's do it!

1.3.2 Random Forest Model

Random Forest Model makes predictions by making multiple decision trees. It takes a data, splits it into random subsets, trains trees to make predictions, then combines the trees. Using the forest of random trees, a decision is made by taking the majority prediction of all the trees in the forest, hence the name Random Forest Model. In the following code, we will use the Random Forest Model to predict the Penguin Species based on the Culmen Length and Depth, as well as the Island recorded within the Palmer Penguin Data set.

[26]:
```
rfm_features = ["Island", "Culmen Length (mm)", "Culmen Depth (mm)"]
# X_train, X_test, y_train, y_test = data_splitting(rfm_penguins, "Species",␣
  ↪test_size= 0.20)
rfm_X_train, rfm_X_test = X_train[rfm_features], X_test[rfm_features]
```

Next, a parameter grid was used to select for the best RandomForestClassifier parameters to be used to make a prediction, using the cross_value_score method.

Notice: this cell may take some time to run.

[27]:
```
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_features': ['sqrt', 'log2', None],
    'max_depth': [5, 10, None],
    'max_leaf_nodes': [5, 10, None],
}
# set a parameter grid to test over.

grid_search = GridSearchCV(RandomForestClassifier(),
                           param_grid=param_grid)
grid_search.fit(rfm_X_train, y_train.squeeze())
# run the RandomForestClassifier with all the parameters, and obtain the best␣
 ↪estimator parameter
print(grid_search.best_estimator_)
```

```
RandomForestClassifier(max_depth=10)
```

Using the parameters found above, the random forest model is trained with the training data. Next, the testing dataset is passed through the model, and the model's prediction of penguin species is checked against the actual penguin species for accuracy.

[28]:
```
rf = grid_search.best_estimator_
# set up a RandomForestClassifier with the best estimator parameters found␣
 ↪above.

rf.fit(rfm_X_train, y_train)
# train it with the training data

rfm_y_pred = rf.predict(rfm_X_test)
# predict the testing data
accuracy = accuracy_score(y_test, rfm_y_pred)
# check for testing data accuracy
print("Accuracy:", accuracy)
# print the accuracy score.
```

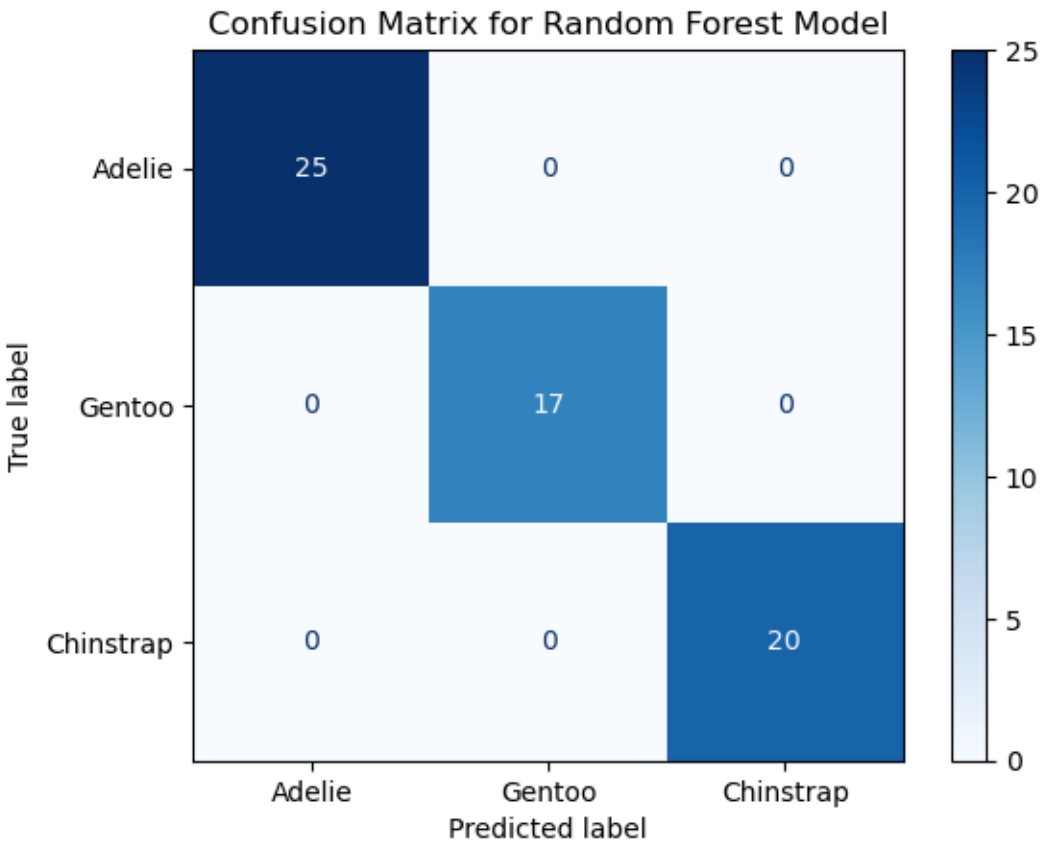
```
Accuracy: 1.0
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:1474:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  return fit_method(estimator, *args, **kwargs)
```

Common misclassifications: There appears to be no significant common misclassifications. The model is able to predict the penguin species at a high accuracy, above 90%.

The accuracy of the model is visualized using a Confusion Matrix, where the prediction of the model is plotted against the actual data.

[29]:
```
cm = confusion_matrix(y_test, rfm_y_pred)
# creates a confusion matrix with the test and prediction data
cm_display = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels =␣
↪["Adelie", "Gentoo", "Chinstrap"])
# save the confusion matrix as a variable, and plot it.
cm_display.plot(cmap="Blues")
plt.title("Confusion Matrix for Random Forest Model")

plt.show()
```

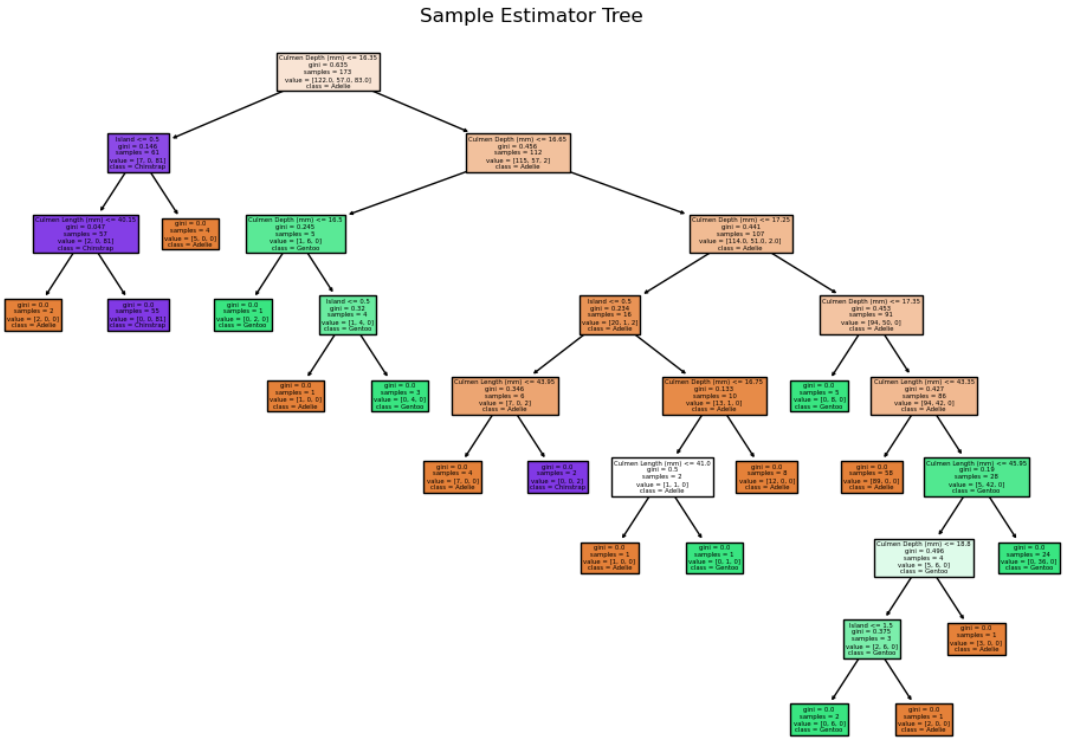


A sample decision tree within the random forest is visualized below. The actual Random Forest Model is made up of many such decision trees

[30]:
```
fn=["Island", "Culmen Length (mm)", "Culmen Depth (mm)"]
cn=["Adelie", "Gentoo", "Chinstrap"]
# create lists for the feature names and the species names
```

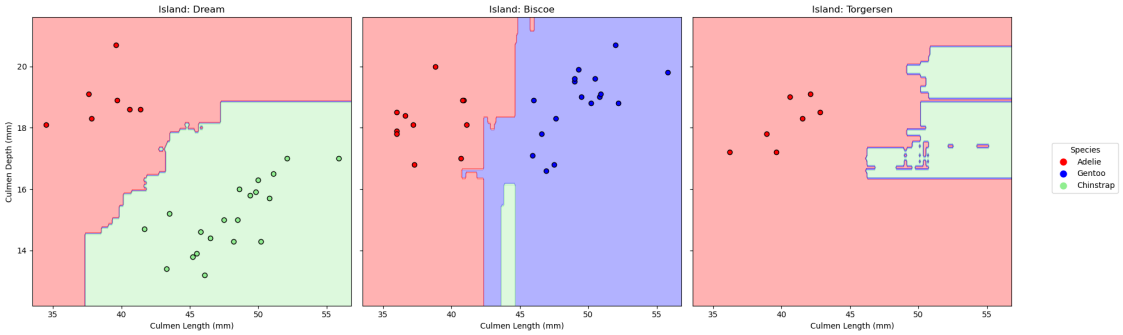
```
rf = grid_search.best_estimator_
# set up a RandomForestClassifier with the best estimator parameters found␣
 ↪above.

rf.fit(rfm_X_train, y_train)
# train it with the training data

individual_tree = rf.estimators_[0]  # Get the first tree (you can choose any␣
 ↪index)
# extract the first tree from the forest

plt.figure(figsize=(12, 8))
tree.plot_tree(individual_tree, feature_names=fn, class_names=cn, filled=True)
# plot the tree.
plt.title("Sample Estimator Tree")
plt.show()
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/base.py:1474:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  return fit_method(estimator, *args, **kwargs)
```



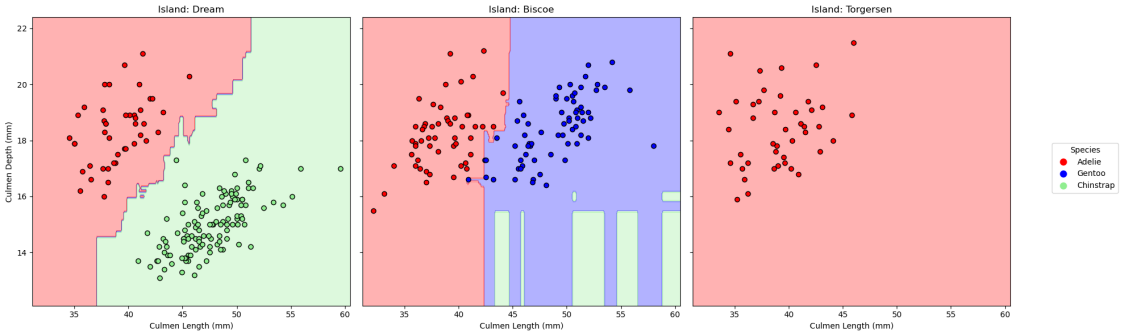
Using the Decision Region plotting function above (in the logistic regression model), we first plot the decision regions against the data of penguins used to test the accuracy of the model across each island. It can be seen that the decision region successfully predicted each data point within the correct region.

[31]:
```
rf = grid_search.best_estimator_
rf.fit(rfm_X_train, y_train.squeeze())
# reload the random forest classifier.

decision_regions_island(X_test, y_test, rf)
```



Next, the Decision Region plotting function was used on all the datapoints from the Palmer Penguin dataset, to visualize it against all the Palmer Penguin data present. The decision regions are highly accurate.

[32]:
```
rf = grid_search.best_estimator_
rf.fit(rfm_X_train, y_train.squeeze())
# reload the random forest classifier.

decision_regions_island(X, y, rf)
```



In Summary, the Random Forest Model is a highly accurate machine learning model that can result in almost perfect predictions when given enough data. However, it is also very complexed, leading

to longer run times, and the usage of large amounts of storage spaces. Because of the high amount of decision trees used to make a prediction, it can sometimes be difficult to understand the logic behind each prediction. Additionally, the Random Forest Model will not work with a small amount of data.

1.3.3 Support Vector Machine(SVM)

The implementation of the Support Vector Machine (SVM) model uses culmen length, culmen depth, and island as features for penguin species classification. The grid search optimization identified the best hyperparameters as C=10, gamma='scale', and kernel='linear'. The cross-validation accuracy of 0.9846 on the training set indicates that the model performs exceptionally well under these configurations, achieving near-perfect classification on the training data.

[33]:
```
svm_features = ["Island", "Culmen Length (mm)", "Culmen Depth (mm)"]
svm_X_train, svm_X_test = X_train[svm_features], X_test[svm_features]
```

[34]:
```
# Cross-validation to choose complexity parameters.
param_grid = {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto'], 'kernel':␣
 ↪['linear', 'rbf']}
svm_model = GridSearchCV(SVC(), param_grid, cv=5)  # cross-validation with 5␣
 ↪folds inside GridSearchCV

# fit model with cross-validation
svm_model.fit(svm_X_train, y_train.squeeze())
# best hyperparameters
print("Best Parameters:", svm_model.best_params_)
best_svm_model = svm_model.best_estimator_
```

```
Best Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'linear'}
```

[35]:
```
# cross-validation accuracy on training data
cv_scores = cross_val_score(best_svm_model, svm_X_train, y_train.squeeze(),␣
 ↪cv=5)
print(f"\nCross-Validation Accuracy (Training Set): {cv_scores.mean():.4f}")
```

```
Cross-Validation Accuracy (Training Set): 0.9846
```

The reported cross-validation accuracy of 0.9846 for the Support Vector Machine (SVM) model on the training set reflects excellent performance, suggesting that the model is highly effective in distinguishing between penguin species using the selected features (culmen length, culmen depth, and island). The use of 5-fold cross-validation ensures that the evaluation accounts for variance within the training data and provides a robust measure of model performance. This result indicates that the linear kernel, combined with the hyperparameters (C=10, gamma='scale'), is well-suited for this dataset.

However, high cross-validation accuracy raises two considerations. First, while it demonstrates strong predictive power within the training data, it does not guarantee equivalent performance on unseen test data. Second, the near-perfect accuracy could be indicative of a dataset with clear

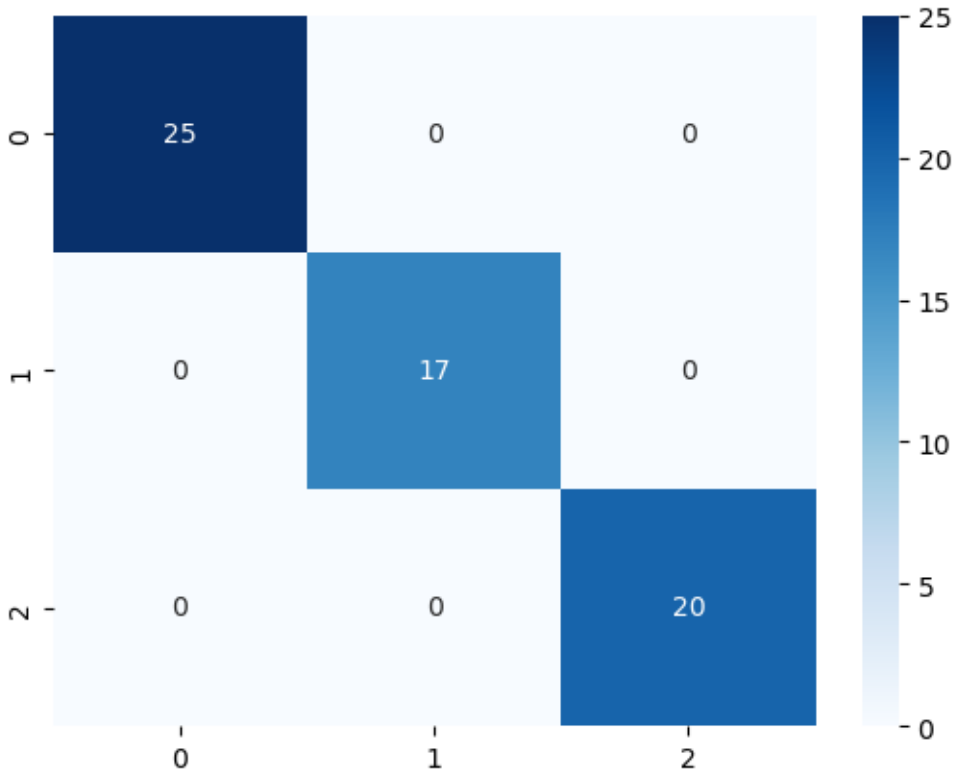
separability or low noise, which might limit the model's generalization if the test data contains more variability or noise.

To strengthen the analysis, the next steps should include evaluating the model on an independent test set and analyzing the confusion matrix to identify any misclassifications.

[36]:
```
# Evaluation on unseen testing data, including a confusion matrix.
svm_y_pred = best_svm_model.predict(svm_X_test)
print("\nClassification Report:\n", classification_report(y_test, svm_y_pred))
print(f"Testing Set Accuracy: {accuracy_score(y_test, svm_y_pred):.4f}\n\n")
print("\nConfusion Matrix:\n")
sns.heatmap(confusion_matrix(y_test, svm_y_pred), annot=True, fmt='d',␣
 ↪cmap="Blues")
```

```
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00        25
           1       1.00      1.00      1.00        17
           2       1.00      1.00      1.00        20

    accuracy                           1.00        62
   macro avg       1.00      1.00      1.00        62
weighted avg       1.00      1.00      1.00        62

Testing Set Accuracy: 1.0000



Confusion Matrix:
```

[36]:
```
<Axes: >
```



The testing results for the SVM model indicate perfect classification performance, achieving a testing set accuracy of 1.000. The classification report confirms that precision, recall, and F1-scores for all three species (Adelie, Chinstrap, and Gentoo) are 1.00. The confusion matrix reveals no misclassifications, with all predicted labels aligning perfectly with the true labels (25 Adelie, 17 Chinstrap, and 20 Gentoo correctly classified). This flawless performance on the testing data validates the model's effectiveness and demonstrates its capability to generalize well beyond the training set.

The high accuracy across both training and testing data suggests that the dataset features (culmen length, culmen depth, and island) are highly discriminative for species classification. Additionally, the linear kernel chosen during hyperparameter tuning appears to have been an optimal choice for this dataset, as the decision boundaries are well-suited to the feature separability.

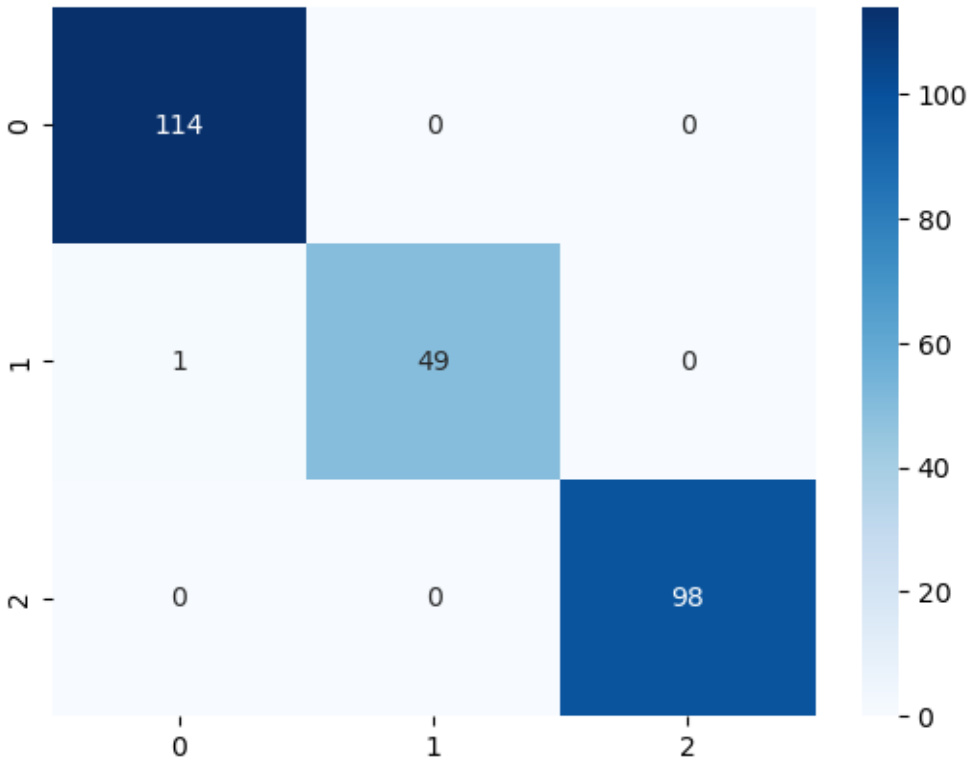
However, such perfect accuracy raises questions about the potential simplicity of the dataset, such as low variability, minimal noise, or an imbalanced test set. While these results are encouraging, further testing on an independent dataset or through cross-validation could provide additional confidence in the model's robustness.

[37]:
```
svm_y_pred = best_svm_model.predict(svm_X_train)
print("\nClassification Report:\n", classification_report(y_train, svm_y_pred))
print(f"Tranining Set Accuracy: {accuracy_score(y_train, svm_y_pred):.4f}")
print("\nConfusion Matrix:\n")
```

```
sns.heatmap(confusion_matrix(y_train, svm_y_pred), annot=True, fmt='d',␣
 ↪cmap="Blues")
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       114
           1       1.00      0.98      0.99        50
           2       1.00      1.00      1.00        98

    accuracy                           1.00       262
   macro avg       1.00      0.99      1.00       262
weighted avg       1.00      1.00      1.00       262

Tranining Set Accuracy: 0.9962

Confusion Matrix:
```

[37]: <Axes: >

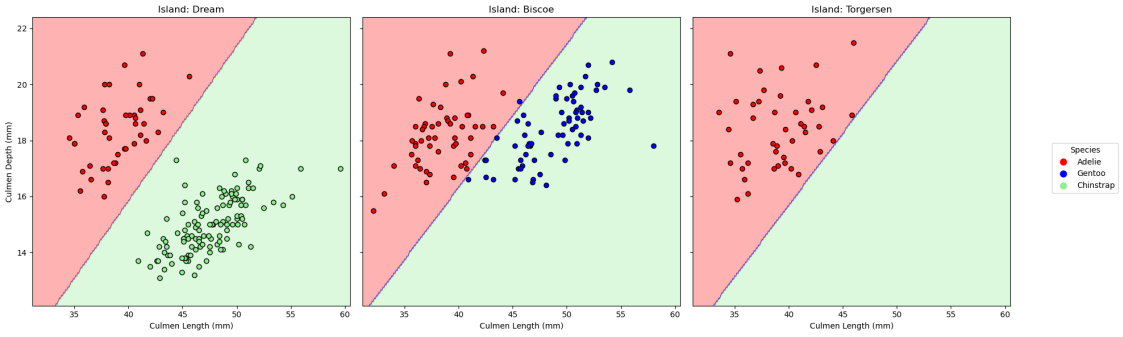


The confusion matrix further reinforces the strong performance, with the vast majority of predic-

tions aligning perfectly with the true labels (114 Adelie, 49 Chinstrap correctly classified out of 50, and 98 Gentoo). The slight deviation from perfect accuracy (due to one misclassification) indicates a well-fitted model without significant overfitting.

Decision Regions for All of the Data Set

[38]:
```
decision_regions_island(X, y, best_svm_model)
```



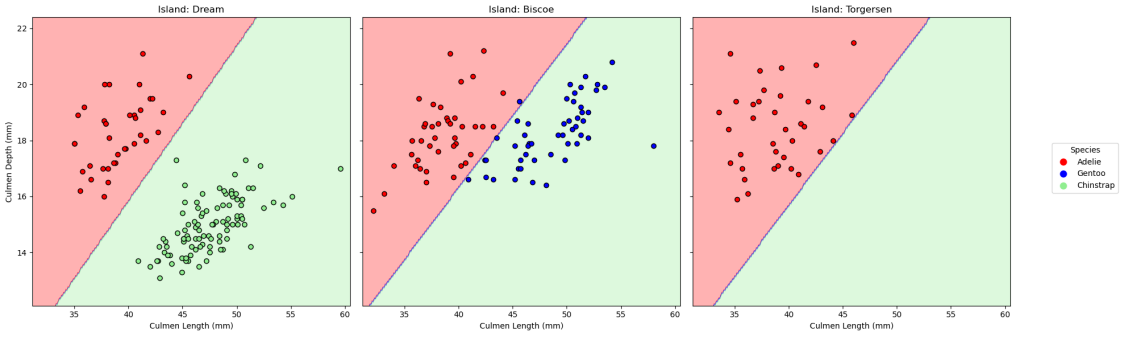
The regions are distinctly divided, reflecting the SVM's capability to construct linear boundaries that effectively segment the feature space.

For Island Dream, the decision boundary clearly separates Adelie (red) and Chinstrap (green) species, with no apparent overlap, indicating well-separated data points. On Island Biscoe, the boundaries differentiate the Gentoo (blue) species effectively, while Adelie and Chinstrap have slight overlaps near the boundary, which could lead to minor misclassifications in cases of variability in feature values. On Island Torgersen, the model also displays well-defined decision regions, though the data is predominantly split between Adelie and Chinstrap, showing minimal complexity.

These decision regions confirm the linear kernel's suitability for this dataset, as the species exhibit significant separability in the feature space.

Decision Regions for the Training Data Set

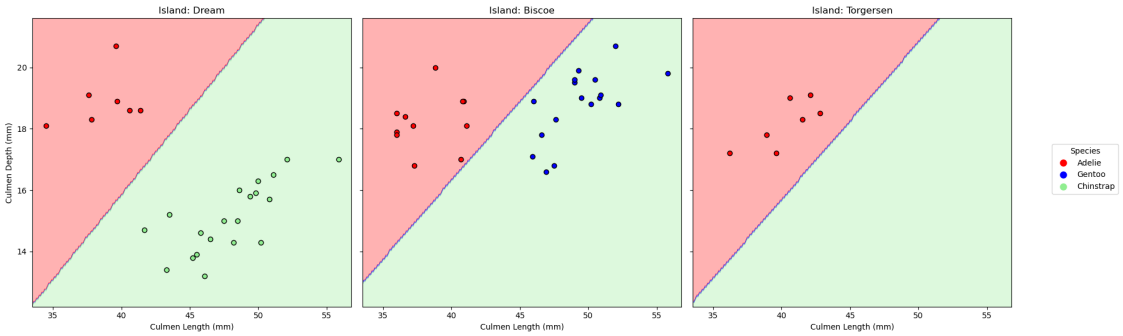
[39]:
```
decision_regions_island(X_train, y_train, best_svm_model)
```



The decision regions for the training dataset visualize the linear SVM model's boundaries separating the three penguin species (Adelie, Gentoo, and Chinstrap) across the islands (Dream, Biscoe, and Torgersen) based on culmen length and depth. On Island Dream, the boundaries perfectly separate Adelie (red) and Chinstrap (green), with no overlap, indicating clear feature separability. For Island Biscoe, Gentoo (blue) is well-separated from Adelie and Chinstrap, though the boundaries between Adelie and Chinstrap are closer, suggesting sensitivity to feature variations. On Island Torgersen, Adelie and Chinstrap are similarly well-separated without any visible misclassifications. These regions confirm that the linear kernel effectively models the feature separability in the training data, aligning with the near-perfect training accuracy. However, while the boundaries appear robust for the training data, further testing on unseen or noisier data is necessary to validate the model's generalization.

Decision Regions for the Test Data Set

[40]:
```
decision_regions_island(X_test, y_test, best_svm_model)
```



For Island Dream, the separation between Adelie (red region) and Chinstrap (green region) remains robust, with all test points correctly classified and no overlap near the boundary. On Island Biscoe, Gentoo (blue region) is distinctly separated from Adelie and Chinstrap, aligning well with the linear separability observed in the training dataset. On Island Torgersen, the boundaries also remain effective, clearly distinguishing Adelie and Chinstrap without any visible misclassifications. These results reflect the SVM model's excellent performance on unseen data, corroborating its perfect test accuracy and confirming the adequacy of a linear kernel for this dataset. However, additional tests with more complex datasets or noisy features would further validate its robustness and highlight potential areas for improvement.

1.4 Discussion

For the overall performance of the models the best model out of the three was the **Random Forest Model** due to its high accuracy. The features we recommend are **Island, Culmen Length(mm), and Culmen Depth (mm)**, which are highly accurate at predicting the penguin species. The qualitative data Island narrowed down the potential penguin species to 2 or 1 depending on which Island. The three species of penguins showed different trends in the combination of quantitative columns of Culmen Length and Depth.

The logistic regression model is simple and effective for linear data. However, its performance can be limited if the relationships in the data are nonlinear. It provides simplicity but is likely to

underperform when compared to more flexible models. Despite its simplicity, Logistic Regression is often a strong baseline model, offering transparency in how predictions are made.The SVM model is suitable for high-dimensional data and performs well on complex decision boundaries. It is computationally intensive for larger datasets, requires scaling of input features, and is sensitive to choice of kernel and hyperparameters. The random forest model achieved high accuracy and handles nonlinear relationships well, is resistant to overfitting, and can handle categorical variables natively.

The model could be improved if there were additional biological features such as body mass, flipper length, or environmental features such as habitat data, nesting locations, climate zones, or proximity to water sources. Additionally, if more qualitative data such as behavioral traits or feeding patterns were available, that might enhance prediction accuracy. Another way to improve the accuracy of the models employed is through the use of more amounts of data. One concern about highly accurate machine learning models is that they may be overfitting against the data provided, prioritizing accuracy over overall trends. This can be solved by feeding the machine learning models with more training data, which reduces the chances of the model picking up on noise, and leads to better generalization of trends across the whole data.