Final_report

May 16, 2019

1 Load data and preprocess data

In [1]:
```
# ### Get data from bigquery and save it to local

# import pandas as pd
# import os
# pd.set_option('display.max_columns', 500)
# os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="../ecbm4040-yt2639-d3ee184230ba.json"
# from google.cloud import bigquery
# client = bigquery.Client()
# query = (
#     """
#     SELECT * FROM
#     (
#     SELECT *,
#     DATETIME_DIFF( dropoff_datetime, pickup_datetime, SECOND) as travel_time,
#      EXTRACT (DATE FROM pickup_datetime) as date_of_year,
#      EXTRACT (DAY FROM pickup_datetime) as day_of_year,
#      EXTRACT (MONTH FROM pickup_datetime) as month_of_year,
#      EXTRACT (YEAR FROM pickup_datetime) as year_of_year
#     FROM `bigquery-public-data.new_york_taxi_trips.tlc_yellow_trips_2016` ) a
#     LEFT JOIN
#     (
#      select concat(year,'-',mo,'-',da) as date_of_year2,temp,visib,wdsp,gust,prcp,sn
#      from `bigquery-public-data.noaa_gsod.gsod2016` where stn='725053'
#     ) weather_data
#     on CAST(a.date_of_year AS STRING)=weather_data.date_of_year2 WHERE CAST(year_of_
#     LIMIT 3000000"""
# )

# df = pd.io.gbq.read_gbq(query,dialect='standard')
```

In [2]:
```
# Turn off the warnings
import warnings; warnings.simplefilter('ignore')
```

In [3]:
```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns

# Get data from saved pkl
df = pd.read_pickle('../Data/raw data.pkl') # this is 3m data
print(df.shape)
```

```
(3000000, 36)
```

In [5]:
```
def RADIANS(x):
    rad = x * np.pi / 180
    return rad
def RADIANS_TO_KM(y):
    distance_to_km = 111.045 * 180 * y / np.pi
    return distance_to_km
def HAVERSINE(lat1, long1, lat2, long2):
    distance = RADIANS_TO_KM(np.arccos(np.cos(RADIANS(lat1)) * np.cos(RADIANS(lat2)) *
    return distance

# Add Manhattan distance
df['distancce_in_km'] = (HAVERSINE(df.pickup_latitude, df.pickup_longitude, df.dropoff_
```

In [6]:
```
# Because weather stations may not report when a certain weather do not happen,
# this leads to so many missing data in weather database.
df['sndp'] = df['sndp'].replace(999.9, 0)

# Convert fog, rain_drizzle, snow_ice_pellets, string to int type
df['vendor_id'] = df['vendor_id'].astype(int)
df['fog'] = df['fog'].astype(int)
df['rain_drizzle'] = df['rain_drizzle'].astype(int)
df['snow_ice_pellets'] = df['snow_ice_pellets'].astype(int)
df['wdsp'] = df['wdsp'].astype(float)

# Compute speed in km/h
df['speed'] = (df.trip_distance/(df.travel_time/3600))

# Add a new column indicating weekday
df['weekday'] = df.pickup_datetime.dt.weekday_name
df['day_of_week'] = df.pickup_datetime.dt.weekday

# Add pick-up hour
df['pickup_hour'] = df.pickup_datetime.dt.hour
```

Data Cleaning and Feature Engineering

In [7]:
```
# Clean NaN records in the dataset
df_clean = df.dropna()
```

```python
# Clean samples with travel time over 2.5 hr or under 1 min
df_clean = df_clean[df_clean['travel_time'] <= 9000]
df_clean = df_clean[df_clean['travel_time'] >= 60]

# Drop records with 0, 7, 8, 9 passengers
df_clean = df_clean[df_clean['passenger_count'] != 0]
df_clean = df_clean[df_clean['passenger_count'] <= 6]

# Discard outliers in trip diatance
df_clean = df_clean[df_clean['trip_distance'] <= 40]
df_clean = df_clean[df_clean['trip_distance'] != 0]

# Filter out records located out of NYC
df_clean = df_clean[df_clean['pickup_longitude'] <= -73.7]
df_clean = df_clean[df_clean['pickup_longitude'] >= -74.2]
df_clean = df_clean[df_clean['pickup_latitude'] <= 40.9]
df_clean = df_clean[df_clean['pickup_latitude'] >= 40.5]
df_clean = df_clean[df_clean['dropoff_longitude'] <= -73.7]
df_clean = df_clean[df_clean['dropoff_longitude'] >= -74.2]
df_clean = df_clean[df_clean['dropoff_latitude'] <= 40.9]
df_clean = df_clean[df_clean['dropoff_latitude'] >= 40.5]

# Clean records with speed beyond 120 km/h
df_clean = df_clean[df_clean['speed'] != 0]
df_clean = df_clean[df_clean['speed'] <= 120]

# Clean negative fare and outliers
df_clean = df_clean[df_clean['fare_amount'] > 0]
df_clean = df_clean[df_clean['tip_amount'] >= 0]

# Clean the missing weather data
df_clean = df_clean[df_clean['visib'] != 999.9]
df_clean = df_clean[df_clean['gust'] != 999.9]

df_clean.reset_index(drop=True, inplace=True)
df_clean.shape
```

Out[7]: (1106579, 41)

In [8]:
```python
df_clean.columns
```

Out[8]:
```
Index(['vendor_id', 'pickup_datetime', 'dropoff_datetime', 'passenger_count',
       'trip_distance', 'pickup_longitude', 'pickup_latitude', 'rate_code',
       'store_and_fwd_flag', 'dropoff_longitude', 'dropoff_latitude',
       'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
       'tolls_amount', 'imp_surcharge', 'total_amount', 'travel_time',
       'date_of_year', 'day_of_year', 'month_of_year', 'year_of_year',
```

```
       'date_of_year2', 'temp', 'visib', 'wdsp', 'gust', 'prcp', 'sndp', 'fog',
       'rain_drizzle', 'snow_ice_pellets', 'hail', 'thunder',
       'distancce_in_km', 'speed', 'weekday', 'day_of_week', 'pickup_hour'],
      dtype='object')
```

Data Dummify

In [9]:
```
# Weekday
dummy = pd.get_dummies(df_clean['weekday'], prefix='weekday')
dummy.drop(dummy.columns[0], axis=1, inplace=True) #avoid dummy trap
df_dummy = pd.concat([df_clean,dummy], axis = 1)

# Month
dummy = pd.get_dummies(df_clean['month_of_year'], prefix='month')
dummy.drop(dummy.columns[0], axis=1, inplace=True) #avoid dummy trap
df_dummy = pd.concat([df_dummy,dummy], axis = 1)

# pickup hour
dummy = pd.get_dummies(df_clean['pickup_hour'], prefix='pickup_hour')
dummy.drop(dummy.columns[0], axis=1, inplace=True) #avoid dummy trap
df_dummy = pd.concat([df_dummy,dummy], axis = 1)

# Flag
dummy = pd.get_dummies(df_clean['store_and_fwd_flag'], prefix='flag')
dummy.drop(dummy.columns[0], axis=1, inplace=True) #avoid dummy trap
df_dummy = pd.concat([df_dummy,dummy], axis = 1)

df_dummy.shape
```

Out[9]:
```
(1106579, 76)
```

In [10]:
```
df_dummy.columns
```

Out[10]:
```
Index(['vendor_id', 'pickup_datetime', 'dropoff_datetime', 'passenger_count',
       'trip_distance', 'pickup_longitude', 'pickup_latitude', 'rate_code',
       'store_and_fwd_flag', 'dropoff_longitude', 'dropoff_latitude',
       'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
       'tolls_amount', 'imp_surcharge', 'total_amount', 'travel_time',
       'date_of_year', 'day_of_year', 'month_of_year', 'year_of_year',
       'date_of_year2', 'temp', 'visib', 'wdsp', 'gust', 'prcp', 'sndp', 'fog',
       'rain_drizzle', 'snow_ice_pellets', 'hail', 'thunder',
       'distancce_in_km', 'speed', 'weekday', 'day_of_week', 'pickup_hour',
       'weekday_Monday', 'weekday_Saturday', 'weekday_Sunday',
       'weekday_Thursday', 'weekday_Tuesday', 'weekday_Wednesday', 'month_2',
       'month_3', 'month_4', 'month_5', 'month_6', 'pickup_hour_1',
       'pickup_hour_2', 'pickup_hour_3', 'pickup_hour_4', 'pickup_hour_5',
       'pickup_hour_6', 'pickup_hour_7', 'pickup_hour_8', 'pickup_hour_9',
       'pickup_hour_10', 'pickup_hour_11', 'pickup_hour_12', 'pickup_hour_13',
       'pickup_hour_14', 'pickup_hour_15', 'pickup_hour_16', 'pickup_hour_17',
```

```
 'pickup_hour_18', 'pickup_hour_19', 'pickup_hour_20', 'pickup_hour_21',
 'pickup_hour_22', 'pickup_hour_23', 'flag_Y'],
dtype='object')
```

2 Exploratory Data Analysis

2.1 Univariate Analysis

In [10]:
```
data = df_dummy
```

In this section, we explored each features in the data. It would help us know what the outliers are and how to clean them for further analysis.

The analysis of each feature was did at least twice, before and after cleaning. The information shown below is based on cleaned data.

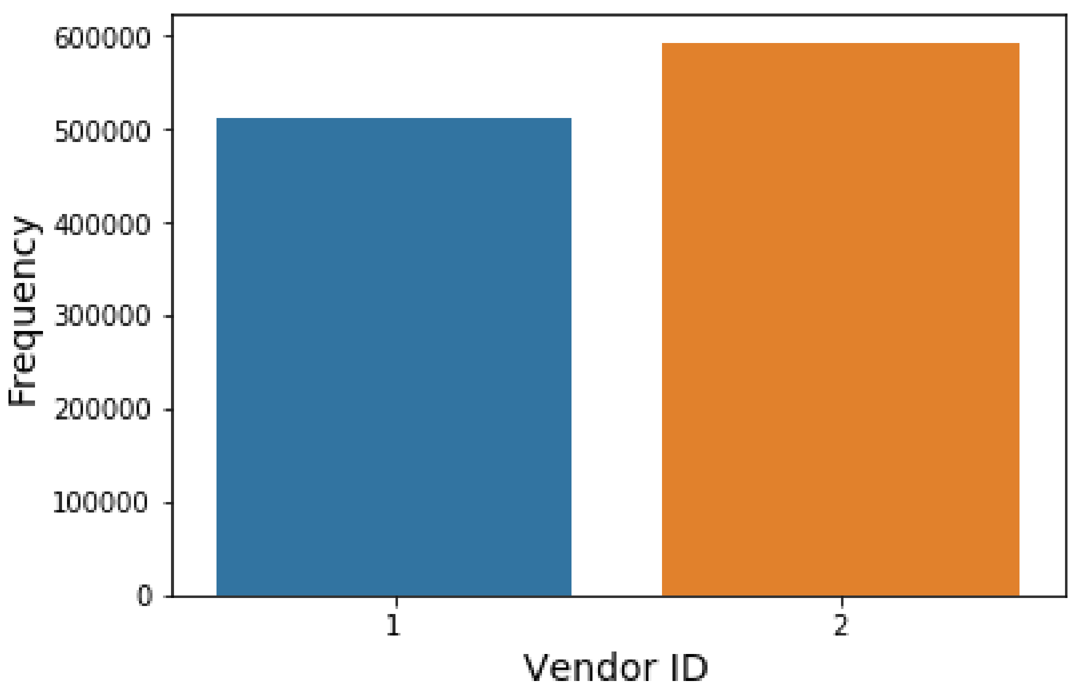
2.1.1 Vendor ID

Vendor ID may indicate different taxi companies.

- **Observations:**

More trips are marked with vendor 2, but its effect on trip duration is unclear.

In [11]:
```
sns.countplot(data['vendor_id'])
plt.xlabel('Vendor ID', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.xticks(plt.xticks()[0], rotation=0)
plt.show()
```



2.1.2 Number of passengers

In New York, a maximum of 4 passengers can ride in traditional cabs, and there are also minivans-like cabs that can accommodate 5 passengers. A child under 7 is allowed to sit on a passenger's lap in the rear seat in addition to the passenger limit. Therefore, in total we can assume that a maximum of 6 passenger can board the new york taxi.

- **Observations:**

The original data showed that there are trips with 0 - 9 passengers. Most trips consist of 1 or 2 passengers.

- **Idea of cleaning data:**

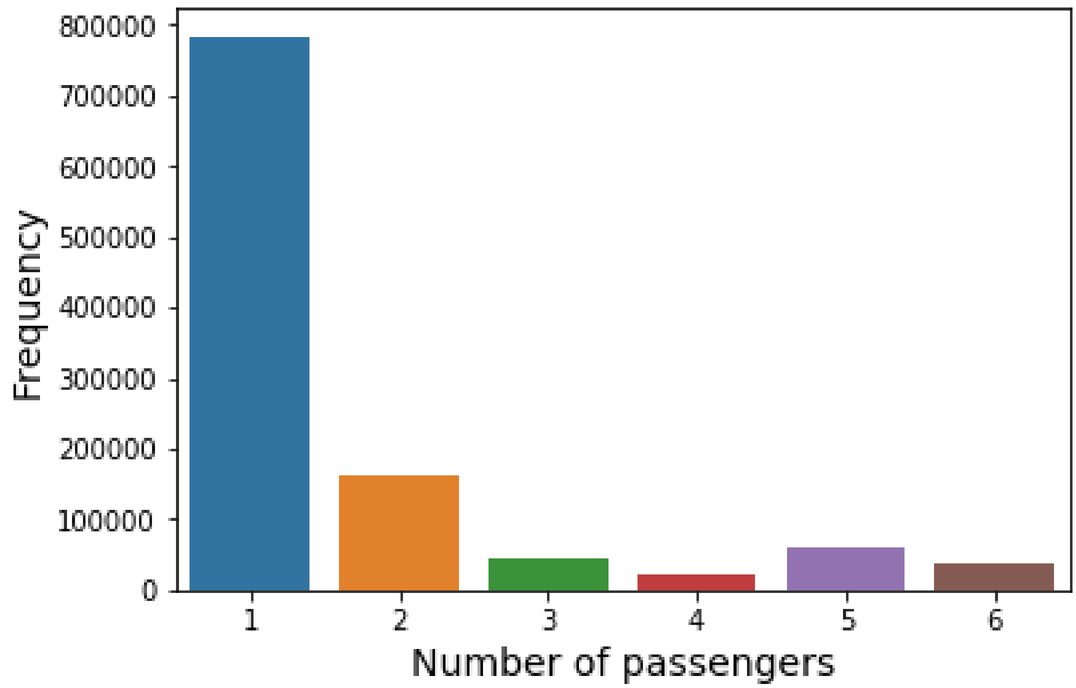
First, the trips without passengers does not count. Or it is because the drivers did not provide the number of passengers. In this case, we have enough samples so we just omitted records with 0 passengers.

Second, there are also trips with 7, 8 or 9 passengers. It is impossible for taxi ride in New York so they are obviously outliers.

```
In [12]: data['passenger_count'].value_counts()
```

```
Out[12]: 1    783591
         2    160096
         5     59638
         3     45179
         6     36768
         4     21212
         Name: passenger_count, dtype: int64
```

```
In [13]: sns.countplot(data['passenger_count'])
         plt.xlabel('Number of passengers', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.xticks(plt.xticks()[0], rotation=0)
         plt.show()
```



2.1.3 Trip distance

- **Observations:**

The distance value of some trips is 0 km. Also, the distance of some trips is over 200 km distance.

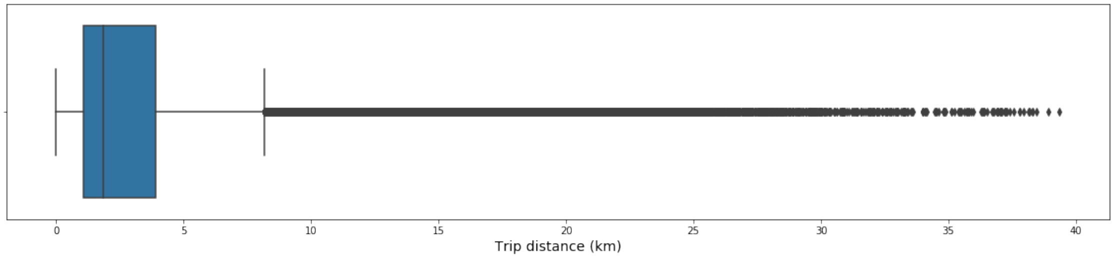
- **Idea of cleaning data:**

A taxi ride with 0 km or over 40 km distance is abnormal.

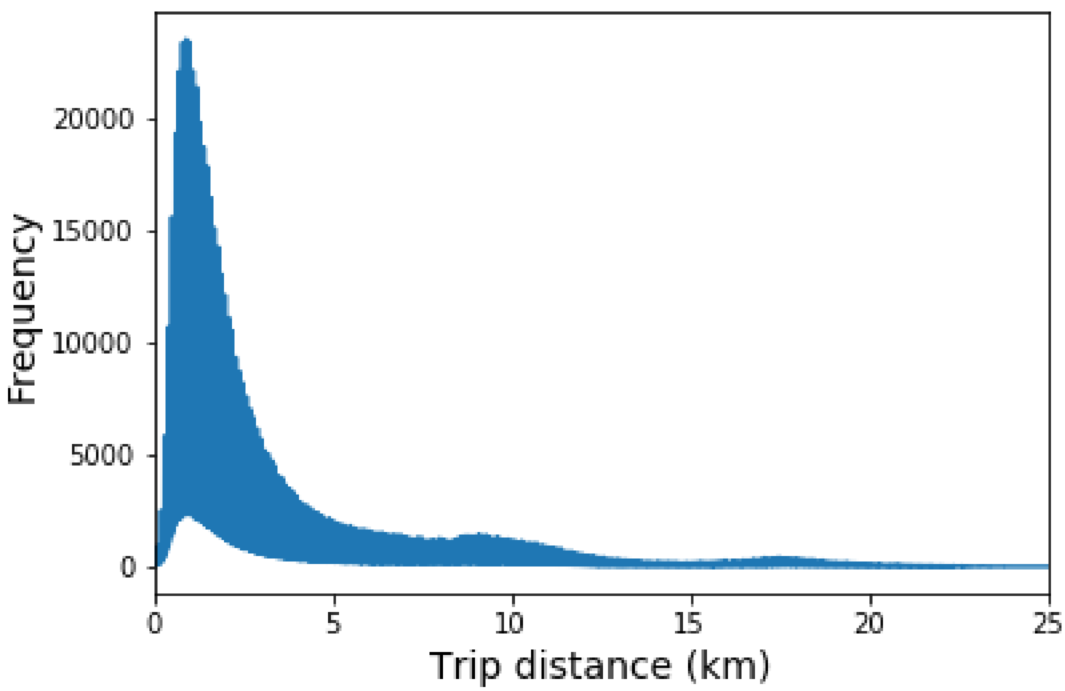
In [14]:
```
data['trip_distance'].describe()
```

Out[14]:
```
count    1.106484e+06
mean     3.422050e+00
std      3.932481e+00
min      1.000000e-02
25%      1.060000e+00
50%      1.840000e+00
75%      3.900000e+00
max      3.934000e+01
Name: trip_distance, dtype: float64
```

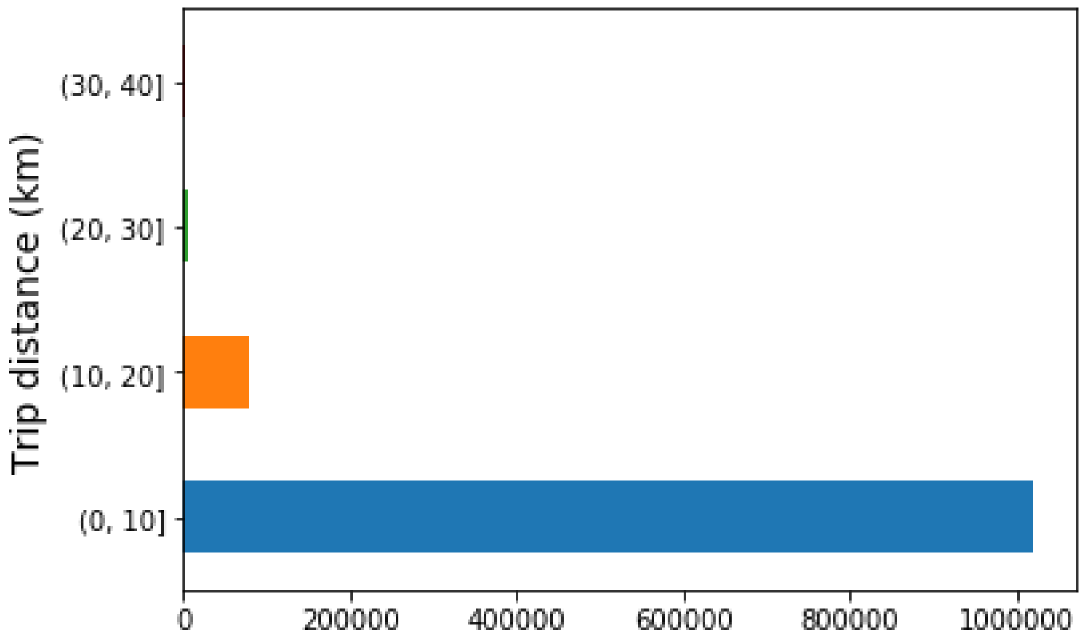
In [15]:
```
plt.figure(figsize = (20,4))
sns.boxplot(data['trip_distance'])
plt.xlabel('Trip distance (km)', fontsize=14)
plt.show()
```




```
In [16]: data['trip_distance'].value_counts().sort_index().plot.line()
         plt.xlabel('Trip distance (km)', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.xticks(plt.xticks()[0], rotation=0)
         plt.xlim(0,25)
         plt.show()
```




```
In [17]: data['trip_distance'].groupby(pd.cut(data['trip_distance'], np.arange(0,50,10))).coun
         plt.ylabel('Trip distance (km)', fontsize=14)
         plt.show()
```



2.1.4 Location

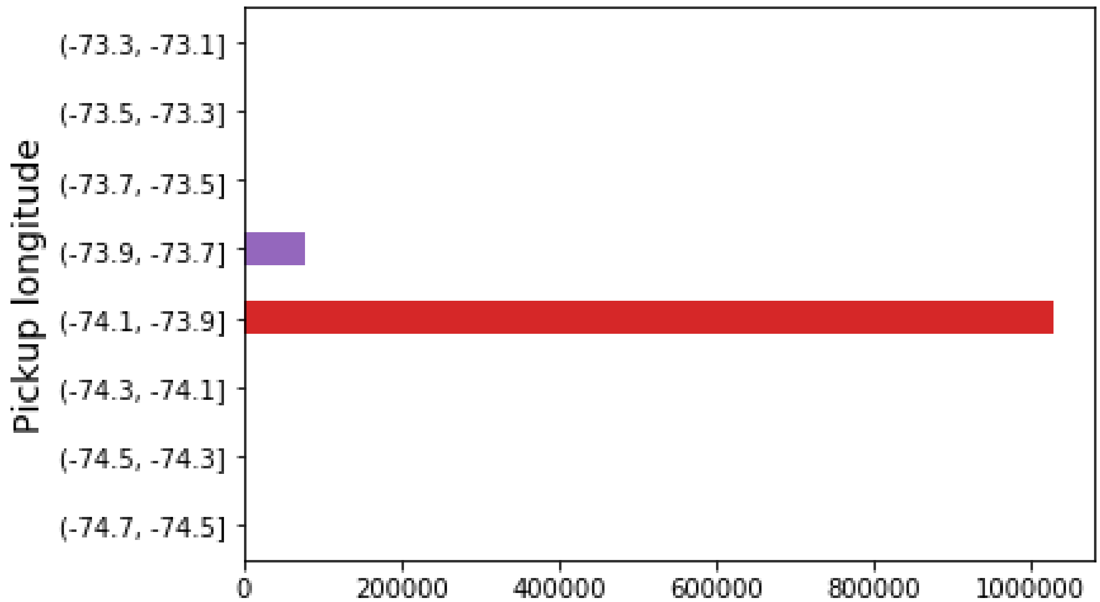
- **Observations:**

Most pickup locations are in the area of [40.5, 40.9] latitude and [-74.1, 73.7] longitude while most dropoff locations are in the area of [40.6, 41] latitude and [-74.05, 73.65] longitude. Also, there are some locations which are out of NYC.

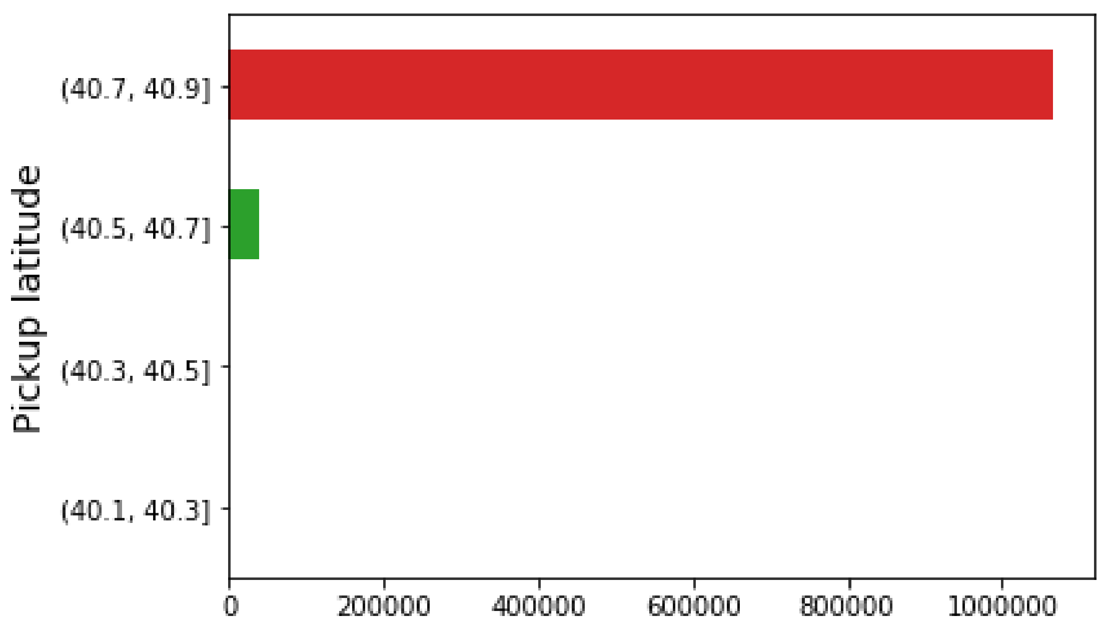
- **Idea of cleaning data:**

The latitude and longitude range of NYC is [40.5, 40.9] and [-74.2, -73.7].

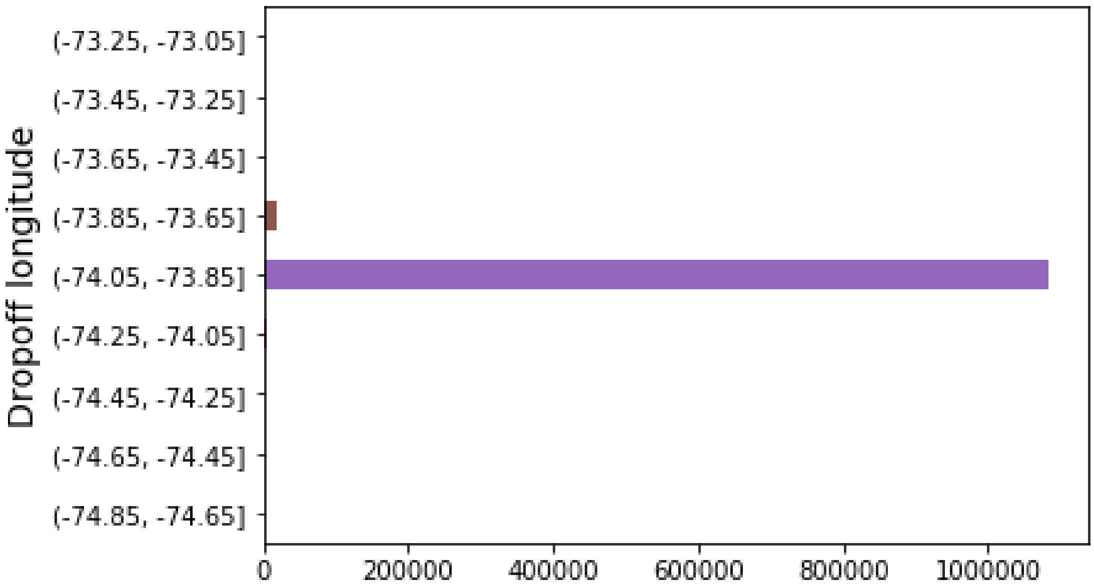
```
In [18]: # Pickup longitude
         data['pickup_longitude'].groupby(pd.cut(data['pickup_longitude'], np.arange(-74.7, -73
         plt.ylabel('Pickup longitude', fontsize=14)
         plt.show()
```



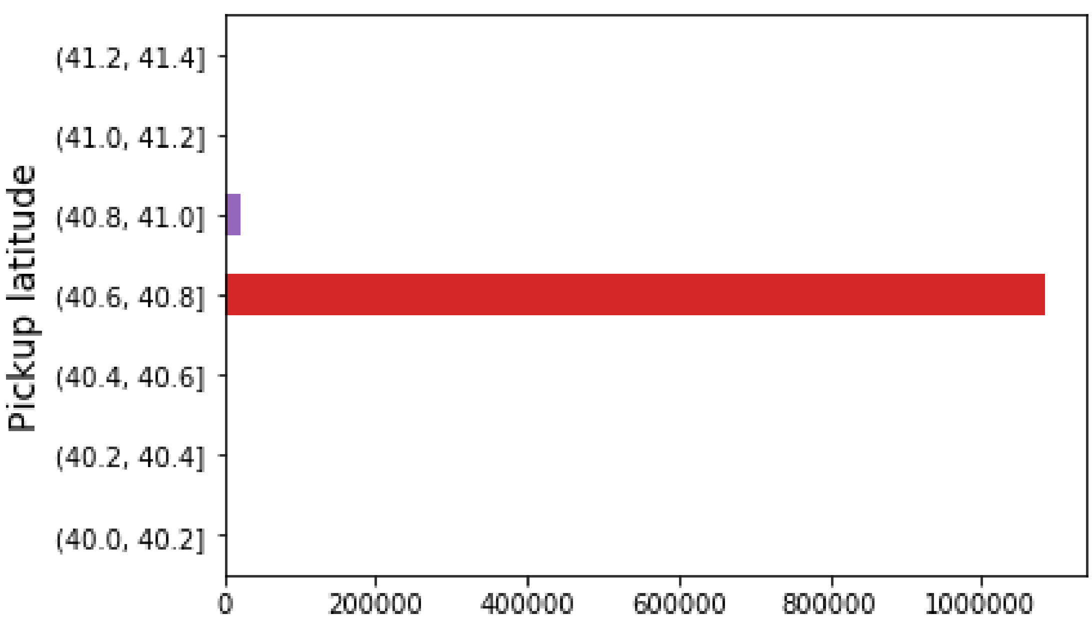
In [19]:
```
# pickup latitude
data['pickup_latitude'].groupby(pd.cut(data['pickup_latitude'], np.arange(40.1, 41.1,
plt.ylabel('Pickup latitude', fontsize=14)
plt.show()
```



```
In [20]: # Dropoff longitude
         data['dropoff_longitude'].groupby(pd.cut(data['dropoff_longitude'], np.arange(-74.85,
         plt.ylabel('Dropoff longitude', fontsize=14)
         plt.show()
```

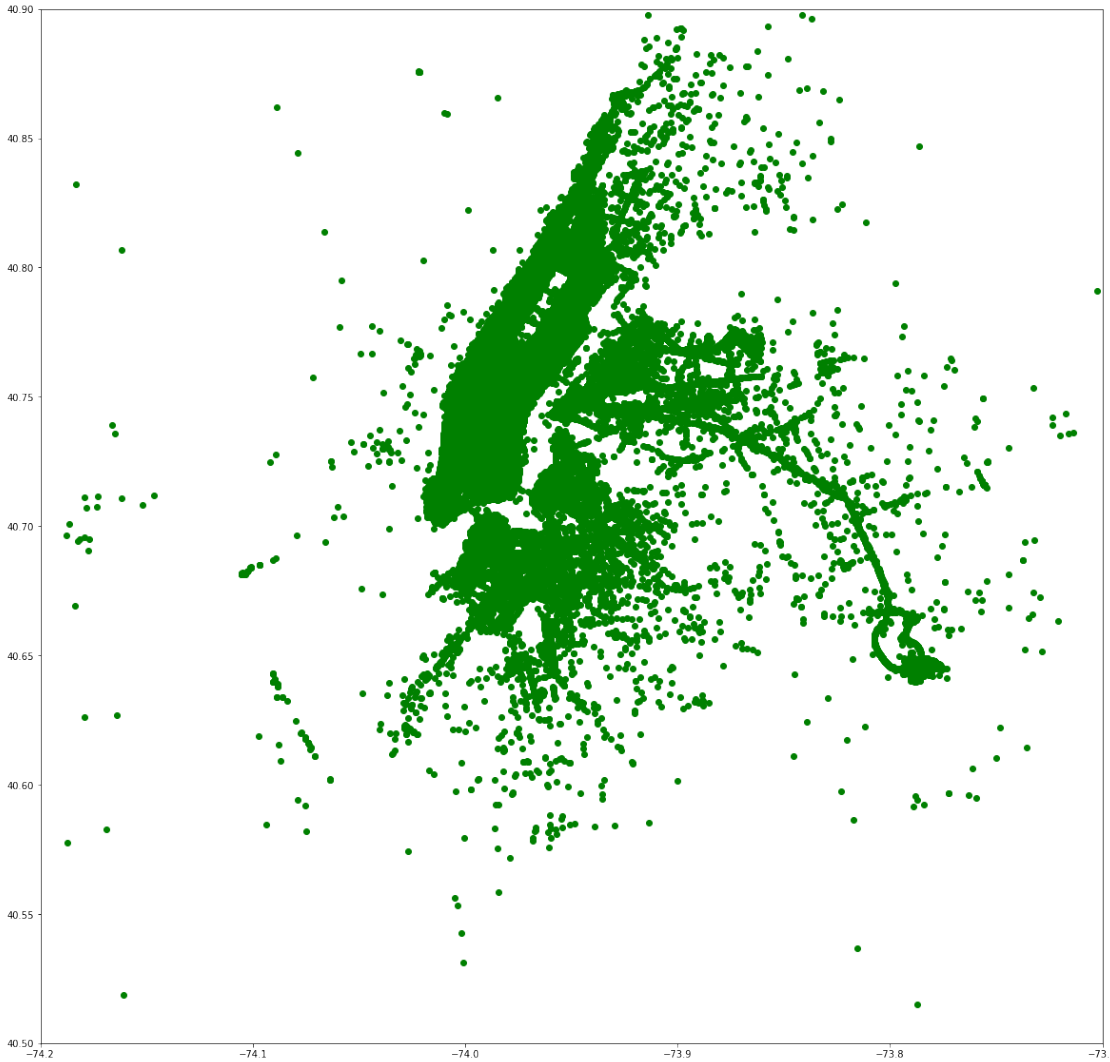



```
In [21]: # pickup longitude
         data['pickup_latitude'].groupby(pd.cut(data['pickup_latitude'], np.arange(40, 41.5, 0
         plt.ylabel('Pickup latitude', fontsize=14)
         plt.show()
```



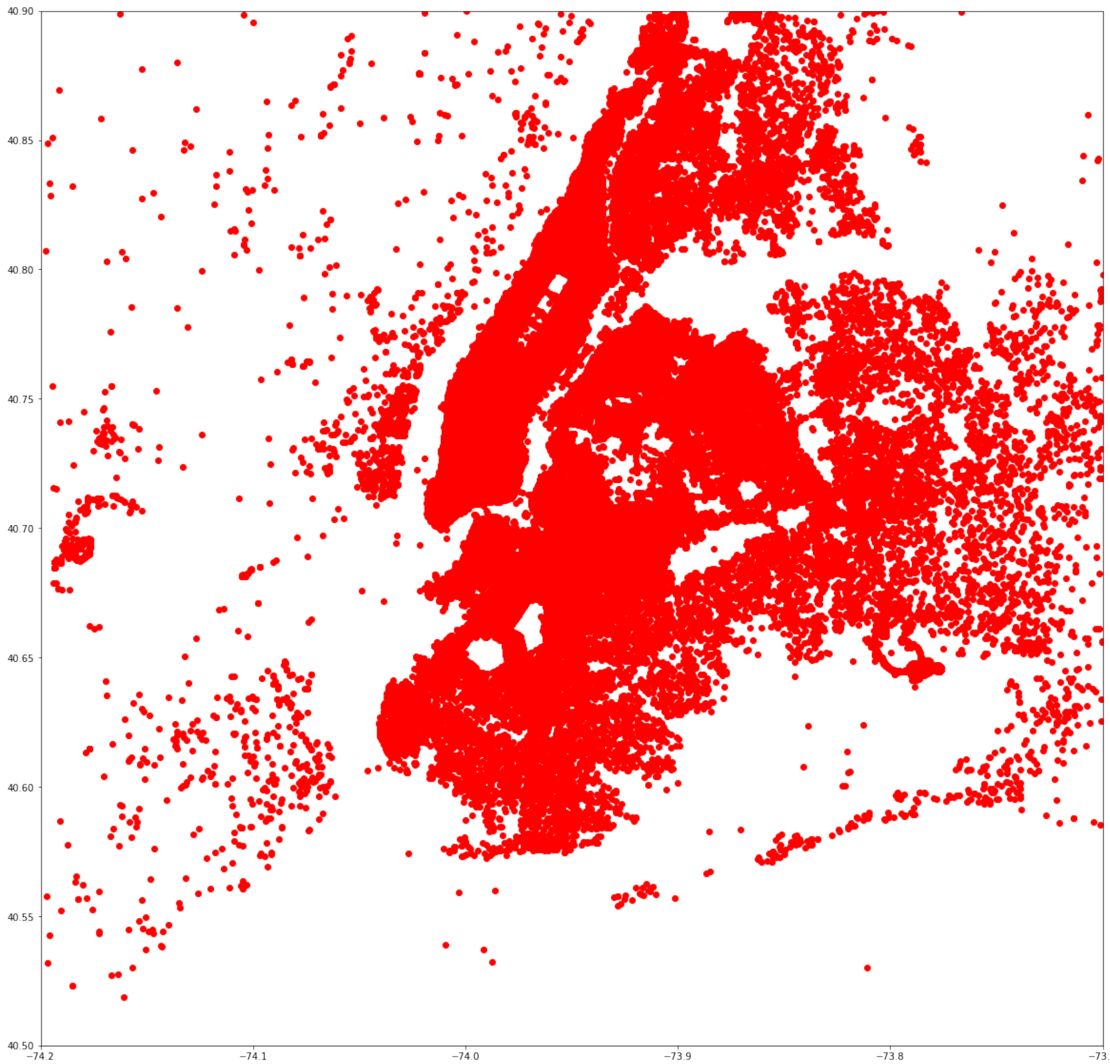
In [22]: # Pickup location

```
plt.figure(figsize=(20,20))
plt.scatter(data['pickup_longitude'], data['pickup_latitude'], c='g')
plt.xlim((-74.2, -73.7))
plt.ylim((40.5, 40.9))
plt.show()
```



In [23]: # Dropoff location

```
plt.figure(figsize=(20,20))
plt.scatter(data['dropoff_longitude'], data['dropoff_latitude'], c='r')
plt.xlim((-74.2, -73.7))
plt.ylim((40.5, 40.9))
plt.show()
```



2.1.5 Travel duration

- **Observations:**

Some trips took only seconds while some traveled over 24 hours.
Most trips were finished within 20 minutes.

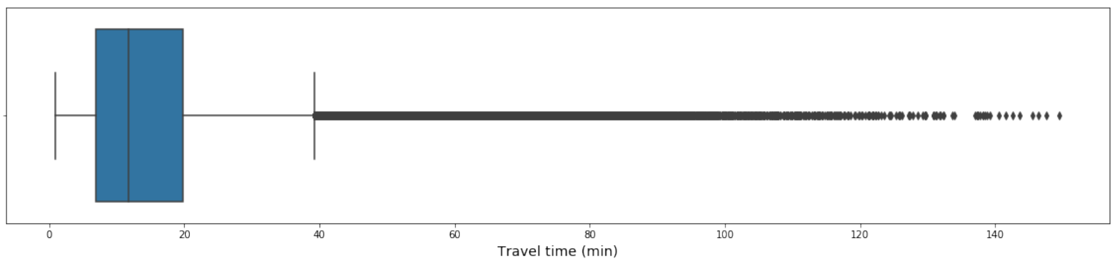
- **Idea of cleaning data:**

Samples with travel time over 23 hr or under 1 min should be cleaned.

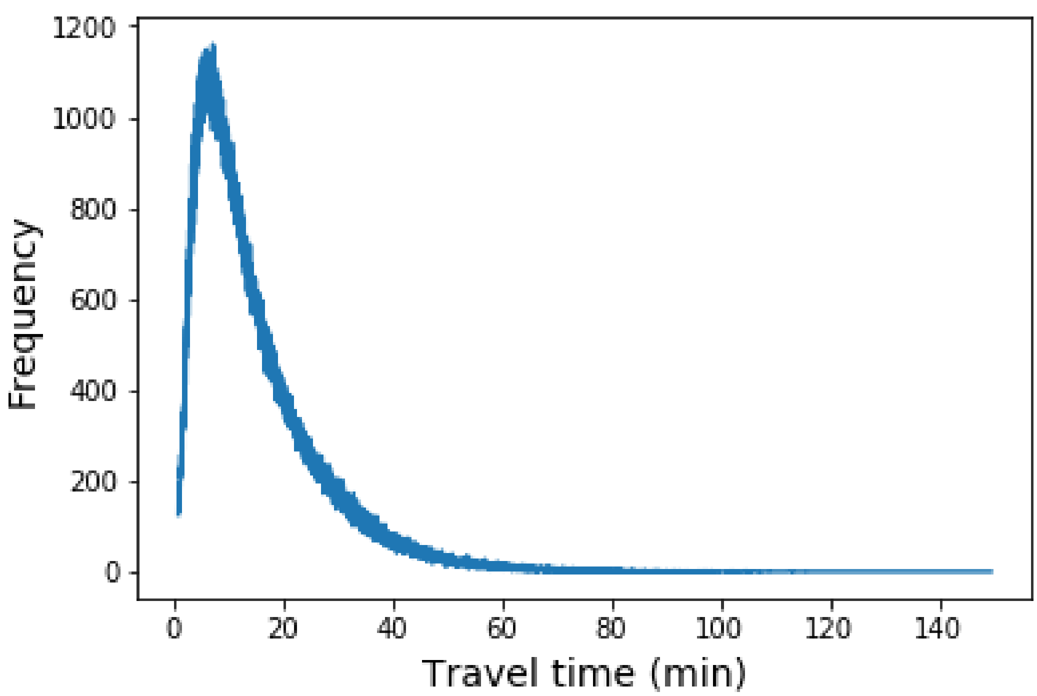
```
In [24]: data['travel_time'].describe()

Out[24]: count    1.106484e+06
         mean     9.053012e+02
```

```
std      6.963099e+02
min      6.000000e+01
25%      4.160000e+02
50%      7.060000e+02
75%      1.193000e+03
max      8.967000e+03
Name: travel_time, dtype: float64
```

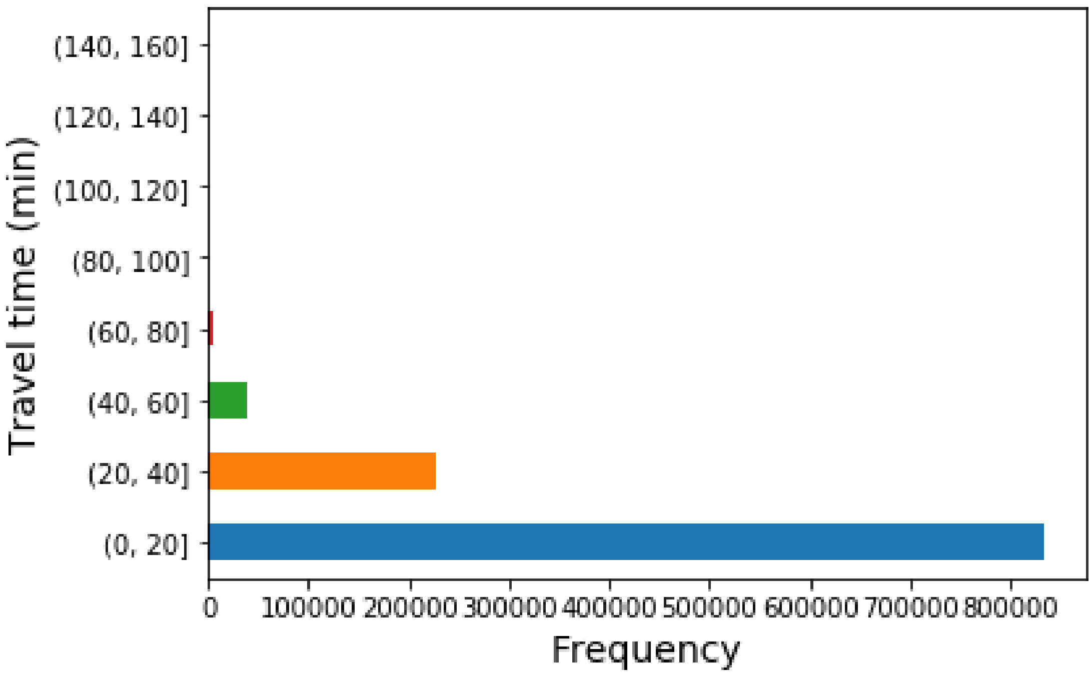
In [25]:
```python
plt.figure(figsize = (20,4))
sns.boxplot(data['travel_time']/60)
plt.xlabel('Travel time (min)', fontsize=14)
plt.show()
```



In [26]:
```python
(data['travel_time']/60).value_counts().sort_index().plot.line()
plt.xlabel('Travel time (min)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()
```



```
In [27]: (data['travel_time']/60).groupby(pd.cut((data['travel_time']/60), np.arange(0,170,20)
         plt.xlabel('Frequency', fontsize=14)
         plt.ylabel('Travel time (min)', fontsize=14)
         plt.show()
```



2.1.6 Speed

- **Observations:**

  Some trips were done at a speed of 0.
  Most records are with speed below 40 km/h, which meet the maximum speed limit of urban area in New York City.
  Trips on highway should also be considered.

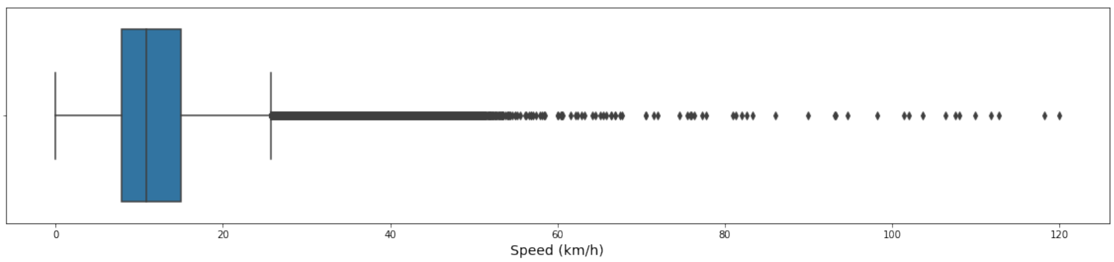
- **Idea of cleaning data:**

  Our analysis should focus on samples with speed between 0 to 120 km/h.

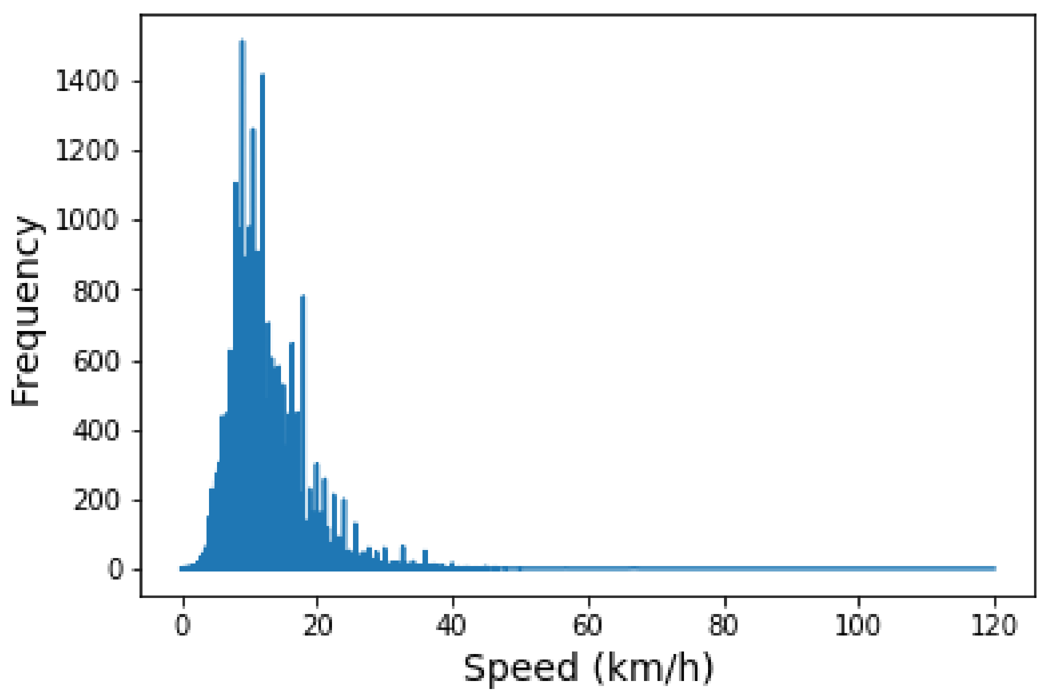
```
In [28]: data['speed'].describe()

Out[28]: count    1.106484e+06
         mean     1.243565e+01
```

```
std      6.684449e+00
min      4.853714e-03
25%      7.929515e+00
50%      1.081545e+01
75%      1.503000e+01
max      1.200000e+02
Name: speed, dtype: float64
```

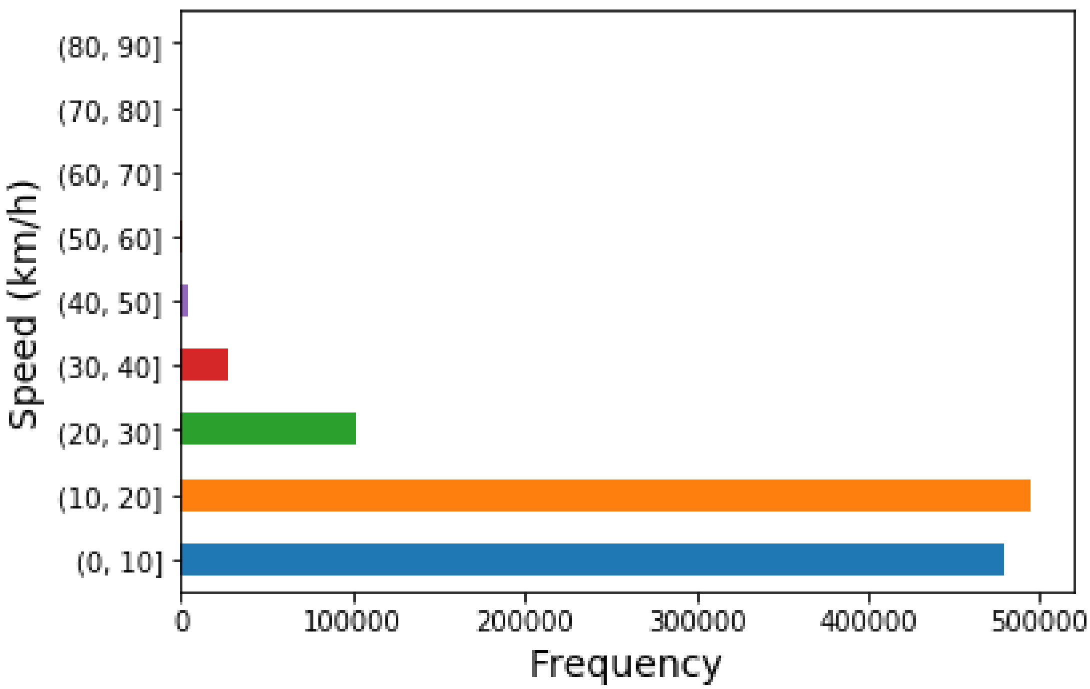
In [29]:
```python
plt.figure(figsize = (20,4))
sns.boxplot(data['speed'])
plt.xlabel('Speed (km/h)', fontsize=14)
plt.show()
```



In [30]:
```python
data['speed'].value_counts().sort_index().plot.line()
plt.xlabel('Speed (km/h)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()
```



```
In [31]: data['speed'].groupby(pd.cut(data['speed'], np.arange(0,100,10))).count().plot(kind =
         plt.xlabel('Frequency', fontsize=14)
         plt.ylabel('Speed (km/h)', fontsize=14)
         plt.show()
```



2.1.7 Weekday

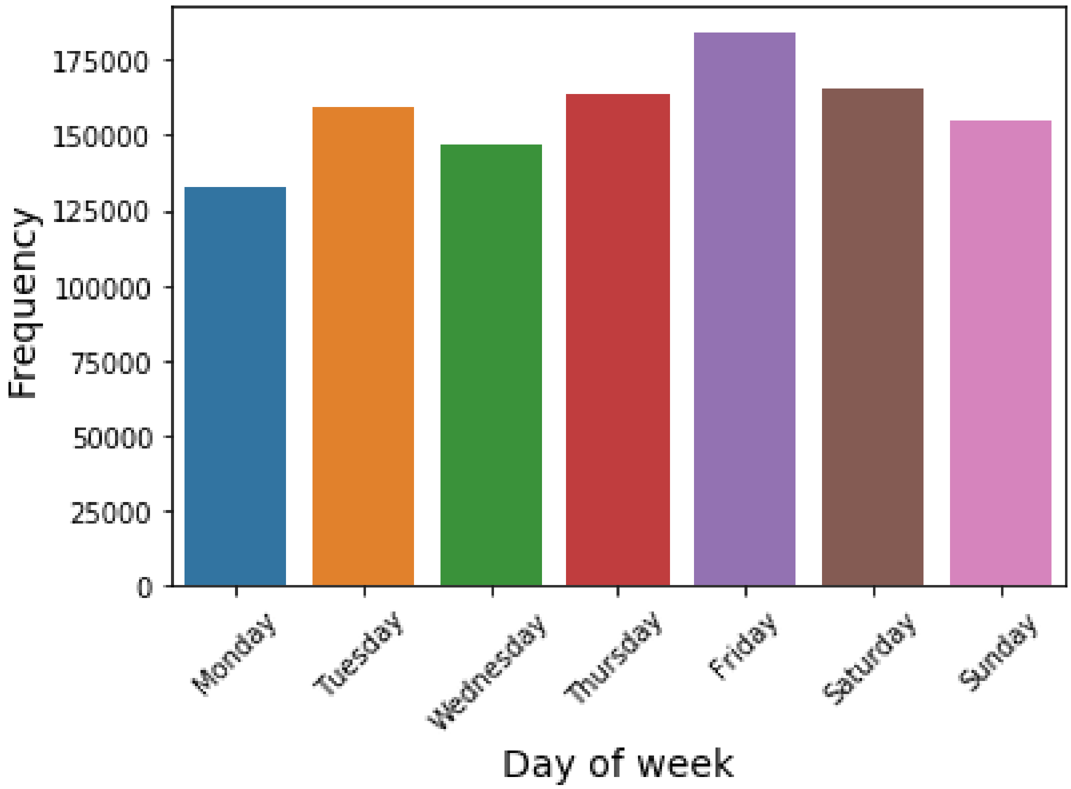
- **Observations:**

Here we can see the taxi pickups start increasing from Monday till Friday, then start declining from Saturday till Monday.

```
In [32]: data['weekday'].value_counts()
```

```
Out[32]: Friday       183850
         Saturday     165085
         Thursday     163513
         Tuesday      159200
         Sunday       155195
         Wednesday    147221
         Monday       132420
         Name: weekday, dtype: int64
```

```
In [33]: sns.countplot(data['day_of_week'])
         plt.xlabel('Day of week', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.xticks(plt.xticks()[0],
                    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunda
                    rotation=45)
         plt.show()
```



2.1.8 Pickup hour

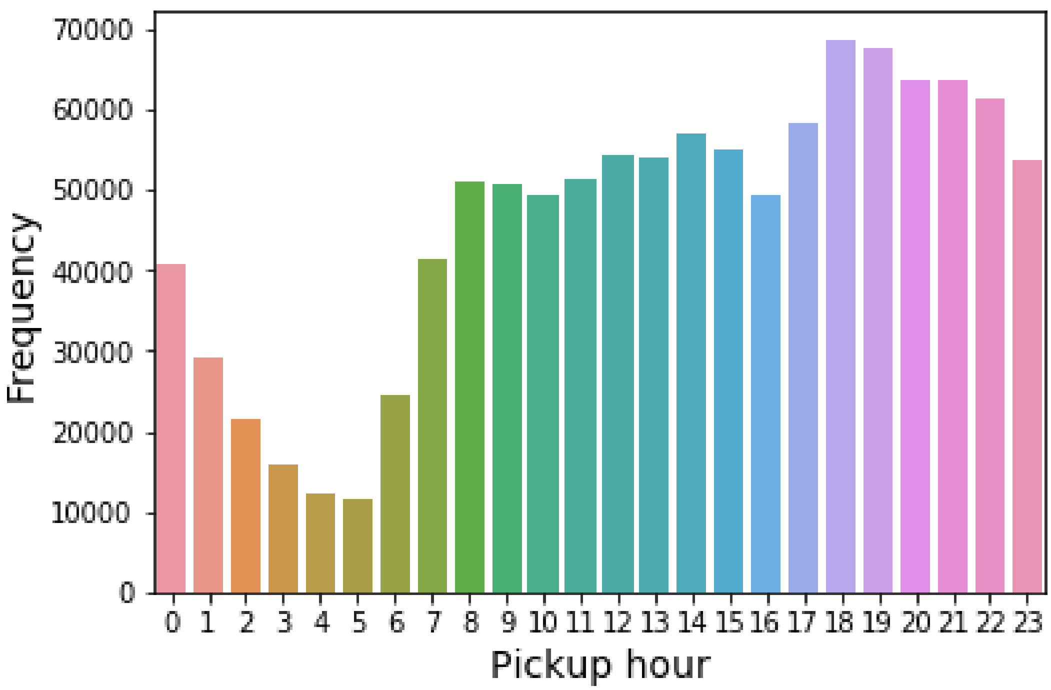
- **Observations:**

It accords with common sense that taxi pickups increase from 6am and the rush hour starts around 6pm.

```
In [34]: data['pickup_hour'].value_counts()

Out[34]: 18    68715
         19    67746
         20    63779
         21    63615
         22    61399
```

```
17    58213
14    56882
15    55156
12    54286
13    54089
23    53570
11    51488
8     50882
9     50764
10    49433
16    49395
7     41363
0     40900
1     29158
6     24428
2     21480
3     15779
4     12320
5     11644
Name: pickup_hour, dtype: int64
```

In [35]:
```
sns.countplot(data['pickup_hour'])
plt.xlabel('Pickup hour', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.show()
```



2.1.9 Month

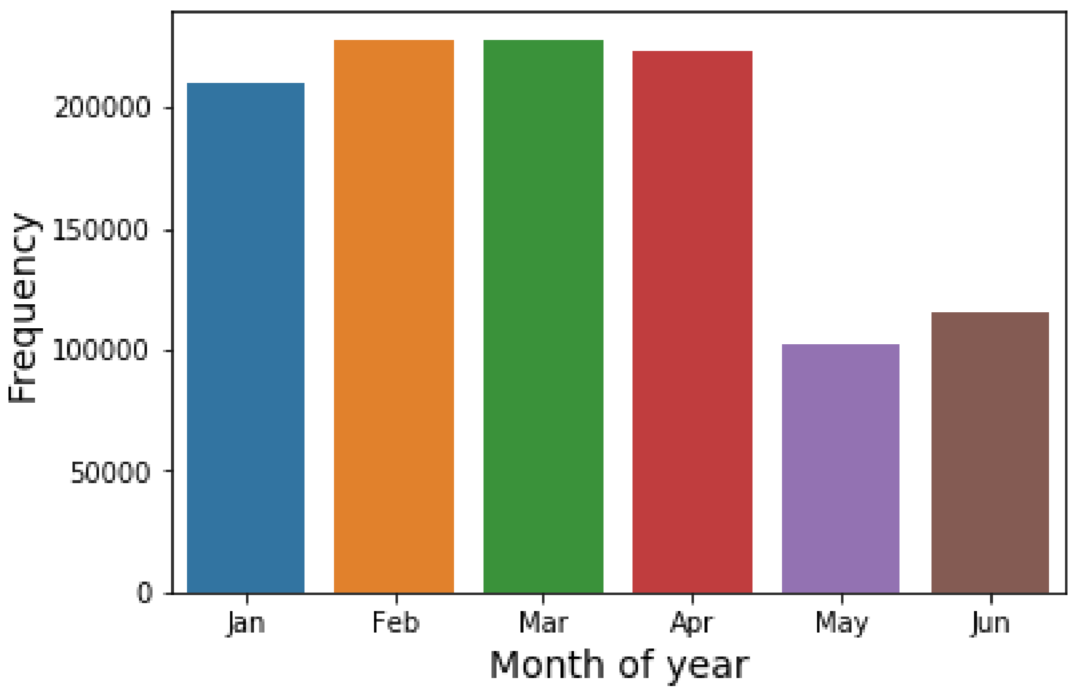
- **Observations:**

Did not observe obvious trend here.

```
In [36]: data['month_of_year'].value_counts()
```

```
Out[36]: 2    227956
         3    227650
         4    223362
         1    210363
         6    115357
         5    101796
         Name: month_of_year, dtype: int64
```

```
In [37]: sns.countplot(data['month_of_year'])
         plt.xlabel('Month of year', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.xticks(plt.xticks()[0],
                    ['Jan','Feb', 'Mar', 'Apr', 'May', 'Jun'],
                    rotation=0)
         plt.show()
```



2.1.10 Fare

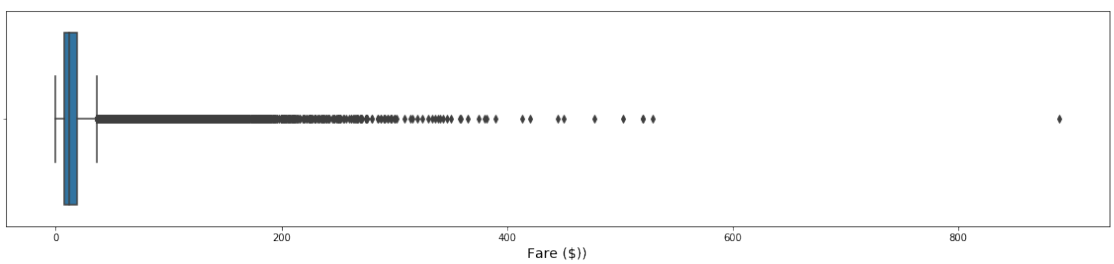
- **Idea of cleaning data:**

Records with negative fare were cleaned.

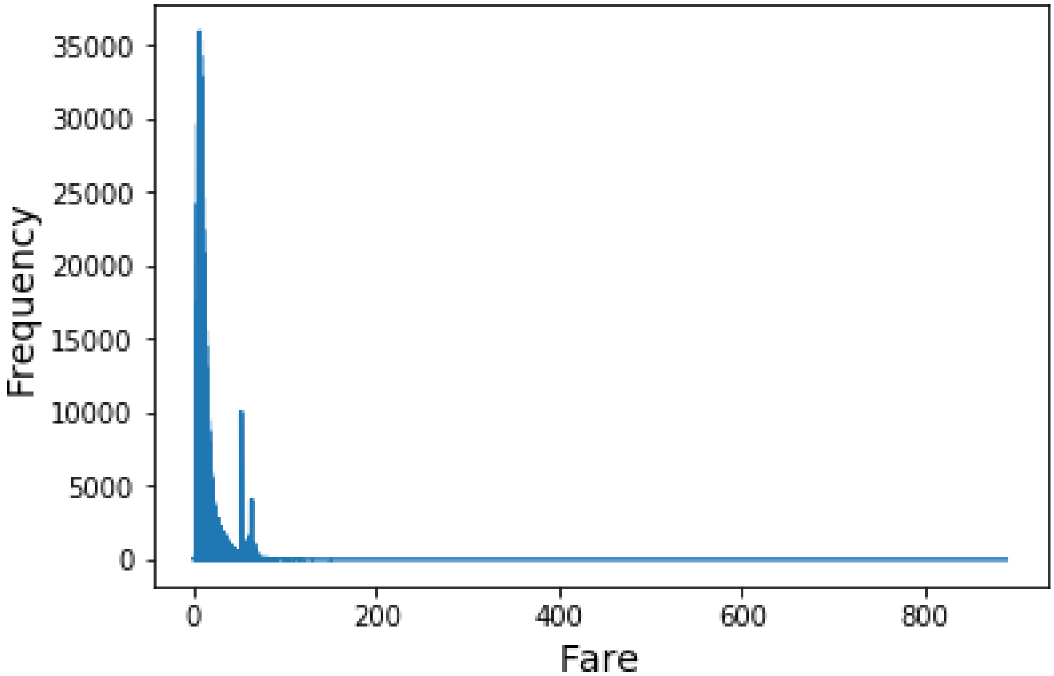
```
In [78]: data['fare+tip'].describe()
```

```
Out[78]: count    1.547970e+06
         mean     1.637340e+01
         std      1.409114e+01
         min      1.000000e-02
         25%      7.550000e+00
         50%      1.150000e+01
         75%      1.900000e+01
         max      8.893500e+02
         Name: fare+tip, dtype: float64
```

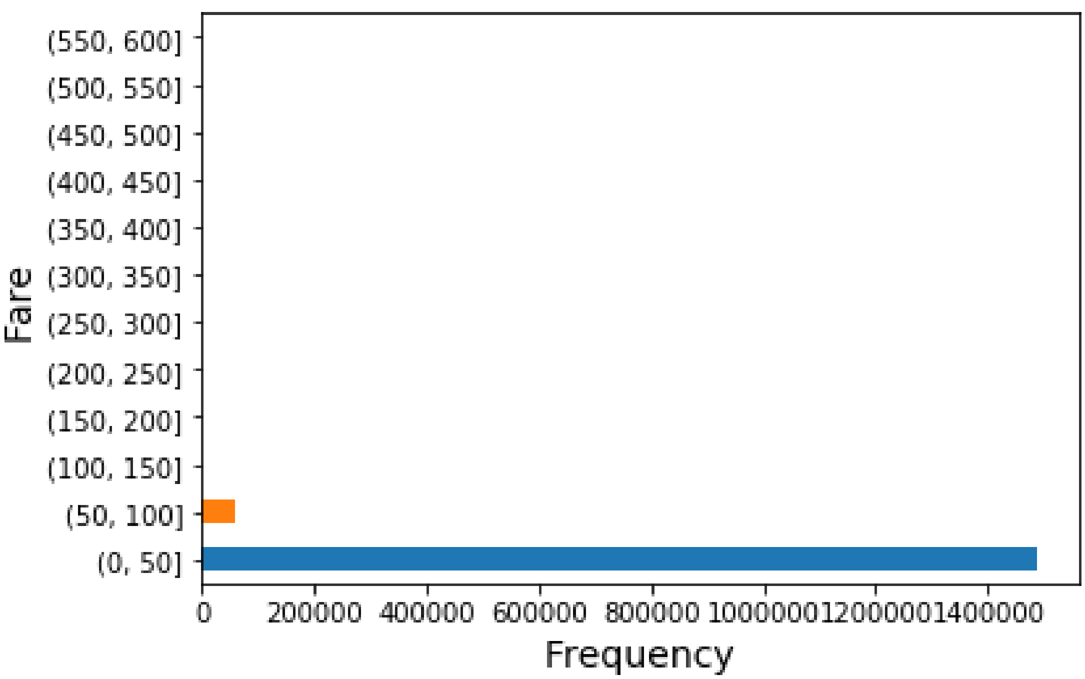
```
In [79]: plt.figure(figsize = (20,4))
         sns.boxplot(data['fare+tip'])
         plt.xlabel('Fare ($))', fontsize=14)
         plt.show()
```




```
In [81]: data['fare+tip'].value_counts().sort_index().plot.line()
         plt.xlabel('Fare', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.show()
```




```
In [87]: data['fare+tip'].groupby(pd.cut(data['fare+tip'], np.arange(0,650,50))).count().plot(l
         plt.xlabel('Frequency', fontsize=14)
         plt.ylabel('Fare', fontsize=14)
         plt.show()
```



2.1.11 Temperature

- **Observations:**

The temperature was in a range of 0 to 80 F.

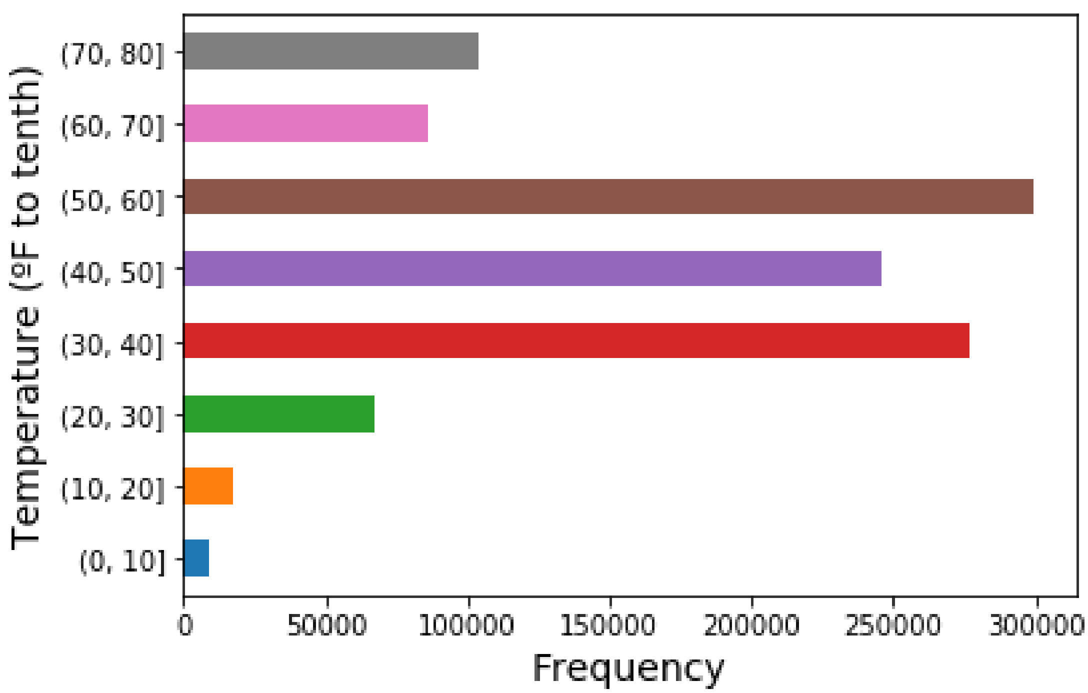
- **Idea of cleaning data:**

No outliers were seen here. We only needed to clean the missed records.

In [38]:
```
data['temp'].describe()
```

Out[38]:
```
count    1.106484e+06
mean     4.746541e+01
std      1.447619e+01
min      6.900000e+00
25%      3.740000e+01
50%      4.730000e+01
75%      5.510000e+01
max      7.960000e+01
Name: temp, dtype: float64
```

In [39]:
```
data['temp'].groupby(pd.cut(data['temp'], np.arange(0,90,10))).count().plot(kind = 'ba
plt.xlabel('Frequency', fontsize=14)
plt.ylabel('Temperature (žF to tenth)', fontsize=14)
plt.show()
```



2.1.12 Visibility

- **Observations:**

Most trips were done with a visibility of between 0.5 to 1 miles.

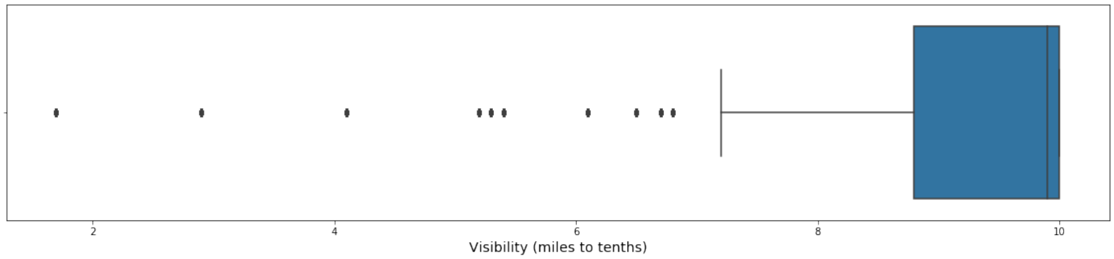
- **Idea of cleaning data:**

Missing data should be cleaned.

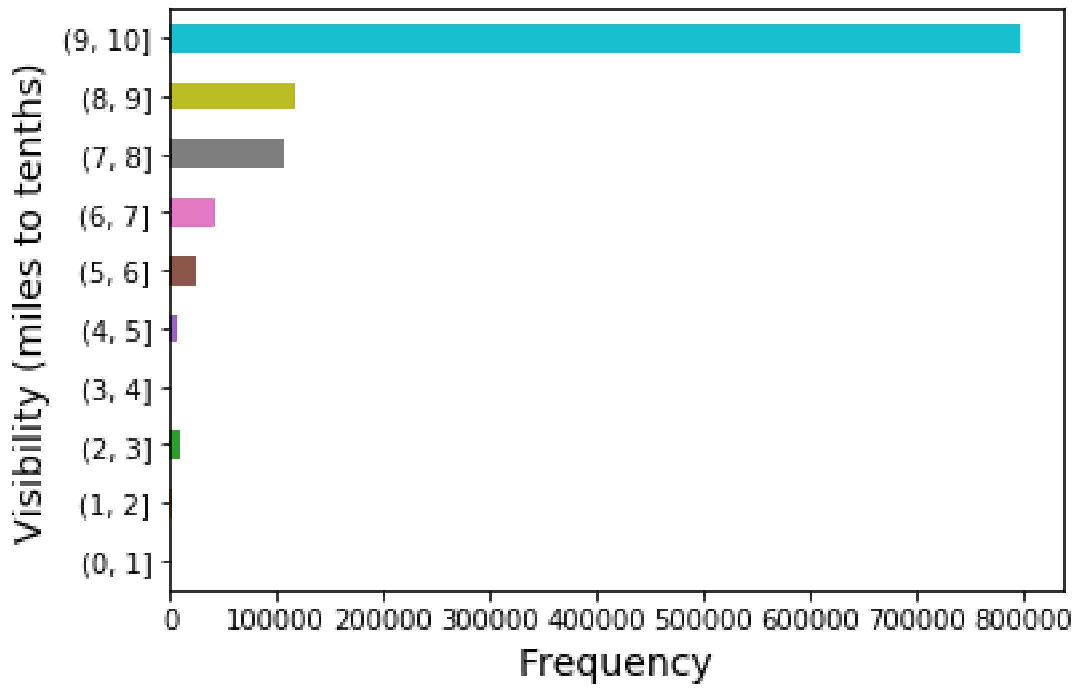
```
In [40]: data['visib'].describe()
```

```
Out[40]: count    1.106484e+06
         mean     9.155323e+00
         std      1.364160e+00
         min      1.700000e+00
         25%      8.800000e+00
         50%      9.900000e+00
         75%      1.000000e+01
         max      1.000000e+01
         Name: visib, dtype: float64
```

```
In [41]: plt.figure(figsize = (20,4))
         sns.boxplot(data['visib'])
         plt.xlabel('Visibility (miles to tenths)', fontsize=14)
         plt.show()
```




```
In [42]: data['visib'].groupby(pd.cut(data['visib'], np.arange(0,11,1))).count().plot(kind = 'l
         plt.xlabel('Frequency', fontsize=14)
         plt.ylabel('Visibility (miles to tenths)', fontsize=14)
         plt.show()
```



2.1.13 Wind speed

- **Obseravtions:**

No outlier was noticed.

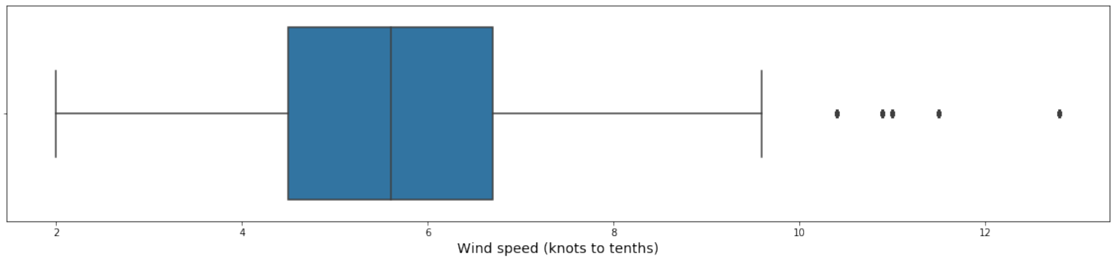
- **Idea of cleaning data:**

The original type of wind speed is string, so we converted it to float type.

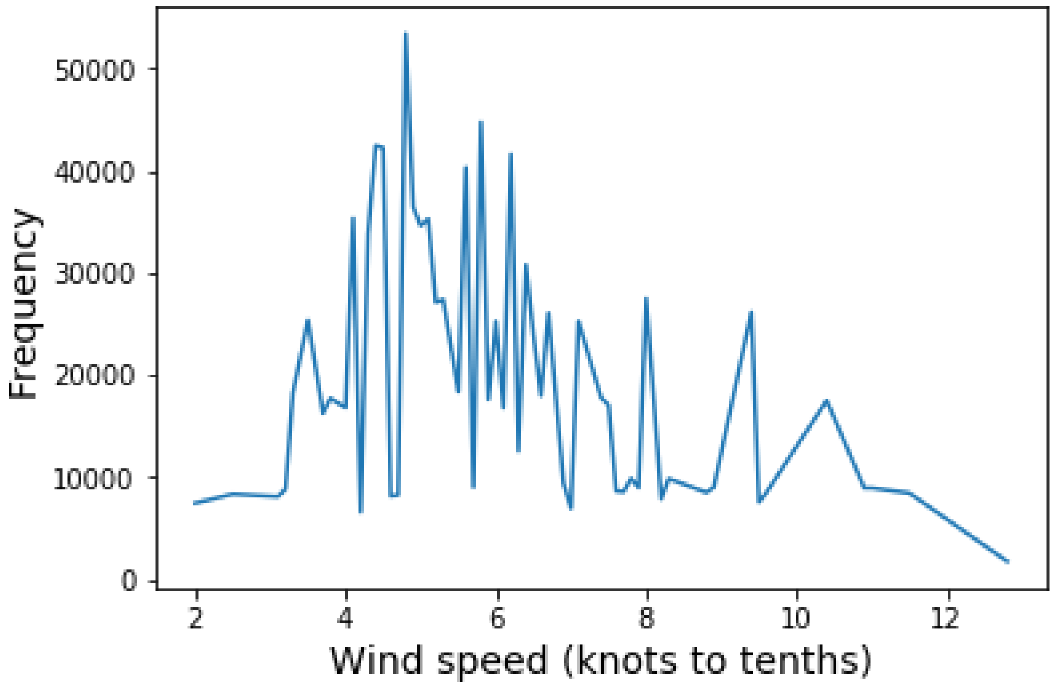
In [43]:
```
data['wdsp'].describe()
```

Out[43]:
```
count    1.106484e+06
mean     5.881155e+00
std      1.854835e+00
min      2.000000e+00
25%      4.500000e+00
50%      5.600000e+00
75%      6.700000e+00
max      1.280000e+01
Name: wdsp, dtype: float64
```

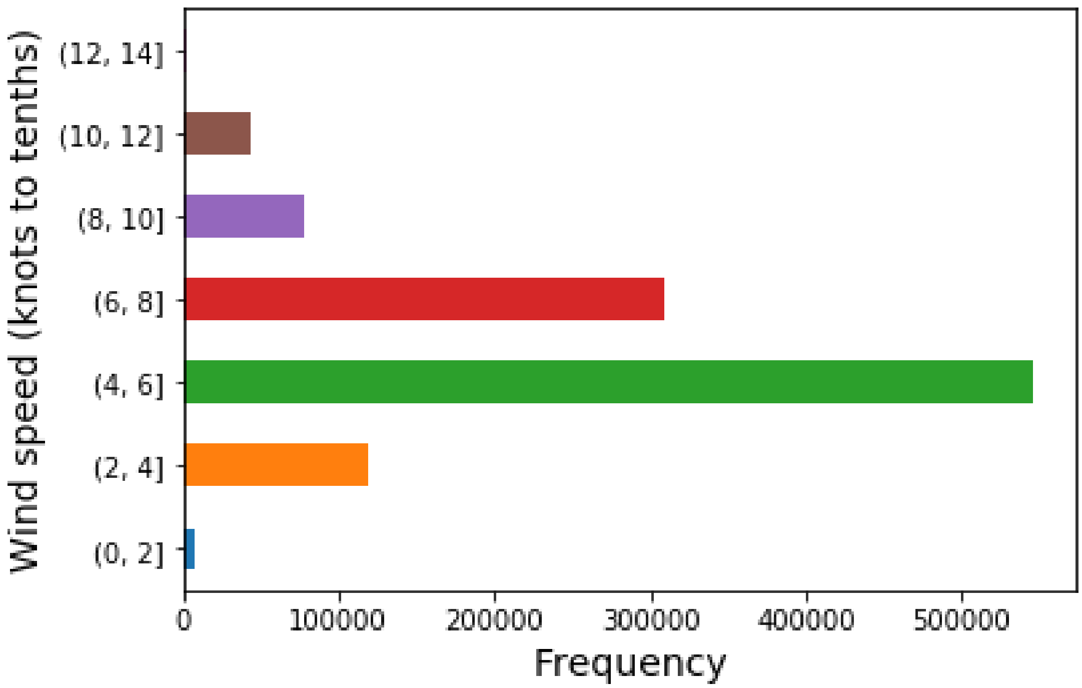
In [44]:
```
plt.figure(figsize = (20,4))
sns.boxplot(data['wdsp'])
plt.xlabel('Wind speed (knots to tenths)', fontsize=14)
plt.show()
```




```
In [45]: data['wdsp'].value_counts().sort_index().plot.line()
         plt.xlabel('Wind speed (knots to tenths)', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.show()
```




```
In [46]: data['wdsp'].groupby(pd.cut(data['wdsp'], np.arange(0,16,2))).count().plot(kind = 'ba
         plt.xlabel('Frequency', fontsize=14)
         plt.ylabel('Wind speed (knots to tenths)', fontsize=14)
         plt.show()
```



2.1.14 Wind gust

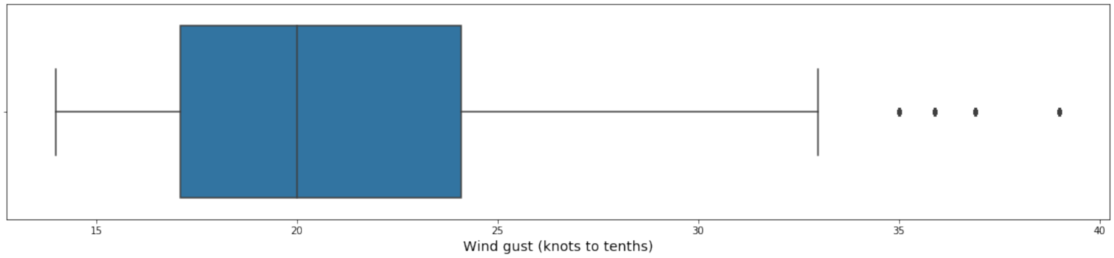
- **Idea of cleaning data:**

Records with missing value were cleaned.

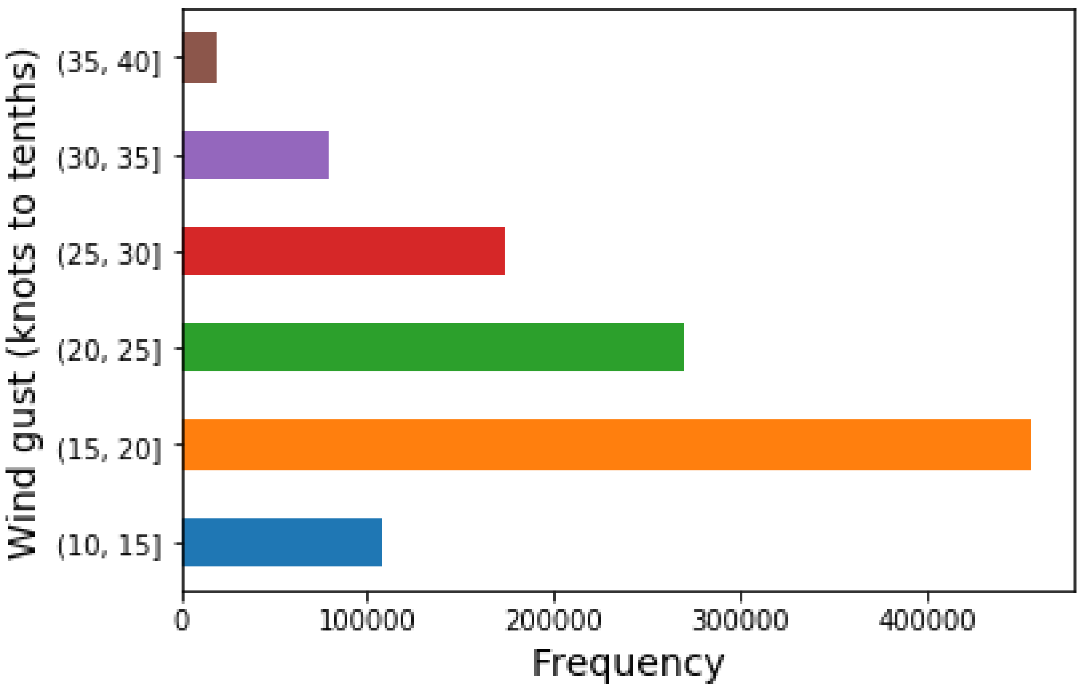
```
In [47]: data['gust'].describe()
```

```
Out[47]: count    1.106484e+06
         mean     2.150665e+01
         std      5.496530e+00
         min      1.400000e+01
         25%      1.710000e+01
         50%      2.000000e+01
         75%      2.410000e+01
         max      3.900000e+01
         Name: gust, dtype: float64
```

```
In [48]: plt.figure(figsize = (20,4))
         sns.boxplot(data['gust'])
         plt.xlabel('Wind gust (knots to tenths)', fontsize=14)
         plt.show()
```




```
In [49]: data['gust'].groupby(pd.cut(data['gust'], np.arange(10,45,5))).count().plot(kind = 'ba
         plt.xlabel('Frequency', fontsize=14)
         plt.ylabel('Wind gust (knots to tenths)', fontsize=14)
         plt.show()
```



2.1.15 Total precipitation

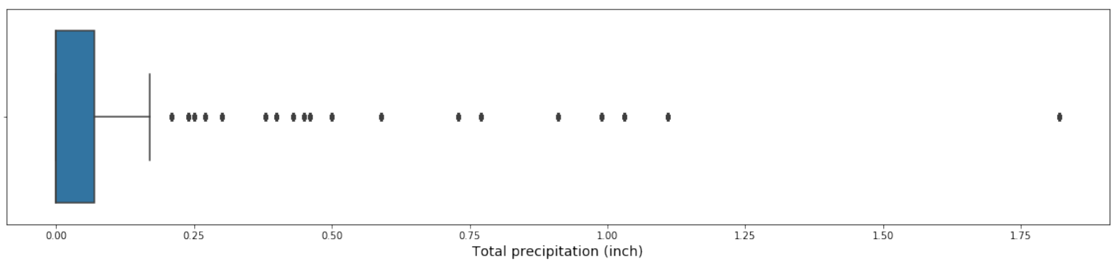
- **Ideas of cleaning data:**

There are so many missing data. Also, for example, a station may only report a 6-hour amount for the period during which rain fell, which may cause the inaccurate prediction. So we might not use precipitation as a feature.

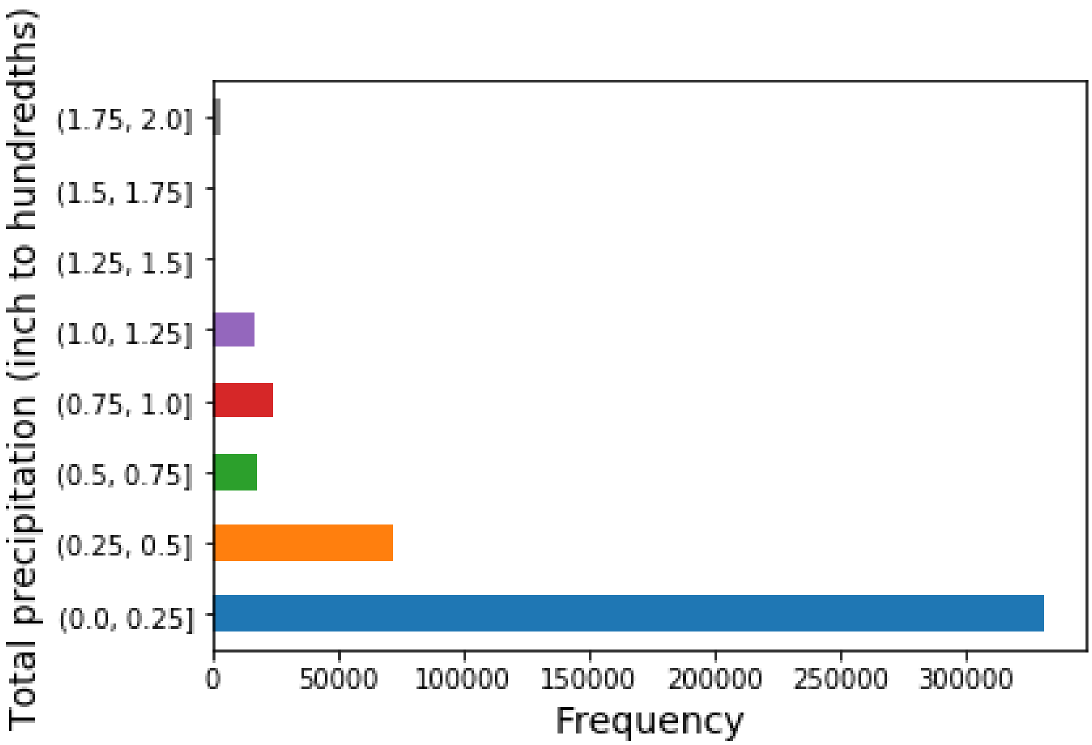
```
In [50]: data['prcp'].describe()
```

```
Out[50]: count    1.106484e+06
         mean     1.031113e-01
         std      2.383314e-01
         min      0.000000e+00
         25%      0.000000e+00
         50%      0.000000e+00
         75%      7.000000e-02
         max      1.820000e+00
         Name: prcp, dtype: float64
```

```
In [51]: plt.figure(figsize = (20,4))
         sns.boxplot(data['prcp'])
         plt.xlabel('Total precipitation (inch to hundredths)', fontsize=14)
         plt.show()
```




```
In [52]: data['prcp'].groupby(pd.cut(data['prcp'], np.arange(0,2.25,0.25))).count().plot(kind =
         plt.xlabel('Frequency', fontsize=14)
         plt.ylabel('Total precipitation (inch to hundredths)', fontsize=14)
         plt.show()
```



2.1.16 Snow depth

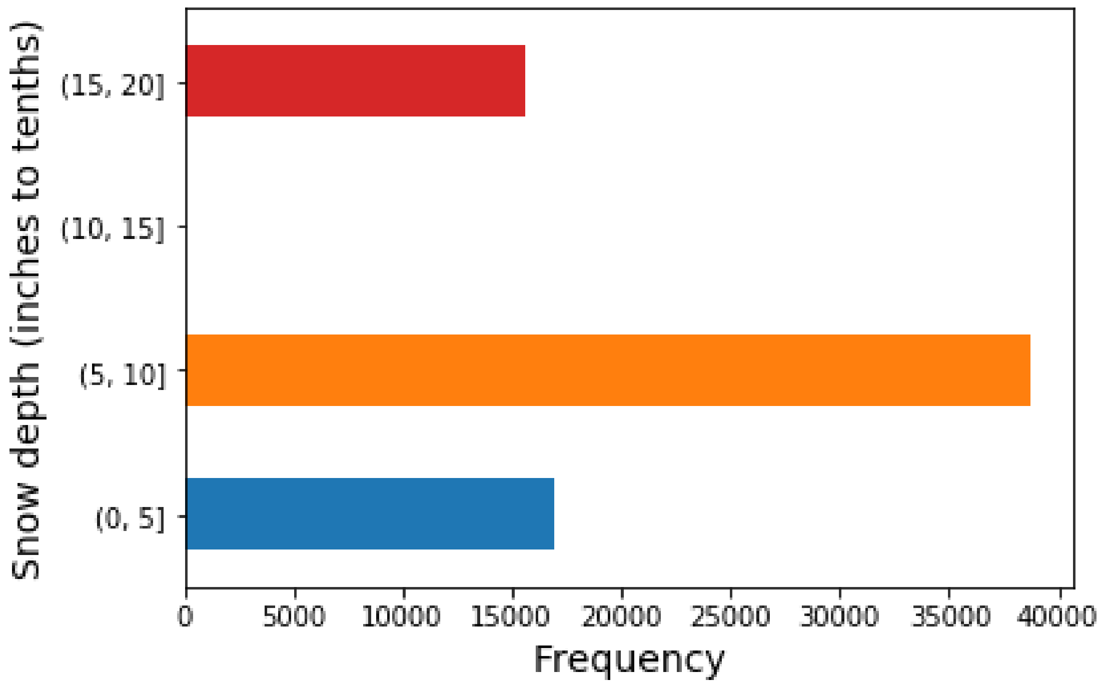
- **Idea of cleaning data:**

Missing data were labelled 999.9. Most stations do not report '0' ondays with no snow on the ground. Therefore, '999.9' will often appear on these days. We thought about if cleaning those records or replacing 999.9 with 0.

```
In [53]: data['sndp'].describe()
```

```
Out[53]: count    1.106484e+06
         mean     5.210219e-01
         std      2.441617e+00
         min      0.000000e+00
         25%      0.000000e+00
         50%      0.000000e+00
         75%      0.000000e+00
         max      1.890000e+01
         Name: sndp, dtype: float64
```

```
In [54]: data['sndp'].groupby(pd.cut(data['sndp'], np.arange(0,25,5))).count().plot(kind = 'ba
         plt.xlabel('Frequency', fontsize=14)
         plt.ylabel('Snow depth (inches to tenths)', fontsize=14)
         plt.show()
```

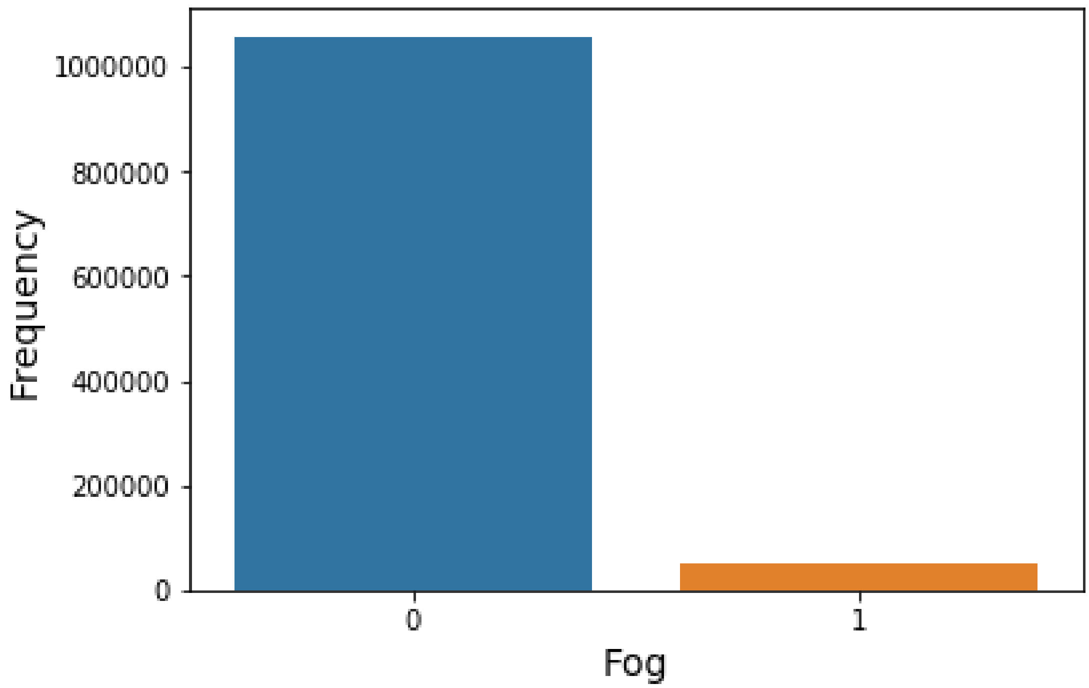


2.1.17 Fog

- **Observations:**

Most of days in the first half of 2016 is no fog.
It is worth noting that 0 in fog as well as rain drizzle, snow ice pellets, hail, and thunder can also mean no reported.

```
In [55]: sns.countplot(data['fog'])
         plt.xlabel('Fog', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.xticks(plt.xticks()[0], rotation=0)
         plt.show()
```

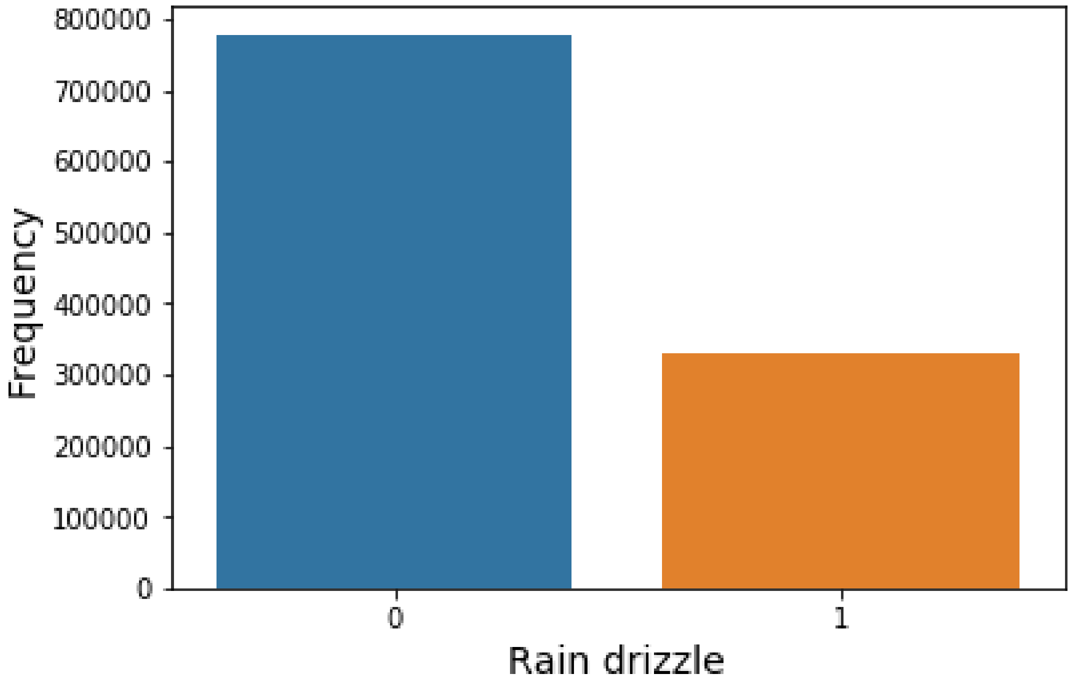


2.1.18 Rain drizzle

- **Observations:**

The ratio of the number of the trips with or without rain drizzle is about 2:5.

```
In [56]: sns.countplot(data['rain_drizzle'])
         plt.xlabel('Rain drizzle', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.xticks(plt.xticks()[0], rotation=0)
         plt.show()
```

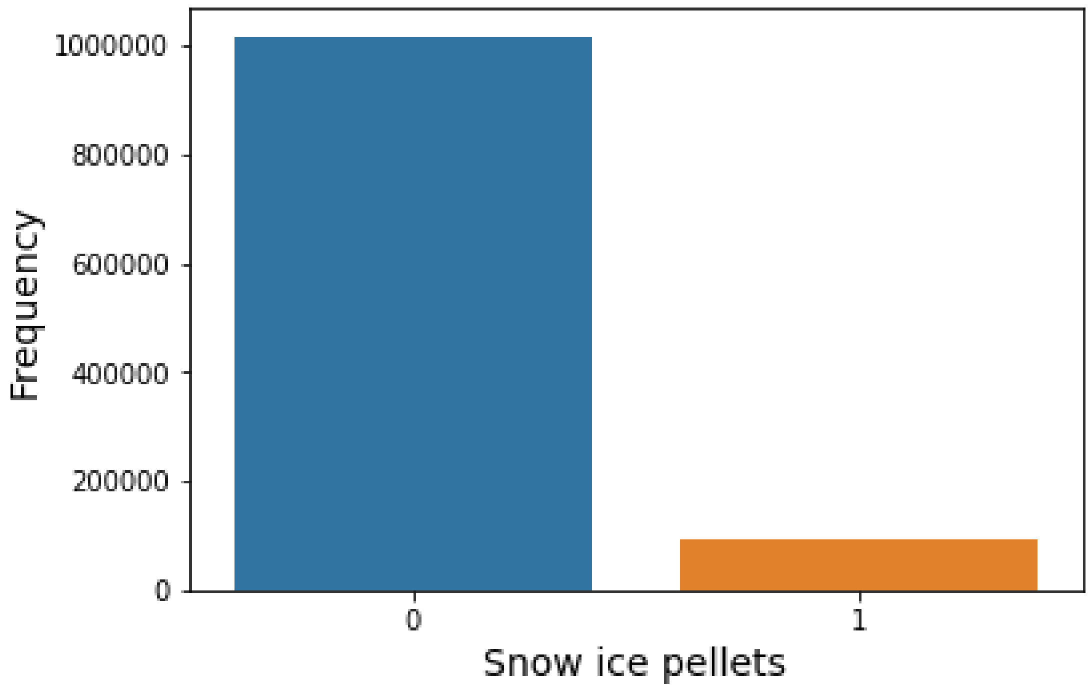


2.1.19 Snow ice pellets

- **Observations:**

In the first 6 months in 2016, most days were no snow ice pellets.

```
In [57]: sns.countplot(data['snow_ice_pellets'])
         plt.xlabel('Snow ice pellets', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.xticks(plt.xticks()[0], rotation=0)
         plt.show()
```

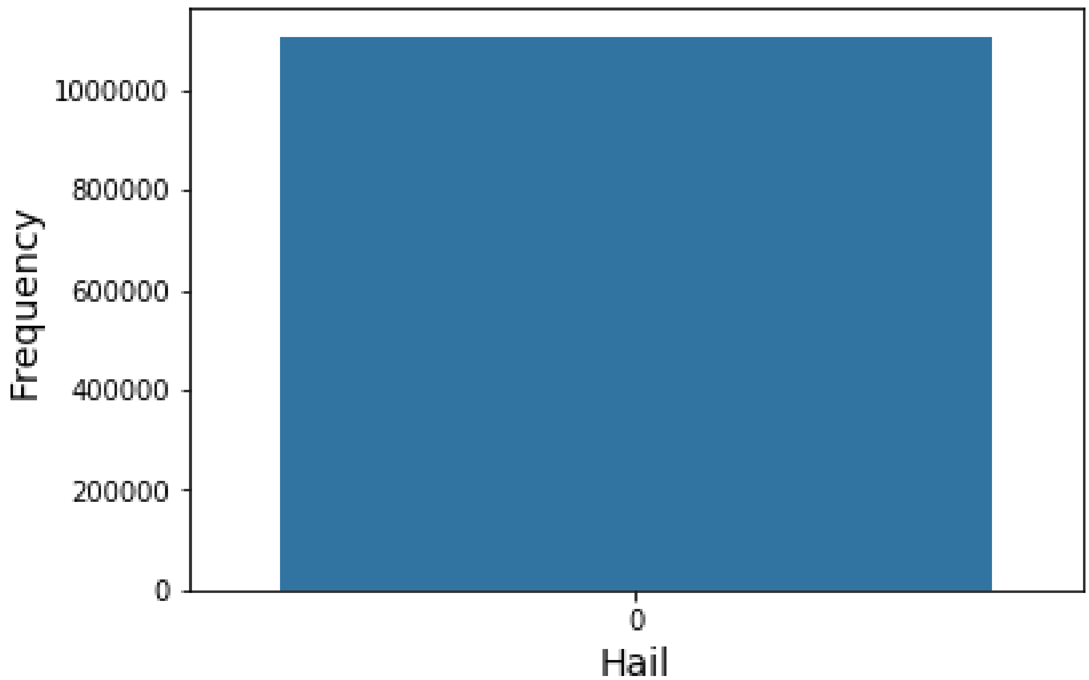


2.1.20 Hail

- **Observations:**

There are only 'no hail' record, so it will not be selected as a feature.

```
In [58]: sns.countplot(data['hail'])
         plt.xlabel('Hail', fontsize=14)
         plt.ylabel('Frequency', fontsize=14)
         plt.xticks(plt.xticks()[0], rotation=0)
         plt.show()
```

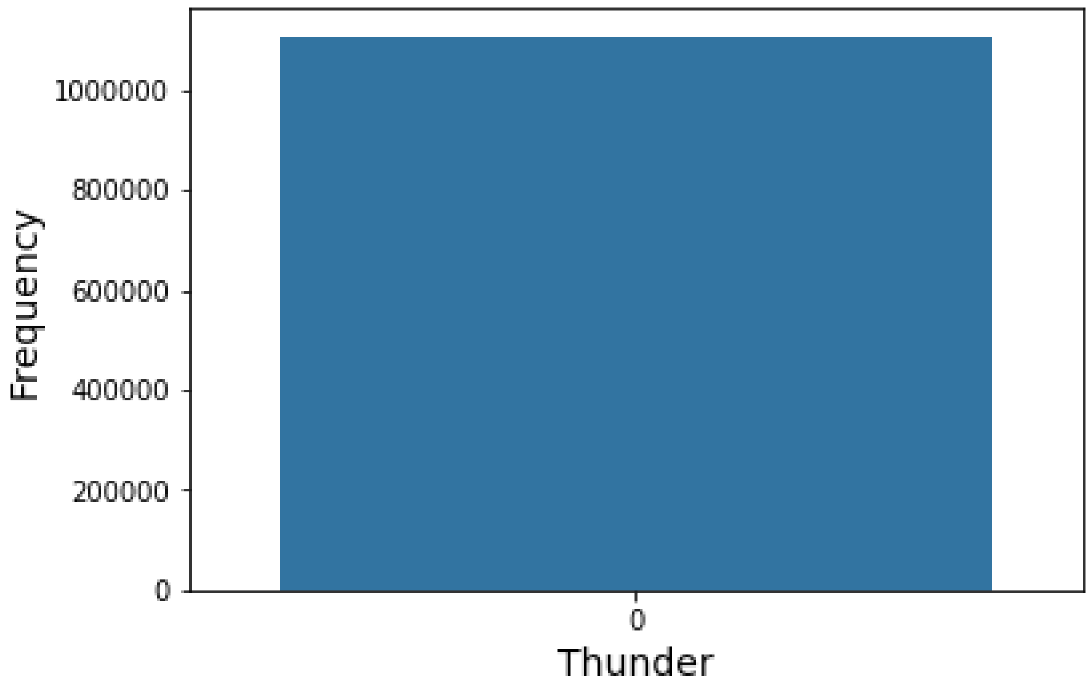


2.1.21 Thunder

- **Observations:**

There are only 'no thunder' record, so it will not be selected as a feature.

In [59]:
```
sns.countplot(data['thunder'])
plt.xlabel('Thunder', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.xticks(plt.xticks()[0], rotation=0)
plt.show()
```



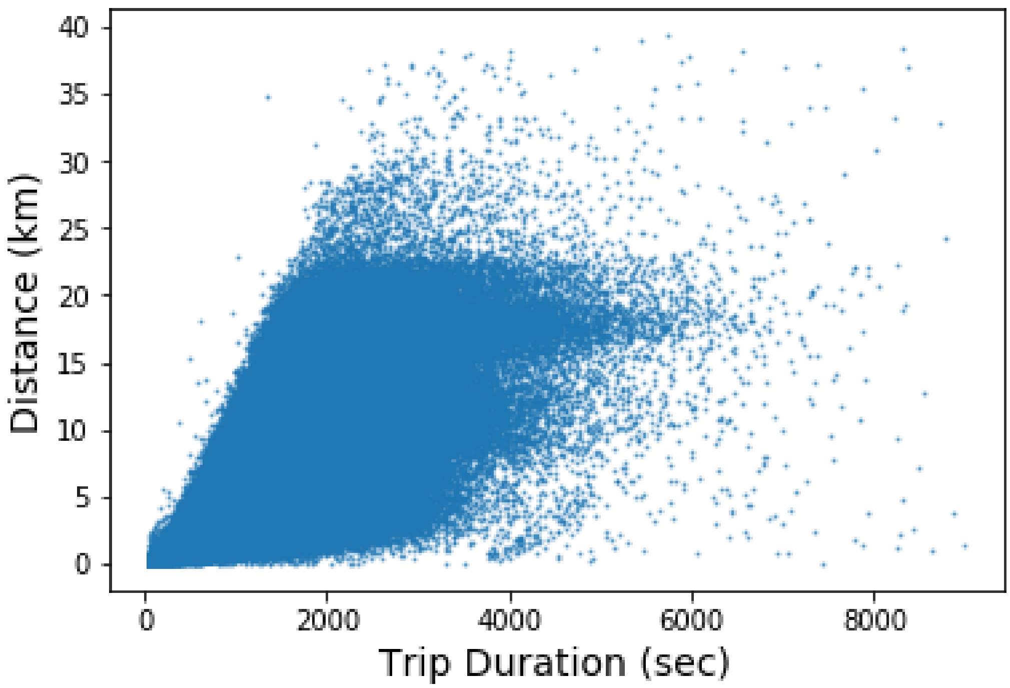

```
In [ ]: # # Drop undesired columns

        # data = data.drop(labels='pickup_datetime', axis=1)
        # data = data.drop(labels='dropoff_datetime', axis=1)
        # data = data.drop(labels='rate_code', axis=1)
        # data = data.drop(labels='store_and_fwd_flag', axis=1)
        # data = data.drop(labels='payment_type', axis=1)
        # data = data.drop(labels='date_of_year2', axis=1)
        # data = data.drop(labels='year_of_year', axis=1)
        # data = data.drop(labels='fare_amount', axis=1)
        # data = data.drop(labels='extra', axis=1)
        # data = data.drop(labels='mta_tax', axis=1)
        # data = data.drop(labels='tip_amount', axis=1)
        # data = data.drop(labels='tolls_amount', axis=1)
        # data = data.drop(labels='imp_surcharge', axis=1)
        # data = data.drop(labels='total_amount', axis=1)
        # data = data.drop(labels='hail', axis=1)
        # data = data.drop(labels='thunder', axis=1)

        # data.head()
```

2.2 Bivariate Analysis

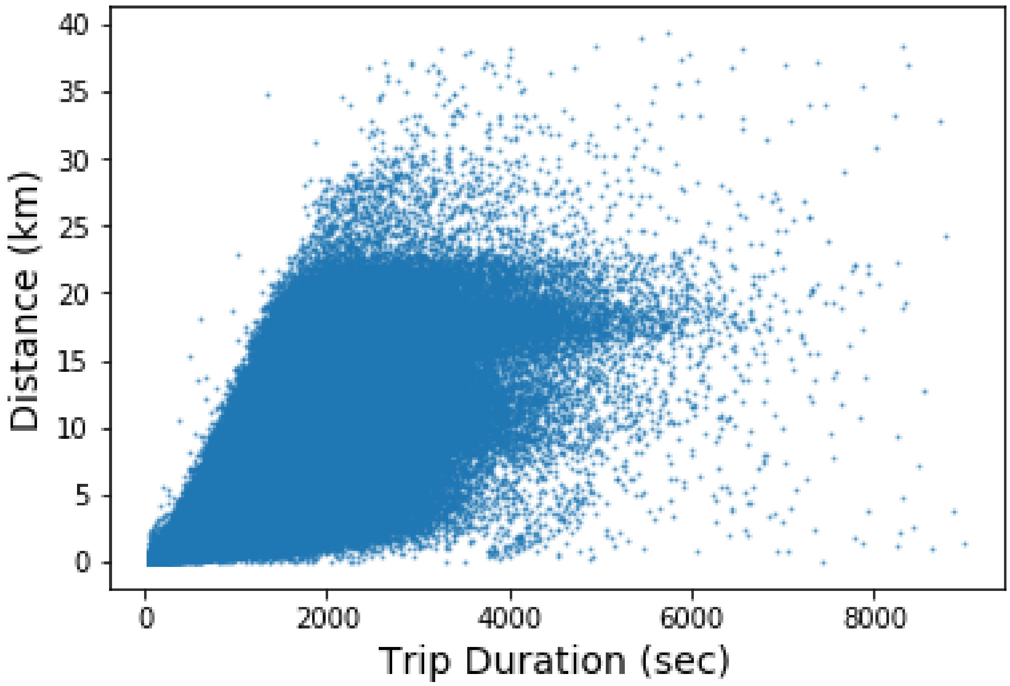
In this section, we roughly explored the correlation between different features and trip duration, distance and speed. This might help the following feature engineering and feature selection.

2.2.1 Travel duration vs. Distance

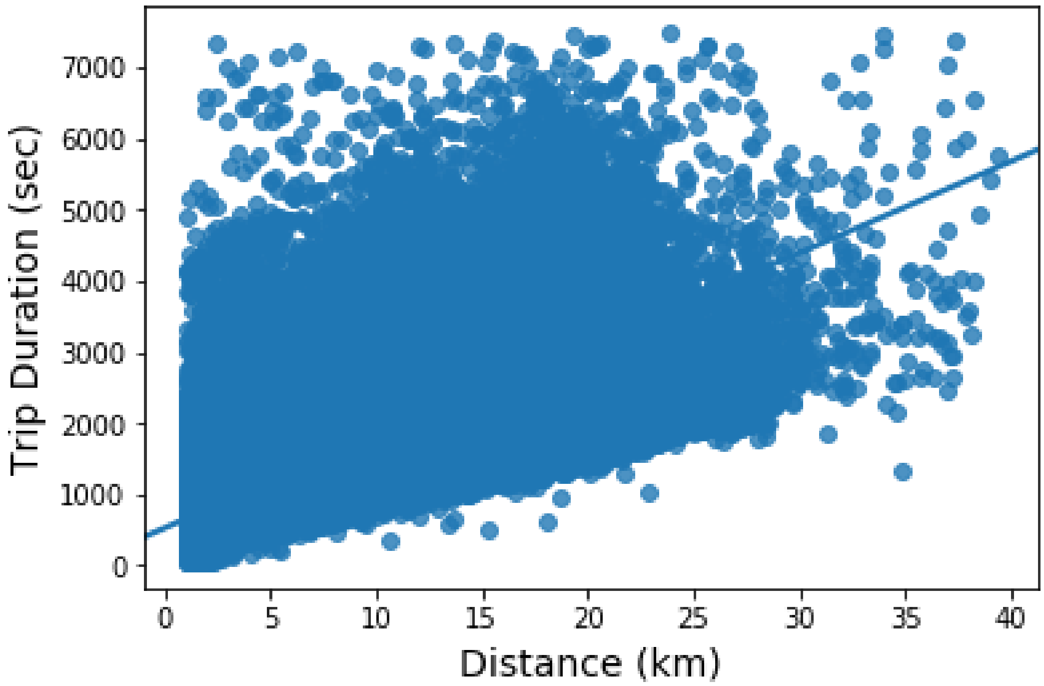
```
In [60]: plt.scatter(data['travel_time'], data['trip_distance'] , s=1, alpha=0.5)
         plt.ylabel('Distance (km)', fontsize=14)
         plt.xlabel('Trip Duration (sec)', fontsize=14)
         plt.show()
```




```
In [61]: dur_dist = data.loc[(data['travel_time'] > 50) & (data['travel_time'] < 10000), ['tri
         plt.scatter(dur_dist['travel_time'], dur_dist['trip_distance'], s=1, alpha=0.5)
         plt.ylabel('Distance (km)', fontsize=14)
         plt.xlabel('Trip Duration (sec)', fontsize=14)
         plt.show()
```

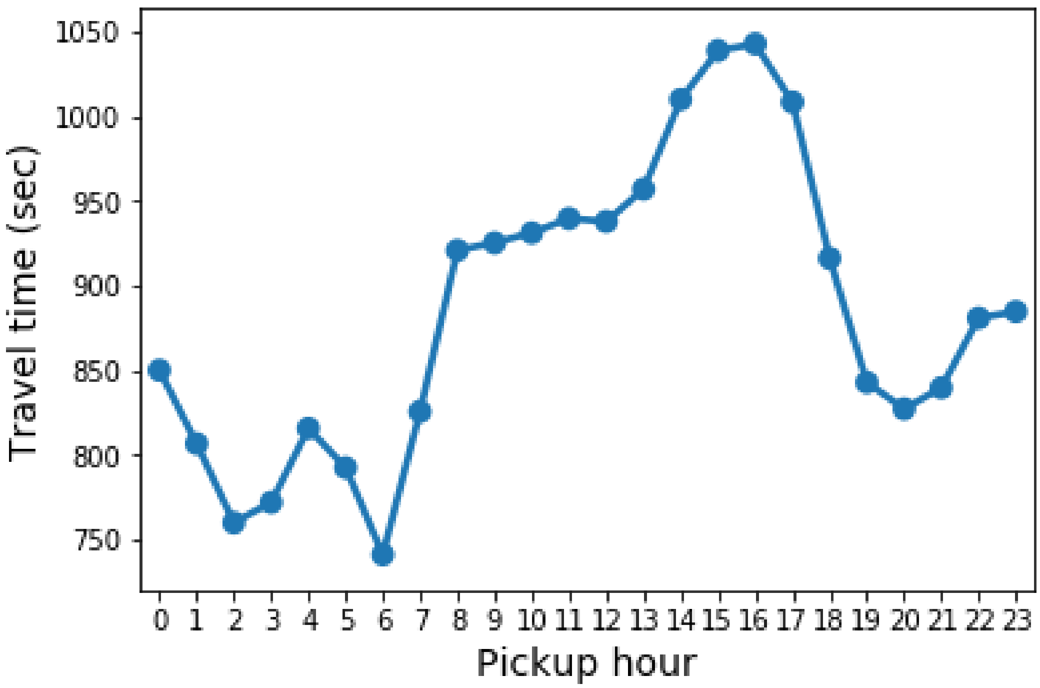



```
In [62]: dur_dist_regr = data.loc[(data['trip_distance']  >= 1) & (data['travel_time'] <= 7500

In [63]: sns.regplot(dur_dist_regr.trip_distance, dur_dist_regr.travel_time)
         plt.xlabel('Distance (km)', fontsize=14)
         plt.ylabel('Trip Duration (sec)', fontsize=14)
         plt.show()
```



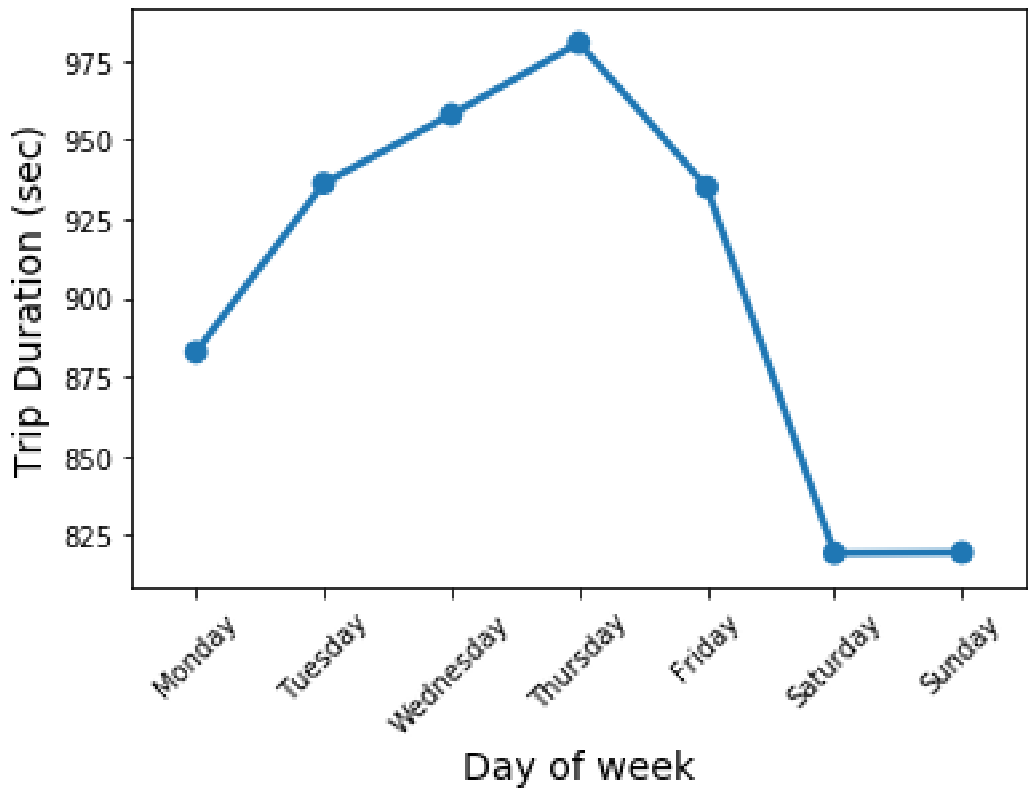
2.2.2 Travel duration vs. Pickup hour

```
In [64]: dur_hr = data.groupby('pickup_hour').travel_time.mean()
         sns. pointplot(dur_hr.index, dur_hr.values)
         plt.ylabel('Travel time (sec)', fontsize=14)
         plt.xlabel('Pickup hour', fontsize=14)
         plt.show()
```



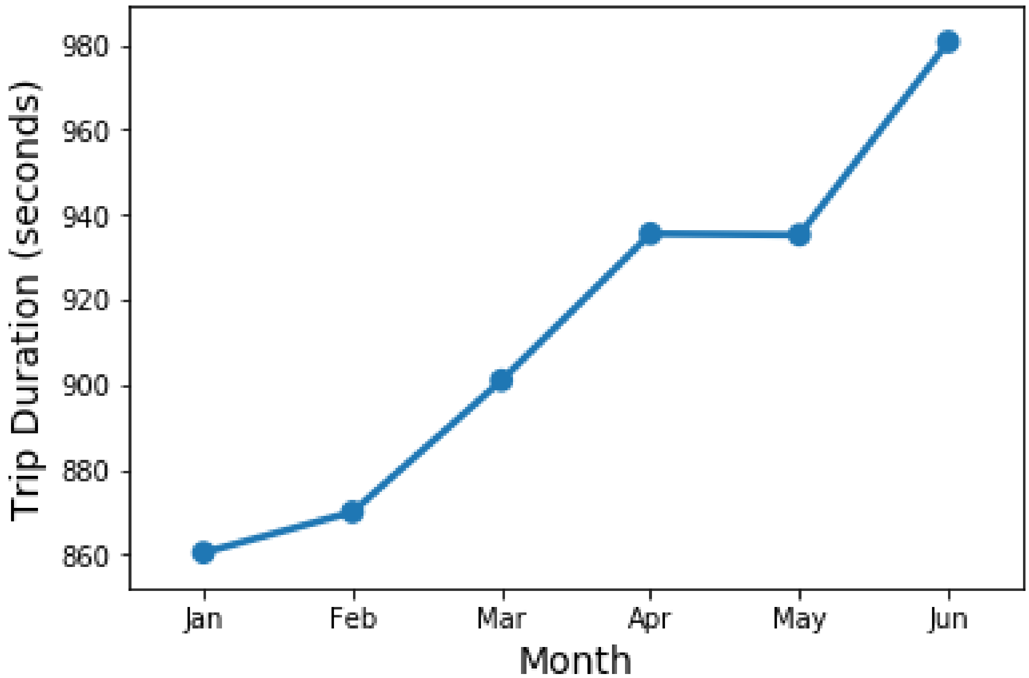
2.2.3 Trip duration vs. Day of week

```
In [65]: dur_da = data.groupby('day_of_week').travel_time.mean()
         sns.pointplot(dur_da.index, dur_da.values)
         plt.ylabel('Trip Duration (sec)', fontsize=14)
         plt.xlabel('Day of week', fontsize=14)
         plt.xticks(plt.xticks()[0],
                    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunda
                    rotation=45)
         plt.show()
```



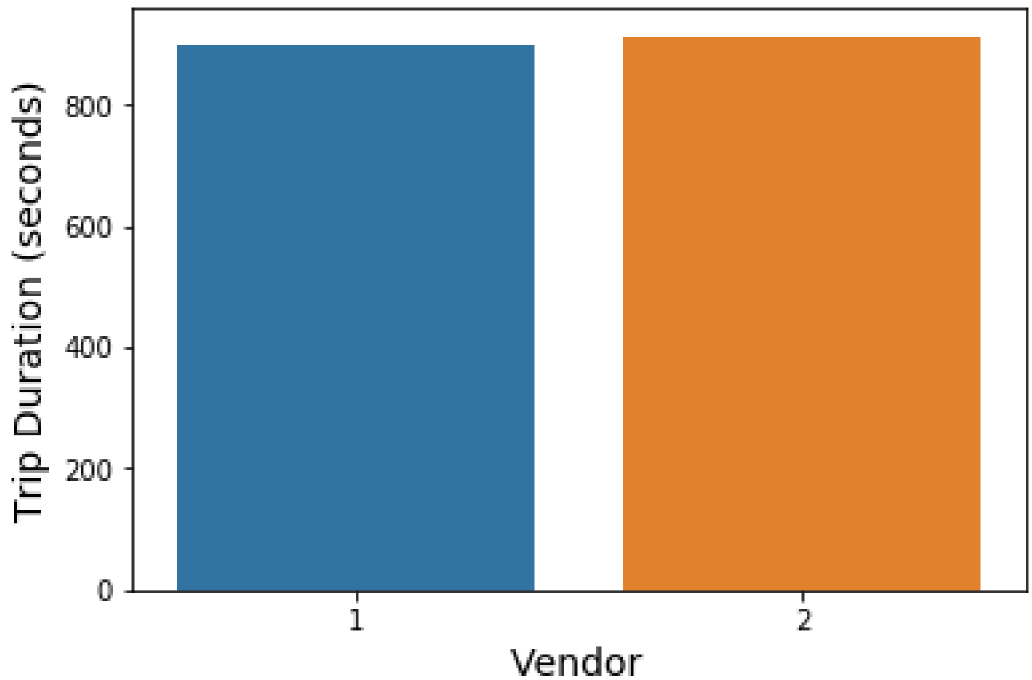
2.2.4 Trip duration vs. Month of year

```
In [66]: dur_mo = data.groupby('month_of_year').travel_time.mean()
         sns.pointplot(dur_mo.index, dur_mo.values)
         plt.ylabel('Trip Duration (seconds)', fontsize=14)
         plt.xlabel('Month', fontsize=14)
         plt.xticks(plt.xticks()[0],
                    ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'])
         plt.show()
```



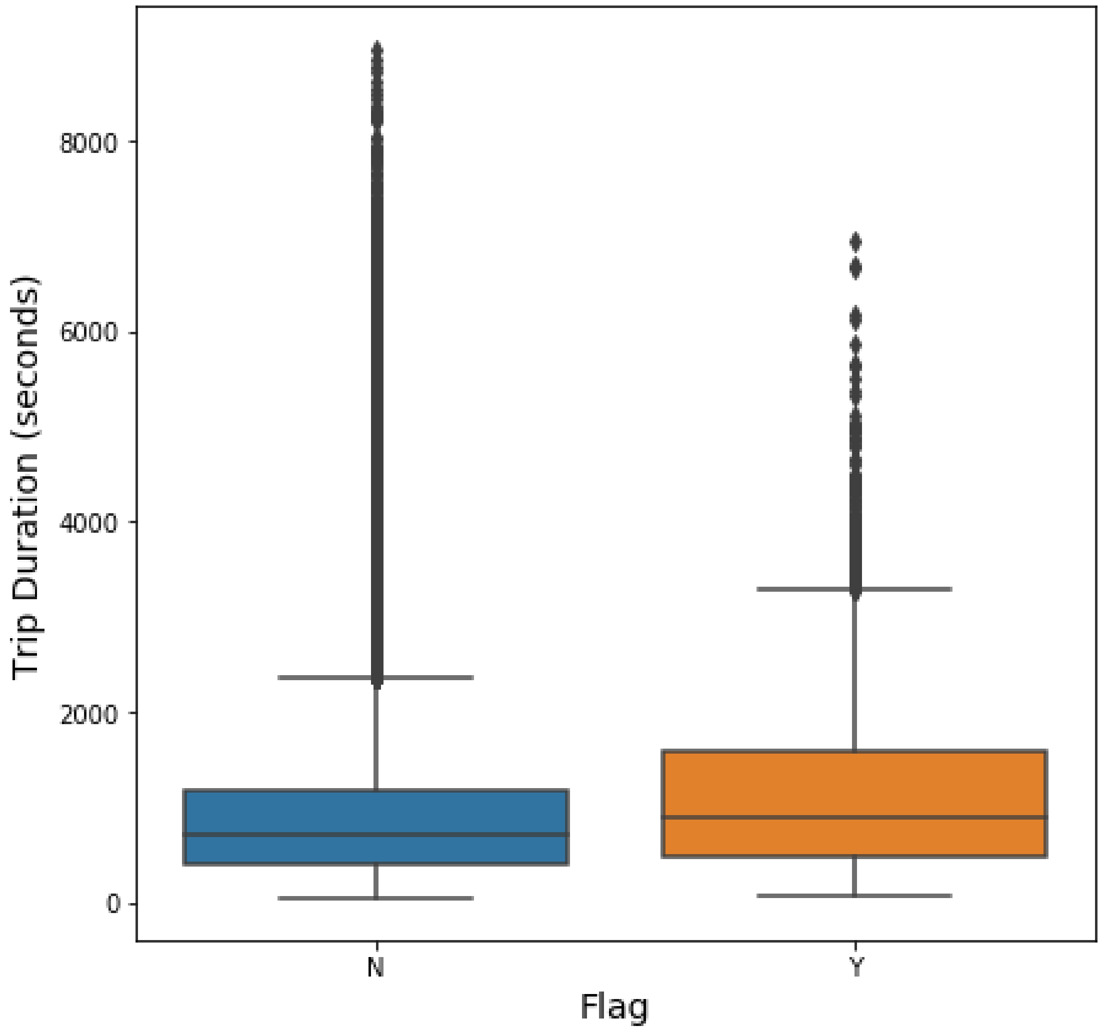
2.2.5 Trip duration vs. Vendor

```
In [67]: dur_ven = data.groupby('vendor_id').travel_time.mean()
         sns.barplot(dur_ven.index, dur_ven.values)
         plt.xlabel('Vendor', fontsize=14)
         plt.ylabel('Trip Duration (seconds)', fontsize=14)
         plt.show()
```



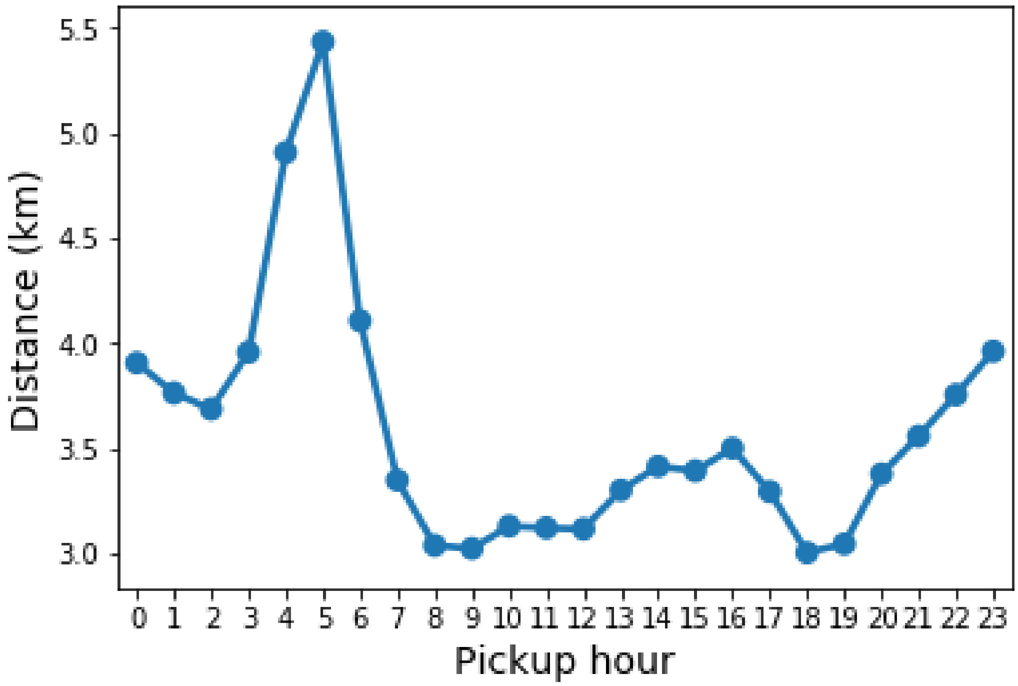
2.2.6 Trip duration vs. Flag

```
In [68]: plt.figure(figsize = (7,7))
         dur = data.loc[(data['travel_time']) < 10000]
         sns.boxplot(x = 'flag_Y', y = 'travel_time', data = dur)
         plt.xlabel('Flag', fontsize=14)
         plt.ylabel('Trip Duration (seconds)', fontsize=14)
         plt.xticks(plt.xticks()[0],
                    ['N', 'Y'])
         plt.show()
```



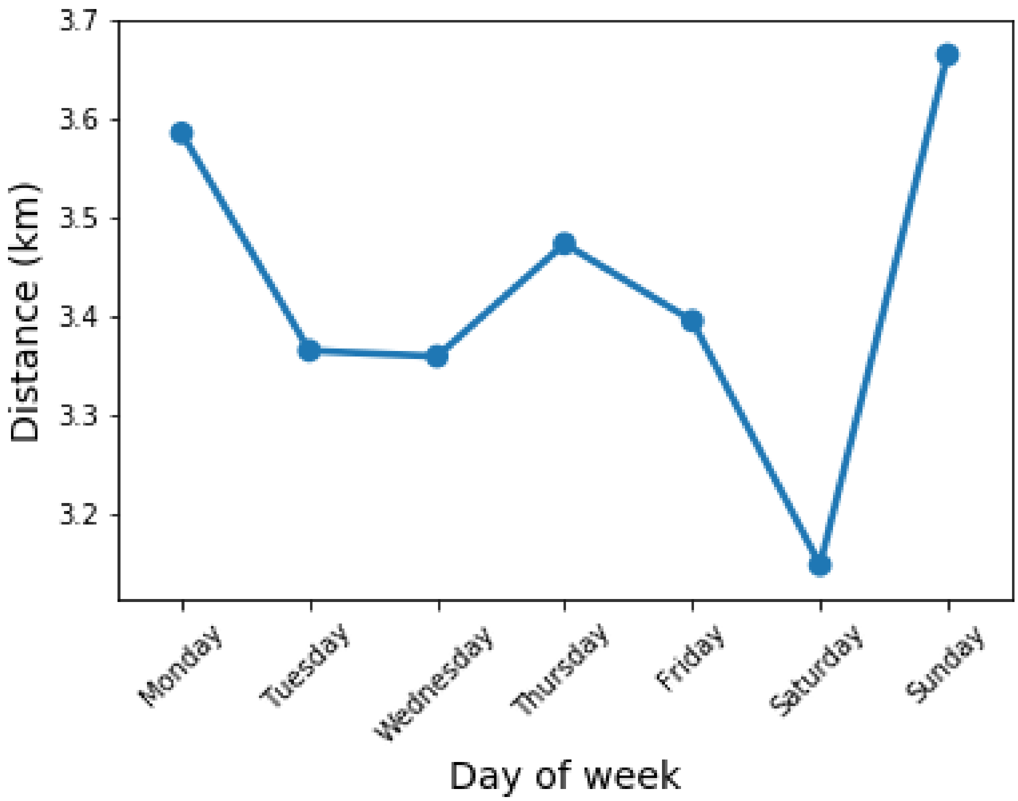
2.2.7 Distance vs. Pickup hour

In [69]:
```
dist_hr = data.groupby('pickup_hour').trip_distance.mean()
sns.pointplot(dist_hr.index, dist_hr.values)
plt.ylabel('Distance (km)', fontsize=14)
plt.xlabel('Pickup hour', fontsize=14)
plt.show()
```



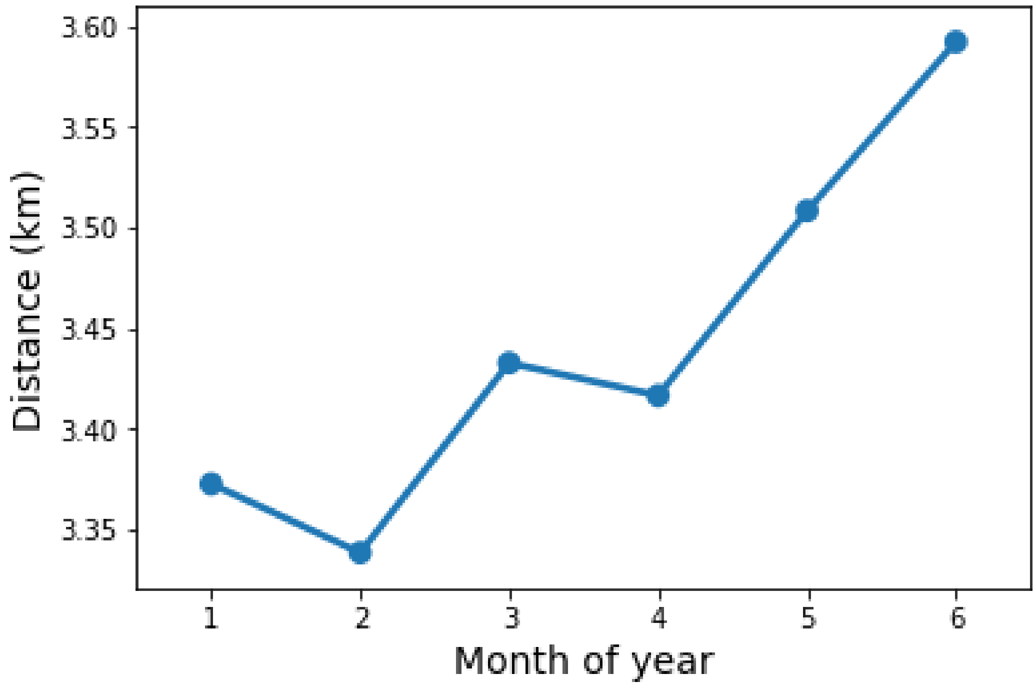
2.2.8 Distance vs. Day of week

```
In [70]: dist_da = data.groupby('day_of_week').trip_distance.mean()
         sns.pointplot(dist_da.index, dist_da.values)
         plt.xlabel('Day of week', fontsize=14)
         plt.ylabel('Distance (km)', fontsize=14)
         plt.xticks(plt.xticks()[0],
                    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunda
                    rotation=45)
         plt.show()
```



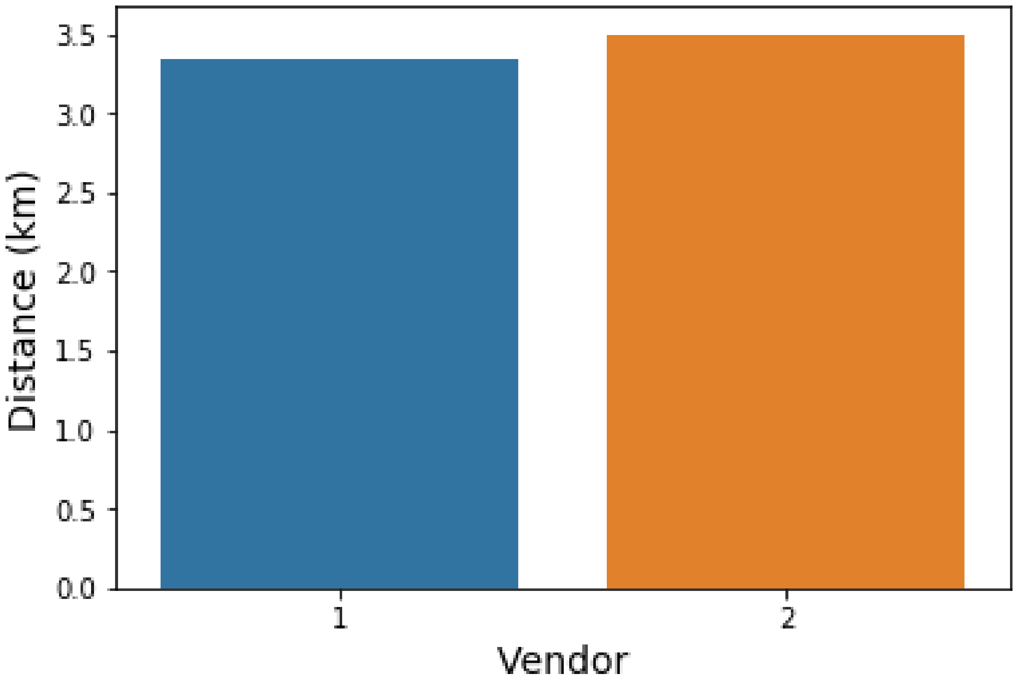
2.2.9 Distance vs. Month of year

```
In [71]: dist_mo = data.groupby('month_of_year').trip_distance.mean()
         sns.pointplot(dist_mo.index, dist_mo.values)
         plt.xlabel('Month of year', fontsize=14)
         plt.ylabel('Distance (km)', fontsize=14)
         plt.show()
```



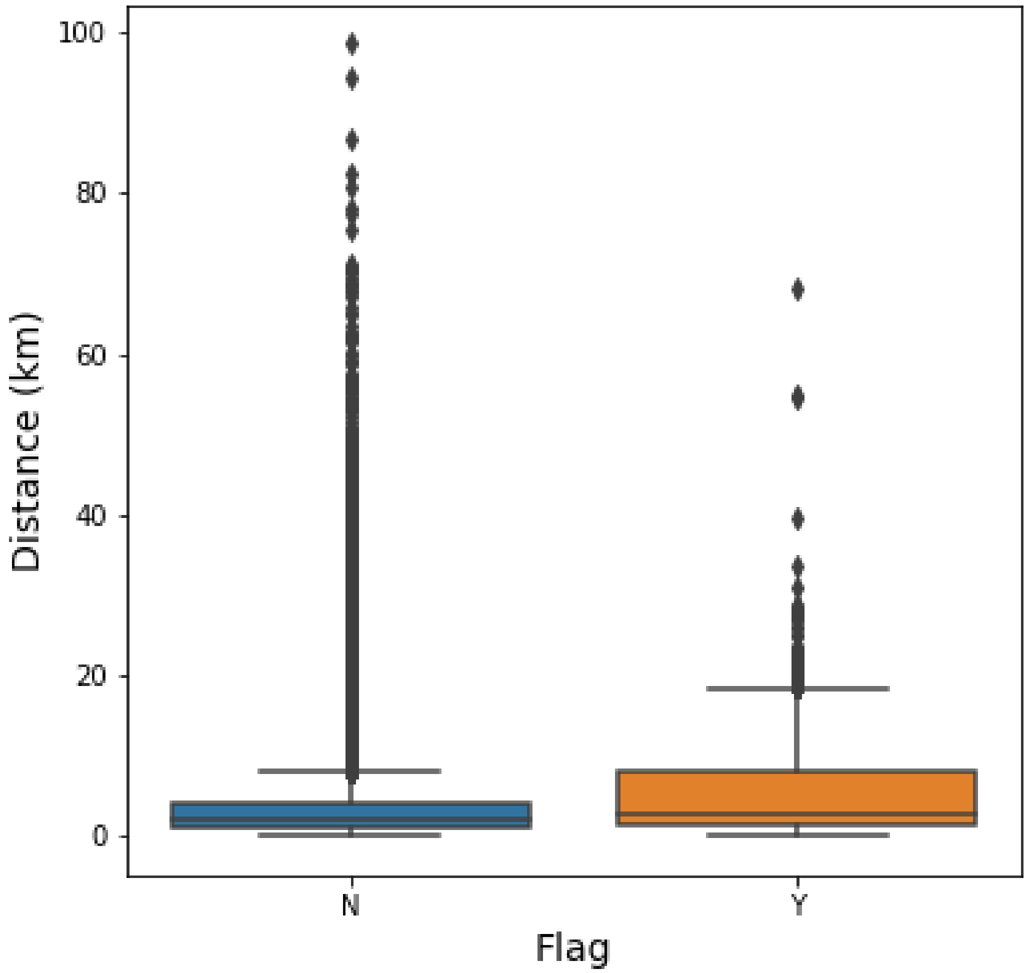
10. Distance vs. Vendor

```
In [66]: dist_ven = data.groupby('vendor_id').trip_distance.mean()
         sns.barplot(dist_ven.index, dist_ven.values)
         plt.xlabel('Vendor', fontsize=14)
         plt.ylabel("Distance (km)", fontsize=14)
         plt.show()
```



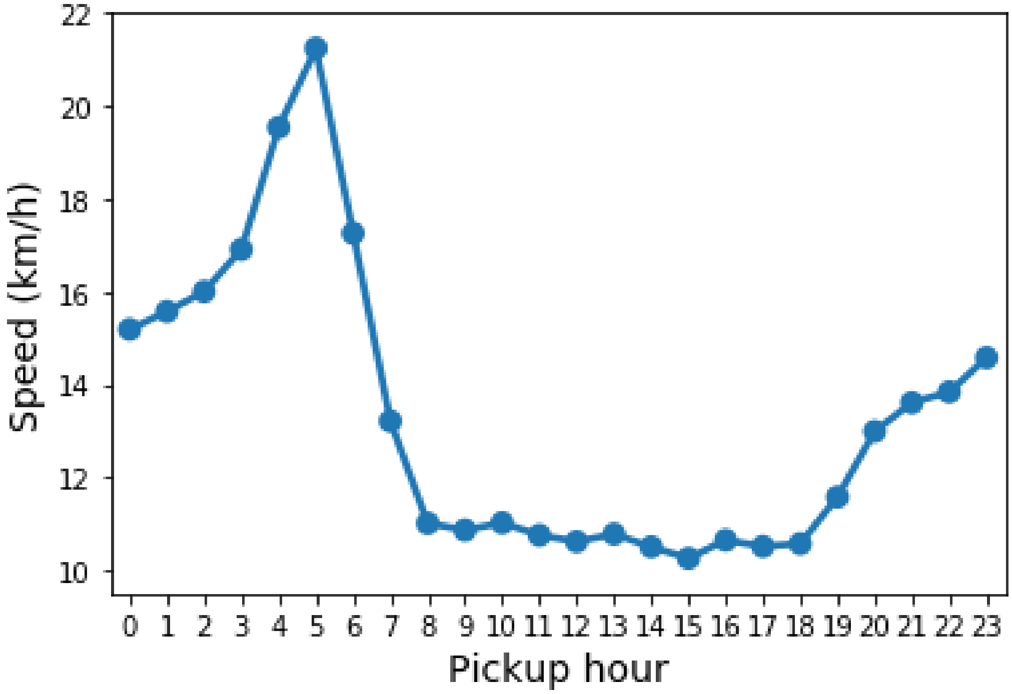
11. Distance vs. Flag

```
In [66]: plt.figure(figsize = (6,6))
         plot_dist = data.loc[(data['trip_distance'] < 100)]
         sns.boxplot(x = 'flag_Y', y = 'trip_distance', data = plot_dist)
         plt.xlabel('Flag', fontsize=14)
         plt.ylabel('Distance (km)', fontsize=14)
         plt.xticks(plt.xticks()[0],
                    ['N', 'Y'])
         plt.show()
```



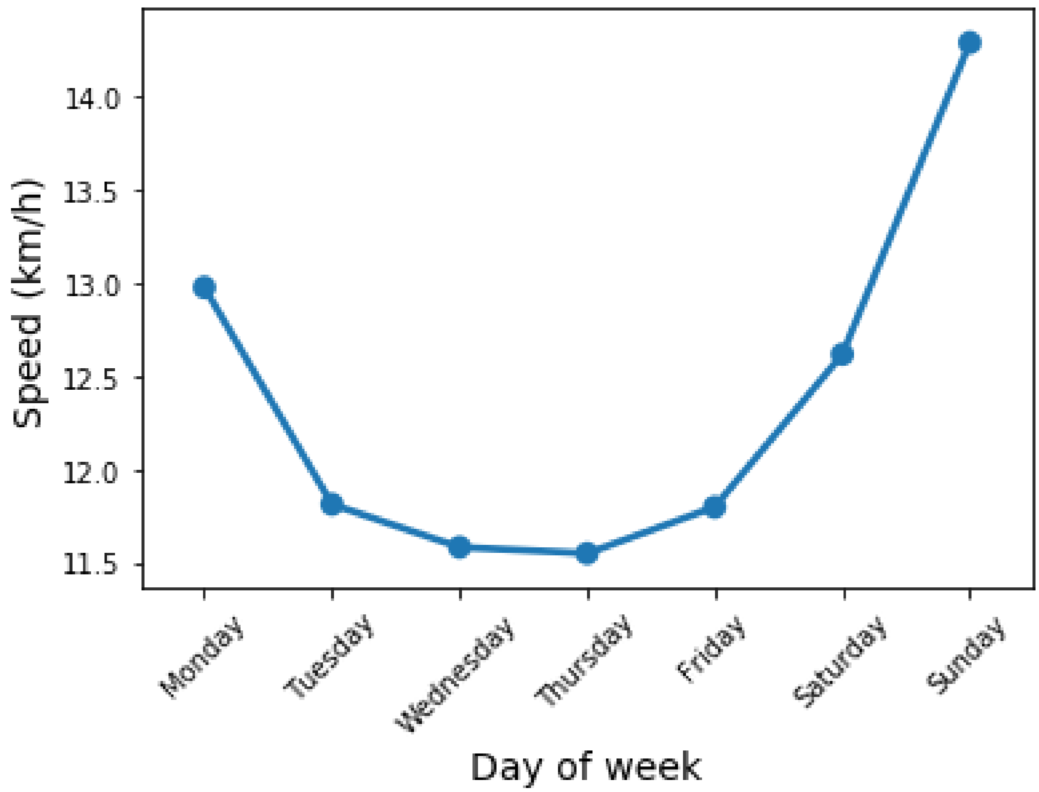
12. Speed vs. Pickup hour

```
In [67]: sp_hr = data.groupby('pickup_hour').speed.mean()
         sns.pointplot(sp_hr.index, sp_hr.values)
         plt.xlabel('Pickup hour', fontsize=14)
         plt.ylabel("Speed (km/h)", fontsize=14)
         plt.show()
```



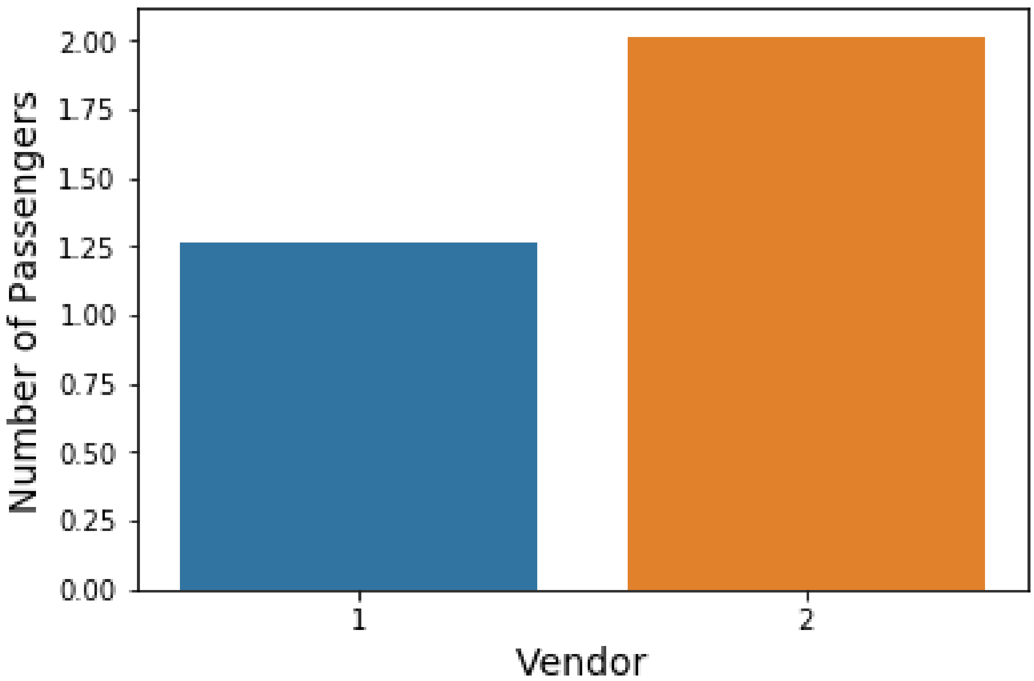
13. Speed vs. Day of week

```
In [68]: sp_da = data.groupby('day_of_week').speed.mean()
         sns.pointplot(sp_da.index, sp_da.values)
         plt.xlabel('Day of week', fontsize=14)
         plt.ylabel("Speed (km/h)", fontsize=14)
         plt.xticks(plt.xticks()[0],
                    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunda
                    rotation=45)
         plt.show()
```

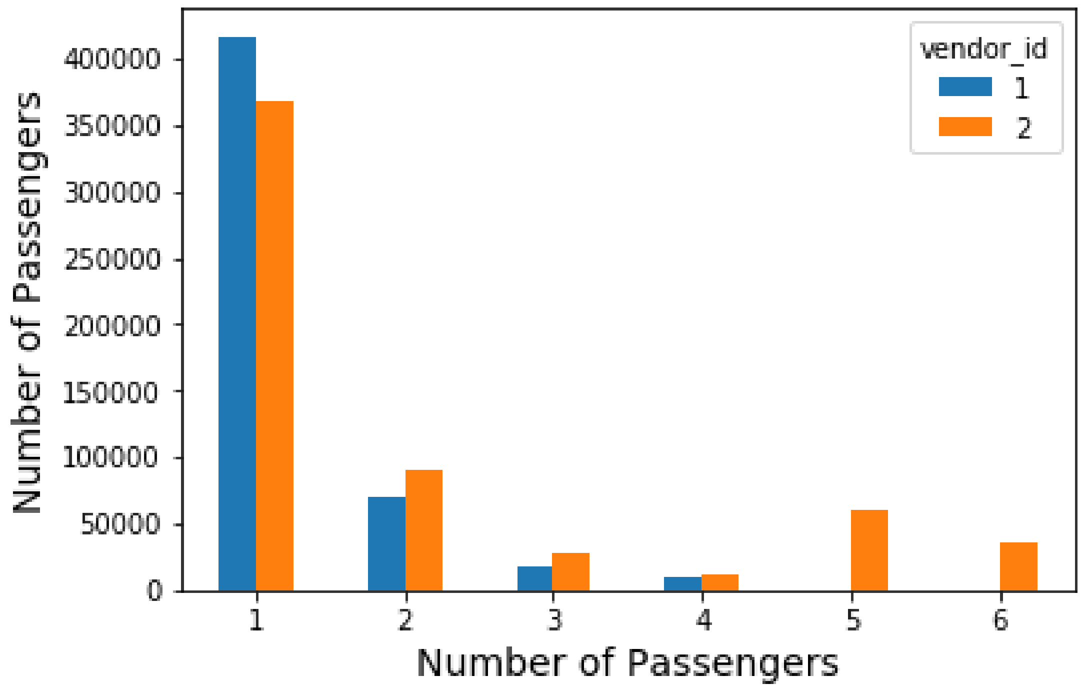


14. Number of passengers vs. Vendor

```
In [69]: no_ven = data.groupby('vendor_id').passenger_count.mean()
         sns.barplot(no_ven.index, no_ven.values)
         plt.xlabel('Vendor', fontsize=14)
         plt.ylabel('Number of Passengers', fontsize=14)
         plt.show()
```

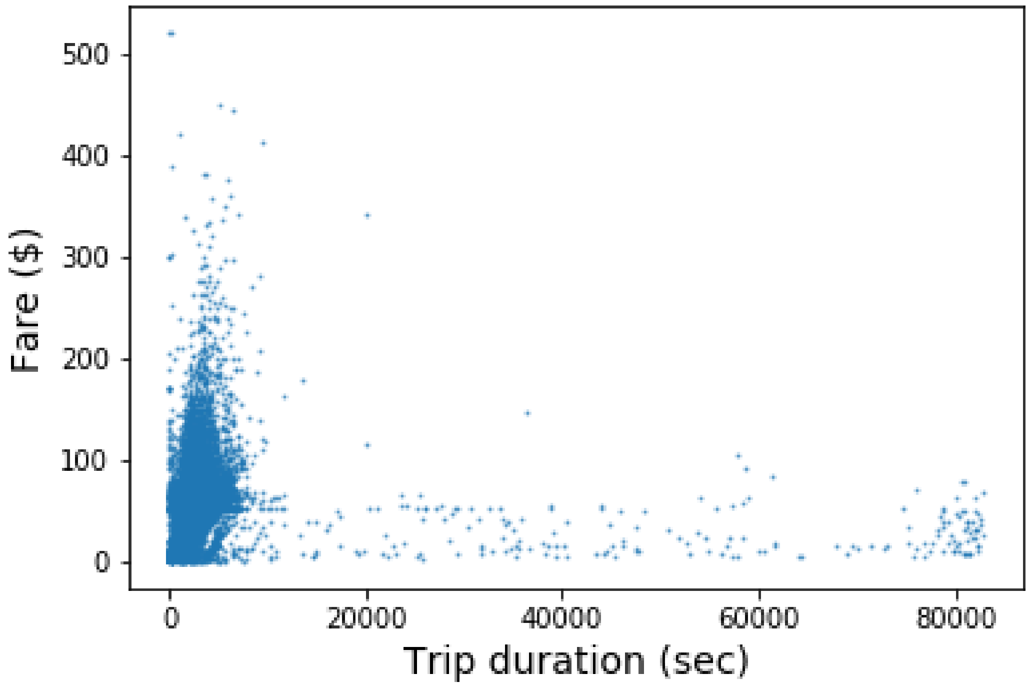



```
In [79]: data.groupby('passenger_count').vendor_id.value_counts().reset_index(name='count').pi
         plt.xlabel('Number of Passengers', fontsize=14)
         plt.ylabel('Number of Passengers', fontsize=14)
         plt.xticks(plt.xticks()[0], rotation=0)
         plt.show()
```

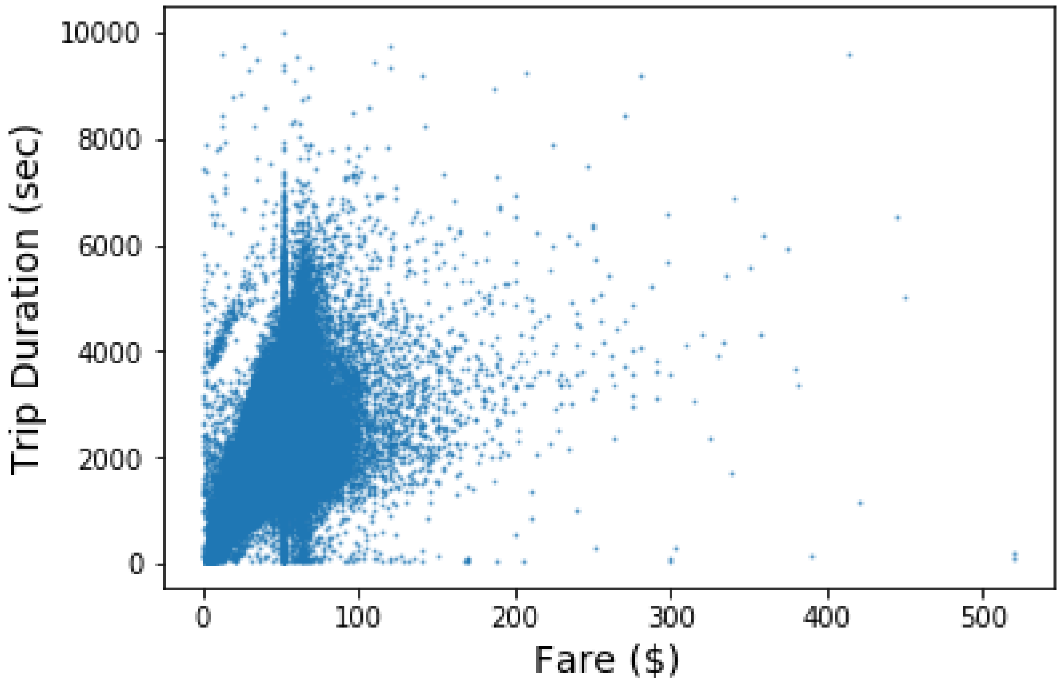


2.2.10 Trip duration vs. Fare

In [138]:
```
plt.scatter(data['travel_time'], data['fare+tip'] , s=1, alpha=0.5)
plt.xlabel('Trip duration (sec)', fontsize=14)
plt.ylabel('Fare ($)', fontsize=14)
plt.show()
```

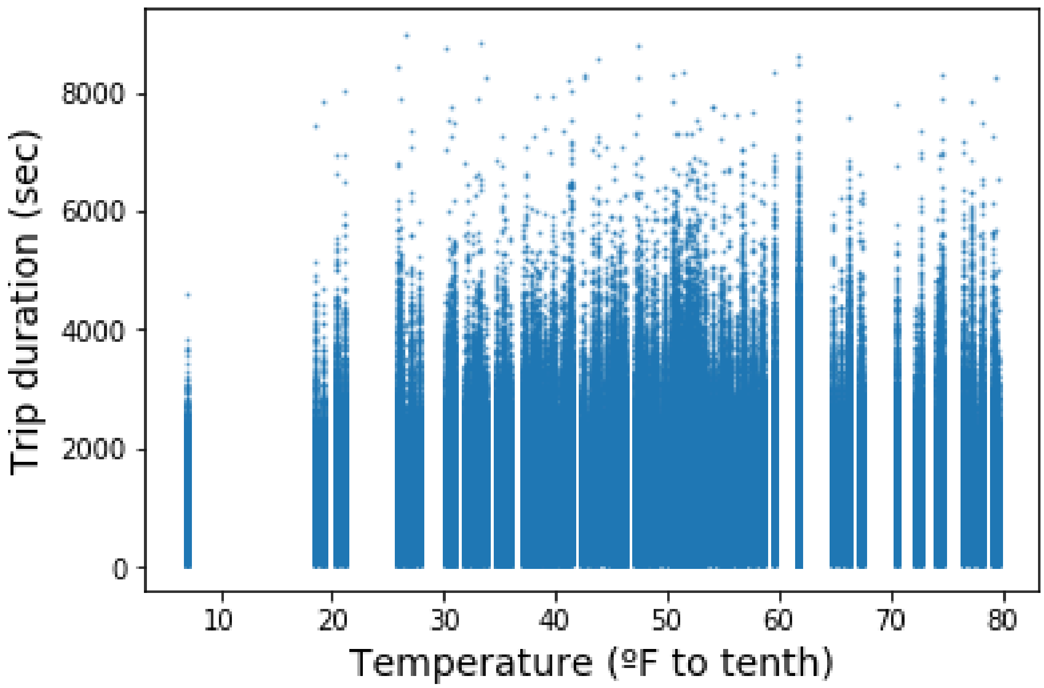



```
In [141]: dur_dist = data.loc[(data['travel_time'] < 10000), ['travel_time', 'fare+tip']]
          plt.scatter(dur_dist['fare+tip'], dur_dist['travel_time'], s=1, alpha=0.5)
          plt.xlabel('Fare ($)', fontsize=14)
          plt.ylabel('Trip Duration (sec)', fontsize=14)
          plt.show()
```

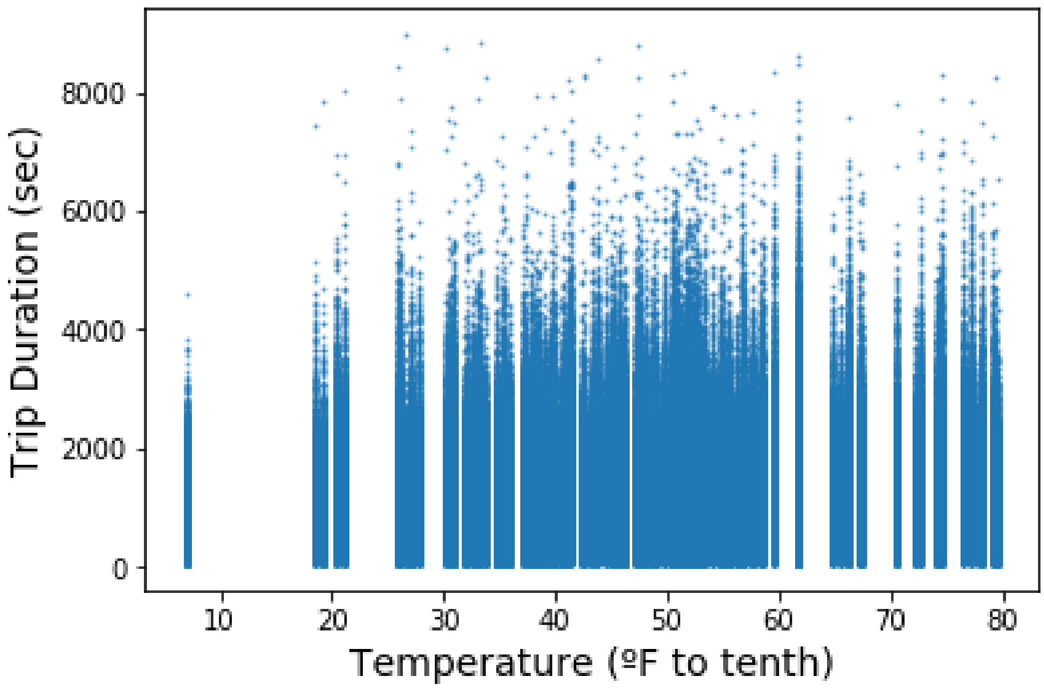


2.2.11 Trip duration vs. Temperature

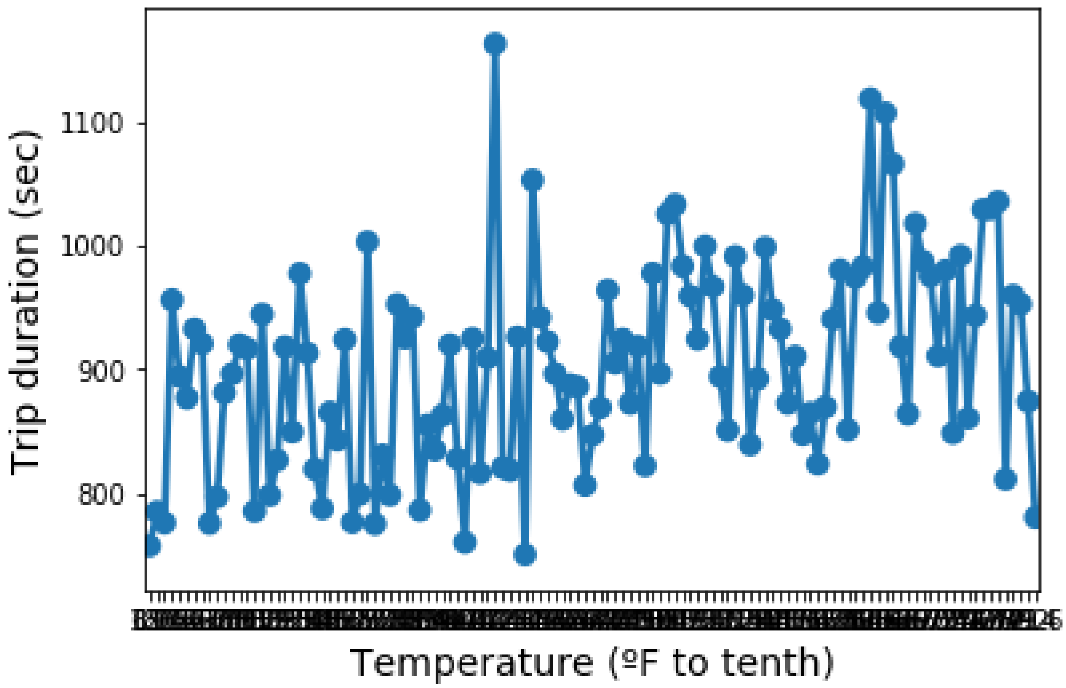
```
In [80]: plt.scatter((data['temp']), data['travel_time'] , s=1, alpha=0.5)
         plt.xlabel('Temperature (žF to tenth)', fontsize=14)
         plt.ylabel('Trip duration (sec)', fontsize=14)
         plt.show()
```




```
In [81]: dur_dist = data.loc[(data['travel_time'] < 10000), ['travel_time', 'temp']]
         plt.scatter(dur_dist['temp'], dur_dist['travel_time'], s=1, alpha=0.5)
         plt.xlabel('Temperature (žF to tenth)', fontsize=14)
         plt.ylabel('Trip Duration (sec)', fontsize=14)
         plt.show()
```

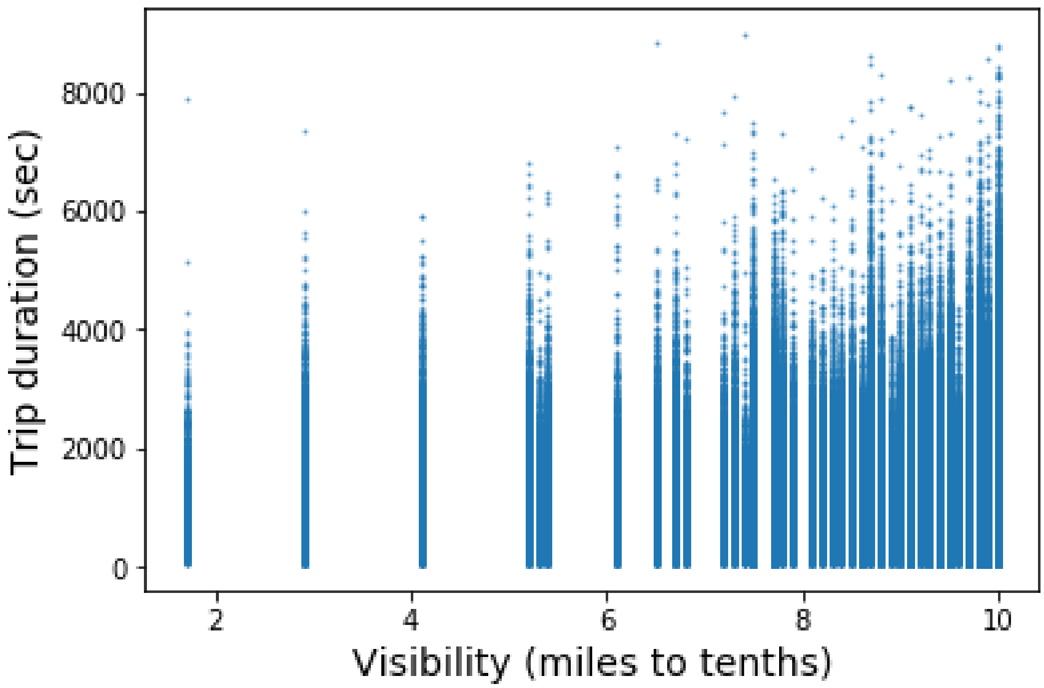



```
In [82]: dur_temp = data.groupby('temp').travel_time.mean()
         sns.pointplot(dur_temp.index, dur_temp.values)
         plt.xlabel('Temperature (žF to tenth)', fontsize=14)
         plt.ylabel('Trip duration (sec)', fontsize=14)
         plt.show()
```

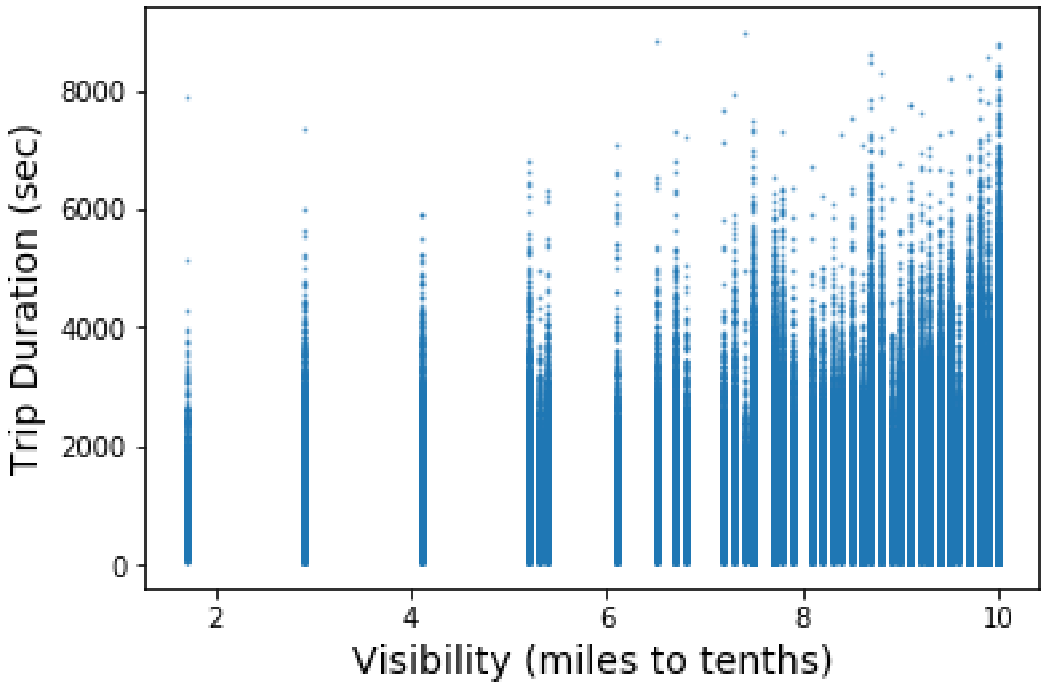


2.2.12 Trip duration vs. Visibility

```
In [83]: plt.scatter((data['visib']), data['travel_time'] , s=1, alpha=0.5)
         plt.xlabel('Visibility (miles to tenths)', fontsize=14)
         plt.ylabel('Trip duration (sec)', fontsize=14)
         plt.show()
```

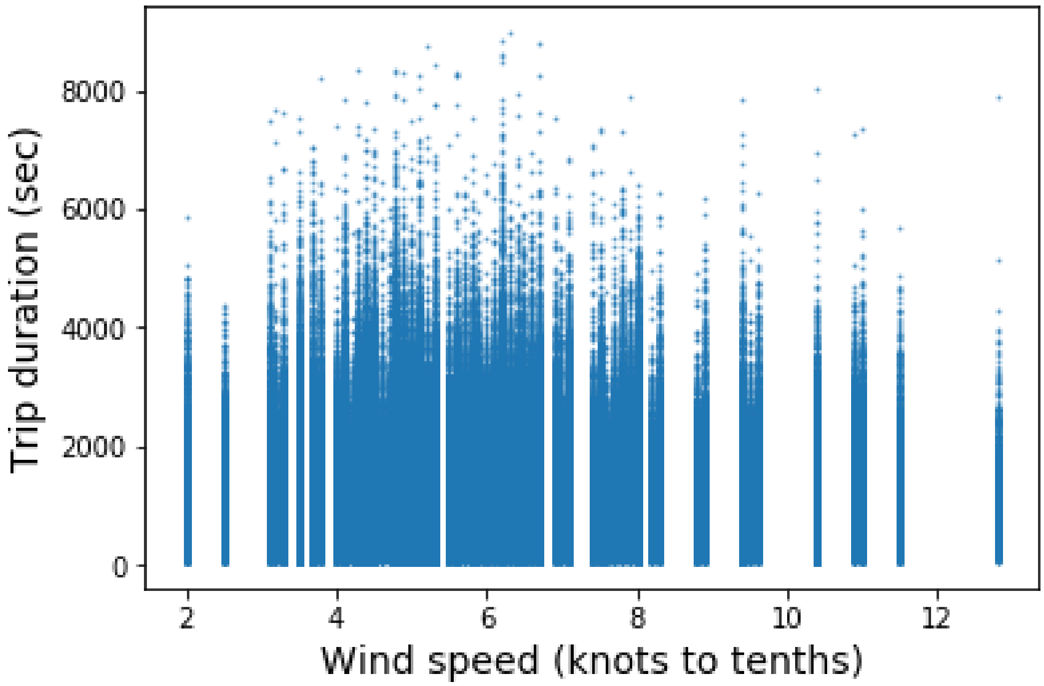



```
In [84]: dur_dist = data.loc[(data['travel_time'] < 10000), ['travel_time', 'visib']]
         plt.scatter(dur_dist['visib'], dur_dist['travel_time'], s=1, alpha=0.5)
         plt.xlabel('Visibility (miles to tenths)', fontsize=14)
         plt.ylabel('Trip Duration (sec)', fontsize=14)
         plt.show()
```



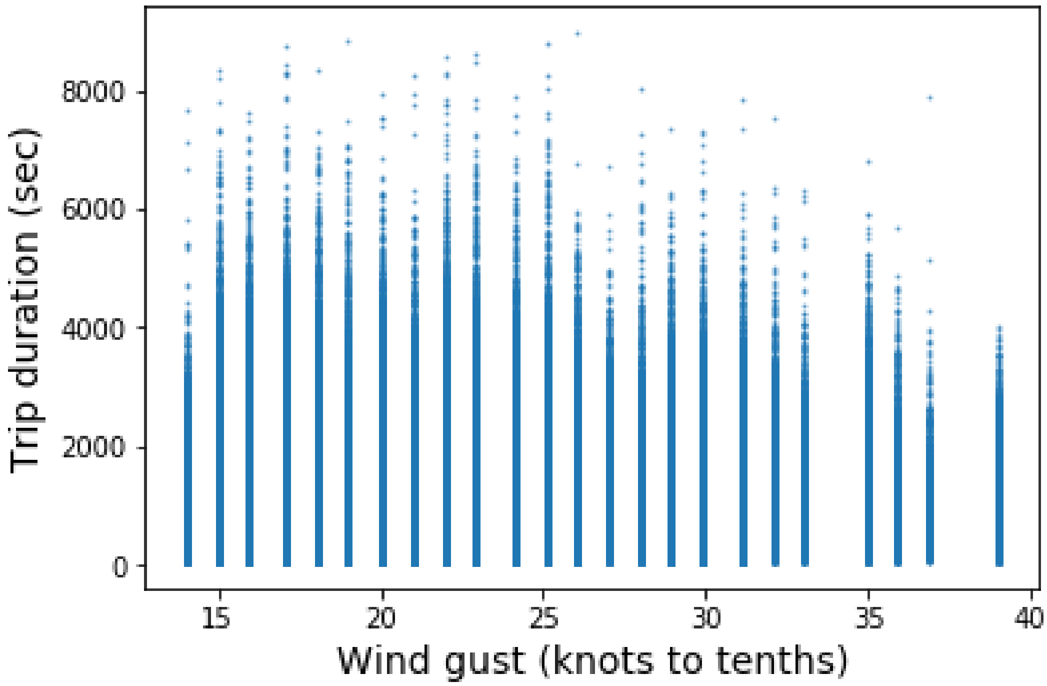
2.2.13 Trip duration vs. Wind speed

```
In [85]: plt.scatter((data['wdsp']), data['travel_time'] , s=1, alpha=0.5)
         plt.xlabel('Wind speed (knots to tenths)', fontsize=14)
         plt.ylabel('Trip duration (sec)', fontsize=14)
         plt.show()
```



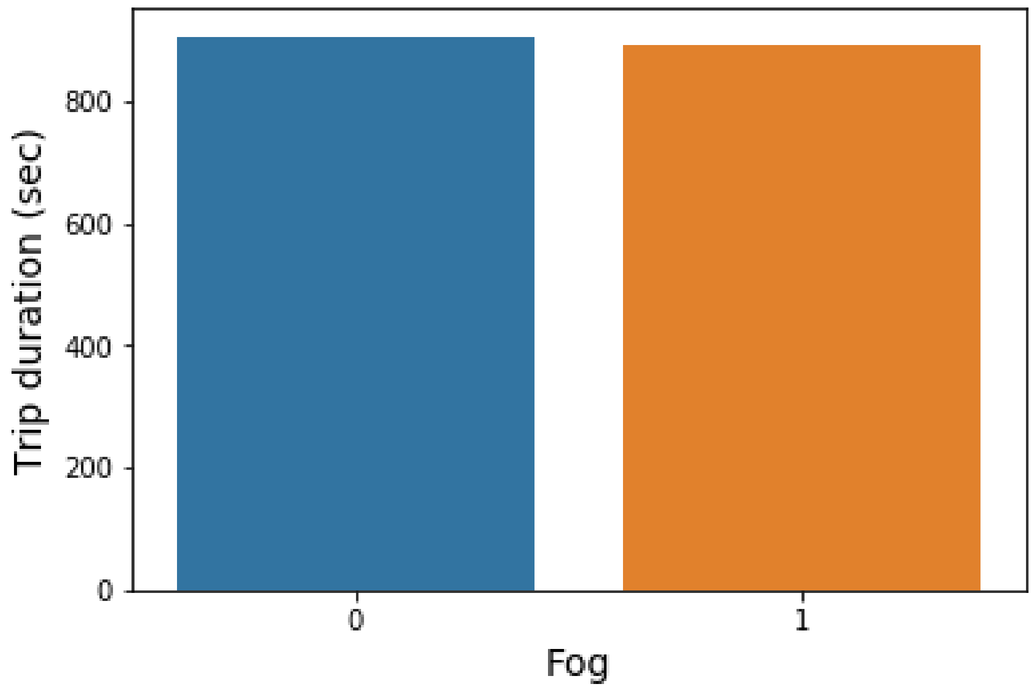
2.2.14 Trip duration vs. Wind gust

```
In [86]: plt.scatter((data['gust']), data['travel_time'] , s=1, alpha=0.5)
         plt.xlabel('Wind gust (knots to tenths)', fontsize=14)
         plt.ylabel('Trip duration (sec)', fontsize=14)
         plt.show()
```



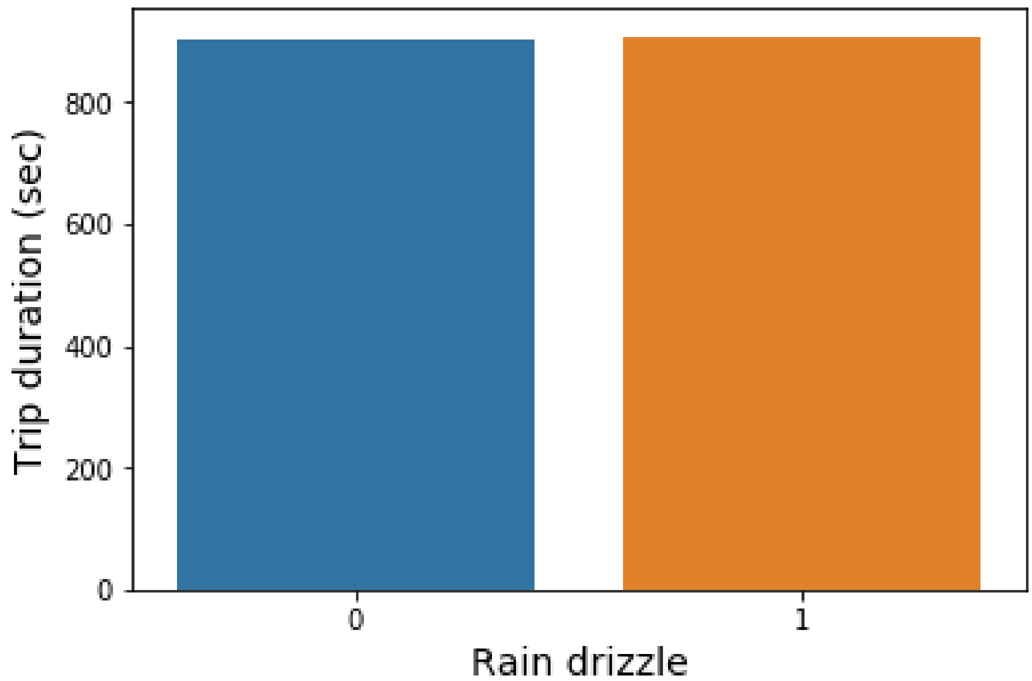
2.2.15 Trip duration vs. Fog

```
In [88]: dur_fog = data.groupby('fog').travel_time.mean()
         sns.barplot(dur_fog.index, dur_fog.values)
         plt.xlabel('Fog', fontsize=14)
         plt.ylabel('Trip duration (sec)', fontsize=14)
         plt.show()
```



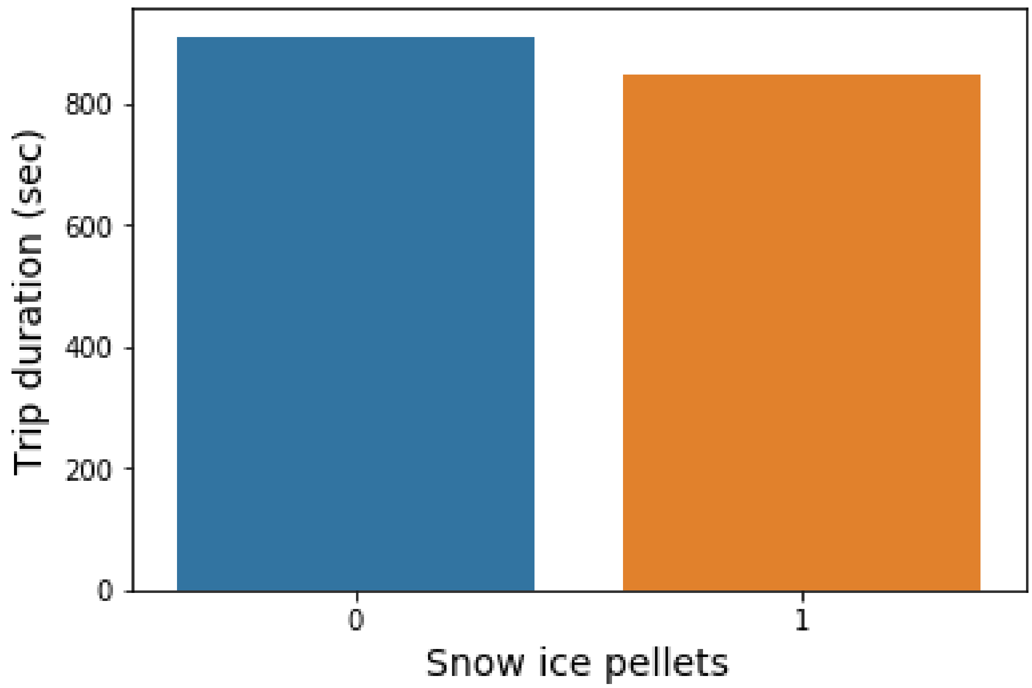
2.2.16 Trip duration vs. Rain drizzle

```
In [89]: dur_rain = data.groupby('rain_drizzle').travel_time.mean()
         sns.barplot(dur_rain.index, dur_rain.values)
         plt.xlabel('Rain drizzle', fontsize=14)
         plt.ylabel('Trip duration (sec)', fontsize=14)
         plt.show()
```



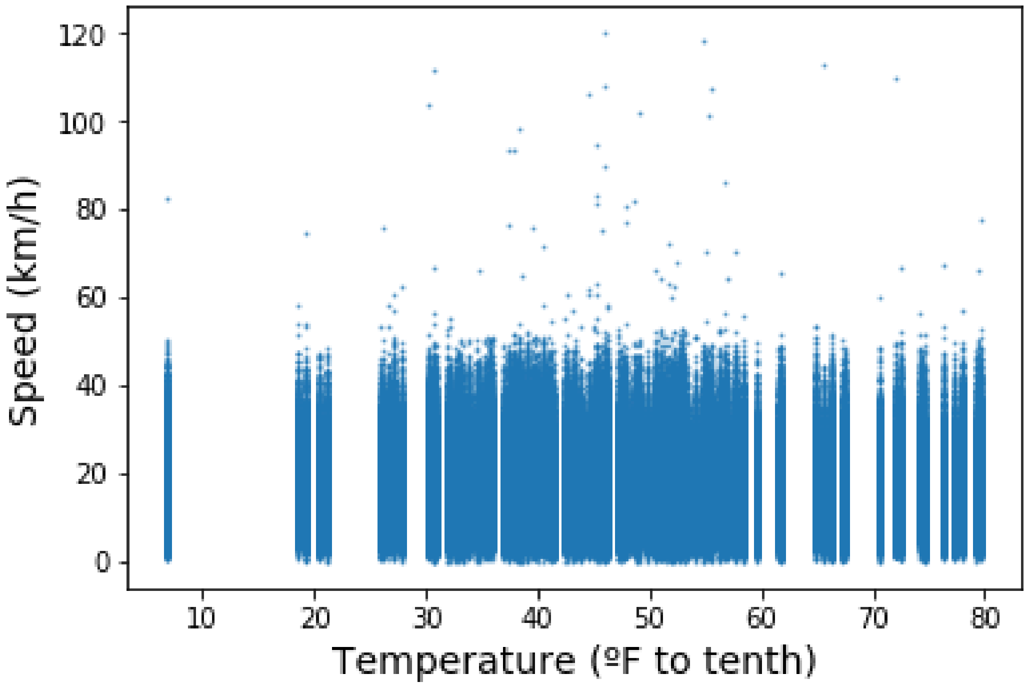
2.2.17 Trip duration vs. Snow ice pellets

```
In [90]: dur_snow = data.groupby('snow_ice_pellets').travel_time.mean()
         sns.barplot(dur_snow.index, dur_snow.values)
         plt.xlabel('Snow ice pellets', fontsize=14)
         plt.ylabel('Trip duration (sec)', fontsize=14)
         plt.show()
```



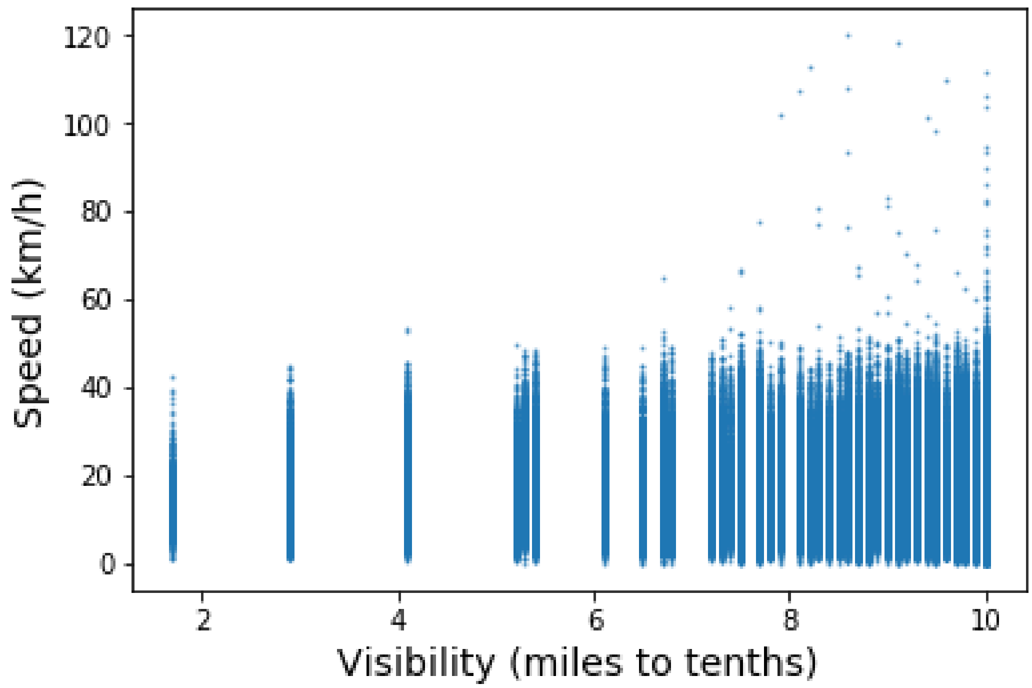
2.2.18 Speed vs. Temperature

```
In [91]: plt.scatter((data['temp']), data['speed'] , s=1, alpha=0.5)
         plt.xlabel('Temperature (žF to tenth)', fontsize=14)
         plt.ylabel('Speed (km/h)', fontsize=14)
         plt.show()
```



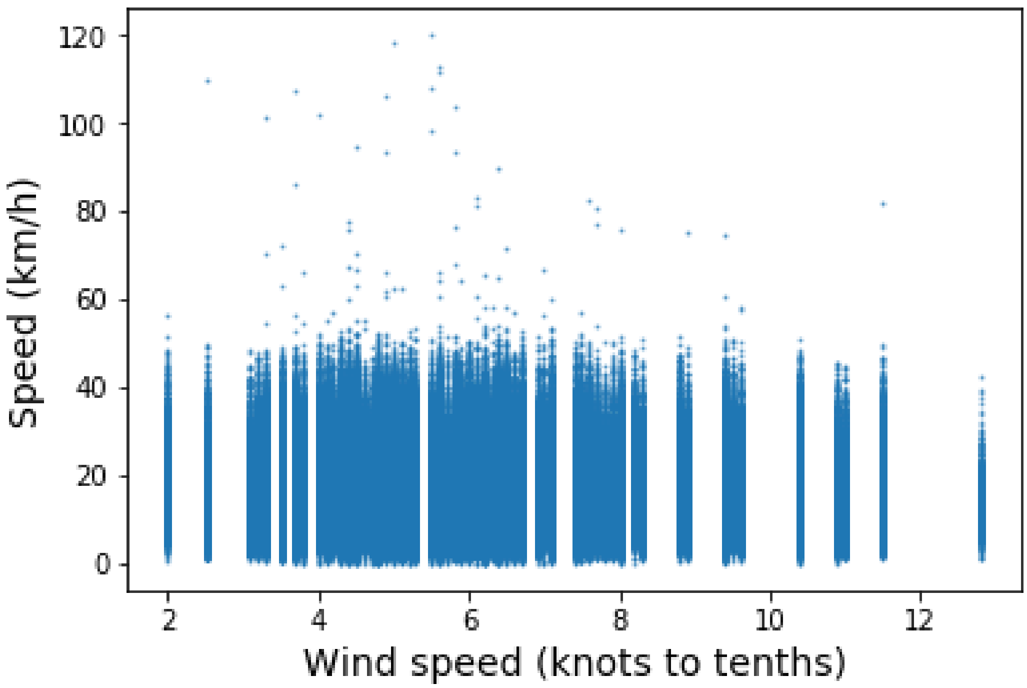
2.2.19 Speed vs. Visibility

```
In [92]: plt.scatter((data['visib']), data['speed'] , s=1, alpha=0.5)
         plt.xlabel('Visibility (miles to tenths)', fontsize=14)
         plt.ylabel('Speed (km/h)', fontsize=14)
         plt.show()
```



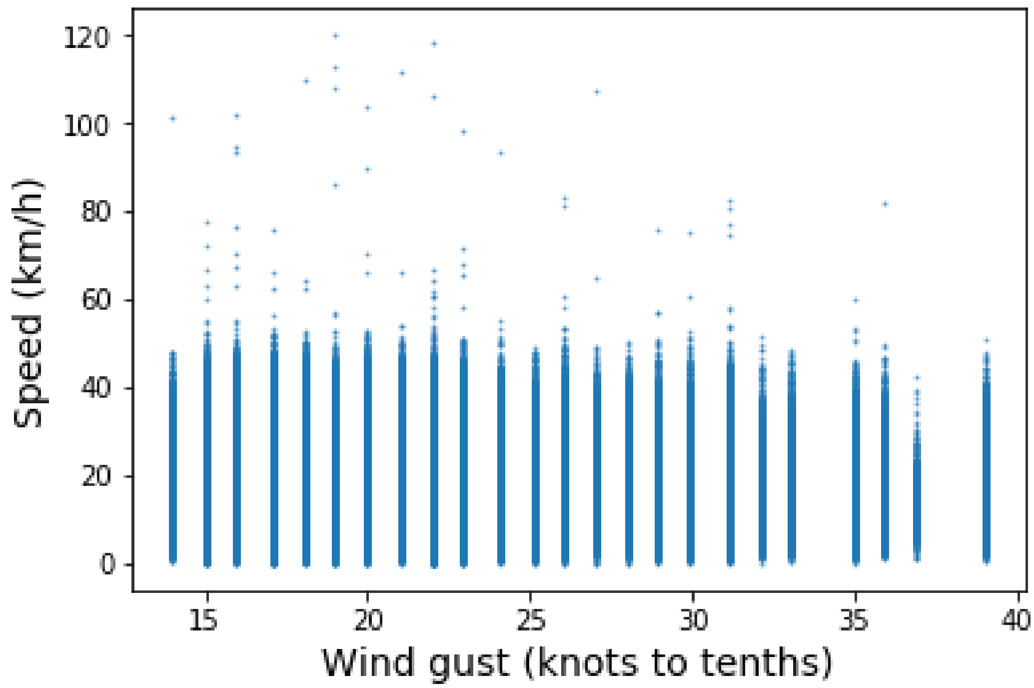
2.2.20 Speed vs. Wind speed

```
In [93]: plt.scatter((data['wdsp']), data['speed'] , s=1, alpha=0.5)
         plt.xlabel('Wind speed (knots to tenths)', fontsize=14)
         plt.ylabel('Speed (km/h)', fontsize=14)
         plt.show()
```



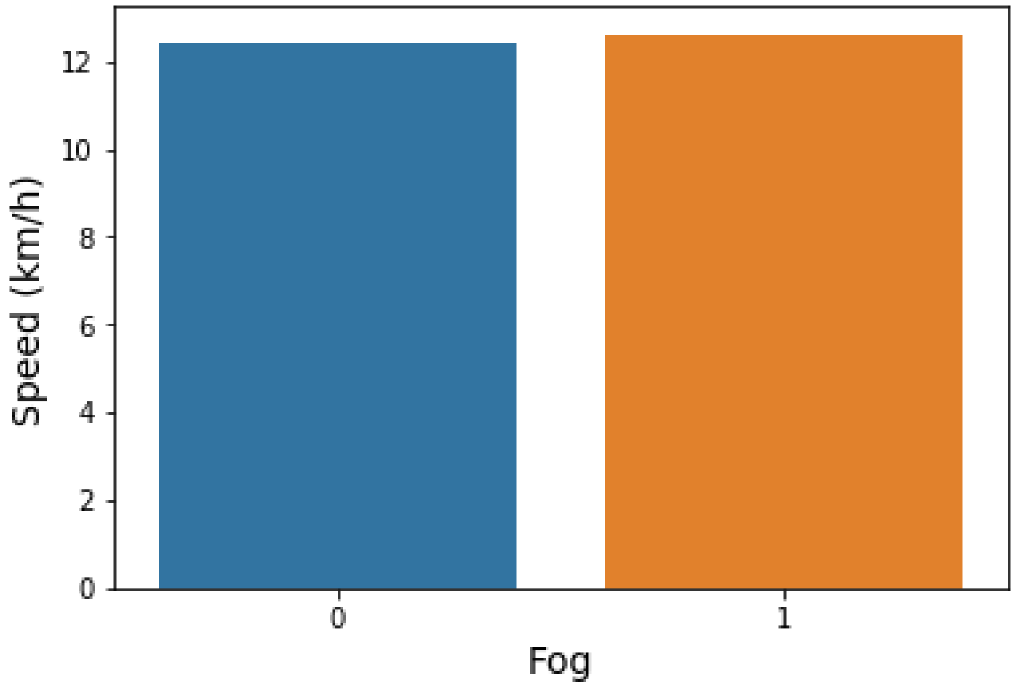
2.2.21 Speed vs. Wind gust

```
In [94]: plt.scatter((data['gust']), data['speed'] , s=1, alpha=0.5)
         plt.xlabel('Wind gust (knots to tenths)', fontsize=14)
         plt.ylabel('Speed (km/h)', fontsize=14)
         plt.show()
```



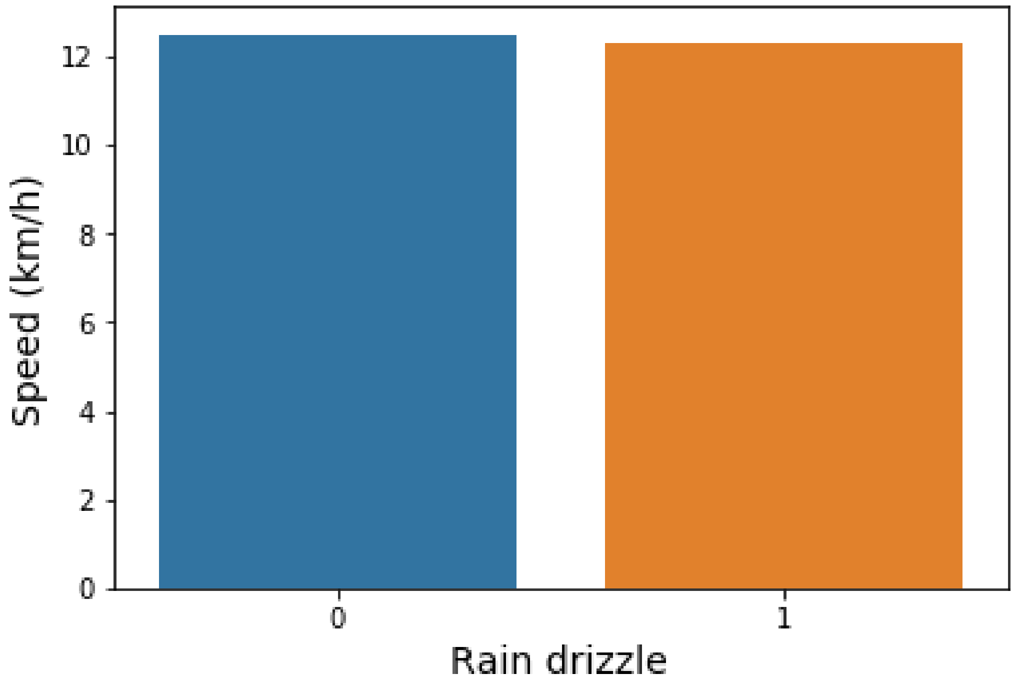
2.2.22 Speed vs. Fog

```
In [95]: sp_fog = data.groupby('fog').speed.mean()
         sns.barplot(sp_fog.index, sp_fog.values)
         plt.xlabel('Fog', fontsize=14)
         plt.ylabel('Speed (km/h)', fontsize=14)
         plt.show()
```



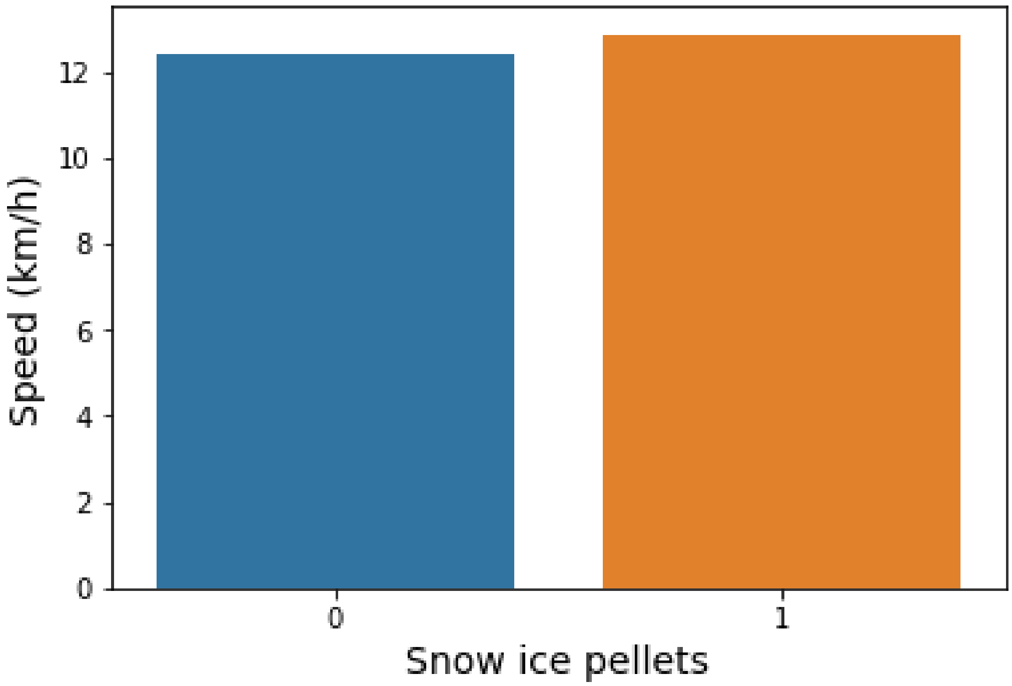
2.2.23 Speed vs. Rain drizzle

```
In [96]: sp_rain = data.groupby('rain_drizzle').speed.mean()
         sns.barplot(sp_rain.index, sp_rain.values)
         plt.xlabel('Rain drizzle', fontsize=14)
         plt.ylabel('Speed (km/h)', fontsize=14)
         plt.show()
```



2.2.24 Speed vs. Snow ice pellets

```
In [97]: sp_snow = data.groupby('snow_ice_pellets').speed.mean()
         sns.barplot(sp_snow.index, sp_snow.values)
         plt.xlabel('Snow ice pellets', fontsize=14)
         plt.ylabel('Speed (km/h)', fontsize=14)
         plt.show()
```



3 Feature selection

- Our computers all cannot run these code in python. So we have run feature selection in R.
- Chi-square: CHI-SQ computes the correlation between variable-class (v,l) using their expected and observed probabilities.
- RReliefF: RReliefF penalizes the predictors that give different values to neighbors with the same predicted values (nearHit), and rewards predictors that give different values to neighbors with different predicted values (nearMiss). Rank the features based on the their weights.

$$W_i = W_i - (x_i - \text{ nearHit }_i)^2 + (x_i - \text{ nearMiss }_i)^2$$

```
In [10]: # from sklearn.feature_selection import SelectPercentile
         # from sklearn.feature_selection import chi2

         # X=df_clean[['vendor_id','passenger_count',#'pickup_longitude', 'pickup_latitude','d
         #              'day_of_year', 'month_of_year','temp', 'visib', 'wdsp', 'gust', 'prcp',
         #              'rain_drizzle', 'snow_ice_pellets','distancce_in_km','day_of_week', 'pi
         # y=df_clean['travel_time']
         # print(X.shape,y.shape)
```

```
# mask=np.random.choice(X.shape[0],round(X.shape[0]*0.5),replace=False)
```

In [11]:
```
# chi2_feat_select=chi2(X.iloc[mask,:], y.iloc[mask,])
# X.iloc[:,chi2_feat_select[1]<0.05] # p-value of these col < 0.05
```

In [12]:
```
# from skrebate import SURF
# relief=SURF(n_features_to_select=10, n_jobs=-1,discrete_threshold=15)
# relief.fit(X.iloc[mask,:].values, y.iloc[mask,].values)
```

Selected feature sets:
`distance_in_km, pickup_hour, temp, passenger_count, day_of_week`
`pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude`

4 Model

Here we have:
(1). df_clean: cleaned data
(2). df_dummy: cleaned data after dummified

In [11]:
```
df_dummy.columns
```

Out[11]:
```
Index(['vendor_id', 'pickup_datetime', 'dropoff_datetime', 'passenger_count',
       'trip_distance', 'pickup_longitude', 'pickup_latitude', 'rate_code',
       'store_and_fwd_flag', 'dropoff_longitude', 'dropoff_latitude',
       'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount',
       'tolls_amount', 'imp_surcharge', 'total_amount', 'travel_time',
       'date_of_year', 'day_of_year', 'month_of_year', 'year_of_year',
       'date_of_year2', 'temp', 'visib', 'wdsp', 'gust', 'prcp', 'sndp', 'fog',
       'rain_drizzle', 'snow_ice_pellets', 'hail', 'thunder',
       'distancce_in_km', 'speed', 'weekday', 'day_of_week', 'pickup_hour',
       'weekday_Monday', 'weekday_Saturday', 'weekday_Sunday',
       'weekday_Thursday', 'weekday_Tuesday', 'weekday_Wednesday', 'month_2',
       'month_3', 'month_4', 'month_5', 'month_6', 'pickup_hour_1',
       'pickup_hour_2', 'pickup_hour_3', 'pickup_hour_4', 'pickup_hour_5',
       'pickup_hour_6', 'pickup_hour_7', 'pickup_hour_8', 'pickup_hour_9',
       'pickup_hour_10', 'pickup_hour_11', 'pickup_hour_12', 'pickup_hour_13',
       'pickup_hour_14', 'pickup_hour_15', 'pickup_hour_16', 'pickup_hour_17',
       'pickup_hour_18', 'pickup_hour_19', 'pickup_hour_20', 'pickup_hour_21',
       'pickup_hour_22', 'pickup_hour_23', 'flag_Y'],
      dtype='object')
```

In [12]:
```
X=df_clean[['passenger_count', 'pickup_longitude', 'pickup_latitude','dropoff_longitu
            'day_of_year', 'month_of_year','temp', 'visib', 'wdsp', 'gust', 'prcp', 's
            'rain_drizzle', 'snow_ice_pellets','distancce_in_km','day_of_week', 'picku
X_fs=df_clean[['passenger_count', 'pickup_longitude', 'pickup_latitude','dropoff_longi
            'temp', 'distancce_in_km','day_of_week', 'pickup_hour']] #'vendor_id',
y=df_clean['travel_time']
```

```
X_lr=df_dummy[['passenger_count',
            'day_of_year','temp', 'visib', 'wdsp', 'gust', 'prcp', 'sndp', 'fog','rai
               'distancce_in_km',
               'weekday_Monday', 'weekday_Saturday', 'weekday_Sunday',
               'weekday_Thursday', 'weekday_Tuesday', 'weekday_Wednesday', 'month_2',
               'month_3', 'month_4', 'month_5', 'month_6', 'pickup_hour_1',
               'pickup_hour_2', 'pickup_hour_3', 'pickup_hour_4', 'pickup_hour_5',
               'pickup_hour_6', 'pickup_hour_7', 'pickup_hour_8', 'pickup_hour_9',
               'pickup_hour_10', 'pickup_hour_11', 'pickup_hour_12', 'pickup_hour_13'
               'pickup_hour_14', 'pickup_hour_15', 'pickup_hour_16', 'pickup_hour_17'
               'pickup_hour_18', 'pickup_hour_19', 'pickup_hour_20', 'pickup_hour_21'
               'pickup_hour_22', 'pickup_hour_23']]
X_lr_fs=df_dummy[['passenger_count',
            'temp', 'distancce_in_km',
                 'weekday_Monday', 'weekday_Saturday', 'weekday_Sunday',
               'weekday_Thursday', 'weekday_Tuesday', 'weekday_Wednesday',
                 'pickup_hour_1',
               'pickup_hour_2', 'pickup_hour_3', 'pickup_hour_4', 'pickup_hour_5',
               'pickup_hour_6', 'pickup_hour_7', 'pickup_hour_8', 'pickup_hour_9',
               'pickup_hour_10', 'pickup_hour_11', 'pickup_hour_12', 'pickup_hour_13'
               'pickup_hour_14', 'pickup_hour_15', 'pickup_hour_16', 'pickup_hour_17'
               'pickup_hour_18', 'pickup_hour_19', 'pickup_hour_20', 'pickup_hour_21'
               'pickup_hour_22', 'pickup_hour_23']]
```

4.1 XGBoost

In [13]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

# X_train, X_test, Y_train, Y_test = train_test_split(X,y, test_size=0.2, random_stat
# print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)

# # Grid search CV to find best parameters

# xgb_raw = XGBRegressor()
# params = {
#     'objective':['reg:squarederror'],
#     "n_estimators": [350], # 100, 150, 200, 250, 300, 350
#     "learning_rate": [0.08], # 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08
#     "max_depth": [2, 3, 4, 5, 6, 7, 8], # 2, 3, 4, 5, 6, 7, 8
#     "colsample_bytree": [0.75],
#     "gamma": [0],
#     "subsample": [0.75],
# }
# search_params_raw = GridSearchCV(xgb_raw, params, cv=3, n_jobs=1, return_train_scor
# search_params_raw.fit(X_train, y_train)
# print(search_params_raw.best_score_)
```

```
# print(search_params_raw.best_params_)
```

In [14]:
```
from Models import XGBoost

xgb_raw_pred, xgb_raw_rmse, xgb_raw_rmsle, xgb_raw_r2 = XGBoost(X, y, n_splits=5, ranc
                                                                n_estimators = 350,
                                                                learning_rate = 0.08,
                                                                gamma = 0,
                                                                subsample = 0.75,
                                                                colsample_bytree = 1,
                                                                max_depth = 2,
                                                                min_child_weight = 4,
                                                                silent = 1,
                                                                n_jobs = -1)
```

In [15]:
```
print(xgb_raw_rmse,xgb_raw_r2)
```

```
360.47448583072674 0.7319780582933644
```

In [16]:
```
# X_train_fs, X_test_fs, y_train_fs, y_test_fs = train_test_split(X_fs, y, test_size=
# print(X_train_fs.shape, X_test_fs.shape, Y_train_fs.shape, Y_test_fs.shape)

# # Grid search CV to find best parameters

# xgb_fs = XGBRegressor()
# params_fs = {
#     'objective':['reg:squarederror'],
#     "n_estimators": [350], # [100, 150, 200, 250, 300, 350]
#     "learning_rate": [0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08], # [0.02, 0.03, 0.0
#     "max_depth": [8], # [2, 3, 4, 5, 6, 7, 8]
#     "colsample_bytree": [0.75],
#     "gamma": [0],
#     "subsample": [0.75],
# }
# search_params_fs = GridSearchCV(xgb_fs, params_fs, cv=3, n_jobs=1, return_train_sco
# search_params_fs.fit(X_train_fs, y_train_fs)
# print(search_params_fs.best_score_)
# print(search_params_fs.best_params_)
```

In [17]:
```
xgb_fs_pred, xgb_fs_rmse, xgb_fs_rmsle, xgb_fs_r2 = XGBoost(X_fs, y, n_splits=5, rando
                                                            n_estimators = 350,
                                                            learning_rate = 0.08,
                                                            gamma = 0,
                                                            subsample = 0.75,
                                                            colsample_bytree = 1,
                                                            max_depth = 8,
                                                            min_child_weight = 4,
                                                            silent = 1,
                                                            n_jobs = -1)
```

```
In [18]: print(xgb_fs_rmse,xgb_fs_r2)
         # 305.65750048848906 0.8072961730522138 lr_fs
```

```
298.23508324355834 0.8165400110279851
```

4.2 kNN

```
In [19]: # # Grid search CV to find best parameters

         # classifier = GridSearchCV(KNeighborsRegressor(),param_grid = {'n_neighbors': [5,10,
         # classifier.fit(X_train,y_train)
         # print(classifier.best_params_)
```

```
In [20]: from Models import KNN

         knn_raw_pred, knn_raw_rmse, knn_raw_rmsle, knn_raw_r2 = KNN(X, y, n_splits=5, random_s
                                                                      n_neighbors=25)
```

```
In [21]: print(knn_raw_rmse,knn_raw_r2)
```

```
365.4948916896883 0.7244627671467907
```

```
In [22]: # # Grid search CV to find best parameters

         # classifier = GridSearchCV(KNeighborsRegressor(),param_grid = {'n_neighbors': [5,10,
         # classifier.fit(X_train_fs,y_train_fs)
         # print(classifier.best_params_)
```

```
In [23]: knn_fs_pred, knn_fs_rmse, knn_fs_rmsle, knn_fs_r2 = KNN(X_fs, y, n_splits=5, random_st
                                                                  n_neighbors=25)
```

```
In [24]: print(knn_fs_rmse,knn_fs_r2)
```

```
366.2463155615901 0.7233275250171606
```

4.3 Lasso

```
In [25]: from sklearn import linear_model

         # X_train_lr, X_test_lr, y_train_lr, y_test_lr = train_test_split(X_lr, y, test_size=
         # print(X_train_lr.shape, X_test_lr.shape, y_train_lr.shape, y_test_lr.shape)


         # Cs = [0.001,0.01,0.1,1,10]
         # param_grid = {'alpha': Cs}

         # classifier = GridSearchCV(linear_model.Lasso(normalize=True), param_grid, cv=3)
         # classifier.fit(X_train_lr,y_train_lr)
         # print(classifier.best_params_)
```

In [26]:
```
from Models import LASSO

lasso_raw_pred, lasso_raw_rmse, lasso_raw_rmsle, lasso_raw_r2 = LASSO(X_lr, y, n_split
                                                                        alpha=0.001)
```

In [27]:
```
print(lasso_raw_rmse,lasso_raw_r2)
```

```
419.8905933247012 0.6363468198662707
```

In [28]:
```
# X_train_lr_fs, X_test_lr_fs, y_train_lr_fs, y_test_lr_fs = train_test_split(X_lr_fs
# print(X_train_lr_fs.shape, X_test_lr_fs.shape, Y_train_lr_fs.shape, Y_test_lr_fs.sh

# Cs = [0.001,0.01,0.1,1,10]
# param_grid = {'alpha': Cs}

# classifier = GridSearchCV(linear_model.Lasso(normalize=True), param_grid, cv=3)
# classifier.fit(X_train_lr_fs,y_train_lr_fs)
# print(classifier.best_params_)
```

In [59]:
```
lasso_fs_pred, lasso_fs_rmse, lasso_fs_rmsle, lasso_fs_r2 = LASSO(X_lr_fs, y, n_split
                                                                   alpha=0.001)
```

In [60]:
```
print(lasso_fs_rmse,lasso_fs_r2)
```

```
421.4801247390257 0.6335880320081078
```

4.4 Ridge

In [31]:
```
# Cs = [0.001,0.01,0.1,1,10]
# param_grid = {'alpha': Cs}

# classifier = GridSearchCV(linear_model.Ridge(normalize=True), param_grid, cv=3)
# classifier.fit(X_train_lr,y_train_lr)
# print(classifier.best_params_)
```

In [49]:
```
from Models import RIDGE

ridge_raw_pred, ridge_raw_rmse, ridge_raw_rmsle, ridge_raw_r2 = RIDGE(X_lr, y, n_split
                                                                      alpha=0.001)
```

In [50]:
```
print(ridge_raw_rmse,ridge_raw_r2)
```

```
419.74969858444564 0.6365907961206398
```

In [34]:
```
# Cs = [0.001,0.01,0.1,1,10]
# param_grid = {'alpha': Cs}
```

```
        # classifier = GridSearchCV(linear_model.Lasso(normalize=True), param_grid, cv=3)
        # classifier.fit(X_train_lr_fs,y_train_lr_fs)
        # print(classifier.best_params_)
```

```
In [53]: ridge_fs_pred, ridge_fs_rmse, ridge_fs_rmsle, ridge_fs_r2 = RIDGE(X_lr_fs, y, n_splits
                                                                          alpha=0.001)
```

```
In [54]: print(ridge_fs_rmse,ridge_fs_r2)
```

```
421.3967216965533 0.6337329660342879
```

4.5 Ensemble model

- In this project, we combined the predictions from four methods as new features and used XGBoost to train the new created dataset and finally gave the predictions.

```
In [69]: df_1=pd.merge(pd.DataFrame(xgb_raw_pred),pd.DataFrame(knn_raw_pred),on=0)
         df_2=pd.merge(pd.DataFrame(lasso_raw_pred),pd.DataFrame(ridge_raw_pred),on=0)
         df_raw_ensem=pd.merge(df_1,df_2,on=0)
         df_raw_ensem.columns=['Index','xgb_raw_pred','knn_raw_pred','lasso_raw_pred','ridge_ra
         df_raw_ensem=df_raw_ensem.sort_values(by='Index')
         df_raw_ensem.reset_index(drop=True, inplace=True)
         df_raw_ensem=df_raw_ensem.drop(labels='Index', axis=1)
```

```
In [70]: df_raw_ensem.head(5)
```

Out[70]:

	xgb_raw_pred	knn_raw_pred	lasso_raw_pred	ridge_raw_pred
0	759.305481	705.84	693.692029	709.734022
1	869.885864	801.92	917.039577	936.026113
2	1548.565186	1463.72	1306.124646	1316.023257
3	617.186096	586.92	720.747877	682.314812
4	863.833008	859.44	694.811048	700.979425

```
In [71]: ensem_raw_pred, ensem_raw_rmse, ensem_raw_rmsle, ensem_raw_r2 = XGBoost(df_raw_ensem,
                                                                                n_estimators =
                                                                                learning_rate
                                                                                gamma = 0,
                                                                                subsample = 0
                                                                                colsample_byt
                                                                                max_depth = 2
                                                                                min_child_wei
                                                                                silent = 1,
                                                                                n_jobs = -1)
```

```
In [72]: print(ensem_raw_rmse,ensem_raw_r2)
```

```
344.82660130307227 0.7547397747480867
```

In [62]:
```
df_1=pd.merge(pd.DataFrame(xgb_fs_pred),pd.DataFrame(knn_fs_pred),on=0)
df_2=pd.merge(pd.DataFrame(lasso_fs_pred),pd.DataFrame(ridge_fs_pred),on=0)
df_fs_ensem=pd.merge(df_1,df_2,on=0)
df_fs_ensem.columns=['Index','xgb_fs_pred','knn_fs_pred','lasso_fs_pred','ridge_fs_pre
df_fs_ensem=df_fs_ensem.sort_values(by='Index')
df_fs_ensem.reset_index(drop=True, inplace=True)
df_fs_ensem=df_fs_ensem.drop(labels='Index', axis=1)
```

In [63]:
```
df_fs_ensem.head(5)
```

Out[63]:

	xgb_fs_pred	knn_fs_pred	lasso_fs_pred	ridge_fs_pred
0	994.176270	705.84	696.936224	701.097918
1	1066.803955	870.88	922.088843	928.867319
2	1627.203369	1459.32	1305.113970	1309.975545
3	460.883362	636.52	745.042086	714.373108
4	656.828430	859.44	694.250235	695.753086

In [66]:
```
ensem_fs_pred, ensem_fs_rmse, ensem_fs_rmsle, ensem_fs_r2 = XGBoost(df_fs_ensem, y, n_
                                                                   n_estimators = 350
                                                                   learning_rate = 0
                                                                   gamma = 0,
                                                                   subsample = 0.75,
                                                                   colsample_bytree =
                                                                   max_depth = 8,
                                                                   min_child_weight =
                                                                   silent = 1,
                                                                   n_jobs = -1)
```

In [113]:
```
print(ensem_fs_rmse,ensem_fs_r2)
```

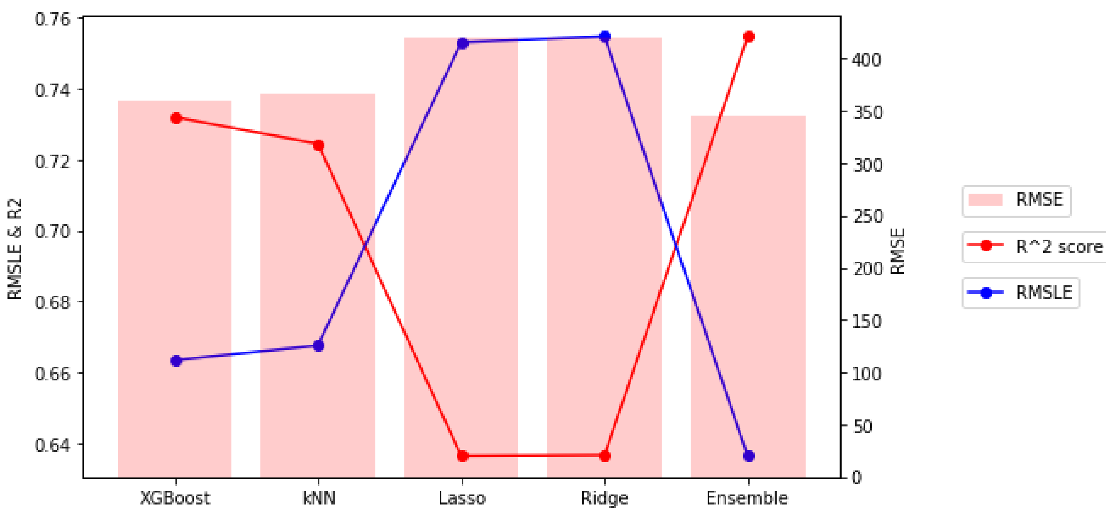
```
297.12934962718657 0.8178975533338816
```

5 Results Comparison

5.1 Raw data

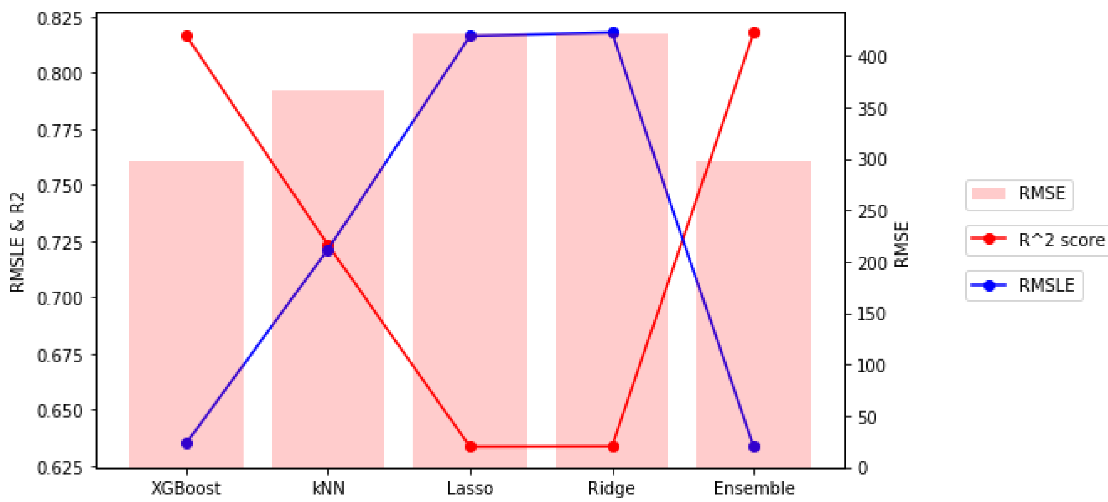
In [73]:
```
fig=plt.figure(figsize=(10,5))

all_raw_r2=[xgb_raw_r2, knn_raw_r2, lasso_raw_r2, ridge_raw_r2, ensem_raw_r2]
all_raw_rmsle=[xgb_raw_rmsle, knn_raw_rmsle, lasso_raw_rmsle, ridge_raw_rmsle, ensem_r
all_raw_rmse=[xgb_raw_rmse, knn_raw_rmse, lasso_raw_rmse, ridge_raw_rmse, ensem_raw_rm
l=[i for i in range(5)]
lx=['XGBoost','kNN','Lasso','Ridge','Ensemble']


ax1 = fig.add_subplot(111)
ax1.plot(l, all_raw_r2, 'or-', label='R^2 score')
# for i,(_x,_y) in enumerate(zip(l,all_raw_r2)):
```

```
#     plt.text(_x,_y,all_raw_r2[i],color='black',fontsize=10,)
box = ax1.get_position()
ax1.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax1.legend(loc='center left', bbox_to_anchor=(1.15, 0.5))
ax1.set_ylabel('RMSLE & R2');
# ax1.set_ylim([0.7, 1])

ax2 = ax1.twinx()
ax2.set_yticks([])
ax2.plot(l, all_raw_rmsle, 'ob-', label='RMSLE')
box = ax2.get_position()
ax2.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax2.legend(loc='center left', bbox_to_anchor=(1.15, 0.4))

ax3 = ax1.twinx()
plt.bar(l,all_raw_rmse,alpha=0.2,color='r',label='RMSE')
box = ax3.get_position()
ax3.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax3.legend(loc='center left', bbox_to_anchor=(1.15, 0.6))
ax3.set_ylabel('RMSE');
# ax3.set_ylim([0, 400])


plt.xticks(l,lx)


plt.show()
```



5.2 After feature selection

In [74]:
```
fig=plt.figure(figsize=(10,5))

all_fs_r2=[xgb_fs_r2, knn_fs_r2, lasso_fs_r2, ridge_fs_r2, ensem_fs_r2]
all_fs_rmsle=[xgb_fs_rmsle, knn_fs_rmsle, lasso_fs_rmsle, ridge_fs_rmsle, ensem_fs_rms
all_fs_rmse=[xgb_fs_rmse, knn_fs_rmse, lasso_fs_rmse, ridge_fs_rmse, ensem_fs_rmse]
l=[i for i in range(5)]
lx=['XGBoost','kNN','Lasso','Ridge','Ensemble']


ax1 = fig.add_subplot(111)
ax1.plot(l, all_fs_r2, 'or-', label='R^2 score')
# for i,(_x,_y) in enumerate(zip(l,all_fs_r2)):
#     plt.text(_x,_y,all_fs_r2[i],color='black',fontsize=10,)
box = ax1.get_position()
ax1.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax1.legend(loc='center left', bbox_to_anchor=(1.15, 0.5))
ax1.set_ylabel('RMSLE & R2');
# ax1.set_ylim([0.7, 1])

ax2 = ax1.twinx()
ax2.set_yticks([])
ax2.plot(l, all_fs_rmsle, 'ob-', label='RMSLE')
box = ax2.get_position()
ax2.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax2.legend(loc='center left', bbox_to_anchor=(1.15, 0.4))

ax3 = ax1.twinx()
plt.bar(l,all_fs_rmse,alpha=0.2,color='r',label='RMSE')
box = ax3.get_position()
ax3.set_position([box.x0, box.y0, box.width * 0.8, box.height])
ax3.legend(loc='center left', bbox_to_anchor=(1.15, 0.6))
ax3.set_ylabel('RMSE');
# ax3.set_ylim([0, 400])


plt.xticks(l,lx)


plt.show()
```



Results Comparison

- From this, we found after feature selection XGBoost does improve its performance, for example, RMSE reducing from 350+ to 300. This suggests feature selection can remove redundant features and reduce the dimensionality of the data, resulting in better predictive ability.
- Among five machine learning methods, XGBoost and Ensemble model are the best and Ensemble model does a better job than XGBoost before feature selection. This data is highly unstructured real-world data and so many outliers and fake records that we might not have cleaned. Therefore, the linear models did not have a good fit on this data. kNN as a non-parametric method performed slightly better than linear models but still not good. The reason why it happens may arise from the neighbors in this data set are not really similar to the sample as some of them might be outliers or fake records. If there should be more time, we might try weighted kNN where neighbors will be weighted by some functions (for example, triangular function or beta function) to see if weighted kNN would perform better.
- XGBoost itself is actually an ensemble learning as it combines many weak tree regressors together and finally gives the predictions. Therefore, it is presumable and expectable that XGBoost performed well, let alone real ensemble learning method. In this project, we combined the predictions from four methods as new features and used XGBoost to train the new created dataset and finally gave the predictions. The results of ensemble model were also desirable.

6 Test

In [70]:
```
import pandas as pd
import os
pd.set_option('display.max_columns', 500)
os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="../ecbm4040-yt2639-d3ee184230ba.json"
from google.cloud import bigquery
client = bigquery.Client()
```

```
# # First we uploaded the test set to big-query and query it with GSOD 2015 data toge
# # Get test data joint with weather data

# query = (
#     """
#     SELECT * FROM
#     (
#     SELECT *, EXTRACT (DATE FROM pickup_datetime) as date_of_year
#     FROM `ecbm4040-yt2639.APM4990_final_test_data.final_data_2015`) a
#     INNER JOIN
#     (
#      select concat(year,'-',mo,'-',da) as date_of_year2,temp,visib,wdsp,gust,max,mi
#      from `bigquery-public-data.noaa_gsod.gsod2015` where stn='725053'
#     ) weather_data
#     on CAST(a.date_of_year AS STRING) = weather_data.date_of_year2
#     """
# )
# df_test_weather=pd.io.gbq.read_gbq(query,dialect='standard')
# print(df_test_weather.shape)
```

```
(695890, 30)
```

In [31]:
```
# # save the queried data
# df_test_weather.to_csv('test_weather.csv')
```

In [75]:
```
df_test_weather=pd.read_csv('../Data/test_weather.csv') # this is test data joint wit

df_test_weather['distancce_in_km'] = HAVERSINE(df_test_weather.pickup_latitude, df_tes

# Add a new column indicating weekday
df_test_weather['pickup_datetime'] = pd.to_datetime(df_test_weather['pickup_datetime']
df_test_weather['weekday'] = df_test_weather.pickup_datetime.dt.weekday_name
df_test_weather['day_of_week'] = df_test_weather.pickup_datetime.dt.weekday

# Add pick-up hour
df_test_weather['pickup_hour'] = df_test_weather.pickup_datetime.dt.hour

# Because the pickup location and dropoff location are the same
# So we replace NaN with 0, meaning the distance is 0
df_test_weather[df_test_weather['distancce_in_km'].isnull()]=0

###### dummyfi
# Weekday
dummy = pd.get_dummies(df_test_weather['weekday'], prefix='weekday')
dummy.drop(dummy.columns[0], axis=1, inplace=True) #avoid dummy trap
df_test_weather_dummy = pd.concat([df_test_weather,dummy], axis = 1)
```

```
# pickup hour
dummy = pd.get_dummies(df_test_weather['pickup_hour'], prefix='pickup_hour')
dummy.drop(dummy.columns[0], axis=1, inplace=True) #avoid dummy trap
df_test_weather_dummy = pd.concat([df_test_weather_dummy,dummy], axis = 1)

df_test_weather.head(5)
```

Out[75]:

```
   Unnamed: 0            pickup_datetime  pickup_latitude  pickup_longitude  \
0           0  2015-01-26 08:07:49+00:00        40.777702        -73.960938
1           1  2015-02-16 12:48:57+00:00        40.748890        -73.977600
2           2  2015-08-29 01:41:34+00:00        40.775406        -73.953438
3           3  2015-02-15 15:40:17+00:00        40.729767        -73.978523
4           4  2015-01-26 14:46:28+00:00        40.776939        -73.949486

   dropoff_latitude  dropoff_longitude  passenger_count date_of_year  \
0         40.783848         -73.956650                0   2015-01-26
1         40.743141         -73.978020                0   2015-02-16
2         40.775402         -73.953445                0   2015-08-29
3         40.730019         -73.979118                0   2015-02-15
4         40.779545         -73.955856                0   2015-01-26

  date_of_year2  temp  ...   min  prcp   sndp  fog  rain_drizzle  \
0    2015-01-26  26.5  ...  21.2  0.00    2.0    1             0
1    2015-02-16   9.8  ...   3.0  0.00    9.1    0             0
2    2015-08-29  75.7  ...  63.0  0.00  999.9    0             0
3    2015-02-15  18.9  ...  10.0  0.02    7.9    0             0
4    2015-01-26  26.5  ...  21.2  0.00    2.0    1             0

   snow_ice_pellets  distancce_in_km   weekday  day_of_week  pickup_hour
0                 1         0.771807    Monday            0            8
1                 0         0.639345    Monday            0           12
2                 0         0.000764  Saturday            5            1
3                 1         0.057352    Sunday            6           15
4                 1         0.608824    Monday            0           14

[5 rows x 24 columns]
```

In [76]:

```
df_test_weather_dummy.head(5)
```

Out[76]:

```
   Unnamed: 0            pickup_datetime  pickup_latitude  pickup_longitude  \
0           0  2015-01-26 08:07:49+00:00        40.777702        -73.960938
1           1  2015-02-16 12:48:57+00:00        40.748890        -73.977600
2           2  2015-08-29 01:41:34+00:00        40.775406        -73.953438
3           3  2015-02-15 15:40:17+00:00        40.729767        -73.978523
4           4  2015-01-26 14:46:28+00:00        40.776939        -73.949486

   dropoff_latitude  dropoff_longitude  passenger_count date_of_year  \
```

```
0         40.783848        -73.956650                0   2015-01-26
1         40.743141        -73.978020                0   2015-02-16
2         40.775402        -73.953445                0   2015-08-29
3         40.730019        -73.979118                0   2015-02-15
4         40.779545        -73.955856                0   2015-01-26

  date_of_year2  temp  ...  pickup_hour_14  pickup_hour_15  pickup_hour_16  \
0    2015-01-26  26.5  ...               0               0               0
1    2015-02-16   9.8  ...               0               0               0
2    2015-08-29  75.7  ...               0               0               0
3    2015-02-15  18.9  ...               0               1               0
4    2015-01-26  26.5  ...               1               0               0

   pickup_hour_17  pickup_hour_18  pickup_hour_19  pickup_hour_20  \
0               0               0               0               0
1               0               0               0               0
2               0               0               0               0
3               0               0               0               0
4               0               0               0               0

   pickup_hour_21  pickup_hour_22  pickup_hour_23
0               0               0               0
1               0               0               0
2               0               0               0
3               0               0               0
4               0               0               0

[5 rows x 54 columns]
```

In [77]:
```
df_test_weather_dummy.columns
```

Out[77]:
```
Index(['Unnamed: 0', 'pickup_datetime', 'pickup_latitude', 'pickup_longitude',
       'dropoff_latitude', 'dropoff_longitude', 'passenger_count',
       'date_of_year', 'date_of_year2', 'temp', 'visib', 'wdsp', 'gust', 'max',
       'min', 'prcp', 'sndp', 'fog', 'rain_drizzle', 'snow_ice_pellets',
       'distancce_in_km', 'weekday', 'day_of_week', 'pickup_hour',
       'weekday_Friday', 'weekday_Monday', 'weekday_Saturday',
       'weekday_Sunday', 'weekday_Thursday', 'weekday_Tuesday',
       'weekday_Wednesday', 'pickup_hour_1', 'pickup_hour_2', 'pickup_hour_3',
       'pickup_hour_4', 'pickup_hour_5', 'pickup_hour_6', 'pickup_hour_7',
       'pickup_hour_8', 'pickup_hour_9', 'pickup_hour_10', 'pickup_hour_11',
       'pickup_hour_12', 'pickup_hour_13', 'pickup_hour_14', 'pickup_hour_15',
       'pickup_hour_16', 'pickup_hour_17', 'pickup_hour_18', 'pickup_hour_19',
       'pickup_hour_20', 'pickup_hour_21', 'pickup_hour_22', 'pickup_hour_23'],
      dtype='object')
```

In [78]:
```
X_bq_fs=df_test_weather[['passenger_count', 'pickup_longitude', 'pickup_latitude','dro
            'temp', 'distancce_in_km','day_of_week', 'pickup_hour']]
```

```
X_bq_lr_fs=df_test_weather_dummy[['passenger_count',
            'temp', 'distancce_in_km',
                 'weekday_Monday', 'weekday_Saturday', 'weekday_Sunday',
               'weekday_Thursday', 'weekday_Tuesday', 'weekday_Wednesday',
                 'pickup_hour_1',
               'pickup_hour_2', 'pickup_hour_3', 'pickup_hour_4', 'pickup_hour_5',
               'pickup_hour_6', 'pickup_hour_7', 'pickup_hour_8', 'pickup_hour_9',
               'pickup_hour_10', 'pickup_hour_11', 'pickup_hour_12', 'pickup_hour_13'
               'pickup_hour_14', 'pickup_hour_15', 'pickup_hour_16', 'pickup_hour_17'
               'pickup_hour_18', 'pickup_hour_19', 'pickup_hour_20', 'pickup_hour_21'
               'pickup_hour_22', 'pickup_hour_23']]
```

In [80]:
```
# Get saved model
import pickle

with open('saved/xgb.pickle', 'rb') as f:
    xgb = pickle.load(f)
with open('saved/knn.pickle', 'rb') as f:
    knn = pickle.load(f)
with open('saved/lasso.pickle', 'rb') as f:
    lasso = pickle.load(f)
with open('saved/ridge.pickle', 'rb') as f:
    ridge = pickle.load(f)
with open('saved/ensem.pickle', 'rb') as f:
    ensem = pickle.load(f)
```

In [81]:
```
# get prediction
test_xgb_pred=xgb.predict(X_bq_fs)
test_knn_pred=knn.predict(X_bq_fs)
test_lasso_pred=lasso.predict(X_bq_lr_fs)
test_ridge_pred=ridge.predict(X_bq_lr_fs)
```

In [98]:
```
# create ensemble data for ensemble model prediction
df_1=pd.concat([pd.DataFrame(test_xgb_pred),pd.DataFrame(test_knn_pred)],axis=1)
df_2=pd.concat([pd.DataFrame(test_lasso_pred),pd.DataFrame(test_ridge_pred)],axis=1)
df_test_fs_ensem=pd.concat([df_1,df_2],axis=1)
df_test_fs_ensem.columns=['xgb_fs_pred','knn_fs_pred','lasso_fs_pred','ridge_fs_pred']
```

In [99]:
```
# get ensemble model prediction
df_test_pred=ensem.predict(df_test_fs_ensem)
df_test_pred.shape
```

In [108]:
```
test_set=pd.read_csv('../Data/APM4990_final_test_data_filtered.csv')
final=pd.concat([test_set,pd.DataFrame(df_test_pred)],axis=1)
final=final.rename(columns={0: 'predictions'})
```

In [112]:
```
final.to_csv('../prediction/Final_predictions.csv')
```