#&nbspCopyright&nbsp2017&nbspGoogle&nbspInc.

#

#&nbspLicensed&nbspunder&nbspthe&nbspApache&nbspLicense,&nbspVersion&nbsp2.0 (the "License");

#&nbspyou&nbspmay&nbspnot&nbspuse&nbspthis&nbspfile&nbspexcept&nbspin&nbspcompliance&nbspwith&nbspthe&nbspLicense.

#&nbspYou&nbspmay&nbspobtain&nbspa&nbspcopy&nbspof&nbspthe&nbspLicense&nbspat

#

#     &nbsphttp://www.apache.org/licenses/LICENSE-2.0

#

#&nbspUnless&nbsprequired&nbspby&nbspapplicable&nbsplaw&nbspor&nbspagreed&nbspto&nbspin&nbspwriting,&nbspsoftware

#&nbspdistributed&nbspunder&nbspthe&nbspLicense&nbspis&nbspdistributed&nbspon&nbspan "AS&nbspIS"&nbspBASIS,

#&nbspWITHOUT&nbspWARRANTIES&nbspOR&nbspCONDITIONS&nbspOF&nbspANY&nbspKIND,&nbspeither&nbspexpress&nbspor&nbspimplied.

#&nbspSee&nbspthe&nbspLicense&nbspfor&nbspthe&nbspspecific&nbsplanguage&nbspgoverning&nbsppermissions&nbspand

#&nbsplimitations&nbspunder&nbspthe&nbspLicense.

"""`data\_ingestion.py`&nbspis&nbspa&nbspDataflow&nbsppipeline&nbspwhich&nbspreads&nbspa&nbspfile&nbspand&nbspwrites&nbspits

contents&nbspto&nbspa&nbspBigQuery&nbsptable.

This&nbspexample&nbspdoes&nbspnot&nbspdo&nbspany&nbsptransformation&nbspon&nbspthe&nbspdata.

"""

import&nbspargparse

import&nbsplogging

import&nbspre

import&nbspapache\_beam as&nbspbeam

from&nbspapache\_beam.options.pipeline\_options import&nbspPipelineOptions

class DataIngestion:

    """A&nbsphelper&nbspclass&nbspwhich&nbspcontains&nbspthe&nbsplogic&nbspto&nbsptranslate&nbspthe&nbspfile&nbspinto

   &nbspa&nbspformat&nbspBigQuery&nbspwill&nbspaccept."""

    def parse\_method(self, string\_input):

        """This&nbspmethod&nbsptranslates&nbspa&nbspsingle&nbspline&nbspof&nbspcomma&nbspseparated&nbspvalues&nbspto&nbspa

       &nbspdictionary&nbspwhich&nbspcan&nbspbe&nbsploaded&nbspinto&nbspBigQuery.

       &nbspArgs:

           &nbspstring\_input:&nbspA&nbspcomma&nbspseparated&nbsplist&nbspof&nbspvalues&nbspin&nbspthe&nbspform&nbspof

               &nbspstate\_abbreviation,gender,year,name,count\_of\_babies,dataset\_created\_date

               &nbspExample&nbspstring\_input:&nbspKS,F,1923,Dorothy,654,11/28/2016

       &nbspReturns:

           &nbspA&nbspdict&nbspmapping&nbspBigQuery&nbspcolumn&nbspnames&nbspas&nbspkeys&nbspto&nbspthe&nbspcorresponding&nbspvalue

           &nbspparsed&nbspfrom&nbspstring\_input.&nbspIn&nbspthis&nbspexample,&nbspthe&nbspdata&nbspis&nbspnot&nbsptransformed,&nbspand

           &nbspremains&nbspin&nbspthe&nbspsame&nbspformat&nbspas&nbspthe&nbspCSV.

           &nbspexample&nbspoutput:

            {

                'state': 'KS',

                'gender': 'F',

                'year': '1923',

                'name': 'Dorothy',

                'number': '654',

                'created\_date': '11/28/2016'

            }

         """

        #&nbspStrip&nbspout&nbspcarriage&nbspreturn,&nbspnewline&nbspand&nbspquote&nbspcharacters.

       &nbspvalues =&nbspre.split(",",

                         &nbspre.sub('\r\n', '',&nbspre.sub('"', '',&nbspstring\_input)))

       &nbsprow = dict(

            zip(('state', 'gender', 'year', 'name', 'number', 'created\_date'),

               &nbspvalues))

        return&nbsprow

def run(argv=None):

    """The&nbspmain&nbspfunction&nbspwhich&nbspcreates&nbspthe&nbsppipeline&nbspand&nbspruns&nbspit."""

   &nbspparser =&nbspargparse.ArgumentParser()

    #&nbspHere&nbspwe&nbspadd&nbspsome&nbspspecific&nbspcommand&nbspline&nbsparguments&nbspwe&nbspexpect.

    #&nbspSpecifically&nbspwe&nbsphave&nbspthe&nbspinput&nbspfile&nbspto&nbspread&nbspand&nbspthe&nbspoutput&nbsptable&nbspto&nbspwrite.

    #&nbspThis&nbspis&nbspthe&nbspfinal&nbspstage&nbspof&nbspthe&nbsppipeline,&nbspwhere&nbspwe&nbspdefine&nbspthe&nbspdestination

    #&nbspof&nbspthe&nbspdata.&nbspIn&nbspthis&nbspcase&nbspwe&nbspare&nbspwriting&nbspto&nbspBigQuery.

   &nbspparser.add\_argument(

        '--input',

        dest='input',

        required=False,

        help='Input&nbspfile&nbspto&nbspread.&nbspThis&nbspcan&nbspbe&nbspa&nbsplocal&nbspfile&nbspor '

        'a&nbspfile&nbspin&nbspa&nbspGoogle&nbspStorage&nbspBucket.',

        #&nbspThis&nbspexample&nbspfile&nbspcontains&nbspa&nbsptotal&nbspof&nbsponly&nbsp10&nbsplines.

        #&nbspUseful&nbspfor&nbspdeveloping&nbspon&nbspa&nbspsmall&nbspset&nbspof&nbspdata.

        default='gs://spls/gsp290/data\_files/head\_usa\_names.csv')

    #&nbspThis&nbspdefaults&nbspto&nbspthe&nbsplake&nbspdataset&nbspin&nbspyour&nbspBigQuery&nbspproject.&nbspYou'll&nbsphave

    #&nbspto&nbspcreate&nbspthe&nbsplake&nbspdataset&nbspyourself&nbspusing&nbspthis&nbspcommand:

    #&nbspbq&nbspmk&nbsplake

   &nbspparser.add\_argument('--output',

                        dest='output',

                        required=False,

                        help='Output&nbspBQ&nbsptable&nbspto&nbspwrite&nbspresults&nbspto.',

                        default='lake.usa\_names')

    #&nbspParse&nbsparguments&nbspfrom&nbspthe&nbspcommand&nbspline.

   &nbspknown\_args,&nbsppipeline\_args =&nbspparser.parse\_known\_args(argv)

    #&nbspDataIngestion&nbspis&nbspa&nbspclass&nbspwe&nbspbuilt&nbspin&nbspthis&nbspscript&nbspto&nbsphold&nbspthe&nbsplogic&nbspfor

    #&nbsptransforming&nbspthe&nbspfile&nbspinto&nbspa&nbspBigQuery&nbsptable.

   &nbspdata\_ingestion =&nbspDataIngestion()

    #&nbspInitiate&nbspthe&nbsppipeline&nbspusing&nbspthe&nbsppipeline&nbsparguments&nbsppassed&nbspin&nbspfrom&nbspthe

    #&nbspcommand&nbspline.&nbspThis&nbspincludes&nbspinformation&nbspsuch&nbspas&nbspthe&nbspproject&nbspID&nbspand

    #&nbspwhere&nbspDataflow&nbspshould&nbspstore&nbsptemp&nbspfiles.

   &nbspp =&nbspbeam.Pipeline(options=PipelineOptions(pipeline\_args))

    (p

     #&nbspRead&nbspthe&nbspfile.&nbspThis&nbspis&nbspthe&nbspsource&nbspof&nbspthe&nbsppipeline.&nbspAll&nbspfurther

     #&nbspprocessing&nbspstarts&nbspwith&nbsplines&nbspread&nbspfrom&nbspthe&nbspfile.&nbspWe&nbspuse&nbspthe&nbspinput

     #&nbspargument&nbspfrom&nbspthe&nbspcommand&nbspline.&nbspWe&nbspalso&nbspskip&nbspthe&nbspfirst&nbspline&nbspwhich&nbspis&nbspa

     #&nbspheader&nbsprow.

     | 'Read&nbspfrom&nbspa&nbspFile' >>&nbspbeam.io.ReadFromText(known\_args.input,

                                                  skip\_header\_lines=1)

     #&nbspThis&nbspstage&nbspof&nbspthe&nbsppipeline&nbsptranslates&nbspfrom&nbspa&nbspCSV&nbspfile&nbspsingle&nbsprow

     #&nbspinput&nbspas&nbspa&nbspstring,&nbspto&nbspa&nbspdictionary&nbspobject&nbspconsumable&nbspby&nbspBigQuery.

     #&nbspIt&nbsprefers&nbspto&nbspa&nbspfunction&nbspwe&nbsphave&nbspwritten.&nbspThis&nbspfunction&nbspwill

     #&nbspbe&nbsprun&nbspin&nbspparallel&nbspon&nbspdifferent&nbspworkers&nbspusing&nbspinput&nbspfrom&nbspthe

     #&nbspprevious&nbspstage&nbspof&nbspthe&nbsppipeline.

     | 'String&nbspTo&nbspBigQuery&nbspRow' >>

    &nbspbeam.Map(lambda s:&nbspdata\_ingestion.parse\_method(s))

     | 'Write&nbspto&nbspBigQuery' >>&nbspbeam.io.Write(

        &nbspbeam.io.BigQuerySink(

             #&nbspThe&nbsptable&nbspname&nbspis&nbspa&nbsprequired&nbspargument&nbspfor&nbspthe&nbspBigQuery&nbspsink.

             #&nbspIn&nbspthis&nbspcase&nbspwe&nbspuse&nbspthe&nbspvalue&nbsppassed&nbspin&nbspfrom&nbspthe&nbspcommand&nbspline.

            &nbspknown\_args.output,

             #&nbspHere&nbspwe&nbspuse&nbspthe&nbspsimplest&nbspway&nbspof&nbspdefining&nbspa&nbspschema:

             #&nbspfieldName:fieldType

             schema='state:STRING,gender:STRING,year:STRING,name:STRING,'

             'number:STRING,created\_date:STRING',

             #&nbspCreates&nbspthe&nbsptable&nbspin&nbspBigQuery&nbspif&nbspit&nbspdoes&nbspnot&nbspyet&nbspexist.

             create\_disposition=beam.io.BigQueryDisposition.CREATE\_IF\_NEEDED,

             #&nbspDeletes&nbspall&nbspdata&nbspin&nbspthe&nbspBigQuery&nbsptable&nbspbefore&nbspwriting.

             write\_disposition=beam.io.BigQueryDisposition.WRITE\_TRUNCATE)))

   &nbspp.run().wait\_until\_finish()

if \_\_name\_\_ == '\_\_main\_\_':

   &nbsplogging.getLogger().setLevel(logging.INFO)

   &nbsprun()

#&nbspCopyright&nbsp2017&nbspGoogle&nbspInc.

#

#&nbspLicensed&nbspunder&nbspthe&nbspApache&nbspLicense,&nbspVersion&nbsp2.0 (the "License");

#&nbspyou&nbspmay&nbspnot&nbspuse&nbspthis&nbspfile&nbspexcept&nbspin&nbspcompliance&nbspwith&nbspthe&nbspLicense.

#&nbspYou&nbspmay&nbspobtain&nbspa&nbspcopy&nbspof&nbspthe&nbspLicense&nbspat

#

#     &nbsphttp://www.apache.org/licenses/LICENSE-2.0

#

#&nbspUnless&nbsprequired&nbspby&nbspapplicable&nbsplaw&nbspor&nbspagreed&nbspto&nbspin&nbspwriting,&nbspsoftware

#&nbspdistributed&nbspunder&nbspthe&nbspLicense&nbspis&nbspdistributed&nbspon&nbspan "AS&nbspIS"&nbspBASIS,

#&nbspWITHOUT&nbspWARRANTIES&nbspOR&nbspCONDITIONS&nbspOF&nbspANY&nbspKIND,&nbspeither&nbspexpress&nbspor&nbspimplied.

#&nbspSee&nbspthe&nbspLicense&nbspfor&nbspthe&nbspspecific&nbsplanguage&nbspgoverning&nbsppermissions&nbspand

#&nbsplimitations&nbspunder&nbspthe&nbspLicense.

"""&nbspdata\_transformation.py&nbspis&nbspa&nbspDataflow&nbsppipeline&nbspwhich&nbspreads&nbspa&nbspfile&nbspand&nbspwrites

its&nbspcontents&nbspto&nbspa&nbspBigQuery&nbsptable.

This&nbspexample&nbspreads&nbspa&nbspjson&nbspschema&nbspof&nbspthe&nbspintended&nbspoutput&nbspinto&nbspBigQuery,

and&nbsptransforms&nbspthe&nbspdate&nbspdata&nbspto&nbspmatch&nbspthe&nbspformat&nbspBigQuery&nbspexpects.

"""

import&nbspargparse

import&nbspcsv

import&nbsplogging

import&nbspos

import&nbspapache\_beam as&nbspbeam

from&nbspapache\_beam.options.pipeline\_options import&nbspPipelineOptions

from&nbspapache\_beam.io.gcp.bigquery\_tools import&nbspparse\_table\_schema\_from\_json

class DataTransformation:

    """A&nbsphelper&nbspclass&nbspwhich&nbspcontains&nbspthe&nbsplogic&nbspto&nbsptranslate&nbspthe&nbspfile&nbspinto&nbspa

 &nbspformat&nbspBigQuery&nbspwill&nbspaccept."""

    def \_\_init\_\_(self):

       &nbspdir\_path =&nbspos.path.dirname(os.path.realpath(\_\_file\_\_))

        self.schema\_str = ''

        #&nbspHere&nbspwe&nbspread&nbspthe&nbspoutput&nbspschema&nbspfrom&nbspa&nbspjson&nbspfile. &nbspThis&nbspis&nbspused&nbspto&nbspspecify&nbspthe&nbsptypes

        #&nbspof&nbspdata&nbspwe&nbspare&nbspwriting&nbspto&nbspBigQuery.

       &nbspschema\_file =&nbspos.path.join(dir\_path, 'resources', 'usa\_names\_year\_as\_date.json')

        with open(schema\_file) \

                as&nbspf:

           &nbspdata =&nbspf.read()

            #&nbspWrapping&nbspthe&nbspschema&nbspin&nbspfields&nbspis&nbsprequired&nbspfor&nbspthe&nbspBigQuery&nbspAPI.

            self.schema\_str = '{"fields": ' +&nbspdata + '}'

    def parse\_method(self, string\_input):

        """This&nbspmethod&nbsptranslates&nbspa&nbspsingle&nbspline&nbspof&nbspcomma&nbspseparated&nbspvalues&nbspto&nbspa

   &nbspdictionary&nbspwhich&nbspcan&nbspbe&nbsploaded&nbspinto&nbspBigQuery.

       &nbspArgs:

           &nbspstring\_input:&nbspA&nbspcomma&nbspseparated&nbsplist&nbspof&nbspvalues&nbspin&nbspthe&nbspform&nbspof

           &nbspstate\_abbreviation,gender,year,name,count\_of\_babies,dataset\_created\_date

               &nbspexample&nbspstring\_input:&nbspKS,F,1923,Dorothy,654,11/28/2016

       &nbspReturns:

           &nbspA&nbspdict&nbspmapping&nbspBigQuery&nbspcolumn&nbspnames&nbspas&nbspkeys&nbspto&nbspthe&nbspcorresponding&nbspvalue

           &nbspparsed&nbspfrom&nbspstring\_input. &nbspIn&nbspthis&nbspexample,&nbspthe&nbspdata&nbspis&nbspnot&nbsptransformed,&nbspand

           &nbspremains&nbspin&nbspthe&nbspsame&nbspformat&nbspas&nbspthe&nbspCSV. &nbspThere&nbspare&nbspno&nbspdate&nbspformat&nbsptransformations.

               &nbspexample&nbspoutput:

                      {'state': 'KS',

                       'gender': 'F',

                       'year': '1923-01-01', <-&nbspThis&nbspis&nbspthe&nbspBigQuery&nbspdate&nbspformat.

                       'name': 'Dorothy',

                       'number': '654',

                       'created\_date': '11/28/2016'

                       }

        """

        #&nbspStrip&nbspout&nbspreturn&nbspcharacters&nbspand&nbspquote&nbspcharacters.

       &nbspschema =&nbspparse\_table\_schema\_from\_json(self.schema\_str)

       &nbspfield\_map = [f for&nbspf in&nbspschema.fields]

        #&nbspUse&nbspa&nbspCSV&nbspReader&nbspwhich&nbspcan&nbsphandle&nbspquoted&nbspstrings&nbspetc.

       &nbspreader =&nbspcsv.reader(string\_input.split('\n'))

        for&nbspcsv\_row in&nbspreader:

            #&nbspOur&nbspsource&nbspdata&nbsponly&nbspcontains&nbspyear,&nbspso&nbspdefault&nbspJanuary&nbsp1st&nbspas&nbspthe

            #&nbspmonth&nbspand&nbspday.

           &nbspmonth = '01'

           &nbspday = '01'

            #&nbspThe&nbspyear&nbspcomes&nbspfrom&nbspour&nbspsource&nbspdata.

           &nbspyear =&nbspcsv\_row[2]

           &nbsprow = {}

           &nbspi = 0

            #&nbspIterate&nbspover&nbspthe&nbspvalues&nbspfrom&nbspour&nbspcsv&nbspfile,&nbspapplying&nbspany&nbsptransformation&nbsplogic.

            for&nbspvalue in&nbspcsv\_row:

                #&nbspIf&nbspthe&nbspschema&nbspindicates&nbspthis&nbspfield&nbspis&nbspa&nbspdate&nbspformat,&nbspwe&nbspmust

                #&nbsptransform&nbspthe&nbspdate&nbspfrom&nbspthe&nbspsource&nbspdata&nbspinto&nbspa&nbspformat&nbspthat

                #&nbspBigQuery&nbspcan&nbspunderstand.

                if&nbspfield\_map[i].type == 'DATE':

                    #&nbspFormat&nbspthe&nbspdate&nbspto&nbspYYYY-MM-DD&nbspformat&nbspwhich&nbspBigQuery

                    #&nbspaccepts.

                   &nbspvalue = '-'.join((year,&nbspmonth,&nbspday))

               &nbsprow[field\_map[i].name] =&nbspvalue

               &nbspi += 1

            return&nbsprow

def run(argv=None):

    """The&nbspmain&nbspfunction&nbspwhich&nbspcreates&nbspthe&nbsppipeline&nbspand&nbspruns&nbspit."""

   &nbspparser =&nbspargparse.ArgumentParser()

    #&nbspHere&nbspwe&nbspadd&nbspsome&nbspspecific&nbspcommand&nbspline&nbsparguments&nbspwe&nbspexpect.  &nbspSpecifically

    #&nbspwe&nbsphave&nbspthe&nbspinput&nbspfile&nbspto&nbspload&nbspand&nbspthe&nbspoutput&nbsptable&nbspto&nbspwrite&nbspto.

   &nbspparser.add\_argument(

        '--input', dest='input', required=False,

        help='Input&nbspfile&nbspto&nbspread. &nbspThis&nbspcan&nbspbe&nbspa&nbsplocal&nbspfile&nbspor '

             'a&nbspfile&nbspin&nbspa&nbspGoogle&nbspStorage&nbspBucket.',

        #&nbspThis&nbspexample&nbspfile&nbspcontains&nbspa&nbsptotal&nbspof&nbsponly&nbsp10&nbsplines.

        #&nbspIt&nbspis&nbspuseful&nbspfor&nbspdeveloping&nbspon&nbspa&nbspsmall&nbspset&nbspof&nbspdata

        default='gs://spls/gsp290/data\_files/head\_usa\_names.csv')

    #&nbspThis&nbspdefaults&nbspto&nbspthe&nbsptemp&nbspdataset&nbspin&nbspyour&nbspBigQuery&nbspproject. &nbspYou'll&nbsphave

    #&nbspto&nbspcreate&nbspthe&nbsptemp&nbspdataset&nbspyourself&nbspusing&nbspbq&nbspmk&nbsptemp

   &nbspparser.add\_argument('--output', dest='output', required=False,

                        help='Output&nbspBQ&nbsptable&nbspto&nbspwrite&nbspresults&nbspto.',

                        default='lake.usa\_names\_transformed')

    #&nbspParse&nbsparguments&nbspfrom&nbspthe&nbspcommand&nbspline.

   &nbspknown\_args,&nbsppipeline\_args =&nbspparser.parse\_known\_args(argv)

    #&nbspDataTransformation&nbspis&nbspa&nbspclass&nbspwe&nbspbuilt&nbspin&nbspthis&nbspscript&nbspto&nbsphold&nbspthe&nbsplogic&nbspfor

    #&nbsptransforming&nbspthe&nbspfile&nbspinto&nbspa&nbspBigQuery&nbsptable.

   &nbspdata\_ingestion =&nbspDataTransformation()

    #&nbspInitiate&nbspthe&nbsppipeline&nbspusing&nbspthe&nbsppipeline&nbsparguments&nbsppassed&nbspin&nbspfrom&nbspthe

    #&nbspcommand&nbspline. &nbspThis&nbspincludes&nbspinformation&nbsplike&nbspwhere&nbspDataflow&nbspshould

    #&nbspstore&nbsptemp&nbspfiles,&nbspand&nbspwhat&nbspthe&nbspproject&nbspid&nbspis.

   &nbspp =&nbspbeam.Pipeline(options=PipelineOptions(pipeline\_args))

   &nbspschema =&nbspparse\_table\_schema\_from\_json(data\_ingestion.schema\_str)

    (p

     #&nbspRead&nbspthe&nbspfile. &nbspThis&nbspis&nbspthe&nbspsource&nbspof&nbspthe&nbsppipeline. &nbspAll&nbspfurther

     #&nbspprocessing&nbspstarts&nbspwith&nbsplines&nbspread&nbspfrom&nbspthe&nbspfile. &nbspWe&nbspuse&nbspthe&nbspinput

     #&nbspargument&nbspfrom&nbspthe&nbspcommand&nbspline. &nbspWe&nbspalso&nbspskip&nbspthe&nbspfirst&nbspline&nbspwhich&nbspis&nbspa

     #&nbspheader&nbsprow.

     | 'Read&nbspFrom&nbspText' >>&nbspbeam.io.ReadFromText(known\_args.input,

                                                skip\_header\_lines=1)

     #&nbspThis&nbspstage&nbspof&nbspthe&nbsppipeline&nbsptranslates&nbspfrom&nbspa&nbspCSV&nbspfile&nbspsingle&nbsprow

     #&nbspinput&nbspas&nbspa&nbspstring,&nbspto&nbspa&nbspdictionary&nbspobject&nbspconsumable&nbspby&nbspBigQuery.

     #&nbspIt&nbsprefers&nbspto&nbspa&nbspfunction&nbspwe&nbsphave&nbspwritten. &nbspThis&nbspfunction&nbspwill

     #&nbspbe&nbsprun&nbspin&nbspparallel&nbspon&nbspdifferent&nbspworkers&nbspusing&nbspinput&nbspfrom&nbspthe

     #&nbspprevious&nbspstage&nbspof&nbspthe&nbsppipeline.

     | 'String&nbspto&nbspBigQuery&nbspRow' >>&nbspbeam.Map(lambda s:

                                           &nbspdata\_ingestion.parse\_method(s))

     | 'Write&nbspto&nbspBigQuery' >>&nbspbeam.io.Write(

       &nbspbeam.io.BigQuerySink(

            #&nbspThe&nbsptable&nbspname&nbspis&nbspa&nbsprequired&nbspargument&nbspfor&nbspthe&nbspBigQuery&nbspsink.

            #&nbspIn&nbspthis&nbspcase&nbspwe&nbspuse&nbspthe&nbspvalue&nbsppassed&nbspin&nbspfrom&nbspthe&nbspcommand&nbspline.

           &nbspknown\_args.output,

            #&nbspHere&nbspwe&nbspuse&nbspthe&nbspJSON&nbspschema&nbspread&nbspin&nbspfrom&nbspa&nbspJSON&nbspfile.

            #&nbspSpecifying&nbspthe&nbspschema&nbspallows&nbspthe&nbspAPI&nbspto&nbspcreate&nbspthe&nbsptable&nbspcorrectly&nbspif&nbspit&nbspdoes&nbspnot&nbspyet&nbspexist.

            schema=schema,

            #&nbspCreates&nbspthe&nbsptable&nbspin&nbspBigQuery&nbspif&nbspit&nbspdoes&nbspnot&nbspyet&nbspexist.

            create\_disposition=beam.io.BigQueryDisposition.CREATE\_IF\_NEEDED,

            #&nbspDeletes&nbspall&nbspdata&nbspin&nbspthe&nbspBigQuery&nbsptable&nbspbefore&nbspwriting.

            write\_disposition=beam.io.BigQueryDisposition.WRITE\_TRUNCATE)))

   &nbspp.run().wait\_until\_finish()

if \_\_name\_\_ == '\_\_main\_\_':

   &nbsplogging.getLogger().setLevel(logging.INFO)

   &nbsprun()

#&nbspCopyright&nbsp2017&nbspGoogle&nbspInc.

#

#&nbspLicensed&nbspunder&nbspthe&nbspApache&nbspLicense,&nbspVersion&nbsp2.0 (the "License");

#&nbspyou&nbspmay&nbspnot&nbspuse&nbspthis&nbspfile&nbspexcept&nbspin&nbspcompliance&nbspwith&nbspthe&nbspLicense.

#&nbspYou&nbspmay&nbspobtain&nbspa&nbspcopy&nbspof&nbspthe&nbspLicense&nbspat

#

#     &nbsphttp://www.apache.org/licenses/LICENSE-2.0

#

#&nbspUnless&nbsprequired&nbspby&nbspapplicable&nbsplaw&nbspor&nbspagreed&nbspto&nbspin&nbspwriting,&nbspsoftware

#&nbspdistributed&nbspunder&nbspthe&nbspLicense&nbspis&nbspdistributed&nbspon&nbspan "AS&nbspIS"&nbspBASIS,

#&nbspWITHOUT&nbspWARRANTIES&nbspOR&nbspCONDITIONS&nbspOF&nbspANY&nbspKIND,&nbspeither&nbspexpress&nbspor&nbspimplied.

#&nbspSee&nbspthe&nbspLicense&nbspfor&nbspthe&nbspspecific&nbsplanguage&nbspgoverning&nbsppermissions&nbspand

#&nbsplimitations&nbspunder&nbspthe&nbspLicense.

"""&nbspdata\_enrichment.py&nbspdemonstrates&nbspa&nbspDataflow&nbsppipeline&nbspwhich&nbspreads&nbspa&nbspfile&nbspand

&nbspwrites&nbspits&nbspcontents&nbspto&nbspa&nbspBigQuery&nbsptable. &nbspAlong&nbspthe&nbspway,&nbspdata&nbspfrom&nbspBigQuery

&nbspis&nbspread&nbspin&nbspas&nbspa&nbspside&nbspinput&nbspand&nbspjoined&nbspin&nbspwith&nbspthe&nbspprimary&nbspdata&nbspfrom&nbspthe&nbspfile.

"""

import&nbspargparse

import&nbspcsv

import&nbsplogging

import&nbspos

import&nbspsys

import&nbspapache\_beam as&nbspbeam

from&nbspapache\_beam.io.gcp import&nbspbigquery

from&nbspapache\_beam.io.gcp.bigquery import&nbspparse\_table\_schema\_from\_json

from&nbspapache\_beam.options.pipeline\_options import&nbspPipelineOptions

from&nbspapache\_beam.pvalue import&nbspAsDict

class DataIngestion(object):

    """A&nbsphelper&nbspclass&nbspwhich&nbspcontains&nbspthe&nbsplogic&nbspto&nbsptranslate&nbspthe&nbspfile&nbspinto&nbspa

 &nbspformat&nbspBigQuery&nbspwill&nbspaccept."""

    def \_\_init\_\_(self):

       &nbspdir\_path =&nbspos.path.dirname(os.path.realpath(\_\_file\_\_))

        self.schema\_str = ''

        #&nbspThis&nbspis&nbspthe&nbspschema&nbspof&nbspthe&nbspdestination&nbsptable&nbspin&nbspBigQuery.

       &nbspschema\_file =&nbspos.path.join(dir\_path, 'resources', 'usa\_names\_with\_full\_state\_name.json')

        with open(schema\_file) \

                as&nbspf:

           &nbspdata =&nbspf.read()

            #&nbspWrapping&nbspthe&nbspschema&nbspin&nbspfields&nbspis&nbsprequired&nbspfor&nbspthe&nbspBigQuery&nbspAPI.

            self.schema\_str = '{"fields": ' +&nbspdata + '}'

    def parse\_method(self, string\_input):

        """This&nbspmethod&nbsptranslates&nbspa&nbspsingle&nbspline&nbspof&nbspcomma&nbspseparated&nbspvalues&nbspto&nbspa

   &nbspdictionary&nbspwhich&nbspcan&nbspbe&nbsploaded&nbspinto&nbspBigQuery.

       &nbspArgs:

           &nbspstring\_input:&nbspA&nbspcomma&nbspseparated&nbsplist&nbspof&nbspvalues&nbspin&nbspthe&nbspform&nbspof

           &nbspstate\_abbreviation,gender,year,name,count\_of\_babies,dataset\_created\_date

               &nbspexample&nbspstring\_input:&nbspKS,F,1923,Dorothy,654,11/28/2016

       &nbspReturns:

           &nbspA&nbspdict&nbspmapping&nbspBigQuery&nbspcolumn&nbspnames&nbspas&nbspkeys&nbspto&nbspthe&nbspcorresponding&nbspvalue

           &nbspparsed&nbspfrom&nbspstring\_input. &nbspIn&nbspthis&nbspexample,&nbspthe&nbspdata&nbspis&nbspnot&nbsptransformed,&nbspand

           &nbspremains&nbspin&nbspthe&nbspsame&nbspformat&nbspas&nbspthe&nbspCSV. &nbspThere&nbspare&nbspno&nbspdate&nbspformat&nbsptransformations.

               &nbspexample&nbspoutput:

                      {'state': 'KS',

                       'gender': 'F',

                       'year': '1923-01-01', <-&nbspThis&nbspis&nbspthe&nbspBigQuery&nbspdate&nbspformat.

                       'name': 'Dorothy',

                       'number': '654',

                       'created\_date': '11/28/2016'

                       }

     """

        #&nbspStrip&nbspout&nbspreturn&nbspcharacters&nbspand&nbspquote&nbspcharacters.

       &nbspschema =&nbspbigquery.parse\_table\_schema\_from\_json(self.schema\_str)

       &nbspfield\_map = [f for&nbspf in&nbspschema.fields]

        #&nbspUse&nbspa&nbspCSV&nbspReader&nbspwhich&nbspcan&nbsphandle&nbspquoted&nbspstrings&nbspetc.

       &nbspreader =&nbspcsv.reader(string\_input.split('\n'))

        for&nbspcsv\_row in&nbspreader:

            if (sys.version\_info.major < 3.0):

               &nbspvalues = [x.decode('utf8') for&nbspx in&nbspcsv\_row]

            else:

               &nbspvalues =&nbspcsv\_row

            #&nbspOur&nbspsource&nbspdata&nbsponly&nbspcontains&nbspyear,&nbspso&nbspdefault&nbspJanuary&nbsp1st&nbspas&nbspthe

            #&nbspmonth&nbspand&nbspday.

           &nbspmonth = '01'

           &nbspday = '01'

            #&nbspThe&nbspyear&nbspcomes&nbspfrom&nbspour&nbspsource&nbspdata.

           &nbspyear =&nbspvalues[2]

           &nbsprow = {}

           &nbspi = 0

            #&nbspIterate&nbspover&nbspthe&nbspvalues&nbspfrom&nbspour&nbspcsv&nbspfile,&nbspapplying&nbspany&nbsptransformation&nbsplogic.

            for&nbspvalue in&nbspvalues:

                #&nbspIf&nbspthe&nbspschema&nbspindicates&nbspthis&nbspfield&nbspis&nbspa&nbspdate&nbspformat,&nbspwe&nbspmust

                #&nbsptransform&nbspthe&nbspdate&nbspfrom&nbspthe&nbspsource&nbspdata&nbspinto&nbspa&nbspformat&nbspthat

                #&nbspBigQuery&nbspcan&nbspunderstand.

                if&nbspfield\_map[i].type == 'DATE':

                    #&nbspFormat&nbspthe&nbspdate&nbspto&nbspYYYY-MM-DD&nbspformat&nbspwhich&nbspBigQuery

                    #&nbspaccepts.

                   &nbspvalue = '-'.join((year,&nbspmonth,&nbspday))

               &nbsprow[field\_map[i].name] =&nbspvalue

               &nbspi += 1

            return&nbsprow

def run(argv=None):

    """The&nbspmain&nbspfunction&nbspwhich&nbspcreates&nbspthe&nbsppipeline&nbspand&nbspruns&nbspit."""

   &nbspparser =&nbspargparse.ArgumentParser()

    #&nbspHere&nbspwe&nbspadd&nbspsome&nbspspecific&nbspcommand&nbspline&nbsparguments&nbspwe&nbspexpect.  &nbspSpecifically

    #&nbspwe&nbsphave&nbspthe&nbspinput&nbspfile&nbspto&nbspload&nbspand&nbspthe&nbspoutput&nbsptable&nbspto&nbspwrite&nbspto.

   &nbspparser.add\_argument(

        '--input', dest='input', required=False,

        help='Input&nbspfile&nbspto&nbspread. &nbspThis&nbspcan&nbspbe&nbspa&nbsplocal&nbspfile&nbspor '

             'a&nbspfile&nbspin&nbspa&nbspGoogle&nbspStorage&nbspBucket.',

        #&nbspThis&nbspexample&nbspfile&nbspcontains&nbspa&nbsptotal&nbspof&nbsponly&nbsp10&nbsplines.

        #&nbspUseful&nbspfor&nbspquickly&nbspdebugging&nbspon&nbspa&nbspsmall&nbspset&nbspof&nbspdata

        default='gs://spls/gsp290/data\_files/head\_usa\_names.csv')

    #&nbspThe&nbspoutput&nbspdefaults&nbspto&nbspthe&nbsplake&nbspdataset&nbspin&nbspyour&nbspBigQuery&nbspproject. &nbspYou'll&nbsphave

    #&nbspto&nbspcreate&nbspthe&nbsplake&nbspdataset&nbspyourself&nbspusing&nbspthis&nbspcommand:

    #&nbspbq&nbspmk&nbsplake

   &nbspparser.add\_argument('--output', dest='output', required=False,

                        help='Output&nbspBQ&nbsptable&nbspto&nbspwrite&nbspresults&nbspto.',

                        default='lake.usa\_names\_enriched')

    #&nbspParse&nbsparguments&nbspfrom&nbspthe&nbspcommand&nbspline.

   &nbspknown\_args,&nbsppipeline\_args =&nbspparser.parse\_known\_args(argv)

    #&nbspDataIngestion&nbspis&nbspa&nbspclass&nbspwe&nbspbuilt&nbspin&nbspthis&nbspscript&nbspto&nbsphold&nbspthe&nbsplogic&nbspfor

    #&nbsptransforming&nbspthe&nbspfile&nbspinto&nbspa&nbspBigQuery&nbsptable.

   &nbspdata\_ingestion =&nbspDataIngestion()

    #&nbspInitiate&nbspthe&nbsppipeline&nbspusing&nbspthe&nbsppipeline&nbsparguments&nbsppassed&nbspin&nbspfrom&nbspthe

    #&nbspcommand&nbspline. &nbspThis&nbspincludes&nbspinformation&nbsplike&nbspwhere&nbspDataflow&nbspshould&nbspstore

    # &nbsptemp&nbspfiles,&nbspand&nbspwhat&nbspthe&nbspproject&nbspid&nbspis

   &nbspp =&nbspbeam.Pipeline(options=PipelineOptions(pipeline\_args))

   &nbspschema =&nbspparse\_table\_schema\_from\_json(data\_ingestion.schema\_str)

    #&nbspThis&nbspfunction&nbspadds&nbspin&nbspa&nbspfull&nbspstate&nbspname&nbspby&nbsplooking&nbspup&nbspthe

    #&nbspfull&nbspname&nbspin&nbspthe&nbspshort\_to\_long\_name\_map. &nbspThe&nbspshort\_to\_long\_name\_map

    #&nbspcomes&nbspfrom&nbspa&nbspread&nbspfrom&nbspBigQuery&nbspin&nbspthe&nbspnext&nbspfew&nbsplines

    def add\_full\_state\_name(row, short\_to\_long\_name\_map):

       &nbsprow['state\_full\_name'] =&nbspshort\_to\_long\_name\_map[row['state']]

        return&nbsprow

    #&nbspThis&nbspis&nbspa&nbspsecond&nbspsource&nbspof&nbspdata. &nbspThe&nbspsource&nbspis&nbspfrom&nbspBigQuery.

    #&nbspThis&nbspwill&nbspcome&nbspinto&nbspour&nbsppipeline&nbspa&nbspside&nbspinput.

   &nbspread\_query = """

   &nbspSELECT

   &nbspname&nbspas&nbspstate\_name,

   &nbspabbreviation&nbspas&nbspstate\_abbreviation

   &nbspFROM

    `qwiklabs-resources.python\_dataflow\_example.state\_abbreviations`"""

   &nbspstate\_abbreviations = (

       &nbspp

        | 'Read&nbspfrom&nbspBigQuery' >>&nbspbeam.io.Read(

           &nbspbeam.io.BigQuerySource(query=read\_query, use\_standard\_sql=True))

        #&nbspWe&nbspmust&nbspcreate&nbspa&nbsppython&nbsptuple&nbspof&nbspkey&nbspto&nbspvalue&nbsppairs&nbsphere&nbspin&nbsporder&nbspto

        #&nbspuse&nbspthe&nbspdata&nbspas&nbspa&nbspside&nbspinput. &nbspDataflow&nbspwill&nbspuse&nbspthe&nbspkeys&nbspto&nbspdistribute&nbspthe

        #&nbspwork&nbspto&nbspthe&nbspcorrect&nbspworker.

        | 'Abbreviation&nbspto&nbspFull&nbspName' >>&nbspbeam.Map(

            lambda row: (row['state\_abbreviation'],&nbsprow['state\_name'])))

    (p

     #&nbspRead&nbspthe&nbspfile. &nbspThis&nbspis&nbspthe&nbspsource&nbspof&nbspthe&nbsppipeline. &nbspAll&nbspfurther

     #&nbspprocessing&nbspstarts&nbspwith&nbsplines&nbspread&nbspfrom&nbspthe&nbspfile. &nbspWe&nbspuse&nbspthe&nbspinput

     #&nbspargument&nbspfrom&nbspthe&nbspcommand&nbspline. &nbspWe&nbspalso&nbspskip&nbspthe&nbspfirst&nbspline&nbspwhich&nbspis

     #&nbspa&nbspheader&nbsprow.

     | 'Read&nbspFrom&nbspText' >>&nbspbeam.io.ReadFromText(known\_args.input,

                                                skip\_header\_lines=1)

     #&nbspTranslates&nbspfrom&nbspthe&nbspraw&nbspstring&nbspdata&nbspin&nbspthe&nbspCSV&nbspto&nbspa&nbspdictionary.

     #&nbspThe&nbspdictionary&nbspis&nbspa&nbspkeyed&nbspby&nbspcolumn&nbspnames&nbspwith&nbspthe&nbspvalues&nbspbeing&nbspthe&nbspvalues

     #&nbspwe&nbspwant&nbspto&nbspstore&nbspin&nbspBigQuery.

     | 'String&nbspto&nbspBigQuery&nbspRow' >>&nbspbeam.Map(lambda s:

                                           &nbspdata\_ingestion.parse\_method(s))

     #&nbspHere&nbspwe&nbsppass&nbspin&nbspa&nbspside&nbspinput,&nbspwhich&nbspis&nbspdata&nbspthat&nbspcomes&nbspfrom&nbspoutside&nbspour

     #&nbspCSV&nbspsource. &nbspThe&nbspside&nbspinput&nbspcontains&nbspa&nbspmap&nbspof&nbspstates&nbspto&nbsptheir&nbspfull&nbspname.

     | 'Join&nbspData' >>&nbspbeam.Map(add\_full\_state\_name,&nbspAsDict(

       &nbspstate\_abbreviations))

     #&nbspThis&nbspis&nbspthe&nbspfinal&nbspstage&nbspof&nbspthe&nbsppipeline,&nbspwhere&nbspwe&nbspdefine&nbspthe&nbspdestination

     # &nbspof&nbspthe&nbspdata. &nbspIn&nbspthis&nbspcase&nbspwe&nbspare&nbspwriting&nbspto&nbspBigQuery.

     | 'Write&nbspto&nbspBigQuery' >>&nbspbeam.io.Write(

       &nbspbeam.io.BigQuerySink(

            #&nbspThe&nbsptable&nbspname&nbspis&nbspa&nbsprequired&nbspargument&nbspfor&nbspthe&nbspBigQuery&nbspsink.

            #&nbspIn&nbspthis&nbspcase&nbspwe&nbspuse&nbspthe&nbspvalue&nbsppassed&nbspin&nbspfrom&nbspthe&nbspcommand&nbspline.

           &nbspknown\_args.output,

            #&nbspHere&nbspwe&nbspuse&nbspthe&nbspJSON&nbspschema&nbspread&nbspin&nbspfrom&nbspa&nbspJSON&nbspfile.

            #&nbspSpecifying&nbspthe&nbspschema&nbspallows&nbspthe&nbspAPI&nbspto&nbspcreate&nbspthe&nbsptable&nbspcorrectly&nbspif&nbspit&nbspdoes&nbspnot&nbspyet&nbspexist.

            schema=schema,

            #&nbspCreates&nbspthe&nbsptable&nbspin&nbspBigQuery&nbspif&nbspit&nbspdoes&nbspnot&nbspyet&nbspexist.

            create\_disposition=beam.io.BigQueryDisposition.CREATE\_IF\_NEEDED,

            #&nbspDeletes&nbspall&nbspdata&nbspin&nbspthe&nbspBigQuery&nbsptable&nbspbefore&nbspwriting.

            write\_disposition=beam.io.BigQueryDisposition.WRITE\_TRUNCATE)))

   &nbspp.run().wait\_until\_finish()

if \_\_name\_\_ == '\_\_main\_\_':

   &nbsplogging.getLogger().setLevel(logging.INFO)

   &nbsprun()