Load the penguin's dataset using the following code.

```python
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the penguins dataset
df = sns.load_dataset("penguins")

df.dropna(inplace=True)

# Filter rows for 'Adelie' and 'Chinstrap' classes
selected_classes = ['Adelie', 'Chinstrap']
df_filtered = df[df['species'].isin(selected_classes)].copy()  # Make
a copy to avoid the warning

# Initialize the LabelEncoder
le = LabelEncoder()

# Encode the species column
y_encoded = le.fit_transform(df_filtered['species'])

df_filtered['class_encoded'] = y_encoded

# Display the filtered and encoded DataFrame
print(df_filtered[['species', 'class_encoded']])

# Split the data into features (X) and target variable (y)

y = df_filtered['class_encoded']  # Target variable
X = df_filtered.drop(['species', 'island', 'sex','class_encoded'],
axis=1)
```

```
       species  class_encoded
0       Adelie              0
1       Adelie              0
2       Adelie              0
4       Adelie              0
5       Adelie              0
..         ...            ...
215  Chinstrap              1
216  Chinstrap              1
217  Chinstrap              1
218  Chinstrap              1
219  Chinstrap              1

[214 rows x 2 columns]
```

1.What is the purpose of "y_encoded = le.fit_transform(df_filtered['species'])" ?

The purpose of this line of code is to encode the target variable 'species' into numerical labels. In machine learning, algorithms often require the target variable to be in numerical form for training. The LabelEncoder from scikit-learn is used to convert the species names (e.g., 'Adelie' and 'Chinstrap') into corresponding numerical labels (e.g., 0 and 1), which can be used for classification

2.What is the purpose of "X = df.drop(['species', 'island', 'sex'], axis=1)" ?

X = df_filtered.drop(['species', 'island', 'sex'], axis=1) creates the feature matrix X by dropping the columns 'species', 'island', and 'sex' from the DataFrame. These columns are removed because they are categorical and not directly usable as features for logistic regression. You typically need to convert categorical variables into numerical representations, or in some cases, perform one-hot encoding, which you later do in the code.

This line of code is used to create the feature matrix 'X' by dropping the columns 'species', 'island', and 'sex' from the original DataFrame 'df'. These columns are typically not used as features for classification in this specific context. The resulting 'X' contains only the numeric features that will be used to train the logistic regression model.

3.Why we cannot use "island" and "sex" features?

"Island" and "sex" are categorical features. While some machine learning algorithms can handle categorical data directly, logistic regression typically requires numeric input features. To use categorical features in logistic regression, they need to be one-hot encoded or otherwise transformed into numeric representations. In this code, these columns are dropped instead of being one-hot encoded, which is a common preprocessing step when dealing with categorical data.

Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Train the logistic regression model. Here we are using saga solver to learn weights.

```
logreg = LogisticRegression(solver='saga')

logreg.fit(X_train, y_train)

# Predict on the testing data
y_pred = logreg.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

print(logreg.coef_, logreg.intercept_)
```

```
Accuracy: 0.5813953488372093
[[ 2.75633615e-03 -8.08986406e-05  4.77783153e-04 -2.87299611e-04]] [-
8.39446233e-06]

c:\Users\MSI\anaconda3\Lib\site-packages\sklearn\linear_model\
_sag.py:350: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
```

4.Why is accuracy low? why does the saga solver perform poorly?

The accuracy of the logistic regression model with the 'saga' solver might be low because the 'saga' solver is sensitive to feature scaling. If features are not properly scaled, it can affect the convergence of the algorithm and lead to suboptimal results. This is why you observe an increase in accuracy when switching to the 'liblinear' solver, which is less sensitive to feature scaling.

The initial accuracy might be low for several reasons:

The features might not be well-suited for classification.

There could be class imbalance in the dataset.

The choice of solver ('saga') may not be optimal for this specific dataset.

The 'saga' solver might perform poorly in this case because it's sensitive to the scale of the features, and logistic regression generally benefits from feature scaling. If the features are not properly scaled, it can affect the convergence and performance of the solver.

Change the solver to "liblinear"

```
logreg = LogisticRegression(solver='liblinear')
logreg.fit(X_train, y_train)
# Predict on the testing data
y_pred = logreg.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

print(logreg.coef_, logreg.intercept_)

Accuracy: 1.0
[[ 1.61343591 -1.4665703  -0.15152349 -0.00398479]] [-0.08866849]
```

5.Why is accuracy now? why does the "liblinear" solver perform better than "saga" solver ?

Changing the solver to "liblinear" often improves accuracy, especially if the data is not well-scaled. "liblinear" is more robust to unscaled features, and it's a good choice when you have a small dataset or when other solvers do not perform well. Accuracy may increase because "liblinear" can handle the data better in its original scale.

Repeat the above tasks after feature normalization and observe the accuracy levels.

```
from sklearn.preprocessing import StandardScaler
# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

logreg = LogisticRegression(solver='saga')
logreg.fit(X_train_scaled, y_train)

# Predict on the testing data
y_pred = logreg.predict(X_test_scaled)
```

6.Now observe the accuracies for both "liblinear" solver and "saga" solver. Why accuracy of the "saga" solver is increased?

Normalizing the features using StandardScaler scales them to have a mean of 0 and a standard deviation of 1. This can help the 'saga' solver converge faster and perform better because it reduces the impact of feature scales on the optimization process. Normalization often makes the solver more stable and effective, especially when features have different scales.

Extra

The accuracy of the "saga" solver may have increased after feature normalization (using the "MaxAbsScaler") because feature scaling can have a significant impact on the performance of logistic regression, especially when using the "saga" solver. Here's why the accuracy of the "saga" solver might have improved:

Feature Scaling: The "saga" solver is sensitive to the scale of the features. When features have different scales, it can lead to slow convergence or convergence to suboptimal solutions. In the original, unscaled data, features like "bill_length_mm" and "bill_depth_mm" could have different scales. Scaling these features to have similar magnitudes can help the solver converge more quickly and reach a better solution.

Normalization Effect: Normalization, in this case using "MaxAbsScaler," can improve the condition of the optimization problem. It ensures that each feature has values within a similar range ([-1, 1]), making the optimization landscape more favorable. This can help the "saga" solver find a better decision boundary, leading to improved classification accuracy.

Reducing Numerical Instabilities: Feature scaling can reduce numerical instabilities that might occur during the optimization process. When features are on different scales, it can cause problems like floating-point overflows or underflows, which can affect the solver's performance. Scaling mitigates these issues.

Convergence: The "saga" solver is designed for large datasets and can handle both L1 and L2 regularization. In some cases, with scaled features, it may converge more efficiently and find a solution that separates the classes better.

Run the following code to load the dataset again, and use logistic regression for the classification.

```python
import seaborn as sns
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# Load the penguins dataset
df = sns.load_dataset("penguins")

df.dropna(inplace=True)

# Filter rows for 'Adelie' and 'Chinstrap' classes
selected_classes = ['Adelie', 'Chinstrap']
df_filtered = df[df['species'].isin(selected_classes)].copy()  # Make
a copy to avoid the warning

# Initialize the LabelEncoder
le = LabelEncoder()

# Encode the species column
y_encoded = le.fit_transform(df_filtered['species'])
df_filtered['class_encoded'] = y_encoded


df_filtered.head()

X = df_filtered.drop(['species', 'class_encoded'], axis=1)  # Choose
features
y = df_filtered['class_encoded']  # Target variable

X.head()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

logreg = LogisticRegression(solver='saga')

#logreg = LogisticRegression(max_iter=166, solver='newton-cg')
# logreg = LogisticRegression(penalty='l2', C=1.0, solver='lbfgs',
max_iter=100, multi_class='ovr', random_state=42)
logreg.fit(X_train, y_train)

# Predict on the testing data
y_pred = logreg.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(logreg.coef_, logreg.intercept_)

-----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[10], line 40
     36 logreg = LogisticRegression(solver='saga')
     38 #logreg = LogisticRegression(max_iter=166, solver='newton-cg')
     39 # logreg = LogisticRegression(penalty='l2', C=1.0,
solver='lbfgs', max_iter=100, multi_class='ovr', random_state=42)
---> 40 logreg.fit(X_train, y_train)
     42 # Predict on the testing data
     43 y_pred = logreg.predict(X_test)

File c:\Users\MSI\anaconda3\Lib\site-packages\sklearn\base.py:1151, in
_fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args,
**kwargs)
   1144     estimator._validate_params()
   1146 with config_context(
   1147     skip_parameter_validation=(
   1148         prefer_skip_nested_validation or
global_skip_validation
   1149     )
   1150 ):
-> 1151     return fit_method(estimator, *args, **kwargs)

File c:\Users\MSI\anaconda3\Lib\site-packages\sklearn\linear_model\
_logistic.py:1207, in LogisticRegression.fit(self, X, y,
sample_weight)
   1204 else:
   1205     _dtype = [np.float64, np.float32]
-> 1207 X, y = self._validate_data(
   1208     X,
   1209     y,
   1210     accept_sparse="csr",
   1211     dtype=_dtype,
   1212     order="C",
   1213     accept_large_sparse=solver not in ["liblinear", "sag",
"saga"],
   1214 )
   1215 check_classification_targets(y)
   1216 self.classes_ = np.unique(y)

File c:\Users\MSI\anaconda3\Lib\site-packages\sklearn\base.py:621, in
BaseEstimator._validate_data(self, X, y, reset, validate_separately,
cast_to_ndarray, **check_params)
    619         y = check_array(y, input_name="y", **check_y_params)
    620     else:
```

```
--> 621         X, y = check_X_y(X, y, **check_params)
    622     out = X, y
    624 if not no_val_X and check_params.get("ensure_2d", True):

File c:\Users\MSI\anaconda3\Lib\site-packages\sklearn\utils\
validation.py:1147, in check_X_y(X, y, accept_sparse,
accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d,
allow_nd, multi_output, ensure_min_samples, ensure_min_features,
y_numeric, estimator)
   1142         estimator_name = _check_estimator_name(estimator)
   1143     raise ValueError(
   1144         f"{estimator_name} requires y to be passed, but the
target y is None"
   1145     )
-> 1147 X = check_array(
   1148     X,
   1149     accept_sparse=accept_sparse,
   1150     accept_large_sparse=accept_large_sparse,
   1151     dtype=dtype,
   1152     order=order,
   1153     copy=copy,
   1154     force_all_finite=force_all_finite,
   1155     ensure_2d=ensure_2d,
   1156     allow_nd=allow_nd,
   1157     ensure_min_samples=ensure_min_samples,
   1158     ensure_min_features=ensure_min_features,
   1159     estimator=estimator,
   1160     input_name="X",
   1161 )
   1163 y = _check_y(y, multi_output=multi_output,
y_numeric=y_numeric, estimator=estimator)
   1165 check_consistent_length(X, y)

File c:\Users\MSI\anaconda3\Lib\site-packages\sklearn\utils\
validation.py:917, in check_array(array, accept_sparse,
accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d,
allow_nd, ensure_min_samples, ensure_min_features, estimator,
input_name)
    915         array = xp.astype(array, dtype, copy=False)
    916     else:
--> 917         array = _asarray_with_order(array, order=order,
dtype=dtype, xp=xp)
    918 except ComplexWarning as complex_warning:
    919     raise ValueError(
    920         "Complex data not supported\n{}\n".format(array)
    921     ) from complex_warning

File c:\Users\MSI\anaconda3\Lib\site-packages\sklearn\utils\
_array_api.py:380, in _asarray_with_order(array, dtype, order, copy,
xp)
```

```
    378     array = numpy.array(array, order=order, dtype=dtype)
    379 else:
--> 380     array = numpy.asarray(array, order=order, dtype=dtype)
    382 # At this point array is a NumPy ndarray. We convert it to an
array
    383 # container that is consistent with the input's namespace.
    384 return xp.asarray(array)

File c:\Users\MSI\anaconda3\Lib\site-packages\pandas\core\
generic.py:2070, in NDFrame.__array__(self, dtype)
   2069 def __array__(self, dtype: npt.DTypeLike | None = None) ->
np.ndarray:
-> 2070     return np.asarray(self._values, dtype=dtype)

ValueError: could not convert string to float: 'Dream'
```

Reason for the above error

The error we encountered is due to the presence of non-numeric (string) values in your DataFrame columns. In particular, the issue is related to the 'island','sex' columns, which contais categorical string values. Logistic Regression, like many other machine learning models, requires numeric input features.

Changes made to address the error:

Import OneHotEncoder from sklearn.preprocessing.

Added 'sex' column to the list of columns to be one-hot encoded: pd.get_dummies(df_filtered, columns=['island', 'sex'], drop_first=True).

Removed any references to the original 'sex' column after encoding since it's no longer needed for modeling.

7.What is the problem? Why algorithm cannot perform classification?

The problem is that the categorical features 'island' and 'sex' have not been transformed into numerical representations (e.g., one-hot encoding or label encoding). Logistic regression, as implemented here, requires numerical features. Without encoding these categorical features, the algorithm cannot process them, leading to errors or poor performance.

8.How to solve this issue??

To solve the issue in the second code block, you should transform the categorical features 'island' and 'sex' into numerical representations. One common approach is to use one-hot encoding, which creates binary columns for each category. You can use the following code to perform one-hot encoding:

df_filtered = pd.get_dummies(df_filtered, columns=['island', 'sex'], drop_first=True)

```python
import seaborn as sns
import pandas as pd
```

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score


# Load the penguins dataset
df = sns.load_dataset("penguins")

df.dropna(inplace=True)

# Filter rows for 'Adelie' and 'Chinstrap' classes
selected_classes = ['Adelie', 'Chinstrap']
df_filtered = df[df['species'].isin(selected_classes)].copy()  # Make
a copy to avoid the warning

# Initialize the LabelEncoder
le = LabelEncoder()

# Encode the species column
y_encoded = le.fit_transform(df_filtered['species'])
df_filtered['class_encoded'] = y_encoded

# One-hot encode the 'island' and 'sex' columns
df_filtered = pd.get_dummies(df_filtered, columns=['island', 'sex'],
drop_first=True)

df_filtered.head()

X = df_filtered.drop(['species', 'class_encoded'], axis=1)  # Choose
features
y = df_filtered['class_encoded']  # Target variable

X.head()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

logreg = LogisticRegression(solver='saga')

#logreg = LogisticRegression(max_iter=166, solver='newton-cg')
# logreg = LogisticRegression(penalty='l2', C=1.0, solver='lbfgs',
max_iter=100, multi_class='ovr', random_state=42)
logreg.fit(X_train, y_train)

# Predict on the testing data
y_pred = logreg.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)

print(logreg.coef_, logreg.intercept_)
```

```
Accuracy: 0.5813953488372093
[[ 2.75545030e-03 -8.65124398e-05  4.49807168e-04 -2.85834541e-04
   1.85223496e-04 -1.04988406e-04  1.07637223e-05]] [-8.6399497e-06]

c:\Users\MSI\anaconda3\Lib\site-packages\sklearn\linear_model\
_sag.py:350: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
  warnings.warn(
```

Use the following code to visualize the encoding

```
samples = df_filtered.groupby('sex_Male').head(1)
print(samples)
print()
samples = df_filtered.groupby('island_Torgersen').head(1)
print(samples)
print()
samples = df_filtered.groupby('island_Dream').head(1)
print(samples)
```

```
  species  bill_length_mm  bill_depth_mm  flipper_length_mm
body_mass_g  \
0  Adelie            39.1           18.7              181.0
3750.0
1  Adelie            39.5           17.4              186.0
3800.0

   class_encoded  island_Dream  island_Torgersen  sex_Male
0              0             0                 1         1
1              0             0                 1         0

   species  bill_length_mm  bill_depth_mm  flipper_length_mm
body_mass_g  \
0   Adelie            39.1           18.7              181.0
3750.0
20  Adelie            37.8           18.3              174.0
3400.0

    class_encoded  island_Dream  island_Torgersen  sex_Male
0               0             0                 1         1
20              0             0                 0         0

   species  bill_length_mm  bill_depth_mm  flipper_length_mm
body_mass_g  \
0   Adelie            39.1           18.7              181.0
3750.0
```

```
30  Adelie            39.5           16.7              178.0
3250.0

    class_encoded  island_Dream  island_Torgersen  sex_Male
0               0             0                 1         1
30              0             1                 0         0
```

Use the following code to apply logistic regression

```
X = df_filtered.drop(['species','class_encoded'], axis=1)

y = df_filtered['class_encoded']  # Target variable
print(X.shape, y.shape)
X.head()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
from sklearn.preprocessing import MaxAbsScaler
scaler=MaxAbsScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

logreg = LogisticRegression(solver='saga',max_iter=150,)

#logreg = LogisticRegression(max_iter=166, solver='newton-cg')
# logreg = LogisticRegression(penalty='l2', C=1.0, solver='lbfgs',
max_iter=100, multi_class='ovr', random_state=42)
logreg.fit(X_train_scaled, y_train)

# Predict on the testing data
y_pred = logreg.predict(X_test_scaled)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

print(logreg.coef_, logreg.intercept_)

(214, 7) (214,)
Accuracy: 1.0
[[ 3.63420412  0.16314238  0.62632368  0.10221005  2.59927011 -
0.87718394
  -0.35907275]] [-5.99637185]
```

Why we are using the "MaxAbsScaler" scaler rather than the "StandardScaler"?

The choice of using "MaxAbsScaler" versus "StandardScaler" depends on the nature of the data and the goals of scaling. "MaxAbsScaler" scales the features by dividing each feature by its maximum absolute value. This scaler is useful when you want to preserve the sparsity of the data, as it doesn't shift the distribution of the data or affect the mean and variance.

In contrast, "StandardScaler" standardizes the features to have a mean of 0 and a standard deviation of 1, which can be beneficial when features have different scales and you want to give them equal importance.

Use the following code to visualize feature scaling before and after normalization.

```
from sklearn.preprocessing import MaxAbsScaler
scaler=MaxAbsScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print(X_test_scaled)

from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print(X_test_scaled)
```

```
[[0.59655172 0.98139535 0.93396226 0.91666667 0.         1.
  1.        ]
 [0.8862069  0.88372093 0.94811321 0.82291667 1.         0.
  1.        ]
 [0.68275862 0.8        0.9245283  0.73958333 0.         1.
  0.        ]
 [0.87586207 0.86046512 0.94811321 0.92708333 1.         0.
  1.        ]
 [0.7137931  0.86046512 0.95283019 0.80729167 0.         1.
  1.        ]
 [0.64310345 0.95348837 0.93867925 0.78645833 0.         1.
  1.        ]
 [0.65172414 0.85116279 0.82075472 0.70833333 0.         0.
  0.        ]
 [0.5862069  0.79534884 0.87264151 0.70833333 1.         0.
  0.        ]
 [0.73965517 0.81860465 0.9245283  0.97916667 0.         1.
  1.        ]
 [0.62068966 0.79534884 0.88207547 0.77083333 1.         0.
  0.        ]
 [0.82068966 0.85116279 0.91981132 0.80208333 1.         0.
  0.        ]
 [0.80517241 0.83255814 0.91981132 0.6875     1.         0.
  0.        ]
 [0.63103448 0.85581395 0.86792453 0.72395833 1.         0.
  0.        ]
 [0.72586207 0.88837209 0.91981132 0.83333333 0.         1.
  1.        ]
 [0.73275862 0.77674419 0.88207547 0.69791667 1.         0.
```

```
  0.        ]
 [0.8362069  0.81395349 0.9009434  0.70833333 1.         0.
  1.        ]
 [0.87758621 0.83255814 0.9245283  0.765625   1.         0.
  0.        ]
 [0.65862069 0.84186047 0.87264151 0.82291667 0.         0.
  1.        ]
 [0.69137931 0.87906977 0.88679245 0.89583333 0.         0.
  1.        ]
 [0.80862069 0.77209302 0.90566038 0.5625     1.         0.
  0.        ]
 [0.65344828 0.86511628 0.91037736 0.609375   0.         0.
  0.        ]
 [0.96206897 0.92093023 0.97641509 0.83333333 1.         0.
  1.        ]
 [0.85517241 0.84651163 0.91037736 0.78645833 1.         0.
  1.        ]
 [0.85862069 0.80465116 0.93396226 0.765625   1.         0.
  0.        ]
 [0.88448276 0.89302326 0.91037736 0.76041667 1.         0.
  1.        ]
 [0.65689655 0.76744186 0.93396226 0.796875   0.         0.
  0.        ]
 [0.67586207 0.98139535 0.9245283  0.86458333 1.         0.
  1.        ]
 [0.68103448 0.77674419 0.83962264 0.67708333 1.         0.
  0.        ]
 [0.91034483 0.93023256 0.96698113 0.94791667 1.         0.
  1.        ]
 [0.65       0.86976744 0.8490566  0.75       0.         0.
  1.        ]
 [0.71206897 0.98139535 0.91981132 0.91666667 0.         0.
  1.        ]
 [0.64310345 0.78139535 0.90566038 0.625      1.         0.
  0.        ]
 [0.81034483 0.80465116 0.87264151 0.77083333 1.         0.
  0.        ]
 [0.74482759 0.86046512 0.90566038 0.85416667 1.         0.
  1.        ]
 [0.73793103 0.86046512 0.91981132 0.88541667 0.         1.
  1.        ]
 [0.85       0.9255814  0.95754717 0.84375    1.         0.
  1.        ]
 [0.89655172 0.88372093 0.92924528 0.86458333 1.         0.
  1.        ]
 [0.79482759 0.84651163 0.83962264 0.67708333 1.         0.
  0.        ]
 [0.65172414 0.93023256 0.89622642 0.88541667 0.         0.
  1.        ]
 [0.61551724 0.8372093  0.95283019 0.73958333 1.         0.
  0.        ]
 [0.65862069 0.93023256 0.89622642 0.8125     0.         0.
  1.        ]
 [0.73275862 0.80465116 0.88207547 0.69791667 1.         0.
  0.        ]
 [0.61206897 0.75348837 0.91981132 0.69791667 0.         0.
  0.        ]]
[[-1.34082659  2.35505035  0.88068888  1.62029553 -1.14490646
1.80969611
   1.01770049]
 [ 1.80078227  0.5480021   1.30057122  0.57562238  0.8734338  -
0.55257896
   1.01770049]
 [-0.40582395 -1.0008964   0.60076732 -0.35297599 -1.14490646
1.80969611
  -0.98260737]
 [ 1.68858195  0.11775252  1.30057122  1.73637033  0.8734338  -
0.55257896
   1.01770049]
 [-0.069223    0.11775252  1.440532    0.40151018 -1.14490646
1.80969611
   1.01770049]
 [-0.83592516  1.83875085  1.02064966  0.16936059 -1.14490646
1.80969611
   1.01770049]
 [-0.7424249  -0.05434732 -2.47836983 -0.70120037 -1.14490646 -
0.55257896
  -0.98260737]
 [-1.4530269  -1.08694632 -0.93880125 -0.70120037  0.8734338  -
0.55257896
  -0.98260737]
 [ 0.21127779 -0.65669673  0.60076732  2.3167443  -1.14490646
1.80969611
   1.01770049]
 [-1.07902585 -1.08694632 -0.65887969 -0.0047516   0.8734338  -
0.55257896
  -0.98260737]
 [ 1.09018027 -0.05434732  0.46080654  0.34347279  0.8734338  -
0.55257896
  -0.98260737]
 [ 0.92187979 -0.39854698  0.46080654 -0.93334996  0.8734338  -
0.55257896
  -0.98260737]
 [-0.96682553  0.0317026  -1.07876203 -0.52708818  0.8734338  -
0.55257896
  -0.98260737]
 [ 0.06167737  0.63405202  0.46080654  0.69169717 -1.14490646
1.80969611
```

```
   1.01770049]
 [ 0.13647758 -1.43114598 -0.65887969 -0.81727517  0.8734338  -
0.55257896
  -0.98260737]
 [ 1.25848074 -0.74274665 -0.09903657 -0.70120037  0.8734338  -
0.55257896
   1.01770049]
 [ 1.70728201 -0.39854698  0.60076732 -0.062789    0.8734338  -
0.55257896
  -0.98260737]
 [-0.66762469 -0.22644715 -0.93880125  0.57562238 -1.14490646 -
0.55257896
   1.01770049]
 [-0.31232369  0.46195218 -0.51891891  1.38814594 -1.14490646 -
0.55257896
   1.01770049]
 [ 0.9592799  -1.5171959   0.0409242  -2.3262475   0.8734338  -
0.55257896
  -0.98260737]
 [-0.72372485  0.20380243  0.18088498 -1.80391093 -1.14490646 -
0.55257896
  -0.98260737]
 [ 2.62358459  1.23640143  2.1403359   0.69169717  0.8734338  -
0.55257896
   1.01770049]
 [ 1.46418132 -0.14039723  0.18088498  0.16936059  0.8734338  -
0.55257896
   1.01770049]
 [ 1.50158143 -0.91484648  0.88068888 -0.062789    0.8734338  -
0.55257896
  -0.98260737]
 [ 1.78208222  0.72010193  0.18088498 -0.1208264   0.8734338  -
0.55257896
   1.01770049]
 [-0.68632474 -1.60324582  0.88068888  0.28543539 -1.14490646 -
0.55257896
  -0.98260737]
 [-0.48062416  2.35505035  0.60076732  1.03992156  0.8734338  -
0.55257896
   1.01770049]
 [-0.424524   -1.43114598 -1.91852671 -1.04942476  0.8734338  -
0.55257896
  -0.98260737]
 [ 2.06258301  1.40850127  1.86041434  1.96851992  0.8734338  -
0.55257896
   1.01770049]
 [-0.76112495  0.28985235 -1.63860515 -0.23690119 -1.14490646 -
0.55257896
   1.01770049]
 [-0.08792305  2.35505035  0.46080654  1.62029553 -1.14490646 -
0.55257896
   1.01770049]
 [-0.83592516 -1.34509607  0.0409242  -1.62979873  0.8734338  -
0.55257896
  -0.98260737]
 [ 0.97797995 -0.91484648 -0.93880125 -0.0047516   0.8734338  -
0.55257896
  -0.98260737]
 [ 0.26737795  0.11775252  0.0409242   0.92384676  0.8734338  -
0.55257896
   1.01770049]
 [ 0.19257774  0.11775252  0.46080654  1.27207115 -1.14490646
1.80969611
   1.01770049]
 [ 1.40808116  1.32245135  1.58049278  0.80777197  0.8734338  -
0.55257896
   1.01770049]
 [ 1.91298259  0.5480021   0.7407281   1.03992156  0.8734338  -
0.55257896
   1.01770049]
 [ 0.80967948 -0.14039723 -1.91852671 -1.04942476  0.8734338  -
0.55257896
  -0.98260737]
 [-0.7424249   1.40850127 -0.23899735  1.27207115 -1.14490646 -
0.55257896
   1.01770049]
 [-1.13512601 -0.31249707  1.440532   -0.35297599  0.8734338  -
0.55257896
  -0.98260737]
 [-0.66762469  1.40850127 -0.23899735  0.45954758 -1.14490646 -
0.55257896
   1.01770049]
 [ 0.13647758 -0.91484648 -0.65887969 -0.81727517  0.8734338  -
0.55257896
  -0.98260737]
 [-1.17252611 -1.86139557  0.46080654 -0.81727517 -1.14490646 -
0.55257896
  -0.98260737]]
```

What can you observe in the values related to "island_Dream", "island_Torgersen" and "sex_Male" features before and after scaling?

Before scaling, the values related to these categorical features are either 0 or 1 because they are binary (indicating the presence or absence of a category). After scaling, using either "MaxAbsScaler" or "StandardScaler" doesn't change the values for these binary features since their maximum absolute value is 1. So, the values for "island_Dream," "island_Torgersen," and "sex_Male" remain the same (0 or 1) before and after scaling. Scaling primarily affects continuous numeric features.