



pesikj /
PythonProDataScience



<> Code

Issues 1

Pull requests

Actions

Projects

Security

Insights

PythonProDataScience / 08 / lekce.ipynb



pesikj Rok 2024

last month



1174 lines (1174 loc) · 1.09 MB

Preview

Code

Blame

Raw



Lekce 8

In [23]:

```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

Popis importů

- TSNE redukce dimenzionality (počtu sloupců) s využitím algoritmu TSNE, dokumentace je [zde](#)
- KMeans shlukování s využitím algoritmu K-Means, dokumentace je [zde](#)
- StandardScaler - objekt pro normalizaci dat, dokumentace je [zde](#)
- OneHotEncoder provádí One Hot Encoding zadaných dat, dokumentace je [zde](#)
- DecisionTreeClassifier - klasifikátor využívající algoritmus rozhodovacího stromu, dokumentace je [zde](#)
- GridSearchCV hledá nejlepší parametry klasifikátoru ze zadaného rozsahu podle zadané metriky, dokumentace je [zde](#)

Unsupervised learning

Doposud jsme se zabývali klasifikací, jako jednou s podtříd supervised learning, tedy úloh, kde předem známe správné odpovědi. Model se pomocí algoritmu a trénovacích dat, které obsahují tyto správné odpovědi, "naučí" jak jednotlivé třídy vypadají, aby později mohl přiřazovat nová data do tříd. V případě regresní úlohy nová data nedostanou přiřazenou třídu, ale nějakou hodnotu jako reálné číslo. Obecný koncept je ale stejný.

V této lekci se budeme věnovat úlohám bez supervise, tedy takovým, kde **předem neznáme správné odpovědi**, nebo tyto odpovědi vůbec neexistují.

Příklady takových úloh jsou:

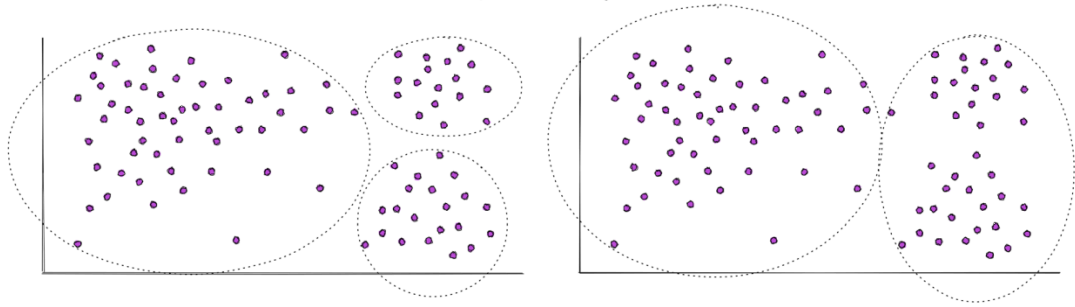
- rozdělování dat na shluky (clustering), na rozdíl od klasifikačních úloh **ale nevíme o žádném záznamu, do kterého shluku patří, ani nevíme, kolik by takových shluků mělo být**,
- hledání anomálií v datech (tj. hledáme něco neobvyklého, například neobvyklé datové toky v síti nebo podivné údaje z nějakého měřicího zařízení).

Shlukování dat (clustering)

Na obrázku například vidíme data, nad kterými by se dalo přemýšlet jako nad dvěma

nebo třemi shluky. Důležité je, že o zadném udaji nevíme, do kterého shluku by měl patřit! Rozdělení je tedy čistě na nás. Je na nás, na kolik shluků data rozdělíme.

Učení bez učitele, Unsupervised learning

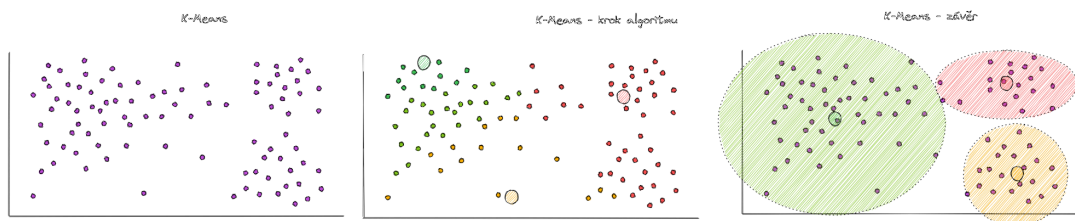


K-means

Ukážeme si shlukovací algoritmus K-means. *K* proto, že očekává zadaný počet shluků od uživatele, a *means* zde znamená průměr - střed shluku neboli *centroid*.

Algoritmus k-means se skládá z následujících kroků:

1. **Výběr počtu shluků (*k*):** Algoritmus k-means vyžaduje, abyste předem určili počet shluků, které chcete vytvořit. Počet těchto shluků se označuje jako '*k*'. Algoritmus můžeme spouštět opakovaně a různým počtem nastavených shluků a sledovat, jak dobře takový počet "sedí" na naše data.
2. **Náhodná inicializace středů shluků:** Poté, co jsme určili hodnotu '*k*', algoritmus náhodně vybere '*k*' bodů z datové sady. Tyto body se stávají prvními 'centroidy' - tedy středy shluků.
3. **Přiřazení bodů k nejbližšímu středům shluků:** Každý bod v datové sadě se nyní přiřadí k tomu centroidu, ke kterému má nejkratší vzdálenost. Tím se vytvoří dočasné shluky.
4. **Přesu středů shluků:** Poté, co jsou všechny body přiřazeny k shlukům, algoritmus spočítá nový střed každého shluku jako průměr všech bodů, které do tohoto shluku patří. Následně je *centroid* přesunutý do středu shluku.
5. **Opakování kroků 3 a 4:** Algoritmus nyní opakuje kroky 3 a 4, dokud se středy shluků nepřestanou posouvat nebo dokud se nevyčerpá předem stanovený počet iterací. Každá iterace upřesňuje středy shluků a přiřazuje body k novým středům.



Úloha, která nás bude provázet lekcí, se klasifikace uživatelů kreditních karet, která jsou v souboru [CC_GENERAL.CSV](#).

Zdroj dat

Popisy sloupců (features):

- CUST_ID: Identifikace držitele kreditní karty (Kategorické)

- **BALANCE:** Zůstatek na účtu pro nákupy
- **BALANCE_FREQUENCY:** Jak často se aktualizuje zůstatek, skóre mezi 0 a 1 (1 = často aktualizováno, 0 = nečasto aktualizováno)
- **PURCHASES:** Částka nákupů provedených z účtu
- **ONEOFF_PURCHASES:** Maximální částka nákupu provedená najednou
- **INSTALLMENTS_PURCHASES:** Částka nákupu provedená na splátky
- **CASH_ADVANCE:** Hotovost předem poskytnutá uživatelem
- **PURCHASES_FREQUENCY:** Jak často se provádějí nákupy, skóre mezi 0 a 1 (1 = často nakupováno, 0 = nečasto nakupováno)
- **ONEOFFPURCHASESFREQUENCY:** Jak často se provádějí nákupy najednou (1 = často nakupováno, 0 = nečasto nakupováno)
- **PURCHASESINSTALLMENTSFREQUENCY:** Jak často se provádějí nákupy na splátky (1 = často prováděno, 0 = nečasto prováděno)
- **CASHADVANCEFREQUENCY:** Jak často se platí hotovost předem
- **CASHADVANCECTR:** Počet transakcí provedených s "Cash in Advanced"
- **PURCHASES_TRX:** Počet provedených nákupních transakcí
- **CREDIT_LIMIT:** Limit kreditní karty pro uživatele
- **PAYMENTS:** Částka platby provedená uživatelem
- **MINIMUM_PAYMENTS:** Minimální částka plateb provedených uživatelem
- **PRCFULLPAYMENT:** Procento úplné platby uhrané uživatelem
- **TENURE:** Doba trvání služby kreditní karty pro uživatele

In [24]:

```
X = pd.read_csv("CC_GENERAL.csv")
X.head()
```

Out[24]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	I
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10005	817.714335	1.000000	16.00	16.00	
4	C10006	1809.828751	1.000000	1333.28	0.00	



In [25]:

```
X = X.head(1000)
```

Určitě musíme z dat odebrat sloupec `CUST_ID`.

In [26]:

```
X = X.drop(columns=["CUST_ID"])
```

In [27]:

```
X.shape
```

Out[27]: (1000, 17)

Data musíme určitě normalizovat s využitím `StandardScaler()`. U unsupervised learning nedělíme data na trénovací a testovací, takže normalizujeme všechna data

najednou.

```
In [28]: scaler = StandardScaler()  
X = scaler.fit_transform(X)
```

Snížení dimensionality

U shlukování je vždy dobré si data **vizualizovat**. Jak ale na to? Máme celkem 17 proměnných, to je jako 17-ti dimenzionální prostor. Vizualizovat umíme ve dvou dimenzích, možná i ve třech. V jedenácti ale těžko.

Modul `scikit-learn` má k dispozici nástroje na snížení dimensionality ("sníže počtu proměnných"). Snížení počtu proměnných má několik výhod. Nejenom, že můžeme pak data snáz vizualizovat, ale zbavíme se problému kolinearity (korelace dvou nebo více proměnných), a navíc snížíme čas a výkon potřebný pro další modelování.

Redukce dimensionality je postup, který se snaží zachovat původní strukturu dat a minimalizovat ztrátu informace, ke které dojde při odebrání některých sloupců.

Jedna z metod, která nám sníží počet proměnných ("redukuje dimensionality") se jmenuje t-SNE (t-distributed Stochastic Neighbor Embedding). Modul `scikit-learn` tuto metodu implementuje, pojďme jí vyzkoušet.

TSNE má několik důležitých parametrů:

- `n_components` říká, kolik dimenzí (proměnných) chceme. U t-SNE typicky volíme hodnotu 2 nebo 3.
- `perplexity` určuje, podle kolika sousedů se má metoda t-SNE řídit. Čím vyšší je náš dataset, tím vyšší hodnotu parametru nastavíme. Běžně se používají hodnoty mezi 5 a 50.

Princip metody spočívá v tom, že se snaží při redukci počtu dimenzí udržovat body, které jsou si v původní dimenzi blízké, blízko u sebe.

```
In [29]: tsne = TSNE(  
        n_components=2,  
        random_state=42,  
    )  
X = tsne.fit_transform(X)
```

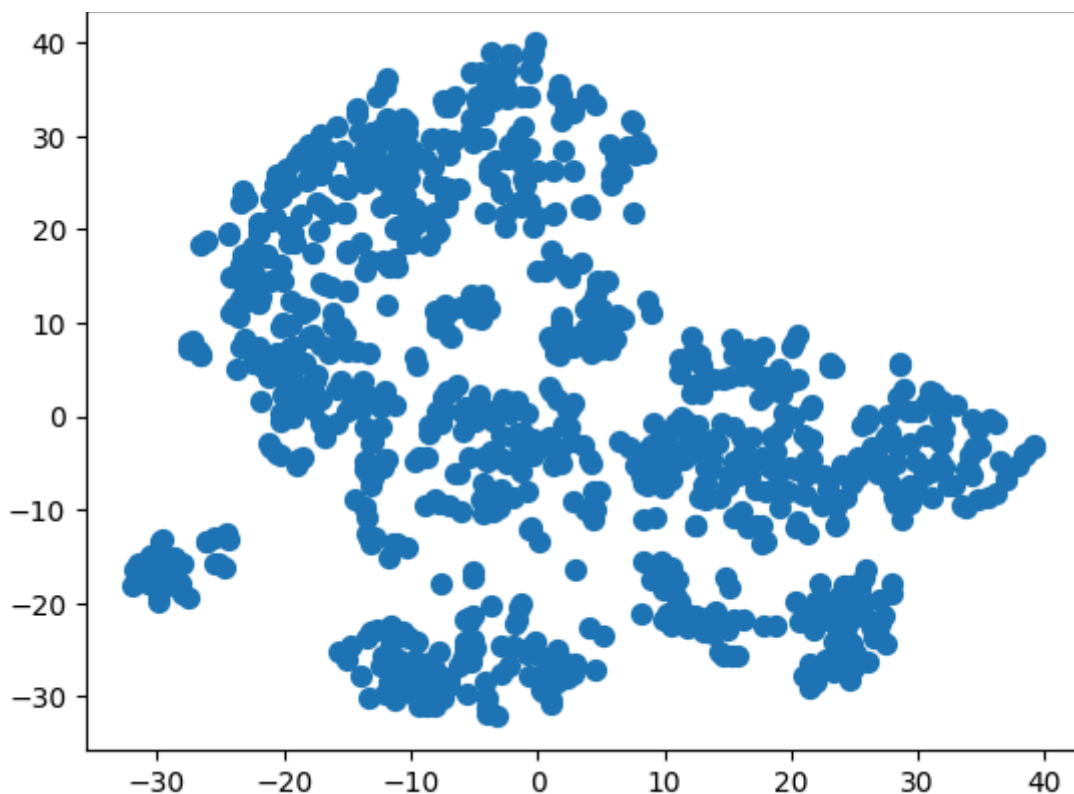
Nyní jsou naše data redukována jen do dvou dimenzí:

```
In [30]: X.shape
```

```
Out[30]: (1000, 2)
```

```
In [31]: plt.scatter(X[:, 0], X[:, 1], s=50)
```

```
Out[31]: <matplotlib.collections.PathCollection at 0x1bd1e4c6780>
```



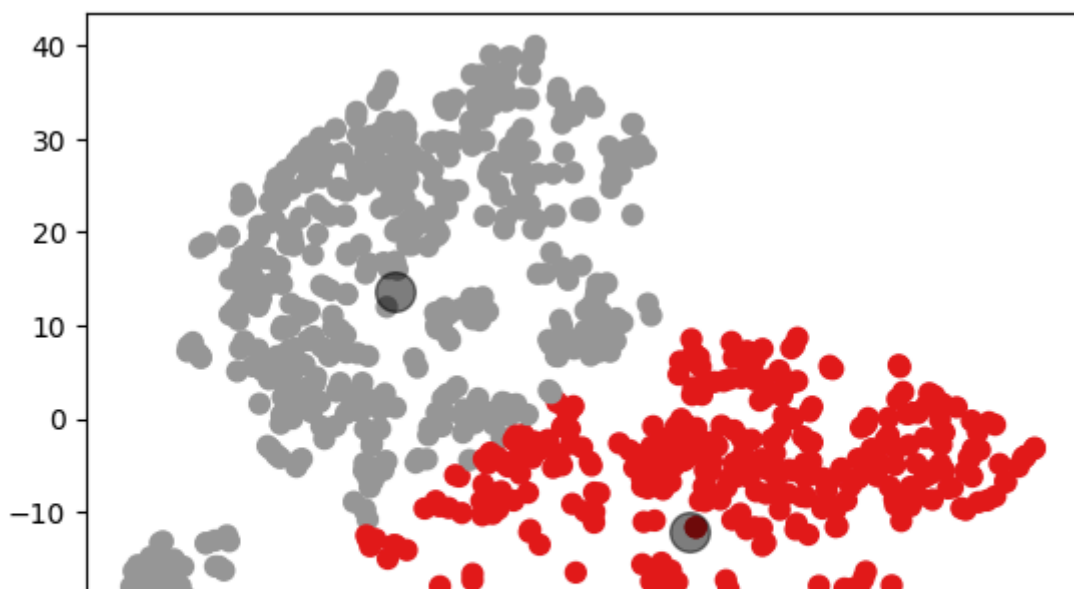
Můžeme spustit algoritmus K-means. Uvažujme nyní, že chceme vytvořit 2 shluky.

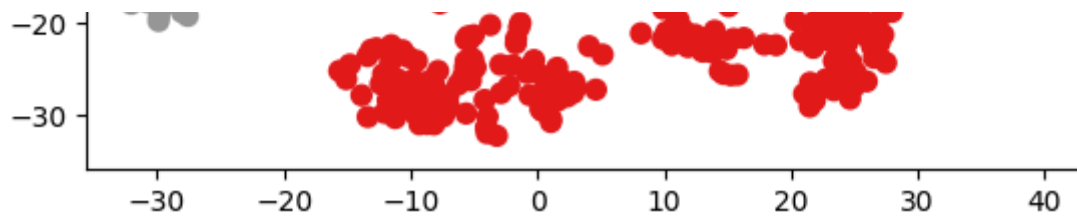
```
In [32]: model = KMeans(n_clusters=2, random_state=42, n_init="auto")
labels = model.fit_predict(X)
```

Výsledek si můžeme zobrazit graficky. Každý bod obarvíme do barvy dle shluku, do kterého byl přiřazen, a doplníme i centroidy.

```
In [33]: # Zobrazení bodů
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap="Set1")
# Zjištění centroidů
centers = model.cluster_centers_
# Zobrazení centroidů
plt.scatter(centers[:, 0], centers[:, 1], c="black", s=200, alpha=0.5)
```

```
Out[33]: <matplotlib.collections.PathCollection at 0x1bd1e66cb30>
```





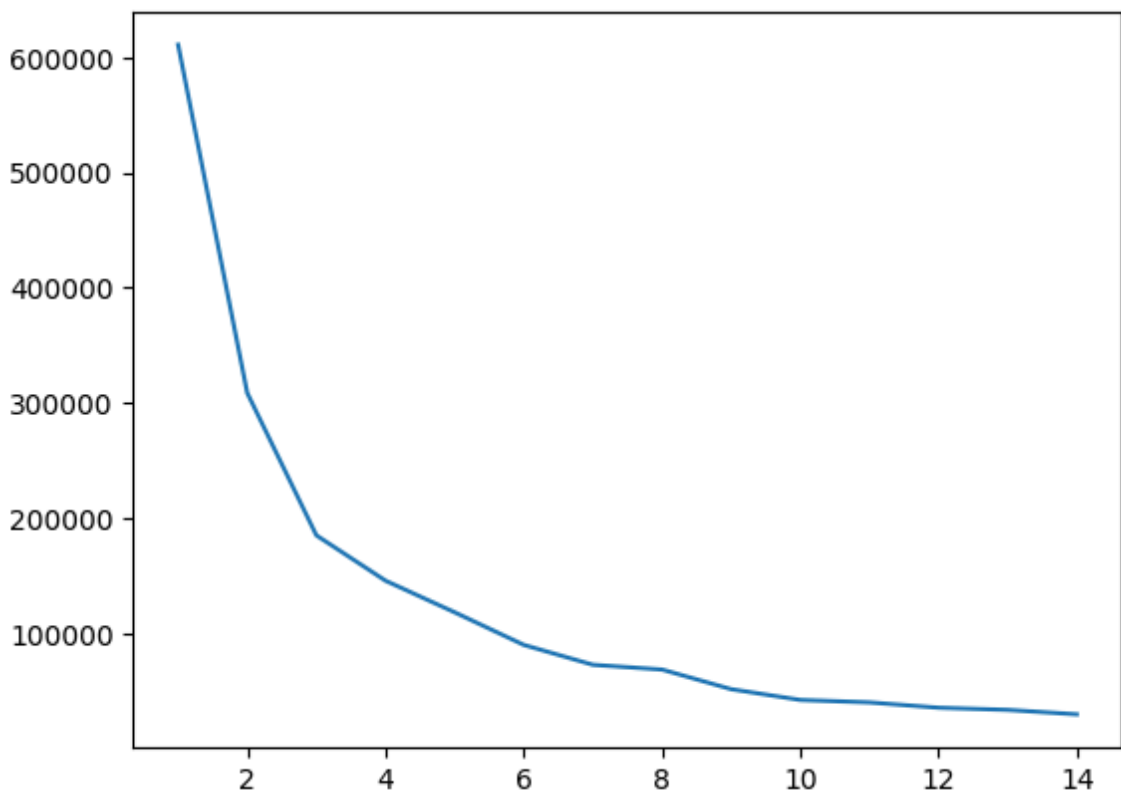
Dva shluky ale nemusí být to správné číslo. Jak poznat, kolik shluků bychom měli mít? K tomu můžeme využít například metriku `inertia`. Tato metrika je výpočet sumy kvadratických vzdáleností všech bodů v shluku ke středu shluku, neboli centroidu. Čím více máme shluků, tím spíše bude vzdálenost klesat. Ale je pravděpodobné, že tempo poklesu bude s růstem klastrů klesat. Není tedy naším cílem vytvořit model s obrovským množstvím shluků (takový by nám toho stejně moc neřekl). Ve skutečnosti chceme vybrat počet, ve kterém počet klastrů výrazně zpomaluje.

```
In [34]: distances = []
cluster_counts = range(1, 15)

for k in cluster_counts:
    model = KMeans(n_clusters=k, n_init="auto").fit(X)
    model.fit(X)
    distances.append(model.inertia_)

plt.plot(cluster_counts, distances,)
```

Out[34]: [`matplotlib.lines.Line2D` at `0x1bd171a4e60`]

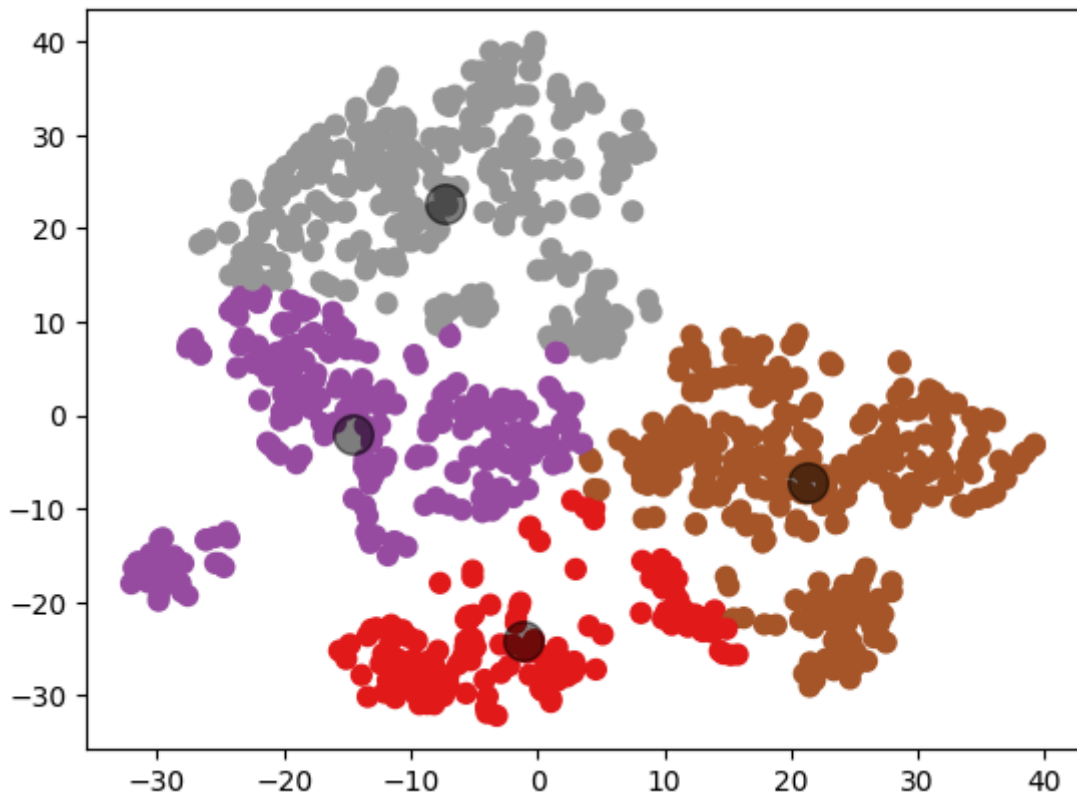


```
In [35]: model = KMeans(n_clusters=4, random_state=42, n_init="auto")
labels = model.fit_predict(X)

# Zobrazení bodů
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap="Set1")
# Zjištění centroidů
```

```
centers = model.cluster_centers_  
# Zobrazení centroidů  
plt.scatter(centers[:, 0], centers[:, 1], c="black", s=200, alpha=0.5)
```

Out[35]: <matplotlib.collections.PathCollection at 0x1bd1e73a060>



Připojení informace o shluku k datům

Cluster můžeme připojit k původním datům. Protože u našich dat jsme provedli redukci dimenzionality a tím pádem ztratili původní sloupce, načteme si data znovu ze souboru. Do pandas tabulky pak můžeme přidat `cluster` každé hodnoty jako sloupec.

```
In [36]: X = pd.read_csv("CC_GENERAL.csv")  
X = X.drop(columns="CUST_ID")  
X = X.head(1000)  
X["cluster"] = labels  
X.head()
```

Out[36]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLME
0	40.900749	0.818182	95.40	0.00	
1	3202.467416	0.909091	0.00	0.00	
2	2495.148862	1.000000	773.17	773.17	
3	817.714335	1.000000	16.00	16.00	
4	1809.828751	1.000000	1333.28	0.00	

Clustery, které jsme spočítali, jsou poměrně anonymní. Můžeme si ale vypočítat průměrné hodnoty jednotlivých sloupců (features) pro jednotlivé clustery a to s


průměrné hodnoty jednotlivých sloupců (features) pro jednotlivé clustery, a to s využitím `pandas` agregace.

Níže třeba vidíme, že shluky se výrazně liší ve sloupci `BALANCE`, přičemž nejnižší hodnotu má shluk 0 a nejvyšší hodnotu shluk 3. Ve sloupcích `PURCHASES`, `ONEOFF_PURCHASES`, `INSTALLMENTS_PURCHASES` a `PURCHASES_FREQUENCY` má nejvyšší hodnotu shluk 2. Pro zákazníky a zákaznice ze shluku 2 by tedy mohly být nejzajímavější bankovní produkty související s nákupem, jako třeba výhodné spotřebitelské úvěry, slevu na platbu kartou atd. Zákazníci ve shluku 3 mají více volné hotovosti, ale méně nakupují, můžeme jim tedy nabídnout například spořicí a investiční produkty. Zákazníkům a zákaznicím ze shluku 0 bychom mohli nabídnout kontokorentní úvěr, protože u nich hrozí, že se jejich bankovní účet dostane do mínusu.

```
In [37]: X.groupby("cluster").mean()
```

```
Out[37]:
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	PURCHASES_FREQUENCY
cluster						
0	231.064785	0.597997	810.297114	333.722148	14.559360	0.509142
1	1333.444765	0.947868	514.621829	392.494634	16.278389	0.804507
2	2473.193768	0.984741	4053.074618	2570.506445	19.298462	0.984741
3	4336.432490	0.983384	614.633355	436.800132	16.278389	0.804507



Můžeme se podívat i na počet zákazníků a zákaznic v jednotlivých shlucích.

```
In [ ]: X.groupby("cluster").size()
```

```
Out[ ]: cluster
0      149
1      246
2      301
3      304
dtype: int64
```

Další zdroje

- [Introduction to t-SNE](#)
- [Understanding Clustering: Unveiling the Principles and Techniques of K-mean Clustering](#)

Cvičení

Vzorky vína

Pracuj se souborem [wine-regions.csv](#). Každý záznam obsahuje informace o vzorku vína. Všechny vzorky pocházejí z jednoho regionu v Itálii, ale v datech se nachází několik různých odrůd. Naší úlohou bude data uspořádat do shluků, které by mohly odpovídat

odrůdám. Na začátku uvažujme, že nevíme, ke které odrůdě jednotlivé záznamy patří, a nevíme ani to, kolik odrůd v datech je.

- Proveď redukci dimenzionality na 2 dimenze. Vykresli data pomocí grafu. Poznáš z grafu, kolik by byl nejlepší počet shluků? Následně zkus určit metriku *inertia* pro 2 až 10 shluků. Pro jaký počet shluků dochází k výraznému snížení tempa jejího poklesu? Následně pro tento počet vykresli graf včetně centroidů.
- Redukce dimenzionality není nutná. Zkus nyní znovu načíst data a vykresli graf metriky *inertia* pro 1 až 10 shluků na datech s původní dimenzí. Změnil se graf nějak výrazně? Změnilo by to tvé rozhodnutí o počtu shluků?
- Vypočítej průměrné hodnoty pro jednotlivé sloupce a pro jednotlivé shluky. Porovnej, jak se průměrné hodnoty mezi shluky liší.

Bonus

Další metrikou, kterou můžeme využít, je takzvaný *Silhouette* koeficient. Ten měří průměrnou vzdálenost mezi bodem a všemi body, které jsou ve stejném shluku, a pak vzdálenost mezi bodem a všemi body v nejbližším jiném shluku.

Jeho hodnota se nachází mezi 1 a -1. Čím vyšší je tento koeficient, tím lépe definované jsou clustery (body se nachází blízko svému shluku a daleko od všech ostatních). Pokud má hodnotu kolem nuly, značí to, že se naše shluky překrývají (existují body na rozhraní dvou shluků). Záporná hodnota značí, že body z odlišných shluků jsou si blíže než body ze stejných shluků.

Hodnotu koeficientu zjíš pomocí funkce `silhouette_score()`. Příklad volání funkce je níže.

```
silhouette_score(X, labels)
```

Vytvoř graf hodnoty koeficientu pro 2 až 10 shluků. V jakém případě je hodnota koeficientu nejvyšší?

Řešení příkladu je [zde](#).

