Parameter Estimation, Probabilistic Graphical Models and Grouping

February 5, 2021

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

0.1 Strojno učenje 2020/2021

http://www.fer.unizg.hr/predmet/su

0.1.1 Laboratorijska vježba 4: Procjena parametara, probabilistički grafički modeli i grupiranje

Verzija: 0.5
Zadnji put ažurirano: 6. siječnja 2021.

Rok za predaju: **18. siječnja 2021. u 06:00h**

0.1.2 Upute

Četvrta laboratorijska vježba sastoji se od tri zadatka. U nastavku slijedite upute navedene u ćelijama s tekstom. Rješavanje vježbe svodi se na **dopunjavanje ove bilježnice**: umetanja ćelije ili više njih **ispod** teksta zadatka, pisanja odgovarajućeg kôda te evaluiranja ćelija.

Osigurajte da u potpunosti **razumijete** kôd koji ste napisali. Kod predaje vježbe, morate biti u stanju na zahtjev asistenta (ili demonstratora) preinačiti i ponovno evaluirati Vaš kôd. Nadalje, morate razumjeti teorijske osnove onoga što radite, u okvirima onoga što smo obradili na predavanju. Ispod nekih zadataka možete naći i pitanja koja služe kao smjernice za bolje razumijevanje gradiva (**nemojte pisati** odgovore na pitanja u bilježnicu). Stoga se nemojte ograničiti samo na to da riješite zadatak, nego slobodno eksperimentirajte. To upravo i jest svrha ovih vježbi.

Vježbe trebate raditi **samostalno** ili u **tandemu**. Možete se konzultirati s drugima o načelnom načinu rješavanja, ali u konačnici morate sami odraditi vježbu. U protivnome vježba nema smisla.

[1]:
```
# Učitaj osnovne biblioteke...
import sklearn
from sklearn.metrics import silhouette_samples, silhouette_score
```

```
import numpy as np
import matplotlib.pyplot as plt
import pgmpy as pgm
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
# Remove warnings
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
def plot_silhouette(n_clusters, X):
    # Kôd preuzet s http://scikit-learn.org/stable/auto_examples/cluster/
 ↪plot_kmeans_silhouette_analysis.html

    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        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

        cmap = plt.cm.get_cmap("Dark2")
        color = cmap(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)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_xlabel("Vrijednosti koeficijenta siluete")
    ax1.set_ylabel("Oznaka grupe")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    cmap = plt.cm.get_cmap("Dark2")
    colors = cmap(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X[:, 0], X[:, 1], marker='o', s=30, lw=0, alpha=1,
                c=colors, edgecolor='k')

    # Labeling the clusters
    centers = clusterer.cluster_centers_
    # Draw white circles at cluster centers # changed to black
    ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                c="black", alpha=1, s=100, edgecolor='k')

    for i, c in enumerate(centers):
        ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                    s=80, edgecolor='k')

    ax2.set_xlabel(r"$x_1$")
    ax2.set_ylabel(r"$x_2$")
```

```
    plt.show()
```

0.1.3 1. Procjena maksimalne izglednosti i procjena maksimalne aposteriorne vjerojatnosti

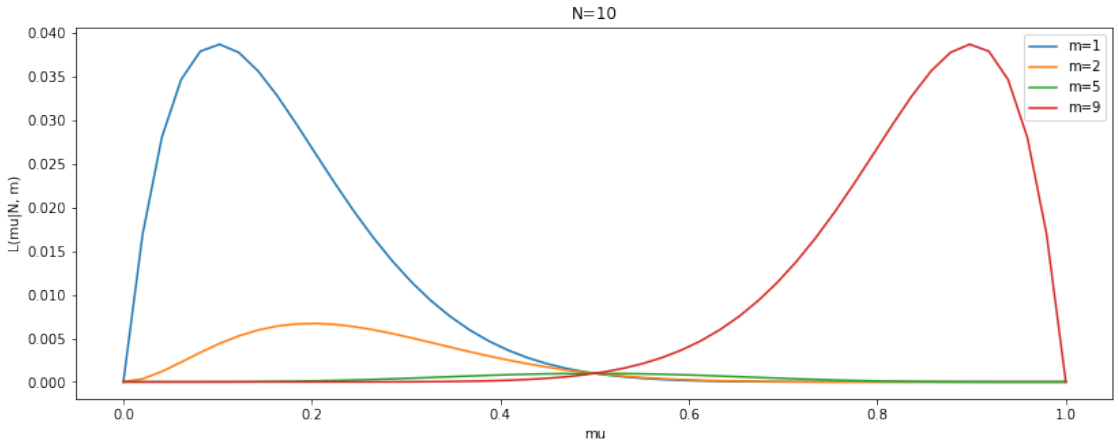
(a) Definirajte funkciju izglednosti $\mathcal{L}(\mu|\mathcal{D})$ za skup $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$ Bernoullijevih varijabli. Neka od N varijabli njih m ima vrijednost 1 (npr. od N bacanja novčića, m puta smo dobili glavu). Definirajte funkciju izglednosti tako da je parametrizirana s N i m, dakle definirajte funkciju $\mathcal{L}(\mu|N, m)$.

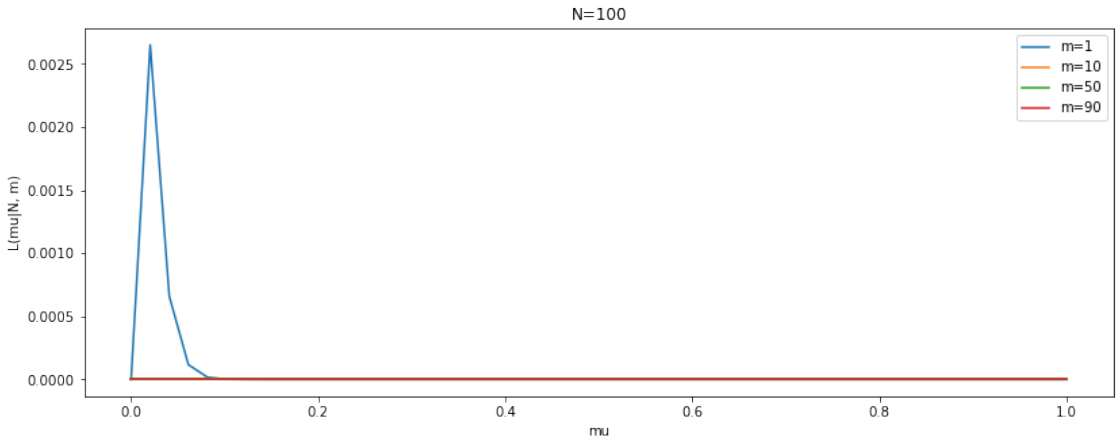
[4]:
```
def L(mu, N, m):
    # Vaš kôd ovdje...
    return mu**m * (1 - mu)**(N - m)
```

(b) Prikažite funkciju $\mathcal{L}(\mu|N, m)$ za (1) $N = 10$ i $m = 1, 2, 5, 9$ te za (2) $N = 100$ i $m = 1, 10, 50, 90$ (dva zasebna grafikona).

[5]:
```
# Vaš kôd ovdje...
mu = np.linspace(0, 1)

plt.figure(figsize=(30, 5))
m1s = [1, 2, 5, 9]
N1 = 10

m2s = [1, 10, 50, 90]
N2 = 100

L1 = [L(mu, N1, m) for m in m1s]

L2 = [L(mu, N1, m) for m in m2s]

for m in m1s:
    plt.subplot(1, 2, 1)
    plt.plot(mu, L(mu, N1, m), label=f'm={m}')

plt.title(f'N={N1}')
plt.xlabel("mu")
plt.ylabel("L(mu|N, m)")
plt.legend()
plt.show()


plt.figure(figsize=(30, 5))
for m in m2s:
    plt.subplot(1, 2, 2)
    plt.plot(mu, L(mu, N2, m), label=f'm={m}')
```

```
plt.title(f'N={N2}')
plt.xlabel("mu")
plt.ylabel("L(mu|N, m)")
plt.legend()
plt.show()
```



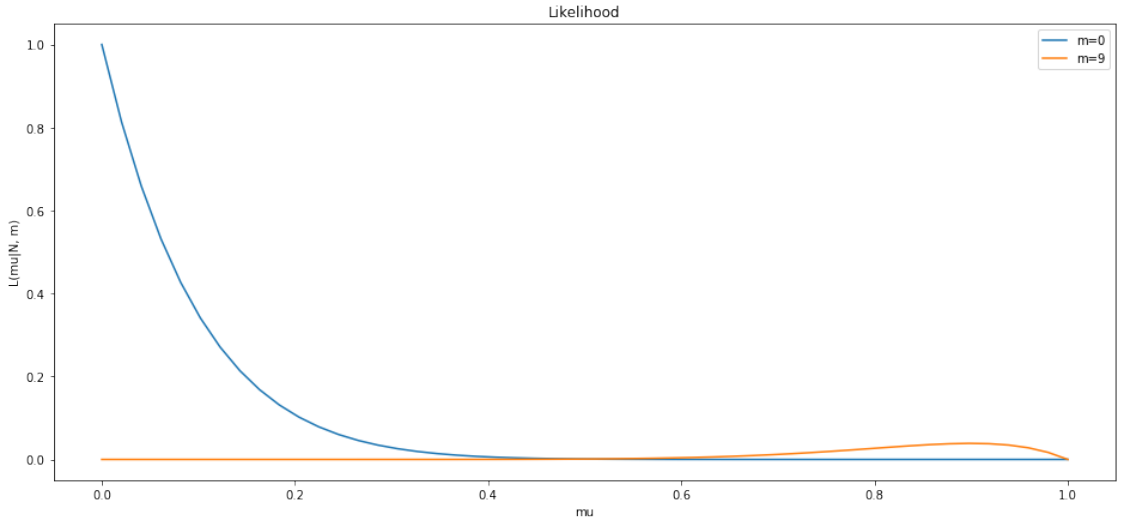


Q: Koja vrijednost odgovara ML-procjenama i zašto?

(c) Prikažite funkciju $\mathcal{L}(\mu|N, m)$ za $N = 10$ i $m = \{0, 9\}$.

[6]:
```
# Vaš kôd ovdje...
plt.figure(figsize=(35, 7))
m = [0, 9]
N = 10
```

```
mu = np.linspace(0, 1)

L1 = L(mu, N, m[0])
L2 = L(mu, N, m[1])

plt.subplot(1, 2, 1)
plt.plot(mu, L1, label="m=" + str(m[0]))
plt.plot(mu, L2, label="m=" + str(m[1]))
plt.legend()
plt.xlabel("mu")
plt.ylabel("L(mu|N, m)")
plt.title("Likelihood")
plt.show()
```

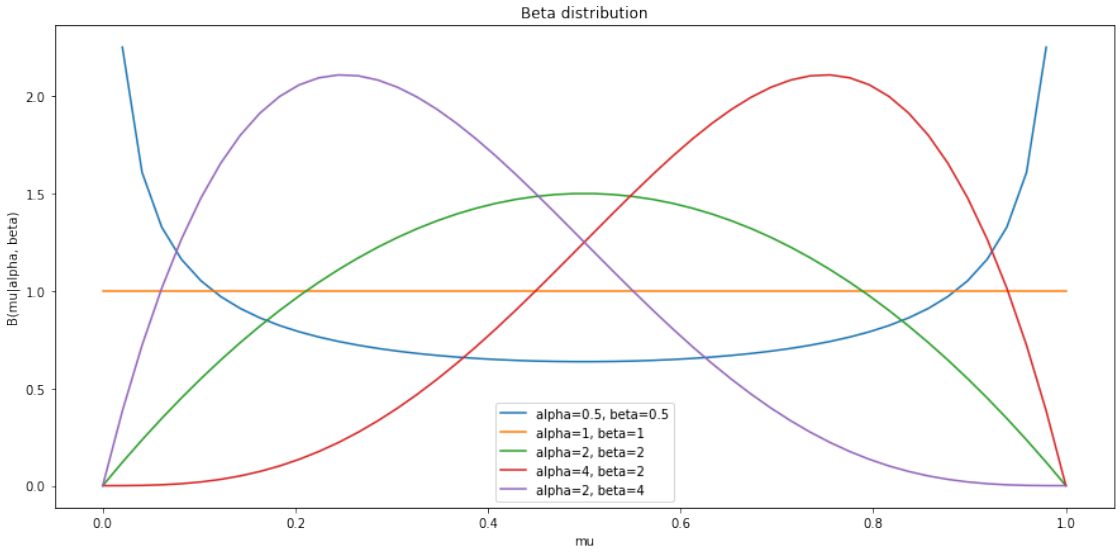


Q: Koja je ML-procjena za μ i što je problem s takvom procjenom u ovome slučaju?

(d) Prikažite beta-distribuciju $B(\mu|\alpha, \beta)$ za različite kombinacije parametara α i β, uključivo $\alpha = \beta = 1$ te $\alpha = \beta = 2$.

[7]:
```
from scipy.stats import beta
plt.figure(figsize=(15, 7))

mu = np.linspace(0, 1)

alphas = [0.5, 1, 2, 4, 2]
betas = [0.5, 1, 2, 2, 4]

for alpha, betac in zip(alphas, betas):
```

```
    plt.plot(mu, beta.pdf(mu, alpha, betac), label=f'alpha={alpha},␣
↪beta={betac}')
    plt.xlabel("mu")
    plt.ylabel("B(mu|alpha, beta)")

plt.legend()
plt.title("Beta distribution")
plt.show()
# Vaš kôd ovdje...
```

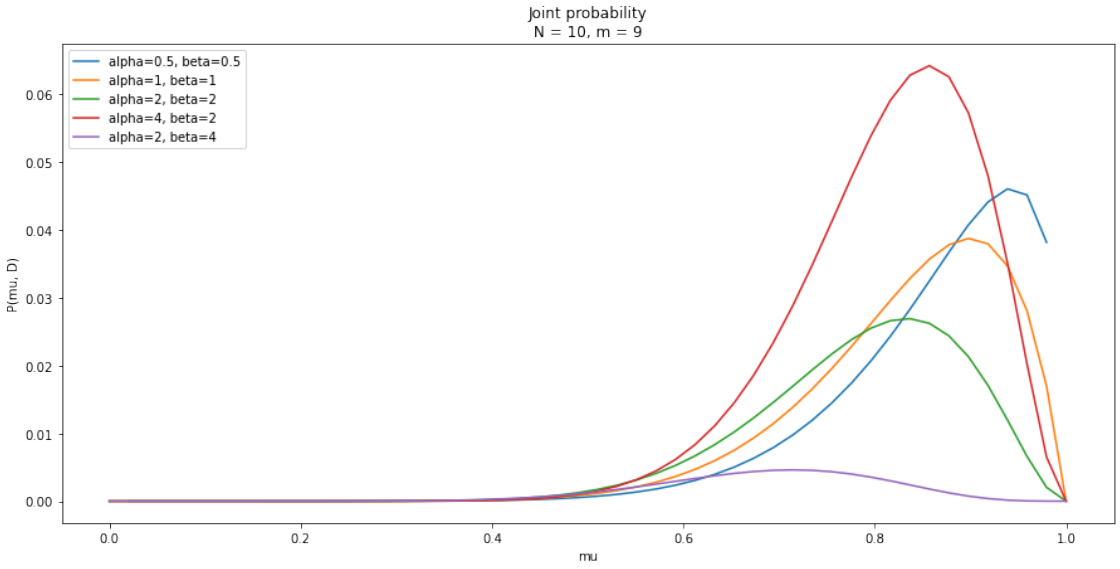


Q: Koje parametere biste odabrali za modeliranje apriornog znanja o parametru μ za novčić za koji mislite da je "donekle pravedan, ali malo češće pada na glavu"? Koje biste parametre odabrali za novčić za koji držite da je posve pravedan? Zašto uopće koristimo beta-distribuciju, a ne neku drugu?

(e) Definirajte funkciju za izračun zajedničke vjerojatnosti $P(\mu, \mathcal{D}) = P(\mathcal{D}|\mu) \cdot P(\mu|\alpha, \beta)$ te prikažite tu funkciju za $N = 10$ i $m = 9$ i nekolicinu kombinacija parametara α i β.

[8]:
```
# Vaš kôd ovdje...
def P(mu, N, m, alpha, betac):
    return L(mu, N, m) * beta.pdf(mu, alpha, betac)

mu = np.linspace(0, 1)
N = 10
m = 9

plt.figure(figsize=(15, 7))
alphas = [0.5, 1, 2, 4, 2]
betas = [0.5, 1, 2, 2, 4]
```

```
for alpha, betac in zip(alphas, betas):
    plt.plot(mu, P(mu, N, m, alpha, betac), label=f'alpha={alpha},␣
 ↪beta={betac}')

plt.xlabel("mu")
plt.ylabel("P(mu, D)")
plt.title(f'Joint probability\nN = {N}, m = {m}')
plt.legend()
plt.show();
```

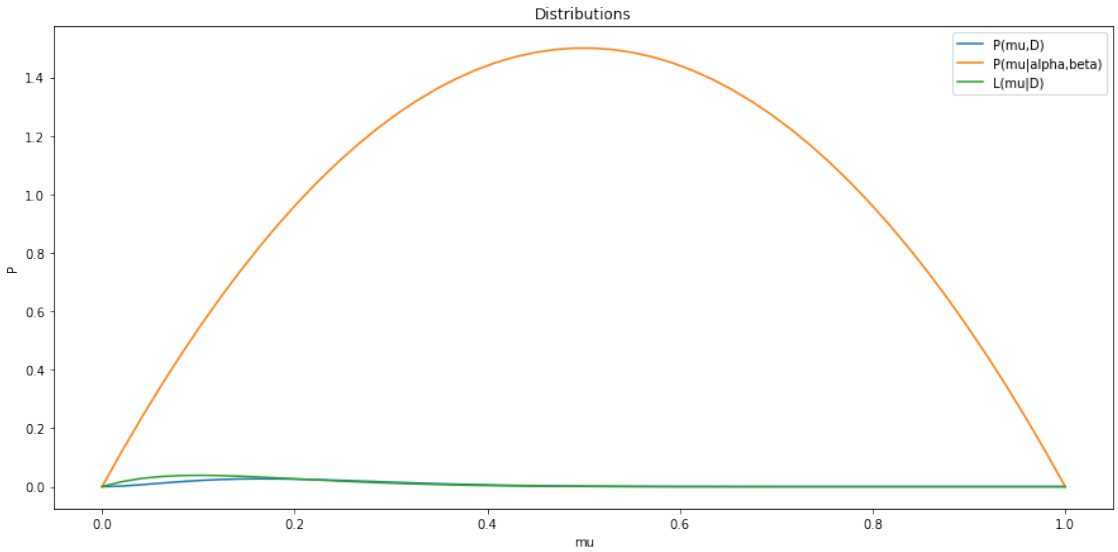


Q: Koje vrijednosti odgovaraju MAP-procjeni za μ? Usporedite ih sa ML-procjenama.

(f) Za $N = 10$ i $m = 1$, na jednome grafikonu prikažite sve tri distribucije: $P(\mu, \mathcal{D})$, $P(\mu|\alpha, \beta)$ i $\mathcal{L}(\mu|\mathcal{D})$.

[9]:
```
# Vaš kôd ovdje...
plt.figure(figsize=(15, 7))

mu = np.linspace(0, 1)
N = 10
m = 1

a = 2
b = 2

plt.plot(mu, P(mu, N, m, a, b), label="P(mu,D)")
plt.plot(mu, beta.pdf(mu, a, b), label="P(mu|alpha,beta)")
```

```
plt.plot(mu, L(mu, N, m), label="L(mu|D)")
plt.legend()
plt.title("Distributions")
plt.xlabel("mu")
plt.ylabel("P");
```

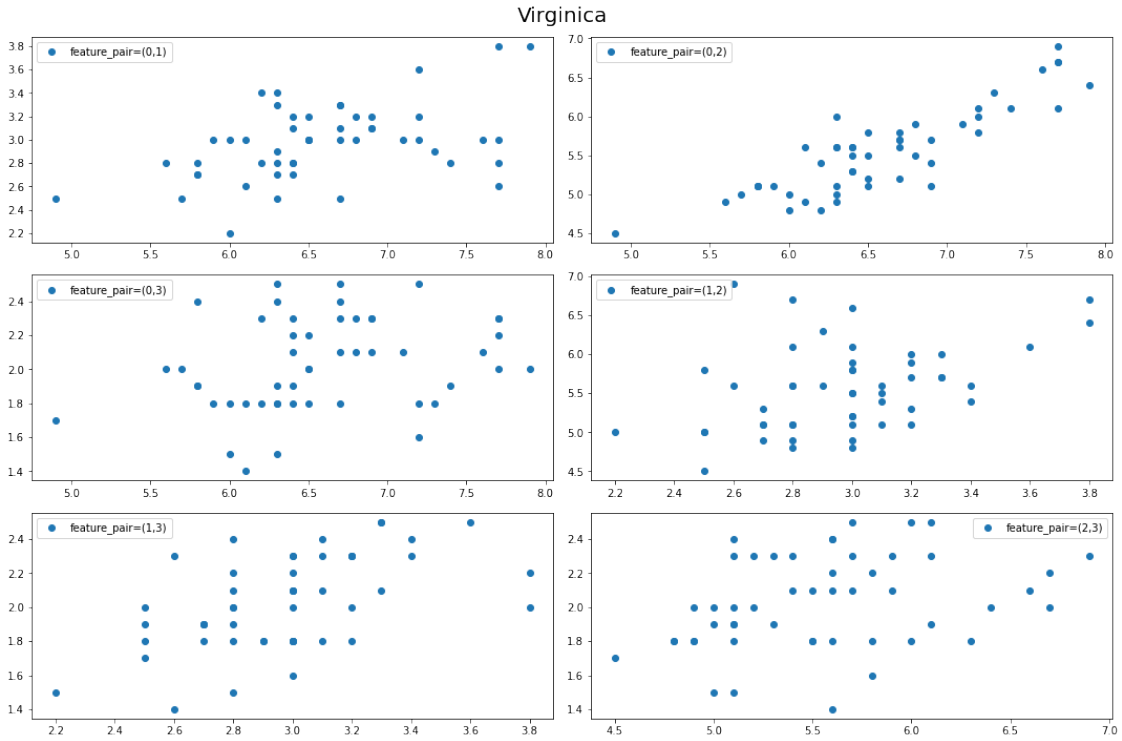


(g) Učitajte skup podataka *Iris* korištenjem funkcije `load_iris`. Taj skup sadrži $n = 4$ značajke i $K = 3$ klase. Odaberite jednu klasu i odaberite sve primjere iz te klase, dok ostale primjere zanemarite (**u nastavku radite isključivo s primjerima iz te jedne klase**). Vizualizirajte podatke tako da načinite 2D-prikaze za svaki par značajki (šest grafikona; za prikaz je najjednostavnije koristiti funkciju `scatter`).

NB: Mogla bi Vam dobro doći funkcija `itertools.combinations`.

[10]:
```
from sklearn.datasets import load_iris
import itertools as it

# Vaš kôd ovdje...
fig = plt.figure(figsize=(15, 10))

iris = load_iris()
setosa = iris.data[iris.target == 0]
versicolor = iris.data[iris.target == 1]
virginica = iris.data[iris.target == 2]

petal = virginica

title = 'Virginica' if np.all(petal == virginica) else 'Versicolor' if np.
 ↪all(petal == versicolor) else 'Setosa'
```

```
fig.suptitle(title, fontsize=20)
subplot_index = 0
for pair in it.combinations(range(4), 2):
    subplot_index += 1
    _1, _2 = pair
    plt.subplot(3, 2, subplot_index)
    feature_col_1 = petal[:,_1]
    feature_col_2 = petal[:,_2]
    plt.scatter(feature_col_1, feature_col_2, label=f'feature_pair=({_1},{_2})')
    plt.legend()

fig.tight_layout()
plt.show()
```



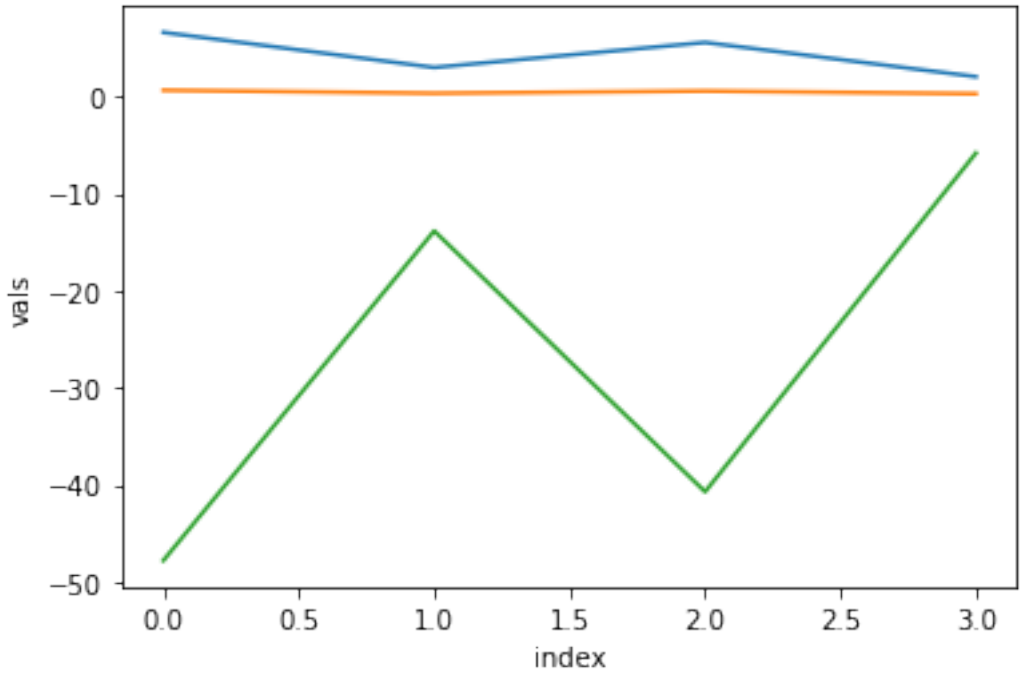
(h) Pogledajte opis modula `stats` te proučite funkciju `norm`. Implementirajte funkciju log-izglednosti za parametre μ i σ^2 normalne distribucije.

[11]:
```
from scipy.stats import norm

def L_gauss(x, mi, sigma):
    # Vaš kôd ovdje...
    return np.sum(norm(mi, sigma).logpdf(x))
```

(i) Izračunajte ML-procjene za (μ, σ^2) za svaku od $n = 4$ značajki iz skupa *Iris*. Ispišite log-izglednosti tih ML-procjena.

[12]:
```
# Vaš kôd ovdje...
N, n = iris.data.shape
print(iris.data.shape)

means = []
sigmas = []
gausses = []
for feature_col_idx in range(n):
    x = petal[:,feature_col_idx]
    mean = np.mean(x)
    sigma = np.std(x)
    gauss = L_gauss(x, np.mean(x), np.std(x))
    print(f'index={feature_col_idx}, mean={mean}, sigma={sigma}, gauss={gauss}')
    means.append(mean)
    sigmas.append(sigma)
    gausses.append(gauss)

plt.plot(range(n), means, label="mean")
plt.plot(range(n), sigmas, label="sigma")
plt.plot(range(n), gausses, label="gauss")
plt.xlabel("index")
plt.ylabel("vals")
plt.show()
```

```
(150, 4)
index=0, mean=6.587999999999998, sigma=0.6294886813914926,
gauss=-47.80455636149741
index=1, mean=2.974, sigma=0.3192553836664309, gauss=-13.858730623718925
index=2, mean=5.5520000000000005, sigma=0.546347874526844,
gauss=-40.721957995876785
index=3, mean=2.0260000000000002, sigma=0.2718896835115301,
gauss=-5.828983146267946
```



(j) Proučite funkciju `pearsonr` za izračun Pearsonovog koeficijenta korelacije. Izračunajte koeficijente korelacije između svih četiri značajki u skupu *Iris*.

[13]:
```
from scipy.stats import pearsonr

# Vaš kôd ovdje...
feature_1 = petal[:, 0]
feature_2 = petal[:, 1]
feature_3 = petal[:, 2]
feature_4 = petal[:, 3]

for pair in it.combinations(range(0, 4), 2):
    _1, _2 = pair
    r, _ = pearsonr(petal[:, _1], petal[:, _2])
    print(f'pearsonr({_1},{_2})={r}')
```

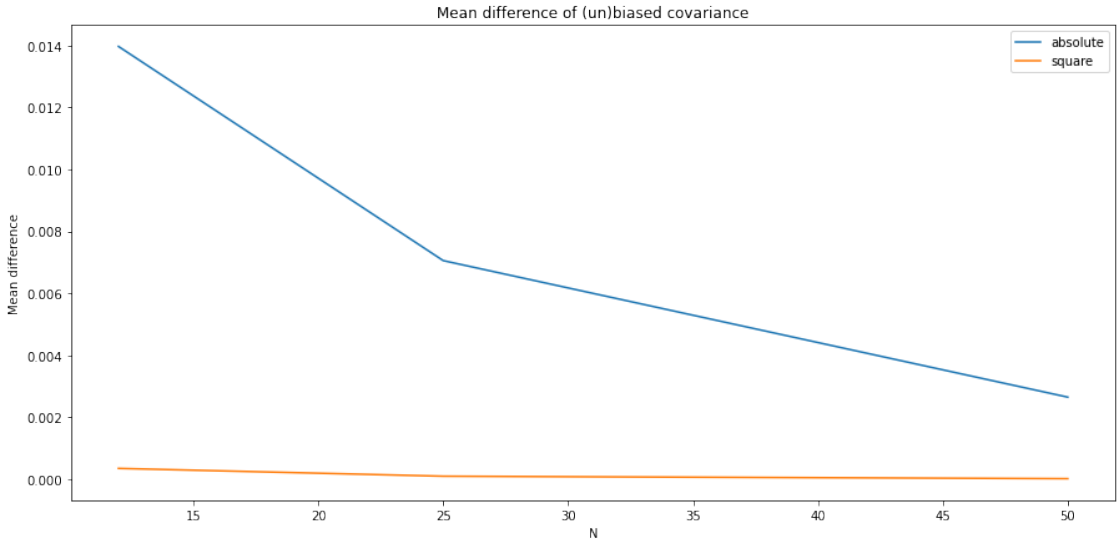
```
pearsonr(0,1)=0.4572278163941132
pearsonr(0,2)=0.8642247329355762
pearsonr(0,3)=0.2811077091573194
pearsonr(1,2)=0.40104457734278554
pearsonr(1,3)=0.5377280262661889
pearsonr(2,3)=0.32210821590031835
```

(k) Proučite funkciju `cov` te izračunajte ML-procjenu za kovarijacijsku matricu za skup *Iris*. Usporedite pristranu i nepristranu procjenu. Pokažite da se razlika (srednja apsolutna i kvadratna)

smanjuje s brojem primjera (npr. isprobajte za $N/4$ i $N/2$ i N primjera).

[14]:
```
# Vaš kôd ovdje...
fig = plt.figure(figsize=(15, 7))
divisors = [4, 2, 1]
N = [len(petal) // divisor for divisor in divisors]
print(N)

abs_diff = []
square_diff = []

for n in N:
    biased_estimation = np.cov(petal[:n], rowvar=False, bias=True)
    unbiased_estimation = np.cov(petal[:n], rowvar=False, bias=False)

    absolute_mean_difference = np.mean(np.abs(unbiased_estimation -␣
 ↪biased_estimation))
    square_mean_difference = np.mean(np.square(unbiased_estimation -␣
 ↪biased_estimation))

    abs_diff.append(absolute_mean_difference)
    square_diff.append(square_mean_difference)

    print(f'N={n}, mean(abs)={absolute_mean_difference},␣
 ↪mean(square)={square_mean_difference}')


plt.plot(N, abs_diff, label="absolute")
plt.plot(N, square_diff, label="square")
plt.xlabel("N")
plt.ylabel("Mean difference")
plt.legend()
plt.title("Mean difference of (un)biased covariance")
plt.show()
```

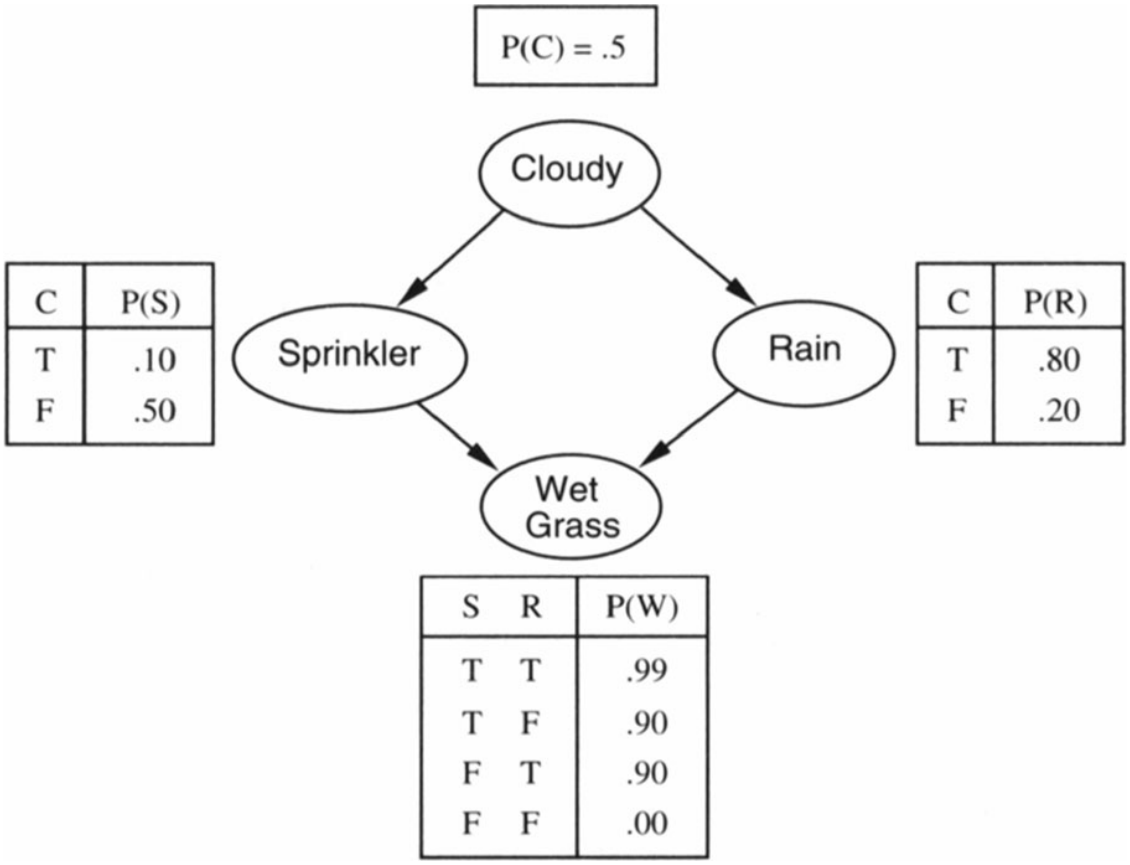
```
[12, 25, 50]
N=12, mean(abs)=0.013972143308080824, mean(square)=0.0003439149450032212
N=25, mean(abs)=0.007054833333333175, mean(square)=9.173363433333285e-05
N=50, mean(abs)=0.0026454081632652903, mean(square)=1.246603536026639e-05
```



0.1.4 2. Probabilistički grafički modeli – Bayesove mreže

Ovaj zadatak bavit će se Bayesovim mrežama, jednim od poznatijih probabilističkih grafičkih modela (*probabilistic graphical models*; PGM). Za lakše eksperimentiranje koristit ćemo programski paket `pgmpy`. Molimo Vas da provjerite imate li ovaj paket te da ga instalirate ako ga nemate. Upute se nalaze na gornjoj poveznici. Za korisnike Anaconde, najlakše je upisati `conda install -c ankurankan pgmpy` (ili `pip install pgmpy` ako ne prolazi) unutar Anaconda Prompta (i ponovno pokrenuti Jupyter).

(a) Prvo ćemo pogledati udžbenički primjer s prskalicom. U ovom primjeru razmatramo Bayesovu mrežu koja modelira zavisnosti između oblačnosti (slučajna varijabla C), kiše (R), prskalice (S) i mokre trave (W). U ovom primjeru također pretpostavljamo da već imamo parametre vjerojatnosnih distribucija svih čvorova. Ova mreža prikazana je na sljedećoj slici:



Koristeći paket `pgmpy`, konstruirajte Bayesovu mrežu iz gornjeg primjera. Zatim, koristeći **egzaktno** zaključivanje, postavite sljedeće posteriorne upite: $P(w = 1)$, $P(s = 1|w = 1)$, $P(r = 1|w = 1)$, $P(c = 1|s = 1, r = 1)$ i $P(c = 1)$. Provedite zaključivanje na papiru i uvjerite se da ste ispravno konstruirali mrežu. Pomoći će vam službena dokumentacija te primjeri korištenja (npr. ovaj).

[15]:
```
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete.CPD import TabularCPD
from pgmpy.inference import VariableElimination
```

[16]:
```
# Vaš kôd ovdje...
model = BayesianModel([('Cloudy', 'Sprinkler'), ('Cloudy', 'Rain'),
                       ('Sprinkler', 'Wet Grass'), ('Rain', 'Wet Grass')])
cpd_c = TabularCPD(variable='Cloudy',
                   variable_card=2,
                   values=[[0.5],
                           [0.5]])

cpd_s = TabularCPD(variable='Sprinkler',
                   variable_card=2,
                   values=[[0.5, 0.9],
                           [0.5, 0.1]],
                   evidence=['Cloudy'],
```

```
                    evidence_card=[2])

cpd_r = TabularCPD(variable='Rain',
                    variable_card=2,
                    values=[[0.8, 0.2],
                            [0.2, 0.8]],
                    evidence=['Cloudy'],
                    evidence_card=[2])

cpd_w = TabularCPD(variable='Wet Grass',
                    variable_card=2,
                    values=[[0.99, 0.1, 0.1, 0.00],
                            [0.01, 0.9, 0.9, 1]],
                    evidence=['Sprinkler', 'Rain'],
                    evidence_card=[2, 2])

# print(cpd_c)
# print(cpd_s)
# print(cpd_r)
# print(cpd_w)

model.add_cpds(cpd_c, cpd_s, cpd_r, cpd_w)
model.get_cpds()
print("Is model valid? " + str(model.check_model()) + '\n')

infer  = VariableElimination(model)
print('P(w = 1)')
print(infer.query(['Wet Grass'], show_progress=False))
print()

posterior_s = infer.query(['Sprinkler'], evidence={'Wet Grass': 1},␣
 ↪show_progress=False)
print('P(s = 1|w = 1)')
print(posterior_s)
print()


posterior_r = infer.query(['Rain'], evidence={'Wet Grass': 1},␣
 ↪show_progress=False)
print('P(r = 1|w = 1)')
print(posterior_r)
print()


posterior_c = infer.query(['Cloudy'], evidence={'Sprinkler': 1, 'Rain': 1},␣
 ↪show_progress=False)
print('P(c = 1|s = 1, r = 1)')
```

```
print(posterior_c)
print()


posterior_c2 = infer.query(['Cloudy'], show_progress=False)
print('P(c = 1)')
print(posterior_c2)
```

```
Is model valid? True

P(w = 1)
+--------------+------------------+
| Wet Grass    |   phi(Wet Grass) |
+==============+==================+
| Wet Grass(0) |           0.3491 |
+--------------+------------------+
| Wet Grass(1) |           0.6509 |
+--------------+------------------+

P(s = 1|w = 1)
+--------------+------------------+
| Sprinkler    |   phi(Sprinkler) |
+==============+==================+
| Sprinkler(0) |           0.5714 |
+--------------+------------------+
| Sprinkler(1) |           0.4286 |
+--------------+------------------+

P(r = 1|w = 1)
+---------+-------------+
| Rain    |   phi(Rain) |
+=========+=============+
| Rain(0) |      0.2948 |
+---------+-------------+
| Rain(1) |      0.7052 |
+---------+-------------+

P(c = 1|s = 1, r = 1)
+-----------+---------------+
| Cloudy    |   phi(Cloudy) |
+===========+===============+
| Cloudy(0) |        0.5556 |
+-----------+---------------+
| Cloudy(1) |        0.4444 |
+-----------+---------------+

P(c = 1)
+-----------+---------------+
```

```
| Cloudy    |   phi(Cloudy) |
+===========+===============+
| Cloudy(0) |        0.5000 |
+-----------+---------------+
| Cloudy(1) |        0.5000 |
+-----------+---------------+
```

Q: Koju zajedničku vjerojatnosnu razdiobu ova mreža modelira? Kako tu informaciju očitati iz mreže?
Q: U zadatku koristimo egzaktno zaključivanje. Kako ono radi?
Q: Koja je razlika između posteriornog upita i MAP-upita?
Q: Zašto je vjerojatnost $P(c = 1)$ drugačija od $P(c = 1|s = 1, r = 1)$ ako znamo da čvorovi S i R nisu roditelji čvora C?

(b) Efekt objašnjavanja (engl. *explaining away*) zanimljiv je fenomen u kojem se događa da se dvije varijable "natječu" za objašnjavanje treće. Ovaj fenomen može se primijetiti na gornjoj mreži. U tom se slučaju varijable prskalice (S) i kiše (R) "natječu" za objašnjavanje mokre trave (W). Vaš zadatak je pokazati da se fenomen zaista događa.

[17]:
```
# Vaš kôd ovdje...

sprinkler_on = infer.query(['Sprinkler'], show_progress=False)
print('Vjerojatnost da je prskalica ukljucena.')
print(sprinkler_on)
print()


is_raining = infer.query(['Rain'], show_progress=False)
print('Vjerojatnost da kiša pada.')
print(is_raining)
print()

wet_grass_sprinkler = infer.query(['Sprinkler'], evidence={'Wet Grass': 1},␣
 ↪show_progress=False)
print('Vjerojatnost da je prskalica uključena, ako znamo da je trava mokra.')
print(wet_grass_sprinkler)
print()


wet_grass_raining = infer.query(['Rain'], evidence={'Wet Grass': 1},␣
 ↪show_progress=False)
print('Vjerojatnost da kiša pada, ako znamo da je trava mokra.')
print(wet_grass_raining)
print()


wet_grass_sprinkler_on = infer.query(['Rain'], evidence={'Wet Grass': 1,␣
 ↪'Sprinkler': 1}, show_progress=False)
```

```
print('Kada uočimo da je trava mokra, te da je prskalica uključena, '
      + 'manja je vjerojatnost da kiša pada.')
print(wet_grass_sprinkler_on)
print()
```

```
Vjerojatnost da je prskalica ukljucena.
+--------------+------------------+
| Sprinkler    |   phi(Sprinkler) |
+==============+==================+
| Sprinkler(0) |           0.7000 |
+--------------+------------------+
| Sprinkler(1) |           0.3000 |
+--------------+------------------+

Vjerojatnost da kiša pada.
+---------+-------------+
| Rain    |   phi(Rain) |
+=========+=============+
| Rain(0) |      0.5000 |
+---------+-------------+
| Rain(1) |      0.5000 |
+---------+-------------+

Vjerojatnost da je prskalica uključena, ako znamo da je trava mokra.
+--------------+------------------+
| Sprinkler    |   phi(Sprinkler) |
+==============+==================+
| Sprinkler(0) |           0.5714 |
+--------------+------------------+
| Sprinkler(1) |           0.4286 |
+--------------+------------------+

Vjerojatnost da kiša pada, ako znamo da je trava mokra.
+---------+-------------+
| Rain    |   phi(Rain) |
+=========+=============+
| Rain(0) |      0.2948 |
+---------+-------------+
| Rain(1) |      0.7052 |
+---------+-------------+

Kada uočimo da je trava mokra, te da je prskalica uključena, manja je
vjerojatnost da kiša pada.
+---------+-------------+
| Rain    |   phi(Rain) |
+=========+=============+
| Rain(0) |      0.6774 |
+---------+-------------+
```

```
| Rain(1) |      0.3226 |
+---------+-------------+
```

Q: Kako biste svojim riječima opisali ovaj fenomen, koristeći se ovim primjerom?

0.1.5 3. Grupiranje

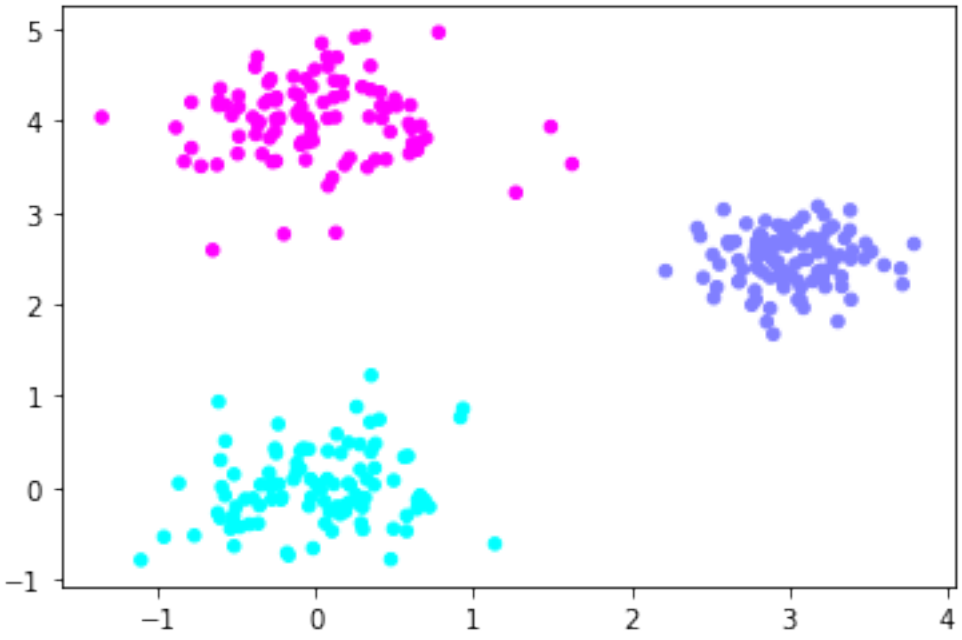
U ovom zadatku ćete se upoznati s algoritmom k-sredina (engl. *k-means*), njegovim glavnim nedostatcima te pretpostavkama. Također ćete isprobati i drugi algoritam grupiranja: model Gaussovih mješavina (engl. *Gaussian mixture model*).

(a) Jedan od nedostataka algoritma k-sredina jest taj što unaprijed zahtjeva broj grupa (K) u koje će grupirati podatke. Ta informacija nam često nije dostupna (kao što nam nisu dostupne ni oznake primjera) te je stoga potrebno nekako izabrati najbolju vrijednost hiperparametra K. Jedan od naivnijih pristupa jest **metoda lakta/koljena** (engl. *elbow method*) koju ćete isprobati u ovom zadatku.

U svojim rješenjima koristite ugrađenu implementaciju algoritma k-sredina, dostupnoj u razredu `cluster.KMeans`.

NB: Kriterijska funkcija algoritma k-sredina još se i naziva **inercijom** (engl. *inertia*). Za naučeni model, vrijednost kriterijske funkcije J dostupna je kroz razredni atribut `inertia_`.

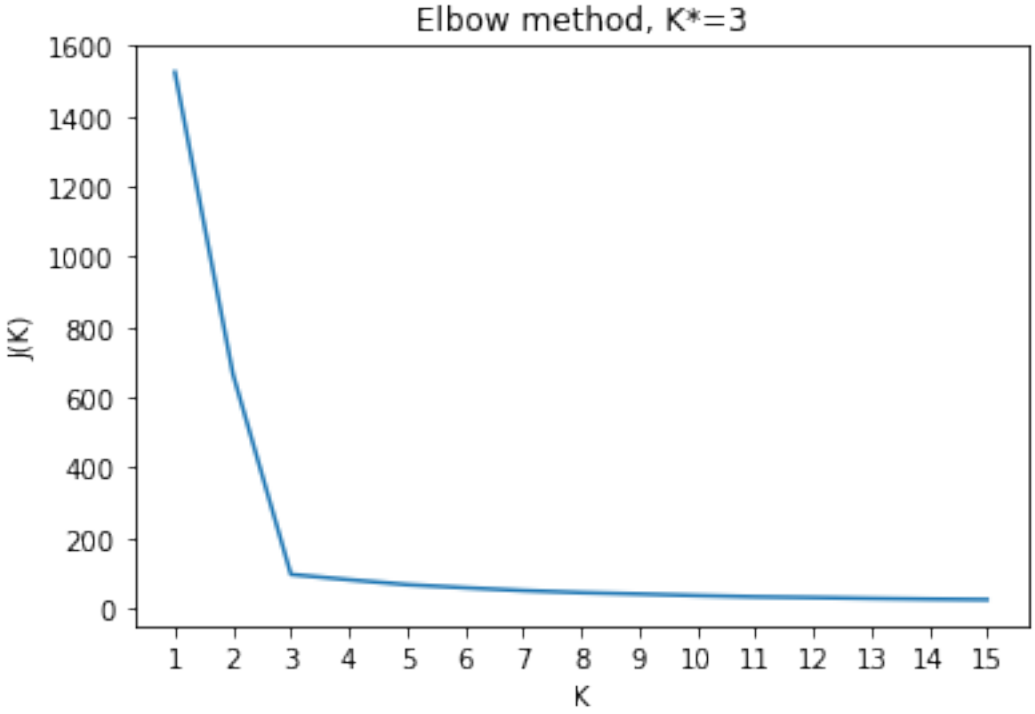
[18]:
```
from sklearn.datasets import make_blobs

Xp, yp = make_blobs(n_samples=300, n_features=2, centers=[[0, 0], [3, 2.5], [0,␣
↪4]],
                    cluster_std=[0.45, 0.3, 0.45], random_state=96)
plt.scatter(Xp[:,0], Xp[:,1], c=yp, cmap=plt.get_cmap("cool"), s=20);
```



Iskoristite skup podataka `Xp` dan gore. Isprobajte vrijednosti hiperparametra K iz $[0, 1, \ldots, 15]$. Ne trebate dirati nikakve hiperparametre modela osim K. Iscrtajte krivulju od J u ovisnosti o broju grupa K. Metodom lakta/koljena odredite vrijednost hiperparametra K.

[19]:
```
from sklearn.cluster import KMeans

# Vaš kôd ovdje...
K = [i for i in range(1, 16)]
print(K)

Js = []
for n_clusters in K:
    kmeans = KMeans(n_clusters).fit(Xp)
    Js.append(kmeans.inertia_)

plt.plot(K, Js)
plt.xticks(K)
plt.xlabel("K")
plt.ylabel("J(K)")
plt.title("Elbow method, K*=3")
plt.show()
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```



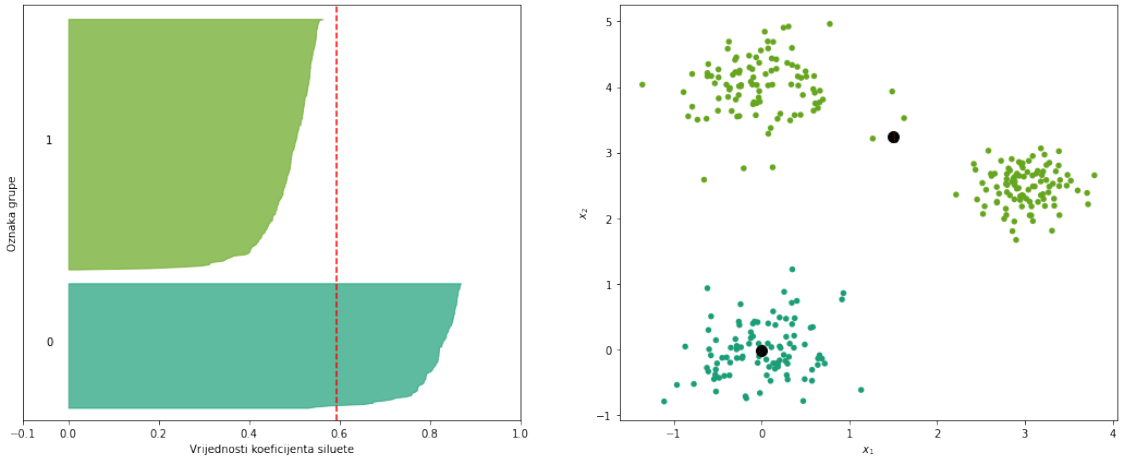
Q: Koju biste vrijednost hiperparametra K izabrali na temelju ovog grafa? Zašto? Je li taj odabir optimalan? Kako to znate?
Q: Je li ova metoda robusna?
Q: Možemo li izabrati onaj K koji minimizira pogrešku J? Objasnite.

(b) Odabir vrijednosti hiperparametra K može se obaviti na mnoštvo načina. Pored metode lakta/koljena, moguće je isto ostvariti i analizom siluete (engl. *silhouette analysis*). Za to smo pripremili funkciju `plot_silhouette` koja za dani broj grupa i podatke iscrtava prosječnu vrijednost koeficijenta siluete i vrijednost koeficijenta svakog primjera (kroz grupe).

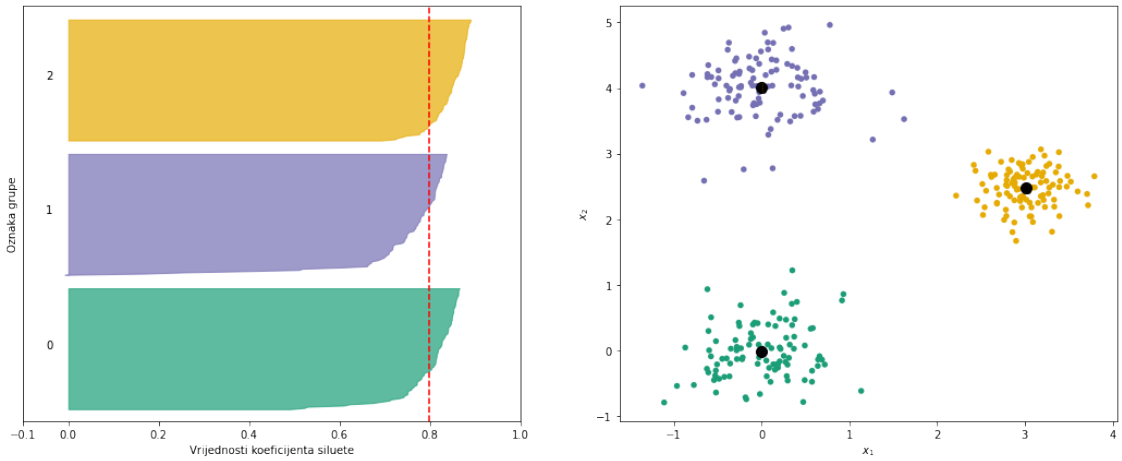
Vaš je zadatak isprobati različite vrijednosti hiperparametra K, $K \in \{2, 3, 5\}$ i na temelju dobivenih grafova odlučiti se za optimalan K.

[20]:
```
# Vaš kôd ovdje...
K = [2, 3, 5]

for n_clusters in K:
    plot_silhouette(n_clusters, Xp)

plt.show()
```

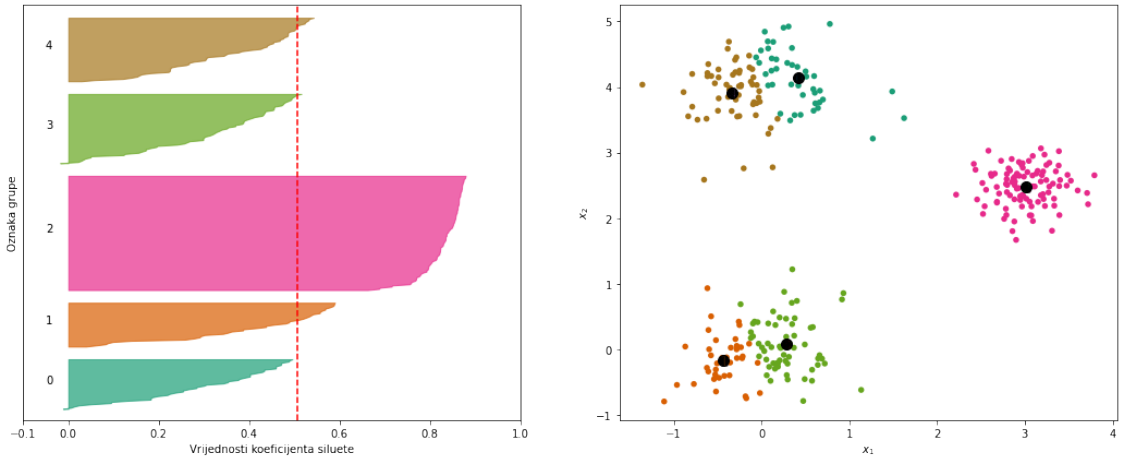
```
For n_clusters = 2 The average silhouette_score is : 0.59280796169837
```




```
For n_clusters = 3 The average silhouette_score is : 0.7975462212061406
```




```
For n_clusters = 5 The average silhouette_score is : 0.5059026123508755
```



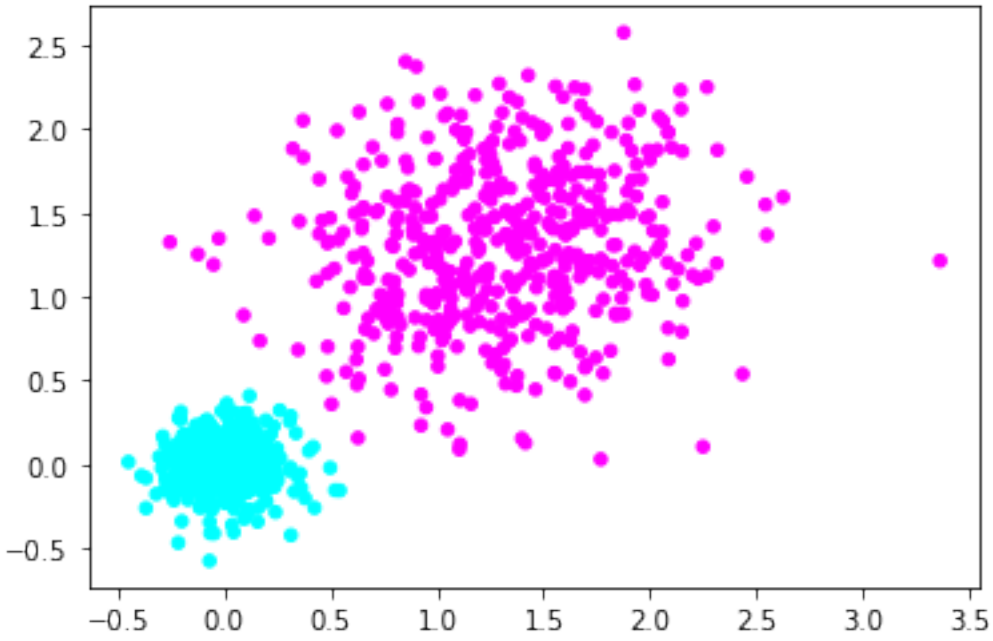
Q: Kako biste se gledajući ove slike odlučili za K?
Q: Koji su problemi ovog pristupa?

(c) U ovom i sljedećim podzadatcima fokusirat ćemo se na temeljne pretpostavke algoritma k-sredina te što se događa ako te pretpostavke nisu zadovoljene. Dodatno, isprobat ćemo i grupiranje modelom Gaussovih mješavina (engl. *Gaussian Mixture Models*; GMM) koji ne nema neke od tih pretpostavki.

Prvo, krenite od podataka `X1`, koji su generirani korištenjem funkcije `datasets.make_blobs`, koja stvara grupe podataka pomoću izotropskih Gaussovih distribucija.

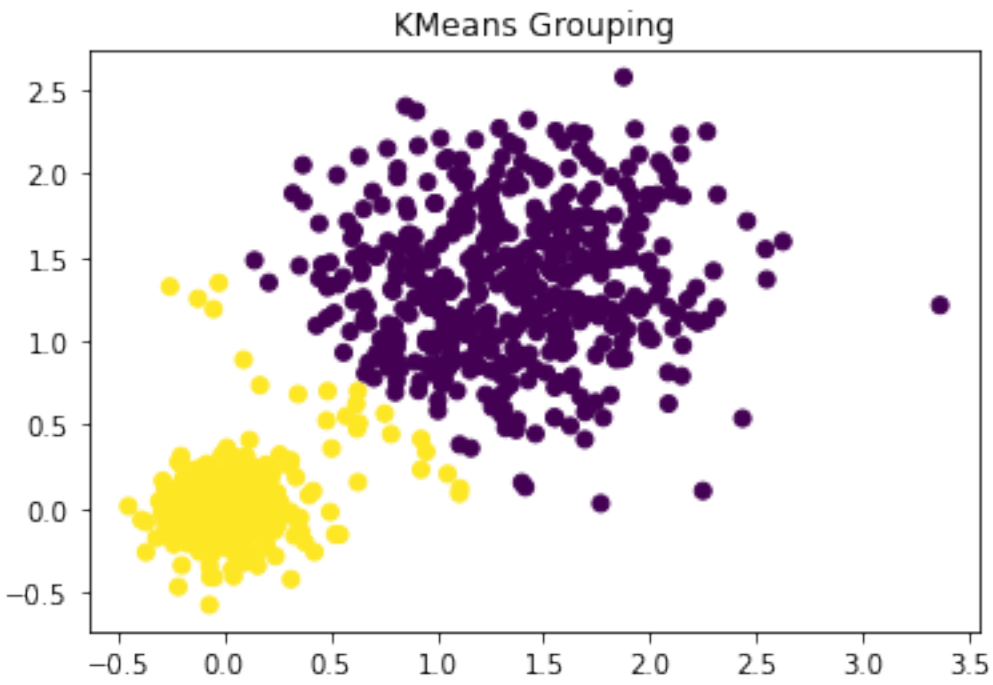
[21]:
```
from sklearn.datasets import make_blobs
```

```
X1, y1 = make_blobs(n_samples=1000, n_features=2, centers=[[0, 0], [1.3, 1.3]],␣
↪cluster_std=[0.15, 0.5], random_state=96)
plt.scatter(X1[:,0], X1[:,1], c=y1, cmap=plt.get_cmap("cool"), s=20);
```



Naučite model k-sredina (idealno pretpostavljajući $K = 2$) na gornjim podatcima i prikažite dobiveno grupiranje (proučite funkciju `scatter`, posebice argument `c`).

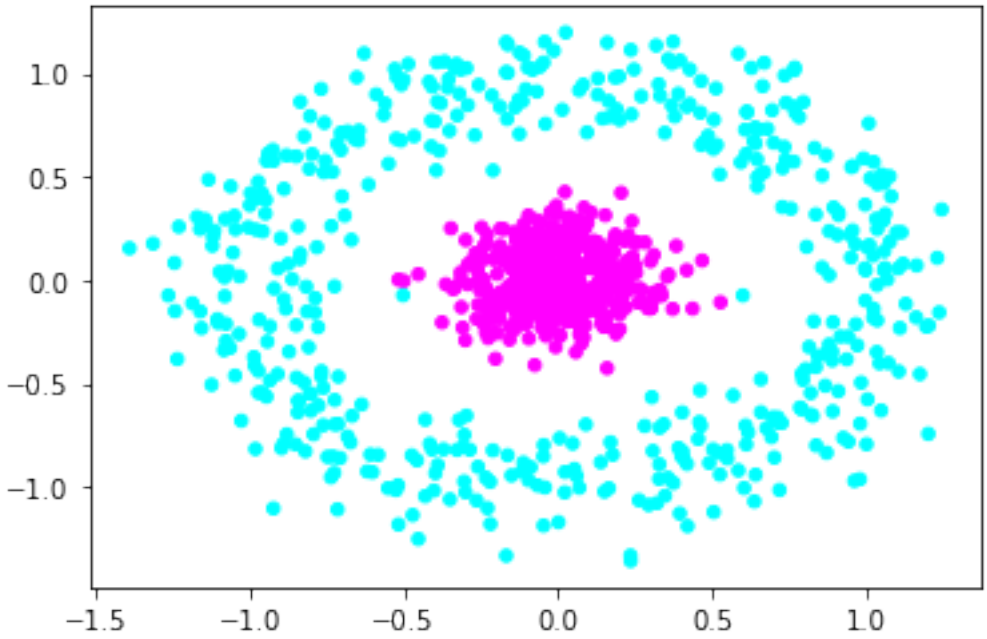
[22]:
```
# Vaš kôd ovdje...
K = 2

feature_col_1 = X1[:,0]
feature_col_2 = X1[:,1]

kmeans = KMeans(K).fit(X1)
plt.scatter(feature_col_1, feature_col_2, c=kmeans.predict(X1));
plt.title("KMeans Grouping")
plt.show()
```



Q: Što se dogodilo? Koja je pretpostavka algoritma k-sredina ovdje narušena?
Q: Što biste morali osigurati kako bi algoritam pronašao ispravne grupe?

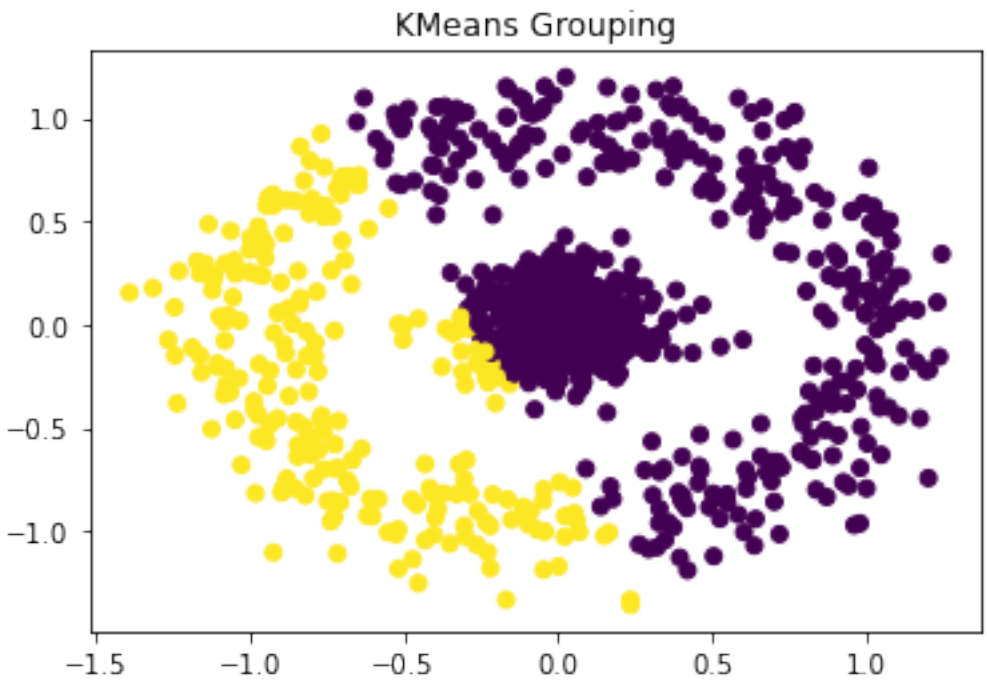
(d) Isprobajte algoritam k-sredina na podatcima generiranim korištenjem funkcije `datasets.make_circles`, koja stvara dvije grupe podataka tako da je jedna unutar druge.

[23]:
```
from sklearn.datasets import make_circles

X2, y2 = make_circles(n_samples=1000, noise=0.15, factor=0.05, random_state=96)
plt.scatter(X2[:,0], X2[:,1], c=y2, cmap=plt.get_cmap("cool"), s=20);
```



Ponovno, naučite model k-sredina (idealno pretpostavljajući $K = 2$) na gornjim podatcima i prikažite dobiveno grupiranje (proučite funkciju `scatter`, posebice argument `c`).

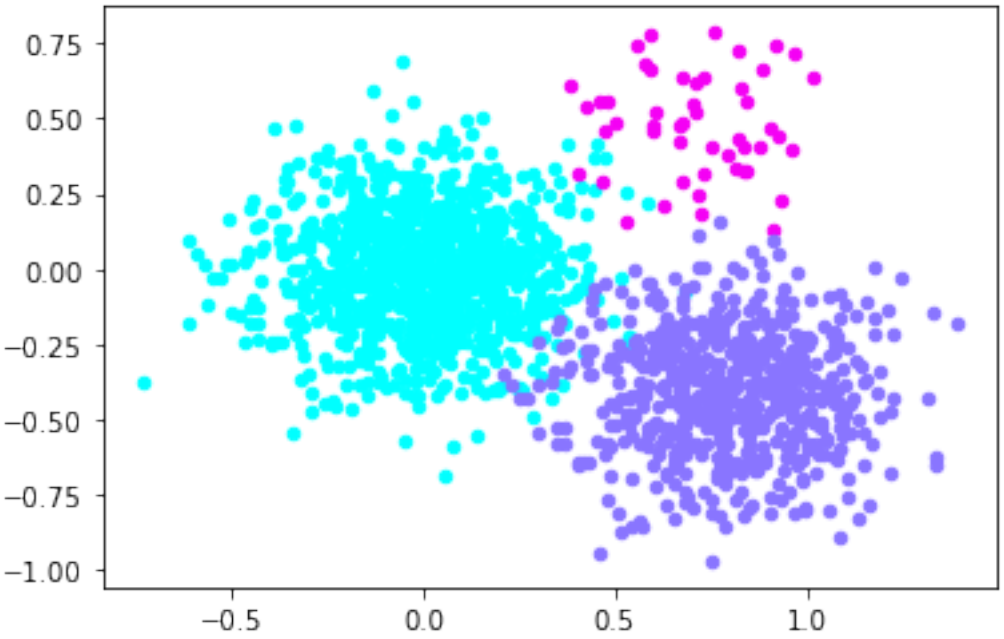
[24]:
```
# Vaš kôd ovdje...
K = 2

feature_col_1 = X2[:,0]
feature_col_2 = X2[:,1]

kmeans = KMeans(K).fit(X2)
plt.scatter(feature_col_1, feature_col_2, c=kmeans.predict(X2));
plt.title("KMeans Grouping")
plt.show()
```



Q: Što se dogodilo? Koja je pretpostavka algoritma k-sredina ovdje narušena?
Q: Što biste morali osigurati kako bi algoritam pronašao ispravne grupe?

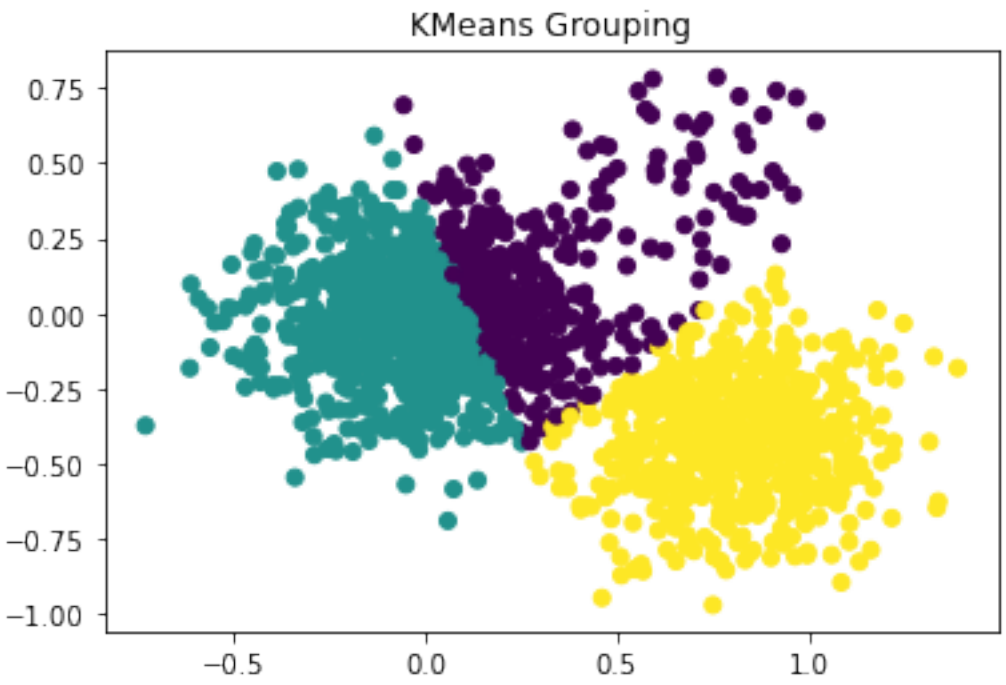
(e) Završno, isprobat ćemo algoritam na sljedećem umjetno stvorenom skupu podataka:

[25]:
```
X31, y31 = make_blobs(n_samples=1000, n_features=2, centers=[[0, 0]],␣
↪cluster_std=[0.2], random_state=69)
X32, y32 = make_blobs(n_samples=50, n_features=2, centers=[[0.7, 0.5]],␣
↪cluster_std=[0.15], random_state=69)
X33, y33 = make_blobs(n_samples=600, n_features=2, centers=[[0.8, -0.4]],␣
↪cluster_std=[0.2], random_state=69)
plt.scatter(X31[:,0], X31[:,1], c="#00FFFF", s=20)
plt.scatter(X32[:,0], X32[:,1], c="#F400F4", s=20)
plt.scatter(X33[:,0], X33[:,1], c="#8975FF", s=20)

# Just join all the groups in a single X.
X3 = np.vstack([X31, X32, X33])
y3 = np.hstack([y31, y32, y33])
```



Ponovno, naučite model k-sredina (ovaj put idealno pretpostavljajući $K = 3$) na gornjim podatcima i prikažite dobiveno grupiranje (proučite funkciju `scatter`, posebice argument `c`).

[26]:
```
# Vaš kôd ovdje...
K = 3

feature_col_1 = X3[:,0]
feature_col_2 = X3[:,1]

kmeans = KMeans(K).fit(X3)
plt.scatter(feature_col_1, feature_col_2, c=kmeans.predict(X3));
plt.title("KMeans Grouping")
plt.show()
```



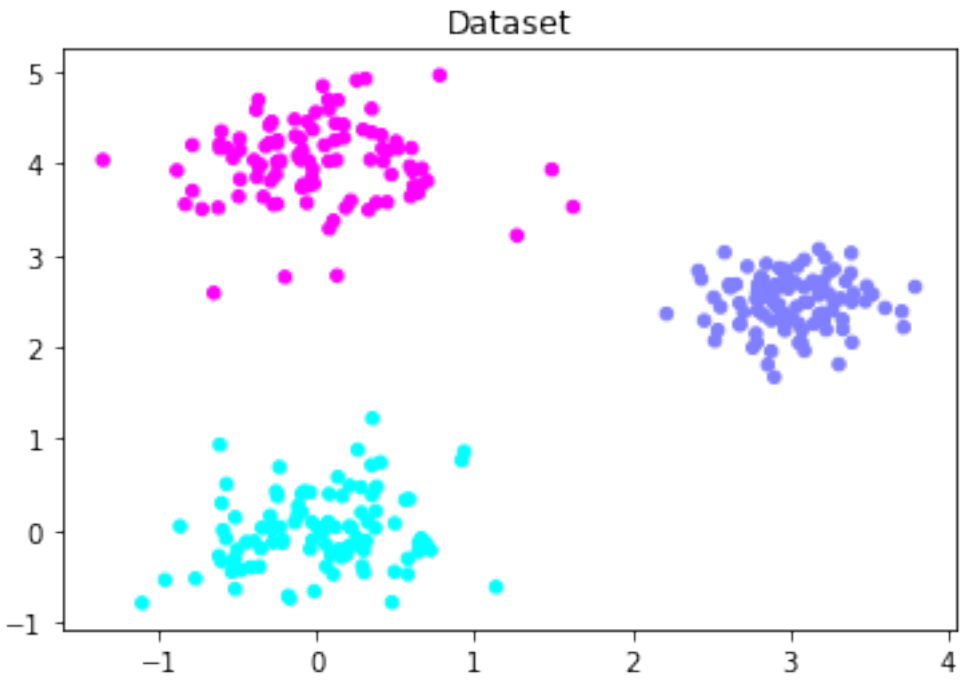
Q: Što se dogodilo? Koja je pretpostavka algoritma k-sredina ovdje narušena?
Q: Što biste morali osigurati kako bi algoritam pronašao ispravne grupe?

(f) Sada kada ste se upoznali s ograničenjima algoritma k-sredina, isprobat ćete grupiranje modelom mješavine Gaussa (*Gaussian Mixture Models; GMM*), koji je generalizacija algoritma k-sredina (odnosno, algoritam k-sredina specijalizacija je GMM-a). Implementacija ovog modela dostupna je u `mixture.GaussianMixture`. Isprobajte ovaj model (s istim pretpostavkama o broju grupa) na podacima iz podzadataka (c)-(e). Ne morate mijenjati nikakve hiperparametre ni postavke osim broja komponenti.

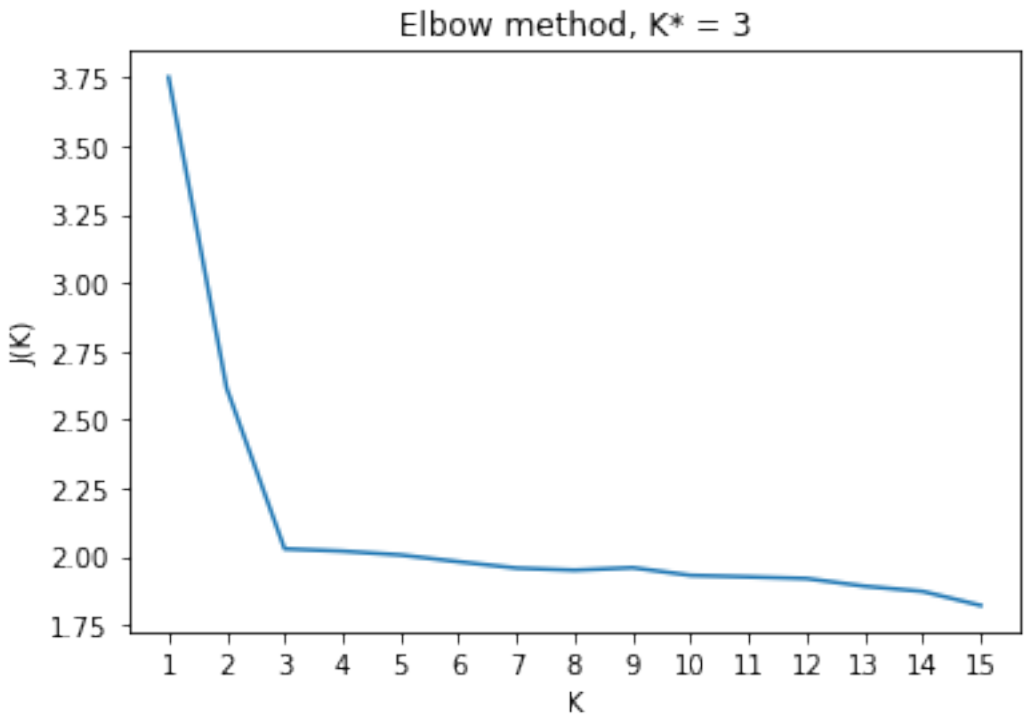
[27]:
```
from sklearn.mixture import GaussianMixture
```

[28]:
```
# Vaš kôd ovdje...
from sklearn.datasets import make_blobs

Xp, yp = make_blobs(n_samples=300, n_features=2, centers=[[0, 0], [3, 2.5], [0,␣
↪4]],
                    cluster_std=[0.45, 0.3, 0.45], random_state=96)
plt.scatter(Xp[:,0], Xp[:,1], c=yp, cmap=plt.get_cmap("cool"), s=20);
plt.title("Dataset")
plt.show()

K = [i for i in range(1, 16)]
print(K)
```

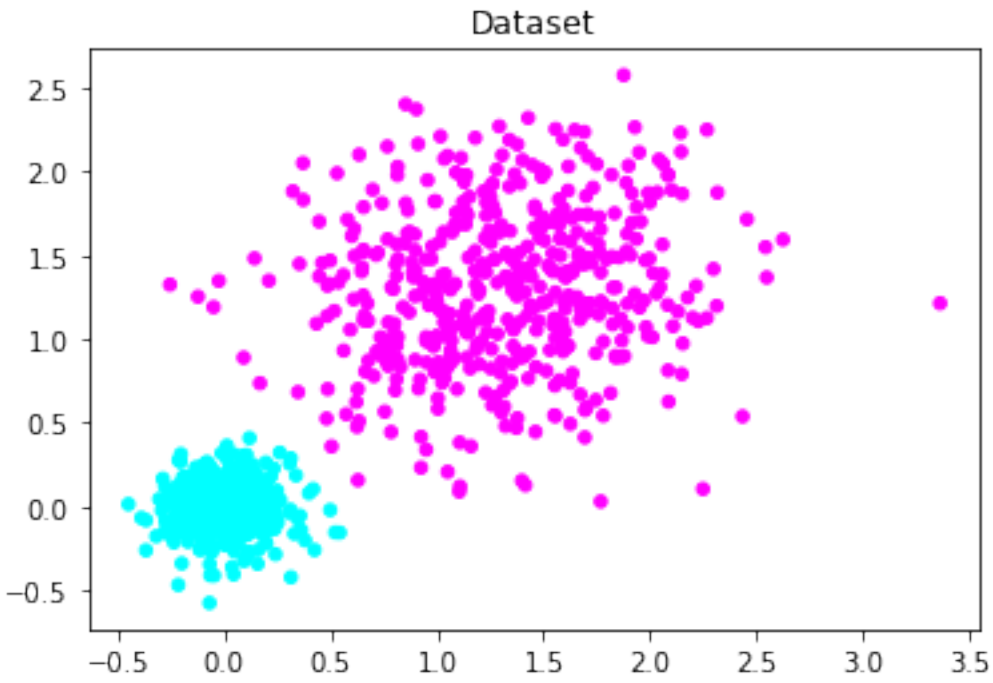
```
Js = []
for n_clusters in K:
    mixture = GaussianMixture(n_clusters).fit(Xp)
    Js.append(np.abs(mixture.lower_bound_))

plt.plot(K, Js)
plt.xticks(K)
plt.xlabel("K")
plt.ylabel("J(K)")
plt.title("Elbow method, K* = 3")
plt.show()
```

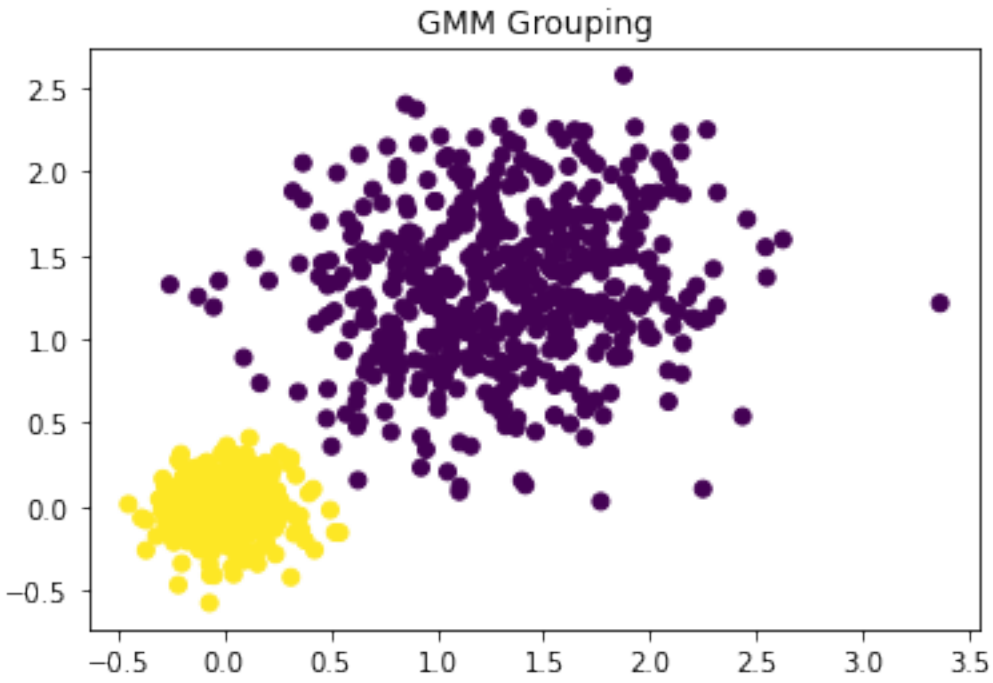



```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

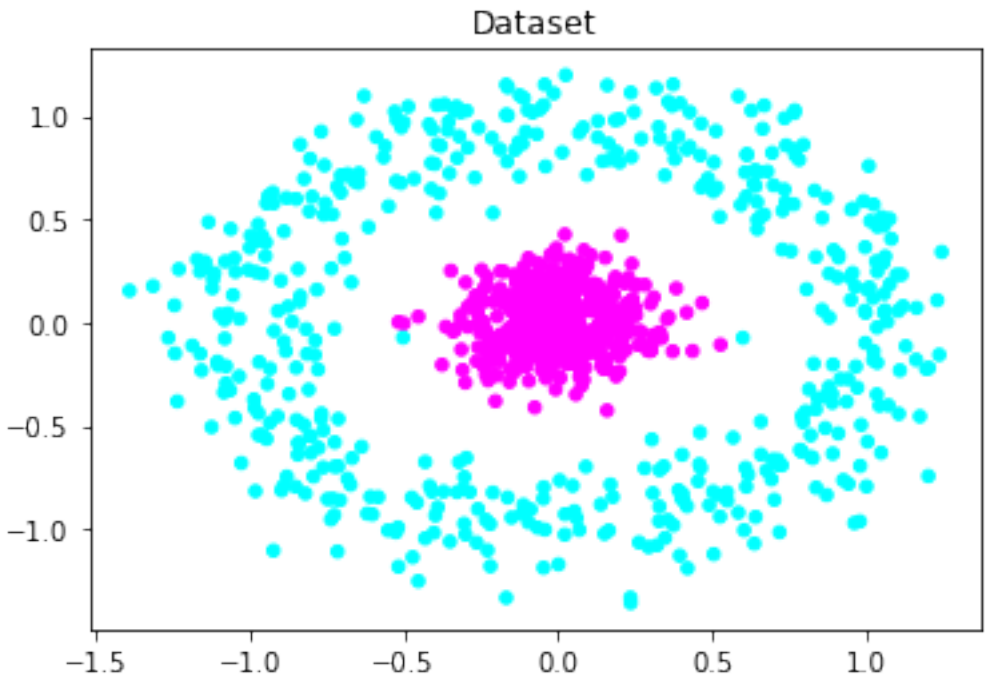


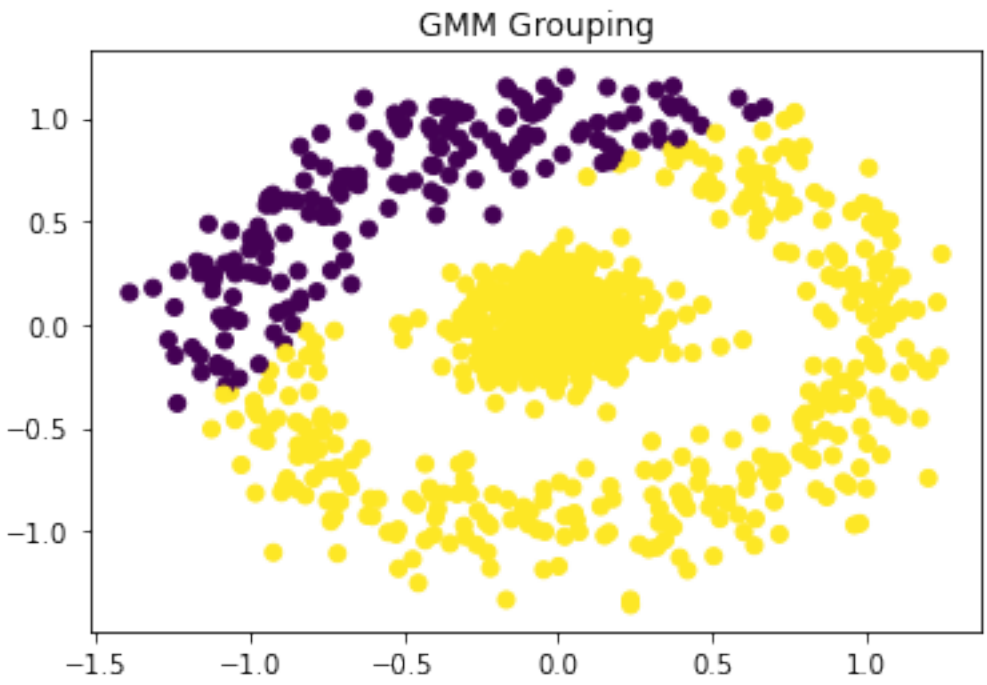
[29]:
```
from sklearn.datasets import make_blobs

X1, y1 = make_blobs(n_samples=1000, n_features=2, centers=[[0, 0], [1.3, 1.3]],␣
 ↪cluster_std=[0.15, 0.5], random_state=96)
plt.scatter(X1[:,0], X1[:,1], c=y1, cmap=plt.get_cmap("cool"), s=20);
plt.title("Dataset")
plt.show()

K = 2

feature_col_1 = X1[:,0]
feature_col_2 = X1[:,1]

mixture = GaussianMixture(K).fit(X1)
plt.scatter(feature_col_1, feature_col_2, c=mixture.predict(X1));
plt.title("GMM Grouping")
plt.show()
```


Dataset
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5


GMM Grouping
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5

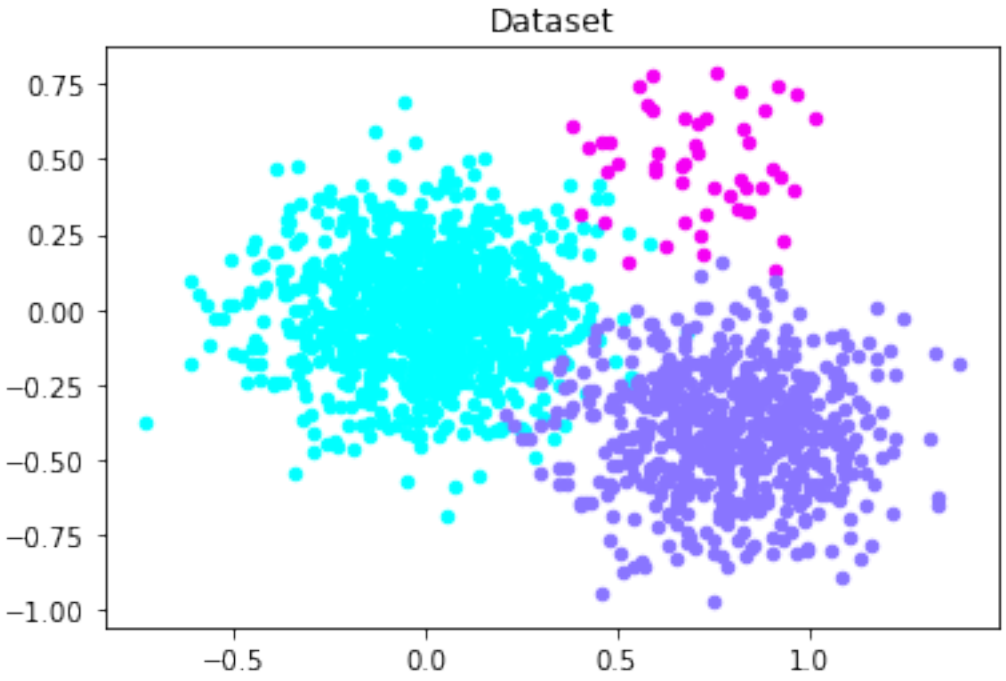
[30]:
```
from sklearn.datasets import make_circles

X2, y2 = make_circles(n_samples=1000, noise=0.15, factor=0.05, random_state=96)
plt.scatter(X2[:,0], X2[:,1], c=y2, cmap=plt.get_cmap("cool"), s=20);
plt.title("Dataset")
plt.show()

K = 2

feature_col_1 = X2[:,0]
feature_col_2 = X2[:,1]

mixture = GaussianMixture(K).fit(X2)
plt.scatter(feature_col_1, feature_col_2, c=mixture.predict(X2));
plt.title("GMM Grouping")
plt.show()
```

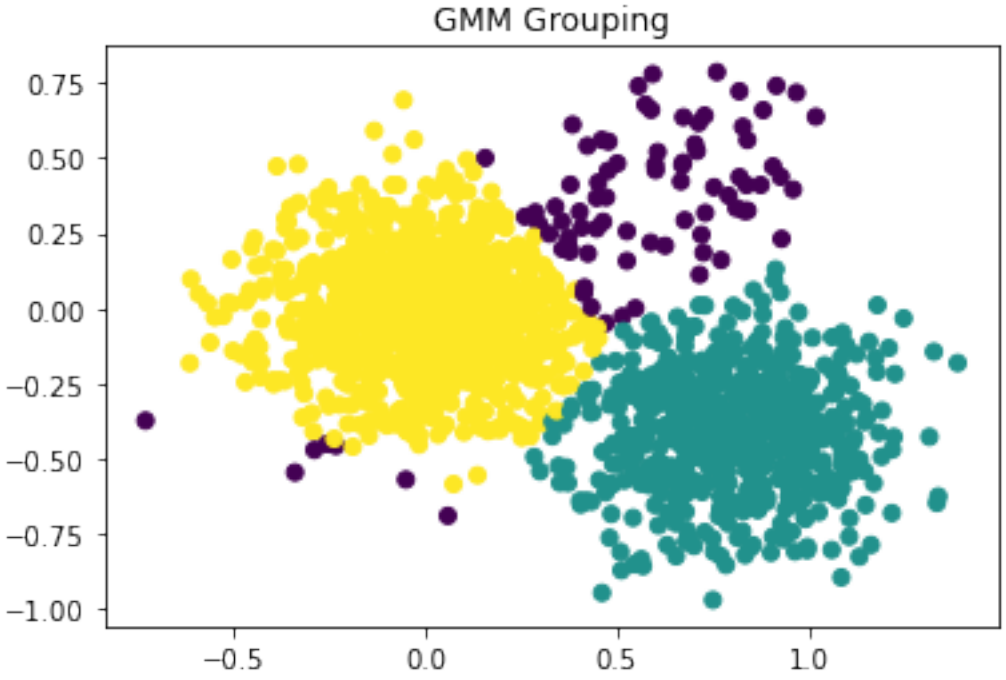




[31]:
```
X31, y31 = make_blobs(n_samples=1000, n_features=2, centers=[[0, 0]],␣
 ↪cluster_std=[0.2], random_state=69)
X32, y32 = make_blobs(n_samples=50, n_features=2, centers=[[0.7, 0.5]],␣
 ↪cluster_std=[0.15], random_state=69)
X33, y33 = make_blobs(n_samples=600, n_features=2, centers=[[0.8, -0.4]],␣
 ↪cluster_std=[0.2], random_state=69)
plt.scatter(X31[:,0], X31[:,1], c="#00FFFF", s=20)
plt.scatter(X32[:,0], X32[:,1], c="#F400F4", s=20)
plt.scatter(X33[:,0], X33[:,1], c="#8975FF", s=20)

# Just join all the groups in a single X.
X3 = np.vstack([X31, X32, X33])
y3 = np.hstack([y31, y32, y33])

plt.title("Dataset")
plt.show()

K = 3

feature_col_1 = X3[:,0]
feature_col_2 = X3[:,1]

mixture = GaussianMixture(K).fit(X3)
plt.scatter(feature_col_1, feature_col_2, c=mixture.predict(X3));
```

```
plt.title("GMM Grouping")
plt.show()
```





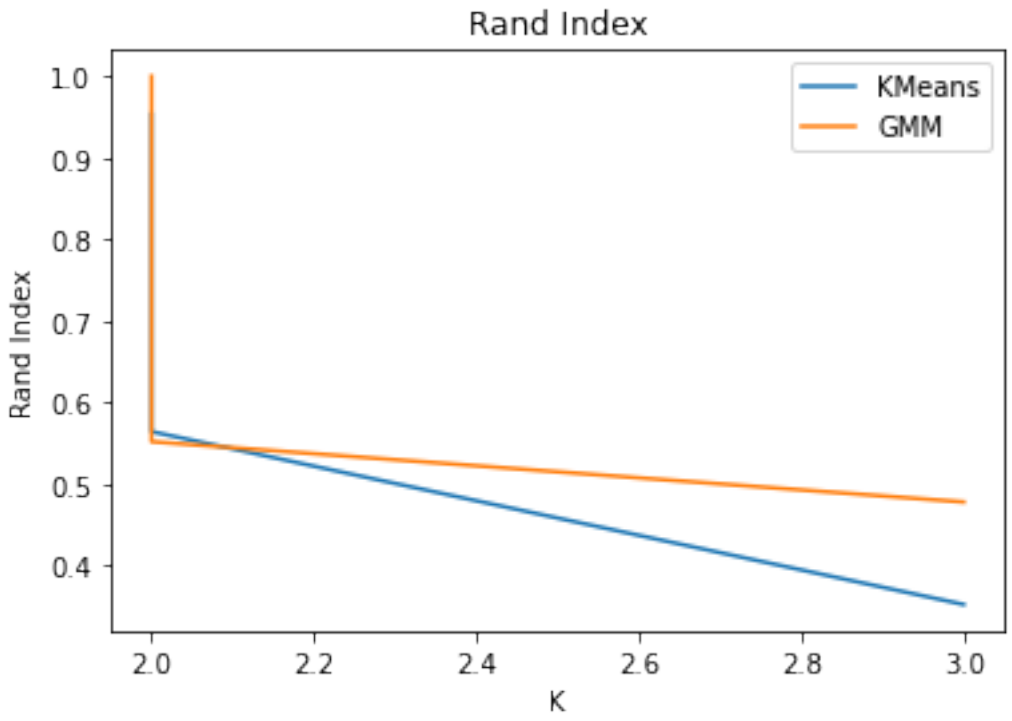
(g) Kako vrednovati točnost modela grupiranja ako imamo stvarne oznake svih primjera (a u našem slučaju imamo, jer smo mi ti koji smo generirali podatke)? Često korištena mjera jest **Randov indeks** koji je zapravo pandan točnosti u zadatcima klasifikacije. Implementirajte funkciju `rand_index_score(y_gold, y_predict)` koja ga računa. Funkcija prima dva argumenta: listu stvarnih grupa kojima primjeri pripadaju (`y_gold`) i listu predviđenih grupa (`y_predict`). Dobro će vam doći funkcija `itertools.combinations`.

[32]:
```
import itertools as it

def rand_index_score(y_gold, y_predict):
    # Vaš kôd ovdje...
    N = len(y_gold)
    combinations_of_two = list(it.combinations(range(N), 2))

    N = len(combinations_of_two)
    count = 0
    for _1, _2 in combinations_of_two:
        truly_belongs_to_group = y_gold[_1] == y_gold[_2]
        predicted_belonging_to_group = y_predict[_1] == y_predict[_2]
        correctly_classified = truly_belongs_to_group ==␣
↪predicted_belonging_to_group



        if correctly_classified:
            count = count + 1

    return count / N
```

[33]:
```
K = [2, 2, 3]
X = [X1, X2, X3]
y = [y1, y2, y3]


dataset = zip(X, y, K)

k_means = []
m_ixture = []
for idx, (X, y, k) in enumerate(dataset):
    print(f'k={k}, X=X{idx+1}, y=y{idx+1}')
    kmeans = KMeans(k, random_state=1337).fit(X)
    gmm = GaussianMixture(k, random_state=1337).fit(X)
    rand_kmeans = rand_index_score(y, kmeans.predict(X))
    rand_gmm = rand_index_score(y, gmm.predict(X))

    k_means.append(rand_kmeans)
    m_ixture.append(rand_gmm)
    print(f'Rand(KMeans)={rand_kmeans}')
```

```
    print(f'Rand(GMM)={rand_gmm}')
    print()


plt.plot(K, k_means, label="KMeans")
plt.plot(K, m_ixture, label="GMM")
plt.xlabel("K")
plt.ylabel("Rand Index")
plt.legend()
plt.title("Rand Index")
plt.show()
```

```
k=2, X=X1, y=y1
Rand(KMeans)=0.9531051051051052
Rand(GMM)=1.0

k=2, X=X2, y=y2
Rand(KMeans)=0.5643643643643643
Rand(GMM)=0.55204004004004

k=3, X=X3, y=y3
Rand(KMeans)=0.3524457430582355
Rand(GMM)=0.4780667806016502
```



Q: Zašto je Randov indeks pandan točnosti u klasifikacijskim problemima?
Q: Koji su glavni problemi ove metrike?
Q: Kako vrednovati kvalitetu grupiranja ako nenamo stvarne oznake primjera? Je li to uopće moguće?