



# Avocado

Abschlussbericht Oktober 2024

Brain Hodgson, Mark Fischer, Mathieu Meleux, Patrick Johns, Tingting Yang

# Teilnehmer

## Brian Hodgson:

- die CSV-Datei importiert und festgestellt,
- die importierten Daten überprüft
- die Typen und Statistiken kontrolliert
- ob es Nullwerte und fehlende Werte gibt.
- Aufteilung der Kosten für die Ost- und Westküste
- KNN ( k-nearest neighbors ), Cross Validation, Grid Search

## Mark Fischer:

- Lineare Regression + Boosting
- Feature Engineering und Encoder:
  - harvest\_indicator (mit Mathieu)
  - Überblick, Säuberung und Kontrolle der Codes
  - Saisonale Effekte in der Datenanalyse (siehe Boxplot San Francisco)
- Datenvorbereitung:
  - Unterteilung der Datensätze in verschiedene homogene Teilmengen (siehe Jupiter-Notebooks bzw. Pickle-Dateien)
  - Bereinigung der jupyter-notebooks (mit Patrick)
- Bericht:
  - Datenvorbereitung
  - Unterteilung der Daten in homogene Teilmengen
  - bestes Testergebnis
- Organisation: Gruppen-/Aufgabenteilung, Zeitplanung, allgemeine Dokumentation

## Mathieu Meleux:

- Aktives Mitdenken und Mitentscheiden
- Date Encoder (Vorlage)
- MLPRegressor
- Bericht: Auswahl der Algorithmen
- Bericht: GridSearch

## Patrick Johns:

- Datenanalyse (Source: avo\_data\_plots\_v01.ipynb) mit grafischer Darstellung (außer saisonale Effekte von Mark Fischer).
- Erste lineare Regression und Dashboard.
- Feature Engineering und Encoder:
  - Regionale Features (dist\_to\_california, dist\_to\_mexico)
  - Zeit-Features (month\_sin, month\_cos)
  - Korrelationen (lag\_7, lag\_30, moving\_avg\_30, inverse-volume).
  - Custom TrainTestSplit

- Support Vector Regression (SVR) inklusive Bagging (Source: PJ\_SV\*.py)
- SVR Pipeline mit PCA (Source: PJ\_SVR\_PIPE\_V03)

### Tinging Yang:

- DecissionTree + RandomForest
- Gesamtplanung des Projekts
  - Zusammenfassung der Reihenfolge und Möglichkeiten der Feature Engineering
  - Empfehlungen für die Auswahl von Modellen und Hinweise auf mögliche Probleme mit einigen Algorithmen
  - Zusammenfassung des Prozesses des Modelltrainings
  - Vorschlagen und Vorhersage, welcher Algorithmus bessere Ergebnisse liefern wird
  - Weisen Sie darauf hin, dass Sie beim Training eines Modells nicht nur versuchen sollten, den Fehler zu reduzieren, sondern auch auf die Kontrolle der fitting achten sollten
  - Vorschlag durch Visualisierung zur Ermittlung eines good fitting Modells für einen Parameter
- Feature Engineering:
  - LebelEncoder
  - Selbst geschriebene OneHotEncoder
  - Überprüfung der Korrektheit und Reihenfolgen des Code (Train-Test-Split und Scaler)
- Bericht
  - Struktur des Teils der Anwenudng verschiedener Algorithmen
  - CrossValidate + Ensemble Methode
  - Visualisierung der Ergebnisse

## Aufgabenstellung

### Modell für Lebensmittelhändler

Das Ziel dieses Projektes ist die Entwicklung eines Vorhersagemodells zur Bestimmung der Avocado-Preise in verschiedenen geografischen Regionen. Dabei liegt der Fokus auf einem Lebensmittelhändler, der die regionalen Preistreiber verstehen und optimieren möchte, um seine Einkaufs- und Verkaufsstrategien zu verbessern.

Durch die Berücksichtigung von Faktoren wie regionale Nachfrage, Transportkosten, Saisonalität und Kaufkraft sollen Preisschwankungen in unterschiedlichen Regionen modelliert und vorhergesagt werden. Die Ergebnisse dieser Arbeit bieten die Grundlage, um Preise optimal zu steuern, Lagerbestände effizient zu verwalten und wettbewerbsfähig zu bleiben.

**Ziel:** Die Vorhersage regionaler Preisentwicklungen ermöglicht es dem Lebensmittelhändler, auf Grundlage von Daten fundierte Entscheidungen zu treffen und seine Marge zu maximieren.

### Arbeitshypothese

Die Preise von Avocados in den USA werden durch eine Kombination aus regionalen, saisonalen, logistischen, gruppenbezogenen und produktionsbezogenen Faktoren beeinflusst. Regionen, die näher an den Anbaugebieten liegen, wie Kalifornien, haben tendenziell niedrigere Preise, während entferntere Regionen wie die Ostküste höhere Preise aufgrund von Transportkosten aufweisen. Saisonale Schwankungen, insbesondere im Winter, sowie die Abhängigkeit von Importen führen zu Preisspitzen. Zudem unterscheiden sich die Preise je nach Fruchtgröße, wobei größere Avocados (PLU 4770) teurer sind als kleinere (PLU 4046). Schließlich haben biologische Avocados oft höhere Preise als konventionelle, da sie teurer in der Produktion und im Vertrieb sind.

### Machine Learning-Verfahren und Scoring

Da dieses Projekt ein nicht lineares Regressionsproblem mit hohem Datenrauschen ist und eine genaue Kontrolle großer Fehler erfordert, ist MSE gut geeignet. Da RMSE jedoch leichter verständlich ist, haben wir es als finales Beurteilungskriterium gewählt. Um die Qualität des Modells zu bewerten, verwenden wir den **Root Mean Squared Error (RMSE)** als Maßstab für die Genauigkeit. Ein Ziel-RMSE von etwa 0.1 wird angestrebt, um eine gute Modellanpassung zu gewährleisten. Da die Daten viele Einflussfaktoren haben, achten wir darauf, dass das Modell weder **Overfitting** (zu stark an die Trainingsdaten angepasst) noch **Underfitting** (zu einfache Modellstruktur) zeigt. Als Machine Learning-Verfahren wählen wir **Regression**, da wir kontinuierliche Preisvorhersagen basierend auf verschiedenen Einflussfaktoren wie Volumen, Region und Jahreszeit anstreben.

## Avocado und Haltbarkeit

Die Avocado hat sich in den letzten Jahren zur Trendfrucht entwickelt, insbesondere in den USA, wo sie wegen ihrer gesundheitlichen Vorteile und Vielseitigkeit in der Küche sehr beliebt ist. Hauptanbaugebiete in den USA sind **Kalifornien** und **Florida**, aber ein Großteil der Avocados wird auch aus **Mexiko** importiert. Aufgrund ihrer **begrenzten Haltbarkeit** und der starken Abhängigkeit von saisonalen Erntezeiten, führen Angebotsschwankungen und Wetterbedingungen zu erheblichen saisonalen Preisschwankungen.

- Unreife Avocados: 4-7 Tage bei Raumtemperatur.
- Reife Avocados: 2-3 Tage im Kühlschrank.
- Aufgeschnittene Avocados: 1-2 Tage im Kühlschrank (mit Zitronensaft abdecken).
- Gefrorene Avocados (püriert): 4-6 Monate.
- Kommerzielle Lagerung: Bis zu 4 Wochen unter kontrollierten Bedingungen (Kühlcontainer).

## Preisabschätzung PLU-Größen

Im Datensatz sind keine Einzelpreise für die verschiedenen Avocado-Größen (PLU 4046, 4225, 4770). Um die Einzelpreise zu berechnen, verwenden wir die folgenden Schritte:

PLU-Größen und Kartoninhalt:

Die PLU-Nummern unterscheiden sich durch die Anzahl der Avocados pro standardisiertem 25-Pfund-Karton:

- PLU 4046: Kleine Avocados, 60 Stück pro Karton.
- PLU 4225: Mittlere Avocados, 40-48 Stück pro Karton.
- PLU 4770: Große Avocados, 36 Stück pro Karton.

Preisformel:

Der Preis pro Avocado ist umgekehrt proportional zur Anzahl der Avocados im Karton.

Daraus ergibt sich folgende Näherungsformel:

$$\text{Preis} = \text{Durchschnittspreis} \times (\text{Standardkarton-Größe} / \text{PLU-Größe})$$

- Durchschnittspreis: Der durchschnittliche Preis pro Avocado (z.B. 1,50 USD).
- Standardkarton-Größe: Eine Referenzgröße (hier die PLU 4225 mit 48 Avocados).
- PLU-Größe: Die Anzahl der Avocados pro Karton für die jeweilige PLU.

Beispiele zur Preisberechnung:

$$\text{Preis\_4046} = 1.50 \times (48 / 60) = 1.20 \text{ USD}$$

$$\text{Preis\_4225} = 1.50 \times (48 / 48) = 1.50 \text{ USD}$$

$$\text{Preis\_4770} = 1.50 \times (48 / 36) = 2.00 \text{ USD}$$

Mit dieser Näherungsformel können die Preise für die verschiedenen PLU-Nummern basierend auf den Fruchtgrößen berechnet werden. Größere Avocados (PLU 4770) sind teurer, während kleinere (PLU 4046) weniger kosten.

## Roadmap zur Untersuchung

- **Datenvorbereitung und Explorative Datenanalyse (EDA)**

Eine Datenbereinigung wurde durchgeführt. Anschließend wurden die numerischen Daten skaliert und die kategorialen Variablen kodiert, um sie für das Modell nutzbar zu machen. Erste Analysen konzentrierten sich auf das Erkennen von Zusammenhängen und statistischen Kennzahlen im Datensatz, wie Preis, Volumen und geografische Regionen.

- **Feature Engineering und Split-Strategien**

Die Daten von 2015 bis 2020 werden analysiert, wobei 2015-2019 für das Training und 2020 für das Testen genutzt werden. Um saisonale Muster zu erfassen, werden Zeitfeatures wie date\_number, month\_sin, und month\_cos eingeführt. Neue Features wie lag\_7, lag\_30 und moving\_avg\_30 berücksichtigen vergangene Preisentwicklungen und glätten kurzfristige Schwankungen. Das inverse Volumen wird hinzugefügt, um die Preis-Volumen-Beziehung zu modellieren. Zur Verbesserung der Modellleistung werden Regionen in West- und Ostküste kodiert, und die Entfernung zu Kalifornien und Mexiko sowie ein harvest\_indicator ergänzen die geografischen Informationen.

- **Modellauswahl und -training**

Es werden Regressionen und Tests mit verschiedenen Algorithmen durchgeführt, darunter DecisionTree, SupportVectorRegressor, KNN, MLPRegressor und Lineare Regression. Das Ziel ist es, ein Modell zu entwickeln, das eine ausgewogene Balance zwischen Overfitting und Underfitting erreicht.

- **Modellbewertung und Optimierung**

Die Modelle werden anhand von RMSE und R<sup>2</sup>-Score bewertet. Eine Überprüfung auf Overfitting und Underfitting erfolgt mittels Cross-Validation. Zusätzliche Feature-Transformationen und Parameteroptimierung, etwa durch GridSearchCV, werden zur weiteren Verbesserung der Algorithmen eingesetzt.

- **Finale Analyse und Zusammenfassung**

Die Vorhersageergebnisse werden für verschiedene Regionen und Avocado-Typen (Bio vs. konventionell) verglichen. Die Ergebnisse werden im Hinblick auf regionale Unterschiede, saisonale Effekte und die Einflussfaktoren auf die Preisentwicklung interpretiert.

- **Ausblick**

Identifikation von Optimierungsmöglichkeiten, etwa durch den Einsatz komplexerer Algorithmen oder eine detailliertere Segmentierung der Daten nach West/Ost und Bio/Konventionell.

# Beschreibung der Daten

## Datenquelle

- Wie leicht ist die Datenbeschaffung?

Die Datenbeschaffung für Avocado-bezogene Einblicke scheint moderat zugänglich zu sein, insbesondere wenn man strukturierte Studien und Verkaufsberichte als Datenquellen verwendet.

Laut den Berichten über **Bagged Avocados** und **Seasonality Avocados** wurden die Daten aus einer Kombination von Quellen gewonnen, darunter:

### 1. Verkaufsdaten des Einzelhandels:

Diese Daten umfassen den Umsatz in Dollar, die Verkaufseinheiten, den durchschnittlichen Verkaufspreis (ASP), die Verteilung und die Verkaufsdynamik von Avocados. Sie werden aus verschiedenen Einzelhandelskanälen (wie Supermärkte, Massenhändler, Clubgeschäfte usw.) gesammelt und analysiert.

### 2. Haushaltspaneldaten:

Diese Daten stammen aus detaillierten Analysen des Verbraucherverhaltens und zeigen, wie oft Konsumenten Avocados kaufen, wie viel sie ausgeben und welche Größen oder Verpackungen sie bevorzugen.

### 3. Versanddaten:

Der Bericht über die Saisonalität von Avocados erwähnt beispielsweise die Verfügbarkeit von Versanddaten, die das Versandvolumen, saisonale Trends und länderbezogene Daten wie Volumen aus Mexiko, Kalifornien und Peru abdecken.

- Größe des Datensatzes (Zeilen, Spalten, Gigabyte ...)

Der Datensatz enthält 33.045 Zeilen und 13 Spalten und ist etwa 0,0087 Gigabyte groß (rund 8,7 Megabyte).

## Daten Kennzahlen

- Maximum, Minimum, Mean

Statistik	average_price	total_volume
Count	33,045	33,045
Mean	1.38	968,399.7
Std	0.38	3,934,533
Min	0.44	84.56
25%	1.10	15,118.95
50%	1.35	129,117
75%	1.62	505,828.5
Max	3.25	63,716,140

- gibt es fehlende Werte? Passen die Größenordnungen zusammen?

Es gibt keine fehlende Werte und Die Größenordnungen der Daten scheinen also insgesamt zusammenzupassen

- Datentypen:

	Spalte	Typ
0	date	Datum(ns)
1	average_price	float64
2	total_volume	float64
3	4046	float64
4	4225	float64
5	4770	float64
6	total_bags	float64
7	small_bags	float64
8	large_bags	float64
9	xlarge_bags	float64
10	type	object
11	year	int64
12	geography	object

## 1.4 Datenvorbereitung und Analyse (EDA)

In diesem Kapitel beschreiben wir die Schritte der Datenvorbereitung und der Explorativen Datenanalyse (EDA), die vor dem Training der Machine-Learning-Modelle durchgeführt wurden. Diese Schritte umfassen insbesondere die Behandlung fehlender Werte, die Skalierung der Daten, sowie die Kodierung kategorialer Variablen.

### Explorativen Datenanalyse

In diesem Abschnitt werden Hinweise sowie Maßnahmen zum Feature-Engineering erarbeitet, um die Analyse zu optimieren. Quelle siehe `avo_data_plots_v01.ipynb`

In dem Datensatz sind folgende Felder enthalten:

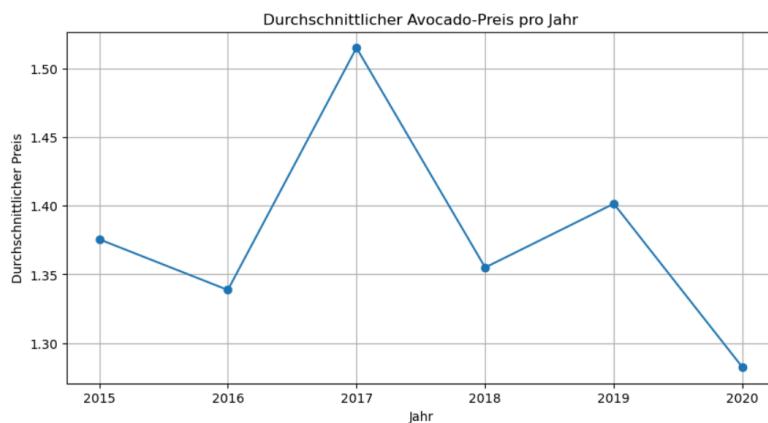
- **date**: Das Datum der Erfassung, an dem die Datenpunkte aufgezeichnet wurden.
- **average\_price**: Der durchschnittliche Verkaufspreis der Avocados in US-Dollar.
- **total\_volume**: Die Gesamtmenge der verkauften Avocados in Pfund.
- **4046**: Verkaufsvolumen für Avocados mit der PLU-Nummer 4046 (kleinere Avocados).
- **4225**: Verkaufsvolumen für Avocados mit der PLU-Nummer 4225 (mittlere Avocados).
- **4770**: Verkaufsvolumen für Avocados mit der PLU-Nummer 4770 (größere Avocados).
- **total\_bags**: Die Anzahl der verkauften Avocado-Taschen.
- **small\_bags**: Anzahl der kleinen Avocado-Taschen.
- **large\_bags**: Anzahl der großen Avocado-Taschen.
- **xlarge\_bags**: Anzahl der extra großen Avocado-Taschen.
- **type**: Art der Avocados, entweder konventionell oder biologisch.
- **year**: Das Jahr, in dem die Daten erfasst wurden.
- **geography**: Geographische Region (z.B. Stadt oder Gebiet), in der die Avocados verkauft wurden.

## Zeitreihe

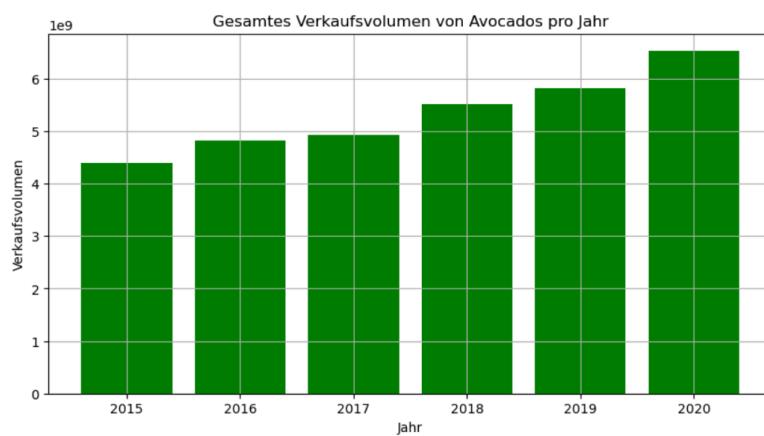
Die Analyse der Zeitreihe ergab die Jahre [2015, 2016, 2017, 2018, 2019, 2020] für einen möglichen Time-Based Train-Test Split (siehe Code).

→ Wir werden die Daten von 2015 bis 2019 als Trainingsdaten und die von 2020 als Testdaten verwenden.

**Durchschnittlicher Preis pro Jahr:** Der Preis der Avocados zeigt über die Jahre Schwankungen. In einigen Jahren sind deutliche Preisanstiege zu erkennen. Es wäre interessant, die möglichen Ursachen für diese Schwankungen zu analysieren, wie etwa saisonale Einflüsse oder Veränderungen in der Nachfrage.

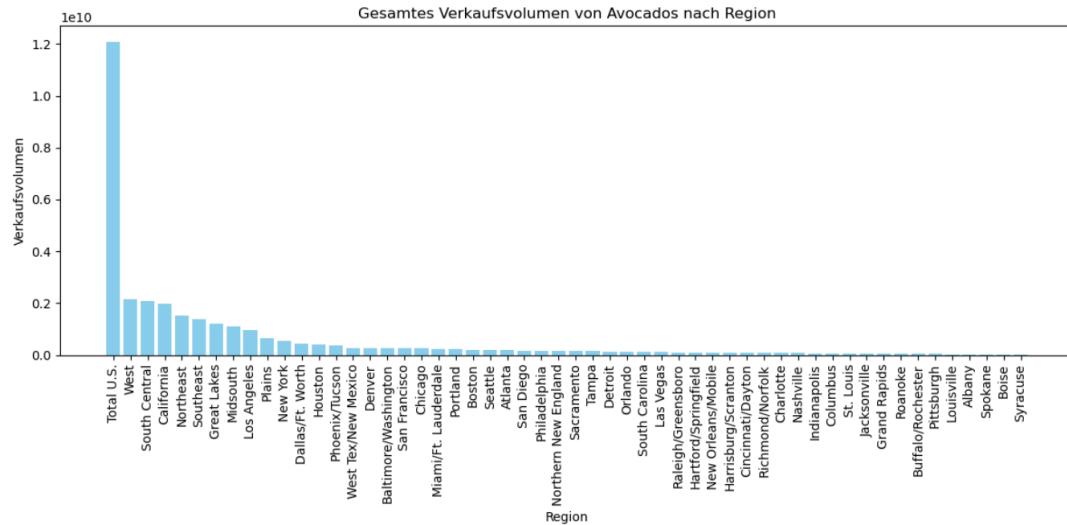


**Gesamtes Verkaufsvolumen pro Jahr:** Hier wird die Entwicklung des Verkaufsvolumens über die Jahre dargestellt. Es sind deutliche Veränderungen erkennbar, wobei das Volumen von Jahr zu Jahr steigt. Dies könnte auf Marktdynamiken, veränderte Anbaubedingungen oder sich wandelnde Verbraucherpräferenzen zurückzuführen sein.



## Geographie

**Geographie im Kontext des Volumens:** Die Spalte 'geography' im Datensatz enthält verschiedene Hierarchieebenen von Regionen, wie z.B. einzelne Staaten, regionale Märkte oder größere Gebiete (z.B. 'Total U.S.'). Um dies zu untersuchen, können wir die einzigartigen Werte der 'geography'-Spalte genauer analysieren.

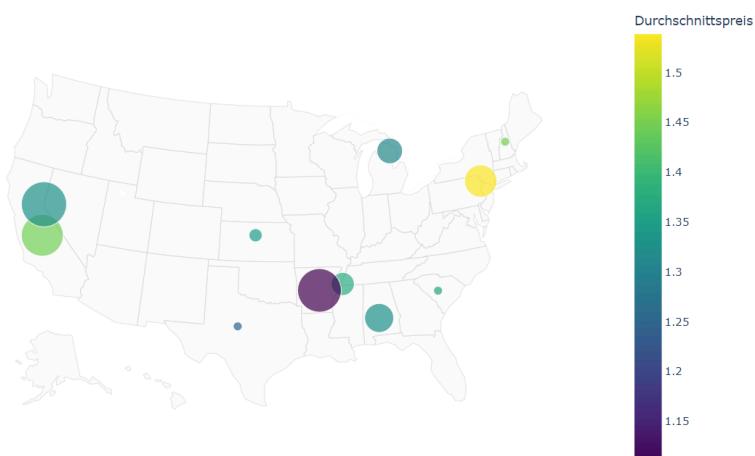


Für die weitere Analyse werden Level eingeführt:

- Level 0: Für gesamt-nationale Daten (z.B. 'Total U.S.').
- Level 1: Für größere Regionen oder Bundesstaaten (z.B. 'California', 'Great Lakes').
- Level 2: Für Städte oder Metropolregionen (z.B. 'Albany', 'Chicago')."

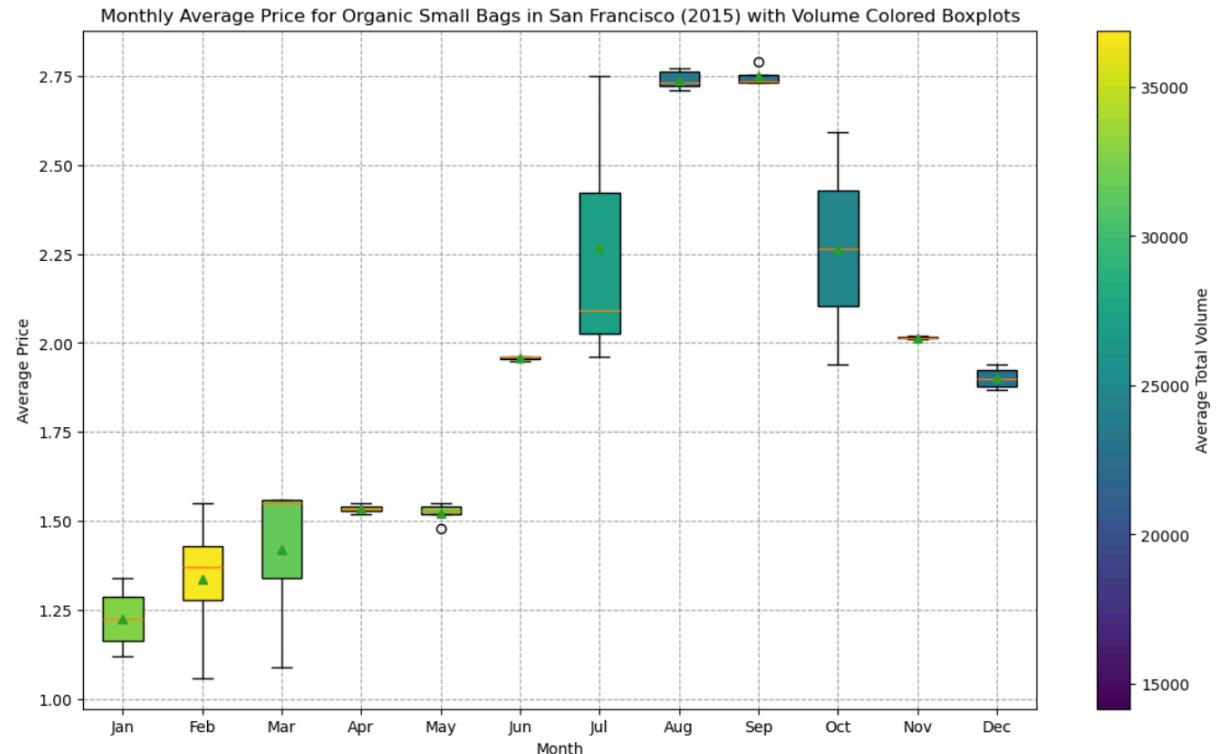
In den größeren Regionen oder Bundesstaaten (Level 1) zeigt sich ein Preisgefälle in zwei Richtungen. Sowohl in Kalifornien als auch im Nordosten sind die Preise tendenziell höher, was auf die höhere Kaufkraft und das Konsumverhalten (z.B. der Fokus auf Superfoods und gesunde Ernährung) zurückzuführen sein könnte. Zudem vermuten wir, dass die Preise in Anbaugebieten wie Kalifornien und Mexiko während der Erntesaison ebenfalls eine entscheidende Rolle spielen.

Durchschnittspreis und Verkaufsvolumen von Avocados nach Level 1 Regionen



## Saisonale Effekte

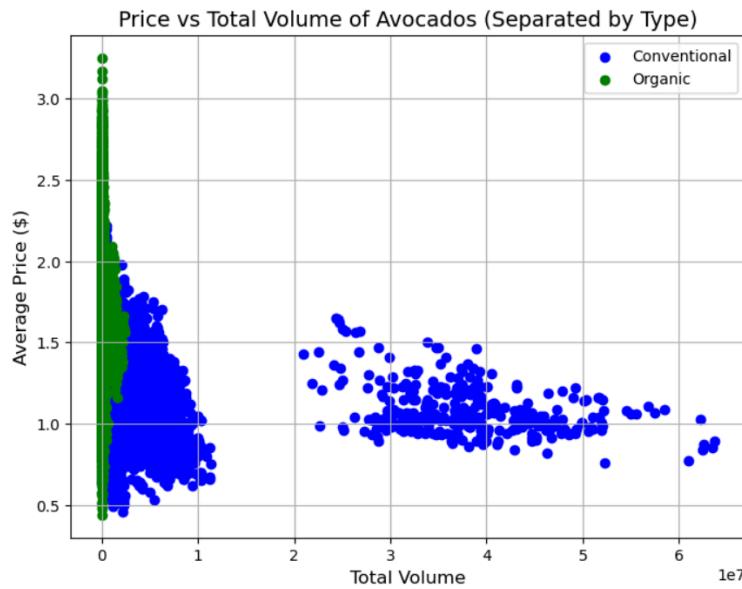
Eine Detailanalyse von Beispielen wie **San Francisco** in Kalifornien zeigt, dass während der Erntephasen die Preise niedrig und das Angebot hoch sind. In den Wachstumsphasen hingegen steigen die Preise, während das verfügbare Volumen sinkt.



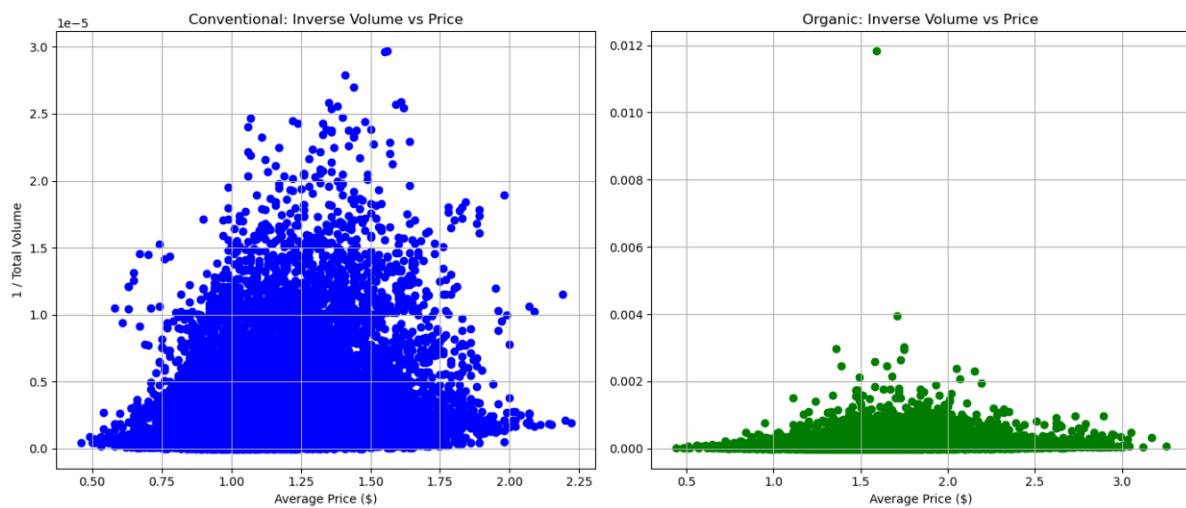
Wir werden Anzahl der Tage seit dem frühesten Datum `date_number` und zyklische Zeiten `month_sin`, `month_cos` nutzen, um kontinuierliche Muster zu erfassen. Zusätzlich wird ein `harvest_indicator` eingeführt, um zwischen Ernte- und Wachstumszeiten zu unterscheiden.

## Bio oder konventionell Herstellung

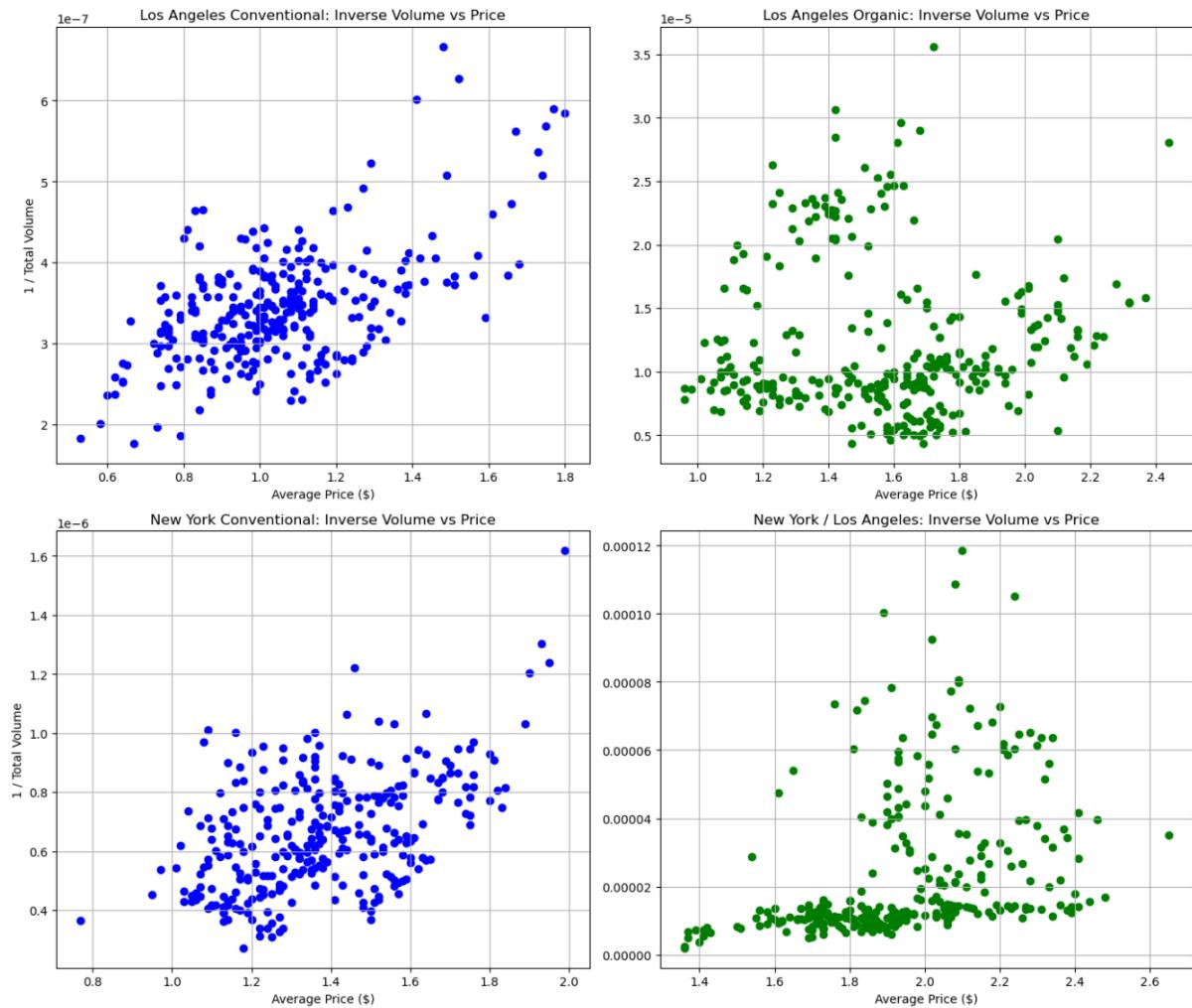
Ein weiterer Faktor, der den Preis von Avocados beeinflusst, ist der Typ: Bio oder konventionell. Die konventionelle Produktion überwiegt, während Bio-Produkte tendenziell teurer sind. Ein Hinweis zur Punktewolke: Die Volumen-Unterschiede in den Daten hängen lediglich damit zusammen, dass größere Regionen mehr Volumen aufweisen als einzelne Städte.



Wir trennen den Datensatz in Bio- und konventionelle Produkte und untersuchen erneut mögliche Korrelationen, wie sie bereits in San Francisco festgestellt wurden. Dabei prüfen wir, ob es einen Zusammenhang zwischen Preis und Volumen gibt. Eine Gegenüberstellung des inversen Volumens und des Preises aller Datensätze zeigt eine Normalverteilung.



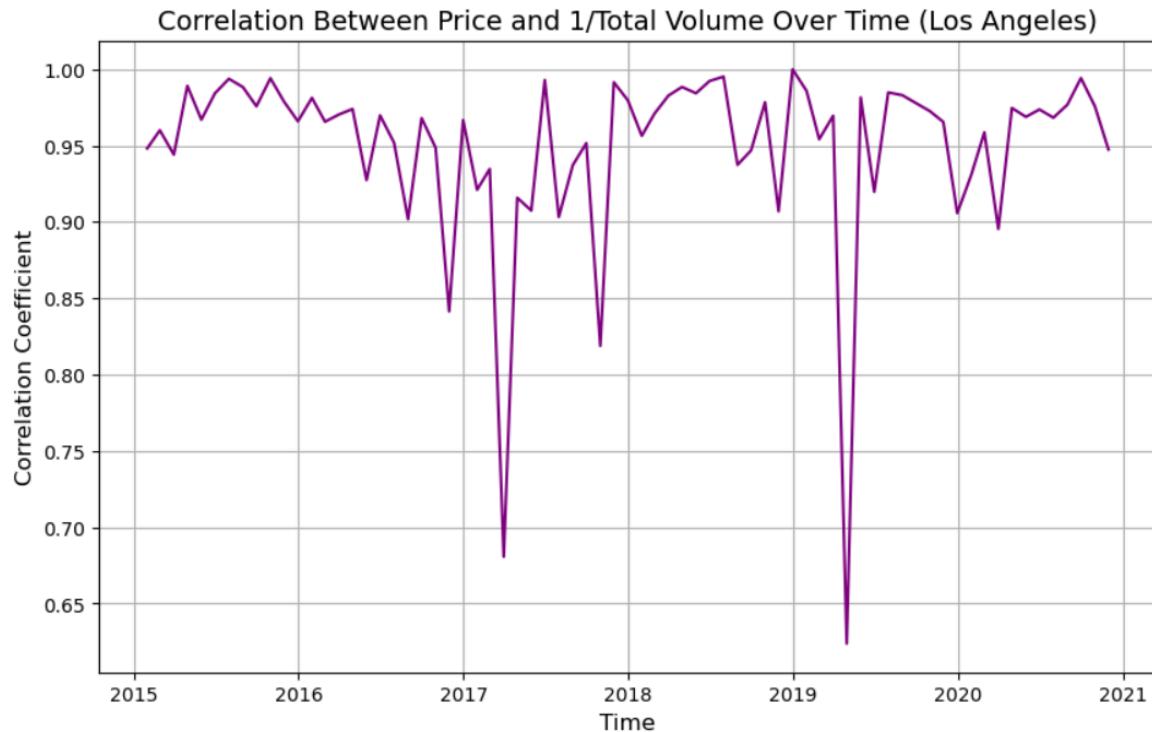
Für zwei große Städte wie Los Angeles (Anbaugebiet) und New York (hohe Kaufkraft) gilt dieser Zusammenhang jedoch nicht mehr.



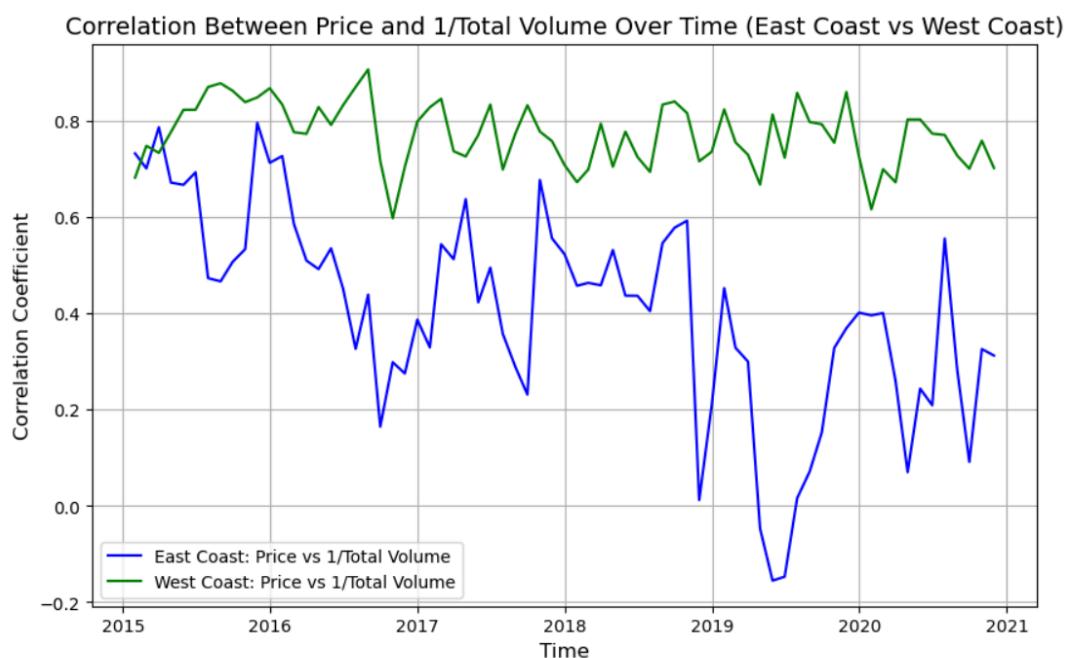
Wir werden die Datensätze entweder entlang vom Typ trennen oder einen Encode nutzen.

## Korrelation

Betrachtet man diese Korrelation in einer Zeitreihe von Los Angeles, bestätigt sich, was wir bereits vermutet haben: Die Nähe zum Anbaugebiet führt zu einer sehr starken Korrelation zwischen dem inversen Volumen und dem Preis.



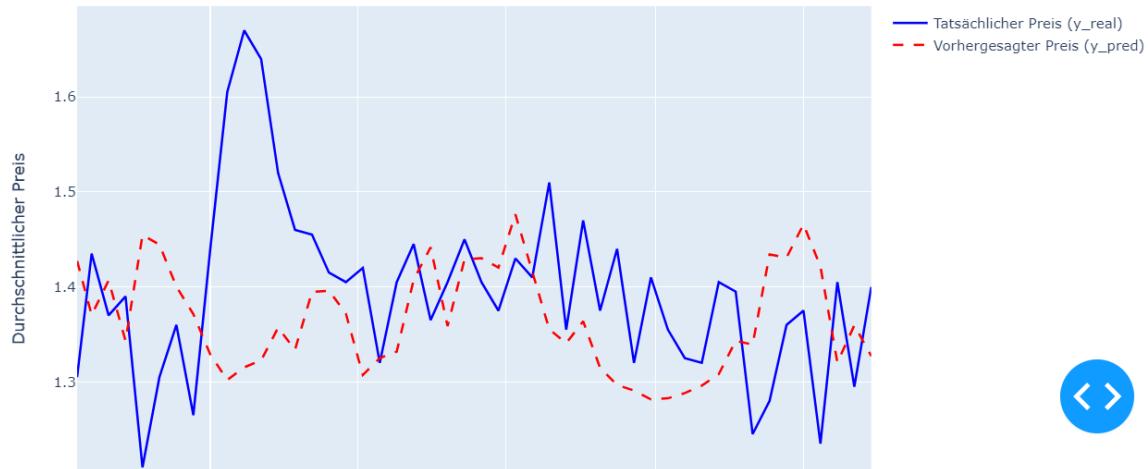
Gleichtes gilt auch für die Westküste ('California', 'Portland', 'Seattle', 'Los Angeles', 'San Francisco') aber nicht für die Ostküste ('New York', 'Boston', 'Miami', 'Philadelphia', 'Baltimore/Washington', 'Atlanta', 'Tampa', 'Orlando', 'Hartford/Springfield', 'Raleigh/Greensboro', 'Richmond/Norfolk'), die wir für unsere Vorhersagen nutzen werden.



Eine erste schnelle Überprüfung des gesamten Datensatzes mittels Linearer Regression unter Verwendung eines OneHotEncoders für die geografischen Daten und eines StandardScalers ergab einen Mean Squared Error von 0,079 und einen R<sup>2</sup>-Score von 0,258. Wir werden daher im weiteren Verlauf dieser Analyse diverse Anpassungen an den Daten vornehmen. Interessant ist zudem, dass in einem nachfolgenden Dashboard erkennbar wird, wo die Vorhersagen teilweise hinter den tatsächlichen Werten zurückbleiben.

Albany  Atlanta  Baltimore/Washington  Boise  Boston  Buffalo/Rochester  California  Charlotte  Chicago  
 Cincinnati/Dayton  Columbus  Dallas/Ft. Worth  Denver  Detroit  Grand Rapids  Great Lakes  Harrisburg/Scranton  
 Hartford/Springfield  Houston  Indianapolis  Jacksonville  Las Vegas  Los Angeles  Louisville  Miami/Ft. Lauderdale  
 Midsouth  Nashville  New Orleans/Mobile  New York  Northeast  Northern New England  Orlando  Philadelphia  
 Phoenix/Tucson  Pittsburgh  Plains  Portland  Raleigh/Greensboro  Richmond/Norfolk  Roanoke  Sacramento  San Diego  
 San Francisco  Seattle  South Carolina  South Central  Southeast  Spokane  St. Louis  Syracuse  Tampa  Total U.S.  West  
 West Tex/New Mexico

Tatsächlicher vs. vorhergesagter Preis im Zeitverlauf



Wir werden daher `lag_7`, `lag_30` einführen. Diese Zeitverzögerungen helfen dabei, zeitlich zurückliegende Datenpunkte (z.B. eine Woche oder einen Monat vorher) einzubeziehen, was für Algorithmen wie Regressionsmodelle wichtig ist, um zeitabhängige Muster zu erkennen. Der `moving_avg_30` gleitende 30-Tage-Durchschnitt hilft, kurzfristige Schwankungen zu glätten und Trends über einen längeren Zeitraum für Algorithmen wie Entscheidungsbäume oder neuronale Netze besser nutzbar zu machen.

## Mögliche Korrelation von total\_volume mit mehreren Spalten

Bei der Untersuchung der Spalten:

- total\_volume
- 4046, 4225, 4770
- total\_bags
- small\_bags, large\_bags, xlarge\_bags

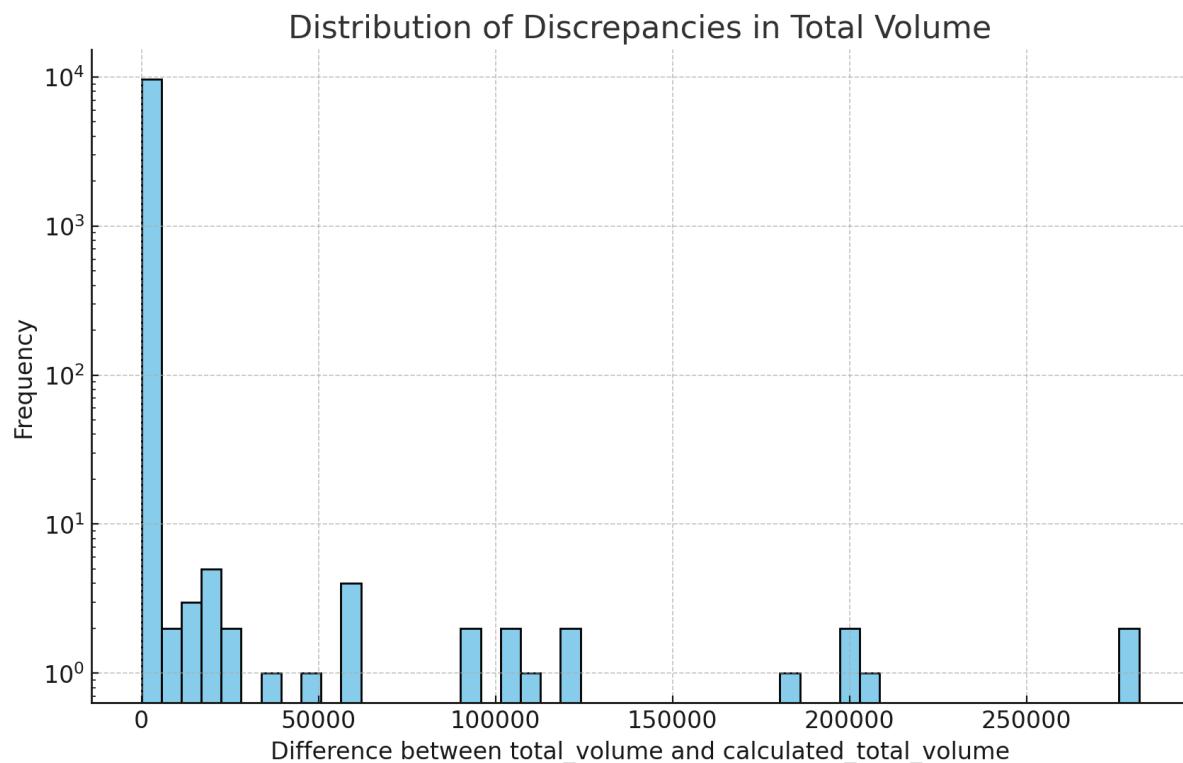
ist uns aufgefallen, dass eine starke Korrelation zwischen den Spalten besteht. Wenn wir einen genaueren Zusammenhang finden, können wir manche Spalten löschen.

**Erkenntnis:** total\_bags = small\_bags + large\_bags + xlarge\_bags

**Hypothese:** total\_volume = calculated\_total\_volume

mit:

$$\text{calculated\_total\_volume} = \mathbf{4046 + 4225 + 4770 + total\_bags}$$



Die Berechnung zeigt, dass von insgesamt 33,045 Zeilen nur 23,358 Zeilen (**total\_volume**) mit der Summe von  $\mathbf{4046 + 4225 + 4770 + total\_bags}$  übereinstimmen. Das bedeutet, dass **etwa 70.7% der Zeilen übereinstimmen**, aber es gibt Unterschiede in den restlichen Zeilen.

Weitere Analysen ergaben:

Etwa **29.23%** der Zeilen weichen um maximal 5% vom **total\_volume** ab. Dies zeigt, dass fast ein Drittel der nicht übereinstimmenden Zeilen nur eine geringe Abweichung aufweist, was auf kleinere Abweichungen oder Rundungsfehler hindeuten könnte.

# Datenvorbereitung

## Fehlende Werte

Bei der Analyse der Daten wurde festgestellt, dass **keine fehlenden Werte** vorhanden sind. Dementsprechend war die Anwendung von Imputing-Methoden nicht notwendig.

## Encoder

Zusätzlich wurden die kategorialen Variablen mithilfe eines Encoders in numerische Werte umgewandelt, um sie für das Machine-Learning-Modell nutzbar zu machen.

Folgende Variablen wurden im gesamten Projekt kodiert:

- **Date Encoder:**
  - date (als datetime) in **date\_number** (Anzahl der Tage seit frühestem Datum)
  - zyklische Kodierung der Monate in **month\_sin** und **month\_cos**
  - **harvest\_indicator**: Unterteilung der Monate in Ernte-/Wachstumszeiten
- **Geography Encoder:**
  - Label Encoder und OneHot Encoder für **geography** (Städte und Regionen)
  - GeoDistance: **dist\_to\_california** und **dist\_to\_mexico**
  - West- / Ostküste: **region**
- **Avocado-Typ**: Label Encoder auf **type** (organic / conventional)

Originaldatei:

	date	type	geography	year
1	2015-01-04	organic	Albany	2015
2	2015-01-04	conventional	Atlanta	2015
3	2015-01-04	organic	Atlanta	2015
4	2015-01-04	conventional	Baltimore/Washington	2015
5	2015-01-04	organic	Baltimore/Washington	2015

1. Durchgang:

	date_number	type	dist_to_california	dist_to_mexico	region	Atlanta	Baltimore/Washington
2	0	1	3698.524169	2845.554511	1.0	0	0
3	0	0	3698.524169	2845.554511	1.0	0	0
4	0	1	3637.489516	2666.220116	1.0	0	0
5	0	0	3637.489516	2666.220116	1.0	0	0

2. Durchgang:

	type	month_sin	month_cos	lag_7	lag_30	moving_avg_30	harvest_indicator	dist_to_california	dist_to_mexico	region	Atlanta	Baltimore/Washington
2	1	0.5	0.866025	1.54	1.54	1.54	1.0	3698.524169	2845.554511	1.0	0	0
3	0	0.5	0.866025	1.13	1.13	1.13	1.0	3698.524169	2845.554511	1.0	0	0
4	1	0.5	0.866025	2.01	2.01	2.01	1.0	3637.489516	2666.220116	1.0	0	0
5	0	0.5	0.866025	1.20	1.20	1.20	1.0	3637.489516	2666.220116	1.0	0	0

## Skalierung der Daten

Die numerischen Daten in der vorliegenden Datenmenge variieren stark in ihren Bereichen. Zur Normalisierung dieser Daten wurde der **StandardScaler** angewendet.

Die folgenden Variablen wurden im gesamten Projekt mit dem StandardScaler skaliert:

- **total\_volume**: Das gesamte Verkaufsvolumen von Avocados, die sehr große Zahlen annehmen und daher den Algorithmus sonst zu stark dominieren würden.
- **distance\_mexico** und **distance\_california**: Die Entfernungen zu den beiden Hauptanbauregionen Mexiko und Kalifornien
- **date\_number**: Eine numerische Darstellung des Datums, um die Zeitverläufe in die Analyse zu integrieren.
- **lag\_7** und **lag\_30**: Zeitverzögerungs-Features, die die Verkaufszahlen der letzten 7 bzw. 30 Tage darstellen.
- **moving\_avg\_30**: Der gleitende 30-Tage-Durchschnitt, um saisonale Effekte besser abzubilden.

Begründung für die Wahl des StandardScaler:

Der **MinMaxScaler** transformiert die Daten, sodass sie in einen vordefinierten Bereich (in der Regel zwischen 0 und 1) fallen. Der Mindestwert wird dabei auf 0 und der Höchstwert auf 1 gesetzt. Eine MinMax-Skalierung könnte die Daten zu sehr beeinflussen und nicht genügend Gewicht auf die Standardabweichungen legen, was die Modellleistung beeinträchtigen könnte.

Der **StandardScaler** transformiert die Daten so, dass sie einen Mittelwert von 0 und eine Standardabweichung von 1 haben. Der StandardScaler lässt also die Verteilung der Daten unverändert und zentriert sie, was das Modell robuster und stabiler gegen Ausreißer macht. Daher haben wir uns in der Gruppe auf den StandardScaler geeinigt.

Für **weitere Verfeinerungen** könnte man später bei den einzelnen Algorithmen die Skalierung entsprechend deren "Vorlieben" noch individuell anpassen (StandardScaler: Lineare Regression, Support Vector Machine; MinMaxScaler: KNN, k-Means; RobustScaler).

## Feature-Auswahl: Spalten entfernen

Im Zuge der Datenvorbereitung wurde eine Auswahl relevanter Features getroffen, um die Modellentwicklung zu optimieren. Die Feature-Auswahl erfolgte sowohl intuitiv basierend auf unserem Hintergrundwissen als auch durch analytische Methoden.

### 1. Durchgang der Feature-Auswahl

Im ersten Durchgang wurden Features weggelassen, die als weniger relevant für das Modell eingeschätzt wurden. Dabei wurden folgende Anpassungen vorgenommen:

- **Avocado Größen und Verpackungsgrößen**: Da diese Variablen eine zu hohe Granularität aufwiesen und insgesamt nur der Durchschnittspreis aller Spalten gegeben ist, haben wir diese einzelnen Spalten ('4046', '4225', '4770', 'total\_bags', 'small\_bags', 'large\_bags', 'xlarge\_bags') in Bezug auf unsere Hauptfrage erstmal als weniger aussagekräftig eingeschätzt.

- **Datum und Jahr:** ‘date’ und ‘year’ wurden in einem Encoder kombiniert, um die Zeitverläufe konsistenter darzustellen.
- **Geographie:** ‘geography’ wurde durch einen Label- bzw. OneHot-Encoder ersetzt.

## 2. Durchgang der Feature-Auswahl

Im zweiten Durchgang der Feature-Auswahl wurden die Anpassungen aus dem ersten Durchgang beibehalten und **zusätzliche Features** (‘lag\_7’, ‘lag\_30’, ‘moving\_avg\_30’) ergänzt, um das Modell weiter zu verbessern.

Feature-Auswahl: Spalten ergänzen und berechnen

Im Rahmen der Datenvorbereitung wurden zusätzliche Spalten berechnet, um die Modellentwicklung zu unterstützen und potenziell relevante Informationen bereitzustellen. Diese Spalten wurden teilweise durch Transformation bestehender Daten erstellt, teilweise mithilfe externer Informationen ergänzt.

### 1. Durchgang: Berechnung neuer Spalten

Im ersten Durchgang wurden folgende zusätzliche Spalten erzeugt:

- **Date\_Number (Encoder):**  
Das `date` - Attribut wurde zunächst in eine kontinuierliche numerische Spalte `date_number` umgewandelt, die die Anzahl der Tage seit dem frühesten Datum in der Datenreihe darstellt. Diese Umwandlung half, **zeitliche Lücken zu schließen** und den **Zeitverlauf gleichmäßig** darzustellen.
- **Region Encoder (0/1):**  
Die geografischen Daten wurden auf die binäre Kategorie **Westküste/Ostküste** reduziert. Hierbei wurde eine Spalte hinzugefügt, die für jede Region einen numerischen Wert (**0** für Ostküste, **1** für Westküste) speichert, basierend auf der geografischen Lage. Diese Reduktion erleichterte die Analyse regionaler Unterschiede zwischen den Küstenregionen.
- **Entferungen zu den Hauptanbauregionen:**  
Die Entferungen `distance_mexico` und `distance_california` wurden durch externe Informationen ergänzt. Diese Entferungen wurden über geografische Daten berechnet, um die **Nähe der Verkaufsregionen zu den Hauptanbauregionen** darzustellen. Die Daten für die Entferungen stammen aus dem **Avocado-Bag-Projekt** und den geografischen Analysen der EDA.
- **OneHot-Encoding für Städte der West/East-Regionen:**  
Ein OneHotEncoder wurde angewendet, um die einzelnen Städte innerhalb der Regionen **Westküste** und **Ostküste** zu kodieren. Diese zusätzliche Unterscheidung ermöglichte es, Abhängigkeiten von spezifischen Verkaufsregionen und deren Beziehung zu den Erntearbeiten besser zu analysieren.

Diese neuen Spalten tragen dazu bei, regionale und zeitliche Zusammenhänge besser darzustellen und die Modellanalyse zu unterstützen. In nachfolgenden Durchgängen der Datenvorbereitung wurden weitere Ergänzungen und Anpassungen vorgenommen, um das Modell kontinuierlich zu verbessern und zusätzliche relevante Informationen aus den Daten zu extrahieren.

## 2. Durchgang: Berechnung neuer Spalten

**Problemstellung:** Zeitvariablen wie Monate oder Tage der Woche haben eine **zyklische Natur** (z.B. Januar folgt auf Dezember). Klassische maschinelle Lernmodelle erkennen diese zyklischen Muster jedoch nicht automatisch. Zudem reichen einfache Zeitreihen nicht aus, um saisonale Schwankungen oder zeitliche Abhängigkeiten korrekt abzubilden.

Im zweiten Durchgang wurden folgende zusätzliche Spalten erzeugt:

- **month\_sin, month\_cos (Encoder):**  
Die kontinuierliche numerische Spalte `date_number` wurde durch zyklische Kodierung (Monat) ersetzt: Durch **Umwandlung der Monate in Sinus- und Cosinus-Werte** wird die zyklische Natur korrekt erfasst. Dies sorgt dafür, dass **Januar und Dezember als benachbart angesehen** werden.
- **lag\_7, lag\_30** Lag-Features (Verzögerung-Merkmale):  
Es wurden die Preise vor einer Woche und einem Monat eingeführt, um den Algorithmen zu helfen, saisonale oder zeitliche Abhängigkeiten besser zu berücksichtigen.
- **moving\_avg\_30:** Gleitende Durchschnitte:  
Kurzfristige Trends und saisonale Muster werden durch gleitende Durchschnitte geglättet dargestellt, um das Modell **robuster gegen kurzfristige Schwankungen** zu machen.

## 3. Durchgang: Berechnung neuer Spalten

- **inverse-volume:**  
Invers Volume könnte sinnvoll sein, weil es eine umgekehrte Beziehung zwischen Volumen und Preis darstellen kann. Wenn das Volumen gering ist (z.B. bei geringem Angebot), steigt der Preis oft. Dieses Feature kann somit helfen, Preisschwankungen besser zu modellieren.
- **Ersetzen bzw. Löschen von Spalten** aufgrund hoher Korrelation (vgl. EDA):
  - `total_volume` durch 4046, 4225, 4770
  - `total_bags` durch `small_bags`, `large_bags`, `xlarge_bags`

Weitere Features (optional)

Weitere spezialisierte Features, die Preistreiber berücksichtigen könnten:

1. **Regionale Nachfrage:** Ein binäres Feature, das Regionen mit hoher Nachfrage wie Los Angeles und New York markiert, wo der urbane, gesundheitsbewusste **Lebensstil** den Preis beeinflusst.
2. **Wetterabhängigkeit:** Ein saisonales Feature, das Regionen mit **harten Wintern** kennzeichnet, in denen höhere Preise durch importierte Avocados entstehen könnten.
3. **Importabhängigkeit:** Ein Feature, das Regionen markiert, die in den Wintermonaten stärker von Importen abhängig sind und daher saisonal niedrigere Preise haben könnten.
4. **Konkurrenzdichte:** Ein Feature, das die Einzelhandelskonkurrenz in städtischen Regionen reflektiert, was den Preis durch Wettbewerb beeinflusst.
5. **Kaufkraft der Region:** Ein Feature, das wohlhabende Regionen markiert, in denen höhere Preise aufgrund der höheren Zahlungsbereitschaft akzeptiert werden.

## Aufteilung des Datensatzes

### Training und Test daten

Für die Aufteilung gibt es folgende Ansätze, wobei wir einen eigenen Splitter nutzen.

#### Time Series Split (sklearn):

Dieser Split-Ansatz verwendet die TimeSeriesSplit-Methode von Scikit-learn, um die Daten schrittweise in mehrere aufeinanderfolgende Train-Test-Splits aufzuteilen. Dies ist nützlich, um die Modelle in einem echten Zeitreihen-Kontext zu validieren, ohne zukünftige Informationen zu verwenden.

Vorteil: Flexibel für mehrere Testdurchgänge und nützlich, wenn in verschiedenen Zeitabschnitten trainieren und testen werden soll.

#### Eigener Split basierend auf dem Datum:

In dieser Variante wird ein einfacher Split basierend auf einem festgelegten Datum verwendet. Daten von 2020 oder früher werden als Trainingsdaten genutzt, und spätere Daten als Testdaten.

```
# Patrick Johns

from sklearn.base import BaseEstimator, TransformerMixin

class TimeBasedTrainTestSplit(BaseEstimator, TransformerMixin):
    def __init__(self, year_column='year', test_year=None):
        """
        Initialisiert den Splitter.
        :param year_column: Name der Spalte, die das Jahr enthält
        :param test_year: Das Jahr, das für die Tests verwendet wird
        """
        self.year_column = year_column
        self.test_year = test_year

    def fit(self, X, y=None):
        """
        Diese Methode ist notwendig, um sich an die Scikit-Learn-Schnittstelle zu halten.
        Hier ist kein Training nötig, daher wird nichts gemacht.
        """
        return self

    def transform(self, X, y=None):
        """
        Führt den eigentlichen Split basierend auf dem Jahr durch.
        :param X: Der DataFrame mit den Daten
        :param y: Zielvariable, falls benötigt
        :return: Train- und Test-Daten
        """

        if self.test_year is None:
            raise ValueError("Es muss ein Testjahr angegeben werden!")

        # Aufteilen der Daten in Trainings- und Testdatensätze
        X_train = X[X[self.year_column] < self.test_year]
        X_test = X[X[self.year_column] == self.test_year]

        if y is not None:
            y_train = y[X[self.year_column] < self.test_year]
            y_test = y[X[self.year_column] == self.test_year]
            return X_train, X_test, y_train, y_test
        return X_train, X_test
```

Vorteil: Sehr simpel und direkt, speziell geeignet für Fälle, bei denen die Testdaten klar von den Trainingsdaten getrennt sein sollen (z.B. vor/nach einem bestimmten Jahr).

## Unterteilung in homogene Datenmengen

Um die Analyse und Modellierung zu verbessern, könnte der Datensatz in homogenere Teilmengen unterteilt werden. Dies kann manchen Algorithmen erleichtern, spezifische Muster und Trends innerhalb einzelner Gruppen besser zu erkennen (siehe später: Lineare Regression).

Folgende Unterteilungen sind sinnvoll:

**1. Organic vs. Conventional:**

- Eine Unterteilung in biologische (organic) und konventionelle (conventional) Avocados kann hilfreich sein, da diese beiden Kategorien unterschiedliche Preis- und Nachfrageverläufe aufweisen können.

**2. Westküste vs. Ostküste:**

- Wir haben bereits die Unterteilung der Regionen in West- und Ostküste aufgrund unterschiedlicher klimatischer, konsumorientierter und geographischer Bedingungen vorgenommen.

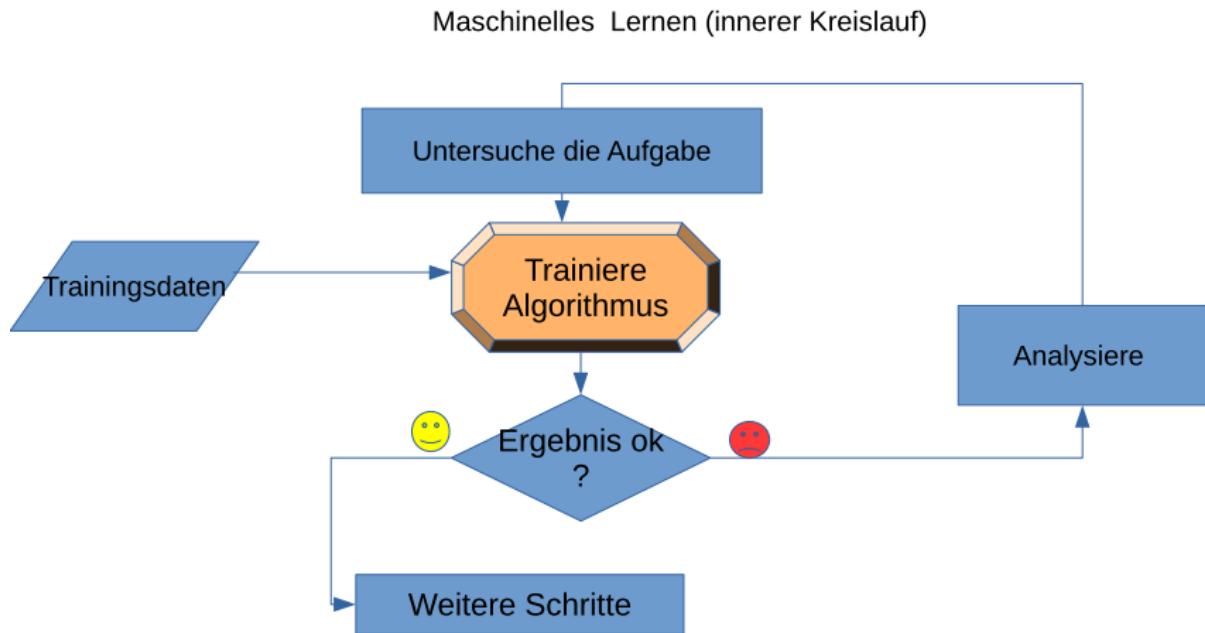
**3. Kombination beider Unterteilungen:**

- Eine Kombination der beiden vorherigen Unterteilungen ermöglicht eine noch feinere Segmentierung in vier homogene Gruppen: *organic West, organic East, conventional West, und conventional East*. Diese detailliertere Unterteilung könnte auch Aufschluss darüber geben, ob bestimmte Regionen einen höheren Verbrauch an biologischen oder konventionellen Avocados aufweisen oder ob regionale Marktunterschiede bestehen.

### *Mögliche Probleme bei der Segmentierung des Datensatzes*

1. **Reduzierte Datenmenge:** Kleinere Teilmengen führen zu weniger Daten pro Gruppe, was die statistische Aussagekraft verringern und das Risiko von Overfitting erhöhen kann.
2. **Inkonsistente Muster:** Unterschiede zwischen Segmenten erschweren die Verallgemeinerung, da Modelle, die in einer Gruppe gut funktionieren, möglicherweise nicht auf andere übertragbar sind.
3. **Übersegmentierung:** Eine zu feine Aufteilung kann übergeordnete Trends verdecken und die Stabilität des Modells beeinträchtigen.
4. **Erhöhter Ressourcenbedarf:** Mehr Modelle führen zu höherem Speicher- und Rechenaufwand, was bei großen Datensätzen oder begrenzter Kapazität problematisch sein kann.

## Anwendung verschiedener Algorithmen



## Fünf verschiedene Algorithmen

Da unsere Daten weitestgehend nicht-linear verlaufen und eine Kategorisierung unserer Zielspalte mit Avocadopreisen nicht sinnvoll wäre, haben wir hauptsächlich Regressionsmodelle ausgewählt. Wir haben uns darauf geeinigt folgende Algorithmen unterschiedlicher Herangehensweisen zu testen:

- DecisionTree (erweitert mit RandomForest) → Tingting
- SupportVectorRegressor (SVR) → Patrick
- kNextNeighbors (KNN) → Brian
- MLPRegression (als neuronales Netzwerk) → Mathieu
- LinearRegression (mit Varianten und Boosting) → Mark

### Decision Tree (Random Forest)

Decision Trees sind flexibel und eignen sich aufgrund der einfachen Interpretierbarkeit und der Unabhängigkeit von den Beziehungen zwischen den Daten besonders gut, um einen Überblick zu gewinnen. Außerdem sind sie skalierungsunabhängig, was in unserem Fall mit den vorskalierten Daten nicht zum Tragen kommt. Allerdings ist ein Decision Tree sehr robust im Umgang mit Ausreißern, die in unserem Datensatz vorhanden sind.

Durch das ensemble mit einem RandomForest werden die Voraussagen stabiler, da RandomForest das Overfitting reduziert, das durch die Aggregation vieler Trees entstehen kann.

Dieser Ansatz ist flexibel, robust gegen Ausreißer und kann mit nicht-linearen und

komplexen Daten umgehen.

## Support Vector Regressor

SVR kann sehr gut mit kleinen bis mittelgroßen nicht-linearen Datensätzen umgehen. Außerdem kann gut gesteuert werden, wie stark es an die Trainingsdaten angepasst werden soll.

## *k* Next Neighbors

KNN ist ein nicht-parametrisches Verfahren und handelt "intuitiv", was in unserem Fall dazu führen könnte, dass unentdeckte Muster aufgedeckt werden. Unser saisonal bedingter nicht-linearer Datensatz ist daher kein Problem für KNN.

## MLP Regression

Die Multi-Layer Perceptrons können komplexe Muster und nicht-lineare Beziehungen in den Daten lernen. MLPRegressor könnte Beziehungen zwischen den einzelnen Daten entdecken, die auf den ersten Blick nicht entdeckt werden. Durch die Anzahl an Hidden Layers und die maximalen Iterationsschritte kann MLPRegressor genau an unsere Daten angepasst werden. Zudem können noch verschiedene Aktivierungsfunktionen ('identity', 'logistic', 'tanh', 'relu') eingesetzt werden, um unterschiedliche Beziehungen zu lernen. Der default-Parameter 'relu' ist allerdings in den meisten Fällen der genaueste. Auch wenn MLP eher bei großen Datensätzen eingesetzt wird, sollten unsere Daten hoffentlich ausreichend sein, um ein geeignetes Regressionsmodell zu modellieren.

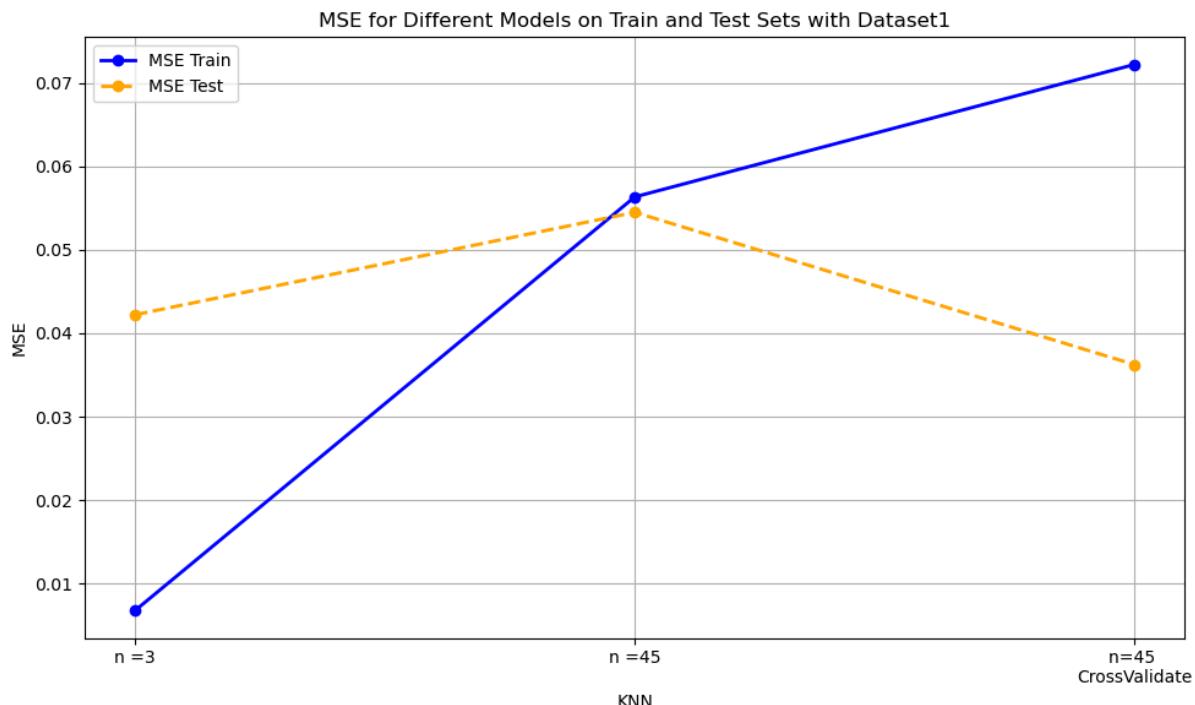
## Lineare Regression

Lineare Regression ist sehr leicht zu verstehen und mit Polynomial Features auch gut für nicht durchgehend lineare Daten geeignet. Ridge-Regression hilft, Overfitting zu vermeiden. Ein weiterer Vorteil ist die Schnelligkeit des Algorithmus. Er ist gut geeignet, um lineare Abschnitte und Beziehungen zwischen den Daten zu erkennen. Für die lineare Regression sind homogene Datenmengen wichtig (s. Kapitel 1.8).

## Verfeinerung

Der KNN-Algorithmus ist einfach, leicht zu verstehen und leicht zu implementieren. Allerdings hat die Wahl des n-Wertes einen großen Einfluss auf die Ergebnisse.

In diesem Projekt war der n-Wert beim ersten Training des Modells zu klein, das Modell war underfitting, durch die Anpassung des n-Wertes erhielten wir eine gute Anpassung, der nächste Schritt ist hoffentlich die **Kreuzvalidierung** unseres Modells, um bessere Ergebnisse zu erhalten, und tatsächlich war das der Fall, die Ergebnisse waren in der Testmenge besser, aber es gab ein Overfitting.

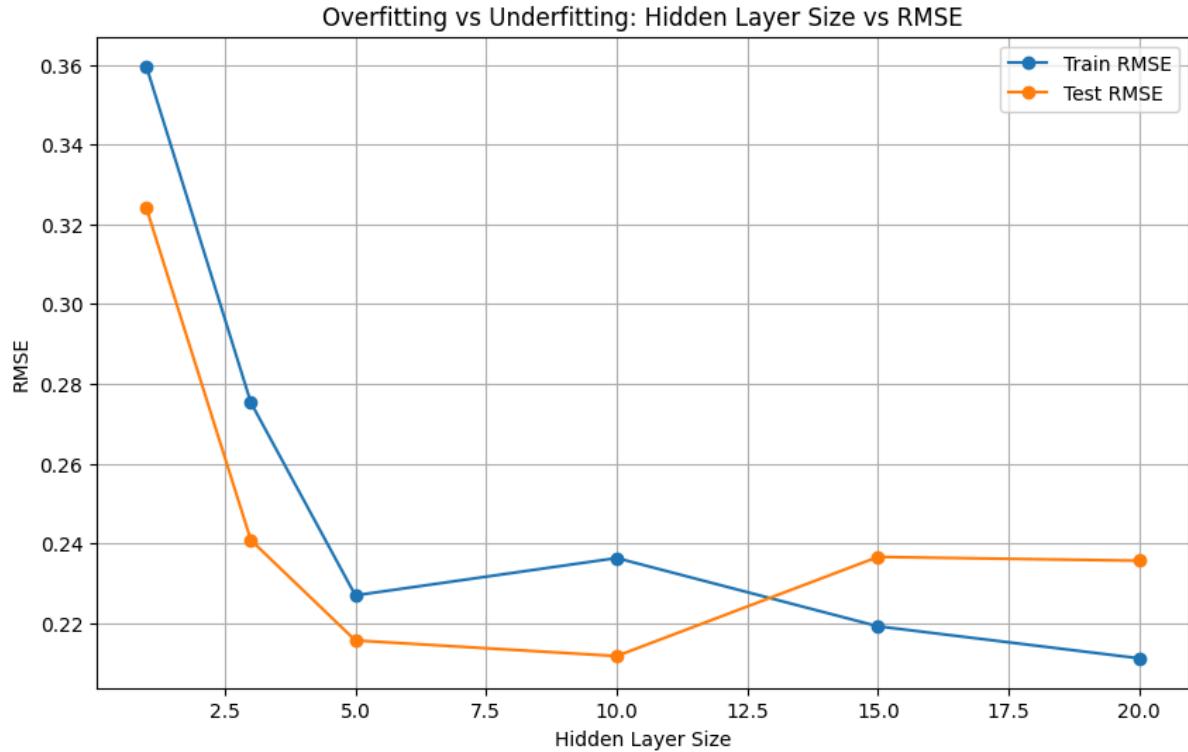


## Grid Search

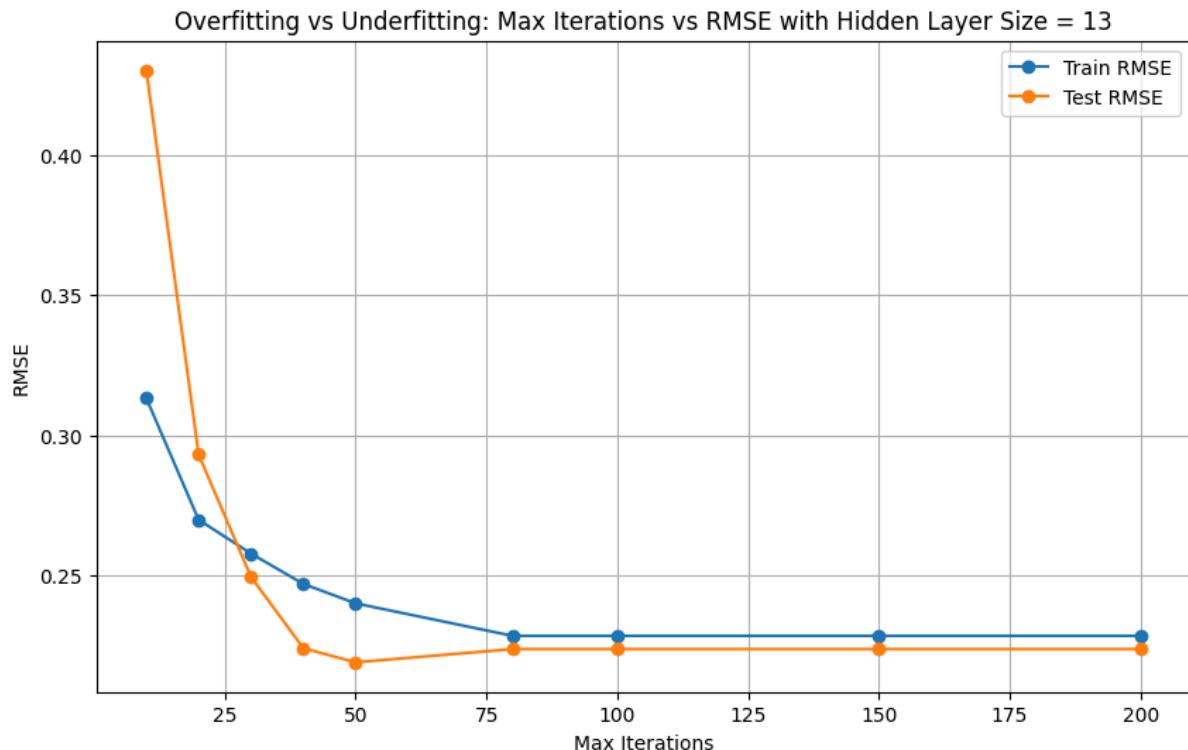
GridSearch untersucht alle übergebenen Parameter in allen möglichen Kombinationen auf den besten Score hin, der zuvor festgelegt wird (hier RMSE).

Um die bestmögliche Parameter-Kombination für unsere Daten zu erhalten (best-fit der Trainings- und Testdaten) haben wir jeweils ein GridSearchCV durchgeführt. Im Fall des MLPRegressors erhielten wir mit einem GridSearchCV mit den Parametern `param_grid={'hidden_layer_sizes':[1,3,5,10,15,20],'max_iter':[10,20,30,40,50,80,100,150,200]}, scoring='neg_root_mean_squared_error'`, `cv=5` die Ergebnis-Parameter 5 Hidden Layers und 80 max. Iterationen (bzw. 15 Hidden Layers und 80 max. Iterationen beim zweiten Datensatz). Da die dem Algorithmus zur Verfügung gestellten Werte sehr statisch waren und ein ranged GridSearchCV über solch große Spannen sehr zeit- und rechenintensiv gewesen wäre, wurden über das Plotten der Trainings-RMSE und Test-RMSE genauere Werte ermittelt. Dafür wurde zuerst nach der besten Anzahl an Hidden Layers gesucht bei einem `max_iter` Wert von 200 Iterationen. Hierbei wurden für den ersten

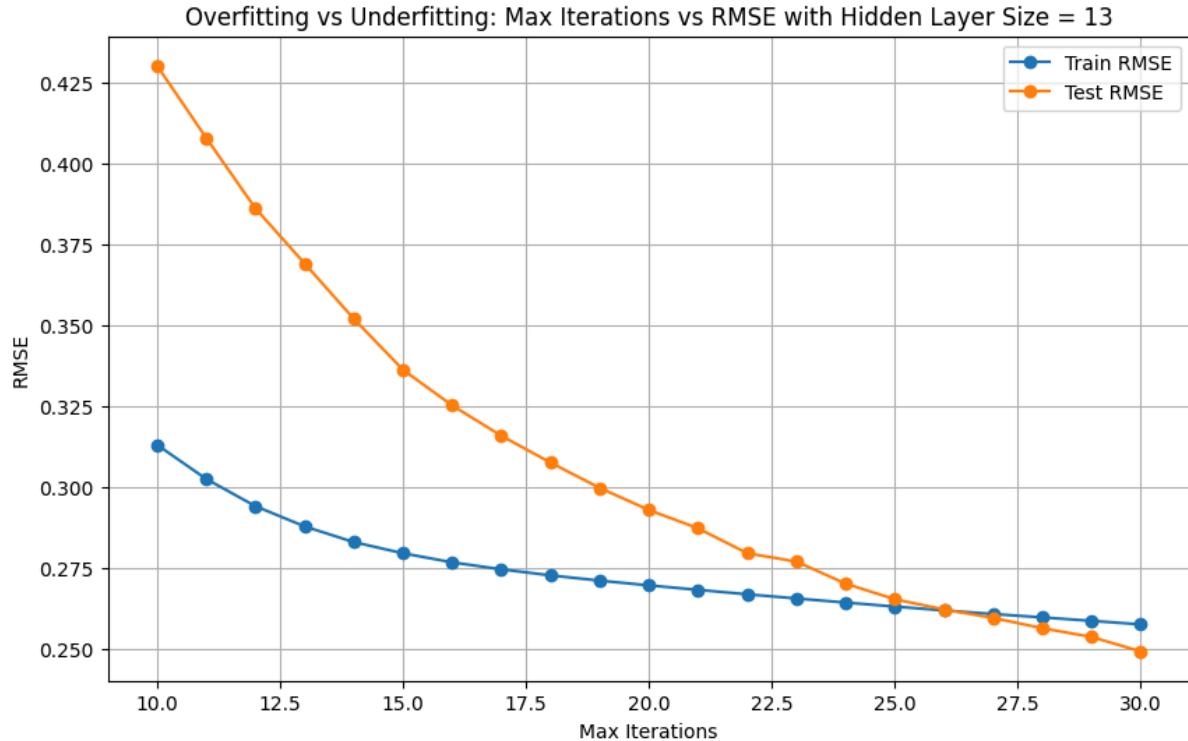
Datensatz 13 (für den zweiten 15) Hidden Layers ermittelt (s. Diagramm).



Ausgehend von 13 (bzw. 15) Hidden Layers wurden Trainings-RMSE und Test-RMSE über die Anzahl an Iterationen bei festgelegten Hidden Layers geplottet.



Die Iterationen wurden aufgrund zu geringer Auflösung nochmals genauer aufgetrennt.

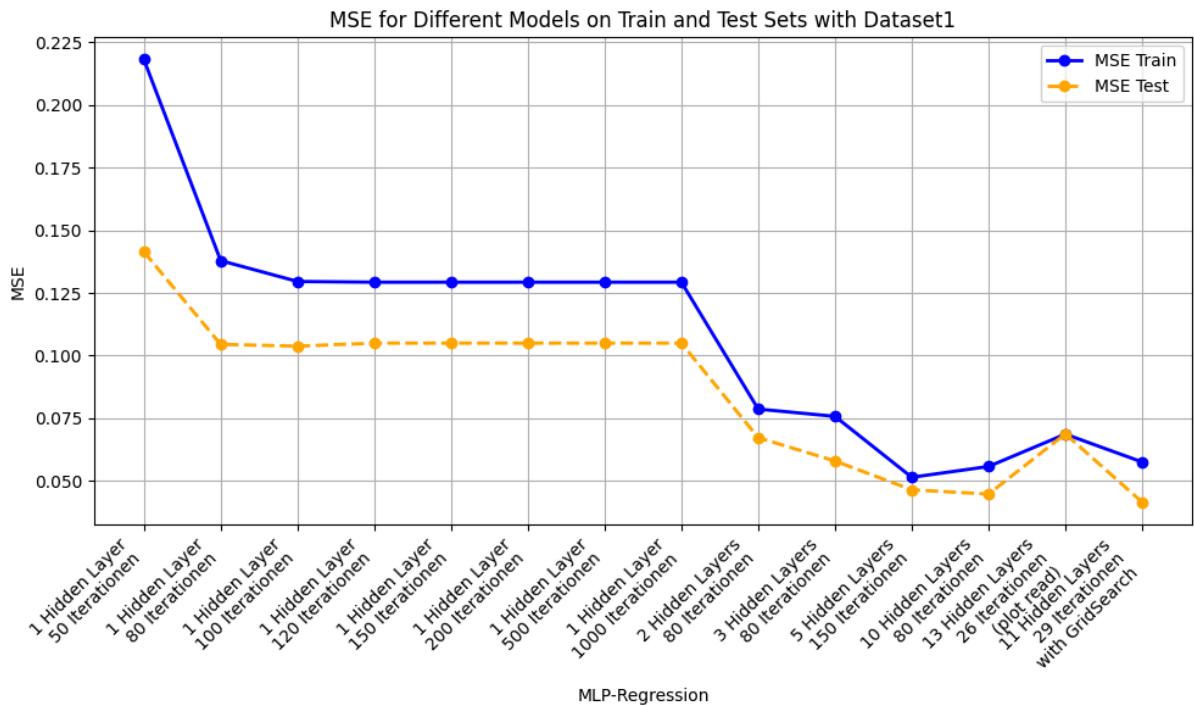


Dadurch wurde ein best-fit der Trainings- und Test-RMSE bei 13 Hidden Layers und 26 max. Iterationen (bzw. 15 Hidden Layers und 66 max. Iterationen) ermittelt.

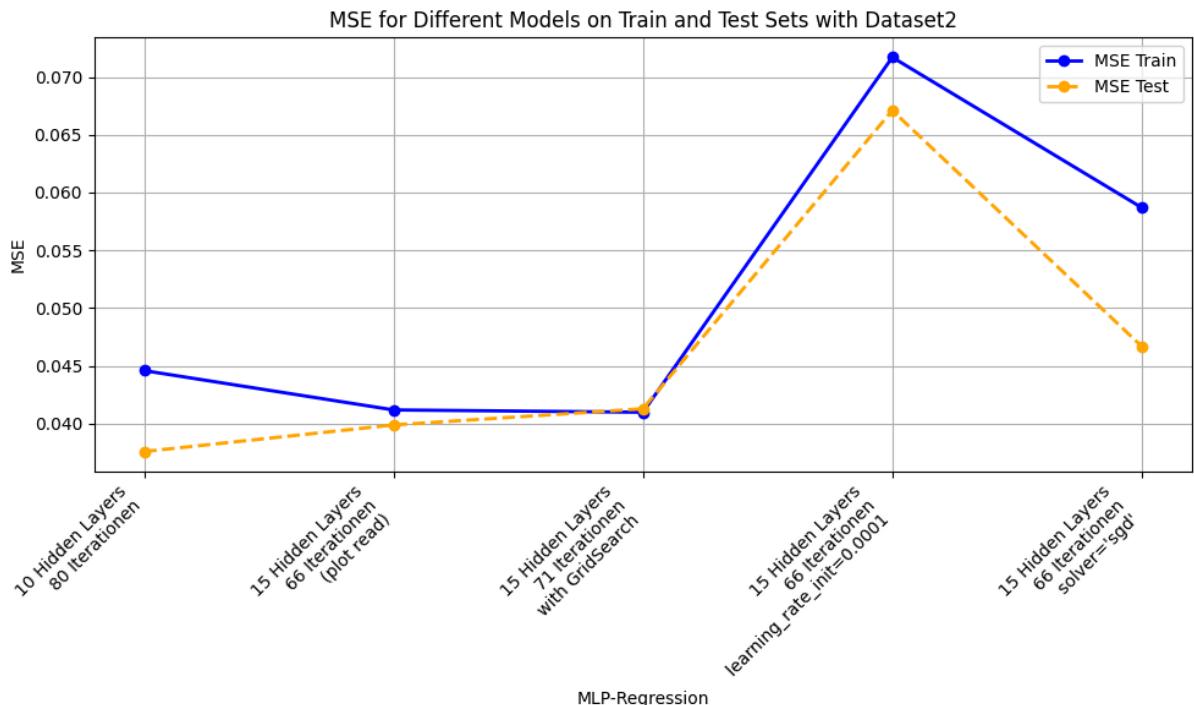
Ein anschließender GridSearchCV mit entsprechenden Range-Parametern (Hidden Layers: 10–15, max\_iter = 20–30 bzw. Hidden Layers: 10–20, max\_iter = 50–75) ergab best-fit Parameter von 11 (bzw. 15) Hidden Layers und einer max. Iteration von 29 (bzw. 71). Auch wenn dies zu einem geringeren RMSE-Wert bei Train und Test führt, ist jedoch im Gegensatz zu 13 Hidden Layers und 26 max. Iterationen ein leichtes Underfitting vorhanden. Bei Datensatz 2 finden sich hier schlechtere Werte für den Testcore und ein leichtes Underfitting, jedoch sind Train- und Testscore sehr nah beieinander (s. Diagramm ganz unten), sodass man diese Parameter dennoch in Betracht ziehen kann.

Eine Herabsetzung der initialen Learning Rate um eine Zehnerpotenz, sowie die Nutzung eines anderen Solvers ('adam' -> 'sgd'), um evtl. die Konvergenz zu erhöhen, brachte keine Besserung.

Übersicht über einen Ausschnitt der getesteten Parameter bei Datensatz 1. Man kann deutlich erkennen, dass sich ab einer bestimmten Iterationszahl (hier bei 1 Hidden Layer und 120 max. Iterationen) keine Besserung des Scores mehr feststellen lässt. Außerdem verbessert sich der Score (= reduziert sich der MSE) mit zunehmender Anzahl an Hidden Layers. Je mehr Hidden Layers eingesetzt werden, umso wahrscheinlicher ist es, dass weniger max. Iterationen nötig sind, um den bestmöglichen Score zu erreichen. Die höchste Übereinstimmung von Trainings- und Test-Score findet sich bei den Werten, die über die Plots abgelesen wurden (13 Hidden Layers und 26 max. Iterationen).



Getestete Parameter bei Datensatz 2. Das leichte Underfitting bei den verwendeten Parametern von 15 Hidden Layers und 71 max. Iterationen, als auch die schlechteren Scores bei Verminderung der initialen Learning Rate, sowie des Solver-Austauschs, sind gut zu erkennen.

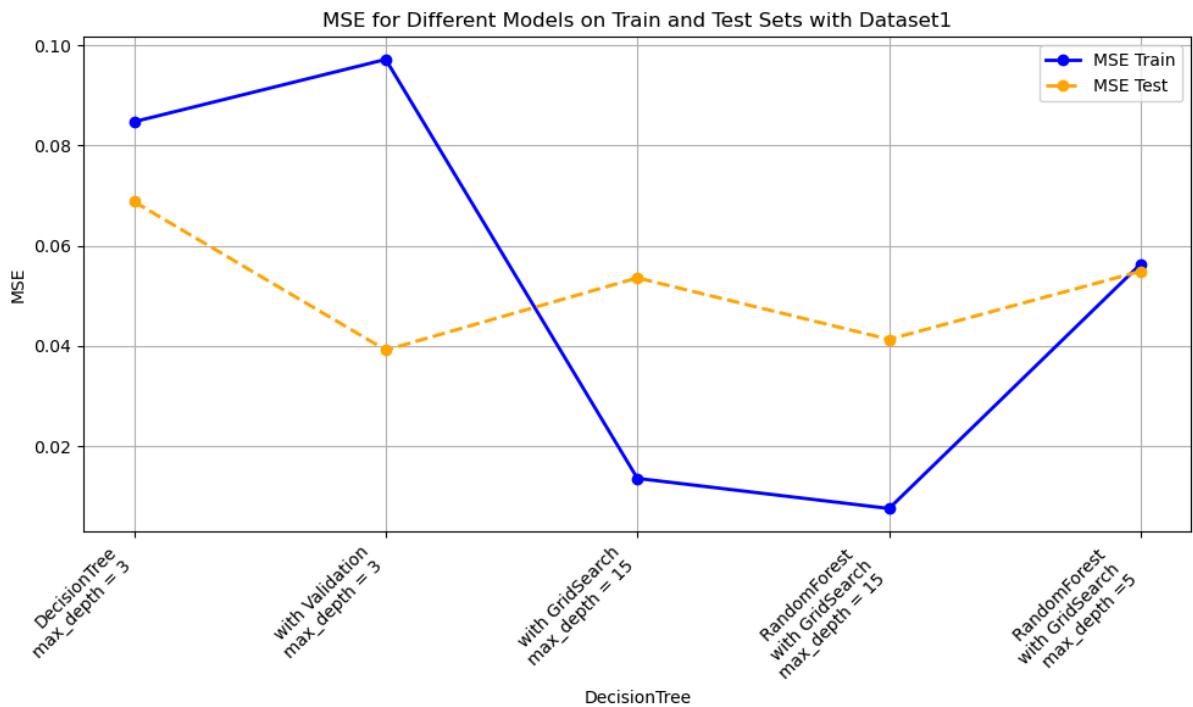


## Benutzung einer Ensemble-Methode

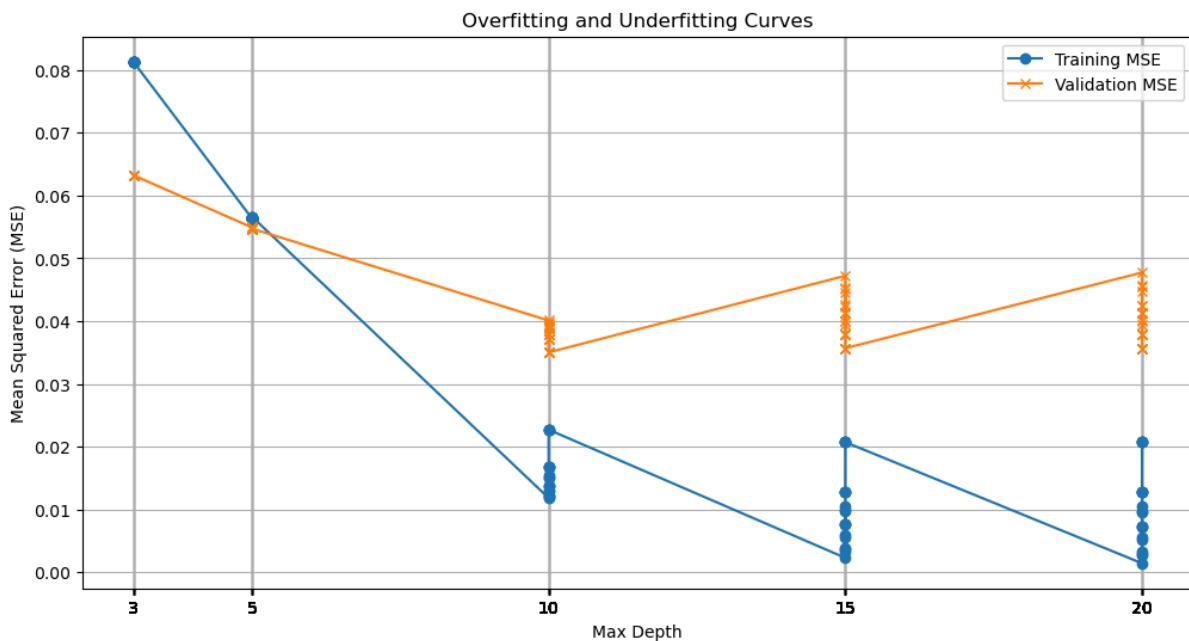
Zu Beginn habe ich eine sehr kleine max\_Depth eingestellt, was dazu führte, dass mein Modell underfitting war. Ich habe die Kreuzvalidierung verwendet, obwohl sich dadurch die Ergebnisse auf der Testmenge verbessert haben, aber das Problem der underfitting bestand immer noch, also habe ich es mit GridSearchCV versucht, um den besten Parameter zu finden, aber dieses Mal wurden die Ergebnisse auf dem Testmenge nicht besser und ich stieß auf ein Problem der overfitting.

Also versuchte ich es mit Gridsearch und Random-Forest. Die Ergebnisse auf der Testmenge verbesserten sich, aber das Problem der overfitting bestand immer noch, also untersuchte ich, wie sich die Ergebnisse mit der max\_Depth während des GridsearchCV veränderten.

Random-Forest-Algorithmus. Die Ergebnisse auf der Testmenge verbesserten sich, aber das Problem der Überanpassung war immer noch da, also untersuchte ich, wie sich die Ergebnisse mit der Tiefe während des Gridsearch-Prozesses veränderten.



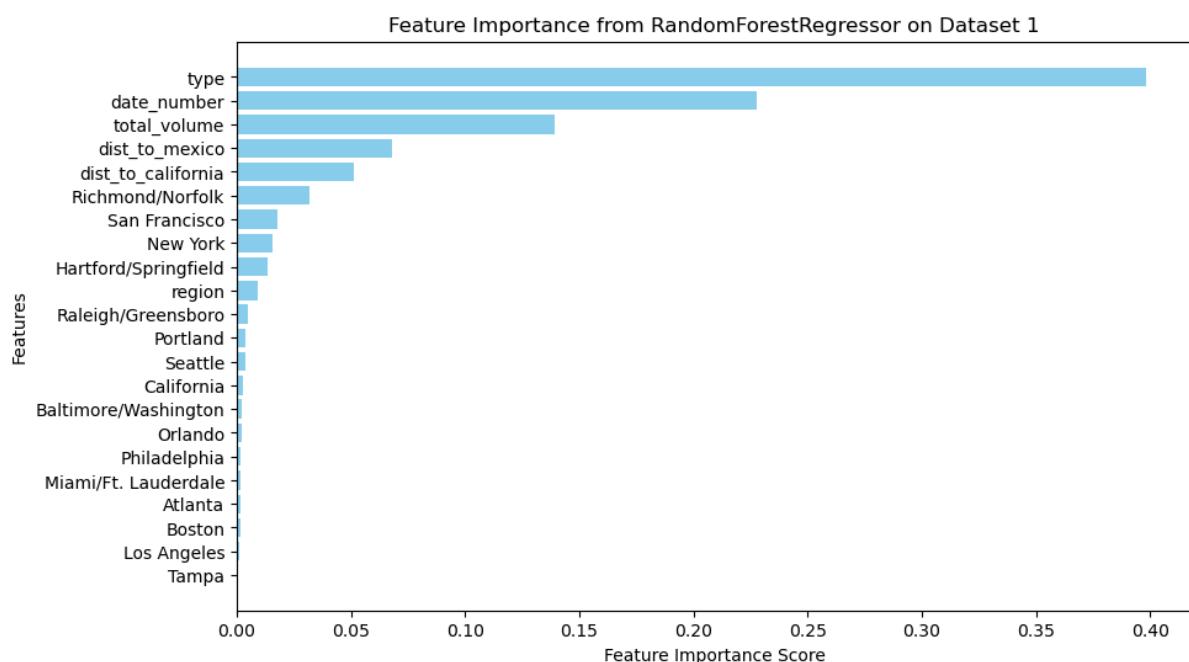
Aus dieser Abbildung können Sie ersehen, dass good fitting bei max\_Depth 5 auftritt, also habe ich das Random Forest-Modell erneut trainiert und eine good fitting erhalten, aber die Ergebnisse auf der Testmenge sind etwas schlechter als beim letzten Mal.

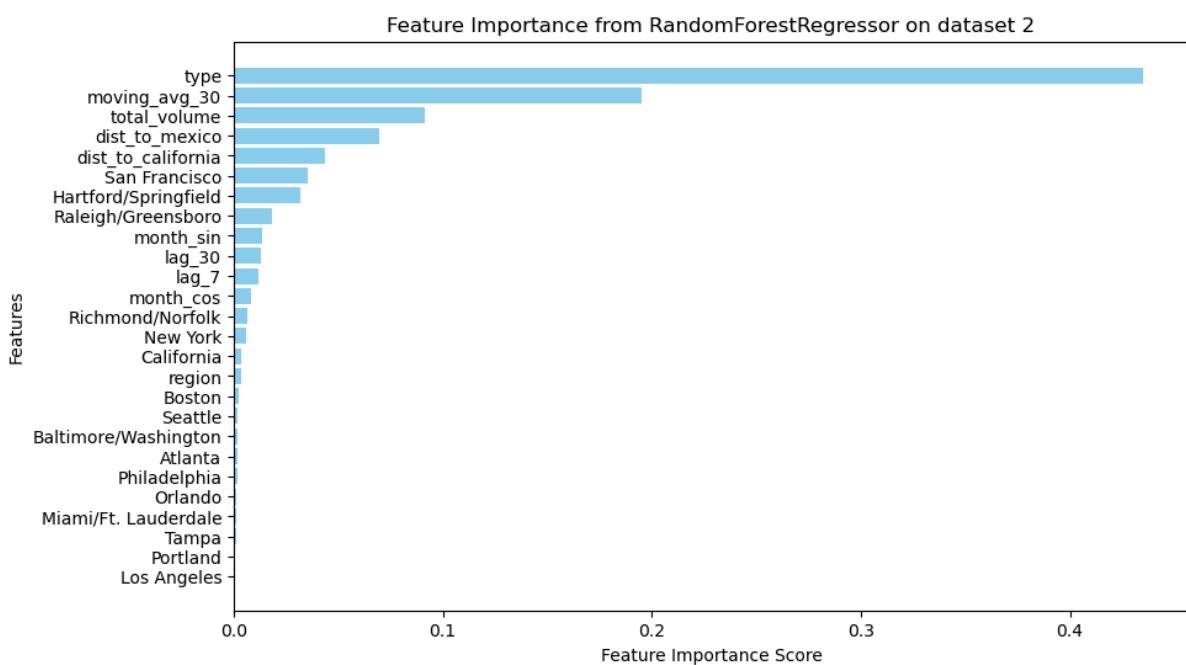


Durch verschiedene Versuche eines Modells wurde festgestellt, dass mit einem integrierten Algorithmus (Ensemble) bessere Ergebnisse erzielt werden können.

Ich habe einen typischen Bagging Algorithmus (Random Forest) verwendet, bei dem der Schwerpunkt auf einer zufälligen Auswahl der Trainingsmenge liegt und durch wiederholtes Sampling neue Modelle trainiert werden, die schließlich auf der Grundlage dieser Modelle gemittelt werden.

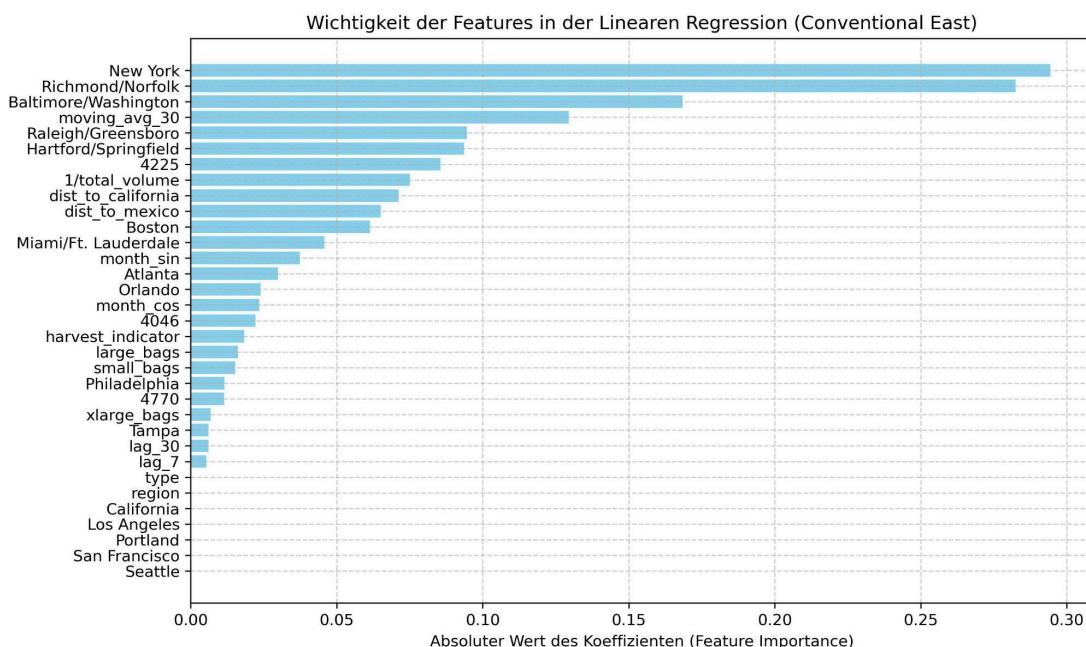
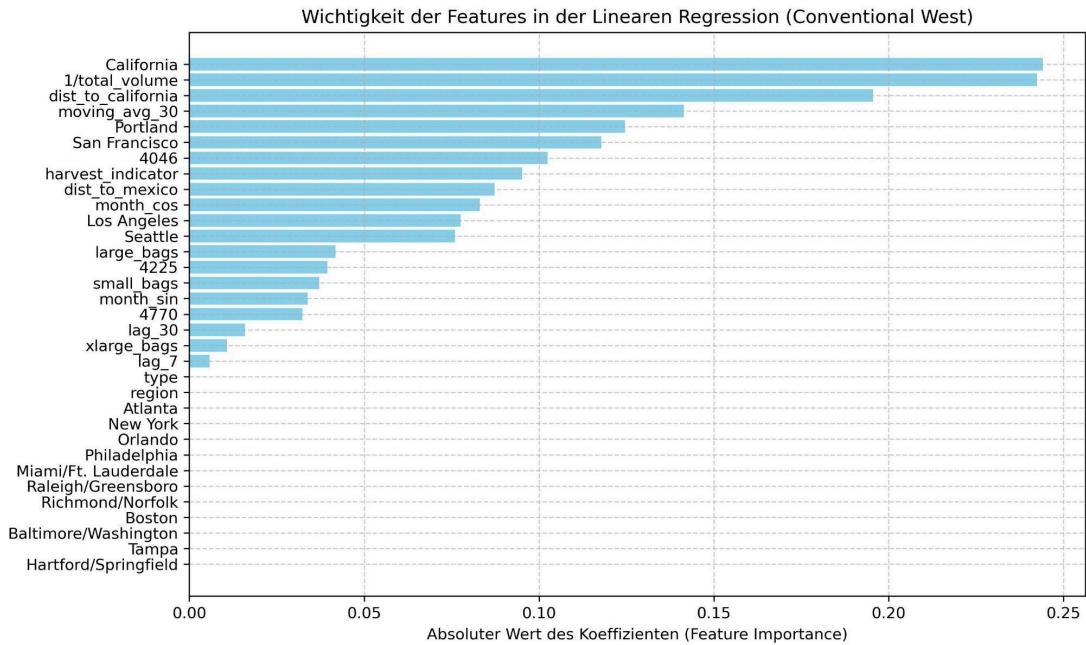
Der weitere Vorteil dieses Algorithmus besteht darin, dass wir bei Zufallsstichproben zusätzlich Gewichte für verschiedene Features ableiten können und bei großen und komplexen Daten unwichtige Feature herausfiltern können.





## Unterteilung der Daten in Untermengen

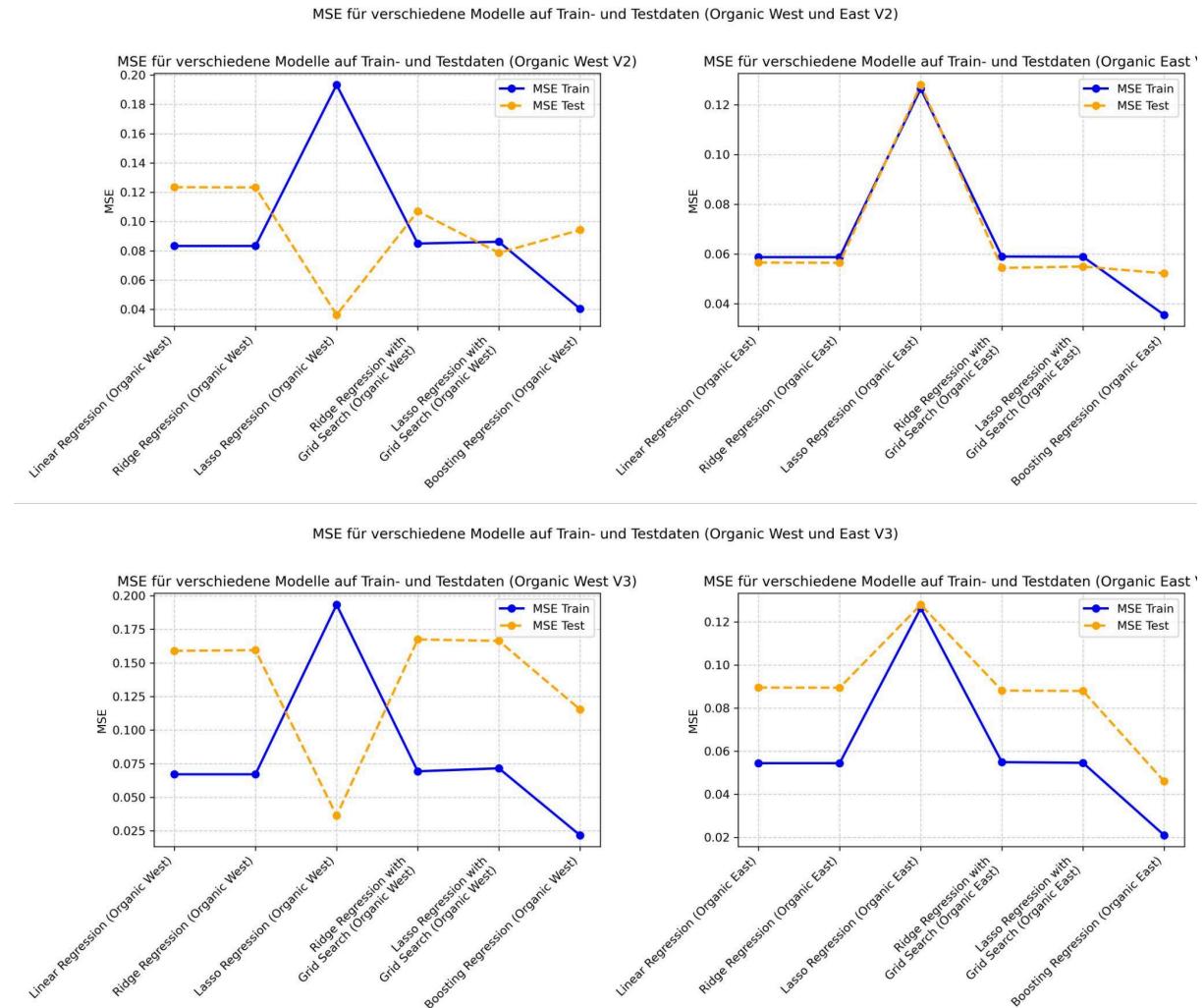
### Feature Importance bei der Linearen Regression:



### Analyse der Feature-Wichtigkeit für Lineare Regression (Conventional East und West)

In der Conventional East Region sind „New York“, „Richmond/Norfolk“ und „Baltimore/Washington“ die dominierenden Features, was darauf hindeutet, dass diese Städte stark mit den Preisvariationen korrelieren. Außerdem scheint „moving\_avg\_30“ saisonale Preisentwicklungen gut auszugleichen. Auf der West-Coast hingegen sind „California“ und „1/total\_volume“ die am stärksten gewichteten Features, was die Bedeutung dieser Region für den Preis erklärt. Zusätzlich ist die Entfernung zu Kalifornien und Mexiko („dist\_to\_california“ und „dist\_to\_mexico“) relevant, was auf logistische Faktoren hindeuten könnte.

## Organic mit West-/East Coast Aufteilung:



## Ergebnis:

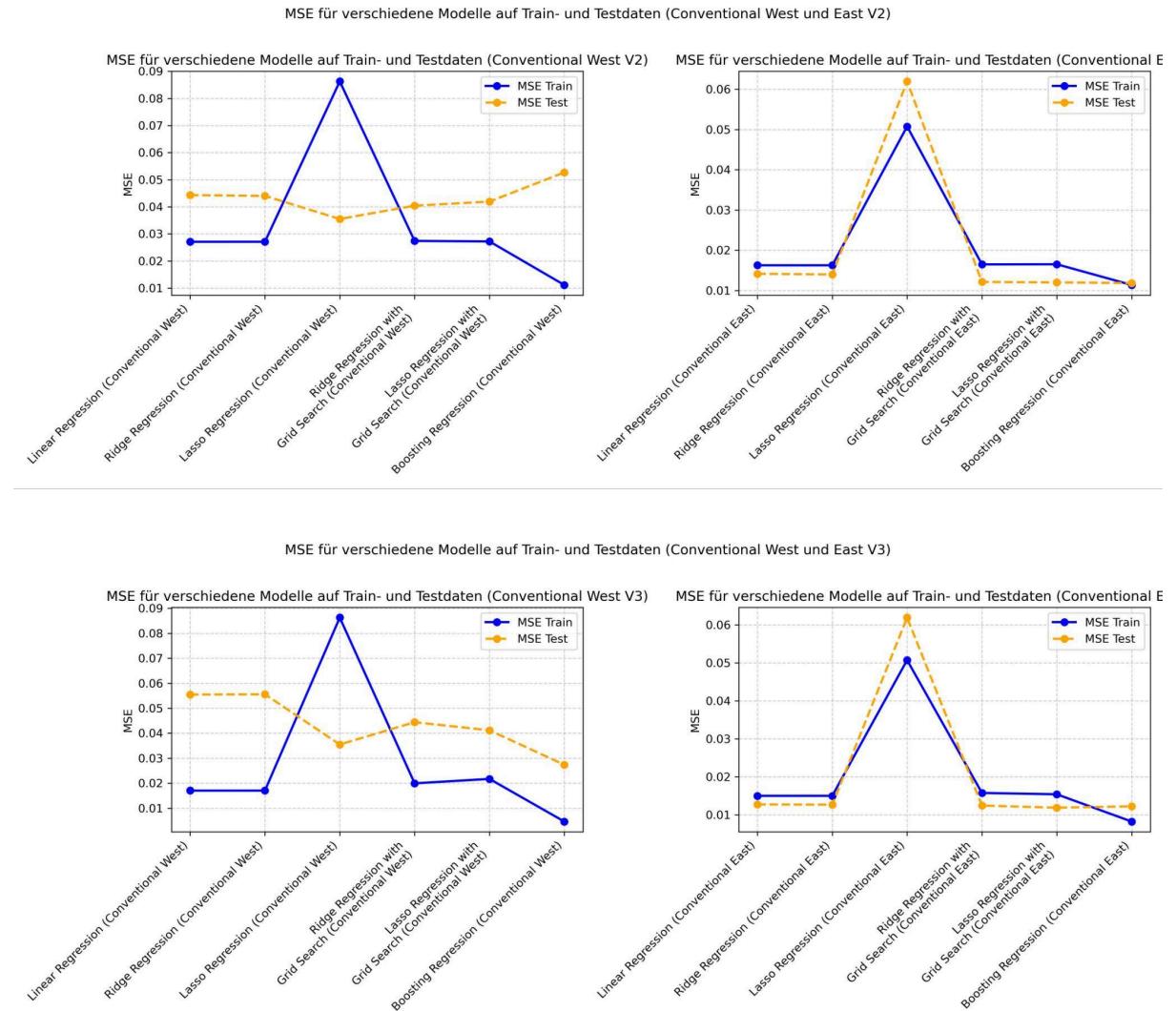
### Organic West:

- Das **Boosting-Modell** zeigt die niedrigste Test-MSE, gefolgt von der **Ridge Regression** mit Grid Search-Optimierung. Beide Modelle erfassen die Preisstrukturen gut.
- Lasso-Modelle** schneiden tendenziell schlechter ab, was auf den Bedarf an mehr Flexibilität hinweist.

### Organic East:

- Auch hier erzielt das **Boosting-Modell** die beste Test-MSE, was darauf hindeutet, dass es regionalen Preisschwankungen gut folgen kann.
- Die **Test-MSE** ist insgesamt etwas niedriger als auf der Westküste, was auf eine einfachere Vorhersagbarkeit der Preisdynamiken auf der Ostküste hinweisen könnte.

## Conventional mit West-/East Coast Aufteilung:



## Ergebnis:

### Organic West:

- Das **Boosting-Modell** erzielt die niedrigste Test-MSE und zeigt eine überlegene Fähigkeit, die Preisstrukturen abzubilden.
- Die **Ridge Regression** mit Grid Search-Optimierung schneidet ebenfalls gut ab, was auf die Wirksamkeit der Regularisierung in dieser Region hinweist.
- Lasso-Modelle** schneiden tendenziell schlechter ab, was darauf hindeutet, dass eine höhere Flexibilität und zusätzliche Features erforderlich sein könnten, um die Preisvariation zu erfassen.

### Organic East:

- Auch hier erzielt das **Boosting-Modell** die beste Test-MSE, was darauf hindeutet, dass es besonders gut in der Lage ist, regionale Preisschwankungen zu modellieren.
- Die Test-MSE ist insgesamt etwas niedriger als auf der Westküste, was auf eine möglicherweise einfachere Vorhersagbarkeit der Preisdynamiken auf der Ostküste hinweist.

## **Regionale Unterschiede bei Conventional und Organic im Allgemeinen:**

- **Conventional:**
  - Die Unterschiede zwischen West- und East-Coast bei konventionellen Produkten sind weniger stark ausgeprägt. Dies könnte darauf hinweisen, dass die Preisschwankungen für konventionelle Produkte weniger regional spezifisch sind und von ähnlichen Marktdynamiken beeinflusst werden.
- **Organic:**
  - Die **Unterschiede zwischen West- und East-Coast** sind bei Bio-Produkten stärker sichtbar. Organic West zeigt tendenziell höhere Test-MSE-Werte als East. Organic East ist durchgängig leichter vorherzusagen, was möglicherweise auf stabilere Nachfrage- und Angebots-Trends für Bio-Produkte auf dieser Küste hinweist.

## **Problem des Overfitting anhand der Ergebnisse:**

Die Test- und Train-MSE-Werte für verschiedene Modelle und Regionen lassen auf mögliche Overfitting-Probleme schließen, besonders bei bestimmten Modellen und Datenaufteilungen:

- **Lasso & Ridge Regression:** Wenig Diskrepanz zwischen Train- und Test-MSE, zeigt stabile Leistung und minimiert Overfitting, jedoch gelegentlich Unteranpassung (besonders bei Lasso) auf komplexen Datensätzen wie Organic West.
- **Boosting Modell:** Sehr niedrige Train-MSE und höhere Test-MSE, besonders auf Organic-Daten, deutet auf Overfitting hin. Modell passt gut an Trainingsdaten, jedoch weniger an Testdaten.
- **Polynomial & Bagging Regression:** Größere Differenzen zwischen Train- und Test-MSE, insbesondere bei hohen Polynomialgraden. Indikator für Overfitting, da Modelle übermäßig komplexe Muster lernen, die schwer übertragbar sind.

Insgesamt zeigt sich bei den **komplexeren Modellen**, insbesondere Boosting und Polynomial Regression, ein höheres Risiko für Overfitting, insbesondere bei **Organic West**. Die einfacheren Modelle wie Ridge Regression liefern oft stabilere und konsistenter Ergebnisse zwischen Training und Test, was sie für eine Anwendung auf komplexen, regional diversifizierten Märkten zu einer guten Wahl machen könnte. Um Overfitting weiter zu reduzieren, könnten Hyperparameter-Tuning, die Einführung von Cross-Validation oder eine stärkere Regularisierung hilfreich sein.

## **Bestes Testergebnis:**

Boosting auf den Datensatz nach dem 2. Durchgang (Datenvorbereitung) aufgeteilt in Conventional + East:

- **Train RMSE: 0.106643**
- **Test RMSE: 0.108862**

## Alle Scores in Tabelle

Dataset 1	Random Forest	SVR + GridSearch	MLP + GridSearch	KNN + CrossValidate	Lineare Regression
MSE (X_train)	0.056373	0.043809	0.0558	0.07224	0.079577
MSE (X_test)	0.054931	0.045385	0.0448	0.03624	0.068524

Dataset 2	Random Forest	SVR + Bagging	MLP + GridSearch	KNN + CrossValidate	Ridge + Grid
MSE (X_train)	0.043187	0.041305	0,0410	0.048208	0.055631
MSE (X_test)	0.043959	0.036085	0,0413	0.045361	0.035555

**Bestes Testergebnis durch Unterteilung des Datensatzes in homogene Teilmengen:**

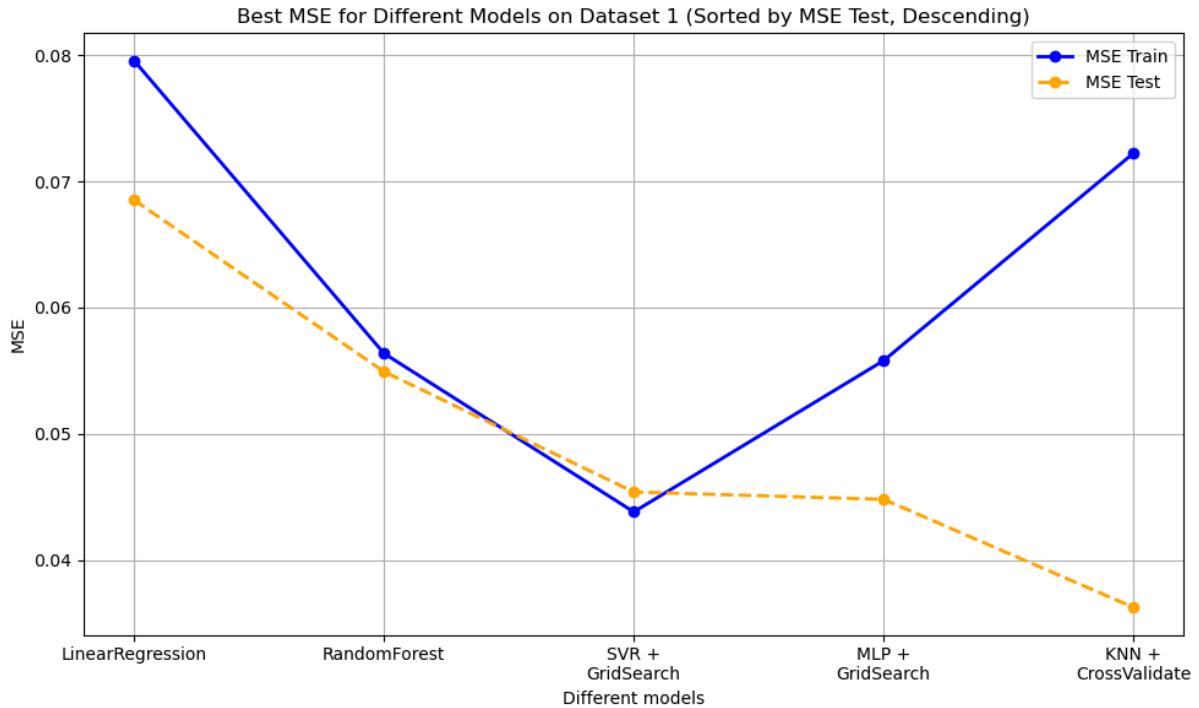
mit **Boosting** auf den Datensatz nach dem 2. Durchgang (Datenvorbereitung) aufgeteilt in Conventional + East:

- Train MSE: 0.011373
- Test MSE: 0.011850

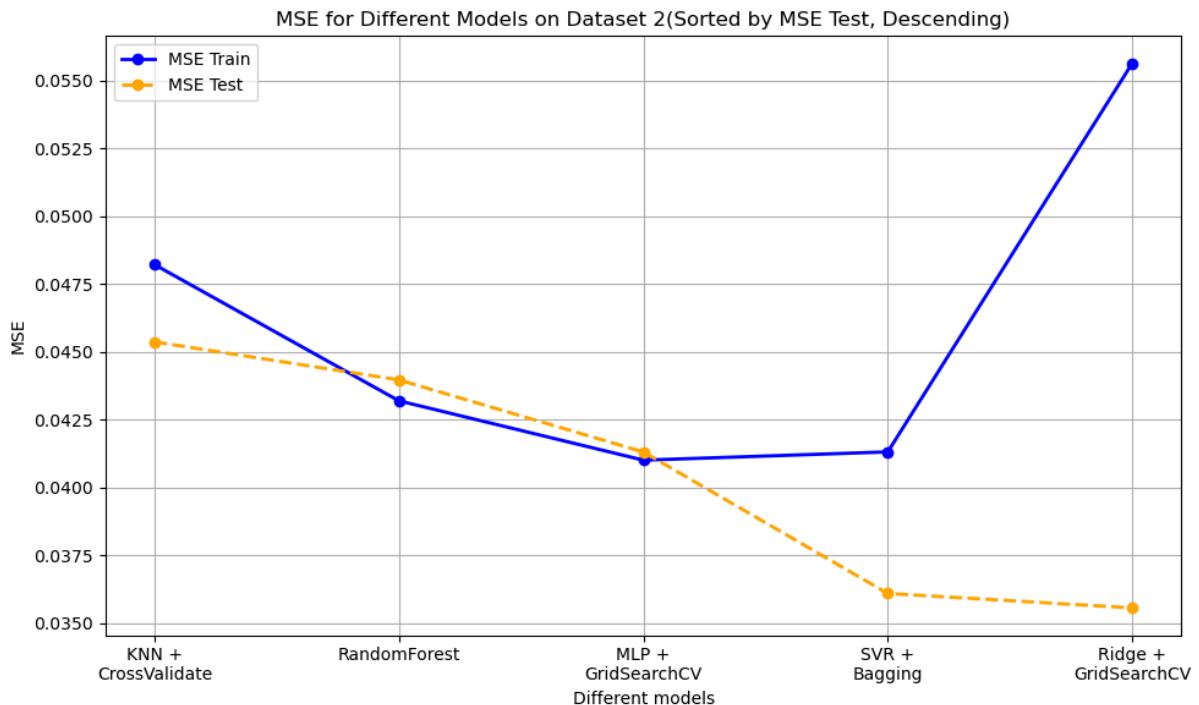
**code:** `boosting_model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)`

## Alle Scores als Gesamt Plot

Vergleich der besten Ergebnisse der verschiedenen Modelle auf der Grundlage von Daten 1



Vergleich der besten Ergebnisse der verschiedenen Modelle auf der Grundlage von Daten 2



### Kommentar:

Die Algorithmen KNN, MLP und RandomForest passen nur wenige Parameter an, und eine weitere Anpassung der Parameter wird sicherlich zu besseren Werten führen.

## Fazit

Der Gesamtscore hat sich bei den Algorithmen ohne Boosting kaum verbessert. Unser Ziel konnten wir nur durch Boosting auf bestimmten Teilmengen erreichen. Um den Regressor effektiv einzusetzen, müssen spezialisierte Einzelmodelle entwickelt werden. Eine Vorlage mit Pipeline und PCA wurde bereits erstellt und kann als Grundlage dienen.

Erfolge wurden durch Feature-Reduktion und Hinzufügen zusätzlicher Features erzielt. Eine PCA könnte noch tiefere Einblicke bieten und ist eine Alternative zum einfachen Entfernen von Features. Entscheidend für den Erfolg ist die Aufteilung des Datensatzes in homogene Gruppen (z.B. Ost/West und Organic/Konventionell), was jedoch zu Overfitting führen könnte, da zu wenig Daten vorhanden sind. Kaggle bietet hierfür Daten aus weiteren Jahren an, um eine regionale Aufteilung zu verbessern.

Die Ergebnisse sind valide, da sowohl auf RMSE als auch auf Overfitting geachtet wurde. Für Prognosen in die Zukunft müssen jedoch zusätzliche Faktoren wie Wechselkurs, Inflation und andere berücksichtigt werden. Methoden wie PCA in Kombination mit externen Daten wie Wetter, Kaufkraft und Einkommensverteilung könnten zukünftig eingesetzt werden.

## ANHANG: Weitere python-Funktionen und Klassen

### PJ\_SVR\_PIPE\_V03

Das entwickelte Verfahren zur Vorhersage von Avocado-Preisen nutzt eine modulare Pipeline auf Basis von Support Vector Regression (SVR), kombiniert mit Vorverarbeitungs- und Optimierungsschritten.

#### 1. Datenvorbereitung:

- Der **AvocadoDataLoader** lädt die Daten und filtert sie für spezifische Städte.
- Die **FeatureEngineer**-Klasse transformiert die Daten: Sie extrahiert den Monat aus dem Datumsfeld, berechnet den inversen Volumenwert und erstellt periodische Features (sinus/cosinus) zur Erfassung von Saisonalitäten.

#### 2. Train-Test-Split:

- Der **TrainTestSplitter** teilt die Daten in Trainings- und Testdaten auf, basierend auf einem festgelegten Jahr (z.B. 2020).

#### 3. Pipeline-Bau:

- Der **SVRPipelineBuilder** baut eine Pipeline mit PCA zur Dimensionenreduktion, Standardisierung und SVR zur Modellierung. GridSearchCV wird verwendet, um die besten Hyperparameter zu finden.

#### 4. Modellbewertung:

- Der **ModelEvaluator** evaluiert das Modell, indem Metriken wie MAE, MSE, RMSE und R<sup>2</sup> sowohl für Training als auch Test berechnet und in einem DataFrame zurückgegeben werden.

Dieses Verfahren ermöglicht eine flexible und skalierbare Vorhersage mit Optimierung der Modellparameter durch GridSearch, unter Berücksichtigung von saisonalen Einflüssen.

### GeoDistanceEncoder

```
# Patrick GeoDistanceEncoder für geography
# Importieren der notwendigen Bibliotheken
from geopy.distance import geodesic
# Definieren der Städte mit ihren Koordinaten
static_coordinates = [
    ('Albany', 42.6526, -73.7562),
    ('Atlanta', 33.749, -84.388),
    ('Baltimore/Washington', 39.2904, -76.6122),
    ('Boise', 43.615, -116.2023),
    ('Boston', 42.3601, -71.0589),
    ('Buffalo/Rochester', 43.1566, -77.6088),
    ('California', 36.7783, -119.4179),
    ('Charlotte', 35.2271, -80.8431),
    ('Chicago', 41.8781, -87.6298),
    ('Cincinnati/Dayton', 39.1031, -84.512),
```

```

('Columbus', 39.9612, -82.9988),
('Dallas/Ft. Worth', 32.7767, -96.797),
('Denver', 39.7392, -104.9903),
('Detroit', 42.3314, -83.0458),
('Grand Rapids', 42.9634, -85.6681),
('Great Lakes', 45.0, -84.0),
('Harrisburg/Scranton', 40.2732, -76.8867),
('Hartford/Springfield', 41.7627, -72.6743),
('Houston', 29.7604, -95.3698),
('Indianapolis', 39.7684, -86.1581),
('Jacksonville', 30.3322, -81.6557),
('Las Vegas', 36.1699, -115.1398),
('Los Angeles', 34.0522, -118.2437),
('Louisville', 38.2527, -85.7585),
('Miami/Ft. Lauderdale', 25.7617, -80.1918),
('Midsouth', 35.1495, -90.0489),
('Nashville', 36.1627, -86.7816),
('New Orleans/Mobile', 29.9511, -90.0715),
('New York', 40.7128, -74.006),
('Northeast', 41.5, -75.0),
('Northern New England', 44.0, -71.5),
('Orlando', 28.5383, -81.3792),
('Philadelphia', 39.9526, -75.1652),
('Phoenix/Tucson', 33.4484, -112.074),
('Pittsburgh', 40.4406, -79.9959),
('Plains', 39.0119, -98.4842),
('Portland', 45.5152, -122.6784),
('Raleigh/Greensboro', 35.7796, -78.6382),
('Richmond/Norfolk', 37.5407, -77.436),
('Roanoke', 37.2709, -79.9414),
('Sacramento', 38.5816, -121.4944),
('San Diego', 32.7157, -117.1611),
('San Francisco', 37.7749, -122.4194),
('Seattle', 47.6062, -122.3321),
('South Carolina', 33.8361, -81.1637),
('South Central', 34.7465, -92.2896),
('Southeast', 32.3182, -86.9023),
('Spokane', 47.6588, -117.426),
('St. Louis', 38.627, -90.1994),
('Syracuse', 43.0481, -76.1474),
('Tampa', 27.9506, -82.4572),
('Total U.S.', 39.8283, -98.5795),
('West', 39.1178, -120.013),
('West Tex/New Mexico', 31.9686, -99.9018)
]

class GeoDistanceEncoder:
    def __init__(self, city_coordinates, reference_points):
        """
        Initialisiert den GeoDistanceEncoder mit den Städten und den Referenzpunkten.

        :param city_coordinates: Dictionary mit Städtenamen und deren (Latitude, Longitude)-Koordinaten
        :param reference_points: Dictionary mit Referenzpunkten und deren (Latitude, Longitude)-Koordinaten
        """
        self.city_coordinates = city_coordinates
        self.reference_points = reference_points

```

```

def fit(self, X, y=None):
    # fit ist nicht notwendig, wird aber für Kompatibilität bereitgestellt.
    return self

def transform(self, X):
    """
    Berechnet die geografischen Distanzen zu den Referenzpunkten. Falls die Spalten 'latitude'
    und 'longitude' nicht im Datensatz vorhanden sind, werden sie anhand der Stadtzuordnung berechnet.
    :param X: DataFrame mit der Spalte 'geography' (Städte) und optional 'latitude' und
    'longitude'
    :return: DataFrame mit den berechneten Distanzen zu den Referenzpunkten
    """
    # Überprüfen, ob die Spalten 'latitude' und 'longitude' bereits vorhanden sind
    if 'latitude' not in X.columns or 'longitude' not in X.columns:
        # Falls die Spalten nicht existieren, die Koordinaten der Städte hinzufügen
        X['latitude'] = X['geography'].map(lambda x: self.city_coordinates.get(x, (None,
None))[0])
        X['longitude'] = X['geography'].map(lambda x: self.city_coordinates.get(x, (None,
None))[1])

    # Entfernen von Zeilen ohne gültige Koordinaten
    X = X.dropna(subset=['latitude', 'longitude'])

    # Berechnen der Entferungen zu den Referenzpunkten
    for name, coords in self.reference_points.items():
        X[f'dist_to_{name}'] = X.apply(lambda row: geodesic((row['latitude'], row['longitude']),
coords).kilometers, axis=1)

    return X

def fit_transform(self, X, y=None):
    """
    Führt fit und transform direkt hintereinander aus.
    :param X: DataFrame mit der Spalte 'geography' (Städte) und optional 'latitude' und
    'longitude'
    :return: DataFrame mit den berechneten Distanzen zu den Referenzpunkten
    """
    return self.fit(X).transform(X)

# Umwandeln der Liste in ein Dictionary für den Encoder
city_coordinates = {city: (lat, lon) for city, lat, lon in static_coordinates}

# Referenzpunkte (Kalifornien und Mexiko)
reference_points = {
    'california': (36.7783, -119.4179),
    'mexico': (23.6345, -102.5528)
}

# Anwendung des GeoDistanceEncoders
encoder = GeoDistanceEncoder(city_coordinates, reference_points)
df = encoder.fit_transform(df)

# Überprüfen der ersten Zeilen des Datensatzes
print(df[['geography', 'latitude', 'longitude', 'dist_to_california', 'dist_to_mexico']].head())

```