Group Contribution Statement

Group members: Matthew Keshishian, Hayk Gargaloyan

Matthew was responsible for completing the discussion worksheets pertaining to this project so we could reuse their code. Hayk did most of the data cleaning and import. Hayk did much of the code for exploratory analysis while Matthew was responsible for feature selection and the accompanying dialogue. Hayk worked mostly on Logistic Regression and the function for plotting the decision regions. Matthew worked mainly on the Random Forest and K Nearest Neighbors Classifier. We reviewed all of each others code and went line-by-line debugging each model and function we created.

Data Import and Cleaning

In order to prevent the cleaning process from influencing our data in unforseen ways, we first perform a train test split. This holds part of the data for testing later, so that we are able to check the trained model without having to reuse the training data. Then, we clean the data to make it usable for our models. This involves ensuring that the columns we need are numeric, dropping excess data columns for convenience (this includes comments and sample number), and further partitioning the data into features and the target label. We want to predict species, so that will be the target label, and at this point features will include all un-dropped labels. As we continue, we will narrow down the features we use in our models, eventually selecting one qualitative and two quantitative features.

In [ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

penguins = pd.read_csv("palmer_penguins.csv")
train, test = train_test_split(penguins, test_size = 0.2, random_state=42)
penguins.head()
```

Out[ ]:

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Sex	Delta 15 N (o/oo)	Delta 13 C (o/oo)	Comments
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	11/11/07	39.1	18.7	181.0	3750.0	MALE	NaN	NaN	Not enough blood for isotopes.
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	11/11/07	39.5	17.4	186.0	3800.0	FEMALE	8.94956	-24.69454	NaN
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	11/16/07	40.3	18.0	195.0	3250.0	FEMALE	8.36821	-25.33302	NaN
3	PAL0708	4	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A2	Yes	11/16/07	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Adult not sampled.
4	PAL0708	5	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N3A1	Yes	11/16/07	36.7	19.3	193.0	3450.0	FEMALE	8.76651	-25.32426	NaN

In [ ]:
```
label_encoder_island = LabelEncoder()   # This will be useful later to ensure consistent encodings and help us convert back to species name
label_encoder_species = LabelEncoder()  # useful later for converting back to island name

def prep_penguin_data(data_df):
    """
    Prepares the penguin dataset for machine learning by encoding categorical variables,
    transforming dates, and splitting the data into features and target labels.

    Parameters:
    data_df (pd.DataFrame): The original penguin dataset.

    Returns:
    tuple: A tuple containing:
        - X (pd.DataFrame): The feature data after preprocessing.
        - y (pd.Series): The target labels (encoded species).

    Steps:
    1. Make a copy of the input DataFrame to prevent errors due to modifications.
    2. Encode non-numeric columns ('Region', 'Stage', 'Clutch Completion', 'Sex') using LabelEncoder.
    3. Encode 'Island' and 'Species' columns using predefined label encoders (`label_encoder_island`, `label_encoder_species`).
    4. Convert 'Date Egg' column to numeric values representing the number of days from the earliest date in the dataset.
    5. Drop irrelevant columns ('studyName', 'Sample Number', 'Individual ID', 'Comments').
    6. Remove rows containing NaN values.
    7. Split the preprocessed DataFrame into features (X) and target labels (y).

    """
    df = data_df.copy()  # prevent errors by making a copy

    le = LabelEncoder()
    non_numerics = ['Region', 'Stage', 'Clutch Completion', 'Sex']
    for col in non_numerics:
        df['old ' + col] = df[col]
        df[col] = le.fit_transform(df[col])  # encode non-numeric columns so our model can use them

    df['Island'] = label_encoder_island.fit_transform(df['Island'])
```

```
    df['Species'] = label_encoder_species.fit_transform(df['Species'])

    df['Date Egg'] = (pd.to_datetime(df['Date Egg'], format='%m/%d/%y') - pd.to_datetime(df['Date Egg'], format='%m/%d/%y').min()).dt.days
    # convert the date to a number of days from the earliest date egg in the data

    df = df.drop(columns=['studyName', 'Sample Number', 'Individual ID', 'Comments']) # remove these columns as they should not be considered
    df = df.dropna()   # remove rows containing NaN

    # split into X and y
    X = df.drop(['Species'], axis = 1)
    y = df['Species']

    return(X, y)
```

In [ ]:
```
X_train, y_train = prep_penguin_data(train)
X_test, y_test = prep_penguin_data(test)
```

Exploratory Analysis

Now that we have prepared and cleaned our data, we need to find which variables are actually conducive to predicting species. We can do this by creating tables and graphs to evaluate the relationships between our quantitative and qualitative data.

In [ ]:
```
from matplotlib import pyplot as plt

def get_mode(group):
  '''
    Gets information about the most frequent value in a dataframe with qualitative data.
    Takes a dataframe as a parameter, we should liekly run groupby before passing
    Returns dataframe with the mode, the frequency of the mode occuring, the total
    number of values, and the percentage of the data that is the mode.
  '''
    mode = group.mode().iloc[0]  # Get the first mode value for each column
    mode_freq = group.apply(lambda x: x.value_counts().iloc[0])  # Count of the mode for each column
    percentage = mode_freq / len(group) * 100  # Percentage of the mode occurrence
    # Combine mode, counts, and percentages into a single DataFrame
    result = pd.DataFrame({
        'mode': mode,
        'count': mode_freq,
        'population': len(group),
        'percentage': percentage.round(2)

    })
    return result

table1 = penguins.groupby('Species')[['Stage', 'Clutch Completion', 'Sex', 'Island', ]].apply(get_mode)
table2 = penguins.groupby('Island')[['Stage', 'Clutch Completion', 'Sex', 'Species']].apply(get_mode)
```

In [ ]:
```
table1
```

Out[ ]:

		mode	count	population	percentage
Species					
Adelie Penguin (Pygoscelis adeliae)	**Stage**	Adult, 1 Egg Stage	152	152	100.00
	Clutch Completion	Yes	138	152	90.79
	Sex	FEMALE	73	152	48.03
	Island	Dream	56	152	36.84
Chinstrap penguin (Pygoscelis antarctica)	**Stage**	Adult, 1 Egg Stage	68	68	100.00
	Clutch Completion	Yes	54	68	79.41
	Sex	FEMALE	34	68	50.00
	Island	Dream	68	68	100.00
Gentoo penguin (Pygoscelis papua)	**Stage**	Adult, 1 Egg Stage	124	124	100.00
	Clutch Completion	Yes	116	124	93.55
	Sex	MALE	61	124	49.19
	Island	Biscoe	124	124	100.00

From these tables, we found that Stage and Sex were not predictors for species or island. In all penguins surveyed, the stage was the same (Adult, 1 Egg Stage). For all islands and species, sex was approximately 50% Male and Female. We will eliminate these from our data.

One thing of note was that for both Chinstrap and Gentoo penguins, 100% of their population was located on the same island. We thus thought island might be a strong indicator of species, so we switched their places in the table (thus making Island the target variable).

In [ ]:
```
table2
```

Out[ ]:

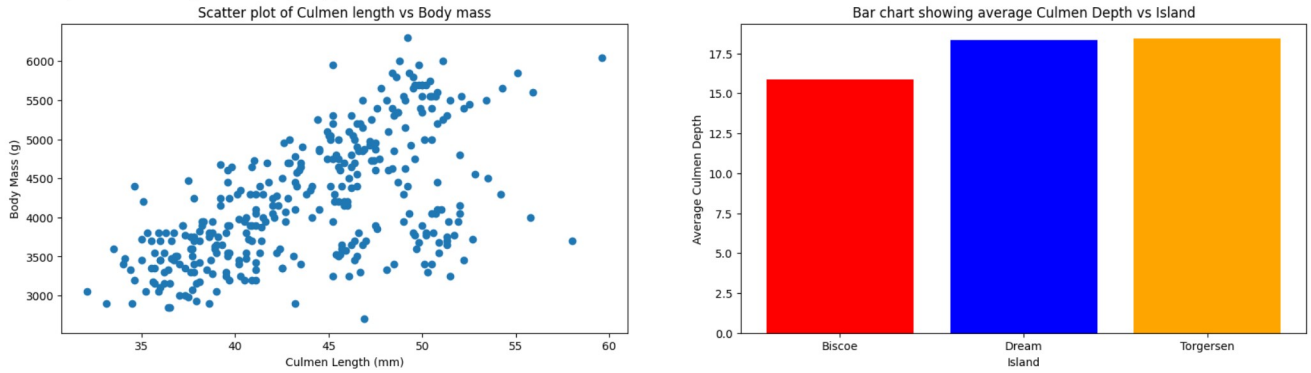
Island		mode	count	population	percentage
Biscoe	**Stage**	Adult, 1 Egg Stage	168	168	100.00
	Clutch Completion	Yes	158	168	94.05
	Sex	MALE	83	168	49.40
	Species	Gentoo penguin (Pygoscelis papua)	124	168	73.81
Dream	**Stage**	Adult, 1 Egg Stage	124	124	100.00
	Clutch Completion	Yes	106	124	85.48
	Sex	MALE	62	124	50.00
	Species	Chinstrap penguin (Pygoscelis antarctica)	68	124	54.84
Torgersen	**Stage**	Adult, 1 Egg Stage	52	52	100.00
	Clutch Completion	Yes	44	52	84.62
	Sex	FEMALE	24	52	46.15
	Species	Adelie Penguin (Pygoscelis adeliae)	52	52	100.00

As we thought, island is by far the strongest predictor of species from the qualitative data (and vice versa). We see that the island of Torgersen consists solely of Adelie Penguin, and they are also spread out among the other two islands. All chinstrap penguins live on the Dream island, while all Gentoo live on Biscoe.

Now that we have a qualitative variable, we also want to investigate which quantitative values would work best to find species. We create a bar and scatter plot comparing qualitative values from the Penguins dataset.

In [ ]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 5))
ax[0].scatter(penguins['Culmen Length (mm)'], penguins['Body Mass (g)'], label = 'Culmen Length vs Body Mass')
ax[0].set(title='Scatter plot of Culmen length vs Body mass' ,xlabel= 'Culmen Length (mm)', ylabel='Body Mass (g)')

avg_culmen_depth = penguins.groupby('Island')['Culmen Depth (mm)'].mean()
ax[1].bar(avg_culmen_depth.index, avg_culmen_depth.values, color=['red', 'blue', 'orange'])
ax[1].set(title='Bar chart showing average Culmen Depth vs Island', xlabel='Island', ylabel='Average Culmen Depth')
```

Out[ ]:
```
[Text(0.5, 1.0, 'Bar chart showing average Culmen Depth vs Island'),
 Text(0.5, 0, 'Island'),
 Text(0, 0.5, 'Average Culmen Depth')]
```



Feature Selection

The scatter plot above shows a positive correlation between Culmen Length and body mass. Just at a glance, this does not seem like a strong correlation, so it may be worthwile to include both culmen length and body mass in our model. The third bar chart shows that the average culmen length is identical on Dream and Torgersen, but not on Biscoe. This suggests that Biscoe might contain a different species distribution than the other two islands.

Now that we have seen the relationship between other variables, we can see how each variable we have found significant relates to the species.

In [ ]:
```
def penguin_summary_table(group_cols, value_cols):
    return penguins.groupby(group_cols)[value_cols].mean().round(2)
```

In [ ]:
```
penguin_summary_table(["Species"], ["Culmen Length (mm)", "Culmen Depth (mm)", "Body Mass (g)", "Flipper Length (mm)"])
```

Out[ ]:

Species	Culmen Length (mm)	Culmen Depth (mm)	Body Mass (g)	Flipper Length (mm)
Adelie Penguin (Pygoscelis adeliae)	38.79	18.35	3700.66	189.95
Chinstrap penguin (Pygoscelis antarctica)	48.83	18.42	3733.09	195.82
Gentoo penguin (Pygoscelis papua)	47.50	14.98	5076.02	217.19

From this table we can see that a strong quantitative predictor for Gentoo penguins is Body mass. We thus know that if a penguin's body mass is around 5000g, it is likely Gentoo. Otherwise it is Chinstrap or Adelie. To distinguish between the two, you can use Culmen Length. Adelie have a significantly shorter culmen length on average than Chinstrap or Gentoo, so a penguin with around 39mm culmen length would likely be Adelie. A penguin with near 3700g mass and 49mm culmen depth is Chinstrap.

As discussed in the previous section, island was a strong indicator of species compared to other qualitative variables. The values chosen allow us to make a mock decision tree.

In [ ]:
```
def decision_tree(island, mass, culmenLength):
  '''
    Takes island, mass, and culmen length variables and predicts which species the penguin is.
    This never actually gets used, just to sample how our variables can be used to
    classify penguins.
    Input: island name (string), mass in g (float), culmenLength in mm (float)
  '''
    if island == "Torgersen"
        return "Adelie"
    elif island == "Biscoe":
        if mass > 4500 or culmenLength > 44
            return "Gentoo "
        else:
            return "Adelie"
    else:
        if culmenLength > 44:
            return "Chinstrap"
        else:
            return "Adelie"
```

Modeling

We now have two qualitative values (Culmen length (mm)) and Body Mass (g)) and one qualitative (Island) for our machine learning models. Our exploratory data analysis found them extremely predictive of species (see above commentary). We will use them to train three models, with the first being a polynomial regression. Our first step is to use cross validation to find the optimal degree.

In [ ]:
```
def drop_useless(data_df):
    """
    Only keep the values we selected in previous section
    """
    df = data_df.copy()
    df = df[['Culmen Length (mm)', 'Body Mass (g)', 'Island']]# only keep the columns containing
    return df
```

In [ ]:
```
X_train = drop_useless(X_train)
X_test = drop_useless(X_test)
```

In [ ]:
```
X_train
```

Out[ ]:

	Culmen Length (mm)	Body Mass (g)	Island
66	35.5	3350.0	0
229	46.8	5150.0	0
7	39.2	4675.0	2
140	40.2	3400.0	1
323	49.1	5500.0	0
...	...	...	...
188	47.6	3850.0	1
71	39.7	3900.0	2
106	38.6	3750.0	0
270	46.6	4850.0	0
102	37.7	3075.0	0

260 rows × 3 columns

We now have cleaned our data to only our chosen relevant variables and can begin training our three machine learning models. For this project, we selected Logistic Regression, Random Forest, and a K Nearest Neighbors Classifier. We will analyze the performance of each model to determine its strengths and weaknesses.

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
import numpy as np
```

Logistic regression is a model that takes input features and uses them to calculate a score. This score is then used to calculate the probability of each outcome. For example, if a score is higher than 0.5 it will predict outcome 1 and if it is lower than 0.5 it will predict outcome 2. By default, the Logistic Regression in scikit learn is a binary regression (meaning it can only work for binary choices, such as male or female). Since we are trying to distinguish between three species, we will use a multionmial Logistic Regression.

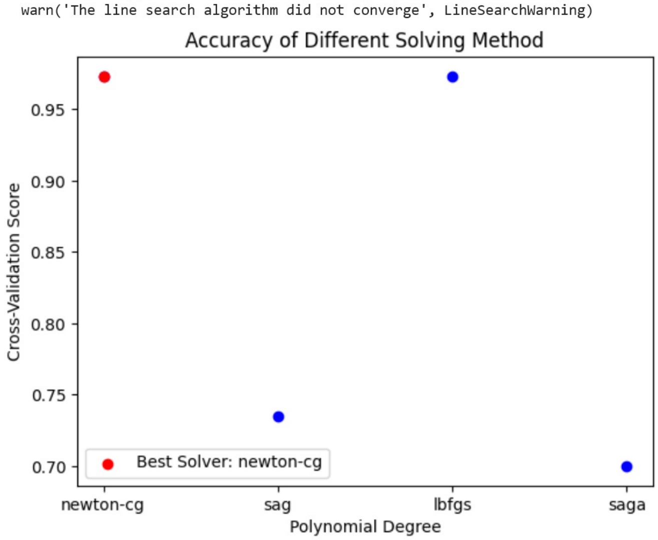
Before we actually train a regression, we have to find which parameters work best for our model. We can do this by employing a cross validation function. This works by using small slices of our training data as a test. For example, it will use a 20% slice of training as a test and train the model on the remaining 80%. After calculating a score for this, it will repeat the process with the next slice of data.

Multinomial Logical Regression has several different solving methods it can use to calculate probability scores. We will calculate a cross validation score for each method and use them to decide which is the best for our data.

In [ ]:
```
from sklearn.linear_model import LogisticRegression
plt.title('Accuracy of Different Solving Method')
plt.xlabel('Polynomial Degree')
```

```python
plt.ylabel('Cross-Validation Score')
plt.xticks(range(0, 10))


solvers = ['newton-cg', 'sag', 'lbfgs', 'saga']
methods = []
scores = []
bestSolver = None
bestScore = 0

for s in solvers:
  #because of size of data set, increased iterations to 10000
  LR = LogisticRegression(multi_class='multinomial', solver=s, max_iter=10000)
  cvScore = np.mean(cross_val_score(LR, X_train, y_train, cv=10))

  if cvScore > bestScore:
      bestScore = cvScore
      bestSolver = s

  scores.append(cvScore)
  methods.append(s)

plt.scatter(methods, scores, color='Blue')
plt.scatter(bestSolver, bestScore, color='red', label=f'Best Solver: {bestSolver}')
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
```

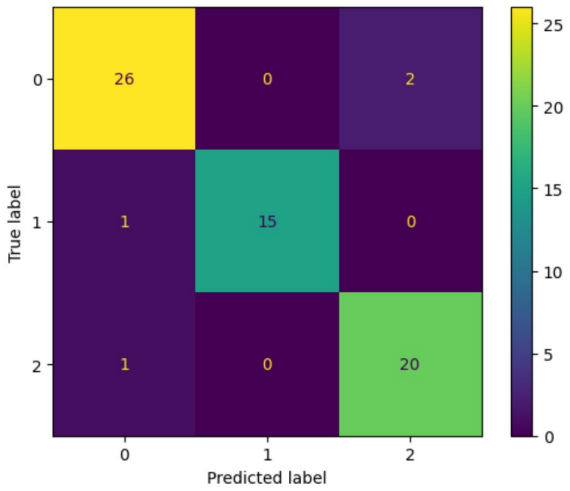


From the above graph, we can see that the best solving method for a multinomial linear regression is newton-cg. (note: liblinear and newton-cholesky are excluded because they do not work with multinomial regressions)

We can now use our model against the test data. We first fit the regression model to our training data. We then use the model on our test data, plotting the predicted vs actual values and calculating the accuracy score.

In [ ]:
```python
from sklearn.metrics import ConfusionMatrixDisplay
LR = LogisticRegression(multi_class='multinomial', solver='newton-cg', max_iter=10000)
LR.fit(X_train, y_train)

y_test_pred = LR.predict(X_test)
ConfusionMatrixDisplay.from_predictions(y_test, y_test_pred)
print("The accuracy score of the Logistic Regression model is: " + str(LR.score(X_test,y_test)))
```

```
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:466: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.10/dist-packages/scipy/optimize/_linesearch.py:314: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
The accuracy score of the Logistic Regression model is: 0.9384615384615385
```

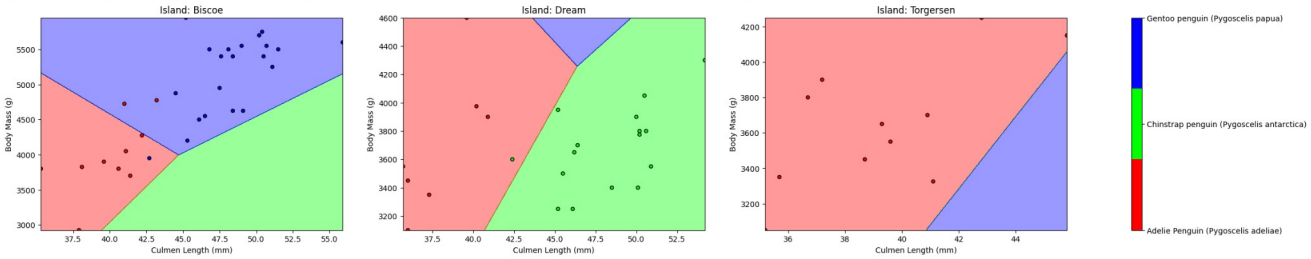


The above graph shows a confusion matrix. This matrix allows us to easily compare the predictions made to the actual values in our test data. The values in the diagonals (0,0, 1,1, 2,2) represent the correctly guessed values. We can see from the confusion matrix that our model only incorrectly predicted four values. We now have both a score for our model (~ .9385) and a matrix of what it got right and wrong. To better visualize its performance, we can use Decision Regions to visualize where the models succeed and fail. The below code will be reused for the other two models.

In [ ]:
```python
from matplotlib.colors import ListedColormap

island_dict = {name: label_encoder_island.transform([name])[0] for name in label_encoder_island.classes_}
species_dict = {name: label_encoder_species.transform([name])[0] for name in label_encoder_species.classes_}
# species_dict_inverse = {v: k for k, v in species_dict.items()}

print(species_dict)

def plot_regions(c, X, y):
    """
    Function to plot decision regions for a classification model c.

    Parameters:
    c : classifier object
        The classification model to visualize.
    X : DataFrame
        The feature data.
    y : Series
        The target data.
    island : str
        The island name to filter the data.
    is_train : bool
        Whether to use training or test data.

    Returns:
    None
    """

    # Create the plot
    fig, ax = plt.subplots(1, len(island_dict), figsize = (30, 5))

    for island_name, island_num in island_dict.items():
        island_indices = X.index[X['Island'] == island_num]
        island_X = X.loc[island_indices]
        island_y = y.loc[island_indices].reset_index()
        island_y = island_y['Species']


        x0 = island_X['Culmen Length (mm)']
        x1 = island_X['Body Mass (g)']

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

        XX = xx.ravel()
        YY = yy.ravel()
        XY = pd.DataFrame({
            'Culmen Length (mm)' : XX,
            'Body Mass (g)'  : YY,
            'Island' : island_num
        })


        p = c.predict(XY)
        p = p.reshape(xx.shape)

        colors = ['#FF0000', '#00FF00', '#0000FF']  # Red, Green, Blue
```

```
        c_map = ListedColormap(colors[:len(label_encoder_species.classes_)])



        contour = ax[island_num].contourf(xx, yy, p, cmap=c_map, alpha=0.4)

        # if island_name == 'Dream':
        #     colors = ['#FF0000', '#0000FF', '#00FF00']  # Red, Green, Blue
        #     c_map = ListedColormap(colors[:len(label_encoder_species.classes_)])
        # Plot the data
        scatter = ax[island_num].scatter(x0, x1, c=island_y, cmap=c_map, edgecolor='k', s=20)
        ax[island_num].set(xlabel="Culmen Length (mm)", ylabel="Body Mass (g)", title=f'Island: {island_name}')



    sm = plt.cm.ScalarMappable(cmap=c_map, norm=plt.Normalize(vmin=min(species_dict.values()), vmax=max(species_dict.values())))
    sm.set_array([])
    cbar = plt.colorbar(sm, ax=ax.ravel().tolist(), ticks=list(species_dict.values()))
    cbar.ax.set_yticklabels(list(species_dict.keys()))



    plt.show()

# Run programming for LR

plot_regions(LR, X_test, y_test)
```

```
{'Adelie Penguin (Pygoscelis adeliae)': 0, 'Chinstrap penguin (Pygoscelis antarctica)': 1, 'Gentoo penguin (Pygoscelis papua)': 2}
```

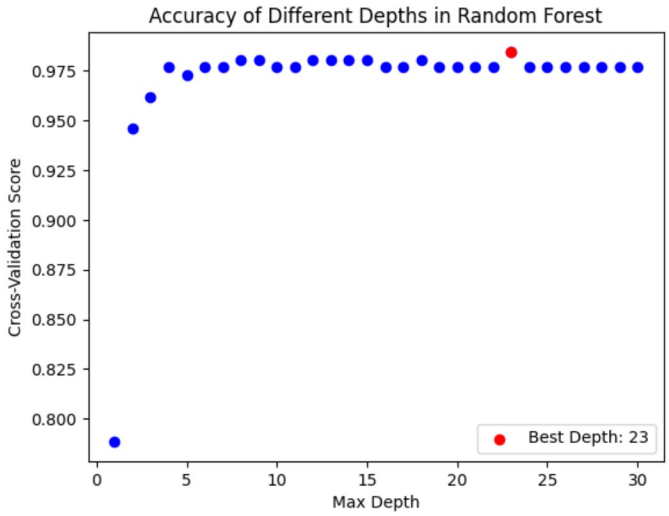


These charts are split by island (our qualitative data), with the axes being Culmen Length and Body Mass (our quantitative data). The bullet points represent the actual test values, while the background regions show what the model predicts for a certain combination of quantities. We can see that the model correctly identified which penguins are on which islands; for Biscoe, it only guessed Adelie and Chinstrap. For Dream, it only guessed Chinstrap and Adelie. For Torgersen, it only guessed Adelie. However, the decision regions reveal that the model still would guess other penguins on each island if they met certain quantitative criteria. We will revisit this after analyzing the other models.

Our next model is a Random Forest. As the name suggests, it is a combination of multiple decision trees. The forest asks all decision trees what outcome they predict and aggregates all these predictions to find the most likely result.

Our complexity parameter for a Random Forest is the maximum depth. This determines how many splits a tree makes from its root node, and the max depth will determine how many levels any of the tres in the forest can have. We cross-validate values between 1 and 30 to find an optimal maximum depth (with a reasonable runtime).

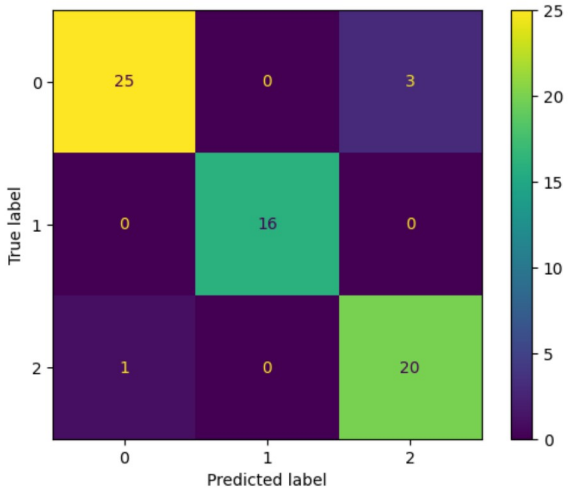
In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
plt.title('Accuracy of Different Depths in Random Forest')
plt.xlabel('Max Depth')
plt.ylabel('Cross-Validation Score')
plt.xticks(range(0, 31, 5))


depths = []
scores = []
bestDepth = None
bestScore = 0

for d in range(1, 31):
  #because of size of data set, increased iterations to 10000
    RF = RandomForestClassifier(max_depth = d)
    cvScore = np.mean(cross_val_score(RF, X_train, y_train, cv=10))
    if cvScore > bestScore:
      bestScore = cvScore
      bestDepth = d

    scores.append(cvScore)
    depths.append(d)

plt.scatter(depths, scores, color='Blue')
plt.scatter(bestDepth, bestScore, color='red', label=f'Best Depth: {bestDepth}')
plt.legend()
plt.show()
```



We find the best maximum depth for the trees in random forest. We can now repeat our previous process to get the confusion matrix and decision regions.

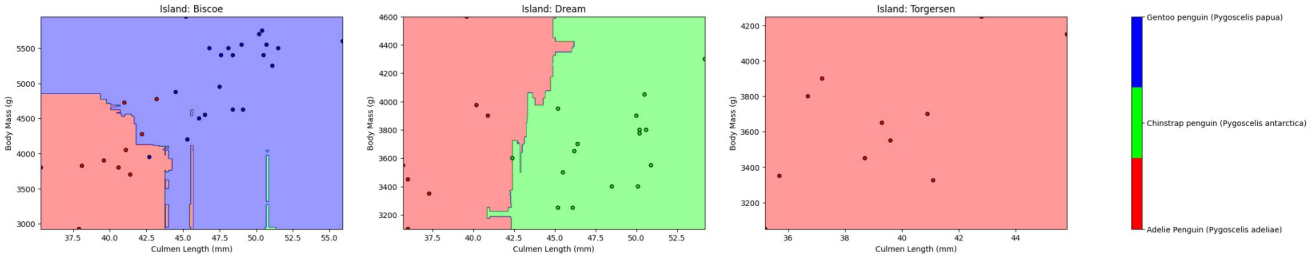
In [ ]:
```
RF = RandomForestClassifier(max_depth = bestDepth)
RF.fit(X_train, y_train)

y_test_pred = RF.predict(X_test)
ConfusionMatrixDisplay.from_predictions(y_test, y_test_pred)
print("The accuracy score of the Random Forest model is: " + str(RF.score(X_test,y_test)))
```

```
The accuracy score of the Random Forest model is: 0.9384615384615385
```



We see that the Random Forest model has a similar accuracy score to our previous model and also only got four predictions incorrect. We can gain further insight into the differences between the two with the decision regions.

In [ ]:
```
plot_regions(RF, X_test, y_test)
```

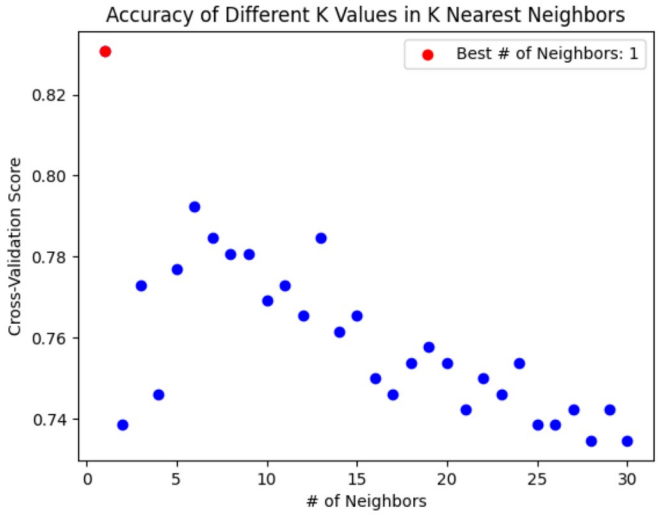


We see some of the strengths of Random Forest come into play here. Unlike Logistic Regression, the decision regions for penguins species are MUCH more accurate per island. Each island only contains regions for penguins that exist on it and the ratio of the regions is almost equal to the ratio of actual data points per species. While this did turn out to be a strength in our test data, this also makes the model much less flexible from its training data. For example, if the test data ended up containing a small population of Gentoo on Torgersen while none were present in the training data, this model would never be able to tell.

Our last model is a K Nearest Neighbors classifier. This data model relies on the assumption that similar data points will have similar labels. When predicting a data point, it will look at the K nearest data points and their labels to decide what the label of a data point should be. In this case, it would look at the species of nearby penguins. Before we train this model, we must decide how many neighbors it will use.

In [ ]:
```
from math import inf
```

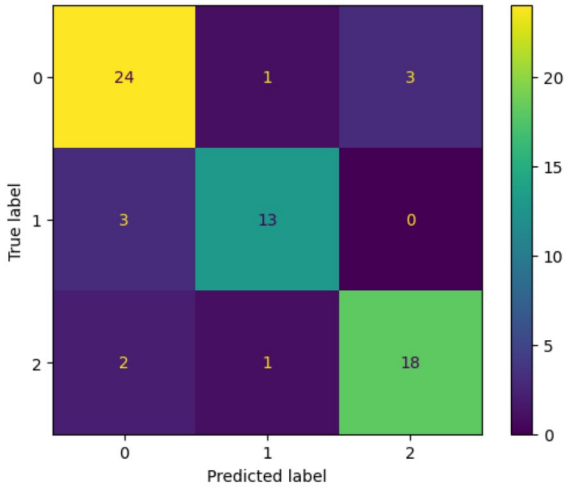
```
from sklearn.neighbors import KNeighborsClassifier
plt.title('Accuracy of Different K Values in K Nearest Neighbors')
plt.xlabel('# of Neighbors')
plt.ylabel('Cross-Validation Score')
plt.xticks([0, 5, 10, 15, 20, 25, 30])


neighbors = []
scores = []
bestk = None
bestScore = -inf

for k in range(1, 31):
    clf = KNeighborsClassifier(n_neighbors = k)
    cvScore = np.mean(cross_val_score(clf, X_train, y_train, cv=10))
    if cvScore > bestScore:
      bestScore = cvScore
      bestk = k

    scores.append(cvScore)
    neighbors.append(k)

plt.scatter(neighbors, scores, color='Blue')
plt.scatter(bestk, bestScore, color='red', label=f'Best # of Neighbors: {bestk}')
plt.legend()
plt.show()
```



We see that only choosing one neighbor is the best way to predict a point. We apply this to train our model and run it on the test data.

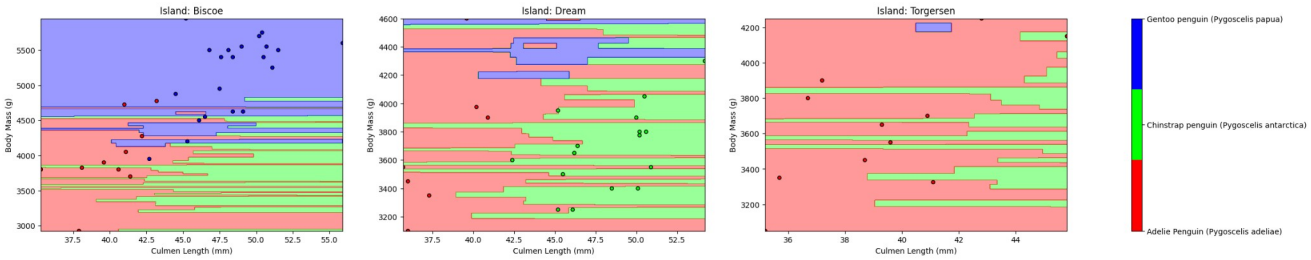
In [ ]:
```
clf = KNeighborsClassifier(n_neighbors = bestk)
clf.fit(X_train, y_train)

y_test_pred = clf.predict(X_test)
ConfusionMatrixDisplay.from_predictions(y_test, y_test_pred)
print("The accuracy score of the K Nearest Neighbors model is: " + str(clf.score(X_test,y_test)))
```

```
The accuracy score of the K Nearest Neighbors model is: 0.8461538461538461
```



This model was much less accurate than our others with nearly a 10% lower score (~.846). This is reflected in its confusion matrix, where it got 10 guesses wrong compared to the other two models' 4. We can see how this model struggled further in the decision regions.

In [ ]:
```
plot_regions(clf, X_test, y_test)
```



The erratic fits for the decision regions compared to our other models is likely because our cross validation chose only one neighbor for our model. This shows that KNN is not a good fit for our model. This could be because of large overlap between the quantitative data in species. For example, a female Gentoo might have similar measurements as a male Chinstrap (similar culmen length, smaller body mass).

Discussion

Overall, it seems that the Nearest Neighbor model performed the worst while the Logistic Regression and Random Forest models performed fairly well. As we found earlier, Island, Culmen Length, and Body Mass seem to be the best predictors of species, as there are the greatest differences in averages by species for these categories. This combination of quantitative features also showed great variance in averages between species. This allowed us to train models with over 93% accuracy, which is extremely high. However, this raises the concern for overfitting, as it is possible that if we had more data it could disagree in some ways. Specifically for the random forest, it classified all penguins on Torgersen as members of the Adelie species. While this worked for our data set, it may be possible to obtain later data sets with other penguins on this island, which would be incorrectly classified by the random forest model. Of course, there could be less obvious over fitting happening in the Logistic Regression model as well. If more data were avaialable, we would have a lower risk of over-fitting, which would help make the models more applicable to different data sets.

To conclude, we recommend using a Logistic Regression or Random Forest, with culmen length, body mass, and island as predictors of species.

In [ ]: