Week 3 Tasks Sub-list 3

Student Name: Zhangli Wang
Student ID: 10868661
Student Central Name: n67315zw

In [1]:

```
# packages
import pandas as pd
import numpy as np
import datetime
import copy
from itertools import combinations
```

In [2]:

```
# Read dataset
df = pd.read_csv('./Datasets/rawpvr_2018-02-01_28d_1083 TueFri.csv')
```

Explanation:
Read the original csv file 'rawpvr_2018-02-01_28d_1083 TueFri.csv' into a dataframe.

In [3]:

```
# slice to acquire the 'Date' column
date = df.loc[:, 'Date']
# type conversion: string > timestamp
date = pd.to_datetime(date)
# calculate days of the week
days_of_week = [item.dayofweek for item in date]
# type conversion: list > series
days_of_week = pd.Series(np.array(days_of_week).T)
# update the column 'Flags' in the original dataframe
df.loc[:, 'Flags'] = days_of_week.map(lambda x: x+1, na_action='ignore')
# update the column 'Flag Text' in the original dataframe
df.loc[:, 'Flag Text'] = days_of_week.map(lambda x: 'Tuesday' if x == 1 else ('Friday' if x
== 4 else ''), na_action='ignore')
# add a new column 'Time' to store the time of the date
df = df.assign(Time = pd.to_datetime(df['Date']).dt.time)
# add a new column 'DateWOTime' to store the date without the time
df = df.assign(DateWOTime = pd.to_datetime(df['Date']).dt.date)
# transform the type of data in the 'Date' column of the dataset: string > timestamp
df = df.assign(Date = pd.to_datetime(df['Date']))
# add a new column 'DateHour' to store the date in the format of '%year%-%month%-%day%
%hour%:00:00'
df['DateHour'] = df['Date'].apply(lambda x: datetime.datetime(x.year, x.month, x.day,
x.hour, 0, 0))
# add a new column 'Hour' to store the hour in the format of '%hour%:00:00'
df['Hour'] = df['Date'].apply(lambda x: datetime.time(x.hour, 0, 0))
```

Explanation:
This code block is the data preparation operations for the input dataset.

- Update the column `Flag` and `Flag Text` with the day of the week id and the day of the week, respectively.
  - Create a series of the days of week.

    ```
    # slice to acquire the 'Date' column
    date = df.loc[:, 'Date']
    # type conversion: string > timestamp
    date = pd.to_datetime(date)
    # calculate days of the week
    days_of_week = [item.dayofweek for item in date]
    # type conversion: list > series
    days_of_week = pd.Series(np.array(days_of_week).T)
    ```

    Firstly, the column `Date` is sliced from the original dataframe and converted to the type timestamp. Then the days of week is calculated and its type is converted from list to series for adding columns in the dataframe.
  - Update the columns `Flag` and `Flag Text`.

    ```
    # update the column 'Flags' in the original dataframe
    df.loc[:, 'Flags'] = days_of_week.map(lambda x: x+1, na_action='ignore')
    # update the column 'Flag Text' in the original dataframe
    df.loc[:, 'Flag Text'] = days_of_week.map(lambda x: 'Tuesday' if x == 1 else ('F
    riday' if x == 4 else ''), na_action='ignore')
    ```

    Use map functions to map the series to the new columns in the dataframe with specified requirements.
- Add a new column `Time` in the dataframe to store the time in dates

  ```
  # add a new column `time` in the dataframe to store the time of the date
  df = df.assign(Time = pd.to_datetime(df['Date']).dt.time)
  ```

- Add a new 'DateWOTime' to store the date without the time

  ```
  # add a new column `DateWOTime` in the dataframe to store the date without the time
  df = df.assign(DateWOTime = pd.to_datetime(df['Date']).dt.date)
  ```

- Change the datatype of `Date` from `string` to `timestamp`

  ```
  # transform the type of data in the 'Date' column of the dataset: string > timestamp
  df = df.assign(Date = pd.to_datetime(df['Date']))
  ```

  This operation is used for filtering the dataframe according to dates in the future and is a necessary step for continuing the very next data preparasion operasion.
- Add a new column `DateHour` to store the date in the format of '%year%-%month%-%day% %hour%:00:00'

  ```
  # add a new column 'DateHour' to store the date in the format of '%year%-%month%-%day%
  %hour%:00:00'
  df['DateHour'] = df['Date'].apply(lambda x: datetime.datetime(x.year, x.month, x.day,
  x.hour, 0, 0))
  ```

  This operation is useful for filtering the dataframe according to one-hour time intervals.

The data preparasion operations mainly focus on altering and adding variaty of the time attribute for the convenience of filtering the dataframe accordign to time. In the future, for each sub-task, more data preparasion operations are executed and they mainly focus on filtering to a new dataframe which is to meet the constraints for achieving the final goals.

Task5

Task5.1

Solution:

In [4]:

```
# filter the dataset with regard to 'Flag Text'
df1 = df.loc[df['Flag Text'] == 'Tuesday']
# filter the dataset with regard to 'Time'
start_time = datetime.time(7, 0, 0)
end_time = datetime.time(19, 0, 0)
df1 = df1.loc[(df1['Time'] > start_time) & (df1['Time'] < end_time)]
```

Explanation:
This code block is the data preparation operations for Task5.1 that filters the dataframe to a new dataframe `df1` according to constraints of the dataset in Task5.1.

- Filter the dataset with regard to `Flag Text` that the day of the week is Tuesday.

  ```
  # filter the dataset with regard to 'Flag Text'
  df1 = df.loc[df['Flag Text'] == 'Tuesday']
  ```

- Filter the dataframe with regard to `Time` that the time is between 7:00 and 19:00.

  ```
  # filter the dataset with regard to 'Time'
  start_time = datetime.time(7, 0, 0)
  end_time = datetime.time(19, 0, 0)
  df1 = df.loc[(df['Time'] > start_time) & (df['Time'] < end_time)]
  ```

In [5]:

```
num_cells = len(df1)   # number of cells

# number of non-empty cells of the column 'Gap (s)'
num_non_e_cells_gap = len(df1['Gap (s)'].notna()[df1['Gap (s)'].notna() == True])
# column completeness of the column 'Gap (s)'
column_completeness_gap = 100 * num_non_e_cells_gap / num_cells

# number of non-empty cells of the column 'Headway (s)'
num_non_e_cells_headway = len(df1['Headway (s)'].notna()[df1['Headway (s)'].notna() ==
True])
# column comleteness of the column 'Headway (s)'
column_completeness_headway = 100 * num_non_e_cells_headway / num_cells

print('column completeness gap: ' + str(column_completeness_gap))
print('column completeness headway: ' + str(column_completeness_headway))
```

```
column completeness gap: 98.03654443753885
column completeness headway: 98.96532007458049
```

Explanation:
This code block is calculation of column completeness for the columns `Gap (s)` and `Headway (s)`. Firstly, the length of the columns in `df1` are calculated by `num_cells = len(df1)`. Then, the number of non-empty cells of each column are calculated like

```
# number of non-empty cells of the column 'Gap (s)'
num_non_e_cells_gap = len(df1['Gap (s)'].notna()[df1['Gap (s)'].notna() == True])
```

. Finally, the the values of column completeness are calculated like

```
# column completeness of the column 'Gap (s)'
column_completeness_gap = 100 * num_non_e_cells_gap / num_cells
```

.

Result:

- column completeness gap: 98.03654443753885
- column completeness headway: 98.96532007458049

Interpretation of the result:
The completeness values of the columns 'Gap (s)' and 'Headway (s)' are both over 98.0, which means that over 98 percent of the values are not empty. So for both the two columns, most of the records are filled with data. But there still exists a small proportion of missing values.

Task5.2

Solution:

In [6]:

```
# initialize dataframe for storing interpolated dataframe
df2 = df1.copy(deep=True)
# filter with regard to 'Lane Name'
df2 = df2.loc[df2['Lane Name'] == 'NB_MID']
```

Explanation:
This code block is the data preparation operations for Task5.2. Because the dataset to be used in this task also requires the same constraints as in Task5.1, a new dataframe `df2` is created from the dataframe used in Task5.1 as initialization.

```
df2 = df1.copy(deep=True)
```

- Filter the dataframe with regard to `Lane Name` that the lane name is 'NB_MID'

  ```
  # filter with regard to 'Lane Name'
  df2 = df2.loc[df2['Lane Name'] == 'NB_MID']
  ```

In conclusion, now the dataframe has additional restrictions that the `Flag Text` is 'Tuesday', the `Time` is ranged from 7:00 to 19:00, and the `Lane Name` is 'NB_MID'. In the next code block, data manipulation on `df2` can be done with the preparation.

In [7]:

```python
medians = dict()   # key: column name, value: medians of the column
for column_name in ['Gap (s)', 'Headway (s)']:
      # record unique hour
      unique_hours = df2['Hour'].unique()

      # calculate median values for all the time intervals
      medians_col = dict()   # key: hour, value: median value
      for unique_hour in unique_hours:
            # filter the dataframe with regard to date hour
            filtered_df2 = df2.loc[df2['Hour'] == unique_hour]

            # get the values of the column
            values = filtered_df2[column_name]

            # record the median
            medians_col.update({unique_hour: values.median()})

      # record the medians of the column
      medians.update({column_name: medians_col})

      # find the series of 'Date' of dataframe if the column_name is null
      ss_date_missing = df2.loc[df2[column_name].isna()]['Date']

      # update the cell of the dataframe with the median value
      for item in ss_date_missing:
            # get the row of the dataframe where the value is missing
            tempdf = df2.loc[df2['Date'] == item]

            # get the corresponding median value
            median_value = medians[column_name].get(tempdf['Hour'].iloc[0])

            # interpolate value to the dataframe
            df2.loc[df2['Date'] == item, column_name] = median_value

# print results
# print(medians)

# clear memory
# del df1
# del df2
```

Explanation:
This code block is the data manipulation on `df2`. Overall, there are two processes explained below.

- Find median values of the columns `Gap (s)` and `Headway (s)` for all one-hour intervals.
  - Firstly we initialize the object to store all the median values. It is a dictionary storing names of the columns `Gap (s)` and `Headway (s)` as two keys. And their values are dictionaries that stores median values of corresponding columns with regard to one-hour time intervals in the day.

    ```python
    medians = dict()  # key: column name, value: medians of the column
    ```

  - Then we iterate over the two columns `Gap (s)` and `Headway (s)` to execute the same operations to them. Following mentioned operations are executed in this iteration.

    ```python
    for column_name in ['Gap (s)', 'Headway (s)']:
    ```

  - To acheive the goal, we may want to iteratively filter `df2` according to each one-hour time interval between 7:00 and 19:00 and calculate median values. So to begin with we find all unique one-hour time intervals in `df2`.

    ```
    # record unique hour
    unique_hours = df2['Hour'].unique()
    ```

  - Then we execute operations mentioned in the previous bullet point and the median values for the column are calculated and updated to `medians`.

    ```
    # calculate median values for all the time intervals
    medians_col = dict()  # key: date hour, value: median value
    for unique_hour in unique_hours:
      # filter the dataframe with regard to hour
      filtered_df2 = df2.loc[df2['Hour'] == unique_hour]

      # get the values of the column
      values = filtered_df2[column_name]

      # record the median
      medians_col.update({unique_hour: values.median()})

    # record the medians of the column
    medians.update({column_name: medians_col})
    ```

- Update the cells with missing values with the corresponding median values.
  - To acheive this goal, we need to find and record all the rows that have missing values. We can only take the column `Date` for future comparison as the dates are unique.

    ```
    # find the series of 'Date' of dataframe if the column_name is null
    ss_date_missing = df2.loc[df2[column_name].isna()]['Date']
    ```

  - Now we can update the values by iteratively find a missing row in `df2` and update it with the corresponding value.

    ```
    # update the cell of the dataframe with the median value
    for item in ss_date_missing:
      # get the row of the dataframe where the value is missing
      tempdf = df2.loc[df2['Date'] == item]

      # get the corresponding median value
      median_value = medians[column_name].get(tempdf['Hour'].iloc[0])

      # interpolate value to the dataframe
      df2.loc[df2['Date'] == item, column_name] = median_value
    ```

    In the iteration, the first step is to get the row that the date (unique key) in `df2` is matched with the one in `ss_date_missing`. The second step is to find the corresponding median value in `medians` according to the time of the row. The final step is updating the row.

Result of median values:

```
{'Gap (s)':
  {datetime.time(7, 0): 1.834,
  datetime.time(8, 0): 2.1020000000000003,
  datetime.time(9, 0): 2.08,
  datetime.time(10, 0): 2.5835,
  datetime.time(11, 0): 2.7785,
  datetime.time(12, 0): 2.795,
  datetime.time(13, 0): 2.7560000000000002,
  datetime.time(14, 0): 2.8049999999999997,
  datetime.time(15, 0): 2.577,
  datetime.time(16, 0): 2.3225,
  datetime.time(17, 0): 2.187,
  datetime.time(18, 0): 2.228},
 'Headway (s)':
  {datetime.time(7, 0): 2.722,
  datetime.time(8, 0): 3.1910000000000003,
  datetime.time(9, 0): 2.72,
  datetime.time(10, 0): 3.0,
  datetime.time(11, 0): 3.21,
  datetime.time(12, 0): 3.16,
  datetime.time(13, 0): 3.102,
  datetime.time(14, 0): 3.133,
  datetime.time(15, 0): 2.92,
  datetime.time(16, 0): 2.853,
  datetime.time(17, 0): 2.9065,
  datetime.time(18, 0): 2.79}}
```

Result of screenshot of the updated dataframe:

	Date	Lane	Lane Name	Direction	Direction Name	Speed (mph)	Headway (s)	Gap (s)	Flags	Flag Text	Time	DateWOTime	DateHour	Hour
68517	2018-02-06 07:00:01.160	2	NB_MID	1	North	27.962	1.720	1.834	2	Tuesday	07:00:01.160000	2018-02-06	2018-02-06 07:00:00	07:00:00
68518	2018-02-06 07:00:03.000	2	NB_MID	1	North	27.962	2.480	1.992	2	Tuesday	07:00:03	2018-02-06	2018-02-06 07:00:00	07:00:00
68521	2018-02-06 07:00:04.020	2	NB_MID	1	North	27.962	1.520	0.856	2	Tuesday	07:00:04.020000	2018-02-06	2018-02-06 07:00:00	07:00:00
68524	2018-02-06 07:00:05.020	2	NB_MID	1	North	29.205	1.226	0.680	2	Tuesday	07:00:05.020000	2018-02-06	2018-02-06 07:00:00	07:00:00
68529	2018-02-06 07:00:07.020	2	NB_MID	1	North	29.825	2.137	1.694	2	Tuesday	07:00:07.020000	2018-02-06	2018-02-06 07:00:00	07:00:00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
496010	2018-02-27 18:58:51.060	2	NB_MID	1	North	35.417	1.200	0.725	2	Tuesday	18:58:51.060000	2018-02-27	2018-02-27 18:00:00	18:00:00
496029	2018-02-27 18:59:26.080	2	NB_MID	1	North	41.010	35.200	34.941	2	Tuesday	18:59:26.080000	2018-02-27	2018-02-27 18:00:00	18:00:00
496050	2018-02-27 18:59:48.040	2	NB_MID	1	North	34.798	22.600	22.382	2	Tuesday	18:59:48.040000	2018-02-27	2018-02-27 18:00:00	18:00:00
496051	2018-02-27 18:59:50.000	2	NB_MID	1	North	33.554	1.800	1.343	2	Tuesday	18:59:50	2018-02-27	2018-02-27 18:00:00	18:00:00
496061	2018-02-27 18:59:58.060	2	NB_MID	1	North	43.495	7.600	7.307	2	Tuesday	18:59:58.060000	2018-02-27	2018-02-27 18:00:00	18:00:00

Task6

Solution:

In [8]:

```
# read the dataset of the site 1415
dfb = pd.read_csv('./Datasets/rawpvr_2018-02-01_28d_1415 TueFri.csv')
```

In [9]:

```
# apply the same data preparation executions to dataframe b

# slice to acquire the 'Date' column
date = dfb.loc[:, 'Date']
# type conversion: string > timestamp
date = pd.to_datetime(date)
# calculate days of the week
days_of_week = [item.dayofweek for item in date]
# type conversion: list > series
days_of_week = pd.Series(np.array(days_of_week).T)
# update the column 'Flags' in the original dataframe
dfb.loc[:, 'Flags'] = days_of_week.map(lambda x: x+1, na_action='ignore')
# update the column 'Flag Text' in the original dataframe
dfb.loc[:, 'Flag Text'] = days_of_week.map(lambda x: 'Tuesday' if x == 1 else ('Friday' if x == 4 else ''), na_action='ignore')
# add a new column 'Time' to store the time of the date
dfb = dfb.assign(Time = pd.to_datetime(dfb['Date']).dt.time)
# add a new column 'DateWOTime' to store the date without the time
dfb = dfb.assign(DateWOTime = pd.to_datetime(dfb['Date']).dt.date)
# transform the type of data in the 'Date' column of the dataset: string > timestamp
dfb = dfb.assign(Date = pd.to_datetime(dfb['Date']))
# add a new column 'DateHour' to store the date in the format of '%year%-%month%-%day%%hour%:00:00'
dfb['DateHour'] = dfb['Date'].apply(lambda x: datetime.datetime(x.year, x.month, x.day, x.hour, 0, 0))
```

Explanation:
In this code block, we simply execute the same data preparation operations for `dfb` as the ones we did for `df` for consistency and future usage.

In [10]:

```
# create a alias for the dataframe of the site 1083 for this task
dfa = df
# use a variable to reference the two dataframes
dfs = {'1083': dfa,
       '1415': dfb}
```

Explanation:
Firstly an alias of the dataframe for the site 1083 is created (`dfa`), this is for standardize the names for the dataframes in avoidance of generating potential inconsistency in the future. Then the dataframes are referenced together in one variable (`dfs`) for the convenience of coding.

In [11]:

```
# filter the two dataframes

# create a copy of the old dataframes
dfs1 = copy.deepcopy(dfs)
for key, df in dfs1.items():
      # filter the dataframe with regard to 'Time'
      start_time = datetime.time(17, 0, 0)
      end_time = datetime.time(18, 0, 0)
      df = df.loc[(df['Time'] > start_time) & (df['Time'] < end_time)]

      # filter the dataframe with regard to 'Direction'
      df = df.loc[df['Direction'] == 1]

      # filter the dataframe with regard to 'Flag Text'
      df = df.loc[df['Flag Text'] == 'Friday']

      # record
      dfs1[key] = df
```

Explanation:
This code block is the data preparation operations for Task6. Because the operations are specificly focused on Task6, we created a new version `dfs1` of the dataframe `dfs` similar to the things we did before `dfs1 = copy.deepcopy(dfs)` .\

Then we iterate on `dfs1` for each dataframe and perform the following operations.

- Filter the dataframe with regard to 'Time' that the time is between 17:00 and 18:00.

  ```
  # filter the dataframe with regard to 'Time'
  start_time = datetime.time(17, 0, 0)
  end_time = datetime.time(18, 0, 0)
  df = df.loc[(df['Time'] > start_time) & (df['Time'] < end_time)]
  ```

- Filter the dataframe with regard to 'Direction' that the direction is heading north.

  ```
  # filter the dataframe with regard to 'Direction'
  df = df.loc[df['Direction'] == 1]
  ```

- Filter the dataframe with regard to 'Flag Text' that the day of the week is Friday.

  ```
  # filter the dataframe with regard to 'Flag Text'
  df = df.loc[df['Flag Text'] == 'Friday']
  ```

In [12]:

```
# record unique lane names
unique_lane_names = dict()
for key, df in dfs1.items():
      # find unique lane names for the dataframe
      unique_lane_name = df['Lane Name'].unique()

      # record
      unique_lane_names.update({key: unique_lane_name})
```

Explanation: This code block is an intermediate process to record unique lane names `unique_lane_names` of the two dataframes for the usage in the next code block.
Result of `unique_lane_names` :

```
{'1083': array(['NB_OS', 'NB_MID', 'NB_NS'], dtype=object),
 '1415': array(['NE_NS', 'NE_OS'], dtype=object)}
```

In [13]:

```
# find average travel time of the lanes
avg_times_lanes = dict()   # the average travel times of the two dataframes
for key, df in dfs1.items():
      avg_times_lanes_df = list()   # the average speeds of the dataframe
      for lane_name in unique_lane_names[key]:
            # filter the dataframe with regard to 'Lane Name'
            filtered_df = df.loc[df['Lane Name'] == lane_name]

            # calculate the average speed
            avg_speed = filtered_df['Speed (mph)'].mean()

            # calculate the average travel time in minute by the equation 'time = distance/speed'
            #  4.86km = 3.019864miles
            avg_time_lane = 3.019864 / avg_speed * 60

            # record
            avg_times_lanes_df.append(avg_time_lane)
      avg_times_lanes.update({key: avg_times_lanes_df})

# print result of average speeds in different lanes
print(avg_times_lanes)

# calculate the average value of the result
templ = list(map(lambda l: (len(l), sum(l)), avg_times_lanes.values()))
sum_templ = tuple(map(sum, zip(*templ)))   # * is for unpacking the list  cool
avg_time = sum_templ[1]/sum_templ[0]

# print result of average time
print(avg_time)
```

```
{'1083': [6.009038525021391, 6.295469340374707, 7.279743256362566], '1415': [7.51940
2241603255, 6.6714418953907915]}
6.7550190517505415
```

Explanation:
This code block is about finding the average travel times in different lanes and calculating their average value. The two processes are introduced below.

- Find the average travel times in different lanes.
  - To begin with, a dictionary `avg_times_lanes` is created `avg_times_lanes = dict()` to store the name of the sites as the keys and their corresponding average values with regard to different lanes.
  - Then the dataframes are iterated. In each iteration, we iteratively filter the dataframe with regard to the lane names.

```
for lane_name in unique_lane_names[key]:
  # filter the dataframe with regard to 'Lane Name'
  filtered_df = df.loc[df['Lane Name'] == lane_name]
```

- Next, the average speed of the `filtered_df` is calculated `avg_speed = filtered_df['Speed (mph)'].mean()` .
- Next, the average time was calculated by the average speed and the given distance, we firstly changed the unit of the distance from 'km' to 'miles' `avg_time_lane = 3.019864 / avg_speed * 60` .
- Finally we record the results, following is the integral process.

```
avg_times_lanes_df = list()  # the average speeds of the dataframe
for lane_name in unique_lane_names[key]:
  # filter the dataframe with regard to 'Lane Name'
  filtered_df = df.loc[df['Lane Name'] == lane_name]

  # calculate the average speed
  avg_speed = filtered_df['Speed (mph)'].mean()

  # calculate the average travel time in minute by the equation 'time = distance/s
peed'
  #  4.86km = 3.019864miles
  avg_time_lane = 3.019864 / avg_speed * 60

  # record
  avg_times_lanes_df.append(avg_time_lane)
avg_times_lanes.update({key: avg_times_lanes_df})
```

- Calculate the average value of the average travel times.
  - For calculating the average value of the result, firstly the number of elements and the sum of the values of the elements are calculated and stored in `templ` .

```
templ = list(map(lambda l: (len(l), sum(l)), avg_times_lanes.values()))
```

```
[(3, 19.584251121758662), (2, 14.190844136994047)]
```

  - Then, a map function is used to calculate the sum of values in the tuples in the list and finally calculate the average travel time by dividing the sum of speeds to the number of elements.

```
sum_templ = tuple(map(sum, zip(*templ)))  # * is for unpacking the list  cool
avg_time = sum_templ[1]/sum_templ[0]
```

Results of average travel time in different lanes:

```
{'1083': [6.009038525021391, 6.295469340374707, 7.279743256362566],
 '1415': [7.519402241603255, 6.6714418953907915]}
```

Result of overall average travel time:

- 6.7550190517505415 minutes

Interpretation of the result:
The result shows that according to this dataset, the average travel time of north lanes between the two sites in the time from 17:00 to 18:00 in Friday is around 6.755 minutes.

Task7.1

Solution (i):

I can describe the *Row Completeness* as a set of fractions with regard to combinations of columns that can possibly have missing values, where the numerators are the numbers of rows that are with missing values in certain columns and the denominators are the total number of rows.

$$RowCompleteness_1 = \{RowCompleteness_C : C\ is\ the\ set\ of\ all\ possible\ combinations\ of\ columns\}$$
$$RowCompleteness_C = NumberOfMissingRows_C \times 100 \div NumberOfRows$$

The row completeness value indicate the proportion of the number of not-missing values to the total number of values.

Besides we can calculate the row completeness of the proportion of the rows that have any missing values to the total number of rows.

$$RowCompleteness_2 = NumberOfRowsWithMissingValues \times 100 \div NumberOfRows$$

Solution (ii):

In [14]:

```python
# filter the dataframe as we did before
df3 = df1.copy(deep=True)
```

Explanation:
This code block is the data preparation operations for Task7.1. Since the filtering requirements in this task are the same as in Task5.1, we can reference to the codes in the Task5.1 section.

In [15]:

```python
# find the columns that contain missing values
ifna_cols = df3.isna().any()
missing_cols = list(ifna_cols[ifna_cols == True].keys())

# find all possible combinations in the missing columns
# reference: https://stackoverflow.com/questions/464864/how-to-get-all-possible-combinations-of-a-list-s-elements?rq=1
combs = sum([list(map(list, combinations(missing_cols, i))) for i in range(len(missing_cols) + 1)], [])
combs.remove([])
```

Explanation:
Before we start the manipulations, we should know which columns contain missing values and find all possible combinations of them. As we do not need the empty set as a combination, we simply remove it from our combinations.

In [16]:

```
# initialize row completeness object
RC_count = dict()   # key: combination of columns, value: count

for comb in combs:
      # filter the dataframe with regard to combination of columns
      filtered_df3 = df3[comb]

      # calculate the number of rows that all cells are of missing values
      count = len(filtered_df3.loc[filtered_df3.isnull().all(axis=1)].index)

      # record
      RC_count.update({str(comb): count})

# calculate the row completeness
total_count = len(df3.index)
RC1 = dict(zip(RC_count.keys(), map(lambda x: 100 * int(x) / total_count,
RC_count.values())))

# calculate the the row completeness for the entire dataframe
count = len(df3.loc[df3.isnull().any(axis=1)].index)
RC2 = 100 * count / total_count
```

Explanation:
This code block is about calculating the row completeness.

- First, we initialize a row completeness count dictionary that stores the combinations of columns as key and the count of the rows that cells in the combination of columns are missing.

  ```
  RC_count = dict()  # key: combination of columns, value: count
  ```

- Then we iteratively filter `df3` with regard to the combination of columns `filtered_df3 = df3[comb]`, calculate the counts, and record them to `RC_count`.

  ```
  for comb in combs:
    # filter the dataframe with regard to combination of columns
    filtered_df3 = df3[comb]

    # calculate the number of rows that all cells are of missing values
    count = len(filtered_df3.loc[filtered_df3.isnull().all(axis=1)].index)

    # record
    RC_count.update({str(comb): count})
  RC_count
  ```

- Finally, we need to map `RC_count` according to the previously defined row completeness equation.

  ```
  # calculate the row completeness
  total_count = len(df3.index)
  RC = dict(zip(RC_count.keys(), map(lambda x: 100 * int(x) / total_count, RC_count.val
  ues())))
  ```

Result of Row Completeness Calculation 1:

```
{"['Speed (mph)']": 0.004474829086389061,
 "['Headway (s)']": 1.0346799254195151,
 "['Gap (s)']": 1.963455562461156,
 "['Speed (mph)', 'Headway (s)']": 0.0,
 "['Speed (mph)', 'Gap (s)']": 0.0,
 "['Headway (s)', 'Gap (s)']": 1.0346799254195151,
 "['Speed (mph)', 'Headway (s)', 'Gap (s)']": 0.0}
```

Result of Row Completeness Calculation 2:
1.967930391547545

Discussion

Interpretation of the Results

The result shows the values of row completeness considering all combinations of columns. Overall, the total rate of rows that have missing values is about 1.96793. For individual columns, the 'Speed (mph)' column has the minimum percentage of missing values, which is around 0.00447. And the column 'Gap (s)' has the highest proportion 1.96345. And we can infer from the result that for all the combinations that have the column 'Speed (mph)' and other columns, all the 'Speed (mph)' columns are filled with values. And in the combination ('Headway (s)', 'Gap (s)'), at least all the values in the column 'Headway (s)' are missing.

Understanding of Completeness

The completeness is an important aspect in data quality. Data completeness can be defined that "the degree to which a data collection has values or all attributes of entities that are supposed to have values" [1]. In my interpretation, to see from the perspective of application, the completeness of data is that within the scale of the potential utilization of the data, values of objective attributes should be there for utilization. Hence completeness is inextricably linked with data quality. For implementation, in a dataset, the completeness can be demonstrated as the proportion of values that are stored to the number of possible records of the dataset. Or by Pipino, Lee, and Wang's definition [2], it is "a function of the missing values in a column of a table".

Comparition of the two Completeness Formulas

I can compare the two formulas in terms of the difference between the representations of the completeness values. The row completeness demonstrates the completeness of values accross multiple columns, while the column completeness demonstrate the completeness of values in columns. Hence the row completeness formula can show the degree of completeness over the entire dataset, while the column completeness formula shows the degree of completeness in specific columns.For this dataset, because only three columns may have missing values, calculating the row completeness formula regarding combinations of columns are easy to implement, and the values calculated can provide a more thorough overview of the completeness than the ones calculated column completeness formula. We can obtain the overview of the rate of any values in these columns are missing as well as the inspect of a specific combination of columns that all the values inside them are missing. In the column completeness calculation, we can only inspect the value for an individual column.

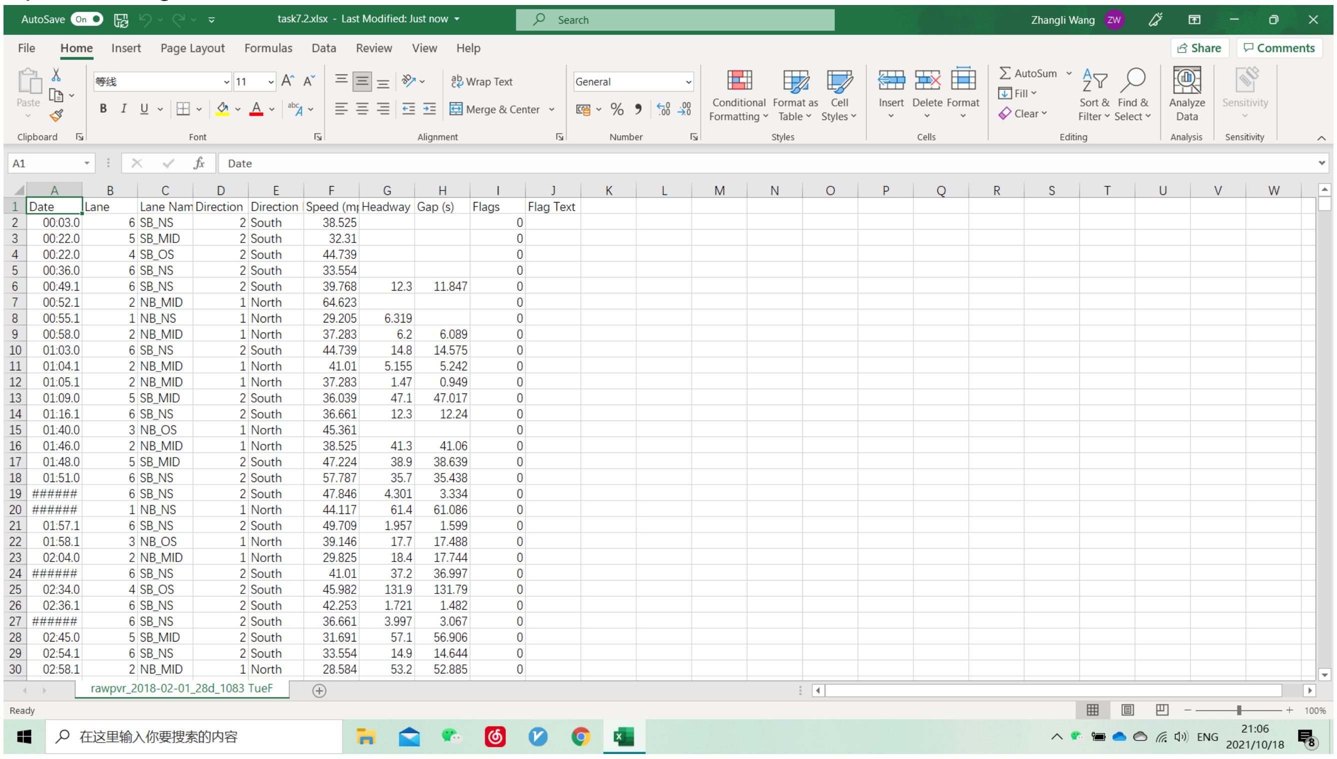
[1] Li, C.: Computing complete answers to queries in the presence of limited access patterns. VLDB J. 12, 211–277 (2003)
[2] Pipino, L.L., Lee, Y.W., Wang, R.Y.: Data quality assessment. Commun. ACM 45, 211–218 (2002)

Task7.2

Solution:

I use **Microsoft Excel** as the tool for this task.

1. Open the original dataset in Excel.



task7.2.xlsx - Last Modified: Just now

A1 | Date

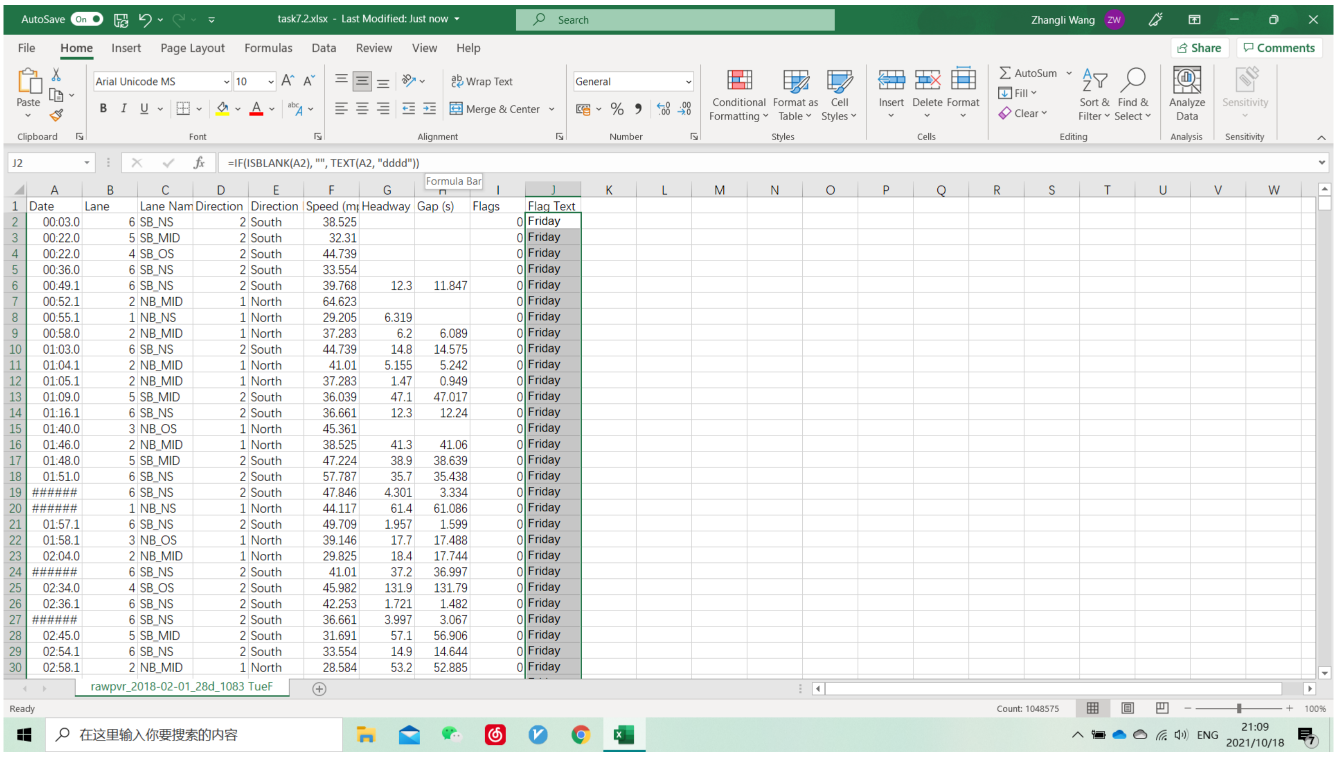
	A	B	C	D	E	F	G	H	I	J
1	Date	Lane	Lane Nam	Direction	Direction	Speed (mp	Headway	Gap (s)	Flags	Flag Text
2	00:03.0	6	SB_NS	2	South	38.525			0	
3	00:22.0	5	SB_MID	2	South	32.31			0	
4	00:22.0	4	SB_OS	2	South	44.739			0	
5	00:36.0	6	SB_NS	2	South	33.554			0	
6	00:49.1	6	SB_NS	2	South	39.768	12.3	11.847	0	
7	00:52.1	2	NB_MID	1	North	64.623			0	
8	00:55.1	1	NB_NS	1	North	29.205	6.319		0	
9	00:58.0	2	NB_MID	1	North	37.283	6.2	6.089	0	
10	01:03.0	6	SB_NS	2	South	44.739	14.8	14.575	0	
11	01:04.1	2	NB_MID	1	North	41.01	5.155	5.242	0	
12	01:05.1	2	NB_MID	1	North	37.283	1.47	0.949	0	
13	01:09.0	5	SB_MID	2	South	36.039	47.1	47.017	0	
14	01:16.1	6	SB_NS	2	South	36.661	12.3	12.24	0	
15	01:40.0	3	NB_OS	1	North	45.361			0	
16	01:46.0	2	NB_MID	1	North	38.525	41.3	41.06	0	
17	01:48.0	5	SB_MID	2	South	47.224	38.9	38.639	0	
18	01:51.0	6	SB_NS	2	South	57.787	35.7	35.438	0	
19	######	6	SB_NS	2	South	47.846	4.301	3.334	0	
20	######	1	NB_NS	1	North	44.117	61.4	61.086	0	
21	01:57.1	6	SB_NS	2	South	49.709	1.957	1.599	0	
22	01:58.1	3	NB_OS	1	North	39.146	17.7	17.488	0	
23	02:04.0	2	NB_MID	1	North	29.825	18.4	17.744	0	
24	######	6	SB_NS	2	South	41.01	37.2	36.997	0	
25	02:34.0	4	SB_OS	2	South	45.982	131.9	131.79	0	
26	02:36.1	6	SB_NS	2	South	42.253	1.721	1.482	0	
27	######	6	SB_NS	2	South	36.661	3.997	3.067	0	
28	02:45.0	5	SB_MID	2	South	31.691	57.1	56.906	0	
29	02:54.1	6	SB_NS	2	South	33.554	14.9	14.644	0	
30	02:58.1	2	NB_MID	1	North	28.584	53.2	52.885	0	

rawpvr_2018-02-01_28d_1083 TueF

2. For the entire column of "Flag Text", apply the function

```
=IF(ISBLANK(A2), "", TEXT(A2, "dddd"))
```

, which calculates days of the week regarding the "Date" column.



task7.2.xlsx - Last Modified: Just now

J2 | =IF(ISBLANK(A2), "", TEXT(A2, "dddd"))

	A	B	C	D	E	F	G	H	I	J
1	Date	Lane	Lane Nam	Direction	Direction	Speed (mp	Headway	Gap (s)	Flags	Flag Text
2	00:03.0	6	SB_NS	2	South	38.525			0	Friday
3	00:22.0	5	SB_MID	2	South	32.31			0	Friday
4	00:22.0	4	SB_OS	2	South	44.739			0	Friday
5	00:36.0	6	SB_NS	2	South	33.554			0	Friday
6	00:49.1	6	SB_NS	2	South	39.768	12.3	11.847	0	Friday
7	00:52.1	2	NB_MID	1	North	64.623			0	Friday
8	00:55.1	1	NB_NS	1	North	29.205	6.319		0	Friday
9	00:58.0	2	NB_MID	1	North	37.283	6.2	6.089	0	Friday
10	01:03.0	6	SB_NS	2	South	44.739	14.8	14.575	0	Friday
11	01:04.1	2	NB_MID	1	North	41.01	5.155	5.242	0	Friday
12	01:05.1	2	NB_MID	1	North	37.283	1.47	0.949	0	Friday
13	01:09.0	5	SB_MID	2	South	36.039	47.1	47.017	0	Friday
14	01:16.1	6	SB_NS	2	South	36.661	12.3	12.24	0	Friday
15	01:40.0	3	NB_OS	1	North	45.361			0	Friday
16	01:46.0	2	NB_MID	1	North	38.525	41.3	41.06	0	Friday
17	01:48.0	5	SB_MID	2	South	47.224	38.9	38.639	0	Friday
18	01:51.0	6	SB_NS	2	South	57.787	35.7	35.438	0	Friday
19	######	6	SB_NS	2	South	47.846	4.301	3.334	0	Friday
20	######	1	NB_NS	1	North	44.117	61.4	61.086	0	Friday
21	01:57.1	6	SB_NS	2	South	49.709	1.957	1.599	0	Friday
22	01:58.1	3	NB_OS	1	North	39.146	17.7	17.488	0	Friday
23	02:04.0	2	NB_MID	1	North	29.825	18.4	17.744	0	Friday
24	######	6	SB_NS	2	South	41.01	37.2	36.997	0	Friday
25	02:34.0	4	SB_OS	2	South	45.982	131.9	131.79	0	Friday
26	02:36.1	6	SB_NS	2	South	42.253	1.721	1.482	0	Friday
27	######	6	SB_NS	2	South	36.661	3.997	3.067	0	Friday
28	02:45.0	5	SB_MID	2	South	31.691	57.1	56.906	0	Friday
29	02:54.1	6	SB_NS	2	South	33.554	14.9	14.644	0	Friday
30	02:58.1	2	NB_MID	1	North	28.584	53.2	52.885	0	Friday

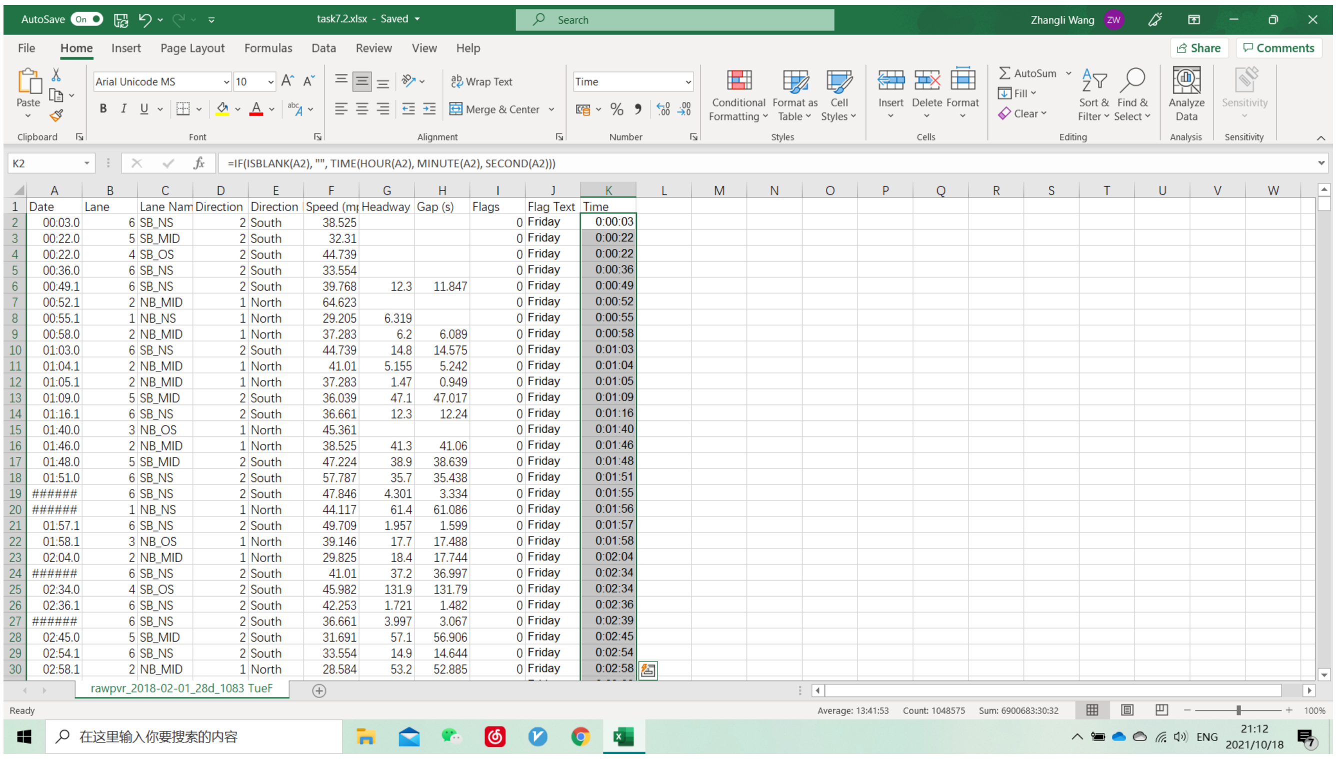
rawpvr_2018-02-01_28d_1083 TueF

Count: 1048575

3. Then we want to add a new column "Time" that stores the corresponding time in the "Date" column, apply this function

```
=IF(ISBLANK(A2), "", TIME(HOUR(A2), MINUTE(A2), SECOND(A2)))
```

to the column to extract hour, minute, and second from the corresponding date. Then, update the type of the column from default "General" to "Time".



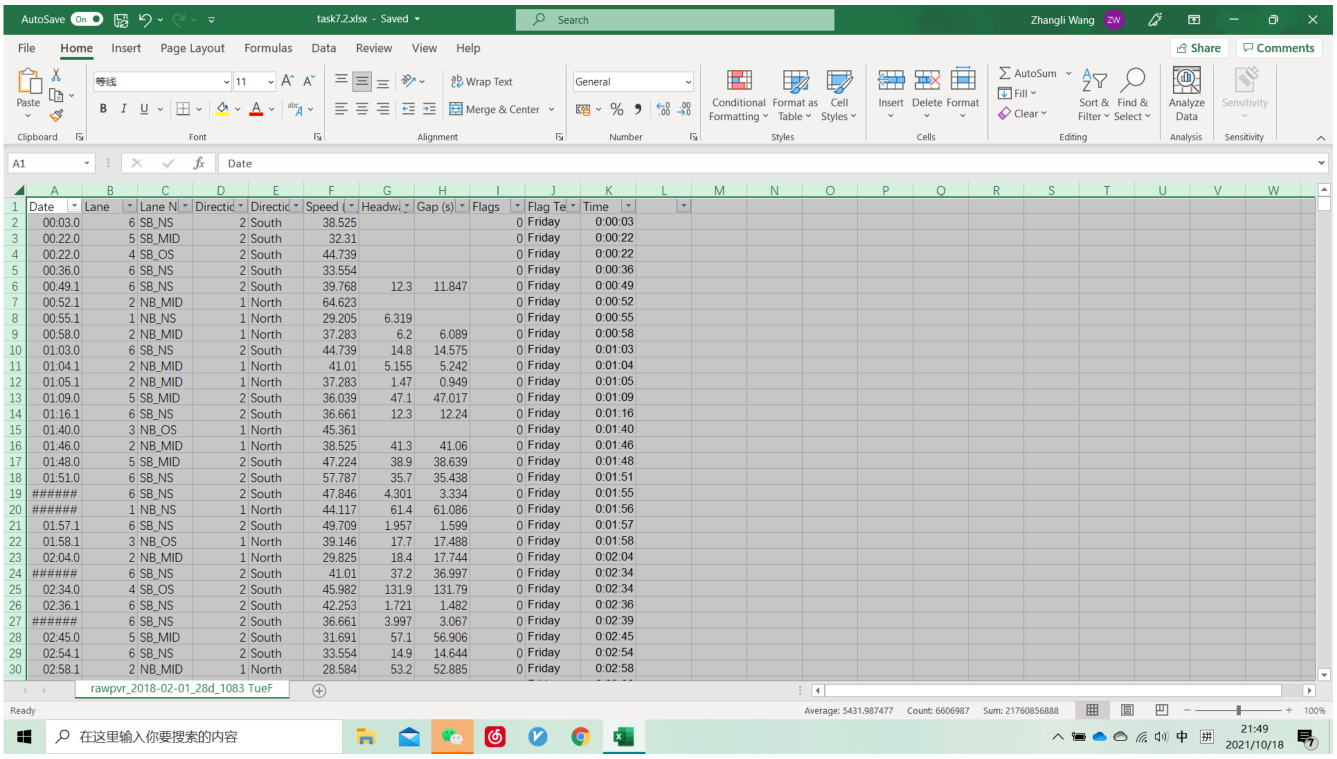
K2 =IF(ISBLANK(A2), "", TIME(HOUR(A2), MINUTE(A2), SECOND(A2)))

	A	B	C	D	E	F	G	H	I	J	K
1	Date	Lane	Lane Nam	Direction	Direction	Speed (mı	Headway	Gap (s)	Flags	Flag Text	Time
2	00:03.0	6	SB_NS	2	South	38.525			0	Friday	0:00:03
3	00:22.0	5	SB_MID	2	South	32.31			0	Friday	0:00:22
4	00:22.0	4	SB_OS	2	South	44.739			0	Friday	0:00:22
5	00:36.0	6	SB_NS	2	South	33.554			0	Friday	0:00:36
6	00:49.1	6	SB_NS	2	South	39.768	12.3	11.847	0	Friday	0:00:49
7	00:52.1	2	NB_MID	1	North	64.623			0	Friday	0:00:52
8	00:55.1	1	NB_NS	1	North	29.205	6.319		0	Friday	0:00:55
9	00:58.0	2	NB_MID	1	North	37.283	6.2	6.089	0	Friday	0:00:58
10	01:03.0	6	SB_NS	2	South	44.739	14.8	14.575	0	Friday	0:01:03
11	01:04.1	2	NB_MID	1	North	41.01	5.155	5.242	0	Friday	0:01:04
12	01:05.1	2	NB_MID	1	North	37.283	1.47	0.949	0	Friday	0:01:05
13	01:09.0	5	SB_MID	2	South	36.039	47.1	47.017	0	Friday	0:01:09
14	01:16.1	6	SB_NS	2	South	36.661	12.3	12.24	0	Friday	0:01:16
15	01:40.0	3	NB_OS	1	North	45.361			0	Friday	0:01:40
16	01:46.0	2	NB_MID	1	North	38.525	41.3	41.06	0	Friday	0:01:46
17	01:48.0	5	SB_MID	2	South	47.224	38.9	38.639	0	Friday	0:01:48
18	01:51.0	6	SB_NS	2	South	57.787	35.7	35.438	0	Friday	0:01:51
19	######	6	SB_NS	2	South	47.846	4.301	3.334	0	Friday	0:01:55
20	######	1	NB_NS	1	North	44.117	61.4	61.086	0	Friday	0:01:56
21	01:57.1	6	SB_NS	2	South	49.709	1.957	1.599	0	Friday	0:01:57
22	01:58.1	3	NB_OS	1	North	39.146	17.7	17.488	0	Friday	0:01:58
23	02:04.0	2	NB_MID	1	North	29.825	18.4	17.744	0	Friday	0:02:04
24	######	6	SB_NS	2	South	41.01	37.2	36.997	0	Friday	0:02:34
25	02:34.0	4	SB_OS	2	South	45.982	131.9	131.79	0	Friday	0:02:34
26	02:36.1	6	SB_NS	2	South	42.253	1.721	1.482	0	Friday	0:02:36
27	######	6	SB_NS	2	South	36.661	3.997	3.067	0	Friday	0:02:39
28	02:45.0	5	SB_MID	2	South	31.691	57.1	56.906	0	Friday	0:02:45
29	02:54.1	6	SB_NS	2	South	33.554	14.9	14.644	0	Friday	0:02:54
30	02:58.1	2	NB_MID	1	North	28.584	53.2	52.885	0	Friday	0:02:58

rawpvr_2018-02-01_28d_1083 TueF

Average: 13:41:53 Count: 1048575 Sum: 6900683:30:32 100%

4. Select the entire table, press `Ctrl` + `Shift` + `L` to initialize the filters. Then we can apply filtering to the table.



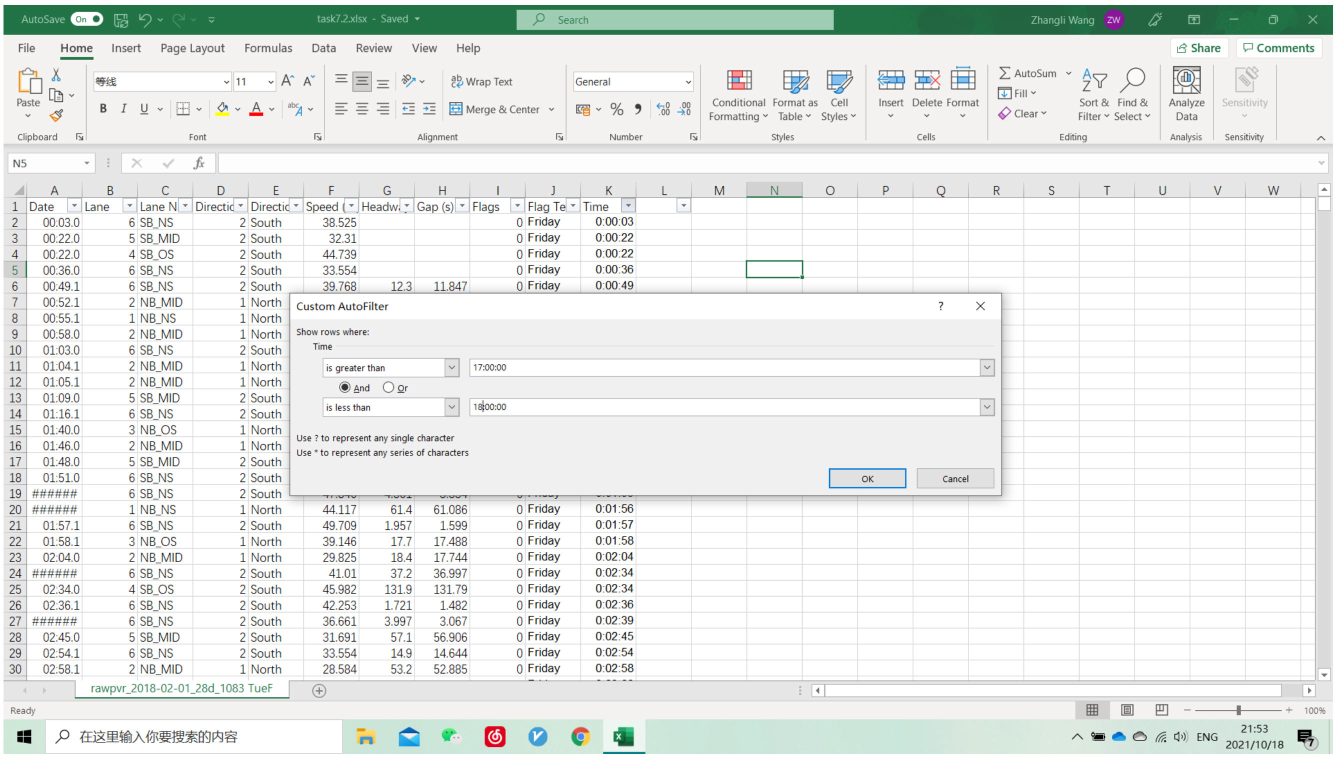
A1 Date

	A	B	C	D	E	F	G	H	I	J	K
1	Date	Lane	Lane N	Directic	Directic	Speed (	Headwa	Gap (s)	Flags	Flag Te	Time
2	00:03.0	6	SB_NS	2	South	38.525			0	Friday	0:00:03
3	00:22.0	5	SB_MID	2	South	32.31			0	Friday	0:00:22
4	00:22.0	4	SB_OS	2	South	44.739			0	Friday	0:00:22
5	00:36.0	6	SB_NS	2	South	33.554			0	Friday	0:00:36
6	00:49.1	6	SB_NS	2	South	39.768	12.3	11.847	0	Friday	0:00:49
7	00:52.1	2	NB_MID	1	North	64.623			0	Friday	0:00:52
8	00:55.1	1	NB_NS	1	North	29.205	6.319		0	Friday	0:00:55
9	00:58.0	2	NB_MID	1	North	37.283	6.2	6.089	0	Friday	0:00:58
10	01:03.0	6	SB_NS	2	South	44.739	14.8	14.575	0	Friday	0:01:03
11	01:04.1	2	NB_MID	1	North	41.01	5.155	5.242	0	Friday	0:01:04
12	01:05.1	2	NB_MID	1	North	37.283	1.47	0.949	0	Friday	0:01:05
13	01:09.0	5	SB_MID	2	South	36.039	47.1	47.017	0	Friday	0:01:09
14	01:16.1	6	SB_NS	2	South	36.661	12.3	12.24	0	Friday	0:01:16
15	01:40.0	3	NB_OS	1	North	45.361			0	Friday	0:01:40
16	01:46.0	2	NB_MID	1	North	38.525	41.3	41.06	0	Friday	0:01:46
17	01:48.0	5	SB_MID	2	South	47.224	38.9	38.639	0	Friday	0:01:48
18	01:51.0	6	SB_NS	2	South	57.787	35.7	35.438	0	Friday	0:01:51
19	######	6	SB_NS	2	South	47.846	4.301	3.334	0	Friday	0:01:55
20	######	1	NB_NS	1	North	44.117	61.4	61.086	0	Friday	0:01:56
21	01:57.1	6	SB_NS	2	South	49.709	1.957	1.599	0	Friday	0:01:57
22	01:58.1	3	NB_OS	1	North	39.146	17.7	17.488	0	Friday	0:01:58
23	02:04.0	2	NB_MID	1	North	29.825	18.4	17.744	0	Friday	0:02:04
24	######	6	SB_NS	2	South	41.01	37.2	36.997	0	Friday	0:02:34
25	02:34.0	4	SB_OS	2	South	45.982	131.9	131.79	0	Friday	0:02:34
26	02:36.1	6	SB_NS	2	South	42.253	1.721	1.482	0	Friday	0:02:36
27	######	6	SB_NS	2	South	36.661	3.997	3.067	0	Friday	0:02:39
28	02:45.0	5	SB_MID	2	South	31.691	57.1	56.906	0	Friday	0:02:45
29	02:54.1	6	SB_NS	2	South	33.554	14.9	14.644	0	Friday	0:02:54
30	02:58.1	2	NB_MID	1	North	28.584	53.2	52.885	0	Friday	0:02:58

rawpvr_2018-02-01_28d_1083 TueF

Average: 5431.987477 Count: 6606987 Sum: 21760856888 100%

5. Filter the sheet that the time is between 17:00 and 18:00, the direction is 1, and the day of the week is Friday.
   Customly filter the 'Time' to be between 17:00 and 18:00.



	Date	Lane	Lane N	Directic	Directic	Speed (	Headw	Gap (s)	Flags	Flag Te	Time
2	00:03.0	6	SB_NS	2	South	38.525			0	Friday	0:00:03
3	00:22.0	5	SB_MID	2	South	32.31			0	Friday	0:00:22
4	00:22.0	4	SB_OS	2	South	44.739			0	Friday	0:00:22
5	00:36.0	6	SB_NS	2	South	33.554			0	Friday	0:00:36
6	00:49.1	6	SB_NS	2	South	39.768	12.3	11.847	0	Friday	0:00:49
7	00:52.1	2	NB_MID	1	North						
8	00:55.1	1	NB_NS	1	North						
9	00:58.0	2	NB_MID	1	North						
10	01:03.0	6	SB_NS	2	South						
11	01:04.1	2	NB_MID	1	North						
12	01:05.1	2	NB_MID	1	North						
13	01:09.0	5	SB_MID	2	South						
14	01:16.1	6	SB_NS	2	South						
15	01:40.0	3	NB_OS	1	North						
16	01:46.0	2	NB_MID	1	North						
17	01:48.0	5	SB_MID	2	South						
18	01:51.0	6	SB_NS	2	South						
19	######	6	SB_NS	2	South						
20	######	1	NB_NS	1	North	44.117	61.4	61.086	0	Friday	0:01:56
21	01:57.1	6	SB_NS	2	South	49.709	1.957	1.599	0	Friday	0:01:57
22	01:58.1	3	NB_OS	1	North	39.146	17.7	17.488	0	Friday	0:01:58
23	02:04.0	2	NB_MID	1	North	29.825	18.4	17.744	0	Friday	0:02:04
24	######	6	SB_NS	2	South	41.01	37.2	36.997	0	Friday	0:02:34
25	02:34.0	4	SB_OS	2	South	45.982	131.9	131.79	0	Friday	0:02:34
26	02:36.1	6	SB_NS	2	South	42.253	1.721	1.482	0	Friday	0:02:36
27	######	6	SB_NS	2	South	36.661	3.997	3.067	0	Friday	0:02:39
28	02:45.0	5	SB_MID	2	South	31.691	57.1	56.906	0	Friday	0:02:45
29	02:54.1	6	SB_NS	2	South	33.554	14.9	14.644	0	Friday	0:02:54
30	02:58.1	2	NB_MID	1	North	28.584	53.2	52.885	0	Friday	0:02:58

Custom AutoFilter

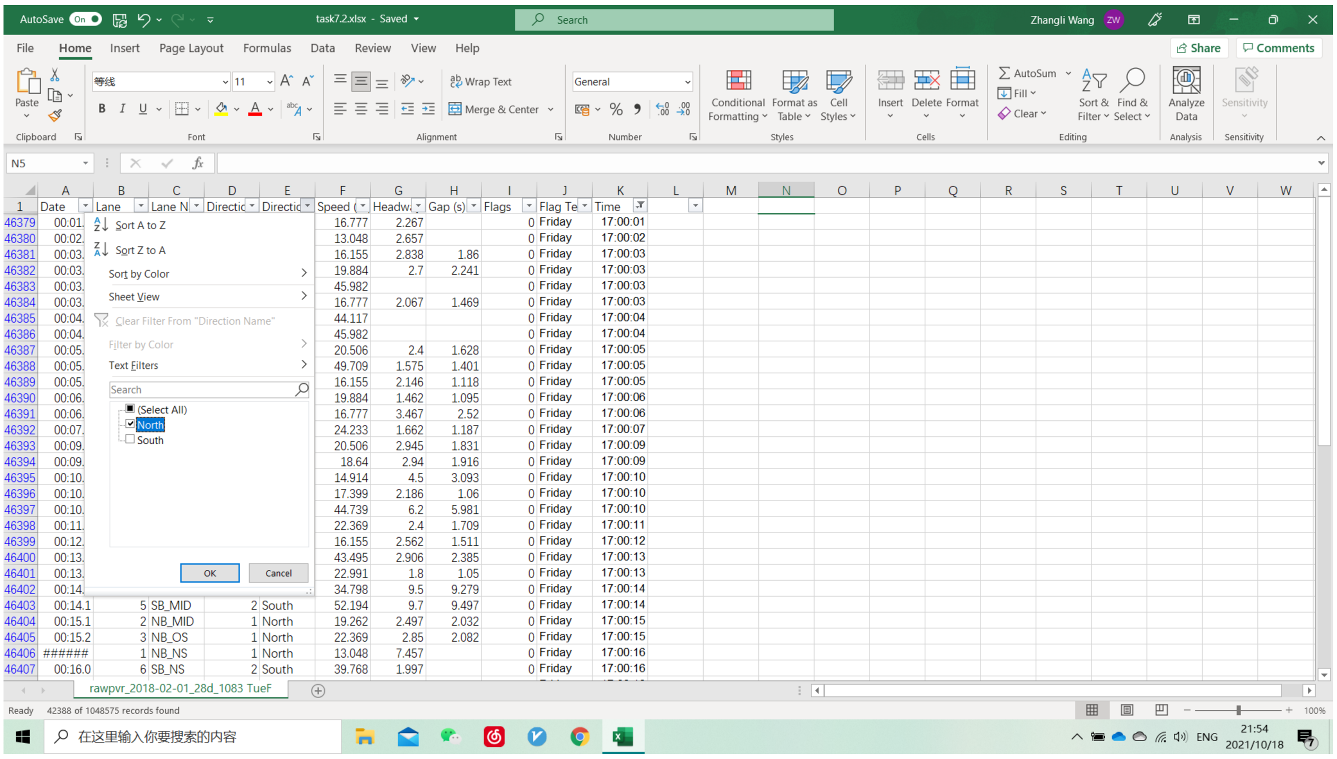
Show rows where:
Time
is greater than 17:00:00
[✓ And] Or
is less than 18:00:00

Use ? to represent any single character
Use * to represent any series of characters

OK Cancel

rawpvr_2018-02-01_28d_1083 TueF

Filter the 'Direction Name' as 'North'.



	Date	Lane	Lane N	Directic	Directic	Speed (	Headw	Gap (s)	Flags	Flag Te	Time
46379	00:01.					16.777	2.267		0	Friday	17:00:01
46380	00:02.					13.048	2.657		0	Friday	17:00:02
46381	00:03.					16.155	2.838	1.86	0	Friday	17:00:03
46382	00:03.					19.884	2.7	2.241	0	Friday	17:00:03
46383	00:03.					45.982			0	Friday	17:00:03
46384	00:03.					16.777	2.067	1.469	0	Friday	17:00:03
46385	00:04.					44.117			0	Friday	17:00:04
46386	00:04.					45.982			0	Friday	17:00:04
46387	00:05.					20.506	2.4	1.628	0	Friday	17:00:05
46388	00:05.					49.709	1.575	1.401	0	Friday	17:00:05
46389	00:05.					16.155	2.146	1.118	0	Friday	17:00:05
46390	00:06.					19.884	1.462	1.095	0	Friday	17:00:06
46391	00:06.					16.777	3.467	2.52	0	Friday	17:00:06
46392	00:07.					24.233	1.662	1.187	0	Friday	17:00:07
46393	00:09.					20.506	2.945	1.831	0	Friday	17:00:09
46394	00:09.					18.64	2.94	1.916	0	Friday	17:00:09
46395	00:10.					14.914	4.5	3.093	0	Friday	17:00:10
46396	00:10.					17.399	2.186	1.06	0	Friday	17:00:10
46397	00:10.					44.739	6.2	5.981	0	Friday	17:00:10
46398	00:11.					22.369	2.4	1.709	0	Friday	17:00:11
46399	00:12.					16.155	2.562	1.511	0	Friday	17:00:12
46400	00:13.					43.495	2.906	2.385	0	Friday	17:00:13
46401	00:13.					22.991	1.8	1.05	0	Friday	17:00:13
46402	00:14.					34.798	9.5	9.279	0	Friday	17:00:14
46403	00:14.1	5	SB_MID	2	South	52.194	9.7	9.497	0	Friday	17:00:14
46404	00:15.1	2	NB_MID	1	North	19.262	2.497	2.032	0	Friday	17:00:15
46405	00:15.2	3	NB_OS	1	North	22.369	2.85	2.082	0	Friday	17:00:15
46406	######	1	NB_NS	1	North	13.048	7.457		0	Friday	17:00:16
46407	00:16.0	6	SB_NS	2	South	39.768	1.997		0	Friday	17:00:16

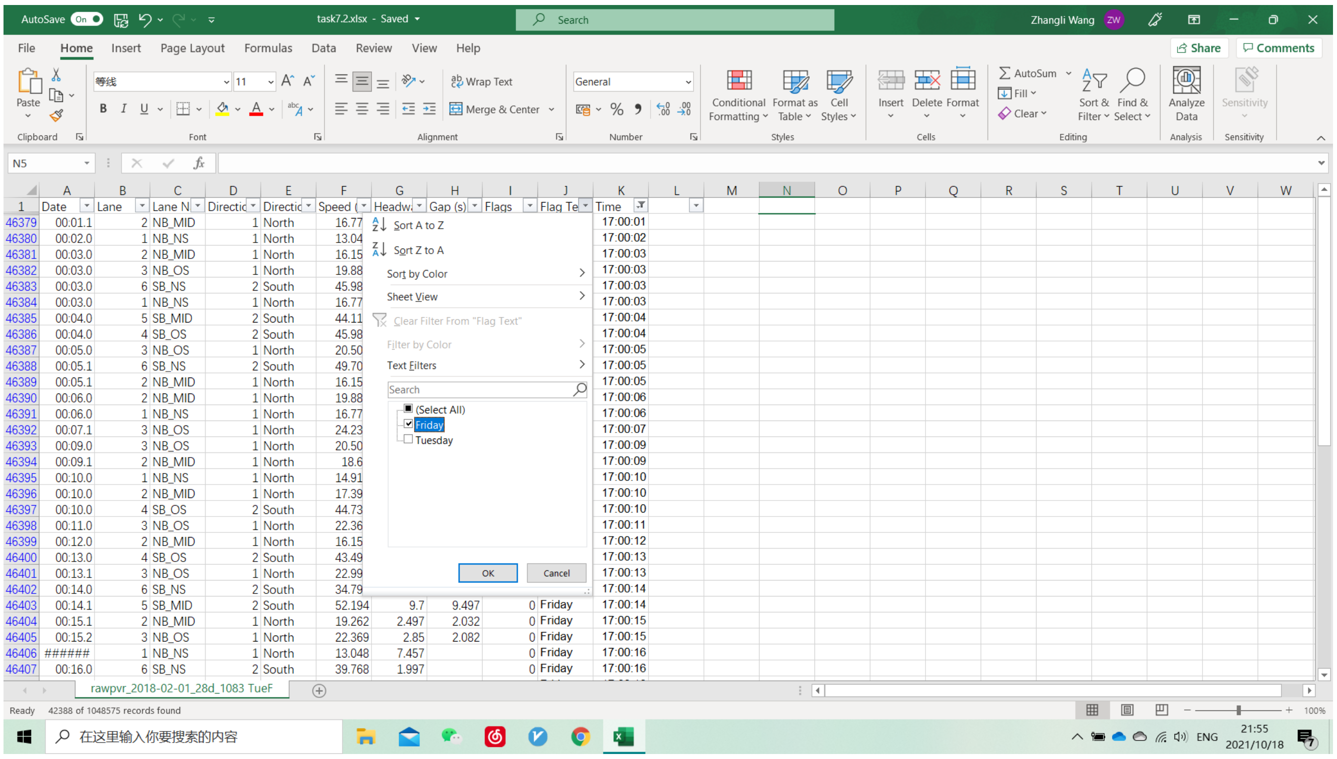
(Select All)
[✓ North]
South

OK Cancel

rawpvr_2018-02-01_28d_1083 TueF

Ready 42388 of 1048575 records found

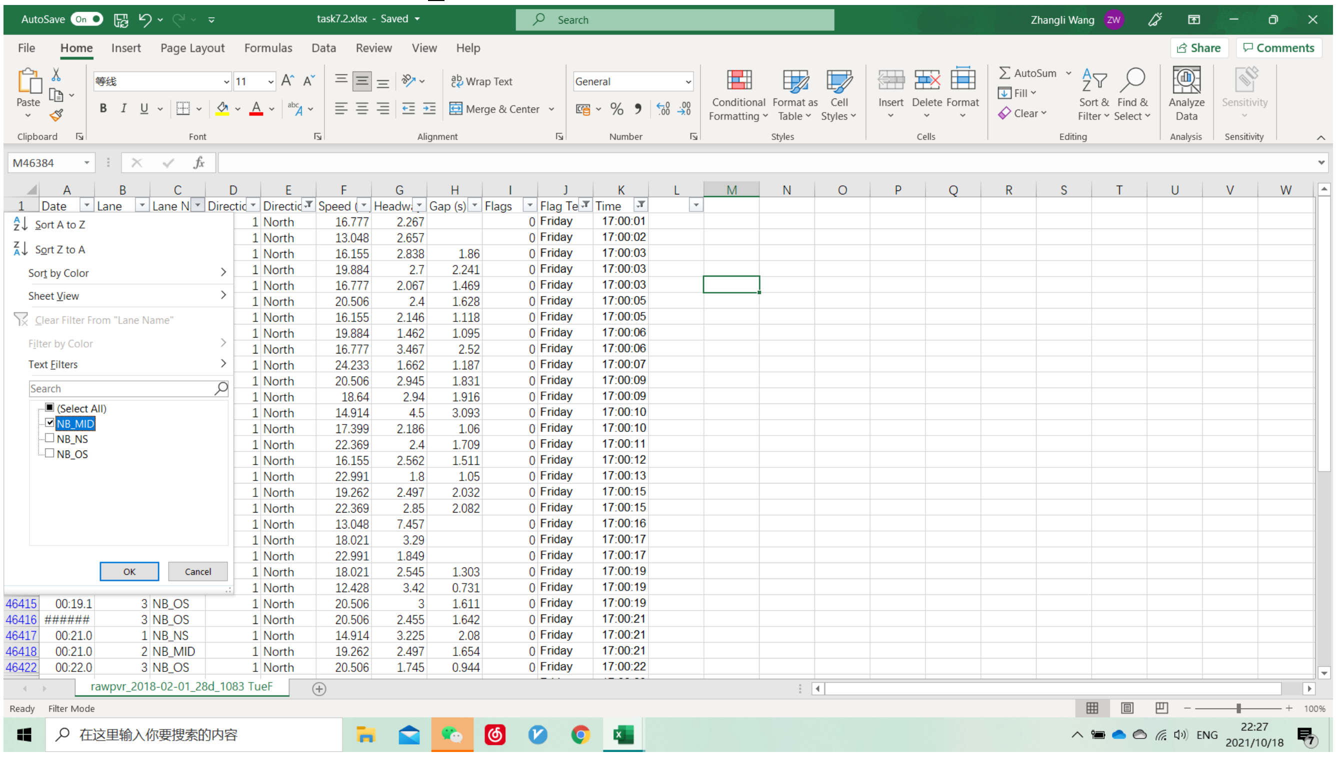
Filter the 'Flag Text' as Friday.



1	Date	Lane	Lane N	Directio	Directio	Speed (	Headw	Gap (s)	Flags	Flag Te	Time
46379	00:01.1	2	NB_MID	1	North	16.77					17:00:01
46380	00:02.0	1	NB_NS	1	North	13.04					17:00:02
46381	00:03.0	2	NB_MID	1	North	16.15					17:00:03
46382	00:03.0	3	NB_OS	1	North	19.88					17:00:03
46383	00:03.0	6	SB_NS	2	South	45.98					17:00:03
46384	00:03.0	1	NB_NS	1	North	16.77					17:00:03
46385	00:04.0	5	SB_MID	2	South	44.11					17:00:04
46386	00:04.0	4	SB_OS	2	South	45.98					17:00:04
46387	00:05.0	3	NB_OS	1	North	20.50					17:00:05
46388	00:05.1	6	SB_NS	2	South	49.70					17:00:05
46389	00:05.1	2	NB_MID	1	North	16.15					17:00:05
46390	00:06.0	2	NB_MID	1	North	19.88					17:00:06
46391	00:06.0	1	NB_NS	1	North	16.77					17:00:06
46392	00:07.1	3	NB_OS	1	North	24.23					17:00:07
46393	00:09.0	3	NB_OS	1	North	20.50					17:00:09
46394	00:09.1	2	NB_MID	1	North	18.6					17:00:09
46395	00:10.0	1	NB_NS	1	North	14.91					17:00:10
46396	00:10.0	2	NB_MID	1	North	17.39					17:00:10
46397	00:10.0	4	SB_OS	2	South	44.73					17:00:10
46398	00:11.0	3	NB_OS	1	North	22.36					17:00:11
46399	00:12.0	2	NB_MID	1	North	16.15					17:00:12
46400	00:13.0	4	SB_OS	2	South	43.49					17:00:13
46401	00:13.1	3	NB_OS	1	North	22.99					17:00:13
46402	00:14.0	6	SB_NS	2	South	34.79					17:00:14
46403	00:14.1	5	SB_MID	2	South	52.194	9.7	9.497	0	Friday	17:00:14
46404	00:15.1	2	NB_MID	1	North	19.262	2.497	2.032	0	Friday	17:00:15
46405	00:15.2	3	NB_OS	1	North	22.369	2.85	2.082	0	Friday	17:00:15
46406	######	1	NB_NS	1	North	13.048	7.457		0	Friday	17:00:16
46407	00:16.0	6	SB_NS	2	South	39.768	1.997		0	Friday	17:00:16

Ready 42388 of 1048575 records found

6. Then, filter the 'Lane Name' to 'NB_MID'.



1	Date	Lane	Lane N	Directio	Directio	Speed (	Headw	Gap (s)	Flags	Flag Te	Time
				1	North	16.777	2.267		0	Friday	17:00:01
				1	North	13.048	2.657		0	Friday	17:00:02
				1	North	16.155	2.838	1.86	0	Friday	17:00:03
				1	North	19.884	2.7	2.241	0	Friday	17:00:03
				1	North	16.777	2.067	1.469	0	Friday	17:00:03
				1	North	20.506	2.4	1.628	0	Friday	17:00:05
				1	North	16.155	2.146	1.118	0	Friday	17:00:05
				1	North	19.884	1.462	1.095	0	Friday	17:00:06
				1	North	16.777	3.467	2.52	0	Friday	17:00:06
				1	North	24.233	1.662	1.187	0	Friday	17:00:07
				1	North	20.506	2.945	1.831	0	Friday	17:00:09
				1	North	18.64	2.94	1.916	0	Friday	17:00:09
				1	North	14.914	4.5	3.093	0	Friday	17:00:10
				1	North	17.399	2.186	1.06	0	Friday	17:00:10
				1	North	22.369	2.4	1.709	0	Friday	17:00:11
				1	North	16.155	2.562	1.511	0	Friday	17:00:12
				1	North	22.991	1.8	1.05	0	Friday	17:00:13
				1	North	19.262	2.497	2.032	0	Friday	17:00:15
				1	North	22.369	2.85	2.082	0	Friday	17:00:15
				1	North	13.048	7.457		0	Friday	17:00:16
				1	North	18.021	3.29		0	Friday	17:00:17
				1	North	22.991	1.849		0	Friday	17:00:17
				1	North	18.021	2.545	1.303	0	Friday	17:00:19
				1	North	12.428	3.42	0.731	0	Friday	17:00:19
46415	00:19.1	3	NB_OS	1	North	20.506	3	1.611	0	Friday	17:00:19
46416	######	3	NB_OS	1	North	20.506	2.455	1.642	0	Friday	17:00:21
46417	00:21.0	1	NB_NS	1	North	14.914	3.225	2.08	0	Friday	17:00:21
46418	00:21.0	2	NB_MID	1	North	19.262	2.497	1.654	0	Friday	17:00:21
46422	00:22.0	3	NB_OS	1	North	20.506	1.745	0.944	0	Friday	17:00:22

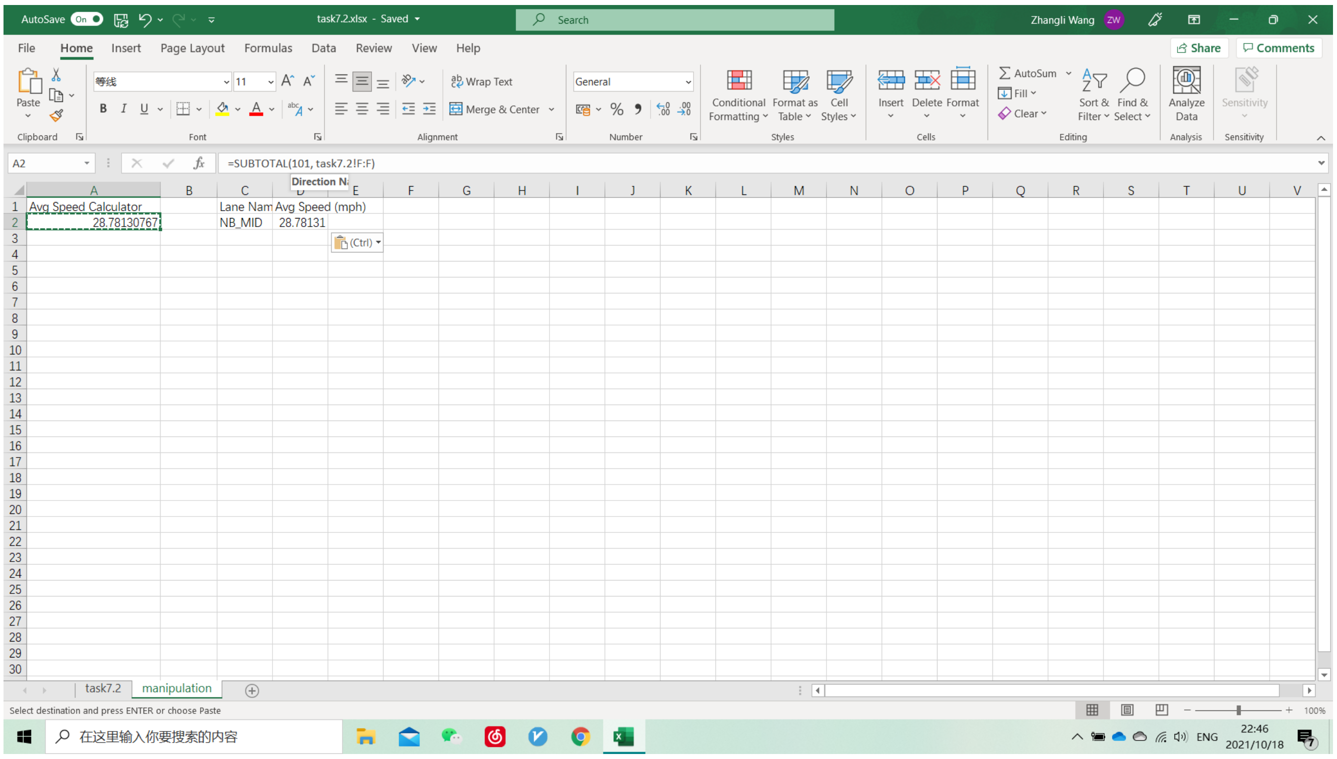
Ready Filter Mode

7. Create a sheet named 'manipulation' for storing the results of calculating the average speeds and other results. Create a column named 'Avg Speed Calculator' as the label of calculating function

```
=SUBTOTAL(101, task7.2!F:F)
```

to calculate the average speed for 'NB_MID' according to the filtering. Note that the first arguement is the code of average function with ignoring hidden values, we use this one because filtering hide rows.

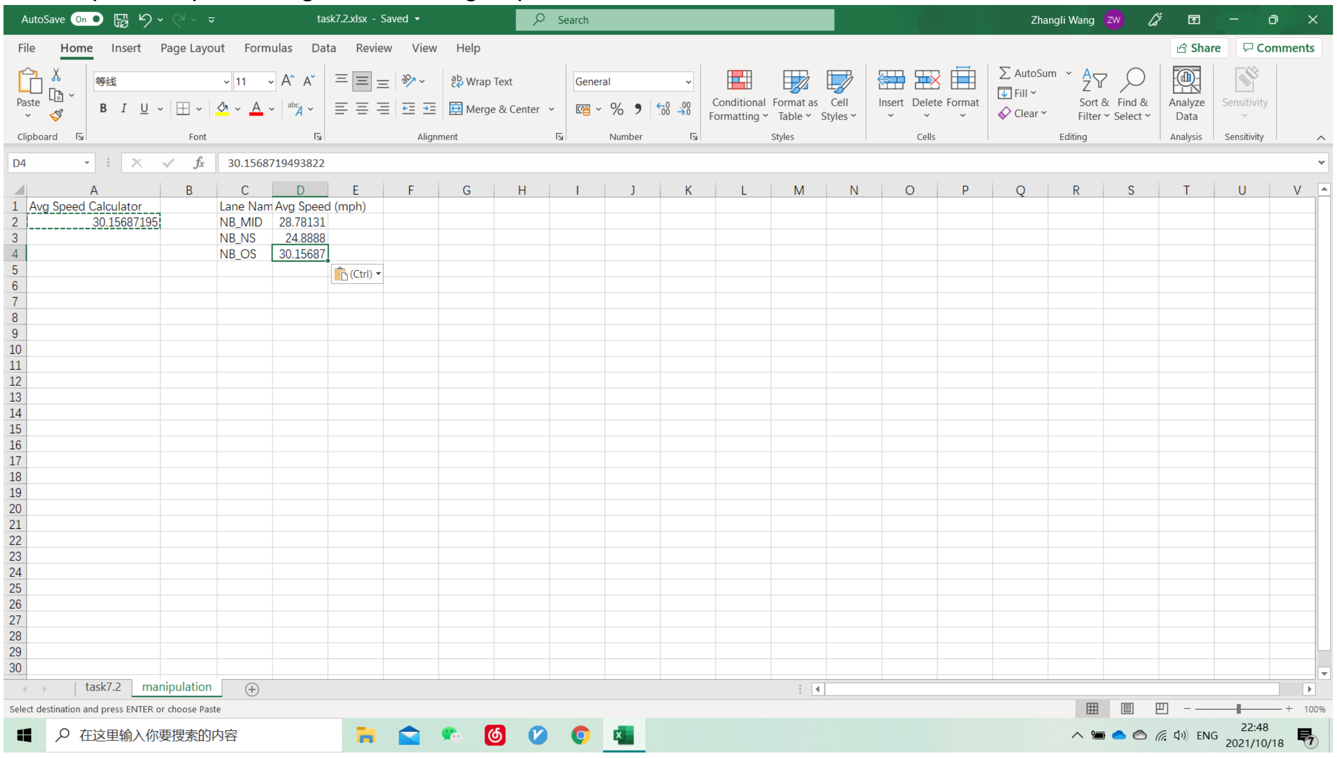
And we also add two columns 'Avg Speed (mph)' and 'Lane Name' for storing the average speeds for lanes and the lane names, respectively. Then we execute the function, copy and paste the value by plain text to the corresponding place in the column 'Avg Speed (mph)'. We do this because the result of this function changes according to filtering.



A2 =SUBTOTAL(101, task7.2!F:F)

	A	B	C	D
1	Avg Speed Calculator		Lane Nam	Avg Speed (mph)
2	28.78130767		NB_MID	28.78131

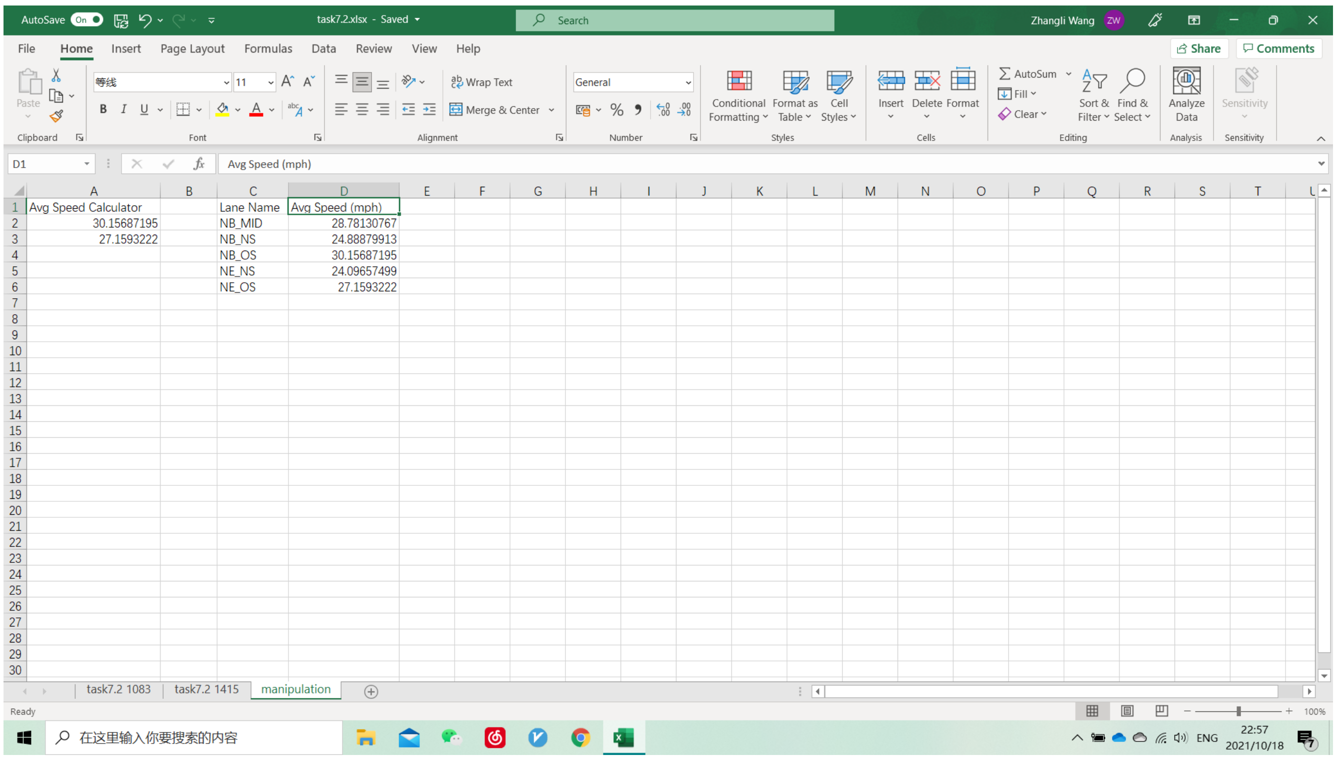
8. Now repeat step 6, 7 to get the average speeds of all lanes.



D4 30.1568719493822

	A	B	C	D
1	Avg Speed Calculator		Lane Nam	Avg Speed (mph)
2	30.15687195		NB_MID	28.78131
3			NB_NS	24.8888
4			NB_OS	30.15687

9. Load the dataset of site 1415 in a newly created sheet named `task7.2 1415` and repeat steps 1 to 8 for this sheet. After all steps are finished, we can acquire the average values of the five north lanes.



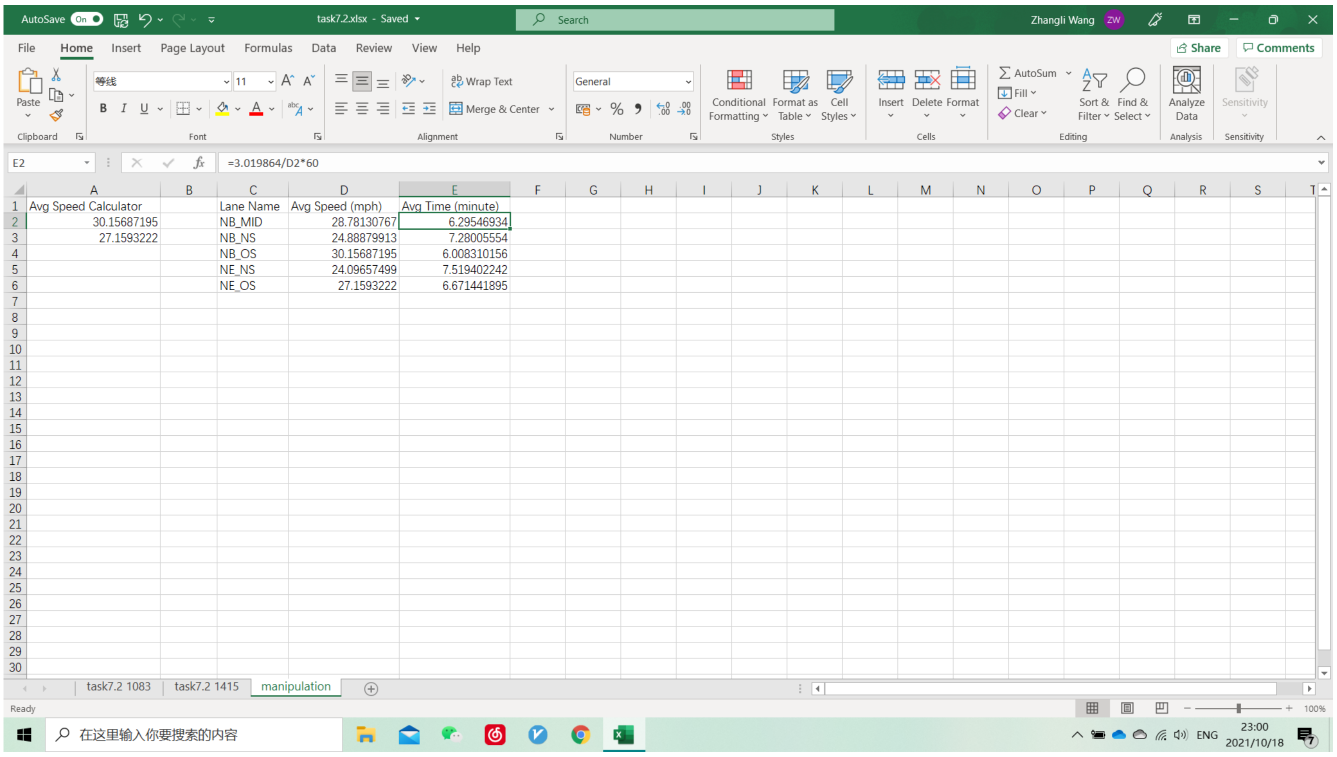
D1 Avg Speed (mph)

	A	B	C	D
1	Avg Speed Calculator		Lane Name	Avg Speed (mph)
2	30.15687195		NB_MID	28.78130767
3	27.1593222		NB_NS	24.88879913
4			NB_OS	30.15687195
5			NE_NS	24.09657499
6			NE_OS	27.1593222

10. Now we can calculate the average time with the given distance and the speed with the function

=3.019864/D2*60

.



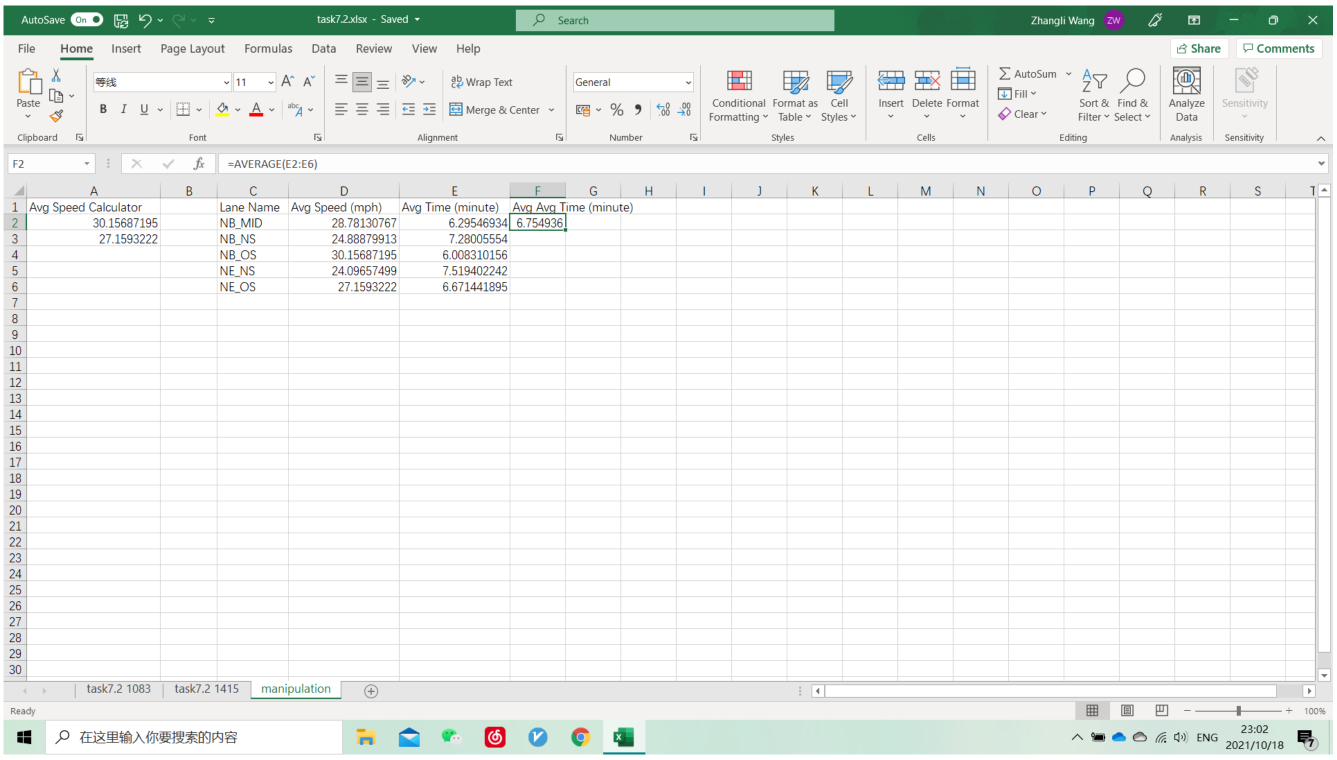
E2 =3.019864/D2*60

	A	B	C	D	E
1	Avg Speed Calculator		Lane Name	Avg Speed (mph)	Avg Time (minute)
2	30.15687195		NB_MID	28.78130767	6.29546934
3	27.1593222		NB_NS	24.88879913	7.28005554
4			NB_OS	30.15687195	6.008310156
5			NE_NS	24.09657499	7.519402242
6			NE_OS	27.1593222	6.671441895

Finally, we can compute the average over the average time by the function

=AVERAGE(E2:E6)

.



F2 `=AVERAGE(E2:E6)`

	A	B	C	D	E	F
1	Avg Speed Calculator		Lane Name	Avg Speed (mph)	Avg Time (minute)	Avg Avg Time (minute)
2	30.15687195		NB_MID	28.78130767	6.29546934	6.754936
3	27.1593222		NB_NS	24.88879913	7.28005554	
4			NB_OS	30.15687195	6.008310156	
5			NE_NS	24.09657499	7.519402242	
6			NE_OS	27.1593222	6.671441895	

Comparition of the Two Technologies

Both of the two technologies have the ability to solve the tasks, they both have built-in functions for the same purposes in terms of data manipulation. And they can both manipulate the tabular dataset by filtering according to columns and aggregating according to rows.

To discuss about the differences, the manipulation logics are different. For python, the manipulation is realized by codes. For Excel, the manipulation is realized by the combination of tools in the interface, such as the tool for applying filters to the 'Lane Name' column, and the built-in functions. Python codes can be compiled statically while every operation in Excel is executed instantaneously.

For python, one advantage is that the process of manipulation is more interpretable and reviewable for a programmer. And using Jupyter Notebook to program this coursework can help me to divide operations into stages and add explanation everywhere I want. For example, for every task, I can split my manipulation operations in different code blocks by mini goals. Another advantage can be that the library pandas imported in python has efficient built-in functions such as `loc` that can automatically optimize the data manipulation operations. But the disadvantage is that bugs can possibly occur everywhere in my codes and the intermediate dataset cannot be entirely displayed during the manipulation.

For Excel, the advantage is that the interface is intuitive and can display the data in a proper tabular structure and the drawing of charts is also very convenient. And for manipulating an entire column, Excel can help me with it with its tools conveniently rather than coding for loops manually. But the disadvantages are also significant. Because most of my manipulations are functions, their dependencies result in everytime I change a part of the Excel file, all the functions are re-calculated, which is time consuming. Another disadvantage can be that using tools in the interface and built-in functions in combination can bring incompatibility in some cases. For example, for calculating the average speed for different lanes, I have to manually filter the table for several times and then record the values manually. When the dataset become larger and more complex, each operation executed in Excel can take much longer time due to the deep function dependencies and large quantity of calculations. And manually manipulate some variables can be cumbersome.