mongo\_Weather\_Template

March 25, 2025

# Weather Dataset

[ ]:

*# Notebook Template for working with the Weather dataset # import all the libraries needed first*

**import pymongo**

**from pprint import** pprint

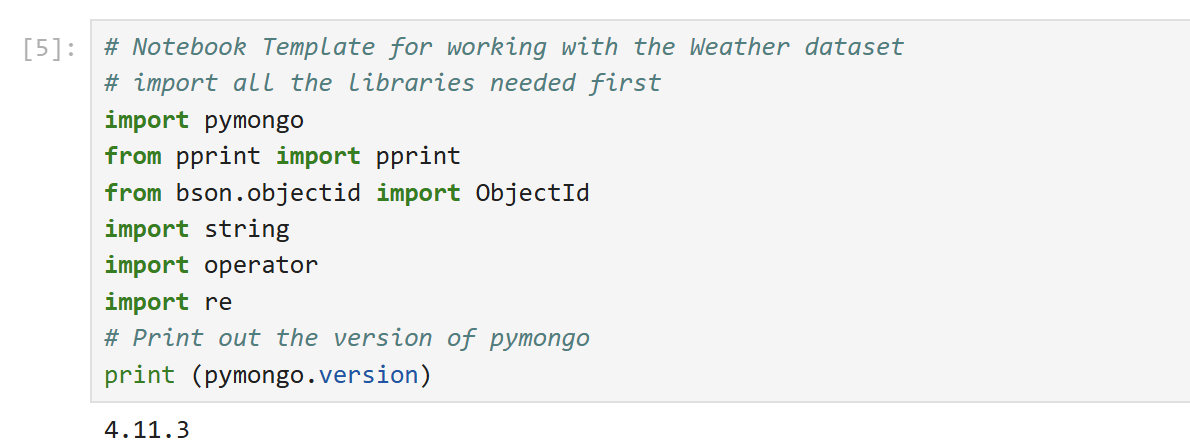
**from bson.objectid import** ObjectId

**import string**

**import operator import re**

*# Print out the version of pymongo*

print (pymongo.version)



[ ]:

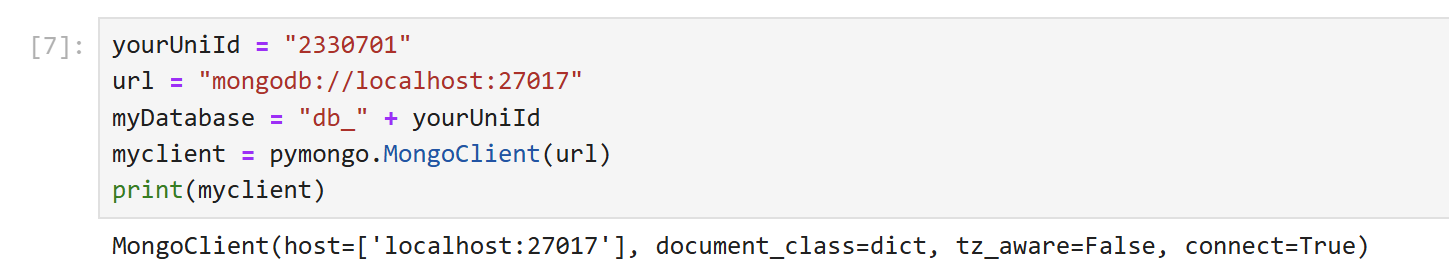
# Connecting to MongoDB

In the next cell you need to fill the below two variables with your *uniId* as well as *mongodb connection string*.

yourUniId = ""

url = "" *# Your copied connection string here*

myDatabase = "db\_" + yourUniId myclient = pymongo.MongoClient(url) print(myclient)

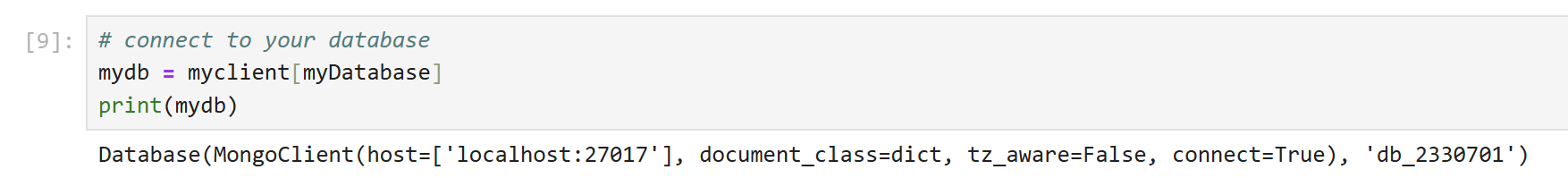


[ ]:

# Connecting to your Database

Next you need to connect to your own database. The variable ‘myDatabase’ has been set above.

*# connect to your database* mydb = myclient[myDatabase] print(mydb)



Ping the MongoDB server to check if the connection is active and working properly.

[ ]:

myclient.admin.command('ping')



[ ]:

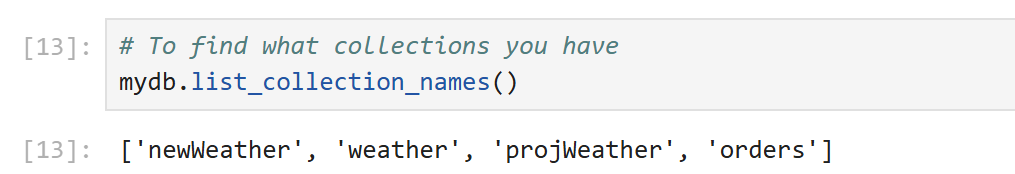
# Using MongoDB from Python

The syntax for running commands in Python is slightly different from the shell. mydb above contains a reference to your database.

Most commands seen in the Mongo Shell have equivalent functions in Python, the syntax can be slightly different. The main difference is the commands below do not need the “db.” suﬀix seen when using the Mongo Shell.

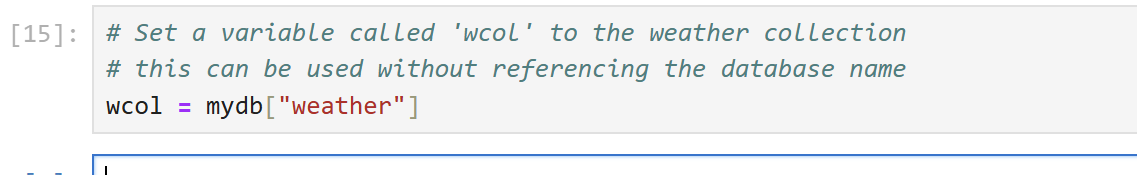
*# To find what collections you have*

mydb.list\_collection\_names()



[ ]:

*# Set a variable called 'wcol' to the weather collection # this can be used without referencing the database name* wcol = mydb["weather"]



[ ]:

# Workbook Examples

The following shows examples from the MongoDB workbook. The Mongo Client command is shown as a comment.

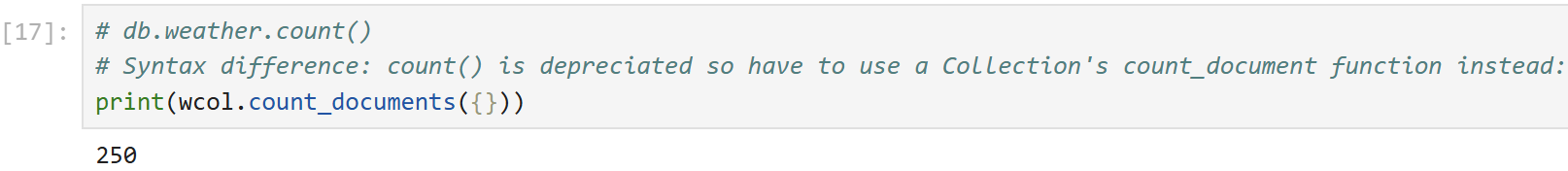
The comments will also mention any variations in syntax.

*# db.weather.count()*

*# Syntax difference: count() is depreciated so have to use a Collection's*␣

𝗌*count\_document function instead:*

print(wcol.count\_documents({}))

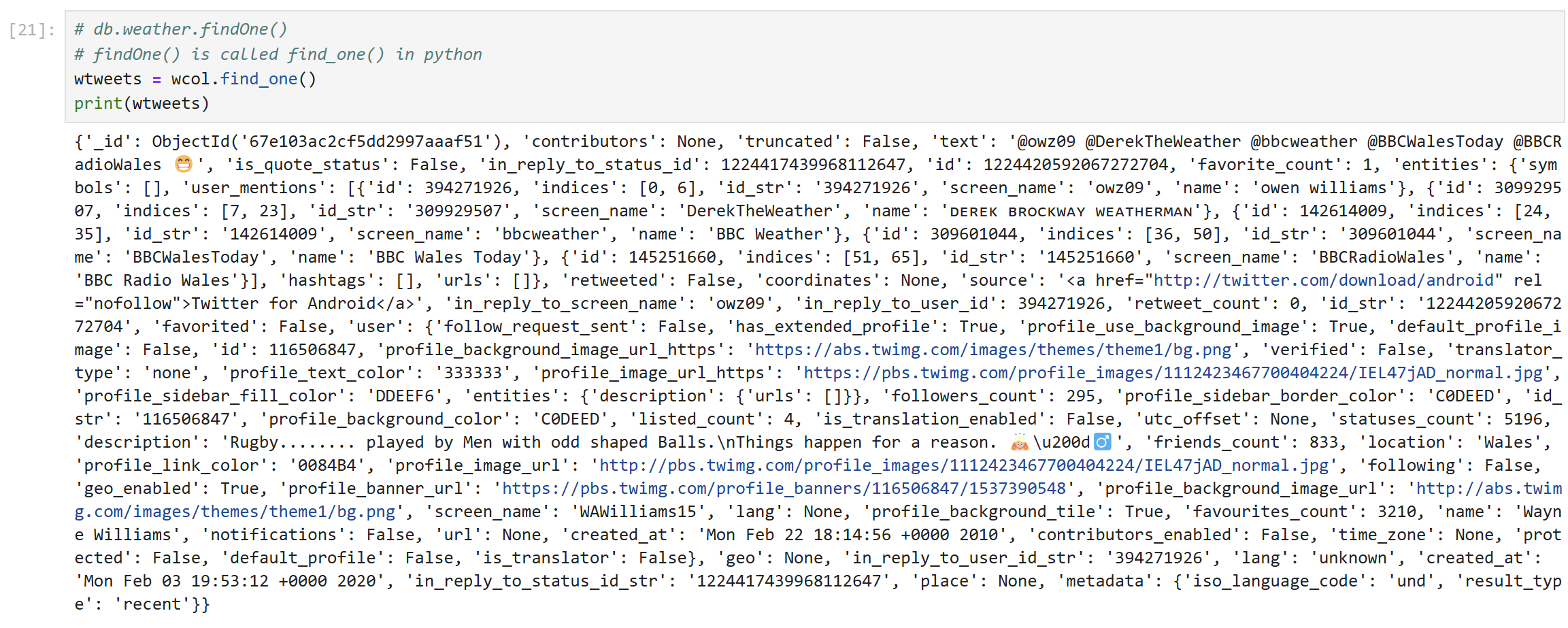


[ ]:

*# db.weather.findOne()*

*# findOne() is called find\_one() in python*

wtweets = wcol.find\_one() print(wtweets)



[ ]:

*# Syntax difference: pretty() is not a function in Python. # Use pprint instead*

pprint(wtweets)



[ ]:

*# Show distinct languages*

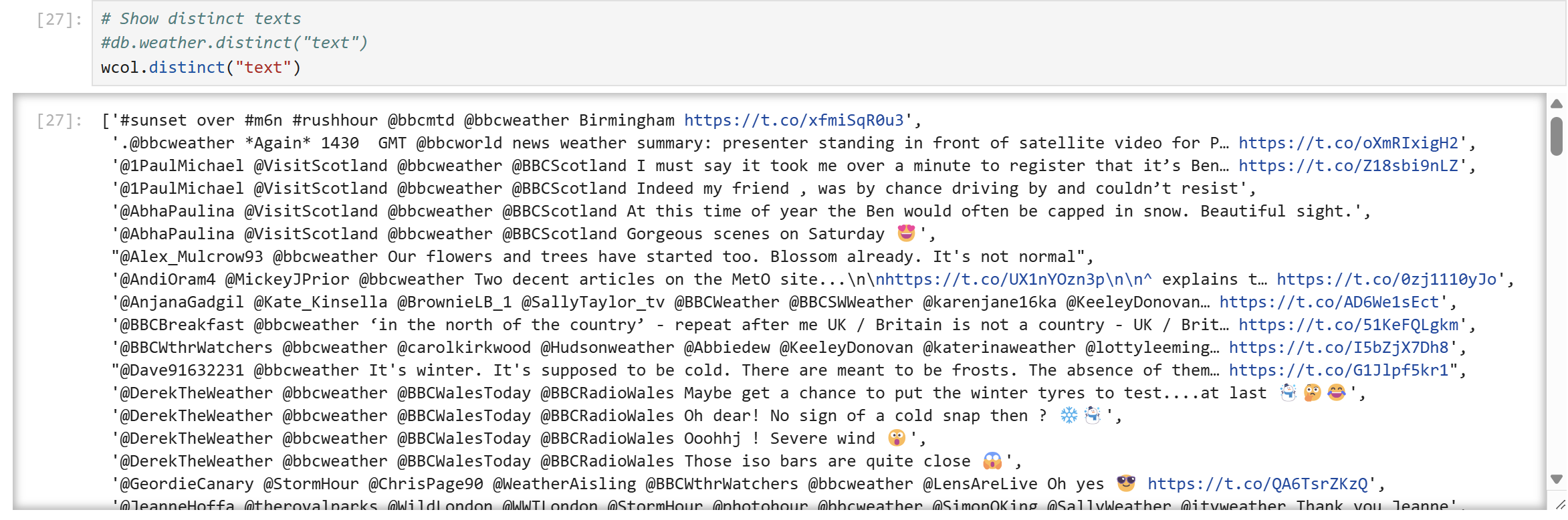
*# db.weather.distinct("lang")*

wcol.distinct("lang")



[ ]:

*# Show distinct texts #db.weather.distinct("text")* wcol.distinct("text")



[ ]:

*# find distinct user names*

*# db.weather.distinct("user.name")*

wcol.distinct("user.name")

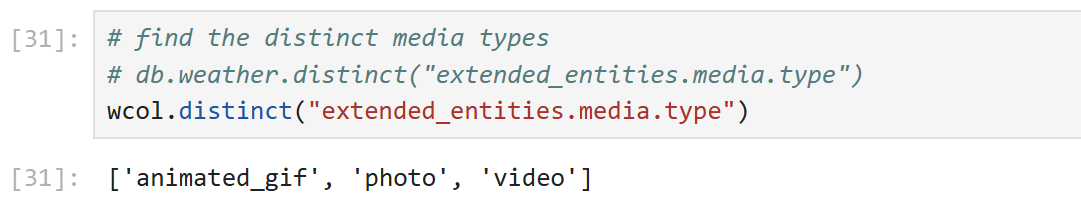


[ ]:

*# find the distinct media types*

*# db.weather.distinct("extended\_entities.media.type")*

wcol.distinct("extended\_entities.media.type")



[ ]:

# Print Functions

We are going to print the output from the find() command regularly, and later from running update commands so create some functions to do aid this.

*# function to iterate through the tweets found*

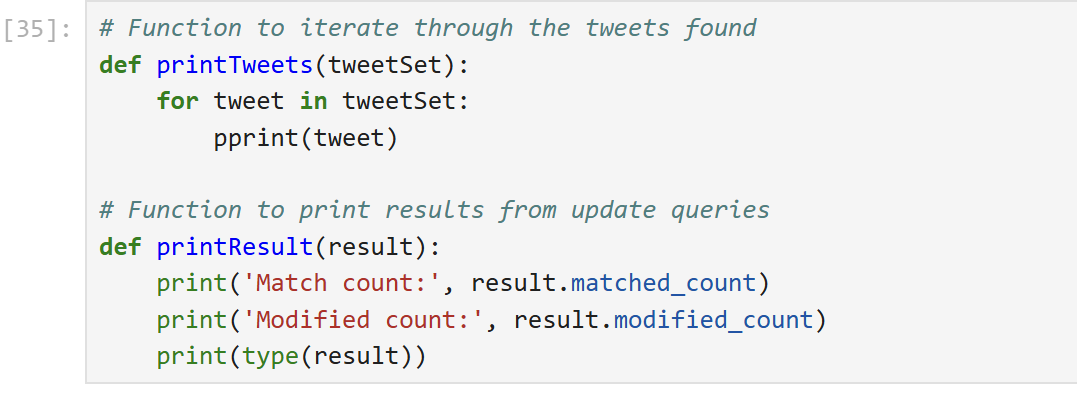
**def** printTweets(tweetSet):

**for** tweet **in** tweetSet: pprint (tweet)

*# function to print results from update queries*

**def** printResult(result):

print ('Match count: ', result.matched\_count) print ('Modified count: ', result.modified\_count) print (type(result))



[ ]:

*# db.weather.find({"user.name":"BBC Highlands"}, {id: 1, \_id:0}).pretty() # Syntax difference: attribute names must be in quotes in Python*

tweets = wcol.find({"user.name":"BBC Highlands"}, {"id": 1, "\_id":0})

*# call the printTweets function*

printTweets(tweets)



# Pattern Matching

[ ]:

*# search for "sun"*

*# db.weather.find({text: "sun"})*

tweets = wcol.find({"text": "sun"}) printTweets(tweets)

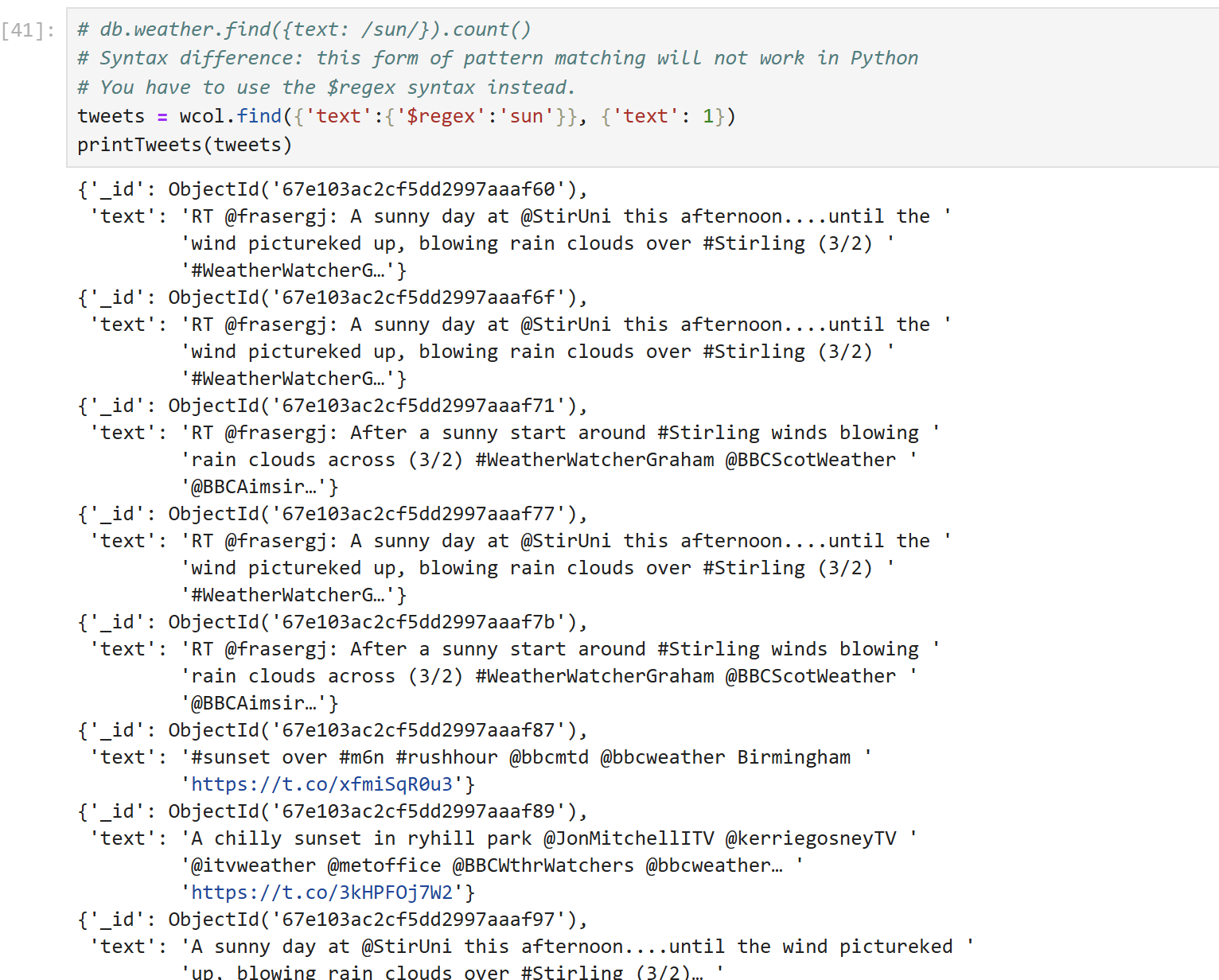


[ ]:

*# db.weather.find({text: /sun/}).count()*

*# Syntax difference: this form of pattern matching will not work in Python # You have to use the $regex syntax instead.*

tweets = wcol.find({'text':{'$regex':'sun'}}, {'text': 1}) printTweets(tweets)



[ ]:

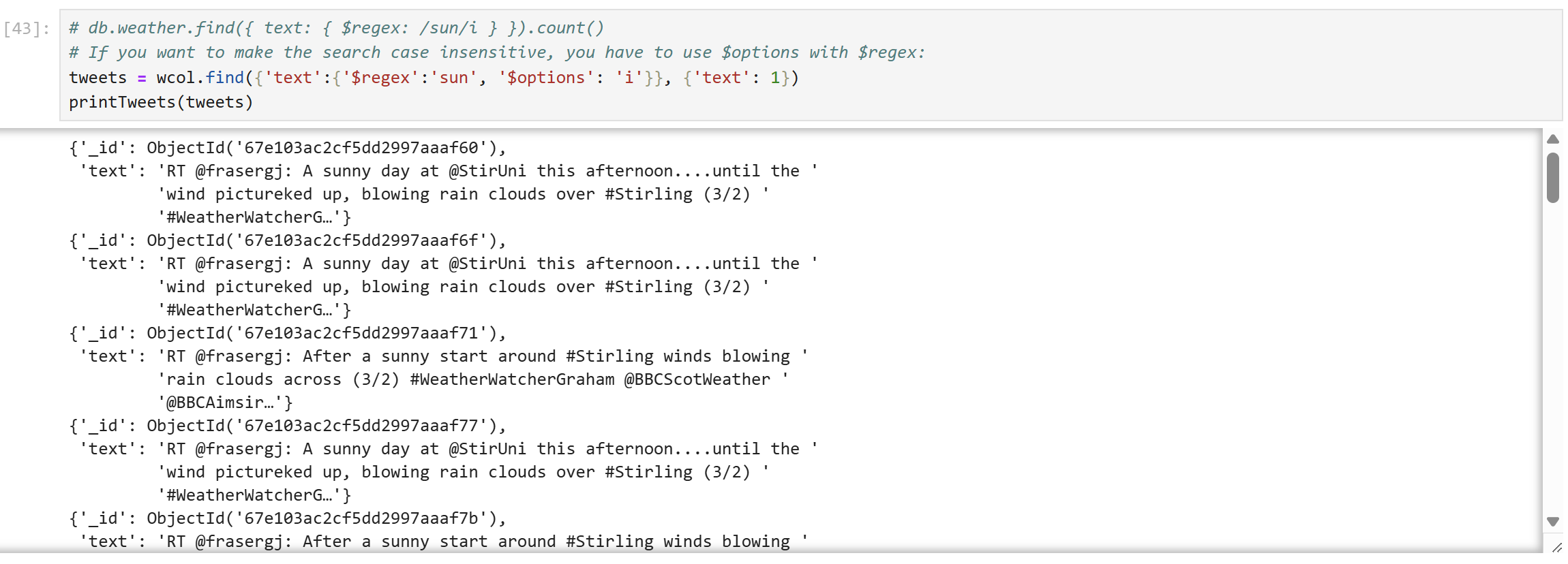
*# db.weather.find({ text: { $regex: /sun/i } }).count()*

*# If you want to make the search case insensitive, you have to use $options*␣

𝗌*with $regex:*

tweets = wcol.find({'text':{'$regex':'sun', '$options': 'i'}}, {'text': 1})

printTweets(tweets)



[ ]:

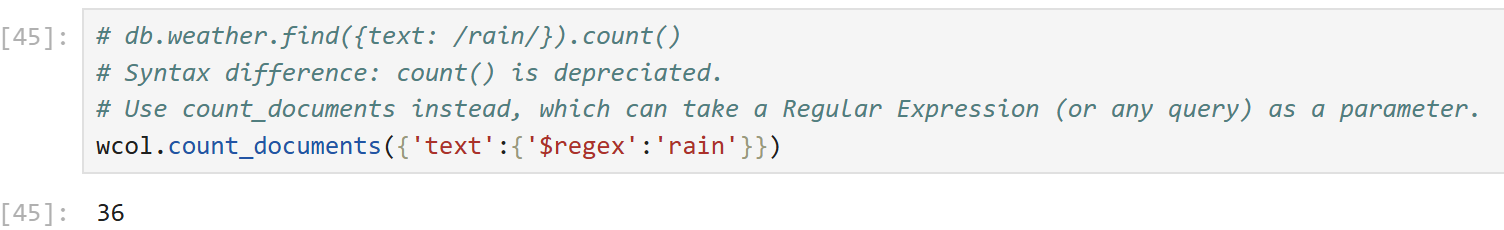
*# db.weather.find({text: /rain/}).count()*

*# Syntax difference: count() is depreciated.*

*# Use count\_documents instead, which can take a Regular Expression (or any*␣

𝗌*query) as a parameter.*

wcol.count\_documents({'text':{'$regex':'rain'}})



[ ]:

*# find retweets - starting with RT*

*# db.weather.find({ text: { $regex: /^RT/i } }).count()*

wcol.count\_documents({'text':{'$regex':'^RT','$options': 'i'}})

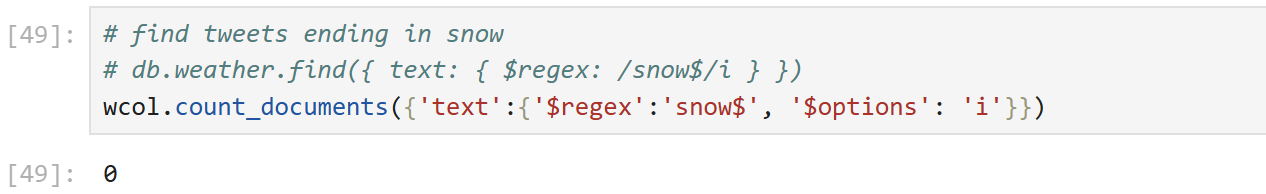


[ ]:

*# find tweets ending in snow*

*# db.weather.find({ text: { $regex: /snow$/i } })*

wcol.count\_documents({'text':{'$regex':'snow$', '$options': 'i'}})



[ ]:

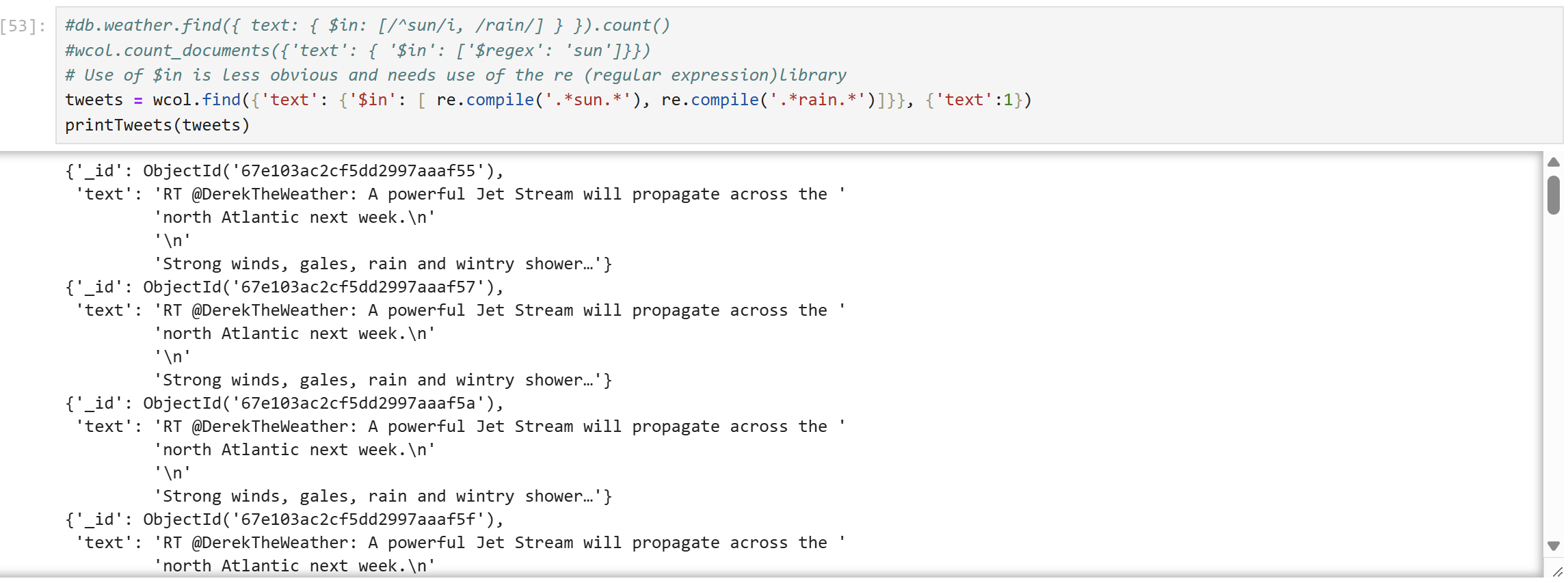
*#db.weather.find({ text: { $in: [/^sun/i, /rain/] } }).count() #wcol.count\_documents({'text': { '$in': ['$regex': 'sun']}})*

*# Use of $in is less obvious and needs use of the re (regular expression*␣

𝗌*)library*

tweets = wcol.find({'text': {'$in': [ re.compile('.\*sun.\*'), re.compile('.\*rain.

𝗌\*')]}}, {'text':1}) printTweets(tweets)



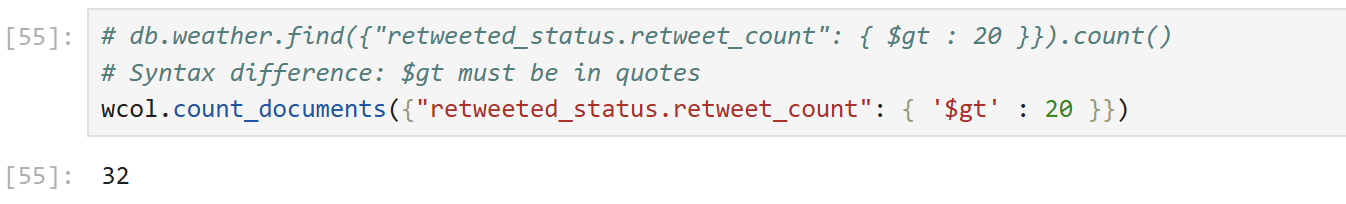
[ ]:

# Further Queries

The following shows examples of: - operators - handling null and boolean data - aggregation

*# db.weather.find({"retweeted\_status.retweet\_count": { $gt : 20 }}).count() # Syntax difference: $gt must be in quotes*

wcol.count\_documents({"retweeted\_status.retweet\_count": { '$gt' : 20 }})



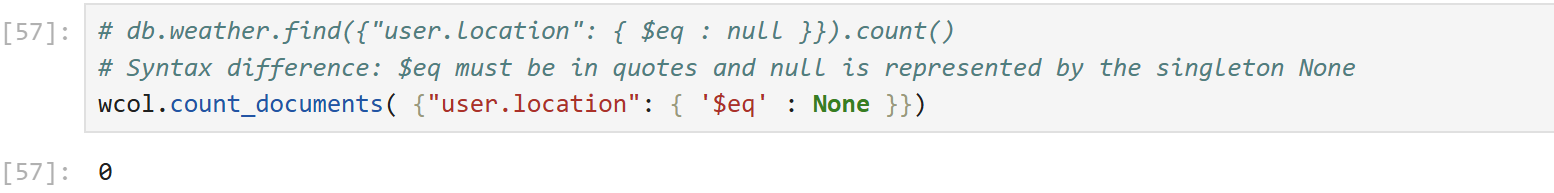
[ ]:

*# db.weather.find({"user.location": { $eq : null }}).count()*

*# Syntax difference: $eq must be in quotes and null is represented by the*␣

𝗌*singleton None*

wcol.count\_documents( {"user.location": { '$eq' : **None** }})



[ ]:

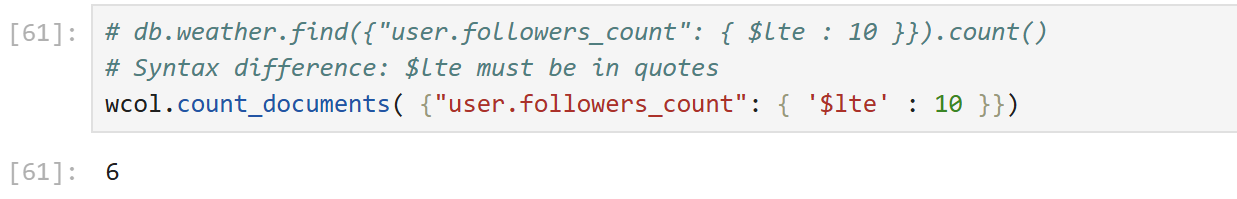
*# db.weather.find({"user.geo\_enabled": { $eq : true }}).count() # Syntax difference: true is represented by True* wcol.count\_documents( {"user.geo\_enabled": {'$eq' : **True** }})



[ ]:

*# db.weather.find({"user.followers\_count": { $lte : 10 }}).count() # Syntax difference: $lte must be in quotes*

wcol.count\_documents( {"user.followers\_count": { '$lte' : 10 }})



[ ]:

*# GROUP BY command and aggregation #db.weather.aggregate( [ {*

*# # # # #*

*$group: {*

*\_id : null,*

*totalAmount: { $sum: "$retweeted\_status.retweet\_count" }, count: { $sum: 1 }*

*} } ] )*

*# Syntax difference: all the parameters must be in quotes*

grpBy = wcol.aggregate( [

{

'$group': {

'\_id' : **None**,

'totalAmount': { '$sum': "$retweeted\_status.retweet\_count" }, 'count': { '$sum': 1 }

}

}

] )

printTweets(grpBy)



[ ]:

# Cleaning Data

For cleaning purposes you may wish to clean Twitter data before producing statistics on it. For example, to remove typing errors, make abbreviations consistent, or remove unnecessary data that is not needed.

MongoDB’s various update() methods can be used to update document into a collection, however, these do restrict you to replacing the whole document. This is not so good if you just want to replace a substring.

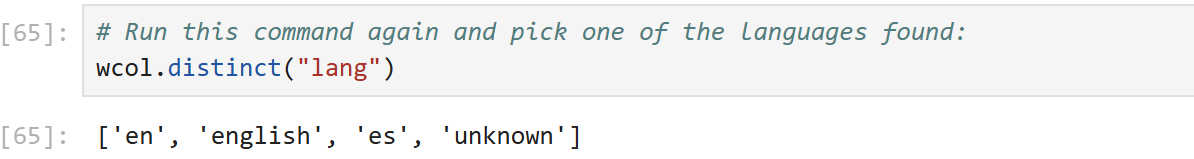
Updating substrings has to be done currently via the Mongo Shell. Let’s make the language code more user-friendly.

See this page for the language associated with each code: https://developer.twitter.com/en/docs/twitter-for-websites/twitter-for-websites-supported- languages/overview

For example, en means English, which is the default. Let’s change one of the languages to the full name.

*# Run this command again and pick one of the languages found:*

wcol.distinct("lang")



[ ]

*# You need to find the ObjectIds in your data # For example, es is Spanish #db.weather.find({ lang: "es"}, {lang:1})*

langs = wcol.find({ 'lang': "Replace\_this\_with\_a\_language\_code"}, {'lang':1})

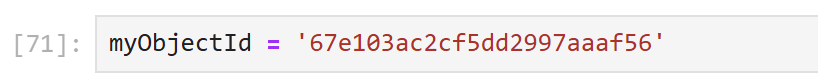
printTweets(langs)



Pick one of the Object Ids seen above and substitute myObjectId below with the value.

[ ]:

myObjectId = '67e100cb4245b488e87c52a6'

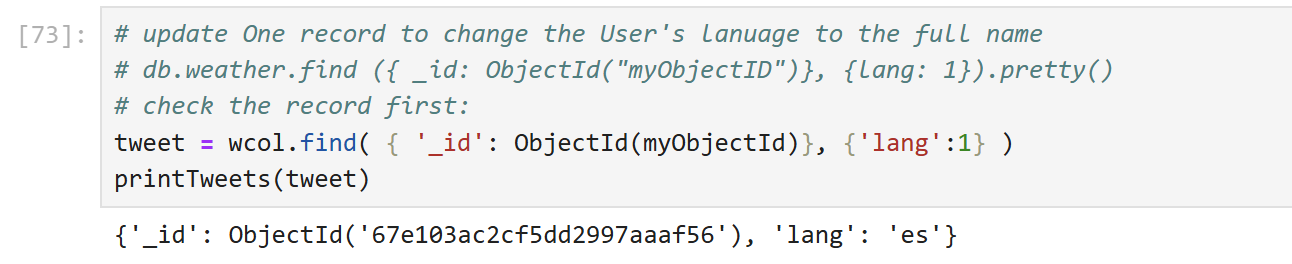


[ ]:

*# update One record to change the User's lanuage to the full name*

*# db.weather.find ({ \_id: ObjectId("myObjectID")}, {lang: 1}).pretty() # check the record first:*

tweet = wcol.find( { '\_id': ObjectId(myObjectId)}, {'lang':1} ) printTweets(tweet)



[ ]:

*# Replace with the full language, for example, to replace es with Spanish would*␣

𝗌*be:*

*#db.weather.update({ \_id: ObjectId("myObjectID") }, {$set: {'lang':*␣

𝗌*'Spanish'}})*

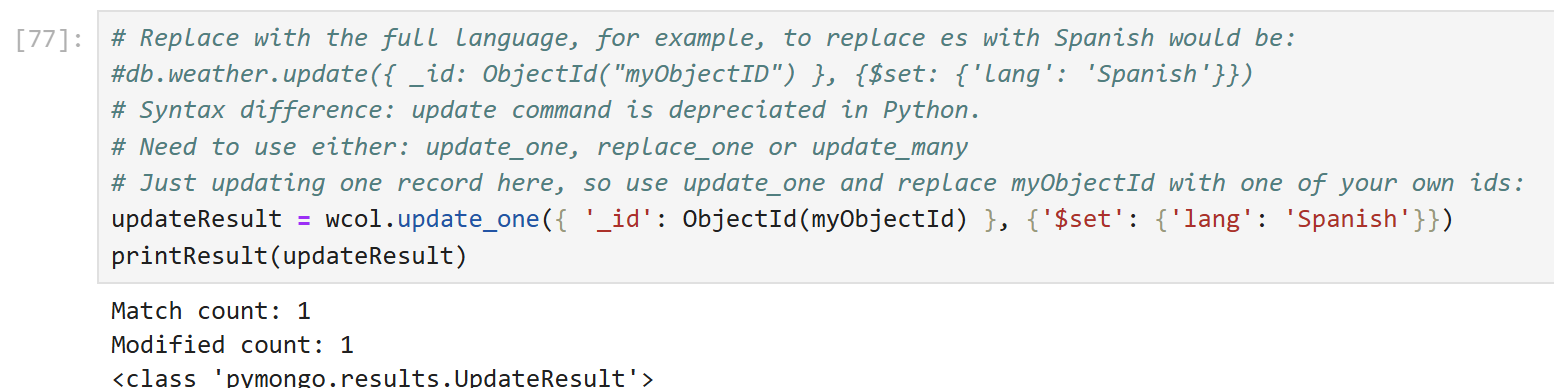
*# Syntax difference: update command is depreciated in Python. # Need to use either: update\_one, replace\_one or update\_many*

*# Just updating one record here, so use update\_one and replace myObjectId with*␣

𝗌*one of your own ids:*

updateResult = wcol.update\_one({ '\_id': ObjectId(myObjectId) }, {'$set': ␣

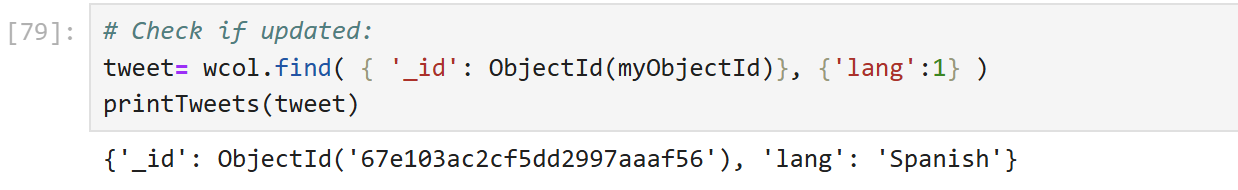
𝗌{'lang': 'English'}}) printResult(updateResult)



[ ]:

*# Check if updated:*

tweet= wcol.find( { '\_id': ObjectId(myObjectId)}, {'lang':1} ) printTweets(tweet)



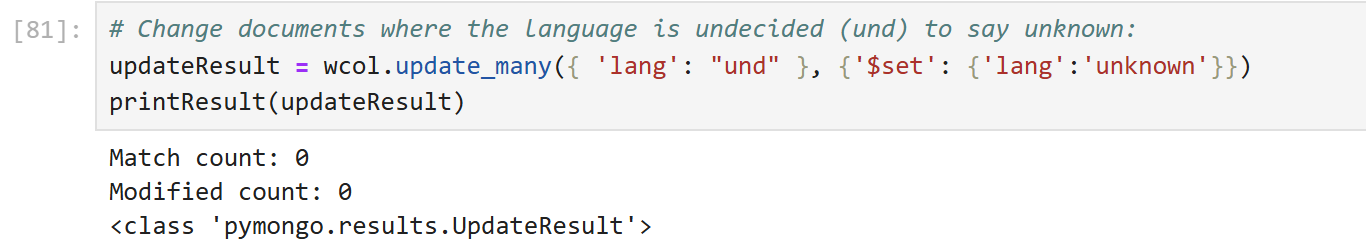
[ ]:

*# Change documents where the language is undecided (und) to say unknown:*

updateResult = wcol.update\_many({ 'lang': "und" }, {'$set': {'lang':␣

𝗌'unknown'}})

printResult(updateResult)



[ ]:

*# Want all the English locations to just say UK*

tweets = wcol.find( { 'user.location': {'$regex':'England','$options': 'i'␣

𝗌}},{'user.location':1, '\_id':0} ) printTweets(tweets)



[ ]:

*# db.weather.updateMany({ 'user.location': {$regex:'England',$options: 'i' }} ,*␣

𝗌 *{$set: {'user.location': 'UK'}})*

*# Syntax difference: update updateMany() is replaced by update\_many() # Note: there is no option to ROLLBACK the update!*

updateResult = wcol.update\_many({ 'user.location': {'$regex':

𝗌'England','$options': 'i' }} , {'$set': {'user.location': 'UK'}}) printResult(updateResult)

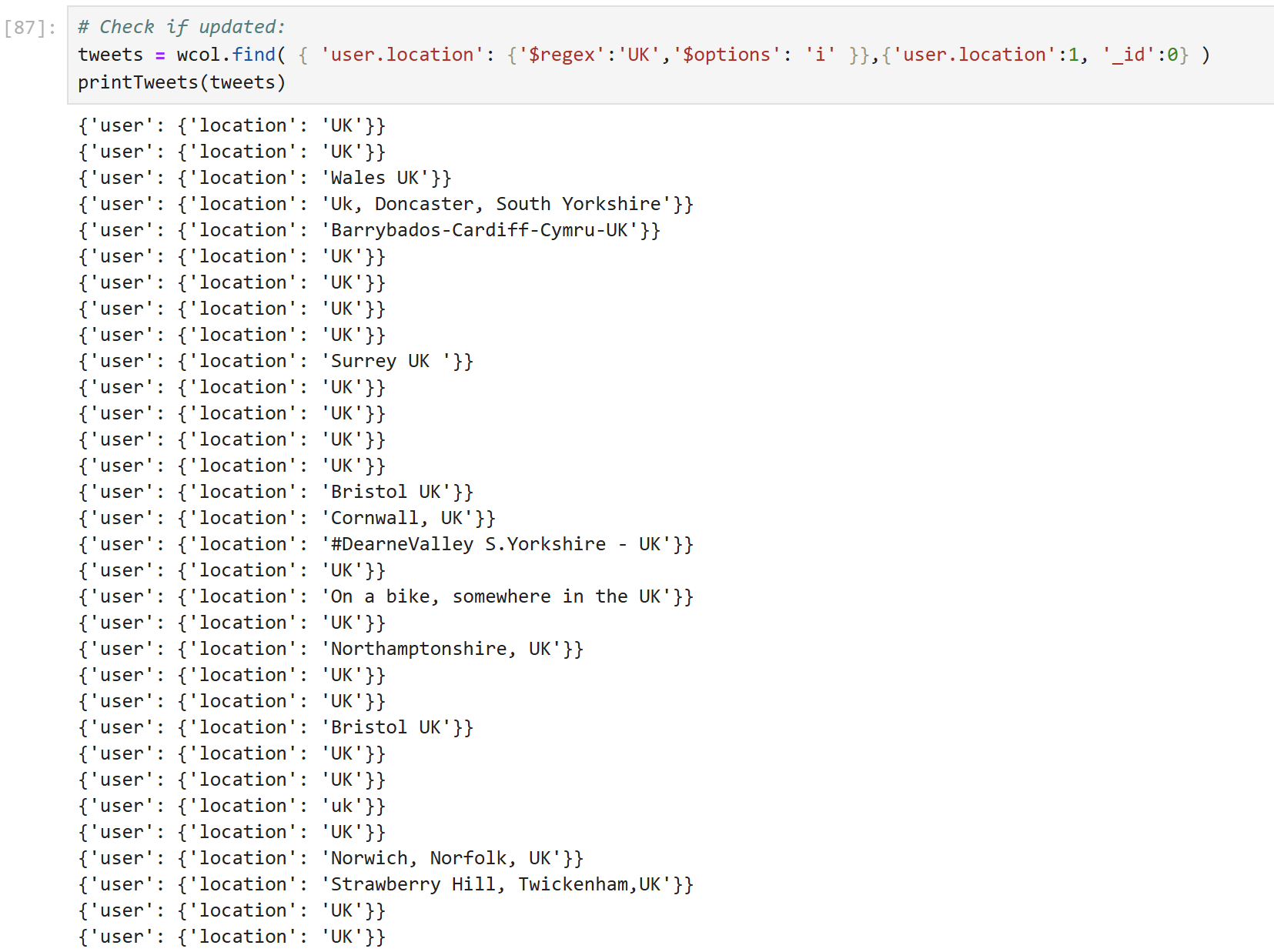


[ ]:

*# Check if updated:*

tweets = wcol.find( { 'user.location': {'$regex':'UK','$options': 'i' }},{'user.

𝗌location':1, '\_id':0} ) printTweets(tweets)



[ ]:

# Reshaping the Data

You may not need all the data in a collection. An aggregation pipeline can be used to pick the columns that you want, or limit the number of rows.

## Reduce the number of columns

*# Reduce the number of columns in a collection*

*# Creating a new collection with just 2 selected fields: user.location and the*␣

𝗌*text.*

*"""db.weather.aggregate([*

*{$project:{"user.location":1, text:1}},*

*{$out:"projWeather"}*

*]) """*

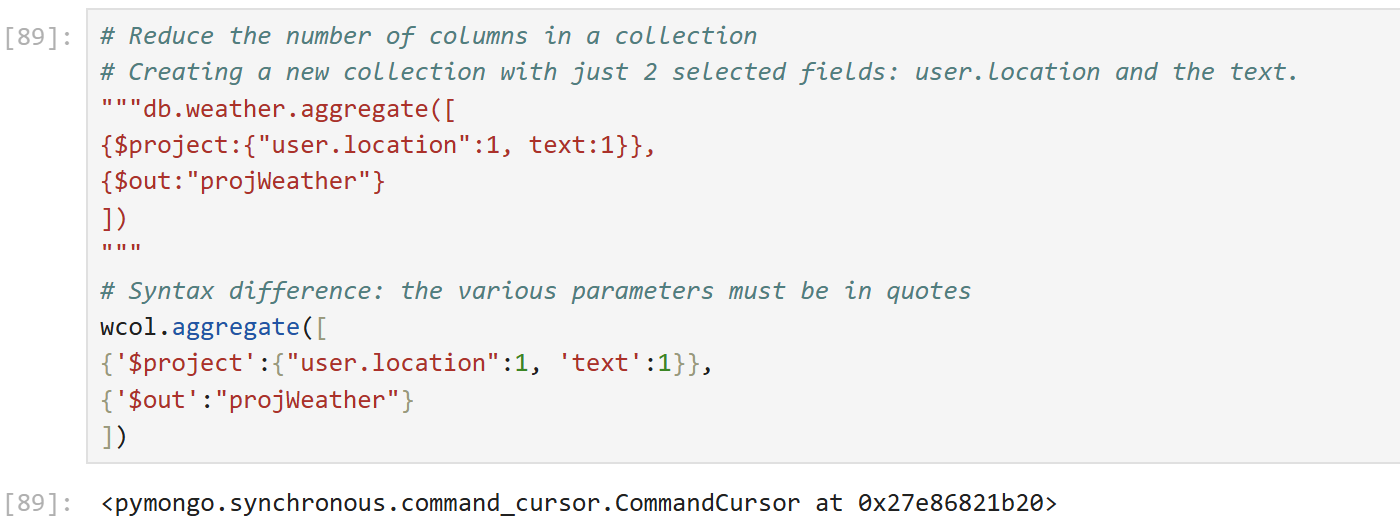
*# Syntax difference: the various parameters must be in quotes*

wcol.aggregate([

{'$project':{"user.location":1, 'text':1}},

{'$out':"projWeather"}

])

****

[ ]:

*# Check that the new collection exists with the required data: # db.newWeather.find().pretty()*

*# Create a variable called newcol to hold the new collection*

projcol = mydb["projWeather"] newData = (projcol.find({})) printTweets(newData)



[ ]:

## Reduce the number of rows.

Use $limit if you do not want to restrict the data based on a query, but want to reduce the number of rows. For example for some initial analysis

*"""db.projWeather.aggregate([*

*{$limit: 50},*

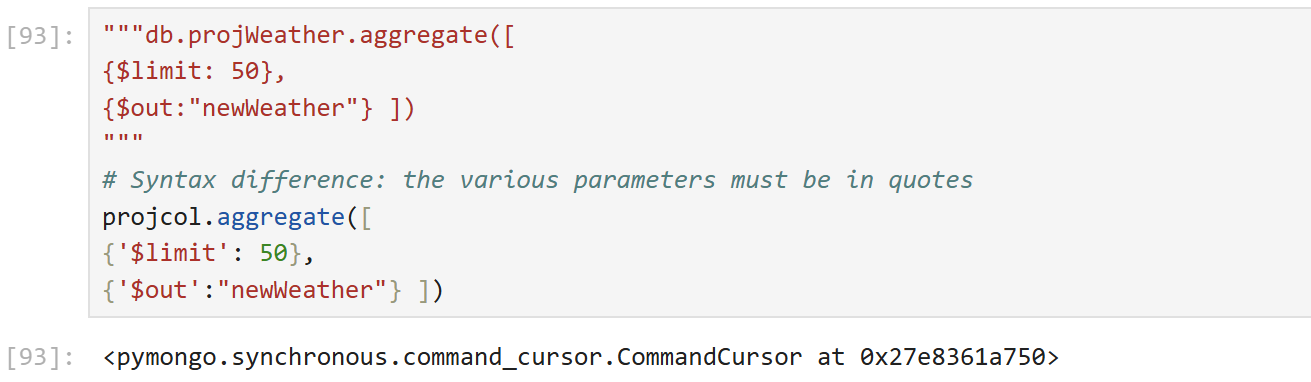
*{$out:"newWeather"} ]) """*

*# Syntax difference: the various parameters must be in quotes*

projcol.aggregate([

{'$limit': 50},

{'$out':"newWeather"} ])



[ ]:

*# Check that the new collection exists with the required data: # db.newWeather.count()*

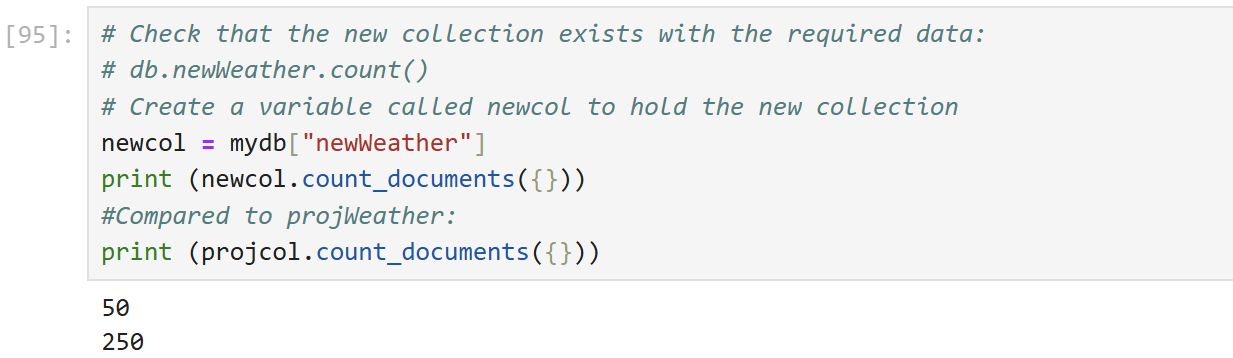
*# Create a variable called newcol to hold the new collection*

newcol = mydb["newWeather"]

print (newcol.count\_documents({}))

*#Compared to projWeather:*

print (projcol.count\_documents({}))



[ ]:

# Further Analysis

This code based on: h[ttps://www.kdnuggets.com/2017/03/beginners-guide-tweet-analytics-](http://www.kdnuggets.com/2017/03/beginners-guide-tweet-analytics-) pandas.html

It has been amended for MongoDB and Python. It has not been replicated in the Mongo Shell. This code counts the top Hashtags and mentions in any tweets where ‘sun’ is mentioned.

*# count\_features will search for Hashtags & Mentions in the text # It will then sort the results so can find the top 10 only*

*# print\_features prints out the results*

**def** print\_features(set, type): print ('**\n**Top 10 ', type, ':') print (' ')

**for** item **in** set:

print (item[0], '-', str(item[1]))

**def** count\_features(tweets): tag\_dict = {} mention\_dict = {}

tweet\_df = tweets

**for** i **in** tweet\_df:

*# convert dict to a string for processing*

tweet\_text = str(i)

tweet = tweet\_text.lower() tweet\_tokenized = tweet.split()

**for** word **in** tweet\_tokenized:

*# Hashtags - tokenize and build dict of tag counts*

**if** (word[0:1] == '#' **and** len(word) > 1):

key = word.translate(str.maketrans("",""))

**if** key **in** tag\_dict: tag\_dict[key] += 1

**else**:

tag\_dict[key] = 1

*# Mentions - tokenize and build dict of mention counts*

**if** (word[0:1] == '@' **and** len(word) > 1):

key = word.translate(str.maketrans("",""))

**if** key **in** mention\_dict: mention\_dict[key] += 1

**else**:

mention\_dict[key] = 1

*# The 10 most popular tags and counts*

*# Python 3 renamed dict.iteritems -> dict.items*

top\_tags = dict(sorted(tag\_dict.items(), key=operator.itemgetter(1),␣

𝗌reverse=**True**)[:10])

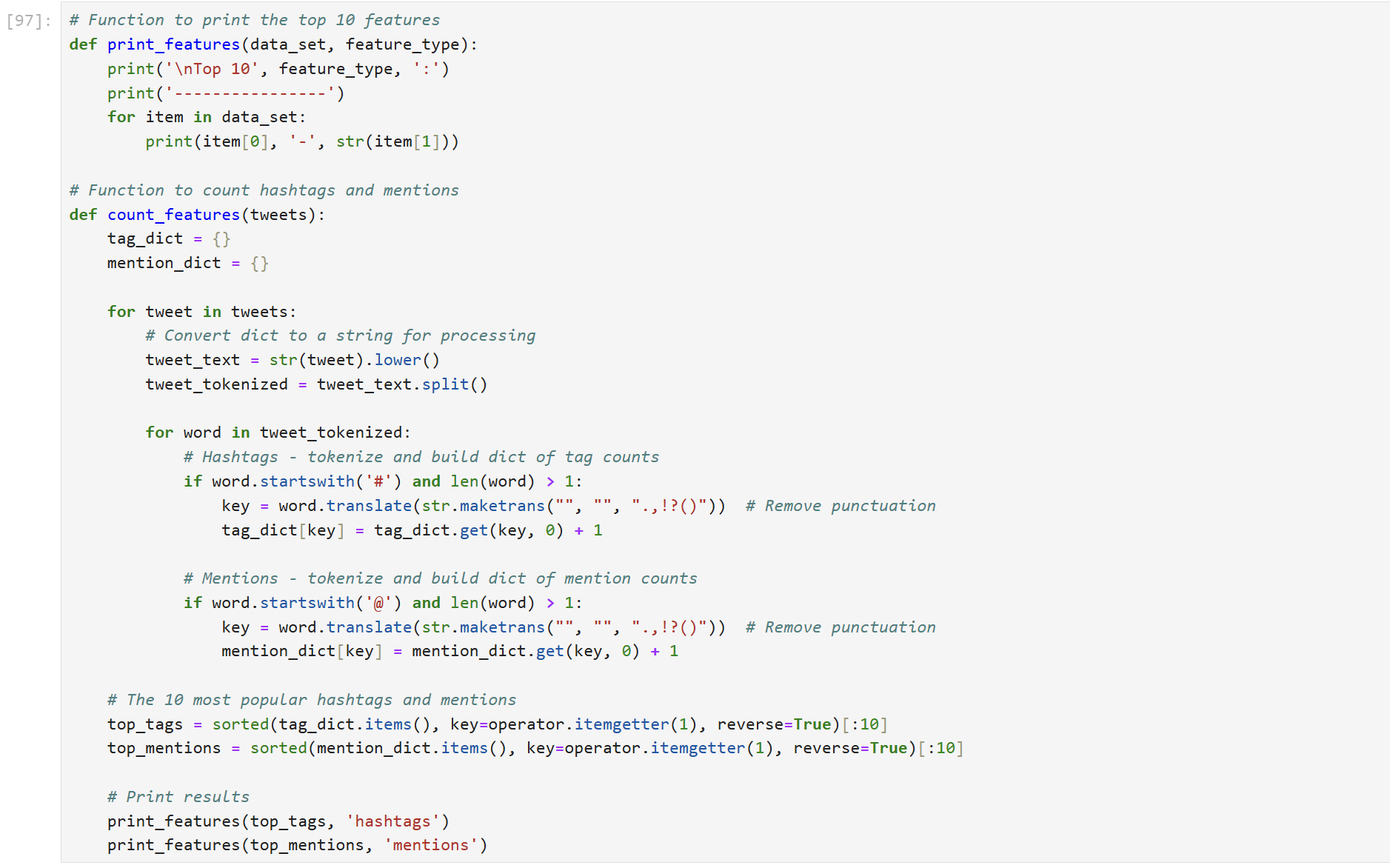
top\_tags\_sorted = sorted(top\_tags.items(), key=**lambda** x: x[1])[::-1] print\_features(top\_tags\_sorted, 'hashtags')

*# The 10 most popular mentions and counts*

top\_mentions = dict(sorted(mention\_dict.items(), key=operator.itemgetter(1),␣

𝗌reverse=**True**)[:10])

top\_mentions\_sorted = sorted(top\_mentions.items(), key=**lambda** x: x[1])[::-1] print\_features(top\_mentions\_sorted, 'mentions')



[ ]:

*# Find the top 10 hastags and mentions for tweets that include sun # Use the new collection: newcol*

tweet\_df = newcol.find({'text':{'$regex':'sun'}}, {'text':1}) count\_features(tweet\_df)

