- Module Code: CS3PP19
- Assignment report Title: Summative Coursework
- Student Number: 28016899
- Date (when the work completed): 12th December 2019
- Actual hrs spent for the assignment: 35 hours
- Assignment evaluation (3 key points):
  - This was a great opportunity for me to reinforce what I learned during lectures
  - The length of time given to us to work on the assignment was helpful
  - I got to learn more about how programming concepts can be used in Data Science

Importing all needed libraries:

In [ ]:

```
import pandas as pd #import pandas and rename it as "pd"
import numpy as np #import numpy and rename it as "np"
import math
import matplotlib.pyplot as plt
from scipy import stats
import scipy
import seaborn as sns #import seaborn and rename it as "sns"
import sklearn
import networkx as nx #import networkx and rename it as "nx"
```

TASK 1 – Bike Journey Data Exploratory Data Analysis

Loading the metro.csv file into a pandas data frame

- The pd.read_csv() function was used to load the metro.csv file into the pandas DataFrame. To see the information in the DataFrame, the .head() method was used.

In [ ]:

```
metro = pd.read_csv('metro.csv')
metro.head()
```

Out[ ]:

	trip_id	duration	start_time	end_time	start_station	start_lat	start_lon	end_station	end_lat	end_lon	bike_id
0	94851140	8	2018-07-01 00:04:00	2018-07-01 00:12:00	3058	34.035801	-118.233170	3082	34.046520	-118.237411	6279
1	94851141	8	2018-07-01 00:04:00	2018-07-01 00:12:00	3058	34.035801	-118.233170	3082	34.046520	-118.237411	6518
2	94851138	15	2018-07-01 00:09:00	2018-07-01 00:24:00	4147	34.145248	-118.150070	4174	34.165291	-118.150970	4823
3	94851137	7	2018-07-01 00:22:00	2018-07-01 00:29:00	4157	34.140999	-118.132088	4162	34.147499	-118.148010	6115
4	94851136	35	2018-07-01 00:23:00	2018-07-01 00:58:00	3013	33.779819	-118.263023	3013	33.779819	-118.263023	12055

Find a sensible way to remove the missing values from the data frame, and explain why you have chosen this method.

- **I used the isnull() and sum() functions to determine if there were any missing values in the DataFrame.**

In [ ]:

```
metro.isnull().sum() #check which ones are null and sum across the columns to see number of missing values
```

Out[ ]:

```
trip_id                  0
duration                 0
start_time               0
end_time                 0
start_station            0
start_lat              559
start_lon              559
end_station              0
end_lat               1838
end_lon               1838
bike_id                  0
plan_duration            0
trip_route_category      0
passholder_type          0
dtype: int64
```

- **I then used the dropna() method to remove all missing values from the DataFrame. I used isnull() and sum() functions again to verify again if all the missing values had been dropped.**

In [ ]:

```
metro = metro.dropna() #dropping missing values
metro.isnull().sum() #checking if there are still any missing values
```

Out[ ]:

```
trip_id                0
duration               0
start_time             0
end_time               0
start_station          0
start_lat              0
start_lon              0
end_station            0
end_lat                0
end_lon                0
bike_id                0
plan_duration          0
trip_route_category    0
passholder_type        0
dtype: int64
```

Explore the distribution of the duration variable. You should produce a plot visualising the distribution, and calculate and discuss briefly statistics of the variable.

- **With the 93,199 different trips through the Los Angeles MetroBike Share service, the maximum length covered in one trip was 1440 minutes, and a minimum of 1 minute was covered. The mean period of time used for a trip was 39.15 minutes with a standard deviation of 106.64 minutes.**
- **Since the duration variable is a continuous variable, a histogram was used to visualise its distribution. From the histogram plotted below, the distribution of the duration variable is right skewed and the trip duration of 1440 minutes is an outlier. From the statistics calculated and the plot, 25% of bike users took a trip for 7 minutes or less. 50% took a trip for 14 minutes or less and 75% took the trip for 33 minutes or less.**

minutes or less, 50% took a trip for 14 minutes or less and 75% took the trip for 33 minutes or less.

In [ ]:

```
metro["duration"].describe() #to describe statistics of the variable
```

Out[ ]:

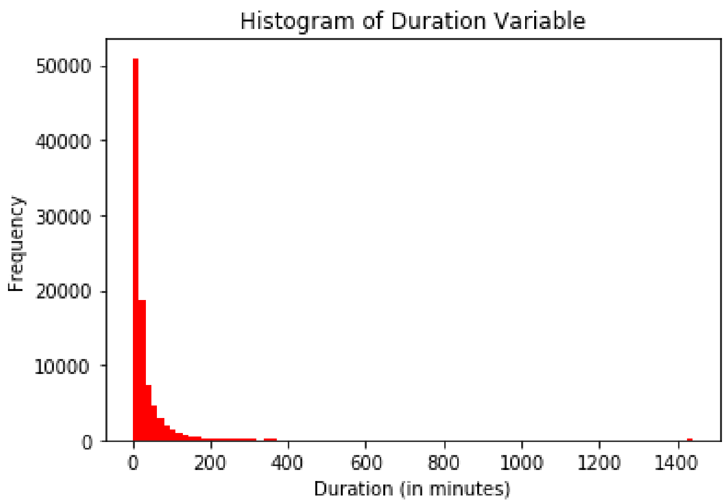
```
count    93199.000000
mean        39.150903
std        106.635812
min          1.000000
25%          7.000000
50%         14.000000
75%         33.000000
max       1440.000000
Name: duration, dtype: float64
```

In [ ]:

```
p=plt.hist(metro['duration'],bins=90, facecolor='r')
plt.xlabel('Duration (in minutes)')
plt.ylabel('Frequency')
plt.title('Histogram of Duration Variable')
```

Out[ ]:

```
Text(0.5, 1.0, 'Histogram of Duration Variable')
```



Produce a plot showing how the distribution of how duration relates to passholder type.

- **The different categories of the passholder type were first identified using the unique() function. In this situation, the series was the passholder_type column of the DataFrame. The groupby() function was then used to categorise the passholder type into the different groups: Monthly Pass, Flex Pass, Walk-up, One Day Pass, and Annual Pass.**
- **The bar plot below shows the relationship between the categories of passholder type and the mean of the duration variable. A bar plot was used since it serves as a good tool for comparing categorical data (passholder type) with a measure (the mean) of the duration. From the bar plot, the mean duration of the One Day Pass holders is the highest with the Monthly Pass holders averagely travelling for the shortest duration of time in a trip.**

In [ ]:

```
metro['passholder_type'].unique() #identify the categories for passholder type
```

Out[ ]:

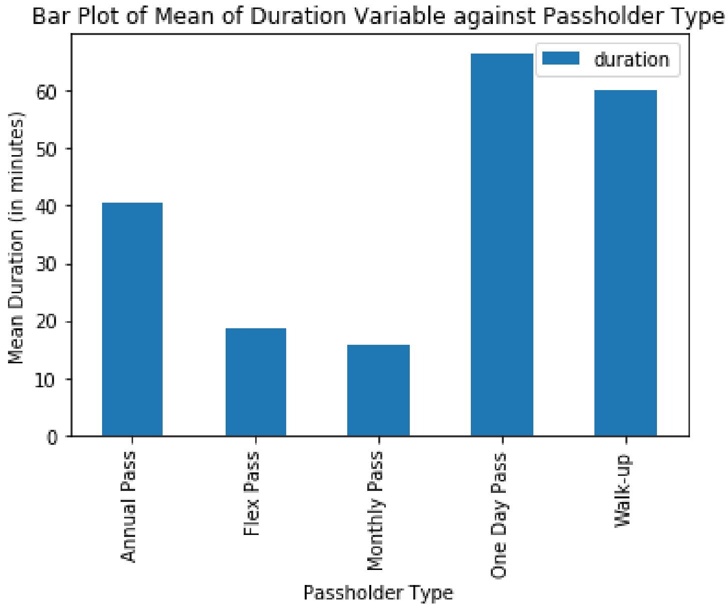
```
array(['Monthly Pass', 'Flex Pass', 'Walk-up', 'One Day Pass',
       'Annual Pass'], dtype=object)
```

In [ ]:

```
passholder_category = metro.groupby("passholder_type")
#generate bar plot for a number of categories
plotA = passholder_category[['duration']].mean().head(10).plot.bar()
plotA.plot()
plt.xlabel('Passholder Type')
plt.ylabel('Mean Duration (in minutes)')
plt.title('Bar Plot of Mean of Duration Variable against Passholder Type')
```

Out[ ]:

```
Text(0.5, 1.0, 'Bar Plot of Mean of Duration Variable against Passholder Type')
```



Perform an appropriate statistical test to check if the mean duration is different between One Day Pass and Flex Pass passholders. What assumptions have you made by using this test?

- **Procedure: I first obtained an array of the duration of the One Day Pass and Flex Pass passholder types using the groupby() function to split the passholder types into the different categories, the get_group method to get a DataFrame of these two groups only and then the .loc method to obtain the "duration" column of the passholder category of interest. I then converted the new DataFrame obtained into an array.**

In [ ]:

```
#get an array of the duration for ONE DAY pass
grouping = metro.groupby(["passholder_type"])
grouping.head()
df=grouping.get_group(("One Day Pass"))
df2 = df.loc[:,'duration']
array1 = pd.Series(df2).values
array1
```

Out[ ]:

```
array([ 7,  7,  7, ..., 22, 22, 78], dtype=int64)
```

In [ ]:

```
#get an array of the duration for FLEX PASS holders
df3=grouping.get_group(("Flex Pass"))
df4= df3.loc[:,'duration']
array2 = pd.Series(df4).values
array2
```

Out[ ]:

```
array([ 8, 10, 12, ..., 74, 11, 16], dtype=int64)
```

- **Assumptions made for the T-test:**
  - **The duration of One Day Pass variable and the duration of Flex Pass variable have the same variance.**
  - **The duration of One Day Pass variable and the duration of Flex Pass variable approximately follow a Normal Distribution**
- **Procedure: I imported ttest_ind from scipy.stats and used it for a two-sided t-test. The Null Hypothesis is that both the duration for One Day Pass and the duration for Flex Pass Holders have the same mean.**

- **Since the t-statistic is quite large (~14.5), there is a large difference between the means of array1 and array2. Also, the p-value is way below 0.05 hence the difference between the means of the two arrays is not zero. Therefore, we reject the Null Hypothesis.**

- **Conclusion: The mean duration is different between One Day Pass and Flex Pass passholders**

In [ ]:

```
from scipy.stats import ttest_ind
test = ttest_ind(array1,array2)
test.statistic, test.pvalue
```

Out[ ]:

```
(14.495319337859621, 7.9422977182875e-47)
```

Convert the start_time and end_time columns to date objects if they are not already.

- **I used the .dtype attribute to initally check if end_time and start_time were date objects.**

In [ ]:

```
metro["end_time"].dtype, metro["start_time"].dtype #(dtype('O'), dtype('O'))
```

Out[ ]:

```
(dtype('O'), dtype('O'))
```

- **Then, I converted them to data objects using the pd.to_datetime method. The pd.to_datetime method is used to convert the needed columns to pandas datetime objects.**

In [ ]:

```
metro['start_time']= pd.to_datetime(metro['start_time'])
metro['end_time']= pd.to_datetime(metro['end_time'])
metro["end_time"].dtype, metro["start_time"].dtype #checking the data type
```

Out[ ]:

```
(dtype('<M8[ns]'), dtype('<M8[ns]'))
```

Create a new column in the data frame that gives the hour of the day that each journey started on.

- **A new column (journey_start_hour_of_day) was created using the dt.hour attribute to extract the hour of the day that each journey started. The .head() method was used to see the first five rows of the updated DataFrame**

In [ ]:

```
metro['journey_start_hour_of_day'] = metro["start_time"].dt.hour
metro.head()
```

Out[ ]:

	trip_id	duration	start_time	end_time	start_station	start_lat	start_lon	end_station	end_lat	end_lon	bike_id
0	94851140	8	2018-07-01 00:04:00	2018-07-01 00:12:00	3058	34.035801	-118.233170	3082	34.046520	-118.237411	6279
1	94851141	8	2018-07-01 00:04:00	2018-07-01 00:12:00	3058	34.035801	-118.233170	3082	34.046520	-118.237411	6518
2	94851138	15	2018-07-01 00:09:00	2018-07-01 00:24:00	4147	34.145248	-118.150070	4174	34.165291	-118.150970	4823
3	94851137	7	2018-07-01 00:22:00	2018-07-01 00:29:00	4157	34.140999	-118.132088	4162	34.147499	-118.148010	6115
4	94851136	35	2018-07-01 00:23:00	2018-07-01 00:58:00	3013	33.779819	-118.263023	3013	33.779819	-118.263023	12055

Explore how the duration variable varies between each journey starting hour of the day, creating a plot to visualise this.

In [ ]:

```
metro.columns
```

Out[ ]:

```
Index(['trip_id', 'duration', 'start_time', 'end_time', 'start_station',
       'start_lat', 'start_lon', 'end_station', 'end_lat', 'end_lon',
       'bike_id', 'plan_duration', 'trip_route_category', 'passholder_type',
       'journey_start_hour_of_day'],
      dtype='object')
```

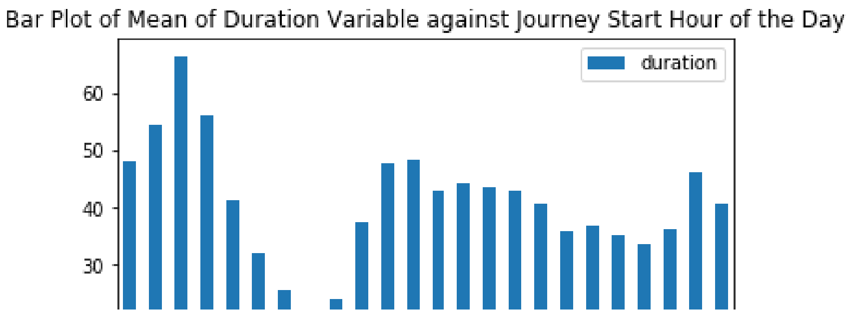
- **I used a bar graph to get a visual representation of the mean of the duration variable and how it varies between each journey starting hour of the day. I used a bar graph since it gives a good visual representation of the different y-values (Mean of Duration Variable) over the different times (Journey Start Hour of the Day).**
- **From the bar graph below, the mean duration of journey was highest in the early hours of the day (0:00 to 3:00), decreased from 4:00 to 7:00 and started increasing from 8:00 onwards with a slight decrease from 11:00 until 21:00.**

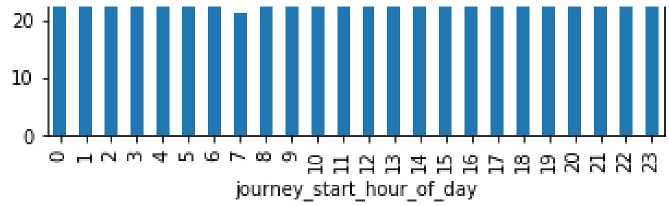
In [ ]:

```
common_journey_start_hour = metro.groupby("journey_start_hour_of_day")
common_journey_start_hour.head()
plotD = common_journey_start_hour[["duration"]].mean().plot.bar()
plotD.plot()
plt.title('Bar Plot of Mean of Duration Variable against Journey Start Hour of the Day')
```

Out[ ]:

```
Text(0.5, 1.0, 'Bar Plot of Mean of Duration Variable against Journey Start Hour of the D
ay')
```





Explore how the distribution of the duration variable varies between each day of the week, creating a plot to visualise this

- **I first created the "Day of the Week" column in the DataFrame, using dt.dayofweek. This produced the whole numbers from 0 to 6 with 0 as Monday, 1 as Tuesday etc. I then created a Bar Plot, similar to the questions above, where I compared the Day of the Week to the Mean Duration of the Trip. Similar to the question above, I used a bar plot since it presents a good visual representation of the categorical data being explored.**

In [ ]:

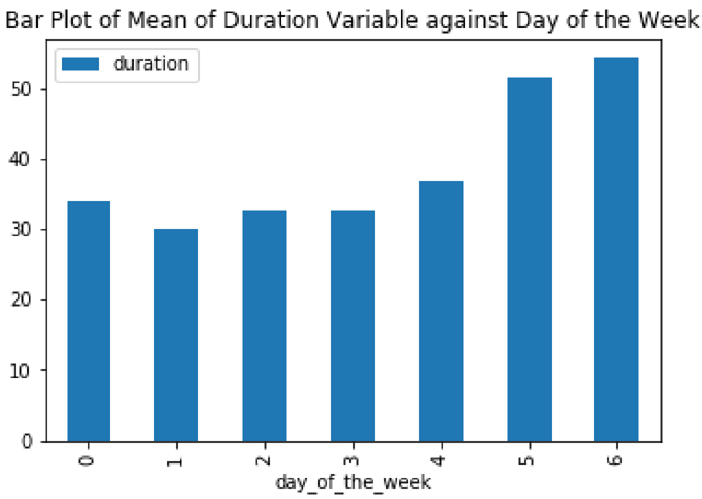
```
#creating day_of_the_week column in the dataframe
metro['day_of_the_week'] =  metro['start_time'].dt.dayofweek
```

In [ ]:

```
#generating a bar plot for each day of the week
common_journey_day_of_week = metro.groupby("day_of_the_week")
common_journey_day_of_week.head()
plotF = common_journey_day_of_week[["duration"]].mean().plot.bar()
plotF.plot()
plt.title('Bar Plot of Mean of Duration Variable against Day of the Week')
```

Out[ ]:

```
Text(0.5, 1.0, 'Bar Plot of Mean of Duration Variable against Day of the Week')
```



Calculate the total numbers of passholders of each type travelling on each week day. Discuss the results.

- **I again used the groupby function here to group the DataFrame into the total numbers of passholders of each type travelling on each week day. I then used count() to make a tally of the total numbers of passholders of each type travelling on each week day. The group is displayed below**
- **From the table below, most Flex Pass and Monthly Pass holders travel during the weekdays (0 = Monday,1 = Tuesday,2 = Wednesday,3 = Thursday,4 = Friday) than during the weekends (5,6). However, most of the One Day Pass and Walk-Up Pass holders travel during the weekend (5= Saturday, 6 = Sunday). The Annual Pass holders only travel during the weekends (5,6)**

In [ ]:

```
metro.columns
metro[["day_of_the_week"]].count()
metro.groupby("day_of_the_week").count()[["passholder_type"]]
passholders_week = metro.groupby(["day_of_the_week", "passholder_type"])
passholders_week.head()
passholders_week[["passholder_type"]].count()
```

Out[ ]:

		passholder_type
day_of_the_week	**passholder_type**	
0	**Flex Pass**	216
	Monthly Pass	6818
	One Day Pass	538
	Walk-up	5235
1	**Flex Pass**	235
	Monthly Pass	7313
	One Day Pass	447
	Walk-up	4960
2	**Flex Pass**	234
	Monthly Pass	6958
	One Day Pass	566
	Walk-up	5236
3	**Flex Pass**	244
	Monthly Pass	7150
	One Day Pass	404
	Walk-up	5432
4	**Flex Pass**	262
	Monthly Pass	6711
	One Day Pass	530
	Walk-up	6161
5	**Annual Pass**	9
	Flex Pass	177
	Monthly Pass	4027
	One Day Pass	845
	Walk-up	8399
6	**Annual Pass**	1
	Flex Pass	198
	Monthly Pass	4125
	One Day Pass	1115
	Walk-up	8653

TASK 2 – Seed shape data

Load the seeds.csv file into a pandas data frame

- I used the pd.read_csv() function to load the seeds.csv file into the pandas dataframe. I then used the .head() method to view the first 5 lines of the dataframe.

In [ ]:

```
seeds = pd.read_csv('seeds.csv') #to read the dataframe from the csv file
seeds.head()
```

Out[ ]:

	area	perimeter	compactness	length	width	asymmetry	groove length
0	15.26	14.84	0.871	5.763	3.312	2.221	5.220
1	14.88	14.57	0.881	5.554	3.333	1.018	4.956
2	14.29	14.09	0.905	5.291	3.337	2.699	4.825
3	13.84	13.94	0.895	5.324	3.379	2.259	4.805
4	16.14	14.99	0.903	5.658	3.562	1.355	5.175

Explore the data, and find a way to cluster the seeds, assigning a cluster to each. Visualise the results, and explain why you have applied the method you have used.

- **I first used .columns to see the different labels for the columns of the seeds dataframe.**

In [ ]:

```
seeds.columns
```

Out[ ]:

```
Index(['area', 'perimeter', 'compactness', 'length', 'width', 'asymmetry',
       'groove length'],
      dtype='object')
```

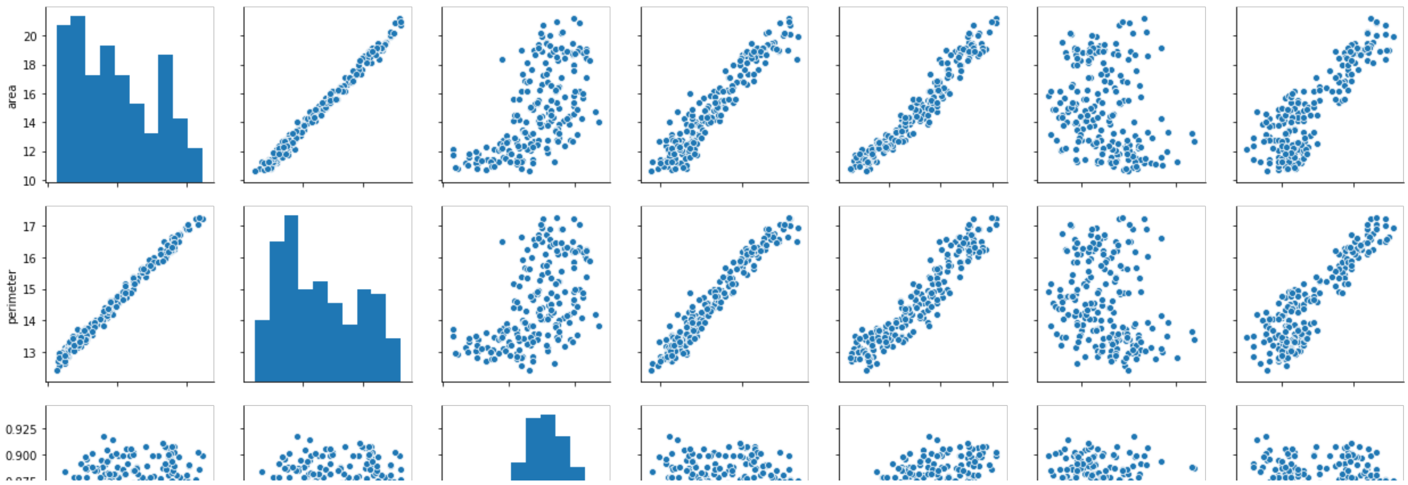
- **To determine the relationship between all combinations of the columns with the other, I used seaborn to visualise these relationships.**
- **From this visualisation, I realized that the following scatterplots showed some sign of clustering of data:**
  - **groove length vs compactness**
  - **groove length vs width**
  - **groove length vs assymmetry**

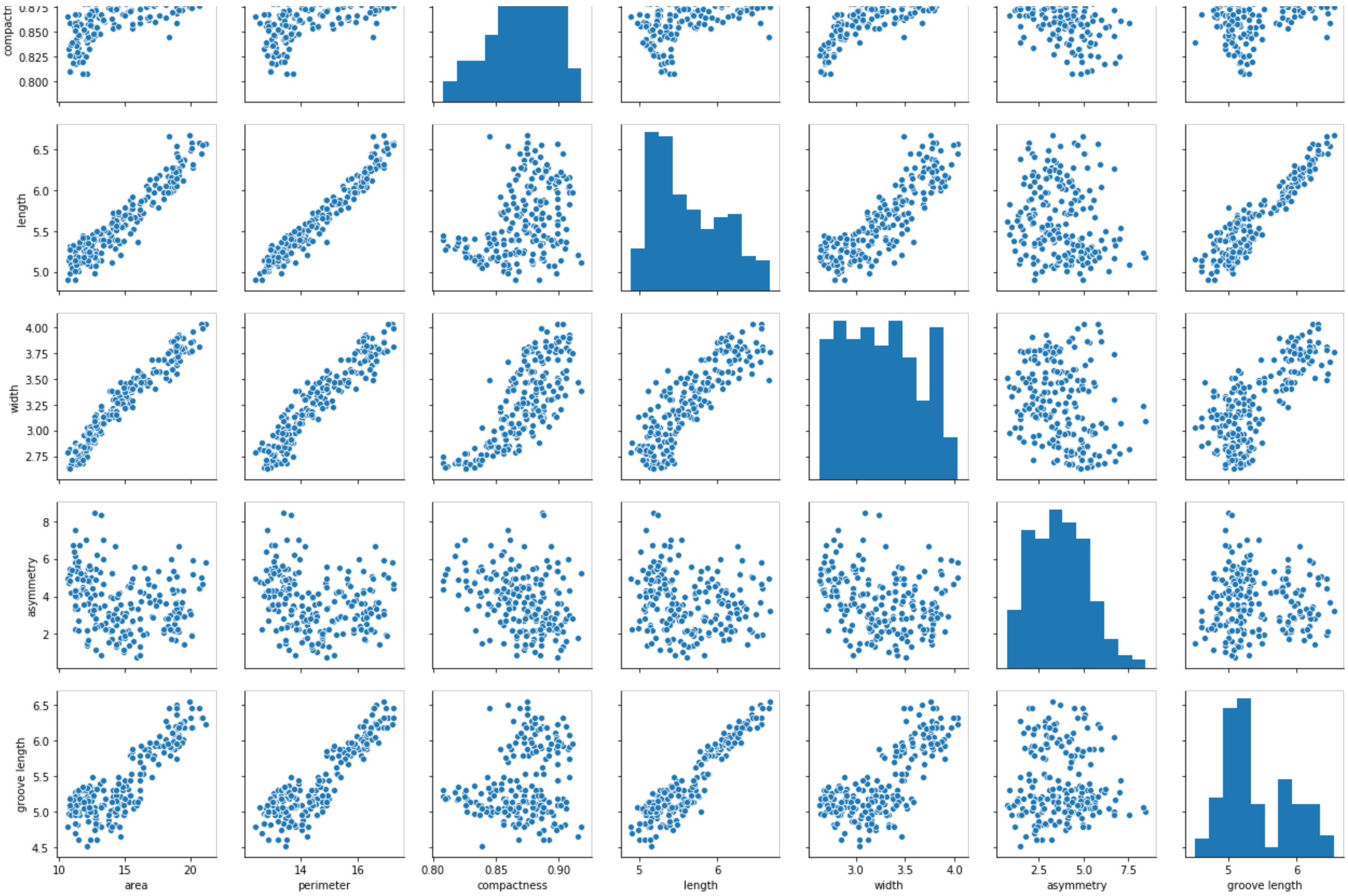
In [ ]:

```
#plotted all to see which of the scatter plots are in clusters
sns.pairplot(seeds)
```

Out[ ]:

```
<seaborn.axisgrid.PairGrid at 0x20806139dc8>
```





- I then clustered the seeds considering groove length versus width using the Gaussian Mixture mddels method. The selection of groove length versus width was made entirely by preference.
- Main reason for using Gaussian Mixture models method:
    - As shown below, the Gaussian Mixture Model helps with grouping the data in their respective clusters using an algorithm.

- The Gaussian Mixture Model built below shows the clustering of the seeds into their respective groups. The model parameters were estimated with the .fit() method. The labels for the data in seeds_updated was predicted using .predict() method and appropriate columns were assigned (in this case: 'groove length', 'width', and 'cluster'.
- Since there are two clusters, a scatter plot was created considering the two columns of the seeds_updated dataframe and color assigned to each cluster

*Sources: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html, https://scikit-learn.org/stable/auto_examples/mixture/plot_gmm_pdf.html#sphx-glr-auto-examples-mixture-plot-gmm-pdf-py

In [ ]:

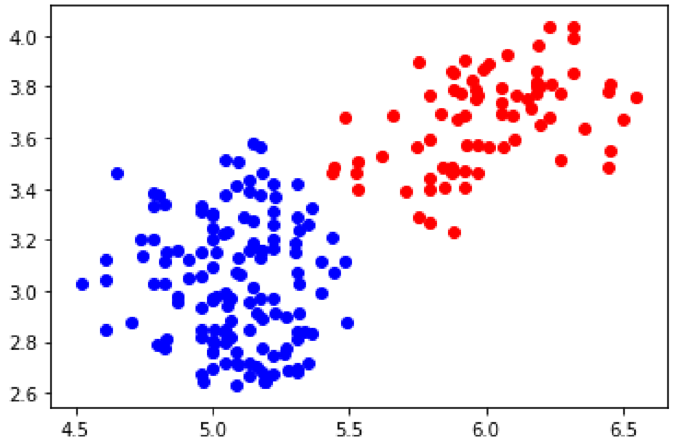
```
#Building a Gaussian Mixture Model
from sklearn.mixture import GaussianMixture
%matplotlib inline
seeds_updated = seeds[['groove length','width']]

gmm = GaussianMixture(n_components=2)
gmm.fit(seeds_updated)

labels = gmm.predict(seeds_updated)
frame = pd.DataFrame(seeds_updated)
frame['cluster'] = labels
frame.columns = ['groove length', 'width', 'cluster']

color=['blue','red']
for j in range(0,2):
    seeds_updated = frame[frame["cluster"]==j]
    plt.scatter(seeds_updated["groove length"],seeds_updated["width"],c=color[j])
```

```
plt.show()
```



A more thorough approach is outlined below:

- **1) I imported GaussianMixture from the scikit learn library. The %matplotlib inline function ensures that the graph produced is displayed in this notebook. I then extracted the groove length and width columns from the seeds dataframe into a new dataframe called "seeds_updated"**

In [ ]:

```
from sklearn.mixture import GaussianMixture
%matplotlib inline
seeds_updated = seeds[['groove length','width']]
```

- **2) I represented my seeds_updated data as a Gaussian Mixture model probability distribution and created an array of each of the columns: groove length and width**

In [ ]:

```
mixturemodel = GaussianMixture(2)
mixturemodel.fit(seeds[["groove length", "width"]])

#creating an array of the seeds length
seeds.length.head()
df7 = seeds.loc[:,"groove length"]
array7 = pd.Series(df7).values


#now, creating an array of the seeds width
seeds.width.head()
df8 = seeds.loc[:,"width"]
array8 = pd.Series(df8).values
```

- **3) I drew a contour plot with the two arrays created. Here, Z represents an array of weighted log probabilities for groove length and width and is obtained using gmm.score_samples()**
- **4) I also generated a scatter plot using plt.scatter().**
- **As the visualisation below shows, the seeds are clustered into two groups**

In [ ]:

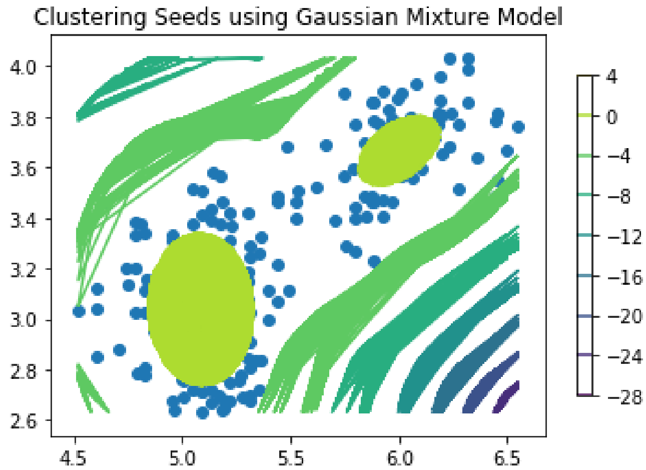
```
#drawing a contour plot
x = array7
y = array8
u = np.linspace(-1,1)
X,Y = np.meshgrid(x,y)

XX = np.array([X.ravel(), Y.ravel()]).T
Z = gmm.score_samples(XX)
Z = Z.reshape((210,210))
```

```
C = plt.contour(X, Y, Z)
C
plt.colorbar(C, shrink=0.8, extend='both')


#Generating the scatter plot
plt.scatter(x,y) #this generates the scatter plot

#code below gives the title of the plot and displays it
plt.title('Clustering Seeds using Gaussian Mixture Model')
plt.show()
```



TASK 3 – Social network analysis

Using networkx, load the social network data in the social-network.csv file.

- **The dataframe was loaded as a graph using nx.read_edgelist() with the delimiters identified as "," and the nodes' data type identified as int**

In [ ]:

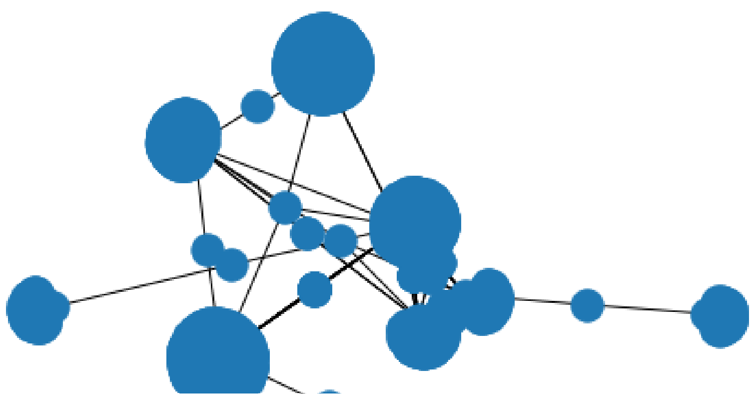
```
import networkx as nx #import network library
G=nx.read_edgelist("social-network.csv", delimiter=',', nodetype=int)
```

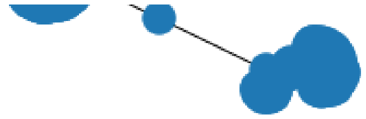
Produce a visualisation of the network and discuss the output.

- **A visualisation of the graph was made using nx.draw()**

In [ ]:

```
nx.draw(G)
```





- **Discussing output below:**
  - **As shown below, the graph has 2888 nodes and 2981 edges. There is only one seperate component that forms part of the network. Therefore, through this component, it is possible to reach all other nodes of that make up this connected component. Also, a maximum of 9 edges are needed for one to move on the shortest path between any two nodes in this network.**

In [ ]:

```
print("The number of nodes in the graph is ", G.number_of_nodes()) #2888
print("The number of edges in the graph is ", G.number_of_edges()) #2981
print("There is", nx.number_connected_components(G), "connected component in the graph.")
#1
print("The diameter of the graph is", nx.diameter(G)) #9    #will return the diameter of
graph g.
```

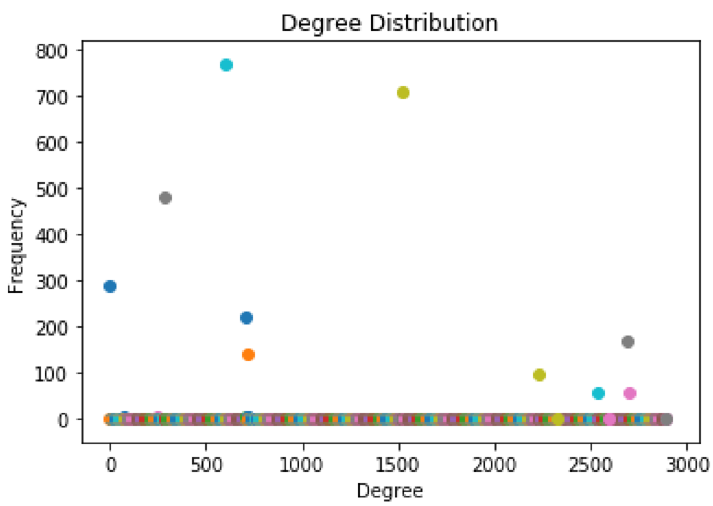
```
The number of nodes in the graph is  2888
The number of edges in the graph is  2981
There is 1 connected component in the graph.
The diameter of the graph is 9
```

Calculate statistics of the network, plot them where relevant, and discuss the results, explaining the meaning of any statistics you have calculated. * NOTE: You can use networkx to calculate statistics of the network, rather than implementing your own Python code to do so.

- **To visualise the degree distribution, a scatter plot was generated. A scatter plot was used as the tool for visualisation since it gives a good visual representation of numerical data in pairs. The scatter plot below shows that most of the nodes have only one edge.**

In [ ]:

```
#plot a scatter plot
for k,v in G.degree():
    plt.scatter(k,v)
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.title("Degree Distribution")
plt.show()
```



- **The clustering coefficient was calculated using nx.clustering on the whole network and a histogram was plotted to visualise this. The histogram indicates that the maximum clustering coefficient is approximately 0. This is further indicated by the if loop below. The average clustering coefficient is calculated with**

This is further indicated by the if loop below. The average clustering coefficient is calculated with nx.average_clustering and turns out to be 0.027. This means that the nodes in this network barely cluster together.

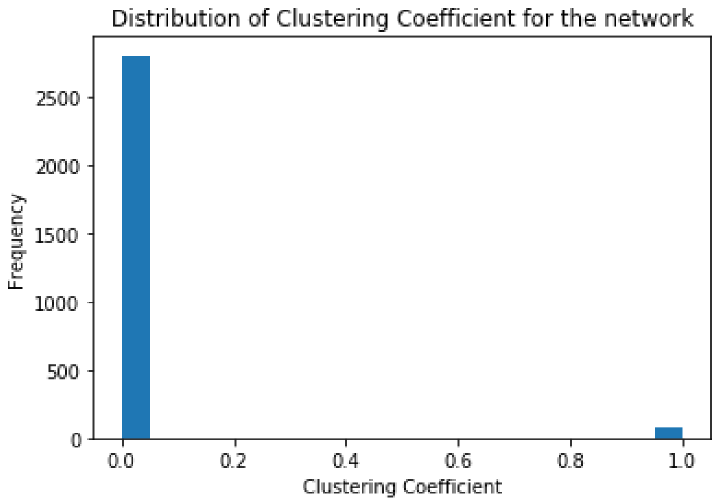
In [ ]:

```
#Clustering Coefficient
cc = nx.clustering(G)
plt.title("Distribution of Clustering Coefficient for the network")
plt.xlabel("Clustering Coefficient")
plt.ylabel("Frequency")
plt.hist(list(cc.values()),bins=20)

maxi = cc[1]
for m in cc:
    if cc[m]>maxi:
        maxi == cc[m]
print("The maximum clustering coefficient is", maxi)


ccc = nx.average_clustering(G)
print("The average clustering coefficient is", ccc)
```

```
The maximum clustering coefficient is 2.436587802441461e-05
The average clustering coefficient is 0.027247421431211827
```



- **The Betweenness Centrality was calculated using nx.betweenness_centrality(G). A histogram was also drawn to get a visual representation of the betweeness centrality. As indicated in the histogram, most nodes have a betweenness centrality of 0 hence removing these particular nodes from the network will not significantly affect the function of the network.**

In [ ]:

```
c4 = nx.betweenness_centrality(G)
c4
```

Out[ ]:

```
{1: 0.1860965105682874,
 2: 0.0,
 3: 0.0,
 4: 0.0,
 5: 0.0,
 6: 0.0,
 7: 0.0,
 8: 0.0,
 9: 0.0,
 10: 0.0,
 11: 0.0,
 12: 0.0,
 13: 0.0,
 14: 0.0,
 15: 0.0,
```

16: 0.0,
17: 0.0,
18: 0.0,
19: 0.0,
20: 0.0,
21: 0.0,
22: 0.0,
23: 0.0,
24: 0.0,
25: 0.0,
26: 0.0,
27: 0.0,
28: 0.0,
29: 0.0,
30: 0.0,
31: 0.0,
32: 0.0,
33: 0.0,
34: 0.0,
35: 0.024013902261217815,
36: 0.0,
37: 0.0,
38: 0.0,
39: 0.0,
40: 0.0,
41: 0.0,
42: 0.0,
43: 0.0,
44: 0.0,
45: 0.0,
46: 0.0,
47: 0.0,
48: 0.0,
49: 0.0,
50: 0.0,
51: 0.0,
52: 0.0,
53: 0.0,
54: 0.0,
55: 0.0,
56: 0.0,
57: 0.0,
58: 0.0,
59: 0.0,
60: 0.0,
61: 0.0,
62: 0.0,
63: 0.0,
64: 0.0,
65: 0.0,
66: 0.0,
67: 0.0,
68: 0.0,
69: 0.02180347729360568,
70: 0.0,
71: 0.008882130911919658,
72: 0.0,
73: 0.0,
74: 0.0,
75: 0.0,
76: 0.0,
77: 0.0,
78: 0.0,
79: 0.0,
80: 0.0,
81: 0.0,
82: 0.0,
83: 0.0,
84: 0.0,
85: 0.0,
86: 0.0,
87: 0.0,

```
    88: 0.0,
    89: 0.0,
    90: 0.0033165636127187605,
    91: 0.0,
    92: 0.0,
    93: 0.0,
    94: 0.0,
    95: 0.0,
    96: 0.0,
    97: 0.0,
    98: 0.0,
    99: 0.0,
    100: 0.0,
    101: 0.0,
    102: 0.0,
    103: 0.0,
    104: 0.0,
    105: 0.0,
    106: 0.0,
    107: 0.0,
    108: 0.0,
    109: 0.0,
    110: 0.0,
    111: 0.0,
    112: 0.0,
    113: 0.0,
    114: 0.0,
    115: 0.0,
    116: 0.0,
    117: 0.0,
    118: 0.0,
    119: 0.0,
    120: 0.0,
    121: 0.0,
    122: 0.0,
    123: 0.0,
    124: 0.0,
    125: 0.0,
    126: 0.0,
    127: 0.0,
    128: 0.0,
    129: 0.0,
    130: 0.0,
    131: 0.0,
    132: 0.0,
    133: 0.0,
    134: 0.0,
    135: 0.0,
    136: 0.0,
    137: 0.0,
    138: 0.0,
    139: 0.0,
    140: 0.0,
    141: 0.0,
    142: 0.0,
    143: 0.0,
    144: 0.0,
    145: 0.0,
    146: 0.0,
    147: 0.0,
    148: 0.0,
    149: 0.0,
    150: 0.0,
    151: 0.0,
    152: 0.0,
    153: 0.0,
    154: 0.0,
    155: 0.0,
    156: 0.0,
    157: 0.0,
    158: 0.0,
    159: 0.0,
```

```
    160: 0.0,
    161: 0.0,
    162: 0.0,
    163: 0.0,
    164: 0.0,
    165: 0.0,
    166: 0.0,
    167: 0.0,
    168: 0.0,
    169: 0.0,
    170: 0.0,
    171: 0.0,
    172: 0.0,
    173: 0.0,
    174: 0.0,
    175: 0.0,
    176: 0.0,
    177: 0.0,
    178: 0.0,
    179: 0.0,
    180: 0.0,
    181: 0.0,
    182: 0.0,
    183: 0.0,
    184: 0.0,
    185: 0.0,
    186: 0.0,
    187: 0.0,
    188: 0.0,
    189: 0.0,
    190: 0.0,
    191: 0.0,
    192: 0.0,
    193: 0.0,
    194: 0.0,
    195: 0.0,
    196: 0.0,
    197: 0.0,
    198: 0.0,
    199: 0.0,
    200: 0.0,
    201: 0.0,
    202: 0.0,
    203: 0.0,
    204: 0.0,
    205: 0.0,
    206: 0.0,
    207: 0.0,
    208: 0.0,
    209: 0.0,
    210: 0.0,
    211: 0.0,
    212: 0.0,
    213: 0.0,
    214: 0.0,
    215: 0.0,
    216: 0.0,
    217: 0.0033165636127187605,
    218: 0.0,
    219: 0.0,
    220: 0.0,
    221: 0.0,
    222: 0.0,
    223: 0.0,
    224: 0.0,
    225: 0.0,
    226: 0.0,
    227: 0.0,
    228: 0.0,
    229: 0.0,
    230: 0.0,
    231: 0.0,
```

232: 0.0,
233: 0.0,
234: 0.0,
235: 0.0,
236: 0.0,
237: 0.0,
238: 0.0,
239: 0.0,
240: 0.0,
241: 0.0,
242: 0.0,
243: 0.0,
244: 0.0,
245: 0.0,
246: 0.0,
247: 0.24124220674273653,
248: 0.0,
249: 0.0,
250: 0.0,
251: 0.0,
252: 0.0,
253: 0.0,
254: 0.0,
255: 0.0,
256: 0.0,
257: 0.0,
258: 0.0,
259: 0.0,
260: 0.0,
261: 0.0,
262: 0.0,
263: 0.0,
264: 0.0,
265: 0.0,
266: 0.0,
267: 0.0,
268: 0.0,
269: 0.0,
270: 0.0,
271: 0.0,
272: 0.0,
273: 0.0,
274: 0.0,
275: 0.0,
276: 0.0,
277: 0.0,
278: 0.0,
279: 0.0,
280: 0.0,
281: 0.0,
282: 0.0,
283: 0.0,
284: 0.0,
285: 0.0,
286: 0.0,
287: 0.0,
288: 0.46612992918844975,
1525: 0.4294450041419194,
603: 0.5497065448918781,
710: 0.12724689931998354,
714: 0.11276804568283787,
289: 0.0,
290: 0.0,
291: 0.0,
292: 0.0,
293: 0.0,
294: 0.0,
295: 0.0,
296: 0.0,
297: 0.0,
298: 0.0,
299: 0.0,

```
300: 0.0,
301: 0.0,
302: 0.0,
303: 0.0,
304: 0.0,
305: 0.0,
306: 0.0,
307: 0.0,
308: 0.0,
309: 0.0,
310: 0.0,
311: 0.0,
312: 0.0,
313: 0.0,
314: 0.0,
315: 0.0,
316: 0.0,
317: 0.0,
318: 0.0,
319: 0.0,
320: 0.0,
321: 0.0,
322: 0.0,
323: 0.0,
324: 0.0,
325: 0.0,
326: 0.0,
327: 0.0,
328: 0.0,
329: 0.0,
330: 0.0,
331: 0.0,
332: 0.0,
333: 0.0,
334: 0.0,
335: 0.06496251387141584,
336: 0.0,
337: 0.0,
338: 0.0,
339: 0.0,
340: 0.0,
341: 0.0,
342: 0.0,
343: 0.0,
344: 0.0,
345: 0.0,
346: 0.0,
347: 0.0,
348: 0.0,
349: 0.0,
350: 0.0,
351: 0.0,
352: 0.0,
353: 0.0,
354: 0.0,
355: 0.0,
356: 0.0,
357: 0.0,
358: 0.0,
359: 0.0,
360: 0.0,
361: 0.0,
362: 0.0,
363: 0.0,
364: 0.0,
365: 0.0,
366: 0.0,
367: 0.0,
368: 0.0,
369: 0.0,
370: 0.0,
371: 0.0,
```

372: 0.0,
373: 0.0,
374: 0.0,
375: 0.0,
376: 0.0,
377: 0.0,
378: 0.0,
379: 0.0,
380: 0.0,
381: 0.0,
382: 0.0,
383: 0.0,
384: 0.0,
385: 0.0,
386: 0.0,
387: 0.0,
388: 0.0,
389: 0.0,
390: 0.0,
391: 0.0,
392: 0.0,
393: 0.0,
394: 0.0,
395: 0.0,
396: 0.0,
397: 0.0,
398: 0.0,
399: 0.0,
400: 0.0,
401: 0.0,
402: 0.0,
403: 0.0,
404: 0.0,
405: 0.0,
406: 0.0,
407: 0.0,
408: 0.0,
409: 0.0,
410: 0.0,
411: 0.0,
412: 0.0,
413: 0.0,
414: 0.0,
415: 0.0,
416: 0.0,
417: 0.0,
418: 0.0,
419: 0.0,
420: 0.0,
421: 0.0,
422: 0.0,
423: 0.0,
424: 0.0,
425: 0.0,
426: 0.0,
427: 0.0,
428: 0.0,
429: 0.0,
430: 0.0,
431: 0.0,
432: 0.0,
433: 0.0,
434: 0.0,
435: 0.0,
436: 0.0,
437: 0.0,
438: 0.0,
439: 0.0,
440: 0.0,
441: 0.0,
442: 0.0,
443: 0.0,

```
    444: 0.0,
    445: 0.0,
    446: 0.0,
    447: 0.0,
    448: 0.0,
    449: 0.0,
    450: 0.0,
    451: 0.0,
    452: 0.0,
    453: 0.0,
    454: 0.0,
    455: 0.0,
    456: 0.0,
    457: 0.0,
    458: 0.0,
    459: 0.0,
    460: 0.0,
    461: 0.0,
    462: 0.0,
    463: 0.0,
    464: 0.0,
    465: 0.0,
    466: 0.0,
    467: 0.0,
    468: 0.0,
    469: 0.0,
    470: 0.0,
    471: 0.0,
    472: 0.0,
    473: 0.0,
    474: 0.0,
    475: 0.0,
    476: 0.0,
    477: 0.0,
    478: 0.0,
    479: 0.0,
    480: 0.0,
    481: 0.0,
    482: 0.0,
    483: 0.0,
    484: 0.0,
    485: 0.0,
    486: 0.0,
    487: 0.0,
    488: 0.0,
    489: 0.0,
    490: 0.0,
    491: 0.0,
    492: 0.0,
    493: 0.0,
    494: 0.0,
    495: 0.0,
    496: 0.0,
    497: 0.0,
    498: 0.0,
    499: 0.0,
    500: 0.0,
    501: 0.0,
    502: 0.0,
    503: 0.0,
    504: 0.0,
    505: 0.0,
    506: 0.0,
    507: 0.0,
    508: 0.0,
    509: 0.0,
    510: 0.0,
    511: 0.0,
    512: 0.0,
    513: 0.0,
    514: 0.0,
    515: 0.0,
```

516: 0.0,
517: 0.0,
518: 0.0,
519: 0.0,
520: 0.0,
521: 0.0,
522: 0.0,
523: 0.0,
524: 0.0,
525: 0.0,
526: 0.0,
527: 0.0,
528: 0.0,
529: 0.0,
530: 0.0,
531: 0.0,
532: 0.0,
533: 0.0,
534: 0.0,
535: 0.0,
536: 0.0,
537: 0.0,
538: 0.0,
539: 0.0,
540: 0.0,
541: 0.0,
542: 0.0,
543: 0.0,
544: 0.0,
545: 0.0,
546: 0.0,
547: 0.0,
548: 0.0,
549: 0.0,
550: 0.0,
551: 0.0,
552: 0.0,
553: 0.0,
554: 0.0,
555: 0.0,
556: 0.0,
557: 0.0,
558: 0.0,
559: 0.0,
560: 0.0,
561: 0.0,
562: 0.0,
563: 0.0,
564: 0.0,
565: 0.0,
566: 0.0,
567: 0.0,
568: 0.0,
569: 0.0,
570: 0.0,
571: 0.0,
572: 0.0,
573: 0.0,
574: 0.0,
575: 0.0,
576: 0.0,
577: 0.0,
578: 0.0,
579: 0.0,
580: 0.0,
581: 0.0,
582: 0.0,
583: 0.0,
584: 0.0,
585: 0.0,
586: 0.0,
587: 0.0,

588: 0.0,
589: 0.0,
590: 0.0,
591: 0.0,
592: 0.0,
593: 0.0,
594: 0.0,
595: 0.0,
596: 0.0,
597: 0.0,
598: 0.0,
599: 0.0,
600: 0.0,
601: 0.0,
602: 0.0,
604: 0.0,
605: 0.0,
606: 0.0,
607: 0.0,
608: 0.0,
609: 0.0,
610: 0.0,
611: 0.0,
612: 0.0,
613: 0.0,
614: 0.0,
615: 0.0,
616: 0.0,
617: 0.0,
618: 0.0,
619: 0.0,
620: 0.0,
621: 0.0,
622: 0.0,
623: 0.0,
624: 0.0,
625: 0.0,
626: 0.0,
627: 0.0,
628: 0.0,
629: 0.0,
630: 0.0,
631: 0.0,
632: 0.0,
633: 0.0,
634: 0.0,
635: 0.0,
636: 0.0,
637: 0.0,
638: 0.0,
639: 0.0,
640: 0.0,
641: 0.0,
642: 0.0,
643: 0.0,
644: 0.0,
645: 0.0,
646: 0.0,
647: 0.0,
648: 0.0,
649: 0.0,
650: 0.0,
651: 0.0,
652: 0.0,
653: 0.0,
654: 0.0,
655: 0.0,
656: 0.0,
657: 0.0,
658: 0.0,
659: 0.0,
660: 0.0,

661: 0.0,
662: 0.0,
663: 0.0,
664: 0.0,
665: 0.0,
666: 0.0,
667: 0.0,
668: 0.0,
669: 0.0,
670: 0.0,
671: 0.0,
672: 0.0,
673: 0.0,
674: 0.0,
675: 0.0,
676: 0.0,
677: 0.0,
678: 0.0,
679: 0.0,
680: 0.0,
681: 0.0,
682: 0.0,
683: 0.0,
684: 0.0,
685: 0.0,
686: 0.0,
687: 0.0,
688: 0.0,
689: 0.0,
690: 0.0,
691: 0.0,
692: 0.0,
693: 0.0,
694: 0.0,
695: 0.0,
696: 0.0,
697: 0.0,
698: 0.0,
699: 0.0,
700: 0.0,
701: 0.0,
702: 0.0,
703: 0.0,
704: 0.0,
705: 0.0,
706: 0.0,
707: 0.0,
708: 0.0,
709: 0.0,
711: 0.0,
712: 0.0,
713: 0.0,
715: 0.0,
716: 0.0630490806278828,
717: 0.0,
718: 0.0,
719: 0.0630490806278828,
720: 0.0,
721: 0.0,
722: 0.0,
723: 0.0,
724: 0.0,
725: 0.0,
726: 0.0,
727: 0.0,
728: 0.0,
729: 0.0,
730: 0.0,
731: 0.0,
732: 0.0,
733: 0.0,
734: 0.0,

```
735: 0.0,
736: 0.0,
737: 0.0,
738: 0.0,
739: 0.0,
740: 0.0,
741: 0.0,
742: 0.0,
743: 0.0,
744: 0.0,
745: 0.0,
746: 0.0,
747: 0.0,
748: 0.0,
749: 0.0,
750: 0.0,
751: 0.0,
752: 0.0,
753: 0.0,
754: 0.0,
755: 0.0,
756: 0.0,
757: 0.0,
758: 0.0,
759: 0.0,
760: 0.0,
761: 0.0,
762: 0.0,
763: 0.0,
764: 0.0,
765: 0.0,
766: 0.0,
767: 0.0,
2232: 0.06541043188081637,
768: 0.0,
769: 0.0,
770: 0.0,
771: 0.0,
772: 0.0,
773: 0.0,
774: 0.0,
775: 0.0,
776: 0.0,
777: 0.0,
778: 0.0,
779: 0.0,
780: 0.0,
781: 0.0,
782: 0.0,
783: 0.0,
784: 0.0,
785: 0.0,
786: 0.0,
787: 0.0,
788: 0.0,
789: 0.0,
790: 0.0,
791: 0.0,
792: 0.0,
793: 0.0,
794: 0.0,
795: 0.0,
796: 0.0,
797: 0.0,
798: 0.0,
799: 0.0,
800: 0.0,
801: 0.0,
802: 0.0,
803: 0.0,
804: 0.0,
805: 0.0,
```

806: 0.0,
807: 0.0,
808: 0.0,
809: 0.0,
810: 0.0,
811: 0.0,
812: 0.0,
813: 0.0,
814: 0.0,
815: 0.0,
816: 0.0,
817: 0.0,
818: 0.0,
819: 0.0,
820: 0.0,
821: 0.0,
822: 0.0,
823: 0.0,
824: 0.0,
825: 0.0,
826: 0.0,
827: 0.0,
828: 0.0,
829: 0.0,
830: 0.0,
831: 0.0,
832: 0.0,
833: 0.0,
834: 0.0,
835: 0.0,
836: 0.0,
837: 0.0,
838: 0.0,
839: 0.0,
840: 0.0,
841: 0.0,
842: 0.0,
843: 0.0,
844: 0.0,
845: 0.0,
846: 0.0,
847: 0.0,
848: 0.0,
849: 0.0,
850: 0.0,
851: 0.0,
852: 0.0,
853: 0.0,
854: 0.0,
855: 0.0,
856: 0.0,
857: 0.0,
858: 0.0,
859: 0.0,
860: 0.0,
861: 0.0,
862: 0.0,
863: 0.0,
864: 0.0,
865: 0.0,
866: 0.0,
867: 0.0,
868: 0.0,
869: 0.0,
870: 0.0,
871: 0.0,
872: 0.0,
873: 0.0,
874: 0.0,
875: 0.0,
876: 0.0,
877: 0.0,

878: 0.0,
879: 0.0,
880: 0.0,
881: 0.0,
882: 0.0,
883: 0.0,
884: 0.0,
885: 0.0,
886: 0.0,
887: 0.0,
888: 0.0,
889: 0.0,
890: 0.0,
891: 0.0,
892: 0.0,
893: 0.0,
894: 0.0,
895: 0.0,
896: 0.0,
897: 0.0,
898: 0.0,
899: 0.0,
900: 0.0,
901: 0.0,
902: 0.0,
903: 0.0,
904: 0.0,
905: 0.0,
906: 0.0,
907: 0.0,
908: 0.0,
909: 0.0,
910: 0.0,
911: 0.0,
912: 0.0,
913: 0.0,
914: 0.0,
915: 0.0,
916: 0.0,
917: 0.0,
918: 0.0,
919: 0.0,
920: 0.0,
921: 0.0,
922: 0.0,
923: 0.0,
924: 0.0,
925: 0.0,
926: 0.0,
927: 0.0,
928: 0.0,
929: 0.0,
930: 0.0,
931: 0.0,
932: 0.0,
933: 0.0,
934: 0.0,
935: 0.0,
936: 0.0,
937: 0.0,
938: 0.0,
939: 0.0,
940: 0.0,
941: 0.0,
942: 0.0,
943: 0.0,
944: 0.0,
945: 0.0,
946: 0.0,
947: 0.0,
948: 0.0,
949: 0.0,

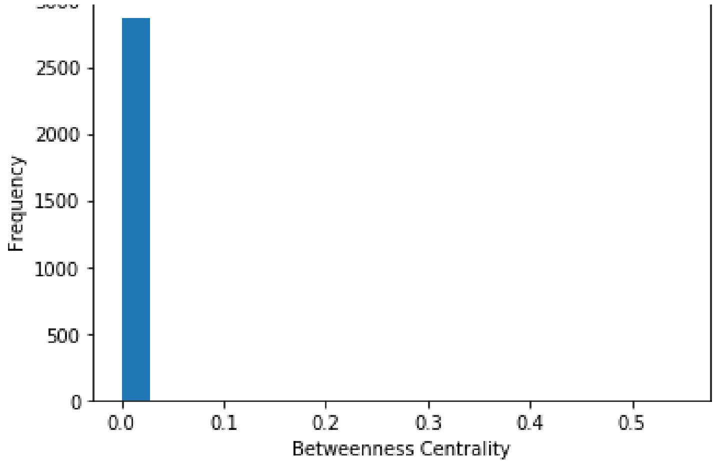
```
 950: 0.0,
 951: 0.0,
 952: 0.0,
 953: 0.0,
 954: 0.0,
 955: 0.0,
 956: 0.0,
 957: 0.0,
 958: 0.0,
 959: 0.0,
 960: 0.0,
 961: 0.0,
 962: 0.0,
 963: 0.0,
 964: 0.0,
 965: 0.0,
 966: 0.0,
 967: 0.0,
 968: 0.0,
 969: 0.0,
 970: 0.0,
 971: 0.0,
 972: 0.0,
 973: 0.0,
 974: 0.0,
 975: 0.0,
 976: 0.0,
 977: 0.0,
 978: 0.0,
 979: 0.0,
 980: 0.0,
 981: 0.0,
 982: 0.0,
 983: 0.0,
 984: 0.0,
 985: 0.0,
 986: 0.0,
 987: 0.0,
 988: 0.0,
 989: 0.0,
 990: 0.0,
 991: 0.0,
 992: 0.0,
 993: 0.0,
 994: 0.0,
 995: 0.0,
 996: 0.0,
 997: 0.0,
 998: 0.0,
 ...}
```

In [ ]:

```
plt.title("Distribution of Betweenness Centrality for the network")
plt.xlabel("Betweenness Centrality")
plt.ylabel("Frequency")
plt.hist(list(c4.values()),bins=20)
```

Out[ ]:

```
(array([2.872e+03, 2.000e+00, 4.000e+00, 1.000e+00, 4.000e+00, 0.000e+00,
        1.000e+00, 0.000e+00, 1.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 1.000e+00, 1.000e+00, 0.000e+00,
        0.000e+00, 1.000e+00]),
 array([0.        , 0.02748533, 0.05497065, 0.08245598, 0.10994131,
        0.13742664, 0.16491196, 0.19239729, 0.21988262, 0.24736795,
        0.27485327, 0.3023386 , 0.32982393, 0.35730925, 0.38479458,
        0.41227991, 0.43976524, 0.46725056, 0.49473589, 0.52222122,
        0.54970654]),
 <a list of 20 Patch objects>)
```



- **max(c4.values()) was used to determine the highest Betweenness Centrality. The node which corresponds to this Betweenness Centrality was obtained from the c4 dictionary. The highest Betweenness Centrality calculated was that of node 603 which is 0.5497. Therefore, removing this particular node from the network will significantly affect the function of the network.**

In [ ]:

```
max(c4.values())
```

Out[ ]:

```
0.5497065448918781
```

In [ ]:

```
for k in c4.keys():
    if c4[k] > 0.5497 or c4[k] == 0.5497:
        print(k)
```

```
603
```

- **The Assortatiivity of the network was calculated using nx.degree_assortativity_coefficient(G). The Assortativity of the network was calculated to be -0.6682 which implies that connected nodes tend to possess different properties. Therefore, in this network, the nodes do not strongly tend to to interact with nodes with similar properties.**

In [ ]:

```
c5 = nx.degree_assortativity_coefficient(G)
c5
```

Out[ ]:

```
-0.6682140067239861
```