

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.learn datasci.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:

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.

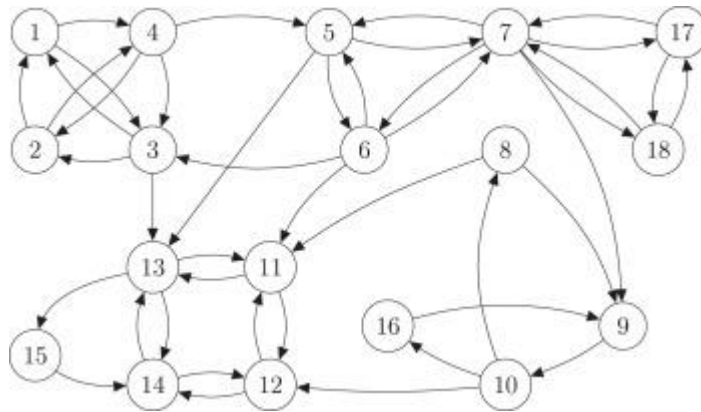
<https://www.kaggle.com/competitions/FacebookRecruiting>

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 for Followers :

In this feature we are calculating the Jaccard similarity for followers between the Source node and Destination node.

8.Jaccard Similarity for Followees :

In this feature we are calculating the Jaccard similarity for followees between the Source node and Destination node.

9.Cosine Similarity for Followers

In this feature we are calculating the Cosine similarity for followers between the Source node and Destination node.

10.Cosine Similarity for Followees :

In this feature we are calculating the Cosine similarity for followees between the Source node and Destination node.

Analysis:

In this section, let's 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. Let's 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 :

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

2.Number of Followees :

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

3.Number of common Followers :

```
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 :

```
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 :

```
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 :

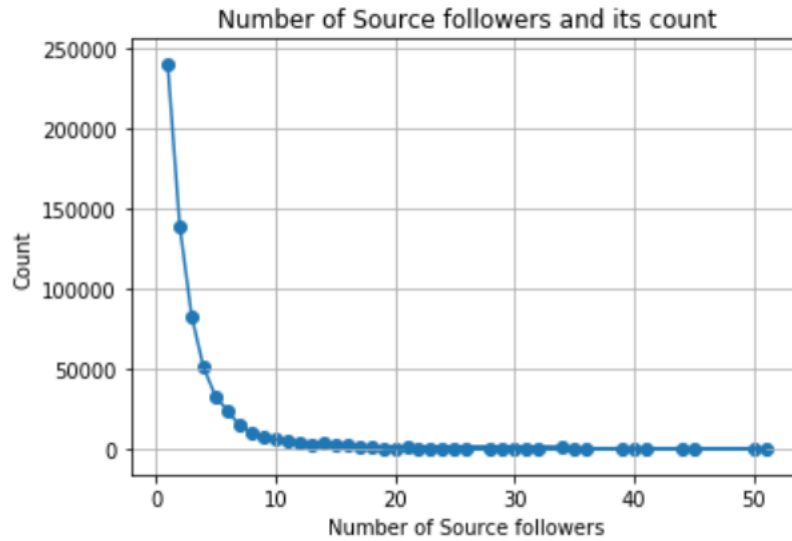
```
def cosine_followees(a,b):
    try:
        X = set(graph.successors(a))
        Y = set(graph.successors(b))
        if len(X) == 0 | len(Y) == 0:
            return 0
        similarity = len(X.intersection(Y))/(math.sqrt(len(X))*
bnmnmbsxzx(len(Y)))
        return similarity
    except:
     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

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.

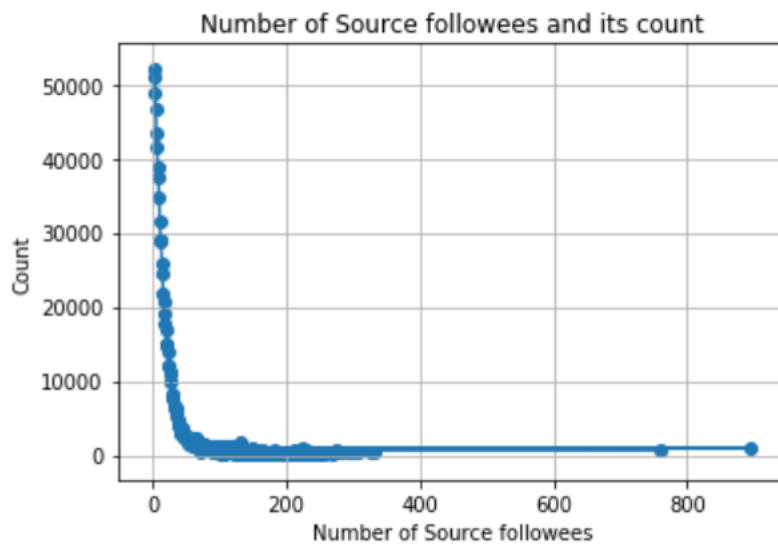
1.Number of Source Followers :

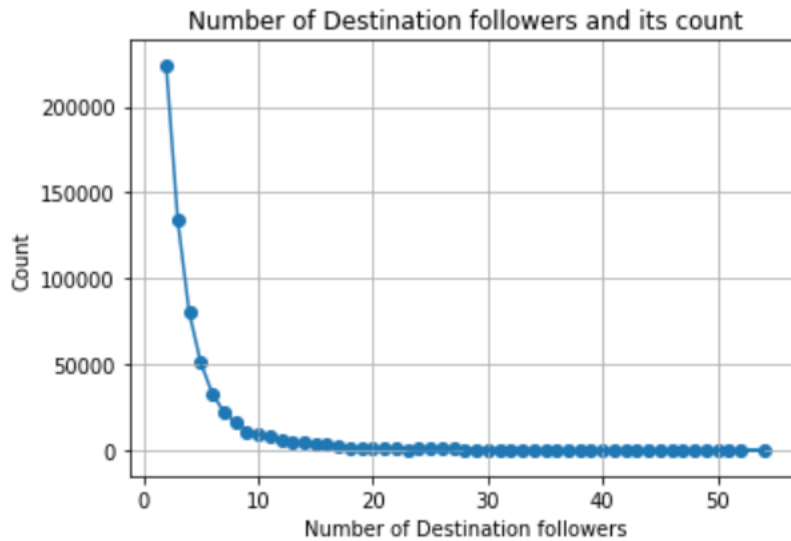
The below screenshots represents number of Source followers and its corresponding count.



2. Number of Source Followees :

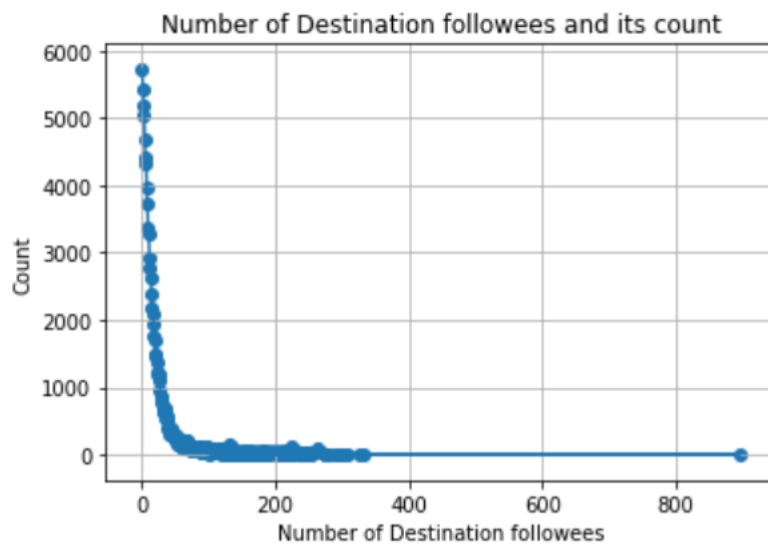
The below screenshots represents number of Source followees and its corresponding count.

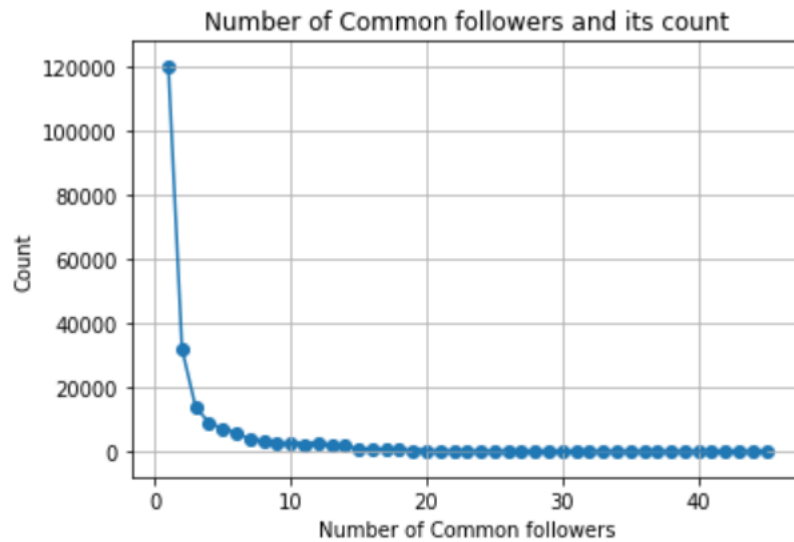




4. Number of Destination node Followees :

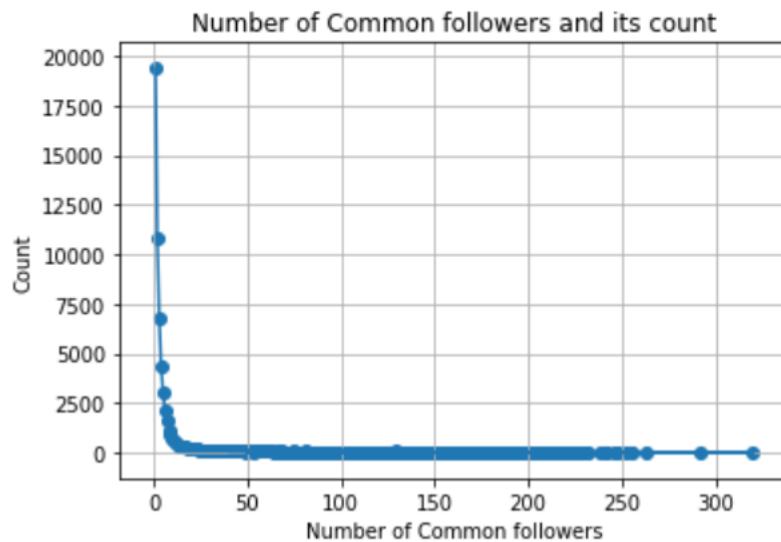
The below screenshot represents the Destination followees and its corresponding count.





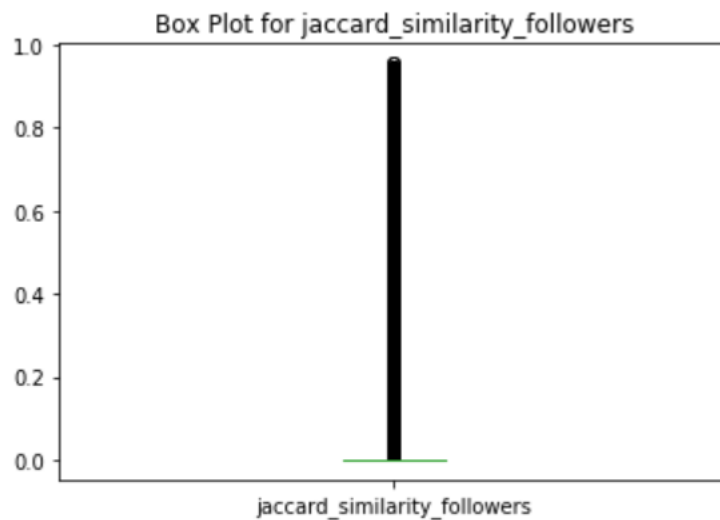
6. Number of common Followees :

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



7. Jaccard Similarity for Followers :

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



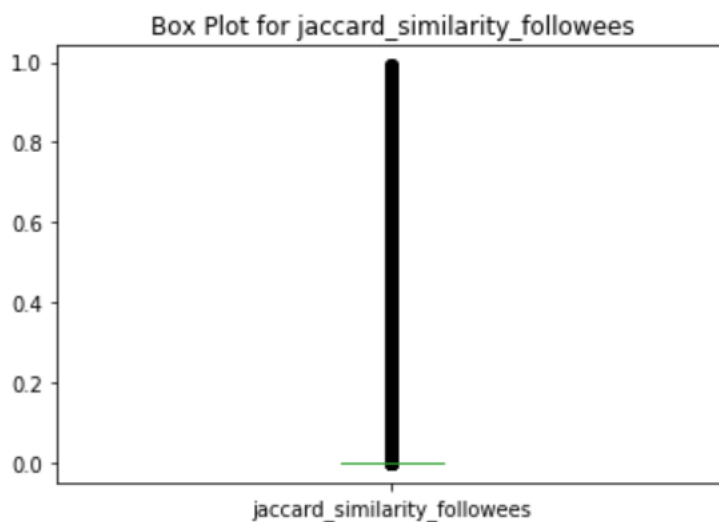
```
train_subset['jaccard_similarity_followers'].describe()
```

```
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.Jaccard Similarity for Followees :



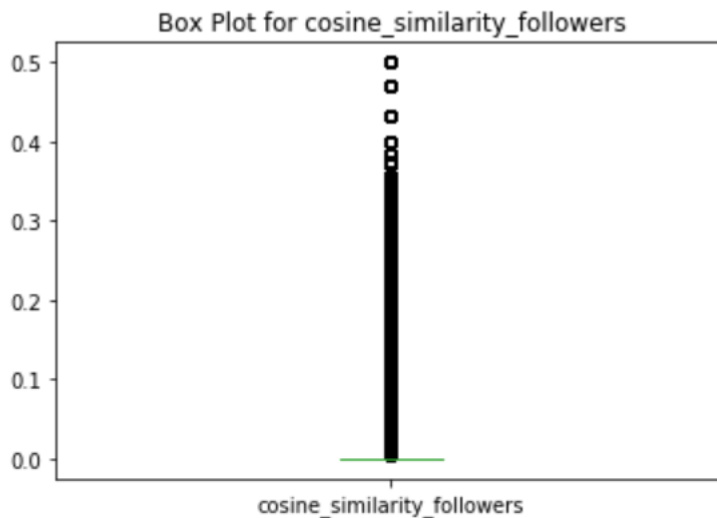
```
train_subset['jaccard_similarity_followees'].describe()
```

```
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:



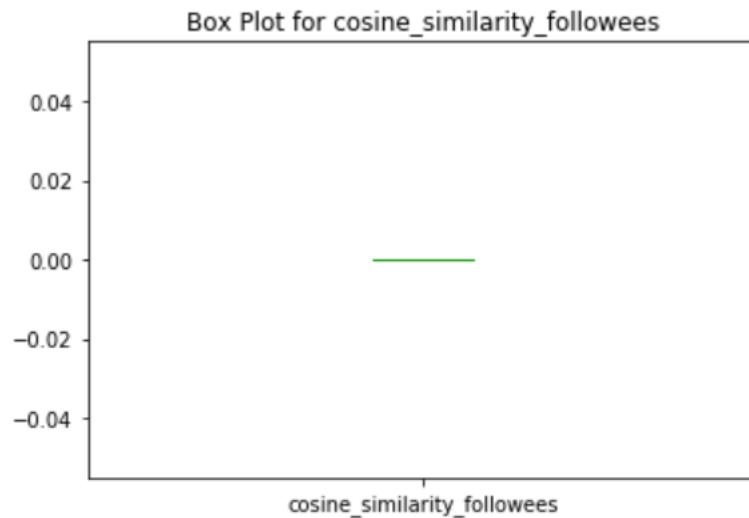
```
train_subset['cosine_similarity_followers'].describe()
```

```
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

Cosine Similarity for Followees :



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

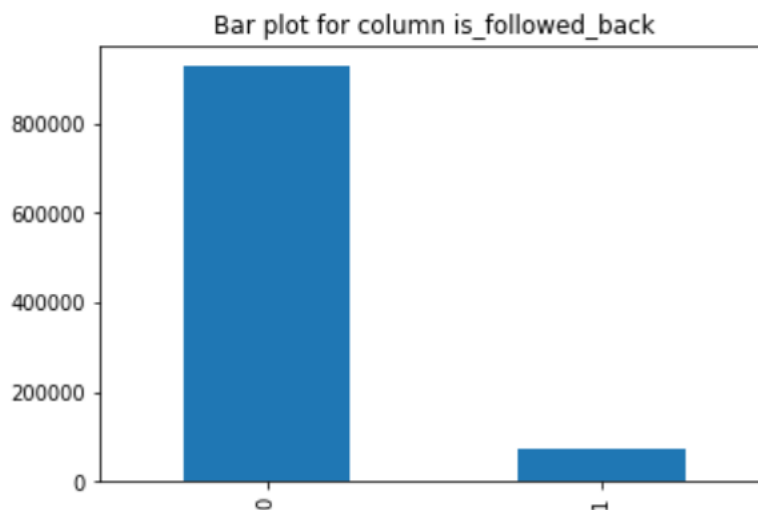
```
count    1000000.0
mean         0.0
std         0.0
min         0.0
25%         0.0
50%         0.0
75%         0.0
max         0.0
Name: cosine_similarity_followees, dtype: float64
```

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
-----
```

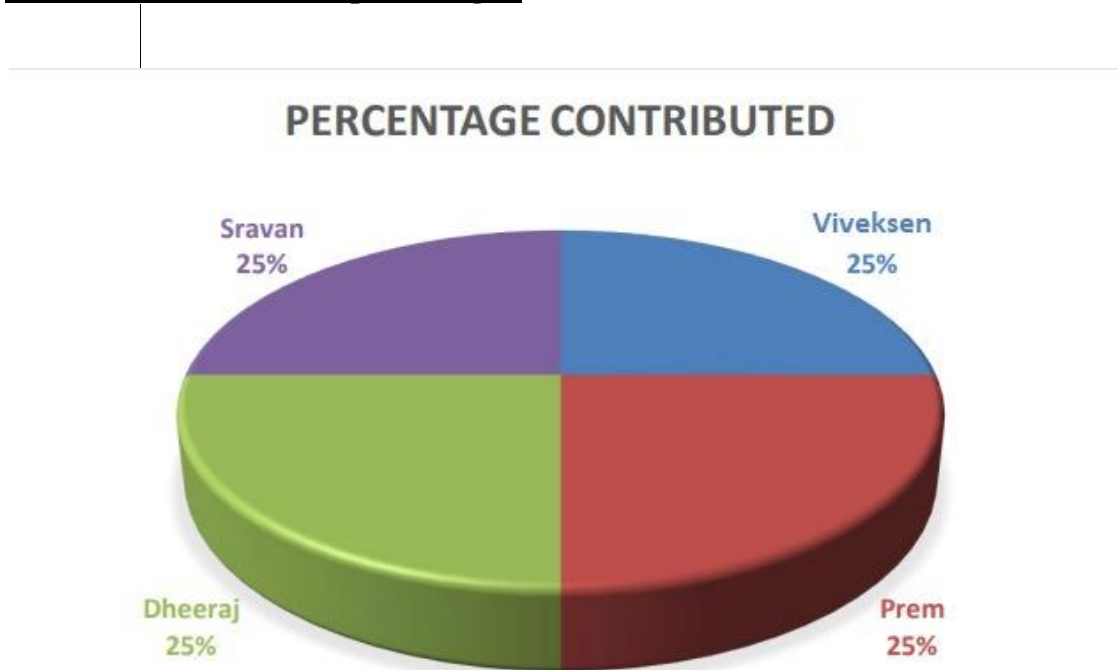
Total created features are : 10

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 performed 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
 - Calculation Similarity score – Dheeraj, Prem

- **Contributions (members/percentage):**



Work to be completed:

- **Description:** In the next increment we will be performing some more graph feature engineering techniques such as 'is following back', 'number of followees', 'page ranking', 'Hits Score', 'Shortest Path', 'Adamic Index' and we will try to add any new features in future if we found any.
- **Responsibility (Task, Person):**
 - Researching new features engineering techniques to apply on this graph data - Vivek, Sravan.
 - Implementation of Page Ranking on our data set - Prem
 - Implementation of Hits Score technique - Dheeraj.
 - Implementation of Shortest Path - Sravan, Prem.
 - Implementation of Adamic Index - Dheeraj, Viveksen
- **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>