4. Data Visualization

This is the second out of the five guiding pdf files that aims to make it easier for the developer to get started with the project. In 4 sections, we will dive into how to generate CSVs of the database's content and to visualize it in bubble- and pie-charts

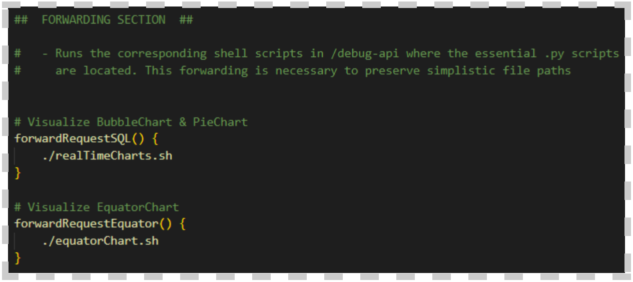
1) **Automation:** The information in this pdf file is based on the shell script "/instructions-executions/executionary/4. data-visualization.sh". Use this script as a reference to the explanations provided in this pdf. To be clear, you can just run "4. data-visualization.sh" and everything is taken care of automatically. With that said, this file is just providing elaborated explanations of the execution procedure. To run the associated shell script, you need to pass certain arguments in the format provided below:

```
./"4. data-visualization.sh" [OPTIONAL PARAMETERS] [OPTIONAL FLAGS]

        OPTIONAL PARAMETERS
                "equatorChart=True"
                "realTimeCharts=True"
                "equatorChart="True" "realTimeCharts=True"

        OPTIONAL FLAGS
                -p
                -b
                -b -p
```

2) **Configurations:** In the very shell script that this pdf is dedicated to, there are no configurations available. Due to the system's architecture, these settings are to be found in the following scripts:
   - *configuration.yml*
   - *debug-api/realTimeCharts.sh*
   - *debug-api/equatorChart.sh*
   - *debug-api/charts/*.py*

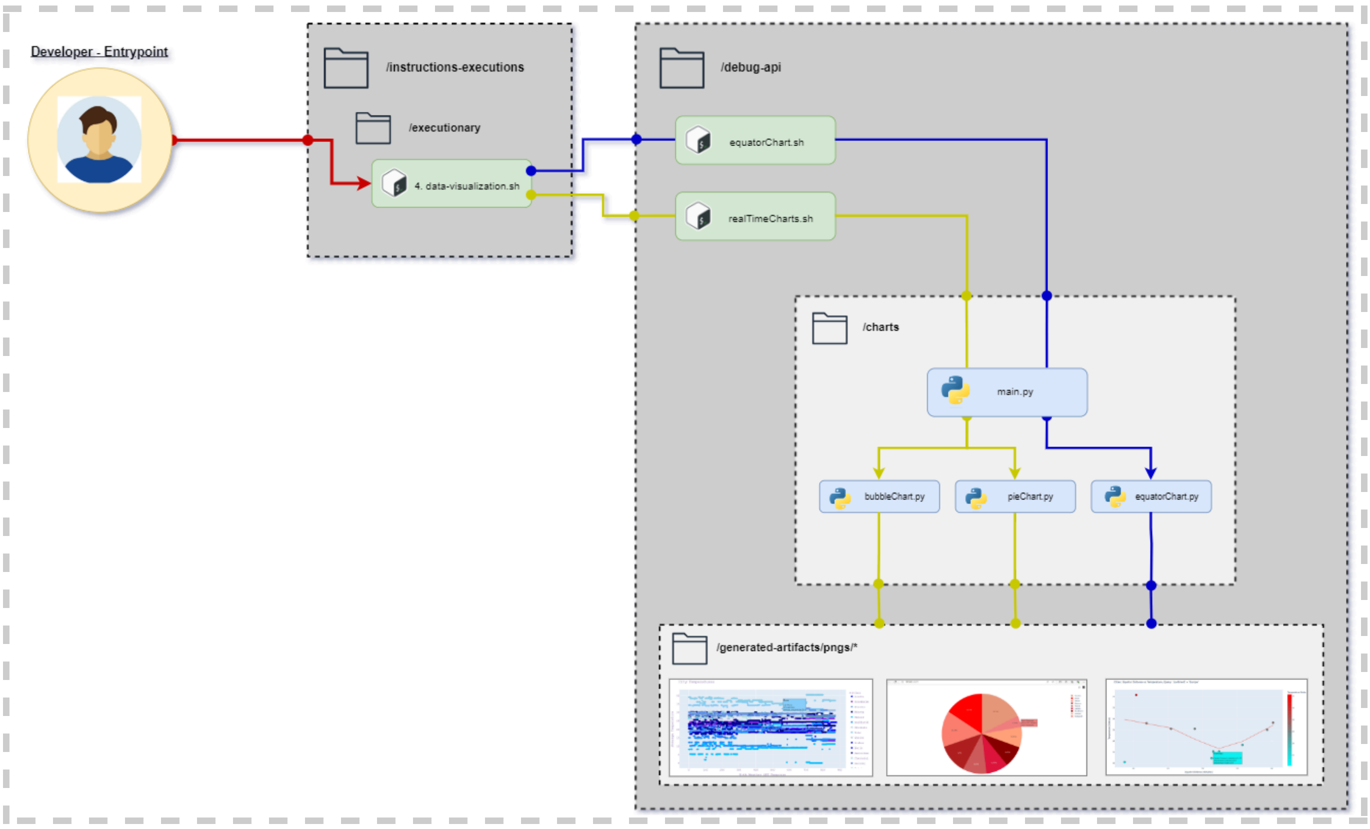
   Since the shell script primarily focused in this tutorial ("4. data-visualization.sh") forwards the dataflow to the scripts closer to the actual implementation, the architecture resembles a hierarchy such that this very script is at the top as the most high-level entity. In other words, this script is the entrypoint of execution, so if a developer wants to change configurations for specific charts that person has to navigate to a low-level script tied to the selected chart. For you to acquire a deeper insight and understand the essentials, I will first touch briefly on the shell-script-forwarding and why it is needed, followed by the general architecture of dependencies between the system's involved modules:

   Here, we forward or redirect the dataflow to the associated shell script in the debug-api module



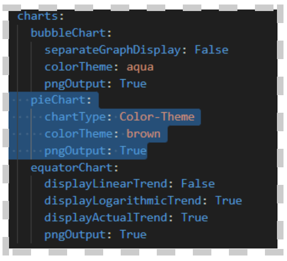
```
##  FORWARDING SECTION  ##

#   - Runs the corresponding shell scripts in /debug-api where the essential .py scripts
#     are located. This forwarding is necessary to preserve simplistic file paths


# Visualize BubbleChart & PieChart
forwardRequestSQL() {
    ./realTimeCharts.sh
}

# Visualize EquatorChart
forwardRequestEquator() {
    ./equatorChart.sh
}
```

It is fundamental that you understand these two shell scripts, as they embody one of the distinguishing features for this project as a whole. Once again, whether the codeflow is directed to realTimeCharts.sh or equatorChart.sh is determined by the passed arguments when running 4. data-visualiation.sh. Depending on what script you select, you need to understand that they are connecting to two distinct databases. "realTimeCharts.sh" is sending requests to the PostgresSQL and is therefore involved in the Kafa-Flink real-time process. "equatorChart.sh" on the other hand, has its own JSON database in "apis/database" and uses queries to derive a filtered response from "db.json" that is stored in "response.json". Both of these shell scripts have one noteworthy boolean variable in common: $RECREATE_DATABASE. This one governs if the code should completely override all existing instances with the "new ones" or not. The "new ones" correspond to your configurations. If you added more cities to the datastream or equatorChart, then setting this variable to 'True' is appropriate. To not confuse you with the dataflow I made a diagram similar to the structure of a component diagram that embodies subfolders as components:



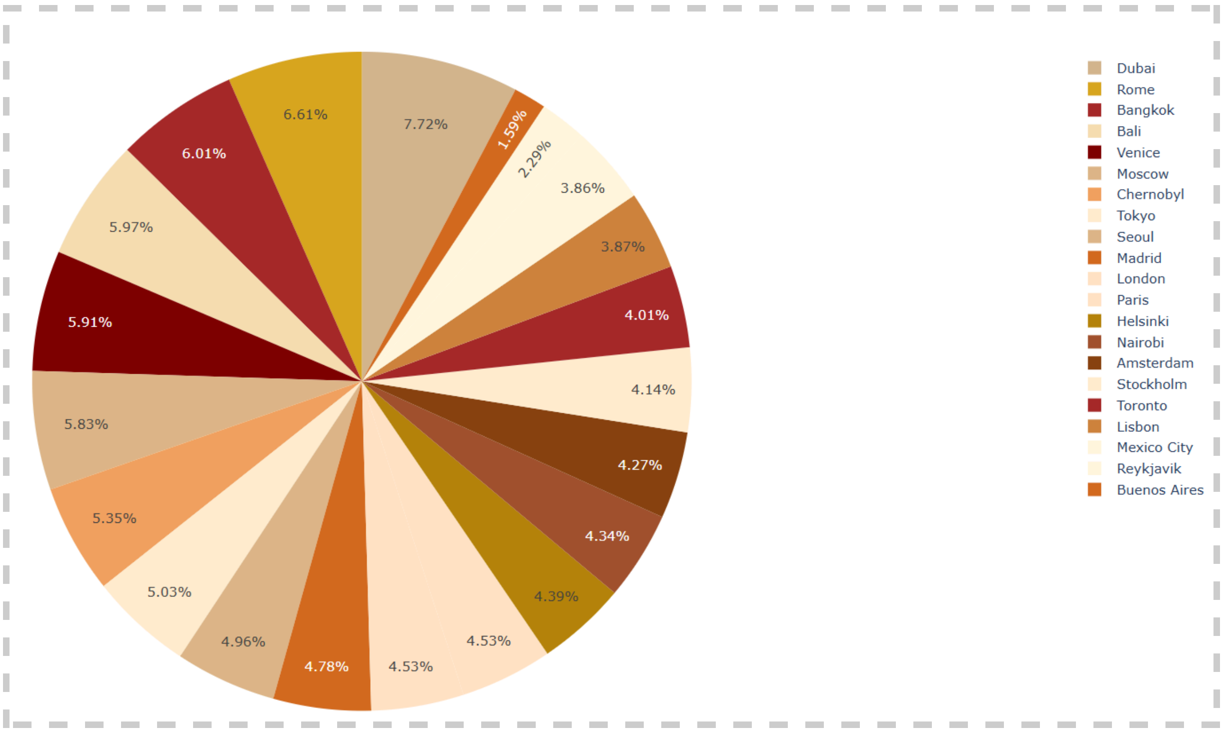
3) **Execution:** Simply run "4. data-visualization.sh". Keep in mind that your settings in "configuration.yml" governs the output to a great extent:

Here, we select the color-themed pie chart and set it to brown:

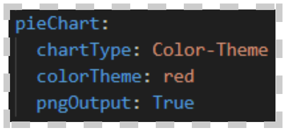


```
charts:
  bubbleChart:
    separateGraphDisplay: False
    colorTheme: aqua
    pngOutput: True
  pieChart:
    chartType: Color-Theme
    colorTheme: brown
    pngOutput: True
  equatorChart:
    displayLinearTrend: False
    displayLogarithmicTrend: True
    displayActualTrend: True
    pngOutput: True
```

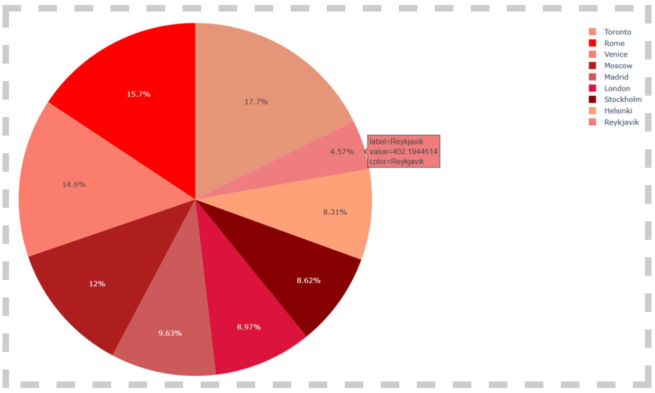
In turn, your output looks like this:



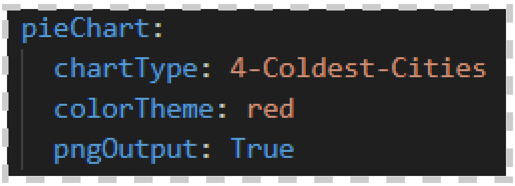
Let's change the theme to red:



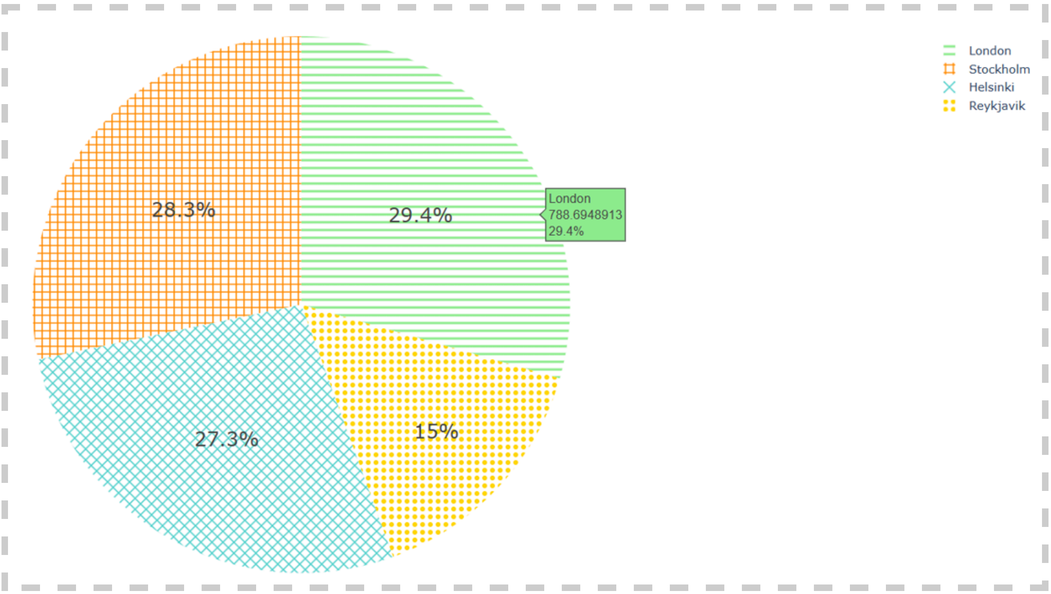
```
pieChart:
  chartType: Color-Theme
  colorTheme: red
  pngOutput: True
```



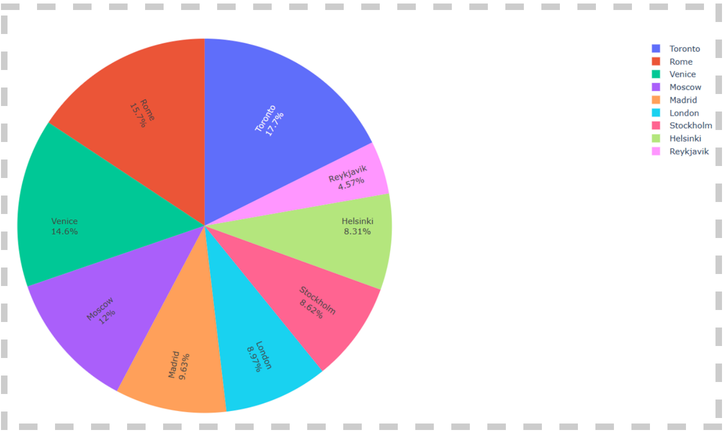
Now, we shift the type from 'Color-Theme' to '4-Coldest Cities'. Note that as this mode is selected, the value of $colorTheme is neglected, so don't remove it from the file:



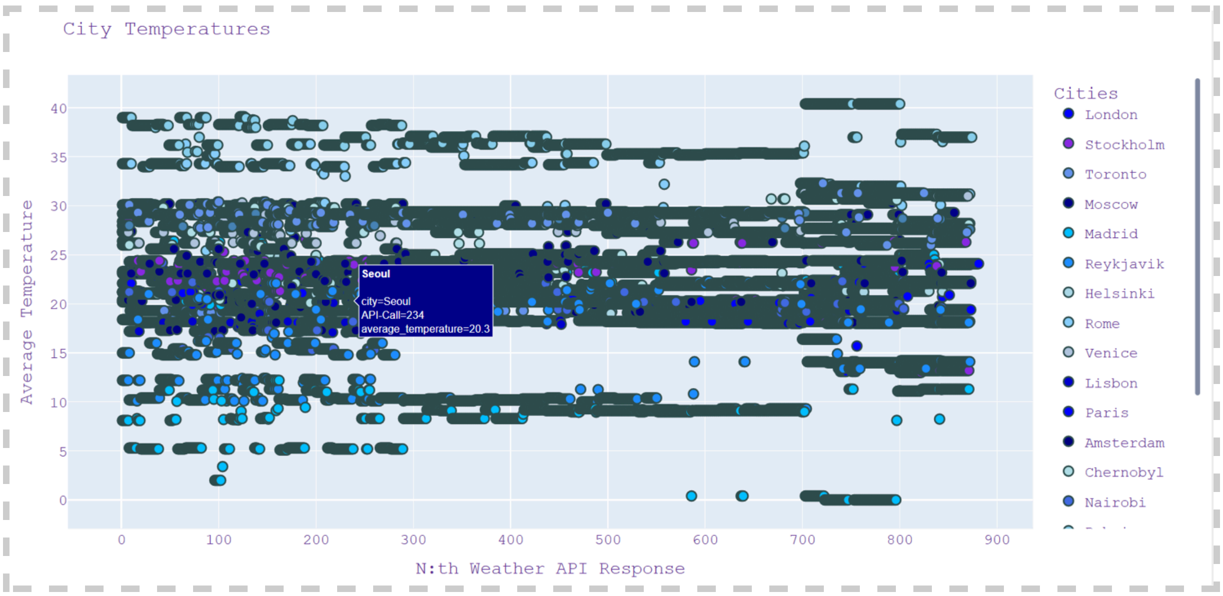
```
pieChart:
  chartType: 4-Coldest-Cities
  colorTheme: red
  pngOutput: True
```



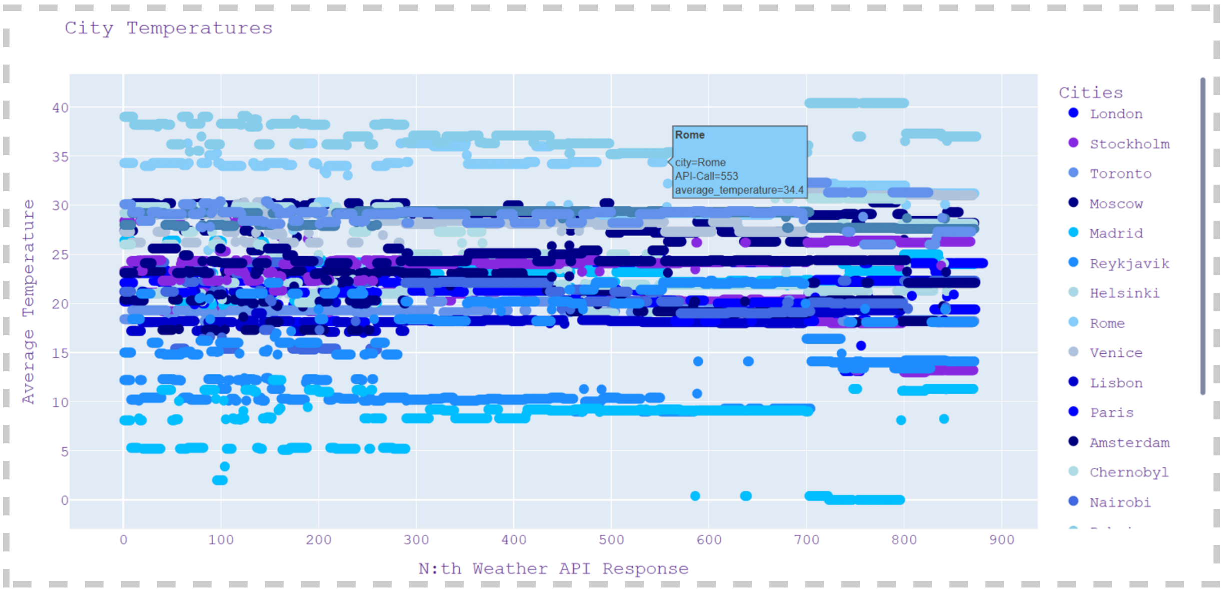
Finally, we set $chartType to ‘Random-Colors’:



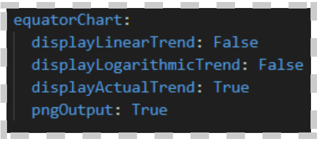
To switch the chart, we pass the argument “-b” for bubbleChart. As mentioned earlier, a few internal settings for the appearance of each chart is contained within respective /debug-api/charts/*.py files. In the output below, we set the outline width to 2 when displaying approximately 18.000 unique database instances from the PostgresSQL database:



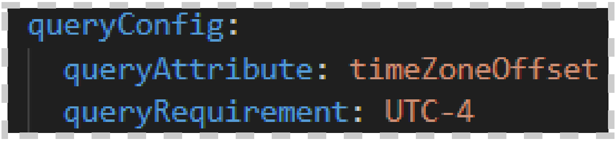
As you can see, the density caused by the large number of instances plotted caused the outline color to dominate the entire visualization, which calls for inaccuracy through the lens of the human-eye. Therefore, let's change the outline width to 0:



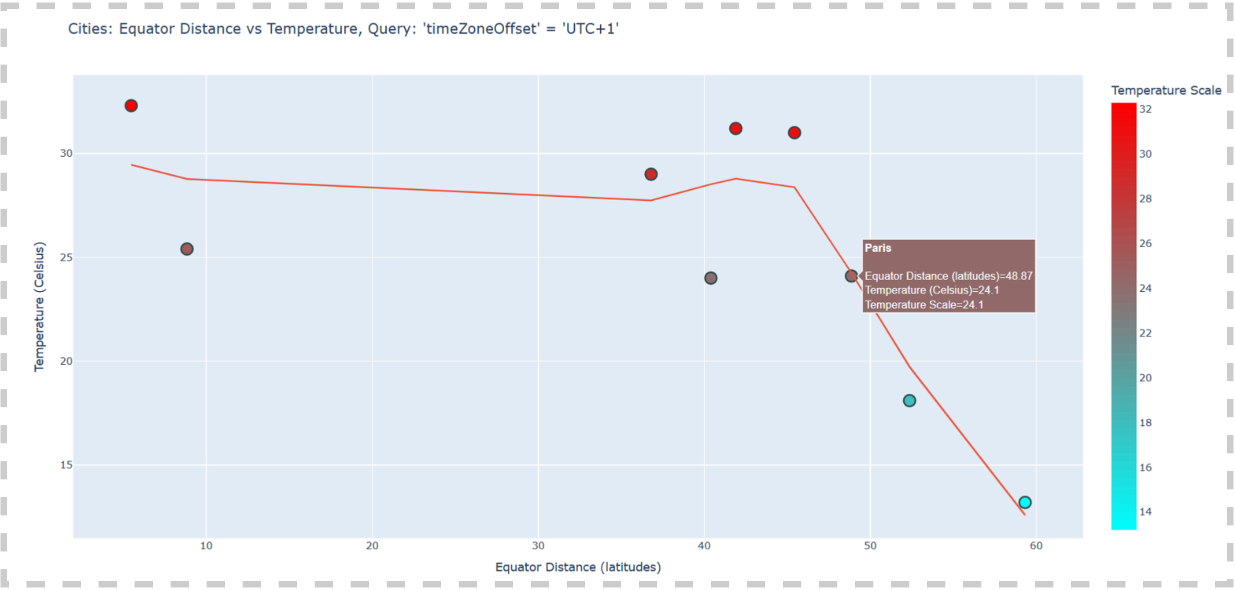
This is much better! Now, we are able to follow each line and its successive y-values along the x-axis. The two charts that are showcased above are commonly denoted as "real-time charts" throughout the documentation of this project, since they are connected to the SQL database that collaborates with the pipeline of Kafka and Flink. In the picture below, the equator chart is shown, which unlike the other charts is derived from a separate JSON database. As always, remember to provide your desired settings in configuration.yml:



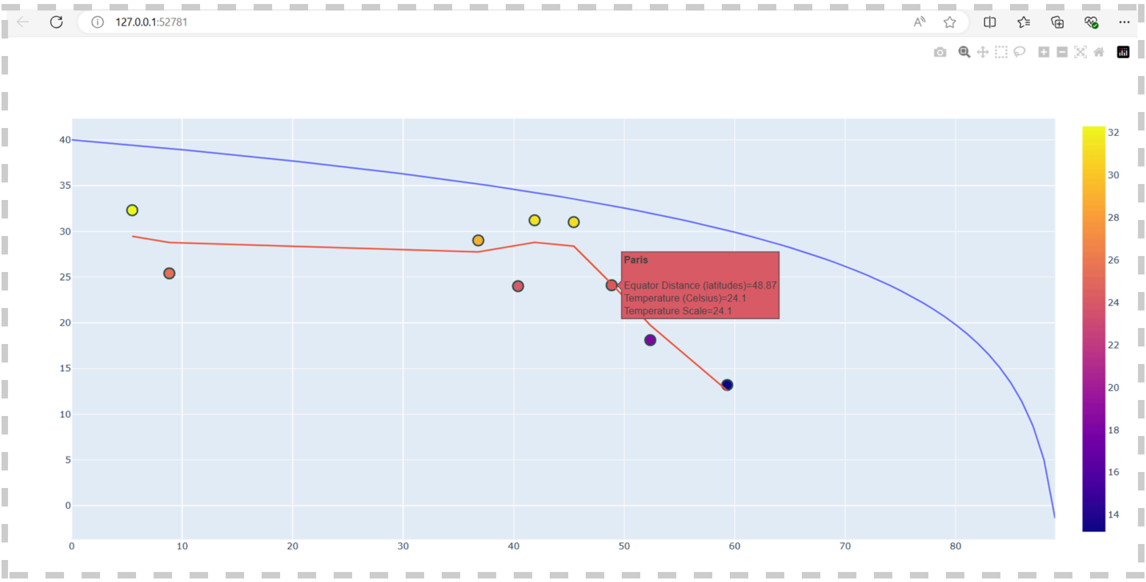
```
equatorChart:
  displayLinearTrend: False
  displayLogarithmicTrend: False
  displayActualTrend: True
  pngOutput: True
```



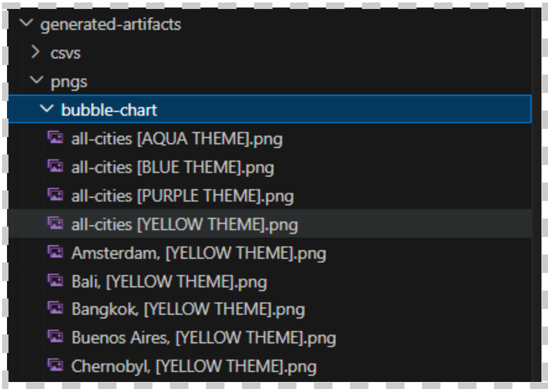
```
queryConfig:
  queryAttribute: timeZoneOffset
  queryRequirement: UTC-4
```



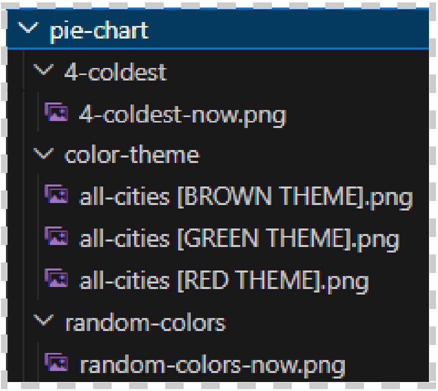
The ‘queryConfig’ section is the unique feature supported by the JSON database. It allows you, as a developer, to query the cities by the timezone offset and continents. Furthermore, don't hesitate to explore the boolean variables in the .yml file. Enabling the expected logarithmic trend on the same chart would generate this output:



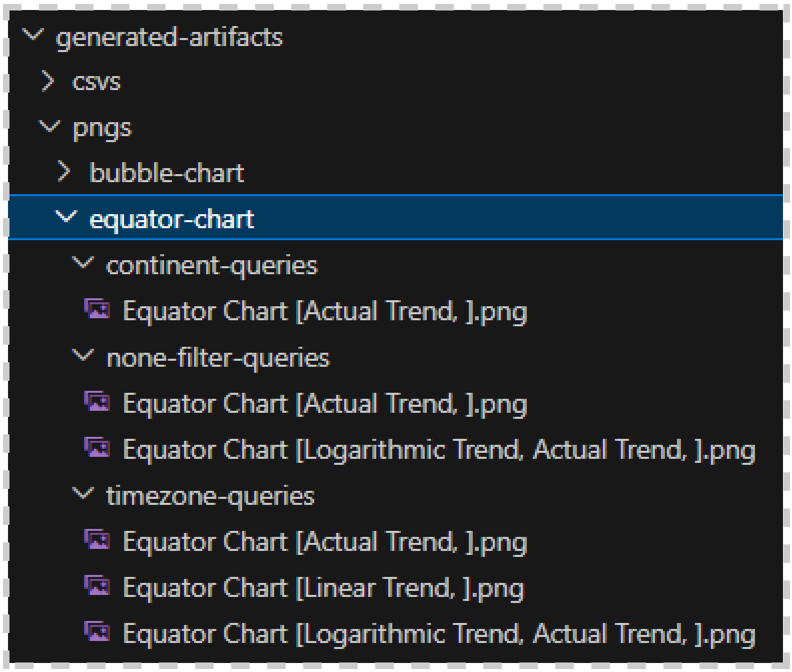
As you probably have observed, each subsection in configuration.yml is accompanied by $pngOutput By default, the charts are displayed in a localhost window once they are plotted. Therefore, enabling this variable would illustrate the visualization in an additional format as a .png file in the local file system of the project. To be specific, the storage locations can be found in "debug-api/generated-artifacts/pngs/*".



```
generated-artifacts
  csvs
  pngs
    bubble-chart
      all-cities [AQUA THEME].png
      all-cities [BLUE THEME].png
      all-cities [PURPLE THEME].png
      all-cities [YELLOW THEME].png
      Amsterdam, [YELLOW THEME].png
      Bali, [YELLOW THEME].png
      Bangkok, [YELLOW THEME].png
      Buenos Aires, [YELLOW THEME].png
      Chernobyl, [YELLOW THEME].png
```



```
pie-chart
  4-coldest
    4-coldest-now.png
  color-theme
    all-cities [BROWN THEME].png
    all-cities [GREEN THEME].png
    all-cities [RED THEME].png
  random-colors
    random-colors-now.png
```



```
generated-artifacts
  csvs
  pngs
    bubble-chart
    equator-chart
      continent-queries
        Equator Chart [Actual Trend, ].png
      none-filter-queries
        Equator Chart [Actual Trend, ].png
        Equator Chart [Logarithmic Trend, Actual Trend, ].png
      timezone-queries
        Equator Chart [Actual Trend, ].png
        Equator Chart [Linear Trend, ].png
        Equator Chart [Logarithmic Trend, Actual Trend, ].png
```

4) **The next step:** In the next tutorial, we will be going through "5. combinatorial-automations.sh" that combines all the shell scripts into a new environment that allows for automation of all possible combinations of configurations for each of the three charts.