Projekt: Marketing-Campaign

1. Vorüberlegungen:
   - Bei der Auswahl des Datensatzes habe ich mich für den Bereich des Marketing entschieden. Es sollen hier Klassifizierungen angewendet und ein Clustering eingesetzt werden.
   - Marketing Unternehmen haben sich in Ihrem Wirkungsbereich in den vergangenen Jahren stark verändert. Hierbei ist die Gewichtung von Online, Store oder auch Katalog ganz unterschiedlich gestaltet. In Form von Kamapagnen wird dann eine Umsatzsteigerung mal mehr, mal weniger erreicht.
   - In diesem Projekt soll es also darum gehen, nach einem bestehenden Datensatz ein Machine Learning Modell zu erstellen, welches möglichst genau durch Klassifizierung vorhersagen kann, wie effektiv eine Kampagne sein wird.
   - Außerdem wird mit einem Algorithmus ein Clustering eingesetzt, um zu sehen, wie klar man dies segmentieren kann.
2. Ziel:
   - ✔ Unter einer Auswahl von verschiedenen Algorithmen aus dem Machine-Learning-Bereich wähle ich nach einem Rechenvergleich ein ML-Modell, mit welchem ich bestmöglich eine präzise Vorhersage treffen kann, ob ein Kunde aufgrund seiner individuellen Charakteristika bei einer Kampagne Produkt(e) kaufen wird oder nicht.
   - ✔ Kunden sollen mit Hilfe von einem Clustering Algorithmus segmentiert werden

Inhaltsverzeichnis

. Datensatzanalyse - Marketing-Kampagne

1. Kostenfreien Datensatz bei Kaggle importieren

- Datensatz eines Unternehmens
- Herunterladen und Einlesen das Datensatzes im .csv-Format

2. Analyse des Datensatzes

A) Datensatz mit seinen spezifischen Voraussetzungen:

- 29 unterschiedliche Features (columns) mit 2240 Samples (rows)
- 6 Features ('AcceptedCmp1', 'AcceptedCmp2','AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5','Response') bieten sich hier als Target („Label“) an, wobei man sich dann für eines entscheiden muß. Ich habe mich hierbei für „Response“ entschieden, weil es die besten Werte hat.
- Features Inhalt:
  - ✗ AcceptedCmp1 – 1, wenn der Kunde das Angebot in der 1. Kampagne angenommen hat, 0 sonst
  - ✗ AcceptedCmp2 – 1, wenn der Kunde das Angebot in der 2. Kampagne angenommen hat, 0 sonst
  - ✗ AcceptedCmp3 – 1, wenn der Kunde das Angebot in der 3. Kampagne angenommen hat, 0 sonst
  - ✗ AcceptedCmp4 – 1, wenn der Kunde hat das Angebot in der 4. Kampagne angenommen, 0 sonst
  - ✗ AcceptedCmp5 – 1, wenn der Kunde das Angebot in der 5. Kampagne angenommen hat, sonst 0
  - ✗ Response (Ziel) – 1, wenn der Kunde das Angebot in der letzten Kampagne angenommen hat, 0 sonst
  - ✗ Complain – 1, wenn der Kunde sich beschwert hat die letzten 2 Jahre

- ✗ DtCustomer – Datum der Anmeldung des Kunden beim Unternehmen.
- ✗ Education – Bildungsniveau des Kunden.
- ✗ Marital_Status – Familienstand des Kunden
- ✗ Kidhome – Anzahl der kleinen Kinder im Haushalt des Kunden
- ✗ Teenhome – Anzahl der Teenager im Haushalt des Kunden
- ✗ Income – jährliches Haushaltseinkommen des Kunden
- ✗ MntFishProducts – Betrag, der in den letzten 2 Jahren für Fischprodukte ausgegeben wurde
- ✗ MntMeatProducts – Betrag, der in den letzten 2 Jahren für Fleischprodukte ausgegeben wurde
- ✗ MntFruits – Betrag Ausgaben für Fruchtprodukte in den letzten 2 Jahren
- ✗ MntSweetProducts – Betrag, der in den letzten 2 Jahren für süße Produkte ausgegeben wurde
- ✗ MntWines – Betrag, der in den letzten 2 Jahren für Weinprodukte ausgegeben wurde
- ✗ MntGoldProds – Betrag, der in den letzten 2 Jahren für Goldprodukte ausgegeben wurde
- ✗ NumDealsPurchases – Anzahl der Käufe mit Rabatt getätigt
- ✗ NumCatalogPurchases – Anzahl der über den Katalog getätigten Käufe
- ✗ NumStorePurchases – Anzahl der direkt in Geschäften getätigten Einkäufe
- ✗ NumWebPurchases – Anzahl der über die Website des Unternehmens getätigten Einkäufe
- ✗ NumWebVisitsMonth – Anzahl der Besuche auf der Website des Unternehmens im letzten Monat
- ✗ Recency – Anzahl der Tage seit dem letzten Kauf

B) den Datensatz im Detail untersuchen:

- notwendige Bibliotheken zur Bearbeitung des Datensatzes importieren (Numpy, Pandas, MatPlotLib, Seaborn...)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_theme()
# Ausführen, für den Fall, dass  Warnungen ignoriert werden sollen
import warnings
warnings.filterwarnings('ignore')

#pd.set_option('display.max_rows', 2500)          # Reihen
pd.set_option('display.max_columns', 35)        # alle Spalten anzeigen
```

- Daten in Jupyter Notebook importieren und DataFrame (Pandas) erstellen mit pd.read_csv("...")
- Datensatzausschnitt:

	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58	635	88	546	172
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38	11	1	6	2
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26	426	49	127	111
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	26	11	4	20	10
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94	173	43	118	46

- Wo gibt es Nullwerte:

```
df01.isnull().sum()
```

```
ID                      0
Year_Birth              0
Education               0
Marital_Status          0
Income                 24
Kidhome                 0
Teenhome                0
Dt_Customer             0
Recency                 0
MntWines                0
MntFruits               0
MntMeatProducts         0
MntFishProducts         0
MntSweetProducts        0
MntGoldProds            0
NumDealsPurchases       0
NumWebPurchases         0
NumCatalogPurchases     0
NumStorePurchases       0
NumWebVisitsMonth       0
AcceptedCmp3            0
AcceptedCmp4            0
AcceptedCmp5            0
AcceptedCmp1            0
AcceptedCmp2            0
Complain                0
Z_CostContact           0
Z_Revenue               0
Response                0
dtype: int64
```

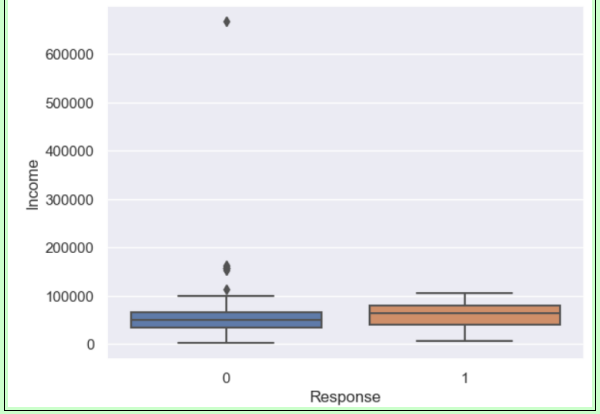
Die Nullwerte durch Durchschnittswerte ersetzen:

```
df01["Income"].fillna(df01["Income"].mean(),inplace=True)
df01_vis["Income"].fillna(df01["Income"].mean(),inplace=True)
```

Im Boxplot kann ich erkennen, dass es noch Ausreißer gibt:

```
sns.boxplot(data=df01,x=df01["Response"],y=df01["Income"])
```

```
<Axes: xlabel='Response', ylabel='Income'>
```



Eine Funktion soll die Ausreißerdaten durch Durchschnittswerte ersetzen:

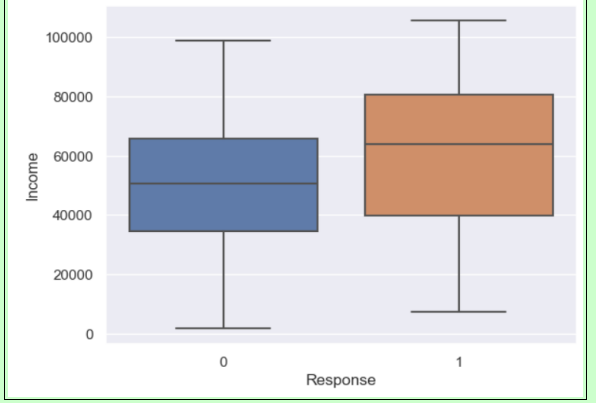
```
def ausr_income(x):
    if x > 110000:
        return df01["Income"].mean()
    else:
        return x
```

```
df01["Income"] = df01.Income.apply(ausr_income)
df01_vis["Income"] = df01_vis.Income.apply(ausr_income)
```

Der nächste Boxplot zeigt die Korektur an:

```
sns.boxplot(data=df01,x=df01["Response"],y=df01["Income"])
```

```
<Axes: xlabel='Response', ylabel='Income'>
```



- Bei dem Feature „Year_Birth“ habe ich diegleiche Problematik wie bei „Income“ und behebe sie auf dieselbe Weise. Außerdem erstelle ich mir eine neue Spalte „Age“, um mir das Alter in Jahren anzeigen zu lassen und lösche die alte Spalte.

```
df01["Age"] = 2023 - df01["Year_Birth"]
df01.drop("Year_Birth",axis=1,inplace=True)
df01_vis["Age"] = 2023 - df01_vis["Year_Birth"]
df01_vis.drop("Year_Birth",axis=1,inplace=True)
```

- Eine neue Spalte für alle Kampagnen soll mir weitere Vergleichsmöglichkeiten bieten. Es sind alle Kampagnen aufsummiert, somit kann man sehen, welcher Kunde wie viele Kampagnen genutzt hat.

```
# neue spalte für Total_Campaigns_Accepted (target gesamt):
df01["AcceptedCmp_Total"] = df01["AcceptedCmp3"] + df01["AcceptedCmp4"] + df01["AcceptedCmp5"] + df01["AcceptedCmp1"] +
df01_vis["AcceptedCmp_Total"] = df01_vis["AcceptedCmp3"] + df01_vis["AcceptedCmp4"] + df01_vis["AcceptedCmp5"] + df01_v
```

- Nun habe ich 2240 Samples und 30 Columns
- Bevor ich den Datensatz exportiere, überprüfe ich nochmal mit .describe(), ob auch keine unnützen Daten vorhanden sind.

```
df01.describe()
```

	ID	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts	Mnt
count	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2
mean	5592.159821	51614.954596	0.444196	0.506250	49.109375	303.935714	26.302232	166.950000	37.525446	27.062946	
std	3246.662198	20523.031693	0.538398	0.544538	28.962453	336.597393	39.773434	225.715373	54.628979	41.280498	
min	0.000000	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2828.250000	35538.750000	0.000000	0.000000	24.000000	23.750000	1.000000	16.000000	3.000000	1.000000	
50%	5458.500000	51741.500000	0.000000	0.000000	49.000000	173.500000	8.000000	67.000000	12.000000	8.000000	
75%	8427.750000	67956.250000	1.000000	1.000000	74.000000	504.250000	33.000000	232.000000	50.000000	33.000000	
max	11191.000000	105471.000000	2.000000	2.000000	99.000000	1493.000000	199.000000	1725.000000	259.000000	263.000000	

- Ich exportiere den preparierten Datensatz, damit ich ihn für Presentationszwecke (Diagramme in Tableau) nutzen kann:

```
# gesamtdatensatz bereinigt exportieren:
df01_vis.to_csv("marketing_campaign_vis.csv",index=False)
```

3. Daten preparieren/bereinigen

A) Überblick verschaffen: → welche unterschiedliche Datentypen gibt es? Bei 3 Features gibt es keine numerischen Werte. Damit ich sie für das ML nutzen kann, muß ich sie noch umwandeln.

```
df01.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 30 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   ID                   2240 non-null   int64
 1   Education            2240 non-null   object
 2   Marital_Status       2240 non-null   object
 3   Income               2240 non-null   float64
 4   Kidhome              2240 non-null   int64
 5   Teenhome             2240 non-null   int64
 6   Dt_Customer          2240 non-null   object
 7   Recency              2240 non-null   int64
 8   MntWines             2240 non-null   int64
 9   MntFruits            2240 non-null   int64
 10  MntMeatProducts      2240 non-null   int64
 11  MntFishProducts      2240 non-null   int64
 12  MntSweetProducts     2240 non-null   int64
 13  MntGoldProds         2240 non-null   int64
 14  NumDealsPurchases    2240 non-null   int64
 15  NumWebPurchases      2240 non-null   int64
 16  NumCatalogPurchases  2240 non-null   int64
 17  NumStorePurchases    2240 non-null   int64
 18  NumWebVisitsMonth    2240 non-null   int64
 19  AcceptedCmp3         2240 non-null   int64
 20  AcceptedCmp4         2240 non-null   int64
 21  AcceptedCmp5         2240 non-null   int64
 22  AcceptedCmp1         2240 non-null   int64
 23  AcceptedCmp2         2240 non-null   int64
 24  Complain             2240 non-null   int64
 25  Z_CostContact        2240 non-null   int64
 26  Z_Revenue            2240 non-null   int64
 27  Response             2240 non-null   int64
 28  Age                  2240 non-null   float64
 29  AcceptedCmp_Total    2240 non-null   int64
dtypes: float64(2), int64(25), object(3)
memory usage: 525.1+ KB
```

```
def mari(x):
    if x == "Absurd" or x == "YOLO" or x == "Alone":
        return 0
    if x == "Single":
        return 1
    if x == "Widow":
        return 2
    if x == "Divorced":
        return 3
    if x == "Together":
        return 4
    if x == "Married":
        return 5
```

```
df01["Marital_Status"] = df01["Marital_Status"].apply(mari)
```

```
def educ(x):
    if x == "Basic":
        return 0
    if x == "Graduation":
        return 1
    if x == "2n Cycle":
        return 2
    if x == "Master":
        return 3
    if x == "PhD":
        return 4

```

```
df01['Education'] = df01['Education'].apply(educ)
```

B) Nicht benötigte Features, z.B. ein einziger Wert in allen Samples, lösche ich:

```
# nicht benötigte spalten löschen
df01.drop(["ID","Dt_Customer","Z_CostContact","Z_Revenue"],axis=1,inplace=True)
df01_vis.drop(["ID","Dt_Customer","Z_CostContact","Z_Revenue"],axis=1,inplace=True)
```

C) Wie viele unterschiedliche Werte sind in den einzelnen Features?

```
df01.nunique()
```

```
Education              5
Marital_Status         8
Income              1966
Kidhome                3
Teenhome               3
Recency              100
MntWines             776
MntFruits            158
MntMeatProducts      558
MntFishProducts      182
MntSweetProducts     177
MntGoldProds         213
NumDealsPurchases     15
NumWebPurchases       15
NumCatalogPurchases   14
NumStorePurchases     14
NumWebVisitsMonth     16
AcceptedCmp3           2
AcceptedCmp4           2
AcceptedCmp5           2
AcceptedCmp1           2
AcceptedCmp2           2
Complain               2
Response               2
Age                   57
AcceptedCmp_Total      6
dtype: int64
```

D) der Datensatz beinhaltet nach Bereinigung/Aufbereitung 26 Features insgesamt, davon sind mögliche Target-Features 7 und es gibt 2240 Samples

```
1 df01.shape
```

```
(2240, 26)
```

E) deskriptive Datenanalyse:

```
1 df01.describe()
```

	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweetProducts	MntGoldProds	Nu
count	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	2240.000000	
mean	51614.954596	0.444196	0.506250	49.109375	303.935714	26.302232	166.950000	37.525446	27.062946	44.021875	
std	20523.031693	0.538398	0.544538	28.962453	336.597393	39.773434	225.715373	54.628979	41.280498	52.167439	
min	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	35538.750000	0.000000	0.000000	24.000000	23.750000	1.000000	16.000000	3.000000	1.000000	9.000000	
50%	51741.500000	0.000000	0.000000	49.000000	173.500000	8.000000	67.000000	12.000000	8.000000	24.000000	
75%	67956.250000	1.000000	1.000000	74.000000	504.250000	33.000000	232.000000	50.000000	33.000000	56.000000	
max	105471.000000	2.000000	2.000000	99.000000	1493.000000	199.000000	1725.000000	259.000000	263.000000	362.000000	

4. Korrelationen der Features aufzeigen:

A) Um die Korrelationen aufzuzeigen, entferne ich die Target-Features bis auf die der Gesamtkäufe, bei welcher alle Kampagnen enthalten sind, damit das Ergebnis aussagekräftiger wird:

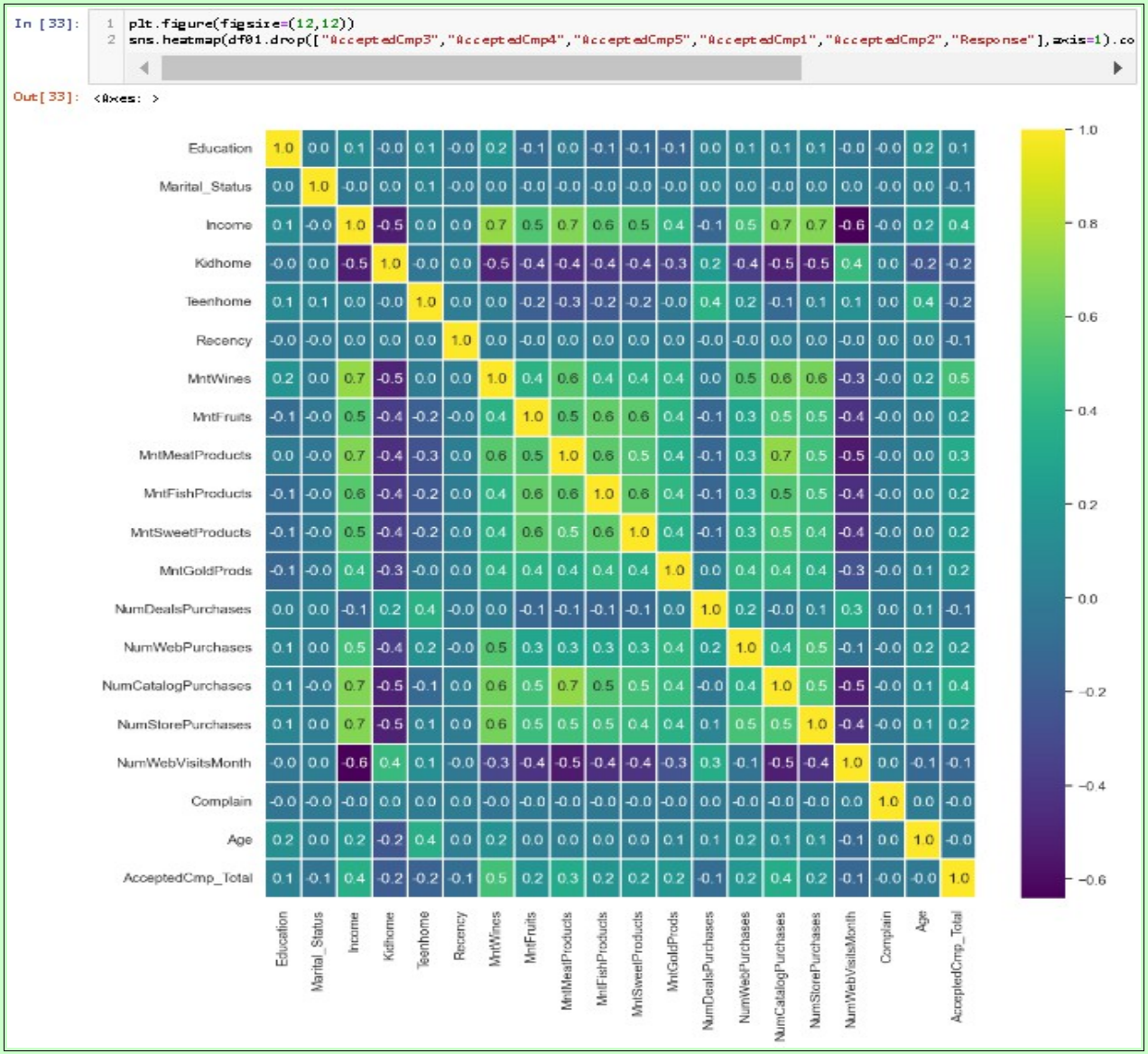
```
# correlationen:
df01.drop(["AcceptedCmp3","AcceptedCmp4","AcceptedCmp5","AcceptedCmp1","AcceptedCmp2","Response"],axis=1).corr()
```

	Education	Marital_Status	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSw
Education	1.000000	0.012903	0.114504	-0.038017	0.102005	-0.025885	0.169076	-0.098455	0.004568	-0.106461	
Marital_Status	0.012903	1.000000	-0.002038	0.011088	0.056618	-0.008141	0.003916	-0.018628	-0.032292	-0.026983	
Income	0.114504	-0.002038	1.000000	-0.526830	0.043259	0.007574	0.729705	0.537378	0.675895	0.551815	
Kidhome	-0.038017	0.011088	-0.526830	1.000000	-0.036133	0.008827	-0.496297	-0.372581	-0.437129	-0.387644	
Teenhome	0.102005	0.056618	0.043259	-0.036133	1.000000	0.016198	0.004846	-0.176764	-0.261160	-0.204187	
Recency	-0.025885	-0.008141	0.007574	0.008827	0.016198	1.000000	0.016064	-0.004306	0.023056	0.001079	
MntWines	0.169076	0.003916	0.729705	-0.496297	0.004846	0.016064	1.000000	0.389637	0.562667	0.399753	
MntFruits	-0.098455	-0.018628	0.537378	-0.372581	-0.176764	-0.004306	0.389637	1.000000	0.543105	0.594804	
MntMeatProducts	0.004568	-0.032292	0.675895	-0.437129	-0.261160	0.023056	0.562667	0.543105	1.000000	0.568402	
MntFishProducts	-0.106461	-0.026983	0.551815	-0.387644	-0.204187	0.001079	0.399753	0.594804	0.568402	1.000000	
MntSweetProducts	-0.102200	-0.013780	0.542997	-0.370673	-0.162475	0.022670	0.386581	0.567164	0.523846	0.579870	
MntGoldProds	-0.121540	-0.022950	0.414867	-0.349595	-0.021725	0.016693	0.387516	0.392995	0.350609	0.422875	
NumDealsPurchases	0.028999	0.040899	-0.127062	0.221798	0.387741	-0.001098	0.010940	-0.132114	-0.122415	-0.139361	
NumWebPurchases	0.063170	0.014800	0.481313	-0.361647	0.155500	-0.010726	0.542265	0.296735	0.293761	0.293681	
NumCatalogPurchases	0.051224	-0.004779	0.677598	-0.502237	-0.110769	0.025110	0.635226	0.487917	0.723827	0.534478	
NumStorePurchases	0.062010	0.015950	0.682698	-0.499683	0.050695	0.000799	0.642100	0.461758	0.479659	0.459855	
NumWebVisitsMonth	-0.034857	0.013436	-0.639735	0.447846	0.134884	-0.021445	-0.320653	-0.418383	-0.539470	-0.446003	
Complain	-0.039920	-0.008113	-0.030214	0.040207	0.003138	0.013231	-0.039007	-0.005166	-0.023483	-0.020953	
Age	0.153679	0.033592	0.208728	-0.234008	0.363140	0.019654	0.162914	0.013728	0.030900	0.042500	
AcceptedCmp_Total	0.051516	-0.052838	0.365659	-0.193419	-0.159110	-0.088962	0.489211	0.172769	0.330172	0.180003	

B) Heatmap mit den 19 Features

```
In [33]:  1 plt.figure(figsize=(12,12))
          2 sns.heatmap(df01.drop(["AcceptedCmp3","AcceptedCmp4","AcceptedCmp5","AcceptedCmp1","AcceptedCmp2","Response"],axis=1).co

Out[33]: <Axes: >
```



C) Korrelation mit der einzelnen Spalte „AcceptedCmp_Total“:

	index	AcceptedCmp_Total
0	Response	0.72
1	AcceptedCmp5	0.68
2	AcceptedCmp1	0.64
3	AcceptedCmp4	0.54
4	MntWines	0.49
5	AcceptedCmp3	0.43
6	AcceptedCmp2	0.42
7	Income	0.37
8	NumCatalogPurchases	0.35
9	MntMeatProducts	0.33
10	NumWebPurchases	0.21
11	MntGoldProds	0.20
12	MntSweetProducts	0.20
13	MntFishProducts	0.18
14	NumStorePurchases	0.17

14	NumStorePurchases	0.17
15	MntFruits	0.17
16	Education	0.05
17	Age	-0.01
18	Complain	-0.02
19	Marital_Status	-0.05
20	NumDealsPurchases	-0.09
21	Recency	-0.09
22	NumWebVisitsMonth	-0.13
23	Teenhome	-0.16
24	Kidhome	-0.19

D) höchste Korrelationen mit folgenden Features:

- ✔ KORRELATION 0.7:
  - Income - MntWines
  - Income - MntMeatProducts
  - Income - NumCatalogPurchases
  - Income - NumStorePurchases
  - MntMeatProducts - NumCatalogPurchases
- ✔ KORRELATION 0.6:
  - MntWines - MntMeatProducts

- MntWines - NumCatalogPurchases
- MntWines - NumCatalogStore
- MntFruits - MntFishProducts
- MntFruits - MntSweetProducts
- MntMeatProducts - MntFish
- MntFish - MntSweetProducts

✔ HÖCHSTE KORRELATIONEN MIT DEM LABEL AcceptedCmp_Total:

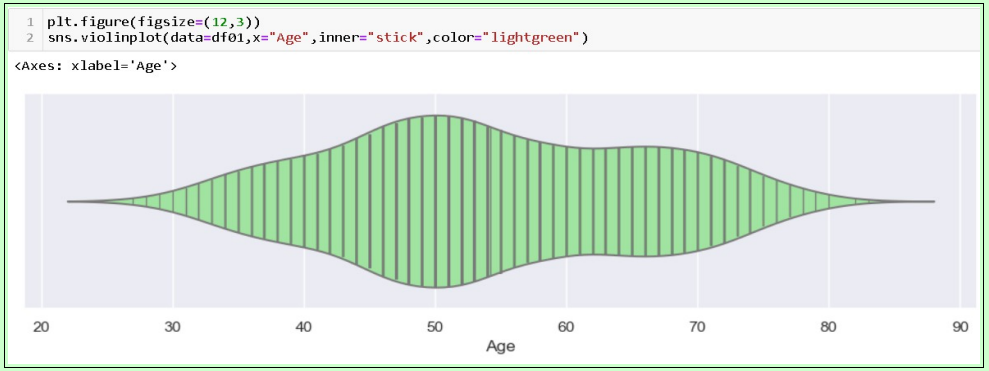
- Response - AcceptedCmp_Total 0.72
- AcceptedCmp5 - AcceptedCmp_Total 0.68
- AcceptedCmp1 - AcceptedCmp_Total 0.64
- AcceptedCmp4 - AcceptedCmp_Total 0.54
- MntWines - AcceptedCmp_Total 0,49
- AcceptedCmp3 - AcceptedCmp_Total 0.43
- AcceptedCmp2 - AcceptedCmp_Total 0.42
- Income - AcceptedCmp_Total 0.36
- MntMeatProducts - AcceptedCmp_Total 0.33

5. Visualisierungen

A) Alters-Verteilung in einem Violinplot:

```
plt.figure(figsize=(12,3))
sns.violinplot(data=df01,x="Age",inner="stick",color="lightgreen")
```

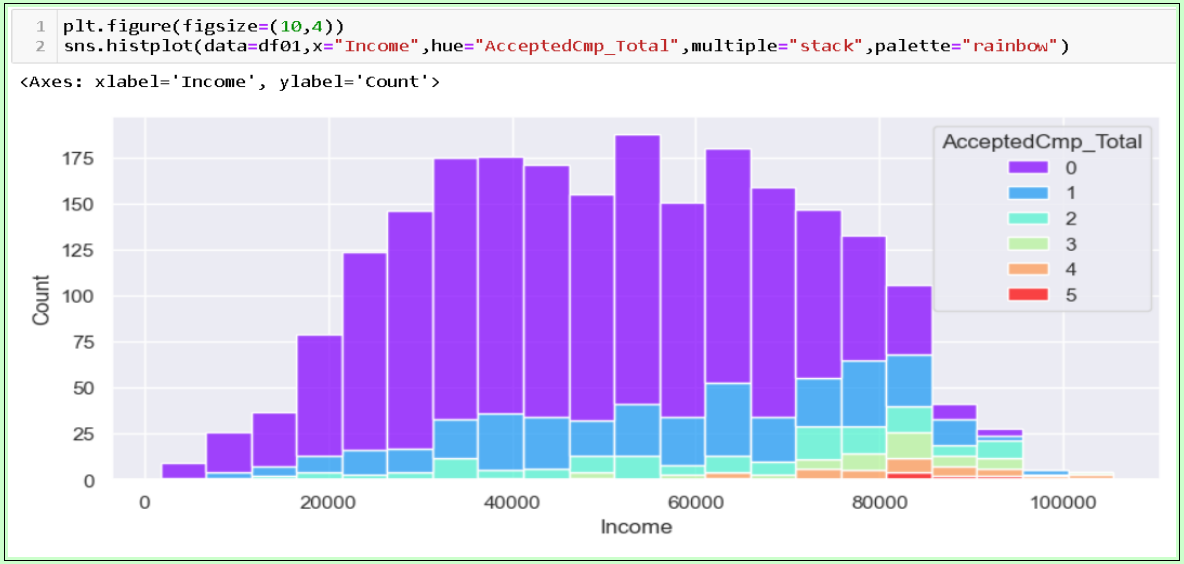
```
<Axes: xlabel='Age'>
```



B) Kampagnen-Verteilung im Bezug auf das Einkommen: Im Konkretem bedeutet es mit „0“, dass ein Kunde eine Kampagne, mit „1“, dass er 2 Kampagnen angenommen hat und soweiter:

```
plt.figure(figsize=(10,4))
sns.histplot(data=df01,x="Income",hue="AcceptedCmp_Total",multiple="stack",palette="rainbow")
```

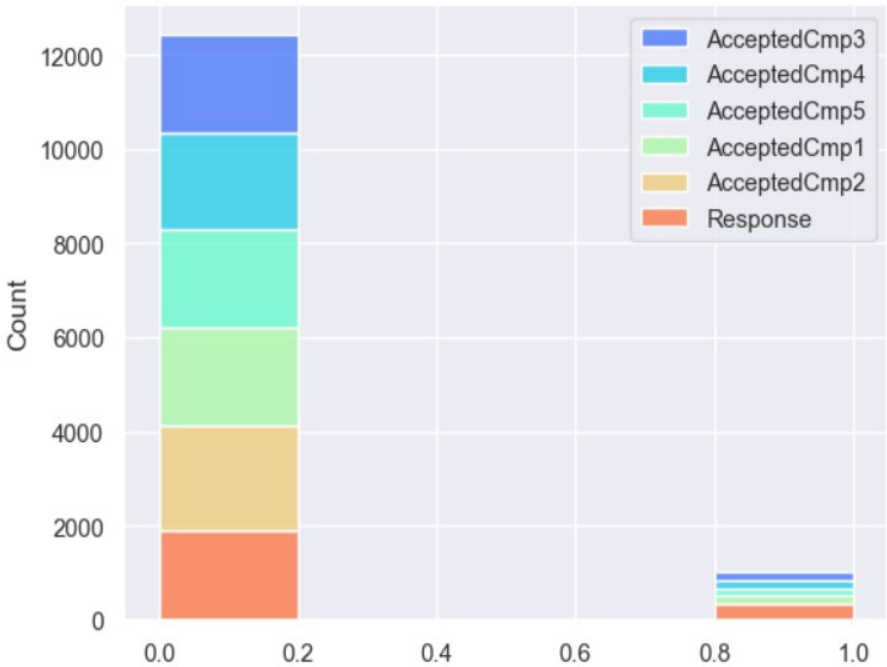
```
<Axes: xlabel='Income', ylabel='Count'>
```



C) Erfolg der einzelnen Kampagnen mit der „Response“-Kampagne an der Spitze (Erfolg: Ja =1,Nein=0):

```
# erfolg der campaigns:
sns.histplot(data=df01[["AcceptedCmp3","AcceptedCmp4","AcceptedCmp5","Acce
```

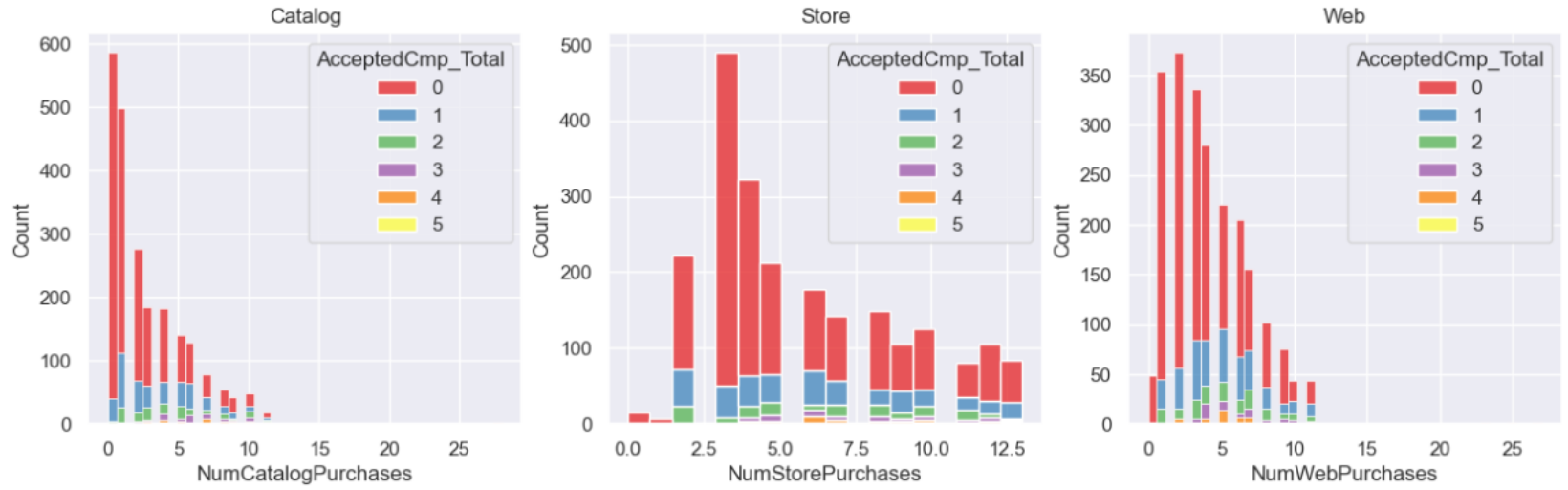
```
<Axes: ylabel='Count'>
```



D) Erfolg der Kampagnen im Bezug auf die Verkaufsarten „Katalog“, „Geschäft“ und „Online“:

```
# vergleich der unterschiedlichen verkaufsarten:
fig, axes = plt.subplots(1, 3,  figsize=(15, 4))
sns.histplot(data=df01_vis, x="NumCatalogPurchases", hue="AcceptedCmp_Total", multiple="stack", palette="Set1",ax=axes[
axes[0].set_title("Catalog")
sns.histplot(data=df01_vis, x="NumStorePurchases", hue="AcceptedCmp_Total", multiple="stack", palette="Set1", ax=axes[1
axes[1].set_title('Store')
sns.histplot(data=df01_vis, x="NumWebPurchases", hue="AcceptedCmp_Total", multiple="stack", palette="Set1", ax=axes[2])
axes[2].set_title('Web')
```

```
Text(0.5, 1.0, 'Web')
```

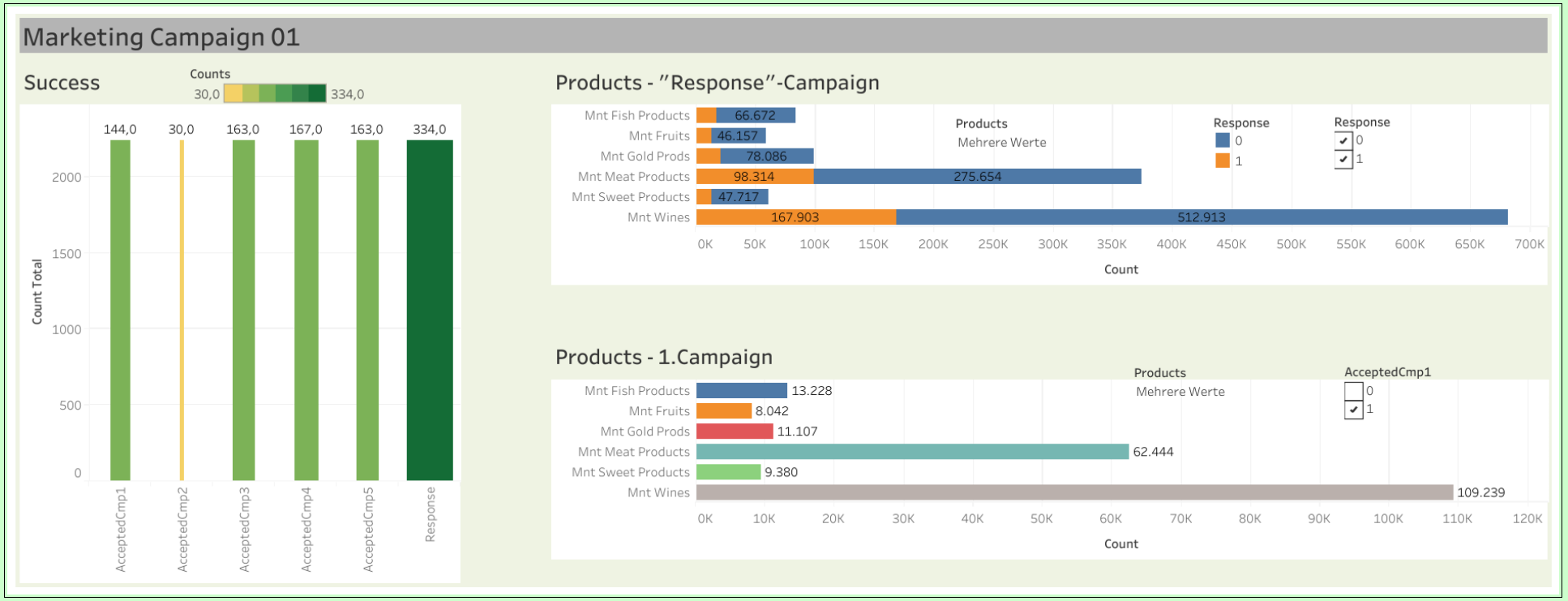


6. DataFrame exportieren für weitere Anwendung in Tableau:

A) Verwendung des exportierten Datensatzes marketing_campaign_vis.csv für die Präsentationen in Tableau

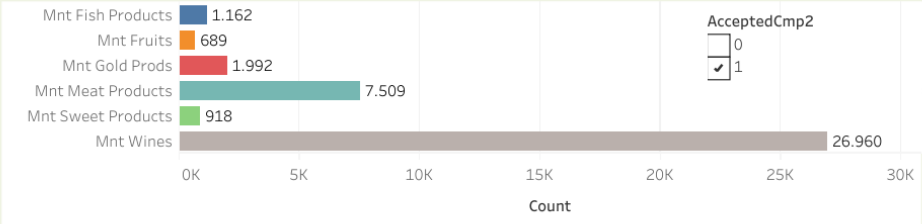
B) Präsentation der Analyse des Datensatzes erreichbar unter folgendem Link:: https://public.tableau.com/app/profile/dietmar.kiendl/viz/Marketing_Campaign_16965832086190/MarketingCampaign01

C) der Link ist notwendig, da man sonst die Filter, die in den Dashboards eingebaut sind, nicht für eine spezifische Betrachtungsweise der Diagramme nutzen kann.

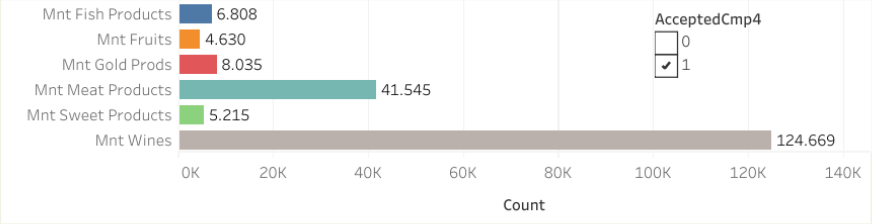


Marketing Campaign 02

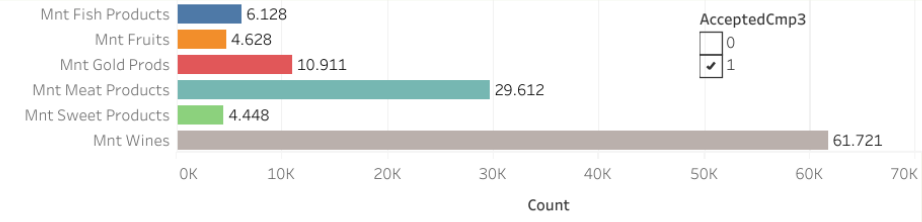
Products - 2.Campaign



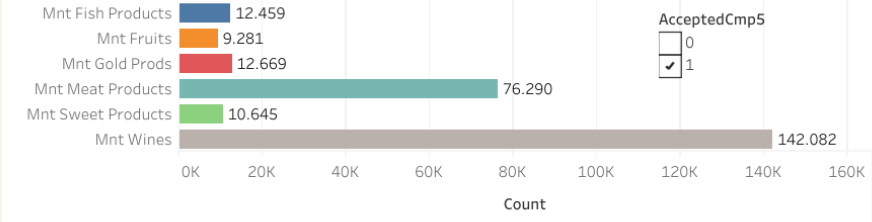
Products - 4.Campaign



Products - 3.Campaign

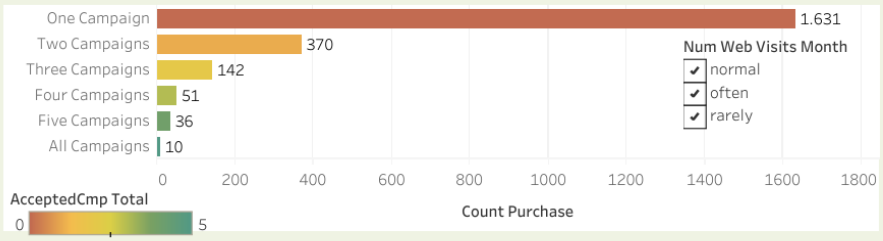


Products - 5.Campaign

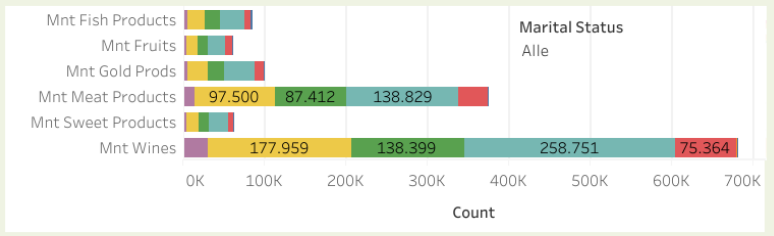


Marketing Campaign 03

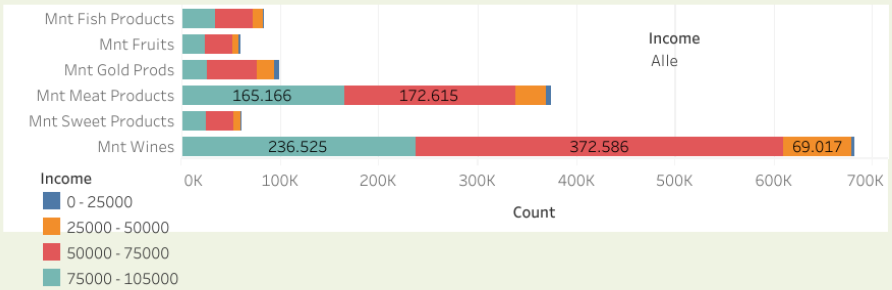
Accepts of One Customer



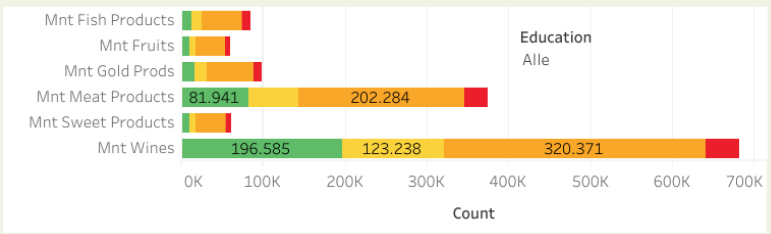
Products - ”Response”-Campaign - Status



Products - ”Response”-Campaign - Income

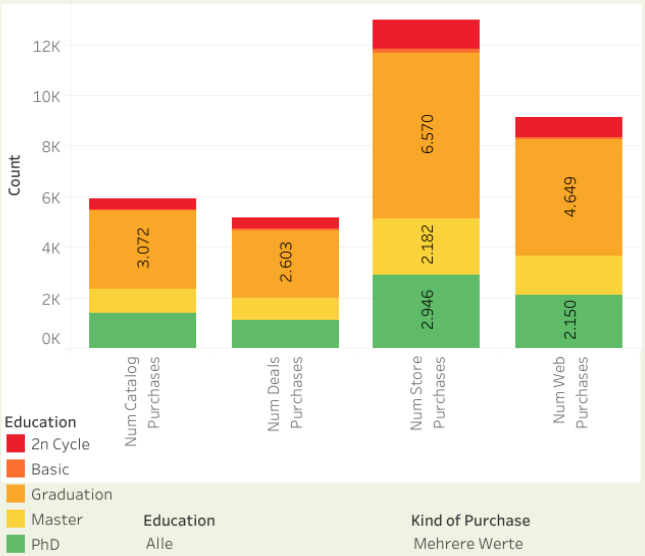


Products - ”Response”-Campaign - Education



Marketing Campaign 04

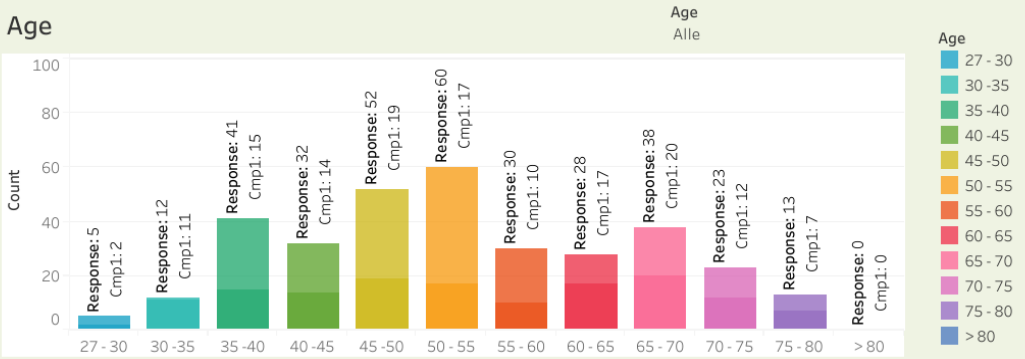
Purchase



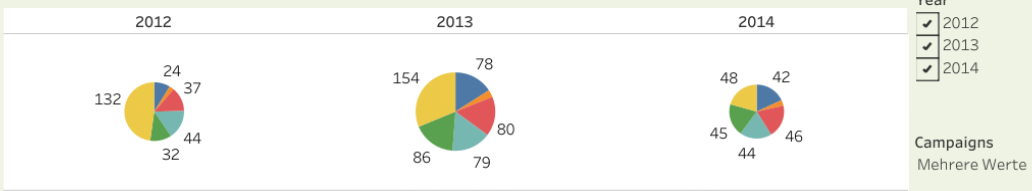
Education
Alle

Kind of Purchase
Mehrere Werte

Age



Duration of Customer Relationship



. Bauen eines Machine Learning Modells

1 train_test_split --> ML, Supervised/Unsupervised Learning

A) Supervised Learning mit allen Features

- ✗ train-/test-daten erstellen → Bibliothek importieren: from sklearn.model_selection import train_test_split

```
# train-test-split
X01 = df01.drop(["Response","AcceptedCmp_Total"],axis=1)
#         (target = 6.campaign statt 1.campaign --> bessere vorhersagen!!)
y01 = df01["Response"]
```

```
X01.shape
```

```
(2240, 24)
```

```
y01.shape
```

```
(2240,)
```

- ✗ X : Train/Test-Daten → alle Features außer "Response"
- ✗ y : Train/Test-Daten → nur Feature "Response"

```

# train-/test-daten:
from sklearn.model_selection import train_test_split

```

```
X01_train, X01_test, y01_train, y01_test = train_test_split(X01, y01, test_size=0.2, random_state=33)
```

- ✗ alle X-daten standardisieren: mit der Bibliothek „StandardScaler“

```
# alle X-daten standardisieren:
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X01_train = scaler.fit_transform(X01_train)
X01_test = scaler.fit_transform(X01_test)
```

✗ alle Bibliotheken mit benötigten Algorithmen importieren:

```
# algorithmen: LogReg,KNeighbor(elbow),SVC(c/gamma-->gridsearch),naiveBayes,randomRorest
```

```
# algorithmen importieren:
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

#initialisieren:
log = LogisticRegression()
knc = KNeighborsClassifier()
nab = GaussianNB()
rfc = RandomForestClassifier()
svc = SVC()
```

(1) LogisticRegression (voraussichtlich am günstigsten für Kauf-Klassifizierungen)

(2) KneighborsClassifieror (Elbow)

(3) NaiveBayes

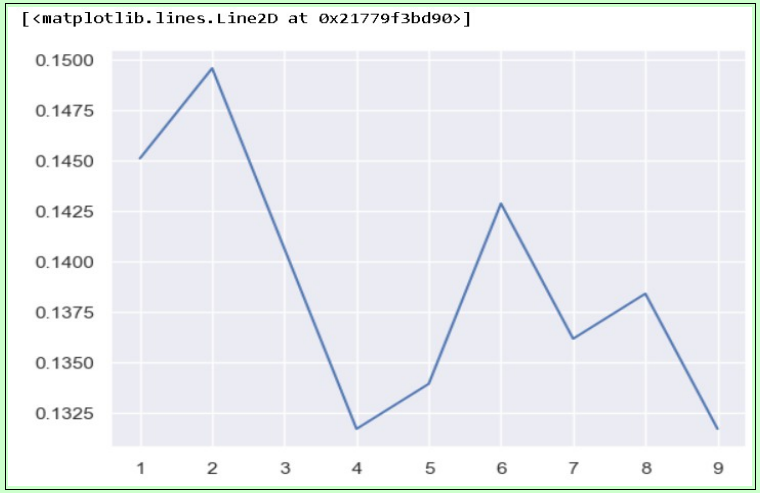
(4) RandomForestClassifier

(5) SVC (mit Gridsearch → bestes C/gamma ermitteln)

✗ mit der Elbow-Method das beste k für n_neighbors ermitteln:

```
# elbow method:
fehler = []
for k in range(1,10):
    knc = KNeighborsClassifier(n_neighbors=k)
    knc.fit(X01_train,y01_train)
    predictions_k = knc.predict(X01_test)
    fehler.append(np.mean(predictions_k != y01_test))
plt.plot(range(1,10),fehler)
#                                wählen  --> k = 4!
```

```
[<matplotlib.lines.Line2D at 0x21779f3bd90>]
```



✗ beste werte für C und gamma (SVC, Hyperebenen) ermitteln

```
#beste werte für C und gamma (SVC,hyperebenen):
hyper_para = {"C":[0.1,1,10,100,1000],"gamma":[1,0.1,0.01,0.001,0.0001]}
# initialisieren von Gridsearch
grid = GridSearchCV(SVC(),hyper_para,refit=True)
grid.fit(X01_train,y01_train)
```

▸ **GridSearchCV**
▸ **estimator: SVC**
▸ SVC

```
grid.best_params_
```

```
{'C': 100, 'gamma': 0.001}
```

- ✗ Initialisieren und Trainieren: log = LogisticRegression().fit(X_train,y_train) und auch mit den anderen 4 Algorithmen, um sie zu vergleichen

```
# neu initialisieren nach "elbow" und "C/gamma" und trainieren:
log = LogisticRegression().fit(X01_train,y01_train)
knc = KNeighborsClassifier(n_neighbors=4).fit(X01_train,y01_train)
nab = GaussianNB().fit(X01_train,y01_train)
rfc = RandomForestClassifier(n_estimators=1000,random_state=33).fit(X01_train,y01_train)
svc = SVC(C=100,gamma=0.001).fit(X01_train,y01_train)
```

- ✗ Vorhersagen für alle 5 Alg.: pred_log = log.predict(X_test)

```
# vorhersagen:
pred_log = log.predict(X01_test)
pred_knc = knc.predict(X01_test)
pred_nab = nab.predict(X01_test)
pred_rfc = rfc.predict(X01_test)
pred_svc = svc.predict(X01_test)
```

- ✗ Bibliotheken importieren, um die Accuracy, Precision, Recall, etc. des jeweiligen Algorithmus zu ermitteln → bester Algorithmus

```
from sklearn.metrics import accuracy_score,confusion_matrix,mean_absolute_error,classification_report
```

- ✗ Errechnung der Accuracies zeigt den besten Wert bei der logistischen Regression mit 88.39%:

```
print("LogReg      : {:.2f} %".format(accuracy_score(y01_test,pred_log)*100))
print("KNeighbor   : {:.2f} %".format(accuracy_score(y01_test,pred_knc)*100))
print("NaiveBayes  : {:.2f} %".format(accuracy_score(y01_test,pred_nab)*100))
print("RandomForest: {:.2f} %".format(accuracy_score(y01_test,pred_rfc)*100))
print("SVC         : {:.2f} %".format(accuracy_score(y01_test,pred_svc)*100))
```

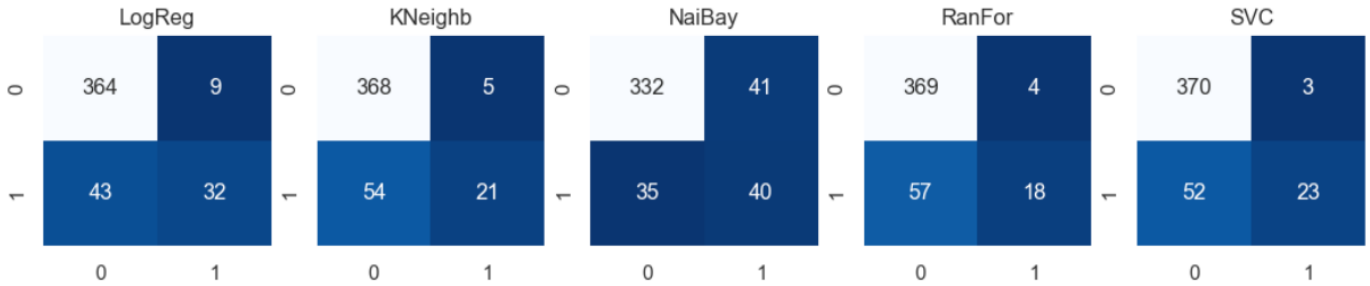
```
LogReg      : 88.39 %
KNeighbor   : 86.83 %
NaiveBayes  : 83.04 %
RandomForest: 86.38 %
SVC         : 87.72 %
```

✗ Confusion-Matrix mit allen Features:

```
fig,axis = plt.subplots(1,5,figsize=(13,2))

sns.heatmap(confusion_matrix(y01_test,pred_log),cbar = False,cmap="Blues_r",annot=True,fmt=".0f",ax=axis[0])
axis[0].set_title("LogReg")
sns.heatmap(confusion_matrix(y01_test,pred_knc),cbar = False,cmap="Blues_r",annot=True,fmt=".0f",ax=axis[1])
axis[1].set_title("KNeighb")
sns.heatmap(confusion_matrix(y01_test,pred_nab),cbar = False,cmap="Blues_r",annot=True,fmt=".0f",ax=axis[2])
axis[2].set_title("NaiBay")
sns.heatmap(confusion_matrix(y01_test,pred_rfc),cbar = False,cmap="Blues_r",annot=True,fmt=".0f",ax=axis[3])
axis[3].set_title("RanFor")
sns.heatmap(confusion_matrix(y01_test,pred_svc),cbar = False,cmap="Blues_r",annot=True,fmt=".0f",ax=axis[4])
axis[4].set_title("SVC")
```

```
Text(0.5, 1.0, 'SVC')
```



2 Confusion-Matrix-Algorithmen-Vergleich:

LogisticRegression

	precision	recall	f1-score	support
0	0.89	0.98	0.93	373
1	0.78	0.43	0.55	75
accuracy			0.88	448
macro avg	0.84	0.70	0.74	448
weighted avg	0.88	0.88	0.87	448

KNeighborsClassifier

	precision	recall	f1-score	support
0	0.87	0.99	0.93	373
1	0.81	0.28	0.42	75
accuracy			0.87	448
macro avg	0.84	0.63	0.67	448
weighted avg	0.86	0.87	0.84	448

NaiveBayes

	precision	recall	f1-score	support
0	0.90	0.89	0.90	373
1	0.49	0.53	0.51	75
accuracy			0.83	448
macro avg	0.70	0.71	0.71	448
weighted avg	0.84	0.83	0.83	448

RandomForestClassifier

	precision	recall	f1-score	support
0	0.87	0.99	0.92	373
1	0.82	0.24	0.37	75
accuracy			0.86	448
macro avg	0.84	0.61	0.65	448
weighted avg	0.86	0.86	0.83	448

SVC

	precision	recall	f1-score	support
0	0.88	0.99	0.93	373
1	0.88	0.31	0.46	75
accuracy			0.88	448
macro avg	0.88	0.65	0.69	448
weighted avg	0.88	0.88	0.85	448

Accuracy-Score-Gesamt-Vergleich:

```
LogReg       : 88.39 %
KNeighbor    : 86.83 %
NaiveBayes   : 83.04 %
RandomForest: 86.38 %
SVC          : 87.72 %
```

3 Schlussfolgerung → bester Logarithmus

A) alle Features werden benutzt, man erhält hier die besten Werte

	accuracy	precision	recall	error
LogisticRegression	0.88	0.98	0.89	0.12
KNeighborsClassifier	0.87	0.99	0.87	0.13
NaiveBayes	0.83	0.89	0.90	0.17
RandomForestClassifier	0.86	0.99	0.87	0.14
SVC	0.88	0.99	0.88	0.12

4 Features sinnvoll mit PCA reduzieren und dann beim Unsupervised Learning bei einem Algorithmus einsetzen, damit man effektiv clustern kann

✗ Überprüfung mit ursprünglichem datensatz, wie viele components tatsächlich wichtig sind: → der Datensatz läßt sich sinnvollerweise auf 2 Features reduzieren, wie die Graphik zeigt:

```
# mit PCA auf die aussagekräftigen components reduzieren, um sie für kmeans clustering effektiv zu nutzen
# reduzieren der daten mit PCA:
from sklearn.decomposition import PCA
```

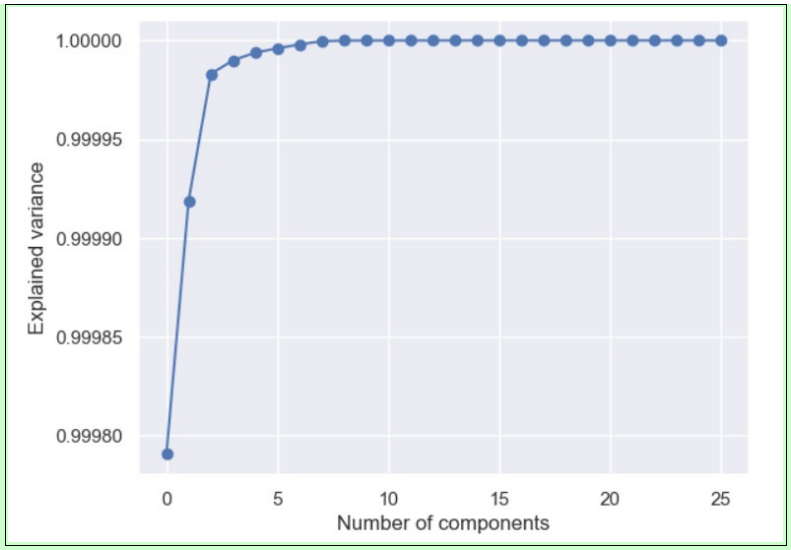
```
X02 = df01.drop(["Response","AcceptedCmp_Total"],axis=1)
```

```
X02.shape
```

```
(2240, 24)
```

```
# überprüfung mit ursprünglichem datensatz, wie viele components wichtig sind:
pca_check = PCA(n_components=26,random_state=33)
pca_check.fit(df01)
plt.plot(np.cumsum(pca_check.explained_variance_ratio_),marker="o")
plt.xlabel("Number of components")
plt.ylabel("Explained variance")
plt.show()
```

✗ auch die Berechnung bestätigt es: mit .explained_variance_ratio_ ermittle ich die Werte der notwenigen Varianz:




```
for i,value in enumerate(pca_check.explained_variance_ratio_):
    print(f"{i+1}. Principal Component erklärt {value*100:.4f}% der Varianz ")
```

```
1. Principal Component erklärt 99.9790% der Varianz
2. Principal Component erklärt 0.0128% der Varianz
3. Principal Component erklärt 0.0064% der Varianz
4. Principal Component erklärt 0.0007% der Varianz
5. Principal Component erklärt 0.0004% der Varianz
6. Principal Component erklärt 0.0002% der Varianz
```

✗ danach prepariere ich die reduzierten Features mit einem Standardscaler:

```
#standardscaler:
scaled_X02 = scaler.fit_transform(X02)
```

```
# 1 principal component --> initialisieren:
pca = PCA(n_components=2,random_state=33)
```

```
# trainieren und transformieren:
x_pca = pca.fit_transform(scaled_X02)
```

```
x_pca.shape
```

```
(2240, 2)
```

5 Mit einem Algorithmus des Unsupervised Learning erhält man ebenfalls gute Werte

A) gewählt wurde der KMeans_Clustering-Algorithmus

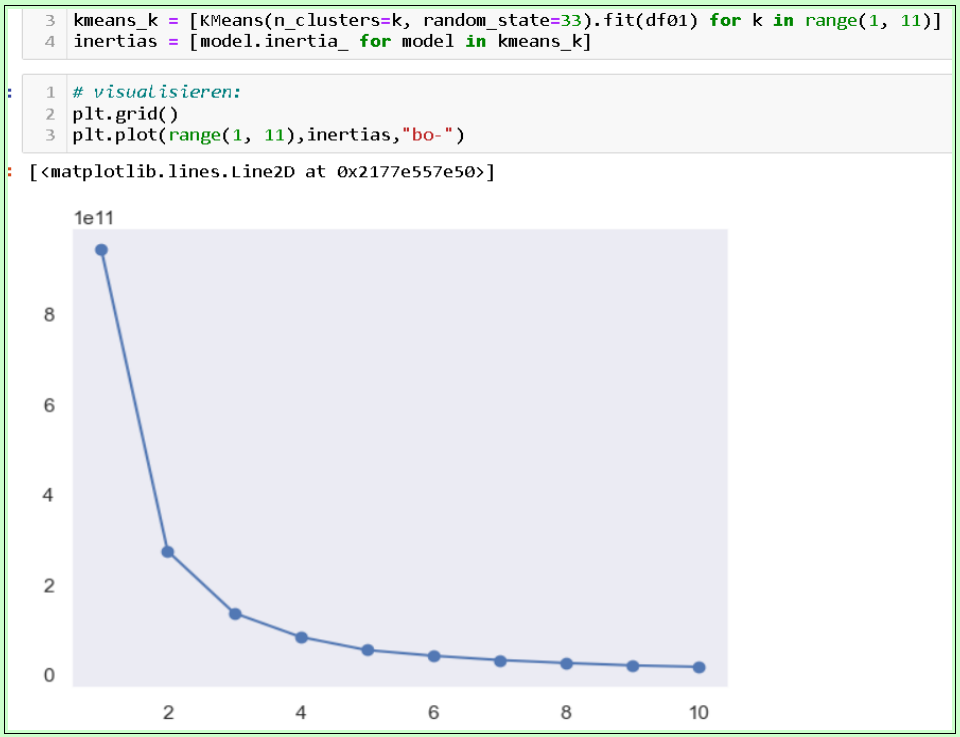
```
# k_mean importieren:
from sklearn.cluster import KMeans
```

B) normalerweise berechnet man die Anzahl von n_cluster über die .inertias_ (Distanzen bzw. Fehler). Die „2“ ist das günstigste Ergebnis für die Anzahl der Cluster.

```
kmeans_k = [KMeans(n_clusters=k, random_state=33).fit(df01) for k in range(1, 11)]
inertias = [model.inertia_ for model in kmeans_k]
```

```
# visualisieren:
plt.grid()
plt.plot(range(1, 11),inertias,"bo-")
```

```
[<matplotlib.lines.Line2D at 0x2177e557e50>]
```



C) Mit n_cluster = 2 trainiere ich den mit PCA reduzierten Datensatz und speichere die Daten in einem neuen Feature „Label“:

```
# k wählen: --> 2

# trainieren mit k von elbow-method gewählt 2:
km01 = KMeans(n_clusters=2,random_state=33)
pred_km01 = km01.fit_predict(x_pca)

# neues label mit den vorhergesagten clustern:
df01["Label"] = pred_km01
```

D) Das Feature zeigt ein sauberes Clustern an:

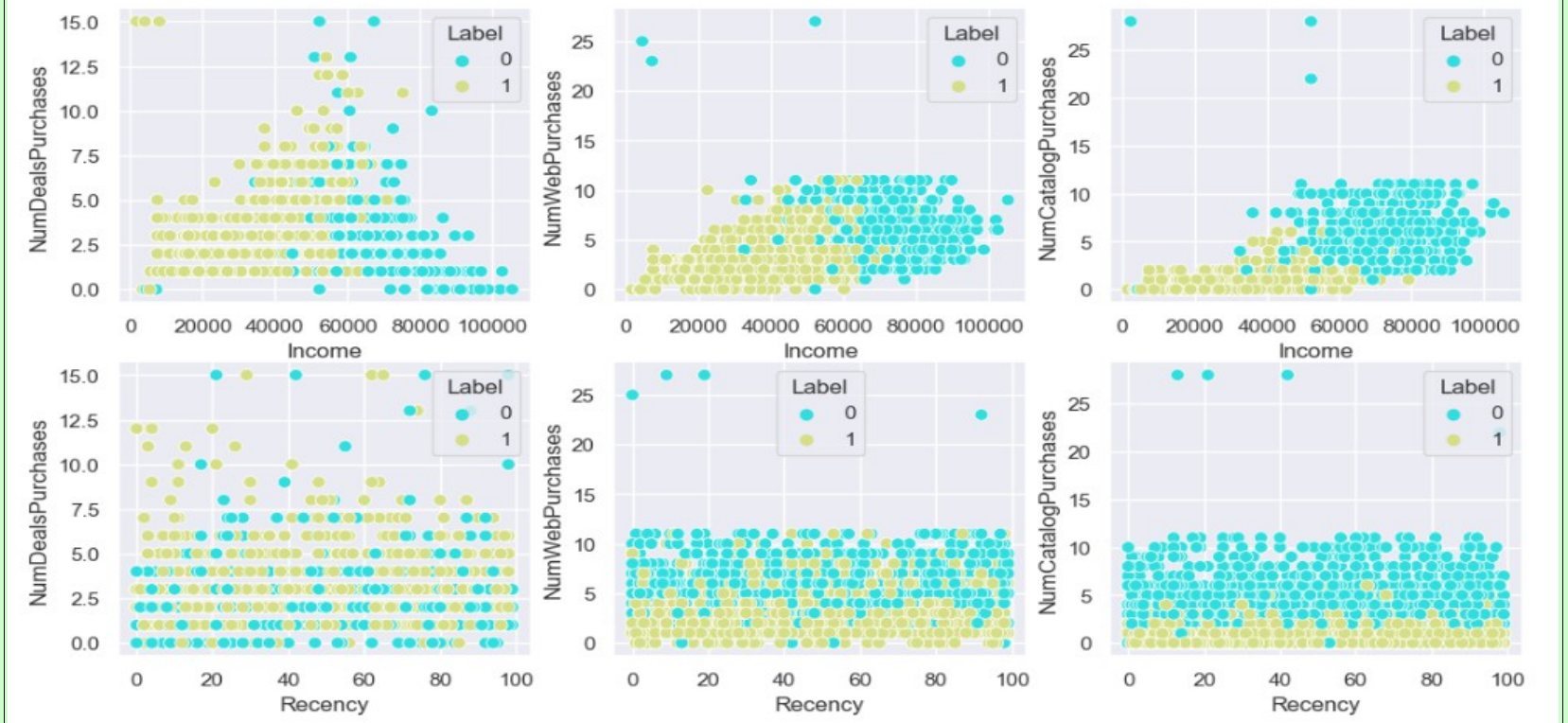
```
sns.scatterplot(x=x_pca[::,0],y=x_pca[::,1],hue=df01["Label"])
```

```
<Axes: >
```

E) auch bei mehreren Features kann man dies erkennen:

```
# erkannte cluster:
fig,axis = plt.subplots(2,3,figsize=(13,7))
sns.scatterplot(data=df01,x="Income",y="NumDealsPurchases",palette="rainbow",hue="Label",s=50,ax=axis[0,0])
sns.scatterplot(data=df01,x="Income",y="NumWebPurchases",palette="rainbow",hue="Label",s=50,ax=axis[0,1])
sns.scatterplot(data=df01,x="Income",y="NumCatalogPurchases",palette="rainbow",hue="Label",s=50,ax=axis[0,2])
sns.scatterplot(data=df01,x="Recency",y="NumDealsPurchases",palette="rainbow",hue="Label",s=50,ax=axis[1,0])
sns.scatterplot(data=df01,x="Recency",y="NumWebPurchases",palette="rainbow",hue="Label",s=50,ax=axis[1,1])
sns.scatterplot(data=df01,x="Recency",y="NumCatalogPurchases",palette="rainbow",hue="Label",s=50,ax=axis[1,2])
```

```
<Axes: xlabel='Recency', ylabel='NumCatalogPurchases'>
```



F) Schlussfolgerung: Man kann bei einigen Features deutlich erkennen, dass der Kmeans_Clustering-Algorithmus sinnvoll zur Anwendung kommen kann. Dies kann aber nur teilweise funktionieren, denn die einzigen Werte des Targets mit 0 und 1 sind nicht die beste Voraussetzung, um zu klassifizieren, da der Kmeans_Algorithmus besser mit mehreren Werten arbeitet.

. Gesamtfazit:

- überschaubare Anzahl an Features ließ eine gute Bearbeitung und Analyse des Datensatzes zu
  - NULL-Werte ließen sich gut ausgleichen
  - mehrere Label „Response“, aber auch 5 weitere Campaigns waren schon bereits vorhanden
- sehr gute Ergebnisse in Supervised Learning konnten mit allen Features erreicht werden
- beste Ergebnisse bezüglich der Wahl des richtigen Algorithmus:
  - Logistische Regression [→ 88.39 % Accuracy]
- minimal schlechtere Ergebnisse mit den weiteren Algorithmen
  - SVC
  - RandomForestClassifier
  - KneighborsClassifier
  - NaiveBayes
- Ergebnisse mit einem PCA reduzierten Datensatz und einem Kmeans_Clustering Algorithmus waren ebenfalls brauchbar und lieferten Ergebnisse mit 2 Clustern, ähnlich dem Label der einzelnen Kampagnen.

. Zukunft:

1. Vermarktung:
    - Man wird sich überlegen müssen, wie in Zukunft die Verteilung, inwieweit es sie noch geben wird, zwischen Online-Handel, Store-Handel aussehen wird und ob es einen Katalog-Handel überhaupt geben wird.
2. Campagnen sind noch lange nicht interaktiv voll genutzt, das kann man an den Werten sehen:
    - über Apps oder Ähnliches ist zwar die Möglichkeit vorhanden, den Kunden zu erreichen, aber das wachsende Angebot erzeugt eine schlechte Struktur, die man verbessern sollte.
    - Vernetzung von Apps untereinander oder in Plattformen zusammengefaßte Apps könnte ein Ansatz sein.