## Data collection

1. **Wind Direction**

We downloaded the wind direction, wind speed and humidity data from the Hong Kong Observatory's data open platform for the past twenty years. By analyzing these accumulated data, we can obtain the statistical distribution of some key meteorological characteristics of the study area, so as to carry out Monte Carlo simulation based on this in the future.

Wind direction data is expressed in degrees, with 0 degrees representing due north and 90 degrees representing due east, and so on. When the meteorological station collects data, there are some incomplete data, and these data are marked as "#" in the data set. We can filter out these data and keep only complete data for analysis.

|  |  |
| --- | --- |
| Notation | Description |
| \*\*\* | unavailable |
| # | data incomplete |
| C | data Complete |

Table 1. Notation of wind direction data integrity

In the statistical process, for the convenience of analysis, we divide the wind direction data into eight directions, namely N, NE, E, SE, S, SW, W, NW. We have counted the distribution of wind direction data. The prevailing wind directions in the study area are north, east and northeast. Among them, the prevailing wind direction of north wind is the highest, followed by east wind, and the prevailing wind direction of northeast wind is the lowest. This is related to the geographical location of the study area. The study area is located in the northwest of Hong Kong, and the prevailing wind directions are north, east and northeast.

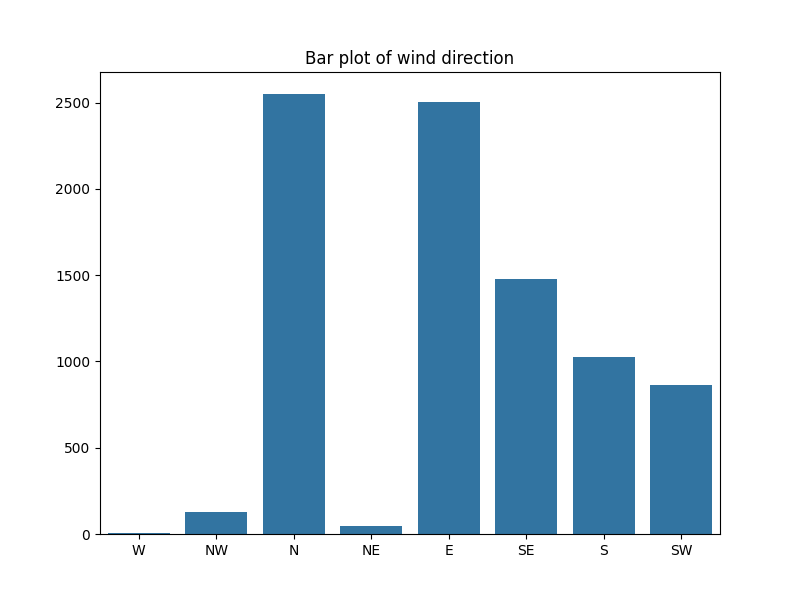


Figure 1. Bar chart of wind direction distribution

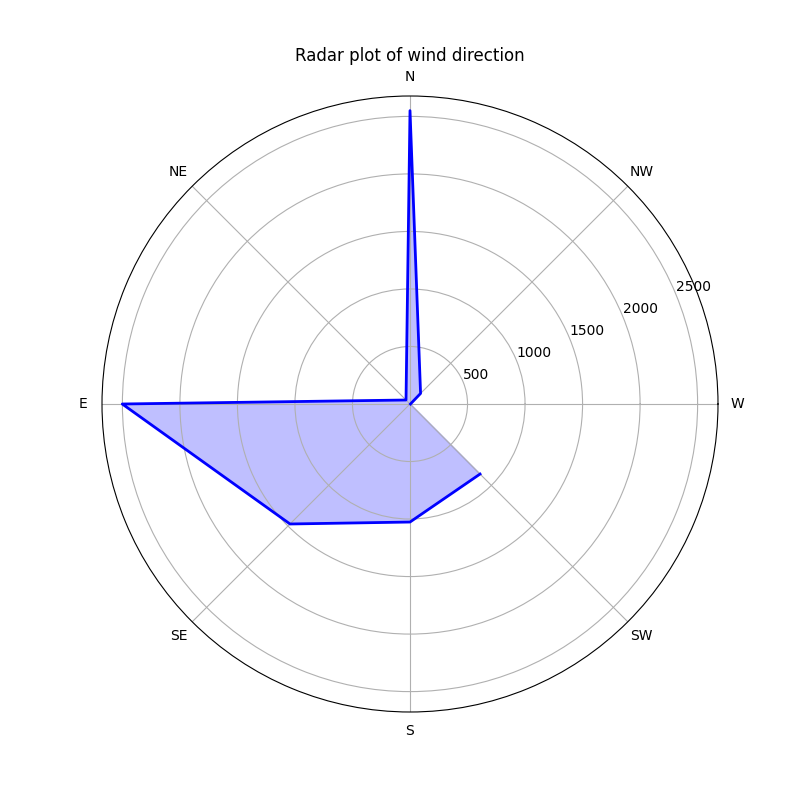


Figure 2. Radar chart of wind direction distribution

After accumulating all the data, we obtained the statistical distribution of wind direction data as follows:

|  |  |
| --- | --- |
| Wind Direction | Proportion |
| N | 0.296650 |
| E | 0.290949 |
| SE | 0.171591 |
| S | 0.119358 |
| SW | 0.100163 |
| NW | 0.015123 |
| NE | 0.005700 |
| W | 0.000465 |

Table 2. Wind direction distribution statistics

1. **Wind speed**

Wind speed data is expressed in meters per second (m/s). We have counted the distribution of wind speed data. The wind speed in the study area is mainly concentrated between 10m/s and 20m/s, with 17m/s wind speed accounting for the highest proportion. The distribution of wind speed is shown in the following figure.

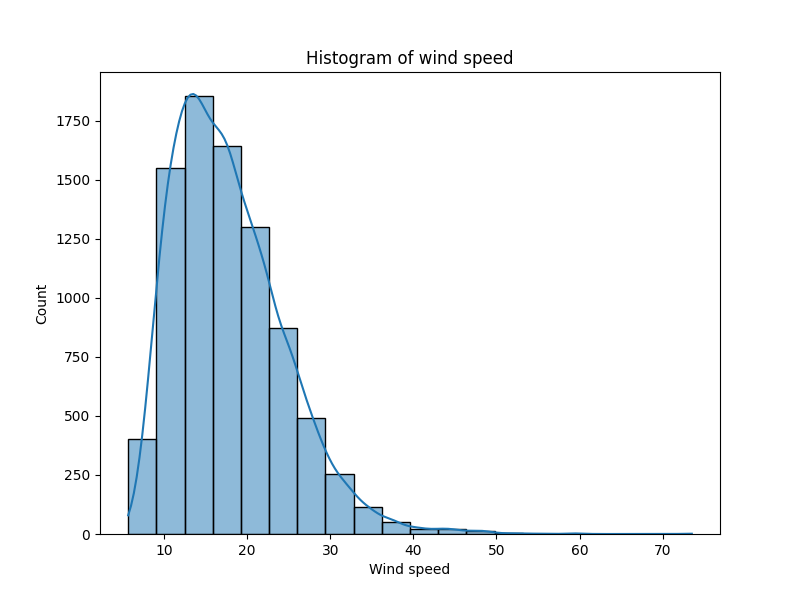


Figure 3. Bar chart of wind speed distribution

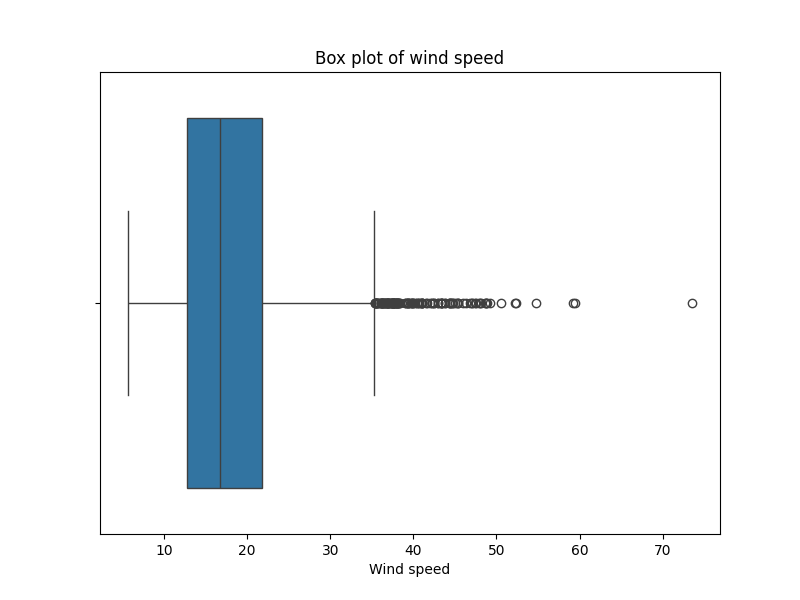


Figure 4. Box plot of wind speed distribution

The wind speed data is visualized year by year, and overlaid on one chart:

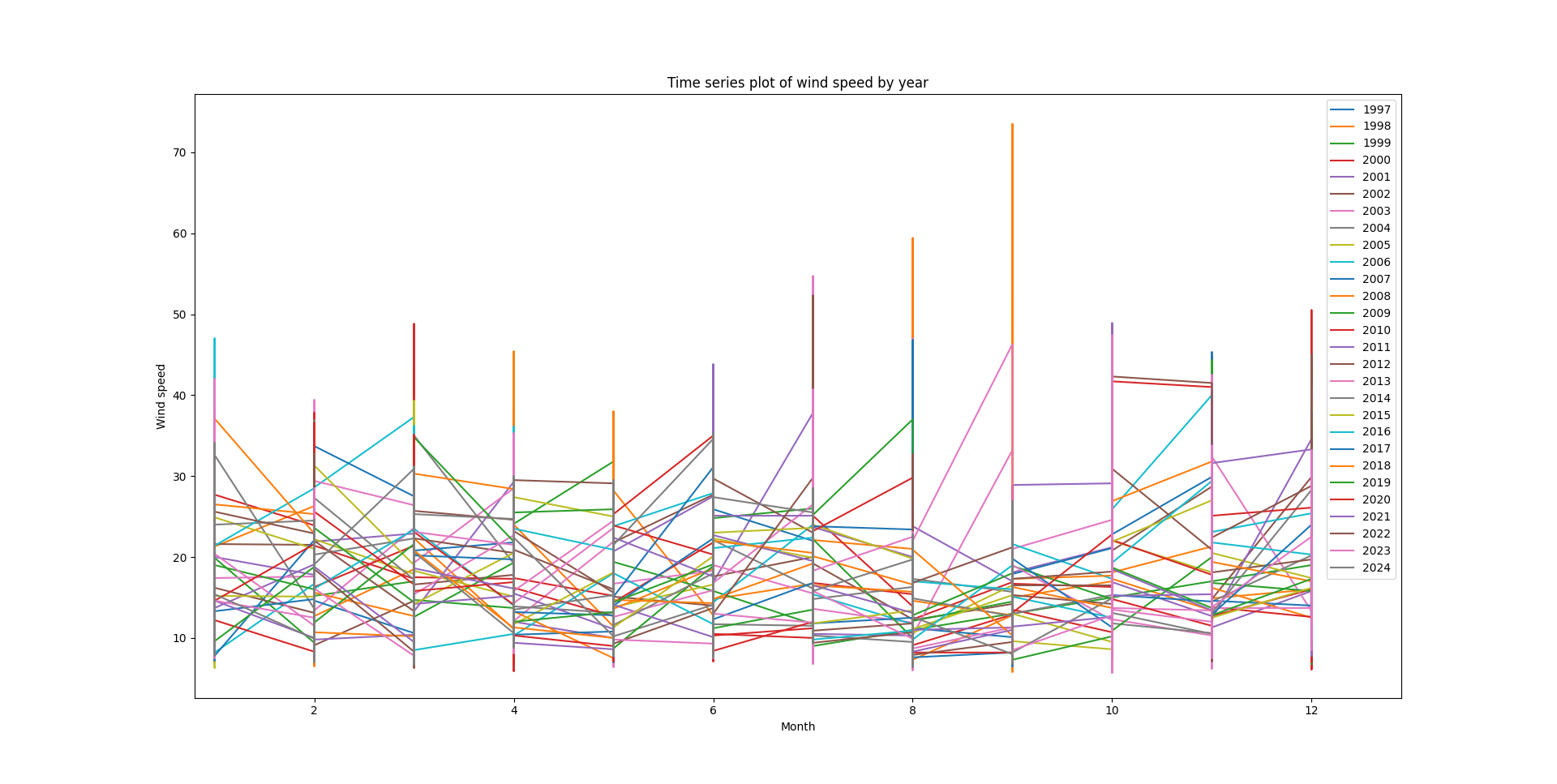


Figure 5. Wind speed distribution by year

After accumulating all the data, we obtained the statistical distribution of wind speed data as follows:

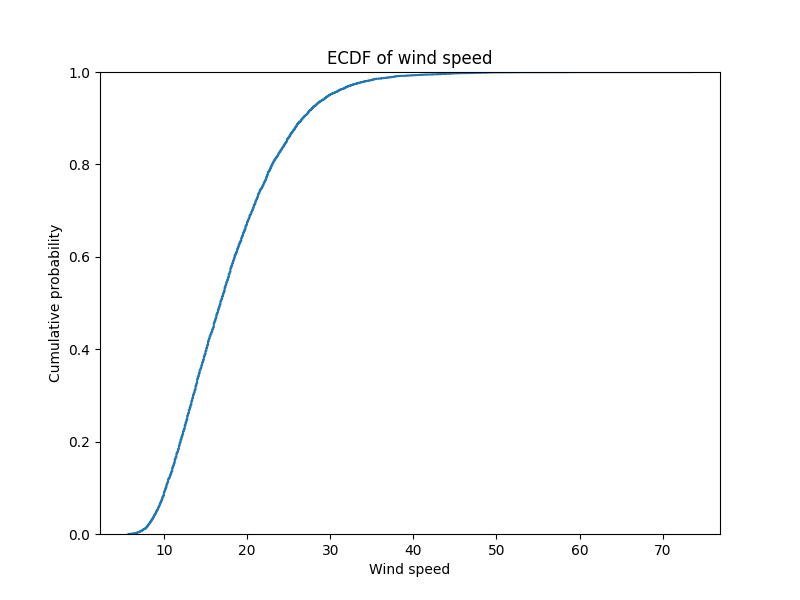


Figure 6. Wind speed accumulation distribution

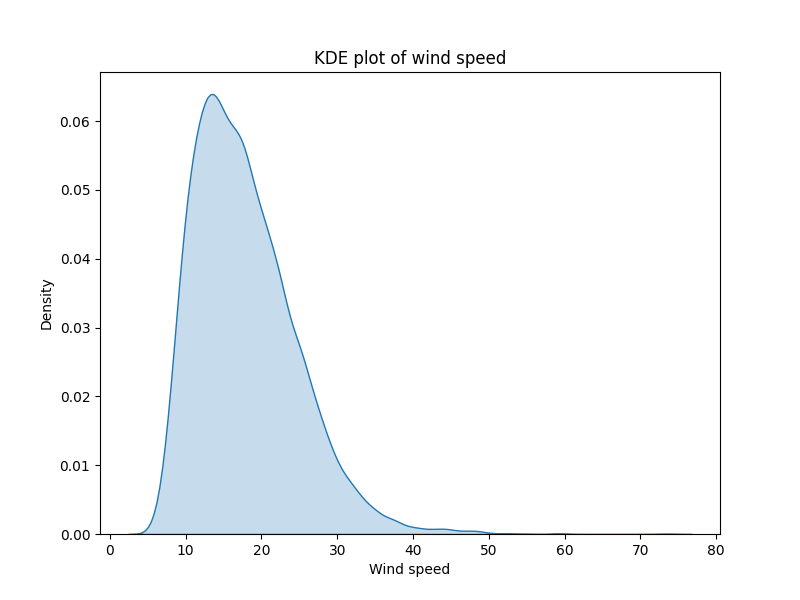


Figure 7. Wind speed probability distribution

|  |  |
| --- | --- |
| Mean Wind Speed | Standard Deviation of Wind Speed |
| 17.842082606166375 | 6.709404241331985 |

Table 3. Wind speed distribution statistics

1. Humidity Data

Humidity data is expressed as a percentage. We have counted the distribution of humidity data. The humidity in the study area is mainly concentrated between 70% and 90%, with 80% humidity accounting for the highest proportion. The distribution of humidity is shown in the following figure.

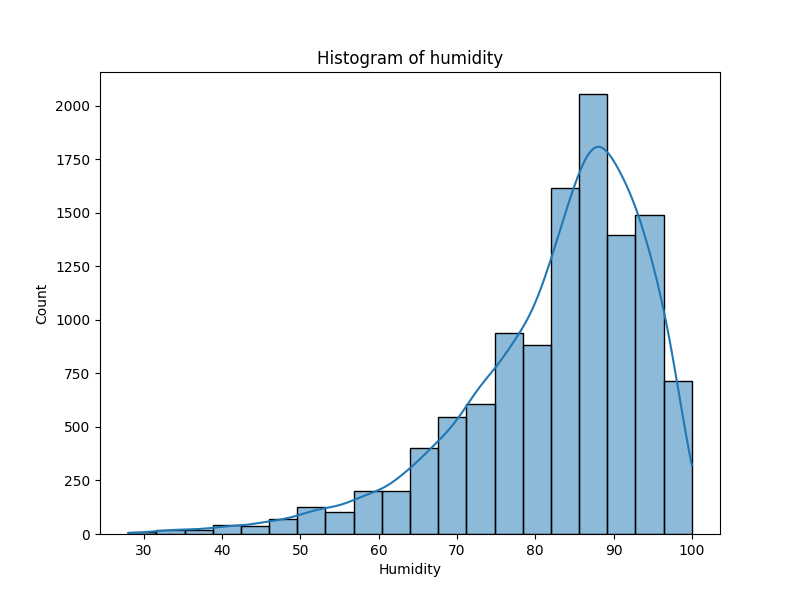


Figure 8. Bar chart of humidity distribution

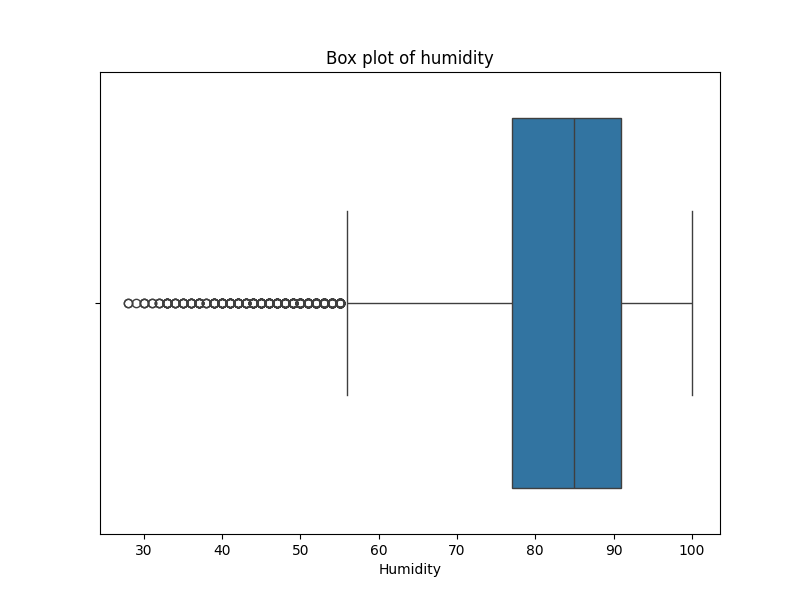


Figure 9. Box plot of humidity distribution

If we overlay them on one chart, we get the following chart:

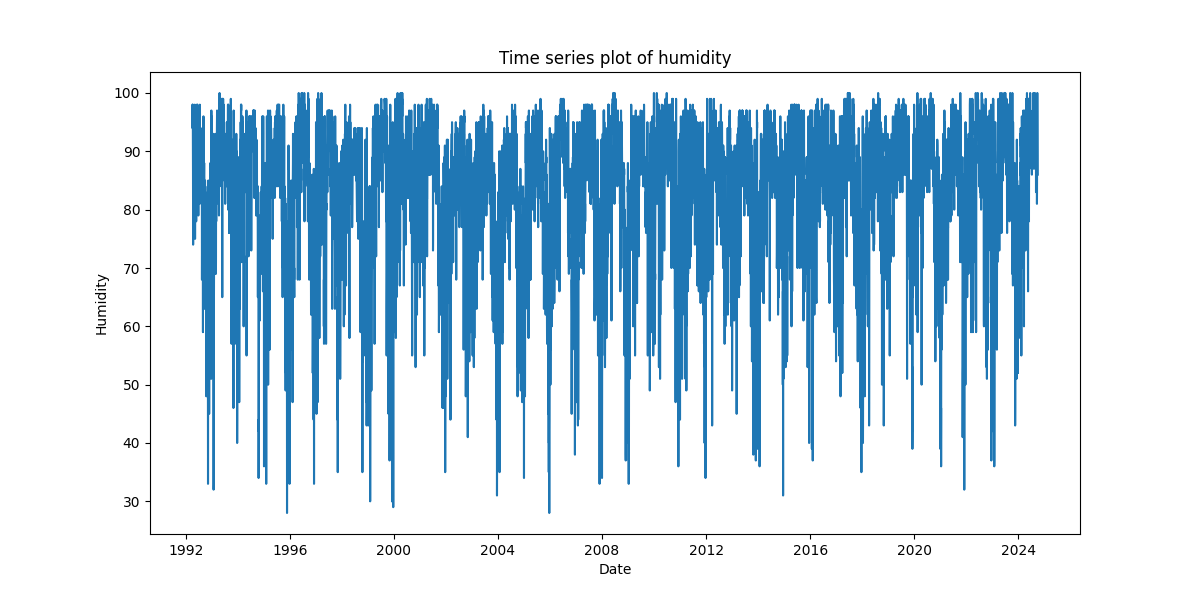


Figure 10. Humidity distribution by year

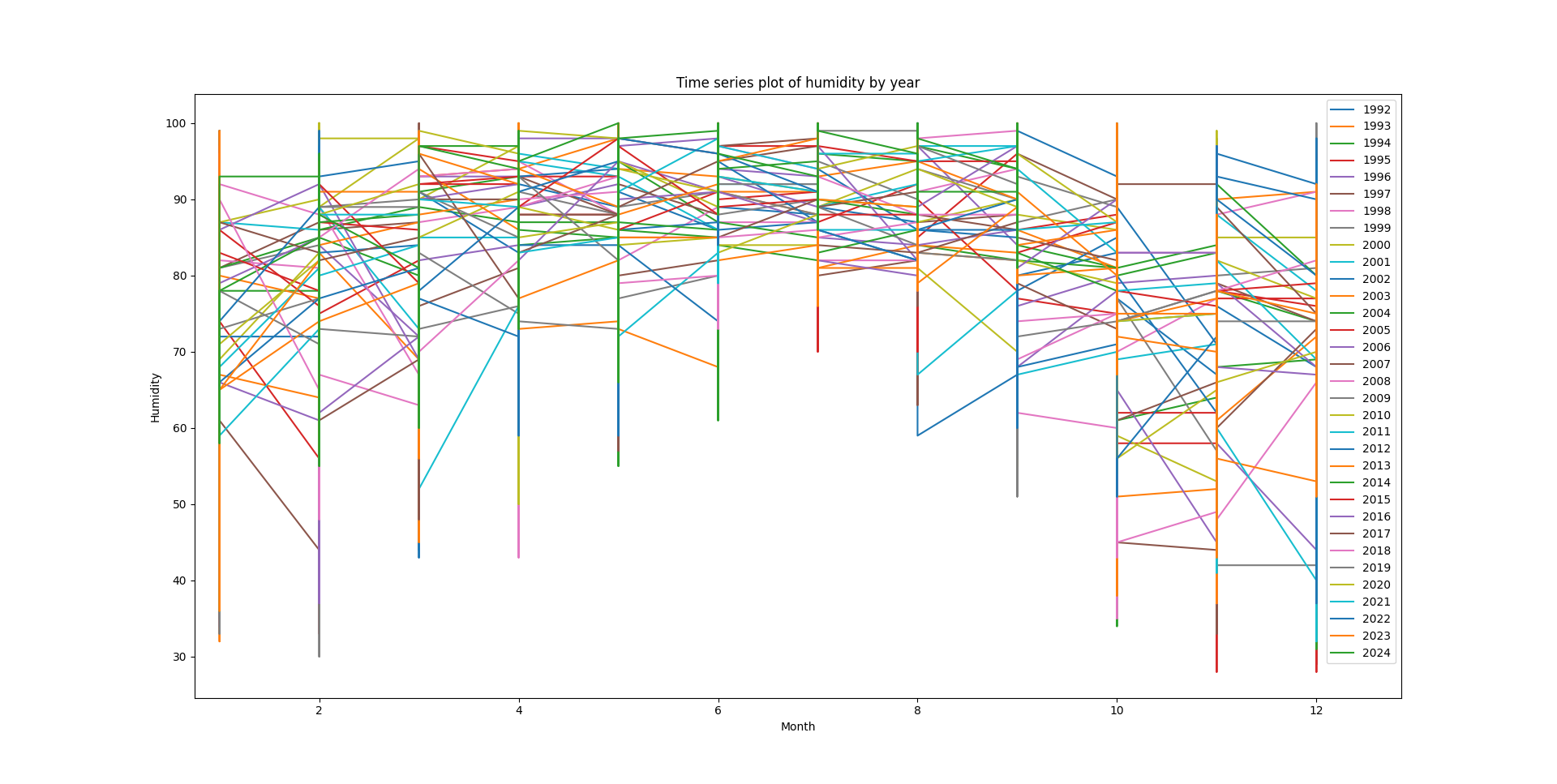


Figure 11. Humidity distribution by year

After accumulating all the data, we obtained the statistical distribution of humidity data as follows:

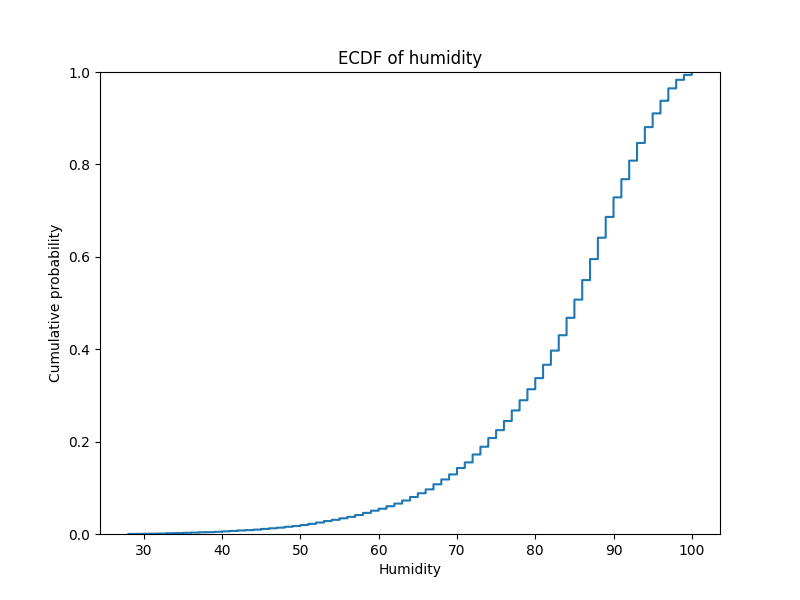


Figure 12. Humidity accumulation distribution

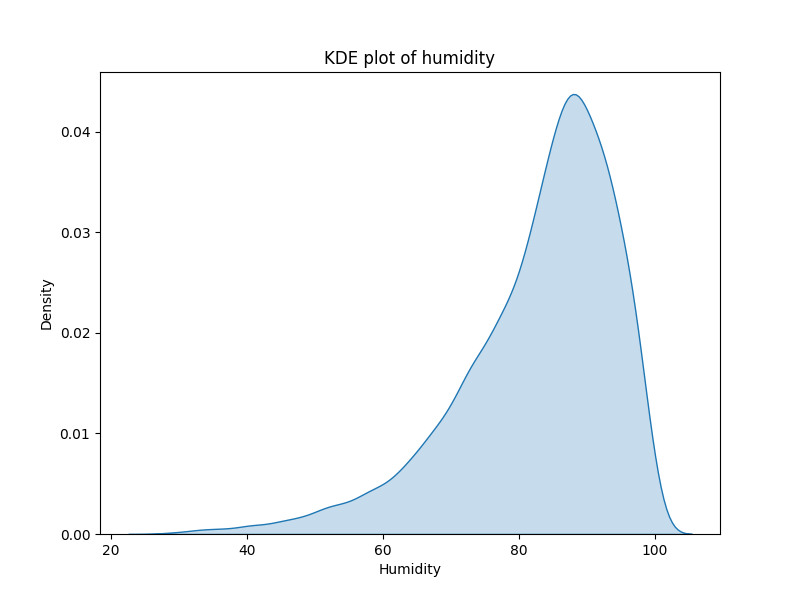


Figure 13. Humidity probability distribution

|  |  |
| --- | --- |
| Mean Humidity | Standard Deviation of Humidity |
| 82.65597489758564 | 11.679318452222946 |

Table 4. Humidity distribution statistics

1. Fire Fighter Data

Firefighter average extinguishing time will vary due to a variety of factors, including fire size, fire type (surface fire, crown fire or underground fire), meteorological conditions, terrain complexity, available resources, and the experience level of the fire brigade. According to the National Interagency Fire Center (NIFC) statistics, fires can be roughly divided into small, medium and large fires, and their extinguishing times are also different. For small fires, covering an area of about a few hectares, firefighters can usually control the fire within a few hours to a day. Medium-sized fires cover an area of tens to hundreds of hectares, requiring more resources such as helicopters, fire-fighting aircraft, and experienced firefighters, and usually take 1 to 7 days to fully control. In areas with drought and strong winds, large fires covering an area of more than a few thousand hectares may take several weeks or even months to fully control, and such fires can generally only be extinguished passively, such as waiting for the weather to cool down, and the burning area to naturally disappear.

Different firefighter teams, different countries and regions have different fire extinguishing efficiency. China's forest firefighting efficiency is relatively high, and small forest fires can usually be basically controlled within 1 to 2 days. However, large-scale mountain fires (such as Yunnan, Sichuan, etc.) require several weeks. The U.S. firefighter team also has a high level of efficiency in fire extinguishing, but due to the vast territory of the United States and frequent mountain fires, it also takes a long time to extinguish large-scale mountain fires. Australia, due to its vast land and sparse population, requires a long time to extinguish large-scale mountain fires. In Canada, the duration of fires is significantly affected by climate and fuel dryness. In the past few decades, due to climate change, the frequency and duration of large fires have increased significantly.

The initial suppression of forest fires is the most critical. If the fire can be controlled at the beginning of the fire, the spread of the fire will be greatly restricted. Therefore, the rapid response and efficient suppression of firefighters are very important. In the early stages of a fire, firefighters usually use aerial resources such as helicopters, fire-fighting aircraft, and ground firefighters to control the fire by blocking the fire line and extinguishing the fire. After the fire expands, firefighters usually use more resources, such as more helicopters, fire-fighting aircraft, fire trucks, and more firefighters, to control the fire by more blocking the fire line and extinguishing the fire.

## Simulation procedures

We implemented a simple discrete event simulation system in JavaScript. Discrete event simulation requires maintaining an ordered queue of events (sorted by time). Each time an event is processed from the queue, the system state is updated based on the type and content of the event, and new events are added to the queue. We used a simple “**EventQueue**” class to implement this functionality.



Figure 14: Discrete Event Simulation Structure

We used a two-dimensional grid model to simulate a burnable area, where each cell has three burning states: unburned, burning, and burned out. The burning state of each cell changes over time, and when a cell is burning, it spreads flames to neighboring cells. The behavior of flame propagation is influenced by Monte Carlo simulations (wind speed, wind direction, etc.). We implemented the "**Cell**" and "**Grid**" classes to manage grid behavior and store grid data.

Next, we will introduce the fire spread model in detail. The fire spread model "FireSpreadEvent" inherits from the "Event" class and is a specific class that describes the behavior of fire spread. This class depends on the probability distribution parameters obtained from real-world statistics passed in by the environment class and simulates environmental parameters through Monte Carlo simulations.

1. **Monte Carlo Distribution**

"**MonteCarloDistribution**" provides a unified distribution abstraction interface, allowing different distribution classes to have consistent sampling and description methods. This class is an abstract base class that defines the basic interface of distribution classes in Monte Carlo simulations. It cannot be instantiated directly and provides a common method structure, requiring inheriting subclasses to implement the following two core methods:

1. "**sample()**": an abstract method that generates random samples that conform to a specific distribution. Subclasses must implement this method to generate random numbers that conform to their specific distribution.

2. "**describe()**": an optional description method used to output distribution information. Subclasses can choose to implement this method to return the characteristics of the distribution.

"**GaussianDistribution**" uses the Box-Muller transform to sample from a normal distribution, suitable for simulating continuous symmetric distribution scenarios. This class inherits from "**MonteCarloDistribution**" and represents a Gaussian (normal) distribution. It uses the Box-Muller transform to generate random numbers from a normal distribution. The distribution is symmetric around the mean, and sample points are mainly concentrated near the mean. The "mean" parameter controls the center position of the distribution, while the "**standardDeviation**" parameter determines the width of the distribution (the degree of data dispersion). The larger the standard deviation, the more dispersed the distribution; the smaller the standard deviation, the more concentrated the distribution.

"**DiscreteDistribution**" uses a cumulative distribution to sample from discrete events by defining the probabilities of different events to simulate discrete distribution scenarios. This class also inherits from "**MonteCarloDistribution**" and is used to represent discrete distributions. It accepts a probability mapping that associates a set of discrete events with their occurrence probabilities. The "\_**buildCumulative()**" method is an internal method used to calculate the cumulative distribution function (CDF). By summing the probabilities of each event, it generates a cumulative probability array for the "**sample()**" method to use. The "sample()" method generates random samples that conform to the discrete distribution. By generating a random number and comparing it with the cumulative distribution array, it returns the corresponding event. This method ensures that the sampling process conforms to the specified probability distribution. The "**describe()**" method returns a string description containing events and their probabilities for a quick understanding of the distribution characteristics.

1. **Fire Spread Event**

"**FirespreadEvent**" is an event used to simulate the spread of a fire. The core logic includes fire spread, updating the remaining burning time, and calculating the impact of wind on the fire. When the "**execute()**" method is called, the following steps are performed:

1) The current cell state is obtained. If the cell is in an "unburned" state, it is marked as "burning," and the burning time is set based on the wind speed.

2) The neighboring cells are traversed, and based on the wind direction and the probability of fire spread, it is determined whether the fire will spread.

3) If the burning time of the current cell is exhausted, a "**BurnOutEvent**" is generated; otherwise, the next "**FireSpreadEvent**" event is generated.

The "**calculateSpreadProbability()**" method calculates the probability of fire spreading to neighboring cells, considering the wind direction and fire resistance. The results are cached to improve efficiency. The "**forEachNeighbor()**" method is used to traverse and process neighboring cells for the spread logic. The "**getDirectionFactor()**" method calculates the angle between the wind direction and the neighboring direction to adjust the spread probability.

"**BurnOutEvent**" is an event that inherits from the "Event" class and is used to represent a cell in the grid that has completed the burning process at a specified time and is in the "burned" state. It receives the time, grid object, and the coordinates (x and y) of the target cell as parameters during initialization. When the event is executed (the "**execute()**" method is called), it updates the state of the target cell to "burned" and removes the "burning" state from the grid (by calling the "**grid.removeBurningCell()**" method), thereby completing the management of the cell's burning lifecycle.

1. Fire Fighter Model

In the fire simulation system, the "Firefighter" class plays a central role in simulating firefighter behavior. Its responsibilities include extinguishing fires, setting isolation zones, and formulating action strategies based on resource constraints. The class's design covers property definitions, behavioral logic, and grid interactions, allowing it to dynamically respond to complex situations in the simulation environment. Firefighters have attributes such as stamina, water level, position, movement direction, and target cells. Combined with mechanisms such as freezing status and extinguishing time, the class ensures that the simulation can realistically represent firefighter behavior.

In terms of behavior design, the "Firefighter" class implements key functions through method modularization. For example, the "move()" method allows firefighters to move in the grid and avoids invalid operations through target validity checks. The core method "act()" selects the corresponding behavior based on the current position of the firefighter: triggering extinguishing events on burning cells; probabilistically setting isolation zones on unburned cells; and searching for the fire source and moving towards it in other cases. This process fully considers resource consumption and freezing status constraints, making firefighter behavior both intelligent and environmentally influenced. To achieve dynamic interaction, the class also provides functions to set isolation zones and move towards the fire source, simulating the operational methods of firefighters in actual tasks, such as randomly adjusting paths to avoid single-mode movement.

The "Firefighter" class's functional design focuses on detail and realism. The operation of setting isolation zones is implemented by triggering events and consumes a certain amount of stamina, enhancing the simulation depth of resource management. The path selection is optimized by vector calculation and random offset during movement. In addition, the method of searching for the nearest burning cell supports filtering a specific number of targets, providing flexibility for the firefighter's action strategy. In general, firefighter behavior can be summarized into several categories: extinguishing fires, setting fire isolation zones, intelligent path movement, and water and stamina management. These behaviors enable firefighters to complete complex tasks in a dynamic environment and enhance the overall expressiveness of the simulation through grid system interaction.

Firefighters follow multiple strategies in the simulation to extinguish fires, prevent fire spread, and optimize their resource management. These strategies are based on the firefighters' environmental awareness and are implemented through logical reasoning and target optimization.

1. Firefighters' primary task is to extinguish fires. When standing on a burning cell, firefighters trigger a series of events to start extinguishing the fire. During this process, firefighters consume water and time to complete the extinguishing, ensuring that the fire source is quickly controlled. If the current cell is already occupied by another firefighter, the firefighter will move to another fire source location first. This strategy reflects the urgency of firefighting.
2. If a firefighter is standing on an unburned cell near the burning area, there is a certain probability that they will set up a firebreak. This strategy limits the spread of the fire by marking the cell as non-flammable, especially in large fire scenarios. The establishment of firebreaks depends on the firefighter's stamina resources, ensuring that the defensive effect is maximized within the available resources.
3. When firefighters are not on a burning cell, they search for the nearest fire source and move towards it. This strategy is based on the shortest path principle, combined with random offsets to avoid the monotony of path planning. Firefighters can also randomly select from multiple targets to ensure comprehensive coverage of the fire. This strategy optimizes the efficiency of firefighter actions while avoiding repeated movements.
4. Firefighter actions are limited by stamina and water level. Each movement consumes stamina, while extinguishing fires consumes both water and stamina. When resources are insufficient, firefighters will prioritize low-consumption actions (such as movement) to avoid wasting limited resources. This strategy ensures the continuity of firefighters' actions and provides an interface for supply logic in complex scenarios.
5. When firefighters are in a frozen state (e.g., due to external events), they stop all actions. This strategy logically ensures that the firefighters' state is linked to environmental events and provides room for introducing more complex scenarios (such as poisoning, obstacles, etc.) in the simulation.

With these strategies, firefighters can quickly adapt to changes in a dynamic environment, balance firefighting and resource management, and effectively limit the spread of fires. The design of these behavioral strategies provides a reliable foundation for complex fire simulations and has flexible scalability.

The design highlights the combination of modularization and event-driven. The implementation of extinguishing and isolation zone setting events decouples behavior logic from environmental changes, improving code scalability. The complementary design of randomness and strategic behavior adds uncertainty and decision challenges to the simulation. The resource consumption model of stamina and water level provides more realistic constraints and a basis for simulation decision design. In addition, flexible target selection logic lays the foundation for extending firefighter behavior, such as introducing cooperative firefighting or dynamic priority adjustment strategies in the future.

Future improvements include optimizing the search algorithm for target cells, such as a distance calculation model based on Euclidean or Manhattan distance, to improve efficiency and accuracy. Additionally, firefighter behavior patterns can be expanded, such as implementing cooperative firefighting through dynamic priority allocation. Furthermore, introducing environmental factors such as wind speed and terrain conditions will further enrich the complexity of firefighter decisions, making the simulation closer to real-world scenarios.

## Productivity and cost estimation and analysis

## Results and conclusions

1

## Appendix:

1