Exercise 1

Exercise 1.1

```
for i in range(1,6):
    print("x"*i)
for y in range(1,6):
    print("x"*(5-y))
```

```
x
xx
xxx
xxxx
xxxxx
xxxx
xxx
xx
x
```

write a code that prints the following pattern. Try to use as few loops as possible.

```
X
X X
X X X
X X X X
X X X X X
X X X X
X X X
X X
X
```

Avoid using trivial solutions like:

print('X')

print('X X')

print('X X X')

.....

print('X X X')

print('X X')

print('X')

Exercise 1.2

Write a script that will sum all numbers in the following string. You can split each number into single digits, so for example, you can consider 45 to be 4 and 5

```
sum = 0
input_str = "n45as29@#8ss6"
for i in input_str:
  if not i.isdigit():
    continue
  sum += int(i)
print( sum )
```

```
34
```

Exercise 1.3

Write a script that will convert an arbitrary integer to a binary number (the number will be represented as a string with only *0* and *1*). Avoid using the *bin()* function or any other pythons default functions

```
def convert_to_binary(number: int) -> str:
  if number ==0:
    return "0"
  quotient = number // 2
  reste = number % 2
  binary =str(reste)
  while quotient != 0:
    reste = quotient % 2
    binary += str(reste)
    quotient = quotient // 2

  print(binary)
  return binary[::-1]
convert_to_binary(255)
```

```
11111111
'11111111'
```

Exercise 1.4 - The Fibonacci Sequence:

The Fibonacci Sequence is a series of numbers. The following number is found by adding up the two numbers before it. The first two numbers are 0 and 1. For example, 0, 1, 1, 2, 3, 5, 8, 13. The following number in this series above is 8 + 13 = 21

Your task is to implement a function *fibonacci* that takes an integer as an input and returns a list that contains all Fibonacci numbers with values lower than the input integer

Example:

print(fibonaci(10))

[0, 1, 1, 2, 3, 5, 8]

```
def fibonacci(upper_threshold: int) -> list:
    result = []
    a, b = 0, 1

    while a < upper_threshold:
        result.append(a)
        a, b = b, a + b

    return result
```

```
fibonacci(22)
```

```
[0, 1, 1, 2, 3, 5, 8, 13, 21]
```

Advanced

Try to implement the function using multiple approaches: *Iterative approach*, *Recusrion*, *Memoization*

Exercise 1.5. - Rock, Paper, Scissors game:

basic

Rock, Paper, Scissors is a well-known and straightforward game. If you do not know the rules, google them. We will write a code for the Rock, Paper, Scissors game where the user plays against a random computer. The code can be written into one function *rock_paper_scissors*.

Notes:

- This implementation uses the **random** library to enable the computer to make a random choice.
- Keyword **Input** may be helpful. Check it
- After the function is run, the program will ask you about your movement, which you type in
- The game is case-insensitive for user input (e.g., "rock", "Rock", and "ROCK" are all valid).
- This script plays one round of the game.
- The script prints the result ('You lose', 'You win', 'It is a tie')

```python
import random
def rock_paper_scissors() -> None:
  user_input = input("Enter your choice (rock, paper, scissors): ").lower()
  machine_choice = random.choice(["rock", "paper", "scissors"])
  print(f"Computer chose: {machine_choice}")
  if user_input not in ["rock", "paper", "scissors"]:
    print("Invalid input. Please enter rock, paper, or scissors.")
    return
  if user_input == "rock":
    if machine_choice == "scissors":
      print("You win!")
    elif machine_choice == "paper":
      print("You lose!")
    elif machine_choice == "rock":
      print("It's a tie!")
  if user_input == "scissors":
    if machine_choice == "paper":
      print("You win!")
    elif machine_choice == "rock":
      print("You lose!")
    elif machine_choice == "scissors":
      print("It's a tie!")
  if user_input == "paper":
    if machine_choice == "rock":
      print("You win!")
    elif machine_choice == "scissors":
      print("You lose!")
    elif machine_choice == "paper":
      print("It's a tie!")


rock_paper_scissors()
```

```
Enter your choice (rock, paper, scissors): paper
Computer chose: scissors
You lose!
```

Advanced

Extend the Rock, Paper, Scissors game to be able to play *n* rounds.

Notes:

- The scrips will contain two counters - user score and computer score
- After each run, the script prints the current scores and rounds.
- When the number of rounds is reached, the script prints the result ('You lose', 'You win', 'It is a tie')

```
import random
rounds=0
user_score=0
computer_score=0
victory_status= ""
i = 0
def rock_paper_scissors() -> None:
  global user_score, computer_score, i, victory_status
  rounds_to_play = int(input("Enter the number of rounds: "))
  while i < rounds_to_play:
    user_input = input("Enter your choice (rock, paper, scissors): ").lower(
    machine_choice = random.choice(["rock", "paper", "scissors"])
    print(f"Computer chose: {machine_choice}")
    if user_input not in ["rock", "paper", "scissors"]:
      print("Invalid input. Please enter rock, paper, or scissors.")
      continue # Continue to the next iteration instead of returning

    if user_input == "rock":
      if machine_choice == "scissors":
        victory_status="You win!"
      elif machine_choice == "paper":
        victory_status="You lose!"
      elif machine_choice == "rock":
        victory_status="It's a tie!"
    elif user_input == "scissors":
      if machine_choice == "paper":
        victory_status="You win!"
      elif machine_choice == "rock":
        victory_status="You lose!"
      elif machine_choice == "scissors":
        victory_status="It's a tie!"
    elif user_input == "paper": # Use elif
      if machine_choice == "rock":
        victory_status="You win!"
      elif machine_choice == "scissors":
        victory_status="You lose!"
      elif machine_choice == "paper":
        victory_status="It's a tie!"

    print(victory_status)

    if victory_status=="You win!":
      user_score+=1
    elif victory_status=="You lose!":
      computer_score+=1
    i += 1 # Increment the round counter
    print(f"Current Scores: Player {user_score} - Computer {computer_score}

  print("\n--- Final Results ---")
  if user_score > computer_score:
    print("You win the game!")
  elif computer_score > user_score:
    print("The computer wins the game!")
  else:
    print("The game is a tie!")


rock_paper_scissors()
```

```
Enter the number of rounds: 3
Enter your choice (rock, paper, scissors): paper
Computer chose: paper
It's a tie!
Current Scores: Player 0 - Computer 0 (Round 1/3)
Enter your choice (rock, paper, scissors): paper
Computer chose: rock
You win!
Current Scores: Player 1 - Computer 0 (Round 2/3)
Enter your choice (rock, paper, scissors): paper
Computer chose: paper
It's a tie!
Current Scores: Player 1 - Computer 0 (Round 3/3)

--- Final Results ---
You win the game!
```

Exercise 2

The purpose of this excercise is to practise working with NumPy library

Exercise 2.1

In this exercise, you will work with NumPy arrays and learn how to efficiently modify and process them using both loops and vectorized operations. Your task is to:

1. Implement the function *create_array_nxn* that generates an n×n NumPy array filled with numbers from n^2 - 1 down to 0
2. Using a loop-based approach (for-loop, while-loop), implement the function *apply_threshold_loop* that takes any NumPy array and replaces all numbers below a user-defined threshold with 0.
3. Implement function *apply_threshold_vectorized* that performs the same operation as *apply_threshold_loop*, but without loops, using NumPy vectorized

operations

4. Implement function *compare_performance* that compares the execution time of functions *apply_threshold_vectorized* and *apply_threshold_loop* with **time** library and print the results

Additional Notes:

- Ensure n is a positive integer (n > 0).
- Use the time library to measure execution times for performance comparison.
- The function *compare_performance(n, threshold)* should:
    - Generate an n×n array.
    - Apply both loop-based and vectorized thresholding.
    - Print execution times.

```
import numpy as np
import time
```

```
def create_array_nxn(n: int) -> np.ndarray:
  if n <= 0:
    raise ValueError("n must be a positive integer")

  # Create a 1D array with numbers from n*n - 1 down to 0
  flat_array = np.arange(n**2 - 1, -1, -1)

  # Reshape the 1D array into an n x n 2D array
  my_array = flat_array.reshape((n, n))

  return my_array

create_array_nxn(5)
```

```
array([[24, 23, 22, 21, 20],
       [19, 18, 17, 16, 15],
       [14, 13, 12, 11, 10],
       [ 9,  8,  7,  6,  5],
       [ 4,  3,  2,  1,  0]])
```

```
def apply_threshold_loop(arr: np.ndarray, threshold: int) -> np.ndarray:
    # Create a copy to avoid modifying the original array
    modified_arr = arr.copy()

    # Iterate through the array using nested loops
    for i in range(modified_arr.shape[0]):
        for j in range(modified_arr.shape[1]):
            if modified_arr[i, j] < threshold:
                modified_arr[i, j] = 0
    return modified_arr
```

```
def apply_threshold_vectorized(arr: np.ndarray, threshold: int) -> np.ndarra
    modified_arr = arr.copy()
    modified_arr[modified_arr < threshold] = 0
    return modified_arr
```

```
def compare_performance(n: int, threshold: int) -> None:
    # Generate the n x n array
    arr = create_array_nxn(n)

    # Measure performance of loop-based approach
    start_time_loop = time.time()
    apply_threshold_loop(arr, threshold)
    end_time_loop = time.time()
    loop_time = end_time_loop - start_time_loop

    # Measure performance of vectorized approach
    start_time_vectorized = time.time()
    apply_threshold_vectorized(arr, threshold)
    end_time_vectorized = time.time()
    vectorized_time = end_time_vectorized - start_time_vectorized

    # Print results
    print(f"Performance comparison for n={n}, threshold={threshold}:")
    print(f"  Loop-based approach: {loop_time:.6f} seconds")
    print(f"  Vectorized approach: {vectorized_time:.6f} seconds")

    # Optionally, print the ratio for clearer comparison
    if vectorized_time > 0:
        print(f"  Vectorized is {loop_time / vectorized_time:.2f}x faster th
    else:
        print("  Vectorized time was zero, cannot calculate speedup ratio.")
```

Exercise 2.2 - Digi display

basic

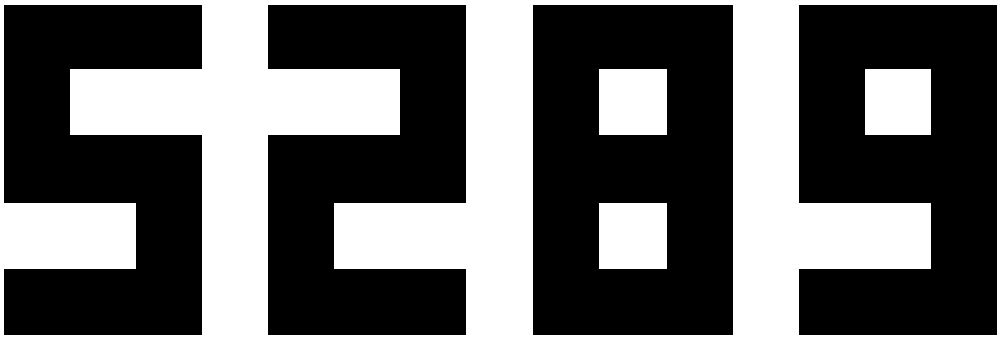
The task will be to create a simulation of a Digi display that will be able to display an arbitrary integer

*hint: to show the image, use the library matplotlib.pyplot. Also, you may find function *np.concatenate* helpful

```
numbs = {
      "1": np.array([[0, 1, 1], [1, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1]])
      "2": np.array([[1, 1, 1], [0, 0, 1], [1, 1, 1], [1, 0, 0], [1, 1, 1]])
      "3": np.array([[1, 1, 1], [0, 0, 1], [1, 1, 1], [0, 0, 1], [1, 1, 1]])
      "4": np.array([[1, 0, 1], [1, 0, 1], [1, 1, 1], [0, 0, 1], [0, 0, 1]])
      "5": np.array([[1, 1, 1], [1, 0, 0], [1, 1, 1], [0, 0, 1], [1, 1, 1]])
      "6": np.array([[1, 1, 1], [1, 0, 0], [1, 1, 1], [1, 0, 1], [1, 1, 1]])
      "7": np.array([[1, 1, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1]])
      "8": np.array([[1, 1, 1], [1, 0, 1], [1, 1, 1], [1, 0, 1], [1, 1, 1]])
```

```
        "9": np.array([[1, 1, 1], [1, 0, 1], [1, 1, 1], [0, 0, 1], [1, 1, 1]])
        "0": np.array([[1, 1, 1], [1, 0, 1], [1, 0, 1], [1, 0, 1], [1, 1, 1]])
    }
```

```
import matplotlib.pyplot as plt
def show_in_digi(input_integer: int) -> None:
    s = str(input_integer)
    digit_images = []
    for digit_char in s:
        if digit_char in numbs:
            digit_images.append(numbs[digit_char])
        else:
            # Handle cases where character is not a digit (e.g., negative si
            # For now, let's assume valid digits. Could add a blank or error
            pass # Or raise an error, or use a placeholder

    if not digit_images:
        print("No valid digits to display.")
        return

    # Concatenate digit images horizontally
    # Add a small separator (e.g., a column of zeros) between digits for bet
    combined_image = digit_images[0]
    for i in range(1, len(digit_images)):
        # Add a blank column as separator
        separator = np.zeros((digit_images[0].shape[0], 1))
        combined_image = np.concatenate((combined_image, separator, digit_im

    plt.imshow(combined_image, cmap='binary') # Use 'binary' colormap for bl
    plt.axis('off') # Hide axes
    plt.show()

show_in_digi(5289)
```

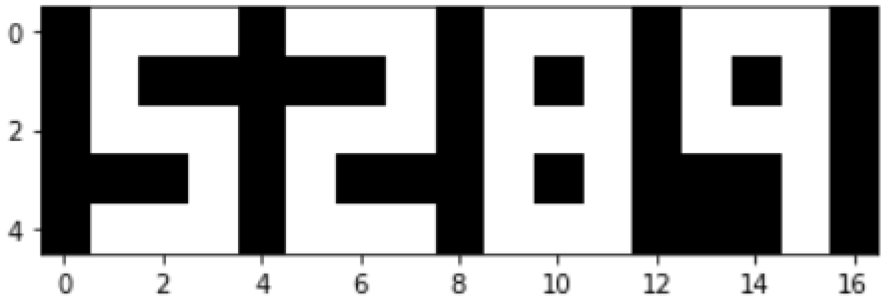


```
show_in_digi(5289)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipython-input-2458880815.py in <cell line: 0>()
----> 1 show_in_digi(5289)

/tmp/ipython-input-2794560182.py in show_in_digi(input_integer)
      4     digit_images = []
      5     for digit_char in s:
----> 6         if digit_char in numbs:
      7             digit_images.append(numbs[digit_char])
      8         else:

NameError: name 'numbs' is not defined
```

Étapes suivantes : Expliquer l'erreur

show_in_digi(5289) will show:



hint2: you may find the following dict usefull

```
numbs = {
      "1": np.array([[0, 1, 1], [1, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1]]),
      "2": np.array([[1, 1, 1], [0, 0, 1], [1, 1, 1], [1, 0, 0], [1, 1, 1]]),
      "3": np.array([[1, 1, 1], [0, 0, 1], [1, 1, 1], [0, 0, 1], [1, 1, 1]]),
      "4": np.array([[1, 0, 1], [1, 0, 1], [1, 1, 1], [0, 0, 1], [0, 0, 1]]),
      "5": np.array([[1, 1, 1], [1, 0, 0], [1, 1, 1], [0, 0, 1], [1, 1, 1]]),
      "6": np.array([[1, 1, 1], [1, 0, 0], [1, 1, 1], [1, 0, 1], [1, 1, 1]]),
      "7": np.array([[1, 1, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1]]),
      "8": np.array([[1, 1, 1], [1, 0, 1], [1, 1, 1], [1, 0, 1], [1, 1, 1]]),
      "9": np.array([[1, 1, 1], [1, 0, 1], [1, 1, 1], [0, 0, 1], [1, 1, 1]]),
      "0": np.array([[1, 1, 1], [1, 0, 1], [1, 0, 1], [1, 0, 1], [1, 1, 1]])
  }
```

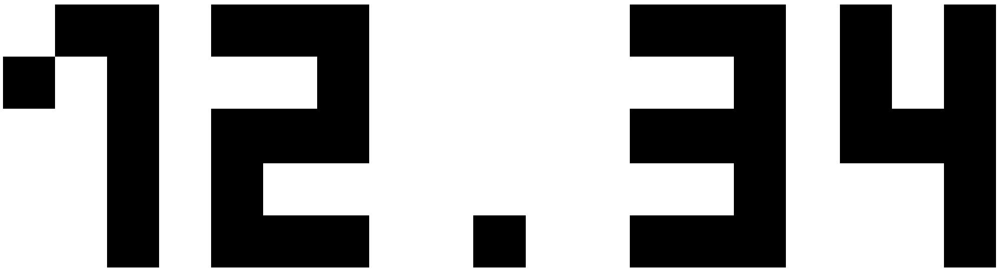
advanced

Extend your *show_in_digi* function to be able to display an arbitary float number and negative numbers

```
import numpy as np
import matplotlib.pyplot as plt
from typing import Union

# Extended numbs dictionary to include negative sign and decimal point
numbs = {
      "1": np.array([[0, 1, 1], [1, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1]])
      "2": np.array([[1, 1, 1], [0, 0, 1], [1, 1, 1], [1, 0, 0], [1, 1, 1]])
      "3": np.array([[1, 1, 1], [0, 0, 1], [1, 1, 1], [0, 0, 1], [1, 1, 1]])
      "4": np.array([[1, 0, 1], [1, 0, 1], [1, 1, 1], [0, 0, 1], [0, 0, 1]])
      "5": np.array([[1, 1, 1], [1, 0, 0], [1, 1, 1], [0, 0, 1], [1, 1, 1]])
      "6": np.array([[1, 1, 1], [1, 0, 0], [1, 1, 1], [1, 0, 1], [1, 1, 1]])
      "7": np.array([[1, 1, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1], [0, 0, 1]])
      "8": np.array([[1, 1, 1], [1, 0, 1], [1, 1, 1], [1, 0, 1], [1, 1, 1]])
      "9": np.array([[1, 1, 1], [1, 0, 1], [1, 1, 1], [0, 0, 1], [1, 1, 1]])
      "0": np.array([[1, 1, 1], [1, 0, 1], [1, 0, 1], [1, 0, 1], [1, 1, 1]])
      "-": np.array([[0,0,0], [0,0,0], [1,1,1], [0,0,0], [0,0,0]]), # Negati
      ".": np.array([[0,0,0], [0,0,0], [0,0,0], [0,0,0], [0,1,0]])  # Decima
  }

def show_in_digi(input_number: Union[int, float]) -> None:
    s = str(input_number)
    digit_images = []
    for digit_char in s:
        if digit_char in numbs:
            digit_images.append(numbs[digit_char])
        else:
            print(f"Warning: Character '{digit_char}' not supported and will
            continue

    if not digit_images:
        print("No valid digits or symbols to display.")
        return

    # Concatenate digit images horizontally
    # Add a small separator (e.g., a column of zeros) between digits for bet
    # The separator width can be adjusted. Here, 1 column of zeros.
    separator_width = 1
    combined_image = digit_images[0]
    for i in range(1, len(digit_images)):
        separator = np.zeros((digit_images[0].shape[0], separator_width))
        combined_image = np.concatenate((combined_image, separator, digit_im

    plt.imshow(combined_image, cmap='binary') # Use 'binary' colormap for bl
    plt.axis('off') # Hide axes
    plt.show()

# Test with float and negative numbers
print("Displaying 12.34:")
show_in_digi(12.34)

print("\nDisplaying -567:")
show_in_digi(-567)

print("\nDisplaying -0.89:")
```

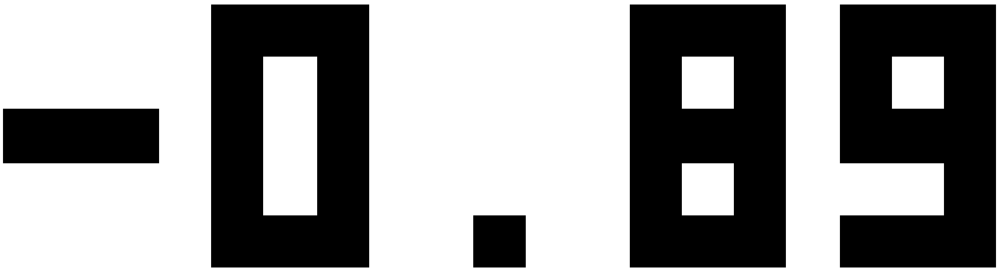
```
show_in_digi(-0.89)

print("\nDisplaying 0:")
show_in_digi(0)
```

```
Displaying 12.34:
```

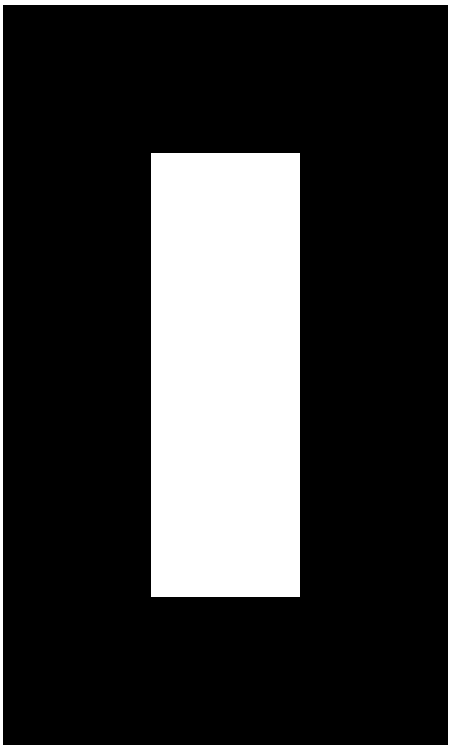


```
Displaying -567:
```

```
Displaying -0.89:
```



```
Displaying 0:
```



Exercise 3 - Playing with California Housing Dataset

The purpose of this exercise is to learn basic operations of **pandas** library. Use *california_housing_dataset* you can find in google colab to do following operations:

- check what *dataset.describe()* does
- display all rows where *total_bedrooms* column is bigger than 310
- drop also first and last row
- save a mean of values in *households* columns to variable, using *matplotlib* display number of households in the graph (x-axis would be ID, y-axis - numbers of households) as dots with the same color, display also mean as a single line in the graph with different color as dots is
- check if any of the columns contain NaN values, if yes, replace the NaN values with the arithmetic mean of the whole dataset
- display *lat* and *long* into plot, *lat* would be on *x* axis and *long* on *y* axis
- choose two arbitrary columns and normalize all values inside using *min-max normalization*:
- Create a Correlation Matrix from all columns

```
Commencez à coder ou à générer avec l'IA.
```

Task

Perform the following operations on the `california_housing_train.csv` dataset: load it into a pandas DataFrame, describe the dataset to get statistical summaries, filter the DataFrame to include only rows where 'total_bedrooms' is greater than 310, and then drop the first and last rows from this filtered DataFrame. Next, calculate the mean of the 'households' column and create a scatter plot showing 'households' against the DataFrame index, adding a horizontal line for the calculated mean. Check for any NaN values in the DataFrame; if found, replace them with the mean of all numeric values in the DataFrame. Then, create a scatter plot of 'latitude' (x-axis) against 'longitude' (y-axis). Choose two arbitrary numerical columns (e.g., 'median_income' and 'housing_median_age') and apply min-max normalization to them. Finally, generate and display a correlation matrix for all numerical columns in the DataFrame and summarize the basic pandas operations and key observations from the analysis.

Load Dataset

Subtask:

Load the 'california_housing_train.csv' dataset into a pandas DataFrame.

Reasoning: First, I'll import the pandas library, which is essential for data manipulation in Python, especially for handling DataFrames. Then, I'll use `pd.read_csv` to load the specified CSV file into a DataFrame named `df` and display its head to confirm the loading process.

```
import pandas as pd

df = pd.read_csv('/content/sample_data/california_housing_train.csv')
print("Dataset loaded successfully. Displaying the first 5 rows:")
print(df.head())
```

```
Dataset loaded successfully. Displaying the first 5 rows:
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -114.31     34.19                15.0       5612.0          1283.0   
1    -114.47     34.40                19.0       7650.0          1901.0   
2    -114.56     33.69                17.0        720.0           174.0   
3    -114.57     33.64                14.0       1501.0           337.0   
4    -114.57     33.57                20.0       1454.0           326.0   

   population  households  median_income  median_house_value  
0      1015.0       472.0         1.4936             66900.0  
1      1129.0       463.0         1.8200             80100.0  
2       333.0       117.0         1.6509             85700.0  
3       515.0       226.0         3.1917             73400.0  
4       624.0       262.0         1.9250             65500.0  
```

Describe Dataset

Subtask:

Utilize the `.describe()` method on the loaded DataFrame to get a statistical summary of the numerical columns, including count, mean, standard deviation, min, max, and quartile values.

Reasoning: To get a statistical summary of the numerical columns in the DataFrame, I will use the `.describe()` method on the `df` DataFrame and print the result.

```
print("Descriptive statistics for the dataset:")
print(df.describe())
```

```
Descriptive statistics for the dataset:
          longitude      latitude  housing_median_age   total_rooms  \
count  17000.000000  17000.000000        17000.000000  17000.000000   
mean    -119.562108     35.625225           28.589353   2643.664412   
std        2.005166      2.137340           12.586937   2179.947071   
min     -124.350000     32.540000            1.000000      2.000000   
25%     -121.790000     33.930000           18.000000   1462.000000   
50%     -118.490000     34.250000           29.000000   2127.000000   
75%     -118.000000     37.720000           37.000000   3151.250000   
max     -114.310000     41.950000           52.000000  37937.000000   

       total_bedrooms    population    households  median_income  \
count    17000.000000  17000.000000  17000.000000   17000.000000   
mean       539.410824   1429.573941    501.221941       3.883578   
std        421.499452   1147.852959    384.520841       1.908157   
min          1.000000      3.000000      1.000000       0.499900   
25%        297.000000    790.000000    282.000000       2.566375   
50%        434.000000   1167.000000    409.000000       3.544600   
75%        648.250000   1721.000000    605.250000       4.767000   
max       6445.000000  35682.000000   6082.000000      15.000100   

       median_house_value  
count        17000.000000  
mean        207300.912353  
std         115983.764387  
min          14999.000000  
25%         119400.000000  
50%         180400.000000  
75%         265000.000000  
max         500001.000000  
```

Reasoning: The next step is to filter the DataFrame `df` to include only rows where the 'total_bedrooms' column is greater than 310, as specified in the instructions. I will then display the head of the new filtered DataFrame to confirm the operation.

```
df_filtered = df[df['total_bedrooms'] > 310]
print("DataFrame filtered for 'total_bedrooms' > 310. Displaying the first 5
print(df_filtered.head())
```

```
DataFrame filtered for 'total_bedrooms' > 310. Displaying the first 5 rows:
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -114.31     34.19                15.0       5612.0          1283.0   
1    -114.47     34.40                19.0       7650.0          1901.0   
3    -114.57     33.64                14.0       1501.0           337.0   
4    -114.57     33.57                20.0       1454.0           326.0   
6    -114.58     33.61                25.0       2907.0           680.0   

   population  households  median_income  median_house_value
```

```
0      1015.0       472.0          1.4936              66900.0
1      1129.0       463.0          1.8200              80100.0
3       515.0       226.0          3.1917              73400.0
4       624.0       262.0          1.9250              65500.0
6      1841.0       633.0          2.6768              82400.0
```

Reasoning: To drop the first and last rows from the `df_filtered` DataFrame, I will use `iloc` with slicing `[1:-1]` to select all rows except the first and the last. Then, I will display the head and tail of the modified DataFrame to confirm the operation.

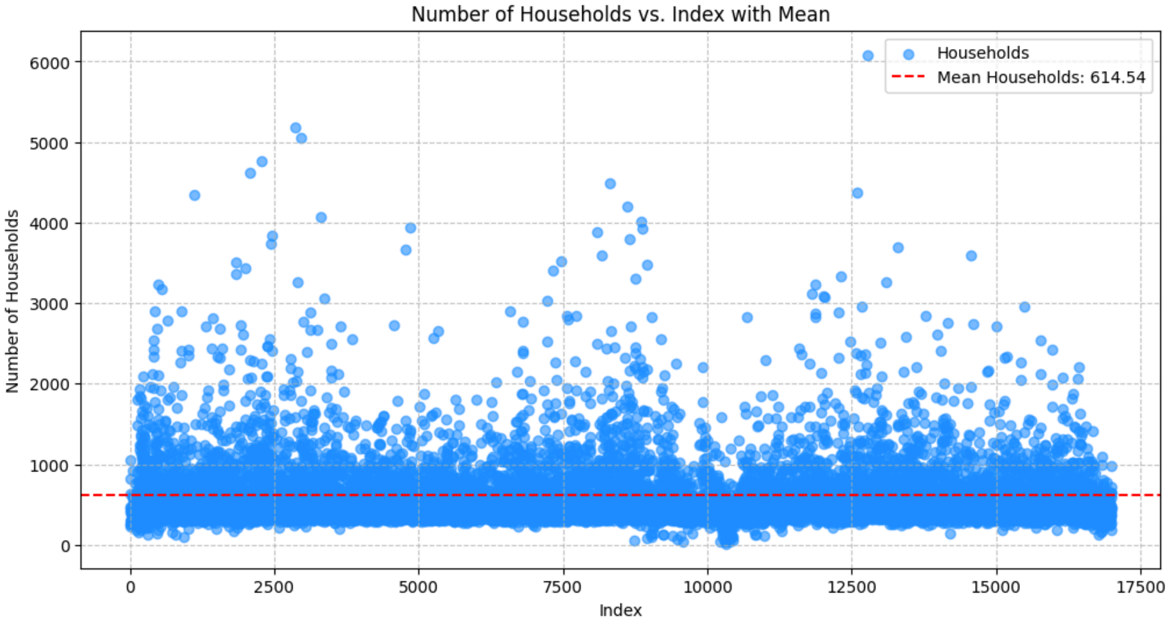
```
df_modified = df_filtered.iloc[1:-1]
print("DataFrame after dropping the first and last rows. Displaying the firs
print(df_modified.head())
print("\nDisplaying the last 5 rows:")
print(df_modified.tail())
```

```
DataFrame after dropping the first and last rows. Displaying the first 5 rows
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
1    -114.47     34.40                19.0       7650.0          1901.0
3    -114.57     33.64                14.0       1501.0           337.0
4    -114.57     33.57                20.0       1454.0           326.0
6    -114.58     33.61                25.0       2907.0           680.0
8    -114.59     33.61                34.0       4789.0          1175.0

   population  households  median_income  median_house_value
1      1129.0       463.0         1.8200             80100.0
3       515.0       226.0         3.1917             73400.0
4       624.0       262.0         1.9250             65500.0
6      1841.0       633.0         2.6768             82400.0
8      3134.0      1056.0         2.1782             58400.0

Displaying the last 5 rows:
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms
16993    -124.23     40.54                52.0       2694.0           453.0
16994    -124.25     40.28                32.0       1430.0           419.0
16995    -124.26     40.58                52.0       2217.0           394.0
16996    -124.27     40.69                36.0       2349.0           528.0
16997    -124.30     41.84                17.0       2677.0           531.0

       population  households  median_income  median_house_value
16993      1152.0       435.0         3.0806            106700.0
16994       434.0       187.0         1.9417             76100.0
16995       907.0       369.0         2.3571            111400.0
16996      1194.0       465.0         2.5179             79000.0
16997      1244.0       456.0         3.0313            103600.0
```

Reasoning: Now, I will calculate the mean of the 'households' column from the `df_modified` DataFrame. Then, I will create a scatter plot of 'households' against the DataFrame index, adding a horizontal line to represent the calculated mean, and ensure the plot has appropriate labels and a title for better visualization.

```
import matplotlib.pyplot as plt

mean_households = df_modified['households'].mean()
print(f"Mean of 'households' column: {mean_households:.2f}")

plt.figure(figsize=(12, 6))
plt.scatter(df_modified.index, df_modified['households'], alpha=0.6, color='
plt.axhline(y=mean_households, color='r', linestyle='--', label=f'Mean House
plt.xlabel('Index')
plt.ylabel('Number of Households')
plt.title('Number of Households vs. Index with Mean')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

```
Mean of 'households' column: 614.54
```



Reasoning: Now I will check for any NaN values in the `df_modified` DataFrame. If NaN values are found, I will replace them with the arithmetic mean of all numerical values in the DataFrame to handle missing data.

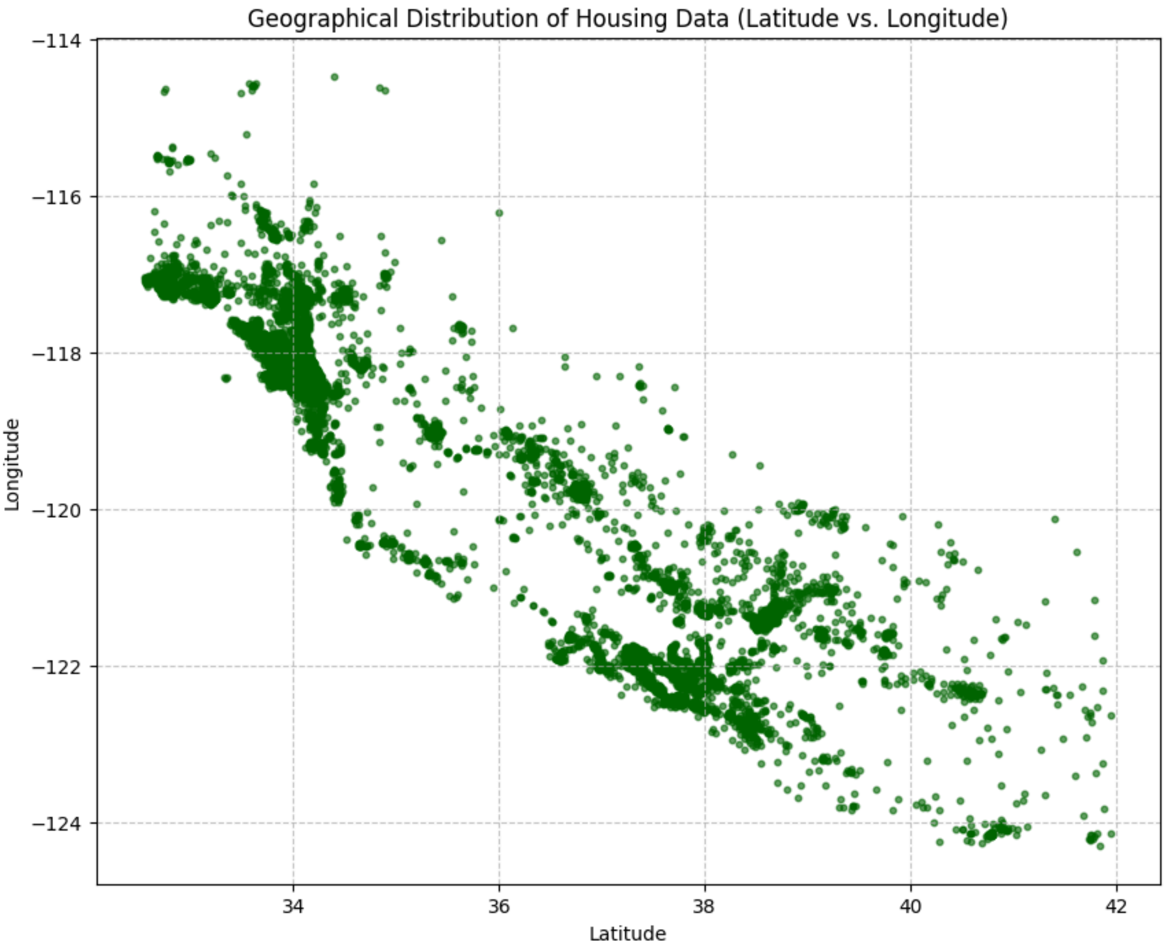
```
print("Checking for NaN values:")
print(df_modified.isnull().sum())
```

```
if df_modified.isnull().sum().any():
    # Calculate the mean of all numerical values in the DataFrame
    # First, select only numerical columns to calculate the mean
    numerical_cols = df_modified.select_dtypes(include=np.number).columns
    overall_mean = df_modified[numerical_cols].stack().mean()

    print(f"\nNaN values found. Replacing with overall mean: {overall_mean:.
    df_modified.fillna(overall_mean, inplace=True)
    print("NaN values after replacement:")
    print(df_modified.isnull().sum())
else:
    print("\nNo NaN values found in the DataFrame.")
```

```
Checking for NaN values:
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
dtype: int64

No NaN values found in the DataFrame.
```

Reasoning: Now, I will create a scatter plot to visualize the spatial distribution of the housing data, with 'latitude' on the x-axis and 'longitude' on the y-axis, as requested.

```
plt.figure(figsize=(10, 8))
plt.scatter(df_modified['latitude'], df_modified['longitude'], alpha=0.6, cc
plt.xlabel('Latitude')
plt.ylabel('Longitude')
plt.title('Geographical Distribution of Housing Data (Latitude vs. Longitude
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```



Reasoning: Now, I will choose two arbitrary numerical columns from the `df_modified` DataFrame, 'median_income' and 'housing_median_age', and apply min-max normalization to them to scale their values between 0 and 1. I will then display the head of the DataFrame to show the normalized columns.

```
selected_cols = ['median_income', 'housing_median_age']

for col in selected_cols:
    min_val = df_modified[col].min()
    max_val = df_modified[col].max()
    df_modified[f'{col}_normalized'] = (df_modified[col] - min_val) / (max_v

print("DataFrame after Min-Max normalization for 'median_income' and 'housin
print(df_modified[['median_income', 'median_income_normalized', 'housing_med
```

```
DataFrame after Min-Max normalization for 'median_income' and 'housing_median
   median_income  median_income_normalized  housing_median_age  \
1         1.8200                  0.079769                19.0
3         3.1917                  0.175541                14.0
4         1.9250                  0.087100                20.0
6         2.6768                  0.139591                25.0
8         2.1782                  0.104778                34.0

   housing_median_age_normalized
1                       0.352941
3                       0.254902
4                       0.372549
6                       0.470588
8                       0.647059
/tmp/ipython-input-2954841273.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df_modified[f'{col}_normalized'] = (df_modified[col] - min_val) / (max_val
/tmp/ipython-input-2954841273.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/s
  df_modified[f'{col}_normalized'] = (df_modified[col] - min_val) / (max_val
```

Reasoning: The previous code produced a `SettingWithCopyWarning`. This warning indicates that operations are being performed on a slice of a DataFrame, which might not modify the original DataFrame as intended. To resolve this, I will explicitly create a deep copy of the DataFrame `df_filtered` when creating `df_modified` to ensure independent modification. Then, I will perform min-max normalization on the selected columns and display the results.

```
df_modified = df_filtered.iloc[1:-1].copy() # Ensure df_modified is a deep c

selected_cols = ['median_income', 'housing_median_age']

for col in selected_cols:
    min_val = df_modified[col].min()
    max_val = df_modified[col].max()
    df_modified[f'{col}_normalized'] = (df_modified[col] - min_val) / (max_v
print("DataFrame after Min-Max normalization for 'median_income' and 'housin
print(df_modified[['median_income', 'median_income_normalized', 'housing_med
```

```
DataFrame after Min-Max normalization for 'median_income' and 'housing_median
   median_income  median_income_normalized  housing_median_age  \
1         1.8200                  0.079769                19.0
3         3.1917                  0.175541                14.0
4         1.9250                  0.087100                20.0
6         2.6768                  0.139591                25.0
8         2.1782                  0.104778                34.0

   housing_median_age_normalized
1                       0.352941
3                       0.254902
4                       0.372549
6                       0.470588
8                       0.647059
```

Reasoning: Now, I will generate a correlation matrix for all numerical columns in the `df_modified` DataFrame to understand the relationships between different features. I will then display this matrix.

```
print("Correlation Matrix for all numerical columns:")
correlation_matrix = df_modified.corr()
print(correlation_matrix)
```

```
Correlation Matrix for all numerical columns:
                               longitude  latitude  housing_median_age  \
longitude                       1.000000 -0.926237           -0.121227
latitude                       -0.926237  1.000000            0.024286
housing_median_age             -0.121227  0.024286            1.000000
total_rooms                     0.041182 -0.023179           -0.375131
total_bedrooms                  0.070216 -0.060194           -0.324652
population                      0.096686 -0.100238           -0.296564
households                      0.052823 -0.063723           -0.304374
median_income                  -0.017161 -0.075634           -0.140311
median_house_value             -0.053489 -0.139220            0.132600
median_income_normalized       -0.017161 -0.075634           -0.140311
housing_median_age_normalized  -0.121227  0.024286            1.000000

                               total_rooms  total_bedrooms  population  \
longitude                         0.041182        0.070216    0.096686
latitude                         -0.023179       -0.060194   -0.100238
housing_median_age               -0.375131       -0.324652   -0.296564
total_rooms                       1.000000        0.911944    0.836176
total_bedrooms                    0.911944        1.000000    0.858352
population                        0.836176        0.858352    1.000000
households                        0.899982        0.975801    0.893430
median_income                     0.260194        0.006390    0.025688
median_house_value                0.132169        0.039259   -0.046994
median_income_normalized          0.260194        0.006390    0.025688
housing_median_age_normalized    -0.375131       -0.324652   -0.296564

                               households  median_income  median_house_valu
longitude                        0.052823      -0.017161           -0.05348
```

```
latitude                        -0.063723     -0.075634           -0.13922
housing_median_age              -0.304374     -0.140311            0.13260
total_rooms                      0.899982      0.260194            0.13216
total_bedrooms                   0.975801      0.006390            0.03925
population                       0.893430      0.025688           -0.04699
households                       1.000000      0.031419            0.05707
median_income                    0.031419      1.000000            0.67816
median_house_value               0.057077      0.678163            1.00000
median_income_normalized         0.031419      1.000000            0.67816
housing_median_age_normalized   -0.304374     -0.140311            0.13260

                               median_income_normalized  \
longitude                                     -0.017161
latitude                                      -0.075634
housing_median_age                            -0.140311
total_rooms                                    0.260194
total_bedrooms                                 0.006390
population                                     0.025688
households                                     0.031419
median_income                                  1.000000
median_house_value                             0.678163
median_income_normalized                       1.000000
housing_median_age_normalized                 -0.140311

                               housing_median_age_normalized
longitude                                          -0.121227
latitude                                            0.024286
housing_median_age                                  1.000000
```

Summary of Operations and Observations

In this exercise, we performed several key data analysis operations using the pandas library on the `california_housing_train.csv` dataset. The steps included:

1. **Loading the Dataset**: We successfully loaded the `california_housing_train.csv` into a pandas DataFrame named `df`.
2. **Descriptive Statistics**: We utilized the `.describe()` method to obtain a statistical summary of the numerical columns. This provided insights into the count, mean, standard deviation, min, max, and quartile values for each feature, offering a quick overview of the data distribution.
3. **Filtering Data**: We filtered the DataFrame to include only rows where the 'total_bedrooms' column was greater than 310, creating `df_filtered`. This allowed us to focus on properties with a significant number of bedrooms.
4. **Dropping Rows**: From the filtered DataFrame, we dropped the first and last rows, resulting in `df_modified`. This operation demonstrated how to selectively remove data entries from the beginning and end of a DataFrame.
5. **Calculating Mean and Visualization**: We calculated the mean of the 'households' column and then created a scatter plot. The plot visualized the

distribution of households across the dataset, with a horizontal line indicating the mean, providing a clear visual reference.

6. **Handling Missing Values**: We checked for NaN values in `df_modified`. In this specific dataset, no NaN values were found, so no replacement was necessary. If they were present, the strategy was to replace them with the overall mean of numerical values.
7. **Geographical Visualization**: A scatter plot of 'latitude' versus 'longitude' was generated to visualize the geographical distribution of the housing data points, revealing clusters and patterns in property locations.
8. **Min-Max Normalization**: We applied Min-Max normalization to 'median_income' and 'housing_median_age' columns. This transformed the values to a scale between 0 and 1, which is crucial for many machine learning