Imports

```
!pip install fake_useragent

    Collecting fake_useragent
      Downloading https://files.pythonhosted.org/packages/d1/79/af647635d6968e2deb57a208d309
    Building wheels for collected packages: fake-useragent
      Building wheel for fake-useragent (setup.py) ... done
      Created wheel for fake-useragent: filename=fake_useragent-0.1.11-cp37-none-any.whl siz
      Stored in directory: /root/.cache/pip/wheels/5e/63/09/d1dc15179f175357d3f5c00cbffbac37
    Successfully built fake-useragent
    Installing collected packages: fake-useragent
    Successfully installed fake-useragent-0.1.11
```

```
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 preparation (Web scraping)

Preparing the data by extracting information from the first three pages of the website.

```
url_1 = 'https://www.pararius.com/apartments/eindhoven'
url_2 = 'https://www.pararius.com/apartments/eindhoven/page-2'
url_3 = 'https://www.pararius.com/apartments/eindhoven/page-3'
urls= ['https://www.pararius.com/apartments/eindhoven','https://www.pararius.com/apartments/e
#var_urls = url_1, url_2, url_3
#urls.append(var_urls)
#result=get(my_url)

page_1 = requests.get(url_1)
page_2 = requests.get(url_2)
page_3 = requests.get(url_3)
#page_2.content
```

```
# html parsing
page1_soup= bs4(page_1.text, "html.parser")
page2_soup= bs4(page_2.text, "html.parser")
page3_soup= bs4(page_3.text, "html.parser")
```

Page 1

```
# grab each product
allHouses1 = page1_soup.findAll("li", {"class": "search-list__item search-list__item--listing
houses1 = page1_soup.findAll("ul", {"class": "search-list"})[0].text
#data_rows = table.findAll('')[2:]
print(len(allHouses1))
print(len(houses1))
```

```
32
13826
```

Page 2

```
allHouses2 = page2_soup.findAll("li", {"class": "search-list__item search-list__item--listing
houses2 = page2_soup.findAll("ul", {"class": "search-list"})[0].text
#data_rows = table.findAll('')[2:]
print(len(allHouses2))
print(len(houses2))
```

```
32
13772
```

Page 3

```
allHouses3 = page3_soup.findAll("li", {"class": "search-list__item search-list__item--listing
houses3 = page3_soup.findAll("ul", {"class": "search-list"})[0].text
#data_rows = table.findAll('')[2:]
print(len(allHouses3))
print(len(houses3))
```

```
32
13592
```

Data description

> - house_type - house/apartment/room
> - street - the name of the street where the property is placed

> - postcode - the postcode of the property
> - price
> - year - the year of construction
> - living_area
> - rooms - how many rooms are in the property

Page 1 - saving the extracted data

```
catalog=[]
for h in allHouses1:
    #data['houses'].append({
        name = h.findAll('a',class_='listing-search-item__link listing-search-item__link--tit
        _name = name.split()
        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()
        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.append(vars)
```

Page 2 - saving the extracted data

```
catalog2=[]
for h in allHouses2:
    #data['houses'].append({
        name = h.findAll('a',class_='listing-search-item__link listing-search-item__link--tit
        _name = name.split()
        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()
        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
        catalog2.append(vars)
```

Page 3 - saving the extracted data

```
catalog3=[]
for h in allHouses3:
    #data['houses'].append({
        name = h.findAll('a',class_='listing-search-item__link listing-search-item__link--tit
        _name = name.split()
        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("\nnew\n", "")
        address = __address.replace("\n ", "")   #actual address after string manipulation -
        new_address = address.split()
        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
        catalog3.append(vars)

print(street)
```

```
Stevinstraat
```

> Saving the scraped data to pandas dataframe (creating the table and giving names to the cokumns)

```
# Create DataFrame
df1 = pd.DataFrame(catalog)
df2 = pd.DataFrame(catalog2)
df3 = pd.DataFrame(catalog3)
df1.columns=['TYPE', 'STREET NAME', 'POSTCODE', 'PRICE', 'LIVING_AREA', 'ROOMS']
df2.columns=['TYPE', 'STREET NAME', 'POSTCODE', 'PRICE', 'LIVING_AREA', 'ROOMS']
df3.columns=['TYPE', 'STREET NAME', 'POSTCODE', 'PRICE', 'LIVING_AREA', 'ROOMS']
```

Data integration

> Using `Union` to integrate the scraped data from the three web pages.

```
frames = [df1, df2, df3]
```

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

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
0	Room	Kronehoefstraat	5622	650	18	1
1	House	Nieuwe	5612	1150	65	3
2	Apartment	P	5611	1395	78	3
3	Apartment	Philitelaan	5617	940	52	1
4	Apartment	Philitelaan	5617	1200	72	3
...	...	...	...	...	...	...
27	Apartment	Karel	5615	1195	95	3
28	House	Leenderweg	5614	1350	150	6
29	Apartment	Paradijslaan	5611	1500	93	3
30	Room	St	5616	495	16	1
31	Apartment	Stevinstraat	5621	795	28	1

96 rows × 6 columns

Data analysis

Here we can see the shape of our data with the .shape. Here we see that we have 31 rows and 7 columns. However, they are always changing because the data is alway up to date by using the web scraping technique.

Checking the dimension of the dataset

```
df.shape
```

```
(96, 6)
```

```
df.head()
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
0	Room	Kronehoefstraat	5622	650	18	1

```
df.describe()
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
count	96	96	96	96	96	96
unique	3	60	21	59	54	7
top	Apartment	Philitelaan	5611	1395	50	2
freq	78	11	21	4	6	30

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96 entries, 0 to 31
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   TYPE         96 non-null     object
 1   STREET NAME  96 non-null     object
 2   POSTCODE     96 non-null     object
 3   PRICE        96 non-null     object
 4   LIVING_AREA  96 non-null     object
 5   ROOMS        96 non-null     object
dtypes: object(6)
memory usage: 5.2+ KB
```

```
df.tail()
```

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
27	Apartment	Karel	5615	1195	95	3
28	House	Leenderweg	5614	1350	150	6
29	Apartment	Paradijslaan	5611	1500	93	3
30	Room	St	5616	495	16	1
31	Apartment	Stevinstraat	5621	795	28	1

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

```
TYPE                       Room
STREET NAME     Kronehoefstraat
POSTCODE                   5622
PRICE                       650
LIVING_AREA                  18
```

```
ROOMS                        1
Name: 0, dtype: object
```

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

	TYPE	STREET NAME	POSTCODE	PRICE	LIVING_AREA	ROOMS
15	Apartment	Accumulatorstraat	5641	800	30	1
2	Apartment	Kruisstraat	5612	950	46	2
0	Apartment	Hoogstraat	5654	1190	80	3
31	Apartment	van	5612	592	18	1
28	Apartment	Dierenriemstraat	5632	790	50	2
...	...	...	...	...	...	...
12	Room	Tongelresestraat	5642	595	19	1
16	Room	Verschaffeltstraat	5623	425	14	1
30	Room	St	5616	495	16	1
25	Room	Leenderweg	5643	580	17	1
0	Room	Kronehoefstraat	5622	650	18	1

96 rows × 6 columns

There are no missing values in the dataset.

```
df.isnull().all()
```

```
TYPE           False
STREET NAME    False
POSTCODE       False
PRICE          False
LIVING_AREA    False
ROOMS          False
dtype: bool
```

```
# 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]
miss_val
```

	MissvalCount	Percent

Pre Processing

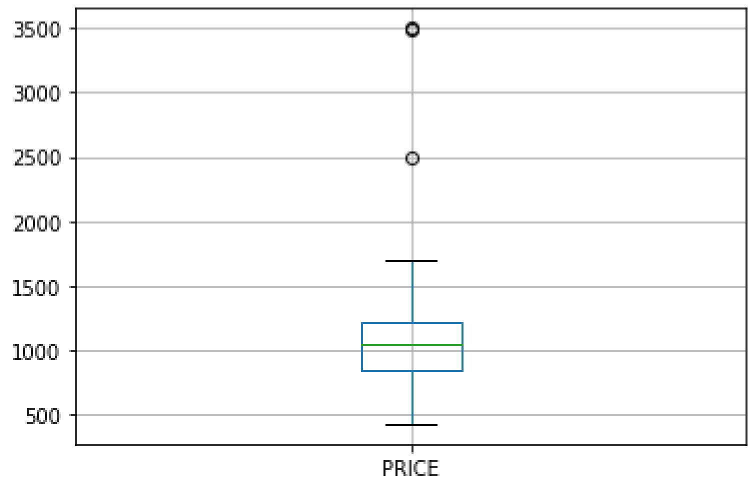
Handling Outlier

> 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=(10, 10), dpi=80)
plt.scatter(df.LIVING_AREA, df.PRICE, c= 'red')
plt.title("Outliers")
plt.xlabel("LivArea")
plt.ylabel("Price")
plt.show()
```


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

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

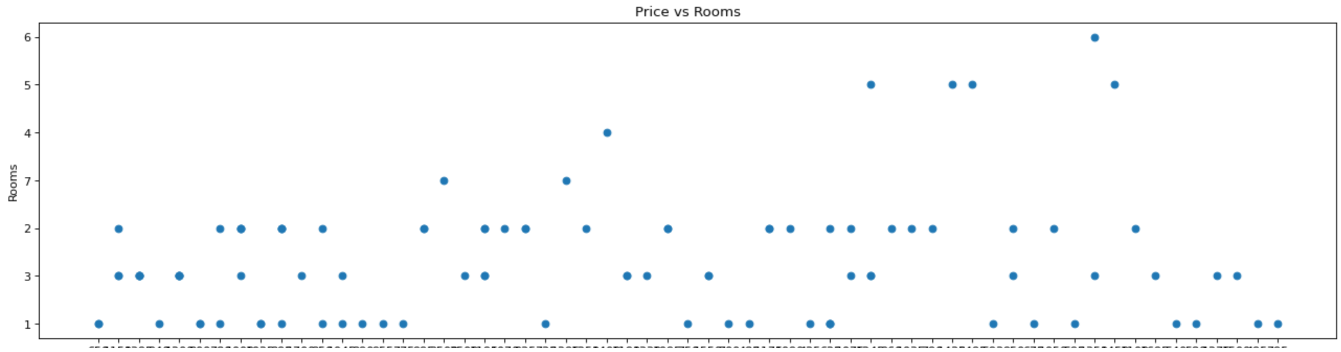


```
sorted(df)
```

```
['LIVING_AREA', 'POSTCODE', 'PRICE', 'ROOMS', 'STREET NAME', 'TYPE']
```

Using scatter plots to visualize the relationship between the variables and the targeted variable - `PRICE`.

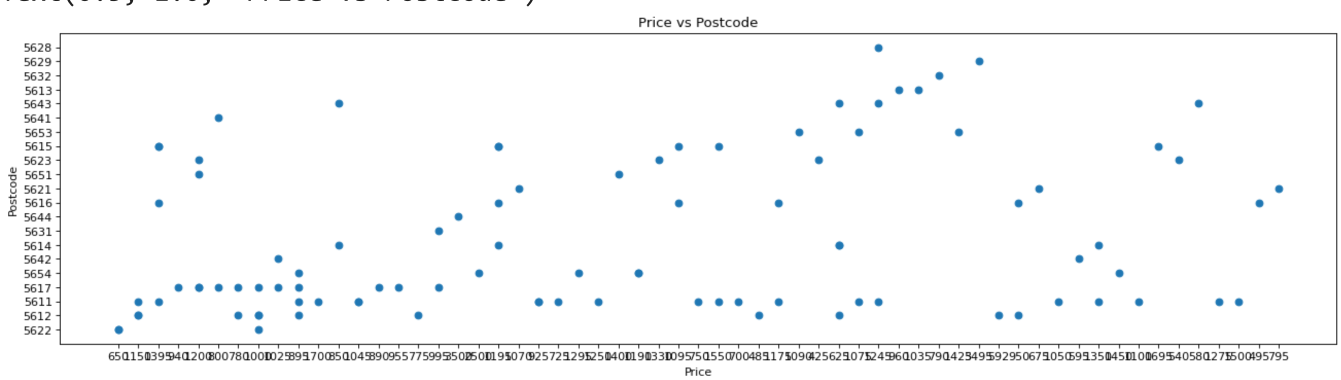
```
plt.figure(figsize=(20, 5), dpi=80)
plt.scatter(df['PRICE'],df['ROOMS'])
plt.title("Price vs Rooms")
plt.xlabel("Price")
plt.ylabel("Rooms")
plt.show()
sns.despine
```



> It can be noticed that there is a positive correlation between the price and the living area, which means that the variables move in tandem—that is, in the same direction. This means that whenever one variable increases, the other decreases. For instance, the price increases with the more rooms the housing has.

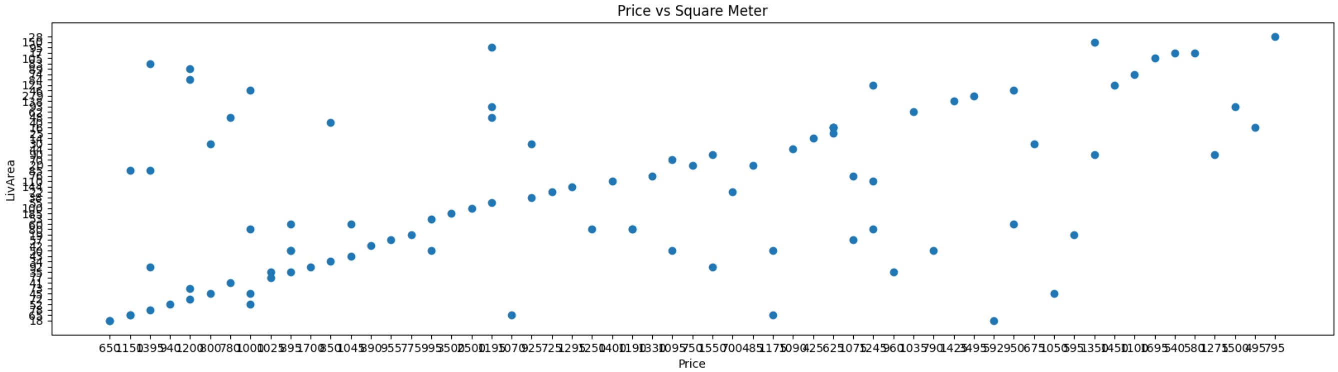
```
plt.figure(figsize=(20, 5), dpi=80)
plt.scatter(df['PRICE'],df['POSTCODE'])
plt.xlabel("Price")
plt.ylabel("Postcode")
plt.title("Price vs Postcode")
```

```
Text(0.5, 1.0, 'Price vs Postcode')
```




```
plt.figure(figsize=(20, 5), dpi=100)
plt.scatter(df['PRICE'],df['LIVING_AREA'])
plt.xlabel("Price")
plt.ylabel("LivArea")
plt.title("Price vs Square Meter")
```

```
Text(0.5, 1.0, 'Price vs Square Meter')
```



> It can be noticed that there is a positive correlation between the price and the living area, which means that the variables move in tandem—that is, in the same direction. This means that whenever one variable increases, the other decreases. For instance, the price increases with the increase in the living area.

```
df['PRICE'] =df['PRICE'].astype(float)
df['ROOMS'] =df['ROOMS'].astype(int)
df['LIVING_AREA'] =df['LIVING_AREA'].astype(int)
```

```
df['ROOMS'].value_counts().plot(kind='bar')
plt.title('Number of Rooms')
plt.xlabel('Rooms')
plt.ylabel('Count')
sns.despine
```

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

```
print(df['PRICE'])
```

```
0      650.0
1     1150.0
2     1395.0
3      940.0
4     1200.0
       ...
27    1195.0
28    1350.0
29    1500.0
30     495.0
31     795.0
Name: PRICE, Length: 96, dtype: float64
```

Rooms

Changing the type of the variable `price` in order to plot it in the next diagram.

```
df['PRICE'] =df['PRICE'].astype(float)
df['POSTCODE'] =df['POSTCODE'].astype(int)
df['LIVING_AREA'] =df['LIVING_AREA'].astype(int)
df['ROOMS'] =df['ROOMS'].astype(int)
code_numeric = {'Apartment': 1, 'Room': 2, 'House': 3}
df ['TYPE'] = df['TYPE'].map(code_numeric)
```

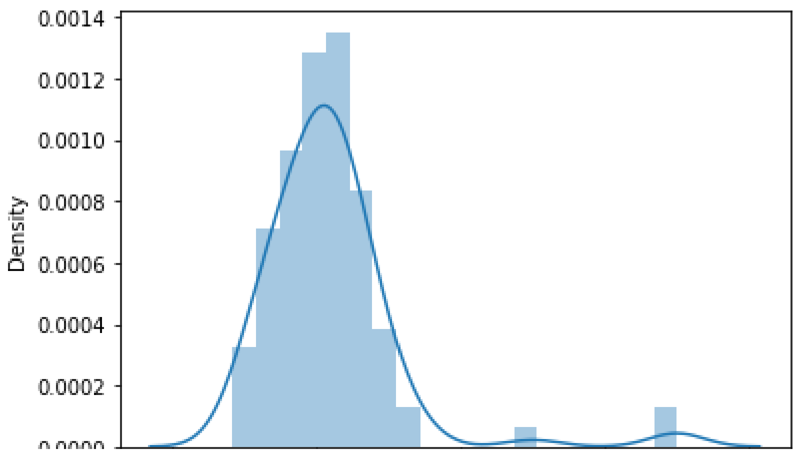
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 96 entries, 0 to 31
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   TYPE         96 non-null     int64
 1   STREET NAME  96 non-null     object
 2   POSTCODE     96 non-null     int64
 3   PRICE        96 non-null     float64
 4   LIVING_AREA  96 non-null     int64
 5   ROOMS        96 non-null     int64
dtypes: float64(1), int64(4), object(1)
memory usage: 5.2+ KB
```

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

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

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7fc1d2221e50>
```



> We can see that the `PRICE` distribution is not skewed, but normally distributed.

> **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.

Reviewing the skewness of each feature

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

```
PRICE          2.935641
LIVING_AREA    2.006517
TYPE           1.960714
ROOMS          1.490395
POSTCODE       1.325435
dtype: float64
```

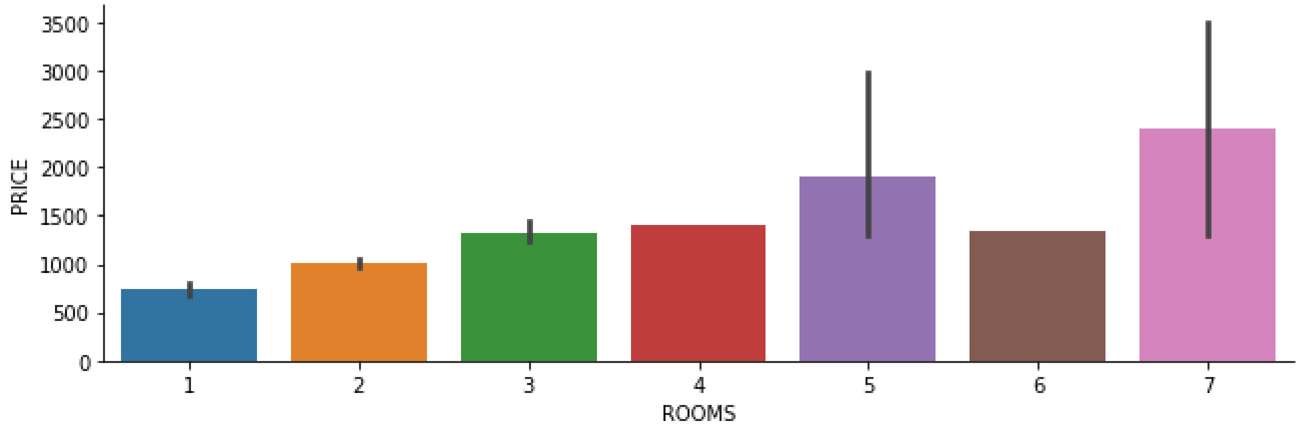
> Values closer to zero are less skewed. The results show some features having a positive (right-tailed) or negative (left-tailed) skew.

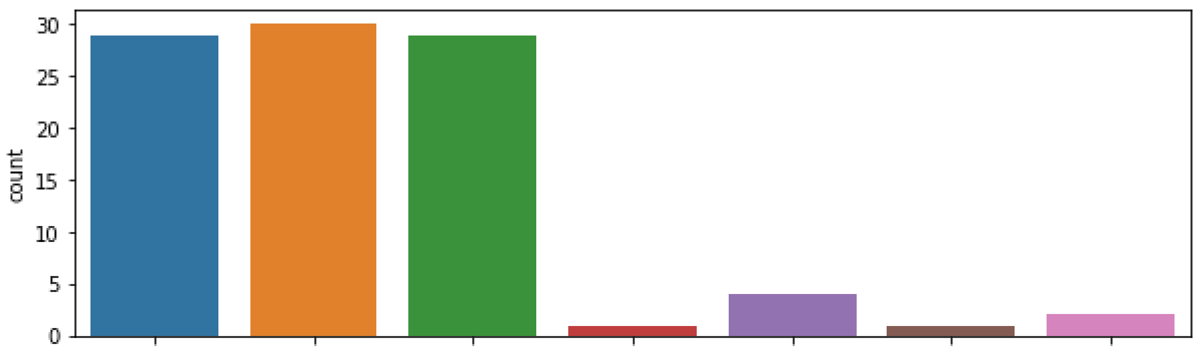
Factor plot is informative when we have multiple groups to compare.

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

```
/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
625.0     4
1000.0    4
1395.0    4
1195.0    4
895.0     4
1200.0    4
1245.0    3
1150.0    3
780.0     2
800.0     2
1350.0    2
850.0     2
995.0     2
925.0     2
1190.0    2
1550.0    2
950.0     2
650.0     2
1025.0    2
1075.0    2
1175.0    2
1095.0    2
1045.0    2
540.0     1
955.0     1
1070.0    1
2500.0    1
3500.0    1
775.0     1
1295.0    1
890.0     1
1250.0    1
1700.0    1
1035.0    1
1425.0    1
940.0     1
1695.0    1
725.0     1
1400.0    1
1100.0    1
795.0     1
1450.0    1
580.0     1
595.0     1
1050.0    1
675.0     1
1500.0    1
592.0     1
790.0     1
```

```
960.0     1
1275.0    1
425.0     1
1090.0    1
485.0     1
700.0     1
495.0     1
750.0     1
1330.0    1
3495.0    1
Name: PRICE, dtype: int64
```





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

```
#g = sns.factorplot(x='POSTCODE', y='Skewed_SP', col='PRICE', data=df, kind='bar', col_wrap=4
sns.factorplot('POSTCODE', 'PRICE', data=df,kind='bar',size=3,aspect=6)
```

```
/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
<seaborn.axisgrid.FacetGrid at 0x7fc1cf753f10>
```

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

Train-Test Split dataset

> Necessary imports

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV
```

Analyzing the numeric features.

```
numeric_features = df.select_dtypes(include=[np.number])
```

```
numeric_features.columns
```

```
Index(['TYPE', 'POSTCODE', 'PRICE', 'LIVING_AREA', 'ROOMS'], dtype='object')
```

```
# set the target and predictors
y = df.PRICE  # target

# use only those input features with numeric data type
df_temp = df.select_dtypes(include=["int64","float64"])

X = df_temp.drop(["PRICE"],axis=1)  # predictors
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
```

Modelling

Linear Regression

```
lr = LinearRegression()
# fit optimal linear regression line on training data
lr.fit((X_train),y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
#predict y_values using X_test set
yr_hat = lr.predict(X_test)
```

```
lr_score =lr.score((X_test),y_test)
print("Accuracy: ", lr_score)
```

```
Accuracy:  0.48155054870991254
```

Using cross-validation to see whether the model is over-fitting the data.

```
# cross validation to find 'validate' score across multiple samples, automatically does Kfold
lr_cv = cross_val_score(lr, X, y, cv = 5, scoring= 'r2')
print("Cross-validation results: ", lr_cv)
print("R2: ", lr_cv.mean())
```

```
Cross-validation results:  [0.48314093 0.4732556  0.80807253 0.87060718 0.60752515]
R2:  0.6485202800858794
```

Random Forest

```
rfr = RandomForestRegressor()
rfr.fit(X_train, y_train) # gets the parameters for the rfr model
rfr_cv = cross_val_score(rfr,X, y, cv = 5, scoring = 'r2')
print("R2: ", rfr_cv.mean())
```

```
R2:  0.649708787686466
```

```
rfr.feature_importances_
```

```
array([0.0016265 , 0.02561403, 0.8863328 , 0.08642667])
```

Plotting the Feature Importance

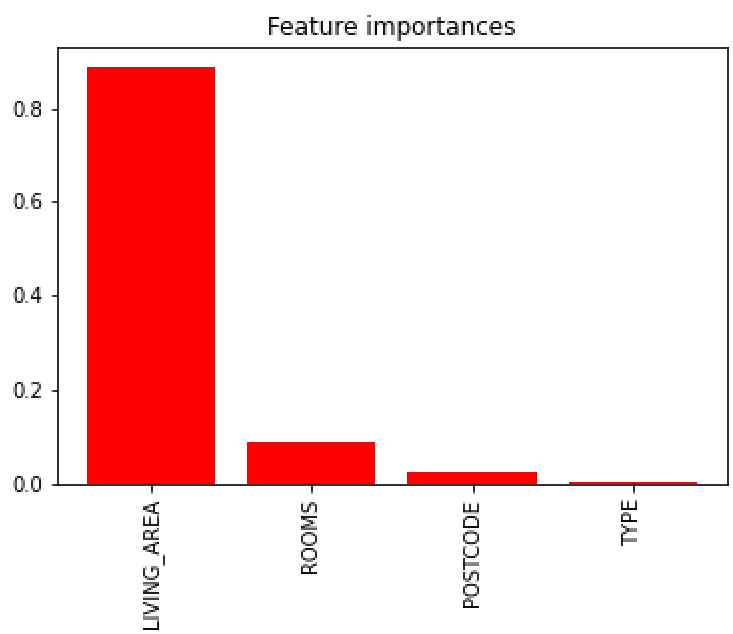
```
importance = rfr.feature_importances_

# map feature importance values to the features
feature_importances = zip(importance, X.columns)
```

```
feature_importances = zip(importance, X.columns)

#list(feature_importances)
sorted_feature_importances = sorted(feature_importances, reverse = True)

#print(sorted_feature_importances)
top_15_predictors = sorted_feature_importances[0:15]
values = [value for value, predictors in top_15_predictors]
predictors = [predictors for value, predictors in top_15_predictors]
print(predictors)
```

```
['LIVING_AREA', 'ROOMS', 'POSTCODE', 'TYPE']
```

Plotting the feauture importance of the Random forest.

```
plt.figure()
plt.title("Feature importances")
plt.bar(range(len(predictors)), values,color="r", align="center");
plt.xticks(range(len(predictors)), predictors, rotation=90);
```



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, make more accurate summaries.

Data preprocessing:

I tried different types of data transfoms to expose the data structure better, so we may be able to improve model accuracy later.

- `Standardizing` was made to the data set so as to reduce the effects of differing distributions.
- `The skewness` of the feautures 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

Modelling:

I used two models to determine the accuracy - Linear Regression and Random Forest.

Linear Regression turns out to be the more accurate model for predicting the house price. It scored an estimated accuracy of 75%, out performing the Random Forest. Random Forest determined that overall the living area of a home is by far the most important predictor. Following are the size of above rooms and postcode.