DATAN VISUALISOINTITEKNIIKAT PYTHONILLA

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
kevät, 2021

Kalle Sundell

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Hämeenlinnan korkeakoulukeskus

Tekijä Kalle Sundell Vuosi 2021

Työn nimi Datan visualisointitekniikat Pythonilla

Ohjaajat Mirlinda Kosova-Alija

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli luoda ohjekirja datan visualisoinnin aloittamiseen Python-ohjelmointikielellä. Datan käsittely- ja visualisointitaidot ovat hyödyllisiä esimerkiksi it-alalla sekä ylipäätänsä nykypäivän datavetoisessa maailmassa. Opinnäytetyöllä ei ollut erillistä toimeksiantajaa, vaan idea syntyi omasta mielenkiinnosta asiaa kohtaan. Tarkoituksena oli selvittää, miten dataa voidaan visualisoida Python-kielellä hyödyntäen Matplotlib ja Seaborn -kirjastoja.

Opinnäytetyön teoreettisessa osassa paneudutaan datan peruskäsitteisiin, sekä tutustutaan käytettäviin ohjelmistoihin, ohjelmointikieleen ja kirjastoihin. Opinnäytetyö on tyypiltään toiminnallinen. Tämä ilmenee käytännön osassa suoritetuissa visualisointiharjoitteissa. Käytännön osan myötä lukijalle jää mielikuva siitä, miten dataa visualisoidaan käytännössä. Hyvä tasapaino teoria- ja käytännönosan välillä antaa kokonaisvaltaisen kuvan datan visualisointiprosessista.

Lopputuloksissa havaittiin Pythonin erinomainen soveltuvuus datan visualisointitehtäviin. Visualisointien toteuttaminen on suhteellisen helppoa, eikä se vaadi syvää ohjelmointiosaamista. Tämän lisäksi huomattiin, miten paljon vähemmän koodirivejä vaaditaan, kun Matplotlib-kirjaston sijaan käytetään Seaborn-kirjastoa. Seabornin tuottamat visualisoinnit ovat myös tyypillisesti katsojaystävällisempiä, ja ne ovat helpommin kustomoitavissa.

Avainsanat Data, visualisointi, Python, Matplotlib, Seaborn, datatiede, data-analytiikka

Sivut 38 sivua ja liitteitä 1 sivu

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Abstract

Author Kalle Sundell
Year 2021
Subject Data visualization techniques with Python
Supervisors Mirlinda Kosova-Alija

ABSTRACT

The purpose of this thesis was to create a guide for starting data visualization using Python. Data processing and visualization skills are essential in modern working life, considering today's data-driven world. This thesis had no commissioner. The idea for this thesis sparked out of the creator's personal interest regarding the subject. The main purpose of this thesis was to discover how data can be visualized using Python programming language and its vast collection of programming libraries, namely the Matplotlib and Seaborn libraries.

In the theory part of the thesis relevant definitions regarding the subject are introduced. In addition, the software used in the thesis are also introduced in-depth. However, this thesis is of practical type. This becomes especially prevalent in the latter part of the thesis, where real life data is visualized using real life techniques. All in all, a good balance between theory and practicality ensures a good general overview of the subject.

Several conclusions were observed in the end results. First, Python is an excellent language for data visualization. As a high-level language, it does not require one to have a deep understanding of programming to understand the syntax and execute basic tasks. It also has a powerful set of open-source libraries for programmers to take advantage of. It was also observed that while both Matplotlib and Seaborn are powerful visualization tools, the latter consumes less code and in general, the visualization plots created by Seaborn are higher in quality and much more customizable.

Keywords Data, visualization, Python, Matplotlib, Seaborn, data science, data analytics
Pages 38 pages and appendices 1 pages

Sanasto

data	Prosessoimatonta tietoa, mitä on kerätty jostain lähteestä.
datatiede	Suhteellisen uusi tieteenala, missä tutkitaan mm. erilaisia tapoja tehdä datasta päätelmiä.
datan puhdistaminen	Prosessi, missä käsiteltävästä datasetistä poistetaan korruptoitunutta ja/tai sinne kuulumatonta dataa.
funktio	Aliohjelma, joka suorittaa jonkun yksittäisen tehtävän.
parametri	Jokin arvo, minkä funktio voi ottaa vastaan (esim. sukupuoli).
argumentti	Jokin arvo, mikä funktiolle syötetään (esim. mies).
tietorakenne	Auttaa datan tallentamisessa ja käsittelyssä. Tietorakenteita on erilaisia ja paras valinta riippuu käsiteltävästä datasta.
tiedostomuoto	Kuvaa sitä, missä muodossa data esiintyy.
ohjelmointikirjasto	Isompi koodikokonaisuus, mikä sisältää valmiita luokkia ja funktioita, joita voi käyttää eri käyttötarkoituksiin.
ohjelmointikieli	Jokin kieli, minkä avulla voidaan syöttää tietokoneelle ohjeita suoritettavaksi.
DataFrame	Pandas-ohjelmointikirjaston kaksiulotteinen tietorakenne, jota voidaan käyttää mm. csv-tiedostojen käsittelyssä.

Sisällys

1 Johdanto 1
2 Data 2
2.1 Olennaista terminologiaa 2
2.2 Datatyypit 2
2.3 Datan historiaa 3
2.4 Data-analyysiprosessi 4
3 Datan visualisointi 5
3.1 Visualisoinnin motivaatio 6
3.2 Onnistuneen visualisoinnin piirteet 6
4 Opinnäytetyössä hyödynnettävät ohjelmistot 9
4.1 Python-ohjelmointikieli 9
4.2 Pandas-kirjasto 10
4.3 Matplotlib-kirjasto 10
4.4 Seaborn-kirjasto 10
4.5 Anaconda-alusta ja Jupyter Notebook -ympäristö 11
4.6 Datasetti 11
5 Asennukset ja valmistelut 12
5.1 Anaconda 12
5.2 Jupyter Notebook 13
5.3 Datasetin esiprosessointi 15
5.3.1 Datasetin lukeminen read_csv()-funktiolla 16
5.3.2 DataFrame-objektin käsittely 16
5.3.3 Ordinaalimuuttujien vaihtaminen kirjainmuotoon 18
6 Visualisointi Matplotlib-kirjastolla 18
6.1 Pylväskaavio 19
6.2 Pistekaavio 21
6.3 Histogrammi 24
6.4 Laatikko-janakuvio 27
7 Visualisointi Seaborn-kirjastolla 30
7.1 Ydinestimointikaavio 30
7.2 Viulukaavio 34
8 Yhteenveto 37
Lähteet 38

Kuvat, ohjelmakoodit ja taulukot

Kuva 1. Esimerkki nominaalisesta datasta (Donges, 2018). 3
Kuva 2. Data-analyysiprosessin eri vaiheet (Ma et al., 2017). 4
Kuva 3. Esimerkki onnistuneesta visualisoinnista (Snyder & Gamio, 2018). 7
Kuva 4. Anacondan eri asennusvaihtoehdot. 12
Kuva 5. Anaconda Navigator -käyttöliittymä. 13
Kuva 6. Jupyter Notebook -ohjelmointiympäristön etusivu. 14
Kuva 7. Tyhjä projektivihko. 14
Kuva 8. Solujen toiminta havainnollistettuna. 15
Kuva 9. Datasetin turhat sarakkeet. 17
Kuva 10. Siistimpi DataFrame-objekti drop()-funktion jälkeen. 17
Kuva 11. Eri nuottiavainten lukumäärä. 18
Kuva 12. Yksinkertainen pylväskaavio. 20
Kuva 13. Ryhmitelty pylväskaavio. 21
Kuva 14. Esimerkki huonosta pistekaaviosta. 22
Kuva 15. Paranneltu pistekaavio. 23
Kuva 16. Yksinkertainen histogrammi. 25
Kuva 17. Kohennettu histogrammi. 26
Kuva 18. Laatikko-janakuvio havainnollistettuna (Galarnyk, 2018). 27
Kuva 19. Boxplot()-funktiosta syntyvä laatikko-janakuvio. 28
Kuva 20. Yksinkertainen laatikko-janakuvio. 29
Kuva 21. Yksinkertainen ydinestimointikaavio. 31
Kuva 22. Ydinestimointikaavion ytimet ja niiden summa. (Akinshin, 2020). 32
Kuva 23. Ydinestimointikaavio eri bw_adjust-arvoilla. 32
Kuva 24. Kahden muuttujan ydinestimointikaavio. 33
Kuva 25. Kahden muuttujan ydinestimointikaavio parannuksilla. 34
Kuva 26. Viulukaavio havainnollistettuna. 35
Kuva 27. Energiatasojakaumat viulukaaviolla. 36

Ohjelmakoodi 1. Yksinkertainen Python-ohjelma. 9
Ohjelmakoodi 2. Pandas-kirjaston tuonti. 16
Ohjelmakoodi 3. Pandasin read_csv()-funktio. 16
Ohjelmakoodi 4. Nuottiavaimien vaihtaminen numeroista kirjaimiksi. 18
Ohjelmakoodi 5. Matplotlib-kirjaston tuontikomento. 19
Ohjelmakoodi 6. Yksinkertainen pylväskaavio bar()-funktiolla. 19
Ohjelmakoodi 7. Ryhmitellyn pylväskaavion esimerkkikoodi. 20
Ohjelmakoodi 8. Pistekaavion luonti. 22
Ohjelmakoodi 9. Pistekaavion tarkemmat määritykset. 23
Ohjelmakoodi 10. Histogrammin luonti. 24
Ohjelmakoodi 11. Tyylitelty histogrammi. 26
Ohjelmakoodi 12. Matplotlibin boxplot()-funktio 28
Ohjelmakoodi 13. Mukautettu laatikko-janakuvio 29
Ohjelmakoodi 14. Seaborn-kirjaston tuonti projektiin. 30
Ohjelmakoodi 15. Ydinestimointikaavion luonti kdeplot()-funktiolla. 30
Ohjelmakoodi 16. Kahden muuttujan ydinestimointikaavion luonti. 33
Ohjelmakoodi 17. Viulukaavion luontikomento 34
Ohjelmakoodi 18. Viulukaavio usealla muuttujalla. 35

Liitteet

Liite 1 Aineistonhallintasuunnitelma

1 Johdanto

Teknologian kehityksen myötä datan määrä kasvaa vuosi vuodelta. Data on äärimmäisen arvokas voimavara esimerkiksi yrityksille, sillä sitä voidaan hyödyntää muun muassa mainonnassa sekä muussa liiketoiminnan päätöksenteossa. Data ei kuitenkaan ole kovin hyödyllistä, mikäli sitä ei osata käsitellä oikein. On siis löydettävä keinoja, joilla datasta saa jotain irti. Yksi tälläinen keino on datan visualisointi. Datan visualisoinnilla tarkoitetaan tiedon muuntamista sellaiseen graafiseen tai kuvalliseen muotoon, mikä on ihmisen ymmärrettävissä.

Opinnäytetyön tarkoituksena on johdattaa lukija datan visualisointiin sekä teoriassa että käytännössä. Käytännön osassa hyödynnetään Python-ohjelmointikieltä. Python on monikäyttöinen, korkean tason ohjelmointikieli. Syntaksiltaan se on erittäin yksinkertainen ja ymmärrettävissä ilman erityistä ohjelmointitaustaa. Pythonin tarjoamat kirjastot ja tietorakenteet tuovat lähes kaikki tarvittavat työkalut nykypäivän visualisointitehtäviin, ja se onkin yksi suositummista data-analytiikassa käytetyistä ohjelmointikielistä.

Kiinnostus opinnäytetyön ideaan heräsi alun perin Business Analytics and Business Intelligence -moduulin aikana, joka järjestettiin Hämeen ammattikorkeakoulussa syksyllä 2020. Moodulissa opiskelijat johdatettiin muun muassa tilasto- ja datatieteen sekä koneoppimisen perusteisiin. Edellä mainituille taidoille on nykypäivän työmarkkinoilla paljon kysyntää, minkä takia koen opinnäytetyön aiheen olevan sen puolesta ajankohtainen.

Opinnäytetyössä tutustutaan dataan ja sen yhteiskunnallisiin vaikutuksiin, minkä jälkeen käydään teoriatasolla läpi datan visualisointia. Tämän jälkeen tutustutaan opinnäytetyössä käytettäviin menetelmiin, kuten Pythoniin ja sen kirjastoihin sekä Anaconda -alustaan ja Jupyter Notebook -ohjelmointiympäristöön. Käytännön osassa suoritetaan tarvittavat asennukset ja muut toimenpiteet, minkä jälkeen suoritetaan visualisointiharjoitteita. Opinnäytetyön tutkimuskysymykset ovat seuraavat:

- Mitä data on?
- Mitä tarkoittaa datan visualisointi?
- Miten dataa visualisoidaan Python-ohjelmointikielellä?

2 Data

Jotta käytännön osasta saisi mahdollisimman paljon irti, on aluksi hyvä tutustua hieman teoriaan. Tässä luvussa tutustutaan tarkemmin opinnäytetyön kannalta olennaisiin määritelmiin, datan yhteiskunnallisiin vaikutuksiin sekä johdatetaan lukija datan visualisoinnin peruskäsitteisiin.

2.1 Olennaista terminologiaa

Latinan kielen sana **data** on monikkomuoto sanasta datum, mikä tarkoittaa suomeksi sanaa annettu Sanan alkuperäisen määritelmän mukaan datalla tarkoitetaan ”matemaattisten laskutoimitusten pohjatiedoksi annettua faktatietoa.” (Etymonline, n.d.) Arkikielessä termejä tieto ja data pidetään usein synonyymeinä (Brown, n.d.). Todellisuudessa data on kuitenkin vain prosessoimatonta tietoa, mistä ei suoraan voida tehdä mitään johtopäätöksiä (Meleen, n.d.). Sanalla **datasetti** tarkoitetaan datakokoelmaa, joka sisältää useita muuttujia. Suurien datasettien kanssa työskennellessä käy usein ilmi, että ne sisältävät korruptoitunutta tai väärää dataa. **Datan puhdistamisella** tarkoitetaan sitä prosessia, missä tällainen data havaitaan ja sille tehdään kontekstista riippuen tarvittavat toimenpiteet. (Alonso, 2019)

2.2 Datatyypit

Data ilmenee eri muodoissa, eli on olemassa erilaisia **datatyyppejä**. Data jakautuu kahteen päätyyppiin: **numeraaliseen** eli kvantitatiiviseen dataan ja **kategoriseen** eli kvalitatiiviseen dataan. Nämä kaksi jakautuvat jälleen neljään alatyyppiin: **nominaaliseen**, **ordinaaliseen**, **diskreettiin** sekä **jatkuvaan** dataan. Datatyypit ovat olennainen osa data-analyysiä, ja ne tulee ymmärtää hyvin. Esimerkiksi kategorisen ja jatkuvan datan analysointiin on erilaiset lähestymistavat. (Donges, 2018)

Numeraalinen data jaetaan diskreettiin ja jatkuvaan dataan. Diskreetti data on mitä tahansa laskettavissa olevaa dataa. Kivelä et al. (2000) toteavat, että data on diskreettiä, kun ”vain tietyt arvot tulevat kysymykseen”. Arvosana-asteikolla 4–10 on käytössä vain seitsemän eri arvoa, eikä esimerkiksi arvosanaa 12 voida antaa. Jatkuva data on dataa, mitä voidaan mitata. Hyviä esimerkkejä jatkuvasta datasta ovat ihmisen pituus tai lämpötila.

Kategorinen data esittää jotain kuvailevaa, kuten paikkakuntaa tai sukupuolta. Vaikka kategorinen data on usein tekstimuodossa, se voidaan myös ilmaista numeraalisesti. Näillä ei kuitenkaan ole mitään matemaattista arvoa. Kategorinen data jakautuu nominaaliseen ja ordinaaliseen dataan. Nominaalinen data kuvaa muuttujien nimiä. Nominaalisella datalla ei voi suorittaa matemaattisia operaatioita. (Donges, 2018) Esimerkiksi paikkakunnista Porvoo, Joensuu, Helsinki, Imatra ei voida laskea keskiarvoa. Kuva 1 sisältää nominaalista dataa.

Kuva 1. Esimerkki nominaalisesta datasta (Donges, 2018).

Ordinaalinen data ei eroa nominaalisesta datasta muuten kuin siten, että se on tietyssä järjestyksessä. Ordinaalista dataa voidaan käyttää esimerkiksi asiakastyytyväisyyden mittaamisessa, missä 1 kuvaa tunnetta ”erittäin tyytymätön” ja 5 tunnetta ”erittäin tyytyväinen”. (Donges, 2018)

2.3 Datan historiaa

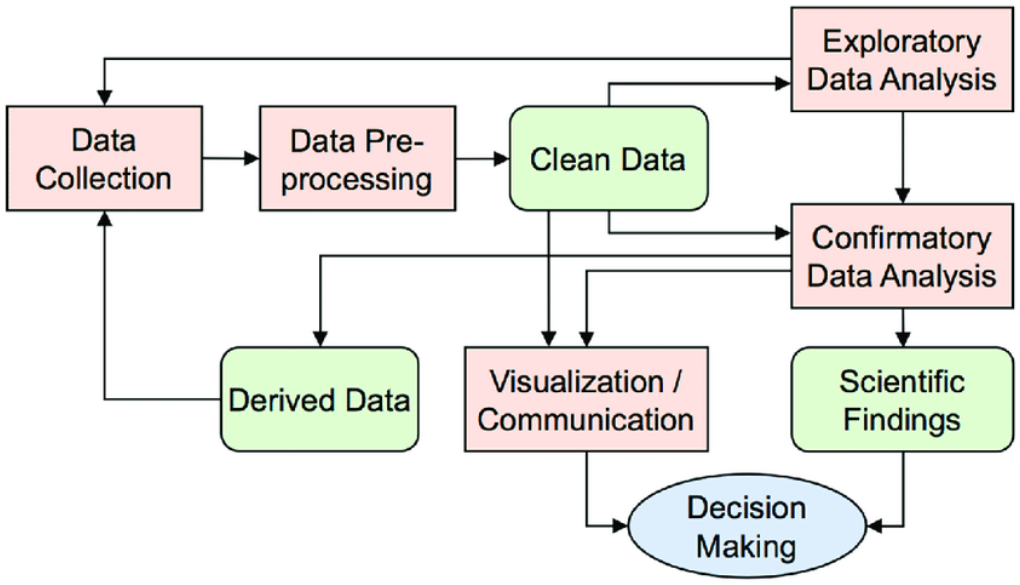
Ihmisillä on kautta aikojen ollut tarve kerätä ja säästää hyödyllistä tietoa. Esihistoriallisina aikoina dataa varastoitiin muun muassa kaivertamalla merkintöjä tikkuihin ja luihin (Marr, 2015). Tiedon määrän kasvaessa kehitettiin kirjastot. Nykykäsityksen mukaan ensimmäinen kirjasto pystytettiin noin vuonna 700 eaa. nykyisen Irakin alueelle (Andrews, 2016). Kenties ensimmäinen oikea data-analyysikokeilu suoritettiin lontoolaisen John Grauntin toimesta vuonna 1663. Kuolintietoja tutkimalla Graunt arveli pystyvänsä kehittämään paiseruttoa ehkäisevän varoitusjärjestelmän. (Marr, 2015)

1940-luvulla yhdysvaltalainen kirjastonhoitaja Fremont Rider havahtui kasvavaan ongelmaan: tietoa syntyi liikaa. Riderin arvioiden mukaan kirjastojen koko tuplaantui kuudentoista vuoden välein. Vuonna 1941 käytettiin ensimmäisen kerran termiä **tietoräjähdys**. (Press, 2013) Käsitteellä **big data** tai **massadata** viitataan ilmiöön missä datan määrä on niin suurta tai monimutkaista, että sen analysointi tai muu käyttö tuottaa merkittäviä haasteita (Press, 2014). Data-alan markkinajätti IDC ennustaa, että vuoteen 2025 mennessä maailmassa olisi yhteensä 175 zettabittiä dataa, kun vuonna 2013 vastaava luku oli 4,4 (Khvoynitskaya, 2020). Datan määrä maailmassa kasvaa räjähdysmäistä vauhtia. Vaikka massadata usein mielletään negatiivisena asiana, siinä on myös hyvät puolensa. Suurien datamäärien avulla voidaan tutkia asiakkaiden käyttäytymistä tai parantaa esimerkiksi asiakaspalvelun tasoa (Ciklum, 2019).

2.4 Data-analyysiprosessi

Kun dataa on paljon, herää kysymys miten suuresta datamäärästä saataisiin kaikki mahdollinen hyöty irti. Data on arvokas resurssi. Se auttaa päätöksenteossa ja oikein käsiteltynä antaa vastauksen moniin kysymyksiin. **Data-analyysillä** kuvataan prosessia, missä dataa tutkitaan ja käsitellään niin, että määriteltyihin tutkimuskysymyksiin saadaan vastaus (*Statistics Canada Quality Guidelines*, 2009). Koko data-analyysiprosessi jakautuu useaan vaiheeseen, joita on havainnollisettu kuvassa 2.

Kuva 2. Data-analyysiprosessin eri vaiheet (Ma et al., 2017).



Prosessin päävaiheet ovat keräys, esiprosessointi, puhdistaminen, analysointi, ja kommunikointi. Keräysvaiheessa määritellään mitä dataa halutaan kerätä, minkä jälkeen suoritetaan itse datan keruu. Dataa voidaan kerätä eri menetelmin, kuten haastattelemalla, tekemällä kyselyitä, tai keräämällä kamera- tai sensoridataa. Yritykset voivat kerätä dataa esimerkiksi asiakkaiden ostotapahtumista. (Nantasenamat, 2020)

Kun haluttu data on kerätty, siirrytään esiprosessointivaiheeseen. Datan esiprosessointi on olennainen osa koko prosessia. Esiprosessointivaiheessa kerättyyn dataan tutustutaan ensimmäisen kerran tarkemmin. Ensimmäiseksi varmistetaan, että data on oikeassa muodossa, ja että se on kerätty oikein. Tämän jälkeen data organisoidaan tarvittaessa ymmärrettävään muotoon. Kun esiprosessointi on suoritettu, data puhdistetaan. Kuten alaluvussa 2.1. mainittiin, datan puhdistamisella tarkoitetaan väärän ja korruptoituneen tiedon poistamista kerätystä datasta. Esiprosessointi ja puhdistaminen ovat pitkälti samaa prosessia. (Nantasenamat, 2020)

Puhdistamisen jälkeen kerätty data analysoidaan. Datasta pyritään löytämään muun muassa virheitä, poikkeamia, muuttujien välisiä suhteita sekä muita ominaisuuksia hyödyntämällä esimerkiksi datan visualisointia ja kuvailevaa tilastoanalyysia (Seltman, 2018).

3 Datan visualisointi

Visualisointi on olennainen osa koko data-analyysiprosessia. Sitä hyödynnetään niin eksploratiivisessa data-analyysissä kuin tulosten kommunikoinnissa. Datan visualisoinnilla tarkoitetaan tiedon esittämistä graafisessa muodossa. Datan visualisoinnissa hyödynnetään visuaalisia elementtejä kuten pisteitä, viivoja, kaavioita ja karttoja, joiden avulla pyritään tekemään johtopäätöksiä. (Tableau Software, 2018) Datan visualisoinnin varhaisesta kehityksestä ei ole tarkkaa tietoa, ja tieto vaihtelee lähteittäin. Yksi ensimmäisiä oikeita graafisia esityksiä lienee kuitenkin 900-luvulta peräisin oleva aikasarjakaavio, joka kuvaa seitsemän suurimman planeetan liikkeitä taivaalla. 1600-luvulla René Descartesin ja Pierre de Fermat’n tekemä työ analyyttisen geometrian ja koordinaatiston eteen oli suuri edistysaskel tilastotieteelle ja sitä myötä tiedon esittämiselle. (Chen et al., 2008)

1800-luvun alkupuolta pidetään modernin visualisoinnin syntyhetkenä. Tällöin kehitettiin muun muassa piste-, linja- ja pylväskaaviot sekä histogrammit. Kaikki nämä menetelmät ovat käytössä edelleen tänä päivänä, ja niihin tutustutaan tarkemmin käytännön osassa. Tietokoneiden ja ohjelmointikielten kehitys mullisti data-analyysin ja datan visualisoinnin. 1900-luvun loppuun mennessä datan visualisoinnista oli tullut monitieteellinen tutkimusala. (Chen et al., 2008)

3.1 Visualisoinnin motivaatio

Nykypäivän datavetoisessa yhteiskunnassa on vaikea ajatella sellaista alaa, mikä ei hyötyisi datan visualisoinnista. Uusia datarivejä syntyy päivittäin miljardeja lisää. Kun datan määrä kasvaa, sen olemuksesta ja siitä tehdyistä johtopäätöksistä tulee myös väistämättä monimutkaisempia. (Tableau Software, 2018) Tiedon esittäminen visuaalisessa muodossa on erittäin tehokas keino esittää monimutkaisia konsepteja. Tämä perustuu ihmisten aivotoimintaan. Ihmisen näkö- ja ajattelukyky hyödyntävät eri osia aivoista. Näkökykymme on tyypillisesti nopea ja tehokas – se siis ikään kuin antaa lisätehoa aivoillemme käsitellä tietoa. Toisin sanoen ihmiset käyttävät enemmän aivokapasiteettia tiedon sisäistämiseen, kun se on esitetty visuaalisessa muodossa. (Treehouse Technology Group, n.d.)

3.2 Onnistuneen visualisoinnin piirteet

Datan visualisoinnin tärkein päämäärä on välittää tietoa lukijalle selkeästi ja tehokkaasti. Data ei saa olla esitettynä liian vaikeasti, vaan siitä vedetyt johtopäätökset tulee ilmetä vaivatta. Onnistunut visualisointi tarvitsee osaavan suunnittelijan. Ongelmana on usein tasapainon löytäminen estetiikan ja toimivuuden välillä. Kun estetiikkaan panostetaan liikaa, visualisoinnin tärkein päämäärä – tiedon välittäminen – unohtuu helposti. Onnistuneessa datan visualisoinnissa yhdistyy kolme pääperiaatetta: kohdeyleisön tunteminen, selkeä esitysmuoto ja tiedon tehokas välittyminen. (Friedman, 2008)

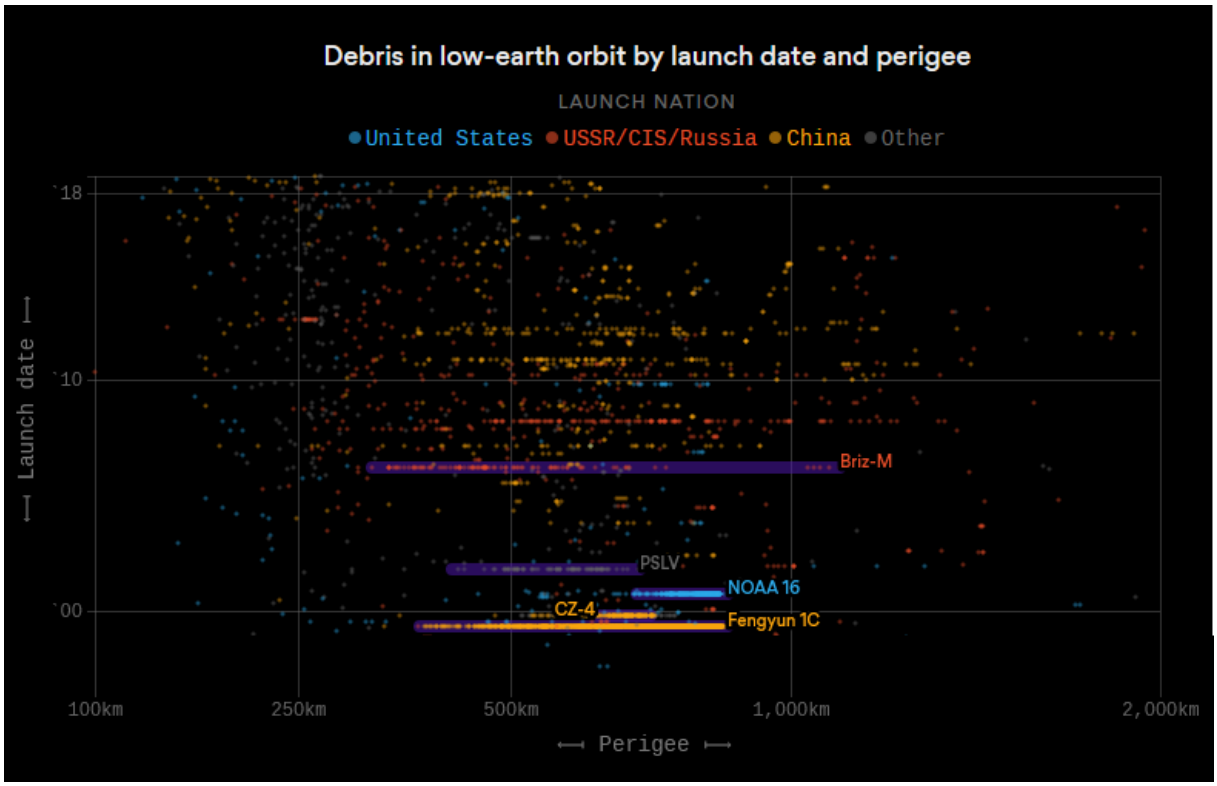
Visualisointia laatiessa on olennaista tietää, kenelle visualisointia tekee. Onko kohdeyleisö ala-asteluokka vai kenties joukko tilastotieteen ammattilaisia? Millaista terminologiaa on sopiva käyttää esityksessä? Mihin kysymykseen kohdeyleisö haluaa saada vastauksen? Miten esitetystä datasta voidaan tehdä johtopäätöksiä, ja miten sitä voidaan hyödyntää

päätöksenteossa? (Stikeleather, 2013) Esimerkiksi edellä mainittuja kysymyksiä on hyvä pohtia ennen prosessin aloittamista.

Visualisoinnin laatijan on varmistettava, että esitys on selkeä. Kaikkien esityksen tulkitsijoiden tulee olla yhtä mieltä siitä, mitä visualisoinnissa esitetään. Esityksen semantiikan täytyy siis olla johdonmukaista; esimerkiksi kaikki muuttujat tulee nimetä oikein ja jokaisella esityksessä esiintyvällä asialla tulee olla jokin tarkoitus. (Stikeleather, 2013) Englannin kielen termillä **chart junk** tarkoitetaan kaikkea sellaista, mikä ei tuo esitykseen mitään lisäarvoa, vaan lähinnä vaikeuttaa sen ymmärtämistä (Tufte, 1983, s. 107).

Stikeleatherin (2013) mukaan visualisointi on onnistunut, kun se on mukaansatempaava ja sen tieto välittyy vakuuttavasti ja tarinanmukaisesti. Jotta tieto välittyy vakuuttavasti, tulee visualisoinnin laatijalla olla hyvä käsitys käsiteltävän datasetin sisällöstä. Datan visualisointi toimii erinomaisena tarinankerrontatyökaluna etenkin tilanteissa, missä data on monimutkaista. Tämä johtuu siitä, että ihmisen näkökyky auttaa tulkitsemaan sellaisia konsepteja, mitkä olisivat muuten haastavia sisäistää. Kuvassa 3 esitetään satelliittien muodostaman avaruusjätteen määrää maittain.

Kuva 3. Esimerkki onnistuneesta visualisoinnista (Snyder & Gamio, 2018).



Maiden värit eroavat selkeästi toisistaan. Esityksessä ei ole turhaa tekstiä, ja jokaisella merkinnällä on selkeä tarkoitus. Vertikaalinen akseli kuvaa satelliitin laukaisuvuotta. Horisontaalinen akseli kuvaa sitä korkeutta, missä satelliitti on lähimpänä maan pintaa sen kiertoradalla.

4 Opinnäytetyössä hyödynnettävät ohjelmistot

Nykypäivän data-analyysihaasteisiin tarvitaan nykyaikaiset välineet. Mitä käytettäviin teknologioihin tulee, valinnan varaa on paljon – niin maksullisia kuin maksuttomia. Valinta tulee tehdä sen perusteella mitä on tekemässä. Tutkimustyössä voidaan esimerkiksi hyödyntää maksullista mutta tehokasta MATLABia, kun taas jokapäiväiseen ohjelmointiin soveltuu paremmin ilmainen Python. Tässä luvussa tutustutaan opinnäytetyössä käytettävään välineistöön. Kaikki opinnäytetyössä käytettävät välineet (alusta, ohjelmointikieli ja kirjastot) ovat ilmaisia ja perustuvat avoimeen lähdekoodiin, eli ne ovat kenen tahansa ladattavissa ja muokattavissa.

4.1 Python-ohjelmointikieli

Python on hollantilaisen Guido van Rossumin kehittämä ohjelmointikieli. Van Rossum aloitti kielen kehittämisen jouluna 1989, ja ensimmäinen versio siitä julkaistiin helmikuussa 1991. Python on yleiskäyttöinen, korkean tason ohjelmointikieli, eli sitä voidaan käyttää lähes mihin tahansa käyttötarkoitukseen. (Python Software Foundation, 2021a) Ohjelmointikielen taso kuvaa sen syntaksin abstraktiotasoa. Toisin sanoen, korkean tason ohjelmointikielet vastaavat pitkälti ihmisen ymmärtämää kieltä. Korkean tason kielet eivät ole parempia kuin matalan tason kielet, vaan ne ovat vain helpommin luettavissa. Matalan tason kieliä kutsutaan matalaksi siksi, koska ne muistuttavat pitkälti konekieltä. (Beal, n.d.) Ohjelmakoodissa 1 on esimerkki Python-ohjelmasta.

Ohjelmakoodi 1. Yksinkertainen Python-ohjelma.

```
numerot = [2, 5, 7, 12, 10]

print("Printataan for-silmukan avulla listassa esiintyvät numerot")

for numero in numerot:
    print(numero)
```

Python soveltuu erinomaisesti datan analysointi- ja visualisointitehtäviin. Pythonin valttikortteja ovat sen luettavuus sekä kattavat kirjastot, mitkä mahdollistavat muun muassa datan tehokkaan käsittelyn ja visualisoinnin. (Thumar, 2019) Alaluvuissa 4.2., 4.3. ja 4.4. tutustutaan tässä opinnäytetyössä käytettäviin Python-kirjastoihin.

4.2 Pandas-kirjasto

Pandas on yhdysvaltalaisen Wes McKinneyn kehittämä kirjasto Python-kielelle, joka perustuu tunnettuun NumPy-kirjastoon. Pandas tarjoaa käyttäjälle tehokkaita ja joustavia datan käsittelytyökaluja. Sen kaksi päätietorakennetta – **DataFrame** ja **Series** – soveltuvat erinomaisesti finanssialan, tilastotieteen ja muiden insinöörialojen tyypillisiin analysointitehtäviin. Pandasilla voi myös visualisoida dataa, mutta tässä opinnäytetyössä visualisointi suoritetaan Matplotlib-kirjastoa käyttäen. Pandas-kirjaston kehittäjien tavoite on tehdä siitä kaikista tehokkain avoimeen lähdekoodiin perustuva datan käsittelytyökalu. (Python Software Foundation, 2021b)

4.3 Matplotlib-kirjasto

Opinnäytetyössä käytettävä Matplotlib-kirjasto on yksi Pythonin datan visualisointityökaluista. Se tarjoaa käyttäjälle mahdollisuuden luoda staattisia, liikkuvia sekä vuorovaikutteisia visualisointiratkaisuja Python-kielellä (Hunter, 2007). Yhdysvaltalainen neurobiologi John Hunter laittoi projektin aluilleen vuonna 2002. Se kehitettiin aluksi IPython-komentotulkin lisäosaksi, ja sen tarkoitus oli tarjota mahdollisuus muodostaa interaktiivisia, MATLAB-tyylisiä kaavioita. Lopulta siitä kuitenkin muodostui oma kirjasto, ja versio 0.1 julkaistiin vuonna 2003. (VanderPlas, 2013)

Heti julkaisun jälkeen kirjasto keräsi laajaa suosiota esimerkiksi tähtitieteen alalla. Kirjaston menestyksen taustalla piilee lukuisia seikkoja. Ensinnäkin, Hunter itse oli intohimoinen Matplotlibin puolestapuhuja ja mainostaja. Toiseksi, kirjastoa voidaan käyttää kaikissa käyttöjärjestelmissä mitkä vain tukevat Pythonia. Lisäksi sen MATLAB-tyylinen käyttöliittymä oli monen mieleen. (VanderPlas, 2013)

4.4 Seaborn-kirjasto

Seaborn on Matplotlibiin perustuva kirjasto. Seaborn pyrkii tarjoamaan käyttäjälleen mahdollisuuden luoda kauniimpia ja kehittyneempiä kaavioita kuin Matplotlib. Sitä voidaan siis pitää ikään kuin Matplotlibin lisäosana. Seabornin syntaksi on selkeämpää ja kaavioiden

tyylittely on helpompaa. Lisäksi Seaborn sopii paremmin Pandasin DataFrame ja Series -tietorakenteiden käsittelyyn. (Misal, 2019)

4.5 Anaconda-alusta ja Jupyter Notebook -ympäristö

Anaconda on data-analytiikassa käytettävä alusta. Anacondaan sisältyy muun muassa erilaisia komentotulkkeja, ohjelmointiympäristöjä ja visualisointityökaluja. Alustalla on laaja yhteisöllinen tuki, ja sille on saatavissa lukuisia avoimeen lähdekoodiin perustuvia lisäpakkauksia. Alustasta on olemassa sekä ilmainen että maksullinen versio. Se on lähtökohtaisesti ilmainen, mutta tietyt lisäominaisuudet ovat maksullisia. Toisin sanoen alusta perustuu niin kutsuttuun freemium-lisenssiin. (Anaconda, 2021) Yksi Anaconda-alustan sisäänrakennetuista ohjelmistoista on verkkopohjainen ohjelmointiympäristö Jupyter Notebook, jolla voi ohjelmoida muun muassa data-analytiikassa suosituilla Python ja R -ohjelmointikielillä. Pelkän ohjelmointikoodin lisäksi sillä voi kirjoittaa tavallista tekstiä, matemaattisia yhtälöitä ja esittää visualisointeja. Jupyterin projektitiedostoja kutsutaan nimellä notebook eli vihko. Jupyter Notebook on yksi useista Project Jupyter -säätiön kehittämistä ohjelmistoista. (Project Jupyter, 2021)

4.6 Datasetti

Opinnäytetyössä käytettävä datasetti on noudettu kaggle.com-sivustolta. Kagglen nettisivuilta löytyy yli 40 000 julkista datasettiä, joita voi käyttää vapaasti esimerkiksi visualisointi- tai koneoppimisharjoitteissa (Kaggle, n.d.). Käytännön osassa käytettävä datasetti sisältää tietoa 42 305:sta Spotify-musiikinkuuntelualustalla olevasta kappaleesta. Jokaisesta kappaleesta on mitattu muun muassa niiden energia-, äänenvoimakkuus- ja akustiikkatasot sekä monia muita piirteitä. Datasetti perustuu CC0 Public Domain -lisenssiin, eli sen muokkaaminen ja uudelleenkäyttäminen on sallittua (Creative Commons, n.d.).

5 Asennukset ja valmistelut

Tässä luvussa suoritetaan visualisointia edeltävät toimenpiteet. Onnistuneen visualisoinnin edellytyksenä on hyvä valmistautuminen. Aluksi asennetaan tarvittavat ohjelmistot, minkä jälkeen datalle suoritetaan pienimuotoista puhdistamista visualisointia varten. Opinnäytetyössä käytettävän datasetin saa ladattua ilmaiseksi osoitteesta https://www.kaggle.com/mrmorj/dataset-of-songs-in-spotify.

5.1 Anaconda

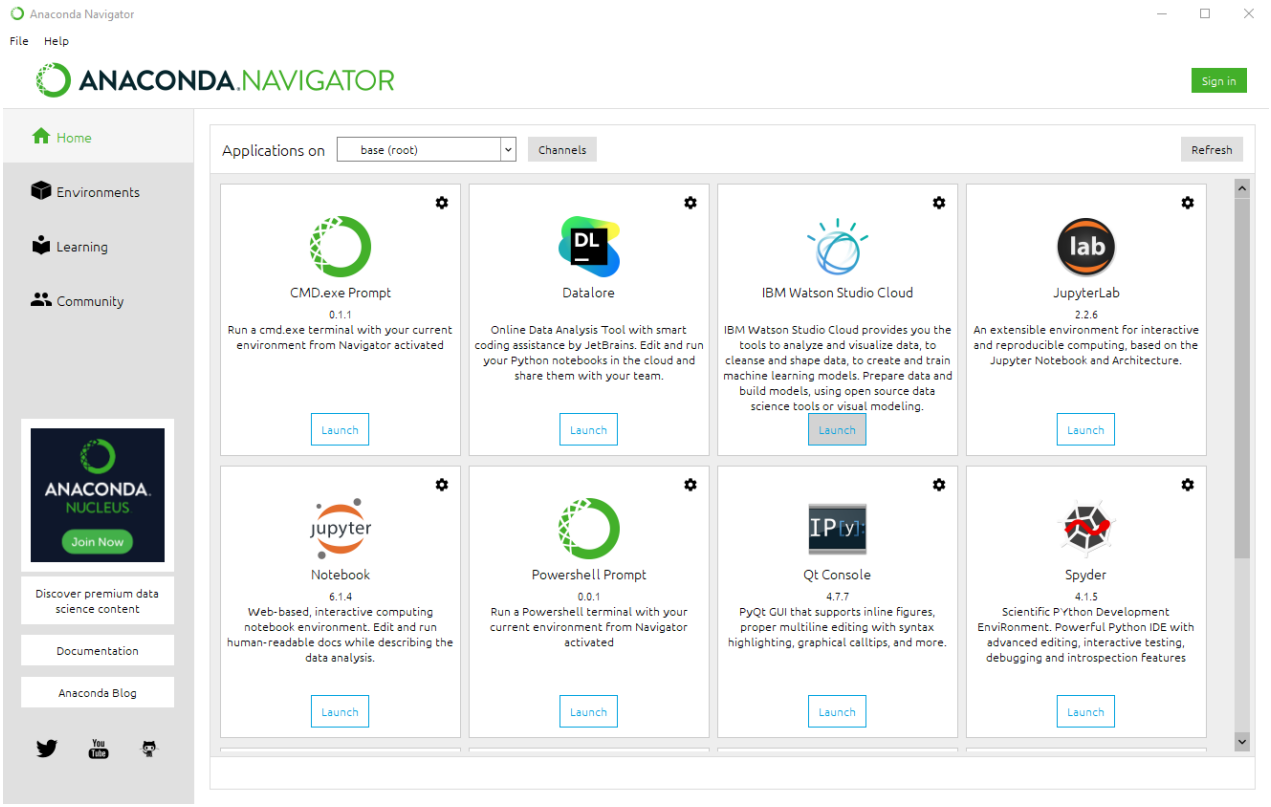
Opinnäytetyön käytännön osuus aloitetaan asentamalla Anaconda-alusta. Alustan saa ladattua osoitteesta https://www.anaconda.com/products/individual. Kyseessä on ilmainen **Individual Edition**, joka soveltuu opinnäytetyöhön erinomaisesti. Kuva 4 on kuvakaappaus sivun alareunasta, mistä löytyy eri asennusvaihtoehtoja käyttöjärjestelmästä sekä suorittimen bittisyydestä riippuen.

Kuva 4. Anacondan eri asennusvaihtoehdot.

Anacondan mukana tulee valtava määrä automaattisesti asennettuja lisäpakkauksia. Näistä opinnäytetyön kannalta olennaisimmat ovat Anaconda Navigator -käyttöliittymä, Jupyter Notebook -ohjelmointiympäristö sekä Python-ohjelmointikieli ja sen lukuiset lisäkirjastot. Yksi Anacondan hyvistä puolista on sen nopea käyttöönotto – alustan asennuksen jälkeen

juuri mitään lisäasennuksia ei tarvitse tehdä. Kun Anaconda on asennettu tietokoneelle, käynnistetään kuvassa 5 Anaconda Navigator -käyttöliittymä.

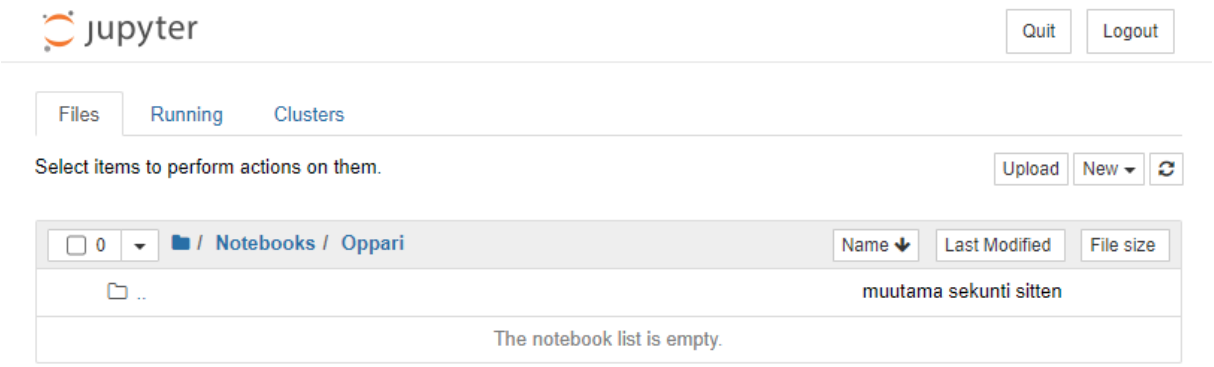
Kuva 5. Anaconda Navigator -käyttöliittymä.



5.2 Jupyter Notebook

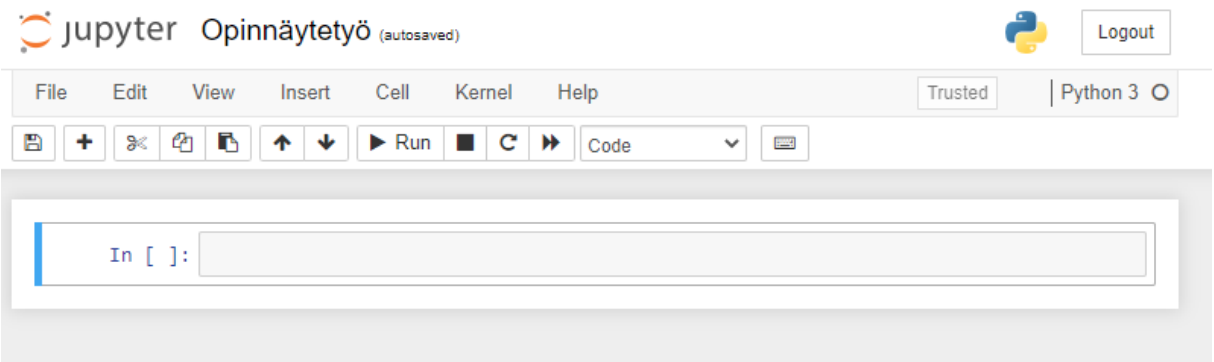
Kuten kuvasta 5 huomaa, Anaconda Navigator tarjoaa käyttäjäystävällisen käyttöliittymän, mitä kautta eri ohjelmistoihin pääsee vaivatta käsiksi. Ohjelmistoja ja pakkauksia saa tarvittaessa ladattua lisää ja niitä voidaan asentaa eri ympäristöille mikäli käyttäjiä tai käyttötarkoituksia on useita. Jupyter Notebook -ohjelmointiympäristön saa avattua painamalla sen alapuolella olevaa Launch-painiketta. Jupyter Notebook avautuu internet-selaimessa, ja sen käyttöliittymä on havainnollistettuna kuvassa 6.

Kuva 6. Jupyter Notebook -ohjelmointiympäristön etusivu.



Uutta projektia varten täytyy luoda uusi kansio. Uusi kansio luodaan painamalla oikealla olevaa **New**-painiketta, minkä jälkeen valitaan **Folder**. Tässä tapauksessa projektikansio sijaitsee osoitteessa **C:\Users\kayttajanimi\Notebooks\Oppari**. Kansion sijainti ja nimi voi olla mikä tahansa, mutta projektissa käsiteltävien tiedostojen tulisi lähtökohtaisesti sijaita samassa osoitteessa. Ennen kuin luodaan itse projektivihko, pitää käsiteltävä datasetti siirtää projektikansioon. Kaggle.com-sivustolta ladattu .csv-tiedosto tulee siirtää luotuun projektikansioon. Kun datasetti on siirretty kansioon, luodaan uusi tyhjä projektivihko painamalla jälleen **New**-painiketta, minkä jälkeen valitaan **Python 3**. Uusi projektivihko aukeaa välittömästi luomisen jälkeen. Projektivihkon saa halutessaan uudelleennimettyä kaksoisklikkaamalla sen vasemmassa yläkulmassa sijaitsevaa nimeä, mikä on oletuksena **Untitled**. Kuva 7 sisältää kuvakaappauksen projektivihkosta.

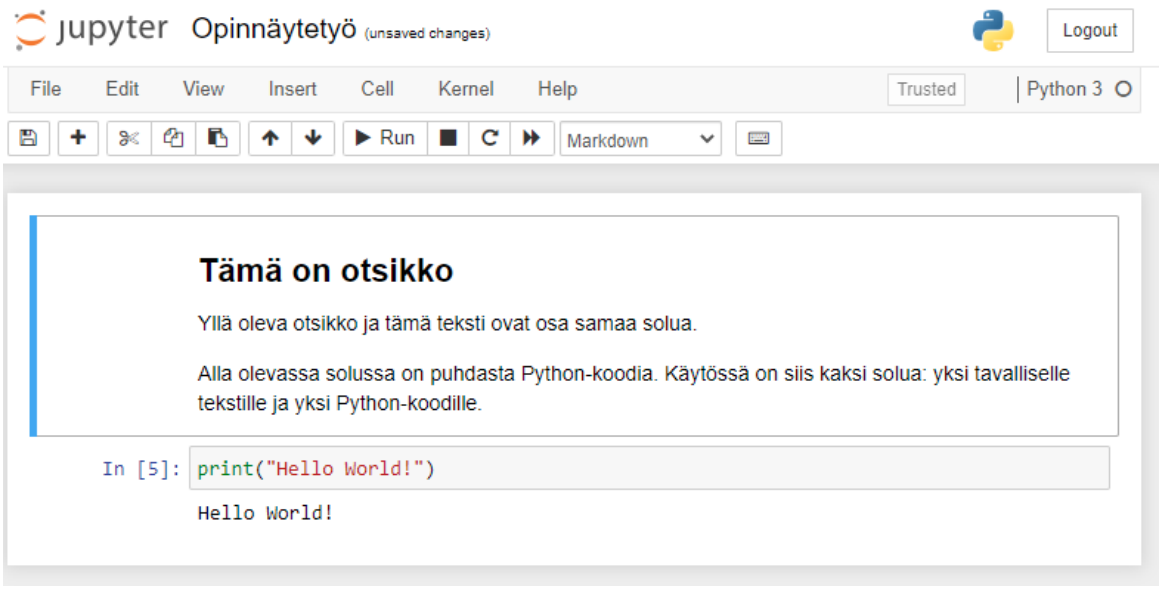
Kuva 7. Tyhjä projektivihko.



Projektivihkot koostuvat niin kutsutuista soluista (cell). Kuvassa 8 havainnollistetaan solujen toimintaa. Soluihin voidaan kirjoittaa tässä tapauksessa joko Python-koodia tai tekstiä.

Yhdessä solussa voi olla yksi tai useampi rivi koodia tai tekstiä. Työkalupalkin pudotusvalikosta voidaan valita, kirjoitetaanko valittuun soluun koodia (code) vai tavallista tekstiä (markdown). Tavallinen teksti ja koodi tarvitsevat omat solunsa, eli niitä ei voi kirjoittaa sekaisin yhteen soluun. Yksittäinen solu ajetaan painamalla Shift+Enter -näppäinyhdistelmää. Projekteihin kertyy kuitenkin aina useita, jopa satoja soluja. Kaikki projektin solut voidaan ajaa järjestyksessä valitsemalla työkalupalkista välilehti **Cell** ja painamalla sieltä **Run All**. Näin jokaista solua ei tarvitse ajaa erikseen. Tämä on hyödyllistä silloin kun käsitellään isoja projekteja. Solun voi poistaa klikkaamalla solua, minkä jälkeen työkalupalkista avataan välilehti **Edit** ja valitaan **Delete Cells**.

Kuva 8. Solujen toiminta havainnollistettuna.



5.3 Datasetin esiprosessointi

Alaluvussa 2.4. käytiin lävitse data-analyysiprosessia. Todettiin, että ennen kuin dataa voidaan analysoida, se tulee putsata. Tässä alaluvussa datasetti esiprosessoidaan ja tutustutaan ensimmäisen kerran konkreettisesti käytettävään datasettiin. Datan putsaaminen ja tutkiminen suoritetaan hyödyntämällä Pandas-kirjastoa. Koska Pandas tulee Anaconda-asennuksen mukana, kirjastoa ei tarvitse erikseen asentaa vaan sen saa käyttöön kätevästi ohjelmakoodissa 2 esiintyvällä koodirivillä.

Ohjelmakoodi 2. Pandas-kirjaston tuonti.

```
import pandas as pd
```

Pythonin `import`-komentoa käytetään haluttujen moduulien tai pakettien tuontiin. Ohjelmakoodissa 2 oleva koodi kertoo siis Pythonille, että Pandas-kirjastoa halutaan käyttää projektissa. Komento `as` on valinnainen, ja sille annettu arvo voi olla mikä tahansa. Komennon idea on antaa tuodulle moduulille jokin helpompi nimi niin, ettei kirjaston koko nimeä tarvitse aina käyttää, kun sitä kutsutaan myöhemmin koodissa.

5.3.1 Datasetin lukeminen read_csv()-funktiolla

Alaluvussa 5.2. käytettävä datasetti siirrettiin samaan projektikansioon, missä projektivihko sijaitsee. Datasetti on tiedostotyypiltään .csv-tiedosto, mikä on yleinen formaatti dataseteille. Pandas tarjoaakin hyödyllisen `read_csv()`-funktion, millä datasetin sisältö voidaan lukea Pandasin DataFrame-tietorakenteeseen, mitä kautta siihen päästään helposti käsiksi. Ohjelmakoodissa 3 on esitetty `read_csv()`-funktion toimintatapa. Mikäli datasetti ei sijaitse projektikansiossa, tulee tiedoston nimen lisäksi syöttää sen sijainti tietokoneella.

Ohjelmakoodi 3. Pandasin read_csv()-funktio.

```
muuttujan_nimi = pandas.read_csv('tiedoston_nimi')
```

5.3.2 DataFrame-objektin käsittely

DataFrame-objektiin, eli toisin sanoen käytettävän datasetin sisältöön, voi tutustua käyttämällä Pandasin `head()`-funktiota. Oletuksena `head()` palauttaa datasetin viisi ensimmäistä riviä. Nopeasti katsottuna sisältö näyttää hyvältä ja johdonmukaiselta. Kun sarakkeisiin tutustutaan tarkemmin, voidaan kuitenkin todeta, että se sisältää opinnäytetyön kannalta turhia sarakkeita. Kuvassa 9 havaitaan turhat sarakkeet.

Kuva 9. Datasetin turhat sarakkeet.

```
In [7]: df.head()
```

Out[7]:

uri	track_href	analysis_url	duration_ms	time_signature	genre	song_name	Unnamed: 0	title
9gD9q343XFRKx	https://api.spotify.com/v1/tracks/2Vc6NJ9PW9gD...	https://api.spotify.com/v1/audio-analysis/2Vc6...	124539	4	Dark Trap	Mercury: Retrograde	NaN	NaN
'mnL7uGHmRj6p	https://api.spotify.com/v1/tracks/7pgJBLVz5Vmn...	https://api.spotify.com/v1/audio-analysis/7pgJ...	224427	4	Dark Trap	Pathology	NaN	NaN
0WCGeNmuNhy	https://api.spotify.com/v1/tracks/0vSWgAlfpye0...	https://api.spotify.com/v1/audio-analysis/0vSW...	98821	4	Dark Trap	Symbiote	NaN	NaN
wuH2ei1nOQ1nu	https://api.spotify.com/v1/tracks/0VSXnJqQkwuH...	https://api.spotify.com/v1/audio-analysis/0VSX...	123661	3	Dark Trap	ProductOfDrugs (Prod. The Virus and Antidote)	NaN	NaN
TlbMmPHuO7S3	https://api.spotify.com/v1/tracks/4jCeguq9rMTl...	https://api.spotify.com/v1/audio-analysis/4jCe...	123298	4	Dark Trap	Venom	NaN	NaN

```
In [9]: df.columns
Out[9]: Index(['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
               'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo',
               'type', 'id', 'uri', 'track_href', 'analysis_url', 'duration_ms',
               'time_signature', 'genre', 'song_name', 'Unnamed: 0', 'title'],
              dtype='object')
```

Datasetin kaikki sarakkeet saadaan näkyviin hyödyntämällä DataFrame-objektin `columns`-attribuuttia. Kuvassa 9 turhat sarakkeet ovat merkitty punaisella. Turhien sarakkeiden poisto datasetistä onnistuu vaivattomasti `drop()`-funktiolla. Kuvassa 10 vanha DataFrame-objekti yli kirjoitetaan uudella versiolla, missä turhia sarakkeita ei ole.

Kuva 10. Siistimpi DataFrame-objekti drop()-funktion jälkeen.

```
In [5]: df = df.drop(columns=['type', 'id', 'uri', 'track_href', 'analysis_url', 'song_name', 'Unnamed: 0', 'title'])
In [6]: df.head()
```

Out[6]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	genre
0	0.831	0.814	2	-7.364	1	0.4200	0.0598	0.013400	0.0556	0.3890	156.985	124539	4	Dark Trap
1	0.719	0.493	8	-7.230	1	0.0794	0.4010	0.000000	0.1180	0.1240	115.080	224427	4	Dark Trap
2	0.850	0.893	5	-4.783	1	0.0623	0.0138	0.000004	0.3720	0.0391	218.050	98821	4	Dark Trap
3	0.476	0.781	0	-4.710	1	0.1030	0.0237	0.000000	0.1140	0.1750	186.948	123661	3	Dark Trap
4	0.798	0.624	2	-7.668	1	0.2930	0.2170	0.000000	0.1660	0.5910	147.988	123298	4	Dark Trap

Funktiolle `drop()` syötetään listamuodossa kaikki ne sarakkeet, jotka halutaan poistaa. Turhien sarakkeiden poistaminen on tärkeää, koska se tekee datasetistä luettavamman ja tehokkaamman käsitellä. Kun dataa on paljon, turhat sarakkeet hidastavat esimerkiksi funktioiden käsittelynopeutta. Nyt jäljellä on enää sarakkeita, jotka ovat mahdollisesti visualisoinnin kannalta olennaisia.

5.3.3 Ordinaalimuuttujien vaihtaminen kirjainmuotoon

Datasetin sarake **key** sisältää dataa kappaleiden nuottiavaimista. Nuottiavaimet ovat merkitty numeroin 0-11, missä esimerkiksi numero 0 vastaa C-nuottia, numero 1 C#-nuottia, numero 2 D-nuottia ja niin edelleen. Voimme tutustua tarkemmin sarakkeen sisältöön Pandasin `value_counts()`-funktion avulla, minkä toimintaa on esitetty kuvassa 11.

Kuva 11. Eri nuottiavainten lukumäärä.

```
In [10]: df['key'].value_counts()

Out[10]: 1     7537
         7     4275
         11    4150
         6     3714
         0     3470
         8     3345
         9     3254
         10    3251
         2     3047
         5     2994
         4     2368
         3      900
         Name: key, dtype: int64
```

Tulevaa visualisointia varten nämä tulisi olla kirjainmuodossa, eli jokaisen numeron tulisi vastata jotain nuottiavainta. Sarakkeen arvojen sisällön saa helposti vaihdettua `replace()`-funktiota hyödyntäen. Tässä tapauksessa funktiolle syötetään kaksi listaa: ensimmäisessä listassa on arvot, jotka halutaan muokata (numerot) ja toisessa listassa on uudet arvot, joilla vanhat korvataan. Huomion arvoista on se, että arvojen tulee olla oikeassa järjestyksessä molemmissa listoissa. Toimenpide suoritetaan ohjelmakoodin 4 tavoin.

Ohjelmakoodi 4. Nuottiavaimien vaihtaminen numeroista kirjaimiksi.

```
df['key'] = df['key'].replace(
    [0,1,2,3,4,5,6,7,8,9,10,11],
    ["C","C#","D","D#","E","F","F#","G","G#","A","A#","B"])
```

6 Visualisointi Matplotlib-kirjastolla

Kun alustat on asennettu ja datasetti putsattu, päästään datasetin esittämisvaiheeseen eli datan visualisointiin. Tässä opinnäytetyössä dataa visualisoidaan Matplotlib-kirjastoa hyödyntäen. Kirjaston saa tuotua mukaan projektiin samalla logiikalla kuin aikaisemminkin. Matplotlib tuodaan projektiin ohjelmakoodin 5 tavoin.

Ohjelmakoodi 5. Matplotlib-kirjaston tuontikomento.

```
import matplotlib.pyplot as plt
%matplotlib inline
```

Tarkemmin sanottuna ohjelmakoodin 5 ensimmäisellä rivillä projektiin tuodaan Matplotlibin Pyplot-moduuli, joka tarjoaa MATLAB-tyylisen käyttöliittymän. Toisella rivillä on niin kutsuttu **taikafunktio**, joka kertoo Jupyter Notebookille, että Python haluaa esittää ja tallentaa kaavioita projektivihkon sisällä. Seuraavissa alaluvuissa tutustutaan aluksi yksinkertaisiin tapoihin esittää dataa, mistä edetään hieman monimutkaisimpiin tapoihin.

6.1 Pylväskaavio

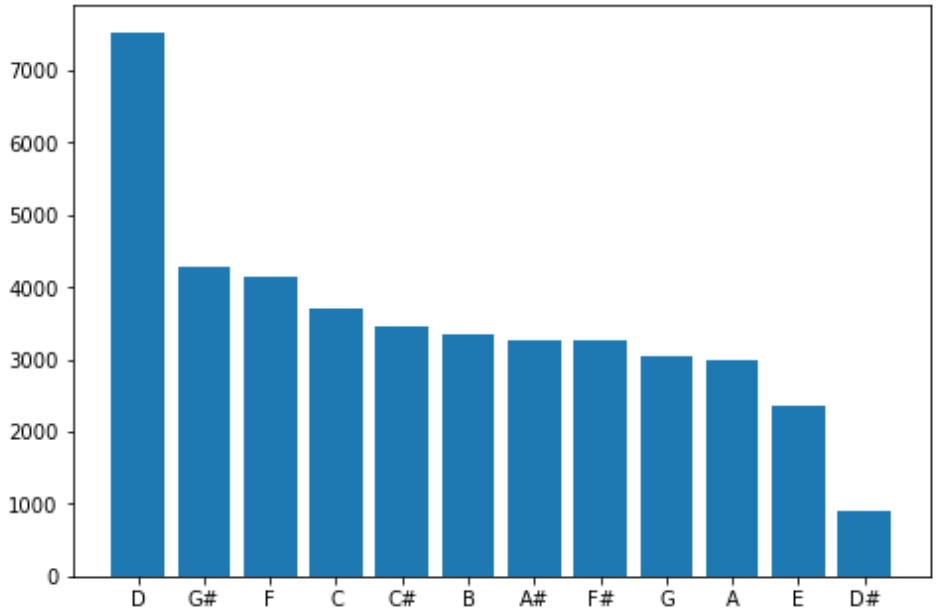
Pylväskaavio on yksi yksinkertaisimmista tavoista esittää dataa. Se hyödyntää nimensä mukaisesti pylväitä esittääkseen jotain kategorista dataa. Ohjelmakoodissa 6 esiintyvä `bar()`-funktio antaa käyttäjälleen mahdollisuuden luoda pylväskaavioita. Parametreinä sille tulee syöttää vähintään x- ja y-akselin arvot.

Ohjelmakoodi 6. Yksinkertainen pylväskaavio bar()-funktiolla.

```
x = df['key'].unique()
y = df['key'].value_counts()
fig = plt.figure()
ax = fig.add.axes([0,0,1,1])
ax.bar(x,y)
plt.show()
```

Ohjelmakoodissa 6 ensimmäisellä rivillä muuttujaan **x** tallennetaan datasetissä esiintyvät 12 eri nuottiavainta hyödyntäen Pandasin `unique()`-funktiota. Muuttujaan **y** sen sijaan tallennetaan jokaisen nuottiavaimien esiintyvyysluku `value_counts()`-funktiolla, jonka toimintaa esitettiin kuvassa 11. Matplotlibin `figure()`-funktiolla luodaan tyhjä kaavio, mihin lisätään akselit. Tämän jälkeen `bar()`-funktiolle syötetään x- ja y-akselien arvot ja kaavio printataan `show()`-funktiolla. Lopputulos nähdään kuvassa 12.

Kuva 12. Yksinkertainen pylväskaavio.



Pylväskaaviot ovat siitä erinomaisia, että ne suoriutuvat tehtävästään ilman sen suurempaa konfigurointia. Ne eivät tyypillisesti tarvitse samanlaista lisäkohennusta, kuten esimerkiksi alaluvun 6.2. pistekaavio. Kuvan 12 kaaviosta voidaan esimerkiksi heti havaita kaavion suurin ja pienin muuttuja, eli suosituin ja vähiten suosituin nuottiavain. Matplotlibiä hyödyntäen voidaan myös tehdä ohjelmakoodin 7 tavoin ryhmiteltyjä pylväskaavioita missä on useampi kuin yksi muuttuja.

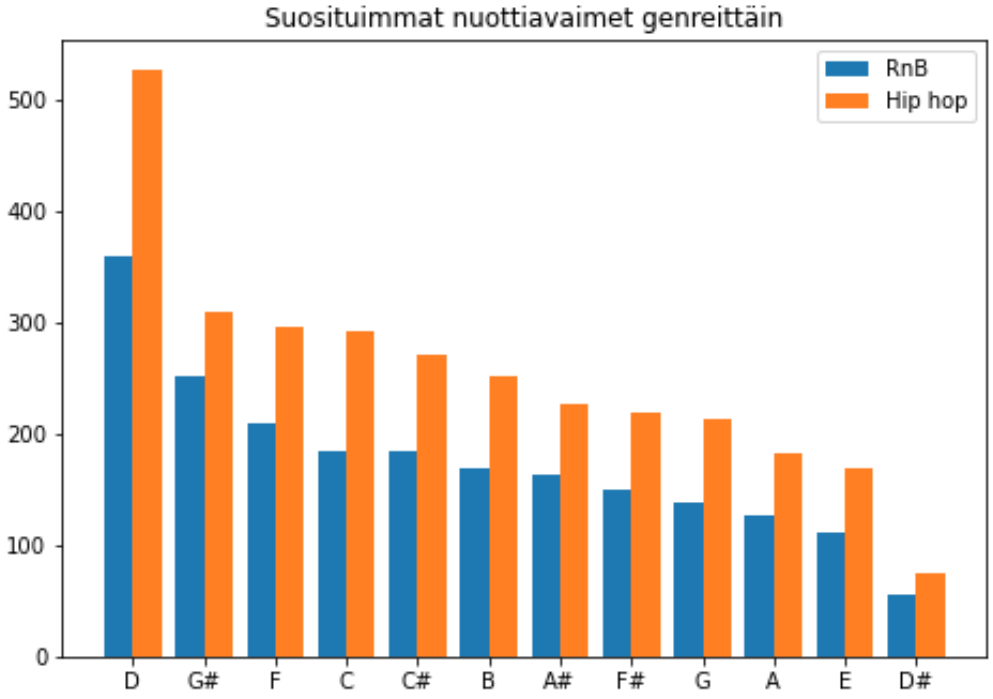
Ohjelmakoodi 7. Ryhmitellyn pylväskaavion esimerkkikoodi.

```
rnb = df.loc[(df['genre'] == 'RnB')]
rnb = rnb['key'].value_counts()
hiphop = df.loc[(df['genre'] == 'Hiphop')]
hiphop = hiphop['key'].value_counts()

import numpy as np
pylvaat = np.arange(len(x)) #len(x) = nuottiavainten lukumäärä (12)
leveys = 0.4

fig, ax = plt.subplots()
ax.bar(pylvaat - leveys/2, rnb, leveys, label='RnB')
ax.bar(pylvaat + leveys/2, hiphop, leveys, label='Hip hop')

ax.set_xticks(pylvaat) #x-akselille numerot 0-11
ax.set_xticklabels(x) #korvataan numerot nuottiavainten nimillä
ax.set_title("Suosituimmat nuottiavaimet genreittäin") #otsikko
ax.legend() #lisätään selitteet oikeaan yläkulmaan
plt.show()
```

Ohjelmakoodissa 7 luodaan ryhmitelty pylväskaavio, missä verrataan kahden eri genren nuottiavainjakaumaa. Valitut genret ovat **RnB** ja **hip hop**. Rivillä 1 alkuperäisestä df-

objektista luodaan uusi DataFrame-objekti nimeltä `rnb`. Pandasin `loc`-ominaisuutta hyödyntäen muuttujaan tallennetaan vain ne kappaleet, joiden genreksi on merkitty RnB. Tämän jälkeen `value_counts()`-funktiolla lasketaan jokaisen nuottiavaimen esiintyvyysluku, eli nyt muttuja `rnb` sisältää Series-objektin, joka sisältää RnB-kappaleiden nuottiavainjakauman. Tätä voidaan hyödyntää myöhemmin `bar()`-funktiossa. Riveillä 3 ja 4 sama toistetaan niille kappaleille, joiden genre on hip hop. Matplotlibin lisäksi hyödynnetään Numpy-kirjastoa. Numpyn `arange()`-funktiolla luodaan 12-alkioinen lista, missä jokainen arvo esittää yhtä nuottiavainta. Pylväiden leveysarvoksi annetaan 0.4 jotta ne erottuvat kaaviossa. Tämän jälkeen luodaan tyhjä kaavio, mille syötetään akselit ja pylväät. Pylväille syötetään alussa luodut `rnb` ja `hiphop` -nimiset Series-objektit. Lopputulos nähdään kuvassa 13.

Kuva 13. Ryhmitelty pylväskaavio.



6.2 Pistekaavio

Pistekaaviota käytetään kahden numeerisen muuttujan välisen suhteen tutkimiseen. Tässä tapauksessa datasetistä valitaan kaksi saraketta: tanssittavuus (danceability) ja kappaleen nopeus (tempo). Tutkitaan, onko näiden välillä jotain yhteyttä. Matplotlibin `scatter()`-

funktiolla voidaan luoda yksinkertainen pistekaavio, kun x- ja y-akselit ovat määritelty. Ohjelmakoodissa 8 esitetään funktion toimintaa.

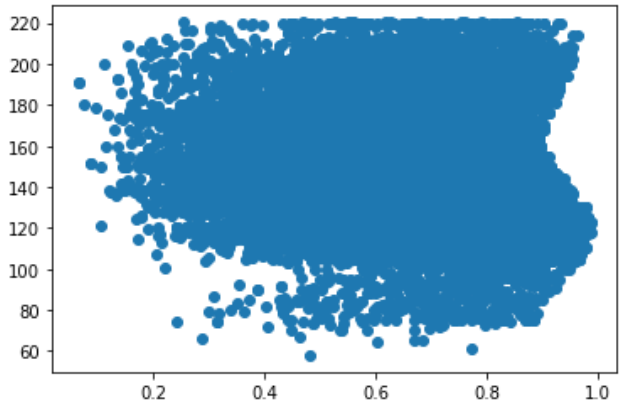
Ohjelmakoodi 8. Pistekaavion luonti.

```
x = df['danceability']
y = df['tempo']

plt.scatter(x, y)
```

Horisontaaliakselin muuttujaksi valitaan tanssittavuutta mittaava sarake. Vertikaaliakselin muuttujaksi valitaan kappaleen tempoa kuvaava sarake. Tämän jälkeen arvot syötetään Matplotlibin `scatter()`-funktiolle, minkä jälkeen ohjelmakoodin 8 solu voidaan ajaa. Tästä syntyvä lopputulos on esitettynä kuvassa 14.

Kuva 14. Esimerkki huonosta pistekaaviosta.

```
Out[8]: <matplotlib.collections.PathCollection at 0x2094c1058b0>
```



On sanomattakin selvää, että kuvan 14 kaavio on sekava, eikä siitä voida vetää minkäänlaisia johtopäätöksiä. Tätä ilmiötä kutsutaan englanniksi nimellä **overplotting**. Sanalle ei ole tarkkaa suomennosta, mutta sillä tarkoitetaan tilannetta, missä visualisoitavia datapisteitä on liikaa, minkä myötä johtopäätösten tekeminen vaikeutuu. Tämän lisäksi myös akseleiden otsikot puuttuvat. Lukija ei siis tiedä, mitä kaaviossa tarkalleen ottaen mitataan. Nämä ovat yleisiä ongelmia, minkä takia Matplotlib tarjoaa työkalut kaikkien näiden ongelmien ratkaisemiseen. Ohjelmakoodissa 9 kuvan 14 kaavio luodaan uusiksi, tällä kertaa parannusten kera.

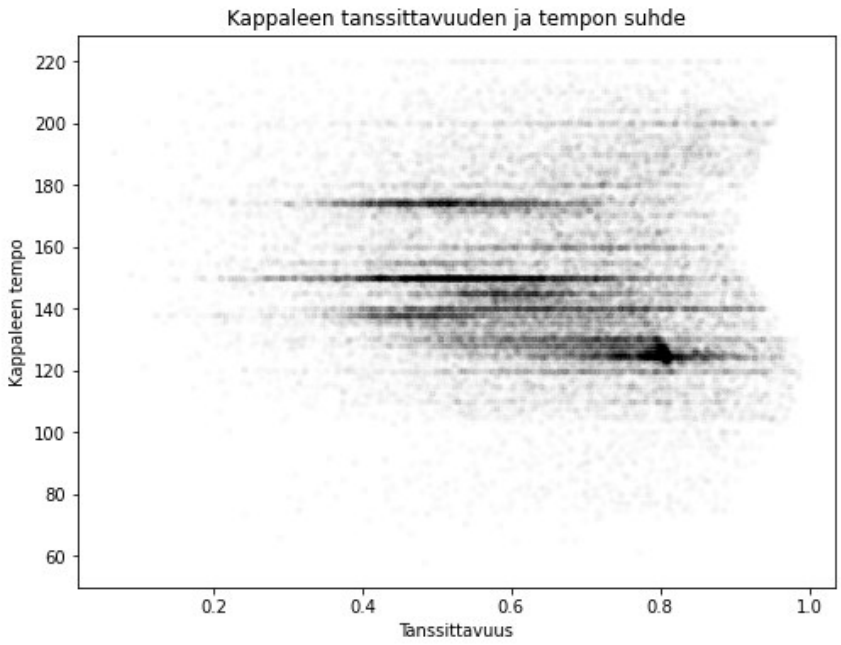
Ohjelmakoodi 9. Pistekaavion tarkemmat määritykset.

```
plt.figure(figsize=(8,6)) #kaavion koko
plt.title(”Kappaleen tanssittavuuden ja tempon suhde”) #otsikko
plt.xlabel(”Tanssittavuus”) #x-akselin otsikko
plt.ylabel(”Kappaleen tempo”) #y-akselin otsikko
plt.scatter(x, y, alpha=0.01, s=10, c=’black’)
```

Neljällä ensimmäisellä rivillä määritellään muun muassa kaavion koko sekä akselien otsikot. Tällä kertaa `scatter()`-funktiolle syötetään lisäargumentteja. Ensimmäiseksi syötetään x- ja y-akselit niin kuin ennen, mutta näiden lisäksi annetaan muutama lisäargumentti. Parametrin `alpha` avulla voidaan määrittää pisteiden läpinäkyvyys. Läpinäkyvyyden hyödyntäminen on yksi parhaimpia tekniikoita, kun isoja datamääriä halutaan esittää pistekaaviomuodossa. Parametrilla `s` määritellään datapisteiden koko. Kuvassa 14 esiintyvät pisteet ovat hieman suuria, joten niitä pienennetään vakioarvosta. Viimeisenä, parametri `c` määrittää datapisteiden värin. Tässä tapauksessa musta väri sopii valkoiseen taustaan erinomaisesti. Kuvassa 15 pistekaavio on paremmin tulkittavissa.

Kuva 15. Paranneltu pistekaavio.

```
Out[29]: <matplotlib.collections.PathCollection at 0x2094d9f4e50>
```



Kirjoitettavaa koodia lisättiin vain muutaman rivin verran, mutta kuten huomaa, ero on valtava. Pelkästään tätä yksinkertaista kaavioita silmäilemällä voidaan huomata, että kappaleita, joiden tempo on lähellä lukua 130, on paljon ja ne ovat tyypillisesti

tanssittavuustasoltaan korkeita. Tämän lisäksi kappaleita, joiden tempo on lähellä lukua 150 on paljon, mutta niiden tanssittavuustaso on paljon tasaisemmin jakautunut.

6.3 Histogrammi

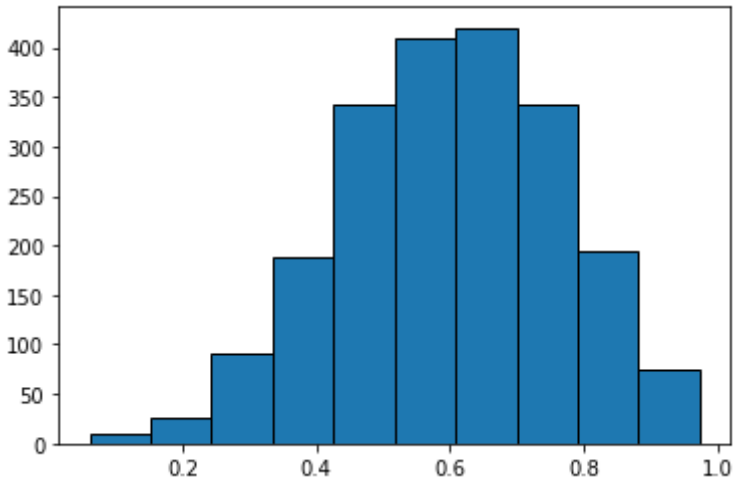
Histogrammi on kaavio, joka muistuttaa ulkonäöllisesti pylväskaaviota. Kaavioiden ero on kuitenkin se, että pylväskaavioilla tutkitaan kategorisia dataa, kun taas histogrammeilla tutkitaan diskreettiä dataa. Toisin sanoen, histogrammeja käytetään muuttujien jakaumien vertailuun. Histogrammeissa horisontaaliakselin vertailun kohteena olevan muuttujan eri arvot jaetaan niin kutsuttuihin koreihin (bins). (Robbins, 2012). Vertikaaliakseli kuvaa jokaisen korin taajuutta, eli sitä kuinka monta kertaa ne esiintyvät datasetissä. Tämän luvun esimerkissä tutkitaan RnB-kappaleiden energiatasojen jakaumaa. Matplotlibin `hist()`-funktiolla voidaan luoda histogrammi. Funktiolle tulee syöttää vähintään yksi argumentti: horisontaaliakselin muuttuja, jota halutaan tutkia. Kappaleen esimerkissä muuttujaksi valitaan lista, joka sisältää arvot jokaisen RnB-kappaleen energiatasosta. Tämän lisäksi funktion `edgecolor`-parametri saa argumentiksi arvon `black`, jotta pylväät saadaan erotettua paremmin toisistaan ääriviivojen avulla.

Kuvassa 16 nähdään ohjelmakoodin 10 lopputulos. Matplotlib jakaa tässä tapauksessa datan automaattisesti kymmeneen eri koriin, eli esimerkiksi oikeanpuolimmainen pylväs kuvaa arvoja väliltä 0,9–1,0. Kappaleita, joiden energiataso on tällä välillä, on noin 75.

Ohjelmakoodi 10. Histogrammin luonti.

```
x = df.loc[df.genre=='RnB', 'energy']
plt.hist(x, edgecolor='black')
plt.show()
```

Kuva 16. Yksinkertainen histogrammi.



Myös pylväskaaviot tekevät tehtävänsä ilman suurempia konfigurointeja, mutta lisäämällä hieman koodia niistä saa kuitenkin tehtyä silmälle mukavamman. Lisäksi, kuvan 16 kaaviosta puuttuu otsikot. Korien lukumäärää voidaan muokata käyttämällä `hist()`-funktion lisäparametria `bins`. Visualisoinnin kannalta sopiva korien lukumäärä riippuu käsiteltävästä datasta

Matplotlib tarjoaa käyttäjälleen mahdollisuuden hyödyntää värikarttoja pylväiden värittämiseen. Kaikki Matplotlibin värikartat löytyvät osoitteesta https://matplotlib.org/stable/tutorials/colors/colormaps.html. Tämän lisäksi myös eri kaaviotyylejä voidaan valita. Ohjelmakoodissa 11 käytetty `seaborn-whitegrid` muokkaa hieman otsikoiden fonttia sekä tuo taustalle havainnollistamista helpottavan kehikon (grid). Kaikki Matplotlibin tarjoamat tyylit löytyvät osoitteesta https://matplotlib.org/3.3.4/gallery/style_sheets/style_sheets_reference.html.

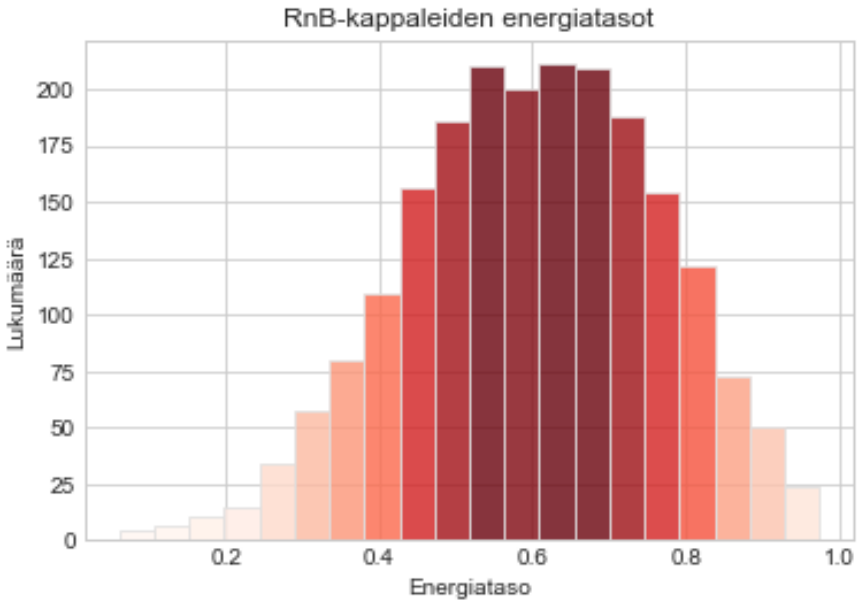
Ohjelmakoodin 11 tapauksessa `hist()`-funktio palauttaa arvon kolmelle muuttujalle. Muuttujaan `lkm` tallennetaan jokaisen pylvään korkeus eli toisin sanoen jokaisen korin taajuus. Muuttujaan `korit` tallennetaan korien jakauma. Kun käsiteltävän datan suurimmasta arvosta vähennetään pienin arvo ja erotus jaetaan korien lukumäärällä (tässä tapauksessa 20), saadaan tulokseksi pylväiden leveys. Muuttujaan `pylvaat` tallennetaan jokaista pylvästä vastaava Rectangle-objekti, jotta väritys voidaan suorittaa `for`-silmukalla.

Listaan `variarvot` määritetään jokaisen pylvään väri sen taajuusarvon perusteella. Logiikka toimii siis siten, että mitä korkeampi pylväs on, sitä tummempi sen väri tulee olemaan. Silmukassa kaikki 20 pylvästä käydään läpi ja jokaisen pylvään yksittäinen väriarvo määrittyy sen y-akselin korkeuden perusteella. Lopuksi määritellään jälleen akselien otsikot. Lopputulos nähdään kuvassa 17.

Ohjelmakoodi 11. Tyylitelty histogrammi.

```
x = df.loc[df.genre=='RnB', 'energy']

cm = plt.cm.get_cmap('Reds')
plt.style.use('seaborn-whitegrid')

lkm, korit, pylvaat = plt.hist(x, bins=20, edgecolor='#e0e0e0',
alpha=0.8)

variarvot = (lkm-lkm.min())/(lkm.max()-lkm.min())
for variarvo, pylvas in zip(variarvot, pylvaat):
    plt.setp(pylvas, 'facecolor', cm(variarvo))

plt.ylabel("Lukumäärä")
plt.xlabel("Energiataso")
plt.title("RnB-kappaleiden energiatasot")
plt.show()
```

Kuva 17. Kohennettu histogrammi.

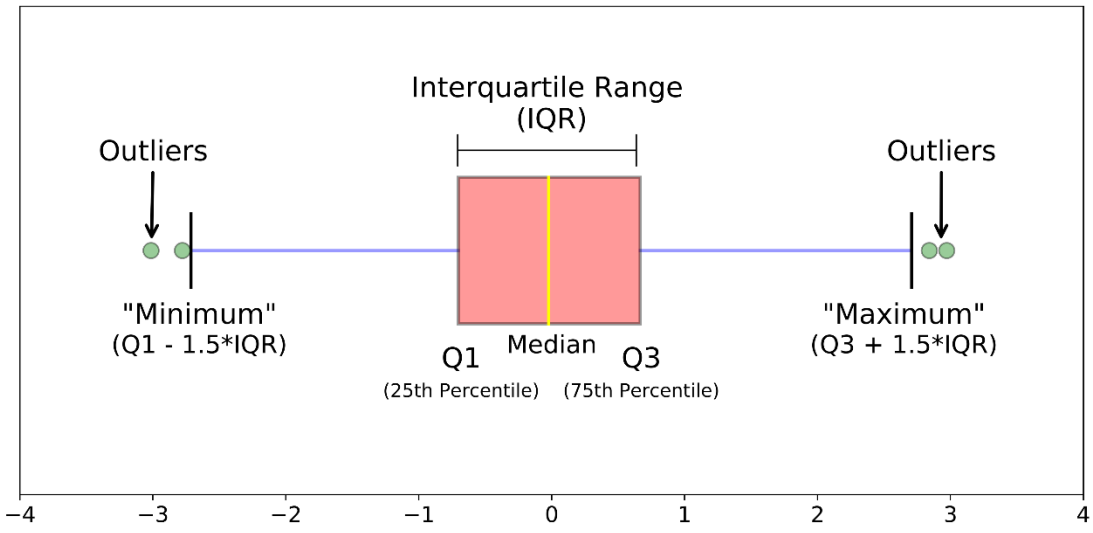


6.4 Laatikko-janakuvio

Toinen yleinen tapa visualisoida jakaumia on hyödyntää niin kutsuttua laatikko-janakuviota. Laatikko-janakuviota voidaan käyttää esimerkiksi useiden jakaumien vertailuun. Aluksi laatikko-janakuvion toimintaa voi olla vaikea sisäistää, mutta todellisuudessa se on erittäin looginen.

Kuvassa 18 on esimerkki laatikko-janakuviosta. Laatikko-janakuvioissa on tyypillisesti viisi pääelementtiä: **minimiarvo**, **alakvartiili**, **mediaani**, **yläkvartiili** ja **maksimiarvo**. Tämän lisäksi myös datasetin **poikkeamat** (outliers) voidaan havainnollistaa. Keskellä oleva keltainen viiva kuvaa datasetin keskimmäistä arvoa eli mediaania. Kuvissa mediaani on kuitenkin harvoin täysin keskellä, ja sen tarkka sijainti riippuu jakauman vinoudesta. Kuvan 18 esimerkki esittää normaalijakaumaa, joten mediaani on tässä tapauksessa tismalleen kuvan keskellä.

Kuva 18. Laatikko-janakuvio havainnollistettuna (Galarnyk, 2018).



Laatikko-janakuviossa esiintyvä laatikko esittää niin kutsuttua kvartiiliväliä (IQR). Kvartiiliväli kuvaa yläkvartiilin (Q3) ja alakvartiilin (Q1) erotusta. Toisin sanoen, se sisältää jakauman keskellä olevat havaintoarvot eli 50 % kaikista tapauksista. (Tilastokeskus, n.d.) Laatikosta lähtee myös kaksi sinistä viivaa. Vasemmanpuoleinen viiva kuvaa jakaumaa minimiarvon ja alakvartiilin välillä, oikeanpuoleinen taas yläkvartiilin ja maksimiarvon jakaumaa. Viimeisenä, datasetin poikkeamat (outliers) esitetään yksittäisinä arvoina. Laatikko-janakuvio voi sijaita

joko vaaka- tai pystyakselilla. Matplotlib-kirjaston `boxplot()`-funktiolla voidaan luoda laatikko-janakuvio.

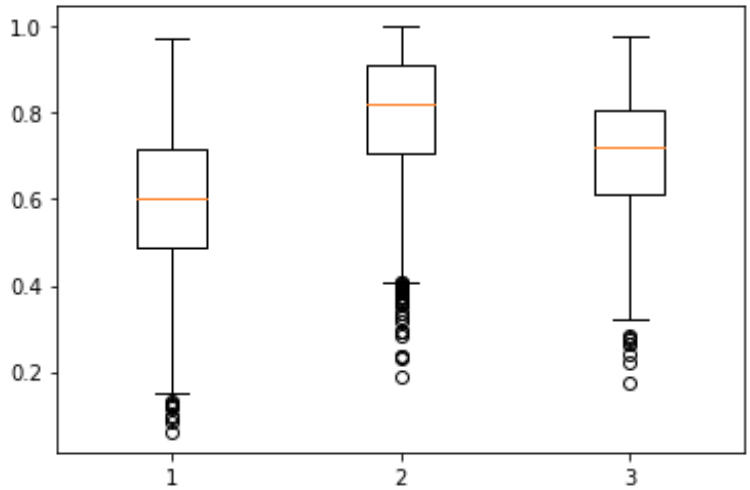
Ohjelmakoodissa 12 tallennetaan jokaisen genren energiatasot omaan muuttujaan. Tämän jälkeen luodaan lista, joka sisältää kaikki kolme muuttujaa. Lista syötetään tämän jälkeen `boxplot()`-funktiolle.

Ohjelmakoodi 12. Matplotlibin boxplot()-funktio

```
rnb_energia = df.loc[df.genre=='RnB', 'energy']
techno_energia = df.loc[df.genre=='techno', 'energy']
pop_energia = df.loc[df.genre=='Pop', 'energy']

genret = [rnb_energia, techno_energia, pop_energia]

plt.boxplot(genret)
plt.show()
```

Kuvassa 19 nähdään ohjelmakoodin 12 lopputulos. Funktio `boxplot()` luo laatikko-janakuvion automaattisesti pystyasentoon, mutta tämän saa tarvittaessa vaihdettua syöttämällä funktiolle argumentin `vert=False`. Kuvasta puuttuu toistaiseksi otsikot, mutta havaintoja voidaan jo tehdä esimerkiksi jakaumien vinoudesta: keskimmäinen jakauma näyttäisi olevan negatiivisesti vinoutunut, mikä tarkoittaa, että suurin osa arvoista on keskiarvoa suurempia. Ilmiötä kutsutaan englanniksi termillä negative skewness (tai vastaavasti toiseen suuntaan positive skewness). Jakaumat vaikkutavat siis olevan asymmetrisiä, toisin kuin kuvan 18 esimerkki.

Kuva 19. Boxplot()-funktiosta syntyvä laatikko-janakuvio.



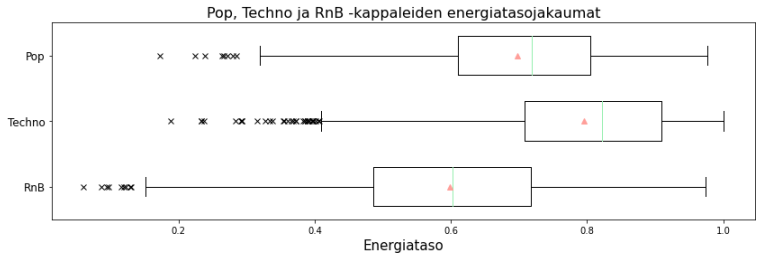
Ohjelmakoodissa 13 muokataan kaavion kokoa sopivammaksi ja lisätään otsikot. Vaikka koodia on näennäisesti paljon, suurimmat erot ohjelmakoodin 12 esimerkkiin tulee kuitenkin `boxplot()`-funktion parametriosassa. Funktion `widths`-parametrilla voidaan säätää laatikkojen leveyttä tai korkeutta, riippuen ollaanko vaaka- vai pystytasossa. Parametrilla `showmeans` voidaan näyttää jakauman keskiarvo. Lisäksi `vert`-parametri määrittää, onko kuvio pysty- vai vaakatasossa. Tässä tapauksessa valitaan vaakataso. Datasetin poikkeamat merkitään o-merkin sijaan x-merkillä.

Ohjelmakoodi 13. Mukautettu laatikko-janakuvio

```
rnb_energia = df.loc[df.genre=='RnB', 'energy']
techno_energia = df.loc[df.genre=='techno', 'energy']
pop_energia = df.loc[df.genre=='Pop', 'energy']
genret = [rnb_energia, techno_energia, pop_energia]

fig, ax = plt.subplots(figsize=(14,4))
ax.set_title('Pop, Techno ja RnB -kappaleiden energiatasojakaumat',
fontsize=16)
ax.set_xlabel('Energiataso', fontsize=15)

paksuudet = [.6,.6,.6]

plt.style.use('seaborn-pastel')
plt.boxplot(genret, widths=paksuudet, showmeans=True, vert=False,
sym='x')
plt.yticks([1,2,3],['RnB', 'Techno', 'Pop'], fontsize=12)
plt.show()
```

Kuvassa 20 on selkeämpi esitys kolmen eri genren kappaleiden energiatasojakaumista. Punainen kolmio esittää jakaumien **keskiarvoa**, ja vihreä viiva **mediaania**. Jakaumien vasemmalla puolella havaitut yksittäiset poikkeamat ovat merkitty x-merkillä. Kaikista voimakkaammin vinoutunut jakauma on keskellä, kun taas alimmassa jakaumassa arvot ovat jakautuneet yllättävän tasaisesti. Keskiarvo ja mediaani ovat melko lähellä toisiaan, mikä yleensä signaloi jakauman symmetrisyyttä. Negatiivista vinoutta on kuitenkin havaittavissa myös alimmassa jakaumassa.

Kuva 20. Yksinkertainen laatikko-janakuvio.



7 Visualisointi Seaborn-kirjastolla

Vaikka Matplotlib on tehokas ja paljon käytetty kirjasto, sillä on myös omat haasteensa. Koodirivejä tulee yleensä melko paljon, ja monimutkaisimmissa visualisoinneissa puhutaan useista kymmenistä riveistä. Tämän lisäksi kaavioiden estetiikka on tyypillisesti melko ankea. Seaborn luotiin osittain ratkaisemaan nämä kaksi ongelmaa: millä keinoin voidaan tehdä kauniita kaavioita vähällä koodimäärällä? Kaikki luvussa 6 esiintyvät kaaviot voidaan tehdä myös Seabornia hyödyntäen. Emme kuitenkaan lähde tekemään niitä toistamiseen. Seaborn tarjoaa lukuisia kehittyneempiä kaavioita, mitä Matplotlib ei kykene tuottamaan. Seaborn-kirjasto voidaan tuoda projektiin tutulla `import`-komennolla. Ohjelmakoodin 14 tapauksessa sille annetaan nimeksi `sns`, samalla tavalla kuin luvussa 6 Matplotlibille annettiin lyhenne `plt`.

Ohjelmakoodi 14. Seaborn-kirjaston tuonti projektiin.

```
import seaborn as sns
```

7.1 Ydinestimointikaavio

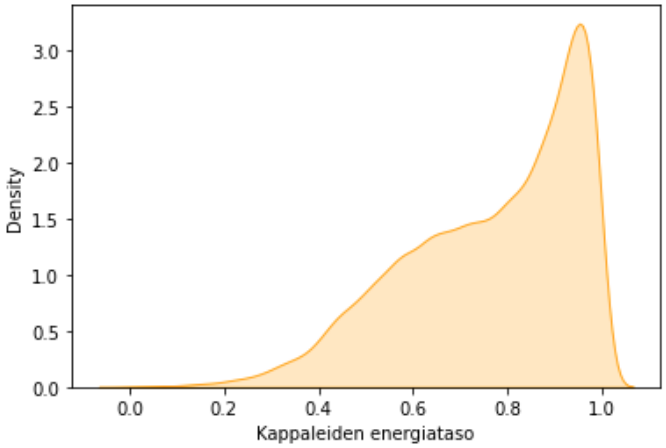
Ydinestimointikaavio toimii samalla logiikalla kuin alaluvussa 6.3. esitetty histogrammi, mutta pylväiden sijaan jakauma esitetään käyrän avulla. Ydinestimointikaavio on ikään kuin pehmeämpi versio histogrammista. Seabornin `kdeplot()`-funktion avulla voidaan luoda ydinestimointikaavioita.

Seabornin `kdeplot()`-funktio voi ottaa vastaan useita argumentteja, mutta ainoa pakollinen argumentti on sille syötettävä data. Ohjelmakoodin 15 esimerkissä käytämme jälleen datasetissä esiintyvien kappaleiden energiatasoa. Tämän lisäksi funktiolle syötetään haluttu väri, ja `shade`-parametrillä todetaan, että syötetystä datasta syntyvä käyrä halutaan värittää annetulla värillä. Lopputuloksena syntyy kuvassa 21 esiintyvä kaavio.

Ohjelmakoodi 15. Ydinestimointikaavion luonti kdeplot()-funktiolla.

```
x = df['energy']
sns.kdeplot(x, color='g', shade=True)
plt.xlabel("Kappaleiden energiataso")
plt.show()
```

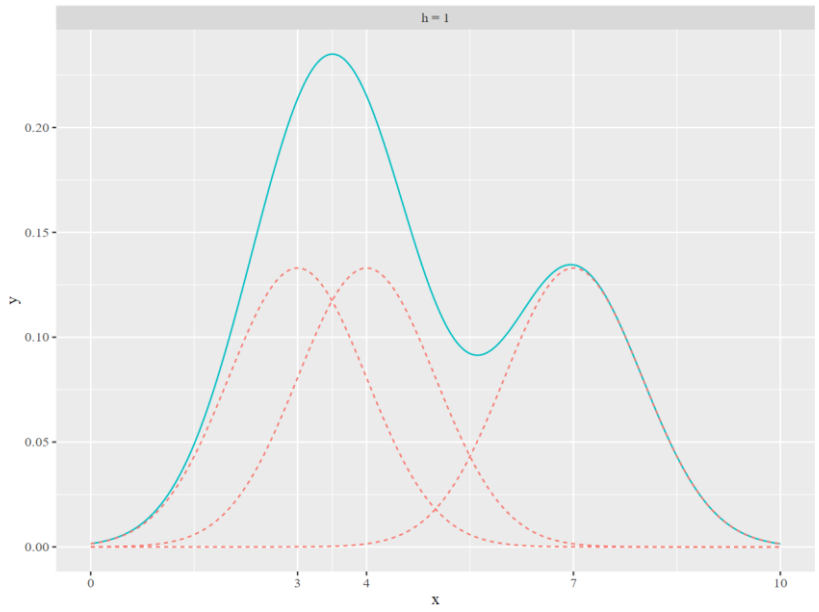
Kuva 21. Yksinkertainen ydinestimointikaavio.



Ydinestimointikaavio koostuu nimensä mukaisesti niin kutsutuista ytimistä. Ydinestimointikaavion idea on, että jokaisesta datapisteestä luodaan oma ydin (kernel) ja lopuksi nämä summataan yhteen.

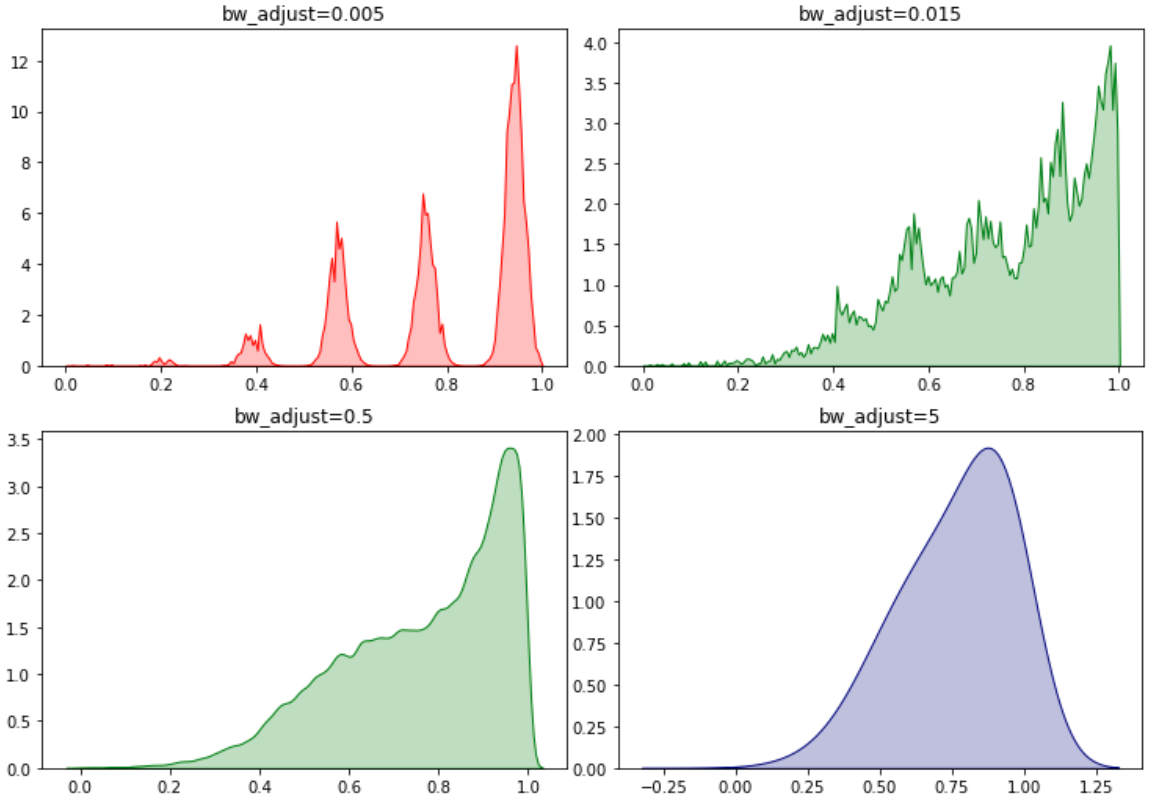
Kuvassa 22 havainnollistetaan ydinestimointikaavion toimintatapaa. Datasetissä on kolme datapistettä x-akselin kohdissa 3, 4 ja 7. Jokaisesta datapisteestä luodaan samanlainen, normaalijakaumaa muodostuma jakauma eli ydin. Lopuksi ytimet summataan yhteen ja saadaan turkoosilla merkitty lopputulos, jota kutsutaan ydinestimointikaavioksi. Ydinestimointikaavioita luodessa voi olla haastavaa löytää ytimille sopiva leveysarvo. Kuvan 22 esimerkissä leveysarvoksi on merkitty 1, mikä on myös `kdeplot()`-funktion oletusarvo.

Kuva 22. Ydinestimointikaavion ytimet ja niiden summa. (Akinshin, 2020).



Ydinten leveyttä voidaan kuitenkin muokata funktion `bw_adjust`-parametrin avulla. Kuvassa 23 sama data on esitetty neljällä eri leveysarvolla. Kuten voidaan huomata, ero on valtava.

Kuva 23. Ydinestimointikaavio eri bw_adjust-arvoilla.



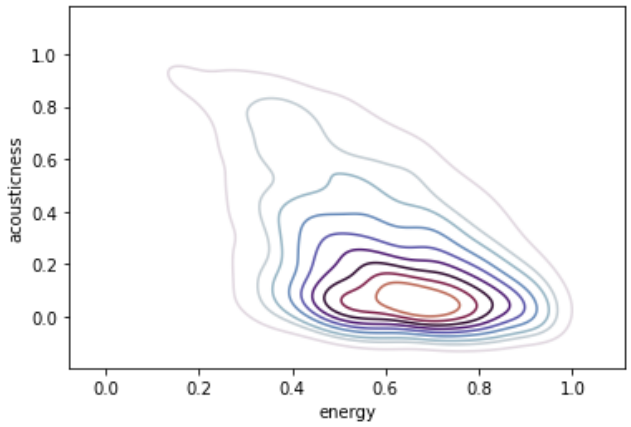
Punaisen kaavion leveysarvo on aivan liian pieni. Sinisessä kaaviossa asia on taas päinvastoin – liian suuri leveysarvo saa datan näyttämään normaalijakaumalta. Vihreällä merkityt kaaviot ovat sen sijaan paremmin luettavissa. Visualisoinnin kannalta sopiva leveysarvo tulee valita käsiteltävän datan perusteella.

Ydinestimointikaaviolla voidaan kuvata myös kahden muuttujan suhteen esiintyvyystiheyttä. Tällöin jokaisesta datapisteestä luodaan käyrän sijaan oma “rengas”, ja lopuksi renkaat summataan yhteen samalla logiikalla kuin kuvan 22 esimerkissä. Ohjelmakoodissa 16 määritetään tuttuun tapaan ensiksi x- ja y-akselien arvot. Tällä kertaa `kdeplot()`-funktion parametri `levels` saa arvokseen 10. Parametrin avulla määritetään, kuinka monta tiheysrengasta kaaviossa on, ohjelmakoodin 16 tapauksessa kymmenen. Lopputulos nähdään kuvassa 24.

Ohjelmakoodi 16. Kahden muuttujan ydinestimointikaavion luonti.

```
x1 = df.loc[df.genre=='RnB', 'energy']
y1 = df.loc[df.genre=='RnB', 'acousticness']
sns.kdeplot(x=x1, y=y1, levels=10, cmap='twilight')
plt.show()
```

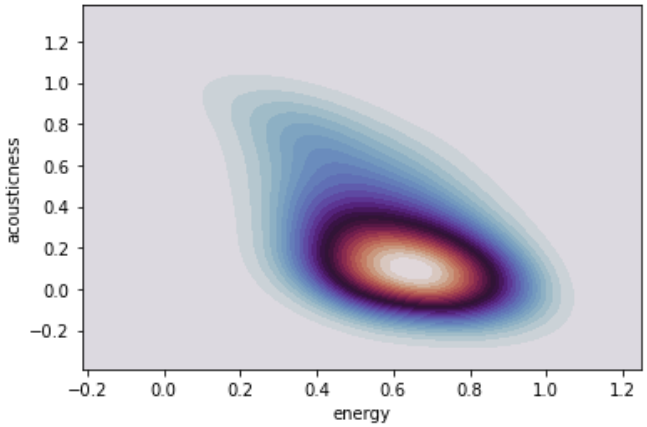
Kuva 24. Kahden muuttujan ydinestimointikaavio.



Tästäkin voidaan vielä parantaa. Kuvassa 25 `levels`-parametrin arvoksi on tällä kertaa asetettu 30. Tämän lisäksi funktiolle on syötetty kolme lisäargumenttia: `bw_adjust=2`, `shade=True` ja `thresh=0`. Tällä kertaa `bw_adjust`-parametrin arvoksi on asetettu 2, mikä hieman pyöristää `levels`-parametrin luomia renkaita. Parametri `shade` on niin kutsuttu boolean-muuttuja, joka määrittää väritetäänkö renkaiden sisältö. Tässä tapauksessa renkaat

halutaan värittää, joten sille annetaan arvo `True`. Viimeiseksi, parametri `thresh` määrittää arvojen ulkopuolelle jäävän tilan värityksen, eli kuvan 25 tapauksessa nolla-arvot saavat `twilight`-värikartasta harmaan värin, mikä tekee kaaviosta hieman katsojaystävällisemmän.

Kuva 25. Kahden muuttujan ydinestimointikaavio parannuksilla.



7.2 Viulukaavio

Toinen mielenkiintoinen Seabornin tarjoama kaavio on niin kutsuttu viulukaavio. Viulukaavio on ikään kuin laatikko-janakuvion ja ydinestimointikaavion yhdistelmä. Seaborn-kirjasto tarjoaa `violinplot()`-funktion, jolla voidaan luoda viulukaavio. Funktion toimintaa esitetään ohjelmakoodissa 17.

Ohjelmakoodi 17. Viulukaavion luontikomento

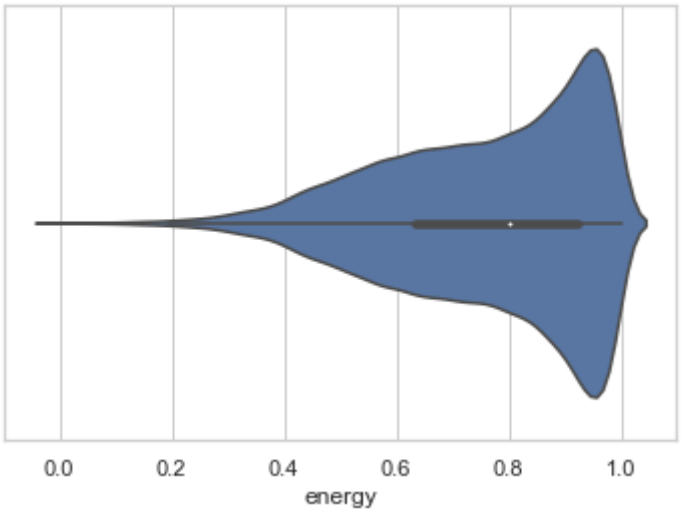
```
sns.set_theme(style="whitegrid")
sns.violinplot(x=df['energy'])
plt.show()
```

Yksinkertaista viulukaaviota varten funktiolle ei tarvitse syöttää muuta kuin sarake, jonka jakauma halutaan visualisoida. Ohjelmakoodin 17 tapauksessa sille syötetään sarake `energy`. Ensimmäisellä rivillä oleva `set_theme()`-funktio on täysin valinnainen Seabornin tarjoama funktio, jolla voi muokata kaavion teemaa.

Kuvassa 26 on esitetty koodista syntyvä viulukaavio. Sinisellä oleva jakauma on toteutettu luvussa 7.1. käsitellyn ydinestimointikaavion tavoin. Tämän lisäksi siinä esiintyy laatikko-

janakaaviosta tuttu laatikko, jolla kuvataan jakauman kvartiiliväliä. Tämän lisäksi kvartiilivälin sisällä on piste, joka kertoo jakauman mediaanin.

Kuva 26. Viulukaavio havainnollistettuna.



Ohjelmakoodissa 18 luodaan usean muuttujan viulukaavio. Tarkoituksena on vertailla kolmen eri genren energiajakaumaa, samalla tavalla kuin kuvan 20 esimerkissä.

Ohjelmakoodi 18. Viulukaavio usealla muuttujalla.

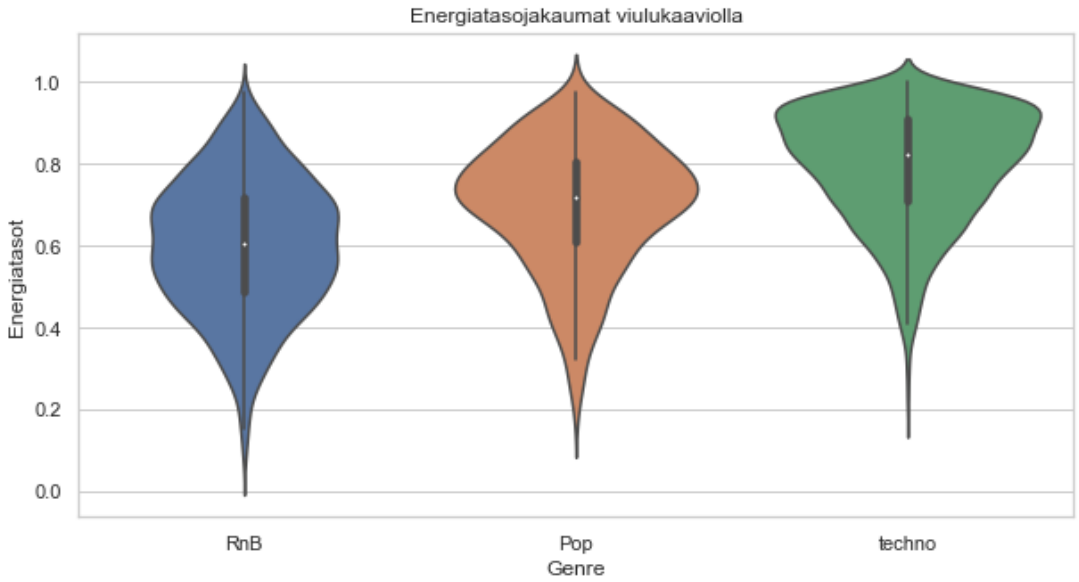
```
genret = df[
    (df['genre'] == 'Pop') | (df['genre'] == 'RnB') | (df['genre'] ==
'techno')
]

fig, ax = plt.subplots(figsize=(10,5))
sns.violinplot(x=genret['genre'], y=genret['energy'])
ax.set_xlabel("Genre")
ax.set_ylabel("Energiatasot")
ax.set_title("Energiatasojakaumat viulukaaviolla")
plt.show()
```

Vertailtavista genreistä luodaan ensiksi oma DataFrame-objekti, minkä jälkeen luodun objektin `genre`-sarake valitaan x-akselin muuttujaksi ja syötetään `violinplot()`-funktiolle. Vertikaaliakselin muuttujaksi valitaan `energy`-sarake.

Kuten kuvasta 27 voi päätellä, viulukaaviot soveltuvat laatikko-janakuvioiden tapaan hyvin esimerkiksi jakaumien vertailuun. Viulukaavio on katsojaystävällisempi tapa visualisoida

jakaumia, ja niistä on helppo tehdä johtopäätöksiä. Tästä huolimatta viulukaavio ei ole yhtä suosittu, kuin esimerkiksi laatikko-janakuvio.

Kuva 27. Energiatasojakaumat viulukaaviolla.



8 Yhteenveto

Datan visualisointi on välttämätön taito esimerkiksi datatieteen alalla, ja se on olennainen osa tiedon välittämistä nykypäivän työelämässä. Tämän opinnäytetyön päätavoitteena oli lisätä tietoisuutta siitä, miten dataa visualisoidaan Python kielellä sekä vastata johdannossa mainittuihin tutkimuskysymyksiin. Mielestäni tässä onnistuttiin hyvin. Kattava teoria- ja käytännön osa antoi kokonaisvaltaisen kuvan datasta, datan visualisoinnista ja siitä, miten dataa voidaan visualisoida hyödyntämällä Pythonin kirjastoja.

Työssä käytiin lähinnä läpi vain suosituimpia kaavioita, minkä lisäksi Seaborn-kirjastolla tehtiin pari vähemmän tunnettua kaaviototeutusta. Opinnäytetyön pituuden nimissä kaikkiin mahdollisiin visualisointimenetelmiin ei voitu tutustua, vaan työn tarkoitus on toimia ikään kuin ohjekirjana visualisoinnin aloittamiselle, sekä tutustua yksinkertaisimpiin datan visualisointiratkaisuihin. Lisäksi tarkoituksena oli selvittää, miten nykyaikaisia, ilmaisia ja avoimeen lähdekoodiin perustuva ohjelmistoratkaisuja hyödyntämällä voidaan tuottaa visualisointeja. Todettakoon, että Jupyter Notebook ei ole suinkaan ainoa ympäristö toteuttaa kaavioita.

Ohjelmointikielenä Python soveltuu erinomaisesti datan visualisointiin. Se on syntaksiltaan korkeatasoinen kieli, eli se on helposti ymmärrettävissä ilman syvällistä ohjelmointiosaamista. Pandas-kirjastolla pystytään käsittelemään tehokkaasti csv-muodossa olevaa dataa. Kirjaston DataFrame-tietorakenne sallii datasetin vaivattoman manipuloinnin, minkä lisäksi se on suhteellisen nopea isojakin datamääriä käsitellessä.

Itse visualisoinnissa hyödynnetyt Matplotlib ja Seaborn -kirjastot todettiin toimiviksi. Huomattiin, että Matplotlibillä tehtyjä kaavioita on tyypillisesti hieman vaikeampi tyylitellä, ja että niihin käytettiin enemmän koodia. Tästä huolimatta Matplotlib on edelleen tänä päivänä Python-visualisoinnin de facto -standardi. Seaborn on kuitenkin ottamassa visualisoinnissa jalansijaa. Se käyttää vähemmän koodia, ja kaavioita on helpompi kustomoida. Seaborn on opinnäytetyön kirjoittamisen aikaan kirjastona vielä suhteellisen uusi, ja on mielenkiintoista seurata, miten kirjastoa jatkokehitetään tästä eteenpäin.

Lähteet

Akinshin, A. (2020). *The importance of kernel density estimation bandwidth | Andrey Akinshin*. https://aakinshin.net/posts/kde-bw/

Alonso, E. (2019, September 25). *Datan puhdistaminen: Mitä se on ja miksi CRM:si tarvitsee sitä*. https://www.vainu.com/fi/blogi/datan-puhdistaminen/

Anaconda. (2021). *Anaconda | The World's Most Popular Data Science Platform*. https://www.anaconda.com/

Andrews, E. (2016, November 17). *8 Legendary Ancient Libraries - HISTORY*. https://www.history.com/news/8-impressive-ancient-libraries

Beal, V. (n.d.). *What is a High Level Programming Language?* Webopedia. Retrieved April 2, 2021, from https://www.webopedia.com/definitions/high-level-language/

Brown, J. (n.d.). *Data vs. Information vs. Insight*. Retrieved January 21, 2021, from https://online.ben.edu/programs/mba/resources/data-vs-information-vs-insight

Chen, C., Härdle, W., Unwin, A., & Friendly, M. (2008). A Brief History of Data Visualization. In *Handbook of Data Visualization* (pp. 15–56). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-33037-0_2

Ciklum. (2019). *Pros and Cons of Big Data*. Ciklum. https://www.ciklum.com/blog/pros-and-cons-of-big-data/

Creative Commons. (n.d.). *CC0 - Creative Commons*. Retrieved February 2, 2021, from https://creativecommons.org/share-your-work/public-domain/cc0/

Donges, N. (2018, March 19). *Data Types in Statistics. Data Types are an important concept of… | by Niklas Donges | Towards Data Science*. https://towardsdatascience.com/data-types-in-statistics-347e152e8bee

Etymonline. (n.d.). *data | Origin and meaning of data by Online Etymology Dictionary*. Retrieved February 1, 2021, from https://www.etymonline.com/word/data

Friedman, V. (2008, January 14). *Data Visualization and Infographics — Smashing Magazine*. Smashing. https://www.smashingmagazine.com/2008/01/monday-inspiration-data-visualization-and-infographics/

Galarnyk, M. (2018, September 12). *Understanding Boxplots*. https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Kaggle. (n.d.). *Dataset of songs in Spotify*. Retrieved February 2, 2021, from

https://www.kaggle.com/mrmorj/dataset-of-songs-in-spotify

Khvoynitskaya, S. (2020, January 30). *The future of big data: 5 predictions from experts for 2020-2025*. https://www.itransition.com/blog/the-future-of-big-data

Kivelä, S., Nurmiainen, R., & Spåra, M. (2000). *Tilastodata*. https://matta.hut.fi/matta2/isom/html/tilasto1.html

Ma, X., Hummer, D., Golden, J., Fox, P., Hazen, R., Morrison, S., Downs, R., Madhikarmi, B., Wang, C., & Meyer, M. (2017). Using Visual Exploratory Data Analysis to Facilitate Collaboration and Hypothesis Generation in Cross-Disciplinary Research. In *International Journal of Geo-Information* (Vol. 6, Issue 11). https://www.researchgate.net/publication/321111181_Using_Visual_Exploratory_Data_Analysis_to_Facilitate_Collaboration_and_Hypothesis_Generation_in_Cross-Disciplinary_Research

Marr, B. (2015, February 25). *A Brief History of Big Data Everyone Should Read | Bernard Marr | LinkedIn*. Linked In. https://www.linkedin.com/pulse/brief-history-big-data-everyone-should-read-bernard-marr

Meleen, M. (n.d.). *Difference Between Data and Information Explained*. YourDictionary. Retrieved April 2, 2021, from https://examples.yourdictionary.com/difference-between-data-and-information-explained.html

Misal, D. (2019). *Comparing Python Data Visualization Tools: Matplotlib vs Seaborn*. https://analyticsindiamag.com/comparing-python-data-visualization-tools-matplotlib-vs-seaborn/

Nantasenamat, C. (2020). *The Data Science Process.* https://towardsdatascience.com/the-data-science-process-a19eb7ebc41b

Press, G. (2013, May 12). *A Very Short History Of Big Data*. https://www.forbes.com/sites/gilpress/2013/05/09/a-very-short-history-of-big-data/?sh=3add5c7d65a1

Press, G. (2014, September 3). *12 Big Data Definitions: What's Yours?* https://www.forbes.com/sites/gilpress/2014/09/03/12-big-data-definitions-whats-yours/?sh=1b35645013ae

Project Jupyter. (2021). *Project Jupyter | Try Jupyter*. https://jupyter.org/try

Python Software Foundation. (2021a). *General Python FAQ* . https://docs.python.org/3/faq/general.html

Python Software Foundation. (2021b). *pandas - PyPI*. https://pypi.org/project/pandas/

Robbins, N. (2012, January 4). *A Histogram is NOT a Bar Chart*. Forbes. https://www.forbes.com/sites/naomirobbins/2012/01/04/a-histogram-is-not-a-bar-chart/?sh=568a45ce6d77

Seltman, H. J. (2018). *Experimental Design and Analysis*. https://www.stat.cmu.edu/~hseltman/309/Book/Book.pdf

Statistics Canada Quality Guidelines. (2009). https://unstats.un.org/unsd/dnss/docs-nqaf/Canada-12-539-x2009001-eng.pdf

Stikeleather, J. (2013, April 18). *The Three Elements of Successful Data Visualizations*. https://hbr.org/2013/04/the-three-elements-of-successf

Tableau Software. (2018). *What is data visualization? A definition, examples, and resources*. Tableau.Com. https://www.tableau.com/learn/articles/data-visualization

Thumar, C. (2019). *Why Python is Good for Data Analytics?* ThriveGlobal. https://thriveglobal.com/stories/why-python-is-good-for-data-analytics/

Tilastokeskus. (n.d.). *Kvartiiliväli*. Retrieved February 23, 2021, from https://www.stat.fi/meta/kas/kvartiilivali.html

Treehouse Technology Group. (n.d.). *The Psychology Behind Data Visualization*. Retrieved February 4, 2021, from https://treehousetechgroup.com/the-psychology-behind-data-visualization/

Tufte, E. (1983). *The Visual Display of Quantitative Information*. https://www.edwardtufte.com/bboard/q-and-a-fetch-msg?msg_id=00040Z

VanderPlas, J. (2013, March 13). *Matplotlib and the Future of Visualization in Python*. https://jakevdp.github.io/blog/2013/03/23/matplotlib-and-the-future-of-visualization-in-python/

Liite 1: Aineistonhallintasuunnitelma

Opinnäytetyössä käytetty datasetti on haettu kaggle.com-sivustolta. Datasetti pohjautuu Creative Commonsin CC0 Public Domain –lisenssiin, eli se on luovutettu vapaaseen yleiseen käyttöön.

Datasetin ja opinnäytetyöstä syntyvien tiedostojen pääasiallisena säilytyspaikkana toimii kotikoneen C-asema. Syy tälle on se, että Jupyter Notebook -alustan on löydettävä datasetti jostakin tietokoneen kansiosta. Eli toisin sanoen datasetin on fyysisesti sijaittava koneella, jolla opinnäytetyön käytännön osa tehdään. Jokaisen kirjoitussession jälkeen syntynyt aineisto varmuuskopioidaan. Varmuuskopiot säilytetään henkilökohtaisella Google Cloud –palvelimella sekä Hämeen ammattikorkeakoulun MyFiles–palvelussa.

Opinnäytetyössä ei käsitellä luottamuksellista tietoa. Opinnäytetyöstä ei synny sellaista aineistoa, mikä olisi hyödynnettävissä esimerkiksi kaupallisessa tarkoituksessa. Datasetti voidaan kuitenkin poistaa opinnäytetyön käytännön osan jälkeen, vaikka sille ei varsinaisesti ole mitään tarvetta.