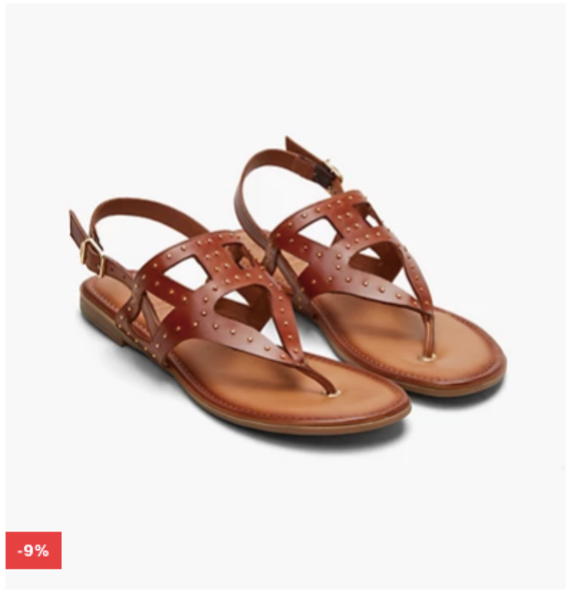
FALLSTUDIE 3

Web Scraping & Data Management

- Jumpsuits
- Bademode
- Unterwäsche
- Strick & Sweats
- Jacken & Mäntel
- Blazer & Westen
- Ponchos & Kimonos
- Schals, Mützen & Handschuhe
- **Schuhe**
- Taschen
- Socken
- Gürtel
- Portemonnaies
- Schmuck
- Sonnenbrillen
- Uhren

Sortierung: **Bestseller** | Größe | Farbe | Marke | Alle Filter



Leder-Sandalen mit Nieten-Besatz

~~49,95 EUR~~ **44,99 EUR**

Leder-Sandalen mit Nieten-Besatz

~~49,95 EUR~~ **44,99 EUR**

Lederslipper mit Quasten

~~59,95 EUR~~ **49,99 EUR**

gescrapte Inhalte

Elemente | Konsole | Quellen | Netzwerk | Timelines | Speicher | Grafiken | Layer | Prüfung

```
<span class="kk_labels o-tag m-buyBox__tag ta_promoText kk_asBox kk_tallsizes" data-short="ts">Tall Sizes</span>
</div>
<div class="c-containerOverlay__sectionItem ta_saleBadge">…</div>
</div>
</div>
</div>
</div>
<div class="c-productPanel__overlayBottom">…</div>
</div>
</div>
<div class="c-productPanel__bottom c-productPanel__bottom--noGrow">
<a href="/p/leder-sandalen-mit-nieten-besatz-101.101.103.28114.html" class="c-productPanel__link js-ovlistview-productdetaillink">
::before
<p class="o-copy o-copy--sizeS o-copy--responsive c-productPanel__name ta_shortdesc"> Leder-Sandalen mit Nieten-Besatz </p> = $0
</a>
<div class="c-productPanel__prices ta_price">
<span class="o-copy o-copy--sizeS o-copy--responsive c-productPanel__priceWas ta_oldPrice"> 49,95 EUR </span>
<span class="o-copy o-copy--sizeS o-copy--responsive o-copy--bold c-productPanel__priceSale ta_newPrice"> 44,99 EUR </span>
</div>
<div class="c-productPanel__colors">
<div class="c-optionList c-optionList--maxItems" data-ds-optionlist="{"maxItems":4,"maxItemsWithLabel":3,"breakpoints":{"s-up":{"maxItems":7,"maxItemsWithLabel":5}}
}" data-ds-mounted>
<div class="c-optionList__inner" data-ds-optionlist-inner data-ds-mounted>
```

Active | Focus | Hover | Visited | Errechnet | Layout | Schrift | Ändern

Stilattribut { }

@media (min-width: 768px)
.o-copy.o-copy--sizeS.o-copy--responsive { soliver-design-system.css:4:39884
font-size: .875rem;
line-height: 1.3125rem;
letter-spacing: .01875rem;
}

.o-copy.o-copy--sizeS.o-copy--responsive { soliver-design-system.css:4:39755
~~font-size: .75rem;~~
~~line-height: 1.125rem;~~
~~letter-spacing: .01875rem;~~
}

.o-copy.o-copy--sizeS { soliver-design-system.css:4:39668
~~font-size: .875rem;~~
~~line-height: 1.3125rem;~~

Box-Modell

Eigenschaften
-apple-color-filter: none;
-apple-trailing-word: auto;
-webkit-border-horizontal-spacing: 0px;
-webkit-border-vertical-spacing: 0px;
-webkit-box-direction: normal;
-webkit-cursor-visibility: auto;
-webkit-font-kerning: auto;
-webkit-font-smoothing: auto;
-webkit-hyphenate-character: auto;
-webkit-hyphenate-limit-after: auto;
-webkit-hyphenate-limit-before: auto;
-webkit-hyphenate-limit-lines: no-limit;
-webkit-hyphens: manual;
-webkit-line-align: none;
-webkit-line-box-contain: block inline repla
-webkit-line-grid: none;

Filter | Klassen | Filter | Autom.

```ipython3
%matplotlib inline
from IPython.display import IFrame

import requests
from bs4 import BeautifulSoup

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

import requests
from bs4 import BeautifulSoup
```

```ipython3
# Einlesen der URL
url = 'https://www.soliver.de/c/damen/sale/schuhe/'


# "requests.get() "holt alle Inhalte von der Website und gibt eine Antwort zurück
html = requests.get(url)
```

```ipython3
# Die Antwort = Response war erfolgreich:
# 200 = Erfolgreich; 400 = nicht erfolgreich
html
```

```
<Response [200]>
```

```ipython3
# soup = Dokument als verschachtelte Datenstruktur dar
# html.text = gibt uns nur den reinen Text aus
# Der Parser zerlegt die Zeichenkette, um diese Verarbeiten zu können
soup = BeautifulSoup(html.text, 'html.parser')
```

```ipython3
# durch den Parser haben wir ein BeautifulSoup-Objekt erhalten
print(soup.prettify())
```

```
Output exceeds the size limit. Open the full output data in a text editor
<!DOCTYPE html>
<html class="no-js" itemscope="" itemtype="http://schema.org/WebPage">
 <head prefix="og: http://ogp.me/ns#">
  <meta charset="utf-8"/>
  <meta content="IE=edge" http-equiv="X-UA-Compatible">
   <meta content="width=device-width,minimum-scale=1,maximum-scale=5,initial-scale=1" name="viewport"/>
   <meta content="Mit Schuhen von s.Oliver liegen Frauen voll im Trend. Schuhe in Top-Qualität gibt es im Online Shop von s.Oliver zu kaufen." name="description"/>
   <title>
    Damen Schuhe im Sale bei s.Oliver: Reduzierte Schuhe für Damen
   </title>
   <meta content="02:00:12 - 27.06.2022" name="robotshreflangtimestamp"/>
   <link href="https://www.soliver.de/c/damen/sale/schuhe/" hreflang="de-DE" rel="alternate"/>
   <link href="https://www.soliver.at/c/damen/sale/schuhe/" hreflang="de-AT" rel="alternate"/>
```

- Importieren der Bibliotheken und Methoden
- Wir lesen und die URL ein und prüfen mit request.get(), ob die Seite gescrappt werden kann -> Response 200 = scrapebar
- In soup haben wir nun unsere Datenstruktur, welche wir verarbeiten können

Scrapping des Inhalts

Der eigentlich wichtige Teil: Scrapen der Webseite, um Informationen über den Namen, den Originalpreis und den reduzierten Preis zu bekommen:

```
# mti find_all können wir die Datenstruktur nach bestimmten Tags und Klassen durchsuchen

originalpreis = soup.find_all('span', {'class': 'o-copy o-copy--sizeS o-copy--responsive c-productPanel__priceWas ta_oldPrice'})
```

✓ 0.7s Python

```
# wir erstellen eine Laufvariable a und lassen diese in soup.find.all durchlaufen.
# Jedes Mal wenn i diese Klasse findet, nimmt sie den Wert an und fügt den text in die originalliste hinzu
a =0
originalpreisliste= []
while a<5:
    originalpreis = soup.find_all("span", class_="o-copy o-copy--sizeS o-copy--responsive c-productPanel__priceWas ta_oldPrice") [a]

    # Diesen Schritt müssen wir machen, damit wir die Escape-Sequnez des Strings entfernen.
    originalpreis = originalpreis.text.replace ("\n", "")
    a = a+1
    originalpreisliste.append(originalpreis)
```

✓ 0.1s Python

```
originalpreisliste
```

✓ 0.5s Python

```
['49,95 EUR', '49,95 EUR', '59,95 EUR', '59,95 EUR', '59,95 EUR']
```

Das Scrappen nach .text hat auf dieser Seite nicht geklappt, deshalb habe ich, nachdem ich die class herausgefunden habe, die Inhalte direkt in eine Liste gespeichert und die Zeilenumbrüche entfernt.
Außerdem ist diese Methode ist übersichtlicher, kompakter und geht schneller

```
name = soup.find_all('p', {'class': 'o-copy o-copy--sizeS o-copy--responsive c-productPanel__name ta_shortdesc'})
```

✓ 0.2s

```
a =0
nameliste= []
while a<5:
    name = soup.find_all("p", class_="o-copy o-copy--sizeS o-copy--responsive c-productPanel__name ta_shortdesc") [a]
    name = name.text.replace ("\n", "")
    a = a+1
    nameliste.append(name)
```

✓ 0.1s

```
nameliste
```

✓ 0.3s

```
['Leder-Sandalen mit Nieten-Besatz',
 'Leder-Sandalen mit Nieten-Besatz',
 'Sandalen mit Keilabsatz',
 'Sandalen mit Keilabsatz',
 'Lederslipper mit Quasten']
```

```
aktuellerPreis = soup.find_all('span', {'class': 'o-copy o-copy--sizeS o-copy--responsive o-copy--bold c-productPanel__priceSale ta_newPrice'})
```

✓ 0.3s

```
a =0
aktuellePreiliste= []
while a<5:
    aktuellerPreis = soup.find_all("span", class_="o-copy o-copy--sizeS o-copy--responsive o-copy--bold c-productPanel__priceSale ta_newPrice") [a]
    aktuellerPreis = aktuellerPreis.text.replace ("\n", "")
    a = a+1
    aktuellePreiliste.append(aktuellerPreis)
```

✓ 0.1s

Wir scrappen nach dem gleichen Prinzip die Inhalte für den Namen der Schuhe und den aktuellen Preis

```python
# Wir erstellen ein DataFrame und eine CSV, damit unsere Kollgen weitere Analysen durchführen können
df = pd.DataFrame (
    {"originalpreis":originalpreisliste,
    "aktuellerpreis":aktuellePreiliste,
    "name":nameliste})
```

✓ 0.4s

```python
df
```

✓ 0.6s

	originalpreis	aktuellerpreis	name
0	49,95 EUR	44,99 EUR	Leder-Sandalen mit Nieten-Besatz
1	49,95 EUR	44,99 EUR	Leder-Sandalen mit Nieten-Besatz
2	59,95 EUR	54,99 EUR	Sandalen mit Keilabsatz
3	59,95 EUR	54,99 EUR	Sandalen mit Keilabsatz
4	59,95 EUR	49,99 EUR	Lederslipper mit Quasten

```python
df.to_csv("/Users/rafaela/test/Webscraping.csv")
```

✓ 0.5s

Erstellen eines Dataframes und einer CSV

```
import sqlalchemy as sa

engine = sa.create_engine('postgresql://postgres:461436@localhost:5432/postgres')
```

```
conda install -c conda-forge ipython-sql
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

```
%load_ext sql
```

```
%sql $engine.url
```

```
'Connected: postgres@postgres'
```

```
import pandas as pd
df = pd.read_csv("/Users/rafaela/test/webscraping.csv")
df.to_sql('webscraping', engine)
engine.table_names()
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:4: SADeprecationWarning: The Engine.table_names() method is deprecated and will be removed in a future release.  Please refer to
Inspector.get_table_names(). (deprecated since: 1.4)
  after removing the cwd from sys.path.

['webscraping']
```

```
%sql SELECT * FROM webscraping
```

Daten in Datenbank gespeichert

Predictive Analytics

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("https://raw.githubusercontent.com/kirenz/analytics/main/_static/data/commerce.csv")
pd.set_option('display.max_columns', None)
```

✓ 1.3s

Klassifiaktion: "Kauf" = Positive Klasse "Kein Kauf" = Negative Klasse

```python
# Als erstes schauen wir, ob die Datentypen in Ordnung sind oder ob man welche umändern/bereinigen muss
# In unserem Fall sind die Datentypen ok, da es sich nur um numerische Datentypen handelt

df.info()
```

✓ 0.1s

1.Schritt - Datenvorbereitung:

- Importieren der wichtigen Methoden
- CSV auslesen
- DataFrame analysieren

```python
# Wir kürzen DF so, dass wir nur die Variablen haben, welche unserer Meinung nach relevant sein könnten.
df_neu = df.loc[ : , [  "Kidhome", "Income", "Teenhome" ,"Recency" ,"Age", "Response"]]

#df_neu
```

✓ 0.1s

```python
# Response ist die abhängige Variable welche untersucht wird, um zu überprüfen, welche Kunden kaufen
# 1 = Kunden die kaufen
# 0 = Kunden die nicht kaufen
# Wir sehen jetzt schon, dass deutlich mehr Kunden nicht kaufen!

df_neu['Response'].value_counts()
```

✓ 0.5s

```
0    1872
1     333
Name: Response, dtype: int64
```

```python
# Datenvorbereiten für den Algorithmus:
# In X stehen alle unabhängigen Variablen
# in y steht die abhängige Variable
X = df_neu.drop(columns=['Response'])

y = df_neu["Response"]
```

✓ 0.5s

```python
# Train test split anwenden:
from sklearn.model_selection import train_test_split
```

✓ 0.6s

```python
# Traintest split aus X und y
# test_size = Testdaten 30% und Trainigsdaten = 70%
# random_state = Zufallszahlengenerator = generiert irgendwelche Zahlen
# aus X und y entstehen 4 verschiedene Variablen 2 Tests und 2 Training
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 1)
X_train
```

✓ 0.1s

```python
# Erstellung neuer Training-Datensatzes durch die Kopie des oberen Datensatzes:
train_dataset = pd.DataFrame(X_train).copy()
train_dataset
```

✓ 0.1s

	Kidhome	Income	Teenhome	Recency	Age
478	0	59432.0	1	88	58
669	0	69476.0	0	3	51
1009	0	68316.0	1	54	56
1138	0	79410.0	0	19	59
625	0	45906.0	1	20	56
...	...	...	...	...	...
960	1	46779.0	1	55	60
905	0	24401.0	0	98	41
1096	1	33569.0	0	10	37
235	0	65176.0	1	57	51
1061	1	36715.0	0	16	49

1543 rows × 5 columns

```python
# In die Kopie fügen wir nun unsere abhängige Variable ein:
# Trainingsdaten als Datenexplorationsobjekt: Nun könnenn wir herausfinden, welche Featurevariablen für unseren Test relevant sind und welche nicht

train_dataset['Response'] = pd.DataFrame(y_train)
train_dataset
```

✓ 0.6s

	Kidhome	Income	Teenhome	Recency	Age	Response
478	0	59432.0	1	88	58	0
669	0	69476.0	0	3	51	0
1009	0	68316.0	1	54	56	0
1138	0	79410.0	0	19	59	0
625	0	45906.0	1	20	56	0
...	...	...	...	...	...	...
960	1	46779.0	1	55	60	0
905	0	24401.0	0	98	41	0
1096	1	33569.0	0	10	37	1
235	0	65176.0	1	57	51	0
1061	1	36715.0	0	16	49	1

1543 rows × 6 columns

Exploratory data analysis (EDA)

```
# wir gruppieren nach unserer Zielvariable
# .T transpondiert den DF -> Übersichtlicher

describetraindataset = train_dataset.groupby(by=['Response']).describe().T
```

✓ 0.4s

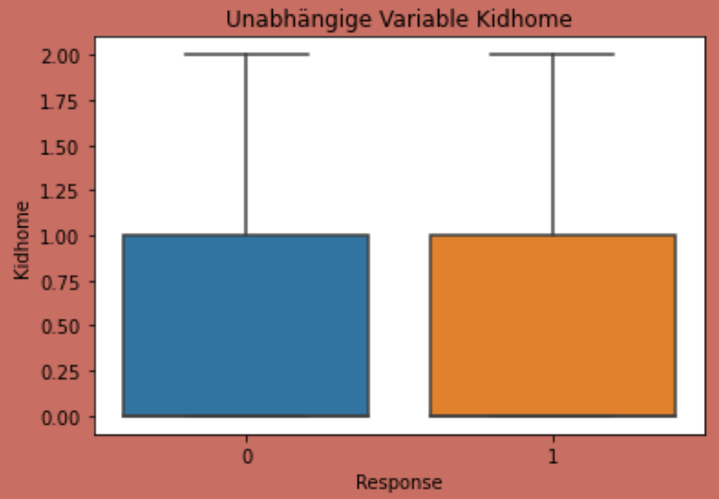
2. Schritt: Datenexploration

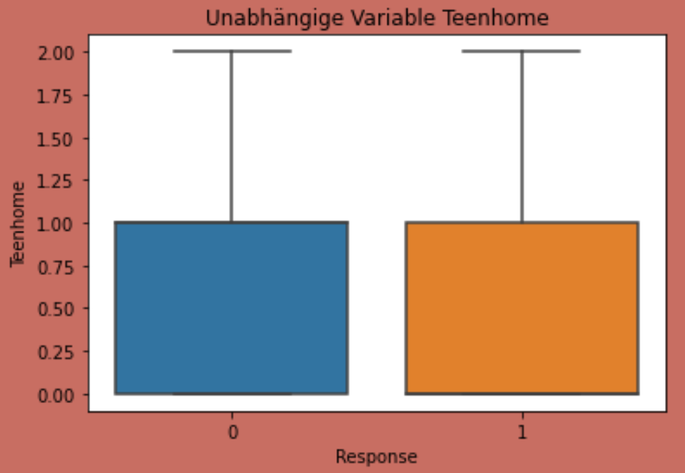
```
df_neu.corr().round(2)
```

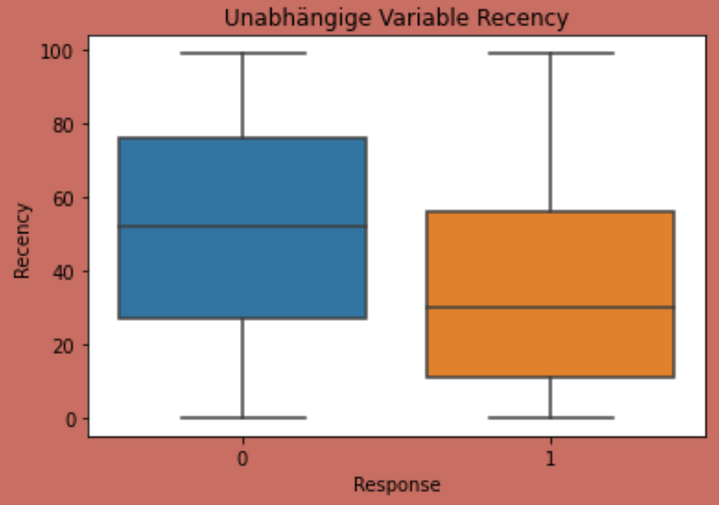
✓ 0.1s

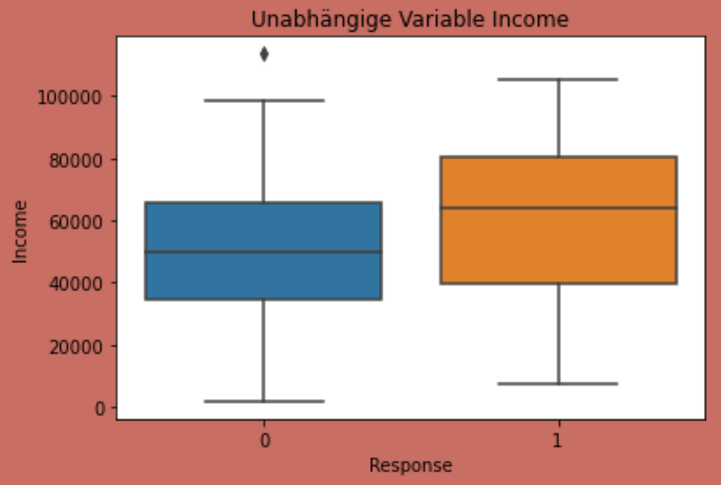
	Kidhome	Income	Teenhome	Recency	Age	Response
Kidhome	1.00	-0.53	-0.04	0.01	-0.24	-0.08
Income	-0.53	1.00	0.04	0.01	0.21	0.17
Teenhome	-0.04	0.04	1.00	0.01	0.36	-0.16
Recency	0.01	0.01	0.01	1.00	0.01	-0.20
Age	-0.24	0.21	0.36	0.01	1.00	-0.02
Response	-0.08	0.17	-0.16	-0.20	-0.02	1.00

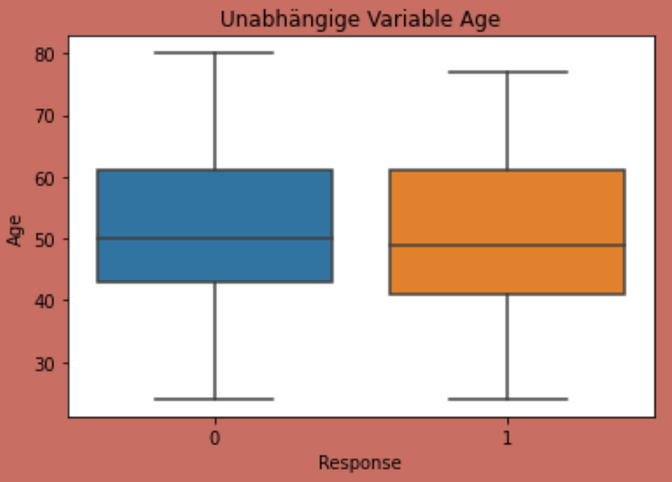
Anhand der Korrelation kann man erkennen, dass außer die Variable "Income", keine andere Variable eine positive Beziehung zur abhängigen Variable "Response" hat.


Unabhängige Variable Kidhome
Kidhome
Response
0
1
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Unabhängige Variable Teenhome
Teenhome
Response
0
1
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Unabhängige Variable Recency
Recency
Response
0
1
0
20
40
60
80
100


Unabhängige Variable Income
Income
Response
0
1
0
20000
40000
60000
80000
100000


Unabhängige Variable Age
Age
Response
0
1
30
40
50
60
70
80

Erkenntnisse aus Boxplots:

- Systematische Unterschiede in den Variablen "Income" und "Recency"
- Keine systematischen Unterschiede in den anderen unabhängigen Variablen

Regressionsmodell:

```
# Liste mit den unabhängigen Variralben welche ich in mein Modell aufnehme
features_model = ['Recency', 'Income', 'Age']
```

✓ 0.6s

```
# Eingrenzung des Featuremodells in einem neuen Trainingsmodell
X_train =  X_train[features_model]
X_test = X_test[features_model]
```

✓ 0.6s

```
# Regressionsmodell mit LogisticRegression
from sklearn.linear_model import LogisticRegression

#Erzeugung des Modells
clf = LogisticRegression()
```

✓ 0.2s

```
# Anwednunge des Modells auf die Trainingsdaten

clf.fit (X_train, y_train)
```

✓ 0.1s

```
LogisticRegression()
```

```
# Vorhersage

y_pred = clf.predict(X_test)

y_pred
```

✓ 0.8s

```
Output exceeds the size limit. Open the full output data in a text editor
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

3. Schritt
- Regressionsmodell:
Features anhand der EDA ausgesucht

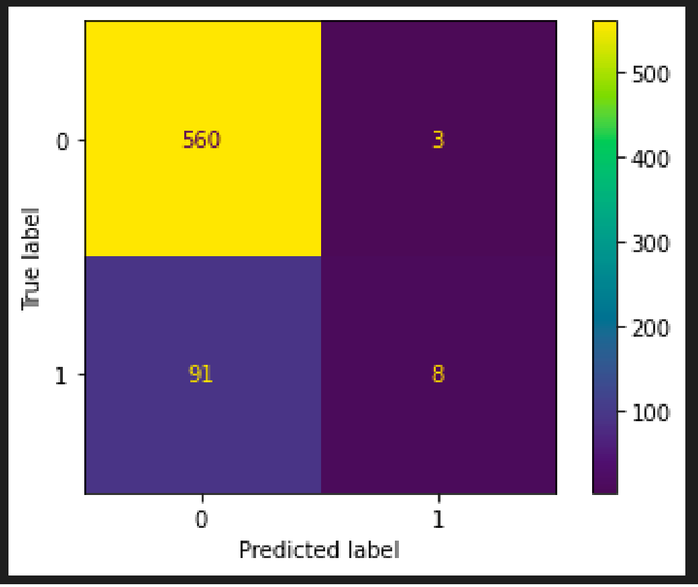
```
# Genauigkeit des Modells = Accuracy
clf.score(X_test, y_test)
```
✓ 0.9s

```
0.8580060422960725
```

```
from sklearn.metrics import ConfusionMatrixDisplay
```
✓ 0.4s

```
ConfusionMatrixDisplay.from_estimator(clf,X_test,y_test)
```
✓ 0.2s

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f8103667b10>
```



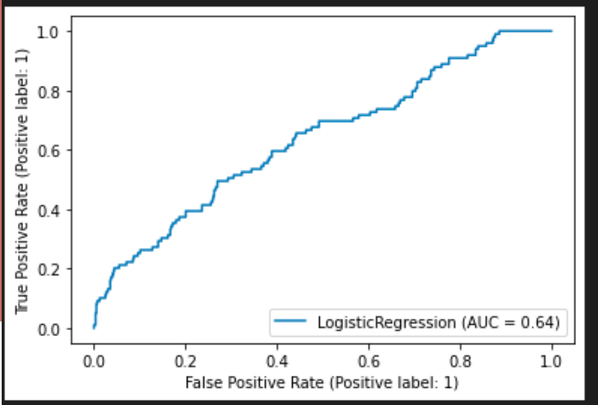

```
print (classification_report(y_test, y_pred))
```
✓ 0.1s

```
              precision    recall  f1-score   support

           0       0.86      0.99      0.92       563
           1       0.73      0.08      0.15        99

    accuracy                           0.86       662
   macro avg       0.79      0.54      0.53       662
weighted avg       0.84      0.86      0.81       662
```

```
Modell liefert uns sehr genaue Ergebnisse! -> Wir haben ein ideales System
```

```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(clf, X_test, y_test)
```
✓ 0.2s

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f810162d490>
```




```
from sklearn.metrics import roc_auc_score

y_score = clf.predict_proba(X_test)[:, 1]
roc_auc_score(y_test, y_score)
```
✓ 0.1s

```
0.6370723217970109
```

Zusammenfassung

- Unabhängig vom Income der Kunden kaufen wenige die Produkte
- Unser Modell ist sehr genau und sagt voraus, dass die Kampagne auch nicht mehr Käufe generieren wird.