GitHub Link :
https://github.com/dheerajreddykomandla/Feature-Engineering

CSCE 5222
Feature Engineering
Feature Engineering on Facebook Dataset
Increment-1

Project Title and Members:

- **Project Title: Feature Engineering on Facebook Dataset**
- **Team Number :** Team 15
- **Team Members:**
  Dheeraj Reddy Komandla (11526265)
  Prem Sai Vuppula (11528103)
  Sravan Boinapalli (11553755)
  Viveksen Naroju (11611662)

Idea Description:

The idea in this project is about feature engineering, it involves adding new features by extracting hidden information from the existing data. Some of the popular feature engineering techniques are One Hot Encoding, TFIDF, and Word2Vec Etc., Here in this project we would like to select a unique dataset and extract the important insights from the data.

Goals and Objectives:

We as a team did lot of Research on this project, when we are going through the project we found couple of challenging tasks. In this project we have choose to perform feature engineering on graph dataset.

The Graph dataset contains only 2 columns source and destination. Here source and destination represents the id's of each social media user. Each row represents the source id following the destination id. It is really where interesting and challenging part to do feature engineering with 2 columns dataset.

Motivation

Generally feature engineering is mostly performed on the structured data or unstructured data like image and audio files. We can find lot of popular techniques available in the internet. Our main motivation in the project is taking a challenging dataset and performing feature engineering on the dataset, because it has only two columns of data. We as a team felt that it will be very interesting to extract hidden features from the dataset having only two columns.

Significance:

Here, in this project the concept is straightforward, we are using feature engineering drastically where we can justify the importance of that. It is a Process of making decisions from the data and to ensure the model is working correctly. In this project data source and the ways the data has been processed and managed in Unique, because the dataset is having only two columns. Extracting few feature from the dataset is very important for building predictive models in machine learning, it is proven that machine learning models do perform well when we can able extract the hidden information from the dataset. It is not just only for predictive analysis it is also a key factor for descriptive analysis. Based on the Extracted features we can perform exploratory data analysis (EDA) and provide significant insights. Such analysis will be helpful to take Business Decisions.

Literature Survey:

In this project we are using kaggle dataset. The dataset was organically provided by the Facebook. This kaggle repository is actually a challenge given by the Facebook to recommend the friends based on the existing friend connections. Each row in the dataset is nothing but the friend connection from source to destinations. Since the connection is like followers and followees, we assume the dataset is related to Instagram.

As we did research, our team found some of the existing solutions for the above dataset the links for those are given below.

https://www.kaggle.com/code/genialgokul1099/social-network-graph-link-prediction

https://www.kaggle.com/code/curioso/link-prediction-facebook

https://www.kaggle.com/code/ajaysh/stackoverflow-tag-prediction

https://www.kaggle.com/code/vohoangbaoduy/lab3-link-prediction

Objectives:

The main objective in this project is to built the feature engineering to the dataset. In general based on the dataset structure and type we have different kinds of feature engineering techniques,

Example 1: If the given dataset is of type text then we will perform the feature engineering techniques like TFIDF, Word2Vec.

Example 2: If the dataset is of type image we will convert the image into numpy array.

Example 3: If the dataset is of type audio file then we have special feature engineering technique called NFCC Features.

In the same way for the graph dataset, we have some graph data specific features engineering techniques. Our main objective is to implement those feature engineering on our Facebook graph dataset. In the part of our research we found some of the graph feature engineering techniques.

- jaccard followers
- jaccard followees
- cosine followers
- cosine followees
- number of followers source
- number of followees source
- number of followers destination
- number of followees destination
- is following back

➢ shortest path between source and destination

These are the initial feature Engineering techniques we would like to implement on top the given Facebook dataset. If you find future more techniques we will implement those techniques as well.

Features:

As we already discussed above, we are working on the graph dataset. General graph dataset Will have nodes and edges. Hence in our dataset we have only two features. They are:

- Source
- Destination

The important point to be noted in the given graph dataset is that the edges are not bidirectional. This represents there is no rule that if a follows b and b follows a. These are some of the important characteristics.

Related Work(Background):

This dataset was offered by Facebook as part of a kaggle competition. The goal of the kaggle challenge is to promote friends to the user. In other words, we must forecast graph links. As a result, the resources to which we have mentioned are mostly employed for graph link prediction. However, instead of predicting, our goal here is to extract the traits. All of the blogs we've read about this issue focused on tackling the social media link prediction problem.

We also conducted some study on several graph feature engineering strategies. The sources we used to learn about graph feature engineering techniques are

https://networkx.org/documentation/stable/tutorial.html

https://www.learndatasci.com/glossary/jaccard-similarity/#:~:text=The%20Jaccard%20similarity%20measures%20the,of%20observations%20in%20either%20set.

https://docs.tigergraph.com/graph-ml/current/similarity-algorithms/cosine-similarity-of-neighborhoods-single-source

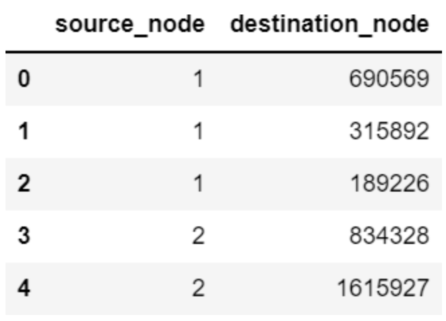
Data Set:

Detail description of Dataset:

The dataset for this project was obtained from the kaggle repository. Facebook actually gave the dataset. It is essentially a Facebook challenge to predict the link between the social media accounts. This is simply the social media buddy referral dilemma. Whoever comes up with the greatest solution, Facebook offers them a job with a large salary. Find the link to the Facebook dataset kaggle repository.

https://www.kaggle.com/competitions/FacebookRecruiting

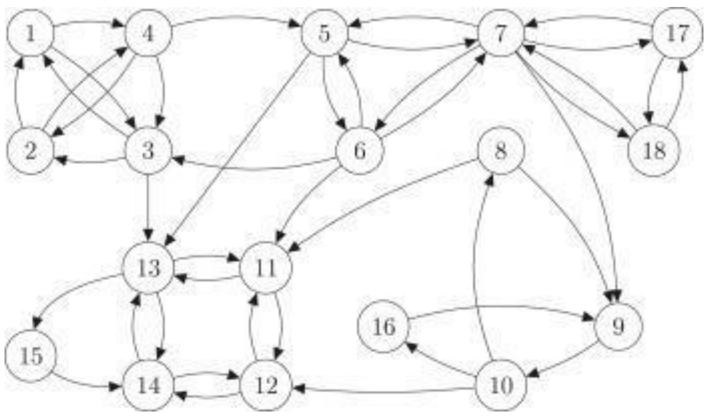
The dataset is bidirectional graph dataset. In the dataset we have only columns 1. source_node and 2. destination_node. Each row represents link between source to destination. In the below screenshot, we have printed the first 5 rows of the dataset.



	source_node	destination_node
0	1	690569
1	1	315892
2	1	189226
3	2	834328
4	2	1615927

We are convinced that this dataset is from the Instagram application. Because the Facebook application requires us to become friends, the link must be unidirectional. In the Instagram app, we normally follow a user and vice versa. In our scenario, we have a bidirectional dataset, thus we infer it is from the Instagram application.

The diagram of a bidirectional graph



Detail Design of Features:

As previously explained, we simply have two columns: source node and destination node. Extracting features using only columns is an interesting and difficult topic. Because we have the Instagram dataset, we should concentrate on the followers and followees. Obtaining information about the source and destination nodes for each row will undoubtedly add value to the collection. The following are the features that we have so far extracted. All of the features we collected provide us with valuable information about source and destination nodes.

1. Number of Source node Followers :

In this feature we are extracting number of followers does the Source node have.

2. Number of Destination node Followers :

In this feature we are extracting number of followers does the Destination node have.

3. Number of Source node Followees :

In this feature we are extracting number of followees does the Source node have.

4. Number of Destination node Followees :

In this feature we are extracting number of followees does the Destination node have.

5. Number of common Followers :
In this feature we are extracting the common followers between Source node and Destination node.

6. Number of common Followees :
In this feature we are extracting the common followees between Source node and Destination node.

7. Jaccard Similarity:

The Jaccard Similarity, also called the Jaccard Index or Jaccard Similarity Coefficient, is a classic measure of similarity between two sets. Given two sets, A and B, the Jaccard Similarity is defined as the size of the intersection of set A and set B (i.e. the number of common elements) over the size of the union of set A and set B (i.e. the number of unique elements).

$$js(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

8. Cosine Similarity :

This algorithm calculates the similarity between a given vertex and every other vertex in the graph using cosine similarity.

$$cos(A,B) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum_i A_i B_i}{\sqrt{\sum_i A_i^2}\sqrt{\sum_i B_i^2}}$$

9. Is Followed Back :

This feature represents if the Destination node followed back Source node.

10. Shortest_path_length:

This feature represents the shortest path length to reach from source node to destination node. It ignores the current link between the source and destination. If there is no path at all, it returns -1.

11. Page_rank:

The PageRank algorithm measures the importance of each node within the graph, based on the number incoming relationships and the importance of the corresponding source nodes. The underlying assumption roughly speaking is that a page is only as important as the pages that link to it.

$$PR(A) = (1-d) + d(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)})$$

12. Hits_Score:

Hyperlink Induced Topic Search (HITS) is an algorithm used in link analysis. It could discover and rank the web pages relevant for a particular search. The idea of this algorithm originated from the fact that an ideal website should link to other relevant sites and also being linked by other important sites.

HITS algorithm returns us the 2 values per each node. 1. Authorities and 2. Hubs.
Authority: A node is high-quality if many high-quality nodes link to it
Hub: A node is high-quality if it links to many high-quality nodes.

13. Degree_centrality:

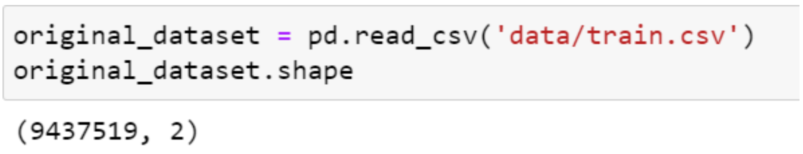
Degree is a simple centrality measure that counts how many neighbors a node has. If the network is directed, we have two versions of the measure: in-degree is the number of in-coming links, or the number of predecessor nodes; out-degree is the number of out-going links, or the number of successor nodes. Typically, we are interested in in-degree, since in-links are given by other nodes in the network, while out-links are determined by the node itself.

14.Out_degree_centrality:

Compute the out-degree centrality for nodes.The out-degree centrality for a node v is the fraction of nodes its outgoing edges are connected to.

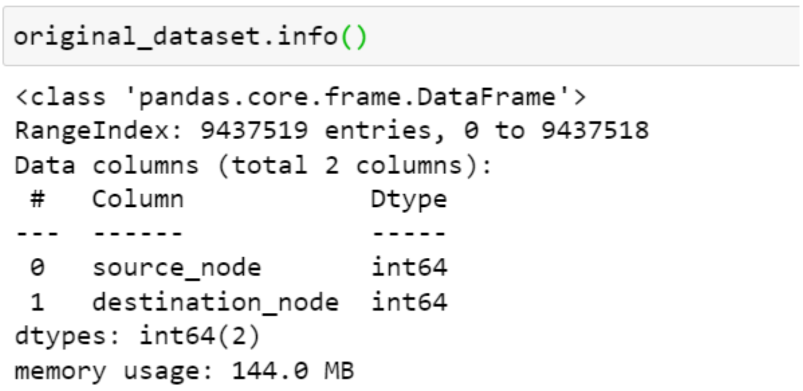
Analysis:

In this section, lets discuss basic analysis that we have done before doing the Feature Engineering. In general the dataset released by Facebook is very huge. It contains more than 9 million records in the dataset.



```
original_dataset = pd.read_csv('data/train.csv')
original_dataset.shape
```

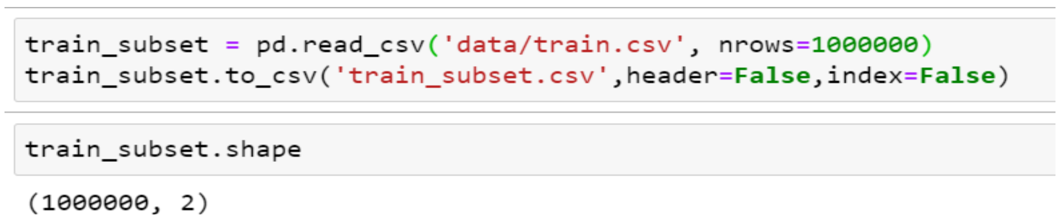
```
(9437519, 2)
```



```
original_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9437519 entries, 0 to 9437518
Data columns (total 2 columns):
 #   Column            Dtype
---  ------            -----
 0   source_node       int64
 1   destination_node  int64
dtypes: int64(2)
memory usage: 144.0 MB
```

In order to process these many records , it is taking huge time for the computation. As we are only doing it for educational purpose we have decided to take the subset of the original dataset. We have sampled the dataset with 1 million records. Lets try to do some analysis of our sampled dataset.

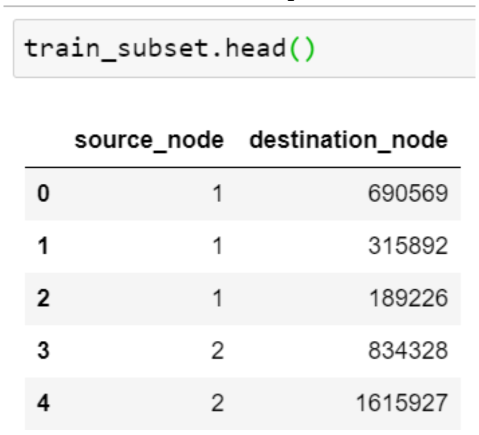


```
train_subset = pd.read_csv('data/train.csv', nrows=1000000)
train_subset.to_csv('train_subset.csv',header=False,index=False)
```

```
train_subset.shape
```

```
(1000000, 2)
```

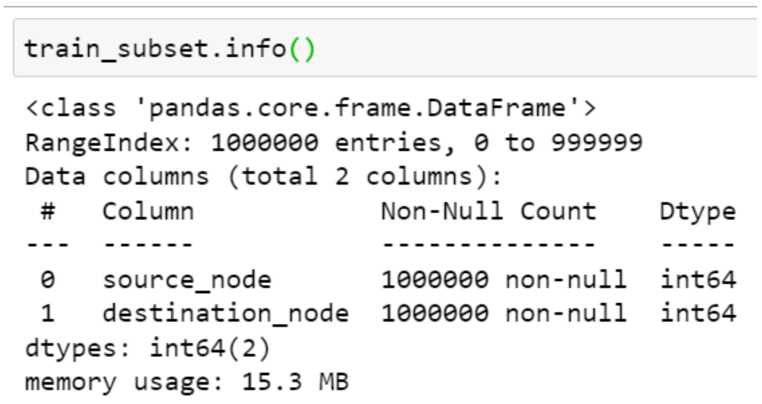
The below screenshot represents the initial 5 rows of the sampled dataset.



```
train_subset.head()
```

	source_node	destination_node
0	1	690569
1	1	315892
2	1	189226
3	2	834328
4	2	1615927

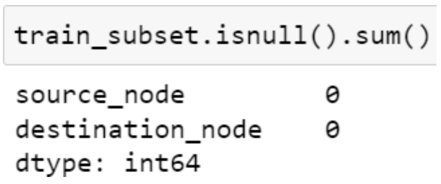
The below screenshot provides us the basic information like column name, count datatype etc.



```
train_subset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 2 columns):
 #   Column            Non-Null Count    Dtype
---  ------            --------------    -----
 0   source_node       1000000 non-null  int64
 1   destination_node  1000000 non-null  int64
dtypes: int64(2)
memory usage: 15.3 MB
```

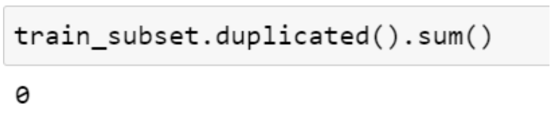
In the below screenshot , we have tested if the dataset is having any null values.



```
train_subset.isnull().sum()
```

```
source_node         0
destination_node    0
dtype: int64
```

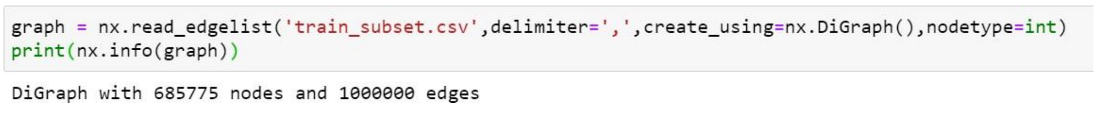
In the below screenshot , we have tested is there any duplicated rows available or not.



```
train_subset.duplicated().sum()
```

```
0
```

In the below screenshot we have loaded the dataset into graph and printed number of nodes and number of edges.



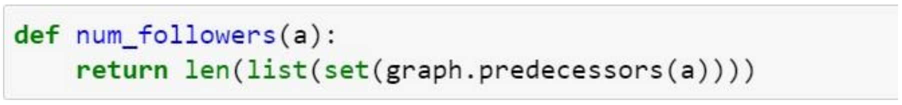
```
graph = nx.read_edgelist('train_subset.csv',delimiter=',',create_using=nx.DiGraph(),nodetype=int)
print(nx.info(graph))
```

```
DiGraph with 685775 nodes and 1000000 edges
```

Implementation:

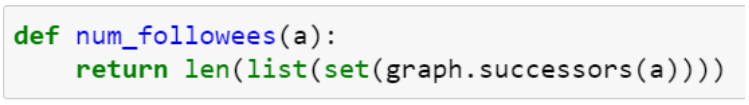
In this section we are pasting the screenshots of the code that we have used to create each feature.

1.Number of Followers :



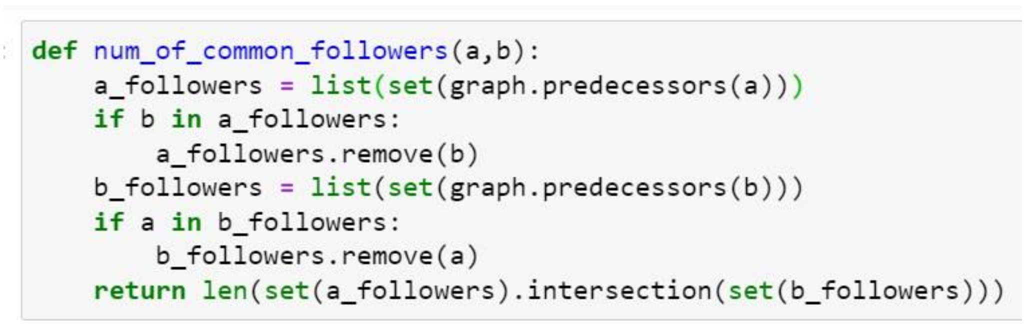
```python
def num_followers(a):
    return len(list(set(graph.predecessors(a))))
```

2.Number of Followees :



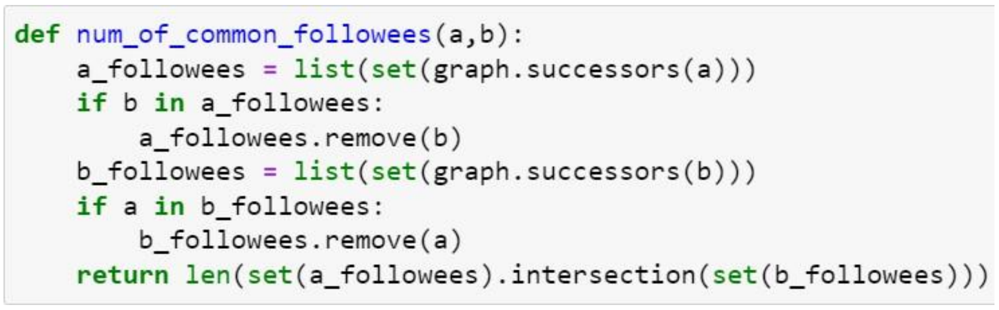
```python
def num_followees(a):
    return len(list(set(graph.successors(a))))
```

3.Number of common Followers :



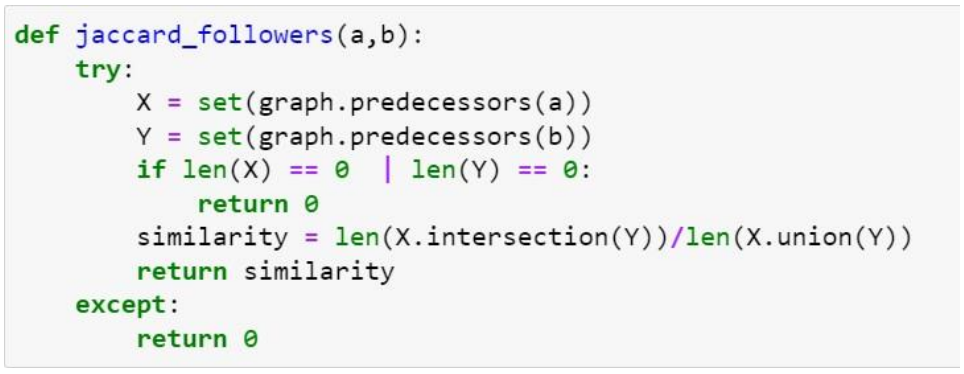
```python
def num_of_common_followers(a,b):
    a_followers = list(set(graph.predecessors(a)))
    if b in a_followers:
        a_followers.remove(b)
    b_followers = list(set(graph.predecessors(b)))
    if a in b_followers:
        b_followers.remove(a)
    return len(set(a_followers).intersection(set(b_followers)))
```

4. Number of common Followees :



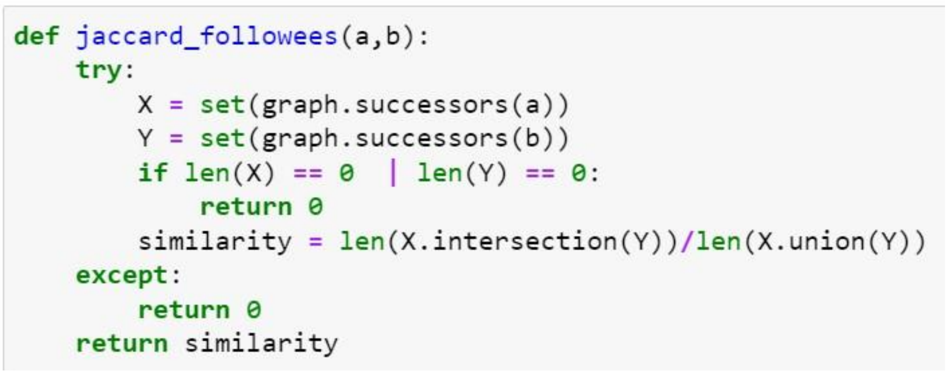
```python
def num_of_common_followees(a,b):
    a_followees = list(set(graph.successors(a)))
    if b in a_followees:
        a_followees.remove(b)
    b_followees = list(set(graph.successors(b)))
    if a in b_followees:
        b_followees.remove(a)
    return len(set(a_followees).intersection(set(b_followees)))
```

5. Jaccard Similarity for Followers :



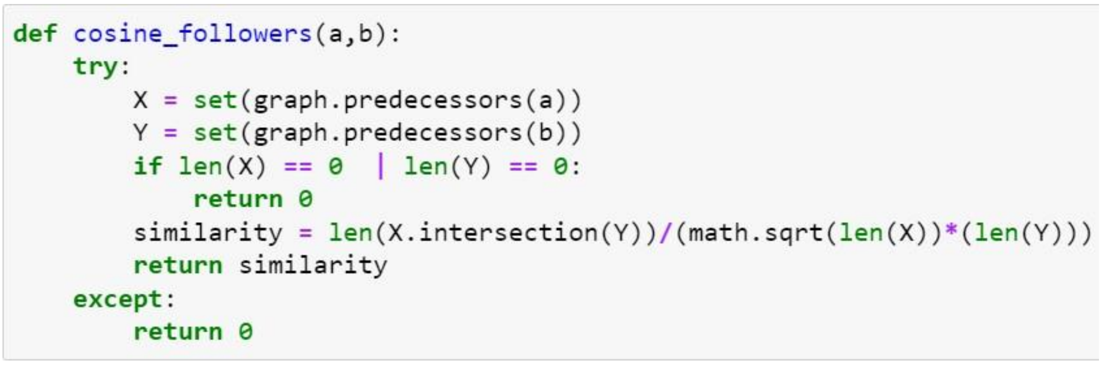
```python
def jaccard_followers(a,b):
    try:
        X = set(graph.predecessors(a))
        Y = set(graph.predecessors(b))
        if len(X) == 0  | len(Y) == 0:
            return 0
        similarity = len(X.intersection(Y))/len(X.union(Y))
        return similarity
    except:
        return 0
```

6.Jaccard Similarity for Followees :



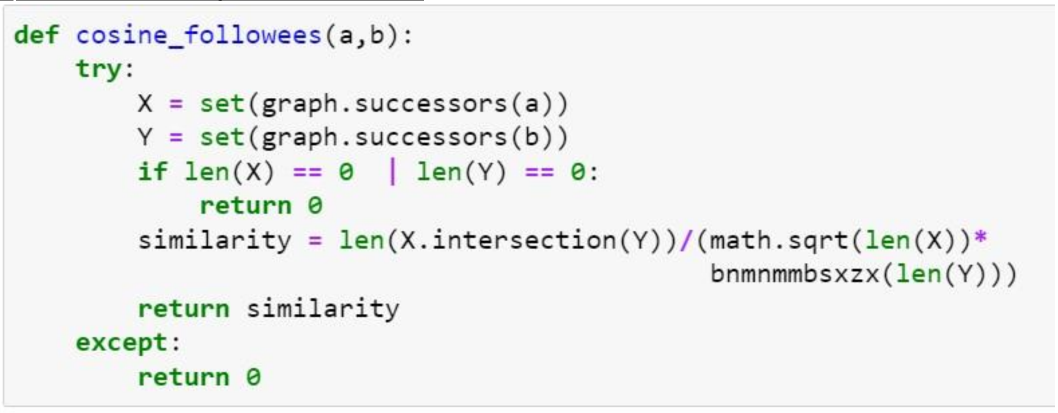
```
def jaccard_followees(a,b):
    try:
        X = set(graph.successors(a))
        Y = set(graph.successors(b))
        if len(X) == 0  | len(Y) == 0:
            return 0
        similarity = len(X.intersection(Y))/len(X.union(Y))
    except:
        return 0
    return similarity
```

7.Cosine Similarity for Followers:



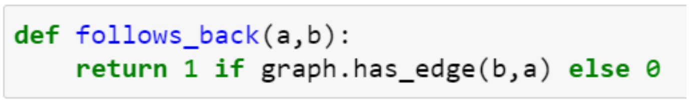
```
def cosine_followers(a,b):
    try:
        X = set(graph.predecessors(a))
        Y = set(graph.predecessors(b))
        if len(X) == 0  | len(Y) == 0:
            return 0
        similarity = len(X.intersection(Y))/(math.sqrt(len(X))*(len(Y)))
        return similarity
    except:
        return 0
```

8.Cosine Similarity for Followees :



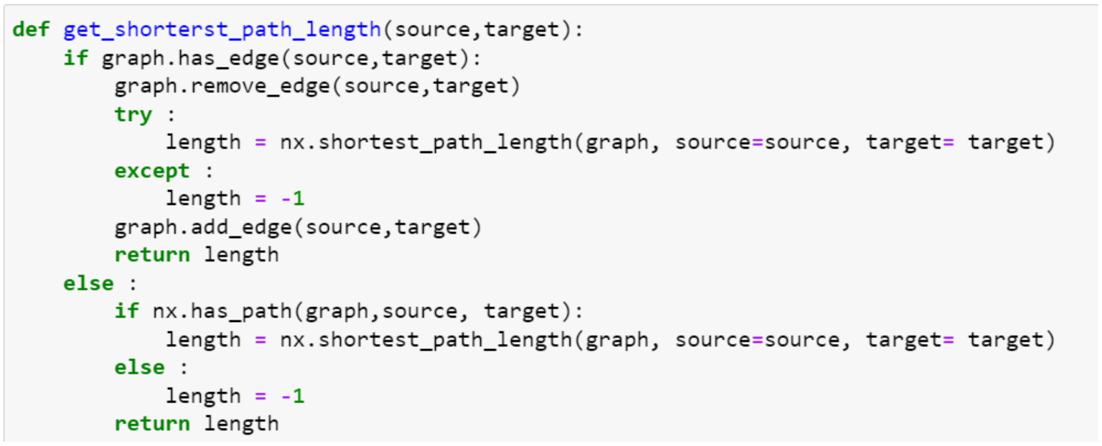
```
def cosine_followees(a,b):
    try:
        X = set(graph.successors(a))
        Y = set(graph.successors(b))
        if len(X) == 0  | len(Y) == 0:
            return 0
        similarity = len(X.intersection(Y))/(math.sqrt(len(X))*
                                              bnmnmmbsxzx(len(Y)))
        return similarity
    except:
        return 0
```

9.Is Followed Back:



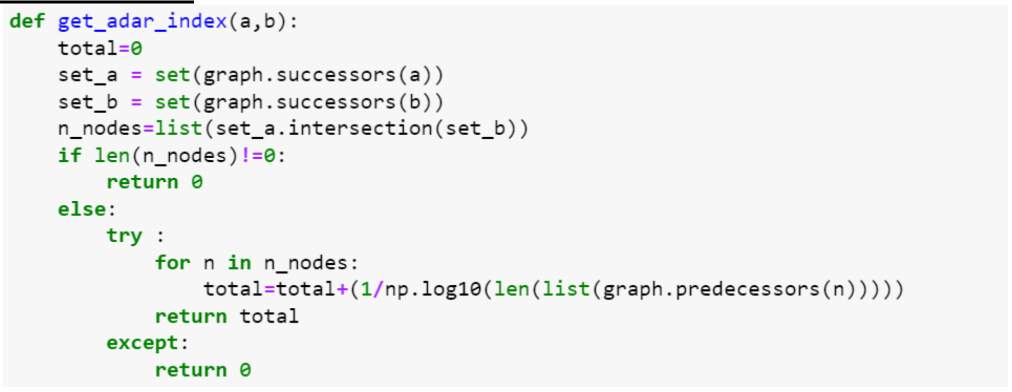
```
def follows_back(a,b):
    return 1 if graph.has_edge(b,a) else 0
```

10.Shortest path length:



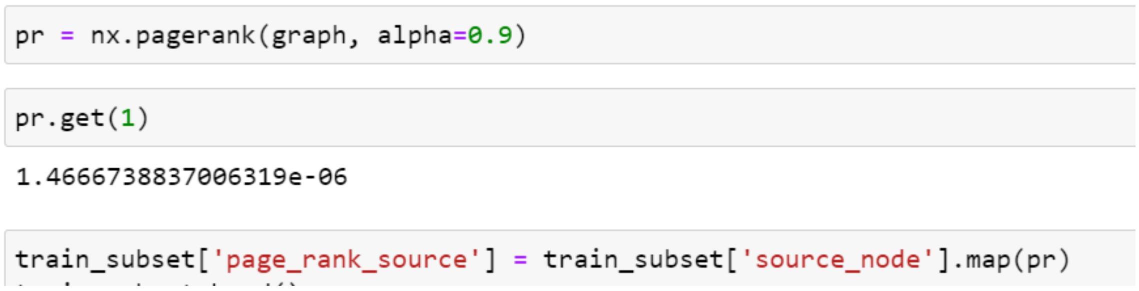
```
def get_shorterst_path_length(source,target):
    if graph.has_edge(source,target):
        graph.remove_edge(source,target)
        try :
            length = nx.shortest_path_length(graph, source=source, target= target)
        except :
            length = -1
        graph.add_edge(source,target)
        return length
    else :
        if nx.has_path(graph,source, target):
            length = nx.shortest_path_length(graph, source=source, target= target)
        else :
            length = -1
        return length
```

11. Adar Index:



```
def get_adar_index(a,b):
    total=0
    set_a = set(graph.successors(a))
    set_b = set(graph.successors(b))
    n_nodes=list(set_a.intersection(set_b))
    if len(n_nodes)!=0:
        return 0
    else:
        try :
            for n in n_nodes:
                total=total+(1/np.log10(len(list(graph.predecessors(n)))))
            return total
        except:
            return 0
```

12.Page rank:



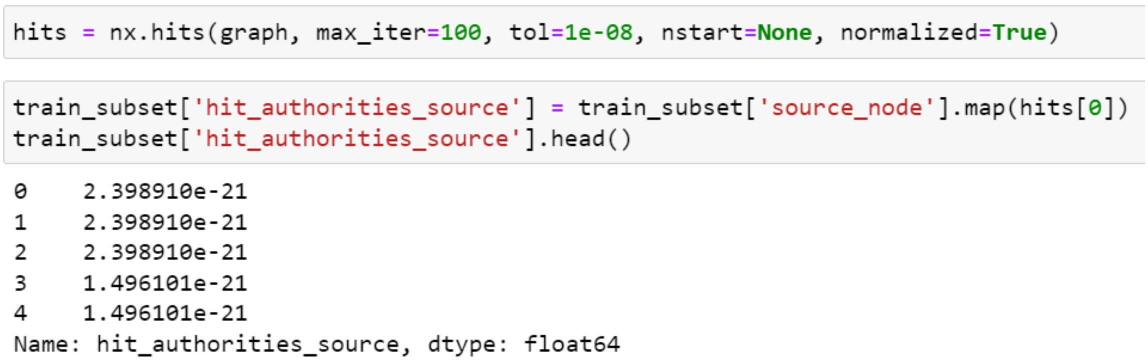
```
pr = nx.pagerank(graph, alpha=0.9)
```

```
pr.get(1)
```

```
1.4666738837006319e-06
```

```
train_subset['page_rank_source'] = train_subset['source_node'].map(pr)
```

13.HITS Score:

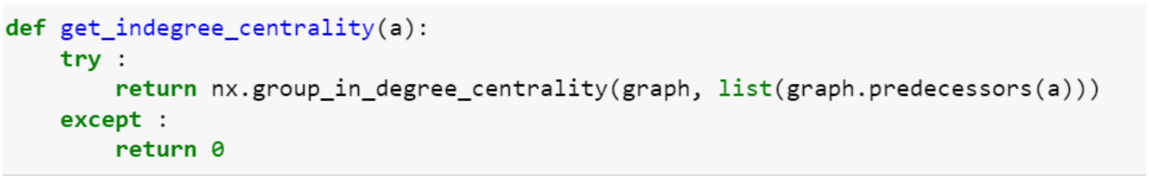


```
hits = nx.hits(graph, max_iter=100, tol=1e-08, nstart=None, normalized=True)
```

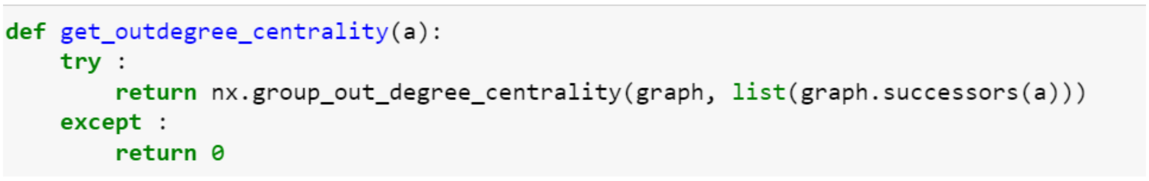
```
train_subset['hit_authorities_source'] = train_subset['source_node'].map(hits[0])
train_subset['hit_authorities_source'].head()
```

```
0     2.398910e-21
1     2.398910e-21
2     2.398910e-21
3     1.496101e-21
4     1.496101e-21
Name: hit_authorities_source, dtype: float64
```

14. Indegree and Out degree centrality :

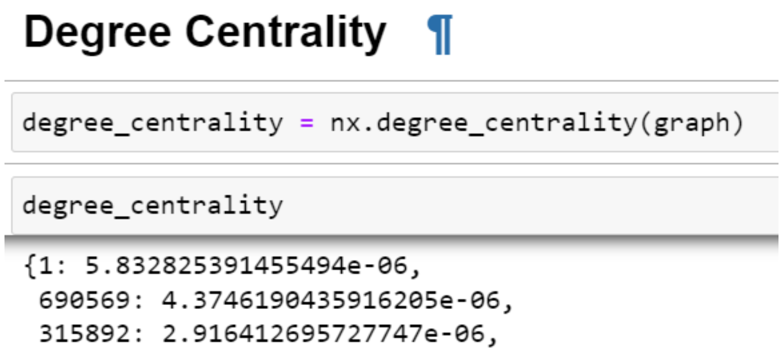


```
def get_indegree_centrality(a):
    try :
        return nx.group_in_degree_centrality(graph, list(graph.predecessors(a)))
    except :
        return 0
```



```
def get_outdegree_centrality(a):
    try :
        return nx.group_out_degree_centrality(graph, list(graph.successors(a)))
    except :
        return 0
```

15. Degree Centrality:



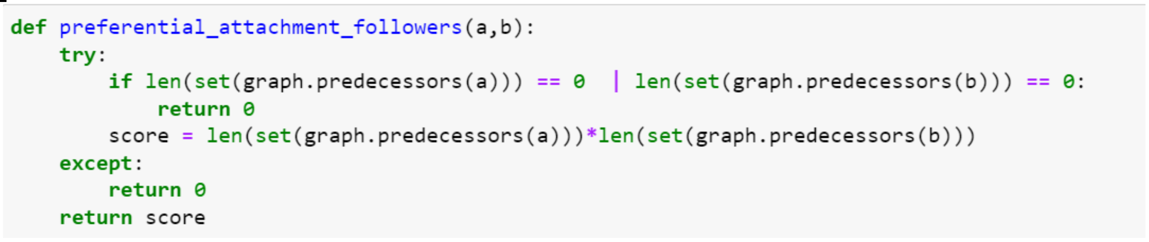
Degree Centrality ¶

```
degree_centrality = nx.degree_centrality(graph)
```

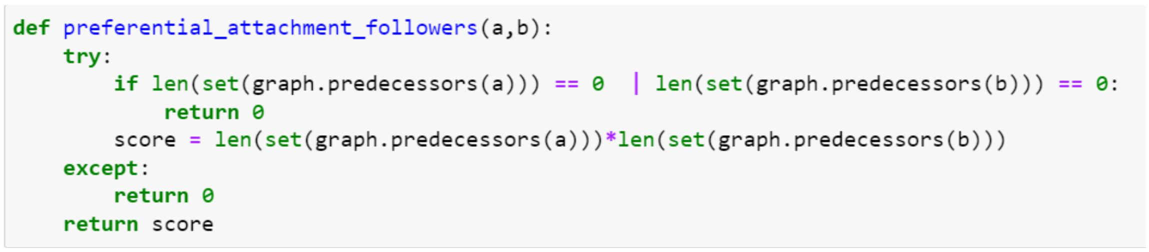
```
degree_centrality
```

```
{1: 5.832825391455494e-06,
 690569: 4.3746190435916205e-06,
 315892: 2.916412695727747e-06,
```

16.Prefeerential Attachment:



```
def preferential_attachment_followers(a,b):
    try:
        if len(set(graph.predecessors(a))) == 0  | len(set(graph.predecessors(b))) == 0:
            return 0
        score = len(set(graph.predecessors(a)))*len(set(graph.predecessors(b)))
    except:
        return 0
    return score
```



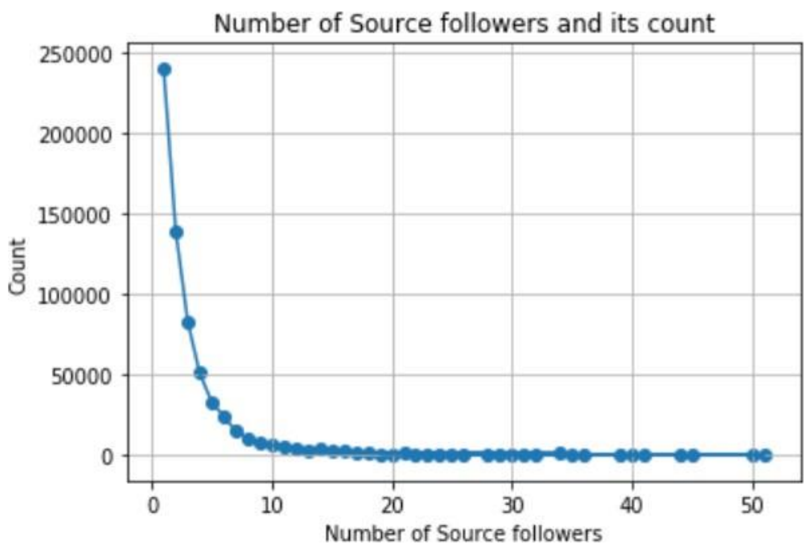
```
def preferential_attachment_followers(a,b):
    try:
        if len(set(graph.predecessors(a))) == 0  | len(set(graph.predecessors(b))) == 0:
            return 0
        score = len(set(graph.predecessors(a)))*len(set(graph.predecessors(b)))
    except:
        return 0
    return score
```

Preliminary Results:

Note : for all the line charts , we have excluded the count of value 0 in order to make plot understandable by removing outlier.

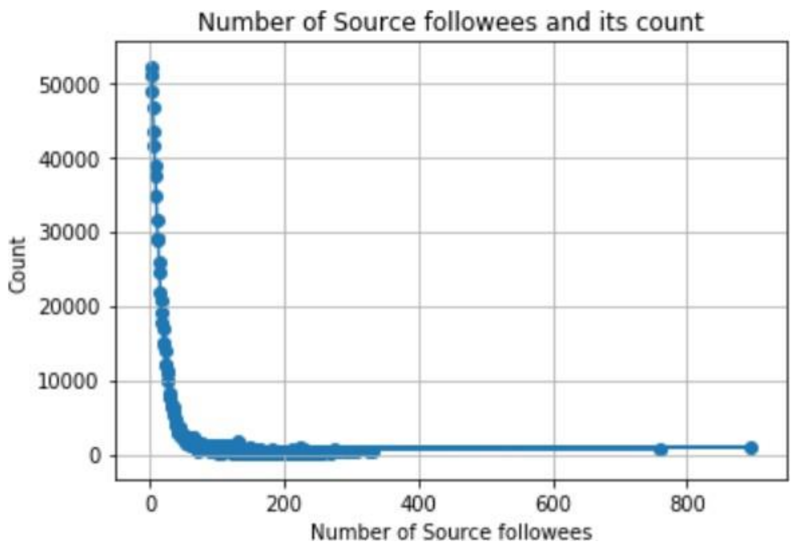
1.Number of Source Followers :

The below screenshots represents number of Source followers and its corresponding count.



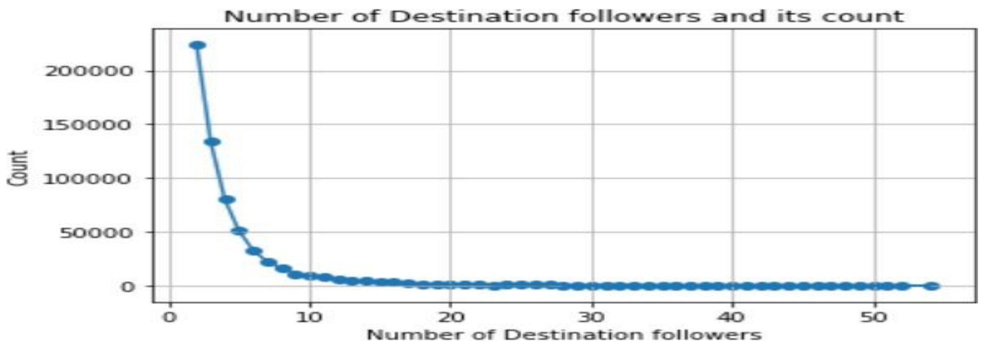
2. **<u>Number of Source Followees :</u>**

The below screenshots represents number of Source followees and its corresponding count.



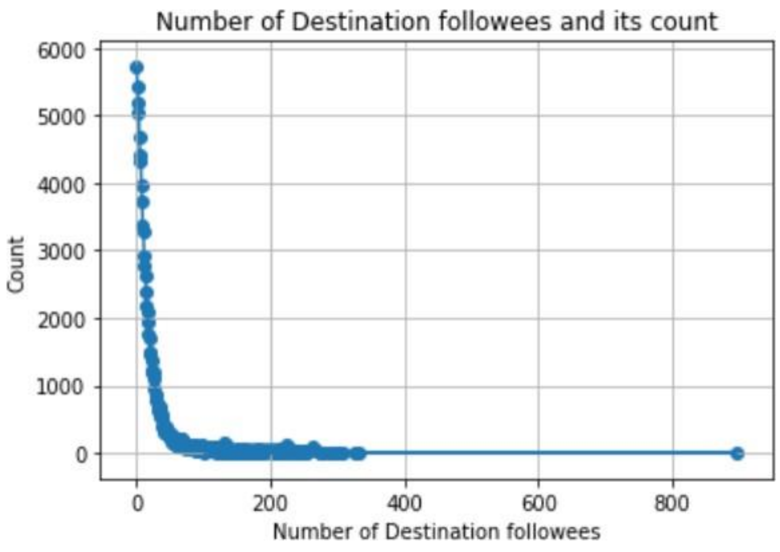
3. **<u>Number of Destination node Followers :</u>**

The below screenshot represents the Destination followers and its corresponding count.



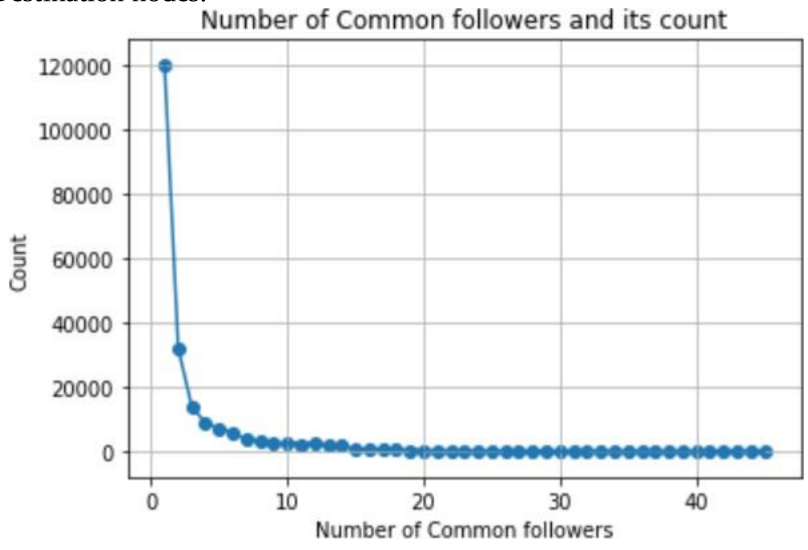
4. Number of Destination node Followees :

The below screenshot represents the Destination followees and its corresponding count.



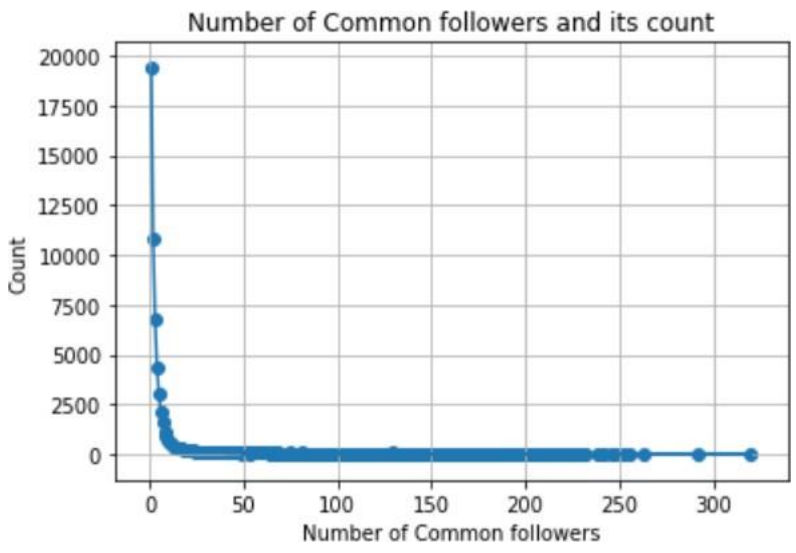
5. Number of common Followers :

The below screenshot represents number of common followers between Source and Destination nodes.



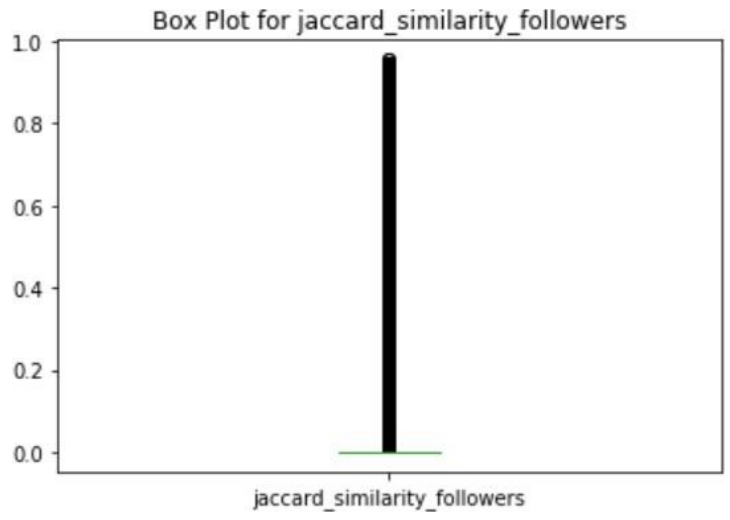
6. Number of common Followees :

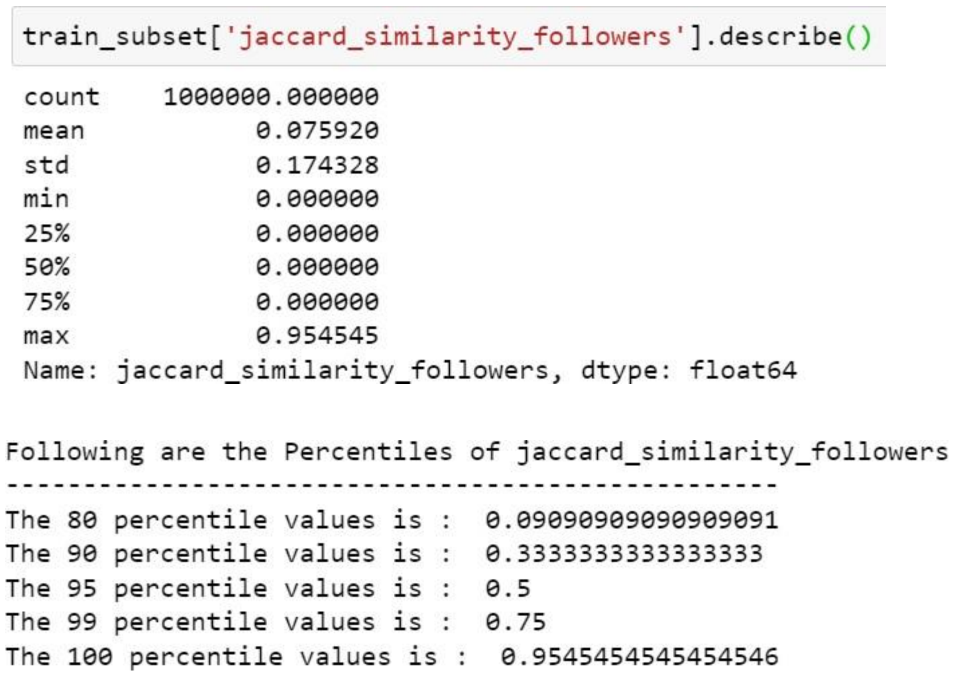
The below screenshot represents number of common followees between Source and Destination nodes.



7. <u>Jaccard Similarity for Followers :</u>

In the below screenshot represents the box plot for Jaccard similarity for followers.



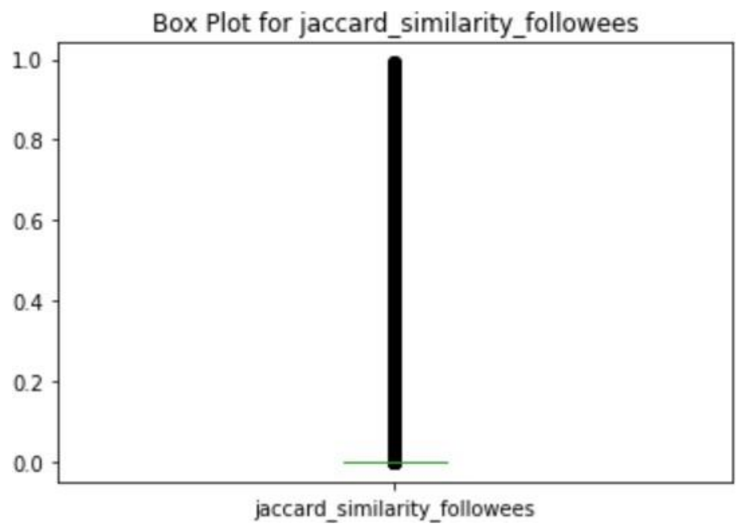


```
train_subset['jaccard_similarity_followers'].describe()
```

```
count    1000000.000000
mean           0.075920
std            0.174328
min            0.000000
25%            0.000000
50%            0.000000
75%            0.000000
max            0.954545
Name: jaccard_similarity_followers, dtype: float64
```

```
Following are the Percentiles of jaccard_similarity_followers
--------------------------------------------------
The 80 percentile values is :  0.09090909090909091
The 90 percentile values is :  0.3333333333333333
The 95 percentile values is :  0.5
The 99 percentile values is :  0.75
The 100 percentile values is :  0.9545454545454546
```

8. <u>Jaccard Similarity for Followees :</u>

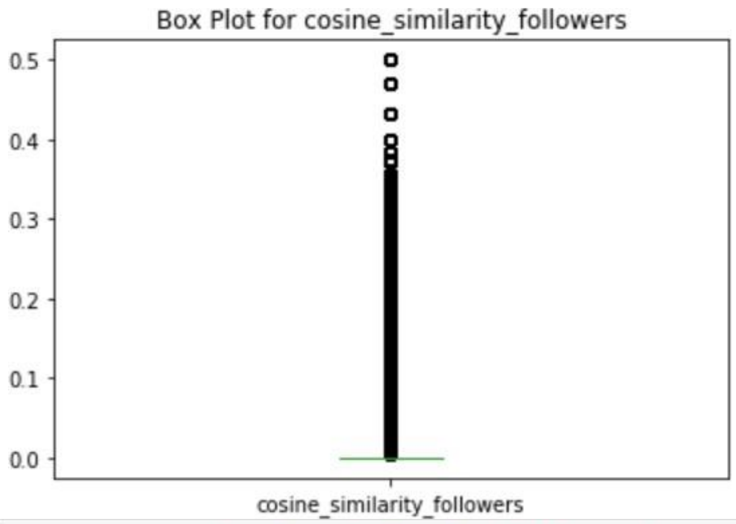


```
train_subset['jaccard_similarity_followees'].describe()
```

```
count    1000000.000000
mean           0.011588
std            0.068378
min            0.000000
25%            0.000000
50%            0.000000
75%            0.000000
max            0.991379
Name: jaccard_similarity_followees, dtype: float64
```

```
Following are the Percentiles of jaccard_similarity_followees
--------------------------------------------------
The 80 percentile values is :  0.0
The 90 percentile values is :  0.0
The 95 percentile values is :  0.04
The 99 percentile values is :  0.3333333333333333
The 100 percentile values is :  0.9913793103448276
```

9. __Cosine Similarity for Followers:__

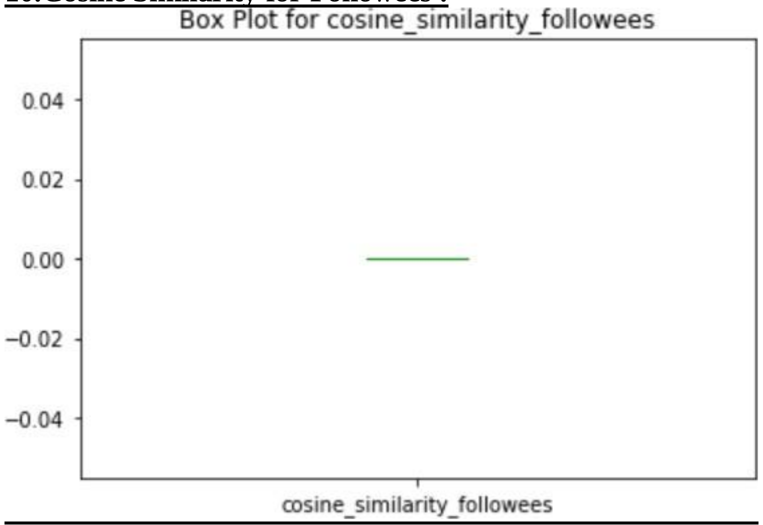


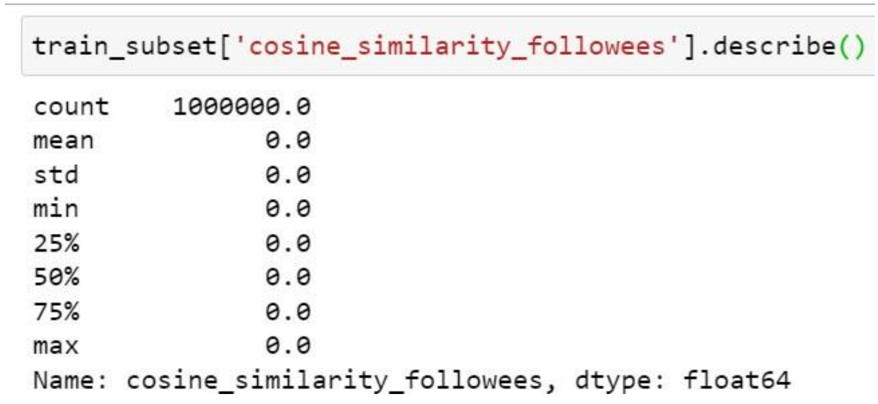
```
train_subset['cosine_similarity_followers'].describe()
```

```
count    1000000.000000
mean           0.054660
std            0.121311
min            0.000000
25%            0.000000
50%            0.000000
75%            0.000000
max            0.500000
Name: cosine_similarity_followers, dtype: float64
```

```
Following are the Percentiles of cosine_similarity_followers
--------------------------------------------------
The 80 percentile values is :  0.07155417527999328
The 90 percentile values is :  0.25
The 95 percentile values is :  0.35355339059327373
The 99 percentile values is :  0.5
The 100 percentile values is :  0.5
```

10.Cosine Similarity for Followees :

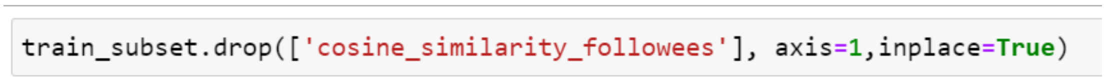




```
train_subset['cosine_similarity_followees'].describe()
```

```
count    1000000.0
mean           0.0
std            0.0
min            0.0
25%            0.0
50%            0.0
75%            0.0
max            0.0
Name: cosine_similarity_followees, dtype: float64
```

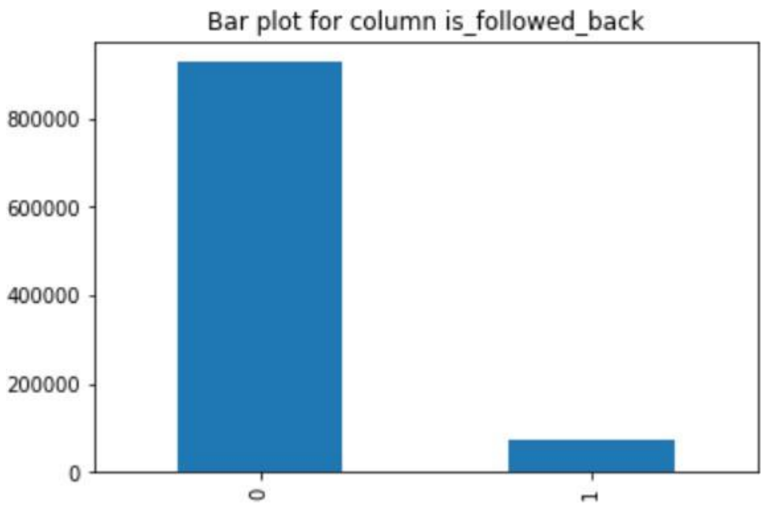
The above screenshots shows us , there are no cosine similarities in terms of followees. Hence we have deleted the column.



```
train_subset.drop(['cosine_similarity_followees'], axis=1,inplace=True)
```

11.Is Followed Back :

The below screenshot represents the bar plot of is-followed-back.

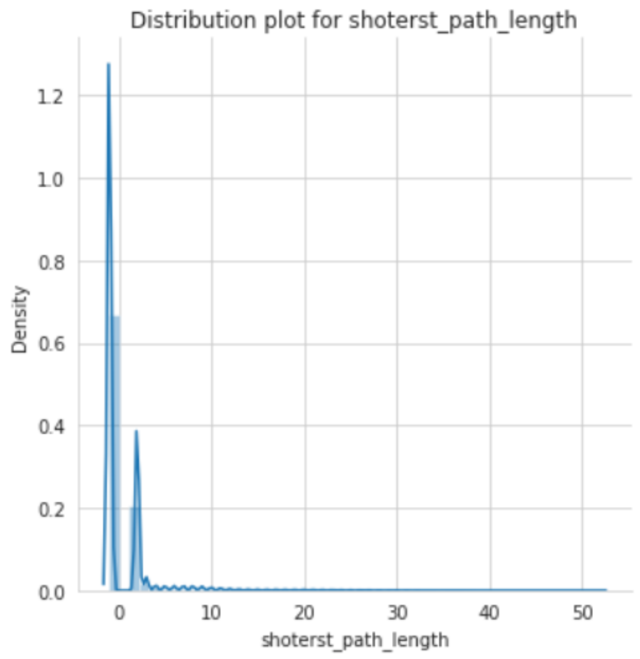


12. The final created features :

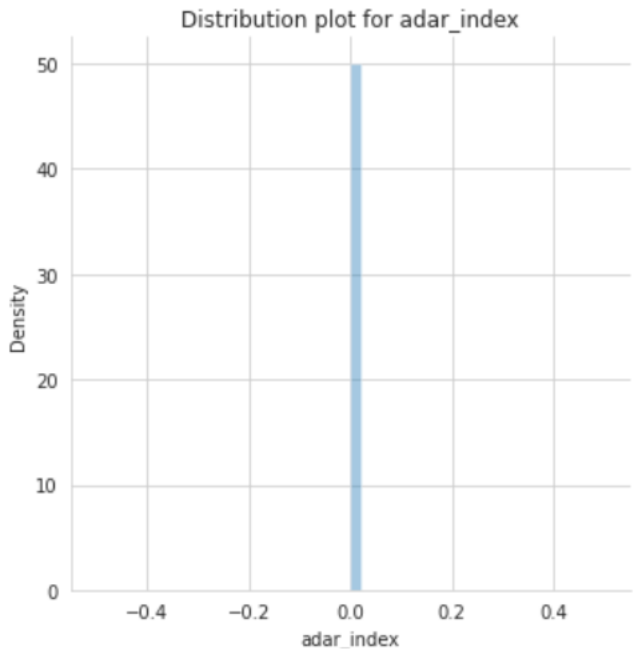
```
print("The final Feature we have so far:")
print("-"*50)
for column in train_subset.columns:
    print(column)
print("-"*50)
print("Total created features are :",len(train_subset.columns)-2)
```

```
The final Feature we have so far:
--------------------------------------------------
source_node
destination_node
num_of_source_followers
num_of_source_followees
num_of_destination_followers
num_of_destination_followees
num_of_common_followers
num_of_common_followees
jaccard_similarity_followers
jaccard_similarity_followees
cosine_similarity_followers
is_followed_back
--------------------------------------------------
Total created features are : 10
```

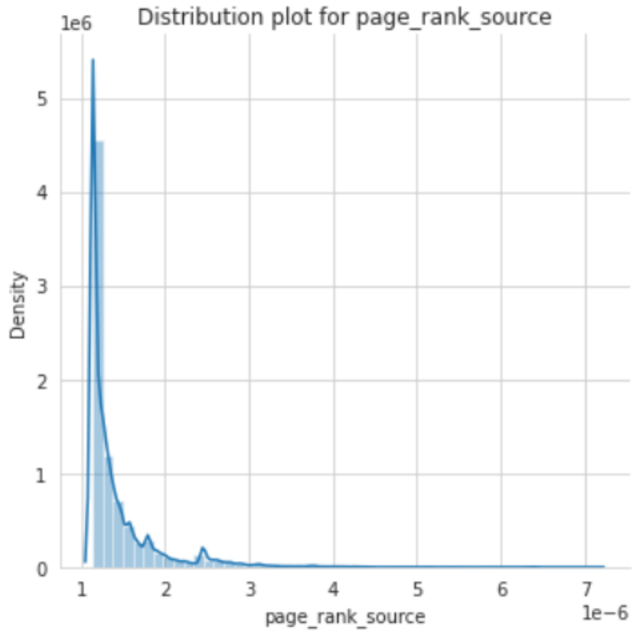
13.Shortest Path Length:



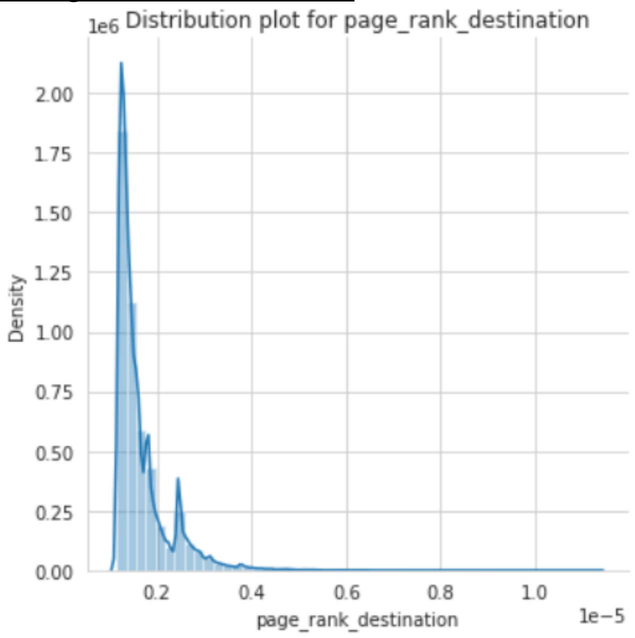
14.Adar Index:



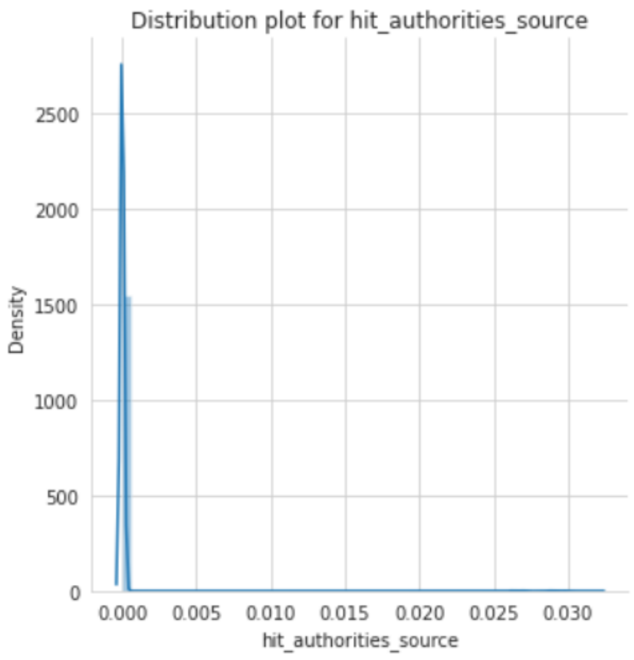
15.Page Rank Source:



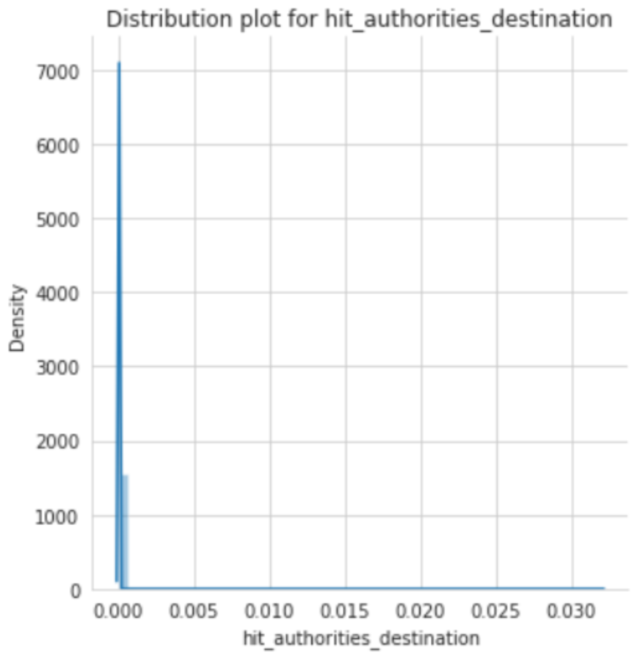
16.Page Rank Destination:



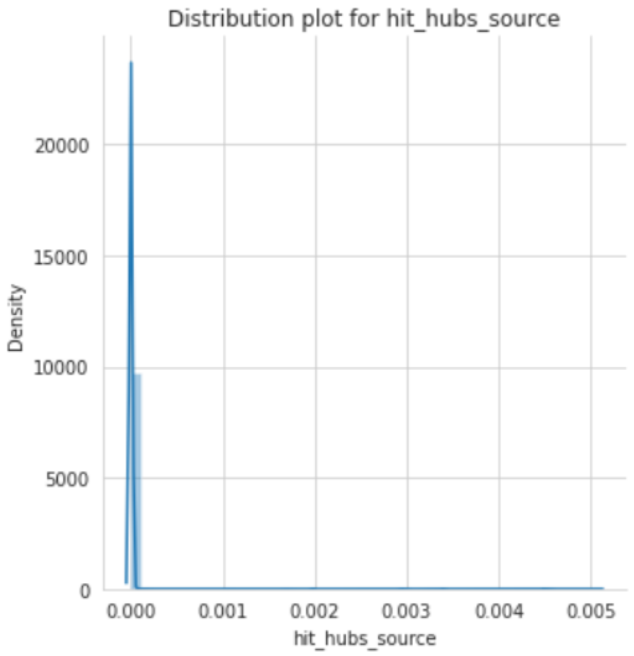
17. Hit Authorities Source:



18.Hit Authorities Destination:

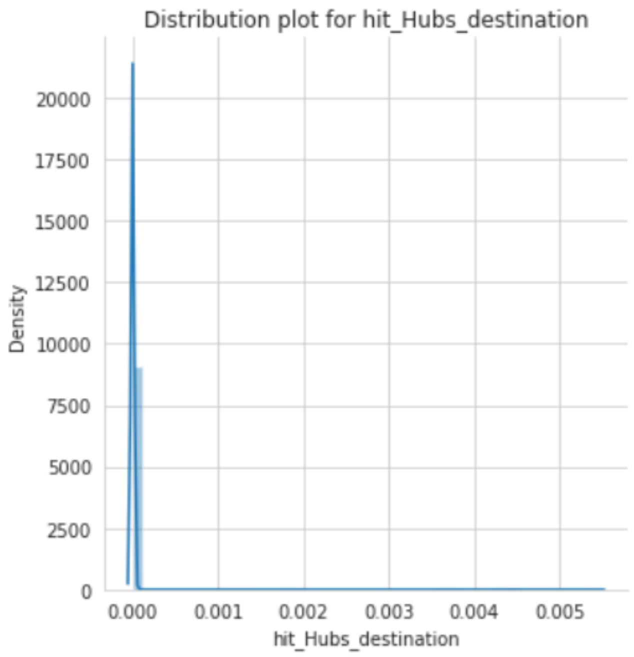


19.Hit Hubs Source:

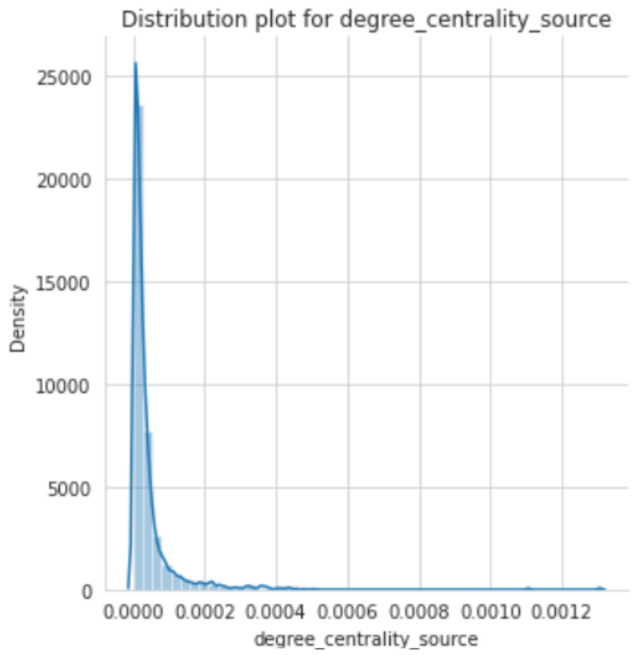


-

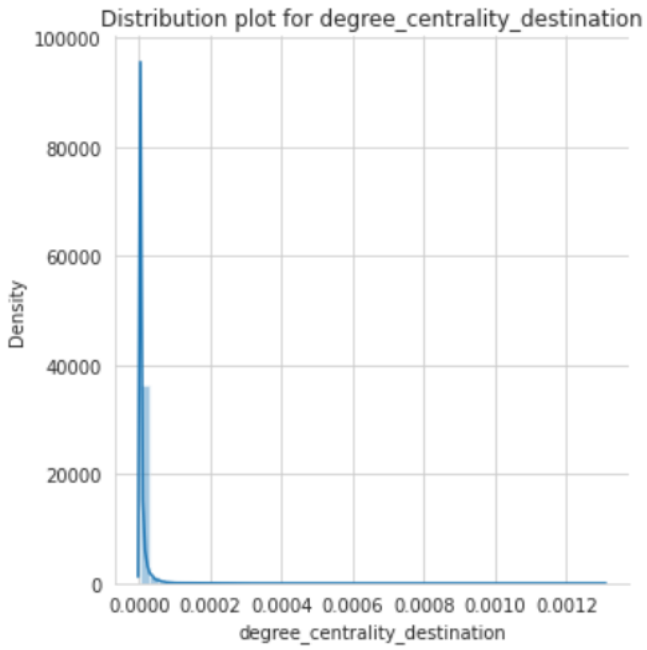
20.Hit hubs Destination:



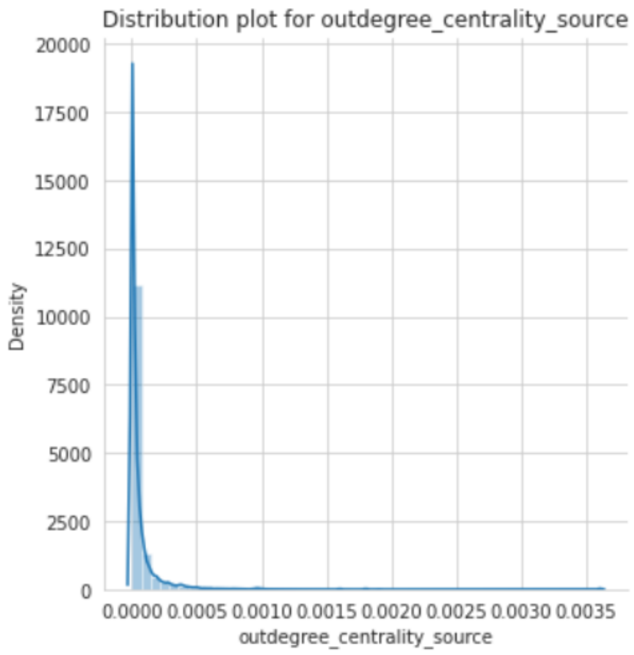
21.Degree centrality source:



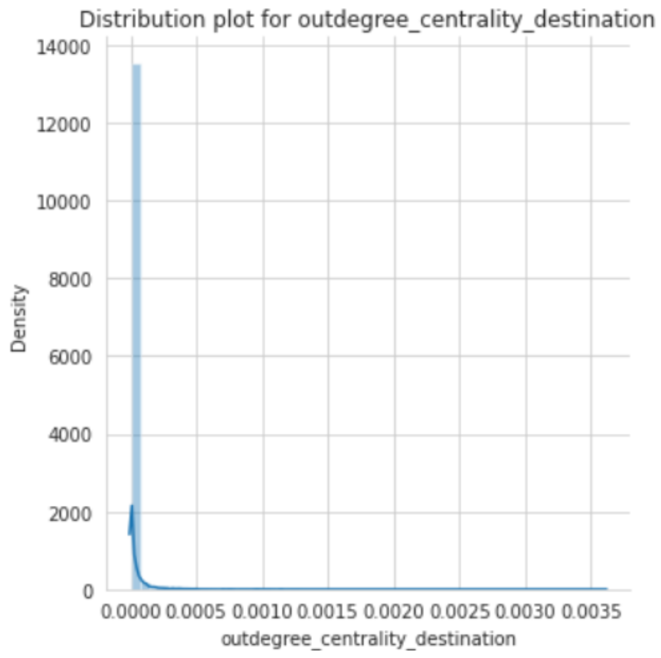
22.Degree Centrality Destination:



23.Outdegree Centrality Score:



24.Outdegree Centrality Destination:



The Final Features are:

The final features are:

1. source_node
2. destination_node
3. num_of_source_followers
4. num_of_source_followees
5. num_of_destination_followers
6. num_of_destination_followees
7. num_of_common_followers
8. num_of_common_followees
9. jaccard_similarity_followers
10. jaccard_similarity_followees
11. cosine_similarity_followers
12. is_followed_back
13. shoterst_path_length
14. page_rank_source
15. page_rank_destination
16. hit_authorities_source
17. hit_authorities_destination
18. hit_hubs_source
19. hit_Hubs_destination
20. degree_centrality_source
21. degree_centrality_destination
22. outdegree_centrality_source
23. outdegree_centrality_destination
24. preferential_attachment_followers

Conclusion : As mentioned earlier, we wanted to explore more about working with graph datasets. The facebook graph dataset helped us to do more research and understand about how featurization is done using graph datasets. In this project , we have extracted 22 features from the given source and destination links. The extracted features can be used in the applications of Instagram friend recommendation system.

Project Management:

Work Completed:

- **Description:** Feature Engineering on Facebook Dataset was basically chosen because usually feature engineering is performed on image and audio files, but here it is a graph dataset which contains two columns source node and destination node.Our first step was to get hold of a dataset which we found it on Kaggle which is an ocean of data. Preparing the data to perform analysis is the next step. Thereafter we processed the extracted text data, then we have perfomed graph features such as Number of followers, Jaccard similarity, Cosine similarities.

- **Responsibility (Task, Person) :**

➢ Gathering dataset and processing the dataset – Prem Sai, Viveksen

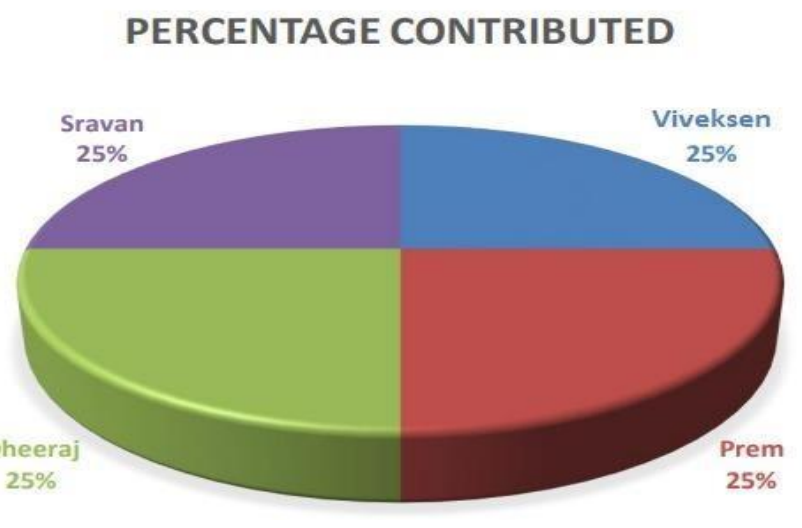
➢ Converting the text data into required form – Viveksen, Dheeraj

➢ Analyzing the data and separating the data – Sravan, Prem Sai

➢ Text Processing and Applying Feature Engineering Techniques – Sravan, Dheeraj

➢ Jaccard Similarity, Cosine Similarity– Dheeraj, Viveksen

➢ Shortest Path Length, Adar Index - Sravan, Prem



- **Issues/Concerns:** N/A

References:

https://towardsdatascience.com/feature-extraction-for-graphs-625f4c5fb8cd

https://www.learndatasci.com/glossary/jaccard-similarity/

https://www.geeksforgeeks.org/find-the-jaccard-index-and-jaccard-distance-between-the-two-given-sets/

https://www.geeksforgeeks.org/cosine-similarity/

https://datascience.stackexchange.com/questions/5121/applications-and-differences-for-jaccard-similarity-and-cosine-similarity

https://betterprogramming.pub/5-ways-to-find-the-shortest-path-in-a-graph-88cfefd0030f