# databricks Graph Example 2023-10-23 11:00:49

(https://databricks.com)

```
%python
# file location and type
file_location = "dbfs:/FileStore/shared_uploads/devjethva234@gmail.com/vertex.csv"
file_type = "csv"

# csv Options
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","

# the applied options are for csv files.for other file types, these will be ignored.
df = spark.read.format(file_type) \
    .option("inferSchema", infer_schema)\
    .option("header",first_row_is_header)\
    .option("sep",delimiter)\
    .load(file_location)

display(df)
```

Table

| | _c0 | _c1 | _c2 |
|---|---|---|---|
| 1 | 1 | Jacob | 48 |
| 2 | 2 | Jessica | 45 |
| 3 | 3 | Andrew | 25 |
| 4 | 4 | Ryan | 53 |
| 5 | 5 | Emily | 22 |
| 6 | 6 | Lily | 52 |

6 rows

```
%python
# file location and type
file_location = "dbfs:/FileStore/shared_uploads/devjethva234@gmail.com/edges.csv"
file_type = "csv"

# csv Options
infer_schema = "false"
first_row_is_header = "false"
delimiter = ","

# the applied options are for csv files.for other file types, these will be ignored.
df1 = spark.read.format(file_type) \
    .option("inferSchema", infer_schema)\
    .option("header",first_row_is_header)\
    .option("sep",delimiter)\
    .load(file_location)

display(df1)
```

Table

| | _c0 | _c1 | _c2 |
|---|---|---|---|
| 1 | 6 | 1 | Sister |
| 2 | 1 | 2 | Husband |
| 3 | 2 | 1 | Wife |
| 4 | 5 | 1 | Daughter |
| 5 | 5 | 2 | Daughter |

| | | | |
|---|---|---|---|
| 6 | 3 | 1 | Son |
| 7 | 3 | 2 | Son |

12 rows

```
%scala
import org.apache.spark.rdd.RDD
```

```
import org.apache.spark.rdd.RDD
```

```
%scala
import org.apache.spark.graphx._
```

```
import org.apache.spark.graphx._
```

```
%scala
val vertexRDD = sc.textFile("dbfs:/FileStore/shared_uploads/devjethva234@gmail.com/vertex.csv")
val edgeRDD = sc.textFile("dbfs:/FileStore/shared_uploads/devjethva234@gmail.com/edges.csv")
edgeRDD.collect()
```

```
vertexRDD: org.apache.spark.rdd.RDD[String] = dbfs:/FileStore/shared_uploads/devjethva234@gmail.com/vertex.csv MapPartitionsRDD[23]
at textFile at command-223217164383677:1
edgeRDD: org.apache.spark.rdd.RDD[String] = dbfs:/FileStore/shared_uploads/devjethva234@gmail.com/edges.csv MapPartitionsRDD[25] at
textFile at command-223217164383677:2
res0: Array[String] = Array(6,1,Sister, 1,2,Husband, 2,1,Wife, 5,1,Daughter, 5,2,Daughter, 3,1,Son, 3,2,Son, 4,1,Friend, 1,5,Father,
1,3,Father, 2,5,Mother, 2,3,Mother)
```

```
%scala
vertexRDD.collect()
```

```
res1: Array[String] = Array(1,Jacob,48, 2,Jessica,45, 3,Andrew,25, 4,Ryan,53, 5,Emily,22, 6,Lily,52)
```

```
%scala
val vertices: RDD[(VertexId, (String, String))] = vertexRDD.map{ line => val fields = line.split(",")
                                                                  (fields(0).toLong, ( fields(1), fields(2)))}
vertices.collect()
```

```
vertices: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, (String, String))] = MapPartitionsRDD[26] at map at command-22
3217164383679:1
res2: Array[(org.apache.spark.graphx.VertexId, (String, String))] = Array((1,(Jacob,48)), (2,(Jessica,45)), (3,(Andrew,25)), (4,(Rya
n,53)), (5,(Emily,22)), (6,(Lily,52)))
```

```
%scala
val edges: RDD[Edge[String]] = edgeRDD.map{ line => val fields = line.split(",")
                                                    Edge(fields(0).toLong, fields(1).toLong, fields(2))}
edges.collect()
```

```
edges: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[String]] = MapPartitionsRDD[27] at map at command-223217164383680:1
res3: Array[org.apache.spark.graphx.Edge[String]] = Array(Edge(6,1,Sister), Edge(1,2,Husband), Edge(2,1,Wife), Edge(5,1,Daughter), E
dge(5,2,Daughter), Edge(3,1,Son), Edge(3,2,Son), Edge(4,1,Friend), Edge(1,5,Father), Edge(1,3,Father), Edge(2,5,Mother), Edge(2,3,Mo
ther))
```

```
%scala
val default = ("unknown", "missing")
val graph = Graph(vertices, edges, default)
```

```
default: (String, String) = (unknown,missing)
graph: org.apache.spark.graphx.Graph[(String, String),String] = org.apache.spark.graphx.impl.GraphImpl@27608e58
```

```
%scala
// joining graphs
case class MoviesWatched(Movie: String, Genre: String)
val movies: RDD[(VertexId, MoviesWatched)] = sc.parallelize(List(
                                                                (1, MoviesWatched("Toy Story 3", "Kids")), (2, MoviesWatched
                                                                ("Titanic", "Love")),
                                                                (3, MoviesWatched("The Hangover ", "Comedy"))
))
```

```
defined class MoviesWatched
movies: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, MoviesWatched)] = ParallelCollectionRDD[40] at parallelize at co
mmand-223217164383682:3
```

```
%scala
val movieOuterJoinedGraph = graph.outerJoinVertices(movies)((_,name, movies) => (name, movies))
```

```
movieOuterJoinedGraph: org.apache.spark.graphx.Graph[((String, String), Option[MoviesWatched]),String] = org.apache.spark.graphx.imp
l.GraphImpl@42a47cbc
```

```
%scala
movieOuterJoinedGraph.vertices.map(t =>t).collect.foreach(println)
```

```
(4,((Ryan,53),None))
(6,((Lily,52),None))
(2,((Jessica,45),Some(MoviesWatched(Titanic,Love))))
(1,((Jacob,48),Some(MoviesWatched(Toy Story 3,Kids))))
(3,((Andrew,25),Some(MoviesWatched(The Hangover ,Comedy))))
(5,((Emily,22),None))
```

```
%scala
val movieOuterJoinedGraph = graph.outerJoinVertices(movies)((_,name, movies) => (name, movies.getOrElse(MoviesWatched("NA",
"NA"))))
```

```
movieOuterJoinedGraph: org.apache.spark.graphx.Graph[((String, String), MoviesWatched),String] = org.apache.spark.graphx.impl.GraphI
mpl@7980553c
```

```
%scala
movieOuterJoinedGraph.vertices.map(t=>t).collect.foreach(println)
```

```
(4,((Ryan,53),MoviesWatched(NA,NA)))
(6,((Lily,52),MoviesWatched(NA,NA)))
(2,((Jessica,45),MoviesWatched(Titanic,Love)))
(1,((Jacob,48),MoviesWatched(Toy Story 3,Kids)))
(3,((Andrew,25),MoviesWatched(The Hangover ,Comedy)))
(5,((Emily,22),MoviesWatched(NA,NA)))
```

```
%scala
val tCount = graph.triangleCount().vertices
```

```
tCount: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[102] at RDD at VertexRDD.scala:57
```

```
%scala
println(tCount.collect().mkString("\n"))
```

```
(4,0)
(6,0)
(2,2)
(1,2)
(3,1)
(5,1)
```

```
%scala
val iterations = 1000
```

```
iterations: Int = 1000
```

```
%scala
val connected = graph.connectedComponents().vertices
```

```
connected: org.apache.spark.graphx.VertexRDD[org.apache.spark.graphx.VertexId] = VertexRDDImpl[126] at RDD at VertexRDD.scala:57
```

```
%scala
val connections = graph.stronglyConnectedComponents(iterations).vertices
```

```
connections: org.apache.spark.graphx.VertexRDD[org.apache.spark.graphx.VertexId] = VertexRDDImpl[377] at RDD at VertexRDD.scala:57
```

```
%scala
val connByPerson = vertices.join(connected).map{ case(id, ((person,age), conn)) => (conn, id,person)}
val connByPersonS = vertices.join(connected).map{ case(id, ((person,age), conn)) => (conn, id,person)}
connByPerson.collect().foreach{ case (conn, id, person) => println(f"Weak $conn $id $person")}
```

```
Weak 1 4 Ryan
Weak 1 6 Lily
Weak 1 2 Jessica
Weak 1 1 Jacob
Weak 1 3 Andrew
Weak 1 5 Emily
connByPerson: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, org.apache.spark.graphx.VertexId, String)] = MapPartitions
RDD[427] at map at command-223217164383692:1
connByPersonS: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, org.apache.spark.graphx.VertexId, String)] = MapPartition
sRDD[431] at map at command-223217164383692:2
```

```
%scala
println("Vertices count:" +graph.vertices.count)
```

```
Vertices count:6
```

```
%scala
println("Vertices count:" +graph.edges.count)
```

```
Vertices count:12
```

```
%scala
val cnt1 = graph.vertices.filter{ case (id,(name, age)) => age.toLong > 40}.count
```

```
cnt1: Long = 4
```

```
%scala
val cnt2 = graph.edges.filter{ case Edge(from ,to, property) => property == "Father" | property == "Mother"}.count
```

```
cnt2: Long = 4
```

```
%scala
def max (a: (VertexId, Int), b: (VertexId, Int)) : (VertexId, Int)={
  if(a._2 > b._2) a else b
}
```

```
max: (a: (org.apache.spark.graphx.VertexId, Int), b: (org.apache.spark.graphx.VertexId, Int))(org.apache.spark.graphx.VertexId, Int)
```

```
%scala
val maxInDegree : (VertexId, Int) = graph.inDegrees.reduce(max)
val maxOutDegree : (VertexId, Int) = graph.outDegrees.reduce(max)
val maxDegree : (VertexId, Int) = graph.degrees.reduce(max)
```

```
maxInDegree: (org.apache.spark.graphx.VertexId, Int) = (1,5)
maxOutDegree: (org.apache.spark.graphx.VertexId, Int) = (2,3)
maxDegree: (org.apache.spark.graphx.VertexId, Int) = (1,8)
```

```
%scala
val minDegrees = graph.outDegrees.filter(_._2 <=1)
minDegrees.collect()
```

```
minDegrees: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[451] at RDD at VertexRDD.scala:57
res10: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((4,1), (6,1))
```

```
%scala
graph.triplets.map(
  triplet => triplet.srcAttr._1 + "is the " + triplet.attr +  "of " + triplet.dstAttr._1
).collect.foreach(println)
```

```
Jacobis the Husbandof Jessica
Jessicais the Wifeof Jacob
Andrewis the Sonof Jacob
Emilyis the Daughterof Jacob
Emilyis the Daughterof Jessica
Lilyis the Sisterof Jacob
Jacobis the Fatherof Andrew
Jacobis the Fatherof Emily
Jessicais the Motherof Andrew
Jessicais the Motherof Emily
Andrewis the Sonof Jessica
Ryanis the Friendof Jacob
```

```
%scala
print(sc.version)
```

```
3.3.2
```

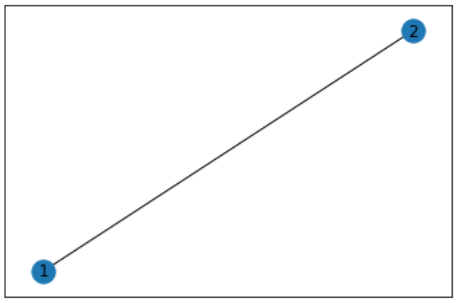
```
%scala
// import org.graphframes._
//  val vertex = spark.createDataFrame(List(
//    ("1", "Jacob", "48"),
//    ("2", "Jessica", "45"),
//    ("3", "Andrew", "25"),
//    ("4", "Jacob", "53"),
//    ("5", "Jaco", "22"),
//    ("6", "Jab", "52"),
//  ).toDF("id", "name", "age"))
```

```
pip install networkx
```

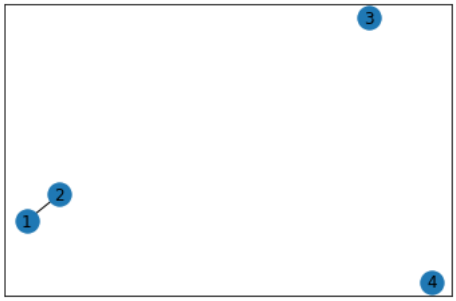
```
Python interpreter will be restarted.
Collecting networkx
  Downloading networkx-3.2-py3-none-any.whl (1.6 MB)
Installing collected packages: networkx
Successfully installed networkx-3.2
Python interpreter will be restarted.
```

```
import networkx as nx
```

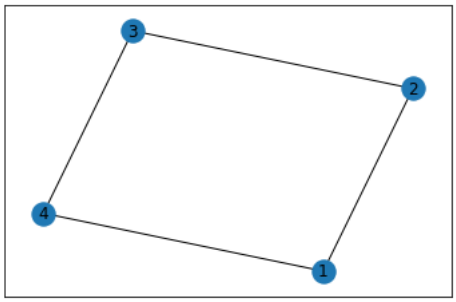
```
# create an Empty undirected graph
G  = nx.Graph()

import matplotlib.pyplot as plt
# adding
G.add_edge(1,2)
nx.draw_networkx(G)
plt.show()
```




```
G.add_nodes_from([3,4])
nx.draw_networkx(G)
plt.show()
```




```
G.add_edge(3,4)
G.add_edges_from([(2,3), (4,1)])
nx.draw_networkx(G)
plt.show()
```




```
G.nodes
```

Out[5]: NodeView((1, 2, 3, 4))

```
G.edges
```

Out[6]: EdgeView([(1, 2), (1, 4), (2, 3), (3, 4)])

```
list(nx.generate_adjlist(G))
```

Out[7]: ['1 2 4', '2 3', '3 4', '4']

```
nx.to_dict_of_lists(G)
```

Out[8]: {1: [2, 4], 2: [1, 3], 3: [4, 2], 4: [3, 1]}

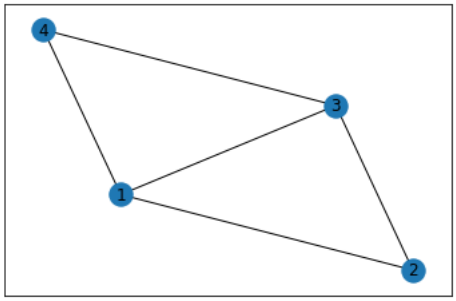
```
nx.to_edgelist(G)
```

Out[10]: EdgeDataView([(1, 2, {}), (1, 4, {}), (2, 3, {}), (3, 4, {})])

```
nx.to_pandas_adjacency(G)
```

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| **1** | 0.0 | 1.0 | 0.0 | 1.0 |
| **2** | 1.0 | 0.0 | 1.0 | 0.0 |
| **3** | 0.0 | 1.0 | 0.0 | 1.0 |
| **4** | 1.0 | 0.0 | 1.0 | 0.0 |

```
G.add_edge(1,3)
nx.draw_networkx(G)
plt.show()
```



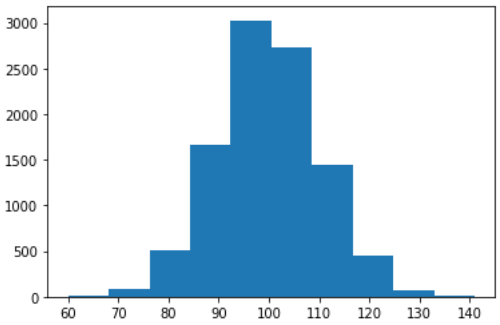

```
# print(nx.to_scipy_sparse_matrix(G))
```

```
G.degree
```

```
Out[13]: DegreeView({1: 3, 2: 2, 3: 3, 4: 2})
```

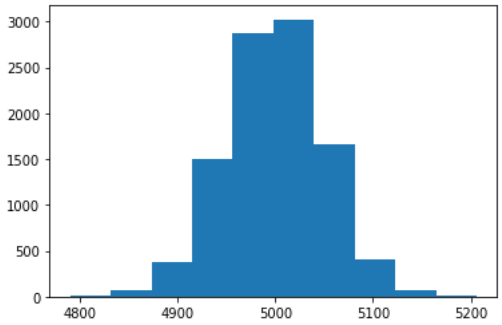
```
k = nx.fast_gnp_random_graph(10000, 0.01).degree()
plt.hist(list(dict(k).values()))
```

```
Out[14]: (array([   5.,   82.,  502., 1671., 3028., 2730., 1450.,  455.,   71.,
           6.]),
 array([ 60. ,  68.1,  76.2,  84.3,  92.4, 100.5, 108.6, 116.7, 124.8,
        132.9, 141. ]),
 <BarContainer object of 10 artists>)
```

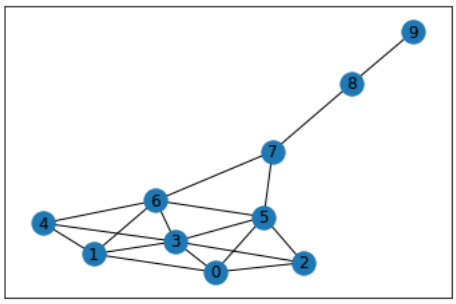



```
k = nx.fast_gnp_random_graph(10000, 0.50).degree()
plt.hist(list(dict(k).values()))
```

```
Out[15]: (array([   6.,   64.,  379., 1507., 2879., 3025., 1660.,  407.,   65.,
          8.]),
 array([4790. , 4831.6, 4873.2, 4914.8, 4956.4, 4998. , 5039.6, 5081.2,
        5122.8, 5164.4, 5206. ]),
 <BarContainer object of 10 artists>)
```




```
G = nx.krackhardt_kite_graph()
nx.draw_networkx(G)
plt.show()
```




```
print(nx.has_path(G, source =1 , target=9))
print(nx.shortest_path(G, source =1 , target=9))
print(nx.shortest_path_length(G, source =1 , target=9))
print(list(nx.shortest_simple_paths(G, source =1 , target=9)))
paths = list(nx.all_pairs_shortest_path(G))
paths[5][1]
```

```
True
[1, 6, 7, 8, 9]
4
[[1, 6, 7, 8, 9], [1, 0, 5, 7, 8, 9], [1, 6, 5, 7, 8, 9], [1, 3, 5, 7, 8, 9], [1, 4, 6, 7, 8, 9], [1, 3, 6, 7, 8, 9], [1, 0, 2,
5, 7, 8, 9], [1, 0, 5, 6, 7, 8, 9], [1, 6, 3, 5, 7, 8, 9], [1, 3, 5, 6, 7, 8, 9], [1, 4, 3, 5, 7, 8, 9], [1, 4, 6, 5, 7, 8, 9],
[1, 3, 0, 5, 7, 8, 9], [1, 3, 6, 5, 7, 8, 9], [1, 0, 3, 5, 7, 8, 9], [1, 4, 3, 6, 7, 8, 9], [1, 3, 2, 5, 7, 8, 9], [1, 0, 3, 6,
7, 8, 9], [1, 3, 4, 6, 7, 8, 9], [1, 0, 2, 3, 5, 7, 8, 9], [1, 0, 2, 5, 6, 7, 8, 9], [1, 0, 5, 3, 6, 7, 8, 9], [1, 6, 4, 3, 5, 7,
8, 9], [1, 6, 3, 0, 5, 7, 8, 9], [1, 4, 3, 5, 6, 7, 8, 9], [1, 4, 6, 3, 5, 7, 8, 9], [1, 3, 0, 2, 5, 7, 8, 9], [1, 3, 0, 5, 6, 7,
8, 9], [1, 0, 3, 5, 6, 7, 8, 9], [1, 4, 3, 0, 5, 7, 8, 9], [1, 4, 3, 6, 5, 7, 8, 9], [1, 3, 2, 0, 5, 7, 8, 9], [1, 3, 2, 5, 6, 7,
8, 9], [1, 0, 3, 2, 5, 7, 8, 9], [1, 0, 3, 6, 5, 7, 8, 9], [1, 3, 4, 6, 5, 7, 8, 9], [1, 0, 2, 3, 6, 7, 8, 9], [1, 6, 3, 2, 5, 7,
8, 9], [1, 4, 3, 2, 5, 7, 8, 9], [1, 0, 3, 4, 6, 7, 8, 9], [1, 0, 2, 3, 5, 6, 7, 8, 9], [1, 0, 2, 5, 3, 6, 7, 8, 9], [1, 0, 5, 2,
3, 6, 7, 8, 9], [1, 0, 5, 3, 4, 6, 7, 8, 9], [1, 6, 4, 3, 0, 5, 7, 8, 9], [1, 6, 3, 0, 2, 5, 7, 8, 9], [1, 4, 6, 3, 0, 5, 7, 8,
9], [1, 3, 0, 2, 5, 6, 7, 8, 9], [1, 4, 3, 0, 2, 5, 7, 8, 9], [1, 4, 3, 0, 5, 6, 7, 8, 9], [1, 3, 2, 0, 5, 6, 7, 8, 9], [1, 0, 3,
2, 5, 6, 7, 8, 9], [1, 0, 2, 3, 4, 6, 7, 8, 9], [1, 0, 2, 3, 6, 5, 7, 8, 9], [1, 6, 3, 2, 0, 5, 7, 8, 9], [1, 4, 3, 2, 0, 5, 7,
8, 9], [1, 4, 3, 2, 5, 6, 7, 8, 9], [1, 0, 3, 4, 6, 5, 7, 8, 9], [1, 6, 4, 3, 2, 5, 7, 8, 9], [1, 4, 6, 3, 2, 5, 7, 8, 9], [1, 0,
2, 5, 3, 4, 6, 7, 8, 9], [1, 0, 5, 2, 3, 4, 6, 7, 8, 9], [1, 6, 4, 3, 0, 2, 5, 7, 8, 9], [1, 4, 6, 3, 0, 2, 5, 7, 8, 9], [1, 4,
3, 0, 2, 5, 6, 7, 8, 9], [1, 0, 2, 3, 4, 6, 5, 7, 8, 9], [1, 4, 3, 2, 0, 5, 6, 7, 8, 9], [1, 6, 4, 3, 2, 0, 5, 7, 8, 9], [1, 4,
6, 3, 2, 0, 5, 7, 8, 9]]
Out[18]: {5: [5],
 0: [5, 0],
 2: [5, 2],
```

```
# importance of nodes inside the network
nx.betweenness_centrality(G)
```

```
Out[19]: {0: 0.023148148148148143,
 1: 0.023148148148148143,
 2: 0.0,
 3: 0.10185185185185183,
 4: 0.0,
 5: 0.23148148148148148,
 6: 0.23148148148148148,
 7: 0.38888888888888884,
 8: 0.2222222222222222,
 9: 0.0}
```

```
nx.degree_centrality(G)
```

```
Out[20]: {0: 0.4444444444444444,
 1: 0.4444444444444444,
 2: 0.3333333333333333,
 3: 0.6666666666666666,
 4: 0.3333333333333333,
 5: 0.5555555555555556,
 6: 0.5555555555555556,
 7: 0.3333333333333333,
 8: 0.2222222222222222,
 9: 0.1111111111111111}
```

```
nx.closeness_centrality(G)
```

```
Out[21]: {0: 0.5294117647058824,
 1: 0.5294117647058824,
 2: 0.5,
 3: 0.6,
 4: 0.5,
 5: 0.6428571428571429,
 6: 0.6428571428571429,
 7: 0.6,
 8: 0.42857142857142855,
 9: 0.3103448275862069}
```

```
nx.harmonic_centrality(G)
```

```
Out[22]: {0: 6.083333333333333,
 1: 6.083333333333333,
 2: 5.583333333333333,
 3: 7.083333333333333,
 4: 5.583333333333333,
 5: 6.833333333333333,
 6: 6.833333333333333,
 7: 6.0,
 8: 4.666666666666666,
 9: 3.4166666666666665}
```

```
nx.eigenvector_centrality(G)
```

```
Out[23]: {0: 0.3522089813920359,
 1: 0.3522089813920358,
 2: 0.28583473531632403,
 3: 0.48102048812210046,
 4: 0.28583473531632403,
 5: 0.3976910106255469,
```

```
6: 0.39769101062554685,
7: 0.19586185175360382,
8: 0.048074775014202924,
9: 0.011164058575824235}
```