project

April 15, 2021

1 CIV1498 - Introduction to Data Science

1.1 Project - Toronto Bike Share

1.2 PART I: Data Wrangling and Cleaning

1.2.1 By: Gneiss Data (Greig Knox and Yoko Yanagimura)

1.3 0. Setup Notebook

[1]:
```
# Import 3rd party libraries
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pylab as plt
import pytz
import fuzzywuzzy
from fuzzywuzzy import process
import chardet
import string
from datetime import datetime
import matplotlib.dates as mdates
from datetime import date
from matplotlib.ticker import FuncFormatter
from matplotlib.dates import MonthLocator, DateFormatter

# Configure Notebook
%matplotlib inline
plt.style.use('fivethirtyeight')
sns.set_context("notebook")
import warnings
warnings.filterwarnings('ignore')

# Centre all the charts displayed in this notebook
from IPython.core.display import HTML
HTML("""
<style>
.output_png {
```

```
display: table-cell;
text-align: center;
vertical-align: middle;
}
</style>
""")
```

[1]:
```
<IPython.core.display.HTML object>
```

2 1. Weather Data - Importing Data

In this section, a dataframe is created containing the weather data from January 2017 to December 2020 collected at the Toronto City Center weather station (Weather Station ID 6158359). The raw weather data was provided in csv format.

[2]:
```
#Get list of Weather Data File Names contained in the repository
weather_filenames = [filename for filename in os.listdir() if '6158359' in␣
↪filename]
print(weather_filenames[0:5])

# Check the format of the weather data
df_weather_data = pd.read_csv(weather_filenames[0])
df_weather_data.head(10)
```

```
['en_climate_hourly_ON_6158359_01-2017_P1H.csv',
'en_climate_hourly_ON_6158359_01-2018_P1H.csv',
'en_climate_hourly_ON_6158359_01-2019_P1H.csv',
'en_climate_hourly_ON_6158359_01-2020_P1H.csv',
'en_climate_hourly_ON_6158359_02-2017_P1H.csv']
```

[2]:

	Longitude (x)	Latitude (y)	Station Name	Climate ID	\
0	-79.4	43.63	TORONTO CITY CENTRE	6158359	
1	-79.4	43.63	TORONTO CITY CENTRE	6158359	
2	-79.4	43.63	TORONTO CITY CENTRE	6158359	
3	-79.4	43.63	TORONTO CITY CENTRE	6158359	
4	-79.4	43.63	TORONTO CITY CENTRE	6158359	
5	-79.4	43.63	TORONTO CITY CENTRE	6158359	
6	-79.4	43.63	TORONTO CITY CENTRE	6158359	
7	-79.4	43.63	TORONTO CITY CENTRE	6158359	
8	-79.4	43.63	TORONTO CITY CENTRE	6158359	
9	-79.4	43.63	TORONTO CITY CENTRE	6158359	

	Date/Time	Year	Month	Day	Time	Temp (°C)	…	Wind Spd Flag	\
0	2017-01-01 00:00	2017	1	1	00:00	1.5	…	NaN	
1	2017-01-01 01:00	2017	1	1	01:00	1.5	…	NaN	
2	2017-01-01 02:00	2017	1	1	02:00	1.0	…	NaN	
3	2017-01-01 03:00	2017	1	1	03:00	1.2	…	NaN	

```
4  2017-01-01 04:00  2017      1    1  04:00        1.3  …            NaN
5  2017-01-01 05:00  2017      1    1  05:00        1.0  …            NaN
6  2017-01-01 06:00  2017      1    1  06:00        0.7  …            NaN
7  2017-01-01 07:00  2017      1    1  07:00        0.0  …            NaN
8  2017-01-01 08:00  2017      1    1  08:00       -0.3  …            NaN
9  2017-01-01 09:00  2017      1    1  09:00       -0.1  …            NaN

   Visibility (km)  Visibility Flag  Stn Press (kPa)  Stn Press Flag  Hmdx  \
0             16.1              NaN            99.81             NaN   NaN
1             16.1              NaN           100.01             NaN   NaN
2             16.1              NaN           100.14             NaN   NaN
3             16.1              NaN           100.32             NaN   NaN
4             16.1              NaN           100.48             NaN   NaN
5             16.1              NaN           100.55             NaN   NaN
6             16.1              NaN           100.65             NaN   NaN
7             16.1              NaN           100.79             NaN   NaN
8             16.1              NaN           100.93             NaN   NaN
9             16.1              NaN           101.06             NaN   NaN

  Hmdx Flag  Wind Chill  Wind Chill Flag  Weather
0       NaN         NaN              NaN      NaN
1       NaN         NaN              NaN      NaN
2       NaN         NaN              NaN      NaN
3       NaN         NaN              NaN      NaN
4       NaN         NaN              NaN      NaN
5       NaN         NaN              NaN      NaN
6       NaN         NaN              NaN      NaN
7       NaN        -6.0              NaN      NaN
8       NaN        -6.0              NaN      NaN
9       NaN        -6.0              NaN      NaN

[10 rows x 28 columns]
```

Now, the weather data is concatenated into a single dataframe

[3]:
```
#Concatenate the weather data from 2017 to 2020 into a single dataframe
df_weather_data = pd.DataFrame()

for file in weather_filenames:
    df_weather_data = pd.concat([df_weather_data,pd.read_csv(file)])
```

3 2. Weather Data - Data Wrangling

Because the weather data included in the repository are collected from a single weather station, the station information including the station name, station/climate ID, latitude and longitude of the station location, is redundant data and can be removed from the dataframe to reduce the number of columns. We show below what this will look like applied to one of the weather dataset

extracted from en_climate_hourly_ON_6158359_08-2020_P1H.csv. The number of column in the dataframe is reduced from 28 to 24.

[4]:
```
#clean column headers
def clean_special_charaters(a_string):
    output = ''
    a_string = a_string.lower()
    for character in a_string:
        if character == ' ' or character == '_':
            output+='_'
        elif character.isalnum():
            output+=character
    return output

punctuations = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

#remove all punctuation from column headers
df_weather_data.columns = [ clean_special_charaters(header)  for header in␣
↪df_weather_data.columns]

df_weather_data.head()
```

[4]:

	longitude_x	latitude_y	station_name	climate_id	datetime	year	month	day	time	temp_c	…	wind_spd_flag	visibility_km	visibility_flag	stn_press_kpa	stn_press_flag	hmdx	hmdx_flag	wind_chill	wind_chill_flag	weather
0	-79.4	43.63	TORONTO CITY CENTRE	6158359	2017-01-01 00:00	2017	1	1	00:00	1.5	…	NaN	16.1	NaN	99.81	NaN	NaN	NaN	NaN	NaN	NaN
1	-79.4	43.63	TORONTO CITY CENTRE	6158359	2017-01-01 01:00	2017	1	1	01:00	1.5	…	NaN	16.1	NaN	100.01	NaN	NaN	NaN	NaN	NaN	NaN
2	-79.4	43.63	TORONTO CITY CENTRE	6158359	2017-01-01 02:00	2017	1	1	02:00	1.0	…	NaN	16.1	NaN	100.14	NaN	NaN	NaN	NaN	NaN	NaN
3	-79.4	43.63	TORONTO CITY CENTRE	6158359	2017-01-01 03:00	2017	1	1	03:00	1.2	…	NaN	16.1	NaN	100.32	NaN	NaN	NaN	NaN	NaN	NaN
4	-79.4	43.63	TORONTO CITY CENTRE	6158359	2017-01-01 04:00	2017	1	1	04:00	1.3	…	NaN	16.1	NaN	100.48	NaN	NaN	NaN	NaN	NaN	NaN

```
3             NaN      NaN
4             NaN      NaN

[5 rows x 28 columns]
```

[5]:
```
#Save the weather station information in a dictionary
dic_weather_station_info = {'lon':df_weather_data['longitude_x'].iloc[0],'lat':␣
 ↪df_weather_data['latitude_y'].iloc[0], 'name':
 ↪df_weather_data['station_name'].iloc[0],'id':df_weather_data['climate_id'].
 ↪iloc[0]}

#Drop the station information, drop 4 columns from the dataframe
df_weather_data = df_weather_data.drop(columns =␣
 ↪['longitude_x','latitude_y','station_name','climate_id'])
print(dic_weather_station_info)
```

```
{'lon': -79.4, 'lat': 43.63, 'name': 'TORONTO CITY CENTRE', 'id': 6158359}
```

Furthermore, it has been identified that the columns with the term "flag" do not contain information required for our analysis. These columns are also removed from the dataframe, further reducing the column number from 24 to 15.

[6]:
```
#Drop list of features that have a nan description
lst_na_description =␣
 ↪['temp_flag','dew_point_temp_flag','rel_hum_flag','wind_dir_flag',␣
 ↪'wind_spd_flag', 'visibility_flag', 'stn_press_flag',␣
 ↪'hmdx_flag','wind_chill_flag']
df_weather_data = df_weather_data.drop(columns = lst_na_description)
print(df_weather_data.shape)
df_weather_data.head()
```

```
(35064, 15)
```

[6]:

	datetime	year	month	day	time	temp_c	dew_point_temp_c	\
0	2017-01-01 00:00	2017	1	1	00:00	1.5	-3.6	
1	2017-01-01 01:00	2017	1	1	01:00	1.5	-3.9	
2	2017-01-01 02:00	2017	1	1	02:00	1.0	-4.3	
3	2017-01-01 03:00	2017	1	1	03:00	1.2	-4.3	
4	2017-01-01 04:00	2017	1	1	04:00	1.3	-4.4	

	rel_hum_	wind_dir_10s_deg	wind_spd_kmh	visibility_km	stn_press_kpa	\
0	69.0	26.0	39.0	16.1	99.81	
1	67.0	27.0	35.0	16.1	100.01	
2	68.0	26.0	32.0	16.1	100.14	
3	67.0	26.0	37.0	16.1	100.32	
4	66.0	26.0	28.0	16.1	100.48	

	hmdx	wind_chill	weather

```
0   NaN         NaN     NaN
1   NaN         NaN     NaN
2   NaN         NaN     NaN
3   NaN         NaN     NaN
4   NaN         NaN     NaN
```

At this point, the weather data has been concatenated into a single dataframe, and the station information and "flag" columns have been dropped from the concatenated dataframe. Below we provide a summary of the percentable of data containing a null value in the dataframe.

[7]:
```
#Percentage of Data Containing Null Record
df_weather_data_missing = pd.DataFrame(df_weather_data.isnull().sum())
df_weather_data_missing = df_weather_data_missing.rename(columns={0:"count"})
df_weather_data_missing['percent_nulldata']=round(df_weather_data_missing['count']/
 ↪df_weather_data.shape[0] * 100,1)

print("")
print('The number of data records in the data frame is:' ,df_weather_data.
 ↪shape[0])

df_weather_data_missing
```

```
The number of data records in the data frame is: 35064
```

[7]:

	count	percent_nulldata
datetime	0	0.0
year	0	0.0
month	0	0.0
day	0	0.0
time	0	0.0
temp_c	459	1.3
dew_point_temp_c	508	1.4
rel_hum_	500	1.4
wind_dir_10s_deg	2385	6.8
wind_spd_kmh	384	1.1
visibility_km	391	1.1
stn_press_kpa	462	1.3
hmdx	29397	83.8
wind_chill	28998	82.7
weather	29547	84.3

It has been determined that more than 80% of the Hmdx and Wind Chill are null values.

Looking into the provided information on https://climate.weather.gc.ca, however, it was determined that hourly humidex values are only displayed when the air temperature is 20 deg C or greater and the humidex value is at least 1 degree greater than the air temperature. Wind Chill is also only displayed when the hourly temperature is less than or equal to 0 deg C. These columns will therefore be left in our dataframe because we have an understanding of when they would be missing

from the records.

[8]:
```
#Hmdx is only calculated if air temperture is greater than 20 deg and humidex
 ↪(H_value) is at least 1 deg or more
e = 6.11 * np.exp(5417.7530 * ( (1/273.15) - (1/
 ↪(df_weather_data['dew_point_temp_c']+273.15))))
df_weather_data['h_value']=(0.5555)*(e - 10.0)

df_weather_data[df_weather_data['hmdx'].notnull()].head()
```

[8]:

	datetime	year	month	day	time	temp_c	dew_point_temp_c
427	2017-05-18 19:00	2017	5	18	19:00	25.5	16.7
428	2017-05-18 20:00	2017	5	18	20:00	24.7	15.8
429	2017-05-18 21:00	2017	5	18	21:00	22.8	13.8
430	2017-05-18 22:00	2017	5	18	22:00	21.8	13.1
687	2017-05-29 15:00	2017	5	29	15:00	23.5	9.8

	rel_hum_	wind_dir_10s_deg	wind_spd_kmh	visibility_km	stn_press_kpa
427	58.0	28.0	28.0	16.1	99.66
428	57.0	30.0	24.0	16.1	99.76
429	56.0	29.0	17.0	16.1	99.84
430	57.0	27.0	18.0	16.1	99.96
687	41.0	25.0	28.0	16.1	99.93

	hmdx	wind_chill	weather	h_value
427	31.0	NaN	NaN	5.087083
428	29.0	NaN	NaN	4.485201
429	26.0	NaN	NaN	3.255232
430	25.0	NaN	NaN	2.857706
687	25.0	NaN	NaN	1.191371

[9]:
```
#Remove the Humidex_Calc column, not needed for future analysis
df_weather_data=df_weather_data.drop(columns = ['h_value'])
```

[10]:
```
#WindChill is Nan for temperatures greater than 0 deg
df_weather_data[df_weather_data['wind_chill'].isnull()].head()
```

[10]:

	datetime	year	month	day	time	temp_c	dew_point_temp_c
0	2017-01-01 00:00	2017	1	1	00:00	1.5	-3.6
1	2017-01-01 01:00	2017	1	1	01:00	1.5	-3.9
2	2017-01-01 02:00	2017	1	1	02:00	1.0	-4.3
3	2017-01-01 03:00	2017	1	1	03:00	1.2	-4.3
4	2017-01-01 04:00	2017	1	1	04:00	1.3	-4.4

	rel_hum_	wind_dir_10s_deg	wind_spd_kmh	visibility_km	stn_press_kpa
0	69.0	26.0	39.0	16.1	99.81
1	67.0	27.0	35.0	16.1	100.01
2	68.0	26.0	32.0	16.1	100.14

```
3      67.0              26.0          37.0          16.1          100.32
4      66.0              26.0          28.0          16.1          100.48

   hmdx  wind_chill weather
0   NaN         NaN     NaN
1   NaN         NaN     NaN
2   NaN         NaN     NaN
3   NaN         NaN     NaN
4   NaN         NaN     NaN
```

It is also noted that 84% of the data in the Weather column are also null. The weather field in this dataset contains observations of the atmospheric phenonemom including the occurence of weather and obstructions to vision. We have determined that there is value in exploring the reason for this.

[11]:
```
#Display unique values in the Weather column
df_weather_data['weather'].unique().tolist()
```

[11]:
```
[nan,
 'Fog',
 'Rain,Fog',
 'Rain',
 'Snow',
 'Moderate Rain',
 'Moderate Rain,Fog',
 'Haze',
 'Rain,Snow',
 'Freezing Rain,Fog',
 'Snow,Blowing Snow',
 'Heavy Snow',
 'Moderate Snow',
 'Haze,Blowing Snow',
 'Heavy Rain,Fog',
 'Thunderstorms,Rain,Fog',
 'Freezing Rain,Snow',
 'Freezing Rain',
 'Thunderstorms,Rain',
 'Thunderstorms,Moderate Rain,Fog',
 'Thunderstorms,Moderate Rain',
 'Thunderstorms',
 'Thunderstorms,Heavy Rain,Fog',
 'Thunderstorms,Heavy Rain',
 'Thunderstorms,Fog']
```

You can see that only abnormal weather events are listed, but there is no reference to any 'clear day'. This means that when `weather_data['Weather'] == NaN`, the conditions are actually clear. Therefore, we've established that we do not want to remove rows where `weather_data['Weather'] == NaN`. Instead, we have change all the NaN values in the weather field to 'clear_day' to reflect this understanding.

[12]:
```
#Change all Nan values into 'clear_day' in the weather column
df_weather_data['weather'] = df_weather_data['weather'].fillna('clear_day')

#Replace spaces and remove capital letters in the weather column
df_weather_data['weather'] = df_weather_data['weather'].str.replace(" ","_")

#Check that change has been effected
df_weather_data['weather'].unique().tolist()
```

[12]:
```
['clear_day',
 'Fog',
 'Rain,Fog',
 'Rain',
 'Snow',
 'Moderate_Rain',
 'Moderate_Rain,Fog',
 'Haze',
 'Rain,Snow',
 'Freezing_Rain,Fog',
 'Snow,Blowing_Snow',
 'Heavy_Snow',
 'Moderate_Snow',
 'Haze,Blowing_Snow',
 'Heavy_Rain,Fog',
 'Thunderstorms,Rain,Fog',
 'Freezing_Rain,Snow',
 'Freezing_Rain',
 'Thunderstorms,Rain',
 'Thunderstorms,Moderate_Rain,Fog',
 'Thunderstorms,Moderate_Rain',
 'Thunderstorms',
 'Thunderstorms,Heavy_Rain,Fog',
 'Thunderstorms,Heavy_Rain',
 'Thunderstorms,Fog']
```

In the weather data, there is also 1.1% or 384 records with null values for the wind speed. Originally we thought that the null values represented instances when no wind was detected (0 km/h), but we also found records where wind speed is already equal to 0 km/h. At this point, we are uncertain what this NaN value indicates for wind speed. It could be that the wind speed was too low and undetectable or it could also mean that it exceeded the maximum detectable wind speed. If the latter were true, it would be erroneous to assume that the wind speed is 0 km/h. For this reason, we decided to leave the null values in there and evaluate the missingness on a case by case basis, if necessary.

[13]:
```
#Examine the unique values assigned to wind speed
print(df_weather_data['wind_spd_kmh'].unique())

#Examine the dataframe where wind speed is null
```

```
df_weather_data[df_weather_data['wind_spd_kmh'].isnull()].head()
```

```
[39. 35. 32. 37. 28. 30. 26. 22. 18. 17. 15. 21. 11.  8.  0.  4. 13. 24.
  9.  5. 41. 52. 46. 50. 55. 45. 48. 34. 43. 61. 54. nan 58. 63. 59. 65.
 74. 68. 72. 67. 76.]
```

[13]:

	datetime	year	month	day	time	temp_c	dew_point_temp_c	\
320	2020-01-14 08:00	2020	1	14	08:00	1.6	-0.2	
323	2020-01-14 11:00	2020	1	14	11:00	2.3	0.2	
326	2020-01-14 14:00	2020	1	14	14:00	1.3	0.4	
334	2020-01-14 22:00	2020	1	14	22:00	NaN	NaN	
343	2020-01-15 07:00	2020	1	15	07:00	3.1	-1.4	

	rel_hum_	wind_dir_10s_deg	wind_spd_kmh	visibility_km	stn_press_kpa	\
320	88.0	NaN	NaN	16.1	101.36	
323	86.0	NaN	NaN	16.1	101.14	
326	94.0	NaN	NaN	9.7	100.67	
334	91.0	NaN	NaN	9.7	NaN	
343	72.0	NaN	NaN	16.1	101.24	

	hmdx	wind_chill	weather
320	NaN	NaN	clear_day
323	NaN	NaN	clear_day
326	NaN	NaN	clear_day
334	NaN	NaN	Fog
343	NaN	NaN	clear_day

In the weather data, there is also 6.8% or 2385 records with null values for the wind direction. In https://climate.weather.gc.ca, it is provided that a wind direction of 0 is assigned to denote a calm wind. However, looking into the data there is no instance of zero in the data. Also looking at the wind speed associated with the null wind direction values, the maximum speed observed was 21 km/hour, while the wind speed in the data ranges from 0 to 76 km/hr. Thus, it has been assumed that a null value as been assigned to what should have been 0 for wind direction, denoting a calm wind. For records with null value for wind speed, the wind direction was also left as a null value.

[14]:
```
#Examine the dataframe where wind direction is equal to 0
print(df_weather_data[df_weather_data['wind_dir_10s_deg']==0].shape)
print('There is no records with wind direction = 0')

#Max wind speed value in the weather dataframe
print('Max wind speed in entire dataframe (km/h):',␣
 ↪max(df_weather_data['wind_spd_kmh'].unique()))

#Examine the range of wind speed for null wind direction
print('Max wind speed for records with null wind direction:
 ↪',max(df_weather_data[df_weather_data['wind_dir_10s_deg'].
 ↪isnull()]['wind_spd_kmh'].unique()))
```

```
#Assume that null values in Wind Direction denotes a calm wind represented by␣
 ↪0, except for when wind speed is null
lst_to_replace = (df_weather_data['wind_spd_kmh'].notnull()) &␣
 ↪(df_weather_data['wind_dir_10s_deg'].isnull())
for i in range(len(lst_to_replace)):
    if lst_to_replace.iloc[i]:
        df_weather_data.iloc[i,8] = 0
```

```
(0, 15)
There is no records with wind direction = 0
Max wind speed in entire dataframe (km/h): 76.0
Max wind speed for records with null wind direction: 21.0
```

[15]:
```
#Check unique values assigned to records where wind speed is not null
print(df_weather_data[df_weather_data['wind_spd_kmh'].
 ↪notnull()]['wind_dir_10s_deg'].unique())
```

```
[26. 27. 23. 24. 25. 22.  0.  3.  5.  6.  8.  7. 10.  9.  4. 33. 28. 29.
 31. 30. 32. 20. 21. 19. 18. 15. 14. 12. 13. 16. 17. 34. 35. 11.  1. 36.
  2.]
```

[16]:
```
#Check unique values assigned to records where wind speed is null
print(df_weather_data[df_weather_data['wind_spd_kmh'].
 ↪isnull()]['wind_dir_10s_deg'].unique())
```

```
[nan]
```

Looking at the records with null values for visibility, we can see that all these days tend to all fall on clear days. As such, we have interpreted the null values to represent no visibility issue. To reflect our understanding of this in the dataframe, we have replaced the null values with a positive infinity value. This will be the largest visibility distance in the dataset.

[17]:
```
#Examine the unique values assigned to Visibility
print(df_weather_data['visibility_km'].unique())

#Examine records with Weather associated with null values assigned to␣
 ↪Visibility column
print(df_weather_data[df_weather_data['visibility_km'].isnull()]['weather'].
 ↪unique())

#Examine records with Weather associated with 16.1 km Visibility
print(df_weather_data[df_weather_data['visibility_km']==16.1]['weather'].
 ↪unique())
```

```
[16.1 12.9  9.7 11.3  6.4  3.2  4.8 14.5  8.1  2.4  2.8  1.6  2.   4.
  1.   0.4  3.6  0.8  0.6  1.2  0.   0.2  nan]
['clear_day']
['clear_day' 'Rain' 'Snow' 'Rain,Snow' 'Moderate_Rain' 'Freezing_Rain'
 'Thunderstorms,Rain' 'Thunderstorms,Moderate_Rain' 'Thunderstorms']
```

[18]:
```
#Replace all null values in visibility with infinity
df_weather_data['visibility_km'] = df_weather_data['visibility_km'].fillna(np.
 ↪inf)

#Check the dataframe
df_weather_data.sort_values('visibility_km', ascending =False).head()
```

[18]:

	datetime	year	month	day	time	temp_c	dew_point_temp_c	\
743	2020-12-31 23:00	2020	12	31	23:00	NaN	NaN	
475	2020-12-20 19:00	2020	12	20	19:00	NaN	NaN	
477	2020-12-20 21:00	2020	12	20	21:00	NaN	NaN	
478	2020-12-20 22:00	2020	12	20	22:00	NaN	NaN	
479	2020-12-20 23:00	2020	12	20	23:00	NaN	NaN	

	rel_hum_	wind_dir_10s_deg	wind_spd_kmh	visibility_km	stn_press_kpa	\
743	NaN	NaN	NaN	inf	NaN	
475	NaN	NaN	NaN	inf	NaN	
477	NaN	NaN	NaN	inf	NaN	
478	NaN	NaN	NaN	inf	NaN	
479	NaN	NaN	NaN	inf	NaN	

	hmdx	wind_chill	weather
743	NaN	NaN	clear_day
475	NaN	NaN	clear_day
477	NaN	NaN	clear_day
478	NaN	NaN	clear_day
479	NaN	NaN	clear_day

Through the process of examining the null values found for each field column and applying the appropriate data wrangling techniques, we were able to reduce the total number of missing records in the weather dataframe.

At this point, we can make sure we have removed all weather records with null values for all the columns.

[19]:
```
original_count=df_weather_data.shape[0]
df_weather_data.dropna(subset =␣
 ↪['temp_c','rel_hum_','wind_dir_10s_deg','wind_spd_kmh','visibility_km','hmdx','wind_chill',
                        how='all', inplace=True)
print('Number of records removed from Trip Dataframe: ', original_count -␣
 ↪df_weather_data.shape[0])
```

```
Number of records removed from Trip Dataframe:  0
```

We can see that the first 6 columns of `weather_data_missing` have no missing data.

For the humidex and wind chill column, we now understand why they are null values. These parameters were not calculated when certain conditions in the weather data was not met.

Other columns such as temperature, wind speed, wind direction, dew point temperature and relative

humidity, there are a few null records but total number of null values in each field is about 1%. We could not confidently determine why there were null values in this column fields, but because there is so few of them, we will address the missingness on a case-by-case basis depending on which columns we're analyzing.

[20]:
```
#Percentage of Data Containing Null Record
weather_data_missing = pd.DataFrame(df_weather_data.isnull().sum())
weather_data_missing = weather_data_missing.rename(columns={0:"count"})
weather_data_missing['percent_nulldata']=round(weather_data_missing['count']/
 ↪df_weather_data.shape[0] * 100,1)

print("")
print('The number of data records in the data frame is:' ,df_weather_data.
 ↪shape[0])

weather_data_missing
```

```
The number of data records in the data frame is: 35064
```

[20]:

	count	percent_nulldata
datetime	0	0.0
year	0	0.0
month	0	0.0
day	0	0.0
time	0	0.0
temp_c	459	1.3
dew_point_temp_c	508	1.4
rel_hum_	500	1.4
wind_dir_10s_deg	384	1.1
wind_spd_kmh	384	1.1
visibility_km	0	0.0
stn_press_kpa	462	1.3
hmdx	29397	83.8
wind_chill	28998	82.7
weather	0	0.0

4 3. Weather Data - Date/Time

Set Date/Time as index and localize to EST time zone

The Date/Time was originally imported as a string object. When they were converted into a datetime object, the data was time zone naive (i.e. no information on the time zone was provided for the data to unambiguously locate itself relative to other date/time objects). In other words, the timestamp was not localized to any specific time zone.

[21]:
```
df_weather_data = df_weather_data.set_index('datetime')
df_weather_data.index = pd.DatetimeIndex(df_weather_data.index)
```

```
df_weather_data=df_weather_data.tz_localize(tz='EST')

# View DataFrame
df_weather_data.head()
```

[21]:

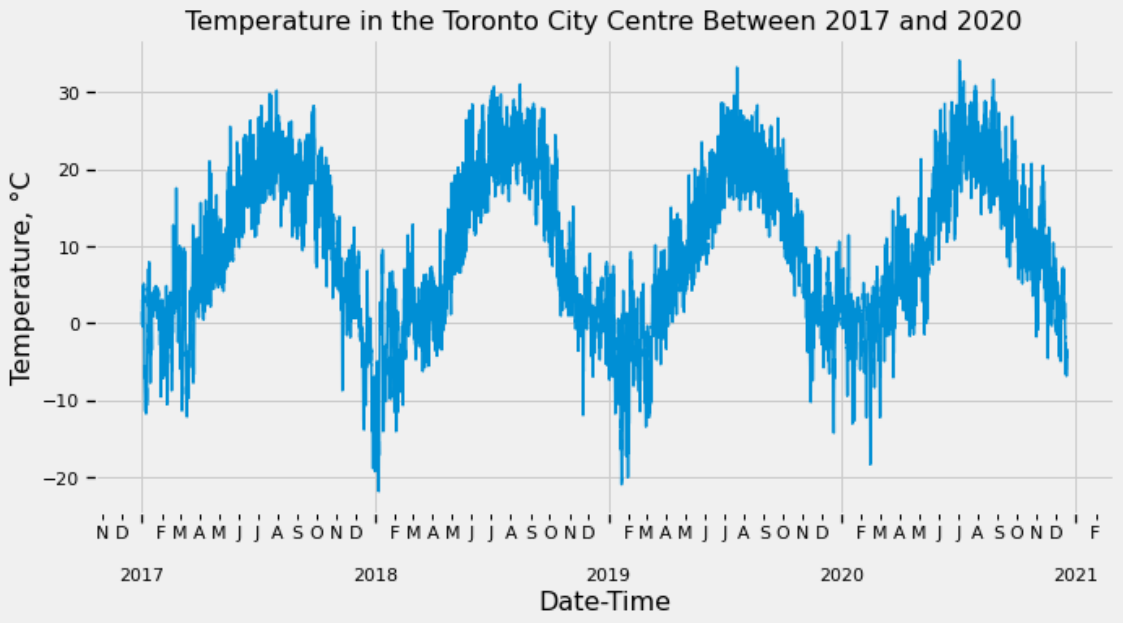
datetime	year	month	day	time	temp_c	dew_point_temp_c	rel_hum_	wind_dir_10s_deg	wind_spd_kmh	visibility_km	stn_press_kpa	hmdx	wind_chill	weather
2017-01-01 00:00:00-05:00	2017	1	1	00:00	1.5	-3.6	69.0	26.0	39.0	16.1	99.81	NaN	NaN	clear_day
2017-01-01 01:00:00-05:00	2017	1	1	01:00	1.5	-3.9	67.0	27.0	35.0	16.1	100.01	NaN	NaN	clear_day
2017-01-01 02:00:00-05:00	2017	1	1	02:00	1.0	-4.3	68.0	26.0	32.0	16.1	100.14	NaN	NaN	clear_day
2017-01-01 03:00:00-05:00	2017	1	1	03:00	1.2	-4.3	67.0	26.0	37.0	16.1	100.32	NaN	NaN	clear_day
2017-01-01 04:00:00-05:00	2017	1	1	04:00	1.3	-4.4	66.0	26.0	28.0	16.1	100.48	NaN	NaN	clear_day

4. Visualization of the Weather Data

This section verifies the robustness of the weather data through various visualization techniques.

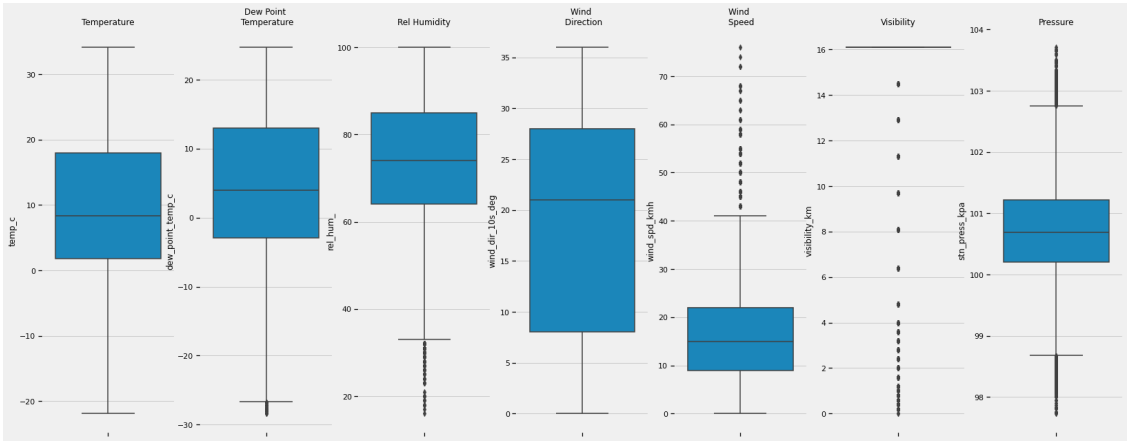
[22]:
```
plt.figure(figsize=(10,5))
temp=sns.lineplot(x=df_weather_data.index,y=df_weather_data['temp_c'])
temp.axes.set_title("Temperature in the Toronto City Centre Between 2017 and␣
 ↪2020",
                    fontsize=16)
temp.set_ylabel("Temperature, °C",
                  fontsize=16)
```

```
temp.set_xlabel("Date-Time",
                  fontsize=16)

#format axis
# Minor ticks every month.
fmt_month = mdates.MonthLocator(interval=1)

#define function to return first letter of every month
month_fmt = DateFormatter('%b')
def m_fmt(x, pos=None):
    return month_fmt(x)[0]

temp.xaxis.set_minor_locator(MonthLocator())
temp.xaxis.set_minor_formatter(FuncFormatter(m_fmt))

# Major ticks every year
years = mdates.YearLocator()
temp.xaxis.set_major_locator(years)
yearsFmt = mdates.DateFormatter('\n\n%Y')  # add some space for the year label
temp.xaxis.set_major_formatter(yearsFmt)

plt.show()
```



[23]:
```
fig, (ax1, ax2, ax3, ax4, ax5, ax6, ax7) = plt.subplots(1,7,figsize = (25,10))

sns.boxplot(y = 'temp_c', data = df_weather_data, ax = ax1)
```

```
ax1.set_title('Temperature')

sns.boxplot(y = 'dew_point_temp_c', data = df_weather_data, ax = ax2)
ax2.set_title('Dew Point \n Temperature')

sns.boxplot(y = 'rel_hum_', data = df_weather_data, ax = ax3)
ax3.set_title('Rel Humidity')

sns.boxplot(y = 'wind_dir_10s_deg', data = df_weather_data, ax = ax4)
ax4.set_title('Wind \n Direction')

sns.boxplot(y = 'wind_spd_kmh', data = df_weather_data, ax = ax5)
ax5.set_title('Wind \n Speed')

sns.boxplot(y = 'visibility_km', data = df_weather_data, ax = ax6)
ax6.set_title('Visibility')

sns.boxplot(y = 'stn_press_kpa', data = df_weather_data, ax = ax7)
ax7.set_title('Pressure')

fig.show()
```



The reason why the boxplot for the visibility is not showing up is because a significant portion of the data indicate a visibility of 16.1 km. For this reason the 1st, 2nd and 3rd quartile is essentially the same value.

[24]:
```
#Estimate quartile values for Visibility column
Q1=np.percentile(df_weather_data['visibility_km'], 25)  # Q1
Q2=np.percentile(df_weather_data['visibility_km'], 50)  # median
Q3=np.percentile(df_weather_data['visibility_km'], 75)  # Q3

print("Q1=",Q1)
```

```
print("Q2=",Q2)
print("Q3=",Q3)
```

```
Q1= 16.1
Q2= 16.1
Q3= 16.1
```

Because of the high number of clear weather days in the data set, it is very plausible that there is also high number of records indicate 16.1 km (or higher) of visibility distance.

[25]:
```
print('{:.2f}% of the weather records indicate a visibility 16.1 km or higher.␣
↪Similarly {:.2f}% of the days are have clear skies'
      .format((df_weather_data['visibility_km'] >= 16.1).sum()/df_weather_data.
↪shape[0]*100
              , (df_weather_data['weather'] == 'clear_day').sum()/
↪df_weather_data.shape[0]*100))
```

```
87.21% of the weather records indicate a visibility 16.1 km or higher. Similarly
84.27% of the days are have clear skies
```

The final process in the data wrangling process is to export the manipulated data as a csv for easy access.

[26]:
```
#Export the weather dataframe in csv format
df_weather_data.to_csv('cleaned_weather_data_set.csv')
```

5 5. Bike Share Trip Data - Import Data

In this section, a dataframe is created containing the historical ridership data in Toronto from January 2017 to October 2020. The bike trip data was provided in csv format with one file for every month.

The main challenge with merging these csv files is the change in column field between 2018 and 2019. For example, the bike triip data from 2017 and 2019 contain 9 fields, while the bike trip data from 2019 and 2020 contain 11 fields.

It as also been identified that the date/time indicated in the 2017/2018 data is in UTC while the 2019/2020 data is in EST. The date/time will be localized to the appropriate time zone during the importation process.

[27]:
```
# Create a list of file names for bike share trips data contained in repository
trips_filenames = [filename for filename in os.listdir() if 'bike_share' in␣
↪filename]

# Create a dictionary where key:value pairs correspond to the file name and␣
↪DataFrame respectively
trips_data = {filename: pd.read_csv(filename) for filename in trips_filenames}

# Now let's print out the column names for the first month of each year
print('2017\n{}\n'.format(trips_data['bike_share_2017-1.csv'].columns.tolist()))
```

```
print('2018\n{}\n'.format(trips_data['bike_share_2018-1.csv'].columns.tolist()))
print('2019\n{}\n'.format(trips_data['bike_share_2019-1.csv'].columns.tolist()))
print('2020\n{}\n'.format(trips_data['bike_share_2020-1.csv'].columns.tolist()))
```

```
2017
['trip_id', 'trip_start_time', 'trip_stop_time', 'trip_duration_seconds',
'from_station_id', 'from_station_name', 'to_station_id', 'to_station_name',
'user_type']

2018
['trip_id', 'trip_duration_seconds', 'from_station_id', 'trip_start_time',
'from_station_name', 'trip_stop_time', 'to_station_id', 'to_station_name',
'user_type']

2019
['Trip Id', 'Subscription Id', 'Trip  Duration', 'Start Station Id', 'Start
Time', 'Start Station Name', 'End Station Id', 'End Time', 'End Station Name',
'Bike Id', 'User Type']

2020
['Trip Id', 'Subscription Id', 'Trip  Duration', 'Start Station Id', 'Start
Time', 'Start Station Name', 'End Station Id', 'End Time', 'End Station Name',
'Bike Id', 'User Type']
```

To overcome this issue, we initially built two dataframes, one for the 2017 and 2018 dataset and another one for the 2018 and 2019 dataset. Once accounted for the missing fields in the 2017 and 2018 dataset, the differences in the field names between the 2017/2018 and 2019/2020 dataset, differences in the reflected time zones, the differences in the datetime format, the dataframes were concatenated to create one dataframe containing all the bike trip data from 2017 to 2020.

[28]:
```
# Build two data frames to be merged later
df_trips_data = pd.DataFrame()
df_trips_data2 = pd.DataFrame()

for file in trips_filenames:
    if '2017' in file:
        df_trips_data= pd.concat([df_trips_data,pd.read_csv(file)],axis =0)

#Add the 2018 files to the dataframe
for file in trips_filenames:
    if '2018' in file:
        df_trips_data= pd.concat([df_trips_data,pd.read_csv(file)], axis =0)

#Correct column names and add additional columns for Bike ID and subscription ID
df_trips_data['Bike Id'] = np.nan
df_trips_data['Subscription Id'] = np.nan
```

```
#Convert columns trip start and end times to datetimes. Then, localize to UTC␣
 ↪then convert to EST
df_trips_data['trip_start_time']=pd.
 ↪to_datetime(df_trips_data['trip_start_time'], format='%Y-%m-%d %H:%M (%Z)')
df_trips_data['trip_start_time']=df_trips_data['trip_start_time'].dt.
 ↪tz_convert(tz='EST')

df_trips_data['trip_stop_time']=pd.to_datetime(df_trips_data['trip_stop_time'],␣
 ↪format='%Y-%m-%d %H:%M (%Z)')
df_trips_data['trip_stop_time']=df_trips_data['trip_stop_time'].dt.
 ↪tz_convert(tz='EST')

#correct the order of the columns to match the 2019/2020 data
cols = df_trips_data.columns.tolist()
new_order = [0,10,3,4,1,5,6,2,7,9,8]
cols = [cols[i] for i in new_order]

#reorder columns of the Data frame
df_trips_data = df_trips_data[cols]

#rename column headers of the data frame
df_trips_data.columns = trips_data['bike_share_2019-1.csv'].columns.tolist()

#complete the reading of all the files to the second dataframe
for file in trips_filenames:
    if ('2019' in file) |('2020' in file) :
        df_trips_data2  = pd.concat([df_trips_data2 ,pd.read_csv(file)], axis=0)

# Let's remove double spaces from the column names
df_trips_data.columns = ['_'.join(col.split()).lower() for col in df_trips_data.
 ↪columns]
df_trips_data2.columns = ['_'.join(col.split()).lower() for col in␣
 ↪df_trips_data2.columns]

#Convert columns trip start and end times to datetimes. Then, localize to EST
df_trips_data2['start_time']=pd.to_datetime(df_trips_data2['start_time'],␣
 ↪format='%d/%m/%Y %H:%M:%S (%Z)')
df_trips_data2['end_time']=pd.to_datetime(df_trips_data2['end_time'],␣
 ↪format='%d/%m/%Y %H:%M:%S (%Z)')


#Combine the two dataframes
df_trips_data=pd.concat([df_trips_data ,df_trips_data2], axis=0)

#view data frame
df_trips_data.head()
```

```
del df_trips_data2
```

Timezone Conversion We conduct a quick check to see if the data contained in the data is in the expected date/time range. All the trip records should occur after January 1, 2017 and before October 31, 2020. Based on this check, we found 58 trips that occured before January 1, 2017 and 0 trips that occured after October 31, 2020.

Based on this check, we found 58 trips that occured before January 1, 2017 and 0 trips that occured after October 31, 2020. We highly recommend that City of Toronto checks the dataset provided to us, particularly for 'bike_share_2017-12.csv'. This csv file has been identified as the file containing trips prior to January 1, 2017. The dataset before January 1, 2017 will be dropped from further analysis because no weather data is available before and we are focusing our analysis between the time frame 2017 and 2020.

[29]:
```
print("{:.0f} trip occur before January 1, 2017".
 ↪format(df_trips_data[(df_trips_data['start_time'] < '2017-01-01')].shape[0]))
df_trips_data[(df_trips_data['start_time'] < '2017-01-01')].head()
```

```
58 trip occur before January 1, 2017
```

[29]:

	trip_id	subscription_id	trip_duration	start_station_id	start_time	start_station_name	end_station_id	end_time	end_station_name	bike_id	user_type
0	712382	NaN	223	7051.0	2016-12-31 20:00:00-05:00	Wellesley St E / Yonge St Green P	7089.0	2016-12-31 20:03:00-05:00	Church St / Wood St	NaN	Member
1	712383	NaN	279	7143.0	2016-12-31 20:00:00-05:00	Kendal Ave / Bernard Ave	7154.0	2016-12-31 20:05:00-05:00	Bathurst Subway Station	NaN	Member
2	712384	NaN	1394	7113.0	2016-12-31 20:05:00-05:00	Parliament St / Aberdeen Ave	7199.0	2016-12-31 20:29:00-05:00	College St W / Markham St	NaN	Member
3	712385	NaN	826	7077.0	2016-12-31 20:07:00-05:00	College Park South	7010.0	2016-12-31 20:21:00-05:00	King St W / Spadina Ave	NaN	Member
4	712386	NaN	279	7079.0	2016-12-31 20:08:00-05:00	McGill St / Church St	7047.0	2016-12-31 20:12:00-05:00	University Ave / Gerrard St W	NaN	Member

[30]:
```
df_trips_data=df_trips_data[(df_trips_data['start_time'] >= '2017-01-01')]
print("{:.0f} trip occur before January 1, 2017".
 ↪format(df_trips_data[(df_trips_data['start_time'] < '2017-01-01')].shape[0]))
```

```
0 trip occur before January 1, 2017
```

We did not find any trips with start date past October 31, 2020.

[31]:
```
print("{:.0f} trip occur after October 31, 2020".
 ↪format(df_trips_data[(df_trips_data['start_time'] > '2020-11-01')].shape[0]))
df_trips_data[(df_trips_data['start_time'] > '2020-11-01')].head()
```

```
0 trip occur after October 31, 2020
```

[31]:
```
Empty DataFrame
Columns: [trip_id, subscription_id, trip_duration, start_station_id, start_time,
start_station_name, end_station_id, end_time, end_station_name, bike_id,
user_type]
Index: []
```

6 6. Bike Share Trip Data - Data Wrangling

The trip ID is an unique identifier for each trip. It is important to verify that the bike trips contained in the concatenated dataframe do not include duplicate IDs.

[32]:
```
print("There are {:.0f} bike trips taken from January 2017 to November 2020".
 ↪format(df_trips_data.shape[0]))

print("{:.0f} % of the trip Id's are unique".format(df_trips_data['trip_id'].
 ↪nunique()/df_trips_data.shape[0]*100))
```

```
There are 8467487 bike trips taken from January 2017 to November 2020
100 % of the trip Id's are unique
```

Special characters tend to cause issues in the code so we will create a new column containing the station names with the special characters removed.

[33]:
```
# create new column field with station name - convert to lower case and remove␣
 ↪all punctuation from string
df_trips_data['start_station_name'] = df_trips_data['start_station_name'].
 ↪astype(str)
df_trips_data['end_station_name'] = df_trips_data['end_station_name'].
 ↪astype(str)

df_trips_data['start_station_name_npl'] = [' '.join(s.translate(str.
 ↪maketrans('','',string.punctuation)).lower().split()).strip() for s in␣
 ↪df_trips_data['start_station_name']]
```

```
df_trips_data['end_station_name_npl'] = [' '.join(s.translate(str.
↪maketrans('','',string.punctuation)).lower().split()).strip() for s in␣
↪df_trips_data['end_station_name']]

#convert all string 'nan' back to null values
df_trips_data['start_station_name'].replace('nan',np.nan, inplace=True)
df_trips_data['end_station_name'] .replace('nan',np.nan, inplace=True)
df_trips_data['start_station_name_npl'].replace('nan',np.nan, inplace=True)
df_trips_data['end_station_name_npl'].replace('nan',np.nan, inplace=True)

df_trips_data.head()
```

[33]:

	trip_id	subscription_id	trip_duration	start_station_id	start_time	start_station_name
0	712441	NaN	274	7006.0	2017-01-01 00:03:00-05:00	Bay St / College St (East Side)
1	712442	NaN	538	7046.0	2017-01-01 00:03:00-05:00	Niagara St / Richmond St W
2	712443	NaN	992	7048.0	2017-01-01 00:05:00-05:00	Front St / Yonge St (Hockey Hall of Fame)
3	712444	NaN	1005	7177.0	2017-01-01 00:09:00-05:00	East Liberty St / Pirandello St
4	712445	NaN	645	7203.0	2017-01-01 00:14:00-05:00	Bathurst St / Queens Quay W

	end_station_id	end_time	end_station_name	bike_id	user_type	start_station_name_npl	end_station_name_npl
0	7021.0	2017-01-01 00:08:00-05:00	Bay St / Albert St	NaN	Member	bay st college st east side	bay st albert st
1	7147.0	2017-01-01 00:12:00-05:00	King St W / Fraser Ave	NaN	Member	niagara st richmond st w	king st w fraser ave
2	7089.0	2017-01-01 00:22:00-05:00	Church St / Wood St	NaN	Member	front st yonge st hockey hall of fame	church st wood st
3	7202.0	2017-01-01 00:26:00-05:00	Queen St W / York St (City Hall)	NaN	Member	east liberty st pirandello st	queen st w york st city hall
4	7010.0	2017-01-01 00:25:00-05:00	King St W / Spadina Ave	NaN	Member	bathurst st queens quay w	king st w spadina ave

[34]:
```
#Percentage of Data Containing Null Record
trips_data_missing = pd.DataFrame(df_trips_data.isnull().sum())
trips_data_missing = trips_data_missing.rename(columns={0:"count"})
trips_data_missing['percent_nulldata']=round(trips_data_missing['count']/
 ↪df_trips_data.shape[0] * 100,1)

trips_data_missing
```

[34]:

	count	percent_nulldata
trip_id	0	0.0
subscription_id	3415266	40.3
trip_duration	0	0.0
start_station_id	1026893	12.1
start_time	0	0.0
start_station_name	164	0.0
end_station_id	1028159	12.1
end_time	1	0.0
end_station_name	1409	0.0
bike_id	3415266	40.3
user_type	0	0.0
start_station_name_npl	164	0.0
end_station_name_npl	1409	0.0

We removed the trip record from the dataset if both the Station Id and Station Name is missing. We are uncertain at to why the start and end location information is missing from some of the trip records. Without this information, we do not have sufficient enough information in these records to verify the start and end locations of the trip. We would need the City of Toronto data team to examine this and advise us whether these records should be included in our analysis.

For now, we determined that these records are unreliable and should not be carried forward in our analysis.

[35]:
```
original_count=df_trips_data.shape[0]
df_trips_data = df_trips_data[~(df_trips_data['start_station_name'].isnull() &␣
 ↪df_trips_data['start_station_id'].isnull())]
df_trips_data = df_trips_data[~(df_trips_data['end_station_name'].isnull() &␣
 ↪df_trips_data['end_station_id'].isnull())]

print("There are {:.0f} bike trips remaining in dataset".format(df_trips_data.
 ↪shape[0]))
print('Number of records removed from Trip Dataframe: ', original_count -␣
 ↪df_trips_data.shape[0])
```

```
There are 8466220 bike trips remaining in dataset
Number of records removed from Trip Dataframe:  1267
```

Now let's evaluate again what the count/percentage of null data is in the bike trip dataset.

[36]:
```
#Percentage of Data Containing Null Record
trips_data_missing = pd.DataFrame(df_trips_data.isnull().sum())
trips_data_missing = trips_data_missing.rename(columns={0:"count"})
trips_data_missing['percent_nulldata']=round(trips_data_missing['count']/
 ↪df_trips_data.shape[0] * 100,1)

trips_data_missing
```

[36]:

	count	percent_nulldata
trip_id	0	0.0
subscription_id	3415265	40.3
trip_duration	0	0.0
start_station_id	1026892	12.1
start_time	0	0.0
start_station_name	164	0.0
end_station_id	1026892	12.1
end_time	0	0.0
end_station_name	142	0.0
bike_id	3415265	40.3
user_type	0	0.0
start_station_name_npl	164	0.0
end_station_name_npl	142	0.0

6.1 Filling in the Gaps

6.1.1 Search for Missing station Id's based on station names with in the dataframe

Looking at the table above, now we must attempt to fill in the missing data in the station_id and station_name column. If we know one, we can find what the corresponding name or id is.

Initially, we searched for the missing station id based on the station name within the df_trips_data dataframe. The function (simple_in_df_name_search) searches within the dataframe for trips with missing station id. For every trip with a missing id, it will search through the entire database for records with the same station name. If a match if found, then the station id from that trip is used to fill in the missing station id.

[37]:
```
def simple_in_df_name_search(df, feature):
    #search stations data frame for missing ID's
    #create list of unique start station names
    feature_name = feature+'_station_name'
    feature_Id = feature+'_station_id'
    feature_name_npl = feature+'_station_name_npl'

    df_missing_station_id = df[df[feature_Id].isnull()]
    df = df[df[feature_Id].notnull()]

    lst_station_names = df_missing_station_id[feature_name_npl].unique().
 ↪tolist()
```

```
    start_station_names = df['start_station_name_npl'].unique().tolist()
    end_station_names = df['end_station_name_npl'].unique().tolist()
    #itterate through list and replace station Id's if found
    for name in lst_station_names:
        #check name exists in list of station names
        if name in start_station_names:
            mask = df_missing_station_id[feature_name_npl] == name
            station_id = df[df['start_station_name_npl'] ==␣
↪name]['start_station_id'].unique()[0]
            df_missing_station_id.loc[mask,feature_Id] = station_id
        elif name in end_station_names:
            mask = df_missing_station_id[feature_name_npl] == name
            station_id = df[df['end_station_name_npl'] ==␣
↪name]['end_station_id'].unique()[0]
            df_missing_station_id.loc[mask,feature_Id] = station_id
    df = df.append(df_missing_station_id)
    return df

df_trips_data = simple_in_df_name_search(df_trips_data,'start')
df_trips_data = simple_in_df_name_search(df_trips_data,'end')

print(df_trips_data.shape)
df_trips_data.head()
```

```
(8466220, 13)
```

[37]:

	trip_id	subscription_id	trip_duration	start_station_id	start_time	start_station_name	end_station_id	end_time	end_station_name
0	712441	NaN	274	7006.0	2017-01-01 00:03:00-05:00	Bay St / College St (East Side)	7021.0	2017-01-01 00:08:00-05:00	Bay St / Albert St
1	712442	NaN	538	7046.0	2017-01-01 00:03:00-05:00	Niagara St / Richmond St W	7147.0	2017-01-01 00:12:00-05:00	King St W / Fraser Ave
2	712443	NaN	992	7048.0	2017-01-01 00:05:00-05:00	Front St / Yonge St (Hockey Hall of Fame)	7089.0	2017-01-01 00:22:00-05:00	Church St / Wood St
3	712444	NaN	1005	7177.0	2017-01-01 00:09:00-05:00	East Liberty St / Pirandello St	7202.0	2017-01-01 00:26:00-05:00	Queen St W / York St (City Hall)
4	712445	NaN	645	7203.0	2017-01-01 00:14:00-05:00	Bathurst St / Queens Quay W	7010.0	2017-01-01 00:25:00-05:00	King St W / Spadina Ave

```
   bike_id user_type                start_station_name_npl  \
0      NaN    Member            bay st college st east side
1      NaN    Member               niagara st richmond st w
2      NaN    Member  front st yonge st hockey hall of fame
3      NaN    Member          east liberty st pirandello st
4      NaN    Member              bathurst st queens quay w

           end_station_name_npl
0             bay st albert st
1          king st w fraser ave
2             church st wood st
3  queen st w york st city hall
4         king st w spadina ave
```

[38]:
```
#Percentage of Data Containing Null Record
trips_data_missing = pd.DataFrame(df_trips_data.isnull().sum())
trips_data_missing = trips_data_missing.rename(columns={0:"count"})
trips_data_missing['percent_nulldata']=round(trips_data_missing['count']/
 ↪df_trips_data.shape[0] * 100,1)

trips_data_missing
```

[38]:
```
                          count  percent_nulldata
trip_id                       0               0.0
subscription_id         3415265              40.3
trip_duration                 0               0.0
start_station_id          27386               0.3
start_time                    0               0.0
start_station_name          164               0.0
end_station_id            29346               0.3
end_time                      0               0.0
end_station_name            142               0.0
bike_id                 3415265              40.3
user_type                     0               0.0
start_station_name_npl      164               0.0
end_station_name_npl        142               0.0
```

[39]:
```
#list of start station names for which station id is null or missing
df_trips_data[df_trips_data['start_station_id'].isnull()]['start_station_name'].
 ↪unique().tolist()
```

[39]:
```
['Beverly St / College St',
 'Dovercourt Rd / Harrison St - SMART',
 'Roxton Rd / College St',
 'Michael Sweet Ave / St. Patrick St',
 'Lansdowne Subway Green P',
 'Lake Shore Blvd W / Ontario Dr(Ontario Place)',
```

```
 'Fringe Next Stage - 7219',
 'Margueretta St / College St',
 'Summerhill Ave / MacLennan Ave - SMART',
 'Base Station']
```

6.2 Filling the gap with the Bike Station Data

For the remainder of trips with missing station id or station names, we will attempt to see if we can continue to fill in the gap using the bike station data.

6.2.1 Determine Missing Station Name from Available Station Id in Bike Station Data

In the custom function (id_name_find_replace), initially we look for the trips that are missing station names but the station id is known. Then bike station data set, it looks for the same station name and assigns the station id associated with that name to the missing station name. In the stations dataset, the information about each station, including the station name, id, location (longitude and latitude), and capacity, is provided in the csv file called "bikeshare_stations.csv" located in the repository.

[40]:
```
#Import the bikeshare station data
stations = pd.read_csv('bikeshare_stations.csv')

print("There are {:.0f} bike stations across Toronto".format(stations.shape[0]))

# Remove spaces in the dataframe column names
stations.columns = [s.replace(' ','_').lower() for s in stations.columns ]

# Similar to bike trip dataframe, we will create new column with station name␣
 ↪but with all special characters removed
stations['station_name_npl'] = [' '.join(s.translate(str.maketrans('','',string.
 ↪punctuation)).lower().split()).strip() for s in stations['station_name']]

# Remove trailing white spaces
stations['station_name']=stations['station_name'].str.strip()

stations.head()
```

```
There are 610 bike stations across Toronto
```

[40]:

	station_id	station_name	lat	lon	\
0	7000	Fort York Blvd / Capreol Ct	43.639832	-79.395954	
1	7001	Lower Jarvis St / The Esplanade	43.647830	-79.370698	
2	7002	St. George St / Bloor St W	43.667333	-79.399429	
3	7003	Madison Ave / Bloor St W	43.667158	-79.402761	
4	7004	University Ave / Elm St	43.656518	-79.389099	

	capacity	station_name_npl

```
0        35      fort york blvd capreol ct
1        15  lower jarvis st the esplanade
2        19         st george st bloor st w
3        15          madison ave bloor st w
4        11           university ave elm st
```

[41]:
```
#Station Name Replacement based on Station ID
def id_name_find_replace(df,feature):

    #build feature name
    feature_name = feature+'_station_name'
    feature_name_npl = feature+'_station_name_npl'
    feature_Id = feature+'_station_id'

    df_missing_name = df[df[feature_name].isnull() & df[feature_Id].notnull()]
    df = df[~(df[feature_name].isnull() & df[feature_Id].notnull())]

    lst_id = df_missing_name[feature_Id].unique().tolist()
    for stat_id in lst_id:
        mask = df_missing_name[feature_Id] == stat_id
        df_missing_name.loc[mask,feature_name] =␣
↪stations[stations['station_id'] == stat_id]['station_name'].tolist()[0]
        df_missing_name.loc[mask,feature_name_npl] =␣
↪stations[stations['station_id'] == stat_id]['station_name_npl'].tolist()[0]

    return df.append(df_missing_name)


df_trips_data = id_name_find_replace(df_trips_data,'start')
df_trips_data = id_name_find_replace(df_trips_data,'end')

df_trips_data.head()
```

[41]:
```
   trip_id  subscription_id  trip_duration  start_station_id  \
0   712441              NaN            274            7006.0
1   712442              NaN            538            7046.0
2   712443              NaN            992            7048.0
3   712444              NaN           1005            7177.0
4   712445              NaN            645            7203.0

                 start_time                        start_station_name  \
0 2017-01-01 00:03:00-05:00           Bay St / College St (East Side)
1 2017-01-01 00:03:00-05:00                Niagara St / Richmond St W
2 2017-01-01 00:05:00-05:00  Front St / Yonge St (Hockey Hall of Fame)
3 2017-01-01 00:09:00-05:00           East Liberty St / Pirandello St
4 2017-01-01 00:14:00-05:00               Bathurst St / Queens Quay W
```

```
   end_station_id                 end_time                 end_station_name  \
0          7021.0 2017-01-01 00:08:00-05:00               Bay St / Albert St
1          7147.0 2017-01-01 00:12:00-05:00           King St W / Fraser Ave
2          7089.0 2017-01-01 00:22:00-05:00             Church St  / Wood St
3          7202.0 2017-01-01 00:26:00-05:00  Queen St W / York St (City Hall)
4          7010.0 2017-01-01 00:25:00-05:00          King St W / Spadina Ave

   bike_id user_type                start_station_name_npl  \
0      NaN    Member            bay st college st east side
1      NaN    Member               niagara st richmond st w
2      NaN    Member  front st yonge st hockey hall of fame
3      NaN    Member          east liberty st pirandello st
4      NaN    Member              bathurst st queens quay w

           end_station_name_npl
0             bay st albert st
1          king st w fraser ave
2             church st wood st
3  queen st w york st city hall
4         king st w spadina ave
```

By populating the station name based on the available station id, we no longer have any records with missing station names.

[42]:
```
#Percentage of Data Containing Null Record
trips_data_missing = pd.DataFrame(df_trips_data.isnull().sum())
trips_data_missing = trips_data_missing.rename(columns={0:"count"})
trips_data_missing['percent_nulldata']=round(trips_data_missing['count']/
↪df_trips_data.shape[0] * 100,1)

trips_data_missing
```

[42]:

	count	percent_nulldata
trip_id	0	0.0
subscription_id	3415265	40.3
trip_duration	0	0.0
start_station_id	27386	0.3
start_time	0	0.0
start_station_name	0	0.0
end_station_id	29346	0.3
end_time	0	0.0
end_station_name	0	0.0
bike_id	3415265	40.3
user_type	0	0.0
start_station_name_npl	0	0.0
end_station_name_npl	0	0.0

6.2.2 Find Missing Station ID based on Station Name in Bike Station Dataset

A number of the station ID's are still missing in the bike trip dataframe, although this has been reduced to 0.3%. We thought perhaps we can further reduce the number of trips with missing station id based on information contained in the bike station dataset. In the stations dataset, the information about each station, including the station name, id, location (longitude and latitude), and capacity, is provided in the csv file called "bikeshare_stations.csv" located in the repository.

The function below (simple_replace_station_ID) searches the dataframe for missing station ids. For the trips with missing id, it searches for the station name in the bike station dataset. If there is a match, the associated station id is assigned to the missing station id.

[43]:
```
def simple_replace_station_ID(df, feature):
    #search stations data frame for missing ID's
    #create list of unique start station names
    feature_name = feature+'_station_name'
    feature_Id = feature+'_station_id'

    df_missing_station_id = df[df[feature_Id].isnull()]
    df = df[df[feature_Id].notnull()]

    lst_station_names = df_missing_station_id[feature_name+'_npl'].unique().
 ↪tolist()
    #itterate through list and replace station Id's if found
    for name in lst_station_names:
        #check name exists in list of station names
        if (stations['station_name_npl'] == name).sum()>0:
            mask = df_missing_station_id[feature_name+'_npl'] == name
            station_id = stations[stations['station_name_npl'] ==␣
 ↪name]['station_id'].to_numpy()[0]
            df_missing_station_id.loc[mask,feature_Id] = station_id
    df = df.append(df_missing_station_id)
    return df

df_trips_data = simple_replace_station_ID(df_trips_data,'start')
df_trips_data = simple_replace_station_ID(df_trips_data,'end')

print(df_trips_data.shape)
df_trips_data.head()
```

```
(8466220, 13)
```

[43]:

	trip_id	subscription_id	trip_duration	start_station_id	\
0	712441	NaN	274	7006.0	
1	712442	NaN	538	7046.0	
2	712443	NaN	992	7048.0	
3	712444	NaN	1005	7177.0	
4	712445	NaN	645	7203.0	

```
                start_time                          start_station_name  \
0 2017-01-01 00:03:00-05:00              Bay St / College St (East Side)
1 2017-01-01 00:03:00-05:00                   Niagara St / Richmond St W
2 2017-01-01 00:05:00-05:00  Front St / Yonge St (Hockey Hall of Fame)
3 2017-01-01 00:09:00-05:00              East Liberty St / Pirandello St
4 2017-01-01 00:14:00-05:00                  Bathurst St / Queens Quay W

   end_station_id                  end_time                  end_station_name  \
0          7021.0 2017-01-01 00:08:00-05:00                 Bay St / Albert St
1          7147.0 2017-01-01 00:12:00-05:00             King St W / Fraser Ave
2          7089.0 2017-01-01 00:22:00-05:00               Church St  / Wood St
3          7202.0 2017-01-01 00:26:00-05:00  Queen St W / York St (City Hall)
4          7010.0 2017-01-01 00:25:00-05:00            King St W / Spadina Ave

   bike_id user_type                 start_station_name_npl  \
0      NaN    Member              bay st college st east side
1      NaN    Member                 niagara st richmond st w
2      NaN    Member  front st yonge st hockey hall of fame
3      NaN    Member            east liberty st pirandello st
4      NaN    Member                bathurst st queens quay w

           end_station_name_npl
0              bay st albert st
1           king st w fraser ave
2              church st wood st
3  queen st w york st city hall
4          king st w spadina ave
```

[44]:
```
#Percentage of Data Containing Null Record
trips_data_missing = pd.DataFrame(df_trips_data.isnull().sum())
trips_data_missing = trips_data_missing.rename(columns={0:"count"})
trips_data_missing['percent_nulldata']=round(trips_data_missing['count']/
 ↪df_trips_data.shape[0] * 100,1)

trips_data_missing
```

[44]:

	count	percent_nulldata
trip_id	0	0.0
subscription_id	3415265	40.3
trip_duration	0	0.0
start_station_id	27386	0.3
start_time	0	0.0
start_station_name	0	0.0
end_station_id	29346	0.3
end_time	0	0.0
end_station_name	0	0.0
bike_id	3415265	40.3

```
user_type                   0               0.0
start_station_name_npl      0               0.0
end_station_name_npl        0               0.0
```

7 Variations in Spelling / Naming convention

In this section, we use the FuzzyWuzzy library to clean up inconsistent data entries. In the dataset, there are variations of the same word such as:

> Front St W vs Front St
>
> Lakeshore vs Lake Shore

These inconsistencies prevent us from assigning station ids from the station names because the code is unable to distinguish between the name variations. The fuzzywuzzy package helps to identify which string are closest to each other.

> Fuzzy matching: The process of automatically finding text strings that are very similar to the target string. In general, a string is considered "closer" to another one the fewer characters you'd need to change if you were transforming one string into another.

7.0.1 Token Set Approach

The fuzzywyuzzy library has various ways in which the 'match' score can be calculated. We have used the token set approach for its versatility. Using this approach, we tokenize both strings, but instead of immediately sorting and comparing, we split the tokens into two groups: intersection and remainder. We use those sets to build up a comparison string and a score greater than 90 (out of 100) to be considered a confident match.

[45]:
```
# function to replace rows in the provided column of the provided dataframe
# that match the provided string above the provided ratio with the provided␣
↪string
def fuzzy_replace_matches(df, column, feature, string_to_match, index,␣
↪fuzzy_scorer, min_ratio = 90):

    # find unique station names
    strings = df[column].unique()

    # get the top 10 closest matches to our input string
    matches = fuzzywuzzy.process.extract(string_to_match, strings,
                                         limit=10, scorer=fuzzy_scorer)

    # only get matches with a ratio > 90
    close_matches = [matches[0] for matches in matches if matches[1] >=␣
↪min_ratio]

    # get the rows of all the close matches in our dataframe
    row_mask = df[column].isin(close_matches)
```

```
    # replace all rows with close matches with the input matches
    df.loc[row_mask, column] = string_to_match
    #replace station Id's
    df.loc[row_mask,feature] = stations.loc[index,'station_id']

    return df
```

[46]:
```
#process missing start stations through fuzzywuzzy function
df_missing_station_id = df_trips_data[df_trips_data['start_station_id'].
 ↪isnull()]
df_intact = df_trips_data[df_trips_data['start_station_id'].notnull()]

for i in range(len(stations)):
    df_missing_station_id=fuzzy_replace_matches(df=df_missing_station_id,␣
 ↪column='start_station_name_npl',
                                                ␣
 ↪feature='start_station_id',
                                                ␣
 ↪string_to_match=stations['station_name_npl'][i],
                                                 index = i,
                                                 fuzzy_scorer =␣
 ↪fuzzywuzzy.fuzz.token_set_ratio)


#append back to original data frame
df_trips_data = df_intact.append(df_missing_station_id)

print('Trips Shape: ', df_trips_data.shape) # test to verify no data loss #␣
 ↪test to verify no data loss

#process missing end stations through fuzzywuzzy function
df_missing_station_id = df_trips_data[df_trips_data['end_station_id'].isnull()]
df_intact = df_trips_data[df_trips_data['end_station_id'].notnull()]

for i in range(len(stations)):
    df_missing_station_id=fuzzy_replace_matches(df=df_missing_station_id,␣
 ↪column='end_station_name_npl',
                                                ␣
 ↪feature='end_station_id',
                                                ␣
 ↪string_to_match=stations['station_name_npl'][i],
                                                 index = i,
                                                 fuzzy_scorer =␣
 ↪fuzzywuzzy.fuzz.token_set_ratio)
```

```
#append back to original data frame
df_trips_data = df_intact.append(df_missing_station_id)
df_trips_data.sort_index(inplace=True)
print('Trips Shape: ', df_trips_data.shape) # test to verify no data loss #␣
 ↪test to verify no data loss
```

```
Trips Shape:  (8466220, 13)
Trips Shape:  (8466220, 13)
```

[47]:
```
#Percentage of Data Containing Null Record
trips_data_missing = pd.DataFrame(df_trips_data.isnull().sum())
trips_data_missing = trips_data_missing.rename(columns={0:"count"})
trips_data_missing['percent_nulldata']=round(trips_data_missing['count']/
 ↪df_trips_data.shape[0] * 100,1)

trips_data_missing
```

[47]:

	count	percent_nulldata
trip_id	0	0.0
subscription_id	3415265	40.3
trip_duration	0	0.0
start_station_id	16293	0.2
start_time	0	0.0
start_station_name	0	0.0
end_station_id	18731	0.2
end_time	0	0.0
end_station_name	0	0.0
bike_id	3415265	40.3
user_type	0	0.0
start_station_name_npl	0	0.0
end_station_name_npl	0	0.0

For `trips_data`, we can see that `'end_station_id'` and `'start_station_id'` has 18731 and 16293 missing values respectively, which is only about 0.2% of the dataset. We were able to bring the total number of records with null station id to under 1%. Below, we check what the station names were for the remainder of the records for which a station id remains missing. The closest matches using the fuzzywuzzy algorithm is also shown to see if we can make any manual corrections.

It appears that for the remaining records with missing station id, the station names could not be matched confidently with any of the station names found in the bike station dataset. This could mean that the station no longer exists or was moved to a different location.

First let's look at the records with missing end station id. We determined that there are 6 station names in the trip data that cannot be confidently matched with any of the station names found in the bike station data.

[48]:
```
test=df_trips_data[df_trips_data['end_station_id'].
 ↪isnull()]['end_station_name_npl'].unique()
print(test)
```

```
test2_end=pd.DataFrame()

for i in range(len(test)):
    matches = fuzzywuzzy.process.extract(␣
 ↪str(test[i]),stations['station_name_npl'], limit=5, scorer=fuzzywuzzy.fuzz.
 ↪token_set_ratio)
    a_series = pd.Series([test[i],␣
 ↪matches[0],matches[1],matches[2],matches[3],matches[4]])
    test2_end = test2_end.append(a_series, ignore_index=True)
test2_end=test2_end.rename(columns={0:'station_name',1:'1st_match',2:
 ↪'2nd_match',3:'3rd_match',4:'4th_match',5:'5th_match'})
test2_end.sort_values(by='station_name', ascending=True).
 ↪reset_index(inplace=True)
test2_end

del test2_end
```

```
['fringe next stage 7219' 'michael sweet ave st patrick st'
 'roxton rd college st' 'lansdowne subway green p'
 'margueretta st college st' 'base station']
```

We repeat the step for the records with missing start station id. We also determined that there are 6 station names in the trip dataset that cannot be matched with any of the station names found in the bike station data. The 6 station names are the same for both the start station and end station.

[49]:
```
test=df_trips_data[df_trips_data['start_station_id'].
 ↪isnull()]['start_station_name_npl'].unique()
print(test)

test2_start=pd.DataFrame()

for i in range(len(test)):
    matches = fuzzywuzzy.process.extract(␣
 ↪str(test[i]),stations['station_name_npl'], limit=5, scorer=fuzzywuzzy.fuzz.
 ↪token_set_ratio)
    a_series = pd.Series([test[i],␣
 ↪matches[0],matches[1],matches[2],matches[3],matches[4]])
    test2_start = test2_start.append(a_series, ignore_index=True)
test2_start=test2_start.rename(columns={0:'station_name',1:'1st_match',2:
 ↪'2nd_match',3:'3rd_match',4:'4th_match',5:'5th_match'})
test2_start.sort_values(by='station_name', ascending=True).
 ↪reset_index(inplace=True)
test2_start

del test2_start
```

```
['michael sweet ave st patrick st' 'roxton rd college st'
```

```
 'lansdowne subway green p' 'margueretta st college st'
 'fringe next stage 7219' 'base station']
```

We recommend the City of Toronto data team look into the 6 stations are found in the trip dataset, but not in the bike station data set to determine how they would like to address this issue in their database moving forward.

[50]:
```
df_trips_data = df_trips_data[~(df_trips_data['start_station_id'].isnull() |␣
 ↪df_trips_data['end_station_id'].isnull())]

print('Number of records in Trip Dataframe: ', df_trips_data.shape[0])

#Percentage of Data Containing Null Record
trips_data_missing = pd.DataFrame(df_trips_data.isnull().sum())
trips_data_missing = trips_data_missing.rename(columns={0:"count"})
trips_data_missing['percent_nulldata']=round(trips_data_missing['count']/
 ↪df_trips_data.shape[0] * 100,1)

trips_data_missing
```

```
Number of records in Trip Dataframe:  8431951
```

[50]:

	count	percent_nulldata
trip_id	0	0.0
subscription_id	3380996	40.1
trip_duration	0	0.0
start_station_id	0	0.0
start_time	0	0.0
start_station_name	0	0.0
end_station_id	0	0.0
end_time	0	0.0
end_station_name	0	0.0
bike_id	3380996	40.1
user_type	0	0.0
start_station_name_npl	0	0.0
end_station_name_npl	0	0.0

8 7. Bike Trip Data - Outliers

In this section, we will determine what kind of outliers we have in the data set.

Outliers in datasets can be both good and bad. One the one hand, they may contain important information while on the other hand, they skew your visualizations and may bias your models.

Below shows the summary statistics for bike trip dataset using .describe().

[51]:
```
df_trips_data.describe()
```

[51]:
```
            trip_id  subscription_id  trip_duration  start_station_id  \
count  8.431951e+06     5.050955e+06   8.431951e+06      8.431951e+06
mean   5.518663e+06     4.852057e+05   1.068391e+03      7.176306e+03
std    2.761344e+06     1.668259e+05   1.148911e+04      1.436678e+02
min    7.124410e+05     6.537700e+04   0.000000e+00      7.000000e+03
25%    3.146136e+06     3.373850e+05   4.400000e+02      7.051000e+03
50%    5.527570e+06     4.751620e+05   7.200000e+02      7.154000e+03
75%    7.927266e+06     6.056830e+05   1.137000e+03      7.266000e+03
max    1.029388e+07     8.634190e+05   1.240378e+07      7.660000e+03

       end_station_id       bike_id
count    8.431951e+06  5.050955e+06
mean     7.175651e+03  3.146359e+03
std      1.434518e+02  1.693900e+03
min      7.000000e+03  1.400000e+01
25%      7.051000e+03  1.844000e+03
50%      7.153000e+03  3.222000e+03
75%      7.264000e+03  4.399000e+03
max      7.660000e+03  6.927000e+03
```

Looking at the summary statistics, we can tell that there is something odd with the data. These observations include: - the minimum duration of a trip was 0 seconds - the maximum duration of a trip was 1.240378e+07, which is equivalent to approximately 143 days

It is assumed for the purpose of this analysis that any trip less than 1 minute is considered a false trip.

It is interesting to note that BikeShare Toronto charges an overage fee of 4 dollars per additional 30 minutes of trip time. For a trip duration of 1.240378e+07 seconds or 143 days, that is about $27,559 owed to the City of Toronto.

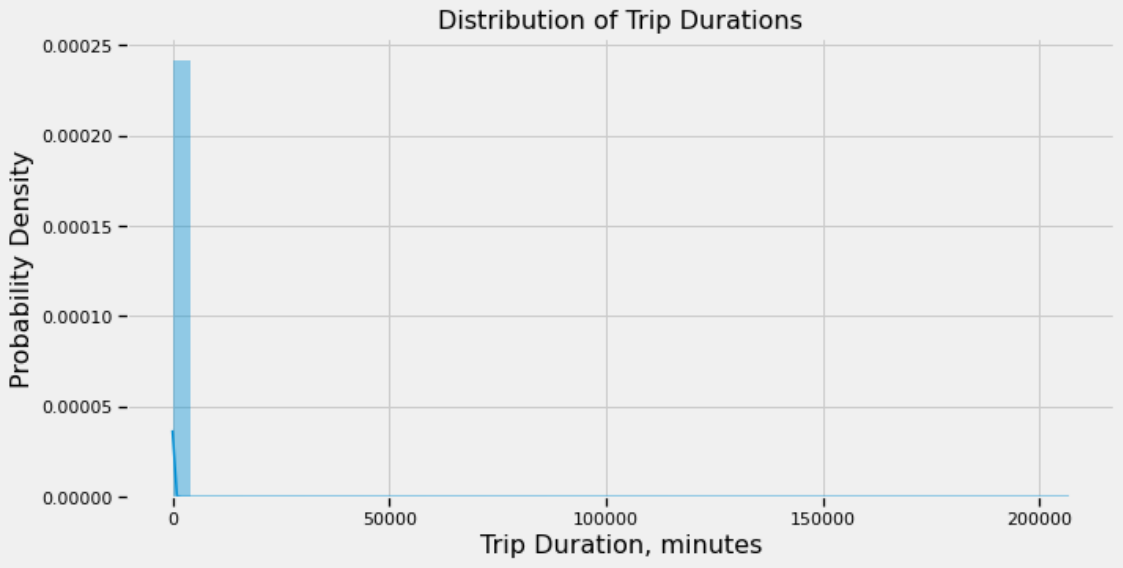
[52]:
```
original_count=df_trips_data.shape[0]
df_trips_data = df_trips_data[df_trips_data['trip_duration'] >= 60]
print('Number of records in Trip Dataframe: ', df_trips_data.shape[0])
print('Number of records removed from Trip Dataframe: ', original_count -␣
 ↪df_trips_data.shape[0])
```

```
Number of records in Trip Dataframe:  8381601
Number of records removed from Trip Dataframe:  50350
```

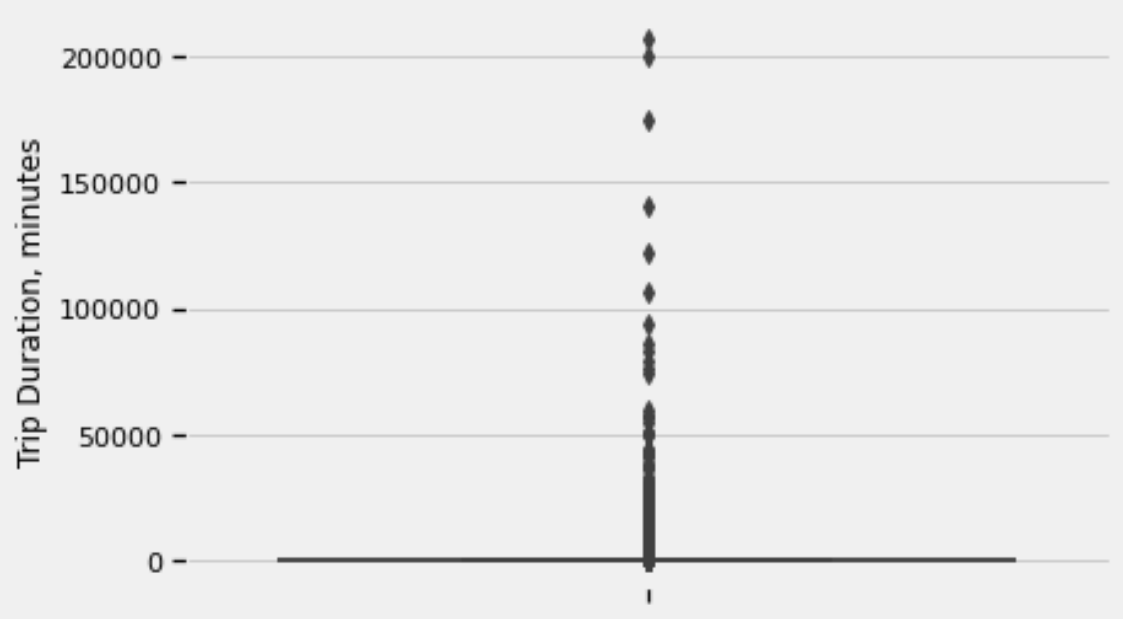
Let's look at the distribution of the trip duration on a histogram + density plot as well as on a box plot.

[53]:
```
plt.figure(figsize=(10,5))
trip_dur=sns.distplot(a=df_trips_data['trip_duration']/60)
trip_dur.axes.set_title("Distribution of Trip Durations",
                        fontsize=16)
trip_dur.set_xlabel("Trip Duration, minutes", fontsize=16)
trip_dur.set_ylabel("Probability Density",
                    fontsize=16)
```

```
plt.show()

del trip_dur
```



[54]:
```
a=df_trips_data['trip_duration']/60
sns.boxplot(y=a).set(
    ylabel='Trip Duration, minutes')
plt.show()

del a
```



It is clear from both plots that there are extreme values in the dataset and these values affect the way the dataset is represented on the graphs. It is important to note, however, that outliers may contain important information about bike sharing, particularly about the way data is collected. We recommend that the data team at the City of Toronto conduct further investigation into whether these outliers are real and how they occur in the data set if they are not.

For the purpose of our analysis, we will remove these outliers because our objective is to analyse and model the typical and normal behavior of the bike share users that can be used to support the City in its future plans to expand the bike-share system. Particularly when using machine learning, models can be greatly improved by removing outliers from the data set.

We will use the interquartile range to remove outliers from the trip dataset. Any `'trip_duration'` values less than `Q1 - 1.5 * IQR` and greater than `Q3 + 1.5 * IQR` will be considered outliers and thus removed from the dataset.

- Q1: The first quartile (`.quantile(0.25)`)
- Q3: The third quartile (`.quantile(0.75)`)
- IQR: The first quartil (`Q3 - Q1`)

[55]:
```
original_count=df_trips_data.shape[0]

#investigate the trip duration
Q1=df_trips_data['trip_duration'].quantile(0.25)  # Q1
Q2=df_trips_data['trip_duration'].quantile(0.5)  # median
Q3=df_trips_data['trip_duration'].quantile(0.75)  # Q3
IQR=Q3-Q1

print("Q1=",Q1)
print("Q2=",Q2)
print("Q3=",Q3)
print("IQR=",IQR)


df_trips_data = df_trips_data[(df_trips_data['trip_duration'] <= (Q3 + 1.5 *␣
 ↪IQR)) & (df_trips_data['trip_duration'] >= (Q1 - 1.5 * IQR))]

# View DataFrame
print('Number of records in Trip Dataframe: ', df_trips_data.shape[0])
print('Number of records removed from Trip Dataframe: ', original_count -␣
 ↪df_trips_data.shape[0])
df_trips_data.sort_values(by=['trip_duration'], ascending=False).head()
```

```
Q1= 445.0
Q2= 724.0
Q3= 1141.0
IQR= 696.0
Number of records in Trip Dataframe:  8008080
```

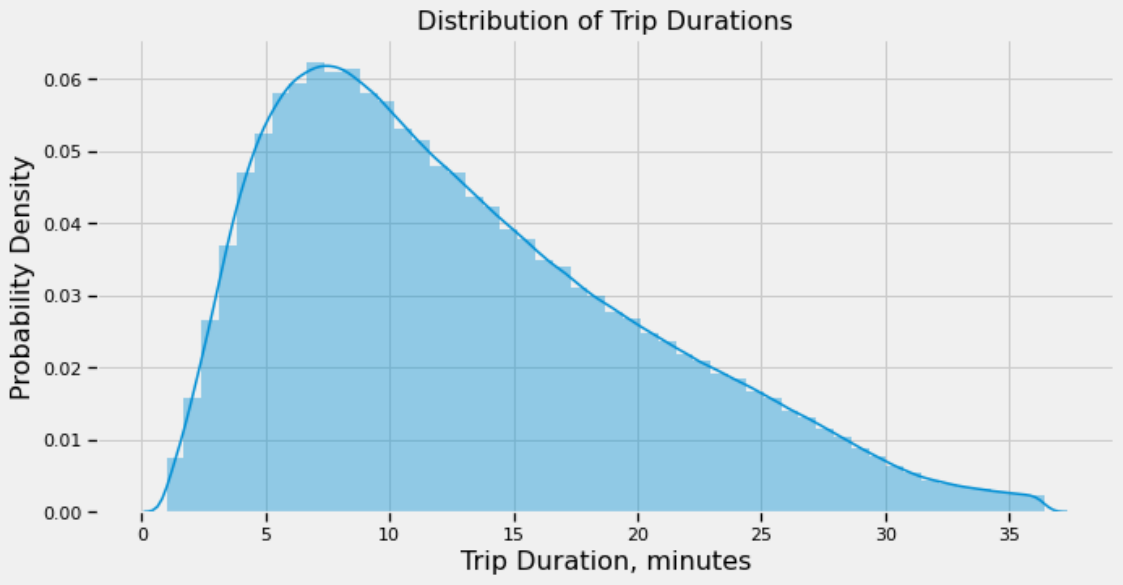
```
Number of records removed from Trip Dataframe:  373521
```

[55]:

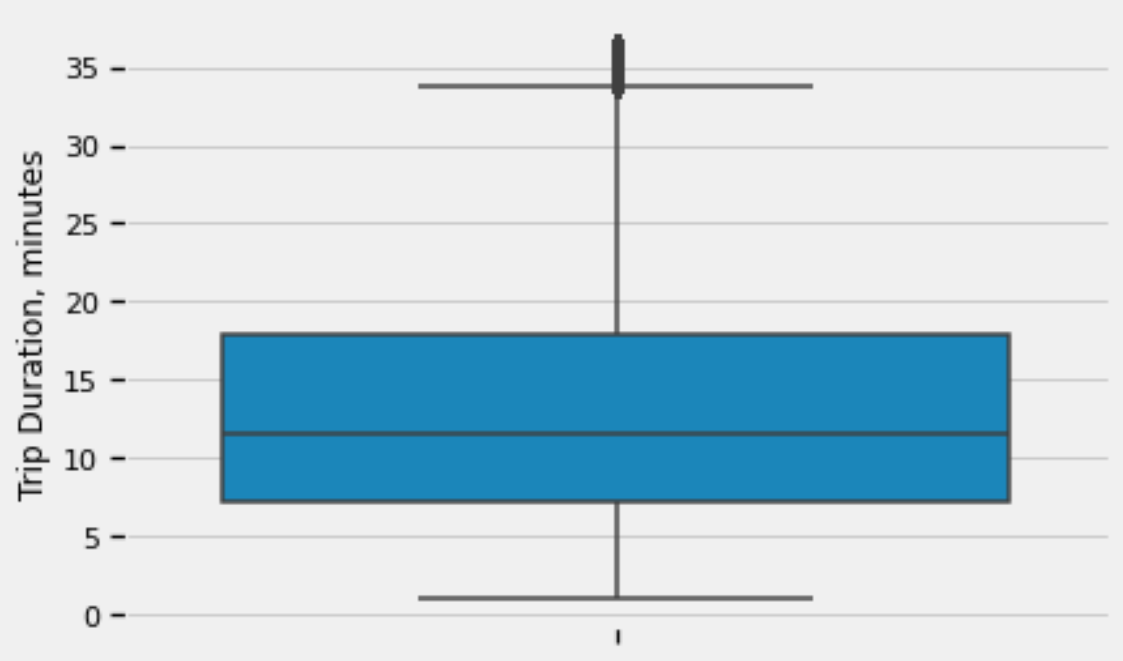
	trip_id	subscription_id	trip_duration	start_station_id	start_time	start_station_name	end_station_id	end_time	end_station_name	bike_id	user_type	start_station_name_npl	end_station_name_npl
102959	1597516	NaN	2185	7086.0	2017-08-16 07:58:00-05:00	Woodbine Subway Green P (Cedarvale Ave)	7008.0	2017-08-16 08:34:00-05:00	Wellesley St / Queen's Park Cres	NaN	Member	woodbine subway green p cedarvale ave	wellesley st queens park cres
299518	8742412	683762.0	2185	7075.0	2020-07-21 15:18:00-05:00	Queens Quay W / Dan Leckie Way	7521.0	2020-07-21 15:55:00-05:00	Emerson Ave / Bloor St W - SMART	5669.0	Annual Member	queens quay w dan leckie way	emerson ave bloor st w smart
15112	3890815	NaN	2185	7171.0	2018-09-02 17:34:00-05:00	Ontario Place Blvd / Lakeshore Blvd W	7226.0	2018-09-02 18:11:00-05:00	Lakeshore Blvd W / The Boulevard Club	NaN	Casual Member	ontario place blvd lakeshore blvd w	lakeshore blvd w the boulevard club
292877	8734838	687271.0	2185	7220.0	2020-07-20 21:12:00-05:00	Lake Shore Blvd W / Ellis Ave	7518.0	2020-07-20 21:48:00-05:00	Lake Shore Blvd W / Colborne Lodge Dr	2409.0	Casual Member	lake shore blvd w ellis ave	lake shore blvd w colborne lodge dr
270162	5510145	306263.0	2185	7117.0	2019-06-29 08:20:00-05:00	Castle Frank Station	7329.0	2019-06-29 08:56:00-05:00	Crawford St / Queen St W	4383.0	Annual Member	castle frank station	crawford st queen st w

Now let's visualise the `'trip_duration'` in a histogram + density plot and box plot again. Please note that the `'trip_duration'` is displayed in minutes

[56]:
```
plt.figure(figsize=(10,5))
trip_dur=sns.distplot(a=df_trips_data['trip_duration']/60)
trip_dur.axes.set_title("Distribution of Trip Durations",
                        fontsize=16)
trip_dur.set_xlabel("Trip Duration, minutes", fontsize=16)
trip_dur.set_ylabel("Probability Density",
                    fontsize=16)
plt.show()

del trip_dur
```



[57]:
```
a=df_trips_data['trip_duration']/60
sns.boxplot(y=a).set(
    ylabel='Trip Duration, minutes')
plt.show()

del a
```



[58]:
```
#final check in trip dataframe to ensure there is no duplicate trip id
print("{:.0f} % of the trip Id's are unique".format(df_trips_data['trip_id'].
 ↪nunique()/df_trips_data.shape[0]*100))
```

```
100 % of the trip Id's are unique
```

9 8. Bike Trip Data - Merging with Bike Station and Weather Data

We will merge the trip data, bike station and weather dataset into one dataframe. By merging the bike station data, this will allow us to merge the coordinates of the start and end location of each trip. By merging the weather data, we will be able to know associate each trip with the weather condition at the time the trip was taken.

[59]:
```
#Merge the stations dataframe and the df_trips_data
df_stations = stations[['station_id', 'lat', 'lon']]

#Add start stations location
df_trips_data = pd.merge(df_trips_data,df_stations, left_on =␣
 ↪'start_station_id', right_on = 'station_id', how = 'left')
df_trips_data.rename(columns = {'lat':'start_station_lat', 'lon':␣
 ↪'start_station_lon'},inplace = True)
df_trips_data.drop('station_id', axis = 1, inplace = True)

#Add end stations location
```

```
df_trips_data = pd.merge(df_trips_data,df_stations, left_on = 'end_station_id',␣
↪right_on = 'station_id', how = 'left')
df_trips_data.rename(columns = {'lat':'end_station_lat', 'lon':␣
↪'end_station_lon'},inplace = True)
df_trips_data.drop('station_id', axis = 1, inplace = True)

df_trips_data.head()
```

[59]:

	trip_id	subscription_id	trip_duration	start_station_id	start_time	start_station_name	end_station_id	end_time	end_station_name	bike_id	user_type	start_station_name_npl	end_station_name_npl	start_station_lat	start_station_lon	end_station_lat	end_station_lon
0	712441	NaN	274	7006.0	2017-01-01 00:03:00-05:00	Bay St / College St (East Side)	7021.0	2017-01-01 00:08:00-05:00	Bay St / Albert St	NaN	Member	bay st college st east side	bay st albert st	43.660439	-79.385525	43.653264	-79.382458
1	4966537	241654.0	670	7195.0	2019-05-01 00:00:00-05:00	Ulster St / Bathurst St	7000.0	2019-05-01 00:11:00-05:00	Fort York Blvd / Capreol Ct	2288.0	Annual Member	ulster st bathurst st	fort york blvd capreol ct	43.660000	-79.408889	43.639832	-79.395954
2	2176076	NaN	349	7272.0	2017-11-01 00:00:00-05:00	Yonge St / Dundonald St - SMART	7041.0	2017-11-01 00:06:00-05:00	Edward St / Yonge St	NaN	Member	yonge st dundonald st smart	edward st yonge st	43.665801	-79.384796	43.656729	-79.382736
3	5206483	259072.0	1400	7155.0	2019-06-01 00:00:00-05:00	Bathurst St / Lennox St	7105.0	2019-06-01 00:23:00-05:00	Queen St E / Sackville St	778.0	Annual Member	ulster st bathurst st	queen st e sackville st	43.663808	-79.410491		
4	868766	NaN	1051	7054.0	2017-04-01 00:02:00-05:00	Navy Wharf Ct. / Bremner Blvd.	7051.0	2017-04-01 00:19:00-05:00	Wellesley St E / Yonge St Green P	NaN	Member	navy wharf ct bremner blvd	wellesley st e yonge st green p	43.640722	-79.391051		

```
3         43.656111       -79.361389
4         43.654879       -79.375091
```

It has been identified that some of the stations named in in the trip data are not included in the bike station dataset. For this reason, we could not assign the location coordinates to the trip record. The stations not included in the station dataset are: - Toronto Bike Shop - Wolfpack - 7219 - 135 Queen's Wharf - SMART - 55 Magnificent Rd. Garage - PBSC-OPS - Make Invisible - 7218

We highly recommend the City of Toronto to investigate verify why these bike stations are not included in the bike station dataset.

There are 242 records missing the start coordinate information. There are 426 records missing the end coordinate information. We have removed these records from our analysis as these bike stations were not included in the provided bike station list. We recommend that the City of Toronto investigate this further and advise us as to whether they would like these trip records included in our analysis. For now, they will be removed as we are uncertain about h

[60]:
```
#end station ids that have null coordinate information
print(df_trips_data[df_trips_data['end_station_lon'].
 ↪isnull()]['end_station_id'].unique())
#end station names that have null coordinate information
print(df_trips_data[df_trips_data['end_station_lon'].
 ↪isnull()]['end_station_name'].unique())
```

```
[7394. 7219. 7532. 7218. 7511.]
['Toronto Bike Shop' 'Wolfpack - 7219' '135 Queen's Wharf - SMART'
 'Make Invisible - 7218' 'PBSC-OPS']
```

[61]:
```
#start station ids that have null coordinate information
print(df_trips_data[df_trips_data['start_station_lon'].
 ↪isnull()]['start_station_id'].unique())
#start station names that have null coordinate information
print(df_trips_data[df_trips_data['start_station_lon'].
 ↪isnull()]['start_station_name'].unique())
```

```
[7394. 7219. 7532. 7393. 7218. 7511.]
['Toronto Bike Shop' 'Wolfpack - 7219' '135 Queen's Wharf - SMART'
 '55 Magnificent Rd. Garage' 'Make Invisible - 7218' 'PBSC-OPS']
```

[62]:
```
#Percentage of Data Containing Null Record for␣
 ↪'start_station_lat','start_station_lon','end_station_lat','end_station_lon'
df_trips_data_missing = pd.DataFrame(df_trips_data.isnull().sum())
df_trips_data_missing = df_trips_data_missing.rename(columns={0:"count"})
df_trips_data_missing['percent_nulldata']=round(df_trips_data_missing['count']/
 ↪df_trips_data.shape[0] * 100,1)

df_trips_data_missing.
 ↪loc[['start_station_lat','start_station_lon','end_station_lat','end_station_lon'],:
 ↪]
```

[62]:

	count	percent_nulldata
start_station_lat	242	0.0
start_station_lon	242	0.0
end_station_lat	426	0.0
end_station_lon	426	0.0

[63]:
```
#Remove all trip records with start or end bike station not included in the␣
 ↪bike station dataset
df_trips_data=df_trips_data[df_trips_data['start_station_lat'].notnull() &␣
 ↪df_trips_data['end_station_lat'].notnull()]
```

We use the .merge() function to combine weather_data and trips_data using datetime information and set the output to a new variable called df_data_merged. In trips_data there are two time stamps corresponding to the start and end of the ride. We have used the'start_time' of the rides to merge. We believe this is the most appropriate method because we anticipate that bike riders will make a decision to use the bike or not based on the weather condition at the start of the trip, not the end.

The datetimes in the trips_data datetimes contain information down to the minute, while weather_data is reported every hour. Thus, we merged based on a common year, month, day, hour.

[64]:
```
df_trips_data['merge_time']=df_trips_data['start_time'].dt.round('H')
df_data_merged = pd.merge(df_trips_data, df_weather_data, left_on='merge_time',␣
 ↪right_index=True, how='left')
df_data_merged.set_index('trip_id', inplace = True)
df_data_merged.sort_index(inplace=True)
print('Number of records in Merged Dataframe: ', df_data_merged.shape[0])
```

```
Number of records in Merged Dataframe:  8007423
```

Based on the information below, we know that: - 0.6% of the trips do not have information about the temperature - 0.7% of the trips do not have information about dew point temperature and relative humidity - 0.2% of the trips do not have information about the wind speed and direction

We will leave these trip records in the dataset for analyses that does not require the weather data.

[65]:
```
#Percentage of Data Containing Null Record
merged_data_missing = pd.DataFrame(df_data_merged.isnull().sum())
merged_data_missing = merged_data_missing.rename(columns={0:"count"})
merged_data_missing['percent_nulldata']=round(merged_data_missing['count']/
 ↪df_data_merged.shape[0] * 100,1)

merged_data_missing.loc[df_data_merged.columns[17:],:]
```

[65]:

	count	percent_nulldata
year	0	0.0
month	0	0.0
day	0	0.0
time	0	0.0

```
temp_c              48475              0.6
dew_point_temp_c    59007              0.7
rel_hum_            57615              0.7
wind_dir_10s_deg    14177              0.2
wind_spd_kmh        14177              0.2
visibility_km           0              0.0
stn_press_kpa       50633              0.6
hmdx              5151443             64.3
wind_chill        7578399             94.6
weather                 0              0.0
```

10 9. User Type Definition

The naming convention is not uniform Annual Member is called:

Member

Annual Member

Casual Member is called:

Member

Casual Member

The below code sets all to either 'casual member' or 'annual member' .

[66]:
```
df_data_merged['user_type'].unique()
```

[66]:
```
array(['Member', 'Casual', 'Annual Member', 'Casual Member'], dtype=object)
```

[67]:
```
#Make all string lower case in the column 'user type'
df_data_merged['user_type'] = df_data_merged['user_type'].str.lower()

#Replace string to make user type consistent through dataframe
df_data_merged['user_type'] = df_data_merged['user_type'].replace({'member':
 ↪'annual member', 'casual':'casual member'})

#Show unique values in column
df_data_merged['user_type'].unique()
```

[67]:
```
array(['annual member', 'casual member'], dtype=object)
```

11 10. Export the Cleaned Data

The merged dataframe will be saved as a csv file to use for the Exploratory Data Analysis.

[68]:
```
#Export the merged dataframe in csv format
df_data_merged.to_csv('df_merged_data.csv')
```

[69]:
```
#See the DataFrame
df_data_merged.head()
```

[69]:

```
         subscription_id  trip_duration  start_station_id  \
trip_id
712441               NaN            274            7006.0
712442               NaN            538            7046.0
712443               NaN            992            7048.0
712444               NaN           1005            7177.0
712445               NaN            645            7203.0

                       start_time                         start_station_name  \
trip_id
712441  2017-01-01 00:03:00-05:00            Bay St / College St (East Side)
712442  2017-01-01 00:03:00-05:00                 Niagara St / Richmond St W
712443  2017-01-01 00:05:00-05:00  Front St / Yonge St (Hockey Hall of Fame)
712444  2017-01-01 00:09:00-05:00            East Liberty St / Pirandello St
712445  2017-01-01 00:14:00-05:00                Bathurst St / Queens Quay W

         end_station_id                  end_time  \
trip_id
712441          7021.0 2017-01-01 00:08:00-05:00
712442          7147.0 2017-01-01 00:12:00-05:00
712443          7089.0 2017-01-01 00:22:00-05:00
712444          7202.0 2017-01-01 00:26:00-05:00
712445          7010.0 2017-01-01 00:25:00-05:00

                        end_station_name  bike_id      user_type  … temp_c  \
trip_id                                                           …
712441                 Bay St / Albert St      NaN  annual member  …    1.5
712442             King St W / Fraser Ave      NaN  annual member  …    1.5
712443               Church St  / Wood St      NaN  annual member  …    1.5
712444   Queen St W / York St (City Hall)      NaN  annual member  …    1.5
712445            King St W / Spadina Ave      NaN  annual member  …    1.5

        dew_point_temp_c  rel_hum_  wind_dir_10s_deg  wind_spd_kmh  \
trip_id
712441             -3.6      69.0              26.0          39.0
712442             -3.6      69.0              26.0          39.0
712443             -3.6      69.0              26.0          39.0
712444             -3.6      69.0              26.0          39.0
712445             -3.6      69.0              26.0          39.0

         visibility_km stn_press_kpa  hmdx  wind_chill    weather
trip_id
712441            16.1         99.81   NaN         NaN  clear_day
712442            16.1         99.81   NaN         NaN  clear_day
```

```
712443            16.1         99.81   NaN         NaN  clear_day
712444            16.1         99.81   NaN         NaN  clear_day
712445            16.1         99.81   NaN         NaN  clear_day

[5 rows x 31 columns]
```

This concludes the data wrangling and cleaning process for the bike trip and weather data. The final output of this file is a dataframe that includes both bike share and weather data features from 2017 to 2020 with each row corresponding to a trip.