**Content**

[Task 5.1](#Task_5_1)

[Task 5.1 Result](#Task_5_1_result)

[Task 5.1 Interpretation](#Task_5_1_interpretation)

[Task 5.2](#Task_5_2)

[Task 5.2 Result](#Task_5_2_result)

[Task 5.2 Interpretation](#Task_5_2_interpretation)

[Task 6](#Task_6)

[Task 6 Result](#Task_6_result)

[Task 6 Interpretation](#Task_6_interpretation)

[Task 7.1](#Task_7_1)

[Task 7.1 (i) Formulas](#Task_7_1_formula)

[Task 7.1 (ii) Result](#Task_7_1_result)

[Task 7.1 (iii) Discussion](#Task_7_1_discussion)

[Task 7.2](#Task_7_2)

[Task 7.2 Result](#Task_7_2_result)

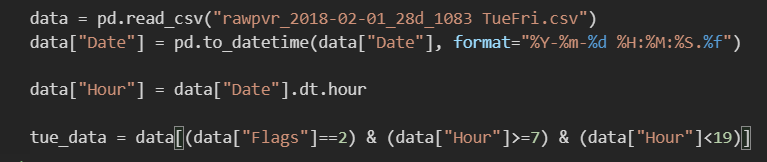
[Task 7.2 Comparison](#Task_7_2_comparison)

**Task 5.1**

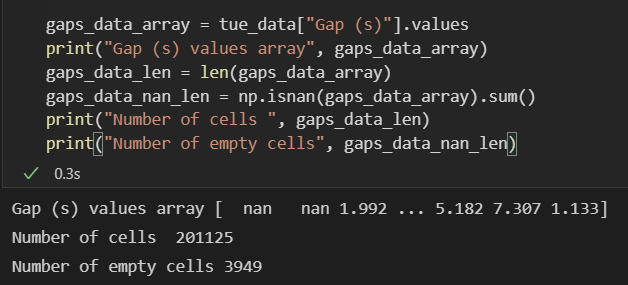
*Python is used for this task with Pandas and NumPy libraries.*

*The columns [Flags] and [Flag Text] are already obtained in Task 1, so they are used directly for this task.*

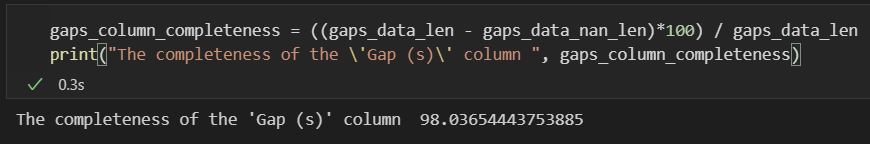
First read the csv file, and to facilitate further operations, the [Date] column is converted to a datetime type. Create a new column called [Hour], which store the specific hour number corresponding to column [Date]. Select the data that meets the criteria (Tuesday between 7:00 and 19:00) and store it in the tue\_data variable.



***To compute the column completeness of [Gap (s)],*** choose [Gap (s)] column from tue\_data, get the values of each cell and make them as an array. Then get the length of this array, which is the number of cells. By applying np.isnan().sum() to the array, the length of cells that is empty is obtained.

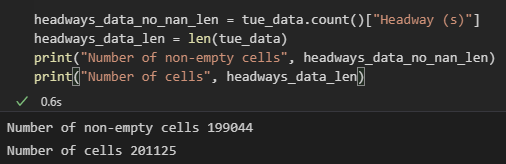


Finally, use number of cells minus empty cells to get the number of non-empty cells, then divided by number of cells to obtain the column completeness of [Gap (s)].

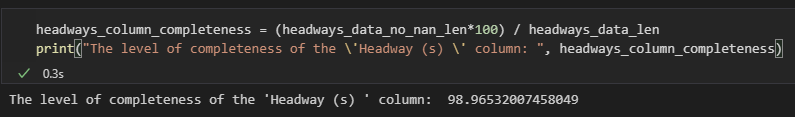


***Result***: Completeness of [Gap (s)]: 98.03654443753885

***To compute the column completeness of [Headway (s)],*** another function count() is used and applied to tue\_data, it return the number of cells without counting nan cells.



Finally using number of non-empty cells divided by number of cells to obtain the column completeness of [Headway (s)].



***Result***: Completeness of [Headway (s)]: 98.96532007458049

***Task 5.1 Result Summary (2 Decimal):***

|  |  |  |
| --- | --- | --- |
|  | Gap (s) | Headway(s) |
| Column Completeness | 98.04 | 98.97 |

***Task 5.1 Interpretation:***

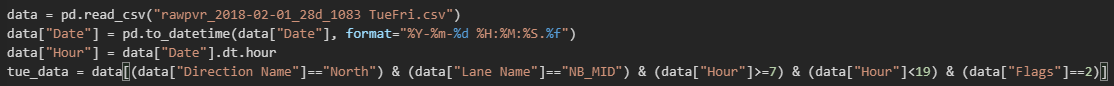
Completeness is one of the characteristics of data quality. both [Gap (s)] and [Headway (s)] have a column data completeness above 98, indicating that they have very high data completeness and that most of the data is recorded. However, this also means that there is some data that is still empty and data cleaning can be performed to improve data quality and completeness.

**Task 5.2**

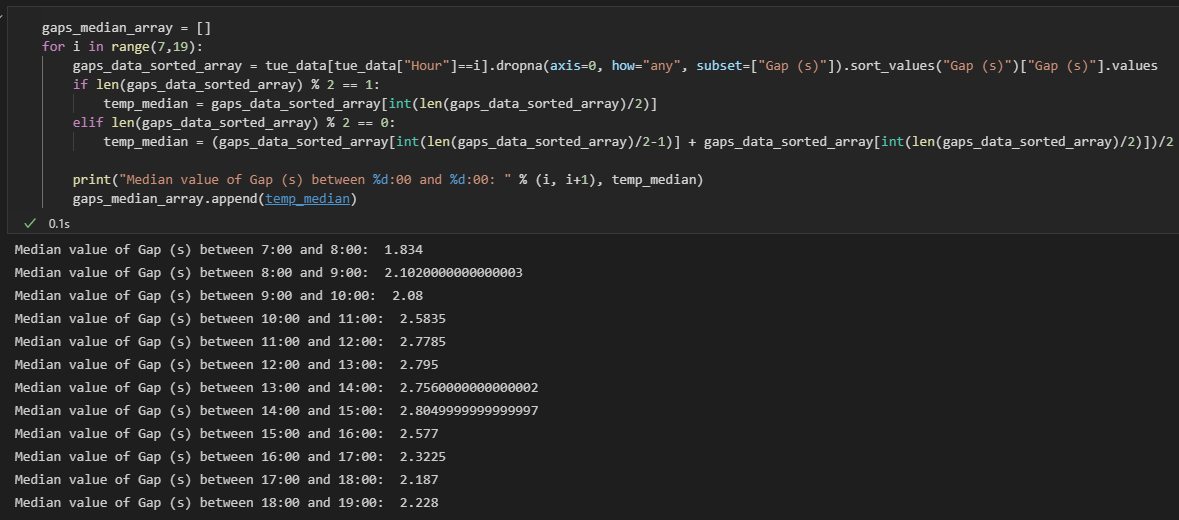
*Python is used for this task with Pandas and NumPy libraries.*

*The columns [Flags] and [Flag Text] are already obtained in Task 1, so they are used directly for this task.*

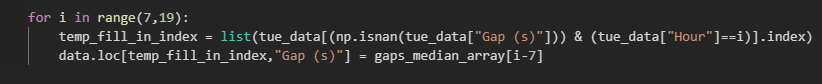
First read the csv file, and to facilitate further operations, the [Date] column is converted to a datetime type. Create a new column called [Hour], which store the specific hour number corresponding to column [Date]. Select the data that meets the criteria (Tuesday between 7:00 and 19:00, NB\_MID lane) and store it in the tue\_data variable.



***To fill the missing values of column [Gap (s)]***, it is necessary to sort the data of each hour in [Gap (s)] to get the median values of each hour. First approach is by doing this manually. Define a for loop from 7 to 19 that represent each hour to get 12 arrays of [Gap (s)] of each hour, and sort them. If array length is odd then the median value is the value at the position of half of the array, otherwise the median value is the two values at the position of half of the array values added then divided by 2.



Finally, get the row index of the missing values, then these indexes are used to locate and fill in with the median value corresponding to specific hour.

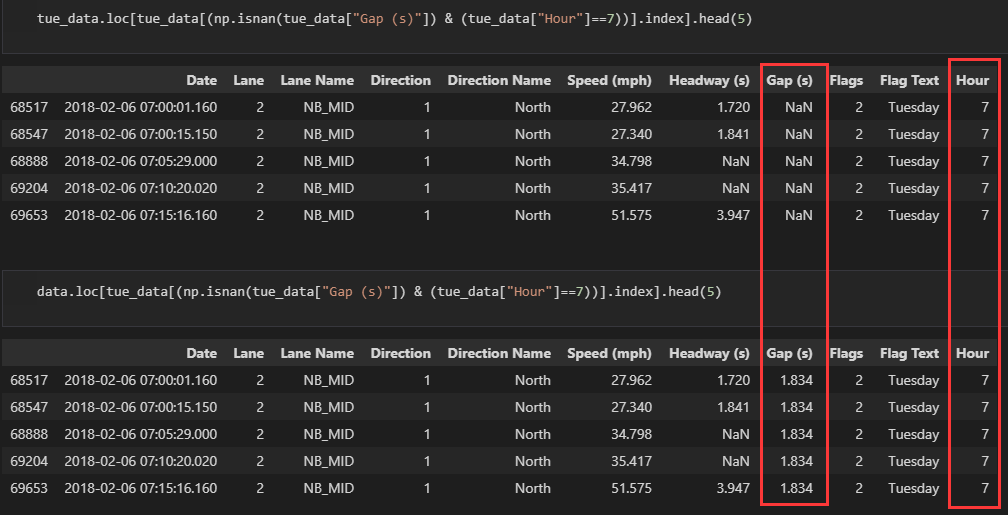


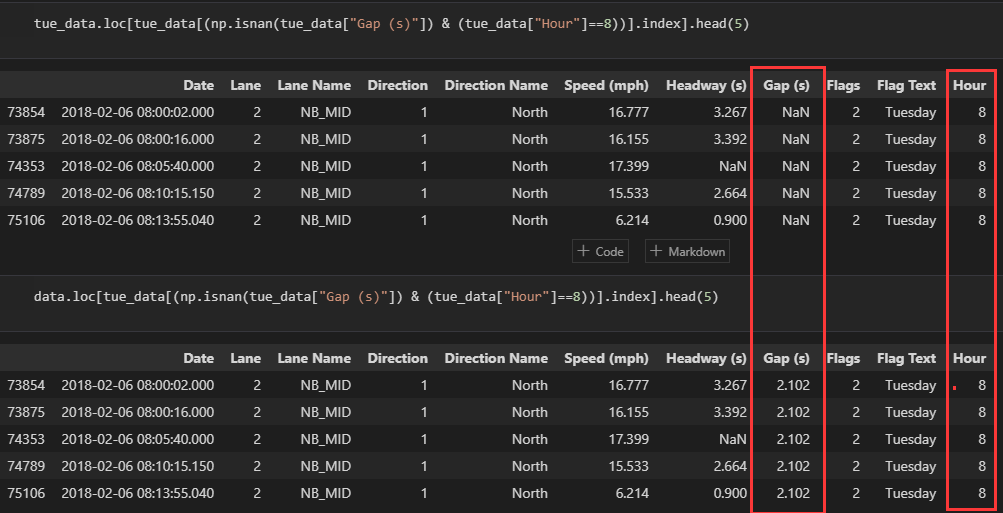
***Result***:

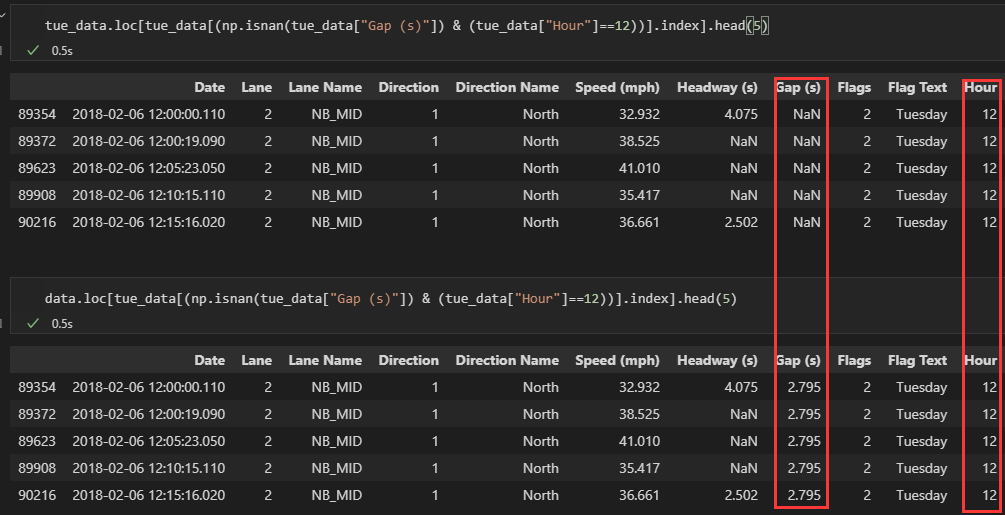
The table of [Gap (s)] median values of each hour:

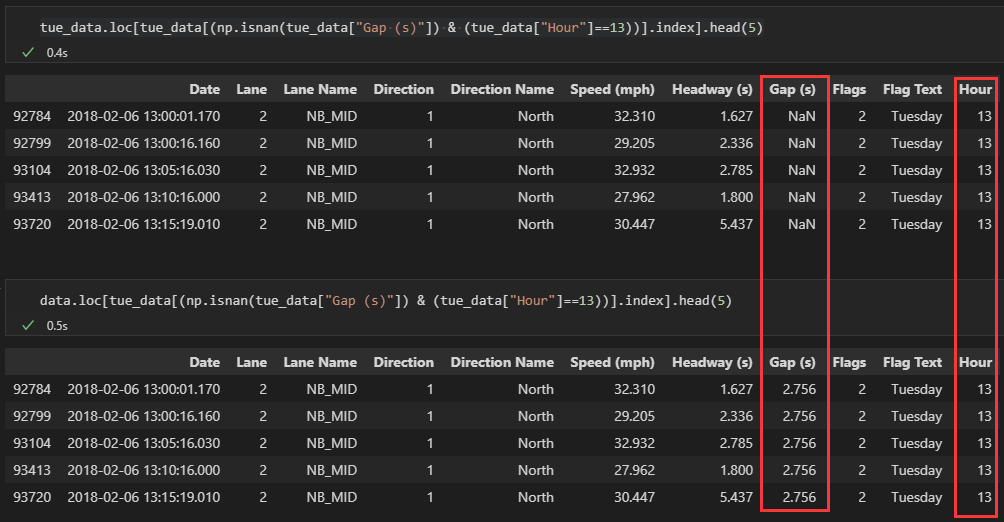
|  |  |  |  |
| --- | --- | --- | --- |
| Hour | Gap (s) | Hour | Gap (s) |
| 7 | 1.834 | 13 | 2.7560000000000002 |
| 8 | 2.1020000000000003 | 14 | 2.8049999999999997 |
| 9 | 2.08 | 15 | 2.577 |
| 10 | 2.5835 | 16 | 2.3225 |
| 11 | 2.7785 | 17 | 2.187 |
| 12 | 2.795 | 18 | 2.228 |

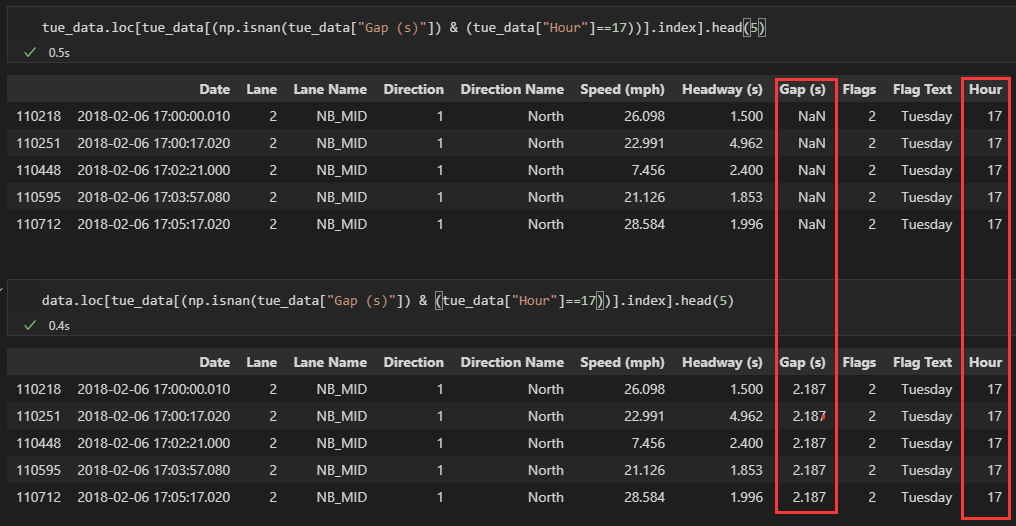
Here are some example screenshots ([Hour]=7,8,12,13,17,18) of first 5 rows before and after filling in.

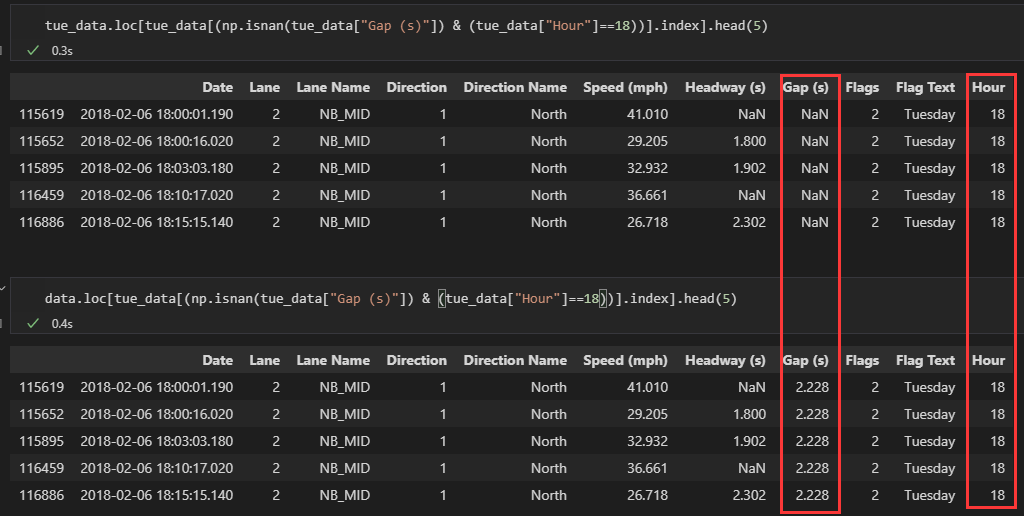




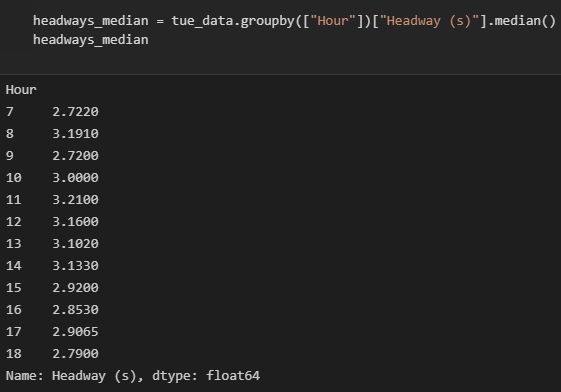




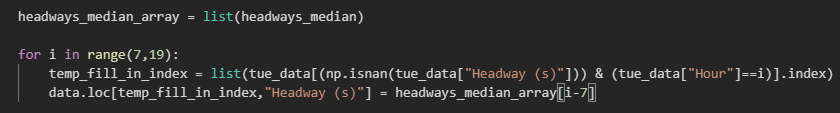




***To fill the missing values of column [Headway (s)]***, another approach is first group by [Hours], then simply applying the median() function on column [Headway (s)]



Finally, use the same way as described above to fill the median values into missing value cells.

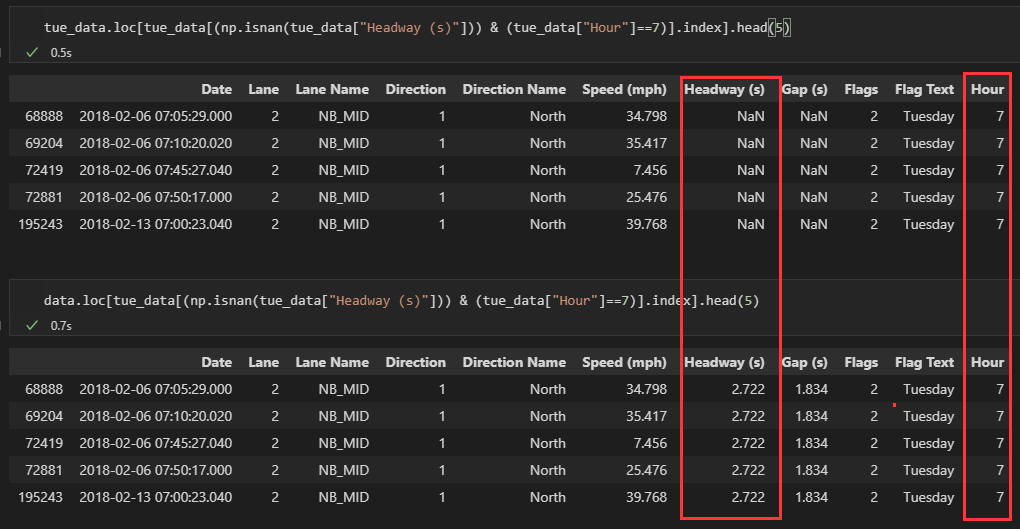


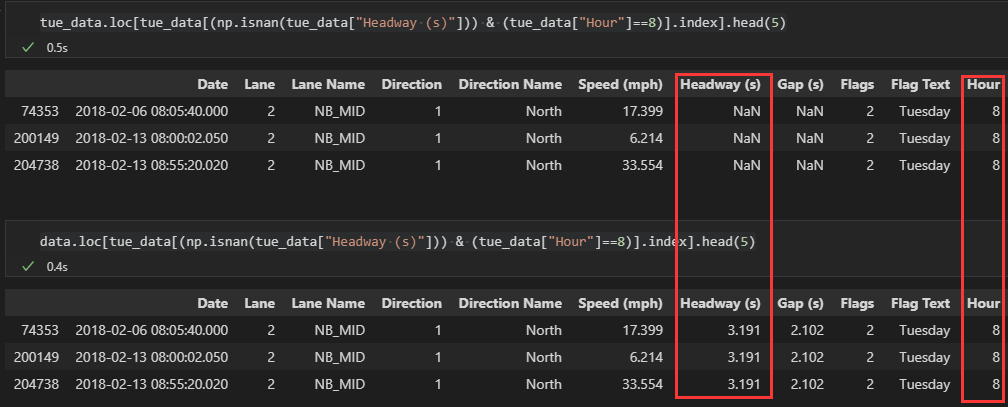
***Result***:

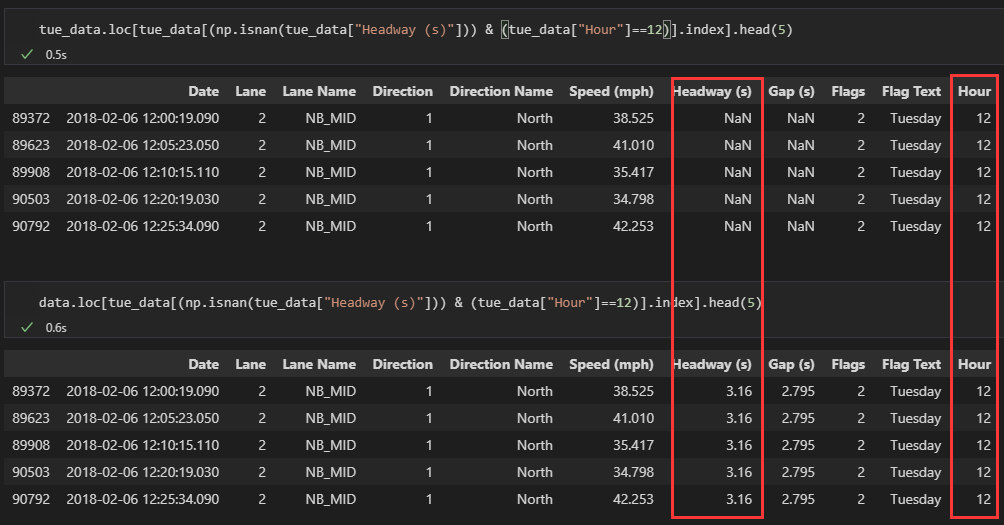
The table of [Headway (s)] median values of each hour:

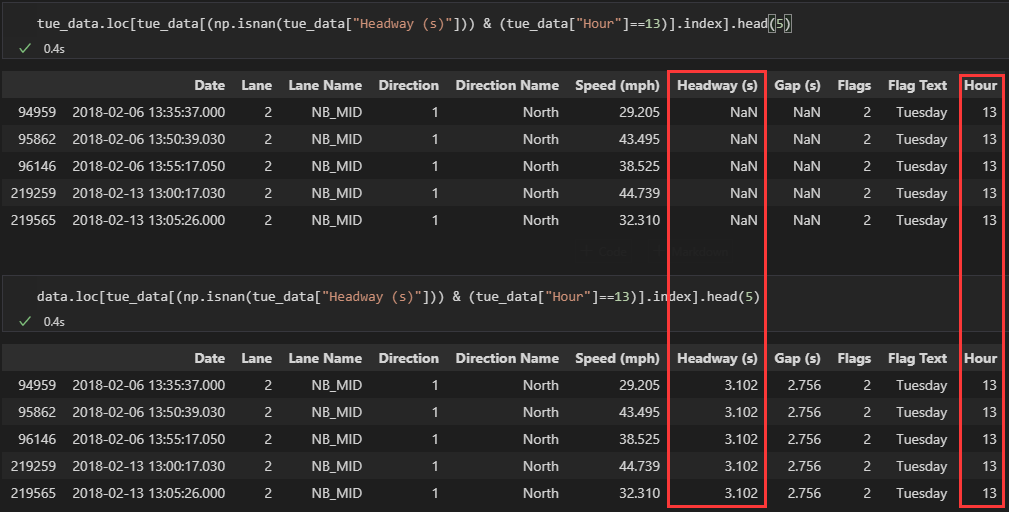
|  |  |  |  |
| --- | --- | --- | --- |
| Hour | Headway (s) | Hour | Headway (s) |
| 7 | 2.7220 | 13 | 3.1020 |
| 8 | 3.1910 | 14 | 3.1330 |
| 9 | 2.7200 | 15 | 2.9200 |
| 10 | 3.0000 | 16 | 2.8530 |
| 11 | 3.2100 | 17 | 2.9065 |
| 12 | 3.1600 | 18 | 2.7900 |

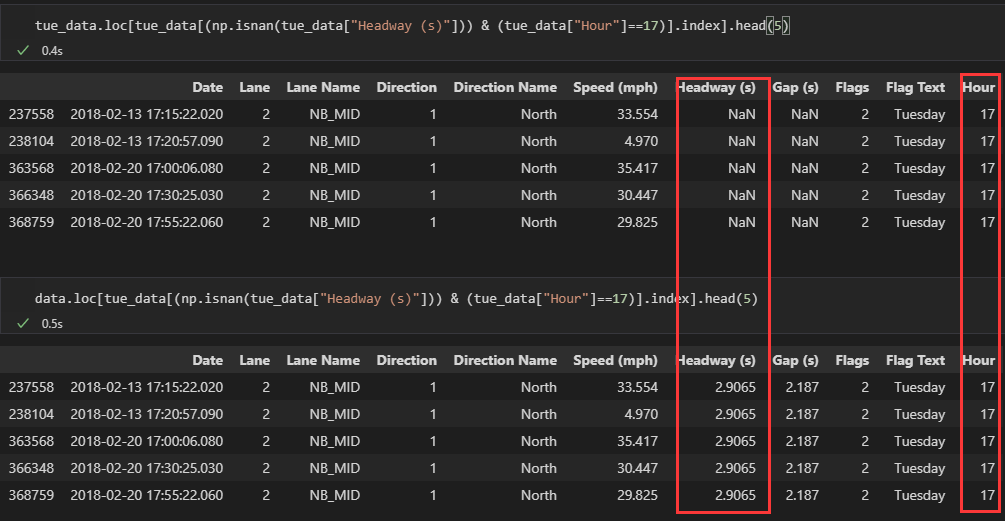
Here are some example screenshots ([Hour]=7,8,12,13,17,18) of first 5 rows before and after filling in.

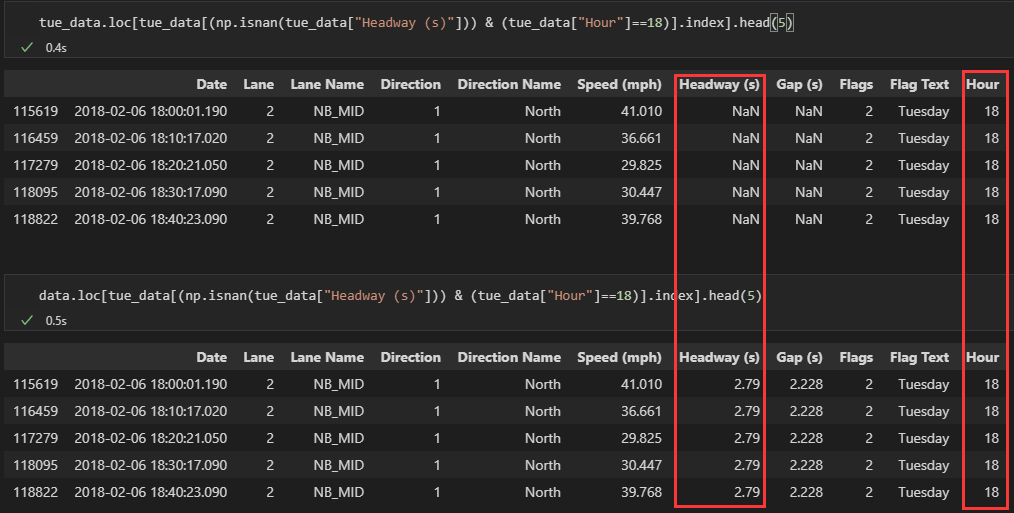












***Task 5.2 Result Summary (4 Decimal):***

Median values table:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hour | Gap (s) | Hour | Gap (s) | Hour | Headway (s) | Hour | Headway (s) |
| 7 | 1.834 | 13 | 2.7560 | 7 | 2.7220 | 13 | 3.1020 |
| 8 | 2.1020 | 14 | 2.8050 | 8 | 3.1910 | 14 | 3.1330 |
| 9 | 2.0800 | 15 | 2.5770 | 9 | 2.7200 | 15 | 2.9200 |
| 10 | 2.5835 | 16 | 2.3225 | 10 | 3.0000 | 16 | 2.8530 |
| 11 | 2.7785 | 17 | 2.1870 | 11 | 3.2100 | 17 | 2.9065 |
| 12 | 2.7950 | 18 | 2.2280 | 12 | 3.1600 | 18 | 2.7900 |

Result example screenshot is above

***Task 5.2 Interpretation:***

In data cleaning, it is a common operation to use the median value to fill in the empty cells. Similarly, the average value can also be used. More advanced, machine learning methods can be used to predict missing values.

From these median values, it can be predicted that the busiest time period is between 7:00 and 8:00, because the values of gap and headway are very small, indicating that the distance between the front and rear cars is very short.

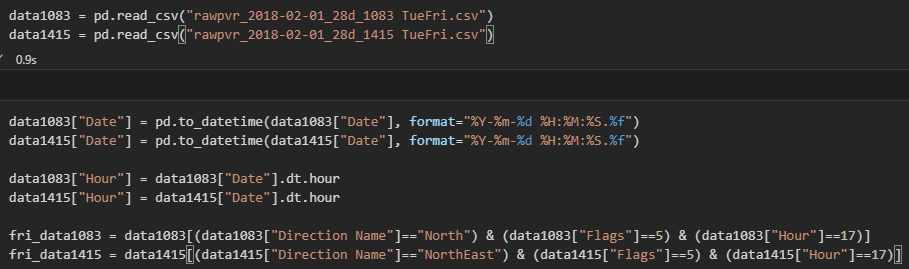
For the filled table, the last [Hour] column should be deleted when writing output file, because it is not existed in the original table, it is just added for the convenience of operation.

**Task 6**

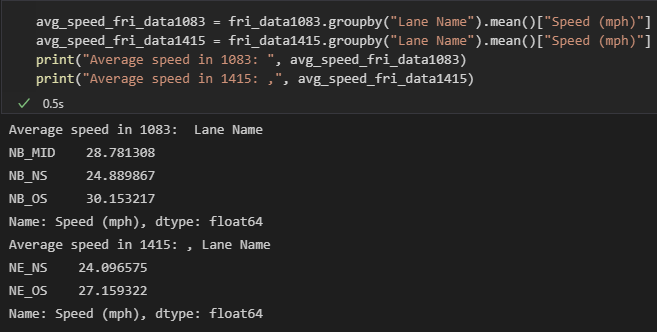
*Python is used for this task with Pandas and NumPy libraries.*

*The columns [Flags] and [Flag Text] are already obtained in Task 1, so they are used directly for this task.*

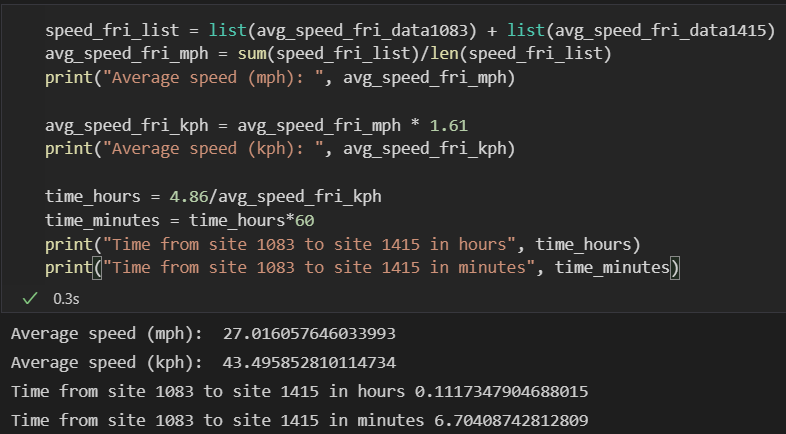
First read the “1083” and “1415” csv files, and to facilitate further operations, each of the [Date] column is converted to a datetime type. Create a new column called [Hour], which store the specific hour number corresponding to column [Date]. Select the data that meets the criteria (for 1083: North direction, Friday between 17:00 and 18:00; for 1415: NorthEast direction, Friday between 17:00 and 18:00) and store them in the fri\_data1083 and fri\_data1415 variables respectively.



Then compute the average speed of each lane



Finally, compute a total average speed by using these five lanes average speed. As this speed is miles per hour (27.02 mph), convert it to kilometers per hour (43.49 kph) by multiply 1.61. Using distance 4.86km divided by kph speed, the time from site 1083 to site 1415 in hour is obtained (0.11 hours). The time from site 1083 to site 1415 in minutes can obtained by time in hour multiply by 60. (6.70 minutes)



***Task 6 Result (3 Decimal)***: Journey time: 6.704 minutes

***Task 6 Interpretation:***

The mean is used to describe the central tendency of the distribution of data. The average of five lanes is used here to estimate the travel time from site 1083 to site 1415. This shows that under normal circumstances, most vehicles take about 6.75 minutes. This can give a time estimate for travelers. Also, it can judge whether this road is congested by comparing the current travel time with the average time.

**Task 7.1**

*Python is used for this task with Pandas and NumPy libraries.*

*The columns [Flags] and [Flag Text] are already obtained in Task 1, so they are used directly for this task.*

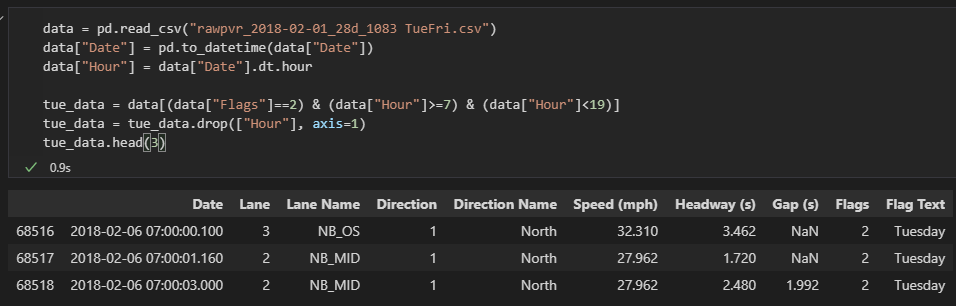
**(i)** Here defined two row completeness formulas

First is:

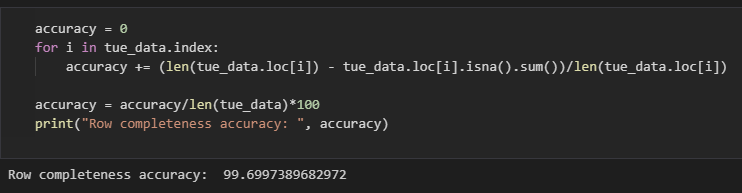
Second is:

**(ii)** First read the csv file, and to facilitate further operations, the [Date] column is converted to a datetime type. Create a new column called [Hour], which store the specific hour number corresponding to column [Date]. Select the data that meets the criteria (Tuesday between 7:00 and 19:00) and store it in the tue\_data variable.

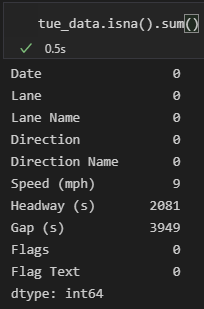
After selection, drop the column [Hour] to compute the row completeness.



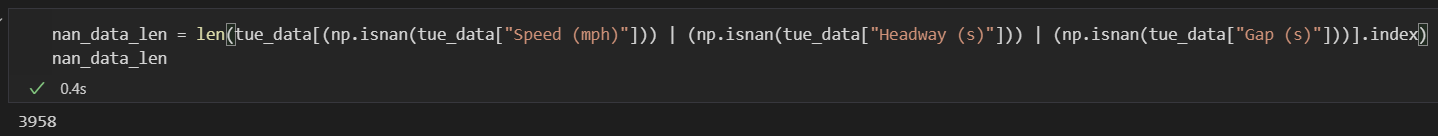
***Using formula 1***:



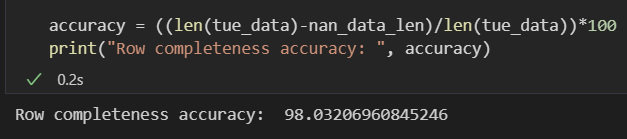
***Using formula 2***: First check which label has missing values.



Then get the number of rows which has missing values



Finally, using calculate the row completeness



***Task 7.1 (ii) Result (2 Decimal):*** By using first formula, row completeness is 99.70; By using second formula, row completeness is 98.03.

**(iii)** **Discussion**

Completeness is used to describe the comprehensiveness or wholeness of the data. The full score is 100, with higher scores representing fewer missing data. This is one aspect of evaluating the quality of the data.

The first formula means the average completeness of each row, and a result of 99.70 means that the average completeness of each row is 99.70.

The second formula means how many rows are complete and the result of 98.03 means that approximately 98 out of every 100 rows are complete.

The first formula is therefore more concerned with the completeness of each row, while the second formula is more concerned with the overall row completeness. However, the data usually requires complete rows and does not care how much data is missing from each row, or more precisely, one missing data of a row versus multiple missing data of a row does not allow for further analysis. Therefore, the second formula seems to make more sense from this point of view.

I think the row completeness of formula 2 is a better measure of file integrity than column completeness in task 5.1, because one column completeness is ill-considered, and missing values in a file could in many columns, whereas row completeness takes into account the missing values in multiple columns.

**Task 7.2**

*Excel is used for this task.*

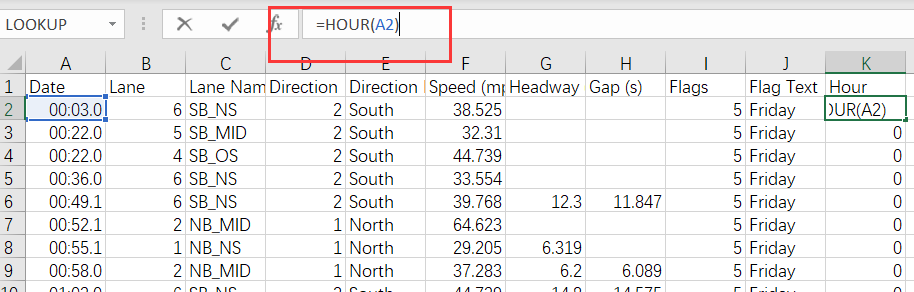
*The columns [Flags] and [Flag Text] are already obtained in Task 1, so they are used directly for this task.*

First put open two files in one single window.

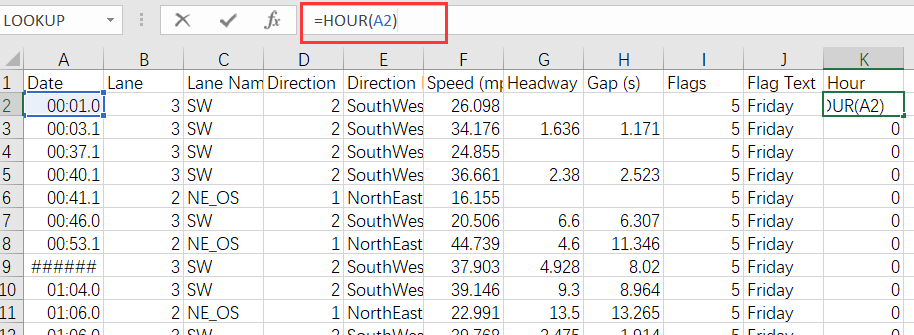


Use function Hour() to get specific hour corresponding to [Date] in both 1083 and 1415 files.

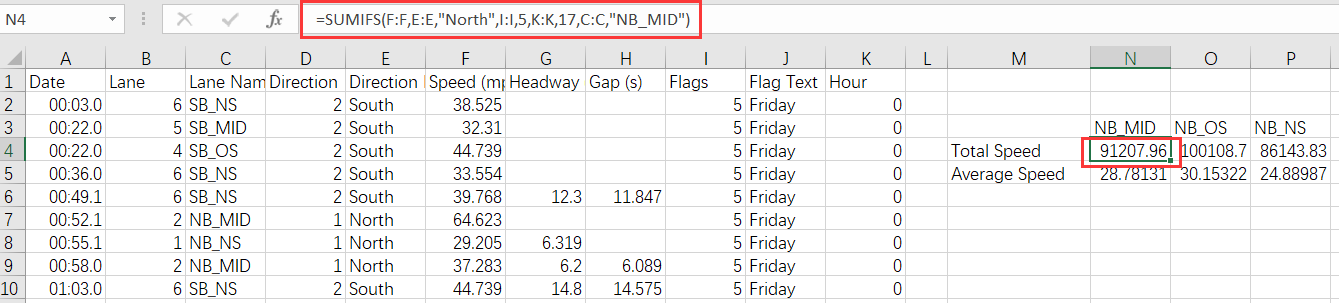
1083:

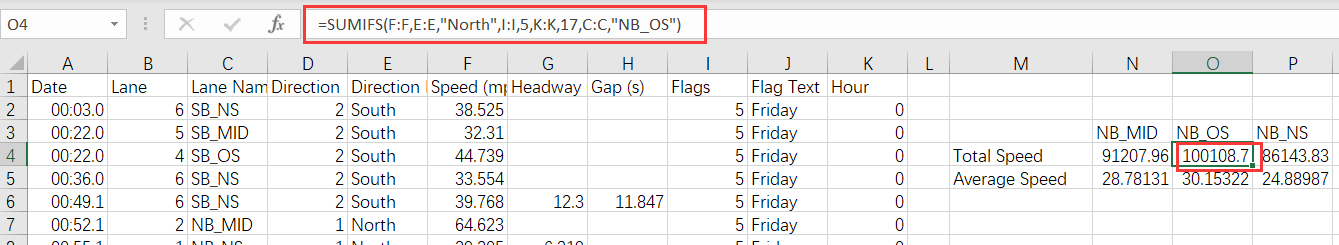


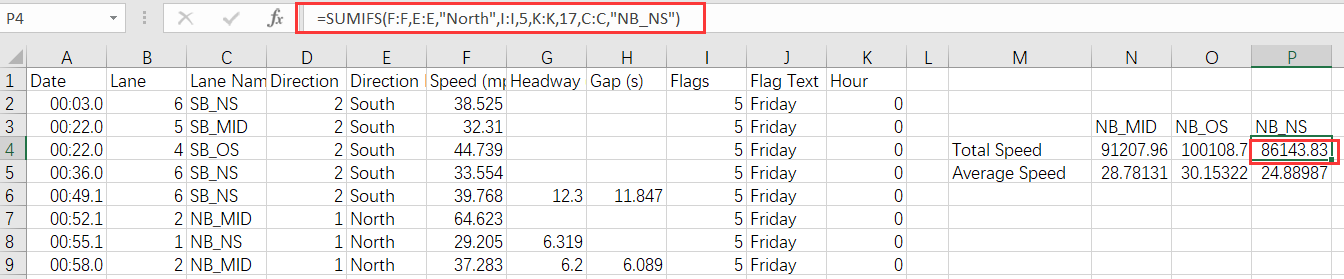
1415:

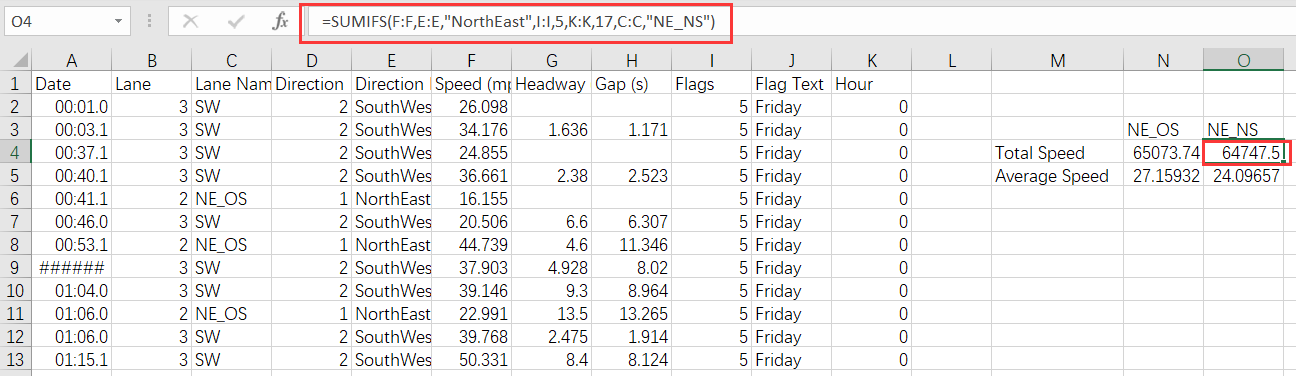
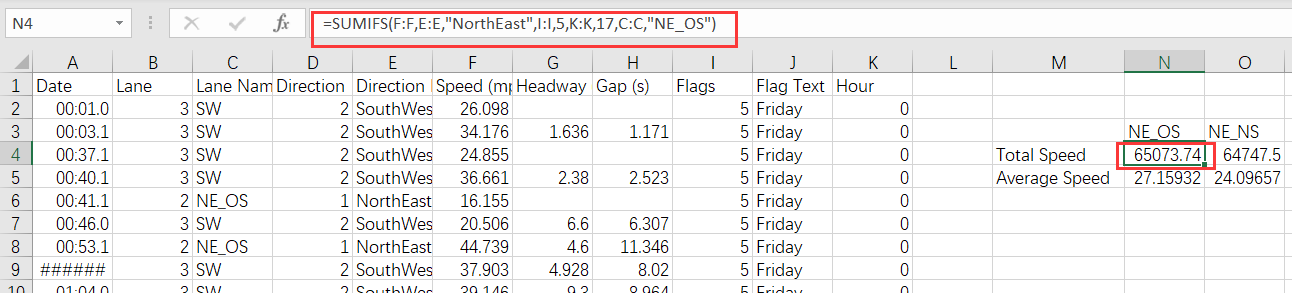


Next, use SUMIFS() function to calculate the total speed of each lane that satisfied the condition (for 1083: Friday between 17:00 and 18:00, NB\_MID lane or NB\_OS lane or NB\_NS lane; for 1415: Friday between 17:00 and 18:00, NE\_OS lane or NE\_NS lane).

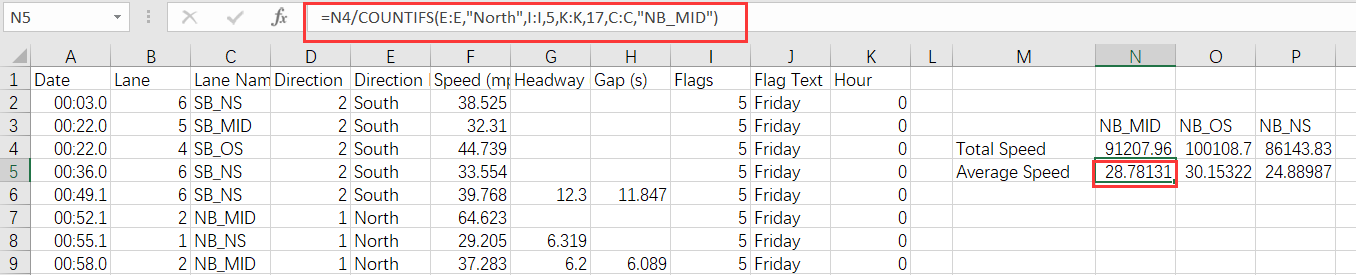


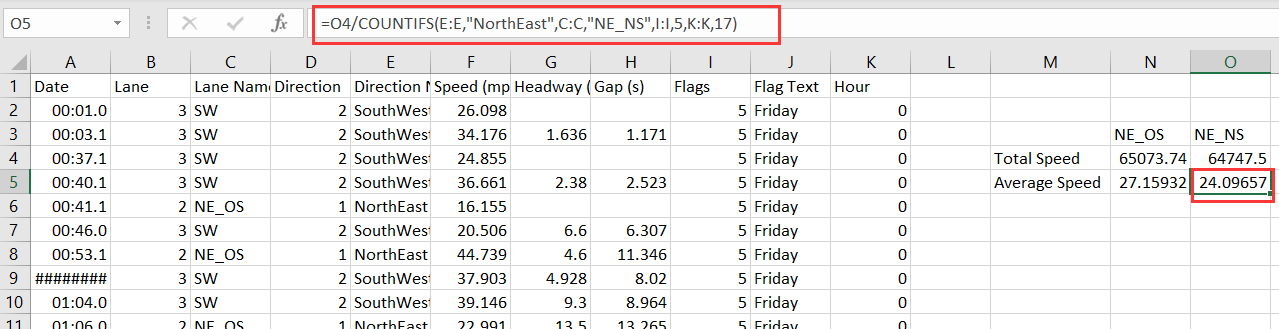
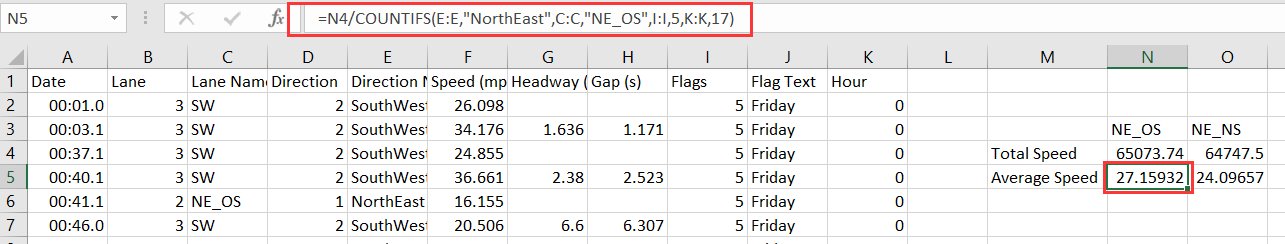
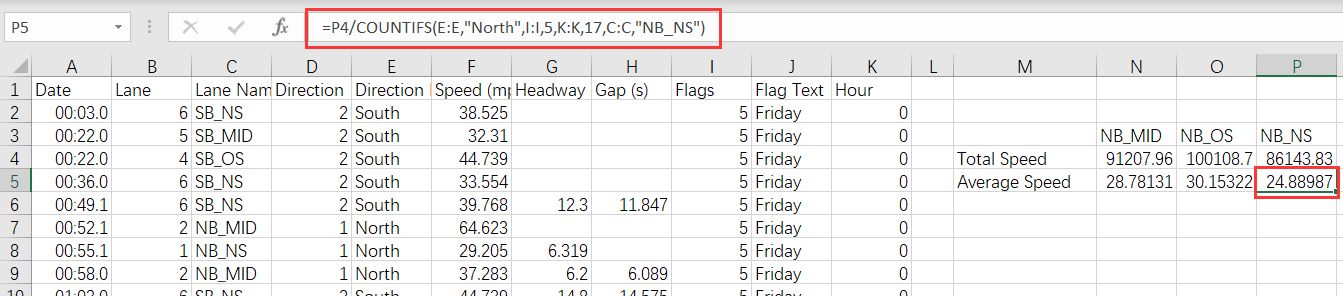
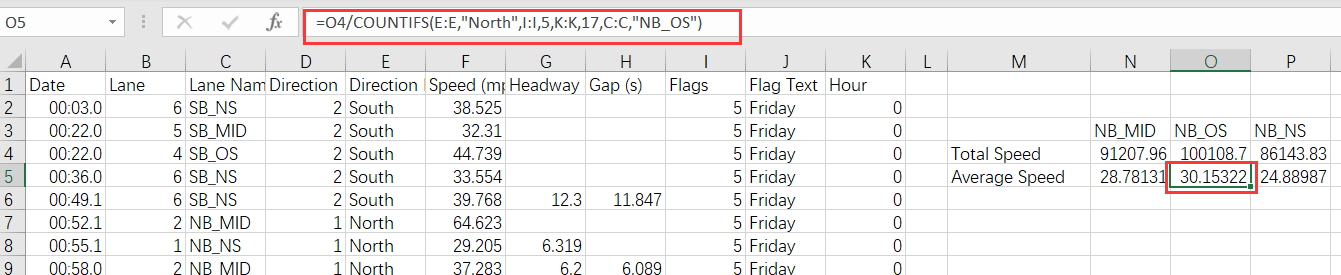




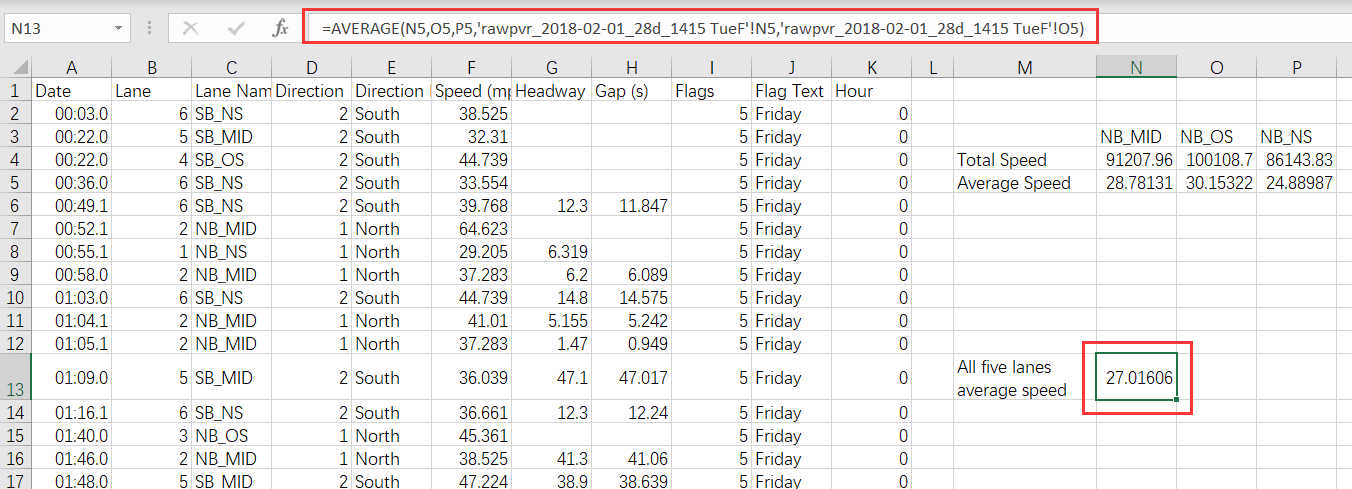


Use COUNTIFS() function to calculate the number of cells that satisfied the condition, and average speed for each lane is calculated by using total speed obtained above divided by number of cells.

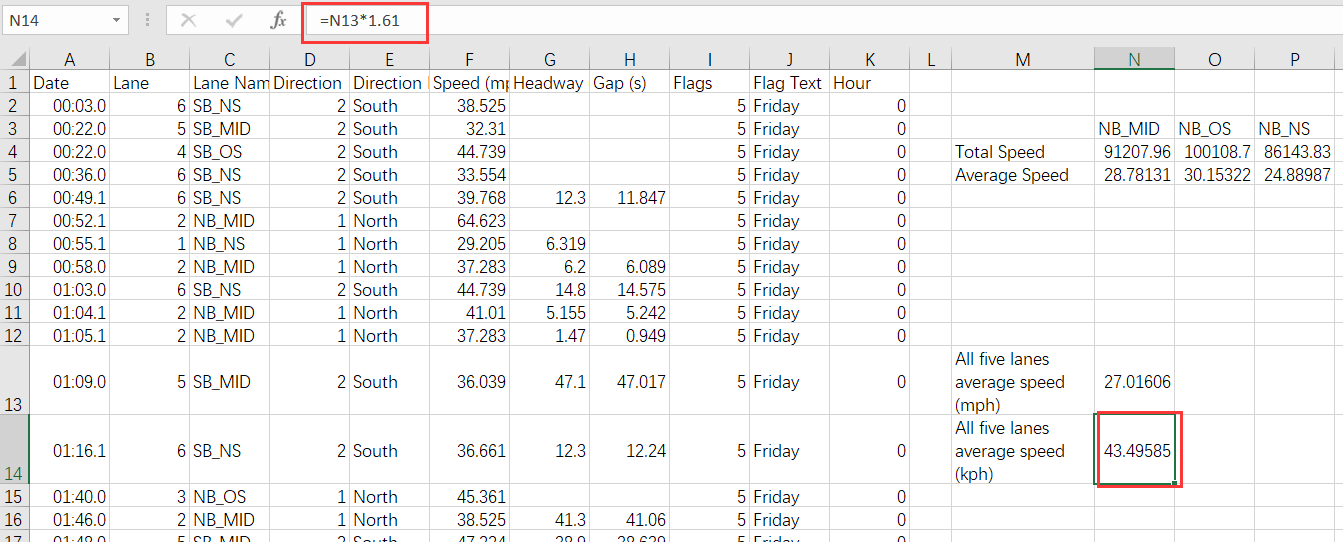




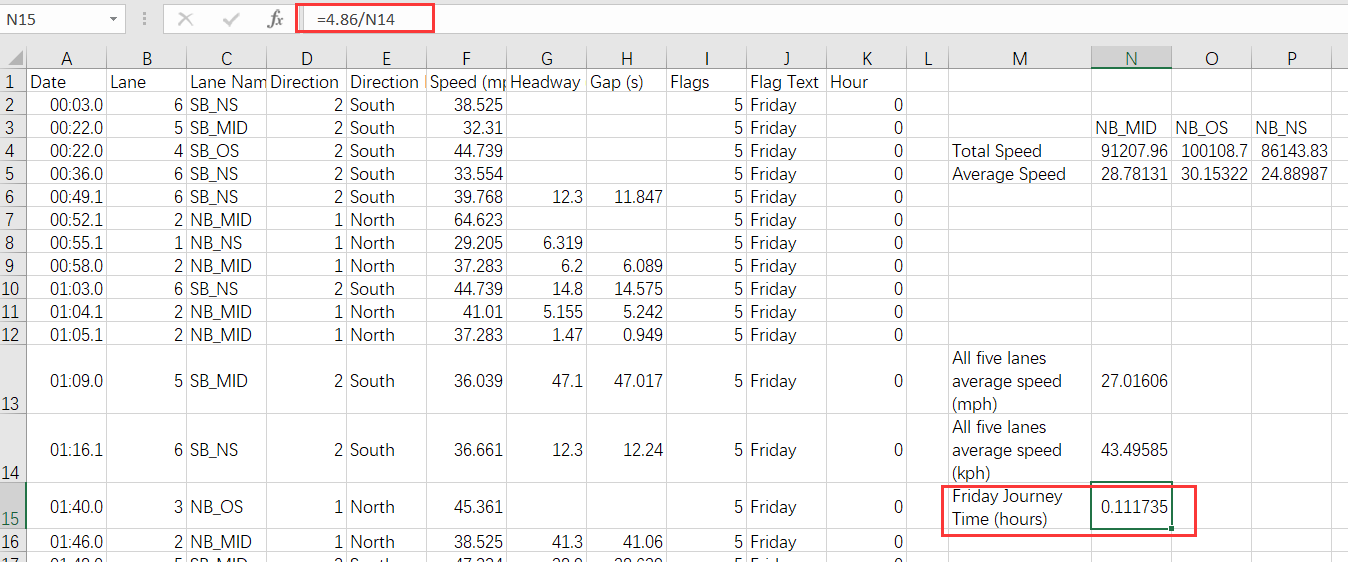
Calculate the average speed of 5 lanes.



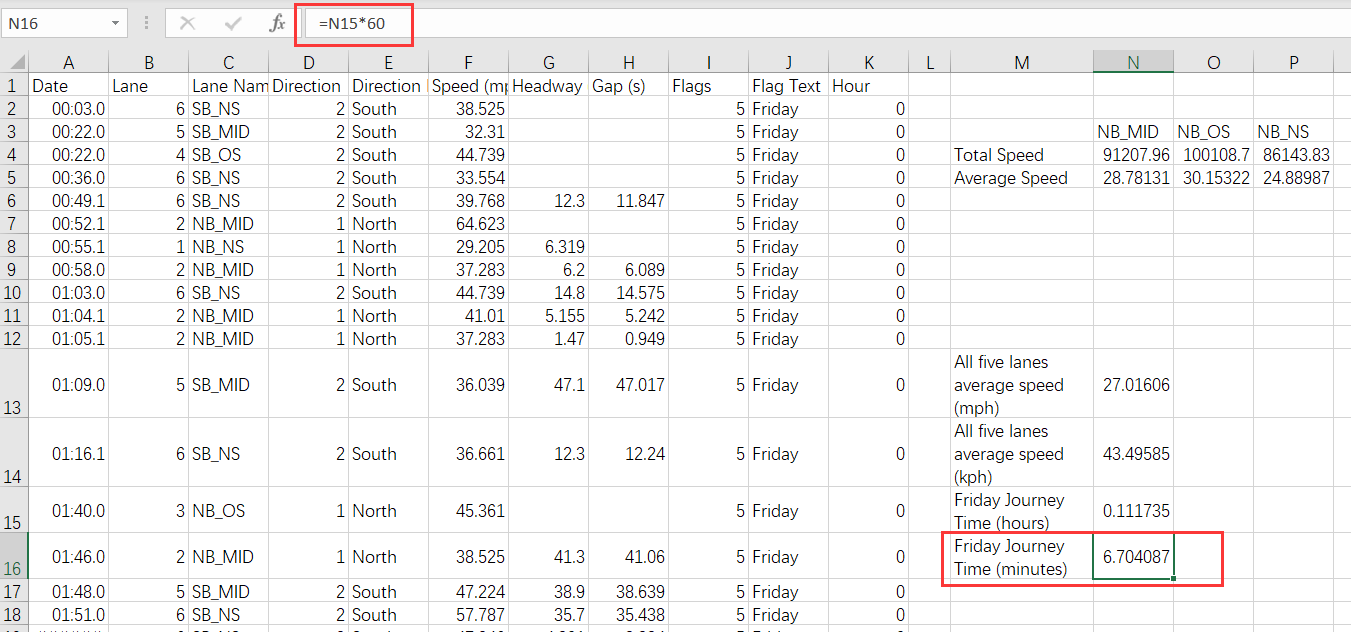
Convert the average speed to kilometers per hour.



Compute the journey time in hours.



Convert the journey time to minutes.



***Task 7.2 Result (2 Decimal):*** Journey time is 6.704 minutes

**Python and Excel Comparison**

**Similarity**

They can all be used for further processing, analysis and visualization of the data. Both have functions already defined and can be used directly. Both can operate on csv files.

**Python**

Advantage:

1. In addition to some self-contained functions, it is easy to download third-party libraries to make data processing simpler and more diverse (suck as machine learning).

2. More efficient when handling large amounts of data.

3. By writing the code once, repetitive operations can be performed.

4. The code is readable and can be saved, so it can be disseminated and improved.

5. It can be used in multiple platforms.

6. Can read and process many types of files

Disadvantage

1. Relatively difficult to learn.

2. May face various bugs.

**Excel**

Advantage

1. It has a graphical interface. Process data with a mouse click without writing a lot of code. More intuitive operation.

2. Easy to learn, excel can be used by short learning time.

3. High penetration rate, most computers have Excel.

Disadvantage

1. It gets laggy when processing large amounts of data

2. When performing the same operation on data, the operation needs to be repeated.

3. The process is not recorded after processing.

For this task, I prefer Python to Excel. Because Excel has too many repetitive operations. For instance, when calculating average speed of each lane, it uses same function repeatedly by just changing a condition. Python can easily do this once by using groupby() function. The running time of both two technology are in small difference.