7_Question_3

November 17, 2023

1 EE4211 Project Group 7

2 Note: Project was coded and tested on only google colab, please run using google colab

3 Question 3.1

PSI Pollutant index: https://beta.data.gov.sg/datasets/d_8a7850dc3993dc45f1620b9972c58d4d/view
Carpark availability: https://beta.data.gov.sg/datasets/d_ca933a644e55d34fe21f28b8052fac63/view
HDB Carpark Information: https://beta.data.gov.sg/datasets/d_23f946fa557947f93a8043bbef41dd09/view

[ ]:
```
##### Imports for this notebook ######
import time
import requests
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import lightgbm as lgb
from geopy.distance import great_circle
from pyproj import Proj, Transformer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler
```

3.1 1. Data collection

NO2 readings and carpark availability are obtained using the code below. HDB carpark information is a CSV file that can be downloaded directly from the website given above

3.1.1 Collecting data for NO2 readings

[ ]:
```python
def get_psi(year: str, month: str, day: str, hour: str, minute: str, second:␣
 ↪str):
    site = f'https://api.data.gov.sg/v1/environment/psi?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)
    items = data["items"][0]
    readings = items['readings']['no2_one_hour_max']
    #readings.pop('national')
    readings['datetime'] = pd.to_datetime(f"{year}-{month}-{day} {hour}:
 ↪{minute}:{second}")
    return readings

year = "2022"
month = "8"
day = "12"
hour = "9"
minute = "30"
second = "0"


# Loop over the months July and August
# Initialize an empty DataFrame for each month

for month in ["07","08"]:
    monthly_data = pd.DataFrame()

    # for the index of the collected data
    counter = [i for i in range(24*2)]

    # Check the data hourly
    for day in range(1, 32):  # Adjust this range for each month if needed
        print(f"Checking data for 2022-{month}-{str(day).zfill(2)}")
        for hour in range(0, 24):  # Hourly data
            # Collect data for each hour of the day
            data = get_psi("2022", month, str(day), str(hour), "00", "00")
            hourly_data = pd.DataFrame(data,index=counter)
            # Append the hourly data to the monthly DataFrame
            monthly_data = pd.concat([monthly_data, hourly_data])

            # Delay to avoid hitting the API rate limit
            time.sleep(1)  # Sleep for 1 second between requests
```

```
    monthly_data.to_csv(f'PSI_{month}.csv')
    print(f"Hourly data for 2022-{month} saved to CSV file.")
```

3.1.2 Collecting data for carpark availability

```
[ ]: 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)
             df['datetime'] = pd.to_datetime(f"{year}-{month}-{day} {hour}:{minute}:
     ↪{second}")
             for heading in ("total_lots", "lot_type", "lots_available"):
                 df[heading] = df["carpark_info"].apply(lambda x: x[0][heading])
             # Drop the carpark_info and update_datetime columns
             df = df.drop(columns=['carpark_info', 'update_datetime'])
         return df

     for month in ["07","08"]:
         monthly_data = pd.DataFrame()

         for day in range(1, 32):  # Adjust this range for each month if needed
             for hour in range(0, 24):  # Hourly data
                 print(f"Checking data for 2022-{month}-{day:02d} {hour:02d}:00:00")

                 hourly_data = get_data("2022", month, str(day), str(hour), "00",␣
     ↪"00")

                 if hourly_data.empty:
                     # Create a placeholder DataFrame
                     hourly_data = pd.DataFrame({
                         'datetime': pd.to_datetime(f"2022-{month}-{day} {hour}:00:
     ↪00"),
                         'carpark_number': 'Unknown',  # Use 'Unknown' or a similar␣
     ↪placeholder
                         'total_lots': None,
                         'lot_type': None,
                         'lots_available': None
                     }, index=[0])

                 monthly_data = pd.concat([monthly_data, hourly_data])
```

```
        time.sleep(1)  # Sleep to avoid hitting rate limits

    monthly_data.to_csv(f'CarParkData_2022_{month}.csv', index=False)
    print(f"Hourly data for 2022-{month} saved to CSV file.")
```

3.2 2. Data Cleaning and Feature Engineering

3.2.1 Processing HDBCarparkInformation

Obtain latitude and longitude for each carpark

[ ]:
```
# Decimal Degrees

# Read the CSV file into a DataFrame
df = pd.read_csv('HDBCarparkInformation.csv')

# Print the column names to check if 'X Coord' and 'Y Coord' are present
print(df.columns)

# Initialize the projection transformer
transformer = Transformer.from_crs("epsg:3414", "epsg:4326", always_xy=True)

# Function to convert SVY21 to WGS84
def svy21_to_wgs84(x, y):
    lon, lat = transformer.transform(x, y)
    return lon, lat

# Replace 'X Coord' and 'Y Coord' with the actual column names
df['Longitude'], df['Latitude'] = zip(*df.apply(lambda row:␣
 ↪svy21_to_wgs84(row['x_coord'], row['y_coord']), axis=1))

# Save the updated DataFrame to a new CSV file
#df.to_csv('HDBCarparkInformation_lat_long.csv', index=False)

# Print the new DataFrame
print(df)
```

```
Index(['car_park_no', 'address', 'x_coord', 'y_coord', 'car_park_type',
       'type_of_parking_system', 'short_term_parking', 'free_parking',
       'night_parking', 'car_park_decks', 'gantry_height',
       'car_park_basement'],
      dtype='object')
     car_park_no                                     address     x_coord  \
0            ACB  BLK 270/271 ALBERT CENTRE BASEMENT CAR PARK  30314.7936
1            ACM                        BLK 98A ALJUNIED CRESCENT  33758.4143
2            AH1                              BLK 101 JALAN DUSUN  29257.7203
3           AK19                 BLOCK 253 ANG MO KIO STREET 21  28185.4359
4           AK31               BLK 302/348 ANG MO KIO STREET 31  29482.0290
```

```
…          …                                      …         …
2209        Y81M                  BLK 476 YISHUN STREET 44  30194.3665
2210        Y82M                  BLK 478 YISHUN STREET 42  29935.5818
2211        Y83L                  BLK 382 YISHUN STREET 31  29649.6679
2212        Y83M                  BLK 382 YISHUN STREET 31  29505.6858
2213          Y9              BLK 747/752 YISHUN STREET 72  28077.2305

         y_coord          car_park_type type_of_parking_system  \
0     31490.4942      BASEMENT CAR PARK     ELECTRONIC PARKING
1     33695.5198  MULTI-STOREY CAR PARK     ELECTRONIC PARKING
2     34500.3599       SURFACE CAR PARK     ELECTRONIC PARKING
3     39012.6664       SURFACE CAR PARK         COUPON PARKING
4     38684.1754       SURFACE CAR PARK         COUPON PARKING
…          …                    …                    …
2209  45535.3488  MULTI-STOREY CAR PARK     ELECTRONIC PARKING
2210  45679.7181  MULTI-STOREY CAR PARK     ELECTRONIC PARKING
2211  45882.3415       SURFACE CAR PARK     ELECTRONIC PARKING
2212  45847.8567  MULTI-STOREY CAR PARK     ELECTRONIC PARKING
2213  45507.8047       SURFACE CAR PARK     ELECTRONIC PARKING

     short_term_parking             free_parking night_parking  \
0             WHOLE DAY                       NO           YES
1             WHOLE DAY  SUN & PH FR 7AM-10.30PM           YES
2             WHOLE DAY  SUN & PH FR 7AM-10.30PM           YES
3               7AM-7PM                       NO            NO
4                    NO                       NO            NO
…                  …                    …           …
2209          WHOLE DAY                       NO           YES
2210          WHOLE DAY  SUN & PH FR 7AM-10.30PM           YES
2211                 NO                       NO            NO
2212          WHOLE DAY                       NO           YES
2213          WHOLE DAY  SUN & PH FR 7AM-10.30PM           YES

      car_park_decks  gantry_height car_park_basement   Longitude  Latitude
0                  1           1.80                 Y  103.854118  1.301063
1                  5           2.10                 N  103.885061  1.321004
2                  0           0.00                 N  103.844620  1.328283
3                  0           0.00                 N  103.834985  1.369091
4                  0           0.00                 N  103.846636  1.366120
…               …            …                …       …       …
2209              15           2.15                 N  103.853037  1.428080
2210              11           2.15                 N  103.850712  1.429386
2211               0           0.00                 N  103.848142  1.431218
2212              16           2.15                 N  103.846849  1.430906
2213               0           4.50                 N  103.834013  1.427831

[2214 rows x 14 columns]
```

Generate region for each carpark

[ ]:
```python
# Latitude and longitude is based on the metadata in the PSI pollutant index API
def return_region(lat, long):
    locations = {
        'west': [1.35735, 103.7],
        'east': [1.35735, 103.94],
        'central': [1.35735, 103.82],
        'south': [1.29587, 103.82],
        'north': [1.41803, 103.82]
    }

    distances = {region: great_circle((lat, long), coord).kilometers for␣
 ↪region, coord in locations.items()}
    closest_region = min(distances, key=distances.get)

    return closest_region

# Read the CSV file into a DataFrame
#df = pd.read_csv("HDBCarparkInformation_Lat_Long.csv")

# Apply the return_region function to each row
df['region'] = df.apply(lambda row: return_region(row['Latitude'],␣
 ↪row['Longitude']), axis=1)

# Save the updated DataFrame to a new CSV file
df.to_csv('HDBCarparkInformation_with_region.csv', index=False)

# Print the DataFrame with the added 'region' column
print(df)
```

```
     car_park_no                                       address     x_coord  \
0            ACB  BLK 270/271 ALBERT CENTRE BASEMENT CAR PARK  30314.7936
1            ACM                       BLK 98A ALJUNIED CRESCENT  33758.4143
2            AH1                             BLK 101 JALAN DUSUN  29257.7203
3           AK19                   BLOCK 253 ANG MO KIO STREET 21  28185.4359
4           AK31                 BLK 302/348 ANG MO KIO STREET 31  29482.0290
…            …                                               …           …
2209        Y81M                        BLK 476 YISHUN STREET 44  30194.3665
2210        Y82M                        BLK 478 YISHUN STREET 42  29935.5818
2211        Y83L                        BLK 382 YISHUN STREET 31  29649.6679
2212        Y83M                        BLK 382 YISHUN STREET 31  29505.6858
2213          Y9                    BLK 747/752 YISHUN STREET 72  28077.2305

         y_coord          car_park_type type_of_parking_system  \
0     31490.4942      BASEMENT CAR PARK     ELECTRONIC PARKING
1     33695.5198  MULTI-STOREY CAR PARK     ELECTRONIC PARKING
2     34500.3599       SURFACE CAR PARK     ELECTRONIC PARKING
3     39012.6664       SURFACE CAR PARK         COUPON PARKING
```

```
4     38684.1754       SURFACE CAR PARK        COUPON PARKING
…          …                  …                    …
2209  45535.3488  MULTI-STOREY CAR PARK     ELECTRONIC PARKING
2210  45679.7181  MULTI-STOREY CAR PARK     ELECTRONIC PARKING
2211  45882.3415       SURFACE CAR PARK     ELECTRONIC PARKING
2212  45847.8567  MULTI-STOREY CAR PARK     ELECTRONIC PARKING
2213  45507.8047       SURFACE CAR PARK     ELECTRONIC PARKING

     short_term_parking             free_parking night_parking  \
0             WHOLE DAY                       NO           YES
1             WHOLE DAY  SUN & PH FR 7AM-10.30PM           YES
2             WHOLE DAY  SUN & PH FR 7AM-10.30PM           YES
3               7AM-7PM                       NO            NO
4                    NO                       NO            NO
…                     …                        …             …
2209          WHOLE DAY                       NO           YES
2210          WHOLE DAY  SUN & PH FR 7AM-10.30PM           YES
2211                 NO                       NO            NO
2212          WHOLE DAY                       NO           YES
2213          WHOLE DAY  SUN & PH FR 7AM-10.30PM           YES

      car_park_decks  gantry_height car_park_basement   Longitude  Latitude  \
0                  1           1.80                 Y  103.854118  1.301063
1                  5           2.10                 N  103.885061  1.321004
2                  0           0.00                 N  103.844620  1.328283
3                  0           0.00                 N  103.834985  1.369091
4                  0           0.00                 N  103.846636  1.366120
…                  …              …                 …           …         …
2209              15           2.15                 N  103.853037  1.428080
2210              11           2.15                 N  103.850712  1.429386
2211               0           0.00                 N  103.848142  1.431218
2212              16           2.15                 N  103.846849  1.430906
2213               0           4.50                 N  103.834013  1.427831

       region
0       south
1        east
2     central
3     central
4     central
…           …
2209    north
2210    north
2211    north
2212    north
2213    north

[2214 rows x 15 columns]
```

3.2.2 Processing CarParkData.csv

[ ]:
```python
# Interpolate and save to a dataframe
def interpolate_data(df):
    interpolated_column = df['lots_available'].
 ↪interpolate(limit_direction='both')
    df['lots_available'] = interpolated_column
    return df

# Splitting the 'datetime' column into 'date' and 'time'
def get_date_hour(df):
  df['datetime'] = pd.to_datetime(df['datetime'])
  df['Hour'] = df['datetime'].dt.hour
  df['Day'] = df['datetime'].dt.day
  return df

# For carpark absolute availability difference
def get_avail_difference(df):
    df_copy = df.copy()
    df_copy['avail_difference'] = df_copy['lots_available'].diff().abs()
    # Calculate average avail_difference
    avg_diff = df_copy['avail_difference'].mean()
    # Fill nulls with average instead of lots_available
    df_copy['avail_difference'].fillna(avg_diff, inplace=True)
    return df_copy
```

[ ]:
```python
##############################################################################
##############          Cleaning up the data          #########################
##############################################################################

############################### July ####################################
# Obtain availability for July
Carpark_July = pd.read_csv("CarParkData_2022_07.csv")
Carpark_July = get_date_hour(Carpark_July)

# Obtain only HDB carparks
df1 = Carpark_July
df2 = pd.read_csv("HDBCarparkInformation_with_region.csv")

# Specify the common columns in each DataFrame
common_column_df1 = 'carpark_number'
common_column_df2 = 'car_park_no'

# Perform the join based on the common columns
Carpark_July = pd.merge(df1, df2, left_on=common_column_df1,␣
 ↪right_on=common_column_df2, how='inner')
```

```
Carpark_July = Carpark_July.drop(columns = ['lot_type', 'address','x_coord',␣
↪'y_coord', 'car_park_type', 'type_of_parking_system', 'short_term_parking',␣
↪'free_parking', 'night_parking', 'car_park_decks', 'gantry_height',␣
↪'car_park_basement', 'car_park_no'])
Carpark_July = Carpark_July[Carpark_July['Hour'] != 0]
Carpark_July = Carpark_July.reset_index(drop=True)

################################## August ###################################
# Obtain availability for August
Carpark_Aug = pd.read_csv("CarParkData_2022_08.csv")
Carpark_Aug = get_date_hour(Carpark_Aug)

# Obtain only HDB carparks
df1 = Carpark_Aug
df2 = pd.read_csv("HDBCarparkInformation_with_region.csv")

# Specify the common columns in each DataFrame
common_column_df1 = 'carpark_number'
common_column_df2 = 'car_park_no'

# Perform the join based on the common columns
Carpark_Aug = pd.merge(df1, df2, left_on=common_column_df1,␣
↪right_on=common_column_df2, how='inner')
Carpark_Aug = Carpark_Aug.drop(columns = ['lot_type', 'address','x_coord',␣
↪'y_coord', 'car_park_type', 'type_of_parking_system', 'short_term_parking',␣
↪'free_parking', 'night_parking', 'car_park_decks', 'gantry_height',␣
↪'car_park_basement', 'car_park_no'])
Carpark_Aug = Carpark_Aug[Carpark_Aug['Hour'] != 0]
Carpark_Aug = Carpark_Aug.reset_index(drop=True)
```

[ ]:
```
##############################################################################
######################        Grouping      ##################################
##############################################################################



############################### July ####################################
# July Carpark grouped by region
July_grouping = Carpark_July.groupby([Carpark_July['region'],␣
↪Carpark_July['Day'], Carpark_July['Hour']]).sum()
July_grouping = interpolate_data(July_grouping)

# July Carpark for each region
july_north = get_avail_difference(July_grouping.loc['north'])
july_south = get_avail_difference(July_grouping.loc['south'])
july_east = get_avail_difference(July_grouping.loc['east'])
july_west = get_avail_difference(July_grouping.loc['west'])
july_central = get_avail_difference(July_grouping.loc['central'])
```

```
################################### August ####################################
# Aug Carpark grouped by region
Aug_grouping = Carpark_Aug.groupby([Carpark_Aug['region'], Carpark_Aug['Day'],␣
 ↪Carpark_Aug['Hour']]).sum()
Aug_grouping = interpolate_data(Aug_grouping)

# Aug Carpark for each region
aug_north = get_avail_difference(Aug_grouping.loc['north'])
aug_south = get_avail_difference(Aug_grouping.loc['south'])
aug_east = get_avail_difference(Aug_grouping.loc['east'])
aug_west = get_avail_difference(Aug_grouping.loc['west'])
aug_central = get_avail_difference(Aug_grouping.loc['central'])


############################## July Overall ################################
july_overall = Carpark_July.groupby([Carpark_July['Day'],␣
 ↪Carpark_July['Hour']]).sum()
july_overall_diff = get_avail_difference(july_overall)
```

```
<ipython-input-13-87fa53918249>:8: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  July_grouping = Carpark_July.groupby([Carpark_July['region'],
Carpark_July['Day'], Carpark_July['Hour']]).sum()
<ipython-input-13-87fa53918249>:21: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  Aug_grouping = Carpark_Aug.groupby([Carpark_Aug['region'], Carpark_Aug['Day'],
Carpark_Aug['Hour']]).sum()
<ipython-input-13-87fa53918249>:33: FutureWarning: The default value of
numeric_only in DataFrameGroupBy.sum is deprecated. In a future version,
numeric_only will default to False. Either specify numeric_only or select only
columns which should be valid for the function.
  july_overall = Carpark_July.groupby([Carpark_July['Day'],
Carpark_July['Hour']]).sum()
```

3.2.3 Processing NO2 Readings

[ ]:
```
#July

# Path to the CSV file
filename = 'PSI_07.csv'
```

```
# Read the data from the CSV file
JulyNO2 = pd.read_csv(filename,index_col=0)

# Convert 'datetime' to pandas datetime format and keep the original column
def extract_day_hour(df):
    df['datetime'] = pd.to_datetime(df['datetime'])
    # Splitting the 'datetime' column into 'date' and 'time'
    df['Hour'] = df['datetime'].dt.hour
    df['Day'] = df['datetime'].dt.day
    return df

JulyNO2 = extract_day_hour(JulyNO2)

# Calculate the average of the five regions and create a new column 'avg_no2'
regions = ['north', 'south', 'east', 'west', 'central']
JulyNO2['avg_no2'] = JulyNO2[regions].mean(axis=1)

#Dropping all hour 0 value to match the carpark data
JulyNO2 = JulyNO2[JulyNO2['Hour'] != 0]
JulyNO2 = JulyNO2.reset_index(drop=True)

JulyNO2
```

[ ]:

```
     west  east  central  south  north             datetime  Hour  Day  avg_no2
0      27    27       31     19     18  2022-07-01 01:00:00     1    1     24.4
1      29    30       30     20     18  2022-07-01 02:00:00     2    1     25.4
2      26    33       21     21     26  2022-07-01 03:00:00     3    1     25.4
3      15    26       24     25     21  2022-07-01 04:00:00     4    1     22.2
4       6    17       26     26     24  2022-07-01 05:00:00     5    1     19.8
..    …     …        …      …      …                    …     …    …        …
708    27    20       19     19     12  2022-07-31 19:00:00    19   31     19.4
709    25    28       32     32     15  2022-07-31 20:00:00    20   31     26.4
710    23    35       30     30     36  2022-07-31 21:00:00    21   31     30.8
711    28    39       26     26     13  2022-07-31 22:00:00    22   31     26.4
712    26    44       32     32     39  2022-07-31 23:00:00    23   31     34.6

[713 rows x 9 columns]
```

[ ]:

```
#August
# Path to the CSV file August
filename = 'PSI_08.csv'

# Read the data from the CSV file
AugNO2 = pd.read_csv(filename,index_col=0)

# Convert 'datetime' to pandas datetime format and keep the original column
def extract_day_hour(df):
```

```
    df['datetime'] = pd.to_datetime(df['datetime'])
    # Splitting the 'datetime' column into 'date' and 'time'
    df['Hour'] = df['datetime'].dt.hour
    df['Day'] = df['datetime'].dt.day
    return df

AugNO2 = extract_day_hour(AugNO2)

# Calculate the average of the five regions and create a new column 'avg_no2'
regions = ['north', 'south', 'east', 'west', 'central']
AugNO2['avg_no2'] = AugNO2[regions].mean(axis=1)

#Dropping all hour 0 value to match the carpark data
AugNO2 = AugNO2[AugNO2['Hour'] != 0]
AugNO2 = AugNO2.reset_index(drop=True)

AugNO2
```

[ ]:

```
     west  east  central  south  north            datetime  Hour  Day  avg_no2
0      29    34       28     28     30 2022-08-01 01:00:00     1    1     29.8
1      25    28       30     30     26 2022-08-01 02:00:00     2    1     27.8
2      27    36       29     29     14 2022-08-01 03:00:00     3    1     27.0
3      27    38       27     27     12 2022-08-01 04:00:00     4    1     26.2
4      25    32       24     24     11 2022-08-01 05:00:00     5    1     23.2
..    …   …      …    …   …                  …   …  …      …
708    10    38       26     21     47 2022-08-31 19:00:00    19   31     28.4
709    23    39       39     27     55 2022-08-31 20:00:00    20   31     36.6
710    37    49       31     28     32 2022-08-31 21:00:00    21   31     35.4
711    45    39       28     23     28 2022-08-31 22:00:00    22   31     32.6
712    44    32       31     20     34 2022-08-31 23:00:00    23   31     32.2

[713 rows x 9 columns]
```

3.3 3. Correlation Observations

3.3.1 3a. Correlation Observation of CarPark Abs Availability Difference and NO2 (OVERALL)

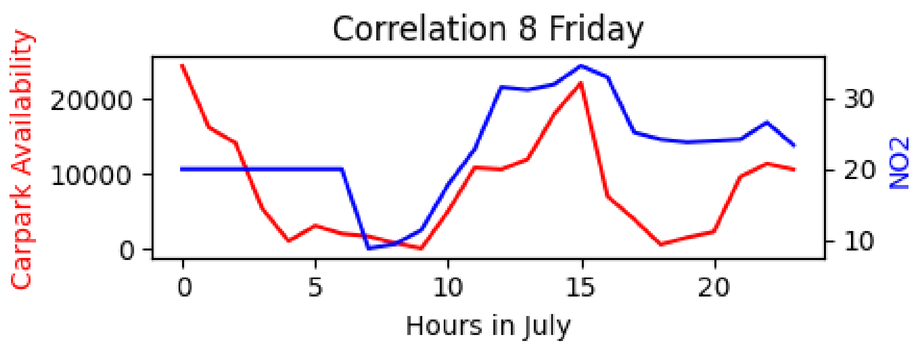
[ ]:

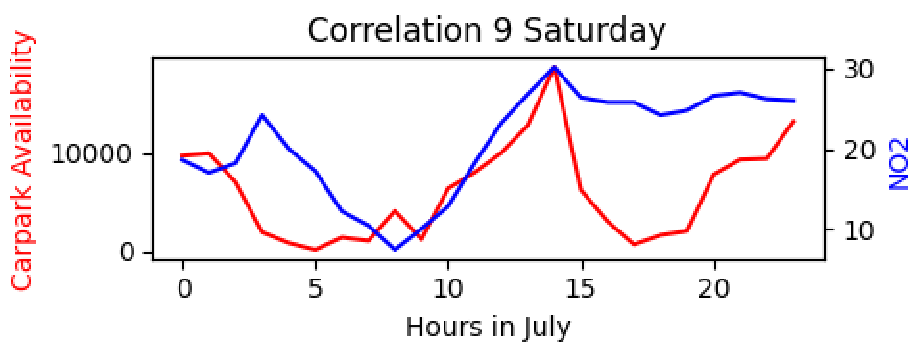
```
print('OVERALL PLOTS')
fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_overall_diff['avail_difference'][192:216], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
```

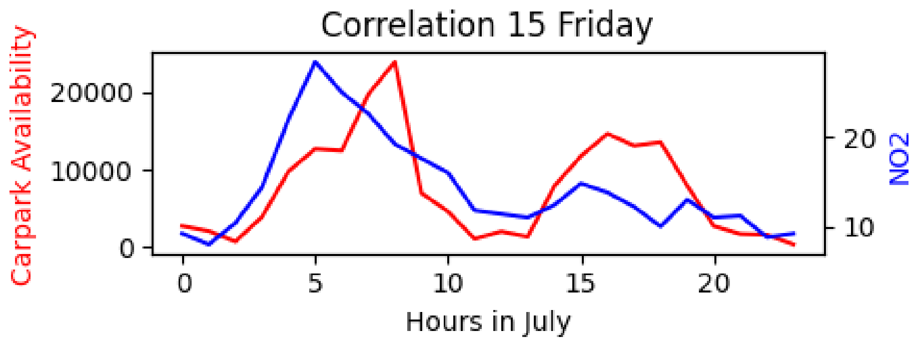
```
ax2.plot(range(24), JulyNO2['avg_no2'][192:216], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation 8 Friday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_overall_diff['avail_difference'][216:240], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['avg_no2'][216:240], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation 9 Saturday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_overall_diff['avail_difference'][360:384], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['avg_no2'][360:384], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation 15 Friday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_overall_diff['avail_difference'][408:432], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
```

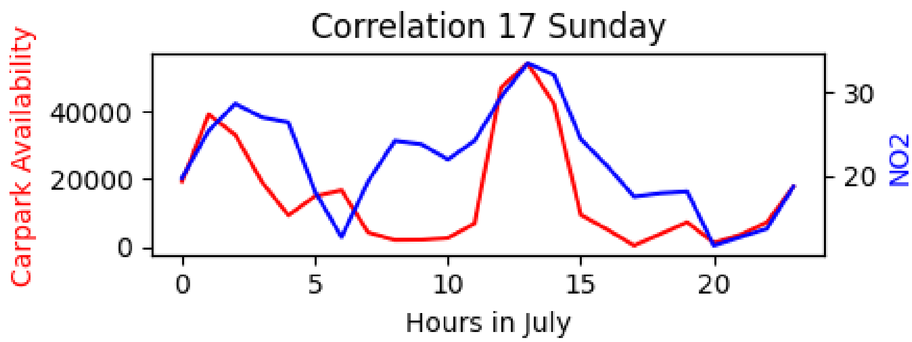
```
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['avg_no2'][408:432], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation 17 Sunday")
plt.tight_layout()
plt.show()
print("\n")
print("\n")
```

OVERALL PLOTS









3.3.2 3b. Correlation Observation of CarPark Abs Availability Difference and NO2 (Regional)

[ ]:

```
# Plotting to obeserve Correlation - Using Abs Difference in Carpark␣
↪Availability and NO2
#Dates are: July 9 (Sat) - 216:240, July 11(Mon) - 264:288, July 15(Fri) - 360:
↪384, July 17(Sun) - 408:432
##################################North##################################
print('NORTH PLOTS')
fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_north['avail_difference'][216:240], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
```

```
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['north'][ 216:240], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation North 9 Saturday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_north['avail_difference'][264:288], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['north'][264:288], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation North 11 Monday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_north['avail_difference'][360:384], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['north'][360:384], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation North 15 Friday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_north['avail_difference'][408:432], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
```

```
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['north'][408:432], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation North 17 Sunday")
plt.tight_layout()
plt.show()
print("\n")
print("\n")

################################South################################
print('SOUTH PLOTS')
fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_south['avail_difference'][216:240], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['south'][ 216:240], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation South 9 Saturday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_south['avail_difference'][264:288], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['south'][264:288], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation South 11 Monday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
```

```
# Plot carpark on left y-axis
ax1.plot(range(24), july_south['avail_difference'][360:384], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['south'][360:384], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation South 15 Friday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_south['avail_difference'][408:432], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['south'][408:432], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation South 17 Sunday")
plt.tight_layout()
plt.show()
print("\n")
print("\n")


################################East###############################
print('EAST PLOTS')
fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_east['avail_difference'][216:240], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['east'][ 216:240], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
```

```
ax1.set_xlabel('Hours in July')
plt.title("Correlation East 9 Saturday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_east['avail_difference'][264:288], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['east'][264:288], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation East 11 Monday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_east['avail_difference'][360:384], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['east'][360:384], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation East 15 Friday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_east['avail_difference'][408:432], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['east'][408:432], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
```

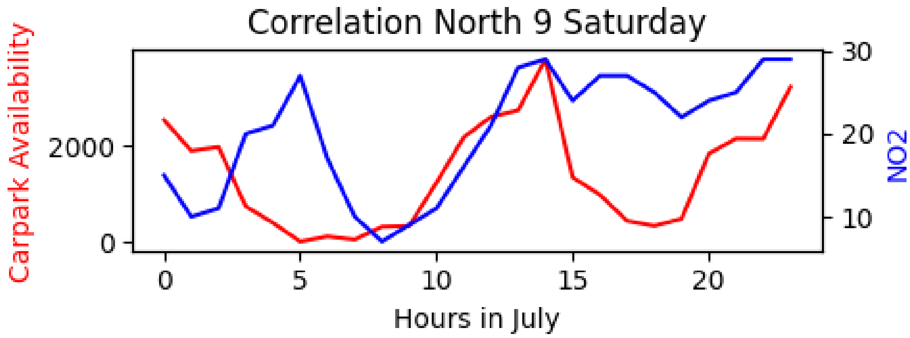
```
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation East 17 Sunday")
plt.tight_layout()
plt.show()
print("\n")
print("\n")

################################West################################
print('WEST PLOTS')
fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_west['avail_difference'][216:240], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['west'][ 216:240], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation West 9 Saturday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_west['avail_difference'][264:288], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['west'][264:288], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation West 11 Monday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_west['avail_difference'][360:384], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
```

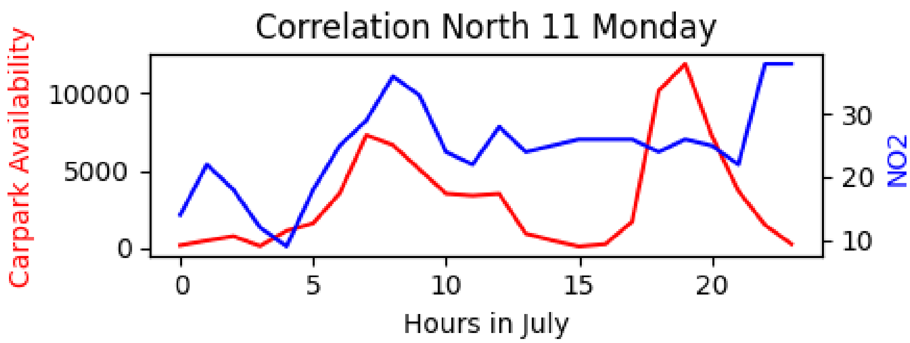
```
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['west'][360:384], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation West 15 Friday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_west['avail_difference'][408:432], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['west'][408:432], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation West 17 Sunday")
plt.tight_layout()
plt.show()
print("\n")
print("\n")

################################Central###############################
print('CENTRAL PLOTS')
fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_central['avail_difference'][216:240], 'r-',␣
↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['central'][ 216:240], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation Central 9 Saturday")
plt.tight_layout()
plt.show()
```

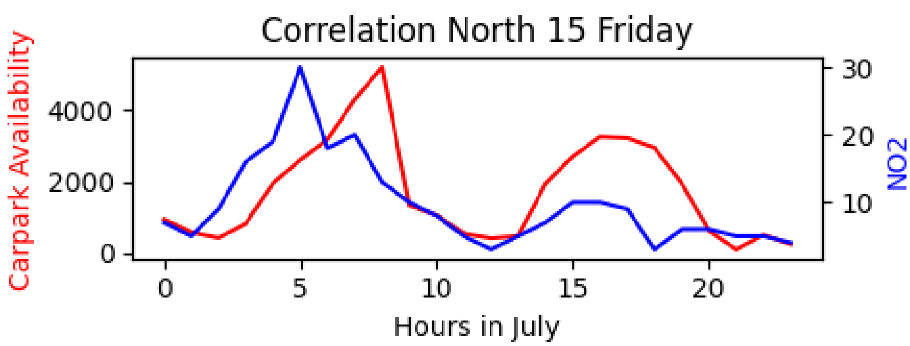
```
fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_central['avail_difference'][264:288], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['central'][264:288], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation Central 11 Monday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_central['avail_difference'][360:384], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['central'][360:384], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation Central 15 Friday")
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(5,2))
# Plot carpark on left y-axis
ax1.plot(range(24), july_central['avail_difference'][408:432], 'r-',␣
 ↪label='Carpark')
ax1.set_ylabel('Carpark Availability', color='r')
# Create a second axes for the second y-axis
ax2 = ax1.twinx()
# Plot NO2 on right y-axis
ax2.plot(range(24), JulyNO2['central'][408:432], 'b-', label='NO2')
ax2.set_ylabel('NO2', color='b')
# Adjust layout and add legend
ax1.set_xlabel('Hours in July')
plt.title("Correlation Central 17 Sunday")
plt.tight_layout()
plt.show()
```

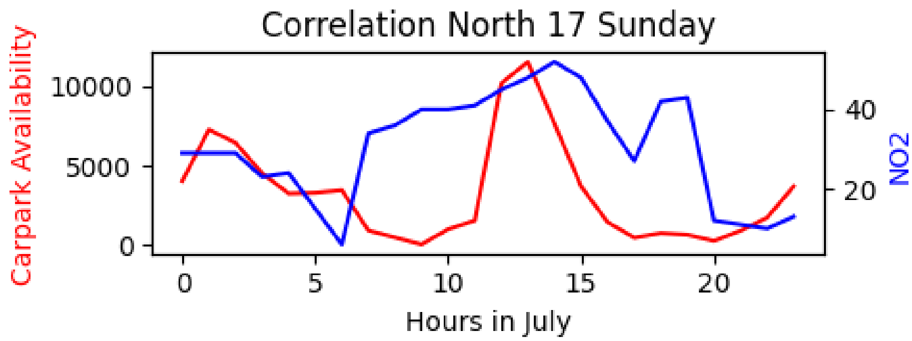
```python
print("\n")
print("\n")
```

NORTH PLOTS

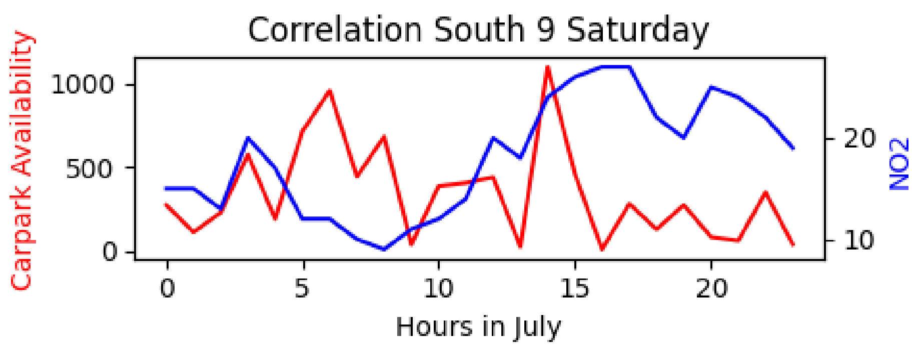


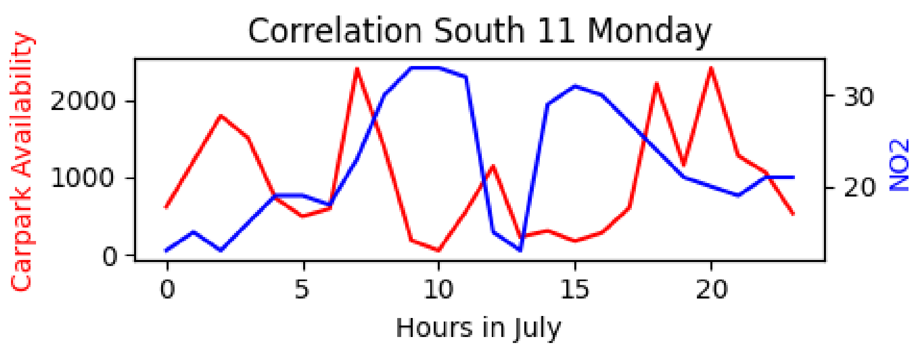


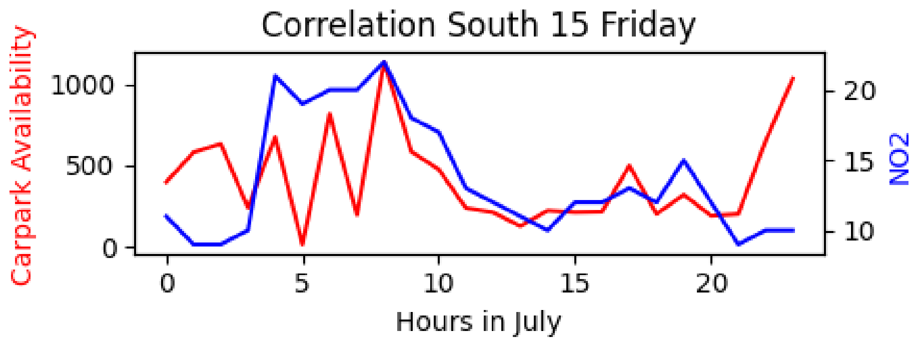


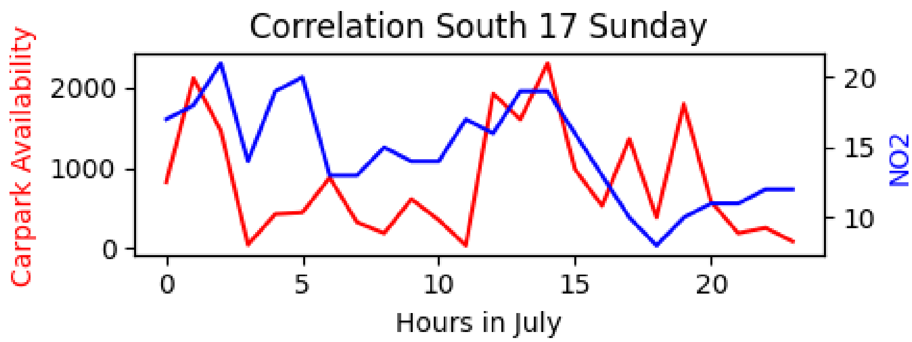

Correlation North 17 Sunday
Carpark Availability
10000
5000
0
40
20
NO2
0
5
10
15
20
Hours in July

SOUTH PLOTS

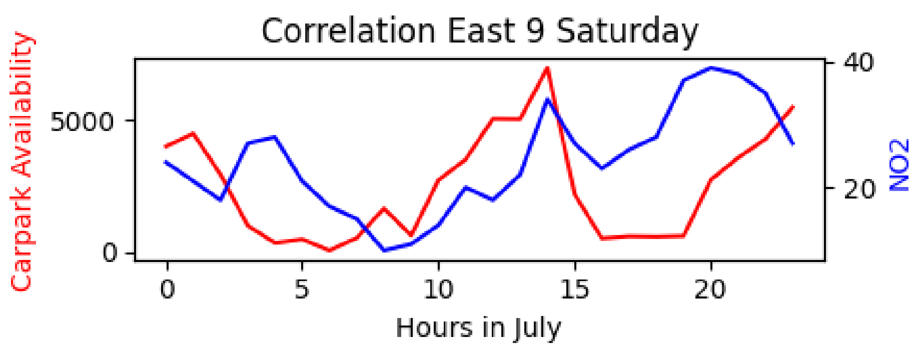

Correlation South 9 Saturday
Carpark Availability
1000
500
0
20
10
NO2
0
5
10
15
20
Hours in July

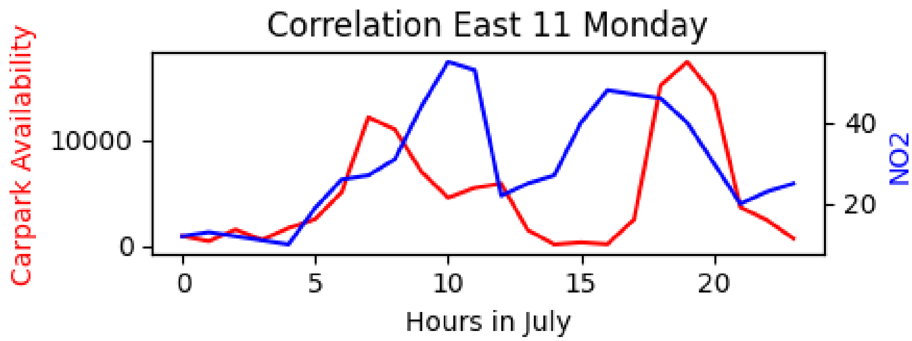

Correlation South 11 Monday
Carpark Availability
2000
1000
0
30
20
NO2
0
5
10
15
20
Hours in July


Correlation South 15 Friday
Carpark Availability
1000
500
0
0
5
10
15
20
Hours in July
20
15
10
NO2

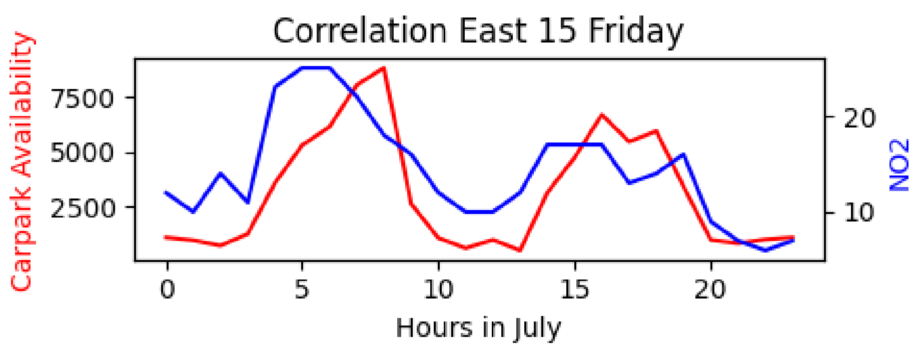

Correlation South 17 Sunday
Carpark Availability
2000
1000
0
0
5
10
15
20
Hours in July
20
15
10
NO2

EAST PLOTS

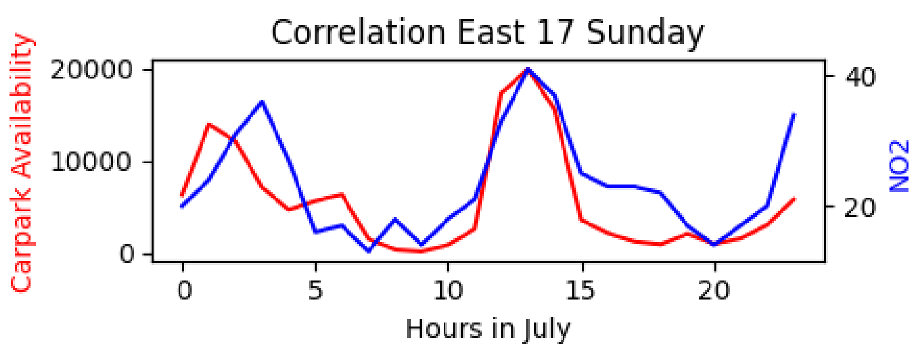

Correlation East 9 Saturday
Carpark Availability
5000
0
0
5
10
15
20
Hours in July
40
20
NO2



Correlation East 11 Monday
Carpark Availability
NO2
Hours in July

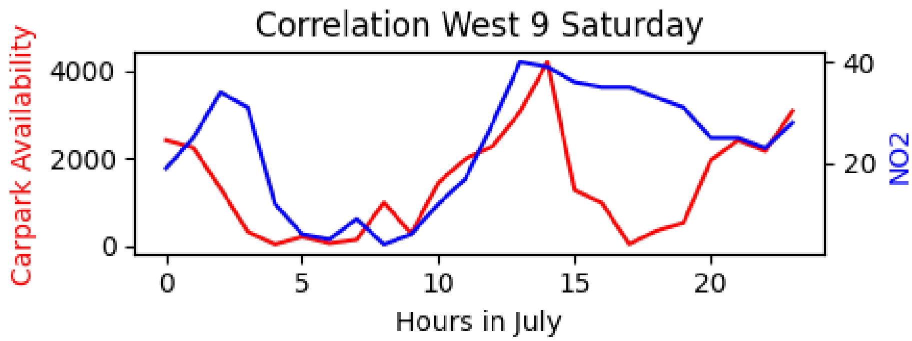


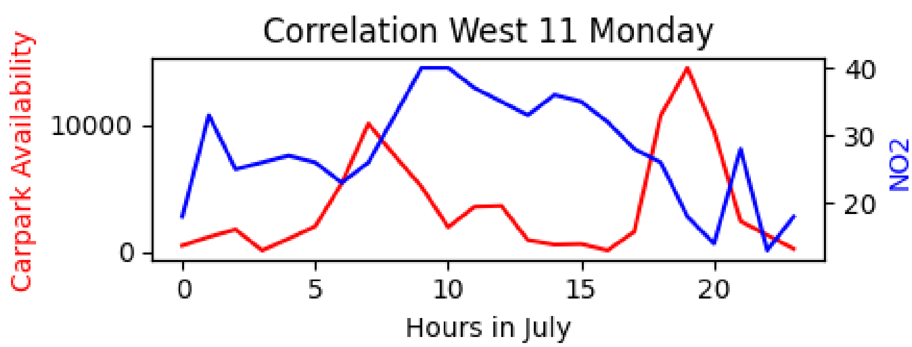
Correlation East 15 Friday
Carpark Availability
NO2
Hours in July

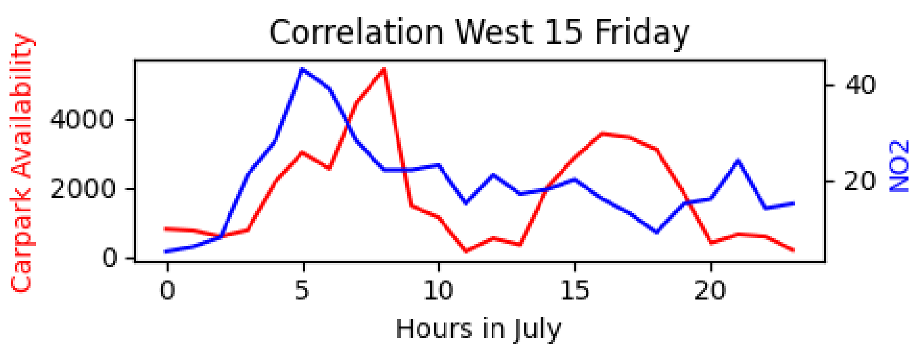


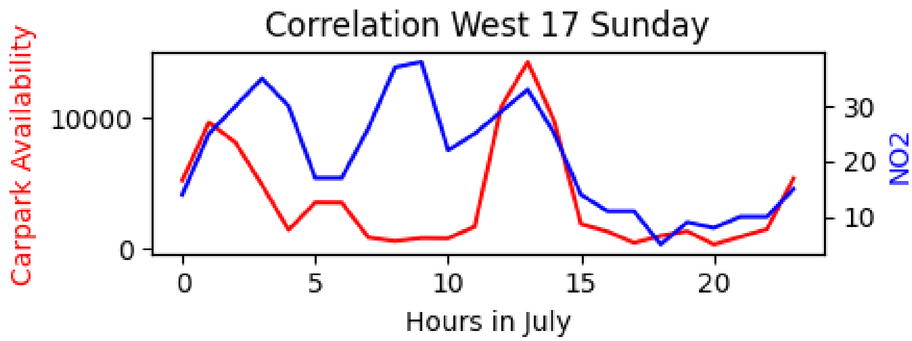
Correlation East 17 Sunday
Carpark Availability
NO2
Hours in July

WEST PLOTS

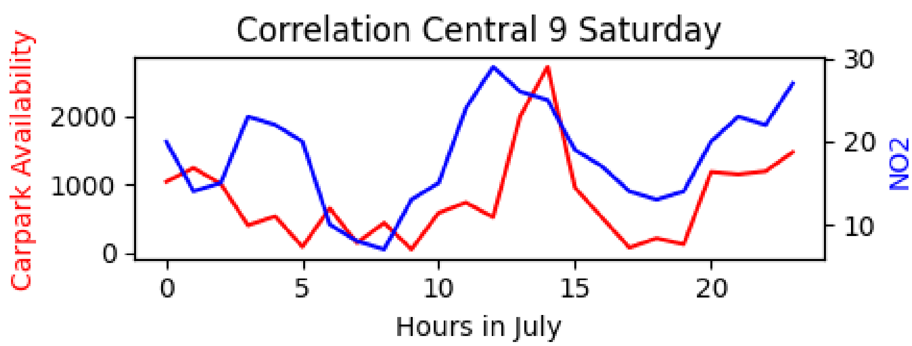

Correlation West 9 Saturday
Carpark Availability
NO2
Hours in July
0
2000
4000
20
40
0
5
10
15
20

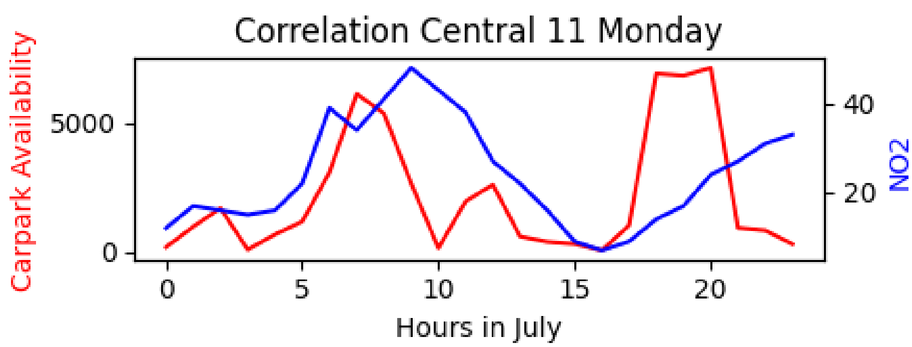

Correlation West 11 Monday
Carpark Availability
NO2
Hours in July
0
10000
20
30
40
0
5
10
15
20

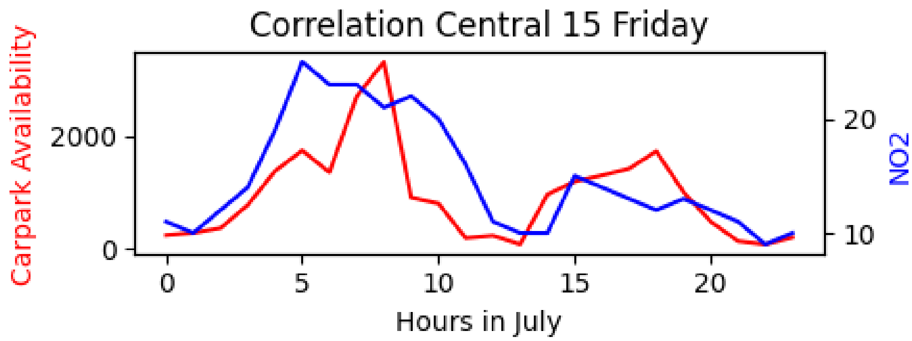

Correlation West 15 Friday
Carpark Availability
NO2
Hours in July
0
2000
4000
20
40
0
5
10
15
20

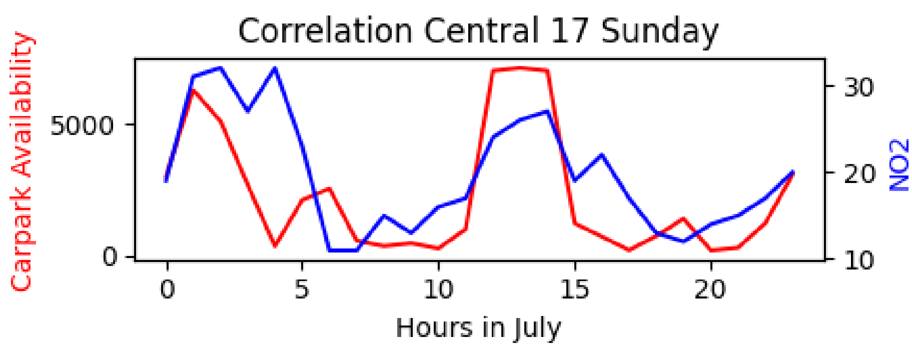


CENTRAL PLOTS









3.4 4a. Create a new df and export to csv for prediction and selection of model

Since Carpark Absolute Availability Difference is correlated to NO2 level, we can make use of the absolute availability difference to predict the NO2 level.

[ ]:
```
# Clean df with only the columns that will be needed for model training and␣
↪prediction
new_order = ['Day', 'Hour', 'west', 'east', 'central', 'south', 'north',␣
↪'avg_no2']
July = JulyNO2[new_order].copy()
July.columns = ['Day', 'Hour', 'NO2_west', 'NO2_east', 'NO2_central',␣
↪'NO2_south', 'NO2_north', 'NO2_mean']

new_order = ['Day', 'Hour', 'west', 'east', 'central', 'south', 'north',␣
↪'avg_no2']
Aug = AugNO2[new_order].copy()
```

```
Aug.columns = ['Day', 'Hour', 'NO2_west', 'NO2_east', 'NO2_central',␣
 ↪'NO2_south', 'NO2_north', 'NO2_mean']

regions = ['north','south','east','west','central']
renamed_cols = [i+'_avail_diff' for i in regions]

for i in range(5):
  # Reset the index before calling get_avail_difference
  July_region_df = July_grouping.loc[regions[i]].reset_index()
  Aug_region_df = Aug_grouping.loc[regions[i]].reset_index()

  July[renamed_cols[i]] =␣
 ↪get_avail_difference(July_region_df)['avail_difference']
  Aug[renamed_cols[i]] = get_avail_difference(Aug_region_df)['avail_difference']

# Calculate mean_avail_diff
July['mean_avail_diff'] = July.iloc[:, 8:13].mean(axis=1)
Aug['mean_avail_diff'] = Aug.iloc[:, 8:13].mean(axis=1)

July.to_csv('July_clean.csv')
Aug.to_csv('Aug_clean.csv')
```

3.5 4b. Model Selection

3.5.1 Importing necessary files

[ ]:
```
July = pd.read_csv("July_clean.csv",index_col=0)
Aug = pd.read_csv("Aug_clean.csv",index_col=0)

print(July)
print(Aug)
```

```
     Day  Hour  NO2_west  NO2_east  NO2_central  NO2_south  NO2_north  \
0      1     1        27        27           31         19         18
1      1     2        29        30           30         20         18
2      1     3        26        33           21         21         26
3      1     4        15        26           24         25         21
4      1     5         6        17           26         26         24
..   …     …        …         …          …          …          …
708   31    19        27        20           19         19         12
709   31    20        25        28           32         32         15
710   31    21        23        35           30         30         36
711   31    22        28        39           26         26         13
712   31    23        26        44           32         32         39

     NO2_mean  north_avail_diff  south_avail_diff  east_avail_diff  \
0        24.4       2712.675562        662.891854      4621.446629
1        25.4        653.000000        465.000000      2342.000000
```

```
2        25.4      433.000000       372.000000       515.000000
3        22.2       39.000000       134.000000       343.000000
4        19.8       77.000000       110.000000       107.000000
..        …            …                …                …
708      19.4     1763.000000       421.000000      2678.000000
709      26.4     1960.000000       283.000000      3524.000000
710      30.8     2689.000000       209.000000      4904.000000
711      26.4     3072.000000       308.000000      5668.000000
712      34.6     3147.000000         8.000000      4451.000000

     west_avail_diff  central_avail_diff  mean_avail_diff
0        2981.456461         1771.115169      2549.917135
1         969.000000          871.000000      1060.000000
2         622.000000          486.000000       485.600000
3         774.000000          632.000000       384.400000
4         163.000000           18.000000        95.000000
..               …                  …                …
708      1860.000000         1304.000000      1605.200000
709      2102.000000          787.000000      1731.200000
710      2993.000000          802.000000      2319.400000
711      3155.000000         1220.000000      2684.600000
712      3004.000000         1503.000000      2422.600000

[713 rows x 14 columns]
     Day  Hour  NO2_west  NO2_east  NO2_central  NO2_south  NO2_north  \
0      1     1        29        34           28         28         30
1      1     2        25        28           30         30         26
2      1     3        27        36           29         29         14
3      1     4        27        38           27         27         12
4      1     5        25        32           24         24         11
..   …    …        …       …           …          …         …
708   31    19        10        38           26         21         47
709   31    20        23        39           39         27         55
710   31    21        37        49           31         28         32
711   31    22        45        39           28         23         28
712   31    23        44        32           31         20         34

     NO2_mean  north_avail_diff  south_avail_diff  east_avail_diff  \
0        29.8       2874.952247        703.292135      4890.882022
1        27.8        943.000000        316.000000      1368.000000
2        27.0        103.000000        210.000000       199.000000
3        26.2        670.000000        490.000000       539.000000
4        23.2        324.000000        262.000000       279.000000
..        …             …                 …                …
708      28.4       6590.000000       1887.000000     12409.000000
709      36.6       6777.000000       1586.000000     10871.000000
710      35.4       4056.000000         98.000000      8399.000000
711      32.6       4696.000000        446.000000      6893.000000
```

```
712      32.2       3909.000000        511.000000      6858.000000

     west_avail_diff  central_avail_diff  mean_avail_diff
0        3121.411517         1895.390449      2697.185674
1         788.000000          564.000000       795.800000
2         614.000000          429.000000       311.000000
3         167.000000          118.000000       396.800000
4         244.000000          165.000000       254.800000
..               …                   …               …
708      8335.000000         4811.000000      6806.400000
709      6750.000000         5140.000000      6224.800000
710      5217.000000         2835.000000      4121.000000
711      3848.000000         1727.000000      3522.000000
712      3867.000000         2239.000000      3476.800000

[713 rows x 14 columns]
```

3.5.2 Assigning to X and Y for July and Aug

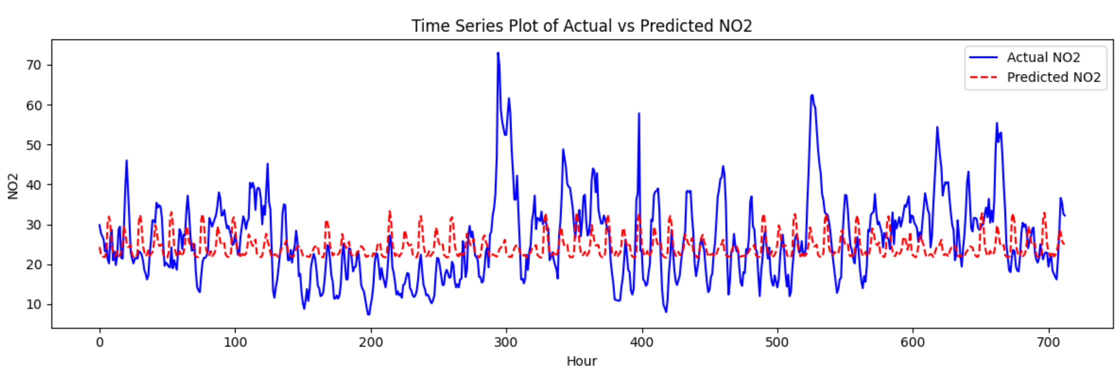
[ ]:
```
timeline = np.arange(len(July)).reshape(-1, 1) # July and Aug have same length

# July_X
July_abs_avail_diff = July['mean_avail_diff'].tolist()
# July Y
July_mean_NO2 = July['NO2_mean'].tolist()


# Aug_X
Aug_abs_avail_diff = Aug['mean_avail_diff'].tolist()
# Aug Y
Aug_mean_NO2 = Aug['NO2_mean'].tolist()
```

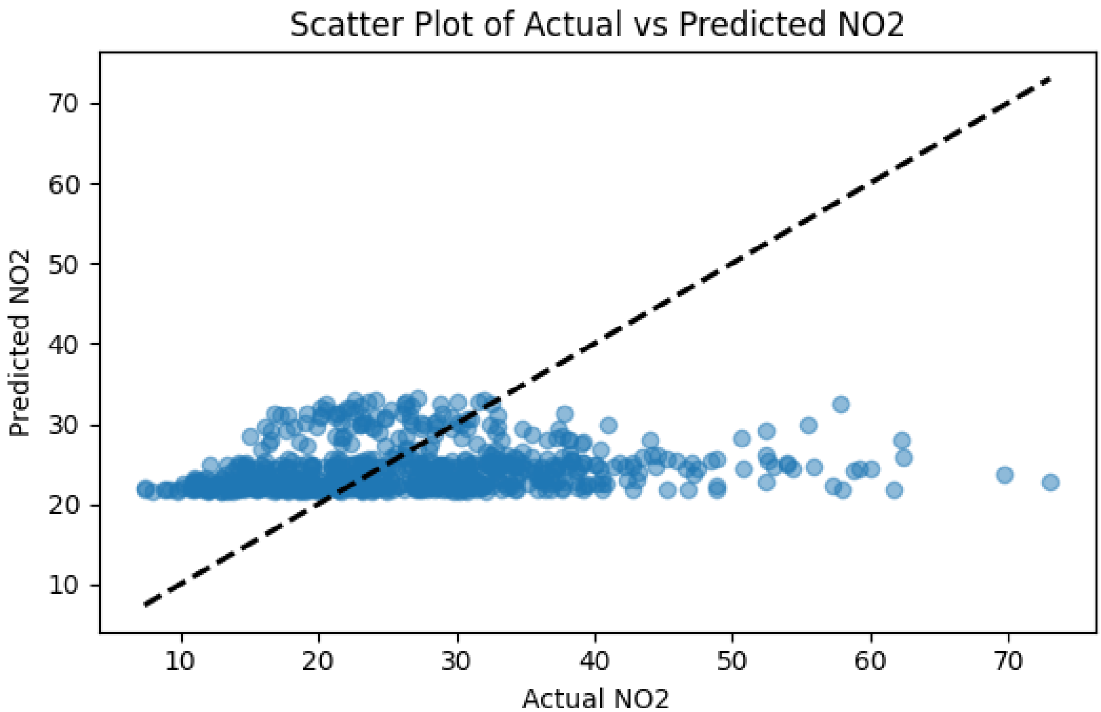
3.5.3 Linear Regression Model

[ ]:
```
################################################################################
###################        LINEAR REGRESSION MODEL        ######################
################################################################################

# Prepare the features and target variable
# Assuming July_abs_avail_diff and July_mean_NO2 are columns in July dataset,␣
 ↪similarly for August
X_July = July['mean_avail_diff'].values.reshape(-1, 1)
y_July = July['NO2_mean']
X_Aug = Aug['mean_avail_diff'].values.reshape(-1, 1)
y_Aug = Aug['NO2_mean']

# Create a Linear Regression model and train on July data
```

```python
lr = LinearRegression()
lr.fit(X_July, y_July)

# Prediction on August data
LR_predictions = lr.predict(X_Aug)

# Calculate RMSE for predictions
LRmse = mean_squared_error(y_Aug, LR_predictions)
LRrmse = np.sqrt(LRmse)
print("\nRMSE value for linear regression is: {}".format(LRrmse))

# Plotting
# Time series plot of the actual and predicted hourly values for August
plt.figure(figsize=(12, 4))
plt.plot(Aug.index, y_Aug, label="Actual NO2", color="blue")
plt.plot(Aug.index, LR_predictions, label="Predicted NO2", color="red",␣
 ↪linestyle="--")
plt.xlabel("Hour")
plt.ylabel("NO2")
plt.title("Time Series Plot of Actual vs Predicted NO2")
plt.legend()
plt.tight_layout()
plt.show()

# Scatter plot of actual vs predicted hourly values for August
plt.figure(figsize=(6, 4))
plt.scatter(y_Aug, LR_predictions, alpha=0.5)
plt.plot([min(y_Aug), max(y_Aug)], [min(y_Aug), max(y_Aug)], 'k--', lw=2)
plt.xlabel("Actual NO2")
plt.ylabel("Predicted NO2")
plt.title("Scatter Plot of Actual vs Predicted NO2")
plt.tight_layout()
plt.show()
```

```
RMSE value for linear regression is: 10.301718692548873
```





3.5.4 Linear Regression Model (with windowing method)

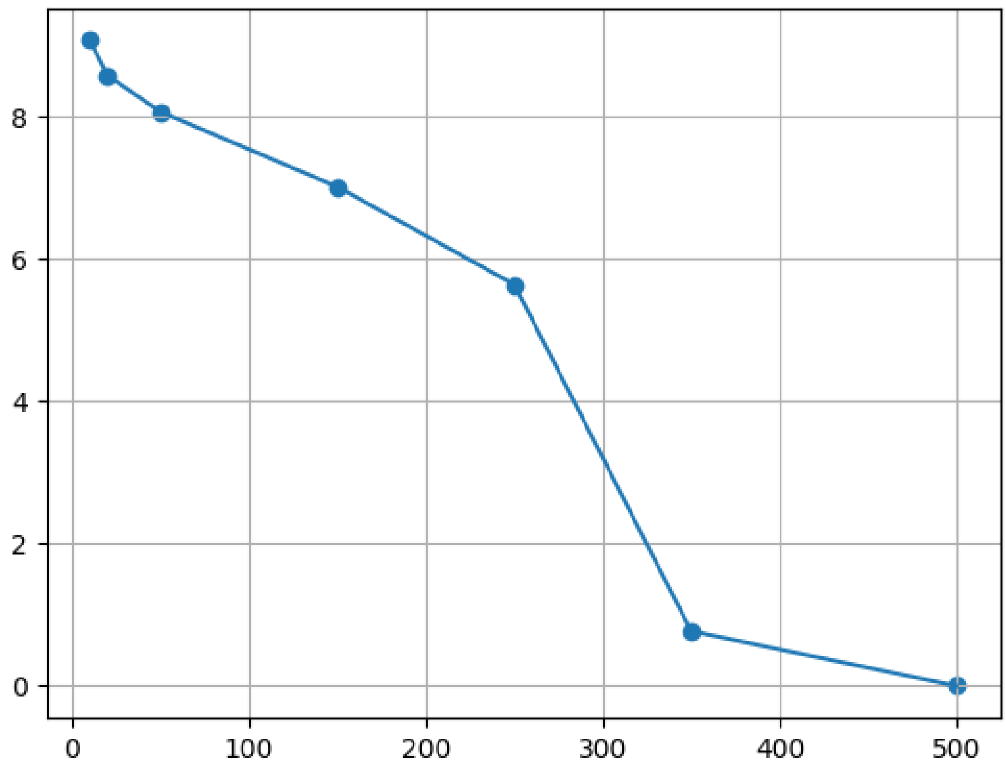
```
[ ]: ################################################################################
     ################        LINEAR REGRESSION SLIDING WINDOW        ################
     ################################################################################

     ######################### SELECTING WINDOW SIZE ###############################

     window_sizes = [10,20,50,150,250,350,500]
     rmse_values = []

     for window_size in window_sizes:
         X = []
         y = []

         for i in range(len(July_abs_avail_diff)-window_size):
             X.append(July_abs_avail_diff[i:i+window_size])
             y.append(July_mean_NO2[i+window_size])

         # create a Linear Regression model
         lr = LinearRegression()
```

```
    lr.fit(X, y)

    y_pred = lr.predict(X)

    mse = mean_squared_error(July_mean_NO2[window_size:],y_pred)
    rmse = np.sqrt(mse)
    rmse_values.append(rmse)

    # print('RMSE with window size, ', window_size)
    # print("Root mean squared error:", rmse)

    # # Visualize the original and predicted data
    # plt.figure(figsize=(10, 3))
    # plt.plot(timeline, July_mean_NO2, label='Actual NO2', color='blue')
    # plt.plot(timeline[window_size:], y_pred, label='Predicted NO2',␣
 ↪color='red')
    # plt.xlabel('Hour')
    # plt.ylabel('NO2')
    # plt.title(f"Time series plot of actual data and predicted hourly values␣
 ↪for July 2022, window_size = {window_size}")
    # plt.legend(loc='upper left')
    # plt.grid(True)
    # plt.show()

for i in range(len(window_sizes)):
    print("Window size: {:^5}, RMSE value: {:^10}".
 ↪format(window_sizes[i],rmse_values[i]) )
plt.scatter(window_sizes,rmse_values)
plt.plot(window_sizes,rmse_values)
plt.grid(True)
plt.show()


#################### TRAINING AFTER SELECTING WINDOW SIZE ####################

# Selected Window Size of 250 to achieve a balance between overfitting and␣
 ↪training performance as much as possible
# Even though RMSE still improves, we did not want to overfit or have the␣
 ↪possibility of loss of relevance

window_size = 250

X = []
y = []

for i in range(len(July_abs_avail_diff)-window_size):
    X.append(July_abs_avail_diff[i:i+window_size])
```

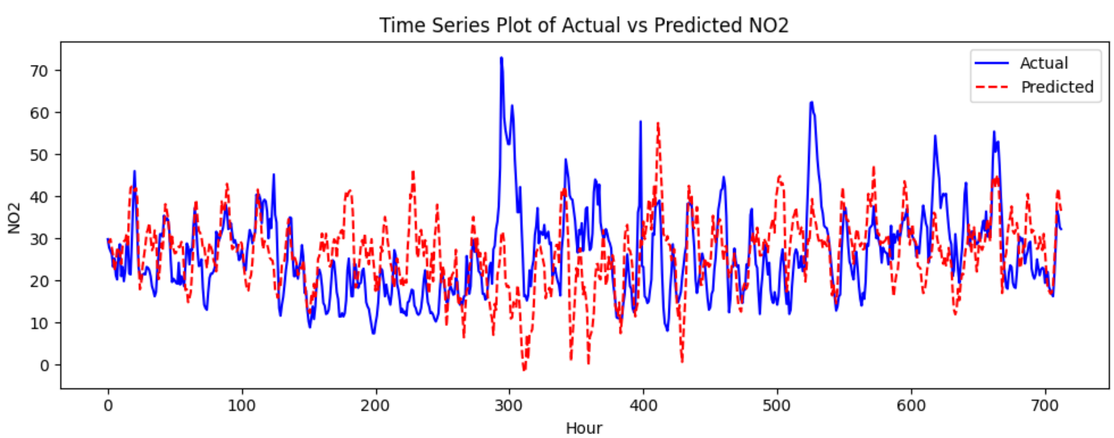
```
    y.append(July_mean_NO2[i+window_size])

# Create a Linear Regression model and train
lr = LinearRegression()
lr.fit(X, y)

# Prediction
dataset = July_abs_avail_diff[-window_size:] + Aug_abs_avail_diff
X2 = []
for i in range(len(dataset)-window_size):
    X2.append(dataset[i:i+window_size])

LRSW_predictions = lr.predict(X2)

LRmse_window = mean_squared_error(Aug_mean_NO2,LRSW_predictions)
LRrmse_window = np.sqrt(LRmse_window)
print("\nWith a window size of {}, the RMSE value for linear regression using␣
↪windowing method is: {}".format(window_size,LRrmse_window))

# Plotting
# Time series plot of the actual and predicted hourly values
plt.figure(figsize=(10, 4))
plt.plot(timeline, Aug_mean_NO2, label="Actual", color="blue")
plt.plot(timeline, LRSW_predictions, label="Predicted", color="red",␣
↪linestyle="--")
plt.xlabel("Hour")
plt.ylabel("NO2")
plt.title("Time Series Plot of Actual vs Predicted NO2")
plt.legend()
plt.tight_layout()
plt.show()

# Scatter plot of actual vs predicted hourly values
plt.figure(figsize=(6,4))
plt.scatter(Aug_mean_NO2, LRSW_predictions, alpha=0.5)
plt.plot([min(Aug_mean_NO2), max(Aug_mean_NO2)], [min(Aug_mean_NO2),␣
↪max(Aug_mean_NO2)], 'k--', lw=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Scatter Plot of Actual vs Predicted NO2")
plt.tight_layout()
plt.show()
```

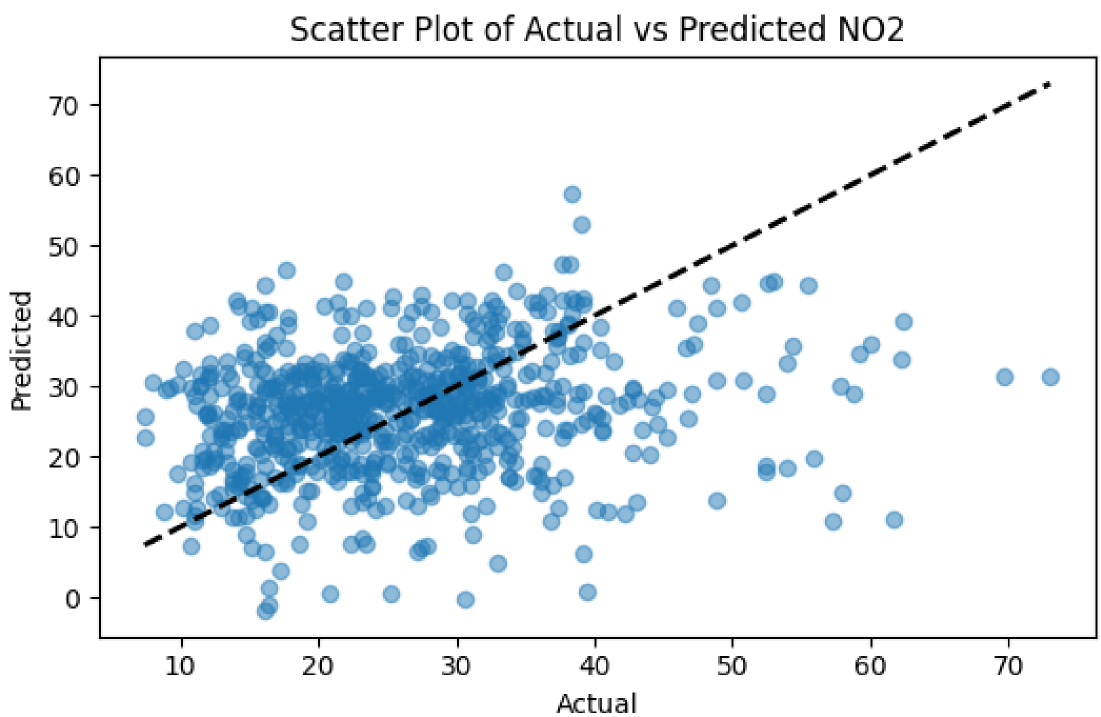
```
Window size:  10  , RMSE value: 9.070763157659982
Window size:  20  , RMSE value: 8.577166280948232
Window size:  50  , RMSE value: 8.066188675148913
Window size:  150 , RMSE value: 7.012680079162362
```

```
Window size:  250 , RMSE value: 5.639621102764824
Window size:  350 , RMSE value: 0.76868121728125
Window size:  500 , RMSE value: 4.839810086095278e-14
```




```
With a window size of 250, the RMSE value for linear regression using windowing
method is: 11.92104753968608
```





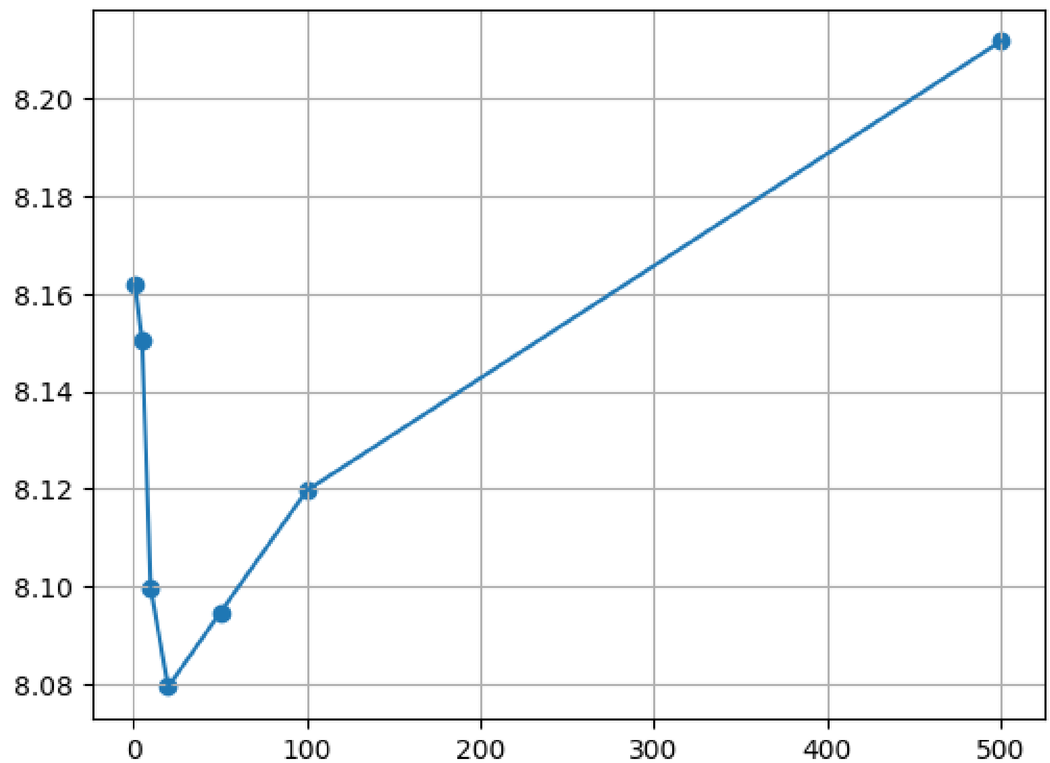
3.5.5 SVR

[ ]:
```
################################################################################
###################         Support Vector Regression         ####################
################################################################################

########################### Preparing variables ################################
X_July = July[['Hour','mean_avail_diff']]
y_July = July['NO2_mean']

X_Aug = Aug[['Hour','mean_avail_diff']]
y_Aug = Aug['NO2_mean']

split_point = int(0.8 * len(X_July))

# Train:0.8 Test Split:0.2 on July Data
X_train, X_test = X_July[:split_point], X_July[split_point:]
y_train, y_test = y_July[:split_point], y_July[split_point:]

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

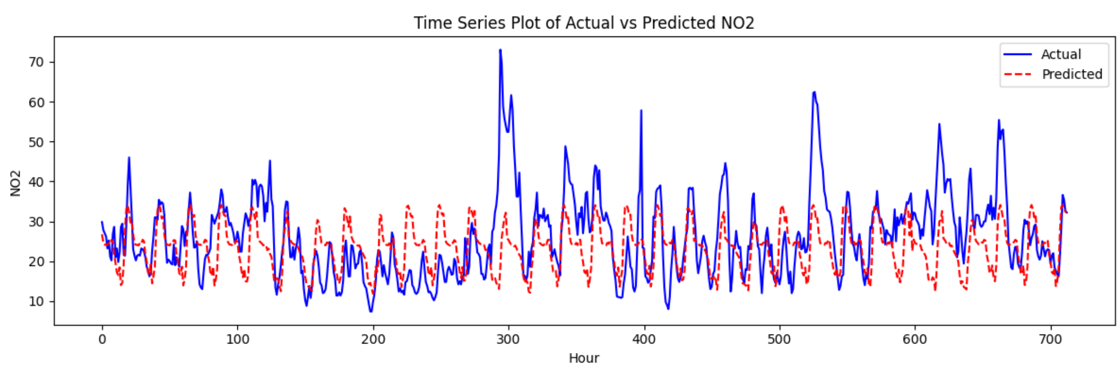
```python
############################       Hypertuning      ############################

rmse_list = []
C_values = [1,5,10,20,50,100,500]

# Hypertuning based on July Data Set
for i in C_values:
  model = SVR(C=i, kernel='rbf',gamma='auto')
  model.fit(X_train, y_train)
  predictions = model.predict(X_test)

  # Calculate and print the root mean squared error (RMSE)
  rmse = np.sqrt(mean_squared_error(y_test, predictions))
  rmse_list.append(rmse)

  # print('RMSE with C value, ', i)
  # print("Root mean squared error:", rmse)

  # plt.figure(figsize=(12, 3))
  # plt.plot(timeline, y_July, label="Actual", color="blue")
  # plt.plot(range(split_point,len(y_July)), predictions, label="Predicted",␣
 ↪color="red", linestyle="--")
  # plt.xlabel("Hour")
  # plt.ylabel("NO2")
  # plt.title("Time Series Plot of Actual vs Predicted NO2")
  # plt.legend()
  # plt.tight_layout()
  # plt.show()

for i in range(len(C_values)):
    print("C value: {:<4}, RMSE: {:<10}".format(C_values[i], rmse_list[i]))
plt.scatter(C_values,rmse_list)
plt.plot(C_values,rmse_list)
plt.grid(True)
plt.show()

############################       Prediction       ############################
# Hypertuning Done -> Select parameter C = 10
# When C = 10 it gives the minimum RMSE

X_July = scaler.fit_transform(X_July)
X_Aug = scaler.transform(X_Aug)

model = SVR(C=10, kernel='rbf',gamma='auto')
model.fit(X_July, y_July)
SVRpredictions = model.predict(X_Aug)
```

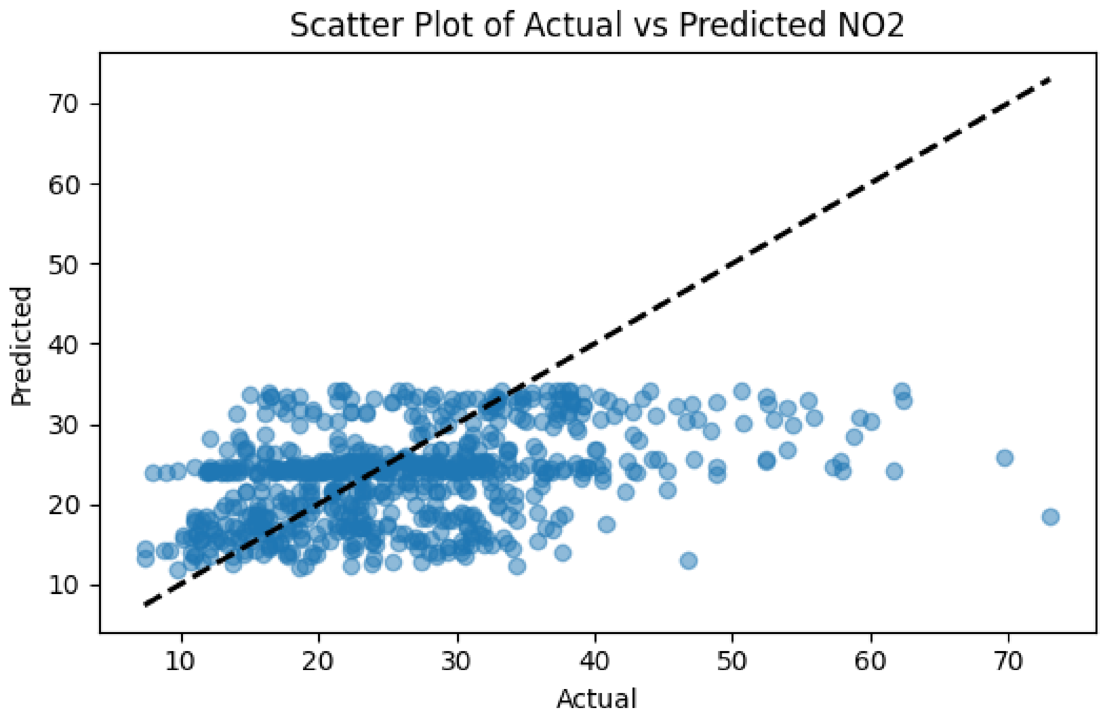
```python
SVRmse = mean_squared_error(Aug_mean_NO2,SVRpredictions)
SVRrmse = np.sqrt(SVRmse)

print("\nWith a C value of {}, the RMSE value for support vector regression is:
 ↪{}".format(10,SVRrmse))

# Plotting
# Time series plot of the actual and predicted hourly values
plt.figure(figsize=(12, 4))
plt.plot(timeline, y_Aug, label="Actual", color="blue")
plt.plot(timeline, SVRpredictions, label="Predicted", color="red",
 ↪linestyle="--")
plt.xlabel("Hour")
plt.ylabel("NO2")
plt.title("Time Series Plot of Actual vs Predicted NO2")
plt.legend()
plt.tight_layout()
plt.show()

# Scatter plot of actual vs predicted hourly values
plt.figure(figsize=(6, 4))
plt.scatter(y_Aug, SVRpredictions, alpha=0.5)
plt.plot([min(y_Aug), max(y_Aug)], [min(y_Aug), max(y_Aug)], 'k--', lw=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Scatter Plot of Actual vs Predicted NO2")
plt.tight_layout()
plt.show()
```

```
C value: 1   , RMSE: 8.161803108593832
C value: 5   , RMSE: 8.150286144508735
C value: 10  , RMSE: 8.099565604018943
C value: 20  , RMSE: 8.07951161901282
C value: 50  , RMSE: 8.094704372552567
C value: 100 , RMSE: 8.119735160359358
C value: 500 , RMSE: 8.21160758370585
```



With a C value of 10, the RMSE value for support vector regression is:
9.945864263624332





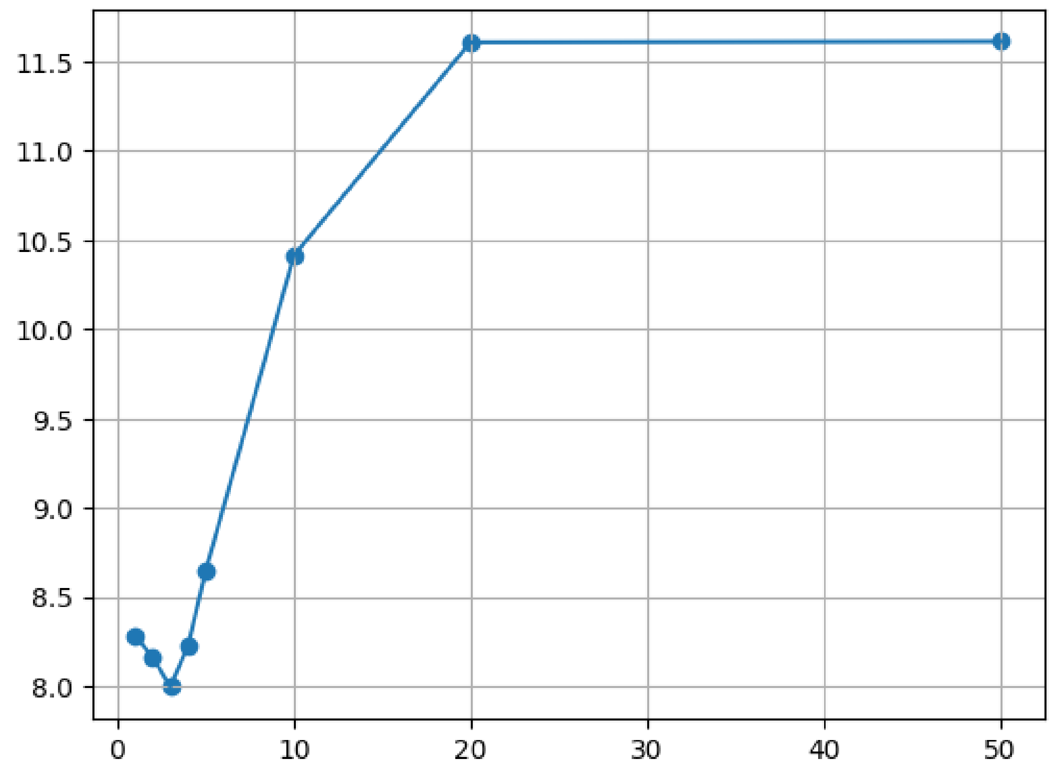
3.5.6 DTR

[ ]:
```
################################################################################
###################        Decision Tree Regression        #####################
################################################################################

########################## Preparing variables ################################
X_July = July[['Hour','mean_avail_diff']]
y_July = July['NO2_mean']

X_Aug = Aug[['Hour','mean_avail_diff']]
y_Aug = Aug['NO2_mean']

split_point = int(0.8 * len(X_July))

# Train:0.8 Test Split:0.2 on July Data
X_train, X_test = X_July[:split_point], X_July[split_point:]
y_train, y_test = y_July[:split_point], y_July[split_point:]

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

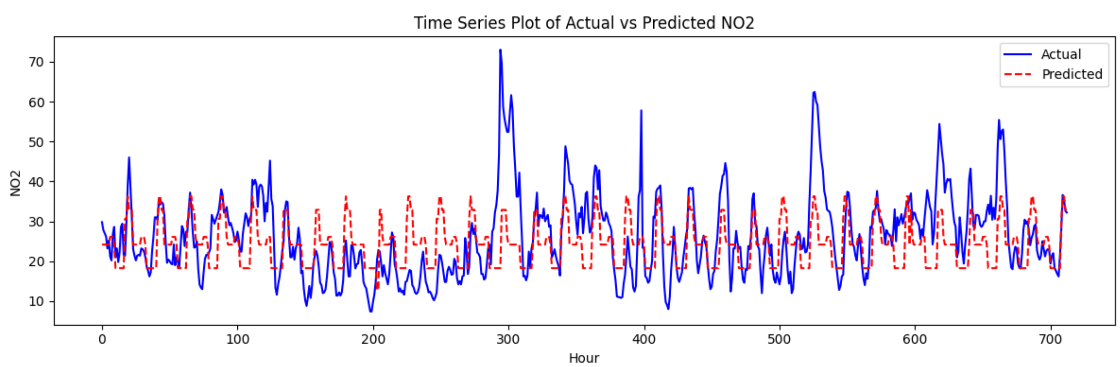
```python
###########################      Hypertuning      ###########################

rmse_list = []
depth = [1, 2, 3, 4, 5, 10, 20, 50]

# Hypertuning based on July Data Set
for i in depth:
    DTmodel = DecisionTreeRegressor(max_depth = i)
    DTmodel.fit(X_train, y_train)
    DTpredictions = DTmodel.predict(X_test)

    # Calculate and print the root mean squared error (RMSE)
    rmse = np.sqrt(mean_squared_error(y_test, DTpredictions))
    rmse_list.append(rmse)

    # print('RMSE with DT max depth value, ', i)
    # print("Root mean squared error:", rmse)

    # plt.figure(figsize=(15, 3))
    # plt.plot(timeline, y_July, label="Actual", color="blue")
    # plt.plot(range(split_point,len(y_July)), DTpredictions,␣
 ↪label="Predicted", color="red", linestyle="--")
    # plt.xlabel("Hour")
    # plt.ylabel("NO2")
    # plt.title("Time Series Plot of Actual vs Predicted NO2")
    # plt.legend()
    # plt.tight_layout()
    # plt.show()

for i in range(len(rmse_list)):
  print("Max depth value: {:<4}, RMSE: {:<10}".format(depth[i], rmse_list[i]))

plt.scatter(depth,rmse_list)
plt.plot(depth,rmse_list)
plt.grid(True)
plt.show()

###########################      Prediction      ###########################
# Hypertuning Done -> Select parameter DT max depth = 3
# When max depth = 3, it gives the minimum RMSE
X_July = scaler.fit_transform(X_July)
X_Aug = scaler.transform(X_Aug)

d = 3
DTmodel = DecisionTreeRegressor(max_depth = d)
DTmodel.fit(X_July, y_July)
DTpredictions = DTmodel.predict(X_Aug)
```

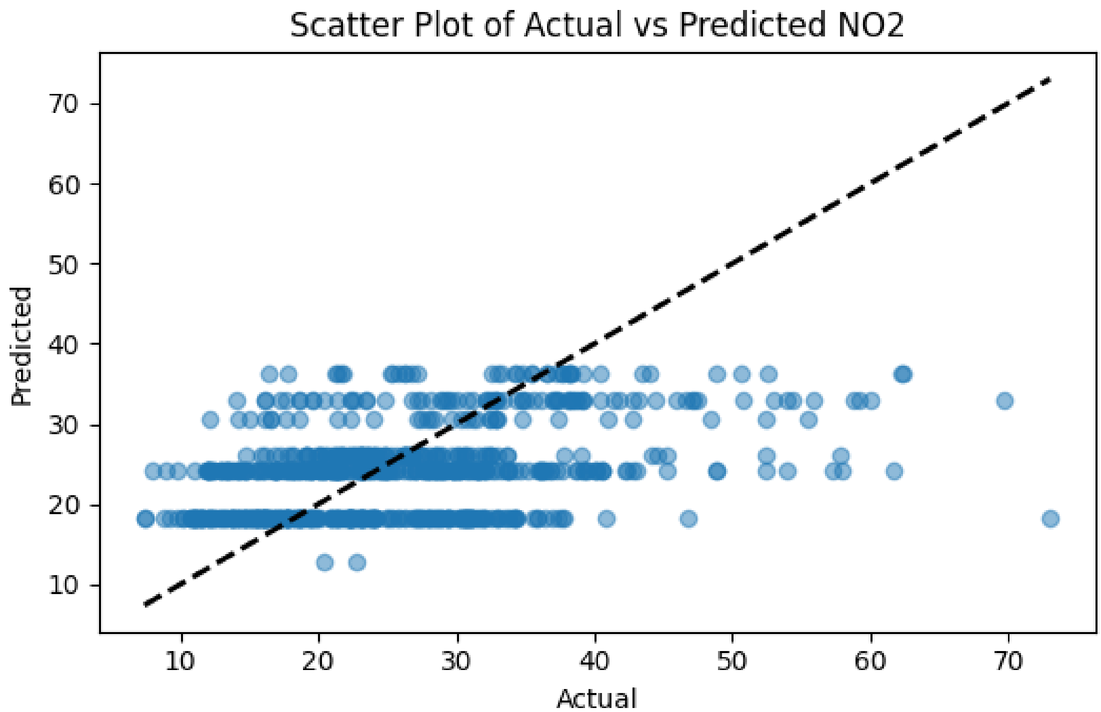
```
DTRmse = mean_squared_error(Aug_mean_NO2,DTpredictions)
DTRrmse = np.sqrt(DTRmse)

print("\nWith a max depth of {}, the RMSE value for decision tree regression is:
 ↪ {}".format(d,DTRrmse))

# Plotting
# Time series plot of the actual and predicted hourly values
plt.figure(figsize=(12, 4))
plt.plot(timeline, y_Aug, label="Actual", color="blue")
plt.plot(timeline, DTpredictions, label="Predicted", color="red",␣
 ↪linestyle="--")
plt.xlabel("Hour")
plt.ylabel("NO2")
plt.title("Time Series Plot of Actual vs Predicted NO2")
plt.legend()
plt.tight_layout()
plt.show()

# Scatter plot of actual vs predicted hourly values
plt.figure(figsize=(6, 4))
plt.scatter(y_Aug, DTpredictions, alpha=0.5)
plt.plot([min(y_Aug), max(y_Aug)], [min(y_Aug), max(y_Aug)], 'k--', lw=2)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Scatter Plot of Actual vs Predicted NO2")
plt.tight_layout()
plt.show()
```

```
Max depth value: 1   , RMSE: 8.281089944706608
Max depth value: 2   , RMSE: 8.16222994973839
Max depth value: 3   , RMSE: 7.993933559648002
Max depth value: 4   , RMSE: 8.222954452085157
Max depth value: 5   , RMSE: 8.641958028357159
Max depth value: 10  , RMSE: 10.413620102000205
Max depth value: 20  , RMSE: 11.609204811856682
Max depth value: 50  , RMSE: 11.61369158225894
```



With a max depth of 3, the RMSE value for decision tree regression is:
9.57763169120061





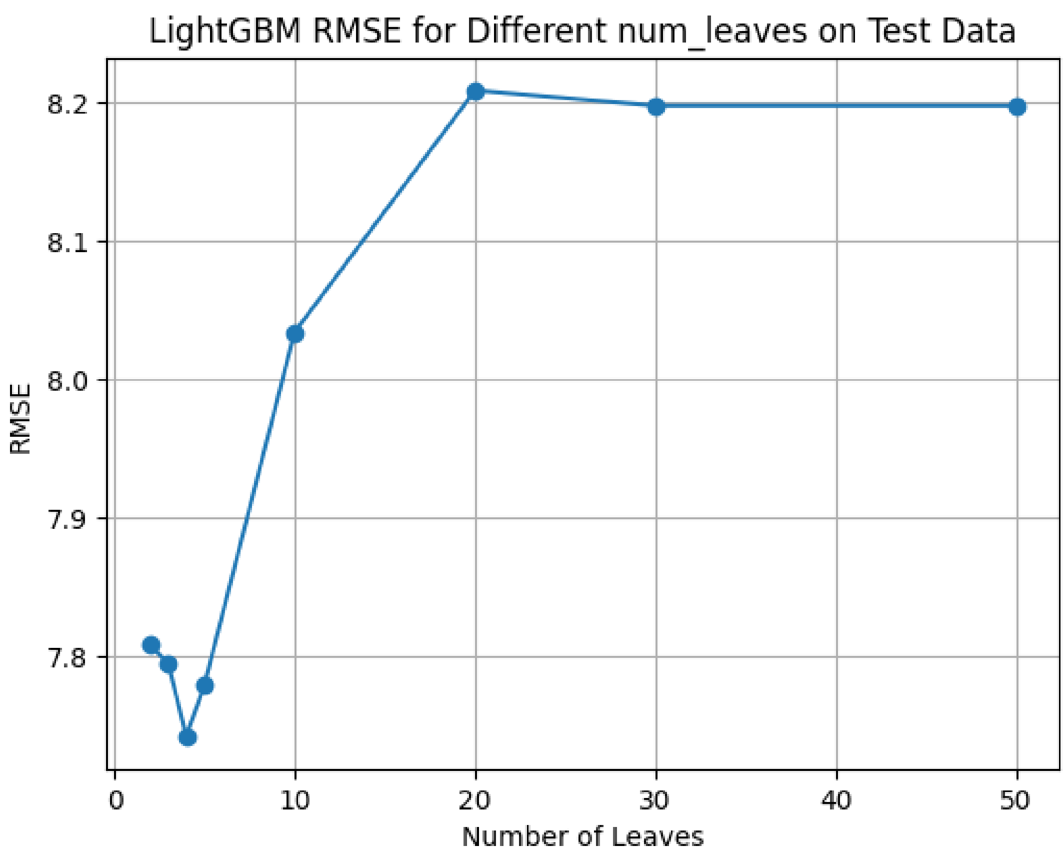
3.5.7 LGBM

```
[ ]: ################################################################################
     ###################        LGBM                              ####################
     ################################################################################

     ########################### Preparing variables ################################
     # Load the data
     July = pd.read_csv("July_clean.csv", index_col=0)
     Aug = pd.read_csv("Aug_clean.csv", index_col=0)

     # Preparing variables
     X_July = July[['Hour', 'mean_avail_diff']]
     y_July = July['NO2_mean']
     X_Aug = Aug[['Hour', 'mean_avail_diff']]
     y_Aug = Aug['NO2_mean']

     split_point = int(0.8 * len(X_July))

     # Train:0.8 Test Split:0.2 on July Data
     X_train, X_test = X_July[:split_point], X_July[split_point:]
     y_train, y_test = y_July[:split_point], y_July[split_point:]
```

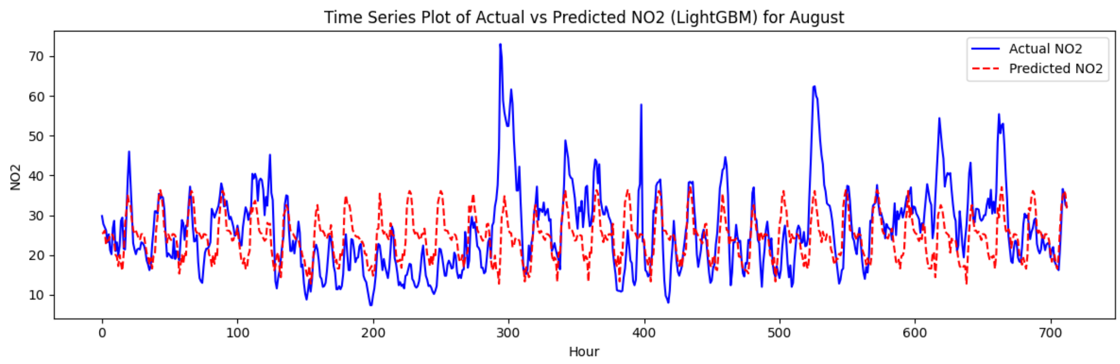
```python
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

############################       Hypertuning      ############################

rmse_list = []
num_leaves_options = [2, 3, 4, 5, 10, 20, 30, 50]

# Hypertuning based on July Data Set
for leaves in num_leaves_options:
    lgbm_model = lgb.LGBMRegressor(num_leaves=leaves, verbose=-1)
    lgbm_model.fit(X_train_scaled, y_train)
    predictions = lgbm_model.predict(X_test_scaled)
    rmse = np.sqrt(mean_squared_error(y_test, predictions))
    rmse_list.append(rmse)

# Finding the minimum RMSE and corresponding num_leaves
min_rmse = min(rmse_list)
best_leaves = num_leaves_options[rmse_list.index(min_rmse)]

# Printing the best num_leaves
print(f"Best num_leaves: {best_leaves} with RMSE: {min_rmse}")

# Plotting RMSE values for different num_leaves
plt.plot(num_leaves_options, rmse_list, marker='o')
plt.xlabel('Number of Leaves')
plt.ylabel('RMSE')
plt.title('LightGBM RMSE for Different num_leaves on Test Data')
plt.grid(True)
plt.show()

# Train the final model with the best hyperparameters
lgbm_model_final = lgb.LGBMRegressor(num_leaves=best_leaves, verbose=-1)
lgbm_model_final.fit(X_train_scaled, y_train)

############################       Prediction       ############################
# Hypertuning Done -> Select parameter num_leaves = 4
# When num_leaves = 4 it gives the minimum RMSE

X_Aug_scaled = scaler.transform(X_Aug)  # Scale the August data
LGBM_predictions = lgbm_model_final.predict(X_Aug_scaled)

# Evaluate the model on August data
LGBM_rmse = np.sqrt(mean_squared_error(y_Aug, LGBM_predictions))
print(f"RMSE for LightGBM model on August data: {LGBM_rmse}")
```

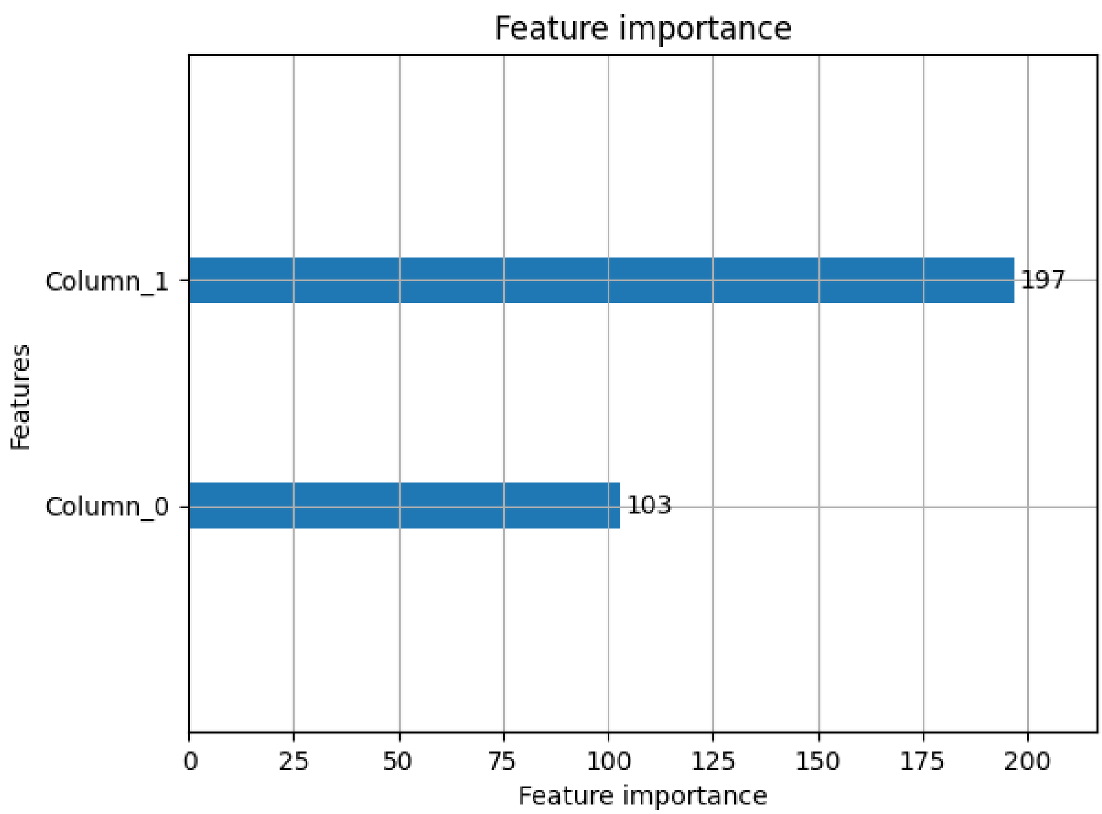
```
# Plotting
# Time series plot of the actual and predicted hourly values
plt.figure(figsize=(12, 4))
plt.plot(y_Aug.index, y_Aug, label="Actual NO2", color="blue")
plt.plot(y_Aug.index, LGBM_predictions, label="Predicted NO2", color="red",
 ↪linestyle="--")
plt.xlabel("Hour")
plt.ylabel("NO2")
plt.title("Time Series Plot of Actual vs Predicted NO2 (LightGBM) for August")
plt.legend()
plt.tight_layout()
plt.show()

lgb.plot_importance(lgbm_model_final, max_num_features=10,
 ↪importance_type='split')
plt.show()
```

```
Best num_leaves: 4 with RMSE: 7.741886702471023
```




```
RMSE for LightGBM model on August data: 9.568728471598165
```





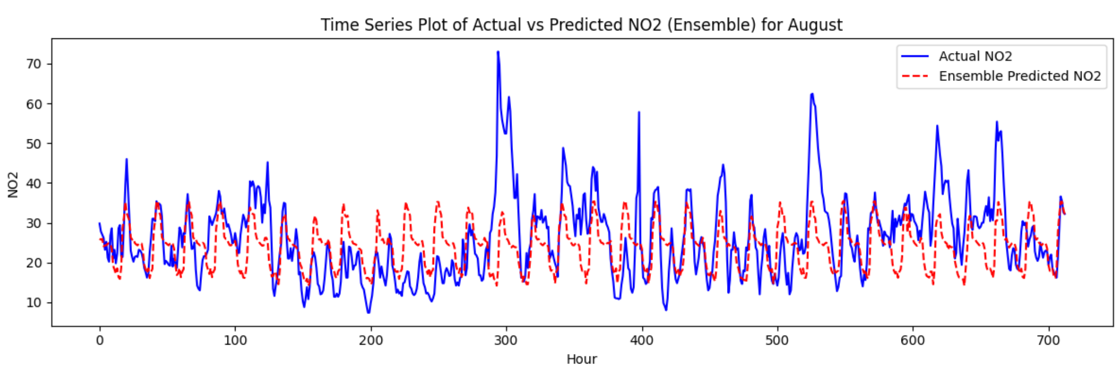
3.5.8 Ensemble

[ ]:
```
###############################################################################
###################        Ensemble                    ########################
###############################################################################

########################### Preparing variables ###############################
```

```python
# Load the data
July = pd.read_csv("July_clean.csv", index_col=0)
Aug = pd.read_csv("Aug_clean.csv", index_col=0)

# Prepare the features and target variable
X_July = July[['Hour', 'mean_avail_diff']]
y_July = July['NO2_mean']

X_Aug = Aug[['Hour', 'mean_avail_diff']]
y_Aug = Aug['NO2_mean']

split_point = int(0.8 * len(X_July))

# Train:0.8 Test Split:0.2 on July Data
X_train, X_test = X_July[:split_point], X_July[split_point:]
y_train, y_test = y_July[:split_point], y_July[split_point:]

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_Aug_scaled = scaler.transform(X_Aug)  # Scale the August data

############################       Train       ############################
# Initialize and train the models with previously tuned parameters
svr = SVR(C=10, kernel='rbf', gamma='auto')
dtr = DecisionTreeRegressor(max_depth=3)
lgbm = lgb.LGBMRegressor(num_leaves=4, verbose=-1)

# Train based on July Data Set
svr.fit(X_train_scaled, y_train)
dtr.fit(X_train_scaled, y_train)
lgbm.fit(X_train_scaled, y_train)

############################       Prediction       ############################
# Make predictions on test set
svr_predictions_test = svr.predict(X_test_scaled)
dtr_predictions_test = dtr.predict(X_test_scaled)
lgbm_predictions_test = lgbm.predict(X_test_scaled)

# Combine the predictions (simple average) for test set
ensemble_predictions_test = (svr_predictions_test + dtr_predictions_test +
 ↪lgbm_predictions_test) / 3

# Evaluate the ensemble model on test set
```

```python
ensemble_rmse_test = np.sqrt(mean_squared_error(y_test,␣
 ↪ensemble_predictions_test))
print(f"RMSE for ensemble model on test set: {ensemble_rmse_test}")

# Make predictions for August
svr_predictions_aug = svr.predict(X_Aug_scaled)
dtr_predictions_aug = dtr.predict(X_Aug_scaled)
lgbm_predictions_aug = lgbm.predict(X_Aug_scaled)

# Combine the predictions (simple average) for August
ensemble_predictions_aug = (svr_predictions_aug + dtr_predictions_aug +␣
 ↪lgbm_predictions_aug) / 3

# Evaluate the ensemble model for August
ensemble_rmse_aug = np.sqrt(mean_squared_error(y_Aug, ensemble_predictions_aug))
print(f"RMSE for ensemble model for August: {ensemble_rmse_aug}")

# Plotting
# Time series plot of the actual and predicted hourly values
plt.figure(figsize=(12, 4))
plt.plot(y_Aug.index, y_Aug, label="Actual NO2", color="blue")
plt.plot(y_Aug.index, ensemble_predictions_aug, label="Ensemble Predicted NO2",␣
 ↪color="red", linestyle="--")
plt.xlabel("Hour")
plt.ylabel("NO2")
plt.title("Time Series Plot of Actual vs Predicted NO2 (Ensemble) for August")
plt.legend()
plt.tight_layout()
plt.show()
```

```
RMSE for ensemble model on test set: 7.841345047784965
RMSE for ensemble model for August: 9.656860781950828
```



3.5.9 Overall

Evaluation

Model	RMSE (rounded off to 2 d.p.)
LR	10.30
LR (with windowing)	11.92
SVR	9.95
DTR	9.58
LGBM	9.57
Ensemble (SVR+DTR+LGBM)	9.66

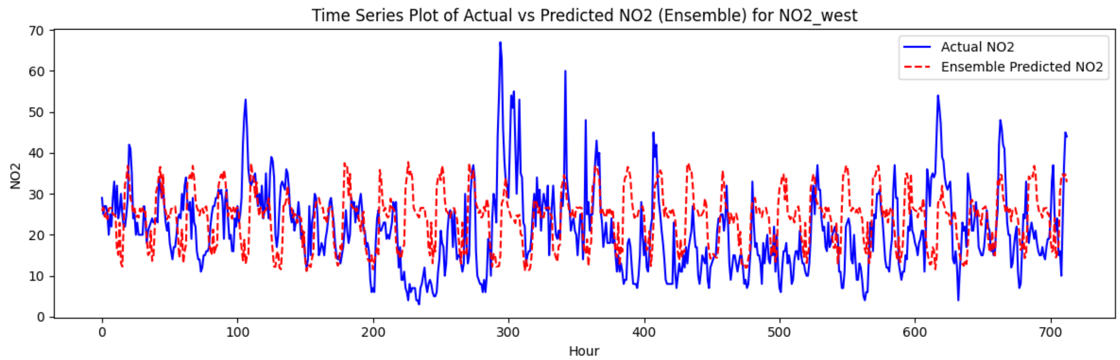
3.6 5. Analysis of selected model

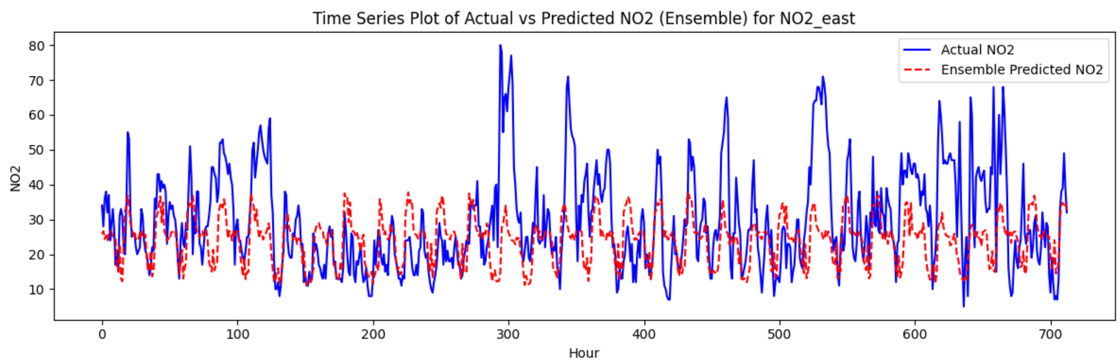
Practical application using the best model

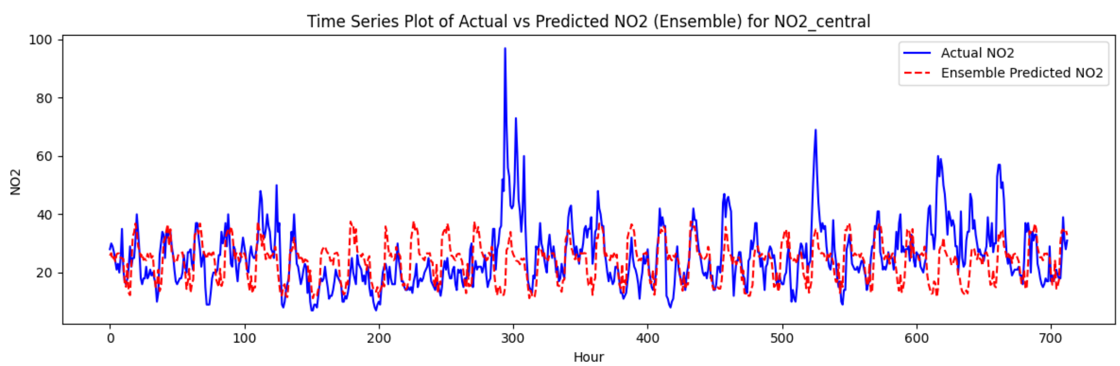
[ ]:
```
################################################################################
###################        Ensemble                         ###################
################################################################################

########################### Preparing variables ################################

# Load the data
July = pd.read_csv("July_clean.csv", index_col=0)
Aug = pd.read_csv("Aug_clean.csv", index_col=0)

# Define the regions
regions = ['NO2_west', 'NO2_east', 'NO2_central', 'NO2_south', 'NO2_north']

# Initialize the scaler
scaler = StandardScaler()

# Dictionary to store the average predicted NO2 values for each region
avg_predicted_NO2 = {'Region': [], 'Average Predicted NO2': []}

# Loop through each region to train, test, and predict
for region in regions:
    # Prepare the features and target variable
    X_July = July[['Hour', 'mean_avail_diff']]
    y_July = July[region]

    split_point = int(0.8 * len(X_July))

    # Train:0.8 Test Split:0.2 on July Data
    X_train, X_test = X_July[:split_point], X_July[split_point:]
    y_train, y_test = y_July[:split_point], y_July[split_point:]

    # Standardize the features
```

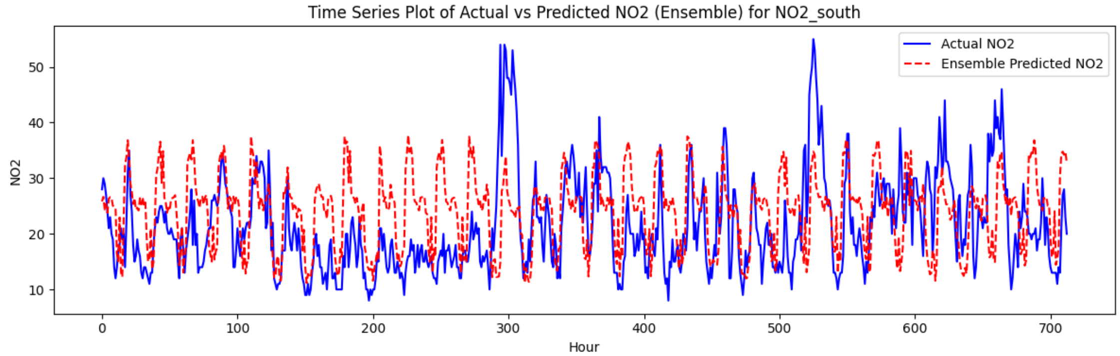
```
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
    X_Aug_scaled = scaler.transform(Aug[['Hour', 'mean_avail_diff']])

############################       Train      ############################

    # Initialize and train the models with previously tuned hyperparameters
    svr = SVR(C=10, kernel='rbf', gamma='auto')
    dtr = DecisionTreeRegressor(max_depth=3)
    lgbm = lgb.LGBMRegressor(num_leaves=30, verbose=-1)

    # Fit models using training data
    svr.fit(X_train_scaled, y_train)
    dtr.fit(X_train_scaled, y_train)
    lgbm.fit(X_train_scaled, y_train)
############################       Prediction       ############################

    # Validate models on test data (Optional: Calculate and print RMSE for␣
 ↪validation)

    # Ensemble Predictions for August
    ensemble_predictions = (svr.predict(X_Aug_scaled) +
                            dtr.predict(X_Aug_scaled) + lgbm.
 ↪predict(X_Aug_scaled)) / 3

    # Calculate and store the average predicted NO2
    avg_NO2 = np.mean(ensemble_predictions)
    avg_predicted_NO2['Region'].append(region)
    avg_predicted_NO2['Average Predicted NO2'].append(avg_NO2)

# Plotting
# Time series plot of the actual and predicted hourly values
for region in regions:
    y_Aug = Aug[region]
    plt.figure(figsize=(12, 4))
    plt.plot(y_Aug.index, y_Aug, label="Actual NO2", color="blue")
    plt.plot(y_Aug.index, ensemble_predictions, label="Ensemble Predicted NO2",␣
 ↪color="red", linestyle="--")
    plt.xlabel("Hour")
    plt.ylabel("NO2")
    plt.title(f"Time Series Plot of Actual vs Predicted NO2 (Ensemble) for␣
 ↪{region}")
    plt.legend()
    plt.tight_layout()
    plt.show()

# Convert the dictionary to a DataFrame and print it
```

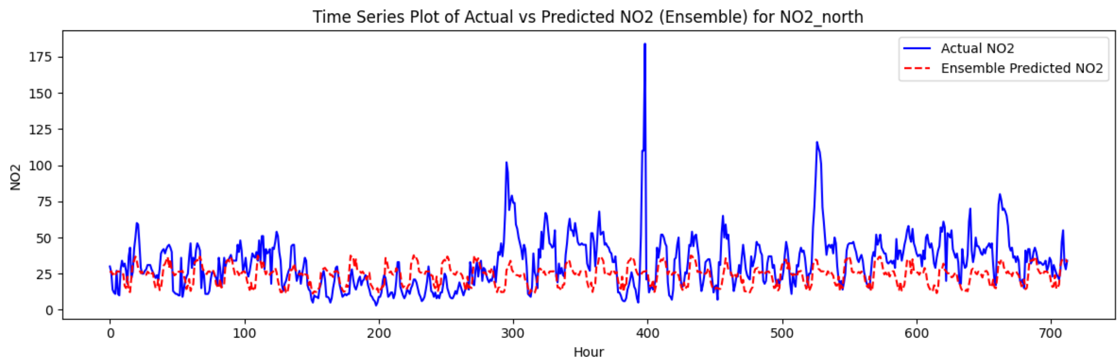
```
avg_predicted_NO2_df = pd.DataFrame(avg_predicted_NO2)
print(avg_predicted_NO2_df)
```












```
       Region  Average Predicted NO2
0    NO2_west              24.767998
1    NO2_east              26.587803
2  NO2_central             23.616286
3   NO2_south              20.495980
4   NO2_north              24.101724
```

4 Question 3.1 Justification of analysis

To forecast which car parks to upgrade, we carried out the above analysis.

In section 2, we used three datasets to obtain The location, and subsequently the region, for each respective car park by translating the X and Y coordinates to Longitude and Latitude and matching by nearest distance to the region's longitude and latitude established by the NO2 dataset The absolute difference in car park availability based on current - previous car park availability, this was done to capture the change in car park lots availability hourly The actual amount of NO2 reading. We performed data cleaning to remove any missing data and interpolated them to keep the trend of the graph accurate.

In section 3, we can observe the correlation between the overall car park availability, as well as the absolute difference in car park availability based on each region, against the actual NO2 reading in singapore. This data is useful in supporting our assumption made in question 1 as it does show correlation that high car park availability means that there will be a high number of cars on the road, which results in higher NO2 emission. This analysis was done for the whole of Singapore and for each of the 5 regions.

In section 4, we carried out model selection. Here we utilised 5 different machine learning methods: LR, SVR, DTR, LGBM, Ensemble(LR, SVR, DTR, LGBM). In order to carry out the model selection, we did a train-test split for our July data (training dataset), we then carried hyper-parameter tuning based on our train-test split of the July data and selected the best parameter based on the RMSE of each selected parameter. We then utilised the best parameter for all models and used it with the August test dataset, we then compared it to the actual data using the RMSE we identified that DTR, LGBM and Ensemble performance was close. We ended choosing Ensemble as the prediction follows the trend of actual data better and in general, Ensemble models tend to generalise better to unseen data and thus it was the model of choice for our group.

In section 5, we showed how we could utilise the selected model to carry out prediction for NO2 for each region.

This analysis is useful in the context of a Data Science Project because it allows us to identify and utilise two different but correlated data to predict the region NO2, this analysis was done with these datasets due to inaccessible data such as EV ownership by region. Which is needed for the purpose of our project goal of Forecasting car park electric vehicles charging station upgrades by region. The steps to our analysis were done with model-displicine in mind when carrying out our train-validate-test and hyper-parameter tuning, furthermore, this analysis is representative of actual real world problems where data directly related to the problem is not always available.

5 Question 3.2

Based on the insights obtained by our analysis, we have selected the Ensemble learning method for our model.

Furthermore, it can be observed that our prediction does indeed pick up the general trend where both the absolute difference in car park availability (Current - Previous car park availability, which is done to capture the change in car park availability every hour) and NO2 increases at the same timings of the day. This is because the car park availability is representative of cars being on the roads and not parked (stationary) thus leading to increase in NO2 as it is a byproduct of fuel being burnt.

The practical use case is as proposed; Forecasting car park electric vehicles charging station upgrades by region. In order to carry out this use case we could follow the steps here: 1. Predict the NO2 for each of the 5 five (North, South, East, West, Central) regions using our model and the absolute difference in car park availability. 2. From the predicted NO2 for each region, we then take the average of the predicted NO2 for each region for the month and rank them. 3. The highest ranking for NO2 is selected as the region of interest for car park EV upgrades for that month - as it is indicative of a larger number of fuel cars travelling in that region. 4. As car parks get upgraded - those car parks could be removed from the data set, as it is assumed that once an upgrade is done, no more upgrades would be carried out on the same car park (at least for the lifetime of utilising this model to prioritise EV upgrades)

Thus our use case allows us to forecast regions that should have their car parks upgraded due to the high number of fuel operated cars in the region, encouraging the residents in the area to swap to EV as one of the main concerns with adoption of EV is the convenience to charge these vehicles.

Below, is the ranking for Predicted August NO2 by region, the average was calculated and used in avg_predicted_NO2_df dataframe for our analysis of which region should carpark EV upgrade be prioritised.

```
[ ]: #From the analysis in Section 5, we obtained the August predicted NO2 for each␣
      ↪region
     df_sorted = avg_predicted_NO2_df.sort_values(by='Average Predicted NO2',␣
      ↪ascending=False)
     df_sorted.reset_index(drop=True, inplace=True)

     # Print the sorted DataFrame
     print("NO2 Ranking by Region:")
     print(df_sorted)
```

```
NO2 Ranking by Region:
        Region  Average Predicted NO2
0     NO2_east              26.587803
1     NO2_west              24.767998
2    NO2_north              24.101724
3  NO2_central              23.616286
4    NO2_south              20.495980
```