Final_Bericht_(1)

January 16, 2023

1 Bericht Immobilienrechner

Angaben:

- Autor: Sean Corrigan, Rami Tarabishi
- Abgabedatum: 16.01.2023
- Challenge: cml1/3Db, Immobilienrechner
- Fachexperte: Fernando Benites
- Schule: FHNW
- Studiengang: BSc Data Science

1.1 1.) Einführung

Im Rahmen der Challenge haben wir einen Datensatz welcher von Fernando gestellt wurde untersucht. Unser Ziel war es, verschiedene Modelle zur Vorhersage des Immobilienpreises und zur Klassifizierung des Immobilientyps zu entwickeln. Zunächst haben wir eine explorative Datenanalyse durchgeführt, um die Daten zu verbessern und genau zu untersuchen. Anschließend haben wir ein einfaches lineares Regressionsmodell entwickelt, um den Immobilienpreis vorherzusagen und es so gut wie möglich optimiert. Im Laufe des Semesters haben wir die folgenden Modelle trainiert, optimiert und miteinander verglichen:

- ElasticNet
- KNN-Regressor
- DecisionTree-Regressor
- RandomForest-Regressor
- GraidentBoosting-Regressor

Wir haben auch den folgenden Klassifikator für die Vorhersage des Immobilientyps optimiert:

- RandomForest-Classifier

Am Ende haben wir eine API entwickelt, über die ein Nutzer Angaben zu seiner Immobilie eingeben kann und anschließend entweder den Preis oder den Immobilientyp als Vorhersage erhält.

1.2 2.) Importierte/verwendete Libraries

Wir haben unseren Code in einem Jupyter Notebook geschrieben, um einzelne Codeblöcke ausführen und die Daten immer wieder ansehen zu können. Wir haben die folgenden Bibliotheken verwendet:

- pandas: pandas ist eine Bibliothek zur Datenmanipulation und -analyse, die schnelle und flexible Datenstrukturen wie DataFrame und Series bereitstellt.
- NumPy: NumPy ist eine Bibliothek für die Programmiersprache Python, die Unterstützung für große, mehrdimensionale Arrays und Matrizen sowie eine große Sammlung von hochwertigen Mathematikfunktionen zur Verarbeitung dieser Arrays bereitstellt.
- Matplotlib: Matplotlib ist eine Plotting-Bibliothek für die Programmiersprache Python, die eine objektorientierte API zur Einbindung von Plots in Anwendungen mittels allgemeiner GUI-Werkzeugkits wie Tkinter, wxPython, Qt oder GTK bereitstellt.
- scikit-learn (sklearn): scikit-learn ist eine Bibliothek für maschinelles Lernen für die Programmiersprache Python, die effiziente Implementierungen einer Vielzahl von maschinellen Lernverfahren und Hilfsfunktionen zur Datenanalyse und -vorverarbeitung bereitstellt.
- statsmodels: statsmodels ist eine Bibliothek für statistische Modellierung und Hypothesentests in Python, die verschiedene statistische Verfahren und Modelle bereitstellt.
- pandas_profiling: pandas_profiling ist eine Bibliothek, die es ermöglicht, automatisch eine umfassende Profilanalyse von Datenframe zu erstellen, mit statistischen Informationen, Histogrammen, Korrelationsmatrizen und Warnungen vor potentiellen Problemen.
- SciPy: SciPy ist eine Bibliothek, die eine Vielzahl von hochwertigen und effizienten mathematischen Algorithmen und Funktionen für Anwendungen wie Optimierung, Integrale, Statistik und lineare Algebra bereitstellt.
- Seabron: Seaborn ist eine erweiterte Bibliothek für Datenvisualisierung, die sich besonders gut für die Erkundung und Darstellung von statistischen Modellen und Beziehungen in großen Datensätzen eignet.

Wir haben innerhalb der Challenge keine eigenen Modelle entwickelt, sondern die in sklearn bereitgestellten modelle, trainiert, optimiert und miteinander verglichen.

1.3 3.) Grundlagen / erhaltene Daten

Wir haben insgesamt zwei Datensätze erhalten:

- immo_data_202208_v2.csv: Trainings- und Validierungsdaten, mit denen wir unsere Modelle trainieren und validieren konnten.
- test_data-Kaggle-v0.11.csv: Daten für den Kaggle-Wettbewerb, in denen die Spalte “Preis” fehlte und von unseren Modellen vorhergesagt werden musste.

Die Daten wurden im August von unserem Fachexperten gesammelt (Quelle: Immoscount & Homegate). Sie wurden auch mit zusätzlichen geo-politischen Daten (z.B: Lärmbelastung, etc.) aus einem anderen Projekt angereichert, wobei die Zuweisung durch die kürzeste Distanz durch Latitude und Longitude erfolgte. Die Latitude und Longitude der gesammelten Daten wurden für “Standort” durch Openstreetmap bestimmt.

Alle Teams in den Challenges hatten die gleichen Grundlagen, konnten die Daten jedoch durch EDA und Datawrangling weiter verbessern. Es war auch möglich, über die Teams hinweg zusammenzuarbeiten, jedoch nur bei algorithmischen Fragen und Verständnisaspekten. Es war nicht erlaubt, zusammenhängende Code- oder Textblöcke von anderen Gruppen oder vom Internet zu kopieren.

Hier ein Ausschnitt aus den erhaltenen Trainings/Validations Datenset:

Und hier ein Ausschnitt aus den erhaltenen Trainings/Validations Datenset:

1.4 4.) Explorative Datenanalyse (EDA) / Datawrangling / Teilaufgabe 1

Nachdem wir die Daten in unser Notebook geladen haben, haben wir eine explorative Datenanalyse durchgeführt. Aufgrund dieser Analyse haben wir die Daten verbessert, um sie aussagekräftiger zu machen. Wir haben die Daten aufbereitet, um einen Pandas Profiling Report erstellen zu können, um einen guten Überblick über unsere Daten zu erhalten. Wir haben darauf geachtet, dass wir unsere Datawrangling-Operationen als Sklearn-Transformerklassen geschrieben haben. So konnten wir später eine Preprocessing-Pipeline erstellen, mit der wir unsere Trainings-/Validierungsdaten und unsere Testdaten für den Kaggle-Wettbewerb ohne großen Aufwand transformieren konnten.

Weiterhin haben wir uns auf die Behandlung von fehlenden Werten und die Umwandlung von kategorialen Variablen konzentriert (OneHotEncoding), um unsere Daten für das maschinelle Lernen vorzubereiten. Wir haben auch die Verteilung der Zielvariable untersucht und gegebenenfalls angepasst, um eine bessere Vorhersagegenauigkeit zu erreichen. Insgesamt war die EDA ein wichtiger Schritt, um unsere Daten aufzubereiten und für das maschinelle Lernen vorzubereiten.

1.4.1 4.0.) Spezifizieren von für das Modell wichtigen Features

In dem Dataframe, den wir erhalten haben, waren viele Features vorhanden, von denen viele für unser Modell nicht wichtig waren (zum Beispiel könnten bei einer kategorischen Variable mit sehr vielen unique values beim One-Hot-Encoding zu viele Features entstehen, was das Modell overfitten und das Trainieren des Modells verlangsamen würde). Deshalb haben wir eine Transformer-Klasse geschrieben, in der die für unser Modell wichtigen Features ausgewählt wurden.

Die folgenden Features haben wir ausgewählt:

- Categorical:
  - Type
  - Canton
  - Elevator
  - Garage
  - New
  - Wheelchair
  - Balcony
  - Terrace
  - Fireplace
  - Child
  - Swimming
  - Parking
  - Minergy
- Numerical:
  - Rooms_combined
  - Living_area
  - Plot_area
  - Floor
  - ForestDensityL
  - ForestDensityM
  - ForestDensityS
  - NoisePollutionRailwayL
  - NoisePollutionRailwayM

- NoisePollutionRailwayS
- NoisePollutionRoadL
- NoisePollutionRoadM
- NoisePollutionRoadS
- PopulationDensityL
- PopulationDensityM
- PopulationDensityS
- RiversAndLakesL
- RiversAndLakesM
- RiversAndLakesS
- WorkplaceDensityL
- WorkplaceDensityM
- WorkplaceDensityS
- DistanceToTrainStation
- Gde_area_agriculture_percentage
- Gde_area_forest_percentage
- Gde_area_nonproductive_percentage
- Gde_area_settlement_percentage
- Gde_average_house_hold
- Gde_empty_apartments
- Gde_foreigners_percentage
- Gde_new_homes_per_1000
- Gde_politics_bdp
- Gde_politics_cvp
- Gde_politics_evp
- Gde_politics_fdp
- Gde_politics_glp
- Gde_politics_gps
- Gde_politics_pda
- Gde_politics_rights
- Gde_politics_sp
- Gde_politics_svp
- Gde_pop_per_km2
- Gde_population
- Gde_private_apartments
- Gde_social_help_quota
- Gde_tax
- Gde_workers_sector1
- Gde_workers_sector2
- Gde_workers_sector3
- Gde_workers_total
- Land_area
- Room_height
- Last_refurbishment
- Year_built
- Volume
- Gross_yield
- Number_of_floors

– Number_of_toilets

Diese Funktion haben wir als Sklearn-Transformer-Klasse geschrieben, da wir diese Transformationen auf unseren Trainings-/Validationsdaten sowie auf den Kaggle-Contest-Daten ausführen mussten.

1.4.2 4.1.) Price Column

Als nächstes haben wir uns mit unserer Target Variable, dem Immobilienpreis, auseinandergesetzt. Dazu haben wir die Spalte "Price" mit der Spalte "Price_Cleaned" verglichen. Um diese vergleichen zu können, mussten wir die Spalte "Price" zunächst "cleanen", indem wir alle nicht-numerischen Zeichen entfernt und den Währungsumrechnungskurs von EUR zu CHF angepasst haben (1 EUR = 0.99 CHF). Dabei stellten wir fest, dass diejenigen Zeilen, in denen wir die Währung umgerechnet haben, sich von denen in der Spalte "Price_Cleaned" unterscheiden, da in dieser Spalte ein Umrechnungskurs von 1 verwendet wurde. Aus diesem Grund haben wir unsere selbst gereinigte Spalte "Price" verwendet, da sie die Währungen korrekt umgerechnet enthält. Da diese Transformationen nur auf unseren Trainings- und Validationsdaten ausgeführt werden mussten, haben wir diese nicht als Sklearn Transformer-Klasse geschrieben.

```
C:\Users\rami0\AppData\Local\Temp/ipykernel_17048/2805829031.py:5:
FutureWarning: The default value of regex will change from True to False in a
future version. In addition, single character regular expressions will *not* be
treated as literal strings when regex=True.
  immo.price = immo.price.str.replace(char, "")
```

1.4.3 4.2.) Droppen von diversen Rows

Nachdem wir unsere Target Variable "gecleaned" hatten, haben wir den Dataframe im Jupyter Notebook Variable Viewer manuell durchgesehen, um weitere Auffälligkeiten zu finden, die uns aufgefallen sind. Dabei haben wir folgende Dinge festgestellt:

- Spezifische Immobilientypen: Es gibt diverse Immobilientypen, die nur sehr selten vorkommen und die für unser Modell nicht relevant sind, da wir uns auf Wohnimmobilien beschränken wollen. Um ein Overfitting zu vermeiden, haben wir die Zeilen mit diesen Typen aus dem Dataframe entfernt. Spezifisch betraf es hier die Typen "Hobby room", "Granny Flat" und "Cellar Compartment".
- Überbauungen: Inserate von Überbauungen kamen häufig mehrfach vor und die Daten waren nicht gut strukturiert. Zum Beispiel waren mehrere Wohnungen in einem Inserat aufgeführt, wie z.B. "3.5/4.5/5.5-Zimmer-Wohnungen", wobei oft der Wert für Living_Area und Rooms nicht mit dem Preis übereingestimmt hat. Deshalb haben wir diese Inserate entfernt.
- Zu niedriger Preis: Einige Inserate hatten einen zu niedrigen Preis und machten aufgrund der Angaben keinen Sinn. Oft handelte es sich hierbei um Monatsmieten statt Kaufpreise. Um unser Modell nicht zu verwirren, haben wir alle Inserate mit einem Preis unter 10'000 CHF entfernt.
- Duplikate: Inserate waren häufig doppelt vorhanden. Wir haben alle Duplikate entfernt, indem wir die Spalte Description verwendet haben.
- Price NAN: Alle Inserate ohne angegebenen Preis haben wir entfernt.

Diese Transformationen haben wir nicht als Sklearn Transformer Klasse geschrieben, da wir sie nur auf unseren Trainings-/Validierungsdaten ausführen mussten.

1.4.4 4.3.) Informationen von der Description Column

Wir haben uns anschließend weiter damit beschäftigt, unsere Daten zu verbessern. Dabei haben wir festgestellt, dass in der Spalte "Description" viele Informationen enthalten sind, die bereits in separate Spalten extrahiert wurden. Um diese Informationen besser zu nutzen, haben wir mehrere Transformer-Klassen erstellt. Diese Klassen ermöglichen es uns, unsere Trainings- und Validierungsdaten sowie unsere Testdaten entsprechend anzupassen. Wir haben die folgenden Informationen extrahiert:

- Living Area: Wir konnten die Wohnfläche aus der Beschreibung extrahieren. Nach einem Vergleich (Vergleich von Auge manuell, aufgrund der Zeilen, in welchen die beiden Values nicht übereingestimmt haben) mit der Spalte "Living_area_unified" haben wir festgestellt, dass die "Living_area_unified" plausibler war. Allerdings gab es dort diverse NAN-Werte, die wir mit den Werten aus der Beschreibung imputieren konnten.
- Räume: Auch die Anzahl der Räume konnten wir aus der Beschreibung extrahieren. Nach einem Vergleich (Vergleich von Auge manuell, aufgrund der Zeilen, in welchen die beiden Values nicht übereingestimmt haben) mit der Spalte "Rooms" haben wir festgestellt, dass unsere extrahierte Variable besser passte. Wir haben jedoch auch festgestellt, dass in der Spalte "Rooms" Werte vorhanden waren, wo wir NANs hatten. Deshalb haben wir dort, wo wir NANs in unserer extrahierten Variable hatten, diese mit den Werten aus der Spalte "Rooms" imputiert.

Wir haben keine weiteren Informationen aus der Beschreibung extrahiert oder weitere Analysen der Beschreibung durchgeführt.

1.4.5 4.4.) Cleanen des Type Features

In einem weiteren Schritt haben wir uns mit dem Feature "Type" beschäftigt und es "gecleaned". Dabei haben wir diverse Typen in gemeinsame Gruppen zusammengefasst, um später weniger einzelne Typen zu haben und das Modell dadurch weniger zu overfitten. Beispielsweise haben wir die Typen "Cellar-Compartment", "Single Room" und "Hobbyroom" alle als "Single Room" zusammengefasst.

Diese Transformation haben wir als Sklearn Transformer Klasse geschrieben, damit wir sie sowohl auf unseren Trainings- und Validierungsdaten als auch auf den Kaggle Contest Daten anwenden konnten.

1.4.6 4.5.) Location Feature in separate Features unterteilen

Als nächstes haben wir das Location Feature in einzelne Columns aufgesteilt. Wir haben dabei aber nur die folgenden Features erzeugt:

- canton
- zip
- city

Diese Transformation haben wir als Sklearn Transformer Klasse geschrieben, damit wir sie sowohl auf unseren Trainings- und Validierungsdaten als auch auf den Kaggle Contest Daten anwenden konnten.

1.4.7 4.6.) Extrahieren von weiteren Features

Als nächstes haben wir aus dem "features" Feature weitere Features erzeugt, so dass wir diese später OneHotEncoden können. Die foglenden Fatrues haben wir des dem "features" Feature extrahiert:

- Elevator
- Garage
- New
- Wheelchair Access
- Balcony
- Terrace
- Fireplace
- Child Friendly
- Swimming Pool
- Parking
- Minergy

Diese Transformation haben wir als Sklearn Transformer Klasse geschrieben, damit wir sie sowohl auf unseren Trainings- und Validierungsdaten als auch auf den Kaggle Contest Daten anwenden konnten.

1.4.8 4.7.) Strippen von Strings

In einem nächsten Schritt haben wir eine Transformer Klasse geschrieben in welcher wir die Numerischen Features von Alphabetischen und Charaktern gestrippt haben. Dies war speziell wichitg bei dem Floor Feature, welches häufig z.B: mit 4 Floor oder ähnlichem aufgeführt war.

Diese Transformation haben wir als Sklearn Transformer Klasse geschrieben, damit wir sie sowohl auf unseren Trainings- und Validierungsdaten als auch auf den Kaggle Contest Daten anwenden konnten.

1.4.9 4.8.) Cleanup pipeline

Die Oben beschrieben Transformer Klassen haben wir dann in einer "CleanUp Pipeline" zusammengefasst, so dass wir diese später in die Preprocessing Pipeline integrieren konnten.

1.4.10 4.8.) Profiling Report von den relevanten Feaures

Wir haben einen Pandas Profiling Report auf den für uns relevanten Daten erstellt. Den Raport haben wir auf einem Subset von 1000 Datenpunkten erstellt. Mit dieser Anzahl von Daten benötigt der Raport nicht zu viel zeit bei der Erstellung benötigt.

Die folgenden Erkenntnisse haben wir aus den Reports erhalten:

- Categorical:
  - Terrace und Balcony haben eine Korrelation
  - Die Extracteten Features (Wheelchair, Swimming Pool, etc.) haben relativ viele 0s/NANs
- Numerical:
  - Wir haben generell viele Missing Values
  - Wir haben keine Negative Values was ein gutes Zeichen ist

  - Wir haben viele korrelierende Variablen, vor allem in den Gruppen, z.B: ForestDensityS korreliert mit ForestDensityM & ForestDensityL

Die beiden Reports sind als HTML Files im GitRepo gespeichert.

1.5 5.) Train/Test Split

Nachdem wir eine erste explorativ Datenanalyse durchgeführt und spezielle Transformer-Klassen für späteres Preprocessing erstellt haben, haben wir unsere Daten in Trainings- und Validierungsdaten unterteilt, indem wir einen Train/Test-Split im Verhältnis 20:80 durchgeführt haben.

1.6 6.) Preprocessing

Für das Preprocessing haben wir die Transformer-Klassen, die wir während der explorativen Datenanalyse für das Data Wrangling erstellt haben, in eine Clean-Up-Pipeline integriert. Anschließend haben wir zwei ColumnTransformer erstellt:

- Numerical_transformer, für numerische Merkmale, mit den folgenden Schritten:
  - Imputer: Imputation von fehlenden Werten mit dem jeweiligen Spaltenmedian. Da wir relativ viele Ausreißer im Datensatz hatten, haben wir den Median verwendet, um den Einfluss von Ausreißern zu reduzieren.
  - Scaler: Standard Scaler, um beim Gradientenabstieg schneller zu konvergieren.
- Categorical_transformer, für kategoriale Merkmale, mit den folgenden Schritten:
  - Imputer: Imputation mit "Missing".
  - OneHotEncoding: Standard One-Hot-Encoding für alle kategorialen Variablen.

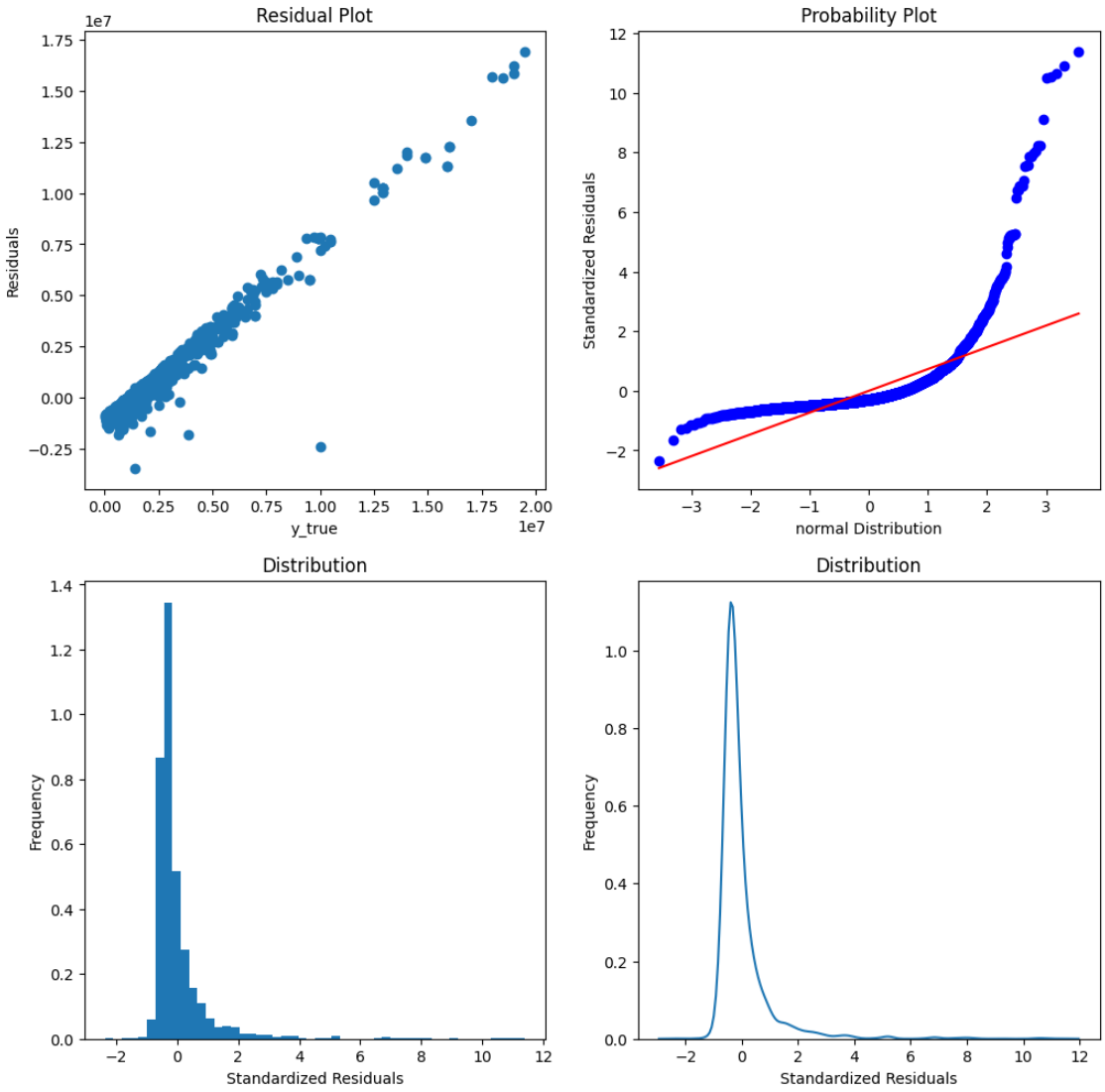
Wir haben das Preprocessing in einer Pipeline zusammengefasst, damit wir die Trainings-, Test- und Kaggle-Wettbewerbsdaten einfach transformieren konnten.

1.7 7.) Einfaches Lineares Regressionsmodell / Teilaufgabe 2.1

Für das einfache Regressionsmodell haben wir den LinearRegressor der sklearn Library verwendet und diesen auf der Variable Square meters trainiert. Wir haben als MAPE einen Wert von 95.27768461453303 erhalten, was relativ hoch ist. Deshalb haben wir die Residuals, sowie die Input und Target Varibale untersucht.

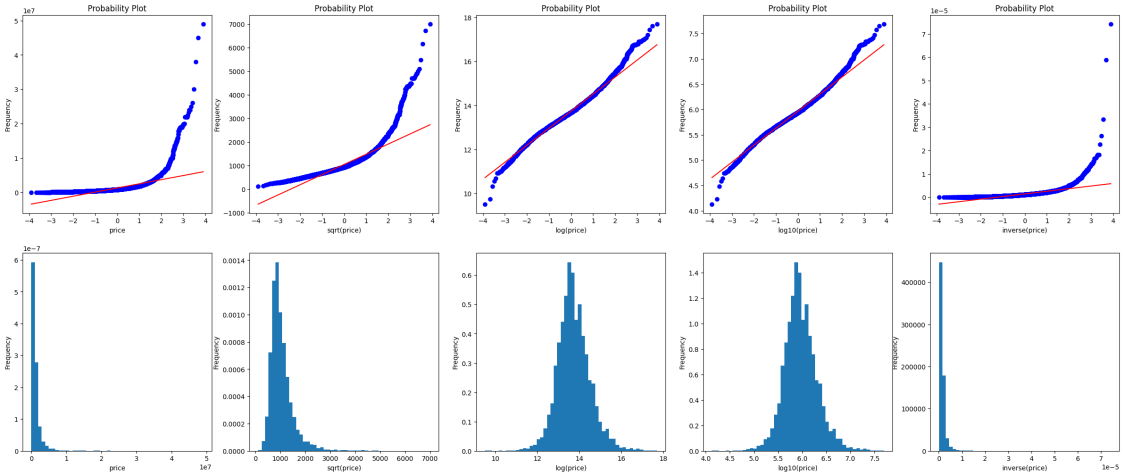
```
MAPE train: 93.835999237628
MAPE test: 95.27768461453053
```

Bei genauerer Betrachtung (in unserem Fall mit diversen Residuen plots) sieht man, dass die Residuen nicht auf eine Normalverteilung passen, und das die Residuen auch nicht um 0 herum schwanken. Somit ist eigentlich klar ersichtlich, dass ein Lineares Regressionsmodel unsere Daten nicht gut beschreibt. Wir können, jedoch unsere Inputvariablen weiter untersuchen, um zu sehen, ob wir durch eine transformation unserer Inputvariablen eine Verbesserung des MAPE und der Verteilung der Residuen erreichen können.



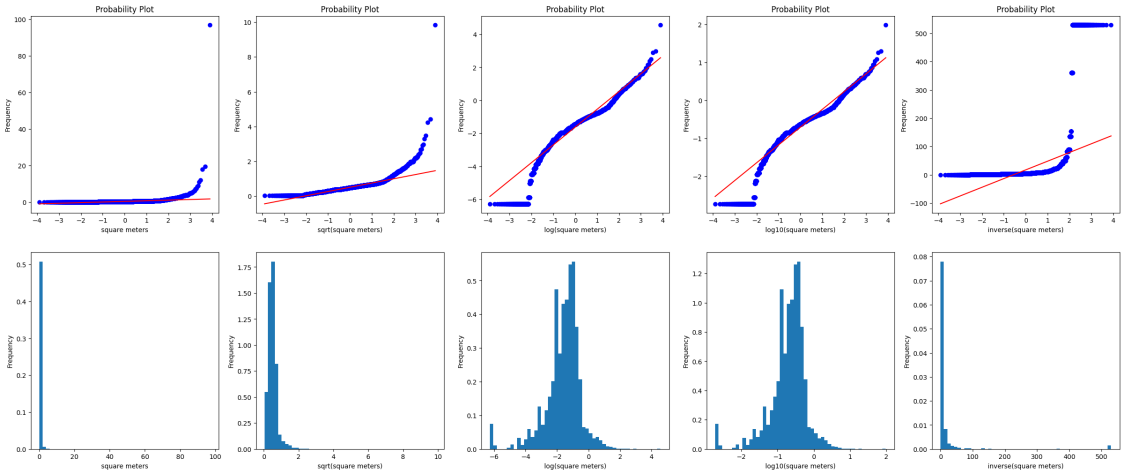
Als erstes haben wir unsere Inputvariable untersucht, ob wir diese durch eine Transformation besser auf deine Normalverteilung bringen können. Anhand der unten gezeigten Plots ist zu erkennen, dass die Verteilung der Values der Target Variable mit einer Log Transformation verbessert werden kann.

```
Transformations for better skewness of feature variable 'price'
-------------
skew price:           8.23741865022083
skew sqrt(price):     2.748049342435257
skew log(price):      0.3750563986012194
skew log10(price):    0.37505639860121454
skew inverse(price):  14.060860033458695
```



Auch bei dem Input Feature square Meters kann eine Log transformation helfen, die Daten besser auf eine Normalverteilung zu bringen.

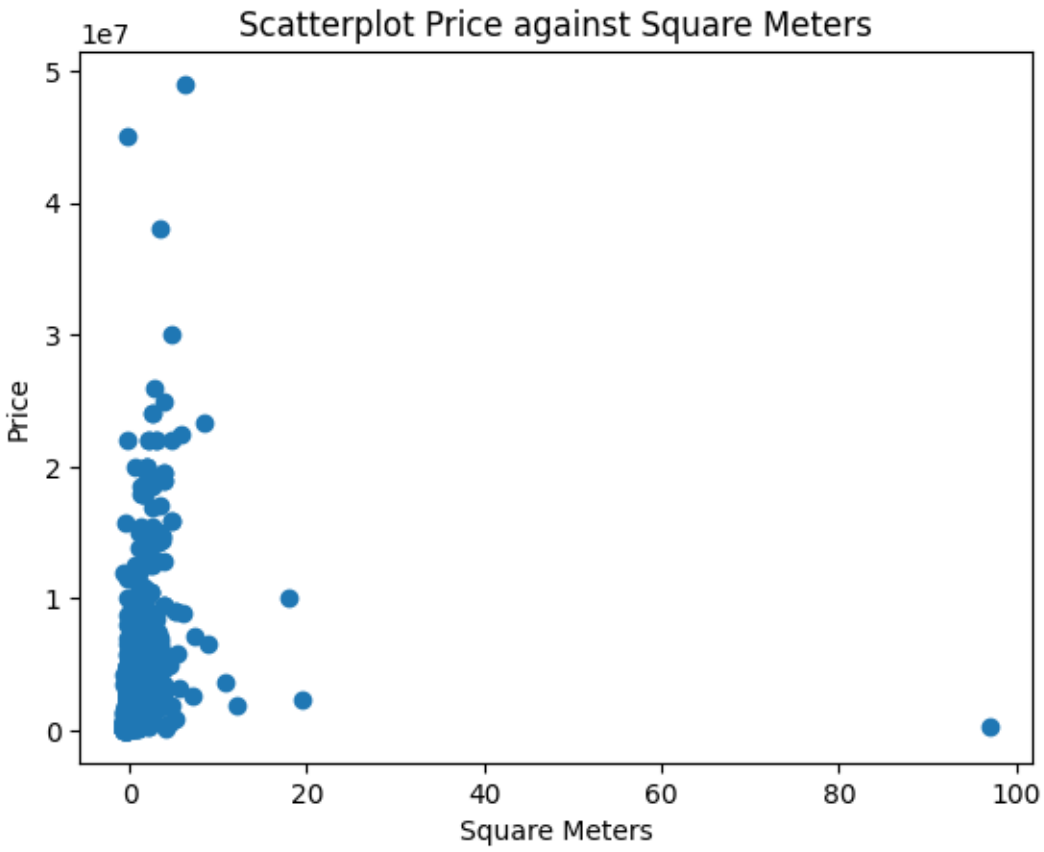
```
Transformations for better skewness of feature variable 'square meters'
-------------
skew square meters:             76.58172682050093
skew sqrt(square meters):       4.888471115751927
skew log(square meters):        -1.3295427591355913
skew log10(square meters):      -1.3295427591355913
skew inverse(square meters):    6.85383197659027
```



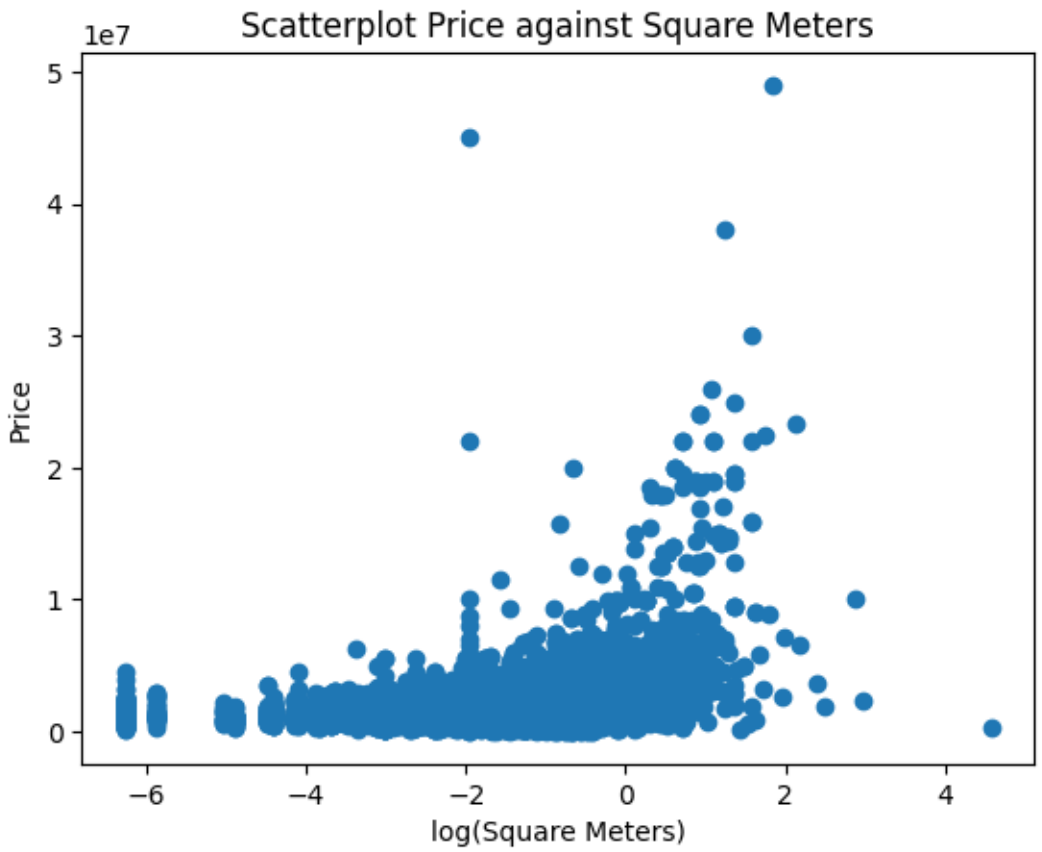
Wenn die Variablen und die Transformierten Variablen gegeneinander geplottet werden, so kann man erkennen, dass keine wirkliche lineare Beziehung zwischen den beiden Variablen besteht. Jedoch kann eine leichte Verbesserung erzielt werden durch die Log Transformation von der Varibale

square meters.

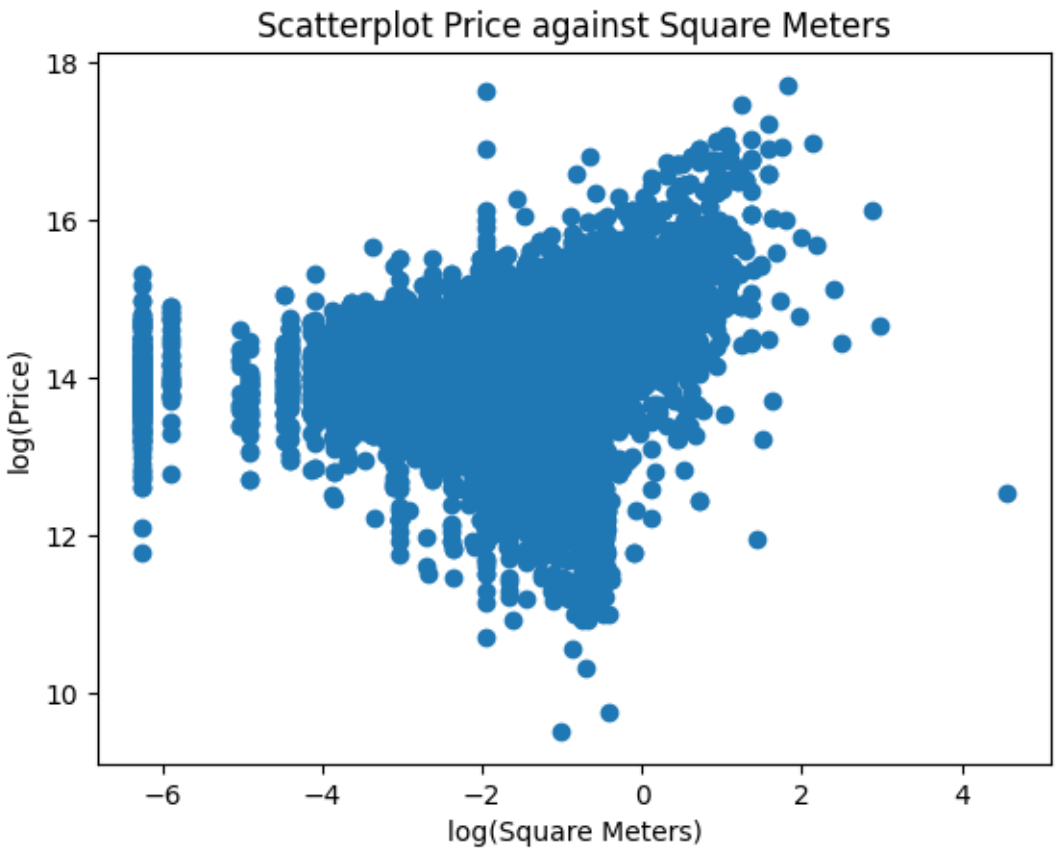
```
Text(0, 0.5, 'Price')
```




```
Text(0, 0.5, 'Price')
```




```
Text(0, 0.5, 'log(Price)')
```

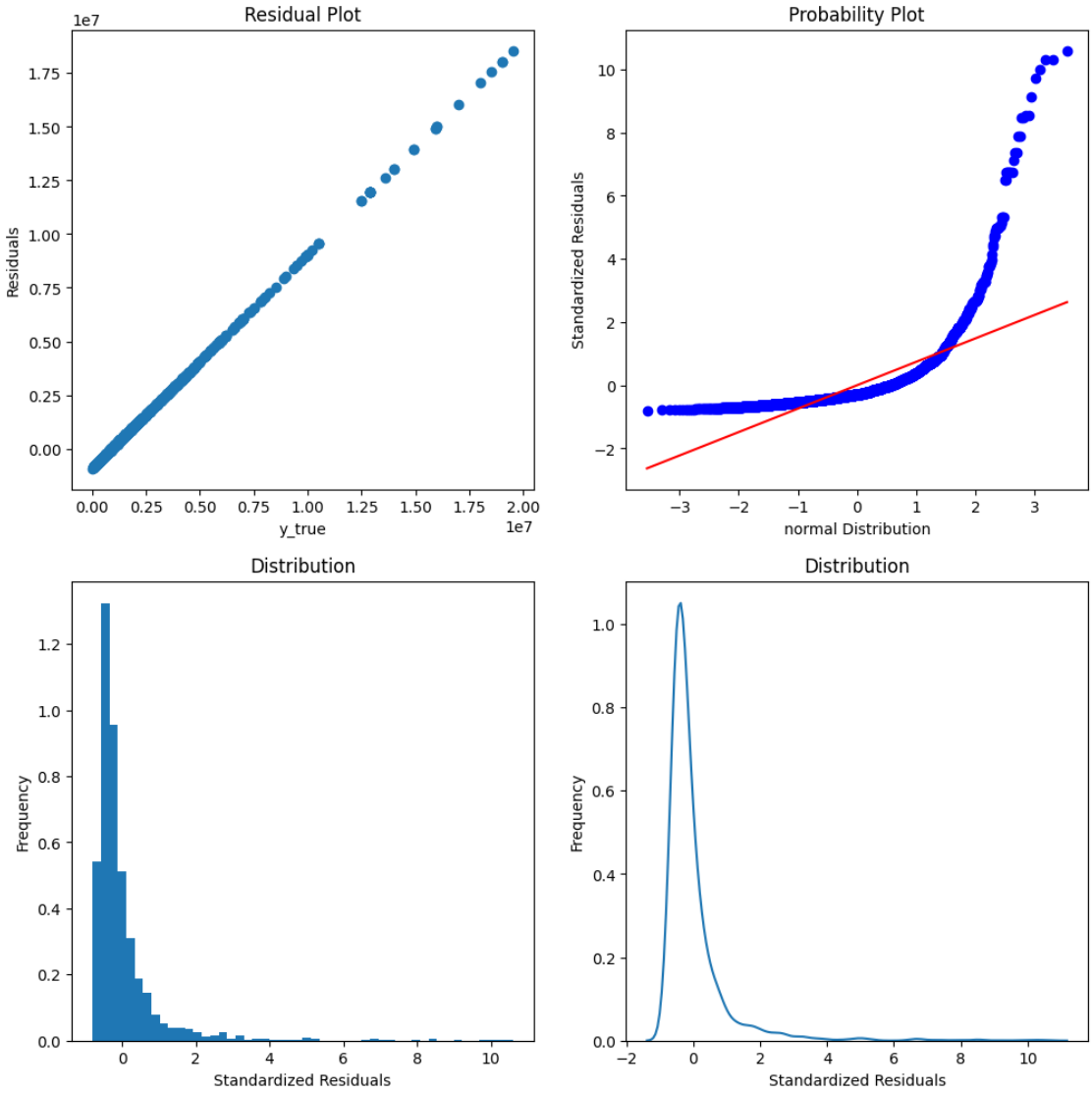



```
MAPE train: 128.1742917745589
MAPE test: 134.72870444105482
```

Der MAPE und die Verteilung der Residuals haben sich etwas verbessert im Vergleich zu dem Model mit den nicht Transformierten Variablen. Jedoch wird aus der Betrachtung der Scatterplots weiter oben klar, dass ein lineares Modell nicht gut für die vorliegenden Daten geeignet ist.

Zur Verbesserung des MAPE muss das Model Komplexer werden (z.B: weitere Features), oder ein anderes Model gewählt werden, welches besser auf die vorliegenden Daten passt.

```
MAPE train: 72.37134728852531
MAPE test: 76.11215145818282
```



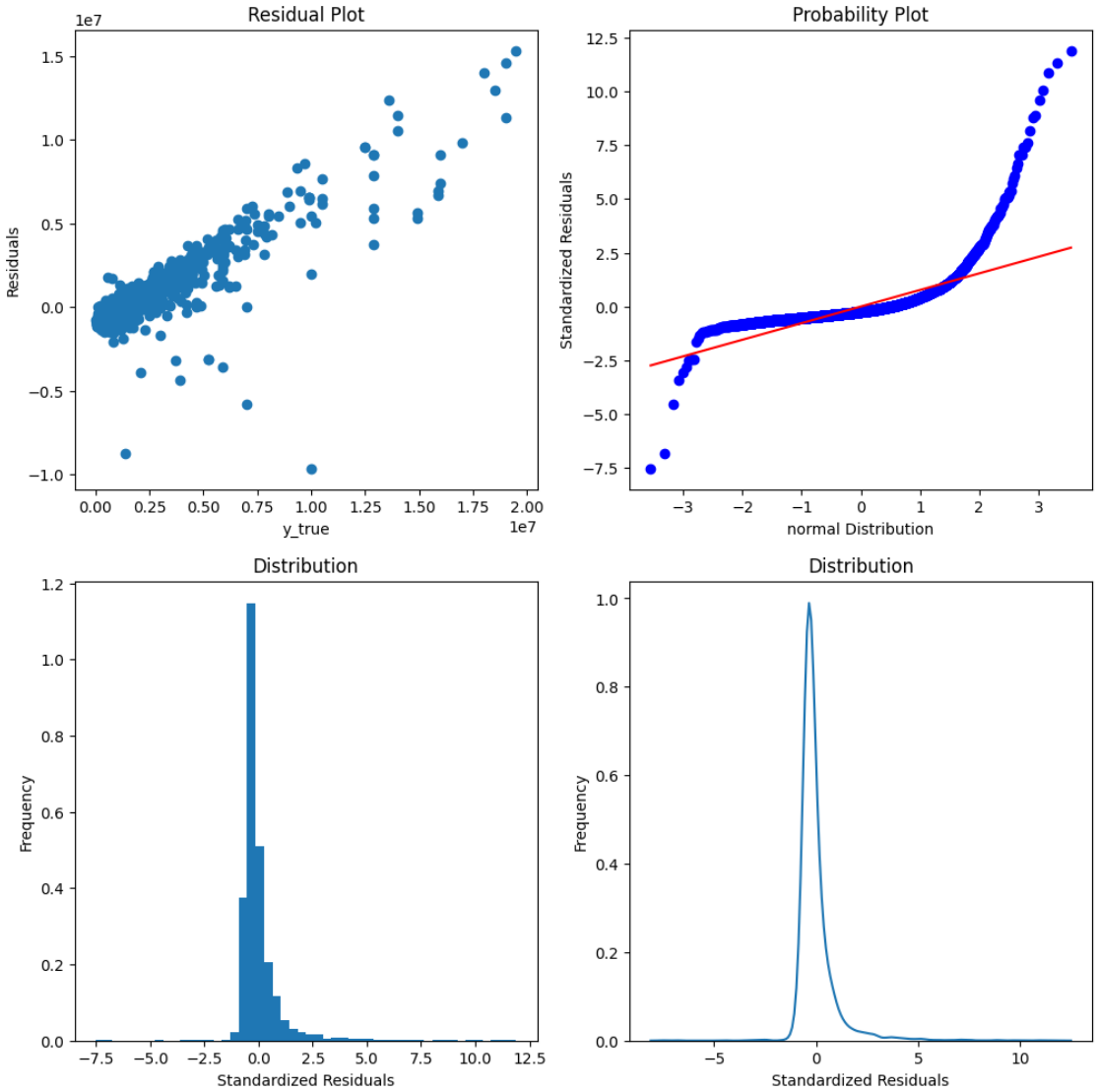
1.8 8.) Weitere Regressionsmodelle / Teilaufgabe 2.2

1.8.1 8.1.) Elastic Net

Beste parameter für das Elastic Net modell:

- k = 30
- alpha = 0.9

```
MAPE train: 81.20%
MAPE test: 84.55%
```



Code zur Optimierung:

```
## Elastic Net pipeline
pipeline_ElasticNet = Pipeline(steps = [
     ('selector',SelectKBest(mutual_info_regression)),
     ('model',ElasticNet())
    ])

## hyperparameter tuning
search = GridSearchCV(
    estimator = pipeline_ElasticNet,
    param_grid = {
        'selector__k':[10, 30, 90, 180],
        'model__alpha':[0.1, 0.3, 0.9],
```

```
        'model__l1_ratio': [0.1, 0.3, 0.5, 0.7, 0.9]
        },
    n_jobs=-1,
    scoring="neg_mean_squared_error",
    cv=5,
    verbose=1
)

## fit best params
search.fit(
    X = X_train_preprocessed[:3000,:],
    y = y_train[:3000]
)

best_params_Elastic = search.best_params_
print(best_params_Elastic)

## set best params
pipeline_ElasticNet.set_params(**best_params_Elastic)

## fit final model
pipeline_ElasticNet.fit(
    X = X_train_preprocessed,
    y = y_train
)

## output scores:
print(f"""
r2 train: {pipeline_ElasticNet.score(X = X_train_preprocessed, y = y_train)}
r2 test: {pipeline_ElasticNet.score(X = X_test_preprocessed, y = y_test)}
""")

## Save the model with numpy
np.save('./data/ElasticNet.npy', pipeline_ElasticNet)
```

1.8.2 8.2.) Logistic Regression

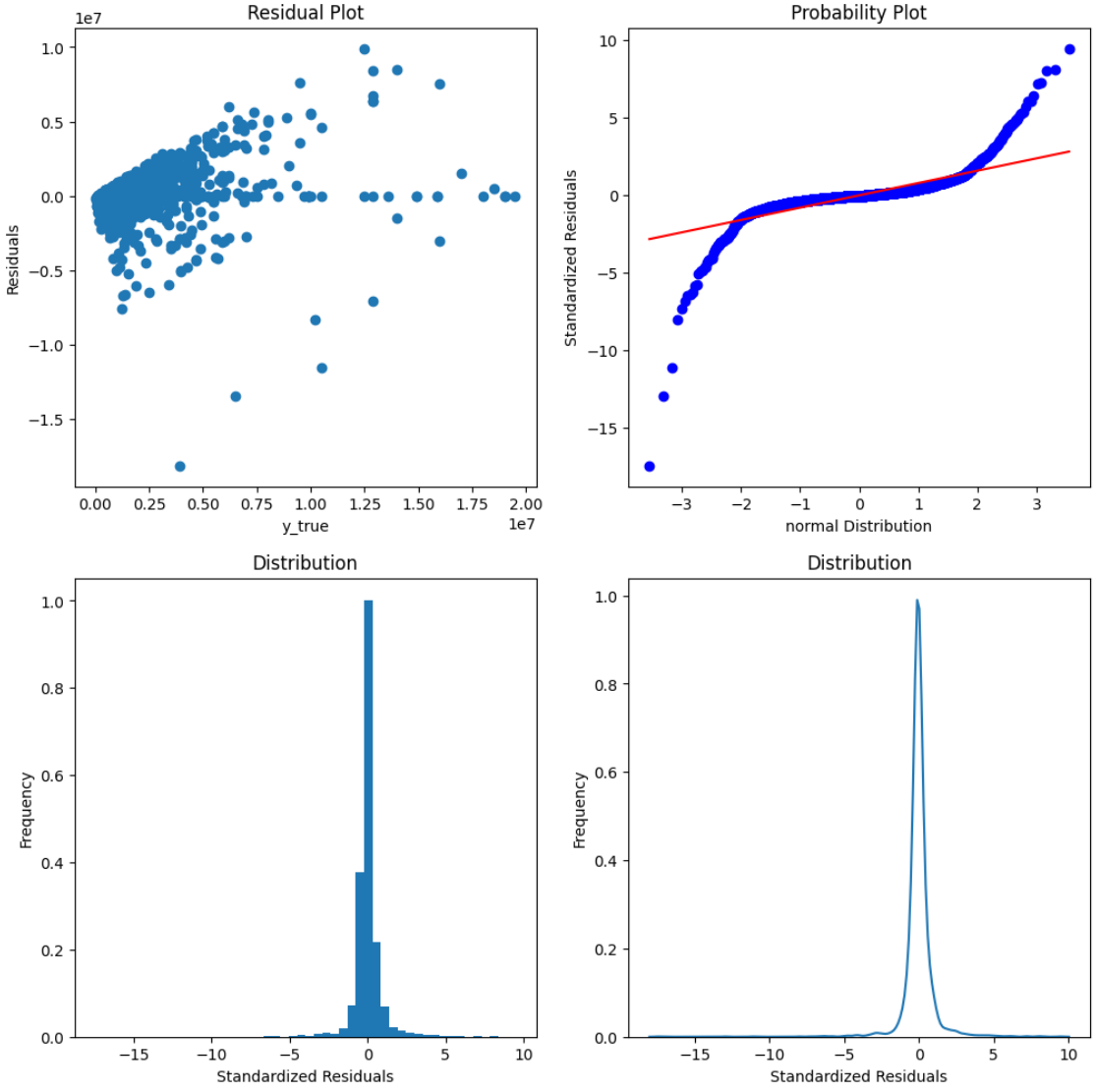
Beste parameter für Logistische Regression:

- k = 90
- C = 1

```
c:\Users\rami0\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_logistic.py:444: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
MAPE train: 38.77%
MAPE test: 42.95%
```



Code zur Optimierung:

```
## Logistic reg pipeline
pipeline_Logistic = Pipeline(steps = [
     ('selector',SelectKBest(mutual_info_regression)),
     ('model',LogisticRegression())
    ])
```

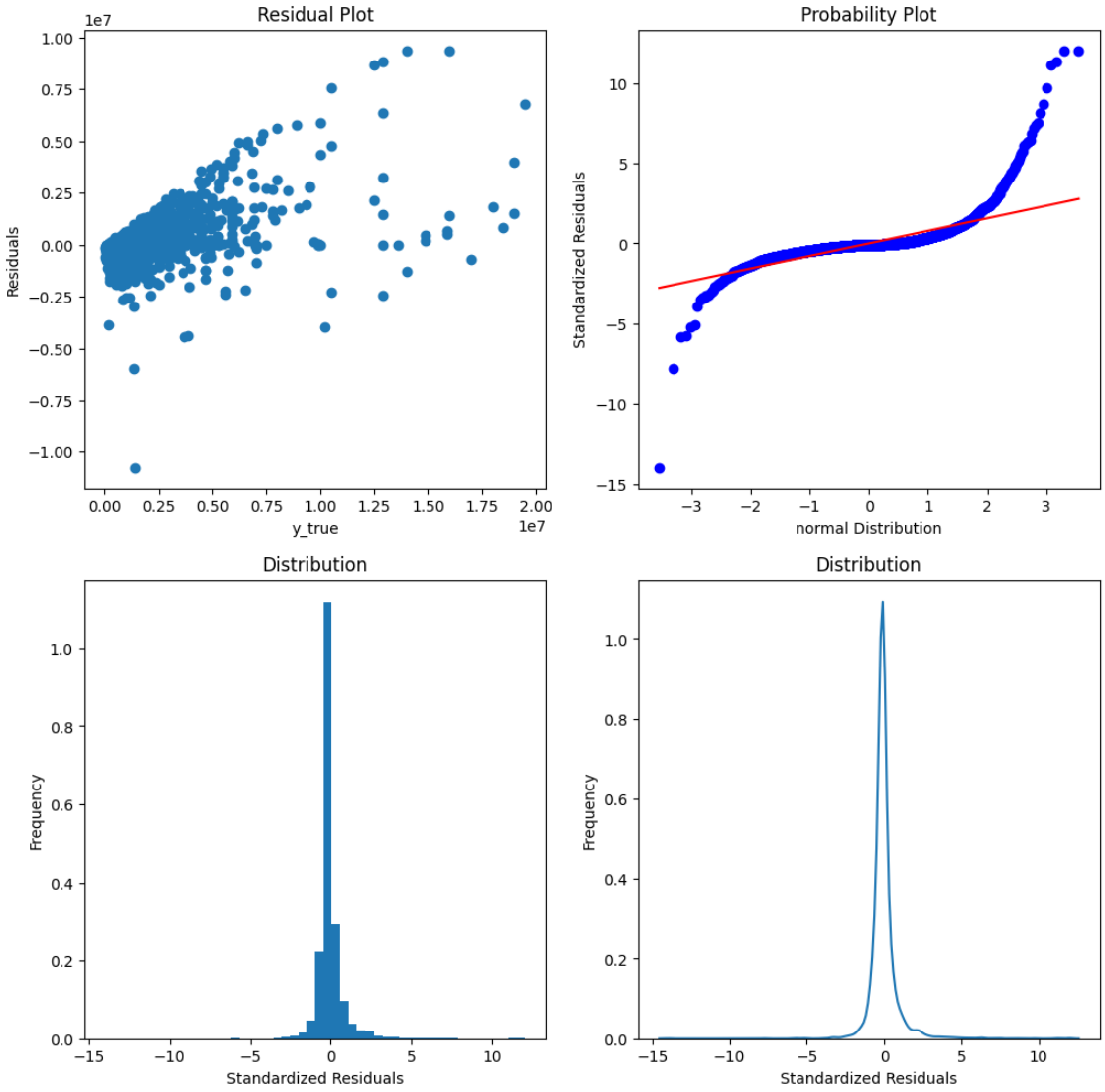
```
## hyperparameter tuning
search = GridSearchCV(
    estimator = pipeline_Logistic,
    param_grid = {
        'selector__k':[10, 30, 90, 180],
        'model__C': [0.1, 1, 10],
        'model__penalty': ['l1', 'l2']
        },
    n_jobs=-1,
    scoring="neg_mean_squared_error",
    cv=5,
    verbose=3
)

## fit best params
search.fit(
    X = X_train_preprocessed[:3000,:],
    y = y_train[:3000]
)

best_params_Logistic = search.best_params_
print(best_params_Logistic)

## set best params
pipeline_Logistic.set_params(**best_params_Logistic)

## fit final model
pipeline_Logistic.fit(
    X = X_train_preprocessed,
    y = y_train
)

## output scores:
print(f"""
r2 train: {pipeline_Logistic.score(X = X_train_preprocessed, y = y_train)}
r2 test: {pipeline_Logistic.score(X = X_test_preprocessed, y = y_test)}
""")

## Save the model with numpy
np.save('./data/Logistic.npy', pipeline_Logistic)
```

1.8.3 8.3.) KNN Regressor

Beste parameter für den KNN Regressor:

- k = 180
- weights = distance

```
MAPE train: 0.71%
MAPE test: 32.74%
```



Code zur Optimierung:

```
## KNN pipeline
pipeline_KNN = Pipeline(steps = [
     ('selector',SelectKBest(mutual_info_regression)),
     ('model',KNeighborsRegressor())
    ])

## hyperparameter tuning
search = GridSearchCV(
    estimator = pipeline_KNN,
```

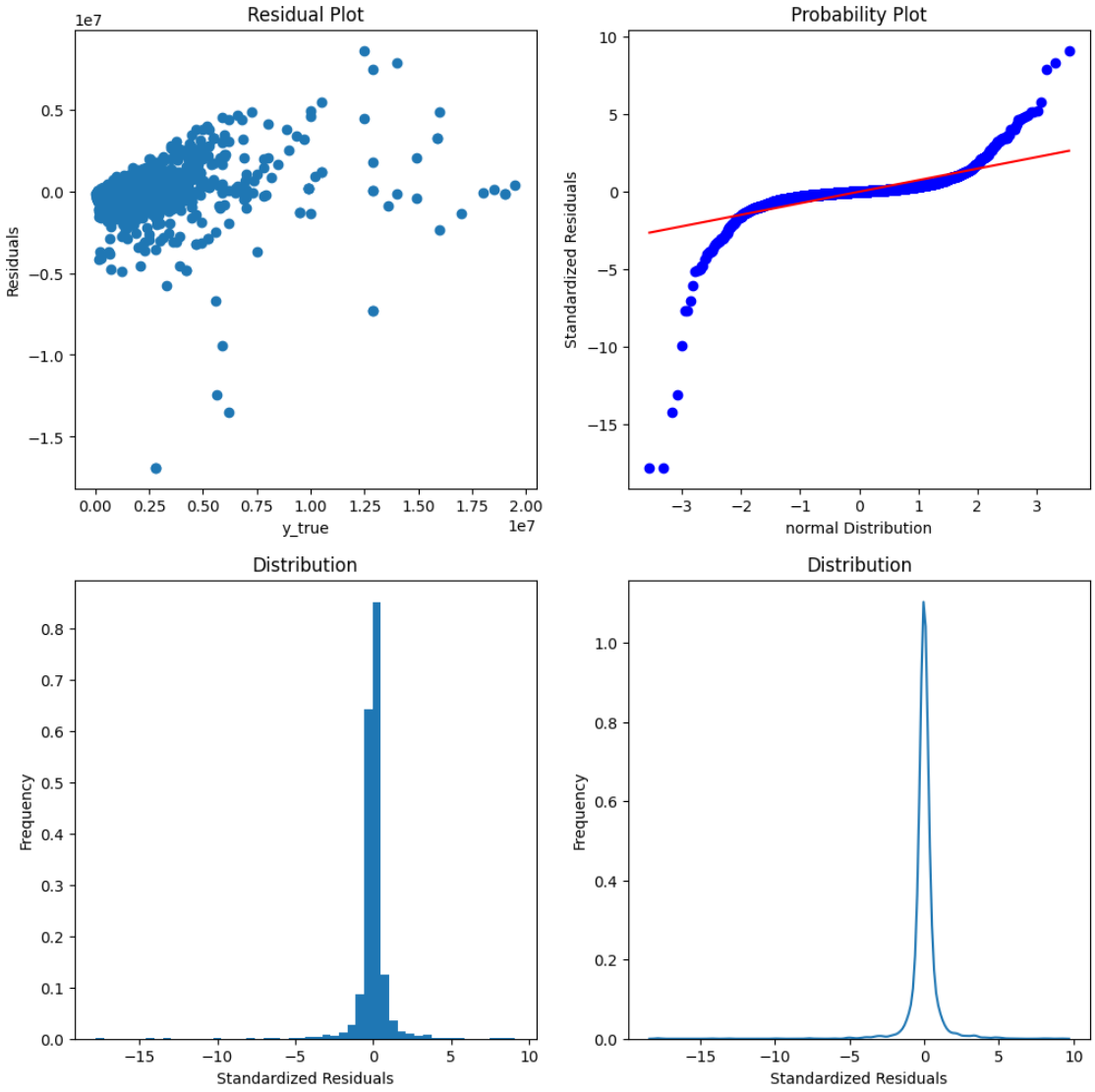
```
    param_grid = {
        'selector__k':[10, 30, 90, 180],
        'model__n_neighbors': [5, 7, 8, 9, 10],
        'model__weights': ['uniform', 'distance']
        },
    n_jobs=-1,
    scoring="neg_mean_squared_error",
    cv=5,
    verbose=3
)

## fit best params
search.fit(
    X = X_train_preprocessed[:3000,:],
    y = y_train[:3000]
)

best_params_KNN = search.best_params_
print(best_params_KNN)

## set best params
pipeline_KNN.set_params(**best_params_KNN)

## fit final model
pipeline_KNN.fit(
    X = X_train_preprocessed,
    y = y_train
)

## output scores:
print(f"""
r2 train: {pipeline_KNN.score(X = X_train_preprocessed, y = y_train)}
r2 test: {pipeline_KNN.score(X = X_test_preprocessed, y = y_test)}
""")

## Save the model with numpy
np.save('./data/KNN.npy', pipeline_KNN)
```

1.8.4 8.4.) Decision Tree

Beste parameter für den Decision Tree Regressor:

- k = 180
- max depth = 15
- min samples leaf = 4

```
MAPE train: 24.92%
MAPE test: 36.87%
```



Code zur Optimierung:

```
## Decision Tree pipeline
pipeline_DecisionTree = Pipeline(steps = [
     ('selector',SelectKBest(mutual_info_regression)),
     ('model',DecisionTreeRegressor())
    ])

## hyperparameter tuning
search = GridSearchCV(
    estimator = pipeline_DecisionTree,
    param_grid = {
        'selector__k':[10, 30, 90, 180],
        'model__max_depth': [9,12,15],
```

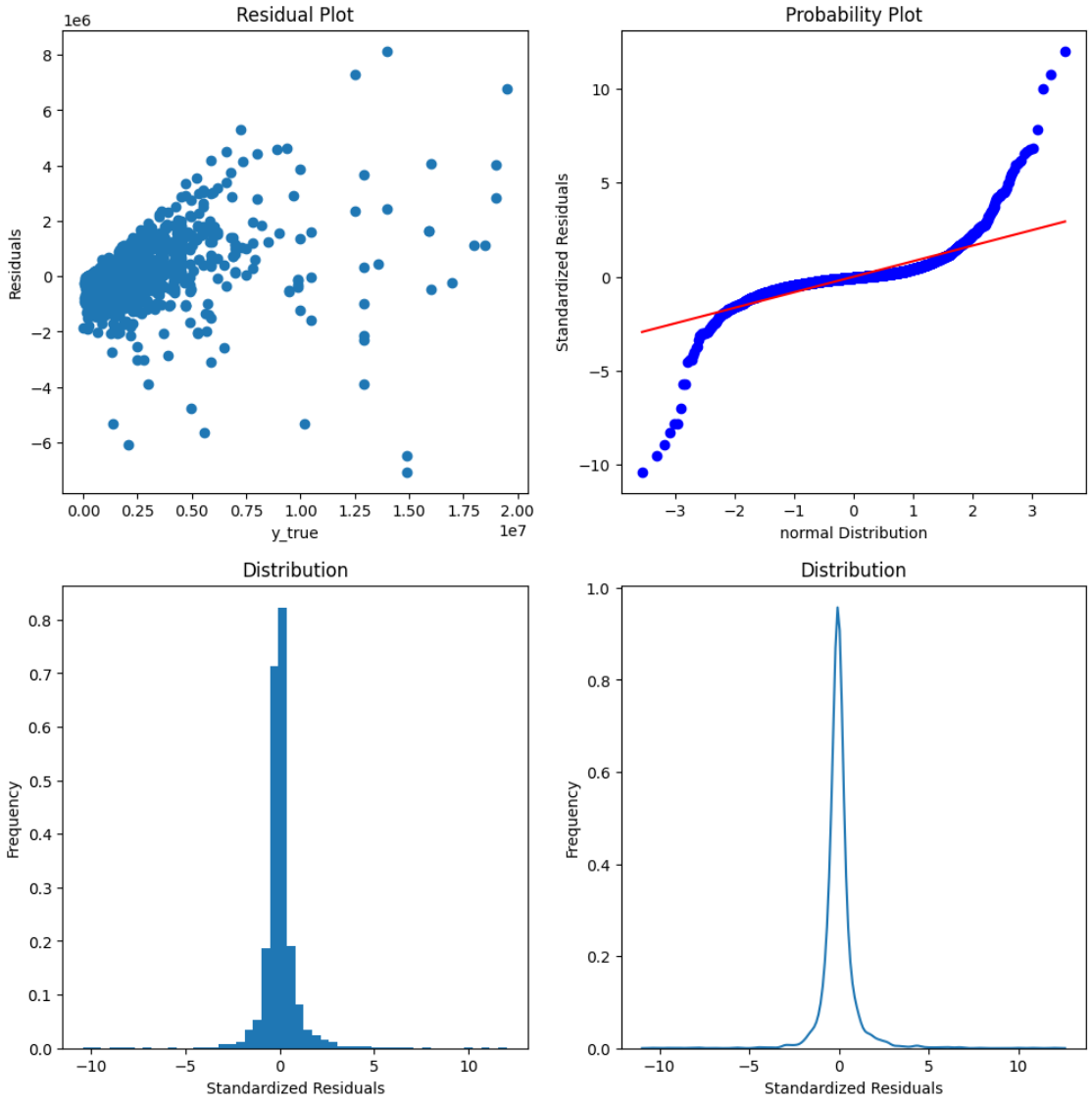
```
        'model__min_samples_split': [2, 4, 8],
        'model__min_samples_leaf': [1, 2, 4]
        },
    n_jobs=-1,
    scoring="neg_mean_squared_error",
    cv=5,
    verbose=3
)

## fit best params
search.fit(
    X = X_train_preprocessed[:3000,:],
    y = y_train[:3000]
)

best_params_DecisionTree = search.best_params_
print(best_params_DecisionTree)

## set best params
pipeline_DecisionTree.set_params(**best_params_DecisionTree)

## fit final model
pipeline_DecisionTree.fit(
    X = X_train_preprocessed,
    y = y_train
)

## output scores:
print(f"""
r2 train: {pipeline_DecisionTree.score(X = X_train_preprocessed, y = y_train)}
r2 test: {pipeline_DecisionTree.score(X = X_test_preprocessed, y = y_test)}
""")

## Save the model with numpy
np.save('./data/DecisionTree.npy', pipeline_DecisionTree)
```

1.8.5 8.5.) Random Forest

Beste parameter für den Random Forest Regressor:

- k = 180
- max depth = 15
- max features = 'sqrt'
- min samples leaf = 4

```
MAPE train: 28.52%
MAPE test: 37.76%
```



Code zur Optimierung:

```
## Random Forest pipleine
pipeline_RandomForest = Pipeline(steps = [
     ('selector',SelectKBest(mutual_info_regression)),
     ('model',RandomForestRegressor())
    ])

## hyperparameter tuning
search = GridSearchCV(
    estimator = pipeline_RandomForest,
    param_grid = {
        'selector__k':[10, 30, 90, 180],
```

```
        'model__n_estimators': [100, 200, 300],
        'model__max_depth': [9, 12, 15],
        'model__min_samples_split': [2, 4, 8],
        'model__min_samples_leaf': [1, 2, 4],
        'model__max_features': ['auto', 'sqrt', 'log2']
        },
    n_jobs=-1,
    scoring="neg_mean_squared_error",
    cv=5,
    verbose=3
)

## fit best params
search.fit(
    X = X_train_preprocessed_reduced[:100,:],
    y = y_train[:100]
)

best_params_RandomForest = search.best_params_
print(best_params_RandomForest)

## set best params
pipeline_RandomForest.set_params(**best_params_RandomForest)

## fit final model
pipeline_RandomForest.fit(
    X = X_train_preprocessed_reduced,
    y = y_train
)

## output scores:
print(f"""
r2 train: {pipeline_RandomForest.score(X = X_train_preprocessed_reduced, y = y_train)}
r2 test: {pipeline_RandomForest.score(X = X_test_preprocessed_reduced, y = y_test)}
""")

## Save the model with numpy
np.save('./data/RandomForest.npy', pipeline_RandomForest)
```

1.8.6 8.6.) Gradient Boosting - Feature Selection with Mutual Information Selection (without Polynomial Features)

Für das folgende Model haben wir den GradientBoosting Algorithmus von sklearn auf unsere Daten Traniert. Dazu haben wir in der Model Pipeline einfach nur das GradientBoosting Modell integriert. Den Feature_Selector haben wir nicht integriert, da das Model im Hintergrund mit DecisionTrees arbeitet, welche automatisch eine Feature Selection durchführen.

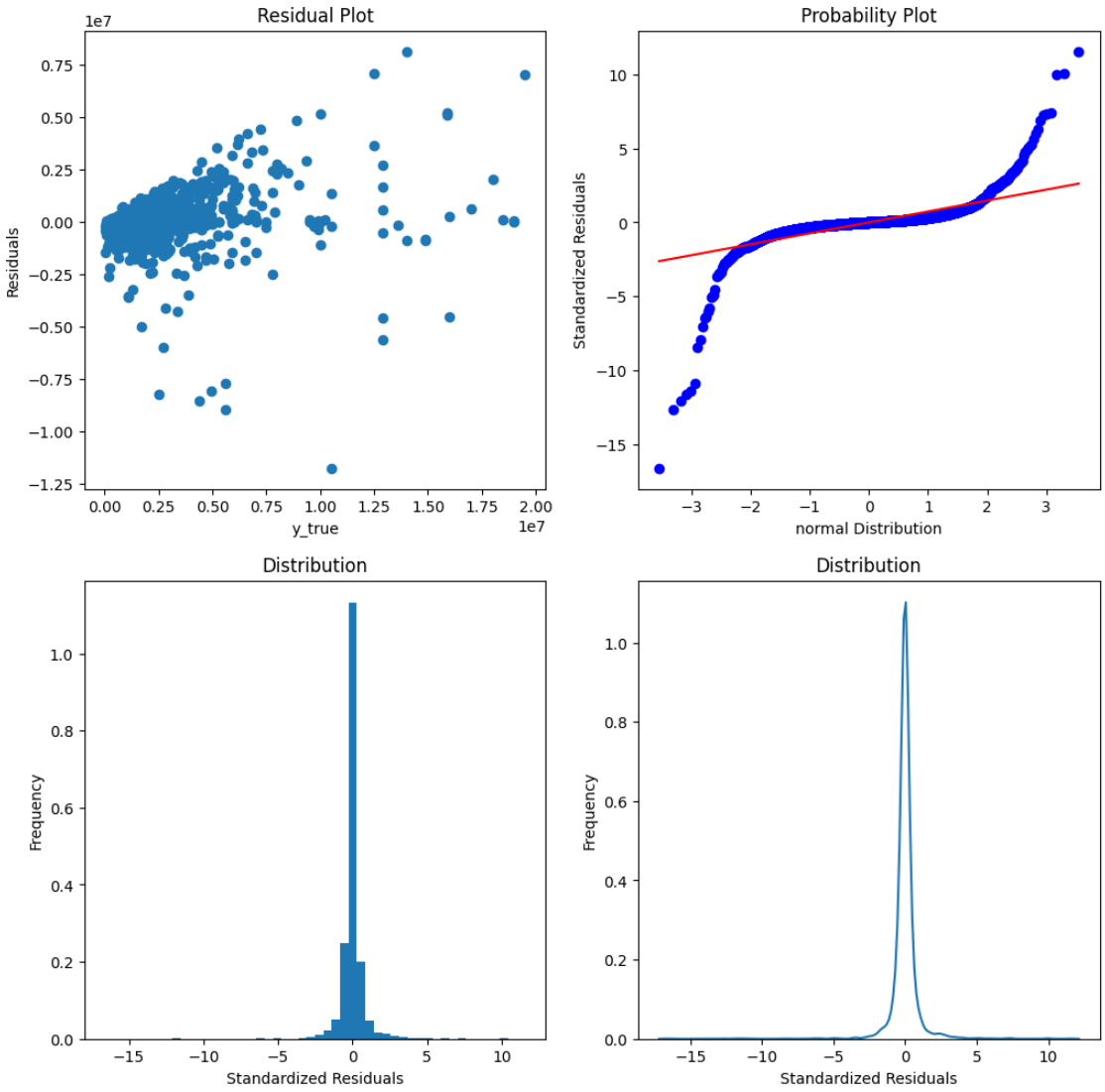
Das Modell ist ein ensemble Modell von der sklearn libary, und funktioniert wie foglt:

1. Das Modell erstellt ein “Initial Guess” für alle Theta Values. Diese werden initialisiert mit den Mean Values.
2. Es werden die Resiudals vom Inital guess als pseudo-Residuals gespeichert
3. Das Modell traniert einen Decision Tree welcher versucht die (pseudo-Residuals) vom inital guess hervorzusagen.
4. Es werden immer wie weitere Modelle aufgrund der Fehler (pseudo-Residuals) der vorherigen Modelle traniert, bis ein endkriterium erreicht wurde, oder die Maximale Anzahl der Modelle erreicht wurde.
5. Die Predictions werden am Schluss aus der Summer des Inital Guess und aller predicteten pseudo-Residuals erzeugt.

Es ist ein relativ weit verbreiteter Algorithmus, weil er für viele verschiedene Probleme eingesetzt werden kann.

Für uns ist es mit einem MAPE von 28.52 das beste Regressionsmodel

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
{'model__max_depth': 9, 'model__min_samples_leaf': 1, 'model__n_estimators':
150}

MAPE train: 13.781684965500107
MAPE test: 28.87439438263531
```



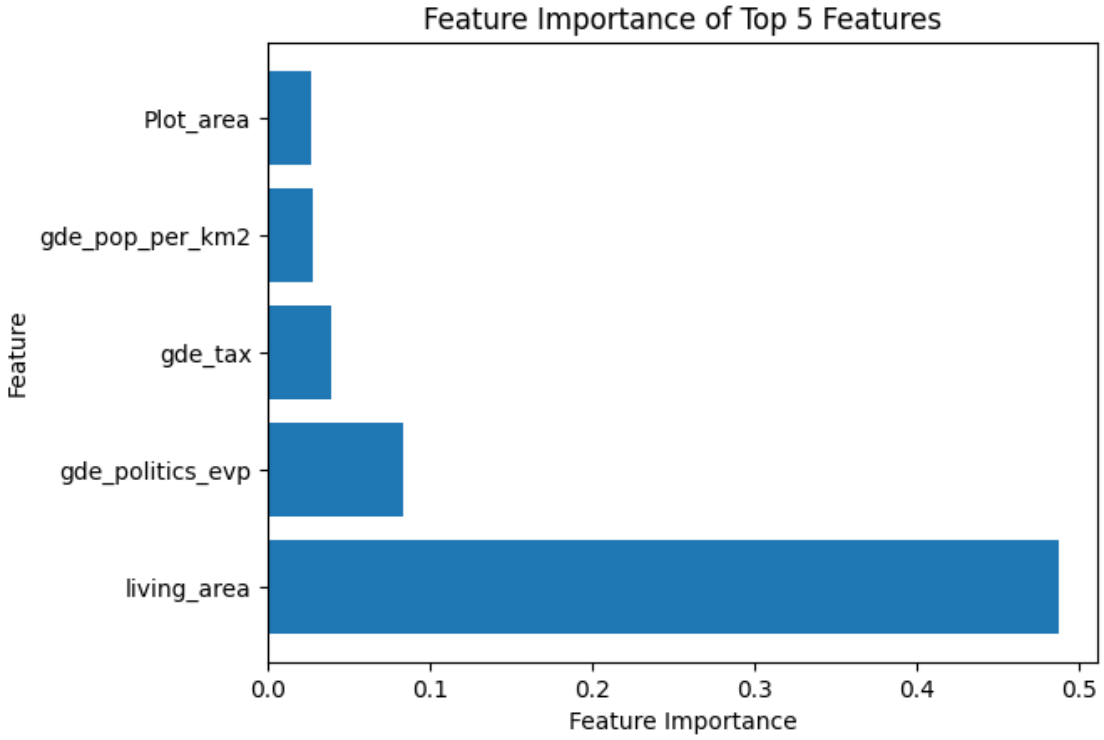
Aufgrund der Feature Importance vom Modell kann erklärt werden, dass das Feature welches mit Abstand am meisten Einfluss auf den Preis hat, das Feature Living Area ist. Mit einem Wert von ca. 0.48, dies war unserer Meinung auch anzunehmen, weil Wohungspreise aus eigener Erfahrung oftmals von der Wohnungsgrösse abhängen.

Weitere Faktoren, welche wir nicht im Dataset hatten, aber gemäss unserer Erfahrung auch einen grossen Einfluss auf die Wohnungspreise haben:

- Wie modern die Küche ist
- Wie modern das Bad ist
- Design der Wohnung (Laminat/Parkett, Betonwände/Gipswände, etc.)

```
c:\Users\rami0\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\utils\deprecation.py:87: FutureWarning: Function
get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will
```

```
be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```



1.9 9.) Reduzierten Classifier und Regressions modelle / Teilaufgabe 3

Da der Classifier nicht Teil der Kaggle Competition ist, verwenden wir nicht denselben Preprocessed Datensatz. Stattdessen wird ein reduzierter Datensatz auf die Features: Fläche, Zimmer, Jahr, PLZ und Preis, angewendet da dies die Merkmale sind, die wir von den Nutzereingaben über unsere API erhalten werden.

1.9.1 9.1.) Daten vorbereitung

Um die Daten für unsere kleine modelle vorbereiten, erstellen wir zwie reduzierte Pipelines, die auf unsere Zielvariablen angepasst sind. Wir erstellen und speichern zwei weil es zwischen dem Regressions und Classifier modelle verschiedene Vorhersagevariablen ja gibt. Aber im Allgemeinen ist die Bereinigungspipeline die gleiche wie die Pipeline des Kaggle-Wettbewerbs

9.1.1.) Test/Train split regressor

```
C:\Users\rami0\AppData\Local\Temp/ipykernel_17048/4162621491.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_reduced_reg['plz'] = X_reduced_reg['plz'].astype(str)
```

```
['./preprocessing_pipeline_reduced_reg.joblib']
```

9.1.2.) Test/Train split classifier

```
C:\Users\rami0\AppData\Local\Temp/ipykernel_17048/793587284.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_reduced_cls['plz'] = X_reduced_cls['plz'].astype(str)
```

```
['./preprocessing_pipeline_reduced_cls.joblib']
```

1.9.2 9.2.) Classifier modelle

9.2.1.) Random forest

	precision	recall	f1-score	support
attic-flat	0.00	0.00	0.00	94
attic-room	0.00	0.00	0.00	3
chalet	1.00	0.03	0.05	119
detached-house	0.55	0.33	0.41	963
duplex-maisonette	0.00	0.00	0.00	113
farmhouse	0.00	0.00	0.00	28
flat	0.53	0.97	0.68	1636
furnished-residential-property	0.00	0.00	0.00	2
loft	0.00	0.00	0.00	6
penthouse	0.00	0.00	0.00	97
rustico	0.00	0.00	0.00	16
semi-detached-house	0.00	0.00	0.00	151
stepped-apartment	0.00	0.00	0.00	29
stepped-house	0.00	0.00	0.00	9
studio	0.00	0.00	0.00	16
terrace-house	0.00	0.00	0.00	89
villa	0.00	0.00	0.00	223
accuracy			0.53	3594
macro avg	0.12	0.08	0.07	3594
weighted avg	0.42	0.53	0.42	3594

9.2.2.) K Neighbours Classifier

	precision	recall	f1-score	support
attic-flat	0.34	0.11	0.16	94
attic-room	0.00	0.00	0.00	3
chalet	0.51	0.30	0.38	119
detached-house	0.60	0.68	0.64	963
duplex-maisonette	0.33	0.08	0.13	113
farmhouse	0.25	0.04	0.06	28
flat	0.69	0.91	0.79	1636
furnished-residential-property	0.00	0.00	0.00	2
loft	1.00	0.17	0.29	6
penthouse	0.16	0.04	0.07	97
rustico	0.00	0.00	0.00	16
semi-detached-house	0.49	0.24	0.32	151
stepped-apartment	0.50	0.10	0.17	29
stepped-house	0.20	0.11	0.14	9
studio	0.67	0.12	0.21	16
terrace-house	0.40	0.21	0.28	89
villa	0.42	0.17	0.24	223
accuracy			0.64	3594
macro avg	0.39	0.19	0.23	3594
weighted avg	0.59	0.64	0.59	3594

9.2.3.) SVC Classifier

	precision	recall	f1-score	support
attic-flat	0.00	0.00	0.00	94
attic-room	0.00	0.00	0.00	3
chalet	0.59	0.32	0.42	119
detached-house	0.57	0.72	0.63	963
duplex-maisonette	0.00	0.00	0.00	113
farmhouse	0.00	0.00	0.00	28
flat	0.66	0.92	0.77	1636
furnished-residential-property	0.00	0.00	0.00	2
loft	0.00	0.00	0.00	6
penthouse	0.00	0.00	0.00	97
rustico	0.00	0.00	0.00	16
semi-detached-house	0.33	0.01	0.03	151
stepped-apartment	0.00	0.00	0.00	29
stepped-house	0.00	0.00	0.00	9
studio	0.00	0.00	0.00	16
terrace-house	0.40	0.02	0.04	89
villa	0.63	0.05	0.10	223
accuracy			0.63	3594

macro avg	0.19	0.12	0.12	3594
weighted avg	0.54	0.63	0.54	3594

9.2.4.) MLP Classifier

	precision	recall	f1-score	support
attic-flat	0.22	0.19	0.20	94
attic-room	0.50	0.33	0.40	3
chalet	0.51	0.48	0.49	119
detached-house	0.61	0.59	0.60	963
duplex-maisonette	0.21	0.14	0.17	113
farmhouse	0.18	0.14	0.16	28
flat	0.73	0.81	0.77	1636
furnished-residential-property	0.00	0.00	0.00	2
loft	0.50	0.17	0.25	6
penthouse	0.13	0.11	0.12	97
rustico	0.07	0.06	0.07	16
semi-detached-house	0.33	0.28	0.30	151
stepped-apartment	0.31	0.17	0.22	29
stepped-house	0.33	0.56	0.42	9
studio	0.23	0.31	0.26	16
terrace-house	0.25	0.26	0.25	89
villa	0.39	0.34	0.36	223
accuracy			0.60	3594
macro avg	0.32	0.29	0.30	3594
weighted avg	0.58	0.60	0.59	3594

9.2.5.) Modell Speichern Da der k-nearest Neighbors Classifier das kleinste Modell ist und die höchste Genauigkeit aufweist, speichern wir ihn zum Einsatz mit unserer API.

1.9.3 9.3.) Regressions modell

```
R2 score:  0.679287992518643
MAPE:  0.3902630937568712
```

9.3.1.) Modell Speichern

1.10 10.) API / Teilaufgabe 2.3

Unsere API ist mit FastAPI aufgebaut und verwendet hypercorn als ASGI-Server. Die API hat ein einzigen Route /predictor/, wo wir die gewünschte Vorhersage typ (Preis oder Haustyp) und die eingegebene Features erhalten.

Die features gehen denn durch die preprocessing pipeline, um sie in die richtige Datenstruktur zu bringen, und gehen danach inzu das angeforderte Model. Sobald wir eine Vorhersage haben wird

sie an den benutzer zurückgeschickt.

Die features die wir erhalten vom user sind:

- Area
- Raume
- Bau Jahr
- Die postleitzahl
- Haustyp/Preis (hängt vom Vorhersagetyp ab)

Die Vorhersage kommt entweder als:

- Preis, gerundet auf die nächsten 10000
- Haustyp

Den code für die API ist im main.py zu finden.

```
[NbConvertApp] Converting notebook Final_Bericht_(1).ipynb to pdf
[NbConvertApp] Support files will be in Final_Bericht_(1)_files\
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Making directory .\Final_Bericht_(1)_files
[NbConvertApp] Writing 135498 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | b had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 926103 bytes to Final_Bericht_(1).pdf
```