Analyse Retentionrate der Kohorten im Loyalty Program

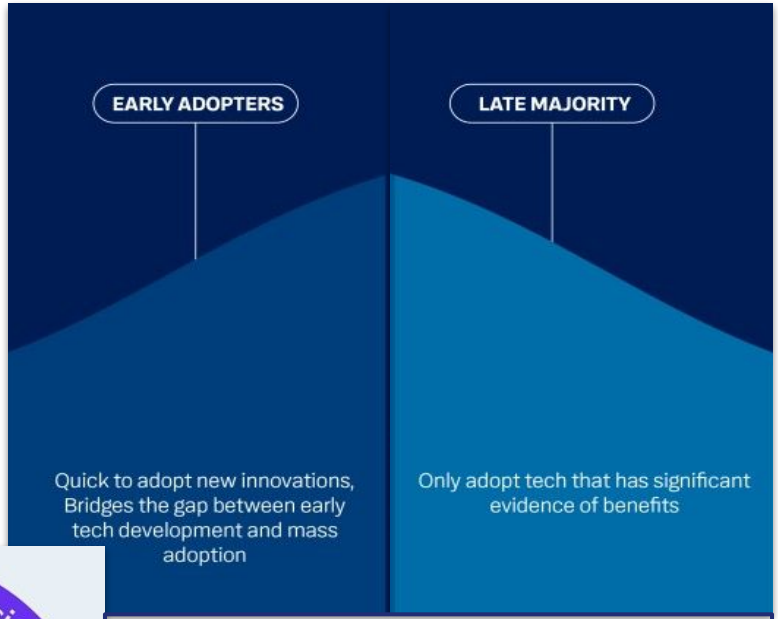
Zeigen Late Adopters eine schlechtere Retention?

Abschlusspräsentation Data Analytics with Statistics
16.01.2025
Michelle Mai

Einführung in das Thema

Identifiziert sich heute ein Kunde das erste Mal an der Kasse mit der App, so gehört er zu der Kohorte 202501.
Alle Kunden, die sich im Januar 2025 das erste Mal identifizieren, gehören zu einer Kohorte.

→ Neukundengruppe



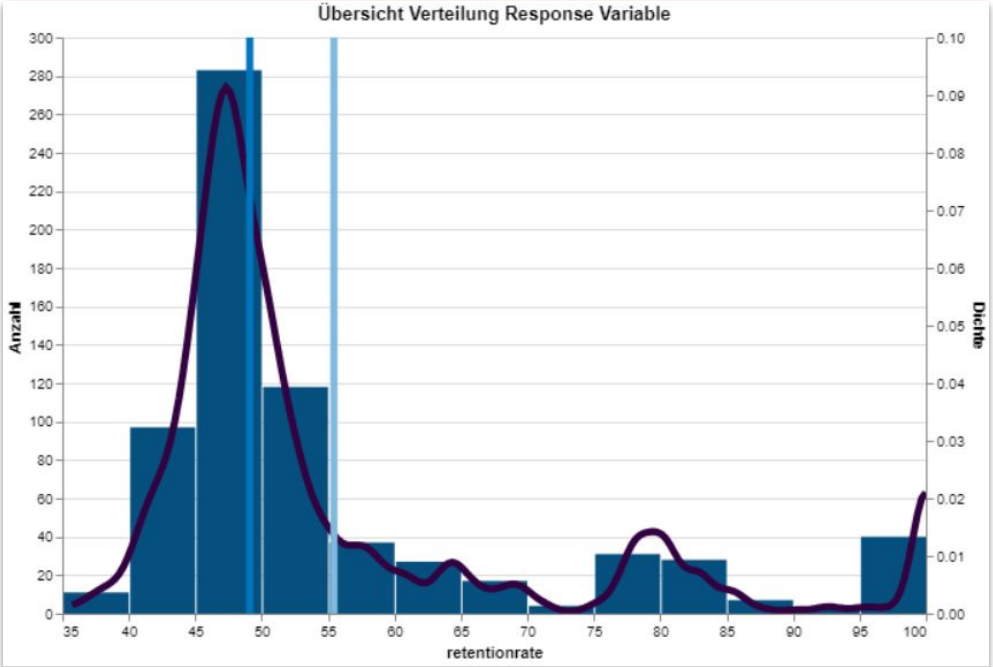
Je früher ein Kunde einer Technologie beitritt, desto höher ist die Wahrscheinlichkeit in eine nachhaltige Retentionphase zu gelangen.

Haben frühere Kohorten also eine bessere Retentionrate?

Datenübersicht

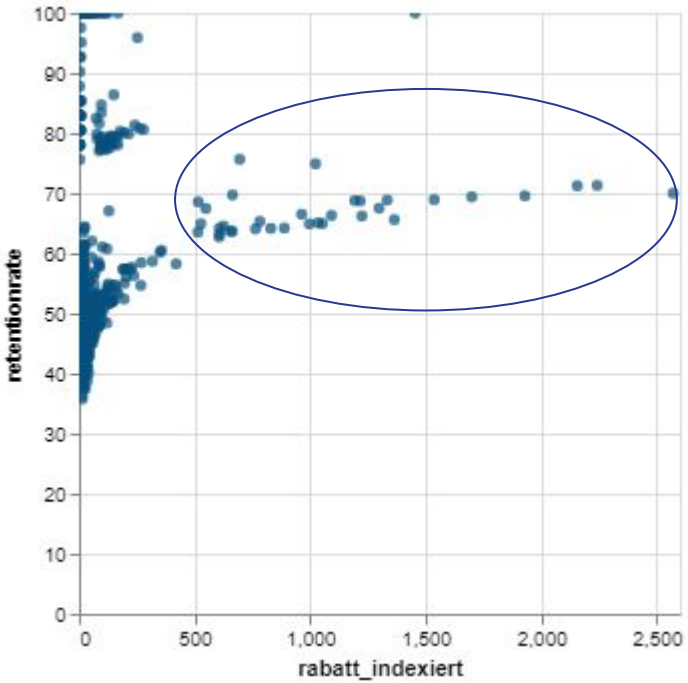
	Name	Format	Type	Role	Description
0	monate_seit_einfuehrung_programm_kohorte	int64	number - discrete	Predictor	Hier wird gezeigt, in welchem Zeitraum seit offizieller Einführung des Programms die Kohorte entstanden ist. Ist der Wert hier 0 so ist die Kohorte entstanden, in dem Monat, in dem auch das Programm eingeführt wurde. Negative Werte resultieren aus Testzeiträumen. Da alle Daten in Monaten erhoben werden, wird diese Saplte als diskret betrachtet.
1	monat	object	ordinal	Predictor	Der Monat sagt aus, in welchem Monat das Einkaufsverhalten einer Kohorte aufgenommen wurde.
2	monat_jahr	object	ordinal	Predictor	Der Jahreswert aus der Spalte "monat".
3	monat_monat	object	ordinal	Predictor	Der Monatswert aus der Spalte "monat".
4	monat_jahreszeit	object	ordinal	Predictor	Die Jahreszeit aus der Spalte "monat".
5	kohorte	object	ordinal	Predictor	Der Monat, an dem die Kohorte entstanden ist.
6	kohorte_jahr	object	ordinal	Predictor	Der Jahreswert aus der Spalte "kohorte".
7	kohorte_monat	object	ordinal	Predictor	Der Monatswert aus der Spalte "kohorte".
8	kohorte_jahreszeit	object	ordinal	Predictor	Die Jahreszeit aus der Spalte "kohorte".
9	erster_monat_kohorte_fg	bool	nominal	Predictor	Hier wird eine Flag gesetzt, wenn der Monat der erste einer Kohorte ist. Hier ist der Wert der Spalte "monat" und der Spalte "kohorte" also gleich. Die Flag könnte für das Modell wichtig sein, da der erste Monat einer Kohorte sehr auffällig im Einkaufsverhalten ist und für das normale Verhalten nicht repräsentativ ist.
10	monate_seit_existenz_kohorte	int64	number - discrete	Predictor	Gibt den Zeitraum in Monaten an, wie lange die Kohorte schon exisitert. Beim Wert 0 ist die Kohorte gerade den ersten Monat aktiv. Da alle Daten in Monaten erhoben werden, wird diese Saplte als diskret betrachtet.
11	kohortengroesse_indexiert	float64	number - continuous	Response	Gibt die Kohortengröße, bzw. die Kundenanzahl, der Kohorte an.
12	identifizierte_kunden_indexiert	float64	number - continuous	Response	Gibt die Anzahl der identifizierten Kunden in dem Monat in der Kohorte an.
13	rabatt_indexiert	float64	number - continuous	Predictor	Gibt den augespielten Rabatt im jeweiligen Monat an die jeweilige Kohorte an.
14	retentionrate	float64	number - continuous	Response	Gibt die Rate an identifizierten Kunden an der Kohortengröße an. Entspricht der Quote aus "identifizierte_kunden_indexiert" und "kohortengroesse_indexiert". Im ersten Monat einer Kohorte (Entstehungsmonat) beträgt der Wert immer 100. Der Wertebereich liegt zwischen 0 und 100.

Einblick Response Variable Retentionrate



	count	mean	std	min	25%	50%	75%	max
retentionrate	703.0	55.442768	15.491696	35.752785	46.394922	49.102845	57.336771	100.0

Zusammenhang zwischen Rabatt und Retention

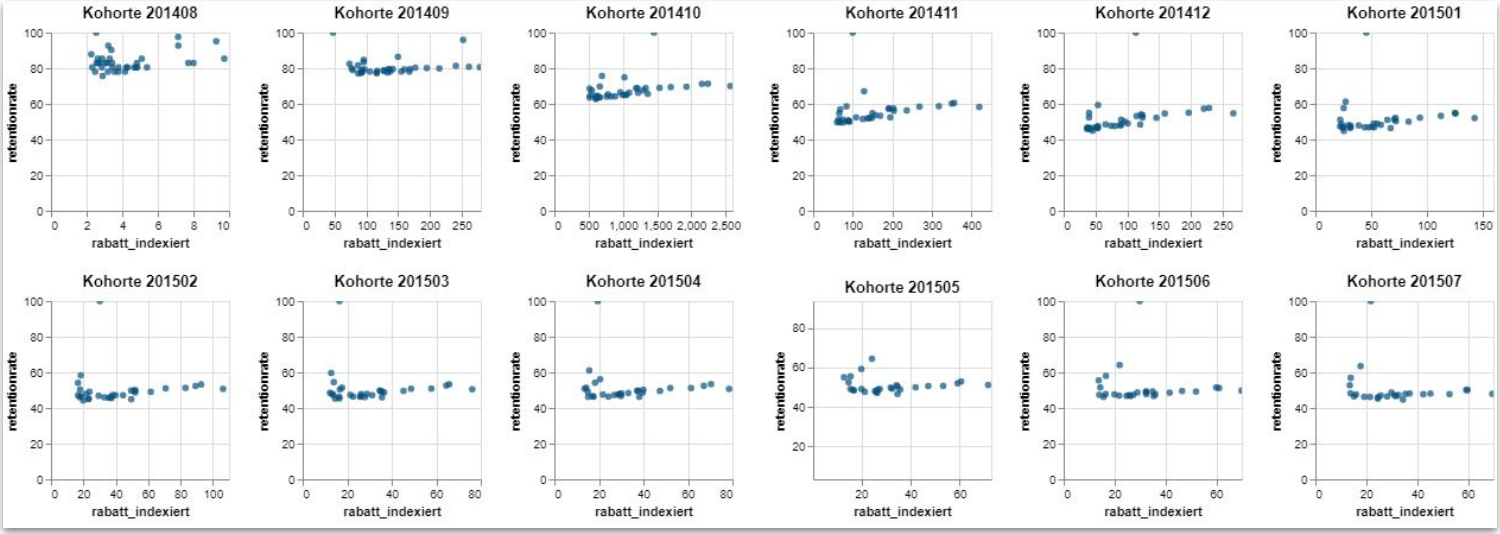


Hohe Rabatte kommen ausschließlich bei der größten Kohorte vor

Gibt es ein Muster je Kohorte?

```python
# Leere Liste, um die Charts zu speichern
charts = []

# Schleife über alle einzigartigen Werte in 'kohorte'
for kohorte in unique_kohorten:
    # Filtert den DataFrame nach der aktuellen Kohorte
    df_filtered = data[data['kohorte'] == kohorte]

    # Erstellt das Diagramm
    chart = alt.Chart(df_filtered).mark_circle(color=color1).encode(
        x=alt.X(x_float),
        y=alt.Y(y_label),
        tooltip=tooltip + [x_float, y_label]
    ).interactive()

    # Fügt eine Überschrift hinzu
    chart = chart.properties(
        title=f'Kohorte {kohorte}',
        width = 150,
        height = 150
    )

    # Fügt das Diagramm der Liste hinzu
    charts.append(chart)

# Erstellt ein Rasterlayout aus den Diagrammen
grid_chart = alt.concat(*charts, columns=6)

grid_chart.display()
```

Zusammenhang zwischen Rabatt und Retention

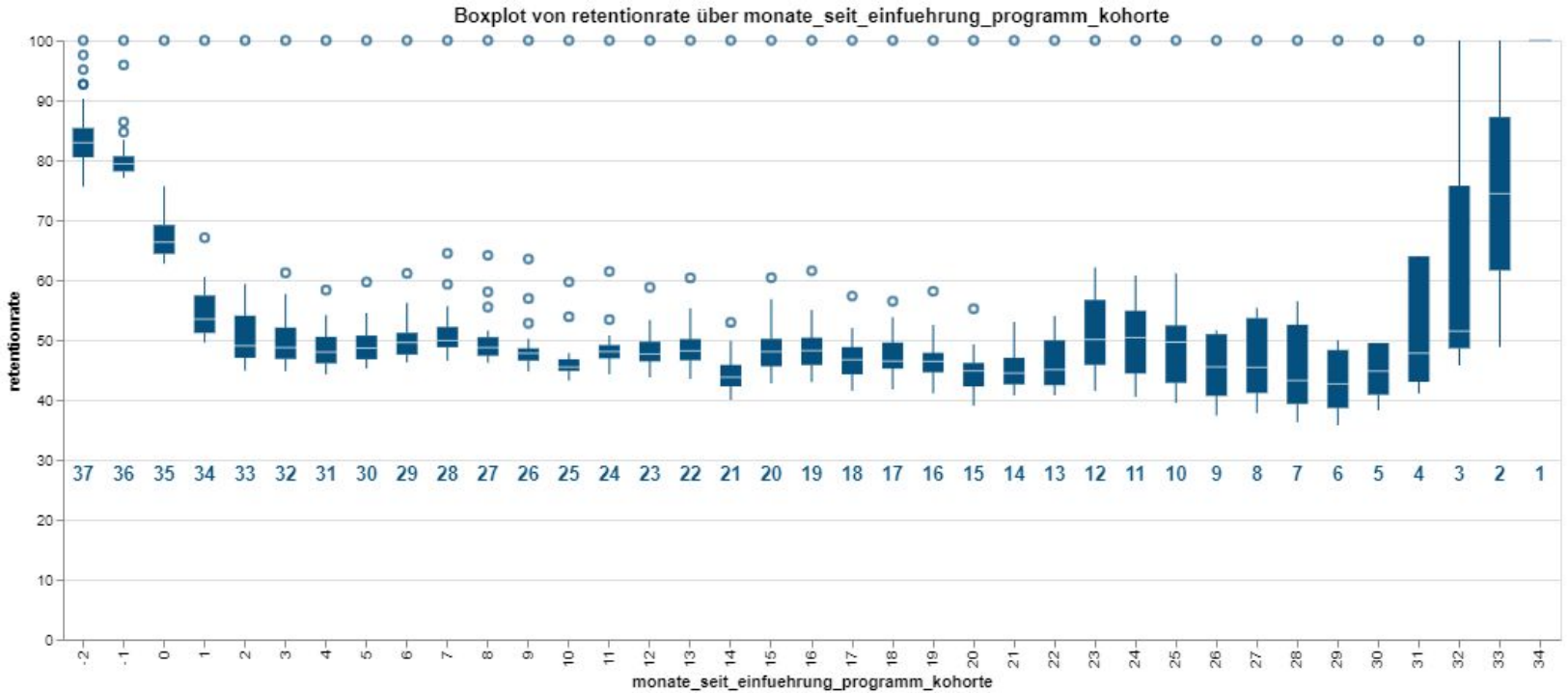


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Kohorte_Zeile1	201408	201409	201410	201411	201412	201501	201502	201503	201504	201505	201506	201507	201508	201509	201510	201511	201512	201601	nan
Korrelation_Zeile1	0.08	0.19	0.65	0.64	0.69	0.42	0.30	0.22	0.12	0.06	0.16	0.02	-0.16	-0.29	-0.36	-0.32	-0.06	-0.14	nan
Kohorte_Zeile2	201602	201603	201604	201605	201606	201607	201608	201609	201610	201611	201612	201701	201702	201703	201704	201705	201706	201707	201708
Korrelation_Zeile2	0.04	-0.05	0.21	0.24	0.46	0.43	0.69	0.80	0.91	0.94	0.83	0.93	0.89	0.94	0.90	1.00	1.00	1.00	nan

Gesamtkorrelation: 0,24

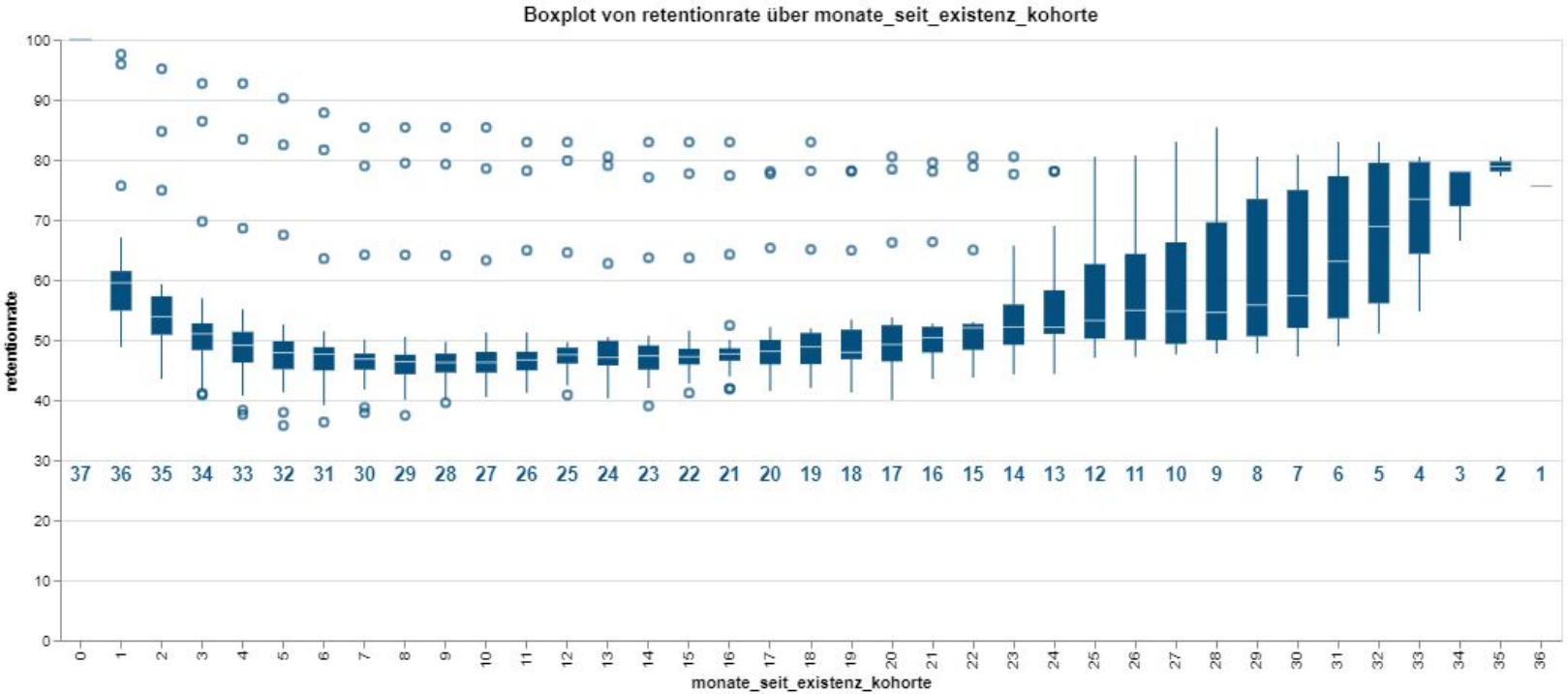
Eignung der Variable als Predictor aufgrund Schwankung, geringer Korrelation und Zweifel an der Validität fraglich

Einfluss Monate seit Einführung Programm



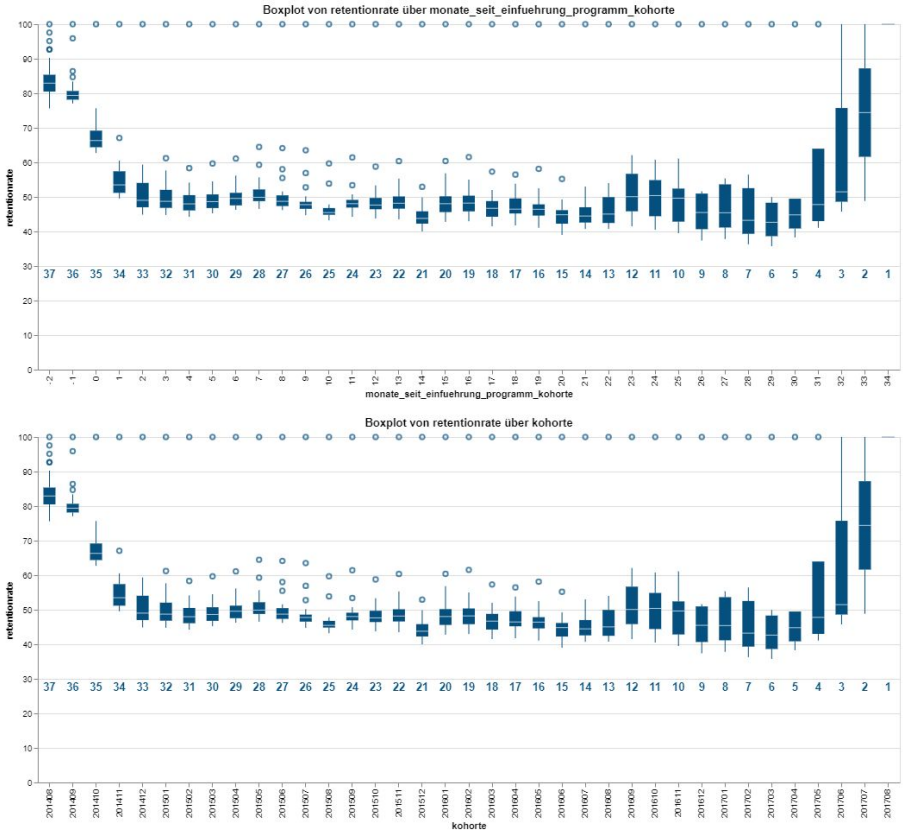
Korrelation: -0,52

Einfluss Monate seit Existenz Kohorte



Korrelation: -0,07

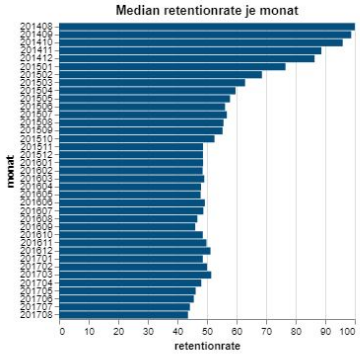
Monate seit Einführung und Kohorte sind gleich!

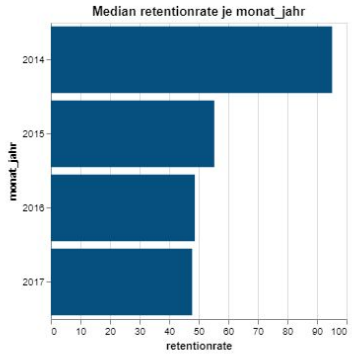


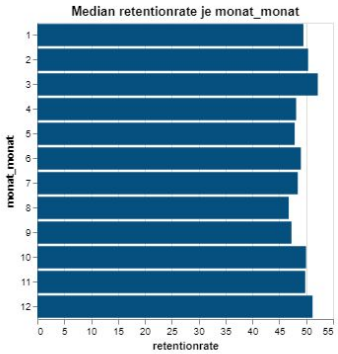
Es dürfen nicht beide Variablen in das Modell inkludiert werden.

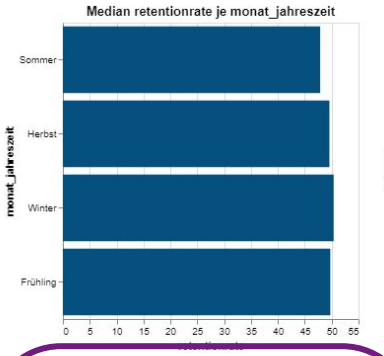
Einzelne Teile der Kohortespalte (Monat und Jahreszeit) können jedoch hilfreich sein, um eventuelle saisonale Effekte abbilden zu können.
Das Jahr zu inkludieren ist nicht sinnig. Dies zeigt das Alter der Kohorte, wofür die Variable Monate seit Einführung Kohorte besser geeignet ist.

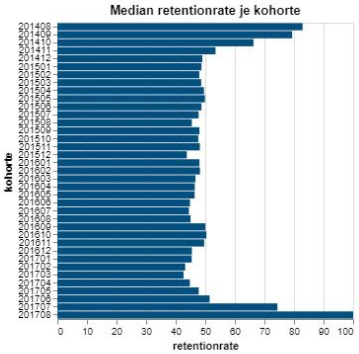
Einblick in kategorische Variablen

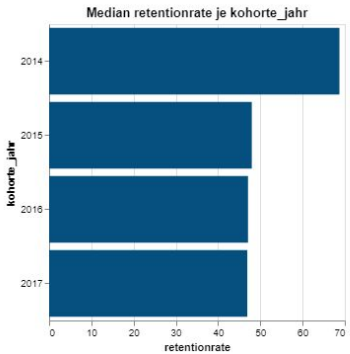


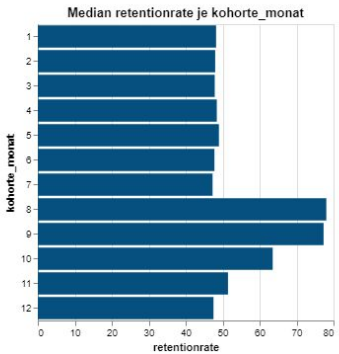


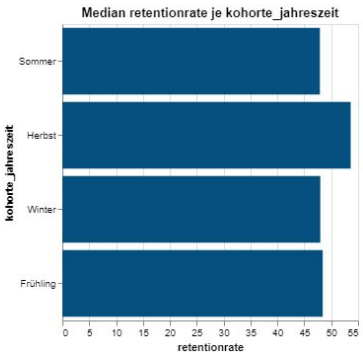


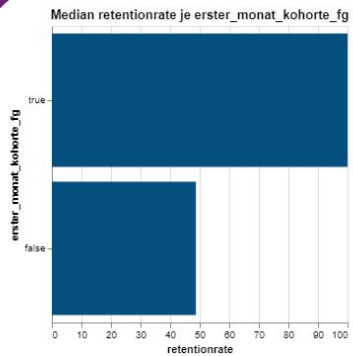












Modell Implikationen

Erwartungshaltung geeignete Features

- Rabatt ist zu volatil für ein gutes Modell
- Erster Monate Kohorte Flag zeigt dem Modell, dass eine Retentionrate von 100 nicht normal ist
- Monate seit Einführung Programm zeigt den negativen Zusammenhang. Je später eine Kohorte, desto schlechter die Retention
- Sowohl die Saisonalität des Monatsverlauf als auch die Entstehung der Kohorte sollten im Modell Berücksichtigung finden

Vorgehensweise Modell

Feature Kombinationen

Es wird einmal ein Modell für jede mögliche Featurekombination erstellt

Daten-transformationen

Aufgrund Rechtsschiefe wird versucht, ob eine logarithmische Transformation die Metriken verbessert.

Ebenfalls wird getestet, ob eine zyklische Kodierung der Monate das Modell verbessert.

Auswahl anhand R^2

Die Top 15 Modelle von allen Versuchen werden anhand des besten R^2 ausgewählt. Die schlussendliche Auswahl erfolgt unter Berücksichtigung R^2, MAE, STD der Kreuzvalidierung und fachliche Sinnhaftigkeit.

Pipeline

```python
def full_pipeline(data, y_label, features, model_name, is_log_transformed=False):
    '''
    Führt den gesamten Prozess der Modellbildung und Bewertung durch, mit optionaler logarithmischer Rücktransformation.

    Args:
        data (pd.DataFrame): Eingabedaten.
        y_label (str): Zielvariable.
        features (list): Feature-Liste.
        model_name (str): Name des Modells.
        is_log_transformed (bool): Gibt an, ob die Zielvariable logarithmisch transformiert wurde.
    '''
    # Preprocessing
    X, y = preprocess_data(data, y_label, features)

    # Splitting
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Training and Validation
    reg, df_scores = train_and_validate_model(X_train, y_train)

    # Evaluation
    metrics, residuals_df = evaluate_model(reg, X_test, y_test)
```

```python
# Vorhersagen
y_pred_transformed = reg.predict(X_test)

# Rücktransformation der Vorhersagen auf die Originalskala
if is_log_transformed:
    y_pred_original = np.exp(y_pred_transformed)  # Rücktransformation der Vorhersagen
    y_test = np.exp(y_test)
else:
    y_pred_original = y_pred_transformed  # Keine Rücktransformation der Vorhersagen

# Residuen auf der Originalskala berechnen
residuals = y_test - y_pred_original  # Residuen nach Rücktransformation (oder nicht)

# Summary Table
summary_table = generate_summary_table(reg, X)

# Speichern der Ergebnisse
save_model(reg, model_name)
save_features(features, model_name)
save_results(y_test, y_pred_original, model_name)  # Ergebnisse auf der Originalskala speichern
save_validation_results(df_scores, model_name)
save_residual_plot(residuals_df, model_name)
save_summary_table(reg, X, model_name)
```

4 Durchgänge der Pipeline

Daten ohne Anpassung → Logarithmische Transformation → Zyklische Kodierung

1. data_transformed[[
   'retentionrate',
   'rabatt_indexiert']] =
   data_transformed[[
   'retentionrate',
   'rabatt_indexiert']]
   .apply(np.log)
2. data_transformed2[[
   'retentionrate']] =
   data_transformed2[[
   'retentionrate']].apply(np.log)

```python
def add_cyclic_features(data, columns, cycle_length=12):
    '''
    Fügt zyklische Kodierungen (Sinus und Cosinus) für
    angegebene Spalten eines DataFrames hinzu.

    Args:
        data (pd.DataFrame): Der DataFrame,
            der die zu kodierenden Spalten enthält.
        columns (list of str): Liste der Spaltennamen,
            die zyklisch kodiert werden sollen.
        cycle_length (int, optional): Die Länge des Zyklus,
            z. B. 12 für Monate im Jahr. Standard ist 12.

    Returns:
        pd.DataFrame: Der DataFrame mit den hinzugefügten
            zyklischen Kodierungen.
    '''
    for column in columns:
        data[f'sin_{column}'] = np.sin(2 * np.pi * (data[column] - 1) / cycle_length)
        data[f'cos_{column}'] = np.cos(2 * np.pi * (data[column] - 1) / cycle_length)
    return data
```

Ergebnisse

Ohne Anpassungen

Beste Featurekombination mit einem **R^2 von 0,704**

Logarithmische Transformation

Beste Featurekombination mit einem **R^2 von 0,699**

Zyklische Kodierung

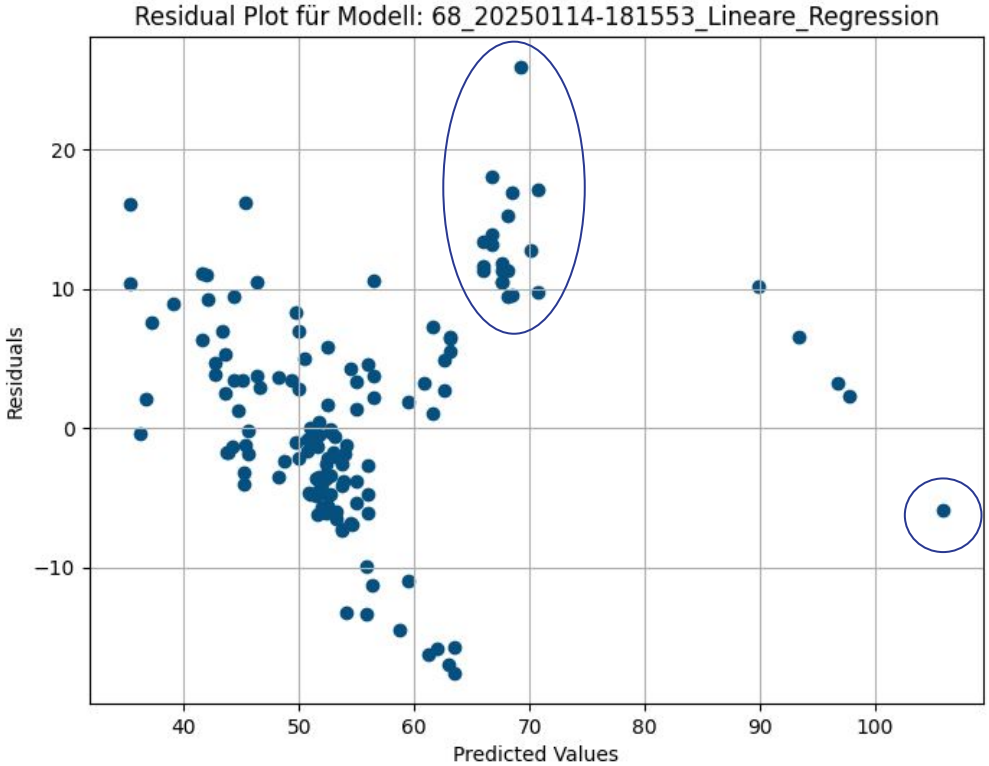
Beste Featurekombination mit einem **R^2 von 0,681**

Modellauswahl

Ausgewählte Features:

- Erster Monat Kohorte Flag
- Monate seit Einführung Programm
- Kohorte_Monat
- Monat_Jahreszeit

Absprung in den Report zur näheren Begründung



	Name	Coefficient
0	Intercept	55.246
1	erster_monat_kohorte_fg	50.907
2	monate_seit_einfuehrung_programm_kohorte	-0.598
3	kohorte_monat_10	7.355
4	kohorte_monat_11	1.282
5	kohorte_monat_12	-1.309
6	kohorte_monat_2	-0.374
7	kohorte_monat_3	-0.055
8	kohorte_monat_4	1.596
9	kohorte_monat_5	3.047
10	kohorte_monat_6	0.941
11	kohorte_monat_7	1.031
12	kohorte_monat_8	13.728
13	kohorte_monat_9	11.801
14	monat_jahreszeit_Herbst	-0.969
15	monat_jahreszeit_Sommer	-1.712
16	monat_jahreszeit_Winter	0.522

- Der erste Monat der Kohorte hat eine Retention von 100. Daher macht der stark positive Koeffizient hier Sinn.
- Bei einem Anstieg der Monate seit Einführung reduziert sich die Retention um -0,598. Das stimmt auch mit der Erwartungshaltung überein, dass Late Adopters eine schlechtere Retention zeigen.
- Entsteht die Kohorte zur Weihnachtszeit, so wirkt sich dies negativ auf die Retention aus
- In der Winterzeit zeigen die Kohorten eine bessere Retention

Fazit

- Modellqualität ist grundsätzlich gut und liefert erwartete Ergebnisse
- MAE mit 6,4 ist sehr hoch bei einem IQR der Daten von 46 bis 57
- Unterschätzung im Bereich 70 und eine Prognose über 100 konnten in der linearen Regression nicht behoben werden
- Weitere Daten wie bspw. Alter der Kohortenmitglieder könnte ein weiterer möglicher Predictor sein
- Time Series Modelle könnten besser mit Saisonalität der Daten umgehen

Backup

Preprocessing

```python
def preprocess_data(data, y_label, features, drop_first=True):
    '''
    Bereitet die Daten für das Modell vor, einschließlich One-Hot-Encoding,
    und vermeidet Multikollinearität durch das Weglassen einer Kategorie pro Feature, wenn es kategorial ist.

    Args:
        data (pd.DataFrame): Der ursprüngliche Datensatz.
        y_label (str): Der Name der Zielvariable.
        features (list): Die Liste der Features.
        drop_first (bool): Gibt an, ob eine Kategorie pro One-Hot-encoded Feature ausgeschlossen werden soll.

    Returns:
        pd.DataFrame, pd.Series: Features (X) und Zielvariable (y).
    '''
    model_data = data[[y_label] + features]
    model_data = pd.get_dummies(model_data, drop_first=drop_first)
    X = model_data.drop(columns=[y_label])
    y = model_data[y_label]
    return X, y
```

Training und Validierung

```python
def train_and_validate_model(X, y, cv=5):
    '''
    Trainiert ein Modell und führt eine Kreuzvalidierung durch.

    Args:
        X (pd.DataFrame): Feature-Daten.
        y (pd.Series): Zielvariable.
        cv (int): Anzahl der Folds für die Kreuzvalidierung.

    Returns:
        LinearRegression, pd.DataFrame: Das trainierte Modell und die Kreuzvalidierungsergebnisse.
    '''
    reg = LinearRegression()
    scores = cross_val_score(reg, X, y, cv=cv, scoring='neg_mean_squared_error') * -1
    df_scores = pd.DataFrame({'MSE': scores})
    df_scores.index += 1
    reg.fit(X, y)
    return reg, df_scores
```

Evaluierung

```python
def evaluate_model(reg, X_test, y_test):
    '''
    Bewertet ein Modell anhand verschiedener Metriken und erstellt Residualdaten.

    Args:
        reg (LinearRegression): Das trainierte Modell.
        X_test (pd.DataFrame): Test-Features.
        y_test (pd.Series): Wahre Zielwerte.

    Returns:
        dict, pd.DataFrame: Metriken und Residualdaten.
    '''
    y_pred = reg.predict(X_test)
    residuals = y_test - y_pred
    metrics = {
        'R_squared': r2_score(y_test, y_pred),
        'MSE': mean_squared_error(y_test, y_pred),
        'RMSE': mean_squared_error(y_test, y_pred, squared=False),
        'MAE': mean_absolute_error(y_test, y_pred)
    }
    residuals_df = pd.DataFrame({
        'True Values': y_test,
        'Predicted Values': y_pred,
        'Residuals': residuals
    })
    return metrics, residuals_df
```

Tabelle für die Koeffizienten

```python
def generate_summary_table(reg, X):
    '''
    Erstellt eine Tabelle mit Intercept und Koeffizienten des Modells.

    Args:
        reg (LinearRegression): Das trainierte Modell.
        X (pd.DataFrame): Die Feature-Daten (einschließlich One-Hot-Encoding).

    Returns:
        pd.DataFrame: Tabelle mit Intercept und Koeffizienten.
    '''
    intercept = pd.DataFrame({
        'Name': ['Intercept'],
        'Coefficient': [reg.intercept_]
    })

    # Verwendet die Spaltennamen von X nach dem One-Hot-Encoding
    slope = pd.DataFrame({
        'Name': X.columns,
        'Coefficient': reg.coef_
    })

    return pd.concat([intercept, slope], ignore_index=True, sort=False).round(3)
```

Featureliste erstellen

```python
def generate_combinations(strings):
    '''
    Generiert alle möglichen Kombinationen von mindestens einem Element aus der gegebenen Liste von Strings.

    Args:
        strings (list): Eine Liste von Strings, die die Variablen repräsentieren.

    Returns:
        list: Eine Liste von Listen, wobei jede Liste eine Kombination von Strings enthält.
    '''
    # Generiere alle möglichen Kombinationen (mindestens eine der Strings)
    combinations = []
    for r in range(1, len(strings) + 1):
        combinations.extend(itertools.combinations(strings, r))

    # Umwandeln in eine Liste von Listen
    return [list(comb) for comb in combinations]
```