7_Question_1

October 29, 2023

1 EE4211 Project Group 7

1.1 Data Cleaning & Exploring the Data

1.1.1 Part 1.1

From the LTA Datamall API User Guide, the values in the column “lot_type” represents which type of vehicle that the parking lot is meant for, where:

Type of lots: - C (for Cars) - H (for Heavy Vehicles) - Y (for Motorcycles)

1.1.2 Part 1.2

[ ]:
```
import requests
import json
import pandas as pd
import datetime
import time


def get_update_datetime(year,month,day,hour,minute,second):
    site = f'https://api.data.gov.sg/v1/transport/carpark-availability?
 ↪date_time={year}-{month.zfill(2)}-{day.zfill(2)}T{hour.zfill(2)}%3A{minute.
 ↪zfill(2)}%3A{second.zfill(2)}'
    response_API = requests.get(site)
    data = response_API.text
    data = json.loads(data)
    data = data["items"][0]["carpark_data"]
    df = pd.DataFrame(data)
    for heading in ("total_lots","lot_type","lots_available"):
        df[heading] = df["carpark_info"].apply(lambda x: x[0][heading])
    df = df.drop(["carpark_info"], axis=1)
    return df['update_datetime'][0]

year = "2022"
month = "4"
day = "12"
hour = "9"
minute = "30"
```

```
second = "0"

def compare(time1_data,time2_data):
    return time1_data == time2_data

time1_data =␣
 ↪get_update_datetime(year=year,month=month,day=day,hour=hour,minute=minute,second=second)
time2_data =␣
 ↪get_update_datetime(year=year,month=month,day=day,hour=hour,minute=minute,second="1")
print(compare(time1_data, time2_data))

time2_data =␣
 ↪get_update_datetime(year=year,month=month,day=day,hour=hour,minute="31",second=second)
print(compare(time1_data,time2_data))

time2_data =␣
 ↪get_update_datetime(year=year,month=month,day=day,hour="10",minute=minute,second=second)
print(compare(time1_data,time2_data))
```

```
True
True
False
```

Minimizing the range to check systematically From the comparison above, the "update time" is unchanged after one second.
It is also not changed after one minute.
This means that the data values are not updated every second and every minute.

However, the data is changed after one hour, this means that:
f = frequency at which the data values are updated
1 min < f <= 1 hour

Using the code below, we can find the frequency using a while loop

```
[ ]: year = "2022"
month = "4"
day = "12"
hour = "9"
minute = "0"
second = "0"
time1_data =␣
 ↪get_update_datetime(year=year,month=month,day=day,hour=hour,minute=minute,second=second)

check = True
while check:
    minute = str(int(minute)+1)
    print(minute)
    time2_data =␣
  ↪get_update_datetime(year=year,month=month,day=day,hour=hour,minute=minute,second=second)
    check = compare(time1_data,time2_data)


print("Values updated at",minute,"min")
```

```
1
2
3
4
5
6
7
8
9
10
11
12
Values updated at 12 min
```

Now we know that the values are first updated at 12 min mark, we can find the next time it updates and compare the time difference to get the frequency at which the data values are updated

[ ]:
```
year = "2022"
month = "4"
day = "12"
hour = "9"
minute = "12" # begin at 12 minute mark
second = "0"
time1_data =␣
  ↪get_update_datetime(year=year,month=month,day=day,hour=hour,minute=minute,second=second)

check = True
while check:
    minute = str(int(minute)+1)
    print(minute)
    if minute == "60":
        hour = str(int(hour)+1)
        minute == "0"
    time2_data =␣
  ↪get_update_datetime(year=year,month=month,day=day,hour=hour,minute=minute,second=second)
    check = compare(time1_data,time2_data)

print("Values updated at",minute,"min")
```

```
13
14
```

```
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
Values updated at 42 min
```

Since the data values are updated is at 12 minute mark, and the next time it updates is at 42 minute mark, the interval is 30 minutes.
Hence the frequency at which the data values are updated is 30 minutes

1.1.3 Part 1.3

(i) To determine for unique carparks, we need to remove duplicates from the 'carpark_number' column.

[ ]:
```
import requests
import json
import pandas as pd

year = "2022"
month = "4"
day = "12"
hour = "12"
minute = "30"
second = "0"
```

```
site = f'https://api.data.gov.sg/v1/transport/carpark-availability?
↪date_time={year}-{month.zfill(2)}-{day.zfill(2)}T{hour.zfill(2)}%3A{minute.
↪zfill(2)}%3A{second.zfill(2)}'
response_API = requests.get(site)
data = response_API.text
data = json.loads(data)
timestamp = data["items"][0]["timestamp"]
print(timestamp)
data = data["items"][0]["carpark_data"]
with open("EE4211data.json", 'w') as fp:
    json.dump(data, fp)
df = pd.read_json("EE4211data.json")
for heading in ("total_lots","lot_type","lots_available"):
    df[heading] = df["carpark_info"].apply(lambda x: x[0][heading])
df = df.drop(["carpark_info"], axis=1)

unique_carparks = df['carpark_number'].nunique()
print(f"Number of unique carparks: {unique_carparks} as of {timestamp}")
```

```
2022-04-12T12:29:27+08:00
Number of unique carparks: 1960 as of 2022-04-12T12:29:27+08:00
```

(ii) To check if the value varies with time, we will compare the findings from consecutive years.

```
[ ]: years = ["2019", "2020", "2021", "2022"]
     months = "4"
     day = "12"
     hour = "12"
     minute = "30"
     second = "0"

     for year in years:
         site = f'https://api.data.gov.sg/v1/transport/carpark-availability?
     ↪date_time={year}-{month.zfill(2)}-{day.zfill(2)}T{hour.zfill(2)}%3A{minute.
     ↪zfill(2)}%3A{second.zfill(2)}'
         print(site)
         response_API = requests.get(site)
         data = response_API.text
         data = json.loads(data)
         timestamp = data["items"][0]["timestamp"]
         print(timestamp)

         data = data["items"][0]["carpark_data"]
         with open("EE4211data.json", 'w') as fp:
             json.dump(data, fp)
         df = pd.read_json("EE4211data.json")
```

```
    for heading in ("total_lots","lot_type","lots_available"):
        df[heading] = df["carpark_info"].apply(lambda x: x[0][heading])
    df = df.drop(["carpark_info"], axis=1)
    unique_carparks = df['carpark_number'].nunique()
    print(f"Number of unique carparks: {unique_carparks}")
```

```
https://api.data.gov.sg/v1/transport/carpark-
availability?date_time=2019-04-12T12%3A30%3A00
2019-04-12T12:29:27+08:00
Number of unique carparks: 1843
https://api.data.gov.sg/v1/transport/carpark-
availability?date_time=2020-04-12T12%3A30%3A00
2020-04-12T12:29:27+08:00
Number of unique carparks: 1904
https://api.data.gov.sg/v1/transport/carpark-
availability?date_time=2021-04-12T12%3A30%3A00
2021-04-12T12:29:27+08:00
Number of unique carparks: 1936
https://api.data.gov.sg/v1/transport/carpark-
availability?date_time=2022-04-12T12%3A30%3A00
2022-04-12T12:29:27+08:00
Number of unique carparks: 1960
```

Q: Explain why this check is important (about 20 words).

It is a practice of data cleaning to ensure data consistency and accuracy. Sudden or unexpected changes in the number of unqiue carparks could indicate errors in data.

1.1.4 Part 1.4

A carpark may have malfunctioning sensors. There are many types of possible malfunctions. Identify one of these carparks that you believe has a malfunctioning sensors. Explain what the "malfunction" is in this case (about 20 words).

```
[ ]: import requests
     import json
     import pandas as pd

     # Change these values for different dates
     year = "2022"
     month = "4"
     day = "12"
     hour = "12"
     minute = "30"
     second = "0"

     site = f'https://api.data.gov.sg/v1/transport/carpark-availability?
      ↪date_time={year}-{month.zfill(2)}-{day.zfill(2)}T{hour.zfill(2)}%3A{minute.
      ↪zfill(2)}%3A{second.zfill(2)}'
```

```python
response_API = requests.get(site)
data = response_API.text
data = json.loads(data)
timestamp = data["items"][0]["timestamp"]

data = data["items"][0]["carpark_data"]
with open("EE4211data.json", 'w') as fp:
    json.dump(data, fp)
df = pd.read_json("EE4211data.json")
for heading in ("total_lots","lot_type","lots_available"):
    df[heading] = df["carpark_info"].apply(lambda x: x[0][heading])
df = df.drop(["carpark_info"], axis=1)

malfunction = pd.DataFrame()
for index, row in df.iterrows():
  lots_avail = int(row['lots_available'])
  total_lots = int(row['total_lots'])
  if lots_avail < 0 or lots_avail > total_lots:
    malfunction = pd.concat([malfunction, row.to_frame().T])

print("Rows identified with malfunctioning sensors:\n",malfunction)
```

```
Rows identified with malfunctioning sensors:
     carpark_number      update_datetime total_lots lot_type lots_available
549          Y49HV  2016-02-15T21:52:35          3        C             28
599          MR567  2016-02-15T21:52:35         38        C             94
648          P40L1  2016-02-15T21:52:35          2        C              6
649          P40L2  2016-02-15T21:52:35          3        C              6
926           SK24  2016-02-19T11:19:28        181        Y            200
927           SK24  2016-02-19T11:19:29        181        H            200
```

The rows above are identified with malfunctioning sensors. A carpark will have malfunctioning sensors if it reads a negative value or the lots available is more than the total number of lots in the carpark.

It is not possible to have a negative number of lots available, neither is it reasonbale to have more lots available than the total number of lots in the carpark.

Hence, carparks Y49HV, MR567, P40L1, P40L2 and SK24 are carparks having malfunctioning sensors

1.1.5 Part 1.5

Create a dataset of hourly carpark availability (i.e., use the ratio: lots available/total lots) from the raw data. Select a one month interval and plot the average (average across all carparks) hourly carpark availability against time for that interval. Identify any patterns in the plot (about 50 words). Note: You will have to decide what to do if there are no carpark readings for a certain hour. For example, some may impute the missing data or ignore it. You also have to decide if you want to (i) compute the ratios for each carpark and then average OR (ii) compute the total lots available and total total lots and take the ratio.

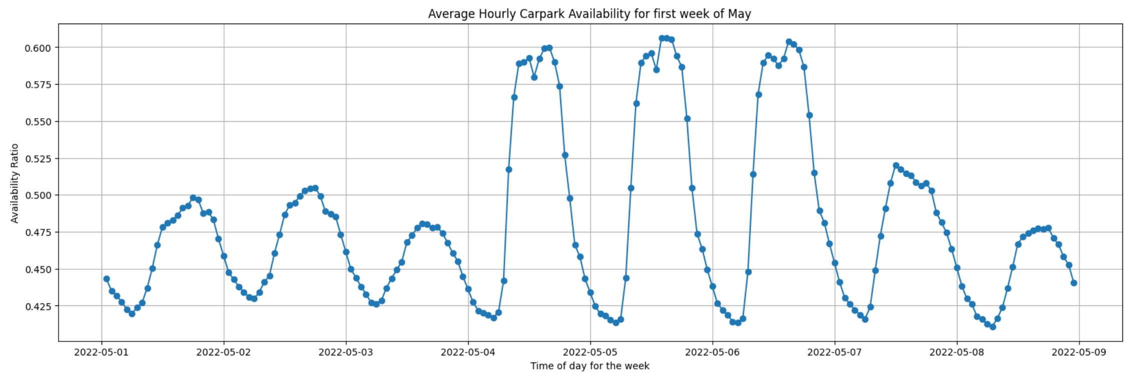
```
[ ]: import requests
     import json
     import pandas as pd
     import matplotlib.pyplot as plt

     def get_data(year, month, day, hour, minute, second):
         site = f'https://api.data.gov.sg/v1/transport/carpark-availability?
      ↪date_time={year}-{month.zfill(2)}-{day.zfill(2)}T{hour.zfill(2)}%3A{minute.
      ↪zfill(2)}%3A{second.zfill(2)}'
         response_API = requests.get(site)
         parsed_data = json.loads(response_API.text)
         df = pd.DataFrame()

         if "items" in parsed_data and parsed_data["items"]:
             data = parsed_data["items"][0]["carpark_data"]
             df = pd.DataFrame(data)
             for heading in ("total_lots", "lot_type", "lots_available"):
                 df[heading] = df["carpark_info"].apply(lambda x: x[0][heading])
             df = df.drop(["carpark_info"], axis=1)

         return df

     availability_ratios = []
     actual_hours = []

     for day in range(1, 32):
         for hour in range(0, 24):
             hourly_df = get_data("2022", "05", str(day), str(hour), "00", "00")

             if hourly_df.empty:
                     availability_ratios.append(None)
             else:
                     # Convert the 'lots_available' and 'total_lots' columns to␣
      ↪numeric values
                     hourly_df['lots_available'] = pd.
      ↪to_numeric(hourly_df['lots_available'], errors='coerce')
                     hourly_df['total_lots'] = pd.to_numeric(hourly_df['total_lots'],␣
      ↪errors='coerce')

                     # Compute the availability ratio
                     hourly_df['availability_ratio'] = hourly_df['lots_available'] /␣
      ↪hourly_df['total_lots']
                     availability_ratios.append(hourly_df['availability_ratio'].mean())

             actual_hours.append([day,hour])
```

```
[ ]: # Interpolate
def interpolate_data(l):
    s = pd.Series(l)
    s = s.interpolate()
    res = s.tolist()
    return res

availability_ratios_processed = interpolate_data(availability_ratios)

from datetime import datetime, timedelta

def conv_timestamp(day,hour):
    year = 2022
    month = 5
    timestamp = datetime(year, month, day, hour, 0, 0)
    return timestamp

date = [ conv_timestamp(i[0],i[1]) for i in actual_hours]
```

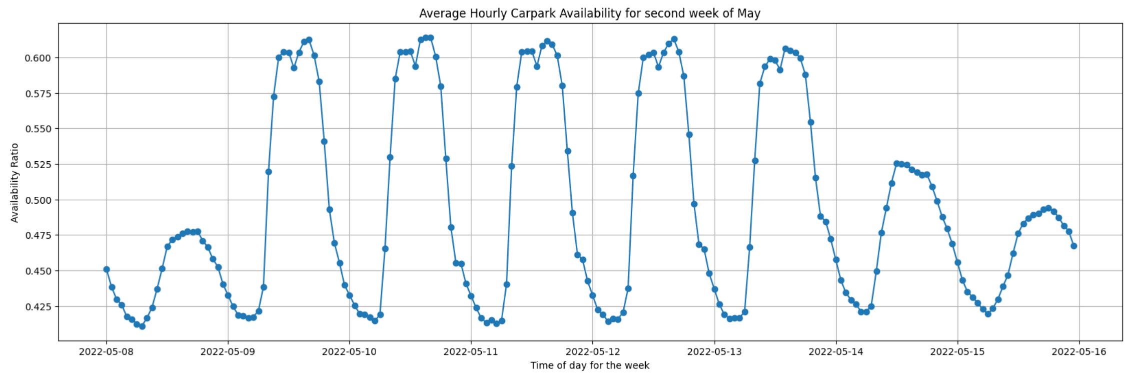
Visualising the trends on weekly basis first before combining to the whole month view

The plots are from sunday to sunday for each week

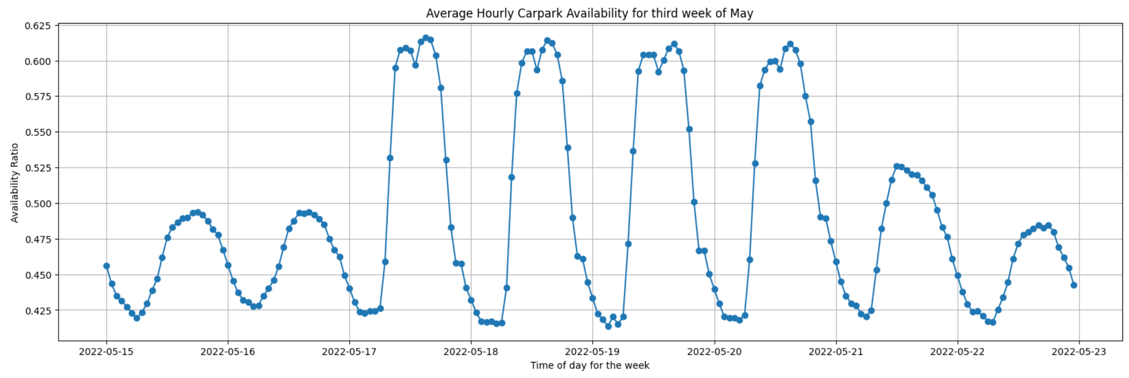
```
[ ]: # Plotting
plt.figure(figsize=(20,6))
plt.plot(date[:192], availability_ratios_processed[:192], '-o')
plt.title("Average Hourly Carpark Availability for first week of May")
plt.xlabel("Time of day for the week")
plt.ylabel("Availability Ratio")
plt.grid(True)
plt.show()
```



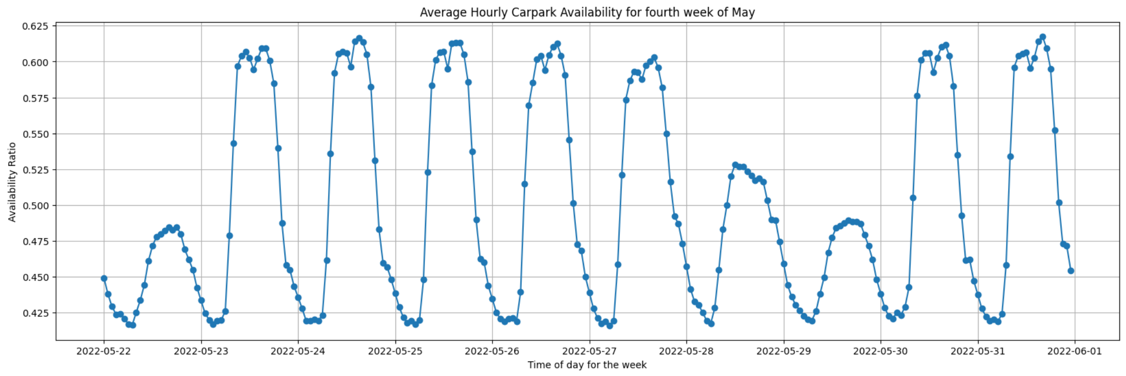
```
[ ]: # Plotting
plt.figure(figsize=(20,6))
plt.plot(date[168:360], availability_ratios_processed[168:360], '-o')
plt.title("Average Hourly Carpark Availability for second week of May")
plt.xlabel("Time of day for the week")
plt.ylabel("Availability Ratio")
plt.grid(True)
plt.show()
```



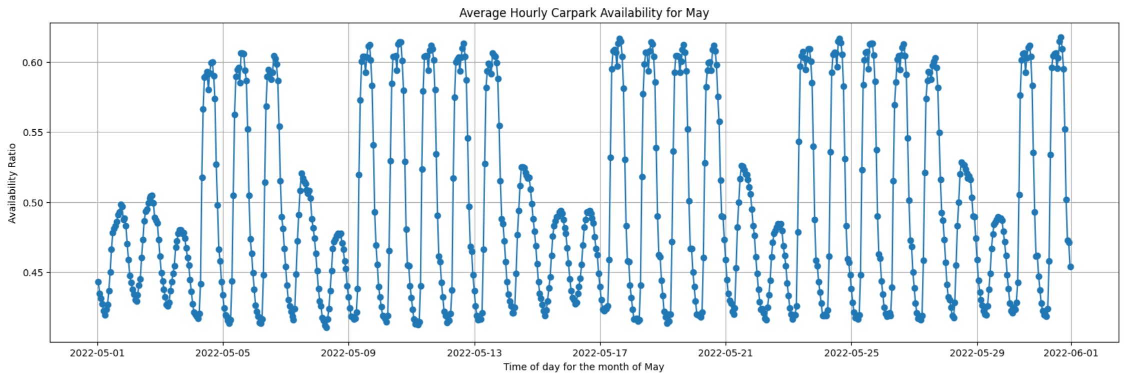
[ ]:
```
# Plotting
plt.figure(figsize=(20,6))
plt.plot(date[336:528], availability_ratios_processed[336:528], '-o')
plt.title("Average Hourly Carpark Availability for third week of May")
plt.xlabel("Time of day for the week")
plt.ylabel("Availability Ratio")
plt.grid(True)
plt.show()
```



[ ]:
```
# Plotting
plt.figure(figsize=(20,6))
plt.plot(date[504:], availability_ratios_processed[504:], '-o')
plt.title("Average Hourly Carpark Availability for fourth week of May")
plt.xlabel("Time of day for the week")
plt.ylabel("Availability Ratio")
plt.grid(True)
plt.show()
```



[ ]:
```
# Plotting
plt.figure(figsize=(20,6))
plt.plot(date, availability_ratios_processed, '-o')
plt.title("Average Hourly Carpark Availability for May")
plt.xlabel("Time of day for the month of May")
plt.ylabel("Availability Ratio")
plt.grid(True)
plt.show()
```



Pattern observed: It can be observed that the carpark availability ratio generally peaks around 0.60 during the weekdays, while dropping to approximately 0.50 during the weekends. However, there are exceptions that contradict the pattern. 2, 3, and 16 May are public holidays that occur on weekdays/ public holiday on weekend carried over to weekday, resulting in lower availability. Additionally, a daily trend where availability is lower throughout nighttime and higher during midday is evident from the graph.

1.1.6 Part 1.6

Intuitively, we expect that carpark availability across certain carparks to be correlated. For example, many housing carparks would experience higher carpark availability during working hours. Using the same interval chosen in 1.5, pick a carpark and find the carpark that is most correlated to it (in terms of carpark availability). State the type of correlation used (e.g. Spearman, Pearson, etc).

[ ]:
```
import requests
import json
import pandas as pd
import heapq
import random

def get_data(year, month, day, hour, minute, second):
    site = f'https://api.data.gov.sg/v1/transport/carpark-availability?
 ↪date_time={year}-{month.zfill(2)}-{day.zfill(2)}T{hour.zfill(2)}%3A{minute.
 ↪zfill(2)}%3A{second.zfill(2)}'
    response_API = requests.get(site)
    data = response_API.text
    parsed_data = json.loads(data)
    df = pd.DataFrame()

    if "items" in parsed_data and parsed_data["items"]:
        data = parsed_data["items"][0]["carpark_data"]
        with open("EE4211data.json", 'w') as fp:
            json.dump(data, fp)
        df = pd.read_json("EE4211data.json")
        for heading in ("total_lots", "lot_type", "lots_available"):
            df[heading] = df["carpark_info"].apply(lambda x: x[0][heading])
        df = df.drop(["carpark_info"], axis=1)
    return df

def get_carpark_names(year, month, day):
    df = get_data(year, month, day, "12", "00", "00")

    if 'carpark_number' in df.columns:
        return df.carpark_number.drop_duplicates()
    else:
        print("Warning: 'carpark_number' column not found in the DataFrame!")
        return pd.Series()

carpark_names = get_carpark_names("2022", "5", "15")
overall_df = pd.DataFrame(0, columns=[hour for hour in range(0,24)], index=␣
 ↪[carpark for _,carpark in carpark_names.items()])

year = "2022"
month = "5"
```

```python
print("Gathering data for a month...")
for day in range(1,32):  # May has 31 days
    print("Processing day ", str(day))
    for hour in range(0, 24):  # Include all 24 hours
        hourly_df = get_data(year, month, str(day), str(hour), "00", "00")
        for _, row in hourly_df.iterrows():
            curr_carpark = row["carpark_number"]
            if curr_carpark in overall_df.index:
                overall_df.at[curr_carpark, hour] += int(row['lots_available'])

def gen_correlation(chosen_carpark, occupancy_df):
    corr_data = {}
    chosen_df = occupancy_df.loc[chosen_carpark]
    for carpark, data in occupancy_df.iterrows():
        if chosen_carpark == carpark:
            continue
        corr_val = chosen_df.corr(data)  # Pearson correlation by default
        corr_data[carpark] = corr_val
    return corr_data

# Choose only one random carpark
random_carpark = random.choice(carpark_names)
corr_data = gen_correlation(random_carpark, overall_df)
top_5 = heapq.nlargest(5, corr_data, key=corr_data.get)
print("Carpark", random_carpark, "has 5 top correlation with", top_5)
```

```
Gathering data for a month…
Processing day  1
Processing day  2
Processing day  3
Processing day  4
Processing day  5
Processing day  6
Processing day  7
Processing day  8
Processing day  9
Processing day  10
Processing day  11
Processing day  12
Processing day  13
Processing day  14
Processing day  15
Processing day  16
Processing day  17
Processing day  18
Processing day  19
Processing day  20
Processing day  21
Processing day  22
Processing day  23
Processing day  24
Processing day  25
Processing day  26
Processing day  27
Processing day  28
Processing day  29
Processing day  30
Processing day  31
Carpark U64 has 5 top correlation with ['BJ56', 'U58', 'U52', 'J85M', 'CK38']
```

Pearson correlation was used in part 1.6

1.1.7 Part 1.7

Group Project Proposal for Question 3: Please include a short proposal (around 500 words) of what your team intends to do for the Group Proposed Project in Question 3. For the group project proposal, you may use additional datasets to supplement your analysis or look at unaggregated data, etc. See Question 3 below for more information about this. Please use markdown in the iPython notebook to present your proposal.

Project title: Forecasting car park electric vehicles charging station upgrades by region

As Singapore aims to phase out petrol and diesel vehicles by 2040, there is a need to provide accessible electric vehicle (EV) charging ports by upgrading the existing car park infrastructure. This is due to the fact that EV takes a longer time to charge as opposed to the way gasoline vehicles are 'refuelled'.

Car park upgrades are not instantaneous, and requires planning and prioritising based on the demands for the upgrade. At present, there is a lack of information with regards to EV ownership by region in Singapore. Through this preliminary analysis our group has identified that there might be a need to utilise other data sets that possibly correlates with EV ownership by region in order to generate a forecast for EV charging port upgrades in car parks.

Our group intends to utilise 3 datasets: Car park Availability, HDB car park information and Pollutant standards index (PSI).

The car park availability dataset provides information pertaining to hourly use of the car park which helps us identify the demand of the car parks. HDB car park information at present would provide us with information about the car park number and their coordinates, we then intend to utilise the coordinates to place the car park into regions within Singapore (North, South, East, West, Central). The PSI dataset would then allow us to analyse the NO2 levels of the regions in Singapore.

Once the data collection, cleaning and preprocessing is done, we intend to approach the problem by first; identifying the correlation between the available lots and the NO2 readings in the region. Then we will identify machine learning models to predict the NO2 in the following month.

This is done so that we are able to forecast regions in Singapore that have high NO2 output, by upgrading the infrastructure at those locations, it would encourage drivers to swap over to EV. Furthermore, the use of our solution is meant to be used continuously in the efforts to upgrade HDB car parks by reanalysing the changes in NO2 output in Singapore as carparks get upgraded, as the ‘highest’ NO2 region would not remain the same as some regions get upgraded over time.

The major assumptions that we have made in pursuit of this solution would be; when there is low availability and high NO2 output, there are many gasoline cars in the region. By upgrading these regions we are encouraging faster and more efficient adoption of EV.