```
from gremlin_python.driver import client, serializer, protocol
from gremlin_python.driver.protocol import GremlinServerError
import sys
import traceback
import asyncio
import numpy as np
import pandas as pd

#read source file
#df_survey = pd.read_csv('Survey_Estimates.csv')
df_survey = pd.read_excel('SurveyEstimates2.xlsx')

#Select required columns
df_survey = df_survey[[ 'Country and areas','United Nations Region','United Nations Sub-
Region','Overweight','Stunting','Underweight']]

#Rename Columns
df_survey.rename(columns={"Country and areas": "Country", "United Nations Region":
"Continents","United Nations Sub-Region":"Sub_Continents"}, inplace=True)


#Country Vertices
country=df_survey['Country'].unique()

#SubContinents Vertices
sub_continents=df_survey['Sub_Continents'].unique()

#Continents Vertices
continents=df_survey['Continents'].unique()

#For Country Vertices and edges

#take mean
df_countries=df_survey.groupby(['Country']).mean()
df_countries.columns = ['Overweight','Stunting','Underweight']

#Create DataFrame for country with Malnutrion Parameter
l=[]
for x in df_countries.index:

l.append([x,df_countries.loc[x]['Overweight'],df_countries.loc[x]['Stunting'],df_countries.lo
c[x]['Underweight']])

df_countries_final = pd.DataFrame(l)
```

```
df_countries_final.columns = ['Country','Overweight', 'Stunting', 'Underweight']

#Create Dataframe for Continent , Country edge
df_cont_country=df_survey[['Continents','Country']] #.groupby(['Sub_Contitent'])
df_cont_country=df_cont_country.drop_duplicates().reset_index()[['Continents','Country']]

#Create Dataframe for Continent , Sub-Continent edge
df_cont_sub=df_survey[['Continents','Sub_Continents']] #.groupby(['Sub_Contitent'])
df_cont_sub=df_cont_sub.drop_duplicates().reset_index()[['Continents','Sub_Continents']]

#Create Dataframe for Country and Sub-Continent edge
df_sub_cont=df_survey[['Sub_Continents','Country']] #.groupby(['Sub_Contitent'])
df_sub_cont=df_sub_cont.drop_duplicates().reset_index()[['Sub_Continents','Country']]

if sys.platform == 'win32':
   asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())

_gremlin_cleanup_graph = "g.V().drop()"


_gremlin_insert_country_vertices=[]
for index,x in df_countries_final.iterrows():
   str="g.addV('Country').property('id','" +x['Country']+"').property('Country', '"
+x['Country']+"').property('Overweight',"+ repr(x['Overweight'])+" ).property('Stunting',"+
repr(x['Stunting'])+" ).property('Underweight',"+ repr(x['Underweight'])+").property('pk', 'pk')"
   _gremlin_insert_country_vertices.append(str)

#print(_gremlin_insert_country_vertices)

#Add Continents to vertexs
_gremlin_insert_continent_vertices=[]
for x in continents:
   str="g.addV('Continent').property('id','" +x+"').property('Continent', '" +x+"').property('pk',
'pk')"
   _gremlin_insert_continent_vertices.append(str)

#Add sub_continents to vertexs

_gremlin_insert_subcontinent_vertices=[]
for x in sub_continents:
   str="g.addV('Continent').property('id','" +x+"').property('Continent', '" +x+"').property('pk',
'pk')"
   _gremlin_insert_subcontinent_vertices.append(str)
```

```
#Query Continent and Country edge
_gremlin_insert_country_continent_edges=[]
for index,x in df_cont_country.iterrows():
   str="g.V('" +x['Continents']+"').addE('has').to(g.V('" +x['Country']+"'))"
   _gremlin_insert_country_continent_edges.append(str)


#Query Continent and Country edge
_gremlin_insert_subcont_continent_edges=[]
for index,x in df_cont_sub.iterrows():
   str="g.V('" +x['Continents']+"').addE('has').to(g.V('" +x['Sub_Continents']+"'))"
   _gremlin_insert_country_continent_edges.append(str)

#Query Sub Continent and Country edge
_gremlin_insert_subcont_country_edges=[]
for index,x in df_sub_cont.iterrows():
   str="g.V('" +x['Sub_Continents']+"').addE('has').to(g.V('" +x['Country']+"'))"
   _gremlin_insert_subcont_country_edges.append(str)

def print_status_attributes(result):
   # This logs the status attributes returned for successful requests.
   # See list of available response status attributes (headers) that Gremlin API can return:
   #    https://docs.microsoft.com/en-us/azure/cosmos-db/gremlin-headers#headers
   #
   # These responses includes total request units charged and total server latency time.
   #
   # IMPORTANT: Make sure to consume ALL results returend by cliient tothe final status
attributes
   # for a request. Gremlin result are stream as a sequence of partial response messages
   # where the last response contents the complete status attributes set.
   #
   # This can be
   print("\tResponse status_attributes:\n\t{0}".format(result.status_attributes))

def cleanup_graph(client):
   print("\n> {0}".format(
      _gremlin_cleanup_graph))
   callback = client.submitAsync(_gremlin_cleanup_graph)
   if callback.result() is not None:
      callback.result().all().result()
   print("\n")
   print_status_attributes(callback.result())
   print("\n")
```

```
def insert_vertices_country(client):
    for query in _gremlin_insert_country_vertices:
        print("\n> {0}\n".format(query))
        callback = client.submitAsync(query)
        if callback.result() is not None:
            print("\tInserted this vertex:\n\t{0}".format(
                callback.result().all().result()))
        else:
            print("Something went wrong with this query: {0}".format(query))
        print("\n")
        print_status_attributes(callback.result())
        print("\n")

    print("\n")


def insert_vertices_continent(client):
    for query in _gremlin_insert_continent_vertices:
        print("\n> {0}\n".format(query))
        callback = client.submitAsync(query)
        if callback.result() is not None:
            print("\tInserted this vertex:\n\t{0}".format(
                callback.result().all().result()))
        else:
            print("Something went wrong with this query: {0}".format(query))
        print("\n")
        print_status_attributes(callback.result())
        print("\n")

    print("\n")


def insert_vertices_subcontinent(client):
    for query in _gremlin_insert_subcontinent_vertices:
        print("\n> {0}\n".format(query))
        callback = client.submitAsync(query)
        if callback.result() is not None:
            print("\tInserted this vertex:\n\t{0}".format(
                callback.result().all().result()))
        else:
            print("Something went wrong with this query: {0}".format(query))
        print("\n")
        print_status_attributes(callback.result())
```

```
        print("\n")

    print("\n")

def insert_edges_country_continent(client):
    for query in _gremlin_insert_country_continent_edges:
        print("\n> {0}\n".format(query))
        callback = client.submitAsync(query)
        if callback.result() is not None:
            print("\tInserted this edge:\n\t{0}\n".format(
                callback.result().all().result()))
        else:
            print("Something went wrong with this query:\n\t{0}".format(query))
        print_status_attributes(callback.result())
        print("\n")

    print("\n")

def insert_edges_subcont_continent(client):
    for query in _gremlin_insert_subcont_continent_edges:
        print("\n> {0}\n".format(query))
        callback = client.submitAsync(query)
        if callback.result() is not None:
            print("\tInserted this edge:\n\t{0}\n".format(
                callback.result().all().result()))
        else:
            print("Something went wrong with this query:\n\t{0}".format(query))
        print_status_attributes(callback.result())
        print("\n")

    print("\n")

def insert_edges_subcont_country(client):
    for query in _gremlin_insert_subcont_country_edges:
        print("\n> {0}\n".format(query))
        callback = client.submitAsync(query)
        if callback.result() is not None:
            print("\tInserted this edge:\n\t{0}\n".format(
                callback.result().all().result()))
        else:
            print("Something went wrong with this query:\n\t{0}".format(query))
        print_status_attributes(callback.result())
        print("\n")
```

```
    print("\n")

try:
    client = client.Client('wss://graphapi.gremlin.cosmos.azure.com:443/', 'g',
                           username="/dbs/graphdb/colls/Malnutrition",
                           password="XXX ",
                           message_serializer=serializer.GraphSONSerializersV2d0()
                           )

    print("Welcome to Azure Cosmos DB + Gremlin on Python!")

    # Drop the entire Graph
    input("We're about to drop whatever graph is on the server. Press any key to continue...")
    cleanup_graph(client)

    # Insert all vertices
    input("Let's insert some vertices into the graph. Press any key to continue...")
    insert_vertices_country(client)

    insert_vertices_continent(client)

    insert_vertices_subcontinent(client)

    # Create edges between vertices
    #input("Now, let's add some edges between the vertices. Press any key to continue...")
    insert_edges_country_continent(client)
    insert_edges_subcont_continent(client)
    insert_edges_subcont_country(client)

except GremlinServerError as e:
    print('Code: {0}, Attributes: {1}'.format(e.status_code, e.status_attributes))

    # GremlinServerError.status_code returns the Gremlin protocol status code
    # These are broad status codes which can cover various scenaios, so for more specific
    # error handling we recommend using GremlinServerError.status_attributes['x-ms-
status-code']
    #
    # Below shows how to capture the Cosmos DB specific status code and perform specific
error handling.
    # See detailed set status codes than can be returned here:
https://docs.microsoft.com/en-us/azure/cosmos-db/gremlin-headers#status-codes
    #
    # See also list of available response status attributes that Gremlin API can return:
    #    https://docs.microsoft.com/en-us/azure/cosmos-db/gremlin-headers#headers
```

```
cosmos_status_code = e.status_attributes["x-ms-status-code"]
if cosmos_status_code == 409:
   print('Conflict error!')
elif cosmos_status_code == 412:
   print('Precondition error!')
elif cosmos_status_code == 429:
   print('Throttling error!');
elif cosmos_status_code == 1009:
   print('Request timeout error!')
else:
   print("Default error handling")

traceback.print_exc(file=sys.stdout)
sys.exit(1)
```