

# Activity\_Build a K-means model

November 23, 2023

## 1 Activity: Build a K-means model

### 1.1 Introduction

K-means clustering is very effective when segmenting data and attempting to find patterns. Because clustering is used in a broad array of industries, becoming proficient in this process will help you expand your skillset in a widely applicable way.

In this activity, you are a consultant for a scientific organization that works to support and sustain penguin colonies. You are tasked with helping other staff members learn more about penguins in order to achieve this mission.

The data for this activity is in a spreadsheet that includes datapoints across a sample size of 345 penguins, such as species, island, and sex. You will use a K-means clustering model to group this data and identify patterns that provide important insights about penguins.

**Note:** Because this lab uses a real dataset, this notebook will first require basic EDA, data cleaning, and other manipulations to prepare the data for modeling.

### 1.2 Step 1: Imports

Import statements including K-means, `silhouette_score`, and `StandardScaler`.

```
[1]: # Import standard operational packages.
import pandas as pd
import numpy as np

# Important tools for modeling and evaluation.
from sklearn.metrics import silhouette_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Import visualization packages.
import seaborn as sns
import matplotlib.pyplot as plt

### YOUR CODE HERE ###
```

Pandas is used to load the penguins dataset, which is built into the `seaborn` library. The resulting `pandas` `DataFrame` is saved in a variable named `penguins`. As shown in this cell, the dataset has been automatically loaded in for you. You do not need to download the `.csv` file, or provide more code, in order to access the dataset and proceed with this lab. Please continue with this activity by completing the following instructions.

```
[2]: # RUN THIS CELL TO IMPORT YOUR DATA.

# Save the `pandas` DataFrame in variable `penguins`.

### YOUR CODE HERE ###

penguins = pd.read_csv("penguins.csv")
```

Hint 1

Use the `load_dataset` function.

Hint 2

The function is from `seaborn` (`sns`). It should be passed in the dataset name `'penguins'` as a string.

Now, review the first 10 rows of data.

```
[3]: # Review the first 10 rows.

### YOUR CODE HERE ###

penguins.head(10)
```

```
[3]:  species      island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0  Adelie  Torgersen         39.1           18.7           181.0
1  Adelie  Torgersen         39.5           17.4           186.0
2  Adelie  Torgersen         40.3           18.0           195.0
3  Adelie  Torgersen          NaN            NaN            NaN
4  Adelie  Torgersen         36.7           19.3           193.0
5  Adelie  Torgersen         39.3           20.6           190.0
6  Adelie  Torgersen         38.9           17.8           181.0
7  Adelie  Torgersen         39.2           19.6           195.0
8  Adelie  Torgersen         34.1           18.1           193.0
9  Adelie  Torgersen         42.0           20.2           190.0

      body_mass_g      sex
0         3750.0    male
1         3800.0  female
2         3250.0  female
3            NaN      NaN
4         3450.0  female
5         3650.0    male
6         3625.0  female
```

7	4675.0	male
8	3475.0	NaN
9	4250.0	NaN

Hint 1

Use the `head()` method.

Hint 2

By default, the method only returns five rows. To change this, specify how many rows (`n =` ) you want.

### 1.3 Step 2: Data exploration

After loading the dataset, the next step is to prepare the data to be suitable for clustering. This includes:

- Exploring data
- Checking for missing values
- Encoding data
- Dropping a column
- Scaling the features using `StandardScaler`

#### 1.3.1 Explore data

To cluster penguins of multiple different species, determine how many different types of penguin species are in the dataset.

```
[4]: # Find out how many penguin types there are.
```

```
### YOUR CODE HERE ###  
penguins['species'].unique()
```

```
[4]: array(['Adelie', 'Chinstrap', 'Gentoo'], dtype=object)
```

Hint 1

Use the `unique()` method.

Hint 2

Use the `unique()` method on the column `'species'`.

```
[5]: # Find the count of each species type.
```

```
### YOUR CODE HERE ###  
penguins['species'].value_counts()
```

```
[5]: Adelie      152
      Gentoo     124
      Chinstrap  68
      Name: species, dtype: int64
```

Hint 1

Use the `value_counts()` method.

Hint 2

Use the `value_counts()` method on the column `'species'`.

**Question:** How many types of species are present in the dataset?

There are three types of penguin species in the data set.

**Question:** Why is it helpful to determine the perfect number of clusters using K-means when you already know how many penguin species the dataset contains?

It might help us to evaluate how well the model clusters/labels penguins based on their characteristics. Also while inertia and silhouette score might be better for the number of clusters different than 3 it may not be very meaningful.

### 1.3.2 Check for missing values

An assumption of K-means is that there are no missing values. Check for missing values in the rows of the data.

```
[6]: # Check for missing values.

    ### YOUR CODE HERE ###
    penguins.isna().sum()
```

```
[6]: species      0
      island      0
      bill_length_mm  2
      bill_depth_mm  2
      flipper_length_mm  2
      body_mass_g    2
      sex          11
      dtype: int64
```

Hint 1

Use the `isnull` and `sum` methods.

Now, drop the rows with missing values and save the resulting pandas DataFrame in a variable named `penguins_subset`.

```
[7]: # Drop rows with missing values.
      # Save DataFrame in variable `penguins_subset`.
```

```

### YOUR CODE HERE ###
penguins_subset = penguins.dropna(axis = 0)

penguins.info()
#penguins_subset.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   bill_length_mm        342 non-null   float64
 3   bill_depth_mm         342 non-null   float64
 4   flipper_length_mm     342 non-null   float64
 5   body_mass_g           342 non-null   float64
 6   sex                   333 non-null   object
dtypes: float64(4), object(3)
memory usage: 18.9+ KB

```

Hint 1

Use `dropna`. Note that an `axis` parameter passed in to this function should be set to 0 if you want to drop rows containing missing values or 1 if you want to drop columns containing missing values. Optionally, `reset_index` may also be used to avoid a `SettingWithCopy` warning later in the notebook.

Next, check to make sure that `penguins_subset` does not contain any missing values.

```
[8]: # Check for missing values.
```

```

### YOUR CODE HERE ###
penguins_subset.isna().sum()
#penguins_subset.info()

```

```

[8]: species                0
     island                0
     bill_length_mm        0
     bill_depth_mm         0
     flipper_length_mm     0
     body_mass_g           0
     sex                   0
     dtype: int64

```

Now, review the first 10 rows of the subset.

```
[9]: # View first 10 rows.

### YOUR CODE HERE ###
penguins_subset.head(10)
```

```
[9]:   species      island  bill_length_mm  bill_depth_mm  flipper_length_mm  \
0   Adelie  Torgersen         39.1          18.7           181.0
1   Adelie  Torgersen         39.5          17.4           186.0
2   Adelie  Torgersen         40.3          18.0           195.0
4   Adelie  Torgersen         36.7          19.3           193.0
5   Adelie  Torgersen         39.3          20.6           190.0
6   Adelie  Torgersen         38.9          17.8           181.0
7   Adelie  Torgersen         39.2          19.6           195.0
12  Adelie  Torgersen         41.1          17.6           182.0
13  Adelie  Torgersen         38.6          21.2           191.0
14  Adelie  Torgersen         34.6          21.1           198.0

      body_mass_g      sex
0          3750.0    male
1          3800.0  female
2          3250.0  female
4          3450.0  female
5          3650.0    male
6          3625.0  female
7          4675.0    male
12         3200.0  female
13         3800.0    male
14         4400.0    male
```

### 1.3.3 Encode data

Some versions of the penguins dataset have values encoded in the sex column as ‘Male’ and ‘Female’ instead of ‘MALE’ and ‘FEMALE’. The code below will make sure all values are ALL CAPS.

```
[10]: penguins_subset['sex'] = penguins_subset['sex'].str.upper()
```

K-means needs numeric columns for clustering. Convert the categorical column 'sex' into numeric. There is no need to convert the 'species' column because it isn't being used as a feature in the clustering algorithm.

```
[11]: # Convert `sex` column from categorical to numeric.

### YOUR CODE HERE ###
penguins_subset['sex'] = pd.get_dummies(penguins_subset['sex'], drop_first = 
    ↪ True)
```

Hint 1

Use the `get_dummies` function.

Hint 2

The `drop_first` parameter should be set to `True`. This removes redundant data. The `columns` parameter can **optionally** be set to `['sex']` to specify that only the 'sex' column gets this operation performed on it.

### 1.3.4 Drop a column

Drop the categorical column `island` from the dataset. While it has value, this notebook is trying to confirm if penguins of the same species exhibit different physical characteristics based on sex. This doesn't include location.

Note that the 'species' column is not numeric. Don't drop the 'species' column for now. It could potentially be used to help understand the clusters later.

```
[12]: # Drop the island column.

### YOUR CODE HERE ###
penguins_subset.drop(columns = 'island', axis = 1, inplace = True)
penguins_subset.head()
```

```
[12]:   species  bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  sex
0  Adelie             39.1             18.7             181.0        3750.0    1
1  Adelie             39.5             17.4             186.0        3800.0    0
2  Adelie             40.3             18.0             195.0        3250.0    0
4  Adelie             36.7             19.3             193.0        3450.0    0
5  Adelie             39.3             20.6             190.0        3650.0    1
```

### 1.3.5 Scale the features

Because K-means uses distance between observations as its measure of similarity, it's important to scale the data before modeling. Use a third-party tool, such as scikit-learn's `StandardScaler` function. `StandardScaler` scales each point  $x$  by subtracting the mean observed value for that feature and dividing by the standard deviation:

$$x\text{-scaled} = (x - \text{mean}(X)) /$$

This ensures that all variables have a mean of 0 and variance/standard deviation of 1.

**Note:** Because the species column isn't a feature, it doesn't need to be scaled.

First, copy all the features except the 'species' column to a DataFrame `X`.

```
[13]: # Exclude `species` variable from X

### YOUR CODE HERE ###
X = penguins_subset.copy()
X.drop(columns = 'species', axis = 1, inplace = True)
```

```
X.head()
```

```
[13]:   bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  sex
0           39.1           18.7           181.0        3750.0    1
1           39.5           17.4           186.0        3800.0    0
2           40.3           18.0           195.0        3250.0    0
4           36.7           19.3           193.0        3450.0    0
5           39.3           20.6           190.0        3650.0    1
```

Hint 1

Usedrop().

Hint 2

Select all columns except 'species'. The `axis` parameter passed in to this method should be set to 1 if you want to drop columns.

Scale the features in `X` using `StandardScaler`, and assign the scaled data to a new variable `X_scaled`.

```
[14]: #Scale the features.
      #Assign the scaled data to variable `X_scaled`.

      ### YOUR CODE HERE ###
      scaler = StandardScaler().fit(X)
      X_scaled = scaler.transform(X)
      X_scaled
```

```
[14]: array([[ -0.89604189,  0.7807321 , -1.42675157, -0.56847478,  0.99103121],
        [ -0.82278787,  0.11958397, -1.06947358, -0.50628618, -1.00904996],
        [ -0.67627982,  0.42472926, -0.42637319, -1.1903608 , -1.00904996],
        ...,
        [ 1.17338426, -0.74499437,  1.50292796,  1.91906927,  0.99103121],
        [ 0.22108196, -1.20271231,  0.78837197,  1.23499466, -1.00904996],
        [ 1.08181673, -0.54156417,  0.85982757,  1.48374906,  0.99103121]])
```

Hint 1

Instantiate `StandardScaler` to transform the data in a single step.

Hint 2

Use the `.fit_transform()` method and pass in the data as an argument.

## 1.4 Step 3: Data modeling

Now, fit K-means and evaluate inertia for different values of `k`. Because you may not know how many clusters exist in the data, start by fitting K-means and examining the inertia values for different values of `k`. To do this, write a function called `kmeans_inertia` that takes in `num_clusters` and `x_vals` (`X_scaled`) and returns a list of each `k`-value's inertia.



When using K-means inside the function, set the `random_state` to 42. This way, others can reproduce your results.

```
[30]: # Fit K-means and evaluate inertia for different values of k.
```

```
### YOUR CODE HERE ###
def kmeans_inertia(num_clusters, x_vals):

    inertia = []

    for i in num_clusters:
        kmeans = KMeans(n_clusters=i, random_state=42)
        kmeans.fit(x_vals)
        inertia.append(kmeans.inertia_)
    return inertia
```

Use the `kmeans_inertia` function to return a list of inertia for `k=2` to 10.

```
[31]: # Return a list of inertia for k=2 to 10.
```

```
### YOUR CODE HERE ###
num_clusters = list(range(2,11))

inertia = kmeans_inertia(num_clusters, X_scaled)
inertia
```

```
[31]: [885.6224143652249,
      578.8284278107235,
      386.14534424773285,
      284.5464837898288,
      217.92858573807678,
      201.39287843423264,
      186.82270634899209,
      173.47283154242746,
      164.55854201979943]
```

Hint 1

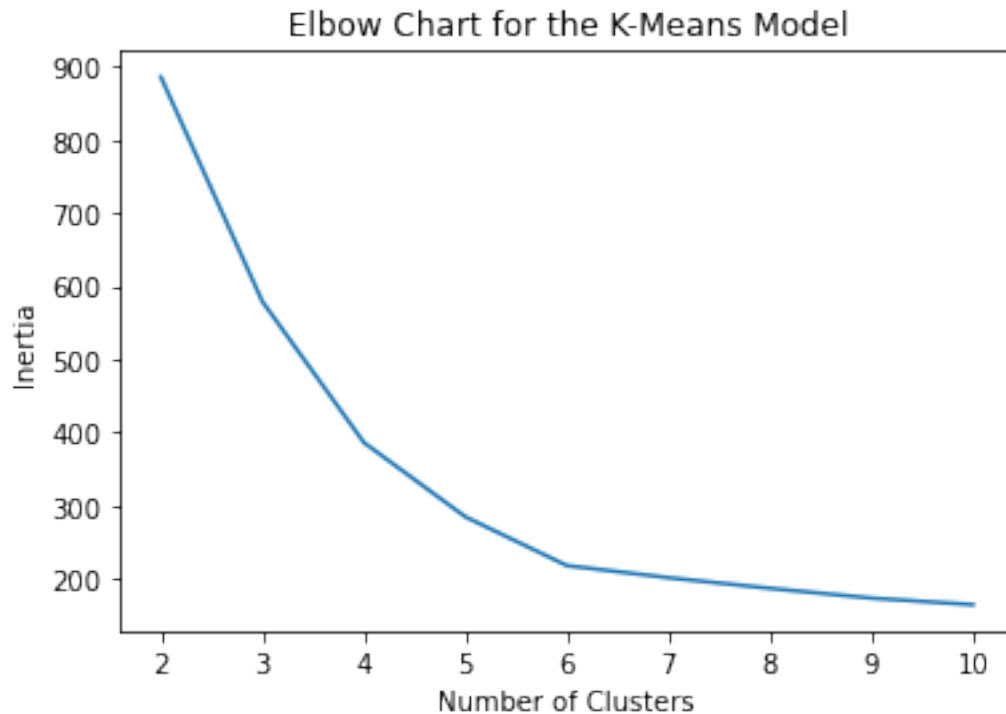
Review the material about the `kmeans_inertia` function.

Next, create a line plot that shows the relationship between `num_clusters` and `inertia`. Use either `seaborn` or `matplotlib` to visualize this relationship.

```
[66]: # Create a line plot.
```

```
### YOUR CODE HERE ###
sns.lineplot(x = num_clusters, y = inertia);
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
```

```
plt.title('Elbow Chart for the K-Means Model');
```



Hint 1

Use `sns.lineplot`.

Hint 2

Include `x=num_clusters` and `y=inertia`.

**Question:** Where is the elbow in the plot?

The elbow is when the cluster number is 5 or even 6.

## 1.5 Step 4: Results and evaluation

Now, evaluate the silhouette score using the `silhouette_score()` function. Silhouette scores are used to study the distance between clusters.

Then, compare the silhouette score of each value of `k`, from 2 through 10. To do this, write a function called `kmeans_sil` that takes in `num_clusters` and `x_vals` (`X_scaled`) and returns a list of each `k`-value's silhouette score.

```
[67]: # Evaluate silhouette score.  
      # Write a function to return a list of each k-value's score.
```

```

### YOUR CODE HERE ###
def kmeans_sil(num_clusters,x_vals):
    sil_score = []
    for i in num_clusters:
        kmeans = KMeans(n_clusters = i, random_state = 42)
        kmeans.fit(x_vals)
        sil_score.append(silhouette_score(x_vals, kmeans.labels_))
    return sil_score

silhouette = kmeans_sil(num_clusters,X_scaled)
silhouette

```

```

[67]: [0.44398088353055243,
      0.45101024097188364,
      0.5080140996630784,
      0.519998574860868,
      0.5263224884981607,
      0.47774022332151733,
      0.42680523270292947,
      0.35977478703657334,
      0.3589883410610364]

```

Hint 1

Review the `kmeans_sil` function video.

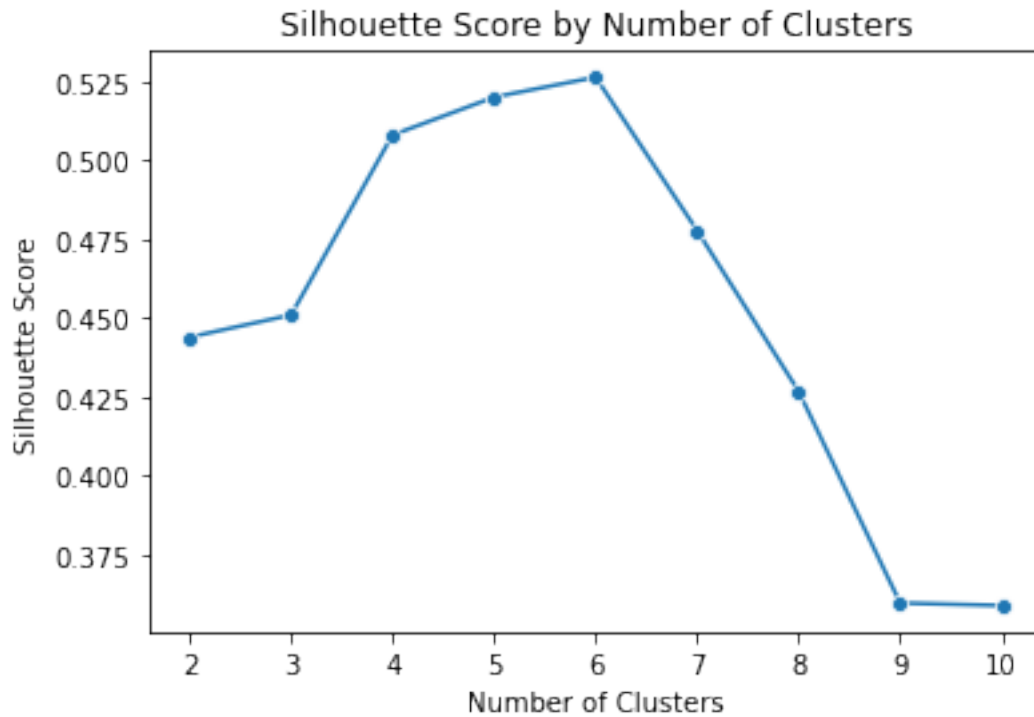
Next, create a line plot that shows the relationship between `num_clusters` and `sil_score`. Use either `seaborn` or `matplotlib` to visualize this relationship.

```

[69]: # Create a line plot.

### YOUR CODE HERE ###
plot = sns.lineplot(x = num_clusters, y = silhouette, marker = 'o')
plot.set_title('Silhouette Score by Number of Clusters')
plot.set_xlabel('Number of Clusters')
plot.set_ylabel('Silhouette Score');

```



Hint 1

Use `sns.lineplot`.

Hint 2

Include `x=num_clusters` and `y=sil_score`.

**Question:** What does the graph show?

The silhouette score is maximized when the number of clusters is set to 6.

### 1.5.1 Optimal k-value

To decide on an optimal k-value, fit a six-cluster model to the dataset.

```
[40]: # Fit a 6-cluster model.
```

```
### YOUR CODE HERE ###
```

```
kmeans6 = KMeans(n_clusters = 6, random_state = 42)
kmeans6.fit(X_scaled)
```

```
[40]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
            n_clusters=6, n_init=10, n_jobs=None, precompute_distances='auto',
            random_state=42, tol=0.0001, verbose=0)
```

Hint 1

Make an instance of the model with `num_clusters = 6` and use the `fit` function on `X_scaled`.

Print out the unique labels of the fit model.

```
[45]: # Print unique labels.

### YOUR CODE HERE ###
np.unique(kmeans6.labels_)
```

```
[45]: array([0, 1, 2, 3, 4, 5], dtype=int32)
```

Now, create a new column `cluster` that indicates cluster assignment in the DataFrame `penguins_subset`. It's important to understand the meaning of each cluster's labels, then decide whether the clustering makes sense.

**Note:** This task is done using `penguins_subset` because it is often easier to interpret unscaled data.

```
[47]: # Create a new column `cluster`.

### YOUR CODE HERE ###
penguins_subset['cluster'] = kmeans6.labels_
penguins_subset.head()
```

```
[47]:  species  bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  sex  \
0  Adelie             39.1             18.7             181.0        3750.0    1
1  Adelie             39.5             17.4             186.0        3800.0    0
2  Adelie             40.3             18.0             195.0        3250.0    0
4  Adelie             36.7             19.3             193.0        3450.0    0
5  Adelie             39.3             20.6             190.0        3650.0    1

   cluster
0        0
1        2
2        2
4        2
5        0
```

Use `groupby` to verify if any 'cluster' can be differentiated by 'species'.

```
[59]: # Verify if any `cluster` can be differentiated by `species`.

### YOUR CODE HERE ###
penguins_subset.groupby(by = ['cluster', 'species']).size()
```

```
[59]: cluster  species
0        Adelie    71
1        Gentoo   58
```

```

2      Adelie      73
      Chinstrap    5
3      Gentoo     61
4      Adelie      2
      Chinstrap    34
5      Chinstrap   29
dtype: int64

```

Hint 1

Use `groupby` (`by=['cluster', 'species']`).

Hint 2

Use an aggregation function such as `size`.

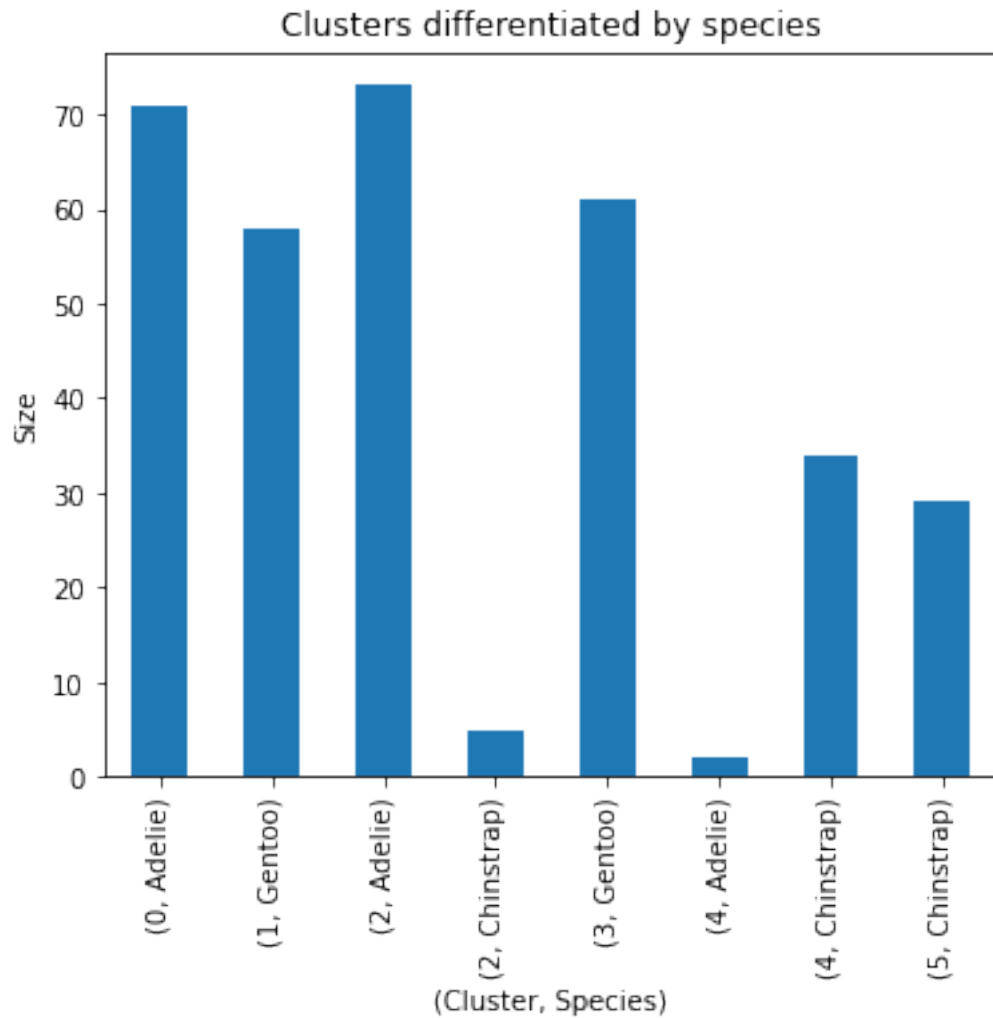
Next, interpret the `groupby` outputs. Although the results of the `groupby` show that each `'cluster'` can be differentiated by `'species'`, it is useful to visualize these results. The graph shows that each `'cluster'` can be differentiated by `'species'`.

**Note:** The code for the graph below is outside the scope of this lab.

```

[58]: penguins_subset.groupby(by=['cluster', 'species']).size().plot.
      ↪ bar(title='Clusters differentiated by species',
      ↪                                     figsize=(6, 5),
      ↪                                     )
      ↪ ylabel='Size',
      ↪                                     )
      ↪ xlabel='(Cluster, Species)');

```



Use `groupby` to verify if each 'cluster' can be differentiated by 'species' AND 'sex\_MALE'.

```
[64]: # Verify if each `cluster` can be differentiated by `species` AND `sex_MALE`.
```

```
### YOUR CODE HERE ###
```

```
penguins_subset.groupby(by = ['cluster', 'species', 'sex']).size()
```

```
[64]: cluster  species  sex
0         Adelie    1    71
1         Gentoo    0    58
2         Adelie    0    73
          Chinstrap 0     5
3         Gentoo    1    61
4         Adelie    1     2
          Chinstrap 1    34
```

```
5          Chinstrap  0          29
dtype: int64
```

Hint 1

Use `groupby(by=['cluster','species', 'sex_MALE'])`.

Hint 2

Use an aggregation function such as `size`.

**Question:** Are the clusters differentiated by 'species' and 'sex\_MALE'?

Yes. The male and female adelic penguins are mostly assigned to cluster 0 and cluster 2, respectively. The exceptions were two adelic males that were put into a different cluster, cluster 4. Cluster 4 was mostly male chinstrap penguins. The female chinstrap penguins were mostly assigned to cluster 5. However, 5 of them were erroneously assigned to the adelic female penguins cluster, cluster 2. This suggests that adelic and chinstrap penguins might be similar in certain characteristics.

Gentoo penguins had their own exclusive clusters. The females were assigned to cluster 1 and the males to cluster 3.

Finally, interpret the groupby outputs and visualize these results. The graph shows that each 'cluster' can be differentiated by 'species' and 'sex\_MALE'. Furthermore, each cluster is mostly comprised of one sex and one species.

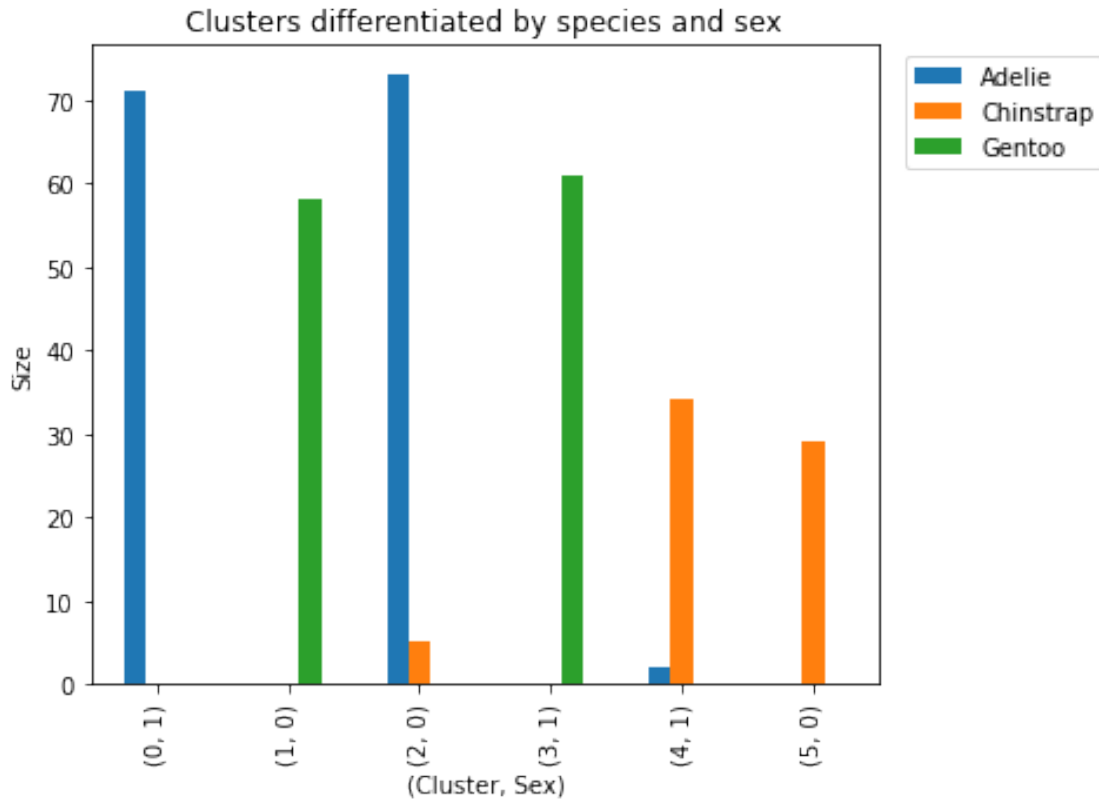
**Note:** The code for the graph below is outside the scope of this lab.

```
[65]: penguins_subset.groupby(by=['cluster','species','sex']).size().unstack(level =
      ↳ 'species', fill_value=0).plot.bar(title='Clusters differentiated by species
      ↳ and sex',

      ↳                                     figsize=(6, 5),
      ↳                                     ylabel='Size',
      ↳                                     xlabel='(Cluster, Sex)')
      plt.legend(bbox_to_anchor=(1.3, 1.0))
```

```
[65]: <matplotlib.legend.Legend at 0x7fabf33dc590>
```





## 1.6 Considerations

**What are some key takeaways that you learned during this lab? Consider the process you used, key tools, and the results of your investigation.**

- Kmeans is a useful tool to separate unlabeled data into meaningful clusters.
- Inertia and silhouette scores are useful metrics to identify the number of clusters.
- Even if we know the number of types/categories/clusters apriori, using the elbow method and silhouette scores can help us to discover more useful categorization.

**What summary would you provide to stakeholders?**

Penguins can be meaningfully clustered by species and sex. So for the current dataset, it is useful to have 6 clusters (3 species times 2 sex categories). When the cluster number is set to 6, the kmeans model seems to perform the best. This is confirmed by elbow method. Moreover, the silhouette score is maximized at cluster number being 6.

### 1.6.1 References

Gorman, Kristen B., et al. "Ecological Sexual Dimorphism and Environmental Variability within a Community of Antarctic Penguins (Genus *Pygoscelis*)." PLOS ONE, vol. 9, no. 3, Mar. 2014,

[p. e90081. PLoS Journals](#)

[Sklearn Preprocessing StandardScaler](#) [scikit-learn](#)

**Congratulations!** You’ve completed this lab. However, you may not notice a green check mark next to this item on Coursera’s platform. Please continue your progress regardless of the check mark. Just click on the “save” icon at the top of this notebook to ensure your work has been logged