Jan Bauer - Selected Works

Binz Wind - Text

GUIDEURBANIST INTENTIONS Stadtplan Towards 3 sides, the Binz is distanced by slopes from the surrounding Friesenberg garden-city.

Today we see a patch of the rare regulatory zone IG III drawn along the shape of the former clay pit.

The zone meets almost-urban residential substance on the fourth, most narrow side.

The seam where the two regulatory zones meet could host some urban gesture along the lines of Announcing the Binz or Gate to Binz, negotiating the confrontation.

Instead, we find a warehouse-sized wall of offices stretching 130m right along the border.

The erection dubbed B2Binz was completed in 2023 and carries the modernist core principle of separation strong.

Modell B2Binz is an excellent illustration of the shortcomings of the BZO building code.

BZO Art. 19 Abs. 4 dictates that the building line setback increases by the same amount that a building is higher than 12m.

With noble intentions, this regulation seeks to mitigate the impact of the clash between business and residential bodies, to the benefit of the latter.

In order to maximise HNF m2 , the bureau vessel jumps back precisely as the rule dictates.

Collage 1 (feels like) The character stays wall.

Collage 2 (could, should) Changing face towards the residential

Soften the impact of 130m strict repetition with more depth, change and action

Situationsplan Permeability for the urban context

2 Sides business ravine and residential vibe corridor are connected by passages in two places on the ground floor

Passages are a continuation of the residential streets

Bande Actif Collage The Bande Actif idea is to move all wet spaces outside of the apartment. This activates the facade to become a true display of life happening in a building.

In a representative facade, the size and shape of windows might let you guess whats behind

Bande Actif is not guessing.

GUIDESUFFIZIENZ Section/FloorplanWhen touching an existing construction the ambition is to repurpose with minimalist destruction.

No drilling or cutting concrete: Housing needs additional riser shafts, all are mounting on the facade

The housing gets everything it needs via the facade

Floorplan/Axo Baukasten Access to the active band for a chamber means removing an element of the existing facade system

The rhythm of chambers and accesses seeks to minimise the removal ratio

Floorplan Keeping the openness the floor plan advertises, but introducing beneficial compartments

The basic private units can be linked to serve many shapes and sizes of cohabitation

Visus At night, activity in the inner bedrooms and living areas would be apparent from low-level illumination interrupted by that of

kitchens and bathrooms used only intermittently and for short periods.

From the outside, the building facade would truthfully indicate the living going on within

Binz Wind - Images

Image: Invis myth

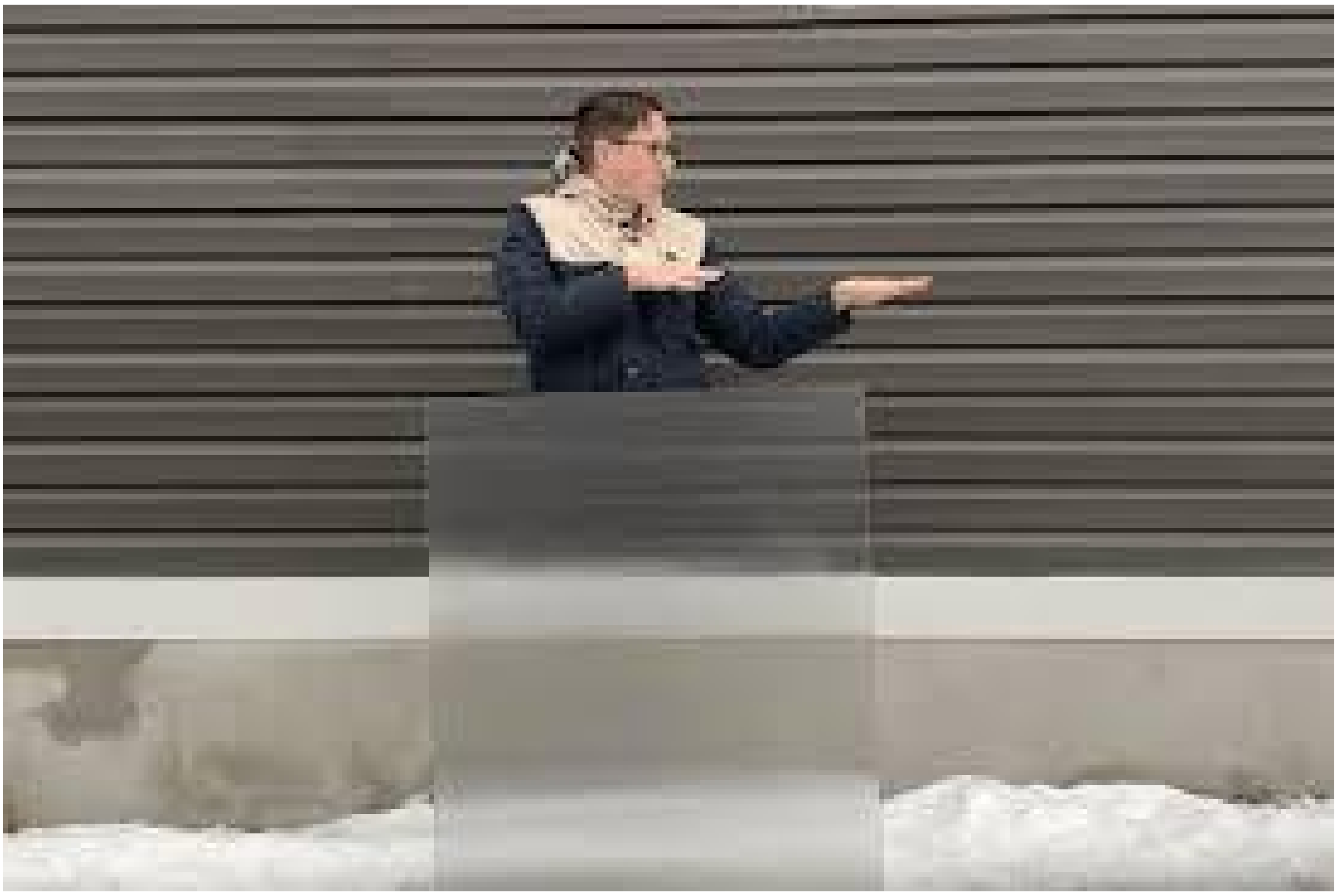


Image: Schwarzplan extended

Image: Sdvhiwedf

Image: Col 3

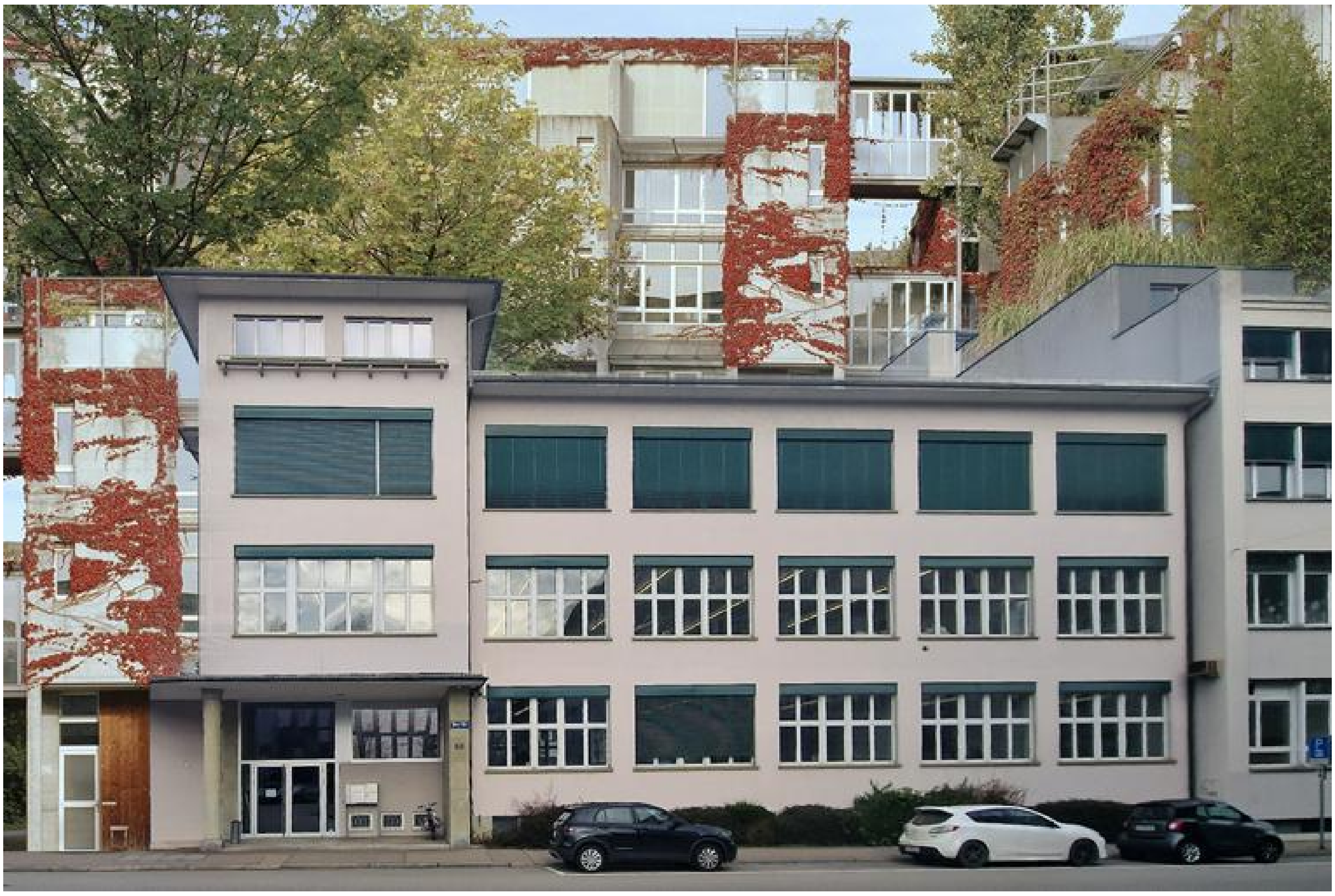
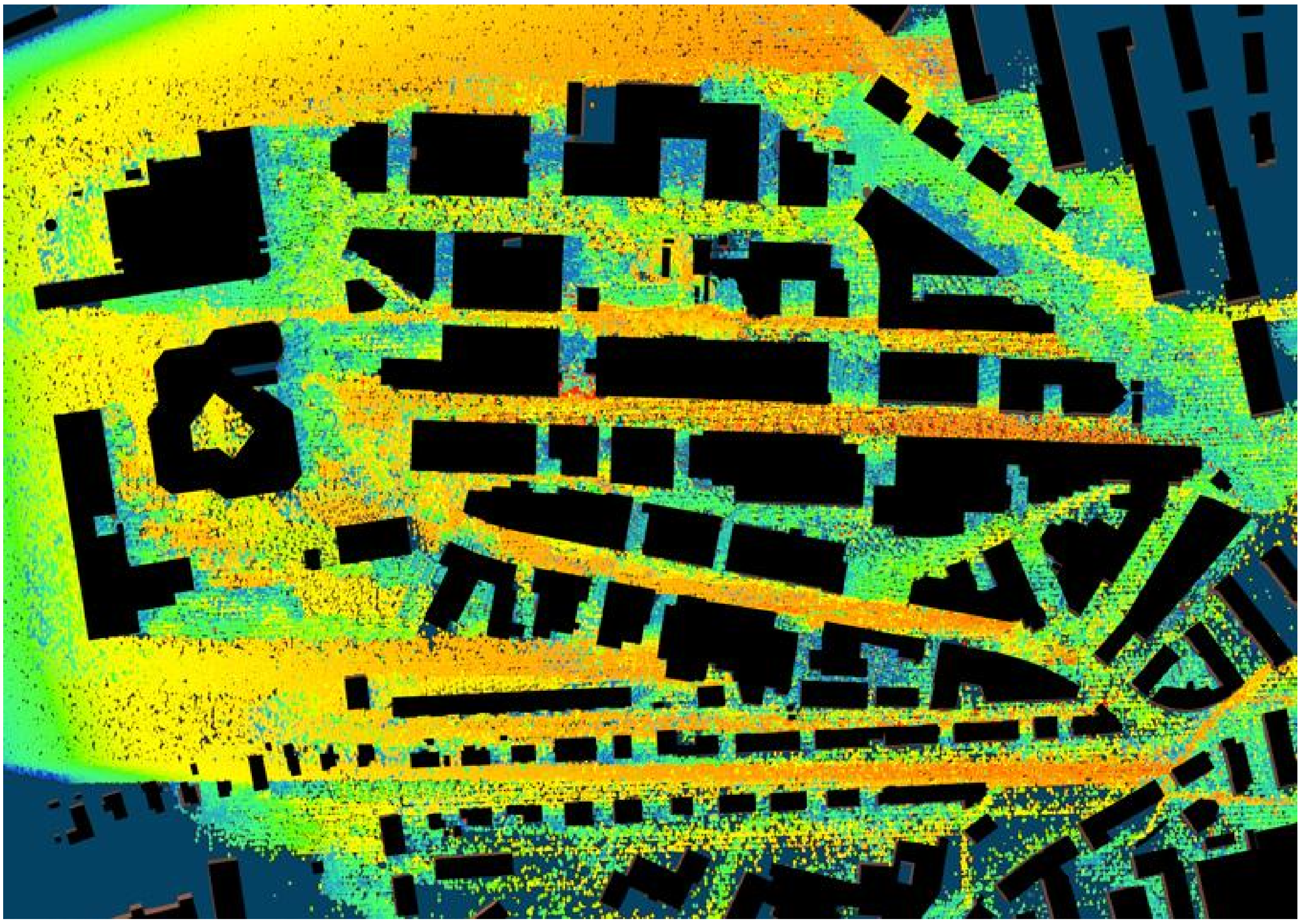


Image: Windhund lu?chtet



Binz Wind - PDFs

Binz Wind - Code

Notebook: Binz_Numbers_Game.ipynb

[Code Cell]

```
#@title Wie heisst das Gebäudedatä file? { run: "auto", display-mode: "form" }
import pandas as pd
import os
import matplotlib.pyplot as plt
import numpy as np

csv_file_path = "buildings_data.csv"#@param {type:"string"}
data = pd.read_csv(csv_file_path)

# Function to load existing data from CSV or create it if it doesn't exist
def load_existing_data(file_path):
    if os.path.exists(file_path):
        return pd.read_csv(file_path)
    else:
        # Create an empty DataFrame with the specified columns if the CSV does not exist
        df = pd.DataFrame(columns=data_columns)
        # Save the empty DataFrame to a new CSV file to create it
        df.to_csv(file_path, index=False)
        return df

# Load or create the CSV file
buildings_data = load_existing_data(csv_file_path)

print('Merci')
print(f"Du hesch es tiptops file mit {len(buildings_data)} definierte Gebäude
hinzuegfüegt")
print(data)
```

[Code Cell]

```
#@title Neui Gebäude hinzuefüege? { run: "auto", display-mode: "form" }

# Define the columns for the DataFrame, adding 'Floors'
data_columns = ['Area Name', 'Building ID', 'Footprint Area (m²)', 'Height (m)',
'Floors', 'Geschossfläche (m²)', 'Parzelle']

# Function to save data to CSV
def save_data_to_csv(df, file_path):
    df.to_csv(file_path, index=False)

# Function to calculate Geschossfläche, modified to also return floors
def calculate_geschossfläche(footprint_area, height, floor_height=3):
    floors = max(1, height // floor_height)
    geschossfläche = footprint_area * floors
    return floors, geschossfläche
```

```
# Function to add a building entry
def add_building_data(area_name, footprint_area, height, parzelle, df):
    floors, geschossfläche = calculate_geschossfläche(footprint_area, height)
    building_id = len(df) + 1 if len(df) > 0 else 1
     new_entry = pd.DataFrame([[area_name, building_id, footprint_area, height, floors,
geschossfläche, parzelle]], columns=data_columns)
    updated_df = pd.concat([df, new_entry], ignore_index=True)
    return updated_df

# Function to remove a building entry
def remove_building_data(building_id, df):
    return df[df['Building ID'] != building_id].reset_index(drop=True)

# Input loop for adding/removing data
while True:
    action = input("Do you want to add or remove a building, or quit? (add/remove/quit):
")
    if action.lower() == 'add':
        area_name = input("Enter the name of the area: ")
        try:
            footprint_area = float(input("Enter building footprint area (m²): "))
            height = float(input("Enter building height (m): "))
               parzelle = int(input("Enter Parzelle number: "))  # Prompt for Parzelle
number
                  buildings_data = add_building_data(area_name, footprint_area, height,
parzelle, buildings_data)
            save_data_to_csv(buildings_data, csv_file_path)
            print("Building added.")
        except ValueError:
              print("Invalid input. Please enter numerical values for area, height, and
Parzelle.")
    elif action.lower() == 'remove':
        try:
            building_id = int(input("Enter the Building ID to remove: "))
            buildings_data = remove_building_data(building_id, buildings_data)
            save_data_to_csv(buildings_data, csv_file_path)
            print("Building removed.")
        except ValueError:
            print("Invalid input. Please enter a valid Building ID.")
    elif action.lower() == 'quit':
        break
    else:
        print("Invalid action. Please enter 'add', 'remove', or 'quit'.")

# Display final DataFrame
print("\nFinal Buildings Data:")
print(buildings_data)
```

[Code Cell]

```
#@title Numbers Game spilä { run: "auto", display-mode: "form" }
```

```
# Functions
def update_geschossfläche_for_aufstockung(df, new_height):
    for index, row in df.iterrows():
       new_residential_height = new_height - row['Height (m)']      # Recalculate floors
and Geschossfläche with the new height
          floors, geschossfläche = calculate_geschossfläche(row['Footprint Area (m²)'],
new_residential_height)
      df.at[index, 'Floors'] = floors
      df.at[index, 'Geschossfläche (m²)'] = geschossfläche
    return df

def get_numbers(scenario, data):
  print(f"\nEstimates Scenario: {scenario}\n")

  tot_gf_binz = data[data['Area Name'] == 'Binz']['Geschossfläche (m²)'].sum()
   print(f"\nTotal Geschossfläche in Binz for Scenario {scenario}: {round(tot_gf_binz)}
m²")

  est_apts = (tot_gf_binz / avg_apt_size) * 0.8
    print(f"\nEstimated number of apartments (based on {round(avg_apt_size)} m² per
apartment and the assumption that 80% of the total Geschossfläche are occupied by
apartements): {round(est_apts)}")

  est_tot_inhab = avg_pers_per_apt * est_apts
   print(f"\nEstimated number of inhabitants (based on {average_space_per_person} m² per
inhabitant): {round(est_tot_inhab)}")

  est_az = (tot_gf_binz / total_area_binz) * 0.8
  print(f"\n\nEstimated Ausnützungsziffer (based on total area of {total_area_binz} m²):
{round(est_az, 2)}")

       #print(f"\nTotal number of apartments in Zurich before {scenario}:
{round(init_total_apts_zh)}")

  new_empty_apts_zh = empty_apts_zh + est_apts
      #print(f"\nNew total of empty apartments in Zurich after {scenario}:
{round(new_empty_apts_zh)}")

  new_total_apts_zh = init_total_apts_zh + est_apts
      #print(f"\nTotal number of apartments in Zurich after rezoning Binz:
{round(new_total_apts_zh)}")

  new_empty_apts_ratio_zh = (new_empty_apts_zh / new_total_apts_zh) * 100
   print(f"\nScenario {scenario}, assuming the added apartments are vacant, brings the
Leerwohnungsziffer to a new value of: {new_empty_apts_ratio_zh:.2f}%")

  average_floors_binz = data[data['Area Name'] == 'Binz']['Floors'].mean()
    print(f"\nAverage number of floors per building in Binz for Scenario {scenario}:
{round(average_floors_binz, 2)}\n")

# Zurich & Binz numbers & averages
```

```
empty_apts_zh = 177
empty_apts_ratio_zh = 0.06
init_total_apts_zh = empty_apts_zh / (empty_apts_ratio_zh / 100)
init_empty_apts_binz = 0
init_inhab_binz = 64

avg_pers_per_apt = 2.18
average_space_per_person = 45 # @param {type:"integer"}
avg_apt_size = avg_pers_per_apt * average_space_per_person

total_area_binz = 159172#@param {type:"string"}  # in square meters (m²)

total_gf_büro_binz = total_area_binz * 1.5
avg_büro_size = 40#@param {type:"integer"} # in square meters

aufstockung_new_height = 40 # @param {type:"integer"}

# Set scenarios
original = True # @param {type:"boolean"}
radical = True # @param {type:"boolean"}
introduce_aufstockung = True # @param {type:"boolean"}
introduce_officehome = True # @param {type:"boolean"}

if radical:
  scenario = "Radical"
  data = pd.read_csv(csv_file_path)
  get_numbers(scenario, data)

if introduce_aufstockung:
  scenario = "Aufstockung"
  data = update_geschossfläche_for_aufstockung(buildings_data, aufstockung_new_height)
  get_numbers(scenario, data)

if introduce_officehome:
  scenario = "Officehome"
  est_inhabitans_officehome = total_gf_büro_binz / avg_büro_size
  est_apts_officehome = est_inhabitans_officehome / avg_pers_per_apt

  officehome_new_empty_apartments_zurich = empty_apts_zh + est_apts_officehome
  officehome_new_total_apartments_zurich = init_total_apts_zh + est_apts_officehome
   officehome_new_empty_apartments_ratio_zurich = (officehome_new_empty_apartments_zurich
/ officehome_new_total_apartments_zurich) * 100

  print("\nEstimates Officehome Binz:")
   print(f"\nTotal potential Büros with average Büro size per person of {avg_büro_size}
m²: {round(est_inhabitans_officehome)}\n")
     print(f"\nTotal number of apartments in Zurich after Officehoming Binz:
{round(officehome_new_total_apartments_zurich)}")
    print(f"\nNew total of empty apartments in Zurich after adding Officehome Binz:
{round(officehome_new_empty_apartments_zurich)}")
   print(f"\nOfficehoming Binz, assuming the added apartments are vacant, brings the
Leerwohnungsziffer to a new value of:
```

```
{officehome_new_empty_apartments_ratio_zurich:.2f}%\n")

# Plotting
plot = True # @param {type:"boolean"}

if plot:
  print("\nPlot:\n")
  # Labels for each scenario
  scenarios = ['Radical', 'Aufstockung', 'Officehome']

  # Data for plotting
  total_apartments = [empty_apts_zh, new_total_apts_zh,
officehome_new_total_apartments_zurich]
  leerwohnungsziffer = [empty_apts_ratio_zh, new_empty_apts_ratio_zh,
officehome_new_empty_apartments_ratio_zurich]
  estimated_inhabitants = [estimated_total_inhabitans, estimated_total_inhabitans,
est_inhabitans_officehome]

  x = np.arange(len(scenarios))  # the label locations
  width = 0.25  # the width of the bars

  fig, ax1 = plt.subplots(figsize=(12, 8))

  # Plotting total apartments
  rects1 = ax1.bar(x - width, total_apartments, width, label='Total Apartments',
color='SkyBlue')

  # Creating a twin of the original axis to plot the Leerwohnungsziffer
  ax2 = ax1.twinx()
  rects2 = ax2.bar(x, leerwohnungsziffer, width, label='Leerwohnungsziffer (%)',
color='IndianRed')

  # Plotting estimated inhabitants for each scenario
  rects3 = ax1.bar(x + width, estimated_inhabitants, width, label='Estimated
Inhabitants', color='LightGreen')

  # Adding some text for labels, title, and custom x-axis tick labels, etc.
  ax1.set_xlabel('Scenario')
  ax1.set_ylabel('Total Apartments and Estimated Inhabitants', color='black')
  ax2.set_ylabel('Leerwohnungsziffer (%)', color='black')
  ax1.set_title('Impact of Aufstockung and Officehome on Zurich Housing')
  ax1.set_xticks(x)
  ax1.set_xticklabels(scenarios)
  ax1.legend(loc='upper left')
  ax2.legend(loc='upper right')

  fig.tight_layout()
  plt.show()
else:
    print("\nKe plot (zum plotte es chrützli setzä)\n")
```

[Code Cell]

```
# Plot settings
fig, axes = plt.subplots(1, 3, figsize=(18, 6))  # Create 1 row of 3 plots
scenarios = ['Original', 'Aufstockung', 'Officehome']
colors = ['SkyBlue', 'IndianRed', 'LightGreen']
metrics = ['Total Apartments', 'Leerwohnungsziffer (%)', 'Estimated Inhabitants']
data = [total_apartments, leerwohnungsziffer, estimated_inhabitants]

# Iterate over each subplot to create individual plots
for i in range(3):
    axes[i].bar(scenarios, data[i], color=colors[i])
    axes[i].set_title(metrics[i])
    axes[i].set_xlabel('Scenario')
    axes[i].set_ylabel(metrics[i])
    for index, value in enumerate(data[i]):
        axes[i].text(index, value, f'{round(value, 2)}', ha='center', va='bottom')

# Adjust layout to prevent overlap
plt.tight_layout()

plt.show()
```

[Code Cell]

```
#@title Gebäudedate aus PNG spichärä { run: "auto", display-mode: "form" }

import matplotlib.pyplot as plt
import pandas as pd

# Assuming `data` is your DataFrame
csv_file_path = "buildings_data.csv"
data = pd.read_csv(csv_file_path)

# Create a figure and axis to 'plot' the DataFrame
fig, ax = plt.subplots(figsize=(12, len(data) * 0.25))  # Adjust size as needed
ax.axis('tight')
ax.axis('off')
the_table   =   ax.table(cellText=data.values,   colLabels=data.columns,   loc='center',
cellLoc='center')

# Improve table aesthetics
the_table.auto_set_font_size(False)
the_table.set_fontsize(10)  # Adjust font size
the_table.auto_set_column_width(col=list(range(len(data.columns))))    #  Adjust  to  fit
column width

plt.savefig("buildings_data_df.png")
```

[Code Cell]

```
#@title Speziaufäu (nur ändärä wennd weisch wasd machsch) { run: "auto", display-mode:
"form" }

def update_parzelle_interactively(df):
    # Check if 'Parzelle' column exists, add it if not
    if 'Parzelle' not in df.columns:
        df['Parzelle'] = None  # Initialize with None to indicate that it's not set

    # Iterate over the DataFrame to prompt for 'Parzelle' number for each building
    for index, row in df.iterrows():
        # Prompt for 'Parzelle' number
        print(f"Current data for Building ID {row['Building ID']}:")
        print(row)
           new_parzelle = input(f"Enter Parzelle number for Building ID {row['Building
ID']}: ")

        # Check if the input is a number
        try:
            new_parzelle = int(new_parzelle)  # Convert to integer
            df.at[index, 'Parzelle'] = new_parzelle  # Update the DataFrame
        except ValueError:
            print("Invalid input, not a number. Skipping to the next building.")

    return df

# To redo the calculation of floor area for a new floor height
def update_floors(df, floor_height=3):
    # Iterate over the DataFrame rows
    for index, row in df.iterrows():
        # Check if the "Floors" value is NaN
        if pd.isna(row['Floors']):
            # Calculate the floors based on the building height
            floors = max(1, row['Height (m)'] // floor_height)
            # Update the DataFrame with the calculated floors value
            df.at[index, 'Floors'] = floors
    return df

def update_data(df, floor_height=3):
    # Iterate over the DataFrame rows
    for index, row in df.iterrows():
        # Recalculate floors based on the building height with the new floor height
        floors = max(1, row['Height (m)'] // floor_height)
        # Recalculate Geschossfläche
        geschossfläche = row['Footprint Area (m²)'] * floors
        # Update the DataFrame with the recalculated floors and Geschossfläche values
        df.at[index, 'Floors'] = floors
        df.at[index, 'Geschossfläche (m²)'] = geschossfläche
    return df

def scale_data(df, floor_height=2.8):
    # Iterate over the DataFrame rows
    for index, row in df.iterrows():
        # Scale the height by a factor of 5
        scaled_height = row['Height (m)'] * 5
        # Recalculate floors based on the scaled height
        floors = max(1, scaled_height // floor_height)
        # Recalculate Geschossfläche using the new floors calculation
        geschossfläche = row['Footprint Area (m²)'] * floors
        # Update the DataFrame with the new height, floors, and Geschossfläche values
        df.at[index, 'Height (m)'] = scaled_height
        df.at[index, 'Floors'] = floors
        df.at[index, 'Geschossfläche (m²)'] = geschossfläche
    return df

'''
buildings_data = update_parzelle_interactively(buildings_data)
save_data_to_csv(buildings_data, csv_file_path)

# Update the DataFrame with missing floor counts
buildings_data = update_data(buildings_data)
buildings_data = scale_data(buildings_data)

# Save the updated DataFrame to CSV
save_data_to_csv(buildings_data_scaled, csv_file_path)

# Display the updated DataFrame
print("\nUpdated Buildings Data with Floor Counts:")
print(buildings_data)
'''
```