1

```
#from google.colab import drive
#drive.mount('/content/drive')
#%cd /content/drive/My Drive/Colab Notebooks
#import warnings
#warnings.filterwarnings(action='ignore', category=FutureWarning)


import json
import math
from typing import List, Optional
#!pip install parsel
#!pip install scrapfly-sdk
from parsel import Selector
from typing_extensions import TypedDict
from scrapfly import ScrapflyClient, ScrapeConfig

import re
import pandas as pd
from bs4 import BeautifulSoup
from scrapfly import ScrapflyClient, ScrapeConfig




client = ScrapflyClient("scp-live-e845369bb67a4b8298658e087cb182e5")

def find_properties(state: str, city: str, pages: int = 30):
    house_info_list = []
    for page in range(1, pages + 1):
        print(f"Scraping page {page} for {city}, {state}")
        page_url = f"https://www.realtor.com/realestateandhomes-
search/{city}_{state.upper()}/pg-{page}"

        scrape_result = client.scrape(ScrapeConfig(url=page_url,
country="US", asp=True))
        html_content = scrape_result.content

        soup = BeautifulSoup(html_content, 'html.parser')
        properties_info = soup.find_all('li', attrs={'data-testid':
re.compile(r'property-meta-.+')})
        properties_set = set([info.find_parent('ul') for info in
properties_info])


        for prop in properties_set:
            if prop:
                beds = prop.find('li', {'data-testid': 'property-meta-
```

```
beds'}).find('span', {'data-testid': 'meta-
value'}).get_text(strip=True) if prop.find('li', {'data-testid':
'property-meta-beds'}) else 'N/A'
                baths = prop.find('li', {'data-testid': 'property-
meta-baths'}).find('span', {'data-testid': 'meta-
value'}).get_text(strip=True) if prop.find('li', {'data-testid':
'property-meta-baths'}) else 'N/A'
                sqft = prop.find('li', {'data-testid': 'property-meta-
sqft'}).find('span', {'data-testid': 'meta-
value'}).get_text(strip=True) if prop.find('li', {'data-testid':
'property-meta-sqft'}) else 'N/A'
                lot_size = prop.find('li', {'data-testid': 'property-
meta-lot-size'}).find('span', {'data-testid': 'meta-
value'}).get_text(strip=True) if prop.find('li', {'data-testid':
'property-meta-lot-size'}) else 'N/A'

                price_wrapper = prop.find_previous_sibling('div',
class_='price-wrapper')
                price = price_wrapper.find('div', {'data-testid':
'card-price'}).get_text(strip=True) if price_wrapper else 'N/A'

                house_info = {
                    'Price': price,
                    'Beds': beds,
                    'Baths': baths,
                    'Area (sqft)': sqft,
                    'Lot Size': lot_size
                }

                house_info_list.append(house_info)

    return house_info_list




locations = [
    ("Los-Angeles","CA"),
    ("San-Francisco", "CA"),
    ("New-York", "NY"),
    ("Seattle", "WA"),
    ("Dallas", "TX")
]

dfs = {}

for city, state in locations:
    house_info_list = find_properties(state, city, 30)
    dfs[city] = pd.DataFrame(house_info_list)
```

```python
print(dfs["San-Francisco"])
```

```
Scraping page 1 for Los-Angeles, CA
Scraping page 2 for Los-Angeles, CA
Scraping page 3 for Los-Angeles, CA
Scraping page 4 for Los-Angeles, CA
Scraping page 5 for Los-Angeles, CA
Scraping page 6 for Los-Angeles, CA
Scraping page 7 for Los-Angeles, CA
Scraping page 8 for Los-Angeles, CA
Scraping page 9 for Los-Angeles, CA
Scraping page 10 for Los-Angeles, CA
Scraping page 11 for Los-Angeles, CA
Scraping page 12 for Los-Angeles, CA
Scraping page 13 for Los-Angeles, CA
Scraping page 14 for Los-Angeles, CA
Scraping page 15 for Los-Angeles, CA
Scraping page 16 for Los-Angeles, CA
Scraping page 17 for Los-Angeles, CA
Scraping page 18 for Los-Angeles, CA
Scraping page 19 for Los-Angeles, CA
Scraping page 20 for Los-Angeles, CA
Scraping page 21 for Los-Angeles, CA
Scraping page 22 for Los-Angeles, CA
Scraping page 23 for Los-Angeles, CA
Scraping page 24 for Los-Angeles, CA
Scraping page 25 for Los-Angeles, CA
Scraping page 26 for Los-Angeles, CA
Scraping page 27 for Los-Angeles, CA
Scraping page 28 for Los-Angeles, CA
Scraping page 29 for Los-Angeles, CA
Scraping page 30 for Los-Angeles, CA
Scraping page 1 for San-Francisco, CA
Scraping page 2 for San-Francisco, CA
Scraping page 3 for San-Francisco, CA
Scraping page 4 for San-Francisco, CA
Scraping page 5 for San-Francisco, CA
Scraping page 6 for San-Francisco, CA
Scraping page 7 for San-Francisco, CA
Scraping page 8 for San-Francisco, CA
Scraping page 9 for San-Francisco, CA
Scraping page 10 for San-Francisco, CA
Scraping page 11 for San-Francisco, CA
Scraping page 12 for San-Francisco, CA
Scraping page 13 for San-Francisco, CA
Scraping page 14 for San-Francisco, CA
Scraping page 15 for San-Francisco, CA
Scraping page 16 for San-Francisco, CA
Scraping page 17 for San-Francisco, CA
```

```
Scraping page 18 for San-Francisco, CA
Scraping page 19 for San-Francisco, CA
Scraping page 20 for San-Francisco, CA
Scraping page 21 for San-Francisco, CA
Scraping page 22 for San-Francisco, CA
Scraping page 23 for San-Francisco, CA
Scraping page 24 for San-Francisco, CA
Scraping page 25 for San-Francisco, CA
Scraping page 26 for San-Francisco, CA
Scraping page 27 for San-Francisco, CA
Scraping page 28 for San-Francisco, CA
Scraping page 29 for San-Francisco, CA
Scraping page 30 for San-Francisco, CA
Scraping page 1 for New-York, NY
Scraping page 2 for New-York, NY
Scraping page 3 for New-York, NY
Scraping page 4 for New-York, NY
Scraping page 5 for New-York, NY
Scraping page 6 for New-York, NY
Scraping page 7 for New-York, NY
Scraping page 8 for New-York, NY
Scraping page 9 for New-York, NY
Scraping page 10 for New-York, NY
Scraping page 11 for New-York, NY
Scraping page 12 for New-York, NY
Scraping page 13 for New-York, NY
Scraping page 14 for New-York, NY
Scraping page 15 for New-York, NY
Scraping page 16 for New-York, NY
Scraping page 17 for New-York, NY
Scraping page 18 for New-York, NY
Scraping page 19 for New-York, NY
Scraping page 20 for New-York, NY
Scraping page 21 for New-York, NY
Scraping page 22 for New-York, NY
Scraping page 23 for New-York, NY
Scraping page 24 for New-York, NY
Scraping page 25 for New-York, NY
Scraping page 26 for New-York, NY
Scraping page 27 for New-York, NY
Scraping page 28 for New-York, NY
Scraping page 29 for New-York, NY
Scraping page 30 for New-York, NY
Scraping page 1 for Seattle, WA
Scraping page 2 for Seattle, WA
Scraping page 3 for Seattle, WA
Scraping page 4 for Seattle, WA
Scraping page 5 for Seattle, WA
Scraping page 6 for Seattle, WA
Scraping page 7 for Seattle, WA
```

```
Scraping page 8 for Seattle, WA
Scraping page 9 for Seattle, WA
Scraping page 10 for Seattle, WA
Scraping page 11 for Seattle, WA
Scraping page 12 for Seattle, WA
Scraping page 13 for Seattle, WA
Scraping page 14 for Seattle, WA
Scraping page 15 for Seattle, WA
Scraping page 16 for Seattle, WA
Scraping page 17 for Seattle, WA
Scraping page 18 for Seattle, WA
Scraping page 19 for Seattle, WA
Scraping page 20 for Seattle, WA
Scraping page 21 for Seattle, WA
Scraping page 22 for Seattle, WA
Scraping page 23 for Seattle, WA
Scraping page 24 for Seattle, WA
Scraping page 25 for Seattle, WA
Scraping page 26 for Seattle, WA
Scraping page 27 for Seattle, WA
Scraping page 28 for Seattle, WA
Scraping page 29 for Seattle, WA
Scraping page 30 for Seattle, WA
Scraping page 1 for Dallas, TX
Scraping page 2 for Dallas, TX
Scraping page 3 for Dallas, TX
Scraping page 4 for Dallas, TX
Scraping page 5 for Dallas, TX
Scraping page 6 for Dallas, TX
Scraping page 7 for Dallas, TX
Scraping page 8 for Dallas, TX
Scraping page 9 for Dallas, TX
Scraping page 10 for Dallas, TX
Scraping page 11 for Dallas, TX
Scraping page 12 for Dallas, TX
Scraping page 13 for Dallas, TX
Scraping page 14 for Dallas, TX
Scraping page 15 for Dallas, TX
Scraping page 16 for Dallas, TX
Scraping page 17 for Dallas, TX
Scraping page 18 for Dallas, TX
Scraping page 19 for Dallas, TX
Scraping page 20 for Dallas, TX
Scraping page 21 for Dallas, TX
Scraping page 22 for Dallas, TX
Scraping page 23 for Dallas, TX
Scraping page 24 for Dallas, TX
Scraping page 25 for Dallas, TX
Scraping page 26 for Dallas, TX
Scraping page 27 for Dallas, TX
```

```
Scraping page 28 for Dallas, TX
Scraping page 29 for Dallas, TX
Scraping page 30 for Dallas, TX
          Price    Beds Baths Area (sqft) Lot Size
0    $1,350,000       2   1.5       1,281    2,500
1      $361,677       2     2       1,089     0.41
2    $1,100,000  Studio     1       1,075    2,996
3      $107,500       3     1         N/A      N/A
4    $2,495,000       3   3.5       2,836    3,332
..          ...     ...   ...         ...      ...
235    $598,000  Studio     1         586     1.06
236    $629,000  Studio     1         465     0.46
237    $437,361       1     1         693     1.84
238  $2,400,000      10   N/A       5,170    1,999
239    $758,000       1     1         845     0.34

[240 rows x 5 columns]

import pandas as pd

for city, df in dfs.items():
    df['City'] = city




all_cities_df = pd.concat(dfs.values(), ignore_index=True)

all_cities_df['Price'] = all_cities_df['Price'].astype(str)

# Remove non-numeric characters from 'Price' column
all_cities_df['Price'] = all_cities_df['Price'].str.replace('[^\d.]',
'', regex=True)

# Convert 'Price' column to numeric
all_cities_df['Price'] = pd.to_numeric(all_cities_df['Price'],
errors='coerce')



import pandas as pd
import requests
import requests_cache
from bs4 import BeautifulSoup
import re

def scrape_city_data(locations):
    base_url = "https://www.city-data.com/"

    results = []
```

```
    for city, state in locations:
        path = f'/city/{city.replace(" ", "-")}-{state.replace(" ",
"-")}.html'

        response = requests.get(base_url + path)
        response.raise_for_status()
        html_content = response.text

        soup = BeautifulSoup(html_content, 'html.parser')

        cost_of_living_section = soup.find('section', id='cost-of-
living-index')
        median_income_section = soup.find('section', id='median-
income')
        crime_section = soup.find('section', id='crime')

        city_data = {"City": city, "State": state}

        if cost_of_living_section:
            cost_of_living_text = cost_of_living_section.get_text()
            index_value = cost_of_living_text.split(':')[1].split()[0]
            city_data["Cost of Living Index"] = index_value

        if median_income_section:
            income_text = median_income_section.get_text()
            income_value = income_text.split(':')[1].split()[0]
            income_value = re.sub(r'[^\d.]', '', income_value)
            city_data["Median Income"] = income_value

        if crime_section:
            headers =
crime_section.find('thead').find('tr').find_all('th')

            index_2020 = None
            for i, header in enumerate(headers):
                if header.get_text().strip() == "2020":
                    index_2020 = i
                    break

            if index_2020 is not None:
                crime_index_row =
crime_section.find('tfoot').find('tr').find_all('td')
                crime_index_2020 =
crime_index_row[index_2020].get_text()
                city_data["Crime Index "] = crime_index_2020

        results.append(city_data)

    return results
```

```
locations = [
    ("Los-Angeles", "California"),
    ("San-Francisco", "California"),
    ("New-York", "New York"),
    ("Seattle", "Washington"),
    ("Dallas", "Texas")
]


city_data = scrape_city_data(locations)

df = pd.DataFrame(city_data)

df.head()
```

```
            City       State Cost of Living Index Median Income Crime
Index
0    Los-Angeles  California                145.1         70372
327.4
1  San-Francisco  California                141.1        121826
387.4
2       New-York    New York                160.2         67997
229.7
3        Seattle  Washington                118.5        110781
440.8
4         Dallas       Texas                 96.1         57995
439.5
```

We scraped through https://www.city-data.com/ to get the cost of living index, median income, and crime index of all of the cities we were interested in to see if this will make an impact on the house price per city.

```
merged_df = pd.merge(df, all_cities_df, on='City')
merged_df.head()


merged_df.to_csv('real_estate_data.csv', index=False)
print("The merged DataFrame has been saved to real_estate_data.csv.")
```

```
The merged DataFrame has been saved to real_estate_data.csv.
```

```
merged_df=pd.read_csv('real_estate_data.csv')
merged_df['Price'] = merged_df['Price'].replace('[\$,]', '',
regex=True).astype(float)

merged_df.head()
```

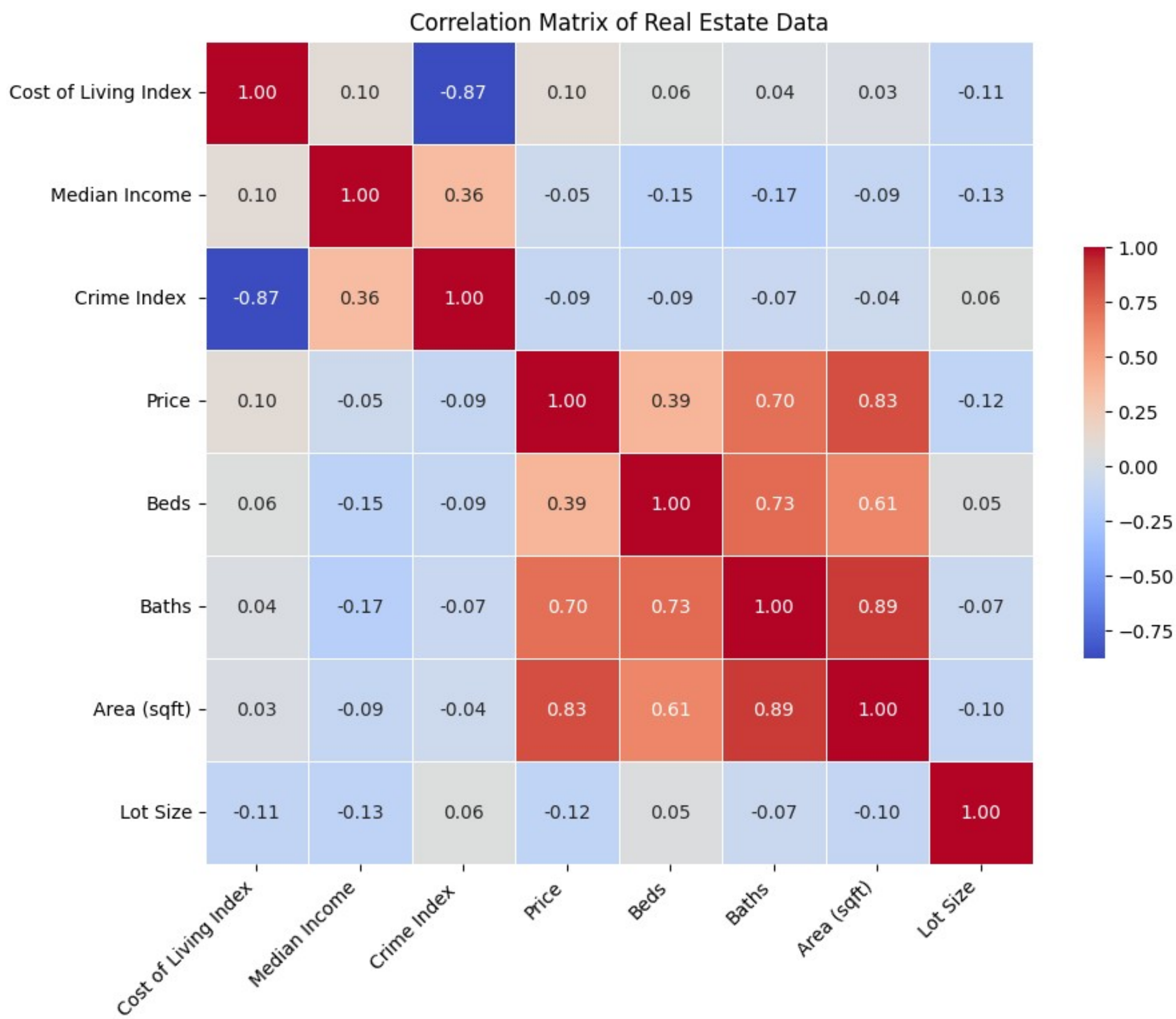
```
          City       State  Cost of Living Index  Median Income  Crime
Index   \
```

```
0  Los-Angeles  California                  145.1          70372
327.4
1  Los-Angeles  California                  145.1          70372
327.4
2  Los-Angeles  California                  145.1          70372
327.4
3  Los-Angeles  California                  145.1          70372
327.4
4  Los-Angeles  California                  145.1          70372
327.4

         Price Beds  Baths Area (sqft) Lot Size
0  139000000.0   12     17         NaN     2.08
1  155000000.0   14  16.5+      56,500      4.6
2  195000000.0    7     20         NaN      8.4
3   85000000.0   13     16      28,000      2.2
4    3395000.0    5      6       4,816    9,408
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'merged_df' is your pandas DataFrame
# Clean and convert necessary columns to numeric as done previously
merged_df['Beds'] = pd.to_numeric(merged_df['Beds'], errors='coerce')
merged_df['Baths'] = pd.to_numeric(merged_df['Baths'],
errors='coerce')
merged_df['Area (sqft)'] = pd.to_numeric(merged_df['Area
(sqft)'].str.replace(',', ''), errors='coerce')
merged_df['Lot Size'] = pd.to_numeric(merged_df['Lot
Size'].str.replace(',', ''), errors='coerce')

# Compute the correlation matrix
corr = merged_df.corr()

# Create the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm',
linewidths=.5, cbar_kws={"shrink": .5})

# Adjust the layout
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.title('Correlation Matrix of Real Estate Data')
plt.show()
```

```
/var/folders/lx/4_g1bf5951j3ls64g8b047yr0000gn/T/
ipykernel_62777/620889705.py:13: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
```

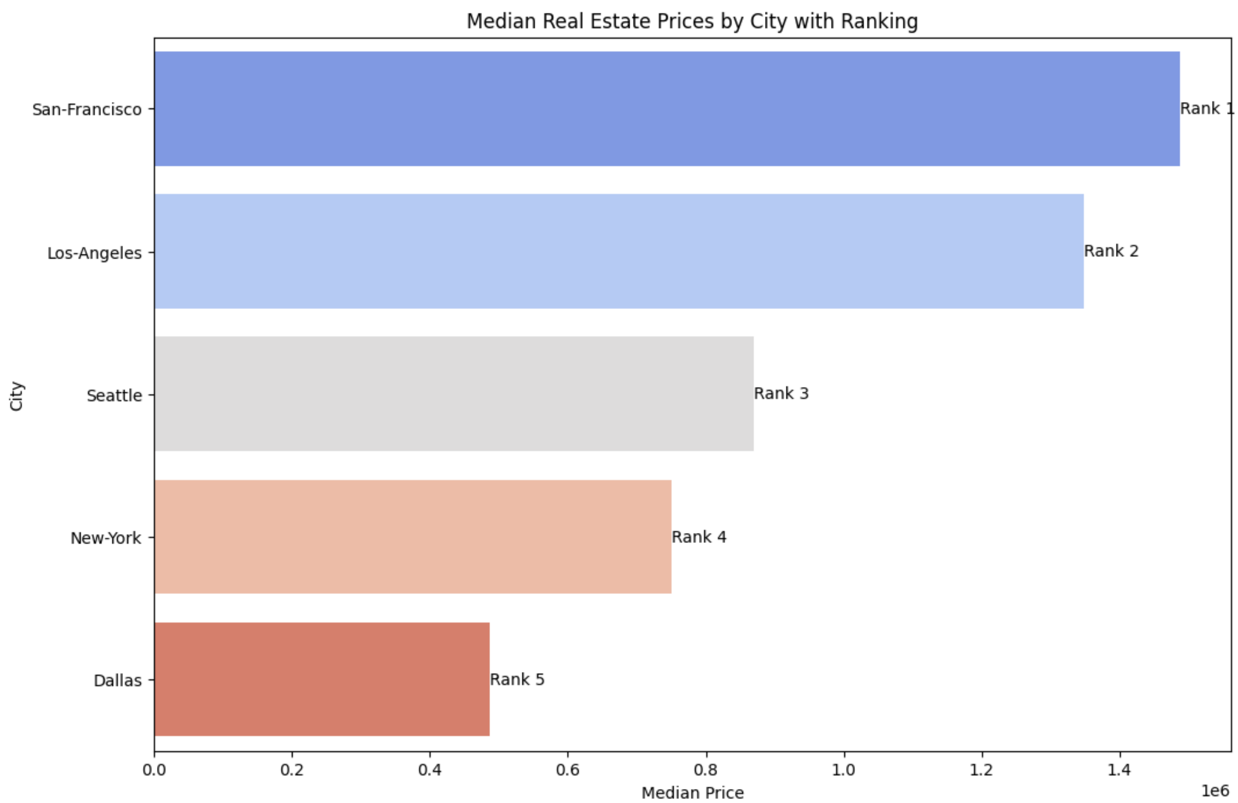
```
of numeric_only to silence this warning.
  corr = merged_df.corr()
```




```python
# Calculate median prices by city and rank them
median_prices_by_city = merged_df.groupby('City')
['Price'].median().sort_values(ascending=False).reset_index()

# Add rank based on median price
median_prices_by_city['Rank'] = median_prices_by_city.index + 1

# Plotting median prices by city with ranking
plt.figure(figsize=(12, 8))
sns.barplot(x='Price', y='City', data=median_prices_by_city,
palette='coolwarm')
plt.title('Median Real Estate Prices by City with Ranking')
plt.xlabel('Median Price')
```

```
plt.ylabel('City')

# Annotate ranks on the bars
for index, row in median_prices_by_city.iterrows():
    plt.text(row['Price'], index, f'Rank {row["Rank"]}',
color='black', ha="left", va="center")

plt.show()

median_prices_by_city
```

```
/var/folders/lx/4_g1bf5951j3ls64g8b047yr0000gn/T/
ipykernel_62777/1247306549.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `y` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x='Price', y='City', data=median_prices_by_city,
palette='coolwarm')
```




```
              City      Price  Rank
0  San-Francisco  1487500.0     1
1    Los-Angeles  1349000.0     2
2        Seattle   869500.0     3
```

```
3       New-York   750000.0     4
4         Dallas   486999.5     5
```

2

https://api.developer.attomdata.com/docs#!/Valuation32V1/assessmentHistoryDetailID

```
url
="https://api.gateway.attomdata.com/propertyapi/v1.0.0/property/id?
geoid=PL0820000&minBeds=1"
headers = {
    "accept": "application/json",
    "apikey": "37b77047fa1778ca3c56c8871e08a387"
}
response = requests.get(url, headers=headers)
json_data=response.json()

ids = [property["identifier"]["Id"] for property in
json_data["property"]]

ids
```

```
[143367,
 143382,
 143899,
 144393,
 144394,
 145233,
 145234,
 145235,
 146764,
 147146]
```

```
for attomId in ids:
    url =
"https://api.gateway.attomdata.com/propertyapi/v1.0.0/assessmenthistor
y/detail"
    params = {"attomId": attomId}
    response = requests.get(url, headers=headers, params=params)

    if response.status_code == 200:
        json_data = response.json()
        assessment_history = json_data['property'][0]
['assessmenthistory']
```

```
        historical_prices = []
        for history in assessment_history:
            assessed_info = {
                'year': history.get('tax', {}).get('assessorYear'),
                'assessed_improvement_value': history.get('assessed',
{}).get('assdImprValue'),
                'assessed_land_value': history.get('assessed',
{}).get('assdLandValue'),
                'total_assessed_value': history.get('assessed',
{}).get('assdTtlValue'),
                'market_improvement_value': history.get('market',
{}).get('mktImprValue'),
                'market_land_value': history.get('market',
{}).get('mktLandValue'),
                'total_market_value': history.get('market',
{}).get('mktTtlValue'),
                'tax_amount': history.get('tax', {}).get('taxAmt'),
            }
            historical_prices.append(assessed_info)

        df = pd.DataFrame(historical_prices)
        csv_file_path = f'attomId_{attomId}_historical_prices.csv'
        df.to_csv(csv_file_path, index=False)
    else:
        print(f"Error fetching historical information for attomId
{attomId}: {response.status_code}")
```

3

```
import pandas as pd
import numpy as np

# Load the dataset
file_path = 'real_estate_data.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataframe to understand its
structure and content
data.tail(50)
print(data.columns)
```

```
Index(['City', 'State', 'Cost of Living Index', 'Median Income',
       'Crime Index ', 'Price', 'Beds', 'Baths', 'Area (sqft)', 'Lot
Size'],
      dtype='object')
```

```
# Remove currency symbols and commas from 'Price', then convert to
numeric
data['Price'] = data['Price'].replace('[\$,]', '',
regex=True).astype(float)

# Convert 'Beds', 'Baths', and 'Lot Size' to numeric, handling missing
values as NaN
columns_to_numeric = ['Beds', 'Baths', 'Lot Size']
data[columns_to_numeric] = data[columns_to_numeric].replace('None',
np.nan).apply(pd.to_numeric, errors='coerce')

# Remove commas from 'Area (sqft)' and convert to numeric
data['Area (sqft)'] = data['Area (sqft)'].str.replace(',',
'').astype(float)

data.to_csv('cleaned_data.csv', index=False)


columns_to_remove = ['Cost of Living Index', 'Median Income','Crime
Index ']
data = data.drop(columns_to_remove, axis=1)

# Re-check the cleaned data
cleaned_data_info = data.info()
cleaned_data_head = data.head()

cleaned_data_info, cleaned_data_head
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1189 entries, 0 to 1188
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   City         1189 non-null   object
 1   State        1189 non-null   object
 2   Price        1189 non-null   float64
 3   Beds         1107 non-null   float64
 4   Baths        1051 non-null   float64
 5   Area (sqft)  1050 non-null   float64
 6   Lot Size     314 non-null    float64
dtypes: float64(5), object(2)
memory usage: 65.1+ KB
```

```
(None,
           City       State        Price  Beds  Baths  Area (sqft)
Lot Size
 0  Los-Angeles  California  139000000.0  12.0   17.0          NaN
2.08
 1  Los-Angeles  California  155000000.0  14.0    NaN      56500.0
```

```
4.60
 2  Los-Angeles  California  195000000.0   7.0   20.0          NaN
8.40
 3  Los-Angeles  California   85000000.0  13.0   16.0      28000.0
2.20
 4  Los-Angeles  California    3395000.0   5.0    6.0       4816.0
NaN)


data_cleaned = data.dropna()

data_cleaned_info = data_cleaned.info()
data_cleaned_head = data_cleaned.head()

data_cleaned_info, data_cleaned_head

<class 'pandas.core.frame.DataFrame'>
Int64Index: 261 entries, 3 to 1186
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   City         261 non-null    object
 1   State        261 non-null    object
 2   Price        261 non-null    float64
 3   Beds         261 non-null    float64
 4   Baths        261 non-null    float64
 5   Area (sqft)  261 non-null    float64
 6   Lot Size     261 non-null    float64
dtypes: float64(5), object(2)
memory usage: 16.3+ KB

(None,
            City       State        Price  Beds  Baths  Area (sqft)
Lot Size
 3   Los-Angeles  California   85000000.0  13.0   16.0      28000.0
2.20
 5   Los-Angeles  California  126000000.0   8.0   20.0      30610.0
9.90
 6   Los-Angeles  California    9500000.0   5.0    7.0       9375.0
0.74
 14  Los-Angeles  California     789000.0   3.0    3.0       1902.0
0.36
 15  Los-Angeles  California    1050000.0   3.0    2.0       1914.0
0.46)

# Calculate price per square foot
data_cleaned['Price_per_sqft'] = data_cleaned['Price'] /
data_cleaned['Area (sqft)']

# Display the first few rows of the data with the calculated price per
```

```python
square foot
data_cleaned[['Price', 'Area (sqft)', 'Price_per_sqft']].head()
```

```
/var/folders/lx/4_g1bf5951j3ls64g8b047yr0000gn/T/
ipykernel_62777/1802613730.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data_cleaned['Price_per_sqft'] = data_cleaned['Price'] /
data_cleaned['Area (sqft)']
```

	Price	Area (sqft)	Price_per_sqft
3	85000000.0	28000.0	3035.714286
5	126000000.0	30610.0	4116.301862
6	9500000.0	9375.0	1013.333333
14	789000.0	1902.0	414.826498
15	1050000.0	1914.0	548.589342

```python
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.ensemble import GradientBoostingRegressor,
RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Dropping the 'Lot Size' column
data_preprocessed = data_cleaned.drop(columns=['Lot Size'])

# Imputing missing values for 'Beds', 'Baths', and 'Area (sqft)'
imputer = SimpleImputer(strategy='mean')

# Encoding categorical variables
categorical_features = ['City', 'State']
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

# Setting up preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', categorical_transformer, categorical_features),
        ('imputer', imputer, ['Beds', 'Baths', 'Area (sqft)'])
    ],
    remainder='passthrough'
)

# Splitting the data into features and target variable
X = data_preprocessed.drop('Price', axis=1)
```

```
y = data_preprocessed['Price']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Gradient Boosting Regressor pipeline
gbr_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('regressor',
GradientBoostingRegressor(random_state=42))])

# Random Forest Regressor pipeline
rfr_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('regressor',
RandomForestRegressor(random_state=42))])

# Training the models
gbr_pipeline.fit(X_train, y_train)
rfr_pipeline.fit(X_train, y_train)

# Making predictions and evaluating
gbr_predictions = gbr_pipeline.predict(X_test)
gbr_mse = mean_squared_error(y_test, gbr_predictions)
gbr_rmse = gbr_mse ** 0.5

rfr_predictions = rfr_pipeline.predict(X_test)
rfr_mse = mean_squared_error(y_test, rfr_predictions)
gbr_rmse = rfr_mse ** 0.5

print(f"Gradient Boosting Regressor RMSE: {gbr_rmse}")
print(f"Random Forest Regressor RMSE: {rfr_rmse}")
```

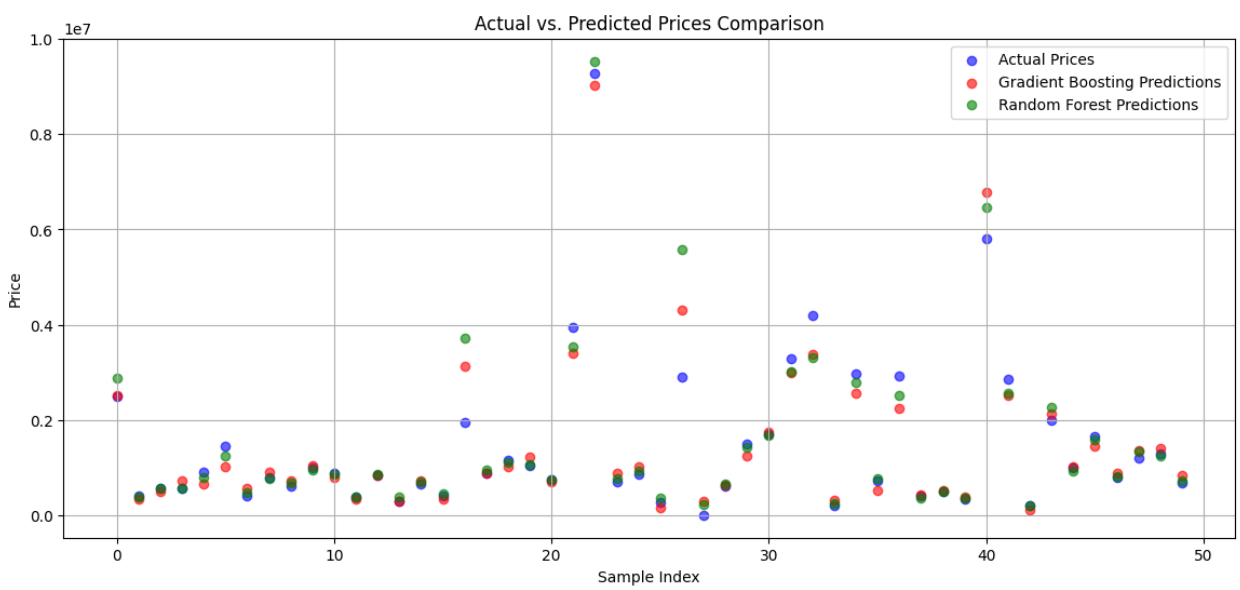
```
Gradient Boosting Regressor RMSE: 596821.0123913316
Random Forest Regressor RMSE: 6863261.678899245
```

```
from sklearn.metrics import r2_score

# Calculating R-squared for both models
gbr_r2 = r2_score(y_test, gbr_predictions)
rfr_r2 = r2_score(y_test, rfr_predictions)

print(f"Gradient Boosting Regressor R-squared: {gbr_r2}")
print(f"Random Forest Regressor R-squared: {rfr_r2}")
```

```
Gradient Boosting Regressor R-squared: 0.916629882541568
Random Forest Regressor R-squared: 0.8602937386705457
```

```
import matplotlib.pyplot as plt
import numpy as np
```

```python
# Select a subset of the test data for visualization
subset_size = 50  # Choose a manageable number for clear visualization
indices = np.random.choice(range(len(y_test)), size=subset_size,
replace=False)
y_test_subset = y_test.iloc[indices]
gbr_predictions_subset = gbr_predictions[indices]
rfr_predictions_subset = rfr_predictions[indices]

# Plotting the actual vs. predicted prices
plt.figure(figsize=(14, 6))

plt.scatter(range(subset_size), y_test_subset, color='blue',
label='Actual Prices', alpha=0.6)
plt.scatter(range(subset_size), gbr_predictions_subset, color='red',
label='Gradient Boosting Predictions', alpha=0.6)
plt.scatter(range(subset_size), rfr_predictions_subset, color='green',
label='Random Forest Predictions', alpha=0.6)

plt.title('Actual vs. Predicted Prices Comparison')
plt.xlabel('Sample Index')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```



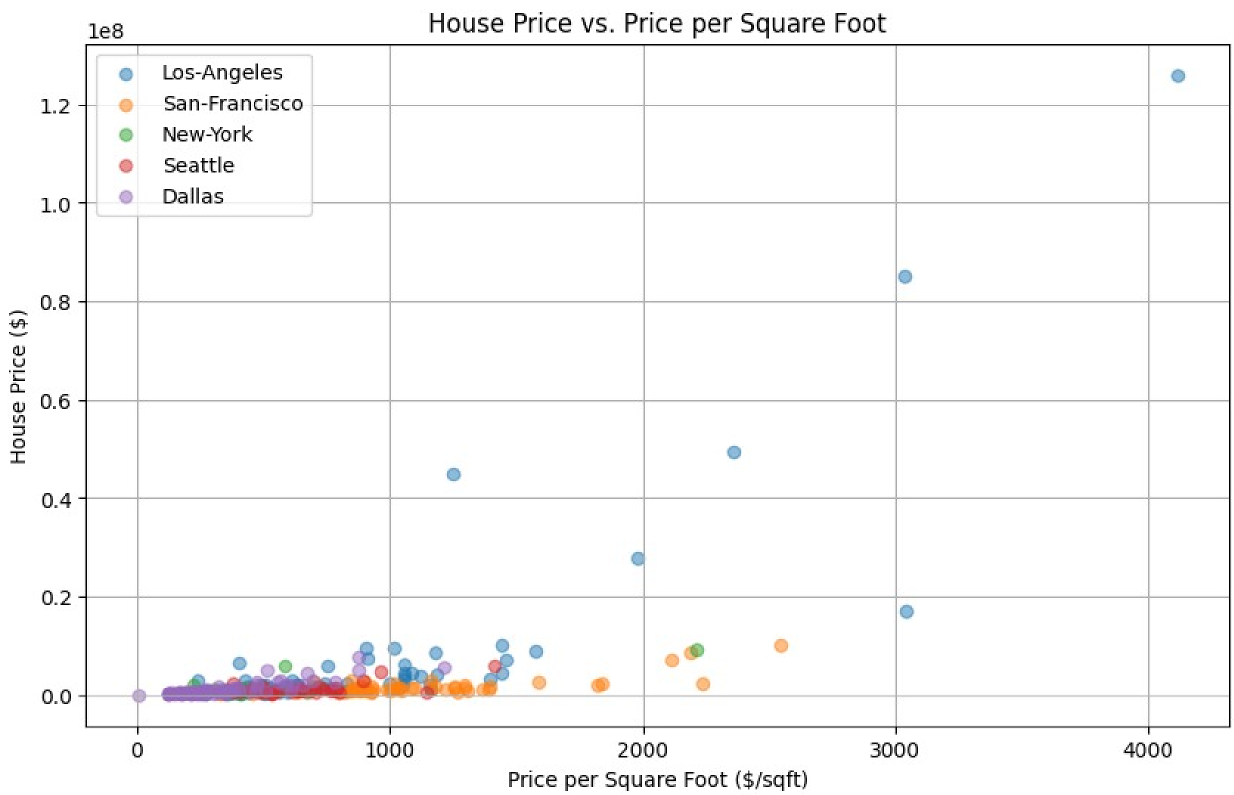

```python
import matplotlib.pyplot as plt

# Create a scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(data_cleaned['Price_per_sqft'], data_cleaned['Price'],
```

```
alpha=0.5)
plt.title('House Price vs. Price per Square Foot')
plt.xlabel('Price per Square Foot ($/sqft)')
plt.ylabel('House Price ($)')
plt.grid(True)
plt.show()
```


```
import matplotlib.pyplot as plt

# Calculate price per square foot considering the city
data_cleaned['Price_per_sqft'] = data_cleaned['Price'] /
data_cleaned['Area (sqft)']

# Create a scatter plot with different colors for each city
plt.figure(figsize=(10, 6))
for city in data_cleaned['City'].unique():
    city_data = data_cleaned[data_cleaned['City'] == city]
    plt.scatter(city_data['Price_per_sqft'], city_data['Price'],
alpha=0.5, label=city)

plt.title('House Price vs. Price per Square Foot')
plt.xlabel('Price per Square Foot ($/sqft)')
plt.ylabel('House Price ($)')
plt.legend()
```

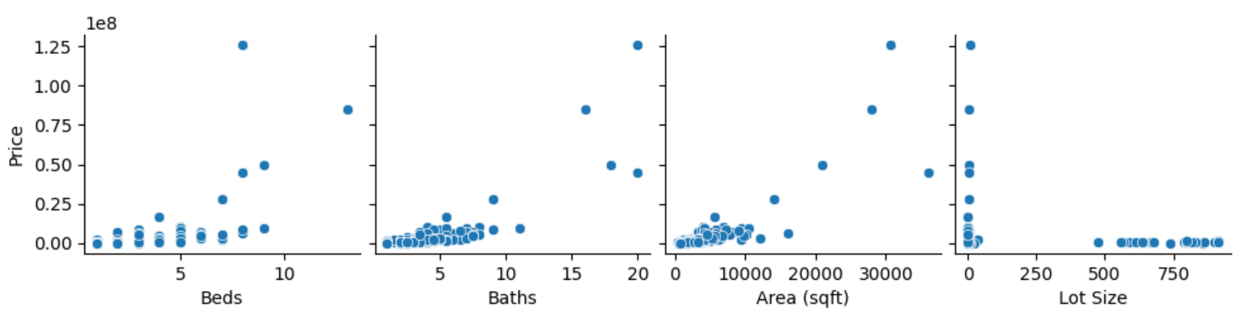
```
plt.grid(True)
plt.show()
```

```
/var/folders/lx/4_g1bf5951j3ls64g8b047yr0000gn/T/
ipykernel_62777/1675166368.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#
returning-a-view-versus-a-copy
  data_cleaned['Price_per_sqft'] = data_cleaned['Price'] /
data_cleaned['Area (sqft)']
```




```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import seaborn as sns

# Select relevant features (factors) and target variable (price)
X = data_cleaned[['Beds', 'Baths', 'Area (sqft)', 'Lot Size']]
y = data_cleaned['Price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
```

```
test_size=0.2, random_state=42)

# Train linear regression model
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train, y_train)

# Train multiple linear regression model
multiple_linear_reg_model = LinearRegression()
multiple_linear_reg_model.fit(X_train, y_train)

# Evaluate the models
linear_reg_train_rmse = mean_squared_error(y_train,
linear_reg_model.predict(X_train), squared=False)
linear_reg_test_rmse = mean_squared_error(y_test,
linear_reg_model.predict(X_test), squared=False)

multiple_linear_reg_train_rmse = mean_squared_error(y_train,
multiple_linear_reg_model.predict(X_train), squared=False)
multiple_linear_reg_test_rmse = mean_squared_error(y_test,
multiple_linear_reg_model.predict(X_test), squared=False)

# Display the root mean squared error (RMSE) for both models
print("Linear Regression Train RMSE:", linear_reg_train_rmse)
print("Linear Regression Test RMSE:", linear_reg_test_rmse)
print("Multiple Linear Regression Train RMSE:",
multiple_linear_reg_train_rmse)
print("Multiple Linear Regression Test RMSE:",
multiple_linear_reg_test_rmse)

# Visualize the relationships between factors and price using pairplot
sns.pairplot(data_cleaned, x_vars=['Beds', 'Baths', 'Area (sqft)',
'Lot Size'], y_vars=['Price'])
plt.show()

Linear Regression Train RMSE: 5846488.193332167
Linear Regression Test RMSE: 3047699.9940630407
Multiple Linear Regression Train RMSE: 5846488.193332167
Multiple Linear Regression Test RMSE: 3047699.9940630407
```



```
import pandas as pd

file_path = "attomId_147146_historical_prices.csv"
data = pd.read_csv(file_path)

print(data.head())

   year  assessed_improvement_value  assessed_land_value  \
0  2015                       12620                 1740
1  2014                        8110                 1990
2  2013                        8110                 1990
3  2011                        8651                 2189
4  2022                       21640                 3950

   total_assessed_value  market_improvement_value
market_land_value  \
0                 14360                  158500.0            21900.0

1                 10100                  101900.0            25000.0

2                 10100                  101900.0            25000.0

3                 10840                       NaN                NaN

4                 25590                  311400.0            56900.0


   total_market_value  tax_amount
0              180400     1490.95
1              126900     1167.10
2              126900     1167.46
3              136200     1125.27
4              368300     2492.26

import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt

file_path = "attomId_147146_historical_prices.csv"
data = pd.read_csv(file_path)

data['year'] = pd.to_datetime(data['year'], format='%Y')
data.set_index('year', inplace=True)

# Sort the index and set a frequency
data = data.sort_index().asfreq('AS')

model = ARIMA(data['total_market_value'], order=(1,3,1))
results = model.fit()
```

```
forecast = results.forecast(steps=10)

plt.plot(data.index, data['total_market_value'], label='Original')
plt.plot(pd.date_range(start=data.index[-1], periods=11, freq='AS')
[1:], forecast, label='Forecast', color='red')
plt.legend()
plt.show()
```