Taxi Travel Data Analysis

In this demo, we will be doing some demos on temporal feature engineering with the Kaggle Dataset

Loading libraries, datasets

In [ ]:
```
import numpy as np, pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import holidays
import json
from folium.plugins import HeatMap
import folium
```

In [ ]:
```
# These are all of the files you are given
df_tr = pd.read_csv("archive/train.csv")
```

In [ ]:
```
df_tr.head()
```

Out[ ]:

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE
0	1372636858620000589	C	NaN	NaN	20000589	1372636858	A
1	1372637303620000596	B	NaN	7.0	20000596	1372637303	A
2	1372636951620000320	C	NaN	NaN	20000320	1372636951	A
3	1372636854620000520	C	NaN	NaN	20000520	1372636854	A
4	1372637091620000337	C	NaN	NaN	20000337	1372637091	A

In [ ]:
```
df_tr.shape
```

Out[ ]:
```
(1710670, 9)
```

Get Computed Time from POLYLINE

Our goal is to predict the travel-time of the taxi, which can be derived from the POLYLINE length.

Recall:

> The travel time of the trip (the prediction target of this project)
> is defined as the (number of points-1) x 15 seconds.
> For example, a trip with 101 data points in POLYLINE has a length of
> (101-1) * 15 = 1500 seconds. Some trips have missing
> data points in POLYLINE, indicated by MISSING_DATA column, and it is
> part of the challenge how you utilize this knowledge.

We are not doing anything with the MISSING_DATA. It is up to you to find a way to use (or ignore) that information.

In [ ]:
```
# Over every single
def polyline_to_trip_duration(polyline):
    return max(polyline.count("[") - 2, 0) * 15

# This code creates a new column, "LEN", in our dataframe. The value is
# the (polyline_length - 1) * 15, where polyline_length = count("[") - 1
df_tr["LEN"] = df_tr["POLYLINE"].apply(polyline_to_trip_duration)
```

In [ ]:
```
Portugal_holidays = holidays.PT()
def parse_time(x):
    # We are using python's builtin datetime library
    # https://docs.python.org/3/library/datetime.html#datetime.date.fromtimestamp

    # Each x is essentially a 1 row, 1 column pandas Series
    dt = datetime.fromtimestamp(x["TIMESTAMP"])
    is_holiday = datetime(dt.year, dt.month, dt.day) in Portugal_holidays
    day_before_holiday = False
    try:
        day_before_holiday = datetime(dt.year, dt.month, dt.day + 1) in Portugal_holic
    except:
        pass
    else:
        day_before_holiday = datetime(dt.year, dt.month, dt.day + 1) in Portugal_holic

    return dt.year, dt.month, dt.day, dt.hour, dt.weekday(), is_holiday, day_before_ho

# Because we are assigning multiple values at a time, we need to "expand" our computed
# the column axis, or axis 1
# https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html
df_tr[["YR", "MON", "DAY", "HR", "WK", "is_holiday", "day_before_holiday"]] = df_tr[['
```

In [ ]:
```
df_tr.head(5)
```

Out[ ]:

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE
0	1372636858620000589	C	NaN	NaN	20000589	1372636858	A
1	1372637303620000596	B	NaN	7.0	20000596	1372637303	A
2	1372636951620000320	C	NaN	NaN	20000320	1372636951	A
3	1372636854620000520	C	NaN	NaN	20000520	1372636854	A
4	1372637091620000337	C	NaN	NaN	20000337	1372637091	A

In [ ]:
```
df_tr[df_tr['POLYLINE'] == '[]'].head(10)
```

Out[ ]:

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_T
762	1372665673620000353	C	NaN	NaN	20000353	1372665673	
1161	1372669158620000562	C	NaN	NaN	20000562	1372669158	
1459	1372665875620000496	C	NaN	NaN	20000496	1372665875	
1677	1372667320620000288	C	NaN	NaN	20000288	1372667320	
1719	1372676112620000600	C	NaN	NaN	20000600	1372676112	
1946	1372679607620000080	C	NaN	NaN	20000080	1372679607	
2468	1372679017620000118	A	11499.0	NaN	20000118	1372679017	
2789	1372689298620000901	C	NaN	NaN	20000901	1372689298	
2893	1372691354620000021	B	NaN	10.0	20000021	1372691354	
3036	1372692506620000981	C	NaN	NaN	20000981	1372692506	

In [ ]:
```
df_tr[df_tr['MISSING_DATA'] == True].head(10)
```

Out[ ]:

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DA
105621	1374554455620000625	B	NaN	23.0	20000625	1374554455	
171397	1375863510620000454	B	NaN	62.0	20000454	1375863510	
299137	1378544246620000057	B	NaN	55.0	20000057	1378544246	
457486	1381233613620000387	C	NaN	NaN	20000387	1381233613	
738466	1386346894620000904	C	NaN	NaN	20000904	1386346894	
782321	1387137779620000640	C	NaN	NaN	20000640	1387137779	
848552	1388351478620000678	A	9738.0	NaN	20000678	1388351478	
932391	1390005983620000640	C	NaN	NaN	20000640	1390005983	
1275934	1396631707620000163	C	NaN	NaN	20000163	1396631707	
1432196	1399405185620000508	C	NaN	NaN	20000508	1399405185	

In [ ]:
```
df_tr.head(10)
```

Out[ ]:

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE
0	1372636858620000589	C	NaN	NaN	20000589	1372636858	A
1	1372637303620000596	B	NaN	7.0	20000596	1372637303	A
2	1372636951620000320	C	NaN	NaN	20000320	1372636951	A
3	1372636854620000520	C	NaN	NaN	20000520	1372636854	A
4	1372637091620000337	C	NaN	NaN	20000337	1372637091	A
5	1372636965620000231	C	NaN	NaN	20000231	1372636965	A
6	1372637210620000456	C	NaN	NaN	20000456	1372637210	A
7	1372637299620000011	C	NaN	NaN	20000011	1372637299	A
8	1372637274620000403	C	NaN	NaN	20000403	1372637274	A
9	1372637905620000320	C	NaN	NaN	20000320	1372637905	A

Create a Prediction File

In [ ]:
```
mean, std = df_tr["LEN"].mean(), df_tr["LEN"].std()
median = df_tr["LEN"].median()
print(f"{mean=} {median=} {std=}")
```

```
mean=716.4264615618442 median=600.0 std=684.7511617510816
```

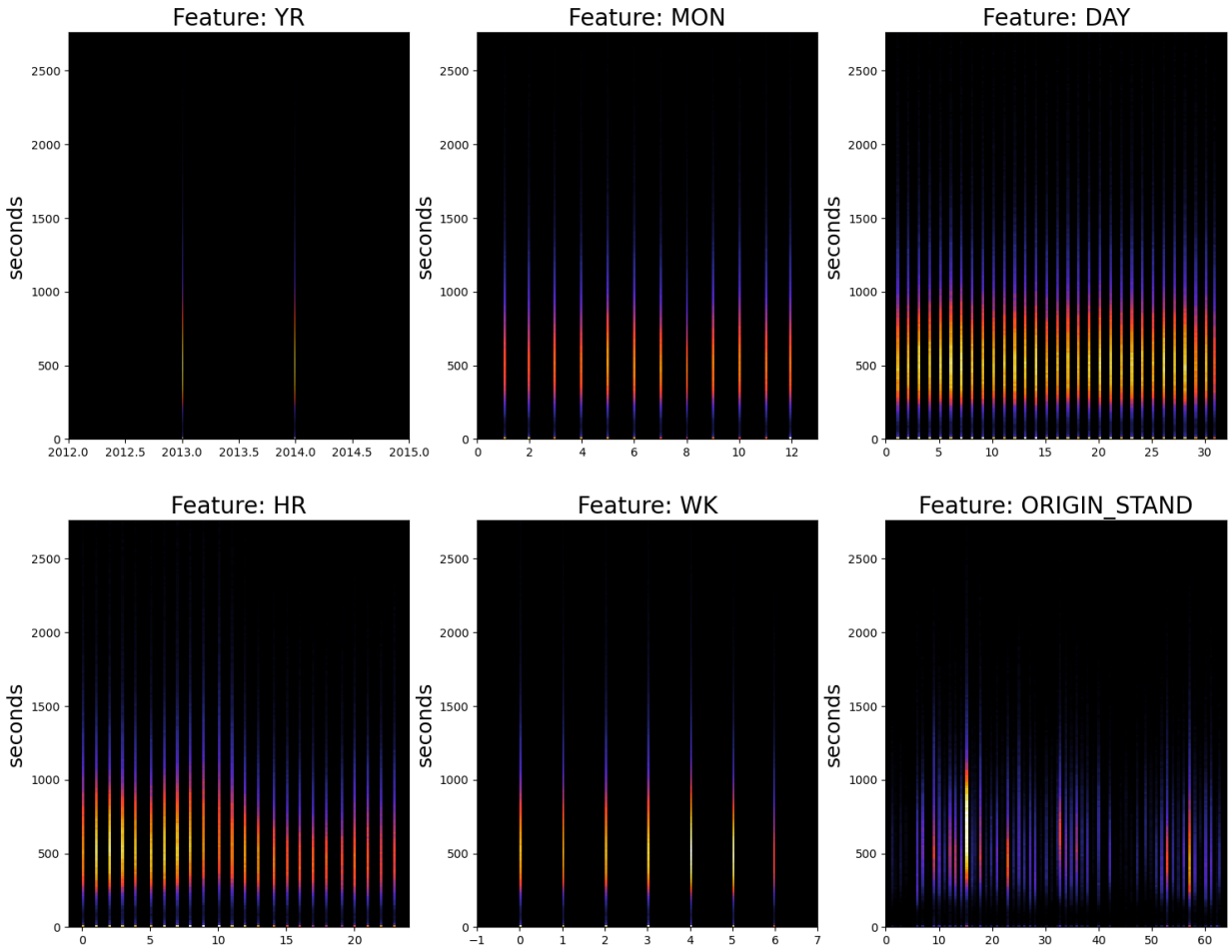
In [ ]:
```
# Sample submission file that is given on kaggle
df_sample = pd.read_csv("sampleSubmission.csv")

df_sample["TRAVEL_TIME"] = 716.43

# mean(716.43) -> 792.73593
# median(600) -> 784.74219
df_sample.to_csv("my_pred.csv", index=None)
```

Do some Feature Analysis

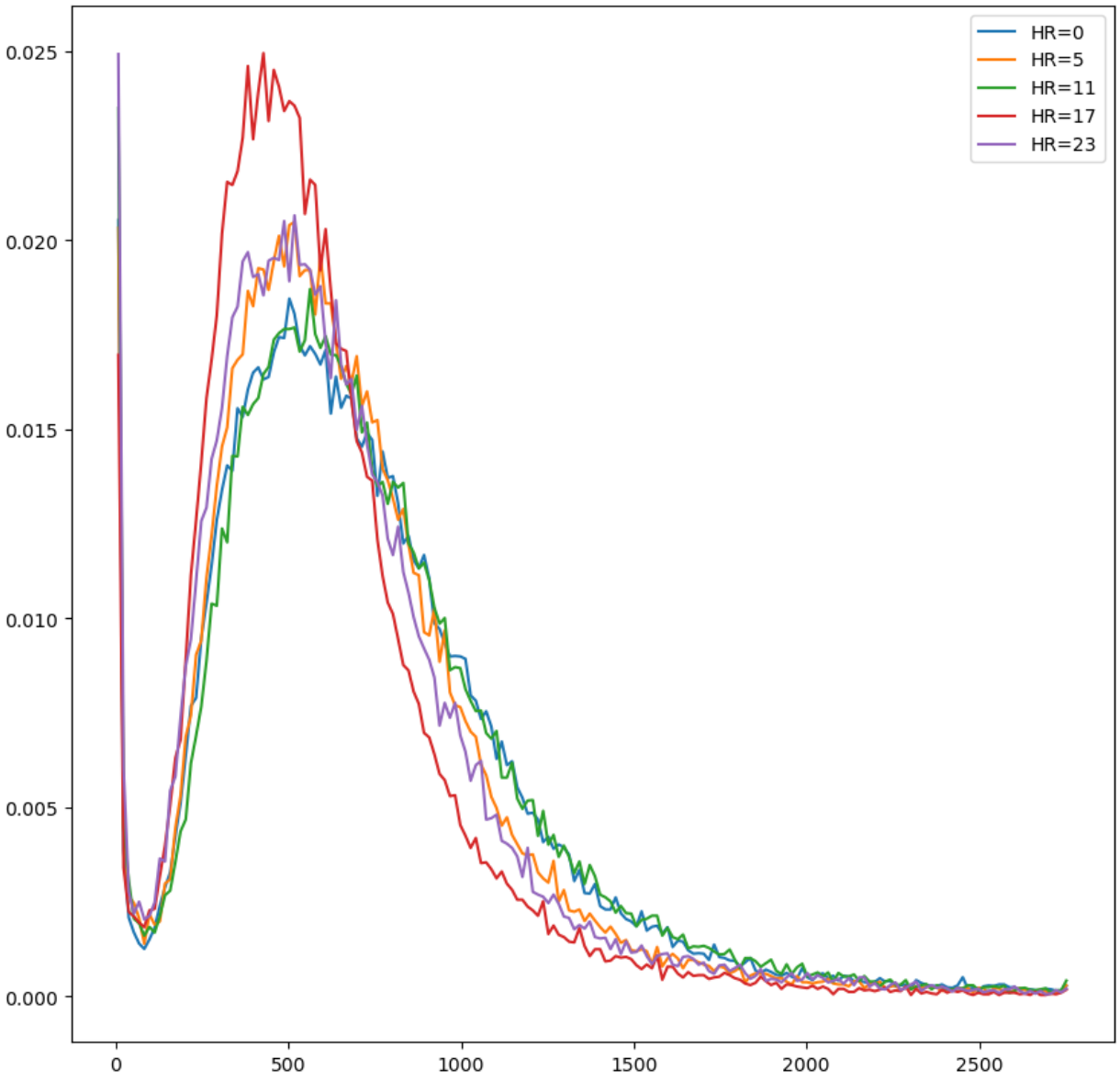
For our feature analysis, we are looking at which of our engineered features may be useful in making a taxicab time regression model

In [ ]:
```
# First n samples to analyze. Set to -1 to use all data
end = -1

outlier_threshold = 3

# "Choose all data, where the trip length is less than 3 standard deviations away from
# This is to remove outliers. Otherwise, our plots would look very squished (since the
# VERRRRRY long taxi trips in the dataset)
df_trimmed = df_tr[df_tr["LEN"] < mean + outlier_threshold * std]

# Because our y-values only take on multiples of 15, we want just enough buckets in a
# such that each buckets counts one value's frequency. (e.x. one bucket counts how mar
# how many 30s trips, etc. )
buckets = (int(mean + outlier_threshold * std) // 15)

print(f"Using: {len(df_trimmed)}/{len(df_tr)}")

fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18,14))

# Now, we visualize some features that we think might be useful
for idx, v in enumerate(["YR", "MON", "DAY", "HR", "WK", "ORIGIN_STAND"]):
  # idx // 3 = row, idx % 3 = column
  ax = axs[idx // 3, idx % 3]

  # Remove any rows with invalid values
  df_subset = df_trimmed.dropna(subset=v)

  # Create a histogram. Look up the documentation for more details
  ax.hist2d(df_subset[v][:end], df_subset["LEN"][:end], cmap="CMRmap", bins=(120,bucke

  # Some stylistic things to make the graphs look nice
  ax.set_xlim(ax.get_xlim()[0] - 1, ax.get_xlim()[1] + 1)
  ax.set_facecolor("black")
  ax.set_ylabel("seconds", fontsize=18)
  ax.set_title(f"Feature: {v}", fontsize=20)
```

```
Using: 1692771/1710670
```



In [ ]:
```
plt.figure(figsize=(10,10))
for v in [0, 5, 11, 17, 23]:
  # Filter data where the HR matches v
  hourly_data = df_trimmed[df_trimmed["HR"] == v]["LEN"]
  histogram, bin_boundary = np.histogram(hourly_data, bins=buckets)
  histogram = histogram / len(hourly_data)
  # The center is the left_bound and right_bound of a bucket
  bin_centers = [(bin_boundary[i] + bin_boundary[i + 1]) / 2 for i in range(buckets)]
  plt.plot(bin_centers, histogram, label=f"HR={v}")
plt.legend();
```


HR=0
HR=5
HR=11
HR=17
HR=23
0.025
0.020
0.015
0.010
0.005
0.000
0
500
1000
1500
2000
2500

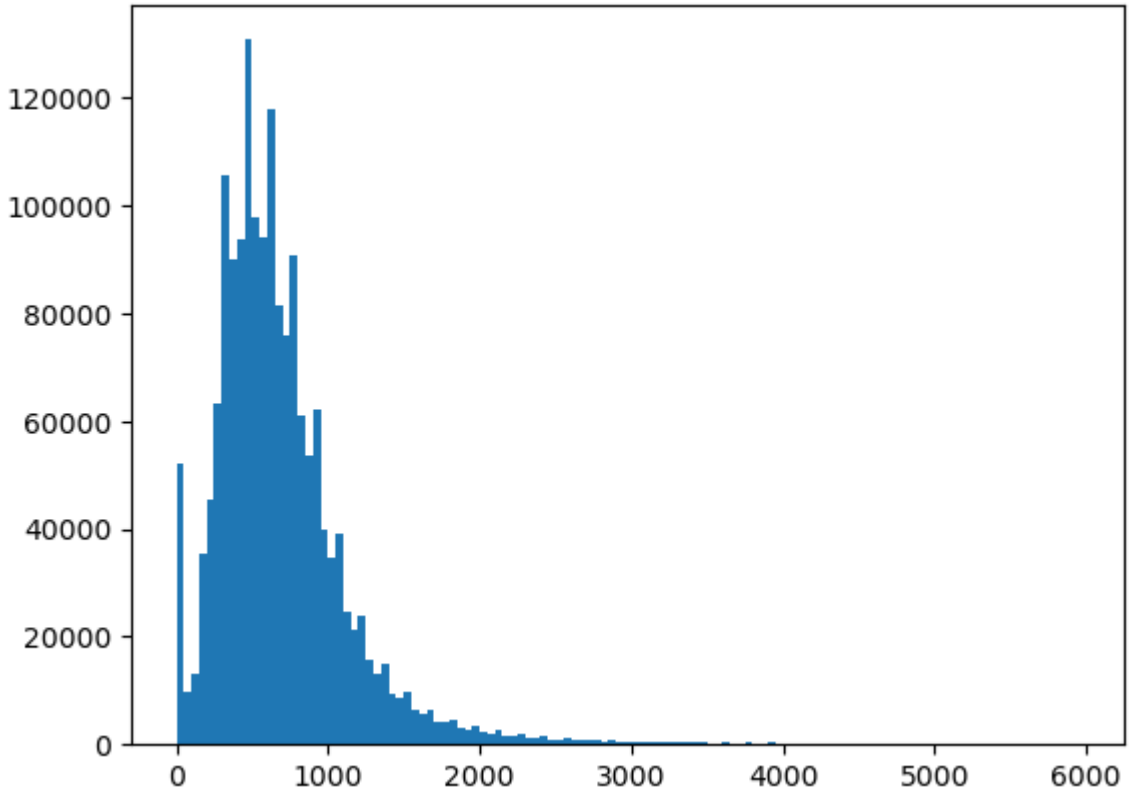
In [ ]:
```
df_tr.head()
```

Out[ ]:

	TRIP_ID	CALL_TYPE	ORIGIN_CALL	ORIGIN_STAND	TAXI_ID	TIMESTAMP	DAY_TYPE
0	1372636858620000589	C	NaN	NaN	20000589	1372636858	A
1	1372637303620000596	B	NaN	7.0	20000596	1372637303	A
2	1372636951620000320	C	NaN	NaN	20000320	1372636951	A
3	1372636854620000520	C	NaN	NaN	20000520	1372636854	A
4	1372637091620000337	C	NaN	NaN	20000337	1372637091	A

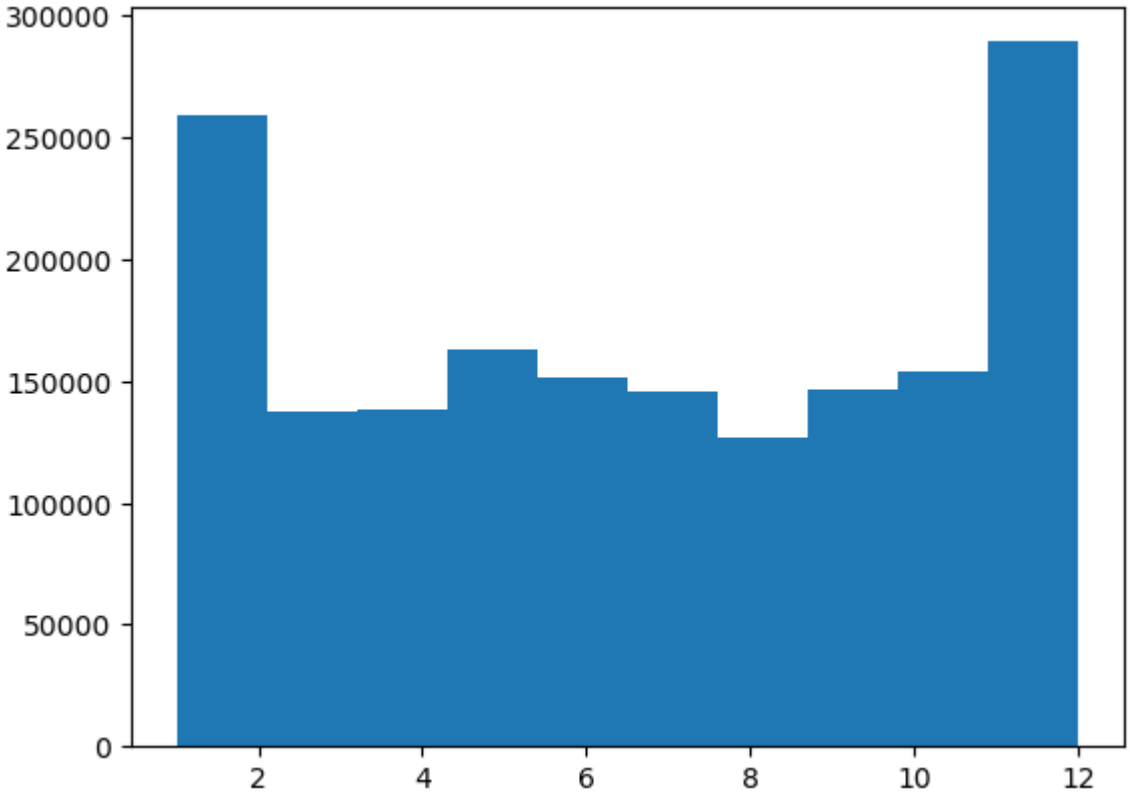
In [ ]:
```
plt.hist(df_tr['LEN'], [i for i in range(0, 6_000, 50)])
```

Out[ ]:
```
(array([5.21030e+04, 9.62000e+03, 1.30080e+04, 3.54730e+04, 4.55930e+04,
        6.31670e+04, 1.05578e+05, 9.00310e+04, 9.37520e+04, 1.30734e+05,
        9.78410e+04, 9.42000e+04, 1.17842e+05, 8.16220e+04, 7.57280e+04,
        9.08600e+04, 6.09000e+04, 5.37100e+04, 6.22460e+04, 4.00130e+04,
        3.47120e+04, 3.90900e+04, 2.46130e+04, 2.13840e+04, 2.38610e+04,
        1.55960e+04, 1.32000e+04, 1.50420e+04, 9.54900e+03, 8.60100e+03,
        9.88300e+03, 6.53500e+03, 5.64900e+03, 6.56300e+03, 4.34900e+03,
        4.04600e+03, 4.71200e+03, 3.14000e+03, 2.76000e+03, 3.40600e+03,
        2.21500e+03, 2.11600e+03, 2.56500e+03, 1.71800e+03, 1.53700e+03,
        1.88400e+03, 1.35500e+03, 1.23900e+03, 1.54000e+03, 1.02200e+03,
        9.80000e+02, 1.21300e+03, 8.10000e+02, 7.27000e+02, 9.16000e+02,
        6.68000e+02, 6.40000e+02, 7.90000e+02, 5.43000e+02, 5.10000e+02,
        6.38000e+02, 4.35000e+02, 4.21000e+02, 5.49000e+02, 4.13000e+02,
        3.52000e+02, 4.41000e+02, 3.39000e+02, 3.22000e+02, 3.40000e+02,
        2.81000e+02, 2.59000e+02, 3.65000e+02, 2.61000e+02, 2.31000e+02,
        3.19000e+02, 2.23000e+02, 2.16000e+02, 2.89000e+02, 2.07000e+02,
        1.67000e+02, 2.54000e+02, 1.70000e+02, 1.60000e+02, 2.14000e+02,
        1.58000e+02, 1.32000e+02, 1.97000e+02, 1.63000e+02, 1.47000e+02,
        1.77000e+02, 1.24000e+02, 1.39000e+02, 1.34000e+02, 1.19000e+02,
        1.16000e+02, 1.39000e+02, 1.00000e+02, 9.60000e+01, 1.19000e+02,
        1.04000e+02, 9.40000e+01, 1.30000e+02, 7.30000e+01, 8.30000e+01,
        9.30000e+01, 7.70000e+01, 8.10000e+01, 9.50000e+01, 7.70000e+01,
        7.40000e+01, 8.50000e+01, 6.30000e+01, 6.90000e+01, 9.10000e+01,
        6.30000e+01, 7.40000e+01, 7.00000e+01, 4.50000e+01]),
 array([   0.,   50.,  100.,  150.,  200.,  250.,  300.,  350.,  400.,
         450.,  500.,  550.,  600.,  650.,  700.,  750.,  800.,  850.,
         900.,  950., 1000., 1050., 1100., 1150., 1200., 1250., 1300.,
        1350., 1400., 1450., 1500., 1550., 1600., 1650., 1700., 1750.,
        1800., 1850., 1900., 1950., 2000., 2050., 2100., 2150., 2200.,
        2250., 2300., 2350., 2400., 2450., 2500., 2550., 2600., 2650.,
        2700., 2750., 2800., 2850., 2900., 2950., 3000., 3050., 3100.,
        3150., 3200., 3250., 3300., 3350., 3400., 3450., 3500., 3550.,
        3600., 3650., 3700., 3750., 3800., 3850., 3900., 3950., 4000.,
        4050., 4100., 4150., 4200., 4250., 4300., 4350., 4400., 4450.,
        4500., 4550., 4600., 4650., 4700., 4750., 4800., 4850., 4900.,
        4950., 5000., 5050., 5100., 5150., 5200., 5250., 5300., 5350.,
        5400., 5450., 5500., 5550., 5600., 5650., 5700., 5750., 5800.,
        5850., 5900., 5950.]),
 <BarContainer object of 119 artists>)
```



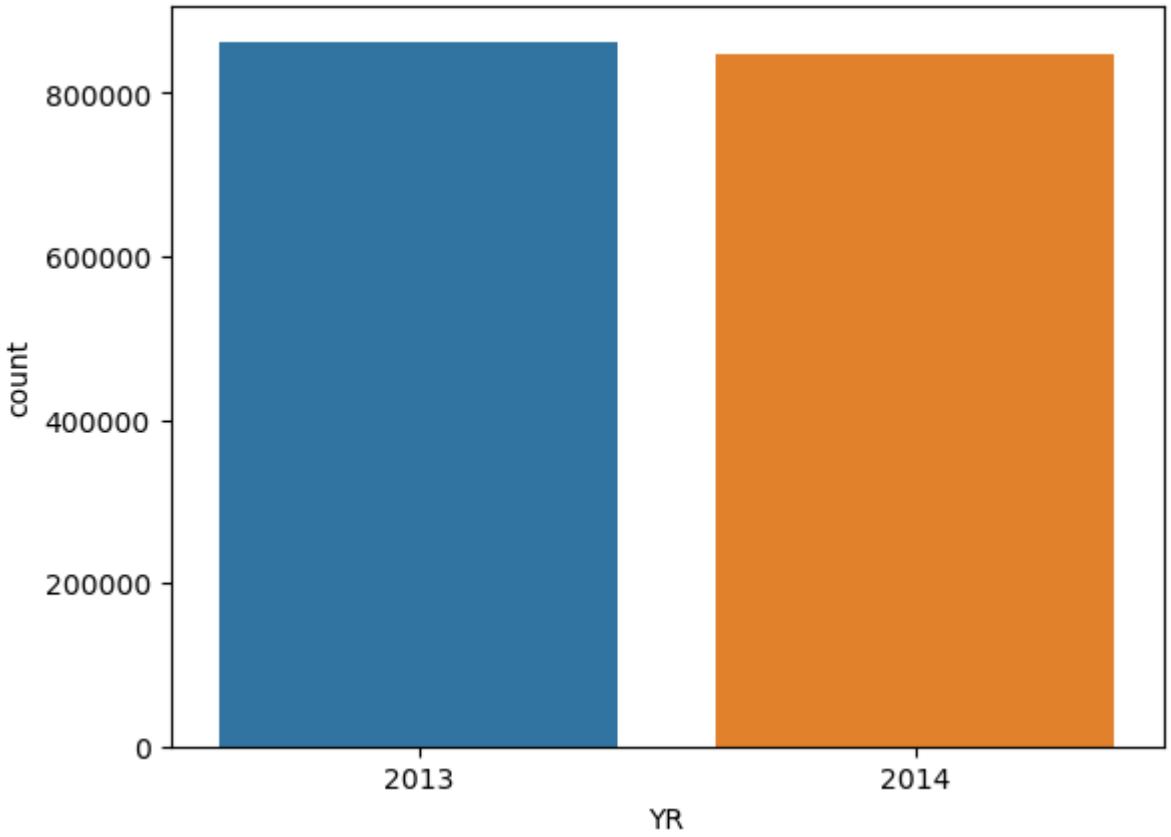
In [ ]:
```
plt.hist(df_tr['MON'])
```

Out[ ]:
```
(array([259075., 137145., 138514., 162744., 151097., 145785., 126642.,
        146654., 153901., 289113.]),
 array([ 1. ,  2.1,  3.2,  4.3,  5.4,  6.5,  7.6,  8.7,  9.8, 10.9, 12. ]),
 <BarContainer object of 10 artists>)
```



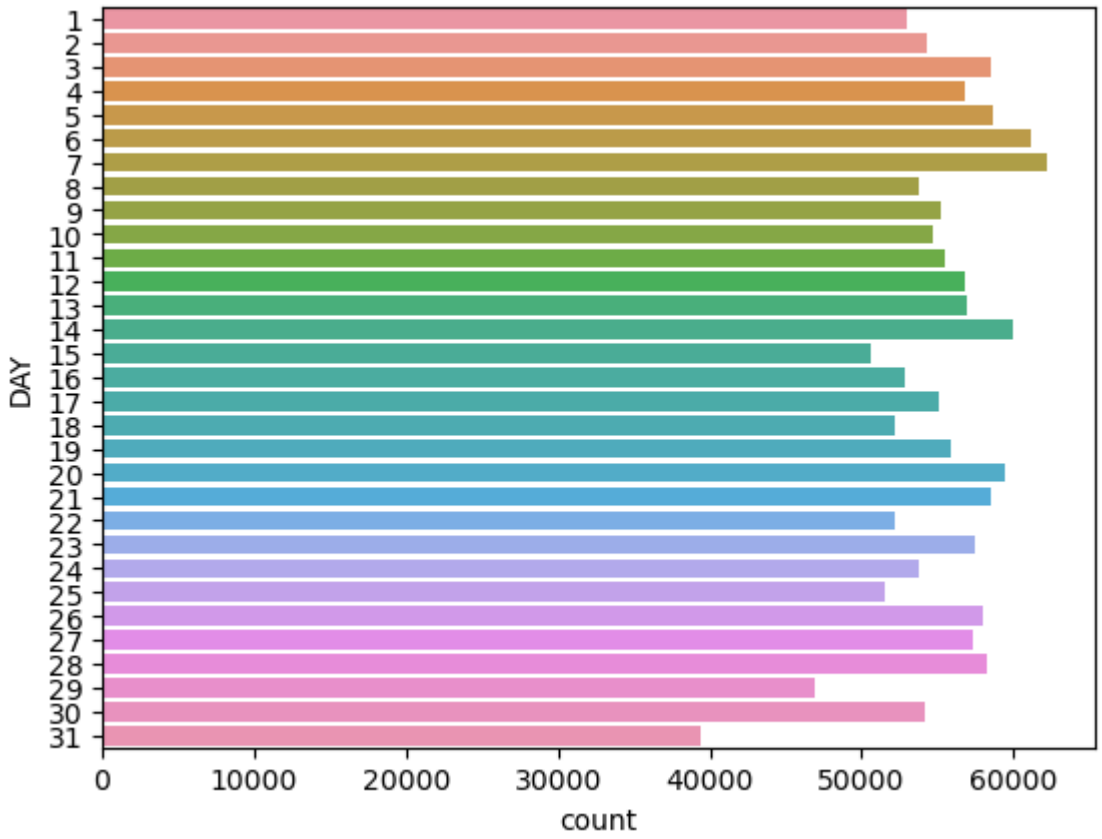
In [ ]:
```
import seaborn
seaborn.countplot(x="YR", data=df_tr)
```

Out[ ]:
```
<Axes: xlabel='YR', ylabel='count'>
```



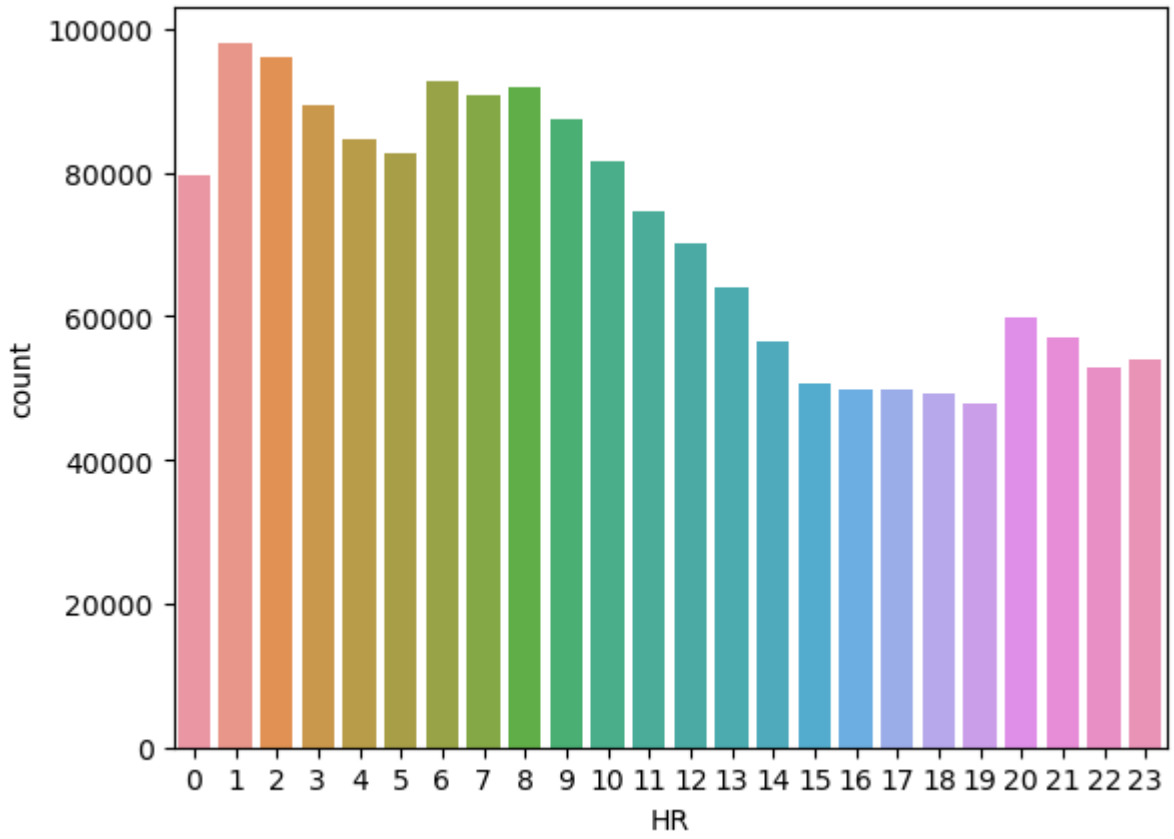
In [ ]:
```
seaborn.countplot(y='DAY', data=df_tr)
```

Out[ ]:
```
<Axes: xlabel='count', ylabel='DAY'>
```



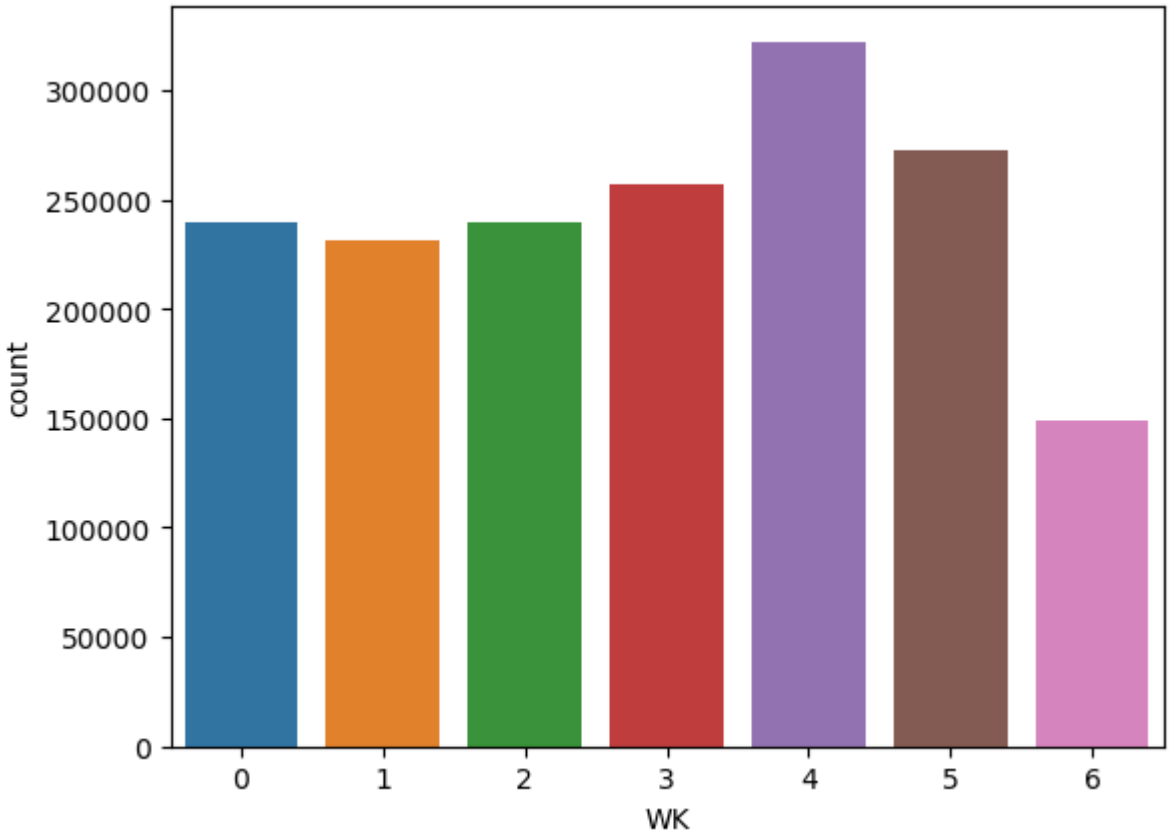
In [ ]:
```
seaborn.countplot(x="HR", data=df_tr)
```

Out[ ]:
```
<Axes: xlabel='HR', ylabel='count'>
```



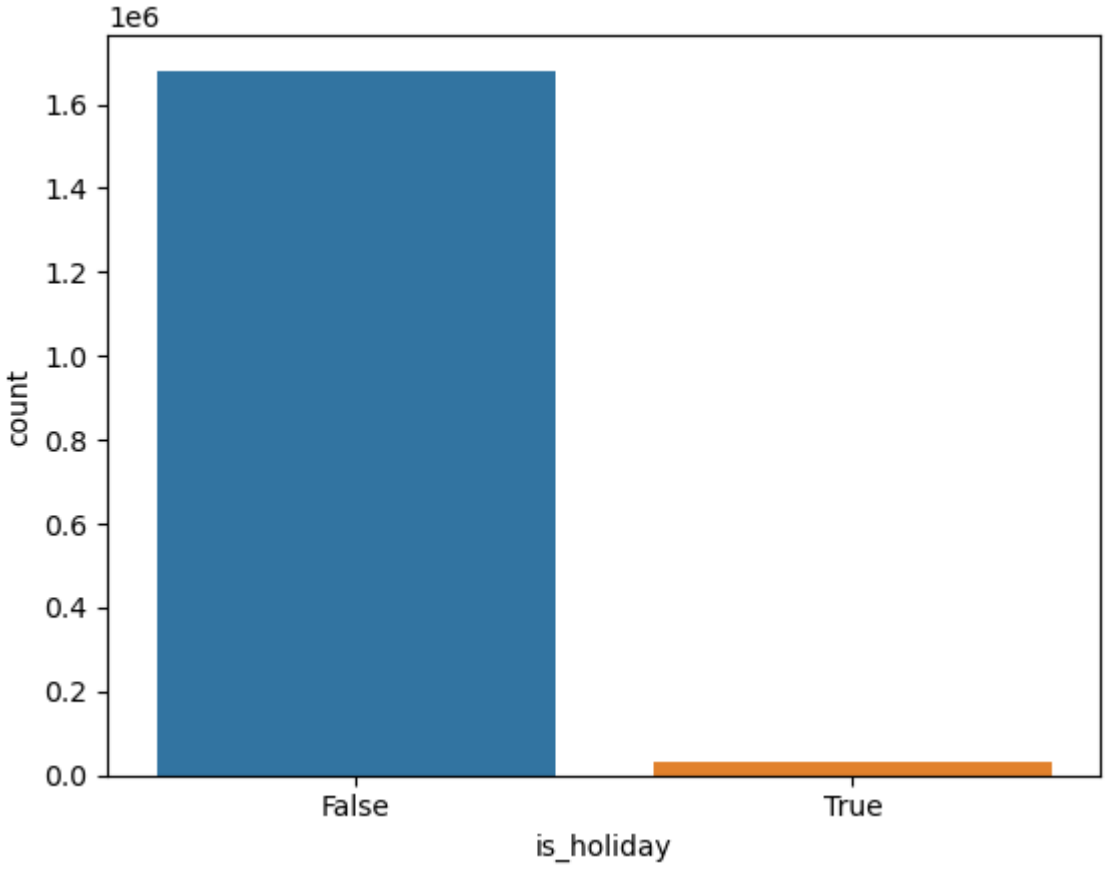
In [ ]:
```
seaborn.countplot(x='WK', data=df_tr)
```

Out[ ]:
```
<Axes: xlabel='WK', ylabel='count'>
```



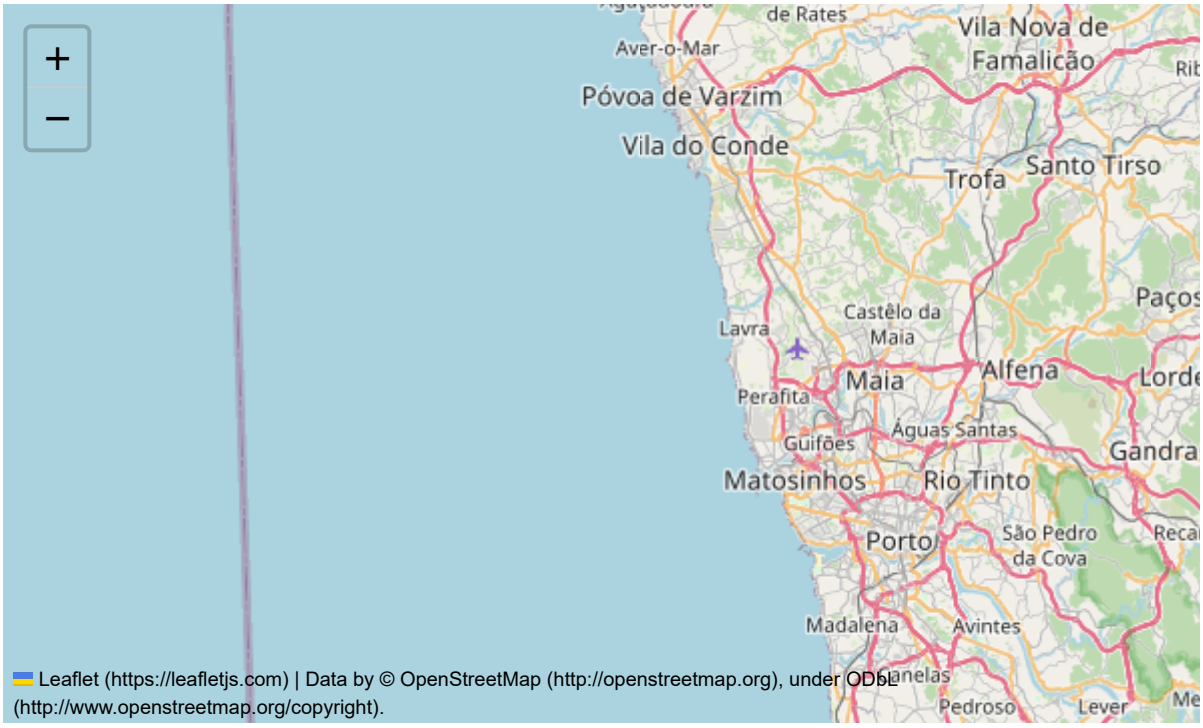
In [ ]:
```
seaborn.countplot(x="is_holiday", data=df_tr)
```

Out[ ]:
```
<Axes: xlabel='is_holiday', ylabel='count'>
```



In [ ]:
```
porto_map = folium.Map(location=[41.1579438, -8.629105299999992])
porto_map
```

Out[ ]:



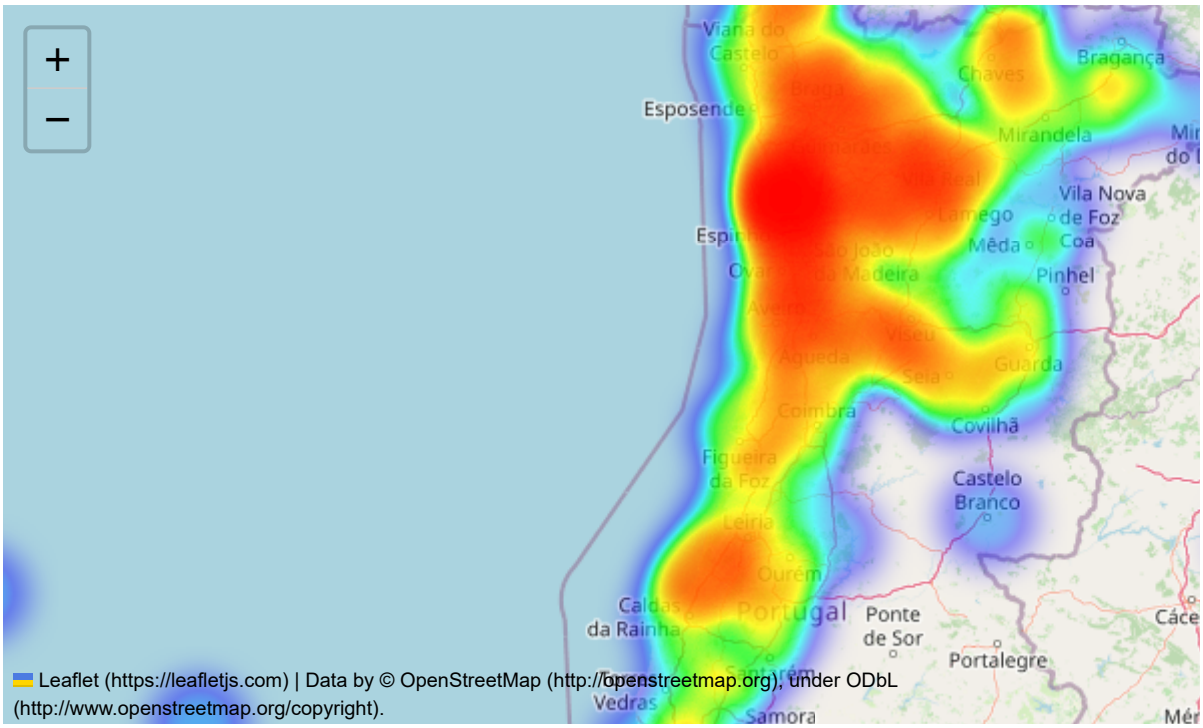
In [ ]:
```
polyline_data = df_tr['POLYLINE']
data = []
for item in polyline_data:
    item_to_json = json.loads(item)
    if len(item_to_json) > 0:
```

```
        coor = item_to_json[0]
        temp = coor[0]
        coor[0] = coor[1]
        coor[1] = temp
        data.append(coor)

len(data)
```

Out[ ]:
```
1704769
```

In [ ]:
```
HeatMap(data).add_to(porto_map)
porto_map
```

Out[ ]:



In [ ]: