Creating the Spark Session and importing the libraries.
Note that we need the libraries developed for this proyect which are located under the utils folder.
That is why we add the line "..addPyFile.."
Also there are some features for the Jupyter notebook execution and visualization (like %autoreload and retina display)

In [9]:
```
import pandas as pd
import sys
from pyspark.sql import SparkSession

sys.path.insert(0,"../utils.zip")
from utils.spark_utils import spark_session

spark = spark_session()
#spark = SparkSession.builder.getOrCreate()

%load_ext autoreload
%autoreload 2

from utils.gf_utils import *
from utils.df_utils import *
from utils.draw_utils import *
from utils.spark_utils import *
from utils.read_write_utils import *
from pyspark.sql import functions as F
%config InlineBackend.figure_format = 'retina'
spark.sparkContext.addPyFile("../utils.zip")
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Here is our session, from where we can observe the cluster's execution of each operation (evaluation or transformation) in spark.

In [10]:
```
spark
```

Out[10]:
SparkSession - in-memory
SparkContext

Spark UI (http://t[redacted])
Version
v2.4.0
Master
local[12]
AppName
pyspark-shell

Size of the data to experiment with :

```
101G    180205
101G    180206
100G    180207
99G     180208  ->> Data for this notebook ALL_DAY_DATA
```

In the next line we define an spark dataframe : df, generated from the data of the last generated 5 files on 08-02-2018

In [16]:
```
#df=spark.read.csv("file:///s[redacted]49[0-4]*",header=True)
df=spark.read.csv("file:///s[redacted]4*",header=True)
```

In [ ]:
```
```

In [3]:
```
#df= (spark.read.csv("file:///s[redacted]*",
#                           header="true", timestampFormat="yyyy-MM-dd HH:mm:ss", escape='"',
#                           ignoreLeadingWhiteSpace="true", ignoreTrailingWhiteSpace="true",
#                           mode="FAILFAST").select("user_ip","referrer_domain"))
```

The raw data contais null or not normalized fields, so the next step is to clean the dataset.

In [10]:
```
df_cleaned = clean( df,"referrer_domain","user_ip" )
```

```
df_utils clean --
```

Generating the graphframe domain-ip, with a filter where the nodes with less than 15 visits are discarded.

def get_graph_domip(df, min_edge):

```
    Get GraphFrame to draw a bipartite graph
    :param df dataframe from our data. Idem format like in get_vertices function.
    :param min_edge: value to dismiss all the nodes on g below the min_edge value
                    (if for a src - dst tuple : edge_weight <  min_edge this row is discarded)
    :return: gf_filtered (GraphFrame graph) filtered by min_edge

    :definition df_vertices: vertices for the graphframe : domains and ips
    :definition df_edges: links between them
```

In [11]:
```
#gf_domip = get_graph_domip(df_cleaned, 15)
```

```
gf_utils get_graph_domip-- :
df_utils get_edges_domip-- : df
df_utils get_edges --
df_utils get_vertices-- :
gf_utils filter_gf filterEdges : {min_edge}
```

def get_df_degree_ratio(g_domip):

```
    Function to get the df_degree_ratio, with the format described below.
    :param g_domip:
    :return df_degree_ratio : dataframe with all the data needed to represent the overlap matrix
    [a,c,count_ips_in_common,id,outDegree,edge_ratio]
    where a is src, c is dst, id is src, and outDegree is the outDegree of src.
    The edge_ratio is calculated with the algorithm proposed.
```

In [12]:
```
#df_degree_ratio=get_df_degree_ratio(gf_domip)
```

```
gf_utils get_motifs -- df_motifs dropDuplicates( ['e', 'e2']
gf_utils get_motifs_count -- df_motifs_count
gf_utils get_df_degree_ratio --  df_degreeRatio :
```

In [7]:
```
#df_degree_ratio.printSchema()
```

```
root
 |-- a: struct (nullable = false)
 |    |-- id: string (nullable = true)
 |-- c: struct (nullable = false)
 |    |-- id: string (nullable = true)
 |-- count_ips_in_common: long (nullable = false)
 |-- id: string (nullable = true)
 |-- outDegree: integer (nullable = false)
 |-- edge_ratio: double (nullable = true)
```

Saving the data to easy and quick consult it later.

In [9]:
```
#df_write_parquet(df_degree_ratio,"file:///s[redacted])
```

```
Salvado parquet df en el path :file:///s[redacted]t/df_degree_ratio_180208
```

Reading the df_degree_ratio saved before. Because of the lazy spark execution, we often read a previous saved graph, in order to get in memory the data that we are needed to use.

In [28]:
```
df_degree_ratio=df_read_parquet(spark,"file:///s[redacted]")
```

```
Readed parquet df from path :file:///s[redacted]/df_degree_ratio_180208_4files
```

Getting the co-visitation domain domain graph :

def get_graph_domdom(g_domip):

```
    Get GraphFrame to draw graph
    :param g_domip: graph from domain_ip_graph .
    :return: gf (GraphFrame graph) domain-domain
    :definition df_vertices: vertices for the graphframe : domains
    :definition df_edges: links between them
```

In [10]:
```
#gf_domdom = get_graph_domdom( gf_domip )
```

```
DomainDomainGraph get_graph_domdom -- g_domip.edges.show()
gf_utils get_motifs -- df_motifs dropDuplicates( ['e', 'e2']
gf_utils get_motifs_count -- df_motifs_count
gf_utils get_df_degree_ratio --  df_degreeRatio :
gf_utils get_df_degree_ratio --  df_degreeRatio division  (edge_ratio = covisitation degree:
df_utils get_edges_domdom -- df que llega ...
df_utils get_edges --
df_utils get_edges_domdom -- df_edges calculado ...
+--------------------+--------------------+--------------------+
|                 src|                 dst|         edge_weight|
+--------------------+--------------------+--------------------+
|           [p[illegible]|  [t[illegible]|  0.9807692307503699|
|   [p[illegible]|[c[illegible]|   0.342857142847347|
|   [w[illegible]|[a[illegible]|  0.4336283185802334|
|[f[illegible]|[f[illegible]| 0.04601226993850917|
|[k[illegible]|[j[illegible]| 0.21556886227415828|
|[[illegible]|[[illegible]| 0.07407407407270233|
|[[illegible]|[[illegible]| 0.32876712328316754|
|[[illegible]|[[illegible]|0.028169014084110294|
|[[illegible]|[[illegible]|  0.8888888887901234|
|[[illegible]|[[illegible]|  0.2272727272623967|
|[[illegible]|[[illegible]|  0.9999999989999999|
|[[illegible]|[[illegible]|      0.999999999875|
|[[illegible]|[[illegible]|  0.9999999996666666|
|[[illegible]|[[illegible]|  0.9999999989999999|
|[[illegible]|[[illegible]| 0.07142857142602041|
|[[illegible]|[[illegible]| 0.07142857142346938|
|[[illegible]|[[illegible]| 0.00934579439243602|
|[[illegible]|[[illegible]|0.012345679012193262|
|[[illegible]|[[illegible]|     0.0399999999984|
|[[illegible]|[[illegible]|       0.19999999996|
+--------------------+--------------------+--------------------+
only showing top 20 rows

df_utils get_vertices-- :
df_utils get_vertices-- df_vertices_withduplicates :
df_utils get_vertices-- df_vertices_sin duplicates :
DomainDomainGraph get_graph_domdom --  df_edges OLAYAs :
+--------------------+--------------------+--------------------+
|                 src|                 dst|         edge_weight|
+--------------------+--------------------+--------------------+
|[[illegible]|[[illegible]|  0.9807692307503699|
|[[illegible]|[[illegible]|   0.342857142847347|
|[[illegible]|[[illegible]|  0.4336283185802334|
|[[illegible]|[[illegible]| 0.04601226993850917|
|[[illegible]|[[illegible]| 0.21556886227415828|
|[[illegible]|[[illegible]| 0.07407407407270233|
|[[illegible]|[[illegible]| 0.32876712328316754|
|[[illegible]|[[illegible]|0.028169014084110294|
|[[illegible]|[[illegible]|  0.8888888887901234|
|[[illegible]|[[illegible]|  0.2272727272623967|
|[[illegible]|[[illegible]|  0.9999999989999999|
|[[illegible]|[[illegible]|      0.999999999875|
|[[illegible]|[[illegible]|  0.9999999996666666|
|[[illegible]|[[illegible]|  0.9999999989999999|
|[[illegible]|[[illegible]| 0.07142857142602041|
|[[illegible]|[[illegible]| 0.07142857142346938|
|[[illegible]|[[illegible]| 0.00934579439243602|
|[[illegible]|[[illegible]|0.012345679012193262|
|[[illegible]|[[illegible]|     0.0399999999984|
|[[illegible]|[[illegible]|       0.19999999996|
+--------------------+--------------------+--------------------+
only showing top 20 rows
```

Saving the graph gf_domip calculated before into disk.

In [7]:
```
#gf_write_parquet(gf_domip,"file:///s[illegible]/output/gf_domip_180208_4files")
```

```
 Salvado parquet grafo en el path :file:///[illegible]a/output/gf_domip_180208_4files
```

Reading the graph saved gf_domip

In [14]:
```
gf_domip=gf_read_parquet(spark,"file:///s[illegible]/output/gf_domip_180208_4files")
```

```
 Readed parquet graph from path :file:///[illegible]a/output/gf_domip_180208_4files
```

Saving the graph gf_domdom calculated before into disk.

In [11]:
```
#gf_write_parquet(gf_domdom,"file:///s[illegible]/output/gf_domdom_180208_4files")
```

```
 Salvado parquet grafo en el path :file:///s[illegible]/output/gf_domdom_180208_4files
```

Reading from disk the graph gf_domdom, saved before.

In [7]:
```
gf_domdom=gf_read_parquet(spark,"file:///s[illegible]a/output/gf_domdom_180208_4files")
```

```
 Readed parquet graph from path :file:///[illegible]a/output/gf_domdom_180208_4files
```

The following functions are defined in the class 'read_write_utils.py', and as the name of the function indicates, are used to save the graphs generated before.

In [ ]:
```
#def gf_write_csv(gf, path):
#    gf.edges.write.csv( f"{path}/{EDGES_WRITED}" )
#    gf.vertices.write.csv( f"{path}/{VERTICES_WRITED}"  )
#    print( f" Salvado CSV grafo en el path :{path}" )
```

In [30]:
```
#def gf_write_csv_new(gf, path):
#    gf.edges.select(F.col("src.id").alias("src")
#                    ,F.col("dst.id").alias("dst"),F.col("edge_weight")).write.csv( f"{path}/{EDGES_WRITED}" )
#    gf.vertices.select(F.col("id.id").alias("id")).write.csv( f"{path}/{VERTICES_WRITED}"  )
#    print( f" Salvado CSV grafo en el path :{path}" )
```

In [52]:
```
#gf_domdom.vertices.select(F.col("id.id").alias("id"))
```

Out[52]:
```
DataFrame[id: string]
```

In [19]:
```
#gf_domdom.edges.select(F.col("src.id").alias("src"),F.col("dst.id").alias("dst"),F.col("edge_weight"))
```

Out[19]:
```
DataFrame[src: string]
```

In [ ]:
```
#gf_write_csv_new(gf_domdom,"file:///[redacted]/output/gf_domdom_180208")
```

Visualizing and saving the ploted graphs into disk .

In [8]:
```
#ig, visual_style = draw_igraph_domain_domain( gf_domdom )
```

In [9]:
```
#plot( ig,**visual_style ).save("file:///[redacted]/output/gf_domdom_180208.png")
```

In [ ]:
```
#ig, visual_style = draw_igraph_domain_ip( gf_domip)
```

In [ ]:
```
#plot( ig,**visual_style ).save("file:///[redacted]/output/gf_domip_180208.png")
```

In [ ]:
```
#draw_nx(gf_domip.edges,"file:///[redacted]/output/gf_domip_nx.png")
```

In [13]:
```
ig, visual_style = draw_igraph_domain_domain( gf_domdom )

plot( ig, **visual_style )#.save(
        #"file:///[redacted]/output/gf_domdom_180208_4files_weighted.png" )
```

```
18180165288, 0.03030303030211203, 0.1333333333288889, 0.19999999996, 0.124999999984375, 0.021739130434546314, 0.19999999996, 0.666666666444444
4, 0.9999999989999999, 0.9999999989999999, 0.7241379310261593, 0.14371257484943886, 0.19999999996, 0.0344827586202933, 0.39999999997333335, 0.
3699999999963, 0.05882352941127039, 0.037037037035635117, 0.6666666664444444, 0.39999999992, 0.04545454545247934, 0.39999999992, 0.999999998999
9999, 0.027027027026296568, 0.9999999989999999, 0.018181818181487605, 0.3333333332222222, 0.19999999996, 0.6969696969485768, 0.142857142855782
3, 0.009345794392436602, 0.3090909090852893, 0.18181818180991735, 0.08333333333101853, 0.9999999989999999, 0.2727272727024793, 0.07079646017636
464, 0.9999999995, 0.9999999989999999, 0.9999999996666666, 0.03738317756974408, 0.005988023952059952, 0.49999999975, 0.9999999989999999, 0.014
705882352724913, 0.9454545454373554, 0.40540540053944485, 0.99999999975, 0.19999999996, 0.2528735632154842, 0.14012738853413934, 0.49999999995,
0.9999999989999999, 0.047619047618594104, 0.01652892561969811, 0.06666666666222222, 0.9999999995, 0.9999999996666666, 0.22222222219753085, 0.9
9999999995, 0.010204081632601, 0.3333333332222222, 0.9999999996666666, 0.7599999999924, 0.6029411764617214, 0.9166666666412038, 0.9259259259087
792, 0.02298850574699432, 0.10476190476090702, 0.9999999998, 0.19999999999555557, 0.9999999989999999, 0.9999999998, 0.3333333332222222, 0.3999
9999992, 0.3333333332222222, 0.9999999996666666, 0.9999999989999999, 0.6301369862927378, 0.7356321838995904, 0.499999999995, 0.238095238091458
83, 0.09782608695545841, 0.9999999999090909, 0.74999999990625, 0.03703703703635117, 0.7333333332844444, 0.015974440894517654, 0.02803738317730
806, 0.01724137931024574, 0.010204081632601, 0.08108108107888971, 0.9999999989999999, 0.027272727272479338, 0.9999999989999999, 0.027777777777
006175, 0.035714285713010205, 0.45454545450413225, 0.6909090909028099, 0.28999999999709997, 0.699999999993, 0.34146341462998214, 0.91549295773
35845, 0.23076923076479292, 0.8799999999648, 0.06389776357807062, 0.9999999999642857, 0.08383233532883934, 0.20588235293814877, 0.9999999995,
0.012195121951070791, 0.9999999989999999, 0.499999999875, 0.035714285713010205, 0.3333333332222222, 0.045454545545247934, 0.39999999996, 0.2499
999999375, 0.024691358024386524, 0.9999999989999999, 0.9999999999642857, 0.3461538461519442, 0.5172413793043995, 0.8181818181446281, 0.6599999
999934, 0.17647058823183392, 0.27731092436741756, 0.208791208788914 38, 0.027472527472376527, 0.0229885057468622, 0.9999999989999999, 0.0190476
19047437642, 0.07843137254748174, 0.49999999975, 0.005988023952059952, 0.13333333332444444, 0.0114942528734311, 0.999999999984127, 0.639999999
9936, 0.999999999875, 0.7254901960642062, 0.12389380530863811, 0.09523809523658353, 0.1111111111090535, 0.99999999975, 0.624999999921875, 0.05
```

In [15]:
```
#plot( ig, **visual_style ).save(
#        "file:///[redacted]/output/gf_domdom_180208_4files_weighted.png" )
```

Searching for the degrees and edge_ratio for the domain [redacted]com", we suspect that it could be a malicious domain.

In [8]:
```
df_degree_ratio.filter("id = '[redacted].com'").show(20)
```

```
+--------------------+--------------------+------------------+--------------+---------+--------------------+
|                   a|                   c|count_ips_in_common|            id|outDegree|          edge_ratio|
+--------------------+--------------------+------------------+--------------+---------+--------------------+
|[[redacted].com]|[[redacted]]|                20|[redacted].com|      206|0.09708737864030541|
|[[redacted].com]|[[redacted]com]|                39|[redacted].com|      206|0.18932038834859555|
|[[redacted].com]|[[redacted]com]|                38|[redacted].com|      206|0.18446601941658028|
|[[redacted].com]|[[redacted]uk]|                 6|[redacted].com|      206|0.02912621359209162|
|[[redacted].com]|[[redacted].com]|                38|[redacted].com|      206|0.18446601941658028|
|[[redacted]|[[redacted]|                70|[redacted]|      206|0.33980582524106895|
|[[redacted]|[[redacted]|                34|[redacted]|      206|0.16504854368851918|
|[[redacted]|[[redacted]|                12|[redacted]|      206|0.05825242718418324|
|[[redacted]|[[redacted]|                 1|[redacted]|      206|0.00485436893201527|
|[[redacted]|[[redacted]|                41|[redacted]|      206|0.19902912621262608|
|[[redacted]|[[redacted]|               118|[redacted]|      206| 0.5728155339778019|
|[[redacted]|[[redacted]|                 4|[redacted]|      206|0.01941747572806108|
|[[redacted]|[[redacted]|                28|[redacted]|      206|0.13592233009642757|
|[[redacted]|[[redacted]|               160|[redacted]|      206| 0.7766990291224433|
|[[redacted]|[[redacted]|                 4|[redacted]|      206|0.01941747572806108|
|[[redacted]|[[redacted]|                 3|[redacted]|      206|0.01456310679604581|
|[[redacted]|[[redacted]|                 1|[redacted]|      206|0.00485436893201527|
|[[redacted]|[[redacted]|                17|[redacted]|      206|0.08252427184425959|
|[[redacted]|[[redacted]|                62|[redacted]|      206|0.30097087378494675|
|[[redacted]|[[redacted]|                 3|[redacted]|      206|0.01456310679604581|
+--------------------+--------------------+------------------+--------------+---------+--------------------+
only showing top 20 rows
```

Looking for a domain that is considerer worthy

In [9]:
```
df_degree_ratio.filter("id = '[redacted].com'").show(20)
```

```
+----------+--------------------+------------------+---------+---------+--------------------+
|         a|                   c|count_ips_in_common|       id|outDegree|          edge_ratio|
+----------+--------------------+------------------+---------+---------+--------------------+
|[[redacted].com]|  [[redacted].com]|                38|[redacted].com|      456| 0.08333333333315059|
|[[redacted].com]|          [[redacted].com]|                10|[redacted].com|      456|0.021929824561355418|
|[redacted]|[redacted]|                 3|[redacted]|      456|0.006578947368406...|
|[redacted]|[redacted]|                 3|[redacted]|      456|0.006578947368406...|
|[redacted]|[redacted]|                34|[redacted]|      456| 0.07456140350860842|
|[redacted]|[redacted]|                 2|[redacted]|      456|0.004385964912271083|
|[redacted]|[redacted]|                 5|[redacted]|      456|0.010964912280677709|
|[redacted]|[redacted]|                 1|[redacted]|      456|0.002192982456135...|
|[redacted]|[redacted]|                 7|[redacted]|      456|0.015350877192948793|
|[redacted]|[redacted]|                 1|[redacted]|      456|0.002192982456135...|
|[redacted]|[redacted]|                25|[redacted]|      456|0.054824561403388546|
|[redacted]|[redacted]|                 1|[redacted]|      456|0.002192982456135...|
|[redacted]|[redacted]|                 2|[redacted]|      456|0.004385964912271083|
|[redacted]|[redacted]|                 1|[redacted]|      456|0.002192982456135...|
|[redacted]|[redacted]|                 1|[redacted]|      456|0.002192982456135...|
|[redacted]|[redacted]|                 1|[redacted]|      456|0.002192982456135...|
|[redacted]|[redacted]|                 2|[redacted]|      456|0.004385964912271083|
|[redacted]|[redacted]|                 1|[redacted]|      456|0.002192982456135...|
|[redacted]|[redacted]|                 2|[redacted]|      456|0.004385964912271083|
|[redacted]|[redacted]|                 4|[redacted]|      456|0.008771929824542167|
+----------+--------------------+------------------+---------+---------+--------------------+
only showing top 20 rows
```

The following function : gf_top_most_visited, defined in gf_utils.py class, show the top 10 domains most visited from the dataset selected.

In [10]:
```
print( "plots_main MAIN-- calculando doms_more_visited : show todo.." )
sorted_degrees = gf_top_most_visited(gf_domip,10)
sorted_degrees.show()
```

```
plots_main MAIN-- calculando doms_more_visited : show todo..
 type  <class 'pyspark.sql.dataframe.DataFrame'>
gf_utils gf_top_most_visited -- sorted_degrees.show ..
+--------------------+------+
|                  id|degree|
+--------------------+------+
|  [redacted].com|   344|
|[redacted]...|   326|
|          [redacted].com|   313|
|  [redacted].com|   266|
|[redacted]...|   258|
|          [redacted]e|   236|
|[redacted]...|   218|
|    [redacted]|   213|
|     [redacted].com|   196|
|   [redacted]d|   190|
+--------------------+------+
```

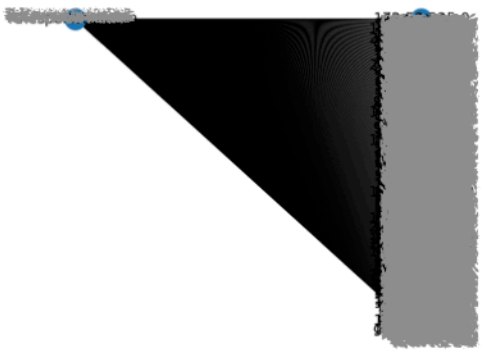
The following line generates a subgraph from the gf_domip with the edges of the suspicious domain '[redacted].com'. All the IPS that visit this domain will be presented.

In [11]:
```
gf_domip_filtered_bad_domain=gf_domip.filterEdges("src='[redacted].com'")
```

The plot for this selection is not so usefull ...

In [21]:
```
draw_nx( gf_domip_filtered_bad_domain.edges )
```

```
draw_utils draw_nx --
draw_utils draw_nx -- -- despues nx.Graph()
draw_utils draw_nx -- -- despues add_nodes_from src
draw_utils draw_nx -- -- despues add_nodes_from dst
draw_utils draw_nx -- -- Nodes added to B
draw_utils draw_nx -- -- despues de pos.update
draw_utils draw_nx -- -- despues for
draw_utils draw_nx -- -- ante de plot
```



The following lines, help us to define later the functions : draw_log_hist and draw_minor_than_list contained in the class : draw_utils.py .

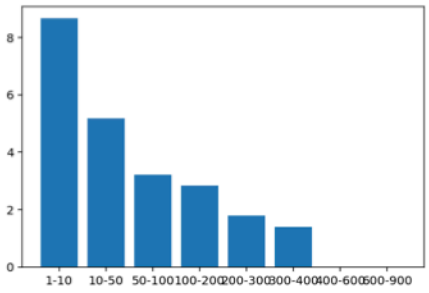
log histogram

def draw_log_hist(degree, bins=10,path=None):

```
    '''
    Function to draw a histogram in logaritmic scale.

    :param degree : node degree
    :param bins   : division of the histogram, dos methods
    :param path   : path where to save the histogram image
    :return : ploted bar histogram

    draw_log_hist(degree,10):
    returns 10 bars with division made by np.histogram

    draw_log_hist(degree,[1,10,100,200]):
    returns plot between the numbers passed as a parameter,
    it means that sums the number of elements between 1-10, 10-100,100-200 ....

    LOGARITHMIC SCALE

    '''
```

In [28]:
```
print( "plots_main MAIN-- Calculando primer histograma --  draw_log_hist -- ..." )
%matplotlib inline
total_degrees = gf_domip.degrees
print( f" type  {type( total_degrees )}  " )
sorted_degrees = total_degrees.orderBy( F.desc( "degree" ) )

degree, id = zip(*[(item.degree, item.id) for item in sorted_degrees.select( "degree", "id" ).collect()] )

draw_log_hist( degree, [1, 10, 50, 100, 200, 300, 400,600,900])#,"file:///s[illegible]/output/log_hist_plot.png" )
```

```
plots_main MAIN-- Calculando primer histograma --  draw_log_hist -- ...
 type  <class 'pyspark.sql.dataframe.DataFrame'>
draw_utils draw_log_hist -- --
```



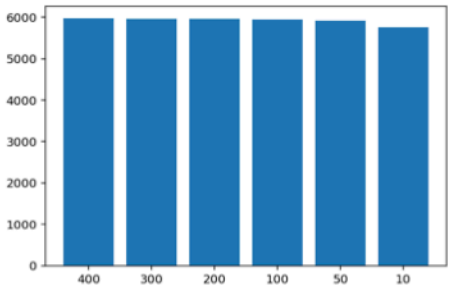
minor than a tope histogram

def draw_minor_than_list(degree, list_tope,path=None):

```
    '''
    Function to represent the elements that are minor than a maximum (tope),
    how many are minor than 400, minor than 300, minor than 200 ....
    :param degree    : degree a pintar
    :param list_tope: integer array with the maximums
    :param path      : path where to save the histogram image
    :return ploted bar histogram
    '''
```

In [30]:
```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_minor_than_list -- ..." )


list_tope = [400, 300, 200, 100, 50, 10]
draw_minor_than_list( degree, list_tope)#"file:///[illegible]a/output/minor_than_list_plot.png" )
```

```
plots_main MAIN-- Calculando segundo histograma --  draw_minor_than_list -- ...
draw_utils draw_minor_than_list -- --
```



Overlap Matrix , co-visitation matrix

The following lines are for calculate a heat map or overlap matrix. In this matrix, the domains have been represented in the 'x' and 'y' coordinates; and the co-visitation degrees between them can be seen by different colors. When higher and to the left of the matrix, we see a biggerdegree of overlap between domains; which means that the probability that the domain could be fraudulent is greater.

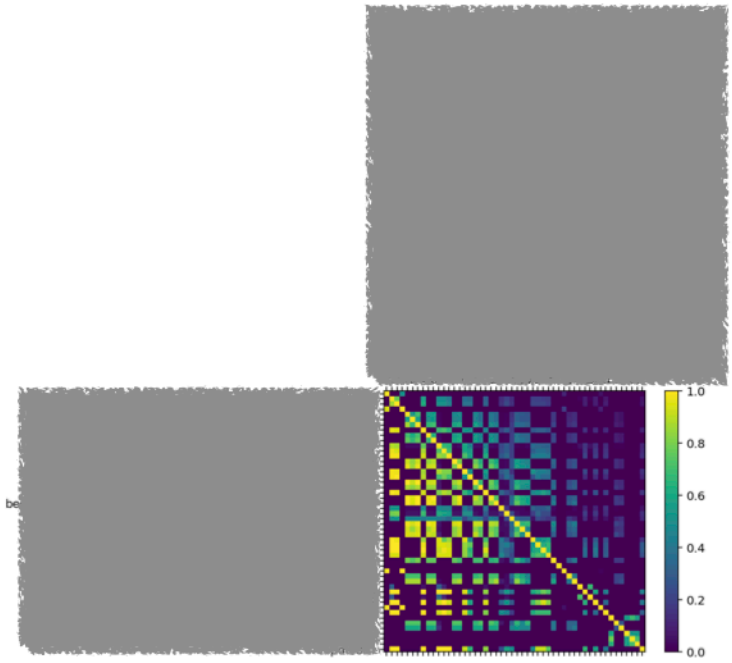
def draw_overlap_matrix(df_degree_ratio, list_top_suspicious,figsize=(10,10),path=None):

```
    '''
    Function to draw an overlap matrix of suspicious domains
    :param df_degree_ratio : dataframe with all the data needed to represent the overlap matrix [a,c,count_ips_in_common,id,outDegree,edge_r
    atio]
                             where a is src, c is dst, id is src, and outDegree is the outDegree of src. The edge_ratio is calculated with t
    he
                            algorithm proposed.
    :param top_suspicious : number of top suspicious domains to plot
    :param figsize
    :param path : path where to save the histogram image
    :return ploted overlap matrix
    '''
```

In [31]:

```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ..." )
top50_suspicious = df_degree_ratio.filter(
    "edge_ratio>0.5 and count_ips_in_common>1  " ).select(
    df_degree_ratio.a.id, df_degree_ratio.outDegree ).distinct().sort( F.desc( "outDegree" ) ).take( 50 )
list_top_suspicious = [row["a.id"] for row in top50_suspicious]
draw_overlap_matrix( df_degree_ratio, list_top_suspicious)#,"/[redacted]/output_fraud/overlap_list_top_suspicious.pdf" )
```

```
plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ...
draw_utils draw_overlap_matrix -- --
```



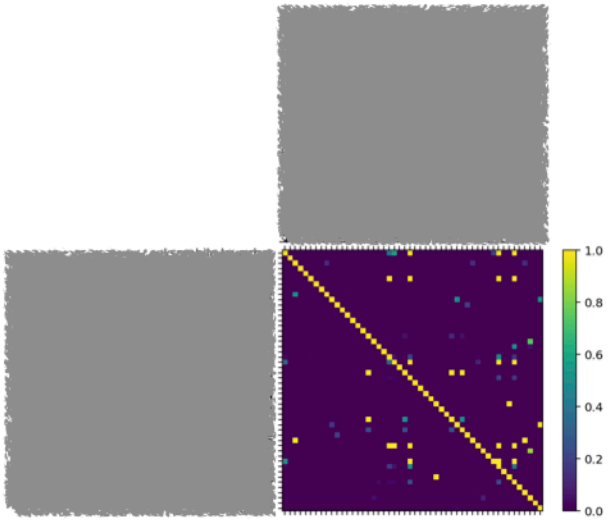
Overlap matrix 20% random data selected

In order to compare, we have made another similar matrix,taking a random sample of domains from the dataset selected.

In [36]:

```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_sample ..." )
sample_50_percent = df_degree_ratio.select( F.col( "a.id" ) ).distinct().sample( 0.2 ).take( 50 )
# cojo un 20% de datos totales (dominios unicos) de la muestra de manera aleatoria
list_sample = [row["id"] for row in sample_50_percent]
draw_overlap_matrix( df_degree_ratio, list_sample)#"/[redacted]M/output_fraud/overlap_list_sample.pdf" )
```

```
plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_sample ...
draw_utils draw_overlap_matrix -- --
```



considered suspicious domain

In [62]:
```
df_degree_ratio=df_read_parquet(spark,"file:///[redacted]/output/df_degree_ratio_180208_ALLDAY_new")
```

```
 Readed parquet df from path :file:///[redacted]/output/df_degree_ratio_180208_ALLDAY_new
```

In [61]:
```
df_degree_ratio=df_read_parquet(spark,"file:///[redacted]/output/df_degree_ratio_180208_4files")
```

```
 Readed parquet df from path :file:///[redacted]/output/df_degree_ratio_180208_4files
```

In [71]:
```
df_degree_ratio.filter("id = 'my[redacted].m' ").select("c","edge_ratio","count_ips_in_common", "outDegree").sort(F.desc("edge_ratio")).show(20,Fa
```

```
+--------------------------------------------------------------+-------------------+-------------------+---------+
|c                                                             |edge_ratio         |count_ips_in_common|outDegree|
+--------------------------------------------------------------+-------------------+-------------------+---------+
|[[redacted]]                                                  |0.2643764633091686 |37602              |142229   |
|[[redacted]]                                                  |0.240647125410428  |34227              |142229   |
|[[redacted]]                                                  |0.22844848800174208|32492              |142229   |
|[[redacted]]                                                  |0.21441478179555357|30496              |142229   |
|[[redacted]]                                                  |0.17352298054545717|24680              |142229   |
|[[redacted]]                                                  |0.17241209598604945|24522              |142229   |
|[[redacted]]                                                  |0.17108325306372   |24333              |142229   |
|[[redacted]]                                                  |0.1707106145722731 |24280              |142229   |
|[[redacted]]                                                  |0.169845812035519  |24157              |142229   |
|[[redacted]]                                                  |0.16766622840630135|23847              |142229   |
|[[redacted]]                                                  |0.16501557347657536|23470              |142229   |
|[[redacted]]                                                  |0.1509326508658561 |21467              |142229   |
|[[redacted]]                                                  |0.14848589246918598|21119              |142229   |
|[[redacted]]                                                  |0.14711486405725874|20924              |142229   |
|[[redacted]]                                                  |0.1456383719213371 |20714              |142229   |
|[[redacted]]                                                  |0.14377517946410268|20449              |142229   |
|[[redacted]]                                                  |0.13988005259124273|19895              |142229   |
|[[redacted]]                                                  |0.13569665820613142|19300              |142229   |
|[[redacted]]                                                  |0.1347967010947125 |19172              |142229   |
|[[redacted]]                                                  |0.13235697361297533|18825              |142229   |
+--------------------------------------------------------------+-------------------+-------------------+---------+
only showing top 20 rows
```

In [54]:
```
df_degree_ratio
```

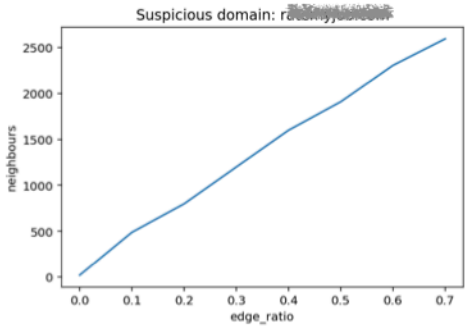
Out[54]:
```
DataFrame[a: struct<id:string>, c: struct<id:string>, count_ips_in_common: bigint, id: string, outDegree: int, edge_ratio: double]
```

In [72]:
```
df_degree_ratio.filter("id = '[redacted].com' ").select("c","edge_ratio","count_ips_in_common", "outDegree").sort(F.desc("edge_ratio")).show(20,F
```

```
+------------------+-------------------+-------------------+---------+
|c                 |edge_ratio         |count_ips_in_common|outDegree|
+------------------+-------------------+-------------------+---------+
|[[redacted]]      |0.743832472747922  |2593               |3486     |
|[[redacted]]      |0.673838209982595  |2349               |3486     |
|[[redacted]]      |0.6689615605276337 |2332               |3486     |
|[[redacted]]      |0.640849110728445  |2234               |3486     |
|[[redacted]]      |0.5955249569705693 |2076               |3486     |
|[[redacted]]      |0.5831899024668437 |2033               |3486     |
|[[redacted]]      |0.5780263912792375 |2015               |3486     |
|[[redacted]]      |0.5679862306366701 |1980               |3486     |
|[[redacted]]      |0.5676993689040253 |1979               |3486     |
|[[redacted]]      |0.5636833046469983 |1965               |3486     |
|[[redacted]]      |0.5490533562821145 |1914               |3486     |
|[[redacted]]      |0.5438898450945083 |1896               |3486     |
|[[redacted]]      |0.540447504302771  |1884               |3486     |
|[[redacted]]      |0.5315547905907827 |1853               |3486     |
|[[redacted]]      |0.5255306942052422 |1832               |3486     |
|[[redacted]]      |0.5083189902465552 |1772               |3486     |
|[[redacted]]      |0.5071715433159761 |1768               |3486     |
|[[redacted]]      |0.5017211703957253 |1749               |3486     |
|[[redacted]]      |0.49741824440605353|1734               |3486     |
|[[redacted]]      |0.4956970740101848 |1728               |3486     |
+------------------+-------------------+-------------------+---------+
only showing top 20 rows
```

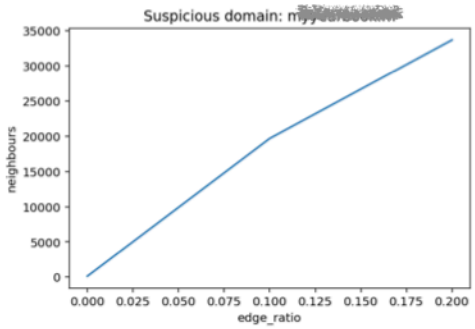
In [74]:
```
from scipy import stats

cnt_edge =  df_degree_ratio.filter("id = '[redacted]' ").select("count_ips_in_common","edge_ratio").sort("edge_ratio").collect()

edge_ratio,cnt_ip =zip(*[(row.edge_ratio,row.count_ips_in_common) for row in  cnt_edge])

res_5=stats.binned_statistic(edge_ratio,cnt_ip,statistic='mean',bins=np.arange(0,1.0,0.1))

plt.xlabel('edge_ratio')
plt.ylabel('neighbours')
plt.title('Suspicious domain: [redacted]')


plt.plot(res_5.bin_edges[:-1],res_5.statistic)
```

Out[74]:
```
[<matplotlib.lines.Line2D at 0x7f713bd10b00>]
```



In [67]:
```
from scipy import stats

cnt_edge =  df_degree_ratio.filter("id = '[redacted]' ").select("count_ips_in_common","edge_ratio").sort("edge_ratio").collect()

edge_ratio,cnt_ip =zip(*[(row.edge_ratio,row.count_ips_in_common) for row in  cnt_edge])

res_5=stats.binned_statistic(edge_ratio,cnt_ip,statistic='mean',bins=np.arange(0,1.0,0.1))

plt.xlabel('edge_ratio')
plt.ylabel('neighbours')
plt.title('Suspicious domain: [redacted]')


plt.plot(res_5.bin_edges[:-1],res_5.statistic)
```

Out[67]:
```
[<matplotlib.lines.Line2D at 0x7f713ef3c358>]
```



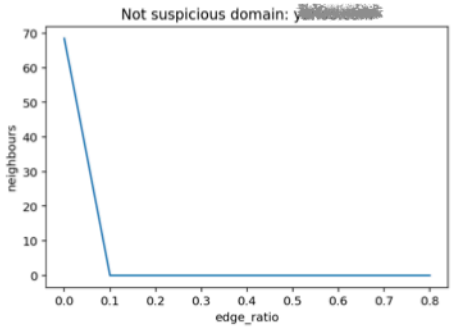
Considered legitim domain

In [73]:
```
df_degree_ratio.filter("id = '[redacted]' ").select("c","edge_ratio","count_ips_in_common", "outDegree").sort(F.desc("edge_ratio")).show(20,False
```

```
+------------------------+--------------------+-------------------+---------+
|c                       |edge_ratio          |count_ips_in_common|outDegree|
+------------------------+--------------------+-------------------+---------+
|[[redacted]]            |0.04518765504296389 |19163              |424076   |
|[[redacted]]            |0.02948763900810225 |12505              |424076   |
|[[redacted]]            |0.0283015308576763 87|12002              |424076   |
|[[redacted]]            |0.027971401352587678|11862              |424076   |
|[[redacted]]            |0.027190880879842227|11531              |424076   |
|[[redacted]]            |0.02590809194578324 |10987              |424076   |
|[[redacted]]            |0.022630377573831052|9597               |424076   |
|[[redacted]]            |0.022599722691215674|9584               |424076   |
|[[redacted]]            |0.017032324394683933|7223               |424076   |
|[[redacted]]            |0.016881408049500522|7159               |424076   |
|[[redacted]]            |0.01679415953744136 |7122               |424076   |
|[[redacted]]            |0.016291891076127826|6909               |424076   |
|[[redacted]]            |0.016112677916222527|6833               |424076   |
|[[redacted]]            |0.01600656486101544 |6788               |424076   |
|[[redacted]]            |0.015414689819749255|6537               |424076   |
|[[redacted]]            |0.014587007989133989|6186               |424076   |
|[[redacted]]            |0.014388930286080763|6102               |424076   |
|[[redacted]]            |0.014334694724530476|6079               |424076   |
|[[redacted]]            |0.013433912789216997|5697               |424076   |
|[[redacted]]            |0.013318367462435947|5648               |424076   |
+------------------------+--------------------+-------------------+---------+
only showing top 20 rows
```

In [70]:
```
cnt_edge =  df_degree_ratio.filter("id = 'y[redacted]m' ").select("count_ips_in_common","edge_ratio").sort("edge_ratio").collect()


edge_ratio,cnt_ip =zip(*[(row.edge_ratio,row.count_ips_in_common) for row in  cnt_edge])


res_7=stats.binned_statistic(edge_ratio,cnt_ip,statistic='mean',bins=np.arange(0,1.0,0.1))

plt.xlabel('edge_ratio')
plt.ylabel('neighbours')
plt.title('Not suspicious domain: [redacted]com')

plt.plot(res_7.bin_edges[:-1],np.nan_to_num(res_7.statistic,0))
```

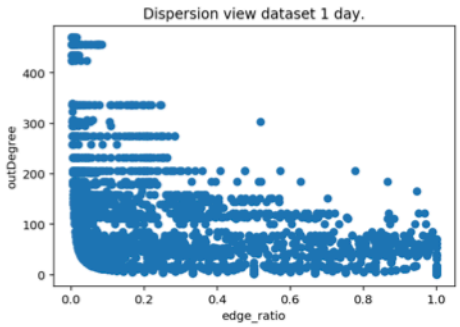
Out[70]:
```
[<matplotlib.lines.Line2D at 0x7f715a2590f0>]
```



Dispersion dataset 4 files

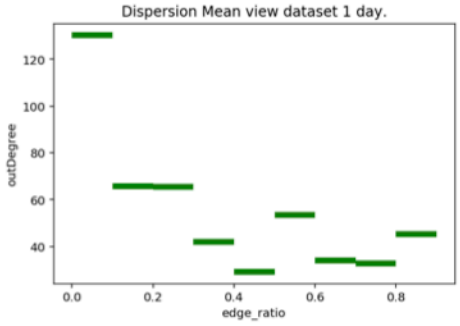
In [36]:
```
outDegree,edge_ratio= zip(*[ (row.outDegree,row.edge_ratio)   for row in  df_degree_ratio.sort("edge_ratio").collect() ])
plt.xlabel('edge_ratio')
plt.ylabel('outDegree')
plt.title('Dispersion view dataset 1 day.')
plt.plot(edge_ratio,outDegree,"o")
```

Out[36]:
```
[<matplotlib.lines.Line2D at 0x7f53dda33128>]
```



In [37]:
```
from scipy import stats
plt.xlabel('edge_ratio')
plt.ylabel('outDegree')
plt.title('Dispersion Mean view dataset 1 day.')
res=stats.binned_statistic(edge_ratio, outDegree, statistic='mean',bins=np.arange(0,1.0,0.1))
plt.hlines(res.statistic, res.bin_edges[:-1], res.bin_edges[1:], colors='g', lw=5)
```

Out[37]:
```
<matplotlib.collections.LineCollection at 0x7f550a73bf60>
```



In [ ]:
```

```

ALL DAY DATASET

We are gonna use all the files for the day selected to generate the dataset in order to obtain significant results.

The following lines of code, are the same ones that were described before, with an smaller dataset

In [3]:
```
df=spark.read.csv("file:///[redacted]8/ssp_bid_compressed_*",header=True)
```

In [4]:
```
df_cleaned = clean( df,"referrer_domain","user_ip" )
```

```
df_utils clean --
```

In [5]:
```
gf_domip = get_graph_domip(df_cleaned, 15)
```

```
gf_utils get_graph_domip-- :
df_utils get_edges_domip-- : df
df_utils get_edges --
df_utils get_vertices-- :
gf_utils filter_gf filterEdges : {min_edge}
```

In [6]:
```
gf_write_parquet(gf_domip,"file:///s[redacted]output/gf_domip_180208_ALLDAY_new")
```

```
Saved parquet graph into path :file:///s[redacted]/output/gf_domip_180208_ALLDAY_new
```

In [3]:
```
gf_domip=gf_read_parquet(spark,"file:///s[redacted]/output/gf_domip_180208_ALLDAY_new")
```

```
Readed parquet graph from path :file:///s[redacted]/output/gf_domip_180208_ALLDAY_new
```

In [ ]:

In [10]:
```
df_degree_ratio=get_df_degree_ratio(gf_domip)
```

```
gf_utils get_motifs -- df_motifs dropDuplicates( ['e', 'e2']
gf_utils get_motifs_count -- df_motifs_count
gf_utils get_df_degree_ratio --  df_degreeRatio :
```

In [11]:
```
df_write_parquet(df_degree_ratio,"file:///[redacted]a/output/df_degree_ratio_180208_ALLDAY_new")
```

```
Saved parquet df into path :file:///s[redacted]/output/df_degree_ratio_180208_ALLDAY_new
```

In [4]:
```
df_degree_ratio=df_read_parquet(spark,"file:///[redacted]a/output/df_degree_ratio_180208_ALLDAY")
```

```
Readed parquet df from path :file:///s[redacted]output/df_degree_ratio_180208_ALLDAY
```

In [60]:
```
df_degree_ratio=df_read_parquet(spark,"file:///s[redacted]/output/df_degree_ratio_180208_ALLDAY_new")
```

```
Readed parquet df from path :file:///s[redacted]/output/df_degree_ratio_180208_ALLDAY_new
```

In [ ]:

gf_domdom all day dataset 180208

In [45]:
```
gf_domdom = get_graph_domdom( gf_domip )
```

```
DomainDomainGraph get_graph_domdom -- g_domip.edges.show()
gf_utils get_motifs -- df_motifs dropDuplicates( ['e', 'e2']
gf_utils get_motifs_count -- df_motifs_count
gf_utils get_df_degree_ratio --  df_degreeRatio :
gf_utils get_df_degree_ratio --  df_degreeRatio division  (edge_ratio = covisitation degree:
df_utils get_edges_domdom -- df que llega ...
df_utils get_edges --
df_utils get_edges_domdom -- df_edges calculado ...
+--------------+--------------------+-------------------+
|           src|                 dst|        edge_weight|
+--------------+--------------------+-------------------+
|    [0[redacted]u]|         [i[redacted]m]|      0.49999999975|
|    [0[redacted]d]|      [j[redacted]om]|      0.49999999975|
|    [0[redacted]a]|       [p[redacted]a]|      0.49999999975|
|    [0[redacted]a]|         [r[redacted]d]|      0.49999999975|
|[redacted]|[redacted]|      0.49999999975|
|[redacted]|[redacted]|      0.49999999975|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
+--------------+--------------------+-------------------+
only showing top 20 rows

df_utils get_vertices-- :
df_utils get_vertices-- df_vertices_withduplicates :
df_utils get_vertices-- df_vertices_sin duplicates :
DomainDomainGraph get_graph_domdom --  df_edges OLAYAs :
+--------------+--------------------+-------------------+
|           src|                 dst|        edge_weight|
+--------------+--------------------+-------------------+
|    [0[redacted]a]|         [i[redacted]m]|      0.49999999975|
|    [0[redacted]a]|      [j[redacted]]|      0.49999999975|
|    [0[redacted]a]|       [p[redacted]a]|      0.49999999975|
|    [0[redacted]]|         [r[redacted]k]|      0.49999999975|
|[redacted]|[redacted]|      0.49999999975|
|[redacted]|[redacted]|      0.49999999975|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
|[redacted]|[redacted]|0.9999999989999999|
+--------------+--------------------+-------------------+
only showing top 20 rows
```

In [46]:
```
gf_write_parquet(gf_domdom,"file:///s[redacted]/output/gf_domdom_180208_ALLDAY")
```

```
 Salvado parquet grafo en el path :file:///s[redacted]/output/gf_domdom_180208_ALLDAY
```

In [5]:
```
gf_domdom=gf_read_parquet(spark,"file:///s[redacted]a/output/gf_domdom_180208_ALLDAY")
```

```
 Readed parquet graph from path :file:///s[redacted]a/output/gf_domdom_180208_ALLDAY
```

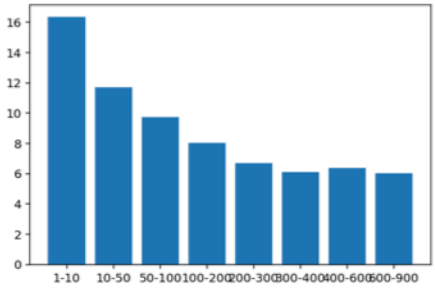
In [6]:
```
ig, visual_style = draw_igraph_domain_domain( gf_domdom )

plot( ig, **visual_style )#.save(
        #"file:///s[redacted]a/output/gf_domdom_180208_4files_weighted.png" )
```

```
draw_utils draw_igraph --

---------------------------------------------------------------------------
Error                                     Traceback (most recent call last)
<ipython-input-6-649edcb914a9> in <module>
      1 ig, visual_style = draw_igraph_domain_domain( gf_domdom )
      2
----> 3 plot( ig, **visual_style )#.save(
      4         #"file:///s[redacted]/output/gf_domdom_180208_4files_weighted.png" )

/usr/local/lib/python3.6/site-packages/igraph/drawing/__init__.py in plot(obj, target, bbox, *args, **kwds)
    448         bbox = BoundingBox(bbox)
    449
--> 450     result = Plot(target, bbox, background=kwds.get("background", "white"))
    451
    452     if "margin" in kwds:

/usr/local/lib/python3.6/site-packages/igraph/drawing/__init__.py in __init__(self, target, bbox, palette, background)
    139             self._need_tmpfile = True
    140             self._surface = cairo.ImageSurface(cairo.FORMAT_ARGB32, \
--> 141                 int(self.bbox.width), int(self.bbox.height))
    142         elif isinstance(target, cairo.Surface):
    143             self._surface = target

Error: invalid value (typically too big) for the size of the input (surface, pattern, etc.)
```

log histogram, first histogram

In [14]:
```
print( "plots_main MAIN-- Calculando primer histograma --  draw_log_hist -- ..." )
%matplotlib inline
total_degrees = gf_domip.degrees
print( f" type  {type( total_degrees )}  " )
sorted_degrees = total_degrees.orderBy( F.desc( "degree" ) )

degree, id = zip(*[(item.degree, item.id) for item in sorted_degrees.select( "degree", "id" ).collect()] )

draw_log_hist( degree, [1, 10, 50, 100, 200, 300, 400,600,900])#,"file:///s[redacted]a/output/log_hist_plot.png" )
```

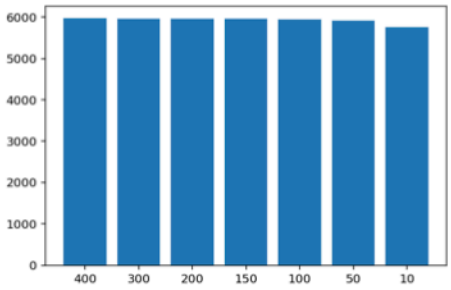
```
plots_main MAIN-- Calculando primer histograma --  draw_log_hist -- ...
 type  <class 'pyspark.sql.dataframe.DataFrame'>
draw_utils draw_log_hist -- --
```



Minor than a tope histogram, second histogram

In [68]:
```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_minor_than_list -- ..." )


list_tope = [400, 300, 200, 150, 100, 50, 10]
draw_minor_than_list( degree, list_tope)#"file:///s[redacted]/output/minor_than_list_plot.png" )
```

```
plots_main MAIN-- Calculando segundo histograma --  draw_minor_than_list -- ...
draw_utils draw_minor_than_list -- --
```



Overlap matrix for all day data 180208 taking 50 domain to represent

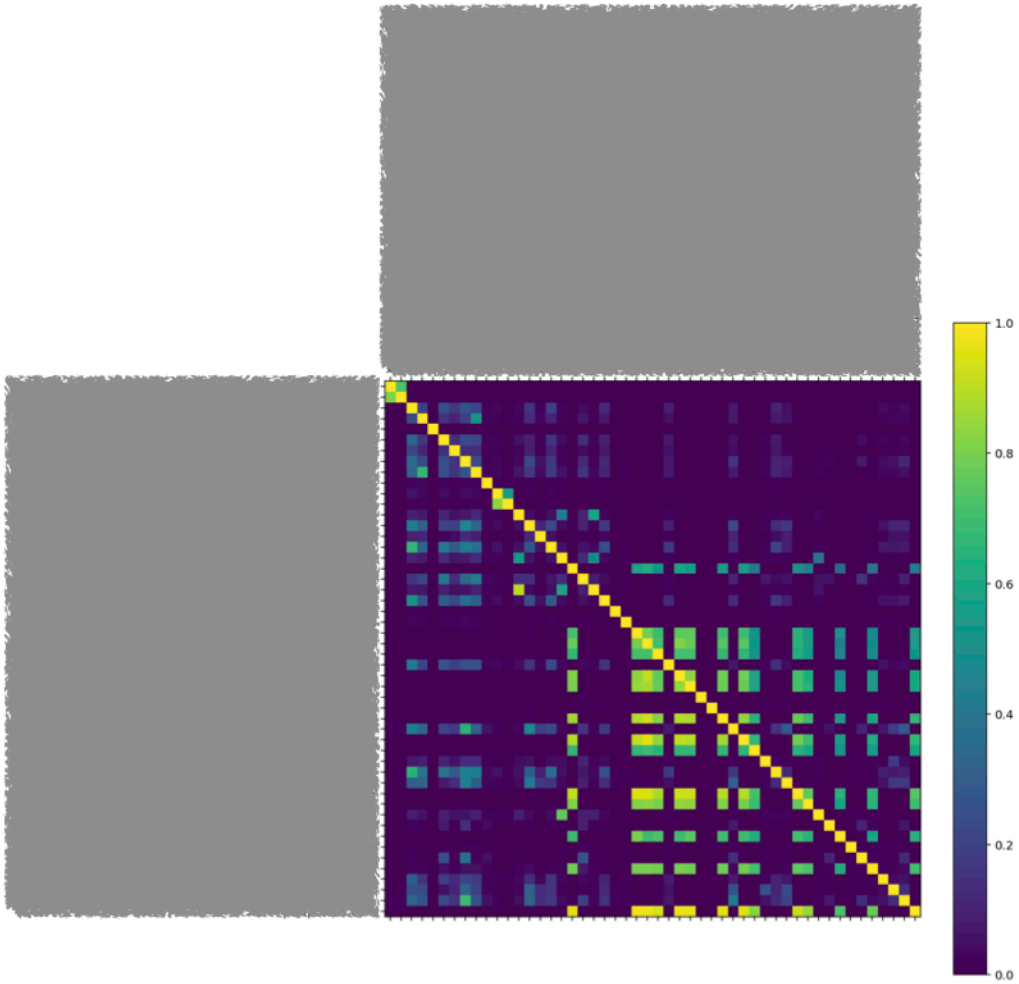
In [64]:
```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ..." )
top50_suspicious = df_degree_ratio.filter(
    "edge_ratio>0.5 and count_ips_in_common>1  " ).select(
    df_degree_ratio.a.id, df_degree_ratio.outDegree ).distinct().sort( F.desc( "outDegree" ) ).take( 50 )
list_top_suspicious = [row["a.id"] for row in top50_suspicious]
draw_overlap_matrix( df_degree_ratio, list_top_suspicious)#,"/[redacted]M/output_fraud/overlap_list_top_suspicious.pdf" )
```

```
plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ...
```

Con df_degree_ratio new, porque no me fio del dato

In [5]:
```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ..." )
top50_suspicious = df_degree_ratio.filter(
    "edge_ratio>0.5 and count_ips_in_common>1  " ).select(
    df_degree_ratio.a.id, df_degree_ratio.outDegree ).distinct().sort( F.desc( "outDegree" ) ).take( 50 )
list_top_suspicious = [row["a.id"] for row in top50_suspicious]
draw_overlap_matrix( df_degree_ratio, list_top_suspicious)#,"/[redacted]/output_fraud/overlap_list_top_suspicious.pdf" )
```

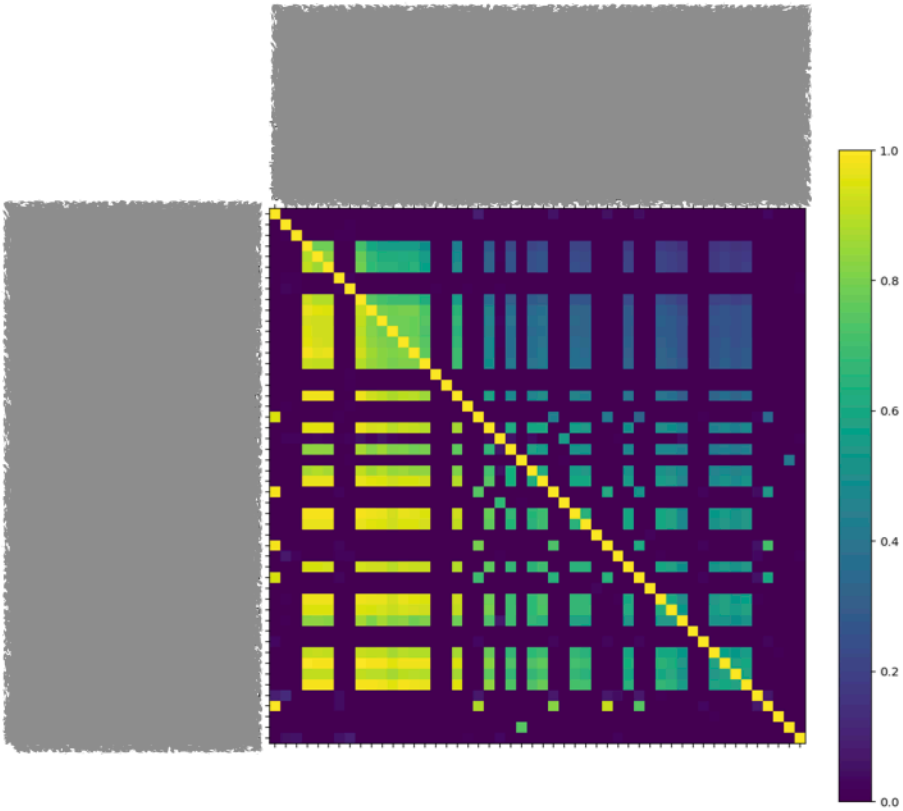
```
plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ...
```



Nueva ordenacion edge_ratio >0.9

In [7]:
```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ..." )
top50_suspicious = df_degree_ratio.filter(
    "edge_ratio>0.9 and count_ips_in_common>1  " ).select(
    df_degree_ratio.a.id, df_degree_ratio.outDegree ).distinct().sort( F.desc( "outDegree" ) ).take( 50 )
list_top_suspicious = [row["a.id"] for row in top50_suspicious]
draw_overlap_matrix( df_degree_ratio, list_top_suspicious)#,"/[redacted]/output_fraud/overlap_list_top_suspicious.pdf" )
```

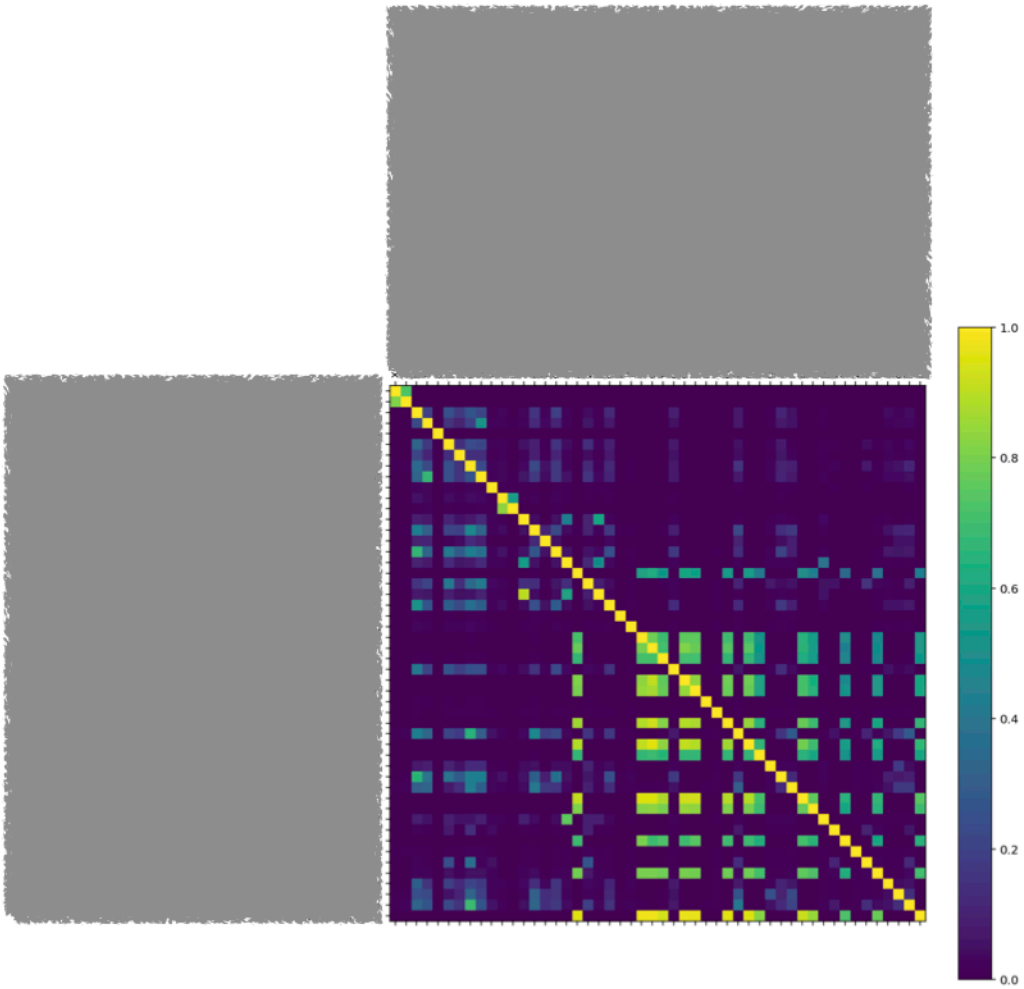
```
plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ...
```



Nueva ordenacion edge_ratio>0.5

In [11]:
```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ..." )
top50_suspicious = df_degree_ratio.filter(
    "edge_ratio>0.5 and count_ips_in_common>1  " ).select(
    df_degree_ratio.a.id, df_degree_ratio.outDegree ).distinct().sort( F.desc( "outDegree" ) ).take( 50 )
list_top_suspicious = [row["a.id"] for row in top50_suspicious]
draw_overlap_matrix( df_degree_ratio, list_top_suspicious)#,"[redacted]output_fraud/overlap_list_top_suspicious.pdf" )
```

```
plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ...
```



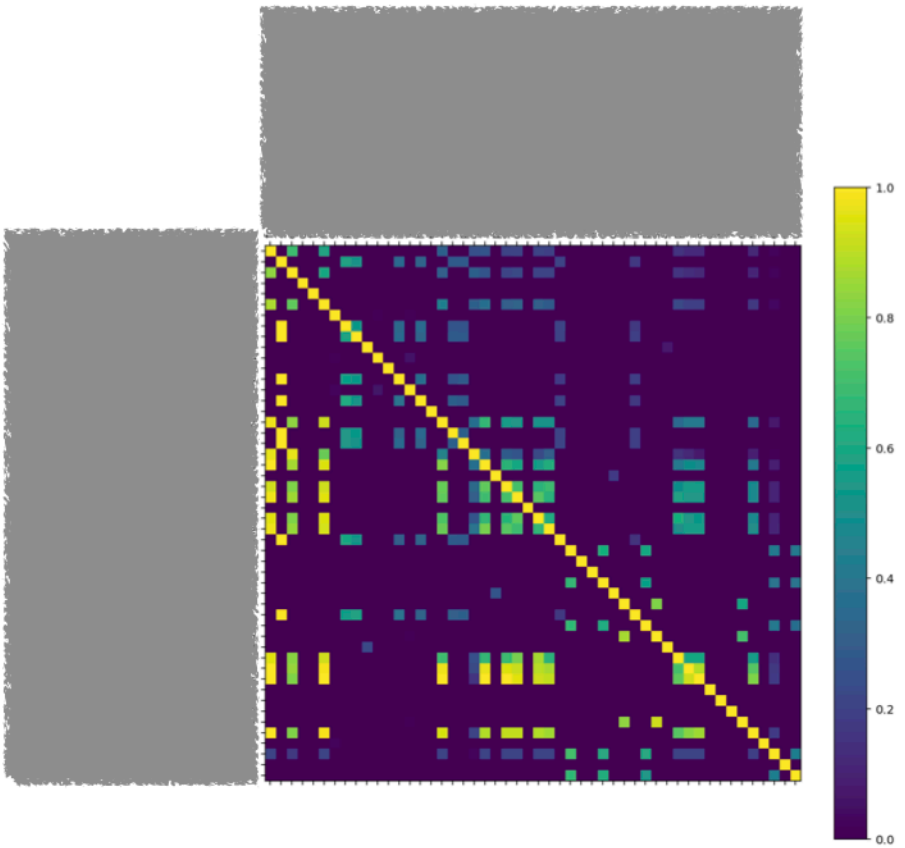
In [29]:
```
top50_suspicious = df_degree_ratio.filter(
    "edge_ratio>0.5 and count_ips_in_common>2 " ).select(
    df_degree_ratio.a.id, df_degree_ratio.outDegree, df_degree_ratio.edge_ratio ).distinct().sort(F.desc( "edge_ratio" ),F.desc( "outDegree" )).
```

In [31]:
```
top50_suspicious.head(100,False)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-31-ce552bafle2c> in <module>
----> 1 top50_suspicious.head(100,False)

AttributeError: 'list' object has no attribute 'head'
```

In [32]:
```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ..." )
top50_suspicious = df_degree_ratio.filter(
    "edge_ratio>0.5 and count_ips_in_common>2   " ).select(
    df_degree_ratio.a.id, df_degree_ratio.outDegree, df_degree_ratio.edge_ratio ).distinct().sort(F.desc( "edge_ratio" ),F.desc( "outDegree" )).
list_top_suspicious = [row["a.id"] for row in top50_suspicious]
draw_overlap_matrix( df_degree_ratio, list_top_suspicious)#,"/[redacted]/output_fraud/overlap_list_top_suspicious.pdf" )
```

```
plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_top_suspicious ...
```

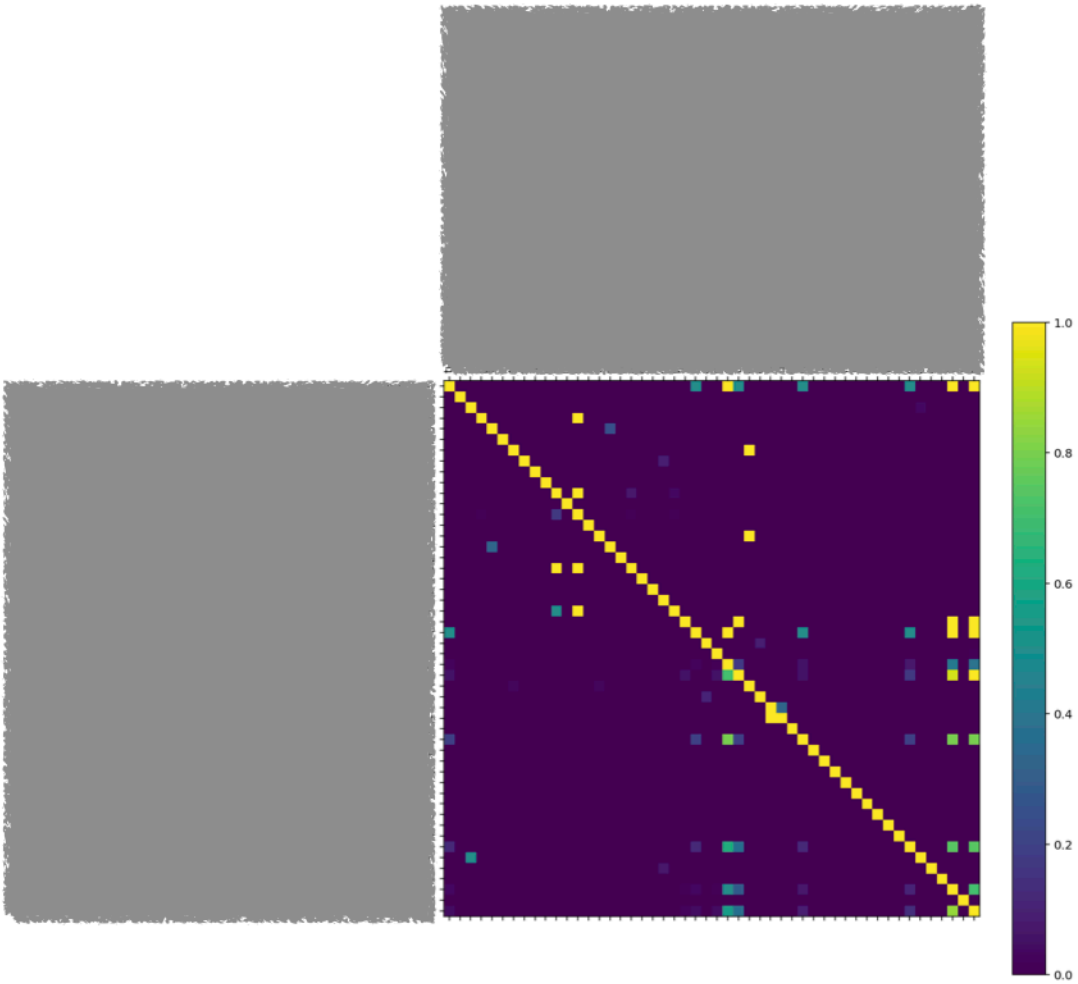


In [ ]:
```
```

Overlap matrix 20% random data for all day data 180208

In [58]:
```
print( "plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_sample ..." )
sample_50_percent = df_degree_ratio.select( F.col( "a.id" ) ).distinct().sample( 0.2 ).take( 50 )
# cojo un 20% de datos totales (dominios unicos) de la muestra de manera aleatoria
list_sample = [row["id"] for row in sample_50_percent]
draw_overlap_matrix( df_degree_ratio, list_sample)
```

```
plots_main MAIN-- Calculando segundo histograma --  draw_overlap_matrix  -- list_sample ...
```



Code for the function to draw the overlap matrix : 'draw_overlap_matrix', stored in the 'draw_utils.py' class.

In [57]:
```
def draw_overlap_matrix(df_degree_ratio,list_top_suspicious,figsize=(10,10),path=None):
    '''
    Function to draw an overlap matrix of suspicious domains
    :param df_degree_ratio : dataframe with all the data needed to represent the overlap matrix [a,c,count_ips_in_common,id,outDegree,edge_ratio
                             where a is src, c is dst, id is src, and outDegree is the outDegree of src. The edge_ratio is calculated with the
                            algorithm proposed.
    :param top_suspicious : number of top suspicious domains to plot
    :param figsize
    :param path : path where to save the histogram image
    :return ploted overlap matrix

    '''
    import matplotlib.ticker as ticker



    matrix_src_dsc = df_degree_ratio.filter(
        (F.col("a.id").isin(list_top_suspicious)) & (F.col("c.id").isin(list_top_suspicious))).select(
        F.col("a.id").alias("src"),F.col("c.id").alias("dst"),F.col("edge_ratio")).collect()

    dom_idx = dict([ (v,k) for k,v in enumerate(list_top_suspicious)]) # diccionario con indice-dominio y lo invierto
    # para que me lo de dominio-indice de la matriz

    dom_matrix=  np.eye(len(list_top_suspicious))  # matriz con diagonal en 1's de la long de la lista de top_susp

    for s,d,e in matrix_src_dsc:  # relleno la matriz con los valores
        dom_matrix[dom_idx[s],dom_idx[d]]=e

    fig = plt.figure(figsize=figsize) # tamaño de la matriz

    #fig.suptitle("Overlap domain matriz")
    ax = fig.add_subplot(111)
    cax = ax.matshow(dom_matrix)
    fig.colorbar(cax)

    # Set up axes
    # el primero vacio porque si no no pinta el q esta en la posicion 0
    ax.set_xticklabels(['']+list_top_suspicious, rotation=90)
    ax.set_yticklabels(['']+list_top_suspicious)

    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    if f"{path}" is not None:
        fig.savefig( f"{path}",format='pdf')
```

Looking for a malicious domain

We want to find a malicious domain, to see what is the entropy and other characteristics.

For that we filter the domains with less than 40 visits, and the ones that have the same outDegree and count_ips_in_common, in order to find one domain with an extrange behaviour.

Also in order to manage less dataset, we filter the domain that have less than 3 ips in common (total_ips_in_common).

Here in the results, we can observe that the edge_ratio near to one, indicates the possibility of a malicious one, also if this one has a big outDegree and a huge cuantity of ips_in_common con other domains.

In [72]:
```
df_degree_ratio.sort(F.desc("edge_ratio")).filter((df_degree_ratio.outDegree>40) & (df_degree_ratio.outDegree!=df_degree_ratio.count_ips_in_comm
```

[2[illegible]]	[[illegible]]		39
2[illegible]1	41	0.9512195121719216	
[t[illegible]m]	[[illegible]		116
t[illegible]	122	0.9508196721233539	
[[illegible]75]	[m[illegible]		58
c[illegible]	61	0.9508196721155604	
[a[illegible]e]	[j[illegible]		157
a[illegible]e	166	0.945783132524423	
[[illegible]]	[m[illegible]		51
[illegible]t	54	0.9444444444269547	
[t[illegible]m]	[[illegible]		95
[illegible]m	101	0.9405940593966278	
[j[illegible]]	[[illegible]]		44
j[illegible]	47	0.936170212746039	
[j[illegible]]	[[illegible]		44
j[illegible]	47	0.936170212746039	
[[illegible]]	[[illegible]		58
t[illegible]	62	0.9354838709526535	
[c[illegible]]	[[illegible]		43
c[illegible]	46	0.9347826086753309	

If we order the list of df_degree_ratio in ascendent mode by the edge_ratio, the firsts domains should be the licit or legitim ones.

In [71]:
```
df_degree_ratio.sort(F.asc("edge_ratio")).show(500)
```

```
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
|[redacted]|[redacted]|   1|[redacted]|   122|0.00819672131140823|
```

We want to take a view of the disperssion for the edge_ratio and the outDegree values in our dataset.

In order to represent that, we make 2 lists withs the columns data, and zip it in order to not use a loop to reduce the amount of time calculation.

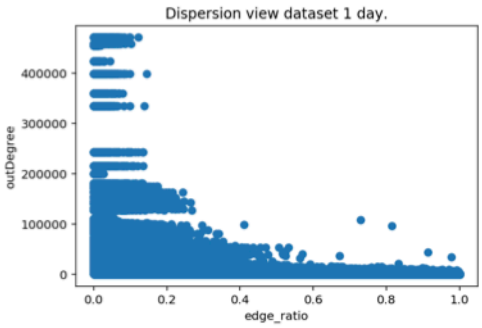
Then there is the plot obtained.

In [ ]:

In [30]:
```
outDegree,edge_ratio= zip(*[ (row.outDegree,row.edge_ratio)   for row in  df_degree_ratio.sort("edge_ratio").collect() ])
```

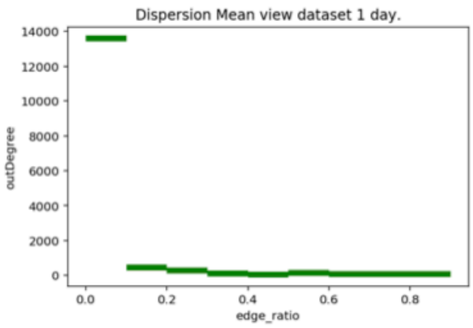
In [33]:
```
plt.xlabel('edge_ratio')
plt.ylabel('outDegree')
plt.title('Dispersion view dataset 1 day.')
plt.plot(edge_ratio,outDegree,"o")
```

Out[33]: [<matplotlib.lines.Line2D at 0x7f550b7d3c88>]



In [34]:
```
from scipy import stats
plt.xlabel('edge_ratio')
plt.ylabel('outDegree')
plt.title('Dispersion Mean view dataset 1 day.')
res=stats.binned_statistic(edge_ratio, outDegree, statistic='mean',bins=np.arange(0,1.0,0.1))
plt.hlines(res.statistic, res.bin_edges[:-1], res.bin_edges[1:], colors='g', lw=5)
```

Out[34]: <matplotlib.collections.LineCollection at 0x7f5438059fd0>



In order to locate the possible malicious domains, we think that could be a good idea to calculate the mean for the edge_ratio and the count_ips_in_common values per domain

In [15]:
```
df_mean_ratio =  df_degree_ratio.groupBy("id").agg(F.count("*").alias("cnt"),F.mean("edge_ratio").alias("mean_ratio"),F.mean("count_ips_in_commo
```

Looking for malicious (we order desc) :

In [89]:
```
df_mean_ratio.filter("cnt>3 and mean_ips_in_common>3").sort(F.desc("mean_ratio")).show(200,False)
```

```
+----------------------------------------+---+------------------+------------------+
|id                                      |cnt|mean_ratio        |mean_ips_in_common|
+----------------------------------------+---+------------------+------------------+
|[redacted]                              |66 |0.9515151513248468|4.757575757575758 |
|                                        |13 |0.916666666590278 |11.0              |
|                                        |11 |0.9090909088636363|3.6363636363636362|
|                                        |13 |0.8956043955404239|12.538461538461538|
|                                        |5  |0.8857142855877551|6.2               |
|                                        |6  |0.87499999978125  |3.5               |
|                                        |6  |0.87499999978125  |3.5               |
|                                        |6  |0.87499999978125  |3.5               |
|                                        |6  |0.87499999978125  |3.5               |
|                                        |6  |0.87499999978125  |3.5               |
|                                        |6  |0.87499999978125  |3.5               |
|                                        |19 |0.8684210524144738|3.473684210526316 |
|                                        |19 |0.8684210524144735|3.473684210526316 |
|                                        |19 |0.8684210524144735|3.473684210526316 |
|                                        |11 |0.8636363634924241|5.181818181818182 |
|[redacted]                              |12 |0.8571428570204084|6.0               |
```

In [19]:
```
df_mean_ratio.filter("cnt>3 and mean_ips_in_common>3").sort(F.desc("mean_ratio")).show(5,False)
```

```
+----------------------+---+------------------+------------------+
|id                    |cnt|mean_ratio        |mean_ips_in_common|
+----------------------+---+------------------+------------------+
|[redacted]            |66 |0.9515151513248468|4.757575757575758 |
|                      |13 |0.916666666590278 |11.0              |
|                      |11 |0.9090909088636363|3.6363636363636362|
|                      |13 |0.8956043955404239|12.538461538461538|
|                      |5  |0.8857142855877551|6.2               |
+----------------------+---+------------------+------------------+
only showing top 5 rows
```

Looking for licit domain (ordering asc)

In [90]:
```
df_mean_ratio.filter("cnt>5 and mean_ips_in_common>5").sort(F.asc("mean_ratio")).show(200,False)
```

```
+----------------------------------------+-----+----------------------+------------------+
|id                                      |cnt  |mean_ratio            |mean_ips_in_common|
+----------------------------------------+-----+----------------------+------------------+
|[redacted]                              |4900 |7.558606650946203E-5  |12.391428571428571|
|                                        |8164 |1.1705999993549429E-4 |53.16303282704557 |
|                                        |15186|1.6160109770223293E-4 |68.53114710917951 |
|                                        |3884 |1.9411516689410465E-4 |13.524974253347064|
|                                        |2626 |2.2996102016326143E-4 |8.006092916984006 |
|                                        |6422 |2.570392301598806E-4  |51.21223917782622 |
|                                        |4328 |2.7073294922421 87E-4 |12.916127541589649|
|                                        |5632 |3.281655448663306E-4  |28.9609375        |
|                                        |3308 |3.4779829269320937E-4 |20.59522370012092 |
|                                        |3012 |3.5889789614123164E-4 |38.743027888446214|
|                                        |3203 |3.6107461183281013E-4 |12.94380268498283 |
|                                        |3194 |3.645484892275633E-4  |14.325297432686288|
|                                        |3494 |3.953481090557965E-4  |38.23411562678878 |
|                                        |11451|4.058539233028869E-4  |145.8748580909964 |
|                                        |2916 |4.1343508416403287E-4 |12.054526748971194|
|[redacted]                              |1942 |4.1417464726059984E-4 |7.408341915550978 |
```

In [24]:
```
df_mean_ratio.filter("cnt>5 and mean_ips_in_common>5").sort(F.asc("mean_ratio")).show(5,False)
```

```
+--------------------+-----+----------------------+------------------+
|id                  |cnt  |mean_ratio            |mean_ips_in_common|
+--------------------+-----+----------------------+------------------+
|[redacted]          |4900 |7.558606650946203E-5  |12.391428571428571|
|                    |8164 |1.1705999993549429E-4 |53.16303282704557 |
|                    |15186|1.6160109770223293E-4 |68.53114710917951 |
|                    |3884 |1.9411516689410465E-4 |13.524974253347064|
|                    |2626 |2.2996102016326143E-4 |8.006092916984006 |
+--------------------+-----+----------------------+------------------+
only showing top 5 rows
```

We select as a maliciuous domain :

```
                id  |cnt|mean_ratio           |mean_ips_in_common|
[redacted]          |66 |0.21132376395348923 |24.09090909090909 |
```

We filter the dataset looking for this domain and save it in cnt_edge.

Also the use of zip to not use a for loop.

In [80]:
```
cnt_edge =  df_degree_ratio.filter("id = '[redacted]' ").select("count_ips_in_common","edge_ratio").sort("edge_ratio").collect()
```

In [81]:
```
edge_ratio,cnt_ip =zip(*[(row.edge_ratio,row.count_ips_in_common) for row in  cnt_edge])
```

Use this function to represent the sample into an histogram :

```
scipy.stats.binned_statistic (x, values, statistic='mean', bins=10, range=None)[source]

    Compute a binned statistic for one or more sets of data.

    This is a generalization of a histogram function.
    A histogram divides the space into bins, and returns the count of the number of points in each bin.
    This function allows the computation of the sum, mean, median, or other statistic of the values (or set of values) within each bin.
```

In [82]:
```
from scipy import stats

res_5=stats.binned_statistic(edge_ratio,cnt_ip,statistic='mean',bins=np.arange(0,1.0,0.1))
```

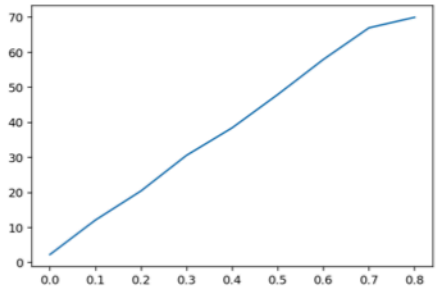
Here is the plot of the average for the total visits to the

'considered malicious'

domain selected before, represented with the co-visitation rate associated.

In [83]:
```
plt.plot(res_5.bin_edges[:-1],res_5.statistic)
```

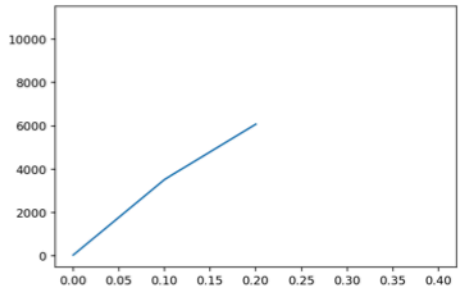
Out[83]:
```
[<matplotlib.lines.Line2D at 0x7f5400bf6278>]
```



[redacted]

In [7]:
```
cnt_edge =  df_degree_ratio.filter("id = '[redacted]' ").select("count_ips_in_common","edge_ratio").sort("edge_ratio").collect()
edge_ratio,cnt_ip =zip(*[(row.edge_ratio,row.count_ips_in_common) for row in  cnt_edge])
from scipy import stats

res_5=stats.binned_statistic(edge_ratio,cnt_ip,statistic='mean',bins=np.arange(0,1.0,0.1))
plt.plot(res_5.bin_edges[:-1],res_5.statistic)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x7f8f2f463588>]
```



We follow the same clasification but for a consider worthy domain . For this case we select the domain : "[redacted]

In [19]:
```
cnt_edge =  df_degree_ratio.filter("id = '[redacted]' ").select("count_ips_in_common","edge_ratio").sort("edge_ratio").collect()
```

In [20]:
```
edge_ratio,cnt_ip =zip(*[  (row.edge_ratio,row.count_ips_in_common) for row in  cnt_edge])
```

In [21]:
```
from scipy import stats

res_6=stats.binned_statistic(edge_ratio,cnt_ip,statistic='mean',bins=np.arange(0,1.0,0.1))
```

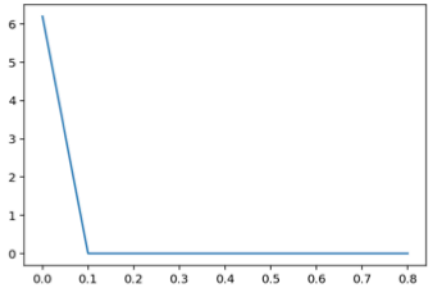
Here is the plot of the average for the total visits to the

'considered worthy'

domain selected before, represented with the co-visitation rate associated.

In [22]:
```
plt.plot(res_6.bin_edges[:-1],np.nan_to_num(res_6.statistic,0))
```

Out[22]:
```
[<matplotlib.lines.Line2D at 0x7f53f79c3f60>]
```



Also if we calculate this plot for othe domain considered worthy, the plot looks similar.

We choose the worthy domain "[redacted]"

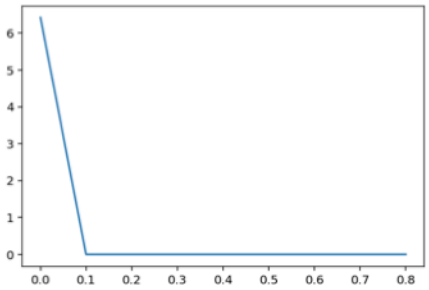
In [68]:
```
cnt_edge =  df_degree_ratio.filter("id = '[redacted]' ").select("count_ips_in_common","edge_ratio").sort("edge_ratio").collect()
```

In [69]:
```
edge_ratio,cnt_ip =zip(*[(row.edge_ratio,row.count_ips_in_common) for row in  cnt_edge])
```

In [70]:
```
res_7=stats.binned_statistic(edge_ratio,cnt_ip,statistic='mean',bins=np.arange(0,1.0,0.1))
```

In [71]:
```
plt.plot(res_7.bin_edges[:-1],np.nan_to_num(res_7.statistic,0))
```

Out[71]:
```
[<matplotlib.lines.Line2D at 0x7f5400d3ff28>]
```



Malicious Domains selected between days (4 days sample)

In [5]:
```
df_mean_ratio.filter("id='[redacted]'").show(2,False) #day5
df_mean_ratio.filter("id='[redacted]'").show(2,False) #day6
df_mean_ratio.filter("id='[redacted]'").show(2,False) #day7
df_mean_ratio.filter("id='[redacted]'").show(2,False) #day8
```

```
+-----------------+---+-----------------+-----------------+
|id               |cnt|mean_ratio       |mean_ips_in_common|
+-----------------+---+-----------------+-----------------+
|l[redacted]m|320|0.6999999999300017|7.0              |
+-----------------+---+-----------------+-----------------+

+---------------------+---+------------------+------------------+
|id                   |cnt|mean_ratio        |mean_ips_in_common|
+---------------------+---+------------------+------------------+
|h[redacted]k|155|0.5661290322108875|6.7935483870967746|
+---------------------+---+------------------+------------------+

+--------------------+---+------------------+------------------+
|id                  |cnt|mean_ratio        |mean_ips_in_common|
+--------------------+---+------------------+------------------+
|p[redacted]|66 |0.6774891774407851|9.484848484848484 |
+--------------------+---+------------------+------------------+

+------------+----+--------------------+------------------+
|id          |cnt |mean_ratio          |mean_ips_in_common|
+------------+----+--------------------+------------------+
|m[redacted]n|8736|0.0013612820314511961|193.61378205128204|
+------------+----+--------------------+------------------+
```

Legitim Domains selected between days (4 days sample)

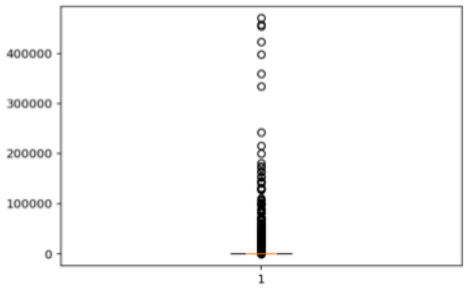
In [6]:
```
df_mean_ratio.filter("id='[redacted]'").show(2,False) #day5
df_mean_ratio.filter("id='[redacted]'").show(2,False) #day6
df_mean_ratio.filter("id='[redacted]'").show(2,False) #day7
df_mean_ratio.filter("id='[redacted]'").show(2,False) #day8
```

```
+---------+----+--------------------+------------------+
|id       |cnt |mean_ratio          |mean_ips_in_common|
+---------+----+--------------------+------------------+
|w[redacted]n|2626|2.2996102016326143E-4|8.006092916984006 |
+---------+----+--------------------+------------------+

+-----+----+--------------------+------------------+
|id   |cnt |mean_ratio          |mean_ips_in_common|
+-----+----+--------------------+------------------+
|d[redacted]|4900|7.558606650946203E-5|12.391428571428571|
+-----+----+--------------------+------------------+

+------------+----+--------------------+------------------+
|id          |cnt |mean_ratio          |mean_ips_in_common|
+------------+----+--------------------+------------------+
|m[redacted]n|3884|1.9411516689410465E-4|13.524974253347064|
+------------+----+--------------------+------------------+

+---------+-----+--------------------+------------------+
|id       |cnt  |mean_ratio          |mean_ips_in_common|
+---------+-----+--------------------+------------------+
|y[redacted]|15186|1.6160109770223293E-4|68.53114710917951 |
+---------+-----+--------------------+------------------+
```

In [5]:
```
total_degrees = gf_domip.degrees
print( f" type  {type( total_degrees )}  " )
sorted_degrees = total_degrees.orderBy( F.desc( "degree" ) )
```

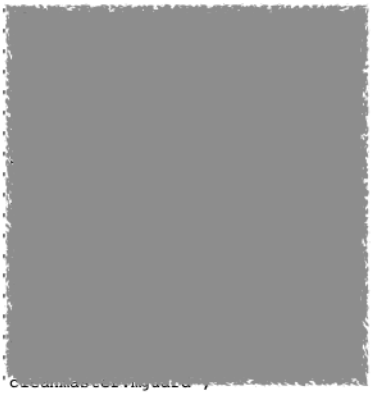
```
 type  <class 'pyspark.sql.dataframe.DataFrame'>
```

In [6]:
```
from numpy.random import seed
from numpy.random import randn
from matplotlib import pyplot
# seed the random number generator
#seed(1)
# random numbers drawn from a Gaussian distribution
degree, id = zip(*[(item.degree, item.id) for item in sorted_degrees.select( "degree", "id" ).collect()] )

#x = [randn(1000), 5 * randn(1000), 10 * randn(1000)]
# create box and whisker plot
#pyplot.boxplot(x)
pyplot.boxplot(degree)
# show line plot
pyplot.show()
```

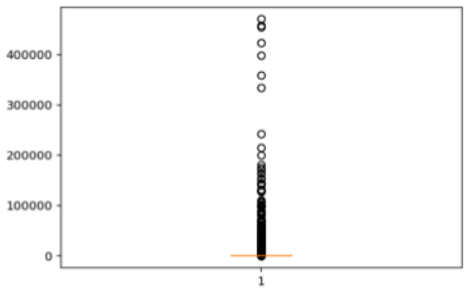


In [10]:
```
id
```

Out[10]:
```
(
```



In [11]:
```
box_plot_data=[degree]
pyplot.boxplot(box_plot_data)
pyplot.show()
```



In [16]:
```
cnt_edge =  df_degree_ratio.filter("id = '[redacted]' ").select("count_ips_in_common","edge_ratio").sort("edge_ratio").collect()
```

In [17]:
```
pyplot.boxplot(cnt_edge)
pyplot.show()
```

In [26]:
```
print( "plots_main MAIN-- Calculando primer histograma --  draw_log_hist -- ..." )
%matplotlib inline
total_degrees = gf_domip.degrees
print( f" type  {type( total_degrees )}  " )
sorted_degrees = total_degrees.orderBy( F.desc( "degree" ) )

degree, id = zip(*[(item.degree, item.id) for item in sorted_degrees.select( "degree", "id" ).collect()] )

draw_log_hist( degree, [1, 10, 50, 100, 200, 300, 400,600,900])#,"file:///srv-tardigrado/ogarcia/output/log_hist_plot.png" )
```

Out[26]:
```
[Row(id='m[illegible]', degree=470528),
 Row(id='e[illegible]', degree=458575),
 Row(id='l[illegible]', degree=454152),
 Row(id='y[illegible]', degree=424076),
 Row(id='w[illegible]r', degree=398448)]
```

In [108]:
```
%matplotlib inline
df_sorted_mean_ratio=df_mean_ratio.filter("cnt>100 and mean_ips_in_common>3 and mean_ratio>0.2").sort(F.desc("mean_ratio")).select("mean_ratio")
```

In [109]:
```
df_sorted_mean_ratio.show(500,False)
```

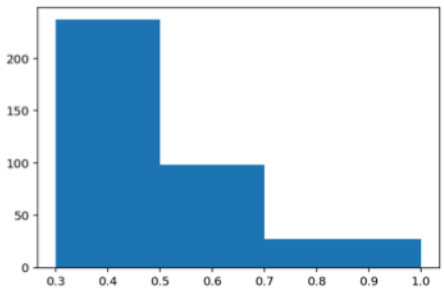
```
+-------------------+
|mean_ratio         |
+-------------------+
|0.8102605861166549 |
|0.8029850744662668 |
|0.793478260671197  |
|0.7917721517403773 |
|0.7871352783178055 |
|0.7771577379009464 |
|0.7741100322076352 |
|0.7698675494764052 |
|0.7685534589657838 |
|0.7603833863913755 |
|0.7463829785741367 |
|0.7463829785741366 |
|0.7460925038380309 |
|0.7460925038380299 |
|0.7455357141925216 |
|0.7346230157680704 |
```

In [ ]:
```
#mean_ratio=df_sorted_mean_ratio.withColumn(np.round(df_sorted_mean_ratio["mean_ratio"]))
#mean_ratio.show(20)
```

In [183]:
```
def draw_log_hist(degree, bins=10,path=None):
    '''
    Function to draw a histogram in logaritmic scale.

    :param degree : node degree
    :param bins   : division of the histogram, dos methods
    :param path   : path where to save the histogram image
    :return : ploted bar histogram

    draw_log_hist(degree,10):
    returns 10 bars with division made by np.histogram

    draw_log_hist(degree,[1,10,100,200]):
    returns plot between the numbers passed as a parameter,
    it means that sums the number of elements between 1-10, 10-100,100-200 ....

    LOGARITHMIC SCALE

    '''
    degree = np.array( degree )
    hist_y, hist_x = np.histogram( degree, bins )
    label_x = [str( int( hist_x[i] ) ) + "-" + str( int( hist_x[i + 1] ) ) for i in range( hist_x.shape[0] - 1 )]

    if f"{path}" is not None:
        plt.savefig( f"{path}" )

    #plt.ylim(0, 2)
    #plt.xlim(0,10)
    #plt.bar( label_x, np.log( hist_y + 1 ) )
    plt.bar( label_x,  np.log( hist_y + 1 )  )
```

In [184]:
```
plt.hist(degree, [0.3,0.5,0.7,1])
```

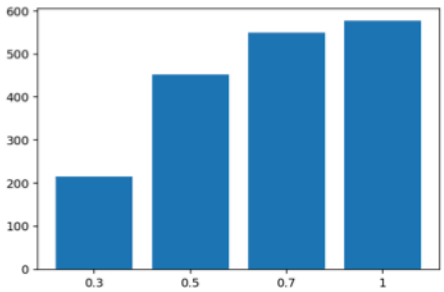
Out[184]:
```
(array([237.,  98.,  27.]),
 array([0.3, 0.5, 0.7, 1. ]),
 <a list of 3 Patch objects>)
```



In [ ]:

In [178]:
```
list_tope = [0.3,0.5,0.7,1]
draw_minor_than_list( degree, list_tope)#"file:///s[redacted]/output/minor_than_list_plot.png" )
```

```
draw_utils draw_minor_than_list -- --
```

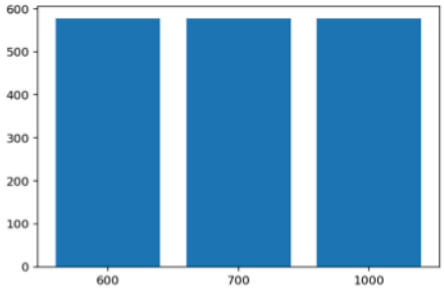


In [61]:
```
import numpy as np
```

In [161]:
```
df_sorted_mean_ratio=df_mean_ratio.filter("cnt>100 and mean_ips_in_common>3 and mean_ratio>0.2 ").sort(F.desc("mean_ratio"))
degree, id = zip(*[(item.mean_ratio, item.id) for item in df_sorted_mean_ratio.select( "mean_ratio", "id" ).collect()] )
```

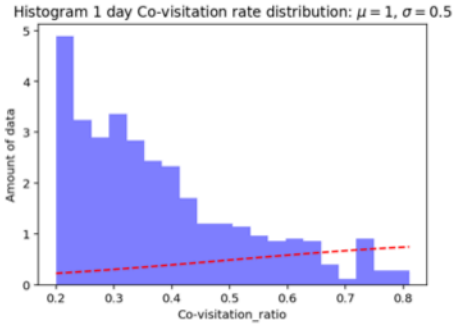
In [173]:
```
list_tope = [400, 300, 200, 150, 100, 50, 10]
draw_minor_than_list( degree, list_tope)#"file:///s[redacted]/output/minor_than_list_plot.png" )
```

```
draw_utils draw_minor_than_list -- --
```



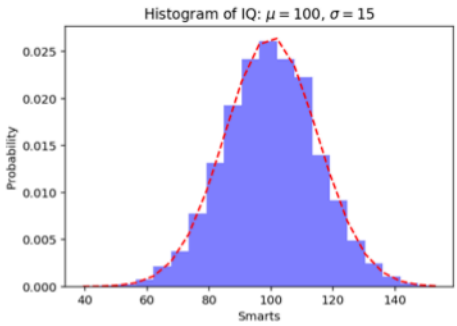
In [185]:
```
#!/usr/bin/env python

import numpy as np
import matplotlib.mlab as mlab
import scipy.stats
import matplotlib.pyplot as plt

#degree = np.array( degree )
df_degree_ratio=df_read_parquet(spark,"file:///s[redacted]/df_degree_ratio_180208_4files")
#df_degree_ratio=df_read_parquet(spark,"file:///[redacted]t/df_degree_ratio_180208_ALLDAY_new")
df_sorted_mean_ratio=df_mean_ratio.filter("cnt>100 and mean_ips_in_common>3 and mean_ratio>0.2 ").sort(F.desc("mean_ratio"))
degree, id = zip(*[(item.mean_ratio, item.id) for item in df_sorted_mean_ratio.select( "mean_ratio", "id" ).collect()] )

# example data
mu = 1 # mean of distribution
sigma = 0.5 # standard deviation of distribution
x = degree #mu + sigma * np.random.randn(10000)

num_bins = 20
# the histogram of the data
n, bins, patches = plt.hist(x, num_bins, density=1, facecolor='blue', alpha=0.5)

# add a 'best fit' line
y = mlab.normpdf(bins, mu, sigma)
#y=scipy.stats.norm.pdf(x, mu, sigma)
y=mlab.normpdf(bins, mu, sigma) * sum(n * np.diff(bins))
plt.plot(bins, y, 'r--')
plt.xlabel('Co-visitation_ratio')
plt.ylabel('Amount of data')
plt.title(r'Histogram 1 day Co-visitation rate distribution: $\mu=1$, $\sigma=0.5$')

# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()
```

```
Readed parquet df from path :file:///s[redacted]/df_degree_ratio_180208_4files

/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:24: MatplotlibDeprecationWarning: scipy.stats.norm.pdf
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:26: MatplotlibDeprecationWarning: scipy.stats.norm.pdf
```



In [96]:
```
#!/usr/bin/env python

import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

# example data
mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(10000)

num_bins = 20
# the histogram of the data
n, bins, patches = plt.hist(x, num_bins, density=1, facecolor='blue', alpha=0.5)

# add a 'best fit' line
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')

# Tweak spacing to prevent clipping of ylabel
plt.subplots_adjust(left=0.15)
plt.show()
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:17: MatplotlibDeprecationWarning: scipy.stats.norm.pdf
```



In [4]:
```
spark
```

Out[4]:

SparkSession - in-memory

SparkContext

Spark UI ([redacted])

Version

v2.4.0

Master

local[12]

AppName

pyspark-shell

PRUEBA MALICIOUS GRAPH

In [14]:
```
#gf_domip=gf_read_parquet(spark,"file:[redacted]a/output/gf_domip_1180208_ALLDAY_new")
```

In [20]:
```
gf_domip=gf_read_parquet(spark,"file:[redacted]a/output/gf_domip_180208_4files")
```

```
 Readed parquet graph from path :file:/[redacted]a/output/gf_domip_180208_4files
```

In [62]:
```
outDeg = gf_domip.outDegrees
outDeg.show(20,False)
```

```
+-------------------------+---------+
|id                       |outDegree|
+-------------------------+---------+
|j[redacted]c             |87       |
|[redacted]er             |4        |
|[redacted]om             |1        |
|[redacted]om             |1        |
|[redacted]               |1        |
|[redacted]               |1        |
|[redacted]               |1        |
|[redacted]               |1        |
|[redacted]               |3        |
|[redacted]               |1        |
|[redacted]               |2        |
|[redacted]               |1        |
|[redacted]               |9        |
|[redacted]               |26       |
|[redacted]               |7        |
|[redacted]               |4        |
|[redacted]               |29       |
|[redacted]               |1        |
|[redacted]               |2        |
|[redacted]               |2        |
+-------------------------+---------+
only showing top 20 rows
```

In [ ]:
```
```

In [59]:
```python
df_motifs = gf_domip.find( "(a)-[e]->(b); (c)-[e2]->(b)" ).filter( "a != c" ).dropDuplicates( ['e', 'e2'] )
df_motifs.show(20,False)
```

```
+----------------------------------------+----------------------------------------------------------+---------------+-----------------------
--+--------------------------------------------+
|a                                       |e                                                         |b              |c
|e2                                      |
+----------------------------------------+----------------------------------------------------------+---------------+-----------------------
--+--------------------------------------------+
|[[redacted]]                            |[[redacted], 35]                                          |[[redacted]]   |[[redacted]]
|[[redacted]]                            |
|[[redacted]]                            |[[redacted], 17]                                          |[[redacted]]   |[[redacted]]
|[[redacted]]                            |
|[[redacted]]                            |[[redacted], 19]                                          |[[redacted]]   |[[redacted]]
|[[redacted]]                            |
|[[redacted]]                            |[[redacted], 16]                                          |[[redacted]]   |[[redacted]]
[redacted]
+----------------------------------------+----------------------------------------------------------+---------------+-----------------------
--+--------------------------------------------+
only showing top 20 rows
```

In [60]:
```python
df_motifs_count = df_motifs.groupBy( 'a', 'c' ).agg( F.count( F.col( "b" ) ).alias( "count_ips_in_common" ) )
print( "gf_utils get_motifs_count -- df_motifs_count " )
df_motifs_count.show()
```

```
gf_utils get_motifs_count -- df_motifs_count
+--------------------+--------------------+-------------------+
|                   a|                   c|count_ips_in_common|
+--------------------+--------------------+-------------------+
|        [[redacted]]|        [[redacted]]|                 51|
|        [[redacted]]|        [[redacted]]|                 12|
|        [[redacted]]|        [[redacted]]|                 49|
|        [[redacted]]|        [[redacted]]|                 15|
|          [redacted]|          [redacted]|                 36|
|                    |                    |                  4|
|                    |                    |                 24|
|                    |                    |                  2|
|                    |                    |                  8|
|                    |                    |                  5|
|                    |                    |                  1|
|                    |                    |                  8|
|                    |                    |                  3|
|                    |                    |                  1|
|                    |                    |                  2|
|                    |                    |                  1|
|                    |                    |                  1|
|                    |                    |                  1|
|                    |                    |                  1|
|                    |                    |                  1|
+--------------------+--------------------+-------------------+
only showing top 20 rows
```

In [12]:
```python
df_degree_ratio=df_read_parquet(spark,"file:///[redacted]/output/df_degree_ratio_180208_ALLDAY_new")
```

```
Readed parquet df from path :file:///[redacted]/output/df_degree_ratio_180208_ALLDAY_new
```

In [54]:
```
df_degree_ratio.show(1000,False)
```

```
+------------+--------------------------------------------------+------------------+----------+---------+------------------+
|[redacted]                                                                                                                  |
+------------+--------------------------------------------------+------------------+----------+---------+------------------+
only showing top 1000 rows
```

In [13]:
```
gf_domdom=gf_read_parquet(spark,"file:///[redacted]/output/gf_domdom_180208_ALLDAY")
```

```
 Readed parquet graph from path :file:///[redacted]/output/gf_domdom_180208_ALLDAY
```

In [ ]:
```
df=spark.read.csv("file:///s[redacted]*",header=True)
```

In [17]:
```
df=spark.read.csv("file:///s[redacted][0-3]*",header=True)
```

In [98]:
```
def malicious_ones(df, neighbor=None):
    """
    Creating a df_edges to use GraphFrames only with the suspicious domains
    :param df: dataframe from our data. Idem format like in get_vertices function. df_degree_ratio
    :param neighbor : minimum number of neighbors of the node.
    :return: df_edges
    """
    if f"{neighbor}" is None:
        neighbor==f"{neighbor}"
        print (f"EN IF NEIGHBOR: {neighbor}")
    else:
        neighbor=80
        print (f"EN ELSE NEIGHBOR: {neighbor}")


    print( "df_utils get_edges_domdom -- df que llega ..." )
    df.show(5,False)
    df.printSchema()

    # this is only for fitler all suspicius domains
    print (f"NEIGHBOR: {neighbor}")
    df_filtered_neighbor = df.filter(F.col('outDegree') > f"{neighbor}" )
    print( "df_utils get_edges_domdom_malicious_ones --df_filtered_neighbor.show()" )
    df_filtered_neighbor.show(20,False)
    df_filtered_ratio= df_filtered_neighbor.filter(F.col('edge_ratio') >0.5)
    print( "df_utils df_otro -- df que llega ..." )
    df_filtered_ratio.show(20,False)


    #df_edges_DD_exists = df_filtered_neighbours.select( df.a, df.c,
    #                                                    F.when( df_filtered_neighbors['edge_ratio'] > 0.5,
    #                                                            1 ).otherwise( 0 ).alias(
    #                                                "edge_ratio" ) )  # .show()

    ##df_filtered_neighbor = df.select( df.a, df.c, F.when( df['edge_ratio'] > 0.5, df['edge_ratio'] ).alias(
    ##                                                    "edge_ratio" ), F.when(df['count_ips_in_common'] > 5, df['count_ips_in_common'] ).
    ##                                                    "count_ips_in_common" ))

    df_edges_DD_exists = df_filtered_ratio.select( df.a, df.c, df.edge_ratio )

    # print( "df_utils get_edges_domdom -- antes get_edges ..." )

    df_edges = get_edges( df_edges_DD_exists, "a", "c", "edge_ratio" )

    print( "df_utils get_edges_domdom_malicious_ones -- df_edges calculado MALICIOUS..." )
    df_edges.show()

    return df_edges
```

In [99]:
```
malicious_ones(df_degree_ratio)
```

EN ELSE NEIGHBOR: 80
df_utils get_edges_domdom -- df que llega ...

a	c	count_ips_in_common	id	outDegree	edge_ratio
[redacted]	[redacted]	1	[redacted]	1	0.9999999989999999
[redacted]	[redacted]	1	[redacted]	1	0.9999999989999999
[redacted]	[redacted]	1	[redacted]	1	0.9999999989999999
[redacted]	[redacted]	1	[redacted]	1	0.9999999989999999
[redacted]	[redacted]	1	[redacted]	1	0.9999999989999999

only showing top 5 rows

```
root
 |-- a: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |-- c: struct (nullable = true)
 |    |-- id: string (nullable = true)
 |-- count_ips_in_common: long (nullable = true)
 |-- id: string (nullable = true)
 |-- outDegree: integer (nullable = true)
 |-- edge_ratio: double (nullable = true)
```

NEIGHBOR: 80
df_utils get_edges_domdom_malicious_ones --df_filtered_neighbor.show()

a	c	count_ips_in_common	id	outDegree	edge_ratio
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	2	[redacted]	113	0.01769911504409116
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	11	[redacted]	113	0.09734513274250137
[redacted]	[redacted]	6	[redacted]	113	0.053097345132273474
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	4	[redacted]	113	0.03539823008818232
[redacted]	[redacted]	2	[redacted]	113	0.01769911504409116
[redacted]	[redacted]	4	[redacted]	113	0.03539823008818232
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	1	[redacted]	113	0.00884955752204558
[redacted]	[redacted]	5	[redacted]	113	0.04424778761022789

only showing top 20 rows

df_utils df_otro -- df que llega ...

a	c	count_ips_in_common	id	outDegree	edge_ratio
[redacted]	[redacted]	116	[redacted]	119	0.974789915958195
[redacted]	[redacted]	69	[redacted]	119	0.5798319327682366
[redacted]	[redacted]	115	[redacted]	119	0.9663865546137278
[redacted]	[redacted]	113	[redacted]	119	0.9495798319247934
[redacted]	[redacted]	97	[redacted]	119	0.8151260504133182
[redacted]	[redacted]	111	[redacted]	119	0.932773109235859
[redacted]	[redacted]	93	[redacted]	119	0.7815126050354495
[redacted]	[redacted]	100	[redacted]	119	0.8403361344467198
[redacted]	[redacted]	104	[redacted]	119	0.8739495798245887
[redacted]	[redacted]	106	[redacted]	119	0.890756302513523
[redacted]	[redacted]	88	[redacted]	119	0.7394957983131134
[redacted]	[redacted]	98	[redacted]	119	0.8235294117577855
[redacted]	[redacted]	95	[redacted]	119	0.7983193277243839
[redacted]	[redacted]	60	[redacted]	119	0.504201680668032
[redacted]	[redacted]	116	[redacted]	119	0.974789915958195
[redacted]	[redacted]	116	[redacted]	119	0.974789915958195
[redacted]	[redacted]	88	[redacted]	119	0.7394957983131134
[redacted]	[redacted]	86	[redacted]	119	0.7226890756241791
[redacted]	[redacted]	116	[redacted]	119	0.974789915958195
[redacted]	[redacted]	109	[redacted]	119	0.9159663865469246

only showing top 20 rows

df_utils get_edges --
df_utils get_edges_domdom_malicious_ones -- df_edges calculado MALICIOUS...

src	dst	edge_weight
[redacted]	[redacted]	0.974789915958195
[redacted]	[redacted]	0.5798319327682366
[redacted]	[redacted]	0.9663865546137278
[redacted]	[redacted]	0.9495798319247934
[redacted]	[redacted]	0.8151260504133182
[redacted]	[redacted]	0.932773109235859
[redacted]	[redacted]	0.7815126050354495
[redacted]	[redacted]	0.8403361344467198
[redacted]	[redacted]	0.8739495798245887
[redacted]	[redacted]	0.890756302513523
[redacted]	[redacted]	0.7394957983131134
[redacted]	[redacted]	0.8235294117577855
[redacted]	[redacted]	0.7983193277243839
[redacted]	[redacted]	0.504201680668032
[redacted]	[redacted]	0.974789915958195
[redacted]	[redacted]	0.974789915958195
[redacted]	[redacted]	0.7394957983131134
[redacted]	[redacted]	0.7226890756241791
[redacted]	[redacted]	0.974789915958195
[redacted]	[redacted]	0.9159663865469246

```
+-----------+------------------+------------------+
only showing top 20 rows
```

Out[99]:
```
DataFrame[src: struct<id:string>, dst: struct<id:string>, edge_weight: double]
```

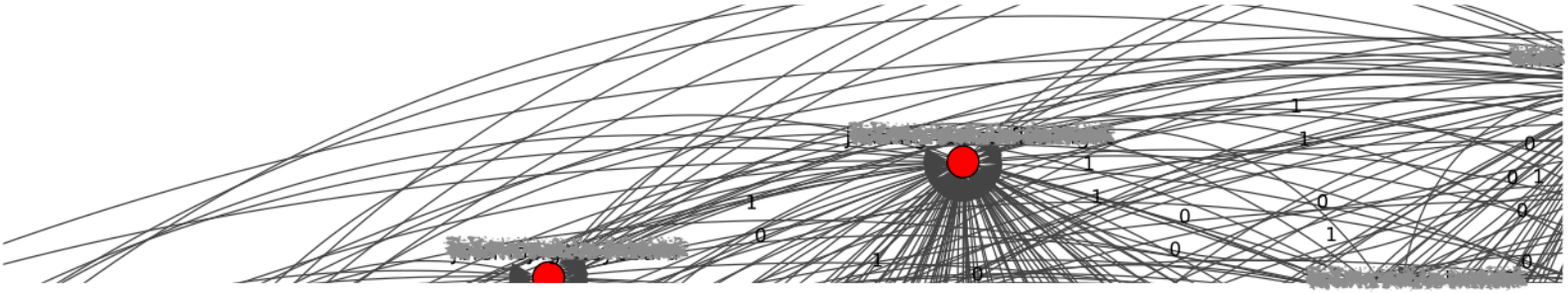
In [26]:
```
gf_domdom_total, gf_domdom_malicious = get_graph_domdom( gf_domip )#.persist()
```

```
DomainDomainGraph get_graph_domdom -- g_domip.edges.show()
gf_utils get_motifs -- df_motifs dropDuplicates( ['e', 'e2']
gf_utils get_motifs_count -- df_motifs_count
gf_utils get_df_degree_ratio --  df_degreeRatio :
df_utils get_edges_domdom -- df que llega ...
df_utils get_edges --
df_utils get_edges_domdom -- df_edges calculado ...
+--------------------+--------------------+--------------------+
|                 src|                 dst|         edge_weight|
+--------------------+--------------------+--------------------+
|[redacted]          |[redacted]          |  0.9807692307503699|
|                    |                    |   0.342857142847347|
|                    |                    |  0.4336283185802334|
|                    |                    | 0.04601226993850917|
|                    |                    | 0.21556886227415828|
|                    |                    | 0.07407407407270233|
|                    |                    | 0.32876712328316754|
|                    |                    |0.028169014084110294|
|                    |                    |  0.8888888887901234|
|                    |                    |  0.2272727272623967|
|                    |                    |  0.9999999989999999|
|                    |                    |      0.999999999875|
|                    |                    |  0.9999999996666666|
|                    |                    |  0.9999999989999999|
|                    |                    | 0.07142857142602041|
|                    |                    | 0.07142857142346938|
|                    |                    | 0.00934579439243602|
|                    |                    |0.012345679012193262|
|                    |                    |     0.0399999999984|
|[redacted]          |[redacted]          |       0.19999999996|
+--------------------+--------------------+--------------------+
only showing top 20 rows

df_utils get_vertices-- :
df_utils get_edges_domdom -- df que llega ...
df_utils get_edges --
df_utils get_edges_domdom_malicious_ones -- df_edges calculado MALICIOUS...
+--------------------+--------------------+-----------+
|                 src|                 dst|edge_weight|
+--------------------+--------------------+-----------+
|[redacted]          |[redacted]          |          1|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          1|
|                    |                    |          0|
|                    |                    |          1|
|                    |                    |          1|
|                    |                    |          1|
|                    |                    |          1|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|[redacted]          |[redacted]          |          0|
+--------------------+--------------------+-----------+
only showing top 20 rows

df_utils get_vertices-- :
```

In [100]:
```
gf_domdom_total, gf_domdom_malicious = get_graph_domdom( gf_domip )  # .persist()

print( "DomainDomainGraph MAIN-- gf_domdom_total.edges.show: " )
gf_domdom_total.edges.show( 10, False )
print( "DomainDomainGraph MAIN-- gf_domdom_malicious.edges.show: " )
gf_domdom_malicious.edges.show( 10, False )
```

```
DomainDomainGraph get_graph_domdom -- g_domip.edges.show()
gf_utils get_motifs -- df_motifs dropDuplicates( ['e', 'e2']
gf_utils get_motifs_count -- df_motifs_count
gf_utils get_df_degree_ratio --  df_degreeRatio :
df_utils get_edges_domdom -- df que llega ...
df_utils get_edges --
df_utils get_edges_domdom -- df_edges calculado ...
+--------------------+--------------------+--------------------+
|                 src|                 dst|         edge_weight|
+--------------------+--------------------+--------------------+
|[redacted]          |[redacted]          |  0.9807692307503699|
|                    |                    |   0.342857142847347|
|                    |                    |  0.4336283185802334|
|                    |                    | 0.04601226993850917|
|                    |                    | 0.21556886227415828|
|                    |                    | 0.07407407407270233|
|                    |                    | 0.32876712328316754|
|                    |                    |0.028169014084110294|
|                    |                    |  0.8888888887901234|
|                    |                    |  0.2272727272623967|
|                    |                    |  0.9999999989999999|
|                    |                    |      0.999999999875|
|                    |                    |  0.9999999996666666|
|                    |                    |  0.9999999989999999|
|                    |                    | 0.07142857142602041|
|                    |                    | 0.07142857142346938|
|                    |                    | 0.00934579439243602|
|                    |                    |0.012345679012193262|
|                    |                    |     0.0399999999984|
|                    |                    |       0.19999999996|
+--------------------+--------------------+--------------------+
only showing top 20 rows

df_utils get_vertices-- :
df_utils get_edges_domdom -- df que llega ...
df_utils get_edges --
df_utils get_edges_domdom_malicious_ones -- df_edges calculado MALICIOUS...
+--------------------+--------------------+-----------+
|                 src|                 dst|edge_weight|
+--------------------+--------------------+-----------+
|[redacted]          |[redacted]          |          1|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          1|
|                    |                    |          0|
|                    |                    |          1|
|                    |                    |          1|
|                    |                    |          1|
|                    |                    |          1|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
|                    |                    |          0|
+--------------------+--------------------+-----------+
only showing top 20 rows

df_utils get_vertices-- :
DomainDomainGraph MAIN-- gf_domdom_total.edges.show:
+--------------------------------+----------------------------------------+--------------------+
|src                             |dst                                     |edge_weight         |
+--------------------------------+----------------------------------------+--------------------+
|[redacted]                      |[redacted]                              |0.9807692307503699  |
|                                |                                        |0.342857142847347   |
|                                |                                        |0.4336283185802334  |
|                                |                                        |0.04601226993850917 |
|                                |                                        |0.21556886227415828 |
|                                |                                        |0.07407407407270233 |
|                                |                                        |0.32876712328316754 |
|                                |                                        |0.028169014084110294|
|                                |                                        |0.8888888887901234  |
|                                |                                        |0.2272727272623967  |
+--------------------------------+----------------------------------------+--------------------+
only showing top 10 rows

DomainDomainGraph MAIN-- gf_domdom_malicious.edges.show:
+--------------------------------+----------------------------------------+-----------+
|src                             |dst                                     |edge_weight|
+--------------------------------+----------------------------------------+-----------+
|[redacted]                      |[redacted]                              |1          |
|                                |                                        |0          |
|                                |                                        |0          |
|                                |                                        |0          |
|                                |                                        |0          |
|                                |                                        |0          |
|                                |                                        |0          |
|                                |                                        |0          |
|                                |                                        |1          |
|                                |                                        |0          |
+--------------------------------+----------------------------------------+-----------+
only showing top 10 rows
```

In [101]:
```
ig_mal, visual_style_mal = draw_igraph_domain_domain( gf_domdom_malicious )
plot( ig_mal, **visual_style_mal)
```



In [103]:
```
sub_graph_malicious_domain_1 = gf_domdom_malicious.filterEdges( "src.id = '[redacted]'")
igl, visual_style1 = draw_igraph_domain_domain( sub_graph_malicious_domain_1 )
plot( igl, **visual_style1)
```

```
draw_utils draw_igraph --
```

Out[103]: