Entrée [139]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Loading Datasets

Entrée [2]:
```
df_cities = pd.read_csv('./data/cities.csv')
df_providers = pd.read_csv('./data/providers.csv')
df_station = pd.read_csv('./data/stations.csv')
df_ticket = pd.read_csv('./data/ticket_data.csv')
```

Explore Datasets

Cities DataFrame

Entrée [3]:
```
df_cities.head(5)
```

Out[3]:

	id	local_name	unique_name	latitude	longitude	population
0	5159	Padua, Veneto, Italia	padua	45.406435	11.876761	209678.0
1	76	Barcelona, Cataluña, España	barcelona	41.385064	2.173404	1611822.0
2	81	Basel, Basel-Stadt, Schweiz	basel	47.593437	7.619812	NaN
3	259	Erlangen, Bayern, Deutschland	erlangen	49.589674	11.011961	105412.0
4	11979	Balș, Olt, România	balș	44.353354	24.095672	NaN

Entrée [76]:
```
print('Missing values (%) : ')

M1 =pd.DataFrame((df_cities.isnull().sum()/df_cities.shape[0])*100).transpose()
M1.style.applymap(lambda x: 'background-color : RED' if x>0 else 'background-colo
```

```
Missing values (%) :
```

Out[76]:

	id	local_name	unique_name	latitude	longitude	population
0	0	0	0.0124378	0	0	95.4104

There is alot of missing values within the population column (more than 95%). To solve this problem we can :

* **Drop** Population Column from the DataFrame.
* **Scrap** Population for each city.

I tried with the second option (you can find the scraper script in mygithub repository)

Entrée [290]:
```
df_new_cities = pd.read_csv('./data/new_cities_csv.csv')
df_new_cities.sample(10)
```

Out[290]:

	id	local_name	unique_name	latitude	longitude	population
1704	5927	Brionne, Normandie, France	brionne	49.196450	0.712040	4326.0
6259	6659	Frutigen, Berne, Schweiz	frutigen	46.588600	7.651400	6682.0
1271	1818	Kharkiv, Kharkiv, Ukraine	kharkiv	49.990279	36.230389	1419000.0
1133	6377	Deinze, Vlaanderen, Belgique	deinze	50.978200	3.534900	NaN
6465	7850	Munster, Grand-Est, France	munster-france	48.915560	6.905830	4603.0
7393	9527	Chessy (Rhône), Auvergne-Rhône-Alpes, France	chessy-	45.885620	4.622560	1982.0
6870	8421	Saint-Antoine-de-Breuilh, Nouvelle-Aquitaine, ...	saint-antoine-de-breuilh	44.846630	0.152380	1932.0
5976	5866	Bourg-en-Bresse, Auvergne-Rhône-Alpes, France	bourg-en-bresse	46.200270	5.215130	40819.0
0	5159	Padua, Veneto, Italia	padua	45.406435	11.876761	209829.0
2076	7942	Nyons, Auvergne-Rhône-Alpes, France	nyons	44.360400	5.140000	6690.0

Entrée [291]:
```
print('Missing values (%) : ')

M2 =pd.DataFrame((df_new_cities.isnull().sum()/df_new_cities.shape[0])*100).trar
M2.style.applymap(lambda x: 'background-color : RED' if x>0 else 'background-col
```

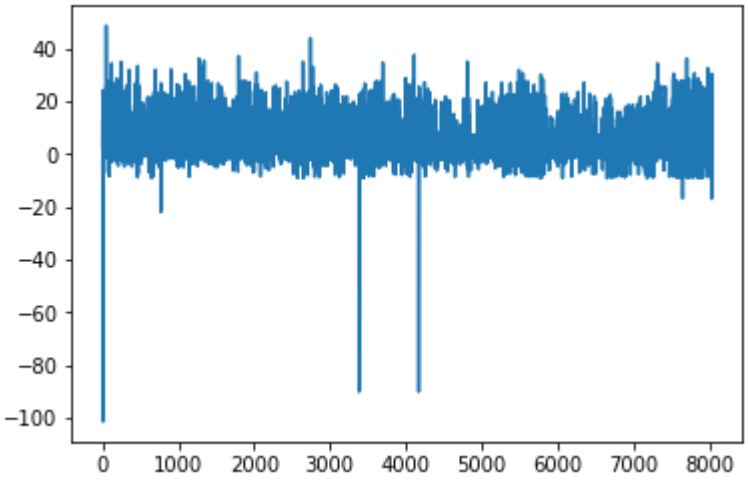
```
Missing values (%) :
```

Out[291]:

	id	local_name	unique_name	latitude	longitude	population
0	0	0	0.0124378	0	0	34.6891

Entrée [5]:
```
plt.plot(df_cities.longitude)
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x1fe955ae0b8>]
```



Entrée [311]:
```
import folium

m3 = folium.Map(location=[45.7020953,6.0528321], tiles='openstreetmap', zoom_sta

for i in range(0,len(df_cities)):
    folium.Circle(location=[df_cities.iloc[i]['latitude'], df_cities.iloc[i]['lo
m3.save('map3.html')
```

Entrée [312]:
```
## distribution of cities
m3
```

Out[312]:

Providers DataFrame

Entrée [6]:
```
df_providers.head()
```

Out[6]:

	id	company_id	provider_id	name	fullname	has_wifi	has_plug	has_adjustable_seat
0	9	1	NaN	ouibus	Ouibus	True	True	Tru
1	10	2	NaN	deinbus	Deinbus.de	False	False	Fals
2	11	3	NaN	infobus	Infobus	False	False	Fals
3	12	4	NaN	studentAgency	Student Agency	False	False	Fals
4	13	5	NaN	flixbus	Flixbus	True	False	Fals

Entrée [9]:
```
df_providers.shape
```

Out[9]:
```
(227, 10)
```

Entrée [74]:
```
print('Number of missing values:')
M1 =pd.DataFrame(df_providers.isnull().sum()).transpose()
M1.style.applymap(lambda x: 'background-color : RED' if x>0 else 'background-colo
```

```
Number of missing values:
```

Out[74]:

	id	company_id	provider_id	name	fullname	has_wifi	has_plug	has_adjustable_seats	has_bic
0	0	0	14	0	0	3	3	3	

Station DataFrame

Entrée [15]:
```
df_station.head()
```

Out[15]:

	id	unique_name	latitude	longitude
0	1	Aalen (Stuttgarter Straße)	48.835296	10.092956
1	2	Aéroport Bordeaux-Mérignac	44.830226	-0.700883
2	3	Aéroport CDG	49.009900	2.559310
3	4	Aéroport de Berlin-Schönefeld	52.389446	13.520345
4	5	Aéroport de Dresden	51.123604	13.764737

Entrée [72]:
```
print('There is no missing values')
M1 =pd.DataFrame((df_station.isnull().sum())).transpose()
M1.style.applymap(lambda x: 'background-color : RED' if x>0 else 'background-colo
```

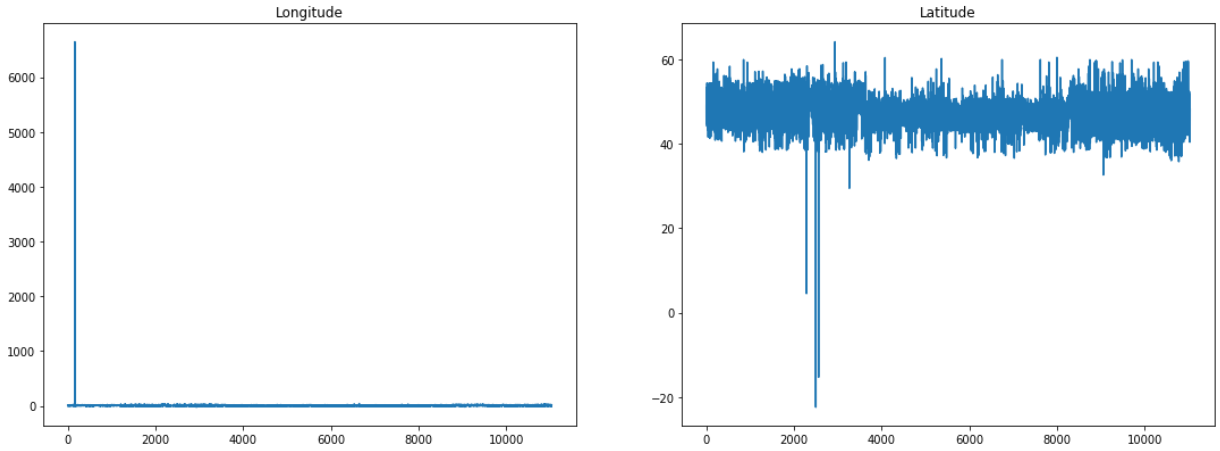
```
There is no missing values
```

Out[72]:

	id	unique_name	latitude	longitude
0	0	0	0	0

Entrée [28]:
```
fig, axes = plt.subplots(1,2)
fig.set_size_inches(18.5, 6.5)

axes[0].plot(df_station.longitude)
axes[0].set_title('Longitude')

axes[1].plot(df_station.latitude)
axes[1].set_title('Latitude')
```

Out[28]:
```
Text(0.5, 1.0, 'Latitude')
```



Longitude column contains outliers. we need to correct them

Entrée [37]:
```
df_station[df_station.longitude >1000]
```

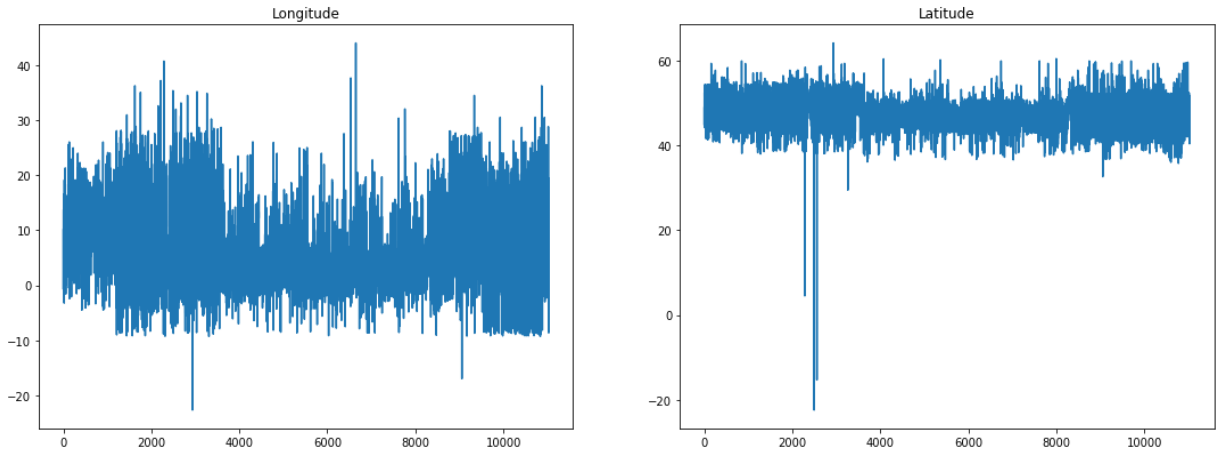
Out[37]:

	id	unique_name	latitude	longitude
161	162	Combloux - Office du tourisme	45.894811	6645.0

Entrée [38]:
```
df_station.at[df_station['id']==162 , 'longitude' ] = 6.645
```

Entrée [39]:
```
fig, axes = plt.subplots(1,2)
fig.set_size_inches(18.5, 6.5)

axes[0].plot(df_station.longitude)
axes[0].set_title('Longitude')

axes[1].plot(df_station.latitude)
axes[1].set_title('Latitude')
```

Out[39]: Text(0.5, 1.0, 'Latitude')



Tickets DataFrame

Entrée [293]:
```
df_ticket.head().style.background_gradient(cmap='Blues' ,subset=['price_in_cents
```

Out[293]:

	id	company	o_station	d_station	departure_ts	arrival_ts	price_in_cents	search_
0	6795025	8385	nan	nan	2017-10-13 14:00:00+00	2017-10-13 20:10:00+00	4550	2017-10-00:13:31.327+
1	6795026	9	63	1044	2017-10-13 13:05:00+00	2017-10-14 06:55:00+00	1450	2017-10-00:13:35.773+
2	6795027	8377	5905	6495	2017-10-13 13:27:00+00	2017-10-14 21:24:00+00	7400	2017-10-00:13:40.212+
3	6795028	8377	5905	6495	2017-10-13 13:27:00+00	2017-10-14 11:02:00+00	13500	2017-10-00:13:40.213+
4	6795029	8381	5905	6495	2017-10-13 21:46:00+00	2017-10-14 19:32:00+00	7710	2017-10-00:13:40.213+

Entrée [68]:
```
print('Missing Values(%):')
M1 =pd.DataFrame((df_ticket.isnull().sum()/df_ticket.shape[0])*100).transpose()
M1.style.applymap(lambda x: 'background-color : RED' if x>0 else 'background-colo
```

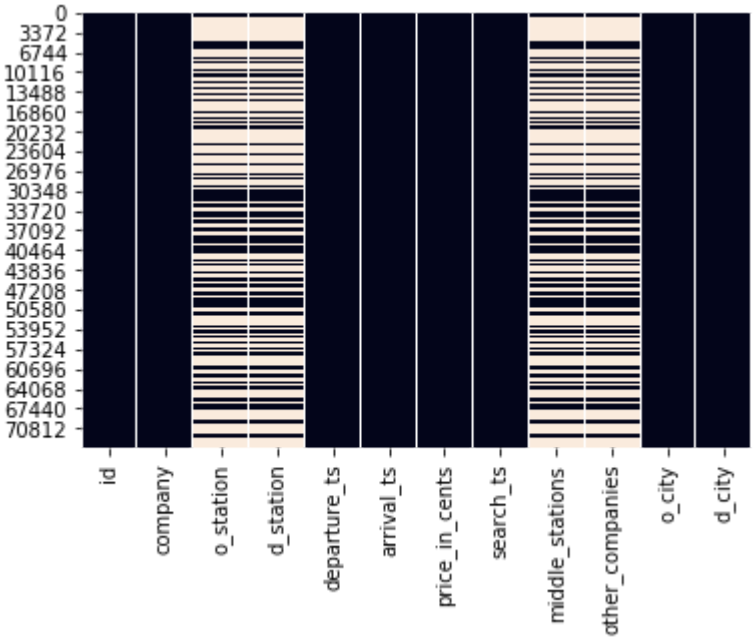
```
Missing Values(%):
```

Out[68]:

	id	company	o_station	d_station	departure_ts	arrival_ts	price_in_cents	search_ts	middle_st
0	0	0	55.8745	55.8745	0	0	0	0	5

Entrée [81]:
```
import seaborn as sns
sns.heatmap(df_ticket.isnull(), cbar=False)
```

Out[81]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fe9ed78860>
```



There is no need to drop all rows that contains missing values (we can replace o_station and d_station with o_city and d_city

Calculate disatance

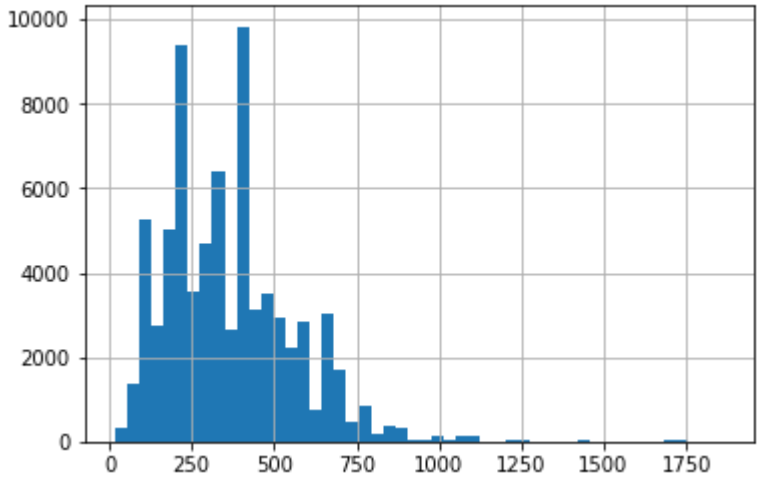
We need to calculate the distance for each trip. The distance between two point is given by this formula

Entrée [83]:
```
import math
from math import radians, sin, cos, acos

def calc_distance(lat_A, lon_A, lat_B, lon_B):
    """Retourne la distance en mètres entre les 2 points A et B connus grâce à
       leurs coordonnées(latitude et longitude).
    """
    RT= 6371.0 #rayon de la terre

    a = sin(radians(lat_A)) * sin(radians(lat_B))
    b = cos(radians(lat_A)) * cos(radians(lat_B))
    c = cos(radians(lon_A) - radians(lon_B))

    S= acos(a + b*c)
    return (RT * S)

def getCoords(Id1, df):
    '''Retourne les coordonnées d'un ville ou bien d'une station'''
    return (df[df['id'] == Id1].latitude.values[0] , df[df['id']==Id1].longitude.

def Trip_distance(row):
    ''' Retourner la distance d'un voyage  '''
    total_distance = 0
    try:
        start_point_coords = getCoords(float(row['o_station']),df_station)
        destination_point_coords = getCoords(float(row['d_station']),df_station)
    except :
        start_point_coords = getCoords(float(row['o_city']), df_cities)
        destination_point_coords = getCoords(float(row['d_city']), df_cities)

    try :
        transit_points_coords = [getCoords(float(x),df_station) for x in row['mid
    except:
        transit_points_coords = []

    try:
        start_point = start_point_coords
        for transit_point in transit_points_coords:
            total_distance += calc_distance(start_point[0],start_point[1],transit
            start_point = transit_point

        total_distance += calc_distance(start_point[0],start_point[1] , destinati
    except ValueError:
        total_distance = -1

    return total_distance
```

Entrée [84]:
```
df_ticket["trip_distance"] = df_ticket.apply(Trip_distance, axis=1)
```

Entrée [86]:
```
df_ticket.trip_distance.hist(bins=50)
```

Out[86]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fe9898e898>
```



Calculate Trip Duration

Entrée [102]:
```
from dateutil.parser import parse
import datetime

def parse2timestamp(datetime):
    return parse(datetime).timestamp()

df_ticket["trip_duration"] = df_ticket.apply(lambda row: parse2timestamp(row["ar

def secToHour(secs):
    time = datetime.timedelta(seconds = secs)
    return time
```

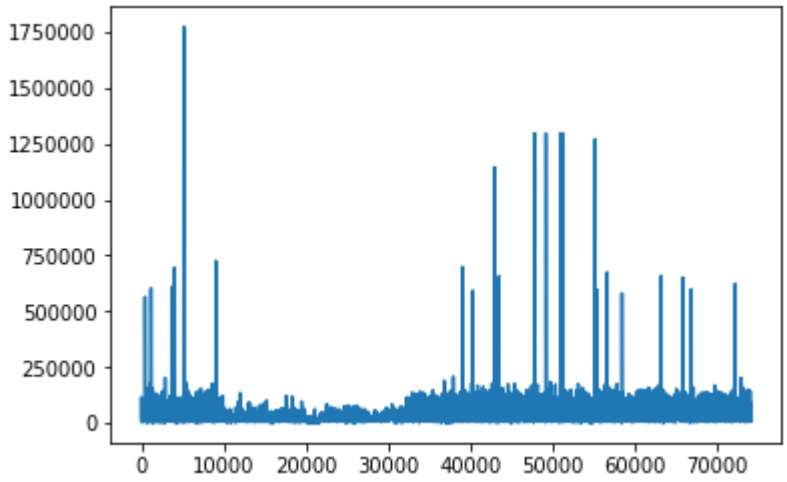
Entrée [98]:
```
df_ticket.head().style.format({"price_in_cents": "${:20,.0f}"})\
                      .background_gradient(cmap='Blues' ,subset=['price_in_cents
```

Out[98]:

	id	company	o_station	d_station	departure_ts	arrival_ts	price_in_cents	search_
0	6795025	8385	nan	nan	2017-10-13 14:00:00+00	2017-10-13 20:10:00+00	$ 4,550	2017-10-00:13:31.327+
1	6795026	9	63	1044	2017-10-13 13:05:00+00	2017-10-14 06:55:00+00	$ 1,450	2017-10-00:13:35.773+
2	6795027	8377	5905	6495	2017-10-13 13:27:00+00	2017-10-14 21:24:00+00	$ 7,400	2017-10-00:13:40.212+
3	6795028	8377	5905	6495	2017-10-13 13:27:00+00	2017-10-14 11:02:00+00	$ 13,500	2017-10-00:13:40.213+
4	6795029	8381	5905	6495	2017-10-13 21:46:00+00	2017-10-14 19:32:00+00	$ 7,710	2017-10-00:13:40.213+

Entrée [108]:
```
plt.plot(df_ticket['trip_duration'])
```

Out[108]:
```
[<matplotlib.lines.Line2D at 0x1fe9f5a2048>]
```



Entrée [133]:
```
outliers_duration = df_ticket[df_ticket.trip_duration > 500000]
print('on a plus de '+ str(outliers_duration.shape[0]) +' valeurs à vérifier')
```

```
on a plus de 54 valeurs à vérifier
```

Entrée [140]:
```
for idx , row in outliers_duration.iterrows():
    outliers_duration.at[outliers_duration['id']==row['id'] , 'time_travel' ] =
    outliers_duration.at[outliers_duration['id']==row['id'] , 'dep_city' ] = df_
    outliers_duration.at[outliers_duration['id']==row['id'] , 'des_city' ] = df_
```

Entrée [141]:
```
outliers_duration[['time_travel','dep_city','des_city']].head(5)
```

Out[141]:

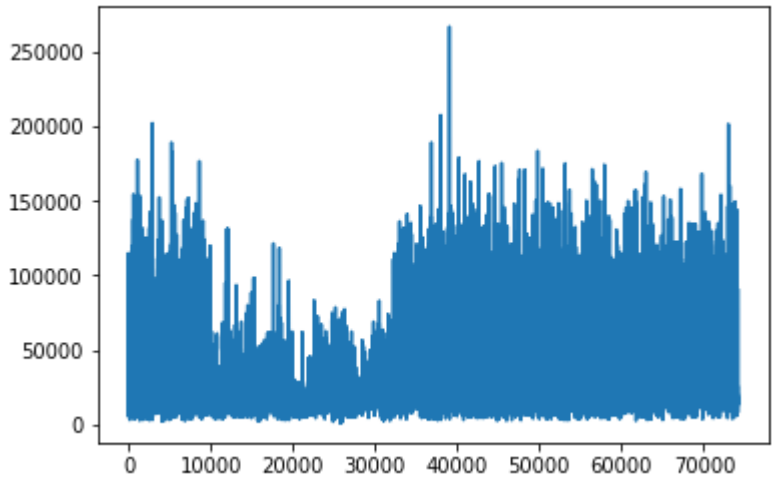
	time_travel	dep_city	des_city
463	6 days 12:51:00	Bordeaux, Nouvelle-Aquitaine, France	Lille, Hauts-de-France, France
464	6 days 06:41:00	Bordeaux, Nouvelle-Aquitaine, France	Lille, Hauts-de-France, France
1189	6 days 23:25:00	Paris, Île-de-France, France	Biarritz, Nouvelle-Aquitaine, France
1190	6 days 13:40:00	Paris, Île-de-France, France	Biarritz, Nouvelle-Aquitaine, France
3730	7 days 01:15:00	Paris, Île-de-France, France	Venezia, Veneto, Italia

```
###### <font color='red'> **6 days** </font> to go from <font color='blue'>
**Bordeaux** </font> to <font color='blue'> **Lille** </font> <font color='red'>
**!!** </font> This should be a mistake xD
###### We will drop these outliers from the dataset, otherwise the statics will not be
accurate.
```

Entrée [146]:
```
df_ticket.drop(df_ticket.index[list(outliers_duration.index)], inplace=True)
```

Entrée [147]:
```
plt.plot(df_ticket['trip_duration'])
```

Out[147]:
```
[<matplotlib.lines.Line2D at 0x1fe9f8ae940>]
```



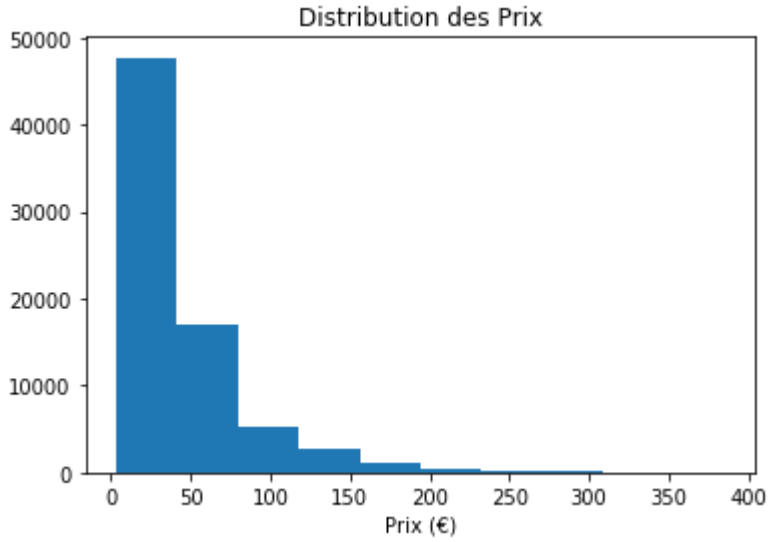
Analyze the data

Price

Entrée [148]:
```
# selectionner la colonne prix
prix = df_ticket['price_in_cents']

# statistique de base
print('Statistique des prix (sans considérer la distance)\n')
moyenne = np.mean(prix)
std = np.std(prix)
minimum = np.min(prix)
maximum = np.max(prix)

print('Prix moyen {:.2f} $'.format(moyenne/100))
print('Ecart-type {: .2f} $'.format(std/100))
print('Prix minimum {:.2f} $'.format(minimum/100))
print('Prix maximum {:.2f} $'.format(maximum/100))

# distribution des données (prix)
plt.hist(prix/100)
plt.title('Distribution des Prix')
plt.xlabel('Prix ($)')
plt.show()
print('En observant la distribution des prix nous pouvons conclure que ces prix

# analyse des extremum
minimum_loc = np.argmin(prix)
maximum_loc = np.argmax(prix)

# minimum location
o_city_min = df_ticket['o_city'].loc[minimum_loc]
d_city_min = df_ticket['d_city'].loc[minimum_loc]
o_city_min = df_cities.loc[df_cities['id'] == o_city_min]
d_city_min = df_cities.loc[df_cities['id'] == d_city_min]
o_name_min = str(o_city_min['local_name']).split()[1:4]
d_name_min = str(d_city_min['local_name']).split()[1:4]
print('\n\nVoyage prix minimum, entre: {} {} {}'.format(o_name_min[0], o_name_mi
print('et: {} {} {}'.format(d_name_min[0], d_name_min[1], d_name_min[2]))

# distance pour prix mini
lat_o = float(str(o_city_min['latitude']).split()[1])
lon_o = float(str(o_city_min['longitude']).split()[1])
lat_d = float(str(d_city_min['latitude']).split()[1])
lon_d = float(str(d_city_min['longitude']).split()[1])
dist = calc_distance(lat_o, lon_o, lat_d, lon_d)
print('Soit une distance de: {:.2f} km (entre villes)'.format(dist))

# maximum location
o_city_max = df_ticket['o_city'].loc[maximum_loc]
d_city_max = df_ticket['d_city'].loc[maximum_loc]
o_city_max = df_cities.loc[df_cities['id'] == o_city_max]
d_city_max = df_cities.loc[df_cities['id'] == d_city_max]
o_name_max = str(o_city_max['local_name']).split()[1:5]
d_name_max = str(d_city_max['local_name']).split()[1:4]
print('\n\nVoyage prix maximum, entre: {} {} {} {}'.format(o_name_max[0], o_name
print('et: {} {} {}'.format(d_name_max[0], d_name_max[1], d_name_max[2]))

o_station_max = df_ticket['o_station'].loc[maximum_loc]
d_station_max = df_ticket['d_station'].loc[maximum_loc]
```

```
o_station_max = df_station.loc[df_station['id'] == o_station_max]
d_station_max = df_station.loc[df_station['id'] == d_station_max]
lat_o = float(str(o_station_max['latitude']).split()[1])
lon_o = float(str(o_station_max['longitude']).split()[1])
lat_d = float(str(d_station_max['latitude']).split()[1])
lon_d = float(str(d_station_max['longitude']).split()[1])
dist = calc_distance(lat_o, lon_o, lat_d, lon_d)
print('Soit une distance de: {:.2f} km (entre gares)'.format(dist))
```

```
Statistique des prix (sans considérer la distance)

Prix moyen 43.81 €
Ecart-type  37.39 €
Prix minimum 3.00 €
Prix maximum 385.50 €
```




```
En observant la distribution des prix nous pouvons conclure que ces prix suiven
t une loi exponentielle. Ce qui indique que la majorité des billets sont acheté
s entre 3 € et 50 €


Voyage prix minimum, entre: Auxerre, Bourgogne-Franche-Comté, France
et: Clamecy, Bourgogne-Franche-Comté, France
Soit une distance de: 36.98 km (entre villes)


Voyage prix maximum, entre: London, England, United Kingdom
et: Bordeaux, Nouvelle-Aquitaine, France
Soit une distance de: 745.78 km (entre gares)
```

Duration

Entrée [149]:
```
# selectionner la colonne trip_duration
duration = df_ticket['trip_duration']

moyenne = secToHour(np.mean(duration))
std = secToHour(np.std(duration))
minimum = secToHour(np.min(duration))
maximum = secToHour(np.max(duration))

print('Durée moyenne {}'.format(moyenne))
print('Ecart-type {}'.format(std))
print('Durée minimum {}'.format(minimum))
print('Durée maximum {}'.format(maximum))
```

```
Durée moyenne 6:52:58.363872
Ecart-type 6:17:58.336759
Durée minimum 0:20:00
Durée maximum 3 days, 2:10:00
```

Distance

Entrée [313]:
```
# selectionner la colonne trip_duration
distance = df_ticket['trip_distance']

moyenne = np.mean(distance)
std = np.std(distance)
minimum = np.min(distance)
maximum = np.max(distance)

print('Distance moyenne {:.2f} km'.format(moyenne))
print('Ecart-type {:.2f} km'.format(std))
print('Distance minimum {:.2f} km'.format(minimum))
print('Distance maximum {:.2f} km'.format(maximum))
```

```
Distance moyenne 362.04 km
Ecart-type 194.27 km
Distance minimum 16.57 km
Distance maximum 1865.47 km
```

Extraire des infos intéressantes par trajet

Entrée [219]:
```
Trips_info = pd.DataFrame(columns=['Departure_station', 'Arrival_station', 'Lowe
```

Entrée [220]:
```
grouped_by_station = df_ticket.groupby(["o_station", "d_station"])

for (start_point, end_point), group in grouped_by_station:

    min_price = group["price_in_cents"].min()/100
    max_price = group["price_in_cents"].max()/100
    avg_price = group["price_in_cents"].mean()/100

    min_duration = secToHour(group['trip_distance'].min())
    max_duration = secToHour(group['trip_duration'].max())
    avg_duration = secToHour(group['trip_duration'].mean())

    row = {'Departure_station': df_station[df_station['id']== start_point].uniqu
           'Arrival_station': df_station[df_station['id']== end_point].unique_na
           'Lowest_price':min_price,
           'Average_price':avg_price,
           'Highest_price':max_price,
           'Min_Duration':min_duration ,
           'Average_duration':avg_duration,
           'Max_duration':max_duration}

    Trips_info=Trips_info.append(row,ignore_index=True)
```

Entrée [221]:
```
def highlight_cols(x):
    r = 'background-color: red'
    g = 'background-color: green'
    df1 = pd.DataFrame('', index=x.index, columns=x.columns)
    df1.iloc[:, 4] = r
    df1.iloc[:, 7] = r
    df1.iloc[:, 2] = g
    df1.iloc[:, 5] = g

    return df1
```

Entrée [228]:
```
Trips_info.sample(10).style.format({"Lowest_price": "${:20,.2f}",
                                    'Average_price': "${:20,.2f}",
                                    'Highest_price': "${:20,.2f}"})\
                                   .background_gradient(cmap='Blues',subset=['Lowest
```

Out[228]:

	Departure_station	Arrival_station	Lowest_price	Average_price	Highest_price	Min_Duration
1478	Nevers (Central bus station)	Gare SNCF Lyon Part-Dieu	$ 20.00	$ 43.46	$ 61.80	0 days 00:03:09.490286
1300	Gare SNCF Lyon-Perrache	Gare routière de Toulouse	$ 20.00	$ 24.52	$ 31.90	0 days 00:05:57.062609
1628	P&O Ferries Calais	Manchester ZOB	$ 44.27	$ 44.27	$ 44.27	0 days 00:06:34.837116
912	Gare routière de Quimper	Gare routière de Strasbourg	$ 34.00	$ 39.66	$ 47.00	0 days 00:14:38.134646
1121	Gare SNCF Auxerre	Paris Gare de Lyon	$ 30.00	$ 30.00	$ 30.00	0 days 00:02:26.956268
1604	Parking St. Bénézet	Gare Lille-Europe	$ 43.90	$ 52.20	$ 58.90	0 days 00:12:34.800899
2105	Nice Saint-Augustin	Marne-La-Vallée - Chessy - Gare Tgv et OuiGo (À 200m Du Parc Disneyland Paris)	$ 135.70	$ 138.08	$ 147.60	0 days 00:11:10.648481
429	Gare de Bercy	Bourg-les-Valence	$ 29.00	$ 30.30	$ 32.90	0 days 00:07:50.698309
716	Gare routière d'Angers	Gare Routière Internationale de Paris-Gallieni - Paris Gallieni Porte Bagnolet	$ 10.00	$ 12.00	$ 18.00	0 days 00:04:29.983363
2025	Marne-La-Vallée - Chessy - Gare Tgv et OuiGo (À 200m Du Parc Disneyland Paris)	Gare routière d'Angers	$ 58.50	$ 62.30	$ 66.00	0 days 00:04:52.811012

différence de prix moyen et durée selon le train, le bus et le covoit selon la distance du trajet (0-200km, 201-800km, 800-2000km, 2000+km)

Entrée [229]:
```
df_providers.columns = ['company','company_id','provider_id','name','fullname',
table = pd.merge(df_ticket, df_providers, on='company')
table.head()
```

Out[229]:

	id	company	o_station	d_station	departure_ts	arrival_ts	price_in_cents	search_
0	6795025	8385	NaN	NaN	2017-10-13 14:00:00+00	2017-10-13 20:10:00+00	4550	2017-10-00:13:31.327+
1	6795030	8385	NaN	NaN	2017-10-06 05:30:00+00	2017-10-06 08:30:00+00	1800	2017-10-01:03:18.948+
2	6795031	8385	NaN	NaN	2017-10-06 07:00:42+00	2017-10-06 09:30:42+00	2150	2017-10-01:03:18.948+
3	6795032	8385	NaN	NaN	2017-10-06 07:10:00+00	2017-10-06 09:40:00+00	1700	2017-10-01:03:18.948+
4	6795033	8385	NaN	NaN	2017-10-06 10:00:00+00	2017-10-06 12:50:00+00	1700	2017-10-01:03:18.948+

5 rows × 23 columns

Entrée [249]:
```
summary_df = pd.DataFrame(columns=['bus','carpooling','train'])

nb_utilisation = table.groupby('transport_type')['transport_type'].count()
nb_utilisation.name= "Nombres d'Utilisation"

prix_avg = table.groupby('transport_type')['price_in_cents'].sum() / table.group
prix_avg.name = "prix moyen $"

duration_avg = table.groupby('transport_type')['trip_duration'].sum()/table.grou
duration_avg.name = "durée moyenne"

summary_df = summary_df.append(nb_utilisation)
summary_df = summary_df.append(prix_avg/100)
summary_df = summary_df.append(duration_avg.map(secToHour))

summary_df.transpose().head().style.bar(subset=["prix moyen $"], color='#ee1f5f
```

Out[249]:

	Nombres d'Utilisation	prix moyen $	durée moyenne
bus	13744	36.4116	0 days 14:37:12.088184
carpooling	41441	27.4217	0 days 04:06:38.330156
train	18929	85.0663	0 days 07:20:03.435998

Comparatif des modes de transport avec tranches de distance

Entrée [284]:
```
summary_df_2 = pd.DataFrame()
summary_df_2[['Prix (0:200)', 'Durée (0:200)']] =table.loc[table["trip_distance'
summary_df_2[['Prix (200:800)', 'Durée (200:800)']] = table.loc[table["trip_dist
summary_df_2[['Prix (800:2000)', 'Durée (800:2000)']] = table.loc[table["trip_di
summary_df_2[['Prix (>=2000)', 'Durée (>=2000)']] = table.loc[table["trip_distar
```

Entrée [285]:
```
summary_df_2
```

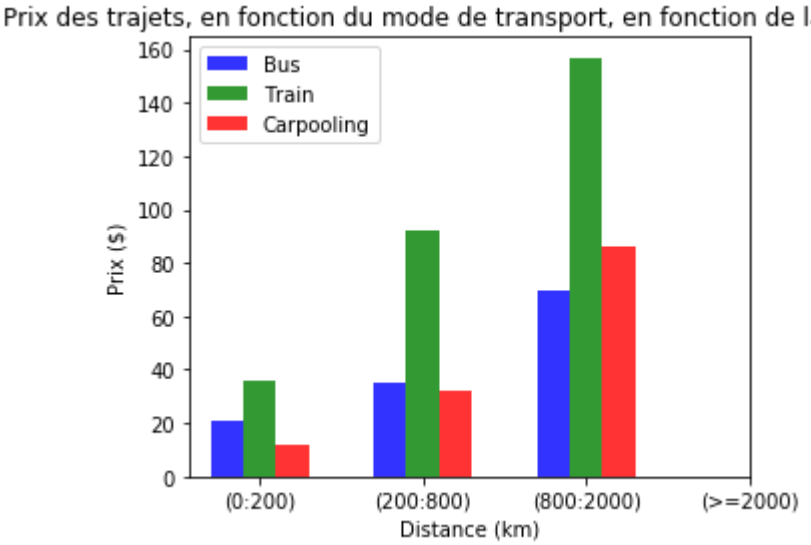
Out[285]:

transport_type	Prix (0:200)	Durée (0:200)	Prix (200:800)	Durée (200:800)	Prix (800:2000)	Durée (800:2000)
bus	2094.513676	37297.745163	3528.332616	51094.213498	6929.113051	89529.375000
carpooling	1177.352385	7027.342428	3218.064310	17147.272436	8626.545455	47160.839161
train	3597.305970	15700.097015	9189.725709	27854.126287	15680.126183	43982.460568

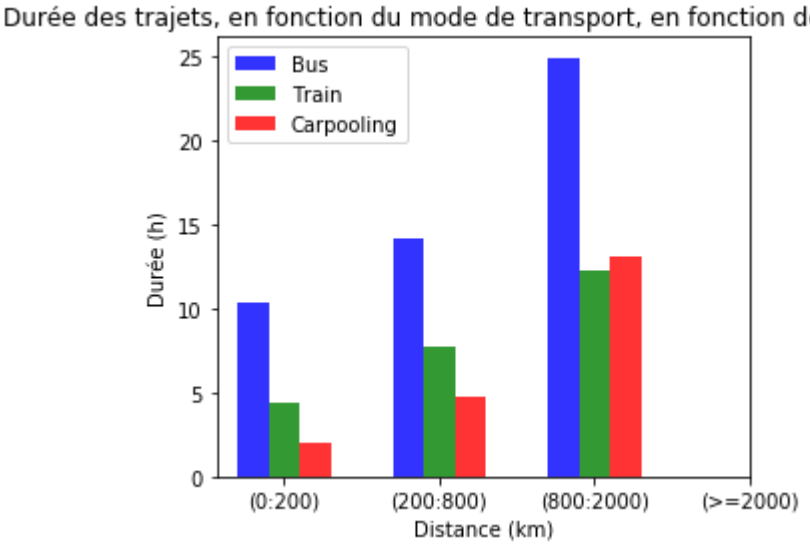
Entrée [286]:
```
summary_df_2.loc['carpooling']['Prix (0:200)']
```

Out[286]:
```
1177.352384745594
```

Entrée [287]:
```
# data to plot
n_groups = 4
Bus_Price = (summary_df_2.loc['bus']['Prix (0:200)']/100, summary_df_2.loc['bus
Train_Price = (summary_df_2.loc['train']['Prix (0:200)']/100, summary_df_2.loc[
Carpooling_Price = (summary_df_2.loc['carpooling']['Prix (0:200)']/100, summary_

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.2
opacity = 0.8

rects1 = plt.bar(index, Bus_Price, bar_width,
alpha=opacity,
color='b',
label='Bus')

rects2 = plt.bar(index + bar_width, Train_Price, bar_width,
alpha=opacity,
color='g',
label='Train')

rects3 = plt.bar(index + 2*bar_width, Carpooling_Price, bar_width,
alpha=opacity,
color='r',
label='Carpooling')

plt.xlabel('Distance (km)')
plt.ylabel('Prix ($)')
plt.title('Prix des trajets, en fonction du mode de transport, en fonction de la
plt.xticks(index + bar_width, ('(0:200)', '(200:800)', '(800:2000)', '(>=2000)')
plt.legend()

plt.tight_layout()
plt.show()
```



Entrée [289]:
```
# data to plot
n_groups = 4
Bus_Price = (summary_df_2.loc['bus']['Durée (0:200)']/3600, summary_df_2.loc['bu
Train_Price = (summary_df_2.loc['train']['Durée (0:200)']/3600, summary_df_2.loc
Carpooling_Price = (summary_df_2.loc['carpooling']['Durée (0:200)']/3600, summar

# create plot
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.2
opacity = 0.8

rects1 = plt.bar(index, Bus_Price, bar_width,
alpha=opacity,
color='b',
label='Bus')

rects2 = plt.bar(index + bar_width, Train_Price, bar_width,
alpha=opacity,
color='g',
label='Train')

rects3 = plt.bar(index + 2*bar_width, Carpooling_Price, bar_width,
alpha=opacity,
color='r',
label='Carpooling')

plt.xlabel('Distance (km)')
plt.ylabel('Durée (h)')
plt.title('Durée des trajets, en fonction du mode de transport, en fonction de l
plt.xticks(index + bar_width, ('(0:200)', '(200:800)', '(800:2000)', '(>=2000)')
plt.legend()

plt.tight_layout()
plt.show()
```



Entrée [341]:
```
### plus grande distance
latA,lngA = getCoords(df_ticket[df_ticket.trip_distance == df_ticket.trip_distar
latC,lngC = getCoords(int(df_ticket[df_ticket.trip_distance == df_ticket.trip_di
latB,lngB = getCoords(df_ticket[df_ticket.trip_distance == df_ticket.trip_distar

### plus grande durée
latAA,lngAA = getCoords(df_ticket[df_ticket.trip_duration == df_ticket.trip_dura
#latCC,lngCC = getCoords(int(df_ticket[df_ticket.trip_duration == df_ticket.trip
latBB,lngBB = getCoords(df_ticket[df_ticket.trip_duration == df_ticket.trip_dura
```

Entrée [371]:
```
import folium

m2 = folium.Map(location=[48.7020953,18.0528321], tiles='openstreetmap', zoom_st
points = [(latA,lngA),(latC,lngC),(latB,lngB)]
points2 = [(latAA,lngAA),(latBB,lngBB)]

folium.PolyLine(points, color="red", weight=2.5, opacity=1).add_to(m2)
folium.PolyLine(points2, color="blue", weight=2.5, opacity=1).add_to(m2)

m2.save('map3.html')
```

Entrée [374]:
```
m2
```

Out[374]:

Entrée [ ]: