House Price Prediction with Linear Regression and Random Forest

The aim of this project is to predict real-estate prices using the machine learning algorithm, Linear, Ridge and Lasso Regression, and Random Forest.

This file is the EDA and its purpose is to go through the sevarl steps of working with data - data gathering, data understanding, data preparation. Visualization of the information is made for better understanding.

Imports

```
from bs4 import BeautifulSoup as bs4
from requests import get
import json
import pandas as pd
import requests
import matplotlib.pyplot as plt
import seaborn as sns
import mpl_toolkits
import numpy as np
%matplotlib inline
#from fake_useragent import UserAgent
```

Data collection (Web scraping)

Scraping data from the first website - 'FriendlyHousing'

```
url_1 = 'https://www.friendlyhousing.nl/nl/aanbod/kamer'
url_2 = 'https://www.friendlyhousing.nl/nl/aanbod/studio'
url_3 = 'https://www.friendlyhousing.nl/nl/aanbod/appartement'
urls= [url_1, url_2, url_3]
```

Scraping data from the second website - 'Pararius'

```
url_1p = 'https://www.pararius.com/apartments/eindhoven'
url_2p = 'https://www.pararius.com/apartments/eindhoven/page-2'
```

```
url_3p = 'https://www.pararius.com/apartments/eindhoven/page-3'
urls_p= [url_1p, url_2p, url_3p]
```

'FriendlyHousing'

```
#user_agent = UserAgent()
#headers={"user-agent": user_agent.chrome}
soup_array=[]
for url in urls:
    ## getting the reponse from the page using get method of requests module
    page = get(url)

    ## storing the content of the page in a variable
    html = page.content

    ## creating BeautifulSoup object
    soup = bs4(html, "html.parser")
    soup_array.append(soup)
```

'Pararius'

```
soup_array_p=[]
for url in urls_p:
    ## getting the reponse from the page using get method of requests module
    page = get(url)

    ## storing the content of the page in a variable
    html = page.content

    ## creating BeautifulSoup object
    soup = bs4(html, "html.parser")
    soup_array_p.append(soup)
```

'FriendlyHousing' - finding the elements from the html file

```
houses=[]
for s in soup_array:
    allHouses = s.find("ul", {"class": "list list-unstyled row equal-row"})
    #print(len(allHouses))
    for h in allHouses.find_all("li", {"class": "col-xs-12 col-sm-6 col-md-4 equal-col"}):
     # print(h)

      houses.append(h)
     # print(h.findAll("li", {"class": "search-list__item search-list__item--listing"}))
```

```
catalog=[]
for h in houses:
  #data['houses'].append({
      type__= h.find('div', class_= 'specs').text
      t = type__.split()
      type_=t[0]
      street_ = h.find('h3').text
      s = street_.split()
      street = s[0]
      address = h.find('p').text
      a = address.split()
      postcode = a[0]
      #city = a[2]
      price = h.find('div', class_= 'price').text
      vars = type_,street, postcode, price
      catalog.append(vars)
      #print(city)
```

'Pararius' - finding the elements from the html file

```
houses_p=[]
for s in soup_array_p:
    allHouses = s.find("ul", {"class": "search-list"})
    #print(len(allHouses))
    for h in allHouses.find_all("li", {"class": "search-list__item search-list__item--listing
     # print(h)

      houses_p.append(h)
     # print(h.findAll("li", {"class": "search-list__item search-list__item--listing"}))
```

```
catalog_p=[]
for h in houses_p:
  #data['houses'].append({
        name = h.find('a',class_='listing-search-item__link listing-search-item__link--title'
        _name = name.split()
       # if len(_name) == 3:
        #  house_type = _name[0] + _name[1] + _name[2]
        #elif len(_name) == 2:
         # house_type = _name[0]
        #else:
        house_type = _name[0]
        street = _name[1]
        _address= h.findAll('div', class_='listing-search-item__location')[0].text
        #String manipulation to remove the unwanted signs from the address
        __address = _address.replace("\nnew\n ", "")
        address = __address.replace("\n ", "")   #actual address after string manipulation -
        new_address = address.split()
        if new_address[0] == 'new':
```

```
    if new_address[0] == "new":
        postcode=0
    else:
        postcode = new_address[0]
    price_ = h.findAll('span', class_='listing-search-item__price')[0].text
    #splitting the string to find the price
    p=price_.split()
    _price = p[0] #actual price before string manipulation
    __price = _price.replace("€", "")  #actual price before full string manipulation
    price = __price.replace(",", "")   #actual price after string manipulation - ready to

    #finding the whole element from the web page
    ylr= h.findAll('section', class_= 'illustrated-features illustrated-features--vertica

    #splitting the string to find the living are, rooms and year
    lry= ylr.split()

    #living_area after taking the indexes that define it
    living_area = lry[0]

    #rooms after taking the index that defines the variable
    rooms = lry[4]

    vars = house_type, street, postcode,price,living_area,rooms
    catalog_p.append(vars)
    #print(_name)
    print(postcode)
```

```
5624
5653
5652
5622
5612
5612
5643
5616
5612
5612
5612
5644
5612
5642
5612
5615
5616
5616
5611
5622
5611
5645
5616
5616
5652
5611
```

```
5644
5615
5616
5611
5611
5616
5642
5612
5611
5611
5611
5616
5612
5614
5612
5611
5615
5611
5623
5611
5613
5611
5643
5615
5622
5611
5654
5658
5611
5654
5612
5611
5611
5611
```

'FriendlyHousing' - creating the dataframe

```
dataframe = pd.DataFrame(catalog)
dataframe.columns=['TYPE', 'STREET NAME', 'POSTCODE', 'PRICE']
dataframe
```

	TYPE	STREET NAME	POSTCODE	PRICE
0	Kamer	Paul	5642	660
1	Kamer	Heezerweg	5614	390
2	Kamer	Willem	5611	320
3	Kamer	Willem	5611	310
...	...	...	...	...

'Pararius'- creating the dataframe

```
df_ = pd.DataFrame(catalog_p)
df_.columns=['TYPE', 'STREET NAME', 'POSTCODE', 'PRICE','LIVING_AREA', 'ROOMS']
df_
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
0	Apartment	Kronehoefstraat	5622	925	45	2
1	House	van	5622	1450	115	5
2	Room	Heezerweg	5614	595	15	1
3	Apartment	Kerkstraat	5611	895	60	2
4	House	St	5652	1500	161	5
...	...	...	...	...	...	...
88	Apartment	Blaarthemseweg	5654	875	45	2
89	Apartment	Boschdijk	5612	560	20	1
90	Apartment	Sophia	5616	755	40	2
91	Apartment	Hoogstraat	5654	925	42	1
92	Apartment	Treurenburgstraat	5613	1150	78	4

93 rows × 6 columns

Data integration

> Using concat to create a `Union` between the two datasets and then, integrate them into one dataset.

```
frames = [dataframe, df_]
```

```
df = pd.concat(frames)
df
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
0	Kamer	Paul	5642	660	NaN	NaN
1	Kamer	Heezerweg	5614	390	NaN	NaN
2	Kamer	Willem	5611	320	NaN	NaN
3	Kamer	Willem	5611	310	NaN	NaN
4	Kamer	Julianastraat	5611	375	NaN	NaN
...	...	...	...	...	...	...
88	Apartment	Blaarthemseweg	5654	875	45	2
89	Apartment	Boschdijk	5612	560	20	1
90	Apartment	Sophia	5616	755	40	2
91	Apartment	Hoogstraat	5654	925	42	1
92	Apartment	Treurenburgstraat	5613	1150	78	4

225 rows × 6 columns

> After the integration, it is noticeable there are missing values.

Data analysis

Checking the dimension of the dataset and the features.

Take a look at the summary of the numerical fields.

```
#Description of the dataset
df.describe()
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
count	225	225	225	225	93	93
unique	6	102	22	123	55	6
top	Apartment	Leenderweg	5611	415	45	2
freq	75	11	46	14	8	35

```
# Check the dimension of the dataset
```

```
# Check the dimension of the dataset
df.shape
```

```
(225, 6)
```

> The dataset has changing observations(rows), depending on the housing properties on the websites, and 6 features. The data is scraped and this means it is up to date. Whenever there is a change on the websites, there is a change in the dataset.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 225 entries, 0 to 92
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   TYPE         225 non-null    object
 1   STREET NAME  225 non-null    object
 2   POSTCODE     225 non-null    object
 3   PRICE        225 non-null    object
 4   LIVING_AREA  93 non-null     object
 5   ROOMS        93 non-null     object
dtypes: object(6)
memory usage: 12.3+ KB
```

> It can be seen that none features are numeric, but objects. Later, they will have to be converted into either float or int in order to be plotted and then used for the trainig of the models. There are also missing values in the dataset.

After reviewing all the columns, in the next cell is outlined only the quantitive columns.

```
quantitative = [f for f in df.columns if df.dtypes[f] != 'object']
print(quantitative)
```

```
[]
```

To look at the data I'll use the .head() method from pandas. This will show the first 5 items in the dataframe.

```
#First 5 rows of our dataset
df.head()
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
0	Kamer	Paul	5642	660	NaN	NaN
1	Kamer	Heezerweg	5614	390	NaN	NaN
2	Kamer	Willem	5611	320	NaN	NaN
3	Kamer	Willem	5611	310	NaN	NaN

To look at the data I'll use the .tail() method from pandas. This will show us the last 5 items in the dataframe.

```
#Last 5 rows of our dataset
df.tail()
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
88	Apartment	Blaarthemseweg	5654	875	45	2
89	Apartment	Boschdijk	5612	560	20	1
90	Apartment	Sophia	5616	755	40	2
91	Apartment	Hoogstraat	5654	925	42	1
92	Apartment	Treurenburgstraat	5613	1150	78	4

This is a representation of one row content, which helps by showing what to look for and what to expect to be in each other row.

```
df.iloc[0]
```

```
TYPE             Kamer
STREET NAME       Paul
POSTCODE          5642
PRICE              660
LIVING_AREA        NaN
ROOMS              NaN
Name: 0, dtype: object
```

Get the unique values and their frequency of variable. (Checking how many times the certain value occurs.)

```
df['TYPE'].value_counts()
```

```
Apartment      75
Kamer          48
Studio         45
Appartement    39
```

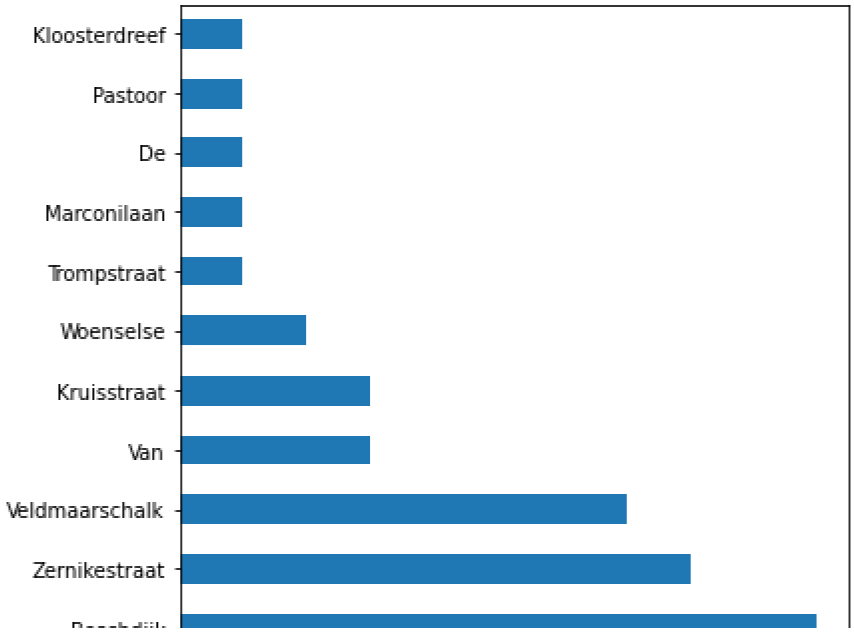
```
Room          10
House          8
Name: TYPE, dtype: int64
```

```
df.groupby('POSTCODE').count()
```

POSTCODE	TYPE	STREET NAME	PRICE	LIVING_AREA	ROOMS
5503	1	1	1	0	0
5611	46	46	46	27	27
5612	38	38	38	14	14
5613	6	6	6	2	2
5614	11	11	11	2	2
5615	17	17	17	6	6
5616	13	13	13	11	11
5621	8	8	8	1	1
5622	12	12	12	6	6
5623	9	9	9	1	1
5624	2	2	2	1	1
5625	2	2	2	1	1
5631	3	3	3	0	0
5642	11	11	11	4	4
5643	13	13	13	2	2
5644	9	9	9	5	5
5645	1	1	1	1	1
5651	4	4	4	0	0
5652	3	3	3	3	3
5653	5	5	5	1	1
5654	10	10	10	4	4
5658	1	1	1	1	1

```
df[(df['POSTCODE'] == '5612')]['STREET NAME'].value_counts().plot(kind='barh', figsize=(6, 6)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fe2c3930810>
```



Sorting the data by `Type`.

```
df.sort_values('TYPE', ascending = True)
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
92	Apartment	Treurenburgstraat	5613	1150	78	4
24	Apartment	Zernikestraat	5612	580	24	1
26	Apartment	Boschdijk	5612	570	21	1
27	Apartment	Hoogstraat	5615	1150	61	3
28	Apartment	Beukenlaan	5616	780	45	2
...	...	...	...	...	...	...
67	Studio	Koenraadlaan	5651	525	NaN	NaN
68	Studio	Koenraadlaan	5651	645	NaN	NaN
69	Studio	Woenselsestraat	5623	520	NaN	NaN
72	Studio	Koenraadlaan	5651	665	NaN	NaN
48	Studio	Melkweg	5642	690	NaN	NaN

225 rows × 6 columns

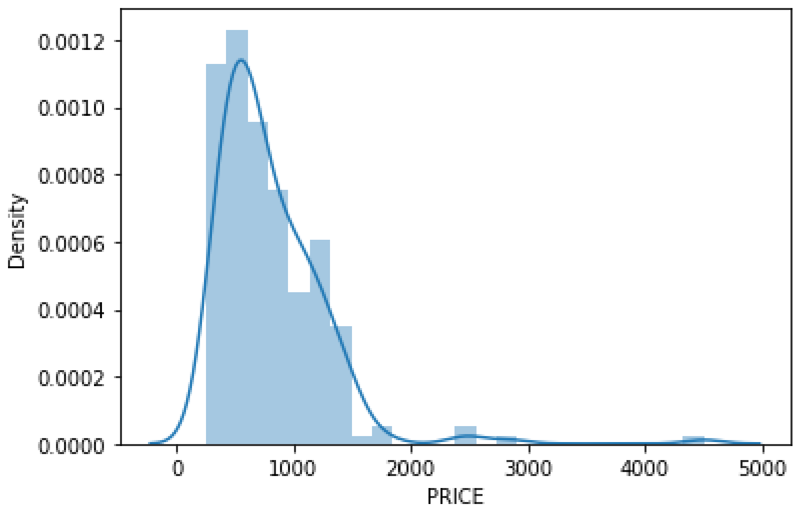
Distribution Analysis

Examining the data distributions of the features. I will start with the target variable, `PRICE`, to make sure it's normally distributed.

This is important because most machine learning algorithms make the assumption that the data is normally distributed. When data fits a normal distribution, statements about the price using analytical techniques will be made.

```
sns.distplot(df['PRICE'])
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
```




```
# Transform the target variable
df['PRICE'] =df['PRICE'].astype(float)

sns.distplot(np.log(df['PRICE']))
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
```

> It can be seen that the `PRICE` distribution is not skewed after the transformation, but normally distributed. The transformed data will be used in in the dataframe and remove the skewed distribution: **Normally distributed** means that the data is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean.

```
df['PRICE_LOG'] = np.log(df.PRICE)
df.drop(["PRICE"], axis=1, inplace=True)
```

Skew is the degree of distortion from a normal distribution. If the values of a certain independent variable (feature) are skewed, depending on the model, skewness may violate model assumptions (e.g. logistic regression) or may impair the interpretation of feature importance.

Reviewing the skewness of each feature:

```
df.skew().sort_values(ascending=False)
```

```
ROOMS          1.563438
LIVING_AREA    1.036734
PRICE_LOG      0.369567
POSTCODE      -0.835538
dtype: float64
```

```
print(df['PRICE_LOG'].skew())
df['PRICE_LOG'].describe()
```

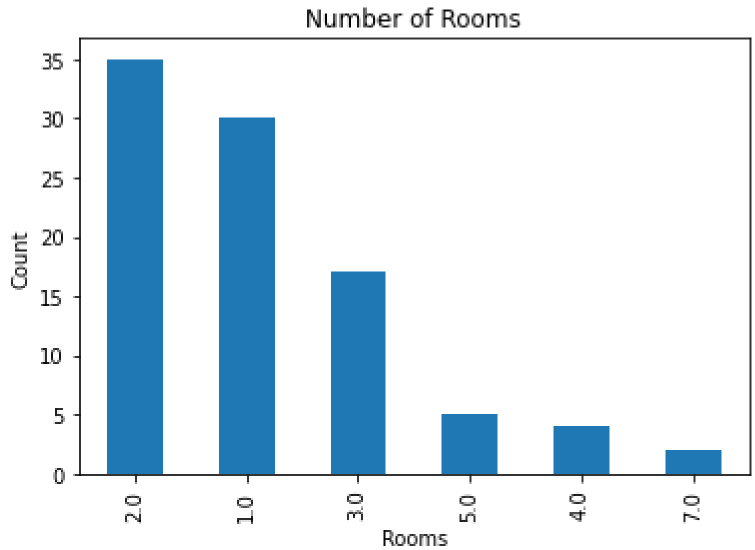
```
0.3695667462353024
count    225.000000
mean       6.568611
std        0.482102
min        5.541264
25%        6.184149
50%        6.522093
75%        6.932448
max        8.411833
Name: PRICE_LOG, dtype: float64
```

> Values closer to zero are less skewed. The results show some features having a positive (right-tailed) or negative (left-tailed) skew.

```
df['ROOMS'].value_counts().plot(kind='bar')
plt.title('Number of Rooms')
```

```
plt.title('Number of Rooms')
plt.xlabel('Rooms')
plt.ylabel('Count')
sns.despine
```

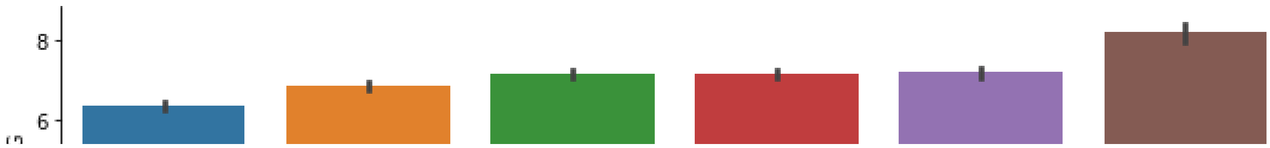
```
<function seaborn.utils.despine>
```



Factor plot is informative when there are multiple groups to compare.

```
sns.factorplot('ROOMS', 'PRICE_LOG', data=df,kind='bar',size=3,aspect=3)
fig, (axis1) = plt.subplots(1,1,figsize=(10,3))
sns.countplot('ROOMS', data=df)
df['PRICE_LOG'].value_counts()
plt.show()
```

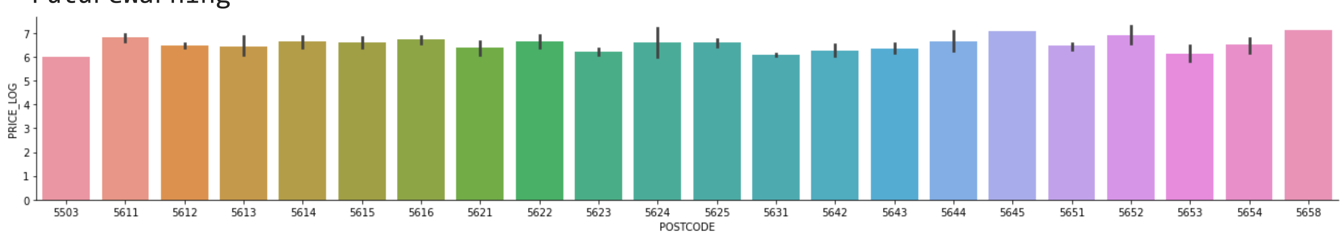
```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3714: UserWarning: The `fa
  warnings.warn(msg)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3720: UserWarning: The `si
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```



> Real estate with 5 rooms has the highest `Price` while the sales of others with rooms of 2 is the most sold ones.

```
sns.factorplot('POSTCODE', 'PRICE_LOG', data=df,kind='bar',size=3,aspect=6)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3714: UserWarning: The `fa
  warnings.warn(msg)
/usr/local/lib/python3.7/dist-packages/seaborn/categorical.py:3720: UserWarning: The `si
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```

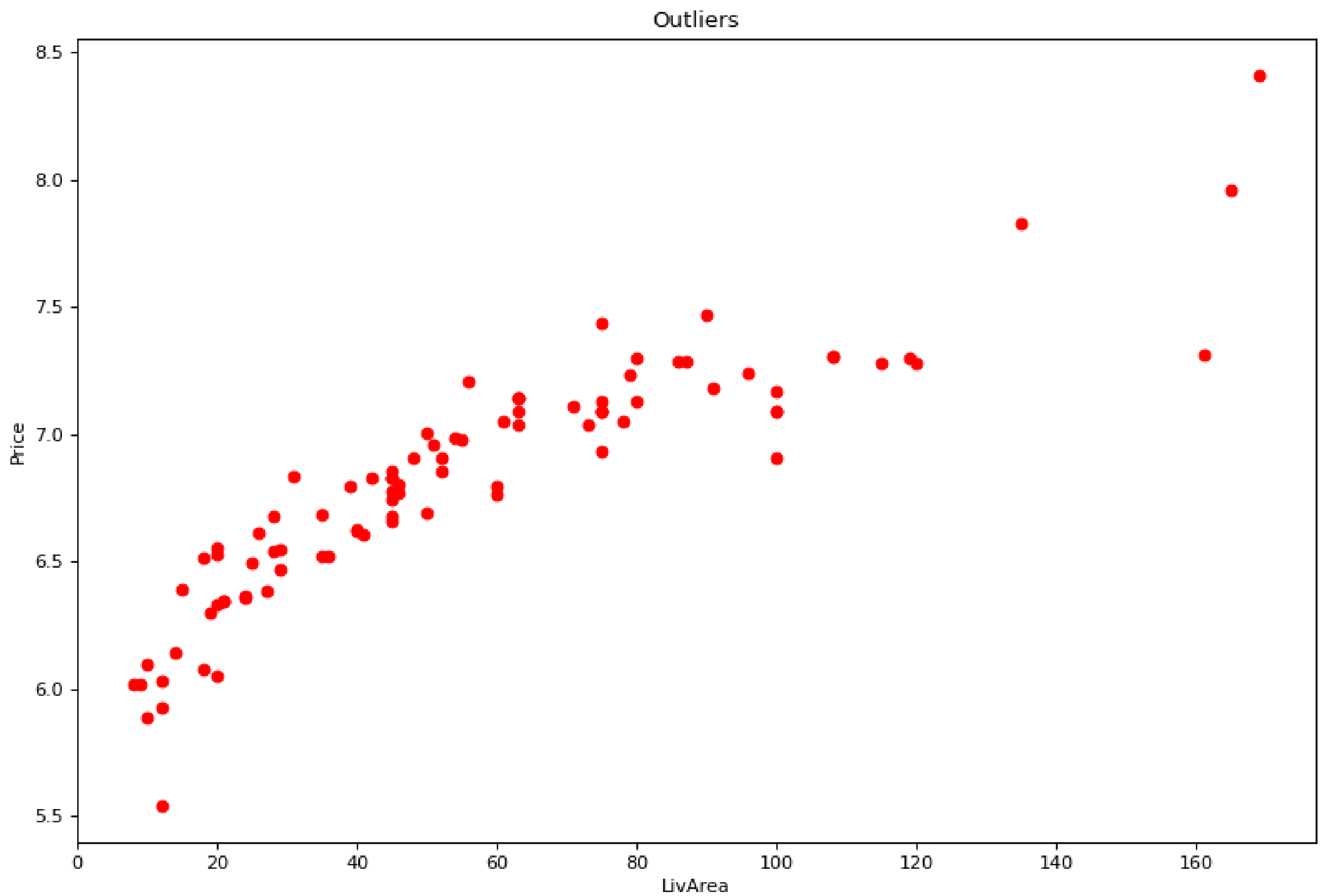


> The diagram represents the `price` of a rpoperty, depending on its `postcode`.

Finding outliers

> An **outlier** is a data point in a data set that is distant from all other observations (a data point that lies outside the overall distribution of the dataset.)

```
plt.figure(figsize=(12, 8), dpi=80)
plt.scatter(df.LIVING_AREA, df.PRICE_LOG, c= 'red')
plt.title("Outliers")
plt.xlabel("LivArea")
plt.ylabel("Price")
plt.show()
```



Converting

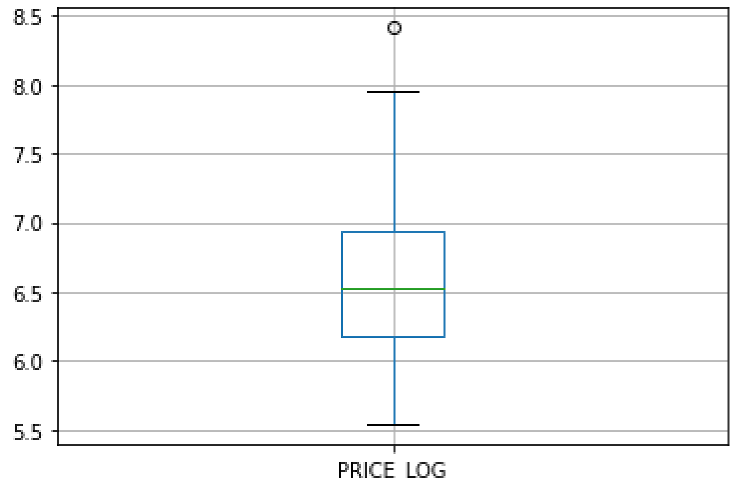
- Converting all categorical variables into numeric ones to use them in the training of the models.

One-Hot Encoding: This process takes categorical variables and converts them to a numerical representation without an arbitrary ordering. What computers know is numbers and for machine learning it is vital to accommodate the feautures into numeric values.

```
df['PRICE_LOG'] =df['PRICE_LOG'].astype(float)
df['POSTCODE'] =df['POSTCODE'].astype(int)
df['LIVING_AREA'] =df['LIVING_AREA'].astype(float)
df['ROOMS'] =df['ROOMS'].astype(float)
code_numeric = {'Kamer': 5,'Apartment': 1, 'Appartement': 1, 'Room': 2, 'Studio': 4, 'House':
df ['TYPE'] = df['TYPE'].map(code_numeric)
df['TYPE'] =df['TYPE'].astype(float)
```

```
df.boxplot(column=['PRICE_LOG'])
plt.show
```

```
<function matplotlib.pyplot.show>
```



Most regression methods explicitly require outliers be removed from the dataset as they may significantly affect the results. To remove the outlier I used the following function:

```
#Check the mean values
df['LIVING_AREA'].mean()
```

```
56.29032258064516
```

```
#Check the median
df['LIVING_AREA'].median()
```

```
48.0
```

```
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
TYPE            3.000000
POSTCODE       30.000000
LIVING_AREA    47.000000
ROOMS           2.000000
```

```
PRICE_LOG       0.748299
dtype: float64
```

```
print(df['PRICE_LOG'].quantile(0.10))
print(df['PRICE_LOG'].quantile(0.90))
```

```
6.00134159214413
7.155306384458393
```

Data cleaning & Data processing

Showing that the values are already transformed to numeric and only the missing values have to be handled.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 225 entries, 0 to 92
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   TYPE         225 non-null    float64
 1   STREET NAME  225 non-null    object 
 2   POSTCODE     225 non-null    int64  
 3   LIVING_AREA  93 non-null     float64
 4   ROOMS        93 non-null     float64
 5   PRICE_LOG    225 non-null    float64
dtypes: float64(4), int64(1), object(1)
memory usage: 17.3+ KB
```

There are missing values in the dataset, which appeared after the data integration of the two datasets. This will be fixed later before the training of the models.

```
df.isnull().sum()
```

```
TYPE             0
STREET NAME      0
POSTCODE         0
LIVING_AREA    132
ROOMS          132
PRICE_LOG        0
dtype: int64
```

Checking if the percentage of missing values of each value and whic has to be dropped if any.

```
# Find columns with missing values and their percent missing
df.isnull().sum()
miss_val = df.isnull().sum().sort_values(ascending=False)
miss_val = pd.DataFrame(data=df.isnull().sum().sort_values(ascending=False), columns=['Missva

# Add a new column to the dataframe and fill it with the percentage of missing values
miss_val['Percent'] = miss_val.MissvalCount.apply(lambda x : '{:.2f}'.format(float(x)/df.shap
miss_val = miss_val[miss_val.MissvalCount > 0].style.background_gradient(cmap='Reds')
miss_val
```

	MissvalCount	Percent
ROOMS	132	58.67
LIVING_AREA	132	58.67

> The light red color shows the small amount of NaN values. If the features were with a high percent of missing values, they would have to be removed. Yet, in this case, they have relatively low percentage so they can be used in future. Then, the NaN values will be replaced.

Filling up the null values in order to train the model.

```
df.fillna(0)
```

	TYPE	STREET NAME	POSTCODE	LIVING_AREA	ROOMS	PRICE_LOG
0	5.0	Paul	5642	0.0	0.0	6.492240
1	5.0	Heezerweg	5614	0.0	0.0	5.966147
2	5.0	Willem	5611	0.0	0.0	5.768321
3	5.0	Willem	5611	0.0	0.0	5.736572
4	5.0	Julianastraat	5611	0.0	0.0	5.926926
...	...	...	...	...	...	...
88	1.0	Blaarthemseweg	5654	45.0	2.0	6.774224
89	1.0	Boschdijk	5612	20.0	1.0	6.327937
90	1.0	Sophia	5616	40.0	2.0	6.626718
91	1.0	Hoogstraat	5654	42.0	1.0	6.829794
92	1.0	Treurenburgstraat	5613	78.0	4.0	7.047517

225 rows × 6 columns

```
df.dropna(inplace=True)
```

```
df.isnull()
```

	TYPE	STREET NAME	POSTCODE	LIVING_AREA	ROOMS	PRICE_LOG
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
88	False	False	False	False	False	False
89	False	False	False	False	False	False
90	False	False	False	False	False	False
91	False	False	False	False	False	False
92	False	False	False	False	False	False

93 rows × 6 columns

Results

Saving into csv file. This decision was made in order to store the results from the extracting data from two websites. Then, the csv can be used in the next part of the project - Modelling.

```
df.to_csv('data.csv')
```

Conclusion

Data collection:

For the data collection part, I decided to use `web scraping` as e technique because it gives the opportunity to work with a data set that is up to date and therefore, makes more accurate summaries.

Data analysis:

From the data analysis it was concluded that:

- There are missing values after the data integration of the two dataframes of the websites.

- The variable vary in types, so they will have to be handled in the next part of the EDA.

Data preprocessing:

I tried different types of data transforms to expose the data structure better, so we may be able to improve model accuracy later. What was noticed during the analysing:

- There are certain outliers which will not interpret with the training of the modelling.
- `Standardizing` was made to the data set so as to reduce the effects of differing distributions.
- `The skewness` of the features was checked in order to see how distorted a data sample is from the normal distribution.
- `Rescaling (normalizing)` the dataset was also included to reduce the effects of differing scales
- The NaN values were filled in in order fo rthe model to be properly trained and give accurate results.