In [1]:
```python
# This Code Does an Import of a CSV file an alternative may be an excel file
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import StandardScaler
pd.options.mode.chained_assignment = None

import warnings
warnings.filterwarnings('ignore')


#Phase 1 collecting the data
pd.set_option("expand_frame_repr", False) #Avoids Printing on the next line
df= pd.read_csv('C:/Users/Marc/Dropbox/University of Pretoria/791/Cheat Shee
df.columns =["species","island", "bill_length_mm", "bill_depth_mm", "flipper
df
```

Out[1]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	bo
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	
...	...	...	...	...	...	
339	Gentoo	Biscoe	NaN	NaN	NaN	
340	Gentoo	Biscoe	46.8	14.3	215.0	
341	Gentoo	Biscoe	50.4	15.7	222.0	
342	Gentoo	Biscoe	45.2	14.8	212.0	
343	Gentoo	Biscoe	49.9	16.1	213.0	

344 rows × 7 columns

In [2]:
```python
df.shape
```

Out[2]: (344, 7)

In [3]:
```python
unique_values = df['sex'].unique()
print(unique_values)
df.shape
```

```
['MALE' 'FEMALE' nan '.']
```

Out[3]: (344, 7)

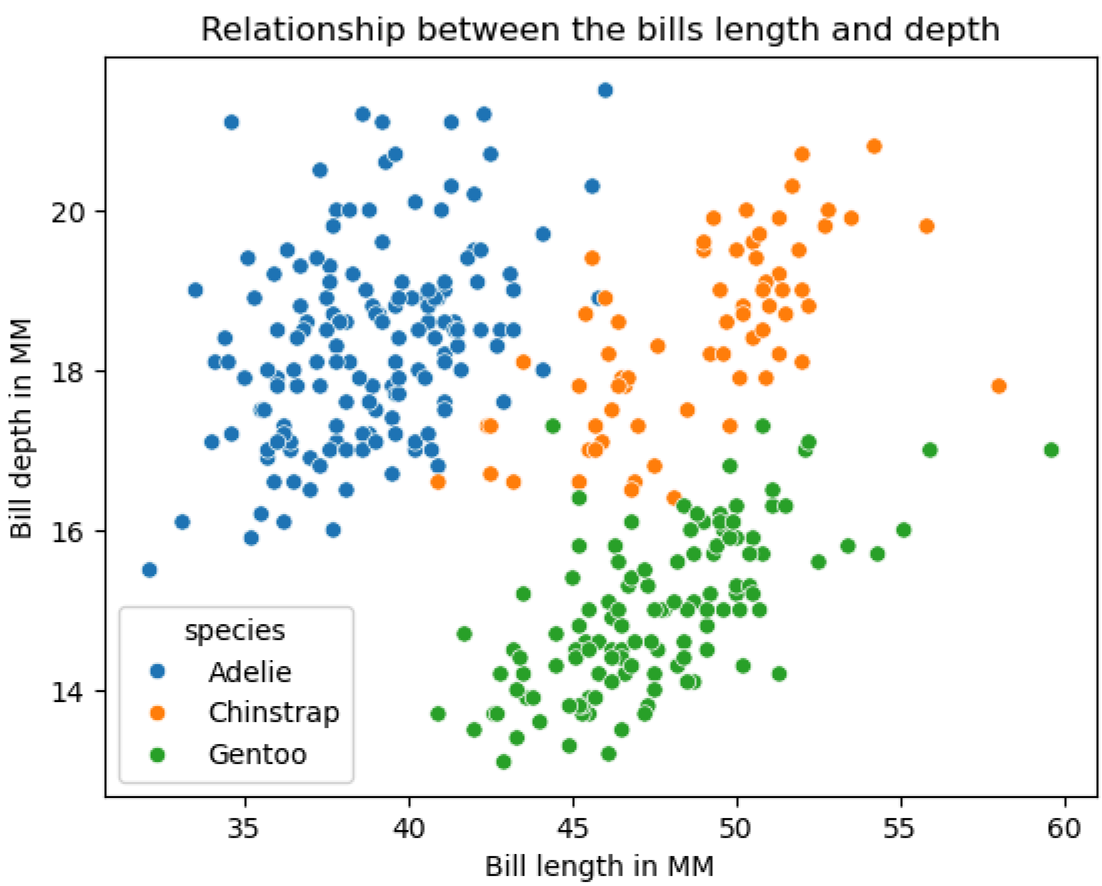
In [4]:
```
df = df[df['sex'] != '.']
print(df.shape)
```

```
(343, 7)
```

In [5]:
```
# sns.set_theme()
# # For the image quality of the graphic.
# sns.set(rc={"figure.dpi":300})
# # For the size of the graphics
# sns.set(rc = {"figure.figsize":(6,3)})
```

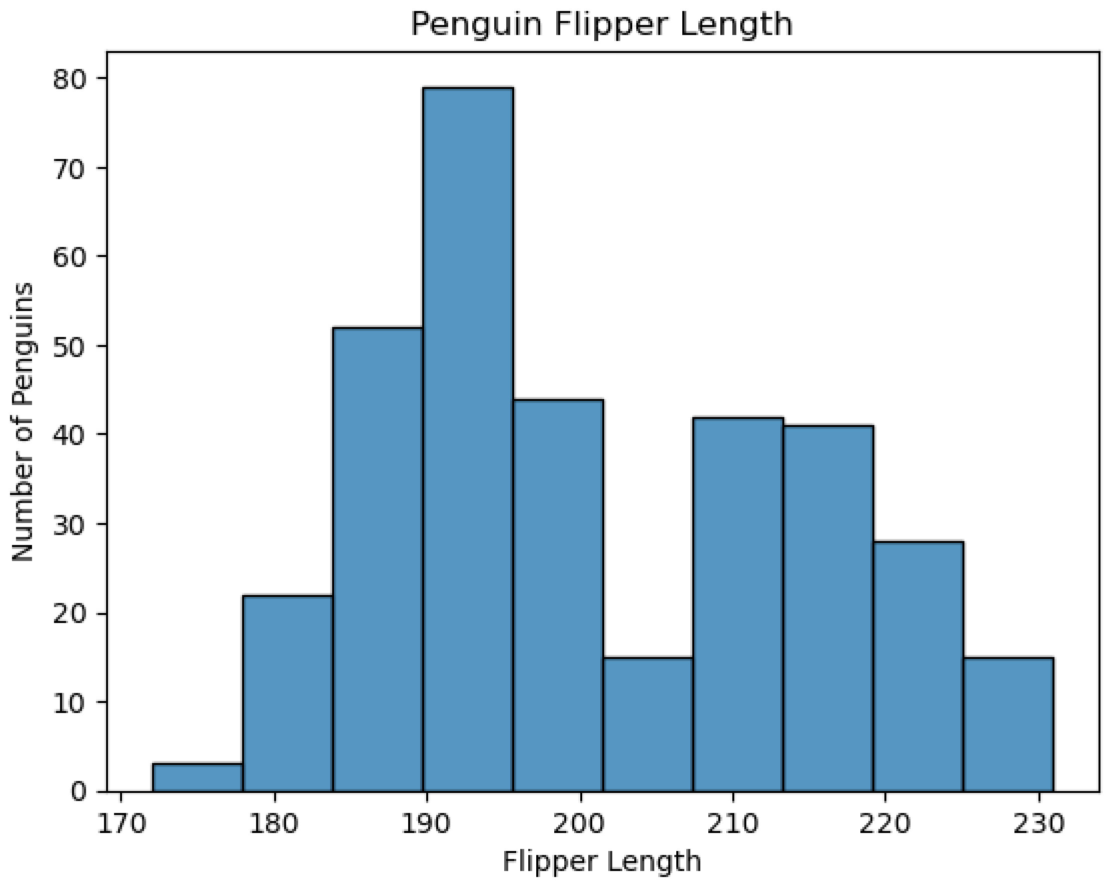
In [6]:
```
#Drawing a scatter plot, just note that you can have a scatter plot with tte
sns.scatterplot( x = "bill_length_mm",
                 y = "bill_depth_mm",
                 data = df,
                 hue = "species")
plt.title("Relationship between the bills length and depth")
plt.xlabel("Bill length in MM")
plt.ylabel("Bill depth in MM")
```

Out[6]:
```
Text(0, 0.5, 'Bill depth in MM')
```



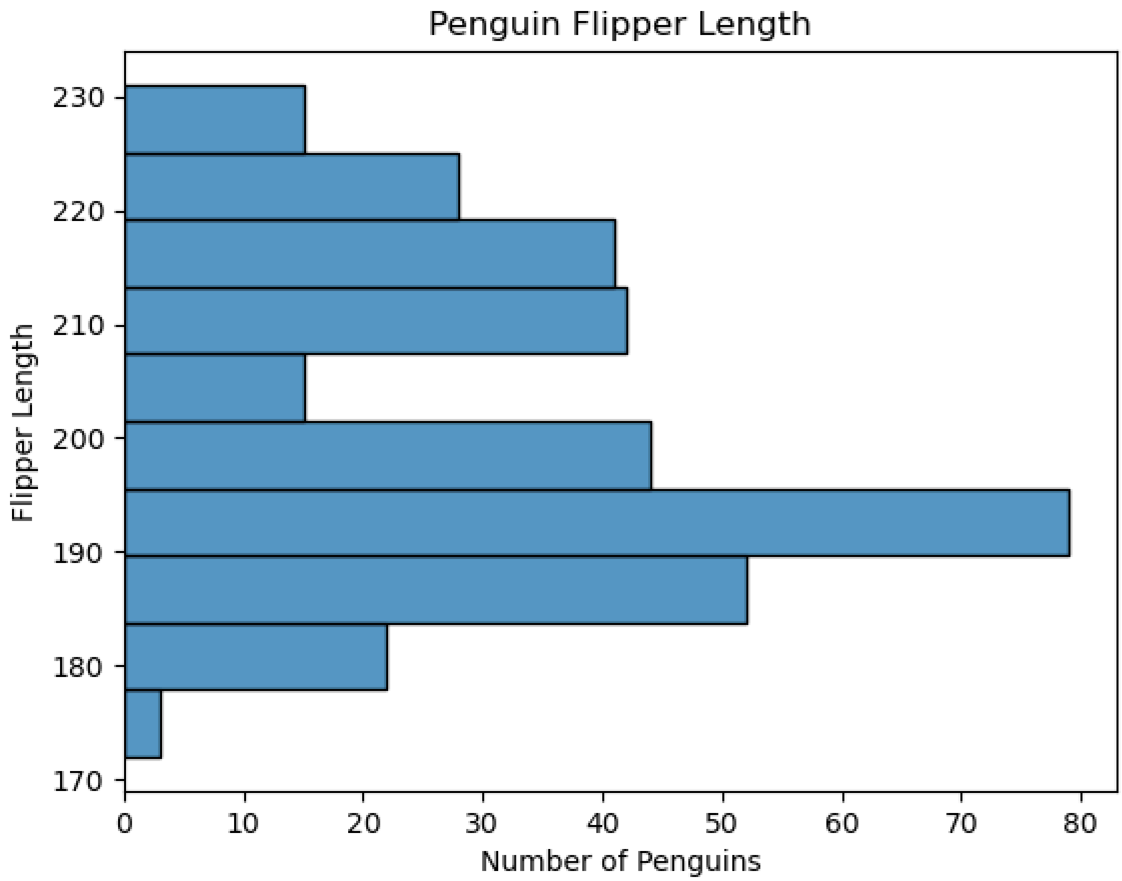
In [7]:
```
sns.histplot(x = "flipper_length_mm", data = df)
plt.title("Penguin Flipper Length")
plt.xlabel("Flipper Length")
plt.ylabel("Number of Penguins")
```

Out[7]:
```
Text(0, 0.5, 'Number of Penguins')
```



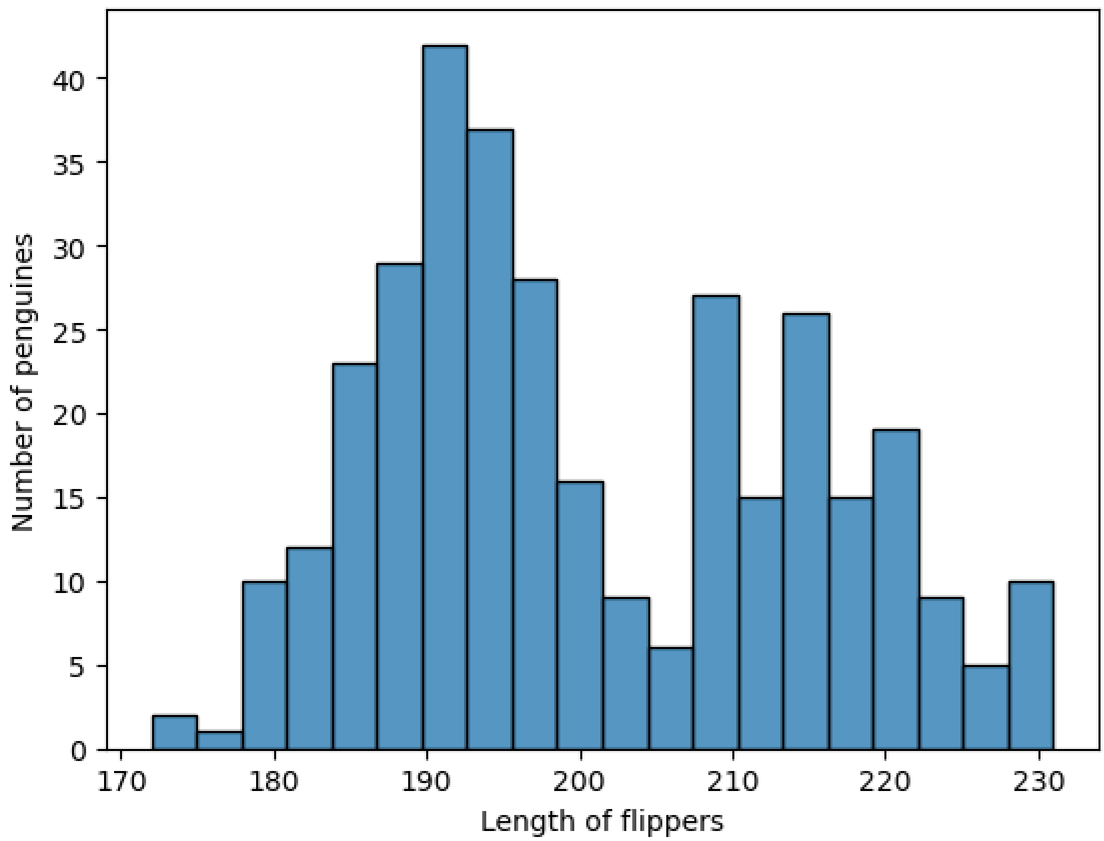
In [8]:
```
sns.histplot(y = "flipper_length_mm", data = df)
plt.title("Penguin Flipper Length")
plt.ylabel("Flipper Length")
plt.xlabel("Number of Penguins")
```

Out[8]:
```
Text(0.5, 0, 'Number of Penguins')
```



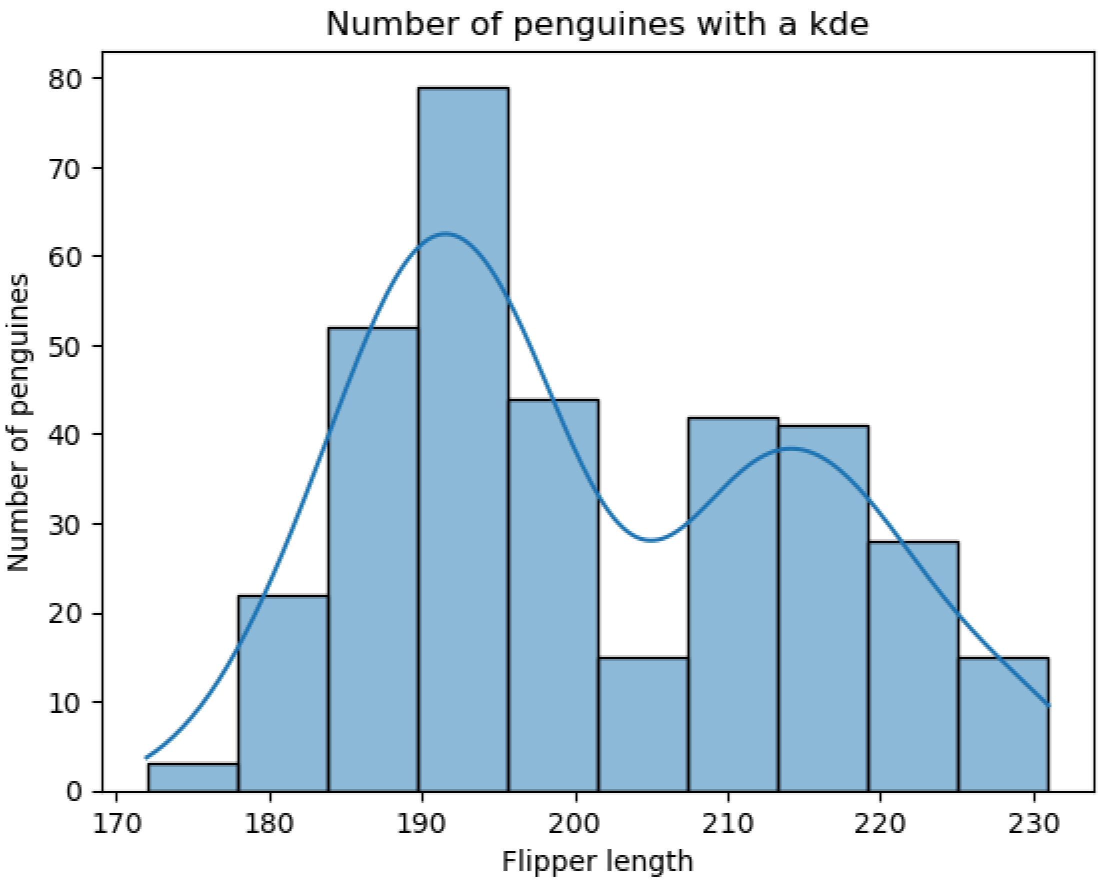
In [9]:
```
#Controlling the bandwidth of the bars
sns.histplot(data=df, x="flipper_length_mm", binwidth=3)
plt.xlabel("Length of flippers")
plt.ylabel("Number of penguines")
```

Out[9]:
```
Text(0, 0.5, 'Number of penguines')
```



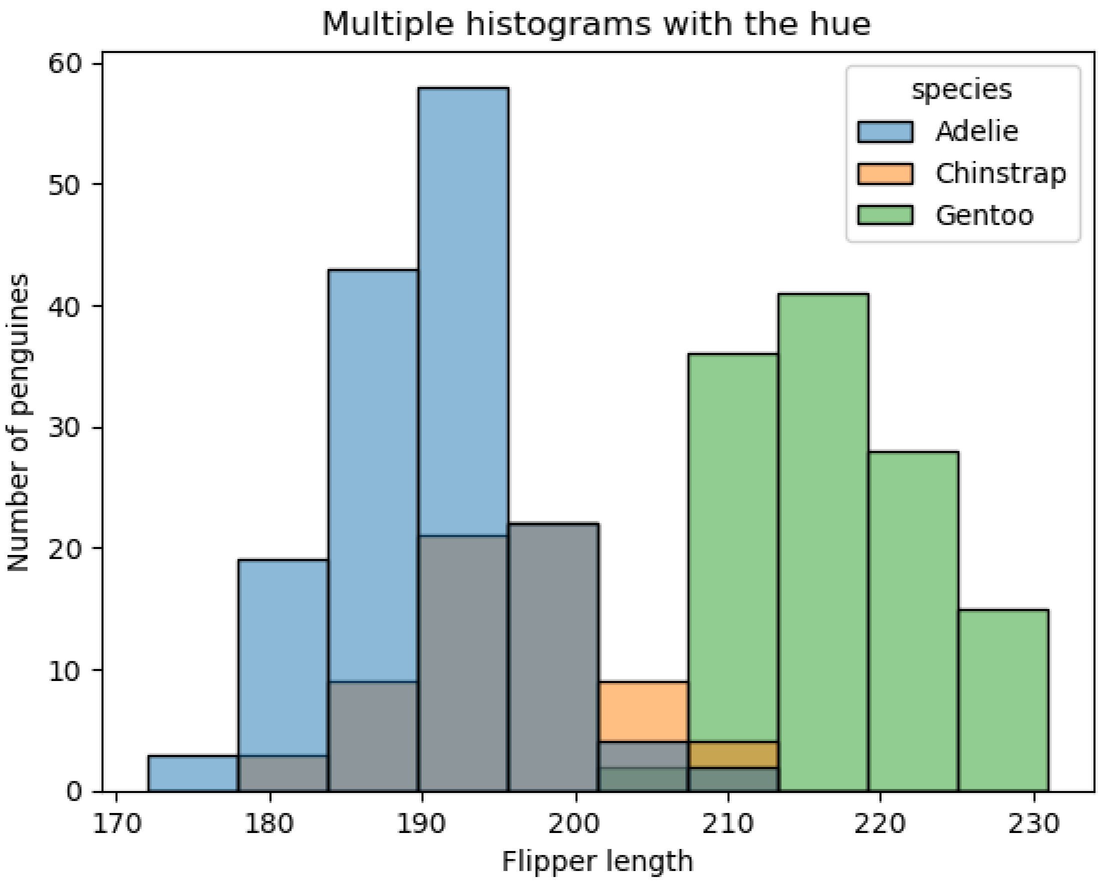
In [10]:
```
#Histoplot with a kde, just enable it using hitoplot kde = True
sns.histplot(data=df, x="flipper_length_mm", kde=True)
plt.xlabel("Flipper length")
plt.ylabel("Number of penguines")
plt.title("Number of penguines with a kde")
```

Out[10]:
```
Text(0.5, 1.0, 'Number of penguines with a kde')
```



In [11]:
```
#You Can use the hue parameter to see the distribution of the penguines
sns.histplot(data=df, x="flipper_length_mm", hue="species")
plt.xlabel("Flipper length")
plt.ylabel("Number of penguines")
plt.title("Multiple histograms with the hue")
```

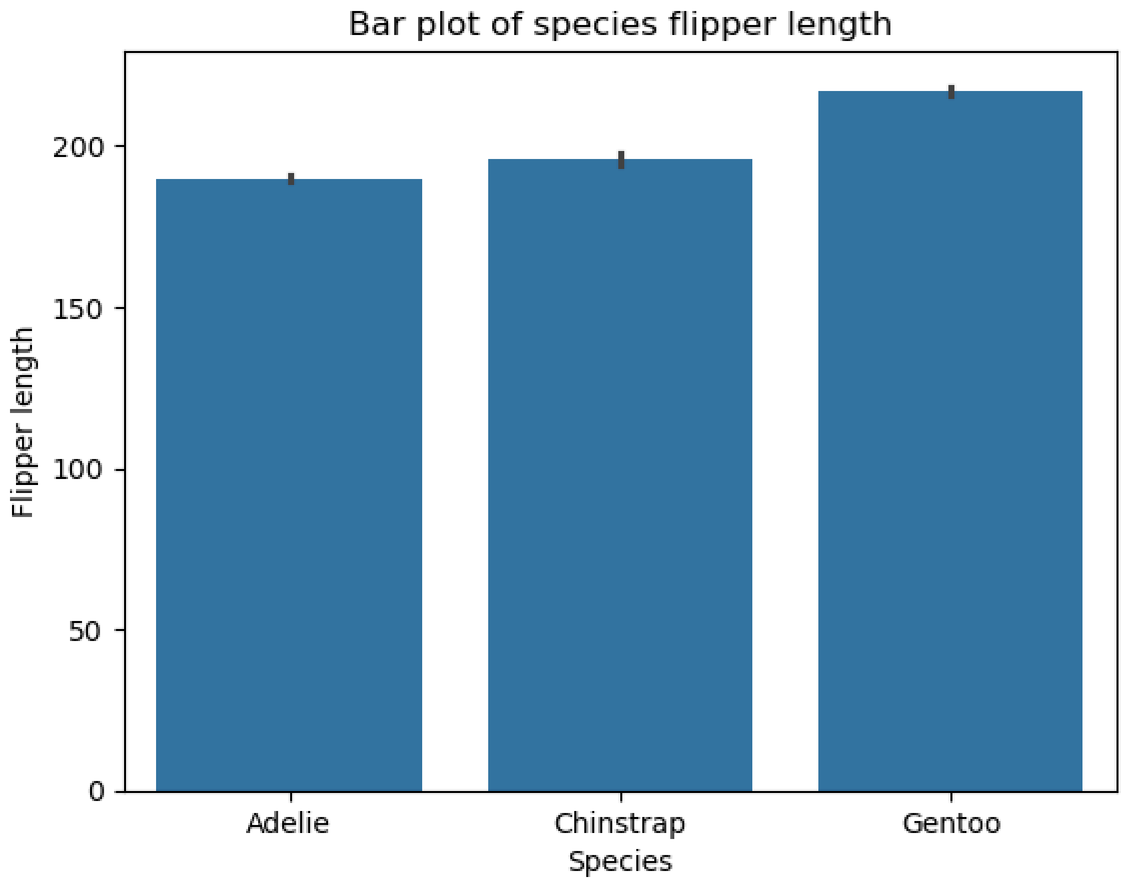
Out[11]: `Text(0.5, 1.0, 'Multiple histograms with the hue')`




```
In [12]: #By default, the bars are calculated based on the mean of the values. You ca
         sns.barplot(x = "species", y = "flipper_length_mm", data = df)
         plt.xlabel("Species")
         plt.ylabel("Flipper length")
         plt.title("Bar plot of species flipper length")
```

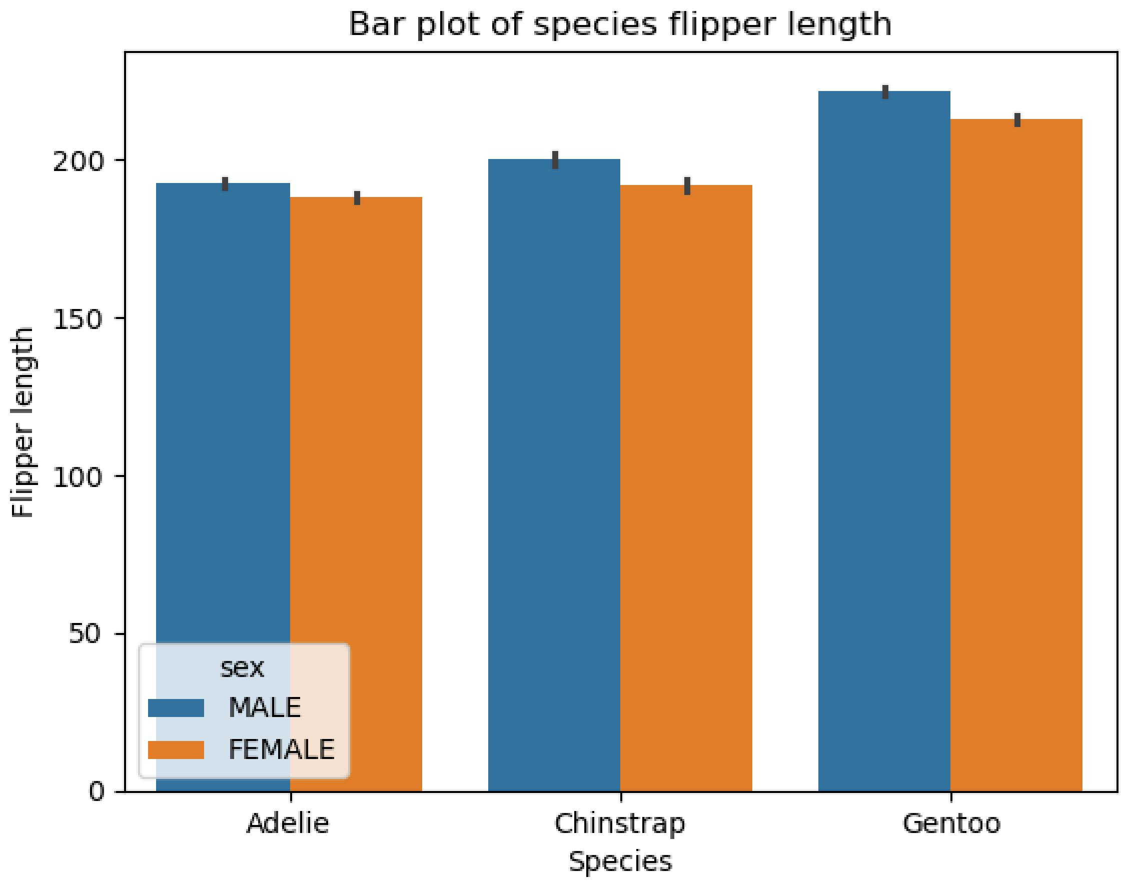
```
Out[12]: Text(0.5, 1.0, 'Bar plot of species flipper length')
```

Loading [MathJax]/extensions/Safe.js



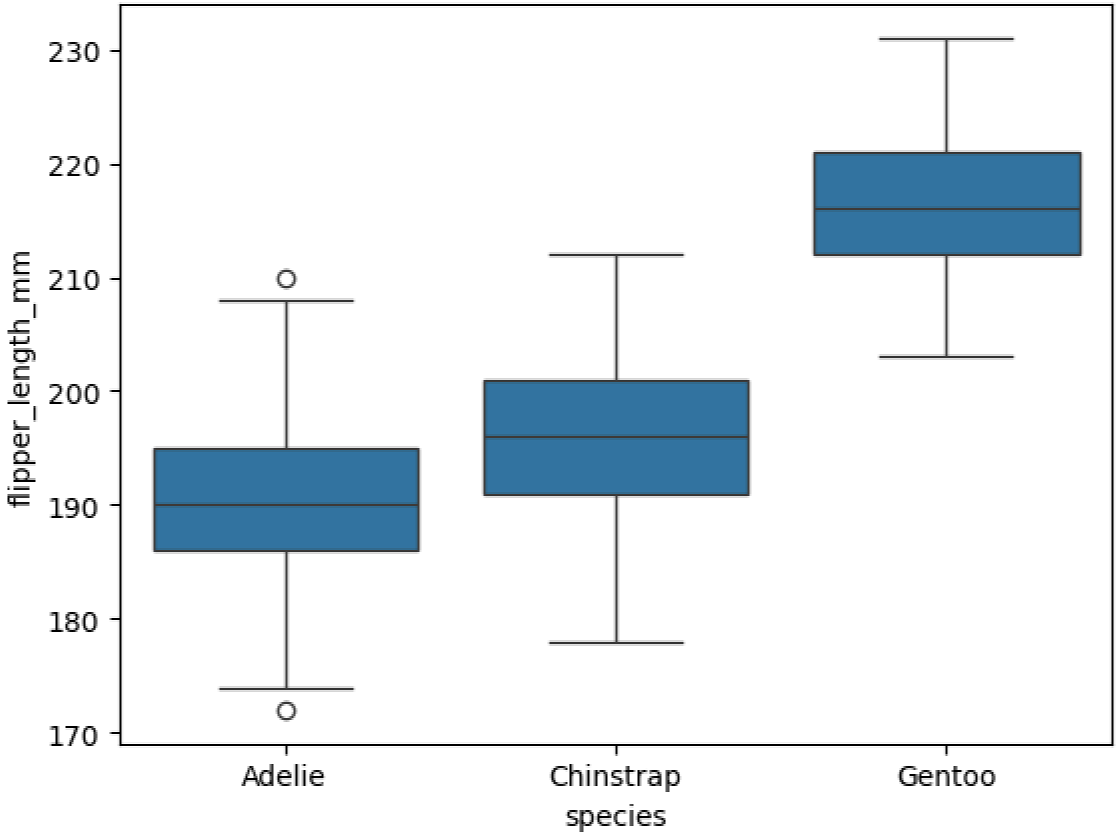
In [13]:
```
#The hue parameter can be used to see the flipper lengths of the species by
#The Bar Plot -> the bars are calculated on the mean of the statistic
sns.barplot(x = "species", y = "flipper_length_mm", data = df, hue="sex")
plt.xlabel("Species")
plt.ylabel("Flipper length")
plt.title("Bar plot of species flipper length")
```

Out[13]:
```
Text(0.5, 1.0, 'Bar plot of species flipper length')
```



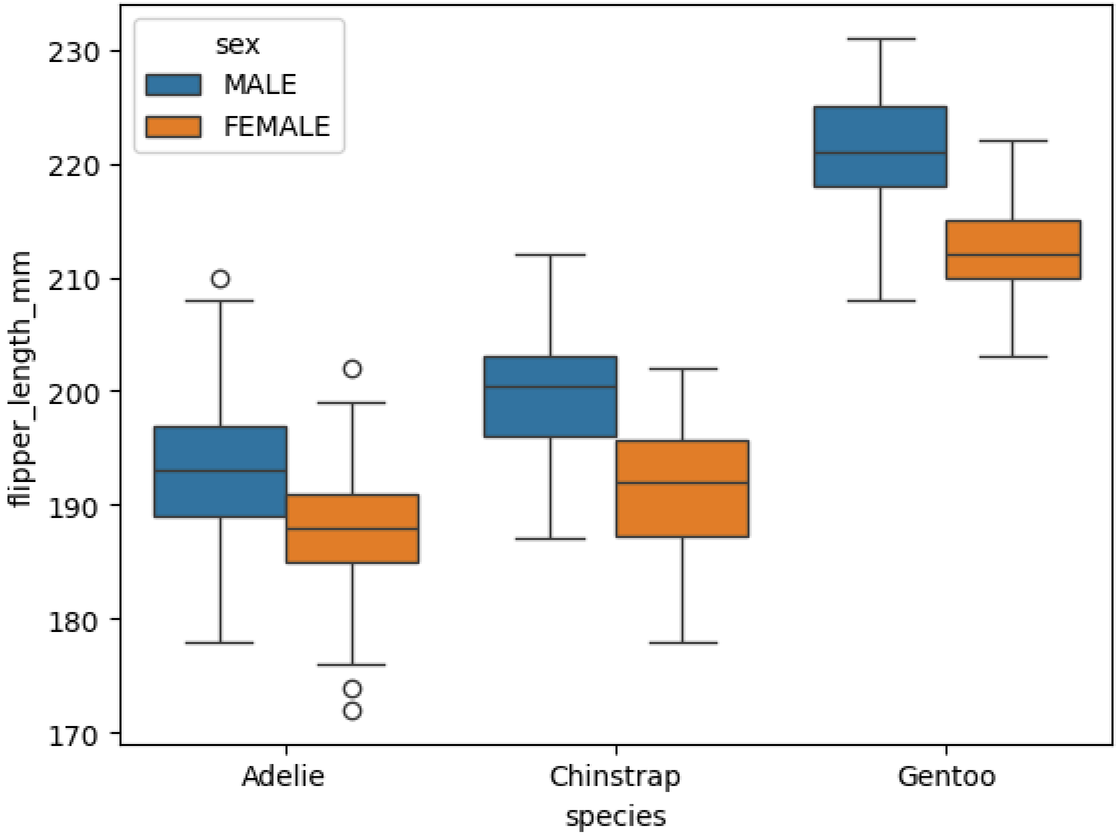
In [14]:
```
#The Box Plot
sns.boxplot(x = "species", y = "flipper_length_mm", data = df)
```

Out[14]:
```
<Axes: xlabel='species', ylabel='flipper_length_mm'>
```



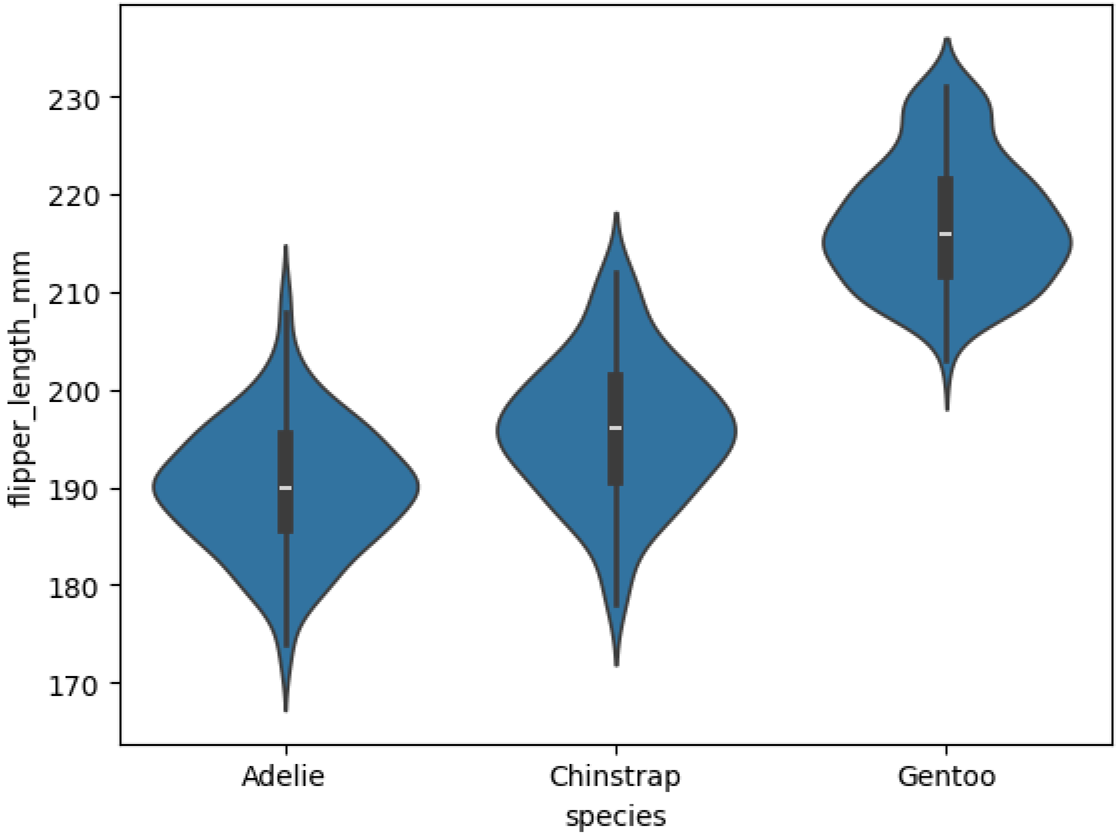
In [15]:
```
sns.boxplot(x = "species",
            y = "flipper_length_mm",
            data = df,
            hue = "sex")
```

Out[15]: <Axes: xlabel='species', ylabel='flipper_length_mm'>



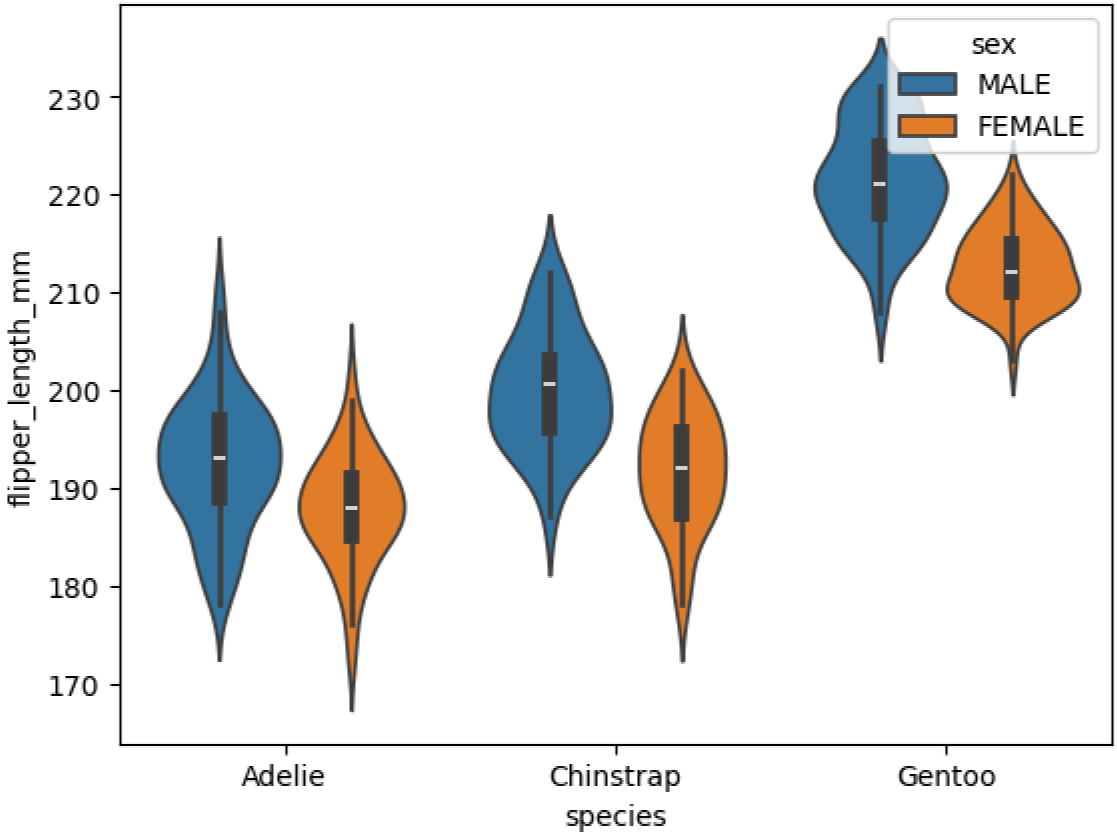
In [16]:
```
#The violin plot
sns.violinplot(x = "species", y = "flipper_length_mm", data = df)
```

Out[16]: <Axes: xlabel='species', ylabel='flipper_length_mm'>



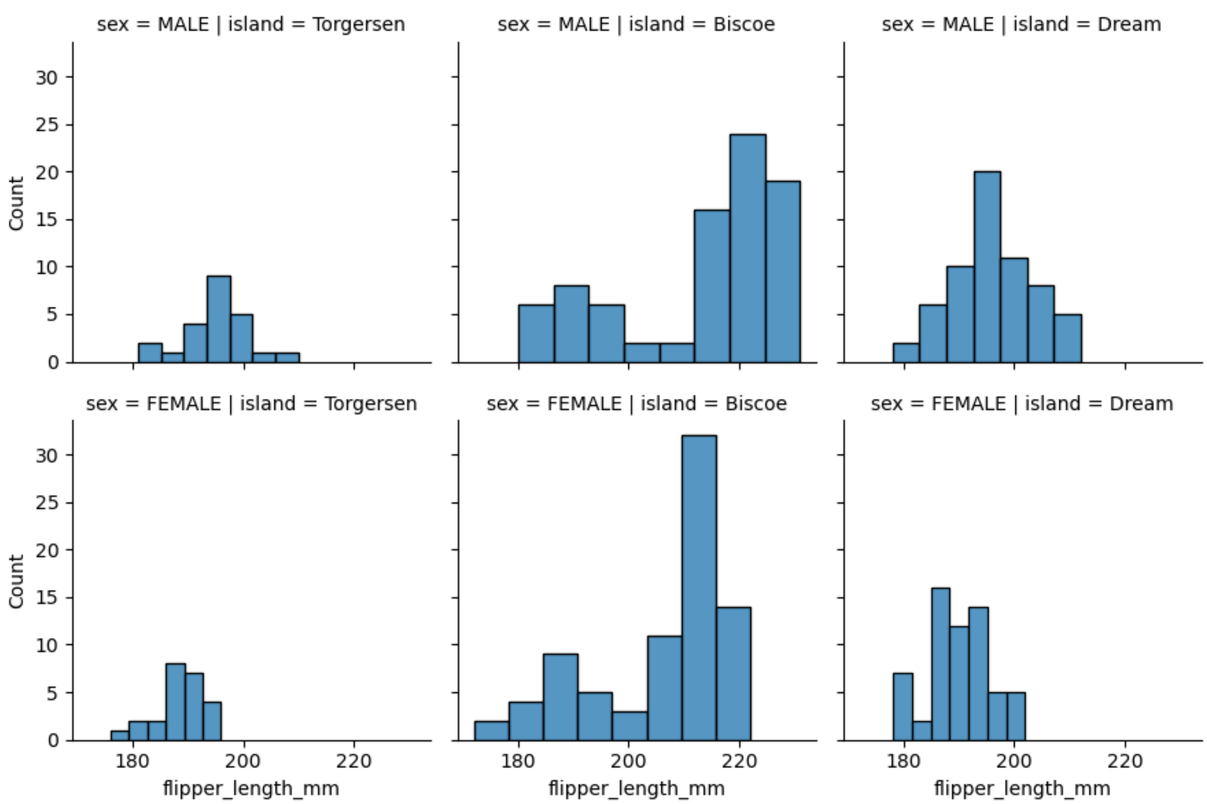
In [17]:
```
sns.violinplot(x = "species",
               y = "flipper_length_mm",
               data = df,
               hue = "sex")
```

Out[17]: <Axes: xlabel='species', ylabel='flipper_length_mm'>



In [18]:
```
#FacetGrid
sns.FacetGrid(df, col="island", row="sex").map(sns.histplot, "flipper_length
```

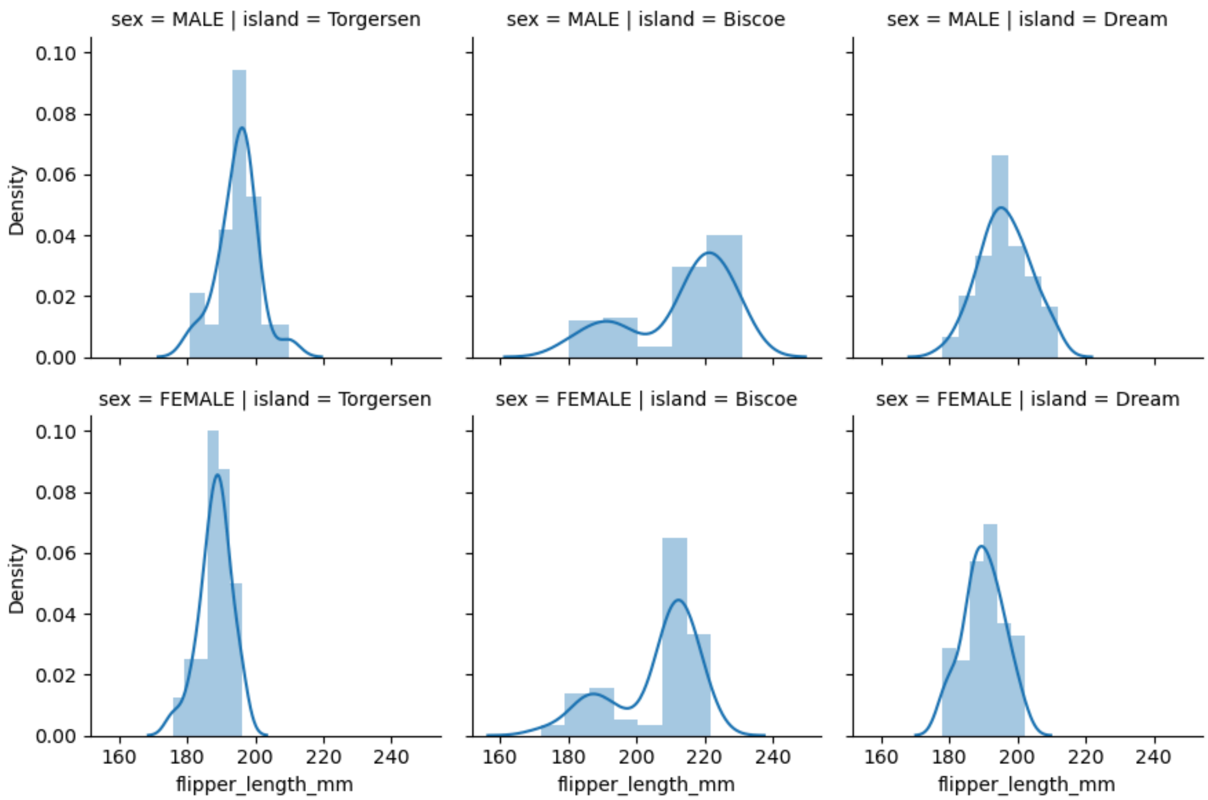
Out[18]: <seaborn.axisgrid.FacetGrid at 0x2043544e9c0>



Loading [MathJax]/extensions/Safe.js

In [19]:
```
sns.FacetGrid(df, col="island", row="sex").map(sns.distplot, "flipper_length
```

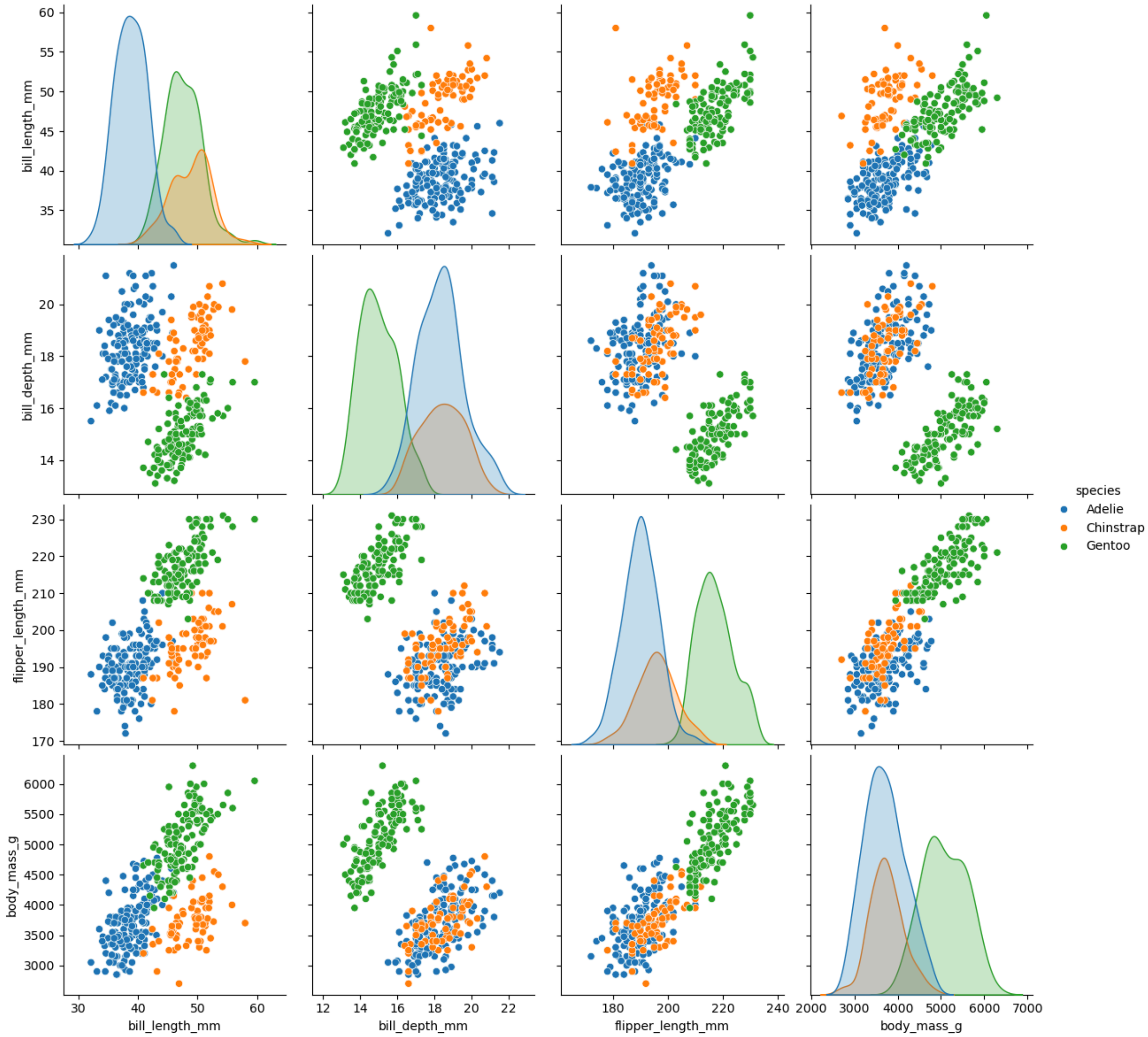
Out[19]: <seaborn.axisgrid.FacetGrid at 0x20437a3ecc0>



In [20]:
```
#Pair Plots
sns.pairplot(df, hue="species", height=3)
```

Out[20]: <seaborn.axisgrid.PairGrid at 0x2043871af00>

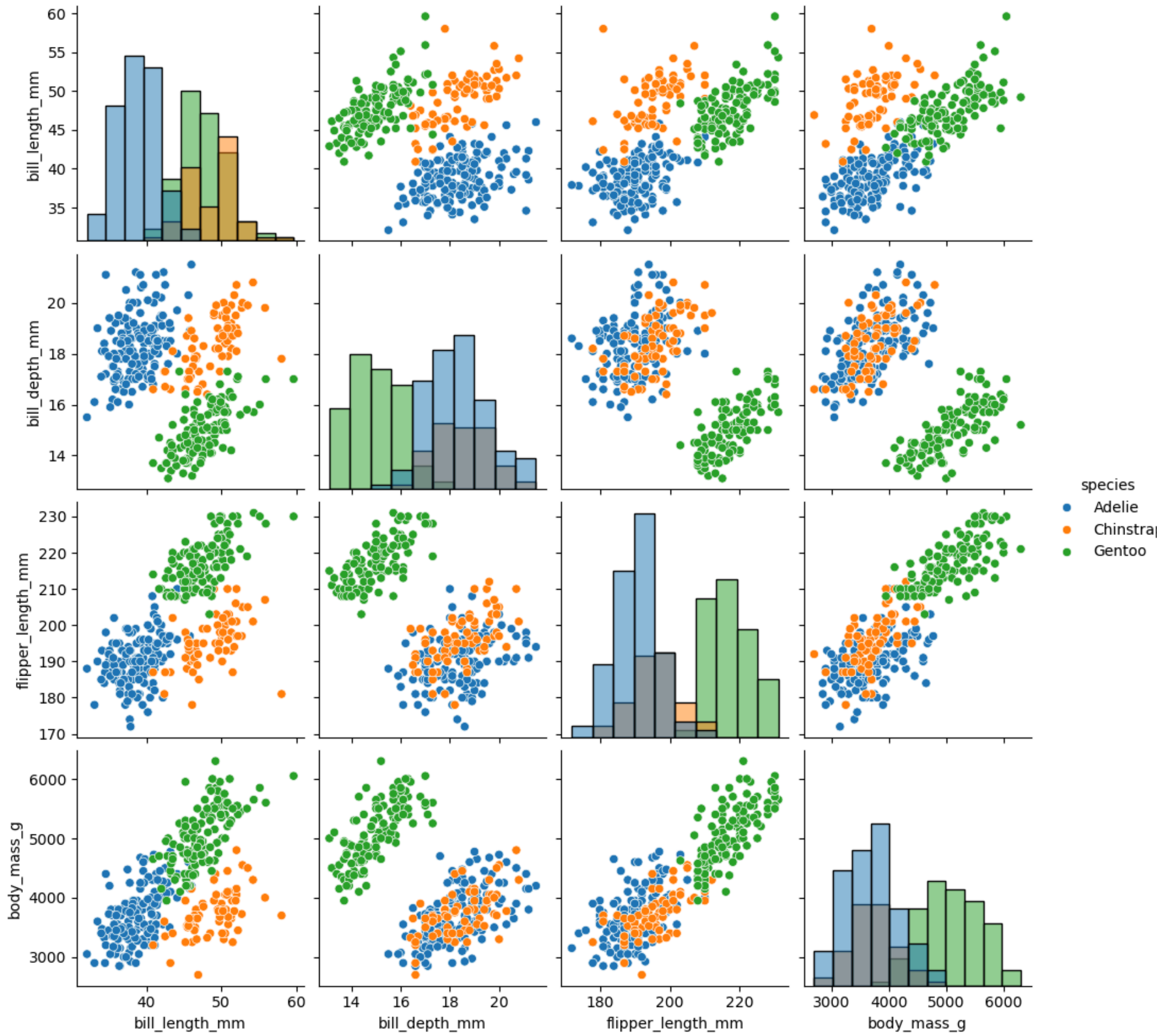
Loading [MathJax]/extensions/Safe.js



In [21]:
```python
#Adding a histogram on the diagonal
sns.pairplot(df, hue="species", diag_kind="hist")
```

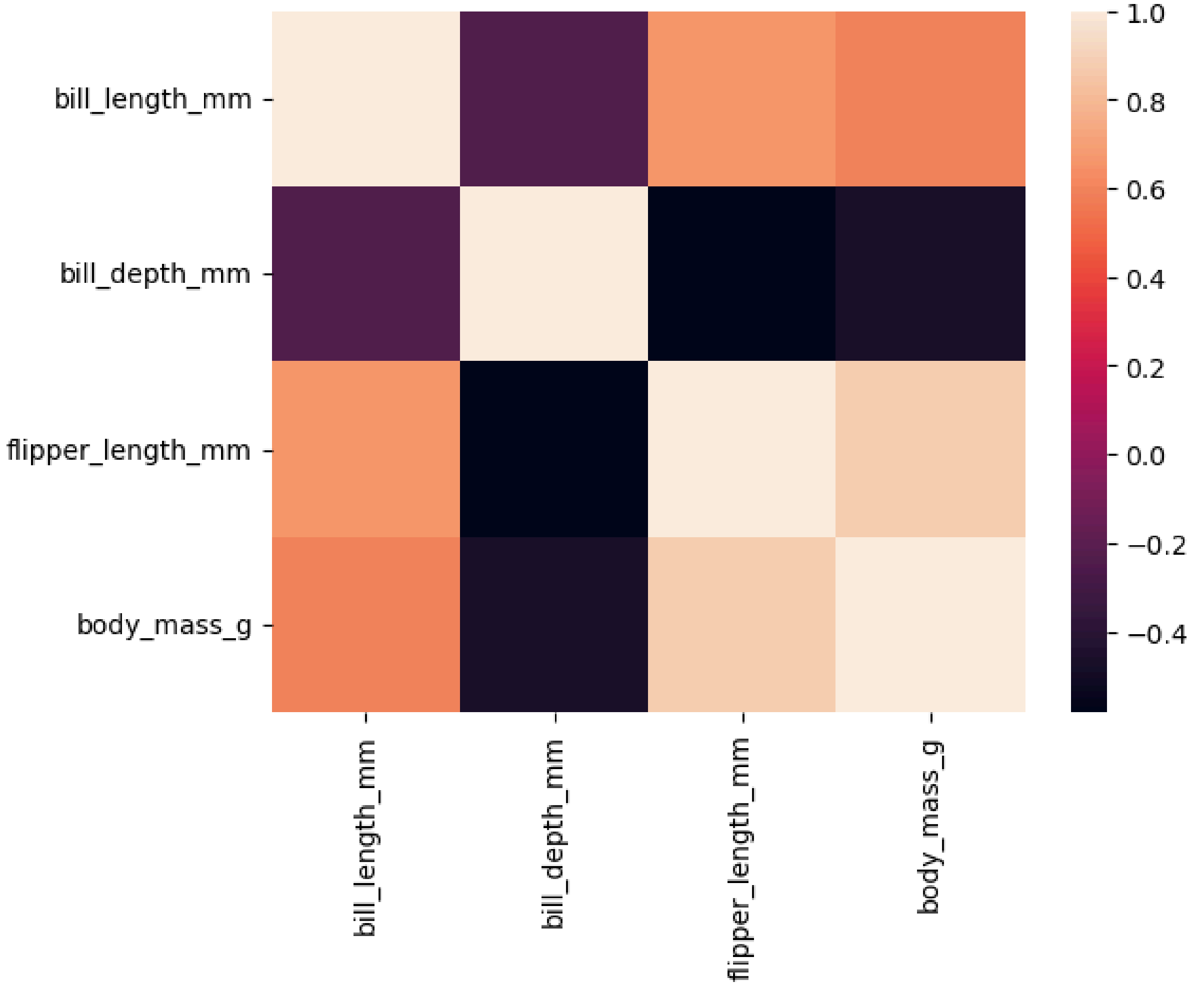
Out[21]: `<seaborn.axisgrid.PairGrid at 0x20438f4e9c0>`

Loading [MathJax]/extensions/Safe.js



In [22]:
```
numeric_df = df.select_dtypes(include='number')

# Plot the heatmap
sns.heatmap(numeric_df.corr())
```

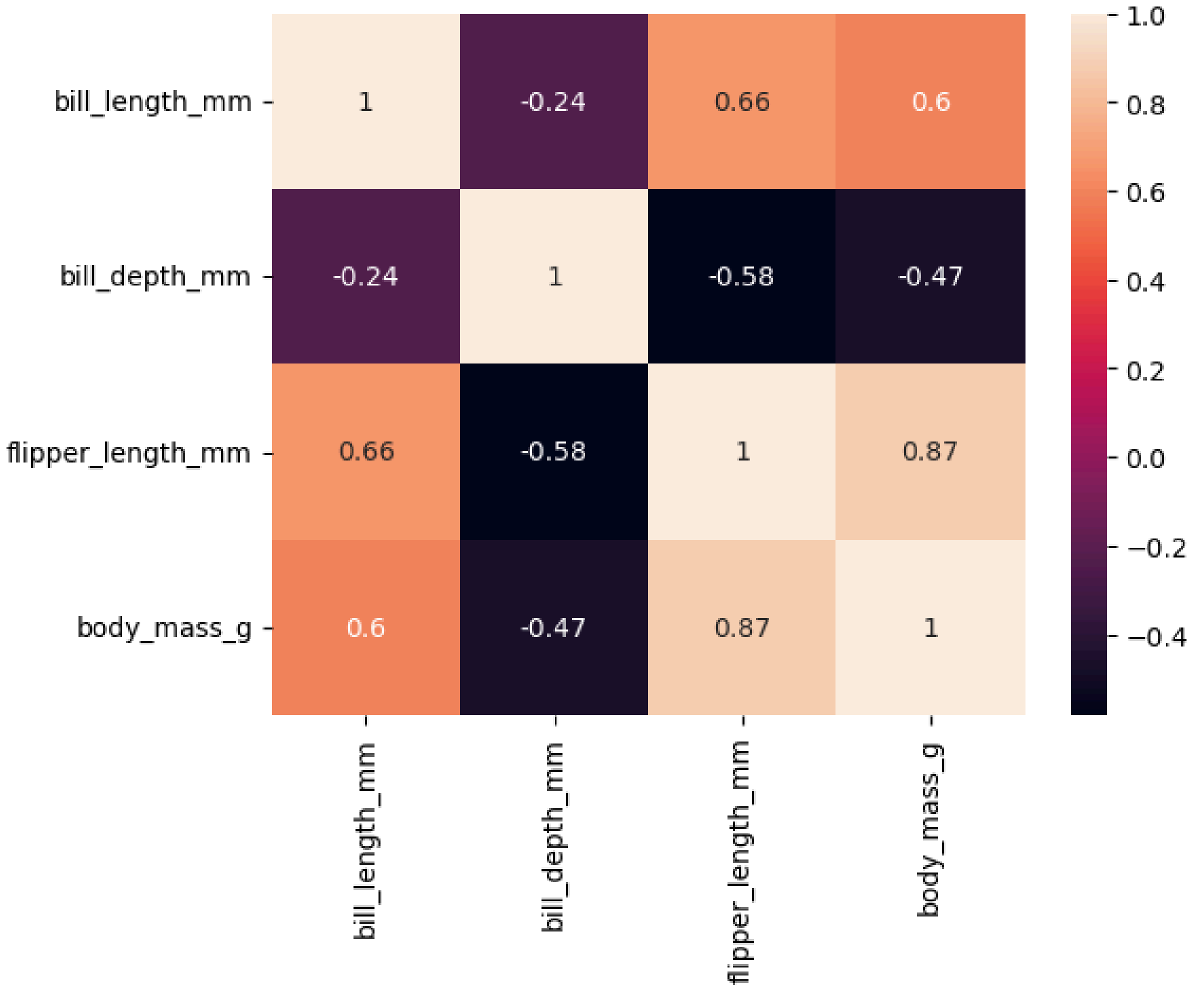
Out[22]: <Axes: >



In [23]:
```
sns.heatmap(numeric_df.corr(), annot=True)
```

Out[23]: `<Axes: >`

Loading [MathJax]/extensions/Safe.js



In [24]:
```
#Can we actually determine the type of species based on the bill length, bil
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import StackingClassifier #ensmbl method of stacking c
from sklearn.metrics import accuracy_score, precision_score, recall_score, f
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report


from sklearn.tree import DecisionTreeClassifier   #estimator in GA
import numpy as np

import warnings
warnings.filterwarnings('ignore')
```

In [41]:
```
# Convert levels to numeric
feature_encoder= LabelEncoder()
df['species'] = feature_encoder.fit_transform(df['species'])
df['island'] = feature_encoder.fit_transform(df['island'])
df['bill_length_mm'] = feature_encoder.fit_transform(df['bill_length_mm'])
df['bill_depth_mm'] = feature_encoder.fit_transform(df['bill_depth_mm'])
er_length_mm'] = feature_encoder.fit_transform(df['flipper_length_m
```

Loading [MathJax]/extensions/Safe.js

```
df['body_mass_g'] = feature_encoder.fit_transform(df['body_mass_g'])
df['sex'] = feature_encoder.fit_transform(df['sex'])

# Define the input features (Defender Score, Attacker Score, Log Time)
X = df[['island', 'bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'b
y = df['species']

# Split the data into training and testing sets (80% train, 20% test)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar


print(df.head())
# Output the shapes of the training and test sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
   species  island  bill_length_mm  bill_depth_mm  flipper_length_mm  body_m
ass_g  sex
0        0       2              42             56                  6
31    1
1        0       2              45             43                 11
33    0
2        0       2              51             49                 20
12    0
3        0       2             164             80                 55
94    2
4        0       2              22             62                 18
19    0
```

Out[41]: ((274, 6), (69, 6), (274,), (69,))

In [44]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accurac
from sklearn.utils import class_weight

# Step 1: Initialize the Random Forest model with class_weight to handle imb
rf_classifier = RandomForestClassifier(random_state=100, class_weight='balar

# Step 2: Train the Random Forest model
rf_classifier.fit(X_train, y_train)

# Step 3: Make predictions
y_pred = rf_classifier.predict(X_test)

# Step 4: Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.9565217391304348
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.97      0.96        35
           1       1.00      0.83      0.91        12
           2       0.96      1.00      0.98        22

    accuracy                           0.96        69
   macro avg       0.97      0.93      0.95        69
weighted avg       0.96      0.96      0.96        69

Confusion Matrix:
[[34  0  1]
 [ 2 10  0]
 [ 0  0 22]]
```

In [47]:

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Train the SVM model (if not already done)
svm_classifier = SVC(random_state=42, probability=True)
svm_classifier.fit(X_train, y_train)

# Step 2: Predict on the test set
y_pred_svm = svm_classifier.predict(X_test)

# Step 3: Calculate accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)

# Step 4: Generate a classification report
classification_rep_svm = classification_report(y_test, y_pred_svm)

# Output the accuracy and classification report
print(f"SVM Accuracy: {accuracy_svm}")
print(f"Classification Report:\n{classification_rep_svm}")

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_svm))
```

```
SVM Accuracy: 0.9420289855072463
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        35
           1       0.91      0.83      0.87        12
           2       0.96      1.00      0.98        22

    accuracy                           0.94        69
   macro avg       0.94      0.93      0.93        69
weighted avg       0.94      0.94      0.94        69

Confusion Matrix:
[[33  1  1]
 [ 2 10  0]
 [ 0  0 22]]
```

In [50]:
```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Train the Naive Bayes model
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

# Step 2: Predict on the test set
y_pred_nb = nb_classifier.predict(X_test)

# Step 3: Calculate accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)

# Step 4: Generate a classification report
classification_rep_nb = classification_report(y_test, y_pred_nb)

# Output the accuracy and classification report
print(f"Naive Bayes Accuracy: {accuracy_nb}")
print(f"Classification Report:\n{classification_rep_nb}")

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_nb))
```

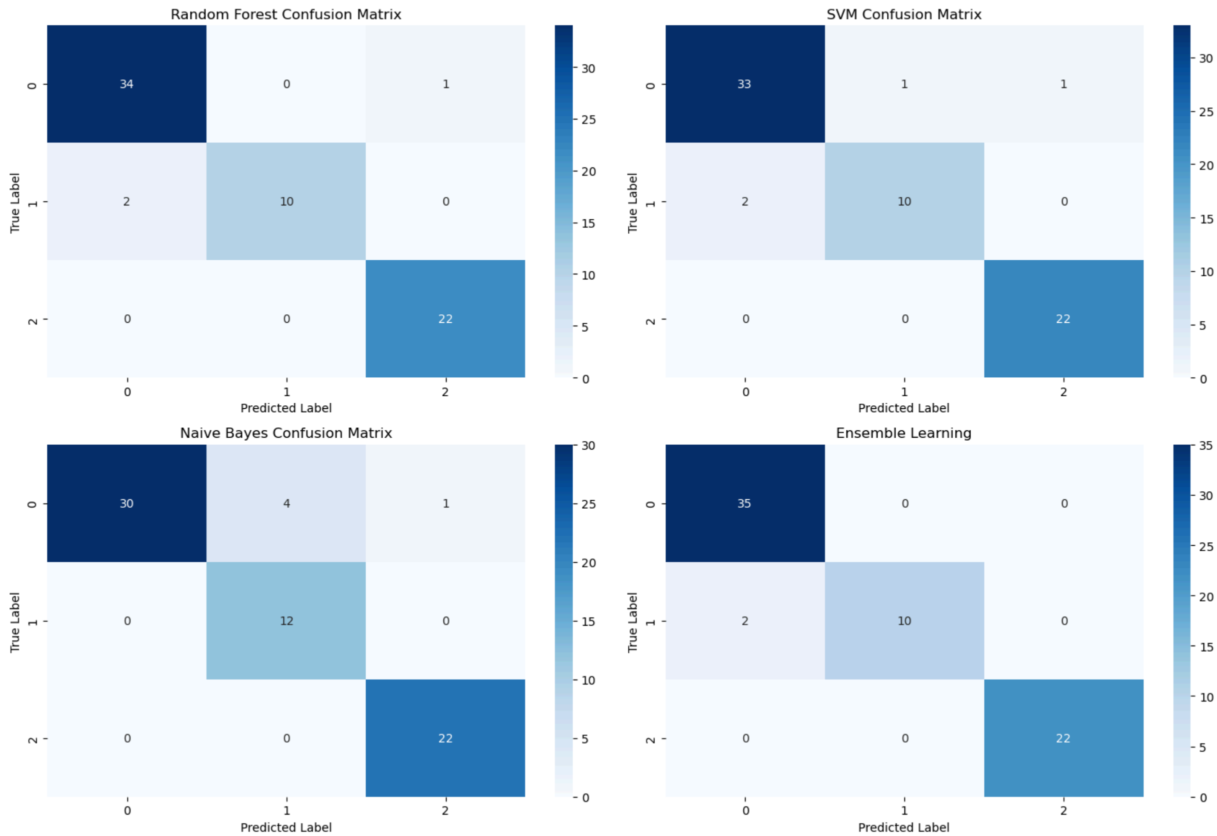
```
Naive Bayes Accuracy: 0.927536231884058
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.86      0.92        35
           1       0.75      1.00      0.86        12
           2       0.96      1.00      0.98        22

    accuracy                           0.93        69
   macro avg       0.90      0.95      0.92        69
weighted avg       0.94      0.93      0.93        69

Confusion Matrix:
[[30  4  1]
 [ 0 12  0]
 [ 0  0 22]]
```

In [53]:
```python
from sklearn.linear_model import LogisticRegression


# Step 2: Define the base models
estimators = [
    ('rf', RandomForestClassifier(random_state=42)),
    ('svm', SVC(random_state=42, probability=True)),  # Use probability=True
    ('nb', GaussianNB())
]

# Step 3: Define the Stacking Classifier with a meta-model (Logistic Regress
stacking_classifier = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression(),
    cv=5  # 5-fold cross-validation
```

```python
# Step 4: Train the Stacking Classifier
stacking_classifier.fit(X_train, y_train)

# Step 5: Predict on the test set
y_pred_stack = stacking_classifier.predict(X_test)

# Step 6: Evaluate the Stacking Classifier
accuracy_stack = accuracy_score(y_test, y_pred_stack)
classification_rep_stack = classification_report(y_test, y_pred_stack)

# Output the accuracy and classification report
print(f"Stacking Classifier Accuracy: {accuracy_stack}")
print(f"Classification Report:\n{classification_rep_stack}")

print(confusion_matrix(y_test, y_pred_stack))
```

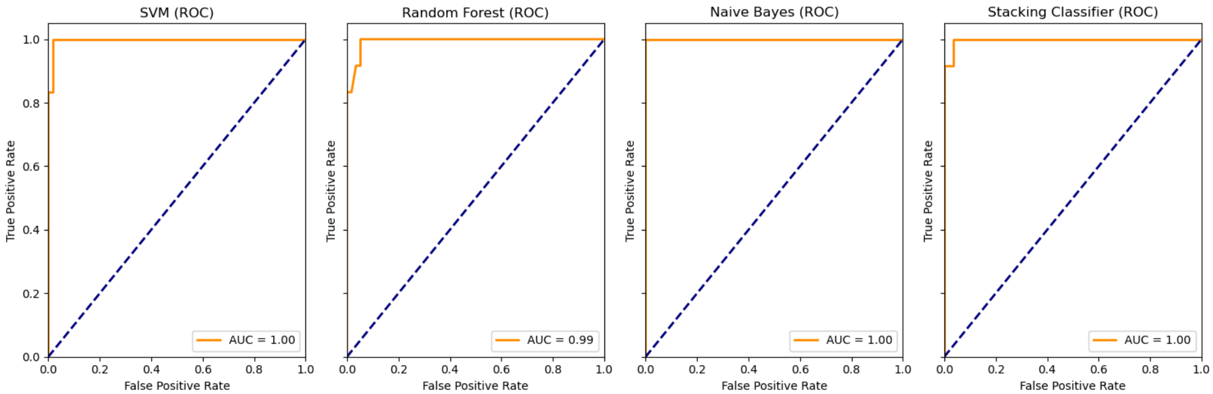
```
Stacking Classifier Accuracy: 0.9710144927536232
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        35
           1       1.00      0.83      0.91        12
           2       1.00      1.00      1.00        22

    accuracy                           0.97        69
   macro avg       0.98      0.94      0.96        69
weighted avg       0.97      0.97      0.97        69

[[35  0  0]
 [ 2 10  0]
 [ 0  0 22]]
```

In [56]:
```python
fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title, ax):
    cm = confusion_matrix(y_true, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=ax)
    ax.set_title(title)
    ax.set_ylabel("True Label")
    ax.set_xlabel("Predicted Label")



# Plot confusion matrices for each model
plot_confusion_matrix(y_test, y_pred, "Random Forest Confusion Matrix", axes
plot_confusion_matrix(y_test, y_pred_svm, "SVM Confusion Matrix", axes[0, 1]
plot_confusion_matrix(y_test, y_pred_nb, "Naive Bayes Confusion Matrix", axe
plot_confusion_matrix(y_test, y_pred_stack, "Ensemble Learning", axes[1, 1])



# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```



In [62]:
```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Example precomputed probabilities for each model (replace with your actual
y_prob_svm = svm_classifier.predict_proba(X_test)[:, 1]  # SVM probabilities
y_prob_rf = rf_classifier.predict_proba(X_test)[:, 1]     # Random Forest pro
y_prob_nb = nb_classifier.predict_proba(X_test)[:, 1]     # Naive Bayes
y_prob_stacking_classifier = stacking_classifier.predict_proba(X_test)[:, 1]

# Compute ROC curves and AUC for each model
roc_data = {}
for model_name, y_proba in zip(['SVM', 'Random Forest', 'Naive Bayes', 'Stac
                                [y_prob_svm, y_prob_rf, y_prob_nb, y_prob_sta
    fpr, tpr, _ = roc_curve(y_test, y_proba, pos_label=1)
    roc_auc = auc(fpr, tpr)
    roc_data[model_name] = (fpr, tpr, roc_auc)

# Plot ROC curves in subplots
fig, axes = plt.subplots(1, len(roc_data), figsize=(15, 5), sharex=True, sha

for ax, (model_name, (fpr, tpr, roc_auc)) in zip(axes, roc_data.items()):
    ax.plot(fpr, tpr, color='darkorange', lw=2, label=f'AUC = {roc_auc:.2f}'
    ax.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_title(f'{model_name} (ROC)')
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.legend(loc="lower right")
```

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



In [ ]:

Loading [MathJax]/extensions/Safe.js