SAC-strategian käyttö ryhmitellyn suuren datan analyysissa

Matleena Lehto

LuK-tutkielma
Tammikuu 2026

MATEMATIIKAN JA TILASTOTIETEEN LAITOS

TURUN YLIOPISTO
Matematiikan ja tilastotieteen laitos

Matleena Lehto: SAC-strategian käyttö ryhmitellyn suuren datan analyysissa
LuK-tutkielma, 23 s.
Tilastotiede
Tammikuu 2026

Tämän tutkielman tavoitteena on tarkastella funktionaalisen ohjelmoinnin käyttöä tilastollisessa data-analyysissa, erityisesti jaa–sovita–yhdistä (eng. split–apply–combine) -strategian näkökulmasta. Funktionaalinen ohjelmointi on ohjelmointiparadigma, jossa ohjelmat rakentuvat funktioiden yhdistämisestä ja se tuottaa toistettavaa, puhdasta ja luotettavaa koodia. R-kielessä paradigmaa tukevat sisäänrakennetut funktiot, kuten apply, lapply ja tapply sekä laajemmat työkalut `dplyr`- ja `purrr`-paketit.

Tutkielman empiirisessä osassa sovelletaan split-apply-combine -strategiaa NBA:n viralliselta sivustolta ladattuun aineistoon, joka käsittelee 572 pelaajan dataa eli pisteitä, pelattuja minuutteja ja pelaajien ikää kaudelta 2023-2024. Analyysissa aineisto jaetaan joukkueittain, minkä jälkeen pelaajakohtaisia tilastoja tutkitaan funktionaalisen ohjelmoinnin menetelmillä. Täten havainnollistetaan, kuinka suuret datakokonaisuudet voidaan käsitellä tehokkaasti jakamalla ne ensin osiin (split), toteuttamalla annettu analyysitehtävä funktionaalisesti (apply), ja lopuksi kokoamalla tulokset yhteen (combine).

Asiasanat: funktionaalinen ohjelmointi, split, apply, combine, R-kieli

Sisällys

Johdanto

Funktionaalinen ohjelmointi on ohjelmointiparadigma eli ohjelmointityyli, jonka tavoitteena on vähentää toisteisuutta, lisätä koodin luotettavuutta ja tehdä datan käsittelystä järjestelmällistä sekä helposti toistettavaa. Paradigma rakentuu funktioiden yhdistämiselle ja vaiheittain etenevälle aineistonkäsittelylle. Tämä rakenne vähentää virheiden mahdollisuutta ja tekee analyysiprosessista toistettavan, mikä on tärkeää tilastollisessa ohjelmoinnissa. Tilastolliset analyysit koostuvat usein useista samankaltaisista laskentavaiheista, jotka suoritetaan eri muuttujille, ryhmille tai ajanjaksoille. Funktionaalinen ohjelmointi tarjoaa keinon toteuttaa nämä vaiheet ilman ylimääräistä, toisteista koodia yhtenäisen ohjelmarungon avulla.

Tässä tutkielmassa funktionaalista ohjelmointia tarkastellaan niin sanotun jaa–sovita–yhdistä -strategian (eng. split–apply–combine, SAC-strategia[1]) avulla. Strategian idea perustuu siihen, että jaetaan analyysi pienempiin osiin (split), suoritetaan annettu tehtävä funktionaalisesti (apply) ja yhdistetään tulokset takaisin yhdeksi kokonaisuudeksi (combine).

Empiirisessä osassa paradigmaa sovelletaan NBA:n viralliselta sivulta kerättyyn tilastodataan, joka sisältää 572 pelaajaa tavalliselta pelikaudelta 2023-2024. Aineiston avulla tarkastellaan, miten toistuvat joukkuekohtaisesti tehtävät analyysit voidaan automatisoida funktionaalisella ohjelmoinnilla. Esimerkiksi pisteiden, peliaikojen ja muiden suoritusten laskenta voidaan toteuttaa samalla funktiokokonaisuudella riippumatta siitä, mitä joukkuetta tarkastellaan. Ohjelma toimii esimerkkinä tilastollisesta mallinuksesta, jossa funktionaalinen ohjelmointi vähentää toisteisuutta, pienentää virhemahdollisuutta ja mahdollistaa analyysin tekemisen uudelle datalle.

Tutkielman teoriaosuudessa esitellään funktionaalisen ohjelmoinnin peruskäsitteistöä, joka luo pohjan myöhemmin esiteltäville R-kielen keinoille, jotka mahdollistavat funktionaalisen ohjelmoinnin käytännön toteutuksen. Käsitteiden määrittelyssä on hyödynnetty esimerkiksi Wikipediaa, Tieteen termipankkia sekä K. Häkkisen matematiikan propedeuttista kurssia. Englanninkielisten termien suomennokset on tarkistettu Suomen Tilastoseura ry:n sanastosta.

Lähteinä teoriaosuuksissa käytettiin Hadley Wickhamin artikkelia [18] ja kirjaa [3] sekä John McKinley Chambersin kirjan [17] funktionaalista ohjelmointia käsitteleviä lukuja. Näiden pohjalta tutkielma rakentaa selkeän kuvan siitä, miten funktionaalinen ohjelmointi mahdollistaa systemaattisen ja toistettavan data-analyysin.

[1] SAC-strategialle ei ole vakiintunutta suomenkielistä vastinetta, joten tässä tutkielmassa käytetään englanninkieliseen nimeen perustuvaa lyhennettä

1 Funktionaalinen ohjelmointi

Tässä luvussa esitellään funktionaaliseen ohjelmointiin kuuluvia käsitteitä ja periaatteita. Funktionaaliselle ohjelmoinnille ei ole yleistä määritelmää. Lisäksi luvussa annetaan yksinkertainen määritelmä funktiolle ja esimerkki sen toteuttamisesta R-kielessä. Lopuksi vertailen funktionaalista ja imperatiivista ohjelmointiparadigmaa havainnollistaakseni näiden eroja. Luvussa on käytetty lähteinä [1-17].

1.1 Tärkeitä käsitteitä

Tässä alaluvussa esitellään funktionaalisen ohjelmointiparadigman keskeiset käsitteet. Käsitteet on järjestetty siten, että ne etenevät matemaattisesta funktiokäsitteestä ohjelmoinnin näkökulmaan. Luvussa luodaan luoda teoreettinen pohja myöhemmissä luvuissa esiteltävälle SAC-strategialle sekä sitä toteuttaville R-kielen keinoille.

Funktio

Matemaattisesti määriteltynä funktio kuvaa alkioiden tai olioiden välistä riippuvuutta. Se voi olla sääntö, jonka mukaan toinen alkio liitetään toisen joukon alkioon [1].

Määritelmä 1.1. *Olkoot A ja B joukkoja. Funktio eli kuvaus joukolta A joukolle B on sääntö, joka liittää jokaisen joukon A alkioon täsmälleen yhden joukon B alkion. Tätä funktiota merkitään*

$$f : A \to B$$

(Lähde: [2])

Esimerkki 1.1. *Matemaattinen funktio:*

$$f : \mathbb{R} \to \mathbb{R}, f(x) = x^2 + 7$$

Funktion f syöte on reaaliluku x, ja tulos on $x^2 + 7$.

Ohjelmoinnissa funktio toimii samalla tavalla: se ottaa syötteen, suorittaa sille määritellyn laskennan ja palauttaa tuloksen. R-kielessä funktio voidaan määritellä ohjelmoimalla ja tallentamalla sen kuvaus muuttujaan. Tehdään nyt sama esimerkki, mutta ohjelmoiden R-kielessä.

(Lähde: [3])

Esimerkki 1.2. *R-kielen funktio:*

```
f <- function(x) {x^2+7}
f(3)
# [1] 16
```

Esimerkissä 1.2 määritellään esimerkissä 1.1 annettu funktio R-kielessä. Funktio tallennetaan muuttujaan `f`, jonka jälkeen sitä kutsutaan komennolla $f(x)$, jossa x korvataan halutulla syötteellä, eli tässä tapauksessa luvulla 3.

Funktion f määritelmä on nyt ohjelman muistissa.

Ohjelma

Ohjelma on tarkasti määritelty suunnitelma tai menettely tietokoneella. Suorittaessa se tuottaa halutun analyysin tai laskennallisen tuloksen. Ideana on ratkaista jokin ongelma. Ohjelman suoritus perustuu tietokoneen muistiin tallennettuihin olioihin, kuten muuttujiin ja funktioihin [4].

Ensimmäisen luokan funktiot

Ensimmäisen luokan funktiot tarkoittavat sitä, että funktiot ovat ohjelmassa samanarvoisia muiden muuttujien kanssa. R-kielessä kaikki funktiot ovat ensimmäisen luokan funktioita. Kuten esimerkissä 1.2, funktio voidaan tallentaa muuttujaan ja sitä voidaan käyttää myöhemmin ohjelmassa kuten mitä tahansa muuta muuttujaa [5].

Tämä ominaisuus yhdistyy ohjelman tilaan, joka määritellään seuraavaksi.

Tila

Tila (state) tarkoittaa ohjelman suorituksen aikaista hetkellistä tilannetta. Se määriytyy muistissa olevien muuttujien, tietorakenteiden ja muiden tallennettujen arvojen perusteella. Tila kuvaa kokonaisuutta, joka sisältää kaiken sen tiedon, jonka perusteella ohjelma jatkaa suoritustaan seuraavaan vaiheeseen. Ohjelman tila muuttuu, kun se suorittaa käskyn, kuten arvojen sijoittamisen muuttujiin [6, 7].

Funktion tilattomuus

Nimensä mukaisesti tilattomuus tarkoittaa, ettei funktiossa ole syötteen jälkeen eikä sitä ennen tilaa. Tilaton funktio on sellainen, jonka tulos ei riipu ohjelman muusta tilasta, vaan sen syötteistä ja määritelmästä. Funktio ei muuta ulkoista tilaa. Tämän seurauksena toiminta on ennustettavaa. Myöskään funktiot eivät tallenna muistiin edellisiä koodeja, koska edellisillä tapahtumilla ei ole vaikutusta tuloksiin. [8].

Sivuvaikutus

Sivuvaikutus (side-effect) tarkoitta ohjelman tilan muuttamista funktion suorituksen yhteydessä. Sivuvaikutusten välttäminen tekee funktiosta ennustettavan ja helpottaa sen testaamista [3].

Arvon läpinäkyvyys

Arvon läpinäkyvyys tarkoittaa sitä, että funktion tulos riippuu ainoastaan sen syötteistä. Saamme aina saman tulosteen tietyllä syötteellä. Funktion kutsun voi korvata

sen palauttamalla arvolla ilman, että ohjelman käyttäytyminen muuttuu. Tämä on yksi tärkeistä funktionaalisen ohjelmoinnin käsitteistä ja erittäin tärkeä ohjelmakoodin luotettavuuden, ennustettavuuden ja testattavuuden kannalta [9].

Esimerkki 1.3. *Läpinäkyvyys ja ei-läpinäkyvyys esimerkki:*

```
increment <- function(x) { x + 1 }
increment(10)
# [1] 11
```

Tämä funktio on läpinäkyvä, koska sen tulos määräytyy vain sen syötteen `x` mukaan.

```
increment_opaque <- function(x) { x + spam }
```

[9]

Tämä funktio ei ole läpinäkyvä, koska se riippuu ulkoisesta muuttujasta `spam`. Tulos vaihtelee sen mukaan, mikä `spam` on.

Puhtaat funktiot

Puhtaat funktiot (pure functions) ovat funktioita, jotka täyttävät kaksi ehtoa: ne toteuttavat arvon läpinäkyvyyden ja niillä ei ole sivuvaikutuksia. Ulkoiset tekijät eivät vaikuta puhtaiden funktioiden tuloksiin [10].

Ennustettavuus

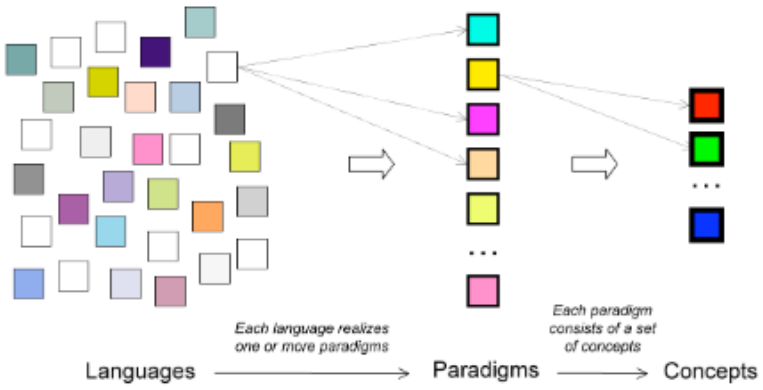
Ennustettavuus tässä kontekstissa tarkoittaa, että sivuvaikutuksia ei synny ja funktio saa samoilla syötteillä samoja tuloksia [8]. Toisin sanoen tiedetään etukäteen, mitä mahdollisesti kullakin funktiokutsulla tulostuu. Ennustettavuus on keskeistä ohjelman luotettavuuden ja analysoitavuuden kannalta.

Yhteenveto

Funktionaalisen ohjelmoinnin keskeiset käsitteet luovat loogisen kokonaisuuden. Näiden periaatteiden avulla ohjelmakoodista tulee luotettavaa, ennustettavaa ja helposti testattavaa.

1.2 Ohjelmointiparadigmat

Ohjelmointiparadigma tarkoittaa ohjelmoinnin ajattelutapaa, joka määrittelee, miten ohjelmat muodostetaan ja miten ohjelmoija hahmottaa laskennan rakenteen. Paradigmat voidaan ymmärtää kolmen näkökulman kautta: miten ohjelmassa käsitellään tietoa, miten ohjelman tila nähdään ja miten laskenta etenee [11].



Kuva 1: Kielet, paradigmat ja käsite. Lähde:[12]

Tärkeimpiä paradigmoja funktionaalisen ohjelmoinnin lisäksi ovat olio-ohjelmointi, proseduaalinen ja imperatiivinen ohjelmointi sekä monet muut [13]. Kukin paradigma painottaa erilaisia ohjelmoinnin näkökulmia, ja siksi paradigman valinta vaikuttaa ohjelman rakenteeseen.

Paradigman valinta on tärkeä tilastollisessa ohjelmoinnissa, jossa usein sama operaatio suoritetaan useille aineiston osille. Tilastollinen analyysi on toistuva, vaiheittainen ja usein monimutkainen prosessi. Aineistoa muokataan, analysoidaan, ryhmitellään, visualisoidaan ja raportoidaan [3, 14].

Seuraavaksi vertailen funktionaalista ja imperatiivista ohjelmointia lyhyesti, koska ne poikkeavat ajatusmalleiltaan toisistaan selvästi ja imperatiivinen ohjelmointi on edelleen yleisin käytetty paradigma [15].

Imperatiivinen ohjelmointiparadigma

Imperatiivinen ohjelmointiparadigma perustuu ohjelman suorittamiseen askel askeleelta. Toisin kuin funktionaalisessa ohjelmoinnissa, imperatiivisessa ohjelmoinnissa sivuvaikutukset ovat sallittuja, mikä on keskeinen osa ohjelman toimintaa. Ohjelma rakentuu selkeästi määriteltyjen käskyjen varaan, jotka määrittelevät, miten ja missä järjestyksessä tilaa muutetaan [3, 15].

Imperatiivisessa ohjelmoinissa toistuvat tehtävät toteutetaan perinteisten silmukoiden, kuten `for` ja `while` sekä `if-else` -ehtolauseiden avulla. Imperatiivisen ohjelmoinnin vahvuuksia ovat helppo opittavuus, sen suoraviivaisuus ja tarkka kontrolli suorituksen kulusta. Sen heikkouksia ovat modulaarisuuden puute, monimutkaisuuden hallinnan vaikeus ja sivuvaikutusten hallinnan haastavuus [6, 16].

Esitellään kolme keskeisintä piirrettä esimerkkien avulla.

Esimerkki 1.4. *Tilan muutos:*

```
sum<-0
```

```
for (i in 1:5) {
sum<-sum+1
}
print(sum)
# [1] 15
```

Tila muuttuu, kun muuttuja `sum` määritellään, sitten silmukan jokaisella askeleella sitä muutetaan jälleen.

Esimerkki 1.5. *Sivuvaikutukset:*

```
total <- 0
add_to_total <- function(x) {
    total <<- total + x
}
add_to_total(5)
print(total)
# [1] 5
```

Funktio `add_to_total` aiheuttaa sivuvaikutuksen, koska se muuttaa ulkoista muuttujaa `total` -operaation avulla. Funktio ei siis palauta pelkästään arvoa, vaan vaikuttaa ohjelman tilaan sen ulkopuolella.

Esimerkki 1.6. *Toistuvat tehtävät silmukoilla:*

```
for (i in 1:3) {
    print(i^2)
}
# [1] 1
# [1] 4
# [1] 9
```

Imperatiivisessa ohjelmoinnissa toistuvat tehtävät suoritetaan usein silmukoiden avulla, minkä vuoksi koodia voi olla hankala lukea ja hahmottaa.

Funktionaalinen ohjelmointiparadigma

Funktionaalisessa ohjelmointiparadigmassa tavoitteena on kirjoittaa luotettavia ja toistettavia funktioita, jotka soveltuvat erilaisiin laskennallisiin malleihin. Funktionaalinen ohjelmointi perustuu ohjelman tehtävien pilkkomiseen funktioihin, jotka voidaan yhdistää monimutkaisempien laskentojen toteuttamiseen.

Funktiot ovat ensimmäisen luokan objekteja ja usein puhtaita, eli niiden tulos riippuu vain syötteistä eivätkä ne tuota sivuvaikutuksia [17]. Tämä mahdollistaa korkean tason abstraktiot ja funktionaalisten rakenteiden, kuten `map` ja `apply` hyödyntäminen. Voidaan todeta, että funktioiden ei välttämättä tarvitse olla täysin puhtaita, jotta ne soveltuisivat funktionaaliseen ohjelmointiin.

Esitellään kolme keskeistä piirrettä esimerkkien avulla.

Kuten luvussa 1.1 esitetty `increment`-funktio (esimerkki 1.3) havainnollistaa, funktion tulos määräytyy ainoastaan syötteistä, eikä se vaikuta ohjelman muuhun tilaan.

Esimerkki 1.7. *Ensimmäisen luokan funktiot:*

```
apply_twice <- function(f, x) { f(f(x)) }
square <- function(y) { y^2 }
apply_twice(square, 2)
# [1] 16
```

Funktiot voidaan tallentaa muuttujiin ja välittää toisille funktioille; funktio voidaan suorittaa useita kertoja ketjutettuna.

Esimerkki 1.8. *Tilattomuus:*

```
sum_function <- function(a,b) {
sum(a:b)
}
sum_function(1,5)
# [1] 15
```

Funktio `sum_function` laskee annettujen lukujen `a:b` väliltä summan. Funktio on tilaton, eli se tuottaa aina saman tuloksen samoilla syötteillä. Lisäksi esimerkki 1.8 tarjoaa funktionaalisen tavan toteuttaa esimerkin 1.4 tehtävä.

Luvussa 1.1 määritelty ohjelman tilan käsite muodostaa keskeisen lähtökohdan imperatiivisen ja funktionaalisen ohjelmointiparadigman vertailulle. Imperatiivinen ohjelmointi nojaa ohjelman tilan muuttamiseen osana suorituksen etenemistä, kun taas funktionaalisessa ohjelmoinnissa laskenta pyritään ilmaisemaan ilman tilamuutoksia eli sivuvaikutuksia [3, 6, 7].

Kuvassa 1 eri ohjelmointikielet (languages) on esitetty erivärisinä neliöinä. Jokainen kieli voi toteuttaa yhden tai useamman paradigman. Ohjelmointiparadigmat (paradigms) muodostuvat eri käsitteitä (concepts). Paradigmat yhdistävät kielten toteuttamia ominaisuuksia. Paradigmat sisältävät käsitteitä, kuten tilan muuttaminen, funktiot, puhtaat funktiot ja silmukat.

Tämä kyseinen kuva auttaa ymmärtämään, miten ohjelmointikielet, niiden käyttämät paradigmat ja käsitteet liittyvät hierarkkisesti toisiinsa.

1.3 Funktionaalisen ohjelmointiparadigman huonot puolet

Funktionaallisella ohjelmonnilla on joitakin heikkouksia R-kielen käytössä. R-kieli ei ole täysin funktionaalinen, ja sivuvaikutuksia voi esiintyä esimerkiksi tuloksissa. Kuten esimerkissä 1.5 nähtiin, ohjelmoija voi helposti käyttää globaaleja muuttujia tai muokata objekteja, mikä aiheuttaa sivuvaikutuksia ja tekee lopputuloksen ennustettavuudesta epävarman. Globaalit muuttujat ovat funktion ulkopuolisia, ja R-kieli ei pakota täysin funktionaaliseen ohjelmointiin, jolloin vastuu jää ohjelmoijalle [17].

Haasteita ilmenee myös suorituskyvyssä. Funktionaalisessa ohjelmoinnissa luodaan usein uusia kopioita datasta sen sijaan, että sitä muokattaisiin suoraan. Jokainen kopiointi kuluttaa muistia ja heikentää suorituskykyä eristyisesti suurilla aineistoilla. Vaikka funktionaalisuus tekee R:n käytöstä ennustettavampaa ja turvallisempaa, se voi myös lisätä muistinkulutusta ja hidastaa suoritusta [17].

Funktionaalinen paradigma sopii SAC-strategiaan, joka esitellään seuraavaksi.

2 Split–Apply–Combine -strategia

Tässä luvussa esitellään jaa–sovita–yhdistä-strategia (eng. split–apply–combine), jota tässä tutkielmassa lyhennetään SAC-strategiaksi. SAC-strategia tunnetaan laajalti R-kielen data-analyysin teoreettisena perustana ja sitä hyödynnetään tilanteissa, joissa suuri aineisto halutaan jakaa pienempiin osiin ja suorittaa samoja laskentavaiheita jokaiselle osalle. Luvussa esitellään strategian periaatteet, historia, ajattelutapa sekä sen merkitys tilastollisessa ohjelmoinnissa. Lähteinä hyödynnetään pääosin Hadley Wickhamin artikkelia [18], mikä kertoo SAC-strategiasta.

2.1 Strategian yleiskuvaus

SAC-strategia on data-analyysin malli, jossa suuri ongelma pilkotaan pienempiin osiin. Pienemmissä osissa dataa voidaan käsitellä helpommin ja yksityiskohtaisemmin. Lopuksi nämä osat liitetään takaisin yhteen yhdeksi kokonaisuudeksi.

Strategia vastaa yleiseen ongelmaan data-analyysissä: usein aineistot ovat rakenteellisia, jaettavissa luonnollisiin osiin ja analyytikot haluavat toistaa samat operaatiot näille osille. Perinteiset ohjelmointitavat eivät tarjoa selkeää tapaa ilmaista toistuvaa rakennetta, mikä voi johtaa monimutkaisuuteen ja virheisiin. SAC-strategia tarjoaa systemaattisen ja helpommin hallittavamman mallin.

Strategia sai alkunsa vasta 2010-luvulla, mutta sen periaatteet ovat olleet osa R-kielen ja sen edeltäjän S:n ohjelmointia jo paljon aiemmin. R:n monet klassiset funktiot, kuten `tapply`, `by` ja `aggregate` noudattavat samaa logiikkaan kuin SAC-strategia. Strategia sai selkeän teoreettisen muodon 2010-luvulla, ja sen terminologia on vakiitunut osaksi R:n ja sen eri pakettien perustaa. Tämä historiallinen kehitys osoittaa, että SAC-strategia rakentuu vakiintuneiden käytäntöjen päälle ja tarjoaa systemaattisen lähestymistavan monimutkaisten rakenteiden käsittelyyn.

Strategian kolmivaiheinen rakenne voidaan nähdä siltana datan käsittelyn ja funktionaalisen ohjelmoinnin välillä. Tätä ei tule ymmärtää siten, että SAC-strategia olisi tarkasti määritelty menetelmä, vaan se toimii yleisenä paradigmana, joka ohjaa kohti toistettavuutta ja järjestelmällistä datan käsittelyä.

Useat R-kielen menetelmät tarjoavat keinoja monimutkaisiin analyyseihin. Strategian toimivuutta selittää sen yksinkertaisuus, yhteensopivuus ja johdonmukaisuus funktionaalisen ohjelmoinnin kanssa. Se mahdollistaa monimutkaisen datankäsittelyn rakentamisen yksinkertaisista osista ilman, että analyysi menettää selkeytensä.

Tilastollisessa ohjelmoinnissa strategian käyttö on luontevaa, koska melkein aina analyysi tehdään ryhmittäin ja ne noudattavat toistuvaa rakenteellista kaavaa. Toistuva rakenteellisuus näkyy esim. eri tunnuslukujen laskennassa, ryhmävertailussa sekä mallien sovittamisessa. SAC ajattelutapa tekee ohjelmoinnista selkeää, toistettavaa ja virheettömämpää.

Esimerkkinä tällaisesta ajattelusta toimii hyvin kouluarvosanojen analysointi. Aineisto voidaan jakaa ensin luokittain (split) jokaiselle luokalle lasketaan keskiarvo (apply) ja lopuksi verrataan luokkien keskiarvoja keskenään (combine). Tämä osoittaa, kuinka SAC-strategia auttaa tekemään analyysistä hyvin selkeän, vaiheittaisen ja toistettavan.

2.2 Strategian toiminta

SAC-strategia toimii erityisen hyvin R-kieleen, koska se hyödyntää kielen funktionaalista ja vektoripohjaista rakennetta. Strategian perusidea on jakaa aineisto osiin, soveltaa funktioita jokaiseen osaan ja yhdistää tulokset takaisin kokonaisuudeksi. R:ssä laskenta perustuu vektoreiden, listojen tai datakehikkojen käsittelyyn yhtenäisinä operaatioina.

Seuraavaksi kuvataan SAC-strategian vaiheet yleisellä tasolla tämän tutkielman empiirisen sovelluksen kontekstissa. Luvussa 4 hyödynnän strategiaa analysoimaan, miten tehtyjen pisteiden määrä riippuu muista muuttujista NBA:n joukkueiden tasolla. Toteutetan analyysin joukkuetasolla, mikä tekee SAC-strategiasta soveltuvan tämän tyyppiseen tutkimukseen.

Split

Split-vaiheessa aineisto jaetaan pieniin osiin, joita voidaan käsitellä erikseen. Jaottelu voi perustua esim. kategorisiin muuttujiin, ajalliseen rakenteeseen tai mihin tahansa muuhun dataan liittyvään ryhmittelyyn. Empiirisessä osassa aineisto jaetaan (split) NBA joukkueiden mukaan eli jokainen joukkue muodostaa oman osansa. Tämä mahdollistaa analyysin suorittamisen erikseen jokaiselle joukkueelle samalla tavalla, mikä tekee tuloksista vertailukelpoisia.

Apply

Apply-operaatio voidaan ajatella olevan koko strategian ydin, koska tässä tehdään mallien sovitukset ja laskutoimitukset. Tehtävä operaatio määritellään vain kerran funktiona ja tätä käytetään kaikille osille. Funktion tulisi olla mieluiten puhdas, jolloin ei synny sivuvaikutuksia ja voidaan helposti toistaa muille osille. Empiirisessä osiossa sovitettiin lineaarinen regressiomalli jokaiselle osalle erikseen ja kerätään halutut tunnusluvut sekä palautetaan ne.

Combine

Combine-vaiheessa kootaan edellisessä vaiheessa saadut tulokset yhteen niin, että voidaan tulkita ja muodostaa jatkoanalyysiä niille. Vaihe voidaan tehdä yksinkertaisesti, esim. luomalla taulukko tai lista. Tässä vaiheessa on tärkeää huomioida tulosten rakenne ja muoto, jotta ne voidaan yhdistää oikealla tavalla. Empiirisessä osassa tehtiin yksinkertainen taulukko ja visualisoitiin se.

3 R-kielen keinot

Luvussa kolme esitellään R-kielen keinoja avaamalla tarkemmin luvussa 1.3 mainittuja datapaketteja ja näihin liittyviä funktioita. Lähteinä käytetään [3], [21], [14], [22] ja [23]. R tarjoaa useita keinoja toteutaa funktionaalista ohjelmointia. Monimutkaiset toistotoiminnot voidaan kirjoittaa lyhyemmin ja selkeämmin.

3.1 Base R ja apply-perhe

`Base R` eli suomeksi `perus-R` tarkoittaa R-kielen ydinosaa. R:ssä on sisäänrakennettu työkalupakki, joka sisältää perusfunktioita, kuten `mean(), sum(), length(), sort()`, tietorakenteita, kuten matriisit, vektorit, listat, datakehikko -rakenteet sekä perusohjelmointirakenteita, kuten `for, if, while`.

Näiden lisäksi `perus-R` tarjoaa useita funktionaalisen ohjelmoinnin toteutukseen sopivia ratkaisuja, kuten `apply`-funktioperheen `(apply(), lapply(), tapply())` [3]. Nämä funktiot mahdollistavat toistojen tekemisen ilman perinteisiä silmukoita eli for-silmukoita. Koodista tulee tiiviimpää ja sivuvaikutusten riski vähenee.

Apply-perhe edustaa R-kielen tapaa ilmaista toistoa funktionaalisesti: funktiot toimivat argumentteina muille funktioille ja palauttavat uusia arvoja muuttamalla alkuperäistä dataa [3]. Lähestymistapa korostaa R:n kykyä yhdistää tilastollinen laskenta ja funktionaalinen ohjelmointiparadigma.

Esimerkki 3.1. *R:n `apply()`-funktio [21] mahdollistaa funktion soveltamisen matriisin riveihin tai sarakkeisiin.*

```
x <- matrix(c(3,1,2,6,4,5), nrow=3)
x
#      [,1] [,2]
# [1,]    3    6
# [2,]    1    4
# [3,]    2    5

apply(x, 2, sort)
#      [,1] [,2]
# [1,]    1    4
# [2,]    2    5
# [3,]    3    6
```

Tässä `apply()` -funktiolle annetaan kolme argumenttia: dataobjekti `x`, ulottuvuusargumentti `2`, joka viittaa sarakkeisiin ja funktio `sort`, jota sovelletaan jokaiseen sarakkeeseen. Funktio järjestää tiedot – tässä tapauksessa matriisin `x` jokaisen sarakkeen arvot nousevaan järjestykseen – ja tuloksena syntyy uusi matriisi.

3.2 Purrr ja dplyr -paketit

R-kielessä on datan käsittelyyn suunnattuja paketteja, kuten `dplyr` ja `purrr`, jotka pohjautuvat luvussa 2 esitettyyn strategiaan [3]. `Purrr` ja `dplyr` tarjoavat selkeämmän, modernin ja turvallisemman tavan toteuttaa samoja operaatioita, kuin edellisessä luvussa esitelty `apply`-perhe. `Perus R`:n `apply`-perhe nojaa joustavaan ja tulostyypiltään usein arvaamattomaan toiminnallisuuteen [3, 14].

`Purrr` parantaa funktionaalisen ohjelmoinnin työkaluja R:ssä tarjoamalla johdonmukaisia työkalusarjoja käsittelemään funktioita ja vektoreita. Sen avulla voidaaan korvata suurin osa `apply`-perheen funktioista, mutta tulokset palautuvat aina `list`- tai `tabble`-muotoisina rakenteina. Tämä vähentää virheitä ja tekee koodista luotettavampaa [22].

Map

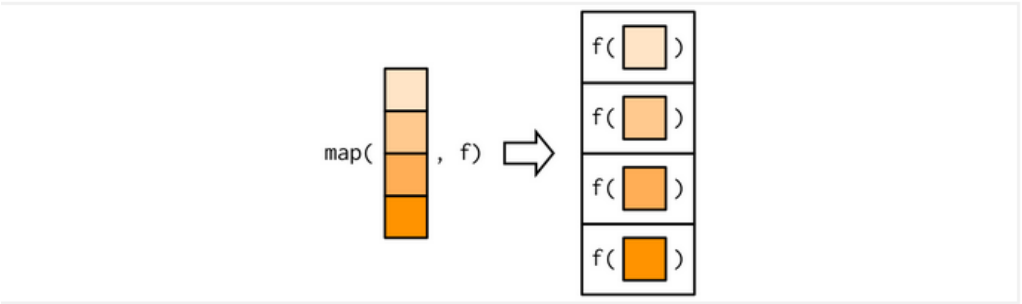
`Purrr`-paketin keskeinen `map`-funktio ottaa syötteenä vektorin tai listan ja soveltaa annettua funktiota jokaiseen elementtiin. Kuva 2 mallintaa tilannetta. Purrr-paketin etuna on, että sen tulostyyppi on aina ennustettavissa: esim. `map` palauttaa listan,

`map-dbl` numeerisen vektorin ja `map-chr` merkkivektorin. Tämä vähentää virheitä ja tekee koodista luotettavampaa [3, 21].

`Dplyr` puolestaan toteuttaa SAC-strategian datakehikon tasolla. Funktio `group_by()` mahdollistaa datan ryhmittelyn haluttujen muuttujien perusteella ja `summarise()` -funktiolla voidaan laskea ryhmäkohtaiset tunnusluvut tai muita arvoja. Ryhmittelystä tulee selkeää, loogista ja helposti tulkittavaa ja korvaa monin tavoin esim. perus-R:n `tapply` -funktion [23].

Lisäksi `dplyr` tarjoaa ryhmäkohtaiset funktiot `group_map()` ja `group_modify()`, joita hyödynnetään empiirisessä kokeessa seuraavassa luvussa. `group_map()` soveltaa annettua funktiota jokaiselle ryhmälle ja palauttaa listan tuloksista. Funktio saa argumenteikseen ryhmän datan ja avainmuuttujat. Tämä toiminto on hyödyllistä, kun halutaan tehdä ryhmäkohtaisia laskelmia ilman, että automaattisesti tulokset yhdistetään datakehikoksi. `group_modify()` toimii samalla tavalla kuin `group_map()`, mutta palauttaa yhdistetyn datakehikon, mikä mahdollistaa ryhmäkohtaisten rivien muokkaamisen tai luomisen ja säilyttää datakehikon rakenteen analyysiä varten [23].

Pakettien avulla strategia toteutuu koodissa läpinäkyvästi ja analyyttinen ajattelutapa näkyy ohjelmakoodin rakenteessa.



Kuva 2: Esimerkki map-funktiosta. Lähde:[24]

Esimerkki 3.2. *map-funktio:*

```
joukkueet <- list(
    Lakers = c(25, 28, 30, 22),
    Celtics = c(27, 29, 24, 26)
)

keski_ika <- map(joukkueet, mean)
keski_ika
# $Lakers
# [1] 26.25
# $Celtics
# [1] 26.5
```

Esimerkissä funktio käy läpi jokaisen joukkueen vektorin ja laskee keski-iän. Tuloksena on lista, jossa on kunkin joukkueen keski-ikä.

4 Empiirinen sovellus

Tässä osiossa sovelletaan aiemmin esitelty SAC-strategiaa NBA:n kauden 2023-2024 pelaajadataan. Tavoitteena on havainnollistaa, miten pelaajien iät, peliminuutit ja tilastot kuten pisteet ja syötöt, vaikuttavat joukkueiden sisällä. Esittelen kaksi mentelmää: `dplyr` ja `perus-R` ja lopuksi kootaan tulokset.

4.1 Aineiston esittely

Data on kerätty lähteestä [25] ja ladattu taulukoksi R studioon. Taulukossa on 572:n pelaajan 30 joukkueesta tilastot NBA:n tavalliselta pelikaudelta ajalta 2023-2024. NBA on Pohjois-Amerikassa pelattava koripallosarja, jossa palkitaan kauden loputtua voittajajoukkue ja parhaat pelaajat pelikohtaisesti. Koripallossa kerätään tilastoja pelaajista, jotta voidaan seurata yksittäisten pelaajien kehitystä ja vertailla kauden parhaita pelaajia keskenään.

Kerätyssä datassa on 29 eri muuttujaa, joista olen kiinnostunut vain viidestä. Tarkasteluksi on otettu henkilökohtaiset peliminuutit (Min), tehdyt pisteet (PTS), syöttöpisteet ennen koria (AST) ja ikä (Age). Näiden muuttujien avulla tarkastellaan pelaajien iän ja peliajan yhteyttä henkilökohtaisiin pisteisiin. Vertailen joukkueiden sisäisiä eroja käyttämällä SAC-strategiaa.

Aineisto on havainnollistettu R-studiossa havaintomatriisiksi, jossa vaakariveillä on pelaajat ja pystyriveillä eli sarakkeilla ovat tilastomuuttujat. Havaintomatriisi on taulukko, josta lasketaan tunnuslukuja kuten keskiarvoja ja piirretään graafisia esityksiä [20]. Aineistossa kukin rivi vastaa yhtä pelaajaa ja sarakkeet kuvaavat heidän ominaisuuksia kaudelta 2023-2024.

Taulukossa 1 esitetään aineiston keskeiset deskriptiiviset tunnusluvut joukkueittain. Raportoin mm. pelaajien lukumäärän sekä valittujen muuttujien (peliminuutit, pisteet, syötöt ja ikä) keskiarvot ja keskihajonnat. Tunnusluvut antavat yleiskuvan joukkueiden välisistä eroista sekä muuttujien mittakaavasta ennen varsinaista mallintamista.

Taulukko 1: Deskriptiiviset tunnusluvut joukkueittain (pyöristetty yhteen desimaaliin)

Team	n	$\bar{\mu}_{\text{Min}}$	$\widehat{\sigma}_{\text{Min}}$	$\bar{\mu}_{\text{PTS}}$	$\widehat{\sigma}_{\text{PTS}}$	$\bar{\mu}_{\text{AST}}$	$\widehat{\sigma}_{\text{AST}}$	$\bar{\mu}_{\text{Age}}$	$\widehat{\sigma}_{\text{Age}}$
ATL	18	20.8	10.0	8.8	7.4	2.7	2.7	27.2	3.8
BOS	18	22.7	9.5	9.0	6.5	2.5	2.2	25.5	3.0
BKN	19	20.2	9.1	7.6	5.0	2.0	2.0	26.9	3.4
CHA	17	20.0	9.6	8.0	6.2	2.7	2.5	25.8	3.1
CHI	19	19.9	9.3	7.9	5.7	2.1	2.0	26.7	3.4
CLE	17	19.9	9.8	7.4	5.5	2.0	1.9	26.5	3.3
DAL	19	20.3	9.9	8.2	6.3	2.0	2.2	26.7	3.4
DEN	17	18.7	9.1	7.4	5.1	2.1	1.9	25.8	3.2
DET	17	20.3	9.7	7.7	5.4	2.0	2.1	25.9	3.3
GSW	18	18.9	9.3	7.6	5.9	2.4	2.4	26.1	3.3
HOU	19	19.1	9.2	7.8	5.7	2.0	2.0	26.4	3.1
IND	18	19.7	9.7	7.9	5.9	2.0	2.0	26.3	3.2
LAC	19	19.9	9.3	8.2	6.2	2.1	2.2	27.0	3.4
LAL	19	19.1	9.6	7.4	5.7	2.0	2.0	26.7	3.3
MEM	18	19.5	9.4	7.7	5.7	2.1	2.1	26.2	3.3
MIA	19	19.6	9.5	7.9	5.8	2.1	2.1	26.9	3.3
MIL	19	19.0	9.2	7.7	5.5	2.1	2.1	26.1	3.3
MIN	19	19.4	9.5	7.7	5.6	2.1	2.1	26.0	3.2
NOP	19	19.3	9.5	7.8	5.6	2.1	2.1	26.3	3.3
NYK	19	19.1	9.3	7.8	5.6	2.1	2.1	26.3	3.3
OKC	18	19.0	9.4	7.9	5.7	2.1	2.1	26.4	3.4
ORL	18	19.2	9.4	7.7	5.6	2.1	2.1	26.1	3.2
PHI	18	19.1	9.4	7.7	5.6	2.1	2.1	26.2	3.3
PHX	19	19.3	9.5	7.8	5.7	2.1	2.1	26.3	3.3
POR	18	19.2	9.5	7.8	5.7	2.1	2.1	26.4	3.3
SAC	18	19.3	9.5	7.8	5.7	2.1	2.1	26.4	3.4
SAS	19	19.2	9.4	7.8	5.7	2.1	2.1	26.2	3.3
TOR	18	19.1	9.4	7.7	5.6	2.1	2.1	26.2	3.3
UTA	19	19.2	9.4	7.8	5.7	2.1	2.1	26.3	3.3
WAS	19	19.1	9.3	7.7	5.6	2.1	2.1	26.3	3.3

Taulukossa μ tarkoittaa keskiarvoa ja σ keskihajontaa.

4.2 Aineiston mallintaminen ja tutkimuskysymykset

Aineisto voidaan esittää havaintomatriisina $X \in \mathbb{R}^{n \times p}$, jossa on 572 pelaajaa ja viisi muuttujaa: joukkue (Team), peliminuutit (Min), pisteet (PTS), syötöt (AST) ja ikä (Age).

Aineisto jaetaan joukkueittain, jotta voidaan tarkastella riippuvuuksia joukkuekoh-

taisesti. Sovitettu malli on lineaarinen regressio, jossa pelaajan tehtyjä pisteitä selitetään peliminuuteilla, syöttöjen määrällä ja iällä. Regressiokertoimet kuvaavat selittävien muuttujien ja pisteiden tilastollista yhteyttä.

Tutkimuskysymykset:

1. Missä joukkueessa syöttöjen määrällä (AST) on voimakkain vaikutus pisteisiin?

2. Missä joukkueessa peliminuuttien määrällä (Min) on voimakkaimmin vaikutus pisteisiin?

3. Onko pelaajien ikä (Age) tilastollisesti merkitsevä selittäjä pisteiden määrälle?

Aineiston käsittelyssä käytettiin seuraavaa mentelmää: aineisto jaettiin joukkueittain (split), regressiomalli sovitettiin jokaiselle joukkueelle erikseen (apply) ja tulokset yhdistettiin analysoitavaksi (combine).

Sovitettu malli on muotoa:

$$\text{PTS} = \beta_0 + \beta_1 \cdot \text{Min} + \beta_2 \cdot \text{AST} + \beta_3 \cdot \text{Age} + \varepsilon$$

Tässä tutkielmassa regressiomallin sovittamista ei ole arvioitu kattavasti diagnostisten tarkastelujen kautta, vaan keskitytään pelkästään mallin sovittamiseen ja tulosten tulkintaan. Mallin diagnostiset tarkastelut on rajattu työn ulkopuolelle työn laajuuden vuoksi. Regressiokertoimien avulla voidaan arvioida selittävien muuttujien yhteyttä pisteiden määrään joukkuekohtaisesti. Regressiomalli sovitettiin pienimmän neliösumman menetelmällä (PNS). Kun kertoimille laitetaan hattu päälle, puhutaan saadusta estimaatista. Peliminuuttien regressiokerroin $\hat{\beta}_1$ kuvaa, kuinka paljon pelaajan odotetaan tekevän pisteitä yhden peliminuutin aikana muiden muuttujien pysyessä vakioina. Syöttöjen kerroin $\hat{\beta}_2$ ilmaisee, kuinka voimakkaasti syöttöjen määrä on yhteydessä tehtyihin pisteisiin. Iän kerroin $\hat{\beta}_3$ ilmaisee, liittyykö pelaajan ikään positiivinen vai negatiivine yhteys pisteisiin. Kaikkien estimaattien kohdalta tutkitaan myös joukkuekohtaista tilastollista merkitsevyyttä.

4.3 Tulokset

Tässä luvussa käydään läpi kaksi menetelmää: `dplyr` ja `perus-R`. Lopussa tuloksia tulkitaan suhteessa asetettuihin tutkimuskysymyksiin ja arvioidaan, millä tavoin selittävien muuttujien vaikutukset vaihtelevat joukkueiden välillä. Regressiokertoimia tarkastellaan sekä numeerisesti että visualisoimalla ne pistediagrammeina, jotka mahdollistavat joukkueiden välisen vertailun.

`Dplyr`:

```
library(dplyr)
library(purrr)
```

```
library(broom)
# liitetään csv-tiedosto muuttujaksi data
data <- read_csv("nbastats.csv")

# Valitaan tarvittavat muuttujat, 5 kpl
# Tämä muodostaa havaintomatriisin
data_small<-data %>%
  select(Team, Min, PTS, AST, Age)

# Tarkistetaan ensimmäiset rivit
head(data$Player)

# SPLIT:
# Aineisto jaetaan joukkueittain group_by()-funktion
   avulla.
# dplyr toteuttaa jaon implisiittisesti ilman split()-
   funktiota.
# Jokainen ryhmä vastaa yhtä NBA-joukkuetta.
data_grouped <- data_small %>%
  group_by(Team)

# APPLY:
# Funktio, joka sovittaa lineaarisen regressiomallin
# erikseen jokaiselle joukkueelle
fit_model<-function(df) {
  lm(PTS~Min+AST+Age,data=df) %>% tidy() }

# Regressiomalli sovitetaan jokaiselle joukkueelle
   erikseen
# group_modify() ajaa funktion jokaiselle ryhmälle
results <- data_grouped %>%
  group_modify(~ fit_model(.x))

# COMBINE:
# group_modify() yhdistää tulokset automaattisesti
# yhdeksi datakehikoksi
results <- results %>%
  ungroup()
```

Perus-R:

```
data <- read_csv("nbastats.csv")

# Valitaan tarvittavat muuttujat, havaintomatriisi
data_small<-data %>%
  select(Team, Min, PTS, AST, Age)
```

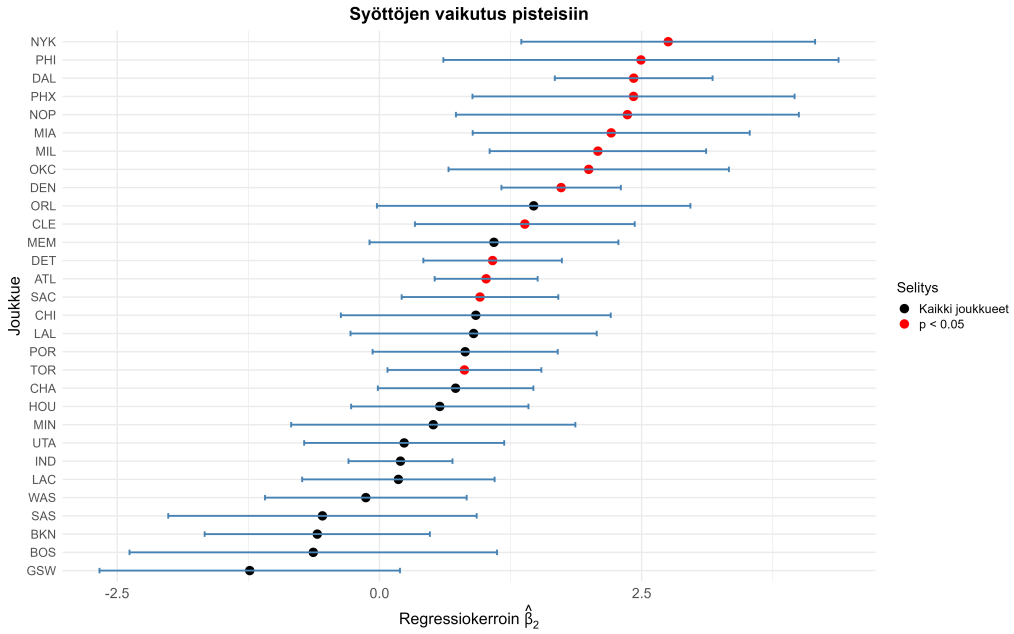
```
# Split: Jaa aineisto joukkueittain
# Jokainen listan alkio sisältää yhden joukkueen pelaajat

teams_list <- split(data_small, data_small$Team)

# Apply:
# Funktio, joka sovittaa lineaarisen mallin
# jokaiselle joukkueelle erikseen
fit_model_base <- function(df) {
  model <- lm(PTS ~ Min + AST + Age, data = df)
  summary_model <- summary(model)

# Kerätään beta-kertoimet, p-arvot ja termien nimet data.
   frameen
data.frame(
    term = rownames(summary_model$coefficients),
    estimate = summary_model$coefficients[, "Estimate"],
    p_value = summary_model$coefficients[, "Pr(>|t|)"],
    row.names = NULL
  )
}
# Regressiomalli sovitetaan jokaiselle joukkueelle
sovite_lista <- lapply(teams_list, fit_model_base)

# Combine: Yhdistä tulokset yhteen data.frameen
results_base <- do.call(
  rbind,
  lapply(names(sovite_lista), function(team) {
    out <- sovite_lista[[team]]
    out$Team <- team
    out
  })
)
```

Syöttöjen vaikutus pisteisiin:

Syöttöjen määrällä havaittiin positiivinen yhteys pisteiden määrässä useissa joukkueissa. Suurimmat syöttöjen regressiokertoimien estimaatit havaittiin joukkueilla New York Knicks ($\hat{\beta}_2^{\text{NYK}} = 2.75$), Philadelphia 76ers ($\hat{\beta}_2^{\text{PHI}} = 2.49$), Dallas Mavericks ($\hat{\beta}_2^{\text{DAL}} = 2.42$), Phoenix Suns ($\hat{\beta}_2^{\text{PHX}} = 2.42$) ja New Orleans Pelicans ($\hat{\beta}_2^{\text{NOP}} = 2.36$). Näistä selkeästi voimakkain vaikutus oli New York Knicksillä.

Kuvassa 3 on esitetty kaikkien joukkueiden syöttöjen regressiokertoimien estimaatit $\hat{\beta}_2$ sekä niitä vastaavat 95%:n luottamusvälit (vaaleansininen jana). Punaisella merkityt pisteet kuvaavat joukkueita, joissa syöttöjen vaikutus pisteiden määrään oli tilastollisesti merkitsevä ($p<0.05$). Luottamusvälien perusteella voidaan havai-

ta, että useissa edellä mainituissa joukkueissa luottamusväli ei sisällä nollaa, mikä tukee tuloksen tilastollista merkitsevyyttä.

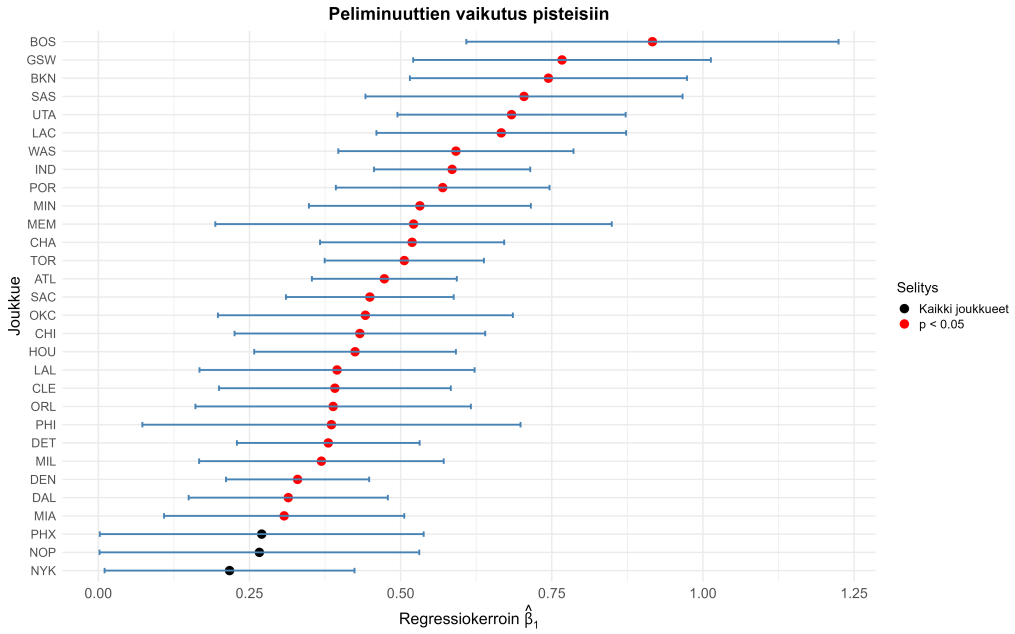


Kuva 3: Syöttöjen vaikutus pisteisiin

Peliminuuttien vaikutus pisteisiin:

Peliminuuttien regressiokertoimet olivat positiivisa kaikissa joukkueissa. Jyrkimmät regressiokertoimet peliminuuteille havaittiin Los Angeles Lakersilla (LAL), Los Angeles Clippersillä (LAC), Memphis Grizzliesillä (MEM) ja Cleveland Cavaliersilla (CLE). Näistä selvästi voimakkain vaikutus oli Los Angeles Lakersilla, mikä tarkoittaa, että pisteiden määrä kasvaa siellä minuutteja kohden kaikkein eniten.

Kuvassa 4 on esitetty kaikkien joukkueiden peliminuuttien regressiokertoimien estimaatit $\hat{\beta}_1$ sekä 95 %:n luottamusvälit. Useissa joukkueissa luottamusvälit eivät sisällä nollaa, mikä osoittaa peliminuuttien vaikutuksen olevan tilastollisesti merkitsevä ($p<0.05$).

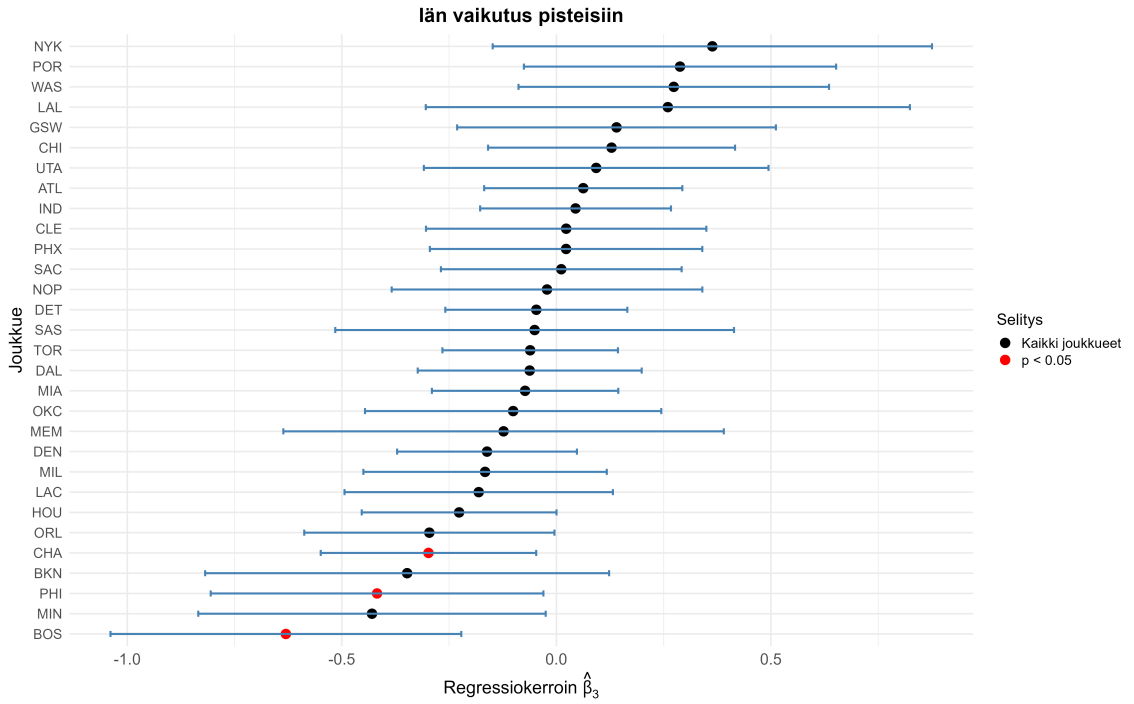


Kuva 4: Minuuttien vaikutus pisteisiin

Iän vaikutus pisteisiin:

Pelaajien iän vaikutus pisteiden määrään ei ollut tilastollisesti merkitsevä suurimmassa osassa joukkueita. Tilastollisesti merkitsevä yhteys havaittiin Boston Celticsin ($p^{\text{BOS}} = 0.00981$) Charlotte Hornetsin ($p^{\text{CHA}} = 0.0319$) ja Philadelphia 76ersissa ($p^{\text{PHI}} = 0.021$). Näissä joukkueissa iän regressiokertoimen estimaatin $\hat{\beta}_3$ 95 %:n luottamusväli ei sisältynyt nollaan.

Muissa joukkueissa iän yhteys pisteiden määrään ei ollut tilastolisesti merkitsevä. Kuvassa 5 on esitetty kaikkien joukkueiden iän regressiokertoimien estimaatit $\hat{\beta}_3$ sekä niitä vastaavat 95 %:n luottamusvälit.



Kuva 5: Iän vaikutus pisteisiin

4.4 Yhteenveto

Tässä empiirisessä osiossa tarkasteltiin pelaajien peliminuuttien, syötöjen ja iän yhteyttä tehtyihin pisteisiin NBA:n kauden 2023-2024 pelaaja-aineistossa joukkuekohtaisesti. Analyysi toteutettiin soveltamalla SAC-strategiaa, jossa lineaarinen regressiomalli sovitettiin erikseen jokaiselle joukkueelle.

Tulosten perusteella syöttöjen määrällä oli useissa joukkueissa positiivinen ja tilastollisesti merkitsevä yhteys pisteiden määrään. Suurimmat syöttöjen regressiokertoimien estimaatit havaittiin New York Knicksillä, Philadelphia 76ersilla, Dallas Mavericksillä, Phoenix Sunsilla ja New Orleans Pelicansilla. Näissä joukkueissa syöttöjen vaikutus pisteisiin oli tilastollisesti merkitsevä 5%:n riskitasolla.

Peliminuuttien vaikutus pisteisiin oli positiivinen kaikissa joukkueissa, ja useimmissa tapauksissa vaikutus oli myös tilastollisesti merkitsevä. Voimakkain peliminuuttien regressiokerroin havaittiin Los Angeles Lakersilla, mikä tarkoittaa, että peliminuuttien lisäys oli keskimäärin yhteydessä suurempaan pisteiden kasvuun kuin muissa joukkueissa.

Pelaajien iän vaikutuspisteisiin oli pääosin tilastollisesti ei-merkitsevä. Tilastollisesti merkitsevä yhteys havaittiin kuitenkin Boston Celticsissä, Charlotte Hornetissa ja Philadelphia 76ersissa. Näissä joukkueissa iän regressiokertoimien luottamusväli ei sisältänyt nollaa, mikä viittaa iän itsenäiseen vaikutukseen pisteiden määrään.

Kokonaisuutena tulokset osoittavat, että selittävien muuttujien vaikutukset pisteisiin vaihtelevat huomattavasti joukkueiden välillä. Tämä havainnollistaa SAC-strategian hyödyllisyyttä tilanteissa, joissa kiinnostuksen kohteena ovat ryhmäkohtaiset erot ja niiden tilastollinen tulkinta.

Viitteet

[1] Tieteen termipankki, The Helsinki term bank for the arts and sciences: https://tieteentermipankki.fi/wiki/Termipankki:Etusivu, luettu 13.5.2025

[2] K. Häkkinen. (1998). Matematiikan propedeuttinen kurssi. Jyväskylän yliopisto.

[3] H. Wickham. (2019). Functional Programming. Advanced R. https://adv-r.hadley.nz/

[4] Britannica Editors (2025, July 11). computer program. Encyclopedia Britannica. https://www.britannica.com/technology/computer-program

[5] Wikipedia: First-class function, https://en.wikipedia.org/wiki/First-class_function, luettu 6.1.2026

[6] B. Rodrigues. (2023) Functional programming. Building reproducible analytical pipelines with R. Part 1. https://raps-with-r.dev/fprog.html

[7] B. Rodrigues. (2025) Functional Programming. Building Reproducible Analytical Pipelines. https://rap4mads.eu/04-functional-programming.html

[8] Wikipedia: Funktional programming, https://en.wikipedia.org/wiki Functional programming, luettu 12.5.2025

[9] B. Rodrigues. (2022). Functional programming. Modern R with the tidyverse. https://modern-rstats.eu/functional-programming.html

[10] Wikipedia: Pure function. https://en.wikipedia.org/wiki. Luettu 15.12.2025.

[11] P. Van Roy. (2014). Programming paradigms for dummies: What every programmer should know. New computational paradigms for computer music, Vol 104, 616-621. https://webperso.info.ucl.ac.be/ pvr/VanRoyChapter.pdf

[12] P. Van Roy. (2014). Programming paradigms for dummies: What every programmer should know, figure 1. New computational paradigms for computer music, Vol 104, p. 7. https://webperso.info.ucl.ac.be/ pvr/VanRoyChapter.pdf

[13] Wikipedia: Ohjelmointiparadigma. https://en.wikipedia.org/wiki Luettu 12.5.2025

[14] H. Wickham, G. Grolemund. (2017). R for Data Science. O'Reilly Media. https://r4ds.hadley.nz/

[15] GeeksforGeeks, Difference between functional and imperative programming. https://www.geeksforgeeks.org/theory-of-computation/difference-between-functional-and-imperative-programming/

[16] Z. Hu, J. Hughes, ja M. Wang. (2015). How functional programming mattered. National Science Review, 2(3), 349–370. https://doi.org/10.1093/nsr/nwv042

[17] J. Chambers. (2014). Object-Oriented Programming, Functional Programming and R. Statistical Science, Vol 29, No 2, 167–180. https://www.jstor.org/stable/43288468

[18] H. Wickham. (2011). The Split-Apply-Combine Strategy for Data Analysis. Journal of Statistical Software, 40(1). https://doi.org/10.18637/jss.v040.i01

[19] J. Alho, E. Arjas, E. Läärä, P. Pere. Suomen tilastoseura, kustantaja. (2023). Tilastotieteen sanasto (2. laitos). Suomen Tilastoseura ry.

[20] Tietoarkisto, mittaaminen: havaintomatriisi - KvantiMOTV. https://www.fsd.tuni.fi/menetelmaopetus/mittaaminen/havaintomatriisi.html (luettu 5.10.2025)

[21] R Core Team.(2024). apply: Apply functions over array margins. Documentation of R package base. https://rdrr.io/r/base/apply.html

[22] H. Wickham, Henry L (2025). purrr: Functional Programming Tools. R package version 1.2.0, https://purrr.tidyverse.org/.

[23] H. Wickham, R. François, Henry L, K. Müller, D. Vaughan (2025). dplyr: A Grammar of Data Manipulation. R package version 1.1.4, https://dplyr.tidyverse.org.

[24] H. Wickham. (2019). Functional Programming. Advanced R, luvun 9.2 kuva 1 https://adv-r.hadley.nz/

[25] NBA: Player stats (2023-2024)

https://www.nba.com/stats/players/traditionalPlayerExperience. Regular season per game, kerätty 27.3.2025