

```
In [3]: # Installiere python packages falls noch nicht installiert
!pip install pandas numpy matplotlib scipy sqlalchemy tabulate scikit-learn

Requirement already satisfied: pandas in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (2.2.3)
Requirement already satisfied: numpy in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (2.2.2)
Requirement already satisfied: matplotlib in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (3.10.0)
Requirement already satisfied: scipy in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (1.15.1)
Requirement already satisfied: sqlalchemy in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (2.0.37)
Requirement already satisfied: tabulate in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (0.9.0)
Requirement already satisfied: scikit-learn in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (1.6.1)
Requirement already satisfied: seaborn in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (0.13.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from pandas) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from matplotlib) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from matplotlib) (3.2.1)
Requirement already satisfied: greenlet!=0.4.17 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from sqlalchemy) (3.1.1)
Requirement already satisfied: typing-extensions>=4.6.0 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from sqlalchemy) (4.12.2)
Requirement already satisfied: joblib>=1.2.0 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in /home/jan-david/Documents/engeneer/.venv/lib/python3.12/site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)

[notice] A new release of pip is available: 23.2.1 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
```

```
In [3]: #imports
import os
```

```

import sys
import platform
import zipfile
import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats as stats
import matplotlib.pyplot as plt
import itertools
import random

from collections import Counter
from sqlalchemy import func, Integer, case, distinct, and_
from tabulate import tabulate
from scipy.stats import chi2_contingency
from IPython.display import HTML, display, Markdown
from matplotlib import colormaps
from matplotlib.patches import Patch
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_curve, auc
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
from sqlalchemy import func

if not os.path.exists("./DABI1.db"):
    if platform.system() == "Windows":
        zip_folder = os.path.abspath(os.path.join(os.getcwd(), "../../Data"))

        for file_name in os.listdir(zip_folder):
            if file_name.endswith(".zip"):
                zip_path = os.path.join(zip_folder, file_name)
                with zipfile.ZipFile(zip_path, 'r') as zip_ref:
                    zip_ref.extractall(zip_folder)
                print(f"Entpackt: {zip_path}")

    # dauert etwa 10 - 20 Minuten
    os.system(f"python {os.path.abspath('../scripts/load_data_into_db.py')}")
else:
    # UNIX/Linux/macOS-spezifischer Code
    os.system("sh ../scripts/unzip.sh")
    # dauert etwa 10 - 15 Minuten
    os.system("python ../scripts/load_data_into_db.py")

notebook_dir = os.getcwd()
module_path = os.path.abspath(os.path.join(notebook_dir, '..'))
if module_path not in sys.path:
    sys.path.append(module_path)

from utils.db import get_engine, get_session
from custom_types.db_models import Order, Product, Einkaufskorb, Departme
engine = get_engine(path = "sqlite:///DABI1.db")
session = get_session(engine)

```

```
abbildung = 1
```

Verbindung erfolgreich



Datenanalyse des Instacarts-Datensatzes

für die Marketing- und Dispositionsabteilung



Projektteam

Name	Matrikelnummer
Benjamin Saur	71254
Vladimir Dvornik	80542
Jan-David Wiederstein	88219



Inhaltsverzeichnis

1. [EDA Marketing](#)
 2. [EDA Disposition](#)
 3. [Konkrete Fragen](#)
 - A. [Frage 1](#)
 - B. [Frage 2](#)
 - C. [Frage 3](#)
 4. [Handlungsempfehlungen](#)
-

EDA Marketing

```
In [5]: # Abfrage: Produkte mit ihrer Reihenfolge innerhalb einer Bestellung
result = session.query(
    Einkaufskorb.add_to_cart_order,
    Product.product_name,
    Einkaufskorb.order_id
).join(Product).order_by(Einkaufskorb.order_id, Einkaufskorb.add_to_cart_)

df = pd.DataFrame(result, columns=["add_to_cart_order", "product_name", "o

# Produkte, die häufig am Anfang (add_to_cart_order = 1) stehen
first_products = df[df['add_to_cart_order'] == 1]['product_name'].value_c
```

```
last_products_per_order = df.loc[df.groupby('order_id')['add_to_cart_order'] > 0].groupby('order_id').size().reset_index(name='count')
last_product_counts = last_products_per_order['product_name'].value_counts()

print("Produkte am Anfang:")
print(first_products.head(10))
print("\nProdukte am Ende:")
print(last_product_counts.head(10))
```

Produkte am Anfang:

product_name	count
Banana	57776
Bag of Organic Bananas	41701
Organic Whole Milk	15981
Organic Strawberries	14552
Organic Hass Avocado	12658
Organic Baby Spinach	12142
Organic Avocado	11619
Spring Water	8914
Strawberries	8398
Organic Raspberries	7430

Name: count, dtype: int64

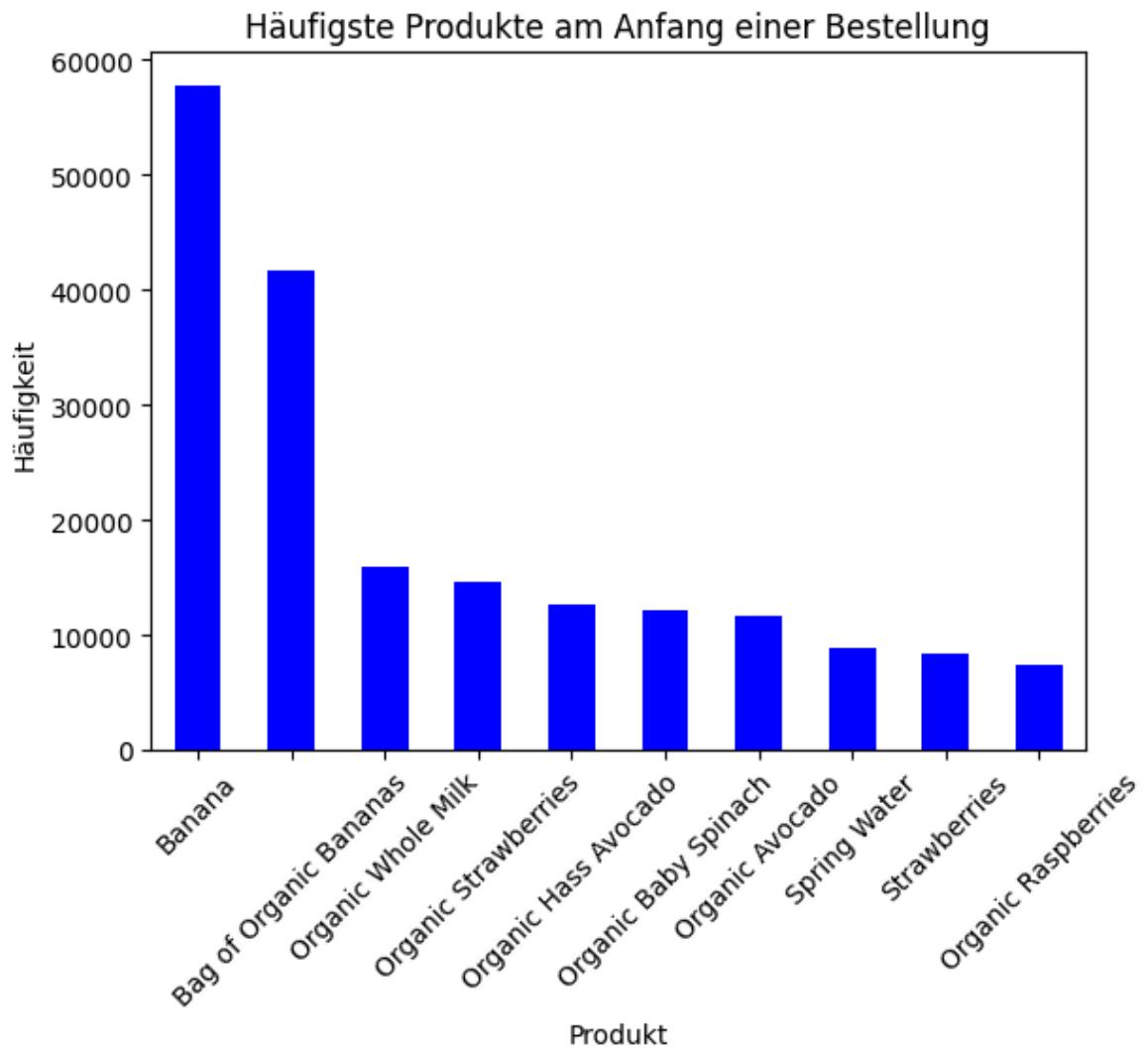
Produkte am Ende:

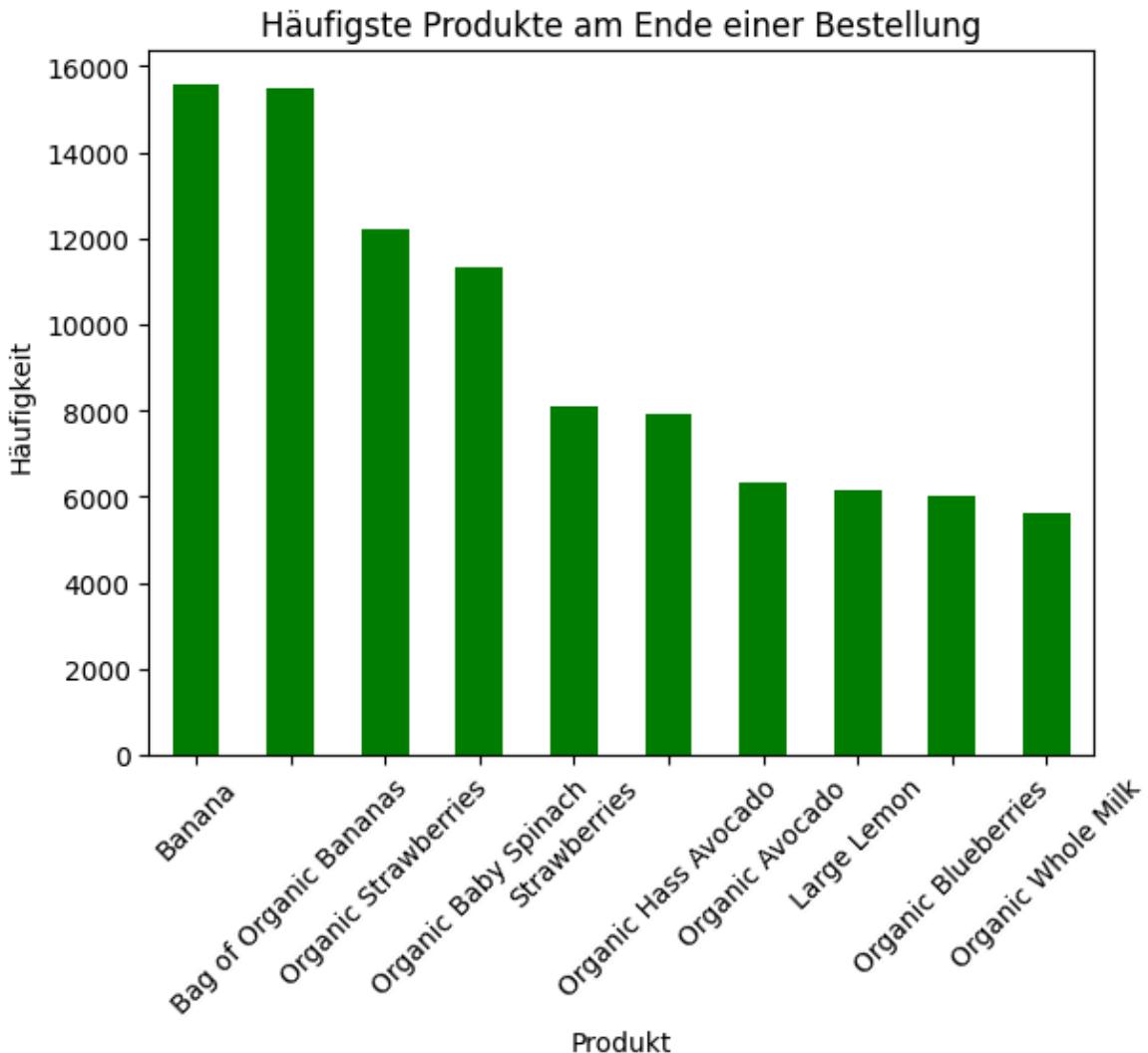
product_name	count
Banana	15573
Bag of Organic Bananas	15497
Organic Strawberries	12199
Organic Baby Spinach	11311
Strawberries	8079
Organic Hass Avocado	7926
Organic Avocado	6319
Large Lemon	6165
Organic Blueberries	6023
Organic Whole Milk	5611

Name: count, dtype: int64

```
In [6]: # Häufigste Produkte am Anfang
first_products.head(10).plot(kind='bar', title='Häufigste Produkte am Anfang')
plt.ylabel('Häufigkeit')
plt.xlabel('Produkt')
plt.xticks(rotation=45)
plt.show()

# Häufigste Produkte am Ende
last_product_counts.head(10).plot(kind='bar', title='Häufigste Produkte am Ende')
plt.ylabel('Häufigkeit')
plt.xlabel('Produkt')
plt.xticks(rotation=45)
plt.show()
```





```
In [7]: result = session.query(
    Einkaufskorb.order_id,
    Product.product_name
).join(Product).order_by(Einkaufskorb.order_id).limit(500000).all()

df = pd.DataFrame(result, columns=["order_id", "product_name"])
```

```
In [8]: # Gruppiere Produkte nach Bestellung
order_groups = df.groupby("order_id")["product_name"].apply(list)

# Finde alle Paar-Kombinationen innerhalb jeder Bestellung
product_combinations = order_groups.apply(lambda products: list(itertools

# Alle Kombinationen in eine flache Liste umwandeln
flat_combinations = [pair for pairs in product_combinations for pair in p

# Häufigkeit jeder Kombination zählen
combination_counts = Counter(flat_combinations)

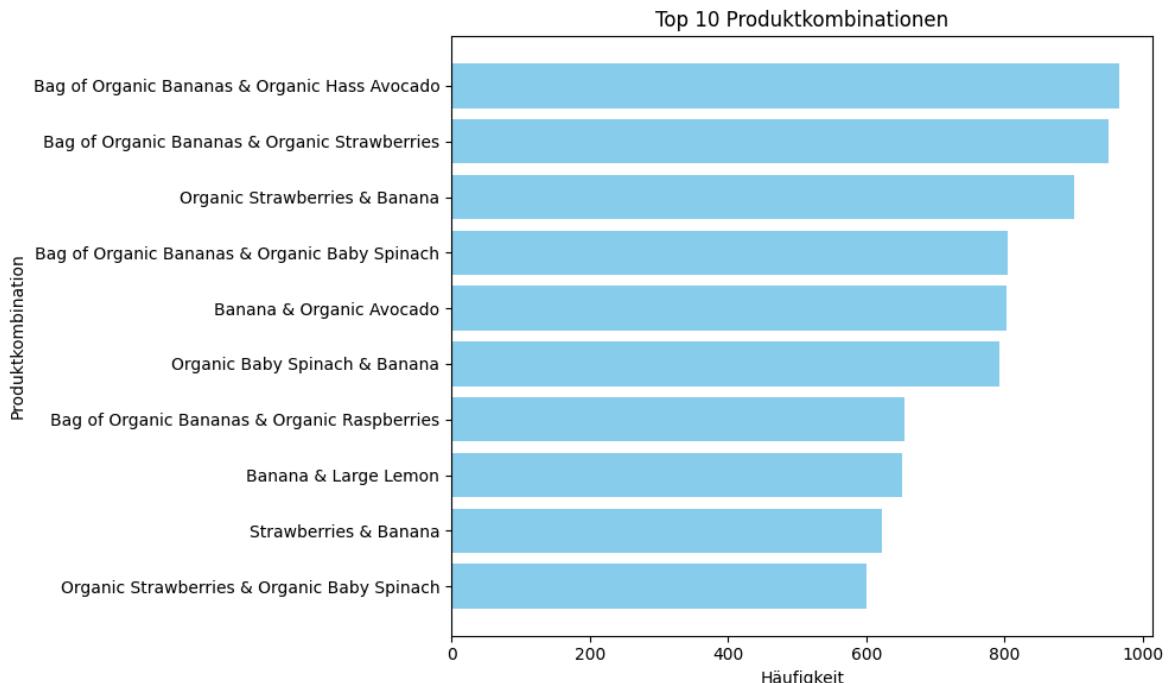
# Die Top-10-Kombinationen anzeigen
top_combinations = combination_counts.most_common(10)
print("Top 10 Produktkombinationen:")
for combo, count in top_combinations:
    print(f"{combo}: {count}")
```

Top 10 Produktkombinationen:

```
('Bag of Organic Bananas', 'Organic Hass Avocado'): 966
('Bag of Organic Bananas', 'Organic Strawberries'): 950
('Organic Strawberries', 'Banana'): 900
('Bag of Organic Bananas', 'Organic Baby Spinach'): 804
('Banana', 'Organic Avocado'): 803
('Organic Baby Spinach', 'Banana'): 792
('Bag of Organic Bananas', 'Organic Raspberries'): 655
('Banana', 'Large Lemon'): 651
('Strawberries', 'Banana'): 622
('Organic Strawberries', 'Organic Baby Spinach'): 600
```

```
In [9]: combos, counts = zip(*top_combinations)

plt.figure(figsize=(10, 6))
plt.barh([f"{combo[0]} & {combo[1]}" for combo in combos], counts, color="blue")
plt.title("Top 10 Produktkombinationen")
plt.xlabel("Häufigkeit")
plt.ylabel("Produktkombination")
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```



```
In [10]: result = session.query(
    Einkaufskorb.order_id,
    Department.department_name
).join(Product, Einkaufskorb.product_id == Product.product_id) \
.join(Department, Product.department_id == Department.department_id) \
.distinct() # Entfernt Duplikate innerhalb derselben Bestellung

f = pd.DataFrame(result, columns=["order_id", "department"])
```

```
In [11]: grouped = f.groupby('order_id')['department'].apply(set)

# Schritt 2: Generiere paarweise Kombinationen von Departments
department_pairs = grouped.apply(lambda x: list(itertools.combinations(x,

# Schritt 3: Flache Liste aller Paare erstellen
```

```

flat_pairs = [pair for pairs in department_pairs for pair in pairs]

# Schritt 4: Häufigkeit der Co-Käufe zählen
pair_counts = Counter(flat_pairs)

# Schritt 5: Ergebnisse als DataFrame darstellen
pair_counts_df = pd.DataFrame(pair_counts.items(), columns=['department_p
# Ergebnisse anzeigen
print(pair_counts_df.head(10))

```

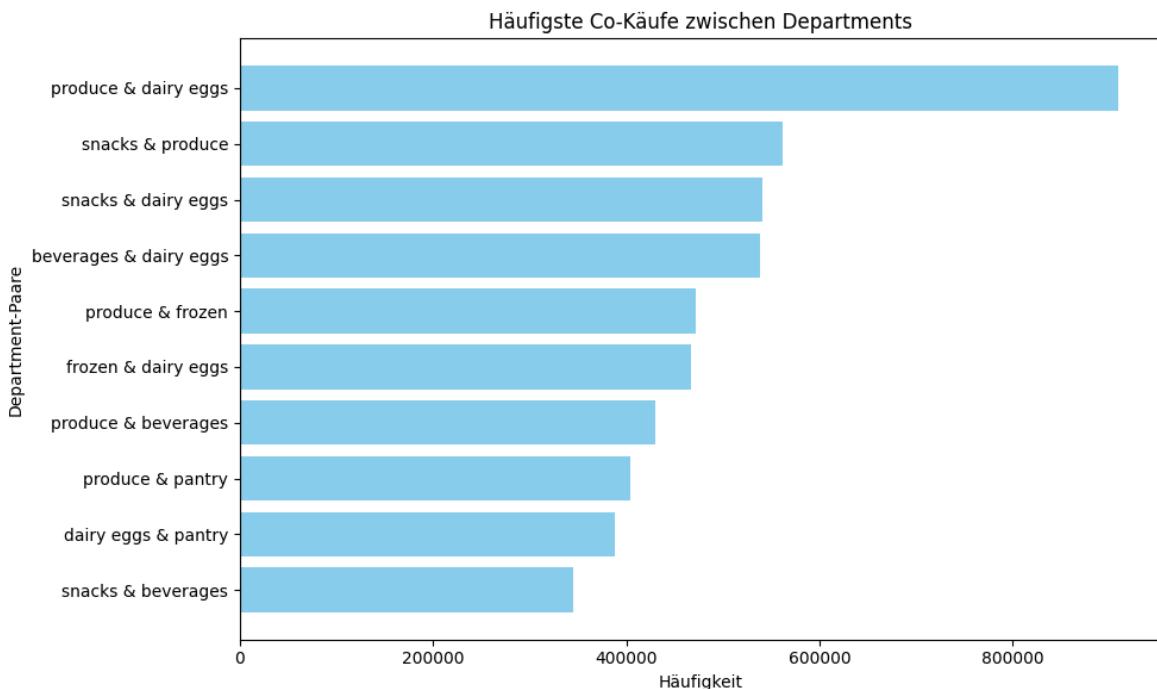
	department_pair	count
2	(produce, dairy eggs)	909105
19	(snacks, produce)	562081
20	(snacks, dairy eggs)	540482
57	(beverages, dairy eggs)	537715
103	(produce, frozen)	472014
110	(frozen, dairy eggs)	466914
53	(produce, beverages)	429962
56	(produce, pantry)	404552
62	(dairy eggs, pantry)	387856
10	(snacks, beverages)	344402

In [12]: top_pairs = pair_counts_df.head(10)

```

plt.figure(figsize=(10, 6))
plt.barh([f'{pair[0]} & {pair[1]}' for pair in top_pairs['department_pair']],
plt.xlabel('Häufigkeit')
plt.ylabel('Department-Paare')
plt.title('Häufigste Co-Käufe zwischen Departments')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()

```



In [13]: result = session.query(

```

Order.user_id,
Einkaufskorb.order_id,
Einkaufskorb.product_id,
Product.product_name

```

```

).join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id) \
.join(Product, Einkaufskorb.product_id == Product.product_id) \
.distinct().limit(100000)

df = pd.DataFrame(result, columns=["user_id", "order_id", "product_id", ""]

```

```

In [14]: from sqlalchemy import func

# Gesamtzahl der Bio-Produkte
total_bio_products = session.query(func.count(func.distinct(Product.produ
    .filter(Product.product_name.ilike('%organic%')) \
    .scalar()

# Gesamtzahl der Produkte im Sortiment
total_products = session.query(func.count(func.distinct(Product.product_i

# Berechnung des Anteils der Bio-Produkte an allen Produkten
bio_percentage = total_bio_products / total_products if total_products >

bio_percentage

```

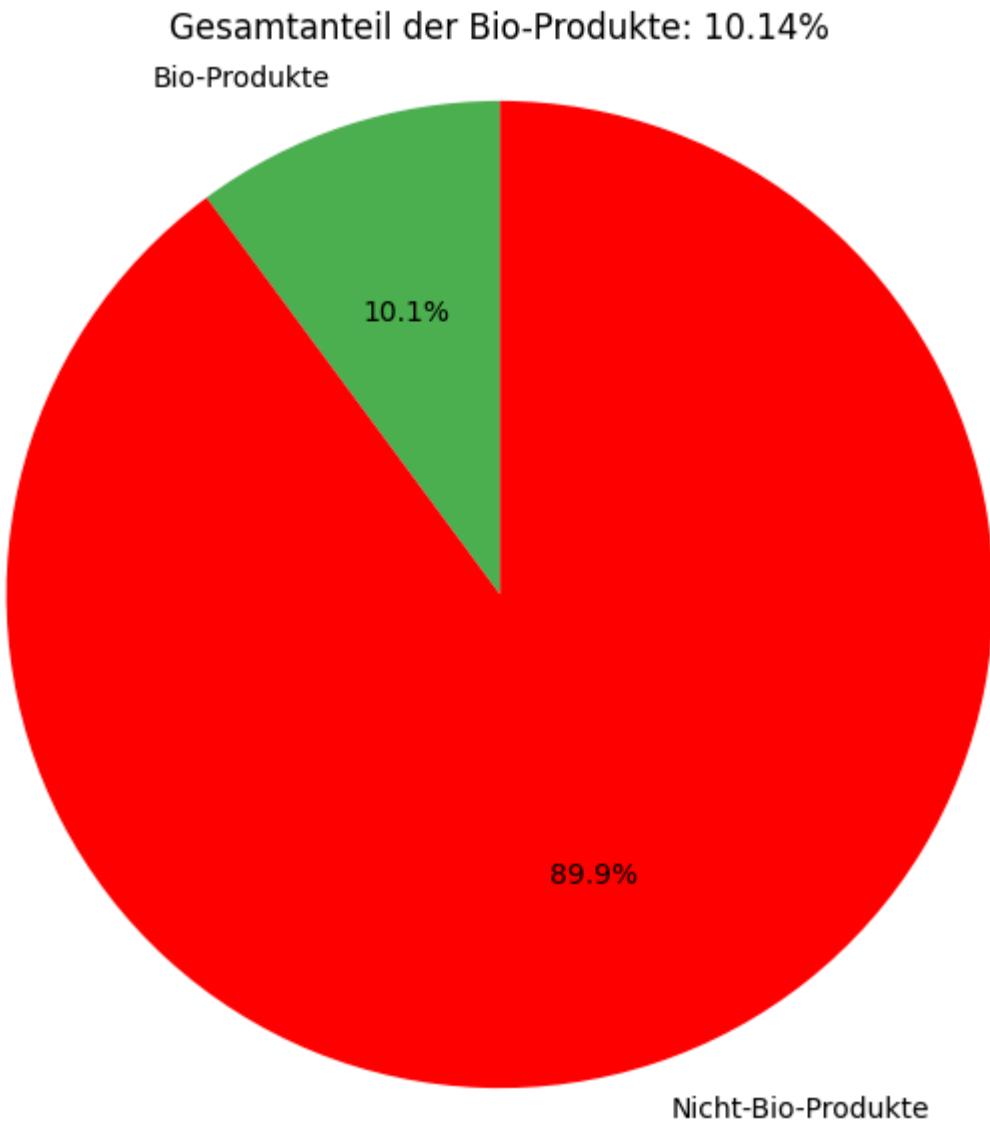
Out[14]: 0.1013524392207374

```

In [16]: labels = 'Bio-Produkte', 'Nicht-Bio-Produkte'
sizes = [bio_percentage, 1 - bio_percentage]
colors = ['#4CAF50', '#FF0000'] # Grün für Bio-Produkte, Rot für Nicht-B

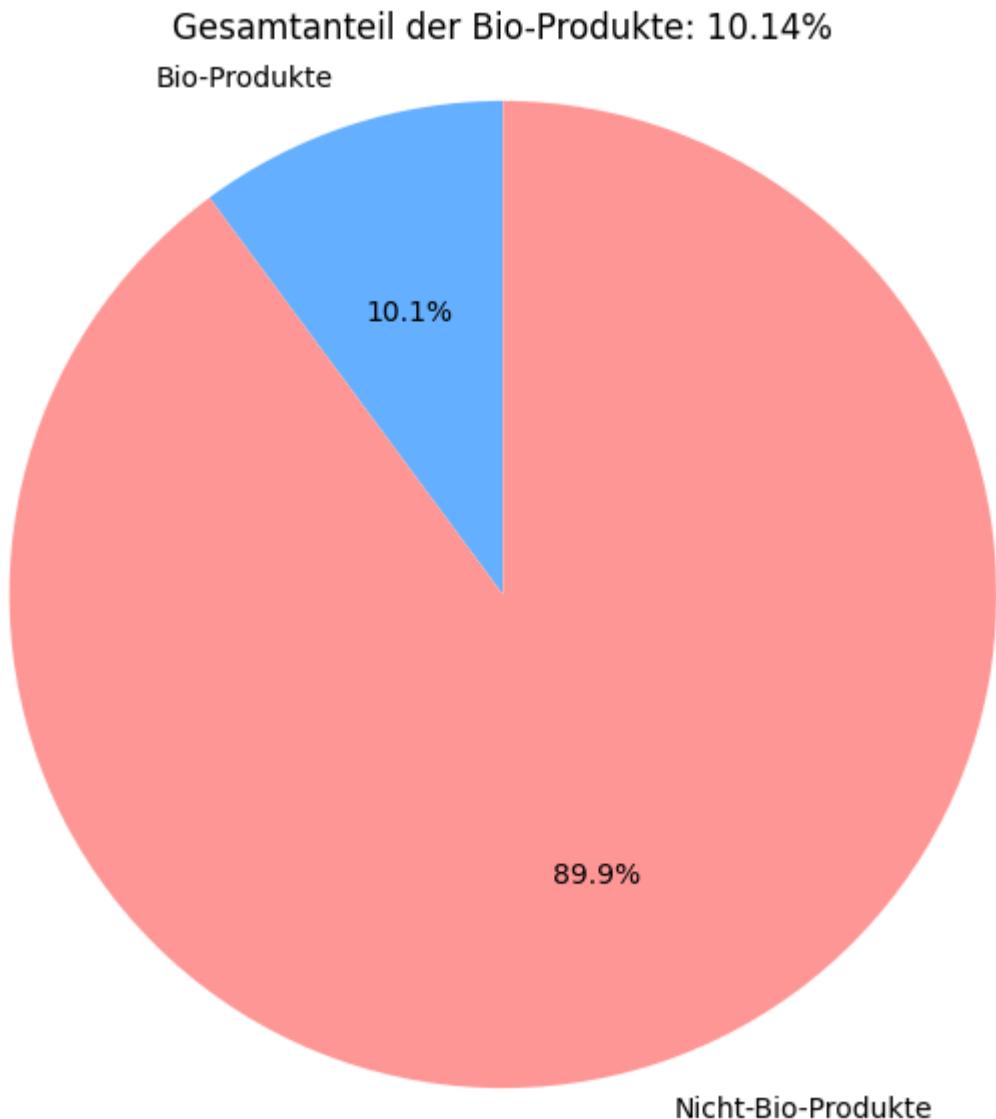
plt.figure(figsize=(7, 7))
plt.pie(sizes, labels=labels, colors=colors, autopct='%.1f%%', startangl
# plt.title(f'Gesamtanteil der Bio-Produkte: {gesamt_bio_anteil*100:.2f}%')
plt.title(f'Gesamtanteil der Bio-Produkte: {bio_percentage*100:.2f}%')
plt.axis('equal') # Kreis wird als Kreis angezeigt
plt.show()

```



```
In [19]: # Visualisierung des Gesamtanteils der Bio-Produkte (Kreisdiagramm)
gesamt_bio_anteil = bio_percentage # new
labels = 'Bio-Produkte', 'Nicht-Bio-Produkte'
sizes = [gesamt_bio_anteil, 1 - bio_percentage]
colors = ['#66b3ff', '#ff9999']
plt.figure(figsize=(7, 7))
plt.pie(sizes, labels=labels, colors=colors, autopct='%.1f%%', startangle=90)
plt.title(f'Gesamtanteil der Bio-Produkte: {gesamt_bio_anteil*100:.2f}%')
plt.axis('equal') # Kreis
```

```
Out[19]: (np.float64(-1.099999777803624),
           np.float64(1.0999997941924113),
           np.float64(-1.0999999704709424),
           np.float64(1.0999999985938544))
```



```
In [20]: # Berechnung der Anzahl der Produkte pro Department
department_product_count = session.query(
    Department.department_name,
    func.count(func.distinct(Product.product_id)).label('product_count')
) \
    .join(Product, Product.department_id == Department.department_id) \
    .group_by(Department.department_name) \
    .all()

# Ausgabe der Ergebnisse
for department in department_product_count:
    print(f"Department: {department.department_name}, Number of Products: {department.product_count}")
```

```
Department: alcohol, Number of Products: 1054
Department: babies, Number of Products: 1081
Department: bakery, Number of Products: 1516
Department: beverages, Number of Products: 4365
Department: breakfast, Number of Products: 1115
Department: bulk, Number of Products: 38
Department: canned goods, Number of Products: 2092
Department: dairy eggs, Number of Products: 3449
Department: deli, Number of Products: 1322
Department: dry goods pasta, Number of Products: 1858
Department: frozen, Number of Products: 4007
Department: household, Number of Products: 3085
Department: international, Number of Products: 1139
Department: meat seafood, Number of Products: 907
Department: missing, Number of Products: 1258
Department: other, Number of Products: 548
Department: pantry, Number of Products: 5371
Department: personal care, Number of Products: 6563
Department: pets, Number of Products: 972
Department: produce, Number of Products: 1684
Department: snacks, Number of Products: 6264
```

```
In [21]: # Berechnung der Anzahl der Bio-Produkte pro Department
department_bio_product_count = session.query(
    Department.department_name,
    func.count(func.distinct(Product.product_id)).label('bio_product_count')
) \
    .join(Product, Product.department_id == Department.department_id) \
    .filter(Product.product_name.ilike('%organic%')) \
    .group_by(Department.department_name) \
    .all()

# Ausgabe der Ergebnisse
for department in department_bio_product_count:
    print(f"Department: {department.department_name}, Bio Products: {depa
```

```
Department: alcohol, Bio Products: 15
Department: babies, Bio Products: 260
Department: bakery, Bio Products: 116
Department: beverages, Bio Products: 624
Department: breakfast, Bio Products: 155
Department: bulk, Bio Products: 22
Department: canned goods, Bio Products: 320
Department: dairy eggs, Bio Products: 435
Department: deli, Bio Products: 100
Department: dry goods pasta, Bio Products: 300
Department: frozen, Bio Products: 252
Department: household, Bio Products: 9
Department: international, Bio Products: 84
Department: meat seafood, Bio Products: 47
Department: missing, Bio Products: 175
Department: other, Bio Products: 37
Department: pantry, Bio Products: 746
Department: personal care, Bio Products: 187
Department: pets, Bio Products: 23
Department: produce, Bio Products: 473
Department: snacks, Bio Products: 656
```

```
In [24]: # Berechnung des Anteils der Bio-Produkte pro Department
department_bio_percentage = []
```

```

# Gehe durch jedes Department und berechne den Anteil
for dept in department_bio_product_count:
    for dept_total in department_product_count:
        if dept.department_name == dept_total.department_name:
            bio_percentage = dept.bio_product_count / dept_total.product_
            department_bio_percentage.append({
                "department_name": dept.department_name,
                "bio_products_count": dept.bio_product_count,
                "total_products_count": dept_total.product_count,
                "bio_percentage": bio_percentage
            })

# Ausgabe der Ergebnisse
for result in department_bio_percentage:
    print(f"Department: {result['department_name']}, Bio Products: {result['bio_products_count']}, Total Products: {result['total_products_count']}, Bio Product Percentage: {result['bio_percentage'] * 100:.2f}%")

```

Department: alcohol, Bio Products: 15, Total Products: 1054, Bio Products Percentage: 1.42%

Department: babies, Bio Products: 260, Total Products: 1081, Bio Products Percentage: 24.05%

Department: bakery, Bio Products: 116, Total Products: 1516, Bio Products Percentage: 7.65%

Department: beverages, Bio Products: 624, Total Products: 4365, Bio Products Percentage: 14.30%

Department: breakfast, Bio Products: 155, Total Products: 1115, Bio Products Percentage: 13.90%

Department: bulk, Bio Products: 22, Total Products: 38, Bio Products Percentage: 57.89%

Department: canned goods, Bio Products: 320, Total Products: 2092, Bio Products Percentage: 15.30%

Department: dairy eggs, Bio Products: 435, Total Products: 3449, Bio Products Percentage: 12.61%

Department: deli, Bio Products: 100, Total Products: 1322, Bio Products Percentage: 7.56%

Department: dry goods pasta, Bio Products: 300, Total Products: 1858, Bio Products Percentage: 16.15%

Department: frozen, Bio Products: 252, Total Products: 4007, Bio Products Percentage: 6.29%

Department: household, Bio Products: 9, Total Products: 3085, Bio Products Percentage: 0.29%

Department: international, Bio Products: 84, Total Products: 1139, Bio Products Percentage: 7.37%

Department: meat seafood, Bio Products: 47, Total Products: 907, Bio Products Percentage: 5.18%

Department: missing, Bio Products: 175, Total Products: 1258, Bio Products Percentage: 13.91%

Department: other, Bio Products: 37, Total Products: 548, Bio Products Percentage: 6.75%

Department: pantry, Bio Products: 746, Total Products: 5371, Bio Products Percentage: 13.89%

Department: personal care, Bio Products: 187, Total Products: 6563, Bio Products Percentage: 2.85%

Department: pets, Bio Products: 23, Total Products: 972, Bio Products Percentage: 2.37%

Department: produce, Bio Products: 473, Total Products: 1684, Bio Products Percentage: 28.09%

Department: snacks, Bio Products: 656, Total Products: 6264, Bio Products Percentage: 10.47%

```
In [26]: gesamt_bio_produkte = sum([result['bio_products_count']] for result in departments)
gesamt_produkte = sum([result['total_products_count']] for result in departments)

gesamt_bio_anteil = gesamt_bio_produkte / gesamt_produkte if gesamt_produkte != 0 else 0

# Umwandlung der Anteile in Prozentsätze
bio_anteile_prozent = [result['bio_percentage'] * 100 for result in departments]

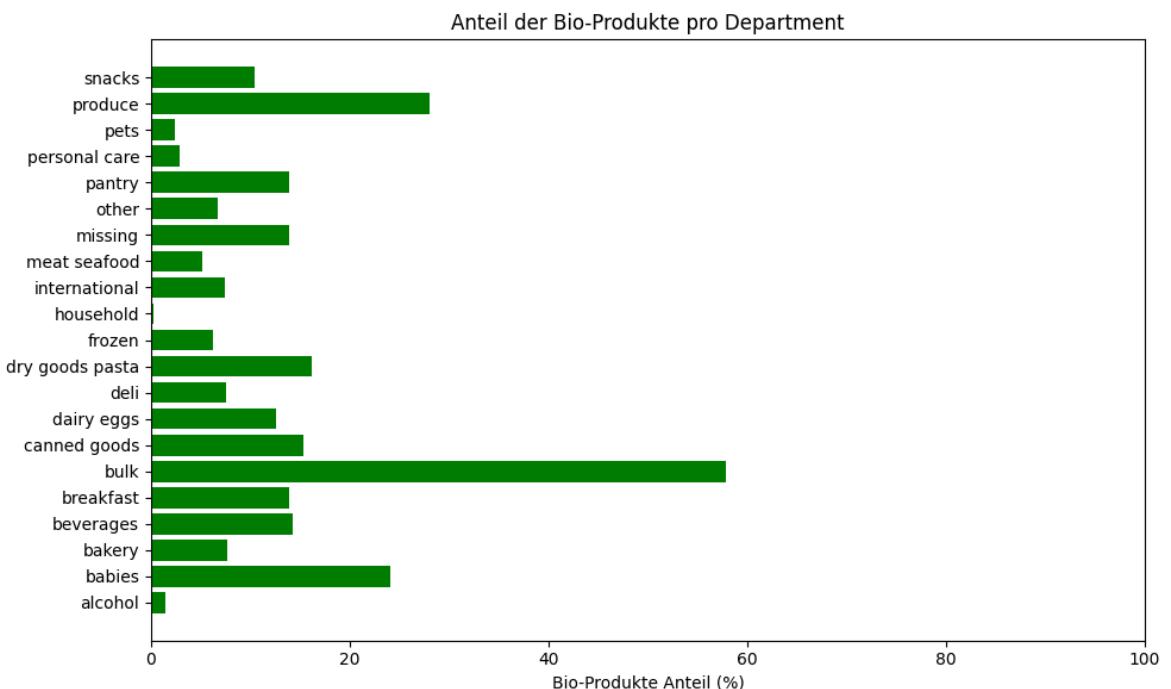
# Visualisierung des Anteils der Bio-Produkte pro Department (Balkendiagramm)
departments = [result['department_name'] for result in department_bio_percentage]

plt.figure(figsize=(10, 6))
plt.barh(departments, bio_anteile_prozent, color='green')

# Setze die X-Achse auf den Bereich von 0 bis 100% (0 bis 100)
plt.xlim(0, 100)

plt.xlabel('Bio-Produkte Anteil (%)')
plt.title('Anteil der Bio-Produkte pro Department')

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



Schlussfolgerungen:

Obwohl Bio-Produkte nur 10% des gesamten Produktsortiments ausmachen, sind sie in 73,5% der Bestellungen mindestens einmal vertreten. Dies zeigt, dass Bio-Produkte eine hohe Nachfrage haben, obwohl sie in der Produktvielfalt relativ gering vertreten sind. Dies könnte darauf hindeuten, dass Kunden, die sich für Bio-Produkte entscheiden, diese häufiger in ihren Bestellungen einbeziehen. Ein höherer Anteil von Bio-Produkten könnte daher potenziell das Einkaufserlebnis und die Zufriedenheit der Kunden steigern.

Für die Marketing Abteilung wurde ein Empfehlungsdienst für die Nutzer implementiert

```
In [28]: # Schritt 1: Alle Benutzer-IDs erfassen
all_users = session.query(Order.user_id).distinct().all()

# Benutzer-IDs extrahieren
all_user_ids = [user.user_id for user in all_users]

# Schritt 2: 10 % der Benutzer zufällig auswählen
sampled_user_ids = random.sample(all_user_ids, int(len(all_user_ids) * 0.1))

# Schritt 3: Abfrage für Bestellungen dieser Benutzer
result = session.query(
    Order.user_id,
    Einkaufskorb.order_id,
    Einkaufskorb.product_id,
    Product.product_name
).join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id) \
.join(Product, Einkaufskorb.product_id == Product.product_id) \
.filter(Order.user_id.in_(sampled_user_ids)) \
.distinct()

df = pd.DataFrame(result, columns=["user_id", "order_id", "product_id", "product_name"])

print(df.head())

```

	user_id	order_id	product_id	product_name
0	179300	144	1503	Low Fat Cottage Cheese
1	179300	144	7948	Organic Unsalted Butter
2	179300	144	8424	Broccoli Crown
3	179300	144	12384	Organic Lactose Free 1% Lowfat Milk
4	179300	144	17794	Carrots

```
In [29]: user_product_matrix = df.pivot_table(index='user_id', columns='product_id')

# Schritt 2: Cosine Similarity berechnen
user_similarity = cosine_similarity(user_product_matrix)
user_similarity_df = pd.DataFrame(user_similarity, index=user_product_matrix.index)

# Schritt 3: Empfehlungen generieren
def recommend_products(user_id, n_recommendations=5):
    # Ähnlichkeit des Nutzers zu anderen Nutzern
    similar_users = user_similarity_df[user_id].sort_values(ascending=False)

    # Produkte des angegebenen Nutzers
    user_products = set(user_product_matrix.columns[user_product_matrix.loc[user_id] > 0])

    # Empfehlungen sammeln
    recommendations = {}
    for similar_user in similar_users:
        similar_user_products = set(user_product_matrix.columns[user_product_matrix.loc[similar_user] > 0])
        new_products = similar_user_products - user_products
        for product in new_products:
            if product not in recommendations:
                recommendations[product] = 0
            recommendations[product] += user_similarity_df.loc[user_id, similar_user]

    # Sortiere Empfehlungen nach Gewichtung
    sorted_recommendations = sorted(recommendations.items(), key=lambda x: x[1], reverse=True)
    return sorted_recommendations[:n_recommendations]
```

```
# Gib die Top N Produkt-IDs zurück
return [product for product, score in sorted_recommendations[:n_recom]]
```

In [30]: recommendations = recommend_products(sampled_user_ids[1], n_recommendations)
recommended_products = df[df["product_id"].isin(recommendations)]
recommended_product_names = recommended_products["product_name"].unique()

In [32]: print(f"Empfohlene Produkte für Nutzer {sampled_user_ids[1]}:")
for name in recommended_product_names:
 print(name)

```
Empfohlene Produkte für Nutzer 53987:
Cucumber Kirby
Organic Lemon
Bag of Organic Bananas
Banana
Large Lemon
Organic Cucumber
Organic Avocado
Organic Whole Milk
Strawberries
Organic Raspberries
```

EDA Disposition



1. Allgemeine Analyse des Trinkgeldverhaltens

⌚ Zielsetzung:

Untersuchung der generellen Verteilung der Trinkgeldvergabe über alle Bestellungen hinweg.

In [33]: tip_counts = (
 session.query(
 Order.tips,
 func.count(Order.order_id)
)
 .group_by(Order.tips)
 .all()
)

tip_counts_df = pd.DataFrame(tip_counts, columns=['tips', 'count'])
total_orders = tip_counts_df['count'].sum()
tip_counts_df['percentage'] = (tip_counts_df['count'] / total_orders) * 1
table_markdown = tabulate(tip_counts_df, headers='keys', tablefmt='pipe')

plt.figure(figsize=(6, 6))

labels = ['Ohne Trinkgeld', 'Mit Trinkgeld']
colors = ['salmon', 'lightgreen']

```

patches, texts, autotexts = plt.pie(tip_counts_df['count'],
                                    autopct='%1.1f%%',
                                    colors=colors,
                                    startangle=90)

plt.legend(patches, labels, title="Legende", loc="center left", bbox_to_a

fig = plt.gcf()
fig.text(0.415, 0.875, 'Verteilung der Bestellungen mit/ohne Trinkgeld',
         fontweight='bold', fontsize=16, ha='center')
description = 'Die Analyse zeigt die prozentuale Verteilung von Bestellun
fig.text(0, 0.17, f'Abbildung 1:', fontweight='bold', ha='center')
fig.text(0.42, 0.14, description, ha='center')

plt.subplots_adjust(bottom=0.2, right=0.85)

plt.show()
display(Markdown(f"#{table_markdown}"))

```

Verteilung der Bestellungen mit/ohne Trinkgeld

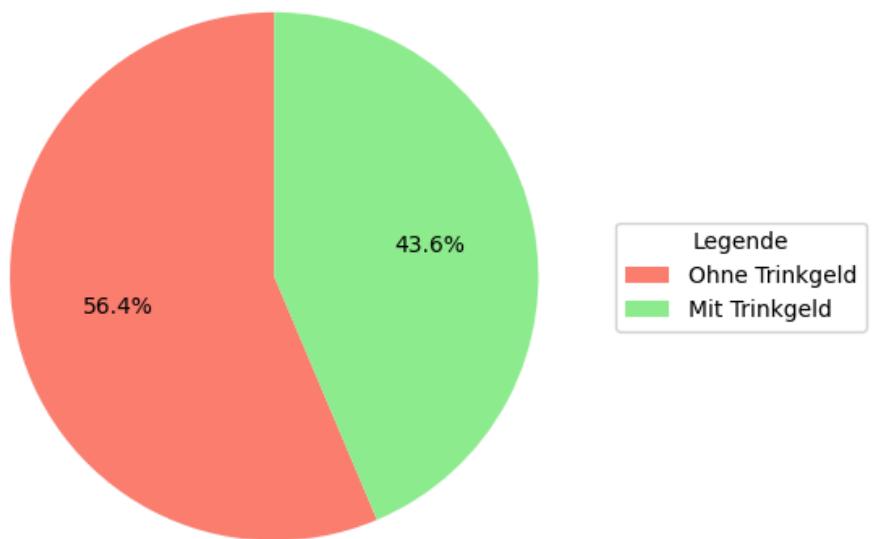


Abbildung 1:

Die Analyse zeigt die prozentuale Verteilung von Bestellungen mit und ohne Trinkgeld.

	tips	count	percentage
0	False	942908	56.3596
1	True	730113	43.6404

📊 Interpretation:

Es gibt im Allgemeinen etwas mehr Bestellungen ohne Trinkgeld als Bestellungen mit Trinkgeld.

📌 Nächster Schritt:

Wir möchten nun herausfinden welche Ursachen es haben könnte, wieso es weniger Bestellungen ohne Trinkgeld gibt. Außerdem wollen wir herausfinden ob

es möglicherweise Variablen gibt die im Zusammenhang mit der Trinkgeldvergabe stehen



2. Produktspezifische Analyse

Zielsetzung:

Analyse des Trinkgeldverhaltens in Bezug auf spezifische Produktkategorien und deren Einfluss auf das Kundenverhalten.

In [35]:

```
# Query: Anzahl der Bestellungen mit und ohne Trinkgeld pro Abteilung
orders_by_department_with_tips = session.query(
    Department.department_name,
    func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_tips'),
    func.sum(case((Order.tips == False, 1), else_=0)).label('orders_without_tips')
).join(Product, Product.department_id == Department.department_id)\ 
.join(Einkaufskorb, Einkaufskorb.product_id == Product.product_id)\ 
.join(Order, Einkaufskorb.order_id == Order.order_id)\ 
.group_by(Department.department_name).all()

departments = [department for department, _, _ in orders_by_department_with_tips]
orders_with_tips = [orders_with_tips for _, orders_with_tips, _ in orders_by_department_with_tips]
orders_without_tips = [orders_without_tips for _, _, orders_without_tips in orders_by_department_with_tips]
```



Analyse des Trinkgeldverhaltens nach Produktabteilungen

Visualisierung

Der folgende Graph zeigt die Verteilung von Bestellungen mit und ohne Trinkgeld für verschiedene Produktabteilungen. Die Abteilungen sind nach dem Verhältnis von Trinkgeld-Bestellungen zu Gesamtbestellungen sortiert.

Wichtig: Eine Bestellung wird einer Produktabteilung zugeordnet, wenn sie mindestens ein Produkt dieser Abteilung enthält. Dabei kann eine Bestellung mehreren Produktabteilungen zugewiesen werden.

Lesehinweise

Balken: Zeigen absolute Anzahl der Bestellungen

Grün: Bestellungen mit Trinkgeld

Rot: Bestellungen ohne Trinkgeld

Verhältnis: Zahlen rechts zeigen Quotient (Trinkgeld/Gesamt)

In [37]:

```
# plot code
# Positionen für die Balken (nun nach Quotienten sortiert)
ratios = [
    orders_with_tips[i] / (orders_with_tips[i] + orders_without_tips[i])
]
```

```

    for i in range(len(departments))
]

# Abteilungen nach dem Quotienten sortieren (Index des Quotienten als Sort
sorted_indices = np.argsort(ratios)

# Abteilungen und Bestellungen nach der Sortierung anpassen
departments_sorted = [departments[i] for i in sorted_indices]
orders_with_tips_sorted = [orders_with_tips[i] for i in sorted_indices]
orders_without_tips_sorted = [orders_without_tips[i] for i in sorted_indices]
ratios_sorted = [ratios[i] for i in sorted_indices]

x = np.arange(len(departments_sorted))

fig, ax = plt.subplots(figsize=(10, 7))
plt.subplots_adjust(left=0.08)
bar_width = 0.5

ax.barh(x, orders_with_tips_sorted, color='lightgreen', label='Mit Trinkgeld')
ax.barh(x, orders_without_tips_sorted, left=orders_with_tips_sorted, color='lightblue')

ax.set_xlabel('Anzahl Bestellungen')
ax.set_yticks(x)
ax.set_yticklabels(departments_sorted)
ax.set_title('Bestellungen mit und ohne Trinkgeld nach Produktabteilung\n'
             'loc='left',
             weight='bold',
             x=-0.125
             )

# Quotienten (Bestellungen mit Trinkgeld / Gesamtbestellungen) berechnen
for i, department in enumerate(departments_sorted):
    total_orders = orders_with_tips_sorted[i] + orders_without_tips_sorted[i]
    ratio = ratios_sorted[i]

    ax.text(
        orders_with_tips_sorted[i] + orders_without_tips_sorted[i] + 100,
        i,
        f'{ratio:.2f}',
        va='center',
        ha='left',
        fontsize=10,
        color='black'
    )

ax.legend(loc='upper right')
ax.ticklabel_format(style='plain', axis='x')
ax.grid(True, axis='x', linestyle='--', alpha=0.7)

title = 'Abbildung 2:'
description = 'Die Analyse zeigt die Verteilung der Bestellungen nach Pro'

fig.text(0.03, 0, title, fontweight='bold')
fig.text(0.03, -0.05, description, wrap=True)

plt.tight_layout()
plt.show()

```

**Bestellungen mit und ohne Trinkgeld nach Produktabteilung
(Bestellungen die min. 1 Produkt der jeweiligen Abteilung enthalten)**

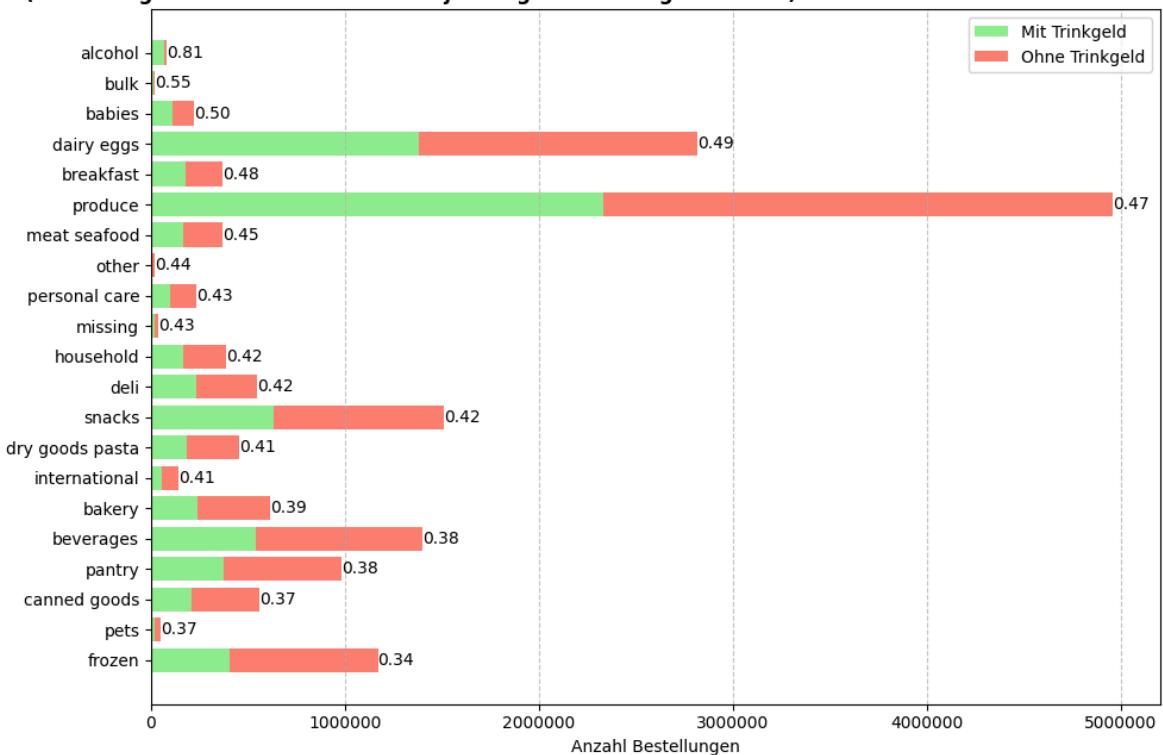


Abbildung 2:

Die Analyse zeigt die Verteilung der Bestellungen nach Produktabteilungen, wobei zwischen Bestellungen mit und ohne Trinkgeld unterschieden wird. Die Zahlen rechts geben das Verhältnis von Bestellungen mit Trinkgeld zur Gesamtanzahl der Bestellungen an.



Chi-Quadrat-Test: Analyse der Verteilungsunterschiede

🎯 Zielsetzung:

Statistische Überprüfung, ob die beobachteten Unterschiede in der Verteilung von Bestellungen mit und ohne Trinkgeld signifikant sind.

📌 Hypothesen:

- H_0 : Keine Abhängigkeit zwischen Produktabteilung und Trinkgeldverhalten
- H_1 : Signifikanter Zusammenhang zwischen Produktabteilung und Trinkgeldverhalten

⚙️ Testparameter:

- Signifikanzniveau: $\alpha = 0.05$
- Statistische Methode: Chi-Quadrat-Unabhängigkeitstest

In [38]:

```
# chi² code
contingency_table = pd.DataFrame({
    'Mit_Trinkgeld': orders_with_tips,
    'Ohne_Trinkgeld': orders_without_tips
}, index=departments)

chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Testergebnisse in Tabelle formatieren
```

```
test_results = [
    ["Chi-Quadrat Statistik", f"{chi2}"],
    ["p-Wert", f"{p_value}"],
    ["Freiheitsgrade", dof]
]

print("\nChi-Quadrat Testergebnisse:")
print(tabulate(test_results, headers=["Metrik", "Wert"],
               tablefmt="fancy_grid", numalign="right"))

alpha = 0.05
print("\nInterpretation:")
if p_value < alpha:
    interpretation = [
        ["Ergebnis", "Statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value}) < Signifikanzniveau ({alpha})"],
        ["Schlussfolgerung", "Department und Trinkgeldvergabe sind abhäng"]
    ]
else:
    interpretation = [
        ["Ergebnis", "Kein statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value}) > Signifikanzniveau ({alpha})"],
        ["Schlussfolgerung", "Department und Trinkgeldvergabe sind unabhä"]
    ]

print(tabulate(interpretation, headers=["", ""],
               tablefmt="fancy_grid"))

contingency_table['Gesamt'] = contingency_table['Mit_Trinkgeld'] + contingency_table['Ohne_Trinkgeld']
contingency_table['Anteil_mit_Trinkgeld'] = (contingency_table['Mit_Trinkgeld'] / contingency_table['Gesamt']) * 100

print("\nKontingenztabelle:")
print(tabulate(contingency_table, headers='keys', tablefmt="fancy_grid",
               numalign="right"))
```

Chi-Quadrat Testergebnisse:

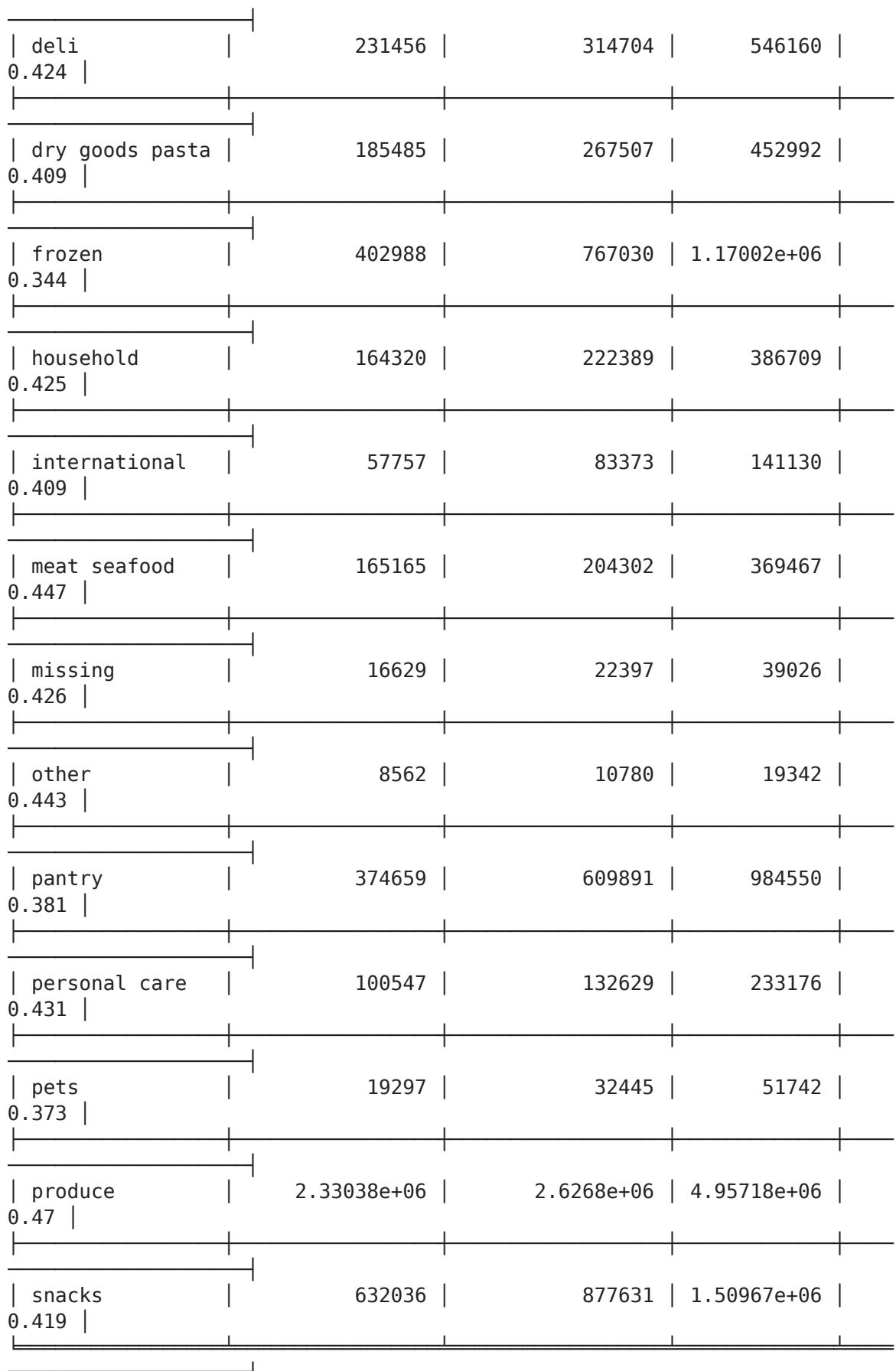
Metrik	Wert
Chi-Quadrat Statistik	199365
p-Wert	0
Freiheitsgrade	20

Interpretation:

Ergebnis	Statistisch signifikanter Zusammenhang
Begründung	p-Wert (0.0) < Signifikanzniveau (0.05)
Schlussfolgerung	Department und Trinkgeldvergabe sind abhängig

Kontingenztabelle:

	Mit_Trinkgeld	Ohne_Trinkgeld	Gesamt	A
anteil_mit_Trinkgeld				
alcohol 0.813	66754	15391	82145	
babies 0.496	108596	110400	218996	
bakery 0.388	238201	375143	613344	
beverages 0.384	538026	864297	1.40232e+06	
breakfast 0.476	176053	193967	370020	
bulk 0.554	9991	8045	18036	
canned goods 0.373	209170	351018	560188	
dairy eggs 0.491	1.38308e+06	1.43295e+06	2.81603e+06	



🔥 Visualisierung der Chi-Quadrat-Residuen

📊 Analyse der prozentualen Abweichungen

Um die Ergebnisse des Chi-Quadrat-Tests detaillierter zu untersuchen, visualisieren wir die prozentualen Abweichungen in Form einer Heatmap. Diese zeigt, wie stark die beobachteten von den erwarteten Häufigkeiten abweichen.

Berechnung der prozentualen Abweichung:

$$\text{Prozentuale Abweichung} = ((O - E) / E) * 100$$

wobei:

O = beobachtete Häufigkeit

E = erwartete Häufigkeit

Interpretation der Farben:

- **Rot (negative Prozente):** Weniger Fälle als statistisch erwartet
- Etwa so viele Fälle wie erwartet
- **Blau (positive Prozente):** Mehr Fälle als statistisch erwartet

Interpretation der Werte:

- +50% bedeutet: 50% mehr Fälle als erwartet
- -25% bedeutet: 25% weniger Fälle als erwartet
- 0% bedeutet: Genau so viele Fälle wie erwartet

```
In [39]: # plot code
observed = contingency_table[['Mit_Trinkgeld', 'Ohne_Trinkgeld']].values
_, _, _, expected = chi2_contingency(observed)

# Prozentuale Abweichung berechnen
# Formel: ((O - E) / E) * 100
percentage_deviation = ((observed - expected) / expected)

deviation_df = pd.DataFrame(
    percentage_deviation,
    index=departments,
    columns=['Mit_Trinkgeld', 'Ohne_Trinkgeld']
)

deviation_df = deviation_df.reindex(departments_sorted[::-1])

fig, ax = plt.subplots(figsize=(8, 8))
sns.heatmap(deviation_df,
            annot=True,
            cmap='RdBu',
            center=0,
            fmt='.1%',
            cbar_kws={'label': 'Prozentuale Abweichung'},
            ax=ax)

ax.set_title('Prozentuale Abweichung zwischen beobachteten und erwarteten',
             loc='left',
```

```

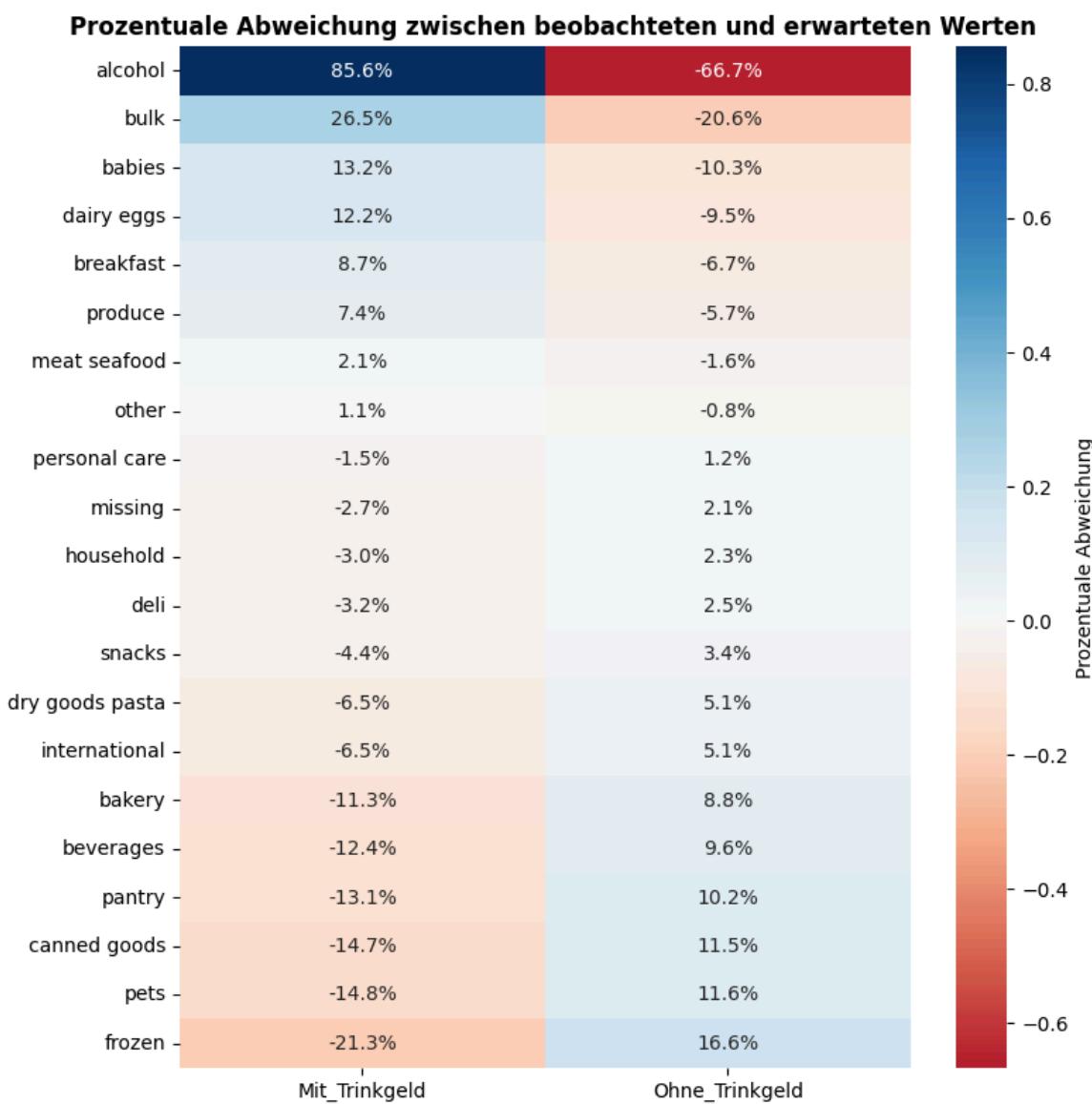
        weight='bold',
        x=-0.15)

description = 'Die Heatmap zeigt die prozentuale Abweichung zwischen beobachteten und erwarteten Häufigkeiten. Negative Werte (rot) zeigen einen niedrigeren Prozentsatz als erwartet, positive Werte (blau) einen höheren Prozentsatz als erwartet.'
fig.text(0.05, 0, 'Abbildung 3:', weight='bold', ha='left')
fig.text(0.05, -0.06, description, wrap=True)

plt.tight_layout()
plt.show()

print("\nInterpretation der prozentualen Abweichungen:")
interpretation_table = [
    ["Negative Werte (rot)", "Weniger Fälle als erwartet"],
    ["Werte nahe 0 (weiß)", "Etwa so viele Fälle wie erwartet"],
    ["Positive Werte (blau)", "Mehr Fälle als erwartet"]
]
print(tabulate(interpretation_table, headers=["Werte", "Bedeutung"],
               tablefmt="fancy_grid"))

```

**Abbildung 3:**

Die Heatmap zeigt die prozentuale Abweichung zwischen beobachteten und erwarteten Häufigkeiten. Negative Werte (rot) zeigen einen niedrigeren Prozentsatz als erwartet, positive Werte (blau) einen höheren Prozentsatz als erwartet.

Interpretation der prozentualen Abweichungen:

Werte	Bedeutung
Negative Werte (rot)	Weniger Fälle als erwartet
Werte nahe 0 (weiß)	Etwa so viele Fälle wie erwartet
Positive Werte (blau)	Mehr Fälle als erwartet

```
In [40]: # Query: Bestellungen mit und ohne Trinkgeld pro Abteilung, aber nur für
orders_with_single_department = session.query(
    Order.order_id,
    func.count(distinct(Product.department_id)).label('dept_count'))
).join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
.join(Product, Product.product_id == Einkaufskorb.product_id) \
.group_by(Order.order_id) \
.having(func.count(distinct(Product.department_id)) == 1) \
.subquery()

orders_by_department_with_tips = session.query(
    Department.department_name,
    func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_t'),
    func.sum(case((Order.tips == False, 1), else_=0)).label('orders_witho_')
).join(Product, Product.department_id == Department.department_id) \
.join(Einkaufskorb, Einkaufskorb.product_id == Product.product_id) \
.join(Order, Einkaufskorb.order_id == Order.order_id) \
.join(orders_with_single_department, orders_with_single_department.c.order_id == Order.order_id) \
.group_by(Department.department_name).all()

departments = [department for department, _, _ in orders_by_department_with_tips]
orders_with_tips = [orders_with_tips for _, orders_with_tips, _ in orders_by_department_with_tips]
orders_without_tips = [orders_without_tips for _, _, orders_without_tips in orders_by_department_with_tips]
```

📊 Analyse von Single-Department-Bestellungen

🔍 Fokussierte Departmentanalyse

Nach der Analyse aller Bestellungen betrachten wir nun spezifisch die Bestellungen, die ausschließlich Produkte aus einer einzelnen Produktabteilung enthalten. Dies ermöglicht eine präzisere Untersuchung des Trinkgeldverhaltens ohne Überschneidungseffekte.

📌 Besonderheiten der Analyse:

- Nur Bestellungen mit Produkten aus genau einer Abteilung
- Keine Überschneidungen zwischen Abteilungen
- Reinere Darstellung des abteilungsspezifischen Trinkgeldverhaltens

📊 Darstellung:

- Horizontale Balken zeigen absolute Anzahl der Bestellungen
- Grün: Bestellungen mit Trinkgeld
- Rot: Bestellungen ohne Trinkgeld

- Zahlen rechts: Verhältnis (Trinkgeld/Gesamt)

```
In [41]: # plot code
# Quotient berechnen
ratios = [
    orders_with_tips[i] / (orders_with_tips[i] + orders_without_tips[i])
    for i in range(len(departments))
]

sorted_indices = np.argsort(ratios)

departments_sorted = [departments[i] for i in sorted_indices]
orders_with_tips_sorted = [orders_with_tips[i] for i in sorted_indices]
orders_without_tips_sorted = [orders_without_tips[i] for i in sorted_indices]
ratios_sorted = [ratios[i] for i in sorted_indices]

x = np.arange(len(departments_sorted))
fig, ax = plt.subplots(figsize=(10, 7))
plt.subplots_adjust(left=0.08) # Anpassung für einheitliche Einrückung
bar_width = 0.5

ax.barh(x, orders_with_tips_sorted, color='lightgreen', label='Mit Trinkgeld')
ax.barh(x, orders_without_tips_sorted, left=orders_with_tips_sorted, color='lightblue')

ax.set_xlabel('Anzahl Bestellungen')
ax.set_yticks(x)
ax.set_yticklabels(departments_sorted)
ax.set_title('Bestellungen mit und ohne Trinkgeld nach Produktabteilung\n'
             'loc='left',
             weight='bold',
             x=-0.115)

for i, department in enumerate(departments_sorted):
    total_orders = orders_with_tips_sorted[i] + orders_without_tips_sorted[i]
    ratio = ratios_sorted[i]

    ax.text(
        orders_with_tips_sorted[i] + orders_without_tips_sorted[i] + 100,
        i,
        f'{ratio:.2f}',
        va='center',
        ha='left',
        fontsize=10,
        color='black'
    )

ax.legend(loc='upper right')
ax.ticklabel_format(style='plain', axis='x')
ax.grid(True, axis='x', linestyle='--', alpha=0.7)

description = 'Die Grafik zeigt die Verteilung von Bestellungen mit und ohne Trinkgeld nach Produktabteilung. Die Y-Achse listet die Abteilungen auf, die X-Achse zeigt die Anzahl der Bestellungen. Die Farben verdeutlichen, ob Trinkgeld bezahlt wurde: hellgrün für mit Trinkgeld, hellblau für ohne. Die Zahlen rechts neben den Balken stellen das Verhältnis von Trinkgeld zu Gesamt dar.'
fig.text(0.04, 0, 'Abbildung 4:', weight='bold', ha='left')
fig.text(0.04, -0.05, description, wrap=True)

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

**Bestellungen mit und ohne Trinkgeld nach Produktabteilung
(nur Bestellungen mit Produkten aus einem Department)**

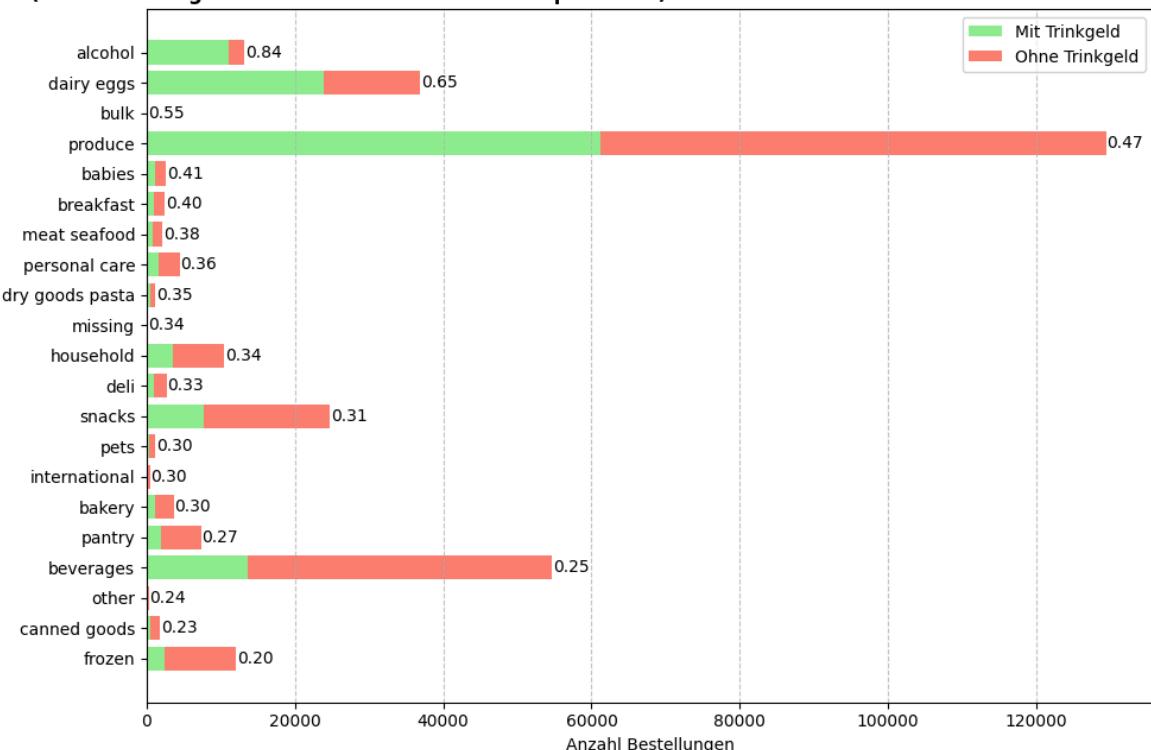


Abbildung 4:

Die Grafik zeigt die Verteilung von Bestellungen mit und ohne Trinkgeld für Bestellungen, die ausschließlich Produkte aus einer einzigen Abteilung enthalten. Die Zahlen rechts geben das Verhältnis von Bestellungen mit Trinkgeld zur Gesamtzahl der Bestellungen an.

🔍 Zentrale Erkenntnisse der Single-Department-Analyse

⚡ Überraschungen

📈 Auffälliger Zuwachs

- Dairy Eggs

📈 Auffälliger Rückgang

- Frozen
- Beverages

```
In [42]: # chi2 code
contingency_table = pd.DataFrame({
    'Mit_Trinkgeld': orders_with_tips,
    'Ohne_Trinkgeld': orders_without_tips
}, index=departments)

chi2, p_value, dof, expected = chi2_contingency(contingency_table)

test_results = [
    ["Chi-Quadrat Statistik", f"{chi2}"],
    ["p-Wert", f"{p_value}"],
    ["Freiheitsgrade", dof]
```

```
[1]

print("\nChi-Quadrat Testergebnisse:")
print(tabulate(test_results, headers=["Metrik", "Wert"],
               tablefmt="fancy_grid", numalign="right"))

alpha = 0.05
print("\nInterpretation:")
if p_value < alpha:
    interpretation = [
        ["Ergebnis", "Statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value}) < Signifikanzniveau ({alpha})"],
        ["Schlussfolgerung", "Department und Trinkgeldvergabe sind abhäng"]
    ]
else:
    interpretation = [
        ["Ergebnis", "Kein statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value}) > Signifikanzniveau ({alpha})"],
        ["Schlussfolgerung", "Department und Trinkgeldvergabe sind unabhä"]
    ]

print(tabulate(interpretation, headers=["", ""],
               tablefmt="fancy_grid"))

contingency_table['Gesamt'] = contingency_table['Mit_Trinkgeld'] + contingency_table['Ohne_Trinkgeld']
contingency_table['Anteil_mit_Trinkgeld'] = (contingency_table['Mit_Trinkgeld'] / contingency_table['Gesamt']) * 100

print("\nKontingenztabelle:")
print(tabulate(contingency_table, headers='keys', tablefmt="fancy_grid",
               numalign="right", floatfmt=".3f"))
```

Chi-Quadrat Testergebnisse:

Metrik	Wert
Chi-Quadrat Statistik	30670.2
p-Wert	0
Freiheitsgrade	20

Interpretation:

Ergebnis	Statistisch signifikanter Zusammenhang
Begründung	p-Wert (0.0) < Signifikanzniveau (0.05)
Schlussfolgerung	Department und Trinkgeldvergabe sind abhängig

Kontingenztabelle:

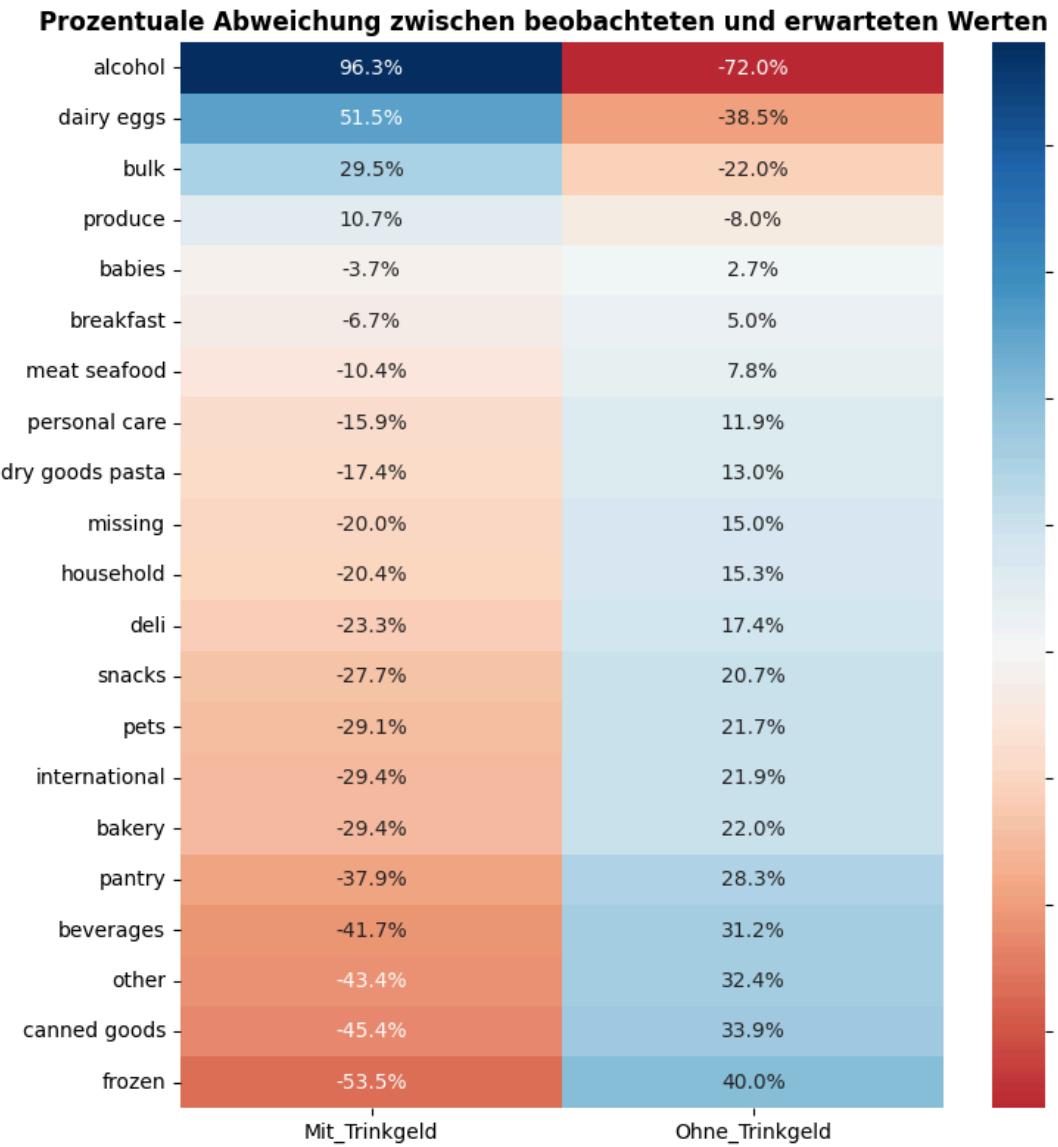
	Mit_Trinkgeld	Ohne_Trinkgeld	Gesamt	An
teil_mit_Trinkgeld				
alcohol 0.840	11040.000	2110.000	13150.000	
babies 0.412	1062.000	1515.000	2577.000	
bakery 0.302	1094.000	2529.000	3623.000	
beverages 0.249	13618.000	41024.000	54642.000	
breakfast 0.399	982.000	1479.000	2461.000	
bulk 0.554	72.000	58.000	130.000	
canned goods 0.234	427.000	1401.000	1828.000	
dairy eggs 0.648	23911.000	12979.000	36890.000	

deli 0.328	893.000	1830.000	2723.000
dry goods pasta 0.353	416.000	762.000	1178.000
frozen 0.199	2394.000	9630.000	12024.000
household 0.340	3551.000	6883.000	10434.000
international 0.302	139.000	321.000	460.000
meat seafood 0.383	802.000	1290.000	2092.000
missing 0.342	39.000	75.000	114.000
other 0.242	62.000	194.000	256.000
pantry 0.266	1947.000	5386.000	7333.000
personal care 0.360	1596.000	2842.000	4438.000
pets 0.303	358.000	822.000	1180.000
produce 0.473	61260.000	68144.000	129404.000
snacks 0.309	7632.000	17064.000	24696.000

```
In [43]: # code plot
observed = contingency_table[['Mit_Trinkgeld', 'Ohne_Trinkgeld']].values
_, _, _ , expected = chi2_contingency(observed)

# Prozentuale Abweichung berechnen
percentage_deviation = ((observed - expected) / expected)
```

```
deviation_df = pd.DataFrame(  
    percentage_deviation,  
    index=departments,  
    columns=['Mit_Trinkgeld', 'Ohne_Trinkgeld'])  
  
deviation_df = deviation_df.reindex(departments_sorted[::-1])  
  
fig, ax = plt.subplots(figsize=(8, 8))  
plt.subplots_adjust(left=0.08)  
  
sns.heatmap(deviation_df,  
            annot=True,  
            cmap='RdBu',  
            center=0,  
            fmt='.1%',  
            cbar_kws={'label': 'Prozentuale Abweichung'},  
            ax=ax)  
  
ax.set_title('Prozentuale Abweichung zwischen beobachteten und erwarteten  
              Werten',  
             loc='left',  
             weight='bold',  
             x=-0.185)  
  
description = 'Die Heatmap visualisiert die prozentualen Abweichungen für  
fig.text(0.05, 0, 'Abbildung 5:', weight='bold', ha='left')  
fig.text(0.05, -0.06, description, wrap=True)  
  
plt.tight_layout()  
plt.show()  
  
print("\nInterpretation der prozentualen Abweichungen:")  
interpretation_table = [  
    ["Negative Werte (rot)", "Weniger Fälle als erwartet"],  
    ["Werte nahe 0% (weiß)", "Etwa so viele Fälle wie erwartet"],  
    ["Positive Werte (blau)", "Mehr Fälle als erwartet"]  
]  
print(tabulate(interpretation_table, headers=["Werte", "Bedeutung"],  
               tablefmt="fancy_grid"))
```

**Abbildung 5:**

Die Heatmap visualisiert die prozentualen Abweichungen für jede Produktabteilung. Negative Werte (rot) zeigen weniger Fälle als erwartet, positive Werte (blau) mehr Fälle als erwartet. Werte nahe 0% entsprechen etwa den erwarteten Häufigkeiten.

Interpretation der prozentualen Abweichungen:

Werte	Bedeutung
Negative Werte (rot)	Weniger Fälle als erwartet
Werte nahe 0% (weiß)	Etwa so viele Fälle wie erwartet
Positive Werte (blau)	Mehr Fälle als erwartet

```
In [44]: # Anzahl der Gesamtbestellungen und Bestellungen mit Trinkgeld pro Gang (aisles_stats = session.query(
    Aisle.aisle_name,
    func.count(Order.order_id).label('total_orders'),
    func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_t')).join(Product, Product.aisle_id == Aisle.aisle_id) \
    .join(Einkaufskorb, Einkaufskorb.product_id == Product.product_id) \
    .join(Order, Einkaufskorb.order_id == Order.order_id) \
    .group_by(Aisle.aisle_name) \
    .all()
```

```
In [45]: # utils code
aisles_df = pd.DataFrame(aisles_stats, columns=['aisle_name', 'total_orders'])
aisles_df['orders_without_tips'] = aisles_df['total_orders'] - aisles_df['orders_with_tips']
aisles_df['tip_ratio'] = aisles_df['orders_with_tips'] / aisles_df['total_orders']

aisles_top_10 = aisles_df.nlargest(10, 'tip_ratio').iloc[::-1] # Top 10
aisles_flop_10 = aisles_df.nsmallest(10, 'tip_ratio')

def plot_aisles(aisles_data, title, ax):
    x = np.arange(len(aisles_data))

    ax.barh(x, aisles_data['orders_with_tips'], color='lightgreen', label='Bestellungen mit Trinkgeld')
    ax.barh(x, aisles_data['orders_without_tips'], left=aisles_data['orders_with_tips'], color='red', label='Bestellungen ohne Trinkgeld')

    ax.set_xlabel('Anzahl Bestellungen')
    ax.set_yticks(x)
    ax.set_yticklabels(aisles_data['aisle_name'])
    ax.set_title(title, loc='left', weight='bold')

    for tick in ax.get_xticks():
        ax.axvline(x=tick, color='grey', linestyle='--', linewidth=0.5)

    for i, aisle in enumerate(aisles_data['aisle_name']):
        ratio = aisles_data['tip_ratio'].iloc[i]
        total_orders = aisles_data['total_orders'].iloc[i]
        ax.text(
            x=aisles_data['orders_with_tips'].iloc[i] + aisles_data['orders_without_tips'].iloc[i],
            y=i,
            s=f'{ratio:.2f}',
            va='center',
            ha='left',
            fontsize=10,
            color='black'
        )

    ax.legend(loc='upper right')
```



Detailanalyse der Produktkategorien (Aisles)



Top & Flop Analyse

Nach der Untersuchung der übergeordneten Departments zoomen wir nun tiefer in die einzelnen Produktkategorien (Aisles) hinein. Diese detailliertere Betrachtung ermöglicht es uns, spezifischere Muster im Trinkgeldverhalten zu identifizieren.



Visualisierungsdetails:

- **Linker Plot:** Top 10 Aisles mit höchstem Trinkgeld-Verhältnis
- **Rechter Plot:** Flop 10 Aisles mit niedrigstem Trinkgeld-Verhältnis
- Grüne Balken: Bestellungen mit Trinkgeld
- Rote Balken: Bestellungen ohne Trinkgeld
- Zahlen rechts: Verhältnis (Trinkgeld/Gesamt)

🎯 Analyseziel:

Identifikation von spezifischen Produktkategorien, die besonders stark mit dem Trinkgeldverhalten korrelieren, um gezieltere Erkenntnisse für Marketing- und Service-Strategien zu gewinnen.

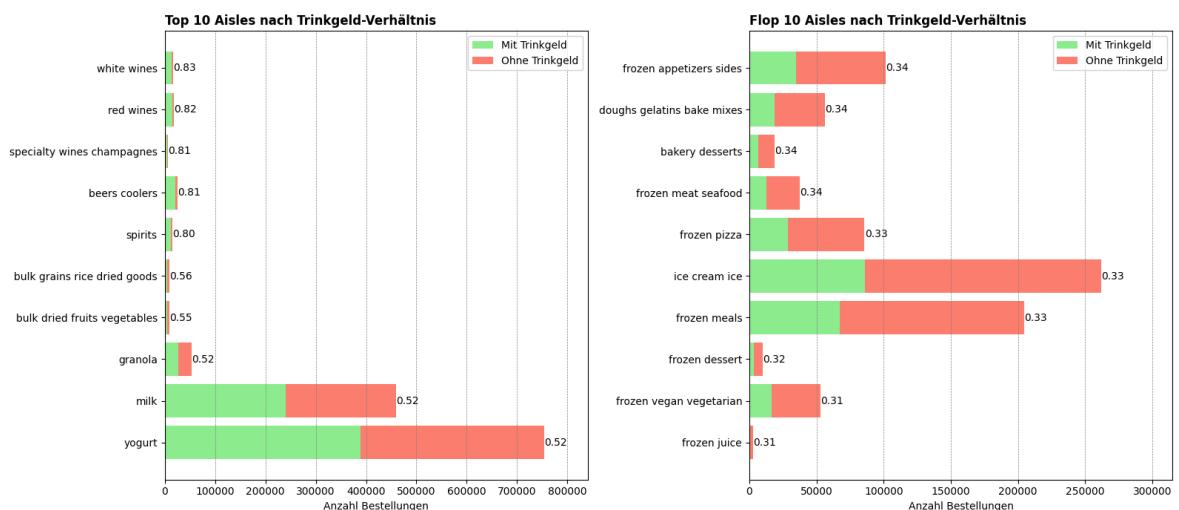
In [46]:

```
# plot code
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 7))
plt.subplots_adjust(left=0.08)

plot_aisles(aisles_top_10, 'Top 10 Aisles nach Trinkgeld-Verhältnis', ax1)
plot_aisles(aisles_flop_10, 'Flop 10 Aisles nach Trinkgeld-Verhältnis', ax2)

description = 'Die Grafiken zeigen die zehn Produktkategorien (Aisles) mit dem höchsten (links) und niedrigsten (rechts) Anteil an Trinkgeld-Bestellungen. Die Balken zeigen die absolute Anzahl der Bestellungen, die Zahlen rechts das Verhältnis von Bestellungen mit Trinkgeld zur Gesamtanzahl der Bestellungen.'
fig.text(0.05, 0, 'Abbildung 6:', weight='bold', ha='left')
fig.text(0.05, -0.05, description, wrap=True)

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

**Abbildung 6:**

Die Grafiken zeigen die zehn Produktkategorien (Aisles) mit dem höchsten (links) und niedrigsten (rechts) Anteil an Trinkgeld-Bestellungen. Die Balken zeigen die absolute Anzahl der Bestellungen, die Zahlen rechts das Verhältnis von Bestellungen mit Trinkgeld zur Gesamtanzahl der Bestellungen.

🔍 Bestätigung übergeordneter Muster

🍷 Alkoholische Produkte in den Top 10

Die detaillierte Analyse der Aisles bestätigt den bereits auf Department-Ebene beobachteten Trend: Alkoholische Produkte sind überproportional häufig in Bestellungen mit Trinkgeld vertreten. Dies zeigt sich durch:

- Mehrere alkoholbezogene Kategorien in den Top 10
- Konsistenz mit der hohen Trinkgeldquote des Alcohol-Departments
- Möglicherweise höhere Serviceerwartung bei diesen Produkten

Tiefkühlprodukte in den Flop 10

Auch am unteren Ende der Skala bestätigen sich die Department-Level-Erkenntnisse: Tiefkühlprodukte weisen durchgängig niedrigere Trinkgeldquoten auf. Dies spiegelt sich wider in:

- Mehreren Tiefkühl-Kategorien unter den Flop 10
- Übereinstimmung mit der niedrigen Quote des Frozen-Departments
- Konsistentes Muster über verschiedene Tiefkühl-Produktkategorien hinweg

Fazit:

Die Aisle-Level-Analyse bestätigt und verfeinert die Erkenntnisse der Department-Analyse. Dies unterstreicht die Robustheit der beobachteten Muster im Trinkgeldverhalten und ermöglicht eine noch gezieltere Ansprache verschiedener Produktsegmente.

```
In [48]: # Statistik über Produkte die mindestens 500 mal bestellt wurden: Gesamt
products_stats = session.query(
    Product.product_name,
    func.count(Order.order_id).label('total_orders'),
    func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_t
).join(Einkaufskorb, Einkaufskorb.product_id == Product.product_id) \
    .join(Order, Einkaufskorb.order_id == Order.order_id) \
    .group_by(Product.product_name) \
    .having(func.count(Order.order_id) >= 500).all()

products_df = pd.DataFrame(products_stats, columns=['product_name', 'tota
products_df['orders_without_tips'] = products_df['total_orders'] - produc
products_df['tip_ratio'] = products_df['orders_with_tips'] / products_df[

products_top_10 = products_df.nlargest(10, 'tip_ratio').iloc[::-1] # Top
products_flop_10 = products_df.nsmallest(10, 'tip_ratio')

def plot_products(products_data, title, ax):
    x = np.arange(len(products_data))

    ax.barh(x, products_data['orders_with_tips'], color='lightgreen', lab
    ax.barh(x, products_data['orders_without_tips'], left=products_data['

    ax.set_xlabel('Anzahl Bestellungen')
    ax.set_yticks(x)
    ax.set_yticklabels(products_data['product_name'])
    ax.set_title(title, loc='left', weight='bold')

    for tick in ax.get_xticks():
        ax.axvline(x=tick, color='grey', linestyle='--', linewidth=0.5)

    for i, product in enumerate(products_data['product_name']):
        ratio = products_data['tip_ratio'].iloc[i]
```

```

total_orders = products_data['total_orders'].iloc[i]
ax.text(
    products_data['orders_with_tips'].iloc[i] + products_data['or
    i,
    f'{ratio:.2f}',
    va='center',
    ha='left',
    fontsize=10,
    color='black'
)

ax.legend(loc='upper right')

```

Analyse auf Produkteinheit

Detailanalyse einzelner Produkte

Nach der Analyse der Departments und Aisles gehen wir nun auf die granularste Ebene: die einzelnen Produkte. Diese Mikroanalyse ermöglicht es uns, sehr spezifische Muster im Trinkgeldverhalten zu identifizieren.

Visualisierungsaufbau:

- **Linker Plot:** Die 10 Produkte mit der höchsten Trinkgeldquote
- **Rechter Plot:** Die 10 Produkte mit der niedrigsten Trinkgeldquote
- Absolute Anzahl der Bestellungen (Balkenlänge)
- Verhältnis Trinkgeld/Gesamt (Zahlen rechts)

Analyseziele:

- Identifikation von Produkten mit außergewöhnlichem Trinkgeldverhalten
- Verständnis produktsspezifischer Kundenentscheidungen
- Ableitung von Handlungsempfehlungen auf Produkteinheit

Besonderheit:

Diese Analyse auf Produkteinheit ermöglicht die präziseste Sicht auf das Trinkgeldverhalten und kann wichtige Einblicke für Produktplatzierung, Marketing und Serviceoptimierung liefern.

```

In [49]: # plot code
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 7))
plt.subplots_adjust(left=0.08)

plot_products(products_top_10, 'Top 10 Produkte nach Trinkgeld-Verhältnis')

plot_products(products_flop_10, 'Flop 10 Produkte nach Trinkgeld-Verhältnis')

```

```
description = 'Die Grafiken zeigen die zehn Produkte mit dem höchsten (li-  
fig.text(0.05, 0, 'Abbildung 7:', weight='bold', ha='left')  
fig.text(0.05, -0.05, description, wrap=True)  
  
plt.tight_layout()  
plt.show()'
```

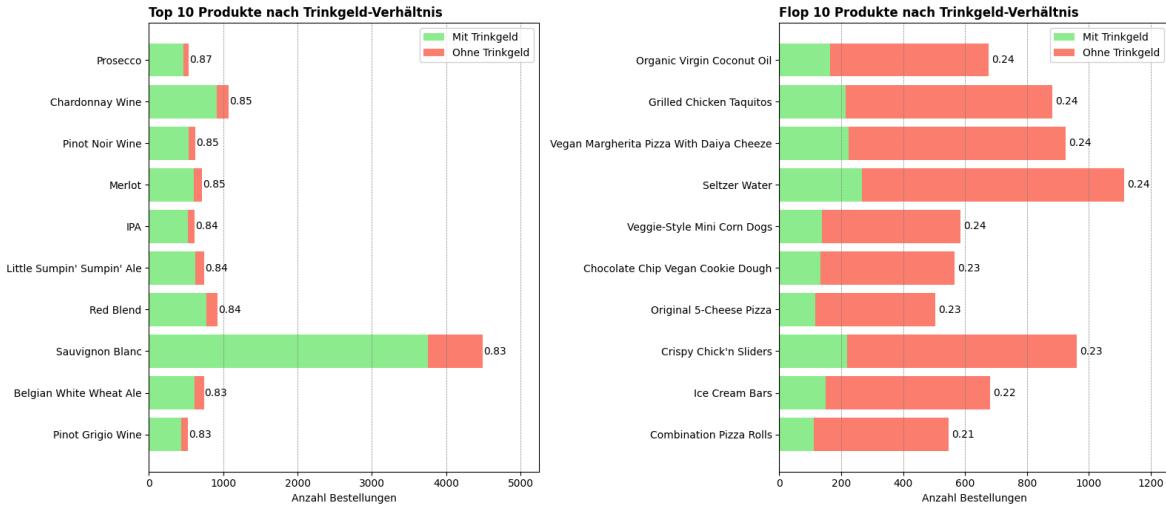


Abbildung 7:

Die Grafiken zeigen die zehn Produkte mit dem höchsten (links) und niedrigsten (rechts) Anteil an Trinkgeld-Bestellungen. Die Balken zeigen die absolute Anzahl der Bestellungen, die Zahlen rechts das Verhältnis von Bestellungen mit Trinkgeld zur Gesamtanzahl der Bestellungen.



Erkenntnisse der Produktanalyse



Konsistente Muster über alle Analyseebenen



Top 10 Produkte: Dominanz alkoholischer Getränke

Die Analyse auf Produktebene bestätigt eindrucksvoll die bereits beobachteten Trends:

- Starke Präsenz alkoholischer Getränke in den Top 10
- Konsistenz mit den Ergebnissen der Aisle-Analyse
- Bestätigung des Department-Level-Trends für alkoholische Produkte



Flop 10 Produkte: Tiefkühlprodukte dominieren

Auch am unteren Ende der Skala setzen sich die bereits erkannten Muster fort:

- Häufiges Auftreten von Tiefkühlprodukten unter den Flop 10
- Spiegelung der Erkenntnisse aus der Aisle-Analyse
- Weitere Bestätigung der niedrigen Trinkgeldquote im Frozen-Department



Fazit:

Die Analyse auf Produktebene verstärkt die Erkenntnisse der vorherigen Untersuchungen und zeigt eine bemerkenswerte Konsistenz über alle

Analyseebenen hinweg - von Department über Aisles bis hin zu einzelnen Produkten. Diese durchgängigen Muster unterstreichen die Robustheit unserer Erkenntnisse zum Trinkgeldverhalten.

Temporale Analyse des Trinkgeldverhaltens nach Produktkategorien Alkohol und Frozen

Zeitliche Muster auf Stunden- und Tagesebene

Nach der Identifikation der Produktkategorien mit besonders hohen (Alcohol) und niedrigen (Frozen) Trinkgeldquoten, untersuchen wir nun deren zeitliche Dimension. Diese mehrstufige temporale Analyse betrachtet sowohl Tages- als auch Stundenverläufe, um ein umfassendes Bild der zeitlichen Dynamik zu erhalten.

Zeitliche Dimensionen:

- **Stundenanalyse:** Verlauf über 24 Stunden (0-23 Uhr)
- **Tagesanalyse:** Verteilung über die Wochentage

Analysefokus:

- Vergleich von absoluten Bestellzahlen und Trinkgeldquoten
- Separate Betrachtung der Extremkategorien:
 -  Alcohol (höchste Trinkgeldquote)
 -  Frozen (niedrigste Trinkgeldquote)

Erkenntnisziele:

- Identifikation von Tageszeit- und Wochentagsmustern
- Verständnis kategorienpezifischer Unterschiede im Zeitverlauf
- Erkennung von Stoßzeiten und ruhigen Perioden
- Ableitung von zeit- und kategorieabhängigen Handlungsempfehlungen

Alkoholbestellungen im Tagesverlauf

Nach der Identifikation von Alkoholprodukten als Kategorie mit der höchsten Trinkgeldquote untersuchen wir nun das zeitliche Muster dieser Bestellungen. Diese Analyse soll Aufschluss darüber geben, ob und wie sich das Trinkgeldverhalten bei Alkoholbestellungen im Tagesverlauf verändert.

Visualisierungsdetails:

- **Linker Plot:** Absolute Zahlen der Bestellungen nach Stunden
- **Rechter Plot:** Verlauf der Trinkgeldquote über den Tag
- 24-Stunden-Analyse (0-23 Uhr)
- Berücksichtigung aller Bestellungen mit alkoholischen Produkten

Analyseziele:

- Identifikation von Stoßzeiten für Alkoholbestellungen
- Erkennung von Mustern im Trinkgeldverhalten
- Verständnis des zeitlichen Einflusses auf das Kundenverhalten
- Ableitung möglicher operativer Empfehlungen

```
In [50]: # Bestellungen von alkoholischen Produkten nach Tageszeit: Gesamtbestellung
alcohol_orders_stats = session.query(
    Order.hour_of_day,
    func.count(Order.order_id).label('total_orders'),
    func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_t')
).join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
.join(Product, Product.product_id == Einkaufskorb.product_id) \
.join(Department, Department.department_id == Product.department_id) \
.filter(Department.department_name.ilike('%alcohol%')) \
.group_by(Order.hour_of_day) \
.all()
```

```
In [51]: # plot code
alcohol_orders_df = pd.DataFrame(alcohol_orders_stats, columns=['hour_of_day',
alcohol_orders_df['orders_without_tips'] = alcohol_orders_df['total_order']
alcohol_orders_df['tip_ratio'] = alcohol_orders_df['orders_with_tips'] /

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
plt.subplots_adjust(left=0.08)

bar_with_tips = ax1.bar(alcohol_orders_df['hour_of_day'], alcohol_orders_
                           bar_width, label='Mit Trinkgeld', color='lightgre
bar_without_tips = ax1.bar(alcohol_orders_df['hour_of_day'], alcohol_orde
                           bar_width, bottom=alcohol_orders_df['orders_wit
                           label='Ohne Trinkgeld', color='salmon')

for i, rect in enumerate(bar_with_tips):
    height = rect.get_height() + bar_without_tips[i].get_height()
    ratio = alcohol_orders_df['tip_ratio'].iloc[i]
    ax1.text(rect.get_x() + rect.get_width() / 2, height + 2,
             f'{ratio:.2f}', ha='center', va='bottom')

ax1.set_title('Bestellungen mit und ohne Trinkgeld\nnach Uhrzeit (Alkohol
               loc='left', weight='bold')
ax1.set_xlabel('Uhrzeit (Stunde des Tages)')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.legend()
ax1.grid(True, axis='y', linestyle='--', alpha=0.9)
```

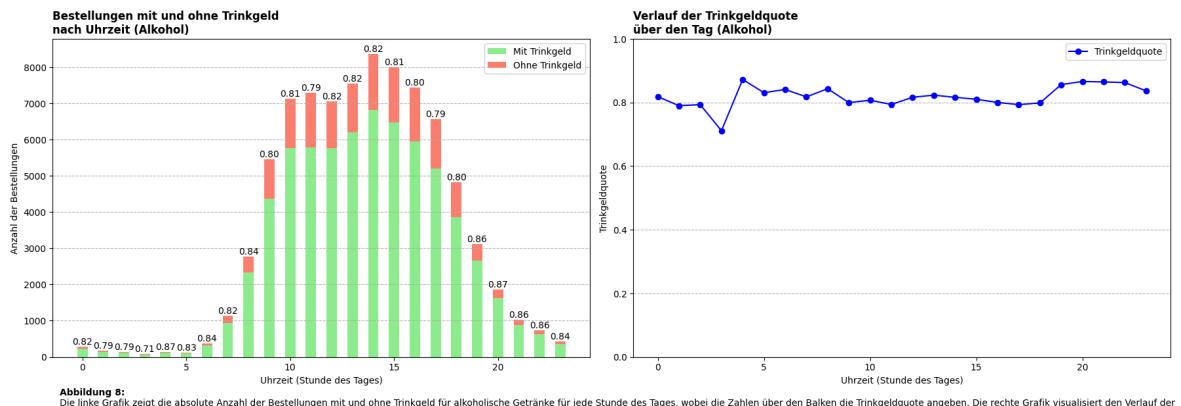
```

ax2.plot(alcohol_orders_df['hour_of_day'], alcohol_orders_df['tip_ratio']
         marker='o', color='blue', label='Trinkgeldquote')
ax2.set_title('Verlauf der Trinkgeldquote über den Tag (Alkohol)',
              loc='left', weight='bold')
ax2.set_xlabel('Uhrzeit (Stunde des Tages)')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_ylim(0, 1)
ax2.legend()
ax2.grid(True, axis='y', linestyle='--', alpha=0.9)

description = 'Die linke Grafik zeigt die absolute Anzahl der Bestellungen mit und ohne Trinkgeld für alkoholische Getränke für jede Stunde des Tages, wobei die Zahlen über den Balken die Trinkgeldquote angeben. Die rechte Grafik visualisiert den Verlauf der Trinkgeldquote für Alkoholbestellungen über den Tag.'
fig.text(0.05, 0, 'Abbildung 8:', weight='bold', ha='left')
fig.text(0.05, -0.05, description, wrap=True)

plt.tight_layout()
plt.show()

```



Erkenntnisse der Stundenanalyse: Alkoholbestellungen



Zeitliche Muster bei Alkoholbestellungen



Bestellvolumen im Tagesverlauf

Die Analyse zeigt einen deutlichen Tagesrhythmus bei Alkoholbestellungen:

- Deutlich höheres Bestellaufkommen während der Tagesstunden
- Signifikanter Rückgang der Bestellungen in den Nachtstunden
- Erkennbares Muster typischen Konsumverhaltens



Stabilität der Trinkgeldquote

Bemerkenswerterweise zeigt die Trinkgeldquote eine hohe Konstanz:

- Kaum Schwankungen im Tagesverlauf
- Stabile Trinkgeldbereitschaft unabhängig von der Tageszeit
- Keine erkennbare Korrelation zwischen Bestellvolumen und Trinkgeldquote

Fazit:

Während das Bestellvolumen deutlichen tageszeitlichen Schwankungen unterliegt, bleibt die Bereitschaft Trinkgeld zu geben bei Alkoholbestellungen konstant hoch. Dies deutet darauf hin, dass die Trinkgeldvergabe bei Alkoholbestellungen weniger von der Tageszeit als vielmehr von der Produktkategorie selbst beeinflusst wird.

```
In [52]: # Vergleich von Gesamtbestellungen und Alkoholbestellungen pro Stunde des
total_orders = session.query(
    Order.hour_of_day,
    func.count(distinct(Order.order_id)).label('total_orders')
).group_by(Order.hour_of_day).subquery()

alcohol_orders = session.query(
    Order.hour_of_day,
    func.count(distinct(Order.order_id)).label('alcohol_orders')
).join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
    .join(Product, Product.product_id == Einkaufskorb.product_id) \
    .join(Department, Department.department_id == Product.department_id) \
    .filter(Department.department_name.ilike('%alcohol%')) \
    .group_by(Order.hour_of_day).subquery()

combined_stats = session.query(
    total_orders.c.hour_of_day,
    total_orders.c.total_orders,
    func.coalesce(alcohol_orders.c.alcohol_orders, 0).label('alcohol_order')
).outerjoin(
    alcohol_orders,
    total_orders.c.hour_of_day == alcohol_orders.c.hour_of_day
).order_by(total_orders.c.hour_of_day).all()

combined_df = pd.DataFrame(combined_stats, columns=['hour_of_day', 'total_orders'])
combined_df['alcohol_ratio'] = combined_df['alcohol_orders'] / combined_df['total_orders']
```



Einschub: Alkoholanteil im Bestellungsmix



Relativer Anteil von Alkoholbestellungen

Um ein vollständigeres Bild der Alkoholbestellungen zu erhalten, betrachten wir ergänzend den relativen Anteil alkoholhaltiger Bestellungen am gesamten Bestellvolumen im Tagesverlauf. Diese Analyse hilft uns zu verstehen, ob Alkoholbestellungen zu bestimmten Tageszeiten überproportional vertreten sind.

Analysefokus:

- Verhältnis: Bestellungen mit Alkohol / Gesamtbestellungen
- Stundenweise Betrachtung über 24 Stunden

- Prozentuale Darstellung der Anteile

```
In [53]: # plot code
fig, ax = plt.subplots(figsize=(10, 6))
plt.subplots_adjust(left=0.08)

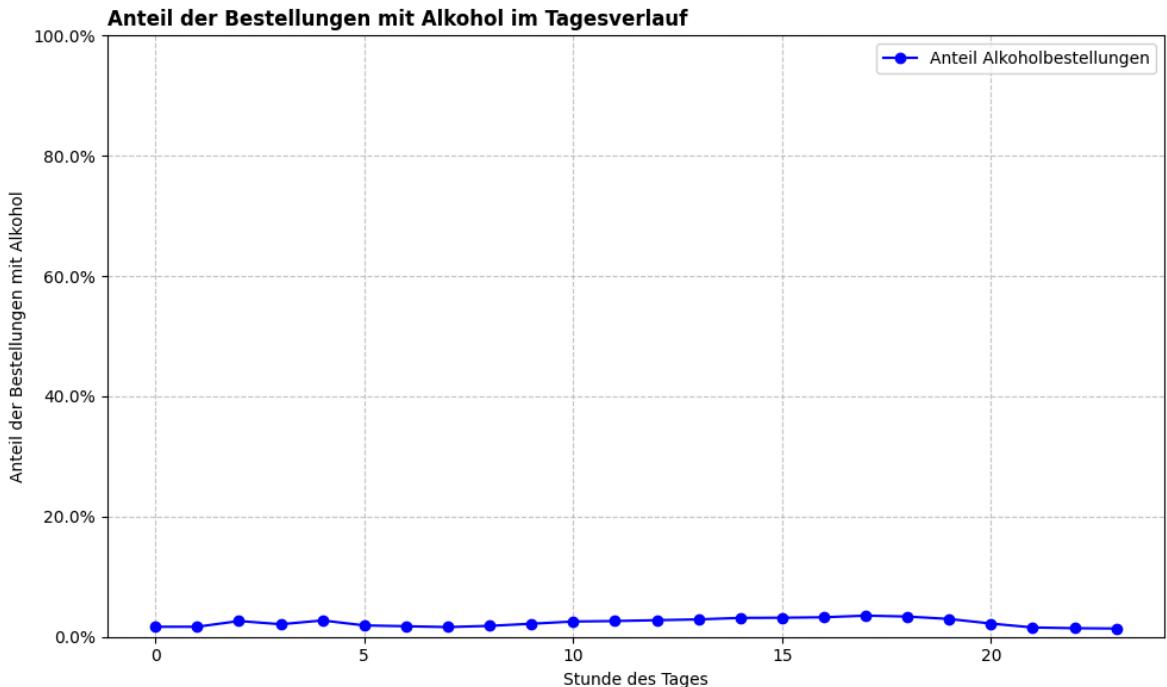
ax.plot(combined_df['hour_of_day'], combined_df['alcohol_ratio'],
        marker='o', color='blue', label='Anteil Alkoholbestellungen')

ax.set_title('Anteil der Bestellungen mit Alkohol im Tagesverlauf',
             loc='left', weight='bold')
ax.set_xlabel('Stunde des Tages')
ax.set_ylabel('Anteil der Bestellungen mit Alkohol')
ax.grid(True, linestyle='--', alpha=0.7)
ax.legend()

ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda y, _: '{:.1%}'.format(y)))
ax.set_ylim(0, 1)

description = 'Die Grafik zeigt den prozentualen Anteil der Bestellungen, die alkoholische Getränke enthalten, im Verlauf des Tages. Die Werte sind als Prozentsatz aller Bestellungen in der jeweiligen Stunde dargestellt.'
fig.text(0.08, 0, 'Abbildung 9:', weight='bold', ha='left')
fig.text(0.08, -0.05, description, wrap=True)

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

**Abbildung 9:**

Die Grafik zeigt den prozentualen Anteil der Bestellungen, die alkoholische Getränke enthalten, im Verlauf des Tages. Die Werte sind als Prozentsatz aller Bestellungen in der jeweiligen Stunde dargestellt.



Erkenntnisse zum Bestellmix



Konstanz im Bestellverhalten

Analyse des relativen Anteils

Die Untersuchung des relativen Anteils von Alkoholbestellungen zeigt ein bemerkenswertes Muster:

- Weitgehend konstanter Anteil über den gesamten Tagesverlauf
- Nur minimale Schwankungen ohne erkennbare zeitliche Systematik
- Keine signifikanten Ausreißer zu bestimmten Tageszeiten

Fazit:

Die Stabilität des relativen Anteils deutet darauf hin, dass Alkoholbestellungen ein konstanter Bestandteil des Bestellmix sind. Während die absolute Anzahl der Bestellungen im Tagesverlauf variiert, bleibt der proportionale Anteil von Alkoholbestellungen bemerkenswert stabil.

```
In [54]: # Bestellungen von alkoholischen Produkten nach Wochentag: Gesamtbestellung

alcohol_orders_weekday_stats = session.query(
    Order.day_of_the_week,
    func.count(Order.order_id).label('total_orders'),
    func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_tips'))
    .join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
    .join(Product, Product.product_id == Einkaufskorb.product_id) \
    .join(Department, Department.department_id == Product.department_id) \
    .filter(Department.department_name.ilike('%alcohol%')) \
    .group_by(Order.day_of_the_week) \
    .all()
```

Wochenanalyse der Alkoholbestellungen

Nach der Betrachtung des Tagesverlaufs erweitern wir unsere zeitliche Analyse auf die Wochentage. Dies ermöglicht uns, potenzielle Unterschiede zwischen Werktagen und Wochenenden sowie wöchentliche Muster im Bestell- und Trinkgeldverhalten bei Alkoholprodukten zu identifizieren.

```
In [55]: # plot code
alcohol_orders_weekday_df = pd.DataFrame(alcohol_orders_weekday_stats, columns=['day_of_the_week', 'total_orders', 'orders_with_tips'])

alcohol_orders_weekday_df['orders_without_tips'] = alcohol_orders_weekday_df['total_orders'] - alcohol_orders_weekday_df['orders_with_tips']
alcohol_orders_weekday_df['tip_ratio'] = alcohol_orders_weekday_df['orders_with_tips'] / alcohol_orders_weekday_df['total_orders']

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
plt.subplots_adjust(left=0.08)
```

```

bar_width = 0.35

bar_with_tips = ax1.bar(alcohol_orders_weekday_df['day_of_the_week'],
                       alcohol_orders_weekday_df['orders_with_tips'],
                       bar_width, label='Mit Trinkgeld', color='lightgreen')
bar_without_tips = ax1.bar(alcohol_orders_weekday_df['day_of_the_week'],
                           alcohol_orders_weekday_df['orders_without_tips'],
                           bar_width, bottom=alcohol_orders_weekday_df['orders_with_tips'],
                           label='Ohne Trinkgeld', color='salmon')

for i, rect in enumerate(bar_with_tips):
    height = rect.get_height() + bar_without_tips[i].get_height()
    ratio = alcohol_orders_weekday_df['tip_ratio'].iloc[i]
    ax1.text(rect.get_x() + rect.get_width() / 2, height + 2,
             f'{ratio:.2f}', ha='center', va='bottom')

ax1.set_title('Bestellungen mit und ohne Trinkgeld nach Wochentag (Alkohol)', loc='left', weight='bold')
ax1.set_xlabel('Wochentag')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.legend()
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)

ax2.plot(alcohol_orders_weekday_df['day_of_the_week'],
         alcohol_orders_weekday_df['tip_ratio'],
         marker='o', color='blue', label='Trinkgeldquote')
ax2.set_title('Verlauf der Trinkgeldquote nach Wochentag (Alkohol)', loc='left', weight='bold')
ax2.set_xlabel('Wochentag')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_ylim(0, 1)
ax2.legend()
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)

description = 'Die linke Grafik zeigt die absolute Anzahl der Alkoholbestellungen mit und ohne Trinkgeld für jeden Wochentag, wobei die Zahlen über den Balken die Trinkgeldquote angeben. Die rechte Grafik visualisiert den Verlauf der Trinkgeldquote für Alkoholbestellungen im Wochenverlauf.'

plt.tight_layout()
plt.xticks(rotation=45)
plt.show()

```

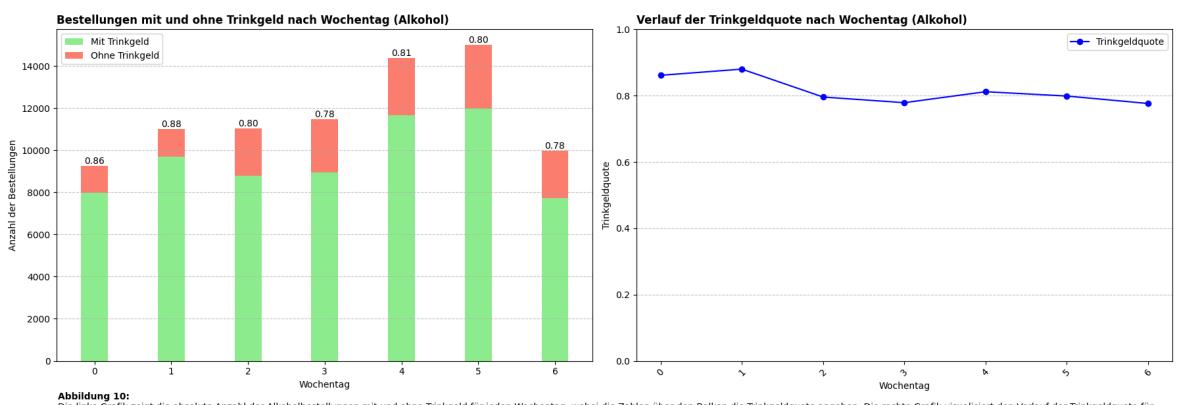


Abbildung 10:
Die linke Grafik zeigt die absolute Anzahl der Alkoholbestellungen mit und ohne Trinkgeld für jeden Wochentag, wobei die Zahlen über den Balken die Trinkgeldquote angeben. Die rechte Grafik visualisiert den Verlauf der Trinkgeldquote für Alkoholbestellungen im Wochenverlauf.



Erkenntnisse der Wochenanalyse



Wöchentliche Muster bei Alkoholbestellungen

Trinkgeldverhalten (Tag 0-1)

Zu Beginn der Woche zeigt sich eine erhöhte Trinkgeldbereitschaft:

- Höhere Trinkgeldquote an den ersten beiden Wochentagen
- Möglicherweise erhöhte Kundenfreundlichkeit nach dem Wochenende

Bestellvolumen (Tag 4-5)

Zum Ende der Woche ist ein deutlicher Anstieg der Bestellmenge erkennbar:

- Spurze im Bestellvolumen an Tag 4 und 5
- Typisches Muster für Wochenend-Einkäufe alkoholischer Getränke

```
In [56]: # Detaillierte Trinkgeld-Analyse für Alkoholbestellungen: Anzahl und Verh
results = (
    session.query(
        Order.day_of_the_week,
        Order.hour_of_day,
        func.sum(case((Order.tips == True, 1), else_=0)).label('orders_wi
        func.count(Order.order_id).label('total_orders'),
        (func.sum(case((Order.tips == True, 1), else_=0)) / func.count(Or
    )
    .join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id) # Verkn
    .join(Product, Einkaufskorb.product_id == Product.product_id) # Verk
    .join(Department, Product.department_id == Department.department_id)
    .filter(Department.department_name == 'alcohol') # Filter auf Alkoho
    .group_by(Order.day_of_the_week, Order.hour_of_day)
    .order_by(Order.day_of_the_week, Order.hour_of_day)
    .all()
)

df = pd.DataFrame(results, columns=['day_of_the_week', 'hour_of_day', 'or
df['tip_ratio'] = pd.to_numeric(df['tip_ratio'], errors='coerce')

all_days = range(7) # 0-6
all_hours = range(24) # 0-23
full_index = pd.DataFrame(itertools.product(all_days, all_hours), columns

df = full_index.merge(df, on=['day_of_the_week', 'hour_of_day'], how='lef
df['tip_ratio'] = df['tip_ratio'].fillna(0)

heatmap_data = df.pivot_table(
    index='day_of_the_week',
    columns='hour_of_day',
    values='tip_ratio'
)

total_orders_heatmap = df.pivot_table(
    index='day_of_the_week',
    columns='hour_of_day',
```

```

        values='total_orders'
    )

orders_with_tips_heatmap = df.pivot_table(
    index='day_of_the_week',
    columns='hour_of_day',
    values='orders_with_tips'
)

```

🌡️ Kombinierte Zeitanalyse: Heatmap-Visualisierung

🔍 Zweidimensionale Zeitanalyse

Nach der separaten Betrachtung von Tages- und Wochenverlauf kombinieren wir nun beide zeitliche Dimensionen in einer Heatmap. Diese Visualisierung ermöglicht es uns, spezifische Zeitfenster mit besonders hoher oder niedriger Trinkgeldquote zu identifizieren.

📊 Visualisierungsdetails:

- **X-Achse:** Stunden des Tages (0-23)
- **Y-Achse:** Wochentage
- **Farbskala:**
 - Rot = Hohe Trinkgeldquote
 - Blau = Niedrige Trinkgeldquote

In [57]: # plot code

```

fig, ax = plt.subplots(figsize=(10, 7))
plt.subplots_adjust(left=0.08)

sns.heatmap(
    heatmap_data,
    annot=True,
    fmt=".2f",
    cmap="coolwarm",
    cbar_kws={'label': 'Trinkgeldquote'},
    ax=ax
)

ax.set_title('Trinkgeldquote nach Wochentag und Stunde',
            loc='left',
            weight='bold',
            fontsize=16)
ax.set_xlabel('Stunde des Tages', fontsize=12)
ax.set_ylabel('Wochentag', fontsize=12)

description = 'Die Heatmap visualisiert die Trinkgeldquote für verschiedene'
fig.text(0.05, -0, 'Abbildung 11:', weight='bold', ha='left')
fig.text(0.05, -0.07, description, wrap=True)

```

```
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

```
total_orders_heatmap
```

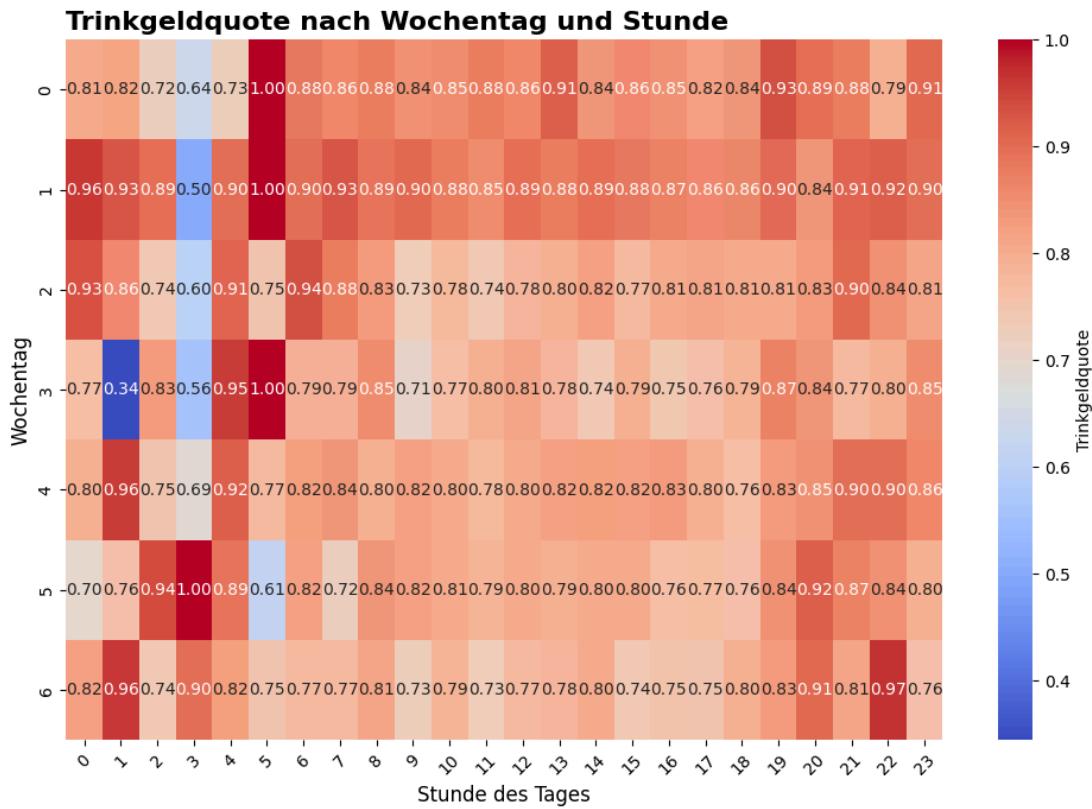


Abbildung 11:

Die Heatmap visualisiert die Trinkgeldquote für verschiedene Kombinationen von Wochentagen und Tageszeiten. Rötere Farben zeigen eine höhere Trinkgeldquote an, während bläuerne Farben eine niedrigere Quote anzeigen. Die Zahlen in den Zellen geben die genaue Trinkgeldquote an.

Out[57]:

	hour_of_day	0	1	2	3	4	5	6	7	8	9	...	
day_of_the_week	0	36.0	22.0	18.0	11.0	22.0	19.0	52.0	135.0	296.0	611.0	...	10
1	27.0	27.0	19.0	14.0	10.0	10.0	40.0	127.0	480.0	837.0	...	10	
2	44.0	21.0	19.0	10.0	11.0	16.0	48.0	178.0	371.0	717.0	...	10	
3	30.0	29.0	18.0	9.0	22.0	19.0	67.0	153.0	343.0	764.0	...	11	
4	49.0	23.0	20.0	16.0	24.0	22.0	57.0	172.0	450.0	1003.0	...	14	
5	50.0	29.0	17.0	10.0	27.0	18.0	56.0	245.0	487.0	1002.0	...	16	
6	39.0	25.0	34.0	20.0	17.0	20.0	57.0	135.0	340.0	535.0	...	10	

7 rows × 24 columns



Erkenntnisse der Heatmap-Analyse



Identifizierte Muster und Einschränkungen



Einschränkungen in der Nachtzeit

Die Nachtstunden zeigen starke Schwankungen in der Trinkgeldquote, die jedoch aufgrund des geringen Bestellvolumens statistisch wenig aussagekräftig sind:

- Hohe Variabilität durch kleine Stichprobengrößen
- Zufallsbedingte Ausschläge in den Nachtstunden
- Eingeschränkte Interpretierbarkeit der nächtlichen Werte

Stabile Muster

Zuverlässige Erkenntnisse lassen sich aus den Tagesstunden und dem Wochenverlauf ableiten:

- Erhöhte Trinkgeldquote an den Tagen 0 und 1
- Stabilere Quoten während der Tagesstunden
- Konsistenteres Muster im regulären Tagesgeschäft

Fazit:

Die aussagekräftigsten Muster zeigen sich in den Tagesstunden, während die Nachtzeit aufgrund geringer Bestellzahlen keine verlässlichen Schlüsse zulässt. Die Wochentagseffekte sind besonders zu Beginn der Woche deutlich erkennbar.

```
In [58]: # Bestellungen von Tiefkühlprodukten nach Tageszeit: Gesamtbestellungen u
frozen_orders_stats = session.query(
    Order.hour_of_day,
    func.count(Order.order_id).label('total_orders'),
    func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_t
).join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
.join(Product, Product.product_id == Einkaufskorb.product_id) \
.join(Department, Department.department_id == Product.department_id) \
.filter(Department.department_name.ilike('%frozen%')) \
.group_by(Order.hour_of_day) \
.all()
```

Zeitliche Analyse der Frozen-Kategorie

Vergleichsanalyse: Frozen Department

Nach der detaillierten Analyse der Alkoholkategorie wenden wir nun die gleiche methodische Vorgehensweise auf das Frozen Department an. Als Kategorie mit der niedrigsten Trinkgeldquote bildet sie einen interessanten Gegenpol zu den Alkoholbestellungen.

Analysestruktur:

- Stundenweise Analyse des Bestellverhaltens
- Wöchentliche Verteilung der Bestellungen
- Kombinierte Betrachtung in Form einer Heatmap

Analyseziel:

Identifikation möglicher zeitlicher Muster, die die niedrige Trinkgeldquote dieser Kategorie erklären könnten, sowie Vergleich mit den Erkenntnissen aus der Alkohol-Analyse.

In [59]:

```
# plot code
frozen_orders_df = pd.DataFrame(frozen_orders_stats, columns=['hour_of_day'])
frozen_orders_df['orders_without_tips'] = frozen_orders_df['total_orders'] - frozen_orders_df['orders_with_tips']
frozen_orders_df['tip_ratio'] = frozen_orders_df['orders_with_tips'] / frozen_orders_df['total_orders']

# Subplots erstellen
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
plt.subplots_adjust(left=0.08)

# Linker Plot: Gestapeltes Balkendiagramm
bar_with_tips = ax1.bar(frozen_orders_df['hour_of_day'], frozen_orders_df['orders_with_tips'],
                        bar_width, label='Mit Trinkgeld', color='lightgreen')
bar_without_tips = ax1.bar(frozen_orders_df['hour_of_day'], frozen_orders_df['orders_without_tips'],
                           bar_width, bottom=frozen_orders_df['orders_with_tips'],
                           label='Ohne Trinkgeld', color='salmon')

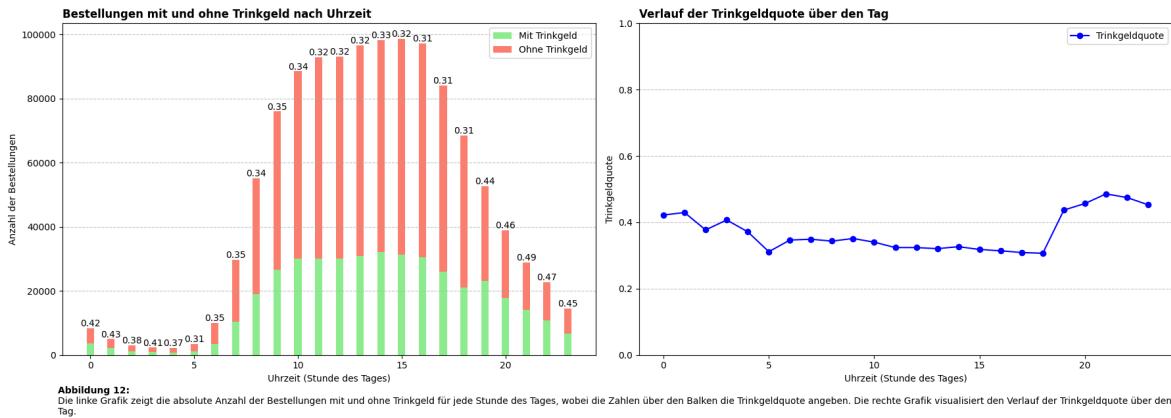
# Quotienten oben auf den Balken im linken Plot anzeigen
for i, rect in enumerate(bar_with_tips):
    height = rect.get_height() + bar_without_tips[i].get_height()
    ratio = frozen_orders_df['tip_ratio'].iloc[i]
    ax1.text(rect.get_x() + rect.get_width() / 2, height + 2,
             f'{ratio:.2f}', ha='center', va='bottom')

# Titel und Achsenbeschriftungen für linken Plot
ax1.set_title('Bestellungen mit und ohne Trinkgeld nach Uhrzeit',
              loc='left', weight='bold')
ax1.set_xlabel('Uhrzeit (Stunde des Tages)')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.legend()
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)

# Rechter Plot: Liniendiagramm der Trinkgeldquote
ax2.plot(frozen_orders_df['hour_of_day'], frozen_orders_df['tip_ratio'],
          marker='o', color='blue', label='Trinkgeldquote')
ax2.set_title('Verlauf der Trinkgeldquote über den Tag',
              loc='left', weight='bold')
ax2.set_xlabel('Uhrzeit (Stunde des Tages)')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_ylim(0, 1)
ax2.legend()
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)
```

```
# Bildunterschrift
description = 'Die linke Grafik zeigt die absolute Anzahl der Bestellungen mit und ohne Trinkgeld für jede Stunde des Tages, wobei die Zahlen über den Balken die Trinkgeldquote angeben. Die rechte Grafik visualisiert den Verlauf der Trinkgeldquote über den Tag.'
fig.text(0.05, 0, 'Abbildung 12:', weight='bold', ha='left')
fig.text(0.05, -0.05, description, wrap=True)

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



In [60]: # Vergleich von Gesamtbestellungen und Tiefkühlprodukt-Bestellungen pro Stunde

```
total_orders = session.query(
    Order.hour_of_day,
    func.count(distinct(Order.order_id)).label('total_orders'))
    .group_by(Order.hour_of_day).subquery()

frozen_orders = session.query(
    Order.hour_of_day,
    func.count(distinct(Order.order_id)).label('frozen_orders'))
    .join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
    .join(Product, Product.product_id == Einkaufskorb.product_id) \
    .join(Department, Department.department_id == Product.department_id) \
    .filter(Department.department_name.ilike('%frozen%')) \
    .group_by(Order.hour_of_day).subquery()

combined_stats = session.query(
    total_orders.c.hour_of_day,
    total_orders.c.total_orders,
    func.coalesce(frozen_orders.c.frozen_orders, 0).label('frozen_orders'))
    .outerjoin(
        frozen_orders,
        total_orders.c.hour_of_day == frozen_orders.c.hour_of_day)
    .order_by(total_orders.c.hour_of_day).all()
```

In [61]: # plot code

```
combined_df = pd.DataFrame(combined_stats, columns=['hour_of_day', 'total_orders'])
combined_df['frozen_ratio'] = combined_df['frozen_orders'] / combined_df['total_orders']

fig, ax = plt.subplots(figsize=(10, 6))
plt.subplots_adjust(left=0.08)

ax.plot(combined_df['hour_of_day'], combined_df['frozen_ratio'],
        marker='o', color='blue', label='Anteil Tiefkühlbestellungen')

ax.set_title('Anteil der Bestellungen mit Tiefkühlprodukten\nim Tagesverlauf')
ax.set_xlabel('Stunde des Tages')
```

```

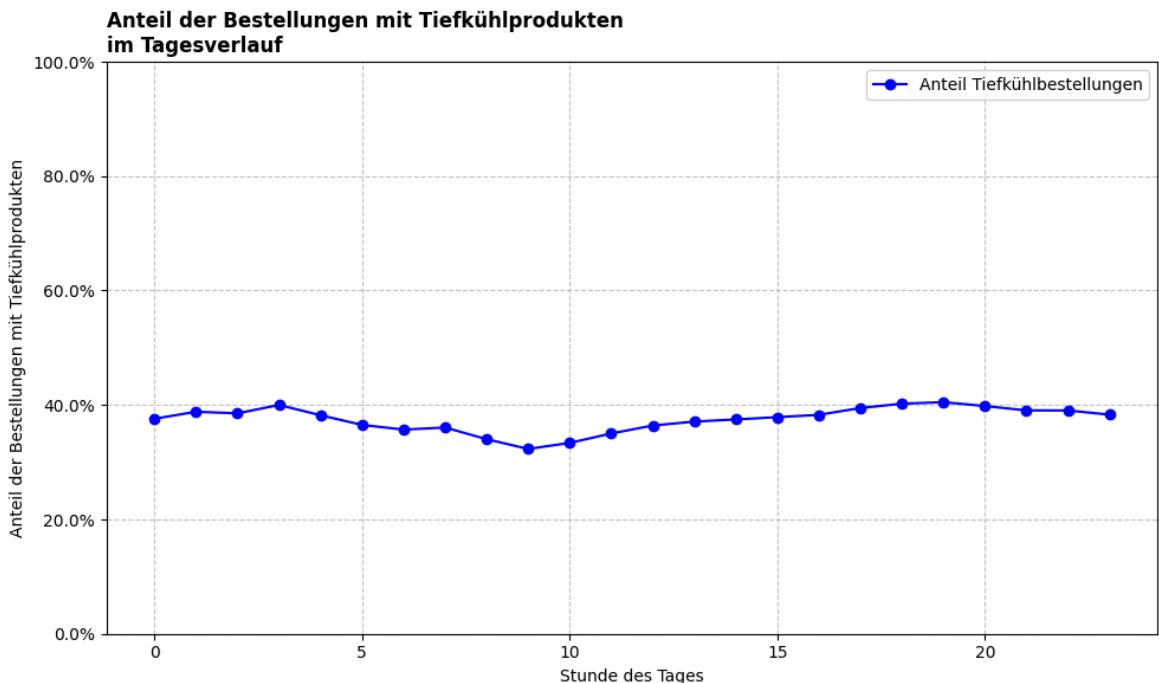
ax.set_ylabel('Anteil der Bestellungen mit Tiefkühlprodukten')
ax.grid(True, linestyle='--', alpha=0.7)
ax.legend()

ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda y, _: '{:.1%}'.format(y)))
ax.set_ylim(0, 1)

description = 'Die Grafik zeigt den prozentualen Anteil der Bestellungen, die Tiefkühlprodukte enthalten, im Verlauf des Tages. Die Werte sind als Prozentsatz aller Bestellungen in der jeweiligen Stunde dargestellt.'
fig.text(0.08, 0, 'Abbildung 13:', weight='bold', ha='left')
fig.text(0.08, -0.05, description, wrap=True)

plt.tight_layout()
plt.show()

```

**Abbildung 13:**

Die Grafik zeigt den prozentualen Anteil der Bestellungen, die Tiefkühlprodukte enthalten, im Verlauf des Tages. Die Werte sind als Prozentsatz aller Bestellungen in der jeweiligen Stunde dargestellt.

In [62]: # Bestellungen von Tiefkühlprodukten nach Wochentag: Gesamtbestellungen und Anzahl der Bestellungen mit Tiefkühlprodukten

```

frozen_orders_weekday_stats = session.query(
    Order.day_of_the_week,
    func.count(Order.order_id).label('total_orders'),
    func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_tips'))
    .join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
    .join(Product, Product.product_id == Einkaufskorb.product_id) \
    .join(Department, Department.department_id == Product.department_id) \
    .filter(Department.department_name.ilike('%frozen%')) \
    .group_by(Order.day_of_the_week) \
    .all()

```

In [63]: # plot code

```

frozen_orders_weekday_df = pd.DataFrame(frozen_orders_weekday_stats, columns=['day_of_the_week', 'total_orders', 'orders_with_tips'])

frozen_orders_weekday_df['orders_without_tips'] = frozen_orders_weekday_df['total_orders'] - frozen_orders_weekday_df['orders_with_tips']
frozen_orders_weekday_df['tip_ratio'] = frozen_orders_weekday_df['orders_with_tips'] / frozen_orders_weekday_df['total_orders']

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
plt.subplots_adjust(left=0.08)

bar_width = 0.35

```

```

bar_with_tips = ax1.bar(frozen_orders_weekday_df['day_of_the_week'],
                       frozen_orders_weekday_df['orders_with_tips'],
                       bar_width, label='Mit Trinkgeld', color='lightgreen')
bar_without_tips = ax1.bar(frozen_orders_weekday_df['day_of_the_week'],
                           frozen_orders_weekday_df['orders_without_tips'],
                           bar_width, bottom=frozen_orders_weekday_df['orders_with_tips'],
                           label='Ohne Trinkgeld', color='salmon')

for i, rect in enumerate(bar_with_tips):
    height = rect.get_height() + bar_without_tips[i].get_height()
    ratio = frozen_orders_weekday_df['tip_ratio'].iloc[i]
    ax1.text(rect.get_x() + rect.get_width() / 2, height + 2,
              f'{ratio:.2f}', ha='center', va='bottom')

ax1.set_title('Bestellungen mit und ohne Trinkgeld nach Wochentag (Tiefkühlprodukte)')
ax1.set_xlabel('Wochentag')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.legend()
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)

ax2.plot(frozen_orders_weekday_df['day_of_the_week'],
          frozen_orders_weekday_df['tip_ratio'],
          marker='o', color='blue', label='Trinkgeldquote')
ax2.set_title('Verlauf der Trinkgeldquote nach Wochentag (Tiefkühlprodukte)')
ax2.set_xlabel('Wochentag')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_ylim(0, 1)
ax2.legend()
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)

description = 'Die linke Grafik zeigt die absolute Anzahl der Bestellungen mit und ohne Trinkgeld für jeden Wochentag. Die rechte Grafik visualisiert den Verlauf der Trinkgeldquote für Bestellungen mit Tiefkühlprodukten im Wochenverlauf.'
fig.text(0.05, 0, 'Abbildung 14:', weight='bold', ha='left')
fig.text(0.05, -0.05, description, wrap=True)

plt.tight_layout()
plt.xticks(rotation=45)
plt.show()

```

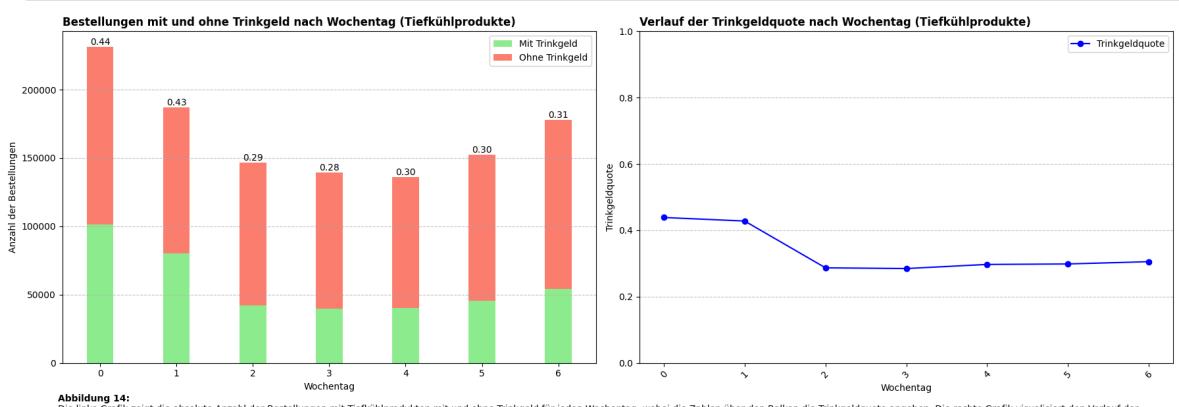


Abbildung 14:
Die linke Grafik zeigt die absolute Anzahl der Bestellungen mit und ohne Trinkgeld für jeden Wochentag, wobei die Zahlen über den Balken die Trinkgeldquote angeben. Die rechte Grafik visualisiert den Verlauf der Trinkgeldquote für Bestellungen mit Tiefkühlprodukten im Wochenverlauf.

In [64]: # Vergleich von Gesamtbestellungen und Tiefkühlprodukt-Bestellungen pro Wochentag

```

total_orders = session.query(
    Order.day_of_the_week,
    func.count(distinct(Order.order_id)).label('total_orders')
).group_by(Order.day_of_the_week).subquery()

```

```

frozen_orders = session.query(
    Order.day_of_the_week,
    func.count(distinct(Order.order_id)).label('frozen_orders')
).join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
.join(Product, Product.product_id == Einkaufskorb.product_id) \
.join(Department, Department.department_id == Product.department_id) \
.filter(Department.department_name.ilike('%frozen%')) \
.group_by(Order.day_of_the_week).subquery()

combined_stats = session.query(
    total_orders.c.day_of_the_week,
    total_orders.c.total_orders,
    func.coalesce(frozen_orders.c.frozen_orders, 0).label('frozen_orders')
).outerjoin(
    frozen_orders,
    total_orders.c.day_of_the_week == frozen_orders.c.day_of_the_week
).order_by(total_orders.c.day_of_the_week).all()

combined_df = pd.DataFrame(combined_stats, columns=['day_of_week', 'total_orders'])
combined_df['frozen_ratio'] = combined_df['frozen_orders'] / combined_df['total_orders']

```

In [65]:

```

# plot code
fig, ax = plt.subplots(figsize=(12, 6))
plt.subplots_adjust(left=0.08)

ax.plot(combined_df['day_of_week'], combined_df['frozen_ratio'],
        marker='o', color='blue', label='Anteil Tiefkühlprodukte')

ax.set_title('Anteil der Bestellungen mit Tiefkühlprodukten im Wochenverlauf',
             loc='left', weight='bold')
ax.set_xlabel('Tag der Woche')
ax.set_ylabel('Anteil der Bestellungen mit Tiefkühlprodukten')
ax.grid(True, linestyle='--', alpha=0.7)
ax.legend()

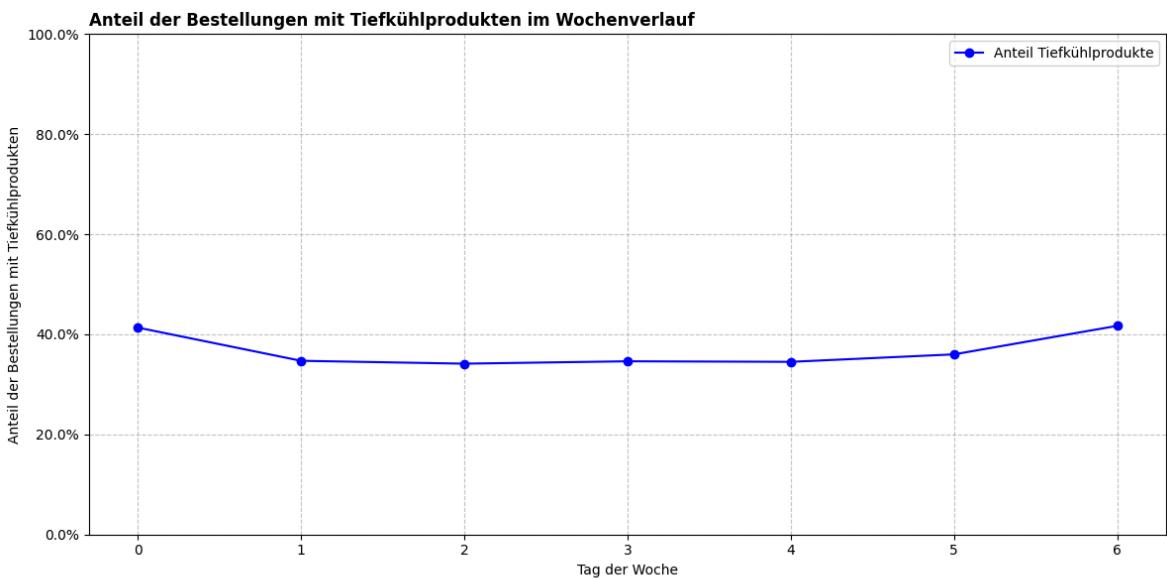
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda y, _: '{:.1%}'.format(y)))
ax.set_ylim(0, 1)

ax.set_xticks(range(7))

description = 'Die Grafik zeigt den prozentualen Anteil der Bestellungen, die Tiefkühlprodukte enthalten. Die X-Achse stellt die Tage der Woche dar, von Montag bis Sonntag. Der Y-Achse ist der prozentuale Anteil der Bestellungen mit Tiefkühlprodukten. Die Legende markiert den blauen Kreis als "Anteil Tiefkühlprodukte".'
fig.text(0.08, 0, 'Abbildung 15:', weight='bold', ha='left')
fig.text(0.08, -0.05, description, wrap=True)

plt.tight_layout()
plt.show()

```

**Abbildung 15:**

Die Grafik zeigt den prozentualen Anteil der Bestellungen, die Tiefkühlprodukte enthalten, im Verlauf der Woche. Die Werte sind als Prozentsatz aller Bestellungen am jeweiligen Wochentag dargestellt.

In [66]: # Detaillierte Trinkgeld-Analyse für Tiefkühlprodukte: Anzahl und Verhältnis

```
results = (
    session.query(
        Order.day_of_the_week,
        Order.hour_of_day,
        func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_tips'),
        func.count(Order.order_id).label('total_orders'),
        (func.sum(case((Order.tips == True, 1), else_=0)) / func.count(Order.order_id))
    )
    .join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id) # Verknüpfung
    .join(Product, Einkaufskorb.product_id == Product.product_id) # Verknüpfung
    .join(Department, Product.department_id == Department.department_id)
    .filter(Department.department_name == 'frozen') # Filter auf Alkohol
    .group_by(Order.day_of_the_week, Order.hour_of_day)
    .order_by(Order.day_of_the_week, Order.hour_of_day)
    .all()
)
```

In [67]: # plot code

```
df = pd.DataFrame(results, columns=['day_of_the_week', 'hour_of_day', 'orders_with_tips'])
df['tip_ratio'] = pd.to_numeric(df['tip_ratio'], errors='coerce')

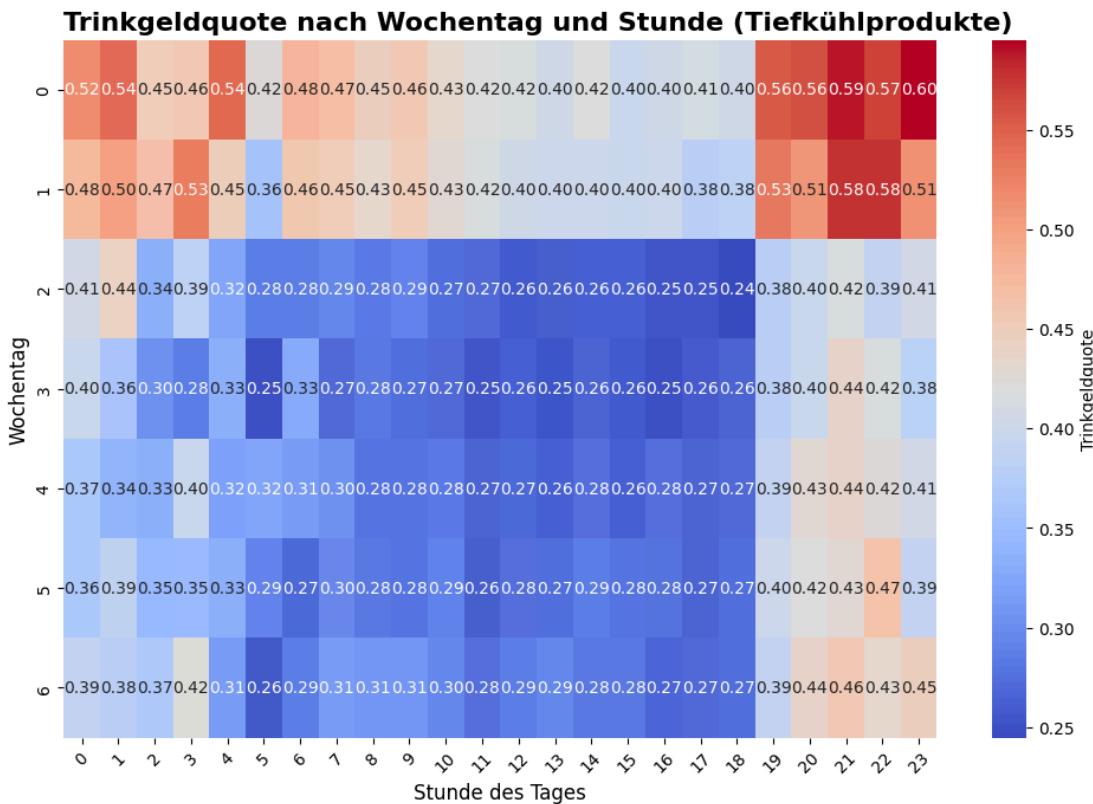
all_days = range(7) # 0–6
all_hours = range(24) # 0–23
full_index = pd.DataFrame(itertools.product(all_days, all_hours), columns=['day_of_the_week', 'hour_of_day'])

df = full_index.merge(df, on=['day_of_the_week', 'hour_of_day'], how='left')
df['tip_ratio'] = df['tip_ratio'].fillna(0)

heatmap_data = df.pivot_table(
    index='day_of_the_week',
    columns='hour_of_day',
    values='tip_ratio'
)

total_orders_heatmap = df.pivot_table(
    index='day_of_the_week',
    columns='hour_of_day',
    values='total_orders'
)
```

```
orders_with_tips_heatmap = df.pivot_table(  
    index='day_of_the_week',  
    columns='hour_of_day',  
    values='orders_with_tips'  
)  
  
fig, ax = plt.subplots(figsize=(10, 7))  
plt.subplots_adjust(left=0.08)  
  
sns.heatmap(  
    heatmap_data,  
    annot=True,  
    fmt=".2f",  
    cmap="coolwarm",  
    cbar_kws={'label': 'Trinkgeldquote'},  
    ax=ax  
)  
  
ax.set_title('Trinkgeldquote nach Wochentag und Stunde (Tiefkühlprodukte)',  
            loc='left',  
            weight='bold',  
            fontsize=16)  
ax.set_xlabel('Stunde des Tages', fontsize=12)  
ax.set_ylabel('Wochentag', fontsize=12)  
  
description = 'Die Heatmap visualisiert die Trinkgeldquote für Bestellung  
fig.text(0.05, 0, 'Abbildung 16:', weight='bold', ha='left')  
fig.text(0.05, -0.065, description, wrap=True)  
  
plt.xticks(rotation=45)  
plt.tight_layout()  
plt.show()  
  
total_orders_heatmap
```

**Abbildung 16:**

Die Heatmap visualisiert die Trinkgeldquote für Bestellungen mit Tiefkühlprodukten für verschiedene Kombinationen von Wochentagen und Tageszeiten. Rötere Farben zeigen eine höhere Trinkgeldquote an, während bläuer Farben eine niedrigere Quote anzeigen. Die Zahlen in den Zellen geben die genaue Trinkgeldquote an.

Out[67]:

	hour_of_day	0	1	2	3	4	5	6	7	8
day_of_the_week		0	1	2	3	4	5	6	7	8
Mo	0	1454.0	915.0	550.0	406.0	301.0	454.0	1380.0	4633.0	10573.0
Di	1	1294.0	735.0	402.0	292.0	278.0	480.0	1560.0	4855.0	9083.0
Mi	2	1096.0	621.0	333.0	283.0	313.0	478.0	1328.0	3928.0	6817.0
Fr	3	1040.0	658.0	408.0	232.0	299.0	456.0	1420.0	3832.0	6507.0
So	4	995.0	520.0	318.0	273.0	241.0	461.0	1360.0	3982.0	6219.0
Sa	5	1089.0	637.0	480.0	409.0	411.0	622.0	1645.0	4373.0	7194.0
Su	6	1322.0	856.0	450.0	377.0	331.0	504.0	1311.0	4093.0	8677.0

7 rows × 24 columns



Zentrale Erkenntnisse der Frozen-Analyse



Zeitliche Muster im Frozen Department



Im Tagesverlauf zeigen sich interessante Muster:

- Höheres Bestellaufkommen während der Tagesstunden, ähnlich wie bei Alkohol
- Überraschend erhöhte Trinkgeldquote in den Nachtstunden

- Leicht erhöhter relativer Anteil von Frozen-Produkten in Nachtbestellungen

Wochenverlauf

Im Wochenverlauf sind deutliche Variationen erkennbar:

- Erhöhte Trinkgeldquote an den Tagen 0 und 1
- Gesteigerter Anteil von Frozen-Bestellungen an den Tagen 6 und 0
- Niedrigere Trinkgeldquoten im mittleren Wochenverlauf

Heatmap-Erkenntnisse

Die kombinierte Zeit-Analyse zeigt ein besonders markantes Muster:

- Deutlich erhöhte Trinkgeldquote in den Nachtstunden der Tage 0 und 1
- Kontrastierend niedrigere Quoten an den Tagen 2-6 während der Tagesstunden
- Klare zeitliche und wöchentliche Strukturierung des Trinkgeldverhaltens

```
In [68]: # Query: Anzahl der Produkte pro Bestellung und ob Trinkgeld gegeben wurde
all_orders_item_count_with_tips = session.query(
    Order.order_id,
    func.count(Einkaufskorb.product_id).label('item_count'),
    Order.tips,
    func.count(Order.order_id).label('total_orders')
).join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
.join(Product, Product.product_id == Einkaufskorb.product_id) \
.group_by(Order.order_id) \
.all()
```

Analyse des Zusammenhangs zwischen Bestellgröße und Trinkgeldverhalten

Einfluss der Produktanzahl auf Trinkgeld

Nach den zeitlichen Analysen untersuchen wir nun den Zusammenhang zwischen der Anzahl der Produkte in einer Bestellung und der Wahrscheinlichkeit einer Trinkgeldgabe. Diese Analyse soll Aufschluss darüber geben, ob größere Bestellungen zu einem anderen Trinkgeldverhalten führen.

Visualisierungsdetails:

- **Linker Plot:** Absolute Verteilung der Bestellungen nach Produktanzahl
- **Rechter Plot:** Trinkgeldquote in Abhängigkeit von der Bestellgröße
- **Besonderheit:** Markierung der Grenze, ab der die Stichprobengröße für statistisch belastbare Aussagen zu klein wird (< 500 Bestellungen)

```
In [69]: # plot code
results = all_orders_item_count_with_tips

df = pd.DataFrame(results, columns=['order_id', 'item_count', 'tips', 'to
df['with_tips'] = df['tips'].apply(lambda x: 1 if x else 0)
df['without_tips'] = df['tips'].apply(lambda x: 0 if x else 1)

df = df[df['item_count'] <= 50]

agg_df = df.groupby('item_count').agg(
    orders_with_tips=('with_tips', 'sum'),
    orders_without_tips=('without_tips', 'sum'),
    total_orders=('order_id', 'count')
).reset_index()

agg_df['tips_ratio'] = agg_df['orders_with_tips'] / agg_df['total_orders']

threshold_item_count = agg_df[agg_df['total_orders'] < 500]['item_count']

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8))
plt.subplots_adjust(left=0.08)

bar_width = 0.8

bar_with_tips = ax1.bar(agg_df['item_count'], agg_df['orders_with_tips'],
                       bar_width, label='Mit Trinkgeld', color='lightgreen')
bar_without_tips = ax1.bar(agg_df['item_count'], agg_df['orders_without_tips'],
                           bar_width, bottom=agg_df['orders_with_tips'],
                           label='Ohne Trinkgeld', color='salmon')

ax1.set_xlabel('Anzahl der Produkte pro Bestellung')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.set_title('Bestellungen mit und ohne Trinkgeld nach Anzahl der Produkte',
              loc='left', weight='bold')
ax1.legend()
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)

ax2.plot(agg_df['item_count'], agg_df['tips_ratio'],
         marker='o', color='blue', label='Trinkgeldquote')
ax2.set_xlabel('Anzahl der Produkte pro Bestellung')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_title('Verlauf der Trinkgeldquote nach Anzahl der Produkte',
              loc='left', weight='bold')
ax2.legend()
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)
```

```

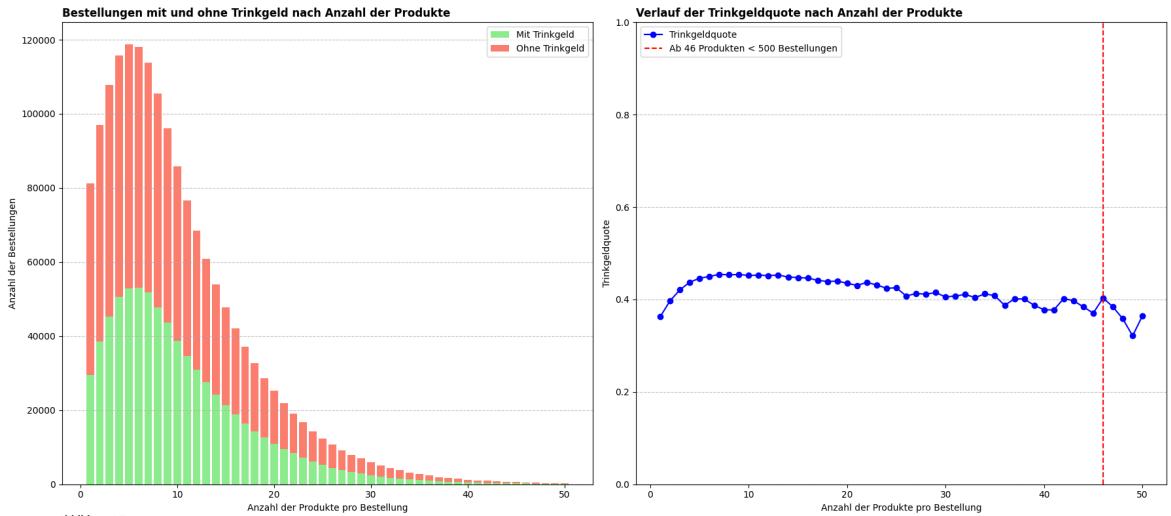
ax2.set_ylim(0, 1)

if not pd.isna(threshold_item_count):
    ax2.axvline(x=threshold_item_count, color='red', linestyle='--',
                label=f'Ab {threshold_item_count} Produkten < 500 Bestell'
    ax2.legend()

description = 'Die linke Grafik zeigt die Verteilung der Bestellungen mit
fig.text(0.05, 0, 'Abbildung 17:', weight='bold', ha='left')
fig.text(0.05, -0.045, description, wrap=True)

plt.tight_layout()
plt.show()

```



Erkenntnisse zur Bestellgröße und Trinkgeldverhalten



Beobachtete Muster



Entwicklung der Trinkgeldquote

Die Analyse zeigt ein charakteristisches Muster:

- Kleiner Anstieg der Trinkgeldquote bis ca. 8 Produkte pro Bestellung
- Anschließende Stabilisierung auf diesem erhöhten Niveau
- Keine weitere Steigerung bei noch größeren Bestellungen



Fazit:

Es scheint einen "Schwellenwert" von etwa 8 Produkten zu geben, bis zu dem die Bestellgröße einen positiven Einfluss auf die Trinkgeldbereitschaft hat. Darüber hinaus führen zusätzliche Produkte zu keiner weiteren Steigerung der Trinkgeldquote.



Detailanalyse: Bestellgröße nach Departments



Abteilungsspezifische Untersuchung des Bestellvolumens

Nach der allgemeinen Analyse des Zusammenhangs zwischen Bestellgröße und Trinkgeldverhalten vertiefen wir nun die Untersuchung auf Abteilungsebene. Dies ermöglicht uns, potenzielle departmentspezifische Unterschiede in diesem Zusammenhang zu identifizieren.



Analysestruktur:

- Separate Analyse für jede Produktabteilung
- Identische Visualisierung wie in der Gesamtanalyse:
 - Links: Absolute Verteilung der Bestellungen
 - Rechts: Trinkgeldquote nach Produktanzahl
- Markierung der statistischen Relevanzschwelle ($n < 500$)



Erkenntnisziel:

Identifikation von abteilungsspezifischen Mustern im Zusammenhang zwischen Bestellgröße und Trinkgeldverhalten, um mögliche Unterschiede zwischen den Produktkategorien aufzudecken.

In [70]:

```
# plot code
# Query: Alle Abteilungen
departments = session.query(Department.department_name).all()

for department in departments:
    department_name = department[0]

    # Query: Anzahl der Produkte pro Bestellung und ob Trinkgeld gegeben
    all_orders_item_count_with_tips = session.query(
        Order.order_id,
        func.count(Einkaufskorb.product_id).label('item_count'),
        Order.tips,
        func.count(Order.order_id).label('total_orders')
    ).join(Einkaufskorb, Einkaufskorb.order_id == Order.order_id) \
    .join(Product, Product.product_id == Einkaufskorb.product_id) \
    .join(Department, Department.department_id == Product.department_id) \
    .filter(Department.department_name == department_name) \
    .group_by(Order.order_id) \
    .all()

    results = all_orders_item_count_with_tips

    df = pd.DataFrame(results, columns=['order_id', 'item_count', 'tips',
                                         'with_tips'])

    df['with_tips'] = df['tips'].apply(lambda x: 1 if x else 0)
    df['without_tips'] = df['tips'].apply(lambda x: 0 if x else 1)
```

```

df = df[df['item_count'] <= 50]

agg_df = df.groupby('item_count').agg(
    orders_with_tips=('with_tips', 'sum'),
    orders_without_tips=('without_tips', 'sum'),
    total_orders=('order_id', 'count')
).reset_index()

agg_df['tips_ratio'] = agg_df['orders_with_tips'] / agg_df['total_orders']

threshold_item_count = agg_df[agg_df['total_orders'] < 500]['item_count'].max()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
plt.subplots_adjust(left=0.08)

bar_width = 0.8

bar_with_tips = ax1.bar(agg_df['item_count'], agg_df['orders_with_tips'],
                       bar_width, label='Mit Trinkgeld', color='lightblue')
bar_without_tips = ax1.bar(agg_df['item_count'], agg_df['orders_without_tips'],
                           bar_width, bottom=agg_df['orders_with_tips'],
                           label='Ohne Trinkgeld', color='salmon')

ax1.set_xlabel('Anzahl der Produkte pro Bestellung')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.set_title(f'Bestellungen mit und ohne Trinkgeld\nnach Anzahl der Produkte')
ax1.legend()
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)

ax2.plot(agg_df['item_count'], agg_df['tips_ratio'],
         marker='o', color='b', label='Trinkgeldquote')

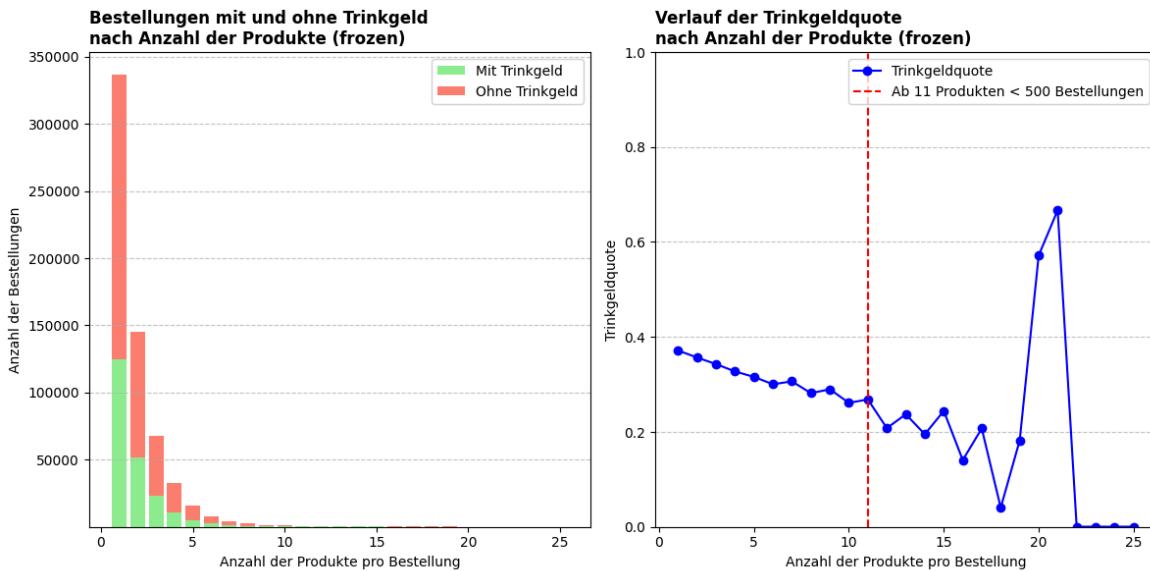
if not pd.isna(threshold_item_count):
    ax2.axvline(x=threshold_item_count, color='red', linestyle='--',
                 label=f'Ab {threshold_item_count} Produkten < 500 Bestellungen')

ax2.set_xlabel('Anzahl der Produkte pro Bestellung')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_title(f'Verlauf der Trinkgeldquote\nnach Anzahl der Produkte')
ax2.legend()
ax2.set_ylim(0, 1)
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)

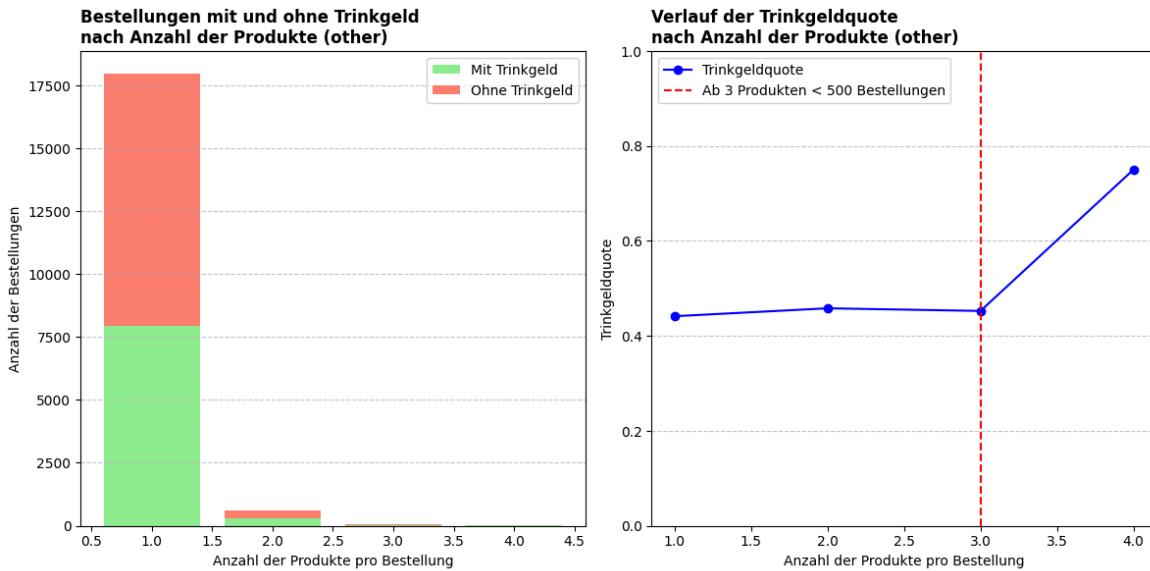
# Bildunterschrift
description = f'Die linke Grafik zeigt die Verteilung der Bestellungen nach Anzahl der Produkte. Die rechte Grafik zeigt den Verlauf der Trinkgeldquote nach Anzahl der Produkte. Eine horizontale rote Linie markiert die Bestellgrenze von 500 Produkten pro Bestellung, ab der die Trinkgeldquote steigt.'
fig.text(0.06, 0, f'Abbildung {18+departments.index(department)}:', wrap=True)
fig.text(0.06, -0.075, description, wrap=True)

plt.tight_layout()
plt.show()

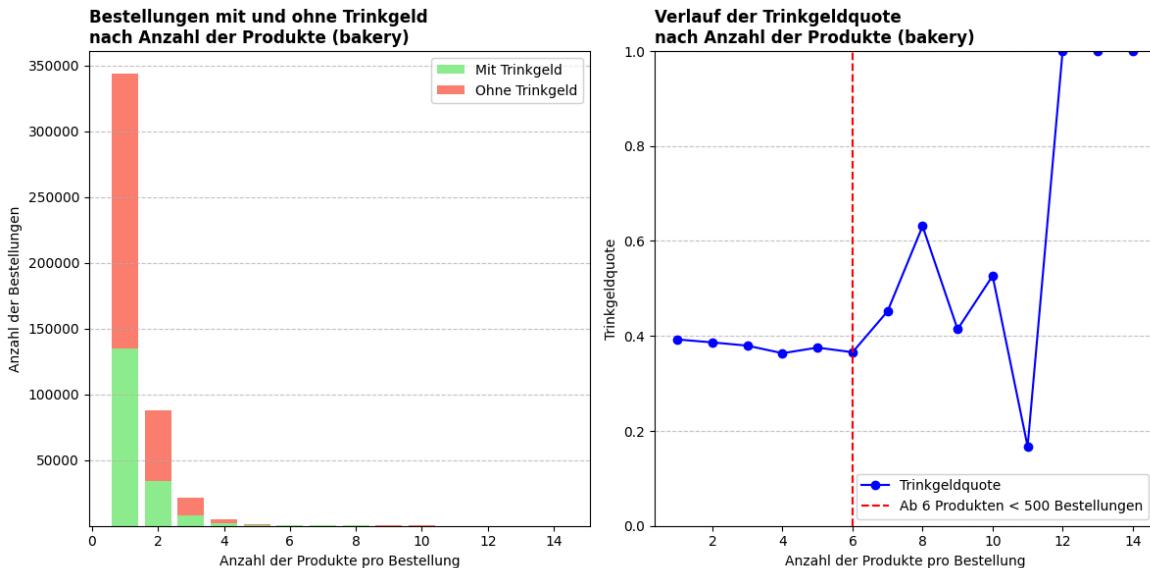
```

**Abbildung 18:**

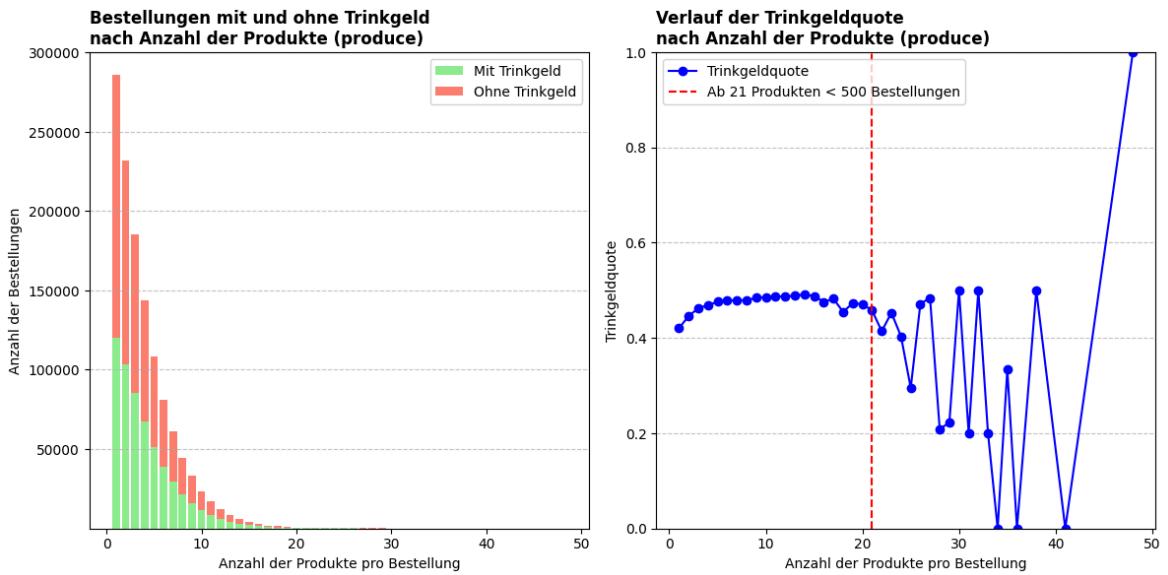
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung frozen in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 19:**

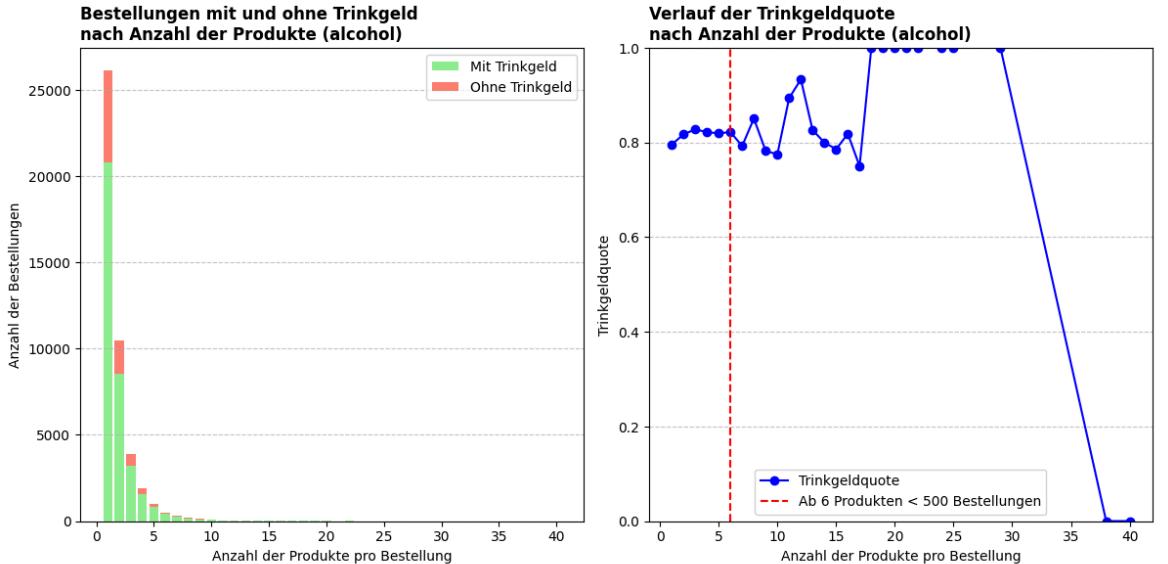
Die linke Balkengrafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung other in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Liniegrafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 20:**

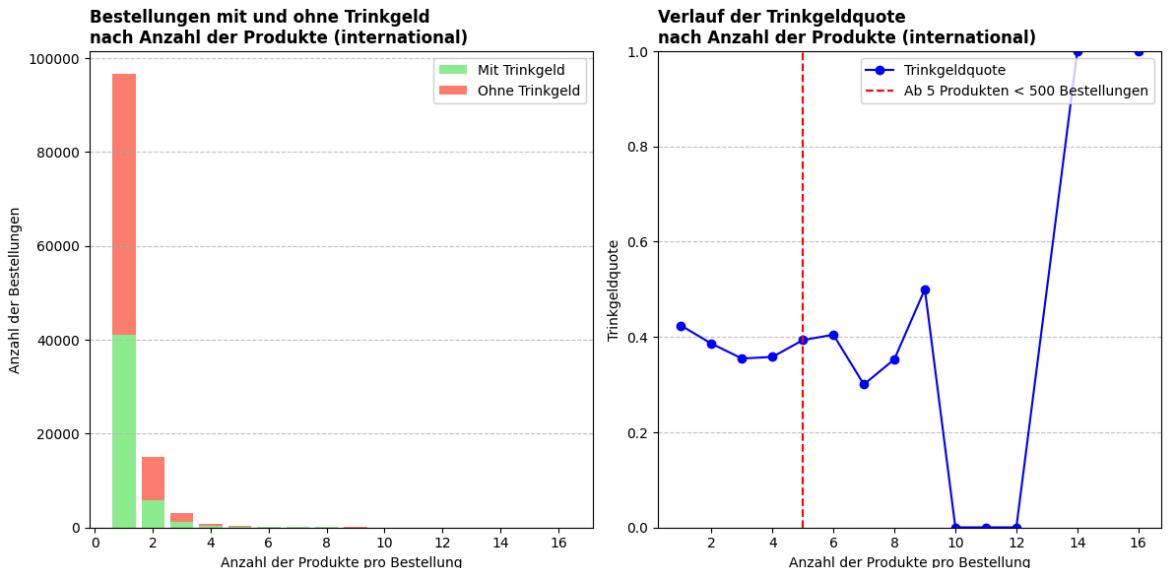
Die linke Balkengrafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung bakery in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Liniegrafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 21:**

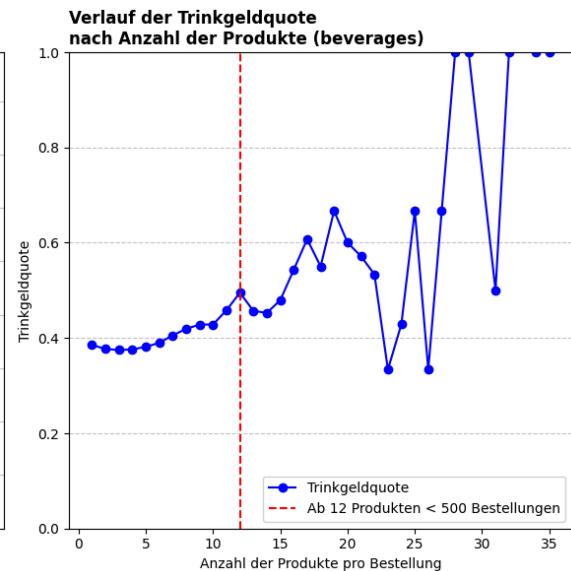
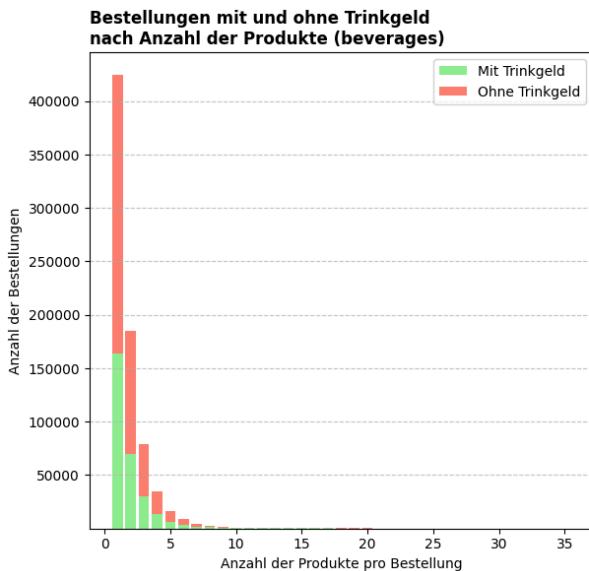
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung produce in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 22:**

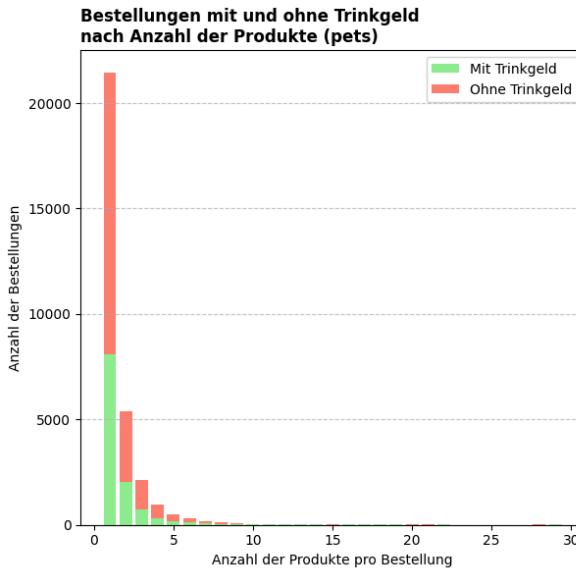
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung alcohol in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 23:**

Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung international in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 24:**

Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung beverages in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Verlauf der Trinkgeldquote nach Anzahl der Produkte (pets)**

Anzahl der Produkte pro Bestellung	Trinkgeldquote
1	~0.38
2	~0.35
3	~0.32
4	~0.35
5	~0.38
6	~0.42
7	~0.42
8	~0.45
9	~0.35
10	~0.52
11	~0.42
12	~0.42
13	~0.48
14	~0.55
15	~0.25
16	~0.7
17	~0.72
18	~0.68
19	~0.65
20	~0.68
21	~0.35
22	~0.5
23	~0.25
24	~0.98
25	~0.98
26	~0.98
27	~0.98
28	~0.98
29	~0.98
30	~0.98

Abbildung 25:

Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung pets in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

Bestellungen mit und ohne Trinkgeld nach Anzahl der Produkte (dry goods pasta)

Anzahl der Produkte pro Bestellung	Mit Trinkgeld	Ohne Trinkgeld	Total
0	~90,000	~120,000	~210,000
1	~30,000	~40,000	~70,000
2	~10,000	~15,000	~25,000
3	~5,000	~7,000	~12,000
4	~2,000	~3,000	~5,000
5	~1,000	~1,500	~2,500
6	~500	~700	~1,200
7	~200	~300	~500
8	~100	~150	~250
9	~50	~75	~125
10	~25	~35	~60
11	~15	~22	~37
12	~8	~12	~20
13	~4	~6	~10
14	~2	~3	~5
15	~1	~2	~3
16	~0.5	~0.8	~1.3
17	~0.2	~0.3	~0.5
18	~0.1	~0.15	~0.25
19	~0.05	~0.08	~0.13
20	~0.02	~0.03	~0.05
21	~0.01	~0.02	~0.03
22	~0.005	~0.008	~0.013
23	~0.002	~0.003	~0.005
24	~0.001	~0.002	~0.003
25	~0.0005	~0.0008	~0.0013
26	~0.0002	~0.0003	~0.0005
27	~0.0001	~0.0002	~0.0003
28	~0.00005	~0.00008	~0.00013
29	~0.00002	~0.00003	~0.00005
30	~0.00001	~0.00002	~0.00003

Verlauf der Trinkgeldquote nach Anzahl der Produkte (dry goods pasta)

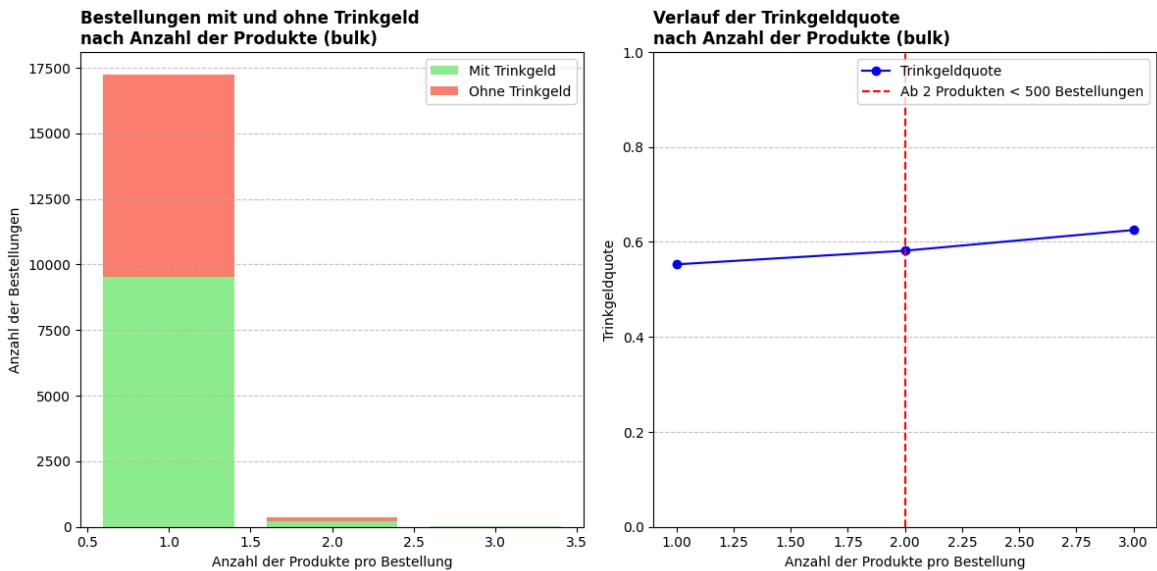
Anzahl der Produkte pro Bestellung	Trinkgeldquote
1	~0.42
2	~0.4
3	~0.38
4	~0.38
5	~0.38
6	~0.32
7	~0.32
8	~0.38
9	~0.18
10	~0.18
11	~0.12
12	~0.32
13	~0.45
14	~0.05
15	~0.05
16	~0.05
17	~0.05
18	~0.05
19	~0.05
20	~0.05
21	~0.05
22	~0.05
23	~0.05
24	~0.05
25	~0.05
26	~0.05
27	~0.05
28	~0.05
29	~0.05
30	~0.05

Abbildung 26:

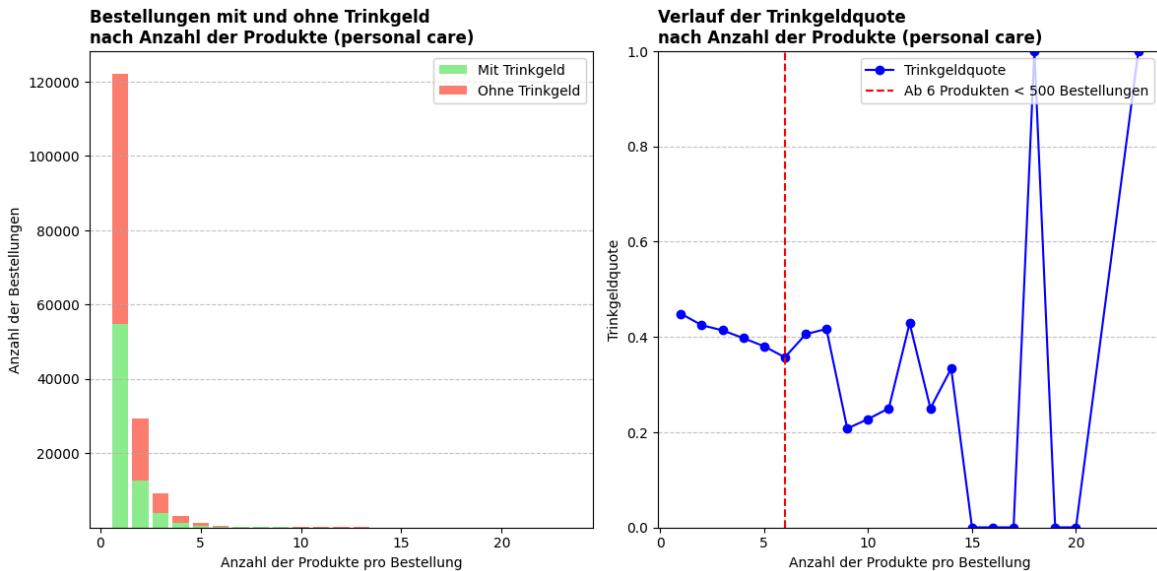
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung dry goods pasta in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

file:///home/jan-david/Documents/dabi1-neu/src/notebook/EDA.html

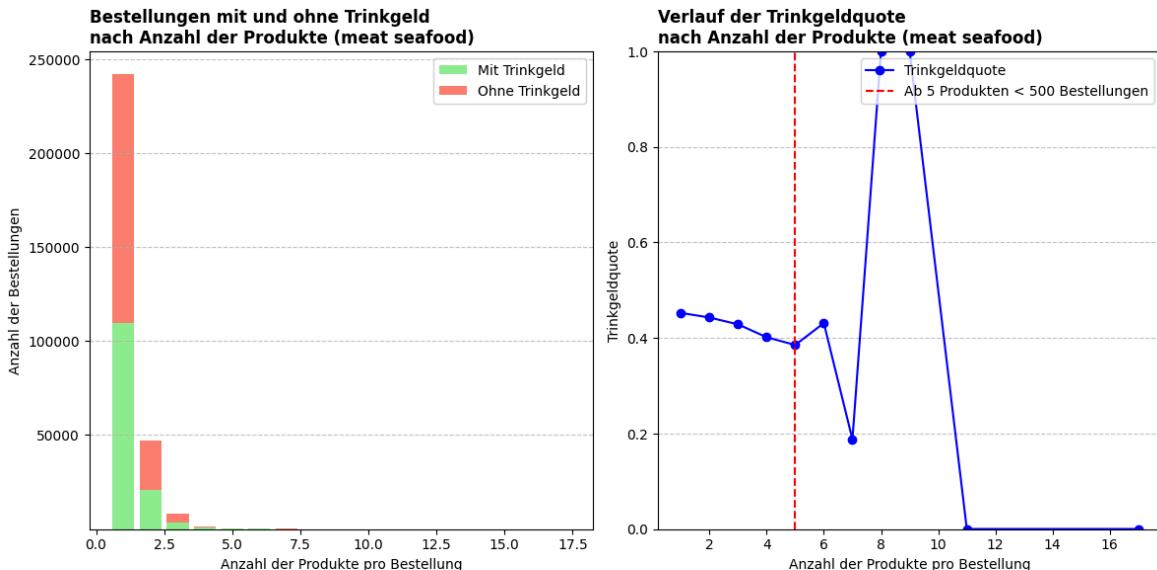
65/162

**Abbildung 27:**

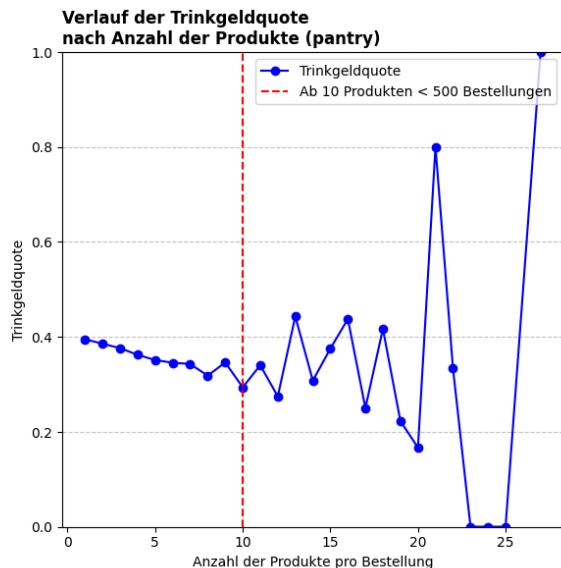
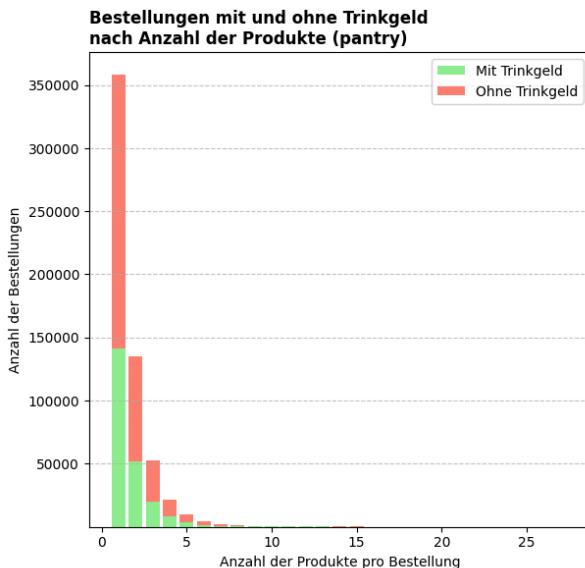
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung bulk in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 28:**

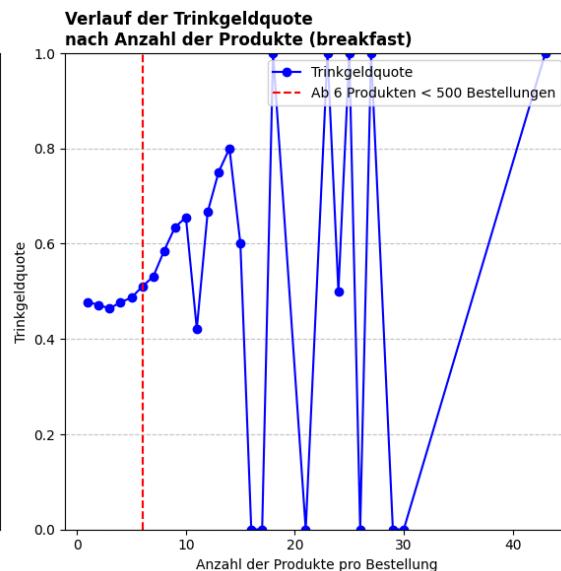
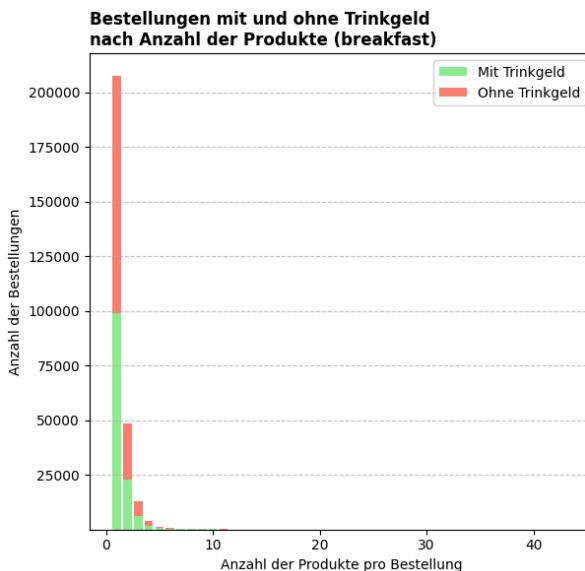
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung personal care in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 29:**

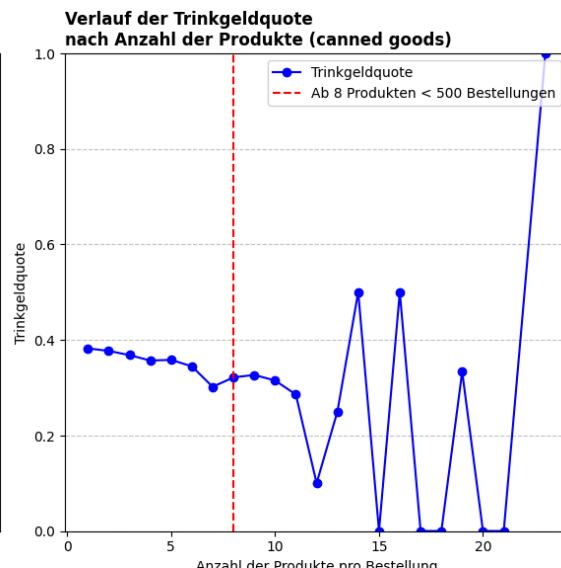
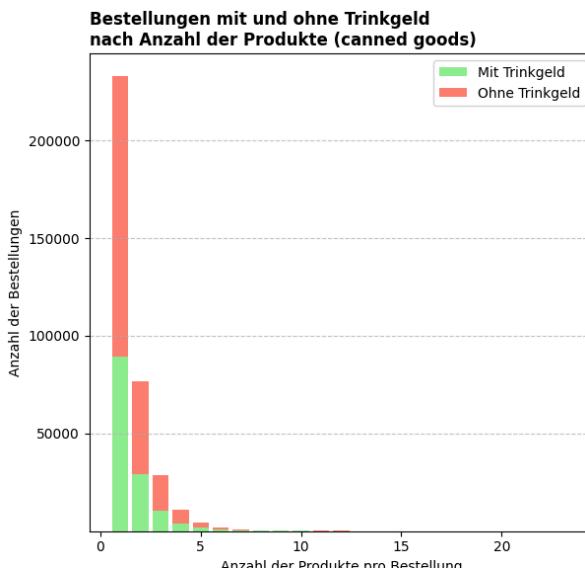
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung meat seafood in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 30:**

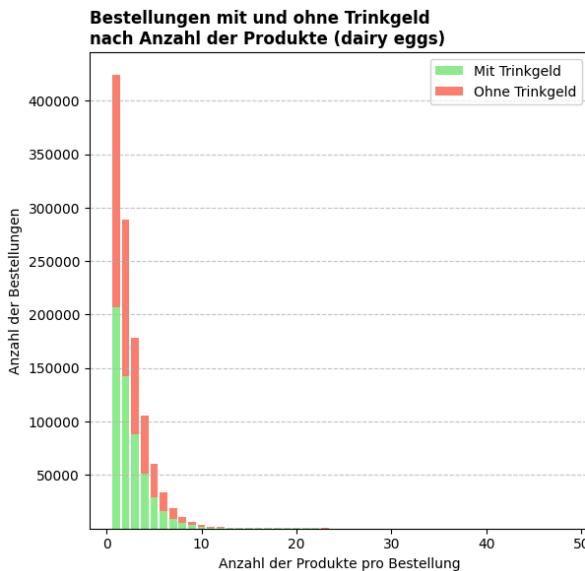
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung pantry in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 31:**

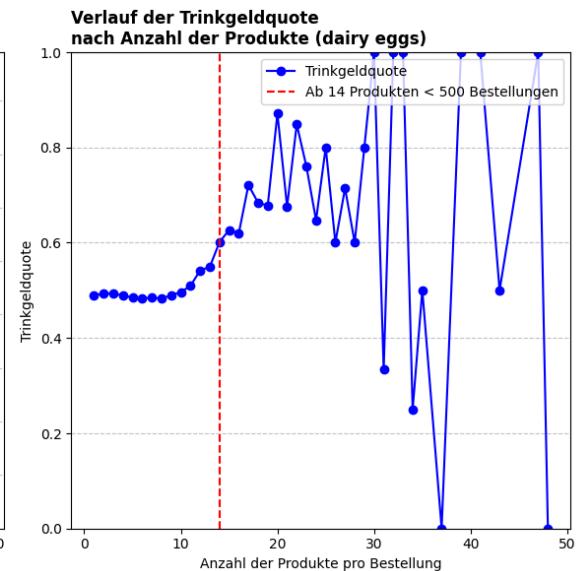
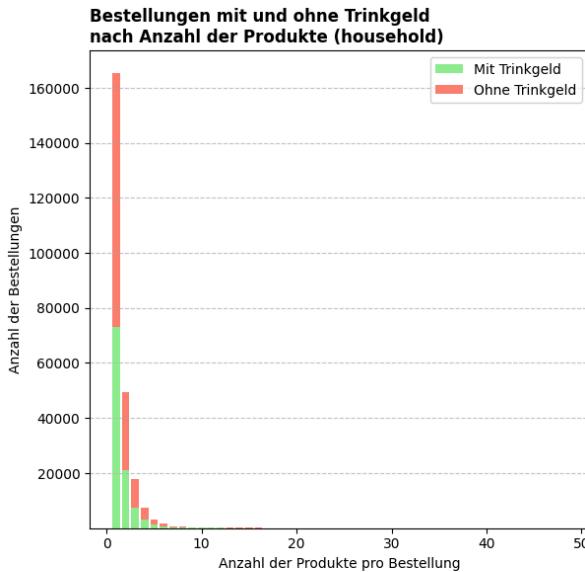
Die linke Balkengrafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung breakfast in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Liniegrafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 32:**

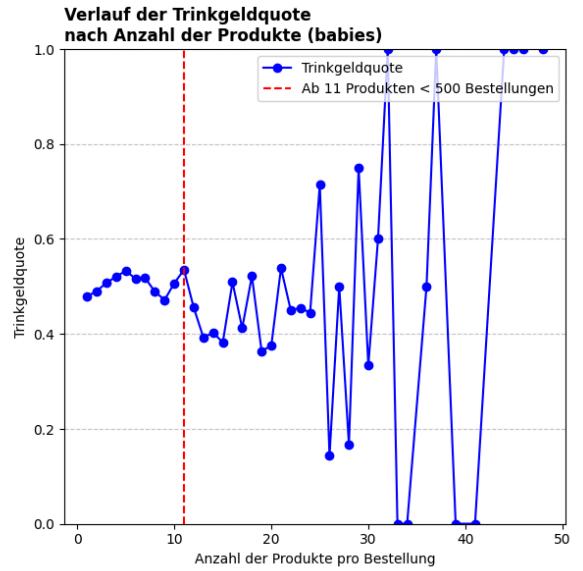
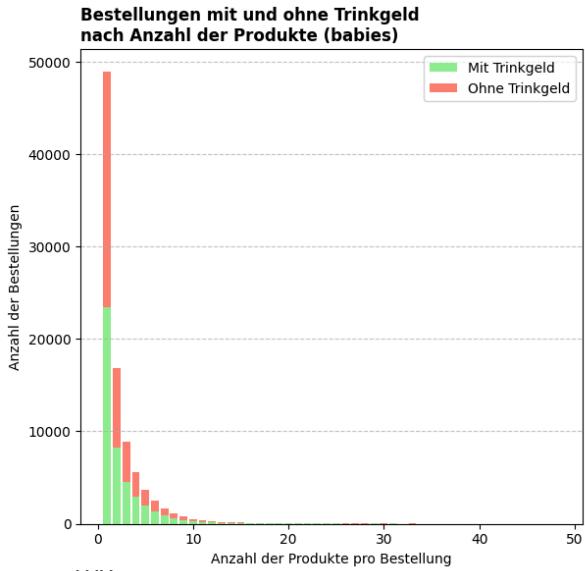
Die linke Balkengrafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung canned goods in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Liniegrafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 33:**

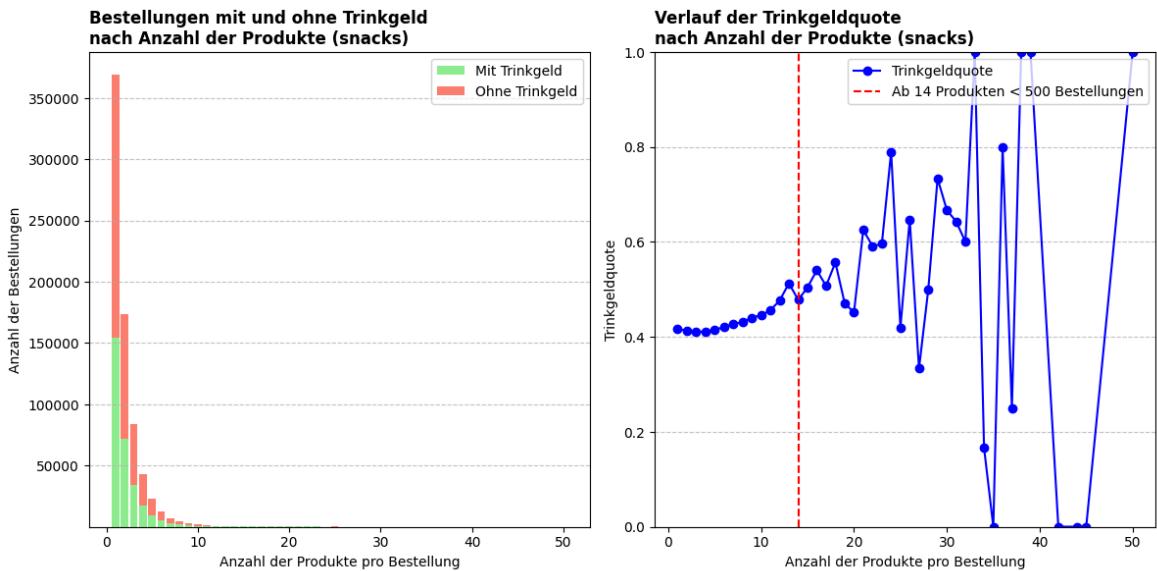
Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung dairy eggs in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 34:**

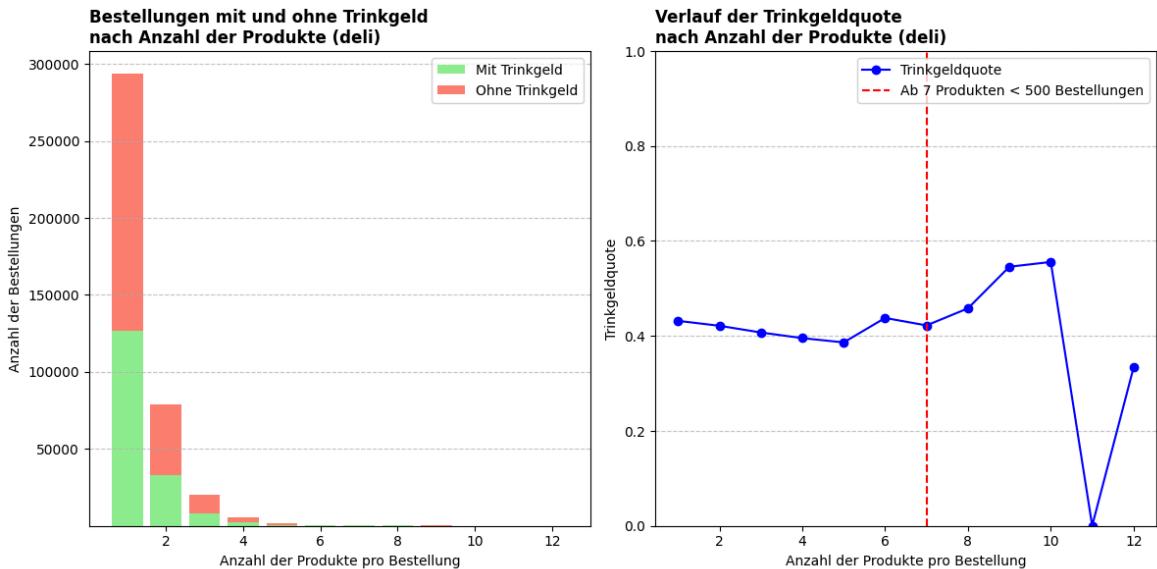
Die linke Balkengrafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung household in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Liniengrafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 35:**

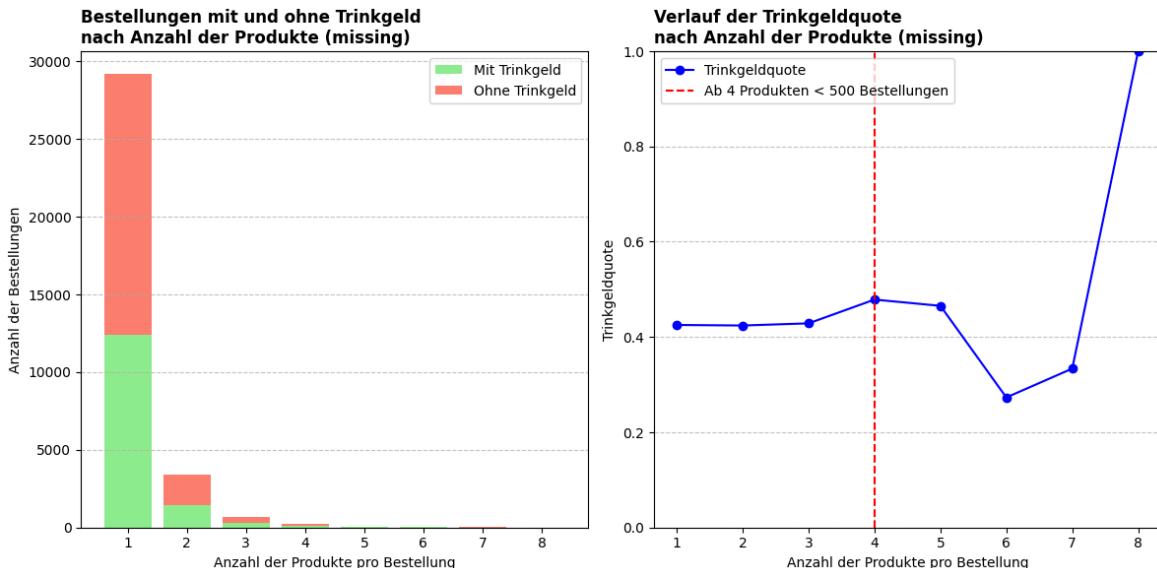
Die linke Balkengrafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung babies in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Liniengrafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 36:**

Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung snacks in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 37:**

Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung deli in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

**Abbildung 38:**

Die linke Grafik zeigt die Verteilung der Bestellungen mit und ohne Trinkgeld in der Abteilung missing in Abhängigkeit von der Anzahl der Produkte pro Bestellung. Die rechte Grafik visualisiert die Entwicklung der Trinkgeldquote mit zunehmender Produktanzahl. Die rote gestrichelte Linie markiert die Grenze, ab der weniger als 500 Bestellungen vorliegen.

```
In [71]: # Query: Anzahl der Bestellungen mit Bio-Produkten vs. Gesamtanzahl der B
orders_with_organic = (
```

```

    session.query(Order.order_id)
    .join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id)
    .join(Product, Einkaufskorb.product_id == Product.product_id)
    .filter(Product.product_name.ilike('%organic%'))
    .distinct()
    .group_by(Order.order_id)
    .count()
)

total_orders = session.query(Order).count()

```

Analyse des Bio-Produkt-Einflusses auf das Trinkgeldverhalten

Einfluss von Bio-Produkten auf Trinkgeld

Ein weiterer interessanter Aspekt unserer Analyse ist der potenzielle Zusammenhang zwischen Bio-Produkten und der Trinkgeldbereitschaft. Die Hypothese: Kunden, die Bio-Produkte bestellen, könnten ein anderes Trinkgeldverhalten aufweisen als Kunden mit konventionellen Produkten.

Untersuchungsaspekte:

- Verteilung von Bio- vs. konventionellen Bestellungen
- Vergleich der Trinkgeldquoten
- Mögliche Korrelation zwischen Bio-Präferenz und Trinkgeldverhalten

Hintergrund:

Bio-Produkte werden oft mit erhöhtem Qualitätsbewusstsein und Zahlungsbereitschaft in Verbindung gebracht. Diese Analyse soll zeigen, ob sich dieses Verhalten auch im Trinkgeldverhalten widerspiegelt.

```

In [72]: orders_without_organic = total_orders - orders_with_organic

labels = ['Mit "organic"', 'Ohne "organic"']
sizes = [orders_with_organic, orders_without_organic]
colors = ['#4CAF50', '#FFC107']
explode = (0.1, 0)

fig, ax = plt.subplots(figsize=(6, 6))
plt.subplots_adjust(left=0.08)

# Berechne die Prozentanteile
percentages = [s/total_orders*100 for s in sizes]
# Erstelle Labels mit absoluten Zahlen und Prozenten
labels_with_values = [f'{labels[i]}\n{sizes[i]} | {percentages[i]:.1f}' for i in range(len(labels))]

wedges, texts = ax.pie(sizes, labels=None, colors=colors, explode=explode)

```

```

ax.legend(wedges, labels_with_values,
          title="Bestellungen",
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1))

ax.set_title('Verteilung der Bestellungen mit und ohne "organic"-Produkte',
             loc='left', weight='bold')

description = 'Das Kreisdiagramm zeigt die prozentuale Verteilung der Bes...'
fig.text(0.05, 0.1, 'Abbildung 38:', weight='bold', ha='left')
fig.text(0.05, 0.02, description, wrap=True)

plt.tight_layout()
plt.show()

```

Verteilung der Bestellungen mit und ohne "organic"-Produkte

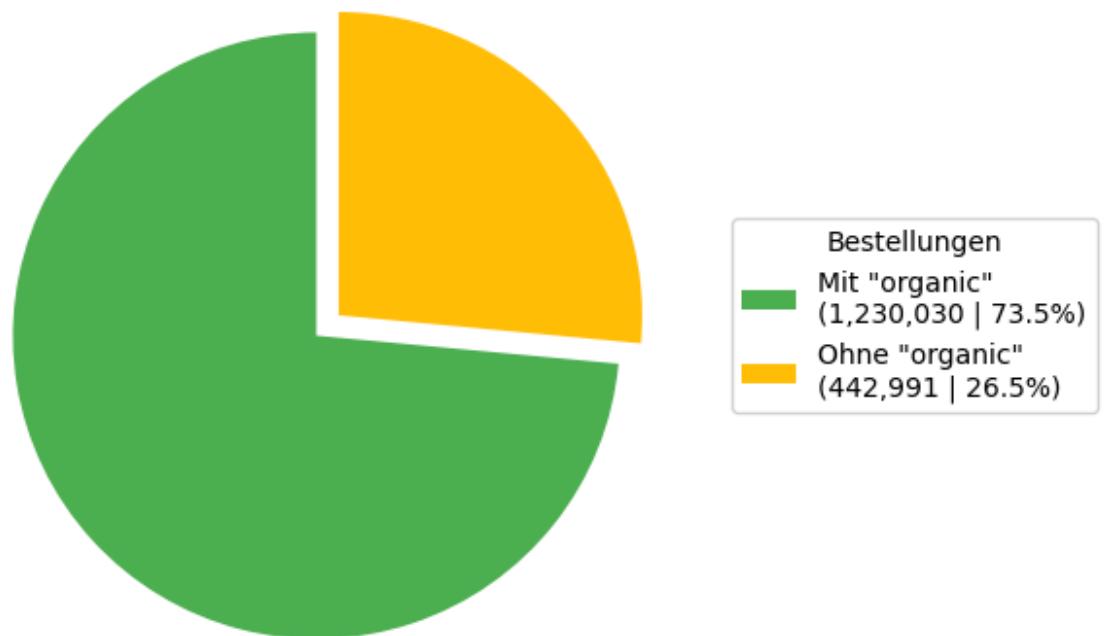


Abbildung 38:

Das Kreisdiagramm zeigt die prozentuale Verteilung der Bestellungen, die Bio-Produkte ("organic") enthalten, im Vergleich zu Bestellungen ohne Bio-Produkte.



Verteilung von Bio-Produkten in Bestellungen



Hoher Anteil von Bio-Produkten

Zentrale Erkenntnis

Die Analyse zeigt eine deutliche Präferenz für Bio-Produkte im Bestellverhalten:

- **73,5%** aller Bestellungen enthalten mindestens ein Bio-Produkt

- Nur **26,5%** der Bestellungen bestehen ausschließlich aus konventionellen Produkten

Bedeutung:

Diese deutliche Mehrheit von Bestellungen mit Bio-Produkten bildet eine solide Basis für die weitere Analyse des Zusammenhangs zwischen Bio-Präferenz und Trinkgeldverhalten.

```
In [73]: # Vergleichsanalyse: Trinkgeldverhalten bei Bestellungen mit und ohne Bio
organic_with_tips = (
    session.query(Order)
    .join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id)
    .join(Product, Einkaufskorb.product_id == Product.product_id)
    .filter(Product.product_name.ilike('%organic%'))
    .filter(Order.tips == True)
    .distinct()
    .count()
)

organic_without_tips = (
    session.query(Order)
    .join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id)
    .join(Product, Einkaufskorb.product_id == Product.product_id)
    .filter(Product.product_name.ilike('%organic%'))
    .filter(Order.tips == False)
    .distinct()
    .count()
)

total_organic_orders = organic_with_tips + organic_without_tips

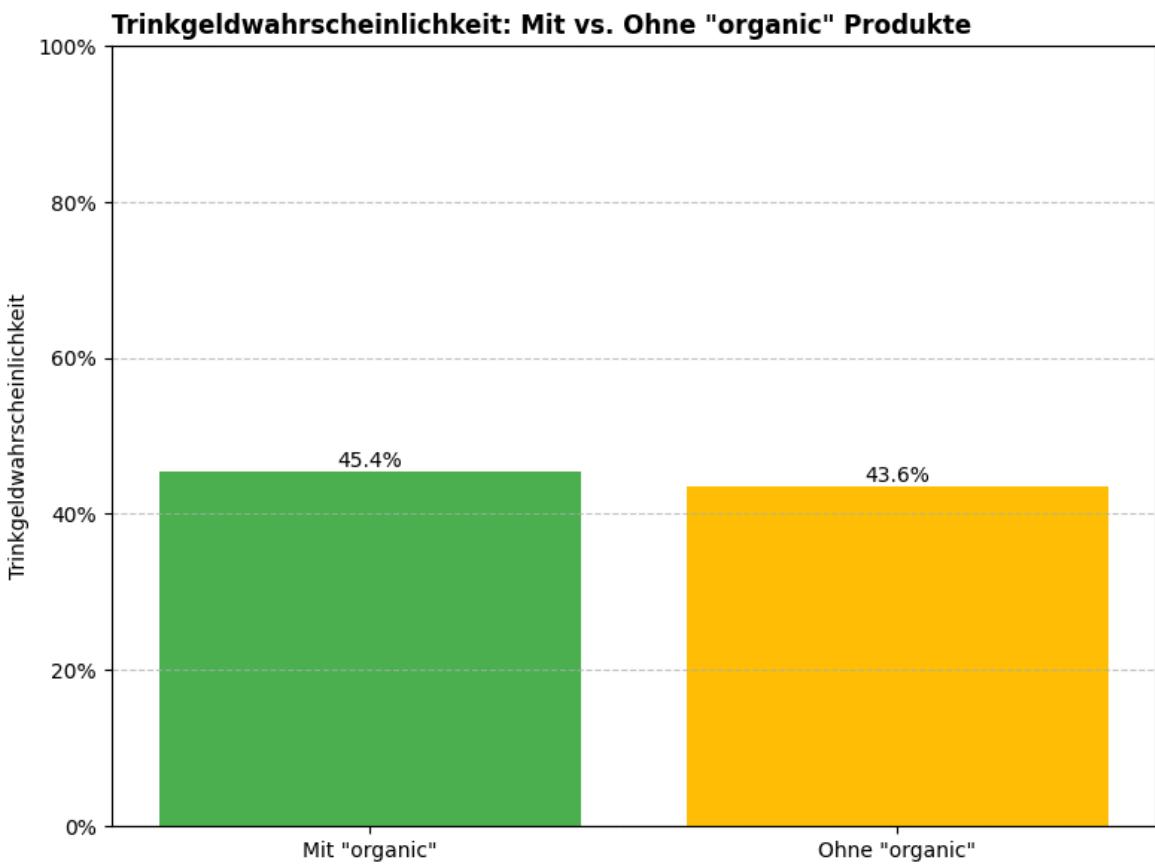
non_organic_with_tips = (
    session.query(Order)
    .outerjoin(Einkaufskorb, Order.order_id == Einkaufskorb.order_id)
    .outerjoin(Product, Einkaufskorb.product_id == Product.product_id)
    .filter(~Product.product_name.ilike('%organic%'))
    .filter(Order.tips == True)
    .distinct()
    .count()
)

non_organic_without_tips = (
    session.query(Order)
    .outerjoin(Einkaufskorb, Order.order_id == Einkaufskorb.order_id)
    .outerjoin(Product, Einkaufskorb.product_id == Product.product_id)
    .filter(~Product.product_name.ilike('%organic%'))
    .filter(Order.tips == False)
    .distinct()
    .count()
)

total_non_organic_orders = non_organic_with_tips + non_organic_without_tips
```

```
tip_probability_organic = organic_with_tips / total_organic_orders if tot  
tip_probability_non_organic = non_organic_with_tips / total_non_organic_o
```

```
In [74]: # plot code  
labels = ['Mit "organic"', 'Ohne "organic"']  
probabilities = [tip_probability_organic, tip_probability_non_organic]  
colors = ['#4CAF50', '#FFC107']  
  
fig, ax = plt.subplots(figsize=(8, 6))  
plt.subplots_adjust(left=0.08)  
  
bars = ax.bar(labels, probabilities, color=colors)  
  
# Füge Prozentangaben auf den Balken hinzu  
for bar in bars:  
    height = bar.get_height()  
    ax.text(bar.get_x() + bar.get_width()/2., height,  
            f'{height:.1%}',  
            ha='center', va='bottom')  
  
ax.set_ylabel('Trinkgeldwahrscheinlichkeit')  
ax.set_title('Trinkgeldwahrscheinlichkeit: Mit vs. Ohne "organic" Produkt  
loc='left', weight='bold')  
ax.set_ylim(0, 1)  
  
ax.yaxis.set_major_formatter(plt.FuncFormatter(lambda y, _: '{:.0%}'.format(y)))  
ax.grid(True, axis='y', linestyle='--', alpha=0.7)  
  
description = 'Die Grafik vergleicht die Wahrscheinlichkeit, dass Trinkgeldwahrscheinlichkeit  
fig.text(0.05, 0, 'Abbildung 39:', weight='bold', ha='left')  
fig.text(0.05, -0.05, description, wrap=True)  
  
plt.tight_layout()  
plt.show()
```

**Abbildung 39:**

Die Grafik vergleicht die Wahrscheinlichkeit, dass Trinkgeld gegeben wird, zwischen Bestellungen mit und ohne Bio-Produkte ("organic").

💡 Trinkgeldverhalten bei Bio- vs. konventionellen Bestellungen



Minimale Unterschiede im Trinkgeldverhalten

📊 Analyseergebnis

Entgegen möglicher Erwartungen zeigt die Analyse nur marginale Unterschiede in der Trinkgeldwahrscheinlichkeit:

- Die Unterschiede in der Trinkgeldquote sind minimal
- Bio-Präferenz scheint kein ausschlaggebender Faktor für das Trinkgeldverhalten zu sein
- Das Vorhandensein von Bio-Produkten in der Bestellung hat keinen signifikanten Einfluss auf die Trinkgeldbereitschaft

🎯 Fazit:

Trotz des hohen Anteils von Bio-Bestellungen (73,5%) lässt sich kein nennenswerter Zusammenhang zwischen der Wahl von Bio-Produkten und

der Bereitschaft, Trinkgeld zu geben, feststellen. Dies deutet darauf hin, dass das Trinkgeldverhalten von anderen Faktoren stärker beeinflusst wird.



Granulare Analyse des Bio-Anteils



Detailbetrachtung der Bio-Intensität

Da die binäre Unterscheidung (Bio ja/nein) keine signifikanten Unterschiede im Trinkgeldverhalten zeigt, verfeinern wir unsere Analyse. Wir untersuchen nun, ob der prozentuale Anteil von Bio-Produkten innerhalb einer Bestellung einen Einfluss auf das Trinkgeldverhalten hat.



Analyseansatz:

- Gruppierung der Bestellungen nach Bio-Anteil
- Berechnung der Trinkgeldquote pro Gruppe
- Identifikation möglicher Schwellenwerte oder gradueller Effekte



Erkenntnisziel:

Diese differenziertere Betrachtung soll aufzeigen, ob ein höherer Bio-Anteil in der Bestellung mit einer veränderten Trinkgeldbereitschaft einhergeht und ob es bestimmte Schwellenwerte gibt, ab denen sich das Trinkgeldverhalten signifikant ändert.

In [75]:

```
from sqlalchemy import func, case, cast, Float

# Abfrage für die Verteilung der Bio-Produkte
organic_distribution = (
    session.query(
        Order.order_id,
        (func.count(case((Product.product_name.ilike('%organic%'), 1))) *
         func.count(Product.product_id)).label('organic_percentage')
    )
    .join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id)
    .join(Product, Einkaufskorb.product_id == Product.product_id)
    .group_by(Order.order_id)
    .subquery()
)

# Gruppierung der Ergebnisse
groups = {
    '0%': 0,
    '1-25%': 0,
    '26-50%': 0,
    '51-75%': 0,
    '76-99%': 0,
```

```

        '100%': 0
    }

results = session.query(
    case(
        (organic_distribution.c.organic_percentage == 0, '0%'),
        (organic_distribution.c.organic_percentage <= 25, '1-25%'),
        (organic_distribution.c.organic_percentage <= 50, '26-50%'),
        (organic_distribution.c.organic_percentage <= 75, '51-75%'),
        (organic_distribution.c.organic_percentage < 100, '76-99%'),
        (organic_distribution.c.organic_percentage == 100, '100%')
    ).label('group'),
    func.count('*').label('count')
).group_by('group').all()

# Ergebnisse in Dictionary speichern
for group, count in results:
    if group in groups:
        groups[group] = count

# Gesamtanzahl der Bestellungen
total_orders = sum(groups.values())

# Ausgabe der Ergebnisse
print("Verteilung der Bio-Produkte in Bestellungen:")
for group, count in groups.items():
    percentage = (count / total_orders) * 100 if total_orders > 0 else 0
    print(f"{group}: {count} Bestellungen ({percentage:.2f}%)")

```

Verteilung der Bio-Produkte in Bestellungen:

- 0%: 442991 Bestellungen (26.48%)
- 1-25%: 375675 Bestellungen (22.45%)
- 26-50%: 539189 Bestellungen (32.23%)
- 51-75%: 237870 Bestellungen (14.22%)
- 76-99%: 36717 Bestellungen (2.19%)
- 100%: 40579 Bestellungen (2.43%)

```

In [76]: fig, ax = plt.subplots(figsize=(10, 6))
plt.subplots_adjust(left=0.08)

# Daten vorbereiten
group_names = list(groups.keys())
group_values = list(groups.values())

# Farbpalette erstellen (Farbverlauf von hellgrün zu dunkelgrün)
colors = plt.cm.Greens(np.linspace(0.3, 0.8, len(group_names)))

# Balkendiagramm erstellen
bars = ax.bar(group_names, group_values, color=colors)

# Achsentitel und Formatierung
ax.set_ylabel('Anzahl Bestellungen')
ax.set_xlabel('Anteil Bio-Produkte')
ax.set_title('Verteilung der Bio-Produkte in Bestellungen',
            loc='left', weight='bold')
ax.set_xticks(range(len(group_names)))
ax.set_xticklabels(group_names, rotation=45)

```

```

# Grid hinzufügen
ax.grid(True, axis='y', linestyle='--', alpha=0.7)

# Y-Achse formatieren
max_value = max(group_values)
ax.set_ylim(0, max_value * 1.1) # 10% Platz für Labels

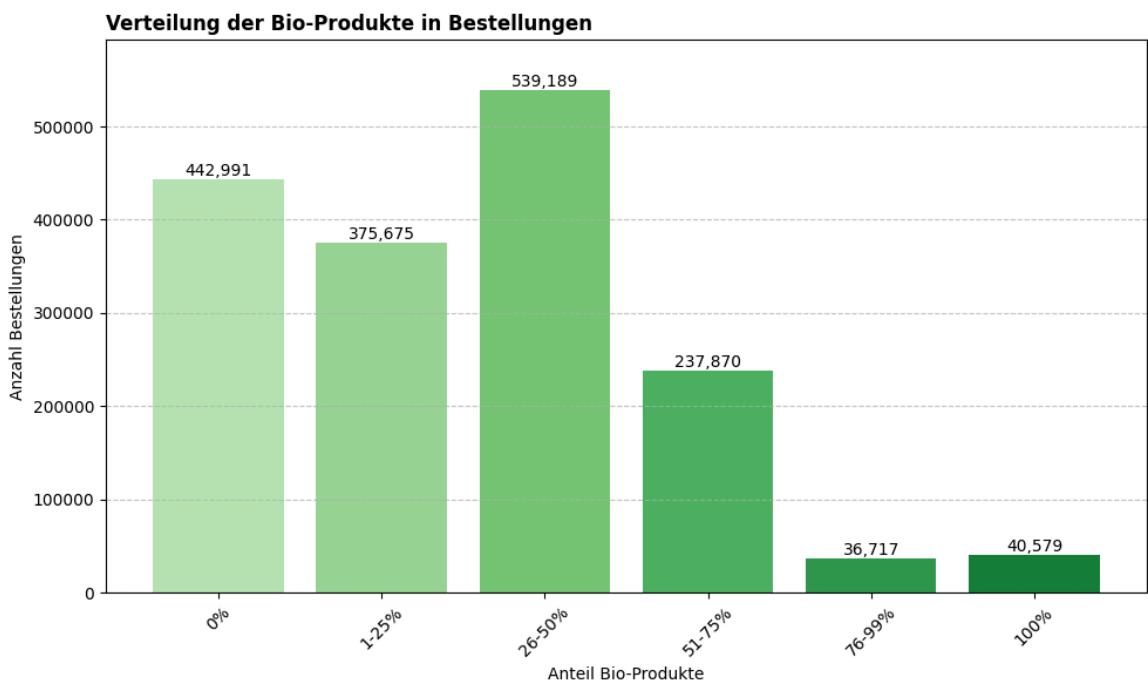
# Werte über den Balken anzeigen
for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height,
            f"int(height):,{})", ha='center', va='bottom', fontsize=10)

# Bildunterschrift
description = ('Die Grafik zeigt die Verteilung der Bestellungen nach dem  
' 'Die Zahlen über den Balken geben die absolute Anzahl der B  
' 'Die Gruppen sind nach dem prozentualen Anteil der Bio-Prod

fig.text(0.09, 0, 'Abbildung:', weight='bold', ha='left')
fig.text(0.09, -0.095, description, wrap=True)

plt.tight_layout()
plt.show()

```

**Abbildung:**

Die Grafik zeigt die Verteilung der Bestellungen nach dem Anteil der Bio-Produkte. Die Zahlen über den Balken geben die absolute Anzahl der Bestellungen in jeder Gruppe an. Die Gruppen sind nach dem prozentualen Anteil der Bio-Produkte in der Bestellung eingeteilt.



Interpretation der Bio-Produkt-Verteilung



Zentrale Erkenntnisse

Die Analyse der Verteilung von Bio-Produkten in den Bestellungen zeigt ein deutliches Muster:



Hauptbeobachtungen:

- Der Großteil der Bestellungen enthält einen geringen Anteil an Bio-Produkten
- Mit steigendem Bio-Anteil nimmt die Anzahl der Bestellungen ab
- Bestellungen mit einem sehr hohen Bio-Anteil (>75%) sind verhältnismäßig selten

Mögliche Gründe:

- Höhere Preise von Bio-Produkten
- Begrenzte Verfügbarkeit von Bio-Alternativen

Analyse: Zusammenhang zwischen Bio-Produkten und Trinkgeldverhalten

Untersuchungsansatz

Um einen möglichen Zusammenhang zwischen dem Kauf von Bio-Produkten und der Trinkgeldbereitschaft zu untersuchen, analysieren wir die Trinkgeldwahrscheinlichkeit in Abhängigkeit vom Bio-Anteil in den Bestellungen.

Methodische Details:

- Gruppierung der Bestellungen nach Bio-Anteil (0% bis 100%)
- Berechnung der Trinkgeldwahrscheinlichkeit pro Gruppe
- Berücksichtigung aller Bestellungen im Datensatz

Untersuchungsziel:

Identifikation möglicher Korrelationen zwischen dem Anteil von Bio-Produkten in einer Bestellung und der Wahrscheinlichkeit einer Trinkgeldzahlung. Dies könnte wichtige Erkenntnisse für Kundenverhaltensmuster liefern.

```
In [77]: # Hauptanalyse
organic_percentage_subquery = (
    session.query(
        Order.order_id,
        (cast(func.count(case((Product.product_name.ilike('%organic%'), 1
            cast(func.count(Product.product_id), Float)).label('organic_perc
        Order.tips
    )
    .join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id)
    .join(Product, Einkaufskorb.product_id == Product.product_id)
    .group_by(Order.order_id, Order.tips)
    .subquery()
)
```

```

# Berechne Trinkgeldwahrscheinlichkeit für jede Gruppe
tips_probability = session.query(
    case(
        (organic_percentage_subquery.c.organic_percentage == 0, '0%'),
        (and_(
            organic_percentage_subquery.c.organic_percentage > 0,
            organic_percentage_subquery.c.organic_percentage <= 25
        ), '1-25%'),
        (and_(
            organic_percentage_subquery.c.organic_percentage > 25,
            organic_percentage_subquery.c.organic_percentage <= 50
        ), '26-50%'),
        (and_(
            organic_percentage_subquery.c.organic_percentage > 50,
            organic_percentage_subquery.c.organic_percentage <= 75
        ), '51-75%'),
        (and_(
            organic_percentage_subquery.c.organic_percentage > 75,
            organic_percentage_subquery.c.organic_percentage < 100
        ), '76-99%'),
        (organic_percentage_subquery.c.organic_percentage >= 99.99, '100%')
    ).label('group'),
    func.count('*').label('total_orders'),
    func.sum(case((organic_percentage_subquery.c.tips == True, 1), else_=0))
).group_by('group').all()

# Sortierung der Ergebnisse
order_dict = {'0%': 0, '1-25%': 1, '26-50%': 2, '51-75%': 3, '76-99%': 4, '100%': 5}
tips_probability = sorted(tips_probability, key=lambda x: order_dict[x[0]])

# Ausgabe der Ergebnisse
print("\nTrinkgeldwahrscheinlichkeit nach Bio-Anteil:")
for group, total, tips in tips_probability:
    probability = (tips / total) if total > 0 else 0
    print(f"Gruppe {group}:")
    print(f"  Anzahl Bestellungen: {total}")
    print(f"  Bestellungen mit Trinkgeld: {tips}")
    print(f"  Trinkgeldwahrscheinlichkeit: {probability:.2%}")
    print()

groups = [result[0] for result in tips_probability]
probabilities = [(result[2]/result[1]) if result[1] > 0 else 0 for result in tips_probability]

# Plot erstellen
fig, ax = plt.subplots(figsize=(10, 6))
plt.subplots_adjust(left=0.08)

# Farbpalette erstellen (Farbverlauf von hellgrün zu dunkelgrün)
colors = plt.cm.Greens(np.linspace(0.3, 0.8, len(groups)))

# Balkendiagramm erstellen
bars = ax.bar(groups, probabilities, color=colors)

# Achsentitel und Formatierung
ax.set_ylabel('Trinkgeldwahrscheinlichkeit (%)')
ax.set_xlabel('Anteil Bio-Produkte')

```

```

ax.set_title('Trinkgeldwahrscheinlichkeit nach Bio-Anteil in der Bestellung',
             loc='left', weight='bold')
ax.set_ylim(0, 1)
ax.set_xticks(range(len(groups)))
ax.set_xticklabels(groups, rotation=45)
ax.set_yticks([i / 10 for i in range(0, 11)])
ax.set_yticklabels([f'{i * 10}%' for i in range(0, 11)])
ax.grid(True, axis='y', linestyle='--', alpha=0.7)

# Werte über den Balken anzeigen
for bar in bars:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width()/2, height + 0.02,
            f'{height:.1%}', ha='center', va='bottom', fontsize=10)

# Bildunterschrift
description = ('Die Grafik zeigt die Wahrscheinlichkeit einer Trinkgeldzahlung\n'
               'Die Prozentangaben über den Balken zeigen die genauen Wahr-'
               'scheinlichkeiten für jede Gruppe.\n'
               'Die Gruppen sind nach dem prozentualen Anteil der Bio-Produkte\n'
               'geordnet.')
fig.text(0.09, 0, 'Abbildung 48:', weight='bold', ha='left')
fig.text(0.09, -0.095, description, wrap=True)

plt.tight_layout()
plt.show()

```

Trinkgeldwahrscheinlichkeit nach Bio-Anteil:

Gruppe 0%:

Anzahl Bestellungen: 442991
 Bestellungen mit Trinkgeld: 171310
 Trinkgeldwahrscheinlichkeit: 38.67%

Gruppe 1-25%:

Anzahl Bestellungen: 375675
 Bestellungen mit Trinkgeld: 160968
 Trinkgeldwahrscheinlichkeit: 42.85%

Gruppe 26-50%:

Anzahl Bestellungen: 539189
 Bestellungen mit Trinkgeld: 244978
 Trinkgeldwahrscheinlichkeit: 45.43%

Gruppe 51-75%:

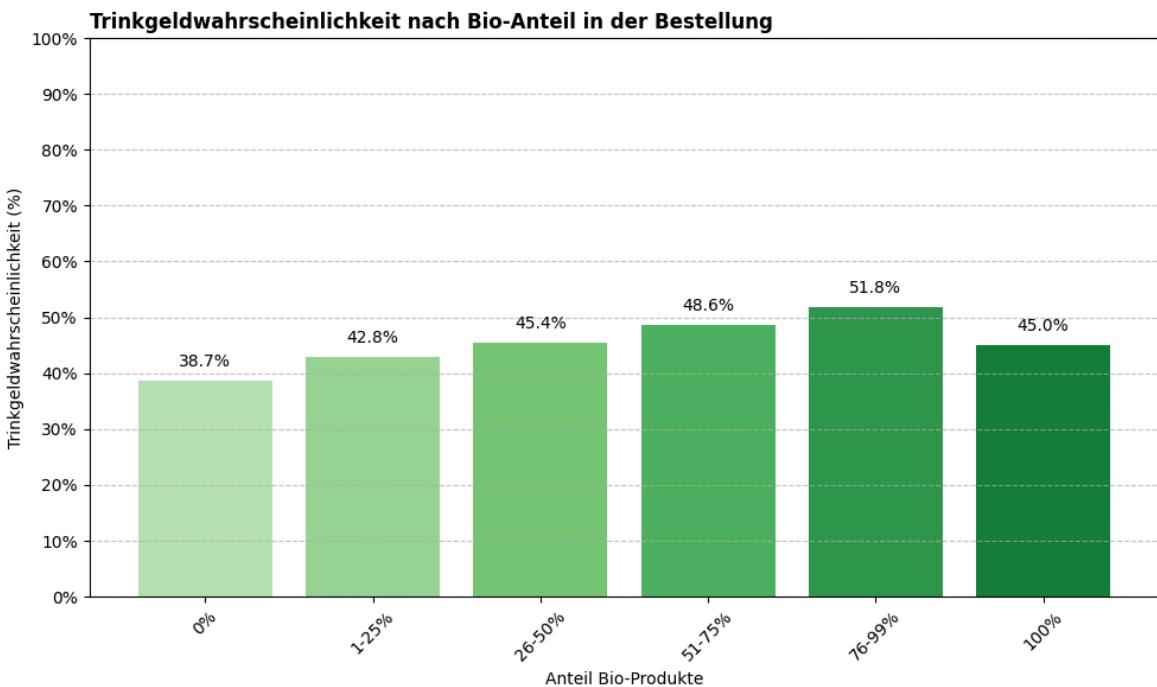
Anzahl Bestellungen: 237870
 Bestellungen mit Trinkgeld: 115561
 Trinkgeldwahrscheinlichkeit: 48.58%

Gruppe 76-99%:

Anzahl Bestellungen: 36717
 Bestellungen mit Trinkgeld: 19024
 Trinkgeldwahrscheinlichkeit: 51.81%

Gruppe 100%:

Anzahl Bestellungen: 40579
 Bestellungen mit Trinkgeld: 18272
 Trinkgeldwahrscheinlichkeit: 45.03%

**Abbildung 48:**

Die Grafik zeigt die Wahrscheinlichkeit einer Trinkgeldzahlung in Abhängigkeit vom Anteil der Bio-Produkte in der Bestellung. Die Prozentangaben über den Balken zeigen die genauen Wahrscheinlichkeiten für jede Gruppe. Die Gruppen sind nach dem prozentualen Anteil der Bio-Produkte in der Bestellung eingeteilt.



Ergebnisse der Bio-Produkt-Trinkgeld-Analyse

🎯 Zentrale Erkenntnisse

Die Analyse zeigt einen positiven Zusammenhang zwischen dem Anteil an Bio-Produkten in einer Bestellung und der Wahrscheinlichkeit einer Trinkgeldzahlung. Außer bei 100% Bio bestellungen, da geht die Trinkgeldwahrscheinlichkeit wieder zurück

📊 Beobachteter Trend:

- Je höher der Bio-Anteil, desto höher die Trinkgeldwahrscheinlichkeit
- Bestellungen mit 76-99% Bio-Produkten zeigen die höchste Trinkgeldquote
- Kontinuierlicher Anstieg über alle Prozentgruppen hinweg

💡 Mögliche Interpretationen:

- Bio-Käufer könnten ein stärkeres Bewusstsein für faire Entlohnung haben
- Höhere Zahlungsbereitschaft bei nachhaltigkeitsorientierten Kunden
- Möglicher Zusammenhang zwischen Einkommensniveau und Bio-Präferenz

```
In [78]: # chi^2 code
observed_data = []
for group, total, tips in tips_probability:
    observed_data.append([tips, total - tips])
```

```

observed = np.array(observed_data)

# Chi-Quadrat-Test durchführen
chi2, p_value, dof, expected = chi2_contingency(observed)

# Testergebnisse in Tabelle formatieren
test_results = [
    ["Chi-Quadrat Statistik", f"{chi2:.4f}"],
    ["p-Wert", f"{p_value:.4f}"],
    ["Freiheitsgrade", dof]
]

print("\nKontingenztabelle:")
contingency_table = []
for i, (group, total, tips) in enumerate(tips_probability):
    contingency_table.append([
        group,
        tips,
        total - tips,
        total
    ])

print(tabulate(contingency_table,
               headers=["Gruppe", "Mit Trinkgeld", "Ohne Trinkgeld", "Gesamt"],
               tablefmt="fancy_grid", numalign="right"))

print("\nChi-Quadrat Testergebnisse:")
print(tabulate(test_results, headers=["Metrik", "Wert"],
               tablefmt="fancy_grid", numalign="right"))

alpha = 0.05
print("\nInterpretation:")
if p_value < alpha:
    interpretation = [
        ["Ergebnis", "Statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value:.4f}) < Signifikanzniveau ({alpha:.4f})"],
        ["Schlussfolgerung", "Bio-Anteil und Trinkgeldvergabe sind abhängig"]
    ]
else:
    interpretation = [
        ["Ergebnis", "Kein statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value:.4f}) > Signifikanzniveau ({alpha:.4f})"],
        ["Schlussfolgerung", "Bio-Anteil und Trinkgeldvergabe sind unabhängig"]
]

print(tabulate(interpretation, headers=["", ""],
               tablefmt="fancy_grid"))

```

Kontingenztabelle:

Gruppe	Mit Trinkgeld	Ohne Trinkgeld	Gesamt
0%	171310	271681	442991
1-25%	160968	214707	375675
26-50%	244978	294211	539189
51-75%	115561	122309	237870
76-99%	19024	17693	36717
100%	18272	22307	40579

Chi-Quadrat Testergebnisse:

Metrik	Wert
Chi-Quadrat Statistik	8639.07
p-Wert	0
Freiheitsgrade	5

Interpretation:

Ergebnis	Statistisch signifikanter Zusammenhang
Begründung	p-Wert (0.0000) < Signifikanzniveau (0.05)
Schlussfolgerung	Bio-Anteil und Trinkgeldvergabe sind abhängig

```
In [79]: # plot code
observed_matrix = []
group_labels = []
for group, total, tips in tips_probability:
    observed_matrix.append([tips, total - tips])
    group_labels.append(group)

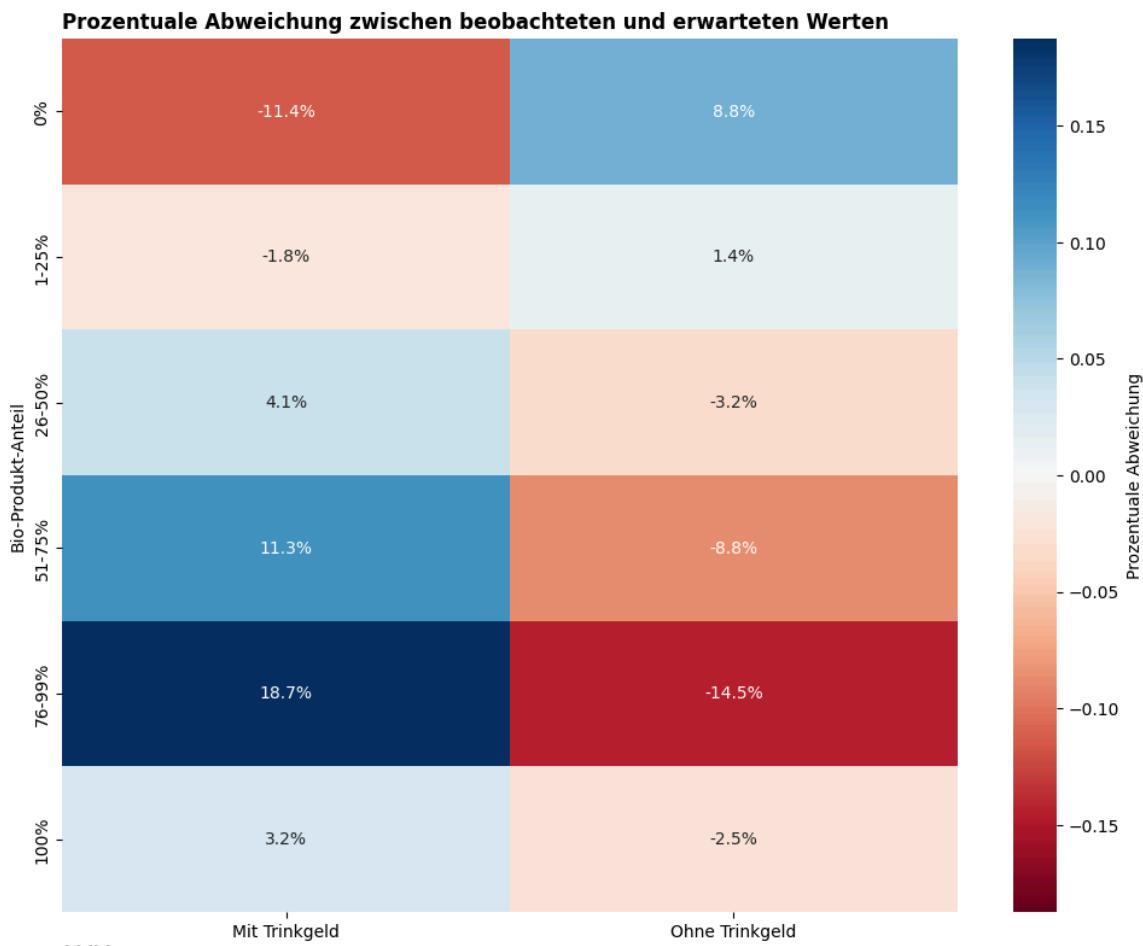
observed = np.array(observed_matrix)
col_labels = ['Mit Trinkgeld', 'Ohne Trinkgeld']

# Chi-Quadrat-Test und erwartete Häufigkeiten berechnen
_, _, _, expected = chi2_contingency(observed)

# Prozentuale Abweichung berechnen
percentage_deviation = ((observed - expected) / expected)

# DataFrame für die prozentualen Abweichungen erstellen
deviation_df = pd.DataFrame(
    percentage_deviation,
    index=group_labels,
    columns=col_labels)
```

```
)\n\n# Heatmap erstellen\nfig, ax = plt.subplots(figsize=(10, 8))\nplt.subplots_adjust(left=0.08)\n\n# Anpassung der Farbskala basierend auf den maximalen absoluten Werten\nmax_abs_value = np.max(np.abs(percentage_deviation))\nvmin = -max_abs_value\nvmax = max_abs_value\n\nsns.heatmap(deviation_df,\n            annot=True,\n            cmap='RdBu', # Geändert von 'RdBu' zu 'BuRd'\n            center=0,\n            fmt='.1%', # Geändert auf eine Dezimalstelle\n            cbar_kws={'label': 'Prozentuale Abweichung'},\n            vmin=vmin,\n            vmax=vmax,\n            ax=ax)\n\nax.set_title('Prozentuale Abweichung zwischen beobachteten und erwarteten\n              Werten',\n              loc='left',\n              weight='bold')\n\n# Beschreibung anpassen\ndescription = ('Die Heatmap visualisiert die prozentualen Abweichungen für\n  Negative Werte (rot) zeigen weniger Fälle als erwartet, '\n  'positive Werte (blau) mehr Fälle als erwartet. '\n  'Werte nahe 0% entsprechen etwa den erwarteten Häufigkeiten')\n\n# Textposition und -formatierung\nfig.text(0.05, 0, 'Abbildung:', weight='bold', ha='left')\nfig.text(0.05, -0.06, description, wrap=True)\n\n# Y-Achsen-Label anpassen\nax.set_ylabel('Bio-Produkt-Anteil')\n\nplt.tight_layout()\nplt.show()
```



Chi-Quadrat-Test: Bio-Produkte und Trinkgeldverhalten



Statistische Auswertung

Der durchgeführte Chi-Quadrat-Test bestätigt einen statistisch signifikanten Zusammenhang zwischen dem Bio-Produkt-Anteil in Bestellungen und der Trinkgeldwahrscheinlichkeit.



Interpretation der Heatmap:

- Rote Bereiche: Weniger Fälle als statistisch erwartet
- Blaue Bereiche: Mehr Fälle als statistisch erwartet
- Werte nahe 0%: Entsprechen den statistischen Erwartungen



Bedeutung für das Geschäft:

- Klare Abhängigkeit zwischen Bio-Anteil und Trinkgeldverhalten
- Stärkere Abweichungen bei höheren Bio-Anteilen
- Potenzial für gezielte Marketing- und Verkaufsstrategien

2) Analyse des Trinkgeldverhaltens des Kunden

```
In [80]: # Benutzerstatistiken: Gesamtbestellungen, Bestellungen mit Trinkgeld und
user_stats = (
    session.query(
        Order.user_id,
        func.count(Order.order_id).label('total_orders'),
        func.sum(case((Order.tips == True, 1), else_=0)).label('orders_with_tips')
    )
    .group_by(Order.user_id)
    .order_by(Order.user_id)
    .all()
)

avg_order_size_subquery = (
    session.query(
        Order.user_id,
        func.count(Einkaufskorb.product_id).label('order_size')
    )
    .join(Einkaufskorb, Order.order_id == Einkaufskorb.order_id)
    .group_by(Order.order_id)
).subquery()

avg_order_size = (
    session.query(
        avg_order_size_subquery.c.user_id,
        func.avg(avg_order_size_subquery.c.order_size).label('avg_order_size')
    )
    .group_by(avg_order_size_subquery.c.user_id)
    .order_by(avg_order_size_subquery.c.user_id)
    .all()
)

avg_order_size_df = pd.DataFrame(avg_order_size, columns=['user_id', 'avg_order_size'])

user_stats_df = pd.DataFrame(user_stats, columns=['user_id', 'total_orders'])

user_stats_df['orders_without_tips'] = user_stats_df['total_orders'] - user_stats_df['orders_with_tips']

user_stats_df = user_stats_df.merge(avg_order_size_df, on='user_id')

user_stats_df['order_group'] = pd.cut(user_stats_df['total_orders'], bins=[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
user_stats_df['item_count_group'] = pd.cut(user_stats_df['avg_order_size'], bins=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

grouped_stats_df_item_count = user_stats_df.groupby('item_count_group', observed=True).agg(
    orders_with_tips=('orders_with_tips', 'sum'),
    orders_without_tips=('orders_without_tips', 'sum'),
    total_orders=('total_orders', 'sum')
).reset_index()

grouped_stats_df = user_stats_df.groupby('order_group', observed=True).agg(
    orders_with_tips=('orders_with_tips', 'sum'),
    orders_without_tips=('orders_without_tips', 'sum'),
    total_orders=('total_orders', 'sum')
).reset_index()

grouped_stats_df['tips_ratio'] = grouped_stats_df['orders_with_tips'] / grouped_stats_df['total_orders']
```

```
grouped_stats_df_item_count['tips_ratio'] = grouped_stats_df_item_count['  
#
```

In [81]: `grouped_stats_df`

Out[81]:

	<code>order_group</code>	<code>orders_with_tips</code>	<code>orders_without_tips</code>	<code>total_orders</code>	<code>tips_ratio</code>
0	[1, 11)	89983	236698	326681	0.275446
1	[11, 21)	133246	226480	359726	0.370410
2	[21, 31)	122669	141444	264113	0.464457
3	[31, 41)	103809	88585	192394	0.539565
4	[41, 51)	91546	67904	159450	0.574136
5	[51, 61)	64700	49222	113922	0.567932
6	[61, 71)	39767	34737	74504	0.533757
7	[71, 81)	25089	25303	50392	0.497877
8	[81, 91)	19023	19412	38435	0.494940
9	[91, 101)	40281	53123	93404	0.431256

In [82]: `grouped_stats_df_item_count`

Out[82]:

	<code>item_count_group</code>	<code>orders_with_tips</code>	<code>orders_without_tips</code>	<code>total_orders</code>	<code>tips_ratio</code>
0	[1, 4)	65194	110654	175848	0.370741
1	[4, 7)	154210	199935	354145	0.435443
2	[7, 10)	185093	226977	412070	0.449179
3	[10, 13)	141483	170454	311937	0.453563
4	[13, 16)	84475	103825	188300	0.448619
5	[16, 19)	47323	59990	107313	0.440981
6	[19, 22)	26517	33158	59675	0.444357
7	[22, 25)	12662	18285	30947	0.409151
8	[25, 28)	6394	9226	15620	0.409347
9	[28, 31)	3439	5451	8890	0.386839



Analyse des Kundentrinkgeldverhaltens in Abhängigkeit der Bestellhäufigkeit



Untersuchungsfokus

In dieser Analyse untersuchen wir den Zusammenhang zwischen der Bestellhäufigkeit eines Kunden und seinem Trinkgeldverhalten. Dabei betrachten wir sowohl die absolute Verteilung als auch die relative Entwicklung der Trinkgeldquote.



Visualisierungsaufbau:

- **Linker Plot:** Gestapeltes Balkendiagramm zur Darstellung der absoluten Häufigkeiten
- **Rechter Plot:** Verlauf der Trinkgeldquote über die Bestellhäufigkeit
- Gruppierung der Kunden nach Anzahl ihrer Bestellungen in Intervallen

🎯 Analyseziele:

- Identifikation von Mustern im Trinkgeldverhalten von Stammkunden
- Untersuchung der Kundenverteilung nach Bestellhäufigkeit
- Analyse der Entwicklung der Trinkgeldquote bei steigender Bestellanzahl

```
In [83]: # plot code
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
plt.subplots_adjust(left=0.08)

# Linker Plot: Gestapelte Balkendiagramme
bars_with_tips = ax1.bar(grouped_stats_df['order_group'].astype(str), gro
                        width=0.35, color='lightgreen', label='Mit Trink
bars_without_tips = ax1.bar(grouped_stats_df['order_group'].astype(str),
                           width=0.35, bottom=grouped_stats_df['orders_w
                           color='salmon', label='Ohne Trinkgeld')

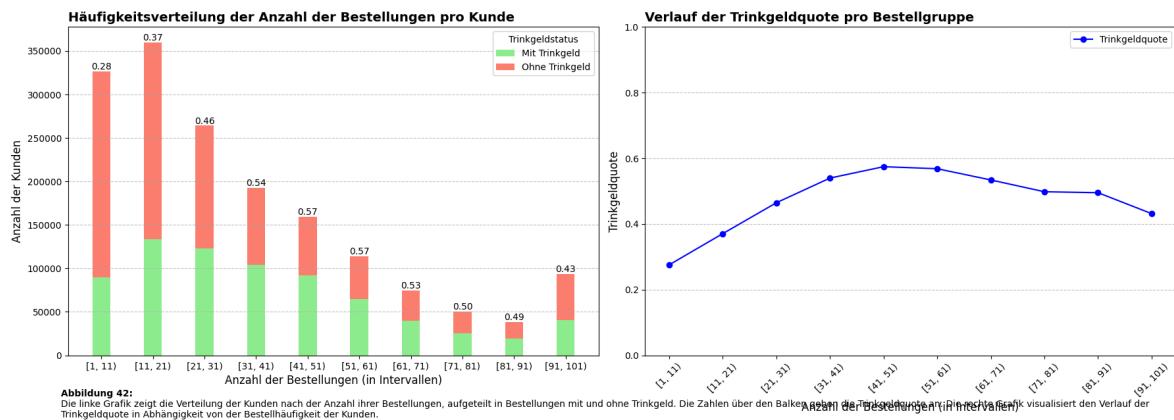
# Quotienten anzeigen
for i, rect in enumerate(bars_with_tips):
    total_height = rect.get_height() + bars_without_tips[i].get_height()
    ratio = grouped_stats_df['tips_ratio'].iloc[i]
    ax1.text(rect.get_x() + rect.get_width() / 2, total_height + 2,
             f'{ratio:.2f}', ha='center', va='bottom', fontsize=10)

# Formatierung linker Plot
ax1.set_title('Häufigkeitsverteilung der Anzahl der Bestellungen pro Kund',
              loc='left', weight='bold', fontsize=14)
ax1.set_xlabel('Anzahl der Bestellungen (in Intervallen)', fontsize=12)
ax1.set_ylabel('Anzahl der Kunden', fontsize=12)
ax1.legend(title='Trinkgeldstatus')
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)

# Rechter Plot: Liniendiagramm
ax2.plot(grouped_stats_df['order_group'].astype(str), grouped_stats_df['t
                    marker='o', color='blue', label='Trinkgeldquote')
ax2.set_title('Verlauf der Trinkgeldquote pro Bestellgruppe',
              loc='left', weight='bold', fontsize=14)
ax2.set_xlabel('Anzahl der Bestellungen (in Intervallen)', fontsize=12)
ax2.set_ylabel('Trinkgeldquote', fontsize=12)
ax2.set_ylim(0, 1)
ax2.legend()
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)

# Bildunterschrift
description = 'Die linke Grafik zeigt die Verteilung der Kunden nach der
fig.text(0.05, 0, 'Abbildung 42:', weight='bold', ha='left')
fig.text(0.05, -0.05, description, wrap=True)
```

```
plt.tight_layout()
plt.xticks(rotation=45)
plt.show()
```



📈 Interpretation der Kundenverteilung und Trinkgeldquoten

🔍 Zentrale Beobachtungen

📊 Kundenverteilung:

- Initial steigende Tendenz in den ersten beiden Bestellgruppen
- Danach kontinuierlicher Rückgang der Kundenanzahl
- Überraschender erneuter Anstieg in der letzten Gruppe (>50 Bestellungen)

💰 Entwicklung der Trinkgeldquote:

- Positive Korrelation zwischen Bestellhäufigkeit und Trinkgeldquote bis zur Gruppe 41-50
- Höchste Trinkgeldquote bei Kunden mit 41-50 Bestellungen
- Unerwarteter Rückgang der Quote bei Kunden mit mehr als 50 Bestellungen

💡 Mögliche Schlussfolgerungen:

- Stammkunden entwickeln tendenziell eine höhere Trinkgeldbereitschaft
- Es gibt eine "optimale" Bestellhäufigkeit für die Trinkgeldwahrscheinlichkeit
- Sehr häufige Besteller (>50) zeigen ein abweichendes Verhaltensmuster

```
In [84]: # Stelle sicher, dass 'order_group' eine reguläre Spalte ist (nicht der Index)
grouped_stats_df.reset_index(inplace=True)
```

```
# Setze 'order_group' als Index
crosstab = grouped_stats_df.set_index('order_group')[['orders_with_tips',
```

```
In [85]: # chi² test code
chi2, p, dof, expected = chi2_contingency(crosstab)

test_results = [
    ["Chi-Quadrat Statistik", f"{chi2}"],
    ["p-Wert", f"{p_value}"],
    ["Freiheitsgrade", dof]
]

print("\nChi-Quadrat Testergebnisse:")
print(tabulate(test_results, headers=["Metrik", "Wert"],
               tablefmt="fancy_grid", numalign="right"))

alpha = 0.05
print("\nInterpretation:")
if p_value < alpha:
    interpretation = [
        ["Ergebnis", "Statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value}) < Signifikanzniveau ({alpha})"],
        ["Schlussfolgerung", "Department und Trinkgeldvergabe sind abhängig"]
    ]
else:
    interpretation = [
        ["Ergebnis", "Kein statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value}) > Signifikanzniveau ({alpha})"],
        ["Schlussfolgerung", "Department und Trinkgeldvergabe sind unabhängig"]
]

print(tabulate(interpretation, headers=["", ""],
               tablefmt="fancy_grid"))
```

Chi-Quadrat Testergebnisse:

Metrik	Wert
Chi-Quadrat Statistik	74451.7
p-Wert	0
Freiheitsgrade	9

Interpretation:

Ergebnis	Statistisch signifikanter Zusammenhang
Begründung	p-Wert (0.0) < Signifikanzniveau (0.05)
Schlussfolgerung	Department und Trinkgeldvergabe sind abhängig

```
In [86]: # plot code
observed = crosstab[['orders_with_tips', 'orders_without_tips']].values
_, _, _, expected = chi2_contingency(observed)
```

```
# Prozentuale Abweichung berechnen
percentage_deviation = ((observed - expected) / expected)

deviation_df = pd.DataFrame(
    percentage_deviation,
    index=crosstab.index,
    columns=['Mit_Trinkgeld', 'Ohne_Trinkgeld']
)

fig, ax = plt.subplots(figsize=(8, 8))
plt.subplots_adjust(left=0.08)

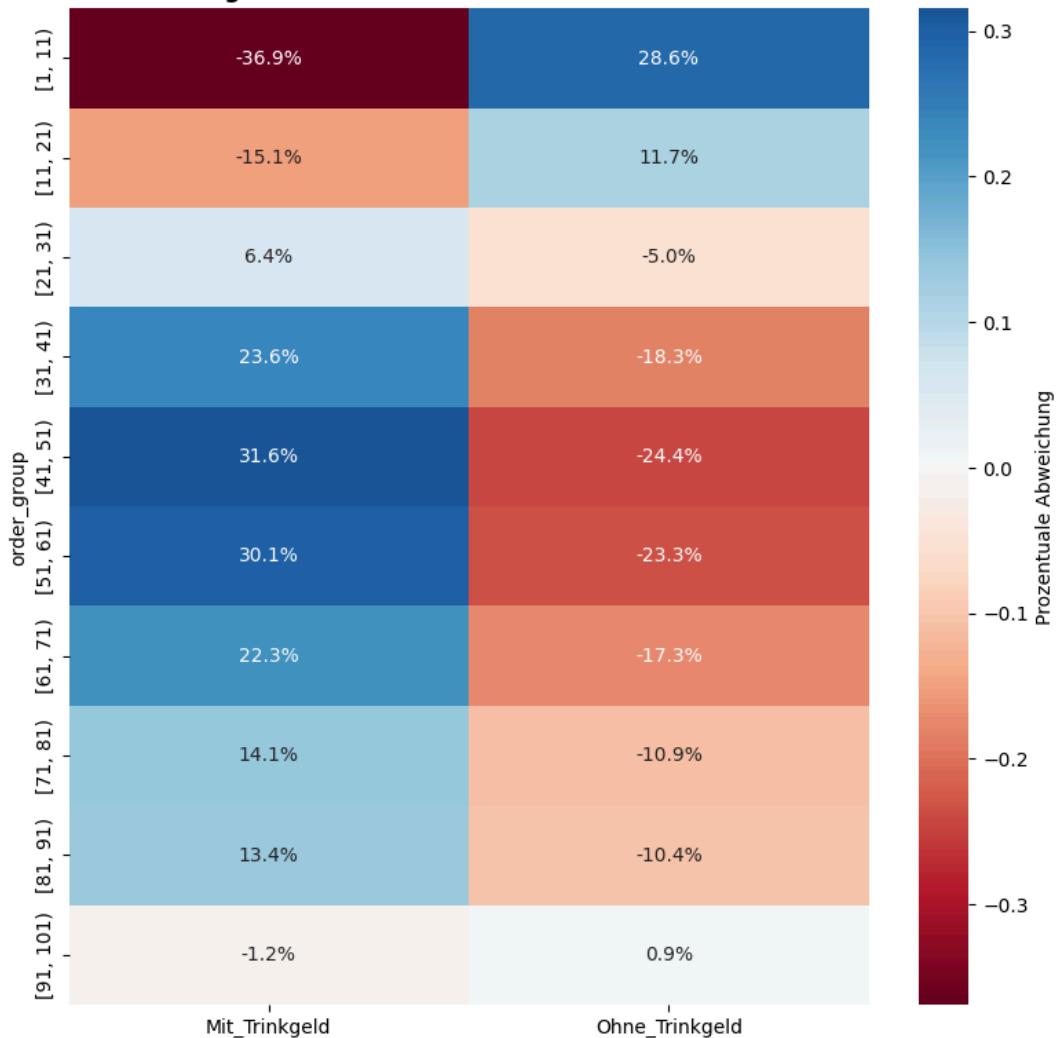
sns.heatmap(deviation_df,
            annot=True,
            cmap='RdBu',
            center=0,
            fmt='.1%',
            cbar_kws={'label': 'Prozentuale Abweichung'},
            ax=ax)

ax.set_title('Prozentuale Abweichung zwischen beobachteten und erwarteten',
             loc='left',
             weight='bold',
             x=-0.185)

description = 'Die Heatmap visualisiert die prozentualen Abweichungen für'
fig.text(0.05, 0, 'Abbildung 5:', weight='bold', ha='left')
fig.text(0.05, -0.06, description, wrap=True)

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

Prozentuale Abweichung zwischen beobachteten und erwarteten Werten

**Abbildung 5:**

Die Heatmap visualisiert die prozentualen Abweichungen für die verschiedenen Bestellungsgruppen. Negative Werte (rot) zeigen weniger Fälle als erwartet, positive Werte (blau) mehr Fälle als erwartet. Werte nahe 0% entsprechen etwa den erwarteten Häufigkeiten.



Detailanalyse der Trinkgeldquoten-Verteilung



Verteilungsanalyse mittels Boxplot

Um ein tieferes Verständnis der Trinkgeldverteilung zu erlangen, analysieren wir die Streuung der Trinkgeldquoten innerhalb verschiedener Kundengruppen mittels eines Boxplots.



Visualisierungselemente:

- **Box:** Zeigt das erste bis dritte Quartil mit dem Median
- **Whisker:** Erstrecken sich bis zu den Extremwerten (ohne Ausreißer)
- **Punkte:** Markieren statistische Ausreißer



Analysefokus:

- Verteilung der Trinkgeldquoten in verschiedenen Bestellhäufigkeitsgruppen

- Identifikation von Ausreißern und Extremwerten
- Erkennung von Mustern in der Streuung der Trinkgeldquoten

```
In [87]: # plot code
user_stats_df['order_group'] = pd.cut(
    user_stats_df['total_orders'],
    bins=range(1, 102, 10),
    right=False
)

user_stats_df['tip_ratio'] = user_stats_df['orders_with_tips'] / user_st
fig, ax = plt.subplots(figsize=(12, 6))
plt.subplots_adjust(left=0.08)

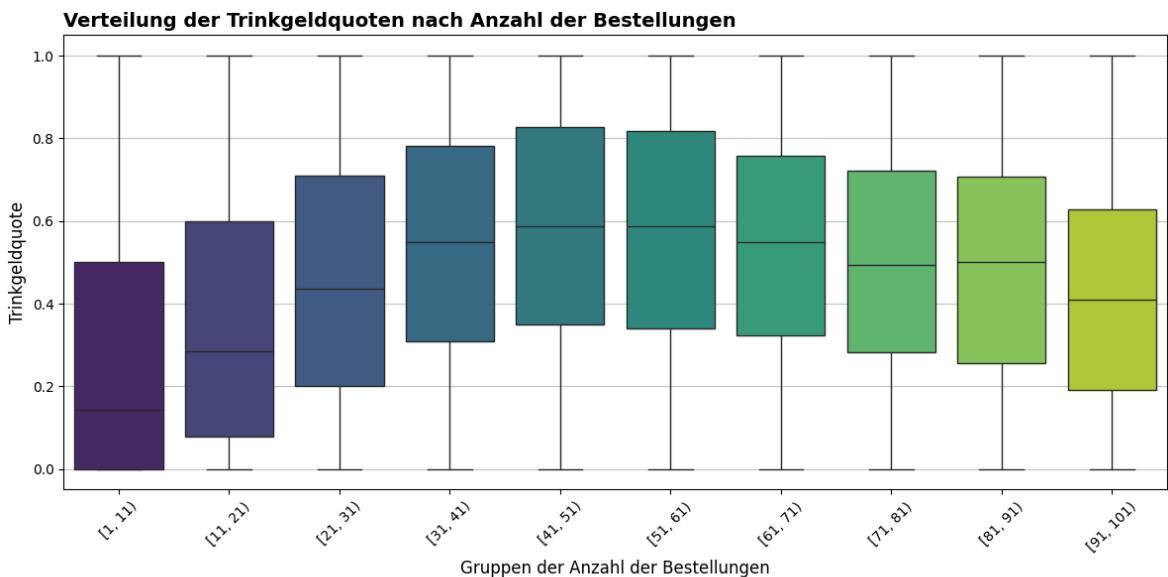
sns.boxplot(
    data=user_stats_df,
    x='order_group',
    y='tip_ratio',
    palette='viridis',
    hue='order_group',
    legend=False,
    ax=ax
)

ax.set_title('Verteilung der Trinkgeldquoten nach Anzahl der Bestellungen',
             loc='left', weight='bold', fontsize=14)
ax.set_xlabel('Gruppen der Anzahl der Bestellungen', fontsize=12)
ax.set_ylabel('Trinkgeldquote', fontsize=12)
ax.set_xticks(range(len(ax.get_xticklabels())))
ax.tick_params(axis='x', rotation=45)
ax.grid(axis='y', alpha=0.7)

description = 'Der Boxplot zeigt die Verteilung der Trinkgeldquoten für v
fig.text(0.08, 0, 'Abbildung 42:', weight='bold', ha='left')
fig.text(0.08, -0.05, description, wrap=True)

plt.tight_layout()
plt.show()

summary_stats = user_stats_df.groupby('order_group')['tip_ratio'].describ
markdown_table = tabulate(summary_stats, headers='keys', tablefmt='pipe',
                           display=Markdown(markdown_table))
```

**Abbildung 42:**

Der Boxplot zeigt die Verteilung der Trinkgeldquoten für verschiedene Gruppen von Kunden, eingeteilt nach ihrer Bestellhäufigkeit. Die Boxen zeigen den Median sowie das erste und dritte Quartil, die Whisker die Streuung der Daten, und die Punkte stellen Ausreißer dar.

```
/tmp/ipykernel_807097/2661236874.py:38: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
```

```
summary_stats = user_stats_df.groupby('order_group')['tip_ratio'].describe().reset_index()
```

order_group	count	mean	std	min	25%	50%	75%	max
[1, 11)	53685	0.268971	0.329745	0	0	0.142857	0.5	1
[11, 21)	24482	0.364814	0.319921	0	0.0785256	0.285714	0.6	1
[21, 31)	10618	0.461799	0.306749	0	0.2	0.434783	0.708333	1
[31, 41)	5491	0.538614	0.297895	0	0.307692	0.548387	0.78125	1
[41, 51)	3537	0.57336	0.295694	0	0.348837	0.586957	0.826087	1
[51, 61)	2072	0.56848	0.294252	0	0.339286	0.586207	0.818182	1
[61, 71)	1147	0.534291	0.285655	0	0.322581	0.548387	0.757576	1
[71, 81)	669	0.497768	0.279148	0	0.28169	0.493506	0.722222	1
[81, 91)	452	0.49548	0.288117	0	0.255962	0.5	0.708224	1
[91, 101)	951	0.431953	0.280161	0	0.191919	0.41	0.628263	1



Interpretation der Trinkgeldquoten-Verteilung



Detaillierte Beobachtungen



Bestätigung des Grundmusters:

- Medienverlauf entspricht den Erkenntnissen der vorherigen Analyse
- Generell steigende Tendenz der Trinkgeldquoten mit zunehmender Bestellhäufigkeit

 **Besondere Auffälligkeiten:**

- **Gruppe 1-10 Bestellungen:**
 - Schiefe Verteilung im Interquartilsbereich
 - Median näher am 25%-Quantil
 - Einzige Gruppe mit 25%-Quantil bei 0%
- **Durchgängiges Muster:**
 - In allen Gruppen gibt es Kunden mit 100% Trinkgeldquote
 - Parallel existieren in jeder Gruppe auch Kunden ohne Trinkgeldzahlungen

 **Zentrale Erkenntnisse:**

- Deutliche Unterschiede im Trinkgeldverhalten von Gelegenheits- und Stammkunden
- Konsistentes Verhalten einzelner Kunden (durchgehend mit/ohne Trinkgeld)
- Größte Variabilität im Trinkgeldverhalten bei Gelegenheitskunden (1-10 Bestellungen)

Analyse des Trinkgeldverhaltens in Abhängigkeit der Bestellgröße

Untersuchungsfokus

Nach der Analyse des Bestellverhaltens über die Zeit untersuchen wir nun den Zusammenhang zwischen der Bestellgröße (Anzahl der Produkte pro Bestellung) und dem Trinkgeldverhalten der Kunden.

Visualisierungsaufbau:

- **Linker Plot:** Absolute Verteilung der Bestellungen nach Größe und Trinkgeldstatus
- **Rechter Plot:** Entwicklung der Trinkgeldquote mit zunehmender Bestellgröße
- Gruppierung der Bestellungen nach Anzahl der enthaltenen Produkte

Kernfragen der Analyse:

- Gibt es einen Zusammenhang zwischen Bestellgröße und Trinkgeldwahrscheinlichkeit?
- Wie verteilen sich die Bestellgrößen im Gesamtdatensatz?
- Lassen sich kritische Schwellenwerte in der Bestellgröße identifizieren?

```
In [88]: # plot code
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))
plt.subplots_adjust(left=0.08)

bars_with_tips = ax1.bar(grouped_stats_df_item_count['item_count_group'],
                         grouped_stats_df_item_count['orders_with_tips'],
                         width=0.35, color='lightgreen', label='Mit Trinkgeld')
bars_without_tips = ax1.bar(grouped_stats_df_item_count['item_count_group'],
                            grouped_stats_df_item_count['orders_without_tips'],
                            width=0.35, bottom=grouped_stats_df_item_count['orders_with_tips'],
                            color='salmon', label='Ohne Trinkgeld')

for i, rect in enumerate(bars_with_tips):
    total_height = rect.get_height() + bars_without_tips[i].get_height()
    ratio = grouped_stats_df_item_count['tips_ratio'].iloc[i]
    ax1.text(rect.get_x() + rect.get_width() / 2, total_height + 2,
             f'{ratio:.2f}', ha='center', va='bottom', fontsize=10)

ax1.set_title('Häufigkeitsverteilung der Größe der Bestellungen',
              loc='left', weight='bold', fontsize=14)
ax1.set_xlabel('Anzahl der Produkte pro Bestellung (in Intervallen)', fontweight='bold')
ax1.set_ylabel('Anzahl der Bestellungen', fontsize=12)
ax1.legend(title='Trinkgeldstatus')
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)

ax2.plot(grouped_stats_df_item_count['item_count_group'].astype(str),
```

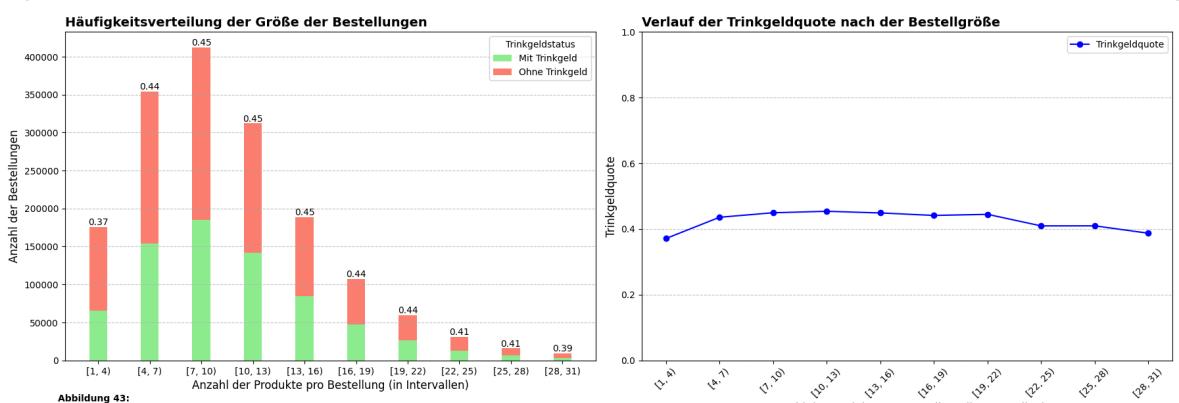
```

        grouped_stats_df_item_count['tips_ratio'],
        marker='o', color='blue', label='Trinkgeldquote')
ax2.set_title('Verlauf der Trinkgeldquote nach der Bestellgröße',
              loc='left', weight='bold', fontsize=14)
ax2.set_xlabel('Anzahl der Produkte pro Bestellung (in Intervallen)', fontweight='bold')
ax2.set_ylabel('Trinkgeldquote', fontsize=12)
ax2.set_ylim(0, 1)
ax2.legend()
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)

description = 'Die linke Grafik zeigt die Verteilung der Bestellungen nach der Anzahl der Produkte pro Bestellung (in Intervallen). Die rechte Grafik visualisiert den Verlauf der Trinkgeldquote in Abhängigkeit von der Bestellgröße.'
fig.text(0.05, 0, 'Abbildung 43:', weight='bold', ha='left')
fig.text(0.05, -0.05, description, wrap=True)

plt.tight_layout()
plt.xticks(rotation=45)
plt.show()

```



Interpretation der Bestellgrößen-Analyse



Zentrale Beobachtungen



Verteilung der Bestellgrößen:

- Deutliche Konzentration der Bestellungen im Bereich von 7-9 Produkten
- Diese Größenordnung scheint die bevorzugte Bestellmenge zu sein
- Abnehmende Häufigkeit bei sehr kleinen und sehr großen Bestellungen



Trinkgeldverhalten:

- Relativ konstante Trinkgeldquote über verschiedene Bestellgrößen
- Keine stark ausgeprägten Trends oder Muster erkennbar
- Geringe Schwankungen zwischen den Größengruppen



Weiterführende Analyse:

Um die statistische Signifikanz dieser Beobachtungen zu überprüfen, führen wir im Folgenden einen Chi-Quadrat-Test durch. Dieser wird klären, ob die Bestellgröße und das Trinkgeldverhalten statistisch voneinander abhängig sind.

```
In [89]: # Stelle sicher, dass 'order_group' eine reguläre Spalte ist (nicht der Index)
grouped_stats_df_item_count.reset_index(inplace=True)

# Setze 'order_group' als Index
crosstab = grouped_stats_df_item_count.set_index('item_count_group')[['order_group', 'trinkgeld']]
```



```
In [90]: # chi² code
chi2, p_value, dof, expected = chi2_contingency(crosstab)

test_results = [
    ["Chi-Quadrat Statistik", f"{chi2}"],
    ["p-Wert", f"{p_value}"],
    ["Freiheitsgrade", dof]
]

print("\nChi-Quadrat Testergebnisse:")
print(tabulate(test_results, headers=["Metrik", "Wert"],
               tablefmt="fancy_grid", numalign="right"))

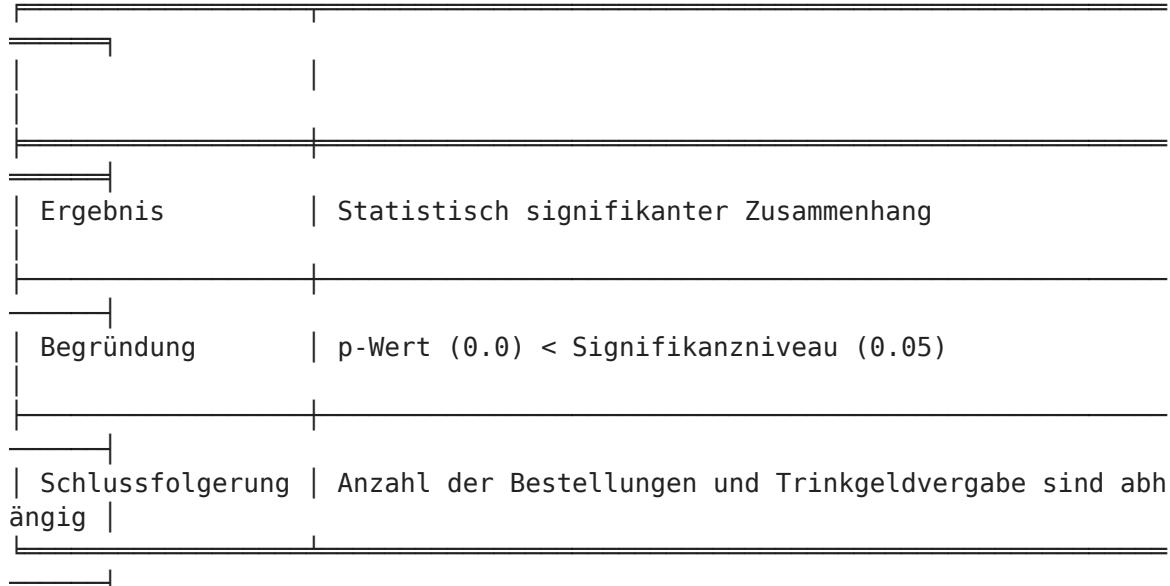
alpha = 0.05
print("\nInterpretation:")
if p_value < alpha:
    interpretation = [
        ["Ergebnis", "Statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value}) < Signifikanzniveau ({alpha})"],
        ["Schlussfolgerung", "Anzahl der Bestellungen und Trinkgeldvergabe"]
    ]
else:
    interpretation = [
        ["Ergebnis", "Kein statistisch signifikanter Zusammenhang"],
        ["Begründung", f"p-Wert ({p_value}) > Signifikanzniveau ({alpha})"],
        ["Schlussfolgerung", "Anzahl der Bestellungen und Trinkgeldvergabe"]
]

print(tabulate(interpretation, headers=["", ""],
               tablefmt="fancy_grid"))
```

Chi-Quadrat Testergebnisse:

Metrik	Wert
Chi-Quadrat Statistik	4097.7
p-Wert	0
Freiheitsgrade	9

Interpretation:



```
In [91]: # plot code
observed = crosstab[['orders_with_tips', 'orders_without_tips']].values

# Chi-Quadrat-Test durchführen für erwartete Häufigkeiten
_, _, _, expected = chi2_contingency(observed)

# Prozentuale Abweichung berechnen
percentage_deviation = ((observed - expected) / expected)

# DataFrame für die prozentualen Abweichungen erstellen
deviation_df = pd.DataFrame(
    percentage_deviation,
    index=crosstab.index,
    columns=['Mit_Trinkgeld', 'Ohne_Trinkgeld']
)

# Heatmap erstellen
fig, ax = plt.subplots(figsize=(8, 8))
plt.subplots_adjust(left=0.08)

sns.heatmap(deviation_df,
            annot=True,
            cmap='RdBu', # Geändert von 'RdBu' zu 'BuRd'
            center=0,
            fmt='.2%', # Geändert auf eine Dezimalstelle
            cbar_kws={'label': 'Prozentuale Abweichung'},
            ax=ax)

ax.set_title('Prozentuale Abweichung zwischen beobachteten und erwarteten')

```

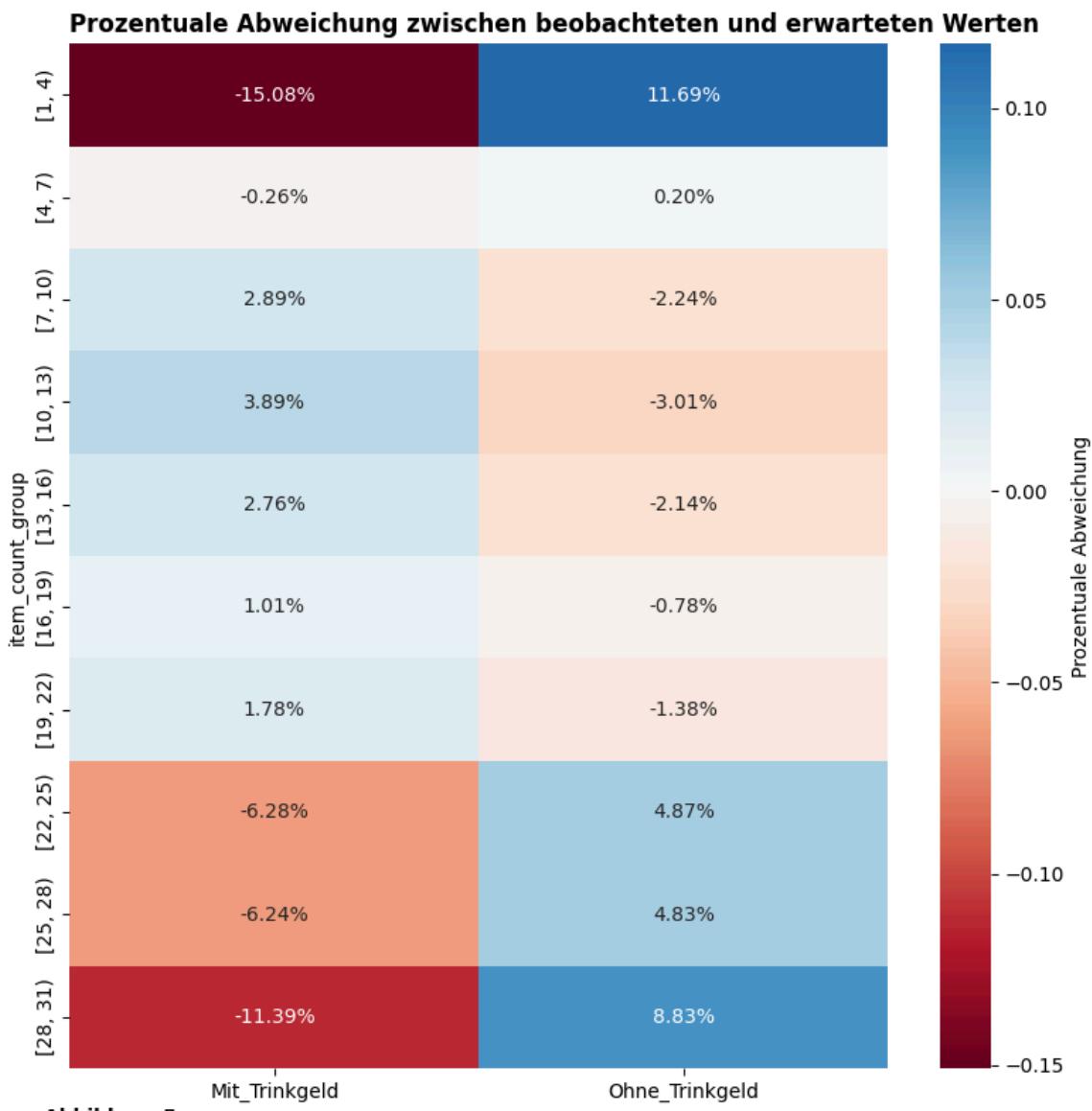
```

        loc='left',
        weight='bold')

description = 'Die Heatmap visualisiert die prozentualen Abweichungen für
fig.text(0.05, 0, 'Abbildung 5:', weight='bold', ha='left')
fig.text(0.05, -0.06, description, wrap=True)

plt.tight_layout()
plt.show()

```

**Abbildung 5:**

Die Heatmap visualisiert die prozentualen Abweichungen für verschiedene Artikelanzahl-Gruppen. Negative Werte (rot) zeigen weniger Fälle als erwartet, positive Werte (blau) mehr Fälle als erwartet. Werte nahe 0% entsprechen etwa den erwarteten Häufigkeiten.



Interpretation der Chi-Quadrat-Analyse



Statistische Erkenntnisse

Der Chi-Quadrat-Test bestätigt eine signifikante Abhängigkeit zwischen Bestellgröße und Trinkgeldverhalten, wobei die Effekte besonders an den Extremen der Verteilung deutlich werden.

Zentrale Beobachtungen:

- Statistisch signifikante Abhangigkeit nachgewiesen
- Starkste Abweichungen bei:
 - Sehr kleinen Bestellungen
 - Besonders groen Bestellungen
- Mittlere Bestellgroen zeigen geringere Abweichungen vom Erwartungswert

Implikationen:

- Das Trinkgeldverhalten wird besonders von extremen Bestellgroen beeinflusst
- Der Effekt ist bei durchschnittlichen Bestellgroen weniger ausgepragt
- Die Bestellgroe ist ein relevanter, aber nicht der einzige Einflussfaktor auf das Trinkgeldverhalten

```
In [92]: # plot code
user_stats_df['item_count_group'] = pd.cut(
    user_stats_df['avg_order_size'],
    bins=range(1, 32, 3),
    right=False
)

user_stats_df['tip_ratio'] = user_stats_df['orders_with_tips'] / user_st
fig, ax = plt.subplots(figsize=(12, 6))
plt.subplots_adjust(left=0.08)

sns.boxplot(
    data=user_stats_df,
    x='item_count_group',
    y='tip_ratio',
    palette='viridis',
    hue='item_count_group',
    legend=False,
    ax=ax
)

ax.set_title('Verteilung der Trinkgeldquoten nach Bestellgroe',
            loc='left', weight='bold', fontsize=14)
ax.set_xlabel('Anzahl der Produkte pro Bestellung (in Intervallen)', font
ax.set_ylabel('Trinkgeldquote', fontsize=12)
ax.set_xticks(range(len(ax.get_xticklabels())))
ax.tick_params(axis='x', rotation=45)
ax.grid(axis='y', alpha=0.7)

description = 'Der Boxplot visualisiert die Verteilung der Trinkgeldquote
fig.text(0.08, 0, 'Abbildung 44:', weight='bold', ha='left')
fig.text(0.08, -0.05, description, wrap=True)

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

```
summary_stats = user_stats_df.groupby('item_count_group')['tip_ratio'].describe()
display(Markdown(tabulate(summary_stats, headers='keys', tablefmt='pipe')))
```

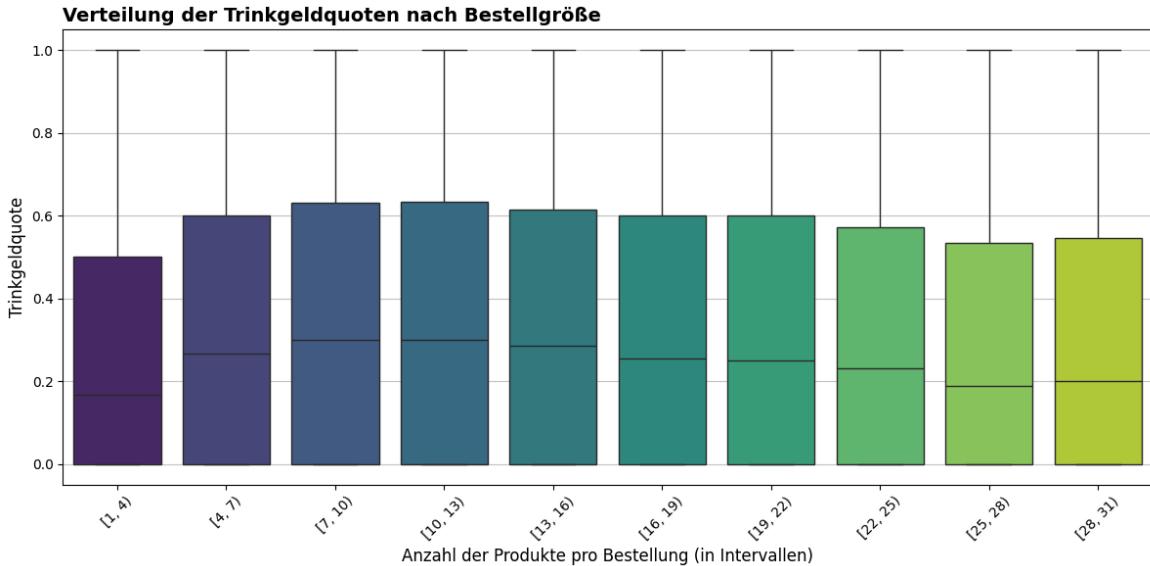


Abbildung 44:
Der Boxplot visualisiert die Verteilung der Trinkgeldquoten für verschiedene Bestellgrößen (Anzahl der Produkte pro Bestellung). Die Boxen zeigen den Median sowie das erste und dritte Quartil, die Whisker die Streuung der Daten, und die Punkte stellen Ausreißer dar.

```
/tmp/ipykernel_807097/280523961.py:38: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
```

```
summary_stats = user_stats_df.groupby('item_count_group')['tip_ratio'].describe().reset_index()
```

item_count_group	count	mean	std	min	25%	50%	75%	max
[1, 4)	13178	0.290442	0.331754	0	0	0.166667	0.5	1
[4, 7)	21991	0.351854	0.336874	0	0	0.266667	0.6	1
[7, 10)	23822	0.367149	0.334939	0	0	0.3	0.631579	1
[10, 13)	18155	0.369306	0.334755	0	0	0.3	0.634146	1
[13, 16)	11259	0.359589	0.333961	0	0	0.285714	0.615385	1
[16, 19)	6649	0.349186	0.332531	0	0	0.255102	0.6	1
[19, 22)	3759	0.345208	0.336782	0	0	0.25	0.6	1
[22, 25)	2022	0.322679	0.329247	0	0	0.230769	0.571429	1
[25, 28)	1077	0.302422	0.330092	0	0	0.1875	0.533333	1
[28, 31)	607	0.307518	0.328821	0	0	0.2	0.545455	1



Kategorisierung der Kunden nach Trinkgeldverhalten



Kundensegmentierung

Um ein detaillierteres Verständnis des Kundenverhaltens zu erlangen, kategorisieren wir die Kunden basierend auf ihrer individuellen Trinkgeldquote in fünf distinkte Gruppen.

Kategorien-Definition:

- **Nie Trinkgeld:** Kunden mit 0% Trinkgeldquote
- **Selten Trinkgeld:** Trinkgeldquote $\leq 25\%$
- **Gelegentlich Trinkgeld:** Trinkgeldquote 26-75%
- **Häufig Trinkgeld:** Trinkgeldquote 76-99%
- **Immer Trinkgeld:** Trinkgeldquote 100%

Analyseziele:

- Identifikation der Hauptkundengruppen nach Trinkgeldverhalten
- Erkennung von Verhaltensmustern in der Kundenbasis
- Grundlage für gezielte Marketing- und Service-Strategien

```
In [93]: # plot code
def categorize_tip_ratio(ratio):
    if ratio == 0:
        return "Nie Trinkgeld"
    elif ratio == 1:
        return "Immer Trinkgeld"
    elif ratio <= 0.25:
        return "Selten Trinkgeld"
    elif ratio <= 0.75:
        return "Gelegentlich Trinkgeld"
    elif ratio < 1.0:
        return "Häufig Trinkgeld"
    else:
        return "immer Trinkgeld"

user_stats_df['tip_category'] = user_stats_df['tip_ratio'].apply(categorize_tip_ratio)

tip_category_counts = user_stats_df['tip_category'].value_counts()

fig, ax = plt.subplots(figsize=(12, 8))
plt.subplots_adjust(left=0.08)

category_order = ["Nie Trinkgeld", "Selten Trinkgeld", "Gelegentlich Trinkgeld",
                  "Häufig Trinkgeld", "Immer Trinkgeld"]

tip_category_counts = tip_category_counts.reindex(category_order)

colors = plt.cm.viridis(np.linspace(0, 1, len(category_order)))

wedges, texts, autotexts = ax.pie(
    tip_category_counts.values,
    labels=None,
    autopct='%1.1f%%',
    colors=colors,
    startangle=90)

ax.set_title('Verteilung der Benutzer basierend auf Trinkgeld-Verhalten',
             loc='left', weight='bold', x=-0.025)

legend_elements_1 = [
```

```
Patch(facecolor=colors[i], label=f'{cat} ({n=tip_category_counts[cat]})')
]

legend_1 = ax.legend(
    handles=legend_elements_1,
    title="Trinkgeld-Verhalten",
    loc="center left",
    bbox_to_anchor=(0.95, 0.8),
    frameon=True
)

ratio_explanations = [
    "ratio = 0".ljust(20),
    "0 < ratio ≤ 0.25".ljust(20),
    "0.25 < ratio ≤ 0.75".ljust(20),
    "0.75 < ratio < 1".ljust(20),
    "ratio = 1".ljust(20)
]

legend_elements_2 = [
    Patch(facecolor=colors[i], label=exp)
    for i, exp in enumerate(ratio_explanations)
]

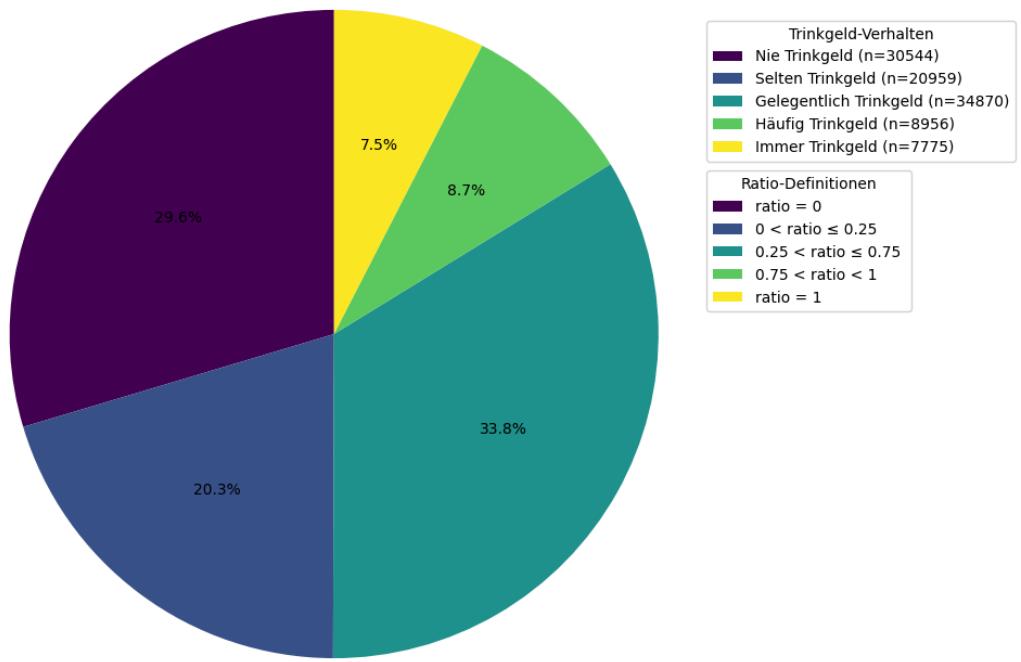
legend_2 = ax.legend(
    handles=legend_elements_2,
    title="Ratio-Definitionen",
    loc='center left',
    bbox_to_anchor=(0.95, 0.615),
    frameon=True
)

ax.add_artist(legend_1)

description = 'Das Kreisdiagramm zeigt die prozentuale Verteilung der Ben'
fig.text(0.1, 0.05, 'Abbildung 45:', weight='bold', ha='left')
fig.text(0.1, -0.015, description, wrap=True)

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

Verteilung der Benutzer basierend auf Trinkgeld-Verhalten

**Abbildung 45:**

Das Kreisdiagramm zeigt die prozentuale Verteilung der Benutzer nach ihrem Trinkgeld-Verhalten. Die Benutzer wurden in fünf Kategorien eingeteilt, basierend auf dem Verhältnis ihrer Bestellungen mit Trinkgeld zur Gesamtzahl ihrer Bestellungen (ratio). Die erste Legende zeigt die absolute Anzahl der Benutzer pro Kategorie, die zweite Legende erklärt die Ratio-Bereiche der Kategorien.



Analyse der Bestellhäufigkeit nach Trinkgeld-Kategorien



Verteilungsanalyse

Nach der Kategorisierung der Kunden nach ihrem Trinkgeldverhalten untersuchen wir nun, wie sich die Anzahl der Bestellungen innerhalb dieser Gruppen verteilt.



Visualisierungselemente:

- Box:** Zeigt IQR (25%-75% Quartil) mit Medianline
- Whisker:** Reichweite der Daten (ohne Ausreißer)
- Punkte:** Einzelne Ausreißer
- Farbskala:** Kategorien von "Nie" bis "Immer" Trinkgeld



Untersuchungsfokus:

- Zusammenhang zwischen Bestellhäufigkeit und Trinkgeldverhalten
- Identifikation von Unterschieden in der Bestellaktivität
- Erkennung von Ausreißern in den verschiedenen Kategorien

```
In [94]: # plot code
category_order = ["Nie Trinkgeld", "Selten Trinkgeld", "Gelegentlich Trin
```

```

    "Häufig Trinkgeld", "Immer Trinkgeld"]

group_counts = user_stats_df['tip_category'].value_counts()

user_stats_df['tip_category'] = pd.Categorical(user_stats_df['tip_category'])
user_stats_df = user_stats_df.sort_values(by='tip_category')

fig, ax = plt.subplots(figsize=(14, 8))
plt.subplots_adjust(left=0.08)

sns.boxplot(
    data=user_stats_df,
    x='tip_category',
    y='total_orders',
    order=category_order,
    palette='viridis',
    ax=ax,
    hue='tip_category'
)

ax.set_title('Verteilung der Bestellungen nach Trinkgeld-Kategorien',
             loc='left', weight='bold', fontsize=16)
ax.set_xlabel('Trinkgeld-Kategorie', fontsize=12)
ax.set_ylabel('Anzahl der Bestellungen', fontsize=12)
ax.set_xticks(range(len(ax.get_xticklabels())))
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

legend_elements_1 = [
    Patch(facecolor=sns.color_palette('viridis', len(category_order))[i],
          label=f'{cat.ljust(20)} ({n=group_counts[cat]}:{<3})')
    for i, cat in enumerate(category_order)
]

legend_1 = ax.legend(
    handles=legend_elements_1,
    title="Trinkgeld-Gruppen (Anzahl Benutzer)".ljust(20),
    loc='upper right',
    bbox_to_anchor=(1.35, 1),
    frameon=True
)

ratio_explanations = [
    "ratio = 0".ljust(20),
    "0 < ratio ≤ 0.25".ljust(20),
    "0.25 < ratio ≤ 0.75".ljust(20),
    "0.75 < ratio < 1".ljust(20),
    "ratio = 1".ljust(20)
]

legend_elements_2 = [
    Patch(facecolor=sns.color_palette('viridis', len(ratio_explanations)))
    for i, exp in enumerate(ratio_explanations)
]

legend_2 = ax.legend(
    handles=legend_elements_2,
    title="Ratio-Definitionen".ljust(51),
    loc='lower right',
    bbox_to_anchor=(1.35, 0.5),
    frameon=True
)

```

```

)
ax.add_artist(legend_1)
ax.grid(axis='y', linestyle='--', alpha=0.9)

description = 'Der Boxplot zeigt die Verteilung der Bestellanzahl für ver-'
fig.text(0.05, 0, 'Abbildung 46:', weight='bold', ha='left')
fig.text(0.05, -0.06, description, wrap=True)

plt.tight_layout()
plt.show()

summary_stats = user_stats_df.groupby('tip_category')['total_orders'].des-
markdown_table = tabulate(summary_stats, headers='keys', tablefmt='pipe', 

display(Markdown(markdown_table))

```

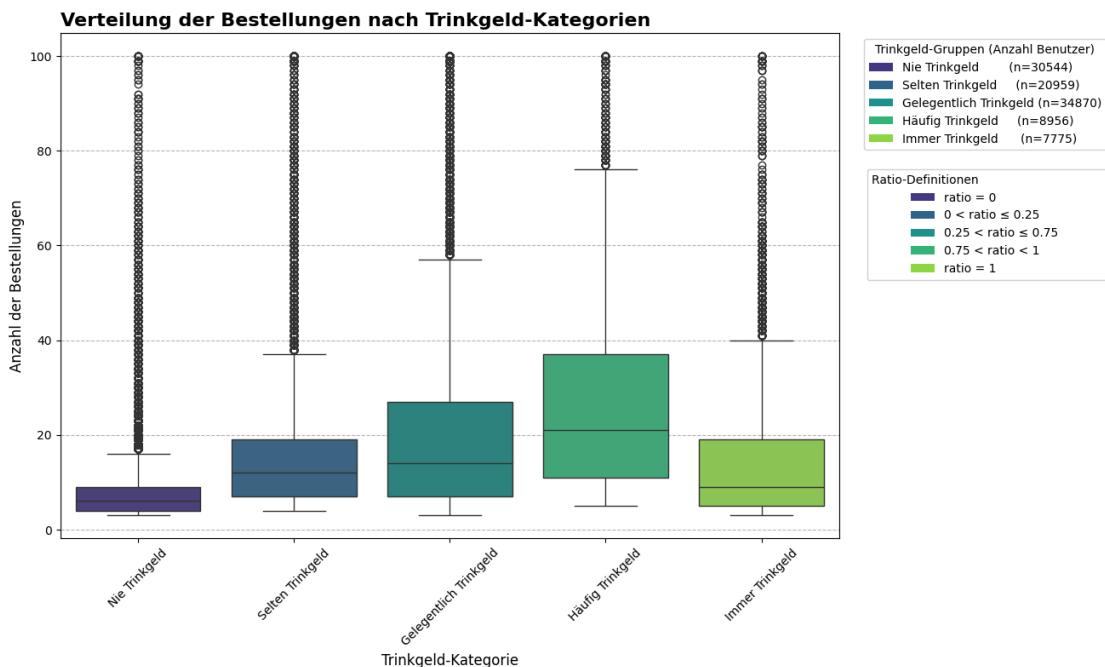


Abbildung 46:
Der Boxplot zeigt die Verteilung der Bestellanzahl für verschiedene Trinkgeld-Kategorien. Die Benutzer wurden basierend auf ihrem Trinkgeld-Verhalten in fünf Gruppen eingeteilt. Die erste Legende zeigt die Anzahl der Benutzer pro Kategorie, die zweite Legende erklärt die Ratio-Bereiche der Kategorien. Die Boxen zeigen den Median sowie das erste und dritte Quartil, die Whisker die Streuung der Daten, und die Punkte stellen Ausreißer dar.

```
/tmp/ipykernel_807097/127993441.py:75: FutureWarning: The default of obser-
ved=False is deprecated and will be changed to True in a future version of
pandas. Pass observed=False to retain current behavior or observed=True to
adopt the future default and silence this warning.
```

```
summary_stats = user_stats_df.groupby('tip_category')['total_orders'].de-
scribe().reset_index()
```

tip_category	count	mean	std	min	25%	50%	75%	max
Nie Trinkgeld	30544	8.42267	8.31828	3	4	6	9	100
Selten Trinkgeld	20959	16.1157	15.583	4	7	12	19	100
Gelegentlich Trinkgeld	34870	20.8696	19.0922	3	7	14	27	100
Häufig Trinkgeld	8956	25.9238	19.3059	5	11	21	37	100
Immer Trinkgeld	7775	15.1888	15.5573	3	5	9	19	100



Interpretation der Bestellhäufigkeitsverteilung



Zentrale Beobachtungen

 **Entwicklung des Medians:**

- Kontinuierlicher Anstieg von "Nie Trinkgeld" bis "Häufig Trinkgeld"
- Unerwarteter Rückgang in der Gruppe "Immer Trinkgeld"
- Höchster Median in der Kategorie "Häufig Trinkgeld"

 **Streuungsverhalten:**

- Zunehmende Varianz bis zur Kategorie "Häufig Trinkgeld"
- Größte Streuung bei Kunden mit häufigen Trinkgeldzahlungen
- Breite Interquartilsabstände in den mittleren Kategorien

 **Schlussfolgerungen:**

- Positive Korrelation zwischen Bestellhäufigkeit und Trinkgeldneigung bis zur vorletzten Gruppe
- Kunden mit häufigen Trinkgeldzahlungen zeigen das diverseste Bestellverhalten
- Die "Immer Trinkgeld"-Gruppe zeigt ein abweichendes Muster vom allgemeinen Trend

```
In [95]: # show df
category_order = ["Nie Trinkgeld", "Selten Trinkgeld", "Gelegentlich Trin
              "Häufig Trinkgeld", "Immer Trinkgeld"]

group_counts = user_stats_df['tip_category'].value_counts()

user_stats_df['tip_category'] = pd.Categorical(user_stats_df['tip_categor
user_stats_df = user_stats_df.sort_values(by='tip_category')
user_stats_df
```

Out[95]:

	<code>user_id</code>	<code>total_orders</code>	<code>orders_with_tips</code>	<code>orders_without_tips</code>	<code>avg_order_size</code>	<code>order_size_min</code>
0	1	11	0	11	6.363636	1.0
12309	24308	18	0	18	7.055556	1.0
99315	198758	5	0	5	3.800000	1.0
85833	171397	10	0	10	8.300000	1.0
85832	171396	4	0	4	3.250000	1.0
...
43099	85859	26	26	0	3.769231	1.0
81396	162495	17	17	0	13.058824	1.0
15395	30554	64	64	0	13.296875	1.0
35446	70553	5	5	0	11.400000	1.0
86135	172000	6	6	0	14.000000	1.0

103104 rows × 9 columns



📊 Analyse der durchschnittlichen Bestellgröße nach Trinkgeld-Kategorien

🔍 Abschließende Untersuchung

Als letzten Aspekt unserer Analyse untersuchen wir den Zusammenhang zwischen der durchschnittlichen Bestellgröße (Anzahl Produkte pro Bestellung) und dem Trinkgeldverhalten der Kunden.

📈 Analyseaspekte:

- Vergleich der Bestellgrößen zwischen den Trinkgeld-Kategorien
- Untersuchung der Streuung innerhalb der Kategorien
- Identifikation möglicher Zusammenhänge zwischen Bestellgröße und Trinkgeldverhalten

🎯 Zentrale Fragen:

- Bestellen Kunden mit höherer Trinkgeldneigung durchschnittlich mehr Produkte?
- Gibt es charakteristische Unterschiede in der Bestellgröße zwischen den Kategorien?
- Welche Rolle spielt die Bestellgröße beim Trinkgeldverhalten?

```
In [96]: # plot code
summary_stats = user_stats_df.groupby('tip_category')['avg_order_size'].d
markdown_table = tabulate(summary_stats, headers='keys', tablefmt='pipe',


fig, ax = plt.subplots(figsize=(14, 6))
plt.subplots_adjust(left=0.08)

sns.boxplot(
    data=user_stats_df,
    x='tip_category',
    y='avg_order_size',
    order=category_order,
    palette='viridis',
    ax=ax,
    hue='tip_category'
)

ax.set_title('Verteilung der durchschnittlichen Bestellgröße nach Trinkgeld-Kategorie', loc='left', weight='bold', fontsize=16)
ax.set_xlabel('Trinkgeld-Kategorie', fontsize=12)
ax.set_ylabel('Durchschnittliche Bestellgröße', fontsize=12)
ax.set_xticks(range(len(ax.get_xticklabels())))
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

legend_elements_1 = [
    Patch(facecolor=sns.color_palette('viridis', len(category_order))[i],
          label=f'{cat.ljust(20)} ({group_counts[cat]:<3})')
    for i, cat in enumerate(category_order)
]

legend_1 = ax.legend(
    handles=legend_elements_1,
    title="Trinkgeld-Gruppen (Anzahl Benutzer)".ljust(20),
    loc='upper right',
    bbox_to_anchor=(1.35, 1),
    frameon=True
)

ratio_explanations = [
    "ratio = 0".ljust(20),
    "0 < ratio ≤ 0.25".ljust(20),
    "0.25 < ratio ≤ 0.75".ljust(20),
    "0.75 < ratio < 1".ljust(20),
    "ratio = 1".ljust(20)
]

legend_elements_2 = [
    Patch(facecolor=sns.color_palette('viridis', len(ratio_explanations))[i],
          label=ratio_explanations[i])
    for i, exp in enumerate(ratio_explanations)
]

legend_2 = ax.legend(
    handles=legend_elements_2,
    title="Ratio-Definitionen".ljust(51),
    loc='lower right',
    bbox_to_anchor=(1.35, 0.28),
    frameon=True
)
```

```

ax.add_artist(legend_1)
ax.grid(axis='y', linestyle='--', alpha=0.9)

description = 'Der Boxplot zeigt die Verteilung der durchschnittlichen Be'
fig.text(0.05, 0, 'Abbildung 47:', weight='bold', ha='left')
fig.text(0.05, -0.075, description, wrap=True)

plt.tight_layout()
plt.show()

display(Markdown(markdown_table))

```

/tmp/ipykernel_807097/14700677.py:2: FutureWarning: The default of observe=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

summary_stats = user_stats_df.groupby('tip_category')['avg_order_size'].describe().reset_index()

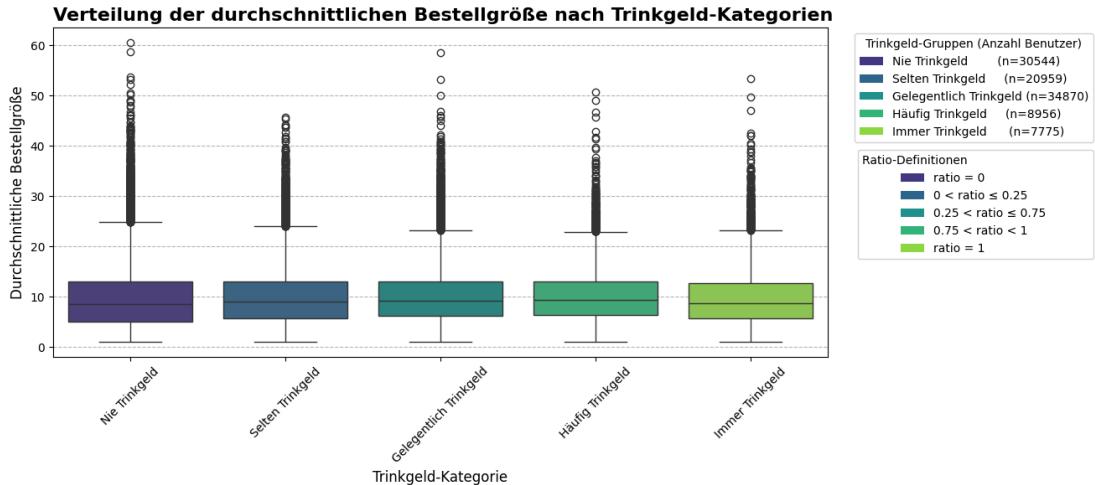


Abbildung 47:
Der Boxplot zeigt die Verteilung der durchschnittlichen Bestellgrößen (Anzahl Produkte pro Bestellung) für verschiedene Trinkgeld-Kategorien. Die erste Legende zeigt die Anzahl der Benutzer pro Kategorie, die zweite Legende erklärt die Ratio-Bereiche der Kategorien. Die Boxen zeigen den Median sowie das erste und dritte Quartil, die Whisker die Streuung der Daten, und die Punkte stellen Ausreißer dar.

tip_category	count	mean	std	min	25%	50%	75%	max
Nie Trinkgeld	30544	9.77409	6.30377	1	5.11111	8.6	13	60.5
Selten Trinkgeld	20959	10.0188	5.85337	1	5.75894	9	13.0625	45.75
Gelegentlich Trinkgeld	34870	10.1248	5.57082	1	6.16667	9.16667	13	58.5
Häufig Trinkgeld	8956	10.2351	5.4301	1	6.3837	9.31401	13	50.7083
Immer Trinkgeld	7775	9.81899	5.73614	1	5.71429	8.76471	12.7071	53.2759

Ergebnisse der Bestellgrößen-Analyse

Zentrale Erkenntnisse

Die Analyse der durchschnittlichen Bestellgrößen zeigt überraschend deutlich, dass kein signifikanter Zusammenhang zwischen Trinkgeldverhalten und Bestellgröße besteht.

Kernbeobachtungen:

- Nahezu identische Mediane über alle Trinkgeld-Kategorien hinweg
- Ähnliche Streuungsmuster in allen Gruppen
- Keine erkennbaren systematischen Unterschiede zwischen den Kategorien

Schlussfolgerung:

- Die Zugehörigkeit zu einer Trinkgeld-Kategorie hat keinen erkennbaren Einfluss auf die durchschnittliche Bestellgröße
- Trinkgeldverhalten scheint von anderen Faktoren als der Bestellgröße abhängig zu sein
- Kunden bestellen unabhängig von ihrem Trinkgeldverhalten ähnliche Mengen an Produkten

```
In [97]: # Liste aller Benutzer die nur eine einzige Bestellung getätigt haben
users_with_one_order = session.query(Order.user_id) \
    .group_by(Order.user_id) \
    .having(func.count(Order.order_id) == 1) \
    .all()

print(f"Anzahl der Benutzer mit nur einer Bestellung: {len(users_with_one_order)}")
Anzahl der Benutzer mit nur einer Bestellung: 0
```

Konkrete Fragen

Frage 1:

Hängt die Wahrscheinlichkeit, dass bei einer Bestellung Trinkgeld gegeben wird, vom Trinkgeldverhalten bei früheren Bestellungen desselben Bestellers ab?

- a) Gibt es einen Zusammenhang bezüglich der vorhergehenden Bestellung?

```
In [98]: result = session.query(Order.order_id, Order.user_id, Order.order_number, Order.tip) \
    .filter(Order.tip != None) \
    .order_by(Order.user_id, Order.order_number)

orders = pd.DataFrame(result, columns=['order_id', 'user_id', 'order_number', 'tip'])
```

```
In [99]: orders = orders.sort_values(by=['user_id', 'order_number'])

# Spalte für vorherige und vor-vorherige Trinkgeldangaben
orders['tip_prev'] = orders.groupby('user_id')['tip'].shift(1)
orders['tip_prev2'] = orders.groupby('user_id')['tip'].shift(2)
```

```
In [100...]: pd.crosstab(orders['tip_prev'], orders['tip'], margins=True, normalize='all')
```

Out[100... tip False True All

tip_prev

	False	True	All
False	0.427425	0.130028	0.557452
True	0.131364	0.311183	0.442548
All	0.558789	0.441211	1.000000

In [101... pd.crosstab(orders['tip'], orders['tip_prev'], margins=True, normalize='co

Out[101... tip_prev False True All

tip

	False	True	All
False	0.766746	0.296836	0.558789
True	0.233254	0.703164	0.441211

Gibt ein Kunde bei der vorherigen Bestellung Trinkgeld, beträgt die Wahrscheinlichkeit für Trinkgeld bei der aktuellen Bestellung 71%, was auf einen Zusammenhang hinweist.

- b) Gibt es einen Zusammenhang bezüglich der vor-vorherigen Bestellung?

In [102... pd.crosstab(orders['tip'], orders['tip_prev2'], margins=True, normalize='co

Out[102... tip_prev2 False True All

tip

	False	True	All
False	0.737784	0.324889	0.552248
True	0.262216	0.675111	0.447752

Ähnlich wie in a) beträgt die Wahrscheinlichkeit für die aktuelle Bestellung 68%, wenn der Kunde bei der vorvorherigen Bestellung Trinkgeld gegeben hat.

- c) Liefert das Trinkgeldverhalten der vor-vorhergehende Bestellung Informationen auch über das hinaus, was bereits aus der vorhergehenden Bestellung abgelesen werden kann?

In [103... pd.crosstab([orders['tip_prev2'], orders['tip_prev']], orders["tip"] ,marg

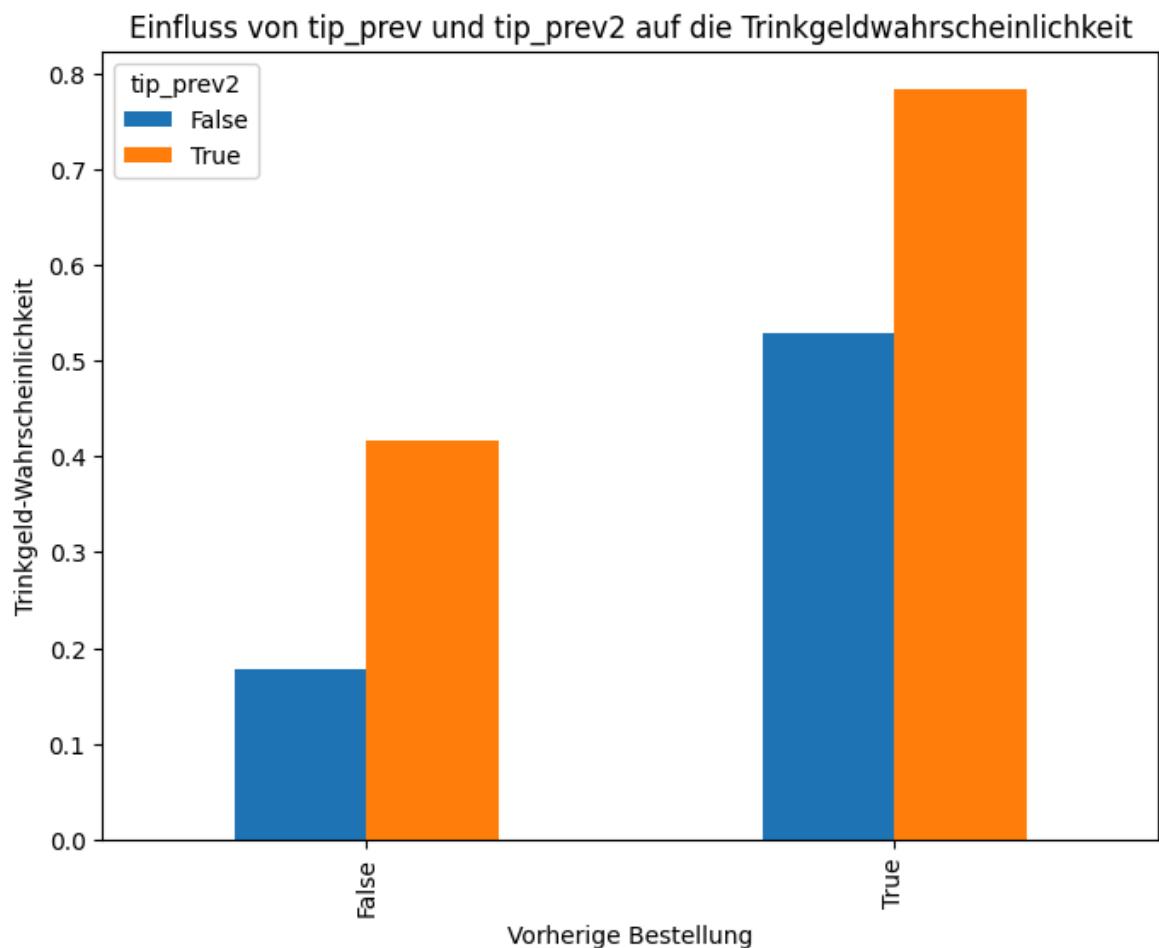
Out[103...]

	tip	False	True
tip_prev2	tip_prev		
False	False	0.821642	0.178358
	True	0.470954	0.529046
True	False	0.582866	0.417134
	True	0.216545	0.783455
All		0.552248	0.447752

Die höchste Trinkgeldwahrscheinlichkeit tritt auf, wenn tip_prev=True und tip_prev2=True, was zeigt, dass tip_prev2 unabhängig von tip_prev zusätzliche Informationen über das Trinkgeldverhalten liefert.

In [104...]

```
grouped = orders.groupby(['tip_prev', 'tip_prev2'])['tip'].mean().unstack()
grouped.plot(kind='bar', stacked=False, figsize=(8, 6), color=['#1f77b4'],
plt.title('Einfluss von tip_prev und tip_prev2 auf die Trinkgeldwahrscheinlichkeit')
plt.ylabel('Trinkgeld-Wahrscheinlichkeit')
plt.xlabel('Vorherige Bestellung')
plt.legend(title='tip_prev2')
plt.show()
```



Schlussfolgerungen:

- Es besteht ein starker Zusammenhang zwischen dem Trinkgeldverhalten bei der vorherigen Bestellung und der aktuellen Bestellung.

- Wenn in den letzten beiden Bestellungen Trinkgeld gegeben wurde, erhöht sich die Wahrscheinlichkeit, dass auch in der aktuellen Bestellung Trinkgeld gegeben wird, zusätzlich und erreicht 78,88%.

Frage 2:

- a) Gibt es einen Zusammenhang zwischen dem Trinkgeldverhalten und den Departments, aus denen bestellt wird?

```
In [4]: aisles = pd.read_csv('../Daten/aisles.csv')
departments = pd.read_csv('../Daten/departments.csv')
order_products = pd.read_csv('../Daten/order_products.csv')
orders = pd.read_csv('../Daten/orders.csv')
products = pd.read_csv('../Daten/products.csv')
tips = pd.read_csv('../Daten/tips.csv')
```

```
In [5]: merged_data = (order_products
                   .merge(products, on='product_id')
                   .merge(departments, on='department_id')
                   .merge(orders, on='order_id')
                   .merge(tips, on='order_id'))

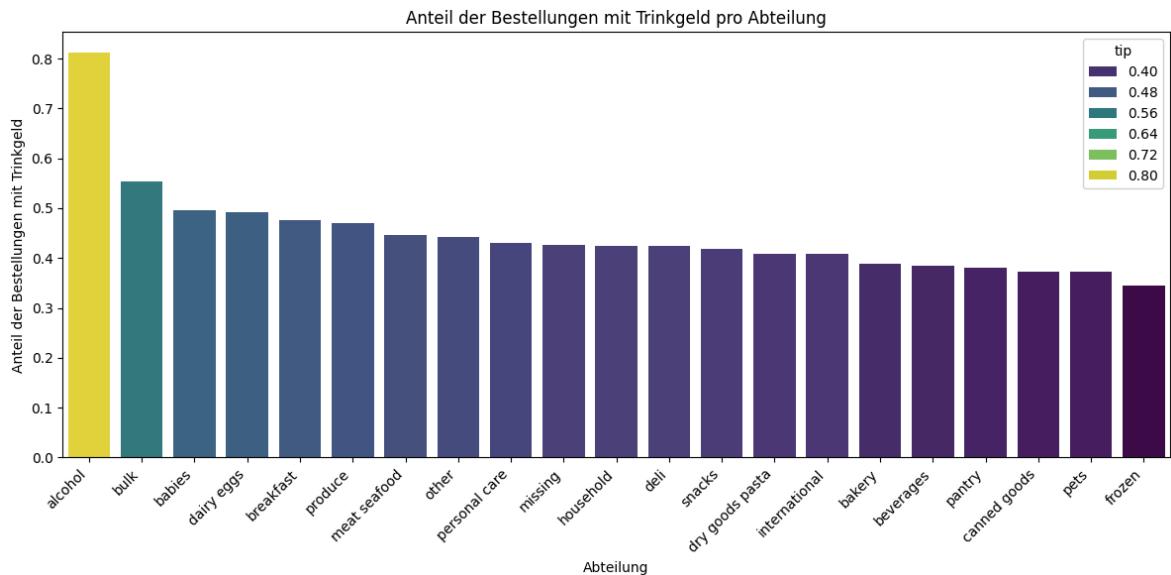
# Aggregate data: Percentage of orders with tips by department
department_tip_rate = (merged_data.groupby('department')['tip']
                        .mean() # Calculate the proportion of True (tippe
                        .reset_index()
                        .sort_values(by='tip', ascending=False))

# Visualize the percentage of tipped orders by department
plt.figure(figsize=(12, 6))
sns.barplot(data=department_tip_rate, x='department', y='tip', palette='viridis')
plt.xticks(rotation=45, ha='right')
plt.title('Anteil der Bestellungen mit Trinkgeld pro Abteilung')
plt.ylabel('Anteil der Bestellungen mit Trinkgeld')
plt.xlabel('Abteilung')
plt.tight_layout()
plt.show()

# Statistical analysis: Chi-square test to check association
# Create a contingency table: Count of tipped vs. non-tipped orders by department
contingency_table = pd.crosstab(merged_data['tip'], merged_data['department'])

# Perform Chi-square test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Output results
print("Chi-square Test Results")
print(f"Chi2 statistic: {chi2:.2f}")
print(f"P-value: {p:.4e}")
print(f"Degrees of Freedom: {dof}")
if p < 0.05:
    print("Schlussfolgerung: Es besteht ein signifikanter Zusammenhang zwis")
else:
    print("Schlussfolgerung: Es besteht kein signifikanter Zusammenhang zwis")
```



Chi-square Test Results

Chi2 statistic: 199364.85

P-value: 0.0000e+00

Degrees of Freedom: 20

Schlussfolgerung: Es besteht ein signifikanter Zusammenhang zwischen dem Trinkgeldverhalten und der Abteilung.

```
In [6]: department_order_count = merged_data.groupby('department')['order_id'].nunique()

# Prozentuale Verteilung berechnen
total_orders = merged_data['order_id'].nunique() # Gesamtzahl der Bestellungen
department_percentage = (department_order_count / total_orders) * 100

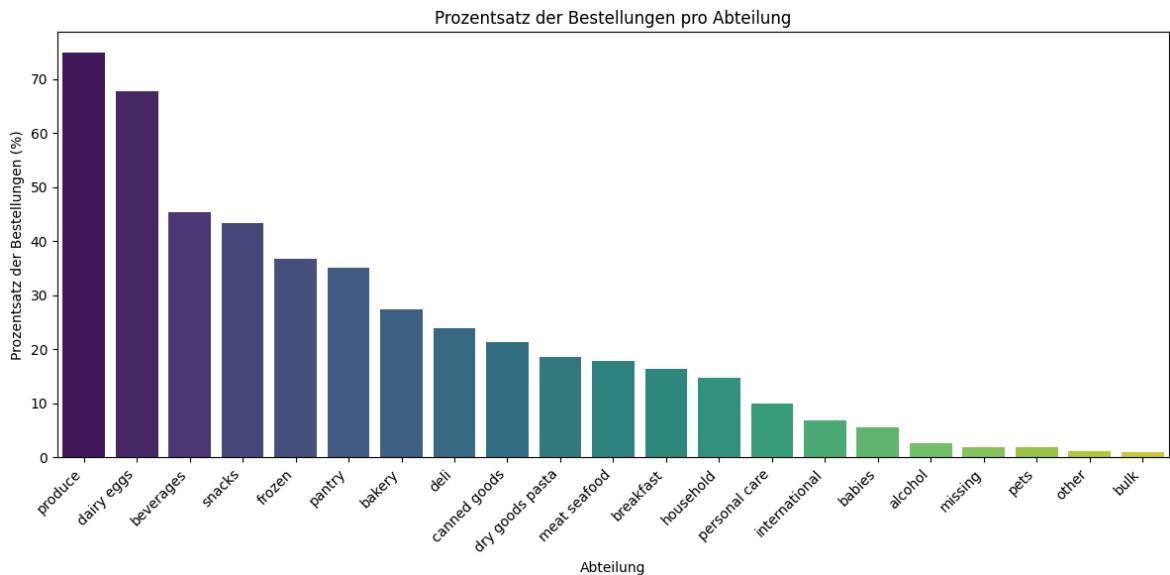
# Sortierung nach Prozentwerten
department_percentage = department_percentage.sort_values(ascending=False)
department_percentage.columns = ['department', 'percentage']

# Visualisierung der prozentualen Verteilung
plt.figure(figsize=(12, 6))
sns.barplot(data=department_percentage, x='department', y='percentage', palette='viridis')
plt.xticks(rotation=45, ha='right')
plt.title('Prozentsatz der Bestellungen pro Abteilung')
plt.ylabel('Prozentsatz der Bestellungen (%)')
plt.xlabel('Abteilung')
plt.tight_layout()
plt.show()
```

/tmp/ipykernel_843853/3008101777.py:13: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(data=department_percentage, x='department', y='percentage',
            palette='viridis')
```



```
In [7]: product_tips_orders = merged_data.groupby('product_name').agg(
    total_tips=('tip', 'sum'),
    total_orders=('order_id', 'count')
).reset_index()

# Filter products with at least 10 orders
product_tips_orders = product_tips_orders[product_tips_orders['total_orders'] >= 10]

# Calculate tip probability as tips/orders
product_tips_orders['tip_probability'] = product_tips_orders['total_tips'] / product_tips_orders['total_orders']

# Sort by tip probability
product_tips_orders = product_tips_orders.sort_values(by='tip_probability', ascending=False)

# Get TOP10 and FLOP10 products
top10_products = product_tips_orders.head(10)
flop10_products = product_tips_orders.tail(10)

plt.figure(figsize=(12, 6))
sns.barplot(
    data=top10_products,
    x='tip_probability',
    y='product_name',
    color="green"
)
plt.title('Top 10 Produkte mit der höchsten Trinkgeldwahrscheinlichkeit')
plt.xlabel('Tipping Probability')
plt.ylabel('Product')
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 6))
sns.barplot(
    data=flop10_products,
    x='tip_probability',
    y='product_name',
    color="red" # Set a single color
)
plt.title('Flop 10 Produkte mit der niedrigsten Trinkgeldwahrscheinlichkeit')
plt.xlabel('Tipping Probability')
plt.ylabel('Product')
```

```

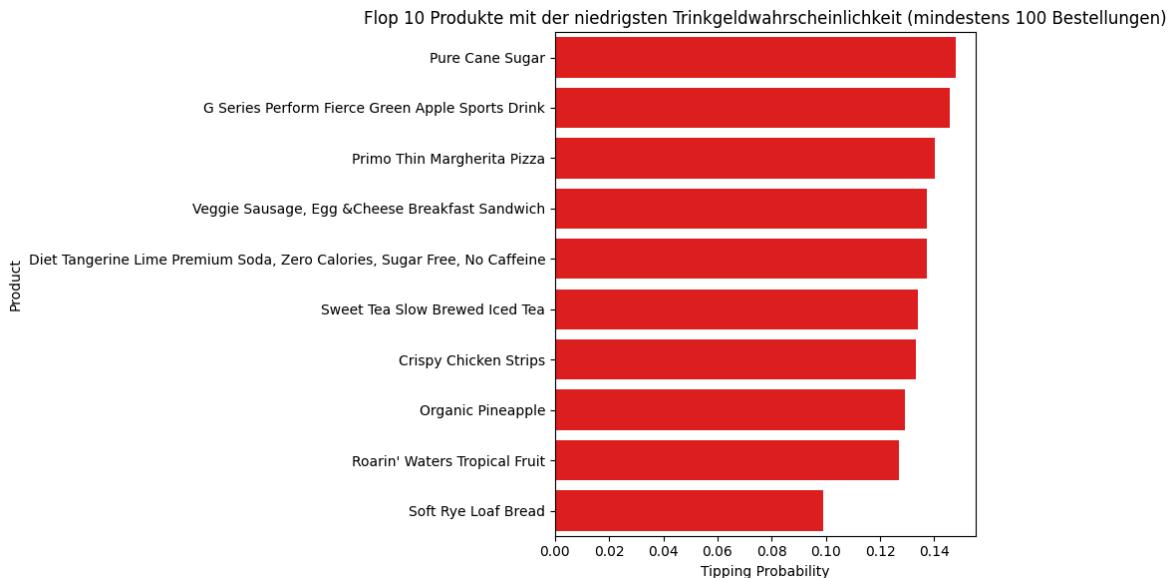
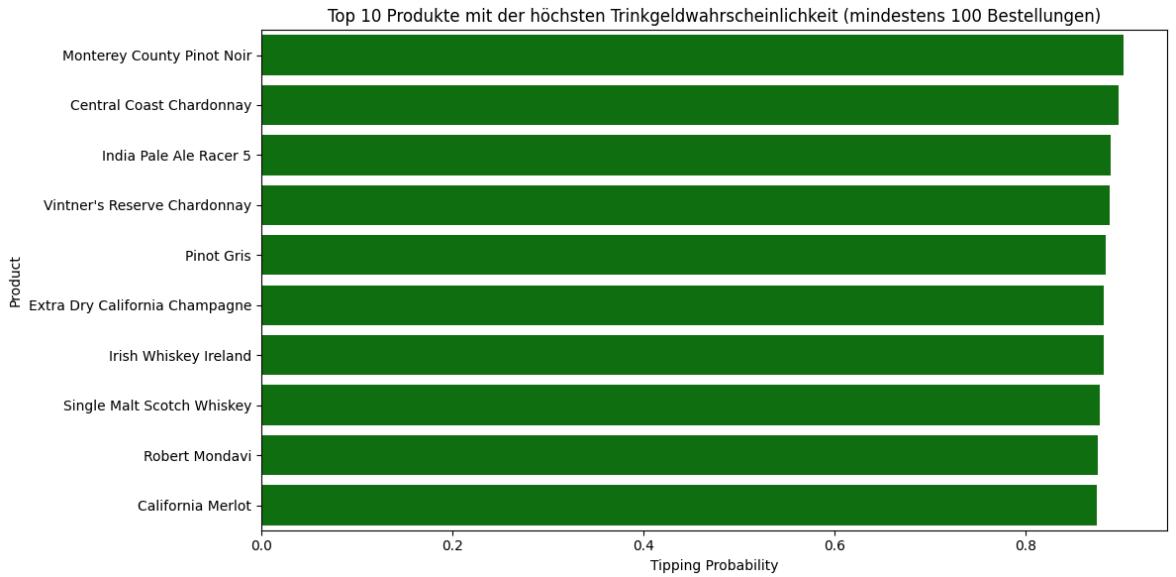
plt.tight_layout()
plt.show()

# Display the results
print("Top 10 Produkte mit der höchsten Trinkgeldwahrscheinlichkeit (mind")
print(top10_products)

print("\nFlop 10 Produkte mit der niedrigsten Trinkgeldwahrscheinlichkeit")
print(flop10_products)

print(product_tips_orders)

```



Top 10 Produkte mit der höchsten Trinkgeldwahrscheinlichkeit (mindestens 100 Bestellungen):

	product_name	total_tips	total_orders	\
25932	Monterey County Pinot Noir	139	154	
6912	Central Coast Chardonnay	156	174	
20771	India Pale Ale Racer 5	199	224	
47284	Vintner's Reserve Chardonnay	261	294	
34664	Pinot Gris	121	137	
14105	Extra Dry California Champagne	127	144	
21015	Irish Whiskey Ireland	141	160	
40146	Single Malt Scotch Whiskey	143	163	
38222	Robert Mondavi	112	128	
6319	California Merlot	201	230	

	tip_probability
25932	0.902597
6912	0.896552
20771	0.888393
47284	0.887755
34664	0.883212
14105	0.881944
21015	0.881250
40146	0.877301
38222	0.875000
6319	0.873913

Flop 10 Produkte mit der niedrigsten Trinkgeldwahrscheinlichkeit (mindestens 100 Bestellungen):

	product_name	total_tips	\
36277	Pure Cane Sugar	21	
16369	G Series Perform Fierce Green Apple Sports Drink	15	
35813	Primo Thin Margherita Pizza	24	
47117	Veggie Sausage, Egg &Cheese Breakfast Sandwich	18	
12695	Diet Tangerine Lime Premium Soda, Zero Calorie...	17	
43677	Sweet Tea Slow Brewed Iced Tea	15	
11267	Crispy Chicken Strips	18	
30876	Organic Pineapple	20	
38026	Roarin' Waters Tropical Fruit	16	
41038	Soft Rye Loaf Bread	11	

	total_orders	tip_probability
36277	142	0.147887
16369	103	0.145631
35813	171	0.140351
47117	131	0.137405
12695	124	0.137097
43677	112	0.133929
11267	135	0.133333
30876	155	0.129032
38026	126	0.126984
41038	111	0.099099

	product_name	total_tips	total_orders	\
25932	Monterey County Pinot Noir	139	154	
6912	Central Coast Chardonnay	156	174	
20771	India Pale Ale Racer 5	199	224	
47284	Vintner's Reserve Chardonnay	261	294	
34664	Pinot Gris	121	137	
...	
43677	Sweet Tea Slow Brewed Iced Tea	15	112	
11267	Crispy Chicken Strips	18	135	

30876	Organic Pineapple	20	155
38026	Roarin' Waters Tropical Fruit	16	126
41038	Soft Rye Loaf Bread	11	111
	tip_probability		
25932	0.902597		
6912	0.896552		
20771	0.888393		
47284	0.887755		
34664	0.883212		
...	...		
43677	0.133929		
11267	0.133333		
30876	0.129032		
38026	0.126984		
41038	0.099099		

[14750 rows x 4 columns]

```
In [8]: from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss, classification_report
import numpy as np

# Prepare data
# One-hot encode departments and top/bottom products
merged_data['is_top_product'] = merged_data['product_name'].isin(top10_products)
merged_data['is_flop_product'] = merged_data['product_name'].isin(flop10_products)

# One-hot encode department
encoder = OneHotEncoder(drop='first', sparse_output=False)
departments_encoded = encoder.fit_transform(merged_data[['department']])
department_columns = encoder.get_feature_names_out(['department'])

# Create feature set
X = pd.DataFrame(departments_encoded, columns=department_columns)
X['is_top_product'] = merged_data['is_top_product']
X['is_flop_product'] = merged_data['is_flop_product']

# Target variable
y = merged_data['tip']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Model 1: Using department only
model_1 = LogisticRegression(max_iter=1000)
model_1.fit(X_train[department_columns], y_train)

# Model 2: Adding specific products
model_2 = LogisticRegression(max_iter=1000)
model_2.fit(X_train, y_train)

# Evaluate models
log_loss_1 = log_loss(y_test, model_1.predict_proba(X_test[department_columns]))
log_loss_2 = log_loss(y_test, model_2.predict_proba(X_test))

print(f"Log Loss (Model 1 - Department Only): {log_loss_1:.7f}")
print(f"Log Loss (Model 2 - Department + Products): {log_loss_2:.7f}")
```

```

# Compare models
if log_loss_2 < log_loss_1:
    print("Adding specific products improves the model, indicating they have an additional effect on tipping probability.")
elif log_loss_1 == log_loss_2:
    print("Same result")
else:
    print("Spezi")

# Optional: Examine coefficients for interpretation
coefficients = pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': model_2.coef_[0]
}).sort_values(by='Coefficient', ascending=False)

print(coefficients)

```

Log Loss (Model 1 - Department Only): 0.6795520
 Log Loss (Model 2 - Department + Products): 0.6795432
 Adding specific products improves the model, indicating they have an additional effect on tipping probability.

	Feature	Coefficient
20	is_top_product	0.855154
21	is_flop_product	-0.420120
4	department_bulk	-1.155796
0	department_babies	-1.431471
6	department_dairy eggs	-1.453442
3	department_breakfast	-1.518219
18	department_produce	-1.535526
12	department_meat seafood	-1.628444
16	department_personal care	-1.663833
10	department_household	-1.710618
7	department_deli	-1.728808
13	department_missing	-1.732152
19	department_snacks	-1.744570
14	department_other	-1.751906
8	department_dry goods pasta	-1.781321
11	department_international	-1.799045
1	department_bakery	-1.873020
2	department_beverages	-1.892069
15	department_pantry	-1.907465
5	department_canned goods	-1.934002
17	department_pets	-1.968734
9	department_frozen	-2.057811

Da der Unterschied im Log Loss zwischen Modell 1 (nur Abteilungen) und Modell 2 (Abteilungen + Produkte) minimal ist, lässt sich schlussfolgern, dass die spezifischen Produkte keinen signifikanten zusätzlichen Einfluss auf die Trinkgeldwahrscheinlichkeit haben. Die geringe Verbesserung deutet darauf hin, dass die Abteilung als alleiniger Faktor bereits ausreichend für die Vorhersage der Trinkgeldwahrscheinlichkeit ist.

Frage 3:

Gibt es einen Zusammenhang zwischen dem Trinkgeldverhalten und der Tageszeit, dem Wochentag, oder dem Zeitabstand zur vorhergehenden Bestellung?

Tageszeit & Trinkgeldverhalten

```
In [9]: # Query: Anzahl der Bestellungen pro Wochentag
result_dow = session.query(
    Order.day_of_the_week,
    func.count().label('total_orders'), # Gesamtzahl der Bestellungen
    func.sum(func.cast(Order.tips, Integer)).label('total_tips') # Anzahl der Trinkgeldzahler
).group_by(Order.day_of_the_week).all()

result_dow_tips = session.query(
    Order.day_of_the_week,
    func.sum(case((Order.tips == True, 1), else_=0)).label('tips_given'),
    func.sum(case((Order.tips == False, 1), else_=0)).label('no_tips')
).group_by(Order.day_of_the_week).all()
```

⌚ Zeitliche Analyse des Trinkgeldverhaltens



Untersuchung zeitlicher Muster

In diesem Abschnitt untersuchen wir die zeitlichen Aspekte des Trinkgeldverhaltens, beginnend mit der Analyse der Wochentage. Ziel ist es, mögliche Muster und Zusammenhänge zwischen dem Zeitpunkt der Bestellung und der Trinkgeldvergabe zu identifizieren.



Visualisierungsaufbau:

- **Linker Plot:** Absolute Verteilung der Bestellungen nach Wochentagen
- **Rechter Plot:** Trinkgeldquote für jeden Wochentag



Kernfragen:

- Gibt es Wochentage mit besonders hoher/niedriger Bestellaktivität?
- Unterscheidet sich die Trinkgeldbereitschaft an verschiedenen Wochentagen?
- Lassen sich wiederkehrende Muster im Wochenverlauf erkennen?

```
In [10]: # plot code
days = [row.day_of_the_week for row in result_dow]
total_orders = [row.total_orders for row in result_dow]
total_tips = [row.total_tips if row.total_tips else 0 for row in result_dow]
tip_ratios = [row.total_tips / row.total_orders if row.total_orders > 0 else 0 for row in result_dow]

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(18, 6))
plt.subplots_adjust(left=0.08)

# Daten vorbereiten
width = 0.4
x = range(len(days))

# Diagramm 1: Gesamtbestellungen und Bestellungen mit Trinkgeld
```

```

bars1 = ax1.bar(x, total_orders, width, label='Bestellungen insgesamt', color='blue')
bars2 = ax1.bar([i + width for i in x], total_tips, width, label='Bestellungen mit Trinkgeld',
               color='lightgreen')

# Formatierung Diagramm 1
ax1.set_xlabel('Wochentag')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.set_title('Bestellungen pro Wochentag mit und ohne Trinkgeld',
              loc='left', weight='bold')
ax1.set_xticks([i + width / 2 for i in x])
ax1.set_xticklabels(days, rotation=0)
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)
ax1.legend()

# Werte über den Balken für Diagramm 1
for bars in [bars1, bars2]:
    for bar in bars:
        height = bar.get_height()
        ax1.text(bar.get_x() + bar.get_width()/2, height,
                 f'{int(height)}', ha='center', va='bottom', fontsize=10)

# Diagramm 2: Tip-Quote
bars3 = ax2.bar(days, tip_ratios, color='purple', alpha=0.7)

# Formatierung Diagramm 2
ax2.set_xlabel('Wochentag')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_title('Trinkgeldquote pro Wochentag',
              loc='left', weight='bold')
ax2.set_ylim(0, 1)
ax2.set_yticks([i / 10 for i in range(0, 11)])
ax2.set_yticklabels([f'{i * 10}%' for i in range(0, 11)])
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)

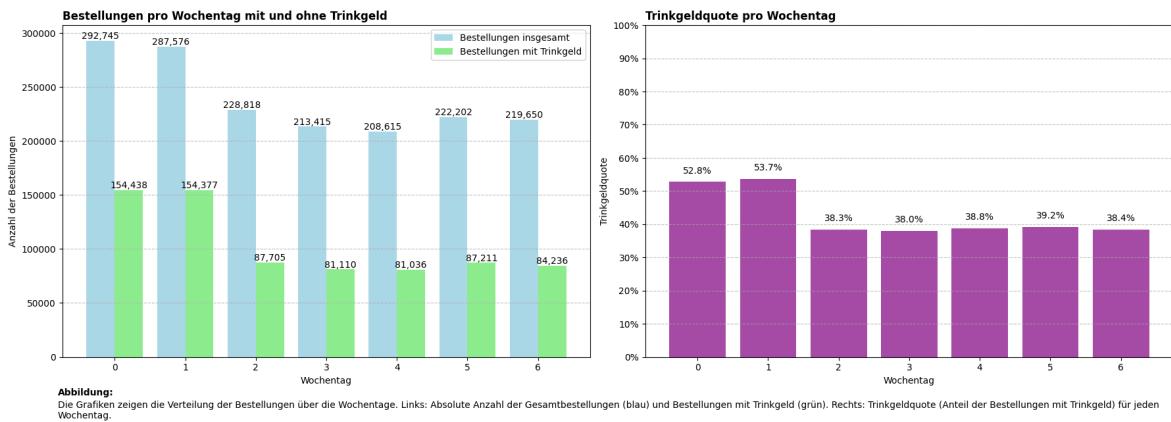
# Werte über den Balken für Diagramm 2
for bar in bars3:
    height = bar.get_height()
    ax2.text(bar.get_x() + bar.get_width()/2, height + 0.02,
             f'{height:.1%}', ha='center', va='bottom', fontsize=10)

# Bildunterschrift
description = ('Die Grafiken zeigen die Verteilung der Bestellungen über\n'
               'Links: Absolute Anzahl der Gesamtbestellungen (blau) und B\n'
               'Rechts: Trinkgeldquote (Anteil der Bestellungen mit Trinkg')

fig.text(0.05, 0, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, -0.055, description, wrap=True)

plt.tight_layout()
plt.show()

```



📈 Interpretation der Wochentagsanalyse

🔍 Zentrale Beobachtungen

📊 Bestellvolumen:

- Deutlich erhöhtes Bestellaufkommen an den Tagen 0 und 1
- Diese Tage heben sich signifikant vom Rest der Woche ab
- Konzentrierung der Bestellaktivität am Wochenanfang

💰 Trinkgeldverhalten:

- Parallel zum Bestellvolumen erhöhte Trinkgeldquote an Tagen 0 und 1
- Positive Korrelation zwischen Bestellaufkommen und Trinkgeldbereitschaft
- Rest der Woche zeigt niedrigere, aber stabile Trinkgeldquoten

💡 Schlussfolgerungen:

- Die ersten beiden Wochentage sind sowohl für das Bestellvolumen als auch für Trinkgelder besonders relevant
- Mögliche Einflüsse könnten Wocheneinkäufe oder Arbeitsrhythmen sein
- Potenzial als Feature in einem Logistischen Regressionsmodell

```
In [11]: data = [
    {'day_of_the_week': row[0], 'tips_given': row[1], 'no_tips': row[2]}
    for row in result_dow_tips
]
df = pd.DataFrame(data)

# Kontingenztabelle erstellen
contingency_table = df.set_index('day_of_the_week')[['tips_given', 'no_tips']]
```

```
In [12]: # Chi-Quadrat-Test
chi2, p, dof, expected = chi2_contingency(contingency_table)
```

```

# Tabellenausgabe: Vergleich von beobachteten und erwarteten Werten
observed = contingency_table.values # Beobachtete Werte
expected = expected # Erwartete Werte

# Vergleichstabelle erstellen
comparison_data = []
for i, day in enumerate(contingency_table.index):
    comparison_data.append([
        day, # Wochentag
        observed[i][0], # Beobachtet: Trinkgeld gegeben
        expected[i][0], # Erwartet: Trinkgeld gegeben
        observed[i][1], # Beobachtet: Kein Trinkgeld
        expected[i][1], # Erwartet: Kein Trinkgeld
    ])

# Tabellenkopf
headers = ["Wochentag", "Beobachtet (Tip gegeben)", "Erwartet (Tip gegeben)", "Beobachtet (kein Tip)", "Erwartet (kein Tip)"]

# Tabelle formatieren
print("\nVergleich der beobachteten und erwarteten Werte:")
table_markdown = tabulate(comparison_data, headers=headers, tablefmt="pipe")
display(Markdown(table_markdown))

# Ergebnisse des Chi2-Tests visualisieren
print("\nErgebnisse des Chi2-Tests:")
test_results = [
    ["Chi-Quadrat-Statistik", f"{chi2:.3f}"],
    ["p-Wert", f"{p}"],
    ["Freiheitsgrade", dof],
    ["Signifikanz", "Ja" if p < 0.05 else "Nein"]
]

# Tabellarische Ausgabe der Testergebnisse
table_results = tabulate(test_results, headers=["Metrik", "Wert"], tablefmt="pipe")
display(Markdown(table_results))

```

Vergleich der beobachteten und erwarteten Werte:

Wochentag	Beobachtet (Tip gegeben)	Erwartet (Tip gegeben)	Beobachtet (kein Tip)	Erwartet (kein Tip)
0	154438	127755	138307	164990
1	154377	125499	133199	162077
2	87705	99857.1	141113	128961
3	81110	93135.2	132305	120280
4	81036	91040.4	127579	117575
5	87211	96969.8	134991	125232
6	84236	95856.1	135414	123794

Ergebnisse des Chi²-Tests:

Metrik	Wert
Chi-Quadrat-Statistik	33249.758
p-Wert	0.0
Freiheitsgrade	6
Signifikanz	Ja

```
In [13]: # Daten vorbereiten
observed = contingency_table.values # Beobachtete Werte
expected = expected # Erwartete Werte
days = contingency_table.index # Wochentage

# Abweichungen berechnen (relativ oder absolut)
deviation = (observed - expected) / expected # Relative Abweichung (Proz)
# deviation = observed - expected # Absolute Abweichung

# DataFrame für Heatmap erstellen
heatmap_data = pd.DataFrame(deviation, columns=["Trinkgeld gegeben", "Kein Trinkgeld"], index=days)

# Heatmap zeichnen
fig, ax = plt.subplots(figsize=(10, 8))
plt.subplots_adjust(left=0.08)

# Abweichungen berechnen
deviation = (observed - expected) / expected

# DataFrame für Heatmap erstellen
heatmap_data = pd.DataFrame(
    deviation,
    columns=["Trinkgeld gegeben", "Kein Trinkgeld"],
    index=days
)

# Maximalen absoluten Wert finden für symmetrische Farbskala
max_abs_value = np.max(np.abs(deviation))
vmin = -max_abs_value
vmax = max_abs_value

# Heatmap erstellen
sns.heatmap(heatmap_data,
            annot=True,
            cmap="RdBu",
            center=0,
            fmt=".2%", # Prozentformat
            cbar_kws={'label': 'Relative Abweichung'},
            vmin=vmin,
            vmax=vmax,
            ax=ax)

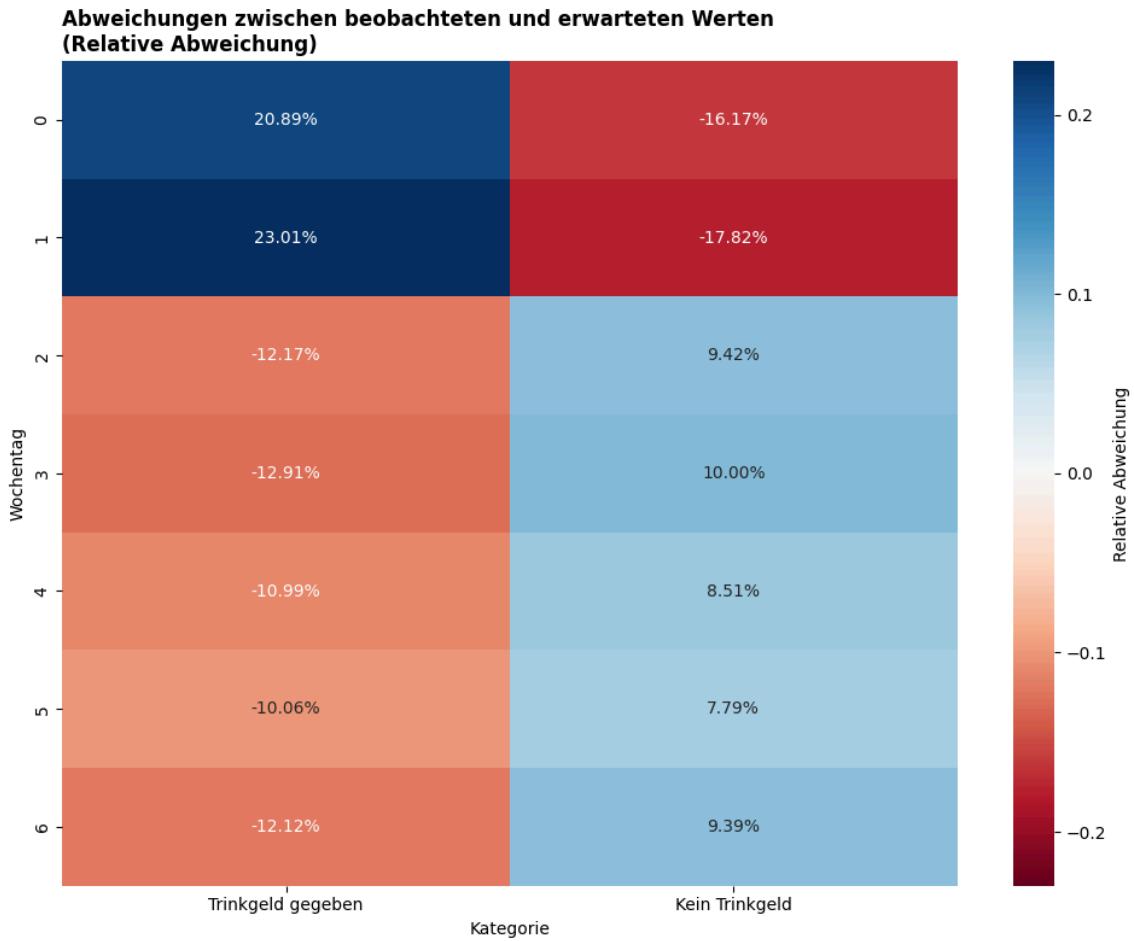
# Titel und Achsenbeschriftungen
ax.set_title('Abweichungen zwischen beobachteten und erwarteten Werten\n'
             'loc="left",\n             weight="bold")')
ax.set_xlabel('Kategorie')
ax.set_ylabel('Wochentag')

# Bildunterschrift
description = ('Die Heatmap zeigt die relativen Abweichungen zwischen beo
```

```
'Positive Werte (rot) zeigen mehr Fälle als erwartet, negative Werte (blau) weniger als erwartet. Die Werte in den Zellen geben die prozentuale Abweichung an'

fig.text(0.05, 0, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, -0.06, description, wrap=True)

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



📊 Statistische Bestätigung der Wochentags-Abhängigkeit

🔍 Chi-Quadrat-Test Ergebnisse

Der Chi-Quadrat-Unabhängigkeitstest bestätigt einen statistisch signifikanten Zusammenhang zwischen Wochentagen und Trinkgeldverhalten. Die Heatmap visualisiert die Abweichungen von den erwarteten Werten.

📈 Kernerkenntnisse:

- Trinkgeldvergabe und Wochentag sind nicht unabhängig voneinander
- Besonders deutliche Abweichungen an den Tagen 0 und 1
- Die Muster bestätigen die vorherige deskriptive Analyse

Bedeutung:

- Der Wochentag hat einen nachweisbaren Einfluss auf das Trinkgeldverhalten
- Die beobachteten Unterschiede sind nicht zufällig
- Diese Erkenntnis ist relevant für zeitbasierte Geschäftsstrategien

```
In [14]: result = session.query(
    Order.day_of_the_week,
    func.cast(Order.tips, Integer).label('tips_given') # Boolean auf Integer
).all()

df = pd.DataFrame(result, columns=['day_of_week', 'tips_given'])
```

Maschinelles Lernen: Vorhersagbarkeit von Trinkgeldzahlungen

Explorativer Machine Learning Ansatz

Um die Bedeutung des Wochentags für das Trinkgeldverhalten weiter zu untersuchen, erweitern wir unsere Analyse um einen maschinellen Lernansatz. Dabei versuchen wir, allein basierend auf dem Wochentag eine Vorhersage über die Wahrscheinlichkeit einer Trinkgeldzahlung zu treffen.

Methodischer Ansatz:

- Verwendung einer logistischen Regression als Klassifikationsmodell
- Wochentage als einziges Feature (One-Hot-Encoded)
- Evaluation mittels ROC-Kurve und AUC-Score

Einschränkungen:

- Bewusst simplifizierende Modellierung
- Fokus auf einzelnen Zeitaspekt
- Exploratives Werkzeug zur Mustererkennung

Interpretationshinweis:

Die folgende ROC-Analyse soll primär die Stärke des Zusammenhangs zwischen Wochentag und Trinkgeldverhalten quantifizieren, nicht als vollständiges Vorhersagemodell dienen.

```
In [16]: # logistische Regression
# One-Hot-Encoding für 'day_of_week' (von numerischen Werten zu binären S
df_encoded = pd.get_dummies(df, columns=['day_of_week'], prefix='day')

# Feature und Zielvariable
```

```

X = df_encoded.drop('tips_given', axis=1)
y = df_encoded['tips_given']

X_train_day, X_test_day, y_train_day, y_test_day = train_test_split(X, y)

model_day = LogisticRegression(max_iter=1000) # max_iter erhöhen, wenn K

model_day.fit(X_train_day, y_train_day)

y_pred = model_day.predict(X_test_day)

accuracy = accuracy_score(y_test_day, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

coefficients = model_day.coef_.flatten()
intercept = model_day.intercept_[0]

days_of_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
coeff_df = pd.DataFrame({
    'Day of Week': days_of_week,
    'Coefficient': coefficients
})

print("Intercept:", intercept)
display(coeff_df)

```

Accuracy: 58.62%

Intercept: -0.25889614670551675

	Day of Week	Coefficient
0	Monday	0.366213
1	Tuesday	0.408174
2	Wednesday	-0.217128
3	Thursday	-0.232649
4	Friday	-0.194186
5	Saturday	-0.176958
6	Sunday	-0.212348

```

In [17]: y_prob_day = model_day.predict_proba(X_test_day)[:, 1] # Nur die Wahrscheinlichkeit der Klassifikation

# Berechnung der ROC-Kurve
fpr, tpr, thresholds = roc_curve(y_test_day, y_prob_day)

# Berechnung der AUC (Area Under the Curve)
roc_auc = auc(fpr, tpr)

# Plot der ROC-Kurve
fig, ax = plt.subplots(figsize=(8, 6))
plt.subplots_adjust(left=0.08)

# ROC-Kurve zeichnen
ax.plot(fpr, tpr, color='darkorange', lw=2,
        label=f'ROC-Kurve (AUC = {roc_auc:.2f})')

# Diagonale Linie (Zufallsklassifikator)
ax.plot([0, 1], [0, 1], color='navy', lw=2)

```

```
    linestyle='--', label='Zufallsklassifikator')

# Achsen formatieren
ax.set_xlim([0.0, 1.0])
ax.set_ylim([0.0, 1.05])
ax.set_xlabel('False Positive Rate')
ax.set_ylabel('True Positive Rate')
ax.set_title('ROC-Kurve für die Vorhersage von Trinkgeldzahlungen\nbasierend auf dem Zufallsklassifikator', loc='left', weight='bold')

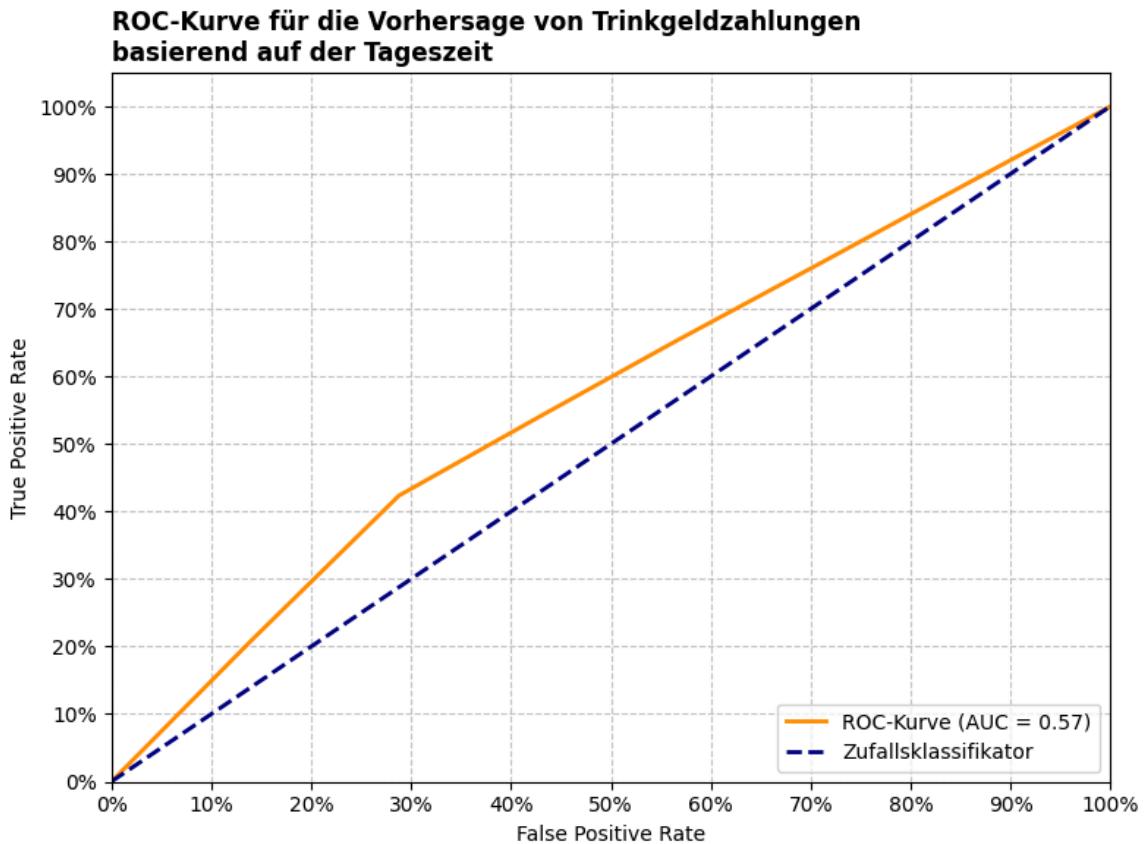
# Grid hinzufügen
ax.grid(True, linestyle='--', alpha=0.7)

# Legende
ax.legend(loc='lower right')

# Achsenbeschriftungen in Prozent
ax.set_xticks([i/10 for i in range(0, 11)])
ax.set_yticks([i/10 for i in range(0, 11)])
ax.set_xticklabels([f"{i*10}%" for i in range(0, 11)])
ax.set_yticklabels([f"{i*10}%" for i in range(0, 11)])

# Bildunterschrift
description = ('Die ROC-Kurve (Receiver Operating Characteristic) zeigt die Trennschärfen der verschiedenen Schwellenwerten. Die AUC (Area Under Curve) von {:.2f} quantifiziert die Güte des Modells. Ein Wert von 0.5 (gestrichelte Linie) entspricht einer zufälligen Klassifikation, während 1.0 eine perfekte Klassifikation bedeutet.'.format(auc))
fig.text(0.09, 0, 'Abbildung:', weight='bold', ha='left')
fig.text(0.09, -0.11, description, wrap=True)

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

**Abbildung:**

Die ROC-Kurve (Receiver Operating Characteristic) zeigt die Performance des Modells bei verschiedenen Schwellenwerten. Die AUC (Area Under Curve) von 0.57 quantifiziert die Gesamtleistung des Modells. Ein Wert von 0.5 (gestrichelte Linie) entspricht einem Zufallsklassifikator, während 1.0 eine perfekte Klassifikation bedeutet.



Interpretation der ROC-Analyse



Einordnung des AUC-Werts von 0.57

Der ermittelte AUC-Wert von 0.57 liegt nur leicht über dem Wert eines Zufallsklassifikators (0.5), was zu mehreren wichtigen Erkenntnissen führt.



Bedeutung des Ergebnisses:

- Das Modell performt nur minimal besser als zufälliges Raten
- Der Wochentag allein ist ein schwacher Prädiktor für Trinkgeldzahlungen
- Die statistisch nachgewiesene Abhängigkeit ist in der Praxis wenig aussagekräftig



Praktische Implikationen:

- Trinkgeldverhalten wird von vielen weiteren Faktoren beeinflusst
- Der Wochentag sollte nicht überbewertet werden
- Für effektive Vorhersagen werden weitere Features benötigt

⌚ Fazit:

Obwohl ein statistischer Zusammenhang zwischen Wochentag und Trinkgeldverhalten existiert, ist dieser zu schwach, um als verlässlicher Prädiktor zu dienen. Dies unterstreicht die Komplexität des Trinkgeldverhaltens und die Notwendigkeit, multiple Faktoren zu berücksichtigen.

Tageszeit & Trinkgeldverhalten

```
In [18]: result_hours = session.query(
    Order.hour_of_day,
    func.count().label('total_orders'), # Gesamtzahl der Bestellungen
    func.sum(func.cast(Order.tips, Integer)).label('total_tips') # Anzahl
).group_by(Order.hour_of_day).all()

result_hours_tips = session.query(
    Order.hour_of_day,
    func.sum(case((Order.tips == True, 1), else_=0)).label('tips_given'),
    func.sum(case((Order.tips == False, 1), else_=0)).label('no_tips')
).group_by(Order.hour_of_day).all()
```

⌚ Analyse des Trinkgeldverhaltens im Tagesverlauf

🔍 Stundenbasierte Untersuchung

Nach der Analyse der Wochentage betrachten wir nun die Verteilung der Bestellungen und des Trinkgeldverhaltens im Verlauf eines Tages. Diese feingranularere zeitliche Analyse könnte wichtige Einblicke in das Kundenverhalten zu verschiedenen Tageszeiten liefern.

📊 Visualisierungsaufbau:

- **Linker Plot:** Absolute Verteilung der Bestellungen nach Stunden
- **Rechter Plot:** Trinkgeldquote für jede Stunde des Tages
- 24-Stunden-Format für präzise Zeitdarstellung

⌚ Kernfragen:

- Gibt es Stoßzeiten für Bestellungen?
- Variiert die Trinkgeldbereitschaft zu verschiedenen Tageszeiten?
- Lassen sich typische Muster (z.B. Mittags-/Abendspitzen) erkennen?
- Gibt es Zusammenhänge zwischen Bestellvolumen und Trinkgeldquote?

```
In [19]: result_hours_data = [
    [row.hour_of_day, row.total_orders, row.total_tips, f"{row.total_tips}"
     for row in result_hours
]
headers_hours = ["Stunde", "Bestellungen gesamt", "Mit Trinkgeld", "Trinkgeldquoten"]
```

```
print("\nBestellungen pro Stunde:")
result_hours_markdown = tabulate(result_hours_data, headers=headers_hours
display(Markdown(result_hours_markdown)))
```

Bestellungen pro Stunde:

Stunde	Bestellungen gesamt	Mit Trinkgeld	Trinkgeldquote (%)
0	11224	6003	53.48%
1	6211	3244	52.23%
2	3773	1931	51.18%
3	2762	1397	50.58%
4	2748	1406	51.16%
5	4764	2005	42.09%
6	15004	6462	43.07%
7	45141	19897	44.08%
8	87513	39013	44.58%
9	125904	56300	44.72%
10	141401	61461	43.47%
11	139520	58575	41.98%
12	133612	55220	41.33%
13	136575	56133	41.10%
14	138816	57426	41.37%
15	138279	56499	40.86%
16	133434	54224	40.64%
17	110832	44257	39.93%
18	88464	35118	39.70%
19	67899	36331	53.51%
20	50955	27821	54.60%
21	38382	21782	56.75%
22	30195	16934	56.08%
23	19613	10674	54.42%

```
In [20]: # Daten für das erste Diagramm
hours = [row.hour_of_day for row in result_hours]
total_orders = [row.total_orders for row in result_hours]
total_tips = [row.total_tips if row.total_tips else 0 for row in result_hours]

# Berechnung der Tip-Quote für das zweite Diagramm
tip_ratios = [row.total_tips / row.total_orders if row.total_orders > 0 else 0 for row in result_hours]

# Daten vorbereiten
width = 0.4
```

```

x = range(len(hours))

# Diagramm 1: Gesamtbestellungen und Bestellungen mit Trinkgeld
bars1 = ax1.bar(x, total_orders, width, label='Bestellungen insgesamt', color='blue')
bars2 = ax1.bar([i + width for i in x], total_tips, width, label='Bestellungen mit Trinkgeld', color='lightgreen')

# Formatierung Diagramm 1
ax1.set_xlabel('Stunde des Tages')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.set_title('Bestellungen pro Stunde mit und ohne Trinkgeld', loc='left', weight='bold')
ax1.set_xticks([i + width / 2 for i in x])
ax1.set_xticklabels([f"{h:02d}" for h in hours])
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)
ax1.legend()

# Diagramm 2: Trinkgeldquote
bars3 = ax2.bar(hours, tip_ratios, color='purple', alpha=0.7)

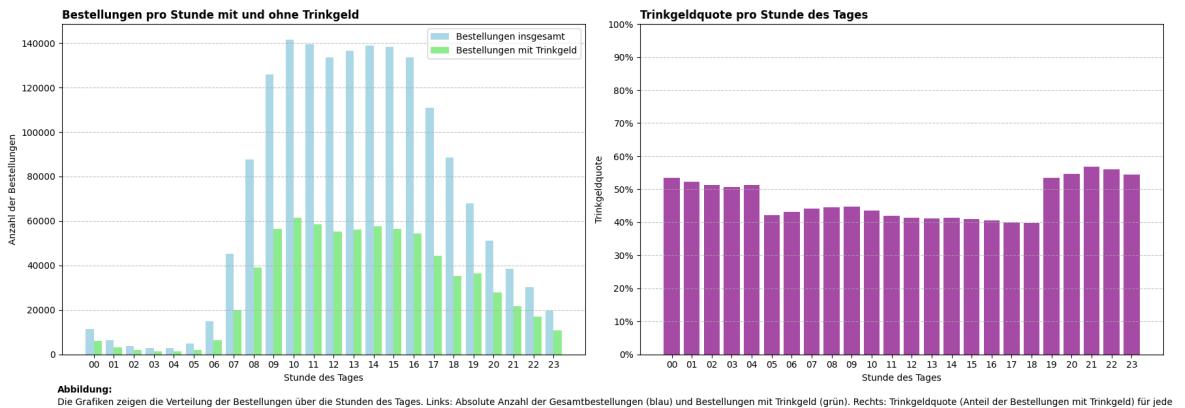
# Formatierung Diagramm 2
ax2.set_xlabel('Stunde des Tages')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_title('Trinkgeldquote pro Stunde des Tages', loc='left', weight='bold')
ax2.set_ylim(0, 1)
ax2.set_yticks([i / 10 for i in range(0, 11)])
ax2.set_yticklabels([f"{i * 10}%" for i in range(0, 11)])
ax2.set_xticks(hours)
ax2.set_xticklabels([f"{h:02d}" for h in hours])
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)

# Bildunterschrift
description = ('Die Grafiken zeigen die Verteilung der Bestellungen über\n'
               'Links: Absolute Anzahl der Gesamtbestellungen (blau) und Bestellungen mit Trinkgeld (grün)\n'
               'Rechts: Trinkgeldquote (Anteil der Bestellungen mit Trinkgeld an den gesamten Bestellungen)')

fig.text(0.05, 0, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, -0.055, description, wrap=True)

plt.tight_layout()
plt.show()

```



Interpretation der Tageszeit-Analyse



Interessanter Kontrast: Bestellvolumen vs. Trinkgeldquote



Bestellvolumen:

- Deutliche Konzentration der Bestellungen während der Tagesstunden
- Typisches Muster entsprechend der üblichen Aktivitätszeiten
- Erwartungsgemäß geringeres Bestellaufkommen in den Nachtstunden



Trinkgeldverhalten:

- Überraschend höhere Trinkgeldquote während der Nachtstunden
- Inverse Korrelation zum Bestellvolumen
- Deutlicher Kontrast zwischen Tag- und Nachtbestellungen



Mögliche Erklärungsansätze:

- Nachts bestellende Kunden könnten den Lieferservice besonders wertschätzen
- Möglicherweise höhere Zahlungsbereitschaft in den Nachtstunden
- Eventuell unterschiedliche Kundengruppen zu verschiedenen Tageszeiten
- Bewusstsein für erschwerte Arbeitsbedingungen der Lieferanten in der Nacht



Vergleichende Analyse der Zeitfaktoren mittels Machine Learning



Gegenüberstellung: Tageszeit vs. Wochentag als Prädiktoren

Um die relative Bedeutung von Tageszeit und Wochentag für das Trinkgeldverhalten zu quantifizieren, vergleichen wir die Vorhersagekraft beider Zeitfaktoren mittels logistischer Regression.



Methodischer Ansatz:

- Zwei separate logistische Regressionsmodelle:
 - Modell 1: Wochentag als einzelner Prädiktor
 - Modell 2: Tageszeit als einzelner Prädiktor
- Vergleich der Modellgüte mittels ROC-Kurven
- Direkte Gegenüberstellung der AUC-Werte



Ziel der Analyse:

- Identifikation des stärkeren Zeitprädiktors

- Quantifizierung der Vorhersagekraft beider Zeitfaktoren
- Bewertung der praktischen Relevanz für Trinkgeldvorhersagen

Interpretationshinweis:

Die folgende Visualisierung ermöglicht einen direkten Vergleich der Vorhersagekraft beider Zeitfaktoren, wobei weiterhin die explorative Natur dieser vereinfachten Modelle zu beachten ist.

```
In [21]: data = [
    {'hour_of_day': row[0], 'tips_given': row[1], 'no_tips': row[2]}
    for row in result_hours_tips
]
df = pd.DataFrame(data)

# Kontingenztabelle erstellen
contingency_table = df.set_index('hour_of_day')[['tips_given', 'no_tips']]
```

```
In [22]: # Chi-Quadrat-Test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Tabellenausgabe: Vergleich von beobachteten und erwarteten Werten
observed = contingency_table.values # Beobachtete Werte
expected = expected # Erwartete Werte

# Vergleichstabelle erstellen
comparison_data = []
for i, day in enumerate(contingency_table.index):
    comparison_data.append([
        day, # Wochentag
        observed[i][0], # Beobachtet: Trinkgeld gegeben
        expected[i][0], # Erwartet: Trinkgeld gegeben
        observed[i][1], # Beobachtet: Kein Trinkgeld
        expected[i][1] # Erwartet: Kein Trinkgeld
    ])

# Tabellenkopf
headers = ["Wochentag", "Beobachtet (Tip gegeben)", "Erwartet (Tip gegeben)", "Beobachtet (kein Tip)", "Erwartet (kein Tip)"]

# Tabelle formatieren
print("\nVergleich der beobachteten und erwarteten Werte:")
comparison_data_markdown = tabulate(comparison_data, headers=headers, tab=display(Markdown(comparison_data_markdown)))

# Ergebnisse des Chi2-Tests visualisieren
print("\nErgebnisse des Chi2-Tests:")
test_results = [
    ["Chi-Quadrat-Statistik", f"{chi2:.3f}"],
    ["p-Wert", f"{p}"],
    ["Freiheitsgrade", dof],
    ["Signifikanz", "Ja" if p < 0.05 else "Nein"]
]

# Tabellarische Ausgabe der Testergebnisse
```

```
test_results_markdown = tabulate(test_results, headers=["Metrik", "Wert"])
display(Markdown(test_results_markdown))
```

Vergleich der beobachteten und erwarteten Werte:

Wochentag	Beobachtet (Tip gegeben)	Erwartet (Tip gegeben)	Beobachtet (kein Tip)	Erwartet (kein Tip)
0	6003	4898.2	5221	6325.8
1	3244	2710.51	2967	3500.49
2	1931	1646.55	1842	2126.45
3	1397	1205.35	1365	1556.65
4	1406	1199.24	1342	1548.76
5	2005	2079.03	2759	2684.97
6	6462	6547.81	8542	8456.19
7	19897	19699.7	25244	25441.3
8	39013	38191	48500	49322
9	56300	54945	69604	70959
10	61461	61708	79940	79693
11	58575	60887.1	80945	78632.9
12	55220	58308.8	78392	75303.2
13	56133	59601.9	80442	76973.1
14	57426	60579.9	81390	78236.1
15	56499	60345.5	81780	77933.5
16	54224	58231.1	79210	75202.9
17	44257	48367.5	66575	62464.5
18	35118	38606	53346	49858
19	36331	29631.4	31568	38267.6
20	27821	22237	23134	28718
21	21782	16750.1	16600	21631.9
22	16934	13177.2	13261	17017.8
23	10674	8559.19	8939	11053.8

Ergebnisse des Chi²-Tests:

Metrik	Wert
Chi-Quadrat-Statistik	14819.908
p-Wert	0.0
Freiheitsgrade	23
Signifikanz	Ja

In [23]: # Daten vorbereiten
observed = contingency_table.values # Beobachtete Werte
expected = expected # Erwartete Werte
days = contingency_table.index # Wochentage

```

# Abweichungen berechnen (relativ oder absolut)
deviation = (observed - expected) / expected # Relative Abweichung (Proz
# deviation = observed - expected # Absolute Abweichung

# DataFrame für Heatmap erstellen
heatmap_data = pd.DataFrame(deviation, columns=["Trinkgeld gegeben", "Kei

# Heatmap zeichnen
fig, ax = plt.subplots(figsize=(10, 8))
plt.subplots_adjust(left=0.08)

# Maximalen absoluten Wert finden für symmetrische Farbskala
max_abs_value = np.max(np.abs(heatmap_data.values))
vmin = -max_abs_value
vmax = max_abs_value

# Heatmap erstellen
sns.heatmap(heatmap_data,
            annot=True,
            cmap="RdBu",
            center=0,
            fmt=".1%", # Prozentformat
            cbar_kws={'label': 'Relative Abweichung'},
            vmin=vmin,
            vmax=vmax,
            ax=ax)

# Titel und Achsenbeschriftungen
ax.set_title('Abweichungen zwischen beobachteten und erwarteten Werten\n',
             loc='left',
             weight='bold')
ax.set_xlabel('Trinkgeldverhalten')
ax.set_ylabel('Wochentag')

# Y-Achsen-Labels ausrichten
ax.set_yticklabels(ax.get_yticklabels(), rotation=0)

# Bildunterschrift
description = ('Die Heatmap visualisiert die relativen Abweichungen zwisc
               'für jeden Wochentag und das jeweilige Trinkgeldverhalten.
               'Positive Werte (rot) zeigen mehr Fälle als erwartet, negat
               'Die Werte in den Zellen geben die prozentuale Abweichung a

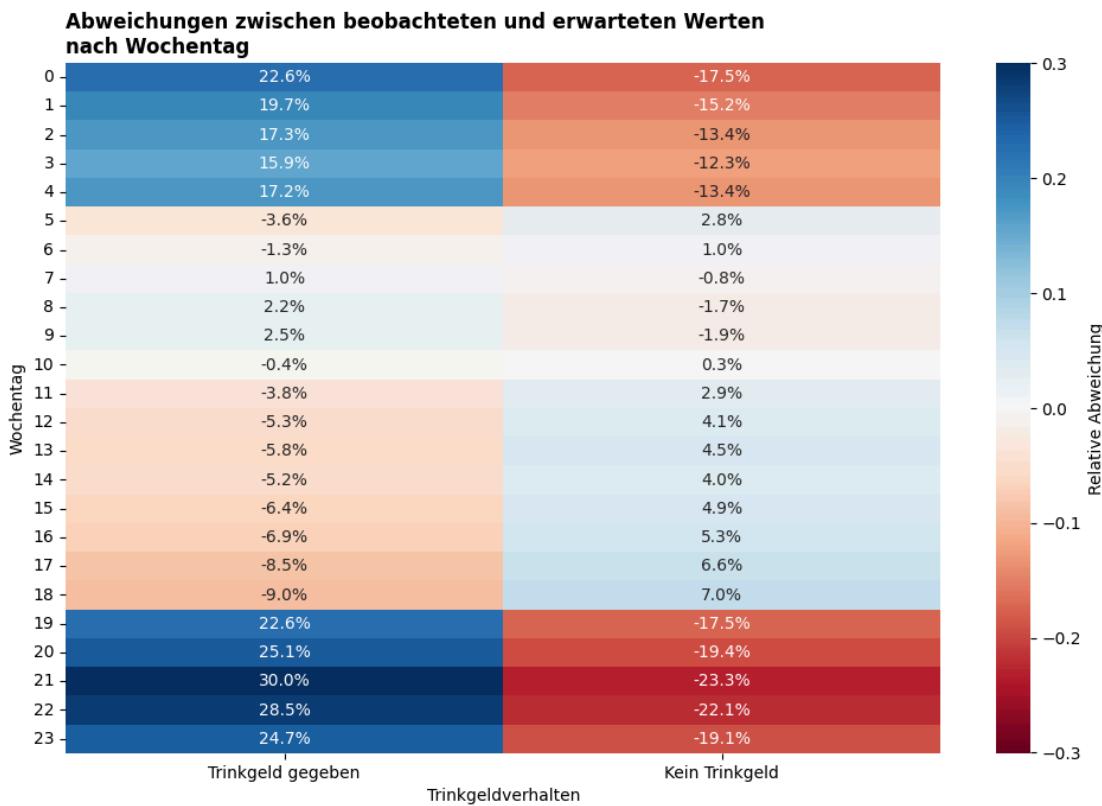
fig.text(0.05, 0.1, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, 0.04, description, wrap=True)

# Layout anpassen
plt.tight_layout()

# Zusätzlicher Platz für die Beschreibung
plt.subplots_adjust(bottom=0.2)

plt.show()

```

**Abbildung:**

Die Heatmap visualisiert die relativen Abweichungen zwischen beobachteten und erwarteten Werten für jeden Wochentag und das jeweilige Trinkgeldverhalten. Positive Werte (rot) zeigen mehr Fälle als erwartet, negative Werte (blau) weniger als erwartet. Die Werte in den Zellen geben die prozentuale Abweichung an.

```
In [24]: # Query:
result = session.query(
    Order.hour_of_day,
    func.cast(Order.tips, Integer).label('tips_given') # Boolean auf Integer
).all()
```

```
In [25]: # Erstelle DataFrame
df = pd.DataFrame(result, columns=['hour_of_day', 'tips_given'])

# Umwandlung in kategoriale Variablen
df['hour_of_day'] = df['hour_of_day'].astype('category')

X = df[['hour_of_day']]
y = df['tips_given']

# Preprocessing mit OneHotEncoder für day_of_week und hour_of_day
preprocessor = ColumnTransformer(
    transformers=[
        ('hour_of_day', OneHotEncoder(), ['hour_of_day'])
    ]
)

# Aufteilen der Daten in Trainings- und Testdatensatz
X_train_hour, X_test_hour, y_train_hour, y_test_hour = train_test_split(X, y, test_size=0.2, random_state=42)

# Pipeline erstellen
model_hour = Pipeline(steps=[
    ('preprocessor', preprocessor), # Vorverarbeitung
    ('classifier', LogisticRegression()) # Klassifikator
])
```

```
# Modell trainieren
model_hour.fit(X_train_hour, y_train_hour)

# Vorhersagen auf den Testdaten
y_pred_hour = model_hour.predict(X_test_hour)

# Genauigkeit des Modells
accuracy = accuracy_score(y_test_hour, y_pred_hour)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Zugriff auf Intercept und Koeffizienten
intercept = model_hour.named_steps['classifier'].intercept_[0] # Der Int
coefficients = model_hour.named_steps['classifier'].coef_.flatten() # Di

# Wenn OneHotEncoder verwendet wird, benötigen wir die Feature-Namen, um
# Dies ist besonders wichtig, da OneHotEncoding für die "day_of_week" und
# Holen der Spaltennamen des OneHotEncoders
ohe_feature_names = (
    model_hour.named_steps['preprocessor'].get_feature_names_out(input_fe
)

# Erstelle DataFrame mit den Koeffizienten und den entsprechenden Feature
coeff_df = pd.DataFrame({
    'Feature': ohe_feature_names,
    'Coefficient': coefficients
})

# Ausgabe des Intercepts und der Koeffizienten
print("Intercept:", intercept)
display(coeff_df)
```

Accuracy: 57.65%

Intercept: -0.12848531701987678

	Feature	Coefficient
0	hour_of_day_hour_of_day_0	0.319291
1	hour_of_day_hour_of_day_1	0.221361
2	hour_of_day_hour_of_day_2	0.148062
3	hour_of_day_hour_of_day_3	0.103731
4	hour_of_day_hour_of_day_4	0.098312
5	hour_of_day_hour_of_day_5	-0.166472
6	hour_of_day_hour_of_day_6	-0.150764
7	hour_of_day_hour_of_day_7	-0.117070
8	hour_of_day_hour_of_day_8	-0.081748
9	hour_of_day_hour_of_day_9	-0.079419
10	hour_of_day_hour_of_day_10	-0.133450
11	hour_of_day_hour_of_day_11	-0.192345
12	hour_of_day_hour_of_day_12	-0.228078
13	hour_of_day_hour_of_day_13	-0.239968
14	hour_of_day_hour_of_day_14	-0.219001
15	hour_of_day_hour_of_day_15	-0.238373
16	hour_of_day_hour_of_day_16	-0.256063
17	hour_of_day_hour_of_day_17	-0.274945
18	hour_of_day_hour_of_day_18	-0.285019
19	hour_of_day_hour_of_day_19	0.264714
20	hour_of_day_hour_of_day_20	0.316966
21	hour_of_day_hour_of_day_21	0.390402
22	hour_of_day_hour_of_day_22	0.357896
23	hour_of_day_hour_of_day_23	0.313719

```
In [ ]: y_prob_hour = model_hour.predict_proba(X_test_hour)[:, 1] # Nur die Wahr

# Berechnung der ROC-Kurve für Stunde des Tages
fpr_hour, tpr_hour, _ = roc_curve(y_test_hour, y_prob_hour)
roc_auc_hour = auc(fpr_hour, tpr_hour)

# Berechnung der Vorhersagewahrscheinlichkeiten für Wochentag
y_prob_day = model_day.predict_proba(X_test_day)[:, 1] # Nur die Wahrsch

# Berechnung der ROC-Kurve für Wochentag
fpr_day, tpr_day, _ = roc_curve(y_test_day, y_prob_day)
roc_auc_day = auc(fpr_day, tpr_day)

# Erstelle Subplots für die beiden ROC-Kurven
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(16, 6))
plt.subplots_adjust(left=0.08)

# Plot 1: ROC-Kurve für Stunde des Tages
```

```

ax1.plot(fpr_hour, tpr_hour, color='darkorange', lw=2,
         label=f'ROC-Kurve (AUC = {roc_auc_hour:.2f})')
ax1.plot([0, 1], [0, 1], color='navy', lw=2,
         linestyle='--', label='Zufallsklassifikator')

# Formatierung Plot 1
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.05])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('ROC-Kurve für die Vorhersage von Trinkgeldzahlungen\nbasierend auf der Tageszeit (AUC = {:.2f})'.format(roc_auc_hour), loc='left', weight='bold')
ax1.grid(True, linestyle='--', alpha=0.7)
ax1.legend(loc='lower right')

# Achsenbeschriftungen in Prozent für Plot 1
ax1.set_xticks([i/10 for i in range(0, 11)])
ax1.set_yticks([i/10 for i in range(0, 11)])
ax1.set_xticklabels([f'{i*10}%' for i in range(0, 11)])
ax1.set_yticklabels([f'{i*10}%' for i in range(0, 11)])

# Plot 2: ROC-Kurve für Wochentag
ax2.plot(fpr_day, tpr_day, color='darkorange', lw=2,
         label=f'ROC-Kurve (AUC = {roc_auc_day:.2f})')
ax2.plot([0, 1], [0, 1], color='navy', lw=2,
         linestyle='--', label='Zufallsklassifikator')

# Formatierung Plot 2
ax2.set_xlim([0.0, 1.0])
ax2.set_ylim([0.0, 1.05])
ax2.set_xlabel('False Positive Rate')
ax2.set_ylabel('True Positive Rate')
ax2.set_title('ROC-Kurve für die Vorhersage von Trinkgeldzahlungen\nbasierend auf dem Wochentag (AUC = {:.2f})'.format(roc_auc_day), loc='left', weight='bold')
ax2.grid(True, linestyle='--', alpha=0.7)
ax2.legend(loc='lower right')

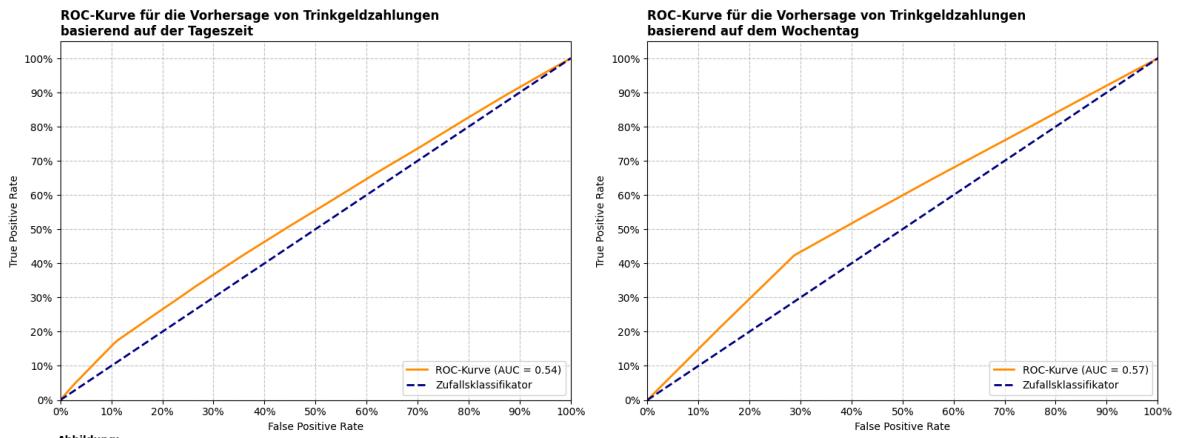
# Achsenbeschriftungen in Prozent für Plot 2
ax2.set_xticks([i/10 for i in range(0, 11)])
ax2.set_yticks([i/10 for i in range(0, 11)])
ax2.set_xticklabels([f'{i*10}%' for i in range(0, 11)])
ax2.set_yticklabels([f'{i*10}%' for i in range(0, 11)])

# Bildunterschrift
description = ('Die ROC-Kurven zeigen die Performance der Vorhersagemodelle.\nLinks: Modell basierend auf der Tageszeit (AUC = {:.2f}).\nRechts: Modell basierend auf dem Wochentag (AUC = {:.2f}).\nDie gestrichelte Linie entspricht einem Zufallsklassifikator.\nJe höher der AUC-Wert, desto besser die Vorhersagekraft des Modells.'.format(roc_auc_hour, roc_auc_day))

fig.text(0.05, 0, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, -0.055, description, wrap=True)

plt.tight_layout()
plt.show()

```



Interpretation der ROC-Kurven-Analyse



Vergleich der Vorhersagekraft



AUC-Werte im Vergleich:

- Wochentag: AUC = 0.57
- Tageszeit: AUC = 0.54
- Beide Werte liegen nur knapp über dem Zufallsniveau (0.50)



Zentrale Erkenntnisse:

- Wochentag hat minimal bessere Vorhersagekraft als Tageszeit
- Beide Zeitfaktoren sind als einzelne Prädiktoren sehr schwach
- Die statistisch nachgewiesenen Zusammenhänge haben geringe praktische Relevanz



Ausblick auf kombiniertes Modell:

Die schwache Einzelperformance der Zeitfaktoren deutet darauf hin, dass erst die Kombination verschiedener Features (zeitliche und nicht-zeitliche) zu einem aussagekräftigen Vorhersagmodell führen könnte. Dies wird in der nachfolgenden Analyse mit einem umfassenderen Modell untersucht.

```
In [26]: result_hours_days = session.query(
    Order.hour_of_day,
    Order.day_of_the_week,
    func.count(Order.order_id).label('total_orders'),
    func.sum(func.cast(Order.tips, Integer)).label('total_tips')
).group_by(Order.hour_of_day, Order.day_of_the_week).all()

# Daten sammeln
data = []
```

```

for row in result_hours_days:
    tip_quote = row.total_tips / row.total_orders if row.total_orders else 0
    data.append([
        row.hour_of_day,
        row.total_orders,
        row.total_tips,
        f"{tip_quote:.2f}" # Formatierte Trinkgeldquote
    ])

```

⌚ Kombinierte Analyse von Wochentag und Tageszeit

🔍 Zweidimensionale Zeitanalyse

Nach der separaten Betrachtung von Wochentagen und Tageszeiten untersuchen wir nun deren Wechselwirkung mittels einer detaillierten Heatmap-Analyse. Diese ermöglicht es uns, zeitliche "Hotspots" für Trinkgeldzahlungen zu identifizieren.

📊 Visualisierungsdetails:

- X-Achse: 24 Stunden eines Tages
- Y-Achse: Wochentage
- Farbintensität: Höhe der Trinkgeldquote
- Zahlenwerte: Exakte Trinkgeldquote in den jeweiligen Zeitfenstern

⌚ Untersuchungsziele:

- Identifikation von Zeitkombinationen mit besonders hoher/niedriger Trinkgeldquote
- Erkennung von Mustern und Anomalien
- Verständnis der Interaktion zwischen Wochentag und Tageszeit
- Basis für gezieltere zeitliche Geschäftsstrategien

In [27]:

```

data = {
    'hour_of_day': [row.hour_of_day for row in result_hours_days],
    'day_of_the_week': [row.day_of_the_week for row in result_hours_days],
    'total_orders': [row.total_orders for row in result_hours_days],
    'total_tips': [row.total_tips if row.total_tips else 0 for row in result_hours_days]
}
df = pd.DataFrame(data)

# Zusätzliche Spalte für Tip-Quote berechnen
df['tip_ratio'] = df['total_tips'] / df['total_orders']

# Pivot-Tabelle erstellen (Stunden als Spalten, Wochentage als Zeilen)
heatmap_data = df.pivot(index='day_of_the_week', columns='hour_of_day', values='total_tips')

# Heatmap erstellen
fig, ax = plt.subplots(figsize=(14, 8))
plt.subplots_adjust(left=0.08)

```

```
# Pivot-Tabelle erstellen
heatmap_data = df.pivot(index='day_of_the_week',
                        columns='hour_of_day',
                        values='tip_ratio')

# Maximalen Wert finden für Farbskala
vmax = np.nanmax(heatmap_data.values)

# Heatmap erstellen
sns.heatmap(heatmap_data,
            annot=True,
            cmap='RdBu_r', # Konsistente Farbpalette
            cbar_kws={'label': 'Trinkgeldquote'},
            xticklabels=[f"{int(h):02d}" for h in heatmap_data.columns],
            vmin=0,
            vmax=vmax,
            ax=ax)

# Titel und Achsenbeschriftungen
ax.set_title('Trinkgeldquote nach Wochentag und Tageszeit',
             loc='left',
             weight='bold')
ax.set_xlabel('Stunde des Tages')
ax.set_ylabel('Wochentag')

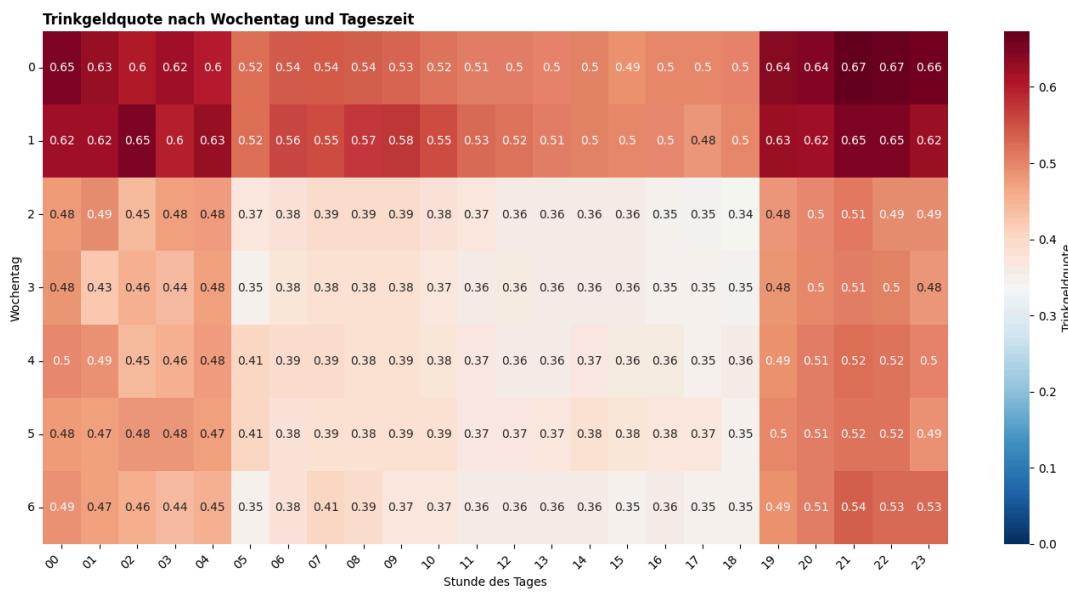
# Achsenbeschriftungen formatieren
ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
ax.set_yticklabels(ax.get_yticklabels(), rotation=0)

# Bildunterschrift
description = ('Die Heatmap zeigt die Trinkgeldquote (Anteil der Bestellung) für verschiedene Kombinationen von Wochentag und Tageszeit. Dunklere Farben repräsentieren einen höheren Anteil an Trinkgeld. Die Werte in den Zellen geben die genaue Trinkgeldquote in Prozent an.')
fig.text(0.05, 0.1, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, 0.05, description, wrap=True)

# Layout anpassen
plt.tight_layout()

# Zusätzlicher Platz für die Beschreibung
plt.subplots_adjust(bottom=0.2)

plt.show()
```

**Abbildung:**

Die Heatmap zeigt die Trinkgeldquote (Anteil der Bestellungen mit Trinkgeld) für verschiedene Kombinationen von Wochentag und Tageszeit. Dunklere Farben repräsentieren einen höheren Anteil an Trinkgeldzahlungen. Die Werte in den Zellen geben die genaue Trinkgeldquote in Prozent an.



Interpretation der kombinierten Zeitanalyse



Verstärkungseffekt der Zeitfaktoren



• Zeitliche Muster:

- Erhöhte Trinkgeldquote an Tagen 0 und 1
- Höhere Trinkgeldbereitschaft in den Nachtstunden (19-4 Uhr)
- Maximale Trinkgeldquoten bei Überlagerung beider Faktoren



Synergieeffekte:

- Die einzelnen beobachteten Effekte verstärken sich gegenseitig
- Besonders hohe Trinkgeldwahrscheinlichkeit in Nachtstunden an den Tagen 0&1
- Klare Identifikation von "Premium-Zeitfenstern" für Trinkgeldzahlungen



Praktische Implikationen:

- Optimale Zeitfenster für serviceorientierte Maßnahmen identifiziert
- Potenzial für zeitbasierte Personalplanung und Service-Optimierung
- Möglichkeit zur gezielten Anpassung von Marketing- und Serviceangeboten
- Grundlage für strategische Entscheidungen in der Ressourcenplanung

Abstand vorheriger Bestellung & Trinkgeldverhalten

```
In [28]: # Query: Zähle die Anzahl der Bestellungen mit und ohne Trinkgeld basiere
result = session.query(
    Order.days_since_prior_order,
    func.count(Order.order_id).label('total_orders'),
    func.sum(func.cast(Order.tips, Integer)).label('total_tips')
).group_by(Order.days_since_prior_order).all()

result_test = session.query(
    Order.days_since_prior_order,
    Order.tips # Hole die Trinkgelder direkt, ohne Aggregation
).all()
```



Analyse des Bestellrhythmus und Trinkgeldverhaltens



Einfluss des Bestellabstands

Als letzten zeitlichen Aspekt untersuchen wir den Zusammenhang zwischen dem Abstand aufeinanderfolgender Bestellungen und dem Trinkgeldverhalten. Diese Analyse könnte Aufschluss über Kundenloyalität und Bestellgewohnheiten geben.



Visualisierungsaufbau:

- **Linker Plot:** Absolute Verteilung der Bestellungen nach Zeitabstand
- **Rechter Plot:** Trinkgeldquote in Abhängigkeit vom Bestellabstand
- **Besonderheit:** "first" kennzeichnet Erstbestellungen neuer Kunden



Kernfragen:

- Gibt es typische Bestellrhythmen bei den Kunden?
- Unterscheidet sich das Trinkgeldverhalten von Neu- und Bestandskunden?
- Existiert ein Zusammenhang zwischen Bestellfrequenz und Trinkgeldbereitschaft?
- Lassen sich Muster in den Bestellabständen erkennen?

```
In [29]: # plot code
days_since_prior = [
    "first" if row.days_since_prior_order is None else str(int(row.days_s
for row in result
]
total_orders = [row.total_orders for row in result]
total_tips = [row.total_tips if row.total_tips else 0 for row in result]
tip_ratios = [row.total_tips / row.total_orders if row.total_orders > 0 e

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(18, 6))
plt.subplots_adjust(left=0.08)

# Daten vorbereiten
width = 0.4
x = range(len(days_since_prior))
```

```

# Diagramm 1: Gesamtbestellungen und Bestellungen mit Trinkgeld
bars1 = ax1.bar(x, total_orders, width, label='Bestellungen insgesamt',
                 color='lightblue')
bars2 = ax1.bar([i + width for i in x], total_tips, width,
                 label='Bestellungen mit Trinkgeld', color='lightgreen')

# Formatierung Diagramm 1
ax1.set_xlabel('Tage seit vorheriger Bestellung')
ax1.set_ylabel('Anzahl der Bestellungen')
ax1.set_title('Bestellungen nach Zeitabstand zur vorherigen Bestellung',
              loc='left', weight='bold')
ax1.set_xticks([i + width / 2 for i in x])
ax1.set_xticklabels(days_since_prior, rotation=45, ha='right')
ax1.grid(True, axis='y', linestyle='--', alpha=0.7)
ax1.legend()

# Diagramm 2: Trinkgeldquote
bars3 = ax2.bar(range(len(days_since_prior)), tip_ratios,
                 color='purple', alpha=0.7)

# Formatierung Diagramm 2
ax2.set_xlabel('Tage seit vorheriger Bestellung')
ax2.set_ylabel('Trinkgeldquote')
ax2.set_title('Trinkgeldquote nach Zeitabstand zur vorherigen Bestellung'
              loc='left', weight='bold')
ax2.set_ylim(0, 1)
ax2.set_yticks([i / 10 for i in range(0, 11)])
ax2.set_yticklabels([f'{i * 10}%' for i in range(0, 11)])
ax2.set_xticks(range(len(days_since_prior)))
ax2.set_xticklabels(days_since_prior, rotation=45, ha='right')
ax2.grid(True, axis='y', linestyle='--', alpha=0.7)

# Bildunterschrift
description = ('Die Grafiken zeigen die Bestellmuster in Abhängigkeit von  

               'Links: Absolute Anzahl der Gesamtbestellungen (blau) und B  

               'Rechts: Trinkgeldquote (Anteil der Bestellungen mit Trinkg  

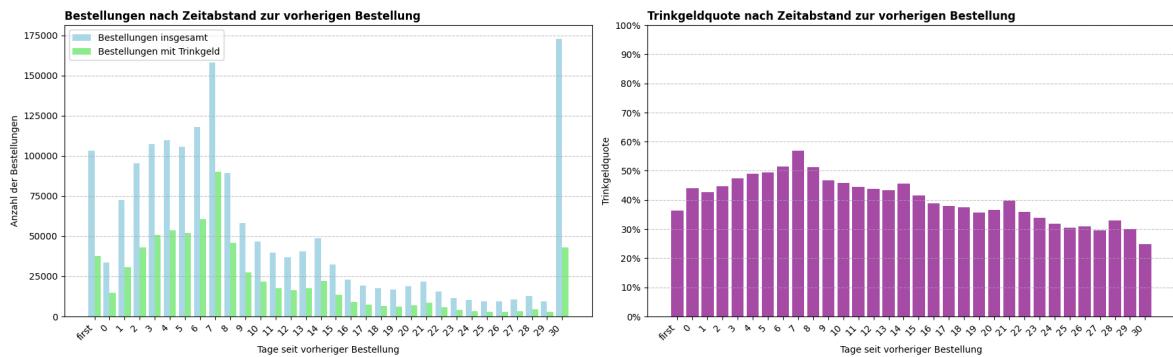
               '"first" bezeichnet dabei Erstbestellungen von Kunden.')
fig.text(0.05, 0, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, -0.055, description, wrap=True)

# Layout anpassen
plt.tight_layout()

# Zusätzlicher Platz für die Beschreibung
plt.subplots_adjust(bottom=0.2)

plt.show()

```



```

        ],
}

df = pd.DataFrame(data)

# Kreuztabelle erstellen
crosstab = pd.crosstab(
    index=df["days_since_prior_order"], # Zeilen: Tage seit vorheriger B
    columns=df["tip_status"],           # Spalten: Mit oder ohne Trinkge
).fillna(0)

```

```

In [31]: chi2, p, dof, expected = chi2_contingency(crosstab)

# Tabellenausgabe: Vergleich von beobachteten und erwarteten Werten
observed = crosstab.values # Beobachtete Werte
expected = expected # Erwartete Werte

# Vergleichstabelle erstellen
comparison_data = []
for i, day in enumerate(contingency_table.index):
    comparison_data.append([
        day, # Wochentag
        observed[i][0], # Beobachtet: Trinkgeld gegeben
        expected[i][0], # Erwartet: Trinkgeld gegeben
        observed[i][1], # Beobachtet: Kein Trinkgeld
        expected[i][1], # Erwartet: Kein Trinkgeld
    ])

# Tabellenkopf
headers = ["Wochentag", "Beobachtet (Tip gegeben)", "Erwartet (Tip gegeben)", "Beobachtet (kein Tip)", "Erwartet (kein Tip)"]

# Tabelle formatieren
print("\nVergleich der beobachteten und erwarteten Werte:")
camparsion_data_markdown = tabulate(comparison_data, headers=headers, tab
display(Markdown(camparsion_data_markdown))

# Ergebnisse des Chi2-Tests visualisieren
print("\nErgebnisse des Chi2-Tests:")
test_results = [
    ["Chi-Quadrat-Statistik", f"{chi2:.3f}"],
    ["p-Wert", f"{p}"],
    ["Freiheitsgrade", dof],
    ["Signifikanz", "Ja" if p < 0.05 else "Nein"]
]
]

# Tabellarische Ausgabe der Testergebnisse
test_results_markdown = tabulate(test_results, headers=["Metrik", "Wert"])
display(Markdown(test_results_markdown))

```

Vergleich der beobachteten und erwarteten Werte:

Wochentag	Beobachtet (Tip gegeben)	Erwartet (Tip gegeben)	Beobachtet (kein Tip)	Erwartet (kein Tip)
0	6003	4898.2	5221	6325.8
1	3244	2710.51	2967	3500.49
2	1931	1646.55	1842	2126.45
3	1397	1205.35	1365	1556.65
4	1406	1199.24	1342	1548.76
5	2005	2079.03	2759	2684.97
6	6462	6547.81	8542	8456.19
7	19897	19699.7	25244	25441.3
8	39013	38191	48500	49322
9	56300	54945	69604	70959
10	61461	61708	79940	79693
11	58575	60887.1	80945	78632.9
12	55220	58308.8	78392	75303.2
13	56133	59601.9	80442	76973.1
14	57426	60579.9	81390	78236.1
15	56499	60345.5	81780	77933.5
16	54224	58231.1	79210	75202.9
17	44257	48367.5	66575	62464.5
18	35118	38606	53346	49858
19	36331	29631.4	31568	38267.6
20	27821	22237	23134	28718
21	21782	16750.1	16600	21631.9
22	16934	13177.2	13261	17017.8
23	10674	8559.19	8939	11053.8

Ergebnisse des Chi²-Tests:

Metrik	Wert
Chi-Quadrat-Statistik	53883.206
p-Wert	0.0
Freiheitsgrade	31
Signifikanz	Ja

In [32]:

```
# plot code
observed = crosstab.values # Beobachtete Werte
expected = expected # Erwartete Werte
days = crosstab.index # Wochentage

# Daten vorbereiten
fig, ax = plt.subplots(figsize=(10, 8))
plt.subplots_adjust(left=0.08)
```

```

# Abweichungen berechnen
deviation = (observed - expected) / expected

# DataFrame für Heatmap erstellen
# Hier verwenden wir die original days als Index
heatmap_data = pd.DataFrame(
    deviation,
    columns=["Trinkgeld gegeben", "Kein Trinkgeld"],
    index=days # days enthält die Werte für days_since_prior_order
)

# Maximalen absoluten Wert finden für symmetrische Farbskala
max_abs_value = np.max(np.abs(deviation))
vmin = -max_abs_value
vmax = max_abs_value

# Heatmap erstellen
sns.heatmap(heatmap_data,
            annot=True,
            cmap="RdBu",
            center=0,
            fmt=".1%", # Prozentformat
            cbar_kws={'label': 'Relative Abweichung'},
            vmin=vmin,
            vmax=vmax,
            ax=ax)

# Titel und Achsenbeschriftungen
ax.set_title('Abweichungen zwischen beobachteten und erwarteten Werten\n' +
             'nach Tagen seit der letzten Bestellung',
             loc='left',
             weight='bold')
ax.set_xlabel('Trinkgeldverhalten')
ax.set_ylabel('Tage seit letzter Bestellung')

# Y-Achsen-Labels ausrichten
ax.set_yticklabels(ax.get_yticklabels(), rotation=0)

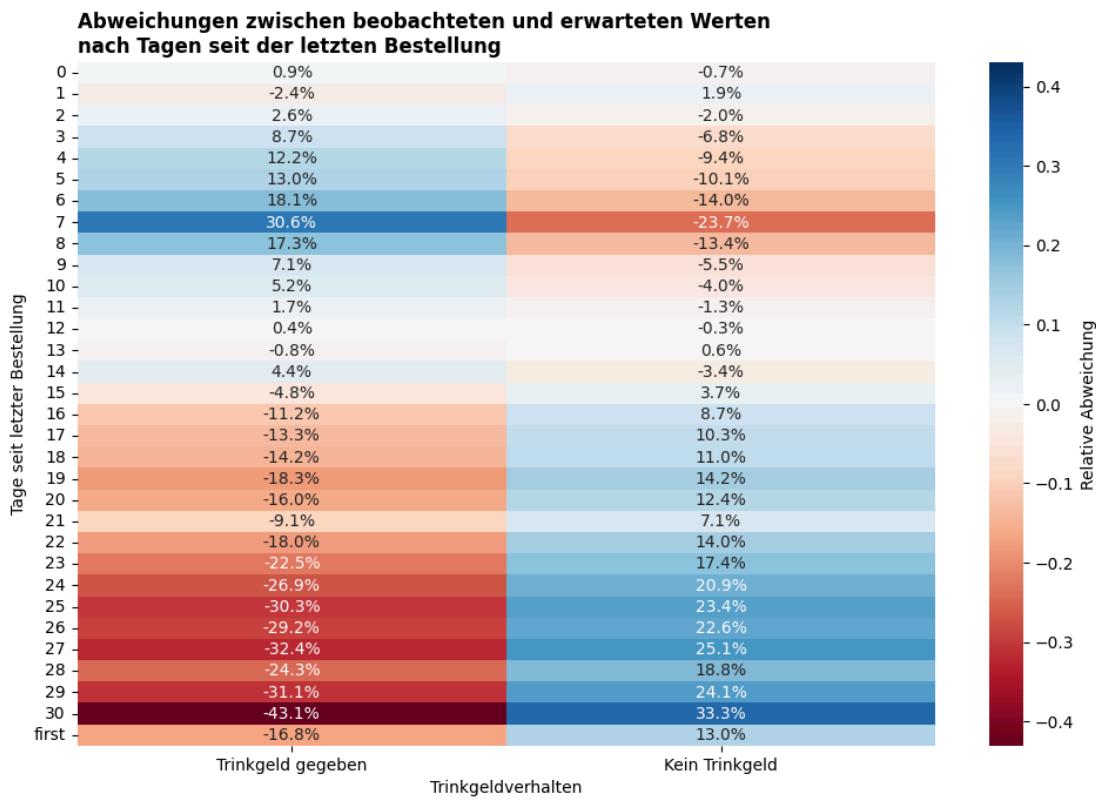
# Bildunterschrift
description = ('Die Heatmap visualisiert die relativen Abweichungen zwisc' +
               'für verschiedene Zeitabstände zur vorherigen Bestellung. ' +
               'Positive Werte (rot) zeigen mehr Fälle als erwartet, negat' +
               'Die Werte in den Zellen geben die prozentuale Abweichung a' +
               '"first" bezeichnet dabei Erstbestellungen von Kunden.')
fig.text(0.05, 0.12, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, 0.06, description, wrap=True)

# Layout anpassen
plt.tight_layout()

# Zusätzlicher Platz für die Beschreibung
plt.subplots_adjust(bottom=0.2)

plt.show()

```

**Abbildung:**

Die Heatmap visualisiert die relativen Abweichungen zwischen beobachteten und erwarteten Werten für verschiedene Zeitabstände zur vorherigen Bestellung. Positive Werte (rot) zeigen mehr Fälle als erwartet, negative Werte (blau) weniger als erwartet. Die Werte in den Zellen geben die prozentuale Abweichung an. "first" bezeichnet dabei Erstbestellungen von Kunden.

In [33]: # Daten aus der Order-Tabelle extrahieren (inkl. Stunde des Tages)

```
result = session.query(
    Order.hour_of_day,
    Order.day_of_the_week,
    Order.days_since_prior_order,
    func.cast(Order.tips, Integer).label('tips_given') # Boolean auf Int
).all()
```

⌚ Vergleichende Analyse aller Zeitfaktoren

🔍 Kombiniertes Vorhersagemodell

Als Abschluss unserer zeitbasierten Analyse kombinieren wir nun alle untersuchten Zeitfaktoren in einem einzigen Vorhersagemodell und vergleichen dessen Leistung mit den vorherigen Einzelmodellen.

📊 Modellvergleich:

- **Einzelmodelle:**
 - Modell 1: Nur Wochentag (AUC = 0.57)
 - Modell 2: Nur Tageszeit (AUC = 0.54)
- **Kombiniertes Modell:**
 - Features: Wochentag, Tageszeit und Bestellabstand
 - Integration aller zeitlichen Dimensionen

⌚ Untersuchungsziele:

- Bewertung möglicher Synergieeffekte der Zeitfaktoren
- Quantifizierung des Mehrwerts der Feature-Kombination
- Abschließende Einschätzung der zeitbasierten Vorhersagekraft

```
In [34]: # Erstelle DataFrame
df = pd.DataFrame(result, columns=['hour_of_day', 'day_of_the_week', 'day'])

# Umwandlung in kategoriale Variablen
df['hour_of_day'] = df['hour_of_day'].astype('category')
df['day_of_the_week'] = df['day_of_the_week'].astype('category')
df['days_since_prior_order'] = df['days_since_prior_order'].astype('category')

X = df[['hour_of_day', 'day_of_the_week', 'days_since_prior_order']]
y = df['tips_given']

# Preprocessing mit OneHotEncoder für day_of_week und hour_of_day
preprocessor = ColumnTransformer(
    transformers=[
        ('hour_of_day', OneHotEncoder(), ['hour_of_day', 'day_of_the_week'])
    ]
)

# Aufteilen der Daten in Trainings- und Testdatensatz
X_train_all, X_test_all, y_train_all, y_test_all = train_test_split(X, y, test_size=0.2, random_state=42)

# Pipeline erstellen
model_all = Pipeline(steps=[
    ('preprocessor', preprocessor), # Vorverarbeitung
    ('classifier', LogisticRegression()) # Klassifikator
])

# Modell trainieren
model_all.fit(X_train_all, y_train_all)

# Vorhersagen auf den Testdaten
y_pred_all = model_all.predict(X_test_all)

# Genauigkeit des Modells
accuracy = accuracy_score(y_test_all, y_pred_all)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Zugriff auf Intercept und Koeffizienten
intercept = model_all.named_steps['classifier'].intercept_[0] # Der Intercept
coefficients = model_all.named_steps['classifier'].coef_.flatten() # Die Koeffizienten

# Wenn OneHotEncoder verwendet wird, benötigen wir die Feature-Namen, um
# Dies ist besonders wichtig, da OneHotEncoding für die "day_of_week" und
# Holen der Spaltennamen des OneHotEncoders
ohe_feature_names = (
    model_all.named_steps['preprocessor'].get_feature_names_out(input_features)
)

# Erstelle DataFrame mit den Koeffizienten und den entsprechenden Feature-Namen
coeff_df = pd.DataFrame({
    'Feature': ohe_feature_names,
```

```

    'Coefficient': coefficients
}

# Ausgabe des Intercepts und der Koeffizienten
print("Intercept:", intercept)
display(coeff_df)

```

Accuracy: 61.46%

Intercept: -0.2577138480810662

	Feature	Coefficient
0	hour_of_day_hour_of_day_0	0.328639
1	hour_of_day_hour_of_day_1	0.231733
2	hour_of_day_hour_of_day_2	0.151920
3	hour_of_day_hour_of_day_3	0.107983
4	hour_of_day_hour_of_day_4	0.114304
...
58	hour_of_day_days_since_prior_order_27.0	-0.479697
59	hour_of_day_days_since_prior_order_28.0	-0.369656
60	hour_of_day_days_since_prior_order_29.0	-0.486733
61	hour_of_day_days_since_prior_order_30.0	-0.726265
62	hour_of_day_days_since_prior_order_nan	-0.172012

63 rows × 2 columns

```

In [35]: y_prob_hour = model_hour.predict_proba(X_test_hour)[:, 1] # Nur die Wahr

# Berechnung der ROC-Kurve für Stunde des Tages
fpr_hour, tpr_hour, _ = roc_curve(y_test_hour, y_prob_hour)
roc_auc_hour = auc(fpr_hour, tpr_hour)

# Berechnung der Vorhersagewahrscheinlichkeiten für Wochentag
y_prob_day = model_day.predict_proba(X_test_day)[:, 1] # Nur die Wahrsch

# Berechnung der ROC-Kurve für Wochentag
fpr_day, tpr_day, _ = roc_curve(y_test_day, y_prob_day)
roc_auc_day = auc(fpr_day, tpr_day)

# Berechnung der Vorhersagewahrscheinlichkeiten für Wochentag
y_prob_all = model_all.predict_proba(X_test_all)[:, 1] # Nur die Wahrsch

# Berechnung der ROC-Kurve für Wochentag
fpr_all, tpr_all, _ = roc_curve(y_test_all, y_prob_all)
roc_auc_all = auc(fpr_all, tpr_all)
# Erstelle Subplots für die beiden ROC-Kurven
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(18, 6))
plt.subplots_adjust(left=0.08)

# Plot 1: ROC-Kurve für alle Features
ax1.plot(fpr_all, tpr_all, color='darkorange', lw=2,
          label=f'ROC-Kurve (AUC = {roc_auc_all:.2f})')
ax1.plot([0, 1], [0, 1], color='navy', lw=2,
          linestyle='--', label='Zufallsklassifikator')

```

```

# Plot 2: ROC-Kurve für Stunde des Tages
ax2.plot(fpr_hour, tpr_hour, color='darkorange', lw=2,
          label=f'ROC-Kurve (AUC = {roc_auc_hour:.2f})')
ax2.plot([0, 1], [0, 1], color='navy', lw=2,
          linestyle='--', label='Zufallsklassifikator')

# Plot 3: ROC-Kurve für Wochentag
ax3.plot(fpr_day, tpr_day, color='darkorange', lw=2,
          label=f'ROC-Kurve (AUC = {roc_auc_day:.2f})')
ax3.plot([0, 1], [0, 1], color='navy', lw=2,
          linestyle='--', label='Zufallsklassifikator')

# Formatierung für alle Subplots
for ax, title in zip([ax1, ax2, ax3],
                     ['Kombiniertes Modell\nAlle Features aus Frage 3',
                      'Modell basierend auf\nTageszeit',
                      'Modell basierend auf\nWochentag']):
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title(title, loc='left', weight='bold')
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.legend(loc='lower right')

# Prozentuale Achsenbeschriftungen
ax.set_xticks([i/10 for i in range(0, 11)])
ax.set_yticks([i/10 for i in range(0, 11)])
ax.set_xticklabels([f"{i*10}%" for i in range(0, 11)])
ax.set_yticklabels([f"{i*10}%" for i in range(0, 11)])

# Bildunterschrift
description = ('Die ROC-Kurven zeigen die Performance der verschiedenen V
               'Links: Modell mit allen Features (AUC = {:.2f}). '
               'Mitte: Modell basierend nur auf der Tageszeit (AUC = {:.2
               'Rechts: Modell basierend nur auf dem Wochentag (AUC = {:.2
               'Die gestrichelte Linie entspricht einem Zufallsklassifika
               'Je höher der AUC-Wert, desto besser die Vorhersagekraft d
               roc_auc_all, roc_auc_hour, roc_auc_day))

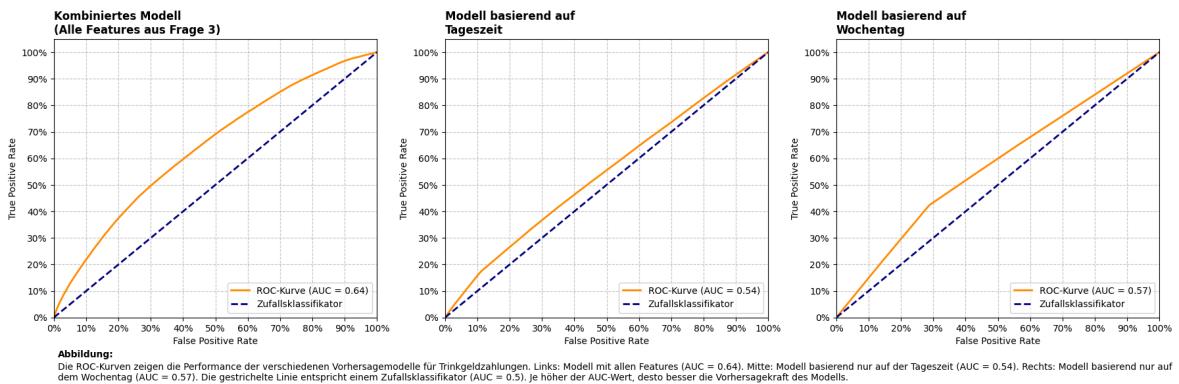
fig.text(0.05, 0.1, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, 0.045, description, wrap=True)

# Layout anpassen
plt.tight_layout()

# Zusätzlicher Platz für die Beschreibung
plt.subplots_adjust(bottom=0.2)

plt.show()

```



Interpretation des kombinierten Modells



Analyse der Modellverbesserung



Vergleich der AUC-Werte:

- Einzelmodelle:
 - Wochentag: AUC = 0.57
 - Tageszeit: AUC = 0.54
- Kombiniertes Modell: AUC = 0.64



Zentrale Erkenntnisse:

- Verbesserung gegenüber den Einzelmodellen
- Bestätigung von Synergieeffekten zwischen den Zeitfaktoren
- Überlegenheit des multidimensionalen Ansatzes



Praktische Bedeutung:

- Moderate, aber signifikante Vorhersagekraft der kombinierten Zeitfaktoren
- Zeitliche Faktoren erklären etwa 14% mehr als zufällige Vorhersagen
- Potenzial für weitere Verbesserungen durch zusätzliche Features (z.B. Kundenmerkmale, Bestellwert)
- Basis für differenziertere Geschäftsstrategien in der Zeitplanung



Fazit:

Die Kombination aller Zeitfaktoren führt zu einer spürbaren Verbesserung der Vorhersagekraft, zeigt aber auch, dass das Trinkgeldverhalten von weiteren, nicht-zeitlichen Faktoren beeinflusst wird.

Handlungsempfehlungen

Abteilung Disposition

```
In [36]: def get_enhanced_features(session):
    query = session.query(
        Order.order_id,
        Order.hour_of_day,
        Order.day_of_the_week,
        Order.days_since_prior_order,
        Order.tips.label('tips_given'),
        Order.user_id,
        # Alkohol-Produkte in Bestellung
        func.max(case((Department.department_name == 'alcohol', 1), else_=
        # Tiefkühl-Produkte in Bestellung
        func.max(case((Department.department_name == 'frozen', 1), else_=
        # Organische Produkte
        func.sum(case((Product.product_name.like('%organic%'), 1), else_=
        func.count(Product.product_id).label('total_products'))
    ).select_from(Order).join(Einkaufskorb).join(Product).join(Department
        Order.order_id,
        Order.hour_of_day,
        Order.day_of_the_week,
        Order.days_since_prior_order,
        Order.tips,
        Order.user_id
    )

    # Daten in DataFrame laden
    df = pd.read_sql(query.statement, session.bind)

    # Berechne organic_ratio
    df['organic_ratio'] = df['organic_count'] / df['total_products']

    # Berechne user_tip_ratio
    user_tips = df.groupby('user_id')['tips_given'].mean().reset_index()
    user_tips.columns = ['user_id', 'user_tip_ratio']

    # Merge mit Hauptdaten
    df = df.merge(user_tips, on='user_id', how='left')

    # Berechne user_order_count
    user_orders = df.groupby('user_id').size().reset_index()
    user_orders.columns = ['user_id', 'user_order_count']
    df = df.merge(user_orders, on='user_id', how='left')

    # Überprüfe die verfügbaren Spalten
    # print("Available columns:", df.columns.tolist())

    return df

def train_enhanced_model(df):
    # Umwandlung in kategoriale Variablen
    df['hour_of_day'] = df['hour_of_day'].astype('category')
    df['day_of_the_week'] = df['day_of_the_week'].astype('category')
    df['days_since_prior_order'] = df['days_since_prior_order'].astype('c

    # Definiere Features
```

```

categorical_features = ['hour_of_day', 'day_of_the_week', 'days_since']
numerical_features = ['has_alcohol', 'has_frozen', 'organic_ratio',
                      'user_order_count', 'user_tip_ratio']

X = df[categorical_features + numerical_features]
y = df['tips_given']

# Preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_features),
        ('num', 'passthrough', numerical_features)
    ]
)

# Train-Test-Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Pipeline erstellen
model = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=1000))
])

# Modell trainieren
model.fit(X_train, y_train)

# Vorhersagen auf den Testdaten
y_pred = model.predict(X_test)

# Genauigkeit des Modells
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Zugriff auf Intercept und Koeffizienten
intercept = model.named_steps['classifier'].intercept_[0]
coefficients = model.named_steps['classifier'].coef_.flatten()

# Feature-Namen generieren
cat_feature_names = model.named_steps['preprocessor'].named_transformers_
all_feature_names = np.concatenate([cat_feature_names, numerical_featu

# Erstelle DataFrame mit den Koeffizienten
coeff_df = pd.DataFrame({
    'Feature': all_feature_names,
    'Coefficient': coefficients
})

# Ausgabe des Intercepts und der Koeffizienten
print("\nIntercept:", intercept)
display(coeff_df)

return model, accuracy, X_train, X_test, y_train, y_test

# Hauptausführung
try:
    # Daten holen
enhanced_data = get_enhanced_features(session)

```

```

print("\nShape of data:", enhanced_data.shape)

# Model trainieren
model, accuracy, X_train, X_test, y_train, y_test = train_enhanced_mo

except Exception as e:
    print(f"Error occurred: {str(e)}")

```

Shape of data: (1673021, 13)

Accuracy: 80.15%

Intercept: -2.271823117154158

	Feature	Coefficient
0	hour_of_day_0	0.406899
1	hour_of_day_1	0.436991
2	hour_of_day_2	0.314170
3	hour_of_day_3	0.311803
4	hour_of_day_4	0.327951
...
63	has_alcohol	1.184161
64	has_frozen	-0.459821
65	organic_ratio	0.106204
66	user_order_count	0.001337
67	user_tip_ratio	5.924621

68 rows × 2 columns

```

In [37]: y_prob_enhanced = model.predict_proba(X_test)[:, 1]

# Berechnung der ROC-Kurve für das erweiterte Modell
fpr_enhanced, tpr_enhanced, _ = roc_curve(y_test, y_prob_enhanced)
roc_auc_enhanced = auc(fpr_enhanced, tpr_enhanced)

# Erstelle 2x2 Subplots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2, figsize=(1
plt.subplots_adjust(left=0.08)

# Plot 1 (oben links): ROC-Kurve für erweitertes Modell
ax1.plot(fpr_enhanced, tpr_enhanced, color='darkorange', lw=2,
          label=f'ROC-Kurve (AUC = {roc_auc_enhanced:.2f})')
ax1.plot([0, 1], [0, 1], color='navy', lw=2,
          linestyle='--', label='Zufallsklassifikator')

# Plot 2 (oben rechts): ROC-Kurve für alle bisherigen Features
ax2.plot(fpr_all, tpr_all, color='darkorange', lw=2,
          label=f'ROC-Kurve (AUC = {roc_auc_all:.2f})')
ax2.plot([0, 1], [0, 1], color='navy', lw=2,
          linestyle='--', label='Zufallsklassifikator')

# Plot 3 (unten links): ROC-Kurve für Stunde des Tages
ax3.plot(fpr_hour, tpr_hour, color='darkorange', lw=2,
          label=f'ROC-Kurve (AUC = {roc_auc_hour:.2f})')

```

```

ax3.plot([0, 1], [0, 1], color='navy', lw=2,
         linestyle='--', label='Zufallsklassifikator')

# Plot 4 (unten rechts): ROC-Kurve für Wochentag
ax4.plot(fpr_day, tpr_day, color='darkorange', lw=2,
          label=f'ROC-Kurve (AUC = {roc_auc_day:.2f})')
ax4.plot([0, 1], [0, 1], color='navy', lw=2,
          linestyle='--', label='Zufallsklassifikator')

# Formatierung für alle Subplots
for ax, title in zip([ax1, ax2, ax3, ax4],
                      ['Erweitertes Modell\n(Mit zusätzlichen Features)',
                       'Kombiniertes Modell\n(Alle Features aus Frage 3)',
                       'Modell basierend auf\nTageszeit',
                       'Modell basierend auf\nWochentag']):
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.set_title(title, loc='left', weight='bold')
    ax.grid(True, linestyle='--', alpha=0.7)
    ax.legend(loc='lower right')

# Prozentuale Achsenbeschriftungen
ax.set_xticks([i/10 for i in range(0, 11)])
ax.set_yticks([i/10 for i in range(0, 11)])
ax.set_xticklabels([f"{i*10}%" for i in range(0, 11)])
ax.set_yticklabels([f"{i*10}%" for i in range(0, 11)])

# Bildunterschrift
description = ('Die ROC-Kurven zeigen die Performance der verschiedenen V'
               'Oben links: Erweitertes Modell mit zusätzlichen Features'
               'Oben rechts: Modell mit allen Features aus Frage 3 (AUC ='
               'Unten links: Modell basierend nur auf der Tageszeit (AUC'
               'Unten rechts: Modell basierend nur auf dem Wochentag (AUC'
               'Die gestrichelte Linie entspricht einem Zufallsklassifika'
               'Je höher der AUC-Wert, desto besser die Vorhersagekraft d'
               'roc_auc_enhanced, roc_auc_all, roc_auc_hour, roc_auc_d')

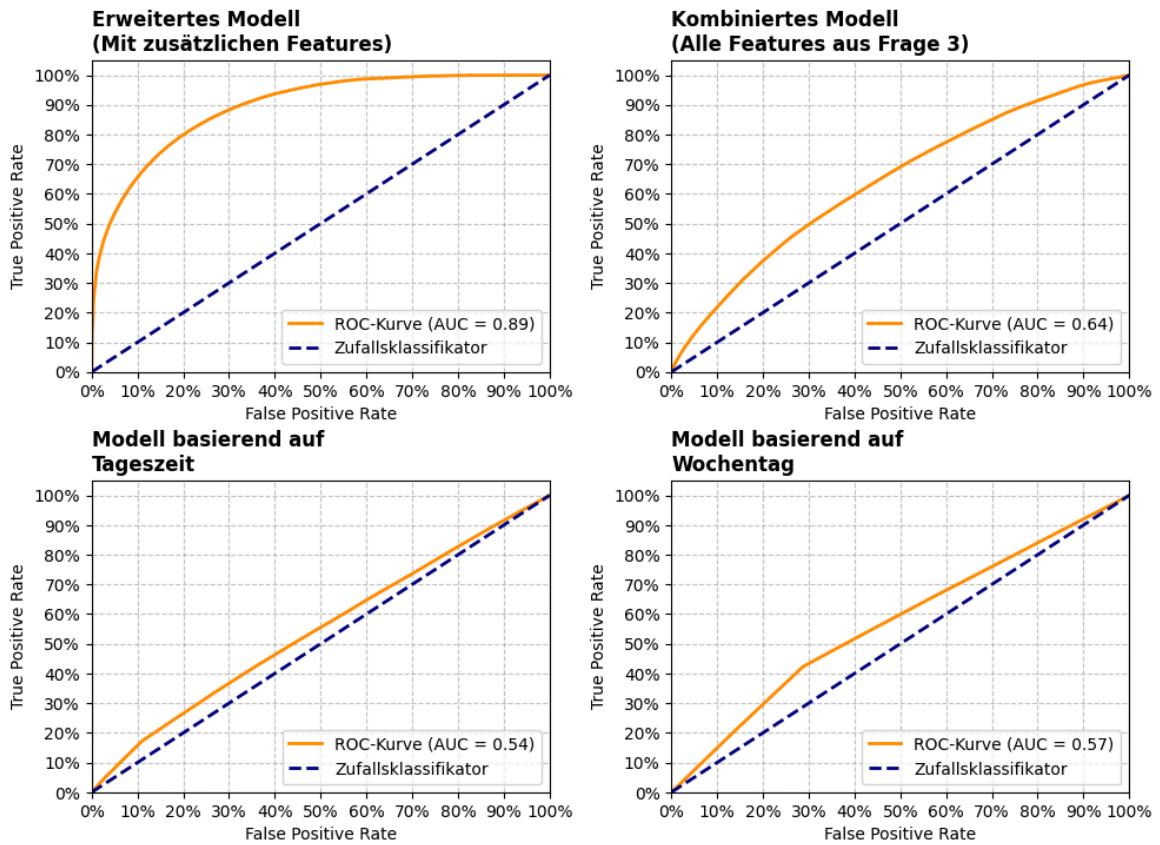
fig.text(0.05, 0.05, 'Abbildung:', weight='bold', ha='left')
fig.text(0.05, -0.04, description, wrap=True)

# Layout anpassen
plt.tight_layout()

# Zusätzlicher Platz für die Beschreibung
plt.subplots_adjust(bottom=0.15)

plt.show()

```

**Abbildung:**

Die ROC-Kurven zeigen die Performance der verschiedenen Vorhersagemodelle für Trinkgeldzahlungen. Oben links: Erweitertes Modell mit zusätzlichen Features (AUC = 0.89), Oben rechts: Modell mit allen Features aus Frage 3 (AUC = 0.64), Unten links: Modell basierend nur auf der Tageszeit (AUC = 0.54), Unten rechts: Modell basierend nur auf dem Wochentag (AUC = 0.57). Die gestrichelte Linie entspricht einem Zufallsklassifikator (AUC = 0.5). Je höher der AUC-Wert, desto besser die Vorhersagekraft des Modells.