Betreut durch Dipl.-Biol. Ralf Darius
WS2023 / 2024
26. Feb 2024
Hochschule Rhein-Waal

Projektarbeit Data Mining and Machine Learning

Analyse von Immobilienbewertungen

Till Wegener [1] und Dennis Adamczyk [2]

[1]: Matrikelnummer: 28891, E-Mail-Adresse: till.wegener@hsrw.org
[2]: Matrikelnummer: 30545, E-Mail-Adresse: dennis.adamczyk@hsrw.org

Abstract

Die Analyse und Bewertung von Immobilien ist ein wichtiger Bestandteil des Immobilienmarktes. In dieser Arbeit wird ein Datensatz von Immobilienbewertungen aus Taiwan auf verschiedene Eigenschaften hin untersucht. Hierzu verwenden wir diverse Methoden aus den Bereichen des Data Mining und Machine Learning. Ziel ist es, die Bewertung von Immobilien, mithilfe verschiedener Regressions-Methoden, anhand verschiedener Eigenschaften vorherzusagen und den Einfluss der unterschiedlichen Faktoren zu analysieren.

Keywords:

Data Mining, lineare Regression, multiple lineare Regression, nicht-lineare Regression, K-Means Clustering, Immobilienmarkt

Inhaltsverzeichnis

1 Eigenständigkeitserklärung

Hiermit erklären wir, dass wir die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt haben. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht.

2 Einleitung

2.1 Vorstellung des Projektes

In dieser Arbeit betrachten wir den Datensatz `Real Estate Valuation` [1] aus dem UCI Machine Learning Repository. Dieser Datensatz enthält Informationen über Immobilienbewertungen in Taiwan. Ziel unseres Projektes ist es, mithilfe unterschiedlicher Regressions-Verfahren zu überprüfen, ob der Wert pro Fläche abhängig von den diversen angegebenen Variablen ist. Dies würde es ermöglichen, für weitere Immobilien Vorhersagen treffen zu können, bevor diese verkauft werden. Diese Vorhersagen könnten auch genutzt werden, um zu entscheiden, ob eine Immobilie über oder unter dem erwartbaren Wert verkauft wird.

Des Weiteren betrachten wir, ob es möglich ist, die Immobilien in verschiedene geografische Cluster zu unterteilen.

2.2 Vorbereitung Code

Bevor wir damit beginnen können die Daten zu analysieren, müssen wir sicherstellen, dass wir alle nötigen Python-Pakete installiert haben. Hierzu liegt diesem Notebook eine `requirements.txt` Datei bei. Der Python Paket-Manager `PiP` [2] kann anhand dieser Datei die Pakete installieren. Dafür muss in diesem Ordner der folgende Kommando-Zeilen Befehl ausgeführt werden:

```
cmd
pip install -r requirements.txt
```

Der erste Schritt für die Bearbeitung des Datensatzes ist das Importieren der benötigten Bibliotheken. In unserem Fall verwenden wir die folgenden Bibliotheken:

- `pandas` [3] zur allgemeinen Datenverarbeitung
- `numpy` [4] zur numerischen Berechnung
- `matplotlib` [5] zur Visualisierung der Daten
- `scikit-learn` [7] zur Implementierung der Regressionsmodelle

Hierzu genügt es, die Biliotheken zu importieren. Des Weiteren können wir hier festlegen, dass die Plots direkt im Notebook angezeigt werden sollen. Wir können auch festlegen, dass die diversen Warnungen, welche die unterschiedlichen Bibliotheken ausgeben, nicht angezeigt werden sollen.

In [ ]:
```
%matplotlib inline

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

import warnings
warnings.filterwarnings('ignore')
```

Der nächste Schritt ist das Einlesen des Datensatzes. Hierzu können wir die Funktion `ExcelFile` von Pandas verwende. Diese Funktion ermöglicht es uns, die Daten aus einer Excel-Datei in ein `DataFrame` zu laden. Ein `DataFrame` ist eine zweidimensionale Datenstruktur, die von Pandas bereitgestellt wird. Ein `DataFrame` ist vergleichbar mit einer Tabelle in Excel oder einer multivariaten Datenmatrix in R.

In [ ]:
```
data = pd.read_excel("./data.xlsx")
data
```

Out[ ]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.916667	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.916667	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583333	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500000	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833333	5.0	390.56840	5	24.97937	121.54245	43.1
...	...	...	...	...	...	...	...	...
409	410	2013.000000	13.7	4082.01500	0	24.94155	121.50381	15.4
410	411	2012.666667	5.6	90.45606	9	24.97433	121.54310	50.0
411	412	2013.250000	18.8	390.96960	7	24.97923	121.53986	40.6
412	413	2013.000000	8.1	104.81010	5	24.96674	121.54067	52.5
413	414	2013.500000	6.5	90.45606	9	24.97433	121.54310	63.9

414 rows × 8 columns

Als nächstes teilen wir den Datensatz in zwei Teile auf. Den großen Teil des Datensatzes (90%) verwenden wir für die weitere Arbeit in diesem Projekt. Die restlichen 10% des Datensatzes verwenden wir, um am Ende des Projektes die Genauigkeit des Modells zu überprüfen. Um sicherzustellen, dass die Aufteilung reproduzierbar ist, verwenden wir einen vordefinierten Seed. Für diese Aufgabe bietet uns das `DataFrame` die Methode `sample`. Diese Methode gibt einen übergebenen Anteil des DataFrames zurück. Des Weiteren nimmt diese Methode einen Parameter `random_state` an, mit welchem wir die Pseudo-Zufällige Aufteilung reproduzierbar gestalten können.

In [ ]:
```
seed = 1354
data_work = data.sample(frac=0.9, random_state=seed)
len(data_work)
```

Out[ ]: 373

Die Variable `data_work` beinhaltet nun die 90% der Daten welche wir für die weitere Arbeit verwenden. Um nun den Test-Datensatz zu bestimmen, können wir nun die Differenz aus `data_work` und `data` bilden.

In [ ]:
```
data_test = data.drop(data_work.index)
len(data_test)
```

Out[ ]: 41

Wir können nun sicherstellen, dass wir keine Daten "veloren" haben, indem wir die Länge des ursprünglichen Datensatzes mit den beiden erstellen Datensätzen vergleichen.

In [ ]:
```
assert(len(data_work) + len(data_test) == len(data))
```

Die beiden Teil-Datensätze können nun als `csv`-Dateien gespeichert werden. Hierzu können wir die Pandas-Funktion `to_csv` der `Dataframes` verwenden.

In [ ]:
```
data_work.to_csv("data_work.csv", index=False)
data_test.to_csv("data_test.csv", index=False)
```

Nun sind wir bereit für die weitere Analyse der Daten.

3 Beschreibung des Datensatzes

Der Datensatz `Real Estate Valuation` [1] welchen wir betrachten, besteht aus 414 Objekten. Jedes der Objekte bildet eine Bewertung einer Immobilie im Sindian Distrikt aus Taiwan ab. Für jedes Objekt ist der Preis pro normierte Fläche als Wert `Y` angegeben. Dieser Wert wird in der Einheit `10000 New Taiwan Dollar/Ping` angegeben. Hier hat `1Ping` eine Größe von `3.3 Quadratmeter`. Zur Zeit der Verfassung dieses Projektberichtes (26. Feb 2024, 16:00 UTC) haben `10000 New Taiwan Dollar` einen Euro-Wert von `291,97€`.

Umgeformt ist die Einheit dieser Variable also

$$\frac{291.97€}{3.3m^2} \approx 88.48€/m^2$$

Diesen Faktor speichern wir uns für die spätere Verwendung.

In [ ]:
```
# Umrechnungsfaktor von 10000 NT$/ping in Euro/m²
price_factor = 88.48
```

Die weiteren Variablen des Datensatzes sind:

- `X1` *Transaction date* - Datum
- `X2` *House Age* - Alter der Immobilie in Jahren
- `X3` *Distance to nearest MRT Station* - Distanz zu der nächsten `Mass Rapid Transit`-Station in Metern. In diesem Fall die `Metro Taipei` [6].
- `X4` *Number of convenience stores* - Nummer der Convenice-Stores (vgl. Mini-Markt) in Fußnähe
- `X5` *Latitude* - geografisches Breitengrad der Immobilie
- `X6` *Longitude* - geografisches Längengrad der Immobilie

Bei den Variablen `X2`, `X3`, `X5` und `X6` handelt es sich um kontinuierliche Variablen. Die Variablen `X1` und `X4` hingegen sind diskrete Variablen.

Bei der Variable `X1` ist zu beachten, dass das Datum nicht in einem normalen Datumsformat vorliegt. Das Datum wird hier als Zahl angegeben, wobei der Integer-Teil die Jahreszahl und der Dezimal-Teil den Prozentsatz des Jahres darstellt. Hierbei wird allerdings nur der Monat berücksichtigt. Dementsprechend ist der Wert

`2013.25` als `März 2013` zu interpretieren. Da in diesem Datensatz nicht das genaue Datum von Bedeutung ist, sondern nur der Monat, werden wir die Variable `X1` als diskrete Variable behandeln, und nicht wie für ein Datum üblich, als kontinuierliche Variable.

Der Datensatz weist auch eine Variable `No` auf, dies ist allerdings nur ein Index, welcher keine statistisch Bedeutung hat. Dementsprechend werden wir diese in unseren Analysen nicht beachten.

Keine der Variablen weist fehlende Werte auf.

4 Deskriptive Betrachtung des Datensatzes

Werte für die deskriptive Beschreibung des Datensatzes können wir uns mithilfe der Funktion `describe` des `DataFrames` ausgeben lassen.

In [ ]:
```
# deskriptive Statistik explizit nicht für 'ID'-Spalte
data_no_id = data.drop("No", axis=1)

data_no_id.describe(percentiles=[])
```

Out[ ]:

	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	u
count	414.000000	414.000000	414.000000	414.000000	414.00000	414.000000	414.
mean	2013.148953	17.712560	1083.885689	4.094203	24.96903	121.533361	37.
std	0.281995	11.392485	1262.109595	2.945562	0.01241	0.015347	13.
min	2012.666667	0.000000	23.382840	0.000000	24.93207	121.473530	7.
50%	2013.166667	16.100000	492.231300	4.000000	24.97110	121.538630	38.
max	2013.583333	43.800000	6488.021000	10.000000	25.01459	121.566270	117.

Anhand dieser Tabelle können die Werte `Anzahl`, `Mittelwert`, `Standardabweichung`, `Minimum`, `Median` (Äquivalent mit dem 50ten Perzentil) und `Maximum` der Variablen abgelesen werden.

Explizit können wir folgende Werte ablesen:

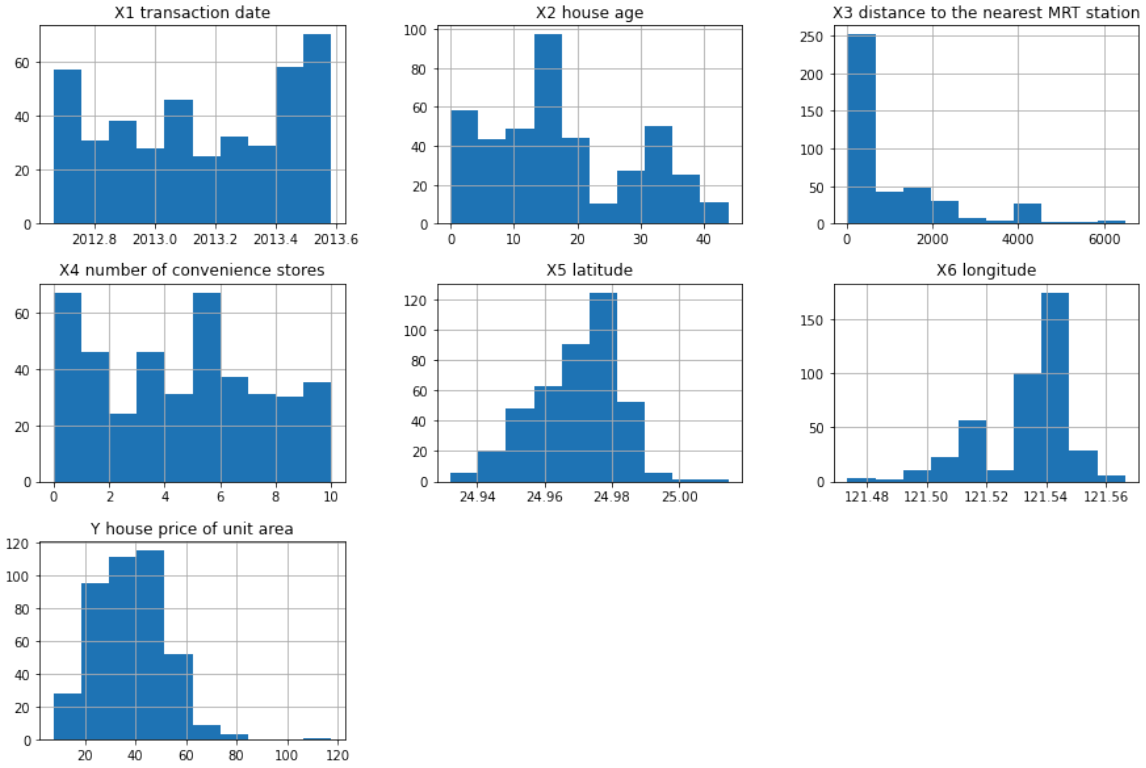
- Das durchschnittliche Alter der Immobilien beträgt 17.712 Jahre
- Die durchschnittliche Distanz zu der nächsten MRT-Station beträgt 1083.885 Meter
- Die durchschnittliche Anzahl der Convenience-Stores in Fußnähe beträgt 4.094
- Der durchschnittliche Preis pro Fläche beträgt $37.98 \frac{10000 NTD}{ping}$ oder $3306.487 €/m^2$
- Der Datensatz beinhaltet Transaktionen aus dem Zeitraum August 2012 bis Juli 2013

- Der Mittelwert und der Median der geografischen Koordinaten sind nahezu identisch.

4.1 Histogramme des Datensatzes

Die deskriptive Beschreibung des Datensatzes gibt uns einen ersten Eindruck über die Verteilung der Daten. Diese Daten können wir auch visuell darstellen. Hierzu können wir die Funktion `hist` des `DataFrames` verwenden. Diese Funktion erstellt für jede Variable ein Histogramm.

In [ ]:
```
hist = data_no_id.hist(figsize=(15, 10))
```



Anhand der Histogramme können wir weitere Informationen über die Verteilung der Daten ablesen. Die Variable `X3` weist eine linkssteile Verteilung auf. Die Variablen `X5` und `X6` weisen eine spitze Verteilung auf. Des Weiteren ist die Varianz dieser Werte sehr gering. Wir können auch ablesen, dass die historische Verteilung der Transaktionen in der Mitte des Jahres 2013 am höchsten war.

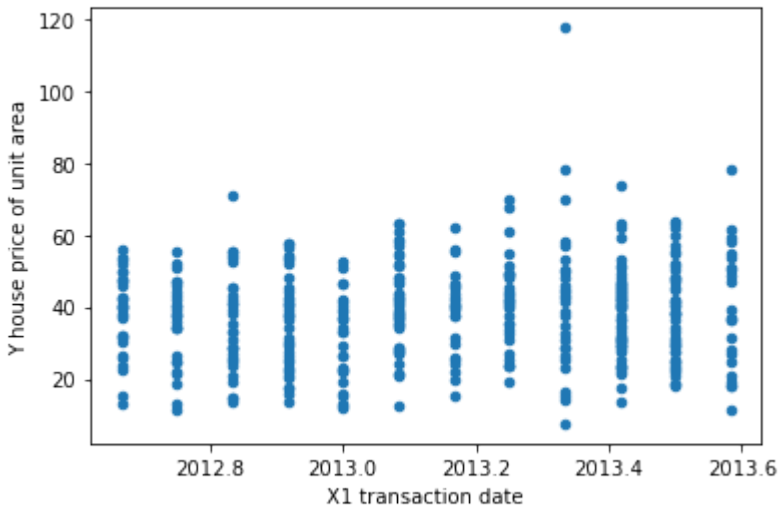
4.2 Scatter-Plots des Datensatzes

Alternativ hierzu können wir die verschiedenen Variablen auch in einem Scatter-Plot darstellen. Hierzu können wir eine Funktion definieren, welche eine der Variablen gegen unsere Zielvariable `Y` aufträgt. Hierbei können wir die Funktion `scatter` des `DataFrames` verwenden.

In [ ]:
```
def plot_scatter(axis):
    data_no_id.plot.scatter(x=axis, y="Y house price of unit area", figsi
```

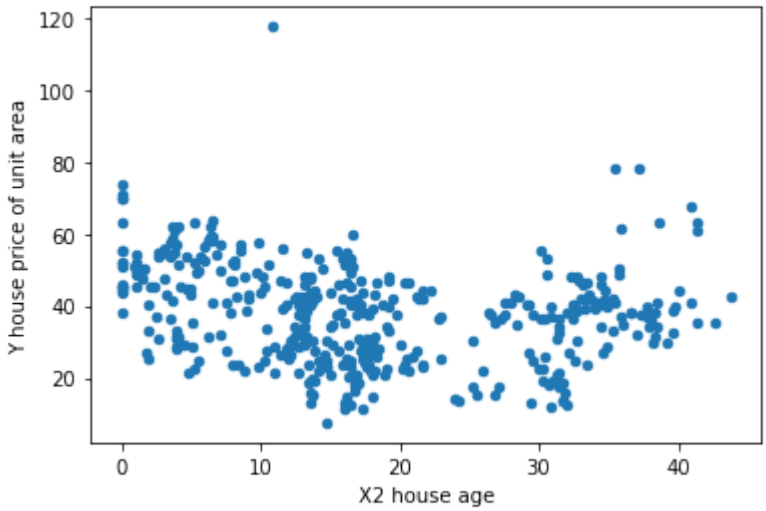
Die Funktion `plot_scatter` können wir nun nutzen, um uns die individuellen Scatter-Plots der Variablen gegen die Zielvariable `Y` anzeigen zu lassen.

In [ ]:
```
plot_scatter("X1 transaction date")
```



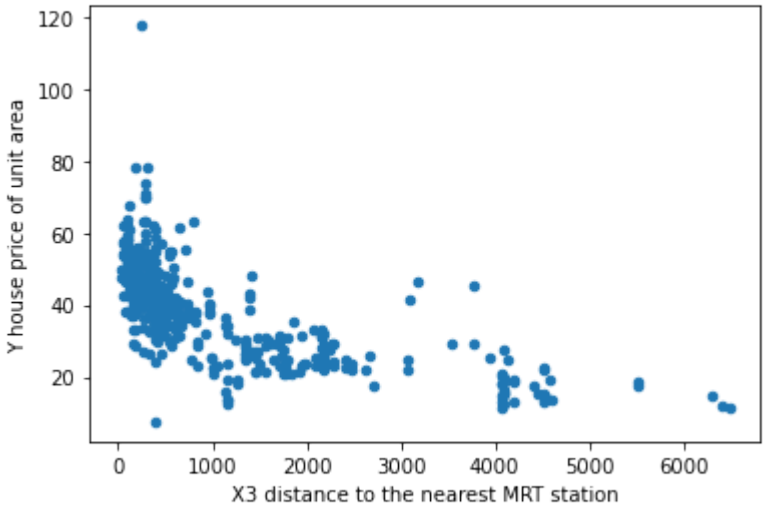
Bei der Betrachtung des Scatter-Plots von `X1` gegen `Y` lässt sich beobachten, dass die Werte von `Y` über die Zeit hinweg leicht ansteigen. Dies könnte darauf hindeuten, dass die Immobilienpreise in dem vom Datensatz abgebildeten Zeitraum gestiegen sind.

In [ ]:
```
plot_scatter("X2 house age")
```



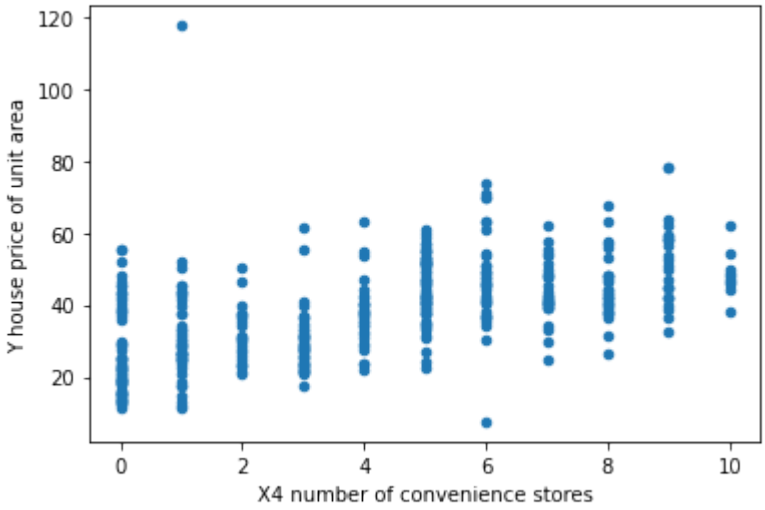
Der Scatter-Plot von `X2` gegen `Y` zeigt, dass die Immobilienpreise mit steigendem Alter der Immobilie sinken. Es lässt sich allerdings auch erkennen, dass die Streuung der Werte bei jüngeren Immobilien, aber auch bei vergleichsweise alten Immobilien (beginnend ab ~30 Jahre), sehr hoch ist.

In [ ]:
```
plot_scatter("X3 distance to the nearest MRT station")
```



Das Auftragen von `X3` gegen `Y` zeigt, dass die Immobilienpreise mit steigender Distanz zur nächsten MRT-Station sinken. Hierbei ist zu beachten, dass die Streuung der Werte bei geringen Distanzen zur MRT-Station sehr hoch ist. Es lässt sich auch erkennen, dass ein Großteil der Immobilien in einem Radius von 1000 Metern um eine MRT-Station liegen.

In [ ]:
```
plot_scatter("X4 number of convenience stores")
```

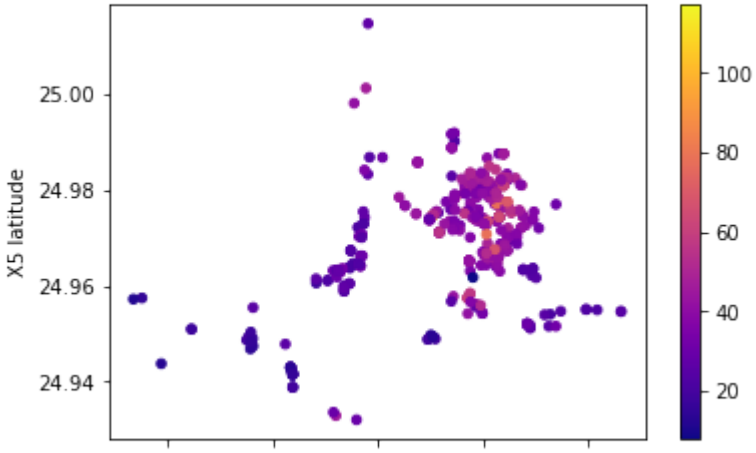


Bei dem Plotten der Zahl der Convenience-Stores in Fußnähe (`X4`) gegen den Zielwert (`Y`) lässt sich erkennen, dass die Immobilienpreise mit steigender Anzahl der Convenience-Stores in Fußnähe steigen. Es ist allerdings auch zu erkennen, dass die Streuung der Werte vergleichsweise hoch ist.

Für die Betrachtung der geografischen Koordinaten `X5` und `X6` gegen `Y` verwenden wir einen gemeinsamen Scatter-Plot. Hier sind die Koordinaten auf den jeweiligen Achsen aufgetragen, und der Preis pro Fläche wird durch die Farbe der Punkte dargestellt.

Hierzu können wir erneut die Funktion `scatter` verwenden, allerdings können wir hier die Farbe der Punkte durch den Parameter `c` bestimmen. Hierbei können wir die Variable `Y` übergeben, um die Punkte nach dem Preis pro Fläche zu färben.

In [ ]:
```
ax = data_no_id.plot.scatter(y="X5 latitude", x="X6 longitude", c=(data_n
```



Wenn wir die geografischen Koordinaten `X5` und `X6` gegen den Zielwert `Y` auftragen, lässt sich erkennen, dass die Immobilienpreise in der Nähe des Breitengrades 24.97 und des Längengrades 121.54 erkennbar höher sind als in anderen Regionen. Des Weiteren lässt sich eine Ballung von Datenpunkten in diesem Bereich erkennen.

5 Regressionsmodelle

Im folgenden schauen wir uns an, in wie weit es möglich ist, mithilfe verschiedener Regressions-Modelle den Preis pro Fläche vorherzusagen. Hierzu verwenden wir die Bibliothek `scikit-learn` [7]. Diese Bibliothek bietet uns diverse Funktionen zur Erstellung von Regressions-Modellen. Von nun an verwenden wir den Datensatz `data_work` für die Erstellung der Modelle. Den Datensatz `data_test` verwenden wir am Ende des Projektes, um die Genauigkeit der verschiedenen Modelle zu überprüfen.

Für die weitere Betrachtung der Datensatzes mithilfe der verschiedenen Regressions-Modelle schauen wir uns alle Objekte des Trainings-Datensatzes an. Die verschiedenen Modelle können wir dann mithilfe des Test-Datensatzes überprüfen.

Die Regressions-Modelle werden sich mit den Variablen `X1`, `X2`, `X3` und `X4` beschäftigen. Die Variablen `X5` und `X6` werden wir gesondert im Anschluss an die Regressions-Modelle betrachten. Als erstes teilen wir die beiden Datensätze in die unabhängigen Variablen `X1` bis `X4` und die abhängige Variable `Y` auf. Hierzu können wir die Funktion `drop` des `DataFrames` verwenden. Die Funktion gibt uns eine Kopie des `DataFrames` zurück, in welcher eine angegebene Spalte entfernt wurden.

In [ ]:
```
reg_data = data_work.drop("No", axis=1)
reg_data.drop("X5 latitude", axis=1, inplace=True)
reg_data.drop("X6 longitude", axis=1, inplace=True)

reg_val = reg_data["Y house price of unit area"]
reg_vars = reg_data.drop("Y house price of unit area", axis=1)
```

```
test_data = data_test.drop("No", axis=1)
test_val = test_data["Y house price of unit area"]
test_vars = test_data.drop("Y house price of unit area", axis=1)
test_vars.drop("X5 latitude", axis=1, inplace=True)
test_vars.drop("X6 longitude", axis=1, inplace=True)
```

5.1 Eindimensionale lineare Regression

Als erstes betrachten wir eindimensionale lineare Regression.

Die eindimensionale lineare Regression ist ein einfaches Regressions-Modell, welches die Beziehung zwischen einer unabhängigen Variable und einer abhängigen Variable beschreibt. Hierbei wird angenommen, dass die abhängige Variable linear von der unabhängigen Variable abhängt. Das Modell kann durch die Gleichung

$$Y = b + m * X + f$$

beschrieben werden. Hierbei ist Y die abhängige Variable, X die unabhängige Variable, b der y-Achsenabschnitt, m die Steigung der Linie und f der Fehler.

Hierbei überprüfen wir, in wie weit der Preis pro Fläche von den unterschiedlichen Variablen abhängig ist. Als ersten Schritt dazu können wir die Korrelation der Variablen untereinander betrachten. Hierzu können wir die Funktion `corr` des `DataFrames` verwenden. Pandas verwendet hierbei die Pearson-Korrelation. Diese Funktion gibt uns die Korrelation der Variablen untereinander zurück.

In [ ]:
```
reg_data.corr()
```

Out[ ]:

	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	Y house price of unit area
X1 transaction date	1.000000	0.011924	0.079257	-0.014421	0.071762
X2 house age	0.011924	1.000000	0.025062	0.051772	-0.193963
X3 distance to the nearest MRT station	0.079257	0.025062	1.000000	-0.596669	-0.668799
X4 number of convenience stores	-0.014421	0.051772	-0.596669	1.000000	0.561066
Y house price of unit area	0.071762	-0.193963	-0.668799	0.561066	1.000000

Anhand dieser Tabelle können wir die Korrelation der Variablen untereinander ablesen. In unserem Fall interessiert uns die Korrelation der Variablen mit der Variable `Y`. Hierbei können wir ablesen, dass die Variable `X3` mit *0.669* die größte absolute Korrelation mit der Variable `Y` aufweist. Die Variablen `X4` weißt mit einem Korrelationskoeffizienten von `0.56` auf eine mittel-starke Korrelation hin. Die

Variablen `X1` und `X2` weisen hingegen nur marginale Korrelations-Koeffizienten von *0.07* und *0.19* auf.

Insgesamt lassen sich anhand der Korrelationen der Variablen bereits einige Aussagen treffen:

- Das Alter der Immobilie scheint negativ mit dem Preis pro Fläche korreliert zu sein. Die Korrelation ist allerdings vergleichsweise gering.
- Die Nummer der Convenience-Stores in Fußnähe scheint positiv mit dem Preis pro Fläche korreliert zu sein. Die Korrelation ist hierbei weder besonders stark noch besonders schwach.
- Die Distanz zu der nächsten MRT-Station scheint negativ mit dem Preis pro Fläche korreliert zu sein. Diese Korrelation weist den stärksten Wert auf.
- Das Transaktionsdatum weist beinah keine Korrelation mit dem Preis pro Fläche auf.

Nach der Betrachtung der Korrelation lässt sich vermuten, dass ein simples eindimensionales lineares Regressions-Modell mit den Variablen `X3` und `X4` die besten Ergebnisse liefern wird. Diese Modelle können wir nun mithilfe der Funktion `LinearRegression` aus der Bibliothek `scikit-learn` erstellen.

Als erstes importieren wir dafür die nötigen Funktionen der Bibliothek `scikit-learn`. Dannach können wir die Funktion `LinearRegression` verwenden, um die Modelle zu erstellen. Wir erstellen jeweils ein Modell pro Variable die wir betrachten wollen.

In [ ]:
```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error

regrX3 = linear_model.LinearRegression()
regrX4 = linear_model.LinearRegression()
```

Als nächstes können wir die beiden Modelle anhand des Trainings-Datensatzes trainieren. Hierzu können wir die Funktion `fit` der Modelle verwenden. Diese Funktion nimmt die unabhängigen Variablen und die abhängige Variable als Parameter an. Hierbei ist zu beachten, dass die unabhängige Variable als zweidimensionales Array übergeben werden muss. Hierzu können wir die Funktion `reshape` verwenden, um die richtige Form zu erhalten.

In [ ]:
```
regrX3.fit(reg_vars["X3 distance to the nearest MRT station"].values.resh
regrX4.fit(reg_vars["X4 number of convenience stores"].values.reshape(-1,
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

Nachdem wir die Modelle nun trainiert haben, können wir die Modelle anhand des Test-Datensatzes überprüfen. Hierzu können wir die Funktion `predict` der Modelle verwenden. Das Ergebnis dieser Funktion können wir dann mit den tatsächlichen

Werten vergleichen. Um diesen Vergleich zu vereinfachen, können wir die Funktion `mean_squared_error` der Bibliothek `scikit-learn` verwenden. Diese Funktion implementiert den mittleren quadratischen Fehler. Dieser Wert gibt uns an, wie weit die vorhergesagten Werte von den tatsächlichen Werten entfernt sind.

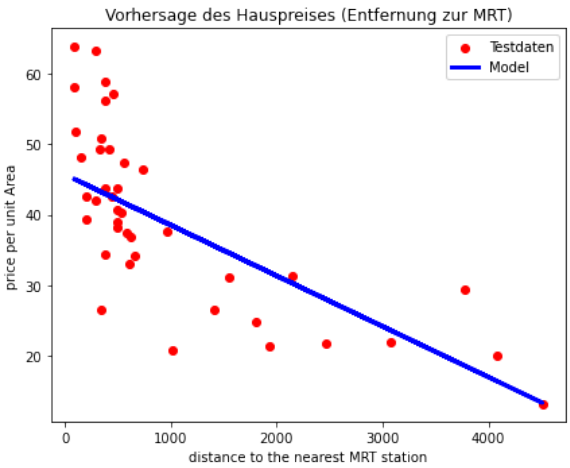
In [ ]:
```
y_predX3 = regrX3.predict(test_vars["X3 distance to the nearest MRT stati
y_predX4 = regrX4.predict(test_vars["X4 number of convenience stores"].va

mse_1dlinregX3 = mean_squared_error(test_val, y_predX3)
mse_1dlinregX4 = mean_squared_error(test_val, y_predX4)

print("Mean squared error X3: %.2f"
      % mse_1dlinregX3)

print("Mean squared error X4: %.2f"
      % mse_1dlinregX4)
```

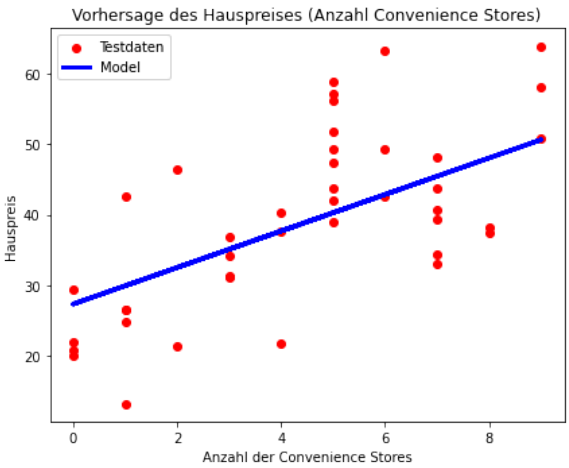
```
Mean squared error X3: 76.45
Mean squared error X4: 88.77
```

Anhand dieser Resultate können wir ablesen, dass das Modell der Variable `X3` einen geringeren mittleren quadratischen Fehler aufweist als das Modell der Variable `X4`. Dies bedeutet, dass das Modell der Variable `X3` die tatsächlichen Werte besser vorhersagen kann als das Modell der Variable `X4`.

Der Vorteil einer eindimensionalen linearen Regression ist, dass wir sie in einem Scatter-Plot darstellen können. Im folgenden stellen wir die Test-Daten und die Vorhersagen der Modelle in einem Scatter-Plot dar.

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(12, 5))

# Plot 1: Variable X3
ax1.scatter(test_vars["X3 distance to the nearest MRT station"], test_val
ax1.plot(test_vars["X3 distance to the nearest MRT station"], y_predX3, c
ax1.legend(["Testdaten", "Model"])
ax1.set_title("Vorhersage des Hauspreises (Entfernung zur MRT)")  # More
ax1.set_xlabel("distance to the nearest MRT station")
ax1.set_ylabel("price per unit Area")

# Plot 2: Variable X4
ax2.scatter(test_vars["X4 number of convenience stores"], test_val, color
ax2.plot(test_vars["X4 number of convenience stores"], y_predX4, color='b
ax2.legend(["Testdaten", "Model"])
ax2.set_title("Vorhersage des Hauspreises (Anzahl Convenience Stores)")
ax2.set_xlabel("Anzahl der Convenience Stores")
ax2.set_ylabel("Hauspreis")

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





Die Diagramme zeigen die Vorhersagen der linearen Modelle und die Verteilung der Testdaten. Anhand dieser Diagramme lässt sich bereits erkennen, dass eine eindimensionale lineare Regression nicht ausreicht, um den Preis pro Fläche akkurat vorherzusagen.

5.2 Multidimensionale lineare Regression

Der nächste Schritt ist die Betrachtung einer multidimensionalen linearen Regression. Hierbei handelt es sich um ein Regressions-Modell, welches die Beziehung zwischen mehreren unabhängigen Variablen und einer abhängigen Variable beschreibt. Die Multidimensionale lineare Regression kann durch die Gleichung

$$Y = b + m_1 * X_1 + m_2 * X_2 + m_\dots * X_\dots + m_n * X_n + f$$

beschrieben werden.

In diesem Fall verwenden wir unsere 4 Variablen `X1`, `X2`, `X3` und `X4` als unabhängige Variablen. Der Code für die Erstellung und das Trainieren der Modelle ist hierbei nahezu identisch mit dem Code für die eindimensionale lineare Regression. Der einzige Unterschied ist, dass wir alle 4 Variablen für das Trainieren des Models verwenden.

In [ ]:
```
multiReg = linear_model.LinearRegression()

multiReg.fit(reg_vars, reg_val)

multiPred = multiReg.predict(test_vars)

mse_ndlinreg = mean_squared_error(test_val, multiPred)

print("Mean squared error Multi: %.2f"
      % mse_ndlinreg)
```

```
Mean squared error Multi: 45.43
```

Mit einem mittleren quadratischen Fehler von `45.43` ist das Modell der multidimensionalen linearen Regression besser als beide Modelle der eindimensionalen linearen Regression. Aufgrund der höheren Dimensionalität des

Modells ist es allerdings nicht mehr möglich, das Modell in einem Scatter-Plot darzustellen.

5.3 Eindimensionale nichtlineare Regression

Nachdem wir unter anderem die eindimensionale lineare Regression untersucht haben, erweitern wir unseren Fokus nun auf die eindimensionale nichtlineare Regression. Diese Art der Regression ist besonders nützlich, wenn die Beziehung zwischen den unabhängigen Variablen und der abhängigen Variable nicht linear ist, was in vielen realen Szenarien der Fall sein kann. Ein nichtlineares Modell kann in solchen Fällen eine bessere Anpassung und höhere Vorhersagegenauigkeit als lineare Modelle bieten.

Die eindimensionale nichtlineare Regression lässt sich durch eine Vielzahl von Modellen darstellen, einschließlich, aber nicht beschränkt auf Polynomfunktionen höherer Ordnung sowie exponentielle und logarithmische Funktionen. Die Wahl des spezifischen Modells richtet sich nach der Natur der Daten und der spezifischen Beziehung, die sie abbilden. Ein einfaches Beispiel für ein solches nichtlineares Modell ist eine quadratische Funktion, die durch die Gleichung

$$Y = a + bX + cX^2 + f$$

ausgedrückt wird, wobei Y die abhängige Variable, X eine unabhängige Variable, a der y-Achsenabschnitt, b und c Koeffizienten, die die Kurve der Funktion bestimmen, und f der Fehlerterm ist.

Um eine nichtlineare Regression auf unsere Datensätze anzuwenden, betrachten wir spezifische Variablen, von denen wir annehmen, dass ihre Beziehung zum Hauspreis pro Flächeneinheit nichtlinear ist. Basierend auf vorherigen Analysen und der Untersuchung der Daten könnten wir beispielsweise die Beziehungen sowohl der Distanz zur nächsten MRT-Station (X3), als auch der Anzahl der Convenience Stores in der Nähe (X4) zum Hauspreis pro Flächeneinheit als potenziell nichtlinear identifizieren.

Zur Umsetzung der nichtlinearen Regression verwenden wir die `PolynomialFeatures`-Komponente aus der `scikit-learn`-Bibliothek, um unsere unabhängigen Variablen in Polynome höherer Ordnung zu transformieren. Dieser Schritt wird gefolgt von einem linearen Modell, das auf die transformierten Daten angepasst wird. Die Transformation ermöglicht es dem linearen Modell, nichtlineare Beziehungen zwischen den unabhängigen Variablen und der abhängigen Variable effektiv zu erfassen und zu modellieren.

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Erstellen eines Polynom-Modells 2. Grades für X3
poly_model_X3 = make_pipeline(PolynomialFeatures(degree=2), LinearRegress
```

```
# Anpassen des Modells an die Daten für X3
poly_model_X3.fit(reg_vars["X3 distance to the nearest MRT station"].valu

# Vorhersagen mit dem Test-Datensatz für X3
y_pred_poly_X3 = poly_model_X3.predict(test_vars["X3 distance to the near

# Berechnung des mittleren quadratischen Fehlers für X3
mse_polyregX3 = mean_squared_error(test_val, y_pred_poly_X3)
print("Mean squared error for Polynomial Model X3: %.2f" % mse_polyregX3)
```

```
Mean squared error for Polynomial Model X3: 63.65
```

In [ ]:
```
# Erstellen eines Polynom-Modells 2. Grades für X4
poly_model_X4 = make_pipeline(PolynomialFeatures(degree=2), LinearRegress

# Anpassen des Modells an die Daten für X4
poly_model_X4.fit(reg_vars["X4 number of convenience stores"].values.resh

# Vorhersagen mit dem Test-Datensatz für X4
y_pred_poly_X4 = poly_model_X4.predict(test_vars["X4 number of convenienc

# Berechnung des mittleren quadratischen Fehlers für X4
mse_polyregX4 = mean_squared_error(test_val, y_pred_poly_X4)
print("Mean squared error for Polynomial Model X4: %.2f" % mse_polyregX4)
```

```
Mean squared error for Polynomial Model X4: 86.28
```

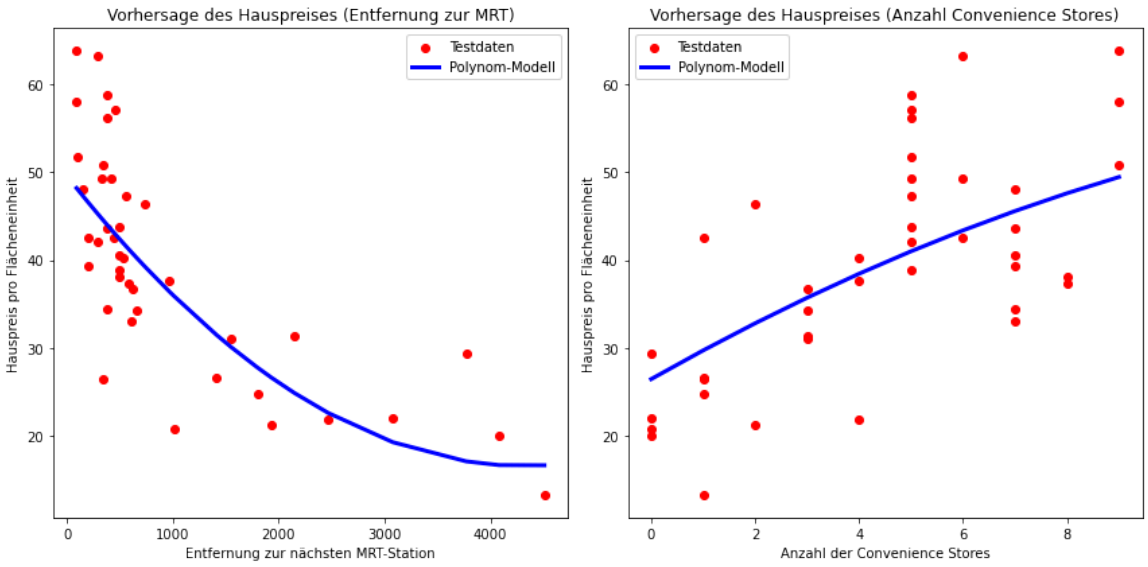
Durch diesen Ansatz gelingt es uns, nichtlineare Muster in den Daten zu erkennen und dadurch präzisere Vorhersagen über den Hauspreis zu machen, die sowohl auf der Distanz zur nächsten MRT-Station (X3) als auch auf der Anzahl der Convenience Stores in der Nähe (X4) basieren. Die Flexibilität, die der Grad des Polynoms bietet, erlaubt es uns, die Modellkomplexität an die spezifischen Daten anzupassen und somit eine optimale Anpassung und Vorhersageleistung zu erreichen.

Um die Ergebnisse zu veranschaulichen, stellen wir die Testdaten und die Vorhersagen der nichtlinearen Modelle in Scatter-Plots dar. Diese Visualisierung hilft uns nicht nur, die Passgenauigkeit des Modells zu bewerten, sondern ermöglicht es uns auch, die Effektivität des nichtlinearen Ansatzes im Vergleich zu linearen Modellen direkt zu beobachten.

Diese Beispiele unterstreichen die Stärke der eindimensionalen nichtlinearen Regression bei der Modellierung komplexer Beziehungen zwischen Variablen. Sie demonstrieren besonders den Nutzen dieser Methode in Fällen, in denen eine lineare Annäherung nicht ausreicht, um die Dynamik zwischen den Variablen und der Zielgröße adäquat zu erfassen.

In [ ]:
```
# Plot für X3 - Entfernung zur nächsten MRT-Station
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1) # 1 row, 2 columns, first subplot
plt.scatter(test_vars["X3 distance to the nearest MRT station"], test_val
# Für eine glatte Linie sortieren wir die Werte
sorted_index_X3 = np.argsort(test_vars["X3 distance to the nearest MRT st
plt.plot(np.array(test_vars["X3 distance to the nearest MRT station"])[so
plt.legend(["Testdaten", "Polynom-Modell"])
plt.title("Vorhersage des Hauspreises (Entfernung zur MRT)")
```

```
plt.xlabel("Entfernung zur nächsten MRT-Station")
plt.ylabel("Hauspreis pro Flächeneinheit")

# Plot für X4 - Anzahl der Convenience Stores
plt.subplot(1, 2, 2) # 1 row, 2 columns, second subplot
plt.scatter(test_vars["X4 number of convenience stores"], test_val, color
# Für eine glatte Linie sortieren wir die Werte
sorted_index_X4 = np.argsort(test_vars["X4 number of convenience stores"]
plt.plot(np.array(test_vars["X4 number of convenience stores"])[sorted_in
plt.legend(["Testdaten", "Polynom-Modell"])
plt.title("Vorhersage des Hauspreises (Anzahl Convenience Stores)")
plt.xlabel("Anzahl der Convenience Stores")
plt.ylabel("Hauspreis pro Flächeneinheit")

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



5.4 Mehrdimensionale nichtlineare Regression

Nachdem wir die Anwendung der eindimensionalen nichtlinearen Regression betrachtet haben, erweitern wir unseren Ansatz nun auf die mehrdimensionale nichtlineare Regression. Dieser Ansatz ist besonders nützlich, wenn die Beziehung zwischen mehreren unabhängigen Variablen und der abhängigen Variable komplex und möglicherweise nichtlinear ist. Durch die Einbeziehung mehrerer Variablen können wir ein genaueres und umfassenderes Modell der zugrundeliegenden Beziehungen erstellen.

Die mehrdimensionale nichtlineare Regression erweitert das Konzept der nichtlinearen Regression, indem sie Polynomfunktionen auf ein multivariates Setting anwendet. Hierbei werden polynomische Merkmale nicht nur für eine einzelne Variable, sondern für alle unabhängigen Variablen im Datensatz erstellt. Dies ermöglicht die Modellierung von Interaktionen zwischen den Variablen sowie nichtlinearen Effekten.

Um ein mehrdimensionales nichtlineares Regressionsmodell zu implementieren, verwenden wir erneut die `PolynomialFeatures`-Funktion aus der `scikit-learn`-Bibliothek. Diesmal wenden wir sie jedoch auf den gesamten Satz

unabhängiger Variablen an, um ein umfassendes Set polynomischer Merkmale zu erzeugen:

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures

# Polynomische Merkmale erstellen
poly = PolynomialFeatures(degree=2)
X_train_poly = poly.fit_transform(reg_vars)
```

Durch die Anwendung der `PolynomialFeatures`-Funktion mit `degree=2` transformieren wir unsere unabhängigen Variablen in ein Format, das nicht nur die ursprünglichen Variablen, sondern auch ihre quadrierten Werte und die Interaktionsterme zwischen ihnen umfasst. Diese Transformation erhöht die Dimensionalität des Feature-Raums, ermöglicht es aber dem linearen Modell, nichtlineare Beziehungen zu erfassen.

Nachdem wir die polynomischen Merkmale erstellt haben, folgt der nächste Schritt, ein lineares Regressionsmodell auf diesen erweiterten Feature-Satz anzuwenden:

In [ ]:
```
# Lineares Regressions-Modell erstellen
model = linear_model.LinearRegression()

# Modell trainieren
model.fit(X_train_poly, reg_val)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

Durch das Training des linearen Regressionsmodells auf dem polynomisch erweiterten Datensatz sind wir in der Lage, ein komplexeres Modell zu erstellen, das die nichtlinearen Beziehungen zwischen den Variablen und der Zielvariable berücksichtigt.

Um die Leistung des Modells zu bewerten, verwenden wir den mittleren quadratischen Fehler (Mean Squared Error, MSE) auf einem Testdatensatz:

In [ ]:
```
mse_polyreg = mean_squared_error(test_val, model.predict(poly.fit_transfo

print("Mean squared error MultiPoly: %.2f"
      % mse_polyreg)
```

```
Mean squared error MultiPoly: 34.91
```

Der resultierende mittlere quadratische Fehler von 34.91 zeigt die Genauigkeit des Modells bei der Vorhersage der Zielvariable auf Basis der Testdaten. Ein niedrigerer MSE-Wert deutet auf eine höhere Vorhersagegenauigkeit des Modells hin.

Die Anwendung der mehrdimensionalen nichtlinearen Regression ermöglicht eine tiefere und genauere Analyse der Beziehungen zwischen mehreren Variablen und der Zielgröße. Durch die Berücksichtigung von Interaktionen und nichtlinearen Effekten können wir ein realistischeres Modell der Realität erstellen, was besonders in

komplexen Anwendungsfällen wie der Vorhersage von Immobilienpreisen von großem Wert ist.

5.5 Random Forrest Regression

Die Random Forest Regression ist eine Erweiterung des Random Forest Algorithmus für Regressionsprobleme. Sie gehört zu den Ensemble-Learning-Methoden, bei denen mehrere Entscheidungsbäume während des Trainingsprozesses erstellt und kombiniert werden, um ein robusteres und genaues Modell zu erzeugen. Die Kernidee hinter der Random Forest Regression ist, die Vorhersagegenauigkeit zu verbessern und das Risiko von Überanpassungen (Overfitting) zu minimieren, indem die Vorhersagen mehrerer Entscheidungsbäume gemittelt werden.

Der Random Forest Algorithmus folgt im Wesentlichen diesen Schritten:

1. **Bootstrap-Auswahl**: Für jeden Baum im Wald wird eine zufällige Stichprobe der Trainingsdaten mit Zurücklegen gezogen (Bootstrap-Sampling). Dies bedeutet, dass einige Beobachtungen in einer Stichprobe mehrmals vorkommen können, während andere möglicherweise gar nicht ausgewählt werden.

2. **Baumwachstum mit Feature-Zufälligkeit**: Beim Erstellen jedes Baums und an jedem Split (Verzweigungspunkt) im Baum wird eine zufällige Auswahl von Merkmalen (Features) getroffen, aus denen das beste Feature zur Teilung des Datensatzes ausgewählt wird. Diese Methode trägt dazu bei, die Korrelation zwischen den Bäumen im Wald zu verringern, was zu einer höheren Gesamtgenauigkeit führt.

3. **Vorhersage**: Nachdem der Wald von Bäumen erstellt wurde, wird für eine Regressionsvorhersage der Durchschnittswert der Vorhersagen aller Bäume im Wald für den gegebenen Datenpunkt berechnet.

Die Vorteile der Random Forest Regression umfassen:

- **Hohe Genauigkeit**: Durch die Kombination der Vorhersagen mehrerer Bäume tendiert Random Forest dazu, genauer zu sein als einzelne Entscheidungsbäume.
- **Robustheit gegenüber Überanpassung**: Da der Algorithmus auf vielen unkorrelierten Bäumen basiert, ist er weniger anfällig für Überanpassung an das Trainingsset, besonders wenn es eine große Anzahl von Bäumen gibt.
- **Flexibilität**: Kann für numerische und kategoriale Daten verwendet werden und benötigt im Allgemeinen wenig Vorbereitung der Daten.
- **Wichtigkeitsbewertung der Merkmale**: Random Forest kann die Wichtigkeit der einzelnen Merkmale für die Vorhersage ermitteln, was nützlich für die Feature-Auswahl sein kann.

Allerdings gibt es auch einige Nachteile, wie z.B. die Tendenz, bei sehr rauschenden Daten überanpassen zu können, und die Tatsache, dass das Modell aufgrund der

vielen Bäume oft sehr groß und komplex sein kann, was es schwieriger macht, zu interpretieren und zu verstehen, wie Entscheidungen getroffen werden.

In [ ]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.inspection import permutation_importance

y_train = np.ravel(reg_val)
y_test = np.ravel(test_val)

params = {
    "n_estimators": 500,
    "max_depth": 4,
    "min_samples_split": 5,
    "warm_start":True,
    "oob_score":True,
    "random_state": 42,
}

reg =RandomForestRegressor(**params)

reg.fit(reg_vars, y_train)

y_pred = reg.predict(test_vars)
mse_randomforrestreg = mean_squared_error(y_test, y_pred, squared=False)

print("Mean squared error RandomForrest: %.2f"
       % mse_randomforrestreg)
```

```
Mean squared error RandomForrest: 5.76
```

Dieser Code demonstriert die Anwendung des Random Forest Regressionsmodells auf den Immobilienbewertungsdatensatz. Hier ist eine schrittweise Erklärung des Codes:

1. **Vorbereitung der Daten**:
   - `y_train` und `y_test` werden als eindimensionale Arrays vorbereitet, die die Zielwerte für Trainings- und Testdatensätze enthalten.
2. **Parameterdefinition**:
   - Ein Dictionary `params` wird definiert, dass verschiedene Hyperparameter für das Random Forest Regressionsmodell enthält:
     - `n_estimators`: Die Anzahl der Bäume im Wald, hier 500.
     - `max_depth`: Die maximale Tiefe der Bäume, hier 4.
     - `min_samples_split`: Die minimale Anzahl von Samples, die notwendig sind, um einen Knoten zu teilen, hier 5.
     - `warm_start`: Wenn auf `True` gesetzt, erlaubt die Wiederverwendung der Lösung des vorherigen Aufrufs, um mehr Bäume zum Ensemble hinzuzufügen.
     - `oob_score`: Wenn auf `True` gesetzt, wird die Out-of-Bag-Schätzung verwendet, um die Generalisierungsgenauigkeit zu bewerten.
     - `random_state`: Ein Seed-Wert für Zufallszahlengenerator, hier 42, um Reproduzierbarkeit zu gewährleisten.

3. **Modelltraining**:
    - Ein `RandomForestRegressor`-Objekt `reg` wird mit den zuvor definierten Parametern initialisiert und auf den Trainingsdaten trainiert.
4. **Vorhersage und Leistungsbewertung**:
    - Das Modell wird verwendet, um Vorhersagen für die Testdatensatzvariablen `test_vars` zu machen.
    - Der mittlere quadratische Fehler (Mean Squared Error, MSE) zwischen den vorhergesagten Werten `y_pred` und den tatsächlichen Zielwerten `y_test` des Testdatensatzes wird berechnet. Der MSE ist eine gängige Metrik zur Bewertung der Leistung von Regressions-Modellen, die die durchschnittliche Größe der Fehler zwischen vorhergesagten und tatsächlichen Werten misst. Ein MSE-Wert von 5.7563 deutet darauf hin, dass die durchschnittliche Vorhersage-Abweichung des Modells in der Einheit des Zielwerts liegt, was bedeutet, dass die Vorhersagen des Modells im Durchschnitt etwa 5.7563 Einheiten vom tatsächlichen Wert entfernt sind.

Der Outputwert von 5.7563 als MSE ist ein direktes Maß für die Vorhersagegenauigkeit des Random Forest Regressions-Modells auf dem Immobilienbewertungsdatensatz. Ein niedrigerer MSE-Wert deutet auf eine höhere Genauigkeit des Modells hin.

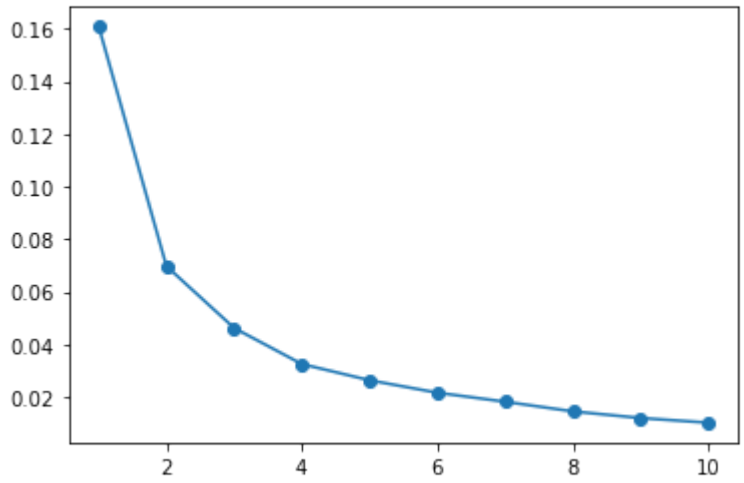
6. Cluster-Analyse von Nachbarschaften

Da der Datensatz auch geografische Koordinaten beinhaltet, können wir auch eine Cluster-Analyse durchführen. Hierbei können wir die geografischen Koordinaten der Immobilien in Cluster einteilen. Dies kann uns Einblicke dazu geben, ob es spezifische Regionen gibt, in denen die Immobilienpreise besonders hoch oder besonders niedrig sind. Hierzu können wir erneut die Bibliothek `scikit-learn` verwenden. Die Methode `KMeans` kann verwendet werden, um eine K-Means-Cluster-Analyse durchzuführen. In unserem Fall wollen wir die Cluster-Analyse für die Features `X5` und `X6` durchführen.

Um die optimale Menge an Clustern zu bestimmen können wir die `Elbow-Method` verwenden. Diese Methode berechnet die Summe der quadratischen Abweichungen der Datenpunkte zu dem jeweiligen Cluster-Zentrum. Wenn man diese Werte gegen die Anzahl der Cluster aufträgt, kann man an der Kurve die optimale Cluster-Anzahl ablesen. Hierfür definieren wir als erstes eine Funktion, welche diesen Wert für eine gegebene Anzahl an Clustern berechnet.

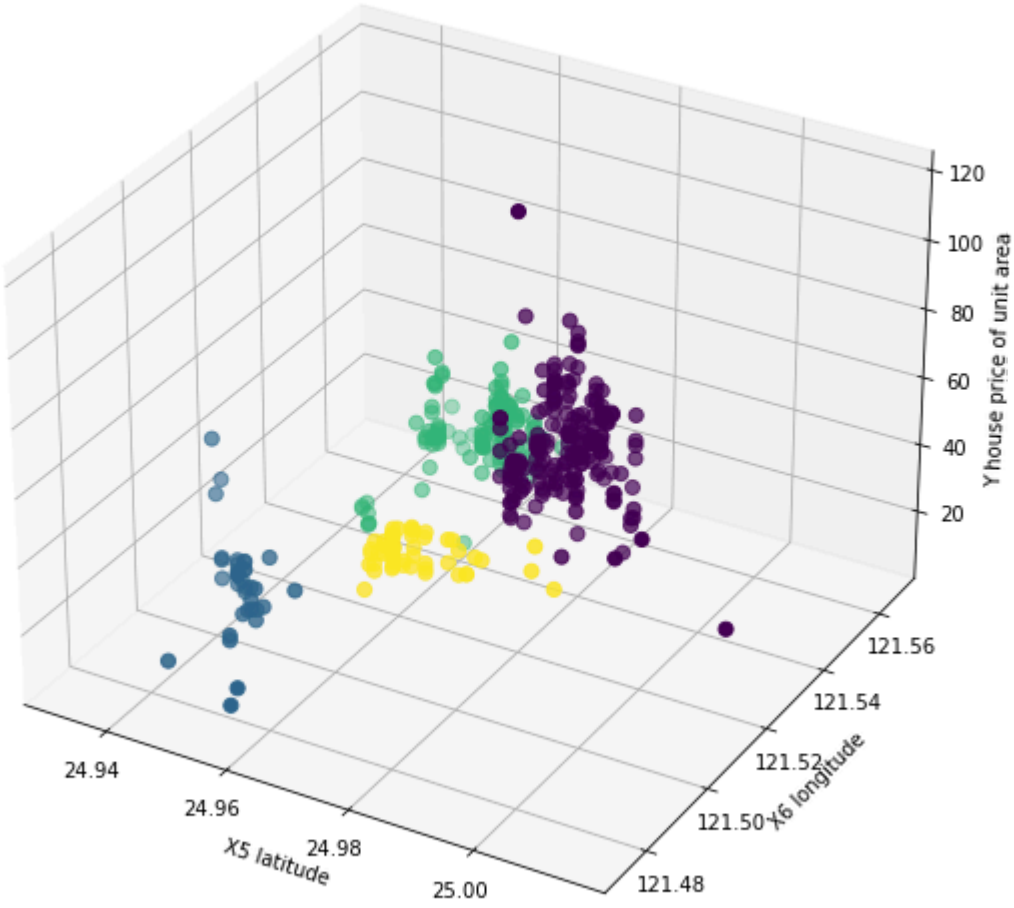
In [ ]:
```python
from sklearn.cluster import KMeans


def SquaredErrorKmeans(data, k):
    kmeans = KMeans(n_clusters=k, random_state=0)
    kmeans.fit(data)
    return kmeans.inertia_
```

```
sqErrors = [SquaredErrorKmeans(data_no_id[['X5 latitude', 'X6 longitude']

plt.plot(range(1, 11), sqErrors, marker='o')
plt.show()
```



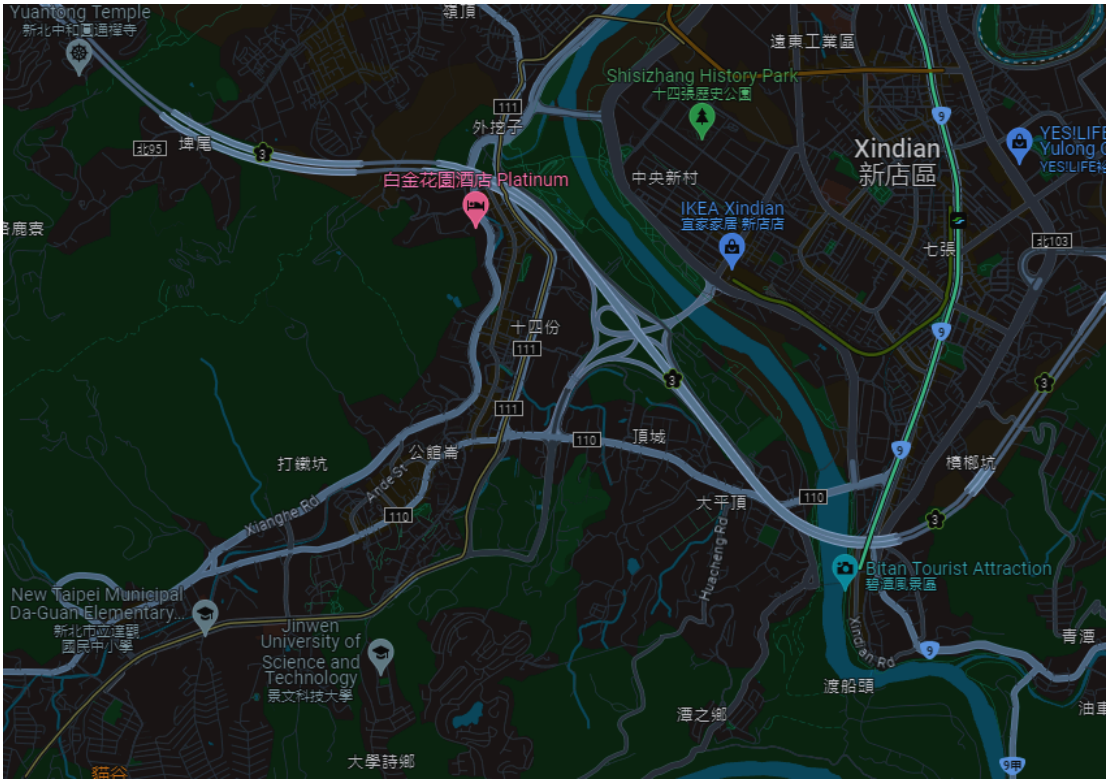
Anhand dieser Kurve können wir ablesen, dass der Punkt, an welchem die Kurve abflacht, bei 4 Clustern liegt. Dementsprechend verwenden wir 4 Cluster für die weitere Analyse. Das Ergebnis der Cluster-Analyse können wir dann in einem Scatter-Plot darstellen. In diesem Plot können wir dann die Farbe der Punkte anhand des Clusters, welchem sie angehören, bestimmen. Damit wir trotzdem den Preis pro Fläche darstellen können, verwenden wir in diesem Fall einen 3D-Scatter-Plot.

In [ ]:
```
from scipy.spatial import ConvexHull

features = data_no_id[["X5 latitude", "X6 longitude"]]
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(features)
labels = kmeans.labels_
data_no_id['cluster'] = labels

# Plotting setup
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('X5 latitude')
ax.set_ylabel('X6 longitude')
ax.set_zlabel('Y house price of unit area')

fig.set_size_inches(15, 10)

ax.set_box_aspect(None, zoom=0.95)
ax.scatter(data_no_id["X5 latitude"].values, data_no_id["X6 longitude"].v


plt.show()
```

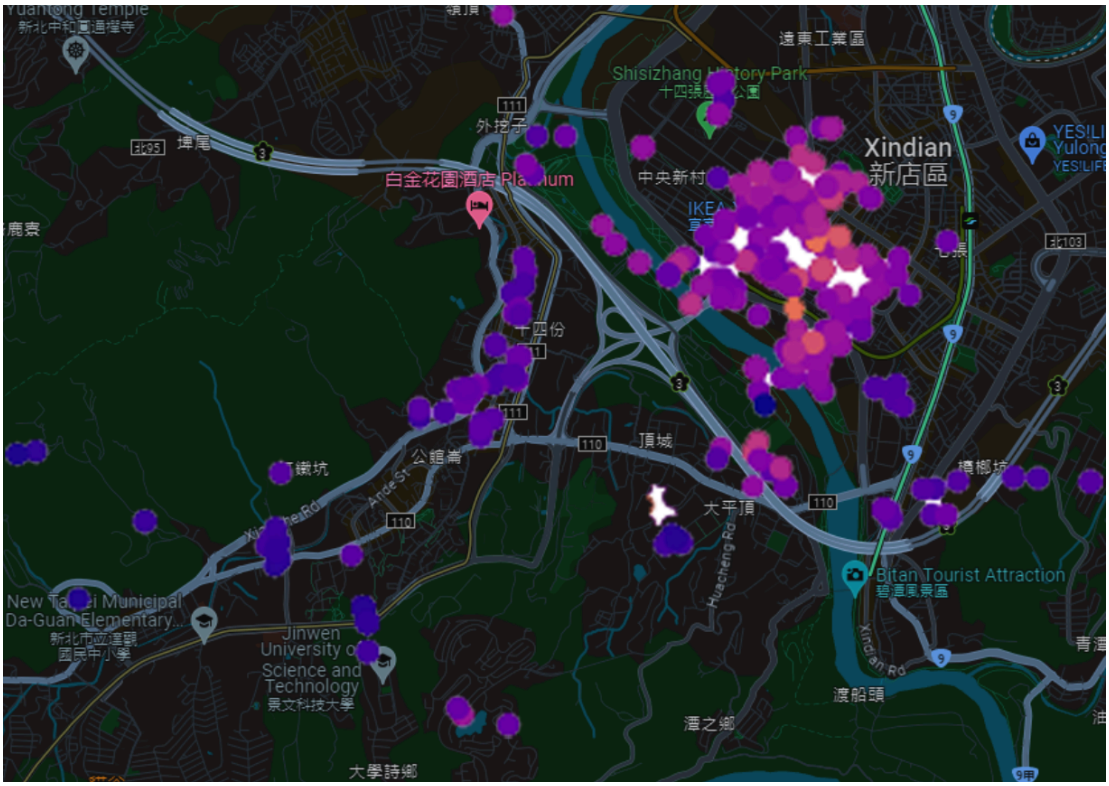


Dieser 3D-Plot zeigt die Aufteilung der Immobilien in 4 Cluster. Es lässt sich beobachten, dass der lila-farbende Cluster die höchsten Preise pro Fläche aufweist. Es lässt sich auch erkennen, dass die Immobilien-Dichte innerhalb des blauen und gelben Clusters geringer ist, als in den anderen Beiden Clustern. Dies könnte darauf schließen lassen, dass wir hier den Vergleich einer eher ländlichen und einer eher städtischen Region haben.

Diese Vermutung lässt sich mithilfe von Google Maps überprüfen.



Dieser Karten-Ausschnitt zeigt den ungefähren Bereich, den der Datensatz abbildet. Wenn wir nun den früheren Plot mit diesem Bild überlagern, ergibt sich folgende Ansicht:

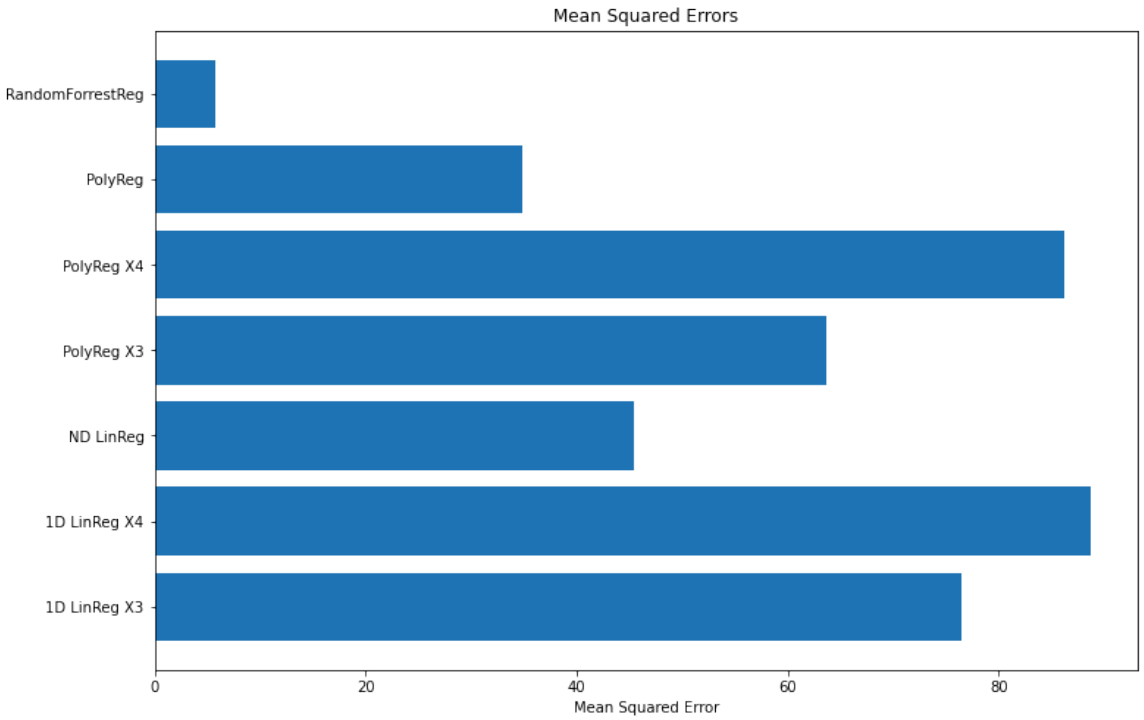


Wir können also feststellen, dass die verschiedenen Cluster tatsächlich unterschiedliche Regionen abbilden.

7 Interpretation der Ergebnisse

Für dieses Projekt haben wir uns eine Vielzahl an verschiedenen Regressionsmethoden zur Analyse des Immobilienbewertungsdatensatzes angesehen. Um diese nun miteinander zu vergleichen, können wir den mittleren quadratischen Fehler (MSE) der verschiedenen Modelle betrachten. Hierzu können wir die zuvor bestimmten Werte in einem Balkendiagramm darstellen.

In [ ]:
```
plt.figure(figsize=(12, 8))

mses = [mse_1dlinregX3, mse_1dlinregX4, mse_ndlinreg, mse_polyregX3, mse_
mse_labels = ["1D LinReg X3", "1D LinReg X4", "ND LinReg", "PolyReg X3",

plt.barh(mse_labels, mses)
plt.title("Mean Squared Errors")
plt.xlabel("Mean Squared Error")
plt.show()
```



Dieser Plot zeigt uns die verschiedenen MSE-Werte für die verwendeten Regressions-Variante. Wir können diesem Plot entnehmen, dass die `Random-Forrest Regression` mit einem signifikanten Abstand die genauste Regressions-Methode war.

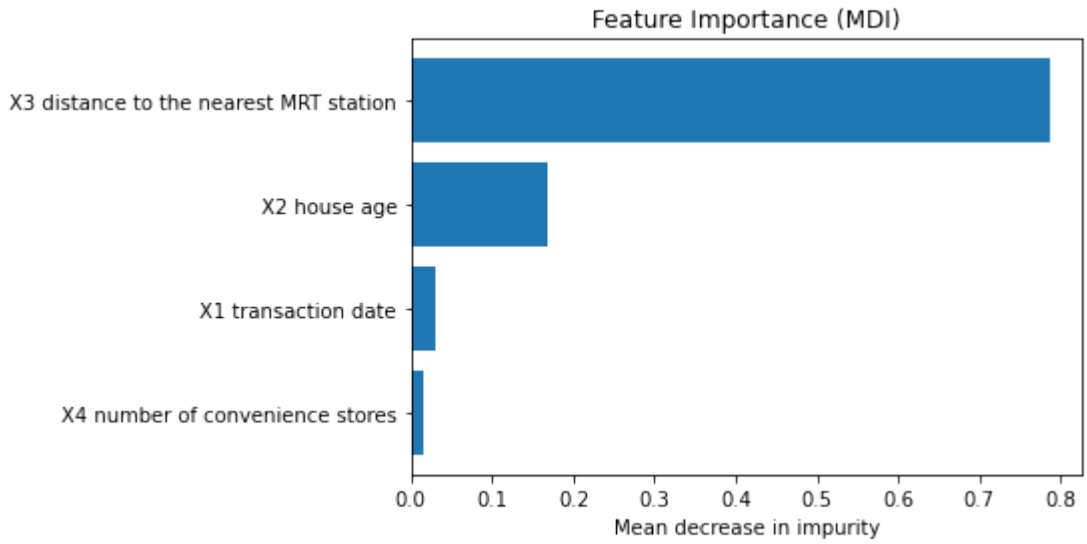
Es lässt sich auch ablesen, dass die Polynomial-Regression, immer einen geringeren MSE-Wert als die vergleichbare lineare Regression aufweist. Dies lässt darauf hindeuten, dass die Beziehung zwischen den Variablen und dem Zielwert nicht linear ist. Sowohl für die polynomiale Regression als auch für die lineare Regression waren die beiden durchgeführten eindimensionale Regressionen ungenauer als die mehrdimensionale Regression.

Um nun auch zu betrachten, welche der verwendeten Variablen den größten Einfluss auf den Zielwert haben, können wir die Feature-Importance der Random-Forrest Regression betrachten. Feature Importance ist ein Maß dafür, wie nützlich oder wertvoll jede Variable (Feature) im Kontext eines vorhergesagten Modells ist. In

maschinellen Lernmodellen, insbesondere in Entscheidungsbaum-basierten Modellen wie Random Forests und Gradient Boosting Machines, wird die Wichtigkeit eines Merkmals oft anhand des Einflusses des Merkmals auf die Modellleistung bewertet. Dies kann beispielsweise durch die Verbesserung der Genauigkeit der Vorhersagen oder durch die Reduzierung der Unsicherheit (Varianz) der Vorhersagen quantifiziert werden. Feature Importance hilft dabei, die Black-Box-Natur komplexer Modelle zu verringern, indem es Einblicke gibt, welche Merkmale die Vorhersagen am stärksten beeinflussen.

Die Berechnung der Feature Importance kann auf verschiedene Weise erfolgen, aber eine gängige Methode bei baum-basierten Modellen ist die Betrachtung der Tiefe, in der ein Feature verwendet wird und wie viel es zur Homogenität der Knoten (beispielsweise durch Gini-Importance oder Mean Decrease in Impurity) beiträgt. Ein Feature, das häufig nahe der Wurzel des Baumes verwendet wird und das zu einer signifikanten Reduzierung der Unreinheit führt, wird als wichtiger angesehen.

In [ ]:
```
feature_importance = reg.feature_importances_
feature_names = reg_vars.columns

# sort features according to importance
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0])

# plot feature importances
plt.barh(pos, feature_importance[sorted_idx], align="center")

plt.yticks(pos, np.array(feature_names)[sorted_idx])
plt.title("Feature Importance (MDI)")
plt.xlabel("Mean decrease in impurity");
```



Das dargestellte Balkendiagramm zeigt die Feature Importance, die auf Mean Decrease in Impurity (MDI) basiert. Der Plot sagt folgendes aus:

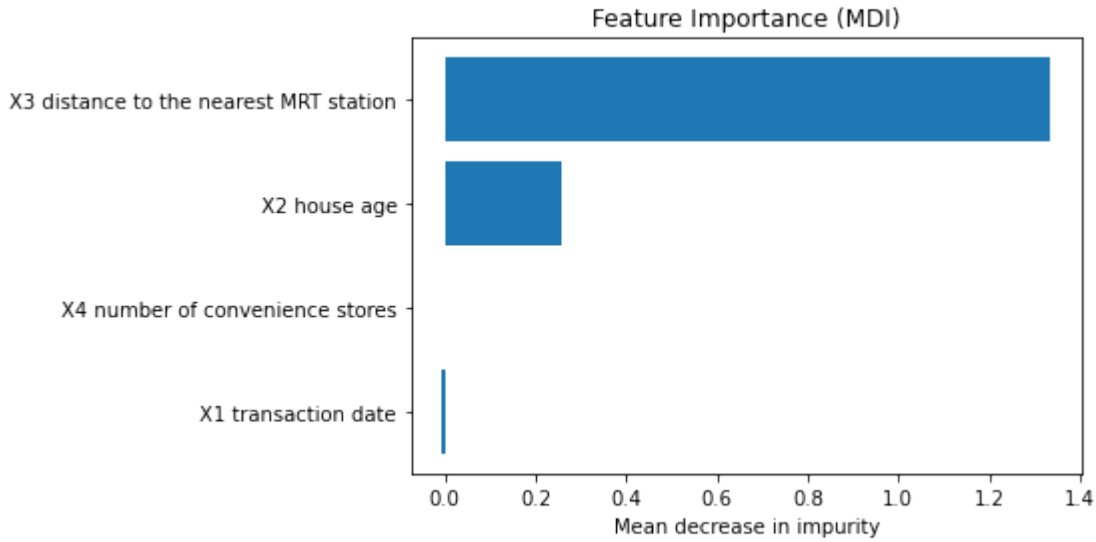
- **X3 distance to the nearest MRT station**: Dieses Merkmal weist die höchste Wichtigkeit auf und ist demnach der stärkste Prädiktor für das Zielmerkmal im Modell. Es hat den größten Einfluss auf die Reduzierung der Unreinheit in den Entscheidungsbäumen.

- **X2 house age**: Dieses Merkmal hat eine mittlere Wichtigkeit. Es trägt weniger als X3 zur Vorhersage bei, ist aber immer noch signifikant wichtiger als X1 und X4.
- **X1 transaction date**: Dieses Merkmal hat eine geringere Wichtigkeit als X3 und X2, was bedeutet, dass es einen kleineren Beitrag zur Verbesserung der Modellvorhersagen leistet.
- **X4 number of convenience stores**: Dieses Merkmal hat die geringste Wichtigkeit von den dargestellten. Es hat den kleinsten Beitrag zur Reduzierung der Unreinheit der Knoten im Entscheidungsbaummodell.

Die Längen der Balken repräsentieren die relative Bedeutung jedes Merkmals. Die Wichtigkeit wird in der Regel so normiert, dass die Summe aller Wichtigkeiten 1 oder 100% ergibt. In diesem Fall deutet die Bedeutung der Merkmale darauf hin, dass die Distanz zur nächsten MRT-Station wahrscheinlich der dominierende Faktor bei der Vorhersage des Zielmerkmals ist, was im Kontext von Immobilienbewertungen darauf hindeuten könnte, dass die Lage in Bezug auf die öffentlichen Verkehrsmittel ein kritischer Faktor für den Wert einer Immobilie ist.

Ein weiterer Aspekt, den wir betrachten können, ist die Permutation Feature Importance. Diese ist eine modell-agnostische Methode, die nach der Modellbildung angewendet wird, um die Wichtigkeit eines Features zu bewerten. Dabei wird die Wichtigkeit eines Merkmals durch Beobachtung der Veränderungen in der Modellleistung gemessen, nachdem die Werte dieses Merkmals im Testdatensatz zufällig vertauscht (permutiert) wurden. Wenn das Vertauschen der Werte eines Merkmals zu einer signifikanten Verschlechterung der Modellleistung führt, gilt dieses Merkmal als wichtig für das Modell. Ein Vorteil dieser Methode ist, dass sie mit jedem Modelltyp verwendet werden kann und nicht von der Modellstruktur abhängt.

In [ ]:
```
result = permutation_importance(
    reg, test_vars, y_test, n_repeats=10, random_state=42, n_jobs=2
)

tree_importances = pd.Series(result.importances_mean, index=feature_names
# sort features according to importance
sorted_idx = np.argsort(tree_importances)
pos = np.arange(sorted_idx.shape[0])

# plot feature importances
plt.barh(pos, tree_importances[sorted_idx], align="center")

plt.yticks(pos, np.array(feature_names)[sorted_idx])
plt.title("Feature Importance (MDI)")
plt.xlabel("Mean decrease in impurity")
plt.show()
```



Das Diagramm zeigt die Permutation Feature Importance für einen Random Forest Regressor, angewendet auf denselben Immobilienbewertungsdatensatz. Die Feature Importance basiert hier auf der Änderung der Modellgenauigkeit, die eintritt, wenn die Werte des jeweiligen Features im Testdatensatz zufällig permutiert werden.

Die Balken repräsentieren, wie stark die Vorhersageleistung des Modells beeinträchtigt wird, wenn das jeweilige Merkmal gestört wird. Ein längerer Balken bedeutet, dass das Weglassen oder Stören des Features zu einer größeren Verschlechterung der Modellgenauigkeit führt, was auf eine höhere Wichtigkeit dieses Features hinweist.

Aus dem Diagramm können wir ablesen:

- **X3 distance to the nearest MRT station**: Dieses Merkmal hat den größten Balken und damit die höchste Permutation Feature Importance. Das bedeutet, dass die Vorhersage des Hauspreises stark von der Entfernung zur nächsten MRT-Station abhängt. Eine Permutation (zufällige Umordnung) der Werte dieses Merkmals führt zu einer signifikanten Verschlechterung der Vorhersagegenauigkeit, was darauf hindeutet, dass es für das Modell entscheidend ist.

- **X2 house age**: Das Alter des Hauses hat eine erkennbare, aber wesentlich geringere Wichtigkeit verglichen mit der Entfernung zur MRT-Station. Es hat einen merklichen Einfluss auf das Modell, aber nicht so stark wie X3.

- **X4 number of convenience stores**: Dieses Merkmal scheint keine Rolle zu spielen. Die Vorhersagegenauigkeit wird durch Permutation dieses Merkmals nicht beeinflusst.

- **X1 transaction date**: Das Transaktionsdatum hat kaum einen Balken, was darauf hinweist, dass dieses Merkmal eine geringe Wichtigkeit hat. Eine Permutation dieses Merkmals hat fast keinen Einfluss auf die Vorhersagegenauigkeit des Modells, was darauf schließen lässt, dass das Datum der Transaktion wenig bis keinen prädiktiven Wert für die Vorhersage des Hauspreises in diesem Modell hat.

Zusammenfassend zeigt das Diagramm, dass in diesem spezifischen Random Forest Modell die Entfernung zur nächsten MRT-Station das mit Abstand wichtigste Merkmal ist und die Anzahl der nahegelegenen Supermärkte nahezu irrelevant für die Vorhersage des Immobilienpreises ist.

Durch diese weitere Betrachtungen der Ergebnisse der Regression konnten wir feststellen, dass die Variablen `X2` und `X3` die größten Einflüsse auf den Zielwert des Datensatzes haben. Des Weiteren konnten wir feststellen, dass die Beziehung zwischen den Variablen und dem Zielwert nicht linear ist. Dies konnten wir anhand der geringeren MSE-Werte der polynomiale Regression feststellen.

Die Cluster-Analyse hat uns zudem gezeigt, dass es Regionen gibt, in denen die Immobilienpreise höher sind als in anderen.

Insgesamt konnten wir verschiedene Methodiken zur Datenanalyse verwenden, um eine akkurate Vorhersage des Immobilienpreises zu treffen.

8 Aufgabenverteilung

Wir haben versucht die Aufgaben möglichst gleichmäßig zu verteilen. Im Folgenden ist in einer Tabelle aufgelistet, wer von uns welche Aufgaben übernommen hat.

Aufgabe	Till	Dennis
Beschreibung und Sichtung der Daten	✅	✅
Eindimensionale Lineare Regression		✅
Mehrdimensionale Lineare Regression	✅	✅
Random Forrest Regression	✅	
Regionale Clusteranalyse	✅	
Interpretation der Ergebnisse	✅	✅

9. Literaturverzeichnis

[1] Yeh,I-Cheng. (2018). Real Estate Valuation. UCI Machine Learning Repository. https://doi.org/10.24432/C5J30W.

[2] Python Paket-Manager pip - https://pypi.org/project/pip/

[3] Pandas Python Bibliothek - https://pandas.pydata.org/

[4] Numpy Python Bibliothek - https://numpy.org/

[5] Matplotlib Python Bibliothek - https://matplotlib.org/

[6] Metro Taipei - https://english.metro.taipei/

[7] Scikit-learn - https://scikit-learn.org/stable/