```
%%capture
!pip install numpy pandas streamlit gdown currencyconverter


import numpy as np

# For readability purposes, we will disable scientific notation for numbers
np.set_printoptions(suppress=True)


import os
import shutil

import gdown
from numpy import genfromtxt

# Download file from Google Drive
# This file is based on data from: http://insideairbnb.com/get-the-data/
file_id_1 = "13fyESiH1ZEnMV6eabAyhe20t4W6peEWK"
downloaded_file_1 = "WK1_Airbnb_Amsterdam_listings_proj.csv"

# Download the file from Google Drive
gdown.download(id=file_id_1, output=downloaded_file_1)
```

```
Downloading...
From: https://drive.google.com/uc?id=13fyESiH1ZEnMV6eabAyhe20t4W6peEWK
To: /content/WK1_Airbnb_Amsterdam_listings_proj.csv
100%|██████████| 246k/246k [00:00<00:00, 24.4MB/s]
'WK1_Airbnb_Amsterdam_listings_proj.csv'
```

```
from numpy import genfromtxt

my_data = genfromtxt(downloaded_file_1, delimiter="|", dtype="unicode")


my_data[:,:4]
```

```
array([['', '0', '1', '2'],
       ['id', '23726706', '35815036', '31553121'],
       ['price', '$88.00', '$105.00', '$152.00'],
       ['latitude', '52.34916', '52.42419', '52.43237'],
       ['longitude', '4.97879', '4.95689', '4.91821']], dtype='<U18')
```

```
# Remove the first column and row
matrix = my_data[1:,1:]

# Print out the first four columns
matrix[:,:4]
```

```
array([['23726706', '35815036', '31553121', '34745823'],
       ['$88.00', '$105.00', '$152.00', '$87.00'],
       ['52.34916', '52.42419', '52.43237', '52.2962'],
       ['4.97879', '4.95689', '4.91821', '5.01231']], dtype='<U18')
```

```
# Shift the matrix by 90 degrees
matrix = my_data[1:,1:].transpose()
# Print out the first five rows
# Entries: airbnb_id, price_usd, latitude, longitude
matrix[:5,:]
```

```
array([['23726706', '$88.00', '52.34916', '4.97879'],
       ['35815036', '$105.00', '52.42419', '4.95689'],
       ['31553121', '$152.00', '52.43237', '4.91821'],
       ['34745823', '$87.00', '52.2962', '5.01231'],
       ['44586947', '$160.00', '52.31475', '5.0303']], dtype='<U18')
```

```
# Remove the dollar sign
matrix = np.char.replace(matrix,"$","")

# Remove the comma
matrix = np.char.replace(matrix,",","")


# Check if the dollar sign is in our dataset
matrix[np.char.find(matrix, "$") > -1]
```

```
array([], dtype='<U18')
```

```
# Check if the comma sign is in our dataset
matrix[np.char.find(matrix, ",") > -1]
```

```
array([], dtype='<U18')
```

```
# Change Unicode to float32
matrix = matrix.astype(np.float32)
# Print out the first five rows (and inspect the dtype for correctness)
```

```
# Entries: airbnb_id, price_usd, latitude, longitude
matrix[:5,:]
```

```
    array([[23726706.     ,       88.     ,       52.34916,        4.97879],
           [35815036.     ,      105.     ,       52.42419,        4.95689],
           [31553120.     ,      152.     ,       52.43237,        4.91821],
           [34745824.     ,       87.     ,       52.2962 ,        5.01231],
           [44586948.     ,      160.     ,       52.31475,        5.0303 ]],
          dtype=float32)
```

```
, matrixlib
from currency_converter import CurrencyConverter

cc = CurrencyConverter()

# Entries: airbnb_id, price_usd, latitude, longitude
matrix[:5,:]
```

```
    array([[23726706.     ,       88.     ,       52.34916,        4.97879],
           [35815036.     ,      105.     ,       52.42419,        4.95689],
           [31553120.     ,      152.     ,       52.43237,        4.91821],
           [34745824.     ,       87.     ,       52.2962 ,        5.01231],
           [44586948.     ,      160.     ,       52.31475,        5.0303 ]],
          dtype=float32)
```

```
matrix[:, 1]
```

```
    array([ 88., 105., 152., ..., 180., 174.,  65.], dtype=float32)
```

```
cc.currencies
```

```
    {'AUD',
     'BGN',
     'BRL',
     'CAD',
     'CHF',
     'CNY',
     'CYP',
     'CZK',
     'DKK',
     'EEK',
     'EUR',
     'GBP',
     'HKD',
     'HRK',
     'HUF',
     'IDR',
     'ILS',
     'INR',
     'ISK',
     'JPY',
     'KRW',
     'LTL',
     'LVL',
     'MTL',
     'MXN',
     'MYR',
     'NOK',
     'NZD',
     'PHP',
     'PLN',
     'ROL',
     'RON',
     'RUB',
     'SEK',
     'SGD',
     'SIT',
     'SKK',
     'THB',
     'TRL',
     'TRY',
     'USD',
     'ZAR'}
```

```
# Get the rate of conversaton from the US dollar to your currency of choice
gbp_rate = cc.convert(1, "USD", "GBP")

# Multiply the dollar column by your currency of choice
matrix[:, 1] = matrix[:, 1] * gbp_rate
matrix[:, 1]
```

```
    array([ 70.820724,  84.502   , 122.326706, ..., 144.86057 , 140.03189 ,
            52.31076 ], dtype=float32)
```

```
# Multiply the dollar column by the inflation percentage (1.00 + inflation)
inflation = 0.091
matrix[:, 1] = matrix[:, 1] * (1.00 + inflation)
matrix[:, 1]
```

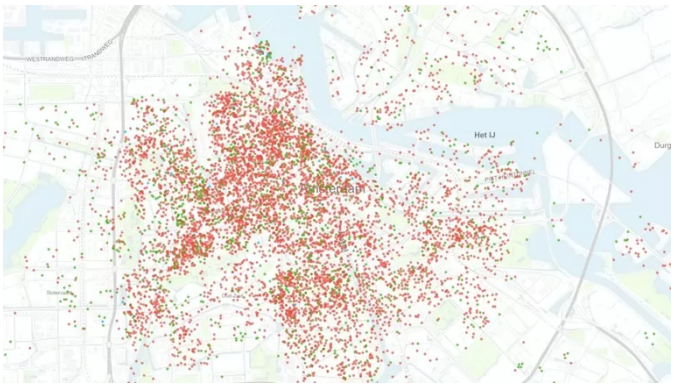
```
array([ 77.26541 ,  92.19168 , 133.45844 , ..., 158.04288 , 152.77478 ,
        57.071037], dtype=float32)
```

```
# Round down the new currency column to 2 decimals
matrix[:,1] = np.round(matrix[:, 1], 2)
matrix[:,1]
```

```
array([ 77.27,  92.19, 133.46, ..., 158.04, 152.77,  57.07], dtype=float32)
```

```
# Favorite location
latitude = 52.3732
longitude = 4.8914
```

Listing All Listings



Imagine Airbnb Amsterdam decided to deviate from Airbnb Global and provide a feature on their website that showed the best listings for you based on the locations you were planning to visit. Wouldn't it make sense to choose a place to stay in a location closest to where you're likely to go most often?

So this is the most exciting part: We're going to calculate just that! We will limit your results to our favorite location in Amsterdam (as chosen above) and the surrounding available Airbnb listings using math and NumPy.

```
import math

def from_location_to_airbnb_listing_in_meters(lat1: float, lon1: float, lat2: list, lon2: list):
    # Source: https://community.esri.com/t5/coordinate-reference-systems-blog
    # /distance-on-a-sphere-the-haversine-formula/ba-p/902128

    R = 6371000  # Radius of Earth in meters
    phi_1 = math.radians(lat1)
    phi_2 = math.radians(lat2)

    delta_phi = math.radians(lat2 - lat1)
    delta_lambda = math.radians(lon2 - lon1)

    a = (
        math.sin(delta_phi / 2.0) ** 2
        + math.cos(phi_1) * math.cos(phi_2) * math.sin(delta_lambda / 2.0) ** 2
    )

    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    meters = R * c  # Output distance in meters

    return round(meters, 0)


# Create a loop or vectorized way to calculate the distance,
# going over all latitude and longitude entries in the dataset
conv_to_meters = np.vectorize(from_location_to_airbnb_listing_in_meters)
conv_to_meters(latitude, longitude, matrix[:, 2], matrix[:, 3])
```

```
array([6508., 7204., 6826., ..., 5944., 6207., 5850.])
```

```
%%timeit -r 4 -n 100

# Allow a Python function to be used in a (semi-)vectorized way
conv_to_meters = np.vectorize(from_location_to_airbnb_listing_in_meters)

# Apply the function, use timeit
conv_to_meters(latitude, longitude, matrix[:, 2], matrix[:, 3])
```

```
20.8 ms ± 4.8 ms per loop (mean ± std. dev. of 4 runs, 100 loops each)
```

```
def from_location_to_airbnb_listing_in_meters(
    lat1: float, lon1: float, lat2: np.ndarray, lon2: np.ndarray
):
    R = 6371000  # radius of Earth in meters
    phi_1 = np.radians(lat1)
    phi_2 = np.radians(lat2)

    delta_phi = np.radians(lat2 - lat1)
    delta_lambda = np.radians(lon2 - lon1)

    a = (
        np.sin(delta_phi / 2.0) ** 2
        + np.cos(phi_1) * np.cos(phi_2) * np.sin(delta_lambda / 2.0) ** 2
    )

    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1 - a))

    meters = R * c  # output distance in meters

    return np.round(meters, 0)


# Run the converted NumPy method and check if it works
from_location_to_airbnb_listing_in_meters(
    latitude, longitude, matrix[:, 2], matrix[:, 3]
)
```

```
array([6508., 7204., 6826., ..., 5944., 6207., 5850.])
```

```
from_location_to_airbnb_listing_in_meters(
    latitude, longitude, matrix[:, 2], matrix[:, 3]
)
```

```
array([6508., 7204., 6826., ..., 5944., 6207., 5850.])
```

```
# Copy the code from Task 12 and add a timeit function above this comment
%%timeit -r 4 -n 100

from_location_to_airbnb_listing_in_meters(
    latitude, longitude, matrix[:, 2], matrix[:, 3]
)
```

```
676 µs ± 116 µs per loop (mean ± std. dev. of 4 runs, 100 loops each)
```

```
%%timeit -r 4 -n 100

from_location_to_airbnb_listing_in_meters(
    latitude, longitude, matrix[:, 2], matrix[:, 3]
)
```

```
672 µs ± 141 µs per loop (mean ± std. dev. of 4 runs, 100 loops each)
```

Prep the Dataset for Download!

Now that we've created a function to calculate the distance in meters for every Airbnb listing, we'll perform this calculation on the entire dataset and add the outputs to the matrix as a new column.

Next to that, we'll add another column that contains only ones and zeros to represent the "color" of an entry/row. This column can be used later if you want to turn this dataset into an app using Streamlit. This resource is great for when you want to translate your Python projects into an interactive website.

We've selected the coordinates of the Royal Palace Amsterdam.

```
# Run the previous method
meters = from_location_to_airbnb_listing_in_meters(
    latitude, longitude, matrix[:, 2], matrix[:, 3]
)

# Add an axis to make concatenation possible
meters = meters.reshape(-1, 1)

# Append the distance in meters to the matrix
matrix = np.concatenate((matrix, meters), axis=1)
```

```
meters.shape
```

```
(6173, 1)
```

```
matrix.shape
```

```
(6173, 5)
```

```
# Append a color to the matrix
colors = np.zeros(meters.shape)
matrix_colors = np.concatenate((matrix, colors), axis=1)

matrix_colors.shape
```

```
    (6173, 6)
```

```
# Append our entry to the matrix
fav_entry = np.array([1, 0, 52.3732, 4.8914, 0, 1]).reshape(1, -1) # coordinates to Royal Palace Amsterdam
fav_entry.shape
```

```
    (1, 6)
```

```
matrix = np.concatenate((matrix_colors,fav_entry),axis=0)

# Entries: airbnb_id, price, latitude, longitude,
# meters from favorite point, color
matrix.shape
```

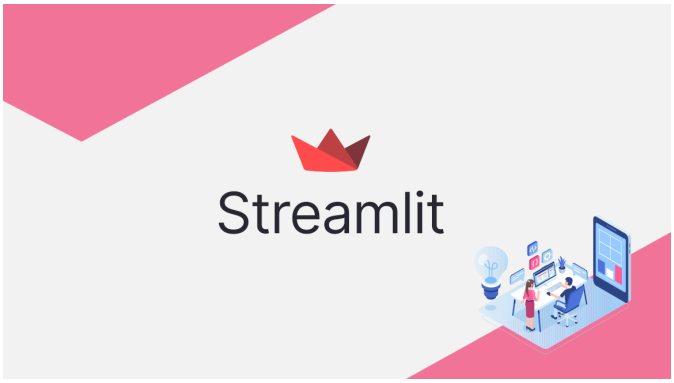
```
    (6174, 6)
```

```
# Export the data to use in the primer for next week
np.savetxt("WK1_Airbnb_Amsterdam_listings_proj_solution.csv", matrix, delimiter=",")
```

```
from google.colab import files

# Download the file locally
files.download('WK1_Airbnb_Amsterdam_listings_proj_solution.csv')
```

> Indented block



We are going to use Streamlit Share to host your projects. It's a website that allows us to host our interactive project.

```
%%writefile streamlit_app.py
import pandas as pd
import plotly.express as px
import streamlit as st

# Display title and text
st.title("Week 1 - Data and visualization")
st.markdown("Here we can see the dataframe created during this weeks project.")

# Read dataframe
dataframe = pd.read_csv(
    "WK1_Airbnb_Amsterdam_listings_proj_solution.csv",
    names=[
        "Airbnb Listing ID",
        "Price",
        "Latitude",
        "Longitude",
        "Meters from chosen location",
        "Location",
    ],
)

# We have a limited budget, therefore we would like to exclude
# listings with a price above 100 pounds per night
dataframe = dataframe[dataframe["Price"] <= 100]

# Display as integer
dataframe["Airbnb Listing ID"] = dataframe["Airbnb Listing ID"].astype(int)
# Round of values
dataframe["Price"] = "£ " + dataframe["Price"].round(2).astype(str) #
```

```
# Rename the number to a string
dataframe["Location"] = dataframe["Location"].replace(
    {1.0: "To visit", 0.0: "Airbnb listing"}
)

# Display dataframe and text
st.dataframe(dataframe)
st.markdown("Below is a map showing all the Airbnb listings with a red dot and the location we've chosen with a blue dot.")

# Create the plotly express figure
fig = px.scatter_mapbox(
    dataframe,
    lat="Latitude",
    lon="Longitude",
    color="Location",
    color_discrete_sequence=["blue", "red"],
    zoom=11,
    height=500,
    width=800,
    hover_name="Price",
    hover_data=["Meters from chosen location", "Location"],
    labels={"color": "Locations"},
)
fig.update_geos(center=dict(lat=dataframe.iloc[0][2], lon=dataframe.iloc[0][3]))
fig.update_layout(mapbox_style="stamen-terrain")

# Show the figure
st.plotly_chart(fig, use_container_width=True)
```

```
    Overwriting streamlit_app.py
```

The **%%writefile [FILE_NAME].[FILE_EXTENSION]** command let's us save the code written in the cells in your Google Colab instance. Having it saved like that enables us to download it as a file, as seen below:

```
from google.colab import files

# Download the file locally
files.download('streamlit_app.py')
```

```
%%writefile requirements.txt
pandas
streamlit
plotly
```

```
    Overwriting requirements.txt
```

```
from google.colab import files

# Download the file locally
files.download('requirements.txt')
```