



- **Peguy Rusty Kana
Donwoung**
- **Nkwanga Nkwanga Mansvell**
- **Atoundem Sonfack Auguste**
- **Nya Ken Ulrich**

Aufgabe 1:

Ablauf:

- Link zur Dokumentation:

https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/

- **Geplanter Ablauf:** „Wir werden die Webseite über den obigen Link öffnen und durch das Beispiel führen, um die Code-Details und Konzepte zu erläutern.“

Aufgabe2:

a)

Importieren der Pandas-Bibliothek

```
[ ]: import pandas as pd
```



b)

▼ Laden des Datensatzes

```
[122]: df = pd.read_csv(r"C:\Users\peguy\Downloads\shopping.csv")
```

c) Berechnen der relativen Häufigkeit der Produkte

```
59]: # Berechnung der Gesamtanzahl der Transaktionen im DataFrame
total_transactions = len(df)

# Zählen der Vorkommen jedes Produkts in allen Spalten
product_counts = df.stack().value_counts()

# Berechnung der relativen Häufigkeit jedes Produkts in Prozent
relative_frequency = (product_counts / total_transactions) * 100

# Erstellen eines DataFrames aus den Vorkommen und relativen Häufigkeiten
df2 = pd.DataFrame({
    'Occurrences': product_counts, # Hinzufügen der Vorkommen
    'Relative Frequency (%)': relative_frequency # Hinzufügen der relativen Häufigkeit
}).reset_index() # Zurücksetzen des Index, um ihn zu einer Spalte zu machen

# Umbenennen der Spalten im DataFrame zur besseren Klarheit
df2.columns = ['Product', 'Occurrences', 'Relative Frequency']

#Anzeige der DataFrame
df2
```

Ergebnis:

[69]:

	Product	Occurences	Relative Frequency
0	whole milk	2513	25.551601
1	other vegetables	1903	19.349263
2	rolls/buns	1809	18.393493
3	soda	1715	17.437722
4	yogurt	1372	13.950178
...
164	bags	4	0.040671
165	kitchen utensil	4	0.040671
166	preservation products	2	0.020336
167	baby food	1	0.010168
168	sound storage medium	1	0.010168

169 rows × 3 columns

d)

▾ Kombinationen von Produkten am Häufigsten gekauft (Warenkörbe)

```
] : # Importiert den TransactionEncoder, um Transaktionsdaten zu transformieren.
    from mlxtend.preprocessing import TransactionEncoder
    # Importiert die apriori-Funktion, um häufige Artikelsets zu extrahieren.
    from mlxtend.frequent_patterns import apriori

    # Konvertiert den DataFrame in eine Liste von Transaktionen, indem NaN-Werte entfernt werden.
    transactions = df.apply(lambda x: x.dropna().tolist(), axis=1).tolist()

    # Initialisiert den TransactionEncoder, um eine binäre Darstellung zu erstellen.
    te = TransactionEncoder()

    # Wandelt die Transaktionen in eine binäre Matrix um.
    te_ary = te.fit(transactions).transform(transactions)

    # Erstellt einen DataFrame aus der binären Matrix, wobei die Spalten nach den Artikeln benannt werden.
    df = pd.DataFrame(te_ary, columns=te.columns_)

    # Wendet den Apriori-Algorithmus an, um häufige Artikelsets mit einem minimalen Support von 1% zu extrahieren.
    frequent_itemsets = apriori(df, min_support=0.01, use_colnames=True)

    # Filtert die häufigen Artikelsets, um nur die Sets mit mehr als einem Artikel zu behalten.
    frequent_itemsets = frequent_itemsets[frequent_itemsets['itemsets'].apply(len) > 1]
```

Ergebnis:

```
] : frequent_itemsets
```

```
] :
```

	support	itemsets
88	0.019725	(beef, other vegetables)
89	0.013625	(beef, rolls/buns)
90	0.017387	(beef, root vegetables)
91	0.021251	(beef, whole milk)
92	0.011693	(beef, yogurt)
...
328	0.011998	(tropical fruit, root vegetables, whole milk)
329	0.014540	(yogurt, root vegetables, whole milk)
330	0.010473	(yogurt, whole milk, soda)
331	0.015150	(tropical fruit, yogurt, whole milk)
332	0.010880	(yogurt, whipped/sour cream, whole milk)

245 rows × 2 columns

Aufgabe3:

a)

```
import pandas as pd

# Laden Sie den Datensatz vehicles.csv
df = pd.read_csv('vehicles.csv')

# Filtern der Fahrzeuge mit Front-, Heck- oder Allradantrieb
filtered_df = df[df['drive'].isin(['Rear-wheel Drive', 'Front-wheel Drive', 'All-wheel Drive'])]

# Ausgabe der gefilterten Daten
print(filtered_df)
```

b) Ergebnis

	barrels08	barrelsA08	charge120	charge240	city08	city08U	cityA08	\
0	14.167143	0.0	0.0	0.0	19	0.0	0	
1	27.046364	0.0	0.0	0.0	9	0.0	0	
2	11.018889	0.0	0.0	0.0	23	0.0	0	
3	27.046364	0.0	0.0	0.0	10	0.0	0	
5	13.523182	0.0	0.0	0.0	21	0.0	0	
...	
47515	12.396250	0.0	0.0	0.0	21	0.0	0	
47516	10.625357	0.0	0.0	0.0	24	0.0	0	
47517	11.900400	0.0	0.0	0.0	21	0.0	0	
47518	13.523182	0.0	0.0	0.0	19	0.0	0	
47519	12.935217	0.0	0.0	0.0	20	0.0	0	

	cityA08U	cityCD	cityE	...	mfrCode	c240Dscr	charge240b	c240bDscr	\
0	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
1	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
2	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
3	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
5	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
...	
47515	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
47516	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
47517	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
47518	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
47519	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	

	createdOn					modifiedOn					startStop	\
0	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
1	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
2	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
3	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
5	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
...	
47515	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
47516	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
47517	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
47518	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	
47519	Tue	Jan	01	00:00:00	EST 2013	Tue	Jan	01	00:00:00	EST 2013	NaN	

b)

```
# Filtern der Fahrzeuge nach Kraftstoffart
filtered_fuel_df = df[
    df['fuelType'].str.contains('diesel|cng|electricity', case=False, na=False) |
    df['fuelType1'].str.contains('diesel|natural gas|electricity', case=False, na=False) |
    df['fuelType2'].str.contains('natural gas|electricity', case=False, na=False)
]

# Ausgabe der gefilterten Daten
print(filtered_fuel_df.head())
print(f"Anzahl der gefilterten Fahrzeuge: {filtered_fuel_df.shape[0]}")
```

Ergebnis:

	barrels08	barrelsA08	charge120	charge240	city08	city08U	cityA08	\
141	18.799737	0.0	0.0	0.0	18	0.0	0	
213	19.844167	0.0	0.0	0.0	15	0.0	0	
225	21.011471	0.0	0.0	0.0	16	0.0	0	
372	21.011471	0.0	0.0	0.0	15	0.0	0	
382	21.011471	0.0	0.0	0.0	15	0.0	0	

	cityA08U	cityCD	cityE	...	mfrCode	c240Dscr	charge240b	c240bDscr	\
141	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
213	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
225	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
372	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
382	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	

	createdOn						modifiedOn						startStop	\
141	Tue	Jan	01	00:00:00	EST	2013	Tue	Jan	01	00:00:00	EST	2013	NaN	
213	Tue	Jan	01	00:00:00	EST	2013	Tue	Jan	01	00:00:00	EST	2013	NaN	
225	Tue	Jan	01	00:00:00	EST	2013	Tue	Jan	01	00:00:00	EST	2013	NaN	
372	Tue	Jan	01	00:00:00	EST	2013	Tue	Jan	01	00:00:00	EST	2013	NaN	
382	Tue	Jan	01	00:00:00	EST	2013	Tue	Jan	01	00:00:00	EST	2013	NaN	

	phevCity	phevHwy	phevComb
141	0	0	0
213	0	0	0
225	0	0	0
372	0	0	0
382	0	0	0

c)

```
# Filtern der Fahrzeuge, die keinen CVT verwenden
filtered_no_cvt_df = df[~df['trany'].str.contains('Automatic \ (variable gear ratios\)', case=False, na=False)]

# Ausgabe der gefilterten Daten
print(filtered_no_cvt_df.head())
print(f"Anzahl der gefilterten Fahrzeuge ohne stufenloses Getriebe: {filtered_no_cvt_df.shape[0]}")
```

Ergebnis

	barrels08	barrelsA08	charge120	charge240	city08	city08U	cityA08	\
0	14.167143	0.0	0.0	0.0	19	0.0	0	
1	27.046364	0.0	0.0	0.0	9	0.0	0	
2	11.018889	0.0	0.0	0.0	23	0.0	0	
3	27.046364	0.0	0.0	0.0	10	0.0	0	
4	15.658421	0.0	0.0	0.0	17	0.0	0	

	cityA08U	cityCD	cityE	...	mfrCode	c240Dscr	charge240b	c240bDscr	\
0	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
1	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
2	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
3	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	
4	0.0	0.0	0.0	...	NaN	NaN	0.0	NaN	

	createdOn				modifiedOn				startStop	\
0	Tue	Jan	01	00:00:00 EST 2013	Tue	Jan	01	00:00:00 EST 2013	NaN	
1	Tue	Jan	01	00:00:00 EST 2013	Tue	Jan	01	00:00:00 EST 2013	NaN	
2	Tue	Jan	01	00:00:00 EST 2013	Tue	Jan	01	00:00:00 EST 2013	NaN	
3	Tue	Jan	01	00:00:00 EST 2013	Tue	Jan	01	00:00:00 EST 2013	NaN	
4	Tue	Jan	01	00:00:00 EST 2013	Tue	Jan	01	00:00:00 EST 2013	NaN	

	phevCity	phevHwy	phevComb
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0

d) Extrahierung der Anzahl der Gänge:

```
: import pandas as pd

# Angenommen, filtered_no_cvt_df ist dein gefilterter Datensatz ohne stufenloses Getriebe
# filtered_no_cvt_df = pd.read_csv('vehicles.csv', low_memory=False) # Setze low_memory=False für gemischte Datentypen

# Funktion zur Extraktion der Anzahl der Gänge und der Getriebeart
def extract_transmission_info(transmission):
    if pd.isna(transmission):
        return None, None # Umgang mit NaN-Werten
    elif 'Manual' in transmission:
        # Extrahieren der Anzahl der Gänge (z.B. "Manual 5-spd" => 5)
        try:
            gears = int(transmission.split()[1].replace('-spd', ''))
            return gears, 'Manual'
        except ValueError:
            return None, 'Manual' # Fehlerbehandlung
    elif 'Automatic' in transmission:
        # Extrahieren der Anzahl der Gänge (z.B. "Automatic 6-spd" => 6)
        try:
            gears = int(transmission.split()[1].replace('-spd', ''))
            return gears, 'Automatic'
        except ValueError:
            return None, 'Automatic' # Fehlerbehandlung
    return None, None # Falls keine passende Übertragung gefunden wird

# Anwendung der Funktion auf die 'trany'-Spalte
df[['Gears', 'Transmission Type']] = df['trany'].apply(extract_transmission_info).apply(pd.Series)

# Zeigen wir die ersten Zeilen des aktualisierten DataFrames an
print(df[['trany', 'Gears', 'Transmission Type']].head())
```

Ergebnis:

	trany	Gears	Transmission Type
0	Manual 5-spd	5.0	Manual
1	Manual 5-spd	5.0	Manual
2	Manual 5-spd	5.0	Manual
3	Automatic 3-spd	3.0	Automatic
4	Manual 5-spd	5.0	Manual

e) Gruppierung:

```
# Gruppieren des Datensatzes nach Jahr, Hersteller und Modell
grouped_df = df.groupby(['year', 'make', 'model']).agg(
    Average_Gears=('Gears', 'mean'), # Durchschnittliche Anzahl der Gänge
    Transmission_Type=('Transmission Type', 'first') # Übertragen des ersten Wertes
).reset_index()

# Zeigen der ersten Zeilen des gruppierten DataFrames an
print(grouped_df.head())
```

Ergebnis:

	year	make	model	Average_Gears	\
0	1984	AM General	DJ Po Vehicle 2WD	3.000000	
1	1984	AM General	FJ8c Post Office	3.000000	
2	1984	Alfa Romeo	GT V6 2.5	5.000000	
3	1984	Alfa Romeo	Spider Veloce 2000	5.000000	
4	1984	American Motors Corporation	Eagle 4WD	4.111111	

	Transmission_Type
0	Automatic
1	Automatic
2	Manual
3	Manual
4	Manual

f) Berechnung der Kennzahlen:

```
# Berechne die statistischen Kennzahlen für den gruppierten DataFrame
stats_df = grouped_df.describe(include='all')

# Ersetze NaN-Werte in den statistischen Kennzahlen durch 0
stats_df.fillna(0, inplace=True)

# Zeige die statistischen Kennzahlen an
print(stats_df)
```

Ergebnis:

	year	make	model	Average_Gears	Transmission_Type
count	23994.000000	23994	23994	15610.000000	23985
unique	0.000000	144	5064	0.000000	2
top	0.000000	BMW	F150 Pickup 2WD	0.000000	Automatic
freq	0.000000	1785	47	0.000000	21259
mean	2008.324414	0	0	5.178350	0
std	11.654304	0	0	1.379541	0
min	1984.000000	0	0	3.000000	0
25%	2000.000000	0	0	4.000000	0
50%	2011.000000	0	0	4.833333	0
75%	2018.000000	0	0	6.000000	0
max	2025.000000	0	0	10.000000	0

Aufgabe 4:

a) Extrahierung der CO2

```

# Aufgabe 4

import pandas as pd
import matplotlib.pyplot as plt

# Setz die Grenzen für die Co2 Klasse
bins = [0, 50, 100, 150, float('inf')] #Beispiele von Grenzen
labels = ['Kein Ausstoß', 'Geringer Ausstoß', 'Moderat', 'Hoher Ausstoß'] # 4 Labels für 5 Bins

# Dieskreditierung der CO2 Spalte
df['CO2_category'] = pd.cut(df['co2'], bins=bins, labels=labels, right=False)

# Die Farben definieren
colors = {
    'Kein Ausstoß': 'green',
    'Geringer Ausstoß': 'yellow',
    'Moderat': 'orange',
    'Hoher Ausstoß': 'red',
    'NaN': 'gray' # 'gray' p NaN
}

# Neue Spalte für die Farben erstellen
df['CO2_category'] = df['CO2_category'].astype('category') # Neue Spalte für die Kategorie erstellen
df['CO2_category'] = df['CO2_category'].cat.add_categories('gray') # 'gray' als Kategorie

# Zuweisung der Farben abhängig von der Kategorien
df['color'] = df['CO2_category'].map(colors)

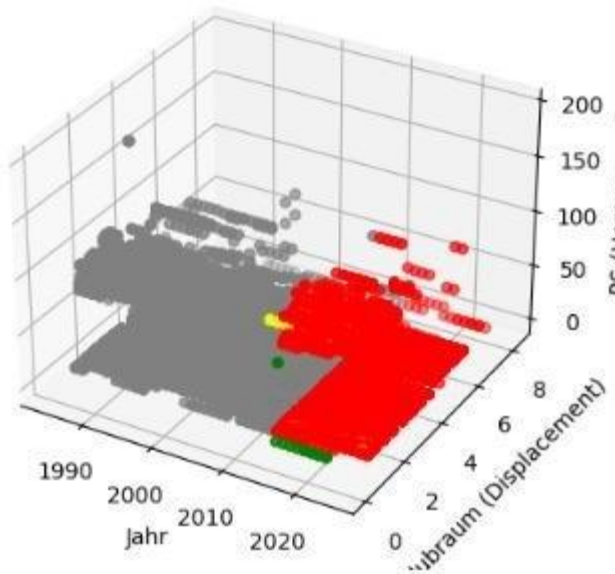
# NaN mit 'gray' ersetzen
df['color'] = df['color'].fillna('gray')

# scatter plot 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Scatterplot erstellen, z.B. mit Jahr, Hubraum (displ) und PS (hpv)
ax.scatter(df['year'], df['displ'], df['hpv'], c=df['color'])
ax.set_xlabel('Jahr')
ax.set_ylabel('Hubraum (Displacement)')
ax.set_zlabel('PS (Horsepower)')
plt.title('3D-Scatterplot der Fahrzeuge nach CO2-Ausstoß')
plt.show()
  
```

Ergebnis:

3D-Scatterplot der Fahrzeuge nach CO₂-Ausstoß



b) Kmeans Algorithmus und Visualisierung des Kluster

```

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Zuerst den kombinierten Verbrauch berechnen, bevor Zeilen mit NaN entfernt werden
df.loc[:, 'combined_consumption'] = 235.215 / df['comb08'] # Umrechnung in L/100 km

# Entferne Zeilen mit fehlenden Werten in den wichtigen Spalten
df = df.dropna(subset=['Gears', 'combined_consumption', 'cylinders'])

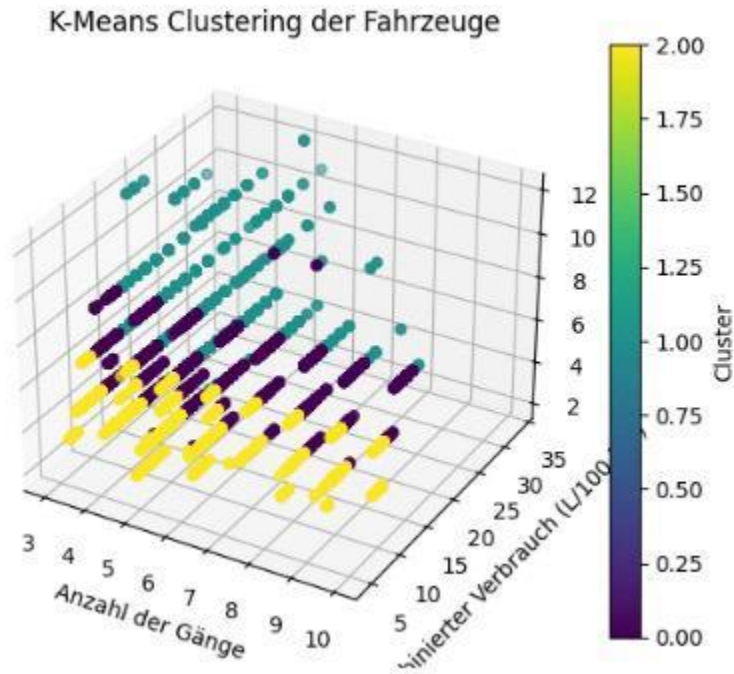
# Auswahl der Features für K-Means
features = df[['Gears', 'combined_consumption', 'cylinders']]

# K-Means Clustering anwenden
kmeans = KMeans(n_clusters=3, random_state=42) # Wähle die Anzahl der Cluster nach Bedarf
df['cluster'] = kmeans.fit_predict(features)

# 3D-Scatterplot erstellen, um die Cluster anzuzeigen
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(df['Gears'], df['combined_consumption'], df['cylinders'], c=df['cluster'], cmap='viridis')
ax.set_xlabel('Anzahl der Gänge')
ax.set_ylabel('Kombinierter Verbrauch (L/100 km)')
ax.set_zlabel('Anzahl der Zylinder')
plt.title('K-Means Clustering der Fahrzeuge')
plt.colorbar(scatter, label='Cluster')
plt.show()

```

Ergebnis:



c) Berechnung der Accuracy:

```
from sklearn.metrics import accuracy_score
from scipy.stats import mode
import numpy as np

# Konvertiere CO2-Kategorien in numerische Werte, falls dies noch nicht geschehen ist
category_mapping = {'Kein Ausstoß': 0, 'Geringer Ausstoß': 1, 'Moderat': 2, 'Hoher Ausstoß': 3}
df['CO2_category_numeric'] = df['CO2_category'].map(category_mapping)

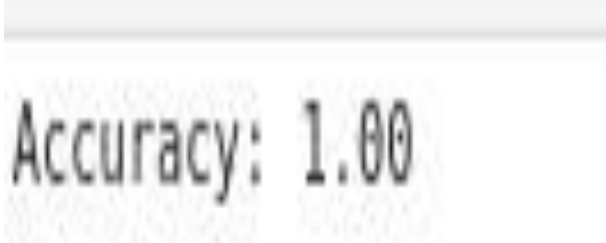
# Entferne Zeilen mit NaN-Werten in der CO2_category_numeric-Spalte und im Cluster
df = df.dropna(subset=['CO2_category_numeric', 'cluster'])

# Überprüfe, ob Cluster und CO2-Kategorien verfügbar sind
if 'CO2_category_numeric' in df.columns and 'cluster' in df.columns:
    # Mappe die Cluster-Labels auf die am häufigsten vorkommenden CO2-Kategorien in jedem Cluster
    cluster_labels = np.zeros_like(df['cluster'])
    for i in range(3): # für 3 Cluster
        mask = df['cluster'] == i
        cluster_labels[mask] = mode(df.loc[mask, 'CO2_category_numeric'])[0]

    # Berechne die Accuracy zwischen den neu zugewiesenen Cluster-Labels und den CO2-Kategorien
    accuracy = accuracy_score(df['CO2_category_numeric'], cluster_labels)
    print(f'Accuracy: {accuracy:.2f}')
else:
    print("Stelle sicher, dass die Spalten 'CO2_category_numeric' und 'cluster' vorhanden sind.")
```

Accuracy: 1.00

Ergebnis:



Accuracy: 1.00



DANKE