Social network Graph Link Prediction - Facebook Challenge

Problem statement:

Given a directed social graph, have to predict missing links to recommend users (Link Prediction in graph)

In social network, each user can be treated as a node/vertex and the connection/link between two users can be represented as a edge. So there can be many users and edges. Now for a user (new/old) having/without having connection with other users, we need to recommend a connection to a user for that user. Means in sort if we create an edge between two nodes (new user, target user), would it be a valid edge or not (Model should recommend a user to the new user or not).

A graph consists of two things (node/vertex, edge) and a edge can be represented by two values (value/id of the two nodes which are connected by that edge).

A edge can have direction. Means for eg. in Fb users are friends of each other. So if U1 is a friend of U2, then U2 must be a friend of U1. Here we dont need direction. But in instagram, people follow each other. Its not mandatory that the user needs to follow the person who follows that user. So simply if U1 follows U2, then edge should be U1 -> U2. If U2 follows U1, it should be U1 <- U2. Similarly if both users follow each other, then edge should be U1 <-> U2.

Path between two nodes is the sequence of edges we need to follow if we want to move from node 1 to node 2. Its means in how many ways two users are connected. As two users can be multiple common friends, there can be multiple connections, which represents the paths between those users.

Now we need convert the simple node values into graph specific terms and we can use different functions of graph using networkx.

Data Overview

Taken data from facebook's recruting challenge on kaggle https://www.kaggle.com/c/FacebookRecruiting
data contains two columns source and destination eac edge in graph

```
- Data columns (total 2 columns):
- source_node         int64
- destination_node    int64
```

One row represents a directional edge between source and destination. But in real world, the connections changes with time.

Mapping the problem into supervised learning problem:

- Generated training samples of good and bad links from given directed graph and for each link got some features like no of followers, is he followed back, page rank, katz score, adar index, some svd fetures of adj matrix, some weight features etc. and trained ml model based on these features to predict link.
- Some reference papers and videos :
  - https://www.cs.cornell.edu/home/kleinber/link-pred.pdf
  - https://www3.nd.edu/~dial/publications/lichtenwalter2010new.pdf
  - https://kaggle2.blob.core.windows.net/forum-message-attachments/2594/supervised_link_prediction.pdf
  - https://www.youtube.com/watch?v=2M77Hgy17cg

We can have following strategy. If two nodes have connection, we can assign 1, else 0. Now for two nodes having 0 value, we will predict edge. This can be done by checking common/mutual connection nodes of the two nodes. If the number exceeds a certain value, we can create a edge between those two nodes. Also if node1 and node2 have an edge, then there is high chance that node2 and node1 will have the edge. There are other graph specific factors we can consider to predict connection between the nodes.

Business objectives and constraints:

- No low-latency requirement.
- Probability of prediction is useful to recommend ighest probability links

Performance metric for supervised learning:

- Both precision and recall is important so F1 score is good choice (precision@10 can be used..as out of all recommended users, we need to show top 10 users etc.)
- Confusion matrix

In [1]:

```
#Importing Libraries
# please do go through this python notebook:
import warnings
warnings.filterwarnings("ignore")
```

```
!pip install xgboost
import csv
import pandas as pd#pandas to create small dataframes
import datetime #Convert to unix time
import time #Convert to unix time
# if numpy is not installed already : pip3 install numpy
import numpy as np#Do aritmetic operations on arrays
# matplotlib: used to plot graphs
import matplotlib
import matplotlib.pylab as plt
import seaborn as sns#Plots
from matplotlib import rcParams#Size of plots
from sklearn.cluster import MiniBatchKMeans, KMeans#Clustering
import math
import pickle
import os
# to install xgboost: pip3 install xgboost
import xgboost as xgb

import warnings
import networkx as nx
import pdb
import pickle
```

```
WARNING: You are using pip version 20.2.1; however, version 20.2.4 is available.
You should consider upgrading via the 'c:\users\hp\appdata\local\programs\python\python36\python.exe -m
pip install --upgrade pip' command.
Requirement already satisfied: xgboost in c:\users\hp\appdata\local\programs\python\python36\lib\site-pac
kages (1.2.1)
Requirement already satisfied: numpy in c:\users\hp\appdata\local\programs\python\python36\lib\site-packa
ges (from xgboost) (1.18.5)
Requirement already satisfied: scipy in c:\users\hp\appdata\local\programs\python\python36\lib\site-packa
ges (from xgboost) (1.4.1)
```

In [2]:

```
#reading graph
if not os.path.isfile('data/after_eda/train_woheader.csv'):
    traincsv = pd.read_csv('data/train.csv')
    print(traincsv[traincsv.isna().any(1)])
    print(traincsv.info())
    print("Number of diplicate entries: ",sum(traincsv.duplicated()))
    traincsv.to_csv('data/after_eda/train_woheader.csv',header=False,index=False)
    print("saved the graph into file")
else:
    # Create a graph using the data points. It should be directedGraph.
    g=nx.read_edgelist('data/after_eda/train_woheader.csv',delimiter=',',create_using=nx.DiGraph(),nodety
    print(nx.info(g))
```

```
Name:
Type: DiGraph
Number of nodes: 1862220
Number of edges: 9437519
Average in degree:   5.0679
Average out degree:   5.0679
```

Node: each point/user is a node.

Edge: Connection between two nodes is edge.

In_degree: Number of edges to a node

Out_degree: Number of edges from a node.

From the dataset, we can get the graph. It will have nodes and edges. Now from graph, we can get different features and analyse.

Displaying a sub graph

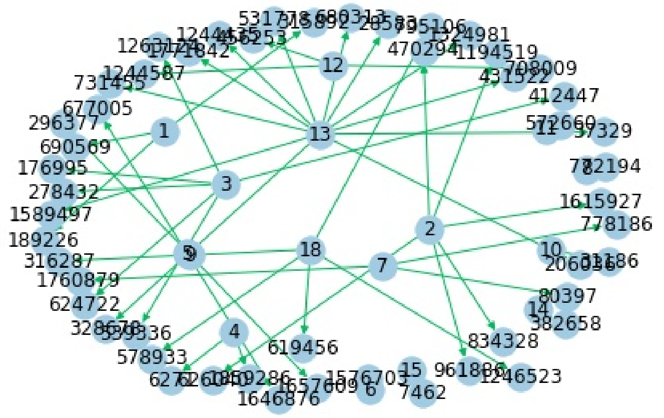
In [3]:

```
if not os.path.isfile('train_woheader_sample.csv'):
    pd.read_csv('data/train.csv', nrows=50).to_csv('train_woheader_sample.csv',header=False,index=False)

subgraph=nx.read_edgelist('train_woheader_sample.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=in
# https://stackoverflow.com/questions/9402255/drawing-a-huge-graph-with-networkx-and-matplotlib

pos=nx.spring_layout(subgraph)
nx.draw(subgraph,pos,node_color='#A0CBE2',edge_color='#00bb5e',width=1,edge_cmap=plt.cm.Blues,with_labels
plt.savefig("graph_sample.pdf")
print(nx.info(subgraph))
```

```
Name:
Type: DiGraph
Number of nodes: 66
Number of edges: 50
Average in degree:   0.7576
Average out degree:   0.7576
```



1. Exploratory Data Analysis

In [4]:

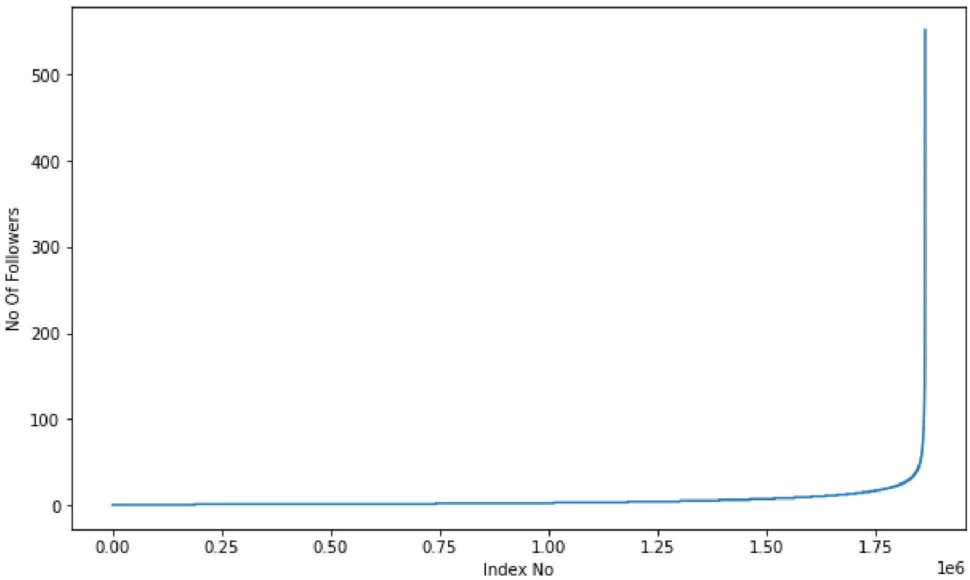
```
# No of Unique persons
print("The number of unique persons",len(g.nodes()))
```

```
The number of unique persons 1862220
```

1.1 No of followers for each person

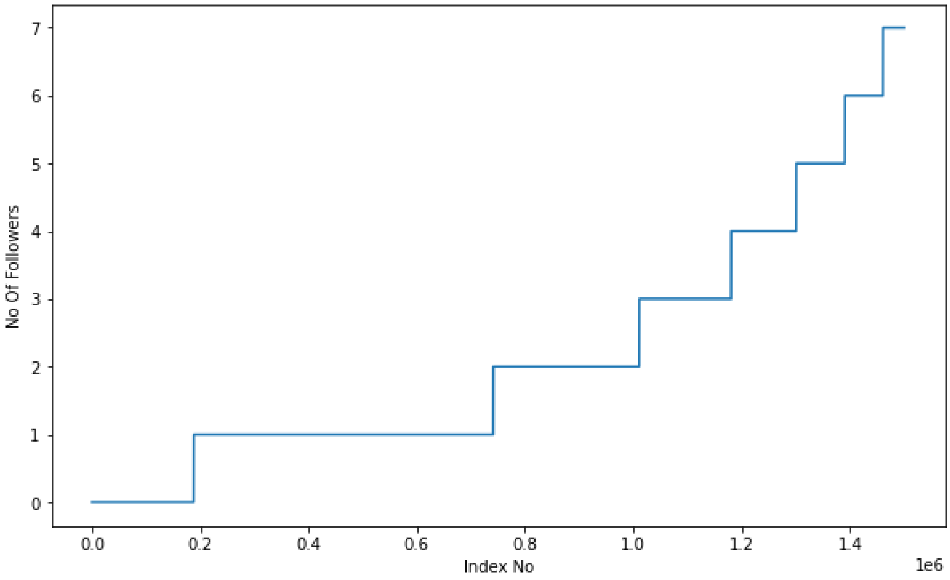
In [5]:

```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```



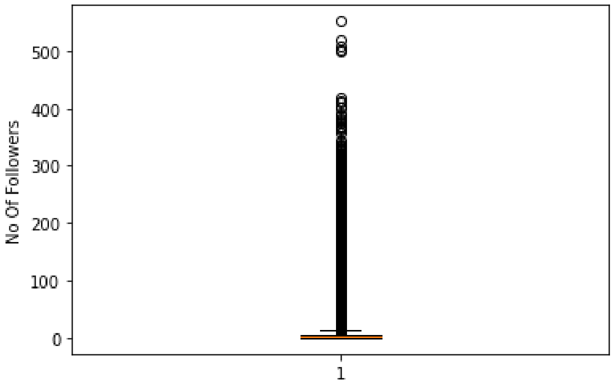
In [6]:

```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(indegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of Followers')
plt.show()
```



In [7]:

```
plt.boxplot(indegree_dist)
plt.ylabel('No Of Followers')
plt.show()
```



In [8]:

```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(indegree_dist,90+i))
```

```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value is 552.0
```

99% of data having followers of 40 only.

From the above plots, we can notice there are some outliers. Most of the users have follows till 40.

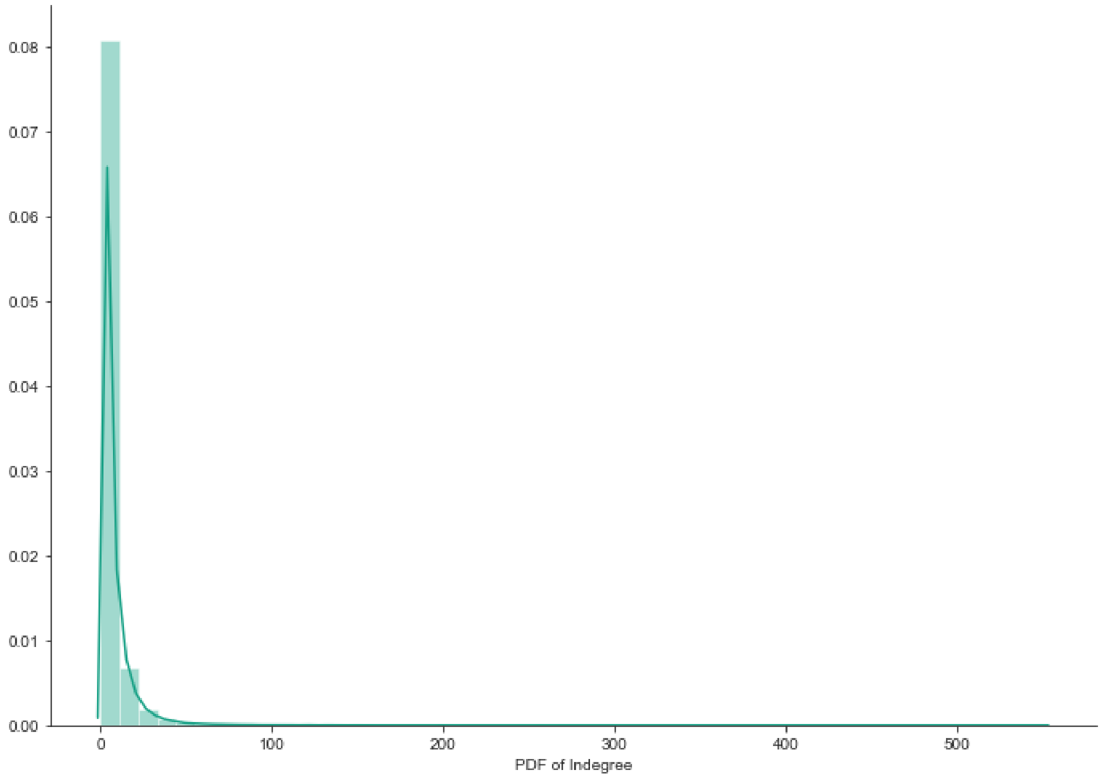
In [9]:

```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(indegree_dist,99+(i/100)))
```

```
99.1 percentile value is 42.0
99.2 percentile value is 44.0
99.3 percentile value is 47.0
99.4 percentile value is 50.0
99.5 percentile value is 55.0
99.6 percentile value is 61.0
99.7 percentile value is 70.0
99.8 percentile value is 84.0
99.9 percentile value is 112.0
100.0 percentile value is 552.0
```

In [10]:

```
%matplotlib inline
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(indegree_dist, color='#16A085')
plt.xlabel('PDF of Indegree')
sns.despine()
#plt.show()
```

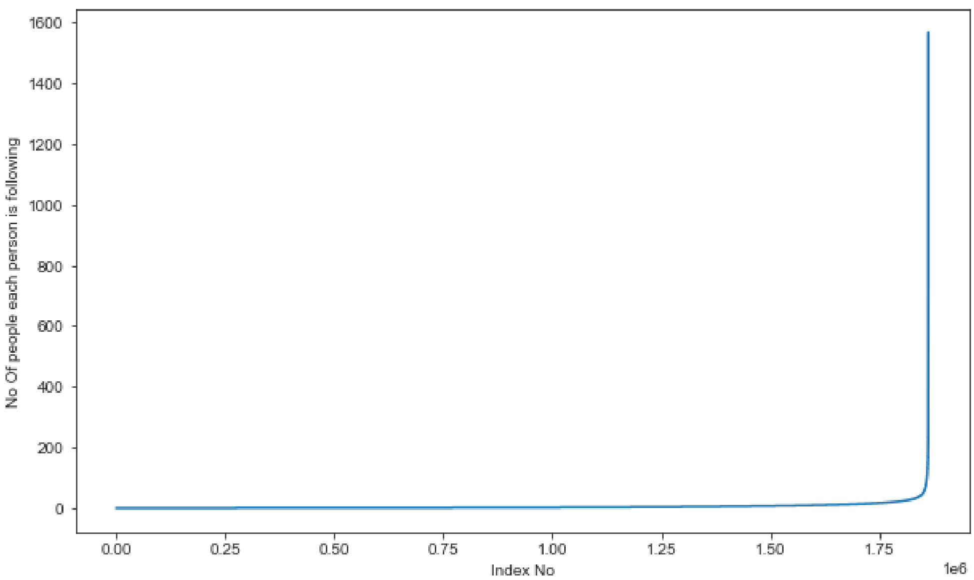


From the above plots and data, we can notice that for maximum users, the followers count is less/nomal (40-112), but for few users, they have followers count much more than others. Even 99% of users have less/average number of followers.

1.2 No of people each person is following

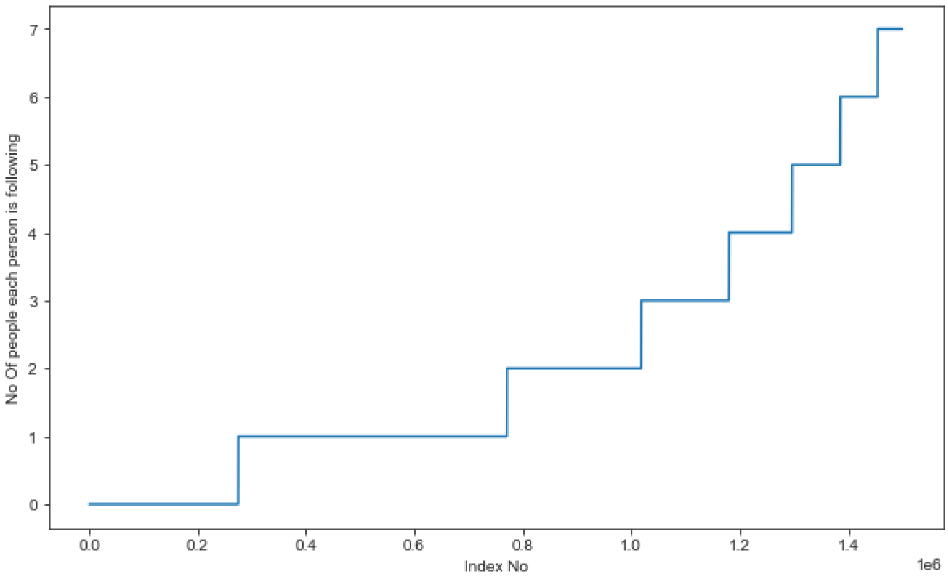
In [11]:

```
outdegree_dist = list(dict(g.out_degree()).values())
outdegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist)
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
plt.show()
```



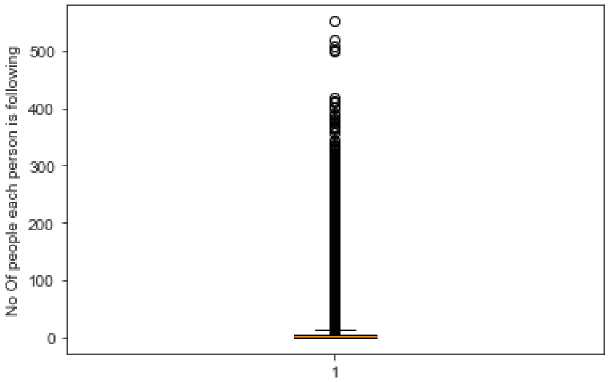
In [12]:

```
indegree_dist = list(dict(g.in_degree()).values())
indegree_dist.sort()
plt.figure(figsize=(10,6))
plt.plot(outdegree_dist[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following')
plt.show()
```



In [13]:

```
plt.boxplot(indegree_dist)
plt.ylabel('No Of people each person is following')
plt.show()
```



In [14]:

```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(outdegree_dist,90+i))
```

```
90 percentile value is 12.0
91 percentile value is 13.0
92 percentile value is 14.0
93 percentile value is 15.0
94 percentile value is 17.0
95 percentile value is 19.0
96 percentile value is 21.0
97 percentile value is 24.0
98 percentile value is 29.0
99 percentile value is 40.0
100 percentile value is 1566.0
```

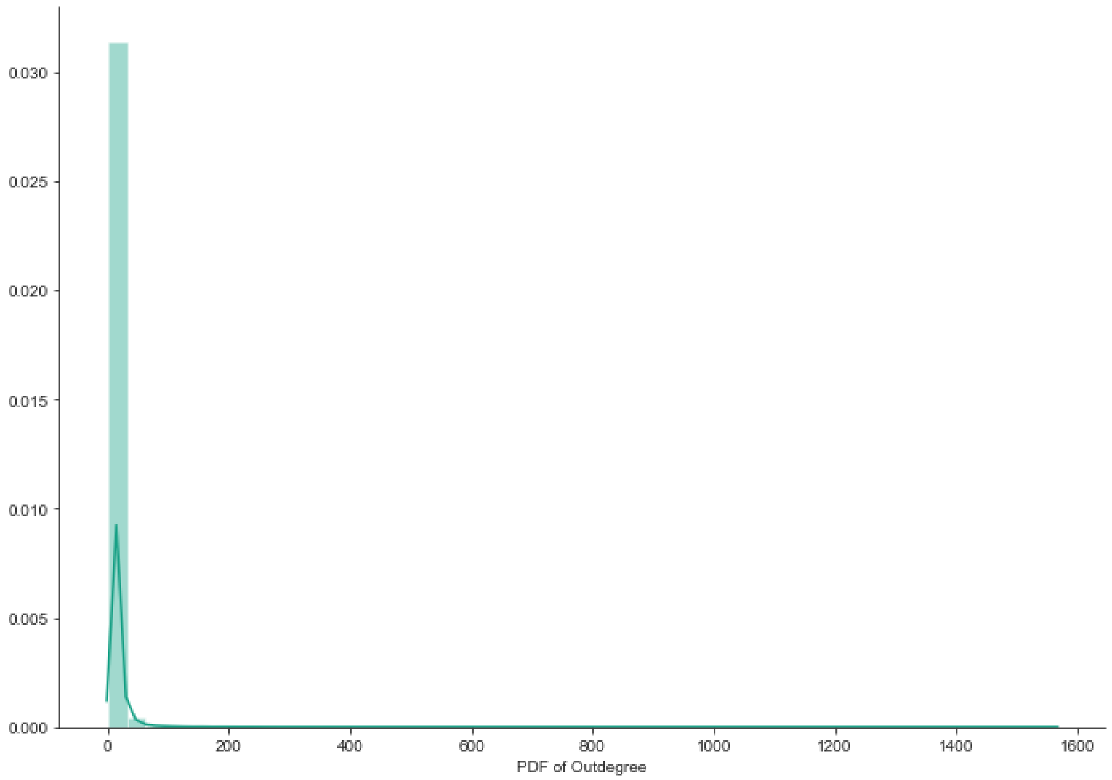
In [15]:

```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(outdegree_dist,99+(i/100)))
```

```
99.1 percentile value is 42.0
99.2 percentile value is 45.0
99.3 percentile value is 48.0
99.4 percentile value is 52.0
99.5 percentile value is 56.0
99.6 percentile value is 63.0
99.7 percentile value is 73.0
99.8 percentile value is 90.0
99.9 percentile value is 123.0
100.0 percentile value is 1566.0
```

In [16]:

```
sns.set_style('ticks')
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
sns.distplot(outdegree_dist, color='#16A085')
plt.xlabel('PDF of Outdegree')
sns.despine()
```



Similarly number of people the users are following are low/normal (40 - 123) for maximum 90-99% of total users, but for some the count is high.

Very few users are following many people.

In [17]:

```
print('No of persons those are not following anyone are' ,sum(np.array(outdegree_dist)==0),'and % is',
                                sum(np.array(outdegree_dist)==0)*100/len(outdegree_dist) )
```

```
No of persons those are not following anyone are 274512 and % is 14.741115442858524
```

In [18]:

```
print('No of persons having zero followers are' ,sum(np.array(indegree_dist)==0),'and % is',
                                sum(np.array(indegree_dist)==0)*100/len(indegree_dist) )
```

```
No of persons having zero followers are 188043 and % is 10.097786512871734
```

In [19]:

```
count=0
for i in g.nodes():
    if len(list(g.predecessors(i)))==0 :
        if len(list(g.successors(i)))==0:
            count+=1
print('No of persons those are not not following anyone and also not having any followers are',count)
```

```
No of persons those are not not following anyone and also not having any followers are 0
```

1.3 both followers + following

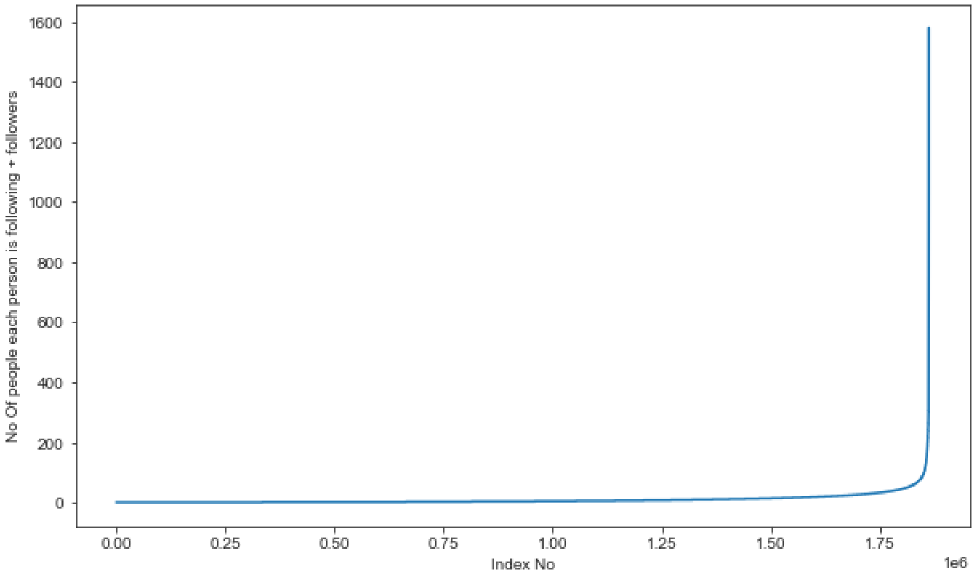
In [20]:

```
from collections import Counter
dict_in = dict(g.in_degree())
```

```
dict_out = dict(g.out_degree())
d = Counter(dict_in) + Counter(dict_out)
in_out_degree = np.array(list(d.values()))
```

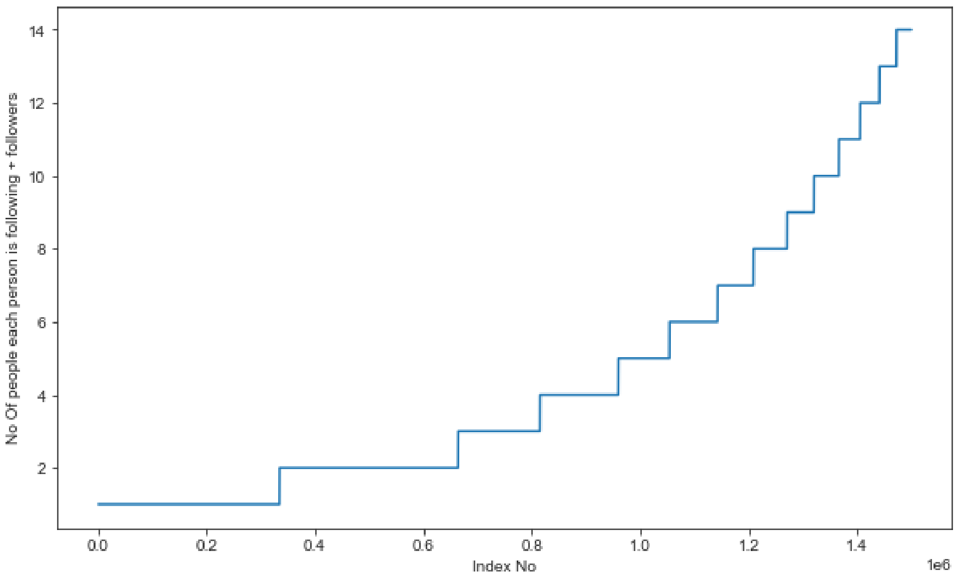
In [21]:

```
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
plt.plot(in_out_degree_sort)
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



In [22]:

```
in_out_degree_sort = sorted(in_out_degree)
plt.figure(figsize=(10,6))
plt.plot(in_out_degree_sort[0:1500000])
plt.xlabel('Index No')
plt.ylabel('No Of people each person is following + followers')
plt.show()
```



In [23]:

```
### 90-100 percentile
for i in range(0,11):
    print(90+i,'percentile value is',np.percentile(in_out_degree_sort,90+i))
```

```
90 percentile value is 24.0
91 percentile value is 26.0
92 percentile value is 28.0
93 percentile value is 31.0
94 percentile value is 33.0
95 percentile value is 37.0
96 percentile value is 41.0
97 percentile value is 48.0
98 percentile value is 58.0
99 percentile value is 79.0
100 percentile value is 1579.0
```

In [24]:

```
### 99-100 percentile
for i in range(10,110,10):
    print(99+(i/100),'percentile value is',np.percentile(in_out_degree_sort,99+(i/100)))
```

```
99.1 percentile value is 83.0
99.2 percentile value is 87.0
99.3 percentile value is 93.0
99.4 percentile value is 99.0
99.5 percentile value is 108.0
99.6 percentile value is 120.0
99.7 percentile value is 138.0
99.8 percentile value is 168.0
99.9 percentile value is 221.0
100.0 percentile value is 1579.0
```

As number of followers and followee are low/normal for maximum number people, the total count is also similar. 90-99% of users have total count as (79 - 221). Very few users have total count much more than the count of other users.

In [25]:

```
print('Min of no of followers + following is',in_out_degree.min())
print(np.sum(in_out_degree==in_out_degree.min()),' persons having minimum no of followers + following')
```

```
Min of no of followers + following is 1
334291  persons having minimum no of followers + following
```

In [26]:

```
print('Max of no of followers + following is',in_out_degree.max())
print(np.sum(in_out_degree==in_out_degree.max()),' persons having maximum no of followers + following')
```

```
Max of no of followers + following is 1579
1  persons having maximum no of followers + following
```

In [27]:

```
print('No of persons having followers + following less than 10 are',np.sum(in_out_degree<10))
```

```
No of persons having followers + following less than 10 are 1320326
```

In [28]:

```
print('No of weakly connected components',len(list(nx.weakly_connected_components(g))))
count=0
for i in list(nx.weakly_connected_components(g)):
    if len(i)==2:
        count+=1
print('weakly connected components with 2 nodes',count)
```

```
No of weakly connected components 45558
weakly connected components with 2 nodes 32195
```

The whole graph can be divided into some subsets, such that from one vertext, we can move to another by ignoring direction. Here we are getting subsets of size 2.

2. Posing a problem as classification problem

We can a edge between 2 nodes. For the given dataset, edge exists between given points. But if total number of points is n, then each node can be connected (n-1) nodes, so number of edges from a single node is (n-1) and in total it will be n(n-1). We already have some edges defined. If we assign 1 for existing edges, for rest of the estimated edges, value will be 0. But n(n-1) - pre_existed edges is still a large number.

So better to sample the nodes from possible edges which are not present (bad-edges) and assign 0 and we can make a balanced dataset with class level 0 and 1.

0- possible edges currently not present

1- currently available edges

2.1 Generating some edges which are not present in graph for supervised learning

Generated Bad links from graph which are not in graph and whose shortest path is greater than 2.

In [29]:

```
%%time
###generating bad edges from given graph
import random
if not os.path.isfile('data/after_eda/missing_edges_final.p'):
    #getting all set of edges
    r = csv.reader(open('data/after_eda/train_woheader.csv','r'))
    edges = dict()
    for edge in r:
        edges[(edge[0], edge[1])] = 1


    missing_edges = set([])
    while (len(missing_edges)<9437519):
        a=random.randint(1, 1862220)
        b=random.randint(1, 1862220)
        tmp = edges.get((a,b),-1)
        if tmp == -1 and a!=b:
            try:
                if nx.shortest_path_length(g,source=a,target=b) > 2:

                    missing_edges.add((a,b))
                else:
                    continue
            except:
                    missing_edges.add((a,b))
        else:
            continue
    pickle.dump(missing_edges,open('data/after_eda/missing_edges_final.p','wb'))
else:
    missing_edges = pickle.load(open('data/after_eda/missing_edges_final.p','rb'))
```

```
Wall time: 25.4 s
```

In [30]:

```
len(missing_edges)
```

Out[30]:

```
9437519
```

To make clssification probem with given node value, class level 1 is assigned if an edge exists between two nodes (For all row values in training, edge exists). Now we can get some random node values and check if there is no edge, we can assign 0 class.(Given the shortest distance between two nodes should be more than 2)

2.2 Training and Test data split:

Removed edges from Graph and used as test data and after removing used that graph for creating features for Train and test data.

Here actually we need to do time based splitting. But as timestamp is not given, we are doing random splitting.

In [31]:

```
from sklearn.model_selection import train_test_split
if (not os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (not os.path.isfile('data/after_eda
    #reading total data df
    df_pos = pd.read_csv('data/train.csv')
    df_neg = pd.DataFrame(list(missing_edges), columns=['source_node', 'destination_node'])

    print("Number of nodes in the graph with edges", df_pos.shape[0])
    print("Number of nodes in the graph without edges", df_neg.shape[0])

    #Trian test split
    #Spiltted data into 80-20
    #positive links and negative links seperatly because we need positive training data only for creatin
    #and for feature generation
    X_train_pos, X_test_pos, y_train_pos, y_test_pos  = train_test_split(df_pos,np.ones(len(df_pos)),test
    X_train_neg, X_test_neg, y_train_neg, y_test_neg  = train_test_split(df_neg,np.zeros(len(df_neg)),tes

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train_pos.shape[0],"=",y_train_pos.shap
    print("Number of nodes in the train data graph without edges", X_train_neg.shape[0],"=", y_train_neg.
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_pos.shape[0],"=",y_test_pos.shape[C
    print("Number of nodes in the test data graph without edges", X_test_neg.shape[0],"=",y_test_neg.shap
```

```
    #removing header and saving
    X_train_pos.to_csv('data/after_eda/train_pos_after_eda.csv',header=False, index=False)
    X_test_pos.to_csv('data/after_eda/test_pos_after_eda.csv',header=False, index=False)
    X_train_neg.to_csv('data/after_eda/train_neg_after_eda.csv',header=False, index=False)
    X_test_neg.to_csv('data/after_eda/test_neg_after_eda.csv',header=False, index=False)
else:
    #Graph from Traing data only
    del missing_edges
```

In [ ]:

```
if (os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and (os.path.isfile('data/after_eda/test_pc
    train_graph=nx.read_edgelist('data/after_eda/train_pos_after_eda.csv',delimiter=',',create_using=nx.I
    test_graph=nx.read_edgelist('data/after_eda/test_pos_after_eda.csv',delimiter=',',create_using=nx.Di(
    print(nx.info(train_graph))
    print(nx.info(test_graph))

    # finding the unique nodes in the both train and test graphs
    train_nodes_pos = set(train_graph.nodes())
    test_nodes_pos = set(test_graph.nodes())

    trY_teY = len(train_nodes_pos.intersection(test_nodes_pos))
    trY_teN = len(train_nodes_pos - test_nodes_pos)
    teY_trN = len(test_nodes_pos - train_nodes_pos)

    print('no of people common in train and test -- ',trY_teY)
    print('no of people present in train but not present in test -- ',trY_teN)

    print('no of people present in test but not present in train -- ',teY_trN)
    print(' % of people not there in Train but exist in Test in total Test data are {} %'.format(teY_trN,
```

```
Name: 
Type: DiGraph
Number of nodes: 1780722
Number of edges: 7550015
Average in degree:   4.2399
Average out degree:   4.2399
Name: 
Type: DiGraph
Number of nodes: 1144623
Number of edges: 1887504
Average in degree:   1.6490
Average out degree:   1.6490
no of people common in train and test --  1063125
no of people present in train but not present in test --  717597
no of people present in test but not present in train --  81498
 % of people not there in Train but exist in Test in total Test data are 7.1200735962845405 %
```

we have a cold start problem here

In [32]:

```
#final train and test data sets
if (not os.path.isfile('data/after_eda/train_after_eda.csv')) and \
(not os.path.isfile('data/after_eda/test_after_eda.csv')) and \
(not os.path.isfile('data/train_y.csv')) and \
(not os.path.isfile('data/test_y.csv')) and \
(os.path.isfile('data/after_eda/train_pos_after_eda.csv')) and \
(os.path.isfile('data/after_eda/test_pos_after_eda.csv')) and \
(os.path.isfile('data/after_eda/train_neg_after_eda.csv')) and \
(os.path.isfile('data/after_eda/test_neg_after_eda.csv')):

    X_train_pos = pd.read_csv('data/after_eda/train_pos_after_eda.csv', names=['source_node', 'destinatic
    X_test_pos = pd.read_csv('data/after_eda/test_pos_after_eda.csv', names=['source_node', 'destination_
    X_train_neg = pd.read_csv('data/after_eda/train_neg_after_eda.csv', names=['source_node', 'destinatic
    X_test_neg = pd.read_csv('data/after_eda/test_neg_after_eda.csv', names=['source_node', 'destination_

    print('='*60)
    print("Number of nodes in the train data graph with edges", X_train_pos.shape[0])
    print("Number of nodes in the train data graph without edges", X_train_neg.shape[0])
    print('='*60)
    print("Number of nodes in the test data graph with edges", X_test_pos.shape[0])
    print("Number of nodes in the test data graph without edges", X_test_neg.shape[0])

    X_train = X_train_pos.append(X_train_neg,ignore_index=True)
    y_train = np.concatenate((y_train_pos,y_train_neg))
    X_test = X_test_pos.append(X_test_neg,ignore_index=True)
    y_test = np.concatenate((y_test_pos,y_test_neg))
```

```
X_train.to_csv('data/after_eda/train_after_eda.csv',header=False,index=False)
X_test.to_csv('data/after_eda/test_after_eda.csv',header=False,index=False)
pd.DataFrame(y_train.astype(int)).to_csv('data/train_y.csv',header=False,index=False)
pd.DataFrame(y_test.astype(int)).to_csv('data/test_y.csv',header=False,index=False)
```

In [35]:

```
X_train = pd.read_csv('data/after_eda/train_after_eda.csv', names=['source_node', 'destination_node'])
X_test = pd.read_csv('data/after_eda/test_after_eda.csv', names=['source_node', 'destination_node'])
y_train = pd.read_csv('data/train_y.csv', names = ['indicator variable'])
y_test = pd.read_csv('data/test_y.csv', names = ['indicator variable'])

print("Data points in train data",X_train.shape)
print("Data points in test data",X_test.shape)
print("Shape of traget variable in train",y_train.shape)
print("Shape of traget variable in test", y_test.shape)
```

```
Data points in train data (15100030, 2)
Data points in test data (3775008, 2)
Shape of traget variable in train (15100030, 1)
Shape of traget variable in test (3775008, 1)
```

In [36]:

```
# computed and store the data for featurization
# please check out FB_featurization.ipynb
```