Iris Dataset

Use sklearn.datasets iris flower dataset to train your model using logistic regression. You need to figure out accuracy of your model and use that to predict different samples in your test dataset. In iris dataset there are 150 samples containing following features,

1. Sepal Length
2. Sepal Width
3. Petal Length
4. Petal Width

Using above 4 features you will clasify a flower in one of the three categories,

1. Setosa
2. Versicolour
3. Virginica

```
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import numpy as np
import seaborn as sns
%matplotlib inline
```

```
iris = load_iris()
```

```
dir(iris)
```

```
['DESCR',
 'data',
 'data_module',
 'feature_names',
 'filename',
 'frame',
 'target',
 'target_names']
```

```
iris.data[[0,1]]
```

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2]])
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Next steps: Generate code with df   View recommended plots   New interactive sheet

```
df.species.unique()
```

```
array([0, 1, 2])
```

```
df['species_name'] = df['species'].apply(lambda x: iris.target_names[x])
df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	species_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
X = df.drop(['species', 'species_name'], axis='columns')
y = df.species
```

```
model = LogisticRegression(max_iter=2000)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

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

```
▾   LogisticRegression
LogisticRegression(max_iter=2000)
```

```
y_predicted = model.predict(X_test)
y_predicted
```

```
array([0, 2, 1, 2, 2, 1, 0, 2, 0, 2, 1, 0, 2, 1, 1, 0, 2, 2, 1, 2, 0, 0,
       0, 0, 0, 2, 1, 0, 0, 2])
```

```
model.score(X_test, y_test)
```

```
0.9333333333333333
```

```
model.coef_, model.intercept_
```

```
(array([[-0.40393932,  0.92584369, -2.39870027, -1.00048259],
        [ 0.5312926 , -0.4115079 , -0.28319445, -0.70078918],
        [-0.12735328, -0.51433578,  2.68189472,  1.70127177]]),
 array([  9.30432111,   2.46066921, -11.76499032]))
```

```
y_test_df = pd.DataFrame(y_test).reset_index(drop=True)
y_pred_df = pd.DataFrame(y_predicted, columns=['y_predicted'])

# Combine X_test, y_test, and predictions
final_df = pd.concat([X_test.reset_index(drop=True),
                      y_test_df,
                      y_pred_df],
                     axis=1)
final_df.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	y_predicted
0	4.9	3.6	1.4	0.1	0	0
1	6.3	2.5	5.0	1.9	2	2
2	6.0	3.4	4.5	1.6	1	1
3	7.7	3.0	6.1	2.3	2	2
4	6.0	3.0	4.8	1.8	2	2

Next steps: Generate code with final_df | View recommended plots | New interactive sheet

```
final_df.tail()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	y_predicted
25	7.7	2.6	6.9	2.3	2	2

```
test = [[6.9, 2.0, 3.1, 1.8]]
# Convert test data to DataFrame with feature names
test_df = pd.DataFrame(test, columns=iris.feature_names)
predicted_species_encoded = model.predict(test_df)
predicted_species_name = iris.target_names[predicted_species_encoded][0]
print(predicted_species_name)
```

```
versicolor
```

```
correct_count = (y_test_df.iloc[:, 0] == y_pred_df['y_predicted']).sum()
incorrect_count = (y_test_df.iloc[:, 0] != y_pred_df['y_predicted']).sum()

print("Correct predictions:", correct_count)
print("Incorrect predictions:", incorrect_count)
```

```
Correct predictions: 28
Incorrect predictions: 2
```

```
# 1. Accuracy
acc = accuracy_score(y_test, y_predicted)
print(f"1. Accuracy: {acc:.4f}\n")

# 2. Confusion Matrix
cm = confusion_matrix(y_test, y_predicted)
print("2. Confusion Matrix:")
print(cm, "\n")

# 3. Classification Report
report = classification_report(y_test, y_predicted)
print("3. Classification Report:")
print(report)
```

```
1. Accuracy: 0.9333

2. Confusion Matrix:
[[12  0  0]
 [ 0  6  1]
 [ 0  1 10]]

3. Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       0.86      0.86      0.86         7
           2       0.91      0.91      0.91        11

    accuracy                           0.93        30
   macro avg       0.92      0.92      0.92        30
weighted avg       0.93      0.93      0.93        30
```