Business Problem

Our goal is to create a model that can help predict a species of a penguin based on physical attributes, then we can use that model to help researchers classify penguins in the field, instead of needing an experienced biologist

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [2]: df = pd.read_csv("penguins_size.csv")
        df.head()
```

Out[2]:

	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0
3	Adelie	Torgersen	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0

"Palmer Penguins" dataset Summary:

- species: penguin species (Chinstrap, Adélie, or Gentoo)
  - culmen_length_mm: culmen length (mm)
  - culmen_depth_mm: culmen depth (mm)
  - flipper_length_mm: flipper length (mm)
  - body_mass_g: body mass (g)
  - island: island name (Dream, Torgersen, or Biscoe) in the Palmer Archipelago (Antarctica)
  - sex: penguin sex

In [3]:

```
df.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   culmen_length_mm   342 non-null    float64
 3   culmen_depth_mm    342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                334 non-null    object 
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
species               0
island                0
culmen_length_mm      2
culmen_depth_mm       2
flipper_length_mm     2
body_mass_g           2
sex                  10
dtype: int64
```

Data Preprocessing

EDA

In [ ]:

```

```

In [ ]:

```

```

Feature Engineering

In [5]:

```
#Droping the missing values
df = df.dropna()
```

In [6]:

```
#shape of data after dropping missing values
df.shape
```

Out[6]:

```
(334, 7)
```

In [7]:
```
df = df[df['sex']!='.']
df.shape
```

Out[7]:
```
(333, 7)
```

X & y

In [8]:
```
X = pd.get_dummies(df.drop('species',axis=1),drop_first=True)
y = df['species']
```

Train Test Split

In [9]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
m_state=101)
```

Modelling

Random Forest Classifier with default parameters

In [10]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [11]:
```
model = RandomForestClassifier()
```

In [12]:
```
model.fit(X_train,y_train)
```

Out[12]:
```
RandomForestClassifier()
```

In [13]:
```
ypred_train = model.predict(X_train)
```

In [14]:
```
ypred_test = model.predict(X_test)
```

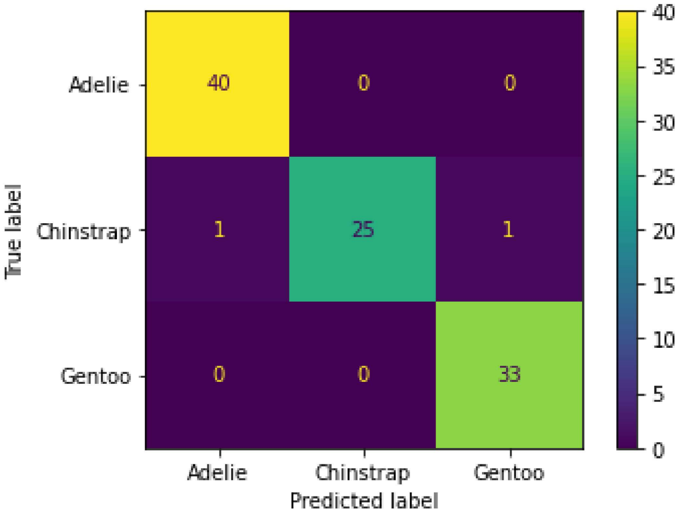
Evaluation

In [15]:
```
from sklearn.metrics import accuracy_score
print("Train accuracy:",accuracy_score(ypred_train,y_train))
print("Test accuracy:",accuracy_score(ypred_test,y_test))
```

```
Train accuracy: 1.0
Test accuracy: 0.98
```

In [16]:

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(model,X_test,y_test)
plt.show()
```



In [17]:

```
from sklearn.metrics import classification_report
print(classification_report(y_test,ypred_test))
```

```
              precision    recall  f1-score   support

      Adelie       0.98      1.00      0.99        40
   Chinstrap       1.00      0.93      0.96        27
      Gentoo       0.97      1.00      0.99        33

    accuracy                           0.98       100
   macro avg       0.98      0.98      0.98       100
weighted avg       0.98      0.98      0.98       100
```

In [18]:

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(model,X,y,cv=5)
print("Cross Validation Score:",scores.mean())
```

```
Cross Validation Score: 0.9909995477159657
```

Feature Importance

In [19]:

```
model.feature_importances_
```

Out[19]:

```
array([0.31153151, 0.11171903, 0.32122507, 0.14451639, 0.08109066,
       0.02100147, 0.00891587])
```

In [20]:

```
pd.DataFrame(index=X.columns,data=model.feature_importances_,columns=['Feature
Importance'])
```

Out[20]:

	Feature Importance
culmen_length_mm	0.311532
culmen_depth_mm	0.111719
flipper_length_mm	0.321225
body_mass_g	0.144516
island_Dream	0.081091
island_Torgersen	0.021001
sex_MALE	0.008916

HyperParameter Tuning

In [21]:

```
from sklearn.model_selection import GridSearchCV
```

In [22]:

```
# model
estimator = RandomForestClassifier()

# parameters (which you want to tune and identify the best)
param_grid = {'n_estimators':list(range(1,101))}
```

In [23]:

```
grid = GridSearchCV(estimator,param_grid, scoring="accuracy",cv=5)
```

In [24]:

```
grid.fit(X_train,y_train)
```

Out[24]:

```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'n_estimators': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12,
                                          13, 14, 15, 16, 17, 18, 19, 20, 21,
                                          22, 23, 24, 25, 26, 27, 28, 29, 30,
...]},
             scoring='accuracy')
```

In [25]:

```
grid.best_params_
```

Out[25]:

```
{'n_estimators': 14}
```