Joni Hämäläinen

MERKKIJONOJEN VERTAILU SUMEAA LOGIIKKAA HYÖDYNTÄEN

Opinnäytetyö

Liiketalouden ammattikorkeakoulututkinto

Tietojenkäsittelyn koulutus

2023

XAMK
Kaakkois-Suomen
ammattikorkeakoulu

Tutkintonimike	Tradenomi (amk)
Tekijä/Tekijät	Joni Hämäläinen
Työn nimi	Merkkijonojen vertailu sumeaa logiikkaa hyödyntäen
Toimeksiantaja	Xamk Datalab Kouvola, Dataamo-hanke.
Vuosi	2023
Sivut	28 sivua
Työn ohjaaja	Janne Turunen

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli toteuttaa Python-ohjelma, joka vertailee kahden aineiston merkkijonoja keskenään ja luoda yhtäläisyyksien pohjalta uusi pandas-taulukko. Työ toteutettiin toimeksiantona, jonka antoi XAMK Datalab Kouvola, Dataamo-hanke.

Opinnäytetyössä tutkitaan ohjelmassa esiintyviä tekniikoita ja käsitteitä. Työssä keskitytään erityisesti sumeaan logiikkaan ja Levenšteinin etäisyyteen, joiden pohjalta on rakennettu työssä käytettävä FuzzyWuzzy-kirjasto, jonka avulla merkkijonojen vertailu toteutetaan. Teoriaosuudessa käsitellään myös pandas, jonka DataFrame-objektina ohjelman tulokset palautetaan.

Käsitteiden ja tekniikoiden esittelyn jälkeen käydään läpi ohjelman ensimmäisen version toimintaperiaate, ohjelman uuden version toteutus sekä ohjelman versioiden välinen vertailu. Ohjelma jakautuu kolmeen suurempaan funktioon, joiden vaiheet käydään tarkemmin läpi. Ohjelman toteutuksen jälkeen suoritetaan vielä testejä, joilla vertaillaan ohjelman ensimmäistä ja tässä työssä toteutettua versiota ja vedetään yhteen tästä saadut tulokset.

Ohjelman tavoitteena on vertailla patenttihakemuksien tekijöitä ja yrityksien nimiä keskenään ja näiden merkkijonojen ollessa tarpeeksi lähellä toisiaan lisätä yhtäläisyys pandas-taulukkoon. Ohjelma myös mahdollisesti muokkaa läpikäytävää dataa haluttuun muotoon erilaisilla RegEx-toiminnoilla. Lopuksi yhtäläisyyksien pohjalta luotu taulukko voidaan lähettää esimerkiksi haluttuun tietokantaan.

Tavoitteet tässä opinnäytetyössä täyttyivät kaikilta osin. Työn lopputuloksena saatiin merkkijonoja vertaileva Python-ohjelma, joka on toiminnaltaan ja tehokkuudeltaan edeltävää versiotaan parempi. Toimeksiantaja sai käyttöönsä ohjelman, jota se voi käyttää hankkeessaan ja mahdollisesti vielä jatkokehittää tulevaisuudessa. Työn lopussa käydään läpi ajatuksia, joita projekti herätti, sekä minkälaisia valmiuksia se antaa tulevaisuuden työuraa ajatellen.

Asiasanat: sumea logiikka, ohjelmointi, Python, Levenšteinin etäisyys, FuzzyWuzzy

XAMK
South-Eastern Finland
University of Applied Sciences

Degree title	Bachelor of Business Administration
Author	Joni Hämäläinen
Thesis title	Comparing strings using fuzzy logic
Commissioned by	Xamk Datalab Kouvola, The Dataamo project
Time	2023
Pages	28 pages
Supervisor	Janne Turunen

ABSTRACT

The objective of the thesis was to implement a Python program that compared strings of two datasets with each other and created a new pandas table based on similarities. The thesis examined the techniques and concepts used in the program. In particular, the focus was on fuzzy logic and Levenshtein distance, based on which the FuzzyWuzzy library, used in the program for comparing strings, was built. The theory part also covered the pandas DataFrame object of which the program returned results.

After presenting the concepts and techniques, the first version of the program was described in terms of its operational principles, followed by the implementation of a new version and a comparison between the two. The program was divided into three major functions, the steps of which were examined in more detail. After the implementation of the program, tests were conducted to compare the first version of the program with the version developed in this thesis, and the results were summarized.

The program's objective was to compare the names of patent applicants and company names with each other, and when their strings were close enough to each other, to add the similarities to the pandas table. The program also possibly modifies the processed data into a desired format with various RegEx functions. Finally, the table created based on similarities can be sent to a chosen database.

All objectives of this thesis were met. The result was a Python program that compared strings and was more efficient and functional than its previous version. The commissioner received a program that it could use in its project and possibly further develop in the future. The discussion at the end of the thesis covered the thoughts that the project raised and the skills it provided for future career development.

Keywords: fuzzy logic, programming, Python, Levenshtein distance, FuzzyWuzzy

SISÄLLYS

1 JOHDANTO ... 5
2 OHJELMAN TEOREETTISET KÄSITTEET ... 6
2.1 Sumea logiikka ... 6
2.2 Levenšteinin etäisyys ... 7
2.3 FuzzyWuzzy ... 8
2.4 Pandas ... 10
2.5 Python ... 13
3 OHJELMAN KEHITTÄMINEN ... 14
3.1 Ohjelman lähtötilanne ... 14
3.2 Ohjelman uuden version toteutus ... 17
4 TULOSTEN VERTAILU JA YHTEENVETO ... 23
5 PÄÄTÄNTÖ ... 25
LÄHTEET ... 27

1 JOHDANTO

Tavoitteena on toteuttaa Python-ohjelma, joka vertailee kahta listaa. Toisessa listassa on patenttihakemuksia ja toisessa yritysten nimiä. Ohjelma luo löydettyjen yhtäläisyyksien perusteella pandas-taulukon, jossa yhdistetään patenttihakemukset oikeille yrityksille käyttäen FuzzyWuzzy-kirjastoa ja RegEx:iä.

Työn toimeksiantajana on Kaakkois-Suomen ammattikorkeakoulun Datalab Kouvola, Dataamo-hanke. Ohjelma tulee hankkeen käyttöön tulevaisuudessa. Ohjelmasta on jo olemassa ensimmäinen versio, joten tavoitteena on ohjelman jatkokehitys.

Opinnäytetyön tutkimusongelmana on luoda Python-ohjelma, joka vertailee kahden aineiston merkkijonoja keskenään eri tekniikoilla ja luo vertailun pohjalta uuden aineiston. Luku kaksi käsittelee työssä käytettäviä tekniikoita, työkaluja sekä käsitteitä. Keskeisimpinä käsitteinä työssä ovat sumea logiikka, Levenšteinin etäisyys sekä FuzzyWuzzy. Näiden työkalujen ja käsitteiden pohjalle aineistojen vertailu ja yhtäläisyyksien etsiminen perustuu. Lisäksi käsitellään myös pandasia, jonka DataFrame-objektina tulokset palautetaan sekä Python-ohjelmointikieltä.

Luvussa kolme käydään läpi ohjelman ensimmäisen version toimintaperiaate ja millainen on lähtötilanne ennen toisen version kehittämistä. Tämän lisäksi käsitellään ohjelman toisen version kehitys vaihe vaiheelta. Ohjelmassa on kaksi keskeistä funktiota, joita tarkastellaan tarkemmin, funktio, jolla muokataan tietokannasta saatu data haluttuun muotoon ennen sen vertailua ja funktio, joka suorittaa aineistojen vertailun sumeaa logiikkaa hyödyntäen.

Neljännessä luvussa vertaillaan ohjelman versioiden ajallista suorituskykyä sekä aineistojen yhdistämisen logiikkaa keskenään ja vedetään yhteen ohjelman kehitys. Tämän jälkeen viidennessä luvussa käsitellään vielä työn päätäntö.

2 OHJELMAN TEOREETTISET KÄSITTEET

2.1 Sumea logiikka

Sumea logiikka (eng. fuzzy logic) on matemaatikko Lofti Zadehin määrittelemä teoria, jossa perinteisen kaksiarvologiikan sijaan sallitaan kaikki totuuden eri vaihtoehdot epätodesta (0) toteen (1). Totuusarvo voidaan siis määritellä siten, että asiat voivat olla muutakin kuin vain täysin tosia tai täysin epätosia. (Kivinen & Uusitalo 1999, 734–735; Hellmann 2001, 1.)

Esimerkkinä voi olla tilanne, jossa päädytään vastaukseen, joka on 0,6 tai 60 %, voidaan siis todeta, että vastaus on hieman enemmän tosi kuin epätosi. Tällä tavoin esimerkiksi erilaisiin tietokoneohjelmiin saadaan enemmän inhimillistä ajattelutapaa. Tästä syystä monet laitteet kodinkoneista autojen automaattivaihteistoon toimivat sumean logiikan pohjalta. Sumean logiikan käyttö on todettu kannattavaksi myös monissa erilaisissa monimutkaisissa prosessien valvonta- ja ohjaustehtävissä. (Kivinen & Uusitalo 1999, 734–735; Hellmann 2001, 1.)

Sumea logiikka käsittelee asteista vaihtelua perinteisen kaksiarvologiikan sijaan, joten sitä voidaan nimittää myös moniarvologiikaksi. Tästä moniarvologiikasta hyvänä esimerkkinä voidaan pitää tilannetta, jossa määritellään, onko ihminen köyhä vai rikas. Mistä tiedetään missä rikkaan tai köyhän raja menee? Tässä tilanteessa asteittainen vaihtelu on tapa, jolla asia voidaan määritellä. Ei siis ole vain kahta arvoa, jotka ovat köyhä ja rikas, vaan monia eri totuuden asteita niiden arvojen välissä. (Kalliala 2019.)

Sumean logiikan hyödyiksi voidaan lukea myös se, että verrattuna perinteiseen sääntöpohjaiseen logiikkaan, jossa sääntöjä voi olla useita satoja voidaan sumean logiikan avulla sääntöjen määrää pudottaa huomattavasti pienemmäksi. Näin ollen systeemi, joka toimii sumealla logiikalla antaa johtopäätöksen johon sääntöjen antama keskiarvo painottuu systeemille annetulla syötteellä. (Kalliala 2019.)

2.2 Levenšteinin etäisyys

Levenšteinin etäisyys on käsite, jonka matemaatikko Vladimir Levenštein esitti vuonna 1966. Käsitteellä pyrittiin laskemaan kahden eri merkkijonon välinen editointietäisyys. Levenšteinin tapa laskea kahden eri merkkijonon välinen editointietäisyys perustuu Frederick J. Dameraun töihin. Damerau oli määritellyt mallin kirjoitusvirheille, tällä mallilla oli neljä erilaista virhetyyppiä, jotka olivat lisäys, poisto, korvaaminen ja permutaatio. (Aouragh ym. 2015, 1–6.)

Dameraun määrittelemästä neljästä virhetyypistä Levenštein otti itselleen käyttöön lisäyksen, poiston sekä korvaamisen ja toteutti näiden pohjalta oman tapansa laskea merkkijonojen välisen etäisyyden toisiinsa. Editointietäisyyden laskeminen perustuu siihen, kuinka monta lisäys-, poisto- tai korvaustoimenpidettä toiselle merkkijonolle tehdään ennen kuin merkkijonot vastaavat toisiaan. Levenšteinin etäisyyden yleinen käyttökohde on erilaisten kirjoitusvirheiden korjaaminen eri kielissä. Käsitteen avulla voidaan esimerkiksi luoda ohjelma, joka ehdottaa virheellisen sanan tilalle listan sanoja, joiden editointietäisyys on lähellä käyttäjän antamaa virheellistä sanaa. (Aouragh ym. 2015 1–6.)

Editointietäisyyden laskemiseen on olemassa valmiita kirjastoja esimerkiksi Pythonille. Levenšteinin etäisyyden kaava voidaan kuitenkin myös kirjoittaa itse (kuva 1). Jokainen sanalle toteutettu toimenpide kasvattaa sanan editointietäisyyttä yhdellä.

```
def levenshteinDistance(A, B):
    N, M = len(A), len(B)
    # Luodaan array joka on kooltaan nxm
    dp = [[0 for i in range(M + 1)] for j in range(N + 1)]

    # lähtötilanne kun N = 0
    for j in range(M + 1):
        dp[0][j] = j
    # lähtötilanne kun M = 0
    for i in range(N + 1):
        dp[i][0] = i
    # Muutokset
    for i in range(1, N + 1):
        for j in range(1, M + 1):
            if A[i - 1] == B[j - 1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(
                    dp[i-1][j], # lisäys
                    dp[i][j-1], # poisto
                    dp[i-1][j-1] # korvaus
                )

    return dp[N][M]

if __name__ == '__main__':
    A = ["Koira", "Nukke", "Joulu", "Maatila"]
    B = ["Kissa", "Nalle", "Koulu", "Hätätila"]
    for i in range(len(A)):
        print("Levenšteinin etäisyys sanojen {} ja {} välillä on = {}".format(A[i], B[i], levenshteinDistance(A[i], B[i])))
```

```
C:\Users\jompp\anaconda3\python.exe C:\Users\jompp\Desktop\fuzzystringmatch\levenstein.py
Levenšteinin etäisyys sanojen Koira ja Kissa välillä on = 3
Levenšteinin etäisyys sanojen Nukke ja Nalle välillä on = 3
Levenšteinin etäisyys sanojen Joulu ja Koulu välillä on = 1
Levenšteinin etäisyys sanojen Maatila ja Hatätila välillä on = 3

Process finished with exit code 0
```

Kuva 1. Ohjelma, joka laskee Levenšteinin etäisyyden sanojen välillä.

Esimerkiksi sanojen kissa ja koira välisen editointietäisyyden ollessa 3 voidaan se selittää seuraavasti, sanojen ollessa yhtä pitkät ei poisto- tai lisäystoimenpiteitä tarvita, näin ollen sanojen editointietäisyys on vielä toistaiseksi 0. Seuraavaksi sanoja aletaan käymään läpi merkki kerrallaan, kunnes sanojen välillä löytyy eroavaisuus, tässä kohtaa merkki korvataan toisen vertailtavan sanan vastaavan merkkijonon kohdan vastaavalla merkillä ja sanojen välinen editointietäisyys kasvaa.

Vertailtavissa sanoissa, jotka olivat kissa ja koira on siis 3 eroavaisuutta, jotka voidaan poistaa käyttämällä korvaustoimenpidettä. Näiden toimenpiteiden jälkeen sanat vastaavat toisiaan ja sanojen väliseksi editointietäisyydeksi on saatu 3.

2.3 FuzzyWuzzy

FuzzyWuzzy tai toiselta nimeltään thefuzz on Python-kirjasto, joka Levenšteinin etäisyyttä käyttäen laskee kahden eri merkkijonon välisen eroavaisuuden. Verrattavien merkkijonojen vastaavuuden läheisyyttä mitataan muokkausetäisyydellä. Muokkausetäisyys on siis tarvittava operaatioiden määrä, jolla voidaan muuttaa merkkijonot täsmäämään keskenään. Näitä operaatioita ovat mm. merkin lisäys, merkin poisto ja merkin korvaaminen. Tällä menetelmällä voidaan esimerkiksi toteuttaa sovelluksia oikeinkirjoituksen tarkistamiseen, plagioinnin havaitsemiseen tai esimerkiksi erilaisten tekstien sensurointiin. (Bonzanini 2015.)

Fuzzywuzzy-kirjastossa käytettävien funktioiden palautusarvo ei suoraan ole Levenšteinin etäisyydessä esiintyvä muokkausetäisyys vaan se palauttaa kahden verrattavan merkkijonon yhtäläisyyspisteet välillä 0–100. Pisteytys siis määrittyy sen mukaan mikä merkkijonojen prosentuaalinen yhtäläisyys on. (Wong 2020).

Kirjasto siis pitää sisällään erilaisia metodeja joilla merkkijonoja voidaan vertailla, esimerkiksi yksinkertaisin tapa on *fuzz.ratio(),* joka palauttaa kahden eri merkkijonon yhtäläisyyspisteet vertaamalla merkkijonoja keskenään yksi yh-

teen ottaen huomioon merkkijonon merkkijärjestyksen, erikoismerkit yms. Toisena esimerkkinä on *fuzz.partial_ratio().* Tällä funktiolla voidaan vertailla merkkijonojen osia, funktio on erittäin hyödyllinen esimerkiksi nimien vertailussa. (Wong 2020).

Viimeinen käsiteltävä funktio on *fuzz.token_sort_ratio().* Tämä funktio poistaa merkkijonoista välimerkit, muuttaa isot kirjaimet pieniksi sekä laittaa merkkijonon aakkosjärjestykseen, minkä jälkeen merkkijonot vertaillaan keskenään ja annetaan yhtäläisyyspisteet. Tällä funktiolla voidaan siis vertailla merkkijonoja välittämättä siitä, miten merkit ovat merkkijonossa lähtötilanteessa aseteltu. (Wong 2020.)

Seuraavaksi käsitellään kolmen edellä mainitun funktion toiminta käytännössä. Testissä käytetään kolmea muuttujaa, jotka ovat x, y ja z. Muuttuja x saa arvoksi: ”Kissoja ja koiria”, muuttuja y saa arvoksi: ”Kissoja ja koiria!” ja muuttuja z saa arvoksi: ”Koiria ja kissoja” (kuva 2).

```
1  from fuzzywuzzy import fuzz
2
3  x = "Kissoja ja koiria"
4  y = "Kissoja ja koiria!"
5  z = "Koiria ja kissoja"
6
7  print("fuzz.ratio : " + str(fuzz.ratio(x, y)))
8  print("fuzz.partial_ratio: " + str(fuzz.partial_ratio(x, y)))
9  print("fuzz.token_sort_ratio : " + str(fuzz.token_sort_ratio(x,z)))
```

```
Run: fuzzyesimerkki
C:\Users\jompp\anaconda3\python.exe C:\Users\jompp\Desktop\fuzzystri
fuzz.ratio : 97
fuzz.partial_ratio: 100
fuzz.token_sort_ratio : 100

Process finished with exit code 0
```

Kuva 2. FuzzyWuzzy-kirjaston metodeja käytännössä

Fuzz.ratio()-esimerkissä käytetään muuttujia x ja y. Funktio antaa pistemääräksi 97, koska merkkijonoja vertaillaan suoraan yksi yhteen ja näin ollen muuttujan y sisältämä erikoismerkki saa tuloksen hieman laskemaan täydestä pistemäärästä, toisin kun taas *fuzz.partial_ratio()*, joka vertailee lyhyempää merkkijonoa pidempään. Muuttujan x ollessa lyhyempi merkkijono ja sisältäen saman osan merkkijonoa kuin muuttuja y saadaan pistemääräksi 100.

Fuzz.token_sort_ratio()-esimerkissä testataan muuttujia x ja z. Funktion toiminnan perustuessa siihen että erikoismerkit sekä välilyönnit poistetaan ja merkkijono muutetaan pieniksi kirjaimiksi ja asetetaan merkit aakkosjärjestykseen, saadaan täydet pisteet vaikka merkkijonot lähtötilanteessa eroaisivat toisistaan huomattavasti.

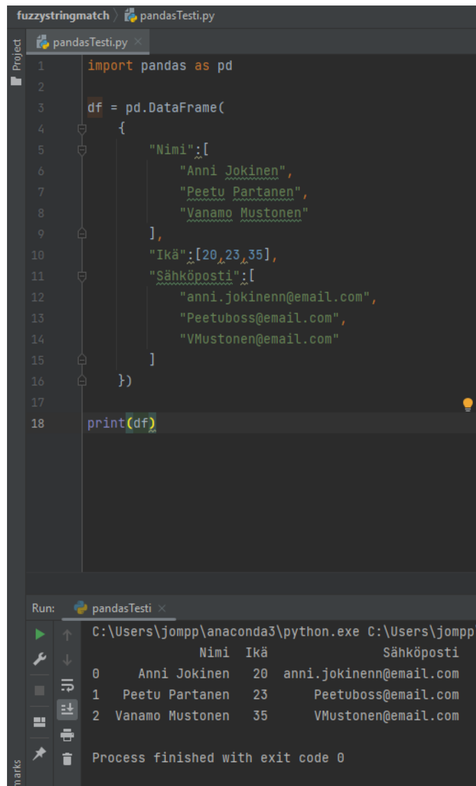
2.4 Pandas

Pandas on avoimen lähdekoodin Python-paketti, jota käytetään datan analysointiin ja käsittelyyn. Nimi pandas tulee sanoista panel data (suom. paneeli data), joka on termi monipuolisesti jäsennetyistä tietoaineistoista ja Pythonin omasta tietojen analysoinnista. Pandas antaa käyttäjälleen todella monipuolisen ja hyvin rakennetun tavan käsitellä sekä muokata dataa nopeasti ja helposti. (Pandas 2023a.)

Tässä työssä käytettäviä pandas-objekteja ovat DataFrame (suom. datakehys) ja Series (suom. sarja). DataFrame on kaksiulotteinen taulu, jossa sekä pysty- että vaakariveillä on omat otsikkonsa. Series on DataFramein yksi rivi, joka pitää sisällään kaiken sillä rivillä olevan tiedon, näin ollen voidaan sanoa, että kokonainen DataFrame on sarja tai kokoelma Seriesiä. (McKinney 2013, 4–5; Chen 2017 30–31.)

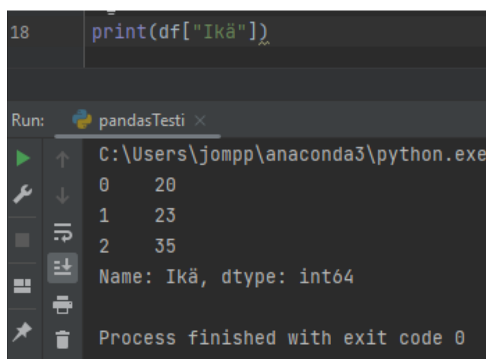
Yksi pandasin hyödyistä on se, että se yhdistää itsessään monia eri tekniikoita kuten NumPy:n joka käsittelee taulukoita, matriiseja sekä matemaattisia funktioita, laskentataulukkojen monipuolisia tapoja käsitellä dataa sekä relaatiotietokannat kuten SQL. Näiden tekniikoiden yhdistelyn vuoksi datan erilainen käsittely pandasin avulla on erityisen tehokasta sekä helppoa. (McKinney 2013, 4–5; Chen 2017 30–31.)

Kuvassa 3 on esimerkki pandasin Dataframe-objektista, sekä siitä millaisessa muodossa objekti tulostuu ilman erillisiä määrityksiä. Objektille on luotu kolme eri saraketta eli Seriesiä, jotka ovat nimi, ikä ja sähköposti. Nimi- sekä sähköpostisarake sisältävät tekstimuotoista tietoa, kun taas ikäsarakkeen tiedot ovat numeromuodossa.



Kuva 3. Esimerkki pandas Dataframe-objektista sekä objektin tulostus.

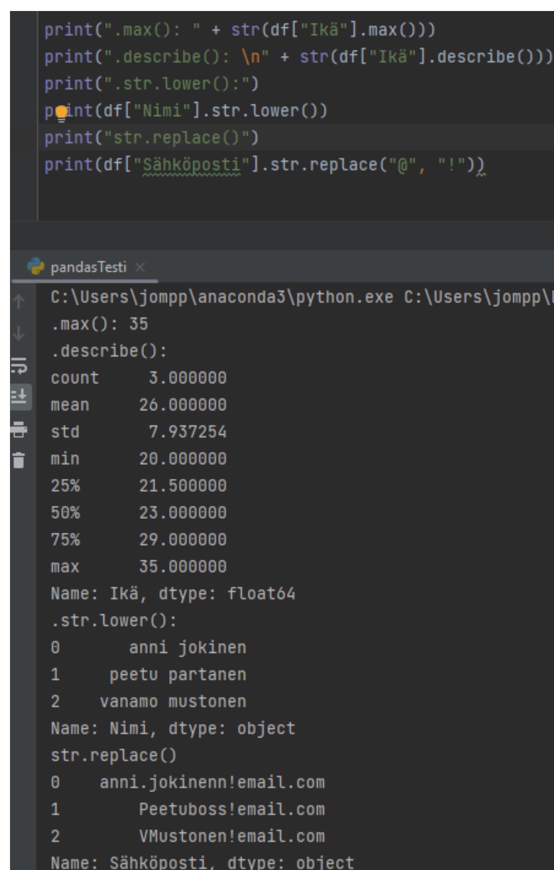
Jokaisen sarake voidaan myös tulostaa erikseen määrittelemällä kyseisen sarakkeen nimi tulostusasetuksiin. Tämä toiminto on erittäin hyödyllinen, jos halutaan käsitellä vain tietyn sarakkeen tietoja. (Pandas 2023b.) Esimerkkiohjelman tulostusasetuksia on muutettu niin että ohjelma tulostaa vain ikäsarakkeen tiedot aiemmin luodusta esimerkistä (kuva 4).



Kuva 4. Esimerkki Ikäsarakkeen tietojen tulostuksesta.

Datan käsittelylle Dataframe-objektissa on olemassa lukuisa määrä eri metodeja. Numeraalisista tiedoista voidaan muun muassa hakea tietyn sarakkeen maksimi käyttämällä *.max()*-funktiota. Tämä funktio siis käy läpi kaikki sarakkeen arvot ja palauttaa näistä suurimman. Dataframe-objektista on myös mahdollista saada ulos yhden sarakkeen perustilastoja käyttämällä *.describe()*-funktiota. (Pandas 2023b.)

Dataframe-objektissa olevaa tekstimuotoista tietoa on myös mahdollista käsitellä monilla eri metodeilla (kuva 5). Objektissa olevaa tekstiä voidaan käsitellä niin kuin mitä tahansa merkkijonoa Pythonissa. Esimerkkinä *str.lower()*-funktiolla voidaan halutessaan muuttaa tietyn sarakkeen kirjaimet pieniksi kirjaimiksi. *str.replace()*-funktion avulla taas voidaan muuttaa haluttu merkki tai merkkijono toiseksi. (Pandas 2023c.)



Kuva 5. Dataframe-objektin tietojen käsittelyä eri metodeilla.

Yhteenvetona voidaan todeta, että pandasin DataFrame-objektin avulla voidaan käsitellä ja hallita dataa todella yksinkertaisesti, mutta se soveltuu todella hyvin huomattavasti monimutkaisempaankin datan käsittelyyn.

2.5 Python

Python on oliopohjainen ohjelmointikieli, joka sisältää muun muassa monia erilaisia moduuleja, dynaamisia tietotyyppejä sekä luokkia. Vaikka pythonista puhutaan oliopohjaisena ohjelmointikielenä, se tukee myös muita erilaisia ohjelmointikonsepteja kuten proseduraalista ja funktionaalista ohjelmointia. Python on myös hyvin siirreltävissä eri käyttöjärjestelmien välillä, se toimii monissa Unix-käyttöjärjestelmissä, kuten Linuxissa. Tämän lisäksi ohjelmointikieli toimii myös macOS:ssä sekä Windowsissa. (Python documentation 2023a.)

Pythonin eduksi voidaan lukea se, että sen ollessa hyvin yleiskäyttöinen ohjelmointikieli sillä voidaan toteuttaa ratkaisuja ongelmiin hyvinkin laajalta skaalalta. Kieli pitää sisällään hyvinkin laajan standardikirjaston, jolla voidaan suorittaa monia hyvinkin erilaisia toimenpiteitä. Kirjastosta löytyy työkaluja esimerkiksi merkkijonojen käsittelyyn, internetprotokolliin ja käyttöjärjestelmien rajapintojen kanssa toimimiseen. Standardikirjaston lisäksi tarjolla on myös laaja valikoima kolmannen osapuolen laajennuksia ja paketteja. (Python documentation 2023b.)

Kuten muillakin ohjelmointikielillä myös Pythonilla on oma syntaksinsa, joka määrittelee millä tavoin Python-ohjelma kirjoitetaan. Pythonin syntaksissa on yhtäläisyyksiä muihin ohjelmointikieliin kuten Perl, C ja Java. Siitä huolimatta kielten välillä on selviä eroja. (Python Basic Tutorial 2023a.)

Monissa ohjelmointikielissä koodilohkoja määritellään aaltosulkeiden avulla ja ohjelman luettavuutta parannetaan sisennyksillä, Pythonissa taas koodilohkot määritellään kaksoispisteiden ja sisennyksien avulla. Sisennyksien määrä voi voi vaihdella, mutta saman lohkon sisällä olevia lausekkeita on sisennettävä yhtä paljon (kuva 6). (Python Basic Tutorial 2023a.)

```java
public class MyClass {
    public static void main(String args[]) {
      if(20 > 19) {
          System.out.println("True");
      } else {
          System.out.println("False");
      }
    }
}
```

```python
if 20 > 19:
    print ("True")
else:
    print ("False")
```

Java

Python

Kuva 6. Esimerkki Java ja Python koodin syntaksierosta

Pythonissa on myös sääntöjä muuttujien, funktioiden, luokkien, moduulien tai muiden objektien nimeämiseen. Tunniste näille edellä mainituille voi alkaa millä tahansa kirjaimella a-z pienellä tai isolla kirjoitettuna tai alaviivalla, jonka jälkeen on haluttu määrä kirjaimia tai numeroita 0-9. Kiellettyjä merkkejä tunnisteissa ovat välimerkit kuten @ ja $. Tunnisteiden luomisessa on myös muutamia käytäntöjä, kuten se että luokkien nimet Pythonissa aloitetaan isolla kirjaimella, kun taas muut tunnisteet aloitetaan pienellä kirjaimella. (Python Basic Tutorial 2023.)

Python on siis todella aloittelijaystävällinen ohjelmointikieli helppolukuisuutensa ja yleiskäyttöisyytensä vuoksi. Sillä voidaan toteuttaa monia erilaisia sovelluksia, nettiselaimia ja pelejä. Tarvittaessa kuitenkin se soveltuu todella monimutkaiseenkin ohjelmointiin vaativammissa tehtävissä. (Python Basic Tutorial 2023b.)

3 OHJELMAN KEHITTÄMINEN

3.1 Ohjelman lähtötilanne

Työn käytännön osuus aloitettiin palavereilla toimeksiantajan kanssa, joissa käytiin läpi ohjelman senhetkinen versio sekä keskusteltiin parannuksista ja muutoksista, joita sille halutaan tehdä. Keskusteluissa nousi esille se, että ohjelman tekemät väärät patenttihakemusten ja yrityksien nimien yhdistämiset halutaan minimoida, jolloin päästään lopputulokseen, jossa kaikkia hakemuksia ei välttämättä voida yhdistää mihinkään yrityksiin, mutta ohjelma ei antaisi virheellistä tietoa. Lisäksi nousi esille se, että ohjelman ajaminen vie runsaasti

aikaa. Ohjelmaa varten työnantaja loi hankkeelle oman GitHub-kansioon, jonka avulla voidaan toteuttaa ohjelman versiohallintaa.

Ennen ohjelman testaamisen ja muokkaamisen aloittamista asennettiin työhön tarvittavia paketteja käyttäen Pythonin *pip install*-komentoa. Pip on Pythonin käyttämä pakettiasennustyökalu, jolla voidaan asentaa paketteja Pythonin omasta pakettihakemistosta tai muista hakemistoista (Pip documentation 2023).

Työhön vaadittiin paketteja pandasia sekä RapidFuzzia varten. RapidFuzz on toimintaperiaatteeltaan sama paketti kuin FuzzyWuzzy, mutta suorittaa merkkijonojen vertailun prosessointia huomattavasti nopeammin. Paketit voidaan asentaa käyttämällä esimerkiksi PyCharm-editorin terminaalia. Pandas-paketti asennettiin komennolla *pip install pandas* ja RapidFuzz-paketti komennolla *pip install rapidfuzz* (kuva 7).

```
PS C:\Users\jompp\Desktop\fuzzystringmatch> pip install rapidfuzz
Collecting rapidfuzz
  Downloading rapidfuzz-3.0.0-cp310-cp310-win_amd64.whl (1.8 MB)
     ---------------------------------------- 1.8/1.8 MB 5.8 MB/s eta 0:00:00
Installing collected packages: rapidfuzz
Successfully installed rapidfuzz-3.0.0
```

Kuva 7. *rapiduzz*-paketin asentaminen *pip install*-komennolla

Asennuksien jälkeen molemmista paketeista tarvittavia metodeja voidaan tuoda projektiin käyttämällä *import*-komentoa. Paketeista voidaan myös hakea yksittäisiä metodeja ja/tai muuttujia kirjoittamalla *import*-komento muodossa *from paketti import funktio*. Kolmas projektiin tuotava paketti on Pythoniin sisään rakennettu paketti nimeltään RegEx. Paketin ollessa Pythoniin sisään rakennettuna ei sitä tarvitse asentaa erikseen vaan paketti toimii suoraan projektissa tuomalla se *import*-komennolla. RegEx:iä käytetään tekstien ja sen osien vertailuun, etsimiseen ja muokkaamiseen erilaisten mallien avulla, jotka sille voi itse määrittää (Mdn web docs2023).

Ohjelman toiminta tällä hetkellä on seuraavanlainen, aluksi ohjelma ottaa yhteyden tietokantaan, josta se hakee yritysten sekä patenttihakemuksien tiedot.

Tämän jälkeen yrityksien nimet sekä patenttihakemuksien tekijöiden nimet käsitellään. Nimet muutetaan kokonaan kirjoitettaviksi pienillä kirjaimilla käyttäen *.lower()*-funktiota.

Kirjainkoon muokkaamisen jälkeen merkkijonoista poistetaan osia käyttäen RegEx:in *.sub()*-funktiota. Funktiota kutsutaan 3 kertaa eri parametreilla. Ensimmäinen funktio muokkaa sille annettua merkkijonoa siten, että siitä poistetaan kaikki merkit, jotka eivät ole aakkosnumeraalisia merkkejä tai å, ä, öü. Merkit korvataan välilyönnillä.

Seuraavan funktion tarkoituksena on poistaa merkkijonoista kaikki peräkkäiset välilyönnit ja korvata ne vain yhdellä välilyönnillä. Funktio on kuitenkin tässä versiossa kirjoitettu vääriin eikä näin ollen tee mitään. Tämän funktion korjaaminen toteutetaan ohjelman kehitysvaiheessa.

Viimeinen funktio tarkistaa onko sanan merkkijonon lopussa jokin *stopwords*-listan sanoista, poistaa sanan ja korvaa sen välilyönnillä. Funktio on määritelty siten, että listassa annettu sana poistetaan vain, jos se esiintyy tekstin lopussa erikseen, esimerkiksi Abloy-sanasta ei poistettaisi mitään, vaikka sana loppuukin oy-merkkeihin. Lopuksi käsitelty merkkijono palautetaan käyttämällä siihen vielä *.strip()*-funktiota, joka poistaa merkkijonon alussa tai lopussa esiintyvät ylimääräiset välilyönnit. (kuva 8).

```
def clean_company_name(name):
    stopwords = ['oy', 'ab', 'ky', 'oyj', 'gmbh', 'ltd', 'corp', 'inc', 'oy ab', 'rf', 'seura']
    name = name.lower()
    name = re.sub(r'[^a-z0-9åäöü]', ' ', name)
    name = re.sub(r'[ ]{2+}', ' ', name)
    # poistaa vain kun merkkijonon lopussa
    name = re.sub(r'(\b' + r'\b|\b'.join(stopwords) + r'\b)$', ' ', name)
    return name.strip()
```

Kuva 8. Funktio yritysten nimien ja patenttihakemusten tekijöiden datan muotoiluun

Datan muokkaamisen jälkeen suoritetaan itse merkkijonojen välinen vertailu. Funktiolle annetaan parametreina lista yrityksien nimiä ja patenttihakemuksia, sekä samankaltaisuuden alaraja, joka *fuzz.ratio()*-funktion on annettava jotta patenttihakemus yhdistetään yritykseen ja lisätään listaan. Ohjelman tämän-

hetkisessä versiossa yhtäläisyyden pistemäärän on oltava vähintään 90. Lisäksi funktio ilmoittaa kuinka monta prosenttia sille annetusta datasta on käsitelty (kuva 9).

```
def match_company_applicants(companies, applicants, cutoff):
    #    companies = df1.loc[:, col1].tolist()
    #    applicants = df2.loc[:, col2].unique().tolist()
    checked = 0
    total = len(companies)
    match_dict = {}
    for company in companies:
        matches = process.extract(company, applicants, scorer=fuzz.ratio, limit=10, score_cutoff=cutoff)
        for match in matches:
            match_dict[company] = []
            print(company, ":", match[0], "=", match[1])
            match_dict[company].append((match[0], match[1]))

        checked += 1
        if checked % 1000 == 0:
            percentage_completed = checked / total * 100
            print(f"{checked} companies have been checked. {percentage_completed:.2f}% completed.")

    #    df1["matches"] = df1[col1].map(match_dict)
    return match_dict


matches = match_company_applicants(company_basenames[200000:210000], applicants, 90)
```

Kuva 9. Funktio merkkijonojen vertailuun sekä funktion kutsu

Nyt kun ohjelman nykytila on käsitelty, voidaan lähteä toteuttamaan ohjelmaan muutoksia, joilla sen toimintaa voidaan kehittää parempaan suuntaan. Seuraavassa kappaleessa käsitellään ohjelmaan tehtyjä muutoksia.

3.2 Ohjelman uuden version toteutus

Ohjelman kehittämisen aloitin muokkaamalla edellä mainitun väärin kirjoitetun *.sub()*-funktion oikeaan muotoon. Toimiakseen oikein funktion ensimmäisen parametrin tulee olla kirjoitettuna muotoon *r'[ ]{2,}'*, näin tekemällä saadaan haluttu lopputulos joka poistaa merkkijonosta useat perättäiset välilyönnit.

Jatkoin ohjelman kehittämistä tekemällä Excel-taulukon tämänhetkisen *clean_company_name()*-funktion muokkaamista patenttihakemuksista ja yritysten nimistä. Taulukkoja tutkiessa etsin usein esiintyä alku- tai loppuliitteitä. Ensimmäisenä havaintona huomasin, että usein yritysmuoto on yrityksen nimen edessä lyhenteenä tai kokonaan kirjoitettuna. Datassa esiintyi tilanteita, joissa selvästi sama yritys oli tehnyt patenttihakemuksen kirjoittaen yhtiön nimen erilaisessa muodossa.

Lähdin muokkaamaan funktiota siten että loin uuden listan sanoista, joita esiintyy usein patenttihakemusten tai yritysten nimien alussa. Tämän jälkeen toteutin muuttujan nimeltään *pattern_for_start_words*. Muuttujan arvoksi asetin mallin, jonka avulla voidaan tarkistaa esiintyykö *stopwords_start*-listassa olevia sanoja merkkijonon alussa. Muutin myös merkkijonon loppuosan tarkastamista siten, että malli, joka tarkastaa merkkijonon loput on asetettu muuttujaan *pattern_for_end_words*.

Mallien ollessa muuttujia voidaan ne asettaa parametreiksi seuraavalla tavalla kun funktiota *.sub()* kutsutaan: *re.sub(pattern_for_end_words, '', name).* Nyt merkkijonoja voidaan muokata sekä alusta että lopusta, jolloin saadaan kasvatettua mahdollisuutta sille, että oikeat hakemukset ja yritykset saadaan yhdistettyä, vaikka hakemukseen yrityksen nimi olisikin kirjoitettu erilaisessa muodossa (kuva 10).

```
def clean_company_name(name):
    stopwords_end = ['oy', 'ab', 'ky', 'oyj', 'gmbh', 'ltd', 'corp', 'inc', 'oy ab', 'rf', 'seura', 'osuuskunta','ab oy']
    pattern_for_end_words = r'(\b' + r'\b|\b'.join(stopwords_end) + r'\b)$'
    stopwords_start = ['oy', 'ab', 'oy ab', 'osakeyhtiö', 'osakeyhtioe', 'osuuskunta', 'ab oy']
    pattern_for_start_words = r'^(' + '|'.join(stopwords_start) + r')\b\s*'

    name = name.lower()
    name = re.sub(r'[^a-z0-9åäöü]', ' ', name)
    name = re.sub(r'[ ]{2,}', ' ', name)

    # poistaa vain kun merkkijonon lopussa
    name = re.sub(pattern_for_end_words, '', name)

    # poistaa vain kun merkkijonojen alussa
    name = re.sub(pattern_for_start_words, '', name)

    return name.strip()
```

Kuva 10. Päivitetty versio *clean_company_names()*-funktiosta.

Vertailtavien merkkijonojen välillä eroavaisuuksia kirjoitusasussa voi olla muissakin kohdissa kuin vain mihin kohtaan merkkijonoa yritysmuoto on kirjoitettu. Tästä ongelmasta hyvänä esimerkkitapauksena toimii Lappeenrannan–Lahden teknillinen yliopisto LUT:n, sekä Åbo Akademin tekemät patenttihakemukset.

Yrityksien nimet ohjelman käyttämissä yritystiedoissa ovat muodossa Lappeenrannan teknillinen yliopisto ja Åbo Akademi, kun taas patenttihakemuksissa molempien yritysten nimet esiintyvät monissa eri muodoissa, kuten esimerkiksi Lappeenrannan-Lahden teknillinen yliopisto lut, sekä åbo akademi

åbo akademi university. Esimerkkinä Lappeenrannan-Lahden teknillinen yliopisto lut ja Lappeenrannan teknillinen yliopisto merkkijonojen välinen yhtäläisyyspistemäärä on *fuzz.ratio()*-funktiolla vain 86, eivätkä ne näin ollen tulisi yhdistymään toisiinsa ohjelman nykyisellä versiolla.

Nostaakseni yhtäläisyyksien pistemäärää tein funktion nimeltään *change_name()*. Funktion toiminta on seuraavanlainen, funktiolla on kolme parametria: *oname, list, cname. oname*-parametri on funktioon sisään tuleva patenttihakijan nimi. *list* on lista merkkijonoista, joka korvataan *cname*-parametrin syötteellä, joka tässä tapauksessa on yrityksen nimi siinä muodossa, jossa se on yritysdatassa, jos *oname*-merkkijono vastaa jotain listan merkkijonoista. Funktion ollessa valmis tein kaksi listaa, joissa molemmissa on sisältönä aineistossa esiintyvät monet eri vaihtoehdot LUT:n ja Åbo Akademin tekemille hakemuksille (kuva 11).

```
list_of_lut_possibilities = ["lappeenrannan lahden teknillinen yliopisto lut", "lappeenrannan lahden teknillinen yliopisto", "lappeenrannan lahden teknillinen yliopisto lut",
               "lappeenrannan teknillinen ylio", "lappeenrannan laehden teknillinen yliopisto lut",
               "lappeenranta lahti univ of technology lut", "lappeenranta univ of tech",
               "lappeenranta univ of technology",
               "lappeenranta university of technology"]

list_of_abo_possibilities = ["åbo akadem", "abo akademi abo akademi university", "åbo akademi åbo akademi university",
                             "abo akademi abo akademi univ", "åbo akademi åbo akademi univ", "abo akademi univ", "åbo akademi univ",
                             "abo akademi university", "åbo akademi university"]


def change_name(oname, list, cname):
    if oname in list:
        return cname
    else:
        return oname
```

Kuva 11. *change_name()*-funktion toiminta sekä listat tarkastettavista sanoista

Funktiota voidaan nyt tarvittaessa kutsua niin monta kertaa *clean_company_name()*-funktion sisällä kuin tarvetta luomalla uusia listoja vastaavanlaisista tilanteista. Tässä ohjelman versiossa funktiota kutsutaan kaksi kertaa, jotta sekä Lappeenrannan teknillisen yliopiston sekä Åbo Akademin patenttihakemukset sekä yrityksien nimet saadaan yhdistymään merkkijonoja vertailtaessa. Lisäksi lisäsin *clean_company_name()*-funktioon parametrin nimeltään *use_changename*, jolle annetaan arvoksi joko True tai False riippuen siitä halutaanko *change_name()*-funktiota käyttää, näin ollen voidaan määrittää että funktiota käytetään vain patenttidatan läpikäynnissä (kuva 12).

```python
def clean_company_name(name, use_changename):

    stopwords_end = ['oy', 'ab', 'ky', 'oyj', 'gmbh', 'ltd', 'corp', 'inc', 'oy ab', 'rf', 'seura', 'osuuskunta',
                     'ab oy']
    pattern_for_end_words = r'(\b' + r'\b|\b'.join(stopwords_end) + r'\b)$'
    stopwords_start = ['oy', 'ab', 'oy ab', 'osakeyhtiö', 'osakeyhtioe', 'osuuskunta', 'ab oy']
    pattern_for_start_words = r'^(' + '|'.join(stopwords_start) + r')\b\s*'

    name = name.lower()
    name = re.sub(r'[^a-z0-9åäöü]', ' ', name)
    name = re.sub(r'[ ]{2,}', ' ', name)

    # poistaa vain kun merkkijonon lopussa
    name = re.sub(pattern_for_end_words, '', name)

    # poistaa vain kun merkkijonojen alussa
    name = re.sub(pattern_for_start_words, '', name)

    # tarkistetaan käytetäänko nimen muokkausta, näin ollen nimen muokkausta ei ole pakko käyttää molempien datojen
    # muotoilussa
    if use_changename:
        name = change_name(name, list_of_lut_possibilities, "lappeenrannan teknillinen yliopisto")
        name = change_name(name, list_of_abo_possibilities, "åbo akademi")

    return name.strip()
```

```python
applicants = [clean_company_name(rows[0], True) for rows in data]
```

Kuva 12. *change_name()*- ja muokatun *clean_company_name()*-funktion kutsuminen

Viimeisenä muokattiin *match_company_applicants()*-funktiota, joka suorittaa varsinaisen aineistojen vertailun. Funktion alku noudattaa pitkälti samaa kaavaa kuin edellisessäkin versiossa. Luovuin ajatuksesta muuttaa haettavat tiedostot pandas-taulukoiksi, sillä se toisi vain yhden työvaiheen lisää funktioon antamatta mitään lisäarvoa koodin toiminnalle. Haettava data siis käsitellään listamuotoisena niin kuin edellisessäkin funktion versiossa.

Ensimmäinen varsinainen isompi muutos koskee tilannetta, jolloin merkkijonossa esiintyy välilyönti. Aineistossa esiintyi useita kertoja tilanteita, joissa yrityksen nimi oli jokin alle kolmen merkin mittainen sana tai lyhene ja sen perässä välilyönnin jälkeen oli jokin pääte kuten esimerkiksi solutions tai innovations. Tällaisia tilanteita varten loin muuttujan nimeltään *space_index,* joka saa arvokseen merkkijonossa mahdollisesti olevan välilyönnin indeksin käyttämällä *.find()*-metodia. Muuttuja etsii välilyöntejä merkkijonoista, joista on jo saatu osuma vertailua tehdessä (kuva 13).

```python
for match in matches:
    space_index = match[0].find(" ")
```

Kuva 13. *space_index*-muuttuja, joka etsii mahdollista välilyöntiä

Seuraavaksi osuman sekä yrityksen nimen merkkijono jaetaan kahteen osaan ensimmäisen merkkijonossa mahdollisesti esiintyvän välilyönnin kohdalta käyttämällä *.split()*-metodia. Tämän jälkeen tarkistetaan ehtolauseen avulla onko osumasta löydetty välilyöntiä ja onko osiin jaetussa yrityksen nimessä

enemmän osia kuin yksi käyttämällä loogista operaattoria *and.* Mikäli nämä ehdot täyttyvät, vertaillaan patentin hakijan ja yrityksen nimen ensimmäisiä ja toisia osia keskenään ja tulokset annetaan muuttujien *first_score* sekä *second_score* arvoiksi (kuva 14).

```
applicant_string_parts = match[0].split(" ", 1)
company_string_parts = company.split(" ", 1)
if space_index != -1 and len(company_string_parts) > 1:
    first_score = fuzz.ratio(company_string_parts[0], applicant_string_parts[0])
    second_score = fuzz.ratio(company_string_parts[1], applicant_string_parts[1])
```

Kuva 14. Merkkijonojen pilkkominen osiin, sekä osien vertailu ja pisteytys

Osien pisteytyksen jälkeen funktio käy läpi sarjan ehtolauseita. Ensimmäinen ehtolause toimii seuraavasti: jos patentin hakijan tai yrityksen ensimmäisen osan merkkijonon pituus on kolme merkkiä tai vähemmän, ja muuttujan *first_score* arvo on 100 sekä *second_score*-muuttujan arvo on vähintään annetun *score*-parametrin arvo, niin kyseinen osuma lisätään listaan. Mikäli nämä ehdot eivät täyty siirrytään seuraavaan ehtoon, joka tarkistaa sisältääkö patentin hakijan tai yrityksen merkkijonon ensimmäinen osa vähintään neljä, mutta enintään kuusi merkkiä.

Tässäkin ehdossa tutkitaan osien saamia pisteitä, mutta pisteytyksen vaatimuksia on muutettu siten, että jotta osuma lisätään listaan, tulee ensimmäisen että toisen osan pistemäärän olla vähintään *scoren* arvo. Viimeinen ehto, joka lisää osuman listaan toimii samalla tavalla kuin edellinen ehto mutta tarkistaa vain onko kummankaan merkkijonon ensimmäisen osan pituus yli kuusi merkkiä. Mikäli näistä yllä mainituista ehdoista ei täyty ehtolauseen *else* lohkossa komento *continue* siirtyy tarkastelemaan seuraavaa osumaa listassa, jota käydään läpi (kuva 15).

```
if (len(applicant_string_parts[0]) <= 3 or len(company_string_parts[0]) <= 3) and first_score == 100 and second_score >= score:
    matched_rows.append({'Company': company, 'Applicant': match[0], 'Score': match[1]})
elif (3 < len(applicant_string_parts[0]) <= 6 or 3 < len(company_string_parts[0]) <= 6) and first_score >= score and second_score >= score:
    matched_rows.append({'Company': company, 'Applicant': match[0], 'Score': match[1]})
elif (len(applicant_string_parts[0]) > 6 or len(company_string_parts[0]) > 6) and first_score >= score and second_score >= score:
    matched_rows.append({'Company': company, 'Applicant': match[0], 'Score': match[1]})
else:
    continue
```

Kuva 15. Ehtolause, joka tarkastelee ositettuja merkkijonoja ja lisää niitä listaan

Palataanpa funktiossa hieman taaksepäin. Seuraavaksi käsiteltävä osio on ehto, joka käsitellään, jos aiemmin mainitun *space_index*-muuttujan arvoksi tulee -1, mikä tarkoittaa sitä, että merkkijono ei sisällä välilyöntiä.

Ensimmäinen ehto jota tarkastellaan mikäli välilyöntiä ei löydy on lähestulkoon samanlainen kuin aiemmin esitelty ehto sille, mikäli merkkijonon osan pituus on enintään kolme merkkiä. Tässä ehdossa tarvitsee tarkistaa vain *match[0]*, joka on osuman ensimmäinen osa joka sisältää patenttihakijan nimen, sekä *match[1]*, joka sisältää osuman saaman pistemäärän. Tämän enempää osumien tarkastelua ei vertailla, mikäli se ei sisällä välilyöntiä, vaan ehtolauseen *else*-lohko lisää osuman listaan muissa tapauksissa (kuva 16).

```
elif len(match[0]) <= 3 and match[1] == 100:
    matched_rows.append({'Company': company, 'Applicant': match[0], 'Score': match[1]})
else:
    matched_rows.append({'Company': company, 'Applicant': match[0], 'Score': match[1]})
```

Kuva 16. Ehdot joita tarkistellaan, mikäli osumassa ei ole välilyöntiä

Säilytin funktiossa sen aiemmassa versiossa mukana olleen osan, joka tulostaa aina ilmoituksen kun 1 000 yritystä on käyty läpi sekä kuinka monta prosenttia kaikista läpi käytävistä se on. Tällä tavoin on funktion edistymistä helpompi seurata. Lopuksi funktio luo *matched_rows*-listasta pandasin DataFrame-objektin, jossa on kolme saraketta, jotka ovat *Companies, Applicants* ja *Score,* eli objektissa on yrityksen nimi, patentin hakijan nimi sekä yhtäläisyyden pistemäärä. Funktio palauttaa tämän objektin palautusarvonaan (kuva 17).

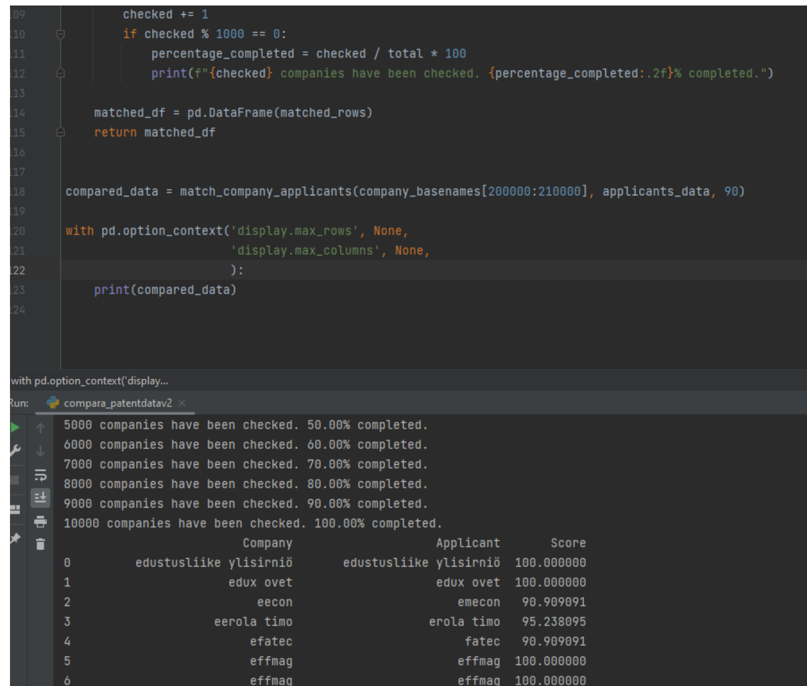
```
    checked += 1
    if checked % 1000 == 0:
        percentage_completed = checked / total * 100
        print(f"{checked} companies have been checked. {percentage_completed:.2f}% completed.")

matched_df = pd.DataFrame(matched_rows)
return matched_df
```

Kuva 17. Funktion edistymisen tarkastelu sekä listan muuttaminen DataFrame-muotoon

Funktion ollessa valmis voidaan sitä kutsua samalla tavalla kuin ohjelman edellisessä versiossa ja antaa siitä saatava DataFrame-objekti muuttujan ar-

voksi. Tätä objektia voidaan tarkistella kokonaisuudessaan kutsumalla *pd.option_context()*-funktiota, jolla voidaan luoda väliaikaiset asetukset sille miten objekti esitetään (Pandas 2023d). Asettamalla asetuksien *display.max_rows* sekä *display.max_columns* arvoksi None, saadaan objektin jokainen rivi ja sarake näkyviin (kuva 18).



Kuva 18. Funktion kutsu sekä DataFrame-objektin tulostaminen

Ohjelman kehityksen ollessa valmis voidaan siirtyä vertailemaan ohjelman versioita keskenään sekä vetämään yhteenveto siitä, toimiiko ohjelma halutulla tavalla ja päästiinkö tavoitteeseen ohjelman kehityksessä.

4 TULOSTEN VERTAILU JA YHTEENVETO

Ensimmäisenä vertailtiin ohjelmien ajoaikaa. Lisäsin molempiin ohjelmien versioihin yksinkertaisen laskurin, jonka avulla saadaan ohjelman ajoaika sekunteina. Testissä ajettiin ohjelmat siten, että ne kävivät läpi kaikki tietokannassa olevat noin 20 000 patenttihakemusta ja noin miljoona yrityksen nimeä.

Kuvassa 19 voidaan nähdä molempien ohjelman versioiden suoritusajat. Ohjelman vanhan version suoritusaika oli noin 9 038 sekuntia, eli 2 tuntia 30 minuuttia ja 38 sekuntia. Uuden version suoritusaika oli hieman lyhyempi, noin 8731, eli 2 tuntia 25 minuuttia ja 31 sekuntia.

Vanha versio

```
zoukem : zokem = 90.9090909090909
zoy : zoy = 100.0
z solutions : zef solutions = 91.66666666666
z solutions : t solutions = 90.9090909090909
z solutions : t solutions = 90.9090909090909
zun : zun = 100.0
Execution time: 9038.25717139244 seconds
```

Päivitetty versio

```
1044000 companies have been checked. 99.54%
1045000 companies have been checked. 99.63%
1046000 companies have been checked. 99.73%
1047000 companies have been checked. 99.83%
1048000 companies have been checked. 99.92%
Execution time: 8731.414925813675 seconds
```

Kuva 19. Ohjelman versioiden suoritusajat

Suurimpana tekijänä siihen, että ohjelman uusi version on nopeampi kuin vanha on se, että ohjelma ei tulosta jokaista löydettyä yhtäläisyyttä, kun niitä löydetään vaan pelkästään lopuksi listan kaikista osumista. Tämän tulostuksen jäädessä pois saadaan siis ohjelmasta nopeampi, vaikka logiikka joka merkkijonoja vertailee, on tietokoneelle hieman työläämpi prosessoida.

Seuraavana lähdin etsimään eroavaisuuksia ohjelman versioiden välisistä osumista. Tätä testiä varten ei tarvitsisi ajaa koko ohjelmaa läpi, joten ohjelmaa kutsuttiin siten että käytiin läpi kaikki patenttihakemukset mutta niitä verrattiin vain 10 000:n yrityksen nimeen.

Esimerkkitapauksena voidaan käyttää tilannetta, jossa ohjelman vanha versio yhdistää patentin hakijan, joka on patenttihakemuksien aineistossa nimellä ej suunnittelu yritykseen, joka on yritysnimien aineistossa nimellä el suunnittelu. Kyseessä ei ole kirjoitusvirhe, sillä kyseessä on täysin eri yritykset (kuva 20).

```
130 el suunnittelu : ej suunnittelu = 92.85714285714286
```

Kuva 20. Esimerkki ohjelman vanhan version virheellisestä yhdistämisestä

Ohjelman uudessa versiossa näin ei kuitenkaan ole, sillä siihen luotu ominaisuus, joka vertailee merkkijonon osia varmistaa, että juuri vastaavat tilanteet vältetään (kuva 21). Molemmat ohjelmien versiot siis löytävät oikeat osumat ej suunnittelulle, mutta ohjelman uusi versio ei anna virheellistä osumaa esimerkkitilanteessa. Tämän aiheuttaa ohjelman uuteen versioon toteutettu merkkijonojen osien erillinen ja portaittainen vertailu.

```
19              ej suunnittelu           ej suunnittelu  100.000
```

Kuva 21. Ohjelman uuden version antama osuma

Yhteenvetona voidaan siis sanoa, että tavoitteisiin, joita työlle oli asetettu, päästiin onnistuneesti. Saatiin luotua toimiva Python-ohjelma, joka vertailee kahta eri aineistoa sumeaa logiikkaa ja RegEx-kirjastoa käyttämällä ja luo tämän vertailun pohjalta uuden pandasin DataFrame-objektin. Ohjelman suorituskykyä sekä ohjelman logiikkaa aineistojen yhdistämiseen saatiin parannettua huomattavasti. Suurimpana tavoitteena ohjelman kehityksen kannalta oli virheellisen aineistojen yhdistämisen minimointi. Yllä olevan esimerkkitilanteen perusteella myös tähän tavoitteeseen on onnistuneesti päästy.

5 PÄÄTÄNTÖ

Tutkittaessa opinnäytetyön alussa asetettuja tavoitteita, voidaan todeta, että näihin tavoitteisiin on päästy. Aikaiseksi saatiin Python-ohjelma, joka vertailee kahta eri aineistoa keskenään, etsii niistä yhtäläisyyksiä ja luo näiden yhtäläisyyksien pohjalta uuden DataFrame-objektin. Ohjelman kehityksessä annettiin hyvin vapaat kädet lähteä muokkaamaan ohjelmaa paremmaksi. Työn toimeksiantaja oli myös tyytyväinen ohjelman kehitykseen. Ohjelman uusi versio ladataan toimeksiantajan GitHub-kansioon mahdollista jatkokehitystä varten sekä otetaan käyttöön toimeksiantajan hankkeessa.

Jatkokehitys ohjelmalle olisi mahdollista monissa tapauksissa, joten voidaan todeta, että tässä opinnäytetyössä toteutettu versio on hyvää kehitystä lähtötilanteeseen, mutta ei missä nimessä täydellinen. Esimerkiksi listat nimivaihtoehdoista voisi tulevaisuudessa automatisoida, tällä hetkellä ne tulee kirjoittaa manuaalisesti käsin. Aineistojen vertailua voisi myös porrastaa vieläkin tar-

kemmin merkki kerrallaan. Pienillä muutoksilla aineistojen vertailuun käytettävästä funktiosta saa myös huomattavasti yleiskäyttöisemmän, jonka avulla voi vertailla mitä tahansa dataa, ei vain tässä kyseisessä toimeksiannossa annettua dataa.

Python-ohjelmointikielenä oli itselle lähes täysin tuntematon ennen tätä työtä. Omissa opinnoissa kieltä ei ollut käytetty lainkaan, vaan ainut kosketuspinta mitä minulle Pythoniin oli, tuli vapaasti valittavien kurssien kautta. Tosin niin kuin itsekin työssä mainitsen, Python on todella aloittelijaystävällinen kieli, joten kun kokemusta oli muusta ohjelmoinnista jo hyvinkin paljon ei Pythonin opetteleminen tuonut suurempia ongelmia. Suurimpana erona muuhun ohjelmointiin, jota olin tehnyt oli se, että yleensä ohjelmissa ja sovelluksissa, joita toteutan, on jonkinlainen käyttöliittymä, tähän sellaista ei toteutettu. Tällä hetkellä koen hallitsevani Python-ohjelmoinnin varsin hyvin.

Olen hyvin tyytyväinen, että lähdin mukaan toteuttamaan tätä toimeksiantoa, vaikka koulun kurssien puolesta asia oli minulle lähestulkoon täysin uusi. Ohjelman toteuttaminen onnistuneesti uudella kielellä nostaa omaa itsetuntoa ohjelmoijana ja antaa hyvää asennetta tulevaisuudessa tarttua uusiin haasteisiin, vaikka ne veisivätkin pois omalta mukavuusalueelta. Toimeksiannon aihealueen ollessa todella mielenkiintoinen työn tekeminen oli kuitenkin todella mielekästä. Työn toteutuksen jälkeen koen olevani ohjelmoijana hieman parempi ja valmiimpi uusiin haasteisiin, joita tulen kohtaamaan työurani varrella enkä suhtaudu enää niin epäilevästi omiin kykyihini.

LÄHTEET

Aouragh, S., Gueddah, H. & Abdellah, Y. 2015. Adapting the Levinstein distance to contextual spelling correction. *International Journal of Computer Science and Applications*. 12, 127-133. PDF-dokumentti. Saatavissa: https://www.researchgate.net/profile/Si-Aouragh-2/publication/273758433_Adaptating_the_Levenshtein_Distance_to_Contextual_Spelling_Correction/links/5b9d8798299bf13e60343f0c/Adaptating-the-Levenshtein-Distance-to-Contextual-Spelling-Correction.pdf [viitattu 10.4.2023]

Bonzanini, M. 2015. Fuzzy String Matching in Python. Blogi. Päivitetty 25.2.2015. Saatavissa: https://marcobonzanini.com/2015/02/25/fuzzy-string-matching-in-python/ [viitattu 14.3.2023]

Chen, D. 2017. Pandas for Everyone: Python Data Analysis (Addison-Wesley Data & Analytics Series). 1. painos. Addison-Wesley Professional. E-kirja. Saatavissa: https://books.google.fi/books?id=7zhDDwAAQBAJ&newbks=1&newbks_redir=0&printsec=frontcover&pg=PT2&dq=Pandas+for+Everyone:+Python+Data+Analysis&hl=fi&redir_esc=y#v=onepage&q&f=false [viitattu 14.3.2023]

Helmann, M. 2001. Fuzzy Logic Introduction. 1. PDF-dokumentti. Saatavissa: http://epsilon.nought.de/tutorials/fuzzy/fuzzy.pdf [viitattu 21.3.2023]

Kalliala, E. 2019. Sumea logiikka --- portti pehmeään tietotekniikkaan. PDF-dokumentti. Päivitetty 4.9.2019. Saatavissa: https://eijakalliala.fi/wp-content/uploads/sites/4/2019/09/sumea.pdf [viitattu 14.3.2023]

Kivinen, V. & Uusitalo J. 1999. Sumea logiikka ja sen soveltaminen hakkuukoneen apteerauksen ohjauksessa. *Metsätieteen aikakausikirja.* 734–735. PDF-dokumentti. Saatavissa: https://jukuri.luke.fi/bitstream/handle/10024/533932/Kivinen.pdf?sequence=1 [viitattu 14.3.2023]

McKinney, W. 2013. Python for Data Analysis. Sebastopol: O'Reilly Media. [viitattu 14.3.2023]

Mdn web docs. 2023. Regular expressions. WWW-dokumentti. Päivitetty 2023. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions [25.4.2023]

Pandas. 2023a. NumFOCUS, Inc. WWW-dokumentti. Päivitetty 2023. Saatavissa: https://pandas.pydata.org/ [viitattu 14.3.2023]

Pandas. 2023b. What kind of data does pandas handle?. WWW-dokumentti. Päivitetty 2023. Saatavissa: https://pandas.pydata.org/docs/getting_started/intro_tutorials/01_table_oriented.html [viitattu 20.3.2023]

Pandas. 2023c. How to manipulate textual data?. WWW-dokumentti. Päivitetty 2023. Saatavissa: https://pandas.pydata.org/docs/getting_started/intro_tutorials/10_text_data.html [viitattu 20.3.2023]

Pandas. 2023d. pandas.option_context. WWW-dokumentti. Päivitetty 2023. Saatavissa: https://pandas.pydata.org/docs/reference/api/pandas.option_context.html [viitattu 2.5.2023]

Pip documentation. 2023. pip. WWW-dokumentti. Päivitetty 2023. Saatavissa: https://pip.pypa.io/en/stable/cli/pip_install/ [viitattu 2.5.2023]

Python documentation. 2023a. What is Python? WWW-dokumentti. Päivitetty 2023. Saatavissa: https://docs.python.org/3/faq/general.html [viitattu 8.5.2023]

Python documentation. 2023b. What is Python good for? WWW-dokumentti. Päivitetty 2023. Saatavissa: https://docs.python.org/3/faq/general.html [viitattu 8.5.2023]

Python Basic Tutorial. 2023a. Python - Basic Syntax. WWW-dokumentti. Päivitetty 2023. Saatavissa: https://www.tutorialspoint.com/python/python_basic_syntax.htm [viitattu 8.5.2023]

Python Basic Tutorial. 2023b. Python – Overview. WWW-dokumentti. Päivitetty 2023. Saatavissa: https://www.tutorialspoint.com/python/python_overview.htm [viitattu 8.5.2023]

Wong, J. 2020. String Matching With FuzzyWuzzy. Blogi. Päivitetty 27.10.2020. Saatavissa: https://towardsdatascience.com/string-matching-with-fuzzywuzzy-e982c61f8a84 [viitattu 14.3.2023