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.

For more in-depth studies or referencing, you might also consider checking out the publications from Palmer Station LTER: 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 penquins dataset from github
penguins = pd.read_csv('https://raw.githubusercontent.com/allisonhorst/palmerpenguins/master/inst/extdata/pengu
penguins.head()
```

Out[ ]:

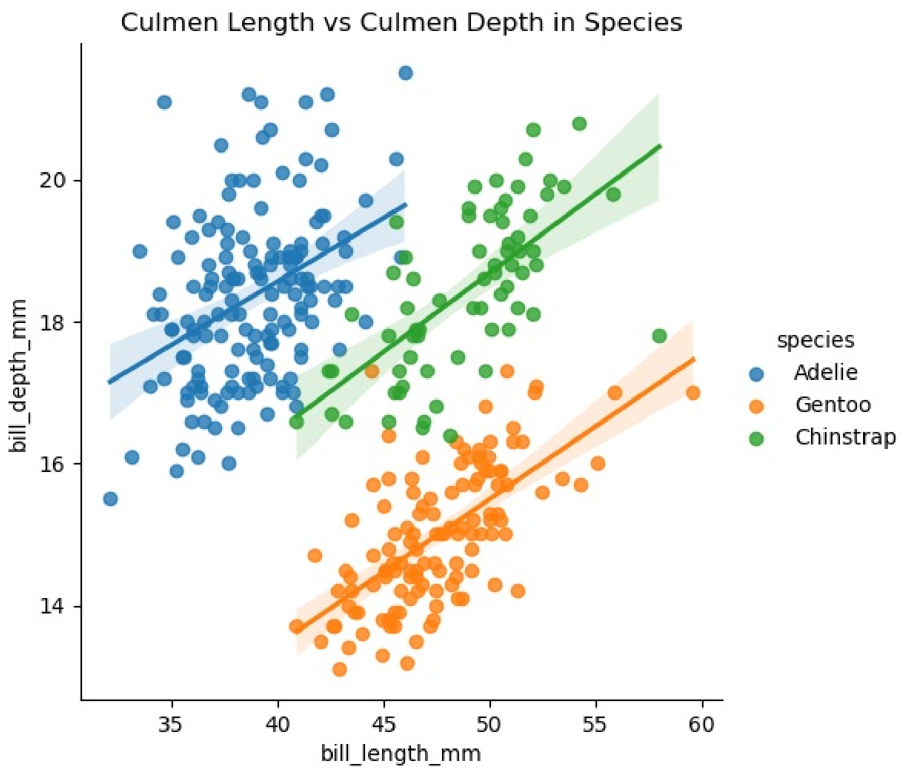
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	male	2007
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	female	2007
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	female	2007
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN	2007
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	female	2007

In [ ]:

```
# drop the year column, it is not useful for our analysis,
# and it has no adequate explanation in the dataset documentation
penguins.drop("year", axis=1, inplace=True)
```

In [ ]:

```
# Create a scatterplot of bill length vs bill depth using seaborn, hue by species.
# Add a title.
sns.lmplot(data=penguins, x="bill_length_mm", y="bill_depth_mm", hue="species").set(title="Culmen Length vs Cul
plt.show()
```



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

In [ ]:
```
# create a pipeline to impute missing values with the mean and scale numeric features
num_pipeline = Pipeline([
    ("impute", SimpleImputer(strategy="mean")),
    ("scale", StandardScaler())
])
# create a pipeline to impute missing values with the most frequent value and one-hot encode categorical featur
cat_pipeline = Pipeline([
    ("impute", SimpleImputer(strategy="most_frequent")),
    ("encode", OneHotEncoder())
])

# create a column transformer to apply the numeric and categorical pipelines to the correct features
# 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
# convert the result to a dataframe
# use the preprocessor's get_feature_names_out() method to get the column names
penguins_prepared = preprocessor.fit_transform(penguins)
df_penguins_prepared = pd.DataFrame(penguins_prepared, columns=preprocessor.get_feature_names_out())

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

Out[ ]:

	num__bill_length_mm	num__bill_depth_mm	num__flipper_length_mm	num__body_mass_g	cat__island_Biscoe	cat__island_Dream	cat__isl
0	-0.887081	0.787743	-1.422488	-0.565789	0.0	0.0	
1	-0.813494	0.126556	-1.065352	-0.503168	0.0	0.0	
2	-0.66632	0.431719	-0.422507	-1.192003	0.0	0.0	
3	-0.0	0.0	0.0	0.0	0.0	0.0	
4	-1.328605	1.092905	-0.565361	-0.941517	0.0	0.0	

In [ ]:
```
# separate the features from the target
# call the features X and the target y
X = df_penguins_prepared.drop("remainder__species", axis=1)
X_train, X_test = X[:275], X[275:]
y = df_penguins_prepared["remainder__species"]
y_train, y_test = y[:275], y[275:]
```

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_train_adelie = (y_train == "Adelie")
y_test_adelie = (y_test == "Adelie")
```

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

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

In [ ]:
```python
# compute the accuracy using cross_val_score with cv=10
acc_score = cross_val_score(SGDclass, X_train, y_train_adelie, cv=10, scoring="accuracy")
acc_score
```

Out[ ]:
```
array([1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 0.96296296, 1.        , 0.96296296])
```

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

Out[ ]:
```
0.9925925925925926
```

In [ ]:
```python
# predict the target using cross_val_predict with cv=10
# call the result y_train_pred
y_train_pred = cross_val_predict(SGDclass, X_train, y_train_adelie, cv=10)
y_train_pred
```

Out[ ]:
```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True, False,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
       False,  True, False, False, False])
```

In [ ]:
```python
# compute the confusion matrix
con_matrix = confusion_matrix(y_train_adelie, y_train_pred)
con_matrix
```

Out[ ]:
```
array([[122,   1],
       [  1, 151]])
```

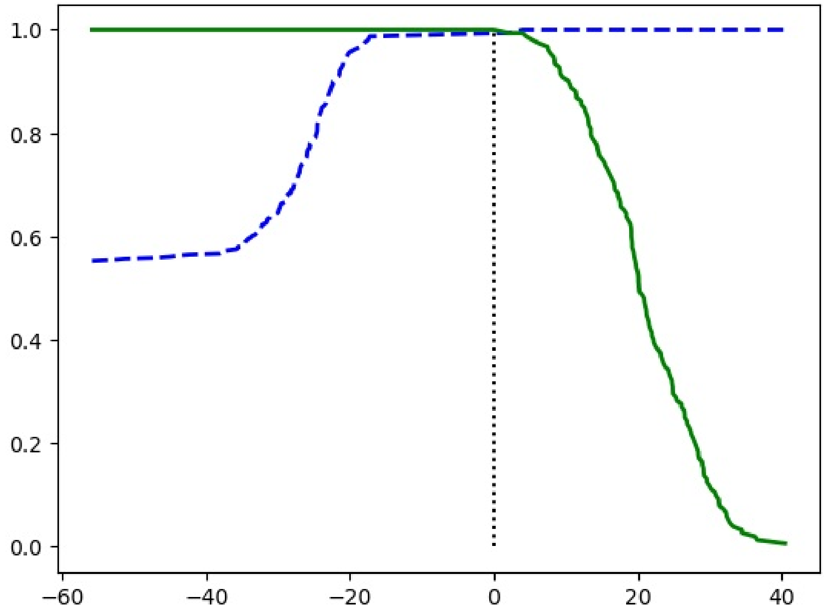
In [ ]:
```python
# compute the precision score using precision_score()
precision_score(y_train_adelie, y_train_pred)
```

Out[ ]:
```
0.993421052631579
```

In [ ]:
```python
# compute the recall score using recall_score()
recall_score(y_train_adelie, y_train_pred)
```

Out[ ]:
```
0.993421052631579
```

In [ ]:
```python
# draw the precision-recall curve
# call the result precisions, recalls, thresholds
y_train_score = cross_val_predict(SGDclass, X_train, y_train_adelie, cv=10, method="decision_function")
precisions, recalls, thresholds = precision_recall_curve(y_train_adelie, y_train_score)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision", linewidth=2)
plt.plot(thresholds, recalls[:-1], "g-", label="Recall", linewidth=2)
plt.vlines(0, 0, 1.0, "k", "dotted", label="threshold")
plt.show()
precisions, recalls, thresholds
```

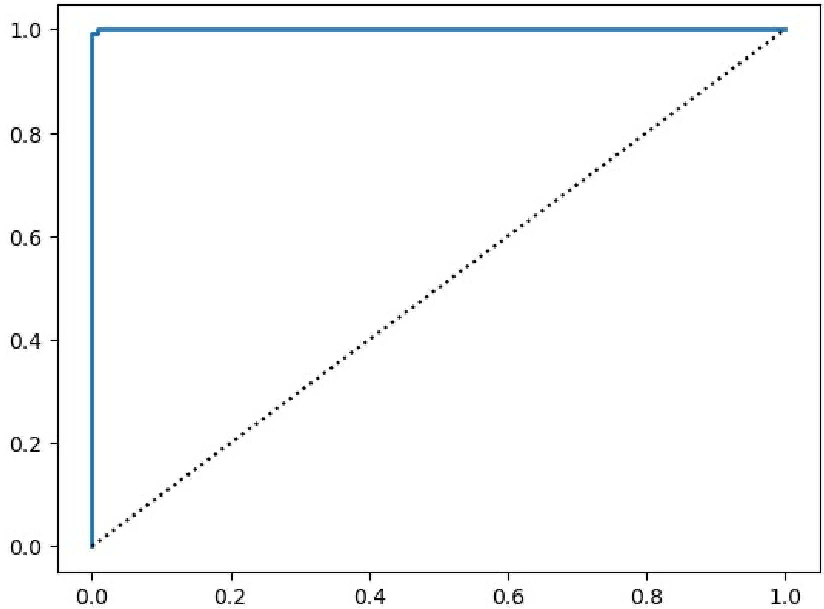



```
Out[ ]: (array([0.55272727, 0.55474453, 0.55677656, 0.55882353, 0.56088561,
        0.56296296, 0.56505576, 0.56716418, 0.56928839, 0.57142857,
        0.57358491, 0.57575758, 0.57794677, 0.58015267, 0.58237548,
        0.58461538, 0.58687259, 0.58914729, 0.59143969, 0.59375   ,
        0.59607843, 0.5984252 , 0.60079051, 0.6031746 , 0.60557769,
        0.608     , 0.61044177, 0.61290323, 0.61538462, 0.61788618,
        0.62040816, 0.62295082, 0.6255144 , 0.62809917, 0.63070539,
        0.63333333, 0.63598326, 0.63865546, 0.64135021, 0.6440678 ,
        0.64680851, 0.64957265, 0.65236052, 0.65517241, 0.65800866,
        0.66086957, 0.66375546, 0.66666667, 0.66960352, 0.67256637,
        0.67555556, 0.67857143, 0.68161435, 0.68468468, 0.68778281,
        0.69090909, 0.69406393, 0.69724771, 0.70046083, 0.7037037 ,
        0.70697674, 0.71028037, 0.71361502, 0.71698113, 0.72037915,
        0.72380952, 0.72727273, 0.73076923, 0.73429952, 0.73786408,
        0.74146341, 0.74509804, 0.74876847, 0.75247525, 0.75621891,
        0.76      , 0.7638191 , 0.76767677, 0.7715736 , 0.7755102 ,
        0.77948718, 0.78350515, 0.78756477, 0.79166667, 0.79581152,
        0.8       , 0.8042328 , 0.80851064, 0.81283422, 0.8172043 ,
        0.82162162, 0.82608696, 0.83060109, 0.83516484, 0.83977901,
        0.84444444, 0.84916201, 0.85393258, 0.85875706, 0.86363636,
        0.86857143, 0.87356322, 0.87861272, 0.88372093, 0.88888889,
        0.89411765, 0.89940828, 0.9047619 , 0.91017964, 0.91566265,
        0.92121212, 0.92682927, 0.93251534, 0.9382716 , 0.94409938,
        0.95      , 0.95597484, 0.96202532, 0.96815287, 0.97435897,
        0.98064516, 0.98701299, 0.99346405, 0.99342105, 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        ]),
 array([1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
        1.        , 1.        , 1.        , 1.        , 1.        ,
```

```
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 0.99342105, 0.99342105,
       0.98684211, 0.98026316, 0.97368421, 0.96710526, 0.96052632,
       0.95394737, 0.94736842, 0.94078947, 0.93421053, 0.92763158,
       0.92105263, 0.91447368, 0.90789474, 0.90131579, 0.89473684,
       0.88815789, 0.88157895, 0.875     , 0.86842105, 0.86184211,
       0.85526316, 0.84868421, 0.84210526, 0.83552632, 0.82894737,
       0.82236842, 0.81578947, 0.80921053, 0.80263158, 0.79605263,
       0.78947368, 0.78289474, 0.77631579, 0.76973684, 0.76315789,
       0.75657895, 0.75      , 0.74342105, 0.73684211, 0.73026316,
       0.72368421, 0.71710526, 0.71052632, 0.70394737, 0.69736842,
       0.69078947, 0.68421053, 0.67763158, 0.67105263, 0.66447368,
       0.65789474, 0.65131579, 0.64473684, 0.63815789, 0.63157895,
       0.625     , 0.61842105, 0.61184211, 0.60526316, 0.59868421,
       0.59210526, 0.58552632, 0.57894737, 0.57236842, 0.56578947,
       0.55921053, 0.55263158, 0.54605263, 0.53947368, 0.53289474,
       0.52631579, 0.51973684, 0.51315789, 0.50657895, 0.5       ,
       0.49342105, 0.48684211, 0.48026316, 0.47368421, 0.46710526,
       0.46052632, 0.45394737, 0.44736842, 0.44078947, 0.43421053,
       0.42763158, 0.42105263, 0.41447368, 0.40789474, 0.40131579,
       0.39473684, 0.38815789, 0.38157895, 0.375     , 0.36842105,
       0.36184211, 0.35526316, 0.34868421, 0.34210526, 0.33552632,
       0.32894737, 0.32236842, 0.31578947, 0.30921053, 0.30263158,
       0.29605263, 0.28947368, 0.28289474, 0.27631579, 0.26973684,
       0.26315789, 0.25657895, 0.25      , 0.24342105, 0.23684211,
       0.23026316, 0.22368421, 0.21710526, 0.21052632, 0.20394737,
       0.19736842, 0.19078947, 0.18421053, 0.17763158, 0.17105263,
       0.16447368, 0.15789474, 0.15131579, 0.14473684, 0.13815789,
       0.13157895, 0.125     , 0.11842105, 0.11184211, 0.10526316,
       0.09868421, 0.09210526, 0.08552632, 0.07894737, 0.07236842,
       0.06578947, 0.05921053, 0.05263158, 0.04605263, 0.03947368,
       0.03289474, 0.02631579, 0.01973684, 0.01315789, 0.00657895,
       0.        ]),
array([-5.59117904e+01, -5.29892985e+01, -5.12665899e+01, -4.69096883e+01,
       -4.50887578e+01, -4.39240932e+01, -4.24503572e+01, -3.81266654e+01,
       -3.78170364e+01, -3.78060386e+01, -3.69277434e+01, -3.57827585e+01,
       -3.55931392e+01, -3.55768265e+01, -3.48102965e+01, -3.48042357e+01,
       -3.47191974e+01, -3.46313509e+01, -3.43753694e+01, -3.42951277e+01,
       -3.40999981e+01, -3.38087329e+01, -3.36847602e+01, -3.32807197e+01,
       -3.30771262e+01, -3.28715091e+01, -3.27195897e+01, -3.26506389e+01,
       -3.23624056e+01, -3.23300086e+01, -3.22444867e+01, -3.22410086e+01,
       -3.18086532e+01, -3.15476353e+01, -3.15422917e+01, -3.15000868e+01,
       -3.11900369e+01, -3.10817792e+01, -3.06078557e+01, -3.02590565e+01,
       -2.99935781e+01, -2.99544696e+01, -2.99333100e+01, -2.97721244e+01,
       -2.97421480e+01, -2.96973315e+01, -2.95569854e+01, -2.89683643e+01,
       -2.88953684e+01, -2.88748414e+01, -2.88341882e+01, -2.86570625e+01,
       -2.86438383e+01, -2.82387918e+01, -2.82285747e+01, -2.81348787e+01,
       -2.78736811e+01, -2.78614299e+01, -2.78067118e+01, -2.77871256e+01,
       -2.77599661e+01, -2.74589588e+01, -2.73690990e+01, -2.72263491e+01,
       -2.71902973e+01, -2.70189660e+01, -2.70131261e+01, -2.69741811e+01,
       -2.69067560e+01, -2.68018886e+01, -2.65401903e+01, -2.63989143e+01,
       -2.61846302e+01, -2.60388518e+01, -2.59818934e+01, -2.59710419e+01,
       -2.59488159e+01, -2.58549008e+01, -2.56348581e+01, -2.56044555e+01,
       -2.55747594e+01, -2.53259857e+01, -2.51558286e+01, -2.48097734e+01,
       -2.47651197e+01, -2.46650380e+01, -2.45996555e+01, -2.45960267e+01,
       -2.45900295e+01, -2.45682127e+01, -2.44900657e+01, -2.43831306e+01,
       -2.43066260e+01, -2.42190266e+01, -2.41108313e+01, -2.39968193e+01,
       -2.39563674e+01, -2.35426480e+01, -2.33059002e+01, -2.32232429e+01,
       -2.29914909e+01, -2.29884259e+01, -2.28640749e+01, -2.27368298e+01,
       -2.26091490e+01, -2.24151458e+01, -2.23809319e+01, -2.20061252e+01,
       -2.15513936e+01, -2.14603408e+01, -2.14526822e+01, -2.11964280e+01,
       -2.09729901e+01, -2.09659638e+01, -2.06441596e+01, -2.03768799e+01,
       -2.01101134e+01, -1.93738725e+01, -1.83093499e+01, -1.80131597e+01,
       -1.74562897e+01, -1.72493338e+01, -3.18520981e-02,  2.12432046e+00,
        3.99809858e+00,  4.52734496e+00,  5.38912539e+00,  6.26265789e+00,
        7.48103200e+00,  7.57909799e+00,  7.90225322e+00,  8.32800205e+00,
        8.43281568e+00,  8.48885054e+00,  9.07906174e+00,  9.17651953e+00,
        9.21642561e+00,  9.51802255e+00,  1.03555841e+01,  1.05849328e+01,
        1.07379263e+01,  1.13647816e+01,  1.14534996e+01,  1.14846191e+01,
        1.21044099e+01,  1.21670046e+01,  1.26164856e+01,  1.26989027e+01,
        1.29138038e+01,  1.30850791e+01,  1.31487898e+01,  1.32115162e+01,
        1.34895984e+01,  1.35069325e+01,  1.35099691e+01,  1.37328979e+01,
        1.40770817e+01,  1.43346420e+01,  1.44129555e+01,  1.45580102e+01,
```

```
1.46548979e+01,  1.50234514e+01,  1.53565901e+01,  1.55205427e+01,
1.58061160e+01,  1.60067744e+01,  1.62498593e+01,  1.64713458e+01,
1.67091662e+01,  1.67963772e+01,  1.68540826e+01,  1.72287713e+01,
1.72874054e+01,  1.74864093e+01,  1.75801713e+01,  1.76143930e+01,
1.81248240e+01,  1.84153955e+01,  1.84208659e+01,  1.87102430e+01,
1.89777183e+01,  1.90935870e+01,  1.91297794e+01,  1.91322135e+01,
1.91378380e+01,  1.92148076e+01,  1.92490551e+01,  1.92558736e+01,
1.94274207e+01,  1.94939235e+01,  1.95720679e+01,  1.96889763e+01,
1.97797733e+01,  1.98378474e+01,  1.99925593e+01,  2.00835213e+01,
2.01208514e+01,  2.01222641e+01,  2.01563737e+01,  2.02018253e+01,
2.02427815e+01,  2.06348791e+01,  2.08842837e+01,  2.09222335e+01,
2.10134904e+01,  2.11590643e+01,  2.12147083e+01,  2.12447279e+01,
2.14278494e+01,  2.15539640e+01,  2.15949195e+01,  2.17718541e+01,
2.19773468e+01,  2.19957306e+01,  2.22015119e+01,  2.22727409e+01,
2.24851571e+01,  2.28092522e+01,  2.31878007e+01,  2.32524926e+01,
2.34113442e+01,  2.35494394e+01,  2.37746472e+01,  2.42796113e+01,
2.43759492e+01,  2.45340064e+01,  2.47607355e+01,  2.48026741e+01,
2.48898295e+01,  2.49131672e+01,  2.49360939e+01,  2.53571168e+01,
2.54601013e+01,  2.60774947e+01,  2.60906817e+01,  2.64680745e+01,
2.65225734e+01,  2.65378228e+01,  2.68432823e+01,  2.68906403e+01,
2.71843162e+01,  2.73832990e+01,  2.74811167e+01,  2.77192097e+01,
2.79035371e+01,  2.81121608e+01,  2.82567645e+01,  2.84397326e+01,
2.84486382e+01,  2.85507119e+01,  2.89604205e+01,  2.90056237e+01,
2.91373625e+01,  2.91730388e+01,  2.92040144e+01,  2.96177193e+01,
2.96201465e+01,  2.99657726e+01,  3.01934647e+01,  3.07814265e+01,
3.09261061e+01,  3.12792404e+01,  3.13077137e+01,  3.13791164e+01,
3.19185835e+01,  3.23256150e+01,  3.23578530e+01,  3.25987227e+01,
3.28236956e+01,  3.32698257e+01,  3.44453308e+01,  3.45458081e+01,
3.63568089e+01,  3.66534677e+01,  4.04687247e+01]))
```

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



In [ ]:
```
# now let's do multiclass classification
# build an SGDClassifier model using X and y
# use random_state=42 for reproducibility
multi_cls = SGDClassifier(random_state=42)
multi_cls.fit(X_train, y_train)
```

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

In [ ]:
```
# show the mean accuracy using cross_val_score with cv=10
np.mean(cross_val_score(multi_cls, X_train, y_train, cv=10, scoring="accuracy"))
```

Out[ ]:
```
0.9925925925925926
```

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(multi_cls, X_train, y_train, cv=10)
y_train_pred
cm = confusion_matrix(y_train, y_train_pred)
cm
```

Out[ ]:
```
array([[151,   1],
       [  1, 122]])
```

In [ ]:
```
# use ConfusionMatrixDisplay to display the confusion matrix
ConfusionMatrixDisplay.from_predictions(y_train, y_train_pred)
plt.show()
```