PLAGIARISM SCAN REPORT

Words	984	Date	April 16,2021
Characters	11385	Excluded URL	

0% Plagiarism	100% Unique	0 Plagiarized Sentences	19 Unique Sentences

Content Checked For Plagiarism

```
"""
PROBLEM STATEMENT:::
Visualzing and Analysing the International Imports and Exports between the year 2013 to 2019..
DataSet from - WTO
PACKAGES USED :::
-> PYVIS - for graph representation
-> PANDAS - to load csv file

WORK DONE SO FAR :::

-> Representing the dataset into Graph.
-> Analysing the Imports and Exports from the dataset.
* Country with Maximum Export
* Country with Minimum Export
* Country with Maximum Import
* Country with Minimum Import
* Best Exporters based on the Products
* Best Relationship between two countries interms of trading.
"""
import pandas as pd
from pyvis.network import Network
import networkx as nx
import matplotlib.pyplot as plt
neighbor_map = [] #list of dictonary of edge details
outd={} #dictionary of exports
ind={} #dictionary of imports
no_outd=dict() #Total number of exports with particular to country
no_ind=dict() #Total number of imports with particular to country
outproduct={} #Dictionary of Product Exports
inproduct={} #Dictionary of Product Imports
maxexpprod={} #Dictionary of maximum exports with respect to products
minexpprod={} #Dictionary of maximum imports with respect to products
#Checking empty cells
def isemptycell (a):
if pd.isna(a):
return 0
else:
return a
#Summing the Imports and Exports of a Country
```

```
def sum_imp_exp_country():
### Total Number of Exports and Imports for each country
for ex_country in outd.keys():
no_outd[ex_country] = len(outd[ex_country])

for im_country in ind.keys():
no_ind[im_country] = len(ind[im_country])

# print("\nOutdegree: ",no_outd,"\n\n")
# print("************************************************************************")
# print("\nIndegree: ",no_ind,"\n\n")
#Minimum and Maximum - Exports to a Country
def minmax_exp_to_country():
maxc=[]
minc=[]

max1 = max(no_outd.values())
min1 = min(no_outd.values())

for c in no_outd:
if no_outd[c] == max1:
maxc.append(c)
if no_outd[c] == min1:
minc.append(c)

return (maxc, max1),(minc, min1)
#Minimum and Maximum - Imports to a Country
def minmax_imp_to_country():
maxc=[]
minc=[]

max1 = max(no_ind.values())
min1 = min(no_ind.values())

for c in no_ind:
if no_ind[c] == max1:
maxc.append(c)
if no_ind[c] == min1:
minc.append(c)

return (maxc, max1),(minc, min1)
#Minimum and Maximum - Exports based on Products
def minmax_exp_product_to_country():
for prod in ['MT2 - 01 - Animal products','MT2 - 02 - Dairy products','MT2 - 03 - Fruits, vegetables, plants','MT2 - 04 -
Coffee, tea']:

maxc=[]
max1 = max(outproduct[prod].values())
for c in outd:

if outproduct[prod].get(c)!= None and outproduct[prod][c] == max1:
maxc.append(c)

maxexpprod[prod]=maxc,max1
return (maxexpprod)
#Best Bond between two countries and to other countries
def bestbond_country():
country_pair={}
for e_node in neighbor_map:
if country_pair.get(e_node['from']+' to '+e_node['to']) == None:
country_pair[e_node['from']+' to '+e_node['to']] = e_node['weight']

else:
tmp = country_pair[e_node['from']+' to '+e_node['to']] + e_node['weight']
country_pair[e_node['from']+' to '+e_node['to']] = tmp
```

```
max1 = max(country_pair.values())
maxc1 = []
maxc2 = []
max2 = 0.0
for c in country_pair:
if c.find('World') == -1 and country_pair[c] > max2:
max2 = country_pair[c]
for c in country_pair:
if country_pair[c] == max1:
maxc1.append(c)
if country_pair[c] == max2:
maxc2.append(c)

return (maxc1,max1),(maxc2,max2)
#Reading the dataset
df = pd.read_csv("w.csv",encoding='latin-1')
#Network Creation
a = Network(directed=True,height="900px",
width="100%",
bgcolor="white",
font_color="black")
G = nx.DiGraph()
for i,j,product,y3,y4,y5,y6,y7,y8,y9 in zip(df['Reporting Economy'], df['Partner Economy'], df['Product/Sector'], df['2013'],
df['2014'], df['2015'], df['2016'], df['2017'], df['2018'], df['2019']):

# Creation Of Nodes
a.add_node(i,i,size=10,title='Country: '+ i + '
')
a.add_node(j,j,size=10,title='Country: '+ j + '
')

tot_export = sum(list(map(isemptycell,[y3,y4,y5,y6,y7,y8,y9])))

#Creation of Edges
a.add_edge(i,j,physics=False,width=tot_export//10**10,weight=tot_export, title=product)

# G.add_node(i)
# G.add_node(j)
G.add_edge(i,j,weight=tot_export)
edg = list(G.edges())
def triadic_closure(edge_list):
n_edges = []
for i in edge_list:
a, b = i
for j in edge_list:
x, y = j
if i != j:
if a == x and (b, y) not in edge_list and (y, b) not in edge_list:
n_edges.append((a,b, y))
if a == y and (b, x) not in edge_list and (x, b) not in edge_list:
n_edges.append((a,b, x))
if b == x and (a, y) not in edge_list and (y, a) not in edge_list:
n_edges.append((b,a, y))
if b == y and (a, x) not in edge_list and (x, a) not in edge_list:
n_edges.append((b,a, x))
return n_edges
print("The possible new edges according to Triadic closure are :")
print(triadic(e))
pos=nx.spring_layout(G)
G = nx.draw_networkx_edge_labels(G,pos)
plt.savefig("path_graph_cities.png")
plt.show()
#Complete dictionary of edges
neighbor_map = a.get_edges()
```

```
#Parsing the Graph
for e_node in neighbor_map:
if outd.get(e_node['from']) == None:
outd[e_node['from']] = {e_node['to']}
else:
outd[e_node['from']].add(e_node['to'])

if ind.get(e_node['to']) == None:
ind[e_node['to']] = {e_node['from']}
else:
ind[e_node['to']].add(e_node['from'])

if outproduct.get(e_node['title']) == None:
outproduct[e_node['title']] = {}
if outproduct[e_node['title']].get(e_node['from']) == None:
outproduct[e_node['title']][e_node['from']] = e_node['weight']
else:
outproduct[e_node['title']][e_node['from']] += e_node['weight']

if inproduct.get(e_node['title']) == None:
inproduct[e_node['title']] = {}
if inproduct[e_node['title']].get(e_node['to']) == None:
inproduct[e_node['title']][e_node['to']] = e_node['weight']
else:
inproduct[e_node['title']][e_node['to']] += e_node['weight']
print(outd)
print()
print("\n****************************************\n")
print()
print(ind)
print()
print("\n****************************************\n")
print()
print(outproduct)
print()
print("\n****************************************\n")
print()
print(inproduct)
products=['MT2 - 01 - Animal products','MT2 - 02 - Dairy products',
'MT2 - 03 - Fruits, vegetables, plants',
'MT2 - 04 - Coffee, tea']
print("\n***************************************************************************\n")
print("Largest Distributor of the products to other countries\n")
distributor_exports=dict()
for prod in products:
temp=dict()
for e in neighbor_map:
if(e['title']==prod):
try:
temp[e['from']]+=1
except:
temp[e['from']]=0
distributor_exports[prod]=temp
for prod,country in distributor_exports.items():
#country['World']=0
print(prod,' : ',max(country, key=country.get))
print("\n***************************************************************************\n")
print("Countries which has largest no of distributors for the products\n")
distributor_imports=dict()
for prod in products:
temp=dict()
for e in neighbor_map:
if(e['title']==prod):
try:
temp[e['to']]+=1
```

```
except:
temp[e['to']]=0
distributor_imports[prod]=temp
for prod,country in distributor_imports.items():
country['World']=0
print(prod,' : ',max(country, key=country.get))
# Printing the Analysis
sum_imp_exp_country()
print("\n\nMaximum Export:")
maxexp,minexp=minmax_exp_to_country()
print(maxexp[0][0],maxexp[1],sep=' - ')
print("\nMinimum Export:")
print(minexp[0][0],minexp[1],sep=' - ')
print("\n***************************************\n")
maximp,minimp=minmax_imp_to_country()
print("Maximum Import:")
print(maximp[0][0],maximp[1],sep=' - ')
print("\nMinimum Import:")
print(minimp[0][0],minimp[0][1],sep=' , ',end=' - ')
print(minimp[1])
print("\n***************************************\n")
print("\nBest Product Exporters:\n")
maxexpproduct=minmax_exp_product_to_country()
for i in maxexpproduct:
print(i,end=' : ')
print(maxexpproduct[i][0][0],' - ',maxexpproduct[i][1])

print("\n***************************************\n")

print("Best relationship with other countries:")
max1,max2=bestbond_country()
print(max1[0][0],' - ',max1[1])
print("\nBest bond between two countries:")
print(max2[0][0],' - ',max2[1])
# Adding the node Details
for node in a.nodes:
if outd.get(node['label']) != None:
node["title"] += "Exports: " + " , ".join(outd[node['label']])
node["title"] += "
"
node["size"] = len(outd[node['label']])
if ind.get(node['label']) != None:
node["title"] += "Imports: " + " , ".join(ind[node['label']])
# Graph Appearance Parameters
a.set_options("""var options = {
"nodes": {
"font": {
"size": 30
},
"color": {
"border": "blue",
"background": "#293250",
"highlight": {
"border": "blue",
"background": "yellow"
}
}

},
"edges": {
"color": {
"color": "#6DD47E",
"highlight": "yellow",
"inherit": false
},
"smooth": false
},
"physics": {
"repulsion": {
"nodeDistance": 395
},
"minVelocity": 0.75,
"solver": "repulsion"
}
}""")
for e_node in neighbor_map:
e_node['title'] = e_node['title']+" ( "+ str(e_node['weight']) +" )"
#Graph Output
a.show("example.html")
"""
FUTURE WORKS :::
-> Indepth Analysis Of Graph Network
-> Building a GUI to display the analysis

"""
```

Sources	Similarity