Geospatial data analysis

January 7, 2022

1 Welcome To The Notebook

In this project, we are going to learn how to analyze Geo-spatial data using python and folium.

1.1 TASK 1

Importing modules

[1]:
```
import pandas as pd
import datetime , calendar
import folium
```

Importing our data

[6]:
```
data = pd.read_csv("dataset.csv")
data.head()
```

[6]:
```
               TRIP_ID CALL_TYPE   TIMESTAMP  \
0  1372636858620000589         C  1372636858
1  1372637303620000596         B  1372637303
2  1372636951620000320         C  1372636951
3  1372636854620000520         C  1372636854
4  1372637091620000337         C  1372637091

                                           TRIP_PATH
0  [(41.141412, -8.618643), (41.141376, -8.618499…
1  [(41.159826, -8.639847), (41.159871, -8.640351…
2  [(41.140359, -8.612964), (41.14035, -8.613378)…
3  [(41.151951, -8.574678), (41.151942, -8.574705…
4  [(41.18049, -8.645994), (41.180517, -8.645949)…
```

1.2 ### Let's talk about the different columns

- CALL_TYPE: The way used to demand taxi service
  - 'A' : if this trip was dispatched from the central
  - 'B' : if this trip was demanded directly to a taxi driver on a specific stand
  - 'C' : other (i.e. a trip demanded on a random street)
- TIMESTAMP: When the trip starts

- TRIP_PATH: Contains a list of coordinates
  - The first element is the coordinates of the trip's starting point
  - The last element is the coordinates of the trip's end point

1.3 TASK 2

1.3.1 Data Preprocessing

Let's extract starting and ending points of each trip

`- Creating two columns START_LOC and END_LOC`

[7]:
```python
#this is a str
data.TRIP_PATH[0]
```

[7]:
```
'[(41.141412, -8.618643), (41.141376, -8.618499), (41.14251, -8.620326),
(41.143815, -8.622153), (41.144373, -8.623953), (41.144778, -8.62668),
(41.144697, -8.627373), (41.14521, -8.630226), (41.14692, -8.632746),
(41.148225, -8.631738), (41.150385, -8.629938), (41.151213, -8.62911),
(41.15124, -8.629128), (41.152203, -8.628786), (41.152374, -8.628687),
(41.152518, -8.628759), (41.15268, -8.630838), (41.153022, -8.632323),
(41.154489, -8.631144), (41.154507, -8.630829), (41.154516, -8.630829),
(41.154498, -8.630829), (41.154489, -8.630838)]'
```

[8]:
```python
# using eval to change the data type
eval(data.TRIP_PATH[0])
```

[8]:
```
[(41.141412, -8.618643),
 (41.141376, -8.618499),
 (41.14251, -8.620326),
 (41.143815, -8.622153),
 (41.144373, -8.623953),
 (41.144778, -8.62668),
 (41.144697, -8.627373),
 (41.14521, -8.630226),
 (41.14692, -8.632746),
 (41.148225, -8.631738),
 (41.150385, -8.629938),
 (41.151213, -8.62911),
 (41.15124, -8.629128),
 (41.152203, -8.628786),
 (41.152374, -8.628687),
 (41.152518, -8.628759),
 (41.15268, -8.630838),
 (41.153022, -8.632323),
 (41.154489, -8.631144),
 (41.154507, -8.630829),
 (41.154516, -8.630829),
```

```
 (41.154498, -8.630829),
 (41.154489, -8.630838)]
```

[9]:
```python
data.TRIP_PATH = data.TRIP_PATH.apply(eval)
data.head()
```

[9]:
```
               TRIP_ID CALL_TYPE   TIMESTAMP  \
0  1372636858620000589         C  1372636858
1  1372637303620000596         B  1372637303
2  1372636951620000320         C  1372636951
3  1372636854620000520         C  1372636854
4  1372637091620000337         C  1372637091

                                           TRIP_PATH
0  [(41.141412, -8.618643), (41.141376, -8.618499…
1  [(41.159826, -8.639847), (41.159871, -8.640351…
2  [(41.140359, -8.612964), (41.14035, -8.613378)…
3  [(41.151951, -8.574678), (41.151942, -8.574705…
4  [(41.18049, -8.645994), (41.180517, -8.645949)…
```

[10]:
```python
data.TRIP_PATH[0][0] #FIRST PATH
```

[10]:
```
(41.141412, -8.618643)
```

[11]:
```python
# GETTING START POINT AND END POINT
extract_starting_point = lambda list_ : list_[0]
extract_ending_point = lambda list_ : list_[-1]
data["START_LOC"] = data.TRIP_PATH.apply(extract_starting_point)
data["END_LOC"] = data.TRIP_PATH.apply(extract_ending_point)

data.head()
```

[11]:
```
               TRIP_ID CALL_TYPE   TIMESTAMP  \
0  1372636858620000589         C  1372636858
1  1372637303620000596         B  1372637303
2  1372636951620000320         C  1372636951
3  1372636854620000520         C  1372636854
4  1372637091620000337         C  1372637091

                                           TRIP_PATH               START_LOC  \
0  [(41.141412, -8.618643), (41.141376, -8.618499…  (41.141412, -8.618643)
1  [(41.159826, -8.639847), (41.159871, -8.640351…  (41.159826, -8.639847)
2  [(41.140359, -8.612964), (41.14035, -8.613378)…  (41.140359, -8.612964)
3  [(41.151951, -8.574678), (41.151942, -8.574705…  (41.151951, -8.574678)
4  [(41.18049, -8.645994), (41.180517, -8.645949)…   (41.18049, -8.645994)

                  END_LOC
```

```
0  (41.154489, -8.630838)
1   (41.170671, -8.66574)
2    (41.14053, -8.61597)
3  (41.142915, -8.607996)
4  (41.178087, -8.687268)
```

Remapping CALL_TYPE column values to the proper values

[ ]:

```

```

[12]:

```
# UPDATING CALL TYPE COLUMN
CALL_TYPES = {
              "A":"CENTRAL_BASED",
              "B":"STAND_BASED",
              "C":"OTHER"
             }
data.CALL_TYPE = data.CALL_TYPE.map(CALL_TYPES)
data.head()
```

[12]:

```
               TRIP_ID    CALL_TYPE   TIMESTAMP  \
0  1372636858620000589        OTHER  1372636858
1  1372637303620000596  STAND_BASED  1372637303
2  1372636951620000320        OTHER  1372636951
3  1372636854620000520        OTHER  1372636854
4  1372637091620000337        OTHER  1372637091

                                           TRIP_PATH               START_LOC  \
0  [(41.141412, -8.618643), (41.141376, -8.618499…  (41.141412, -8.618643)
1  [(41.159826, -8.639847), (41.159871, -8.640351…  (41.159826, -8.639847)
2  [(41.140359, -8.612964), (41.14035, -8.613378)…  (41.140359, -8.612964)
3  [(41.151951, -8.574678), (41.151942, -8.574705…  (41.151951, -8.574678)
4  [(41.18049, -8.645994), (41.180517, -8.645949)…   (41.18049, -8.645994)

                  END_LOC
0  (41.154489, -8.630838)
1   (41.170671, -8.66574)
2    (41.14053, -8.61597)
3  (41.142915, -8.607996)
4  (41.178087, -8.687268)
```

1.3.2 Data Analysis

Let's answer to some analytical questions using our data

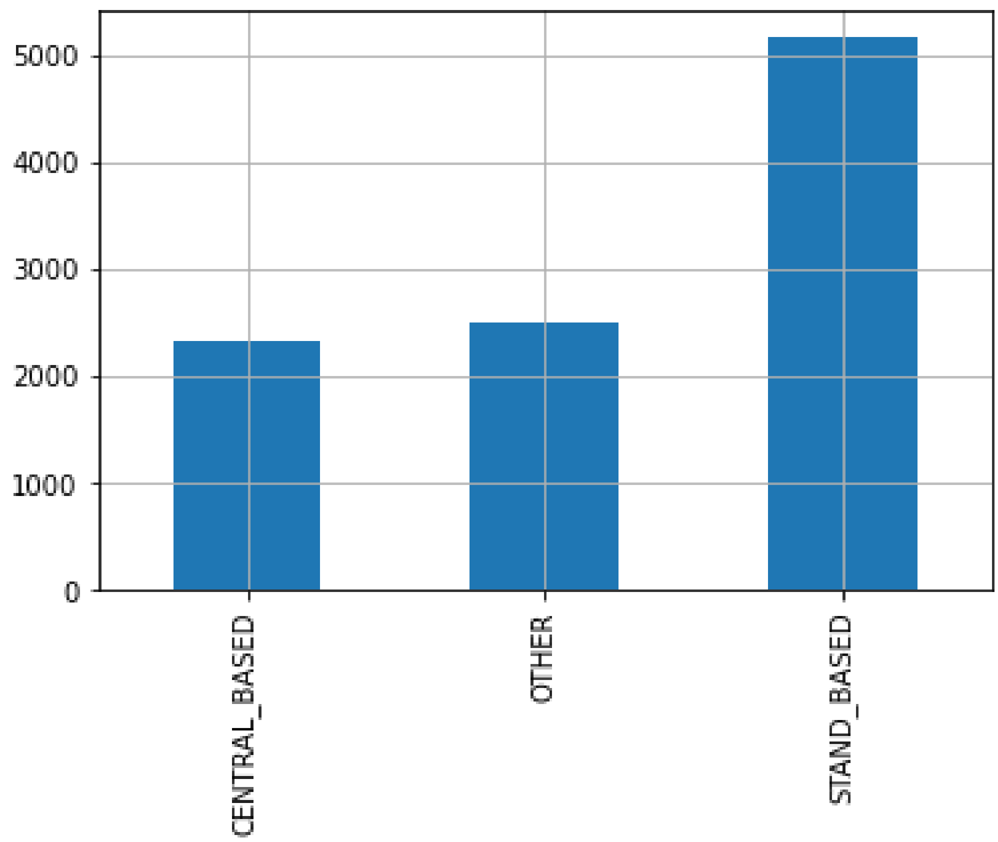
1.3.3 Question 1 - What are the most common ways to get a taxi in Porto?

[13]:
```
q1_data = data.CALL_TYPE.value_counts(sort=False)
q1_data
```

[13]:
```
CENTRAL_BASED    2337
OTHER            2499
STAND_BASED      5164
Name: CALL_TYPE, dtype: int64
```

[14]:
```
q1_data.plot.bar(grid=True)
```

[14]: <AxesSubplot:>



2 TASK 3

2.0.1 Question 2 - Which regions of the city are the best pick up points?

To answer to this question we can use a dotmap.
A dotmap is a scatter plot which is put on a map to represent a geographic data distribution.
And then by looking at the different clusters of the points on the map we can get the answer.

Let's load our map

[15]:
```
Map = folium.Map()
Map
```

[15]:
```
<folium.folium.Map at 0x247fff57a90>
```

[38]:
```
# Porto coordinates (41.15,-8.62)
Map = folium.Map(location = [41.15,-8.62], zoom_start = 14)
Map
```

[38]:
```
<folium.folium.Map at 0x24788b6ffa0>
```

[17]:
```
#ADDING FIRST START POINT TO THE MAP
Map = folium.Map(location = [41.15,-8.62], zoom_start = 14)

point = data.START_LOC.iloc[0]
folium.CircleMarker(location = point , color= "red", radius = 1, weight = 2).
 ↪add_to(Map)
Map
```

[17]:
```
<folium.folium.Map at 0x2478012feb0>
```

[18]:
```
#ADDING ALL FIRST START POINTS TO THE MAP USING fOR LOOP
#by doing this we could have a good representation of different clusters of␣
 ↪data points in Porto
# Answer: Each cluster represent a popular region to get a taxi
Map = folium.Map(location = [41.15,-8.62], zoom_start = 14)

for point in data.START_LOC:
    folium.CircleMarker(location = point , color= "red", radius = 1, weight =␣
 ↪2).add_to(Map)
Map
```

[18]:
```
<folium.folium.Map at 0x2478012fd00>
```

2.0.2 Question 3 - Which regions of the city are the best pick up points for stand-based trips?

To answer to this question we create a dotmap with different Marker clusters, each cluster cor
CALL_TYPE.

Let's check our data again

[19]:
```
data.head()
```

[19]:
```
                TRIP_ID    CALL_TYPE   TIMESTAMP  \
0   1372636858620000589        OTHER  1372636858
1   1372637303620000596  STAND_BASED  1372637303
```

```
2  1372636951620000320        OTHER  1372636951
3  1372636854620000520        OTHER  1372636854
4  1372637091620000337        OTHER  1372637091

                                           TRIP_PATH                START_LOC  \
0  [(41.141412, -8.618643), (41.141376, -8.618499…  (41.141412, -8.618643)
1  [(41.159826, -8.639847), (41.159871, -8.640351…  (41.159826, -8.639847)
2  [(41.140359, -8.612964), (41.14035, -8.613378)…  (41.140359, -8.612964)
3  [(41.151951, -8.574678), (41.151942, -8.574705…  (41.151951, -8.574678)
4  [(41.18049, -8.645994), (41.180517, -8.645949)…   (41.18049, -8.645994)

                  END_LOC
0  (41.154489, -8.630838)
1   (41.170671, -8.66574)
2    (41.14053, -8.61597)
3  (41.142915, -8.607996)
4  (41.178087, -8.687268)
```

Let's create our map

[20]:
```python
# Answer = clusters of color red represent the popular pick up regions for␣
↪stand based trips
q3_Map = folium.Map(location = [41.15, -8.62], zoom_start = 14)

colors = {"OTHER" : "green",
         "STAND_BASED" : "red",
         "CENTRAL_BASED" : "blue"}

q3_data = data[["CALL_TYPE", "START_LOC"]]

for index, row in q3_data.iterrows():
    color = colors[row["CALL_TYPE"]]
    location = row["START_LOC"]
    folium.CircleMarker(location , color = color, radius = 1, weight = 2).
↪add_to(q3_Map)

q3_Map
```

[20]: <folium.folium.Map at 0x247831574f0>

2.1 TASK 4

2.1.1 Question 4 - Which regions of the city are the most common destinations on Mondays?

To answer to this question we need to use the TIMESTAMP column to extract the day each trip to

[21]:
```python
data.TIMESTAMP.iloc[0]
```

[21]: 1372636858

[22]:
```
#finding the index of each day of the week
datetime.datetime.fromtimestamp(data.TIMESTAMP.iloc[0]).weekday()
```

[22]: 6

[23]:
```
day_names = list(calendar.day_name)
day_names
```

[23]: ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']

[24]:
```
#creating a list of weekdays and then adding another column in the data
get_day = lambda timestamp : day_names[datetime.datetime.
 ↪fromtimestamp(timestamp).weekday()]
data["WEEK_DAY"] = data.TIMESTAMP.apply(get_day)
data.head()
```

[24]:
```
               TRIP_ID    CALL_TYPE   TIMESTAMP  \
0  1372636858620000589        OTHER  1372636858
1  1372637303620000596  STAND_BASED  1372637303
2  1372636951620000320        OTHER  1372636951
3  1372636854620000520        OTHER  1372636854
4  1372637091620000337        OTHER  1372637091

                                           TRIP_PATH             START_LOC  \
0  [(41.141412, -8.618643), (41.141376, -8.618499…  (41.141412, -8.618643)
1  [(41.159826, -8.639847), (41.159871, -8.640351…  (41.159826, -8.639847)
2  [(41.140359, -8.612964), (41.14035, -8.613378)…  (41.140359, -8.612964)
3  [(41.151951, -8.574678), (41.151942, -8.574705…  (41.151951, -8.574678)
4  [(41.18049, -8.645994), (41.180517, -8.645949)…   (41.18049, -8.645994)

                  END_LOC WEEK_DAY
0  (41.154489, -8.630838)   Sunday
1   (41.170671, -8.66574)   Sunday
2    (41.14053, -8.61597)   Sunday
3  (41.142915, -8.607996)   Sunday
4  (41.178087, -8.687268)   Sunday
```

Let's get all the trips on Mondays

[26]:
```
q4_data = data[data.WEEK_DAY == "Monday"]
q4_data
```

[26]:
```
                 TRIP_ID      CALL_TYPE   TIMESTAMP  \
220  1372651639620000517  CENTRAL_BASED  1372651639
236  1372651473620000454  CENTRAL_BASED  1372651473
240  1372653466620000403    STAND_BASED  1372653466
```

```
241   1372652617620000477         OTHER  1372652617
242   1372652978620000167  CENTRAL_BASED  1372652978
…                     …             …          …
8324  1372726733620000108         OTHER  1372726733
8471  1372654126620000173  CENTRAL_BASED  1372654126
8597  1372724731620000297    STAND_BASED  1372724731
9302  1372694411620000529         OTHER  1372694411
9499  1372700793620000138    STAND_BASED  1372700793

                                               TRIP_PATH  \
220   [(41.164632, -8.583048), (41.164173, -8.582679…
236   [(41.158269, -8.638272), (41.158278, -8.638263…
240                            [(41.15187, -8.62704)]
241   [(41.15115, -8.6148), (41.151348, -8.614728), …
242   [(41.155389, -8.677953), (41.155389, -8.677962…
…                                                    …
8324  [(41.140404, -8.612955), (41.140404, -8.612946…
8471  [(41.181444, -8.583687), (41.180301, -8.586063…
8597  [(41.144616, -8.60652), (41.145021, -8.607168)…
9302  [(41.155542, -8.628552), (41.155713, -8.627823…
9499  [(41.148027, -8.619867), (41.148027, -8.619858…

                    START_LOC                END_LOC WEEK_DAY
220   (41.164632, -8.583048)  (41.148855, -8.585604)   Monday
236   (41.158269, -8.638272)  (41.148864, -8.585631)   Monday
240     (41.15187, -8.62704)    (41.15187, -8.62704)   Monday
241      (41.15115, -8.6148)   (41.237523, -8.67024)   Monday
242   (41.155389, -8.677953)    (41.1462, -8.617698)   Monday
…                        …                      …       …
8324  (41.140404, -8.612955)  (41.147496, -8.609409)   Monday
8471  (41.181444, -8.583687)  (41.148891, -8.585613)   Monday
8597   (41.144616, -8.60652)  (41.105385, -8.640711)   Monday
9302  (41.155542, -8.628552)  (41.166216, -8.606403)   Monday
9499  (41.148027, -8.619867)   (41.146281, -8.61138)   Monday

[4752 rows x 7 columns]
```

Now Let's visualize the data on the map

[27]:
```
# Each cluster represents popular destinations on MONDAY
q4_map = folium.Map(location = [41.15, -8.62], zoom_start= 14)

for point in q4_data.END_LOC:
    folium.CircleMarker(location = point, color = "blue", radius= 1, weight=2).
 ↪add_to(q4_map)
q4_map
```

[27]:
```
<folium.folium.Map at 0x24783157b50>
```

2.1.2 Question 5 - What are the most common start and end points on Monday Mornings (from 6 am to 9 am)?

`To answer to this question we need to extract the hour information for each of the trips.`

Let's extract the hour column from TIMESTAMP column

[28]:
```python
extract_hour = lambda timestamp : datetime.datetime.fromtimestamp(timestamp).
 ↪hour

data["HOUR"] = data.TIMESTAMP.apply(extract_hour)
data.head()
```

[28]:
```
               TRIP_ID    CALL_TYPE   TIMESTAMP  \
0  1372636858620000589        OTHER  1372636858
1  1372637303620000596  STAND_BASED  1372637303
2  1372636951620000320        OTHER  1372636951
3  1372636854620000520        OTHER  1372636854
4  1372637091620000337        OTHER  1372637091

                                           TRIP_PATH               START_LOC  \
0  [(41.141412, -8.618643), (41.141376, -8.618499…  (41.141412, -8.618643)
1  [(41.159826, -8.639847), (41.159871, -8.640351…  (41.159826, -8.639847)
2  [(41.140359, -8.612964), (41.14035, -8.613378)…  (41.140359, -8.612964)
3  [(41.151951, -8.574678), (41.151942, -8.574705…  (41.151951, -8.574678)
4  [(41.18049, -8.645994), (41.180517, -8.645949)…   (41.18049, -8.645994)

                  END_LOC WEEK_DAY  HOUR
0  (41.154489, -8.630838)   Sunday    20
1   (41.170671, -8.66574)   Sunday    20
2    (41.14053, -8.61597)   Sunday    20
3  (41.142915, -8.607996)   Sunday    20
4  (41.178087, -8.687268)   Sunday    20
```

Let's get the trips which took place between 6 am to 9 am on Mondays

[29]:
```python
q5_data = data[(data.WEEK_DAY == "Monday") & (data.HOUR > 6) & (data.HOUR < 9)]
q5_data.head()
```

[29]:
```
                  TRIP_ID      CALL_TYPE   TIMESTAMP  \
1633  1372676644620000465  CENTRAL_BASED  1372676644
1637  1372676476620000189  CENTRAL_BASED  1372676476
1661  1372677215620000431  CENTRAL_BASED  1372677215
1671  1372677241620000295    STAND_BASED  1372677241
1673  1372677899620000230    STAND_BASED  1372677899

                                              TRIP_PATH  \
1633  [(41.182569, -8.604081), (41.182434, -8.604099…
1637  [(41.161671, -8.647911), (41.161797, -8.64783)…
```

```
1661  [(41.157252, -8.610624), (41.157198, -8.610219…
1671  [(41.162931, -8.584389), (41.16285, -8.584497)…
1673  [(41.147991, -8.619867), (41.148, -8.619867), …

                     START_LOC                 END_LOC WEEK_DAY  HOUR
1633  (41.182569, -8.604081)  (41.148765, -8.584236)   Monday     7
1637  (41.161671, -8.647911)  (41.161653, -8.647848)   Monday     7
1661  (41.157252, -8.610624)  (41.180886, -8.664048)   Monday     7
1671  (41.162931, -8.584389)  (41.175225, -8.586126)   Monday     7
1673  (41.147991, -8.619867)   (41.14431, -8.622288)   Monday     7
```

Now let's visualize this data

[30]:
```
#Answer : Red dots represent most popular START points and blue the most␣
 ↪popular END points
q5_map = folium.Map(location = [41.15, -8.62], zoom_start = 14)
put_start_loc = lambda loc : folium.CircleMarker(loc, color = "red", radius =␣
 ↪1, weight = 2).add_to(q5_map)
put_end_loc = lambda loc : folium.CircleMarker(loc, color= "blue", radius = 1,␣
 ↪weight = 2).add_to(q5_map)

q5_data.START_LOC.apply(put_start_loc)
q5_data.END_LOC.apply(put_end_loc)

q5_map
```

[30]: <folium.folium.Map at 0x247880c62e0>

2.2 TASK 5

2.2.1 Question 6 - Find out the rush hour in Porto

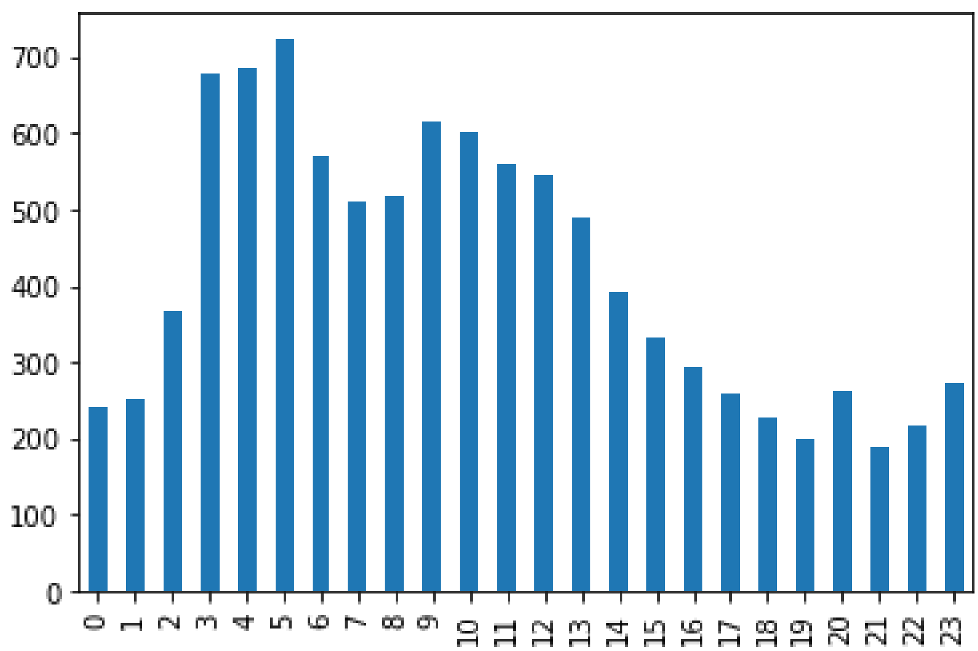
[31]:
```
data.HOUR.value_counts()
```

[31]:
```
5     724
4     684
3     678
9     616
10    602
6     569
11    558
12    544
8     519
7     511
13    490
14    390
2     368
15    331
```

```
16    293
23    274
20    264
17    258
1     252
0     243
18    227
22    217
19    198
21    190
Name: HOUR, dtype: int64
```

[33]:
```python
data.HOUR.value_counts().sort_index().plot.bar()
```

[33]: <AxesSubplot:>



2.2.2 Question 7 - Which streets has the more traffic during the rush hour

To answer to this question we need to use TRIP_PATH column to visualize each path as a line on Firstly, let's filter out the needed data.

[34]:
```python
q7_data = data[data.HOUR==5]
q7_data.head()
```

[34]:
```
                     TRIP_ID      CALL_TYPE   TIMESTAMP  \
937  1372669784620000455          OTHER  1372669784
```

```
947  1372669436620000285    STAND_BASED  1372669436
956  1372669271620000653    STAND_BASED  1372669271
965  1372669911620000455          OTHER  1372669911
972  1372669864620000050  CENTRAL_BASED  1372669864

                                              TRIP_PATH  \
937  [(41.166252, -8.607528), (41.165748, -8.608149…
947  [(41.145606, -8.610813), (41.145777, -8.610813…
956                           [(41.15772, -8.628399)]
965  [(41.164857, -8.608527), (41.16474, -8.608509)…
972  [(41.155866, -8.643798), (41.156811, -8.643591…

                  START_LOC                 END_LOC WEEK_DAY  HOUR
937  (41.166252, -8.607528)  (41.164875, -8.608536)   Monday     5
947  (41.145606, -8.610813)  (41.149242, -8.627157)   Monday     5
956   (41.15772, -8.628399)   (41.15772, -8.628399)   Monday     5
965  (41.164857, -8.608527)  (41.155362, -8.610237)   Monday     5
972  (41.155866, -8.643798)  (41.148882, -8.648964)   Monday     5
```

[35]:

```python
# Second trip
coords = q7_data.TRIP_PATH.iloc[1]

q7_map = folium.Map(location = [41.15, -8.62], zoom_start=13)
folium.PolyLine(coords,color= "red", opacity = 0.5, weight = 2).add_to(q7_map)

q7_map
```

[35]: <folium.folium.Map at 0x247885afd90>

[37]:

```python
# Doing a for loop for every trip. Also opacity is change to 0.05 for better␣
↪visualization
#to distinguish the most used streets during rush hour
q7_map = folium.Map(location = [41.15, -8.62], zoom_start=14)
for coords in q7_data.TRIP_PATH:
    folium.PolyLine(coords,color= "red", opacity = 0.05, weight = 2).
↪add_to(q7_map)

q7_map
```

[37]: <folium.folium.Map at 0x24788b6f1c0>

[ ]:

```python

```