# Assignment 1 Complex Systems (Vingron Part)

## Paul Vogler (4979420) and Yiftach Kolb (5195763)

In [148]:
```
import networkx as nx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
```

# Problem 1:

Using the package NetworkX for Python 3, draw the gene network with the following criteria for edges:

- (A) Draw an edge between genes X and Y if the Euclidean distance < 11
- (B) Draw an edge between genes X and Y if the correlation coefficient |r(X, Y )| > 0.75. Color the edges with positive correlation red and the edges with negative correlation blue.
- (C) Draw an edge between genes X and Y if the L1-norm < 7
- (D) Draw an edge between genes X and Y if the mutual information > 0.65. To calculate the mutual information, bin the RPKM values for each gene into 3 intervals.

### 1 Construct a matrix from the data of the exercise sheet:

In [2]:
```
rpkm_matrix = [["A", 7, 9.2, 14.6, 20, 35.1], ["B", 19, 14.2, 6.6, 1
4.6, 18],
                               ["C", 8.6, 7.0, 6.5, 7.3, 8.7], ["D", 6.8,
7.9, 5.5, 2.3, 2.9], ["E", 0.9, 1.8, 3.9, 4.8, 6.2]]
rpkm_matrix
```

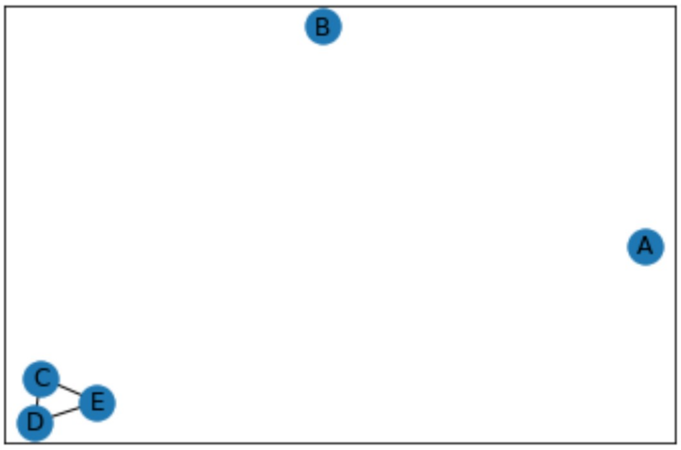
Out[2]:
```
[['A', 7, 9.2, 14.6, 20, 35.1],
 ['B', 19, 14.2, 6.6, 14.6, 18],
 ['C', 8.6, 7.0, 6.5, 7.3, 8.7],
 ['D', 6.8, 7.9, 5.5, 2.3, 2.9],
 ['E', 0.9, 1.8, 3.9, 4.8, 6.2]]
```

### 1 (A) Draw an edge between genes X and Y if the Euclidean distance < 11:

In [3]:
```
def euclidean_distance(x, y):
    return np.sqrt(np.sum((x - y)**2))
```

In [4]:
```python
A = nx.Graph()
for gene in rpkm_matrix:
    A.add_node(gene[0])
print("Euclidean Distances:")
for i in range(len(rpkm_matrix)):
    for j in range(i+1, len(rpkm_matrix)):
        label_1 = rpkm_matrix[i][0]
        label_2 = rpkm_matrix[j][0]
        data_1 = np.array(rpkm_matrix[i][1:])
        data_2 = np.array(rpkm_matrix[j][1:])
        print(label_1, "-", label_2, ":", euclidean_distance(data_1,
data_2))
        if euclidean_distance(data_1, data_2) < 11:
            A.add_edge(label_1, label_2)
nx.draw_networkx(A)
plt.show()
```

```
Euclidean Distances:
A - B : 23.54930996865938
A - C : 30.51655288527851
A - D : 37.877037898969874
A - E : 35.6750613173965
B - C : 17.31444483660969
B - D : 23.854559312634557
B - E : 26.906133129827484
C - D : 7.980601481091509
C - E : 10.275699489572474
D - E : 9.577055915050304
```

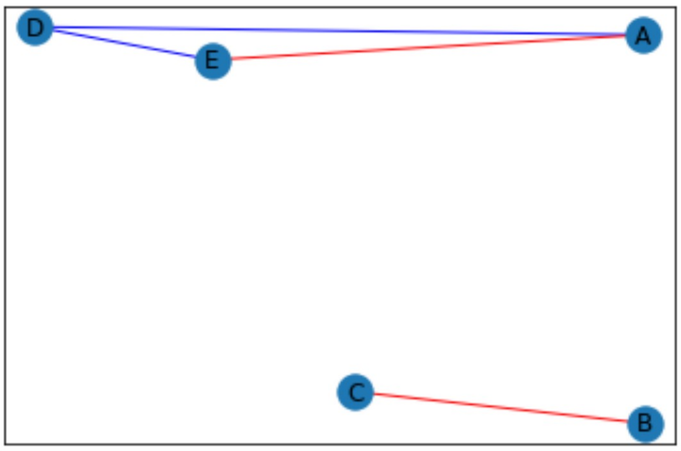


### 1 (B) Draw an edge between genes X and Y if the correlation coefficient |r(X, Y)| > 0.75. Color the edges with positive correlation red and the edges with negative correlation blue.

In [5]:
```python
def sample_correlation(x, y):
    return (np.mean(x*y) - np.mean(x)*np.mean(y)) / (np.sqrt(np.mean
(x**2)-np.mean(x)**2) * np.sqrt(np.mean(y**2)-np.mean(y)**2))
```

In [6]:
```
B = nx.Graph()
for gene in rpkm_matrix:
    B.add_node(gene[0])
print("Correlation coefficients:")
for i in range(len(rpkm_matrix)):
    for j in range(i+1, len(rpkm_matrix)):
        label_1 = rpkm_matrix[i][0]
        label_2 = rpkm_matrix[j][0]
        data_1 = np.array(rpkm_matrix[i][1:])
        data_2 = np.array(rpkm_matrix[j][1:])
        print(label_1, "-", label_2, ":", sample_correlation(data_1,
data_2))
        if sample_correlation(data_1, data_2) > 0.75:
            B.add_edge(label_1, label_2, color='r')
        elif sample_correlation(data_1, data_2) < -0.75:
            B.add_edge(label_1, label_2, color='b')
colors = nx.get_edge_attributes(B,'color').values()
nx.draw_networkx(B, edge_color=colors)
plt.show()
```

```
Correlation coefficients:
A - B : 0.18282164628649436
A - C : 0.36979349454103805
A - D : -0.8082141771478251
A - E : 0.9365791829329874
B - C : 0.8968154506323474
B - D : -0.09154760671741063
B - E : -0.11285372096379764
C - D : -0.20926224800854257
C - E : 0.06545618513053815
D - E : -0.8838215830121594
```

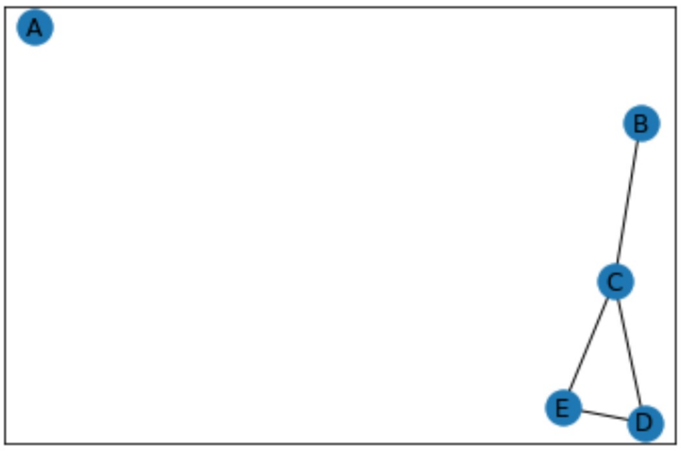


## 1 (C) Draw an edge between genes X and Y if the L1-norm < 7:

In [7]:
```
def l1_norm(x, y):
    return np.sum(np.abs(x-y)) / len(x)
```

In [8]:
```
C = nx.Graph()
for gene in rpkm_matrix:
    C.add_node(gene[0])
print("L1 norm:")
for i in range(len(rpkm_matrix)):
    for j in range(i+1, len(rpkm_matrix)):
        label_1 = rpkm_matrix[i][0]
        label_2 = rpkm_matrix[j][0]
        data_1 = np.array(rpkm_matrix[i][1:])
        data_2 = np.array(rpkm_matrix[j][1:])
        print(label_1, "-", label_2, ":", l1_norm(data_1, data_2))
        if l1_norm(data_1, data_2) < 7:
            C.add_edge(label_1, label_2)
nx.draw_networkx(C)
plt.show()
```

```
L1 norm:
A - B : 9.5
A - C : 10.2
A - D : 12.1
A - E : 13.66
B - C : 6.860000000000001
B - D : 9.4
B - E : 10.959999999999999
C - D : 2.8999999999999995
C - E : 4.1
D - E : 3.8800000000000003
```

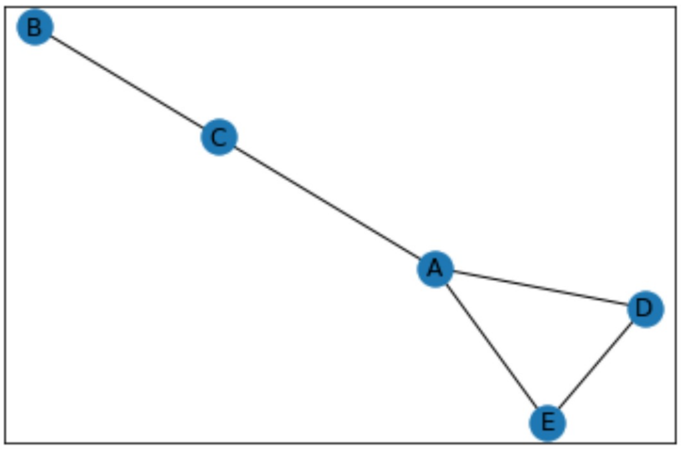


### 1 (D) Draw an edge between genes X and Y if the mutual information > 0.65:

```
In [82]: def mutual_information(x, y):
             hist_x, edges_x = np.histogram(x, bins=3, density=False)
             hist_y, edges_y = np.histogram(y, bins=3, density=False)
             hist_xy, edges_xy1, edges_xy2 = np.histogram2d(x, y, bins=3, den
         sity=False)
             list_x = []
             hist_x, hist_y, hist_xy = hist_x / 5, hist_y / 5, hist_xy / 5
             for i in range(len(hist_x)):
                 list_y = []
                 for j in range(len(hist_y)):
                     if hist_xy[i][j] == 0: #<- to avoid the nans when dividi
         ng by hist_x[i]*hist_y[j] = 0
                         list_y.append(0)
                     else:
                         list_y.append(hist_xy[i][j]*np.log(hist_xy[i][j] /
         (hist_x[i]*hist_y[j])))
                 list_x.append(np.sum(list_y))
             return np.sum(list_x)
```

In [83]:
```
D = nx.Graph()
for gene in rpkm_matrix:
    D.add_node(gene[0])
print("Mutual Information:")
for i in range(len(rpkm_matrix)):
    for j in range(i+1, len(rpkm_matrix)):
        label_1 = rpkm_matrix[i][0]
        label_2 = rpkm_matrix[j][0]
        data_1 = np.array(rpkm_matrix[i][1:])
        data_2 = np.array(rpkm_matrix[j][1:])
        print(label_1, "-", label_2, ":", mutual_information(data_1,
data_2))
        if mutual_information(data_1, data_2) > 0.65:
            D.add_edge(label_1, label_2)
nx.draw_networkx(D)
plt.show()
```

```
Mutual Information:
A - B : 0.39575279478527825
A - C : 0.6730116670092564
A - D : 0.6730116670092564
A - E : 0.6730116670092564
B - C : 0.7776612957621658
B - D : 0.5004024235381878
B - E : 0.5004024235381878
C - D : 0.5004024235381878
C - E : 0.5004024235381878
D - E : 1.054920167986144
```



## Problem 2

Consider two random variables X and Y from which we drew the following samples:

In [15]:
```
samples_x = (0.3, 0.98, 0.54, 0.49, 0.39, 0.14, 0.03, 0.81, 0.65, 0.
18)
samples_y = (0.74, 0.09, 0.48, 0.15, 0.71, 0.8, 0.53, 0.95, 0.63, 0.
88)
```

Therefore the first observation is (x = 0.3, y = 0.74) and so on (10 observations in total). First, bin the data by dividing the interval of [0, 1] into 4 equally wide sub-intervals.

In [40]:
```
hist_xy, edges_xy1, edges_xy2 = np.histogram2d(samples_y, samples_x,
bins=4, range=[[0,1],[0,1]], density=False)
hist_xy
```

Out[40]:
```
array([[0., 1., 0., 1.],
       [0., 0., 1., 0.],
       [1., 2., 1., 0.],
       [2., 0., 0., 1.]])
```

In [84]:
```
edges_xy1
```

Out[84]:
```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

In [85]:
```
# normalize
hist_xy /10
```

Out[85]:
```
array([[0. , 0.1, 0. , 0.1],
       [0. , 0. , 0.1, 0. ],
       [0.1, 0.2, 0.1, 0. ],
       [0.2, 0. , 0. , 0.1]])
```

**The rest of this exercise follows in the PDF of a Word Sheet attached after this Notebook**

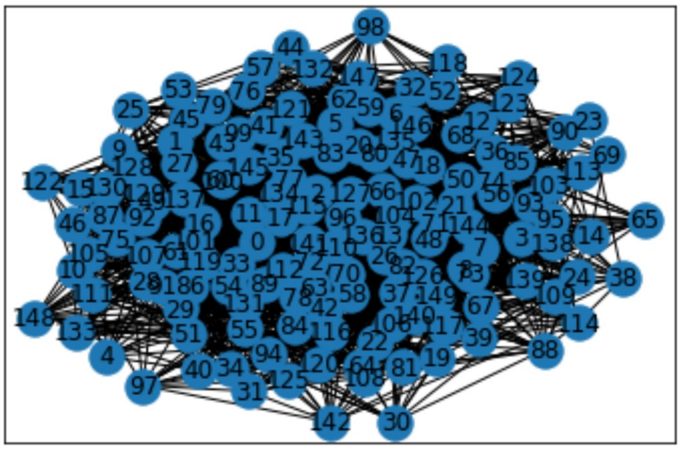
# Problem 3

## (A)

Generate a random network G(n = 150, p = 0.08) according to Erdős-Rényi model where n is the number of nodes and p is the probability of including an edge in the graph (remember to set a random seed, mention it in your report). Analyze your network: calculate the number and size of connected components, plot histograms for the degree values, closeness and betweenness centralities. Normalize the degree histogram and on top of it plot the theoretically appropriate Poisson probability mass function (you may use scipy.stats.poisson). Compare the two. Export the network as GraphML file and the node attributes (centralities, degree) as a csv file. Check what happens when you change the value of p (you do not need to hand in the results for other values of p).

### (A) 1 Generate a random network G(n = 150, p = 0.08) according to Erdős-Rényi model

In [176]:
```python
G = nx.Graph()
n = 150
p = 0.08
np.random.seed(42) #<- here we set the seed for numpy's rng to alway
s get the same result
G.add_nodes_from(range(n))
for i in range(n):
    for j in range(n):
        rand_val = np.random.rand()
        if rand_val < p:
            G.add_edge(i, j)
nx.draw_networkx(G)
plt.show()
```

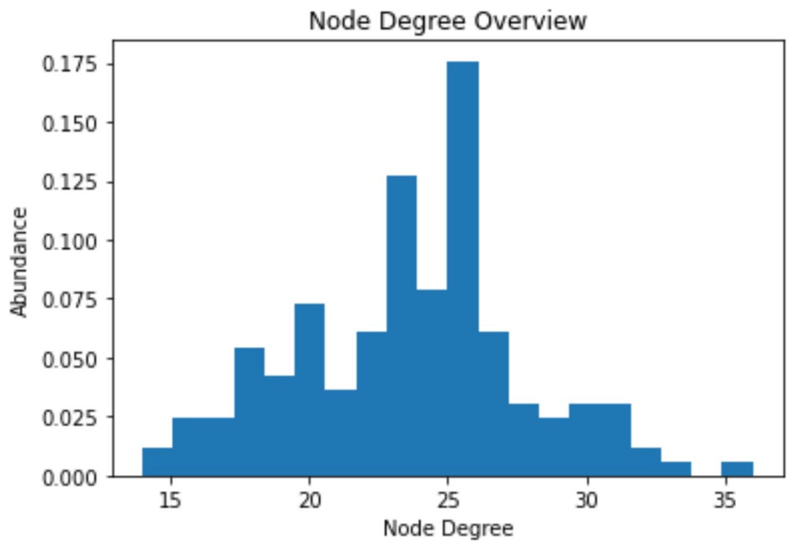


## (A) 2 Analysis of the Network:

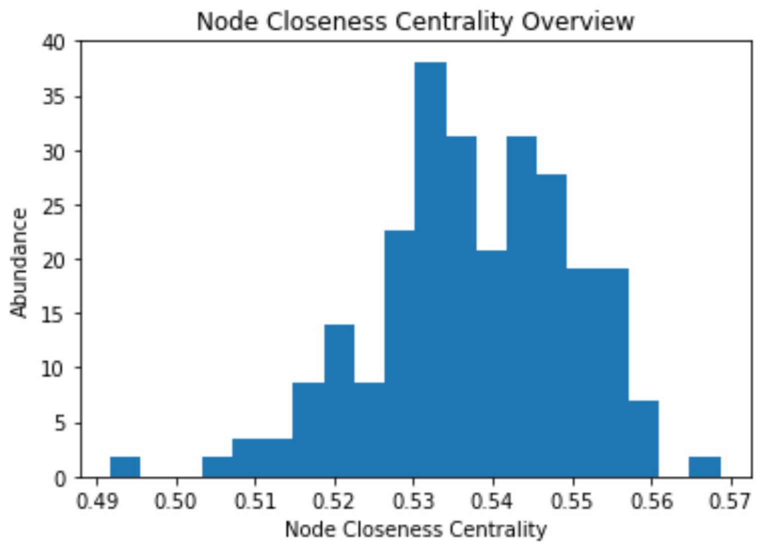
In [177]:
```python
print("Number of connected Components:", len(list(nx.connected_compo
nents(G))))
print("With sizes:")
for i, entry in enumerate(list(nx.connected_components(G))):
    print("\tComponent:",i,"Size:",len(entry))
```

```
Number of connected Components: 1
With sizes:
        Component: 0 Size: 150
```

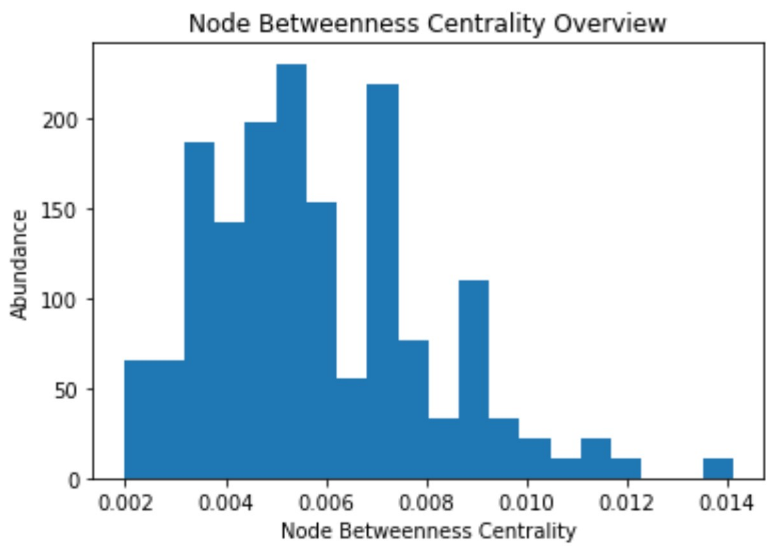
```
In [178]: plt.hist([d for n, d in G.degree()], density=True, bins=20)
          plt.ylabel('Abundance')
          plt.xlabel('Node Degree');
          plt.title('Node Degree Overview')
          plt.show()
```



```
In [179]: plt.hist(nx.algorithms.centrality.closeness_centrality(G).values(),
          density=True, bins=20)
          plt.ylabel('Abundance')
          plt.xlabel('Node Closeness Centrality');
          plt.title('Node Closeness Centrality Overview')
          plt.show()
```

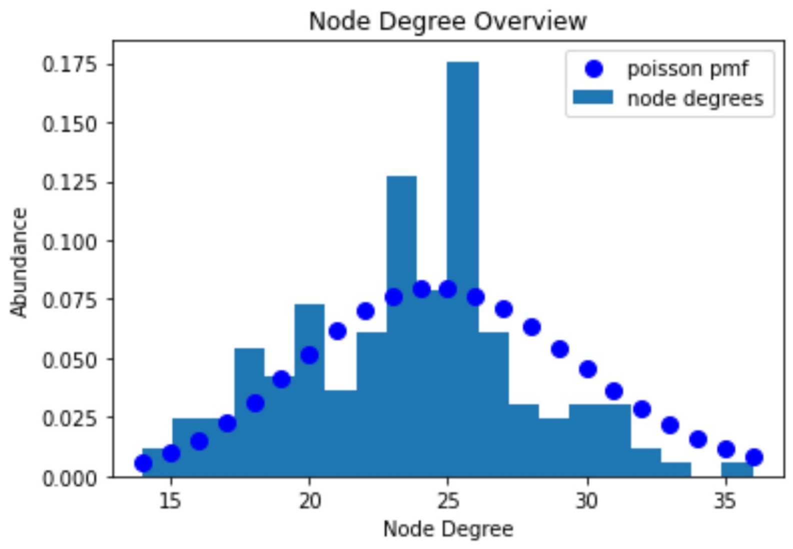


In [180]:
```
plt.hist(nx.algorithms.centrality.betweenness_centrality(G).values
(), density=True, bins=20)
plt.ylabel('Abundance')
plt.xlabel('Node Betweenness Centrality');
plt.title('Node Betweenness Centrality Overview')
plt.show()
```



### (A) 3 Normalize the degree histogram and on top of it plot the theoretically appropriate Poisson probability mass function:

In [181]:
```
plt.hist([d for n, d in G.degree()], density=True, bins=20, label='n
ode degrees')
plt.ylabel('Abundance')
plt.xlabel('Node Degree');
plt.title('Node Degree Overview')
mu = 25
x = np.arange(stats.poisson.ppf(0.01, mu), stats.poisson.ppf(0.99, m
u))
plt.plot(x, stats.poisson.pmf(x, mu), 'bo', ms=8, label='poisson pmf
')
plt.legend()
plt.show()
```



Except for some variability of the histogram, the poisson probability mass function does closely resemble the node degree value distribution.

## (A) 4 Export the network as GraphML file and the node attributes (centralities, degree) as a csv file.

In [182]:
```
nx.readwrite.graphml.write_graphml(G, "erdos_renyi_graph.graphml")
df = pd.DataFrame()
df.index.name = 'Node_ID'
df["degrees"] = [d for n, d in G.degree()]
df["betweenness_centrality"] = nx.algorithms.centrality.betweenness_
centrality(G).values()
df["closeness_centrality"] = nx.algorithms.centrality.closeness_cent
rality(G).values()
df.to_csv("erdos_renyi_graph_data.csv", header=True)
```

## (A) 5 Check what happens when you change the value of p (you do not need to hand in the results for other values of p).

- For smaller p, there typically stays one large connected component and a bunch of unconnected nodes or clusters of size 2, also the closeness centrality increases for most nodes and the betweeness centrality gets a lot smaller overall
- for larger p there is only one connected component and all centrality scores look very normally distributed

**The rest of this exercise follows in the PDF of a Word Sheet attached after this Notebook**

# Assignment 1, Vingron Part, Complex Systems

Paul Vogler (4979420) and Yiftach Kolb (5195763)

## Problem 2 (continued..):

Consider two random variables X and Y from which we drew the following samples:

- x = (0.3, 0.98, 0.54, 0.49, 0.39, 0.14, 0.03, 0.81, 0.65, 0.18)
- y = (0.74, 0.09, 0.48, 0.15, 0.71, 0.8, 0.53, 0.95, 0.63, 0.88)

Therefore the first observation is (x = 0.3, y = 0.74) and so on (10 observations in total). First, bin the data by dividing the interval of [0, 1] into 4 equally wide sub-intervals. Provide the following calculations by hand.

- Bin-Edges: (0, 0.25, 0.5, 0.75, 1)
- Bins (y,x): $\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 2 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{pmatrix}$

A) Calculate the joint probability distribution $p_{X,Y}(x, y)$ of the binned data and write it in the following table:

| Y\|X | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
|---|---|---|---|---|
| $y_1$ | 0 | 0.1 | 0 | 0.1 |
| $y_2$ | 0 | 0 | 0.1 | 0 |
| $y_3$ | 0.1 | 0.2 | 0.1 | 0 |
| $y_4$ | 0.2 | 0 | 0 | 0.1 |

B) Calculate the marginal distributions $p_X(x)$ and $p_Y(y)$
   a. $p_X(x)$: $(0.3, 0.3, 0.2, 0.2)$
   b. $p_Y(y)$: $(0.2, 0.1, 0.4, 0.3)$

C) Calculate the product of the two marginal distributions pY (y) T × pX(x) (matrix multiplication!) and compare it with the joint distribution pX,Y (x, y). Are the variables X and Y stochastically independent? Justify your answer.
   a. $\begin{pmatrix} 0.2 \\ 0.1 \\ 0.4 \\ 0.3 \end{pmatrix} * (0.3 \quad 0.3 \quad 0.2 \quad 0.2) = \begin{pmatrix} 0.06 & 0.06 & 0.04 & 0.04 \\ 0.03 & 0.03 & 0.02 & 0.02 \\ 0.12 & 0.12 & 0.08 & 0.08 \\ 0.09 & 0.09 & 0.06 & 0.06 \end{pmatrix}$
   b. For two independent variables, the joint density is the product of their marginals. This is not the case here. Therefore, the two variables should not be independent.

D) Calculate the conditional distributions $p_{X|Y}(x|y = y_3)$ and $p_{Y|X}(y|x = x_4)$
   a. $p_{X|Y}(x|y = y_3) = \frac{p_{X,Y}(x,\ y=y_3)}{p_Y(y=y_3)} = \frac{(0.1 \quad 0.2 \quad 0.1 \quad 0)}{0.4} = (0.25 \quad 0.5 \quad 0.25 \quad 0)$
   b. $p_{Y|X}(y|x = x_4) = \frac{p_{X,Y}(x=x_4,\ y)}{p_X(x=x_4)} = \frac{(0.1 \quad 0 \quad 0 \quad 0.1)}{0.2} = (0.5 \quad 0 \quad 0 \quad 0.5)$

E) Calculate the joint entropy H(X, Y) and the marginal entropies H(X) and H(Y)
   a. $H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) * \log_2 p(x,y) \ = -(6 * (0.1 * \log_2 0.1) + 2 * (0.2 * \log_2 0.2)) \approx -(6 * -0.332 + 2 * -0.464) = -\left(-1.992 + (-0.928)\right) = 2.92$
   b. $H(X) = -\sum p(x_i) * \log_2 p(x_i) = -\big(2 * (0.3 * \log_2 0.3) + 2 * (0.2 * \log_2 0.2)\big) \approx -(2 * -0.521 + 2 * -0.464) = -(-1.042 + (-0.928)) = 1.97$
   c. $H(Y) = -\sum p(y_i) * \log_2 p(y_i) = -(0.1 * \log_2 0.1 + 0.2 * \log_2 0.2 + 0.3 * \log_2 0.3 + 0.4 * \log_2 0.4) \approx -\big(-0.332 + (-0.464) + (-0.521) + (-0.529)\big) = 1.846$

F) Calculate the conditional entropies H(X|Y) and H(Y|X) using the chain rule.

a. $H(X|Y) = H(X,Y) - H(Y) = 2.92 - 1.846 = 1.074$

b. $H(Y|X) = H(X,Y) - H(X) = 2.92 - 1.97 = 0.95$

G) Calculate the mutual information I(X, Y) using both, the definition and the relation to entropy. Are both results equal? Why?

a. $I(X,Y) = \sum_{x\in X} \sum_{y\in Y} p(x,y) * \log_2 \frac{p(x,y)}{p(x)p(y)} = 0.1 * \log_2 \frac{0.1}{0.3*0.4} + 0.2 * \log_2 \frac{0.2}{0.3*0.3} + 0.1 * \log_2 \frac{0.1}{0.3*0.2} + 0.2 * \log_2 \frac{0.2}{0.3*0.4} + 0.1 * \log_2 \frac{0.1}{0.2*0.1} + 0.1 * \log_2 \frac{0.1}{0.2*0.4} + 0.1 * \log_2 \frac{0.1}{0.2*0.2} + 0.1 * \log_2 \frac{0.1}{0.2*0.3} = 0.895$

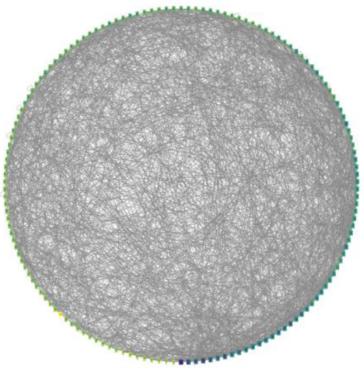
b. $I(X,Y) = H(X) - H(X|Y) = 1.97 - 1.074 = 0.896$

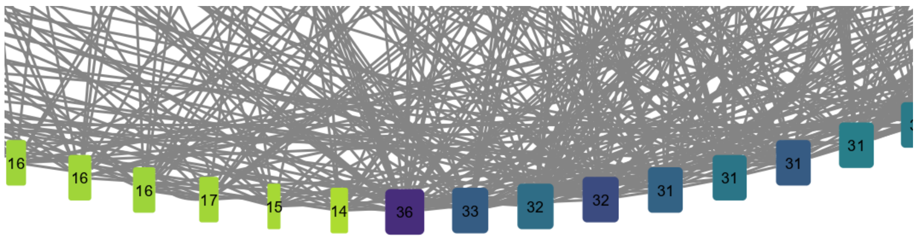
c. $I(X,Y) = H(Y) - H(Y|X) = 1.846 - 0.95 = 0.896$

d. Both results are basically equal, the difference just comes down to rounding precision.

## Problem 3 (continued..):

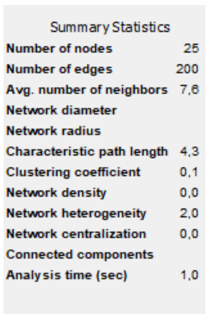
B) Cytoscape visualization

a. 



b.

C) Escherichia coli interactome

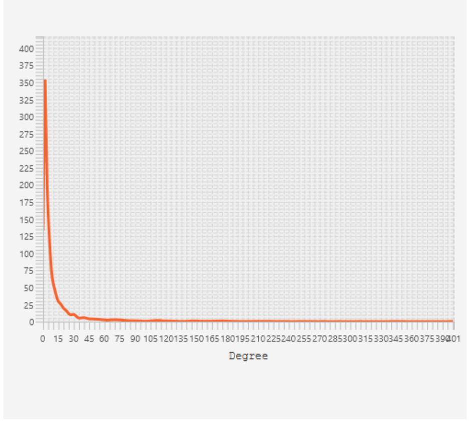
a. 

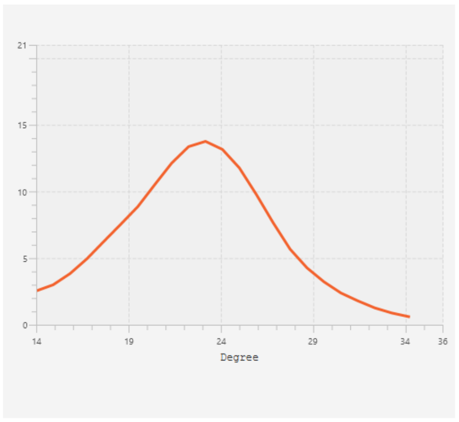
Summary Statistics

| | |
|---|---|
| Number of nodes | 25 |
| Number of edges | 200 |
| Avg. number of neighbors | 7,6 |
| Network diameter | |
| Network radius | |
| Characteristic path length | 4,3 |
| Clustering coefficient | 0,1 |
| Network density | 0,0 |
| Network heterogeneity | 2,0 |
| Network centralization | 0,0 |
| Connected components | |
| Analysis time (sec) | 1,0 |

b. After the filtering of nodes without the Taxonomy ID 83333 and Nodes that do not belong to the largest connected component, there are 25 nodes and 200 edges left.

c. The left Graph shows the Degree Distribution of the E-Coli Network, while the right one shows the Degree Distribution of the Erdős-Rényi Model:





d. These two do not match, the first one is following a exponential decay distribution, while the Erdős-Rényi Model is looking like its degrees are normally distributed.

# Assignment 1 Complex Systems (Vingron part)

Paul Vogler (4979420) and Yiftach Kolb (5195763)

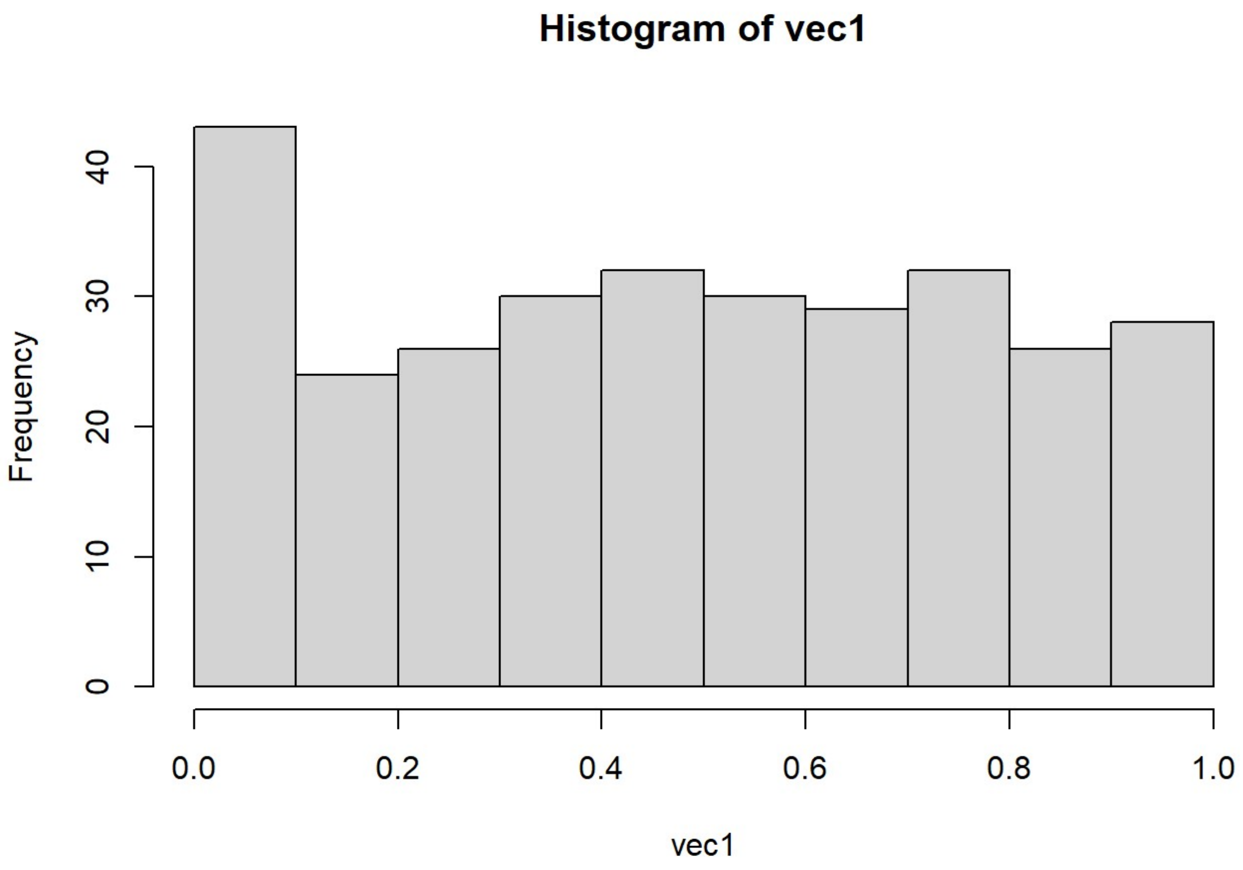
## Exercise 4

```
library(infotheo)
```

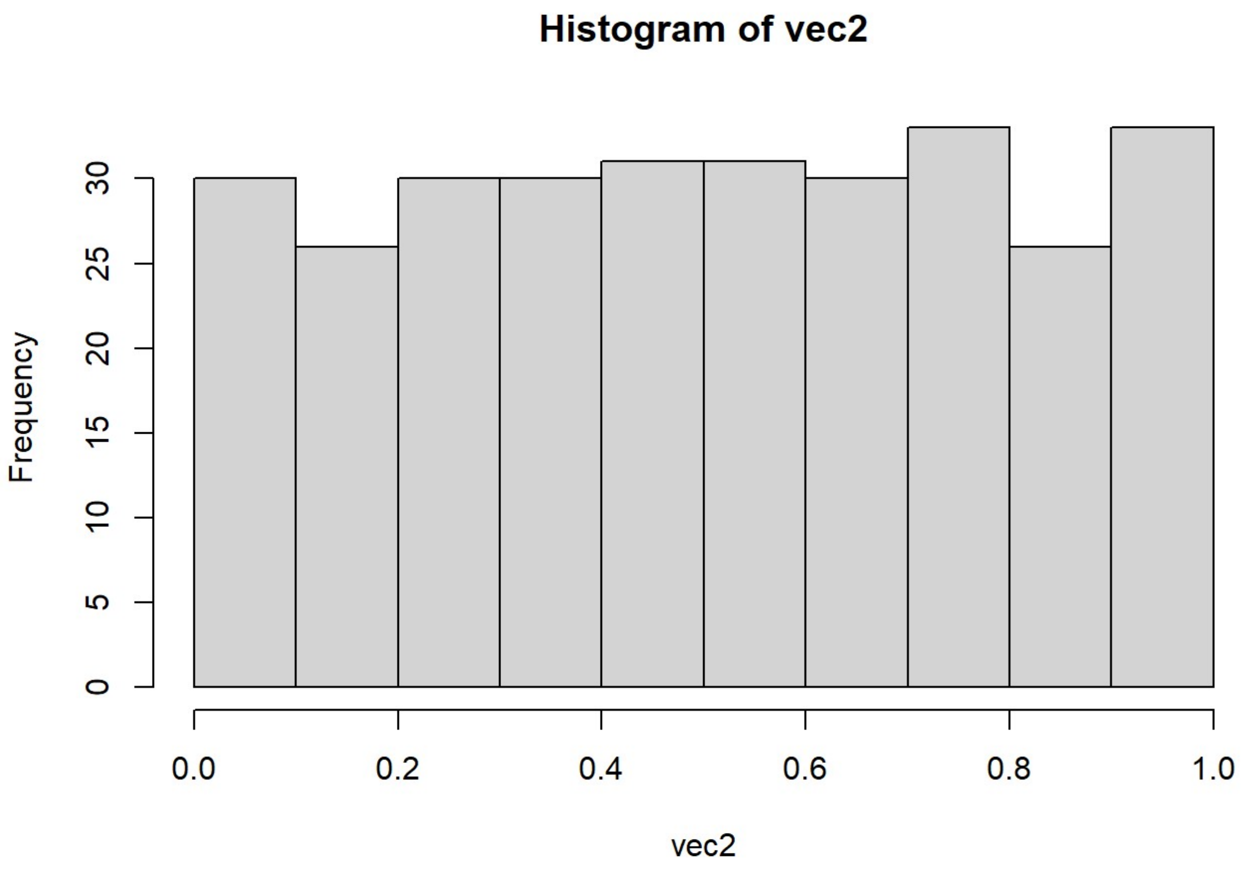
```
## Warning: package 'infotheo' was built under R version 4.0.3
```

```
#library(rmarkdown) #render

# Generate two vectors of length 300 from the continuous uniform distribution:
vec1 <- runif(300, min = 0, max = 1)
vec2 <- runif(300, min = 0, max = 1)
# Discretize the data to the bins of length 0.1:
h1 <- hist(vec1, breaks = seq(from=0, to=1, by=0.1))
```



```
h2 <- hist(vec2, breaks = seq(from=0, to=1, by=0.1))
```



```
# Compute the mutual information:
# We expect the mutual information to be 0 if the variables are independent.
# If they are completely dependent, the mutual information should converge
# towards the entropy of one of the inputs.
mutinformation(h1$counts, h2$counts, method="emp")
```

```
## [1] 0.9162907
```

```
# output: 1.695743
entropy(h1$counts, method="emp")
```

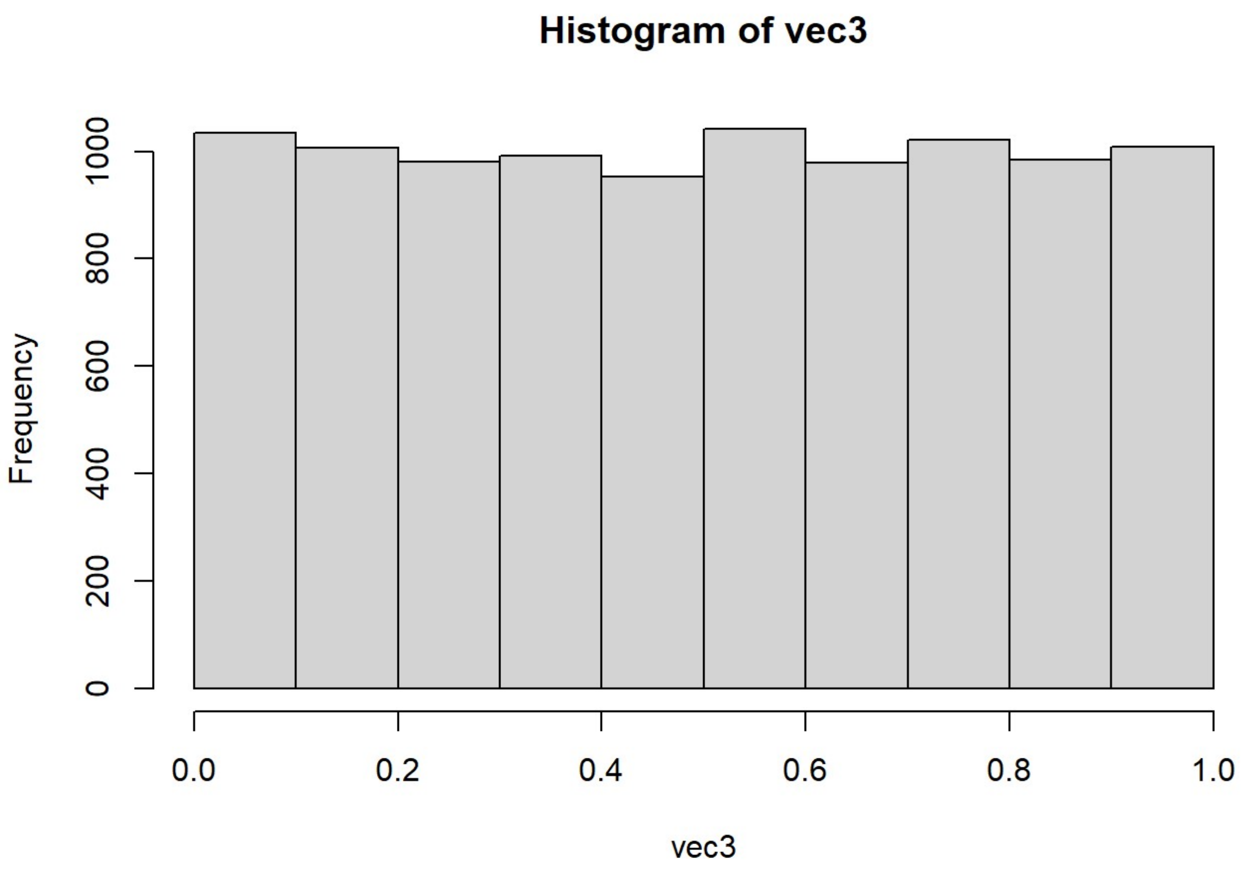
```
## [1] 1.886697
```

```
# output: 1.834372
entropy(h2$counts, method="emp")
```

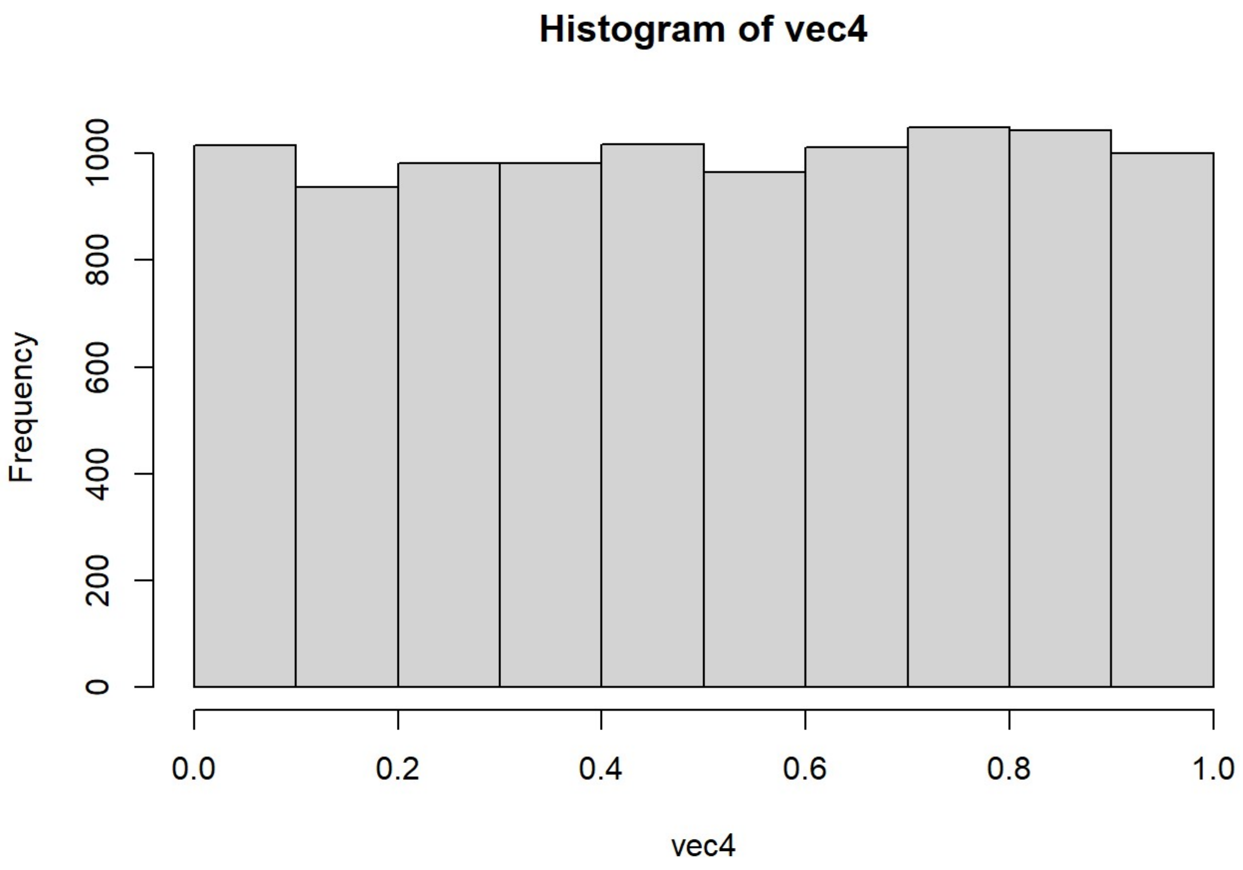
```
## [1] 1.332179
```

```
# output: 2.163956
# We got an output close to the entropies of either of the two inputs,
# therefore they seem to be dependent variables.
# When the number of samples is increased, this also becomes more clear:

# Generate two vectors of length 300 from the continuous uniform distribution:
vec3 <- runif(10000, min = 0, max = 1)
vec4 <- runif(10000, min = 0, max = 1)
# Discretize the data to the bins of length 0.1:
h3 <- hist(vec3, breaks = seq(from=0, to=1, by=0.1))
```



```
h4 <- hist(vec4, breaks = seq(from=0, to=1, by=0.1))
```



```
# Compute the mutual information:
mutinformation(h3$counts, h4$counts, method="emp")
```

```
## [1] 2.302585
```

```
# output: 2.302585
entropy(h3$counts, method="emp")
```

```
## [1] 2.302585
```

```
# output: 2.302585
entropy(h4$counts, method="emp")
```

```
## [1] 2.302585
```

```
# output: 2.302585
# Here the mutual information converged to the entropy of both variables
```