Heli Hyttinen

Tietomassan analysointi ja visualisointi Python

2021

Kaakkois-Suomen ammattikorkeakoulu

SISÄLLYS

1 FREKVENSSITAULU JA TUNNUSLUKUJA

Muuttujan sijoitus arvoiksi on annettu seuraavat:
a, a, a, a, a, b, b, b, c, c, a, a, d, d, a, c, c, e, e, e, f, a, a, c, c, b, b. Jossa a tarkoittaa sijoitusta 1, b sijoitusta 2, c sijoitusta 3, jne.

Muodostan frekvenssitaulun (taulukko 1) sijoittamalla taulun ensimmäiseen sarakkeeseen kaikki muuttujan sijoitus arvot pienimmästä suurimpaan eli a – f, jos arvo a vastaa lukua 1 ja arvo f vastaa lukua 6. Toiseen sarakkeeseen lasketaan vastaavan arvon frekvenssi, eli kuinka monta kertaa kyseinen arvo esiintyy muuttujan arvona. Esimerkiksi arvo f esiintyy vain kerran, joten sen frekvenssi on 1. Kolmanteen sarakkeeseen lasken summafrekvenssin, joka kuvaa arvojen kertymää lukumäärillä. Viimeisen arvon f kohdalla näkyy arvojen yhteismäärä, eli 27. Suhteellista frekvenssiä en tähän laske, sillä arvoja on niin vähän.

Sijoitusarvo	**Frekvenssi**	**Summa-frekvenssi**
a	10	10
b	5	15
c	6	21
d	2	23
e	3	26
f	1	27

Taulukko 1. Frekvenssitaulu.

Moodi on muuttujan yleisin arvo, eli se jonka frekvenssi on suurin. Tässä tapauksessa moodi on arvo a, sen frekvenssin ollessa 10, joka on suurin verrattuna muiden arvojen frekvensseihin.

Mediaani on suuruusjärjestetyn listan keskimmäinen arvo. Arvoja on tässä tapauksessa parillinen määrä, joten mediaaneja ovat kaksi keskimmäistä arvoa eli c ja d.
a, b, c, d, e, f.

Mediaanin voisi myös laskea muuttamalla arvot a – f luvuiksi 1–6 ja laskea keskimmäisten arvojen 3 ja 4 keskiarvo eli 3,5. (3+4) /2 = 3,5. Mutta kyseisessä tapauksessa arvot ovat sijoituksia, eikä sijoitus voi olla 3,5, vaan sijoitus on joko 3 tai 4. Sijoitus 3 esiintyy arvoissa useimmin kuin sijoitus 4, joten mediaani voisi olla 3. Lisäksi, jos yhtä sijoitusarvoa f ei olisi lainkaan, mediaani olisi selkeästi 3.

Kvartiilivälin selvittämiseksi tulee ensin selvittää ylä- ja alakvartiilit. Alakvartiili on sillä paikalla oleva arvo, jonka alapuolella on 25 % arvoista ja yläkvartiili sillä paikalla oleva arvo, jonka alapuolella on 75 % arvoista.

Jakamalla arvot 1, 2, 3, 4, 5, 6 kahteen osaan, toiseen osaan jää arvot 1, 2, 3 ja toiseen 4, 5, 6. Alakvartiili on 2, sen ollessa ensimmäisen osan mediaani. Yläkvartiili on 5, sen ollessa toisen osan mediaani. Arvojen lukumäärä on 6 ja 25 % kuudesta on 1,5 sekä 75 % kuudesta on 4,5.

Kvartiiliväli on ala- ja yläkvartiilin muodostama väli eli [2, 5]. Kvartiilivälin pituus on ylä- ja alakvartiilin erotus eli 3.

1.1 Frekvenssitaulu ja tunnuslukuja

On annettu seuraava reaalilukujen joukko: 4, 1, 2, 5, 6, 10, 2, 15, 16, 20. Lukuja on yhteensä 10.

Lasken arvojen aritmeettisen keskiarvon $\bar{x}$ laskemalla arvot yhteen ja sen jälkeen jakamalla saadun summan arvojen lukumäärällä 10.

$\bar{x} = \frac{4+1+2+5+6+10+2+15+16+20}{10} = \frac{81}{10}$ = 8.1. Eli $\bar{x}$ = 8,1.

Seuraavaksi lasken keskihajonnan saadun keskiarvon avulla. Aloitan laskemisen laskemalla jokaisen arvon poikkeaman keskiarvosta, eli $x_i - \bar{x}$.

$4 - 8{,}1 = -4{,}1$

$1 - 8{,}1 = -7{,}1$

$2 - 8{,}1 = -6{,}1$

$5 - 8{,}1 = -3{,}1$

$6 - 8{,}1 = -2{,}1$

$10 - 8{,}1 = 1{,}9$

$2 - 8{,}1 = -6{,}1$

$15 - 8{,}1 = 6{,}9$

$16 - 8{,}1 = 7{,}9$

$20 - 8{,}1 = 11{,}9$

Seuraavaksi lasken poikkeamien neliöt ja niiden keskiarvon eli varianssin.

$-(4{,}1)^2 = 16{,}81$

$-(7{,}1)^2 = 50{,}41$

$-(6{,}1)^2 = 37{,}21$

$-(3{,}1)^2 = 9{,}61$

$-(2{,}1)^2 = 4{,}41$

$1{,}9^2 = 3{,}61$

$-(6{,}1)^2 = 37{,}21$

$6{,}9^2 = 47{,}61$

$7{,}9^2 = 62{,}41$

$11{,}9^2 = 141{,}61$

Varianssi saadaan laskemalla saadut tulokset yhteen ja jakamalla luvulla 10.

$$\frac{16{,}81 + 50{,}41 + 37{,}21 + 9{,}61 + 4{,}41 + 3{,}61 + 37{,}21 + 47{,}61 + 62{,}41 + 141{,}61}{10}$$

= $\frac{410{,}90}{10}$ = 41,09. Varianssista saadaan laskettua keskihajonta ottamalla luvusta 41,09 neliöjuuri. $\sigma_x = \sqrt{41{,}09} \approx 6$. Eli keskihajonta σ_x on 6.

2 FREKVENSSITAULU JA MOODI

Kuvassa 1, sivu 7, määrään frekvenssitaulun sekä moodin annetusta havaintoaineistosta Python-ohjelmointikielellä.

```
In [1]: import numpy as np

        #koko autoaineisto listana
        hh_data = ["Volkkari", "Volkkari", "Mersu", "Viiatti", "Tojota",
                   "Sitikka", "Sitikka", "Sitikka", "Volkkari", "Viiatti",
                   "Viiatti", "Mersu", "Poso", "Poso", "Tojota",
                   "Poso", "Mersu", "Poso", "Viiatti", "Tatsun",
                   "Tojota", "Tatsun", "Viiatti", "Volkkari", "Tojota",
                   "Tojota", "Volkkari"]

        #listan eri alkiot joukkona hh_autot, aakkosjärjestyksessä
        hh_autot = sorted(set(hh_data))
        hh_autot

Out[1]: ['Mersu', 'Poso', 'Sitikka', 'Tatsun', 'Tojota', 'Viiatti', 'Volkkari']

In [2]: #frekvenssitaulu
        #listan hh_frekv sisällä for-loop,
        #löytää frekvenssit eli montako jokaista autoa (hh_autot) on aineistossa (hh_data)
        hh_frekv = [hh_data.count(x) for x in hh_autot]
        hh_frekv

Out[2]: [3, 4, 3, 2, 5, 5, 5]

In [3]: #etsin listan hh_frekv suurimmat luvut
        #moodeja on kolme, joten käytän argwhere ja amax, jotta kaikki moodien paikat löytyvät

        hh_ind = np.argwhere(hh_frekv == np.amax(hh_frekv))

        #laitan löydetyt paikka-arvot saman listan sisälle
        hh_ind = hh_ind.flatten().tolist()
        hh_ind

Out[3]: [4, 5, 6]

In [5]: #moodien paikat ovat 4, 5 ja 6 listassa hh_autot
        #teen moodi listan ja käyn läpi listan hh_ind jossa on moodien paikat,
        #ja lisään vastaavat autojen nimet moodilistaan
        hh_moodi = []
        for i in range(len(hh_ind)):
            hh_moodi.append(hh_autot[hh_ind[i]])

        hh_moodi

Out[5]: ['Tojota', 'Viiatti', 'Volkkari']
```

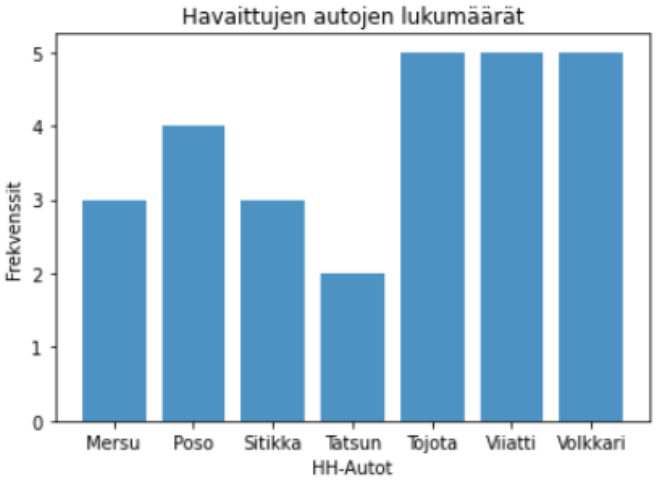
Kuva 1. Frekvenssitaulu ja moodi, kuvakaappaus.

Koodissa on kaksi listaa, toisessa koko aineisto ja toisessa kaikki aineiston eri arvot. Koodi laskee montako jokaista arvoa koko aineistossa on ja näin muodostuu frekvenssitaulu, josta koodi etsii suurimmat frekvenssit ja niiden paikka-arvot. Paikka-arvojen avulla koodi löytää moodit listasta, jossa on alkuperäisen aineiston eri arvot. Moodeja on kolme tässä aineistossa.

2.1 Pylväs- ja piirakkadiagrammit

Kuvassa 2, sivu 8, havainnollistan frekvenssejä pylväsdiagrammin avulla.

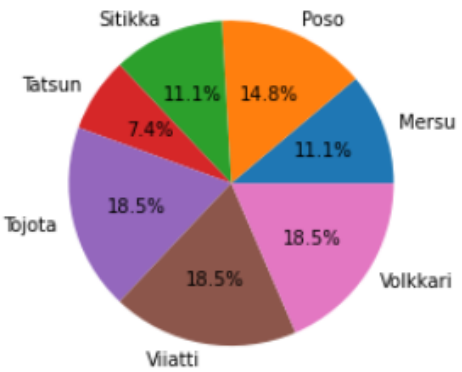
In [6]:
```
import matplotlib.pyplot as plt

#havainnollistan frekvenssit pylväsdiagrammilla
plt.bar(hh_autot,hh_frekv,align = 'center', alpha = 0.8)
plt.xticks(hh_autot)
plt.xlabel("HH-Autot")
plt.ylabel('Frekvenssit')
plt.title("Havaittujen autojen lukumäärät")
plt.savefig("t_3pylvas.jpg")
plt.show()
```



Kuva 2. Pylväsdiagrammi.

Kuvassa 3 havainnollistan frekvenssejä piirakkadiagrammin avulla.

In [7]:
```
#havainnollistan frekvenssit piirakkadiagrammilla
plt.pie(hh_frekv,labels = hh_autot,autopct = '%1.1f%%')
plt.savefig("t_3piirakka.jpg")
plt.show()
```



Kuva 3. Piirakkadiagrammi.

Kokeilin skriptiä muuttamalla autoaineiston arvoja, toimii.

3 TEKSTIN ARVIOINTI PYTHONILLA

Kuvassa 4 ja 5 luen annetun tekstitiedoston, parsin tekstiä, typistän sanoja ja lasken negatiivisten sekä positiivisten sanojen määrät annetusta tekstistä Python-ohjelmointikielellä.

In [1]:
```
import pandas as pd

#luetaan tekstitiedosto Arvio.txt
hh_arvio_txt = pd.read_csv('Arvio.txt',delimiter = '\n',header = None, keep_default_na = False, encoding = 'U

#alkuperäisen tiedoston yksi rivi muodostaa listan
#"Series" parsii listan ja jakaa listan merkkijonon sanat erilleen omiksi merkkijonoikseen splitin avulla
#muodostuu kaksiuloitteinen lista, merkkijonot listojen sisällä ja listat listan sisällä
hh_arvio_list = [pd.Series(hh_arvio_txt.values[i]).str.split() for i in range(len(hh_arvio_txt.values))]

#otsikko on listan ensimmäinen
hh_otsikko_list = hh_arvio_list[0][0]
#otsikko-lista yhdeksi merkkijonoksi "join" avulla
hh_otsikko = ' '.join([str(i) for i in hh_otsikko_list])
#poistetaan vielä ":" otsikon nimestä "strip" avulla
hh_otsikko = hh_otsikko.strip(':')
print(hh_otsikko)

#sanojen analysoimiseksi,
#kaikki sanat samaan listaan käymällä kaikki rivit läpi ja lisäämällä merkkijonot listaan pienin kirjaimin
hh_apuarray = []
for i in range(len(hh_arvio_list)):
    hh_apuarray = hh_apuarray + hh_arvio_list[i][0]
hh_sanat = list(pd.Series(hh_apuarray).str.lower())

#poistetaan merkit sanojen lopusta, eli . ja ,
hh_csanat = [hh_sanat[i].strip('.') for i in range(len(hh_sanat))]
hh_csanat = [hh_csanat[i].strip(',') for i in range(len(hh_sanat))]
```

Kuva 4. Tekstin arviointi, osa 1/3.

Annettu tekstitiedosto luetaan pandas-kirjaston avulla, tekstitiedostossa yksi rivi muodostaa listan. Koodi parsii listan merkkijonon sanat erilleen omiksi merkkijonoikseen ja muodostaa yhden kaksiulotteisen listan, jossa merkkijonot ovat listojen sisällä ja listat listan sisällä. Listan ensimmäinen lista on tekstitiedoston otsikko. Koodi muodostaa otsikon listan yhdeksi merkkijonoksi ja poistaa siitä kaksoispisteen ”:”, sekä tulostaa otsikon.

Koodi käy läpi kaikki listat, eli tekstirivit tekstitiedostosta, muuttaa jokaisen merkkijonon kirjaimet pieniksi kirjaimiksi ja lisää merkkijonot uuteen listaan. Näin saadaan yksi lista, jossa kaikki alkuperäisen tekstin sanat ovat omina merkkijonoinaan pienin kirjaimin. Tämän jälkeen koodi poistaa mahdolliset pisteet ja pilkut sanojen lopusta.

```
#sanojen typistämiseksi, tuodaan snowballin FinnishStemmer
from nltk.stem.snowball import FinnishStemmer
st = FinnishStemmer()

#typistetään sanat uuden listan sisälle, eli muutetaan mahdollisimman perusmuotoon
hh_st_csanat = [st.stem(i) for i in hh_csanat]

#etsittävät sanat typistetään, jotta ne ovat saman muotoisia kuin tekstissä,
#sanojen taivutukset vaikuttavat erityisesti typistämiseen
#esim. taitava on monikossa taitavat, muoto ei juuri muutu
#esim. kaunis on monikossa kauniit, muoto muuttuu merkittävästi stemmauksen kannalta

#etsittävät positiiviset ja negatiiviset sanat, taivutettuna eri tavoin
hh_pos = ["mielenkiintoinen", "mielenkiintoiset", "kaunis", "kauniit", "kauniisti", "erinomainen",
          "erinomaiset", "loistava", "herkka", "taitava"]
hh_neg = ["tylsa", "ruma", "huono", "kylma", "tasapaksu", "taitamaton", "taitamattomat"]

#etsittävien sanojen typistys
hh_st_pos = [st.stem(i) for i in hh_pos]
hh_st_neg = [st.stem(i) for i in hh_neg]

#lasketaan positiivien sanojen määrä arvioitavasta tekstistä
#arvioitava teksti sekä etsittävät sanat eri muodoissa ovat typistetty
a = 0
for i in range(len(hh_st_pos)):
    if hh_st_pos[i] in hh_st_csanat:
        a = a+1

#lasketaan negatiivisten sanojen määrä samalla tavalla
b = 0
for i in range(len(hh_st_neg)):
    if hh_st_neg[i] in hh_st_csanat:
        b = b+1
```

Kuva 5. Tekstin arviointi, osa 2/3.

Snowball-stemmerin avulla typistetään tekstin sanoja mahdollisimman perusmuotoisiksi. Lisätään koodiin listat positiivista ja negatiivisista sanoista, joita on tarkoitus etsiä tekstistä. Positiiviset ja negatiiviset sanat ovat taivutettu listaan eri muodoissa, jotta niiden typistämisen jälkeen sanat löytyvät tekstistä todennäköisemmin. Koodi typistää nämä etsittävät sanat ja käy läpi arvioitavan tekstin sekä typistetyt positiiviset sanat ja typistetyt negatiiviset sanat, laskee näiden määrät arvioitavassa tekstissä ja tallentaa määrät muuttujiin a ja b.

Kuvassa 6 koodi arvioi laskettujen positiivisten ja negatiivisten sanojen määrien perusteella onko tekstiarvio positiivinen vai negatiivinen sekä näyttää tulosteet.

```
#jos b jaettuna a:lla on enemmän kuin kaksi, arvio elokuvasta on positiivinen
ab = a / b
if ab > 2:
    print('Tämä arvio em. elokuvasta on positiivinen')
else:
    print('Tämä arvio em. elokuvasta ei ole positiivinen')
```

```
Stan & Ollie
Tämä arvio em. elokuvasta on positiivinen
```

Kuva 6. Tekstin arviointi, osa 3/3.

Etsittävät sanat ovat samat mitä tehtävänannossa on kuvattu. Kokeilin skriptiä vielä muuttamalla arvioitavaa tekstiä, mutta käyttäen samoja positiivisia ja negatiivisia sanoja.

4 HISTOGRAMMI JA MOODILUOKKA EXCEL-TIEDOSTOSTA

Kuvassa 7, sivu 11, luen annetun Excel-tiedoston, luokittelen kyseisen tiedoston A-sarakkeen arvot viiteen luokkaan tasavälisesti, selvitän frekvenssit ja moodiluokan ja havainnollistan A-sarakkeen tiedot histogrammin avulla.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

#luetaan excel-tiedosto Loki.
#sarakkeen A muuttujat ovat tiedoston kokoja
hh_loki = pd.read_excel('Loki.xlsx', header = None)
hh_A = hh_loki.values.T[0]

#luokitellaan muuttujan A arvot viiteen luokkaan tasavälisesti
#eli A:n arvoista suurin luku jaettuna viidellä olisi ensimmäisen luokkaraja yläraja
#8390/5=1678, arvot jotka esiintyvät välillä 0-1678 (1678 ei sisälly) kuuluvat ensimmäiseen luokkarajaan
#1678 lisättynä 1678 on seuraava luokkaraja eli 3356. Seuraavaan luokkarajaan kuuluu 1678 ja 3356 väli,
#mutta 3356 ei sisälly rajaan, 1678 sisältyy.

#hh_bins kertoo luokkarajat
#hh_freqs kertoo luokkarajojen frekvenssit, kuinka monta arvoa luokkarajassa esiintyy
hh_freqs, hh_bins = np.histogram(hh_A, 5)

#Luodaan histogrammi
plt.title('Histogrammi ladattavien tiedostojen koosta')
plt.xlabel('Tiedoston koko (mt)')
plt.ylabel('Frekvenssit')
plt.hist(hh_A,hh_bins)
plt.show()

#Selvitetään moodiluokka eli luokka jonka frekvenssi on suurin
hh_moodi = np.argmax(hh_freqs)
#Moodiluokan rajat
print('Moodiluokan alaraja on ', hh_bins[hh_moodi])
print('Moodiluokan yläraja on ', hh_bins[hh_moodi+1])
```


```
Moodiluokan alaraja on  0.0
Moodiluokan yläraja on  1678.0
```

Kuva 7. Histogrammi ja moodiluokka Excel-tiedostosta.

Koodi lukee annetun Excel-tiedoston ja tallentaa tiedoston ensimmäisen sarakkeen muuttujaan A, josta numpy-kirjaston histogrammiominaisuuden avulla koodi jakaa A-muuttujan arvot viiteen luokkaan tasavälisesti ja selvittää luokkien frekvenssit.

Viiteen luokkaan jako tapahtuu käytännössä jakamalla A:n arvoista suurin luku viidellä, jonka tulo olisi ensimmäisen luokan yläraja. Eli tässä tapauksessa A-muuttujan suurin arvo on 8390, joka jaetaan viidellä ja saadaan ensimmäisen luokan ylärajaksi 1678. Arvot, jotka esiintyvät välillä 0-1678, mutta ei 1678, kuuluvat ensimmäiseen luokkaan. Seuraavaan luokkarajaan kuuluu 1678 ja 3356 väli ja 1678, mutta ei 3356. Ja niin edelleen. Viimeiseen luokkarajaan sisältyy molemmat raja-arvot, eli myös 8390.

Koodissa histogrammi luodaan matplotlib.pyplot-kirjaston avulla ja moodiluokka selvitetään numpy-kirjaston avulla etsimällä suurin frekvenssi ja sitä vastaavat luokkarajat. Kokeilin koodin toimivuutta vielä erilaisilla arvoilla.

5 TIEDOSTOKOON LUOKAN SELVITYS JA YHTEYSAJAT

Kuvassa 8 ja 9 luen annetun Excel-tiedoston, joka on sama kuin aiemmassa tehtävässä, selvitän mihin aiemmassa tehtävässä määrättyyn luokkaan tiedosto kooltaan 5,5 GB kuuluu. Lisäksi poimin selvitetyn luokan yhteysajat, joista lasken keskiarvon ja keskihajonnan, joiden perusteella arvioin kannattaako tiedostoa ladata annetussa ajassa 8 minuuttia.

In [1]:

```
import pandas as pd
import numpy as np

hh_loki = pd.read_excel('Loki.xlsx', header = None)
hh_A = hh_loki.values.T[0]
hh_freqs, hh_bins = np.histogram(hh_A, 5)

#aika minuutteina, t
hh_t = 8
#muutetaan aika sekunneiksi, jotta voidaan verrata tiedoston Loki.xlsx aikoihin, -
#sillä ne ovat esitetty sekunteina
hh_t = hh_t * 60

#tiedoston koko gigatavuina
hh_gb = 5.5
#tiedoston koko megatavuina, jotta vastaavat koot löytyvät Loki-tiedostosta
hh_mb = hh_gb * 1024

#tarkistetaan funktiolla mihin edellisen tehtävän luokkaan tiedostokoko kuuluu
#funktio palauttaa luokan rajat
hh_apuarray1 = []
def luokanTarkistus(hh_mb):
    for i in range(len(hh_bins)):
        #jos sama tai suurempi kuin ensimmäisen luokan alaraja ja -
        #pienempi kuin ensimmäisen luokan yläraja, palautetaan ensimmäisen luokan luokkarajat
        if hh_mb >= hh_bins[0] and hh_mb < hh_bins[1]:
            hh_apuarray1 = [hh_bins[0], hh_bins[1]]
            return hh_apuarray1
        #jos sama tai suurempi kuin toisen luokan alaraja ja -
        #pienempi kuin toisen luokan yläraja, palautetaan toisen luokan luokkarajat. jne.
        #viimeiseen luokkaan kuuluu myös sen luokan ylärajan arvo
        if hh_mb >= hh_bins[1] and hh_mb < hh_bins[2]:
            hh_apuarray1 = [hh_bins[1], hh_bins[2]]
            return hh_apuarray1
        if hh_mb >= hh_bins[2] and hh_mb < hh_bins[3]:
            hh_apuarray1 = [hh_bins[2], hh_bins[3]]
            return hh_apuarray1
        if hh_mb >= hh_bins[3] and hh_mb < hh_bins[4]:
            hh_apuarray1 = [hh_bins[3], hh_bins[4]]
            return hh_apuarray1
        if hh_mb >= hh_bins[4] and hh_mb <= hh_bins[5]:
            hh_apuarray1 = [hh_bins[4], hh_bins[5]]
            return hh_apuarray1

#funktion kutsu
hh_luokka = luokanTarkistus(hh_mb)

#kun luokka on saatu selville, poimitaan luokkaa vastaavat yhteysajat-
#käymällä läpi koko Loki-tiedosto
hh_apuarray2 = []
for i in range(len(hh_loki.values)):
    #käydään läpi lokitiedoston a sarakkeen transpoosi, eli tiedostojen koot
    #poimitaan luokkaan kuuluvien tiedostokokojen lataamiseen käytetyt yhteysajat -
    #eli jos transpoosin arvo on isompi tai sama kuin luokan alaraja -
    #ja pienempi kuin luokan yläraja -
    #samalla paikalla oleva yhteysaika sarakkeen b transpoosista poimitaan listaan
    if hh_loki.values.T[0][i] >= hh_luokka[0] and hh_loki.values.T[0][i] < hh_luokka[1]:
        #sarakkeen b transpoosi, eli lataamisen käytetyt yhteysajat
        hh_apuarray2 = hh_apuarray2 + [hh_loki.values.T[1][i]]
    hh_yhteysajat = hh_apuarray2
```

Kuva 8. Tiedostokoon luokan selvitys ja yhteysajat, osa 1/2.

Koodi käyttää aiemman tehtävän luokkia. Koodi saa ajan minuutteina ja tiedoston koon gigatavuina. Minuutit muutetaan sekunneiksi ja gigatavut muutetaan megatavuiksi, jotta vastaava aika ja tiedoston koko löytyy analysoitavasta Excel-tiedostosta.

Koodi tarkastaa funktiolla mihin luokkaan annettu tiedoston koko kuuluu, funktio palauttaa luokan rajat muuttujaan luokka. Funktio saa tiedoston koon ja if-lausekkeella selvittää vastaavan luokan. Jos tiedoston koko on sama tai suurempi kuin ensimmäisen luokan alaraja ja pienempi kuin ensimmäisen luokan yläraja, funktio palauttaa ensimmäisen luokan luokkarajat. Jos tiedoston koko ei sovi tähän ehtoon, funktio selvittää onko tiedoston koko sama tai suurempi kuin toisen luokan alaraja ja pienempi kuin toisen luokan yläraja ja niin edelleen.

Kun tiedoston koon luokka on saatu selville, poimitaan luokkaa vastaavat yhteysajat listaan käymällä läpi koko Excel-tiedosto. Koodi käy läpi Excel-tiedoston ensimmäisen sarakkeen transpoosin, eli tiedostojen koot ja poimii aiemmin selvittyä luokkaa vastaavat arvot Excel-tiedoston toisen sarakkeen transpoosista, eli tiedostokokojen lataamiseen käytetyt yhteysajat, ja tallentaa yhteysajat listaan.

```
#lasketaan yhteysaikojen keskiarvo käymällä poimitut yhteysajat läpi ja -
#laskemalla ajat yhteen sekä laskemalla niiden yhteismäärän ja -
#jakamalla saatu summa poimittujen yhteysaikojen määrällä
hh_summa = 0
hh_maara = 0
for i in range(len(hh_yhteysajat)):
    hh_summa += hh_yhteysajat[i]
    hh_maara += 1
hh_keskiarvo = hh_summa / hh_maara

#lasketaan keskihajonta numpyn avulla
#keskiarvon olisi tietysti myös voinut laskea np.mean(hh_yhteysajat),
#mutta tein hienon loopin niin antaa mennä sillä.
hh_keskihajonta = np.std(hh_yhteysajat)

#tarkistetaan onko annettu aika (sekunteina) suurempi kuin keskiarvon ja keskihajonnan summa
if hh_t > hh_keskiarvo + hh_keskihajonta:
    print("Ehkäpä kerkiät spruuttaamaan tiedoston")
else:
    print("Ei kannata näin kiireessä")
```

```
Ehkäpä kerkiät spruuttaamaan tiedoston
```

Kuva 9. Tiedostokoon luokan selvitys ja yhteysajat, osa 2/2.

Koodi laskee poimittujen yhteysaikojen keskiarvon käymällä yhteysajat läpi, laskemalla arvojen yhteismäärän ja laskemalla arvot yhteen, jonka jälkeen saatu summa jaetaan poimittujen yhteysaikojen yhteismäärällä. Keskihajonta lasketaan keskiarvosta numpy-kirjaston avulla. Koodi tarkistaa onko annettu aika sekunteina suurempi kuin keskiarvon ja keskihajonnan summa ja tulostaa vastauksen.

Kokeilin koodia antamalla alussa eri minuuttimäärän ja eri tiedostokoon. Toimii kokoon 8.19 GB saakka, eli 8390 mb jaettuna 1024 mb, mutta ei kai tarvinnutkaan toimia Loki-tiedostossa annettuja tiedostokokoja suuremmille tiedostoille.

6 GRAAFIN G SOLMUJEN ASTELUVUT, K-KLIKIT, KESKEISIN SOLMU JA GRAAFI ETÄISYYSMATRIISISTA

Kuvassa 10 luen annetun Excel-tiedoston, joka sisältää viereisyysmatriisin, muodostan matriisista graafin G, jonka solmujen asteluvut selvitän. Etsin graafin k-klikit, kun $k \geq 4$ ja lasken graafin G etäisyysmatriisin NetworkX-kirjaston avulla, josta selvitän solmujen suhteelliset läheisyysluvut kuvassa 11 ja erityisesti selvitän minkä solmun suhteellisin läheisyysluku on suurin. Tulosteet näkyvät kuvassa 11.

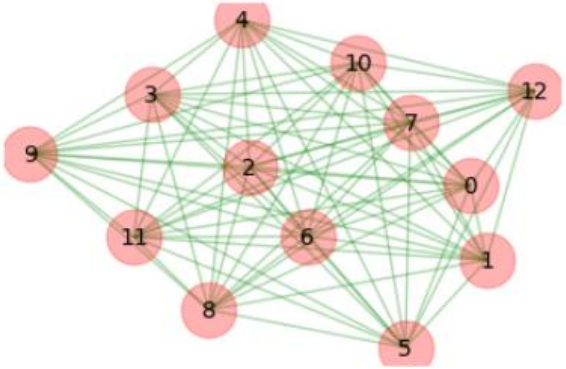
In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd

#luetaan excel-tiedosto Yhteys
hh_yhteys = pd.read_excel('Yhteys.xlsx', header = None)
#viereisyysmatriisi, sama transpoosinsa kanssa
hh_yhteys = hh_yhteys.values

#graafi G
hh_G = nx.from_numpy_matrix(hh_yhteys)

#tulostetaan solmujen asteluvut
hh_asteluvut = hh_G.degree
for i in range(len(hh_asteluvut)):
    print("Solmun ", i, " asteluku on ", hh_asteluvut[i])

#etsitään kaikki klikit networkx avulla
hh_klikit = list(nx.enumerate_all_cliques(hh_G))
#etsitään klikit, jossa on neljä tai enemmän solmua
hh_4klikit = [k for k in hh_klikit if len(k) >= 4]
print("\nk-klikit graafissa G, kun k >= 4: ", hh_4klikit,"\n")

#etäisyysmatriisi networkx avulla
hh_dgmatrix = nx.algorithms.shortest_paths.dense.floyd_warshall_numpy(hh_G, nodelist=None, weight='weight')

#suhteellinen läheisyysluku solmulle selvitetään jakamalla solmujen yhteismäärä-1 (eli 12) -
#solmun lyhyimpien polkujen pituuksien summalla
#graafin keskeisin solmu on se, jolla on suurin suhteellinen läheisyysluku
```

Kuva 10. Graafin G solmujen asteluvut, k-klikit ja etäisyysmatriisi.

Koodi muodostaa networkx-kirjaston avulla graafin ja tulostaa graafin asteluvut. Koodi etsii networkx-kirjaston avulla kaikki graafin klikit, jonka jälkeen etsii ja tulostaa klikit, joissa on neljä tai enemmän solmua. Tulosteet näkyvät kuvassa 11. Tämän jälkeen koodi muodostaa networkx-kirjaston avulla etäisyysmatriisin graafista G. Seuraavaksi selvitetään graafin G keskeisin solmu seuraavassa kuvassa (kuva 11).

```
#lasketaan etäisyysmatriisin avulla jokaisen solmun lyhyimpien polkujen pituudet yhteen
#käydään siis läpi etäisyysmatriisi ja lisätään summat listaan hh_lyhyetpolut_yht
hh_lyhyetpolut_yht = []
for i in range(len(hh_dgmatrix)):
    hh_lyhyetpolut_yht += [sum(hh_dgmatrix[i])]
#jaetaan G:n solmujen yhteismäärä-1 (13-1) aiemmin saaduilla summilla ja lisätään -
#tulos listaan hh_closeness, eli käydään hh_lyhyetpolut_yht-lista läpi for-loopilla
hh_G_solmut = len(hh_G)
hh_closeness = []
for i in range(len(hh_lyhyetpolut_yht)):
    hh_closeness += [(hh_G_solmut-1)/hh_lyhyetpolut_yht[i]]
#hh_closeness sisältää nyt G:n solmujen suhteelliset läheisyysluvut,
#niistä suurin on keskeisin solmu graafissa

hh_keskeisin = max(hh_closeness)
#käydään läpi for-loopilla hh_closeness ja lasketaan monesko arvoista on isoin
def MoneskoSolmu(hh_keskeisin):
    for i in range(len(hh_closeness)):
        if(hh_closeness[i]==hh_keskeisin):
            return i
hh_keskeisin_solmu = MoneskoSolmu(hh_keskeisin)
print("Graafissa G keskeisin solmu on ", hh_keskeisin_solmu)

#graafi etäisyysmatriisista
hh_dg = nx.from_numpy_matrix(hh_dgmatrix)
#positiot linkeille ja solmuille piirtoa varten
hh_pos = nx.spring_layout(hh_dg)
#piirretään solmut
nx.draw_networkx_nodes(hh_dg,hh_pos,node_color='red',node_size=1100,alpha=0.3)
#piirretään linkit
nx.draw_networkx_edges(hh_dg,hh_pos,width=1.5,edge_color='green',alpha=0.3)
#nimet solmuille
hh_labels = {0:0,1:1,2:2,3:3,4:4,5:5,6:6,7:7,8:8,9:9,10:10,11:11,12:12}
nx.draw_networkx_labels(hh_dg,hh_pos,hh_labels,font_size=14)
#kehys pois ja piirretään graafi etäisyysmatriisista
plt.axis('off')
plt.show()
```

```
Solmun  0  asteluku on  3
Solmun  1  asteluku on  2
Solmun  2  asteluku on  1
Solmun  3  asteluku on  1
Solmun  4  asteluku on  3
Solmun  5  asteluku on  5
Solmun  6  asteluku on  2
Solmun  7  asteluku on  4
Solmun  8  asteluku on  4
Solmun  9  asteluku on  4
Solmun  10  asteluku on  3
Solmun  11  asteluku on  2
Solmun  12  asteluku on  2

k-klikit graafissa G, kun k >= 4:  [[0, 7, 8, 10]]

Graafissa G keskeisin solmu on  5
```



Kuva 11. Graafin G solmujen asteluvut, k-klikit, keskeisin solmu ja graafi etäisyysmatriisista.

Suhteellinen läheisyysluku solmulle selvitetään jakamalla solmujen yhteismäärä vähennettynä yhdellä, eli tässä tapauksessa 12, solmun lyhyimpien polkujen pituuksien summalla. Ensin lasketaan etäisyysmatriisin avulla jokaisen solmun lyhyimpien polkujen pituudet yhteen. Koodi käy etäisyysmatriisin läpi ja lisää uuteen listaan jokaisen solmun lyhyimpien polkujen pituuksien summan. Seuraavaksi koodi luo taas uuden tyhjän listan ja lisää sinne jokaisen solmun suhteellisen läheisyysluvun käymällä läpi äsken luodun listan lyhyimpien polkujen pituuksien summista ja jakamalla luvun 12 solmua vastaavalla summalla. Kun koodi on selvittänyt kaikkien solmujen suhteellisen läheisyysluvun, ne löytyvät yhdestä listasta. Tämän listan suurin luku on graafin G keskeisimmän solmun ja sen arvo tallennetaan muuttujaan.

Koodi selvittää funktiolla suurinta arvoa vastaavan solmun käymällä läpi suhteellisten läheisyyslukujen listan ja vertaamalla listan arvoja äsken luotuun muuttujaan, eli muuttuja, jolle on tallennettu suurin suhteellinen läheisyysluku. Funktio palauttaa sen arvon paikka-arvon, joka vastaa muuttujaa suurimmalla suhteellisella läheisyysluvulla. Funktion palauttama paikka-arvo on sama kuin graafin G solmu, sillä solmujen nimet ovat koodissa numeroita 0-12. Koodi tulostaa graafin G:n keskeisimmän solmun (kuva 11). Lopuksi koodi muodostaa etäisyysmatriisista graafin (kuva 11). Kokeilin koodia myös muuttamalla viereisyysmatriisia.