In [ ]:
```
pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-pa
ckages (3.4.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/
dist-packages (from pyspark) (0.10.9.7)
```

Original Image

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img_path = '/content/Images/Penguins.jpg'

img = mpimg.imread(img_path)
plt.imshow(img)
plt.axis('off')
plt.show()
```

In [ ]:
```
from pyspark.sql import SparkSession
import numpy as np
import cv2
import os

spark_session = SparkSession.builder.appName("UnsupervisedImgSegment").getO

def img_seg_with_kmeans(img_file, num_clusters):
  # Reading the image
  img_data = cv2.imread(img_file)
  original_dimensions = img_data.shape
  img_array = np.asarray(img_data).reshape(-1, 3)

  # Convert the image array into a dataframe
  df = spark_session.createDataFrame([tuple(x) for x in img_array.tolist()],

  # Start with k random centroids
  centroids_list = df.rdd.takeSample(False, num_clusters)
```

```
  centroids_list = spark_session.sparkContext.broadcast(centroids_list)

  def calculate_min_distance(data_row):
    data_row_array = np.array(list(data_row))
    computed_distances = [np.linalg.norm(data_row_array - np.array(c)) for c
    return (computed_distances.index(min(computed_distances)), list(data_row

  for iteration in range(10):  # run 10 iterations
      # Assign each row in the DataFrame to the nearest centroid

      assigned_clusters = df.rdd.map(calculate_min_distance).toDF(["cluster_

      # Recalculate the centroids
      new_centroids_list = assigned_clusters.rdd.map(lambda x: (x[0], (x[1],
                                  .reduceByKey(lambda x, y: ([x[0][i] + y[0]
                                  .mapValues(lambda x: [x[0][i] / x[1] for i
                                  .collect()
      new_centroids_list = sorted(new_centroids_list, key=lambda x: x[0])
      new_centroids_list = [x[1] for x in new_centroids_list]
      centroids_list = spark_session.sparkContext.broadcast(new_centroids_li

  # Replace the color of each coordinate with the color of its centroid
  color_map = {i: color for i, color in enumerate(new_centroids_list)}
  color_df = assigned_clusters.rdd.map(lambda x: (x[0], color_map[x[0]])).to
  modified_img_array = np.array(color_df.select('color').rdd.flatMap(list).c

  # Reconstruct the image
  modified_img_array = modified_img_array.astype(np.uint8).reshape(original_

  # Save the new image
  new_img_file = img_file[:-4] + f"_k={num_clusters}_seg.jpg"  # remove '.jp
  cv2.imwrite(new_img_file, modified_img_array)
  return new_img_file

def calc_compression_ratio(initial_path, final_path):
  original_file_size = os.path.getsize(initial_path)
  compressed_file_size = os.path.getsize(final_path)

  compression_ratio = original_file_size / compressed_file_size
  return compression_ratio

# Specify the directory with the images
img_dir = "/content/Images"

# Define the range of k values
cluster_numbers = range(2, 31, 2)  # Adjust as per requirements

# Output images
output_images = {}

# compression_values
compression_values = {}

# Apply the function for all images and for each k value
for img_file in os.listdir(img_dir):
  if img_file.endswith(".jpg"):  # or ".png", ".jpeg", etc. as needed
      full_img_path = os.path.join(img_dir, img_file)
      for num_clusters in cluster_numbers:
        resultant_img_path = img_seg_with_kmeans(full_img_path, num_clusters
        output_images[num_clusters] = resultant_img_path
        compression_val = calc_compression_ratio(full_img_path, resultant_ir
        compression_values[num_clusters] = compression_val
        print(f"Compression ratio for {img_file} with k = {num_clusters} is
```

```
Compression ratio for Penguins.jpg with k = 2 is 5.138769604798964
Compression ratio for Penguins.jpg with k = 4 is 3.8731401995737644
Compression ratio for Penguins.jpg with k = 6 is 3.3783660528144543
Compression ratio for Penguins.jpg with k = 8 is 2.821135439599299
Compression ratio for Penguins.jpg with k = 10 is 2.6561955757108024
Compression ratio for Penguins.jpg with k = 12 is 2.8148262073208246
Compression ratio for Penguins.jpg with k = 14 is 2.62002283743318
Compression ratio for Penguins.jpg with k = 16 is 2.625231190852267
Compression ratio for Penguins.jpg with k = 18 is 2.578943005868506
Compression ratio for Penguins.jpg with k = 20 is 2.562317379688833
Compression ratio for Penguins.jpg with k = 22 is 2.570072459697804
Compression ratio for Penguins.jpg with k = 24 is 2.5084411794109394
Compression ratio for Penguins.jpg with k = 26 is 2.489382674957835
Compression ratio for Penguins.jpg with k = 28 is 2.557271358402977
Compression ratio for Penguins.jpg with k = 30 is 2.4697251609789554
```

In [ ]:

```python
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg


# Iterate over clusters
for cluster, img_path in output_images.items():
  print(f"Cluster {cluster}:")
  plt.figure(figsize=(10,10))

  # Check if the image file exists
  if os.path.isfile(img_path):
      img = mpimg.imread(img_path)  # read the image from the path
      plt.imshow(img)  # display the image
      plt.axis('off')  # turn off the axis
  else:
    print(f"File does not exist: {img_path}")

  plt.show()
```

```
Cluster 2:
```

Cluster 4:

Cluster 6:

Cluster 8:

Cluster 10:

Cluster 12:

Cluster 14:

Cluster 16:

Cluster 18:

Cluster 20:

Cluster 22:

Cluster 24:

Cluster 26:

Cluster 28:

Cluster 30:

In [ ]: