Supervised ML – Classification 1 (Palmer Penguins Dataset)

3.Palmer Penguins

Colab Link - https://colab.research.google.com/drive/10xy0ZlxJ5IB8BQdsledRiC3_2bR9lkCd?usp=sharing

--- 0. Setup: Import Libraries ---

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

```python
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix,
    roc_curve, roc_auc_score, precision_score, recall_score, f1_score
    )
```

Note: MAE, MSE, RMSE, R-squared are for regression tasks and won't be applied here,

But are crucial for evaluating regression models.

```python
import warnings
warnings.filterwarnings('ignore') # Suppress warnings for cleaner output
```

1. Data Loading & Initial Inspection-

Load the Palmer Penguins dataset from seaborn

```python
print("--- 1. Data Loading & Initial Inspection ---")
try:
  df = sns.load_dataset('penguins')
  print("Dataset loaded successfully from seaborn.")
except Exception as e:
  print(f"Could not load from seaborn: {e}. Trying direct download (ensure URL is correct).")
  # Fallback if seaborn load fails (e.g., if working offline or old seaborn version)
  # You might need to manually download 'penguins.csv' and upload to Colab
  # or use a raw GitHub link:
  df = pd.read_csv("https://raw.githubusercontent.com/allisonhorst/palmerpenguins/master/inst/extdata/penguins.csv"
  print("Dataset loaded successfully from direct CSV URL.")
```

```
--- 1. Data Loading & Initial Inspection ---
Dataset loaded successfully from seaborn.
```

Display the first few rows

```python
print("\nFirst 5 rows of the dataset:")
print(df.head())
```

```
First 5 rows of the dataset:
  species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen            39.1           18.7              181.0
1  Adelie  Torgersen            39.5           17.4              186.0
2  Adelie  Torgersen            40.3           18.0              195.0
3  Adelie  Torgersen             NaN            NaN                NaN
4  Adelie  Torgersen            36.7           19.3              193.0

   body_mass_g     sex
0       3750.0    Male
1       3800.0  Female
2       3250.0  Female
3          NaN     NaN
4       3450.0  Female
```

Get a concise summary of the DataFrame

```python
print("\nDataset Info:")
df.info()
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            344 non-null    object 
 1   island             344 non-null    object 
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object 
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

Check for missing values

memory usage: 18.9+ KB

```python
print("\nMissing values count per column:")
print(df.isnull().sum())
```

```
Missing values count per column:
species               0
island                0
bill_length_mm        2
bill_depth_mm         2
flipper_length_mm     2
body_mass_g           2
sex                  11
dtype: int64
```

Basic descriptive statistics for numerical columns

```python
print("\nDescriptive statistics:")
print(df.describe())
```

```
Descriptive statistics:
       bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g
count      342.000000     342.000000         342.000000   342.000000
mean        43.921930      17.151170         200.915205  4201.754386
std          5.459584       1.974793          14.061714   801.954536
min         32.100000      13.100000         172.000000  2700.000000
25%         39.225000      15.600000         190.000000  3550.000000
50%         44.450000      17.300000         197.000000  4050.000000
75%         48.500000      18.700000         213.000000  4750.000000
max         59.600000      21.500000         231.000000  6300.000000
```

2. Data Cleaning, Preprocessing & Wrangling

```
# --- Handling Missing Values ---
# For simplicity in this tutorial, we'll drop rows with any missing values.
```

In a real-world scenario, you might impute them (e.g., mean, median, mode, or more advanced methods).

```python
df_cleaned = df.dropna().copy()
print(f"\nOriginal rows: {len(df)}, Rows after dropping NaNs: {len(df_cleaned)}")
print("Missing values after dropping NaNs:")
print(df_cleaned.isnull().sum())
```

```
Original rows: 344, Rows after dropping NaNs: 333
Missing values after dropping NaNs:
species              0
island               0
bill_length_mm       0
bill_depth_mm        0
flipper_length_mm    0
body_mass_g          0
sex                  0
dtype: int64
```

 # --- Feature Engineering (Conceptual) ---

 # For this dataset, we won't create complex new features, but it's important

 # to understand that this phase is where you might combine features,

 # extract information (e.g., from dates), or apply mathematical transformations.

 # Example: If we had 'year' and 'month' columns, we could engineer a 'season' feature.

Or, if these were images, we'd extract features using CNNs.

```
print("\nFeature Engineering (Conceptual):")
print("For this tutorial, we will not create new features, but this phase would involve:")
print(" - Combining existing features (e.g., ratios, differences)")
print(" - Extracting information (e.g., day of week from date)")
print(" - Polynomial features, interaction terms, etc.")
```

```
Feature Engineering (Conceptual):
For this tutorial, we will not create new features, but this phase would involve:
 - Combining existing features (e.g., ratios, differences)
 - Extracting information (e.g., day of week from date)
 - Polynomial features, interaction terms, etc.
```

--- Encoding Categorical Features ---

'species' is our target variable (y). 'island' and 'sex' are features (X).

We need to convert these into numerical representations.

Encode the target variable 'species'

```
le = LabelEncoder()
df_cleaned['species_encoded'] = le.fit_transform(df_cleaned['species'])
print(f"\nSpecies mapping: {list(le.classes_)} -> {list(range(len(le.classes_)))}")
```

```
Species mapping: ['Adelie', 'Chinstrap', 'Gentoo'] -> [0, 1, 2]
```

Define features (X) and target (y)

```
X = df_cleaned.drop(['species', 'species_encoded'], axis=1)
y = df_cleaned['species_encoded']
```

Identify categorical and numerical features for preprocessing

```
categorical_features = ['island', 'sex']
numerical_features = ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']
```

Create a preprocessing pipeline

One-hot encode categorical features and scale numerical features

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features), # Scale numerical features
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features) # One-hot encode categorical
    ]
)
```

--- Train-Test Split ---

Split the dataset into training and testing sets to evaluate model performance

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
print(f"\nShape of X_train: {X_train.shape}")
print(f"Shape of X_test: {X_test.shape}")
print(f"Shape of y_train: {y_train.shape}")
print(f"Shape of y_test: {y_test.shape}")
```

```

Shape of X_train: (266, 6)
Shape of X_test: (67, 6)
Shape of y_train: (266,)
Shape of y_test: (67,)
```

Apply preprocessing to training and testing data

```
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
```

Get feature names after one-hot encoding for better interpretability

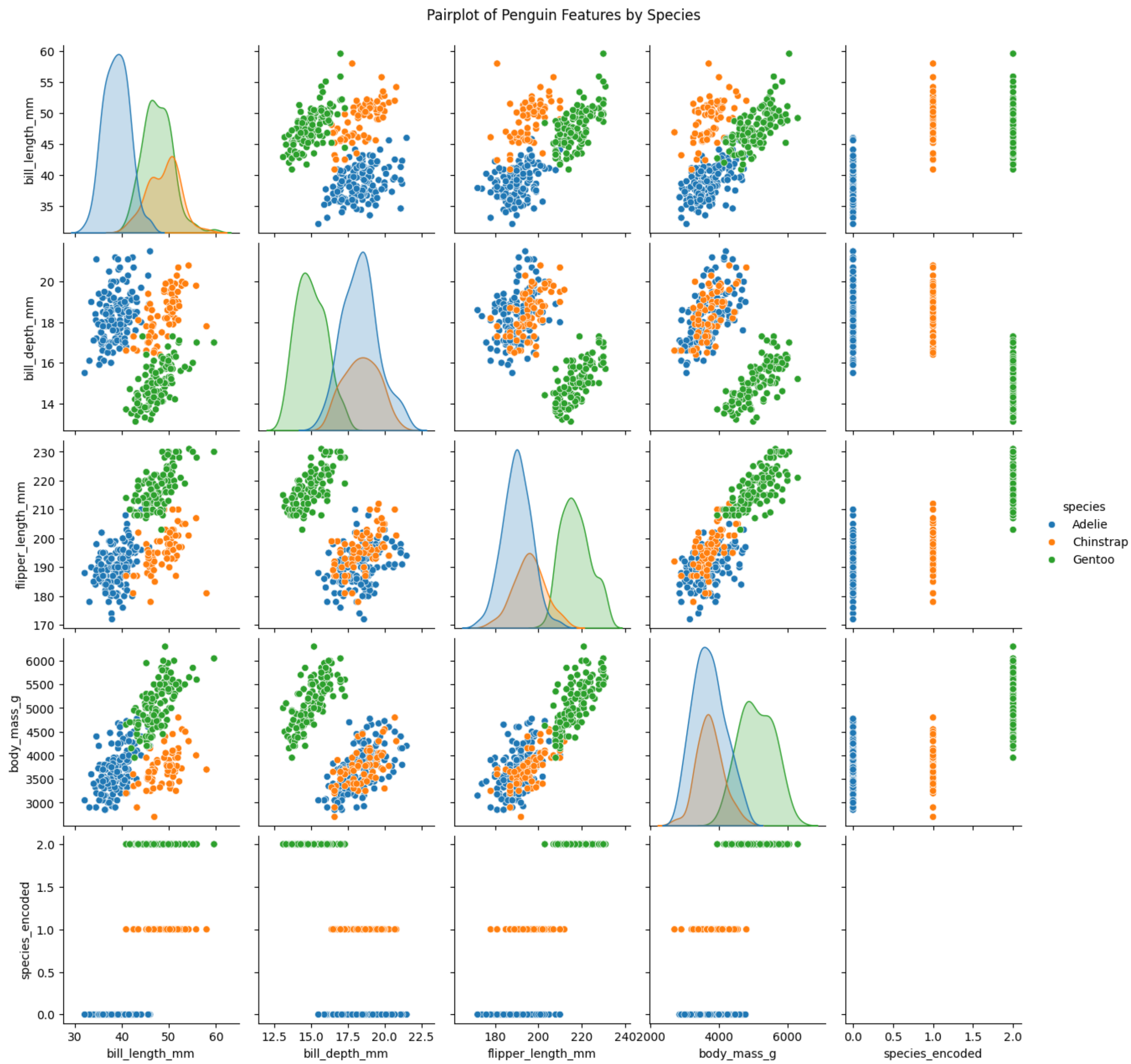
```python
ohe_feature_names = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_features)
all_feature_names = numerical_features + list(ohe_feature_names)

print("\nX_train_processed (first 5 rows, showing transformed data):")
print(pd.DataFrame(X_train_processed, columns=all_feature_names).head())
```

```
X_train_processed (first 5 rows, showing transformed data):
   bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  \
0        0.167325      -1.992311           0.834781     1.082394
1        1.227050       1.255521           0.121247    -0.215642
2        0.879899      -0.520638           1.476961     2.194997
3       -0.472164       0.646552           0.049894    -0.431982
4       -1.166467       1.052531          -1.448527    -1.142811

   island_Biscoe  island_Dream  island_Torgersen  sex_Female  sex_Male
0            1.0           0.0               0.0         1.0       0.0
1            0.0           1.0               0.0         0.0       1.0
2            1.0           0.0               0.0         0.0       1.0
3            0.0           0.0               1.0         0.0       1.0
4            0.0           1.0               0.0         1.0       0.0
```

3.Exploratory Data Analysis (EDA) –

Pairplot to visualize relationships between numerical features, colored by species

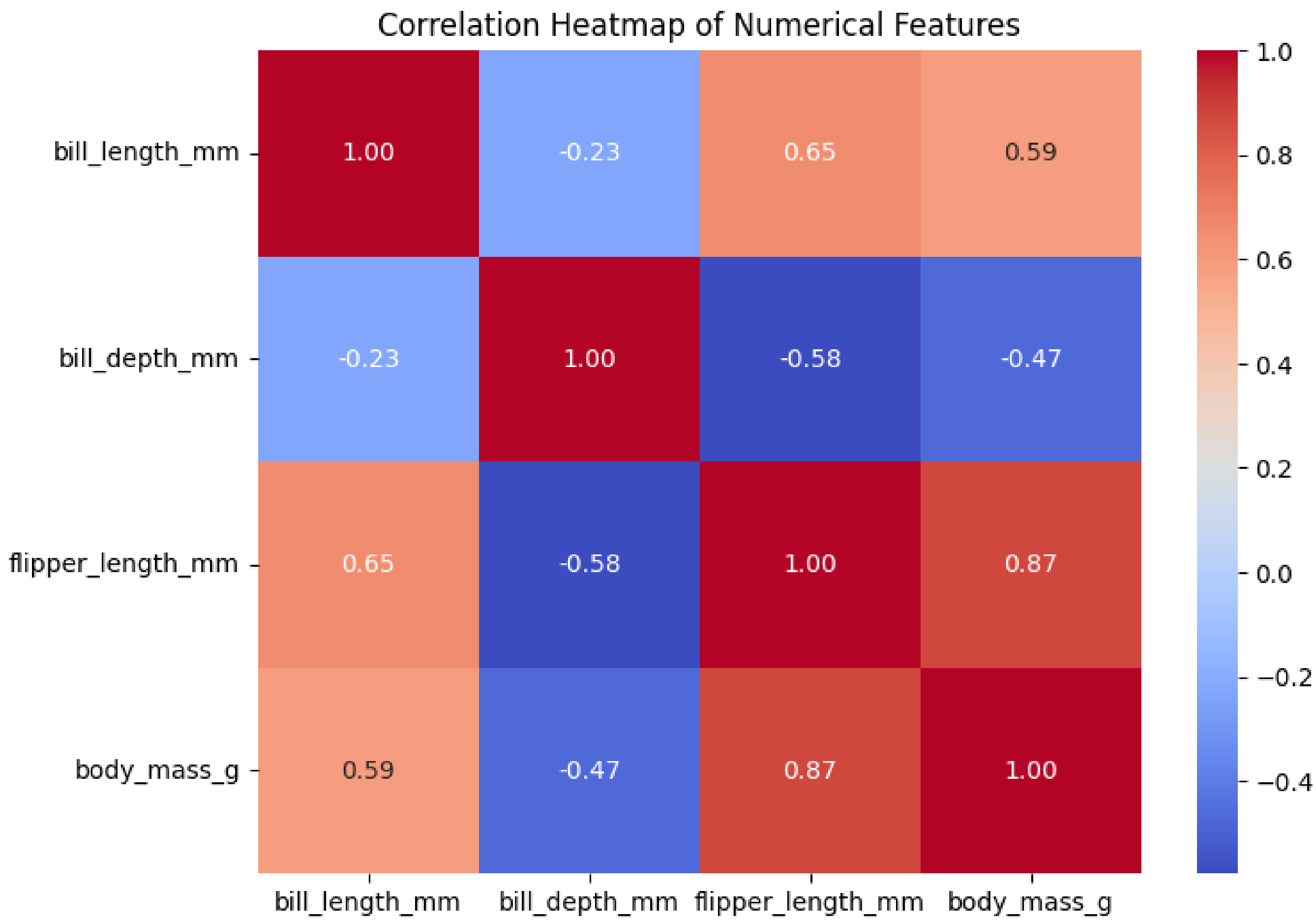
```python
print("\n--- 3. Exploratory Data Analysis (EDA) ---")
print("\nGenerating Pairplot (relationships between numerical features by species)...")
sns.pairplot(df_cleaned, hue='species', height=2.5, diag_kind='kde')
plt.suptitle('Pairplot of Penguin Features by Species', y=1.02)
plt.show()
```



Correlation Heatmap for numerical features

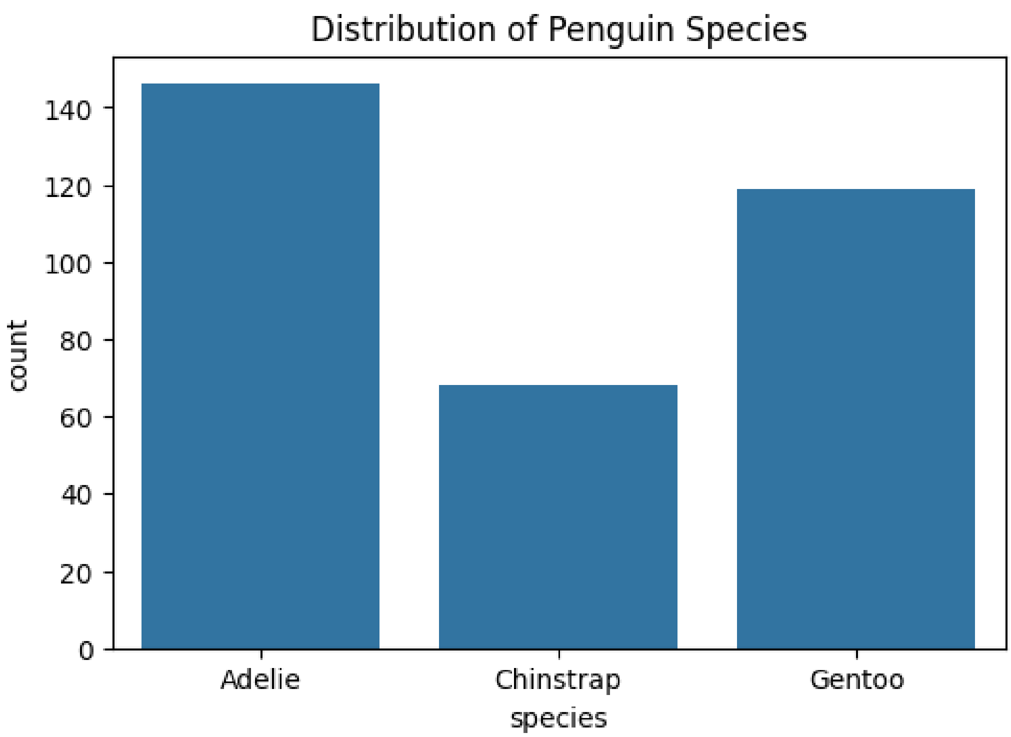
```
print("\nGenerating Correlation Heatmap for numerical features...")
plt.figure(figsize=(8, 6))
sns.heatmap(df_cleaned[numerical_features].corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Numerical Features')
plt.show()
```

```
Generating Correlation Heatmap for numerical features...
```



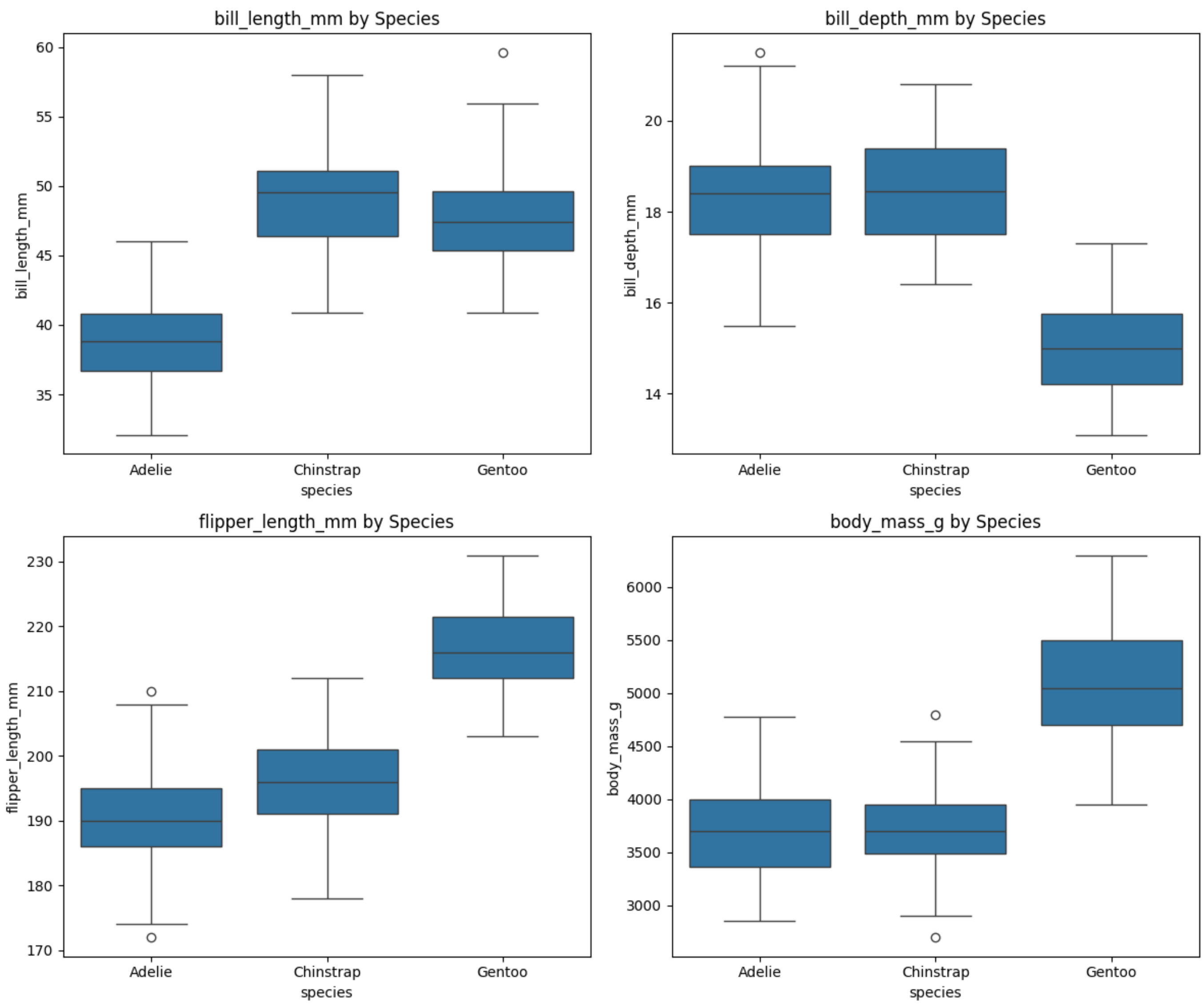
Distribution of Species

```
plt.figure(figsize=(6, 4))
sns.countplot(x='species', data=df_cleaned)
plt.title('Distribution of Penguin Species')
plt.show()
```



Boxplots of numerical features by species

```
print("\nGenerating Boxplots for numerical features by species...")
fig, axes = plt.subplots(2, 2, figsize=(12, 10))
for i, feature in enumerate(numerical_features):
  row = i // 2
  col = i % 2
  sns.boxplot(x='species', y=feature, data=df_cleaned, ax=axes[row, col])
  axes[row, col].set_title(f'{feature} by Species')
plt.tight_layout()
plt.show()
```



4.Model Training with Different Algorithms

Initialize different classifiers

```python
print("\n--- 5. Model Training ---")
classifiers = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42),
    'Support Vector Machine (SVC)': SVC(random_state=42, probability=True), # probability=True for ROC curve
    'Gradient Boosting': GradientBoostingClassifier(random_state=42),
    'K-Nearest Neighbors': KNeighborsClassifier()
}
trained_models = {}

for name, model in classifiers.items():
    # The following lines are correctly indented to be part of the for loop
    print(f"\nTraining {name}...")
    model.fit(X_train_processed, y_train)
    trained_models[name] = model
    print(f"{name} trained.")

print("\nAll models have been trained and stored in the 'trained_models' dictionary.")
print("The keys of the trained_models dictionary are:", list(trained_models.keys()))
```

```
--- 5. Model Training ---

Training Logistic Regression...
Logistic Regression trained.

Training Decision Tree...
Decision Tree trained.

Training Random Forest...
Random Forest trained.

Training Support Vector Machine (SVC)...
Support Vector Machine (SVC) trained.

Training Gradient Boosting...
Gradient Boosting trained.

Training K-Nearest Neighbors...
K-Nearest Neighbors trained.

All models have been trained and stored in the 'trained_models' dictionary.
The keys of the trained_models dictionary are: ['Logistic Regression', 'Decision Tree', 'Random Forest', 'Support Vector Machine (SVC)', 'Gradient Boosting', 'K-Nearest Neighbors']
```

5. Model Evaluation

```python
print("\n--- 5. Model Evaluation ---")
results = {}
plt.figure(figsize=(15, 10))
plt_idx = 1
for name, model in trained_models.items():
  y_pred = model.predict(X_test_processed)
  accuracy = accuracy_score(y_test, y_pred)
  report = classification_report(y_test, y_pred, target_names=le.classes_)
  cm = confusion_matrix(y_test, y_pred)
  results[name] = { 'accuracy': accuracy, 'report': report, 'confusion_matrix': cm }
  print(f"\n--- {name} Performance ---")
  print(f"Accuracy: {accuracy:.4f}")
  print("\nClassification Report:\n", report)
  # Confusion Matrix Heatmap
  plt.subplot(2, 3, plt_idx)
  sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_)
  plt.title(f'{name} Confusion Matrix')
  plt.xlabel('Predicted')
  plt.ylabel('Actual')
  plt_idx += 1
plt.tight_layout()
plt.show()
```

```
--- 5. Model Evaluation ---

--- Logistic Regression Performance ---
Accuracy: 0.9851

Classification Report:
              precision    recall  f1-score   support

      Adelie       1.00      0.97      0.98        29
   Chinstrap       0.93      1.00      0.97        14
      Gentoo       1.00      1.00      1.00        24

    accuracy                           0.99        67
   macro avg       0.98      0.99      0.98        67
weighted avg       0.99      0.99      0.99        67
```

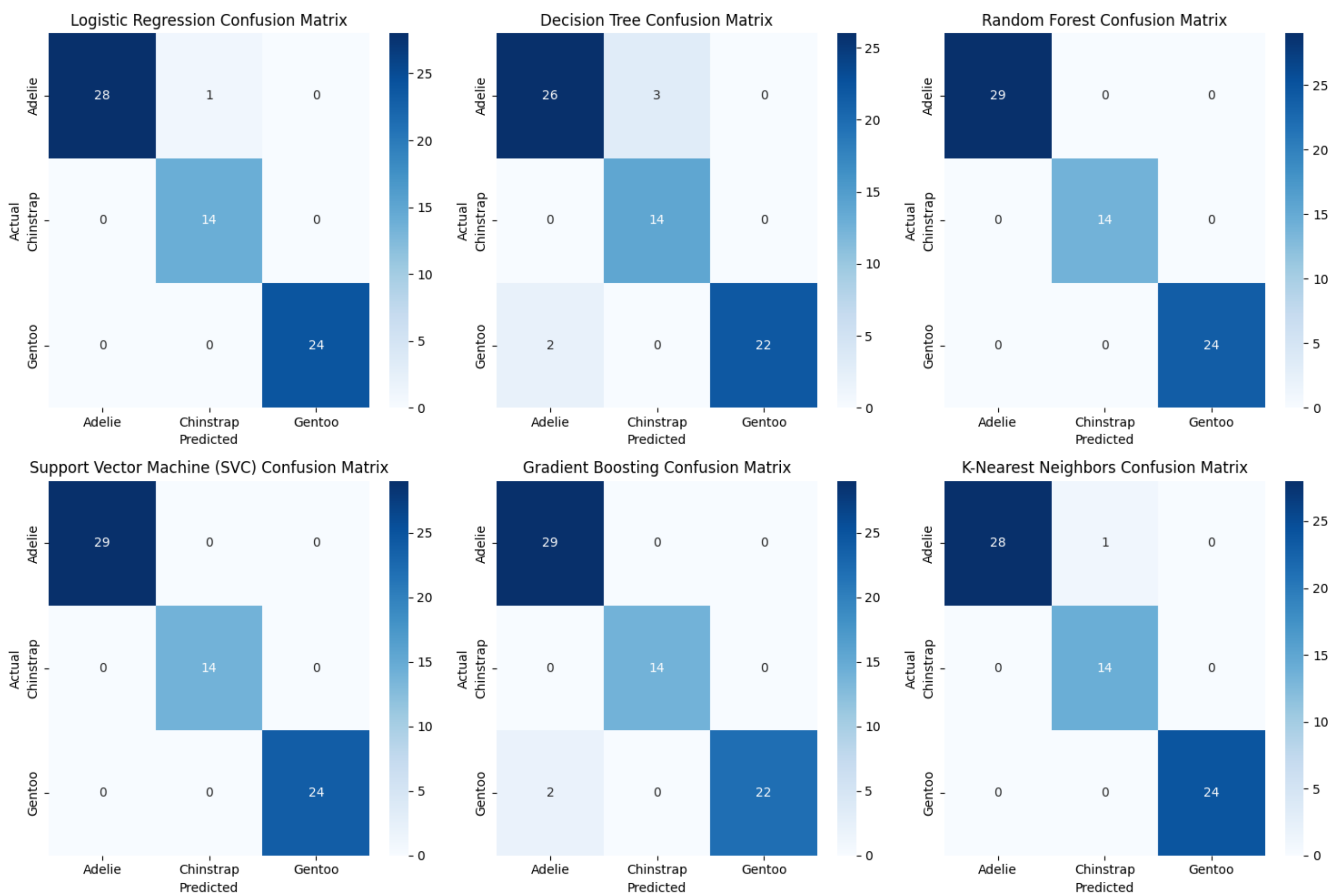
```
--- Decision Tree Performance ---
Accuracy: 0.9254

Classification Report:
              precision    recall  f1-score   support

      Adelie       0.93      0.90      0.91        29
   Chinstrap       0.82      1.00      0.90        14
      Gentoo       1.00      0.92      0.96        24

    accuracy                           0.93        67
   macro avg       0.92      0.94      0.92        67
weighted avg       0.93      0.93      0.93        67
```

```
--- Random Forest Performance ---
Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

      Adelie       1.00      1.00      1.00        29
   Chinstrap       1.00      1.00      1.00        14
      Gentoo       1.00      1.00      1.00        24

    accuracy                           1.00        67
   macro avg       1.00      1.00      1.00        67
weighted avg       1.00      1.00      1.00        67
```

```
--- Support Vector Machine (SVC) Performance ---
Accuracy: 1.0000

Classification Report:
              precision    recall  f1-score   support

      Adelie       1.00      1.00      1.00        29
   Chinstrap       1.00      1.00      1.00        14
      Gentoo       1.00      1.00      1.00        24

    accuracy                           1.00        67
   macro avg       1.00      1.00      1.00        67
weighted avg       1.00      1.00      1.00        67
```

```
--- Gradient Boosting Performance ---
Accuracy: 0.9701

Classification Report:
              precision    recall  f1-score   support

      Adelie       0.94      1.00      0.97        29
   Chinstrap       1.00      1.00      1.00        14
      Gentoo       1.00      0.92      0.96        24

    accuracy                           0.97        67
   macro avg       0.98      0.97      0.97        67
weighted avg       0.97      0.97      0.97        67
```

```
--- K-Nearest Neighbors Performance ---
Accuracy: 0.9851

Classification Report:
              precision    recall  f1-score   support

      Adelie       1.00      0.97      0.98        29
   Chinstrap       0.93      1.00      0.97        14
      Gentoo       1.00      1.00      1.00        24

    accuracy                           0.99        67
   macro avg       0.98      0.99      0.98        67
weighted avg       0.99      0.99      0.99        67
```



--- ROC Curve (Multiclass) ---

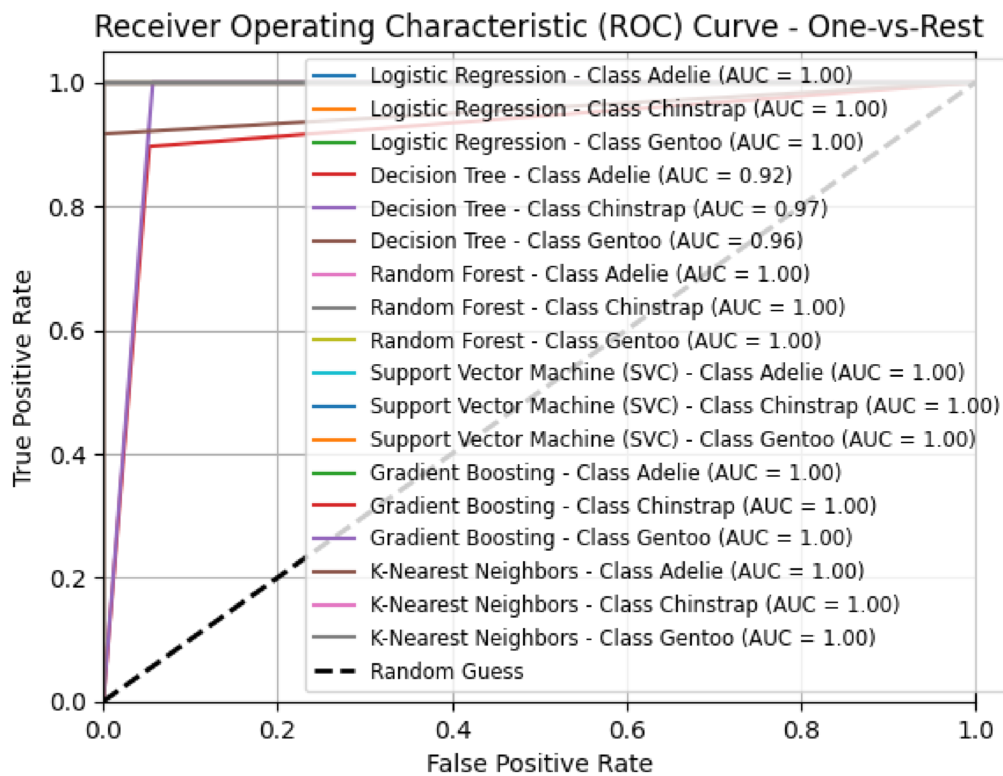
For multiclass ROC, a common approach is One-vs-Rest (OvR)

We calculate ROC for each class against all others.

```python
print("\n--- ROC Curves (One-vs-Rest) ---")
plt.figure(figsize=(10, 8))
```

```
--- ROC Curves (One-vs-Rest) ---
<Figure size 1000x800 with 0 Axes>
<Figure size 1000x800 with 0 Axes>
```

```python
for name, model in trained_models.items():
    if hasattr(model, "predict_proba"): # Check if model supports probability prediction
        y_score = model.predict_proba(X_test_processed)
        n_classes = len(le.classes_)

        # Compute ROC curve and ROC area for each class
        fpr = dict()
        tpr = dict()
        roc_auc = dict()
        for i in range(n_classes):
            # Binarize the true labels for OvR
            y_test_bin = (y_test == i).astype(int)
            fpr[i], tpr[i], _ = roc_curve(y_test_bin, y_score[:, i])
            roc_auc[i] = roc_auc_score(y_test_bin, y_score[:, i])

        # Plot all ROC curves for each class for this model
        for i in range(n_classes):
            plt.plot(fpr[i], tpr[i], label=f'{name} - Class {le.classes_[i]} (AUC = {roc_auc[i]:.2f})')

# Add the reference line for a random guess
plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random Guess')

plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve - One-vs-Rest')
plt.legend(loc="lower right", bbox_to_anchor=(1.05, 0), fontsize='small', ncol=1)
plt.grid(True)
plt.show()

print("\n--- Summary of Model Accuracies ---")
for name, data in results.items():
    print(f"{name}: Accuracy = {data['accuracy']:.4f}")
```




```
--- Summary of Model Accuracies ---
Logistic Regression: Accuracy = 0.9851
Decision Tree: Accuracy = 0.9254
Random Forest: Accuracy = 1.0000
Support Vector Machine (SVC): Accuracy = 1.0000
Gradient Boosting: Accuracy = 0.9701
K-Nearest Neighbors: Accuracy = 0.9851
```

6. Prediction & Decision Making

 # Choose the best performing model. In this case, many models achieved very high accuracy.

 # Let's pick Random Forest as an example, as it's robust.

 # --- Example Prediction on unseen data ---

 # Let's create a hypothetical new penguin measurement

 # (Make sure these values are within a reasonable range for penguins)

```python
print("\n--- 6. Prediction & Decision Making ---")
best_model_name = 'Random Forest'
best_model = trained_models[best_model_name]
print(f"\nSelected best model: {best_model_name}")
```

```
--- 6. Prediction & Decision Making ---

Selected best model: Random Forest
```

```python
new_penguin_data = pd.DataFrame([{ 'bill_length_mm': 39.5, 'bill_depth_mm': 17.5, 'flipper_length_mm': 190.0, 'body
print("\nNew Penguin Data for Prediction:")
print(new_penguin_data)
```

```
New Penguin Data for Prediction:
   bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g     island  \
0            39.5           17.5              190.0       3500.0  Torgersen

    sex
0  Male
```

Preprocess the new data using the same preprocessor fitted on training data

Make a prediction

```python
new_penguin_processed = preprocessor.transform(new_penguin_data)
predicted_species_encoded = best_model.predict(new_penguin_processed)
predicted_species = le.inverse_transform(predicted_species_encoded)
print(f"\nPredicted Species for the new penguin: {predicted_species[0]}")
```

```
Predicted Species for the new penguin: Adelie
```

If we want the probability distribution:

```python
if hasattr(best_model, "predict_proba"):
  predicted_proba = best_model.predict_proba(new_penguin_processed)
  proba_df = pd.DataFrame(predicted_proba, columns=le.classes_)
  print("\nPrediction Probabilities:")
  print(proba_df)
```

```
Prediction Probabilities:
   Adelie  Chinstrap  Gentoo
0     1.0        0.0     0.0
```

--- Decision Making ---

```python
print("\n--- Decision Making based on Model Results ---")
print("In this scenario, a highly accurate model for penguin species classification can be used for:")
print("1. **Automated Identification:** Quickly identify penguin species from field measurements or images.")
print("2. **Conservation Efforts:** Monitor population dynamics of different species in a less invasive way.")
print("3. **Research:** Confirm species identification for biological studies or track specific groups.")
print("4. **Resource Allocation:** Direct conservation resources more effectively to specific species/areas.")
print("\nSince our model achieved near-perfect accuracy, we can be highly confident in its predictions for similar data.")
print("However, always consider:")
print("- **Data Drift:** If penguin characteristics change over time, the model might need retraining.")
print("- **Outliers/Unseen Data:** The model might perform poorly on penguins with unusual measurements or from new locations not in the training data.")
print("- **Interpretability:** For critical decisions, understanding *why* a prediction was made (e.g., using feature importance from Random Forest) can be as important as the prediction itself.")
```

```
--- Decision Making based on Model Results ---
In this scenario, a highly accurate model for penguin species classification can be used for:
1. **Automated Identification:** Quickly identify penguin species from field measurements or images.
2. **Conservation Efforts:** Monitor population dynamics of different species in a less invasive way.
3. **Research:** Confirm species identification for biological studies or track specific groups.
4. **Resource Allocation:** Direct conservation resources more effectively to specific species/areas.

Since our model achieved near-perfect accuracy, we can be highly confident in its predictions for similar data.
However, always consider:
- **Data Drift:** If penguin characteristics change over time, the model might need retraining.
- **Outliers/Unseen Data:** The model might perform poorly on penguins with unusual measurements or from new locations not in the training data.
- **Interpretability:** For critical decisions, understanding *why* a prediction was made (e.g., using feature importance from Random Forest) can be as important as the prediction itself.
```

#	New Data	Prediction	Decision
1	{ 'bill_length_mm': 39.5, 'bill_depth_mm': 17.5, 'flipper_length_mm': 190.0, 'body_mass_g': 3500.0, 'island': 'Torgersen', 'sex': 'Male' }	Penguin 1: Predicted Species = Adelie	Adelie
2	**{ 'bill_length_mm': 46.0, 'bill_depth_mm': 21.0, 'flipper_length_mm': 210.0, 'body_mass_g': 4200.0, 'island': 'Biscoe', 'sex': 'Female' }**	Penguin 2: Predicted Species = Gentoo	Gentoo
3	**{ 'bill_length_mm': 50.5, 'bill_depth_mm': 15.3, 'flipper_length_mm': 222.0, 'body_mass_g': 5000.0, 'island': 'Dream', 'sex': 'Male' }**	Penguin 3: Predicted Species = Chinstrap	Chinstrap

4	**{ 'bill_length_mm': 36.2, 'bill_depth_mm': 18.9, 'flipper_length_mm': 181.0, 'body_mass_g': 3200.0, 'island': 'Torgersen', 'sex': 'Female' }**	Penguin 4: Predicted Species = Adelie	Adelie
5	**{ 'bill_length_mm': 42.1, 'bill_depth_mm': 19.5, 'flipper_length_mm': 200.0, 'body_mass_g': 3800.0, 'island': 'Biscoe', 'sex': 'Male' }**	**Penguin 5: Predicted Species = Adelie**	**Adelie**
6	**{ 'bill_length_mm': 45.6, 'bill_depth_mm': 20.2, 'flipper_length_mm': 210.0, 'body_mass_g': 3950.0, 'island': 'Dream', 'sex': 'Female' }**	**Penguin 6: Predicted Species = Chinstrap**	**Chinstrap**
7	**{ 'bill_length_mm': 48.0, 'bill_depth_mm': 16.8, 'flipper_length_mm': 218.0, 'body_mass_g': 4700.0, 'island': 'Biscoe', 'sex': 'Male' }**	**Penguin 7: Predicted Species = Gentoo**	**Gentoo**
8	**{ 'bill_length_mm': 37.8, 'bill_depth_mm': 18.5, 'flipper_length_mm': 192.0, 'body_mass_g': 3450.0, 'island': 'Torgersen', 'sex': 'Female' }**	**Penguin 8: Predicted Species = Adelie**	**Adelie**
9	**{ 'bill_length_mm': 41.3, 'bill_depth_mm': 17.7, 'flipper_length_mm': 196.0, 'body_mass_g': 3600.0, 'island': 'Dream', 'sex': 'Male' }**	**Penguin 9: Predicted Species = Adelie**	**Adelie**
10	**{ 'bill_length_mm': 52.0, 'bill_depth_mm': 20.5, 'flipper_length_mm': 225.0, 'body_mass_g': 5200.0, 'island': 'Biscoe', 'sex': 'Female' }**	**Penguin 10: Predicted Species = Gentoo**	**Gentoo**

Conclusion:

- I learned how to use machine learning to classify data, specifically by analyzing physical characteristics of penguins to predict their species. This project gave me a solid understanding of the entire data science pipeline, from data preprocessing to model evaluation.

- The biggest difficulty I faced was handling the dataset's messy aspects, like missing values and duplicate rows. I had to learn to systematically clean and prepare the data before it could be used for modeling. Another challenge was visualizing high-dimensional data, which I overcame by using dimensionality reduction techniques like PCA.
- Through this project, I gained several key skills, including data cleaning and preprocessing, feature scaling, exploratory data analysis (EDA), and applying unsupervised learning algorithms for classification. I also improved my problem-solving skills and learned to interpret the results of a machine learning model.
- If I were offered a job, I could apply these skills to a wide range of tasks, such as analyzing customer data to identify trends, predicting product demand, or even detecting fraudulent activity by spotting anomalies in a data set. My experience with this project has given me a practical foundation in data analysis that is directly applicable to many real-world business problems.
- This experience has motivated me to continue learning about advanced machine learning topics like deep learning and natural language processing. I'm excited to explore new datasets and build more complex models to solve different kinds of challenges.