In [ ]:

```
import pandas as pd
import geopandas as gpd
import numpy as np

import glob

from shapely.ops import unary_union

import matplotlib.pyplot as plt
import seaborn as sns

from libpysal.weights import Kernel
from esda.moran import Moran

from scipy import stats
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, LeaveOneOut,
cross_val_predict
from sklearn.inspection import permutation_importance
```

Data cleaning

In [ ]:

```
# read in all PM data
csv_files = glob.glob('data/AQMS' + '/*.csv')
df = pd.concat((pd.read_csv(f) for f in csv_files))
```

In [ ]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 402960 entries, 0 to 52559
Data columns (total 6 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Site                      402960 non-null  object
 1   Species                   402960 non-null  object
 2   ReadingDateTime           402960 non-null  object
 3   Value                     202670 non-null  float64
 4   Units                     402960 non-null  object
 5   Provisional or Ratified   402960 non-null  object
dtypes: float64(1), object(5)
memory usage: 21.5+ MB
```

In [ ]:

```
# drop unnecessary columns
df.drop(['Species', 'Units', 'Provisional or Ratified'], axis=1,
```

```
inplace=True)
```

In [ ]:

```
df.groupby('Site').describe()
```

	Value							
	count	mean	std	min	25%	50%	75%	max
Site								
BL0	8558.0	10.750888	10.112520	-3.3	4.7	7.6	12.7	92.40000
BQ9	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BT4	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
BX9	7169.0	11.813182	10.972091	-3.8	5.3	7.9	13.8	88.10000
BY7	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
CD1	8544.0	11.132924	10.262592	-2.8	4.9	7.8	13.4	88.30000
CD9	8730.0	13.642887	10.411786	-7.3	7.2	10.9	16.3	83.90000
CE2	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
CR8	8711.0	10.115831	9.176507	-3.0	5.0	7.0	12.0	84.00000
CT2	8437.0	13.957568	10.865349	-3.0	8.0	11.0	16.0	441.00000
CT3	7575.0	11.669967	10.486332	-3.0	6.0	9.0	15.0	251.00000
GB0	8637.0	12.176705	9.036808	-1.2	6.7	9.4	14.1	79.80000
GN0	3193.0	11.319449	9.740894	-7.2	4.9	8.3	14.9	65.10000
GN3	8342.0	13.411832	11.277777	-3.5	6.8	9.6	15.5	109.40000
GN6	8252.0	10.966893	9.999743	-4.2	5.1	7.7	12.5	84.10000
GR4	8516.0	10.863269	9.913018	-2.7	5.2	8.0	12.5	97.60000
GR8	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
GR9	8713.0	10.425215	10.639660	-4.3	4.0	6.9	12.6	84.50000
HG1	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
HP1	8756.0	9.933029	9.987813	0.4	4.2	6.5	11.3	90.90000
HR1	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
HV1	8403.0	11.004820	12.916148	-9.0	4.5	7.8	13.5	472.20001
KC1	8723.0	9.579548	9.470490	0.4	4.2	6.3	11.0	121.00000
KF1	8723.0	9.578723	9.470523	0.4	4.1	6.4	11.0	121.00000
LH0	8510.0	9.538249	9.165456	0.4	4.1	6.3	11.2	91.00000
LW2	7742.0	14.953500	11.325410	-6.2	8.0	11.6	17.9	92.60000
LW5	411.0	8.671533	7.945340	-4.0	3.0	7.0	13.0	40.00000
MY7	7948.0	14.347974	10.963840	-3.3	7.5	11.4	17.5	93.00000
NM2	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	Value							
	count	mean	std	min	25%	50%	75%	max
Site								
NM3	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
RB7	3399.0	11.471315	10.159918	-4.0	6.0	9.0	15.0	272.00000
RD0	3249.0	8.211480	8.305129	-5.0	3.0	6.0	11.0	80.00000
SK6	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SK8	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SK9	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SKA	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SKB	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SKC	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ST5	8648.0	11.718316	10.163471	-7.0	6.0	9.0	14.0	99.00000
TD5	8148.0	11.784892	14.797966	0.0	5.8	8.6	13.4	592.79999
TH4	6478.0	13.380195	11.332006	-5.3	6.5	9.7	16.1	152.30000
TK3	7940.0	11.548237	11.593650	-2.0	5.0	8.0	14.0	111.00000
TK9	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
TL6	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
WM0	2215.0	11.681716	9.942168	0.0	7.0	9.0	14.0	339.00000
WMD	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

GN0, LW5, RB7, RD0, WM0 has too few data (less than half of the total amount)

In [ ]:

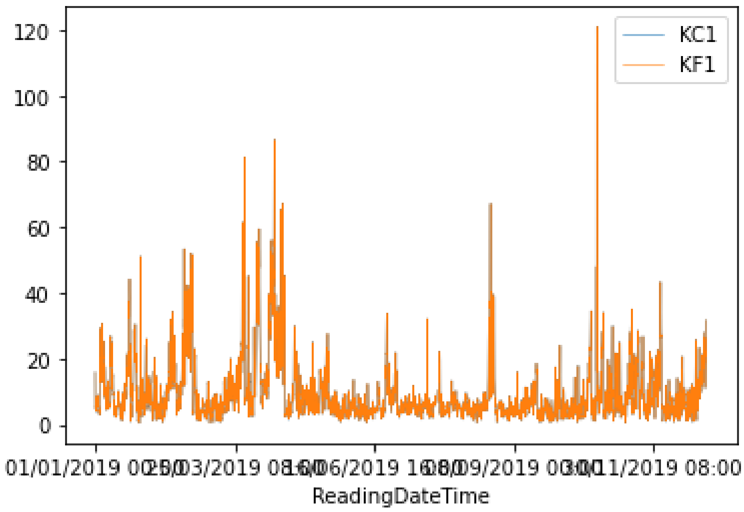
```
# list of site codes with valid PM data
valid_AQMS = df.dropna()['Site'].unique().tolist()

# remove the site with few data from the list
for site in ['GN0', 'LW5', 'RB7', 'RD0', 'WM0']:
    valid_AQMS.remove(site)

# clean the PM dataset
df = df[df['Site'].isin(valid_AQMS)]
df = df.reset_index(drop=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201480 entries, 0 to 201479
Data columns (total 3 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Site            201480 non-null  object
```

```
 1   ReadingDateTime  201480 non-null  object
 2   Value            190203 non-null  float64
dtypes: float64(1), object(2)
memory usage: 4.6+ MB
```

In [ ]:

```python
# KF1 and KC1 are very similar
fig,ax = plt.subplots()
df[df['Site']=='KC1'].plot(x='ReadingDateTime', y='Value', ax=ax,
label='KC1', linewidth=0.5)
df[df['Site']=='KF1'].plot(x='ReadingDateTime', y='Value', ax=ax,
label='KF1', linewidth=0.5)
plt.show()
```



In [ ]:

```python
# Remove KF1
df = df[df['Site']!='KF1']
valid_AQMS.remove('KF1')
```

In [ ]:

```python
len(df['Site'].unique())
```

22

In [ ]:

```python
df.groupby('Site').describe()
```

	Value							
	count	mean	std	min	25%	50%	75%	max
Site								
BL0	8558.0	10.750888	10.112520	-3.3	4.7	7.6	12.7	92.40000
BX9	7169.0	11.813182	10.972091	-3.8	5.3	7.9	13.8	88.10000
CD1	8544.0	11.132924	10.262592	-2.8	4.9	7.8	13.4	88.30000
CD9	8730.0	13.642887	10.411786	-7.3	7.2	10.9	16.3	83.90000
CR8	8711.0	10.115831	9.176507	-3.0	5.0	7.0	12.0	84.00000
CT2	8437.0	13.957568	10.865349	-3.0	8.0	11.0	16.0	441.00000

	Value							
	count	mean	std	min	25%	50%	75%	max
Site								
CT3	7575.0	11.669967	10.486332	-3.0	6.0	9.0	15.0	251.00000
GB0	8637.0	12.176705	9.036808	-1.2	6.7	9.4	14.1	79.80000
GN3	8342.0	13.411832	11.277777	-3.5	6.8	9.6	15.5	109.40000
GN6	8252.0	10.966893	9.999743	-4.2	5.1	7.7	12.5	84.10000
GR4	8516.0	10.863269	9.913018	-2.7	5.2	8.0	12.5	97.60000
GR9	8713.0	10.425215	10.639660	-4.3	4.0	6.9	12.6	84.50000
HP1	8756.0	9.933029	9.987813	0.4	4.2	6.5	11.3	90.90000
HV1	8403.0	11.004820	12.916148	-9.0	4.5	7.8	13.5	472.20001
KC1	8723.0	9.579548	9.470490	0.4	4.2	6.3	11.0	121.00000
LH0	8510.0	9.538249	9.165456	0.4	4.1	6.3	11.2	91.00000
LW2	7742.0	14.953500	11.325410	-6.2	8.0	11.6	17.9	92.60000
MY7	7948.0	14.347974	10.963840	-3.3	7.5	11.4	17.5	93.00000
ST5	8648.0	11.718316	10.163471	-7.0	6.0	9.0	14.0	99.00000
TD5	8148.0	11.784892	14.797966	0.0	5.8	8.6	13.4	592.79999
TH4	6478.0	13.380195	11.332006	-5.3	6.5	9.7	16.1	152.30000
TK3	7940.0	11.548237	11.593650	-2.0	5.0	8.0	14.0	111.00000

In [ ]:

```
# read in AQMS location geometry
gdf = gpd.read_file('data/AQMS/AQMS.gpkg')
gdf.head()
```

```
C:\Users\Yulun\anaconda3\envs\sds2021\lib\site-packages\geopandas\geodatafr
ame.py:577: RuntimeWarning: Sequential read of iterator was interrupted. Re
setting iterator. This can negatively impact the performance.
  for feature in features_lst:
```

	classification	dataowner	easting	latitude	longitude
0	Airport	None	542525.2800145757	51.5028	0.0521
1	Airport	None	542948.1357935619	51.5028	0.058193
2	Breathe London	None	535618.12376207381	51.521017999999998	-0.046672999999999999

	classification	dataowner	easting	latitude	longitude
3	Airport	None	542295.805364199	51.5074	0.049
4	Breathe London	None	524303.28797191242	51.604480000000002	-0.206490000000000001

5 rows × 21 columns

In [ ]:
```
# drop unnecessary columns
gdf = gdf.loc[:,['latitude', 'longitude', 'siteid', 'sitename']]
gdf.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 236 entries, 0 to 235
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   latitude   236 non-null    object
 1   longitude  236 non-null    object
 2   siteid     236 non-null    object
 3   sitename   236 non-null    object
dtypes: object(4)
memory usage: 7.5+ KB
```

In [ ]:
```
# check if all sites with data are within the geometry dataframe
for elem in valid_AQMS:
    if elem not in gdf['siteid'].unique().tolist():
        print(elem)
```

```
TK3
```

TK3: Thurrock - Stanford-le-Hope

```
51.518162000000, 0.4395480000000
```

Thurrock is not in London, so ignore

In [ ]:
```
# remove TK3 from the list and the dataframe
valid_AQMS.remove('TK3')
df = df[df['Site']≠'TK3']
```

In [ ]:
```
len(valid_AQMS)
```

```
21
```

In [ ]:
```
len(df['Site'].unique())
```

```
21
```

In [ ]:
```
# get the geometry of the 21 sites
AQMS_gdf = gdf[gdf['siteid'].isin(valid_AQMS)]
AQMS_gdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21 entries, 115 to 232
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   latitude   21 non-null     object
 1   longitude  21 non-null     object
 2   siteid     21 non-null     object
 3   sitename   21 non-null     object
dtypes: object(4)
memory usage: 840.0+ bytes
```

In [ ]:
```
# set to proper data tyeps
AQMS_gdf = AQMS_gdf.astype({'latitude':'float64',
'longitude':'float64',
                             'siteid':'string', 'sitename':'string'})
AQMS_gdf.dtypes
```

```
latitude      float64
longitude     float64
siteid         string
sitename       string
dtype: object
```

In [ ]:
```
# generate geometry column based on lat and lon
AQMS_gdf = gpd.GeoDataFrame(AQMS_gdf,

geometry=gpd.points_from_xy(AQMS_gdf.longitude, AQMS_gdf.latitude),
                             crs='EPSG:4326')
```

In [ ]:
```
# set the crs to british national grid
AQMS_gdf = AQMS_gdf.to_crs(27700)

# drop the lat and lon columns
AQMS_gdf = AQMS_gdf.drop(['latitude', 'longitude'], axis=1)
```

In [ ]:
```
# save the geometry for future use
AQMS_gdf.to_file('data/AQMS_loc.shp')
```

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 183960 entries, 0 to 201479
Data columns (total 3 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   Site             183960 non-null  object 
 1   ReadingDateTime  183960 non-null  object 
 2   Value            173540 non-null  float64
dtypes: float64(1), object(2)
memory usage: 5.6+ MB
```

In [ ]:
```
df['Value'].describe()
```

```
count    173540.000000
mean         11.721337
std          10.793000
min          -9.000000
25%           5.400000
50%           8.600000
75%          14.000000
max         592.799990
Name: Value, dtype: float64
```

There are many null values and negative values (not efficient because PM readings cannot be negative)

According to this, using mean-before-after is an approach.

In [ ]:
```
# set all negative reading to np.nan
# because you cannot have negative reading for PM concentration
df['Value'] = df['Value'].where(df['Value']>0, np.nan)
```

In [ ]:
```
val = df['Value'].values.copy()
```

In [ ]:
```
# check the number of null values
sum(val<=0), sum(np.isnan(val))
```

```
(0, 11847)
```

In [ ]:
```
# make sure that every site's first and last value is not null
for s in range(21):
    print(val[s*8760], val[s*8760-1])
```

```
13.0 31.3
17.0 29.0
36.0 33.0
21.4 35.0
16.1 30.0
29.1 23.6
23.3 30.0
53.3 38.9
43.1 31.2
14.7 22.6
```

```
15.8 34.5
11.2 31.7
14.0 33.9
26.3 37.0
22.0 34.8
35.1 33.0
20.4 28.6
20.5 32.9
11.0 30.6
18.2 33.9
35.5 24.1
```

In [ ]:

```
# fill null values using mean-before-after method
for i in range(len(val)):
    if np.isnan(val[i]):
        j = 1
        while (np.isnan(val[i+j])) & (j < 12):
            j += 1

        # if there are 12 continous null values,
        # fill them with data for the same period for the previous day
        if j==12:
            for z in range(j+1):
                val[i+z] = val[i+z-24]
        val[i] = val[i-1] + (val[i+j] - val[i-1]) / (j+1)
```

In [ ]:

```
sum(val<=0), sum(np.isnan(val))
```

```
(0, 0)
```

In [ ]:

```
# cover the data in the df
df['Value'] = val
```

In [ ]:

```
df.describe()
```

	Value
count	183960.000000
mean	11.775446
std	10.661761
min	0.100000
25%	5.500000
50%	8.600000
75%	14.000000
max	592.799990

In [ ]:

```
df.groupby('Site').describe()
```

	Value							
	count	mean	std	min	25%	50%	75%	max
Site								
BL0	8760.0	10.908521	10.228363	0.1	4.7	7.600000	12.800000	92.40000
BX9	8760.0	11.170749	10.396609	0.2	5.3	7.380917	12.600000	88.10000
CD1	8760.0	11.058464	10.162193	0.1	4.8	7.800000	13.300000	88.30000
CD9	8760.0	13.712563	10.330619	0.1	7.3	10.900000	16.300000	83.90000
CR8	8760.0	10.125421	9.129344	1.0	5.0	7.000000	12.000000	84.00000
CT2	8760.0	13.902287	10.708376	1.0	8.0	11.000000	16.000000	441.00000
CT3	8760.0	12.142583	10.057463	1.0	6.0	9.000000	16.000000	251.00000
GB0	8760.0	12.569166	9.864263	0.1	6.8	9.400000	14.325000	79.80000
GN3	8760.0	13.363480	11.089737	0.1	6.7	9.600000	15.700000	109.40000
GN6	8760.0	11.039737	9.829372	0.1	5.2	7.800000	12.700000	84.10000
GR4	8760.0	10.887037	9.764517	0.1	5.3	8.000000	12.600000	97.60000
GR9	8760.0	10.482015	10.585919	0.1	4.0	7.000000	12.600000	84.50000
HP1	8760.0	9.931490	9.985798	0.4	4.2	6.500000	11.300000	90.90000
HV1	8760.0	11.368690	12.671719	0.1	4.8	7.900000	13.600000	472.20001
KC1	8760.0	9.567551	9.452367	0.4	4.2	6.400000	11.000000	121.00000
LH0	8760.0	9.412646	9.069698	0.4	4.1	6.300000	10.925000	91.00000
LW2	8760.0	15.422345	11.470787	0.3	8.4	12.000000	18.200000	92.60000
MY7	8760.0	14.190663	10.758123	0.1	7.3	11.400000	17.400000	93.00000
ST5	8760.0	11.732403	10.095116	1.0	6.0	9.000000	14.000000	99.00000
TD5	8760.0	11.686217	14.387273	0.1	5.9	8.600000	13.400000	592.79999
TH4	8760.0	12.610338	10.015622	0.1	6.9	9.700000	14.561054	152.30000

In [ ]:

```
# save for future use
df.to_csv('data/hourly.csv', index=False)
```

Load in data

In [ ]:

```
# set seaborn theme
sns.set_theme(style='darkgrid')
```

In [ ]:

```
# read in AQMS locations
loc_gdf = gpd.read_file('data/AQMS_loc.shp')
```

```python
# read in PM2.5 hourly data
dep_df = pd.read_csv('data/hourly.csv')
```

In [ ]:
```python
# set buffer zones around each site (1km)
loc_gdf['buffer_1km'] = loc_gdf['geometry'].buffer(1000)
```

road modify

In [ ]:
```python
LD_wards = gpd.read_file("data/LD_boundary/London-wards-2018_ESRI/London_Ward_CityMerged.shp")
london = LD_wards.unary_union
london_gdf = gpd.GeoSeries(london)
london_gdf.to_file('data/london_boundary.shp')
```

In [ ]:
```python
for typ in ['RoadLink', 'RoadNode', 'MotorwayJunction']:
    exec("%s = gpd.GeoDataFrame()"%typ)
    for tile in ['SP_','SU_','TL_','TQ_']:
        path = "data/oproad_essh_gb/data/%s%s.shp"%(tile, typ)
        exec("%s%s = gpd.read_file(path)"%(tile, typ))
        exec("%s = %s.append(%s%s, ignore_index=True)"%(typ, typ, tile, typ))
```

In [ ]:
```python
spatial_index = RoadLink.sindex
bbox = london.bounds
sidx = list(spatial_index.intersection(bbox))
RoadLink_sub = RoadLink.iloc[sidx]

RoadLink_clip = RoadLink_sub.copy()
RoadLink_clip['geometry'] = RoadLink_sub.intersection(london)
```

In [ ]:
```python
RoadLink_clip = RoadLink_clip.reset_index(drop=True)
Rd = RoadLink_clip[RoadLink_clip['geometry'] != RoadLink_clip.loc[0, 'geometry']].reset_index(drop=True)
Rd.head()
```

In [ ]:
```python
Rd.to_file('data/london_Road.shp')
```

gsp modify

In [ ]:

```python
# for downloading greenspace geometry
buffer_gdf = loc_gdf[['buffer_1km']]
buffer_gdf = gpd.GeoDataFrame(buffer_gdf, geometry='buffer_1km')
buffer_gdf.to_file('data/buffer.shp')
```

In [ ]:

```python
loc_gdf
```

0575 - LH0

1065, 1070, 1565, 1570 - TD5

2565, 2570, 3065, 3070 - CR8

2565, 3065 - ST5

2080, 2580 - KC1

2580, 2585 - CD1

2580 - MY7

2580, 3080 - BL0, CD9

3080 - CT2, CT3

3570, 3575 - HP1, LW2

3575, 3580, 4075, 4080 - GN6

3580 - TH4

4070, 4075, 4570, 4575 - GB0

4070, 4075 - GR9, GR4

4075, 4575 - GN3

5075 - BX9

5080 - HV1

In [ ]:

```python
def readin_Gsp(file_name, path='data/OSMM Greenspaces/tq/TQ',
suffix='_GreenspaceArea.shp'):
    if type(file_name) == str:
        gdf = gpd.read_file(path+file_name+suffix)
    else:
        gdf = pd.concat(gpd.read_file(path+f+suffix) for f in
file_name)
    return gdf
```

In [ ]:

```python
loc_gdf['Gsp'] = gpd.GeoSeries()
```

```
In [ ]: loc_gdf.columns.get_loc('Gsp')
```

```
In [ ]: def get_Gsp(file_name, index):
            gdf = readin_Gsp(file_name)
            print('Finish reading in shapefile(s)')
            shp = gdf['geometry'].unary_union
            print('Finish unary union.')
            if type(index) == int:
                loc_gdf.iat[index, 4] = shp.intersection(loc_gdf.loc[index,
        'buffer_1km'])
            elif type(index) == list:
                for i in index:
                    loc_gdf.iat[i, 4] = shp.intersection(loc_gdf.loc[i,
        'buffer_1km'])
            else:
                print('invalid type!')
```

```
In [ ]: get_Gsp('0575', 13)
```

```
In [ ]: get_Gsp(['1065','1070','1565','1570'], 17)
```

```
In [ ]: get_Gsp(['2565','2570','3065','3070'], 6)
```

```
In [ ]: get_Gsp(['2565','3065'], 18)
```

```
In [ ]: get_Gsp(['2080','2580'], 14)
```

```
In [ ]: get_Gsp(['2580','2585'], 3)
```

```
In [ ]: get_Gsp('2580', 20)
```

```
In [ ]: get_Gsp(['2580','3080'], [1,2])
```

```
In [ ]: get_Gsp('3080', [4,5])
```

```
In [ ]: get_Gsp(['3570','3575'], [15,16])
```

```
In [ ]: get_Gsp(['3575','3580','4075','4080'], 9)
```

In [ ]:
```python
get_Gsp('3580', 19)
```

In [ ]:
```python
get_Gsp(['4070','4075','4570','4575'], 8)
```

In [ ]:
```python
get_Gsp(['4075', '4575'], [7, 11])
```

In [ ]:
```python
get_Gsp(['4075', '4575'], 10)
```

In [ ]:
```python
get_Gsp('5075', 0)
```

In [ ]:
```python
get_Gsp('5080', 12)
```

In [ ]:
```python
Gsp_gdf = loc_gdf[['siteid','Gsp']]
Gsp_gdf = Gsp_gdf.set_geometry('Gsp')
Gsp_gdf = Gsp_gdf.set_crs(27700)
Gsp_gdf.crs
```

In [ ]:
```python
Gsp_gdf.to_file('data/gsp_buffer_1km.shp')
```

generate near-road gsp

In [ ]:
```python
# read in (modified) greenspace geometry
Gsp_gdf = gpd.read_file('data/gsp_buffer_1km.shp')
```

In [ ]:
```python
# add the gsp geometry column to the location gdf
loc_gdf['Gsp'] = Gsp_gdf['geometry']
loc_gdf.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   siteid      21 non-null     object
 1   sitename    21 non-null     object
 2   geometry    21 non-null     geometry
 3   buffer_1km  21 non-null     geometry
 4   Gsp         21 non-null     geometry
dtypes: geometry(3), object(2)
memory usage: 968.0+ bytes
```

In [ ]:
```python
# save memory
del Gsp_gdf
```

In [ ]:

```python
# Read in road (modified) geometry
Rd_gdf = gpd.read_file('data/london_Road.shp')
Rd_gdf.head()
```

Out[ ]:

	fictitious	identifier	class	roadNumber	name1	name1_lang	name2	name:
0	false	8CC0934A-4A4A-435A-BEBB-521AD3E8C143	Not Classified	None	The Bridlepath	None	None	
1	false	ECE86DA8-118A-46AB-8D5D-56F68B96E7BB	Unclassified	None	Ditches Lane	None	None	
2	false	960A1B1E-15CD-4E9C-816C-4F79CB0442E7	A Road	A233	Main Road	None	None	
3	false	0E0182BB-7E46-4250-B9EE-37D58BA0E73C	Unclassified	None	Grays Road	None	None	
4	false	A6456BD8-2D7F-4CE9-9192-112965FA7AD1	Unclassified	None	Old Fox Close	None	None	

In [ ]:

```python
for c in Rd_gdf['class'].unique():
    print('Number of ' + c + ': ', Rd_gdf[Rd_gdf['class'] ==
c].shape[0])
```

```
Number of Not Classified:  14061
Number of Unclassified:  117392
Number of A Road:  25452
Number of B Road:  6734
Number of Unknown:  36448
Number of Classified Unnumbered:  8925
Number of Motorway:  189
```

In [ ]:

```python
# Get all types of roads
Rd = {}
for c in Rd_gdf['class'].unique():
    Rd[c] = Rd_gdf[Rd_gdf['class'] == c].loc[:, 'geometry'].unary_union
Rd
```

Out[ ]:

```
{'Not Classified': <shapely.geometry.multilinestring.MultiLineString at 0x1
ef30405670>,
 'Unclassified': <shapely.geometry.multilinestring.MultiLineString at 0x1ef
275d6e80>,
 'A Road': <shapely.geometry.multilinestring.MultiLineString at 0x1ef304055
```

```
e0>,
 'B Road': <shapely.geometry.multilinestring.MultiLineString at 0x1ef276997
60>,
 'Unknown': <shapely.geometry.multilinestring.MultiLineString at 0x1ef302cc
0d0>,
 'Classified Unnumbered': <shapely.geometry.multilinestring.MultiLineString
at 0x1ef275d6dc0>,
 'Motorway': <shapely.geometry.multilinestring.MultiLineString at 0x1ef3040
5580>}
```

In [ ]:

```
# merge Not Classified and Unknown into one category
Rd['Other'] = unary_union([Rd['Not Classified'], Rd['Unknown']])
Rd.pop('Not Classified')
Rd.pop('Unknown')
Rd
```

Out[ ]:

```
{'Unclassified': <shapely.geometry.multilinestring.MultiLineString at 0x1ef
275d6e80>,
 'A Road': <shapely.geometry.multilinestring.MultiLineString at 0x1ef304055
e0>,
 'B Road': <shapely.geometry.multilinestring.MultiLineString at 0x1ef276997
60>,
 'Classified Unnumbered': <shapely.geometry.multilinestring.MultiLineString
at 0x1ef275d6dc0>,
 'Motorway': <shapely.geometry.multilinestring.MultiLineString at 0x1ef3040
5580>,
 'Other': <shapely.geometry.multilinestring.MultiLineString at 0x1ef275c322
0>}
```

In [ ]:

```
del Rd_gdf
```

In [ ]:

```
# add the road geometries to the location gdf
for key in Rd.keys():
    loc_gdf[key] = loc_gdf['buffer_1km'].intersection(Rd[key])
loc_gdf.head()
```

Out[ ]:

	siteid	sitename	geometry	buffer_1km	Gsp	Unclassified	A
0	BX9	Bexley - Slade Green FDMS	POINT (551862.205 176375.976)	POLYGON ((552862.205 176375.976, 552857.390 17...	MULTIPOLYGON Z (((551468.680 175909.000 0.000,...	MULTILINESTRING Z ((552075.170 175434.690 0.00...	MULTILINES Z ((5524 175621.08(
1	BL0	Camden - Bloomsbury	POINT (530120.048 182038.807)	POLYGON ((531120.048 182038.807, 531115.233 18...	MULTIPOLYGON Z (((530046.600 181557.850 0.000,...	MULTILINESTRING Z ((530175.051 181041.510 0.00...	MULTILINES Z ((5299 181058.68(

	siteid	sitename	geometry	buffer_1km	Gsp	Unclassified	A
2	CD9	Camden - Euston Road	POINT (529900.870 182666.124)	POLYGON ((530900.870 182666.124, 530896.055 18...	MULTIPOLYGON Z (((530164.744 181702.568 0.000,...	MULTILINESTRING Z ((529650.665 181699.144 0.00...	MULTILINES Z ((5299 181667.10
3	CD1	Camden - Swiss Cottage	POINT (526629.730 184391.024)	POLYGON ((527629.730 184391.024, 527624.915 18...	MULTIPOLYGON Z (((527127.744 183525.057 0.000,...	MULTILINESTRING Z ((526949.610 183444.673 0.00...	MULTILINES Z ((5266 183408.05
4	CT2	City of London - Farringdon Street	POINT (531622.273 181213.818)	POLYGON ((532622.273 181213.818, 532617.458 18...	MULTIPOLYGON Z (((532257.200 181585.050 0.000,...	MULTILINESTRING Z ((531742.953 180221.995 0.00...	MULTILINES Z ((5316 180215.42

In [ ]:

```
del Rd
```

In [ ]:

```
# Rename the columns
loc_gdf.rename(columns={'Unclassified': 'UnC',
                        'A Road': 'A',
                        'B Road': 'B',
                        'Classified Unnumbered': 'CUn',
                        'Motorway': 'Mt'}, inplace=True)

# save the road classification to a list
Rd_type = loc_gdf.columns[-6:].tolist()
Rd_type
```

Out[ ]:

```
['UnC', 'A', 'B', 'CUn', 'Mt', 'Other']
```

In [ ]:

```
# Get all near-road greenspaces
for col in Rd_type:
    loc_gdf['n'+col+'_Gsp'] =
loc_gdf['Gsp'].intersection(loc_gdf[col].buffer(50))

loc_gdf.head()
```

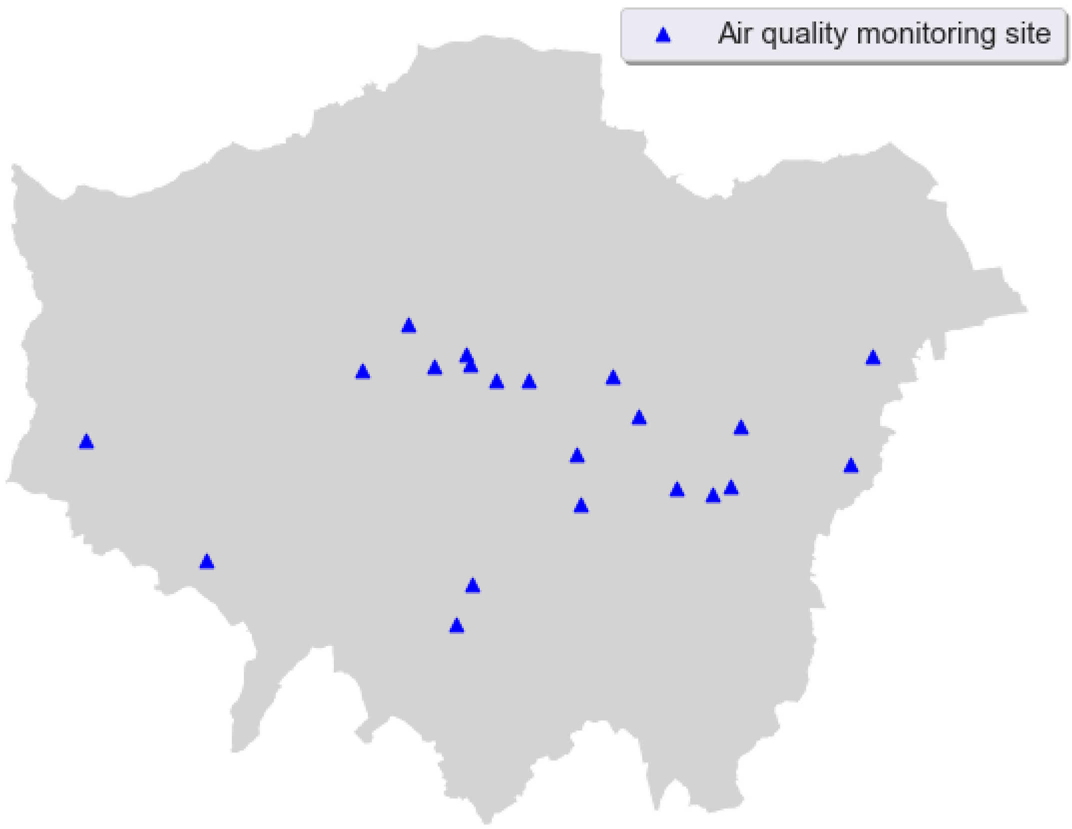
Out[ ]:

	siteid	sitename	geometry	buffer_1km	Gsp	UnC	
0	BX9	Bexley - Slade Green FDMS	POINT (551862.205 176375.976)	POLYGON ((552862.205 176375.976, 552857.390 17...	MULTIPOLYGON Z (((551468.680 175909.000 0.000,...	MULTILINESTRING Z ((552075.170 175434.690 0.00...	MULTILINES Z ((5524 175621.08

	siteid	sitename	geometry	buffer_1km	Gsp	UnC	
1	BL0	Camden - Bloomsbury	POINT (530120.048 182038.807)	POLYGON ((531120.048 182038.807, 531115.233 18...	MULTIPOLYGON Z (((530046.600 181557.850 0.000,...	MULTILINESTRING Z ((530175.051 181041.510 0.00...	MULTILINES Z ((5299 181058.68(
2	CD9	Camden - Euston Road	POINT (529900.870 182666.124)	POLYGON ((530900.870 182666.124, 530896.055 18...	MULTIPOLYGON Z (((530164.744 181702.568 0.000,...	MULTILINESTRING Z ((529650.665 181699.144 0.00...	MULTILINES Z ((5299 181667.10:
3	CD1	Camden - Swiss Cottage	POINT (526629.730 184391.024)	POLYGON ((527629.730 184391.024, 527624.915 18...	MULTIPOLYGON Z (((527127.744 183525.057 0.000,...	MULTILINESTRING Z ((526949.610 183444.673 0.00...	MULTILINES Z ((5266 183408.05(
4	CT2	City of London - Farringdon Street	POINT (531622.273 181213.818)	POLYGON ((532622.273 181213.818, 532617.458 18...	MULTIPOLYGON Z (((532257.200 181585.050 0.000,...	MULTILINESTRING Z ((531742.953 180221.995 0.00...	MULTILINES Z ((5316 180215.42:

In [ ]:

```
# Fig 1
fig,ax = plt.subplots(1, figsize=(12,8))

london_gdf.plot(color='lightgrey', ax=ax)

loc_gdf['geometry'].plot(markersize=45, marker='^', color='blue',
                        label='Air quality monitoring site', ax=ax)

ax.axis('off')

legend = ax.legend(loc='best', shadow=True, fontsize=15)

plt.savefig('figure/Fig1.png', facecolor=None, dpi=500)

plt.show()
```



There are some sites that seem to be very close to each other.

In [ ]:

```
# add a column that specifies the shortest distance of a site to its nearest neighbour
loc_gdf['min_dis'] = pd.Series(dtype='float64')
for index, row in loc_gdf.iterrows():
    dis = []
    for i, v in loc_gdf['geometry'].iteritems():
        dis.append(row['geometry'].distance(v))
    dis.remove(0)
    loc_gdf.loc[index, 'min_dis'] = min(dis)
```

In [ ]:

```
# list sites that are close to each other (within 1.5km)
loc_gdf[loc_gdf['min_dis'] <= 1500]
```

Out[ ]:

	siteid	sitename	geometry	buffer_1km	Gsp	UnC	
1	BL0	Camden - Bloomsbury	POINT (530120.048 182038.807)	POLYGON ((531120.048 182038.807, 531115.233 18...	MULTIPOLYGON Z (((530046.600 181557.850 0.000,...	MULTILINESTRING Z ((530175.051 181041.510 0.00...	MULTILINES Z ((5299 181058.68(
2	CD9	Camden - Euston Road	POINT (529900.870 182666.124)	POLYGON ((530900.870 182666.124, 530896.055 18...	MULTIPOLYGON Z (((530164.744 181702.568 0.000,...	MULTILINESTRING Z ((529650.665 181699.144 0.00...	MULTILINES Z ((5299 181667.10:

	siteid	sitename	geometry	buffer_1km	Gsp	UnC	
7	GR4	Greenwich - Eltham	POINT (543978.694 174655.234)	POLYGON ((544978.694 174655.234, 544973.878 17...	MULTIPOLYGON Z (((544807.871 175213.894 0.000,...	MULTILINESTRING Z ((543437.000 173984.000 0.00...	MULTILINES Z ((5434 173917.89(
8	GB0	Greenwich - Falconwood FDMS	POINT (544997.933 175098.152)	POLYGON ((545997.933 175098.152, 545993.118 17...	MULTIPOLYGON Z (((544142.814 174582.038 0.000,...	MULTILINESTRING Z ((544952.089 174100.404 0.00...	MULTILINES Z ((5447 174454.12(

In [ ]:

```
# check their readings' descriptive statistics
dep_df[dep_df['Site'].isin(['BL0', 'CD9', 'GR4',
'GB0'])].groupby('Site').describe()
```

Out[ ]:

	Value							
	count	mean	std	min	25%	50%	75%	max
Site								
BL0	8760.0	10.908521	10.228363	0.1	4.7	7.6	12.800	92.4
CD9	8760.0	13.712563	10.330619	0.1	7.3	10.9	16.300	83.9
GB0	8760.0	12.569166	9.864263	0.1	6.8	9.4	14.325	79.8
GR4	8760.0	10.887037	9.764517	0.1	5.3	8.0	12.600	97.6

In [ ]:

```
# student's t test
stats.ttest_rel(dep_df[dep_df['Site']=='BL0'].Value.values,
                dep_df[dep_df['Site']=='CD9'].Value.values)
```

Out[ ]:

```
Ttest_relResult(statistic=-59.89747540590601, pvalue=0.0)
```

In [ ]:

```
stats.ttest_rel(dep_df[dep_df['Site']=='GR4'].Value.values,
                dep_df[dep_df['Site']=='GB0'].Value.values)
```

Out[ ]:

```
Ttest_relResult(statistic=-31.347923748114297, pvalue=1.5260626870045138e-2
04)
```

Both indicate that we should reject H0, meaning the two datasets are statistically significantly different.

In [ ]:

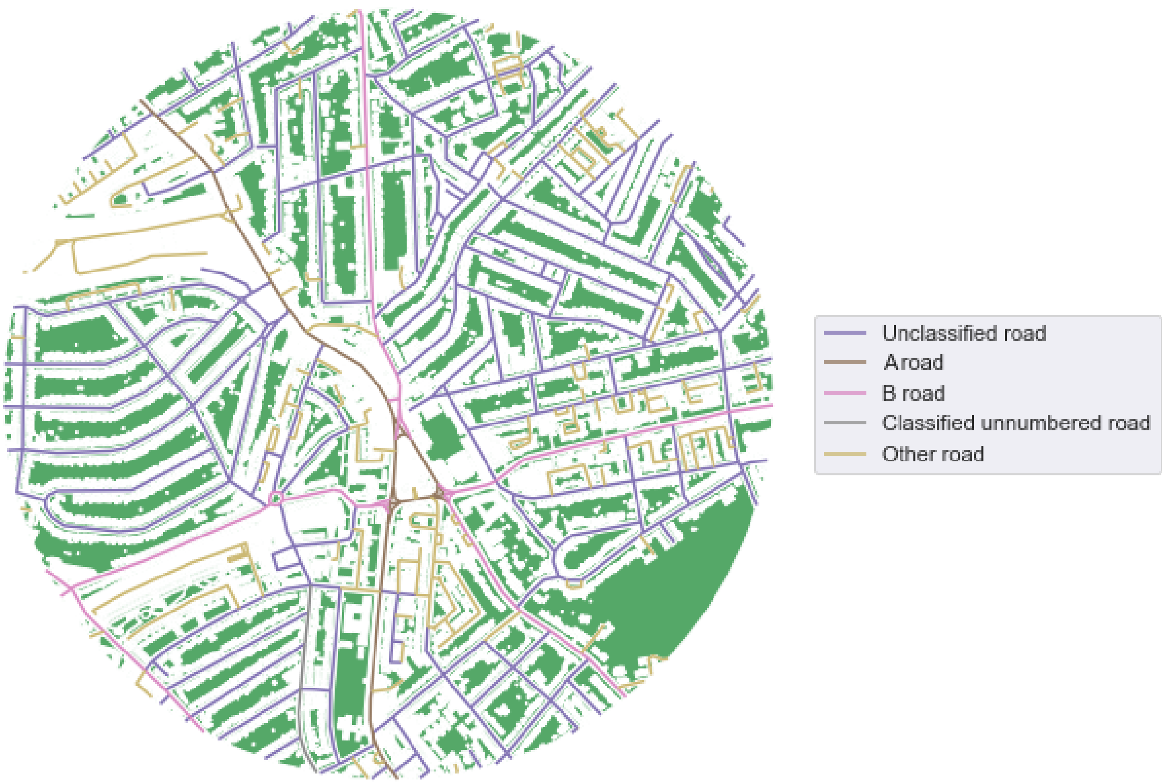
```
sns.color_palette()
```

Out[ ]:

In [ ]:

```
# Fig 4 - example of a site buffer (CD1 Camden-Swiss Cottage)
```

```
fig, ax = plt.subplots(1, figsize=(12,8))
loc_gdf.loc[[3],'buffer_1km'].plot(color='white', edgecolor=None,
ax=ax)
loc_gdf.loc[[3],'Gsp'].plot(color=sns.color_palette()[2],
edgecolor=None, ax=ax, label='Greenspace')
loc_gdf.loc[[3],'UnC'].plot(color=sns.color_palette()[4],
edgecolor=None, ax=ax, label='Unclassified road')
loc_gdf.loc[[3],'A'].plot(color=sns.color_palette()[5], edgecolor=None,
ax=ax, label='A road')
loc_gdf.loc[[3],'B'].plot(color=sns.color_palette()[6], edgecolor=None,
ax=ax, label='B road')
loc_gdf.loc[[3],'CUn'].plot(color=sns.color_palette()[7],
edgecolor=None, ax=ax, label='Classified unnumbered road')
loc_gdf.loc[[3],'Other'].plot(color=sns.color_palette()[8],
edgecolor=None, ax=ax, label='Other road')

plt.legend(bbox_to_anchor=(0.99,0.5), loc='center left')
ax.axis('off')

plt.savefig('figure/Fig4.png', facecolor=None, dpi=500)
plt.show()
```



In [ ]:

```
# get total areas of greenspaces
loc_gdf['Gsp_area'] = loc_gdf['Gsp'].area
```

In [ ]:

```
# get road lengths of each type and near-road greenspaces for each type
for col in Rd_type:
    loc_gdf[col+'_len'] = loc_gdf[col].length
    loc_gdf[col+'_area_per_len'] = loc_gdf['n'+col+'_Gsp'].area /
loc_gdf[col+'_len']
```

In [ ]:

```
loc_gdf.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 31 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   siteid               21 non-null     object
 1   sitename             21 non-null     object
 2   geometry             21 non-null     geometry
 3   buffer_1km           21 non-null     geometry
 4   Gsp                  21 non-null     geometry
 5   UnC                  21 non-null     geometry
 6   A                    21 non-null     geometry
 7   B                    21 non-null     geometry
 8   CUn                  21 non-null     geometry
 9   Mt                   21 non-null     geometry
 10  Other                21 non-null     geometry
 11  nUnC_Gsp             21 non-null     geometry
 12  nA_Gsp               21 non-null     geometry
 13  nB_Gsp               21 non-null     geometry
 14  nCUn_Gsp             21 non-null     geometry
 15  nMt_Gsp              21 non-null     geometry
 16  nOther_Gsp           21 non-null     geometry
 17  min_dis              21 non-null     float64
 18  Gsp_area             21 non-null     float64
 19  UnC_len              21 non-null     float64
 20  UnC_area_per_len     21 non-null     float64
 21  A_len                21 non-null     float64
 22  A_area_per_len       21 non-null     float64
 23  B_len                21 non-null     float64
 24  B_area_per_len       17 non-null     float64
 25  CUn_len              21 non-null     float64
 26  CUn_area_per_len     18 non-null     float64
 27  Mt_len               21 non-null     float64
 28  Mt_area_per_len      1 non-null      float64
 29  Other_len            21 non-null     float64
 30  Other_area_per_len   21 non-null     float64
dtypes: float64(14), geometry(15), object(2)
memory usage: 5.2+ KB
```

In [ ]:

```
exp_df = loc_gdf.loc[:,['siteid']+[col+'_area_per_len' for col in
Rd_type]].copy()
exp_df.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   siteid              21 non-null     object
 1   UnC_area_per_len    21 non-null     float64
 2   A_area_per_len      21 non-null     float64
 3   B_area_per_len      17 non-null     float64
 4   CUn_area_per_len    18 non-null     float64
 5   Mt_area_per_len     1 non-null      float64
 6   Other_area_per_len  21 non-null     float64
dtypes: float64(6), object(1)
memory usage: 1.3+ KB
```

There are many null values in `Mt_area_per_len` .

Because only one site has near motorway.

Remove the variable would be the best.

In [ ]:
```
exp_df.drop('Mt_area_per_len', axis=1, inplace=True)
loc_gdf.drop(['Mt_len', 'Mt_area_per_len'], axis=1, inplace=True)
Rd_type.remove('Mt')
```

Some null values in `B_area_per_len` and `CUn_area_per_len` , which is due to the lengths of B roads or Classified Unnumbered roads in these buffers are zero.

In [ ]:
```
# set the null values to zero
exp_df.fillna(0, inplace=True)
exp_df.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   siteid              21 non-null     object
 1   UnC_area_per_len    21 non-null     float64
 2   A_area_per_len      21 non-null     float64
 3   B_area_per_len      21 non-null     float64
 4   CUn_area_per_len    21 non-null     float64
 5   Other_area_per_len  21 non-null     float64
dtypes: float64(5), object(1)
memory usage: 1.1+ KB
```

In [ ]:
```
loc_gdf.fillna(0, inplace=True)
```

In [ ]:
```
exp_df.to_csv('exp_data.csv', index=False)
```

Data analysis

In [ ]:
```
exp_df = pd.read_csv('exp_data.csv')
exp_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 6 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   siteid               21 non-null     object
 1   UnC_area_per_len     21 non-null     float64
 2   A_area_per_len       21 non-null     float64
 3   B_area_per_len       21 non-null     float64
 4   CUn_area_per_len     21 non-null     float64
 5   Other_area_per_len   21 non-null     float64
dtypes: float64(5), object(1)
memory usage: 1.1+ KB
```

In [ ]:
```
dep_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183960 entries, 0 to 183959
Data columns (total 3 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   Site             183960 non-null  object
 1   ReadingDateTime  183960 non-null  object
 2   Value            183960 non-null  float64
dtypes: float64(1), object(2)
memory usage: 4.2+ MB
```

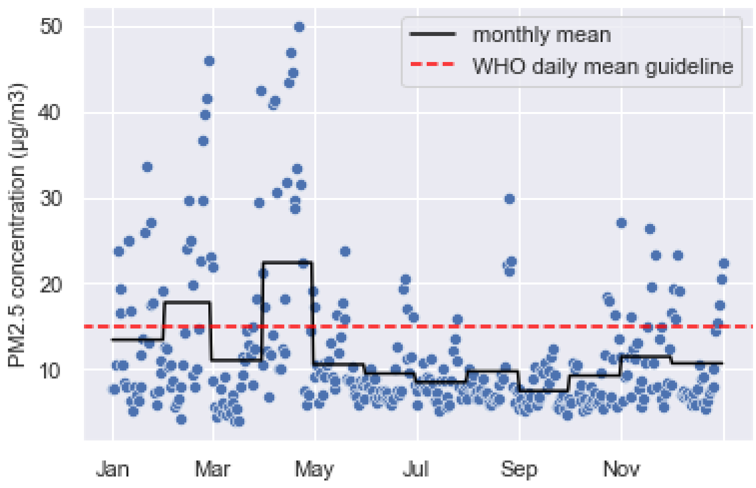
In [ ]:
```
# covert the DateTime column to numpy.datetime variable
dep_df['ReadingDateTime'] = pd.to_datetime(dep_df['ReadingDateTime'],
format="%d/%m/%Y %H:%M")
dep_df.rename(columns={'ReadingDateTime':'DateTime'}, inplace=True)
dep_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 183960 entries, 0 to 183959
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   Site      183960 non-null  object
 1   DateTime  183960 non-null  datetime64[ns]
 2   Value     183960 non-null  float64
dtypes: datetime64[ns](1), float64(1), object(1)
memory usage: 4.2+ MB
```

In [ ]:
```
dep_df['month'] = dep_df['DateTime'].dt.month
dep_df['hour'] = dep_df['DateTime'].dt.hour
dep_df['dayofmonth'] = dep_df['DateTime'].dt.day
dep_df['Date'] = dep_df['DateTime'].dt.date
```

In [ ]:

```
# Fig 2
mlabels =
['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']

sns.scatterplot(x=dep_df['Date'].unique(),
y=dep_df.groupby('Date').mean()['Value'])

plt.plot(dep_df['Date'].unique(),

dep_df.groupby('Date').mean().merge(dep_df.groupby('month').mean()
[['Value']], left_on='month', right_index=True)['Value_y'],
         color='black', label='monthly mean')

plt.axhline(y=15, color='red', linestyle='--', label='WHO daily mean
guideline')

plt.ylabel('PM2.5 concentration (μg/m3)', fontsize=11)
plt.gca().set_xticks(plt.gca().get_xticks())
plt.gca().set_xticklabels([mlabels[2*i] for i in range(6)]+[''])

plt.legend()

plt.savefig('figure/Fig2.png', facecolor=None, dpi=500)

plt.show()
```



In [ ]:

```
# number of date above WHO guideline
(dep_df.groupby('Date').mean()['Value']>15).sum()
```

Out[ ]: 74

In [ ]:

```
# annual mean for each site - table 1
```

```
dep_df.groupby('Site').mean()['Value']
```

Out[ ]:
```
Site
BL0    10.908521
BX9    11.170749
CD1    11.058464
CD9    13.712563
CR8    10.125421
CT2    13.902287
CT3    12.142583
GB0    12.569166
GN3    13.363480
GN6    11.039737
GR4    10.887037
GR9    10.482015
HP1     9.931490
HV1    11.368690
KC1     9.567551
LH0     9.412646
LW2    15.422345
MY7    14.190663
ST5    11.732403
TD5    11.686217
TH4    12.610338
Name: Value, dtype: float64
```

In [ ]:
```
# annual mean for London
dep_df['Value'].mean()
```

Out[ ]:
```
11.775446103783608
```

In [ ]:
```
# explanatory variable names to a list
var_names = exp_df.columns[1:].tolist()
var_names
```

Out[ ]:
```
['UnC_area_per_len',
 'A_area_per_len',
 'B_area_per_len',
 'CUn_area_per_len',
 'Other_area_per_len']
```

In [ ]:
```
# Gaussian kernel weights matrix
weight = Kernel.from_dataframe(loc_gdf, geom_col='geometry',
function='gaussian')
```

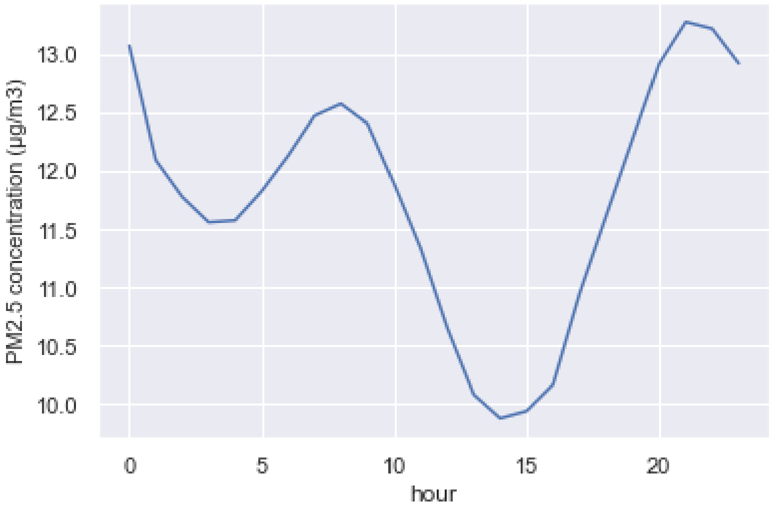
In [ ]:
```
# check global moran's I for the explanatory variables
for var in var_names:
    moran_temp = Moran(exp_df[var].values, weight)
    print("Global Moran's I for " + var + ' is ', round(moran_temp.I,
```

```
5),
          ' p-value: ', round(moran_temp.p_norm, 5))
```

```
Global Moran's I for UnC_area_per_len is  0.22651  p-value:  0.0
Global Moran's I for A_area_per_len is  0.1898  p-value:  4e-05
Global Moran's I for B_area_per_len is  0.02886  p-value:  0.17844
Global Moran's I for CUn_area_per_len is  0.15787  p-value:  0.00039
Global Moran's I for Other_area_per_len is  0.19556  p-value:  3e-05
```

In [ ]:

```
# Fig 3 - annual mean per hour
dep_df.groupby('hour').mean()['Value'].plot()
plt.ylabel('PM2.5 concentration (μg/m3)', fontsize=11)

plt.savefig('figure/Fig3.png', facecolor=None, dpi=500)

plt.show()
```



In [ ]:

```
# table 2
exp_df[var_names].describe()
```

Out[ ]:

	UnC_area_per_len	A_area_per_len	B_area_per_len	CUn_area_per_len	Other_area_per_len
count	21.000000	21.000000	21.000000	21.000000	21.00000C
mean	28.520807	26.246194	25.788827	34.007412	42.49722€
std	14.016818	16.163453	21.169168	22.542926	21.02656€
min	6.461089	7.084152	0.000000	0.000000	16.415021
25%	13.888893	12.595197	5.633153	17.717612	26.381062
50%	32.957256	21.628544	27.486036	33.960430	37.11636¢
75%	38.753776	35.940536	44.131224	50.730381	55.76674¢
max	47.129151	60.890307	56.162862	64.512992	84.75567¢

In [ ]:

```
# feature importance function
```

```
def get_importance(reg, features, target, feature_names, state=25,
rep=50, method='r2'):
    mean = []
    std = []
    reg.fit(features, target)
    importance = permutation_importance(reg, features, target,
n_repeats=rep,
                                        random_state=state,
scoring=method)
    for i in range(len(feature_names)):
        mean.append(round(importance.importances_mean[i], 5))
        std.append(round(importance.importances_std[i], 5))
    return mean, std
```

In [ ]:

```
# cross-validation function
def get_cv(reg, features, target, iter=100, n_splits=5, loo=False):
    cv_r2 = []
    cv_resid = []
    if loo:
        split = LeaveOneOut()
        iter = 1
    for i in range(iter):
        if not loo:
            split = KFold(n_splits=n_splits, shuffle=True,
random_state=i)
        cvprd = cross_val_predict(reg, features, target, cv=split)

        r = stats.pearsonr(target,cvprd)[0]
        resid = cvprd - target

        cv_r2.append(r**2)
        cv_resid.append(resid)

    return [round(np.mean(cv_r2),5), round(np.std(cv_r2),5),
            np.mean(np.array(cv_resid), axis=0)]
```

In [ ]:

```
# initialise linear model
reg = LinearRegression()
```

In [ ]:

```
# df for annual mean
annual = exp_df.merge(dep_df.groupby('Site').mean()[['Value']],
```

```
left_on='siteid', right_index=True)
annual.head()
```

Out[ ]:

	siteid	UnC_area_per_len	A_area_per_len	B_area_per_len	CUn_area_per_len	Other_area_per_
0	BX9	42.547777	34.722332	0.000000	63.364634	52.384
1	BL0	10.218919	9.464790	16.140991	0.000000	21.815
2	CD9	13.888893	12.595197	20.121072	0.000000	24.772
3	CD1	33.768627	16.790598	32.863003	49.419568	37.116
4	CT2	6.777993	7.084152	5.633153	30.535483	17.630

In [ ]:

```
# global moran's I for annual mean
Moran(annual['Value'].values, weight).I
```

Out[ ]:

```
0.0960805356530469
```

In [ ]:

```
# model variables
y = annual['Value'].values
X = annual[var_names].values
```

In [ ]:

```
# feature importance for annual mean model
get_importance(reg, X, y, var_names)
```

Out[ ]:

```
([1.63613, 0.19492, 0.49033, 0.61434, 0.2002],
 [0.55218, 0.10694, 0.22777, 0.24394, 0.13459])
```

In [ ]:

```
# coefficient
reg.coef_
```

Out[ ]:

```
array([-0.10841385,  0.03010942,  0.03779141,  0.03916031, -0.02523263])
```

In [ ]:

```
# r2
reg.score(X, y)
```

Out[ ]:

```
0.365064847871562
```

In [ ]:

```
# cross validation r2 and std
get_cv(reg, X, y, loo=True)
```

Out[ ]:

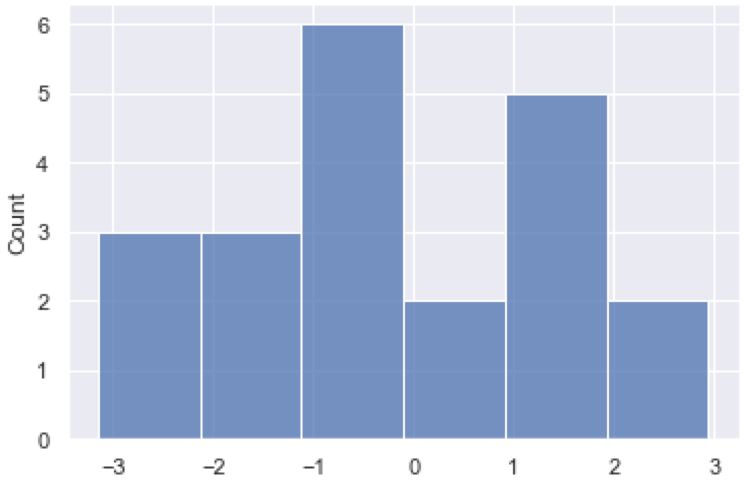
```
[0.08144,
 0.0,
 array([-1.16477859,  1.43446761, -2.28106682,  1.23100786, -0.85619942,
         1.47089549,  1.79574319,  1.87708538, -0.40099349, -0.28459951,
        -1.54635138, -1.44411554, -0.1239867 ,  0.84468749,  2.9402903 ,
         2.36256793, -3.14251197, -0.76127703,  0.03531615, -0.48762403,
        -2.40246668])]
```

In [ ]:

```
# residuals histogram
sns.histplot(get_cv(reg, X, y, loo=True)[2])
```

Out[ ]:

```
<AxesSubplot:ylabel='Count'>
```



In [ ]:

```
# global moran's I for the residuals
Moran(get_cv(reg, X, y, loo=True)[2], weight).I, Moran(get_cv(reg, X,
y, loo=True)[2], weight).p_norm
```

Out[ ]:

```
(0.035801811859351725, 0.14319436427836196)
```

In [ ]:

```
# df for annual mean per hour
hm_dep_df = dep_df.groupby(['hour', 'Site']).mean()
hm_dep_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 504 entries, (0, 'BL0') to (23, 'TH4')
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Value        504 non-null    float64
 1   month        504 non-null    float64
 2   dayofmonth   504 non-null    float64
dtypes: float64(3)
memory usage: 13.3+ KB
```

In [ ]:

```
# drop unnecessary columns
hm_dep_df.drop(['dayofmonth', 'month'], axis=1, inplace=True)

# reset index
hm_dep_df.reset_index(inplace=True)

# add explanatory variables to the df
hm_dep_df = hm_dep_df.merge(exp_df, left_on='Site', right_on='siteid')
hm_dep_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 504 entries, 0 to 503
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   hour                504 non-null    int64
 1   Site                504 non-null    object
 2   Value               504 non-null    float64
 3   siteid              504 non-null    object
 4   UnC_area_per_len    504 non-null    float64
 5   A_area_per_len      504 non-null    float64
 6   B_area_per_len      504 non-null    float64
 7   CUn_area_per_len    504 non-null    float64
 8   Other_area_per_len  504 non-null    float64
dtypes: float64(6), int64(1), object(2)
memory usage: 39.4+ KB
```

In [ ]:

```
# drop repetitive column
hm_dep_df.drop('Site', axis=1, inplace=True)
```

In [ ]:

```
# check global moran's I for the 24 groups of annual means per hour
for h in range(24):
    df = hm_dep_df[hm_dep_df['hour']==h].copy()
    print("Global Moran's I for hour ", h, " is: ",
Moran(df['Value'].values, weight).I)
```

```
Global Moran's I for hour  0  is:  0.06497149669113031
Global Moran's I for hour  1  is:  0.0245776965350352
Global Moran's I for hour  2  is:  0.023679068042262625
Global Moran's I for hour  3  is:  0.022407841453422502
Global Moran's I for hour  4  is:  0.010815062841263887
Global Moran's I for hour  5  is:  0.015475583335916723
Global Moran's I for hour  6  is:  0.022795098109670307
Global Moran's I for hour  7  is:  0.03812792090918269
Global Moran's I for hour  8  is:  0.061859833103432335
Global Moran's I for hour  9  is:  0.06864587291469511
Global Moran's I for hour  10  is:  0.07152289068767917
Global Moran's I for hour  11  is:  0.05695064983154796
Global Moran's I for hour  12  is:  0.044951223755330394
Global Moran's I for hour  13  is:  0.036635569274802986
Global Moran's I for hour  14  is:  0.018729390685179557
Global Moran's I for hour  15  is:  0.001011166962265878
Global Moran's I for hour  16  is:  -0.0071881322024358145
Global Moran's I for hour  17  is:  -0.008536266595833628
Global Moran's I for hour  18  is:  0.014484832240914635
Global Moran's I for hour  19  is:  0.0011745643188229072
Global Moran's I for hour  20  is:  0.01483448920752272
Global Moran's I for hour  21  is:  0.00858989331841518
Global Moran's I for hour  22  is:  0.0030212534070487417
Global Moran's I for hour  23  is:  0.012845802764135312
```

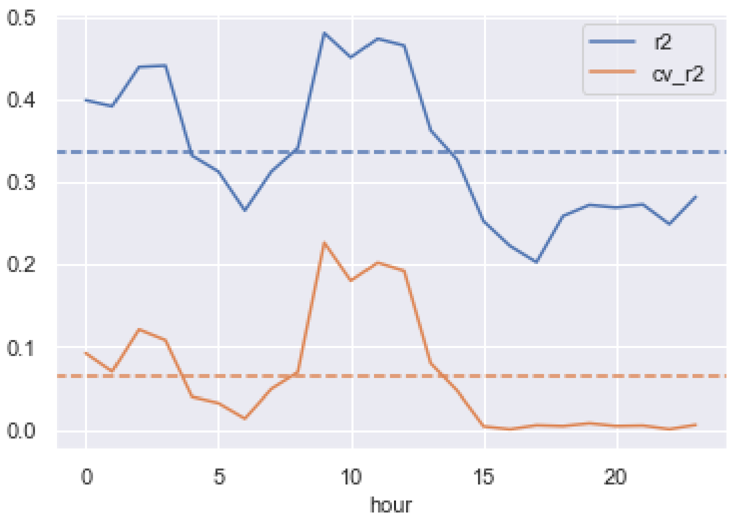
In [ ]:

```
# linear models by each hour
hm_reg = []
```

```
for h in range(24):
    df = hm_dep_df[hm_dep_df['hour']==h].copy()

    X = df[var_names].values
    y = df['Value'].values

    mean, std = get_importance(reg, X, y, var_names)
    coef = reg.coef_.tolist() + [reg.intercept_]
    r2 = reg.score(X, y)
    cv = get_cv(reg, X, y, loo=True)

    hm_reg.append(mean+std+coef+[r2]+cv)

hm_reg = pd.DataFrame(hm_reg, columns=['fi_'+var for var in var_names]+
                      ['fi_std_'+var for var in var_names]+var_names+
                      ['intercept','r2','cv_r2','std_r2','resid'])
```

In [ ]:

```
# Fig 5 - hourly model performance
hm_reg[['r2', 'cv_r2']].plot()
plt.axhline(y=hm_reg['r2'].mean(),linestyle='--')
plt.axhline(y=hm_reg['cv_r2'].mean(), linestyle='--
',color=sns.color_palette()[1])
plt.xlabel('hour', fontsize=11)

plt.savefig('figure/Fig5.png', facecolor=None, dpi=500)

plt.show()
```



In [ ]:

```
# histogram for residuals
fig, ax = plt.subplots(4, 6, figsize=(24, 16))
```

```
for hour in range(24):
    sns.histplot(hm_reg.loc[hour,'resid'], ax=ax[hour//6, hour%6])
    ax[hour//6, hour%6].get_xaxis().set_ticks([])
    ax[hour//6, hour%6].set_title(str(hour)+'-'+str(hour+1))

plt.show()
```

In [ ]:

```
for h in range(24):
    resid = hm_reg.loc[h, 'resid']
    print("Global Moran's I for residuals for hour ", h, " is: ",
          Moran(resid, weight).I, " p-value: ", Moran(resid,
weight).p_norm)
```

```
Global Moran's I for residuals for hour  0  is:  0.08526918977846121  p-val
ue:  0.020996993438289646
Global Moran's I for residuals for hour  1  is:  0.017680790831707732  p-va
lue:  0.24816927037495295
Global Moran's I for residuals for hour  2  is:  0.024961330542104306  p-va
lue:  0.20088574325269315
Global Moran's I for residuals for hour  3  is:  0.031908795653333044  p-va
lue:  0.16224113298007747
Global Moran's I for residuals for hour  4  is:  -0.009099531308112231  p-v
alue:  0.4852605860352657
Global Moran's I for residuals for hour  5  is:  -0.006047326898875781  p-v
alue:  0.45328778340140774
Global Moran's I for residuals for hour  6  is:  0.003905077552398862б  p-v
alue:  0.35769893977484557
Global Moran's I for residuals for hour  7  is:  0.010235774865333181  p-va
lue:  0.30405499039356254
```

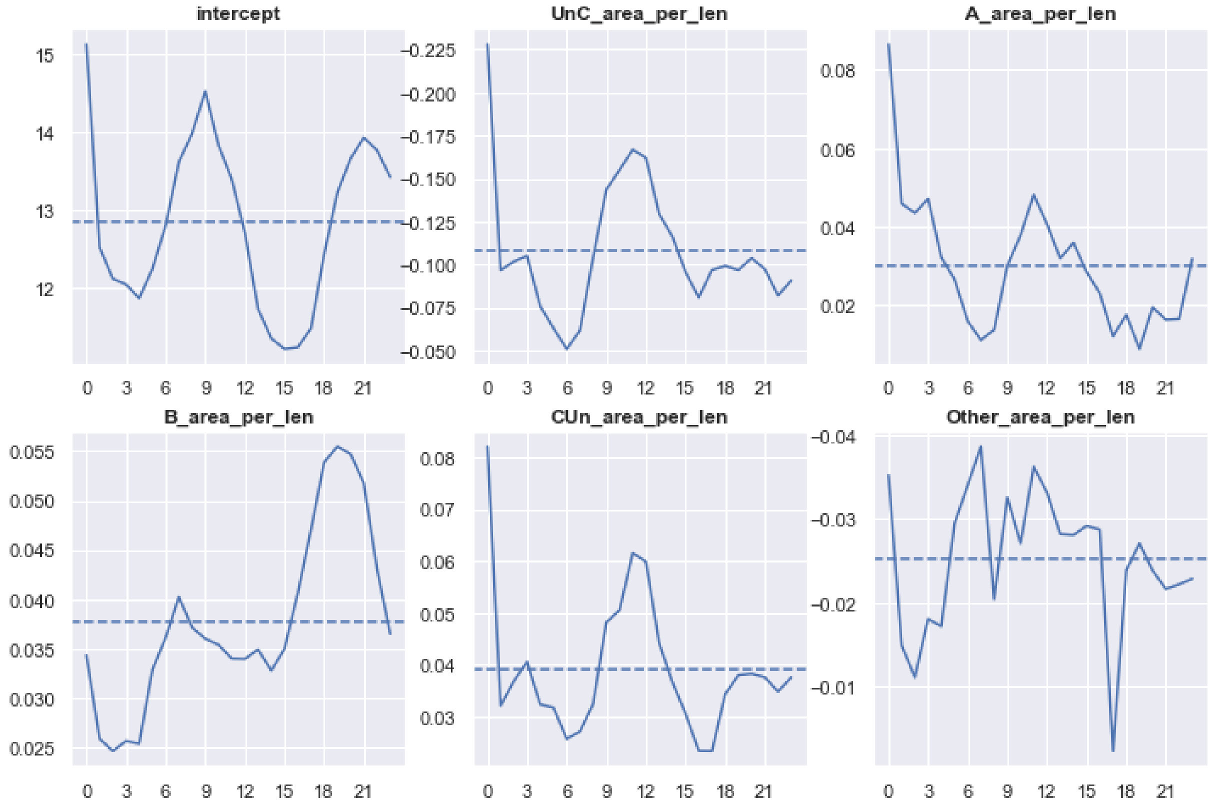
```
Global Moran's I for residuals for hour  8  is:  0.035738418628806425  p-va
lue:  0.14349013902280094
Global Moran's I for residuals for hour  9  is:  0.04407216664248342  p-val
ue:  0.10846918217997881
Global Moran's I for residuals for hour  10  is:  0.05232887893251602  p-va
lue:  0.08081224902883033
Global Moran's I for residuals for hour  11  is:  0.029987947416533163  p-v
alue:  0.17231621037651967
Global Moran's I for residuals for hour  12  is:  0.007196098097735054  p-v
alue:  0.3291081746829687
Global Moran's I for residuals for hour  13  is:  -0.003195079844593924  p-
value:  0.42451559438142517
Global Moran's I for residuals for hour  14  is:  -0.020109263501667963  p-
value:  0.610042735873185
Global Moran's I for residuals for hour  15  is:  -0.031780851791542865  p-
value:  0.7559030315253836
Global Moran's I for residuals for hour  16  is:  -0.03319672746223665 4  p-
value:  0.7743368080970914
Global Moran's I for residuals for hour  17  is:  -0.023254069966454134  p-
value:  0.648135431764304
Global Moran's I for residuals for hour  18  is:  -0.007510291876657277  p-
value:  0.4684623664198557
Global Moran's I for residuals for hour  19  is:  -0.023575108066383316  p-
value:  0.6520787216244954
Global Moran's I for residuals for hour  20  is:  -0.006808895224931003  p-
value:  0.46115228556854815
Global Moran's I for residuals for hour  21  is:  -0.012431268660981638  p-
value:  0.5215105365580477
Global Moran's I for residuals for hour  22  is:  -0.016189530904717885  p-
value:  0.5640116558393389
Global Moran's I for residuals for hour  23  is:  -0.004729874894723929  p-
value:  0.43986323670592986
```

In [ ]:

```
# Fig 6 - hourly feature importance
fig, ax = plt.subplots(4, 6, figsize=(24, 16))

for hour in range(24):
    ax[hour//6, hour%6].bar(['fi_' + elem for elem in var_names],
                            hm_reg.loc[hour, ['fi_' + elem for elem in
var_names]].values,
                            yerr=hm_reg.loc[hour, ['fi_std_' + elem for
elem in var_names]].values)
    ax[hour//6, hour%6].get_xaxis().set_ticks([])
    ax[hour//6, hour%6].set_title(str(hour)+'-'+str(hour+1),
fontweight='bold')

plt.savefig('figure/Fig6.png', facecolor=None, dpi=500)

plt.show()
```

In [ ]:

```
# Fig 7 - coefficient viz
fig,ax=plt.subplots(2,3,figsize=(12,8))


col = ['intercept']+var_names
for i in range(len(col)):
    hm_reg[col[i]].plot(ax=ax[i//3,i%3])
    ax[i//3,i%3].set_title(col[i], fontweight='bold')
    ax[i//3,i%3].set_xticks([3*i for i in range(8)])
    ax[i//3,i%3].axhline(y=hm_reg[col[i]].mean(), linestyle='--')
    if hm_reg[col[i]].mean()<0:
        ax[i//3,i%3].invert_yaxis()


plt.savefig('figure/Fig7.png', facecolor=None, dpi=500)


plt.show()
```



In [ ]:

```
# df for monthly mean
mm_dep_df = dep_df.groupby(['month','Site']).mean()
mm_dep_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 252 entries, (1, 'BL0') to (12, 'TH4')
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Value       252 non-null    float64
 1   hour        252 non-null    float64
 2   dayofmonth  252 non-null    float64
dtypes: float64(3)
memory usage: 6.8+ KB
```

In [ ]:

```
# drop unnecessary columns
mm_dep_df.drop(['hour', 'dayofmonth'], axis=1, inplace=True)


# reset index
mm_dep_df.reset_index(inplace=True)


# add explanatory variables
mm_dep_df = mm_dep_df.merge(exp_df, left_on='Site', right_on='siteid')
mm_dep_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 252 entries, 0 to 251
Data columns (total 9 columns):
 #   Column                Non-Null Count  Dtype
```

```
---  ------                --------------  -----
 0   month                 252 non-null    int64
 1   Site                  252 non-null    object
 2   Value                 252 non-null    float64
 3   siteid                252 non-null    object
 4   UnC_area_per_len      252 non-null    float64
 5   A_area_per_len        252 non-null    float64
 6   B_area_per_len        252 non-null    float64
 7   CUn_area_per_len      252 non-null    float64
 8   Other_area_per_len    252 non-null    float64
dtypes: float64(6), int64(1), object(2)
memory usage: 19.7+ KB
```

In [ ]:

```
# drop repetitive column
mm_dep_df.drop('Site', axis=1, inplace=True)
```

In [ ]:

```
# check global moran's I for the 12 groups of monthly means
for m in range(1,13):
    df = mm_dep_df[mm_dep_df['month']==m].copy()
    print("Global Moran's I for ", mlabels[m-1], "is: ",
Moran(df['Value'].values, weight).I)
```

```
Global Moran's I for  Jan is:  0.003641632057737503
Global Moran's I for  Feb is:  0.0662609644851644
Global Moran's I for  Mar is:  0.13033251481287086
Global Moran's I for  Apr is:  -0.007821811529515265
Global Moran's I for  May is:  0.057814827084503126
Global Moran's I for  Jun is:  0.07317506194877035
Global Moran's I for  Jul is:  0.016624172348547708
Global Moran's I for  Aug is:  0.015710564641922425
Global Moran's I for  Sep is:  0.05979959100051212
Global Moran's I for  Oct is:  0.08902715229331153
Global Moran's I for  Nov is:  0.05306934244532968
Global Moran's I for  Dec is:  0.10564724933858408
```

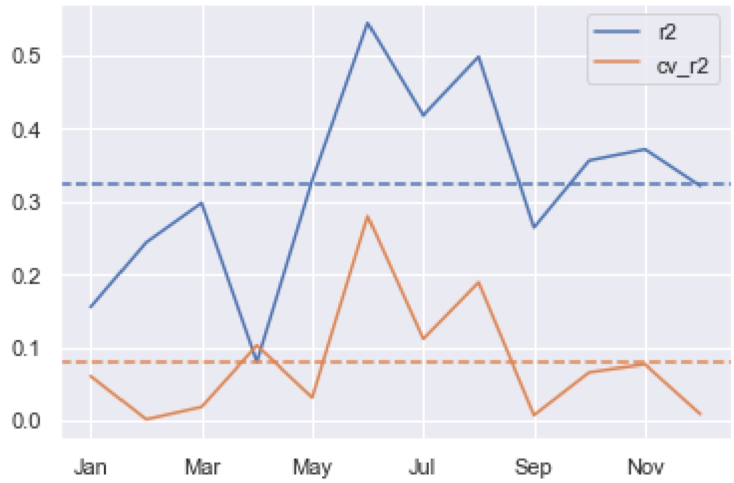
In [ ]:

```
# linear models by each month
mm_reg = []
for m in range(1,13):
    df = mm_dep_df[mm_dep_df['month']==m].copy()

    X = df[var_names].values
    y = df['Value'].values

    mean, std = get_importance(reg, X, y, var_names)
    coef = reg.coef_.tolist() + [reg.intercept_]
    r2 = reg.score(X, y)
    cv = get_cv(reg, X, y, loo=True)

    mm_reg.append(mean+std+coef+[r2]+cv)
```

```
mm_reg = pd.DataFrame(mm_reg, columns=['fi_'+var for var in var_names]+
                      ['fi_std_'+var for var in var_names]+var_names+
                      ['intercept','r2','cv_r2','std_r2','resid'])
```

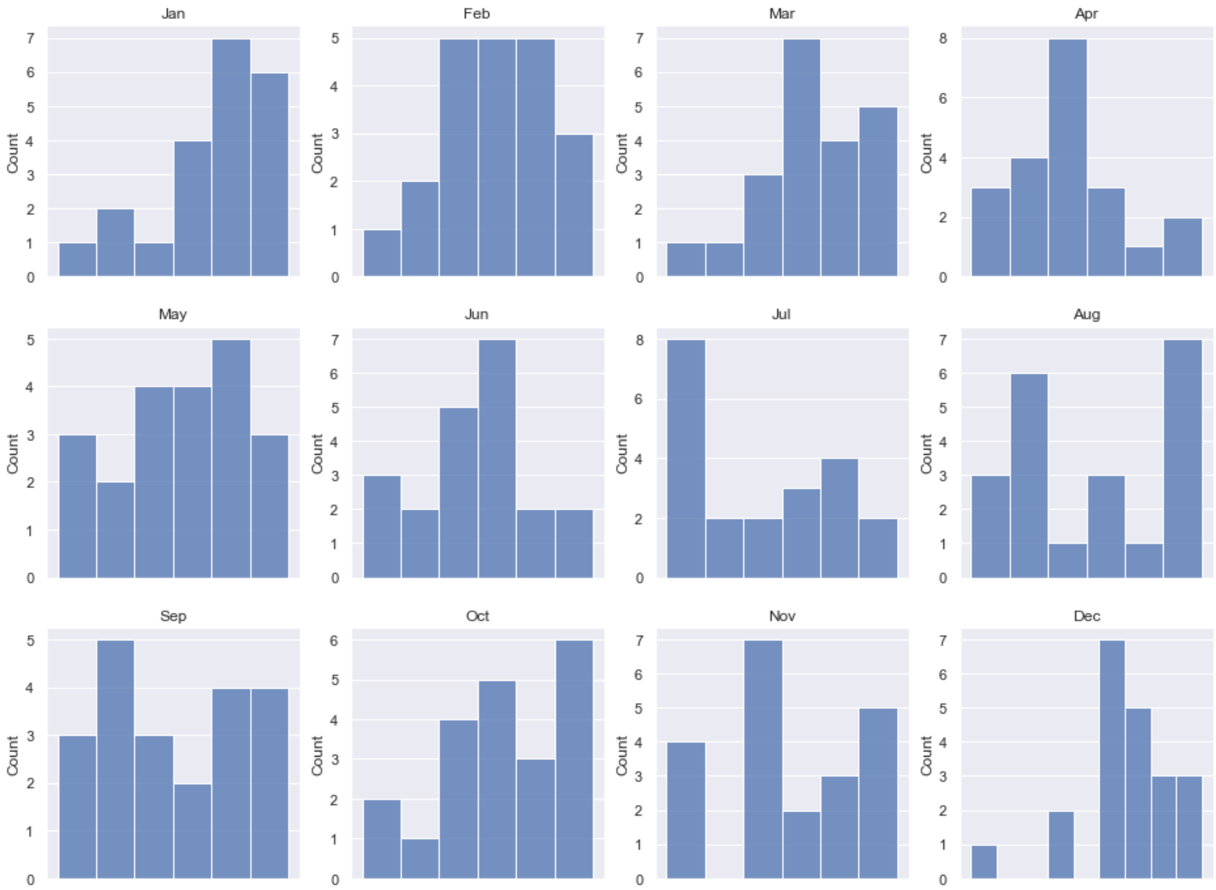
In [ ]:

```
# Fig 8 - model performance
mm_reg[['r2', 'cv_r2']].plot()

plt.axhline(y=mm_reg['r2'].mean(),linestyle='--')
plt.axhline(y=mm_reg['cv_r2'].mean(), linestyle='--
',color=sns.color_palette()[1])

plt.gca().set_xticks([0,2,4,6,8,10])
plt.gca().set_xticklabels([mlabels[2*i] for i in range(6)])

plt.savefig('figure/Fig8.png', facecolor=None, dpi=500)

plt.show()
```



In [ ]:

```
# histogram for residuals
fig, ax = plt.subplots(3, 4, figsize=(16, 12))

for month in range(12):
    sns.histplot(mm_reg.loc[month,'resid'], ax=ax[month//4, month%4])
    ax[month//4, month%4].get_xaxis().set_ticks([])
    ax[month//4, month%4].set_title(mlabels[month])

plt.show()
```



In [ ]:

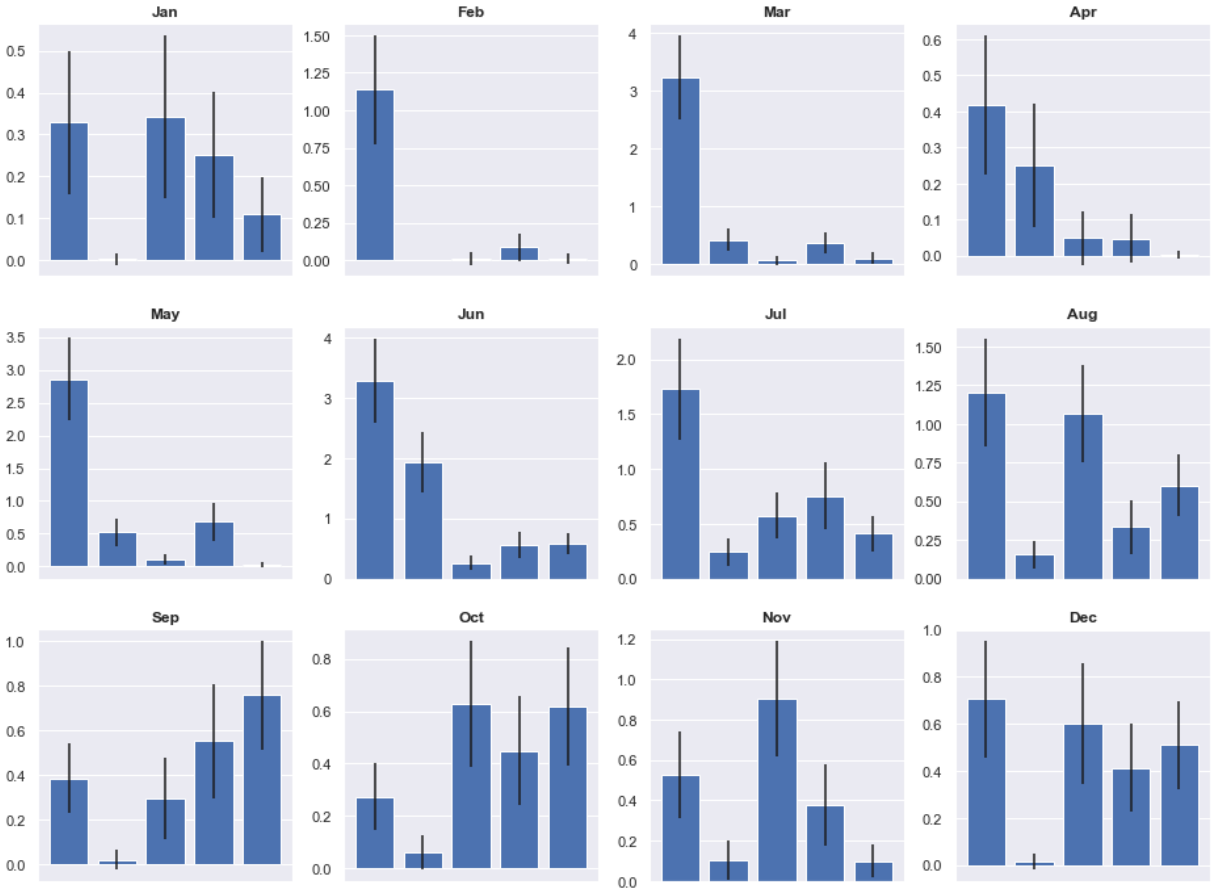
```
for m in range(12):
    resid = mm_reg.loc[m, 'resid']
    print("Global Moran's I for residuals for ", mlabels[m], " is: ",
Moran(resid, weight).I,
          " p-value: ", Moran(resid, weight).p_norm)
```

```
Global Moran's I for residuals for  Jan  is:  -0.0008782486805907636  p-val
ue:  0.4019505186489587
Global Moran's I for residuals for  Feb  is:  0.06609368032560278  p-value:
0.04760750641374334
Global Moran's I for residuals for  Mar  is:  0.0827955347494292  p-value:
0.023461692908614662
Global Moran's I for residuals for  Apr  is:  0.014224891199051569  p-valu
e:  0.27314836431004474
Global Moran's I for residuals for  May  is:  0.014277374642435135  p-valu
e:  0.27275659685651577
Global Moran's I for residuals for  Jun  is:  0.08420680063133579  p-value:
0.022026450070490977
Global Moran's I for residuals for  Jul  is:  0.022846624416685758  p-valu
e:  0.21388632256854834
Global Moran's I for residuals for  Aug  is:  0.018585202020258  p-value:
0.24190477950327383
Global Moran's I for residuals for  Sep  is:  0.017548951329757163  p-valu
e:  0.24909186377633974
Global Moran's I for residuals for  Oct  is:  0.029116238322654784  p-valu
e:  0.17703988607960386
Global Moran's I for residuals for  Nov  is:  0.02780130619328  p-value:
0.18434685142863128
```

```
Global Moran's I for residuals for  Dec  is:  0.11965867076691691  p-value:
0.0037938678648252733
```

In [ ]:

```
# Fig 9 - plot feature importance
fig, ax = plt.subplots(3, 4, figsize=(16, 12))

for month in range(12):
    ax[month//4, month%4].bar(['fi_' + elem for elem in var_names],
                              mm_reg.loc[month, ['fi_' + elem for elem
in var_names]].values,
                              yerr=mm_reg.loc[month, ['fi_std_' + elem
for elem in var_names]].values)
    ax[month//4, month%4].get_xaxis().set_ticks([])
    ax[month//4, month%4].set_title(mlabels[month], fontweight='bold')

plt.savefig('figure/Fig9.png', facecolor=None, dpi=500)

plt.show()
```



In [ ]:

```
# Fig 10 - coefficient viz
fig,ax=plt.subplots(2,3,figsize=(12,8))

for i in range(len(col)):
    mm_reg[col[i]].plot(ax=ax[i//3,i%3])
```

```
    ax[i//3,i%3].set_title(col[i], fontweight='bold')
    ax[i//3,i%3].set_xticks([0,2,4,6,8,10])
    ax[i//3,i%3].set_xticklabels([mlabels[2*i] for i in range(6)])
    ax[i//3,i%3].axhline(y=mm_reg[col[i]].mean(), linestyle='--')
    if mm_reg[col[i]].max()<0:
        ax[i//3,i%3].invert_yaxis()

plt.savefig('figure/Fig10.png', facecolor=None, dpi=500)

plt.show()
```