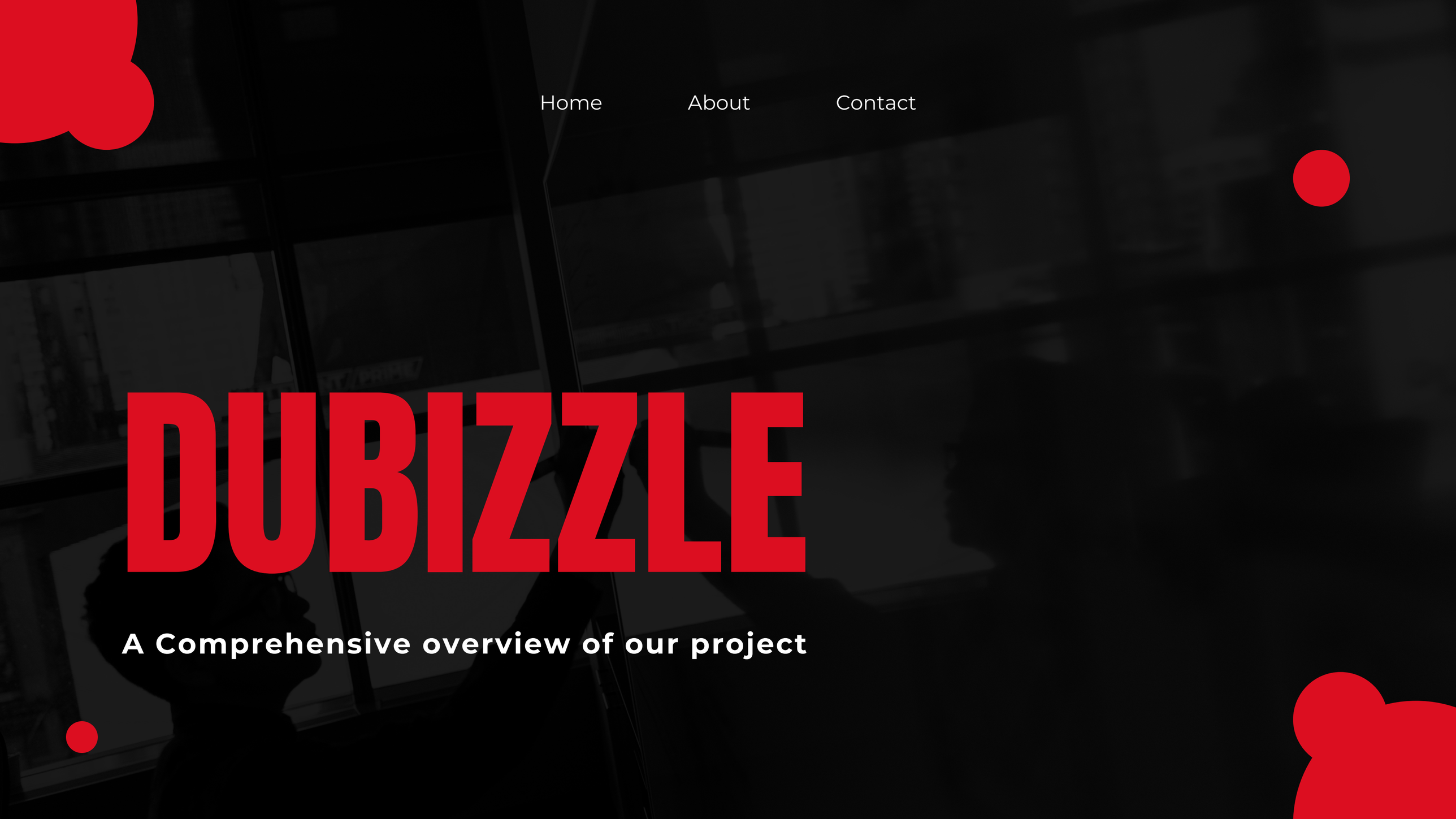


Home About Contact

DUBIZZLE

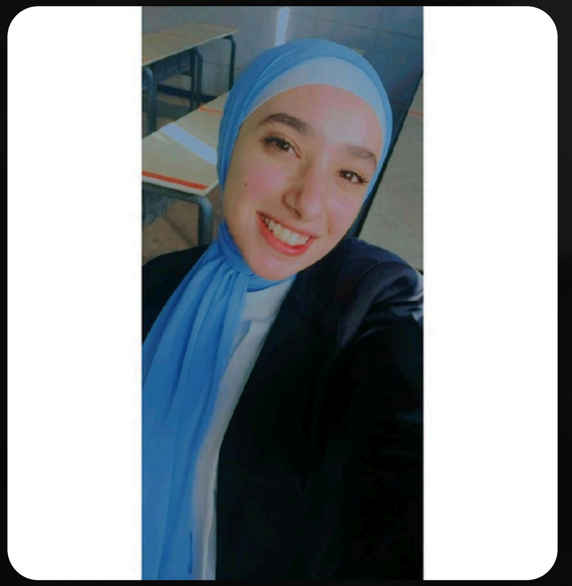
A Comprehensive overview of our project

TEAM MEMBERS

Ali Mohamed

Raghdan Ramadan

Ahmed Fayez



Maram Habashy

Supervisor: Eng. Huda Hemdan

WHAT IS ABOUT DUBIZZLE?

Dubizzle is an online marketplace in Egypt where people can buy, sell, and find a variety of items and services. It’s commonly used for listing and purchasing used items like cars, electronics, furniture, and real estate.

OVERVIEW ABOUT DATASET

captures detailed information on real estate listings

This dataset is a collection of real estate listings scraped from Dubizzle website
The data covers a variety of residential properties, with details about property type, ownership status, and features like area, number of rooms, and amenities, allowing users to analyze trends in property availability, pricing, and regional market characteristics. It also offers valuable insights into financial aspects such as pricing, down payments, and payment options, which can be used to understand affordability and market demand in different locations.

Read More

THE DATA ANALYSIS PROCESS

Data Collection

collecting data by web scrap from website “Dubizzle” by Python

data Cleaning and modeling

cleaning data by Python and divide dataset into 7 tables for data modeling

Data Analysis

Create an interactive dashboard summarizing key metrics like average price, median area, and property type distribution.

Insights and recommendation

Present the most relevant findings and visualizations, detailing key insights

TOOLS USED IN DATA ANALYSIS

Leveraging Technology for Better Insights

1. Python (Pandas, NumPy)

- Data pre-processing, cleaning, and transformation.

2.My SQL by python

- Create Queries on the data to get more info

3. Power BI

- Interactive visualization and dashboard creation.

4. Tableau

Advanced visualization, especially for geographical and trend analysis.

Read More

DATASET OVERVIEW

Dataset consists of 7 tables

- **property_facts**
  - to connect as star modeling
  - area, bedrooms, bathrooms, price, down payment, level, listing date, total Amenities, fully equipped, and all Foreign keys linking to dimensions
- **location_Dim**
  - city
- **Type_Dim**
  - type, furnished, completion status, delivery term
- **Ownership_dim**
  - ownership, ownership_id
- **Payment_Option_Dim**
  - payment option, payment option_id
- **Price_Type_Dim**
  - price type, price type_id
- **Amenities_Dim**

DATA COLLECTION - INITIAL SCRAPING SETUP

Page Request & Parsing (requests and BeautifulSouplibraries):

- Started with making HTTP requests to Dubizzle Egypt's apartment listings page. Utilized requests to fetch the HTML data and BeautifulSoup to parse and navigate the DOM structure effectively.

Pagination Handling:

- Implemented a loop to iterate through multiple pages by dynamically updating the page number in the URL. The loop stops once no more listings are found on the page.

Data Storage:

- Collected each apartment's listing URL and stored the links in a list for later use. These were then saved to a CSV file for streamlined asynchronous processing in the next phase.

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd

# Base URL
base_url = "https://www.dubizzle.com.eg/en/properties/apartments-duplex-for-sale/?page={}"

# Initialize lists to store scraped data
apartments_links = []

# Pagination loop
page = 1
while True:
    print(f"Processing page {page}...")
    # Request the page
    url = base_url.format(page)
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Find all product containers
    print("Finding apartments...")
    apartments = soup.find_all('div', class_='_637fa00f')

    if not apartments:
        print("No apartments found on this page. Stopping.")
        break  # Stop if no products found on the page

    print(f"Found {len(apartments)} apartments on this page.")
    for apartment in apartments:
        # Extract product link
        link = apartment.find('a', href=True)['href']
        full_link = f"https://www.dubizzle.com.eg{link}"  # Make sure the URL is complete
        apartments_links.append(full_link)

    # Increment page count
    page += 1

print("Scraping complete. Creating DataFrame...")
# Create DataFrame
df = pd.DataFrame({

    'Apartment Link': apartments_links
})

print("DataFrame created. Exporting to CSV...")
# Export to CSV
df.to_csv('dubizzle_apartments_links.csv', index=False)
print("CSV exported successfully!")
```

```
Streaming output truncated to the last 5000 lines.
Found 50 apartments on this page.
Processing page 3694...
Finding apartments...
Found 50 apartments on this page.
Processing page 3695...
Finding apartments...
```

DATA EXTRACTION - SEQUENTIAL DETAIL SCRAPING (47 HOURS)

the challenges faced in extracting apartment listing details with sequential scraping

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
from tqdm import tqdm

df=pd.read_csv('/content/dubizzle_apartments_links.csv')
apartments_links=df['Apartment Link'].tolist()


apartments_data = []


for link in tqdm(apartments_links, desc="Apartments Products", unit="Apartment"):
    response = requests.get(link)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extract apartment details
    Highlights = soup.find_all('div', class_='_948d9e0a e655db77 _371e9918 _95d4067f')
    details = soup.find_all('div', class_='_9a8eacd9')
    location =  soup.find('span', {'aria-label': 'Location'})
    listing_date=soup.find('span', {'aria-label': 'Creation date'})
    price=soup.find('span', {'aria-label': 'Price'})
    down_payment=soup.find('span', {'aria-label': 'Price Info'})
    amenities=soup.find_all('span', class_='_241f82d9') + soup.find_all('span', class_='c327b807')

    apartment_data = {}

    for Highlight in Highlights:
        label = Highlight.find_all('span')[0].text.strip()
        value = Highlight.find_all('span')[1].text.strip()

        if label == 'Type':
            apartment_data['Type'] = value
        elif label == 'Ownership':
            apartment_data['Ownership'] = value
        elif label == 'Area (m²)':
            apartment_data['Area (m²)'] = value
        elif label == 'Bedrooms':
            apartment_data['Bedrooms'] = value
        elif label == 'Bathrooms':
            apartment_data['Bathrooms'] = value
        elif label == 'Furnished':
            apartment_data['Furnished'] = value


    for detail in details:
        label = detail.find_all('span')[0].text.strip()
        value = detail.find_all('span')[1].text.strip()

        if label == 'Payment Option':
            apartment_data['Payment Option'] = value
        elif label == 'Delivery Date':
            apartment_data['Delivery Date'] = value
        elif label == 'Level':
            apartment_data['Level'] = value
        elif label == 'Completion status':
            apartment_data['Completion status'] = value
        elif label == 'Delivery Term':
            apartment_data['Delivery Term'] = value

        apartment_data['Location'] = location.text.strip()
        apartment_data['Listing date'] = listing_date.text.strip()
        apartment_data['Price'] = price.text.strip()
        apartment_data['Down Payment'] = down_payment.text.strip() if down_payment else None
        apartment_data['Amenities'] = [amenity.text.strip() for amenity in amenities]

    apartments_data.append(apartment_data)


# Create DataFrame
    df = pd.DataFrame(apartments_data)
```

```
Apartments Products:   0%|          | 498/267950 [05:19<47:39:30,  1.56Apartment/s]
```

DATA EXTRACTION - MULTITHREADED DETAIL SCRAPING (29 HOURS)

the use of multithreading to optimize apartment detail extraction, reducing the runtime from 47 to 29 hours

```python
import requests
from bs4 import BeautifulSoup
import pandas as pd
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor, as_completed

# Read the CSV file
df = pd.read_csv('/content/dubizzle_apartments_links.csv')
apartments_links = df['Apartment Link'].tolist()

# Function to extract data from a link
def extract_apartment_data(link):
    response = requests.get(link)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extract apartment details
    Highlights = soup.find_all('div', class_='_948d9e0a e655db77 _371e9918 _95d4067f')
    details = soup.find_all('div', class_='_3a8eacd9')
    location = soup.find('span', {'aria-label': 'Location'})
    listing_date = soup.find('span', {'aria-label': 'Creation date'})
    price = soup.find('span', {'aria-label': 'Price'})
    down_payment = soup.find('span', {'aria-label': 'Price Info'})
    amenities = soup.find_all('span', class_='_241f82d9') + soup.find_all('span', class_='c327b807')

    apartment_data = {}

    for Highlight in Highlights:
        label = Highlight.find_all('span')[0].text.strip()
        value = Highlight.find_all('span')[1].text.strip()

        if label == 'Type':
            apartment_data['Type'] = value
        elif label == 'Ownership':
            apartment_data['Ownership'] = value
        elif label == 'Area (m²)':
            apartment_data['Area (m²)'] = value
        elif label == 'Bedrooms':
            apartment_data['Bedrooms'] = value
        elif label == 'Bathrooms':
            apartment_data['Bathrooms'] = value
        elif label == 'Furnished':
            apartment_data['Furnished'] = value

    for detail in details:
        label = detail.find_all('span')[0].text.strip()
        value = detail.find_all('span')[1].text.strip()

        if label == 'Payment Option':
            apartment_data['Payment Option'] = value
        elif label == 'Delivery Date':
            apartment_data['Delivery Date'] = value
        elif label == 'Level':
            apartment_data['Level'] = value
        elif label == 'Completion status':
            apartment_data['Completion status'] = value
        elif label == 'Delivery Term':
            apartment_data['Delivery Term'] = value

    apartment_data['Location'] = location.text.strip() if location else None
    apartment_data['Listing date'] = listing_date.text.strip() if listing_date else None
    apartment_data['Price'] = price.text.strip() if price else None
    apartment_data['Down Payment'] = down_payment.text.strip() if down_payment else None
    apartment_data['Amenities'] = [amenity.text.strip() for amenity in amenities]

    return apartment_data

# Create a list to store apartment data
apartments_data = []

# Use ThreadPoolExecutor to make requests in parallel
with ThreadPoolExecutor(max_workers=10) as executor:
    future_to_link = {executor.submit(extract_apartment_data, link): link for link in apartments_links}
    for future in tqdm(as_completed(future_to_link), total=len(apartments_links), desc="Apartments Products", unit="Apartment"):
        try:
            data = future.result()
            apartments_data.append(data)
        except Exception as e:
            print(f"Error processing link {future_to_link[future]}: {e}")

# Create DataFrame
df = pd.DataFrame(apartments_data)

# Export to CSV
df.to_csv('dubizzle_apartments_data.csv', index=False)
```

```
Apartments Products:   0%|          | 921/267950 [06:06<29:29:16,  2.52Apartment/s]
```

DATA COLLECTION - ASYNCHRONOUS DATA SCRAPING

Switch to Asynchronous Scraping (aiohttp and asyncio libraries):

- Used asynchronous requests with aiohttp to efficiently manage multiple requests to Dubizzle, ensuring minimal server load and faster data collection.

Data Parsing (BeautifulSoup):

- Parsed individual apartment pages for key details like location, price, listing date, amenities, and highlights. Extracted each element's text and stored them in a dictionary format.

```python
import aiohttp
import asyncio
from bs4 import BeautifulSoup
import pandas as pd
from tqdm.asyncio import tqdm as tqdm_asyncio
import time

# Read the CSV file
df = pd.read_csv('/content/dubizzle_apartments_links.csv')
apartments_links = df['Apartment Link'].tolist()

# Asynchronous function to extract data from a link
async def fetch(session, link):
    try:
        async with session.get(link) as response:
            if response.status == 200:
                response_text = await response.text()
                return response_text
            else:
                print(f"Failed to fetch {link}, Status Code: {response.status}")
                return None
    except Exception as e:
        print(f"Error fetching {link}: {e}")
        return None

# Function to parse apartment data from HTML content
def parse_apartment_data(html_content):
    soup = BeautifulSoup(html_content, 'html.parser')

    # Extract apartment details
    Highlights = soup.find_all('div', class_='_948d9e0a e655db77 _371e9918 _95d4067f')
    details = soup.find_all('div', class_='_9a8eacd9')
    location = soup.find('span', {'aria-label': 'Location'})
    listing_date = soup.find('span', {'aria-label': 'Creation date'})
    price = soup.find('span', {'aria-label': 'Price'})
    down_payment = soup.find('span', {'aria-label': 'Price Info'})
    amenities = soup.find_all('span', class_='_241f82d9') + soup.find_all('span', class_='c327b807')

    apartment_data = {}

    for Highlight in Highlights:
        spans = Highlight.find_all('span')
        if len(spans) >= 2:
            label = spans[0].text.strip()
            value = spans[1].text.strip()
            apartment_data[label] = value

    for detail in details:
        spans = detail.find_all('span')
        if len(spans) >= 2:
            label = spans[0].text.strip()
            value = spans[1].text.strip()
            apartment_data[label] = value

    apartment_data['Location'] = location.text.strip() if location else None
    apartment_data['Listing date'] = listing_date.text.strip() if listing_date else None
    apartment_data['Price'] = price.text.strip() if price else None
    apartment_data['Down Payment'] = down_payment.text.strip() if down_payment else None
    apartment_data['Amenities'] = [amenity.text.strip() for amenity in amenities]
```

DATA COLLECTION - ASYNCHRONOUS DATA SCRAPING

Batch Processing and Rate Limiting:

- Implemented batch processing and a rate limit to avoid server overload. Processed data in smaller groups, saving each batch to prevent data loss and allowing pauses between requests.

Final Data Export:

- After collecting the complete data set, exported all apartment data into a CSV file, resulting in a consolidated data file ready for analysis.

```python
# Function to handle scraping with rate limiting and batch processing
async def scrape_apartment_data(links, batch_size=1000, max_workers=50):
    apartments_data = []

    # Process links in batches
    for i in range(0, len(links), batch_size):
        batch_links = links[i:i + batch_size]
        print(f"Processing batch {i // batch_size + 1} / {len(links) // batch_size + 1}")

        async with aiohttp.ClientSession() as session:
            tasks = []
            for link in batch_links:
                tasks.append(fetch(session, link))

            # Gather the responses asynchronously with a limit on concurrency
            for task in tqdm_asyncio.as_completed(tasks, total=len(tasks), desc="Apartments Products", unit="Apartment"):
                try:
                    html_content = await task
                    if html_content:
                        data = parse_apartment_data(html_content)
                        apartments_data.append(data)
                except Exception as e:
                    print(f"Error processing task: {e}")

        # Save progress after each batch to prevent data loss
        df = pd.DataFrame(apartments_data)
        df.to_csv(f'dubizzle_apartments_data_batch_{i // batch_size + 1}.csv', index=False)
        print(f"Batch {i // batch_size + 1} saved!")

        # Sleep for a few seconds to prevent overwhelming the server
        await asyncio.sleep(5)

    return apartments_data

# Run the scraping and save the data to a CSV file
async def main():
    # Set batch size and max concurrent workers (tune these values based on your system and server load)
    batch_size = 1000  # Number of links processed in each batch
    max_workers = 50   # Max concurrent requests at a time

    apartments_data = await scrape_apartment_data(apartments_links, batch_size=batch_size, max_workers=max_workers)

    # Create final DataFrame and export to CSV
    df = pd.DataFrame(apartments_data)
    df.to_csv('dubizzle_apartments_data_full.csv', index=False)

# In Google Colab, use await directly to execute the main function
await main()
```

```
Processing batch 1 / 268
Apartments Products:   0%|          | 1/1000 [00:01<17:02,  1.02s/Apartment]Failed to fetch https://www.dubizzle.com.eg/en
Apartments Products:  14%|▏         | 141/1000 [00:20<01:32,  9.29Apartment/s]Failed to fetch https://www.dubizzle.com.eg
Apartments Products:  35%|▎         | 347/1000 [00:45<01:31,  7.13Apartment/s]Failed to fetch https://www.dubizzle.com.eg
Apartments Products:  45%|▍         | 453/1000 [00:58<01:55,  4.73Apartment/s]Failed to fetch https://www.dubizzle.com.eg/
Apartments Products:  72%|▋         | 716/1000 [01:30<00:34,  8.15Apartment/s]Failed to fetch https://www.dubizzle.com.eg
Apartments Products:  81%|▊         | 808/1000 [01:41<00:22,  8.49Apartment/s]Failed to fetch https://www.dubizzle.com.eg/
Failed to fetch https://www.dubizzle.com.eg/en/ad/%D8%B4%D9%82%D9%87-%D8%AA%D9%85%D9%84%D9%8A%D9%83-%D9%84%D9%84%D8%A8%D9
Apartments Products:  90%|▉         | 896/1000 [01:52<00:15,  6.59Apartment/s]Failed to fetch https://www.dubizzle.com.eg
Failed to fetch https://www.dubizzle.com.eg/en/ad/resale-ready-to-move-prime-location-fully-finished-ID50091312 8.html, Sta
Apartments Products: 100%|█         | 1000/1000 [05:00<00:00,  3.33Apartment/s]Error fetching https://www.dubizzle.com.eg/
Error fetching https://www.dubizzle.com.eg/en/ad/%D8%B4%D9%82%D8%A9-%D9%84%D9%84%D8%A8%D9%8A%D8%B9-%D9%85%D8%B7%D9%84-%D8
Error fetching https://www.dubizzle.com.eg/en/ad/apartment-for-sale-on-suez-road-in-front-of-madinaty-in-palm-island-ID207
Error fetching https://www.dubizzle.com.eg/en/ad/apartment-134m-veiw-lagoon-at-lumia-lagoon-new-capital-ID50096202 8.html:
Error fetching https://www.dubizzle.com.eg/en/ad/%D8%B4%D9%82%D9%87-%D9%84%D9%84%D8%A8%D9%8A%D8%B9-%D9%81%D9%8A-%D9%85%D8%B
Error fetching https://www.dubizzle.com.eg/en/ad/%D8%B4%D9%82%D9%87-%D9%84%D9%84%D8%A8%D9%8A%D8%B9-%D8%AA%D8%B4%D8%B7%D9%8
```

CLEANING DATA•

1. First, import the libraries required for data manipulation and visualization:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

2. Then Use the wrangle Function to read data and to know if there are duplicates or any null value and know the info of data, describe of this data set

we find the:

- The total rows is 982623
- total columns is 21
- Null values are present in the following columns: ['ownership', 'furnished', 'Payment option', 'completion status', 'delivery term', 'down payment', 'level', 'price type', 'delivery date', 'compound']
    - Drop columns with excessive nulls and those deemed unnecessary: ['Source.Name', 'Unnamed: 19', 'Unnamed: 20', 'Compound', 'Delivery Date']
- Drop 617,956 duplicate rows, reducing the total row count to 361,205.

```
ass 'pandas.core.frame.DataFrame'>
ex: 361205 entries, 0 to 982621
a columns (total 22 columns):
 Column              Non-Null Count   Dtype
 ------              --------------   -----
 Source.Name         361205 non-null  object
 Type                361205 non-null  object
 Ownership           350164 non-null  object
 Area (m²)           361205 non-null  float64
 Bedrooms            361205 non-null  float64
 Bathrooms           361205 non-null  float64
 Furnished           272233 non-null  object
 Payment Option      355768 non-null  object
 Completion status   348235 non-null  object
 Delivery Term       163912 non-null  object
 Listing date        361205 non-null  object
 Price               361205 non-null  object
 Down Payment        187980 non-null  object
 Amenities           361205 non-null  object
 Level               161874 non-null  object
 Price Type          60131 non-null   object
 Delivery Date       43826 non-null   object
 Compound            528 non-null     object
 Unnamed: 19         0 non-null       float64
 Unnamed: 20         1 non-null       float64
 Location.1          361205 non-null  object
 Location.2          361205 non-null  object
pes: float64(5), object(17)
ory usage: 63.4+ MB
```

```
f wrangle(dubbi):
 # Displaying the initial observations
 print("Initial Observations:")
 print(dubbi.head(5))
 print()

 # Conducting an examination of data types and missing values
 print("Info about data:")
 print(dubbi.info(show_counts=True))
 print()
 #drop na for location and price because no data without essential
 dubbi.dropna(subset=['Location','Price'],inplace=True)

 # Check for duplicated rows
 print("Number of duplicated rows:", dubbi.duplicated().sum())
 print()

 dubbi.drop_duplicates(inplace=True)
 print("Number of duplicated rows after dropping:", dubbi.duplicate
 print()

 # Conducting an examination of data types and missing values
 print("Info about data:")
 print(dubbi.info(show_counts=True))
 print()

 # Descriptive statistics
 print("Descriptive Statistics:")
 print(dubbi.describe())
 print()
```

CLEANING DATA CONT..•

3. Divide the location column into two new columns:
   - districts: The first part of the location data.
   - City: The second part of the location data.

In the price and Down Payment columns:

- Remove non-numeric characters (e.g., EGP A, AAA, AAA) using str.replace(). then Convert cleaned data to float type with .astype(float) for easier numerical operations.

For missing values in the Payment option column:

- If there is a value in the Down Payment column, fill Payment option with "Installment". Otherwise, fill with "Cash".

Convert column of bathrooms and bedrooms to int type

The listing date column contains month and year names but is not in date format. Use a custom function, relative_date, to parse and standardize dates in YYYY-MM-DD format.

```
[16] dt['Price'] = dt['Price'].str.replace('EGP ', '').str.replace(',', '').astype(float)
     dt['Down Payment'] = dt['Down Payment'].str.replace('EGP ', '').str.replace(',', '').str.replace(' Down Payment','').str.replace('%','').astype(float)
```

```
for index in dt.index:
    if pd.isnull(dt.loc[index, 'Payment Option']):
        if pd.notna(dt.loc[index, 'Down Payment']):
            dt.loc[index, 'Payment Option'] = 'Installment'
        else:
            dt.loc[index, 'Payment Option'] = 'Cash'
```

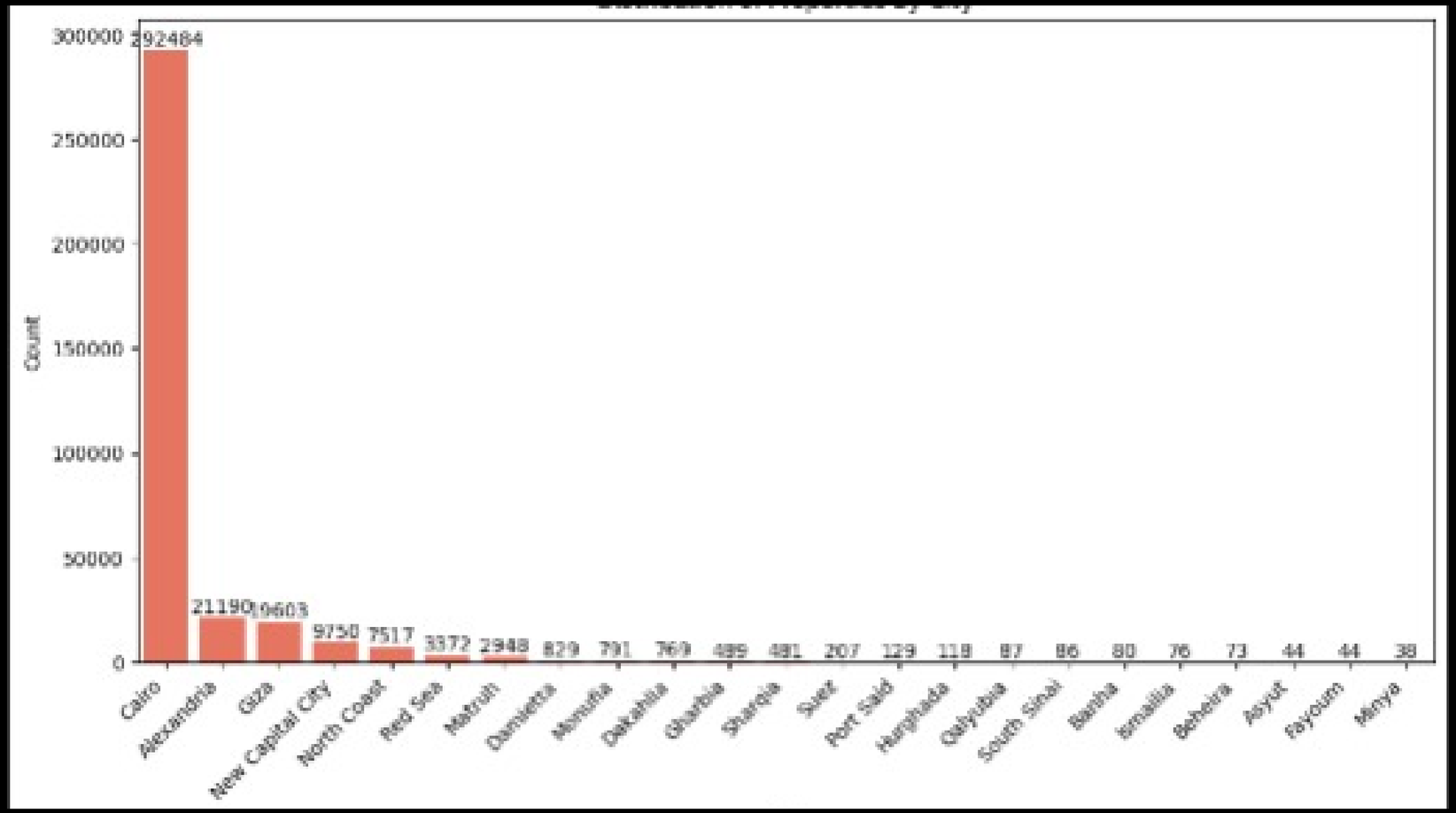
```
from datetime import datetime, timedelta
import re

# Function to convert relative dates to absolute dates
def convert_relative_date(relative_date):
    # Current date as reference
    today = datetime.today()

    # Check if the date mentions weeks, days, or months, and extract the number
    if 'week' in relative_date:
        num_weeks = int(re.search(r'(\d+)', relative_date).group(0))
        return today - timedelta(weeks=num_weeks)

    elif 'day' in relative_date:
        num_days = int(re.search(r'(\d+)', relative_date).group(0))
        return today - timedelta(days=num_days)

    elif 'month' in relative_date:
        num_months = int(re.search(r'(\d+)', relative_date).group(0))
        return today - timedelta(days=num_months * 30)  # Approximate months as 30 days

    elif 'year' in relative_date:
        num_years = int(re.search(r'(\d+)', relative_date).group(0))
        return today - timedelta(days=num_years * 365)  # Approximate year as 365 days

    elif 'yesterday' in relative_date:
        return today - timedelta(days=1)

    # Default if there's no match or unknown format
    return None

# Apply the function to the 'Listing date' column
dt['Listing date'] = dt['Listing date'].apply(lambda x: convert_relative_date(x) if pd.notnull(x

# Display the updated 'Listing date' column
dt[['Listing date']].head()
```

```
import numpy as np
# Handle missing values in 'Ownership' column by replacing NaN with 'Unknown'
dt['Ownership'].fillna('Unknown', inplace=True)

# Handle missing values in 'Completion status' column with 'Unknown'
dt['Completion status'].fillna(dt['Completion status'].mode()[0], inplace=True)

dt['Price Type'].fillna('Non Negotiable', inplace=True)

dt["Down Payment"] = np.where(dt["Payment Option"]=="Cash", 0, dt["Down Payment"])
dt["Down Payment"].isnull().sum()


dt["Furnished"].fillna(dt["Furnished"].mode()[0],inplace=True)
dt["Furnished"].isnull().sum()

dt['Level'] = dt['Level'].replace({'Ground': 0, '10+': 12,"Highest":15})
dt['Level'].isnull().sum()
dt['Level'] = dt['Level'].fillna(method='bfill')
dt['Level'].isnull().sum()
dt['Level'] = dt['Level'].astype(int)

dt["Delivery Term"].fillna(dt["Delivery Term"].mode()[0],inplace=True)
dt["Delivery Term"].isnull().sum()

dt['Listing date'].fillna(dt['Listing date'].mode()[0],inplace=True)
```

CLEANING DATA CONT...

level: Use backfill (bfill) for nulls to fill from the previous row.
Down Payment: Fill with 0 if Payment option is "Cash"; otherwise, keep the original value.
price type: Fill with "Non Negotiable" if null.

9. then we find the location 2 (it become city) contain the districts we need to handle that by Mapping Cities




```
[45] # Step 1: Clean the data in Location.2 by stripping whitespace
     dt['Location.2'] = dt['Location.2'].str.strip()

     # Step 2: Ensure that the mapping works regardless of case
     # Optionally, convert the Location.2 values to lower case
     dt['Location.2'] = dt['Location.2'].str.lower()

     # Update the city_mapping keys to match lower case
     city_mapping = {
         'new cairo': 'cairo',
         'mostakbal city': 'cairo',
         '6th of october': 'cairo',
         'new heliopolis': 'cairo',
         'shorouk city': 'cairo',
         'madinaty': 'cairo',
         'hadayek october': 'cairo',
         'sheraton': 'cairo',
         'badr city': 'cairo',
         'nasr city': 'cairo',
         'heliopolis city': 'cairo',
         'heliopolis': 'cairo',
         'mokattam': 'cairo',
         'zamalek': 'cairo',
         'dokki': 'cairo',
         'obour city': 'cairo',
         'maadi': 'cairo',
         'katameya': 'cairo',
         'zahraa al maadi': 'cairo',
         'sheikh zayed': 'cairo',
         'smoha': 'alexandria',
         'moharam bik': 'alexandria',
         'amreya': 'alexandria',
         'agami': 'alexandria',
         'borg al-arab': 'alexandria',
         'alamein': 'north coast',
         'ain sukhna': 'red sea',
         'makadi bay': 'red sea',
         'new damietta': 'damietta',
         'new mansoura': 'dakahlia'}
     # Step 3: Replace cities with their mapped values
     dt['Location.2'] = dt['Location.2'].replace(city_mapping)
     # Step 4: Convert the Location.2 values to title case for proper capitalization
     dt['Location.2'] = dt['Location.2'].str.title()
     # Rename the column Location.2 to City
     dt.rename(columns={'Location.2': 'City'}, inplace=True)
```

CLEANING DATA CONT..•

10. last, The Amenities column is a list contain all Amenities for each Appartment and need to split it into columns and get new column as total Amenities for each row and fully equipped that contain [electricity meter, water mater, security]

```
# Step 1: Add a new column 'Total_Amenities' with the count of amenities for each lis
dt['Total_Amenities'] = dt[list(all_amenities)].sum(axis=1)

# Step 2: Add a new column 'Fully_Equipped' to indicate if essential amenities are pr
# Define essential amenities (you can customize this list)
essential_amenities = ['Electricity Meter', 'Water Meter', 'Security']

# Check if each listing has all essential amenities
dt['Fully_Equipped'] = dt[essential_amenities].all(axis=1).astype(int)

# Display the updated dataset with new summary columns
dt[['Amenities', 'Total_Amenities', 'Fully_Equipped']].head()
```

	Amenities	Total_Amenities	Fully_Equipped
0	[Pool, Electricity Meter, Water Meter, Natural...	11	1
1	[Maids Room, Pets Allowed, Pool, Electricity M...	13	1
2	[Pool, Electricity Meter, Water Meter, Natural...	12	1
3	[Balcony]	1	0
4	[]	0	0

```
[33] # Step 1: Clean and split the 'Amenities' column into lists of amenities
dt['Amenities'] = (
    dt['Amenities']
    .str.strip("[]")              # Remove brackets
    .str.replace("'", "")         # Remove single quotes
    .str.split(", ")              # Split amenities into lists
)

# Step 2: Extract all unique amenities across all listings
all_amenities = set(amenity for amenities in dt['Amenities'] for amenity in amenities if amenity)

# Step 3: Create a binary indicator column for each amenity
for amenity in all_amenities:
    dt[amenity] = dt['Amenities'].apply(lambda x: 1 if amenity in x else 0)

# Step 4: Summarize the total number of listings that have each amenity
amenities_summary = dt[list(all_amenities)].sum().sort_values(ascending=False)

# Display the top amenities
print("Top Amenities in the Dataset:")
print(amenities_summary.head(10))
```

```
Top Amenities in the Dataset:
Electricity Meter    210746
Security             198515
Water Meter          196269
Balcony              194674
Natural Gas          178331
Covered Parking      169217
Pets Allowed         161209
Landline             139695
Private Garden       126979
Pool                 113219
dtype: int64
```

CLEANING DATA CONT..•

Finally, need to get more than table for data modeling from once dataset sheet by divide it into 7 sheets lets go to know how this ?

after that save 7 sheet and once dataset after clean into csv

```
# Merging dimension IDs with the main dataframe to create the f
dt = dt.merge(location_dim, on=['City', 'Location.1'], how='le
      .merge(type_dim, on=['Type', 'Furnished', 'Completion st
      .merge(ownership_dim, on='Ownership', how='left')\
      .merge(payment_option_dim, on='Payment Option', how='le
      .merge(price_type_dim, on='Price Type', how='left')\
      .merge(amenities_dim, on=amenities_columns, how='left')

# Selecting relevant columns for the fact table
fact_table = dt[['Location_ID', 'Type_ID', 'Ownership_ID', 'Pay
                'Price_Type_ID', 'Amenities_ID', 'Area (m²)',
                'Bathrooms', 'Price', 'Down Payment', 'Level',
                'Listing date', 'Total_Amenities', 'Fully_Equi

# Adding a unique ID for the fact table
fact_table['Property_ID'] = fact_table.index + 1
```

```
# Creating unique dimensions from the original DataFrame

# Dimension Table: Location_Dim
location_dim = dt[['City', 'Location.1']].drop_duplicates().reset_index(drop=True)
location_dim['Location_ID'] = location_dim.index + 1

# Dimension Table: Type_Dim
type_dim = dt[['Type', 'Furnished', 'Completion status', 'Delivery Term']].drop_duplicates().reset_index(drop=True)
type_dim['Type_ID'] = type_dim.index + 1

# Dimension Table: Ownership_Dim
ownership_dim = dt[['Ownership']].drop_duplicates().reset_index(drop=True)
ownership_dim['Ownership_ID'] = ownership_dim.index + 1

# Dimension Table: Payment_Option_Dim
payment_option_dim = dt[['Payment Option']].drop_duplicates().reset_index(drop=True)
payment_option_dim['Payment_Option_ID'] = payment_option_dim.index + 1

# Dimension Table: Price_Type_Dim
price_type_dim = dt[['Price Type']].drop_duplicates().reset_index(drop=True)
price_type_dim['Price_Type_ID'] = price_type_dim.index + 1

# Dimension Table: Amenities_Dim
amenities_columns = [
    'Elevator', 'Maids Room', 'Built in Kitchen Appliances', 'Pets Allowed',
    'Natural Gas', 'Pool', 'Private Garden', 'Covered Parking',
    'Electricity Meter', 'Balcony', 'Central A/C & heating',
    'Water Meter', 'Landline', 'Security'
]
amenities_dim = dt[amenities_columns].drop_duplicates().reset_index(drop=True)
amenities_dim['Amenities_ID'] = amenities_dim.index + 1
```

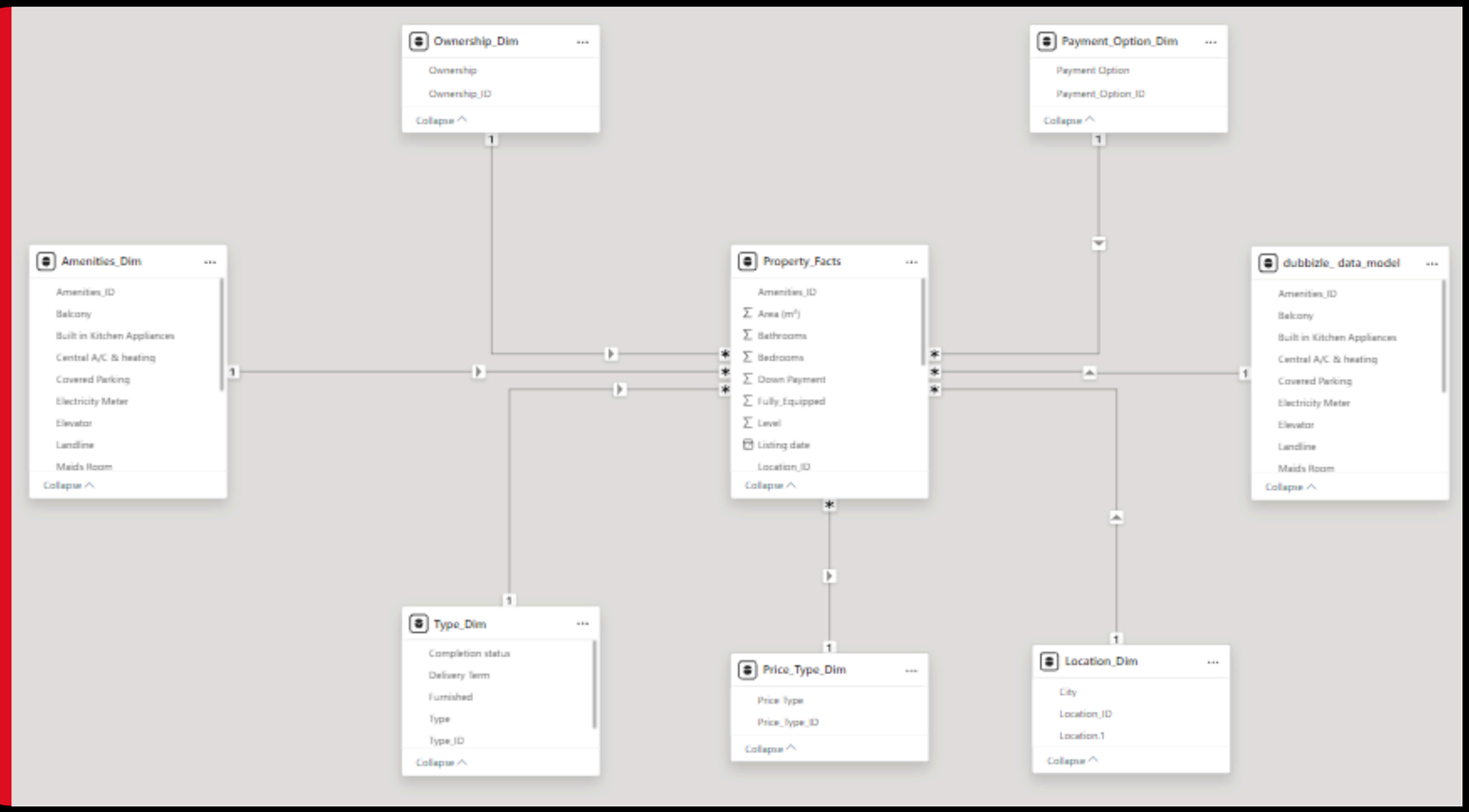
DATA MODELING

using (power bi) and Star schema

A star schema is used to denormalize business data into dimensions and facts. A star schema has a single fact table in the center, containing business "facts"

- Better performance
- Usability
- Faster Refresh
- fact table support Summarization

Read More



SQL USING PYTHON

Tables creation

create the tables by python using mysql for the 7 tables

Data insertion

insert the data ino dataset into 7 tables for data modeling

discovering and insights

discover the data using some overall questions about the data to get deep into the data

TABLE CREATION AND REALTIONS

- insert the connection between the python and sql insert the database
- Property_Facts:
- Type (foreign key) → Type_Dim (primary key)
- Ownership (foreign key) → Ownership_Dim (primary key)
- Payment_Option (foreign key) → Payment_Option_Dim (primary key)
- City (foreign key) → Location_Dim (primary key)
- Amenities (foreign key) → Amenities_Dim (primary key)
- Type_Dim:
- Primary key referenced by Property_Facts.Type.

In [7]:
```
# Connect to MySQL server
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password=""
)
cursor = conn.cursor()
```

In [8]:
```
# Create a new database
cursor.execute("CREATE DATABASE IF NOT EXISTS dubizzle;")
cursor.execute("USE dubizzle;")
```

In [9]:
```
# Step 2: Load CSV files into DataFrames
fact_table = pd.read_csv('Property_Facts.csv')
location_dim = pd.read_csv('Location_Dim.csv')
type_dim = pd.read_csv('Type_Dim.csv')
ownership_dim = pd.read_csv('Ownership_Dim.csv')
payment_option_dim = pd.read_csv('Payment_Option_Dim.csv')
price_type_dim = pd.read_csv('Price_Type_Dim.csv')
amenities_dim = pd.read_csv('Amenities_Dim.csv')
```

In [57]:
```
#SQL queries to create the tables
create_amenities_dim = """
CREATE TABLE IF NOT EXISTS Amenities_Dim (
    Amenities_ID INT PRIMARY KEY,
    Elevator TINYINT,
    Maids_Room TINYINT,
    Built_in_Kitchen_Appliances TINYINT,
    Pets_Allowed TINYINT,
    Natural_Gas TINYINT,
    Pool TINYINT,
    Private_Garden TINYINT,
    Covered_Parking TINYINT,
    Electricity_Meter TINYINT,
    Balcony TINYINT,
    Central_AC_heating TINYINT,
    Water_Meter TINYINT,
    Landline TINYINT,
    Security TINYINT
);
"""

create_location_dim = """
CREATE TABLE IF NOT EXISTS Location_Dim (
    Location_ID INT PRIMARY KEY,
    City VARCHAR(255),
    Location_1 VARCHAR(255)
);
"""
```

TABLE CREATION AND REALTIONS

- Ownership_Dim:
- Primary key referenced by Property_Facts.Ownership.
- Payment_Option_Dim:
- Primary key referenced by Property_Facts.Payment_Option.
- Location_Dim:
- Primary key referenced by Property_Facts.City.
- Amenities_Dim:
- Primary key referenced by Property_Facts.Amenities.

```
create_property_facts = """
CREATE TABLE IF NOT EXISTS Property_Facts (
    Property_ID INT PRIMARY KEY,
    Location_ID INT,
    Type_ID INT,
    Ownership_ID INT,
    Payment_Option_ID INT,
    Price_Type_ID INT,
    Amenities_ID INT,
    Area_M2 DECIMAL(10,2),
    Bedrooms INT,
    Bathrooms INT,
    Price DECIMAL(15,2),
    Down_Payment DECIMAL(15,2),
    Level INT,
    Listing_Date DATETIME,
    Total_Amenities INT,
    Fully_Equipped TINYINT,
    FOREIGN KEY (Location_ID) REFERENCES Location_Dim(Location_ID),
    FOREIGN KEY (Type_ID) REFERENCES Type_Dim(Type_ID),
    FOREIGN KEY (Ownership_ID) REFERENCES Ownership_Dim(Ownership_ID),
    FOREIGN KEY (Payment_Option_ID) REFERENCES Payment_Option_Dim(Payment
    FOREIGN KEY (Amenities_ID) REFERENCES Amenities_Dim(Amenities_ID)
);
"""

# Step 4: Execute the create table queries
cursor.execute(create_amenities_dim)
cursor.execute(create_location_dim)
cursor.execute(create_ownership_dim)
cursor.execute(create_payment_option_dim)
cursor.execute(create_type_dim)
cursor.execute(create_price_type_dim)
cursor.execute(create_property_facts)


print("Tables created successfully in the 'dubblze.")
```

```
Tables created successfully in the 'dubblze.
```

In [57]:

```
#SQL queries to create the tables
create_amenities_dim = """
CREATE TABLE IF NOT EXISTS Amenities_Dim (
    Amenities_ID INT PRIMARY KEY,
    Elevator TINYINT,
    Maids_Room TINYINT,
    Built_in_Kitchen_Appliances TINYINT,
    Pets_Allowed TINYINT,
    Natural_Gas TINYINT,
    Pool TINYINT,
    Private_Garden TINYINT,
    Covered_Parking TINYINT,
    Electricity_Meter TINYINT,
    Balcony TINYINT,
    Central_AC_heating TINYINT,
    Water_Meter TINYINT,
    Landline TINYINT,
    Security TINYINT
);
"""

create_location_dim = """
CREATE TABLE IF NOT EXISTS Location_Dim (
    Location_ID INT PRIMARY KEY,
    City VARCHAR(255),
    Location_1 VARCHAR(255)
);
"""
```

```
create_ownership_dim = """
CREATE TABLE IF NOT EXISTS Ownership_Dim (
    Ownership_ID INT PRIMARY KEY,
    Ownership VARCHAR(255)
);
"""

create_payment_option_dim = """
CREATE TABLE IF NOT EXISTS Payment_Option_Dim (
    Payment_Option_ID INT PRIMARY KEY,
    Payment_Option VARCHAR(255)
);
"""

create_type_dim = """
CREATE TABLE IF NOT EXISTS Type_Dim (
    Type_ID INT PRIMARY KEY,
    Type VARCHAR(255),
    Furnished TINYINT,
    Completion_Status VARCHAR(255),
    Delivery_Term VARCHAR(255)
);
"""

create_price_type_dim = """
CREATE TABLE IF NOT EXISTS Price_Type_Dim (
    Price_Type_ID INT PRIMARY KEY,
    Price_Type VARCHAR(255)
);
"""
```

DATA INSERTION •

- Alter Tables: The code begins by setting their ID columns to auto-increment and establishing them as primary keys.
- Print Column Names: The code attempts to print the names of the columns in each dimension table, ensuring that they are correctly defined and populated.

In [24]:

```
# Step 4: Alter tables to make ID columns auto-increment
alter_location_dim = """
ALTER TABLE Location_Dim
MODIFY Location_ID INT AUTO_INCREMENT;
"""

alter_ownership_dim = """
ALTER TABLE Ownership_Dim
MODIFY Ownership_ID INT AUTO_INCREMENT;
"""

alter_payment_option_dim = """
ALTER TABLE Payment_Option_Dim
MODIFY Payment_Option_ID INT AUTO_INCREMENT;
"""

alter_type_dim = """
ALTER TABLE Type_Dim
MODIFY Type_ID INT AUTO_INCREMENT;
"""

alter_price_type_dim = """
ALTER TABLE Price_Type_Dim
MODIFY Price_Type_ID INT AUTO_INCREMENT;
"""

alter_amenities_dim = """
ALTER TABLE Amenities_Dim
MODIFY Amenities_ID INT AUTO_INCREMENT;
"""


# Execute the ALTER TABLE queries
cursor.execute(alter_location_dim)
cursor.execute(alter_ownership_dim)
cursor.execute(alter_payment_option_dim)
cursor.execute(alter_type_dim)
cursor.execute(alter_price_type_dim)


# Commit the changes
conn.commit()
```

DATA INSERTION

‘Define Insert Queries: It defines SQL insert queries for each table, specifying which columns to populate and using placeholders for the actual data values.

Insert Data Function: A function is created to facilitate the insertion of data into the database. This function iterates over rows in the provided DataFrame and executes the corresponding insert query for each row, committing the changes after all inserts are complete to optimize performance.

Data Insertion: Finally, the function is called for each table, passing in the relevant DataFrames and their respective insert queries. A confirmation message is printed once the data insertion is completed successfully.

```
# Updated Step 4: Define insert queries for each table (without IDs)
insert_amenities_query = """
INSERT INTO Amenities_Dim (Elevator, Maids_Room, Built_in_Kitchen_Appliances, Pets_Allowed,
                           Natural_Gas, Pool, Private_Garden, Covered_Parking, Electricity_Meter,
                           Balcony, Central_AC_heating, Water_Meter, Landline, Security)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
"""

insert_location_query = """
INSERT INTO Location_Dim (City, Location_1)
VALUES (%s, %s)
"""

insert_ownership_query = """
INSERT INTO Ownership_Dim (Ownership)
VALUES (%s)
"""

insert_payment_option_query = """
INSERT INTO Payment_Option_Dim (Payment_Option)
VALUES (%s)
"""

insert_type_query = """
INSERT INTO Type_Dim (Type, Furnished, Completion_Status, Delivery_Term)
VALUES (%s, %s, %s, %s)
"""

insert_price_type_query = """
INSERT INTO Price_Type_Dim (Price_Type)
VALUES (%s)
"""
```

```
# Function to insert data into the database
def insert_data(data, query):
    for _, row in data.iterrows():
        cursor.execute(query, tuple(row))
        conn.commit()
```

```
insert_data(amenities_dim, insert_amenities_query)
```

```
cursor.execute("SET FOREIGN_KEY_CHECKS = 0;")


# Step 3: Define your insert query
insert_property_facts_query = """
INSERT INTO Property_Facts (Property_ID,Location_ID, Type_ID, Ownership_ID, Payment_Option_ID,
                            Price_Type_ID, Amenities_ID, Area_M2, Bedrooms, Bathrooms, Price,
                            Down_Payment, Level, Listing_Date, Total_Amenities, Fully_Equipped)
VALUES (%s,%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
"""

# Step 4: Function to insert data into the database
def insert_data(data, query):
    for index, row in data.iterrows():
        cursor.execute(query, tuple(row))

# Step 5: Insert data into Property_Facts
insert_data(fact_table, insert_property_facts_query)

# Step 6: Re-enable foreign key checks
cursor.execute("SET FOREIGN_KEY_CHECKS = 1;")
```

DISCOVERING AND INSIGHTS

- What is the average price of properties in each city?
- How many properties have a pool and are furnished in each city?
- How many properties are available for each payment option?
- What is the highest-priced property in each city?
- Which type of property has the most listings?
- What is the total number of amenities for properties listed in each location?
- How many properties have a garden and a balcony?

1. What is the average price of properties in each city?

```
query = """
SELECT L.City, AVG(P.Price) AS Average_Price
FROM Property_Facts P
JOIN Location_Dim L ON P.Location_ID = L.Location_ID
GROUP BY L.City;
"""
df = pd.read_sql(query, conn)
print(df)
```

2. How many properties have a pool and are furnished in each city?

```
query = """
SELECT L.City, COUNT(*) AS Total_Properties
FROM Property_Facts P
JOIN Amenities_Dim A ON P.Amenities_ID = A.Amenities_ID
JOIN Location_Dim L ON P.Location_ID = L.Location_ID
WHERE A.Pool = 'Yes' AND A.Furnished = 'Yes'
GROUP BY L.City;
"""
df = pd.read_sql(query, conn)
print(df)
```

3. How many properties are available for each payment option?

```
query = """
SELECT PO.Payment_Option, COUNT(P.Property_ID) AS Property_Count
FROM Property_Facts P
JOIN Payment_Option_Dim PO ON P.Payment_Option_ID = PO.Payment_Option_I
GROUP BY PO.Payment_Option;
"""
df = pd.read_sql(query, conn)
print(df)
```

4. What is the highest-priced property in each city?

```
query = """
SELECT L.City, MAX(P.Price) AS Highest_Price
FROM Property_Facts P
JOIN Location_Dim L ON P.Location_ID = L.Location_ID
GROUP BY L.City;
"""
df = pd.read_sql(query, conn)
print(df)
```

5. Which type of property has the most listings?

```
query = """
SELECT T.Type, COUNT(P.Property_ID) AS Listing_Count
FROM Property_Facts P
JOIN Type_Dim T ON P.Type_ID = T.Type_ID
GROUP BY T.Type
ORDER BY Listing_Count DESC;
"""
df = pd.read_sql(query, conn)
print(df)
```

6. What is the total number of amenities for properties listed in each location?

```
query = """
SELECT L.Location_1, SUM(P.Total_Amenities) AS Total_Amenities
FROM Property_Facts P
JOIN Location_Dim L ON P.Location_ID = L.Location_ID
GROUP BY L.Location_1;
"""
df = pd.read_sql(query, conn)
print(df)
```

7. How many properties have a garden and a balcony?

```
query = """
SELECT COUNT(*) AS Property_Count
FROM Property_Facts P
JOIN Amenities_Dim A ON P.Amenities_ID = A.Amenities_ID
WHERE A.Garden = 'Yes' AND A.Balcony = 'Yes';
"""
df = pd.read_sql(query, conn)
print(df)
```

DISCOVERING AND INSIGHTS •

- What is the minimum property price in each city?
- What is the average down payment required for properties in each location?
- How many properties are listed by each ownership type?
- What is the total price of all properties listed in each city?
- What is the highest price of properties in each type category?
- Find all properties that are fully equipped and have a natural gas connection.
- How many properties are listed each month?
-

8. What is the minimum property price in each city?

```
query = """
SELECT L.City, MIN(P.Price) AS Min_Property_Price
FROM Property_Facts P
JOIN Location_Dim L ON P.Location_ID = L.Location_ID
GROUP BY L.City;
"""
df = pd.read_sql(query, conn)
print(df)
```

9. What is the average down payment required for properties in each location?

```
query = """
SELECT L.Location_1, AVG(P.Down_Payment) AS Average_Down_Payment
FROM Property_Facts P
JOIN Location_Dim L ON P.Location_ID = L.Location_ID
GROUP BY L.Location_1;
"""
df = pd.read_sql(query, conn)
print(df)
```

10. How many properties are listed by each ownership type?

```
query = """
SELECT O.Ownership, COUNT(P.Property_ID) AS Property_Count
FROM Property_Facts P
JOIN Ownership_Dim O ON P.Ownership_ID = O.Ownership_ID
GROUP BY O.Ownership;
"""
df = pd.read_sql(query, conn)
print(df)
```

11. What is the total price of all properties listed in each city?

```
query = """
SELECT L.City, SUM(P.Price) AS Total_Price
FROM Property_Facts P
JOIN Location_Dim L ON P.Location_ID = L.Location_ID
GROUP BY L.City;
"""
df = pd.read_sql(query, conn)
print(df)
```

12. What is the highest price of properties in each type category?

```
query = """
SELECT T.Type, MAX(P.Price) AS Highest_Price
FROM Property_Facts P
JOIN Type_Dim T ON P.Type_ID = T.Type_ID
GROUP BY T.Type;
"""
df = pd.read_sql(query, conn)
print(df)
```

13. Find all properties that are fully equipped and have a natural gas connection.

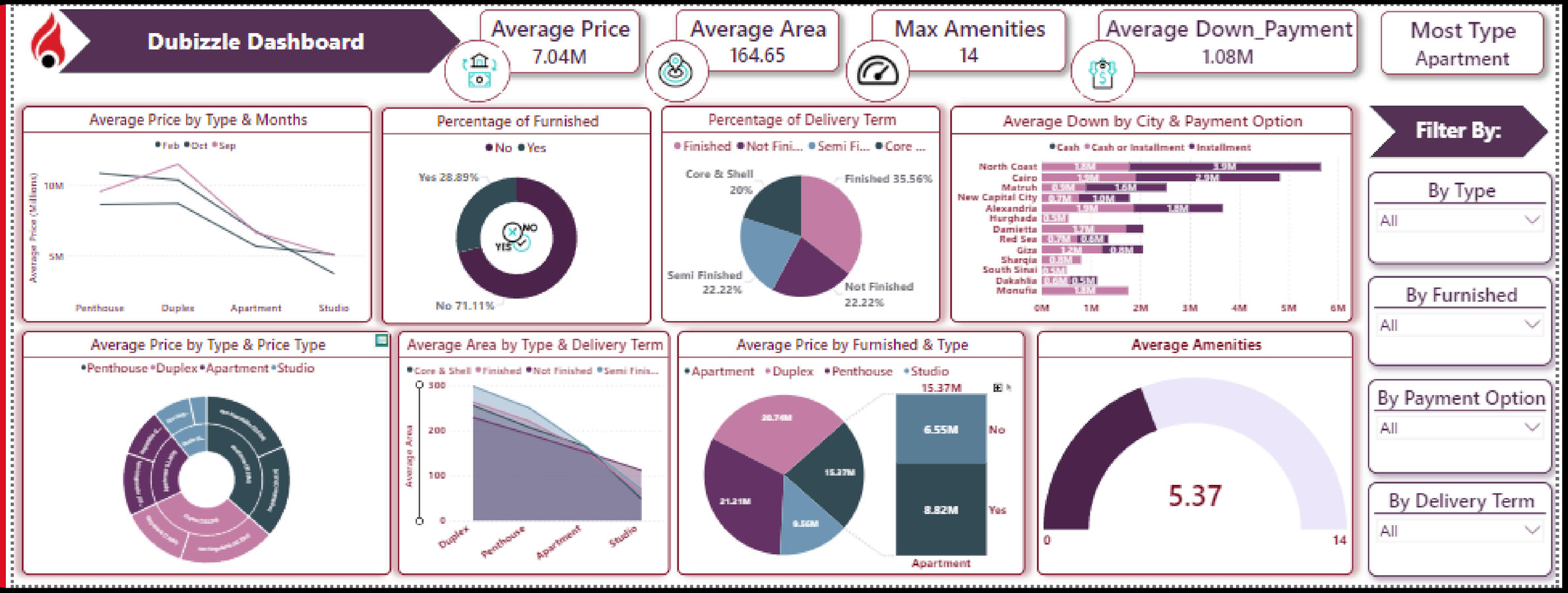
```
query = """
SELECT P.*
FROM Property_Facts P
JOIN Amenities_Dim A ON P.Amenities_ID = A.Amenities_ID
WHERE A.Fully_Equipped = 'Yes' AND A.Natural_Gas = 'Yes';
"""
df = pd.read_sql(query, conn)
print(df)
```

14. How many properties are listed each month?

```
query = """
SELECT DATE_FORMAT(P.Listing_Date, '%Y-%m') AS Month, COUNT(P.Property_ID) AS Property_Count
FROM Property_Facts P
GROUP BY Month;
"""
df = pd.read_sql(query, conn)
print(df)
```

DASHBOARD:(USING POWER BI) •

DASHBOARD:(USING POWER BI) •



• REPORT:(USING POWER BI)

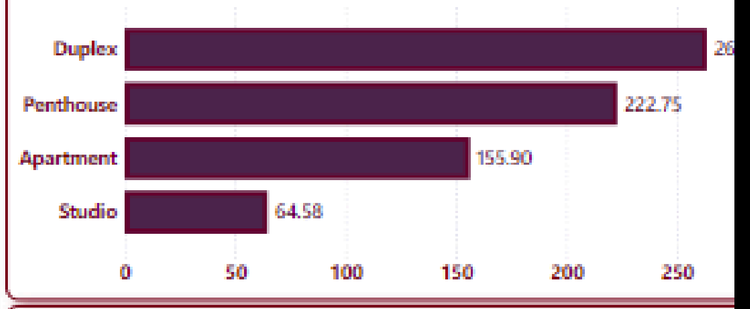
Dubizzle Report

Number Of Type
4

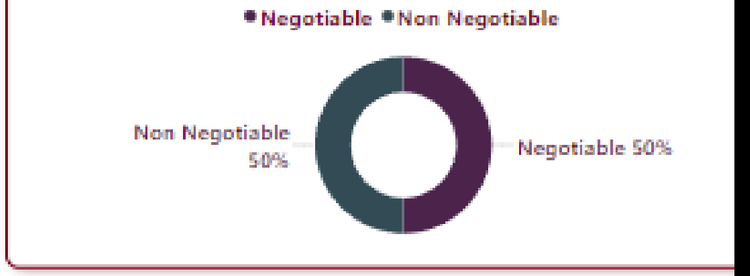
Overview about dataset

Type	Area (m^2)	Bathrooms	Bedrooms	Total_Amenities	Furnished	Delivery Term	Completion status	Level	Price Type	Payment Option	Down Payment	Price
Apartment	10	1	1	9	No	Finished	Off-plan	2	Non Negotiable	Cash or Installment	380000	3800000
Apartment	15	1	1	9	No	Finished	Off-plan	1	Non Negotiable	Cash or Installment	202500	2025000
Apartment	15	2	3	0	No	Finished	Off-plan	2	Non Negotiable	Cash or Installment	863500	3454000
Apartment	20	1	1	9	No	Finished	Off-plan	5	Non Negotiable	Cash or Installment	760000	7600000
Apartment	32	1	1	11	Yes	Finished	Off-plan	15	Non Negotiable	Installment	496000	4960000
Apartment	35	1	1	9	No	Finished	Off-plan	5	Non Negotiable	Cash or Installment	447999	4479993
Apartment	38	1	1	7	Yes	Finished	Off-plan	3	Non Negotiable	Cash or Installment	1710000	6840000
Apartment	38	1	1	13	Yes	Finished	Ready	1	Non Negotiable	Cash or Installment	1700000	6800000
Apartment	38	1	1	13	Yes	Finished	Ready	4	Non Negotiable	Cash or Installment	1170000	6800000
Apartment	40	1	1	7	No	Finished	Ready	2	Non Negotiable	Cash or Installment	1530000	6120000
Apartment	40	1	1	8	No	Finished	Ready	2	Non Negotiable	Cash	0	1153000
Apartment	40	1	1	12	Yes	Finished	Ready	0	Non Negotiable	Cash	0	2300000
Apartment	40	1	1	13	Yes	Finished	Ready	1	Non Negotiable	Cash or Installment	1500000	6300000
Apartment	40	1	1	13	Yes	Finished	Ready	3	Non Negotiable	Cash or Installment	1150000	6300000
Apartment	40	1	1	13	Yes	Finished	Ready	3	Non Negotiable	Cash or Installment	1500000	6300000
Apartment	40	1	1	13	Yes	Finished	Ready	5	Non Negotiable	Cash or Installment	1150000	6300000
Apartment	41	1	1	11	Yes	Finished	Ready	7	Non Negotiable	Cash	0	3300000
Apartment	44	1	1	3	No	Finished	Ready	0	Negotiable	Cash	0	400000
Apartment	45	1	1	4	No	Finished	Ready	8	Non Negotiable	Cash	0	2000000
Apartment	50	1	1	3	No	Finished	Off-plan	0	Negotiable	Cash or Installment	1500000	1500000
Apartment	50	1	1	4	Yes	Finished	Ready	0	Non Negotiable	Cash	0	2500000
Apartment	50	1	1	7	No	Finished	Ready	12	Non Negotiable	Cash	0	850000
Apartment	50	1	1	8	No	Finished	Off-plan	0	Non Negotiable	Installment	512000	1570000

Average of Area (m^2) by Type



Price Type

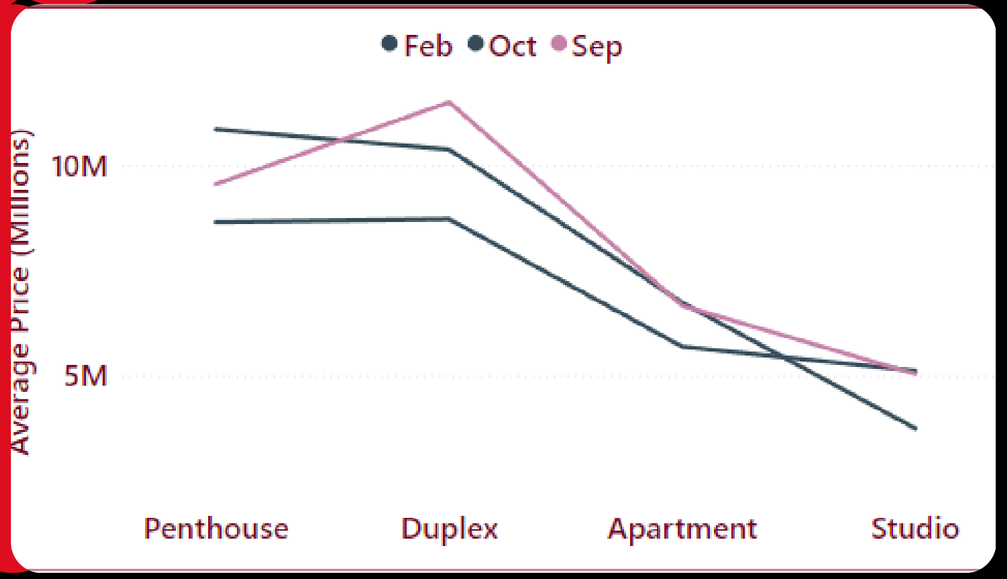


Count of City

INSIGHTS

- Average price is 7.04M
  - The average price of 7.04 million suggests a relatively high market for properties, potentially indicating a premium real estate sector.
  - With an average area of 164.65 m^2, properties in this category may be spacious
- The maximum number of amenities is 14, indicating that the properties likely offer a wide range of features. This could enhance their attractiveness to potential buyers
- An average down payment of 1.08 million represents approximately 15.3% of the average property price. This suggests that financing options may be accessible
- The top property type sold is Apartment
- Number of type sold is 4

INSIGHTS:



In February:

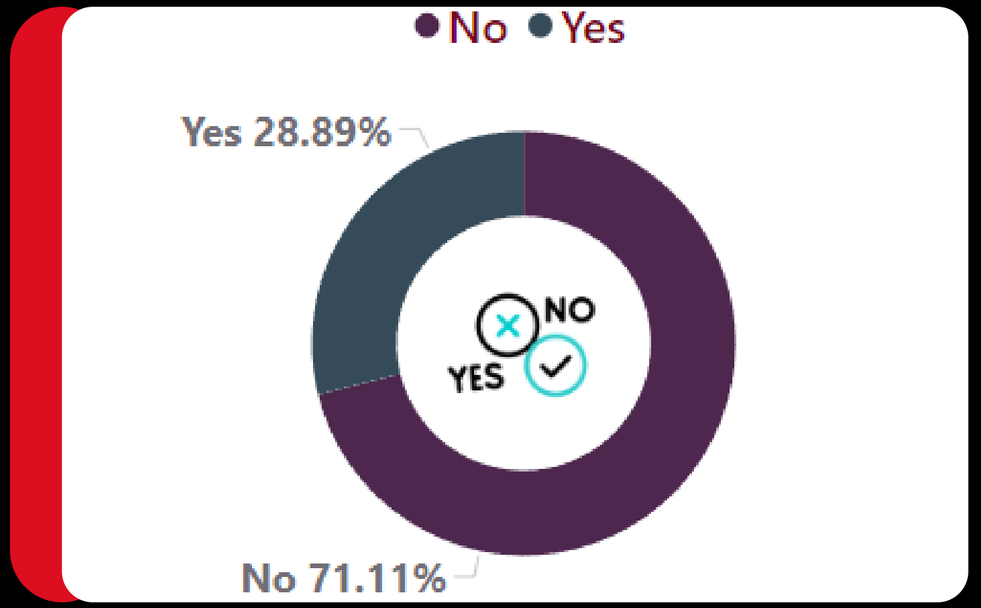
- **The Duplex is the expensive type and it is about 10M**
- **This suggests that the Duplex was a sought-after property type during this month, possibly due to features such as larger living space or unique architectural elements.**

In the September

- **The Duplex again emerges as the most expensive type but with an increased average price of 11 million.**
- **The price increase from February to September indicates rising demand**

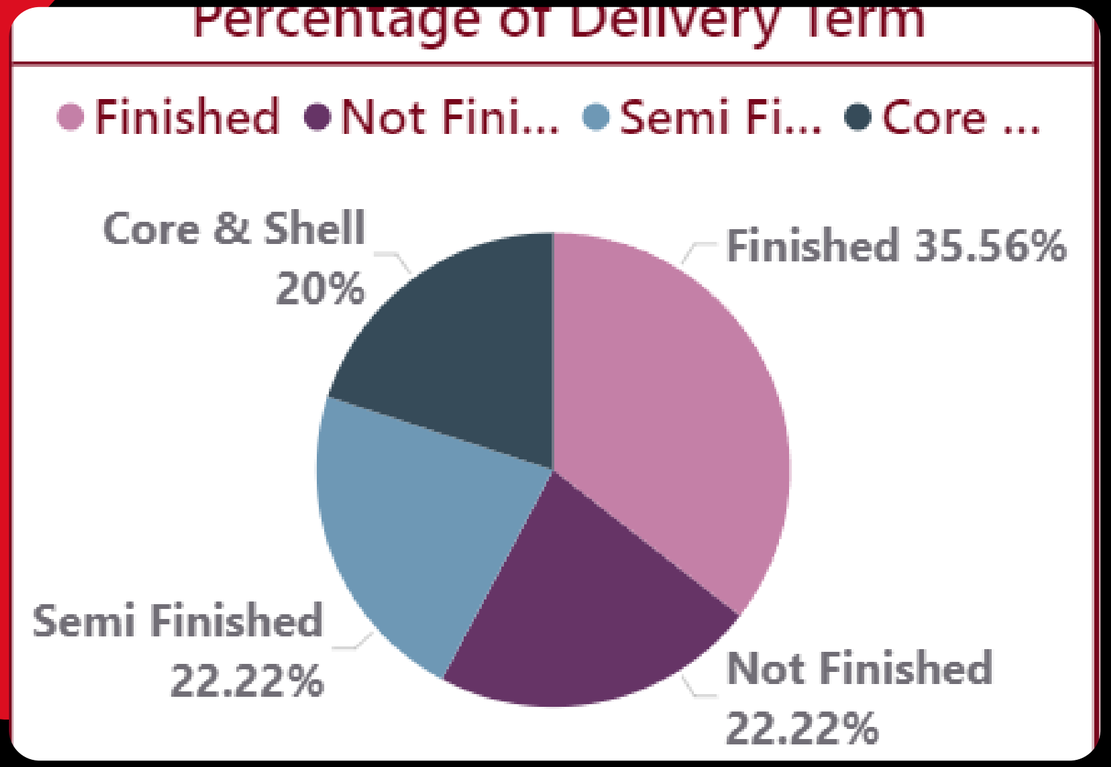
In October

- **The Penthouse takes over as the most expensive type, also with an average price of 10 million**

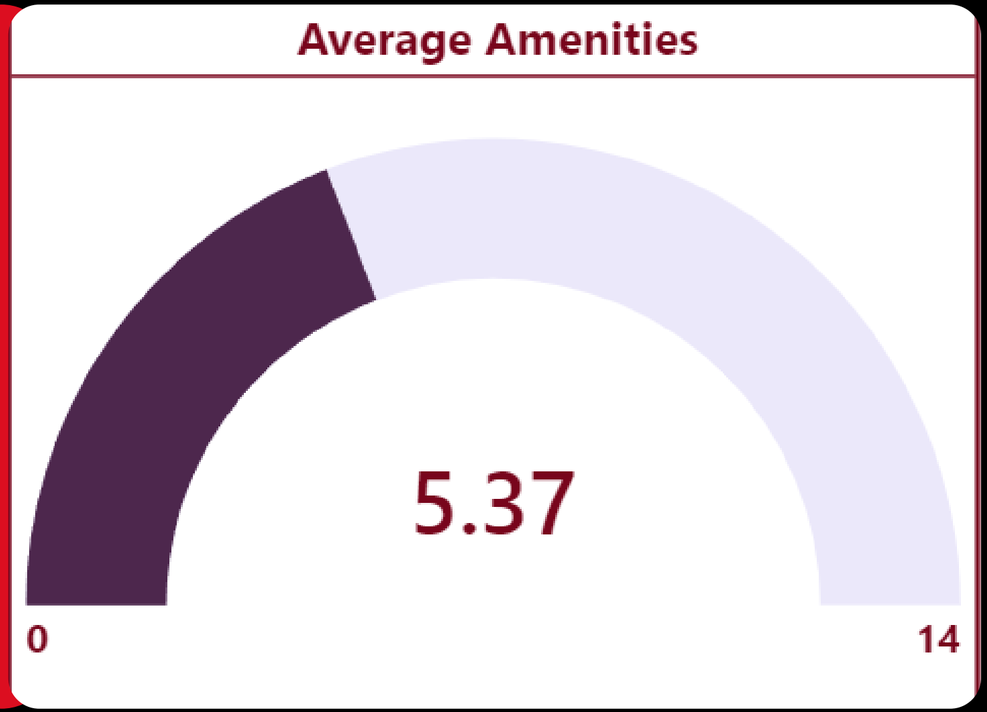


- 28.29% of the properties are furnished. This suggests that a significant portion of the market offers move-in-ready options, which can be appealing to buyers looking for convenience, especially first-time buyers, expatriates, or investors renting out properties.
- 71.11% of the properties are unfurnished. This indicates a substantial majority of the market consists of unfurnished properties

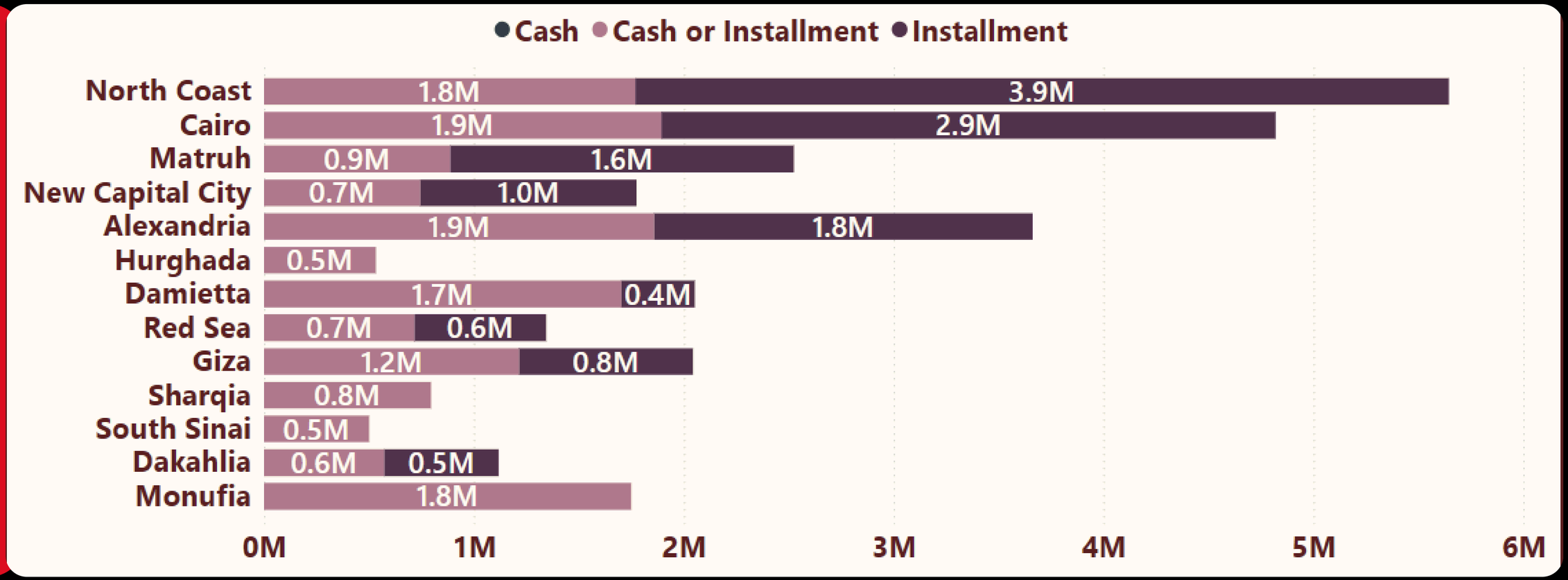
INSIGHTS:



- 20% of the properties are sold as core & shell. This means that these properties are delivered without any internal finishes, allowing buyers to customize their space according to their preferences
- 35.56% of the properties are delivered finished. This option provides buyers with a move-in-ready experience
- 22.22% of the properties fall under the semi-finished category.



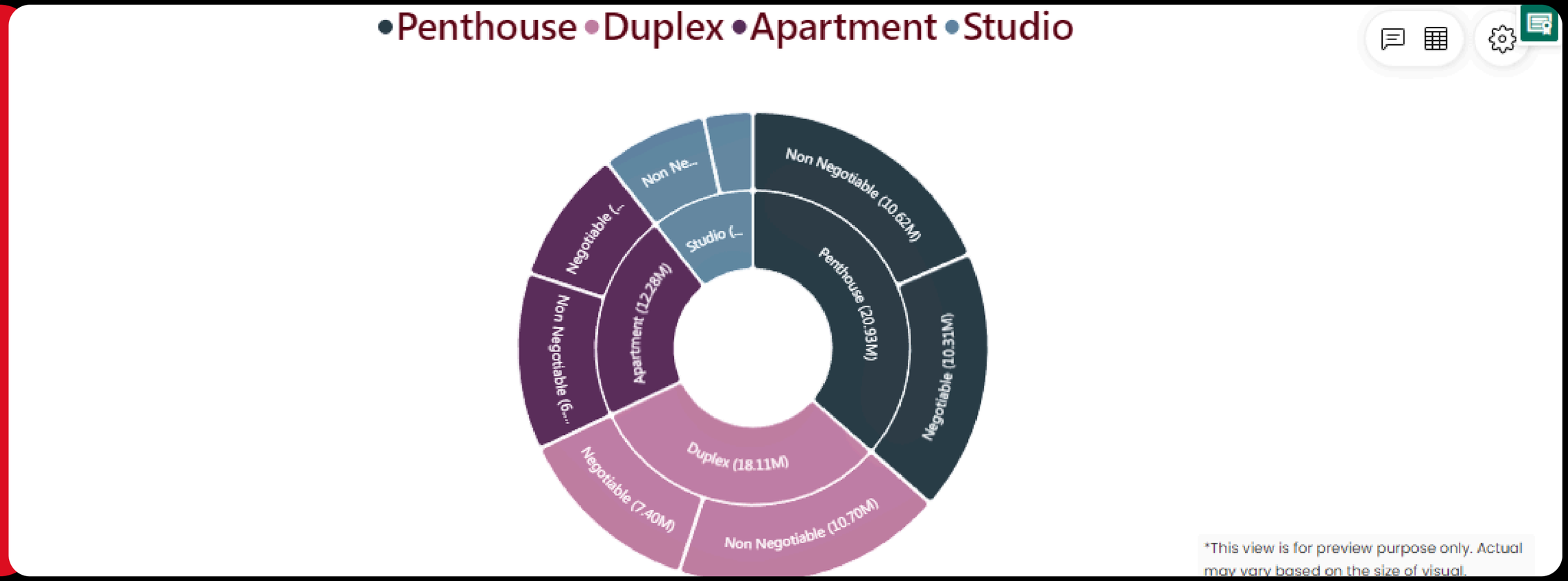
An average of 5.37 amenities suggests that properties offer a moderate level of additional features. This number is significant enough to enhance the living experience without overwhelming potential buyers or renters.



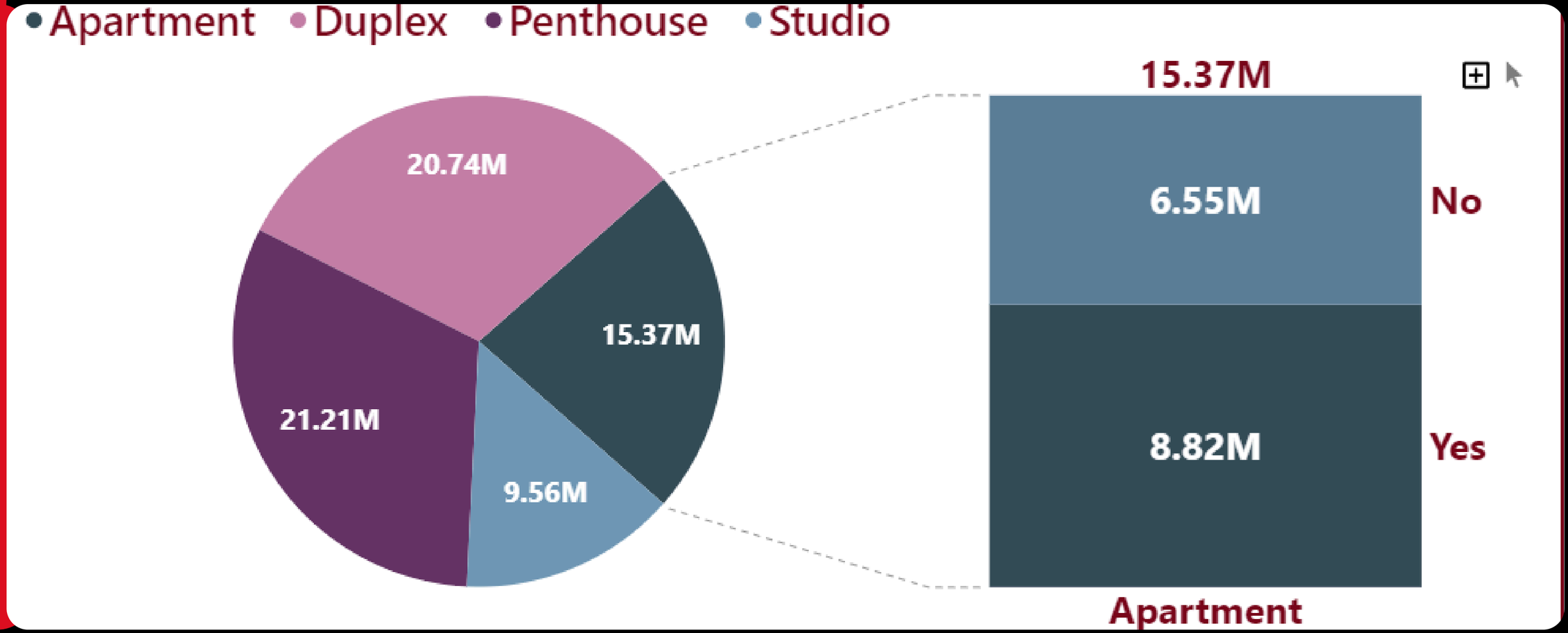
1. **Highest Down Payment in North Coast:**
   - North Coast has the highest average down payment, especially for "Installment" options, followed by "Cash or Installment." This suggests a high demand or premium real estate market in that area where people are willing to commit to larger payments.

2. **Cairo and New Capital City Patterns:**

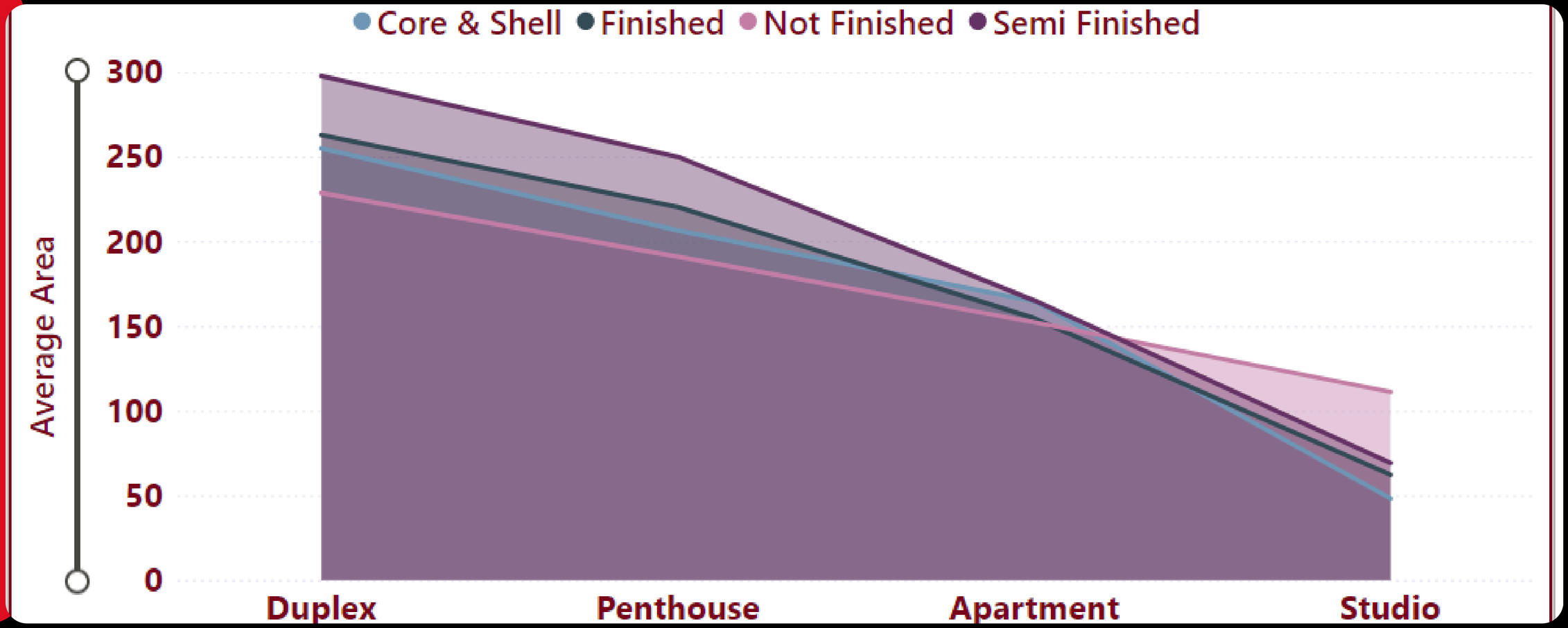
Cairo and New Capital City show considerable down payments, with Installment options slightly dominating. This may indicate that installment plans are popular in high-demand urban areas where buyers prefer spreading costs over time.



1. **Highest Prices for Duplexes:**
   - **Duplexes have the highest average price (18.11M), particularly in the "Non-Negotiable" category. This suggests that duplex properties are marketed to buyers with a higher budget and less flexibility for negotiation.**



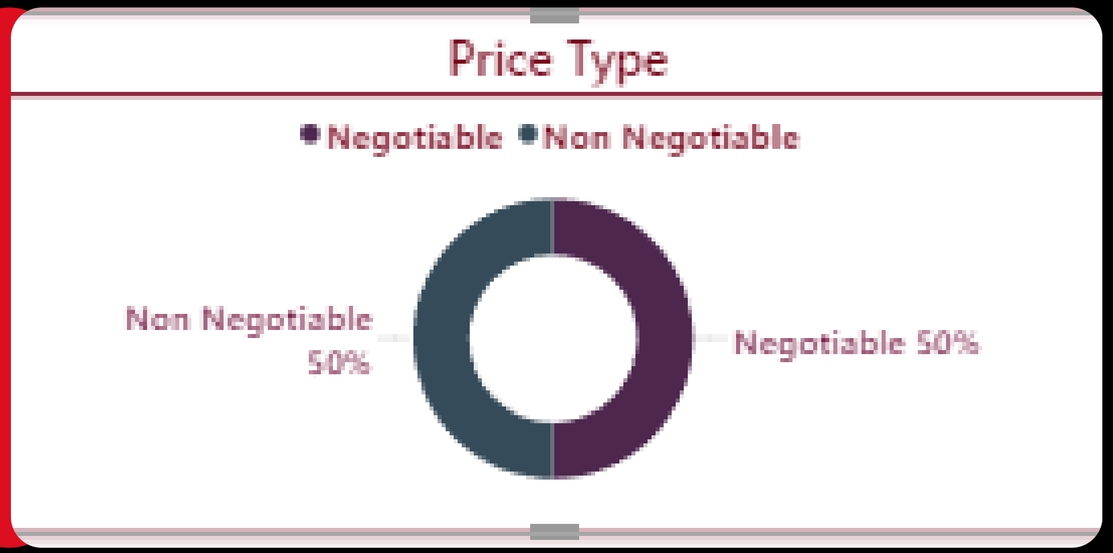
1. **Average Prices by Property Type:**
   a. The highest average price is observed for Penthouse properties (21.21M), followed by Duplexes (20.74M).
   b. Apartments have an average price of 15.37M, while Studios have the lowest average price (9.56M).
2. **Value Impact of Furnishing:**
   a. Furnishing adds a notable value to all
   b. ex.to apartments, as indicated by the price difference of approximately 2.27M between furnished and unfurnished options.



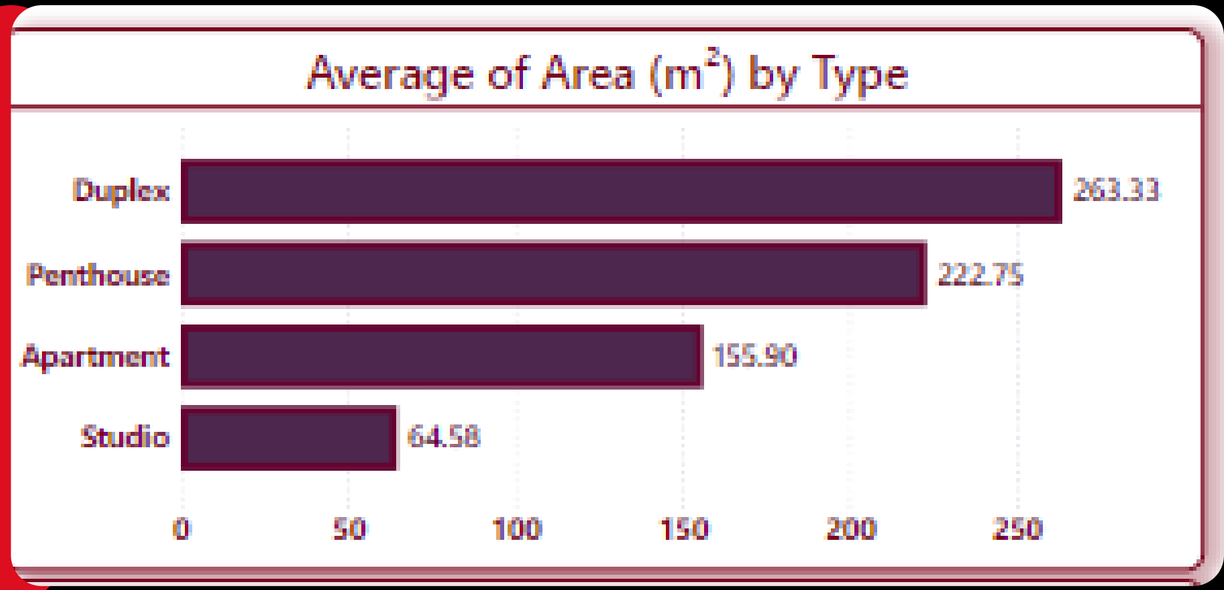
1. **Average Area by Property Type:**
   a. Duplexes have the largest average area across all delivery terms, followed by Penthouses, Apartments, and Studios, with Studios having the smallest average area.
2. **Impact of Delivery Terms:**
   - Semi Finished properties generally show larger average areas than other delivery terms across all property types.
   - Not Finished properties have the smallest average area across most property types,

1. **Cairo stands out as the city with the highest number of property listings, suggesting it's the primary location for real estate activity.**
2. **Alexandria, cost and North city also have notable property listings, though they are smaller in count compared to Cairo.**
3. **Smaller cities and regions like Minya, Dakahlia, and Qalyubia have fewer property listings, which may indicate lower real estate activity or demand in these areas.**

INSIGHTS: .



- Price type is moderate percentage



Duplexes have the largest average area, followed by Penthouses, Apartments, and Studios, with Studios having the smallest average area.

• OVERVIEW DATASET

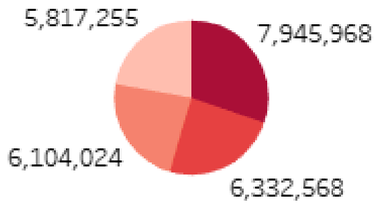
Overview about dataset

Type	Area (m^2)	Bathrooms	Bedrooms	Total_Amenities	Furnished	Delivery Term	Completion status	Level	Price Type	Payment Option	Down Payment	Price
Apartment	10	1	1	9	No	Finished	Off-plan	2	Non Negotiable	Cash or Installment	380000	3800000
Apartment	15	1	1	9	No	Finished	Off-plan	1	Non Negotiable	Cash or Installment	202500	2025000
Apartment	15	2	3	0	No	Finished	Off-plan	2	Non Negotiable	Cash or Installment	863500	3454000
Apartment	20	1	1	9	No	Finished	Off-plan	5	Non Negotiable	Cash or Installment	760000	7600000
Apartment	32	1	1	11	Yes	Finished	Off-plan	15	Non Negotiable	Installment	496000	4960000
Apartment	35	1	1	9	No	Finished	Off-plan	5	Non Negotiable	Cash or Installment	447999	4479993
Apartment	38	1	1	7	Yes	Finished	Off-plan	3	Non Negotiable	Cash or Installment	1710000	6840000
Apartment	38	1	1	13	Yes	Finished	Ready	1	Non Negotiable	Cash or Installment	1700000	6800000
Apartment	38	1	1	13	Yes	Finished	Ready	4	Non Negotiable	Cash or Installment	1170000	6800000
Apartment	40	1	1	7	No	Finished	Ready	2	Non Negotiable	Cash or Installment	1530000	6120000
Apartment	40	1	1	8	No	Finished	Ready	2	Non Negotiable	Cash	0	1153000
Apartment	40	1	1	12	Yes	Finished	Ready	0	Non Negotiable	Cash	0	2300000
Apartment	40	1	1	13	Yes	Finished	Ready	1	Non Negotiable	Cash or Installment	1500000	6300000
Apartment	40	1	1	13	Yes	Finished	Ready	3	Non Negotiable	Cash or Installment	1150000	6300000
Apartment	40	1	1	13	Yes	Finished	Ready	3	Non Negotiable	Cash or Installment	1500000	6300000
Apartment	40	1	1	13	Yes	Finished	Ready	5	Non Negotiable	Cash or Installment	1150000	6300000
Apartment	41	1	1	11	Yes	Finished	Ready	7	Non Negotiable	Cash	0	3300000
Apartment	44	1	1	3	No	Finished	Ready	0	Negotiable	Cash	0	400000
Apartment	45	1	1	4	No	Finished	Ready	8	Non Negotiable	Cash	0	2000000
Apartment	50	1	1	3	No	Finished	Off-plan	0	Negotiable	Cash or Installment	1500000	1500000

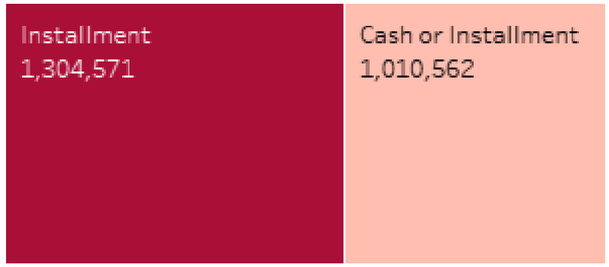
DASHBOARD:(USING TABLEAU)

Dubizzle Dashboard

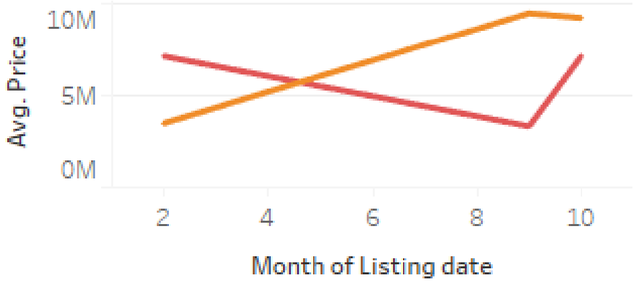
Price Distribution by Property Type



Payment Preference to downpayment



Monthly Listing Trend with Price and Ownership Insights



Type
- Apartment
- Duplex
- Penthouse
- Studio

Avg. Price
26,199,815

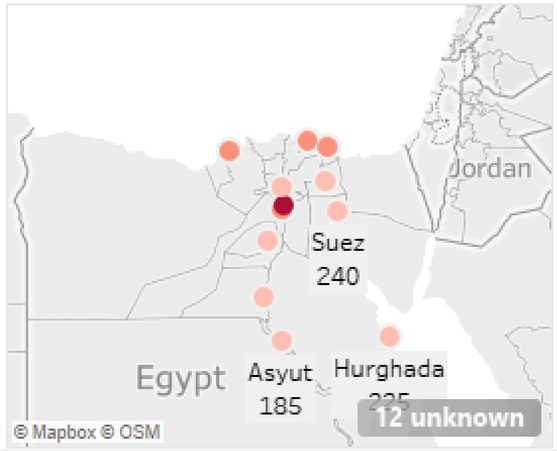
Max. Area (m²)
120 1,200

Ownership
- Primary
- Resale

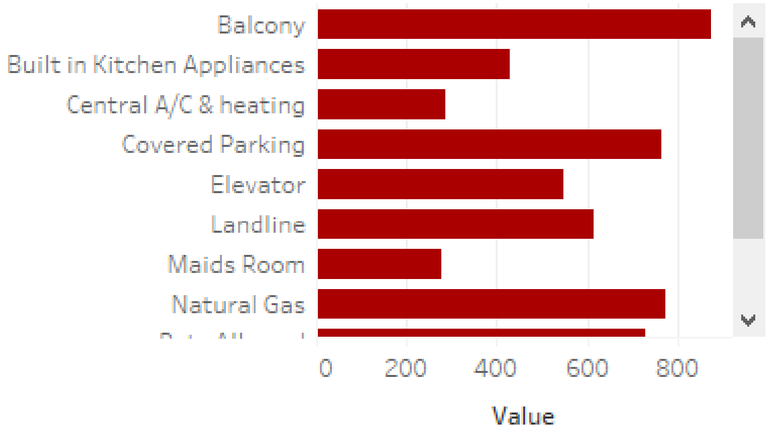
City
- Alexandria
- Cairo
- Damietta
- Gharbia
- Giza
- New Capital City

Avg. Down Payment
1,010,562 1,304,571

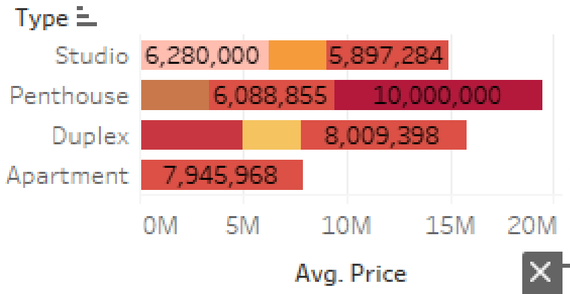
Location analysis



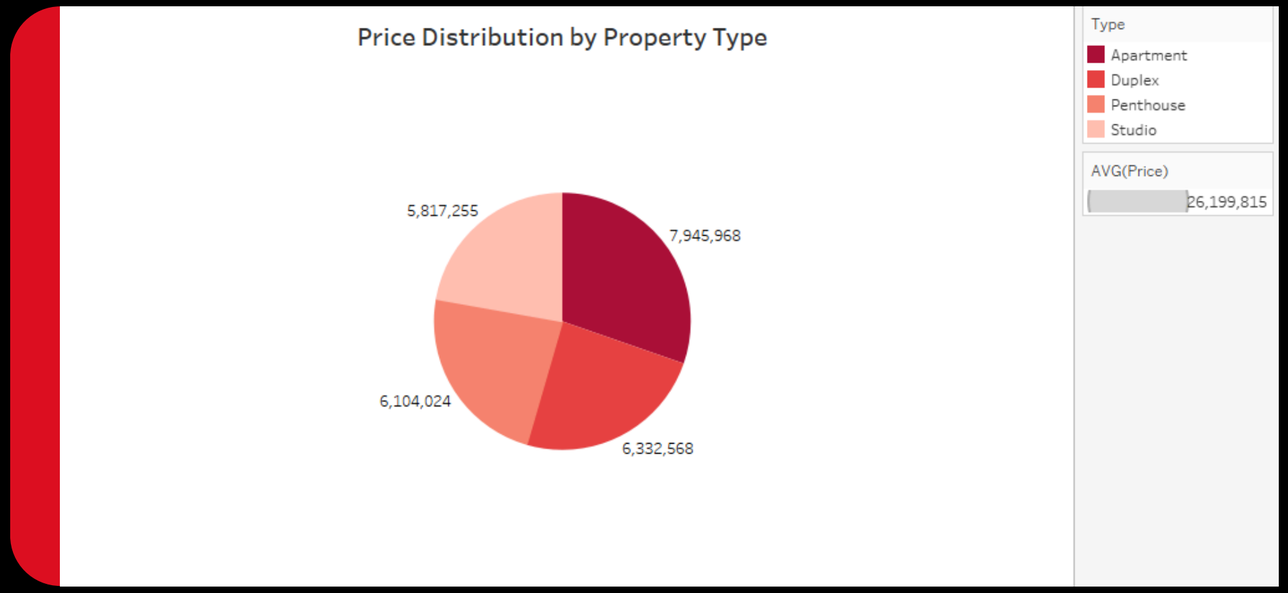
Amenities Impact on Property Price



Demand Trends by Property Type and Price Range



Avg. Price
7,043,605

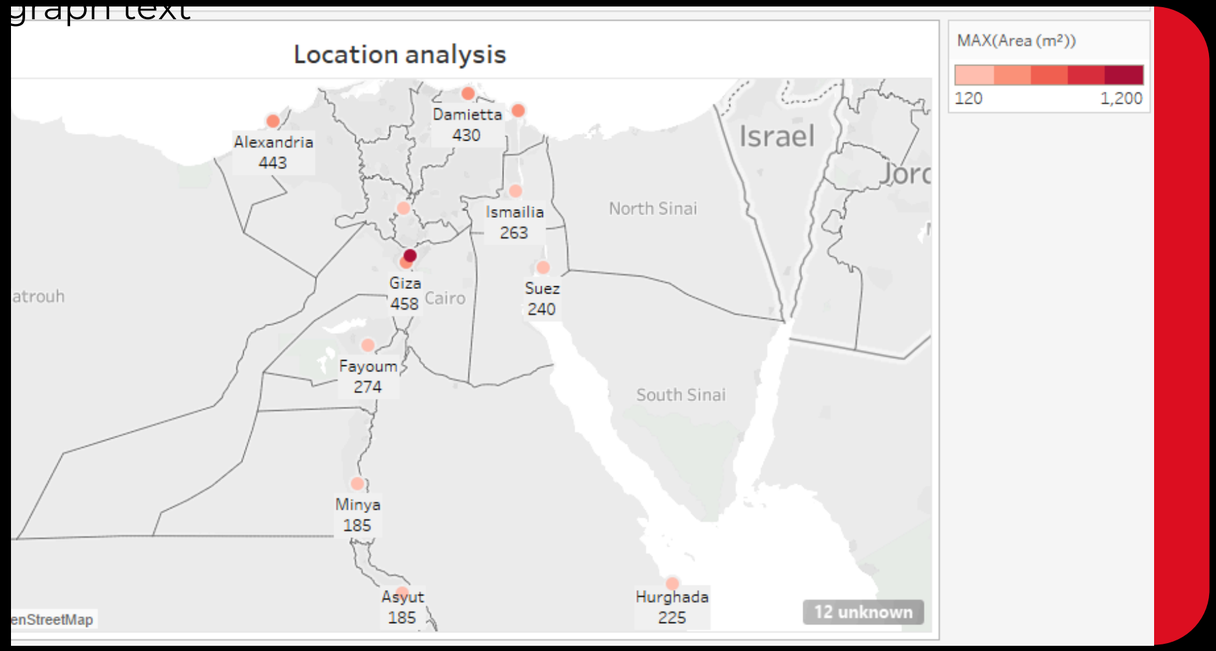


1. Price Distribution by Property Type

Apartments and Duplexes are among the most prevalent property types, likely indicating strong demand or a significant supply in the market for these types of properties.

2. Location analysis

- Cairo and Giza have the highest search activity, indicating high customer interest in specific locations within these cities. This might suggest popular residential or commercial demand in certain neighborhoods.
- Alexandria and Damietta also show significant search volumes, making them secondary but still key focus areas

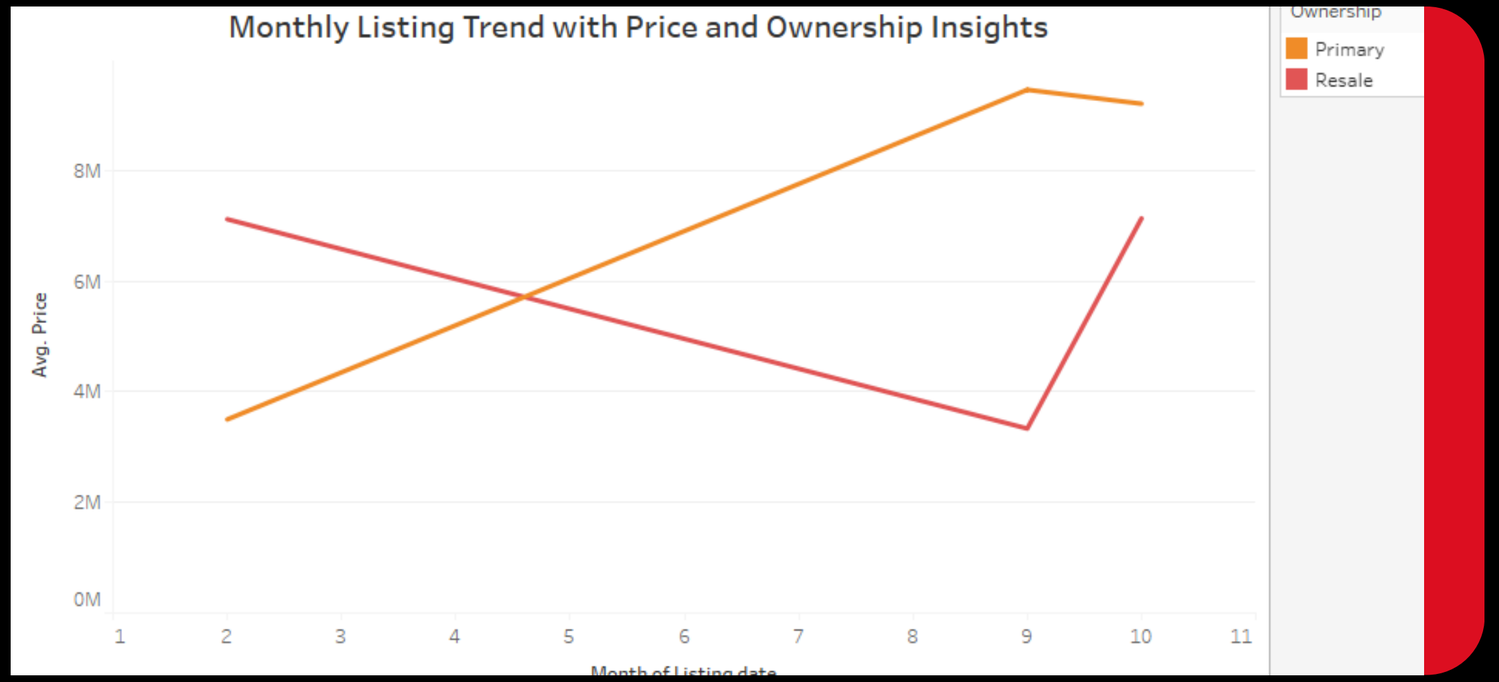


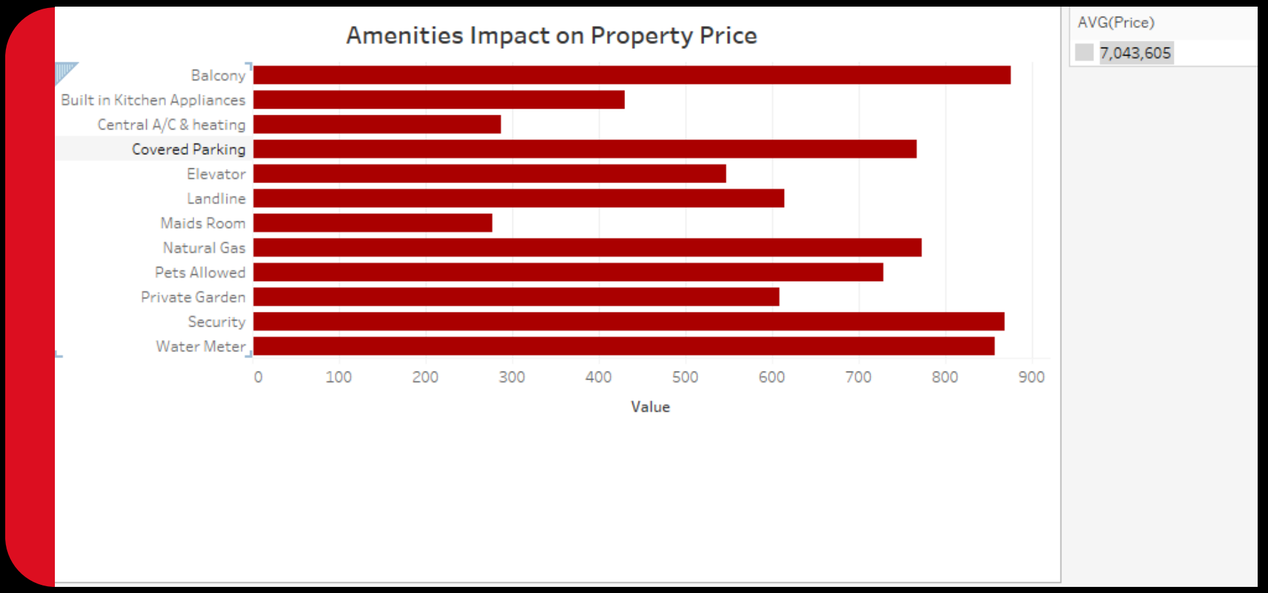
3. Demand Trends by Property Type and Price

penthouses have a strong presence in Alexandria. This suggests that there is a significant demand for premium, spacious properties in this city.
duplexes indeed have a strong representation in Cairo, with a price range that spans from mid to high levels. This suggests that there is a notable demand for spacious, premium properties in Cairo

4. Month listing trend with price and ownership insights

- Resale Prices: There was a decline in resale prices until September, after which prices began to increase through October.
- Primary Sale Prices: Primary sale prices saw a consistent increase over time until September, followed by a slight decline in October.



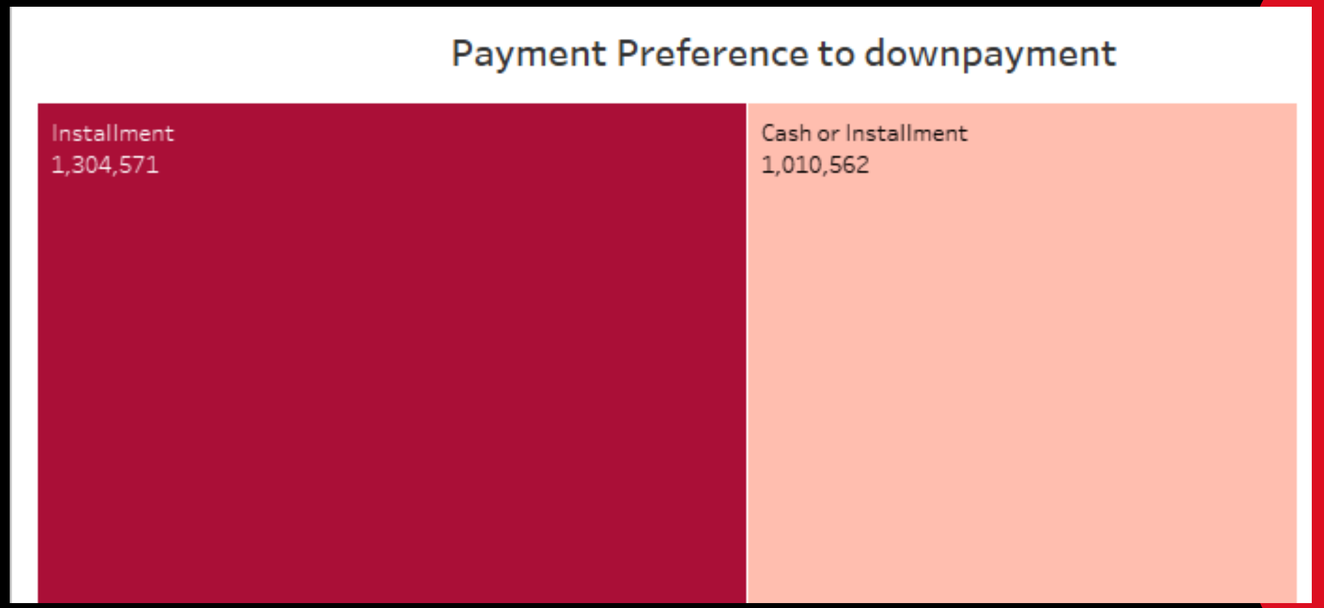


5. Amenities Impact on Property Pricemost

Amenities have a significant impact on property prices, especially when properties feature balconies and security. Most customers place high value on these amenities, which often leads to higher property prices

6. Payment Preference to downpayment Resale

Most customers prefer to pay the down payment in installment by 1,304,571 then cash or installment



CONCLUSION

1. High Average Property Price and Size: The average price of listed properties is around 7.04M, with an average area of approximately 164.65 square meters, indicating a focus on premium or spacious properties.

2. Predominant Furnishing Status: The majority of properties (71.11%) are unfurnished, suggesting a trend toward unfurnished listings, possibly appealing to buyers or renters who prefer customizable interiors.

3. Diverse Delivery Terms: Properties are offered across different delivery terms, with a substantial portion (35.56%) already finished. This variety provides flexibility to buyers interested in different stages of construction.

4. Payment Options by Region: There is a clear preference for installment payments across various regions paying in cash payment or installment are also significant in certain areas. This flexibility in payment options help our customers

RECOMENDATIONS

1. Expand Furnished Property Listings: Since 71.11% of listings are unfurnished, consider increasing the number of furnished properties. so we recomend to increase furnishing property This can attract renters or buyers who prefer move-in-ready options, potentially broadening the customer base.

2. Offer More Flexible Payment Options: With installment plans showing significant demand, increasing flexible financing options could attract more buyers, particularly in premium areas where prices are high.

3. Promote Ready-to-Move Properties: Since a significant portion of properties are already finished (35.56%), highlight "ready-to-move" properties prominently in listings. This can attract buyers who prefer to move in immediately without waiting for construction to complete.

4. Properties with a high number of amenities (above the average of 5.37) can be highlighted as "premium" or "luxury" listings. This will appeal to buyers looking for a superior living experience and can justify higher listing prices, attracting a more affluent audience.

See You Next

THANK YOU

2024 Dubizzle Presentation