**SPARK SOURCE CODE:**

**CODE 1:**

import sys

from pyspark import SparkConf, SparkContext

from csv import reader

conf = SparkConf().setAppName("task5")

sc = SparkContext(conf=conf)

line1 = sc.textFile(sys.argv[1], 1)

line1 = line1.mapPartitions(lambda x: reader(x))

id = line1.map(lambda x: ((x[14],x[16]),1)).reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], False)

topviolator = sc.parallelize(id.take(1)).map(lambda x: (x[0][0], x[0][1], x[1]))

topviolator.map(lambda (k, v, l): "{0}, {1}\t{2}".format(k, v, l)).saveAsTextFile("task5.out")

**CODE 2:**

import sys

from pyspark import SparkConf, SparkContext

from csv import reader

conf = SparkConf().setAppName("task4")

sc = SparkContext(conf=conf)

line1 = sc.textFile(sys.argv[1], 1)

line1 = line1.mapPartitions(lambda x: reader(x))

state = line1.map(lambda x: (("NY" if x[16]=="NY" else "Other" ),1)).reduceByKey(lambda x, y: x + y)

state.map(lambda (k, v): "{0}\t{1}".format(k, v)).saveAsTextFile("task4.out")

**CODE 3:**

import sys

from pyspark import SparkConf, SparkContext

from csv import reader

conf = SparkConf().setAppName("task1")

sc = SparkContext(conf=conf)

line1 = sc.textFile(sys.argv[1], 1)

line1 = line1.mapPartitions(lambda x: reader(x))

line2 = sc.textFile(sys.argv[2], 1)

line2 = line2.mapPartitions(lambda x: reader(x))

allviolation = line1.map(lambda x: (x[0],(x[14],x[6],x[2],x[1])))

openviolation = line2.map(lambda x: (x[0],1))

result = allviolation.subtractByKey(openviolation)

result = result.sortByKey().map(lambda x: (x[0],x[1][0],x[1][1],x[1][2], x[1][3]))

result.map(lambda (a,b,c,d,e): "{0}\t{1}, {2}, {3}, {4}".format(a,b,c,d,e)).saveAsTextFile("task1.out")

**CODE 4:**

import sys

from decimal import Decimal

from pyspark import SparkConf, SparkContext

from csv import reader

conf = SparkConf().setAppName("task7")

sc = SparkContext(conf=conf)

line1 = sc.textFile(sys.argv[1], 1)

line1 = line1.mapPartitions(lambda x: reader(x))

violationweekday = line1.map(lambda x: (x[2],(0 if int(x[1][-2:]) in (5,6,12,13,19,20,26,27) else 1))).reduceByKey(lambda x, y: x + y).map(lambda x: (x[0], Decimal(Decimal(x[1])/23).quantize(Decimal('.01'))))

violationweekend = line1.map(lambda x: (x[2],(1 if int(x[1][-2:]) in (5,6,12,13,19,20,26,27) else 0))).reduceByKey(lambda x, y: x + y).map(lambda x: (x[0], Decimal(Decimal(x[1])/8).quantize(Decimal('.01'))))

violationweekend.fullOuterJoin(violationweekday).map(lambda x: (x[0],x[1][0],x[1][1])).map(lambda (k, v, l): "{0}\t{1}, {2}".format(k, v, l)).saveAsTextFile("task7.out")

**CODE 5:**

import sys

from pyspark import SparkConf, SparkContext

from csv import reader

conf = SparkConf().setAppName("task6")

sc = SparkContext(conf=conf)

line1 = sc.textFile(sys.argv[1], 1)

line1 = line1.mapPartitions(lambda x: reader(x))

id = line1.map(lambda x: ((x[14],x[16]),1)).reduceByKey(lambda x, y: x + y).sortBy(lambda x: x[1], False)

top20violator = sc.parallelize(id.take(20)).map(lambda x: (x[0][0], x[0][1], x[1]))

top20violator.map(lambda (k, v, l): "{0}, {1}\t{2}".format(k, v, l)).saveAsTextFile("task6.out")

**BIGQUERY CODES:**

**SAMPLE BIGQUERY JSON FILE:**

|  |
| --- |
| [{ |
|  | "name": "document\_id", |
|  | "type": "string", |
|  | "description": "Unique key when combined with Good\_through\_date", |
|  | "mode": "required" |
|  | }, { |
|  | "name": "record\_type", |
|  | "type": "string", |
|  | "description": "Always M (Master)", |
|  | "mode": "required" |
|  | }, { |
|  | "name": "crfn", |
|  | "type": "string", |
|  | "description": "City Register File Number, begins 2003", |
|  | "mode": "nullable" |
|  | }, { |
|  | "name": "borough", |
|  | "type": "integer", |
|  | "description": "Borough that this document was recorded in.", |
|  | "mode": "required" |
|  | }, { |
|  | "name": "doc\_type", |
|  | "type": "string", |
|  | "description": "Document type cross-referenced with acris.code\_document\_control", |
|  | "mode": "required" |
|  | }, { |
|  | "name": "doc\_date", |
|  | "type": "timestamp", |
|  | "description": "The date when the transaction described by this document actually occurred; recorded regularly 2003 onwards.", |
|  | "mode": "nullable" |
|  | }, { |
|  | "name": "doc\_amount", |
|  | "type": "float", |
|  | "description": "The recorded amount of the document. Recorded for mortgages from 1982 onwards, and for deeds from 2003 onwards.", |
|  | "mode": "nullable" |
|  | }, { |
|  | "name": "recorded\_filed", |
|  | "type": "timestamp", |
|  | "description": "The date when this document was recorded and filed; may be considerably later than Document\_Date, but is more consistently recorded.", |
|  | "mode": "required" |
|  | }, { |
|  | "name": "modified\_date", |
|  | "type": "timestamp", |
|  | "mode": "nullable" |
|  | }, { |
|  | "name": "reel\_year", |
|  | "type": "integer", |
|  | "description": "Part of Reel ID system, for documents without CRFN.", |
|  | "mode": "nullable" |
|  | }, { |
|  | "name": "reel\_nbr", |
|  | "type": "integer", |
|  | "description": "Part of Reel ID system, for documents without CRFN.", |
|  | "mode": "nullable" |
|  | }, { |
|  | "name": "reel\_pg", |
|  | "type": "integer", |
|  | "description": "Part of Reel ID system, for documents without CRFN.", |
|  | "mode": "nullable" |
|  | }, { |
|  | "name": "percent\_transferred", |
|  | "type": "float", |
|  | "mode": "nullable" |
|  | }, { |
|  | "name": "good\_through\_date", |
|  | "type": "timestamp", |
|  | "description": "Unique key when combined with Document\_id", |
|  | "mode": "required" |
|  | }] |

**SAMPLE BIGQUERY QUERY:**

bq query --format csv "SELECT \* FROM [personal-real-estate:acris.real\_flat] where street\_name like 'VAN BUREN%' and street\_number='60' order by doc\_date desc"

**PYTHON FILE TO CHECK FOR MORE RECENT UPDATES TO UPDATE ON THE EXISTING DATASET:**

**import sys**

**import json**

**def last\_modified(data\_json):**

**is\_dcc\_identifier = lambda d: d['identifier'] == '7isb-wh4c'**

**print(next((d for d in data\_json if is\_dcc\_identifier(d)))['modified'])**

**if \_\_name\_\_ == '\_\_main\_\_':**

**doc = """**

**Call as follows:**

**python last\_modified.py <path/to/data.json>**

**Will return the text string of form YYYY-MM-DD of when ACRIS data was last**

**modified.**

**"""**

**if len(sys.argv) == 2:**

**last\_modified(json.load(open(sys.argv[1], 'r')))**

**else:**

**print(doc)**

**sys.exit(1)**

**RAW SHELL SCRIPT TO DOWNLOAD AND SAVE IN CSV FORMAT:**

**//Below Source code framework is provided by NYC OpenData**

#!/bin/bash -e

source utils/colors.sh

mkdir -p tmp

LAST\_MODIFIED\_FILE=output/last\_modified

LAST\_MODIFIED=$(cat ${LAST\_MODIFIED\_FILE} 2>/dev/null || echo -n '')

if [[ -e output/real && -e output/personal && $LAST\_MODIFIED = '' ]]; then

error "Preexisting output with no ${LAST\_MODIFIED\_FILE}"

error "Please specify last modified date there, for example:"

info "echo 2014-07-08 > ${LAST\_MODIFIED\_FILE}"

echo

exit 1

fi

### Determine whether we need to download new files

DATAURL=https://nycopendata.socrata.com/data.json

DATAJSON=tmp/data.json

info "Determining whether data online is more recent than $LAST\_MODIFIED from $DATAURL..."

mkdir -p logs

wget $DATAURL -o logs/data.json.log -O $DATAJSON

NEW\_LAST\_MODIFIED=$(python last\_modified.py $DATAJSON)

if [[ $LAST\_MODIFIED = $NEW\_LAST\_MODIFIED ]]; then

success "Already have data current through $LAST\_MODIFIED, no need to download."

exit 0

fi

info "Downloading new data for $NEW\_LAST\_MODIFIED, backing up old data..."

### Set up output dirs

mkdir -p logs

mkdir -p output

if [[ -e output/real && -e output/personal ]]; then

mv output/real output/real\_$LAST\_MODIFIED

mv output/personal output/personal\_$LAST\_MODIFIED

fi

### Download datasets from Socrata

# Docs found at https://data.cityofnewyork.us/api/assets/C74AE896-1AF3-4A87-B67C-9D4D15D562D3?download=true

# Array of table names, this is the order of the IDs below

declare -a TABLES=(master legals parties references remarks)

# Array of IDs for Real Property tables on Socrata, in order of $TABLES

declare -a REAL=(bnx9-e6tj 8h5j-fqxa 636b-3b5g pwkr-dpni 9p4w-7npp)

# Array of IDs for Personal Property tables on Socrata, in order of $TABLES

declare -a PERSONAL=(sv7x-dduq uqqa-hym2 nbbg-wtuz 6y3e-jcrc fuzi-5ks9)

# $1: real/personal/code, $2: tablename, $3: tableid

function download {

mkdir -p logs/$1 && mkdir -p output/$1

if [ -e output/$1/$2.csv.gz ]; then

info "Already downloaded $1/$2, skipping..."

elif [ -e output/$1/$2.csv ]; then

info "Currently downloading $1/$2, skipping..."

else

info "Downloading $1/$2.csv ($3)"

# Download via wget.

#

# Use sed to filter '10/30/1974' style dates to '1974-10-30 00:00:00'

# via sed. Filtering allows import as timestamp.

#

# Use grep to:

#

# \* Exclude entries with the date `0200-02-29`, which happens in

# real\_master

#

# \* Allow only records that don't have a double quote in them -- these

# are also bad

#

# \* Allow only records that start with a number or uppercase

# letter through; a quoted beginning means corrupt data (this

# happens in personal\_references.)

wget -o logs/$1/$2.log -O - \

https://data.cityofnewyork.us/api/views/$3/rows.csv?accessType=DOWNLOAD \

| sed -r 's\_,([0-9]{2})/([0-9]{2})/([0-9]{4})\_,\3-\1-\2 00:00:00\_g' \

| grep -v '0200-02-29' \

| grep -v '""' \

| grep '^[0-9A-Z]' > output/$1/$2.csv &

fi

}

# iterate through all table types and download

for i in {0..4}; do

name=${TABLES[$i]}

download real $name ${REAL[$i]}

download personal $name ${PERSONAL[$i]}

done

# download codes

download code document\_control 7isb-wh4c

download code ucc\_collateral q9kp-jvxv # This is switched with `country` on Socrata

download code property\_types 94g4-w6xz

download code states 5c9e-33xj

download code country j2iz-mwzu # This is switched with `ucc\_collateral` on Socrata

# Wait for downloads to complete

info "Waiting for downloads to complete."

wait

# TODO: mappluto data should be handled in a separate script

# ### Download MapPLUTO data

# mkdir -p logs/pluto output/pluto/shpsources

# for release in 02b 03c 04c 05d 06c 07c 09v1 09v2 10v1 10v2 11v1 11v2 12v1 12v2 13v1 13v2; do

# if [ -e output/pluto/shpsources/$release ]; then

# echo "Already downloaded MapPLUTO $release"

# else

# echo "Downloading MapPLUTO $release"

# wget -o logs/pluto/shpsources/$release.log -O output/pluto/$release.zip \

# "http://www.nyc.gov/html/dcp/download/bytes/mappluto\_$release.zip" &

# fi

# done

#

# # Wait for downloads to complete

# echo "Waiting for downloads to complete."

# wait

#

#

# ### Unzip MAPPluto data

# for archive in output/pluto/\*.zip; do

# base=$(basename $archive .zip)

# mkdir -p output/pluto/shpsources/$base

# if [ ! -e output/pluto/shpsources/$base ]; then

# unzip -d output/pluto/shpsources/$base $archive

# rm -f $archive

# fi

# done

#

# ### Convert MAPPluto data to CSV

# mkdir -p output/pluto/csvsources

# for f in $(ls output/pluto/shpsources/\*\*/MapPLUTO\*/\*/\*{PLUTO,pluto}.shp); do

# version=$(basename $(dirname $(dirname $f)))

# borough=$(basename $f .shp)

# outfile=output/pluto/csvsources/${version}\_${borough}.csv

# if [ -e $outfile ]; then

# echo "Skipping $f, $outfile exists already..."

# else

# echo "Converting $f to ${version}\_${borough}.csv via ogr2ogr..."

# ogr2ogr -f csv $outfile $f

# fi

# done

# wait

#

# # It's not possible to just merge all our CSVs together, since the schema

# # changes from PLUTO to PLUTO. Meh.

# # ### Merge all output CSVs to one big'un

# # # First, grab a random header row from output; then, put in all other rows

# # # excluding headers

# # allpluto=output/pluto/pluto.csv

# #

# # ls output/pluto/csvsources/\*.csv | head -n 1 | xargs head -n 1 > $allpluto

# # ls output/pluto/csvsources/\*.csv |

#

# # Instead we just merge together the latest (13v2) and upload that.

# pluto\_version=13v1

# echo "Merging together pluto $pluto\_version and zipping"

# pluto=output/pluto/pluto.csv

# ls output/pluto/csvsources/MapPLUTO\_${pluto\_version}\_\*.csv | head -n 1 | xargs head -n 1 > $pluto

# ls output/pluto/csvsources/MapPLUTO\_${pluto\_version}\_\*.csv | xargs tail -q -n +2 >> $pluto

# gzip -9 $pluto

echo $NEW\_LAST\_MODIFIED > output/last\_modified

info "Gzipping CSVs in output folder..."

gzip -9 output/real/\*.csv output/personal/\*.csv output/code/\*.csv || true

success "Done downloading"

**\*\*\*\*\*\*\*\*END OF DOCUMENT\*\*\*\*\*\*\*\***