# Database Management Homework 1

Po-Yen Chu

## 1 Question 1

(a) According to the question, Table 1 is provided.

| | date | temp | | | cnt |
|---|---|---|---|---|---|
| weathersit | count | mean | median | std | mean |
| 1 | 463 | 20.973542 | 21.390000 | 7.837816 | 4876.786177 |
| 2 | 247 | 19.285263 | 18.790000 | 6.854406 | 4035.862348 |
| 3 | 21 | 17.770476 | 18.040000 | 5.390388 | 1803.285714 |

Table 1: Statistics of `Bike.xlsx` (group by `weathersit`)

Here's the code for question 1:

```python
import pandas as pd

data = pd.read_excel('Bike.xlsx')

# group by weathersit
result = data.groupby('weathersit').agg(
    {'date': 'count',
     'temp': ['mean', 'median', 'std'],
     'cnt': 'mean'})

# convert to latex
print(result.to_latex(column_format='l|c|ccc|c'))
```

(b) According to the question, Table 2 is provided.

| | | date | temp | | | cnt |
|---|---|---|---|---|---|---|
| weathersit | workday | count | mean | median | std | mean |
| 1 | 0 | 156 | 20.193077 | 20.095000 | 8.009583 | 4587.269231 |
| | 1 | 307 | 21.370130 | 21.940000 | 7.732073 | 5023.902280 |
| 2 | 0 | 70 | 18.993000 | 17.715000 | 6.926531 | 3936.828571 |
| | 1 | 177 | 19.400847 | 19.030000 | 6.841984 | 4075.028249 |
| 3 | 0 | 5 | 15.592000 | 16.260000 | 6.135313 | 1815.400000 |
| | 1 | 16 | 18.451250 | 18.535000 | 5.160190 | 1799.500000 |

Table 2: Statistics of `Bike.xlsx` (group by `weathersit` and `workday`)

Here's the code for question 2:

```
result = data.groupby(['weathersit', 'workday']).agg({'
    date': 'count', 'temp': ['mean', 'median', 'std'], 'cnt
    ': 'mean'})
# convert to latex
print(result.to_latex(column_format='ll|c|ccc|c'))
```

(c) According to the question, Table 3 is provided.

| weathersit | correlation coefficient |
|---|---|
| 1 | 0.622190 |
| 2 | 0.644899 |
| 3 | 0.606836 |

Table 3: Correlation coefficient of `temp` and `cnt` (group by `weathersit`)

Figure 1 presents the scatter plots of `temp` and `cnt` among the three groups.

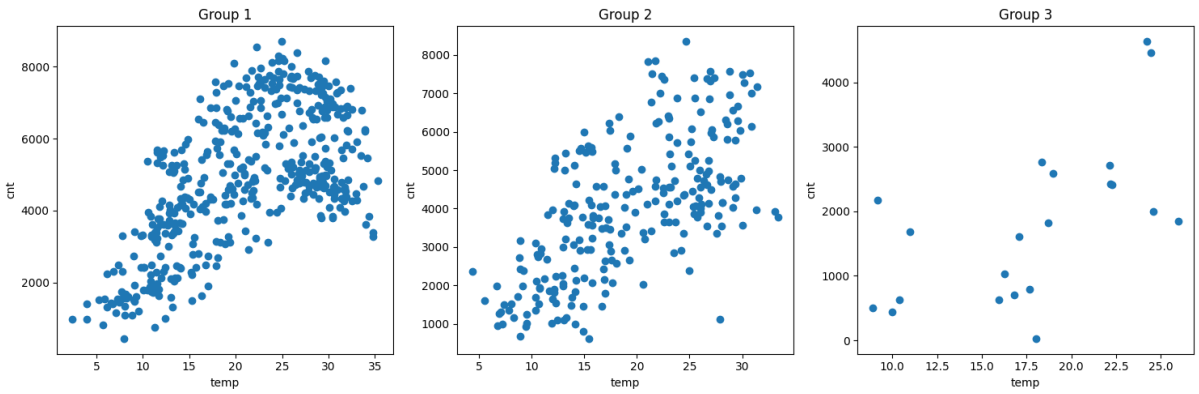


Figure 1: Scatter plots of `temp` and `cnt` (group by `weathersit`)

Here's the code for question 3:

```
result = data.groupby('weathersit')

# calculate correlation between temp and cnt
correlation = result.apply(lambda x: x['temp'].corr(x['cnt
    ']))

print(correlation.to_latex())

# plot scatter plots (in subplots) for the three groups
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 3, figsize=(15, 5))
for i, (key, group) in enumerate(result):
    ax[i].scatter(group['temp'], group['cnt'])
    ax[i].set_title('Group ' + str(key))
    ax[i].set_xlabel('temp')
    ax[i].set_ylabel('cnt')
plt.tight_layout()
plt.show()
```

# Question 2

(a) To implement a class `Queue` by inheriting the `LinkedList` class, we can extend the existing functionalities. Here is how each function would be handled:

(1) `constructor`: Initialize the queue by calling the constructor of the `LinkedList`.

(2) `enqueue(element)`: Use the `insert()` method of the `LinkedList` to add an element to the end of the list.

(3) `dequeue()`: Remove the first element from the list by calling the `delete()` function with the index of the head (position 0).

(4) `isEmpty()`: Return whether the list is empty by checking if the `head` of the `LinkedList` is `NULL`.

(5) `getQueueLength()`: Call the method that returns the size of the `LinkedList`, which traverse until the `next` attribute of iterator become `NULL` and return the iteration count.

In this way, the class `Queue` will act like a FIFO (First In First Out) queue, where elements are added at the end and removed from the front.

(b) After inserting the numbers 1, 5, 16, 18, 13, 7, 19, 2, 27, 13, 35, 4 into a max heap, the resulting heap structure would look like this:

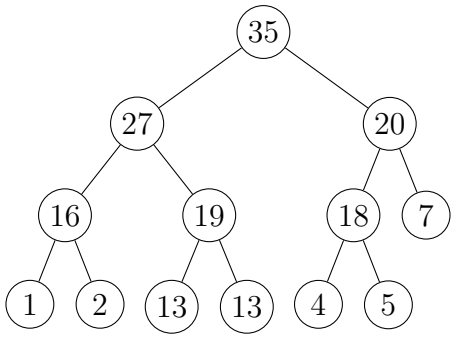


Figure 2: Final max heap for question (b)

This is the final max heap, where each parent node is greater than its children.

(c) To implement a max heap using a one-dimensional array or list, the position of each element in the tree is stored at specific indices in the array. For a node at index $i$:

(1) The left child is stored at index $2i + 1$.

(2) The right child is stored at index $2i + 2$.

(3) The parent of the node is stored at index $\left\lfloor \frac{i-1}{2} \right\rfloor$.

For the final max heap from part (b), the corresponding array representation would be:

| **Index:** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Value:** | 35 | 27 | 20 | 16 | 19 | 18 | 7 | 1 | 2 | 13 | 13 | 4 | 5 |

Table 4: Max heap from part (b) represented in one-dimensional array

Each element in the heap corresponds to its correct position in the array.

# Question 3

(a) The course usually covers topics such as process management, memory management, file systems, input/output systems, and system security. It also often includes inter-process communication, synchronization, deadlocks, and virtualization.

(b) The main difference is that a process is an independent program with its own memory space, while a thread is a smaller execution unit within a process that shares the same memory space with other threads in the same process. Threads allow parallel execution within a single process.

(c) Contiguous allocation is a method where each file is stored in a single contiguous block of memory on the disk. The advantage is that it provides fast sequential access. However, it suffers from fragmentation issues and difficulties in dynamically resizing files, which can lead to wasted space or the need for complex memory management.

(d) Virtual memory allows a system to use disk space as an extension of RAM, enabling the execution of programs that require more memory than physically available. Its primary purpose is to provide the illusion of a large memory space, manage multitasking more efficiently, and isolate memory spaces of different processes.