KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FUKULTETAS

**Intelektikos pagrindai**

*(T120B029)*

**Laboratorinis darbas Nr. 4**

**Duomenų klasifikavimas K-vidurkių metodu**

Darbą atliko:  
IFF-7/14 gr. Studentas  
Eligijus Kiudys

Darbą priėmė:  
lekt. Andrius Nečiūnas  
doc. Agnė Paulauskaitė-Tarasevičienė

KAUNAS 2020

**Turinys**

[Darbo užduoties aprašas, įrankių pasirinkimas 3](#_Toc41051268)

[Įvestys bei išvestis 4](#_Toc41051269)

[Įvestys 4](#_Toc41051270)

[Išvestys 6](#_Toc41051271)

[Klasterizavimas į du klasterius 6](#_Toc41051272)

[Klasterizavimas į tris klasterius 7](#_Toc41051273)

[Klasterizavimas į keturis klasterius 9](#_Toc41051274)

[Modelio Analizė ir klasterių pasirinkimai 11](#_Toc41051275)

[Pirmas bandymas 11](#_Toc41051276)

[Antras bandymas 13](#_Toc41051277)

[Trečias bandymas 15](#_Toc41051278)

[Dalinė išvada 16](#_Toc41051279)

[Išvados 17](#_Toc41051280)

[Programos kodas 18](#_Toc41051281)

[Kodas naudojamas išvestims 18](#_Toc41051282)

[Kodas naudojamas modelio validacijai 22](#_Toc41051283)

[Pav. 1 Koreliacijos matrica 5](#_Toc41051284)

[Pav. 2 Duomenų klasterizavimas į du klasterius 6](#_Toc41051285)

[Pav. 3 Duomenų klasterizavimas į du klasterius 6](#_Toc41051286)

[Pav. 4 Duomenų klasterizavimas į du klasterius 7](#_Toc41051287)

[Pav. 5 Duomenų klasterizavimas į tris klasterius 7](#_Toc41051288)

[Pav. 6 Duomenų klasterizavimas į tris klasterius 8](#_Toc41051289)

[Pav. 7 Duomenų klasterizavimas į tris klasterius 9](#_Toc41051290)

[Pav. 8 Duomenų klasterizavimas į keturis klasterius 9](#_Toc41051291)

[Pav. 9 Duomenų klasterizavimas į keturis klasterius 10](#_Toc41051292)

[Pav. 10 Duomenų klasterizavimas į keturis klasterius 10](#_Toc41051293)

[Pav. 11 Pirmo bandymo analizės grafikas naudojant du klasterius 11](#_Toc41051294)

[Pav. 12 Pirmo bandymo analizės grafikas naudojant tris klasterius 12](#_Toc41051295)

[Pav. 13 Pirmo bandymo analizės grafikas naudojant keturis klasterius 12](#_Toc41051296)

[Pav. 14 Antro bandymo analizės grafikas naudojant du klasterius 13](#_Toc41051297)

[Pav. 15 Antro bandymo analizės grafikas naudojant tris klasterius 14](#_Toc41051298)

[Pav. 16 Antro bandymo analizės grafikas naudojant keturis klasterius 14](#_Toc41051299)

[Pav. 17 Trečio bandymo analizės grafikas naudojant du klasterius 15](#_Toc41051300)

[Pav. 18 Trečio bandymo analizės grafikas naudojant tris klasterius 15](#_Toc41051301)

[Pav. 19 Trečio bandymo analizės grafikas naudojant keturis klasterius 16](#_Toc41051302)

[Lentelė. 1 Naudojamų duomenų paaiškinimas 4](#_Toc41051303)

[Lentelė. 2 Pasirinktos poros 5](#_Toc41051304)

[Lentelė. 3 Modelio įverčių lentelė 12](#_Toc41051305)

# Darbo užduoties aprašas, įrankių pasirinkimas

Pasirinkti duomenų rinkinį (apie 100-300 duomenų) kuriame būtų pateikiami bent 6 tolydiniai atributai (kategoriniai atributai netinka). Duomenų rinkinys gali neturėti išvesties atributo, nes tai neprižiūrimo mokymo algoritmas. Naudojantis pasirinktu duomenis juos suklasterizuoti į dviejų, trijų ir keturių klasterių dydžius atliekant po tris eksperimentus su skirtingomis poromis naudojant K-vidurkių algoritmą.

Užduoties sprendimui yra naudojamas K-vidurkių algoritmas, kuris yra realizuotas naudojant Python programavimo kalbą.

Naudojamos Python bibliotekos laboratoriniame darbe:

1. Numpy – darbui su masyvais ir papildomiems skaičiavimams.
2. Matplotlib – grafikų braižymui.
3. Pandas – HeatMap datos sutvarkymui.
4. Math – matematiniams skaičiavimams.
5. Seaborn – HeatMap braižymui.

Šiame laboratoriniame darbe pasirinkau NBA 2014-2015 metų pilną žaidėjų statistiką

# Įvestys bei išvestis

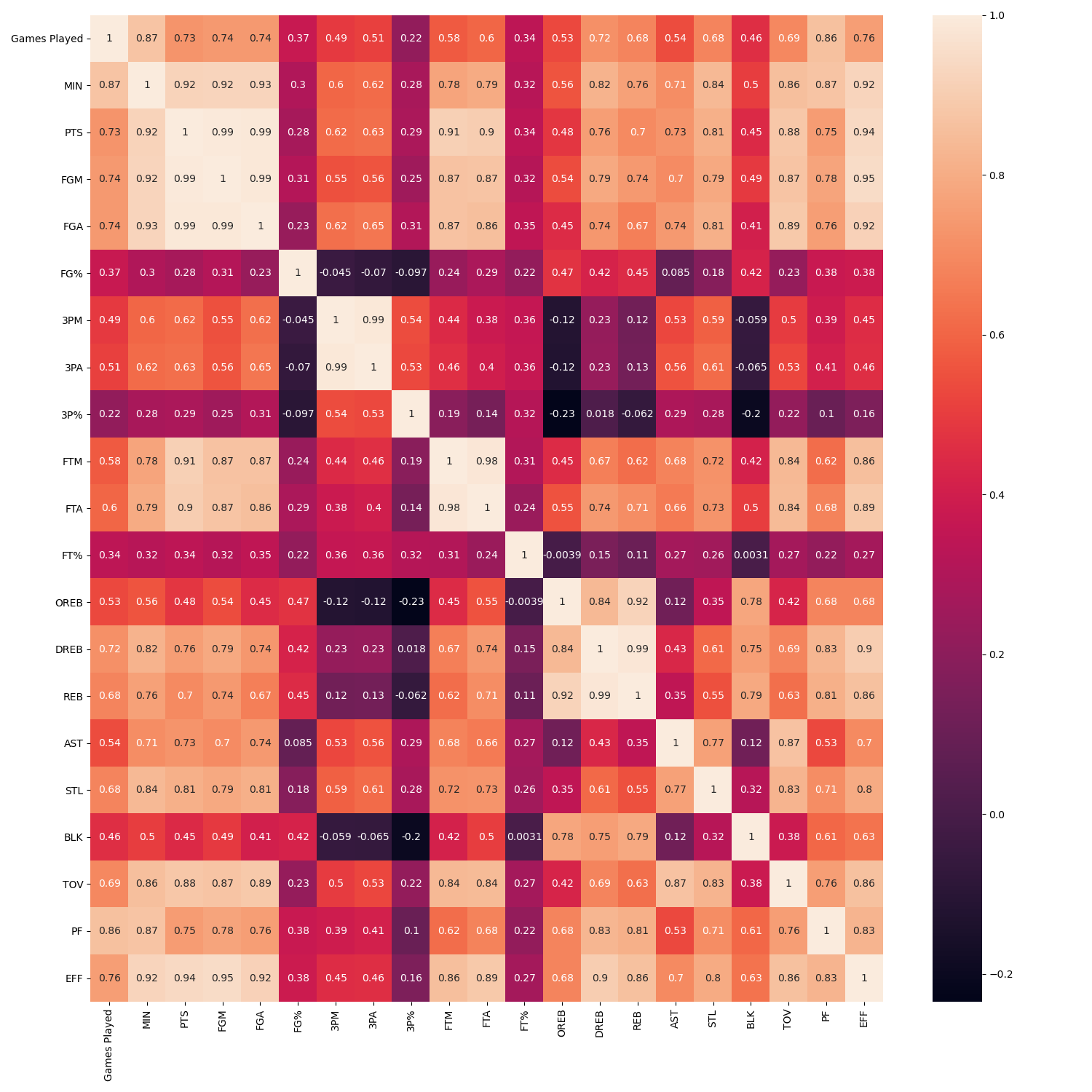
## Įvestys

NBA 2014-2015 metų pilną žaidėjų statistiką. Žemiau įdėta lentelė su pasirinktais duomenimis ir jų paaiškinimais.

|  |  |
| --- | --- |
| Žaista žaidimų (angl. Played Games) | Žaidėjo žaistų žaidimų kiekis |
| MIN | Žaistų minučių kiekis |
| PTS | Taškų kiekis |
| FGM | Pataikytu dvitaškių kiekis |
| FGA | Mestų dvitaškių kiekis |
| FG% | Dvitaškių pataikymo procentas |
| 3PM | Pataikytu tritaškių kiekis |
| 3PA | Mestų tritaškių kiekis |
| 3P% | Tritaškių pataikymo procentas |
| FTM | Pataikytų baudų kiekis |
| FTA | Mestų baudų kiekis |
| FT% | Mestų baudų procentas |
| OREB | Atkovoti kamuoliai po kitos komandos krepšiu |
| DREB | Atkovoti kamuoliai po savo komandos krepšiu |
| REB | Bendrai atkovoti kamuoliai |
| AST | Rezultatyvus kamuolio perdavimas kitam komandos žaidėjui |
| STL | Pavogtų kamuolių kiekis |
| BLK | Blokų kiekis |
| TOV | Prarastų kamuolių kiekis |
| PF | Pražangų kiekis |
| EFF | Efektyvumo reitingas |

Lentelė. 1 Naudojamų duomenų paaiškinimas

Susidarome koreliacijos matricą, pagal kurią sprendžiame kokius duomenis naudosime.



Pav. 1 Koreliacijos matrica

Atsižvelgiant į koreliacijos matricą sudarėme naudojamas poras. Poros buvo nustatytos pagal mažiausią koeficientą kadangi šitame laboratoriniame darbe duomenis reikia sugrupuoti į atskiras grupes.

|  |  |
| --- | --- |
| X | Y |
| 3P% | OREB |
| 3PM | BLK |
| 3PA | STL |

Lentelė. 2 Pasirinktos poros

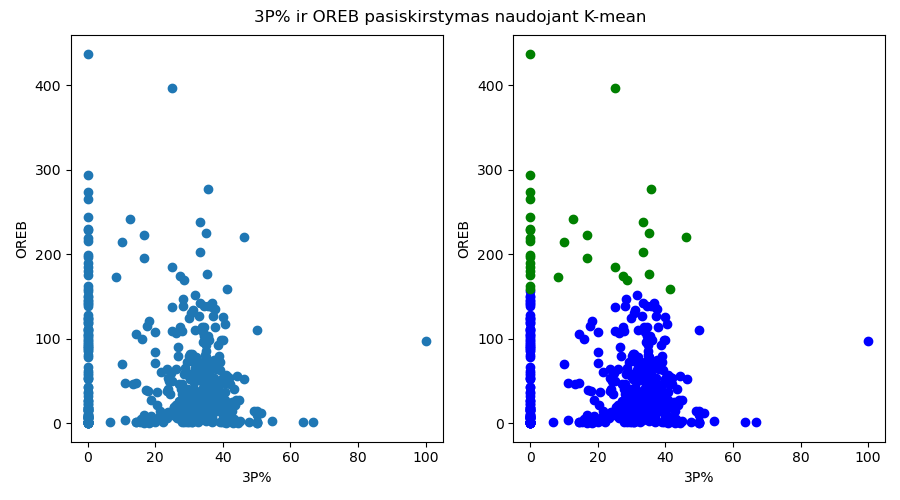
## Išvestys

Suskirstytos pasirinktos poros į dvi, tris ir keturias grupes naudojant K-mean algoritmą. Pirmiausia suklasterizuojame pasirinktus duomenis į du klasterius. Pradinius centroidu taškus pasirenku toliausius taškus tarpusavyje.

Detalesnius grafikus galima matyti Modelio analizėje.

### Klasterizavimas į du klasterius

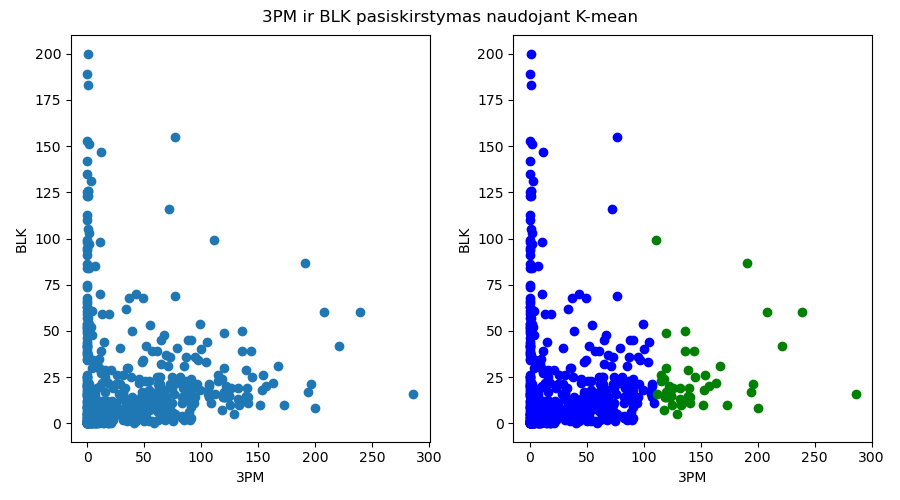
Klasterizavimui naudojama pora, tritaškių metimo procentai ir atkovoti kamuoliai po kitos komandos krepšiu. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.



Pav. 2 Duomenų klasterizavimas į du klasterius

Atlikę klasterizavimą galime pastebėti jog sugrupuota buvo pavienės reikšmės į vieną grupę, o taškai kurie yra vienas arti kito į kitą grupę

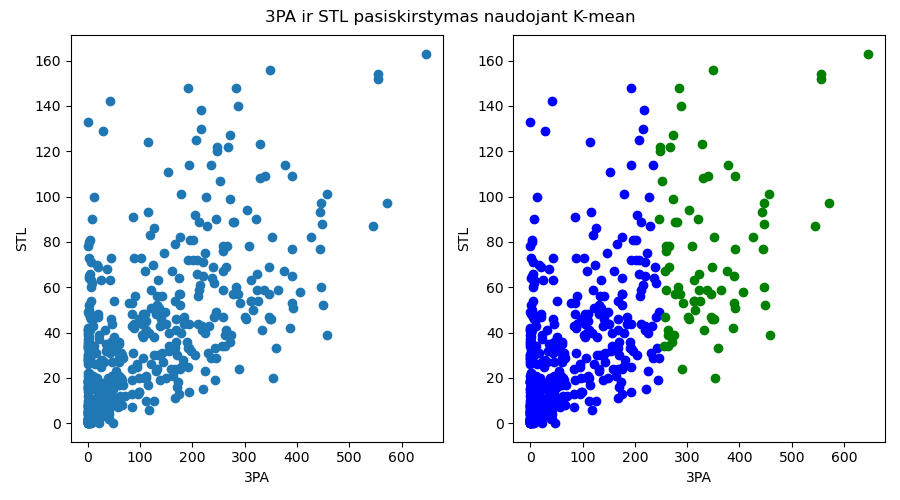
Klasterizavimui naudojama pora, įmestų tritaškių kiekis ir blokuoti metimai. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.



Pav. 3 Duomenų klasterizavimas į du klasterius

Atlikę klasterizavimą matome kad duomenų grupavimas įvyko toje vietoje kur yra mažiausia taškų. Būtent toje vietoje grafikas buvo perskeltas per pusę ir taip duomenys yra suskirstyti į dvi dalis.

Klasterizavimui naudojama pora yra mestų tritaškių kiekis ir pavogtų kamuolių kiekis. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.



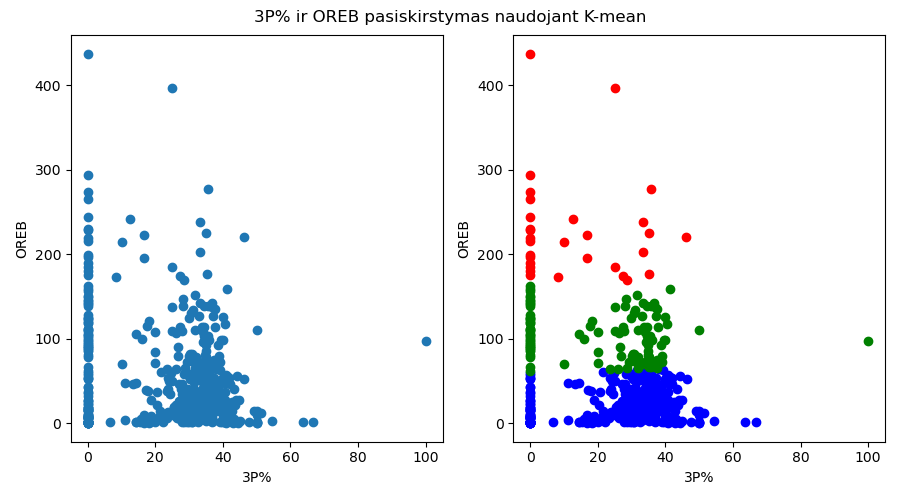
Pav. 4 Duomenų klasterizavimas į du klasterius

Panaudojus klasterizavimą matomas kitos rezultatas kadangi duomenys pasiskirstę vienodžiau ir per visą grafiką. Kadangi imamas dviejų klasterių ir toliausių taškų centroidai matome kad duomenų pasiskirstymas skyla į dvi dalis per grafiko vidurį.

### Klasterizavimas į tris klasterius

Klasterio centroidams naudojau tris tolygiai paskirstytus taškus, jie pasirenkami iš išrikiuotų taškų sąrašo.

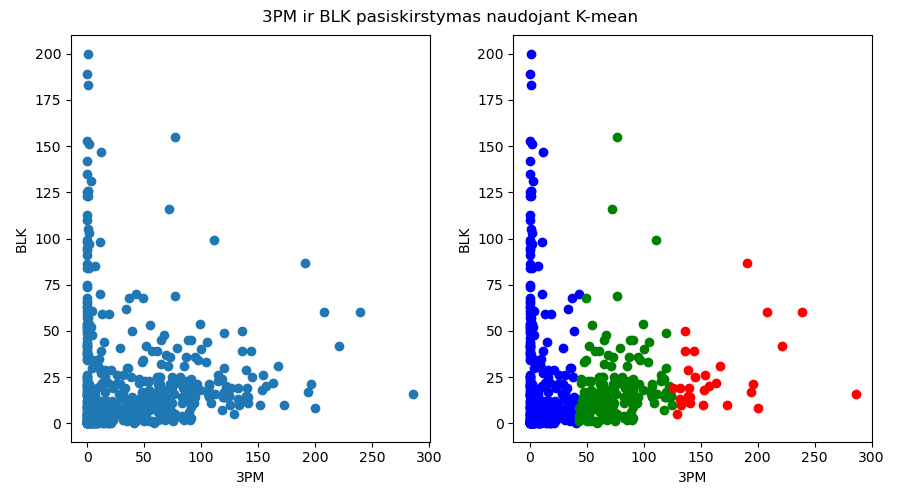
Pirmam bandymui naudojama pora, tritaškių metimo procentai ir atkovoti kamuoliai po kitos komandos krepšiu. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.



Pav. 5 Duomenų klasterizavimas į tris klasterius

Klasterizavus duomenis pastebime, kad tendencija nesikeičia, duomenys išskaidomi pagal centroid taškus kurie pasiskirstę daugmaž tolygiai. Duomenų kiekiai sudarytos grupėse skiriasi kadangi pasiskirstymas grafike nėra vienodas.

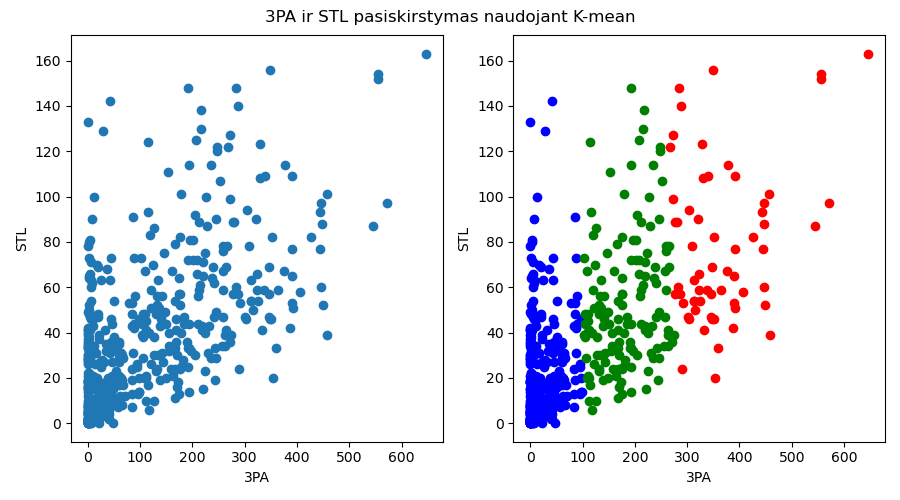
Antram bandymui naudojama pora, įmestų tritaškių kiekis ir blokuoti metimai. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.



Pav. 6 Duomenų klasterizavimas į tris klasterius

Suskirsčius duomenis į tris dalis matome jog duomenys pasiskirstę yra netolygiai, primoje grupėje yra daugiausia taškų, antroje grupėje vidutiniškai ir trečioje mažiausia. Toks pasiskirstymas susidarė kadangi atstumas tarp centroidų yra panašus. Taškai susiskirstę į netolygias grupes dėl to nes taškų pasiskirstymas grafike nėra tolygus.

Trečiam bandymui naudojama pora yra mestų tritaškių kiekis ir pavogtų kamuolių kiekis. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.

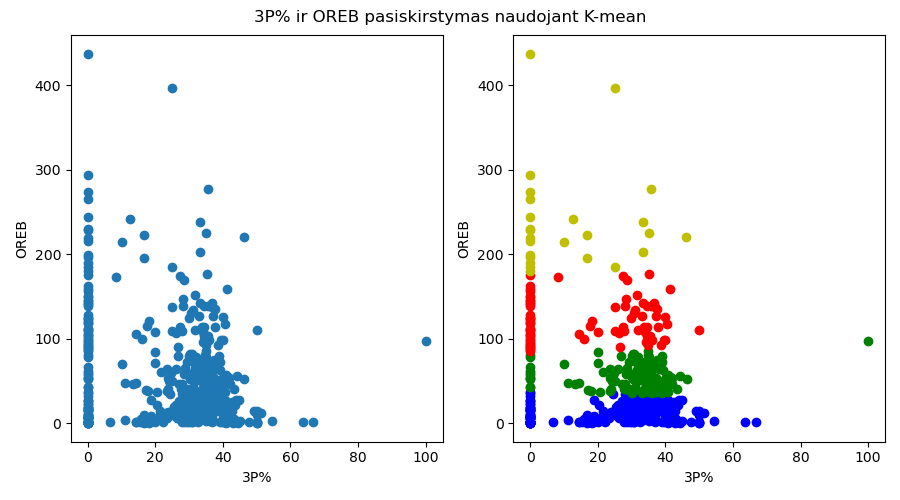


Pav. 7 Duomenų klasterizavimas į tris klasterius

Trečiame bandyme matome jog duomenys išskaidomi panašiai. Toks išskaidymas į grupes mums pasako jog taškų išsiskaidymas grafike yra panašus.

### Klasterizavimas į keturis klasterius

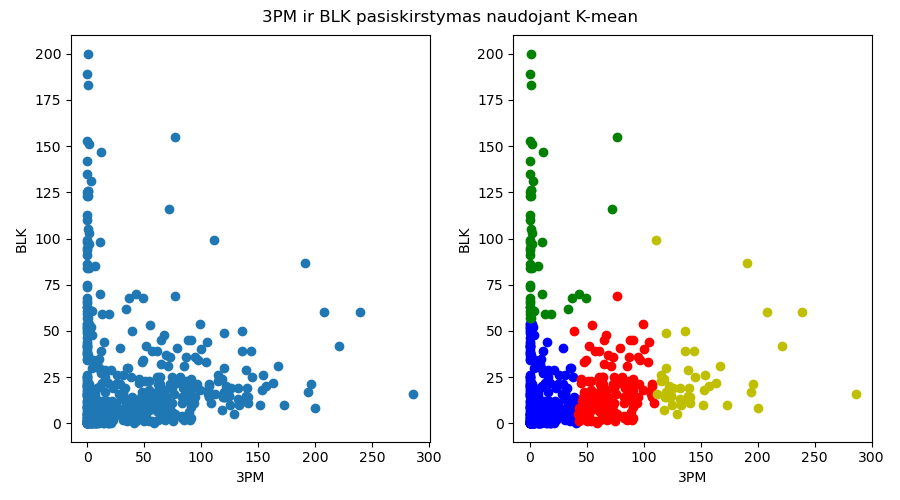
Pirmam bandymui naudojama pora yra tritaškių metimo procentai ir atkovoti kamuoliai po kitos komandos krepšiu. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.



Pav. 8 Duomenų klasterizavimas į keturis klasterius

Kaip ir viršutiniuose bandymuose matome, kad grafikas pasiskirsto į keturias dalis. Duomenų pasiskirstymas ne vienodas, kadangi taškų pasiskirstymas nėra vienodas.

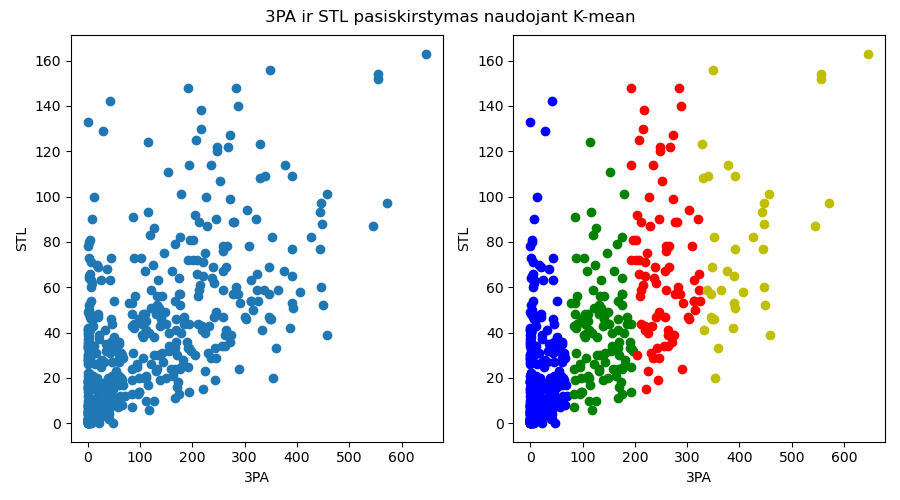
Antram bandymui naudojama pora yra įmestų tritaškių kiekis ir blokuoti metimai. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.



Pav. 9 Duomenų klasterizavimas į keturis klasterius

Atlikus antrą bandymą matome jog šitas bandymas pavyko geriausiai. Pasiskirstymas į grupes yra panašus. Grafike matosi jog rezultatai gavosi beveik kaip atskiros salelės.

Trečiam bandymui naudojama pora yra mestų tritaškių kiekis ir pavogtų kamuolių kiekis. Pirmieji duomenys naudojami kaip x ašies duomenys, antrieji duomenys naudojami kaip y ašies duomenys.



Pav. 10 Duomenų klasterizavimas į keturis klasterius

Paskutiniame bandyme taip pat grupių pasiskirstymas išryškėja daug geriau. Lyginant du grafikus galime pamatyti, kad taškai pasiskirsto į tokias saleles kurias galime pamatyti dar nepaskirstytame grafike.

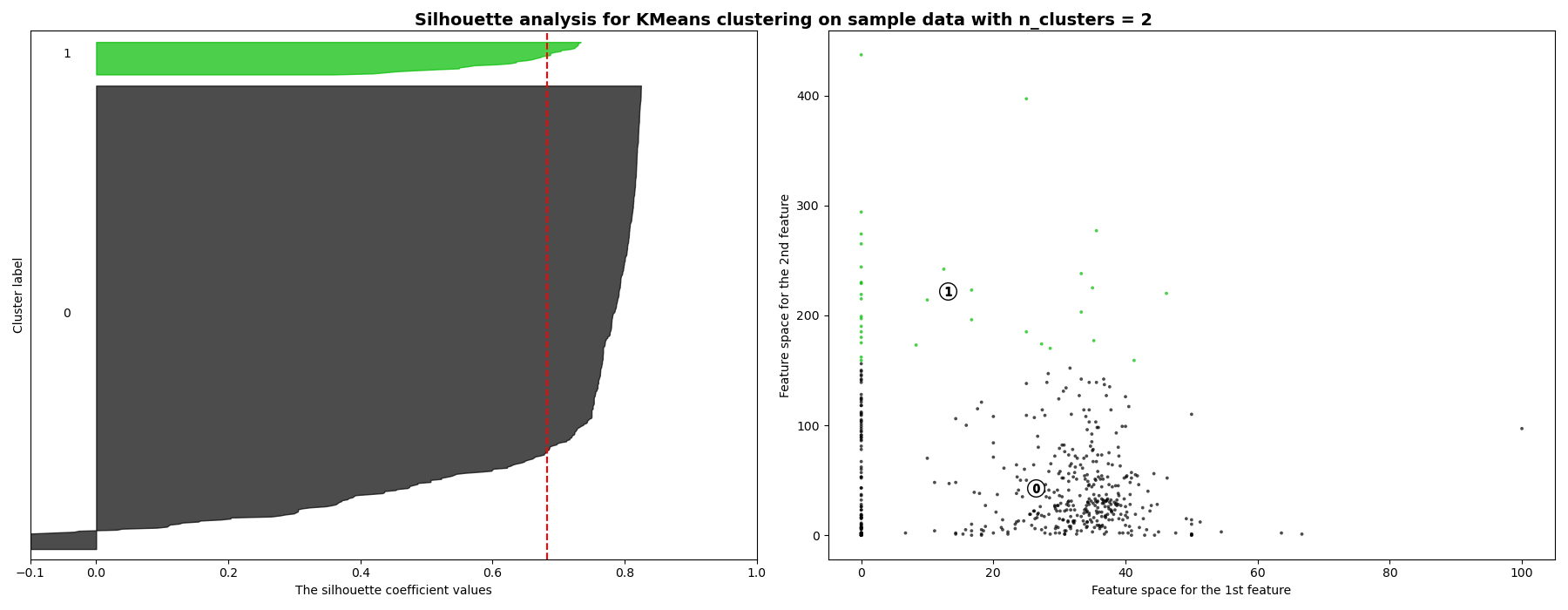
# Modelio Analizė ir klasterių pasirinkimai

Modelio analizei naudosime Silhouette analizę ir sklearn python biblioteką su mano gautais rezultatais. Modelio analizę atliksime norėdami optimaliausią klasterių skaičių su pasirinktais duomenimis.

Modelio analizę išsamiau aprašysiu pirmiems bandymams, kadangi galima spręsti pagal gautus silhouette taškus. Po pirmojo bandymo grafikų aptarsime lentelę su likusiai rezultatais.

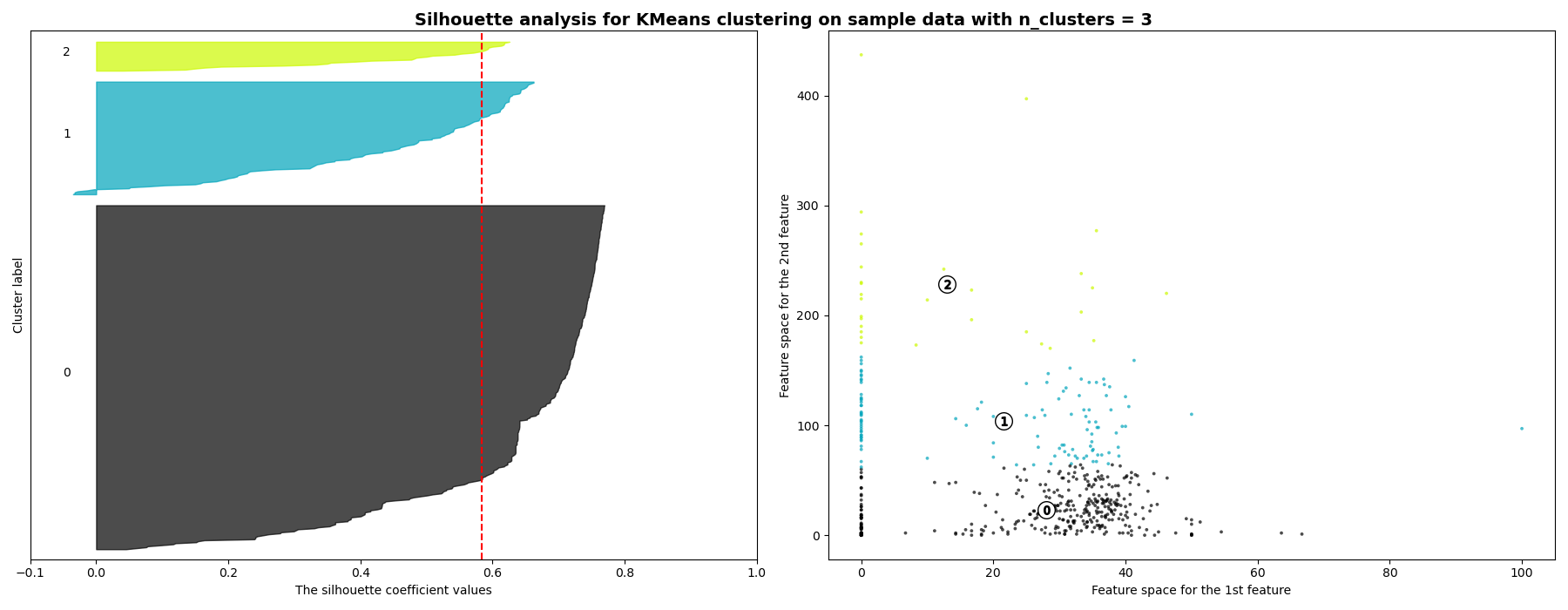
## Pirmas bandymas

Pirmasis grafikas aiškiai rodo jog pasiskirstymas nėra lygus, bet matome jo aiškiai atskirai grupes.



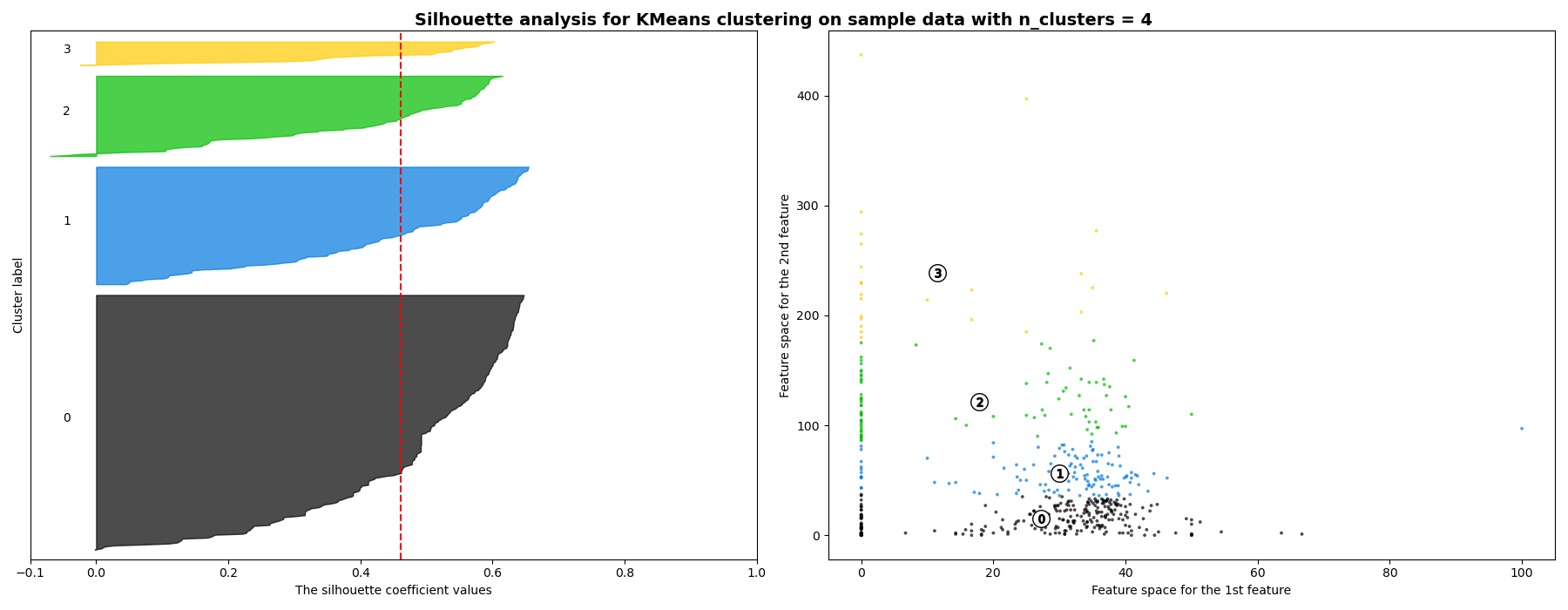
Pav. 11 Pirmo bandymo analizės grafikas naudojant du klasterius

Antrajame grafike matome vis dar aiškų trijų grupių pasiskirstymą. Nulinę grupę sudaro daugiausia taškų kaip ir pirmame grafike.



Pav. 12 Pirmo bandymo analizės grafikas naudojant tris klasterius

Trečiajame grafike pasiskirstymas vis dar matomas aiškiai. Taškų kiekis klasteriuose supanašėjo kaip ir koeficientu reikšmės.



Pav. 13 Pirmo bandymo analizės grafikas naudojant keturis klasterius

Pagal šituos grafikus galime atskirti kokį kiekį klasterių reikėtu naudoti pirmam bandymui. Neatsižvelgiant į vidutinį silhouette įvertį matome jog mažiausiai pasikeitė pirmieji du grafikai. Atsižvelgiant į grafikus ir lentelę esančią apačioje galime spręsti jo geriausia pirmam bandymui yra naudoti du klasterius kadangi aiškiausiai atskiriamos grupės.

Pateiktoje lentelėje matome modelio įverčių vidurkius pagal kuriuos taip pat galima spręsti klasterių kiekį.

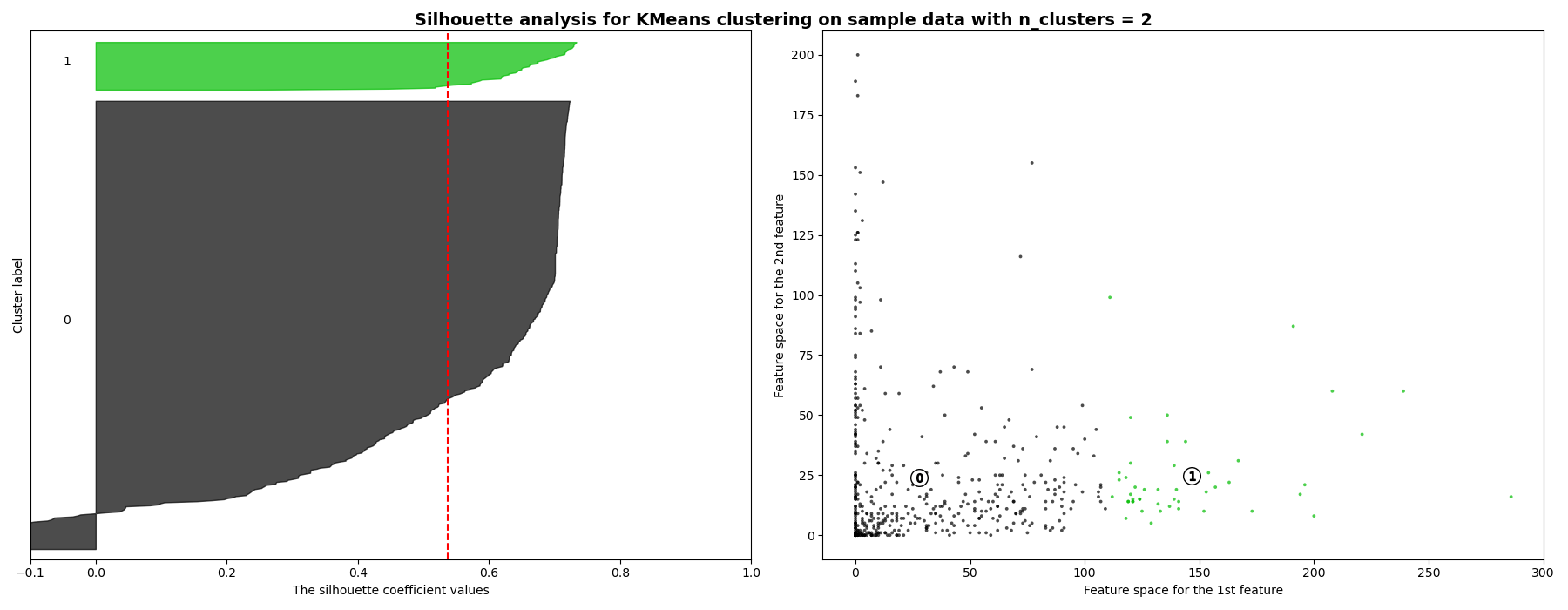
|  |  |  |  |
| --- | --- | --- | --- |
| Klasterių kiekis | Pirmo bandymo modelio vidutinis įverčiai | Antro bandymo modelio vidutinis įverčiai | Trečio bandymo modelio vidutinis įverčiai |
| 2 | 0.6825419882718818 | 0.5375989520508717 | 0.6062526551916785 |
| 3 | 0.5838175234374163 | 0.46715669986218095 | 0.602965962885443 |
| 4 | 0.4610331785394148 | 0.5356632274898092 | 0.5711744976147289 |

Lentelė. 3 Modelio įverčių lentelė

Lentelėje labai geria matosi kuriuos klasterius naudoti pasirinktuose bandymuose. Visoje lentelėje dominuoja antras klasteris. Trečio ir ketvirto klasterių įverčiai kinta, vienu atveju didesnis įvertis trečio, kitu atveju ketvirto.

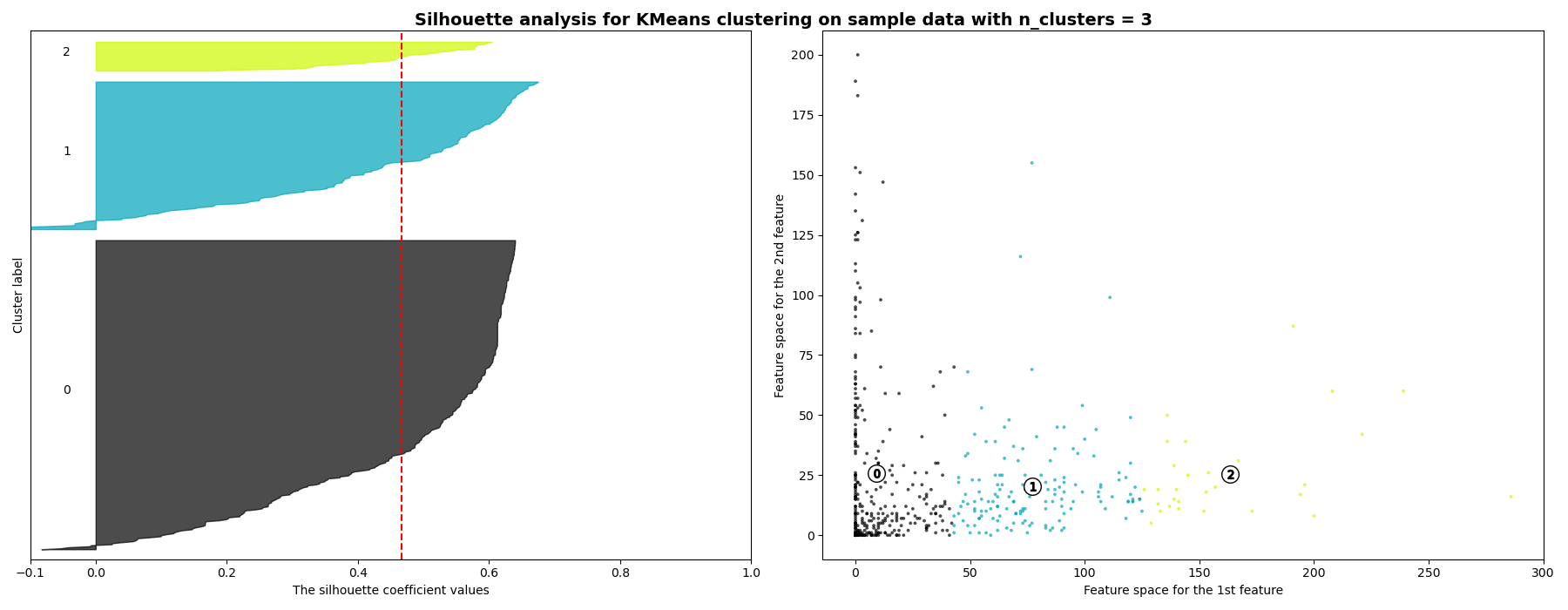
## Antras bandymas

Grafikas aiškiai rodo jog pasiskirstymas nėra lygus, bet matome jo aiškiai atskirtos grupes viena nuo kitos. Atsižvelgiant į grafiką ir į įverčius antras klasteris yra geriausias antro bandymo naudojimui.



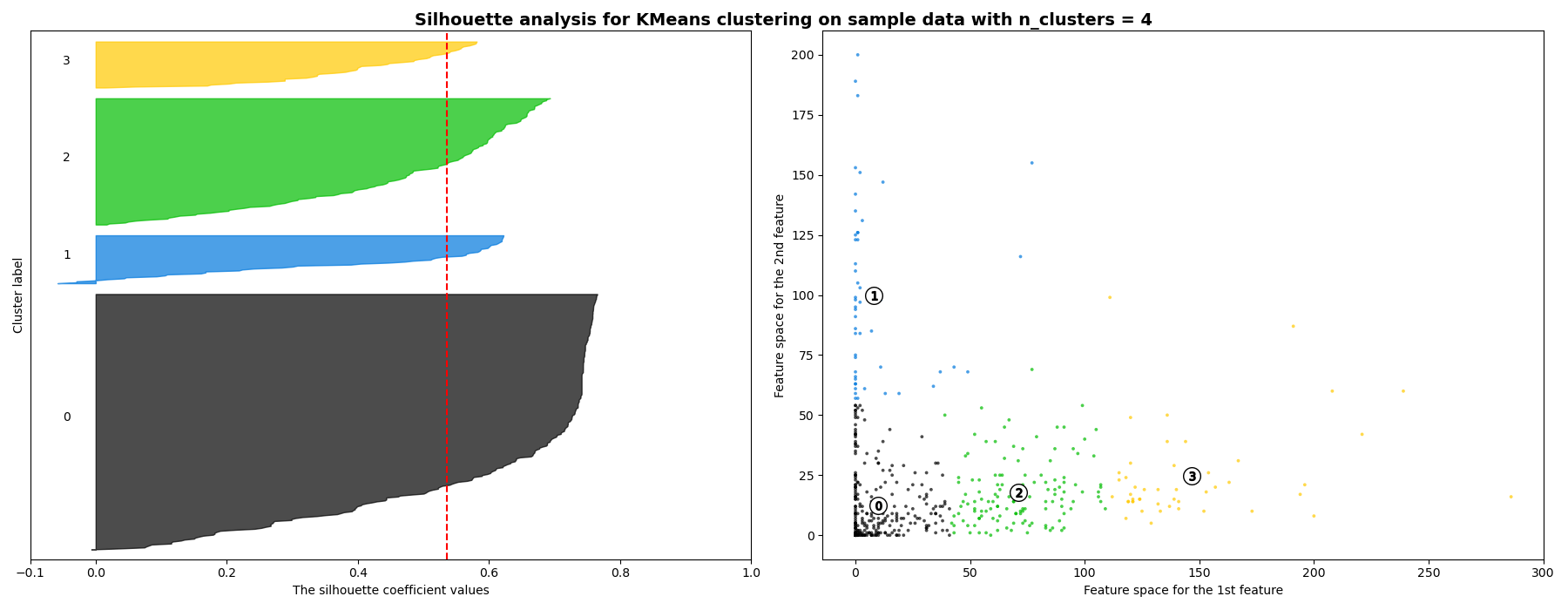
Pav. 14 Antro bandymo analizės grafikas naudojant du klasterius

Antrame grafike matome nemažus pasikeitimus. Paskutinioji grupė dar labiau sumažėjo. Grafike matomas grupių atskyrimas yra aiškus, Atsiradęs dydžių nepastovumas mažina įvertį esantį lentelėje.



Pav. 15 Antro bandymo analizės grafikas naudojant tris klasterius

Trečiame grafike matome jog grupių dydžiai labiau pasiskirsto tarpusavyje. Geresnis pasiskirstymas grupėse gerina įvertį esantį lentelėje.

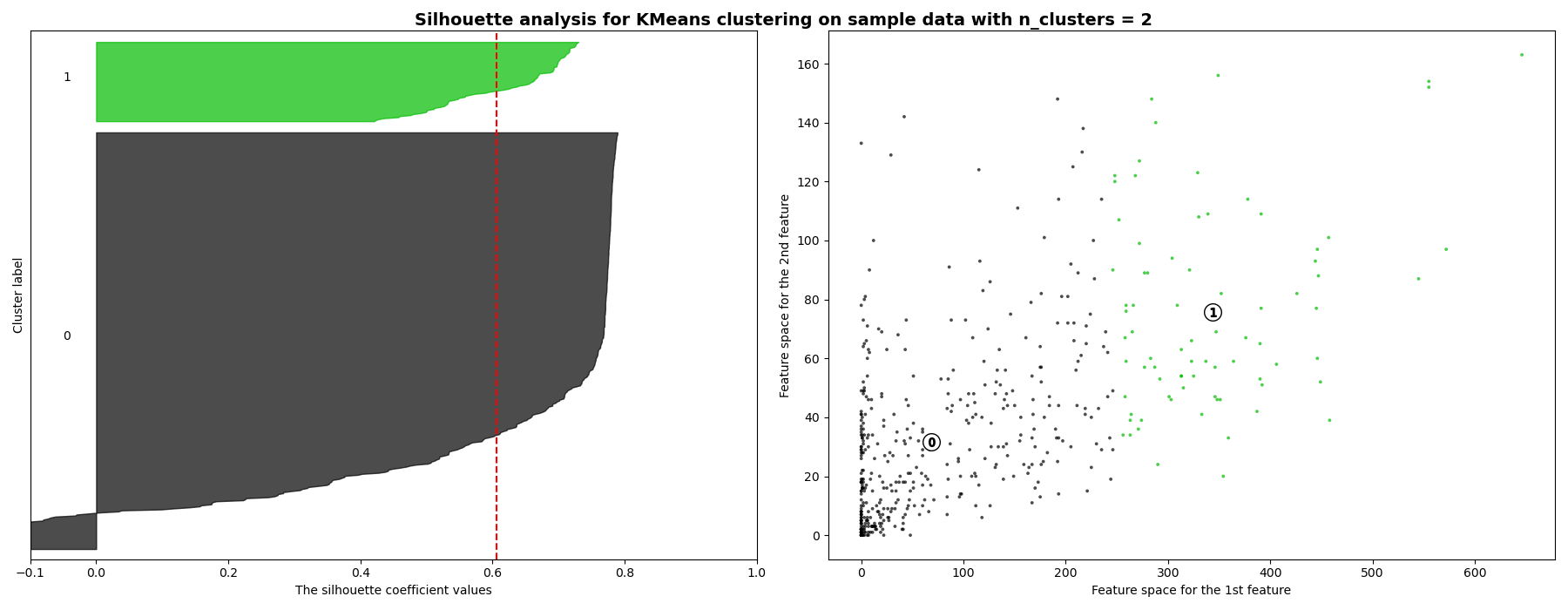


Pav. 16 Antro bandymo analizės grafikas naudojant keturis klasterius

Pagal duotus grafikus kurie yra aukščiau galime atskirti kokį kiekį klasterių reikėtu naudoti antram bandymui. Neatsižvelgiant į vidutinį silhouette įvertį matome jog visi grafikai skyrėsi daug, bet pirmas ir trečias grafikai yra geriausi. Pirmas grafiko grupės paskirstytos į aiškias dvi saleles vieną didelę ir vieną maža. Trečio grafiko salelės taip pat aiškiai atskirtos. Trečio grafiko salelių dydžiai yra panašūs. Dėl paminėtų priežasčių ir duotos lentelė aukščiau, geriausi pasirinkimai yra naudoti du arba keturis klasterius.

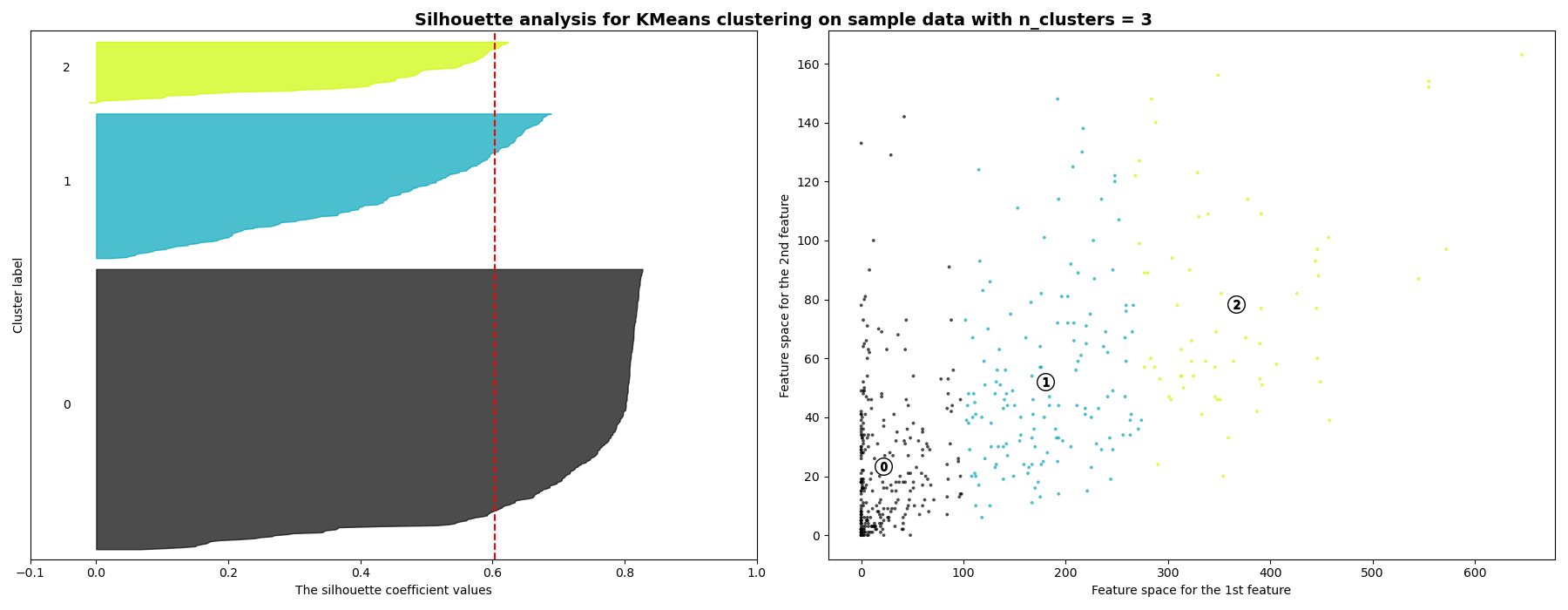
## Trečias bandymas

Grafikas aiškiai rodo jog pasiskirstymas nėra lygus, bet matome jo aiškiai atskirai grupes viena nuo kitos.



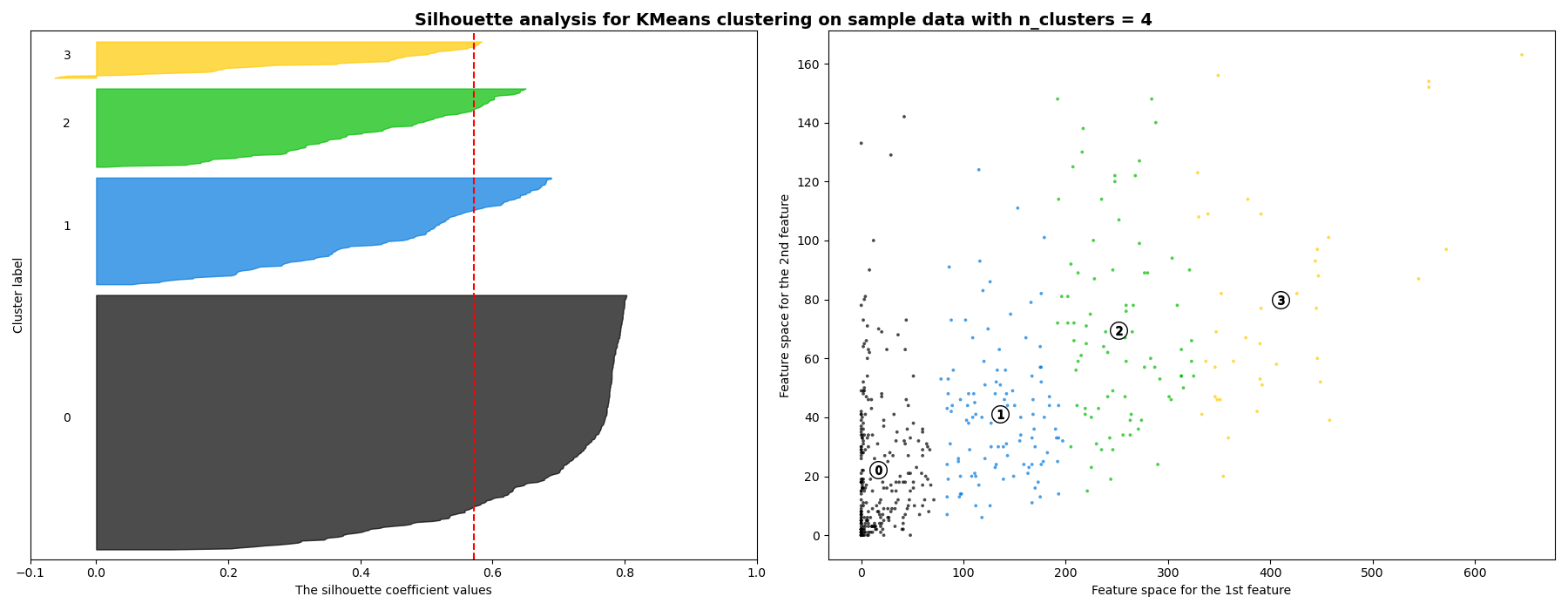
Pav. 17 Trečio bandymo analizės grafikas naudojant du klasterius

Antrame grafike matome, kad daugiausia taškų išlieka pirmoje grupėje, todėl stabilumas ir įvertis keičiasi labai mažai.



Pav. 18 Trečio bandymo analizės grafikas naudojant tris klasterius

Trečiame grafike taip pat matome kad daugiausia taškų išlieka pirmoje grupėje, todėl stabilumas ir įvertis keičiasi mažai kaip ir naudojant tris klasterius.



Pav. 19 Trečio bandymo analizės grafikas naudojant keturis klasterius

Pagal šituos grafikus galime atskirti kokį kiekį klasterių reikėtu naudoti trečiam bandymui. Neatsižvelgiant į vidutinį silhouette įvertį matome jog stabiliausi grafikai yra pirmas ir antras. Kaip ir pirmame bandyme pirmasis grafikas pasiskirstęs į didelę ir mažą grupes. Antras grafikas yra pasiskirstęs į dvi panašias grupes ir trečią didelę. Pasirinktas klasterių kiekis atsižvelgiant į grafikus butų du arba trys klasteriai. Renkantis klasterių kiekį atsižvelgus į grafikus ir aukščiau duotą lentelę klasterių pasirinkimas nesikeičia.

## Dalinė išvada

Išanalizavę visus grafikus matome jog klasterių pasirinkimą įtakoja daug veiksnių. Grafiko pasiskirstymas yra vienas iš didžiausių veiksnių, jei grafikas pasiskirstęs į daug mažų salelių tai greičiausiai geriau naudoti bus didesnį klasterių kiekį. Pirmųjų centroidų pasirinkimas taip pat įtakoja klasterių kiekį. Blogai pasirinkus centroidu taškus galima gauti blogą klasterių paskirstymą kuris įtakoja klasterių kiekį. Mano atveju centroidai buvo pasirinkti tokie, kad jų atstumas būtu panašus.

# Išvados

Išsiaiškinau kaip naudojamas ir kam naudojamas K-mean algoritmas. Praktiškai pavyko realizuoti K-mean algoritmą naudojantis NBA 2014-2015m. duomenis. Duomenim suklasterizuoti naudojau du, tris ir keturis klasterius. Naudojant skirtingų dydžio klasterius pastebėjau jog duomenys klasteriuose pasiskirsto nevienodai. Taškų pasiskirstymą klasteriuose lemia klasterių skaičius, pačių taškų pasiskirstymas grafike ir pasirinkti centroidai. Geriausias klasterių kiekis gali būti nustatytas naudojantis analizės metodais kaip “Silhouette analysis”. Analizė duoda įverčius ir grafikus pagal kuriuos galime pasirinkti optimalų klasterių kiekį. Mano pasirinkti duomenys nebuvo geriausi, kadangi taškų pasiskirstymas grafike buvo kaip vieną didelė grupė. Klasterizavimo algoritmas rado skirtingas salas, bet lyginant grafikus nebuvo aiškiai matomų salelių. Panaudojus algoritmą ir atlikus analizę supratau jog šitas algoritmas skirtas atskiroms grupėms sudaryti ir sudarytų grupių panaudojimui. Pasirinkus geresnius NBA duomenis galėjau žaidėjus suskaidyti į žaidėjų įgūdžių grupes.

# Programos kodas

## Kodas naudojamas išvestims

**import** csv  
**from** pandas **import** DataFrame  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plot  
**import** seaborn **as** sb  
**import** math  
  
  
**def** Heat(nameList, Data):  
 fig, ax = plot.subplots(figsize=(15, 15))  
 df = DataFrame(Data, columns=nameList)  
 corrMatrix = df.corr()  
 ax = sb.heatmap(corrMatrix, annot=**True**)  
 plot.show()  
  
**class** K\_Means:  
  
 **def** \_\_init\_\_(self, k=2, tol=0.001, max\_iter=300):  
 self.k = k  
 self.tol = tol  
 self.max\_iter = max\_iter  
  
 **def** fit(self, data, centroids):  
  
 self.centroids = {}  
  
 **for** i **in** range(self.k):  
 self.centroids[i] = centroids[i]  
  
 **for** i **in** range(self.max\_iter):  
 self.classifications = {}  
 self.labels\_ = []  
 **for** i **in** range(self.k):  
 self.classifications[i] = []  
  
 **for** featureset **in** data:  
 distances = [np.linalg.norm(featureset - self.centroids[centroid]) **for** centroid **in** self.centroids] *# norm vektoriaus esme kaip normalus foras* classification = distances.index(min(distances)) *# gaunamas indeksas* self.labels\_.append(classification) *# pridedamas klasterio indeksas* self.classifications[classification].append(featureset) *# nustatoma kuris klusteris* prev\_centroids = dict(self.centroids)  
  
 **for** classification **in** self.classifications:  
 self.centroids[classification] = np.average(self.classifications[classification], axis=0) *# perskaiciuojami centroidai* optimized = **True  
  
 for** c **in** self.centroids: *# erroru skaiciavimas ir patikrinamas ar testi skaiciavimus* original\_centroid = prev\_centroids[c]  
 current\_centroid = self.centroids[c]  
 **if** np.sum(original\_centroid) > 0:  
 **if** (np.sum((current\_centroid - original\_centroid)) / np.sum(  
 original\_centroid) \* 100.0) > self.tol:  
 optimized = **False  
 else**:  
 optimized = **False  
  
  
 if** optimized:  
 **break  
  
def** sortBy(e): *# saraso rikiavimo funkcija* **return** math.pow((e[0] + e[1]),2)  
  
  
  
dataList = [] *# duomenu is failo nuskaitymas***with** open(**'players\_stats.csv'**) **as** csvfile:  
 reader = csv.reader(csvfile)  
 **for** row **in** reader:  
 dataList.append(row)  
  
dataUsingKey = {}  
keys = [] *# duomenu pavadinimai naudojami kaip raktai pasiekti duomenims*cnt = 0  
  
*# raktu panaudojimo paruosimas***for** x **in** dataList[0]:  
 **if** cnt > 0 **and** cnt < 22:  
 dataUsingKey[x] = []  
 keys.append(x)  
 cnt = cnt + 1  
  
index = 0  
**for** x **in** dataList[1:]: *# duomenu pridejimas i dictionary* index = 0  
 **for** a **in** x[1:22]:  
 dataUsingKey[keys[index]].append(float(a))  
 index = index + 1  
  
Heat(keys, dataUsingKey) *# heat map peisimas*range\_n\_clusters = [2, 3, 4, 5, 6] *# klasteriu kiekiu sarasas*pairs = [[8,12], [6,17], [7,16]] *# naudojamu prou indeksai***class** K\_Means:  
  
 **def** \_\_init\_\_(self, k=2, tol=0.001, max\_iter=300):  
 self.k = k  
 self.tol = tol  
 self.max\_iter = max\_iter  
  
 **def** fit(self, data, centroids):  
  
 self.centroids = {}  
  
 **for** i **in** range(self.k):  
 self.centroids[i] = centroids[i]  
  
 **for** i **in** range(self.max\_iter):  
 self.classifications = {}  
 self.labels\_ = []  
 **for** i **in** range(self.k):  
 self.classifications[i] = []  
  
 **for** featureset **in** data:  
 distances = [np.linalg.norm(featureset - self.centroids[centroid]) **for** centroid **in** self.centroids] *# skaičiuojamas atstumas tap taškų* classification = distances.index(min(distances)) *# gaunamas indeksas* self.labels\_.append(classification) *# pridedamas klasterio indeksas* self.classifications[classification].append(featureset) *# kordinačių įdėjimas į klasterį* prev\_centroids = dict(self.centroids)  
  
 **for** classification **in** self.classifications:  
 self.centroids[classification] = np.average(self.classifications[classification], axis=0) *# perskaiciuojami centroidai* optimized = **True  
  
 for** c **in** self.centroids: *# erorų skaičiavimas ir patikrinamas ar tęsti skaičiavimus* original\_centroid = prev\_centroids[c]  
 current\_centroid = self.centroids[c]  
 **if** np.sum(original\_centroid) > 0:  
 **if** (np.sum((current\_centroid - original\_centroid)) / np.sum(  
 original\_centroid) \* 100.0) > self.tol:  
 optimized = **False  
 else**:  
 optimized = **False  
  
  
 if** optimized:  
 **break  
  
def** sortBy(e): *# sarašo rikiavimo funkcija* **return** math.pow((e[0] + e[1]),2)  
  
  
  
dataList = [] *# duomenų is failo nuskaitymas***with** open(**'players\_stats.csv'**) **as** csvfile:  
 reader = csv.reader(csvfile)  
 **for** row **in** reader:  
 dataList.append(row)  
  
dataUsingKey = {}  
keys = [] *# duomenų pavadinimai naudojami kaip raktai pasiekti duomenims*cnt = 0  
  
*# raktų panaudojimo paruošimas***for** x **in** dataList[0]:  
 **if** cnt > 0 **and** cnt < 22:  
 dataUsingKey[x] = []  
 keys.append(x)  
 cnt = cnt + 1  
  
index = 0  
**for** x **in** dataList[1:]: *# duomenų pridėjimas į dictionary* index = 0  
 **for** a **in** x[1:22]:  
 dataUsingKey[keys[index]].append(float(a))  
 index = index + 1  
  
range\_n\_clusters = [2, 3, 4, 5, 6] *# klasterių kiekių sarašas*pairs = [[8,12], [6,17], [7,16]] *# naudojamu proų indeksai***for** i **in** range(len(pairs)):  
  
 x\_index = pairs[i][0]  
 y\_index = pairs[i][1]  
  
 dataSorting = []  
 coordinates = []  
 data = []  
  
 *# sarašo paruošimas centroidų paieškai* **for** a **in** range(len(dataUsingKey[keys[y\_index]])):  
 dataSorting.append([dataUsingKey[keys[x\_index]][a], dataUsingKey[keys[y\_index]][a]])  
 data.append([dataUsingKey[keys[x\_index]][a], dataUsingKey[keys[y\_index]][a]])  
  
 dataSorting.sort(key=sortBy)  
  
 **for** k\_count **in** range\_n\_clusters:  
  
 *# centroidu paieška atsižvelgiant į klasterių kiekį* init\_centroids = [[dataSorting[0][0], dataSorting[0][1]]]  
 **if** k\_count > 2:  
 splitTo = int(len(dataSorting) / (k\_count - 1))  
 x = splitTo  
 **while** x < splitTo \* (k\_count - 1):  
 init\_centroids.append([dataSorting[x][0], dataSorting[x][1]])  
 x = x + splitTo  
  
 init\_centroids.append([dataSorting[len(dataSorting) - 1][0], dataSorting[len(dataSorting) - 1][1]])  
  
 *# sukurto modelio panaudojimas* kmeans = K\_Means(k\_count)  
 kmeans.fit(np.array(data), np.array(init\_centroids))  
 lables = kmeans.labels\_  
  
 *# x ir y reikšmių formatavimas grafiko naudojimui* dataXAfter = []  
 dataYAfter = []  
 **for** a **in** kmeans.classifications:  
 dataXAfter.append([])  
 dataYAfter.append([])  
 **for** classification **in** kmeans.classifications[a]:  
 dataXAfter[a].append(classification[0])  
 dataYAfter[a].append(classification[1])  
  
 *# dviejų grafikų piešimas* fig, (ax0, ax1) = plot.subplots(nrows=1, ncols=2, figsize=(9, 5,))  
 *# grafikas skirtas neskirtytiems duomenims atvaizduoti* ax0.scatter(dataUsingKey[keys[x\_index]], dataUsingKey[keys[y\_index]])  
 *# grafikai skirti atvaizduoti sugrupuotus duomenis* **if** k\_count > 1:  
 ax1.scatter(dataXAfter[0], dataYAfter[0], c=**'b'**)  
 **if** k\_count > 1:  
 ax1.scatter(dataXAfter[1], dataYAfter[1], c=**'g'**)  
 **if** k\_count > 2:  
 ax1.scatter(dataXAfter[2], dataYAfter[2], c=**'r'**)  
 **if** k\_count > 3:  
 ax1.scatter(dataXAfter[3], dataYAfter[3], c=**'y'**)  
 **if** k\_count > 4:  
 ax1.scatter(dataXAfter[4], dataYAfter[4], c=**'m'**)  
 **if** k\_count > 5:  
 ax1.scatter(dataXAfter[5], dataYAfter[5], c=**'c'**)  
 ax1.set\_xlabel(keys[x\_index])  
 ax1.set\_ylabel(keys[y\_index])  
 ax0.set\_xlabel(keys[x\_index])  
 ax0.set\_ylabel(keys[y\_index])  
 ax0.set\_title(**" "**)  
 ax1.set\_title(**" "**)  
 fig.suptitle( keys[x\_index] +**" ir "**+ keys[y\_index] +**" pasiskirstymas naudojant K-mean"**)  
  
plot.tight\_layout()  
plot.show()

## Kodas naudojamas modelio validacijai

**from** sklearn.metrics **import** silhouette\_samples, silhouette\_score  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.cm **as** cm  
**import** numpy **as** np  
**import** csv  
**import** math  
  
**class** K\_Means:  
  
 **def** \_\_init\_\_(self, k=2, tol=0.001, max\_iter=300):  
 self.k = k  
 self.tol = tol  
 self.max\_iter = max\_iter  
  
 **def** fit(self, data, centroids):  
  
 self.centroids = {}  
  
 **for** i **in** range(self.k):  
 self.centroids[i] = centroids[i]  
  
 **for** i **in** range(self.max\_iter):  
 self.classifications = {}  
 self.labels\_ = []  
 **for** i **in** range(self.k):  
 self.classifications[i] = []  
  
 **for** featureset **in** data:  
 distances = [np.linalg.norm(featureset - self.centroids[centroid]) **for** centroid **in** self.centroids] *# skaičiuojamas atstumas tap taškų* classification = distances.index(min(distances)) *# gaunamas indeksas* self.labels\_.append(classification) *# pridedamas klasterio indeksas* self.classifications[classification].append(featureset) *# kordinačių įdėjimas į klasterį* prev\_centroids = dict(self.centroids)  
  
 **for** classification **in** self.classifications:  
 self.centroids[classification] = np.average(self.classifications[classification], axis=0) *# perskaiciuojami centroidai* optimized = **True  
  
 for** c **in** self.centroids: *# erorų skaičiavimas ir patikrinamas ar tęsti skaičiavimus* original\_centroid = prev\_centroids[c]  
 current\_centroid = self.centroids[c]  
 **if** np.sum(original\_centroid) > 0:  
 **if** (np.sum((current\_centroid - original\_centroid)) / np.sum(  
 original\_centroid) \* 100.0) > self.tol:  
 optimized = **False  
 else**:  
 optimized = **False  
  
  
 if** optimized:  
 **break  
  
def** sortBy(e): *# sarašo rikiavimo funkcija* **return** math.pow((e[0] + e[1]),2)  
  
  
  
dataList = [] *# duomenų is failo nuskaitymas***with** open(**'players\_stats.csv'**) **as** csvfile:  
 reader = csv.reader(csvfile)  
 **for** row **in** reader:  
 dataList.append(row)  
  
dataUsingKey = {}  
keys = [] *# duomenų pavadinimai naudojami kaip raktai pasiekti duomenims*cnt = 0  
  
*# raktų panaudojimo paruošimas***for** x **in** dataList[0]:  
 **if** cnt > 0 **and** cnt < 22:  
 dataUsingKey[x] = []  
 keys.append(x)  
 cnt = cnt + 1  
  
index = 0  
**for** x **in** dataList[1:]: *# duomenų pridėjimas į dictionary* index = 0  
 **for** a **in** x[1:22]:  
 dataUsingKey[keys[index]].append(float(a))  
 index = index + 1  
  
range\_n\_clusters = [2, 3, 4, 5, 6] *# klasterių kiekių sarašas*pairs = [[8,12], [6,17], [7,16]] *# naudojamu proų indeksai***for** i **in** range(len(pairs)):  
  
 print(**"bandymas nr. "** + str(i) + **" -----------------------"**)  
 x\_index = pairs[i][0]  
 y\_index = pairs[i][1]  
  
 dataSorting = []  
 coordinates = []  
 data = []  
  
 *# sarašo paruošimas centroidų paieškai* **for** a **in** range(len(dataUsingKey[keys[y\_index]])):  
 dataSorting.append([dataUsingKey[keys[x\_index]][a], dataUsingKey[keys[y\_index]][a]])  
 data.append([dataUsingKey[keys[x\_index]][a], dataUsingKey[keys[y\_index]][a]])  
 data = np.array(data)  
 dataSorting.sort(key=sortBy)  
  
 **for** n\_clusters **in** range\_n\_clusters:  
  
 *# centroidu paieška atsižvelgiant į klasterių kiekį* init\_centroids = [[dataSorting[0][0], dataSorting[0][1]]]  
 **if** n\_clusters > 2:  
 splitTo = int(len(dataSorting) / (n\_clusters - 1))  
 x = splitTo  
 **while** x < splitTo \* (n\_clusters - 1):  
 init\_centroids.append([dataSorting[x][0], dataSorting[x][1]])  
 x = x + splitTo  
  
 init\_centroids.append([dataSorting[len(dataSorting) - 1][0], dataSorting[len(dataSorting) - 1][1]])  
 init\_centroids = np.array(init\_centroids)  
  
 *# sukuriamas grafikas su dviejais sub grafikais* fig, (ax1, ax2) = plt.subplots(1, 2)  
 fig.set\_size\_inches(18, 7)  
  
 ax1.set\_xlim([-0.1, 1])  
  
 ax1.set\_ylim([0, len(data) + (n\_clusters + 1) \* 10])  
  
 *# sukurto modelio panaudojimas* clusterer = K\_Means(n\_clusters)  
 clusterer.fit(data, init\_centroids)  
 cluster\_labels = np.array(clusterer.labels\_)  
  
 *# centroidu paruošimas naudojimu silhouette analizei* centers = []  
 **for** a **in** clusterer.centroids:  
 centers.append([clusterer.centroids[a][0], clusterer.centroids[a][1]])  
 centers = np.array(centers)  
  
 *# silhouette analizes panaudojimas įverčiui gauti* silhouette\_avg = silhouette\_score(data, cluster\_labels)  
 print(**"For n\_clusters ="**, n\_clusters,  
 **"The average silhouette\_score is :"**, silhouette\_avg)  
  
 *# paskaičiuoti įvertį kiekvienam taškui* sample\_silhouette\_values = silhouette\_samples(data, cluster\_labels)  
  
 y\_lower = 10  
 **for** i **in** range(n\_clusters):  
 *# agreguoti įvererčius naudojanmus pasirinktam klasteriui* ith\_cluster\_silhouette\_values = \  
 sample\_silhouette\_values[cluster\_labels == i]  
  
 ith\_cluster\_silhouette\_values.sort()  
  
 size\_cluster\_i = ith\_cluster\_silhouette\_values.shape[0]  
 y\_upper = y\_lower + size\_cluster\_i  
  
 color = cm.nipy\_spectral(float(i) / n\_clusters)  
 ax1.fill\_betweenx(np.arange(y\_lower, y\_upper), 0, ith\_cluster\_silhouette\_values, facecolor=color, edgecolor=color, alpha=0.7)  
  
 *# legendos teksto sudarymas* ax1.text(-0.05, y\_lower + 0.5 \* size\_cluster\_i, str(i))  
  
 *# paskaičiauoti nauja y apatini kuris naudojamas legendos tekstui* y\_lower = y\_upper + 10  
  
 ax1.set\_title(**" "**)  
 ax1.set\_xlabel(**"The silhouette coefficient values"**)  
 ax1.set\_ylabel(**"Cluster label"**)  
  
 *# Raudona vertikali linija parodyti vidutiniam analizės įverčiui* ax1.axvline(x=silhouette\_avg, color=**"red"**, linestyle=**"--"**)  
  
 ax1.set\_yticks([])  
 ax1.set\_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])  
  
 *# piešiamas antras grafikas pavaizduoti sugrupuotus taškus* colors = cm.nipy\_spectral(np.array(cluster\_labels).astype(float) / n\_clusters)  
 ax2.scatter(data[:, 0], data[:, 1], marker=**'.'**, s=30, lw=0, alpha=0.7,  
 c=colors, edgecolor=**'k'**)  
  
 *# nupiešti centroido rutuliukus* ax2.scatter(centers[:, 0], centers[:, 1], marker=**'o'**,  
 c=**"white"**, alpha=1, s=200, edgecolor=**'k'**)  
  
 **for** i, c **in** enumerate(centers):  
 ax2.scatter(c[0], c[1], marker=**'$%d$'** % i, alpha=1,  
 s=50, edgecolor=**'k'**)  
  
 ax2.set\_title(**" "**)  
 ax2.set\_xlabel(**"Feature space for the 1st feature"**)  
 ax2.set\_ylabel(**"Feature space for the 2nd feature"**)  
  
 plt.suptitle((**"Silhouette analysis for KMeans clustering on sample data "  
 "with n\_clusters = %d"** % n\_clusters),  
 fontsize=14, fontweight=**'bold'**)  
 print(**"--------------------------------------"**)  
  
plt.show()