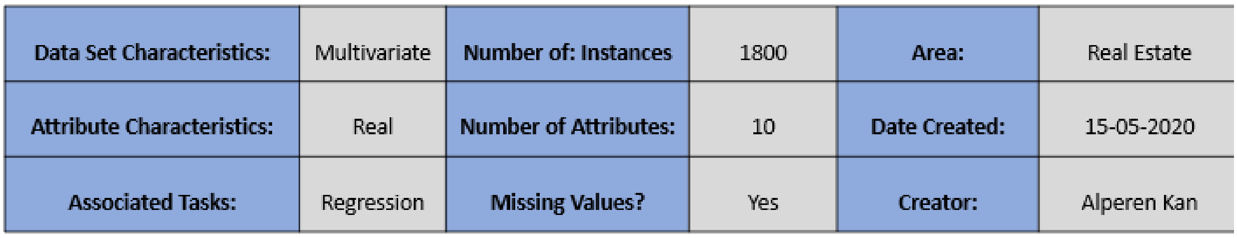
1. Dataset Used:



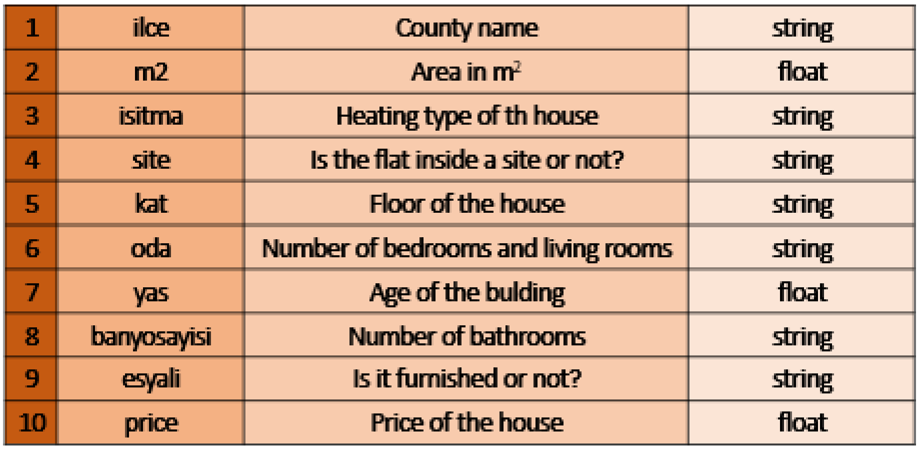
Data Set Characteristics:	Multivariate	Number of: Instances	1800	Area:	Real Estate
Attribute Characteristics:	Real	Number of Attributes:	10	Date Created:	15-05-2020
Associated Tasks:	Regression	Missing Values?	Yes	Creator:	Alperen Kan

I created this dataset with using Python, Selenium and Beautiful Soap. Automation system takes the county list and a website url as an input and then it goes to emlakjet.com real estate website and takes desired attributes of the house. Automation program can print 60 house data per district to a csv file in one run.

The time efficiency of the program is not good enough for now.(For 1800 house data; it takes approximately 100 minutes.)

Data in the dataset are actual values on 15 May 2020.

Attribute Information:

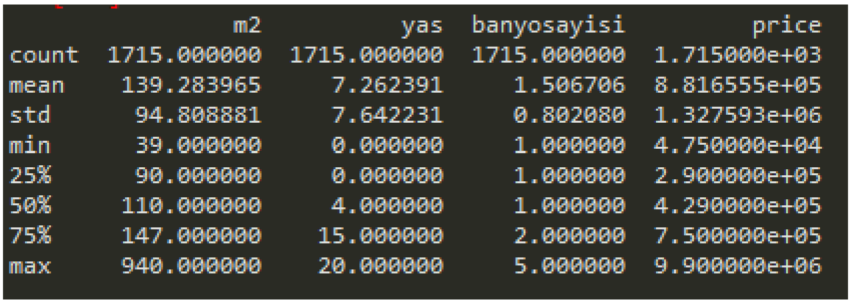


1	ilce	County name	string
2	m2	Area in m^2	float
3	isitma	Heating type of th house	string
4	site	Is the flat inside a site or not?	string
5	kat	Floor of the house	string
6	oda	Number of bedrooms and living rooms	string
7	yas	Age of the bulding	float
8	banyosayisi	Number of bathrooms	string
9	esyali	Is it furnished or not?	string
10	price	Price of the house	float

Automation outputs keeped in seperate csv files (seperated by counties) and then merged into one csv file for easier use.

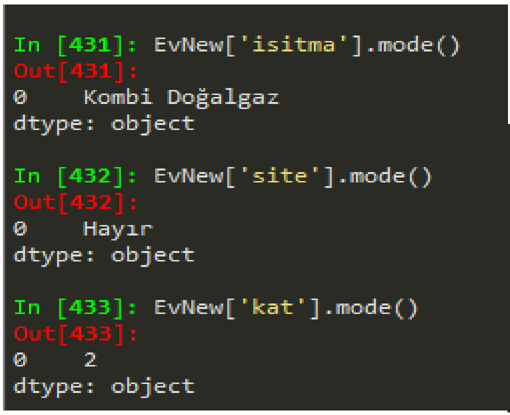
2. Data Visualization:

2.1 Description of the numerical attributes:

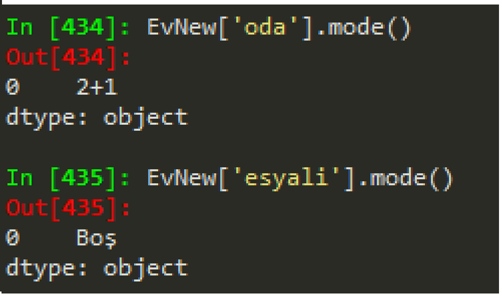


```
                m2          yas  banyosayisi         price
count  1715.000000  1715.000000  1715.000000  1.715000e+03
mean    139.283965     7.262391     1.506706  8.816555e+05
std      94.808881     7.642231     0.802080  1.327593e+06
min      39.000000     0.000000     1.000000  4.750000e+04
25%      90.000000     0.000000     1.000000  2.900000e+05
50%     110.000000     4.000000     1.000000  4.290000e+05
75%     147.000000    15.000000     2.000000  7.500000e+05
max     940.000000    20.000000     5.000000  9.900000e+06
```

2.2 Most frequent values of categorical attributes:

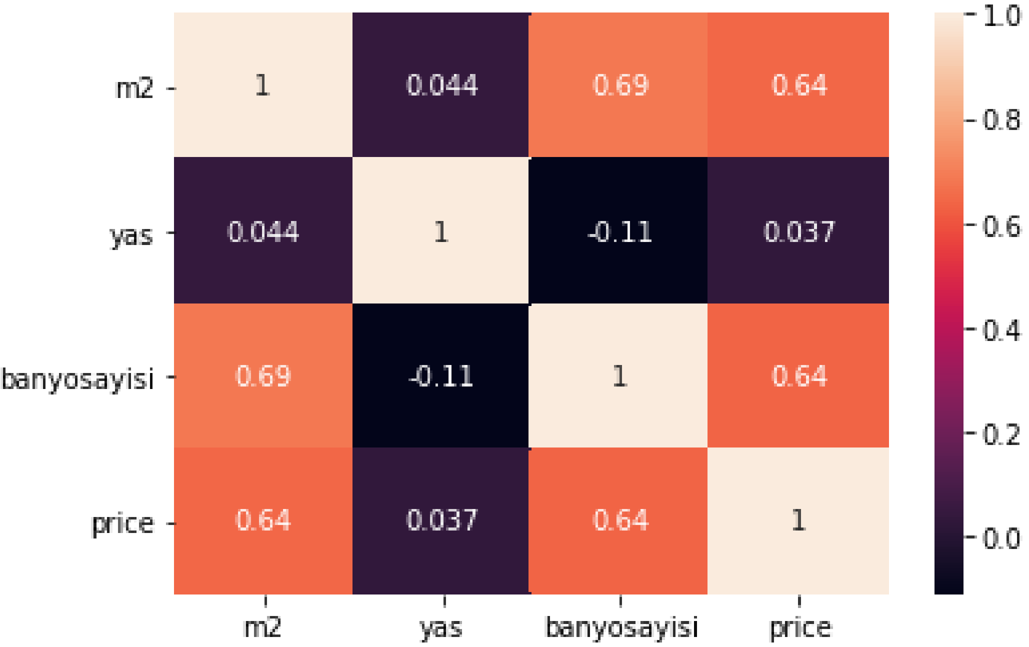


```
In [431]: EvNew['isitma'].mode()
Out[431]:
0    Kombi Doğalgaz
dtype: object

In [432]: EvNew['site'].mode()
Out[432]:
0    Hayır
dtype: object

In [433]: EvNew['kat'].mode()
Out[433]:
0    2
dtype: object
```

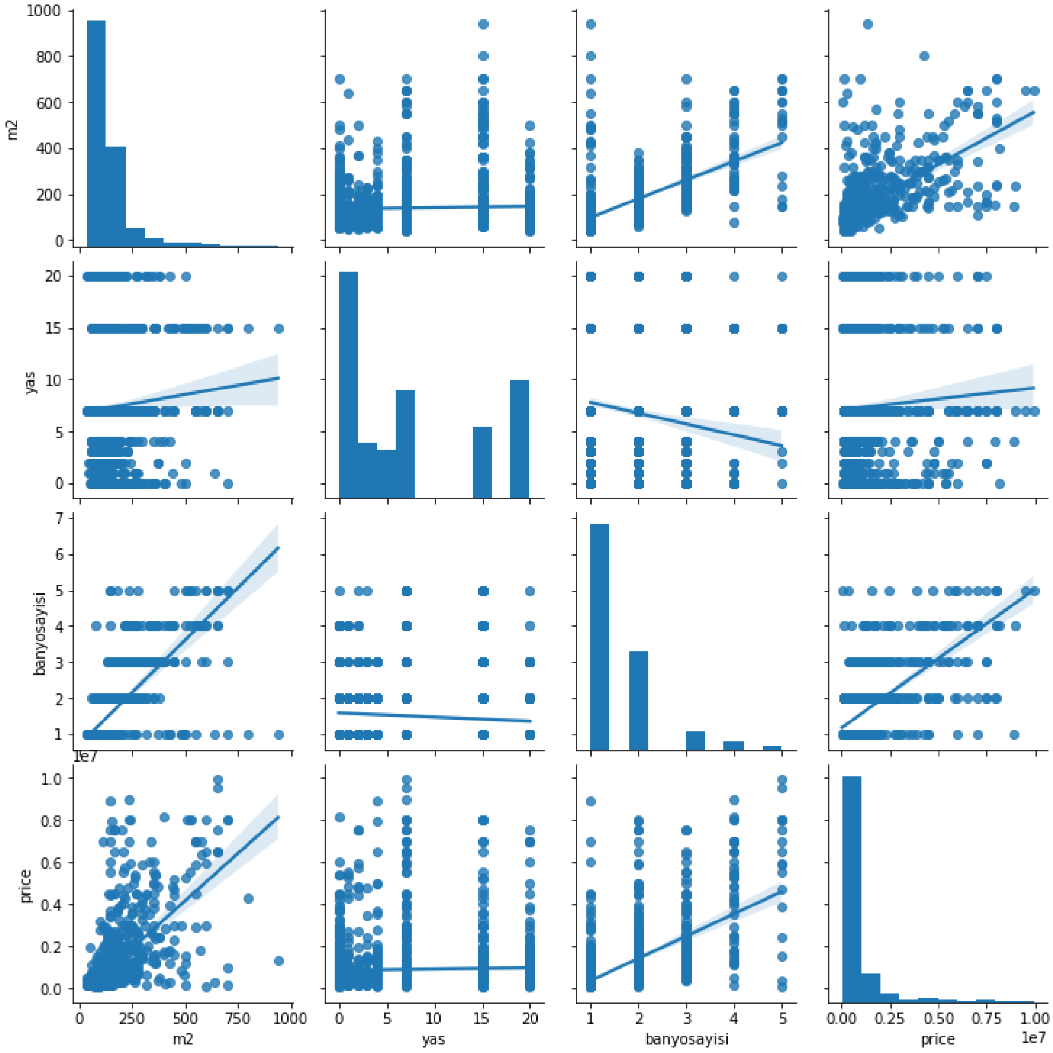


```
In [434]: EvNew['oda'].mode()
Out[434]:
0    2+1
dtype: object

In [435]: EvNew['esyali'].mode()
Out[435]:
0    Boş
dtype: object
```

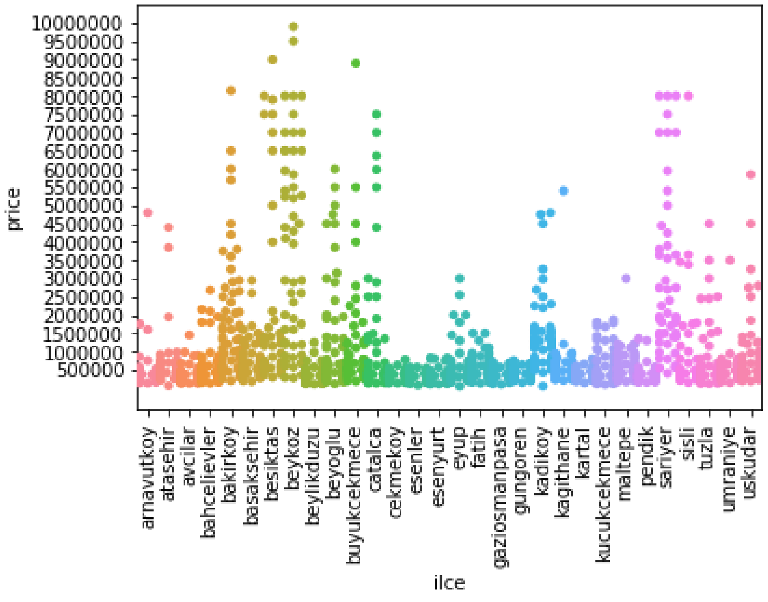
2.3 Heatmap of the numerical attributes:



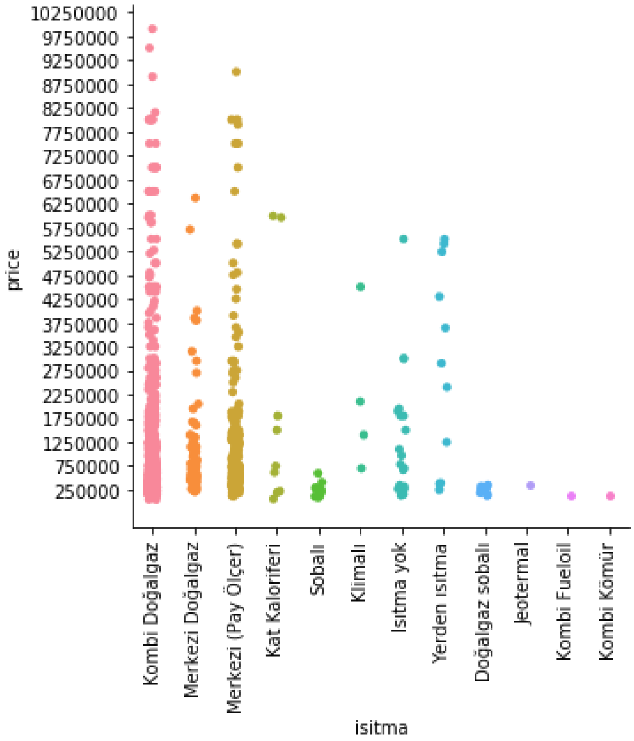
2.4 Pairplot of the numerical attributes:



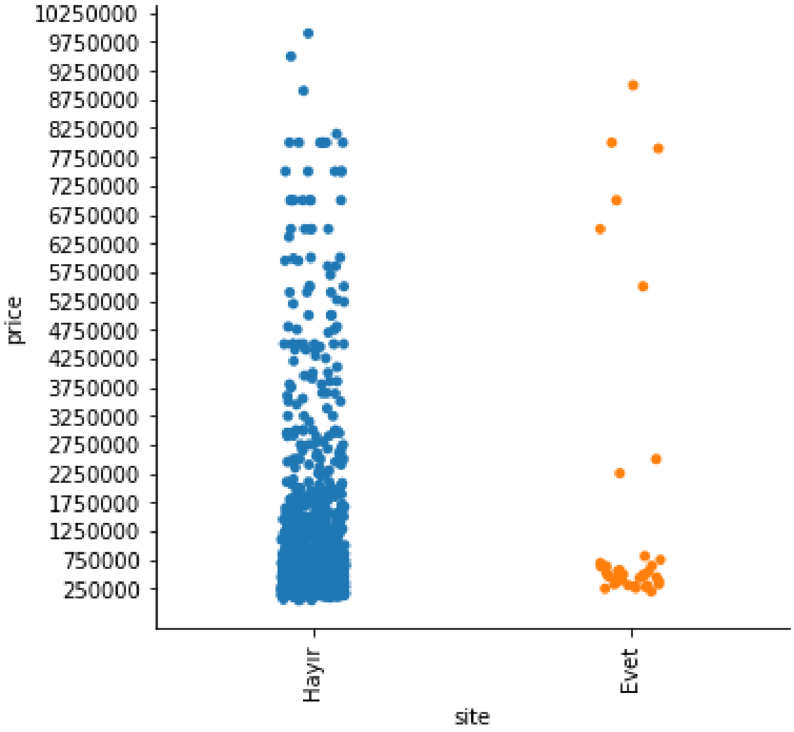
2.5 Price vs County:



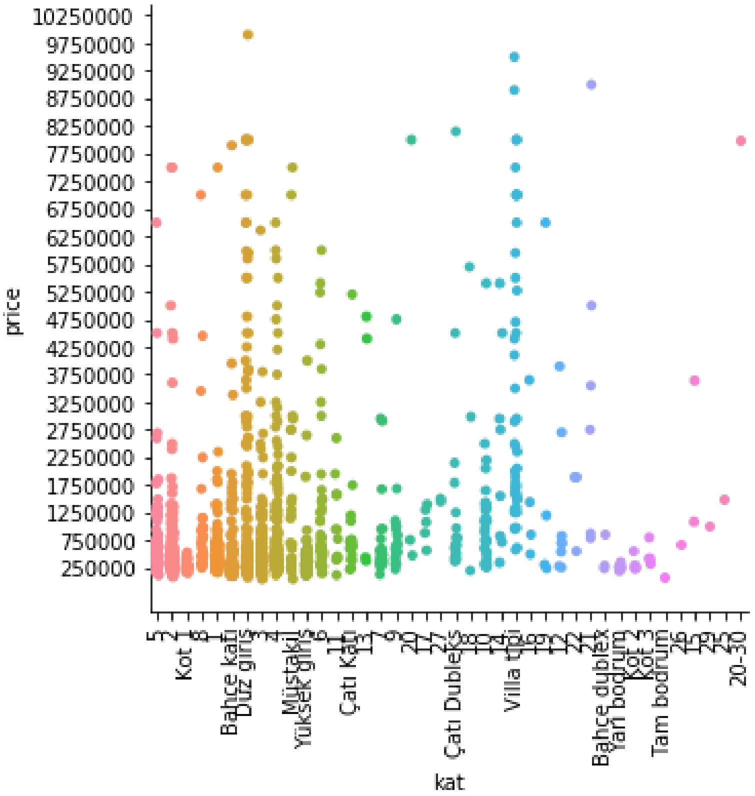
2.6 Price vs Heating Type:



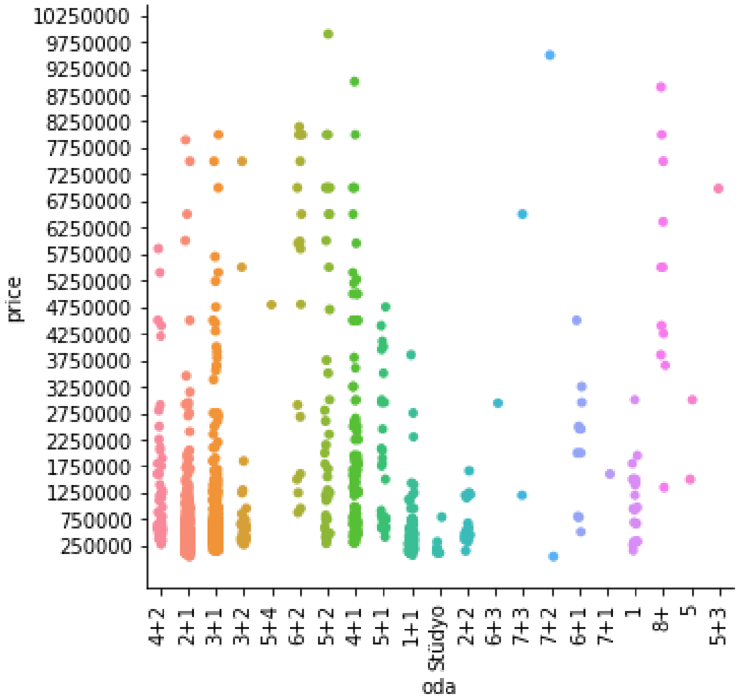
2.7 Price vs In Site or Not:



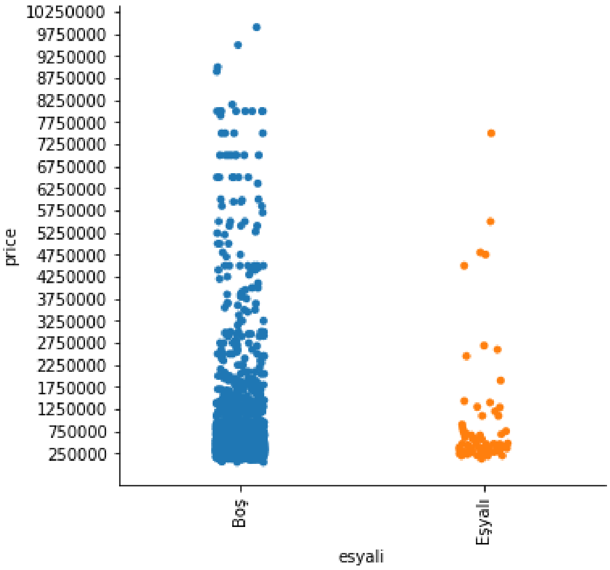
2.8 Price vs Floor:



2.9 Price vs #of Bedrooms:



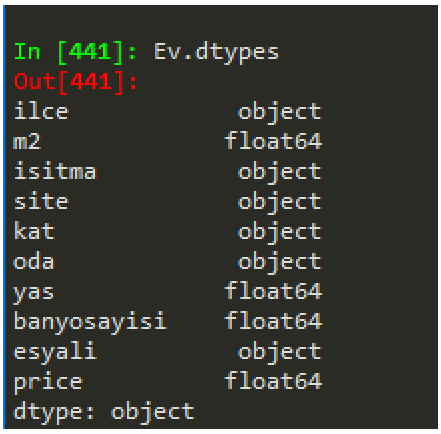
2.10 Price vs Furnished or not:



3. Data Preprocessing:

3.1 Data Formatting:

Automation's output csv has 10 columns but the data types and their formats were not in intended type.



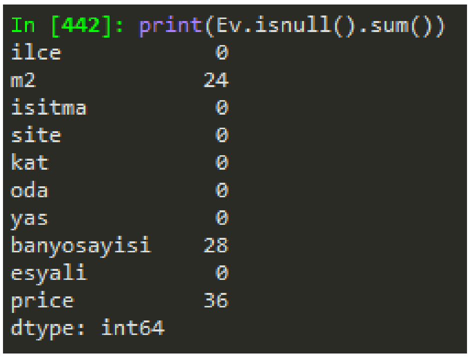
```
In [441]: Ev.dtypes
Out[441]:
ilce             object
m2              float64
isitma           object
site             object
kat              object
oda              object
yas             float64
banyosayisi     float64
esyali           object
price           float64
dtype: object
```

Categorical variables are represented as 'object' and this is fine for me. But numerical values represented as 'float64'. So I convert them to int32 and int64 and also the 'price ' attribute turns to scientific notation (like 1.43324 +7e) after 10.000.000. This situation solved after conversion to int64.

The first version is like this.

3.2 Data Cleaning:

In the begining the missing-value counts of the numerical variables can be seen from the figure.



```
In [442]: print(Ev.isnull().sum())
ilce            0
m2             24
isitma          0
site            0
kat             0
oda             0
yas             0
banyosayisi    28
esyali          0
price          36
dtype: int64
```

'm2' were filled with 'mean'

'price' were filled with 'median'

'banyosayisi' were filled with 'most_frequent'

'esyali' has a lot of 'Belirtilmemiş' value so also this is were filled with 'most_frequent'

4. Data Transformation:

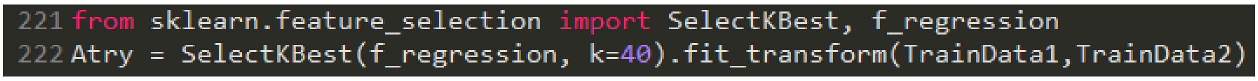
4.1 Scaling:

Firstly I transformed the numerical values with StandardScaler but while performing the model and also while visualizing the data it gave bad and corrupted results so finally I used MinMaxScaler for numerical attributes.

For categorical attributes I used LabelEncoder and this is also gave me a bad results and I understood that while encoding with LabelEncoder it transforms the data to an ordinal value. So I changed this method with OHE(One Hot Encoder).

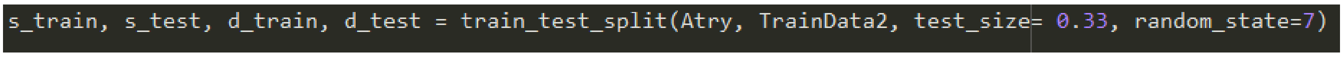
5. Feature Selection – Extraction:

After data transformation my dataframe's shape become (1715,110). So there are 100+ columns. I used SelectKBest and f_regression fort his part and I selected 40 best.



```
from sklearn.feature_selection import SelectKBest, f_regression
Atry = SelectKBest(f_regression, k=40).fit_transform(TrainData1,TrainData2)
```

And for splitting the data into 'train' and 'test' sets; I used 66% of the data for training and 33% for tesing. And also I added a random_state '7' because the dataset is listed like; first 30 rows county A, second 30 rows county B, …

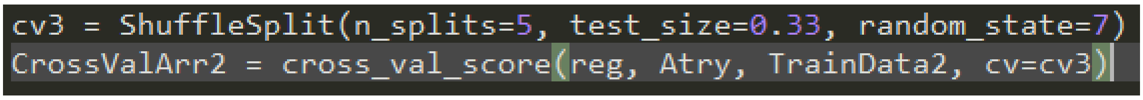


```
s_train, s_test, d_train, d_test = train_test_split(Atry, TrainData2, test_size= 0.33, random_state=7)
```

6. Dataset Splitting:

6.1 Cross Validation Technique:

I used ShuffleSplit and cross_val_score from sklearn for cross validation.



```
cv3 = ShuffleSplit(n_splits=5, test_size=0.33, random_state=7)
CrossValArr2 = cross_val_score(reg, Atry, TrainData2, cv=cv3)
```

7. Modelling:

7.1 Model Training:

I used 2 different models fort his problem, first I used Multi Linear Regression and second I used Random Forest.

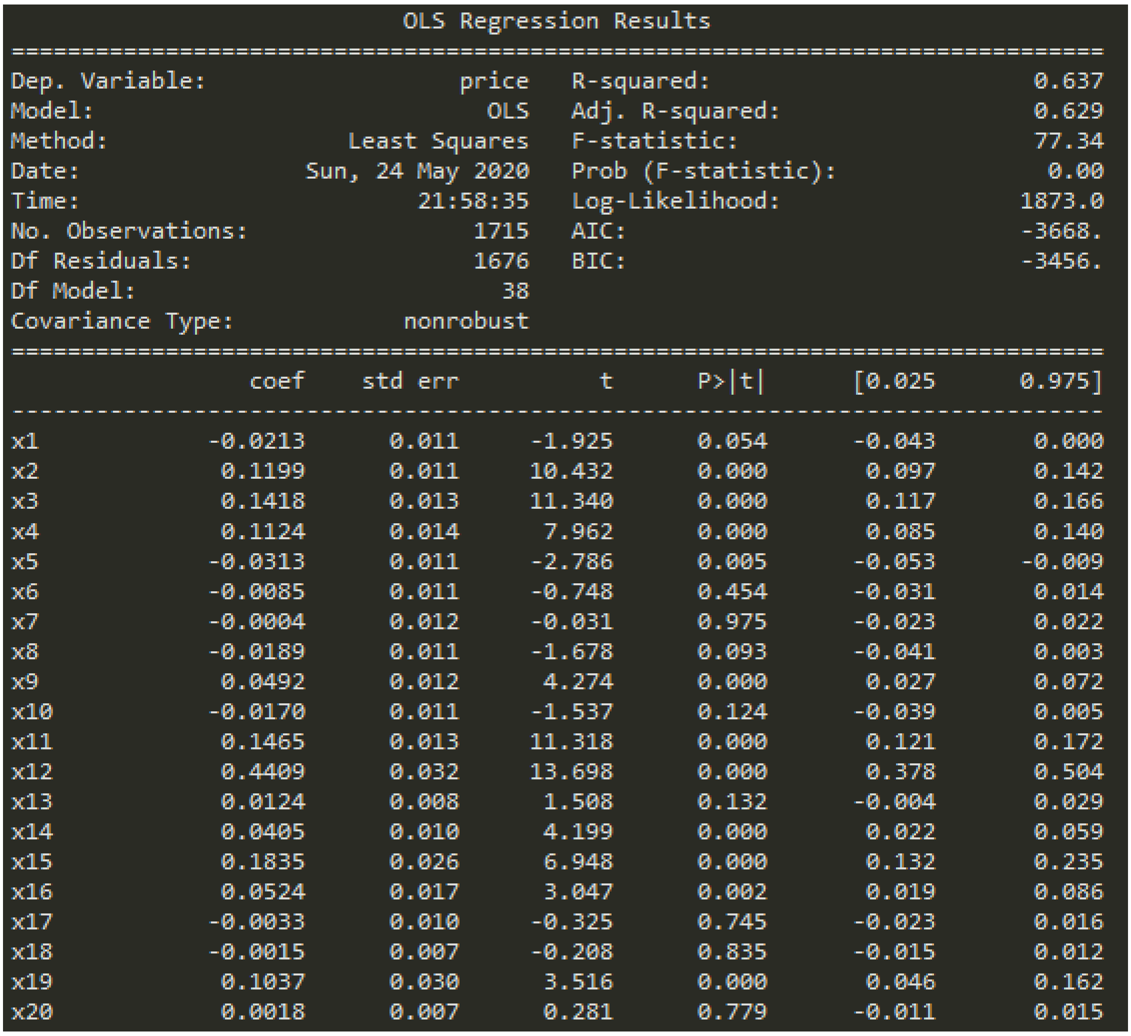
7.2 Model Evaluation and Testing:

7.2.1 Multi Linear Regression results:

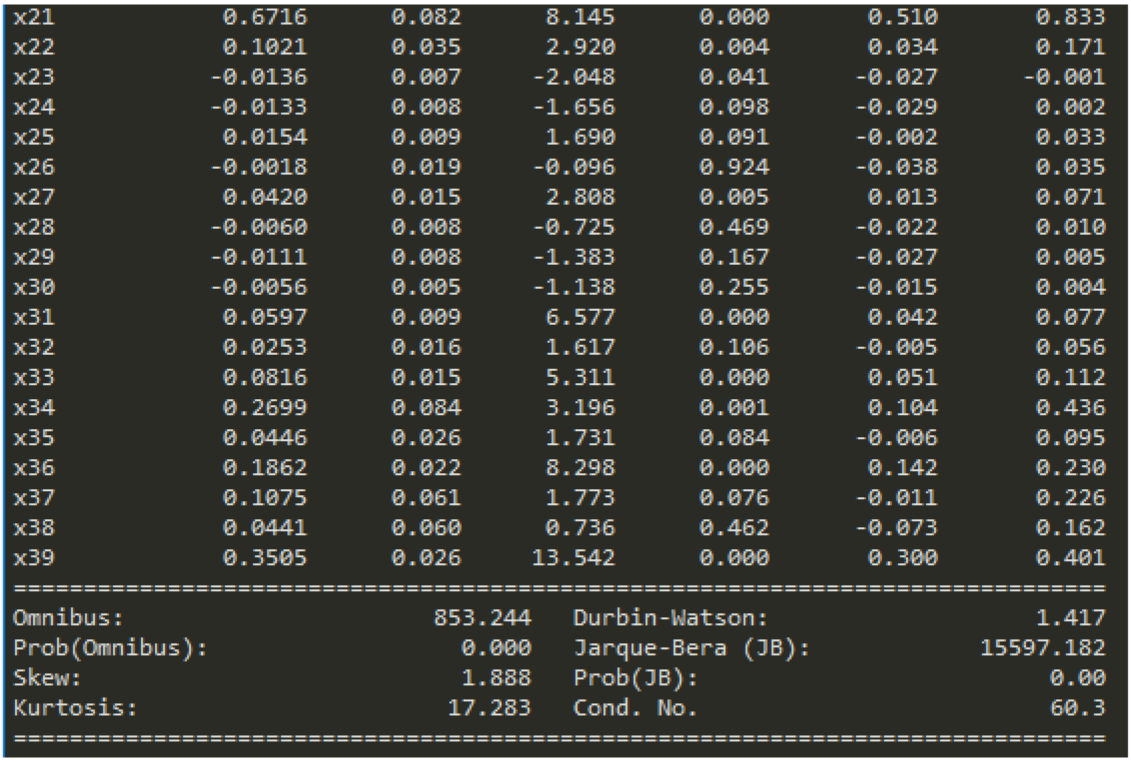
7.2.1.a R^2 score:

R2 = 0.691778845143526

7.2.1.b OLS Summary:

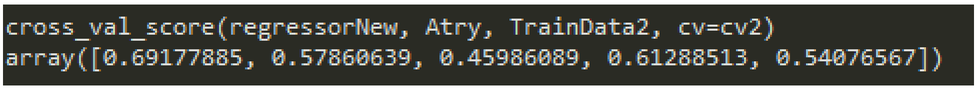


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  price   R-squared:                       0.637
Model:                            OLS   Adj. R-squared:                  0.629
Method:                 Least Squares   F-statistic:                     77.34
Date:                Sun, 24 May 2020   Prob (F-statistic):               0.00
Time:                        21:58:35   Log-Likelihood:                 1873.0
No. Observations:                1715   AIC:                            -3668.
Df Residuals:                    1676   BIC:                            -3456.
Df Model:                          38
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x1            -0.0213      0.011     -1.925      0.054      -0.043       0.000
x2             0.1199      0.011     10.432      0.000       0.097       0.142
x3             0.1418      0.013     11.340      0.000       0.117       0.166
x4             0.1124      0.014      7.962      0.000       0.085       0.140
x5            -0.0313      0.011     -2.786      0.005      -0.053      -0.009
x6            -0.0085      0.011     -0.748      0.454      -0.031       0.014
x7            -0.0004      0.012     -0.031      0.975      -0.023       0.022
x8            -0.0189      0.011     -1.678      0.093      -0.041       0.003
x9             0.0492      0.012      4.274      0.000       0.027       0.072
x10           -0.0170      0.011     -1.537      0.124      -0.039       0.005
x11            0.1465      0.013     11.318      0.000       0.121       0.172
x12            0.4409      0.032     13.698      0.000       0.378       0.504
x13            0.0124      0.008      1.508      0.132      -0.004       0.029
x14            0.0405      0.010      4.199      0.000       0.022       0.059
x15            0.1835      0.026      6.948      0.000       0.132       0.235
x16            0.0524      0.017      3.047      0.002       0.019       0.086
x17           -0.0033      0.010     -0.325      0.745      -0.023       0.016
x18           -0.0015      0.007     -0.208      0.835      -0.015       0.012
x19            0.1037      0.030      3.516      0.000       0.046       0.162
x20            0.0018      0.007      0.281      0.779      -0.011       0.015
```



```
x21            0.6716      0.082      8.145      0.000       0.510       0.833
x22            0.1021      0.035      2.920      0.004       0.034       0.171
x23           -0.0136      0.007     -2.048      0.041      -0.027      -0.001
x24           -0.0133      0.008     -1.656      0.098      -0.029       0.002
x25            0.0154      0.009      1.690      0.091      -0.002       0.033
x26           -0.0018      0.019     -0.096      0.924      -0.038       0.035
x27            0.0420      0.015      2.808      0.005       0.013       0.071
x28           -0.0060      0.008     -0.725      0.469      -0.022       0.010
x29           -0.0111      0.008     -1.383      0.167      -0.027       0.005
x30           -0.0056      0.005     -1.138      0.255      -0.015       0.004
x31            0.0597      0.009      6.577      0.000       0.042       0.077
x32            0.0253      0.016      1.617      0.106      -0.005       0.056
x33            0.0816      0.015      5.311      0.000       0.051       0.112
x34            0.2699      0.084      3.196      0.001       0.104       0.436
x35            0.0446      0.026      1.731      0.084      -0.006       0.095
x36            0.1862      0.022      8.298      0.000       0.142       0.230
x37            0.1075      0.061      1.773      0.076      -0.011       0.226
x38            0.0441      0.060      0.736      0.462      -0.073       0.162
x39            0.3505      0.026     13.542      0.000       0.300       0.401
==============================================================================
Omnibus:                      853.244   Durbin-Watson:                   1.417
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            15597.182
Skew:                           1.888   Prob(JB):                         0.00
Kurtosis:                      17.283   Cond. No.                         60.3
==============================================================================
```

7.2.1.c Cross Validation Score:



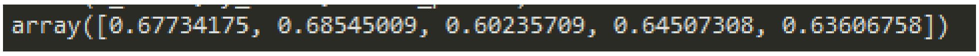
```
cross_val_score(regressorNew, Atry, TrainData2, cv=cv2)
array([0.69177885, 0.57860639, 0.45986089, 0.61288513, 0.54076567])
```

7.2.2 Random Forest:

7.2.2.a R^2 score:

R2 = 0.6773417459977902

7.2.2.b Cross Validation Score:

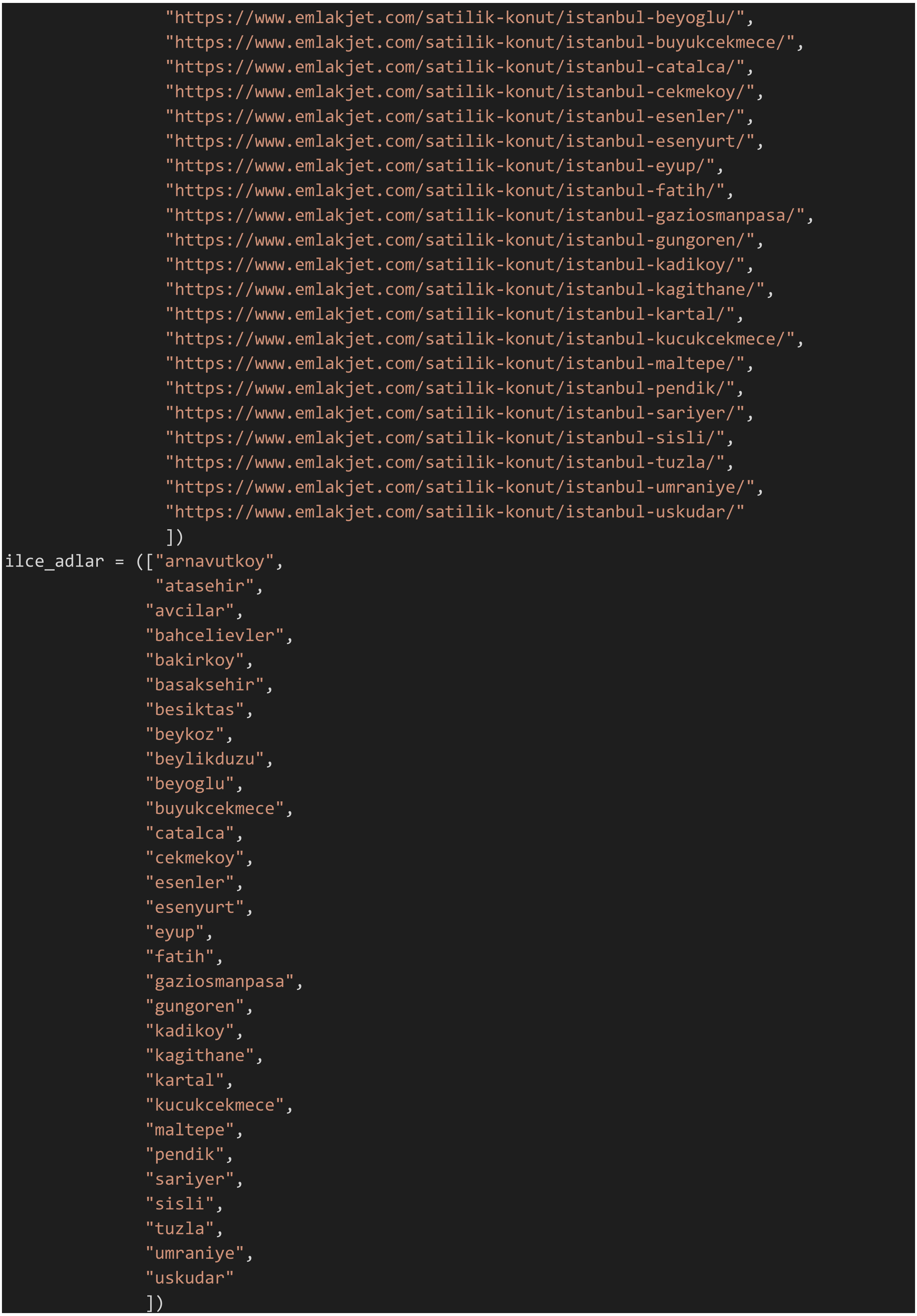


```
array([0.67734175, 0.68545009, 0.60235709, 0.64507308, 0.63606758])
```

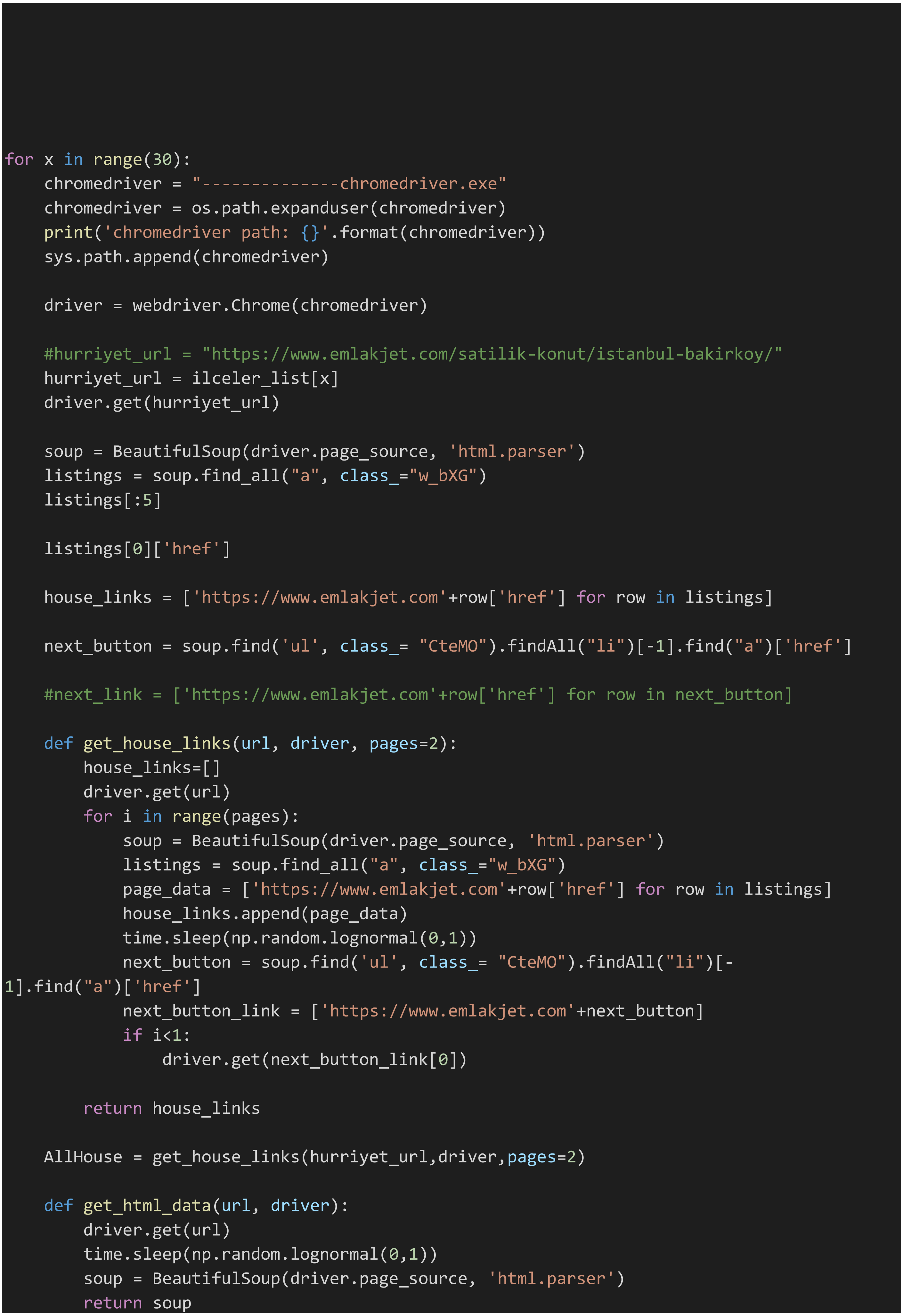
APPENDIX

A. Dataset Creation Automation Code:

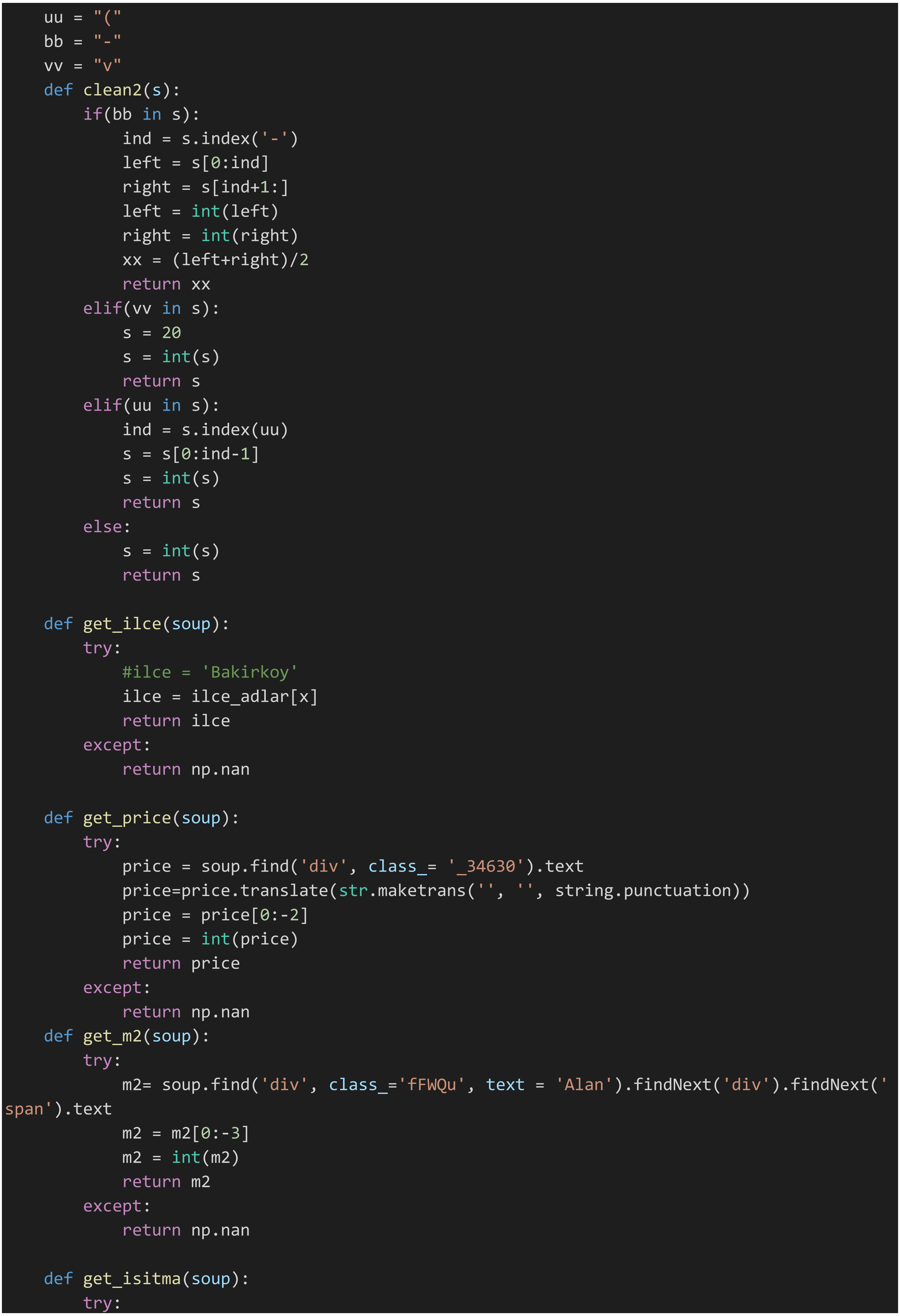
```
# encoding:utf-8
"""
Created on Wed Apr 29 12:45:22 2020

@author: alperen kan
"""
import os
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
import time
import sys
import numpy as np
import pandas as pd
import regex as re
import string
tick = time.clock()


ilceler_list = (["https://www.emlakjet.com/satilik-konut/istanbul-arnavutkoy/",
                "https://www.emlakjet.com/satilik-konut/istanbul-atasehir/",
                "https://www.emlakjet.com/satilik-konut/istanbul-avcilar/",
                "https://www.emlakjet.com/satilik-konut/istanbul-bahcelievler/",
                "https://www.emlakjet.com/satilik-konut/istanbul-bakirkoy/",
                "https://www.emlakjet.com/satilik-konut/istanbul-basaksehir/",
                "https://www.emlakjet.com/satilik-konut/istanbul-besiktas/",
                "https://www.emlakjet.com/satilik-konut/istanbul-beykoz/",
                "https://www.emlakjet.com/satilik-konut/istanbul-beylikduzu/",
```



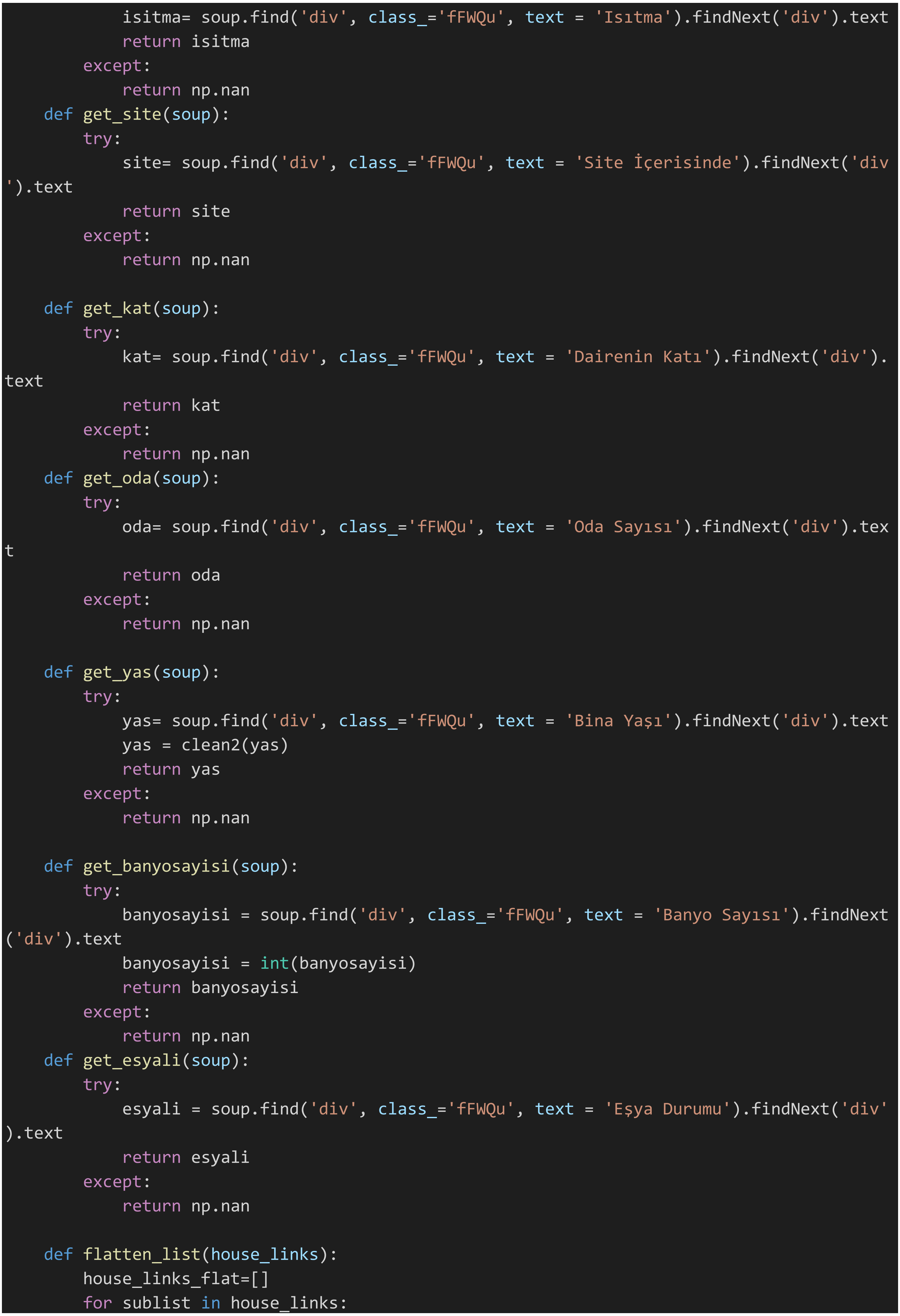
```
                "https://www.emlakjet.com/satilik-konut/istanbul-beyoglu/",
                "https://www.emlakjet.com/satilik-konut/istanbul-buyukcekmece/",
                "https://www.emlakjet.com/satilik-konut/istanbul-catalca/",
                "https://www.emlakjet.com/satilik-konut/istanbul-cekmekoy/",
                "https://www.emlakjet.com/satilik-konut/istanbul-esenler/",
                "https://www.emlakjet.com/satilik-konut/istanbul-esenyurt/",
                "https://www.emlakjet.com/satilik-konut/istanbul-eyup/",
                "https://www.emlakjet.com/satilik-konut/istanbul-fatih/",
                "https://www.emlakjet.com/satilik-konut/istanbul-gaziosmanpasa/",
                "https://www.emlakjet.com/satilik-konut/istanbul-gungoren/",
                "https://www.emlakjet.com/satilik-konut/istanbul-kadikoy/",
                "https://www.emlakjet.com/satilik-konut/istanbul-kagithane/",
                "https://www.emlakjet.com/satilik-konut/istanbul-kartal/",
                "https://www.emlakjet.com/satilik-konut/istanbul-kucukcekmece/",
                "https://www.emlakjet.com/satilik-konut/istanbul-maltepe/",
                "https://www.emlakjet.com/satilik-konut/istanbul-pendik/",
                "https://www.emlakjet.com/satilik-konut/istanbul-sariyer/",
                "https://www.emlakjet.com/satilik-konut/istanbul-sisli/",
                "https://www.emlakjet.com/satilik-konut/istanbul-tuzla/",
                "https://www.emlakjet.com/satilik-konut/istanbul-umraniye/",
                "https://www.emlakjet.com/satilik-konut/istanbul-uskudar/"
                ])
ilce_adlar = (["arnavutkoy",
               "atasehir",
              "avcilar",
              "bahcelievler",
              "bakirkoy",
              "basaksehir",
              "besiktas",
              "beykoz",
              "beylikduzu",
              "beyoglu",
              "buyukcekmece",
              "catalca",
              "cekmekoy",
              "esenler",
              "esenyurt",
              "eyup",
              "fatih",
              "gaziosmanpasa",
              "gungoren",
              "kadikoy",
              "kagithane",
              "kartal",
              "kucukcekmece",
              "maltepe",
              "pendik",
              "sariyer",
              "sisli",
              "tuzla",
              "umraniye",
              "uskudar"
              ])
```



```
for x in range(30):
    chromedriver = "--------------chromedriver.exe"
    chromedriver = os.path.expanduser(chromedriver)
    print('chromedriver path: {}'.format(chromedriver))
    sys.path.append(chromedriver)

    driver = webdriver.Chrome(chromedriver)

    #hurriyet_url = "https://www.emlakjet.com/satilik-konut/istanbul-bakirkoy/"
    hurriyet_url = ilceler_list[x]
    driver.get(hurriyet_url)

    soup = BeautifulSoup(driver.page_source, 'html.parser')
    listings = soup.find_all("a", class_="w_bXG")
    listings[:5]

    listings[0]['href']

    house_links = ['https://www.emlakjet.com'+row['href'] for row in listings]

    next_button = soup.find('ul', class_= "CteMO").findAll("li")[-1].find("a")['href']

    #next_link = ['https://www.emlakjet.com'+row['href'] for row in next_button]

    def get_house_links(url, driver, pages=2):
        house_links=[]
        driver.get(url)
        for i in range(pages):
            soup = BeautifulSoup(driver.page_source, 'html.parser')
            listings = soup.find_all("a", class_="w_bXG")
            page_data = ['https://www.emlakjet.com'+row['href'] for row in listings]
            house_links.append(page_data)
            time.sleep(np.random.lognormal(0,1))
            next_button = soup.find('ul', class_= "CteMO").findAll("li")[-
1].find("a")['href']
            next_button_link = ['https://www.emlakjet.com'+next_button]
            if i<1:
                driver.get(next_button_link[0])

        return house_links

    AllHouse = get_house_links(hurriyet_url,driver,pages=2)

    def get_html_data(url, driver):
        driver.get(url)
        time.sleep(np.random.lognormal(0,1))
        soup = BeautifulSoup(driver.page_source, 'html.parser')
        return soup
```



```
    uu = "("
    bb = "-"
    vv = "v"
    def clean2(s):
        if(bb in s):
            ind = s.index('-')
            left = s[0:ind]
            right = s[ind+1:]
            left = int(left)
            right = int(right)
            xx = (left+right)/2
            return xx
        elif(vv in s):
            s = 20
            s = int(s)
            return s
        elif(uu in s):
            ind = s.index(uu)
            s = s[0:ind-1]
            s = int(s)
            return s
        else:
            s = int(s)
            return s

    def get_ilce(soup):
        try:
            #ilce = 'Bakirkoy'
            ilce = ilce_adlar[x]
            return ilce
        except:
            return np.nan

    def get_price(soup):
        try:
            price = soup.find('div', class_= '_34630').text
            price=price.translate(str.maketrans('', '', string.punctuation))
            price = price[0:-2]
            price = int(price)
            return price
        except:
            return np.nan
    def get_m2(soup):
        try:
            m2= soup.find('div', class_='fFWQu', text = 'Alan').findNext('div').findNext('
span').text
            m2 = m2[0:-3]
            m2 = int(m2)
            return m2
        except:
            return np.nan

    def get_isitma(soup):
        try:
```

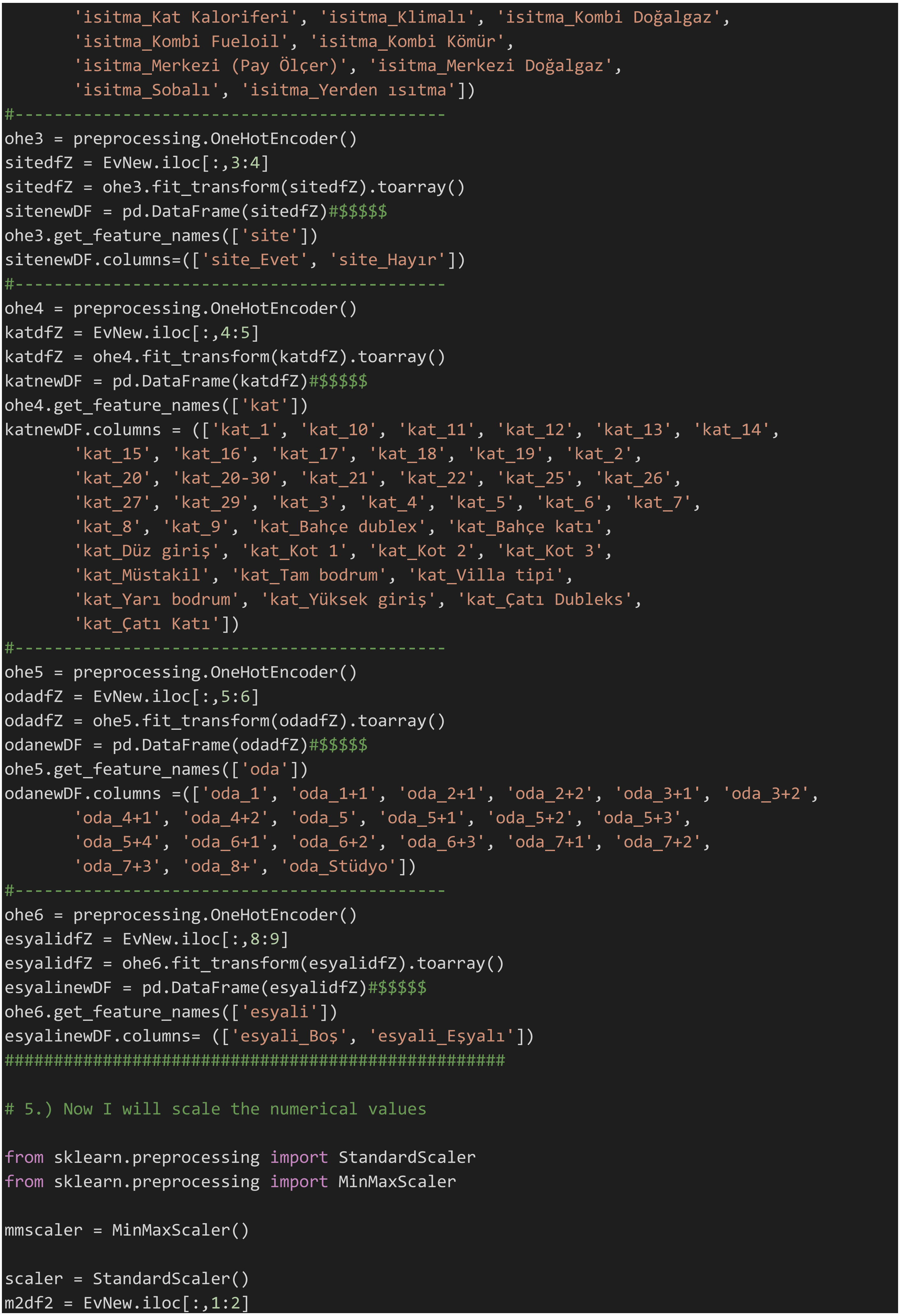


```
            isitma= soup.find('div', class_='fFWQu', text = 'Isıtma').findNext('div').text
            return isitma
        except:
            return np.nan
    def get_site(soup):
        try:
            site= soup.find('div', class_='fFWQu', text = 'Site İçerisinde').findNext('div').text
            return site
        except:
            return np.nan

    def get_kat(soup):
        try:
            kat= soup.find('div', class_='fFWQu', text = 'Dairenin Katı').findNext('div').text
            return kat
        except:
            return np.nan
    def get_oda(soup):
        try:
            oda= soup.find('div', class_='fFWQu', text = 'Oda Sayısı').findNext('div').text
            return oda
        except:
            return np.nan

    def get_yas(soup):
        try:
            yas= soup.find('div', class_='fFWQu', text = 'Bina Yaşı').findNext('div').text
            yas = clean2(yas)
            return yas
        except:
            return np.nan

    def get_banyosayisi(soup):
        try:
            banyosayisi = soup.find('div', class_='fFWQu', text = 'Banyo Sayısı').findNext('div').text
            banyosayisi = int(banyosayisi)
            return banyosayisi
        except:
            return np.nan
    def get_esyali(soup):
        try:
            esyali = soup.find('div', class_='fFWQu', text = 'Eşya Durumu').findNext('div').text
            return esyali
        except:
            return np.nan

    def flatten_list(house_links):
        house_links_flat=[]
        for sublist in house_links:
            for item in sublist:
                house_links_flat.append(item)
        return house_links_flat

    def get_house_data(driver,house_links_flat):
        house_data = []
        for link in house_links_flat:
            soup = get_html_data(link,driver)
            ilce = get_ilce(soup)
            m2 = get_m2(soup)
            isitma = get_isitma(soup)
            site = get_site(soup)
            kat = get_kat(soup)
            oda = get_oda(soup)
            yas = get_yas(soup)
            banyosayisi = get_banyosayisi(soup)
            esyali = get_esyali(soup)
            price = get_price(soup)
            house_data.append([ilce,m2,isitma,site,kat,oda,yas,banyosayisi,esyali,price])

        return house_data

    house_links_5pages = get_house_links(hurriyet_url,driver,pages=2)
    house_links_flat = flatten_list(house_links_5pages)
    house_data_5pages = get_house_data(driver,house_links_flat)

    file_name = "%s_%s.csv" % (str(time.strftime("%Y-%m-
%d")), str(time.strftime("%H%M%S")))
    columns = ['ilce','m2','isitma','site','kat','oda','yas','banyosayisi','esyali','price
']
    pd.DataFrame(house_data_5pages, columns = columns).to_csv(
        file_name, index = True, encoding = "UTF-8"
    )

#asd = pd.DataFrame(house_data_5pages, columns = columns)
tock = time.clock()
zaman = tock - tick
print("Time::")
print(zaman, "seconds")
```

B. Main Project Code:

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  2 13:52:58 2020

@author: alperen
"""
import pandas as pd
import numpy as np

# 0.)Read the csv file in to a DataFrame Object
EvAlpha = pd.read_csv("merge.csv", sep = ",")
EvAlpha.dtypes
#---First column is a mistake which comes from datset-creation-automation
#---program so I remove that column
Ev = EvAlpha.iloc[:,1:]
Ev.dtypes
print(Ev.isnull().sum())
# I need to convert prices to int (they are float for now)
# To do this 1st I need to handle the NaN values
#------------------------------------------------------------------------------
# 1.)Dealing with missing values
#---1st start with "banyosayisi"
banyosayisi = Ev.iloc[:,7:8]
from sklearn.impute import SimpleImputer
impForBanyoSayisi = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
banyosayisi = impForBanyoSayisi.fit_transform(banyosayisi)
banyosayisi = banyosayisi = banyosayisi.astype('int64')
banyosayisidf = pd.DataFrame.from_records(banyosayisi)
banyosayisidf.columns = ['banyosayisi']
#----banyosayisidf --> No nan values and its a DF---#
#------------------------------------------------------------------------------
#---2nd deal with m2
m2 = Ev.iloc[:,1:2]
impForM2 = SimpleImputer(missing_values=np.nan, strategy='mean')
m2 = impForM2.fit_transform(m2)
m2 = m2 = m2.astype('int64')
m2df = pd.DataFrame.from_records(m2)
m2df.columns = ['m2']
# m2 --> No nan values and its a DF
#------------------------------------------------------------------------------
#---3rd now time for "price" but maybe we will need both nan prices
#---and mean value imputed prices so lets create both of them
price = Ev.iloc[:,-1:]## This will be the NaN values included version
print(price.median())
impForPrice = SimpleImputer(missing_values=np.nan, strategy='median')
price = impForPrice.fit_transform(price)
price = price = price.astype('int64')
pricedf = pd.DataFrame.from_records(price)
pricedf.columns = ['price']
##Mean is giving a bad result
## I think median is better
## But maybe dropping that rows will be more clever
```

```
### For that dont combine this DF with whole dataset and the use "dropna"
#---4th deal with esyali
#--- Most of the records are unknown so I fill them with most frequent
esyaliN =  Ev.iloc[:,8:9]
impForEsyali = SimpleImputer(missing_values='Belirtilmemiş', strategy='most_frequent')
esyaliN = impForEsyali.fit_transform(esyaliN)
esyaliNdf = pd.DataFrame.from_records(esyaliN)
esyaliNdf.columns = ['esyali']
# 3.) Now combine these and drop the nan values(price), we will still have
#     more than 1700 records so it will be enough
ilce = Ev.iloc[:,0:1]
two_to_six = Ev.iloc[:,2:7]
esyali = Ev.iloc[:,8:9]
esyali_price = Ev.iloc[:,8:10]
priceUnTouched = Ev.iloc[:,-1:]
EvNew = pd.concat([ilce, m2df, two_to_six, banyosayisidf, esyaliNdf, priceUnTouched], axis =1)
EvNew = EvNew.dropna()
EvNewDrop = pd.concat([ilce, m2df, two_to_six, banyosayisidf, esyali_price], axis =1)
EvNewDrop = EvNewDrop.dropna()
print(EvNewDrop.isnull().sum())
EvNew = EvNew.astype({"price": int})
EvNew.dtypes
EvNew = EvNew.astype({"yas": int})
EvNew.dtypes
EvNew = EvNew.astype({"yas": int})
EvNew.dtypes
print(EvNew.isnull().sum())
## Pre-Processing 1st Step finished
### Now I will look at visualization a bit and then
####I will scale the numerical values and label the cat values

EvNew = EvNew[EvNew.price<10000000 ]# 10.000.000 UNUTMA
# 4.) Encode the categorical variables
from sklearn import preprocessing
import matplotlib.pyplot as plt
###################################################○
ohe = preprocessing.OneHotEncoder()
ilcedfZ = EvNew.iloc[:,0:1]
ilcedfZ2 = ohe.fit_transform(ilcedfZ).toarray()
ilcenewDF = pd.DataFrame(ilcedfZ2)#$$$$$
ilcenewDF.columns = (['arnavutkoy', 'atasehir', 'avcilar', 'bahcelievler', 'bakirkoy',
       'basaksehir', 'besiktas', 'beykoz', 'beylikduzu', 'beyoglu',
       'buyukcekmece', 'catalca', 'cekmekoy', 'esenler', 'esenyurt',
       'eyup', 'fatih', 'gaziosmanpasa', 'gungoren', 'kadikoy',
       'kagithane', 'kartal', 'kucukcekmece', 'maltepe', 'pendik',
       'sariyer', 'sisli', 'tuzla', 'umraniye', 'uskudar'])

#--------------------------------------------
ohe2 = preprocessing.OneHotEncoder()
isitmadfZ = EvNew.iloc[:,2:3]
isitmadfZ = ohe2.fit_transform(isitmadfZ).toarray()
isitmanewDF = pd.DataFrame(isitmadfZ)#$$$$$
isitmanewDF.columns = (['isitma_Doğalgaz sobalı', 'isitma_Isıtma yok', 'isitma_Jeotermal',
```



```
       'isitma_Kat Kaloriferi', 'isitma_Klimalı', 'isitma_Kombi Doğalgaz',
       'isitma_Kombi Fueloil', 'isitma_Kombi Kömür',
       'isitma_Merkezi (Pay Ölçer)', 'isitma_Merkezi Doğalgaz',
       'isitma_Sobalı', 'isitma_Yerden ısıtma'])
#---------------------------------------------
ohe3 = preprocessing.OneHotEncoder()
sitedfZ = EvNew.iloc[:,3:4]
sitedfZ = ohe3.fit_transform(sitedfZ).toarray()
sitenewDF = pd.DataFrame(sitedfZ)#$$$$$
ohe3.get_feature_names(['site'])
sitenewDF.columns=(['site_Evet', 'site_Hayır'])
#---------------------------------------------
ohe4 = preprocessing.OneHotEncoder()
katdfZ = EvNew.iloc[:,4:5]
katdfZ = ohe4.fit_transform(katdfZ).toarray()
katnewDF = pd.DataFrame(katdfZ)#$$$$$
ohe4.get_feature_names(['kat'])
katnewDF.columns = (['kat_1', 'kat_10', 'kat_11', 'kat_12', 'kat_13', 'kat_14',
       'kat_15', 'kat_16', 'kat_17', 'kat_18', 'kat_19', 'kat_2',
       'kat_20', 'kat_20-30', 'kat_21', 'kat_22', 'kat_25', 'kat_26',
       'kat_27', 'kat_29', 'kat_3', 'kat_4', 'kat_5', 'kat_6', 'kat_7',
       'kat_8', 'kat_9', 'kat_Bahçe dublex', 'kat_Bahçe katı',
       'kat_Düz giriş', 'kat_Kot 1', 'kat_Kot 2', 'kat_Kot 3',
       'kat_Müstakil', 'kat_Tam bodrum', 'kat_Villa tipi',
       'kat_Yarı bodrum', 'kat_Yüksek giriş', 'kat_Çatı Dubleks',
       'kat_Çatı Katı'])
#---------------------------------------------
ohe5 = preprocessing.OneHotEncoder()
odadfZ = EvNew.iloc[:,5:6]
odadfZ = ohe5.fit_transform(odadfZ).toarray()
odanewDF = pd.DataFrame(odadfZ)#$$$$$
ohe5.get_feature_names(['oda'])
odanewDF.columns =(['oda_1', 'oda_1+1', 'oda_2+1', 'oda_2+2', 'oda_3+1', 'oda_3+2',
       'oda_4+1', 'oda_4+2', 'oda_5', 'oda_5+1', 'oda_5+2', 'oda_5+3',
       'oda_5+4', 'oda_6+1', 'oda_6+2', 'oda_6+3', 'oda_7+1', 'oda_7+2',
       'oda_7+3', 'oda_8+', 'oda_Stüdyo'])
#---------------------------------------------
ohe6 = preprocessing.OneHotEncoder()
esyalidfZ = EvNew.iloc[:,8:9]
esyalidfZ = ohe6.fit_transform(esyalidfZ).toarray()
esyalinewDF = pd.DataFrame(esyalidfZ)#$$$$$
ohe6.get_feature_names(['esyali'])
esyalinewDF.columns= (['esyali_Boş', 'esyali_Eşyalı'])
###################################################

# 5.) Now I will scale the numerical values

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

mmscaler = MinMaxScaler()

scaler = StandardScaler()
m2df2 = EvNew.iloc[:,1:2]
```

```
m2scaled = mmscaler.fit_transform(m2df2)
m2scaledfC = pd.DataFrame(m2scaled, index=range(m2scaled.shape[0]))
m2scaledfC.columns = ['m2']

scaler2 = StandardScaler()
yasdf2 = EvNew.iloc[:,6:7]
yasscaled = mmscaler.fit_transform(yasdf2)
yasscaledC = pd.DataFrame(yasscaled, index=range(yasscaled.shape[0]))
yasscaledC.columns = ['yas']

scaler3 = StandardScaler()
banyosayisidf2 = EvNew.iloc[:,7:8]
banyosayisiscaled = mmscaler.fit_transform(banyosayisidf2)
banyosayisiscaledC = pd.DataFrame(banyosayisiscaled, index=range(banyosayisiscaled.shape[0]))
banyosayisiscaledC.columns = ['banyosayisi']

scaler4 = StandardScaler()
pricedf2 = EvNew.iloc[:,9:]
pricescaled = mmscaler.fit_transform(pricedf2)
pricescaledC = pd.DataFrame(pricescaled, index=range(pricescaled.shape[0]))
pricescaledC.columns = ['price']

### Train data

TrainData1 = pd.concat([ilcenewDF,m2scaledfC,isitmanewDF,sitenewDF,katnewDF,odanewDF,yasscaledC,banyosayisiscaledC,esyalinewDF], axis=1)
TrainData2 = pricescaledC

from sklearn.model_selection import train_test_split

mlr1_train, mlr1_test, mlr2_train, mlr2_test = train_test_split(TrainData1, TrainData2, test_size= 0.33, random_state=7)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(mlr1_train,mlr2_train)

mlr_pred = regressor.predict(mlr1_test)
from sklearn.metrics import r2_score
print("R2 = ", r2_score(mlr2_test,regressor.predict(mlr1_test)))

dumdum = np.arange(start=0, stop=109, step=1)


import statsmodels.api as sm
X = np.append(arr = np.ones((1715,1)).astype(int), values = TrainData1, axis=1)
X_list = TrainData1.iloc[:,dumdum].values

r_ols2 = sm.OLS(endog=TrainData2,exog=X_list.astype(float)).fit()

print(r_ols2.summary())

print("R2 = ", r2_score(mlr2_test,regressor.predict(mlr1_test)))
```

```
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
A = cross_val_score(regressor, TrainData1, TrainData2, cv=cv)
A = A = A.astype('int64')

TrainData1.describe()
################ Multi Linear Regression with feature selection

from sklearn.feature_selection import SelectKBest, f_regression
Atry = SelectKBest(f_regression, k=40).fit_transform(TrainData1,TrainData2)

##<------>
Atrydf = pd.DataFrame(Atry)
dumdum2 = np.arange(start=0, stop=39, step=1)
X_list2 = Atrydf.iloc[:,dumdum2].values
r_ols3 = sm.OLS(endog=TrainData2,exog=X_list2.astype(float)).fit()
print(r_ols3.summary())
##<------>

s_train, s_test, d_train, d_test = train_test_split(Atry, TrainData2, test_size= 0.33, ran
dom_state=7)

regressorNew = LinearRegression()
regressorNew.fit(s_train,d_train)

FinalPred = regressorNew.predict(s_test)

print("R2 = ", r2_score(d_test,regressorNew.predict(s_test)))


cv2 = ShuffleSplit(n_splits=5, test_size=0.33, random_state=7)
CrossValArr = cross_val_score(regressorNew, Atry, TrainData2, cv=cv2)


import seaborn as sns

sns.heatmap(EvNew.corr(), annot = True)

sns.pairplot(EvNew,kind='reg')
############## Random Forest
from sklearn.ensemble import RandomForestRegressor

reg=RandomForestRegressor(n_estimators=100,random_state=7)
reg.fit(s_train,d_train)
randomForest_predict=reg.predict(s_test)

print("R2 = ", r2_score(d_test,reg.predict(s_test)))
cv3 = ShuffleSplit(n_splits=5, test_size=0.33, random_state=7)
CrossValArr2 = cross_val_score(reg, Atry, TrainData2, cv=cv3)

p = pd.DataFrame(randomForest_predict)
```

```
k = d_test

inverseresult = mmscaler.inverse_transform(p)
inverseresult.reshape(-1,1)
inversetest = mmscaler.inverse_transform(k)
inversetest = inversetest = inversetest.astype('int64')
inverseresult = inverseresult = inverseresult.astype('int64')

sns.swarmplot(x=EvNew['ilce'], y= EvNew['price'])
plt.xticks(rotation=90)
plt.ticklabel_format(style='plain', axis='y')
plt.yticks(np.arange(500000,10500000,step=500000))
plt.show()

sns.catplot(x='isitma', y='price', data=EvNew)
plt.xticks(rotation=90)
plt.ticklabel_format(style='plain', axis='y')
plt.yticks(np.arange(250000,10500000,step=500000))
plt.show()

sns.catplot(x='site', y='price', data=EvNew)
plt.xticks(rotation=90)
plt.ticklabel_format(style='plain', axis='y')
plt.yticks(np.arange(250000,10500000,step=500000))
plt.show()

sns.catplot(x='kat', y='price', data=EvNew)
plt.xticks(rotation=90)
plt.ticklabel_format(style='plain', axis='y')
plt.yticks(np.arange(250000,10500000,step=500000))
plt.show()

sns.catplot(x='oda', y='price', data=EvNew)
plt.xticks(rotation=90)
plt.ticklabel_format(style='plain', axis='y')
plt.yticks(np.arange(250000,10500000,step=500000))
plt.show()

sns.catplot(x='esyali', y='price', data=EvNew)
plt.xticks(rotation=90)
plt.ticklabel_format(style='plain', axis='y')
plt.yticks(np.arange(250000,10500000,step=500000))
plt.show()
```