

# Classification

The Palmer Penguins dataset is a common resource for data exploration and demonstration of data analysis techniques. It was brought into the limelight by Dr. Kristen Gorman and the Palmer Station, Antarctica LTER, which is a member of the Long Term Ecological Research Network.

The dataset includes data for 344 penguins from three different species found on three islands in the Palmer Archipelago, Antarctica. The measured attributes in the dataset include:

1. **Species:** The species of the penguin, which can be Adelie, Gentoo, or Chinstrap.
2. **Island:** The island in the Palmer Archipelago, Antarctica, where the penguin observation was made. The options are Torgersen, Biscoe, or Dream.
3. **Culmen Length (mm):** The length of the penguin's culmen (bill).
4. **Culmen Depth (mm):** The depth of the penguin's culmen (bill).
5. **Flipper Length (mm):** The length of the penguin's flipper.
6. **Body Mass (g):** The body mass of the penguin.
7. **Sex:** The sex of the penguin.

The Palmer Penguins dataset is excellent for practicing data cleaning, exploration, and visualization.

You can find more information about the dataset, including a more detailed explanation of the variables, in this repository: [allisonhorst/palmerpenguins](https://allisonhorst.github.io/palmerpenguins/).

For more in-depth studies or referencing, you might also consider checking out the publications from Palmer Station LTER: [pal.lternet.edu/bibliography](https://pal.lternet.edu/bibliography).

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import ConfusionMatrixDisplay
```

```
In [ ]: # read penguins dataset from github
penguins = pd.read_csv('https://raw.githubusercontent.com/allisonhorst/palmerpenguins/penguins.head()')
```

```
Out[ ]:   species  island  bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g
0  Adelie  Torgersen         39.1           18.7           181.0         3750.0  r
1  Adelie  Torgersen         39.5           17.4           186.0         3800.0  fer
2  Adelie  Torgersen         40.3           18.0           195.0         3250.0  fer
3  Adelie  Torgersen         NaN           NaN           NaN           NaN    I
4  Adelie  Torgersen         36.7           19.3           193.0         3450.0  fer
```

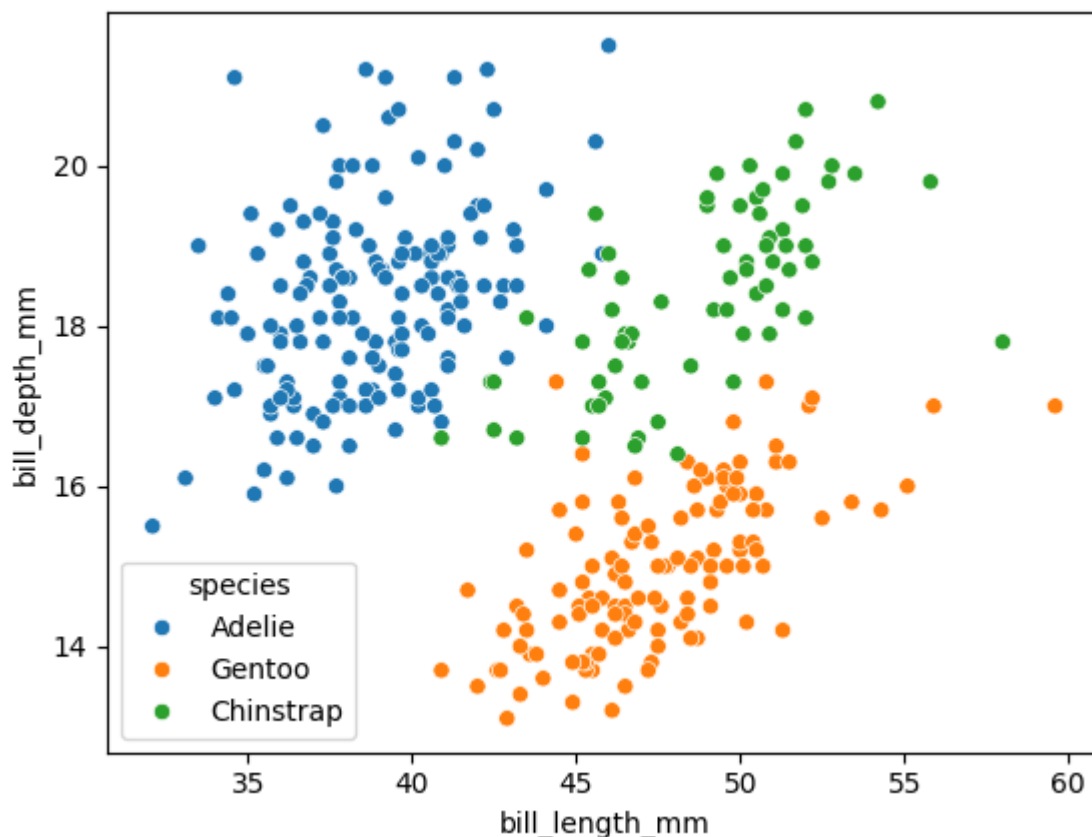
```
In [ ]: # drop the year column, it is not useful for our analysis,
# and it has no adequate explanation in the dataset documentation

penguins = penguins.drop('year', axis=1)
```

```
In [ ]: # Create a scatterplot of bill length vs bill depth using seaborn, hue by species.
# Add a title.

sns.scatterplot(data=penguins, x='bill_length_mm', y='bill_depth_mm', hue='species')
```

```
Out[ ]: <Axes: xlabel='bill_length_mm', ylabel='bill_depth_mm'>
```



```
In [ ]: numeric_features = ['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']
        categorical_features = ['island', 'sex']
```

```
In [ ]: from sklearn.pipeline import make_pipeline
        # create a pipeline to impute missing values with the mean and scale numeric features
        num_pipeline = make_pipeline(
            SimpleImputer(strategy="mean"),
            StandardScaler())

        # create a pipeline to impute missing values with the most frequent value and one-hot encode categorical features
        cat_pipeline = make_pipeline(
            SimpleImputer(strategy="most_frequent"),
            OneHotEncoder(handle_unknown="ignore"))

        # create a column transformer to apply the numeric and categorical pipelines to the data
        # use remainder='passthrough' to keep the remaining features in the dataframe
        preprocessor = ColumnTransformer([
            ("num", num_pipeline, numeric_features),
            ("cat", cat_pipeline, categorical_features),
        ], remainder='passthrough')

        # fit_transform the preprocessor on the penguins dataset
        penguins_prepared = preprocessor.fit_transform(penguins)
        # convert the result to a dataframe
        df_penguins_prepared = pd.DataFrame(penguins_prepared,
                                             columns=preprocessor.get_feature_names_out())
        # use the preprocessor's get_feature_names_out() method to get the column names
        print(preprocessor.get_feature_names_out())

        # display the first 5 rows of the preprocessed dataframe
        df_penguins_prepared.head()
```

```
['num__bill_length_mm' 'num__bill_depth_mm' 'num__flipper_length_mm'
 'num__body_mass_g' 'cat__island_Biscoe' 'cat__island_Dream'
 'cat__island_Torgersen' 'cat__sex_female' 'cat__sex_male'
 'remainder__species']
```

```
Out[ ]:   num__bill_length_mm  num__bill_depth_mm  num__flipper_length_mm  num__body_mass_g
```

0	-0.887081	0.787743	-1.422488	-0.565789
1	-0.813494	0.126556	-1.065352	-0.503168
2	-0.66632	0.431719	-0.422507	-1.192003
3	-0.0	0.0	0.0	0.0
4	-1.328605	1.092905	-0.565361	-0.941517

```
In [ ]: # separate the features from the target
        # call the features X and the target y
        X = df_penguins_prepared.drop("remainder__species", axis=1)
        y = df_penguins_prepared["remainder__species"]
```

```
In [ ]: # setup binary classification for Adelie vs. rest of species
# use the Adelie species as the positive class
# create a new target called y_adelie
y_adelie = (y == 'Adelie')
y_adelie.value_counts()
```

```
Out[ ]: remainder__species
False    192
True     152
Name: count, dtype: int64
```

```
In [ ]: # build an SGDClassifier model using X and y
# use random_state=42 for reproducibility
sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X, y_adelie)
```

```
Out[ ]: SGDClassifier
SGDClassifier(random_state=42)
```

```
In [ ]: # compute the accuracy using cross_val_score with cv=10
accuracy = cross_val_score(sgd_clf, X, y_adelie, cv=10)
```

```
In [ ]: # compute the mean accuracy
accuracy.mean()
```

```
Out[ ]: 0.9942016806722689
```

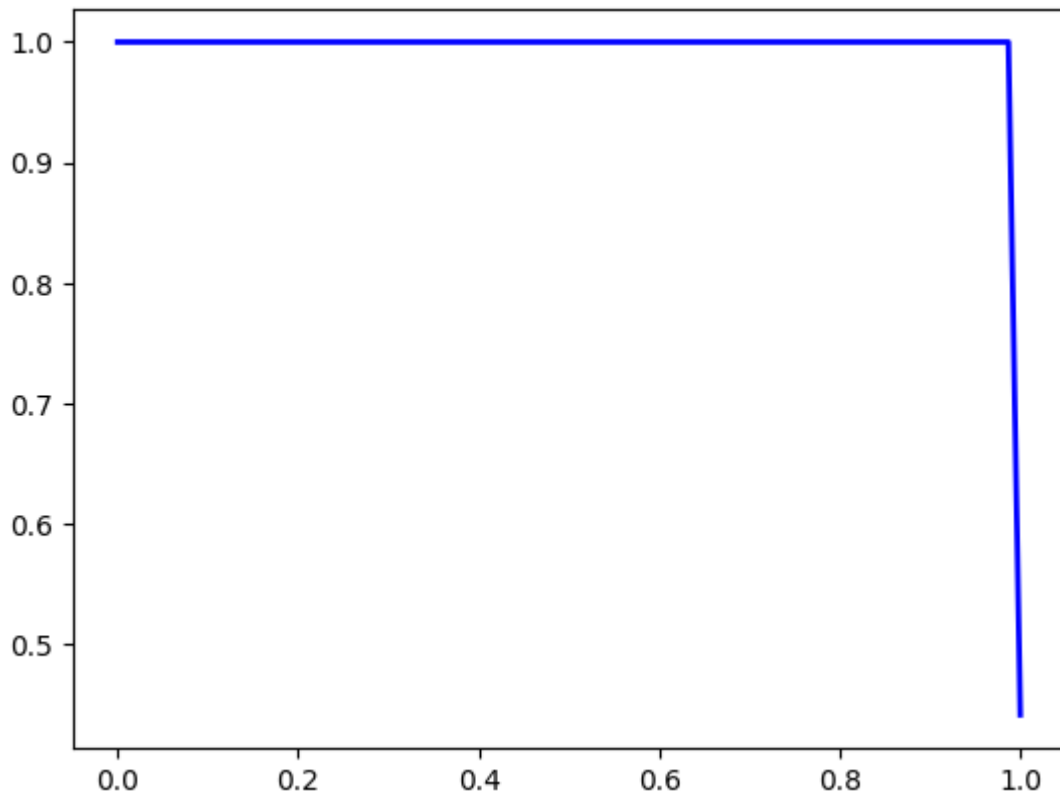
```
In [ ]: # predict the target using cross_val_predict with cv=10
# call the result y_train_pred
y_train_pred = cross_val_predict(sgd_clf, X, y_adelie, cv=10)
y_train_pred
```



```
recall
```

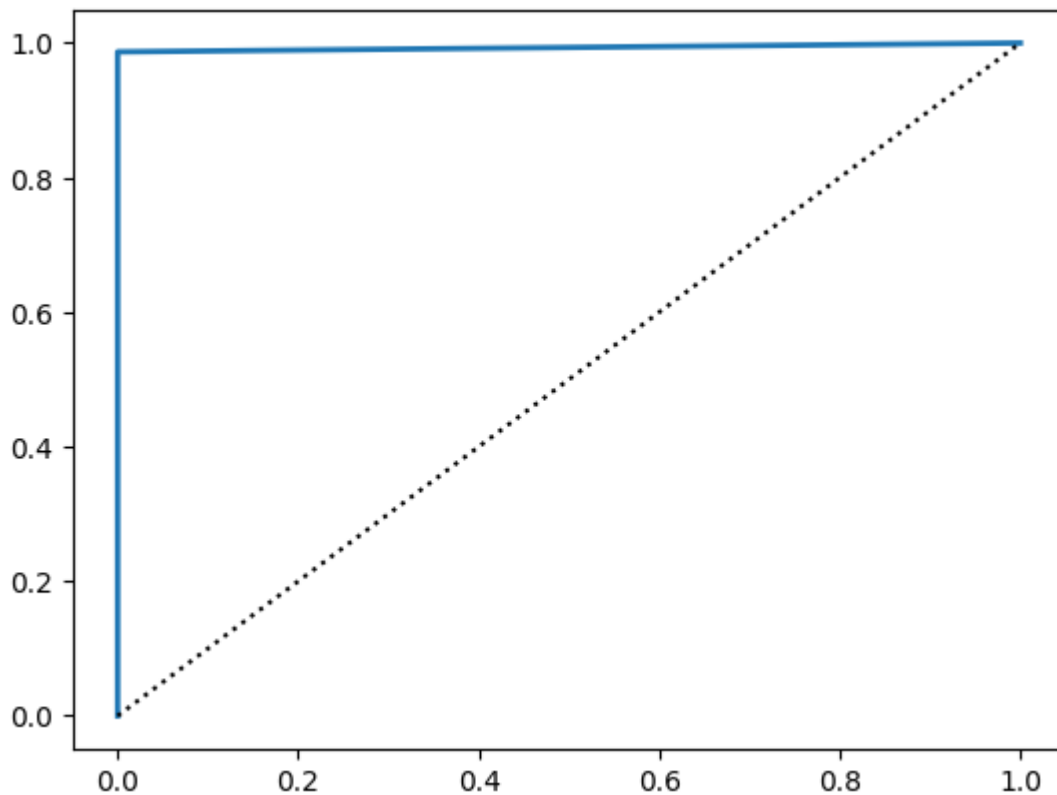
```
Out[ ]: 0.9868421052631579
```

```
In [ ]: # draw the precision-recall curve
# call the result precisions, recalls, thresholds
#y_scores = cross_val_predict(sgd_clf, X, y_train_pred, cv=3, method="decision_func")
precisions, recalls, thresholds = precision_recall_curve(y_adelie, y_train_pred)
plt.plot(recalls, precisions, "b-", linewidth=2)
plt.show()
```



```
In [ ]: # call the result fpr, tpr, thresholds
fpr, tpr, thresholds = roc_curve(y_adelie, y_train_pred)
# plot the roc curve
plt.plot(fpr, tpr, linewidth=2)
plt.plot([0, 1], [0, 1], 'k:', label="Random classifier's ROC curve")
```

```
Out[ ]: [<matplotlib.lines.Line2D at 0x1289ba360>]
```



```
In [ ]: # now let's do multiclass classification
# build an SGDClassifier model using X and y
# use random_state=42 for reproducibility

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X, y)
```

```
Out[ ]: SGDClassifier
SGDClassifier(random_state=42)
```

```
In [ ]: # show the mean accuracy using cross_val_score with cv=10
accuracy = cross_val_score(sgd_clf, X, y, cv=10)
accuracy.mean()
```

```
Out[ ]: 0.9883193277310924
```

```
In [ ]: # predict the target using cross_val_predict with cv=10
# call the result y_train_pred
# show the confusion matrix
y_train_pred = cross_val_predict(sgd_clf, X, y, cv=10)
conf_matrix = confusion_matrix(y, y_train_pred)
conf_matrix
```

```
Out[ ]: array([[150,  2,  0],
               [ 1, 67,  0],
               [ 1,  0, 123]])
```

```
In [ ]: # use ConfusionMatrixDisplay to display the confusion matrix
disp = ConfusionMatrixDisplay(conf_matrix, display_labels=sgd_clf.classes_)
disp.plot()
```

```
Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x12896a4b0>
```

