Trust Models in Bitcoin Network

Team Members **Guide Level Review**

S.No	Reg.No	Name of the Student	Section
1	CB.EN.U4CSE17065	THUMMALA SAI SRIHITHA REDDY	CSE A
2	CB.EN.U4CSE17220	DEVINENI SATYA VANCITA	CSE C
3	CB.EN.U4CSE17258	SNEHA NANDIGAM	CSE C

Project Advisor: Ms. Vidhya S / Asst. Professor (Sr. Gr) / CSE

Project Summary – Phase-1

- *Cryptocurrency* - digital currency based on principles of cryptography *Bitcoin* - a type of digital currency released in 2009 by Satoshi Nakamoto. One of the oldest cryptocurrencies.
- *Financial crimes* such as money laundering, cyber theft and illicit transactions have been on the rise.
- A *Trust System* will enable any users to identify and avoid risky parties when sending transactions.
- There are little to no *trust models* available in the cryptocurrency field. Our project aims to implement models from various fields and calculate their efficiency.

Project Summary – Phase-1 (cont)

- Analyzed the Bitcoin OTC dataset through visualizations and centrality measures.
- Identified the trend of Betweenness Centrality Measures throughout the time
- Applied trust models to the bitcoin data to classify the nodes as trustworthy and untrustworthy using models such as Goodness-and-Fairness and KNN with the help of centrality measures.
- Found that the ratings aren't an absolute measure of trust score.
- Concluded that fairness, goodness, degree centrality and betweenness centrality are the factors influencing Trust

Proposed work till internship begins

- Implement a classic model whose classification values can be used as true labels to measure the efficiency of the newly implemented models.
- Evaluate if the common Machine Learning Algorithms such as SVM,Random Forest can accurately predict Trustworthiness of nodes
- To measure the efficiency of the implemented trust models using various performance metrics like accuracy, precision, recall and AUC.
- To compare the model efficiencies and derive conclusions about their performance.

Modules of the work

1. Eigentrust algorithm

To implement the Eigentrust algorithm, whose predicted values are used as true labels for other trust models.

2. Implementing Agglomerative clustering

To implement clustering and divide the dataset into 2 clusters - trustworthy and untrustworthy nodes.

3. Implementing SVM
4. Implementing K-Means

Modules of the work (cont)

5. Performance Evaluation of all the models

Using a variety of metrics such as accuracy, precision, recall and AUC to get a grasp on the performance of the models by using Eigentrust values as the true labels.

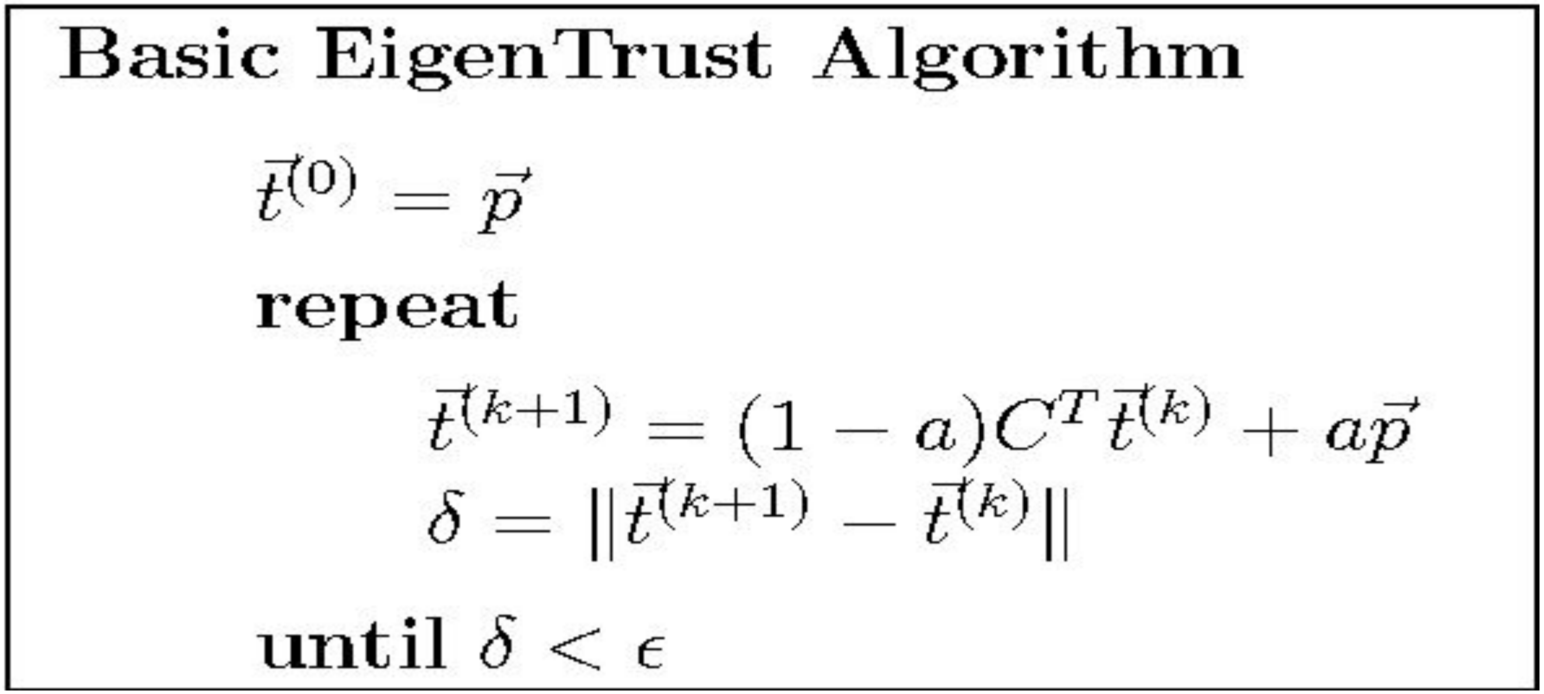
6. Comparing the performance of the implemented models

Comparing the implemented models based on performance and other traits.

Comparing the models and the need for eigentrust

- Need of a common metric and true labels to compare the implemented models
- True labels not available
- Need to implement an accepted model in order to get true labels
- Bitcoin is also a peer-to-peer network
- A classic, accepted trust model in p2p is **eigentrust**.
- After obtaining true labels of the nodes - Metrics such as accuracy, f1 score, auc can be used to compare the implemented models

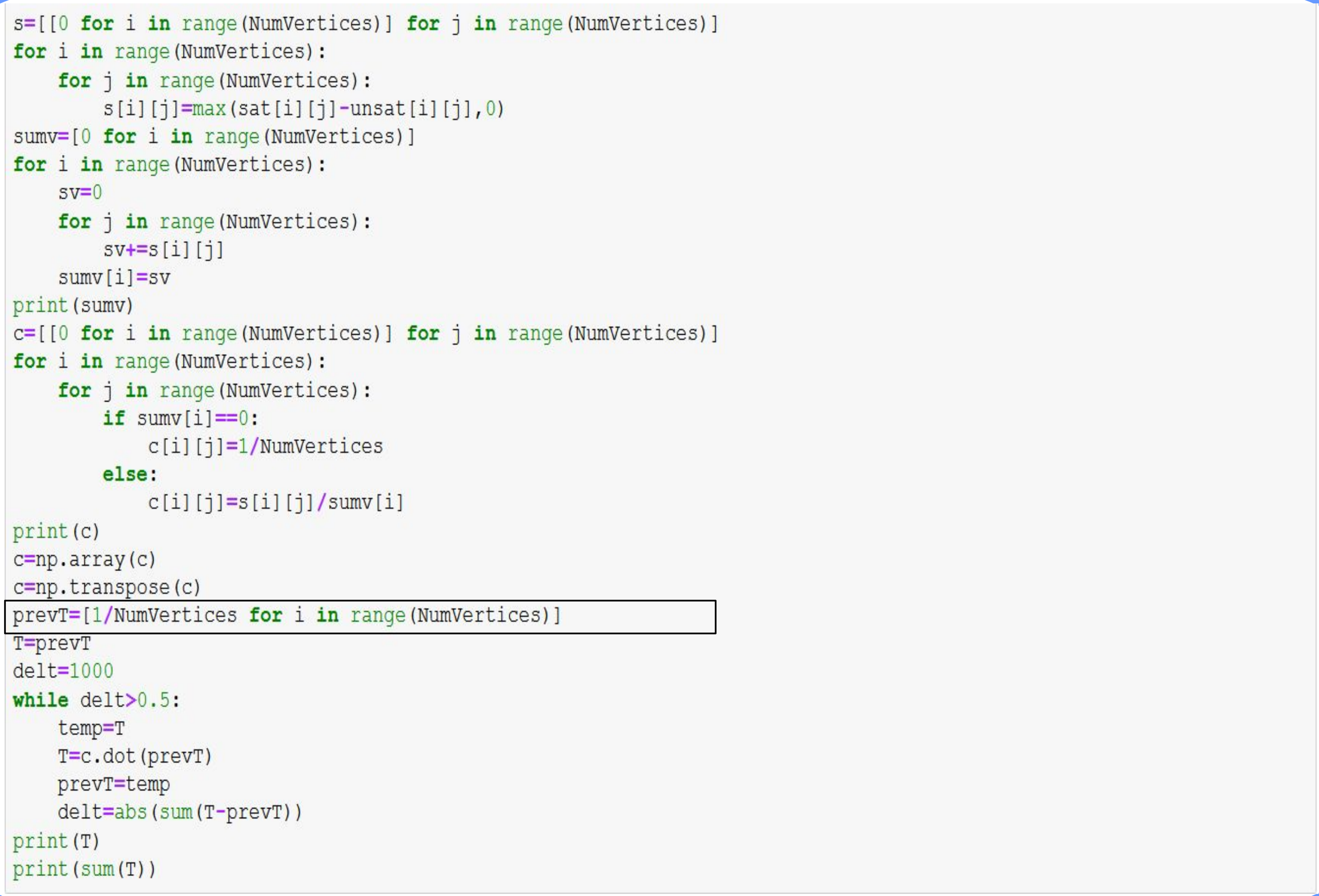
Eigen Trust Pseudo-Code



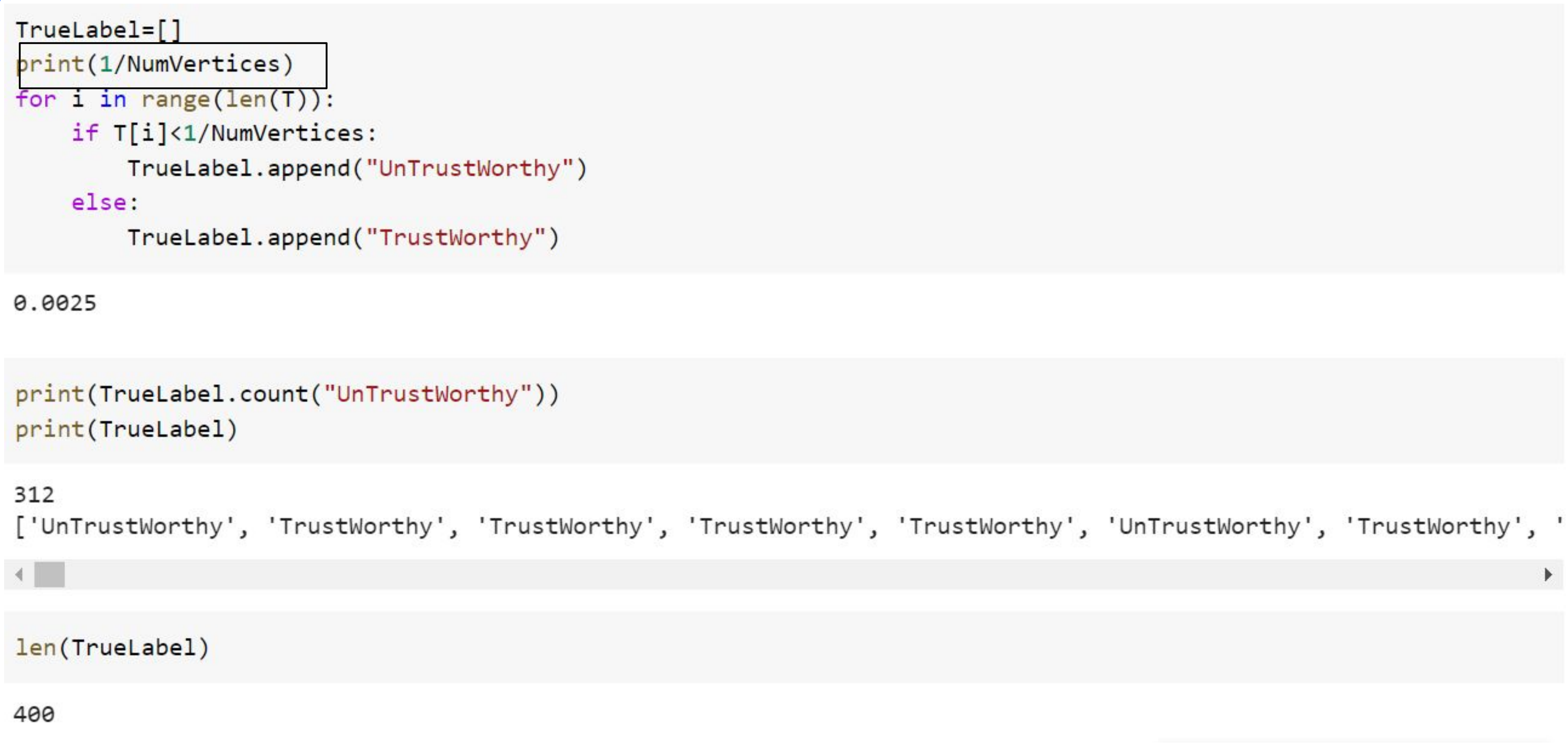
Basic EigenTrust Algorithm

```
$\vec{t}^{(0)} = \vec{p}$
repeat
    $\vec{t}^{(k+1)} = (1-a)C^T\vec{t}^{(k)} + a\vec{p}$
    $\delta = ||\vec{t}^{(k+1)} - \vec{t}^{(k)}||$
until $\delta < \epsilon$
```

Eigen Trust Model Implementation



```
s=[[0 for i in range(NumVertices)] for j in range(NumVertices)]
for i in range(NumVertices):
    for j in range(NumVertices):
        s[i][j]=max(sat[i][j]-unsat[i][j],0)
sumv=[0 for i in range(NumVertices)]
for i in range(NumVertices):
    sv=0
    for j in range(NumVertices):
        sv+=s[i][j]
    sumv[i]=sv
print(sumv)
c=[[0 for i in range(NumVertices)] for j in range(NumVertices)]
for i in range(NumVertices):
    for j in range(NumVertices):
        if sumv[i]==0:
            c[i][j]=1/NumVertices
        else:
            c[i][j]=s[i][j]/sumv[i]
print(c)
c=np.array(c)
c=np.transpose(c)
prevT=[1/NumVertices for i in range(NumVertices)]
T=prevT
delt=1000
while delt>0.5:
    temp=T
    T=c.dot(prevT)
    prevT=temp
    delt=abs(sum(T-prevT))
print(T)
print(sum(T))
```



```
TrueLabel=[]
print(1/NumVertices)
for i in range(len(T)):
    if T[i]<1/NumVertices:
        TrueLabel.append("UnTrustWorthy")
    else:
        TrueLabel.append("TrustWorthy")
```

```
0.0025
```

```
print(TrueLabel.count("UnTrustWorthy"))
print(TrueLabel)
```

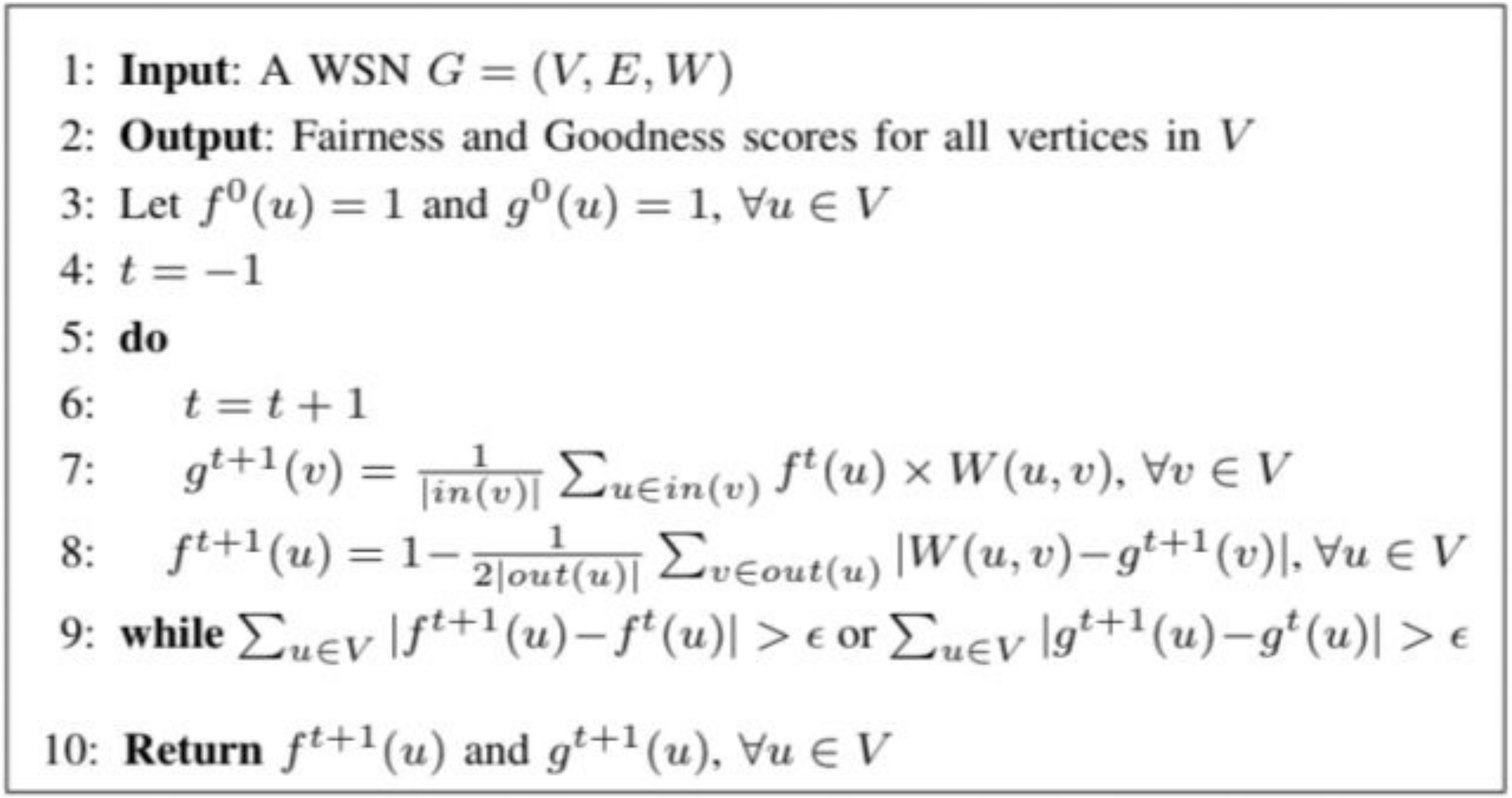
```
312
['UnTrustWorthy', 'TrustWorthy', 'TrustWorthy', 'TrustWorthy', 'TrustWorthy', 'UnTrustWorthy', 'TrustWorthy', '
```

```
len(TrueLabel)
```

```
400
```

Every node gets 1/NumVertices probability of it being an untrustworthy node. By logic, if the end probability of the node is more than 1/NumVertices, it has more probability of being untrustworthy and vice versa.

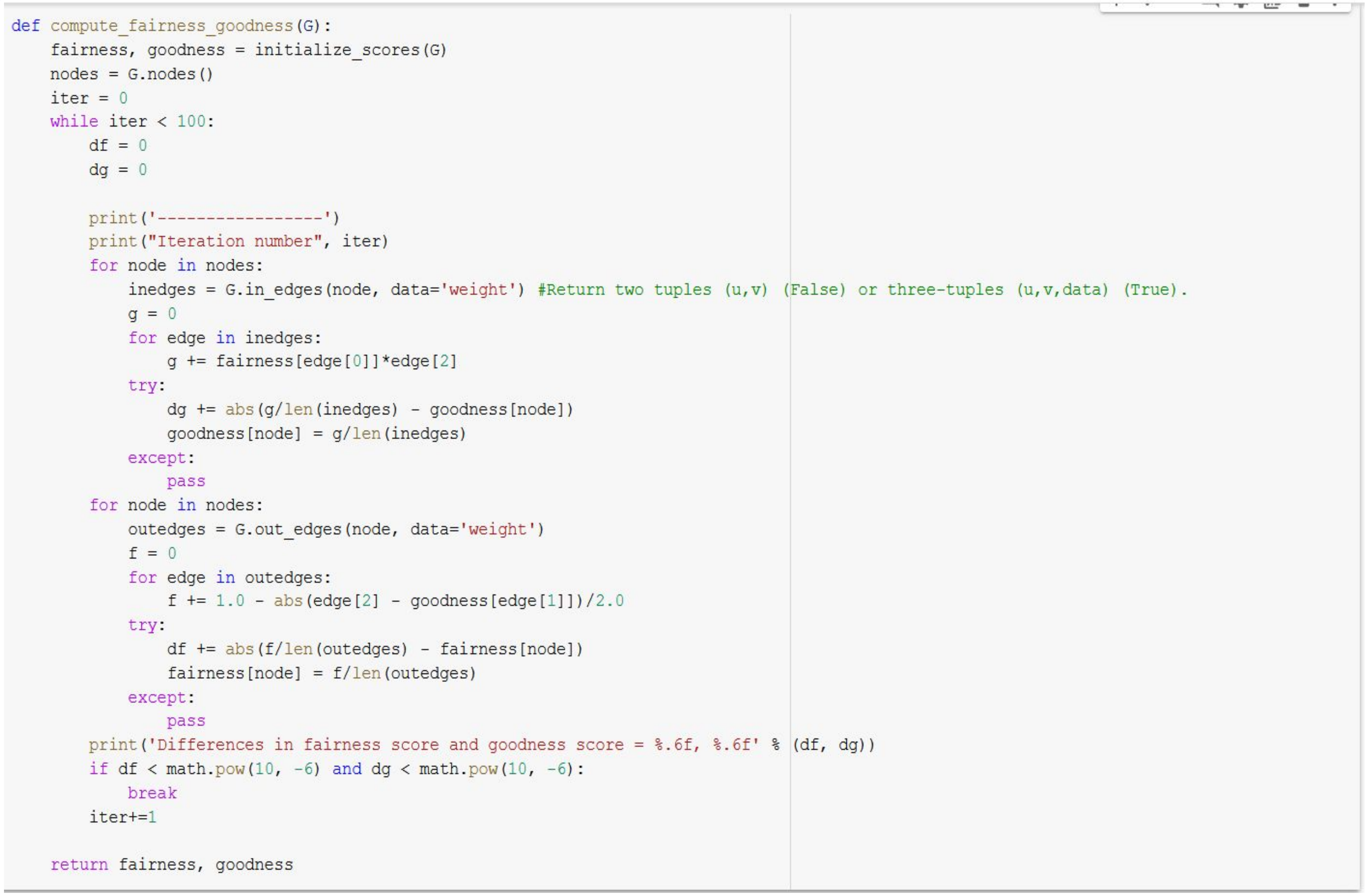
Goodness-Fairness model Pseudo-Code



```
1: Input: A WSN $G = (V, E, W)$
2: Output: Fairness and Goodness scores for all vertices in $V$
3: Let $f^0(u) = 1$ and $g^0(u) = 1$, $\forall u \in V$
4: $t = -1$
5: do
6:     $t = t + 1$
7:     $g^{t+1}(v) = \frac{1}{|in(v)|} \sum_{u \in in(v)} f^t(u) \times W(u, v), \forall v \in V$
8:     $f^{t+1}(u) = 1 - \frac{1}{2|out(u)|} \sum_{v \in out(u)} |W(u, v) - g^{t+1}(v)|, \forall u \in V$
9: while $\sum_{u \in V} |f^{t+1}(u) - f^t(u)| > \epsilon$ or $\sum_{u \in V} |g^{t+1}(u) - g^t(u)| > \epsilon$

10: Return $f^{t+1}(u)$ and $g^{t+1}(u)$, $\forall u \in V$
```

Goodness-Fairness Model implementation

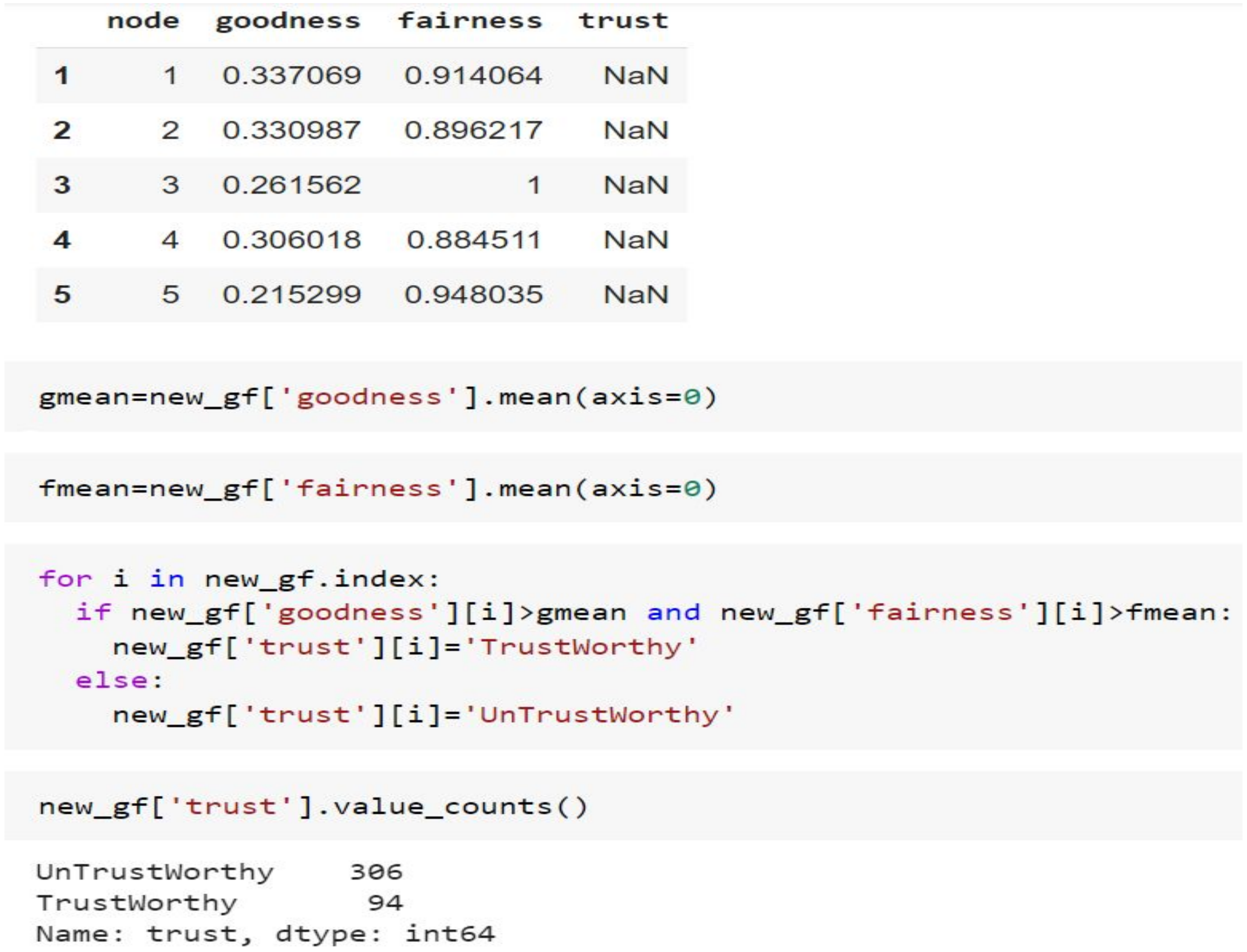


```
def compute_fairness_goodness(G):
    fairness, goodness = initialize_scores(G)
    nodes = G.nodes()
    iter = 0
    while iter < 100:
        df = 0
        dg = 0

        print('-----------------')
        print("Iteration number", iter)
        for node in nodes:
            inedges = G.in_edges(node, data='weight') #Return two tuples (u,v) (False) or three-tuples (u,v,data) (True).
            g = 0
            for edge in inedges:
                g += fairness[edge[0]]*edge[2]
            try:
                dg += abs(g/len(inedges) - goodness[node])
                goodness[node] = g/len(inedges)
            except:
                pass
        for node in nodes:
            outedges = G.out_edges(node, data='weight')
            f = 0
            for edge in outedges:
                f += 1.0 - abs(edge[2] - goodness[edge[1]])/2.0
            try:
                df += abs(f/len(outedges) - fairness[node])
                fairness[node] = f/len(outedges)
            except:
                pass
        print('Differences in fairness score and goodness score = %.6f, %.6f' % (df, dg))
        if df < math.pow(10, -6) and dg < math.pow(10, -6):
            break
        iter+=1

    return fairness, goodness
```

Goodness -Fairness Model Node classification



	node	goodness	fairness	trust
1	1	0.337069	0.914064	NaN
2	2	0.330987	0.896217	NaN
3	3	0.261562	1	NaN
4	4	0.306018	0.884511	NaN
5	5	0.215299	0.948035	NaN

```python
gmean=new_gf['goodness'].mean(axis=0)
```

```python
fmean=new_gf['fairness'].mean(axis=0)
```

```python
for i in new_gf.index:
  if new_gf['goodness'][i]>gmean and new_gf['fairness'][i]>fmean:
    new_gf['trust'][i]='TrustWorthy'
  else:
    new_gf['trust'][i]='UnTrustWorthy'
```

```python
new_gf['trust'].value_counts()
```

```
UnTrustWorthy    306
TrustWorthy       94
Name: trust, dtype: int64
```

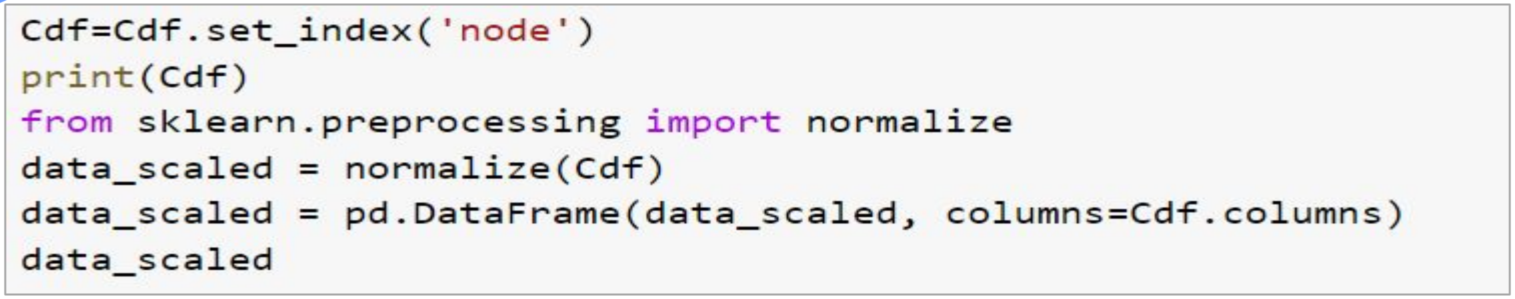
Implementing algorithms based on Centrality Measures

Getting centrality measures for nodes

```
eig_centrality=nx.eigenvector_centrality(G)
bet_centrality = nx.betweenness_centrality(G, normalized = True,endpoints = False)
deg_centrality=nx.degree_centrality(G)
pr = nx.pagerank(G, alpha = 0.8)
```

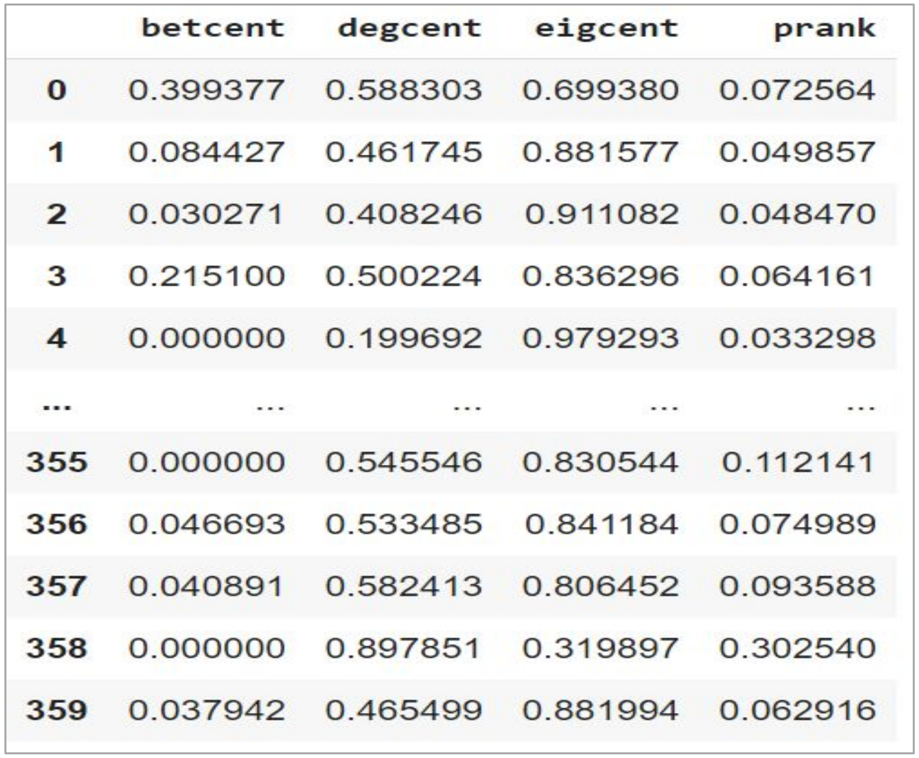
```
        betcent    degcent    eigcent      prank
node
1.0    0.196662   0.289694   0.344391   0.035732
2.0    0.016807   0.091922   0.175500   0.009925
3.0    0.002685   0.036212   0.080814   0.004299
4.0    0.033538   0.077994   0.130394   0.010004
5.0    0.000000   0.008357   0.040981   0.001393
...         ...        ...        ...        ...
395.0  0.000000   0.005571   0.008481   0.001145
396.0  0.001950   0.022284   0.035137   0.003132
397.0  0.000978   0.013928   0.019285   0.002238
398.0  0.000000   0.002786   0.000992   0.000939
399.0  0.001589   0.019499   0.036945   0.002635
```

Agglomerative Clustering



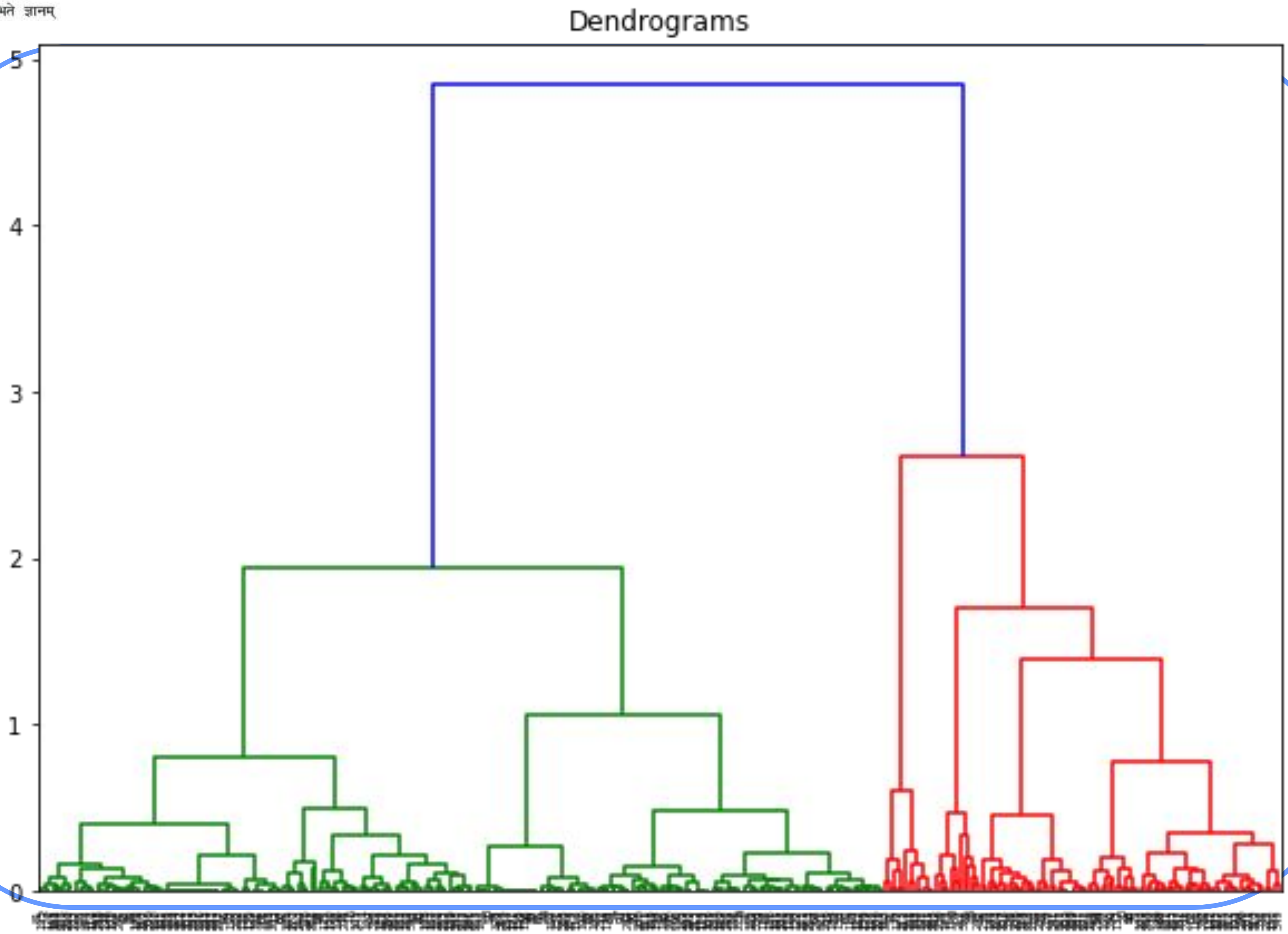
```
Cdf=Cdf.set_index('node')
print(Cdf)
from sklearn.preprocessing import normalize
data_scaled = normalize(Cdf)
data_scaled = pd.DataFrame(data_scaled, columns=Cdf.columns)
data_scaled
```

Matrix with normalized centrality measures for the nodes ->

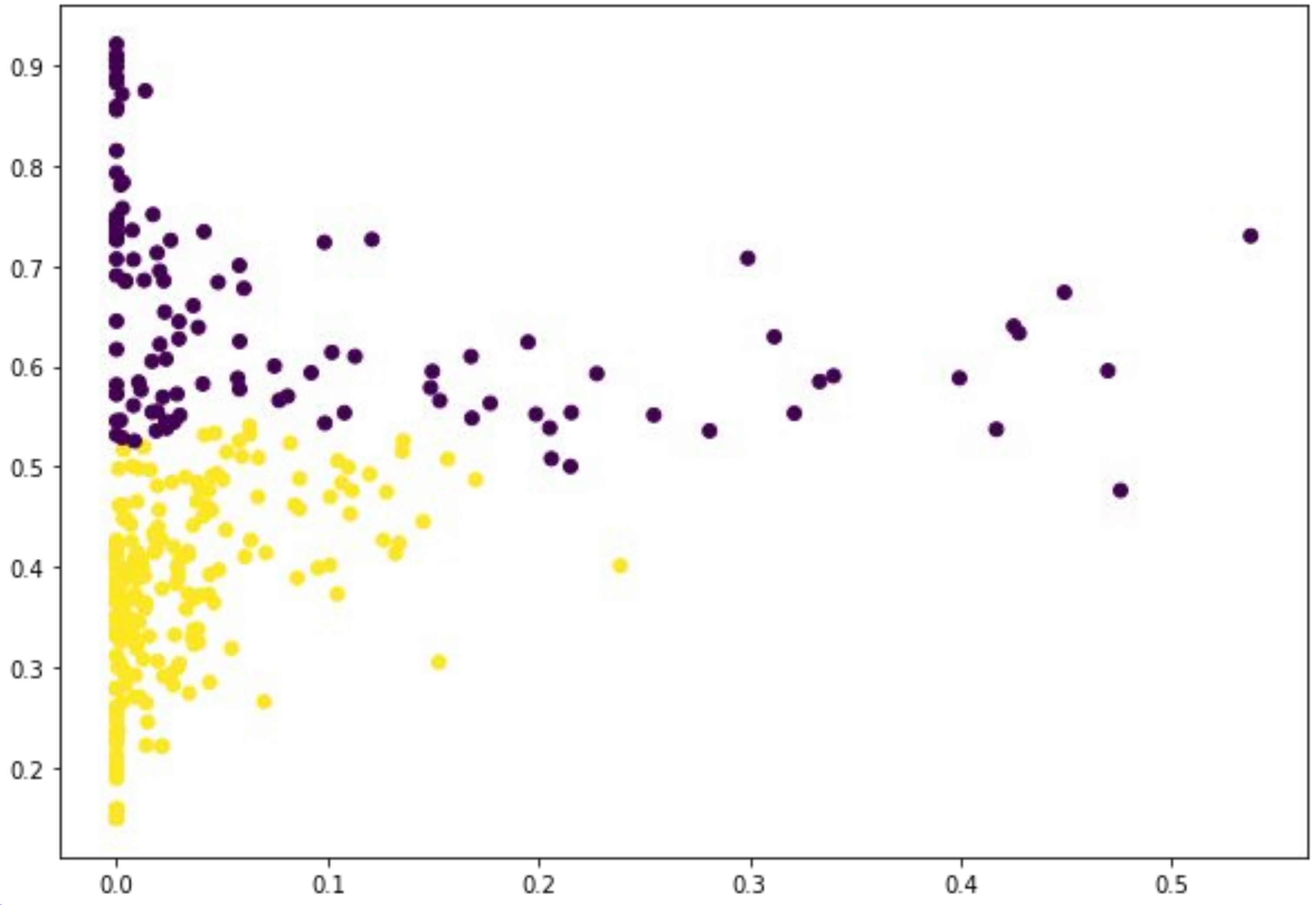


	betcent	degcent	eigcent	prank
0	0.399377	0.588303	0.699380	0.072564
1	0.084427	0.461745	0.881577	0.049857
2	0.030271	0.408246	0.911082	0.048470
3	0.215100	0.500224	0.836296	0.064161
4	0.000000	0.199692	0.979293	0.033298
...	...	...	...	...
355	0.000000	0.545546	0.830544	0.112141
356	0.046693	0.533485	0.841184	0.074989
357	0.040891	0.582413	0.806452	0.093588
358	0.000000	0.897851	0.319897	0.302540
359	0.037942	0.465499	0.881994	0.062916

Hierarchical clustering dendrogram

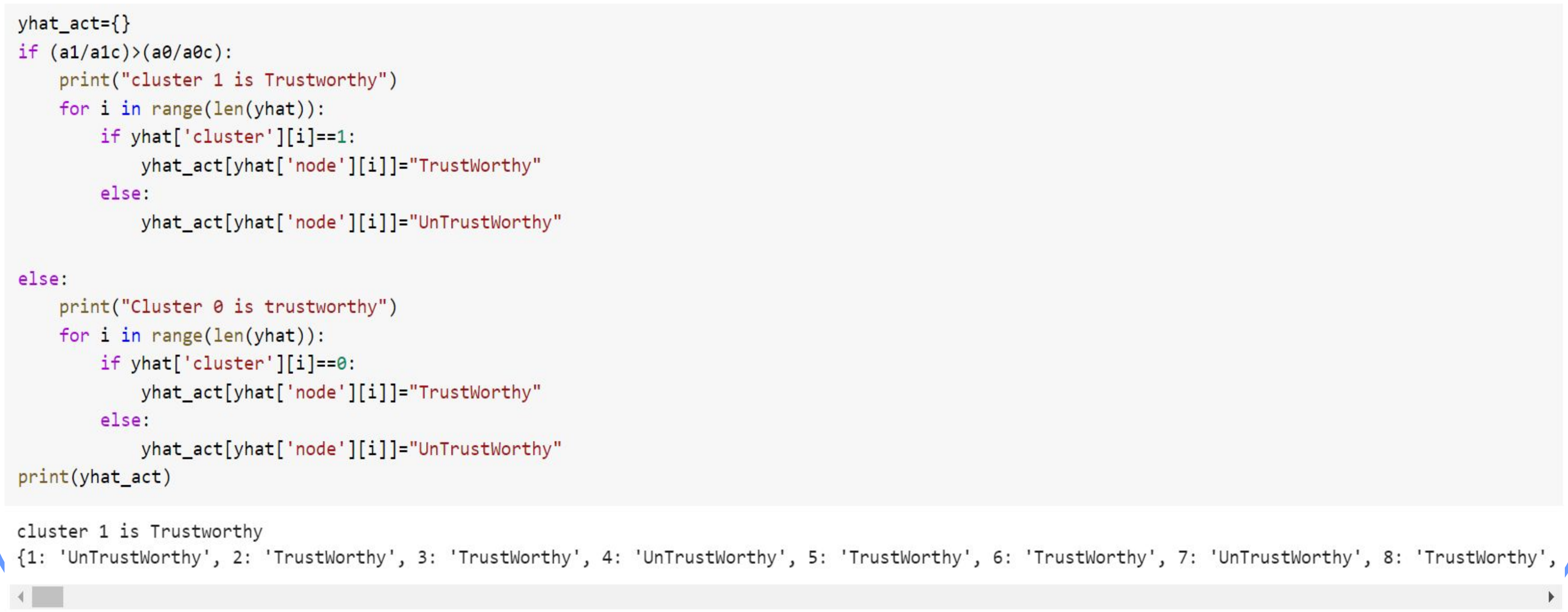


Scatter plot between betcent and degcent colored by the classification labels



Deciding which cluster is trustworthy

- Hierarchical clustering only gives clusters
- labelling the clusters is done based on sum of ratings of the nodes in each cluster.
- the cluster with more total positive rating is decided to be the cluster with trustworthy nodes.



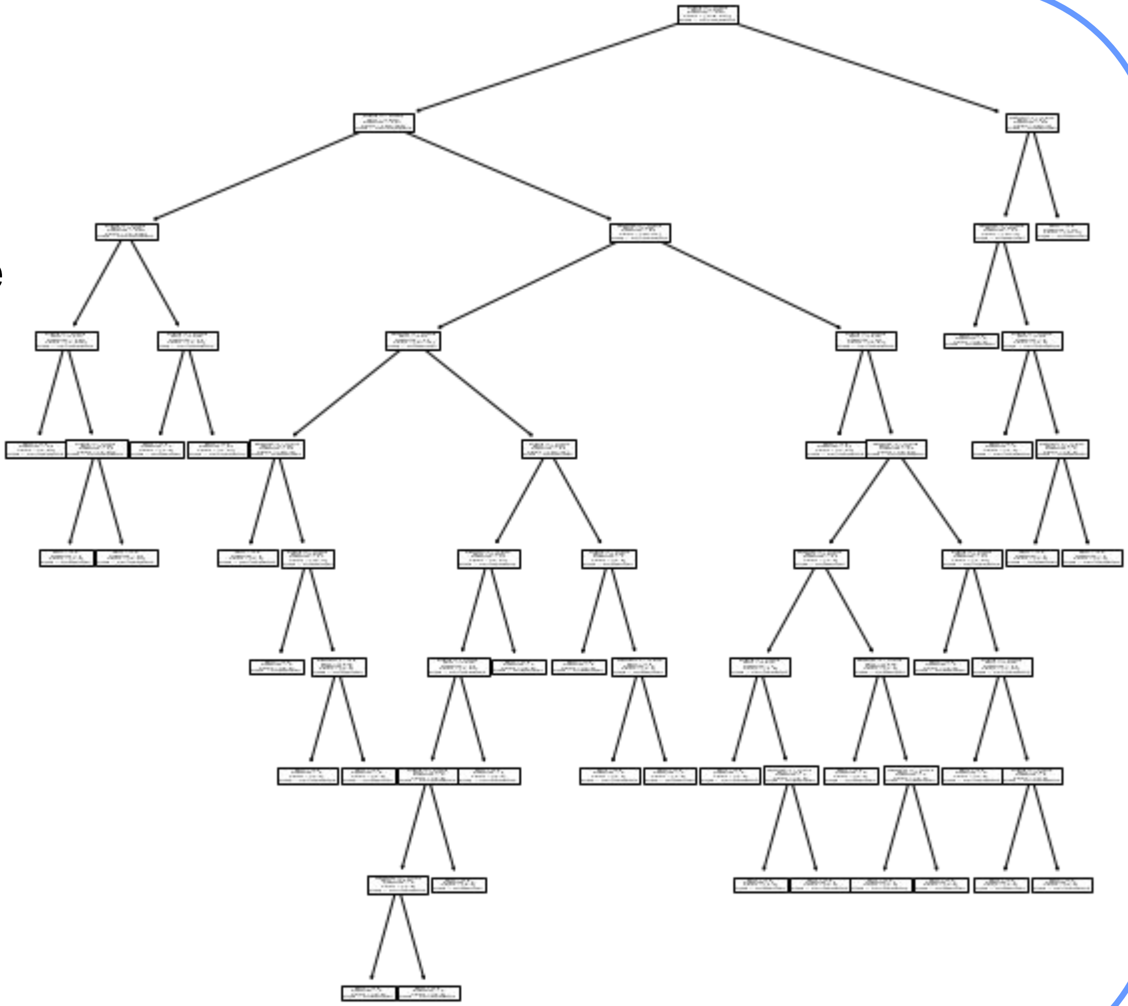
```
yhat_act={}
if (a1/a1c)>(a0/a0c):
    print("cluster 1 is Trustworthy")
    for i in range(len(yhat)):
        if yhat['cluster'][i]==1:
            yhat_act[yhat['node'][i]]="TrustWorthy"
        else:
            yhat_act[yhat['node'][i]]="UnTrustWorthy"

else:
    print("Cluster 0 is trustworthy")
    for i in range(len(yhat)):
        if yhat['cluster'][i]==0:
            yhat_act[yhat['node'][i]]="TrustWorthy"
        else:
            yhat_act[yhat['node'][i]]="UnTrustWorthy"
print(yhat_act)
```

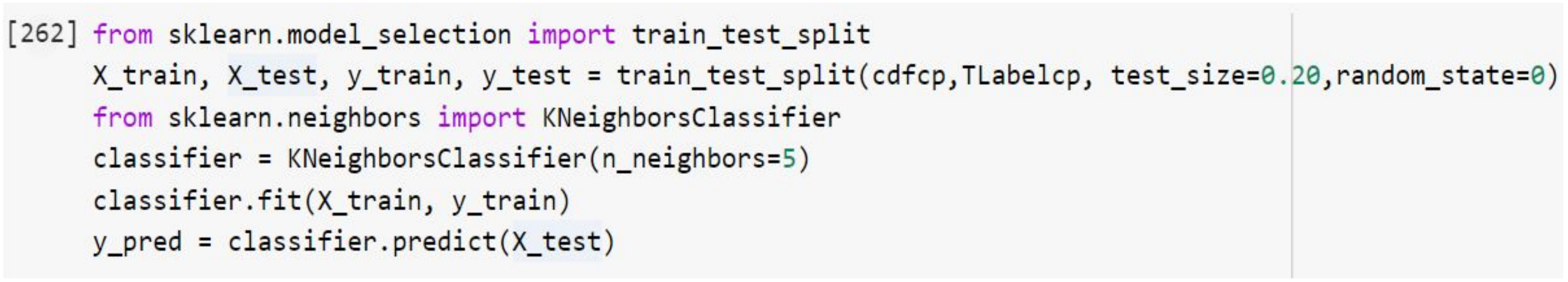
```
cluster 1 is Trustworthy
{1: 'UnTrustWorthy', 2: 'TrustWorthy', 3: 'TrustWorthy', 4: 'UnTrustWorthy', 5: 'TrustWorthy', 6: 'TrustWorthy', 7: 'UnTrustWorthy', 8: 'TrustWorthy',
```

Decision tree classifier

Training the decision tree classifier on 80% of the 400 nodes ->



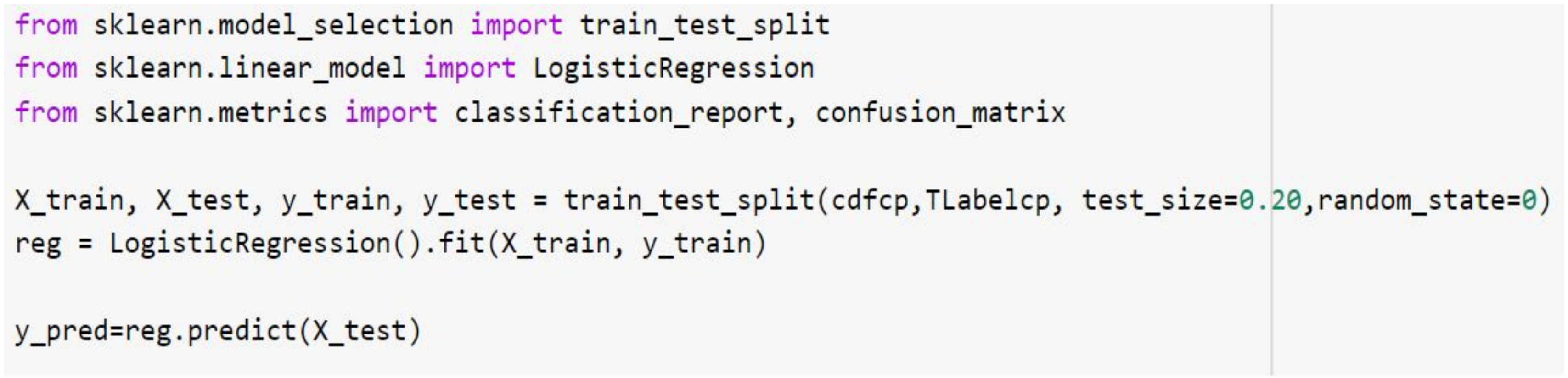
KNN classification



```
[262] from sklearn.model_selection import train_test_split
      X_train, X_test, y_train, y_test = train_test_split(cdfcp,TLabelcp, test_size=0.20,random_state=0)
      from sklearn.neighbors import KNeighborsClassifier
      classifier = KNeighborsClassifier(n_neighbors=5)
      classifier.fit(X_train, y_train)
      y_pred = classifier.predict(X_test)
```

- Similarly classifying based on k neighbours classification
- no of neighbours taken - 5

Logistic Regression



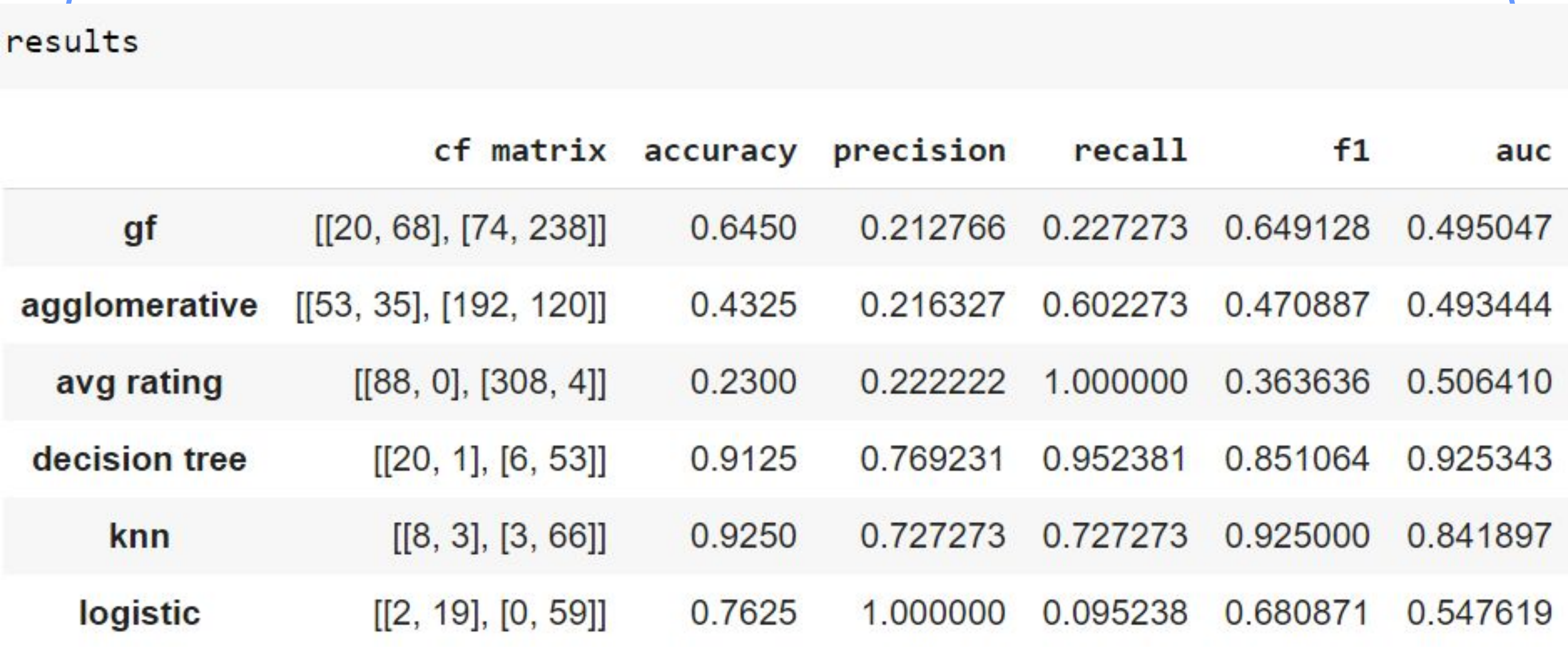
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(cdfcp,TLabelcp, test_size=0.20,random_state=0)
reg = LogisticRegression().fit(X_train, y_train)

y_pred=reg.predict(X_test)
```

- Using Linear Regression to explore whether there’s a linear relationship between the Centrality measures and the True Labels

RESULTS SO FAR

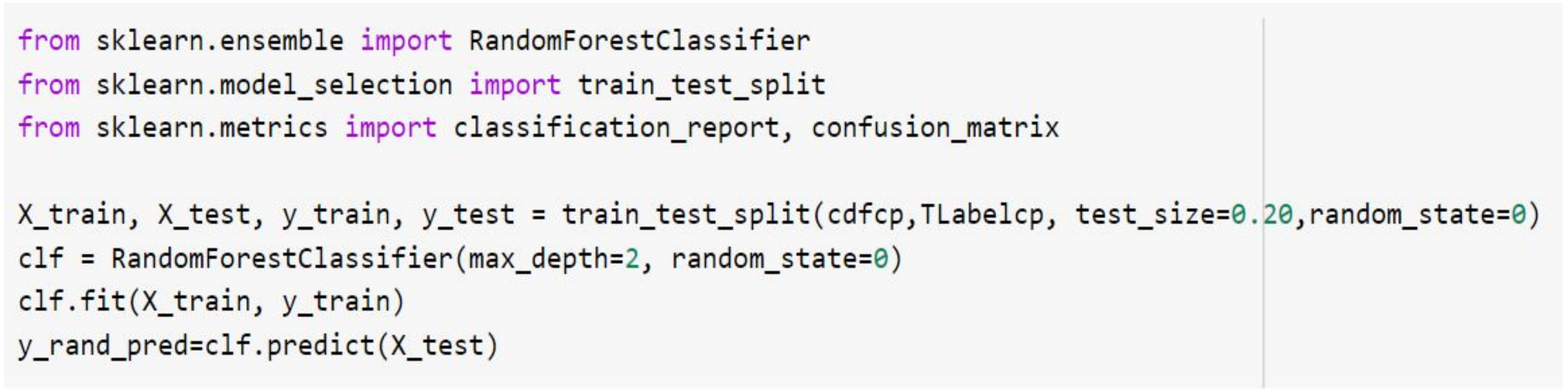


```
results
```

	cf matrix	accuracy	precision	recall	f1	auc
gf	[[20, 68], [74, 238]]	0.6450	0.212766	0.227273	0.649128	0.495047
agglomerative	[[53, 35], [192, 120]]	0.4325	0.216327	0.602273	0.470887	0.493444
avg rating	[[88, 0], [308, 4]]	0.2300	0.222222	1.000000	0.363636	0.506410
decision tree	[[20, 1], [6, 53]]	0.9125	0.769231	0.952381	0.851064	0.925343
knn	[[8, 3], [3, 66]]	0.9250	0.727273	0.727273	0.925000	0.841897
logistic	[[2, 19], [0, 59]]	0.7625	1.000000	0.095238	0.680871	0.547619

ENSEMBLE CLASSIFICATION FOR POTENTIALLY BETTER RESULTS

Random Forest Classifier

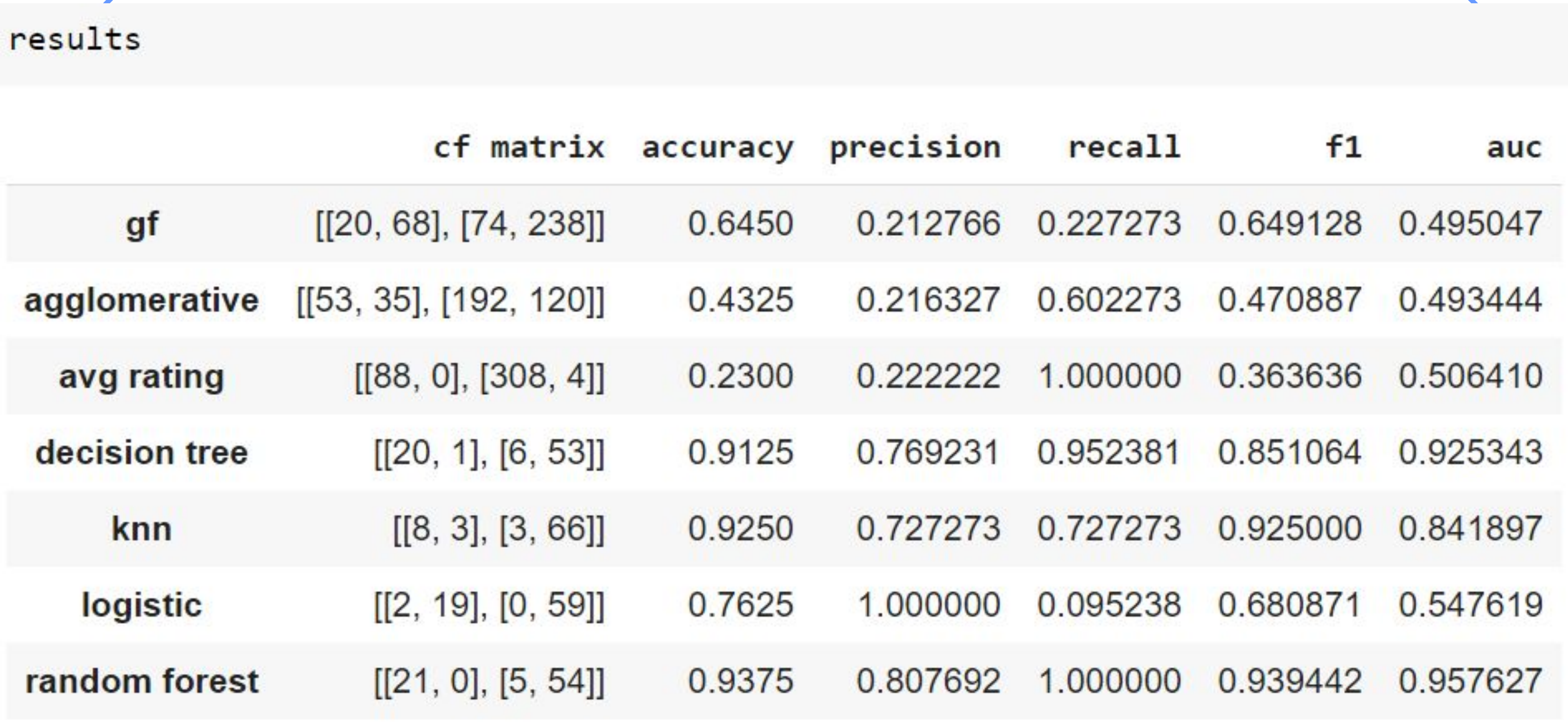


```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(cdfcp,TLabelcp, test_size=0.20,random_state=0)
clf = RandomForestClassifier(max_depth=2, random_state=0)
clf.fit(X_train, y_train)
y_rand_pred=clf.predict(X_test)
```

- Implemented Random Forest Ensemble Classifier to examine whether the results can be improved upon

Results and Analysis

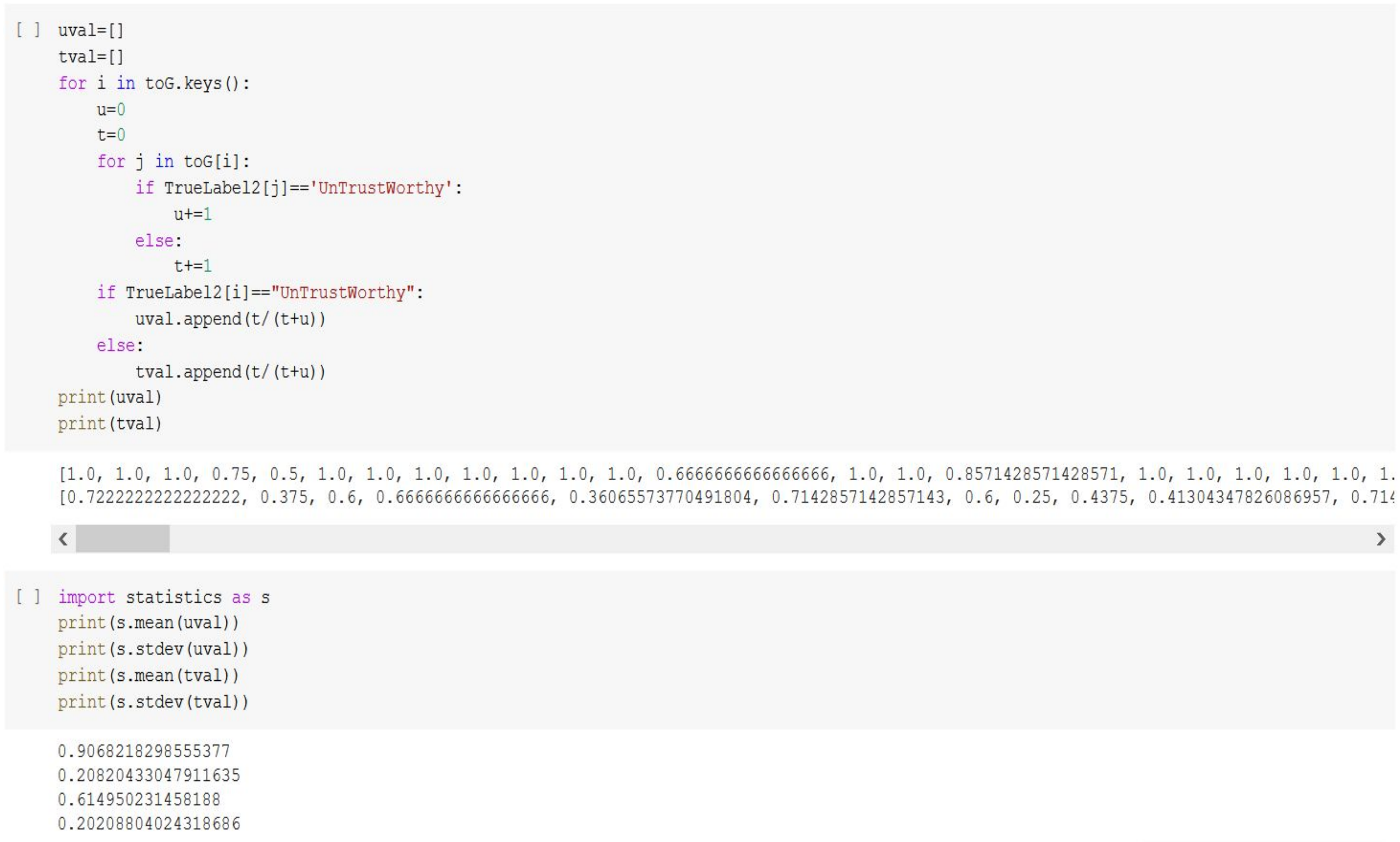


```
results
```

	cf matrix	accuracy	precision	recall	f1	auc
gf	[[20, 68], [74, 238]]	0.6450	0.212766	0.227273	0.649128	0.495047
agglomerative	[[53, 35], [192, 120]]	0.4325	0.216327	0.602273	0.470887	0.493444
avg rating	[[88, 0], [308, 4]]	0.2300	0.222222	1.000000	0.363636	0.506410
decision tree	[[20, 1], [6, 53]]	0.9125	0.769231	0.952381	0.851064	0.925343
knn	[[8, 3], [3, 66]]	0.9250	0.727273	0.727273	0.925000	0.841897
logistic	[[2, 19], [0, 59]]	0.7625	1.000000	0.095238	0.680871	0.547619
random forest	[[21, 0], [5, 54]]	0.9375	0.807692	1.000000	0.939442	0.957627

Performance results of the algorithms based on first 400 nodes of the network

Neighborhood Analysis



```
[ ] uval=[]
    tval=[]
    for i in toG.keys():
        u=0
        t=0
        for j in toG[i]:
            if TrueLabel2[j]=='UnTrustWorthy':
                u+=1
            else:
                t+=1
        if TrueLabel2[i]=="UnTrustWorthy":
            uval.append(t/(t+u))
        else:
            tval.append(t/(t+u))
    print(uval)
    print(tval)
```

```
[1.0, 1.0, 1.0, 0.75, 0.5, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.6666666666666666, 1.0, 1.0, 0.8571428571428571, 1.0, 1.0, 1.0, 1.0, 1.0, 1.
[0.7222222222222222, 0.375, 0.6, 0.6666666666666666, 0.36065573770491804, 0.7142857142857143, 0.6, 0.25, 0.4375, 0.41304347826086957, 0.714
```

```
[ ] import statistics as s
    print(s.mean(uval))
    print(s.stdev(uval))
    print(s.mean(tval))
    print(s.stdev(tval))
```

```
0.9068218298555377
0.20820433047911635
0.614950231458188
0.20208804024318686
```

Observations from Neighborhood Analysis

- From the results, no observed difference in the ratio of number of trustworthy neighbors to the total number of neighbors of a trustworthy node and a untrustworthy node
- No difference seen in the standard deviation as well
- Couldn’t conclude anything about the nature of neighborhood of a trustworthy/untrustworthy node

Observations

- Average Rating is not a reliable way of classifying nodes as Trustworthy or not
- Random Forest Classifier on the centrality measures provides the best performance of all the models compared
- All the models based on Centrality Measures give a decent model performance despite of not considering Average Rating at all
- Neighborhood Analysis of the nodes resulted in NO significant pattern for a trustworthy/untrustworthy neighborhood

Conclusions

- Average Rating ,Neighborhood and the goodness & fairness do not seem to determine True Trust Rating
- From the model performances, we find that Centrality Measures can predict True Trust Rating with a very high accuracy, precision and specificity

Final Conclusion:

- **Centrality Measures are Critical factors affecting Trust**

References

» Shailak, Jani (2018). “The Growth of Cryptocurrency in India: Its Challenges and Potential Impacts on Legislation” Parul University, Vadodara (Thesis). 2. Giudici, Giancarlo, Milne, Alistair, and Vinogradov, Dmitri (2019).

» “Cryptocurrencies: Market Analysis and Perspectives” Journal of Industrial and Business Economics, 47, 1–18. 3. Sadhya, Vikram, Sadhya, Harshali, Hirschheim, Rudy and Watson, Edward (2018).

» “EXPLORING TECHNOLOGY TRUST IN BITCOIN:THE BLOCKCHAIN EXEMPLAR” 26th European Conference on Information Systems. 4. Decker, Christian and Wattenhofer, Roger (2013). “Information propagation in the Bitcoin Network.” 13th IEEE Conference on Peer-to-peer Computing.

» Z. Pi and F. Khan, “An introduction to millimeter-wave mobile broadband systems,” IEEE Commun. Mag., vol. 49, no. 6, pp. 101–107, Jun. 2011.

THANK YOU