* Generate a boto3 client for interacting with s3.
* Specify 'us-east-1' for the region\_name.
* Use AWS\_KEY\_ID and AWS\_SECRET to set up the credentials.
* Print the buckets.

# Generate the boto3 client for interacting with S3

s3 = boto3.client('s3', region\_name='us-east-1',

# Set up AWS credentials

aws\_access\_key\_id=AWS\_KEY\_ID,

aws\_secret\_access\_key=AWS\_SECRET)

# List the buckets

buckets = s3.list\_buckets()

# Print the buckets

print(buckets)

Help Sam initialize a boto3 client for S3, and another client for SNS.

She will use the S3 client to list the buckets in S3. She will use the SNS client to list topics she can publish to

# Generate the boto3 client for interacting with S3 and SNS

s3 = boto3.client('s3', region\_name='us-east-1',

aws\_access\_key\_id=AWS\_KEY\_ID,

aws\_secret\_access\_key=AWS\_SECRET)

sns = boto3.client('sns', region\_name='us-east-1',

aws\_access\_key\_id=AWS\_KEY\_ID,

aws\_secret\_access\_key=AWS\_SECRET)

# List S3 buckets and SNS topics

buckets = s3.list\_buckets()

topics = sns.list\_topics()

# Print out the list of SNS topics

print(topics)

Building a system that would take this repetitive work off

Such a system will:

* Get images taken by city trash trucks and store them;
* If the image contains a cat, the system will send an SMS to the Animal Services manager.

What AWS services would Sam need to implement this system?

* IAM
* S3
* SNS
* Rekognition
* Create a boto3 client to S3.
* Create 'gim-staging', 'gim-processed' and 'gim-test' buckets.
* Print the response from creating the 'gim-staging' bucket.

import boto3

# Create boto3 client to S3

s3 = boto3.client('s3', region\_name='us-east-1',

aws\_access\_key\_id=AWS\_KEY\_ID,

aws\_secret\_access\_key=AWS\_SECRET)

# Create the buckets

response\_staging = s3.create\_bucket(Bucket='gim-staging')

response\_processed = s3.create\_bucket(Bucket='gim-processed')

response\_test = s3.create\_bucket(Bucket='gim-test')

# Print out the response

print(response\_staging)

* Get the buckets from S3.
* Iterate over the bucket key from response to access the list of buckets.
* Print the name of each bucket.

# Get the list\_buckets response

response = s3.list\_buckets()

# Iterate over Buckets from .list\_buckets() response

for bucket in response['Buckets']:

# Print the Name for each bucket

print(bucket['Name'])

* Delete the 'gim-test' bucket.
* Get the list of buckets from S3.
* Print each 'Buckets' 'Name'.

# Delete the gim-test bucket

s3.delete\_bucket(Bucket='gim-test')

# Get the list\_buckets response

response = s3.list\_buckets()

# Print each Buckets Name

for bucket in response['Buckets']:

print(bucket['Name'])

* Get the buckets from S3.
* Delete the buckets that contain 'gim' and create the 'gid-staging' and 'gid-processed' buckets.
* Print the new bucket names.

# Get the list\_buckets response

response = s3.list\_buckets()

# Delete all the buckets with 'gim', create replacements.

for bucket in response['Buckets']:

if 'gim' in bucket['Name']:

s3.delete\_bucket(Bucket=bucket['Name'])

s3.create\_bucket(Bucket='gid-staging')

s3.create\_bucket(Bucket='gid-processed')

# Print bucket listing after deletion

response = s3.list\_buckets()

for bucket in response['Buckets']:

print(bucket['Name'])

* Upload 'final\_report.csv' to the 'gid-staging' bucket with the key '2019/final\_report\_01\_01.csv'.
* Get the object metadata and store it in response.
* Print the object size in bytes.

# Upload final\_report.csv to gid-staging

s3.upload\_file(Bucket='gid-staging',

# Set filename and key

Filename='final\_report.csv',

Key='2019/final\_report\_01\_01.csv')

# Get object metadata and print it

response = s3.head\_object(Bucket='gid-staging',

Key='2019/final\_report\_01\_01.csv')

# Print the size of the uploaded object

print(response['ContentLength'])

* List only objects that start with '2018/final\_' in 'gid-staging' bucket.
* Iterate over the objects, deleting each one.
* Print the keys of remaining objects in the bucket.

# List only objects that start with '2018/final\_'

response = s3.list\_objects(Bucket='gid-staging',

Prefix='2018/final\_')

# Iterate over the objects

if 'Contents' in response:

for obj in response['Contents']:

# Delete the object

s3.delete\_object(Bucket='gid-staging', Key=obj['Key'])

# Print the keys of remaining objects in the bucket

response = s3.list\_objects(Bucket='gid-staging')

for obj in response['Contents']:

print(obj['Key'])

Sam has initialized the boto3 S3 client and assigned it to the s3 variable.

In this exercise, you will help her increase government transparency by uploading public reports to the gid-staging bucket.

# Upload the final\_report.csv to gid-staging bucket

s3.upload\_file(

# Complete the filename

Filename='./final\_report.csv',

# Set the key and bucket

Key='2019/final\_report\_2019\_02\_20.csv',

Bucket='gid-staging',

# During upload, set ACL to public-read

ExtraArgs = {

'ACL': 'public-read'})

open up the data by setting the ACL of every object in the gid-staging bucket to public-read, opening up the objects to the world!

# List only objects that start with '2019/final\_'

response = s3.list\_objects(

Bucket='gid-staging', Prefix='2019/final\_')

# Iterate over the objects

for obj in response['Contents']:

# Give each object ACL of public-read

s3.put\_object\_acl(Bucket='gid-staging',

Key=obj['Key'],

ACL='public-read')

# Print the Public Object URL for each object

print("https://{}.s3.amazonaws.com/{}".format( 'gid-staging', obj['Key']))

Generate a presigned URL valid for 1 hour to 'final\_report.csv' in the 'gid-staging' bucket. Then, print it out for the City Council!

# Generate presigned\_url for the uploaded object

share\_url = s3.generate\_presigned\_url(

# Specify allowable operations

ClientMethod='get\_object',

# Set the expiration time

ExpiresIn=3600,

# Set bucket and shareable object's name

Params={'Bucket': 'gid-staging','Key': 'final\_report.csv'}

)

# Print out the presigned URL

print(share\_url)

Read these private files into pandas and concatenating them into one DataFrame!

She has already initialized the boto3 S3 client and assigned it to the s3 variable. She has listed all the objects in gid-requests in the response variable.

df\_list = [ ]

for file in response['Contents']:

# For each file in response load the object from S3

obj = s3.get\_object(Bucket='gid-requests', Key=file['Key'])

# Load the object's StreamingBody with pandas

obj\_df = pd.read\_csv(obj['Body'])

# Append the resulting DataFrame to list

df\_list.append(obj\_df)

# Concat all the DataFrames with pandas

df = pd.concat(df\_list)

# Preview the resulting DataFrame

df.head()

* Generate an HTML table with no border and only the 'service\_name' and 'link' columns.
* Generate an HTML table with borders and all columns.
* Make sure to set all URLs to be clickable.

# Generate an HTML table with no border and selected columns

services\_df.to\_html('./services\_no\_border.html',

# Keep specific columns only

columns=['service\_name', 'link'],

# Set border

border=0)

# Generate an html table with border and all columns.

services\_df.to\_html('./services\_border\_all\_columns.html',

border=1)

website, providing an interactive dashboard to members of Streets Operations.

* Upload the 'lines.html' file to 'datacamp-public' bucket.
* Specify the proper content type for the uploaded file.
* Specify that the file should be public.
* Print the Public Object URL for the new file.

# Upload the lines.html file to S3

s3.upload\_file(Filename='lines.html',

# Set the bucket name

Bucket='datacamp-public', Key='index.html',

# Configure uploaded file

ExtraArgs = {

# Set proper content type

'ContentType':'text/html',

# Set proper ACL

'ACL': 'public-read'})

# Print the S3 Public Object URL for the new file.

print("http://{}.s3.amazonaws.com/{}".format('datacamp-public', 'index.html'))

aggregate the requests from February by downloading files from the gid-requests bucket and concatenating them into one DataFrame!

* Load each object from s3.
* Read it into pandas and append it to df\_list.
* Concatenate all DataFrames in df\_list.
* Preview the DataFrame.

 publish this month's request statistics.

Write agg\_df to CSV and HTML files, and upload them to S3 as public files.

* Write CSV and HTML versions of agg\_df and name them 'feb\_final\_report.csv' and 'feb\_final\_report.html' respectively.
* Upload both versions of agg\_df to the gid-reports bucket and set them to public read.

generate a new directory listing with the February's uploaded reports and store it in a DataFrame.

update the directory listing, letting the public access reports for February as well as January!

* Write objects\_df to an HTML file 'report\_listing.html' with clickable links.
* The HTML file should only contain 'Link', 'LastModified', and 'Size' columns.
* Overwrite the 'index.html' on S3 by uploading the new version of the file.

df\_list = []

# Load each object from s3

for file in request\_files:

s3\_day\_reqs = s3.get\_object(Bucket='gid-requests',

Key=file['Key'])

# Read the DataFrame into pandas, append it to the list

day\_reqs = pd.read\_csv(s3\_day\_reqs['Body'])

df\_list.append(day\_reqs)

# Concatenate all the DataFrames in the list

all\_reqs = pd.concat(df\_list)

# Preview the DataFrame

all\_reqs.head()

# Write agg\_df to a CSV and HTML file with no border

agg\_df.to\_csv('./feb\_final\_report.csv')

agg\_df.to\_html('./feb\_final\_report.html', border=0)

# Upload the generated CSV to the gid-reports bucket

s3.upload\_file(Filename='./feb\_final\_report.csv',

Key='2019/feb/final\_report.html', Bucket='gid-reports',

ExtraArgs = {'ACL': 'public-read'})

# Upload the generated HTML to the gid-reports bucket

s3.upload\_file(Filename='./feb\_final\_report.html',

Key='2019/feb/final\_report.html', Bucket='gid-reports',

ExtraArgs = {'ContentType': 'text/html',

'ACL': 'public-read'})

# List the gid-reports bucket objects starting with 2019/

objects\_list = s3.list\_objects(Bucket='gid-reports', Prefix='2019/')

# Convert the response contents to DataFrame

objects\_df = pd.DataFrame(objects\_list['Contents'])

# Create a column "Link" that contains Public Object URL

base\_url = "http://gid-reports.s3.amazonaws.com/"

objects\_df['Link'] = base\_url + objects\_df['Key']

# Preview the resulting DataFrame

objects\_df.head()

# Write objects\_df to an HTML file

objects\_df.to\_html('report\_listing.html',

# Set clickable links

render\_links=True,

# Isolate the columns

columns=['Link', 'LastModified', 'Size'])

# Overwrite index.html key by uploading the new file

s3.upload\_file(

Filename='./report\_listing.html', Key='index.html',

Bucket='gid-reports',

ExtraArgs = {

'ContentType': 'text/html',

'ACL': 'public-read'

})

create an alerting system for Council!

* Initialize the boto3 client for SNS.
* Create the 'city\_alerts' topic and extract its topic ARN.
* Re-create the 'city\_alerts' topic and extract its topic ARN with a one-liner.
* Verify the two topic ARNs match.

# Initialize boto3 client for SNS

sns = boto3.client('sns',

region\_name='us-east-1',

aws\_access\_key\_id=AWS\_KEY\_ID,

aws\_secret\_access\_key=AWS\_SECRET)

# Create the city\_alerts topic

response = sns.create\_topic(Name="city\_alerts")

c\_alerts\_arn = response['TopicArn']

# Re-create the city\_alerts topic using a oneliner

c\_alerts\_arn\_1 = sns.create\_topic(Name='city\_alerts')['TopicArn']

# Compare the two to make sure they match

print(c\_alerts\_arn == c\_alerts\_arn\_1)

Managers will subscribe only to critical notifications, while supervisors can monitor general notifications.

create a tiered topic structure... and have friends again!

* For every department, create a general topic.
* For every department, create a critical topic.
* Print all the topics created in SNS

# Create list of departments

departments = ['trash', 'streets', 'water']

for dept in departments:

# For every department, create a general topic

sns.create\_topic(Name="{}\_general".format(dept))

# For every department, create a critical topic

sns.create\_topic(Name="{}\_critical".format(dept))

# Print all the topics in SNS

response = sns.list\_topics()

print(response['Topics'])

Remove any topics that do not have the word critical in them.

* Get the current list of topics.
* For every topic ARN, if it doesn't have the word 'critical' in it, delete it.
* Print the list of remaining critical topics.

# Get the current list of topics

topics = sns.list\_topics()['Topics']

for topic in topics:

# For each topic, if it is not marked critical, delete it

if "critical" not in topic['TopicArn']:

sns.delete\_topic(TopicArn=topic['TopicArn'])

# Print the list of remaining critical topics

print(sns.list\_topics()['Topics'])

topic ARN for streets\_critical in the str\_critical\_arn variable.

Help Sam subscribe her first Council member to the streets\_critical topic!

* Subscribe Elena's phone number to the 'streets\_critical' topic.
* Print the SMS subscription ARN.
* Subscribe Elena's email to the 'streets\_critical topic.
* Print the email subscription ARN.

# Subscribe Elena's phone number to streets\_critical topic

resp\_sms = sns.subscribe(

TopicArn = str\_critical\_arn,

Protocol='sms', Endpoint="+16196777733")

# Print the SubscriptionArn

print(resp\_sms['SubscriptionArn'])

# Subscribe Elena's email to streets\_critical topic.

resp\_email = sns.subscribe(

TopicArn = str\_critical\_arn,

Protocol='email', Endpoint="eblock@sandiegocity.gov")

# Print the SubscriptionArn

print(resp\_email['SubscriptionArn'])

As master of all information by adding all the users in her CSV to the streets\_critical topic!

* For each element in the Email column of contacts, create a subscription to the 'streets\_critical' Topic.
* List subscriptions for the 'streets\_critical' Topic and convert them to a DataFrame.
* Preview the DataFrame.

# For each email in contacts, create subscription to street\_critical

for email in contacts['Email']:

sns.subscribe(TopicArn = str\_critical\_arn,

# Set channel and recipient

Protocol = 'email',

Endpoint = email)

# List subscriptions for streets\_critical topic, convert to DataFrame

response = sns.list\_subscriptions\_by\_topic(

TopicArn = str\_critical\_arn)

subs = pd.DataFrame(response['Subscriptions'])

# Preview the DataFrame

subs.head()

remove all SMS subscribers and make this an email only alerting system.

* List subscriptions for 'streets\_critical' topic.
* For each subscription, if the protocol is 'sms', unsubscribe.
* List subscriptions for 'streets\_critical' topic in one line.
* Print the subscriptions

# List subscriptions for streets\_critical topic.

response = sns.list\_subscriptions\_by\_topic(

TopicArn = str\_critical\_arn)

# For each subscription, if the protocol is SMS, unsubscribe

for sub in response['Subscriptions']:

if sub['Protocol'] == 'sms':

sns.unsubscribe(SubscriptionArn=sub['SubscriptionArn'])

# List subscriptions for streets\_critical topic in one line

subs = sns.list\_subscriptions\_by\_topic(

TopicArn=str\_critical\_arn)['Subscriptions']

# Print the subscriptions

print(subs)

check the current backlog count and send a message only if it exceeds 100.

The fate of District 12, and the results of Elena's election rest on your and Sam's shoulders.

* If there are over 100 potholes, send a message with the current backlog count.
* Create the email subject to also include the current backlog counit.
* Publish message to the streets\_critical Topic ARN.

# If there are over 100 potholes, create a message

if streets\_v\_count > 100:

# The message should contain the number of potholes.

message = "There are {} potholes!".format(streets\_v\_count)

# The email subject should also contain number of potholes

subject = "Latest pothole count is {}".format(streets\_v\_count)

# Publish the email to the streets\_critical topic

sns.publish(

TopicArn = str\_critical\_arn,

# Set subject and message

Message = message,

Subject = subject

)

Sam has created the boto3 SNS client and stored it in the sns variable. The contacts variable contains Elena's contacts as a DataFrame.

Help Sam put together a quick hello to Elena's largest supporters!

* For every contact, send an ad-hoc SMS to the contact's phone number.
* The message sent should include the contact's name.

# Loop through every row in contacts

for idx, row in contacts.iterrows():

# Publish an ad-hoc sms to the user's phone number

response = sns.publish(

# Set the phone number

PhoneNumber = str(row['Phone']),

# The message should include the user's name

Message = 'Hello {}'.format(row['Name'])

)

print(response)

make the City run faster!

You will create multi-level topics with a different set of subscribers that trigger based on different thresholds.

You will effectively be building a smart alerting system!

* For each department create a critical topic and store it in critical.
* For each department, create an extreme topic and store it in extreme.
* Place the created TopicArns into dept\_arns.
* Print the dictionary.

Help Sam subscribe the users in the contacts DataFrame to email or SMS notifications based on their department. This will help get the right alerts to the right people, making the City of San Diego run better and faster!

* Get the topic name by using the 'Department' field in the contacts DataFrame.
* Use the topic name to create the critical and extreme TopicArns for a user's department.
* Subscribe the user's email address to the critical topic.
* Subscribe the user's phone number to the extreme topic.

She has done some calculations and came up with a vcounts dictionary, that contains current requests for 'water', 'streets' and 'trash'.

In this exercise, you will help Sam publish a critical and an extreme alert based on the thresholds!

* If there are over 100 water violations, publish to 'water\_critical' topic.
* If there are over 300 water violations, publish to 'water\_extreme' topic.

dept\_arns = {}

for dept in departments:

# For each deparment, create a critical topic

critical = sns.create\_topic(Name="{}\_critical".format(dept))

# For each department, create an extreme topic

extreme = sns.create\_topic(Name="{}\_extreme".format(dept))

# Place the created TopicARNs into a dictionary

dept\_arns['{}\_critical'.format(dept)] = critical['TopicArn']

dept\_arns['{}\_extreme'.format(dept)] = extreme['TopicArn']

# Print the filled dictionary.

print(dept\_arns)

for index, user\_row in contacts.iterrows():

# Get topic names for the users's dept

critical\_tname = '{}\_critical'.format(user\_row['Department'])

extreme\_tname = '{}\_extreme'.format(user\_row['Department'])

# Get or create the TopicArns for a user's department.

critical\_arn = sns.create\_topic(Name=critical\_tname)['TopicArn']

extreme\_arn = sns.create\_topic(Name=extreme\_tname)['TopicArn']

# Subscribe each users email to the critical Topic

sns.subscribe(TopicArn = critical\_arn,

Protocol='email', Endpoint=user\_row['Email'])

# Subscribe each users phone number for the extreme Topic

sns.subscribe(TopicArn = extreme\_arn,

Protocol='sms', Endpoint=str(user\_row['Phone']))

if vcounts['water'] > 100:

# If over 100 water violations, publish to water\_critical

sns.publish(

TopicArn = dept\_arns['water\_critical'],

Message = "{} water issues".format(vcounts['water']),

Subject = "Help fix water violations NOW!")

if vcounts['water'] > 300:

# If over 300 violations, publish to water\_extreme

sns.publish(

TopicArn = dept\_arns['water\_extreme'],

Message = "{} violations! RUN!".format(vcounts['water']),

Subject = "THIS IS BAD. WE ARE FLOODING!")

Sam has also created the boto3 Rekognition client in the rekog variable.

Help Sam use Rekognition to enable the animal control team to rescue stray cats!

Use the Rekognition client to detect the labels for image1. Return a maximum of 1 label.

Detect the labels for image2 and print the response's labels.

# Use Rekognition client to detect labels

image1\_response = rekog.detect\_labels(

# Specify the image as an S3Object; Return one label

Image=image1, MaxLabels=1)

# Print the labels

print(image1\_response['Labels'])

# Use Rekognition client to detect labels

image2\_response = rekog.detect\_labels(Image=image2, MaxLabels=1)

# Print the labels

print(image2\_response['Labels'])

* Created the Rekognition client.
* Called .detect\_labels() with the Bucket and Key of the image on S3.
* Stored the result in the response variable.

Help Sam save cat lives! Help her count the cats in each image and include that in the alert to Animal Control!

* Iterate over each element of the 'Labels' key in response.
* Once you encounter a label with the name 'Cat', iterate over the label's instance.
* If an instance's confidence level exceeds 85, increment cat\_counts by 1.
* Print the final cat count.

# Create an empty counter variable

cats\_count = 0

# Iterate over the labels in the response

for label in response['Labels']:

# Find the cat label, look over the detected instances

if label['Name'] == 'Cat':

for instance in label['Instances']:

# Only count instances with confidence > 85

if (instance['Confidence'] > 85):

cats\_count += 1

# Print count of cats

print(cats\_count)

* Created the Rekognition client.
* Called .detect\_text() and stored the result in response.

Help Sam create a data source for parking regulations in the City. Use Rekognition's .detect\_text() method to extract lines and words from images of parking signs.

Iterate over each detected text in response, and append each detected text to words if the text's type is 'WORD'.

Iterate over each detected text in response, and append each detected text to lines if the text's type is 'LINE'.

# Create empty list of words

words = []

# Iterate over the TextDetections in the response dictionary

for text\_detection in response['TextDetections']:

# If TextDetection type is WORD, append it to words list

if text\_detection['Type'] == 'WORD':

# Append the detected text

words.append(text\_detection['DetectedText'])

# Print out the words list

print(words)

# Create empty list of lines

lines = []

# Iterate over the TextDetections in the response dictionary

for text\_detection in response['TextDetections']:

# If TextDetection type is Line, append it to lines list

if text\_detection['Type'] == 'LINE':

# Append the detected text

lines.append(text\_detection['DetectedText'])

# Print out the words list

print(lines)

Quantify the demand for a Spanish version of the Get It Done application. Figure out how many requesters use Spanish and print the final result!

* For each row in the DataFrame, detect the dominant language.
* Assign the first selected language to the 'lang' column.
* Count the total number of posts in Spanish.

# For each dataframe row

for index, row in dumping\_df.iterrows():

# Get the public description field

description =dumping\_df.loc[index, 'public\_description']

if description != '':

# Detect language in the field content

resp = comprehend.detect\_dominant\_language(Text=description)

# Assign the top choice language to the lang column.

dumping\_df.loc[index, 'lang'] = resp['Languages'][0]['LanguageCode']

# Count the total number of spanish posts

spanish\_post\_ct = len(dumping\_df[dumping\_df.lang == 'es'])

# Print the result

print("{} posts in Spanish".format(spanish\_post\_ct))

Translate the requests to Spanish by running them through the AWS translate service!

* For each row in the DataFrame, translate it to English.
* Store the original language in the original\_lang column.
* Store the new translation in the translated\_desc column.

for index, row in dumping\_df.iterrows():

# Get the public\_description into a variable

description = dumping\_df.loc[index, 'public\_description']

if description != '':

# Translate the public description

resp = translate.translate\_text(

Text=description,

SourceLanguageCode='auto', TargetLanguageCode='en')

# Store original language in original\_lang column

dumping\_df.loc[index, 'original\_lang'] = resp['SourceLanguageCode']

# Store the translation in the translated\_desc column

dumping\_df.loc[index, 'translated\_desc'] = resp['TranslatedText']

# Preview the resulting DataFrame

dumping\_df = dumping\_df[['service\_request\_id', 'original\_lang', 'translated\_desc']]

dumping\_df.head()

Better understand the moods of the voices of the people that submit Get It Done cases, and whether they are coming into the interaction with the City in a positive mood or a negative one.

* Detect the sentiment of 'public\_description' for every row.
* Store the result in the 'sentiment' column.

for index, row in dumping\_df.iterrows():

# Get the translated\_desc into a variable

description = dumping\_df.loc[index, 'public\_description']

if description != '':

# Get the detect\_sentiment response

response = comprehend.detect\_sentiment(

Text=description,

LanguageCode='en')

# Get the sentiment key value into sentiment column

dumping\_df.loc[index, 'sentiment'] = response['Sentiment']

# Preview the dataframe

dumping\_df.head()

Understand sentiment across many different languages. This will help the City understand how different communities are relating to scooters, something that will affect the votes of City Council members.

* For every DataFrame row, detect the dominant language.
* Use the detected language to determine the sentiment of the description.
* Group the DataFrame by the 'sentiment' and 'lang' columns in that order.

for index, row in scooter\_requests.iterrows():

# For every DataFrame row

desc = scooter\_requests.loc[index, 'public\_description']

if desc != '':

# Detect the dominant language

resp = comprehend.detect\_dominant\_language(Text=desc)

lang\_code = resp['Languages'][0]['LanguageCode']

scooter\_requests.loc[index, 'lang'] = lang\_code

# Use the detected language to determine sentiment

scooter\_requests.loc[index, 'sentiment'] = comprehend.detect\_sentiment(

Text=desc,

LanguageCode=lang\_code)['Sentiment']

# Perform a count of sentiment by group.

counts = scooter\_requests.groupby(['sentiment', 'lang']).count()

counts.head()

Implement a system that dispatches crews based on sentiment and image recognition. You will help Sam pair human and machine for effective City management!

* Get the SNS topic ARN for 'scooter\_notifications'.
* For every row, if sentiment is 'NEGATIVE' and there is an image of a scooter, construct a message to send.
* Publish the notification to the SNS topic.

# Get topic ARN for scooter notifications

topic\_arn = sns.create\_topic(Name='scooter\_notifications')['TopicArn']

for index, row in scooter\_requests.iterrows():

# Check if notification should be sent

if (row['sentiment'] == 'NEGATIVE') & (row['img\_scooter'] == 1):

# Construct a message to publish to the scooter team.

message = "Please remove scooter at {}, {}. Description: {}".format(

row['long'], row['lat'], row['public\_description'])

# Publish the message to the topic!

sns.publish(TopicArn = topic\_arn,

Message = message,

Subject = "Scooter Alert")