In [1]:
```
pip install --upgrade azure-cognitiveservices-vision-computervision
```

```
Collecting azure-cognitiveservices-vision-computervision
  Downloading azure_cognitiveservices_vision_computervision-0.7.0-py2.py3-none-any.whl (35 kB)
Collecting msrest>=0.5.0
  Downloading msrest-0.6.19-py2.py3-none-any.whl (84 kB)
Collecting azure-common~=1.1
  Downloading azure_common-1.1.26-py2.py3-none-any.whl (12 kB)
Requirement already satisfied, skipping upgrade: requests~=2.16 in d:\python\lib\site-packages (from msrest>=
0.5.0->azure-cognitiveservices-vision-computervision) (2.24.0)
Collecting isodate>=0.6.0
  Downloading isodate-0.6.0-py2.py3-none-any.whl (45 kB)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in d:\python\lib\site-packages (from msres
t>=0.5.0->azure-cognitiveservices-vision-computervision) (2020.6.20)
Requirement already satisfied, skipping upgrade: requests-oauthlib>=0.5.0 in d:\python\lib\site-packages (from
msrest>=0.5.0->azure-cognitiveservices-vision-computervision) (1.3.0)
Requirement already satisfied, skipping upgrade: idna<3,>=2.5 in d:\python\lib\site-packages (from requests~=
2.16->msrest>=0.5.0->azure-cognitiveservices-vision-computervision) (2.10)
Requirement already satisfied, skipping upgrade: chardet<4,>=3.0.2 in d:\python\lib\site-packages (from reques
ts~=2.16->msrest>=0.5.0->azure-cognitiveservices-vision-computervision) (3.0.4)
Requirement already satisfied, skipping upgrade: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in d:\python\lib\site
-packages (from requests~=2.16->msrest>=0.5.0->azure-cognitiveservices-vision-computervision) (1.25.9)
Requirement already satisfied, skipping upgrade: six in d:\python\lib\site-packages (from isodate>=0.6.0->msre
st>=0.5.0->azure-cognitiveservices-vision-computervision) (1.15.0)
Requirement already satisfied, skipping upgrade: oauthlib>=3.0.0 in d:\python\lib\site-packages (from requests
-oauthlib>=0.5.0->msrest>=0.5.0->azure-cognitiveservices-vision-computervision) (3.1.0)
Installing collected packages: isodate, msrest, azure-common, azure-cognitiveservices-vision-computervision
Successfully installed azure-cognitiveservices-vision-computervision-0.7.0 azure-common-1.1.26 isodate-0.6.0 m
srest-0.6.19
Note: you may need to restart the kernel to use updated packages.
```

In [1]:
```
from azure.cognitiveservices.vision.computervision import ComputerVisionClient
from azure.cognitiveservices.vision.computervision.models import OperationStatusCodes
from azure.cognitiveservices.vision.computervision.models import VisualFeatureTypes
from msrest.authentication import CognitiveServicesCredentials

from array import array
import os
from PIL import Image
import sys
import time
```

In [2]:
```
subscription_key = "16d670a0ebd7461db042805430872e6f"
endpoint = "https://sjreddy.cognitiveservices.azure.com/"
```

In [3]:
```
computervision_client = ComputerVisionClient(endpoint, CognitiveServicesCredentials(subscription_key))
```

In [41]:
```
remote_image_url = "https://i.guim.co.uk/img/media/933249d24608932fc897fcaa5e8c8bb2bdc9e977/124_0_1800_1080/mast
```

In [ ]:
```
# Describes content of a image
```

In [11]:
```
'''
Describe an Image - remote
This example describes the contents of an image with the confidence score.
'''
print("===== Describe an image - remote =====")
# Call API
description_results = computervision_client.describe_image(remote_image_url )

# Get the captions (descriptions) from the response, with confidence level
print("Description of remote image: ")
if (len(description_results.captions) == 0):
    print("No description detected.")
else:
    for caption in description_results.captions:
        print("'{}' with confidence {:.2f}%".format(caption.text, caption.confidence * 100))
```

```
===== Describe an image - remote =====
Description of remote image:
'Chris Hemsworth wearing a garment' with confidence 50.81%
```

In [ ]:
```
# Extracts general categories of a image
```

In [12]:
```
'''
Categorize an Image - remote
This example extracts (general) categories from a remote image with a confidence score.
'''
print("===== Categorize an image - remote =====")
# Select the visual feature(s) you want.
remote_image_features = ["categories"]
# Call API with URL and features
categorize_results_remote = computervision_client.analyze_image(remote_image_url , remote_image_features)

# Print results with confidence score
print("Categories from remote image: ")
if (len(categorize_results_remote.categories) == 0):
    print("No categories detected.")
else:
    for category in categorize_results_remote.categories:
        print("'{}' with confidence {:.2f}%".format(category.name, category.score * 100))
```

```
===== Categorize an image - remote =====
Categories from remote image:
'people_' with confidence 62.50%
```

In [ ]:
```
# Returns a key work for each thing in the image
```

In [15]:
```
'''
Tag an Image - remote
This example returns a tag (key word) for each thing in the image.
'''
print("===== Tag an image - remote =====")
# Call API with remote image
tags_result_remote = computervision_client.tag_image(remote_image_url )

# Print results with confidence score
print("Tags in the remote image: ")
if (len(tags_result_remote.tags) == 0):
    print("No tags detected.")
else:
    for tag in tags_result_remote.tags:
        print("'{}' with confidence {:.2f}%".format(tag.name, tag.confidence * 100))
```

```
===== Tag an image - remote =====
Tags in the remote image:
'outdoor' with confidence 97.14%
'person' with confidence 95.16%
'man' with confidence 90.82%
'lightning' with confidence 90.29%
```

In [ ]:
```
# Detects different kinds of objects with bounding box
```

In [24]:
```
'''
Detect Objects - remote
This example detects different kinds of objects with bounding boxes in a remote image.
'''
print("===== Detect Objects - remote =====")
# Get URL image with different objects
remote_image_url_objects = "https://i.guim.co.uk/img/media/933249d24608932fc897fcaa5e8c8bb2bdc9e977/124_0_1800_1
# Call API with URL
detect_objects_results_remote = computervision_client.detect_objects(remote_image_url_objects)

# Print detected objects results with bounding boxes
print("Detecting objects in remote image:")
if len(detect_objects_results_remote.objects) == 0:
    print("No objects detected.")
else:
    for object in detect_objects_results_remote.objects:
        print("object at location {}, {}, {}, {}".format( \
        object.rectangle.x, object.rectangle.x + object.rectangle.w, \
        object.rectangle.y, object.rectangle.y + object.rectangle.h))
```

```
===== Detect Objects - remote =====
Detecting objects in remote image:
object at location 32, 1035, 185, 1063
```

In [ ]:
```
# Detects different brands
```

In [28]:
```
'''
Detect Brands - remote
This example detects common brands like logos and puts a bounding box around them.
'''
print("===== Detect Brands - remote =====")
# Get a URL with a brand logo
remote_image_url = "https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/images/gray-shirt-
# Select the visual feature(s) you want
remote_image_features = ["brands"]
# Call API with URL and features
detect_brands_results_remote = computervision_client.analyze_image(remote_image_url, remote_image_features)

print("Detecting brands in remote image: ")
if len(detect_brands_results_remote.brands) == 0:
    print("No brands detected.")
else:
    for brand in detect_brands_results_remote.brands:
        print("'{}' brand detected with confidence {:.1f}% at location {}, {}, {}, {}".format( \
        brand.name, brand.confidence * 100, brand.rectangle.x, brand.rectangle.x + brand.rectangle.w, \
        brand.rectangle.y, brand.rectangle.y + brand.rectangle.h))
```

```
===== Detect Brands - remote =====
Detecting brands in remote image:
'Microsoft' brand detected with confidence 62.5% at location 58, 113, 106, 152
'Microsoft' brand detected with confidence 69.8% at location 58, 260, 86, 149
```

In [ ]:
```
# Detect faces
```

In [29]:
```
'''
Detect Faces - remote
This example detects faces in a remote image, gets their gender and age,
and marks them with a bounding box.
'''
print("===== Detect Faces - remote =====")
# Get an image with faces
remote_image_url_faces = "https://raw.githubusercontent.com/Azure-Samples/cognitive-services-sample-data-files/m
# Select the visual feature(s) you want.
remote_image_features = ["faces"]
# Call the API with remote URL and features
detect_faces_results_remote = computervision_client.analyze_image(remote_image_url_faces, remote_image_features)

# Print the results with gender, age, and bounding box
print("Faces in the remote image: ")
if (len(detect_faces_results_remote.faces) == 0):
    print("No faces detected.")
else:
    for face in detect_faces_results_remote.faces:
        print("'{}' of age {} at location {}, {}, {}, {}".format(face.gender, face.age, \
        face.face_rectangle.left, face.face_rectangle.top, \
        face.face_rectangle.left + face.face_rectangle.width, \
        face.face_rectangle.top + face.face_rectangle.height))
```

```
===== Detect Faces - remote =====
Faces in the remote image:
'Male' of age 39 at location 118, 159, 212, 253
'Male' of age 54 at location 492, 111, 582, 201
'Female' of age 55 at location 18, 153, 102, 237
'Female' of age 33 at location 386, 166, 467, 247
'Female' of age 18 at location 235, 158, 311, 234
'Female' of age 8 at location 323, 163, 391, 231
```

In [ ]:
```
# To detect Adult content
```

In [30]:
```
'''
Detect Adult or Racy Content - remote
This example detects adult or racy content in a remote image, then prints the adult/racy score.
The score is ranged 0.0 - 1.0 with smaller numbers indicating negative results.
'''
print("===== Detect Adult or Racy Content - remote =====")
# Select the visual feature(s) you want
remote_image_features = ["adult"]
# Call API with URL and features
detect_adult_results_remote = computervision_client.analyze_image(remote_image_url, remote_image_features)

# Print results with adult/racy score
print("Analyzing remote image for adult or racy content ... ")
print("Is adult content: {} with confidence {:.2f}".format(detect_adult_results_remote.adult.is_adult_content, d
print("Has racy content: {} with confidence {:.2f}".format(detect_adult_results_remote.adult.is_racy_content, de
```

```
===== Detect Adult or Racy Content - remote =====
Analyzing remote image for adult or racy content ...
Is adult content: False with confidence 0.52
Has racy content: False with confidence 1.35
```

In [ ]:
```
# TO detect color
```

In [31]:
```
'''
Detect Color - remote
This example detects the different aspects of its color scheme in a remote image.
'''
print("===== Detect Color - remote =====")
# Select the feature(s) you want
remote_image_features = ["color"]
# Call API with URL and features
detect_color_results_remote = computervision_client.analyze_image(remote_image_url, remote_image_features)

# Print results of color scheme
print("Getting color scheme of the remote image: ")
print("Is black and white: {}".format(detect_color_results_remote.color.is_bw_img))
print("Accent color: {}".format(detect_color_results_remote.color.accent_color))
print("Dominant background color: {}".format(detect_color_results_remote.color.dominant_color_background))
print("Dominant foreground color: {}".format(detect_color_results_remote.color.dominant_color_foreground))
print("Dominant colors: {}".format(detect_color_results_remote.color.dominant_colors))
```

```
===== Detect Color - remote =====
Getting color scheme of the remote image:
Is black and white: False
Accent color: AD5D1E
Dominant background color: Grey
Dominant foreground color: Grey
Dominant colors: ['Grey']
```

In [39]:
```
'''
Detect Domain-specific Content - remote
This example detects celebrites and landmarks in remote images.
'''
print("===== Detect Domain-specific Content - remote =====")
# URL of one or more celebrities
#remote_image_url_celebs = "https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.polygon.com%2F2019%2F12%2F3%2F
remote_image_url_celebs= "https://i.guim.co.uk/img/media/933249d24608932fc897fcaa5e8c8bb2bdc9e977/124_0_1800_108
# Call API with content type (celebrities) and URL
detect_domain_results_celebs_remote = computervision_client.analyze_image_by_domain("celebrities", remote_image_

# Print detection results with name
print("Celebrities in the remote image:")
if len(detect_domain_results_celebs_remote.result["celebrities"]) == 0:
    print("No celebrities detected.")
else:
    for celeb in detect_domain_results_celebs_remote.result["celebrities"]:
        print(celeb["name"])
```

```
===== Detect Domain-specific Content - remote =====
Celebrities in the remote image:
Chris Hemsworth
```

In [ ]:
```
# To detect land marks
```

In [42]:
```
# Call API with content type (landmarks) and URL
detect_domain_results_landmarks = computervision_client.analyze_image_by_domain("landmarks", remote_image_url)
print()

print("Landmarks in the remote image:")
if len(detect_domain_results_landmarks.result["landmarks"]) == 0:
    print("No landmarks detected.")
else:
    for landmark in detect_domain_results_landmarks.result["landmarks"]:
        print(landmark["name"])
```

```

Landmarks in the remote image:
No landmarks detected.
```

In [ ]:
```
# To read hand written text
```

In [54]:
```
'''
Batch Read File, recognize handwritten text - remote
This example will extract handwritten text in an image, then print results, line by line.
This API call can also recognize handwriting (not shown).
'''
print("===== Batch Read File - remote =====")
# Get an image with handwritten text
remote_image_handw_text_url = "https://raw.githubusercontent.com/MicrosoftDocs/azure-docs/master/articles/cognit

# Call API with URL and raw response (allows you to get the operation location)
recognize_handw_results = computervision_client.read(remote_image_handw_text_url,  raw=True)
```

```
===== Batch Read File - remote =====
```

In [55]:
```
# Get the operation location (URL with an ID at the end) from the response
operation_location_remote = recognize_handw_results.headers["Operation-Location"]
# Grab the ID from the URL
operation_id = operation_location_remote.split("/")[-1]

# Call the "GET" API and wait for it to retrieve the results
while True:
    get_handw_text_results = computervision_client.get_read_result(operation_id)
    if get_handw_text_results.status not in ['notStarted', 'running']:
        break
    time.sleep(1)

# Print the detected text, line by line
if get_handw_text_results.status == OperationStatusCodes.succeeded:
    for text_result in get_handw_text_results.analyze_result.read_results:
        for line in text_result.lines:
            print(line.text)
            print(line.bounding_box)
print()
```

```
The quick brown fox jumps
[38.0, 650.0, 2572.0, 699.0, 2570.0, 854.0, 37.0, 815.0]
over
[184.0, 1053.0, 508.0, 1044.0, 510.0, 1123.0, 184.0, 1128.0]
the lazy dog!
[639.0, 1011.0, 1976.0, 1026.0, 1974.0, 1158.0, 637.0, 1141.0]
```

In [ ]: