meta-real-estate

July 17, 2024

Metaverse Real Estate: An Analysis

1 Introduction & Background

Unlike traditional real estate, ‘metaverse real estate’ (or ‘virtual/digital real estate’) is often questioned for its authenticity. Ruby Home’s article, “Metaverse Real Estate Statistics (2024)”, defines these properties as 3D virtual spaces owned via non-fungible tokens (NFTs), valued between 3300 and 16000 USD in 2022 [1].

Similar to physical real estate, these virtual lands appreciate or depreciate, can be developed, and rented out, providing passive income. Discussions around regulating ‘property rights’ are emerging to sustain the metaverse markets (Laborde et al., 2023) [2]. As noted by Heeseung Yu, the metaverse hosts an economic system linked to the real world, allowing real-world money to be invested in and profits to be extracted from the metaverse (Yu, 2024) [3].

Governments, such as China with its Three-Year Action Plan for the Industrial Innovation and Development of the Metaverse (2023–2025), are also investing in their own version of digital space (Gong & Xue, 2024) [4].

But why invest in virtual real estate? Yu suggests users can earn income and gain valuable assets through unique economic activities in the metaverse, unlike in real life (Yu, 2024) [5]. This shift might be driven by real-world constraints, such as the rising cost of physical real estate.

I chose to examine the connection between physical and virtual real estate by conducting a quantitative analysis of their costs. My goal is to see if house prices and purchasing power influence the motivation to buy virtual land. I use international real estate data for the physical world and data from Decentraland, known for its pioneering role and extensive history in virtual real estate. Additionally, I perform sentiment analysis on textual data collected from the social media platform X to better understand people’s general feelings surrounding virtual real estate.

2 Aims & Objectives

2.1 Project Aim

The primary goal of this project is to address the following questions:

- How much has traditional real estate increased in value over the years?
- Can people afford today’s real estate? How does purchasing power compare to real estate value?

- Is there a relationship between the rise in physical real estate prices and the trend of virtual real estate buying?
- What is the current value of virtual real estate?
- Who is interested in virtual real estate, and what are their sentiments?

2.2 Objectives

To answer these questions, I will:

1. Analyze the increase in house prices over time.
2. Determine the average price of real estate.
3. Analyze the increase in purchasing power over time.
4. Determine the average price of virtual real estate.
5. Perform sentiment analysis on social media data about virtual real estate.
6. Identify which countries are discussing this subject.

3 Ethical Statement

This project utilizes publicly available data to analyze real estate market trends and perform sentiment analysis on X data. The datasets include information from Kaggle, Decentraland's LAND API and other open sources like the Federal Reserve Bank of Dallas which require proper citation and is inlcuded below. I have abided by the 'Terms of Use' of these sources, ensuring all data is used ethically and responsibly.

One of the datasets includes tweets collected from X using the X API, which was shared on Kaggle under the CC0 1.0 Universal license, placing it in the public domain. My analysis of the X data focused on sentiment analysis and the geographical distribution of tweets without disclosing personal information, such as names or other identifiers. Only the locations and usernames, which are publicly available on X, were included in the dataset.

No web scraping techniques were employed; all data was acquired through legitimate channels, respecting the privacy and rights of the data providers.

My analysis aims to derive aggregate insights, such as average prices and sentiment trends, without revealing sensitive personal information.

4 Data (Acquisition, Cleansing & Analysis)

4.1 Metadata

4.1.1 Data & Source

- **International House Price Database (IHPD)**: HPI and PDI data for countries around the globe (from 1975 to 2023) - *Federal Reserve Bank of Dallas*
- **Decentraland LAND API**: prices of parcels from Decentraland - *Decentraland*
- **Metaverse Tweets**: tweets collected from X containing the hashtag #Metaverse - *Kaggle*
- **Countries**: a list of all countries in the world - *GitHub*
- **States**: a list of all states in the US - *GitHub*
- **CoinGecko API**: exchange rates for different fiat/crypto currencies - CoinGecko

4.1.2 Motivation for Data Selection

I chose to work with the International House Price Database (IHPD) because it provides HPI and PDI data for many countries around the globe and is not confined to one or a few countries. This will be helpful to build a clearer image of global real estate prices since everyone in the world has access to metaverse technology and this analysis is focused around anyone who has the potential to buy virtual real estate.

Decentraland's LAND API is important becuase it provides up-to-date virtual real estate prices, but most importantly, I chose them because they are the ones to have pioneered virtual real estate. [6]

Meteverse tweets was chosen because vitual real estate falls under the umbrella of the 'Metaverse'. X being far reaching platform around the world, I hope to find some insight into people's thoughts around the topic.

4.1.3 Shortcomming of the Data

Neither of these datasets cover the entire globe. The International House Price Database (IHPD) doesn't have data on every country in the world, Decentraland isn't the only metaverese but one amongst many, and many people's thoughts and opinions are not shared on X. Please keep this in mind as you read through this paper, I strived to make this analysis as universal as possible but I do not wish to protray it as fully universal.

Time frames may differ. The IHPD covers 1975 through 2023, The Decentraland data is up-to-date because it fetches data from an API, while the metaverse tweets from X were collected in 2021.

4.1.4 Related Data & Research

The following resources where discovered during the making of this notebook and are related to some degree but do not cover in full the scope of this research. More data and research exists out there.

Related Data

- Metaverse Financial Transactions Dataset - Kaggle
  - Covers all transactions in the metaverse without identifying which ones are related to virtual real estate.
- Real Residential Property Prices for United States - Federal Reserve Bank of St. Louis
  - Covers real estate data in the US only.
- Decentraland NFT Virtual Estate Dataset - Kaggle
  - Covers virtual real estate data in 2022 only.

Related Research

- Metaverse in Real Estate Market - Precedence Research
- Real Estate in the Metaverse - Politecnico
- Digital real estate in the metaverse: An empirical analysis of retail investor motivations - ScienceDirect

4.1.5 Dictionary

- **House Price Index (HPI)**: a house price index measures the price changes of residential housing as a percentage change from some specific start date (which has an HPI of 100). [7]
- **Personal Disposable Income (PDI)**: the income that's left after people pay their taxes is disposable personal income, also known as after-tax income. It's the amount that residents are able to spend, save, or invest. This number is important not just to individuals' well-being but also to the whole economy. [8]
- **Base Year Value**: the value assigned to a specific year in the dataset, chosen as a reference point for comparison.
- **Nominal Value**: a value measured in terms of absolute money amounts. [9]
- **Decentraland**: a 3D virtual world browser-based platform. Users may buy virtual plots of land in the platform as NFTs via the MANA cryptocurrency. Designers can create and sell clothes and accessories for the avatars to be used in the virtual world. [10]
- **Parcel**: a plot of virtual land that can be bought, sold, leased, or developed. These parcels are similar to real-world land, but are represented by digital constructs, such as 3D spaces or immersive environments. [11]
- **MANA**: a digital asset token used to pay for goods and services in Decentraland. It is built on Ethereum and can be bought and sold for fiat currency or other digital currencies. [12]
- **wei**: 1 MANA = 10^{18} wei

4.1.6 Libraries & Functions

[1]:
```python
#import libraries and modules
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import requests
import json
import os
import time
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import warnings
```

The following functions read and write .csv files with proper error handling. They will be used throughout the notebook.

[2]:
```python
#load data from a .csv file
def load_csv(file_path):
    try:
        return pd.read_csv(file_path)
    except FileNotFoundError:
        print(f"Error: .csv file not found at {file_path}")
        return None
    except pd.errors.EmptyDataError:
        print(f"Error: Empty .csv file at {file_path}")
        return None
    except pd.errors.ParserError as e:
        print(f"Error: Parsing .csv file at {file_path} failed - {e}")
        return None
#save data to a .csv file
def save_csv(data, file_path):
    try:
        data.to_csv(file_path, index=False)
        print(f"Data saved to: {file_path}")
        return True
    except Exception as e:
        print(f"Error: Failed to save .csv file to {file_path} - {e}")
        return False
```

If you have installed all packages using the `requirements.txt`, the `transformer` package will flag a `future warning` during the sentiment analysis that can be ignored, you can ignore this by leaving the code below uncommented. If you need to fix the warning, go to `.../metarealestate/lib/python3.12/site-packages/transformers/utils/hub.py`, comment out or remove line 409: `resume_download=resume_download`. `resume_download` is a depreciated arguement in the `huggingface_hub` package. It does not need to be replaced.

[3]:
```
#ignore future warnings
warnings.filterwarnings("ignore", category=FutureWarning,␣
 ↪module="huggingface_hub")
```

4.2 Global Real Estate (International House Price Database)

Both the HPI and PDI values are available for many countries from 1975 to 2023. Each year is also broken down in four quarters.

4.2.1 Loading and cleaning IHPD

The IHPD data was in the form of an Excel Workbook and was converted to seprate worksheets in .csv format prior to being loaded into the application.
Here we load in both the HPI and PDI data from their .csv files and check the data.

[4]:
```
#load HPI and PDI data
HPI_data = load_csv('data/IHPD-FRBD/HPI.csv')
PDI_data = load_csv('data/IHPD-FRBD/PDI.csv')
#check
HPI_data.head()
```

[4]:
```
                Australia  Belgium  Canada  Switzerland  Germany  Denmark  Spain  \
0        NaN          NaN      NaN     NaN          NaN      NaN      NaN    NaN
1  1975:Q1           7.59    15.18   16.23        48.83    51.99    15.77   8.67
2  1975:Q2           7.75    15.93   16.45        48.19    52.60    16.19   9.71
3  1975:Q3           8.05    16.65   17.16        47.73    53.25    17.00   9.91
4  1975:Q4           8.29    17.57   17.40        47.14    53.96    17.14  10.65

   Finland  France  …     US  S. Africa  Croatia  Israel  Slovenia  \
0      NaN     NaN  …    NaN         NaN       NaN      NaN        NaN
1    13.49   11.10  …  17.19        3.56       0.0      0.0        0.0
2    13.64   11.50  …  17.39        3.63       0.0      0.0        0.0
3    13.83   11.91  …  17.50        3.64       0.0      0.0        0.0
4    14.20   12.39  …  17.84        3.72       0.0      0.0        0.0

   Colombia  Portugal  Unnamed: 27  Aggregate - 2005 Fixed Weights  \
0       NaN       NaN          NaN                             NaN
1      1.54      3.30          NaN                           22.24
2      1.55      3.48          NaN                           22.51
3      1.58      3.66          NaN                           22.75
4      1.62      3.86          NaN                           23.10

   Aggregate - Dynamic Weights
0                          NaN
1                        22.93
2                        23.19
3                        23.43
4                        23.77

[5 rows x 30 columns]
```

As you can see, there is an empty row at index 0 that that is filled with `NA / NaN` values and needs to be dropped. At the far right, there is another empty column and two others called `'Aggregate - 2005 Fixed Weights'` and `'Aggregate - Dynamic Weights'` that can also be dropped. Their use is out of the scope of this analysis. Another thing we can do in order to better organize the data is to change the index to the first column, which separates the rows by years divided into four quarters.

[5]:
```
#remove futile rows and columns and set the index
def clean_data(data):
    #drop empty row
    data.drop(index=0, inplace=True)
    #drop unnecessary columns
    columns_to_drop = ['Aggregate - Dynamic Weights', 'Aggregate - 2005 Fixed␣
 ↪Weights', 'Unnamed: 27']
    data.drop(columns=columns_to_drop, inplace=True)
    #set the first column as the index
    data.set_index(data.columns[0], inplace=True)

#clean HPI & PDI data
clean_data(HPI_data)
clean_data(PDI_data)
```

We check the data to see if our data is clean and ready for processing

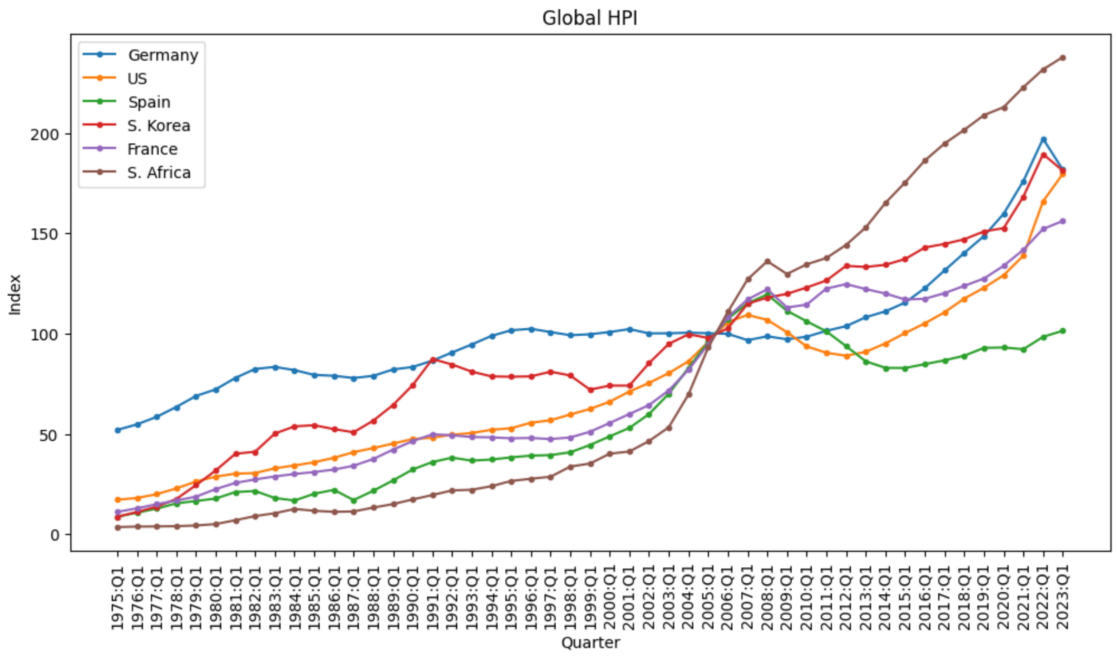
[6]:
```
#check
HPI_data.head()
```

[6]:
```
         Australia  Belgium  Canada  Switzerland  Germany  Denmark  Spain  \

1975:Q1       7.59    15.18   16.23        48.83    51.99    15.77   8.67
1975:Q2       7.75    15.93   16.45        48.19    52.60    16.19   9.71
1975:Q3       8.05    16.65   17.16        47.73    53.25    17.00   9.91
1975:Q4       8.29    17.57   17.40        47.14    53.96    17.14  10.65
1976:Q1       8.57    18.74   17.64        46.43    54.74    17.64  10.72

         Finland  France    UK  …  Norway  New Zealand  Sweden     US  \
                               …
1975:Q1    13.49   11.10  5.92  …   13.37         7.39   14.58  17.19
1975:Q2    13.64   11.50  6.03  …   13.48         7.43   15.06  17.39
1975:Q3    13.83   11.91  6.15  …   13.72         7.53   15.59  17.50
1975:Q4    14.20   12.39  6.25  …   14.16         7.68   16.17  17.84
1976:Q1    14.42   12.89  6.45  …   14.14         7.83   16.73  18.04

         S. Africa  Croatia  Israel  Slovenia  Colombia  Portugal

1975:Q1       3.56      0.0     0.0       0.0      1.54      3.30
1975:Q2       3.63      0.0     0.0       0.0      1.55      3.48
1975:Q3       3.64      0.0     0.0       0.0      1.58      3.66
1975:Q4       3.72      0.0     0.0       0.0      1.62      3.86
1976:Q1       3.79      0.0     0.0       0.0      1.68      4.09

[5 rows x 26 columns]
```

4.2.2 House Price Index (HPI)

Here we're going to plot the HPI values from 1975 till 2023 for a random handful of countries to get an idea of the growth of house prices. In order to fit it all on one graph, I only took the value of the first quarter of every year for each country removing some detail from the graph without effecting what we're trying to show.

[7]:
```
#plot HPI data as line graph
def plot_hpi_data(data, countries, title='Global HPI'):
    #select every fourth index
    x_val = data.index[::4]
    y_vals = data.loc[::4, countries]
    #create plot
    plt.figure(figsize=(12, 6))
    #plot data for each country
    for country in y_vals.columns:
        plt.plot(x_val, y_vals[country], marker='.', linestyle='-',␣
↪label=country)
    #labels
    plt.xlabel('Quarter')
    plt.ylabel('Index')
    plt.title(title)
    plt.xticks(rotation=90)
    #legend
    plt.legend()
    #plot
    plt.show()

#countries to plot as an example
countries_to_plot = ['Germany', 'US', 'Spain', 'S. Korea', 'France', 'S.␣
 ↪Africa']

#HPI data by year
plot_hpi_data(HPI_data, countries_to_plot)
```



We can see here that pretty much every country's real estate prices have grown quite consistently from 1975 till 2023. A clear trend representing the growth in value of real estate globally.

Average price of real estate based off of HPI Based off the HPI, we can derive an average price for real estate in each country during a certain time period with the data we have. The base year is given by the *Federal Reserve Bank of Dallas* as **2005 = 100%**. According to Wikipedia [13] the average house price in the *United States* as of *January (Q1) 2005* was **283,000$**.
Let's put that value in a variable alongside the HPI value in 2005.

[8]:
```
#4 (quarters) x 30 (years, from 1975 to 2005) = 120
HPI_q1_05 = HPI_data.at[HPI_data.index[120], 'US']
#nominal value from January, 2005 in the US (according to Wikipedia)
```

```
nom_val = 283000
print(f"Nominal value (Jan 2005)': {nom_val}")
print(f"HPI value of US (2005 Q1)': {HPI_q1_05}")
```

```
Nominal value (Jan 2005)': 283000
HPI value of US (2005 Q1)': 95.8
```

First, we need a value for the base year. Once we have that, we can calculate the average price using any HPI value in our dataset. The equation to get the base value is below:

$$\begin{aligned}\text{Base Year Value} &= \frac{\text{Nominal Value} \times 100}{\text{HPI}} \\ &= \frac{283000 \times 100}{95.8}\end{aligned}$$

```
[9]: #calculate
     base_year_value = (nom_val * 100) / HPI_q1_05
     #2005 = 100%
     base_year_value
```

```
[9]: 295407.0981210856
```

The **Base Year Value** real estate in Q1 of 2005 is around **295,400$**.
Now let's write out the equation that we will use to get the average price for all others countries, every quarter in every year.

$$\text{Average Price} = \left(\frac{\text{HPI}}{100}\right) \times \text{Base Year Value}$$

```
[10]: #convert HPI to average price
      def HPI_to_avg(index_val, base_val):
          return (index_val / 100) * base_val
      #avergae price data
      avg_data = HPI_data.apply(lambda col: col.apply(lambda x: round(HPI_to_avg(x,␣
       ↪base_year_value), 2)))
```

Let's check our data again to see the changes. All values are now in the form of average price are in USD.

```
[11]: avg_data.head()
```

```
[11]:          Australia    Belgium     Canada  Switzerland    Germany    Denmark  \

1975:Q1   22421.40  44842.80  47944.57    144247.29  153582.15  46585.70
1975:Q2   22894.05  47058.35  48594.47    142356.68  155384.13  47826.41
1975:Q3   23780.27  49185.28  50691.86    140997.81  157304.28  50219.21
1975:Q4   24489.25  51903.03  51400.84    139254.91  159401.67  50632.78
1976:Q1   25316.39  55359.29  52109.81    137157.52  161705.85  52109.81
```

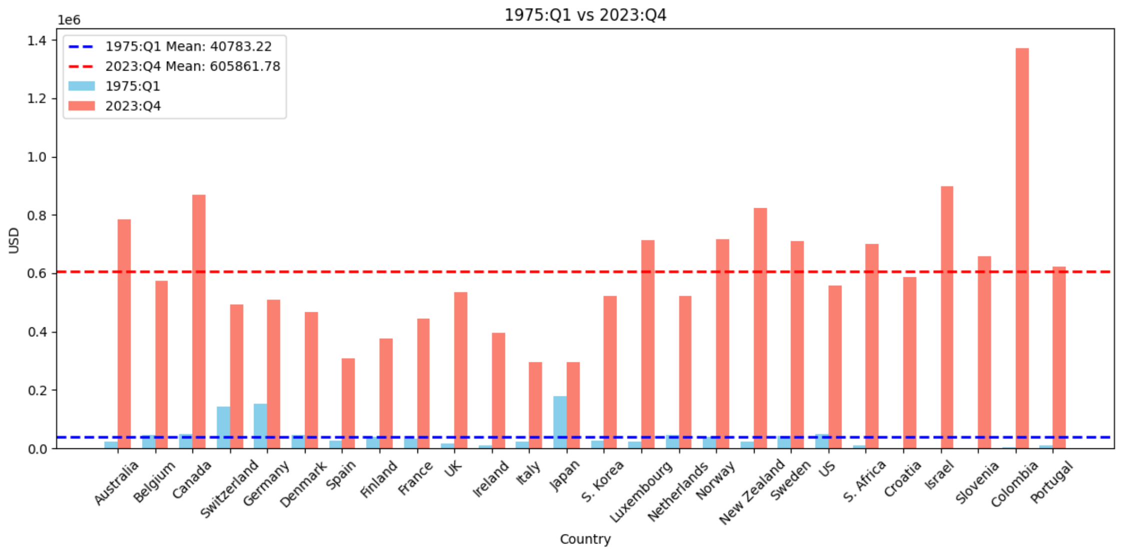
```
              Spain    Finland    France         UK  …     Norway  New Zealand  \
                                                     …
1975:Q1  25611.80  39850.42  32790.19  17488.10  …  39495.93     21830.58
1975:Q2  28684.03  40293.53  33971.82  17813.05  …  39820.88     21948.75
1975:Q3  29274.84  40854.80  35182.99  18167.54  …  40529.85     22244.15
1975:Q4  31460.86  41947.81  36600.94  18462.94  …  41829.65     22687.27
1976:Q1  31667.64  42597.70  38077.97  19053.76  …  41770.56     23130.38

             Sweden        US  S. Africa  Croatia  Israel  Slovenia  Colombia  \

1975:Q1  43070.35  50780.48   10516.49      0.0     0.0       0.0   4549.27
1975:Q2  44488.31  51371.29   10723.28      0.0     0.0       0.0   4578.81
1975:Q3  46053.97  51696.24   10752.82      0.0     0.0       0.0   4667.43
1975:Q4  47767.33  52700.63   10989.14      0.0     0.0       0.0   4785.59
1976:Q1  49421.61  53291.44   11195.93      0.0     0.0       0.0   4962.84

         Portugal

1975:Q1   9748.43
1975:Q2  10280.17
1975:Q3  10811.90
1975:Q4  11402.71
1976:Q1  12082.15

[5 rows x 26 columns]
```

In order to calculate and see how much real estate has grown globally, we'll have to see the difference between the prices back in 1975 and 2023. We can do this by calculating the mean for each of those years and plotting them alongside the averages on the same bar graph. We will show the mean as dotted line across the graph.

[12]:
```
#plots first and last values for each country, and prints sum, mean, and␣
 ↪standard deviation for both
def first_last_graph(data):
    #get indices
    first_index = data.index[0]
    last_index = data.index[-1]
    countries = data.columns
    #values for indices
    first_values = data.loc[first_index, countries]
    last_values = data.loc[last_index, countries]

    #calculate sum, mean, and standard deviation
    first_sum = round(first_values.sum(), 2) #first
    first_mean = round(first_values.mean(), 2)
    first_std_dev = round(first_values.std(), 2)
    last_sum = round(last_values.sum(), 2) #last
    last_mean = round(last_values.mean(), 2)
    last_std_dev = round(last_values.std(), 2)
    #print calculated values
    print(f"{first_index} - Sum: {first_sum}, Mean: {first_mean}, Standard␣
↪Deviation: {first_std_dev}")
    print(f"{last_index} - Sum: {last_sum}, Mean: {last_mean}, Standard␣
↪Deviation: {last_std_dev}")

    #plotting
    fig, ax = plt.subplots(figsize=(12, 6))
    #bar width and positions
    bar_width = 0.35
    index = np.arange(len(countries))
    #bars
    bars1 = ax.bar(index, first_values.values, bar_width, label=first_index,␣
↪color='skyblue')
    bars2 = ax.bar(index + bar_width, last_values.values, bar_width,␣
↪label=last_index, color='salmon')
    #mean lines
    ax.axhline(first_mean, color='blue', linewidth=2, linestyle='--',␣
↪label=f'{first_index} Mean: {first_mean:.2f}')
    ax.axhline(last_mean, color='red', linewidth=2, linestyle='--',␣
↪label=f'{last_index} Mean: {last_mean:.2f}')
    #labels
    ax.set_xlabel('Country')
    ax.set_ylabel('USD')
    ax.set_title(f'{first_index} vs {last_index}')
    ax.set_xticks(index + bar_width / 2)
    ax.set_xticklabels(countries, rotation=45)
    #legend
    ax.legend()
    #layout
    plt.tight_layout()
    #plot
    plt.show()
```

[13]:
```
#plot average price data
first_last_graph(avg_data)
```

```
1975:Q1 - Sum: 1060363.78, Mean: 40783.22, Standard Deviation: 46431.84
2023:Q4 - Sum: 15752406.26, Mean: 605861.78, Standard Deviation: 230676.27
```



Now let's calculate the percentage of growth.

```
[14]: #calculate growth of mean in % from 1975 till 2023
      mean_growth = (553263.73 * 100) / 49812.67 - 100
      round(mean_growth) #round to the nearest integer
```

```
[14]: 1011
```

If we round the means and standards deviation to the nearest thousand we arrive at the following statements:

In 1975, the average price of a house in developing countries around the globe was 41,000$

In 2023, the average price of a house in developing countries around the globe was 606,000$

The average price of a house grew by 1011%

4.2.3 Personal Disposable Income (PDI)

Now that we've seen how much the price of real estate has grown, it's time to see if it is affordable by checking how much purchasing power people had and have.
Again, the base year is given by the *Federal Reserve Bank of Dallas* as **2005 = 100**. According to South Carolina Revenue and Fiscal Affairs Office [14] the PDI in the *United States* in *2005* was **9,043,159,101,000$**. Let's create a variable for that:

```
[15]: #2005 = 100%
      base_year_value_PDI = 9043159101000
```

Now that we have the base year value, we need to write out the equation that will allow us the convert the PDI to a value.

$$\text{Personal Disposable Value} = \left(\frac{\text{PDI}}{100}\right) \times \text{Base Year Value}$$

[16]:
```
#convert PDI to value instead of index
def PDI_to_PDV(index_val, base_val):
    return (index_val / 100) * base_val
#personal disposable value data
PDV_data = PDI_data.apply(lambda col: col.apply(lambda x: round(PDI_to_PDV(x,␣
 ↪base_year_value_PDI), 2)))
```

Let's check the data after the changes. All values are now in USD.

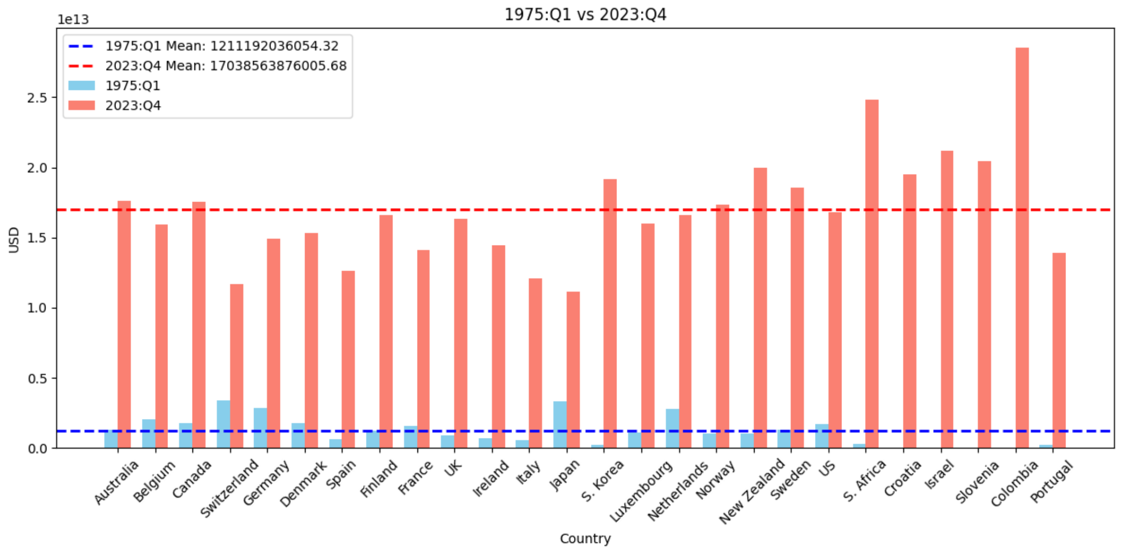
[17]:
```
PDV_data.head()
```

[17]:
```
            Australia       Belgium        Canada   Switzerland       Germany  \

1975:Q1  1.299502e+12  2.025668e+12  1.784215e+12  3.372194e+12  2.879342e+12
1975:Q2  1.327536e+12  2.141420e+12  1.843900e+12  3.368577e+12  2.949878e+12
1975:Q3  1.360091e+12  2.246321e+12  1.922576e+12  3.369481e+12  3.005042e+12
1975:Q4  1.399881e+12  2.338561e+12  1.958748e+12  3.373098e+12  3.046640e+12
1976:Q1  1.436054e+12  2.419949e+12  2.004868e+12  3.379429e+12  3.073770e+12

              Denmark         Spain       Finland        France            UK  \

1975:Q1  1.740808e+12  6.339255e+11  1.196410e+12  1.546380e+12  8.600044e+11
1975:Q2  1.816771e+12  6.538204e+11  1.230774e+12  1.589787e+12  8.762821e+11
1975:Q3  1.890925e+12  6.773326e+11  1.263329e+12  1.649472e+12  9.205936e+11
1975:Q4  1.961461e+12  7.044621e+11  1.295885e+12  1.710061e+12  9.486274e+11
1976:Q1  2.031094e+12  7.361132e+11  1.327536e+12  1.732669e+12  1.000173e+12

         …        Norway   New Zealand        Sweden            US  \
         …
1975:Q1  …  1.015547e+12  1.002886e+12  1.302215e+12  1.672080e+12
1975:Q2  …  1.067093e+12  1.036346e+12  1.346526e+12  1.762512e+12
1975:Q3  …  1.115926e+12  1.068901e+12  1.389934e+12  1.759799e+12
1975:Q4  …  1.165663e+12  1.101457e+12  1.434245e+12  1.794163e+12
1976:Q1  …  1.212688e+12  1.134012e+12  1.478557e+12  1.827622e+12

            S. Africa  Croatia  Israel  Slovenia      Colombia      Portugal

1975:Q1  3.147019e+11      0.0     0.0       0.0  3.255537e+10  2.287919e+11
1975:Q2  3.382142e+11      0.0     0.0       0.0  3.345969e+10  2.324092e+11
1975:Q3  3.418314e+11      0.0     0.0       0.0  3.526832e+10  2.396437e+11
1975:Q4  3.617264e+11      0.0     0.0       0.0  3.707695e+10  2.477826e+11
1976:Q1  3.590134e+11      0.0     0.0       0.0  3.888558e+10  2.595387e+11

[5 rows x 26 columns]
```

We're going to plot the PDI data utilizing the same methods as before to see the differences in averages and means.

```
[18]: #plot personal disposable value data
      first_last_graph(PDV_data)
```

```
1975:Q1 - Sum: 31490992937412.3, Mean: 1211192036054.32, Standard Deviation:
1010640575989.61
2023:Q4 - Sum: 443002660776147.6, Mean: 17038563876005.68, Standard Deviation:
3929853919756.51
```



Let's calculate the percentage of growth.

```
[19]: #calculate the growth of mean in % from 1975 till 2023
      mean_growth = (16604144425346.1 * 100) / 1369173605974.45 - 100
      round(mean_growth) #round to nearest integer
```

```
[19]: 1113
```

If we round the means and standards deviation to the nearest one hundred million we arrive at the following statements:

In 1975, the average PDI value in developing countries around the globe is 1.2 trillion$

In 2023, the average PDI value in developing countries around the globe is 17 trillion$

The average PDI value grew by 1113% … 102% more than the growth of the HPI

4.3 Metaverse Real Estate (Decentraland LAND API)

It's time to compare all this to the virtual world and see how large the difference is in prices. We need to query the Decentraland LAND API in order to get the data we are seeking, which is the current prices of all lands/parcels for sale. We will apply pagination to not overwhelm the server and get blocked in case there are way too many parcels for sale. I've also included a delay between each batch of information.

4.3.1 Parcel Prices

Note that the process might take anywhere between a few seconds to a few minutes depending on how many results there are. If you don't wish to wait, there is pre-saved data that can be used by uncommenting code below.

[20]:
```python
#fetch parcels in batches from API endpoint
def fetch_parcels(url, batch_size=1000, delay=2):
    parcels = []
    skip = 0
    while True:
        #get query
        query = generate_query(batch_size, skip)
        try:
            #POST
            response = requests.post(url, json={'query': query})
            #check for HTTP errors
            response.raise_for_status()
            #parse response
            data = response.json()
            if 'errors' in data:
                raise Exception(f"Query error: {data['errors']}")
            batch = data['data']['nfts']
        except requests.exceptions.RequestException as e:
            print(f"Request error: {e}")
            break
        except Exception as e:
            print(f"An error occurred: {e}")
            break

        if not batch:
            break

        parcels.extend(batch)
        skip += batch_size
        time.sleep(delay)  #delay before fetching next batch in order not to␣
 ↪overwhelm the server
        print('Processing batch...')
    #flag
    print('...Process complete')
    return parcels

#generate query for fetching parcels
def generate_query(batch_size, skip):
    #request (x,y) coordinates and prices of parcels that are for sale
    return f"""
    {{
      nfts(first: {batch_size},
           skip: {skip},
           orderBy: searchOrderPrice,
           where: {{ category: parcel,
                     searchOrderStatus: open,
                     searchOrderExpiresAt_gt: 1611082372 }}) {{
        parcel {{
          x
          y
        }}
        activeOrder {{
          price
        }}
      }}
    }}
    """

#API endpoint
url = "https://api.thegraph.com/subgraphs/name/decentraland/marketplace"

#fetch all parcels
all_parcels = fetch_parcels(url)
```

```
Processing batch…
Processing batch…
…Process complete
```

The code below creates a backup of the data recieved from the endpoint incase the API is ever down. Don't uncomment this unless you wish to save your own copy of fresh results.

[21]:
```
# #path to save .json file
# current_directory = os.getcwd()
# data_folder_path = os.path.join(current_directory, "data")
# path = os.path.join(data_folder_path, "decentraland_data.json")
# #check 'data' directory if it exists
# os.makedirs(data_folder_path, exist_ok=True)

# #save .json file
# def save_json(file_path):
#     try:
#         with open(file_path, 'w') as json_file:
#             json.dump({'parcels': all_parcels}, json_file, indent=4)
#             print(f"Data has been exported to {file_path}")
#     except Exception as e:
#         print(f"Error: Failed to save .json file to {file_path} - {e}")
#         return []

# #save .json to 'data' directory as 'decentraland_data.json'
# save_json(path)
```

You can uncomment the code below in order to load in the backup data that is located in the 'data' directory. Last backup: 06/16/2024.

[22]:
```
# #load .json file
# def load_json(file_path):
#     try:
#         with open(file_path, 'r') as json_file:
#             data = json.load(json_file)
#             return data
#     except Exception as e:
#         print(f"Error: Failed to load .json file from {file_path} - {e}")
#         return []

# #load parcel data
# parcel_data = load_json('data/decentraland_data.json')
```

Let's check our results.

[23]:
```
#print first 5 elements
all_parcels[:5]
```

[23]:
```
[{'parcel': {'x': '-7', 'y': '-116'},
  'activeOrder': {'price': '1099000000000000000000'}},
 {'parcel': {'x': '131', 'y': '-126'},
  'activeOrder': {'price': '1100000000000000000000'}},
 {'parcel': {'x': '-40', 'y': '-140'},
  'activeOrder': {'price': '1200000000000000000000'}},
 {'parcel': {'x': '139', 'y': '-74'},
  'activeOrder': {'price': '1300000000000000000000'}},
 {'parcel': {'x': '139', 'y': '-73'},
  'activeOrder': {'price': '1300000000000000000000'}}]
```

4.3.2 Exchange Rate (WEI to MANA to USD)

All the parcels received are priced in wei, which is a fraction of MANA.

$$1 \text{ MANA} = 10^{18} \text{ wei}$$

Converting wei to MANA is simple becuase it is fixed and is done using one function below. But converting MANA to USD, a currency we're already working with above is bit a more complicated. Due to its volatility we need to access an API with up-to-date crypto-currency information like CoinGecko [15]. We will send a query to their free API and use MANA's exchange rate to convert our values to USD.

[24]:
```python
#convert wei to mana
def wei_to_mana(wei):
    return wei / 10**18

#fetch current exchange rate of mana to usd
def get_mana_to_usd_rate():
    #API endpoint
    url = "https://api.coingecko.com/api/v3/simple/price"
    #query
    params = {
        'ids': 'decentraland',
        'vs_currencies': 'usd'
    }
    try:
        #GET
        response = requests.get(url, params=params)
        #check for HTTP errors
        response.raise_for_status()
        #parse response
        data = response.json()
        mana_usd = data['decentraland']['usd'] #rate
        return mana_usd
    except requests.exceptions.RequestException as e:
        print(f"Request error: {e}")
        return None
    except KeyError as e:
        print(f"Unexpected response structure: {e}")
        return None

#convert wei to usd
def wei_to_usd(wei, rate):
    if rate is None:
        return None
    mana = wei_to_mana(wei)
    usd_value = mana * rate
    return usd_value

#example
mana_to_usd_rate = get_mana_to_usd_rate()
wei_amount = 990000000000000000000
usd_amount = wei_to_usd(wei_amount, mana_to_usd_rate)
if usd_amount is not None: #print
    print(f"{wei_amount} Wei is currently: {usd_amount:.2f}$")
    print(f"The current exchange rate is: {mana_to_usd_rate}$")
else:
    print("Failed to retrieve the exchange rate.")
```

```
999000000000000000000 Wei is currently: 324.85$
The current exchange rate is: 0.325174$
```

Uncomment the line below if API is non-responsive to intitialize the value of `mana_to_usd_rate`. Leave commented if no error message is displayed.

```
[25]: # mana_to_usd_rate = 0.380715 #on 06/16/2023 18:27 (PDT)
```

4.3.3 Restructure Data

Now that we have our data, it's time we restructure it and store it into a dataframe so we can plot it.

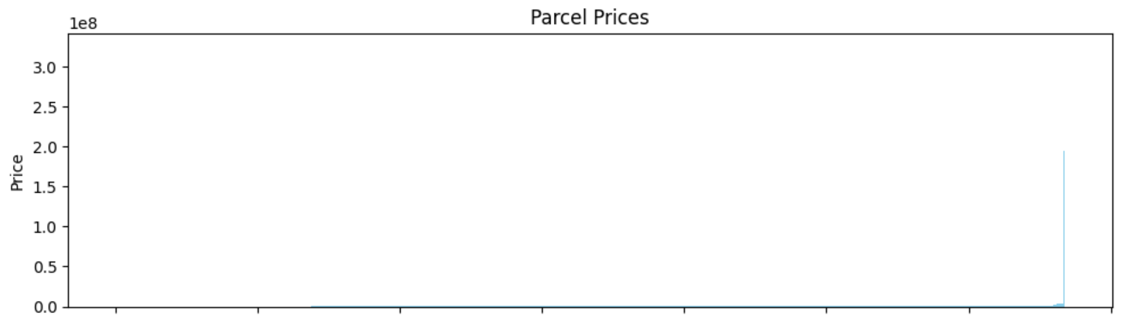
```
[26]: #transform so that parcel data fits into two columns: '(x,y)' and 'price'
parcel_data = {
    '(x, y)': [(int(parcel['parcel']['x']), int(parcel['parcel']['y'])) for␣
  ↪parcel in all_parcels],
    'price': [wei_to_usd(int(parcel['activeOrder']['price']), mana_to_usd_rate)␣
  ↪for parcel in all_parcels]
}
#create parcel dataframe
parcel_df = pd.DataFrame(parcel_data)
parcel_df.head()
```

```
[26]:        (x, y)       price
0    (-7, -116)  357.366226
1   (131, -126)  357.691400
2   (-40, -140)  390.208800
3    (139, -74)  422.726200
4    (139, -73)  422.726200
```

4.3.4 Plot

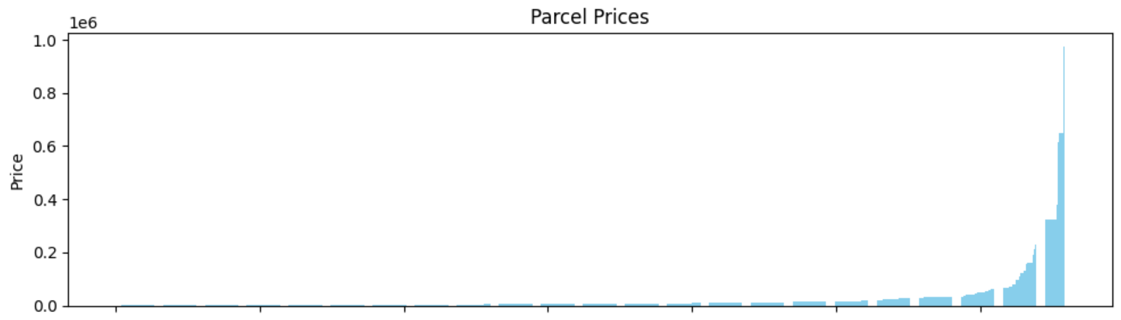
We're going to plot all the data on a bar graph as is in order to get a first impression of the price distribution. Virtual real estate is not a stable market and some lands might be priced extraordinarily high for giggles.

```
[27]: #plot prices of parcels
def plot_parcels(data):
    #extract the 'price' column
    prices = data['price']
    #plotting
    plt.figure(figsize=(10, 3))
    plt.bar(data.index, prices, color='skyblue')
    plt.ylabel('Price')
    plt.title('Parcel Prices')
    #labels
    plt.tick_params(axis='x', labelbottom=False)
    #layout
    plt.tight_layout()
    #plot
    plt.show()

#plot parcel prices
plot_parcels(parcel_df)
```



As expected, some lands are priced so high we can't even see any of the other values. To adjust this, I'm choosing *a bias of 1,000,000$* as the cap for how much a parcel can be priced to get a reasonable mean. Now let's filter out any values larger than the bias.

[28]:
```
#bias
bias = 10**6
#remove values that are extraordinarily high (over 1,000,000 usd)
parcel_df_filtered = parcel_df[parcel_df['price'] <= (bias)]
#replot
plot_parcels(parcel_df_filtered)
```



Many parcels are still priced much lower than our bias, but lowering the bias anymore will cut off a large number of parcels. We calculate our mean below:

[29]:
```
#calculate sum, mean and standard deviation
parcel_sum = parcel_df_filtered.price.sum()
```

```
parcel_mean = parcel_df_filtered.price.mean()
parcel_std = parcel_df_filtered.price.std()

print(f"Parcel Sum: {parcel_sum:.2f}, Parcel Mean: {parcel_mean:.2f}, Parcel␣
 ↪Standard Deviation: {parcel_std:.2f}")
```

```
Parcel Sum: 43163598.14, Parcel Mean: 26191.50, Parcel Standard Deviation:
79169.58
```

In 2024, the average price of virtual real estate in the Decentraland is approximately 30,000$

The term 'approximately' is used here becuase the exchange rate can fluctuate a lot in one year due to volatility

4.4 Social Media (X)

50,000 tweets were collected from the platform X using the hashtag **#Metaverse** as a filter. We will further filter these tweets, perform sentiment analysis and location distribution to better understand how people felt about the topic of virtual real estate and where they come form.

4.4.1 Filter Data

Virtual real estate fits right under the umbrella term **'Metaverse'**, but along with a lot of other things that are unrelated to this analysis. The original data was downloaded and filtered again using keywords like: *virtual land, parcel, property, estate, etc.*
You can leave the code below commented.

[30]:
```
# #load tweets
# metaverse_tweets = load_csv('data/metaverse_tweets.csv')

# #keywords for filtering
# keywords = [
#     'land', 'virtual land', 'virtual real estate', 'estate', 'property',
#     'digital real estate', 'digital land', 'real estate', 'parcel',
#     '#land', '#virtualland', '#virtualrealestate', '#estate', '#property',  #␣
 ↪hashtags
#     '#digitalrealestate', '#digitalland', '#realestate', '#parcel'
# ]

# #filter each tweet based on keywords
# filtered_metaverse_tweets = metaverse_tweets[
#     metaverse_tweets['text'].str.lower().str.contains('|'.join(keywords))
# ].copy() #create a copy instead of view of the original

# #save tweets as a new .csv file
# save_csv(filtered_metaverse_tweets, 'data/filtered_metaverse_tweets.csv'):
```

There were 50,000 tweets before filtering. Only 500 reamined after filtering. Let's load in those 500 and work with them.

[31]:
```
#load filtered metaverse tweets
metaverse_tweets = load_csv('data/filtered_metaverse_tweets.csv')
```

4.4.2 Sentiment Analysis

The code below was inspired by Mehran Shakarami [16]. **roBERTa** is a model that is trained on ~58 million tweet and finetuned for sentiment anaylsis. [17]
For each tweet, we're going to take out usernames and links so that they don't effect our sentiment score. We are then going to run each tweet through the model and produce the output as a dataframe.

Note that this process might take anywhere between a few seconds to a few minutes to complete. There are pre-saved results below if you do not wish to wait. This code might also produce a `future warning` due to an awaited update for the `transformer` package.

[32]:
```
#convert vector of numbers into probability distribution
def vec_to_PD(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum()

#model and tokenizer
roberta = "cardiffnlp/twitter-roberta-base-sentiment"
model = AutoModelForSequenceClassification.from_pretrained(roberta)
tokenizer = AutoTokenizer.from_pretrained(roberta)

#list to store results
results = []

#preprocess tweets
for tweet in metaverse_tweets['text']:
    #preprocess tweet
    tweet_words = []
    for word in tweet.split(' '):
        if word.startswith('@') and len(word) > 1:
            word = ''
        elif word.startswith('http'):
            word = ''
        tweet_words.append(word)
    tweet_proc = ' '.join(tweet_words)
    #sentiment analysis
    encoded_tweet = tokenizer(tweet_proc, return_tensors='pt')
    output = model(**encoded_tweet)
    #model output
    scores = output.logits[0].detach().numpy()
    scores = vec_to_PD(scores)
    #append scores to results
    results.append(scores)
```

```
#scores
metaverse_tweets_with_sentiment = pd.DataFrame(results, columns=['Negative',␣
↪'Neutral', 'Positive'])

#flag
print('...analysis complete')
```

```
…analysis complete
```

Uncomment the line below to save your a copy of fresh results.

```
[33]: # #save metaverse tweeits with sentiment to .csv file in 'data' directory
      # save_csv(metaverse_tweets_with_sentiment, 'data/
      ↪metaverse_tweets_with_sentiment.csv')
```

Uncomment the line below to load in a backup of previously generated results. Last backup: 16/06/2024

```
[34]: # #load backup .csv file
      # metaverse_tweets_with_sentiment = load_csv('data/
      ↪metaverse_tweets_with_sentiment.csv')
```

Let's check our results.

```
[35]: #check
      metaverse_tweets_with_sentiment.head()
```

```
[35]:    Negative   Neutral  Positive
      0  0.005158  0.166955  0.827887
      1  0.460379  0.407318  0.132303
      2  0.006730  0.407362  0.585908
      3  0.012560  0.505947  0.481493
      4  0.004488  0.169177  0.826335
```

In each row we have three results. We need to go through each row, compare the values in each column and keep the largest value. The largest value will represent the general sentiment value of each tweet.

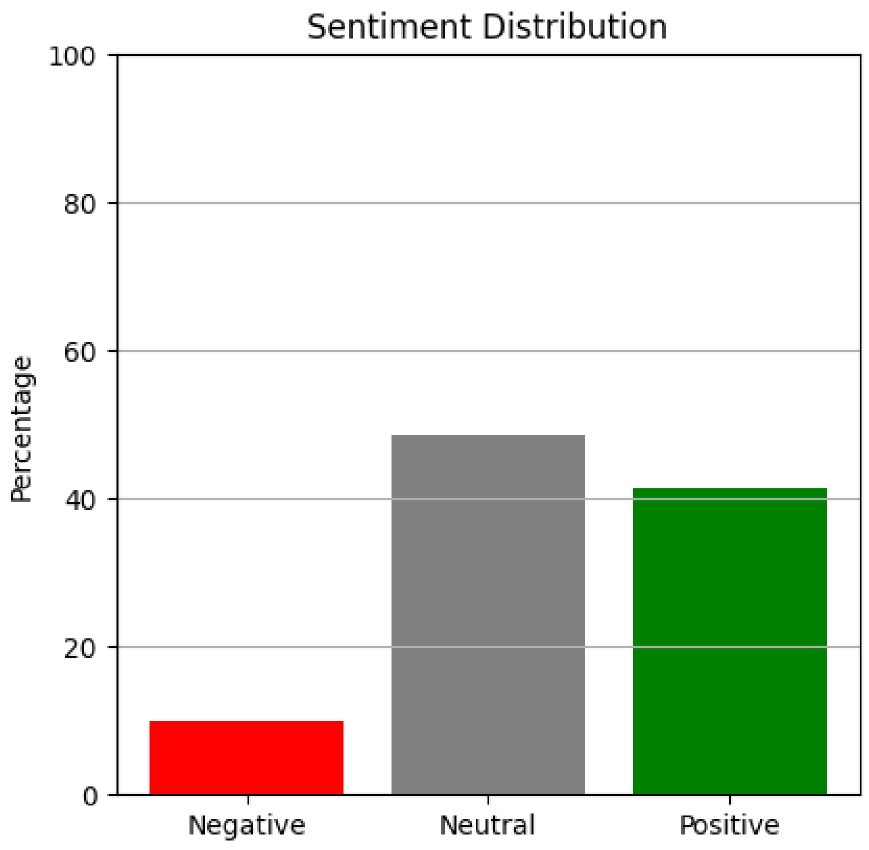
```
[36]: #calculate %
      def count_to_perc(count, total):
          return (count * 100) / total

      #score counts
      negative_count = 0
      neutral_count = 0
      positive_count = 0

      #create new view
      df = metaverse_tweets_with_sentiment.copy()
```

```
#compare scores and increment counts
for i in range(len(df)):
    if df['Negative'][i] < df['Neutral'][i] < df['Positive'][i]:
        positive_count += 1
    elif df['Negative'][i] < df['Positive'][i]:
        neutral_count += 1
    else:
        negative_count += 1

#calculate % for counts
total_tweets = len(df)
negative_perc = round(count_to_perc(negative_count, total_tweets), 2)
neutral_perc = round(count_to_perc(neutral_count, total_tweets), 2)
positive_perc = round(count_to_perc(positive_count, total_tweets), 2)
print(f'Negative: {negative_perc}%') #results
print(f'Neutral: {neutral_perc}%')
print(f'Positive: {positive_perc}%')
```

```
Negative: 9.97%
Neutral: 48.55%
Positive: 41.48%
```

We have our results, we can plot them to see them better.

[37]:
```
#plotting
labels = ['Negative', 'Neutral', 'Positive'] #x-axis labels
percentages = [negative_perc, neutral_perc, positive_perc] #y-axis labels
colors = ['red', 'gray', 'green']
plt.figure(figsize=(5, 5))
plt.bar(labels, percentages, color=colors)
plt.ylabel('Percentage')
plt.title('Sentiment Distribution')
plt.ylim(0, 100)
plt.grid(axis='y')
#plot
plt.show()
```



It is clear here that the majority of tweets are actually neutral. Then we have a large amount that is positive, and a minority of negative tweets.

4.4.3 User Location

Lastly, it's important to understand the origins of this sentiment. Each tweet in our original dataset includes the user's location, provided they have added it to their profile at the time of collection. Some users leave this field blank or enter a fictional place. Those who do specify a location might write the name of a country in various ways or mention a city without indicating the country (e.g., Los Angeles, CA).
I utilized two open-source lists: one of all countries and another of all US states. This enhances accuracy by enabling us to cross-check and identify more locations. Adding a list of major cities worldwide could further refine this process, but it would have minimal impact on our small dataset of 500.
The data from the .json files is loaded into two lists, and all `NA / NaN` values are removed. We then cross-check each user location with both lists, considering case sensitivity. If a match is found, we add the country to a dictionary and increment its count by one.

[38]:
```
#load .json file in lower case
def load_json(file_path):
    try:
        with open(file_path, 'r') as file:
            data = [element.lower() for element in json.load(file)]
            return data
    except Exception as e:
        print(f"Error: Failed to load JSON file from {file_path} - {e}")
        return []
```

[39]:
```
#load country and states lists
country_list = load_json('data/countries.json')
states_list = load_json('data/states.json')

#dict to hold country counts
country_counts = {}

user_location = metaverse_tweets['user_location'].fillna('') #remove NA / NaN␣
 ↪values

#prioritized list to handle multi-word country names
prioritized_countries = sorted(country_list, key=len, reverse=True)

#aliases mapping
aliases = {
    'uk': 'UK',
    'united kingdom': 'UK',
    'united states': 'USA',
    'united arab emirates': 'UAE',
}

#check if any word in the location matches a country or state
def find_country(location, states_list, prioritized_countries, aliases):
    location_lower = location.lower()
    #check for multi-word country names and aliases
    for country in prioritized_countries:
        if country in location_lower:
            return aliases.get(country, country.capitalize())
    #check for states
    for state in states_list:
        if state in location_lower:
            return 'USA'
    return None

#check locations
for location in user_location:
    if location == '':
        continue  #skip if empty
    country = find_country(location, states_list, prioritized_countries,␣
 ↪aliases)
```

```
    if country:
        if country in country_counts:
            country_counts[country] += 1
        else:
            country_counts[country] = 1

#origin and count
tweets_origin = pd.DataFrame(list(country_counts.items()), columns=['country',␣
 ↪'count'])
#check
tweets_origin
```

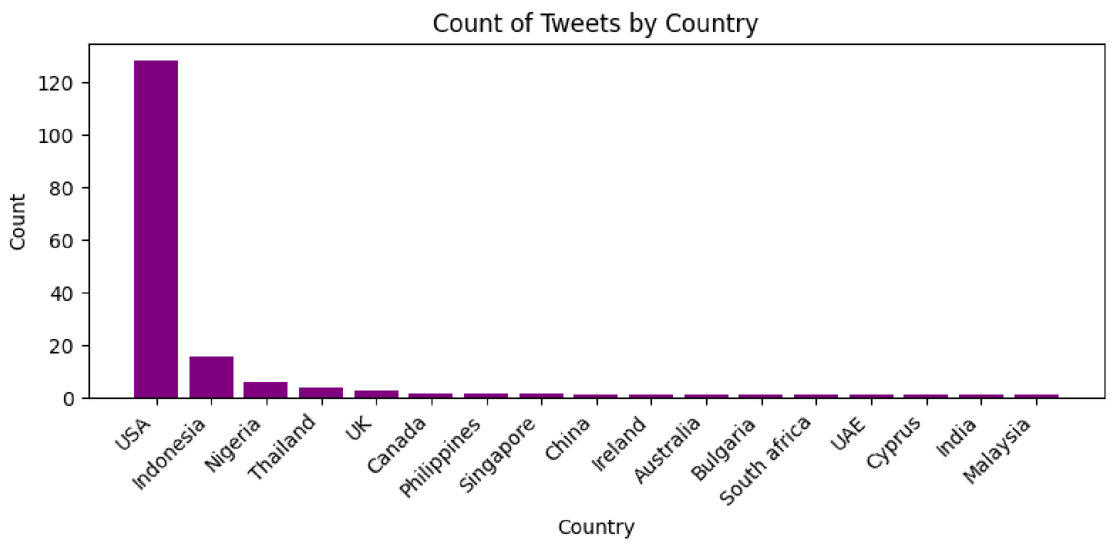
[39]:

```
         country  count
0            USA    128
1    Philippines      2
2        Ireland      1
3         Canada      2
4       Thailand      4
5          China      1
6        Nigeria      6
7      Indonesia     16
8             UK      3
9      Australia      1
10      Bulgaria      1
11     Singapore      2
12  South africa      1
13           UAE      1
14        Cyprus      1
15         India      1
16      Malaysia      1
```

Let's plot this data to get a better view of it.

[40]:

```
#sort by count in descending order
tweets_origin = tweets_origin.sort_values(by='count', ascending=False)

#plotting
plt.figure(figsize=(8, 4))
plt.bar(tweets_origin['country'], tweets_origin['count'], color='purple')
plt.xlabel('Country')
plt.ylabel('Count')
plt.title('Count of Tweets by Country')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
#plot
plt.show()
```



Here we can see that the US is talking about virtual real estate the most, with Indonesia coming in second, then Nigeria. According to Google Trends [18] though, the US, Canada and Australia are the top three googlers for the term *Virtual Real Estate*.

5 Conclusion

In this project, I set out to explore the difference between traditional and virtual real estate by addressing several questions. By analyzing data from the International House Price Database, I found that the House Price Index (HPI) grew by 1,011% from 1975 to 2023, while Personal Disposable Income (PDI) increased by 1,113%. Contrary to my initial assumption that purchasing power was lagging behind real estate prices, this suggest that people today have greater financial capacity to afford real estate (in theory), despite the substantial increase in house prices. Note that PDI does not take into account the distribution of wealth within a given country.

Moreover, after analyzing data from the Decentraland Land API, I showed that the average price of virtual land is approximately 30,000 USD (as of June, 2024). This is significantly more affordable compared to the average *real* house price of 600,000 USD, making virtual land an appealing investment, especially in light of increased purchasing power.

Additionally, sentiment analysis of 500 tweets about virtual real estate revealed predominantly neutral sentiments, with 40% positive and 10% negative. This indicates a generally balanced or slightly favorable perception of virtual real estate. Location distribution showed that the US leads in discussions about virtual real estate, followed by Indonesia and Nigeria.

In conclusion, the findings indicate that the increased purchasing power and the high cost of traditional real estate might be driving interest in more affordable virtual real estate options. This interest is further supported by positive sentiment and active discussions across various regions, suggesting that virtual real estate is becoming a notable alternative investment in today's world.

6 Links

6.1 References

[1]: Metaverse Real Estate Statistics (2024) Link
[2]: The Interplay Between Policy and Technology in Metaverses: Towards Seamless Avatar Interoperability Using Self-Sovereign Identity Link
[3]: Why do people use Metaverse? A uses and gratification theory perspective, Telematics and Informatics Link
[4]: Turning the Virtual into Reality: China's Role in the Metaverse Link
[5]: Why do people use Metaverse? A uses and gratification theory perspective, Telematics and Informatics Link
[6]: Decentraland: The Metaverse's Early Mover Link
[7]: House Price Index Link
[8]: What is Personal Disposable Income (PDI)? Link
[9]: Nominal Values Link
[10]: Decentraland Link
[11]: Parcels and Land Link
[12]: What is MANA? Link
[13]: Timeline of the 2000s United States housing bubble Link
[14]: South Carolina Revenue and Fiscal Affairs Office Link
[15]: roBERTa Pre-Trained Model Link
[16]: Sentiment Analysis by Mehran Shakarami Link
[17]: CoinGecko API Link
[18]: Google Trends Link