Sergejs Kopils

January 15, 2024

1 Title: Market Analysis Using Web Scraping and Data Visualization

2 Introduction

In this project, I focused on analyzing the market, specifically in areas like real estate, using web scraping and data visualization techniques. My aim was to extract valuable insights from online marketplaces and understand trends and patterns within these sectors.

3 Methodology

3.0.1 Data Collection:

I used the requests library to access data from online marketplaces. BeautifulSoup was employed to parse the HTML content and extract relevant information. Regular expressions (Re module) helped in filtering and refining the data.

3.0.2 Data Processing and Analysis:

The extracted data was organized into structured format using Pandas. I performed various operations such as cleaning, transforming, and indexing the data for better analysis.

3.0.3 Data Visualization:

For visual representation, Matplotlib and Seaborn were utilized to create various charts and graphs. These visualizations provided an intuitive understanding of the market trends.

3.0.4 Geospatial Analysis:

To add a geographical perspective to the analysis, OSMNX and Folium were used. OSMNX provided street map data, useful for location-based analysis. Interactive maps were created with Folium, integrating features like HeatMaps to represent data density in different areas.

4 Results

The analysis of the market data using web scraping and data visualization techniques has led to several interesting insights:

4.0.1 Price Trends:

The data shows a wide spread in prices across different categories. Despite the variability, it was possible to determine the average price for each category. This gives a clear picture of the market's pricing structure and helps identify the most and least expensive segments.

4.0.2 Popular Locations:

Analysis of location data revealed popular areas within the market, including specific cities and streets. This information is crucial for understanding regional market preferences and can guide businesses and investors in making location-specific decisions.

4.0.3 Geographical Spread of Listings:

A geographical analysis of the listings shows how the market is distributed across different regions. This helps in identifying areas with high market activity as well as regions that are underrepresented in the marketplace.

4.0.4 Correlation between Location and Price:

A significant correlation was observed between the location of listings and their price. This indicates that the geographical position of a property or item significantly influences its market value. It also suggests potential areas of high demand and higher value. These results provide a comprehensive view of the market, highlighting key areas such as pricing strategy, popular locations for investments, and the impact of location on price.

5 Conclusion

This project demonstrated the power of Python in data analytics, especially in market analysis. The use of libraries like requests, BeautifulSoup, pandas, and various visualization tools allowed for an in-depth understanding of market dynamics. This information is invaluable for stakeholders looking to understand market dynamics and make informed decisions.

6 Future Work

Further analysis could include more advanced statistical methods and machine learning models to predict market trends and provide more nuanced insights into consumer behavior.

7 Reference

1) Pavlyuk, D. (2021). Week 7 - Python Data Analytics. Available at: https://github.com/DmitryPavlyuk/python-da/tree/main/week7 [December 2023].
2) Pavlyuk, D. (2021). Week 6 - Python Data Analytics. Available at: https://github.com/DmitryPavlyuk/python-da/tree/main/week6 [January 2024].
3) Pavlyuk, D. (2021). Week 5 - Python Data Analytics. Available at: https://github.com/DmitryPavlyuk/python-da/tree/main/week5 [December 2023].

4) [@GrudinAndrey] (2023). [Title of the Video]. Available at: https://www.youtube.com/shorts/ZEUOAcURJvw [December 2023].

5) Stack Overflow contributors. (2023). Web scraping. Available at: https://stackoverflow.com/questions/tagged/web-scraping [December 2023].

6) GitHub contributors. (2023). OSMnx. Available at: https://github.com/topics/osmnx [January 2024].

7) GitHub contributors. (2023). Folium Python. Available at: https://github.com/topics/folium-python [January 2024].

8) OpenAI. (2023). ChatGPT at OpenAI. Available at: https://chat.openai.com/ [December 2023].

9) Blackbox AI. (2023). Blackbox AI Homepage. Available at: https://www.blackbox.ai/ [December 2023].

[1]:
```
pip install osmnx
```

```
Requirement already satisfied: osmnx in c:\users\kopil\anaconda3\lib\site-
packages (1.2.2)
Requirement already satisfied: Rtree>=1.0 in c:\users\kopil\anaconda3\lib\site-
packages (from osmnx) (1.1.0)
Requirement already satisfied: Shapely<2.0,>=1.8 in
c:\users\kopil\anaconda3\lib\site-packages (from osmnx) (1.8.5.post1)
Requirement already satisfied: matplotlib>=3.5 in
c:\users\kopil\anaconda3\lib\site-packages (from osmnx) (3.8.2)
Requirement already satisfied: numpy>=1.22 in c:\users\kopil\anaconda3\lib\site-
packages (from osmnx) (1.26.2)
Requirement already satisfied: pandas>=1.4 in c:\users\kopil\anaconda3\lib\site-
packages (from osmnx) (2.1.4)
Requirement already satisfied: networkx>=2.8 in
c:\users\kopil\anaconda3\lib\site-packages (from osmnx) (3.2.1)
Requirement already satisfied: requests>=2.28 in
c:\users\kopil\anaconda3\lib\site-packages (from osmnx) (2.31.0)
Requirement already satisfied: pyproj>=3.3 in c:\users\kopil\anaconda3\lib\site-
packages (from osmnx) (3.6.1)
Requirement already satisfied: geopandas>=0.11 in
c:\users\kopil\anaconda3\lib\site-packages (from osmnx) (0.14.1)
Requirement already satisfied: fiona>=1.8.21 in
c:\users\kopil\anaconda3\lib\site-packages (from geopandas>=0.11->osmnx) (1.9.5)
Requirement already satisfied: packaging in c:\users\kopil\anaconda3\lib\site-
packages (from geopandas>=0.11->osmnx) (21.0)
Requirement already satisfied: attrs>=19.2.0 in
c:\users\kopil\anaconda3\lib\site-packages (from
fiona>=1.8.21->geopandas>=0.11->osmnx) (21.2.0)
Requirement already satisfied: certifi in c:\users\kopil\anaconda3\lib\site-
packages (from fiona>=1.8.21->geopandas>=0.11->osmnx) (2021.10.8)
Requirement already satisfied: setuptools in c:\users\kopil\anaconda3\lib\site-
packages (from fiona>=1.8.21->geopandas>=0.11->osmnx) (69.0.2)
```

```
Requirement already satisfied: click-plugins>=1.0 in
c:\users\kopil\anaconda3\lib\site-packages (from
fiona>=1.8.21->geopandas>=0.11->osmnx) (1.1.1)
Requirement already satisfied: cligj>=0.5 in c:\users\kopil\anaconda3\lib\site-
packages (from fiona>=1.8.21->geopandas>=0.11->osmnx) (0.7.2)
Requirement already satisfied: importlib-metadata in
c:\users\kopil\anaconda3\lib\site-packages (from
fiona>=1.8.21->geopandas>=0.11->osmnx) (4.8.1)
Requirement already satisfied: six in c:\users\kopil\anaconda3\lib\site-packages
(from fiona>=1.8.21->geopandas>=0.11->osmnx) (1.16.0)
Requirement already satisfied: click~=8.0 in c:\users\kopil\anaconda3\lib\site-
packages (from fiona>=1.8.21->geopandas>=0.11->osmnx) (8.0.3)
Requirement already satisfied: colorama in c:\users\kopil\anaconda3\lib\site-
packages (from click~=8.0->fiona>=1.8.21->geopandas>=0.11->osmnx) (0.4.4)
Requirement already satisfied: importlib-resources>=3.2.0 in
c:\users\kopil\anaconda3\lib\site-packages (from matplotlib>=3.5->osmnx) (6.1.1)
Requirement already satisfied: pyparsing>=2.3.1 in
c:\users\kopil\anaconda3\lib\site-packages (from matplotlib>=3.5->osmnx) (3.0.4)
Requirement already satisfied: cycler>=0.10 in
c:\users\kopil\anaconda3\lib\site-packages (from matplotlib>=3.5->osmnx)
(0.10.0)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\kopil\anaconda3\lib\site-packages (from matplotlib>=3.5->osmnx) (1.2.0)
Requirement already satisfied: pillow>=8 in c:\users\kopil\anaconda3\lib\site-
packages (from matplotlib>=3.5->osmnx) (10.1.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
c:\users\kopil\anaconda3\lib\site-packages (from matplotlib>=3.5->osmnx) (1.3.1)
Requirement already satisfied: python-dateutil>=2.7 in
c:\users\kopil\anaconda3\lib\site-packages (from matplotlib>=3.5->osmnx) (2.8.2)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\kopil\anaconda3\lib\site-packages (from matplotlib>=3.5->osmnx)
(4.25.0)
Requirement already satisfied: zipp>=3.1.0 in c:\users\kopil\anaconda3\lib\site-
packages (from importlib-resources>=3.2.0->matplotlib>=3.5->osmnx) (3.6.0)
Requirement already satisfied: pytz>=2020.1 in
c:\users\kopil\anaconda3\lib\site-packages (from pandas>=1.4->osmnx) (2021.3)
Requirement already satisfied: tzdata>=2022.1 in
c:\users\kopil\anaconda3\lib\site-packages (from pandas>=1.4->osmnx) (2023.3)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\kopil\anaconda3\lib\site-packages (from requests>=2.28->osmnx) (3.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\users\kopil\anaconda3\lib\site-packages (from requests>=2.28->osmnx) (1.26.7)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\users\kopil\anaconda3\lib\site-packages (from requests>=2.28->osmnx) (2.0.4)
Note: you may need to restart the kernel to use updated packages.
```

[2]:

```
pip install folium
```

```
Requirement already satisfied: folium in c:\users\kopil\anaconda3\lib\site-
packages (0.15.0)
Requirement already satisfied: requests in c:\users\kopil\anaconda3\lib\site-
packages (from folium) (2.31.0)
Requirement already satisfied: branca>=0.6.0 in
c:\users\kopil\anaconda3\lib\site-packages (from folium) (0.7.0)
Requirement already satisfied: numpy in c:\users\kopil\anaconda3\lib\site-
packages (from folium) (1.26.2)
Requirement already satisfied: jinja2>=2.9 in c:\users\kopil\anaconda3\lib\site-
packages (from folium) (2.11.3)
Requirement already satisfied: MarkupSafe>=0.23 in
c:\users\kopil\anaconda3\lib\site-packages (from jinja2>=2.9->folium) (1.1.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\users\kopil\anaconda3\lib\site-packages (from requests->folium) (1.26.7)
Requirement already satisfied: certifi>=2017.4.17 in
c:\users\kopil\anaconda3\lib\site-packages (from requests->folium) (2021.10.8)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\users\kopil\anaconda3\lib\site-packages (from requests->folium) (2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
c:\users\kopil\anaconda3\lib\site-packages (from requests->folium) (3.2)
Note: you may need to restart the kernel to use updated packages.
```

[3]:
```
from folium.plugins import HeatMap
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import osmnx as ox
import requests
import folium
import re
```

This code is for a web scraping project. It gets data from a website about garages. The determine_category function decides the ad's type. It looks for words like 'Sell' or 'Buy' in the ad. The scrape_page function gets data from one webpage. It finds links to ads and gets details like price and location. The scrape_all_pages function does this for many pages. It collects data from each page on the website. The extract_numeric function gets numbers from text. It's used for turning price text into numbers. The code then puts all the data into a table using pandas. It changes dates and prices to a standard format. This makes the data easy to understand and analyze.

[5]:
```
import requests
from bs4 import BeautifulSoup
import re
import pandas as pd

def determine_category(ad_soup):
    text = ad_soup.get_text()
    if 'Miscellaneous' in text:
```

```
        return 'Miscellaneous'
    elif 'Sell' in text:
        return 'Sell'
    elif 'Buy' in text:
        return 'Buy'
    elif 'Hand over' in text:
        return 'Hand over'
    elif 'Will remove' in text:
        return 'Will remove'
    elif 'Change' in text:
        return 'Change'
    return 'Unknown'

def scrape_page(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    ad_links = soup.find_all('a', class_='am')
    links = ['https://www.ss.com' + ad['href'] for ad in ad_links]

    page_data = []

    date_regex = re.compile(r'Date: (\d{2}\.\d{2}\.\d{4})')

    for link in links:
        ad_response = requests.get(link)
        ad_soup = BeautifulSoup(ad_response.text, 'html.parser')

        city_district_tag = ad_soup.find(string='City, district:')
        city_district = city_district_tag.find_next().text if city_district_tag
 ↪else 'Not found'

        city_civil_parish_tag = ad_soup.find(string='City/civil parish:')
        city_civil_parish = city_civil_parish_tag.find_next().text if
 ↪city_civil_parish_tag else 'Not found'

        street_tag = ad_soup.find(string='Street:')
        street = street_tag.find_next().text.replace('[Map]', '').strip() if
 ↪street_tag else 'Not found'

        price_tag = ad_soup.find(string='Price:')
        price = price_tag.find_next().text if price_tag else 'Not found'

        date_match = date_regex.search(ad_soup.text)
        date = date_match.group(1) if date_match else 'Not found'

        category = determine_category(ad_soup)
```

```python
        map_link = ad_soup.find('a', class_="ads_opt_link_map")
        if map_link and 'onclick' in map_link.attrs:
            onclick_text = map_link['onclick']
            coords = re.search(r'c=(\d+\.\d+),\s*(\d+\.\d+)', onclick_text)
            latitude = coords.group(1) if coords else 'Not found'
            longitude = coords.group(2) if coords else 'Not found'
        else:
            latitude = 'Not found'
            longitude = 'Not found'

        ad_data = {
            'City, district': city_district,
            'City/civil parish': city_civil_parish,
            'Street': street,
            'Price': price,
            'Category': category,
            'Date': date,
            'Latitude': latitude,
            'Longitude': longitude
        }

        page_data.append(ad_data)

    return page_data

def scrape_all_pages(base_url, num_pages):
    all_data = []
    for page in range(1, num_pages + 1):
        url = f"{base_url}page{page}.html"
        all_data.extend(scrape_page(url))
    return all_data

def extract_numeric(value):
    numbers = re.findall(r'\d+', value)
    return float(''.join(numbers)) if numbers else 0.0

# Main URL
base_url = 'https://www.ss.com/en/real-estate/premises/garages/all/'

# Scraping data from 20 page
data = scrape_all_pages(base_url, 20)

# Creating a pandas DataFrame from the scraped data
df_garages = pd.DataFrame(data)

# Converting 'Date' to Datetime Format
df_garages['Date'] = pd.to_datetime(df_garages['Date'], format='%d.%m.%Y')
```

```
# Applying 'extract_numeric' to the 'Price' Column and converting it to Numeric␣
↪Type
df_garages['Price'] = df_garages['Price'].apply(extract_numeric)
df_garages['Price'] = pd.to_numeric(df_garages['Price'])
```

The df_garages.dtypes command displays the data types of each column in the df_garages DataFrame.

[6]:
```
df_garages.dtypes
```

[6]:
```
City, district                object
City/civil parish             object
Street                        object
Price                        float64
Category                      object
Date                  datetime64[ns]
Latitude                      object
Longitude                     object
dtype: object
```

ode counts and prints the total number of collected advertisements from the scraped data.

[7]:
```
number_of_ads = len(data)
print(f"Total number of collected advertisements: {number_of_ads}")
```

```
Total number of collected advertisements: 600
```

ode finds and stores the highest price from the 'Price' column in the df_garages DataFrame.

[8]:
```
max_price = df_garages['Price'].max()
max_price
```

[8]:
```
1370035967.0
```

ode filters the df_garages DataFrame to only include rows where the 'Price' is 100,000 or less.

[9]:
```
df_garages_filtered = df_garages[df_garages['Price'] <= 100000]
df_garages_filtered
```

[9]:
```
          City, district City/civil parish           Street    Price  \
0                   Riga        Agenskalns     Ranka d. 36  11500.0
1                   Riga               VEF       Starta 30  17000.0
2     Daugavpils and reg.        Daugavpils           Raina  11300.0
3                   Riga            Centre          Ganu 3    150.0
4        Jelgava and reg.           Jelgava   Raiņa iela 23     80.0
..                   ...               ...             ...      ...
595    Jekabpils and reg.         Jēkabpils          Cesu 2   1060.0
596      Jelgava and reg.         Kalnciems  Garažu iela 36   4000.0
597       Bauska and reg.            Bauska       Jelavas 1    600.0
```

```
598   Jekabpils and reg.          Jēkabpils         Jaunā 3   4800.0
599                  Riga         Kengarags      Aglonas 15     74.0

           Category        Date             Latitude            Longitude
0              Sell  2024-01-15           56.9490933           24.0768029
1              Sell  2024-01-15   56.979114643525975   24.167311069025306
2              Sell  2024-01-15   55.869177880606834   26.529546571358786
3         Hand over  2024-01-15     56.9617627045073   24.108273800274873
4         Hand over  2024-01-15    56.64671119133531   23.716499074493182
..              ...         ...                  ...                  ...
595            Sell  2023-11-21     56.5128850396676   25.878268824811016
596            Sell  2023-11-21   56.804532988723146    23.59518484259445
597            Sell  2023-11-21    56.33012550215928    24.33390442993776
598            Sell  2023-11-20    56.48774225565657   25.852352268539555
599   Miscellaneous  2023-11-17    56.91374747690266   24.181343302131143

[594 rows x 8 columns]
```

ode counts and prints the number of advertisements in each category from the filtered df_garages_filtered DataFrame.

[10]:
```python
category_counts = df_garages_filtered['Category'].value_counts()
print("Number of ads in each category:")
print(category_counts)
```

```
Number of ads in each category:
Category
Sell             305
Buy              136
Hand over        110
Miscellaneous     32
Will remove        9
Change             2
Name: count, dtype: int64
```

ode calculates and prints descriptive statistics, including the median, for the 'Price' column in the filtered df_garages_filtered DataFrame.

[11]:
```python
price_stats = df_garages_filtered['Price'].describe()
print("Descriptive statistics of prices:")
print(price_stats)

median_price = df_garages_filtered['Price'].median()
print("\nMedian of prices:")
print(median_price)
```

```
Descriptive statistics of prices:
count      594.000000
mean      4346.444444
```

```
std       6916.957327
min          0.000000
25%         22.000000
50%       1040.000000
75%       7436.750000
max      60000.000000
Name: Price, dtype: float64

Median of prices:
1040.0
```

ode loops through specified categories, filtering the DataFrame for each and then calculates and prints descriptive statistics and the median of prices for each category.

[12]:
```
categories = ['Miscellaneous', 'Sell', 'Buy', 'Hand over', 'Will remove',␣
↪'Change']

for category in categories:
    print(f"\n--- {category} Category ---")

    # Filter the DataFrame for the current category
    df_category = df_garages_filtered[df_garages_filtered['Category'] ==␣
↪category]

    # Descriptive statistics for the 'Price' column in the current category
    price_stats = df_category['Price'].describe()
    print("\nDescriptive statistics of prices:")
    print(price_stats)

    # Median of the 'Price' column in the current category
    median_price = df_category['Price'].median()
    print("\nMedian of prices:")
    print(median_price)
```

```
--- Miscellaneous Category ---

Descriptive statistics of prices:
count      32.000000
mean     1100.437500
std      1461.829012
min         0.000000
25%        32.500000
50%        71.500000
75%      1999.000000
max      3750.000000
Name: Price, dtype: float64
```

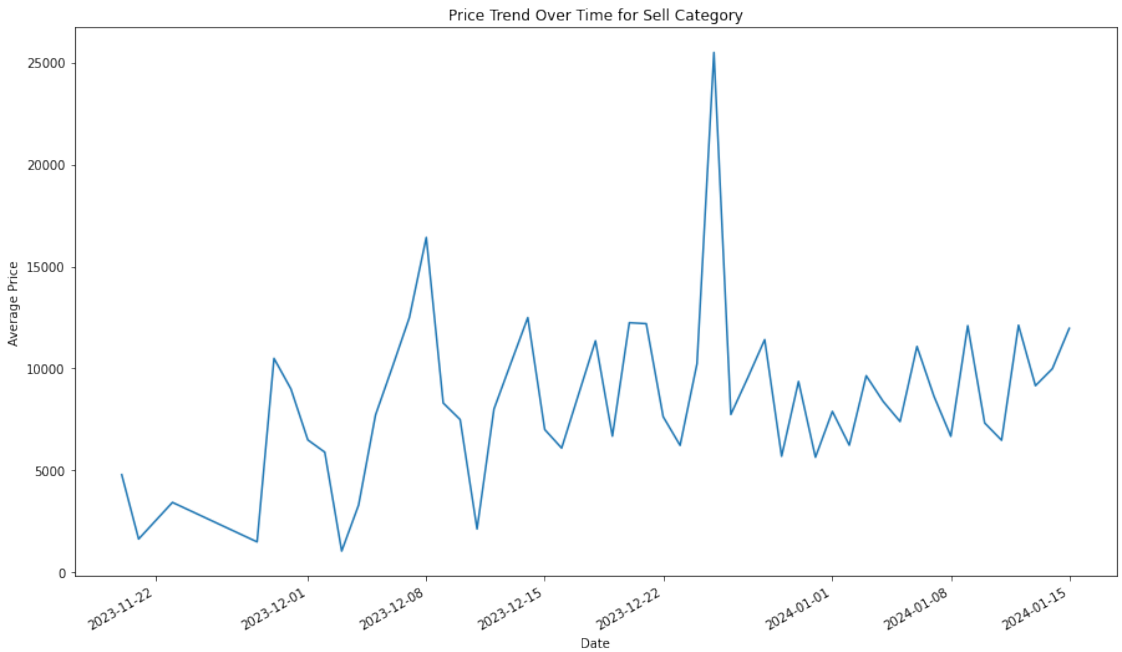
```
Median of prices:
71.5

--- Sell Category ---

Descriptive statistics of prices:
count      305.000000
mean      8317.216393
std       7777.343390
min        600.000000
25%       2500.000000
50%       7200.000000
75%      11600.000000
max      60000.000000
Name: Price, dtype: float64

Median of prices:
7200.0

--- Buy Category ---

Descriptive statistics of prices:
count    136.0
mean       0.0
std        0.0
min        0.0
25%        0.0
50%        0.0
75%        0.0
max        0.0
Name: Price, dtype: float64

Median of prices:
0.0

--- Hand over Category ---

Descriptive statistics of prices:
count    110.000000
mean      88.936364
std       75.609605
min       25.000000
25%       60.000000
50%       75.000000
75%      100.000000
max      750.000000
Name: Price, dtype: float64
```

```
Median of prices:
75.0

--- Will remove Category ---

Descriptive statistics of prices:
count     9.000000
mean      4.444444
std      13.333333
min       0.000000
25%       0.000000
50%       0.000000
75%       0.000000
max      40.000000
Name: Price, dtype: float64

Median of prices:
0.0

--- Change Category ---

Descriptive statistics of prices:
count    2.0
mean     0.0
std      0.0
min      0.0
25%      0.0
50%      0.0
75%      0.0
max      0.0
Name: Price, dtype: float64

Median of prices:
0.0
```

ode creates a copy of the DataFrame, formats dates, filters for 'Sell' category, and plots a line graph showing the average price trend over time in this category.

[13]:
```
# Create a copy of the DataFrame to avoid the SettingWithCopyWarning warning
df_garages_filtered_copy = df_garages_filtered.copy()

# Convert the 'Date' column to the correct date format
df_garages_filtered_copy['Date'] = pd.
 ↪to_datetime(df_garages_filtered_copy['Date'])

# Filter the DataFrame for the 'Sell' category
df_sell = df_garages_filtered_copy[df_garages_filtered_copy['Category'] ==␣
 ↪'Sell']
```

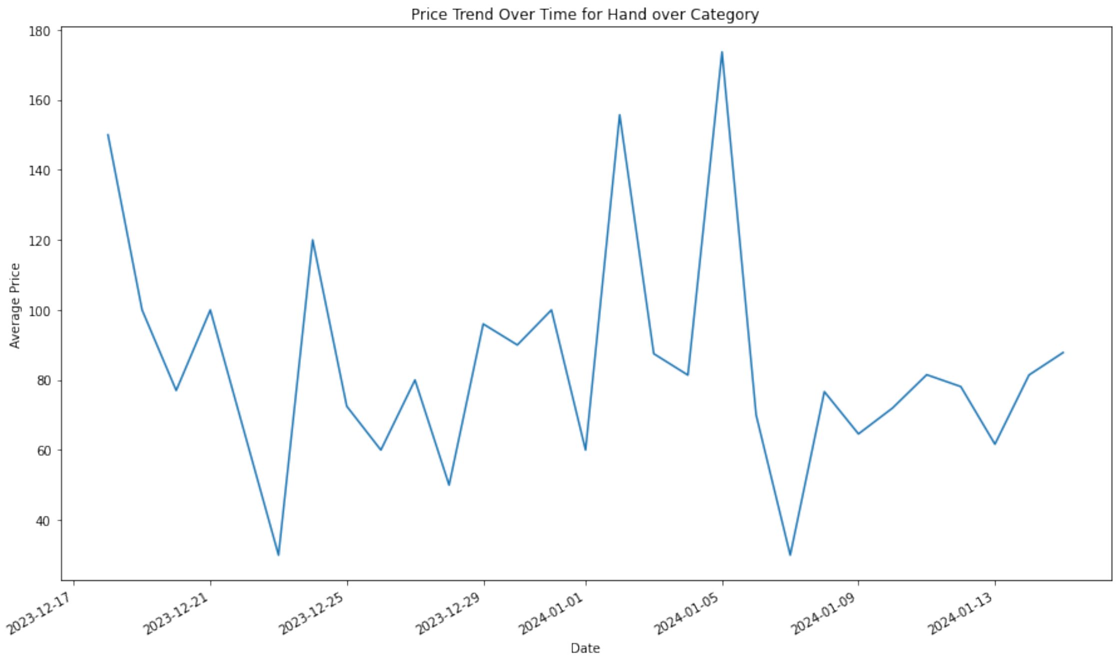
```
# Plot a graph for the price trend in the 'Sell' category
df_sell.groupby('Date')['Price'].mean().plot(kind='line', figsize=(15, 9))
plt.title('Price Trend Over Time for Sell Category')
plt.xlabel('Date')
plt.ylabel('Average Price')
plt.show()
```



ode creates a copy of the DataFrame, formats dates, filters for the 'Hand over' category, and plots a line graph illustrating the average price trend over time in this category.

```
[14]: # Create a copy of the DataFrame to avoid the SettingWithCopyWarning warning
df_garages_filtered_copy = df_garages_filtered.copy()

# Convert the 'Date' column to the correct date format
df_garages_filtered_copy['Date'] = pd.
 ↪to_datetime(df_garages_filtered_copy['Date'])

# Filter the DataFrame for the 'Hand over' category
df_hand_over = df_garages_filtered_copy[df_garages_filtered_copy['Category'] ==␣
 ↪'Hand over']

# Plot a graph for the price trend in the 'Hand over' category
df_hand_over.groupby('Date')['Price'].mean().plot(kind='line', figsize=(15, 9))
plt.title('Price Trend Over Time for Hand over Category')
plt.xlabel('Date')
```

```
plt.ylabel('Average Price')
plt.show()
```



This code calculates and prints the average price per street in the 'Street' column of the df_garages_filtered DataFrame.

[15]:
```
average_price_per_district = df_garages_filtered.groupby('Street')['Price'].
    ↪mean()
print(average_price_per_district)
```

```
Street
1 3                           0.0
18 novembra                2200.0
18.novembra 41a              50.0
4 196                      2500.0
4 Linija 5                 6599.0
                            ...
Žiguļi                       70.0
 , starta'    4200.0
                     4100.0
 503                 4100.0
                3999.0
Name: Price, Length: 344, dtype: float64
```

ode calculates and prints the count of advertisements per street in the 'Street' column of the df_garages_filtered DataFrame.

```
[16]: ads_count_per_district = df_garages_filtered['Street'].value_counts()
print(ads_count_per_district)
```

```
Street
Not found          123
Murjanu 60           9
Rigas 4              7
Krasta 95            7
Dzintara 65          7
                  ...
Lielā iela 10 a      1
Muzeja 2             1
Lepju 2              1
Avotu 8              1
Jaunā 3              1
Name: count, Length: 344, dtype: int64
```

ode counts and prints the number of advertisements for each value in the 'City, district' column of the df_garages_filtered DataFrame.

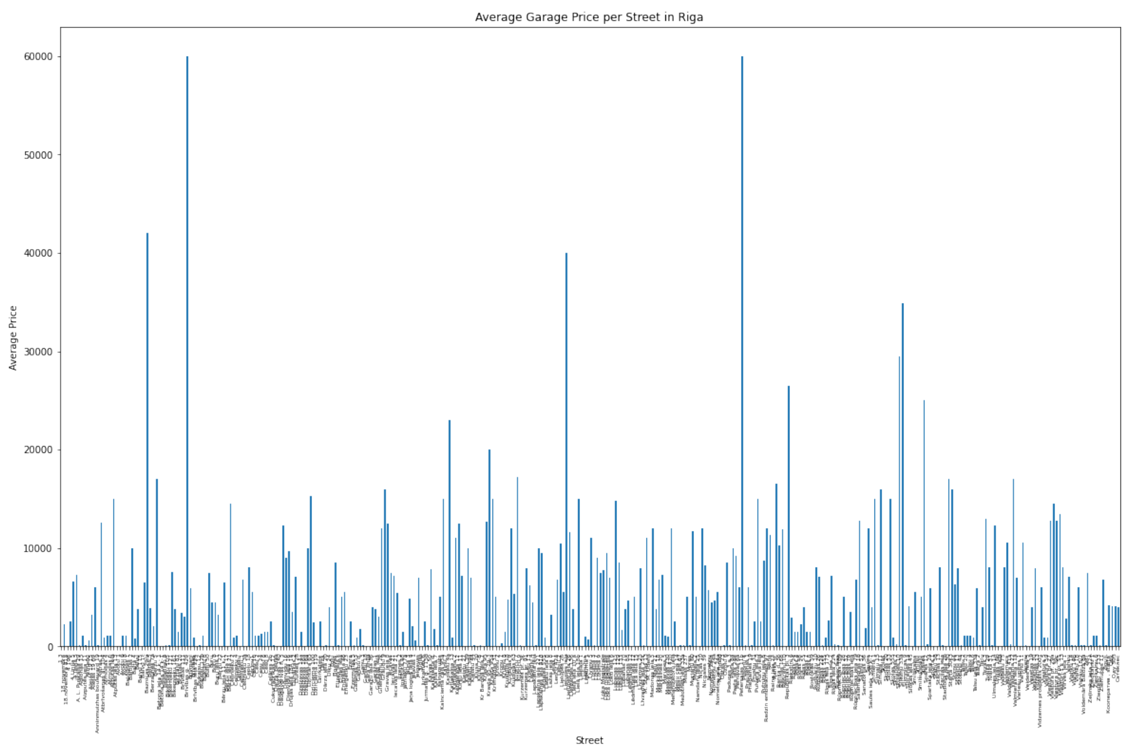
```
[17]: ad_counts = df_garages_filtered['City, district'].value_counts()
print(ad_counts)
```

```
City, district
Riga                    285
Riga district            49
Jelgava and reg.         31
Daugavpils and reg.      29
Liepaja and reg.         28
Yurmala                  21
Ventspils and reg.       14
Jekabpils and reg.       13
Valmiera and reg.        12
Ogre and reg.            11
Rezekne and reg.         10
Saldus and reg.          10
Madona and reg.           9
Tukums and reg.           9
Talsi and reg.            8
Bauska and reg.           8
Cesis and reg.            6
Kuldiga and reg.          6
Valka and reg.            6
Balvi and reg.            5
Aizkraukle and reg.       5
Dobele and reg.           4
Kraslava and reg.         4
Preili and reg.           3
Aluksne and reg.          3
```

```
Gulbene and reg.          3
Not found                 1
Limbadzi and reg.         1
Name: count, dtype: int64
```

Code creates a bar chart showing the average garage price per street in Riga using data from the average_price_per_district series.

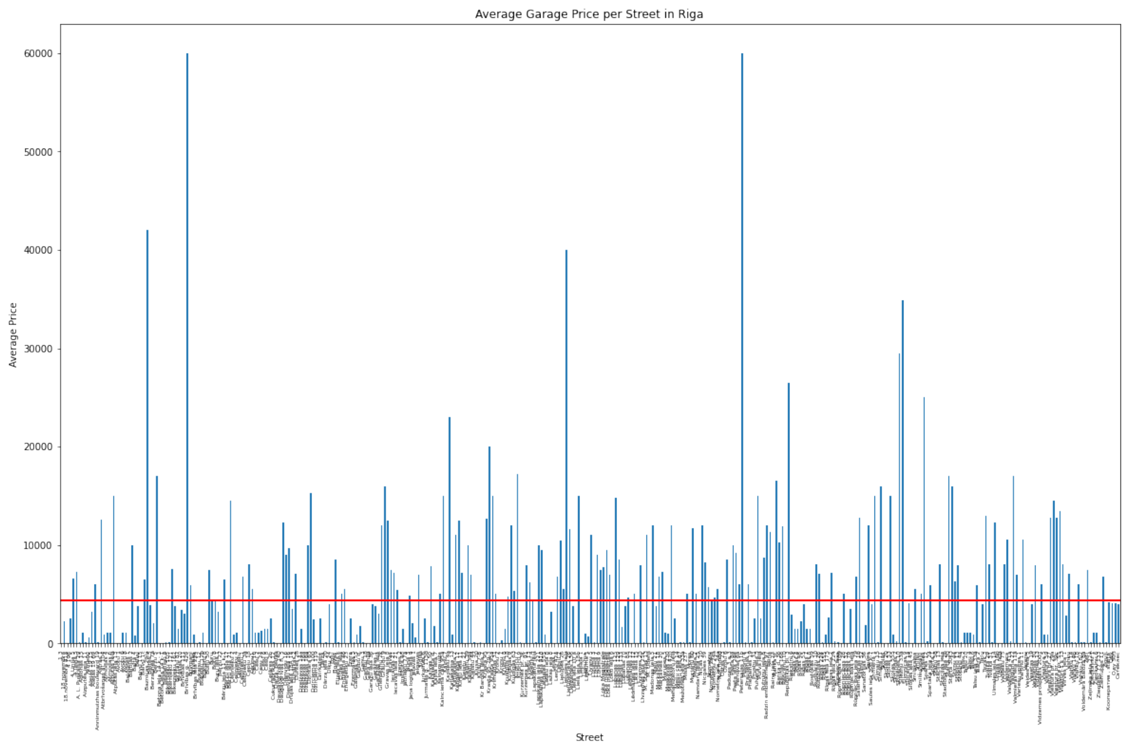
[18]:
```
average_price_per_district.plot(kind='bar', figsize=(20, 12))
plt.title('Average Garage Price per Street in Riga')
plt.xlabel('Street')
plt.ylabel('Average Price')
plt.xticks(rotation=90, fontsize=6)
plt.show()
```



This code calculates the overall average garage price for the 'Riga' district in the df_garages_filtered DataFrame and adds a horizontal red line at that average price on the existing bar chart depicting the average price per street in Riga.

[19]:
```
# Calculate the overall average price for the 'Riga' district
overall_average_price = df_garages_filtered['Price'].mean()

# Your existing code to plot the bar chart
average_price_per_district.plot(kind='bar', figsize=(20, 12))
```

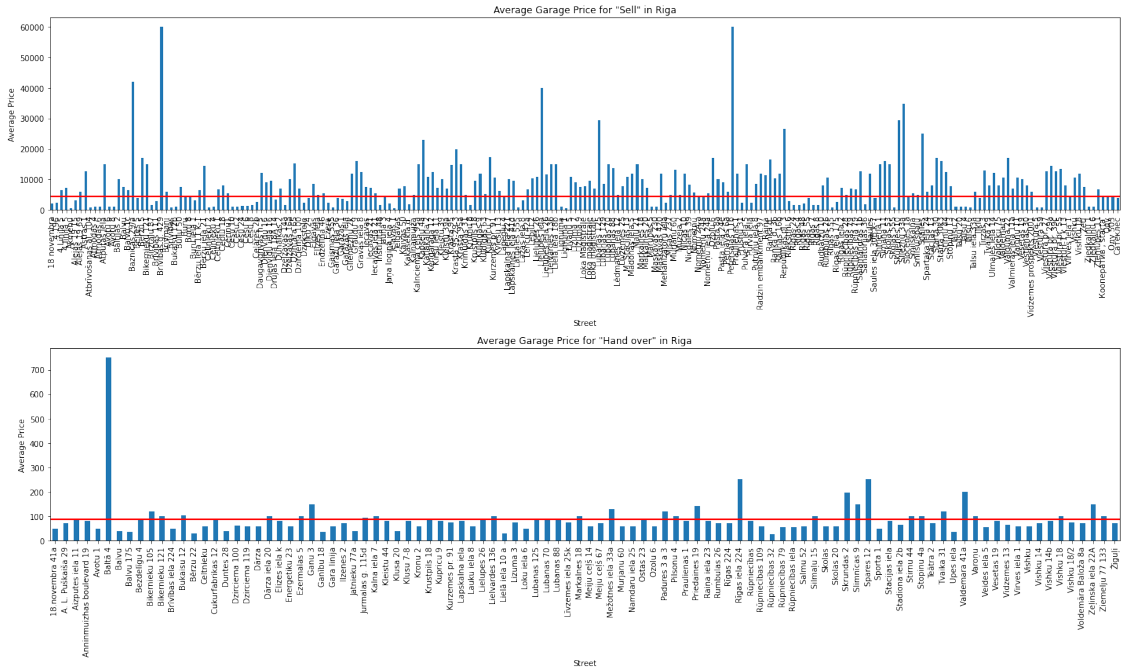
```
plt.title('Average Garage Price per Street in Riga')
plt.xlabel('Street')
plt.ylabel('Average Price')
plt.xticks(rotation=90, fontsize=6)

# Add a horizontal red line at the overall average price
plt.axhline(y=overall_average_price, color='r', linestyle='-', linewidth=2)

plt.show()
```



This code, assuming df_garages_filtered is filtered for Riga, first filters data for the 'Sell' and 'Hand over' categories, calculates the overall average price for the 'Riga' district, and the average price for the 'Hand over' category. It then creates subplots for both categories, plotting the average garage prices per street, and adds horizontal red lines at the overall average price and the average price for 'Hand over' to both subplots for comparison.

[20]:
```
# Filter data for 'Sell' category
sell_data = df_garages_filtered[df_garages_filtered['Category'] == 'Sell']

# Filter data for 'Hand over' category
hand_over_data = df_garages_filtered[df_garages_filtered['Category'] == 'Hand␣
↪over']
```

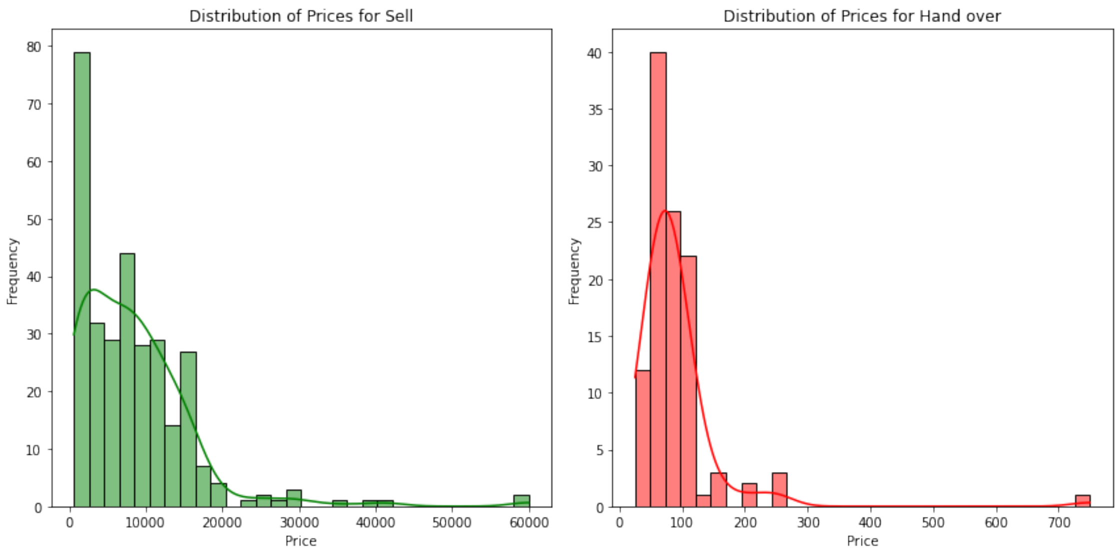
```
# Calculate the overall average price for the 'Riga' district
overall_average_price = df_garages_filtered['Price'].mean()

# Calculate the average 'Price' for the 'Hand over' category
average_price_hand_over = hand_over_data['Price'].mean()

# Create subplots for 'Sell' and 'Hand over' categories
fig, axes = plt.subplots(nrows=2, ncols=1, figsize=(20, 12))

# Plot for 'Sell' category
sell_data.groupby('Street')['Price'].mean().plot(kind='bar', ax=axes[0])
axes[0].set_title('Average Garage Price for "Sell" in Riga')
axes[0].set_xlabel('Street')
axes[0].set_ylabel('Average Price')
axes[0].tick_params(axis='x', rotation=90)

# Plot for 'Hand over' category
hand_over_data.groupby('Street')['Price'].mean().plot(kind='bar', ax=axes[1])
axes[1].set_title('Average Garage Price for "Hand over" in Riga')
axes[1].set_xlabel('Street')
axes[1].set_ylabel('Average Price')
axes[1].tick_params(axis='x', rotation=90)

# Add a horizontal red line at the overall average price to both subplots
axes[0].axhline(y=overall_average_price, color='r', linestyle='-', linewidth=2)
axes[1].axhline(y=average_price_hand_over, color='r', linestyle='-',
 ↪linewidth=2)  # Use average price for 'Hand over'

plt.tight_layout()
plt.show()
```



This code filters data for the 'Sell' and 'Hand over' categories, then creates two histograms to visualize the price distributions for each category, with the first histogram showing the distribution for 'Sell' in green and the second for 'Hand over' in red.

[21]:
```
# Filter data for 'Sell' category
sell_data = df_garages_filtered[df_garages_filtered['Category'] == 'Sell']

# Filter data for 'Hand over' category
hand_over_data = df_garages_filtered[df_garages_filtered['Category'] == 'Hand␣
 ↪over']

# Create histograms
plt.figure(figsize=(12, 6))

# Histogram for 'Sell' category
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
sns.histplot(sell_data['Price'], kde=True, color='green', bins=30)
plt.title('Distribution of Prices for Sell')
plt.xlabel('Price')
plt.ylabel('Frequency')

# Histogram for 'Hand over' category
plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
sns.histplot(hand_over_data['Price'], kde=True, color='red', bins=30)
plt.title('Distribution of Prices for Hand over')
plt.xlabel('Price')
```

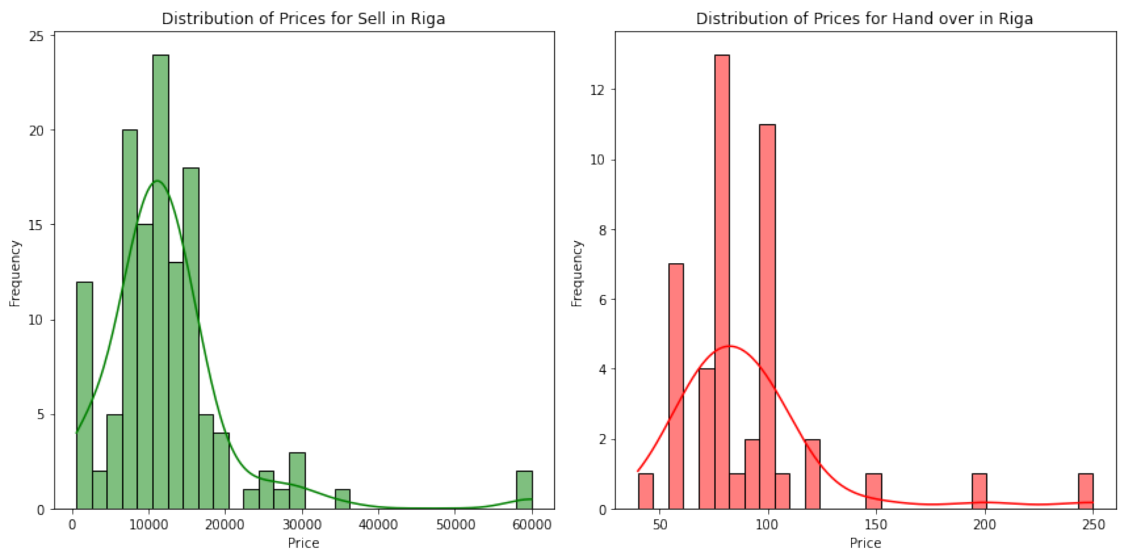
```
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



Assuming df_garages_filtered contains 'Price', 'Category', and 'City, district' columns, filters data for 'City, district' as 'Riga', then further narrows it down to 'Sell' and 'Hand over' categories within Riga, and finally creates histograms to visualize the price distributions for both categories within Riga, with the first histogram showing the distribution for 'Sell' in green and the second for 'Hand over' in red.

[22]:
```
# Filtering data for 'City, district' as 'Riga'
riga_data = df_garages_filtered[df_garages_filtered['City, district'] == 'Riga']

# Further filtering for 'Sell' and 'Hand over' categories within Riga
sell_data_riga = riga_data[riga_data['Category'] == 'Sell']
hand_over_data_riga = riga_data[riga_data['Category'] == 'Hand over']

# Creating histograms for 'Sell' and 'Hand over' in Riga
plt.figure(figsize=(12, 6))

# Histogram for 'Sell' category in Riga
plt.subplot(1, 2, 1)
sns.histplot(sell_data_riga['Price'], kde=True, color='green', bins=30)
plt.title('Distribution of Prices for Sell in Riga')
plt.xlabel('Price')
plt.ylabel('Frequency')

# Histogram for 'Hand over' category in Riga
```

```
plt.subplot(1, 2, 2)
sns.histplot(hand_over_data_riga['Price'], kde=True, color='red', bins=30)
plt.title('Distribution of Prices for Hand over in Riga')
plt.xlabel('Price')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



The DataFrame is filtered to include only advertisements from Riga, and then a predefined list of categories is looped through to analyze each category's price statistics and median in Riga, with a check to ensure data availability for each category before printing the insights.

[23]:
```
# Filtering the DataFrame for advertisements from Riga
df_riga = df_garages_filtered[df_garages_filtered['City, district'] == 'Riga']

# Predefined list of categories
categories = ['Miscellaneous', 'Sell', 'Buy', 'Hand over', 'Will remove',␣
 ↪'Change']

for category in categories:
    print(f"\n--- {category} Category in Riga ---")

    # Filtering the DataFrame for the current category
    df_category = df_riga[df_riga['Category'] == category]

    # Checking if there is data in the category
    if not df_category.empty:
        # Descriptive statistics for the 'Price' column in the current category
```

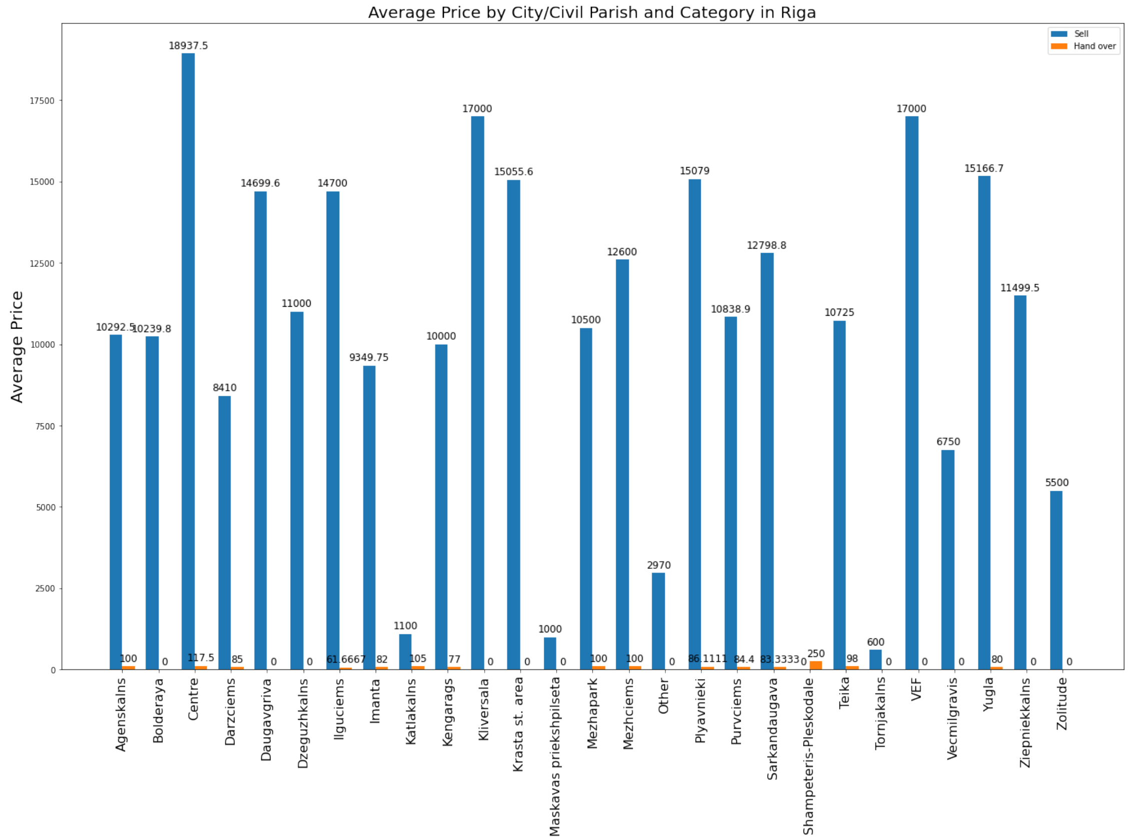
```
        price_stats = df_category['Price'].describe()
        print("\nDescriptive statistics of prices:")
        print(price_stats)

        # Median of the 'Price' column in the current category
        median_price = df_category['Price'].median()
        print("\nMedian of prices:")
        print(median_price)
    else:
        print("No data in this category.")
```

```
--- Miscellaneous Category in Riga ---

Descriptive statistics of prices:
count      28.000000
mean     1185.000000
std      1536.991747
min         0.000000
25%        32.500000
50%        55.500000
75%      2236.750000
max      3750.000000
Name: Price, dtype: float64

Median of prices:
55.5

--- Sell Category in Riga ---

Descriptive statistics of prices:
count      128.000000
mean     12312.750000
std       8625.779527
min        600.000000
25%       8000.000000
50%      11500.000000
75%      15000.000000
max      60000.000000
Name: Price, dtype: float64

Median of prices:
11500.0

--- Buy Category in Riga ---

Descriptive statistics of prices:
count    82.0
```

```
mean      0.0
std       0.0
min       0.0
25%       0.0
50%       0.0
75%       0.0
max       0.0
Name: Price, dtype: float64

Median of prices:
0.0

--- Hand over Category in Riga ---

Descriptive statistics of prices:
count     45.000000
mean      90.622222
std       35.980606
min       40.000000
25%       70.000000
50%       80.000000
75%      100.000000
max      250.000000
Name: Price, dtype: float64

Median of prices:
80.0

--- Will remove Category in Riga ---

Descriptive statistics of prices:
count    2.0
mean     0.0
std      0.0
min      0.0
25%      0.0
50%      0.0
75%      0.0
max      0.0
Name: Price, dtype: float64

Median of prices:
0.0

--- Change Category in Riga ---
No data in this category.
```

Data is filtered to include only advertisements from 'Riga' and belonging to the 'Sell' and 'Hand over' categories. The data is then grouped by 'City/civil parish' and 'Category' to calculate the mean 'Price,' which is visualized as a bar chart showing the average price by city/civil parish and category in Riga.

[24]:
```
# Filter data for 'City, district' as 'Riga' and for 'Category' as 'Sell' and␣
↪'Hand over'
riga_data = df_garages_filtered[(df_garages_filtered['City, district'] ==␣
↪'Riga') &
                                (df_garages_filtered['Category'].isin(['Sell',␣
↪'Hand over']))]

# Group data by 'City/civil parish' and 'Category', then calculate the mean␣
↪'Price'
grouped_data = riga_data.groupby(['City/civil parish', 'Category'])['Price'].
↪mean().unstack()

# If there's no data for some combinations, fill with 0
grouped_data = grouped_data.fillna(0)

# Setup for bar chart
parishes = grouped_data.index
x = np.arange(len(parishes))  # the label locations
width = 0.35  # the width of the bars

# Create a larger figure
fig, ax = plt.subplots(figsize=(19.2, 14.4))

# Creating bars for each category
rects1 = ax.bar(x - width/2, grouped_data['Sell'], width, label='Sell')
rects2 = ax.bar(x + width/2, grouped_data['Hand over'], width, label='Hand over')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Average Price', fontsize=20)
ax.set_title('Average Price by City/Civil Parish and Category in Riga',␣
↪fontsize=20)
ax.set_xticks(x)
ax.set_xticklabels(parishes, rotation=90, fontsize=16)
ax.legend()

# Add bar labels
ax.bar_label(rects1, padding=3, fontsize=12)
ax.bar_label(rects2, padding=3, fontsize=12)

fig.tight_layout()
plt.show()
```

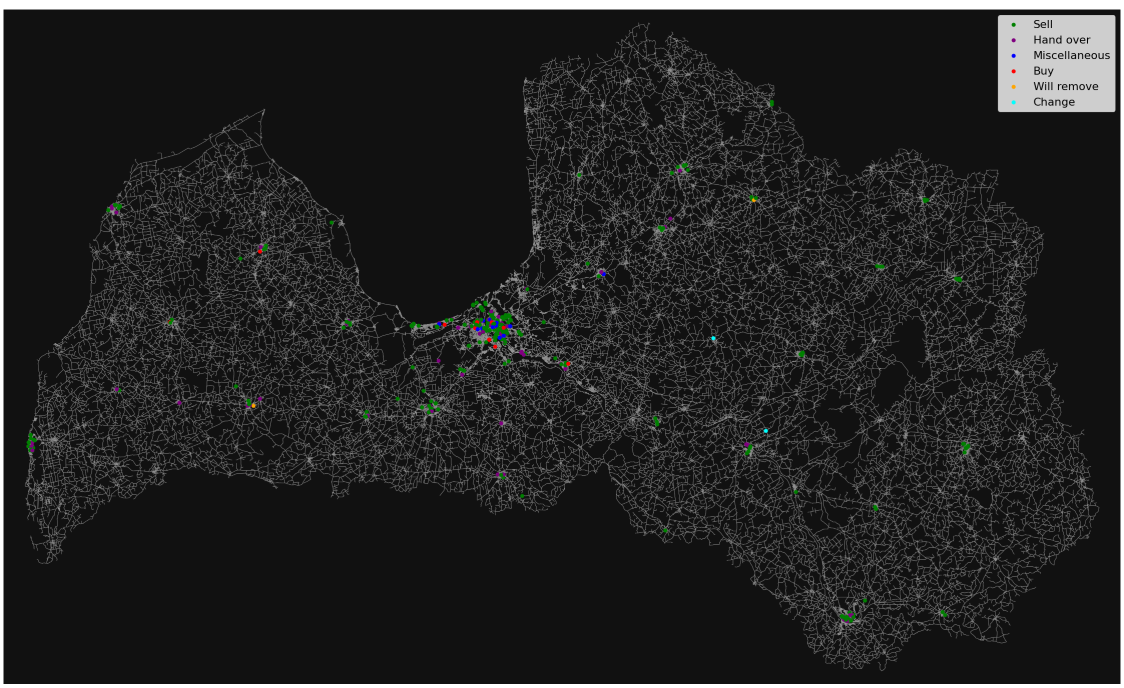


In this code, data from the df_garages_filtered DataFrame is assigned to the variable adv_data_Latvia. Color codes for different categories are defined, and the road network for 'Latvia' is retrieved using OSMnx. The code then plots a map displaying road networks and garage locations, color-coded by category, and includes a legend to distinguish between different categories.

[28]:

```
adv_data_Latvia = df_garages_filtered

# Define colors for each category
category_colors = {
    'Miscellaneous': 'blue',
    'Sell': 'green',
    'Buy': 'red',
    'Hand over': 'purple',
    'Will remove': 'orange',
    'Change': 'cyan',
}

# Specify the place name
place_name = "Latvia"
```

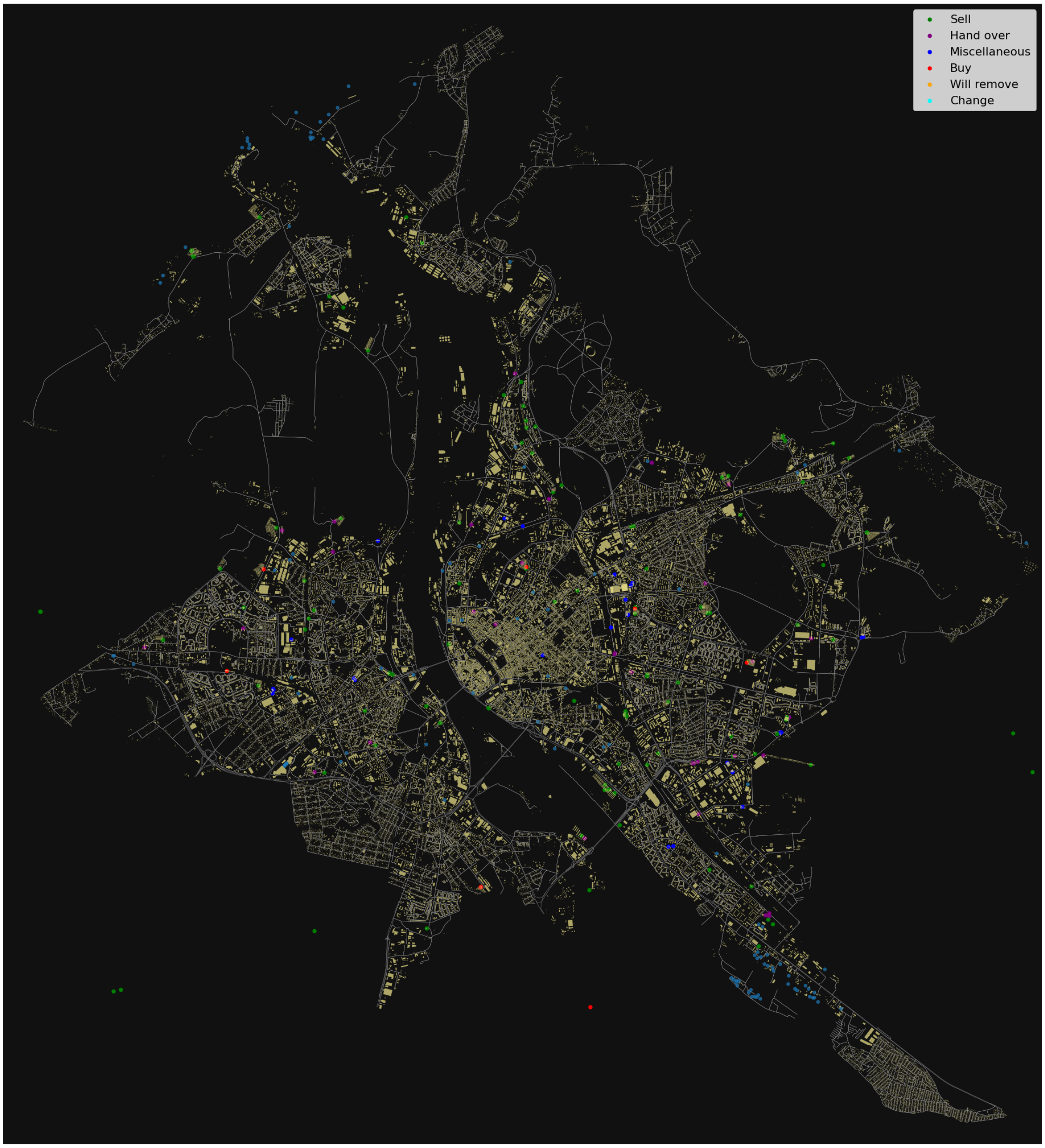
```
# Retrieve the road network for the specified place
G = ox.graph_from_place(place_name, network_type='drive')

# Plot the map with roads and garage locations
fig, ax = ox.plot_graph(G, figsize=(40, 40), show=False, close=False,␣
↪node_size=0, edge_linewidth=0.5)

# Add garage locations to the map with color coding for categories
for _, row in adv_data_Latvia.iterrows():
    if pd.notna(row['Longitude']) and pd.notna(row['Latitude']) and pd.
↪notna(row['Category']):
        category = row['Category']
        if category in category_colors:
            color = category_colors[category]
            ax.scatter(row['Longitude'], row['Latitude'], c=color, s=50,␣
↪label=category)

# Set global font size
plt.rcParams.update({'font.size': 22})

# Create a legend with unique labels
handles, labels = plt.gca().get_legend_handles_labels()
by_label = dict(zip(labels, handles))
ax.legend(by_label.values(), by_label.keys(), loc='best')

plt.show()
```



Data from the df_garages_filtered DataFrame is assigned to the variable adv_data_Riga. Color codes for different categories are defined, and the road network for 'Riga, Latvia' is retrieved using OSMnx, along with building data for the same location. The code then plots a map displaying road networks, garage locations (color-coded by category), and buildings in Riga, with a legend to distinguish between different categories.

[29]:
```python
adv_data_Riga = df_garages_filtered

# Define colors for each category
category_colors = {
    'Miscellaneous': 'blue',
    'Sell': 'green',
    'Buy': 'red',
    'Hand over': 'purple',
    'Will remove': 'orange',
    'Change': 'cyan',
}

# Specify the place name
place_name = "Riga, Latvia"

# Retrieve the road network for the specified place
G = ox.graph_from_place(place_name, network_type='drive')

# Retrieve buildings in the specified place
buildings = ox.geometries_from_place(place_name, tags={'building': True})

# Plot the map with roads, garage locations, and buildings
fig, ax = ox.plot_graph(G, figsize=(40, 40), show=False, close=False,␣
 ↪node_size=0, edge_linewidth=0.5)

# Add garage locations to the map with color coding for categories
for _, row in adv_data_Riga.iterrows():
    if pd.notna(row['Longitude']) and pd.notna(row['Latitude']) and pd.
 ↪notna(row['Category']):
        category = row['Category']
        if category in category_colors:
            color = category_colors[category]
            ax.scatter(row['Longitude'], row['Latitude'], c=color, s=50,␣
 ↪label=category)

# Set global font size
plt.rcParams.update({'font.size': 22})

# Plot buildings on the map
buildings.plot(ax=ax, facecolor='khaki', alpha=0.7)
```

```python
# Create a legend with unique labels
handles, labels = plt.gca().get_legend_handles_labels()
by_label = dict(zip(labels, handles))
ax.legend(by_label.values(), by_label.keys(), loc='best')

plt.show()
```



In this code, rows in the df_garages_filtered DataFrame where 'Latitude' or 'Longitude' is not a valid number are filtered out, and then 'Latitude' and 'Longitude' columns are converted to numeric values. The code prepares the data for a heatmap and generates a Folium map centered around the central point of the garage locations, with a heatmap layer representing the geographic distribution of the garages.

[27]:
```
# Filter out rows where 'Latitude' or 'Longitude' is not a valid number
df_garages_filtered = df_garages_filtered[pd.
 ↪to_numeric(df_garages_filtered['Latitude'], errors='coerce').notnull()]
df_garages_filtered = df_garages_filtered[pd.
 ↪to_numeric(df_garages_filtered['Longitude'], errors='coerce').notnull()]

# Convert 'Latitude' and 'Longitude' to numeric values
df_garages_filtered['Latitude'] = pd.to_numeric(df_garages_filtered['Latitude'])
df_garages_filtered['Longitude'] = pd.
 ↪to_numeric(df_garages_filtered['Longitude'])

# Prepare the data for the heatmap
heatmap_data = df_garages_filtered[['Latitude', 'Longitude']].values.tolist()

# Calculate the central point
central_latitude = df_garages_filtered['Latitude'].mean()
central_longitude = df_garages_filtered['Longitude'].mean()
central_point = {'Latitude': central_latitude, 'Longitude': central_longitude}

# Create a map centered around the central point
folium_map = folium.Map(location=[central_point['Latitude'],␣
 ↪central_point['Longitude']], zoom_start=12)

# Add a heatmap layer
HeatMap(heatmap_data).add_to(folium_map)

folium_map
```

[27]:
```
<folium.folium.Map at 0x209b75ac790>
```

8 Thank you!