Project_Report_Draft

June 30, 2023

```
<div style="display:flex; flex-direction:column">
  <h1 style="position: relative; margin-bottom: 0px">Report: Vehicle Leasing Price Predictor</h
  <h3 style="margin-top: 0px"> A Machine Learning approach </h3>
</div>
<div style="display: flex; align-items: center;">
  <div style="width:auto; height:75px"><a href="https://www.wu.ac.at"><img src="components/wu-l
  <div style="width:auto; height:75px"><a href="https://www.zeb-consulting.com/"><img src="comp
</div>
<!-- absolute; top: 0; right: 150px; -->
<!-- style="position: absolute; top: 0; right: 0;" -->

<p style="margin-bottom:0px">Authors:</p>
<ul style="list-style-type: disc; padding-left: 20px0;margin-top:0px">
  <li>Tobias Ponesch</li>
  <li>Sina Haghgoo</li>
  <li>Finnian John Dempsey</li>
  <li>Vinicius Wolff</li>
  <li>Adrian Lehrner</li>
  <li>Mario Dangev</li>
</ul>

<p style="margin-bottom:0px">Coaches:</p>
<ul style="list-style-type: disc; padding-left: 20px0;margin-top:0px">
  <li>Thomas Dornigg
    <img src="components/zeb-logo2.png" alt="ZEB" width="30px" height="auto">
    <a href="https://at.linkedin.com/in/thomas-dornigg"><img src="components/LinkedIn_logo_init
  </li>
  <li>Jannik Neuböck
    <img src="components/zeb-logo2.png" alt="ZEB" width="30" height="auto">
    <a href="https://www.linkedin.com/in/jannik-neub%C3%B6ck-b71775194/?originalSubdomain=at"><
  </li>
</ul>

<p style="margin-bottom:0px">Advising Professor:</p>
<ul style="list-style-type: disc; padding-left: 20px0;margin-top:0px;">
  <li>
    Ronald Hochreiter
    <img src="components/wu-logo.png" alt="ZEB" width="30" height="auto">
    <a href="https://www.linkedin.com/in/ronaldhochreiter/?originalSubdomain=at"><img src="comp
  </li>
```


We are immensely grateful for the collaboration and support of zeb Consulting, which has greatly contributed to the success of this project.

0.1 Introduction

This project aims to develop a machine learning model to predict the leasing prices of vehicles based on various attributes. In the current macroeconomic environment, accurately forecasting leasing asset values and pricing is crucial for leasing banks. Additionally, the automotive market has experienced significant price fluctuations and supply chain disruptions, further emphasizing the need for reliable predictions. Leveraging a dataset provided by a leasing bank, our research and development efforts focus on building the most accurate prediction models. The final outcome will be a graphical user interface (GUI) that allows users to input vehicle details and obtain leasing rate predictions. By employing state-of-the-art machine learning techniques and addressing challenges such as data quality and model selection, this project aims to provide an effective tool for leasing banks in assessing asset values and making informed pricing decisions.

0.2 Table of Contents

0.3 1 Import Libraries

To develop high-quality machine learning algorithms and streamline data processing, we utilized state-of-the-art libraries such as pandas, numpy, and sklearn in this project. These libraries enabled us to implement advanced techniques and achieve efficient data manipulation and analysis. If you run into any problems with the imported libraries, please refer to the Debugging library versions section.

```
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_clustering.py:35: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _pt_shuffle_rec(i, indexes, index_mask, partition_tree, M, pos):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_clustering.py:54: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def delta_minimization_order(all_masks, max_swap_size=100, num_passes=2):
```

```
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_clustering.py:63: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window(order, start, length):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_clustering.py:69: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window_score_gain(masks, order, start, length):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_clustering.py:77: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _mask_delta_score(m1, m2):
/Users/tobias/opt/anaconda3/lib/python3.9/site-packages/shap/links.py:5:
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to
the 'numba.jit' decorator. The implicit default value for this argument is
currently False, but it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def identity(x):
/Users/tobias/opt/anaconda3/lib/python3.9/site-packages/shap/links.py:10:
```

```
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to
the 'numba.jit' decorator. The implicit default value for this argument is
currently False, but it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _identity_inverse(x):
/Users/tobias/opt/anaconda3/lib/python3.9/site-packages/shap/links.py:15:
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to
the 'numba.jit' decorator. The implicit default value for this argument is
currently False, but it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def logit(x):
/Users/tobias/opt/anaconda3/lib/python3.9/site-packages/shap/links.py:20:
NumbaDeprecationWarning: The 'nopython' keyword argument was not supplied to
the 'numba.jit' decorator. The implicit default value for this argument is
currently False, but it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _logit_inverse(x):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_masked_model.py:363: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _build_fixed_single_output(averaged_outs, last_outs, outputs,
batch_positions, varying_rows, num_varying_rows, link, linearizing_weights):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_masked_model.py:385: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
```

```
  def _build_fixed_multi_output(averaged_outs, last_outs, outputs,
batch_positions, varying_rows, num_varying_rows, link, linearizing_weights):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_masked_model.py:428: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _init_masks(cluster_matrix, M, indices_row_pos, indptr):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/utils/_masked_model.py:439: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _rec_fill_masks(cluster_matrix, indices_row_pos, indptr, indices, M, ind):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/maskers/_tabular.py:186: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _single_delta_mask(dind, masked_inputs, last_mask, data, x, noop_code):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/maskers/_tabular.py:197: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _delta_masking(masks, x, curr_delta_inds, varying_rows_out,
/Users/tobias/opt/anaconda3/lib/python3.9/site-
```

```
packages/shap/maskers/_image.py:175: NumbaDeprecationWarning: The 'nopython'
keyword argument was not supplied to the 'numba.jit' decorator. The implicit
default value for this argument is currently False, but it will be changed to
True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _jit_build_partition_tree(xmin, xmax, ymin, ymax, zmin, zmax,
total_ywidth, total_zwidth, M, clustering, q):
/Users/tobias/opt/anaconda3/lib/python3.9/site-
packages/shap/explainers/_partition.py:676: NumbaDeprecationWarning: The
'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The
implicit default value for this argument is currently False, but it will be
changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
  def lower_credit(i, value, M, values, clustering):
The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but
it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
The 'nopython' keyword argument was not supplied to the 'numba.jit'
decorator. The implicit default value for this argument is currently False, but
it will be changed to True in Numba 0.59.0. See
https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-
of-object-mode-fall-back-behaviour-when-using-jit for details.
```

0.4 2 Computational effort

Building Machine Learning models, or any kind of models, can be very computationally expensive and time consuming. To optimize the computational and time requirements of building machine learning models, we have introduced a section that allows you to choose between computing new models or importing existing ones. Additionally, you will be prompted to evaluate your machine's performance, ranging from "Ludicrous" (highest performance) to "Low" (lowest performance). Your response will determine the number of iterations and cross-validations in the subsequent Random Search process. Moreover, the number of threads used for model building is limited to your available threads minus two, ensuring that your machine remains usable during the process. This approach aims to strike a balance between model generation and system usability. Due to our struggles with using laptops for building the machine learning models, we used a Windows PC with the following

specs for computationally expensive tasks: - **CPU:** Ryzen 7 5800X, 8 Cores, 16 Threads, 3800MHz Base Clock, 4700MHz Boost Clock - **RAM:** 32GB 3200MHz - **GPU:** RTX 2080ti - **Windows Distribution:** Windows 11 Pro, 10.0.22621

```
Your available threads: 8
```

0.5 3 Dataset import

Our dataset was imported from an Excel file that was generously provided by our data coaches. This dataset serves as the foundation for our data science project. However, it is important to note that the information contained in the dataset is artificial and does not necessarily reflect perfect accuracy in relation to real-world data. To ensure flexibility and enable user experimentation, our code includes functionality that allows users to easily change the dataset that will be imported and used to train the models. This feature provides the opportunity for further exploration and analysis with different datasets, enabling users to assess the impact of dataset variations on model performance.

The following table shows the raw dataset.

```
            brand                                model      milage registration  \
0           Skoda  Octavia ŠKODA Combi Style TDI DSG      201 km      03/2023  \
1      Volkswagen                T-Cross VW Life TSI      201 km      03/2023
2            Seat              Ibiza Austria Edition   15.000 km      10/2022
3      Volkswagen                             Polo VW        1 km      01/2023
4            Audi     A4 Avant 40 TDI quattro S line  105.301 km      12/2019
…            …                                    …          …           …
19053        Seat        Ateca FR 2.0 TDI DSG 4Drive      201 km      01/2023
19054       Skoda  Octavia ŠKODA Combi Style TDI DSG      201 km      03/2023
19055        Audi     A4 Avant 40 TDI quattro S line  105.301 km      12/2019
19056  Volkswagen                             Polo VW   18.903 km      06/2020
19057  Volkswagen                 Tiguan VW Life TDI   48.000 km      09/2022

                   duration                gear       fee  emission
0      48 Monat (anpassbar)           Automatik  574,01 €  119 g/km  \
1      48 Monat (anpassbar)  Manuelle Schaltung  382,58 €  131 g/km
2      48 Monat (anpassbar)  Manuelle Schaltung  239,62 €  120 g/km
3      48 Monat (anpassbar)  Manuelle Schaltung  309,11 €  127 g/km
4      48 Monat (anpassbar)           Automatik  587,75 €  138 g/km
…                    …                  …       …       …
19053  48 Monat (anpassbar)           Automatik  692,03 €  146 g/km
19054  48 Monat (anpassbar)           Automatik  574,01 €  187 g/km
19055  48 Monat (anpassbar)           Automatik  587,75 €  143 g/km
19056  48 Monat (anpassbar)  Manuelle Schaltung  256,33 €   40 g/km
19057  48 Monat (anpassbar)  Manuelle Schaltung  539,72 €  185 g/km

        consumption horsepower kilowatts    fuel
0      5,0 l/100 km     150 PS    110 kW  Diesel
1      6,0 l/100 km      95 PS     70 kW  Benzin
2      5,0 l/100 km      80 PS     59 kW  Benzin
3      6,0 l/100 km      80 PS     59 kW  Benzin
```

```
4      5,0 l/100 km     190 PS    140 kW  Diesel
…             …         …       …      …
19053  6,0 l/100 km     150 PS    110 kW  Diesel
19054  8,0 l/100 km     150 PS    110 kW  Diesel
19055  6,0 l/100 km     190 PS    140 kW  Diesel
19056  2,0 l/100 km      80 PS     59 kW  Benzin
19057  8,0 l/100 km     122 PS     90 kW  Diesel

[19058 rows x 12 columns]
```

0.6 4 Basic Preprocessing

In order to ensure data usability, certain features require formatting adjustments. These adjustments involve removing units, replacing commas with decimal points, and calculating the age based on registration information. These transformations are performed within the "basic preprocessing pipeline" to prepare the data for further analysis and modeling.

```
None
```

```
ColumnTransformer(remainder='passthrough',
                  transformers=[('age', CalculateAge(), ['registration']),
                                ('unit', RemoveUnits(),
                                 ['milage', 'duration', 'fee', 'emission',
                                  'consumption', 'horsepower', 'kilowatts'])],
                  verbose_feature_names_out=False)
```

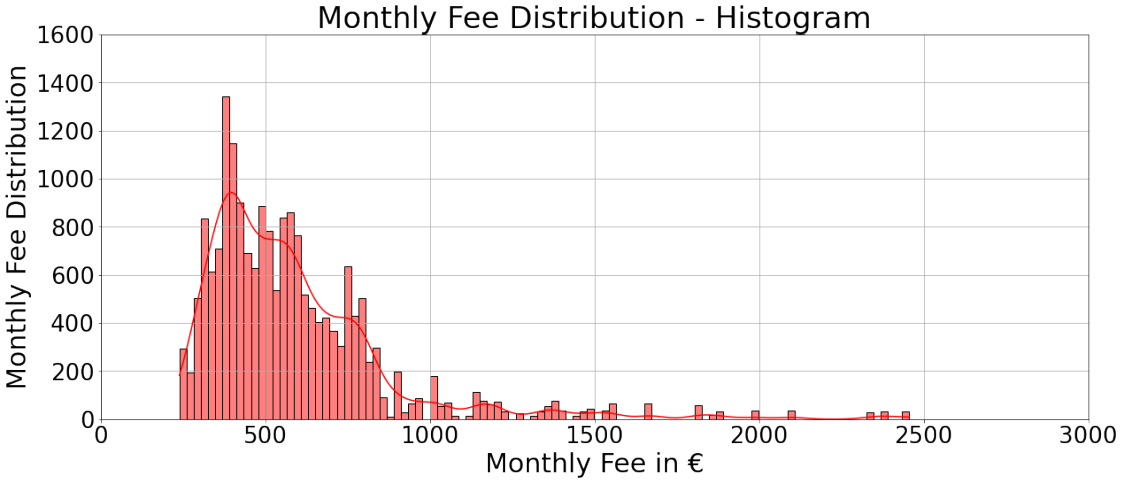
```
   registration    milage  duration     fee  emission  consumption
0             3     201.0      48.0  574.01     119.0          5.0  \
1             3     201.0      48.0  382.58     131.0          6.0
2             8   15000.0      48.0  239.62     120.0          5.0
3             5       1.0      48.0  309.11     127.0          6.0
4            42  105301.0      48.0  587.75     138.0          5.0

   horsepower  kilowatts       brand                              model  \
0       150.0      110.0       Skoda  Octavia ŠKODA Combi Style TDI DSG  \
1        95.0       70.0  Volkswagen                T-Cross VW Life TSI
2        80.0       59.0        Seat               Ibiza Austria Edition
3        80.0       59.0  Volkswagen                            Polo VW
4       190.0      140.0        Audi     A4 Avant 40 TDI quattro S line

                 gear    fuel
0           Automatik  Diesel
1  Manuelle Schaltung  Benzin
2  Manuelle Schaltung  Benzin
3  Manuelle Schaltung  Benzin
4           Automatik  Diesel
```

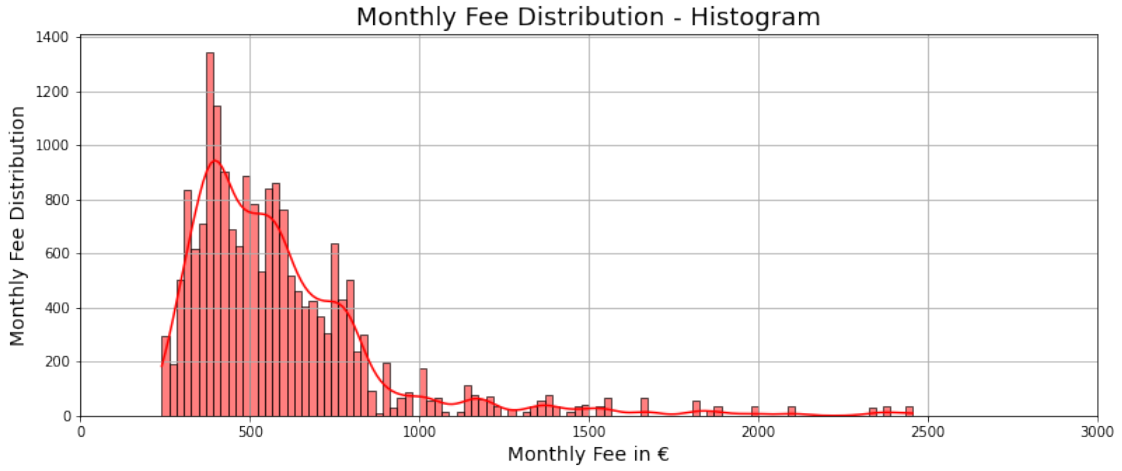
0.7 5 Explanatory Data Analysis

This section contains the visualization of the dataset, which serves as an essential step in understanding the underlying patterns and relationships within the data. This exploratory analysis aims to provide insights into the dataset's characteristics and unveil meaningful trends that can guide subsequent modeling efforts.

0.7.1 5.1 The Traget variable

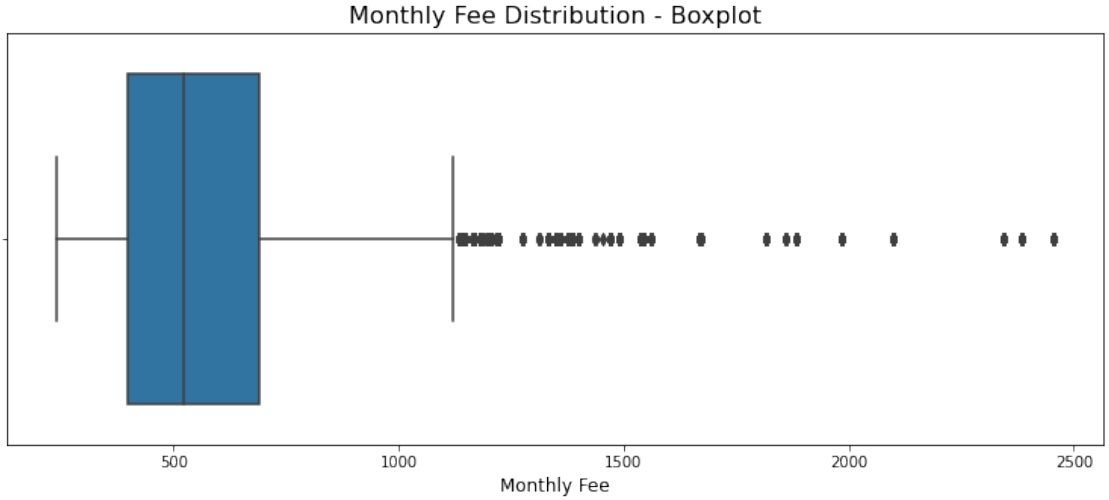
In our scenario, the target variable refers to the variable that we aim to predict. Specifically, the Monthly Fee is the unknown value we are seeking to estimate.





On this histogramm we see the monthly fee distrubtion for the cars in the dataset. The distribution exhibits a right-skewed pattern, indicating that a majority of the leasing rates fall towards the lower end of the scale with minimum leasing rates. starting from approximately 250 euros. The peak of the distribution occurs around 350 euros, with a significant number of cars falling within this

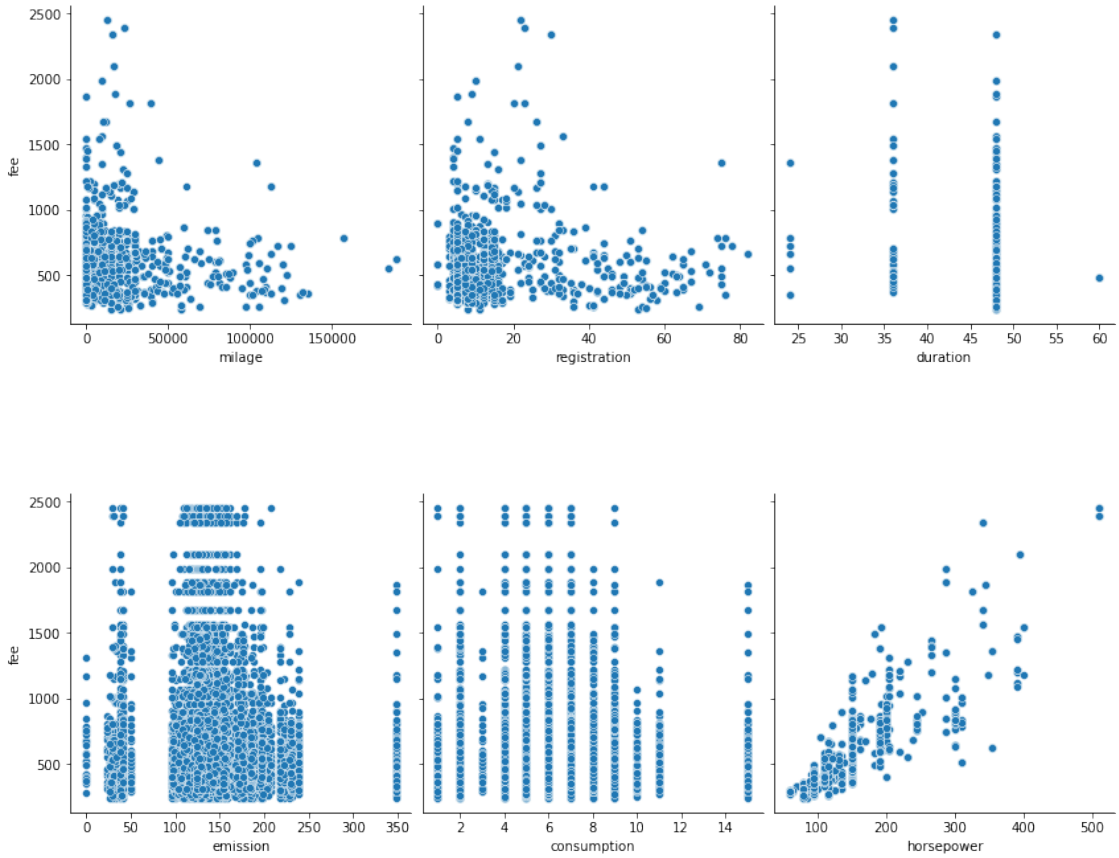
monthly fee range. Most of the values lie between 300-900 euros. What is interresting is that we can see that there are very few cars that have a leasing rate between 2000 and 2500 euros. Those cars are not outliers but rather just very expensive cars.



The boxplot for the monthy fee distribution verifies what we observed from the histogram. The minimum monthly fee value is approximately 250 euros, while the maximum reaches up to 1200 euros. The median, which represents the middle value of the distribution, lies slightly above 500 euros. Upon closer examination, it becomes apparent that the boxplot exhibits numerous outliers. However (as stated earlier), these outliers are not indicative of data anomalies but rather represent the presence of very expensive cars with exceptionally high leasing rates. This observation highlights the diversity within the dataset, as it encompasses both affordable and luxury vehicles.

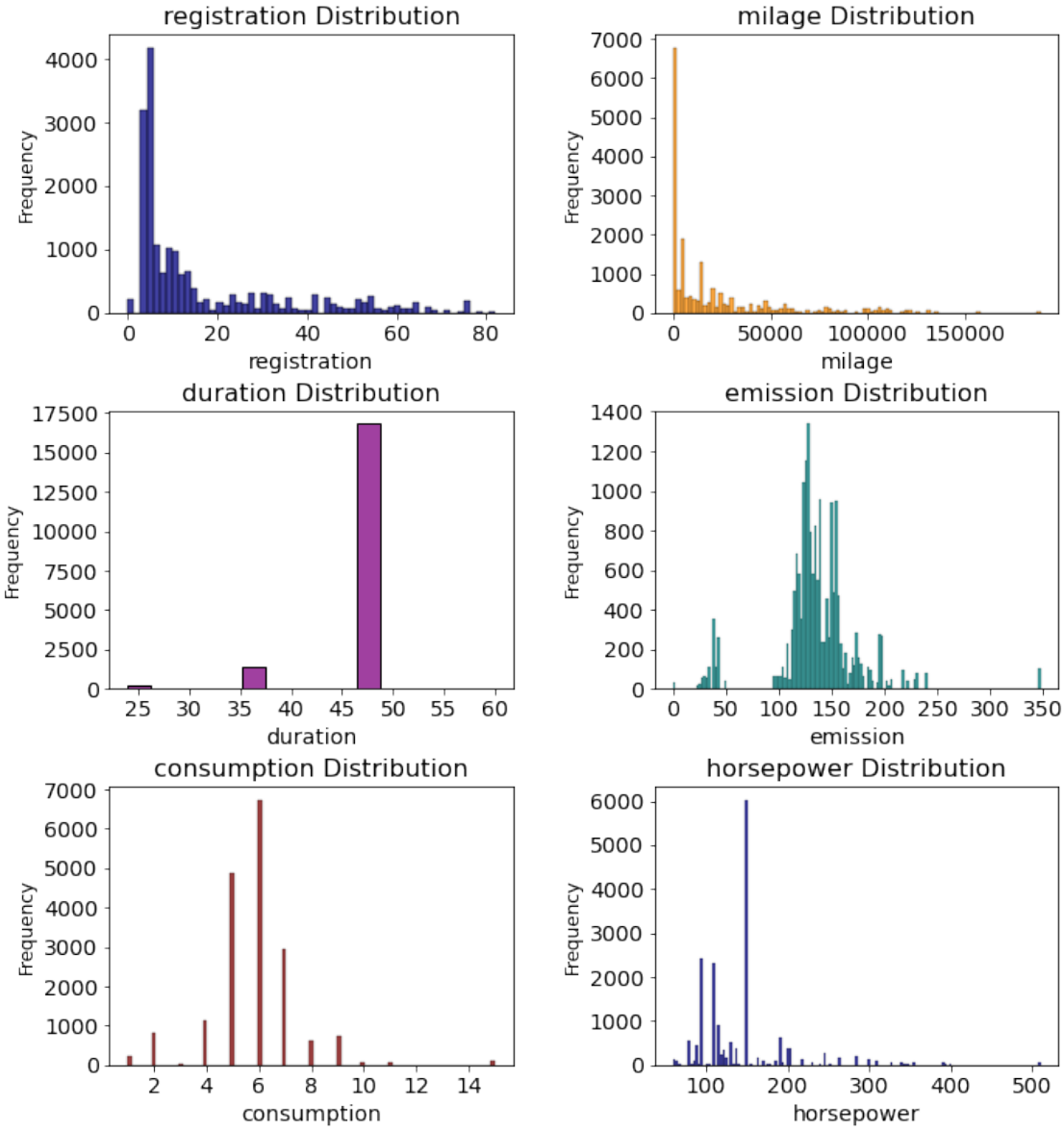
0.7.2 5.2 Numerical Features

Features in a machine learning model are the input variables used to make predictions or classifications. They provide the necessary information for the model to learn and make decisions. The numerical features gathered are: - **Mileage:** The total distance traveled by the vehicle in miles. This feature provides information about the wear and tear on the vehicle and can impact its value and leasing price. - **First Registration:** The duration, represented in months, since the vehicle's initial registration. This feature indicates the age of the vehicle and can affect its depreciation and leasing price. - **Duration:** The duration of the leasing contract in months. This feature represents the length of time for which the vehicle is leased and influences the leasing price. - **Emissions:** The amount of emissions produced by the vehicle, typically measured in grams of CO2 per kilometer. This feature provides insights into the environmental impact of the vehicle and may impact leasing decisions, especially in regions with emissions regulations. - **Consumption:** The fuel consumption of the vehicle, typically measured in liters per 100 kilometers. This feature indicates the efficiency of the vehicle in terms of fuel usage. - **Horsepower:** The measure of the vehicle's engine power. This feature represents the strength and performance capabilities of the vehicle. - **Kilowatts:** The power of the vehicle's engine in kilowatts. This feature provides an alternative representation of the engine power.



Scatterplots provide a visual representation to explore the relationship between numerical variables and the target variable in our dataset. In our analysis, we plotted our target variable on the y-axis against all the numeric variables on the x-axis. From the scatterplots, several key findings emerged. First, a strong linear relationship was observed between the monthly fee and the horsepower. As the horsepower increases, the monthly fee tends to rise as well, indicating a positive correlation between these variables. Additionally, we discovered that lower mileage and more recent initial registrations are associated with higher monthly fees. This relationship is evident from the concentrated distribution of points in the corresponding regions of the scatterplots, suggesting that these factors have a noticeable impact on leasing rates. Examining the scatterplots for consumption and duration, we noticed an interresting pattern. Despite variations in duration and consumption, the monthly fees are relatively evenly distributed. This implies that these variables may not exert a significant influence on the monthly fee, as indicated by the consistent spread of points across different durations and consumption levels. Regarding the emission variable, we observed a dense cluster of points in the middle range, signifying a large number of vehicles with emission values between approx. 100-250. This indicates that both low-cost and high-cost vehicles exist within this emission range, potentially reflecting a diverse market segment with various pricing factors beyond emissions alone. Overall, these insights from the scatterplots shed light on the relationships between the numerical variables and the monthly fee, providing valuable information for feature selection and understanding the factors influencing leasing rates.

5.2.1 Skewness of numerical variables



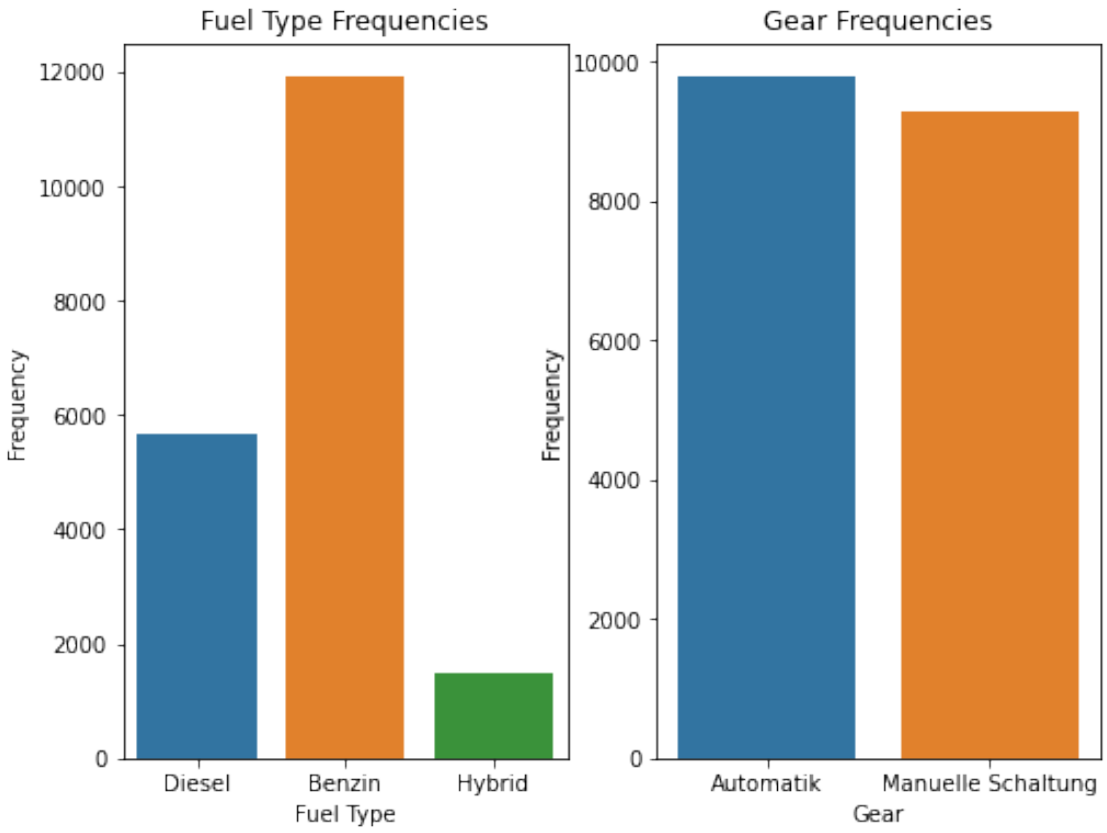
When examining the skewedness of the numerical variables in our dataset, interesting patterns emerge. Specifically, the registration and mileage variables exhibit right-skewed distributions, indicating a high concentration of new vehicles. This suggests that a significant portion of the dataset comprises recently registered vehicles with relatively low mileage.

In contrast, the emission and consumption variables demonstrate symmetric distributions, implying a more balanced distribution of values. The absence of skewness in these variables suggests that the dataset encompasses a diverse range of emission and consumption values without a pronounced bias towards higher or lower values.

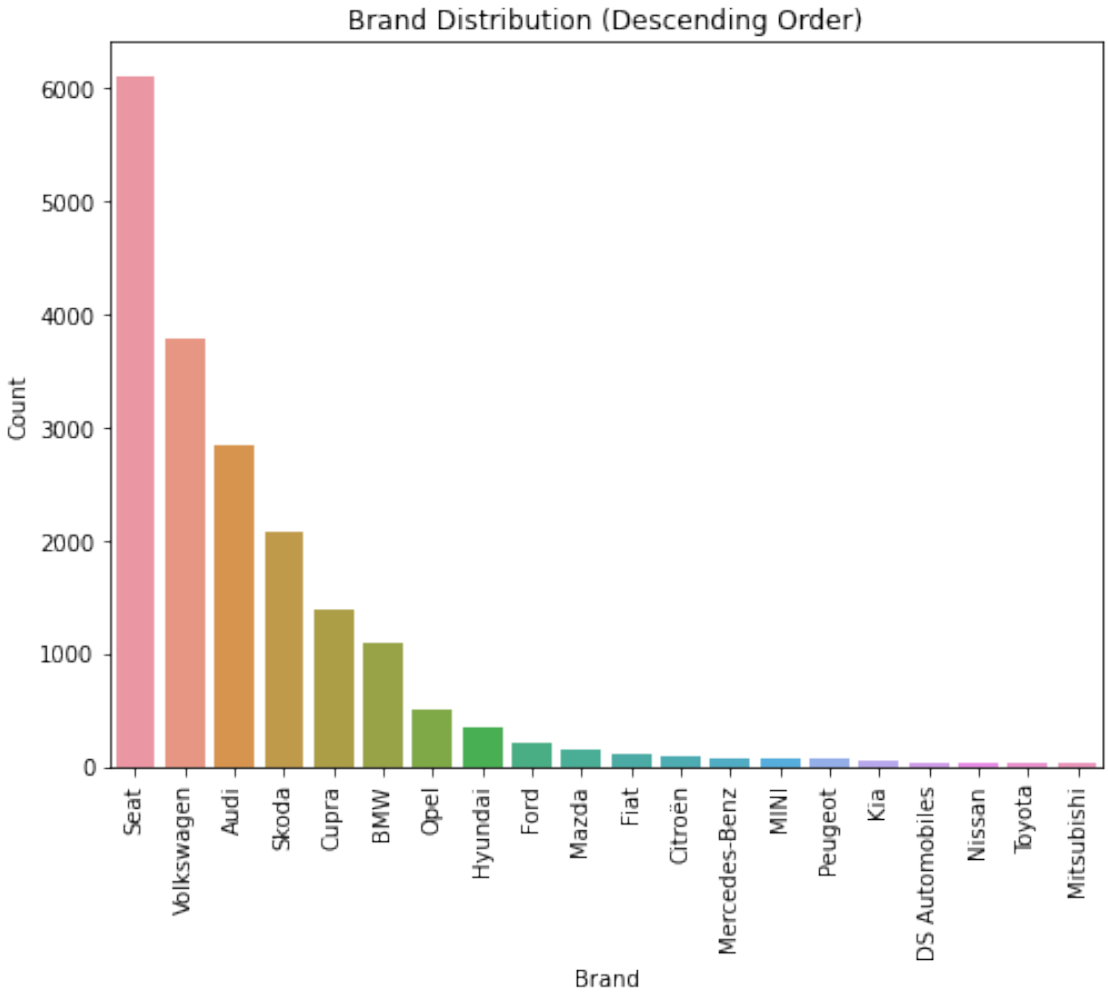
0.7.3 5.3 Categporical Features

The categorical features gathered are: - Brand: The brand or manufacturer of the vehicle. This feature represents the specific company or brand associated with the vehicle's production. - Model: The specific model or variant of the vehicle. This feature provides detailed information about the vehicle's specific version or edition. - Gear: The type of gear or transmission system used in the vehicle. This feature indicates whether the vehicle has a manual or automatic gear system. - Fuel: The type of fuel used by the vehicle. This feature represents the fuel source required for the vehicle's operation, such as gasoline, diesel, or hybrid.

```
            brand                                 model                gear  \
0           Skoda  Octavia ŠKODA Combi Style TDI DSG            Automatik
1      Volkswagen                T-Cross VW Life TSI  Manuelle Schaltung
2            Seat              Ibiza Austria Edition  Manuelle Schaltung
3      Volkswagen                             Polo VW  Manuelle Schaltung
4            Audi     A4 Avant 40 TDI quattro S line            Automatik
…            …                                     …                …
19053        Seat        Ateca FR 2.0 TDI DSG 4Drive            Automatik
19054       Skoda  Octavia ŠKODA Combi Style TDI DSG            Automatik
19055        Audi     A4 Avant 40 TDI quattro S line            Automatik
19056  Volkswagen                             Polo VW  Manuelle Schaltung
19057  Volkswagen                 Tiguan VW Life TDI  Manuelle Schaltung

          fuel
0       Diesel
1       Benzin
2       Benzin
3       Benzin
4       Diesel
…        …
19053   Diesel
19054   Diesel
19055   Diesel
19056   Benzin
19057   Diesel

[19058 rows x 4 columns]
```



The barplots provide insights into the fuel types and gear types of the vehicles in our dataset. We can observe that the dataset comprises three primary fuel types: diesel, gasoline, and hybrid vehicles. Additionally, the barplots reveal the presence of two gear types: automatic and manual shifts. This highlights the transmission options available within the datase.

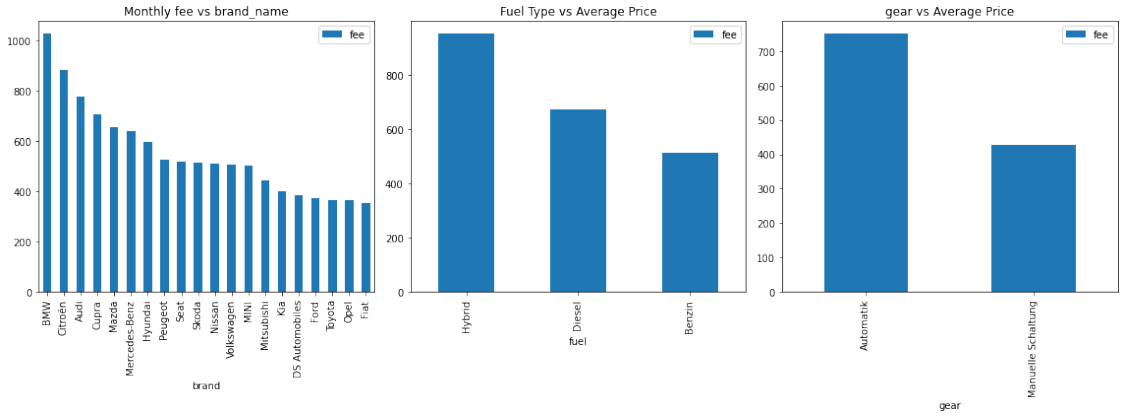


Upon exploring the brands present in our dataset, we identified several notable findings. The most frequently occurring brands among the vehicles in our dataset are Seat, Volkswagen, Audi, Skoda, Cupra, BMW, and Opel.

The inclusion of these brands in our dataset represents a mix of mainstream vehicles, providing a comprehensive view of the market and enabling us to analyze the impact of brand types on the monthly leasing fees.

0.7.4 5.4 Target variable vs. categorical features

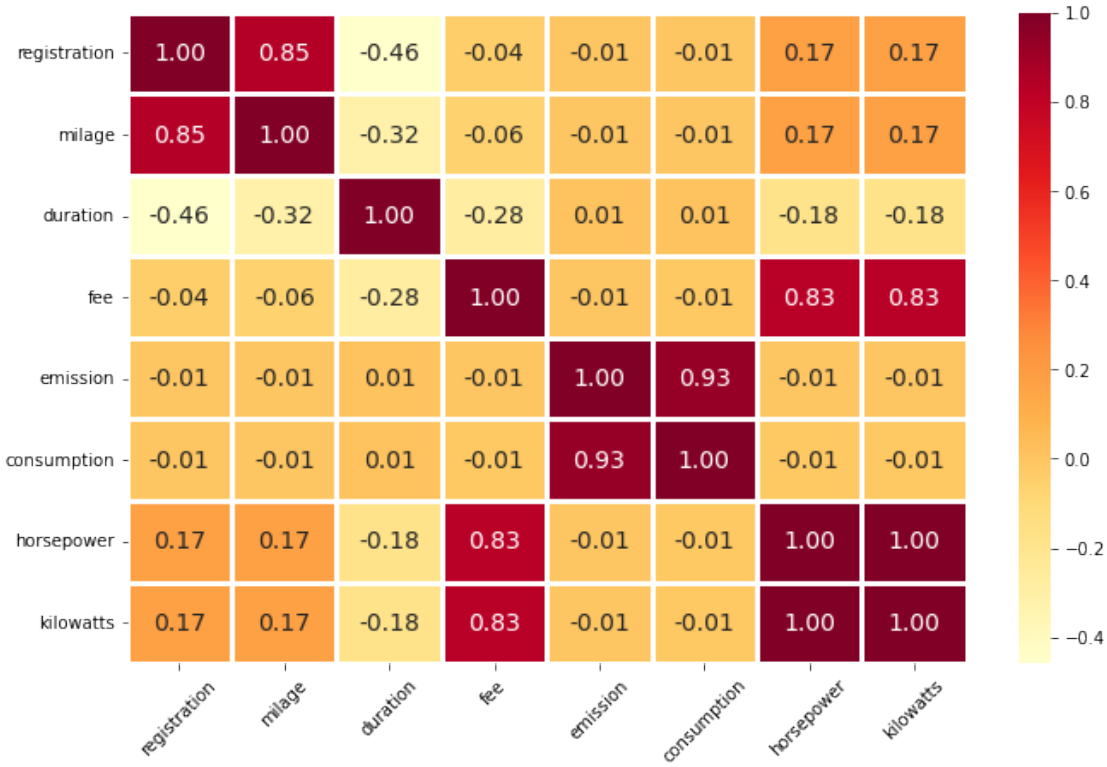
This section provides an overview of how the categorical features influence the target variable, which is the monthly fee.



Analyzing the relationship between the target variable and the categorical variables in our dataset revealed interesting insights. Plotting the target variable against different categorical variables allowed us to examine their impact on leasing rates. From the visualizations, we observed that BMW models, Citroën, and Audi tend to have higher average leasing prices compared to other brands. Moreover, when considering the fuel type, we found that hybrid vehicles tend to have higher monthly fees compared to gasoline or diesel vehicles. This is attributed to the increased cost of hybrid technology and the potential for fuel savings over time. Additionally, we observed that automatic cars are generally more expensive to lease compared to cars with manual shifting. This is very likely perceived convenience and comfort associated with automatic transmission, which can contribute to higher demand and pricing.

0.7.5 5.5 Heatmap (Correlations)

In this section, we have included a correlation plot to examine the relationships between the features within the dataset. This plot serves as a visual representation of the correlations and allows us to gain valuable insights into the interdependencies among the various attributes. By analyzing the correlations, we can determine whether certain variables exhibit positive or negative relationships. This comprehensive overview encompasses both numerical and categorical features, enabling us to identify important patterns and dependencies that may influence the target variable. The insights derived from these correlations are crucial for subsequent analysis and model development, as they provide valuable information for further investigation and interpretation.



This correlation matrix measures the relationships between different variables related to a car, namely: milage, first_registration, duration, monthly_fee, emission_value, consumption, horsepower, and kilowatts. The correlations range from -1 (perfect negative correlation, as one variable increases, the other decreases) through 0 (no correlation, the variables do not move together) to +1 (perfect positive correlation, the variables increase and decrease together).

- **Monthly_fee and Horsepower/Kilowatts**:These pairs show very strong positive correlations (0.827053 and 0.826905, respectively). This suggests that cars with higher horsepower or kilowatts are associated with higher monthly fees. This could mean that more powerful vehicles tend to have higher monthly costs, potentially due to reasons such as higher insurance premiums, increased fuel consumption, or greater maintenance requirements.
- **Monthly_fee and Duration**:There is a moderate negative correlation (-0.280965) between these variables, indicating that as the duration of ownership increases, the monthly fee decreases. This could be because the costs associated with a car (like loan payments or certain insurance costs) often decrease over time.
- **Monthly_fee and Mileage/First_registration**:The correlation between monthly_fee and these two variables is relatively weak (-0.060930 and -0.041417, respectively). This suggests that the monthly fee does not change significantly with changes in the mileage of the car or its first registration date. It's worth noting though, that in some cases, older cars (with an earlier first registration date) or cars with higher mileage could potentially have higher maintenance costs which could affect the monthly fee.
- **Monthly_fee and Emission_value/Consumption**:The correlations here are also very

weak (-0.008253 and -0.012807, respectively). These small negative correlations suggest that cars with higher emissions or consumption are associated with slightly lower monthly fees, although this relationship is very weak. This may be because cars with higher emissions or fuel consumption tend to be older models, which could have lower associated costs in some areas (like lower insurance or depreciated value).

- **Mileage and First_registration**:These two variables show a strong positive correlation of 0.845908, implying that as the age of the car (as suggested by the first registration) increases, so does the mileage it has run. This is a fairly intuitive relationship since older cars have typically been driven more.
- **Duration and First_registration**:These two variables have a moderate negative correlation of -0.459295, indicating that as the duration of ownership increases, the car tends to be newer (i.e., has a later first registration date). This might imply that people tend to keep newer cars for longer periods.

Our analysis of the correlation matrix reveals two key variables that significantly influence the monthly fee, namely horsepower and leasing duration. Horsepower shows a strong positive correlation, suggesting that more powerfull cars generally incur higher monthly fees. Conversely, duration of ownership has a negative correlation with the monthly fee, indication that longer leasing contract durations result in lower monthly fees. Other variables, including mileage, first_registration, emission_value, and consumption, show a weaker correlation with the monthly_fee, suggesting a lesser direct impact on this target variable.

As expected, horsepower and kilowatts show a perfect correlation, because they are different units of the same attribute of a car, the engine power. To prevent multicollinearity, which can complicate interpretation of the model, we decided to exclude kilowatts from our subsequent models. This decision helps to streamline our model by eliminating redundant information, focusing on the most relevant predictors for the monthly fee.

5.5.1 Dropping kilowatts As anticipated, there is a perfect 1:1 correlation between kilowatts (kW) and horsepower (HP), where the relationship is defined as P[kW] = 0.7457 * P[HP]. Given this direct correlation, we decided to exclude the kilowatts feature from the dataset when constructing our models.

0.8 6. Preprocessing and Feature Engineering

This section focuses on the preprocessing and feature engineering steps conducted in our project. These steps play a pivotal role in preparing the data and optimizing its suitability for the machine learning process. Our first task is to split the data into out-of-sample, test, and train sets. This division ensures robust model evaluation and guards against overfitting, enabling us to gauge the generalization performance accurately.

Once the data is appropriately partitioned, we employ the renowned scikit-learn library to define transformer pipelines. These pipelines serve as systematic frameworks for data transformation and feature engineering, ensuring consistency and efficiency throughout the machine learning workflow. Leveraging transformer pipelines allows us to seamlessly apply a range of preprocessing techniques, including scaling, categorical variable encoding, handling missing values, and creating interaction features.

By meticulously designing and implementing these preprocessing and feature engineering steps, we enhance the data's quality and representation. This enhancement empowers our machine learning models to capture meaningful patterns and make accurate predictions. The flexibility and comprehensiveness of scikit-learn's transformer pipelines provide us with a robust framework for streamlining these essential data preparation tasks. Moreover, this approach promotes reproducibility and scalability, facilitating future analysis and experimentation.

0.8.1 6.1 Missing Values

Within our dataset, two features, namely emissions and consumptions, contain missing values. To address this issue, we will incorporate imputing algorithms from the widely-used scikit-learn library into our transformer pipeline. These algorithms are specifically designed to handle missing values effectively, allowing us to impute or fill in the missing data points. By integrating these imputing algorithms into our pipeline, we ensure that the missing values in the emissions and consumptions features are appropriately addressed, thereby maintaining the integrity and completeness of our dataset throughout the machine learning process.

```
Summary of Missing Values:
registration      0
milage            0
duration          0
fee               0
emission        612
consumption     612
horsepower        0
brand             0
model             0
gear              0
fuel              0
dtype: int64
```

0.8.2 6.2 Cardinality of non-numeric features

Analyzing the cardinality helps us understand the number of distinct categories within each feature. In this case, the "Brand" feature consists of 20 unique brands, indicating a moderate level of variation. On the other hand, the "Model" feature has a higher cardinality with 346 unique models, suggesting a more diverse range of vehicle variations.

The "Gear" feature has only two categories, indicating a binary classification of the transmission type (e.g., manual vs. automatic). Similarly, the "Fuel" feature has three categories representing different fuel types (e.g., gasoline, diesel, hybrid).

Understanding the cardinality of categorical features is important for various aspects of data analysis, including feature selection, encoding strategies, and model interpretation. High cardinality may require careful handling to avoid overfitting or computational challenges, while low cardinality features can simplify modeling and analysis tasks. The cardinality of the model feature was a major concern in encoding categorical features. The high cardinality of the "model" feature lead to high dimensionality of the dataframe and the models.

```
brand     20
```

```
model    346
gear       2
fuel       3
dtype: int64
```

0.8.3 6.3 Problems with splitting

During the dataset splitting process using the "train_test_split" function, stratification is required to ensure that One Hot encoding works properly. However, a challenge arises when dealing with entries in the "model" column that appear only once or twice. Stratification cannot be applied to these unique or minimally occurring entries.

To address this issue, we have identified three possible approaches. The first approach involves dropping the single or double occurrence entries of the "model" column. This reduces the complexity introduced by the limited occurrences during stratification. Alternatively, the second approach suggests duplicating or tripling the once or twice occurring entries to increase their representation in the dataset. This approach helps maintain balance and avoids losing potentially valuable information. Another option is to create a combined category for these specific models, treating them as a separate group during the splitting process. This approach can preserve the uniqueness of these entries while ensuring proper stratification.

For the current implementation, we have decided to drop the entries that appear only once or twice. However, we acknowledge that other approaches may be explored in the future to fully utilize the data from these unique or minimally occurring entries.

```
Models dropped: 19
```

0.8.4 6.4 Out of Sample split

In the Data Science industry, it is considered a best practice to initially divide the dataset into two distinct subsets. The first subset is used for training and testing the models, while the second subset remains unseen until the models are evaluated in real-world scenarios. This remaining subset is commonly referred to as the out-of-sample dataset. By segregating the data in this manner, we adhere to industry standards and ensure that our models are rigorously tested and validated on unseen data, enhancing their ability to generalize and perform well in real-world applications.

For this project we split our data into 85% sample data and 15% out of sample data.

```
Size of the sample data: (16183, 11) with a mean of:  593.0066569857258
Size of out of sample data: (2856, 11) with a mean of:  592.3941316526611
```

0.8.5 6.5 Train and Test slpit

Subsequently, the remaining sample data is divided into two subsets: the training data and the test data. In the context of this project, the sample dataset was split with a ratio of 75% for the training data and 25% for the test data.

```
Size of the train data: (12137, 10) with a mean fee of: 593.1457435939689
Size of the test data: (4046, 10) with a mean fee of: 592.5894315373208
```

0.8.6 6.6 Transformer Pipelines

Transformer pipelines in scikit-learn are a powerful tool for automating data preprocessing and feature engineering tasks in machine learning. These pipelines allow for a seamless flow of data transformations, ensuring consistency and efficiency in the preprocessing process. By encapsulating a series of transformers within a pipeline, we can easily apply the same preprocessing steps to both training and test data, promoting reproducibility and scalability. With a wide range of transformer classes available, scikit-learn provides a flexible framework to handle diverse data types and preprocessing requirements. Leveraging transformer pipelines streamlines the data preparation process and enhances the performance of machine learning models.

6.6.1 Challenge: Encoding When it comes to choosing between OneHot Encoder, Label Encoder, and Ordinal Encoder, the decision depends on the specific requirements of the machine learning models being used. Let's evaluate the applicability of these encoders for different models:

Decision Tree and Random Forest: Decision trees and random forests can handle both categorical and numerical features effectively. They are not influenced by the encoding technique used, making them compatible with all three encoders. OneHot Encoder is suitable for decision trees and random forests as it can represent categorical variables without imposing an ordinal relationship. Label Encoder and Ordinal Encoder can also be used, but they might introduce an implicit order that may or may not be appropriate for the model.

XGBoost and AdaBoost:XGBoost and AdaBoost are ensemble learning methods that uses gradient boosting. Similar to decision trees and random forests, they can handle both categorical and numerical features. OneHot Encoder, Label Encoder, and Ordinal Encoder can be used with both models. OneHot Encoder may result in high dimensionality, but XGBoost's ability to handle sparse data makes it feasible. However, considering the potential memory and computational limitations, careful consideration should be given to the choice of encoding.

KNN (K-Nearest Neighbors):KNN is a distance-based algorithm that calculates similarity between data points. OneHot Encoder is not suitable for KNN as it can lead to the curse of dimensionality due to the high dimensionality introduced. Label Encoder and Ordinal Encoder can be used for KNN, but they assume an underlying order or ranking that may not be appropriate for categorical features. Thus, it is advisable to use alternative encoding techniques such as target-based encoding or frequency encoding to preserve the categorical information while mitigating the dimensionality issue.

SVR (Support Vector Regression):SVR is a regression method that uses support vectors to find the best fit. Similar to KNN, OneHot Encoder can result in high dimensionality and is not recommended for SVR. Label Encoder and Ordinal Encoder can be used with SVR, but they assume an ordinal relationship that may not be valid for categorical features. Alternative encoding methods such as target encoding or effect encoding might be more suitable for SVR to capture the impact of categorical features accurately.

In summary, the choice of encoder depends on the specific machine learning models used. OneHot Encoder is generally suitable for decision trees, random forests, and XGBoost, while Label Encoder and Ordinal Encoder may introduce implicit ordering that can be inappropriate for some models. KNN and SVR require careful consideration of encoding techniques to address high dimensionality and preserve the meaningful information within categorical features.

Sklearn's label encoder is used for the target variable, not for feature variables. Ordinal encoding is

supposed to be used on categorical feature variables. Ordinal encoding is easier to use than writing a custom encoder using label encoding. Ordinal implies an underlying rank of values, although there might be no real underlying ranking. Ordinal encoding ranks alphabetically, which might not make sense.

SHAP values for OneHot encoded models can be aggregated by summing the individual SHAP values, according to: https://github.com/slundberg/shap/issues/397

We trained all models with OneHot encoding and with ordinal encoding and recognized, that there is little difference in most of their prediction performances. Even KNN, which could be effected by the introduced ranking of the ordinal encoder, showed similar performance to the KNN model with OneHot encoding. As expected, the ordinal encoding influenced the SVR model's performance immensely. We assume that this is due to the introduced ranking in the ordinal encoding. We additionally introduced AdaBoost to replace the SVR model for Ordinal encoding, because the SVR model was very much effected by the differences in encoding.

As expected, computation times were greatly reduced with the introduction of ordinal encoding.

For evaluation of the differences of OneHot encoding and Ordinal encoding, refer to the Appendix

```
The data is being encoded using ordinal encoding.
```

The complete preprocessing Columntransformer is presented in the following plot.

```
ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                 ('scaler', StandardScaler())]),
                                 Index(['registration', 'milage', 'duration',
'emission', 'consumption',
       'horsepower'],
      dtype='object')),
                                ('cat',
                                 Pipeline(steps=[('imputer',
SimpleImputer(fill_value='missing',
strategy='constant')),
                                                 ('ordinal',
                                                  OrdinalEncoder())]),
                                 Index(['brand', 'model', 'gear', 'fuel'],
dtype='object'))],
                  verbose_feature_names_out=False)
```

0.9 7 Machine Learning Modeling

0.9.1 7.1 Choosing appropriate metric and customization approach

Choosing the appropriate evaluation metrics is a critical step in assessing the performance of machine learning models. These metrics help us understand how well our models are performing and compare different models or algorithms against each other. In the context of our project, where we are predicting monthly leasing prices, we have selected several evaluation metrics to evaluate the quality of our models.

The Mean Squared Error (MSE) is a widely used metric that calculates the average squared difference between the predicted and actual values. It provides a measure of how close our predictions are to the true values, with lower values indicating better performance. The Root Mean Squared Error (RMSE) is derived from MSE by taking the square root of the average squared difference, which provides a more interpretable metric in the original scale of the target variable.

The Mean Absolute Error (MAE) is another commonly used metric that calculates the average absolute difference between the predicted and actual values. Like MSE, lower values of MAE indicate better model performance. MAE is less sensitive to outliers compared to MSE, making it a suitable choice when extreme values are present in the data.

The R-squared (R2) metric measures the proportion of variance in the target variable that is explained by the model. It ranges between 0 and 1, with higher values indicating a better fit. R2 is a valuable metric for assessing the overall goodness-of-fit of the model.

Additionally, we have chosen the Mean Absolute Percentage Error (MAPE or MAPR) as an evaluation metric. MAPE calculates the average percentage difference between the predicted and actual values, providing insights into the relative magnitude of errors. MAPE or MAPR is useful when we want to understand the accuracy of our predictions in relation to the actual values.

By utilizing these evaluation metrics, we can comprehensively evaluate the performance of our models and gain insights into their accuracy, precision, and generalization capabilities. This enables us to make informed decisions regarding model selection and fine-tuning to improve the predictive capabilities of our system.

For evaluation on the train and test set, we chose the following measurements: - MSE: Mean-Square-Error - RMSE: Root-Mean-Square-Error (for easier interpretation) - MAE: Mean-Absolute-Error - R-squared (R^2) - MAPR: Mean Absolute Percentage Residual(Error)

Using a scoring dictionary, the RandomSearch algorithm calculates all scoring values, although it only refits on the given "refit" parameter, for which we used MSE. We also tried MAE and MAPR, but this did not make a noteable difference.

0.9.2 7.2 Decision Tree

In this section, we focus on building a decision tree model for predicting vehicle leasing prices. Decision trees are powerful machine learning algorithms that can effectively handle both numerical and categorical data. They provide interpretable models that mimic the decision-making process, making them widely used and easily understandable. To construct the decision tree model, we define a parameter distribution that includes various hyperparameters such as 'min_samples_split', 'min_samples_leaf', 'ccp_alpha', and 'random_state'. These hyperparameters control the behavior and complexity of the decision tree and need to be optimized to achieve the best performance. We create a pipeline that includes a preprocessor for data transformation and a DecisionTreeRegressor as the main model. The preprocessor ensures that the data is properly prepared before being fed into the decision tree model. To find the optimal combination of hyperparameters, we perform a randomized search with cross-validation using RandomizedSearchCV. This technique allows us to efficiently explore different hyperparameter settings and evaluate their impact on model performance.

After fitting the decision tree model to the training data, we evaluate its performance using various metrics on both the training and test sets. These metrics provide insights into the model's accuracy,

precision, and generalization capabilities.

Additionally, we analyze the best hyperparameter values obtained from the randomized search and present them in a DataFrame. This information helps us understand the configuration of the decision tree model that yielded the best results.

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                                  ('scaler',
StandardScaler())]),
                                                  Index(['registration',
'milage', 'duration', 'emission', 'consumption',
       'horsepower'],
      dtype='object')),
                                                 ('cat',
                                                  Pipeline(steps=[('imputer',
SimpleImputer(fill_value='missing',
 strategy='constant')),
                                                                  ('ordinal',
OrdinalEncoder())]),
                                                  Index(['brand', 'model',
'gear', 'fuel'], dtype='object'))],
                                    verbose_feature_names_out=False)),
                ('regressor', DecisionTreeRegressor())])
```

The evaluation metrics of the Decision Tree model are shown in the following table.

```
Evaluation Metrics:
      Decision Tree Train  Decision Tree Test
MSE             46.588577           93.093921
RMSE             6.825583            9.648519
MAE              2.636798            3.217996
R2               0.999496            0.998996
MAPR             0.005080            0.006203
```

This model is defined by the following hyperparameters.

```
{'ccp_alpha': 0.05648616489184735, 'criterion': 'squared_error', 'max_depth':
None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease':
0.0, 'min_samples_leaf': 2, 'min_samples_split': 20, 'min_weight_fraction_leaf':
0.0, 'random_state': 2023, 'splitter': 'best'}
```

0.9.3 7.3 Random Forest

In this section, our main focus is on constructing a random forest model to predict vehicle leasing prices. Random forest is an ensemble learning algorithm that combines multiple decision trees to make accurate predictions. It is highly regarded for its robustness, capability to handle complex

data, and resilience against overfitting.

To establish the random forest model, we define a parameter distribution encompassing key hyperparameters such as 'n_estimators', 'max_depth', 'min_samples_split', 'min_samples_leaf', and 'random_state'. These hyperparameters play a crucial role in controlling the behavior and complexity of the random forest model, allowing us to fine-tune its performance. Also for this model, a pipeline including the preprocessor and the Random Forest Regressor was used.

To identify the optimal combination of hyperparameters, we conduct a randomized search with cross-validation using RandomizedSearchCV. This approach enables us to systematically explore various hyperparameter settings and assess their impact on the model's performance and predictive capabilities.

Following the fitting of the random forest model to the training data, we proceed to evaluate its performance using a range of metrics on both the training and test sets. This comprehensive evaluation enables us to assess the accuracy of the model's predictions and its ability to generalize well to unseen data. By analyzing these metrics, we can gain valuable insights into the model's overall effectiveness and make informed decisions about its deployment and suitability for real-world applications.

The evaluation metrics of the Random Forest model are shown in the following table.

Evaluation Metrics:

	Random Forest Train	Random Forest Test
MSE	42.209474	74.309919
RMSE	6.496882	8.620320
MAE	2.325318	3.005066
R2	0.999543	0.999199
MAPR	0.004366	0.005742

This model is defined by the following hyperparameters.

```
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'squared_error', 'max_depth':
20, 'max_features': 1.0, 'max_leaf_nodes': None, 'max_samples': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 2, 'min_samples_split': 10,
'min_weight_fraction_leaf': 0.0, 'n_estimators': 550, 'n_jobs': None,
'oob_score': False, 'random_state': 2023, 'verbose': 0, 'warm_start': False}
```

0.9.4 7.4 K-nearest neighbor

In this section, our focus is on constructing a K-Nearest Neighbors (KNN) model for the prediction of vehicle leasing prices. KNN is a non-parametric algorithm that utilizes the nearest neighbors from the training data to make predictions, making it highly adaptable to various data types. To develop the KNN model, we again establish a dedicated pipeline that incorporates a preprocessor responsible for data transformation, along with the KNeighborsRegressor serving as the primary model.

To determine the most optimal hyperparameter configuration, we employ a randomized search in conjunction with cross-validation, utilizing RandomizedSearchCV. The hyperparameters considered in this process include 'n_neighbors', 'leaf_size', 'weights', and 'p'. These hyperparameters control

the number of neighbors, the leaf size of the tree, the weight function employed in predictions, and the distance metric used, respectively.

The evaluation metrics of the KNN model are shown in the following table.

```
Evaluation Metrics:
      KNN Train    KNN Test
MSE   11.761946  336.703067
RMSE   3.429569   18.349470
MAE    0.702970    8.051065
R2     0.999873    0.996370
MAPR   0.001378    0.014772
```

This model is defined by the following hyperparameters.

```
{'algorithm': 'auto', 'leaf_size': 82, 'metric': 'minkowski', 'metric_params':
None, 'n_jobs': None, 'n_neighbors': 6, 'p': 1, 'weights': 'distance'}
```

Despite the fact that KNN should theoretically consider the introduced ranking due to its reliance on distance-based neighbor grouping, it exhibited similar performance to the OneHot encoded KNN model (See A1 Encoding differences). This suggests that the impact of the ranking on the KNN algorithm's predictions may be minimal in this particular context.

0.9.5 7.5 XGBoost

In this section, we focus on building a regression model using XGBoost (Extreme Gradient Boosting). XGBoost is a powerful machine learning algorithm known for its exceptional performance in various domains. It is an ensemble learning method that combines multiple decision trees to make accurate predictions. XGBoost incorporates gradient boosting techniques and introduces additional regularization to enhance model generalization and handle complex data patterns effectively.

To build the XGBoost model, we utilize a RandomizedSearchCV approach to search for the optimal combination of hyperparameters. These hyperparameters include the maximum depth of the trees, learning rate, number of estimators, gamma, subsample, colsample_bytree, min_child_weight, reg_lambda, reg_alpha, tree_method, and random_state. By performing cross-validation during the search, we ensure robust model evaluation and selection of hyperparameters that yield the best performance.

In addition XGBoost offers **GPU acceleration**, which increases computational performance and reduced model building time.

The evaluation metrics of the XGB model are shown in the following table.

```
Evaluation Metrics:
      XGB Train   XGB Test
MSE   31.763895  67.994437
RMSE   5.635947   8.245874
MAE    2.776669   3.727759
R2     0.999656   0.999267
MAPR   0.005346   0.007202
```

This model is defined by the following hyperparameters.

```
{'objective': 'reg:squarederror', 'base_score': None, 'booster': None,
'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': 0.75,
'eval_metric': None, 'gamma': 0.35, 'gpu_id': None, 'grow_policy': None,
'interaction_constraints': None, 'learning_rate': 0.06, 'max_bin': None,
'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None,
'max_depth': 50, 'max_leaves': None, 'min_child_weight': 2,
'monotone_constraints': None, 'n_jobs': None, 'num_parallel_tree': None,
'predictor': None, 'random_state': None, 'reg_alpha': 30, 'reg_lambda': 0.01,
'sampling_method': None, 'scale_pos_weight': None, 'subsample': 0.7,
'tree_method': 'gpu_hist', 'validate_parameters': None, 'verbosity': None}
```

0.9.6 7.6 Support-Vector-Machine (SVM)

In this section, we focus on building a regression model using Support Vector Machines (SVM). SVM is a powerful algorithm that is widely used for regression tasks due to its ability to handle both linear and non-linear relationships in the data.

The SVM model is constructed using a pipeline that incorporates a preprocessor and an SVR (Support Vector Regression) regressor. The preprocessor handles data preprocessing steps, such as feature scaling and encoding, to ensure compatibility with the SVM model.

To find the optimal hyperparameters for the SVM model, we utilize RandomizedSearchCV. This technique performs a randomized search over a specified parameter distribution, allowing us to explore different combinations of hyperparameters efficiently. The hyperparameters we tune include 'C', which controls the regularization strength, 'kernel' for the choice of kernel function, and 'epsilon' that sets the margin of error allowed in the model.

SVM/SVR is a theoretically simple algorithm, but its computational complexity makes it a time-consuming process, often taking several hours to build. The time complexity of SVM is typically in the range of O(n^2) to O(n^3), where n is the number of training samples.

The 'linear' kernel of SVR has could potentially outperfrom the 'rbf' and 'poly' kernels, however, due to its extreme computational inefficiency, we were not able to finish building a model this way. We then removed 'linear' from the kernel parameter list, which reduced computation times from +10 hours to a mere 8 minutes.

Additionally, SVR is sensitive to the introduced ranking of categorical variables through Ordinal encoding, which resulted in very poor performance.

As expected, the SVM Regression is very much effected by the ordinal encoding. The results differ a lot compared to the OneHot encoding. See Encoding differences. The metrics of the SVM Regrssion is shown in the following table.

```
Evaluation Metrics:
          SVR Train      SVR Test
MSE   49434.884550  49742.882297
RMSE    222.339570    223.031124
MAE     131.292027    131.776217
```

```
R2        0.465137      0.463720
MAPR      0.206133      0.206692
```

This model is defined by the following hyperparameters.

```
{'C': 200, 'cache_size': 200, 'coef0': 0.0, 'degree': 3, 'epsilon': 0.001,
'gamma': 'scale', 'kernel': 'rbf', 'max_iter': -1, 'shrinking': True, 'tol':
0.001, 'verbose': False}
```

0.9.7 7.7 AdaBoost Regressor

In the section dedicated to building the AdaBoost Regressor, we employ the AdaBoost algorithm to train a regression model. AdaBoost, short for Adaptive Boosting, is an ensemble learning method renowned for its ability to combine multiple weak learners into a robust and accurate predictor.

To begin, we define a pipeline that comprises a preprocessor and the AdaBoostRegressor. The AdaBoostRegressor serves as the core component, implementing the boosting algorithm.

To optimize the AdaBoostRegressor, we specify a parameter distribution encompassing various hyperparameters. These include the number of estimators, learning rate, loss function, base estimator (e.g., DecisionTreeRegressor or RandomForestRegressor), and random state. These hyperparameters influence the performance and behavior of the AdaBoostRegressor model.

To explore the optimal hyperparameter configuration, we conduct randomized search cross-validation using RandomizedSearchCV. This approach enables us to efficiently sample different combinations of hyperparameters and evaluate their impact on the model's performance.

Furthermore, we introduce the AdaBoost Regressor into our analysis due to the underperformance of the SVM Regression model. By incorporating AdaBoost, we aim to improve the predictive capability and overall performance of our regression model.

The evaluation metrics of the AdaBoost Regressor are shown in the following table.

```
Evaluation Metrics:
      AdaBoost Train  AdaBoost Test
MSE        32.013453      57.512628
RMSE        5.658043       7.583708
MAE         1.869100       2.461741
R2          0.999654       0.999380
MAPR        0.003630       0.004807
```

This model is defined by the following hyperparameters.

```
{'base_estimator': 'deprecated', 'estimator__ccp_alpha': 0.0,
'estimator__criterion': 'squared_error', 'estimator__max_depth': 15,
'estimator__max_features': None, 'estimator__max_leaf_nodes': None,
'estimator__min_impurity_decrease': 0.0, 'estimator__min_samples_leaf': 1,
'estimator__min_samples_split': 2, 'estimator__min_weight_fraction_leaf': 0.0,
'estimator__random_state': None, 'estimator__splitter': 'best', 'estimator':
DecisionTreeRegressor(max_depth=15), 'learning_rate': 0.1, 'loss':
'exponential', 'n_estimators': 100, 'random_state': 2023}
```

0.10 8 Test Data Performance

The evaluation on the test dataset is a critical step in assessing the performance and generalization of our machine learning models. It serves a dual purpose as we utilize the test dataset to evaluate the models' performance and select the optimal hyperparameters. By evaluating the models on the test data, we identify the hyperparameter configurations that yield the best results. This evaluation is conducted using various evaluation metrics and visualizations, enabling us to determine the most effective and reliable model for our specific application.

0.10.1 8.1 Metrics comparison

The tables presented below showcase the performance metrics of all models on both the train and test datasets.

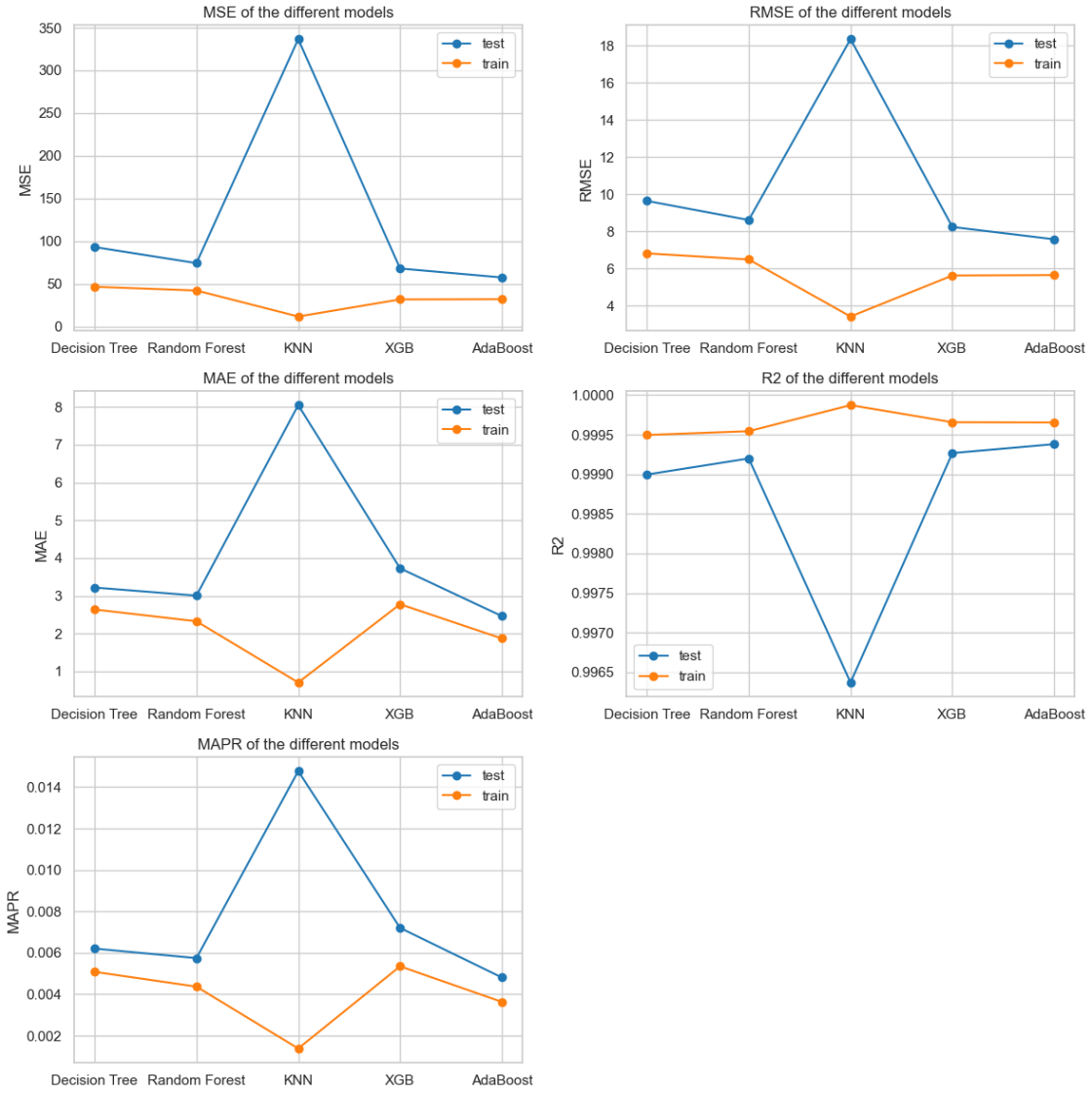
	Decision Tree Train	Random Forest Train	KNN Train	XGB Train	\
MSE	46.588577	42.209474	11.761946	31.763895	\
RMSE	6.825583	6.496882	3.429569	5.635947	
MAE	2.636798	2.325318	0.702970	2.776669	
R2	0.999496	0.999543	0.999873	0.999656	
MAPR	0.005080	0.004366	0.001378	0.005346	

	AdaBoost Train	SVR Train
MSE	32.013453	49434.884550
RMSE	5.658043	222.339570
MAE	1.869100	131.292027
R2	0.999654	0.465137
MAPR	0.003630	0.206133

	Decision Tree Test	Random Forest Test	KNN Test	XGB Test	\
MSE	93.093921	74.309919	336.703067	67.994437	\
RMSE	9.648519	8.620320	18.349470	8.245874	
MAE	3.217996	3.005066	8.051065	3.727759	
R2	0.998996	0.999199	0.996370	0.999267	
MAPR	0.006203	0.005742	0.014772	0.007202	

	AdaBoost Test	SVR Test
MSE	57.512628	49742.882297
RMSE	7.583708	223.031124
MAE	2.461741	131.776217
R2	0.999380	0.463720
MAPR	0.004807	0.206692

To facilitate interpretation, the plots below provide visual representations of the models' performances. However, it is important to note that the SVM Regressor has been excluded from these plots due to its high errors.



0.10.2 8.2 Predicted vs actual plots

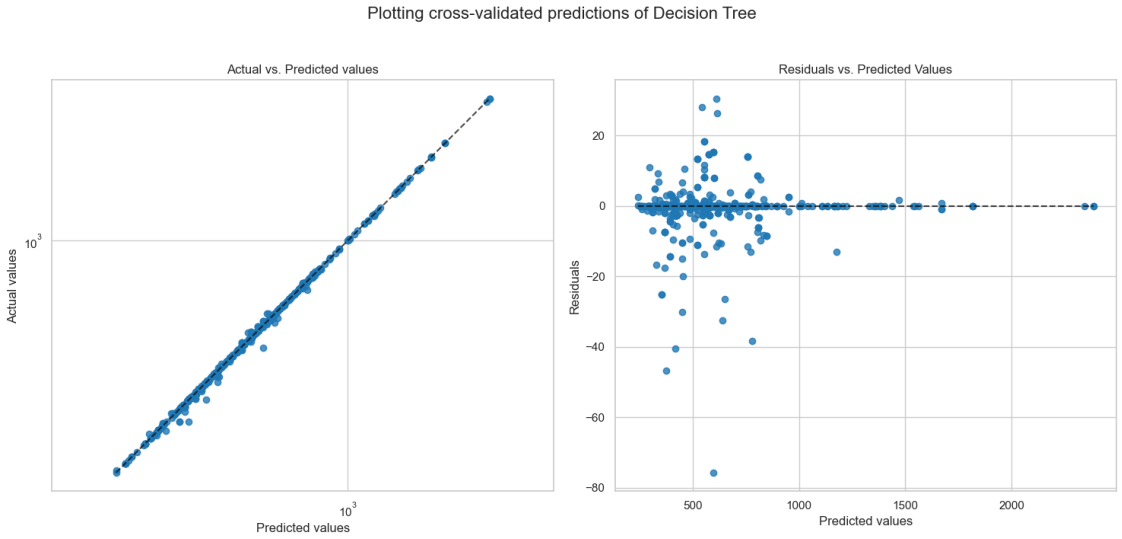
In the evaluation of machine learning models, two essential visualizations are the predicted vs actual plots and the residual vs predicted values plot. These plots provide valuable insights into the performance, accuracy, and limitations of the models.

The predicted vs actual plot illustrates the relationship between the predicted values and the corresponding actual values of the target variable. By examining the proximity of the data points to the diagonal line, we can assess the accuracy and consistency of the model's predictions. Deviations from the diagonal line indicate discrepancies between the predicted and actual values, guiding us in refining the model and identifying data patterns.

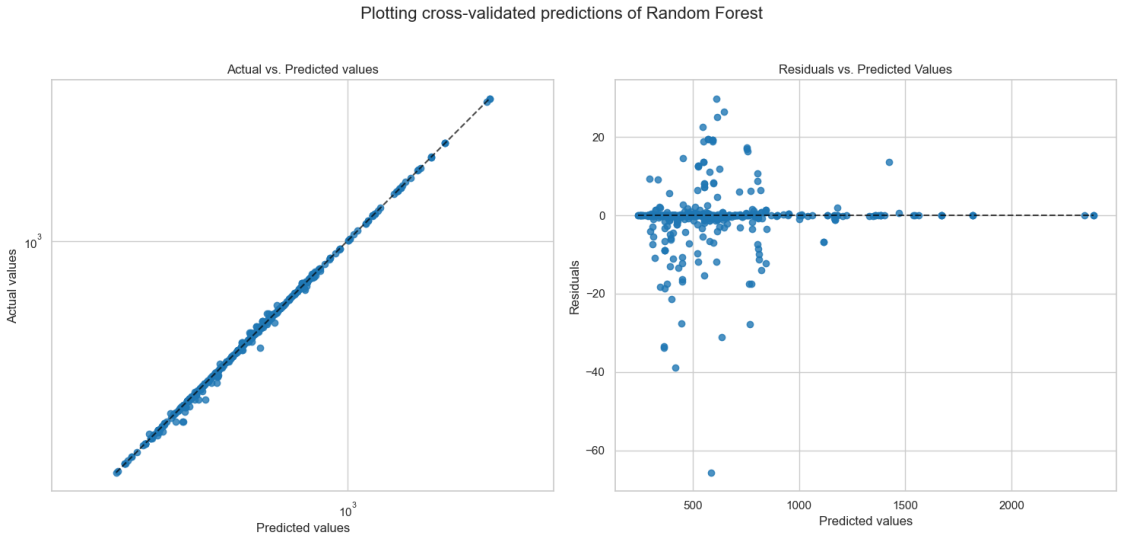
Similarly, the residual vs predicted values plot showcases the relationship between the model's predicted values and the residuals, which represent the differences between the actual and pre-

dicted values. This plot helps us understand the nature and distribution of errors made by the model. Examining the patterns in the residuals enables us to identify potential biases, non-linear relationships, or unequal variances in the data, further refining the model's predictions.

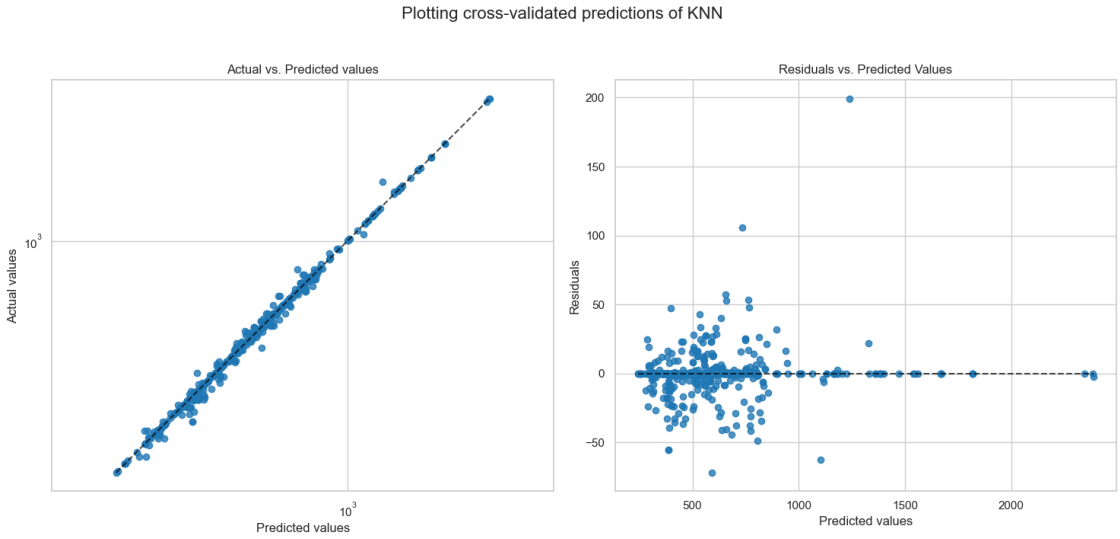
Together, these visualizations provide a comprehensive evaluation of the model's performance and uncover valuable insights for model refinement and improvement. By analyzing the predicted vs actual plot and the residual vs predicted values plot, we can assess the model's accuracy, detect systematic errors, and identify areas for further optimization in real-world applications.



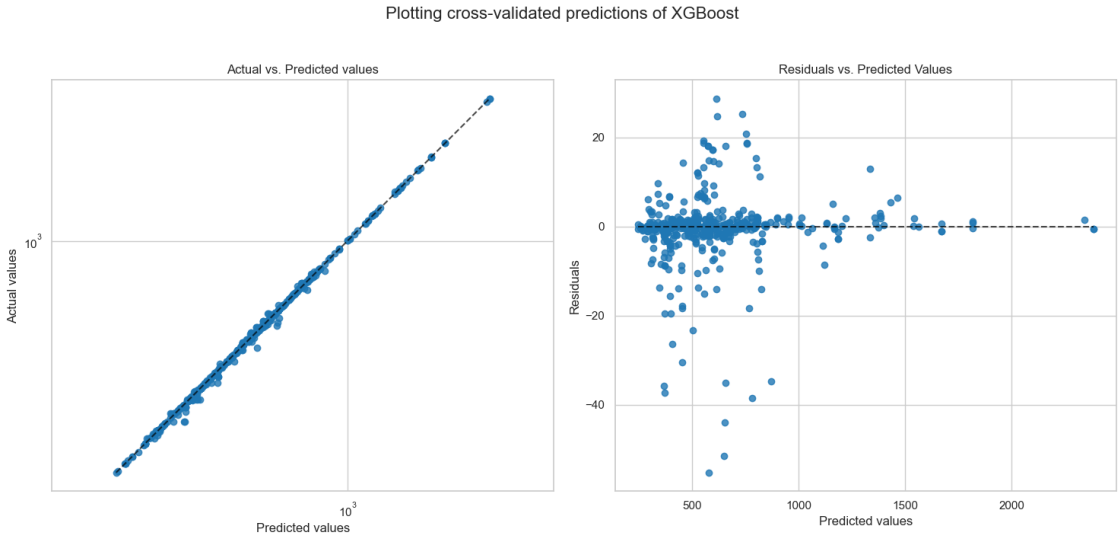
`The mean residual of Decision Tree is: 0.27872454235489874`



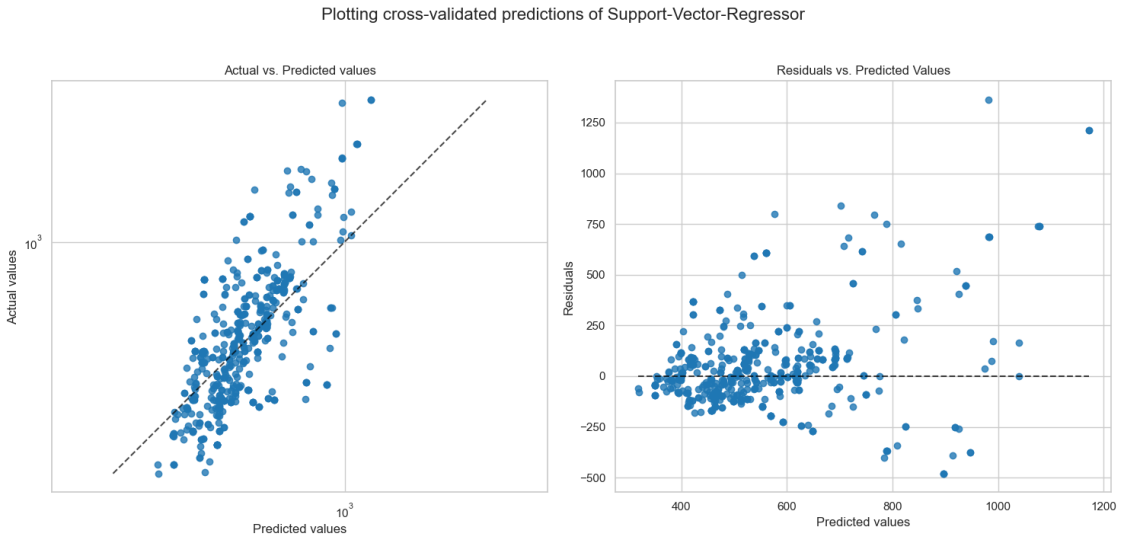
`The mean residual of Random Forest is: 0.2560250310969117`



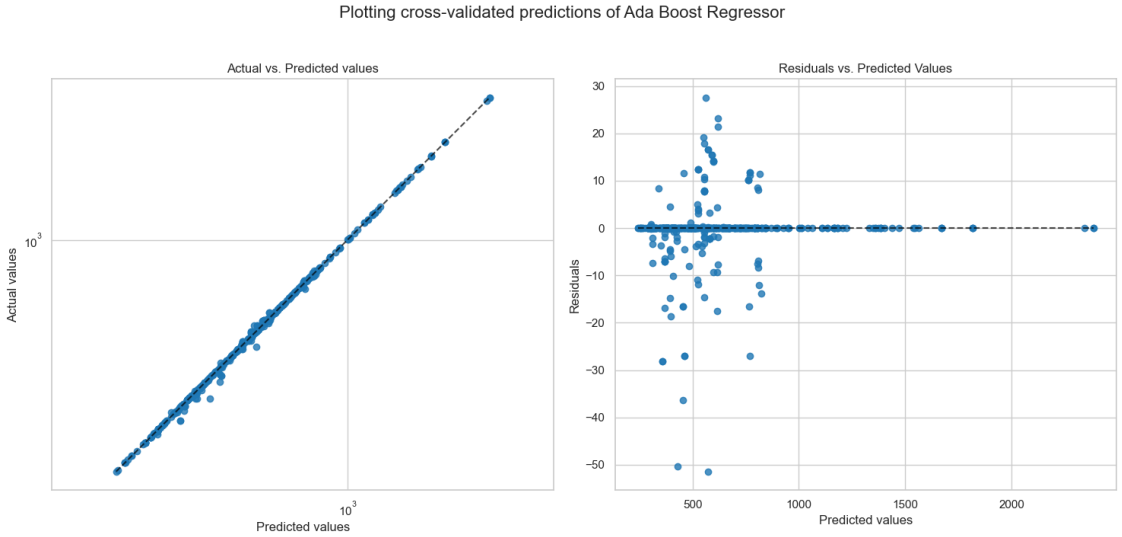

```
The mean residual of KNN is: 0.2874185173522171
```




```
The mean residual of XGBoost is: 0.3873319158775656
```




```
The mean residual of Support-Vector-Regressor is: 41.631615507053944
```




```
The mean residual of Ada Boost Regressor is: 0.1737747323158299
```

Plotting the cross validated predictions for the different models provide us with valuable insights. We can see that that the Decision Tree model shows high accuracy and performs well in predicting leasing rates. The model exhibits a good alignment between the predicted and actual values in the "Actual vs Predicted values" plot and the residuals are centered around the zero line in the "Residuals vs Predicted values" plot. The Random Forest model performs similarly well to the Decision Tree model and shows a strong alignment between predicted and actual values in the "Actual vs Predicted values" plot. The residuals are mainly evenly distributed around the zero line in the "Residuals vs Predicted values" plot, indicating unbiased predictions. The KNN model

shows higher errors in the predicted values vs the actual values compared to the Decision Tree and Random Forest models, so the alignment between predicted and actual values in the "Actual vs Predicted values" plot shows some deviation. The residuals exhibit a much more "visible" pattern in the "Residuals vs Predicted values" plot, suggesting that the model may not capture certain patterns in the data effectively. For the XGBOOST we can observe that The "Actual vs Predicted values" plot displays a strong alignment between the predicted and actual values, indicating a good fit and it actually shows the best distribution of the residuals from all the models. The SVR model performs relatively poorly compared to the other models. The alignment between predicted and actual values in the "Actual vs Predicted values" plot shows noticeable deviation, suggesting the model's limitations and the residuals in the "Residuals vs Predicted values" plot are concentrated at higher values, indicating a higher bias in the predictions. Last but not least, the Ada Boost Regressor shows good performance with relatively low errors.

0.11 9 Out of sample performance

In this section, we analyze the performance of our machine learning models on out-of-sample data. Out-of-sample data refers to data that was not used during the model training and evaluation process. Evaluating the models on out-of-sample data provides a more realistic assessment of their performance and helps us understand how well they can generalize to new, unseen instances.

By examining the performance metrics on the out-of-sample data, we can determine how well our models are likely to perform in real-world scenarios. This evaluation allows us to validate the models' effectiveness, identify any potential issues or limitations, and make informed decisions about their deployment.

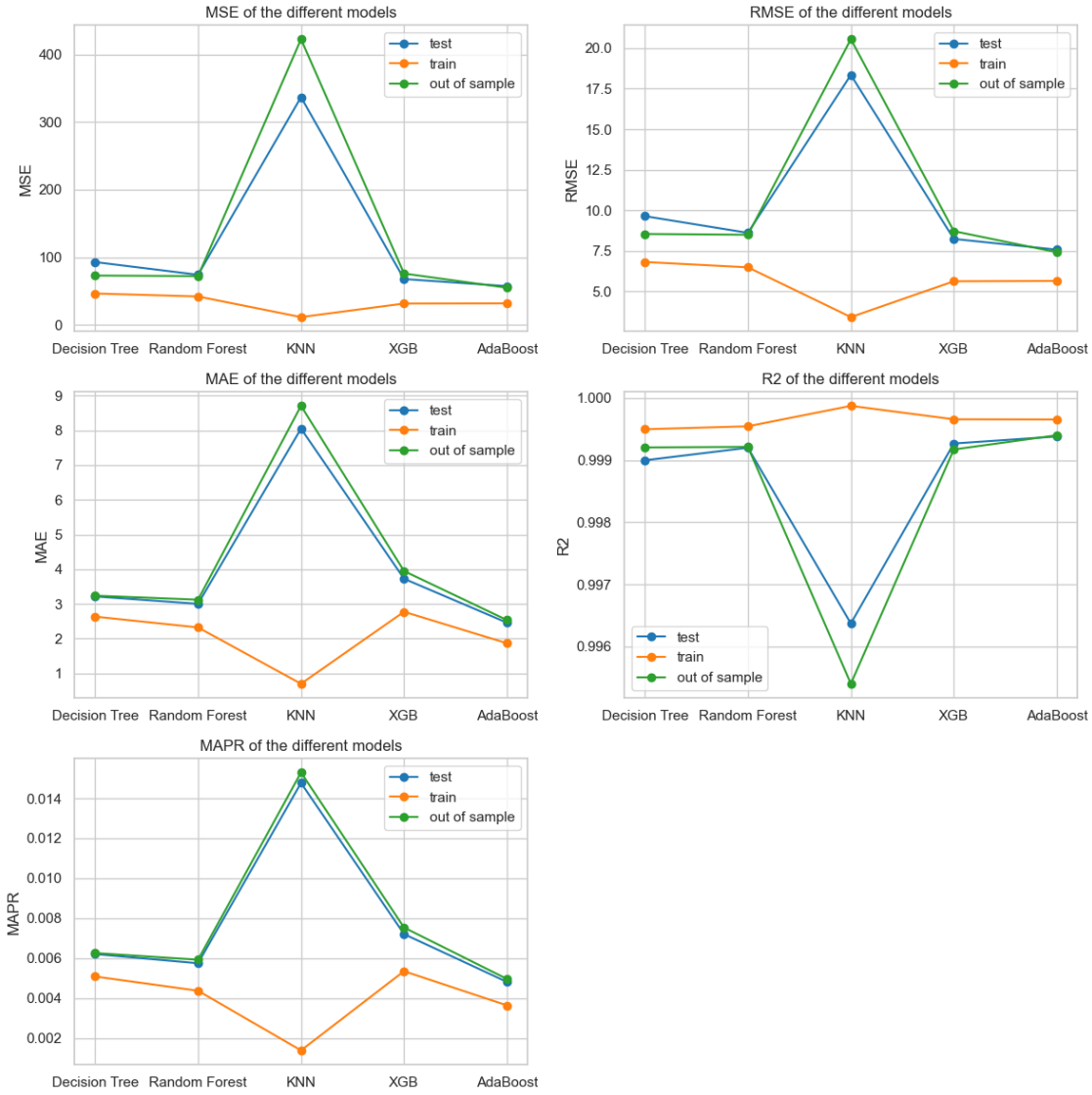
0.11.1 9.1 Metrics comparison

The table provided presents the performance of each model on the out-of-sample dataset, evaluated using various metrics.

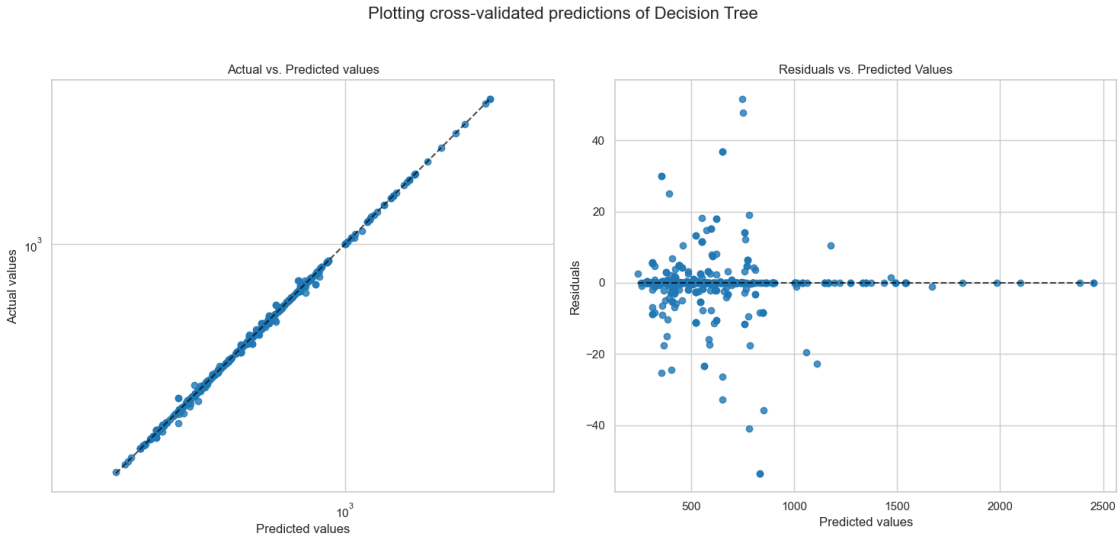
```
      Decision Tree out of sample  Random Forest out of sample
MSE                     73.118461                    72.311337  \
RMSE                     8.550933                     8.503607
MAE                      3.240827                     3.124188
R2                       0.999204                     0.999212
MAPR                     0.006254                     0.005918

      KNN out of sample  XGB out of sample  SVR out of sample
MSE          422.391913          76.123017       49388.193690  \
RMSE          20.552175           8.724851         222.234547
MAE            8.708540           3.950068         131.664068
R2             0.995400           0.999171           0.462136
MAPR           0.015295           0.007532           0.207219

      AdaBoost out of sample
MSE                55.101985
RMSE                7.423071
MAE                 2.536617
R2                  0.999400
```

```
MAPR                    0.004957
```

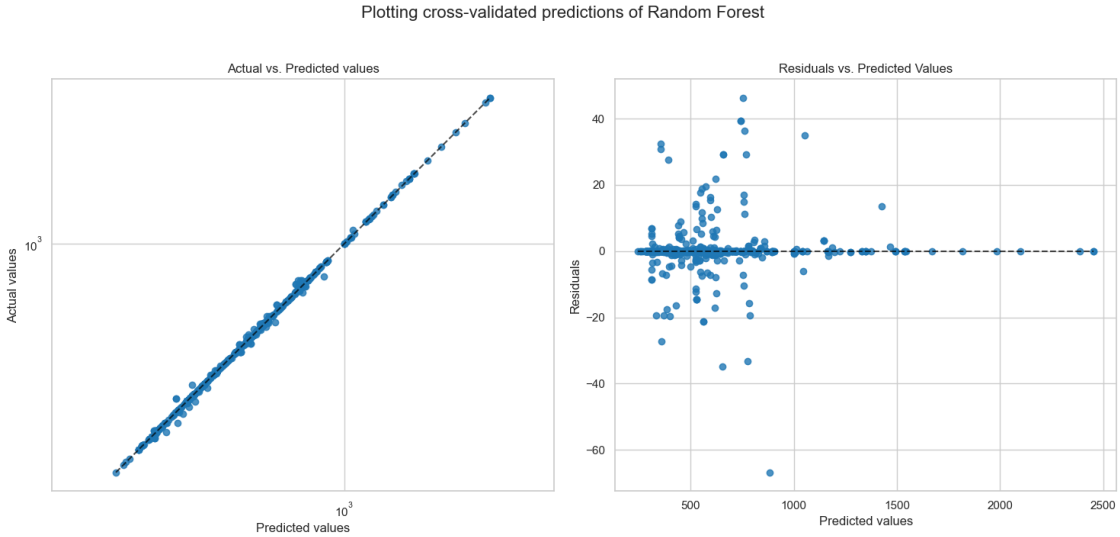
To facilitate interpretation, the plots below provide visual representations of the models' training, test and out of sample performance. However, it is important to note that the SVM Regressor has been excluded from these plots due to its high errors.



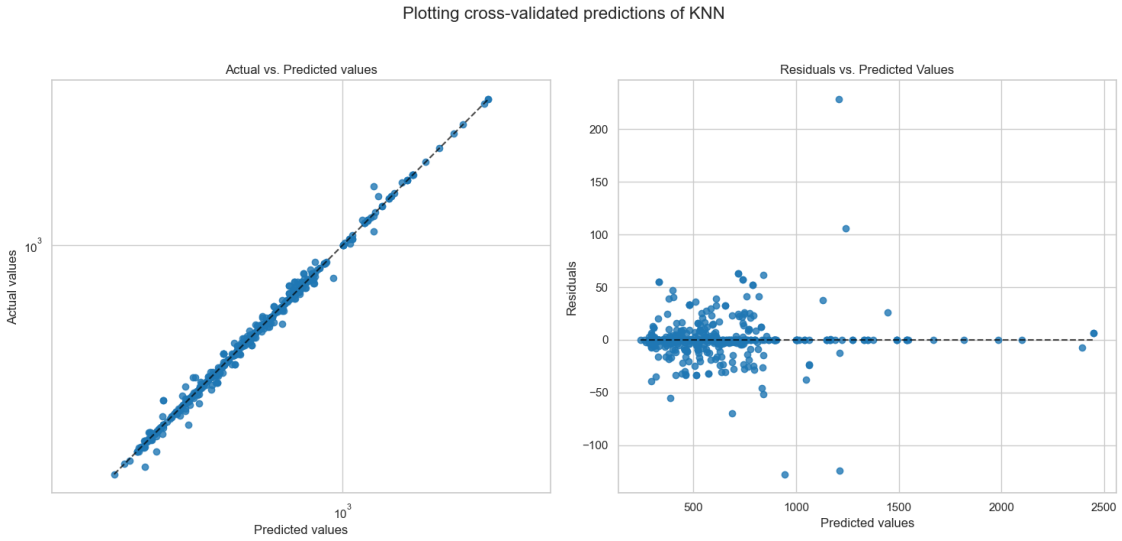
0.11.2 9.2 Predicted vs actual plots



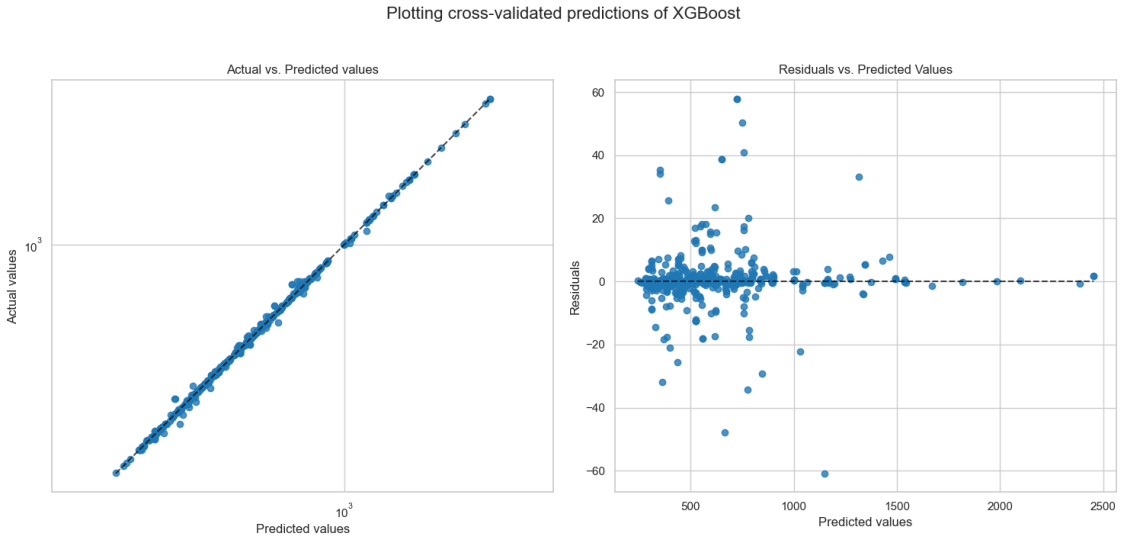
The mean residual of Decision Tree is: -0.22234999227525504



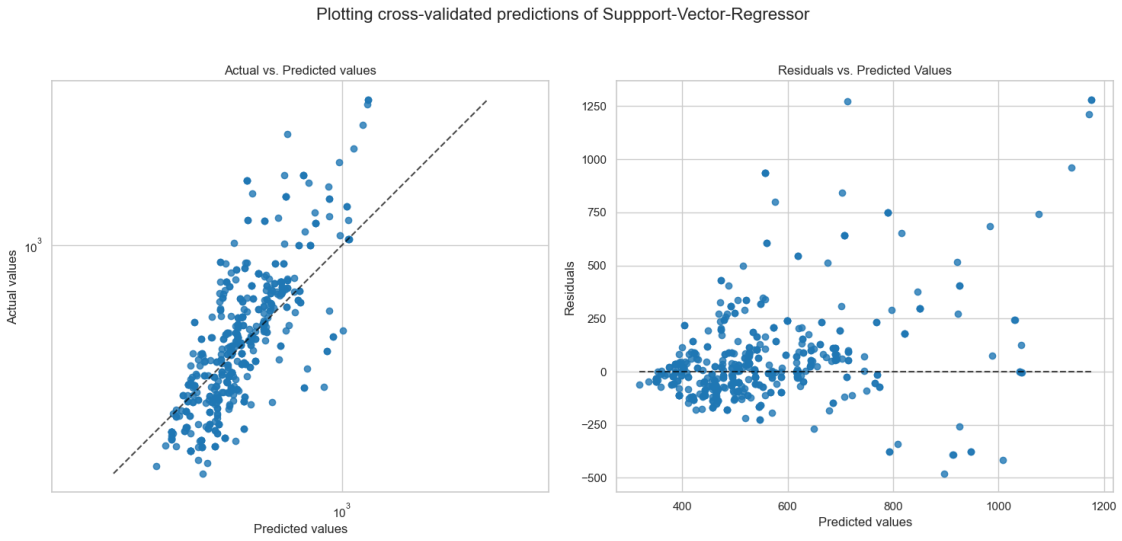
The mean residual of Random Forest is: -0.12240398551644305



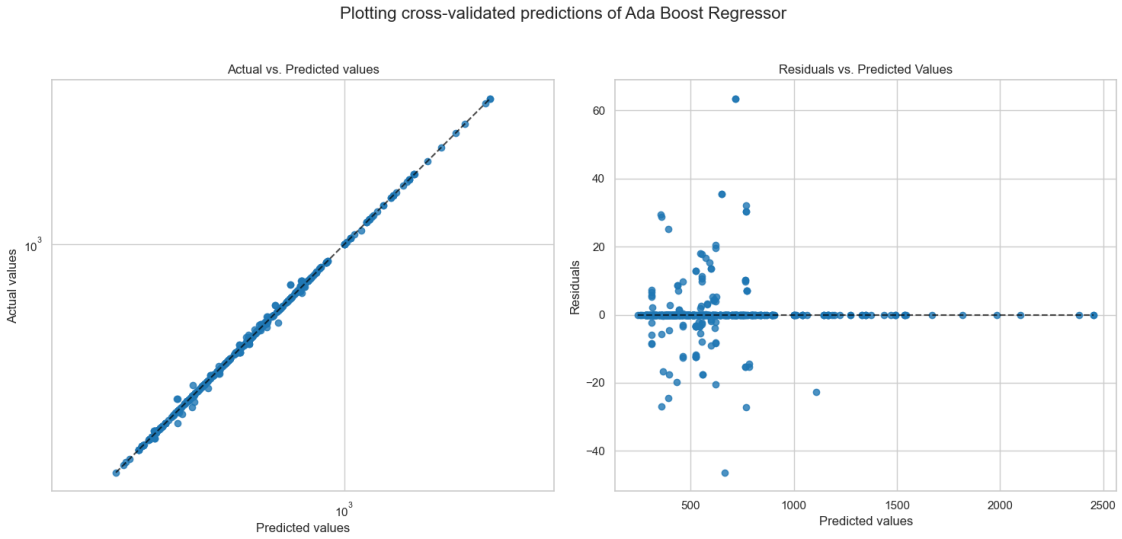

```
The mean residual of KNN is: -0.1064419259358378
```




```
The mean residual of XGBoost is: -0.04810206074006601
```



`The mean residual of Suppport-Vector-Regressor is: 40.993052932640516`



`The mean residual of Ada Boost Regressor is: -0.1042347929474167`

Plotting the cross validated predictions for the out of sample data also provides us with valuable insights. We can see that the actuval vs predicted values are extremely consistent for the ADA-Boost Regressor, the XGBOOST, the Random Forest and the Decision tree. The residuals for these models are all evenly distributed around the zero line in the "Residuals vs Predicted values" plot.

However judging by the plot we can see that the SVR has extremely inconsistent predicted vs actucal values.The residuals from this model in the "Residuals vs Predicted values" plot are also concentrated at much higher values compared to the other models, indicating a higher bias in the

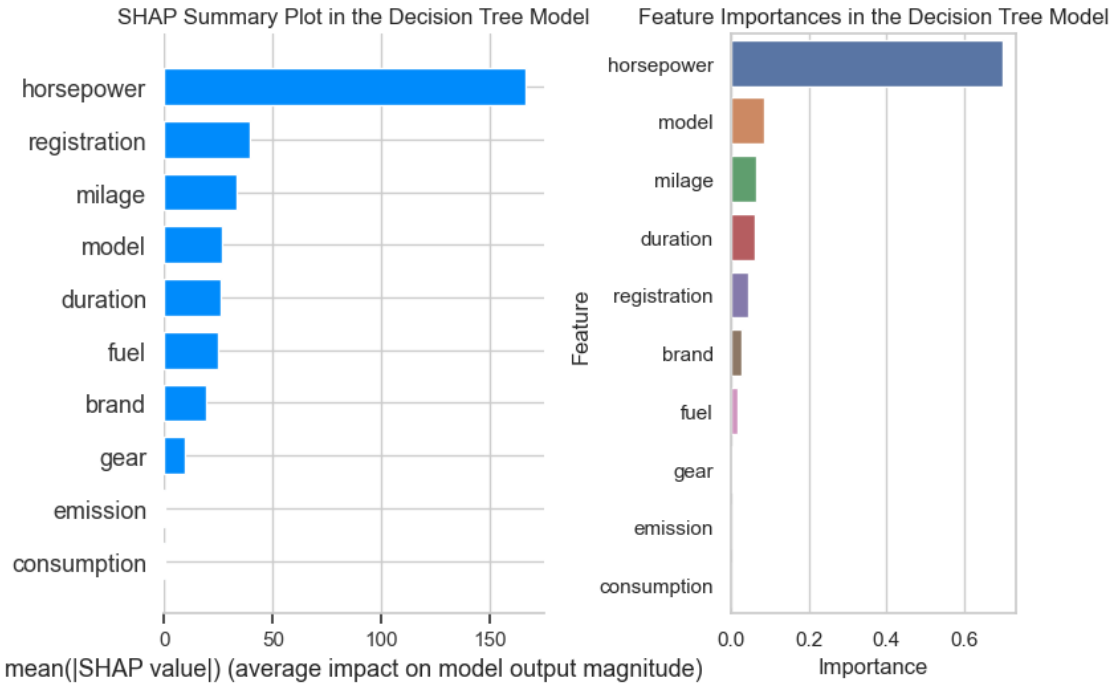
predictions. KNN also shows minor deviation in the alignment between predicted and actual values in the "Actual vs Predicted values" plot, suggesting the model's limitations for the out of sample data. The residuals exhibit also a pattern in the "Residuals vs Predicted values" plot, suggesting that the model may not capture certain patterns in the data effectively

0.12 10 Feature Importance Analysis

In this section, we delve into evaluating the importance of features in our machine learning models. Understanding the significance of different features can provide valuable insights into their impact on the prediction outcome. We use two methods for feature importance assessment: the SHAP (SHapley Additive exPlanations) library and the built-in feature importance functions.

The SHAP library offers a powerful tool for explaining individual predictions by quantifying the contribution of each feature. It provides a comprehensive view of feature importance by considering all possible feature combinations and their respective contributions. Additionally, we utilize the built-in feature importance functions provided by the selected machine learning models. These functions calculate the relevance of features based on various metrics specific to each algorithm.

0.12.1 10.1 Decision tree feature importance



We have two interpretations of feature importance from a Decision Tree model: one is based on Mean SHapley Additive exPlanations (SHAP) values, and the other is based on the inbuilt feature importance of the model. Both interpretations reveal insights into how features contribute to the model's predictive performance.

'Horsepower' is deemed as the most influential feature in both interpretations. With a SHAP value of over 160 and a feature importance score of about 0.7, it is clear that changes in 'Horsepower'

significantly impact the model's predictions. Therefore, 'Horsepower' is a crucial feature for the decision-making process of this model.

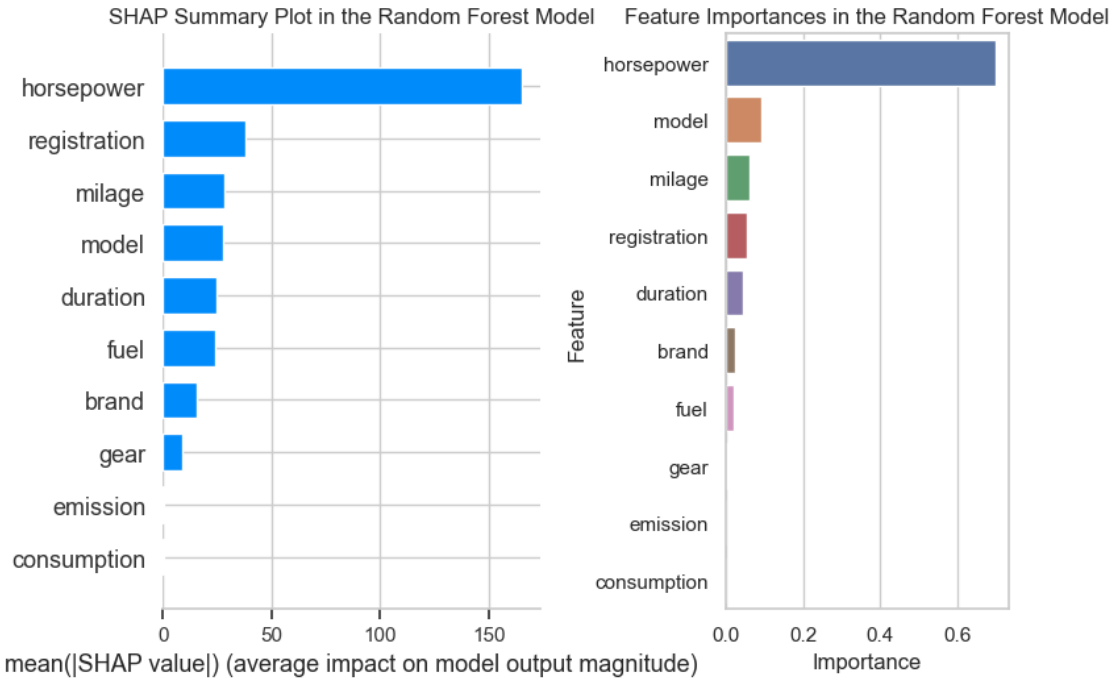
The 'Registration' and 'Model' features are identified as the second most important variables, but in different interpretations. 'Registration' has a significant impact according to SHAP values, while 'Model' stands out in the inbuilt feature importance measure. This disparity may be due to the different ways these metrics calculate importance.

'Mileage' and 'Duration' both have comparable importance levels according to SHAP values and the inbuilt feature importance, with values around 30 and 0.06 respectively. This consistency suggests that while these features play a role in the model's decisions, their impact is less substantial compared to 'Horsepower' and 'Registration' or 'Model'.

Lastly, 'Emission' and 'Consumption' have been identified as having negligible influence in both interpretations. Their low SHAP values and feature importance scores suggest that these features contribute minimally to the model's predictive ability.

In summary, 'Horsepower' is the key feature in this model, followed by 'Registration' or 'Model', and then 'Mileage' and 'Duration'. The features 'Emission' and 'Consumption' have little to no impact on the model's decision-making process, indicating potential for simplifying the model without significantly impacting its accuracy. These interpretations can guide feature selection and engineering in future model iterations, and remind us that different feature importance methods may yield different perspectives.

0.12.2 10.2 Random forest feature importance



From both the Mean SHAP values and the inbuilt feature importance of the Random Forest model, we observe similar patterns:

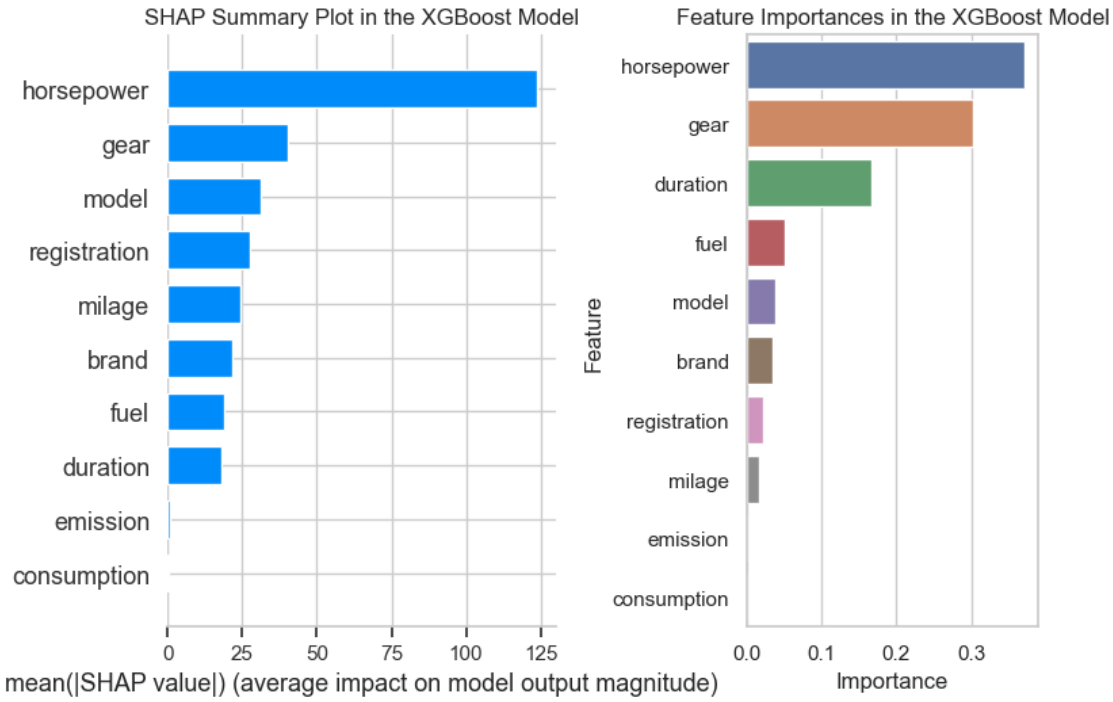
'Horsepower' is the most influential feature, with high SHAP values (~165) and importance score (~0.7), making it crucial for the model's decision-making.

'Registration' and 'Model' are secondary in importance. The SHAP values highlight 'Registration' more (~40), while the inbuilt importance emphasizes 'Model' more (~0.09).

'Mileage' is similarly impactful in both measures (~35 SHAP, ~0.06 importance), indicating its moderate contribution.

Lastly, 'Emission' and 'Consumption' are negligible in both interpretations, indicating their minimal impact on the model's predictive ability.

0.12.3 10.3 XGB feature importance



The interpretations for both Mean SHAP values and the built-in feature importance for the XGBoost model are as follows:

In the SHAP interpretation, 'Horsepower' is the most significant feature (~120), followed by 'Gear' (~40), 'Model' (~30), and 'Registration' (~25). 'Emission' and 'Consumption' are not significant.

The built-in feature importance of XGBoost, often determined by F-score (a measure of how frequently each feature appears in the model splits), indicates a similar importance of 'Horsepower' (~0.35) and 'Gear' (~0.3), followed by 'Duration' (~0.16). Again, 'Emission' and 'Consumption' aren't significant.

So, in both interpretations, 'Horsepower' is paramount, 'Gear' is important, while 'Emission' and 'Consumption' have little influence. The SHAP values emphasize the 'Model' and 'Registration' features, while the built-in importance underscores 'Duration'.

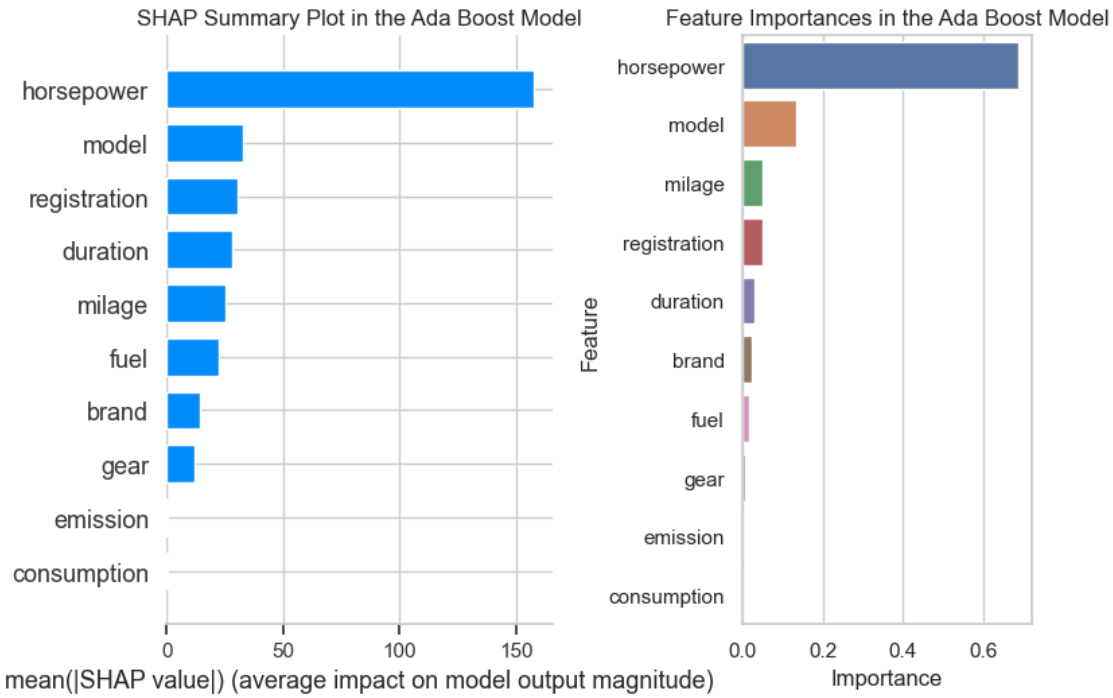
The method to compute this feature importance is through an F-score, which essentially measures how frequently each feature appears in the models created during the boosting process.

In XGBoost, each decision tree is built by repeatedly splitting the data into two groups. Each split involves a single feature at a time. The more frequently a feature is used in making splits across all trees, the higher its F-score, and thus the more important it is considered to be. This is because a feature that is often used for splitting is one that does a good job of separating the data, thereby improving the model's performance.

In your XGBoost model, 'Horsepower' has the highest built-in feature importance, followed by 'Gear' and then 'Duration'. This means that these three features are the ones most often used to split the data in your model, and thus they have the most significant impact on your model's predictions. Conversely, 'Emission' and 'Consumption' are not important, meaning they are rarely used in data splits and have little effect on the predictions.

It's worth noting that while built-in feature importance gives us a good indication of which features are most useful for making predictions, it doesn't tell us anything about the nature of the relationships between these features and the target variable.

0.12.4 10.4 AdaBoost feature importance



The feature importance analysis using SHAP and the built-in feature importance for AdaBoost provides valuable insights into the significance of different features in the model.

According to the SHAP interpretation, the most important feature is 'Horsepower' with a mean SHAP value of approximately 160. This indicates that variations in 'Horsepower' have a substantial impact on the model's predictions. Following 'Horsepower', the 'Model' feature has a mean SHAP

value of around 30, highlighting its relevance in influencing the model's output. 'Registration' and 'Duration' also play a notable role in the predictions with mean SHAP values of approximately 25. On the other hand, 'Emission' and 'Consumption' are considered unimportant as they have negligible or no influence on the model's predictions based on the SHAP analysis.

The built-in feature importance further supports the importance of 'Horsepower' in the AdaBoost model, as it has the highest feature importance value of around 0.7. This suggests that 'Horsepower' is frequently used in the model's splits, indicating its significance in determining the predictions. 'Model' is the next important feature with a feature importance value of approximately 0.13, followed by 'Milage' and 'Registration' with values around 0.05. Similar to the SHAP interpretation, 'Emission' and 'Consumption' are considered unimportant based on their low feature importance values.

Overall, the feature importance analysis using both SHAP and the built-in feature importance in AdaBoost consistently identifies 'Horsepower' as the most influential feature. 'Model' and 'Registration' also play significant roles in the model's predictions. On the other hand, 'Emission' and 'Consumption' are found to be unimportant and have minimal impact on the predictions.

It is important to note that the built-in feature importance is computed based on the frequency of feature usage in the model splits. This information helps identify which features are frequently employed in the boosting process and thus contribute more to the model's predictions. However, it does not provide insights into the specific relationships between these features and the target variable.

In summary, based on the feature importance analysis, focusing on 'Horsepower,' 'Model,' and 'Registration' would likely yield the most informative and influential features for predicting the target variable in the AdaBoost model, while 'Emission' and 'Consumption' can be considered less relevant.

0.13 11 Model Selection

After thoroughly evaluating and comparing the performance of various machine learning models for predicting the leasing rates of vehicles, the AdaBoost Regressor emerged as the final model of choice for implementation in the graphical user interface (GUI). It is noteworthy that the AdaBoost Regressor outperformed several other models that were related to it, such as Decision Tree, Random Forest, and XGBoost, in terms of different evaluation metrics.

AdaBoost, short for Adaptive Boosting, is an ensemble learning method that combines multiple weak learners to create a strong predictive model. The algorithm iteratively trains weak learners, usually decision trees, by assigning higher weights to the misclassified instances in each iteration. The subsequent weak learners focus on correcting the errors made by the previous ones, thereby improving the overall predictive accuracy.

The decision to choose AdaBoost as the preferred model was driven by several factors. Firstly, when analyzing the performance on the testing data, AdaBoost showcased remarkable performance with a low mean squared error (MSE) of 57.512628, a low mean absolute error (MAE) of 2.461741 and a high coefficient of determination (R2) value of 0.999654, indicating its ability to accurately capture the complexities of the leasing price prediction task. Furthermore, AdaBoost exhibited a relatively small mean residual of 0.1737747323158299, signifying its effectiveness in reducing prediction errors. This impressive performance was also the case for the out of sample data (where AdaBoost also outperformed most of the other models), indicating its capacity to produce accurate predictions for

unseen data. Overall, the metrics indicate that AdaBoost performs exceptionally well in accurately predicting the leasing rates of vehicles. Its low values for MSE, RMSE, MAE, and MAPR, along with a high R2 value, demonstrate the model's ability to minimize prediction errors, explain a large portion of the variance, and provide accurate predictions for the leasing rates based on the selected variables.

In conclusion, the selection of AdaBoost as the final model for the GUI implementation was based on its exceptional performance, particularly when compared to related models. Its capability to accurately predict leasing rates, as demonstrated by the evaluation metrics and relatively small mean residual, establishes it as a reliable tool for leasing banks in assessing asset values and making informed pricing decisions. The surprising outperformance of AdaBoost in comparison to other models underscores the significance of rigorous evaluation and highlights the potential of ensemble learning techniques in achieving superior predictive performance.

0.14 12 Graphical User Interface (GUI)

Our script creates a GUI application that allows the user to input the details of a car and get a predicted leasing price. The prediction is done using the previously created best performing model. Our GUI is designed using the libraries customtkinter, pandas, numpy, sklearn, joblib, pyglet and PIL. The script also includes data validation and preprocessing steps to ensure that the input data is suitable for the prediction model.

The structure will be explained in more detail below:

1. Loading the dataset: The script loads a preprocessed dataset from a CSV file. This dataset is used for populating the GUI elements and for making predictions. The dataset is assumed to have columns such as "brand", "model", "milage", "registration", "duration", "gear", "emission", "consumption", "horsepower", and "fuel". Furthermore, numeric values are expected to be floats.
2. Creating the GUI window and frame: The script creates the main GUI window (root) with a specific size and title. Next, a frame is created and placed in the center of the root window. This frame will contain all the other widgets. Additionally the logo is loaded from a PNG file and resized. In order for the image to fit our design, the color is changed to white and assigned to a label at the top of the GUI.
3. Creating entry fields: Several entry fields and labels are created for the user to input the car's details. These details include the brand, model, mileage, first registration date, contract duration, gear, emission value, consumption, engine power, and fuel type. Some of these fields are initially hidden and can be shown by checking a checkbox. The brand and model fields are populated with unique values from the dataset. To enhance the user's experience, the model field is updated based on the selected brand. This way the user will not be able to create a prediction for a car that would not exist, such as "BMW S-Class", for example.
4. Creating the calculate button: The script creates a button that, when clicked, collects the data from the entry fields, validates and preprocesses the data, and then uses a pre-trained model – in our case XGBoost - to predict the leasing price. The predicted price is then displayed in the GUI. The validation checks if the numeric fields contain valid numeric values. If an error occurs in relation to the input, a corresponding error message is displayed. Given that the user corrects his input, all of the given values will be imported in the model and return a prediction. Any features not included by the user will be replaced by NaN-values which will then be used for the calculation.

0.15 13 Possible improvements and potential future work

Although this project successfully developed machine learning models for predicting the leasing prices of vehicles, there are several areas that can be explored for further improvement and future research.

Firstly, expanding the dataset by including additional features such as vehicle price, model year, and regional market conditions could enhance the predictive capabilities of the models. However, theoretically it could also be the case that removing unnecessary features would be the better way to improve the predictions because this strategy could avoid overfitting, because a model could pick up noise from the training data and perform poorly on unseen data. Through feature selection, we might get rid of variables that don't add much to the model's predictive ability, simplifying it and possibly enhancing how well it performs on unobserved data. This can be done through machine learning-based techniques like recursive feature elimination. Both possibilities (adding additional features and implementing feature selection) could be testes for future work that builds on this project.

Additionally, incorporating external data sources, such as economic indicators or industry trends, could provide valuable insights and enhance the models' ability to capture market dynamics. Another aspect that could improve the models a lot would be the implemention feature interactions. In practice, complex interdependencies between variables are possible and may have a major impact on our model's ability to predict outcomes. Therefore, adding interaction effects to our model would be a possible way to make it better. When there is a non-additive link between two or more independent variables and the dependent variable, interaction effects develop. These interactions might help us understand how our attributes interact to affect our predictions in more subtle ways. A car could have more power, but still consume less due to an improved engine. If performance and consumption are considered separately, both variables have a positive effect on the leasing price. However, this would mean that a car with more power and consumption would automatically predict a higher price than a car with the same power but lower consumption.

Furthermore, by performing feature engineering and selection methods, one can enhance the models even more by identifying the most significant attributes for pricing predictions and taking into account interaction effects. Lastly, continuous model monitoring and updating are essential to ensure the models remain accurate and relevant over time, considering the dynamic nature of the automotive market.

By addressing these areas, future work can lead to improved models and a more comprehensive decision support system for consulting firms, facilitating better assessment of asset values and pricing strategies in an ever-changing market environment.

0.16 14 Safe models

To safe all built models, we included a section to write the models into a model folder inside the working directory (the directory where the notebook is located). The models are safed as a ".joblib" file, which according to scikit-learn, is more efficient than pickle.

0.17 15 Debugging library versions

This short section is supposed to help debug any problems related to python or library versions.

0.17.1 Sklearn

The used Scikit-learn version has to be >1.2.2, for that the Python environment has to be >3.8!

0.17.2 Pandas

The used Pandas version is 2.0.1

0.17.3 Numpy

The used numpy version is 1.23.5, which is crucial for using SHAP.

0.17.4 Shap

The used shap version is 0.41.0

0.17.5 Numba, llvmlite (used in shap)

The used Numba version is 0.57.0, for llvmlite it's 0.40.0

```
False
```

0.17.6 XGBoost

The used XGBoost version is 1.7.5

0.18 References

This work draws inspiration from the master thesis conducted by Thomas Dornigg. To delve deeper into Dornigg's thesis, please refer to the following link: Link to Thomas Dornigg's Master Thesis

- Train/test split: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_spli
- OneHot and ordinal encoding: https://stackoverflow.com/questions/69052776/ordinal-encoding-or-one-hot-encoding
- OneHot vs ordinal encoding: https://github.com/slundberg/shap/issues/397
- XGBoost hyperparameter tuning: https://www.kaggle.com/code/prashant111/a-guide-on-xgboost-hyperparameters-tuning
- get hyperparameters XGBoost: https://stackoverflow.com/questions/69639901/retrieve-hyperparameters-from-a-fitted-xgboost-model-object
- SHAP explained: https://shap.readthedocs.io/en/latest/index.html
- Sklearn documentation: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- Why label encoding should not be used on input data: https://stackoverflow.com/questions/59914210/why-shouldnt-the-sklearn-labelencoder-be-used-to-encode-input-data
- Odrinal encoder: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.h

- How shap values would work with OneHot encoding: https://www.reddit.com/r/datascience/comments/s2epy0/computing_categorical_feature_importance_us
- shap values of categorical variables: https://github.com/slundberg/shap/issues/397
- Looked into parallelizing shap calculations: https://towardsdatascience.com/parallelize-your-massive-shap-computations-with-mllib-and-pyspark-b00accc8667c
- Sklearn Decision Tree Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTre
- Sklearn AdaBoost: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.
- Sklearn Random Forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForest
- Sklearn KNN Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsR
- Sklearn SVR: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html
- XGBoost CUDA GPU acceleration: https://xgboost.readthedocs.io/en/stable/gpu/index.html
- XGBoost: https://xgboost.readthedocs.io/en/stable/index.html
- Sklearn Decision Tree Regression explained: https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html
- Support Vector Machines explained: https://en.wikipedia.org/wiki/Support_vector_machine
- XGBoost feature importance: https://mljar.com/blog/feature-importance-xgboost/#:~:text=About%20Xgboost%20Built%2Din%20Feature%20Importance&text=You%20can%20che
- Feature importance explained with shap: https://www.aidancooper.co.uk/a-non-technical-guide-to-interpreting-shap-analyses/

0.19 Appendix

The appendix contains a variety of methods that we explored prior to settling on the current approach.

0.19.1 A1 Encoding differences

As we progressed in the creation of this machine learning notebook, we shifted from OneHot encoding to ordinal encoding. Notably, while some algorithms, like tree-based models, demonstrate adaptability to the choice of encoding, others show heightened sensitivity. In particular, Support Vector Machine (SVM) and Support Vector Regression (SVR) algorithms can be influenced by the biases initiated by varying encoding techniques. This sensitivity emanates from the dependency of SVM and SVR on vector geometry, where actual geometric distances between data points significantly impact their computational process.

`OneHot encoding is used.`

Decision Tree

```
Evaluation Metrics:
      Decision Tree Train OneHot  Decision Tree Test OneHot
MSE                    48.476505                  93.750237
```

```
RMSE                    6.962507                   9.682471
MAE                     2.607555                   3.232638
R2                      0.999476                   0.998989
MAPR                    0.005063                   0.006260
```

```
{'ccp_alpha': 0.05648616489184735, 'criterion': 'squared_error', 'max_depth':
None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease':
0.0, 'min_samples_leaf': 2, 'min_samples_split': 20, 'min_weight_fraction_leaf':
0.0, 'random_state': 2023, 'splitter': 'best'}
```

Random Forest

```
Evaluation Metrics:
      Random Forest Train OneHot  Random Forest Test OneHot
MSE                   69.837894                 103.283831
RMSE                   8.356907                  10.162865
MAE                    2.975286                   3.694924
R2                     0.999244                   0.998886
MAPR                   0.005342                   0.006765
```

```
{'bootstrap': True, 'ccp_alpha': 0.0, 'criterion': 'squared_error', 'max_depth':
40, 'max_features': 1.0, 'max_leaf_nodes': None, 'max_samples': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 4, 'min_samples_split': 10,
'min_weight_fraction_leaf': 0.0, 'n_estimators': 700, 'n_jobs': None,
'oob_score': False, 'random_state': 2023, 'verbose': 0, 'warm_start': False}
```

KNN

```
Evaluation Metrics:
      KNN Train OneHot  KNN Test OneHot
MSE          11.761946       336.703067
RMSE          3.429569        18.349470
MAE           0.702970         8.051065
R2            0.999873         0.996370
MAPR          0.001378         0.014772
```

XGB

```
Evaluation Metrics:
      XGB Train OneHot  XGB Test OneHot
MSE          30.026445        65.719158
RMSE          5.479639         8.106735
MAE           2.533013         3.429232
R2            0.999675         0.999291
MAPR          0.004893         0.006648
```

{'objective': 'reg:squarederror', 'base_score': None, 'booster': None,
'colsample_bylevel': None, 'colsample_bynode': None, 'colsample_bytree': 0.75,
'eval_metric': None, 'gamma': 0.35, 'gpu_id': None, 'grow_policy': None,
'interaction_constraints': None, 'learning_rate': 0.06, 'max_bin': None,
'max_cat_threshold': None, 'max_cat_to_onehot': None, 'max_delta_step': None,
'max_depth': 50, 'max_leaves': None, 'min_child_weight': 2,
'monotone_constraints': None, 'n_jobs': None, 'num_parallel_tree': None,
'predictor': None, 'random_state': None, 'reg_alpha': 30, 'reg_lambda': 0.01,
'sampling_method': None, 'scale_pos_weight': None, 'subsample': 0.7,
'tree_method': 'gpu_hist', 'validate_parameters': None, 'verbosity': None}

SVR

Evaluation Metrics:

	SVR Train OneHot	SVR Test OneHot
MSE	667.777238	970.165947
RMSE	25.841386	31.147487
MAE	9.413403	11.768888
R2	0.992775	0.989541
MAPR	0.015477	0.019455

ADA

Evaluation Metrics:

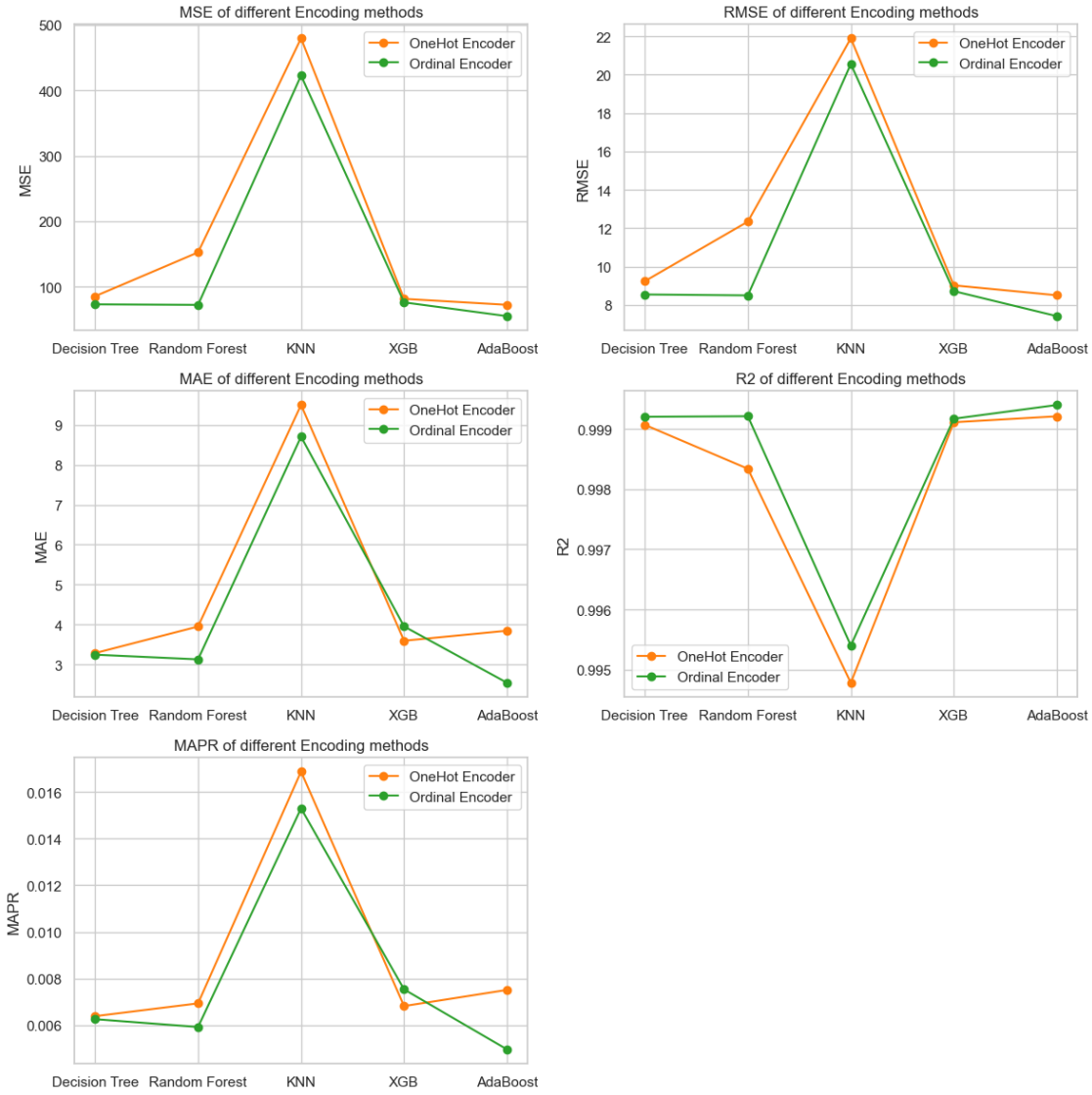
	AdaBoost Train OneHot	AdaBoost Test OneHot
MSE	52.522797	78.499650
RMSE	7.247261	8.860003
MAE	3.231612	3.828675
R2	0.999432	0.999154
MAPR	0.006258	0.007509

	Decision Tree OneHot out of sample	Random Forest OneHot out of sample	\
MSE	85.453642	152.434609	\
RMSE	9.244114	12.346441	
MAE	3.280833	3.948012	
R2	0.999069	0.998340	
MAPR	0.006384	0.006931	

	KNN OneHot out of sample	XGB OneHot out of sample	\
MSE	479.256187	81.498908	\
RMSE	21.891921	9.027675	
MAE	9.504964	3.590986	
R2	0.994781	0.999112	
MAPR	0.016871	0.006816	

	SVR OneHot out of sample	AdaBoost OneHot out of sample

MSE	897.724242	72.393034
RMSE	29.962047	8.508410
MAE	12.000658	3.842671
R2	0.990223	0.999212
MAPR	0.019398	0.007510



With the additional consideration that less hyperparameter tuning was performed on models with OneHot encoding compared to Ordinal encoding, some of the performance differences could be attributed to this imbalance in optimization efforts.

OneHot encoding, while it can lead to a high dimensionality due to the creation of additional binary features, tends to perform better in algorithms such as XGB and AdaBoost as per the observed metrics. However, the hyperparameters for these models may not be as well-tuned as those using Ordinal encoding, potentially affecting the performance comparisons.

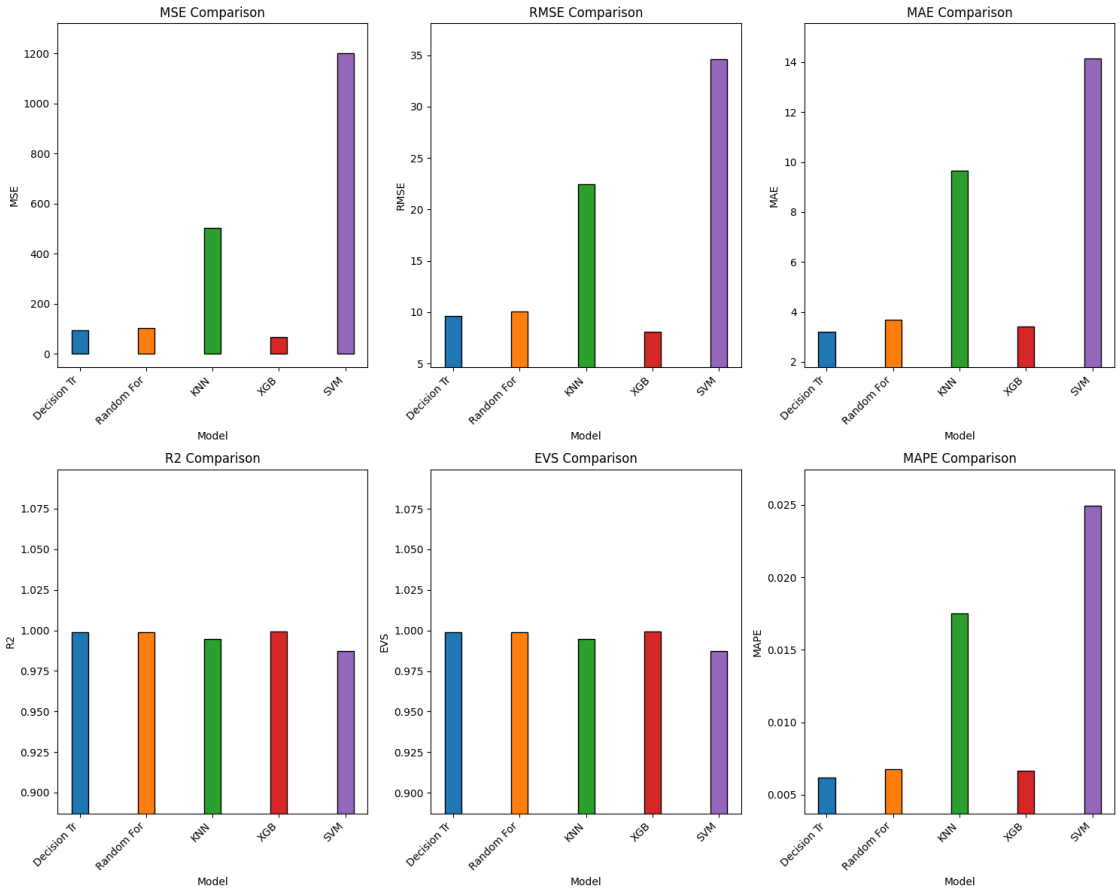
Ordinal encoding, on the other hand, reduces dimensionality and appears to perform better in Decision Trees and Random Forest models based on the metrics like Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE). These models may have been more fine-tuned, providing a more optimized performance.

The Support Vector Machine (SVM) model is notably sensitive to Ordinal encoding, experiencing a significant increase in error rates (MSE, RMSE, and MAE). This is likely because the inherent ranking in Ordinal encoding may distort the data space for SVM, impacting its ability to find an optimal hyperplane.

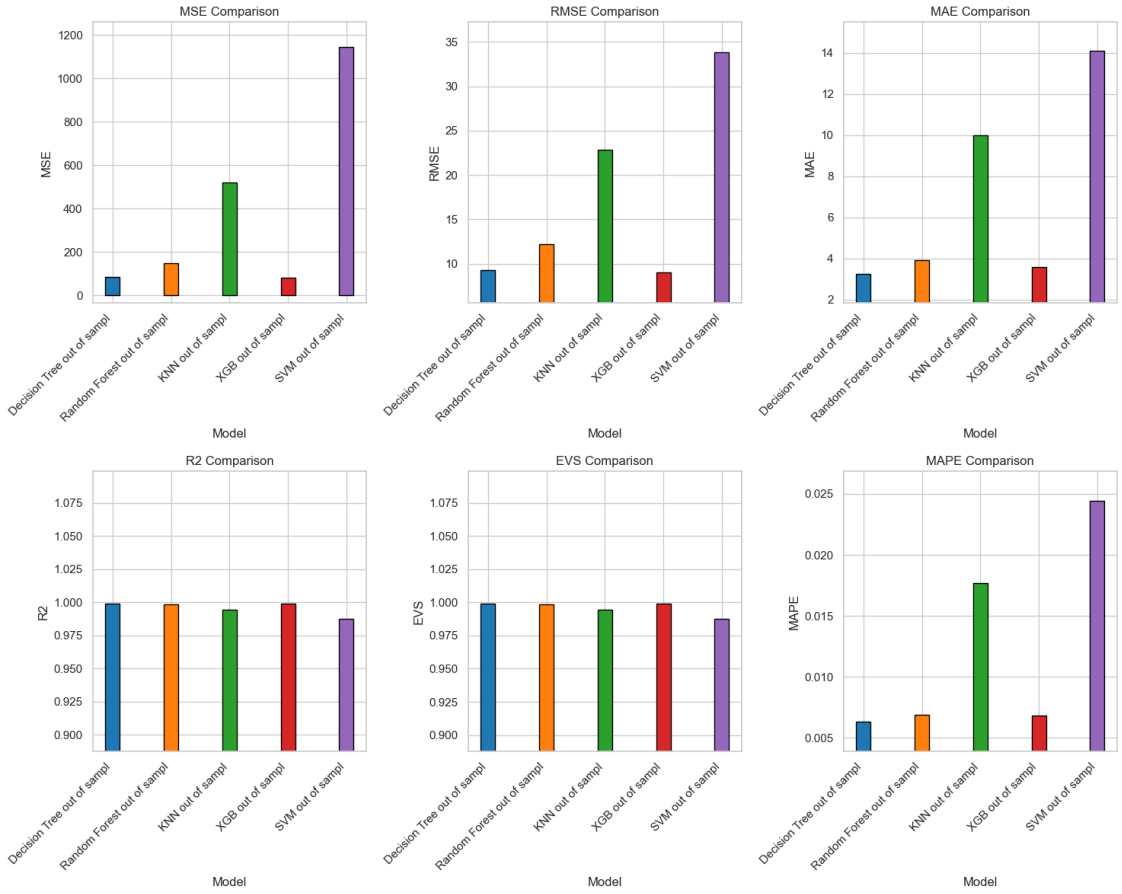
K-Nearest Neighbors (KNN) model is seemingly insensitive to the type of encoding used. However, the performance could potentially improve with more rigorous hyperparameter tuning in the OneHot encoded model.

Overall, it's essential to note that any performance comparison between the models should account for the potential influence of hyperparameter tuning. The observed differences may not solely be due to the choice of encoding method but also the level of optimization for each model. Thus, for a more accurate assessment, it would be beneficial to ensure equal hyperparameter tuning efforts for both OneHot- and Ordinal-encoded models.

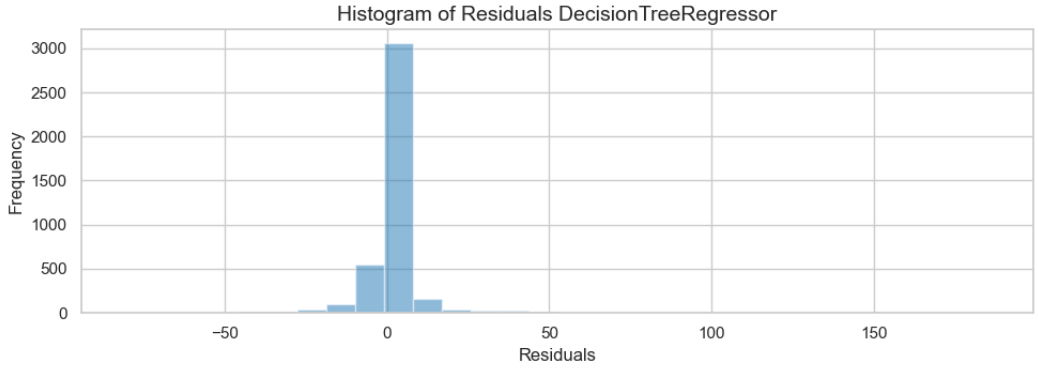
0.19.2 A2 Bar plots Test Performance

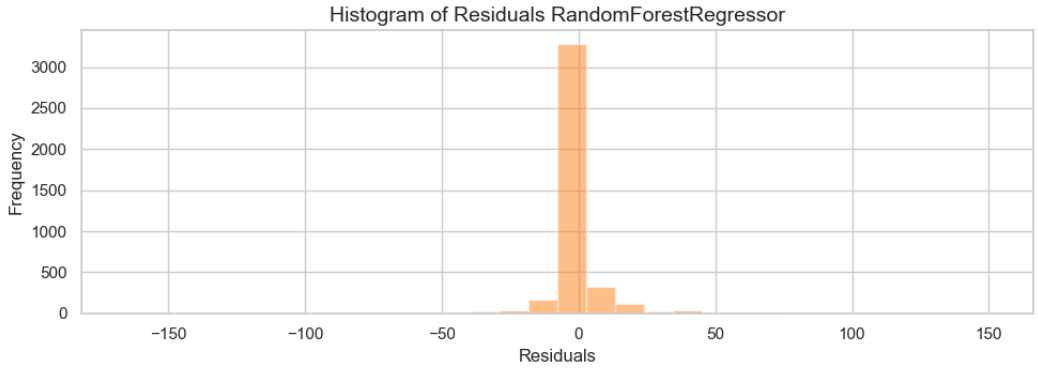


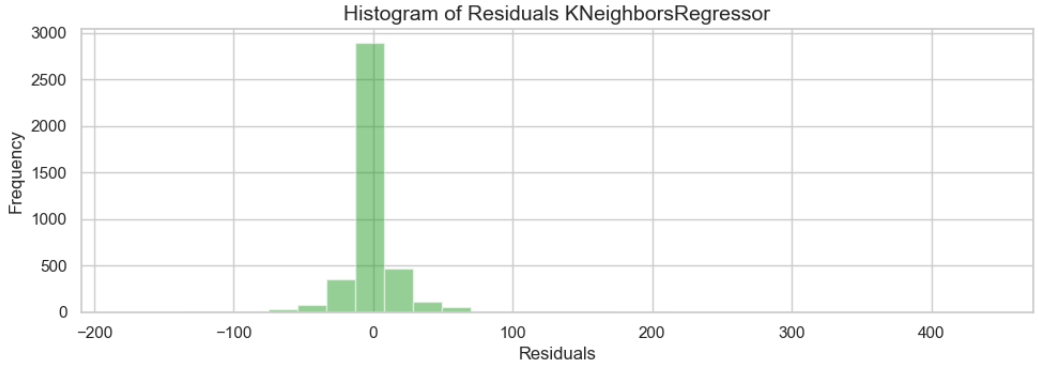
0.19.3 A3 Bar plots Out of sample Performance

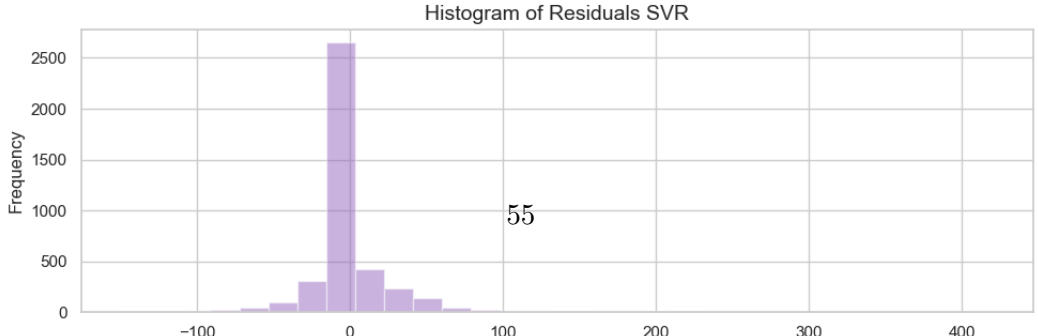


0.19.4 A4 Histogram of residuals test performance

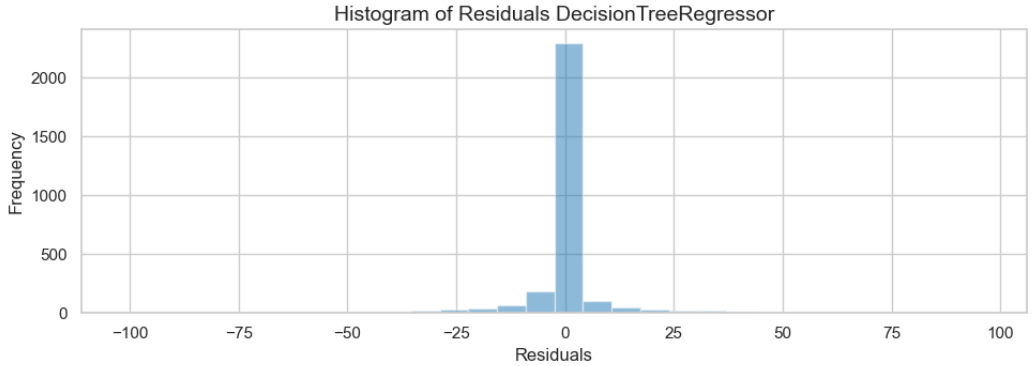


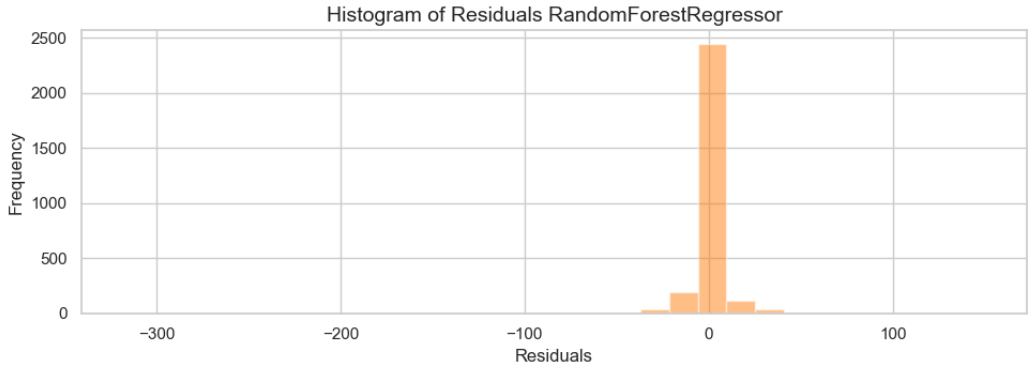


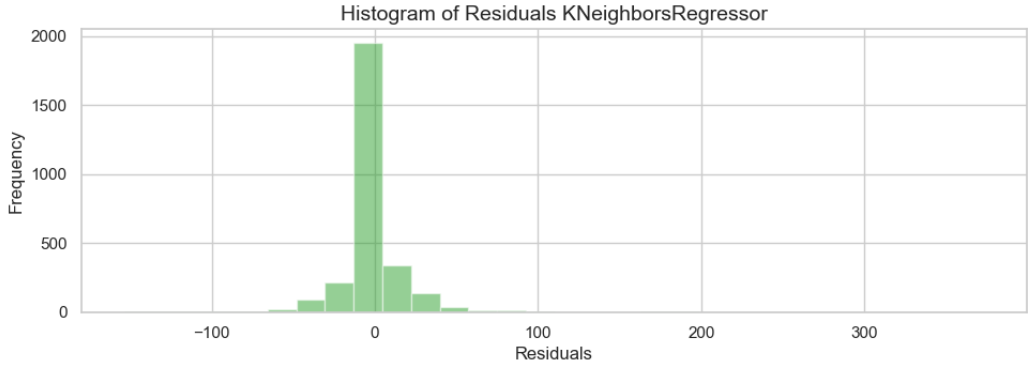


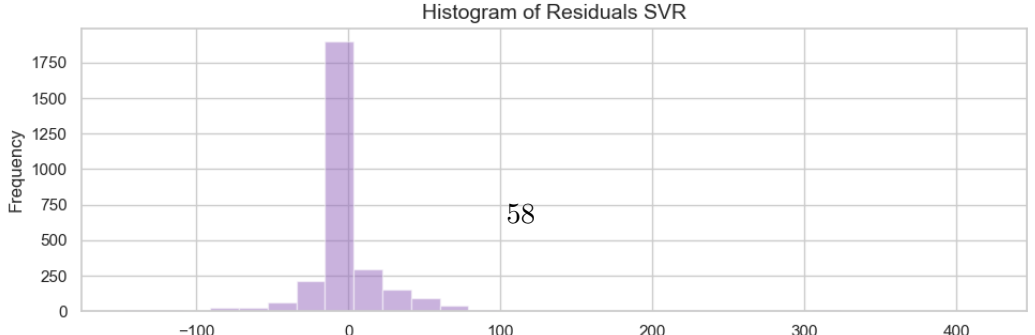


0.19.5 A5 Histogram of residuals out of sample performance









0.19.6 A6 Light models (reduced complexity)

Before switching to ordinal encoding, we used OneHot encoding, which changed the results of the feature importance analysis. In addition, OneHot encoding increases the complexity of a model immensly, because it uses n-1 columns for n values inside a categorical feature.

For the OneHot encoded features, the results were, that the model of the car was not very significant. So we tried to build models with reduced complexity, the light models.

This section shows the building and evaluation of the light models and also shows the effect of leaving out the models in the model building process.

```
Index(['brand', 'gear', 'fuel'], dtype='object')
```

A6.1 Best two performing full models

A6.1.2 Random Forest

```
Evaluation Metrics:
      Decision Tree Train  Decision Tree Test
MSE            209.742431          413.383357
RMSE            14.482487           20.331831
MAE              4.266952            6.098616
R2               0.997731            0.995543
EVS              0.997731            0.995543
MAPE             0.007943            0.011526
```

A6.1.2 Random Forest

```
Evaluation Metrics:
      Random Forest Train  Random Forest Test
MSE            385.868909          451.555470
RMSE            19.643546           21.249835
MAE              7.373744            8.347957
R2               0.995825            0.995132
EVS              0.995825            0.995133
MAPE             0.012751            0.014797
```

A6.1.3 XGB

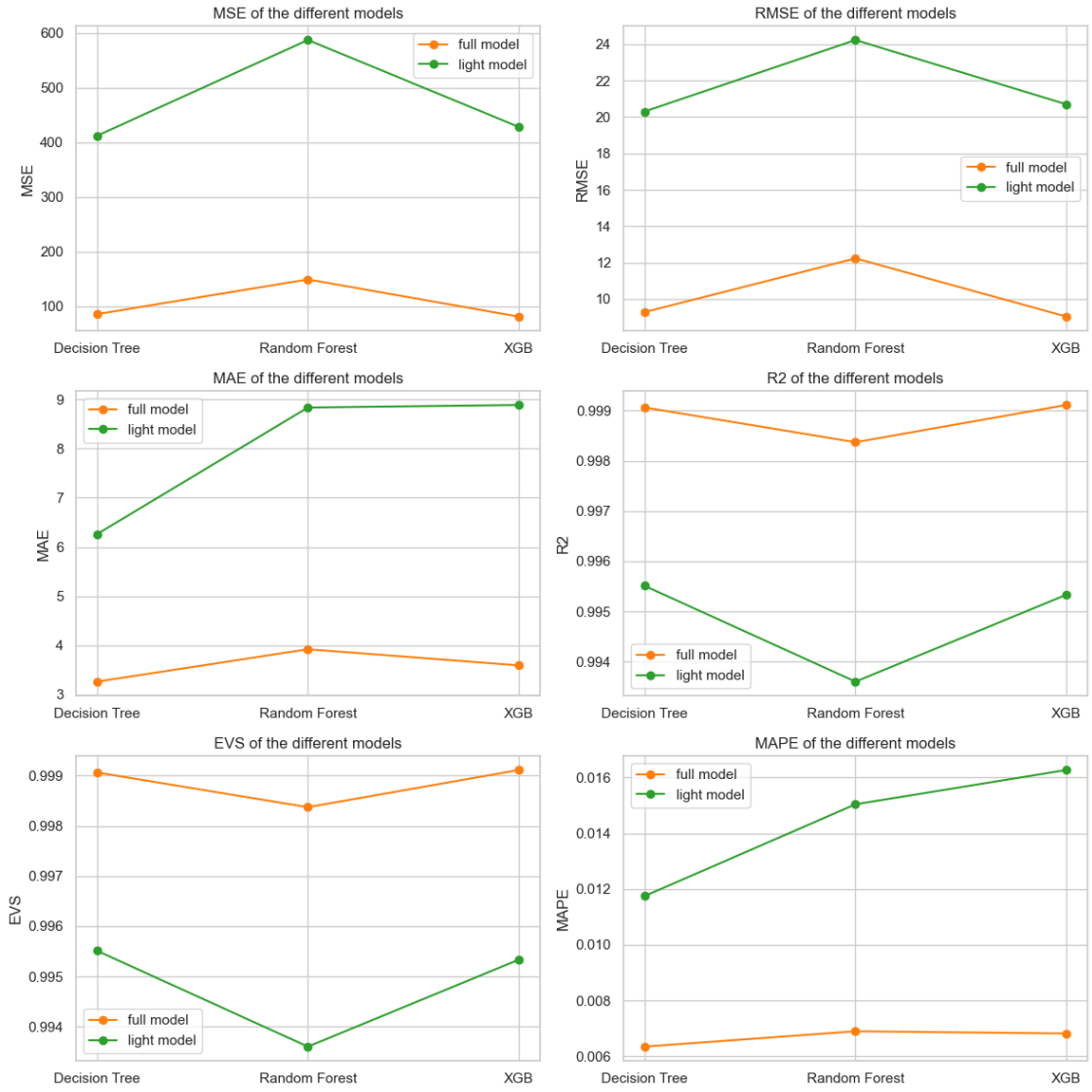
```
Evaluation Metrics:
       XGB Train    XGB Test
MSE   178.409506  365.090424
RMSE   13.357002   19.107340
```

```
MAE     5.696037    8.364257
R2      0.998070    0.996064
EVS     0.998070    0.996065
MAPE    0.010674    0.015629
```

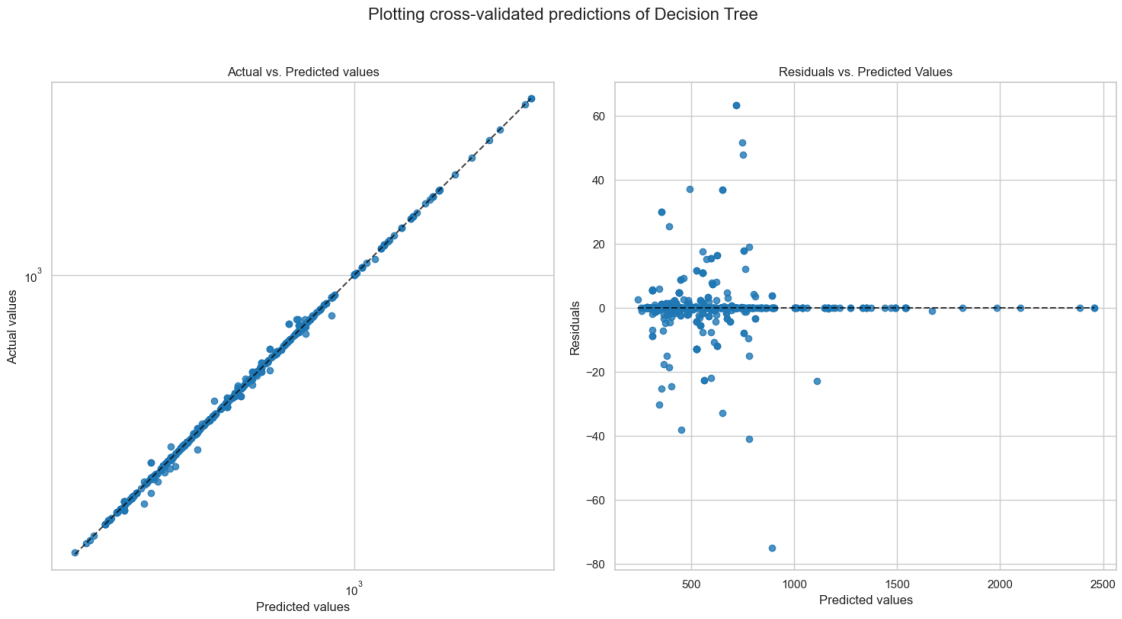
A6.2 Performance full vs light models

```
      Decision Tree out of sample  Random Forest out of sample
MSE                    412.294540                   587.562125  \
RMSE                    20.305037                    24.239681
MAE                      6.260306                     8.837030
R2                       0.995510                     0.993601
EVS                      0.995512                     0.993601
MAPE                     0.011746                     0.015039

      XGB out of sample
MSE          428.523895
RMSE          20.700819
MAE            8.888611
R2             0.995333
EVS            0.995334
MAPE           0.016270
```

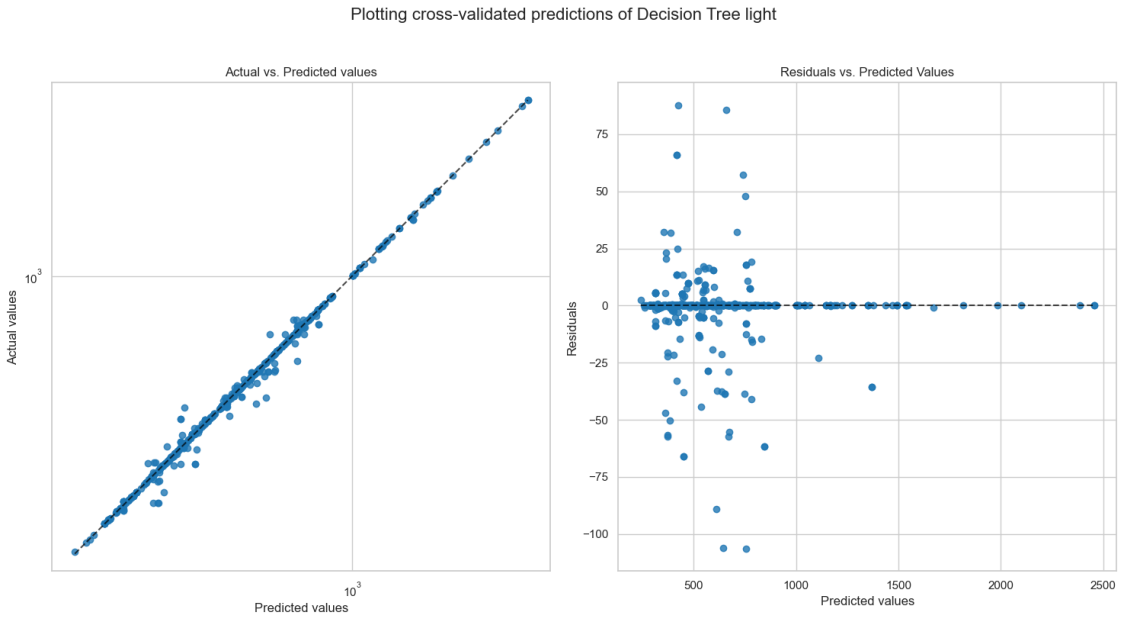
```
      Decision Tree out of sample  Random Forest out of sample
MSE                     86.005198                   149.547716  \
RMSE                     9.273899                    12.228970
MAE                      3.260411                     3.918312
R2                       0.999063                     0.998371
EVS                      0.999064                     0.998371
MAPE                     0.006344                     0.006896

      XGB out of sample
MSE           81.498908
RMSE           9.027675
MAE            3.590986
R2             0.999112
EVS            0.999113
MAPE           0.006816
```



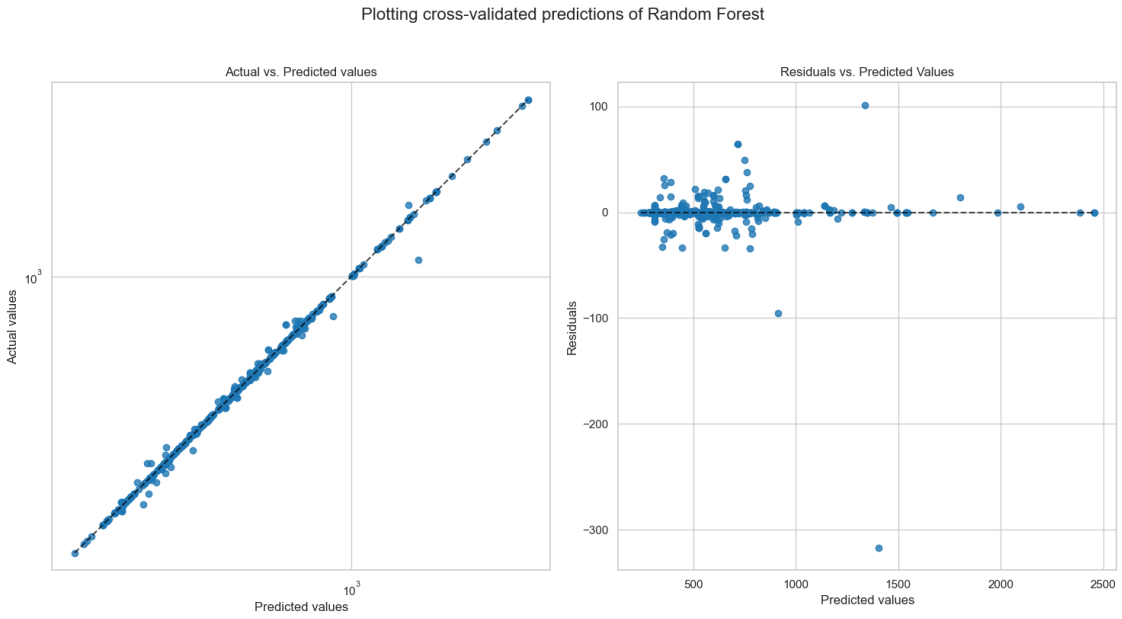
A6.2.1 Performance full vs light models



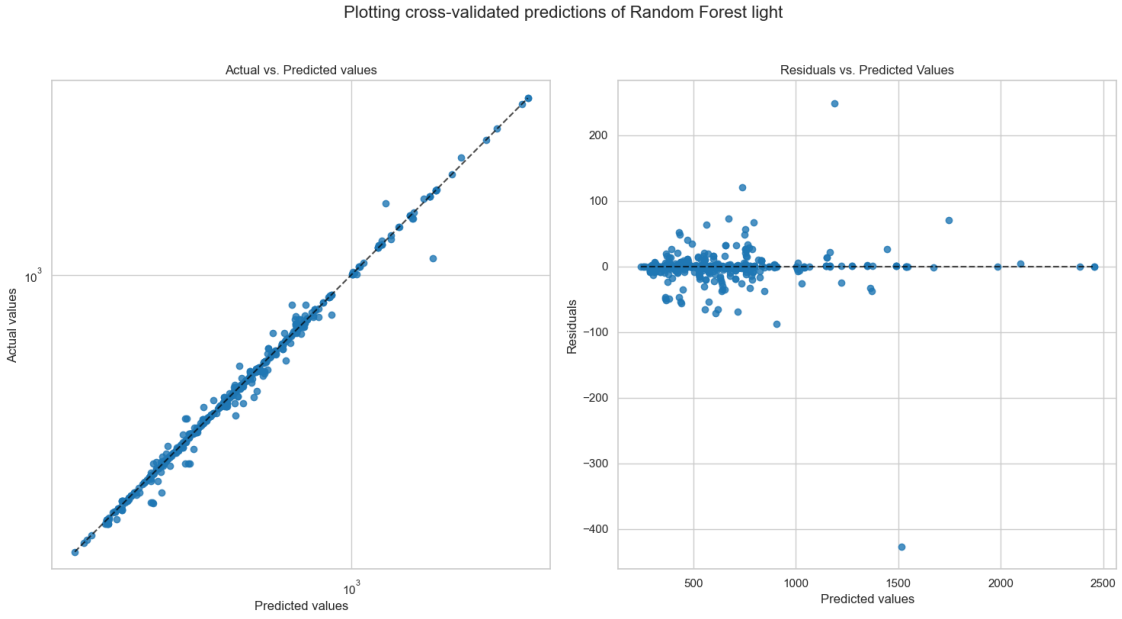
The mean residual of Decision Tree is: -0.14192113936983314



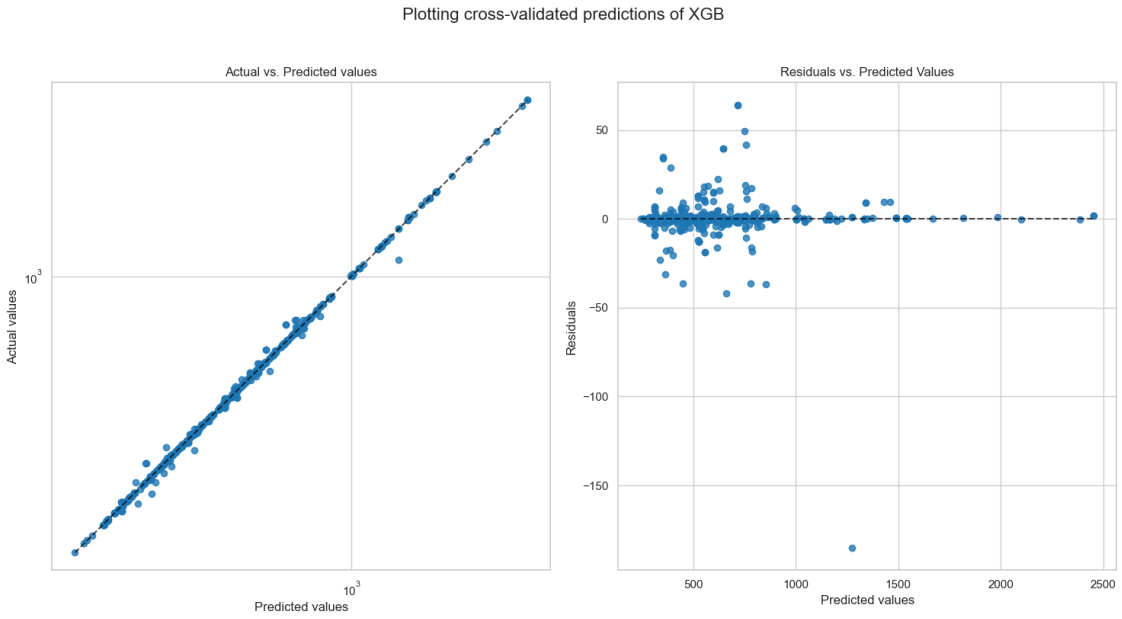
The mean residual of Decision Tree light is: -0.46648864660616046



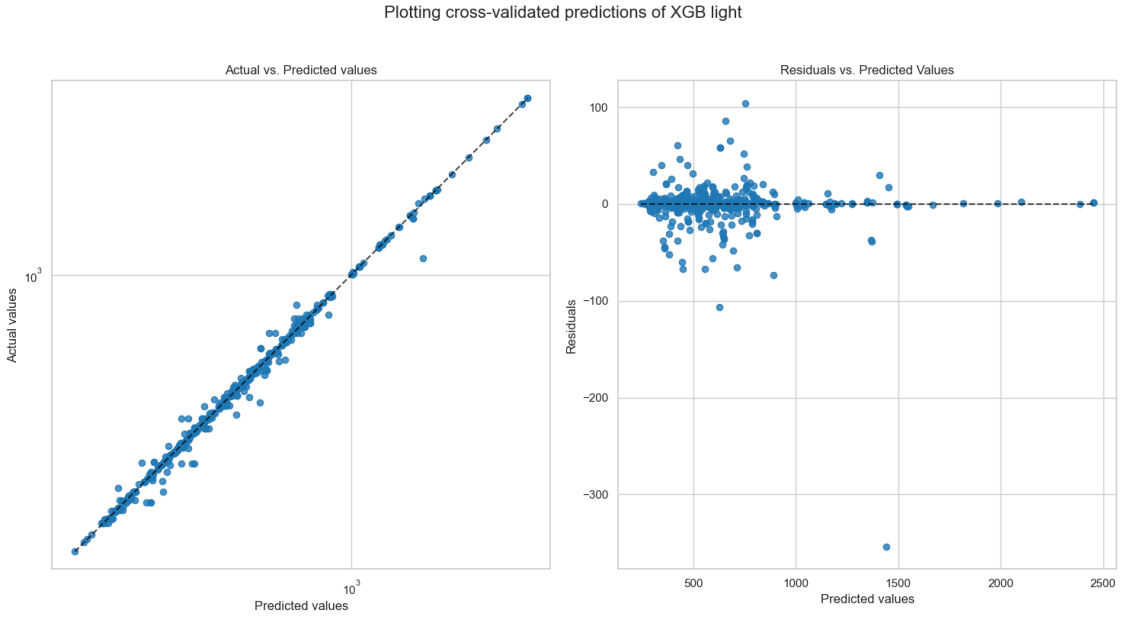
The mean residual of Random Forest is: -0.0004747767460958373



The mean residual of Random Forest light is: 0.10922425080982177




```
The mean residual of XGB is: -0.10674510261257672
```




```
The mean residual of XGB light is: -0.3072127171791553
```