In [117…

```
# iphython options
# delete variables in workspace
%reset -f
#places plots inline
%matplotlib inline
#automatically reloads modules if they are changed
%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

Watershed Statistics

Adrian Wiegman | arhwiegman.github.io | adrian.wiegman@usda.gov

Created: 2023-04-05 Edited: 2023-11-06

Buildnotes:

- Created function for running shapely additive

MEP Report Data

The objective of this notebook is to evaulate statistical relationships between watershed attributes and water quality monitoring data for the streams monitored by the Massachusetts Estuaries Program (MEP) Reports .

The MEP reports cover 109 watersheds and 96 stream monitoring locations within the MA southcoast, Cape Cod and the Islands of Martha's Vinyard and Nantucket. Each stream was monitored for at least one year. During monitoring, stage was records with a pressure tranducer, low tide stage discharge relationships were developed, and water samples were collected at least weekly and analyzed for NO3-N and TN. Stream monitoring occured between 1999 and 2007, with over half of monitoring occuring between 2003 and 2006.

The MEP reports compared measured vs modeled discharge and N loads. Modeled discharge estimates based on watershed area and long term recharge rate. Nitrogen loads were modeled using assumptions about atmospheric deposition and attentuation rates for landcover classes, fertilizer use, the number of housing units, residential water use, wastewater treatment plants, and landfills.

The table below describes the variables in the MEP report summary dataset (`data/MEP_SummaryData_Coords.csv`).

variable	units	description
Qmod	m3/d	modeled discharge (cubic meters per day) using recharge rate and watershed area
Qmeas	m3/d	measured discharge based on low tide stage data and stage-discharge relationships
Qdiff	%	percent difference between Q measured and Q modeled $100 * (Q_{mod} - Q_{meas})/(Q_{meas})$
NOx	mg N/L	mean nitrate + nitrite concentration in samples collected stream water from over the monitoring period
TN	mg N/L	mean total nitrogen concentration in samples collected stream water from over the monitoring period
Atten	%	estimated nitrogen attenuation based $100 * (N_{mod} - N_{meas})/(N_{meas})$
NOx2TN	ratio	NOX divided by TN
Yr_Start	yyyy	year of monitoring start
Yr_End	yyyy	year of monitoring end
Lat	decimal degrees	lattitude WGS84 of monitoring location
Lon	decimal degrees	longitude WGS84 of monitoring location
Region	character	Region of MEP report
MEP	character	name of MEP report on Mass.gov
SiteName	character	Name of stream monitoring site in MEP report
Region_MEP	character	unqiue id string: Region > MEP > SiteName

MEP Watershed Attributes

The watershed boundaries for the MEP datasets for Martha's Vinyard and Cape Cod were obtained from the capecod commision and from Ed Eichner. Watershed attributes are summarized as percent cover for various landcover and soils datasets.within contributing areas to a given monitoring location. Watershed attributes concatenated from the datasource and class value (e.g. `HYDROLGRP_A` is % cover of Hydrologic Soil Group A within contributing subwatersheds). Watershed attributes are described in detail in the file: `Preprocess_Attributes.ipynb`.

- numeric columns with no prefix summarize all portions of the contributing subwatersheds
- numeric columns with prefix `GT5` reflect land cover in areas above the 5th percentile elevation in contributing subwatersheds
- numeric columns with prefix `LE5` summarize land cover in areas at or below the 5th percentile elevation in constributing subs.

Statistical Analysis

I will use machine learning methods available `scikit-learn 1.2.2` to predict the MEP variables (`NOX` , `TN` , `NOX2TN` , `Atten` , `Qmeas` , `Qdiff`) from watershed attributes. My hypotheses are as follows.

1. NOX will increase as (A) measures of human development increase, and (B) as natural areas capable of attenuating NOX decrease.
2. Atten will decrease with declining natural cover
3. NOX2TN will decrease with increasing natural cover

I will apply the following machine learning techniques and evaluate performance using shuffle and split k fold cross validation with root mean squared error as the peformance metric:

1. multiple linear regression with regularization to prevent overfitting

   A. lasso

   B. elastic net

2. random forest regression

3. gradient boosting regression

Bivariate with NOx

COVERNAME_developed open space exponential - positive COVERNAME impervious - exponential positive USE residential single family exponential - positive USE right of way expoential - positive USE tax exempt - negative inverse USE ResComMix - positive SLOPE_D inverse GT5_Slope*D inverse Depth to watertable 0 inverse negative Farmland of unique importance inverse negative Not prime farmland logistic/exponential positive Nleaching high logistic/exponential positive LE5* hydrol group A negative

References

Gu, Q., Hu, H., Ma, L., Sheng, L., Yang, S., Zhang, X., Zhang, M., Zheng, K., & Chen, L. (2019). Characterizing the spatial variations of the relationship between land use and surface water quality using self-organizing map approach. Ecological Indicators, 102(March), 633–643. https://doi.org/10.1016/j.ecolind.2019.03.017

Lee, C. M., Choi, H., Kim, Y., Kim, M. S., Kim, H. K., & Hamm, S. Y. (2021). Characterizing land use effect on shallow groundwater contamination by using self-organizing map and buffer zone. Science of the Total Environment, 800(3), 149632. https://doi.org/10.1016/j.scitotenv.2021.149632

Feng, Z., Xu, C., Zuo, Y., Luo, X., Wang, L., Chen, H., Xie, X., Yan, D., & Liang, T. (2023). Analysis of water quality indexes and their relationships with vegetation using self-organizing map and geographically and temporally weighted regression. Environmental Research, 216(P2), 114587. https://doi.org/10.1016/j.envres.2022.114587

@article{scikit-learn, title={Scikit-learn: Machine Learning in {P}ython}, author={Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V. and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P. and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E.}, journal={Journal of Machine Learning Research}, volume={12}, pages={2825--2830}, year={2011} }

In [118…

```
# this codeblock sets up the environment from jupyter notebooks
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re, os, glob, sys, errno
wdr = os.getcwd()
print("working directory:",wdr)
idr = os.path.join(wdr,'data')
print("input directory:",idr)
odr = os.path.join(wdr,'outputs')
print("output directory:",odr)
scripts = os.path.join(wdr,'scripts')
print("source codes:",scripts)
sys.path.insert(0,scripts) # insert path to directory containing source codes
from functions import *
print("\ntype `fn`+tab to look up user defined functions, \nrun `??fn_{name}` to inspe
fn_hello('Adrian')
print("\nenvironment setup complete")
```

```
working directory: C:\Users\Adrian.Wiegman\Documents\GitHub\Wiegman_USDA_ARS\Cran_Q_C
\_superceded\MEP
input directory: C:\Users\Adrian.Wiegman\Documents\GitHub\Wiegman_USDA_ARS\Cran_Q_C\_
superceded\MEP\data
output directory: C:\Users\Adrian.Wiegman\Documents\GitHub\Wiegman_USDA_ARS\Cran_Q_C
\_superceded\MEP\outputs
source codes: C:\Users\Adrian.Wiegman\Documents\GitHub\Wiegman_USDA_ARS\Cran_Q_C\_sup
erceded\MEP\scripts

type `fn`+tab to look up user defined functions,
run `??fn_{name}` to inspect function source code

hello Adrian

environment setup complete
```

In [119…

```
# read the subwatershed attributes table
filename = "df_monitoring_point_sub_attributes_terminus.csv"
_ = pd.read_csv(os.path.join(idr,filename))
_ = _.rename(columns=lambda x: re.sub('\.0','',x))
_ = _.rename(columns=lambda x: re.sub('\s','_',x))
_ = _.rename(columns=lambda x: re.sub('-','_',x))
_ = _.rename(columns=lambda x: re.sub(',','_',x))
```

```
#_ = _[_.columns.drop(list(_.filter(regex='DEP2')))]
#_ = _[_.columns.drop(list(_.filter(regex='SLOPE_pct')))]
_.replace([np.inf, -np.inf], np.nan, inplace=True)
_.fillna(0,inplace=True)
print(_.info())
for c in _.columns:
    print(c)
display(_.head())
df_point_sub_atts = _
del _ # clear temporary object from memory

# read the subwatershed monitoring data table
filename = "MEP_SummaryData_Coords.csv"
_ = pd.read_csv(os.path.join(idr,filename))
_.info()

#print()
_ = _[~_['Region_MEP'].isna()]# remove rows with >50% Null
#print(_.shape)
_.dropna(axis=1, thresh = int(0.5*_.shape[0]), inplace=True) # remove columns with >50
#print(_.info())
_.replace([np.inf, -np.inf], np.nan, inplace=True)
#_.fillna(0,inplace=True)
df_monitoring = _
del _ # clear temporary object from memory
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Columns: 592 entries, FID to LE5_Use_Mix
dtypes: float64(590), int64(1), object(1)
memory usage: 462.6+ KB
None
FID
Region_MEP
Lat
Lon
Shape_Area
SLOPE_
SLOPE_0
SLOPE_A
SLOPE_B
SLOPE_C
SLOPE_D
SLOPE_E
FRMLNDCLS_
FRMLNDCLS_All_areas_are_prime_farmland
FRMLNDCLS_Farmland_of_statewide_importance
FRMLNDCLS_Farmland_of_unique_importance
FRMLNDCLS_Not_prime_farmland
HYDROLGRP_
HYDROLGRP_A
HYDROLGRP_A/D
HYDROLGRP_B
HYDROLGRP_B/D
HYDROLGRP_C
HYDROLGRP_C/D
HYDROLGRP_D
HYDRCRATNG_
HYDRCRATNG_No
HYDRCRATNG_Unranked
HYDRCRATNG_Yes
DRAINCLASS_
DRAINCLASS_Excessively_drained
DRAINCLASS_Moderately_well_drained
DRAINCLASS_Poorly_drained
DRAINCLASS_Somewhat_excessively_drained
DRAINCLASS_Somewhat_poorly_drained
DRAINCLASS_Subaqueous
DRAINCLASS_Very_poorly_drained
DRAINCLASS_Well_drained
DEP2WATTBL_0
DEP2WATTBL_5
DEP2WATTBL_8
DEP2WATTBL_10
DEP2WATTBL_15
DEP2WATTBL_23
DEP2WATTBL_29
DEP2WATTBL_30
DEP2WATTBL_38
DEP2WATTBL_45
DEP2WATTBL_46
DEP2WATTBL_48
DEP2WATTBL_51
DEP2WATTBL_54
DEP2WATTBL_60
DEP2WATTBL_61
```

DEP2WATTBL_64
DEP2WATTBL_66
DEP2WATTBL_68
DEP2WATTBL_69
DEP2WATTBL_73
DEP2WATTBL_76
DEP2WATTBL_77
DEP2WATTBL_82
CLAY_0
CLAY_0.1
CLAY_0.2
CLAY_0.5
CLAY_1
CLAY_1.3
CLAY_1.5
CLAY_2
CLAY_2.5
CLAY_3
CLAY_3.3000000000000003
CLAY_4
CLAY_4.5
CLAY_4.6000000000000005
CLAY_4.7
CLAY_5
CLAY_5.4
CLAY_6
CLAY_6.5
CLAY_7
CLAY_7.2
CLAY_7.3
CLAY_7.5
CLAY_8
CLAY_9
CLAY_9.200000000000001
CLAY_9.5
CLAY_10
CLAY_10.6
CLAY_11.5
CLAY_13
CLAY_15
CLAY_19
CLAY_20.2
CLAY_25
CLAY_27
CLAY_27.5
CLAY_30
OM_0
OM_05
OM_08
OM_0.1
OM_0.25
OM_0.3
OM_0.4
OM_0.7000000000000001
OM_0.75
OM_1
OM_1.5
OM_1.74
OM_2
OM_2.2

OM_2.3000000000000003
OM_2.5
OM_3
OM_3.35
OM_3.4
OM_3.5
OM_3.8000000000000003
OM_3.92
OM_4
OM_4.12
OM_4.13
OM_4.5
OM_4.97
OM_5
OM_59
OM_5.5
OM_5.75
OM_6
OM_6.4
OM_6.5
OM_6.7
OM_6.9
OM_7
OM_71
OM_7.75
OM_8
OM_8.3
OM_8.34
OM_9.5
OM_10
OM_11
OM_12
OM_14
OM_25
NLEACHING_
NLEACHING_High
NLEACHING_Low
NLEACHING_Moderate
NLEACHING_Not_rated
COVERNAME_
COVERNAME_Bare_Land
COVERNAME_Cultivated
COVERNAME_Deciduous_Forest
COVERNAME_Developed_Open_Space
COVERNAME_Estaurine_Aquatic_bed
COVERNAME_Estuarine_Emergent_Wetland
COVERNAME_Estuarine_Forested_Wetland
COVERNAME_Estuarine_Scrub/Shrub_Wetland
COVERNAME_Evergreen_Forest
COVERNAME_Grassland
COVERNAME_Impervious
COVERNAME_Palustrine_Aquatic_Bed
COVERNAME_Palustrine_Emergent_Wetland
COVERNAME_Palustrine_Forested_Wetland
COVERNAME_Palustrine_Scrub/Shrub_Wetland
COVERNAME_Pasture/Hay
COVERNAME_Scrub/Shrub
COVERNAME_Unconsolidated_Shore
COVERNAME_Water
USEGENNAME_

```
USEGENNAME_Agriculture
USEGENNAME_Commercial
USEGENNAME_Forest
USEGENNAME_Industrial
USEGENNAME_Mixed_use__other
USEGENNAME_Mixed_use__primarily_commercial
USEGENNAME_Mixed_use__primarily_residential
USEGENNAME_Open_land
USEGENNAME_Recreation
USEGENNAME_Residential___multi_family
USEGENNAME_Residential___other
USEGENNAME_Residential___single_family
USEGENNAME_Right_of_way
USEGENNAME_Tax_exempt
USEGENNAME_Unknown
USEGENNAME_Water
CRANBERRY_0
CRANBERRY_1
ACTIVE_0
ACTIVE_1
HSG_D
HYDRIC
NaturalCover
Use_Res
Use_ResComMix
Use_Mix
GT5_Shape_Area
GT5_SLOPE_
GT5_SLOPE_0
GT5_SLOPE_A
GT5_SLOPE_B
GT5_SLOPE_C
GT5_SLOPE_D
GT5_SLOPE_E
GT5_FRMLNDCLS_
GT5_FRMLNDCLS_All_areas_are_prime_farmland
GT5_FRMLNDCLS_Farmland_of_statewide_importance
GT5_FRMLNDCLS_Farmland_of_unique_importance
GT5_FRMLNDCLS_Not_prime_farmland
GT5_HYDROLGRP_
GT5_HYDROLGRP_A
GT5_HYDROLGRP_A/D
GT5_HYDROLGRP_B
GT5_HYDROLGRP_B/D
GT5_HYDROLGRP_C
GT5_HYDROLGRP_C/D
GT5_HYDROLGRP_D
GT5_HYDRCRATNG_
GT5_HYDRCRATNG_No
GT5_HYDRCRATNG_Unranked
GT5_HYDRCRATNG_Yes
GT5_DRAINCLASS_
GT5_DRAINCLASS_Excessively_drained
GT5_DRAINCLASS_Moderately_well_drained
GT5_DRAINCLASS_Poorly_drained
GT5_DRAINCLASS_Somewhat_excessively_drained
GT5_DRAINCLASS_Somewhat_poorly_drained
GT5_DRAINCLASS_Subaqueous
GT5_DRAINCLASS_Very_poorly_drained
GT5_DRAINCLASS_Well_drained
```

GT5_DEP2WATTBL_0
GT5_DEP2WATTBL_5
GT5_DEP2WATTBL_8
GT5_DEP2WATTBL_10
GT5_DEP2WATTBL_15
GT5_DEP2WATTBL_23
GT5_DEP2WATTBL_29
GT5_DEP2WATTBL_30
GT5_DEP2WATTBL_38
GT5_DEP2WATTBL_45
GT5_DEP2WATTBL_46
GT5_DEP2WATTBL_48
GT5_DEP2WATTBL_51
GT5_DEP2WATTBL_54
GT5_DEP2WATTBL_60
GT5_DEP2WATTBL_61
GT5_DEP2WATTBL_64
GT5_DEP2WATTBL_66
GT5_DEP2WATTBL_68
GT5_DEP2WATTBL_69
GT5_DEP2WATTBL_73
GT5_DEP2WATTBL_76
GT5_DEP2WATTBL_77
GT5_DEP2WATTBL_82
GT5_CLAY_0
GT5_CLAY_0.1
GT5_CLAY_0.2
GT5_CLAY_0.5
GT5_CLAY_1
GT5_CLAY_1.3
GT5_CLAY_1.5
GT5_CLAY_2
GT5_CLAY_2.5
GT5_CLAY_3
GT5_CLAY_3.3000000000000003
GT5_CLAY_4
GT5_CLAY_4.5
GT5_CLAY_4.6000000000000005
GT5_CLAY_4.7
GT5_CLAY_5
GT5_CLAY_5.4
GT5_CLAY_6
GT5_CLAY_6.5
GT5_CLAY_7
GT5_CLAY_7.2
GT5_CLAY_7.3
GT5_CLAY_7.5
GT5_CLAY_8
GT5_CLAY_9
GT5_CLAY_9.200000000000001
GT5_CLAY_9.5
GT5_CLAY_10
GT5_CLAY_10.6
GT5_CLAY_11.5
GT5_CLAY_13
GT5_CLAY_15
GT5_CLAY_19
GT5_CLAY_20.2
GT5_CLAY_25
GT5_CLAY_27

GT5_CLAY_27.5
GT5_CLAY_30
GT5_OM_0
GT5_OM_05
GT5_OM_08
GT5_OM_0.1
GT5_OM_0.25
GT5_OM_0.3
GT5_OM_0.4
GT5_OM_0.7000000000000001
GT5_OM_0.75
GT5_OM_1
GT5_OM_1.5
GT5_OM_1.74
GT5_OM_2
GT5_OM_2.2
GT5_OM_2.3000000000000003
GT5_OM_2.5
GT5_OM_3
GT5_OM_3.35
GT5_OM_3.4
GT5_OM_3.5
GT5_OM_3.8000000000000003
GT5_OM_3.92
GT5_OM_4
GT5_OM_4.12
GT5_OM_4.13
GT5_OM_4.5
GT5_OM_4.97
GT5_OM_5
GT5_OM_59
GT5_OM_5.5
GT5_OM_5.75
GT5_OM_6
GT5_OM_6.4
GT5_OM_6.5
GT5_OM_6.7
GT5_OM_6.9
GT5_OM_7
GT5_OM_71
GT5_OM_7.75
GT5_OM_8
GT5_OM_8.3
GT5_OM_8.34
GT5_OM_9.5
GT5_OM_10
GT5_OM_11
GT5_OM_12
GT5_OM_14
GT5_OM_25
GT5_NLEACHING_
GT5_NLEACHING_High
GT5_NLEACHING_Low
GT5_NLEACHING_Moderate
GT5_NLEACHING_Not_rated
GT5_COVERNAME_
GT5_COVERNAME_Bare_Land
GT5_COVERNAME_Cultivated
GT5_COVERNAME_Deciduous_Forest
GT5_COVERNAME_Developed_Open_Space

GT5_COVERNAME_Estaurine_Aquatic_bed
GT5_COVERNAME_Estuarine_Emergent_Wetland
GT5_COVERNAME_Estuarine_Forested_Wetland
GT5_COVERNAME_Estuarine_Scrub/Shrub_Wetland
GT5_COVERNAME_Evergreen_Forest
GT5_COVERNAME_Grassland
GT5_COVERNAME_Impervious
GT5_COVERNAME_Palustrine_Aquatic_Bed
GT5_COVERNAME_Palustrine_Emergent_Wetland
GT5_COVERNAME_Palustrine_Forested_Wetland
GT5_COVERNAME_Palustrine_Scrub/Shrub_Wetland
GT5_COVERNAME_Pasture/Hay
GT5_COVERNAME_Scrub/Shrub
GT5_COVERNAME_Unconsolidated_Shore
GT5_COVERNAME_Water
GT5_USEGENNAME_
GT5_USEGENNAME_Agriculture
GT5_USEGENNAME_Commercial
GT5_USEGENNAME_Forest
GT5_USEGENNAME_Industrial
GT5_USEGENNAME_Mixed_use__other
GT5_USEGENNAME_Mixed_use__primarily_commercial
GT5_USEGENNAME_Mixed_use__primarily_residential
GT5_USEGENNAME_Open_land
GT5_USEGENNAME_Recreation
GT5_USEGENNAME_Residential___multi_family
GT5_USEGENNAME_Residential___other
GT5_USEGENNAME_Residential___single_family
GT5_USEGENNAME_Right_of_way
GT5_USEGENNAME_Tax_exempt
GT5_USEGENNAME_Unknown
GT5_USEGENNAME_Water
GT5_CRANBERRY_0
GT5_CRANBERRY_1
GT5_ACTIVE_0
GT5_ACTIVE_1
GT5_HSG_D
GT5_HYDRIC
GT5_NaturalCover
GT5_Use_Res
GT5_Use_ResComMix
GT5_Use_Mix
LE5_Shape_Area
LE5_SLOPE_
LE5_SLOPE_0
LE5_SLOPE_A
LE5_SLOPE_B
LE5_SLOPE_C
LE5_SLOPE_D
LE5_SLOPE_E
LE5_FRMLNDCLS_
LE5_FRMLNDCLS_All_areas_are_prime_farmland
LE5_FRMLNDCLS_Farmland_of_statewide_importance
LE5_FRMLNDCLS_Farmland_of_unique_importance
LE5_FRMLNDCLS_Not_prime_farmland
LE5_HYDROLGRP_
LE5_HYDROLGRP_A
LE5_HYDROLGRP_A/D
LE5_HYDROLGRP_B
LE5_HYDROLGRP_B/D

LE5_HYDROLGRP_C
LE5_HYDROLGRP_C/D
LE5_HYDROLGRP_D
LE5_HYDRCRATNG_
LE5_HYDRCRATNG_No
LE5_HYDRCRATNG_Unranked
LE5_HYDRCRATNG_Yes
LE5_DRAINCLASS_
LE5_DRAINCLASS_Excessively_drained
LE5_DRAINCLASS_Moderately_well_drained
LE5_DRAINCLASS_Poorly_drained
LE5_DRAINCLASS_Somewhat_excessively_drained
LE5_DRAINCLASS_Somewhat_poorly_drained
LE5_DRAINCLASS_Subaqueous
LE5_DRAINCLASS_Very_poorly_drained
LE5_DRAINCLASS_Well_drained
LE5_DEP2WATTBL_0
LE5_DEP2WATTBL_5
LE5_DEP2WATTBL_8
LE5_DEP2WATTBL_10
LE5_DEP2WATTBL_15
LE5_DEP2WATTBL_23
LE5_DEP2WATTBL_29
LE5_DEP2WATTBL_30
LE5_DEP2WATTBL_38
LE5_DEP2WATTBL_45
LE5_DEP2WATTBL_46
LE5_DEP2WATTBL_48
LE5_DEP2WATTBL_51
LE5_DEP2WATTBL_54
LE5_DEP2WATTBL_60
LE5_DEP2WATTBL_61
LE5_DEP2WATTBL_64
LE5_DEP2WATTBL_66
LE5_DEP2WATTBL_68
LE5_DEP2WATTBL_69
LE5_DEP2WATTBL_73
LE5_DEP2WATTBL_76
LE5_DEP2WATTBL_77
LE5_DEP2WATTBL_82
LE5_CLAY_0
LE5_CLAY_0.1
LE5_CLAY_0.2
LE5_CLAY_0.5
LE5_CLAY_1
LE5_CLAY_1.3
LE5_CLAY_1.5
LE5_CLAY_2
LE5_CLAY_2.5
LE5_CLAY_3
LE5_CLAY_3.3000000000000003
LE5_CLAY_4
LE5_CLAY_4.5
LE5_CLAY_4.6000000000000005
LE5_CLAY_4.7
LE5_CLAY_5
LE5_CLAY_5.4
LE5_CLAY_6
LE5_CLAY_6.5
LE5_CLAY_7

LE5_CLAY_7.2
LE5_CLAY_7.3
LE5_CLAY_7.5
LE5_CLAY_8
LE5_CLAY_9
LE5_CLAY_9.200000000000001
LE5_CLAY_9.5
LE5_CLAY_10
LE5_CLAY_10.6
LE5_CLAY_11.5
LE5_CLAY_13
LE5_CLAY_15
LE5_CLAY_19
LE5_CLAY_20.2
LE5_CLAY_25
LE5_CLAY_27
LE5_CLAY_27.5
LE5_CLAY_30
LE5_OM_0
LE5_OM_05
LE5_OM_08
LE5_OM_0.1
LE5_OM_0.25
LE5_OM_0.3
LE5_OM_0.4
LE5_OM_0.7000000000000001
LE5_OM_0.75
LE5_OM_1
LE5_OM_1.5
LE5_OM_1.74
LE5_OM_2
LE5_OM_2.2
LE5_OM_2.3000000000000003
LE5_OM_2.5
LE5_OM_3
LE5_OM_3.35
LE5_OM_3.4
LE5_OM_3.5
LE5_OM_3.8000000000000003
LE5_OM_3.92
LE5_OM_4
LE5_OM_4.12
LE5_OM_4.13
LE5_OM_4.5
LE5_OM_4.97
LE5_OM_5
LE5_OM_59
LE5_OM_5.5
LE5_OM_5.75
LE5_OM_6
LE5_OM_6.4
LE5_OM_6.5
LE5_OM_6.7
LE5_OM_6.9
LE5_OM_7
LE5_OM_71
LE5_OM_7.75
LE5_OM_8
LE5_OM_8.3
LE5_OM_8.34

```
LE5_OM_9.5
LE5_OM_10
LE5_OM_11
LE5_OM_12
LE5_OM_14
LE5_OM_25
LE5_NLEACHING_
LE5_NLEACHING_High
LE5_NLEACHING_Low
LE5_NLEACHING_Moderate
LE5_NLEACHING_Not_rated
LE5_COVERNAME_
LE5_COVERNAME_Bare_Land
LE5_COVERNAME_Cultivated
LE5_COVERNAME_Deciduous_Forest
LE5_COVERNAME_Developed_Open_Space
LE5_COVERNAME_Estaurine_Aquatic_bed
LE5_COVERNAME_Estuarine_Emergent_Wetland
LE5_COVERNAME_Estuarine_Forested_Wetland
LE5_COVERNAME_Estuarine_Scrub/Shrub_Wetland
LE5_COVERNAME_Evergreen_Forest
LE5_COVERNAME_Grassland
LE5_COVERNAME_Impervious
LE5_COVERNAME_Palustrine_Aquatic_Bed
LE5_COVERNAME_Palustrine_Emergent_Wetland
LE5_COVERNAME_Palustrine_Forested_Wetland
LE5_COVERNAME_Palustrine_Scrub/Shrub_Wetland
LE5_COVERNAME_Pasture/Hay
LE5_COVERNAME_Scrub/Shrub
LE5_COVERNAME_Unconsolidated_Shore
LE5_COVERNAME_Water
LE5_USEGENNAME_
LE5_USEGENNAME_Agriculture
LE5_USEGENNAME_Commercial
LE5_USEGENNAME_Forest
LE5_USEGENNAME_Industrial
LE5_USEGENNAME_Mixed_use__other
LE5_USEGENNAME_Mixed_use__primarily_commercial
LE5_USEGENNAME_Mixed_use__primarily_residential
LE5_USEGENNAME_Open_land
LE5_USEGENNAME_Recreation
LE5_USEGENNAME_Residential___multi_family
LE5_USEGENNAME_Residential___other
LE5_USEGENNAME_Residential___single_family
LE5_USEGENNAME_Right_of_way
LE5_USEGENNAME_Tax_exempt
LE5_USEGENNAME_Unknown
LE5_USEGENNAME_Water
LE5_CRANBERRY_0
LE5_CRANBERRY_1
LE5_ACTIVE_0
LE5_ACTIVE_1
LE5_HSG_D
LE5_HYDRIC
LE5_NaturalCover
LE5_Use_Res
LE5_Use_ResComMix
LE5_Use_Mix
```

	FID	Region_MEP	Lat	Lon	Shape_Area	SLOPE_	SLOPE_0	SLOPE_A	SLOPE_B
0	1	Buzzards Bay > Acushnet > Acushnet River	41.681859	-70.918844	4.510600e+07	0.000000	7.346414	40.006569	41.155998
1	2	Buzzards Bay > Acushnet > Acushnet River	41.681859	-70.918844	4.510600e+07	0.000000	7.346414	40.006569	41.155998
2	3	Buzzards Bay > Acushnet > Acushnet River	41.681859	-70.918844	4.510600e+07	0.000000	7.346414	40.006569	41.155998
3	4	Buzzards Bay > Acushnet > Acushnet River	41.681859	-70.918844	4.510600e+07	0.000000	7.346414	40.006569	41.155998
4	5	Buzzards Bay > Westport > Adamsville Brook	41.553741	-71.126612	1.489267e+07	51.319207	0.620756	21.203183	25.675601

5 rows × 592 columns

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109 entries, 0 to 108
Data columns (total 21 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   idn          109 non-null    int64  
 1   Region_MEP   109 non-null    object 
 2   Region       109 non-null    object 
 3   MEP          109 non-null    object 
 4   SiteName     99 non-null     object 
 5   Lat          98 non-null     object 
 6   Lon          98 non-null     float64
 7   Yr_Start     96 non-null     float64
 8   Yr_End       96 non-null     float64
 9   Dates        96 non-null     object 
 10  Pond_Atten   92 non-null     object 
 11  Qmeas        96 non-null     float64
 12  Qmod         95 non-null     float64
 13  Qdiff        96 non-null     float64
 14  NOx          96 non-null     float64
 15  NH4          16 non-null     object 
 16  TN           96 non-null     float64
 17  NOx2TN       96 non-null     float64
 18  Atten        94 non-null     float64
 19  Notes        23 non-null     object 
 20  Unnamed: 20  2 non-null      float64
dtypes: float64(11), int64(1), object(9)
```

In [120…

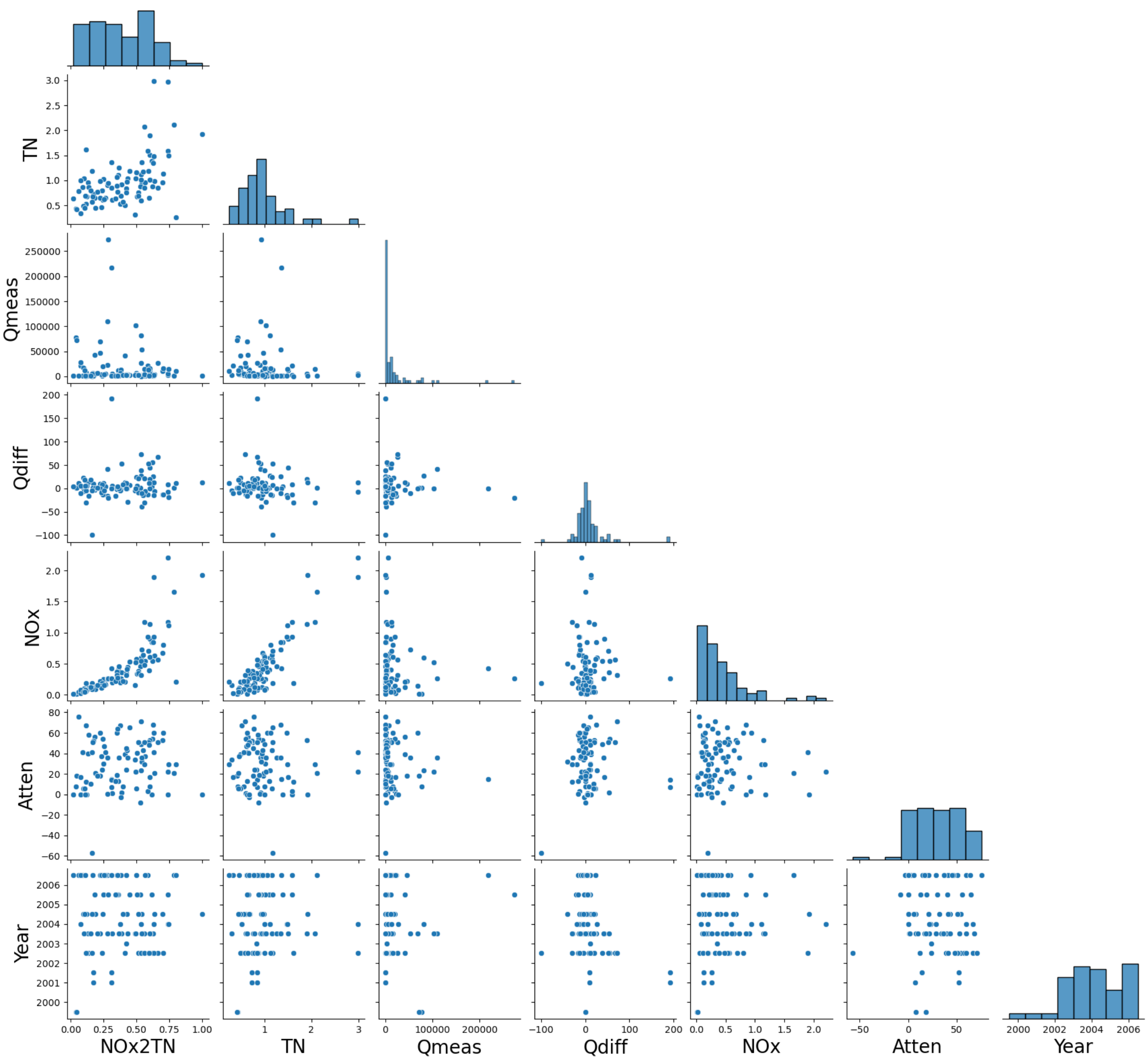
```
# clean up the monitoring data.
#print(df_monitoring.replace(to_replace=r'//', value='', regex=True))
#df_monitoring.rename(columns={'Pond/Stream N Atten. Meas. (%)':'Atten (%)'},inplace=T
```

```
#df_monitoring['Atten (%)'] = pd.to_numeric(df_monitoring['Atten (%)'])
#print(df_monitoring.info())

# add weights of observations based on year of monitoring
df_monitoring['n'] = 1
_ = df_monitoring[["Region_MEP","n"]].dropna().groupby('Region_MEP').aggregate(sum)
df_monitoring.merge(_,on="Region_MEP")
df_monitoring['wt'] = 1/df_monitoring['n']  # make observation wieghts based on monito
df_monitoring.head()
# select columns to retain for analysis
selected_monitoring_cols = ['NOx2TN','TN','Qmeas','Qdiff','NOx','Atten']
grps = ['Region_MEP']

# calculate mean value for all monitoring years at a given station
df_monitoring_avg = df_monitoring.groupby(grps)[selected_monitoring_cols].aggregate(np
df_monitoring_avg['Yr_Start'] = df_monitoring.groupby(grps)['Yr_Start'].aggregate(np.m
df_monitoring_avg['Yr_End'] = df_monitoring.groupby(grps)['Yr_End'].aggregate(np.max)
df_monitoring_avg = df_monitoring_avg.assign(Year = (df_monitoring_avg.Yr_Start+df_mon
df_monitoring_avg.info()
df_monitoring_avg.to_csv(os.path.join(odr,"MEP_SummaryData_Coords_Avg.csv"))
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 106 entries, Buzzards Bay > Acushnet > Acushnet River to Islands > Tisbury  >
Tiasquam River
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   NOx2TN    93 non-null     float64
 1   TN        93 non-null     float64
 2   Qmeas     93 non-null     float64
 3   Qdiff     93 non-null     float64
 4   NOx       93 non-null     float64
 5   Atten     91 non-null     float64
 6   Yr_Start  93 non-null     float64
 7   Yr_End    93 non-null     float64
 8   Year      93 non-null     float64
dtypes: float64(9)
memory usage: 8.3+ KB
```

In [121…

```
with sns.plotting_context(rc={"axes.labelsize":20}):
    sns.pairplot(df_monitoring_avg.drop(['Yr_Start','Yr_End'],axis=1),
             corner=True) # show only lower triangle
```



In [122…

```
#join the data
df_monitoring_watershed = df_monitoring_avg.merge(df_point_sub_atts.drop('FID',axis=1)
df = df_monitoring_watershed # make alias
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df.dropna(0,inplace=True)
#df.set_index(['Region_MEP'],inplace=True)
df.info()
df.to_csv(os.path.join(odr,'df_MEP_Monitoring_Avg_Geo_Merge.csv'),index=False)
print(df.columns)
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 90 entries, 0 to 107
Columns: 600 entries, Region_MEP to LE5_Use_Mix
dtypes: float64(599), object(1)
memory usage: 422.6+ KB
Index(['Region_MEP', 'NOx2TN', 'TN', 'Qmeas', 'Qdiff', 'NOx', 'Atten',
       'Yr_Start', 'Yr_End', 'Year',
       ...
       'LE5_CRANBERRY_0', 'LE5_CRANBERRY_1', 'LE5_ACTIVE_0', 'LE5_ACTIVE_1',
       'LE5_HSG_D', 'LE5_HYDRIC', 'LE5_NaturalCover', 'LE5_Use_Res',
       'LE5_Use_ResComMix', 'LE5_Use_Mix'],
      dtype='object', length=600)
```

In [123…

```
# univariate analysis
df_summary = df.describe().T
df_summary.to_csv(os.path.join(odr,'df_summary.csv'))
df_summary
```

Out[123]:

	count	mean	std	min	25%	50%	
NOx2TN	90.0	0.403491	0.214913	0.020376	0.226936	0.403984	0
TN	90.0	0.992978	0.483595	0.258000	0.677750	0.912500	1
Qmeas	90.0	18673.775297	33856.510248	97.000000	1403.500000	4203.000000	15982
Qdiff	90.0	8.995811	35.947742	-100.000000	-5.070710	3.208602	12
NOx	90.0	0.455427	0.429909	0.013000	0.152500	0.344000	0
...	...	...	...	...	...	...	
LE5_HYDRIC	90.0	61.018698	28.942819	0.000000	36.627118	69.906452	83
LE5_NaturalCover	90.0	83.241338	23.866110	0.822193	82.359013	92.822236	98
LE5_Use_Res	90.0	24.685466	28.281314	0.000000	3.265415	12.557533	37
LE5_Use_ResComMix	90.0	28.640612	28.566186	0.000000	4.273500	17.130482	46
LE5_Use_Mix	90.0	2.048978	5.595689	0.000000	0.000000	0.000000	0

599 rows × 8 columns

In [124…

```
for i in df.index:
    print(i)
    print("index: ",i,"  MEP id: ",df.Region_MEP[i])
```

```
0
index:  0   MEP id:  Buzzards Bay > Acushnet > Acushnet River
1
index:  1   MEP id:  Buzzards Bay > Acushnet > Acushnet River
2
index:  2   MEP id:  Buzzards Bay > Acushnet > Acushnet River
3
index:  3   MEP id:  Buzzards Bay > Acushnet > Acushnet River
4
index:  4   MEP id:  Buzzards Bay > Acushnet > Acushnet River
5
index:  5   MEP id:  Buzzards Bay > Nasketucket > Nasketucket River 1
6
index:  6   MEP id:  Buzzards Bay > Nasketucket > Nasketucket River 2
7
index:  7   MEP id:  Buzzards Bay > Nasketucket > Nonquit Brook
8
index:  8   MEP id:  Buzzards Bay > Nasketucket > Shaws Cove Stream
9
index:  9   MEP id:  Buzzards Bay > Slocums > Barneys Joy Creek
10
index:  10   MEP id:  Buzzards Bay > Slocums > Destruction Brook
11
index:  11   MEP id:  Buzzards Bay > Slocums > Giles Creek
12
index:  12   MEP id:  Buzzards Bay > Slocums > Paskamansett River
13
index:  13   MEP id:  Buzzards Bay > Wareham > Agawam River
14
index:  14   MEP id:  Buzzards Bay > Wareham > Wankinco River
15
index:  15   MEP id:  Buzzards Bay > Westport > Adamsville Brook
16
index:  16   MEP id:  Buzzards Bay > Westport > Angeline Brook
17
index:  17   MEP id:  Buzzards Bay > Westport > Kirby Brook
18
index:  18   MEP id:  Buzzards Bay > Westport > Snell Creek
20
index:  20   MEP id:  Buzzards Bay > Westport > Westport River 2
21
index:  21   MEP id:  Cape Cod > Allen > Cold Spring Brook
22
index:  22   MEP id:  Cape Cod > Allen > E Saquatucket Stream
23
index:  23   MEP id:  Cape Cod > Allen > Un-name Creek at Kilde Rd
24
index:  24   MEP id:  Cape Cod > Barnstable > Alder Brook
25
index:  25   MEP id:  Cape Cod > Barnstable > Boat Cove Creek
26
index:  26   MEP id:  Cape Cod > Barnstable > Bridge Creek
27
index:  27   MEP id:  Cape Cod > Barnstable > Chase Garden Creek
28
index:  28   MEP id:  Cape Cod > Barnstable > Maraspin Creek
29
index:  29   MEP id:  Cape Cod > Barnstable > Whites Brook Discharge
30
index:  30   MEP id:  Cape Cod > Bass River > Fresh Pond
```

31
index: 31   MEP id: Cape Cod > Bass River > Hamblin Brook
33
index: 33   MEP id: Cape Cod > Centerville > Bumps River
34
index: 34   MEP id: Cape Cod > Centerville > Lake Elizabeth Stream
35
index: 35   MEP id: Cape Cod > Centerville > Long Pond Stream
36
index: 36   MEP id: Cape Cod > Centerville > Skunknett River
37
index: 37   MEP id: Cape Cod > Chatham > Ryder Cove
38
index: 38   MEP id: Cape Cod > Chatham > Stillwater Pond
39
index: 39   MEP id: Cape Cod > Falmouth > Morse Pond
40
index: 40   MEP id: Cape Cod > Fiddlers > Cedar Lake
41
index: 41   MEP id: Cape Cod > Fiddlers > Flax Pond
42
index: 42   MEP id: Cape Cod > GBB > Backus Brook
43
index: 43   MEP id: Cape Cod > GBB > Bournes Brook
44
index: 44   MEP id: Cape Cod > GBB > Coonamesset River
45
index: 45   MEP id: Cape Cod > Herring River > Stream 1
46
index: 46   MEP id: Cape Cod > Herring River > Stream 2
47
index: 47   MEP id: Cape Cod > JEHU/Waquoit Bay > Quashnet River
48
index: 48   MEP id: Cape Cod > Lewis Bay > Chase Brook
49
index: 49   MEP id: Cape Cod > Lewis Bay > Halls Creek
50
index: 50   MEP id: Cape Cod > Lewis Bay > Hosptial Bog
51
index: 51   MEP id: Cape Cod > Lewis Bay > Mill Pond
52
index: 52   MEP id: Cape Cod > Lewis Bay > Snow's Creek
53
index: 53   MEP id: Cape Cod > Lewis Bay > Stewart's Creek
54
index: 54   MEP id: Cape Cod > Little Pond > Stream
55
index: 55   MEP id: Cape Cod > Magansett > Cuffs Pond
56
index: 56   MEP id: Cape Cod > Namskaket > Cedar Pond
57
index: 57   MEP id: Cape Cod > Namskaket > Hurley Bog
58
index: 58   MEP id: Cape Cod > Namskaket > Stream (Little)
59
index: 59   MEP id: Cape Cod > Nauset > Mary Chase Marsh Creek
60
index: 60   MEP id: Cape Cod > Oyster Pond > Mosquito/Quivett Creek
61
index: 61   MEP id: Cape Cod > Parkers River > Forest Brook

```
62
index:  62   MEP id:  Cape Cod > Parkers River > Plashes Brook
63
index:  63   MEP id:  Cape Cod > Phinneys > Back River
64
index:  64   MEP id:  Cape Cod > Pleasant Bay > Kescayo Gansett
65
index:  65   MEP id:  Cape Cod > Pleasant Bay > Paw Wah Pond
66
index:  66   MEP id:  Cape Cod > Pleasant Bay > Ryder Cove
67
index:  67   MEP id:  Cape Cod > Pleasant Bay > Stillwater Pond
68
index:  68   MEP id:  Cape Cod > Pleasant Bay > Tar Kiln
69
index:  69   MEP id:  Cape Cod > Plymouth Duxbury > Eel River
70
index:  70   MEP id:  Cape Cod > Plymouth Duxbury > Jones River
71
index:  71   MEP id:  Cape Cod > Plymouth Duxbury > Town Brook
72
index:  72   MEP id:  Cape Cod > Popponesset > Mashpee River
73
index:  73   MEP id:  Cape Cod > Popponesset > Santuit River
78
index:  78   MEP id:  Cape Cod > Sandwich > Shawme Lake
79
index:  79   MEP id:  Cape Cod > Sandwich > Springhill Creek
80
index:  80   MEP id:  Cape Cod > Scorton > Long Creek
81
index:  81   MEP id:  Cape Cod > Scorton > Scorton Creek
82
index:  82   MEP id:  Cape Cod > Swan Pond > Un-Named Creek
83
index:  83   MEP id:  Cape Cod > Three Bays > Little River
84
index:  84   MEP id:  Cape Cod > Three Bays > Marstons Mill River
85
index:  85   MEP id:  Cape Cod > Waquoit Bay > Childs River
86
index:  86   MEP id:  Cape Cod > Wellfleet > Fresh Brook
87
index:  87   MEP id:  Cape Cod > Wellfleet > Hatches Creek
88
index:  88   MEP id:  Cape Cod > Wellfleet > Herring River
91
index:  91   MEP id:  Islands > Chillmark > Fulling Mill Brook East
92
index:  92   MEP id:  Islands > Chillmark > Fulling Mill Brook West
93
index:  93   MEP id:  Islands > Chillmark > Mill Brook
100
index:  100   MEP id:  Islands > Menemsha > Black Brook
101
index:  101   MEP id:  Islands > Menemsha > Pease Point Brook
102
index:  102   MEP id:  Islands > Menemsha > Un-named Brook
107
index:  107   MEP id:  Islands > Tashmoo > Fish Ladder
```

In [125…

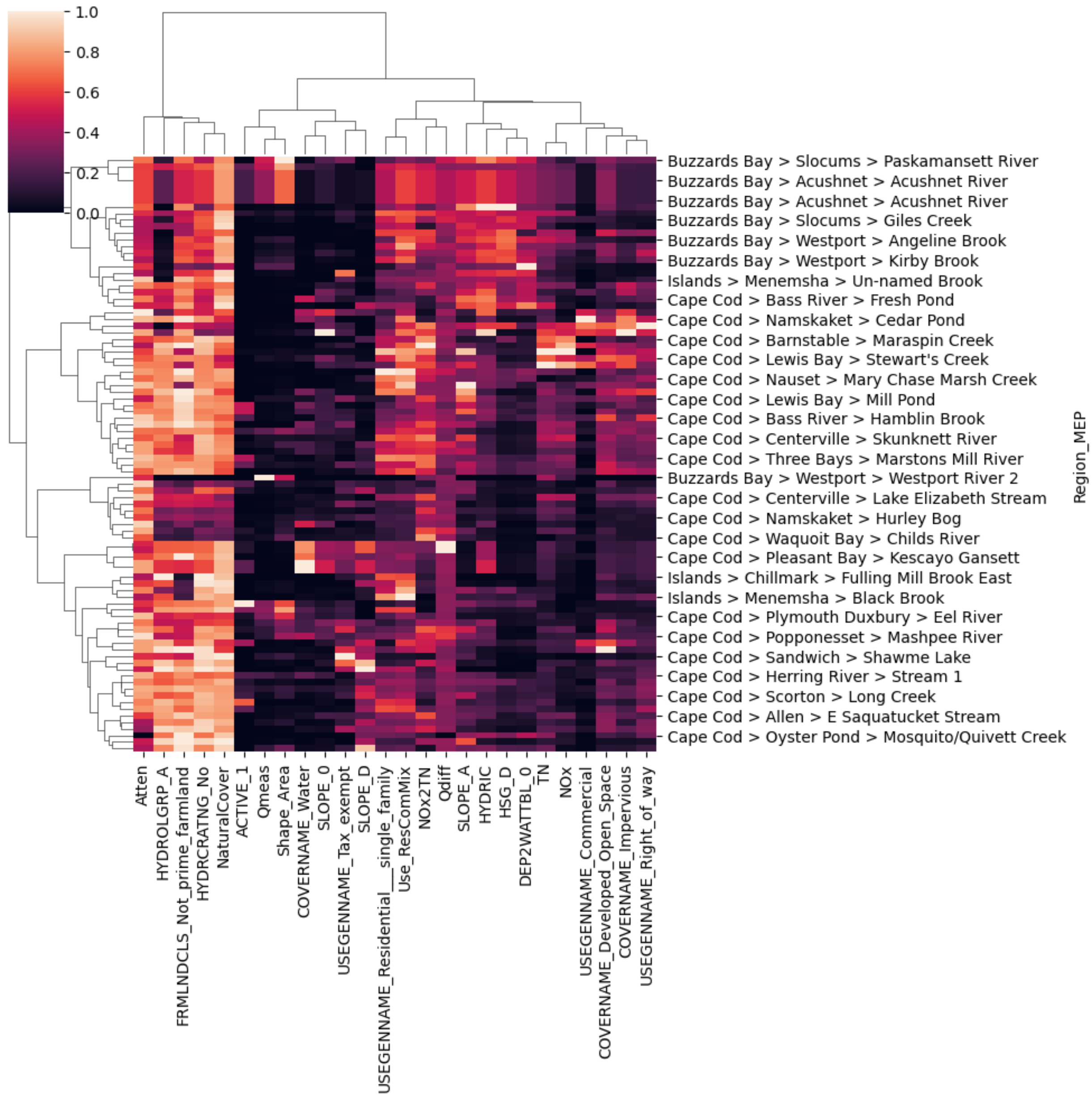
```
selected = ['Shape_Area',
            'HYDRCRATNG_No',
            'COVERNAME_Water','HSG_D','HYDRIC',
            'SLOPE_0','SLOPE_A',
            'NaturalCover','USEGENNAME_Commercial',
            'ACTIVE_1',
            #'NLEACHING_Low',
'COVERNAME_Developed_Open_Space', # positive
'COVERNAME_Impervious', # exponential positive
'USEGENNAME_Residential___single_family', # exponential - positive
'USEGENNAME_Right_of_way', # positive
'USEGENNAME_Tax_exempt', # negative inverse
'Use_ResComMix', # positive
'SLOPE_D', # inverse, slope D is steep slopes betwee 15-25% grade
'DEP2WATTBL_0', # inverse negative
'FRMLNDCLS_Not_prime_farmland', # logistic/exponential positive
#'NLEACHING_High', # logistic/exponential positive
'HYDROLGRP_A', # negative
           ]

selected_le5 = ["LE5_"+i for i in selected]
selected_gt5 = ["GT5_"+i for i in selected]

selected_watershed_features = selected
#selected_watershed_features = selected_le5 + selected_gt5
```

In [126…

```
sns.clustermap(df[selected_monitoring_cols+selected_watershed_features].set_index(df['
```

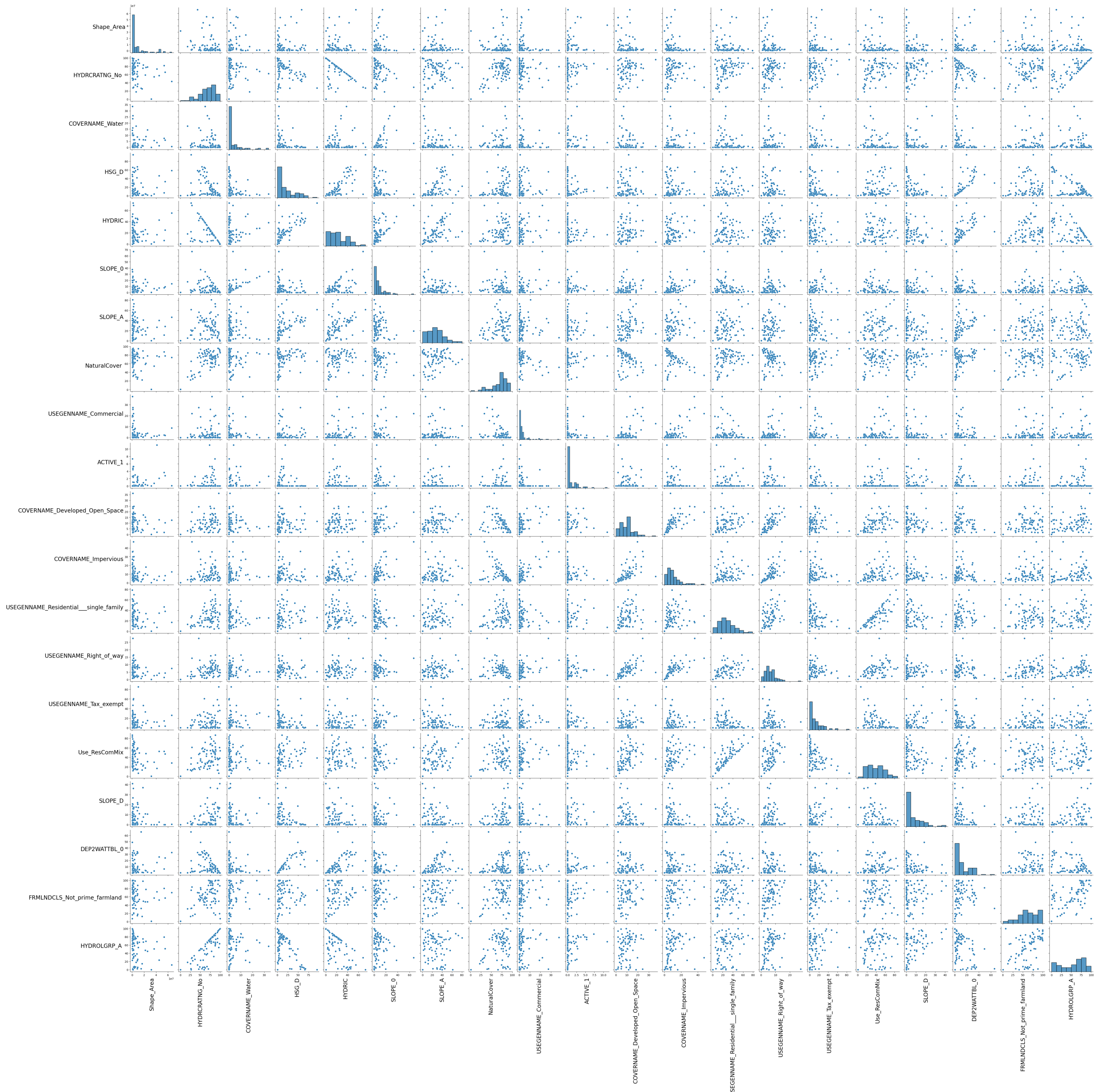
Out[126]:

```
<seaborn.matrix.ClusterGrid at 0x1de83212f70>
```

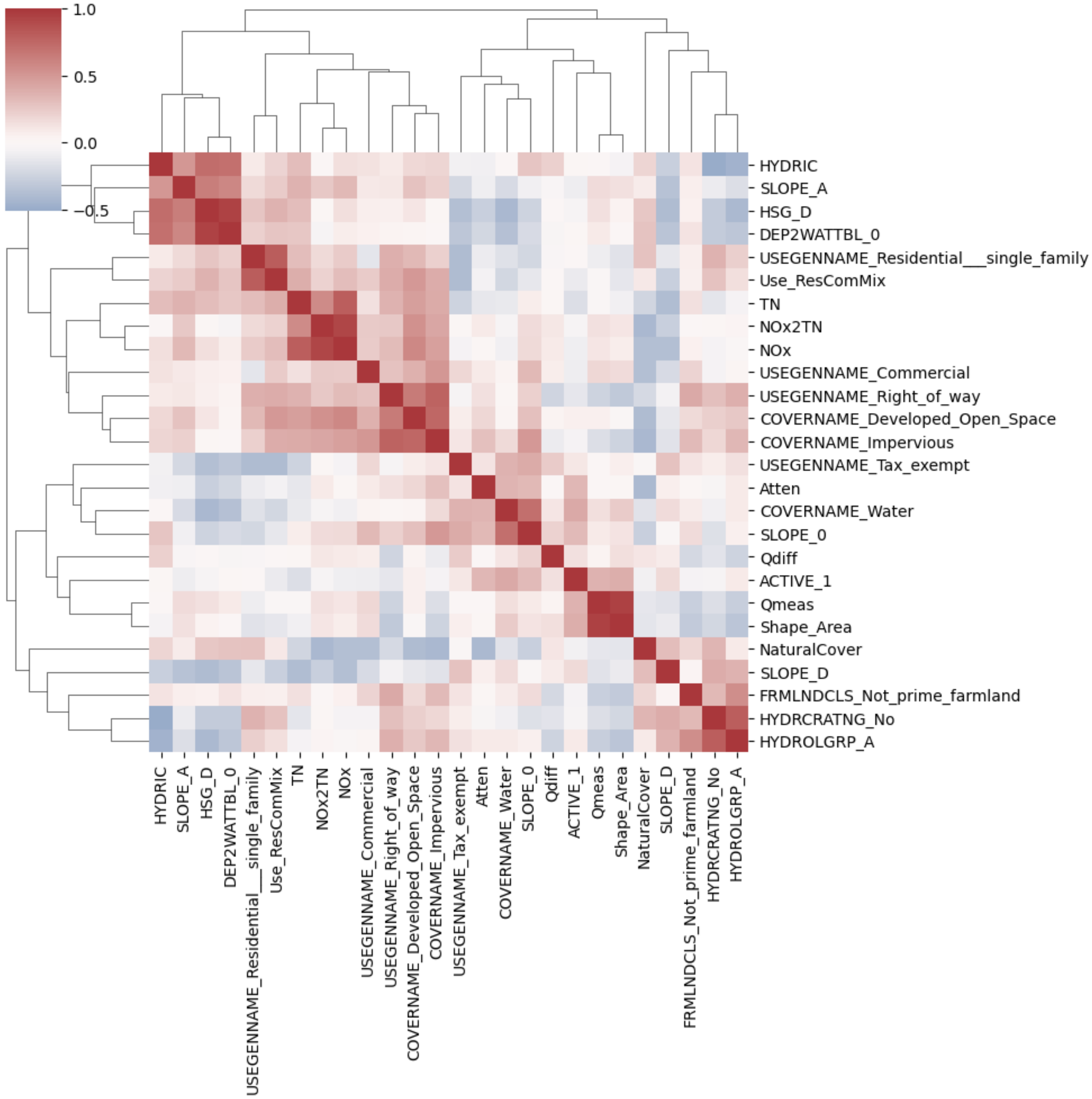


In [127…

```python
with sns.plotting_context(rc={"axes.labelsize":20}):
    g = sns.pairplot(df[selected_watershed_features]) # show only lower triangle
    for ax in g.axes.flatten():
        # rotate x axis labels
        ax.set_xlabel(ax.get_xlabel(), rotation = 90)
        # rotate y axis labels
        ax.set_ylabel(ax.get_ylabel(), rotation = 0)
        # set y labels alignment
        ax.yaxis.get_label().set_horizontalalignment('right')
```



In [128…
```
sns.clustermap(df[selected_monitoring_cols+selected_watershed_features].corr('spearmar
```
Out[128]:
```
<seaborn.matrix.ClusterGrid at 0x1de93977730>
```



with sns.plotting_context(rc={"axes.labelsize":14}): sns.pairplot(selected_watershed_features, corner=True) # show only lower triangle

MODELING

Linear Regression

In [129…

```
# modeling
# Importing libraries for building linear regression model
#import statsmodels
import statsmodels.api as sm

from statsmodels.stats.outliers_influence import variance_inflation_factor

# Importing libraries for scaling the data
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
# To ignore warnings
```

```
import warnings
warnings.filterwarnings("ignore")
```

In [130…

```
# select Y
y_name = 'NOx'
_ = df
#_ = df[df['Atten']>-50] # remove outlier attenuation
_ = _[_[y_name].notna()]
#train_target = np.log(_[y_name]) # the untransformed models perform better than trans
train_target = _[y_name]

# select Xs
#selected_features = selected
selected_features = selected_le5 + selected_gt5 + selected
#selected_cols = selected_cols
train_features_selected = _.loc[:, ~_.columns.isin(df_monitoring_avg.columns)][selecte
train_features_extended = _.loc[:, ~_.columns.isin(df_monitoring_avg.columns)]._get_nu
#train_features = np.log(train_features_selected + 1)
train_features = train_features_selected
```

In [131…

```
# scale the X data
scaler = StandardScaler()
# Applying fit_transform on the training features data
train_features_scaled = scaler.fit_transform(train_features)
# The above scaler returns the data in array format, below we are converting it back t
train_features_scaled = pd.DataFrame(train_features_scaled, index = train_features.inc
train_features_scaled.head()
```

Out[131]:

	LE5_Shape_Area	LE5_HYDRCRATNG_No	LE5_COVERNAME_Water	LE5_HSG_D	LE5_HYDRIC	LE5_SLOI
0	1.694124	-0.392446	-0.332147	0.36588	0.778098	0.22:
1	1.694124	-0.392446	-0.332147	0.36588	0.778098	0.22:
2	1.694124	-0.392446	-0.332147	0.36588	0.778098	0.22:
3	1.694124	-0.392446	-0.332147	0.36588	0.778098	0.22:
4	1.694124	-0.392446	-0.332147	0.36588	0.778098	0.22:

5 rows × 60 columns

In [132…

```
from sklearn.model_selection import train_test_split
y = train_target
X = train_features_scaled
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.3, random_state
```

In [133…

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNet
from sklearn.metrics import r2_score
y = train_target
X = train_features_scaled
rg = Ridge(alpha=0.03)
ls = Lasso(alpha=0.02)
```

```
en = ElasticNet(alpha=0.03,l1_ratio=0.7)
lr = LinearRegression()
```

In [134…

```
rg.fit(X_train, y_train)
ls.fit(X_train, y_train)
lr.fit(X_train, y_train)
en.fit(X_train, y_train)
```

Out[134]:

```
ElasticNet(alpha=0.03, l1_ratio=0.7)
```

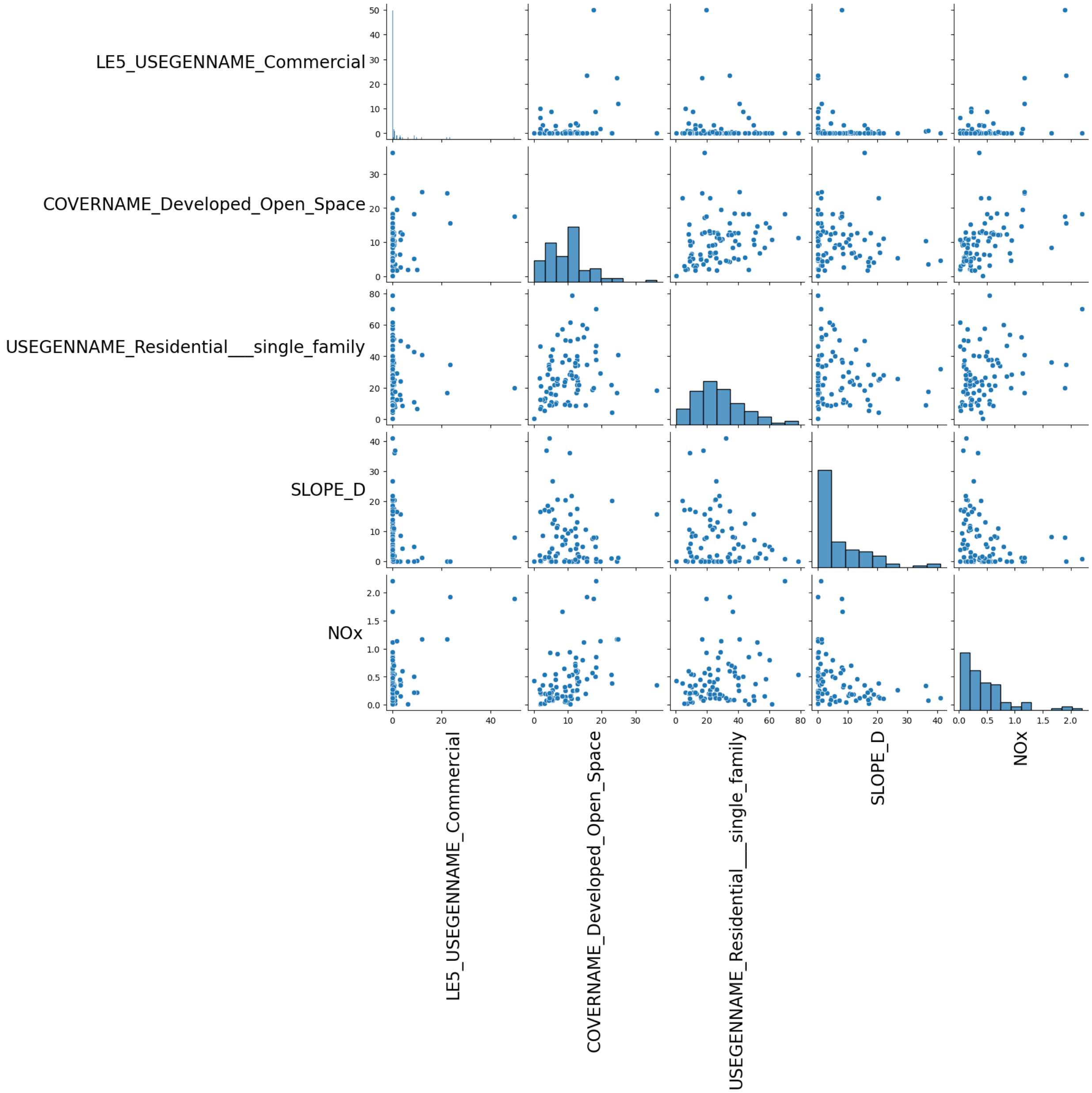
In [135…

```
from sklearn.feature_selection import SelectFromModel, RFECV, RFE
from sklearn.linear_model import LassoCV
from sklearn.linear_model import ElasticNetCV

# Use L1 penalty
estimator = LassoCV(cv=10, normalize = True)
#estimator = ElasticNetCV(cv=10, normalize = True)

n_feats = int(np.floor(X.shape[0]/10/2))
# the optimal number of features based on AIC is 4
print("maximum number of features {}".format(n_feats))
if n_feats > X.shape[1]:
    n_feats=X.shape[1]
# Set a minimum threshold of 0.25
m = SelectFromModel(estimator, prefit=False, norm_order=1, max_features=n_feats)
m.fit(X, y)
feature_idx = m.get_support()
feature_name = X.columns[feature_idx]
print(feature_name)
# get variance inflation factor
vif = fn_get_vif(feature_name,X)
print(vif)
```

```
maximum number of features 4
Index(['LE5_USEGENNAME_Commercial', 'COVERNAME_Developed_Open_Space',
       'USEGENNAME_Residential___single_family', 'SLOPE_D'],
      dtype='object')
VIF is not of concern if less than 3
                                             VIF  Tolerance
LE5_USEGENNAME_Commercial               1.093074   0.914851
COVERNAME_Developed_Open_Space          1.139548   0.877541
USEGENNAME_Residential___single_family  1.114182   0.897520
SLOPE_D                                 1.058202   0.945000
```

In [136…

```
with sns.plotting_context(rc={"axes.labelsize":20}):
    g = sns.pairplot(df[feature_name].join(df[y_name])) # show only lower triangle
    for ax in g.axes.flatten():
        # rotate x axis labels
        ax.set_xlabel(ax.get_xlabel(), rotation = 90)
        # rotate y axis labels
        ax.set_ylabel(ax.get_ylabel(), rotation = 0)
        # set y labels alignment
        ax.yaxis.get_label().set_horizontalalignment('right')
```



In [137…

```
#import cvoxpt
# Adding the intercept term
train_features_scaled_select = sm.add_constant(train_features_scaled[feature_name])
train_features_scaled.columns

# Calling the OLS algorithm on the train features and the target variable
ols_model_0 = sm.OLS(train_target, train_features_scaled_select)

# Fitting the Model
ols_res_0 = ols_model_0.fit()
#ols_res_0 = ols_model_0.fit_regularized(method='Lasso')

display(ols_res_0.summary())
```

OLS Regression Results

Dep. Variable:	NOx	**R-squared:**	0.474
Model:	OLS	**Adj. R-squared:**	0.449
Method:	Least Squares	**F-statistic:**	19.13
Date:	Mon, 06 Nov 2023	**Prob (F-statistic):**	3.00e-11
Time:	12:14:59	**Log-Likelihood:**	-22.341
No. Observations:	90	**AIC:**	54.68
Df Residuals:	85	**BIC:**	67.18
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.4554	0.034	13.538	0.000	0.389	0.522
LE5_USEGENNAME_Commercial	0.1784	0.035	5.072	0.000	0.108	0.248
COVERNAME_Developed_Open_Space	0.1292	0.036	3.598	0.001	0.058	0.201
USEGENNAME_Residential__single_family	0.0876	0.036	2.467	0.016	0.017	0.158
SLOPE_D	-0.0865	0.035	-2.499	0.014	-0.155	-0.018

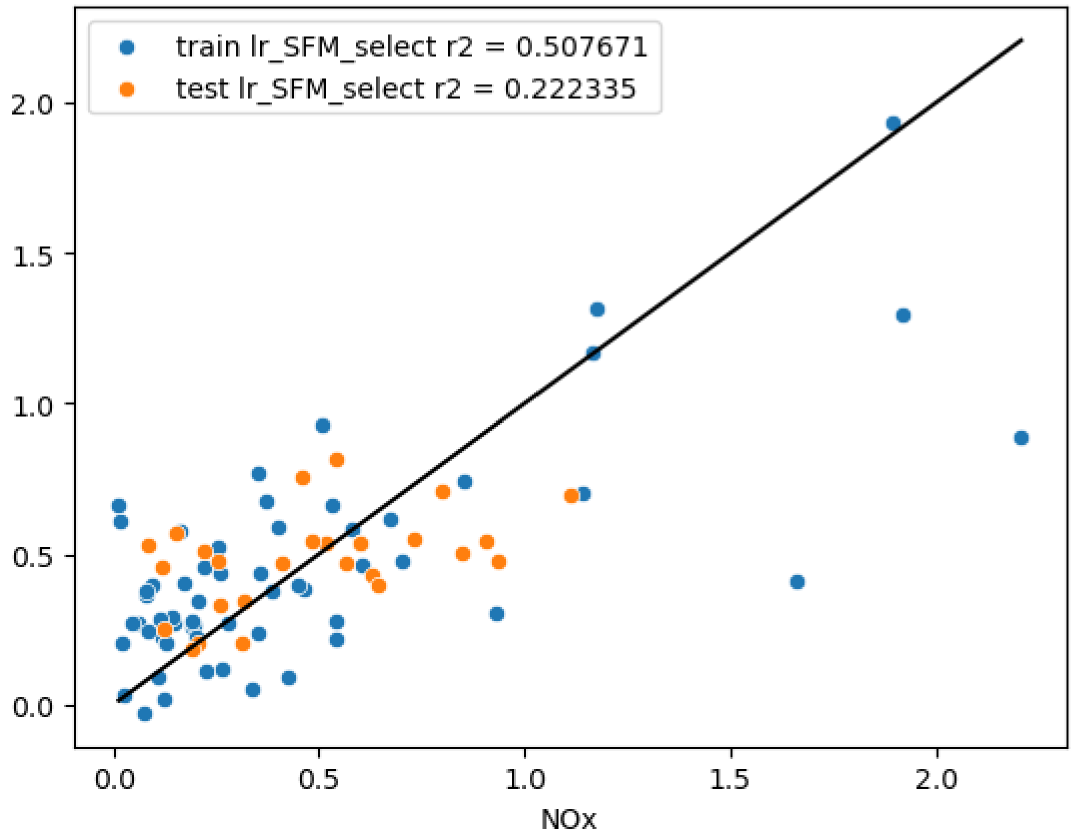
Omnibus:	43.013	**Durbin-Watson:**	1.657
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	134.432
Skew:	1.592	**Prob(JB):**	6.43e-30
Kurtosis:	8.070	**Cond. No.**	1.53

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [138…

```
lr_select = lr.fit(X_train[feature_name],y_train)
fn_sklearn_cross_val_scores(lr_select,X[feature_name],y)
fn_plot_obs_vs_pred(lr_select,X_test[feature_name],X_train[feature_name],y_test,y_trai
```

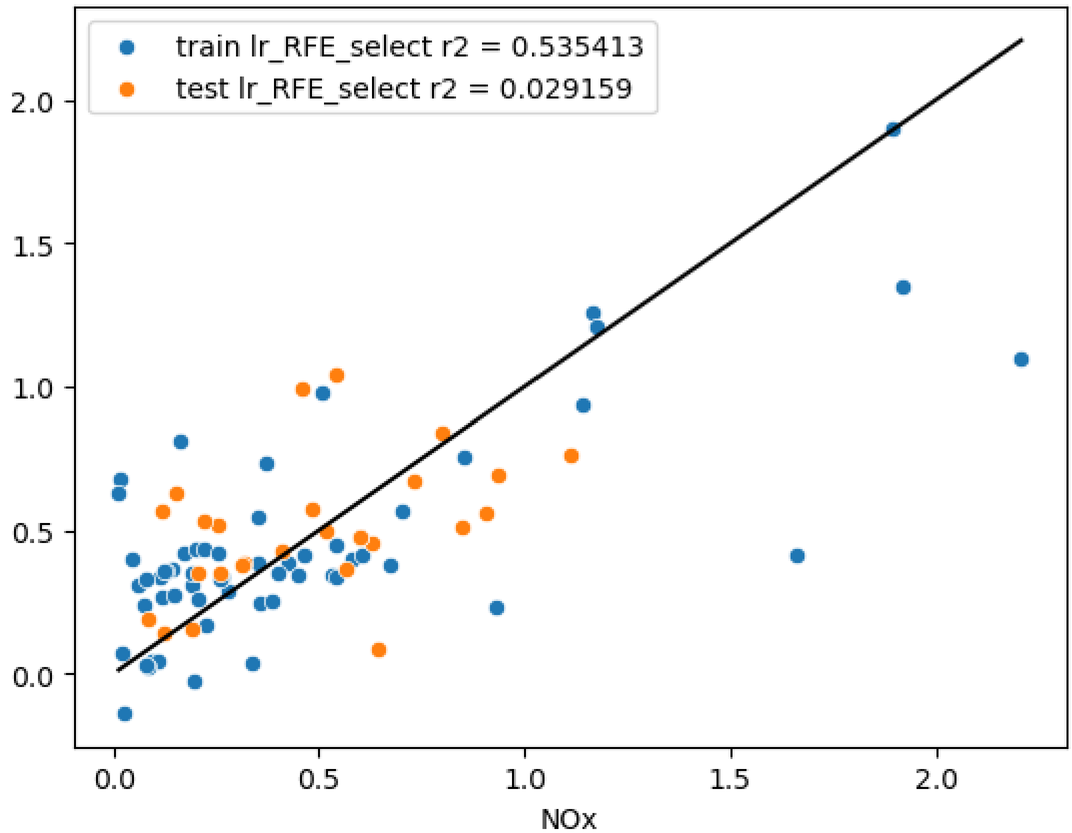
```
running cross validation with ShuffleSplit: n_splits=18, test_size=0.3, rand_state =
0
0.25 accuracy with a standard deviation of 0.34
```



In [139…

```
# recursive feature elimination
# THIS TAKES A LONG TIME TO RUN
m = RFE(estimator,n_features_to_select=n_feats)
m.fit(X,y)
feature_idx = m.get_support()
feature_name = X.columns[feature_idx]
print(feature_name)
print(fn_get_vif(feature_name,X))
lr_select = lr.fit(X_train[feature_name],y_train)
fn_sklearn_cross_val_scores(lr_select,X[feature_name],y)
fn_plot_obs_vs_pred(lr_select,X_test[feature_name],X_train[feature_name],y_test,y_trai
```

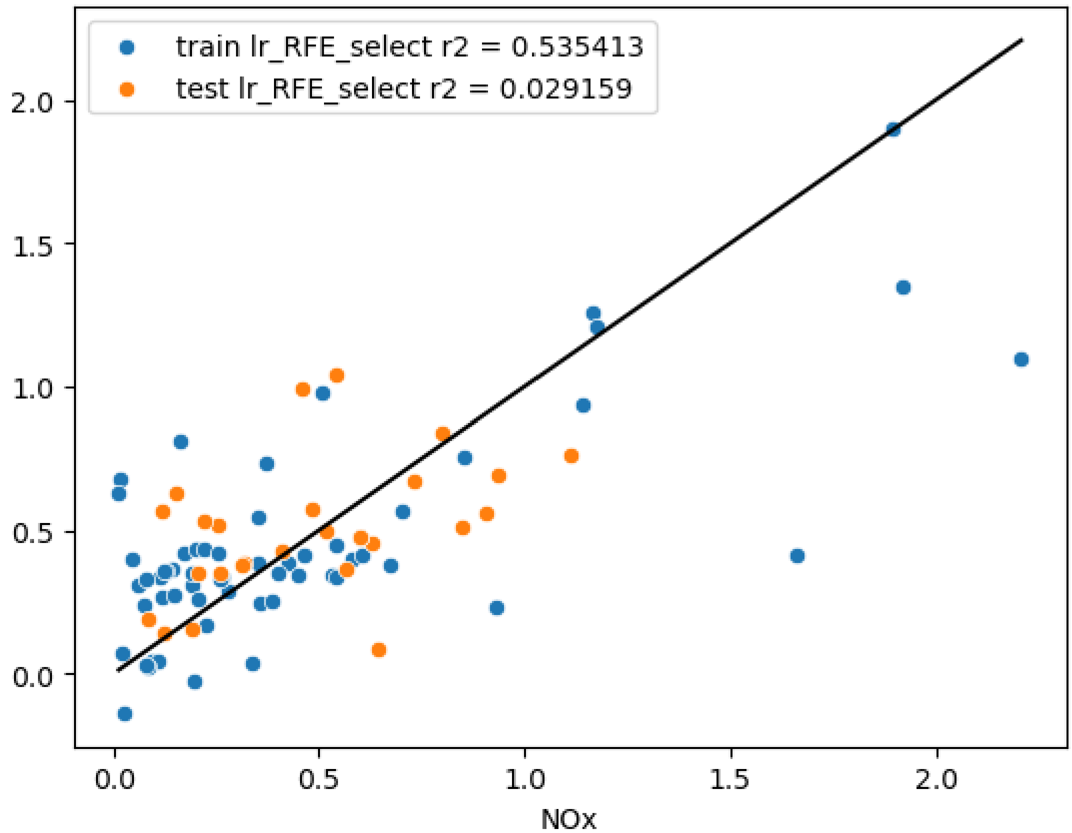
```
Index(['LE5_USEGENNAME_Commercial', 'LE5_FRMLNDCLS_Not_prime_farmland',
       'NaturalCover', 'USEGENNAME_Residential___single_family'],
      dtype='object')
VIF is not of concern if less than 3
                                             VIF  Tolerance
LE5_USEGENNAME_Commercial               1.047878   0.954309
LE5_FRMLNDCLS_Not_prime_farmland        1.164445   0.858778
NaturalCover                            1.285727   0.777770
USEGENNAME_Residential___single_family  1.145267   0.873159
running cross validation with ShuffleSplit: n_splits=18, test_size=0.3, rand_state =
0
0.25 accuracy with a standard deviation of 0.43
```



In [140…

```
# forward feature selection
from sklearn.feature_selection import SequentialFeatureSelector
SequentialFeatureSelector(estimator, n_features_to_select=n_feats)
feature_idx = m.get_support()
feature_name = X.columns[feature_idx]
print(feature_name)
vif = fn_get_vif(feature_name,X)
print(vif)
lr_select = lr.fit(X_train[feature_name],y_train)
fn_sklearn_cross_val_scores(lr_select,X[feature_name],y)
fn_plot_obs_vs_pred(lr_select,X_test[feature_name],X_train[feature_name],y_test,y_trai
```

```
Index(['LE5_USEGENNAME_Commercial', 'LE5_FRMLNDCLS_Not_prime_farmland',
       'NaturalCover', 'USEGENNAME_Residential___single_family'],
      dtype='object')
VIF is not of concern if less than 3
                                             VIF   Tolerance
LE5_USEGENNAME_Commercial               1.047878   0.954309
LE5_FRMLNDCLS_Not_prime_farmland        1.164445   0.858778
NaturalCover                            1.285727   0.777770
USEGENNAME_Residential___single_family  1.145267   0.873159
running cross validation with ShuffleSplit: n_splits=18, test_size=0.3, rand_state =
0
0.25 accuracy with a standard deviation of 0.43
```



In [141…

```
# Adding the intercept term
train_features_scaled_select = sm.add_constant(train_features_scaled[feature_name])
train_features_scaled.columns
from functions import *

# get variance inflation factor
vif = fn_get_vif(feature_name,train_features_scaled_select)
print(vif)
# Calling the OLS algorithm on the train features and the target variable
ols_model_0 = sm.OLS(train_target, train_features_scaled_select)

# Fitting the Model
ols_res_0 = ols_model_0.fit()
#ols_res_0 = ols_model_0.fit_regularized(method='sqrt_lasso')

display(ols_res_0.summary())
```

```
VIF is not of concern if less than 3
                                         VIF  Tolerance
LE5_USEGENNAME_Commercial           1.047878   0.954309
LE5_FRMLNDCLS_Not_prime_farmland    1.164445   0.858778
NaturalCover                        1.285727   0.777770
USEGENNAME_Residential___single_family  1.145267   0.873159
```

OLS Regression Results

Dep. Variable:	NOx	**R-squared:**	0.477
Model:	OLS	**Adj. R-squared:**	0.453
Method:	Least Squares	**F-statistic:**	19.41
Date:	Mon, 06 Nov 2023	**Prob (F-statistic):**	2.26e-11
Time:	12:15:22	**Log-Likelihood:**	-22.031
No. Observations:	90	**AIC:**	54.06
Df Residuals:	85	**BIC:**	66.56
Df Model:	4		
Covariance Type:	nonrobust		

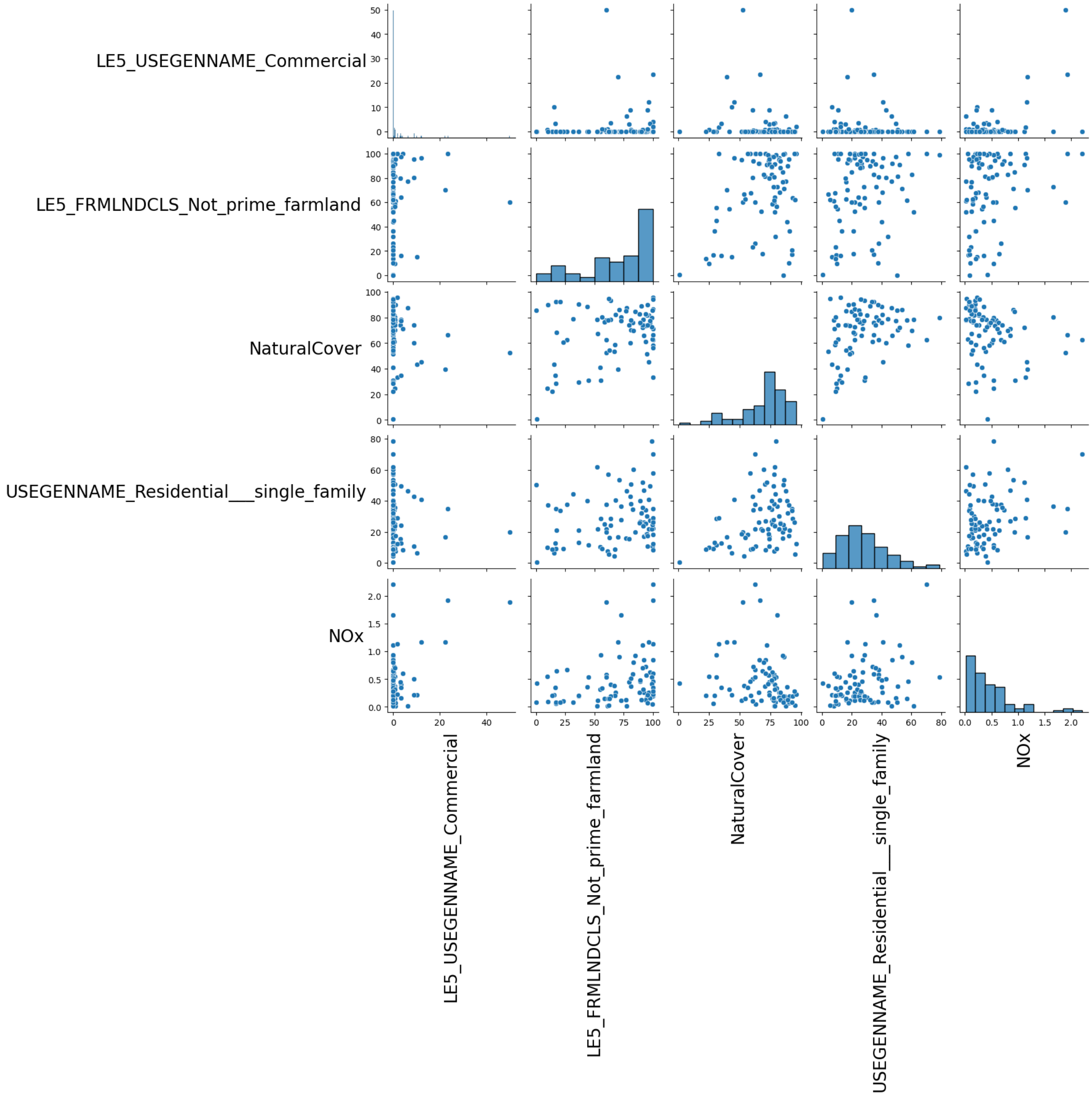
	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	0.4554	0.034	13.585	0.000	0.389	0.522
LE5_USEGENNAME_Commercial	0.1916	0.034	5.584	0.000	0.123	0.260
LE5_FRMLNDCLS_Not_prime_farmland	0.1155	0.036	3.194	0.002	0.044	0.187
NaturalCover	-0.1566	0.038	-4.119	0.000	-0.232	-0.081
USEGENNAME_Residential__single_family	0.1623	0.036	4.523	0.000	0.091	0.234

Omnibus:	33.425	**Durbin-Watson:**	1.711
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	86.223
Skew:	1.273	**Prob(JB):**	1.89e-19
Kurtosis:	7.064	**Cond. No.**	1.69

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [142…

```
with sns.plotting_context(rc={"axes.labelsize":20}):
    g = sns.pairplot(df[feature_name].join(df[y_name])) # show only lower triangle
    for ax in g.axes.flatten():
        # rotate x axis labels
        ax.set_xlabel(ax.get_xlabel(), rotation = 90)
        # rotate y axis labels
        ax.set_ylabel(ax.get_ylabel(), rotation = 0)
        # set y labels alignment
        ax.yaxis.get_label().set_horizontalalignment('right')
```



In [143…

```
# CROSS VALIDATATION FOR MODEL HYPERPARAMETER TUNING
```

In [144…

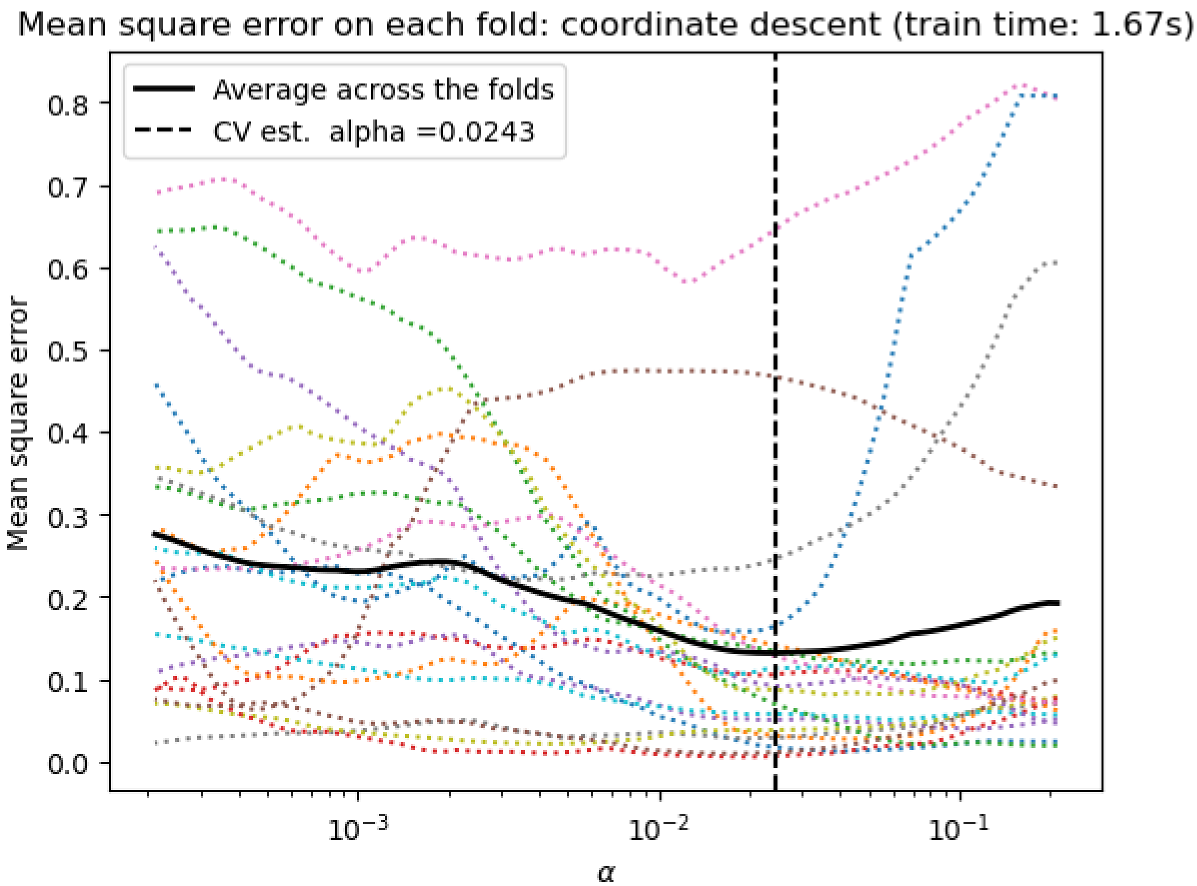
```
from sklearn.linear_model import LassoCV
import time
from sklearn.pipeline import make_pipeline
start_time = time.time()
model = make_pipeline(StandardScaler(), LassoCV(cv=20)).fit(X, y)
fit_time = time.time() - start_time
```

In [145…

```
import matplotlib.pyplot as plt

lasso = model[-1]
plt.semilogx(lasso.alphas_, lasso.mse_path_, linestyle=":")
plt.plot(
    lasso.alphas_,
    lasso.mse_path_.mean(axis=-1),
    color="black",
    label="Average across the folds",
    linewidth=2,
)
```

```
plt.axvline(lasso.alpha_, linestyle="--", color="black", label="CV est.  alpha ="+str(
print(lasso.alpha_)
#plt.ylim()
plt.xlabel(r"$\alpha$")
plt.ylabel("Mean square error")
plt.legend()
_ = plt.title(
    f"Mean square error on each fold: coordinate descent (train time: {fit_time:.2f}s)
)
```

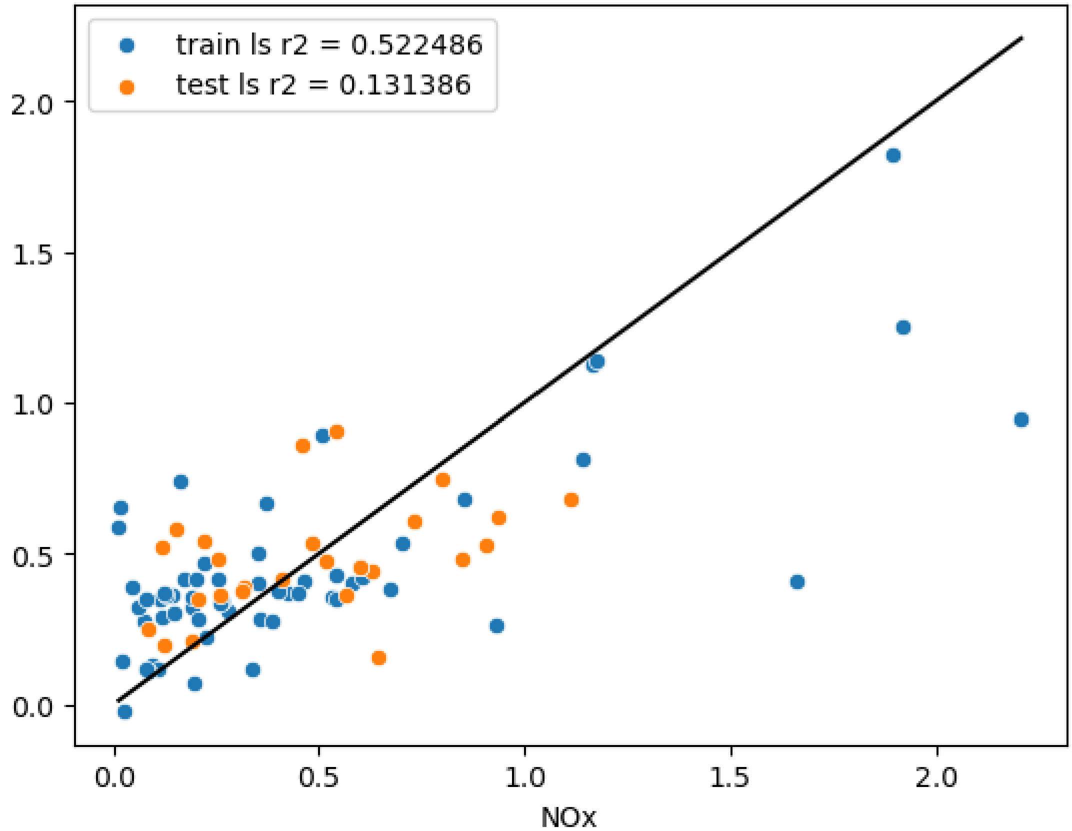
0.024330971411646456



In [146…

```
# fit the model with alpha
ls = Lasso(alpha=lasso.alpha_)
ls.fit(X_train[feature_name],y_train)
fn_sklearn_cross_val_scores(ls,X[feature_name],y)
fn_plot_obs_vs_pred(ls,X_test[feature_name],X_train[feature_name],y_test,y_train,'ls')
```

running cross validation with ShuffleSplit: n_splits=18, test_size=0.3, rand_state =
0
0.27 accuracy with a standard deviation of 0.30



Tree and Ensemble Learning

Here I test methods of regression trees for explaining variation in NOx

I use SHAP - SHapley Addative exPlanations to explain the impact of variables on NOx values

Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. Advances in neural information processing systems, 30.

In [147…

```
# select Y
y_name = 'NOx'
_ = df
#_ = df[df['Atten']>-50] # remove outlier attenuation
_ = _[_[y_name].notna()]
#train_target = np.log(_[y_name])
train_target = _[y_name]
# select Xs
selected_features = selected
#selected_features = selected_le5 + selected_gt5 + selected
#selected_cols = selected_cols
train_features_selected = _.loc[:, ~_.columns.isin(df_monitoring_avg.columns)][selecte
train_features_extended = _.loc[:, ~_.columns.isin(df_monitoring_avg.columns)]._get_nu
#train_features = np.log(train_features_selected + 1)
train_features = train_features_selected
```

In [148…

```
# scale the X data
scaler = StandardScaler()
# Applying fit_transform on the training features data
train_features_scaled = scaler.fit_transform(train_features)
```

```
# The above scaler returns the data in array format, below we are converting it back t
train_features_scaled = pd.DataFrame(train_features_scaled, index = train_features.inc
train_features_scaled.head()
```

Out[148]:

	Shape_Area	HYDRCRATNG_No	COVERNAME_Water	HSG_D	HYDRIC	SLOPE_0	SLOPE_A	Natur
0	2.358451	-0.518085	-0.390442	1.276314	1.216053	-0.080809	0.744286	(
1	2.358451	-0.518085	-0.390442	1.276314	1.216053	-0.080809	0.744286	(
2	2.358451	-0.518085	-0.390442	1.276314	1.216053	-0.080809	0.744286	(
3	2.358451	-0.518085	-0.390442	1.276314	1.216053	-0.080809	0.744286	(
4	2.358451	-0.518085	-0.390442	1.276314	1.216053	-0.080809	0.744286	(

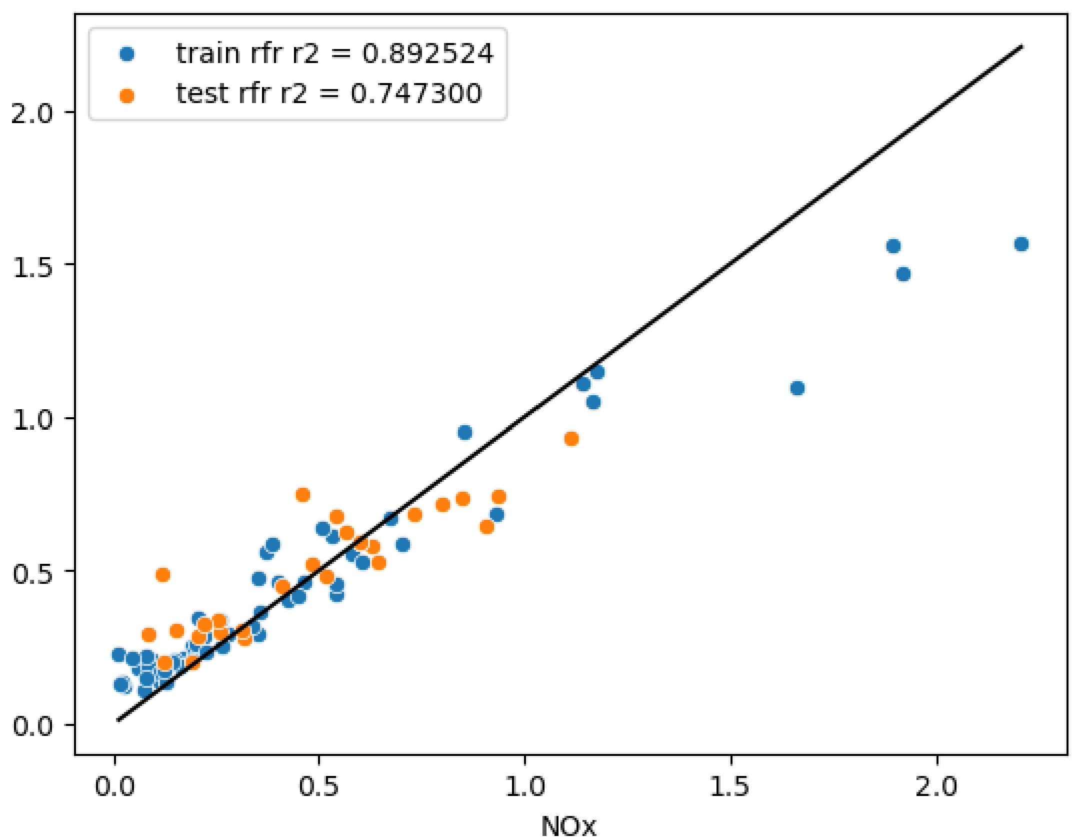
In [149…

```
from sklearn.model_selection import train_test_split
y = train_target
X = train_features_scaled
X_train, X_test, y_train, y_test  = train_test_split(X, y, test_size=0.3, random_state
```

In [150…

```
# random forest
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(X, y)
fn_sklearn_cross_val_scores(rfr,X,y)
# plot obs vs fitted
fn_plot_obs_vs_pred(rfr,X_test,X_train,y_test,y_train,'rfr')
```

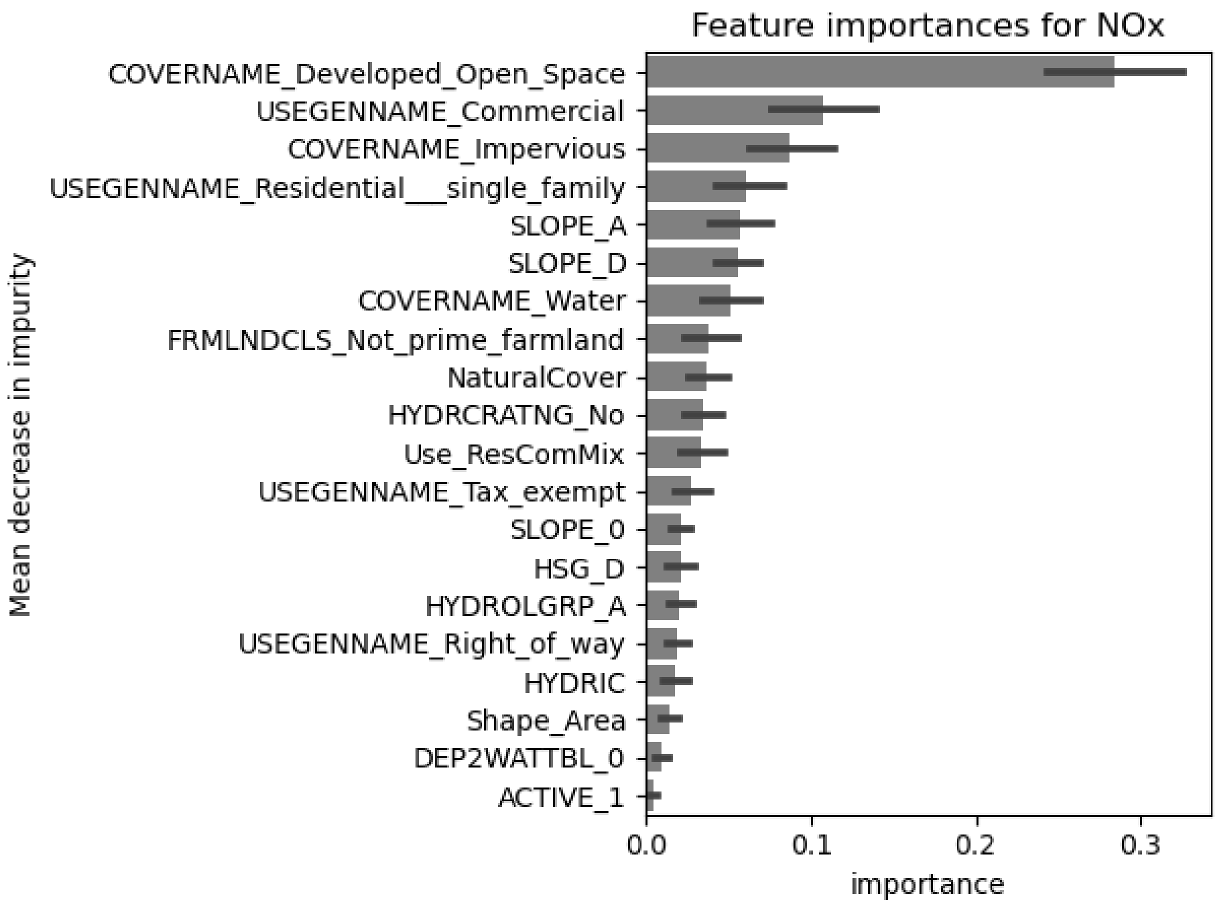
```
running cross validation with ShuffleSplit: n_splits=18, test_size=0.3, rand_state =
0
-0.21 accuracy with a standard deviation of 0.70
```



In [151…

```
df_importance = fn_ensemble_feature_importance_plot(rfr,X.columns,y_name)
```

```
Elapsed time to compute the importances: 0.005 seconds
```



In [152…

```
def fn_shap_model_explainer(
    m,# fitted model
    X,# array of explanatory variables
    fig_prefix="rfr",
    # save figures to png
    save_figs = True
    ):
    %matplotlib auto
    '''
    # use this line to install shap if needed
    !pip install shap
    # https://shap.readthedocs.io/en/latest/example_notebooks/api_examples/plots/heatm
    '''
    import shap
    # Fits the explainer
    explainer = shap.Explainer(m, X) # Calculates the SHAP values - It takes some time
    shap_values = explainer(X)

    # Plot beeswarm
    shap.plots.beeswarm(shap_values,max_display=10)
    plt.savefig('{}_shap_beeswarm.png'.format(fig_prefix), format='png', dpi=600, bbox
    plt.close()

    # Plot heatmap
    shap.plots.heatmap(shap_values,instance_order=shap_values.sum(1),max_display=10,sh
    plt.savefig('{}_shap_heatmap.png'.format(fig_prefix), format='png', dpi=600, bbox_
    plt.close()
```

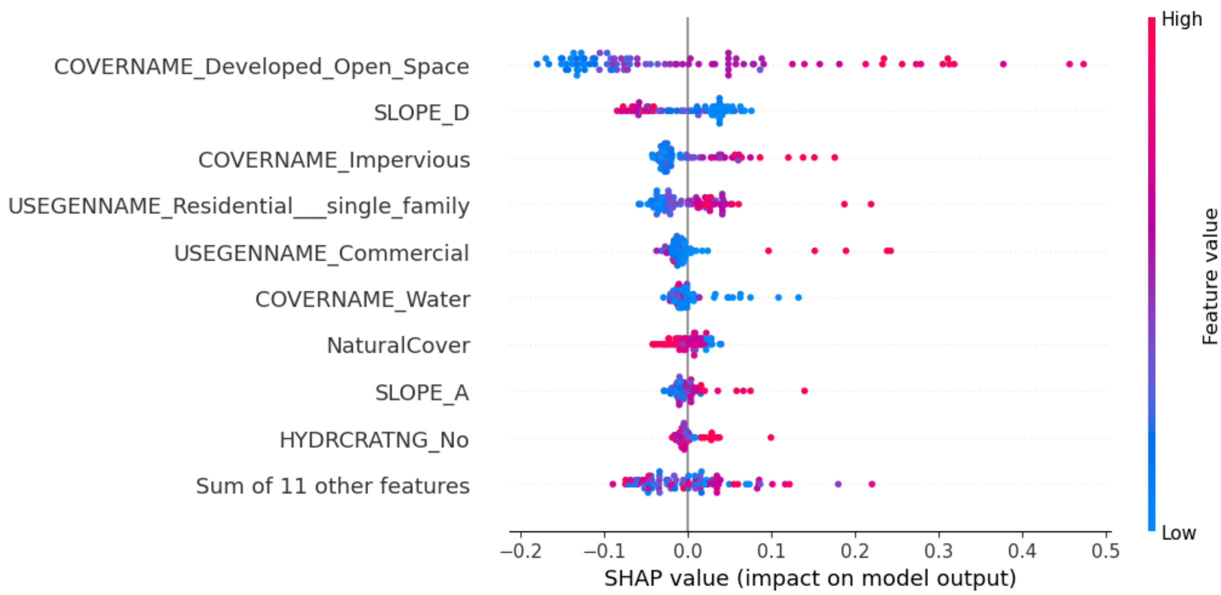
```
    # Plot waterfall
    shap.plots.waterfall(shap_values[0])
    plt.savefig('{}_shap_waterfall.png'.format(fig_prefix), format='png', dpi=600, bbc
    plt.close()
    %matplotlib inline
```

In [153…

```
# Fits the explainer
import shap
m = rfr
fig_prefix = "rfr"
explainer = shap.Explainer(m, X) # Calculates the SHAP values - It takes some time
shap_values = explainer(X)
```

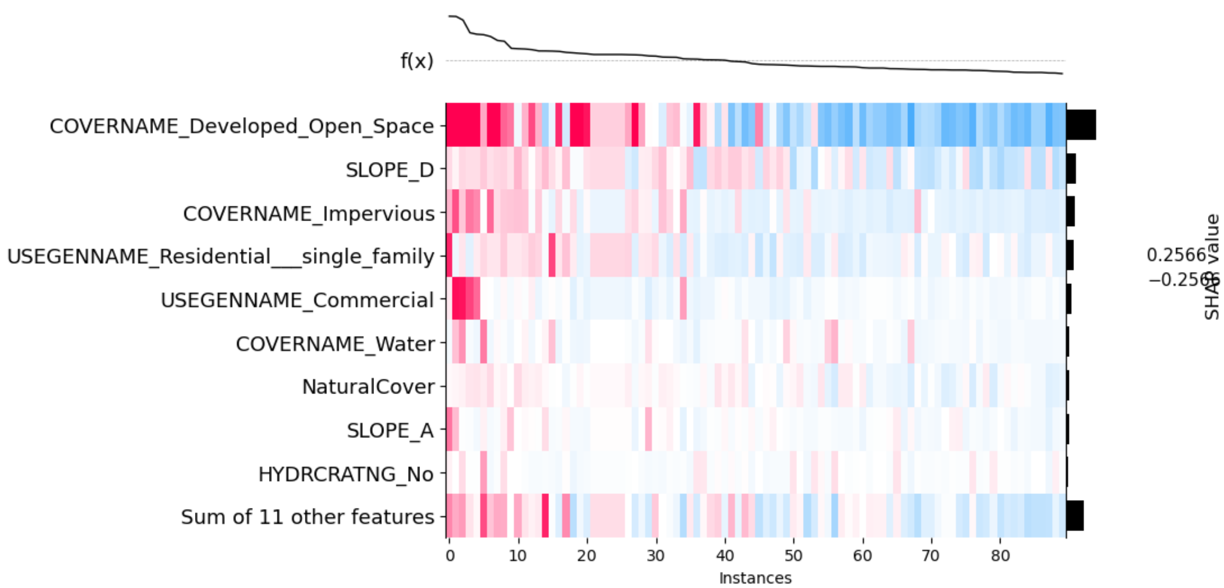
In [154…

```
# Plot beeswarm
shap.plots.beeswarm(shap_values,max_display=10)
plt.savefig('{}_shap_beeswarm.png'.format(fig_prefix), format='png', dpi=600, bbox_inc
plt.close()
```



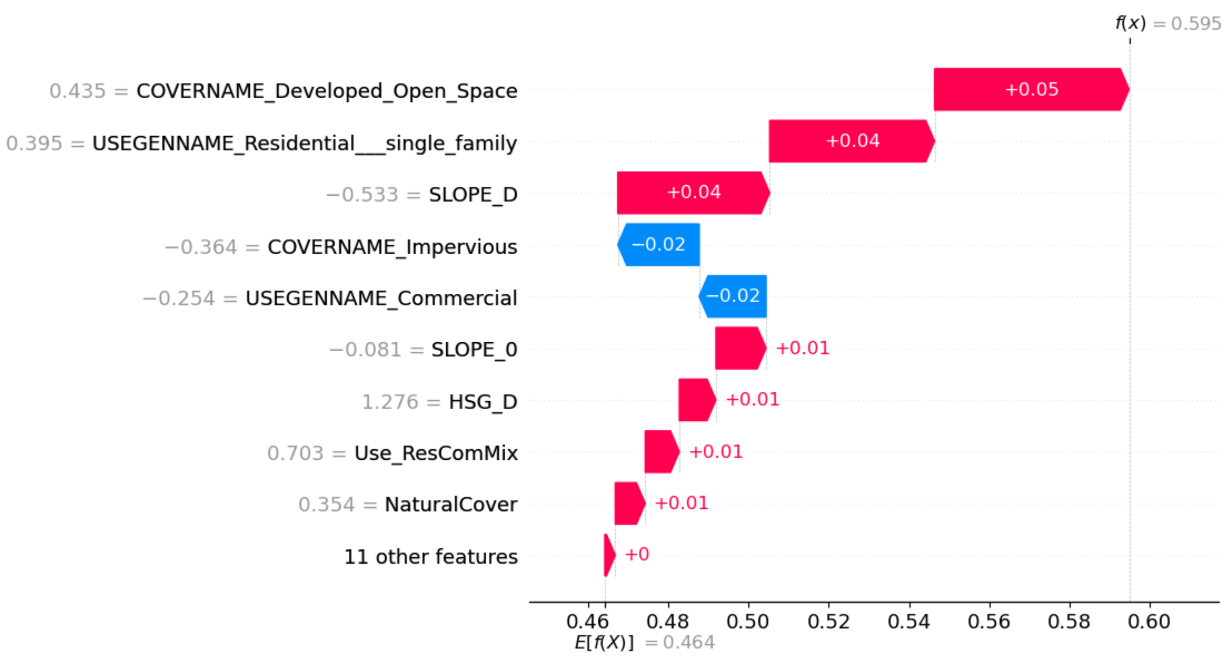
In [155…

```
# Plot heatmap
shap.plots.heatmap(shap_values,instance_order=shap_values.sum(1),max_display=10)
plt.savefig('{}_shap_heatmap.png'.format(fig_prefix), format='png', dpi=600, bbox_inch
plt.close()
```



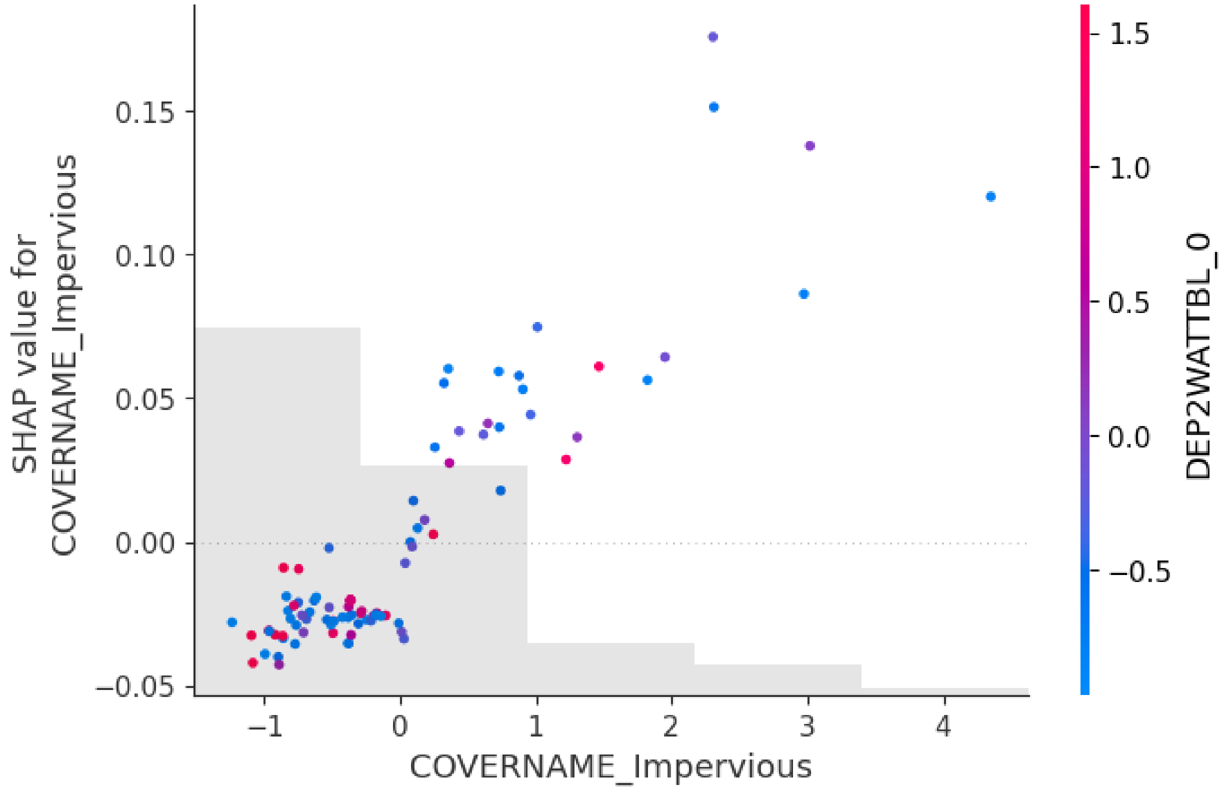
In [156…

```
# Plot waterfall
shap.plots.waterfall(shap_values[0])
plt.savefig('{}_shap_waterfall.png'.format(fig_prefix), format='png', dpi=600, bbox_ir
plt.close()
```



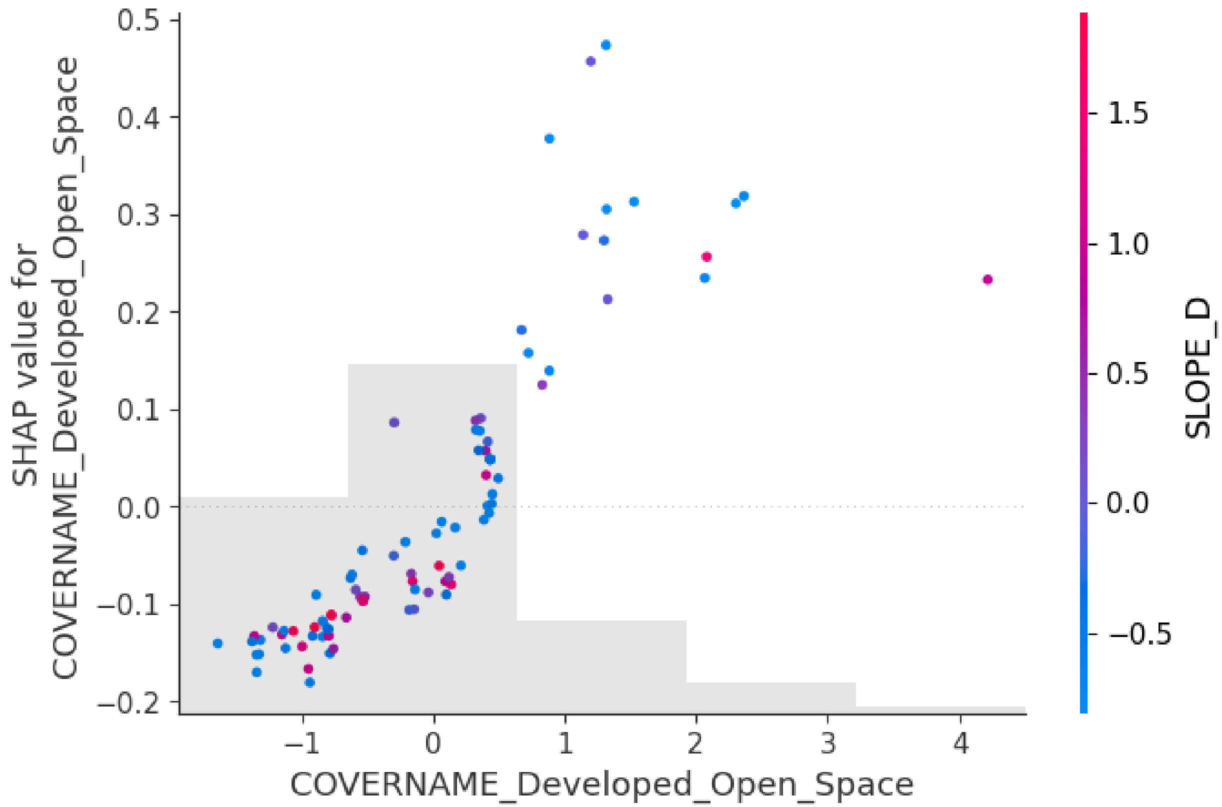
In [157…

```
shap.plots.scatter(shap_values[:,"COVERNAME_Impervious"], color=shap_values)
```



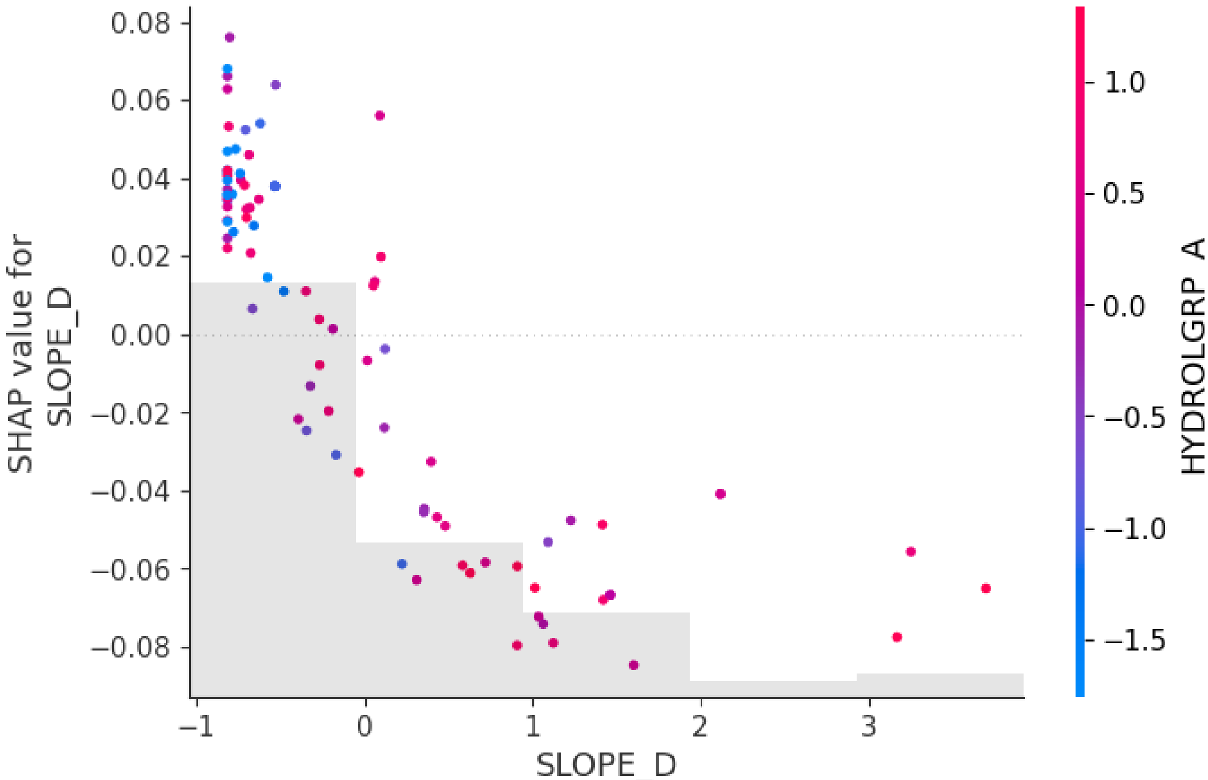
In [158…

```
shap.plots.scatter(shap_values[:,"COVERNAME_Developed_Open_Space"],color=shap_values)
```



In [159…

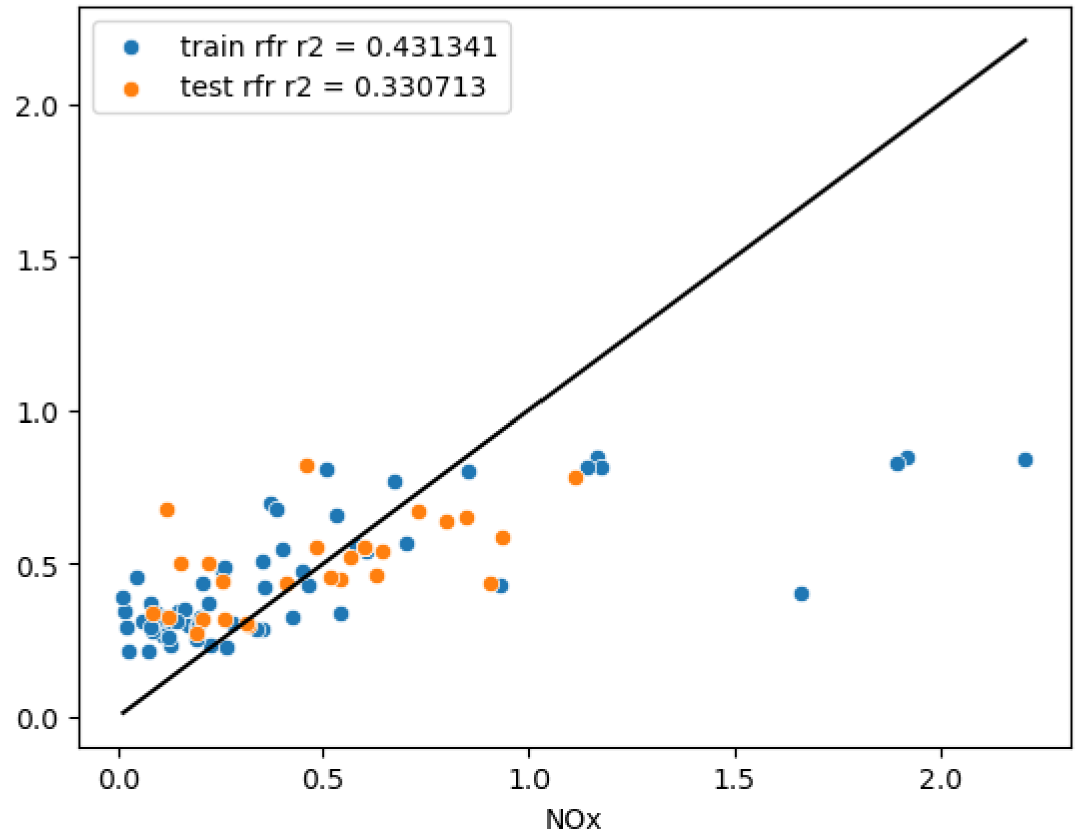
```
shap.plots.scatter(shap_values[:,"SLOPE_D"], color=shap_values)
```



In [160…

```
# random forest
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(max_features=4,max_depth=3,min_samples_leaf=10,n_estimator
rfr.fit(X, y)
fn_sklearn_cross_val_scores(rfr,X,y)
# plot obs vs fitted
fn_plot_obs_vs_pred(rfr,X_test,X_train,y_test,y_train,'rfr')
```

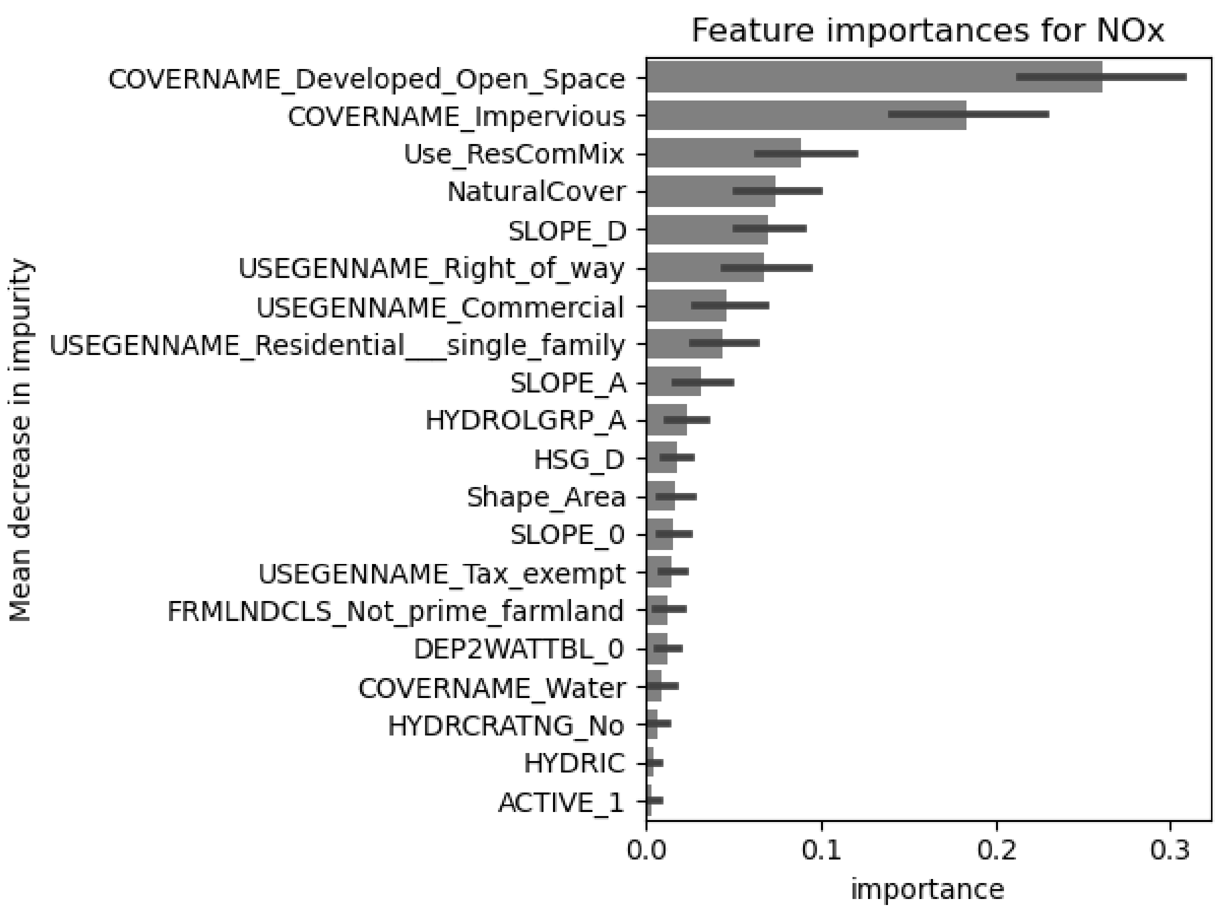
```
running cross validation with ShuffleSplit: n_splits=18, test_size=0.3, rand_state =
0
0.13 accuracy with a standard deviation of 0.16
```



In [161…

```
df_importance = fn_ensemble_feature_importance_plot(rfr,X.columns,y_name)
```

```
Elapsed time to compute the importances: 0.023 seconds
```



In [162…

```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(rfr.get_params())

from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 2, stop = 400, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [1,3,10]
#max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [4, 8, 16]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 3, 6]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print("\nRandom Search Grid:\n")
pprint(random_grid)
```

```
Parameters currently in use:

{'bootstrap': True,
 'ccp_alpha': 0.0,
 'criterion': 'squared_error',
 'max_depth': 3,
 'max_features': 4,
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 10,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 200,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}

Random Search Grid:

{'bootstrap': [True, False],
 'max_depth': [1, 3, 10],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 3, 6],
 'min_samples_split': [4, 8, 16],
 'n_estimators': [2, 46, 90, 134, 178, 223, 267, 311, 355, 400]}
```

In [163…

```
rfr_random = RandomizedSearchCV(estimator = rfr, param_distributions = random_grid, n_
```

In [164…

```python
# Fit the random search model
rfr_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

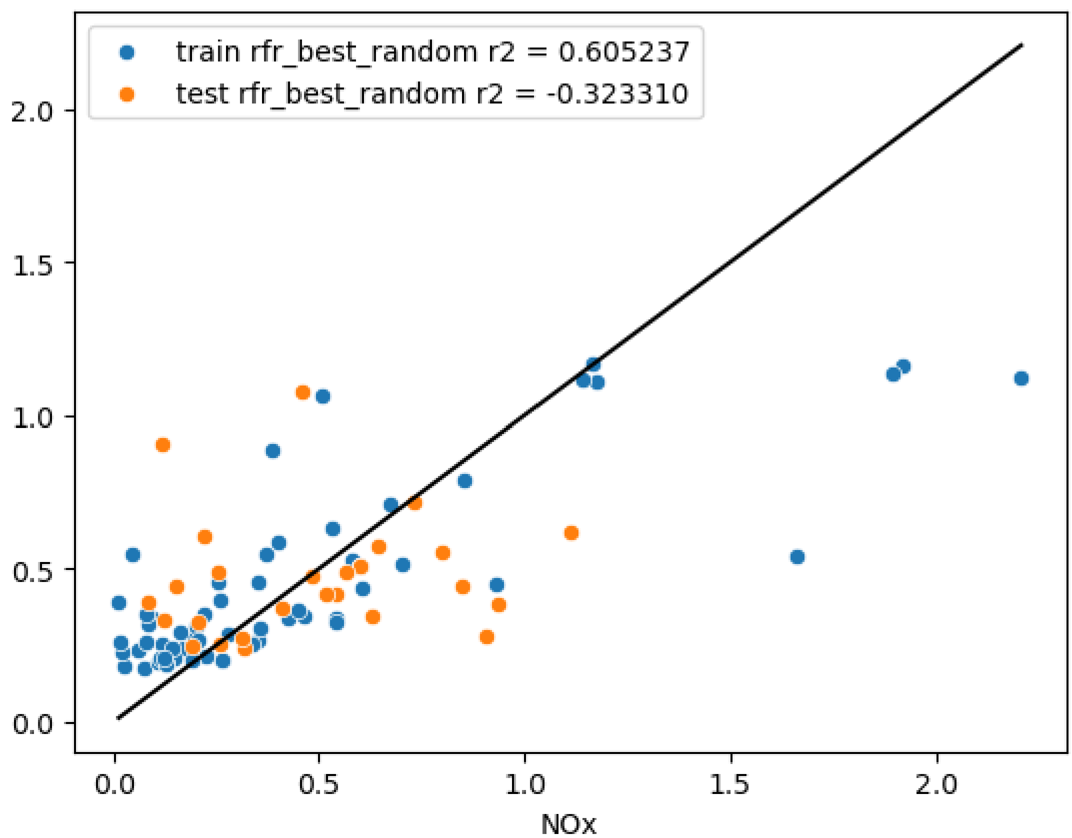
Out[164]:

```
RandomizedSearchCV(cv=3,
                   estimator=RandomForestRegressor(max_depth=3, max_features=4,
                                                   min_samples_leaf=10,
                                                   n_estimators=200),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [1, 3, 10],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 3, 6],
                                        'min_samples_split': [4, 8, 16],
                                        'n_estimators': [2, 46, 90, 134, 178,
                                                         223, 267, 311, 355,
                                                         400]},
                   random_state=42, verbose=2)
```

In [165…

```python
# Fit the random search model
print(rfr_random.best_params_)

# save the best random trianed model
rfr_best_random = rfr_random.best_estimator_
fn_sklearn_cross_val_scores(rfr_best_random,X,y)
# plot obs vs fitted
fn_plot_obs_vs_pred(rfr_best_random,X_test,X_train,y_test,y_train,'rfr_best_random')
```

```
{'n_estimators': 46, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features':
'sqrt', 'max_depth': 3, 'bootstrap': False}
running cross validation with ShuffleSplit: n_splits=18, test_size=0.3, rand_state =
0
0.08 accuracy with a standard deviation of 0.33
```



In [166…

```python
from sklearn.model_selection import GridSearchCV
# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [3,6,12],
    'max_features': ['sqrt','log2'],
    'min_samples_leaf': [3, 4, 5, 6],
    'min_samples_split': [8,10,12],
    'n_estimators': [i for i in range(150,250,by=5)]}
# Instantiate the grid search model
rfr_grid = GridSearchCV(estimator = rfr, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)
# Fit the random search model
rfr_grid.fit(X_train, y_train)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_35740\1795181749.py in <module>
      7     'min_samples_leaf': [3, 4, 5, 6],
      8     'min_samples_split': [8,10,12],
----> 9     'n_estimators': [i for i in range(150,250,by=5)]}
     10 # Instantiate the grid search model
     11 rfr_grid = GridSearchCV(estimator = rfr, param_grid = param_grid,

TypeError: range() takes no keyword arguments
```

In [ ]:

```python
# Fit the random search model
print(rfr_grid.best_params_)

# pull out best estimator
rfr_best_grid = rfr_grid.best_estimator_
fn_sklearn_cross_val_scores(rfr_best_grid,X,y)

# plot obs vs fitted
fn_plot_obs_vs_pred(rfr_best_grid,X_test,X_train,y_test,y_train,'rfr_best_grid')
```

In [ ]:

```python
df_importance = fn_ensemble_feature_importance_plot(rfr_best_grid,X.columns,y_name)
```

In [ ]:

```python
feature_name = df_importance.varname[:10]
ls_select = estimator.fit(X_train[feature_name],y_train)

fn_plot_obs_vs_pred(ls_select,X_test[feature_name],X_train[feature_name],y_test,y_trai
```

In [ ]:

```python
from sklearn.ensemble import GradientBoostingRegressor
# define model and select hyperparameters
gbr = GradientBoostingRegressor(min_samples_split=20,min_samples_leaf=20,max_depth=2,l

# fit model to training data
gbr.fit(X_train,y_train)
fn_sklearn_cross_val_scores(gbr,X,y)
# plot obs vs fitted
fn_plot_obs_vs_pred(gbr,X_test,X_train,y_test,y_train,'gbr')
```

In [ ]:

```python
## DECISION TREE
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor

rng = np.random.RandomState(1)
```

```
regr_1 = DecisionTreeRegressor()

ada = AdaBoostRegressor(
    DecisionTreeRegressor(max_depth=2), n_estimators=1000, random_state=rng
)
ada.fit(X_train,y_train)
fn_plot_obs_vs_pred(ada,X_test,X_train,y_test,y_train,'ada')
```

In [ ]:
```
fn_sklearn_cross_val_scores(ada,X,y)
```

In [ ]:
```
df_importance = fn_ensemble_feature_importance_plot(ada,X.columns,y_name)
```

In [ ]:
```
feature_name = df_importance.varname[:10]
lr_select = lr.fit(X_train[feature_name],y_train)
fn_sklearn_cross_val_scores(lr_select,X,y)
fn_plot_obs_vs_pred(lr_select,X_test[feature_name],X_train[feature_name],y_test,y_trai
```

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression

#X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=rng)

# LinearRegression on PCA
pcr = make_pipeline(StandardScaler(), PCA(n_components=n_feats), ls)
pcr.fit(X_train, y_train)
fn_sklearn_cross_val_scores(pcr,X,y)
pca = pcr.named_steps["pca"]  # retrieve the PCA step of the pipeline

# Partial Least Squares
pls = PLSRegression(n_components=n_feats)
pls.fit(X_train[selected_features], y_train)
fn_sklearn_cross_val_scores(pls,X,y)
```

In [ ]:
```
fn_plot_obs_vs_pred(pcr,X_test,X_train,y_test,y_train,'pcr')
```

In [ ]:
```
fn_plot_obs_vs_pred(pls,X_test[selected_features],X_train[selected_features],y_test,y_
```

Appendix

Unused code snippets

In [ ]:
```
# Creating histograms
vars = ["NOx2TN","TN","NOx",'Atten','Qdiff','Qmeas']
#var = vars[0]
#fig, axes = plt.subplots(2, 3, figsize = (20, 6))
for var in vars:
    sns.histplot(x=var,data=df_monitoring_avg)
    plt.show()
```

```
In [ ]: sns.scatterplot(x='Use_ResComMix',y='NO3 (mg/L)',data=df)
```

y_pred_lr = lr.predict(X_train) y_pred_ls = ls.predict(X_train) y_pred_rg = rg.predict(X_train) y_pred_en = en.predict(X_train) r2_score_lr = r2_score(y_train, y_pred_lr) print("ols r^2 on training data : %f" % r2_score_lr) sns.scatterplot(x=y_train,y=y_pred_lr,alpha=0.5) plt.plot([0,y.max()], [0,y.max()], color='r') r2_score_ls = r2_score(y_train, y_pred_ls) print("lasso r^2 on training data : %f" % r2_score_ls) sns.scatterplot(x=y_train,y=y_pred_ls,alpha=0.5) r2_score_rg = r2_score(y_train, y_pred_rg) print("ridge r^2 on training data : %f" % r2_score_rg) sns.scatterplot(x=y_train,y=y_pred_rg,alpha=0.5) r2_score_en = r2_score(y_train, y_pred_en) print("elastic r^2 on training data : %f" % r2_score_en) sns.scatterplot(x=y_train,y=y_pred_en,alpha=0.5)y_pred_lr = lr.predict(X_test) y_pred_ls = ls.predict(X_test) y_pred_rg = rg.predict(X_test) y_pred_en = en.predict(X_test) r2_score_lr = r2_score(y_test, y_pred_lr) #print("ols r^2 on testing data : %f" % r2_score_lr) #sns.scatterplot(x=y_test,y=y_pred_lr) plt.plot([0,y.max()], [0,y.max()], color='black') r2_score_ls = r2_score(y_test, y_pred_ls) print("lasso r^2 on testing data : %f" % r2_score_ls) sns.scatterplot(x=y_test,y=y_pred_ls) r2_score_rg = r2_score(y_test, y_pred_rg) print("ridge r^2 on testing data : %f" % r2_score_rg) sns.scatterplot(x=y_test,y=y_pred_rg) r2_score_en = r2_score(y_test, y_pred_en) print("elastic r^2 on testing data : %f" % r2_score_en) sns.scatterplot(x=y_test,y=y_pred_en)import time from sklearn.preprocessing import StandardScaler from sklearn.linear_model import LassoLarsIC from sklearn.pipeline import make_pipeline start_time = time.time() lasso_lars_ic = make_pipeline(StandardScaler(), LassoLarsIC(criterion="aic")).fit(X, y) fit_time = time.time() - start_timeresults = pd.DataFrame({ "alphas": lasso_lars_ic[-1].alphas_, "AIC criterion": lasso_lars_ic[-1].criterion_, }).set_index("alphas") alpha_aic = lasso_lars_ic[-1].alpha_lasso_lars_ic.set_params(lassolarsic__criterion="bic").fit(X, y) results["BIC criterion"] = lasso_lars_ic[-1].criterion_ alpha_bic = lasso_lars_ic[-1].alpha_def highlight_min(x): x_min = x.min() return ["font-weight: bold" if v == x_min else "" for v in x] results.style.apply(highlight_min)ax = results.plot() ax.vlines(alpha_aic, results["AIC criterion"].min(), results["AIC criterion"].max(), label="alpha: AIC estimate", linestyles="--", color="tab:blue",) ax.vlines(alpha_bic, results["BIC criterion"].min(), results["BIC criterion"].max(), label="alpha: BIC estimate", linestyle="--", color="tab:orange",) ax.set_xlabel(r"α") ax.set_ylabel("criterion") ax.set_xscale("log") ax.legend() _ = ax.set_title(f"Information-criterion for model selection (training time {fit_time:.2f}s)")