

Check files existence

```
In [27]: import os
print(os.listdir("."))

['flights.csv', 'networkX_sample.ipynb', 'airlines.csv', 'flights2.csv', '620Group3_Project1_ver2.ipynb', '620Group3_Project1.ipynb', '620Group3_Project1_version1.pdf', 'small_flights.csv', 'airports.csv', '.ipynb_checkpoints']
```

Read flights to networkx

either by raw csv or from Saved Pajek file

```
In [28]: import networkx as nx
import matplotlib.pyplot as plt

g=nx.DiGraph()
```

```

In [29]: #read and add to graph if pajet network data doesn't exist

if not os.path.isfile(r'flights_edges.txt'):
    print("read from flights.csv")
    myfile = open("flights.csv", encoding='utf-8' )

    line = myfile.readline()

    while line:
        line = myfile.readline()
        v = line.split(',')

        if( len(v) == 31 ):
            if( len(v[7]) == 3 and len(v[8]) == 3): #skip airport code is not x
xx                g.add_edge(v[7], v[8])

    myfile.close()
    print (nx.info(g))
    #Saving network data
    nx.write_pajek(g, r'flights_edges.txt')

else:
    print("read from flight_edges.txt")
    g = nx.read_pajek(r'flights_edges.txt')
    print (nx.info(g))

read from flights.csv
Name:
Type: DiGraph
Number of nodes: 322
Number of edges: 4693
Average in degree: 14.5745
Average out degree: 14.5745

```

Compute network data

Degrees = the most connected airports

```
In [30]: deg = nx.degree(g)
```

check deg data type

```
In [35]: type(deg)
```

```
Out[35]: networkx.classes.reportviews.DiDegreeView
```

DiDegreeView return two-tuple (node, degree) according to networkX documentation

Sort DiDegreeView to find min and max degree

```
In [32]: def sorted_map(map):  
         ms = sorted(map, key=lambda x: (-x[1],x[0]))  
         return ms  
  
sorted_deg = sorted_map(deg)
```

```
In [33]: for node, degree in sorted_deg:
          print (node, degree)
```

ATL 338
ORD 324
DFW 297
DEN 279
MSP 240
IAH 238
DTW 224
SLC 179
EWR 174
LAX 161
SFO 161
PHX 158
LAS 156
MCO 148
SEA 146
IAD 139
LGA 139
CLT 138
MDW 138
BWI 134
JFK 129
BOS 124
TPA 122
FLL 120
DCA 116
MIA 109
DAL 104
HOU 102
STL 97
PDX 95
BNA 93
PHL 93
SAN 93
AUS 84
CVG 81
MCI 81
RSW 79
MSY 75
CLE 72
RDU 71
OAK 69
ANC 60
IND 60
PIT 60
SAT 60
MKE 58
CMH 57
HNL 56
SMF 54
BDL 50
PBI 50
SJC 50
ABQ 46
CHS 46
MEM 46
SJU 46
JAX 44
OKC 44
SNA 44
OMA 40
TTN 37
BUF 36
OGG 36

Top 10 Nodes by Degree

```
In [56]: sorted_deg[0:10]
```

```
Out[56]: [('ATL', 338),
          ('ORD', 324),
          ('DFW', 297),
          ('DEN', 279),
          ('MSP', 240),
          ('IAH', 238),
          ('DTW', 224),
          ('SLC', 179),
          ('EWR', 174),
          ('LAX', 161)]
```

Bottem 10 Nodes with Degree

```
In [57]: sorted_deg[-10:]
```

```
Out[57]: [('STC', 2),
          ('SUX', 2),
          ('TOL', 2),
          ('TRI', 2),
          ('TWF', 2),
          ('TXK', 2),
          ('VEL', 2),
          ('VLD', 2),
          ('WYS', 2),
          ('YUM', 2)]
```

Trim Degree function

```
In [52]: def trim_degree(g, degree=1):
          g2 = g.copy()
          d = nx.degree(g2)
          for n in g.nodes():
              if d[n] <= degree: g2.remove_node(n)
          return g2
```

get top nodes

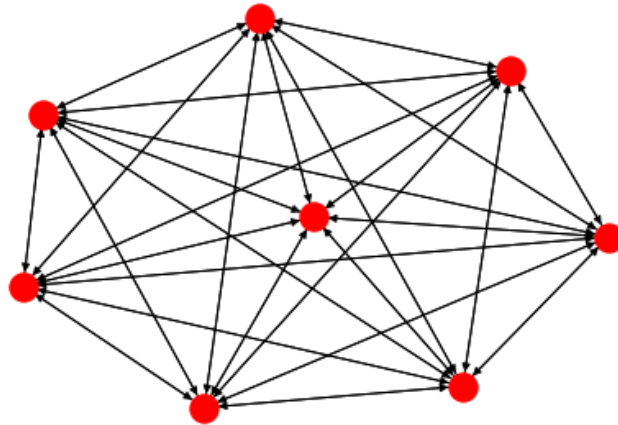
```
In [60]: topnodes = trim_degree(g, 161)
```

Check # of nodes and print nodes

```
In [61]: len(topnodes)
```

```
Out[61]: 8
```

```
In [62]: nx.draw(topnodes)
```



Calculate Centrality

```
In [63]: c = nx.closeness centrality(topnodes)
```

```
In [64]: sorted_c = sorted_map(c)
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-64-ae791ea23401> in <module>()
----> 1 sorted_c = sorted_map(c)

<ipython-input-32-bae802aa058e> in sorted_map(map)
      1 def sorted_map(map):
----> 2     ms = sorted(map, key=lambda x: (-x[1],x[0]))
      3     return ms
      4
      5 sorted_deg = sorted_map(deg)

<ipython-input-32-bae802aa058e> in <lambda>(x)
      1 def sorted_map(map):
----> 2     ms = sorted(map, key=lambda x: (-x[1],x[0]))
      3     return ms
      4
      5 sorted_deg = sorted_map(deg)

TypeError: bad operand type for unary -: 'str'
```

```
In [66]: type(c)
```

```
Out[66]: dict
```

```
In [67]: print(c)
```

```
{'MSP': 1.0, 'DFW': 1.0, 'ATL': 1.0, 'DEN': 1.0, 'SLC': 1.0, 'IAH': 1.0, 'ORD': 1.0, 'DTW': 1.0}
```

eigenvector centrality

```
In [68]: ec = nx.eigenvector centrality(topnodes)
```

```
In [69]: type(ec)
```

```
Out[69]: dict
```

```
In [70]: print(ec)
```

```
{'MSP': 0.3535533905932738, 'DFW': 0.3535533905932738, 'ATL': 0.3535533905932738, 'DEN': 0.3535533905932738, 'SLC': 0.3535533905932738, 'IAH': 0.3535533905932738, 'ORD': 0.3535533905932738, 'DTW': 0.3535533905932738}
```

```
In [ ]:
```