Model Creation and Validation for the Enviornmental Justice Index
Training Random Forest and Geographically Weighted Random Forest Models

Thesis for a Master of Public Health, Epidemiology

Nathan Garcia-Diaz

Brown University, School of Public Health

September 08, 2024

Contents

Statement of Purpose **3**
Defining Package for Geographically Weighted Random Forest Model 3
Defining Hyperparameters 3
Defining: Out of Bag Mean Error Rate 3
Defining: Partially Spatial Nest-Cross Validation Method 4

Outline of Model Building Process **5**

Traditional Random Forest Model **6**
Model Training and Hyperparameter Tuning 6
Model 3 - Exhaustive Grid Search with RMSE as Metric and Partially Spatial Nested Cross Validation 8
RF Model Evaluation 13

Training a Geographically Weighted Random Forest Model **13**
GWRF Model 1 13
GWRF Model 2 15
GWRF Model Evaluation 17

Final Random Forest Model Preformance Metrics and Results **18**
Results for Traditional Random Forest Model 19
Results for Geographically Weighted Random Forest Model 20

Code Appendix **39**

Note: the table of contents acts as in-document hyperlinks

Statement of Purpose

The purpose of the file is to build two final models: a traditional random forest model (RF) and a geographically weighted random forest model (GWFRF). The following two sentence provide a overarching description of the two models. In a RF model, each tree in the forest is built from a different bootstrap sample of the training data, and at each node, a random subset of predictors (features) is considered for splitting, rather than the full set of predictors. A GWRF model expands on this concept by incorporating spatial information by weighting the training samples based on their geographic proximity to the prediction location. The splitting process in a RF model is determined by the mean squared error and in a GWRF is influenced by the spatial weights (i.e., weighted mean squared error), which adjust the contribution of each sample based on its geographic distance. Lastly, the feature importance plots will be generated for the final, and local feature importance plots will also be created.

Defining Package for Geographically Weighted Random Forest Model

Georganos et al (2019) created the `package(SpatialML)`, and subsequently the tuning is made possible by the `SpatialML::grf.bw()` function. The function uses an exhaustive approach (i.e., it tests sequential nearest neighbor bandwidths within a range and with a user defined step, and returns a list of goodness of fit statistics).

Defining Hyperparameters

In James et al 2021, Ch 8.2.2 Random Forests, James et al 2023, Ch 15.2 Definition of Random Forests and Garson 2021, Ch 5 Random Forest, the hyperparameters that are shared between the traditional RF and the geographically-weighted RF models include:

- **Number of randomly selected predictors**: This is the number of predictors (p) considered for splitting at each node. It controls the diversity among the trees. A smaller m leads to greater diversity, while a larger m can make the trees more similar to each other.
  - for regression this defaults to $p/3$, where p is the total of predictor variables
- **Number of trees**: This is the total number of decision trees in the forest (m). More trees generally lead to a more stable and accurate model, but at the cost of increased computational resources and time.
  - for the `randomForest::randomForest()`, this defaults to 500

Additionally, GWRF involves an extra tuning spatial parameters:

- **Bandwidth parameter**: This controls the influence of spatial weights, determining how quickly the weight decreases with distance. A smaller bandwidth means only very close samples have significant influence, while a larger bandwidth allows more distant samples to also contribute to the model.

Defining: Out of Bag Mean Error Rate

In Garson 2021, Ch 5 Random Forest, Garson teaches Random Forest Models by using `randomForest::randomForest()`, and in chapter 5.5.9 (pg. 267), he provides methods for tuning both of these parameters simultaneously using the Out of Bag MSE Error Rates. This value is a measure of the prediction error for data points that were not used in training each tree, hence this value is unique to ensemble methods. It is mathematically expressed as

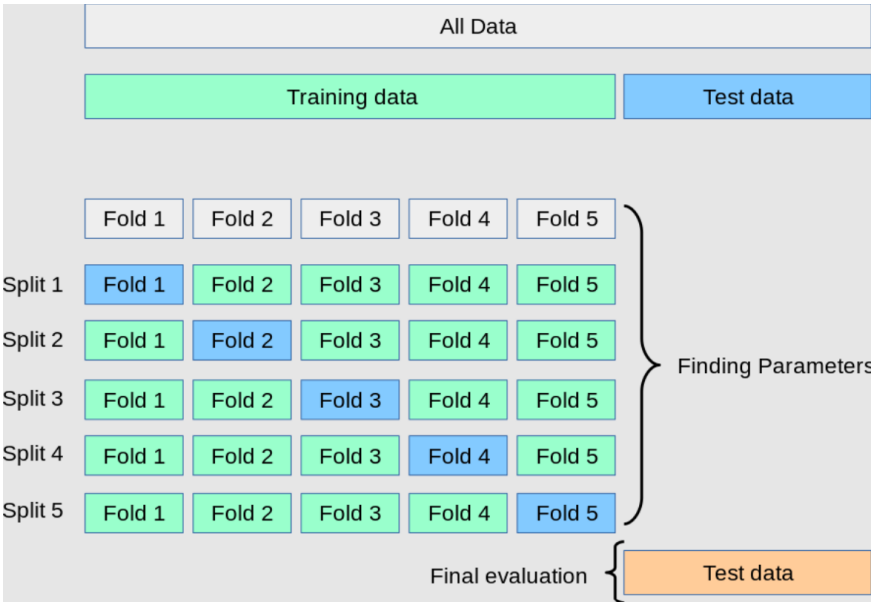
OOB Error Rate $= \frac{1}{n}\Sigma_{i=1}^{N}(y_i - \hat{y}_i^{\text{OOB}})^2$. $\hat{y}_i^{\text{OOB}}$ is the OOB prediction for the i-th observation, which is obtained by averaging the predictions from only those trees that did not include i in their bootstrap sample. To provide a high-level summary, since each tree in a Random Forest is trained on a bootstrap sample (a random sample with replacement) of the data, approximately one-third of the data is not used for training each tree. This subset of data is referred to as the "out-of-bag" data for that tree, and this value is calculated using the data points that were not included in the bootstrap sample used to build each tree. The code in this file has been modified so that cross validation is implemented to ensure consistency across the models, and as such the only difference across models is the metric and the type of nested cross validation being used.

Defining: Partially Spatial Nest-Cross Validation Method

All models will be validated and tuned with a nested cross-validation, a technique used to assess the performance of a model and tuning hyperparameters. It helps to avoid over fitting and provides an unbiased estimate of model performance. A spatial nested cross-validation is a two-level cross-validation procedure designed to evaluate a model's performance and tune its hyperparameters simultaneously. A nested cross-validation is a method that revolves around an outer and liner loop. An example of the workflow include:

- Split the data into "outer_k" folds defined by spatial hierarchical clustering.
- For each fold in the outer loop:
  - Use "outer_k - 1" folds for training.
  - Apply the inner cross-validation on this training set to tune hyperparameters.
  - Evaluate the performance of the model with the selected hyperparameters on the held-out test fold.
- Average the performance metrics across all outer folds to get an overall estimate.

A visual description of the method, which can be in Jian et al (2022) - Rapid Analysis of Cylindrical Bypass Flow Field Based on Deep Learning Model.

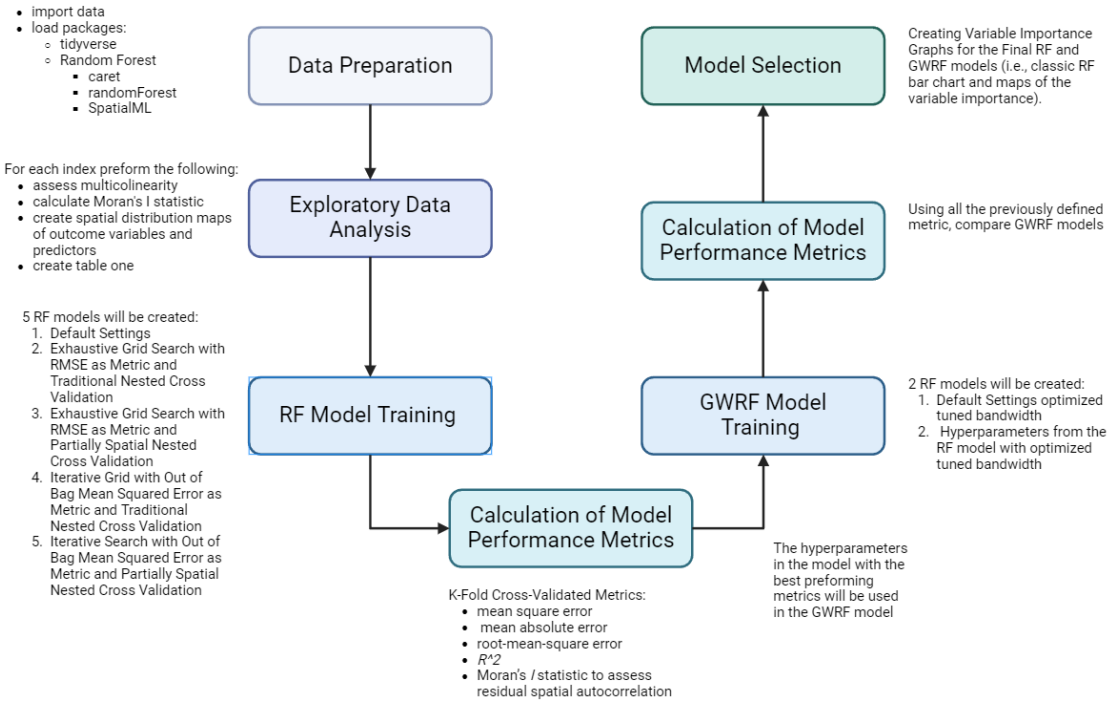


- Outer Cross-Validation Loop:

  - Purpose: To estimate the model's performance on unseen data and provide a more reliable measure of how well the model generalizes to new data.
  - Procedure: The data set is divided into several folds (e.g., 5 or 10). In each iteration, one fold is used as the test set, and the remaining folds are used for training and hyper parameter tuning. Folds are defined by hierarchical clustering. This process is repeated for each fold, ensuring that every data point is used for testing exactly once.
- Inner Cross-Validation Loop:
  - Purpose: To select the best hyperparameters for the model.
  - Procedure: Within each training set from the outer loop, a further cross-validation is performed. This involves splitting the training data into additional folds (e.g., 3 or 5). The model is trained with various hyper parameter combinations on these inner folds, and the performance is evaluated to choose the optimal set of hyperparameters.

Outline of Model Building Process

5 RF models will be built, and they differ based on the different hyperparameters: (1) default settings; (2) Exhaustive Grid Search with RMSE as Metric and Traditional Nested Cross Validation, (3) Exhaustive Grid Search with RMSE as Metric and Partially Spatial Nested Cross Validation, (4)Iterative Grid with Out of Bag Mean Squared Error as Metric and Traditional Nested Cross Validation (i.e., Modified Code from Garson 2021), (5) Iterative Search with Out of Bag Mean Squared Error as Metric and Partially Spatial Nested Cross Validation. For each model, MAE, RMSE, and R^2 will be calculated and the hyperparameters of the best model will continue onto the GWRF. To provide points of comparison in the GWRF, two additional models will be created. Thus, two GWRF models will be created: (1) default *mtry* and *ntrees* with optimized *bandwidth parameter*, and (2) using the previously defined best hyperparameters. The same model evaluation metrics will be compared in addition to calculating the residual autocorrelation.



Traditional Random Forest Model

Model Training and Hyperparameter Tuning

Models will be created and compared at the end of the section.

RF Model 1 - Default Settings

Background: The default settings for the RF model is $mtry = p/3 = 12$, and ntrees = 500, where p is the number of predictors. Nested cross validation is not preformed because the hyperparameters have already been predefined by default.

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 500, mtry = param$mtry, importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 12
##
##           Mean of squared residuals: 0.01051627
##                     % Var explained: 83.44
```

Model 2 - Exhaustive Grid Search with RMSE as Metric and Traditional Nested Cross Validation

Background: To preform an exhaustive Grid Search, Brownlee (2020) created a custom function that preforms the grid search. This function checks every combination of *mtry* and *ntree* values determines the final values with RMSE.

Table 1: Model 2 - Traditional Cross Validation: Hyperparametyer Tuning and Performance Metrics

Fold	Tuned_mtry	Tuned_ntree	RMSE	MAE	R_squared
1	12	400	0.139	0.115	0.713
2	23	150	0.124	0.102	0.800
3	24	200	0.128	0.100	0.757
4	29	150	0.139	0.107	0.700
5	7	100	0.120	0.099	0.769

Model 2 has hyperparameters set to $mtry = 7$, and $ntrees = 100$.

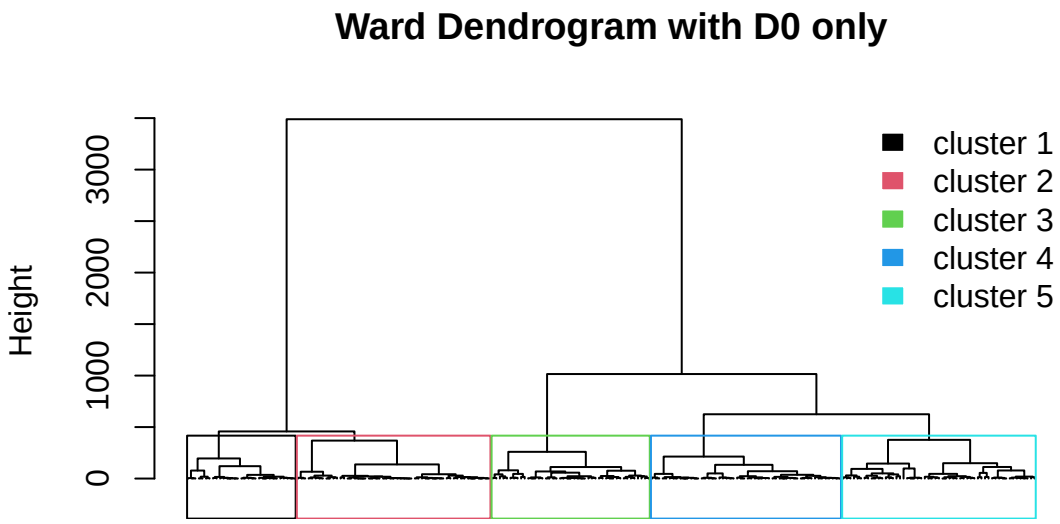
Model 3 - Exhaustive Grid Search with RMSE as Metric and Partially Spatial Nested Cross Validation

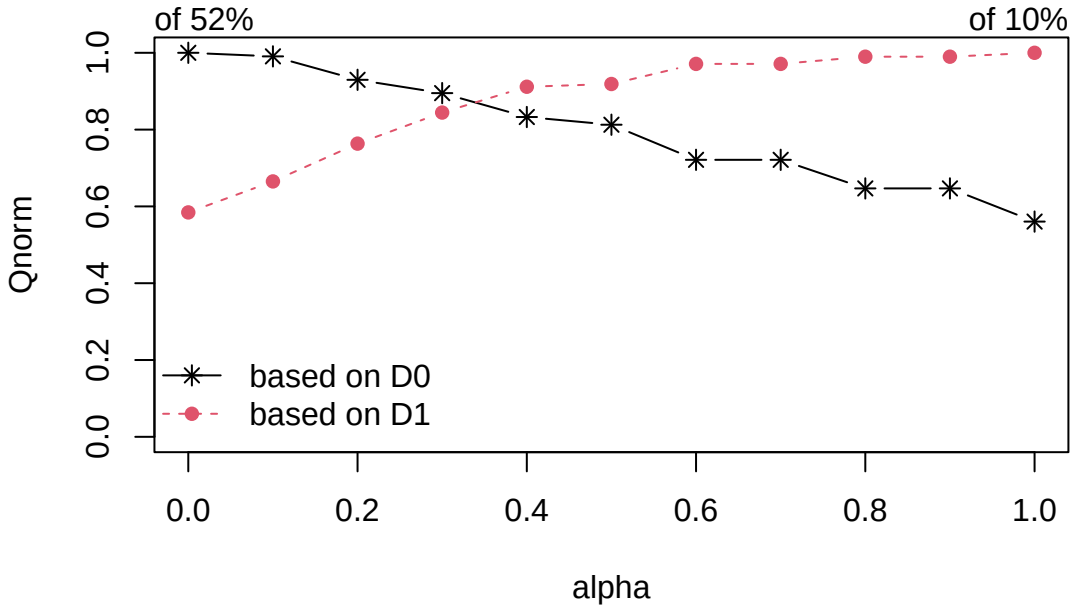
Background: Preforms the same task as model 2 (i.e., tune hyperparameters with an exhaustive grid search), however where this model differs is occurs based on the nested cross validation. The outer loop is defined by `ClustGeo` package, which implements hierarchical clustering with soft contiguity constraint. The main arguments of the function are:

- a matrix D0 with the dissimilarities in the "feature space" (here EJI variables for instance).
- a matrix D1 with the dissimilarities in the "constraint" space (here a matrix of geographical dissimilarities).
- a mixing parameter alpha between 0 an 1. The mixing parameter sets the importance of the constraint in the clustering procedure.
- a scaling parameter scale with a logical value. If TRUE the dissimilarity matrices D0 and D1 are scaled between 0 and 1 (that is divided by their maximum value).

For more information on the package and the code implement please visit the following link Introduction to ClustGeo.

```
## [1] "\nYou cut the dendrogram horizontally at a level that \nrepresents a reasonable trade-
```





Ward Dendrogram eluded to 5 clusters because the trade off is balances (i.e., trade-off between the number of clusters and the within-cluster similarity, which is achieved by cutting the goal to look for large vertical gaps between successive merges), and all groups seem similar in size. Additionally, the qnorm v. alpha plot suggested a mixing parameter of 0.3.

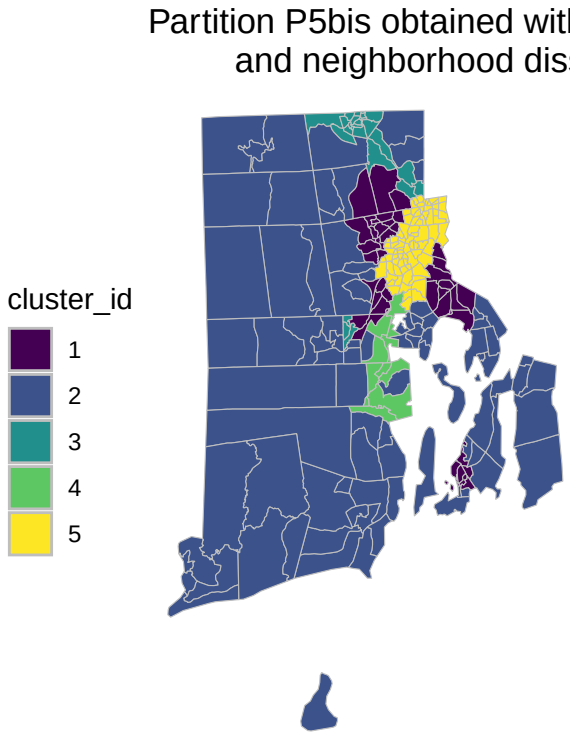


Table 2: Model 3 - Partially Spatial Cross Validation: Hyperparametyer Tuning and Performance Metrics

Fold	Tuned_mtry	Tuned_ntree	RMSE	MAE	Rsquared
1	21	250	0.096	0.077	0.821
2	8	250	0.122	0.098	0.788
3	22	400	0.089	0.072	0.891
4	7	150	0.110	0.083	0.867
5	21	100	0.112	0.095	0.829

Model 3 has hyperparameters set to $mtry = 22$, and $ntrees = 400$.

Model 4 - Iterative Grid with Out of Bag Mean Squared Error as Metric and Traditional Nested Cross Validation

In the inner loop, this code snippet is designed to optimize the hyperparameters *mtry* and *ntree* in a Random Forest model and by examining the OOB MSE across these combinations, the code identifies which parameters yield the lowest error, helping to optimize the Random Forest model. This inner loop was designed by Garson 2021, Ch 5 Random Forest. This is how the code meets this tunes the parameters:

- Hyper parameter Search Strategy:
  - Iterative Search for *mtry*: The `mtry_iter` function generates an iterable sequence of *mtry* values, starting from 1 up to the number of predictors, incremented by a step factor. This allows the code to explore different numbers of predictors used at each split in the trees.
  - Specification of *ntree* Values: A predefined vector *vntree* contains different values for the number of trees to be grown in the forest. This allows the code to assess how the number of trees impacts the model performance.
- Traditional Nested Cross-Validation:
  - Outer loop: data is randomly split into 5 folds (i.e., 80% training 20% testing split), thus ensuring all osbervations have been used in training and testing.
  - Inner Loop: Within each inner loop, the function calculations the error across the provided hyperparameter combinations. The tune function performs a grid search over the specified *mtry* values and the maximum number of trees specified in *vntree.* For each combination, the function trains a Random Forest model and calculates the OOB error rate, MSE since y is continuous.
  - Model Training: After identifying the optimal mtry and ntree for each outer fold, a final Random Forest model is trained using these parameters on the entire training subset. The model's performance is then assessed on the corresponding test fold.

Table 3: Model 4 - Traditional Cross Validation: Hyperparametyer Tuning and Performance Metrics

Fold	Best_mtry	Best_ntree	Test_Error	RMSE	MAE	R_squared
1	23	150	0.010	0.102	0.088	0.809
2	23	150	0.009	0.094	0.077	0.870
3	6	400	0.013	0.114	0.089	0.791
4	5	700	0.013	0.115	0.094	0.807
5	27	150	0.014	0.119	0.092	0.779

Model 4 has hyperparameters set to $mtry = 23$, and $ntrees = 150$.

Model 5 - Iterative Search with Out of Bag Mean Squared Error as Metric and Partially Spatial Nested Cross Validation

Table 4: Model 5 - Partially Spatial Cross Validation: Hyperparametyer Tuning and Performance Metrics

Fold	Best_mtry	Best_ntree	Test_Error	RMSE	MAE	R_squared
1	9	400	0.012	0.111	0.087	0.750
2	13	100	0.015	0.122	0.099	0.809
3	14	150	0.010	0.101	0.075	0.818
4	14	150	0.008	0.092	0.080	0.878
5	9	700	0.010	0.101	0.079	0.836

Model 5 has hyperparameters set to $mtry = 14$, and $ntrees = 150$.

RF Model Evaluation

Table 5: Performance Metrics for Each Model

Model	Best_mtry	Best_ntree	Test_Error	RMSE	MAE	R_squared
1	5	500	NA	0.105	0.085	0.835
2	7	100	NA	0.120	0.099	0.769
3	22	400	NA	0.089	0.072	0.891
4	23	150	0.009	0.094	0.077	0.870
5	14	150	0.008	0.092	0.080	0.878

Training a Geographically Weighted Random Forest Model

GWRF Model 1

This model has hyperparameters defined with *mtry* and *trees* by the default, and optimized *bandwidth*.

```
## Ranger result
##
## Call:
##  ranger(rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr + e_npl +      e_tri + e_tsd + e_rmp +
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      240
## Number of independent variables:  36
## Mtry:                             12
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.0109019
## R squared (OOB):                  0.8290535
##     e_ozone        e_pm     e_dslpm     e_totcr       e_npl       e_tri
##  0.04136486  0.21954436  0.18069919  1.16569825  0.16542656  2.00077366
##       e_tsd       e_rmp      e_coal      e_lead      e_park    e_houage
##  0.02715322  0.29746194  0.00000000  0.00000000  0.04864063  0.09932929
##    e_wlkind      e_rail      e_road    e_airprt    e_impwtr   ep_minrty
##  0.09889609  0.48993411  0.19441835  0.09497684  0.08934265  0.88383071
##   ep_pov200   ep_nohsdp    ep_unemp   ep_renter   ep_houbdn  ep_uninsur
##  0.87035332  1.21914494  0.09525797  0.84669758  0.33715106  0.16371822
##    ep_noint    ep_age65    ep_age17   ep_disabl   ep_limeng   ep_mobile
##  0.49722702  0.09393216  0.08644416  0.17644139  1.29600300  0.04511835
##   ep_groupq   ep_bphigh   ep_asthma   ep_cancer    ep_mhlth ep_diabetes
##  0.12096842  0.23610811  0.53490669  0.10258044  0.15787913  2.09727927
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.280563 -0.078433  0.016074  0.008437  0.083182  0.444150
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.0929508 -0.0055250  0.0008768  0.0011193  0.0089035  0.0875191
```

```
##                     Min        Max        Mean         StD
## e_ozone     1.132016e-04 0.05269154 0.009382782 0.009005878
## e_pm        2.550884e-03 0.07507584 0.017436053 0.013121648
## e_dslpm     2.006259e-03 0.10253525 0.026617545 0.022233799
## e_totcr     2.137893e-03 0.18382751 0.035115107 0.036200081
## e_npl       0.000000e+00 0.32031910 0.032445345 0.058590481
## e_tri       7.054309e-05 0.24557787 0.067606790 0.063773028
## e_tsd       0.000000e+00 0.02968211 0.004255862 0.007723482
## e_rmp       1.818377e-04 0.29084768 0.032270942 0.055254034
## e_coal      0.000000e+00 0.00000000 0.000000000 0.000000000
## e_lead      0.000000e+00 0.00000000 0.000000000 0.000000000
## e_park      0.000000e+00 0.07452369 0.015871405 0.016398027
## e_houage    2.898853e-03 0.08696559 0.020288387 0.016820822
## e_wlkind    4.915058e-03 0.06122151 0.018474726 0.010489002
## e_rail      6.304736e-04 0.07379972 0.017991987 0.020165970
## e_road      7.161204e-05 0.11404621 0.022409891 0.023476440
## e_airprt    0.000000e+00 0.22933330 0.021880614 0.039834674
## e_impwtr    3.508172e-03 0.13015411 0.028195056 0.025968296
## ep_minrty   9.405772e-03 0.16546698 0.060064634 0.036206367
## ep_pov200   1.542195e-02 0.38705278 0.092570048 0.080595209
## ep_nohsdp   2.051034e-02 0.41885615 0.174277469 0.108370073
## ep_unemp    5.953104e-03 0.15145354 0.033400925 0.027888545
## ep_renter   5.273553e-03 0.22162016 0.058641075 0.042803101
## ep_houbdn   7.697587e-03 0.15940607 0.054206159 0.032010779
## ep_uninsur  4.457254e-03 0.27469959 0.046151283 0.037448958
## ep_noint    6.513784e-03 0.27943432 0.108076700 0.071936088
## ep_age65    1.747754e-03 0.30269678 0.040199784 0.067681096
## ep_age17    1.879601e-03 0.11188990 0.029795946 0.019348040
## ep_disabl   5.370925e-03 0.21184166 0.056110249 0.035511442
## ep_limeng   4.682103e-03 0.28543615 0.062268600 0.050866598
## ep_mobile   4.142951e-05 0.04471641 0.007761010 0.008713201
## ep_groupq   2.056840e-03 0.06683322 0.019969326 0.015255903
## ep_bphigh   7.214481e-03 0.31650453 0.079345351 0.079114936
## ep_asthma   6.823074e-03 0.43927617 0.089875902 0.105678019
## ep_cancer   2.571483e-03 0.18430587 0.027652917 0.036854073
## ep_mhlth    5.377774e-03 0.27167655 0.073742978 0.073366717
## ep_diabetes 1.024561e-02 0.39782023 0.167664625 0.105326830
```

The final model hyperparameters have been set to $bandwidth = 49$, $mtry = 5$, and $ntrees = 500$.

GWRF Model 2

This model has hyperparameters defined with *mtry* and *trees* by previously defined best preforming Random Forest Model, and optimized *bandwith.*

```
## Ranger result
##
## Call:
##  ranger(rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr + e_npl +      e_tri + e_tsd + e_rmp +
##
## Type:                             Regression
## Number of trees:                  400
## Sample size:                      240
## Number of independent variables:  36
## Mtry:                             22
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.01051481
## R squared (OOB):                  0.8351232
##     e_ozone        e_pm     e_dslpm     e_totcr       e_npl       e_tri
##  0.02352145  0.17742667  0.13265762  1.29728828  0.24378993  2.59123205
##       e_tsd       e_rmp      e_coal      e_lead      e_park    e_houage
##  0.02452246  0.33005248  0.00000000  0.00000000  0.03526659  0.09630006
##    e_wlkind      e_rail      e_road    e_airprt    e_impwtr   ep_minrty
##  0.07684574  0.47253038  0.19613993  0.09451683  0.08928839  0.58933411
##   ep_pov200   ep_nohsdp    ep_unemp   ep_renter   ep_houbdn  ep_uninsur
##  0.72366716  0.80172685  0.08540884  0.84593858  0.19562116  0.10060827
##    ep_noint    ep_age65    ep_age17   ep_disabl   ep_limeng   ep_mobile
##  0.30565295  0.08144511  0.07439766  0.19988770  1.40124421  0.04105413
##   ep_groupq   ep_bphigh   ep_asthma   ep_cancer    ep_mhlth ep_diabetes
##  0.13866373  0.19818009  0.32693352  0.08778004  0.09695416  2.96718896
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.328800 -0.073195  0.010022  0.003254  0.075309  0.314807
##       Min.    1st Qu.     Median       Mean    3rd Qu.       Max.
## -0.0969252 -0.0069139  0.0006291  0.0006154  0.0076119  0.0824595
##                      Min        Max        Mean         StD
## e_ozone     4.281836e-05 0.03603583 0.006944291 0.007449317
## e_pm        1.626216e-03 0.09084060 0.014609496 0.014725043
## e_dslpm     1.163799e-03 0.09728328 0.024214352 0.020980293
## e_totcr     1.660234e-03 0.25743338 0.030911648 0.040400840
## e_npl       0.000000e+00 0.40018883 0.037897573 0.075645706
## e_tri       1.107169e-05 0.34631427 0.069597597 0.072643264
## e_tsd       0.000000e+00 0.03563938 0.004408077 0.008398708
## e_rmp       0.000000e+00 0.45876898 0.037275509 0.084281493
## e_coal      0.000000e+00 0.00000000 0.000000000 0.000000000
## e_lead      0.000000e+00 0.00000000 0.000000000 0.000000000
## e_park      0.000000e+00 0.05547940 0.013863542 0.014292433
```

```
## e_houage    2.140380e-03 0.09725865 0.016793001 0.016979854
## e_wlkind    2.182337e-03 0.05339576 0.015303367 0.010268259
## e_rail      4.249273e-04 0.08349197 0.015043648 0.017364139
## e_road      2.120557e-05 0.11043410 0.018548103 0.022057341
## e_airprt    0.000000e+00 0.25813772 0.021729947 0.046198018
## e_impwtr    1.384512e-03 0.15053368 0.026111528 0.027887707
## ep_minrty   9.046875e-03 0.12974740 0.043346355 0.025395526
## ep_pov200   1.826498e-03 0.60149787 0.092473011 0.115512329
## ep_nohsdp   1.471098e-02 0.59930656 0.204845801 0.156304258
## ep_unemp    3.537878e-03 0.13545883 0.026240516 0.023465281
## ep_renter   2.948681e-03 0.29867724 0.050644369 0.053148291
## ep_houbdn   6.905513e-03 0.13644747 0.044884847 0.030753028
## ep_uninsur  2.724157e-03 0.30042945 0.043149770 0.038653027
## ep_noint    1.766462e-03 0.34447945 0.102786385 0.081567618
## ep_age65    6.859107e-04 0.42362237 0.042112160 0.086246856
## ep_age17    8.770283e-04 0.09719170 0.027917088 0.020390230
## ep_disabl   2.001671e-03 0.25046254 0.048605416 0.035796184
## ep_limeng   3.591536e-03 0.36468356 0.057312246 0.053539017
## ep_mobile   1.775036e-06 0.04567469 0.006515326 0.008349462
## ep_groupq   8.418923e-04 0.06698714 0.018573759 0.016021071
## ep_bphigh   4.270986e-03 0.38538318 0.081427154 0.098240920
## ep_asthma   1.708577e-03 0.61948112 0.092566366 0.143968159
## ep_cancer   1.457209e-03 0.19069659 0.024739161 0.037396411
## ep_mhlth    1.537052e-03 0.34532599 0.070808352 0.081803740
## ep_diabetes 9.076525e-03 0.52620051 0.187404534 0.144656170
```

The final model hyperparameters have been set to $bandwidth = 48$, $mtry = 22$, and $ntrees = 400$.

GWRF Model Evaluation

The models both preform nearly identically because the hyperparameters preform nearly identically. Therefore, the model define by the previous traditional random forest model.

Table 6: Performance Metrics for Each Model

Model	bw	mtry	ntree	MAE	RMSE	R_Squared
6	49	5	500	0.084	0.104	0.828
7	48	22	400	0.082	0.103	0.834

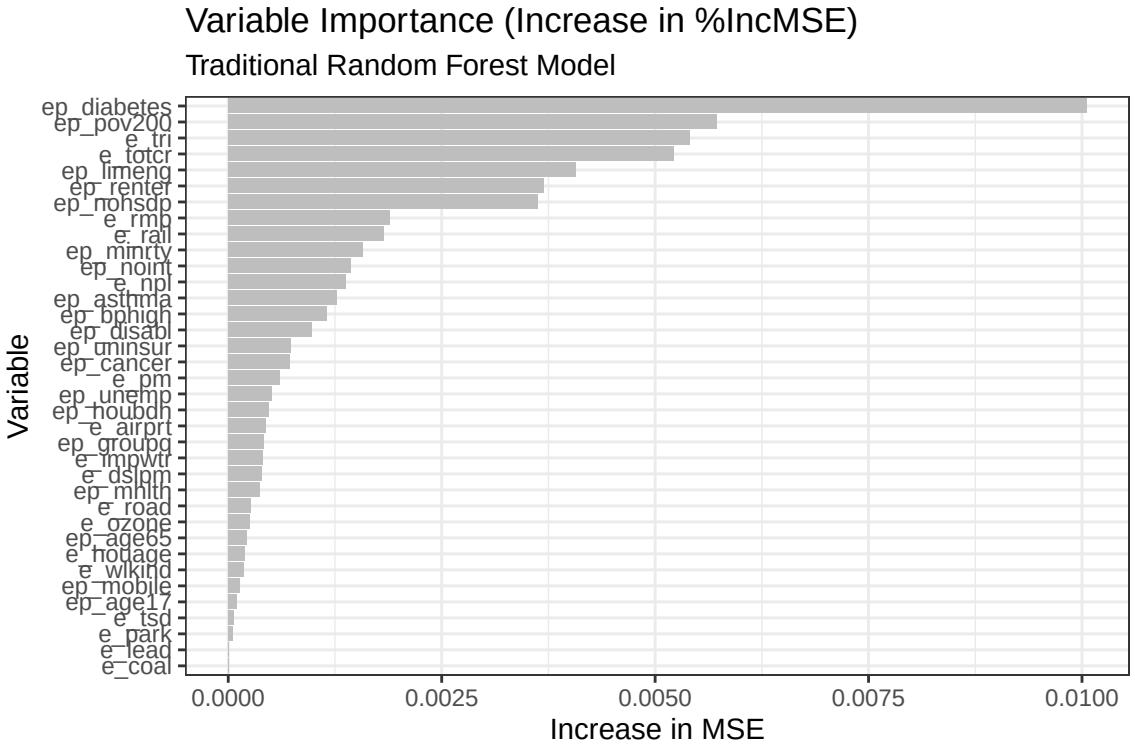
Final Random Forest Model Preformance Metrics and Results

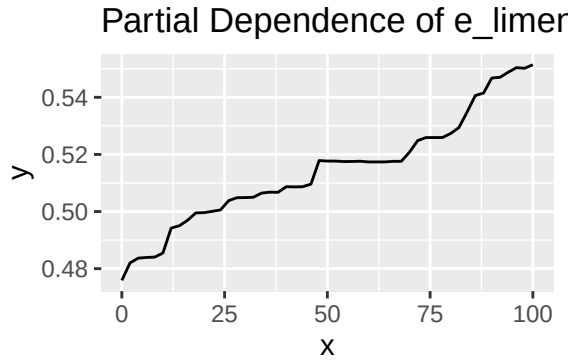
Table 7: Performance Metrics for Each Model

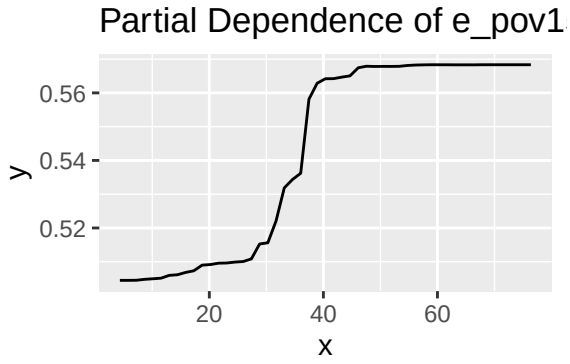
Model	Bandwidth	Best_mtry	Best_ntree	RMSE	MAE	R_squared
3	NA	22	400	0.089	0.072	0.891
7	48	22	400	0.082	0.103	0.834

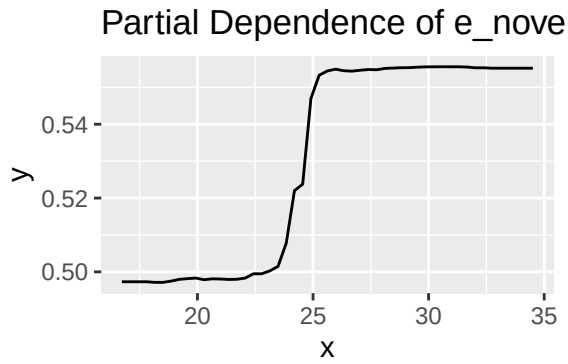
Results for Traditional Random Forest Model

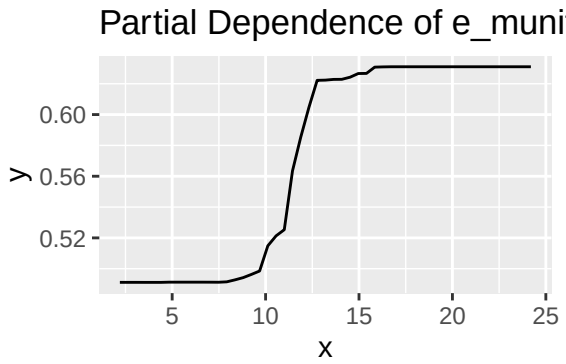
Partial Dependency Maps are coming forward.



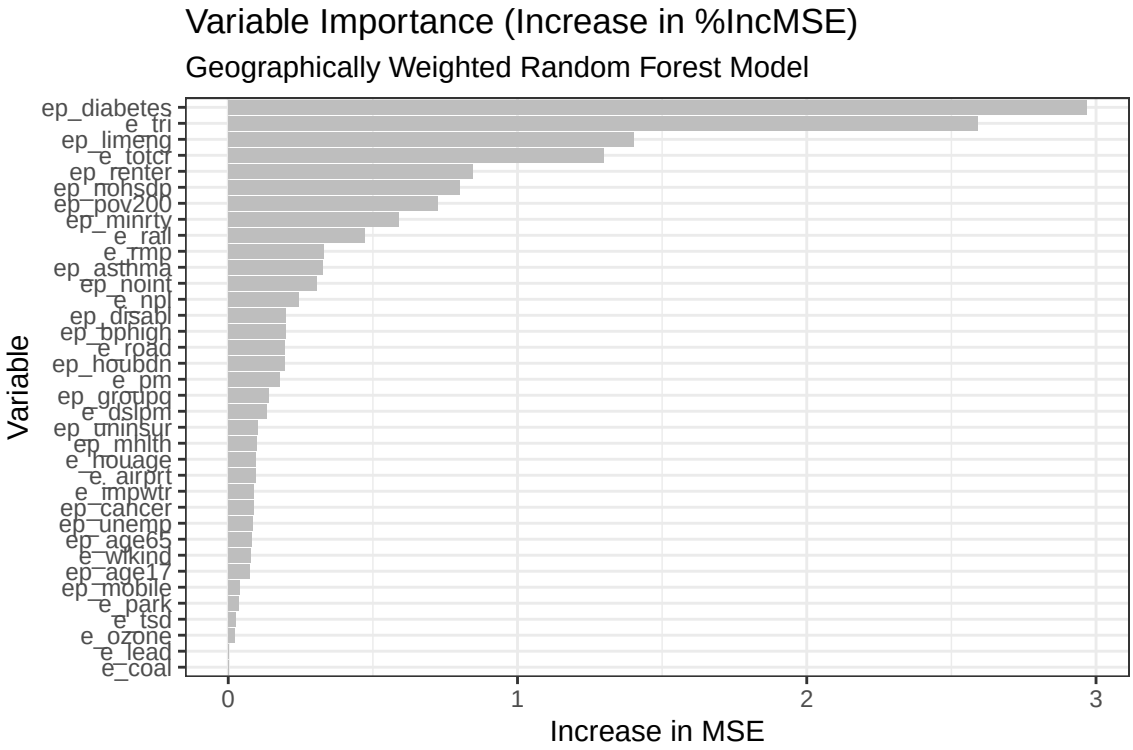








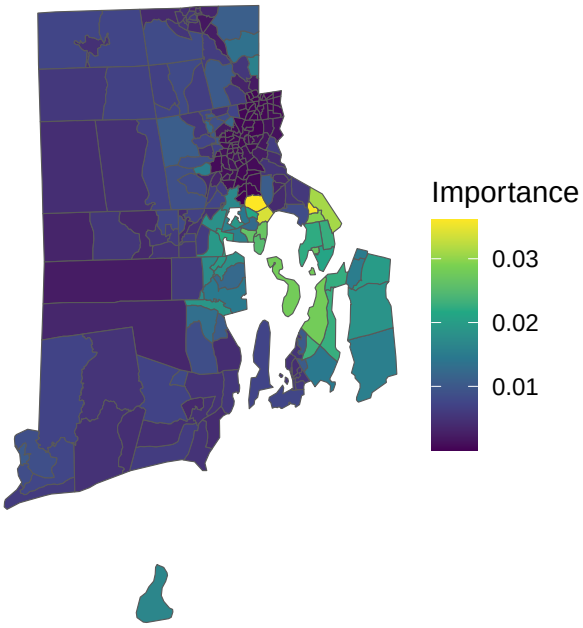
Results for Geographically Weighted Random Forest Model



Local Variable Importance of Enviornmental Burden

Distibution of Annual mean days above O3 regulatory standard (3 yr avg)

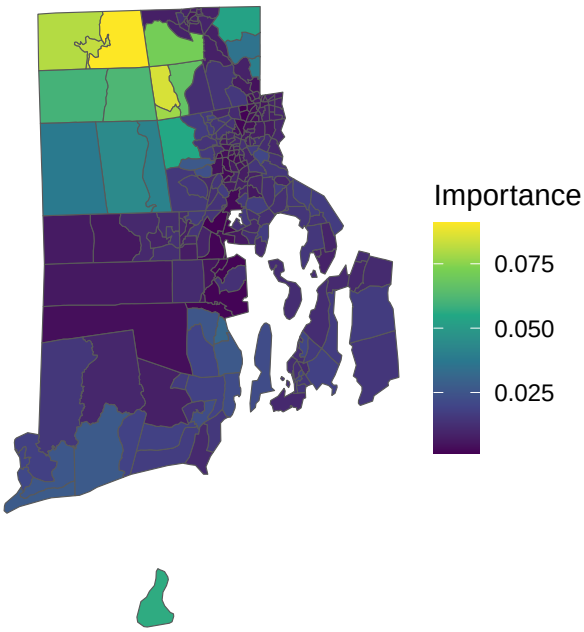
e_ozone



EPA AQS 2014−2016

Distibu tion of Annual mean days above PM2.5 regulatory standard (3 yr avg)

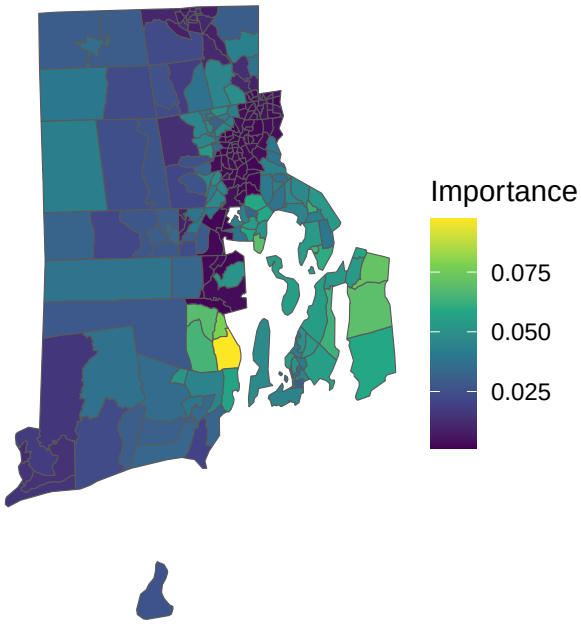
e_pm



EPA AQS 2014−2016

Distibution of Ambient concentrations of Diesel (PM/m3)

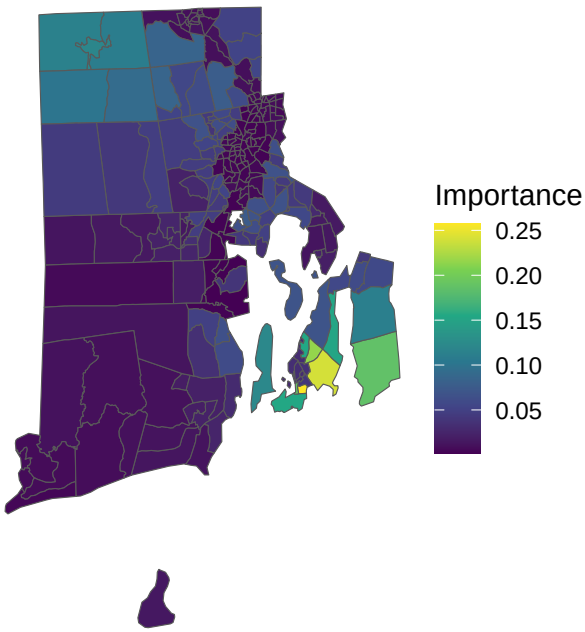
e_dslpm



EPA AQS 2014−2016

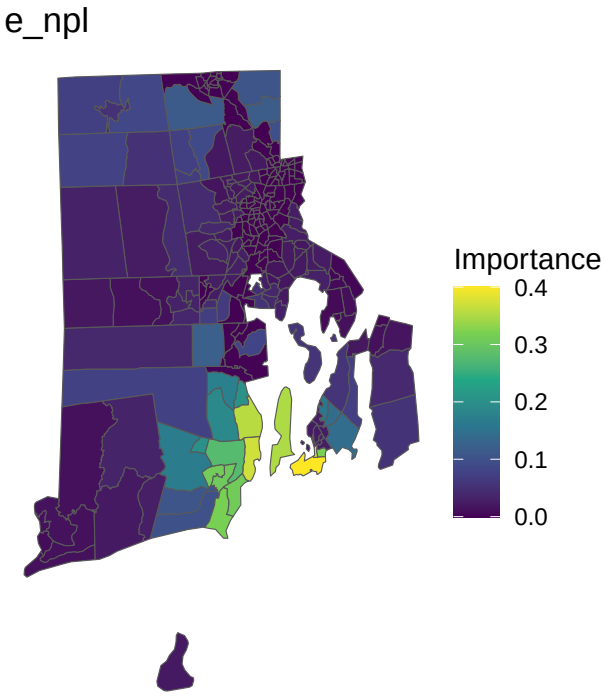
Distibution of Probability of Contracting Cancer (assuming continuous exposure)

e_totcr



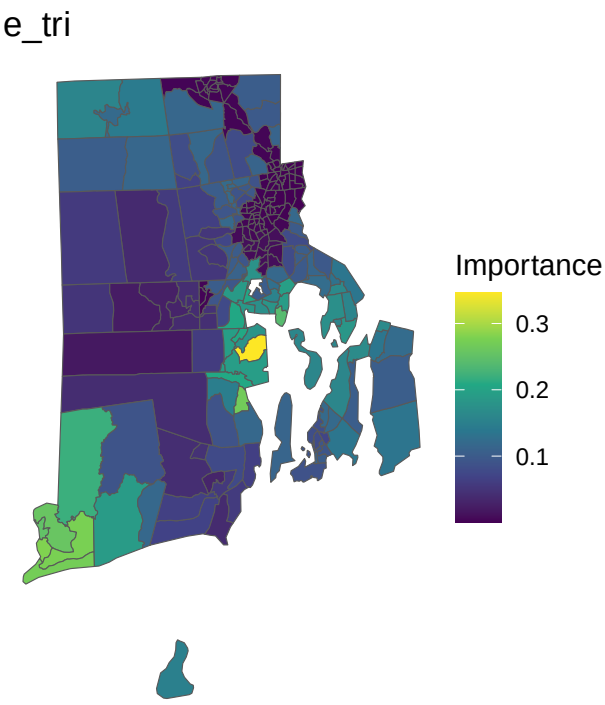
EPA AQS 2014−2016

Distibution of Proportion of Tract's Area within 1−mi buffer of EPA National Priority List Site



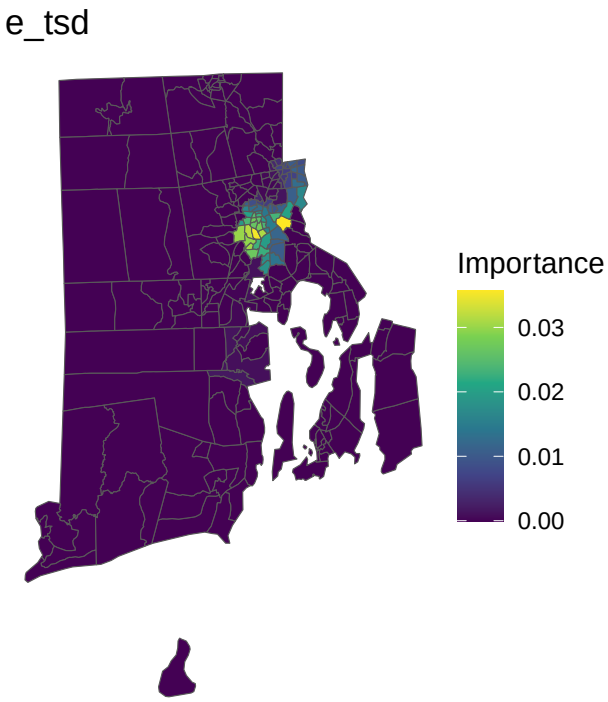
EPA Geospatial Download

Distibution of Proportion of Tract's Area within 1−mi buffer of EPA Toxic Release Inventory Site



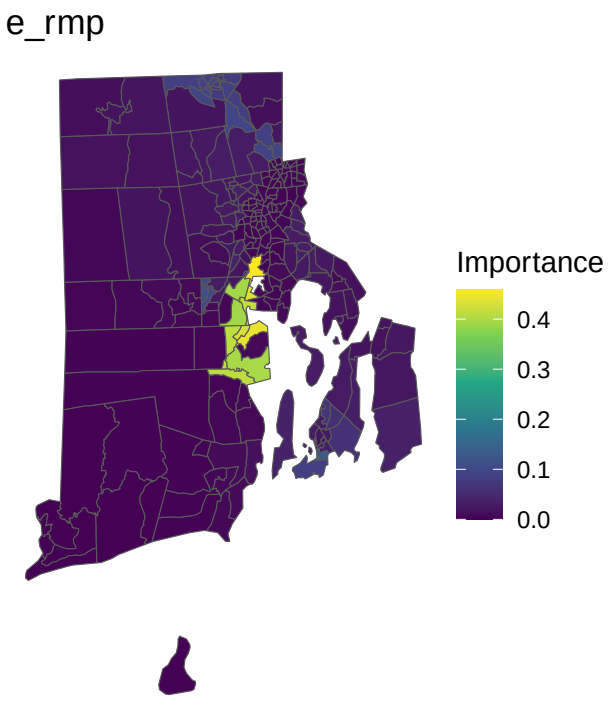
EPA Geospatial Download

Distibution of Proportion of Tract's Area within 1−mi buffer of EPA Treatment, Storage, and Disposal site



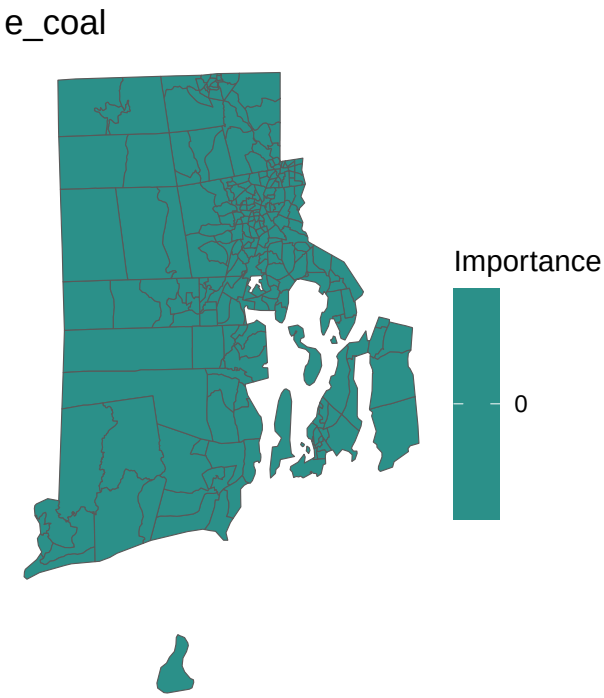
EPA Geospatial Download

Distibution of Proportion of Tract's Area within 1−mi buffer of EPA risk management plan site



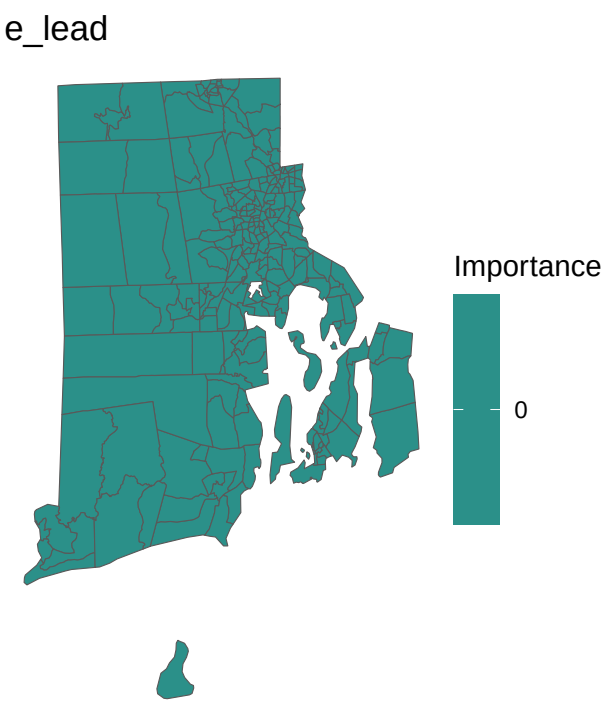
EPA Geospatial Download

Distibution of Proportion of tract's area within 1−mi buffer of coal mines



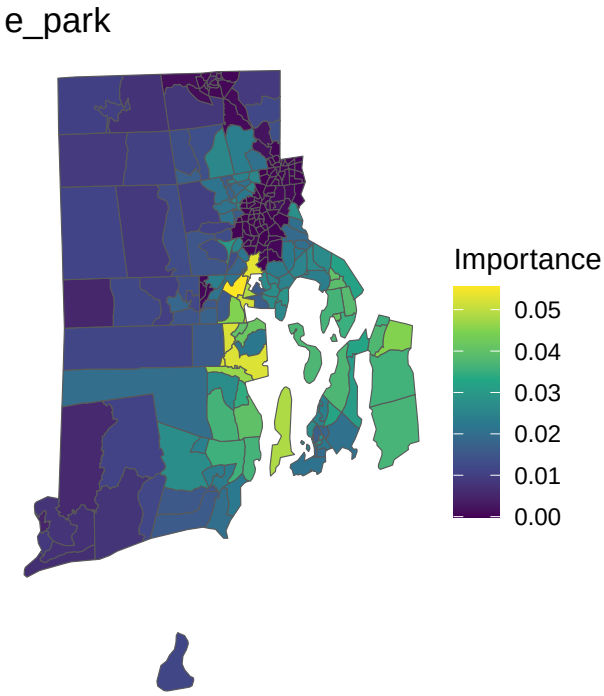
EPA Geospatial Download

Distibution of Proportion of tract's area within 1−mi buffer of lead mines



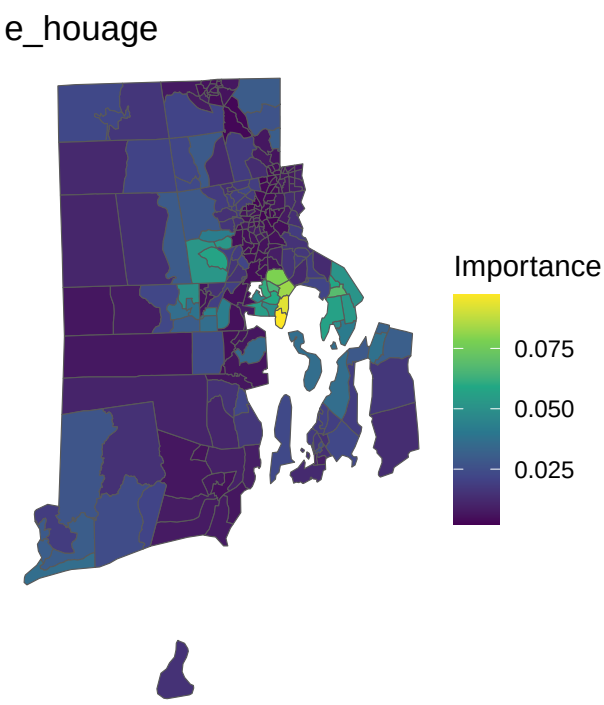
EPA Geospatial Download

Distibution of Proportion of tract's area within 1−mi buffer of green space



2020 TomTom MultiNet Enterpirse Dataset

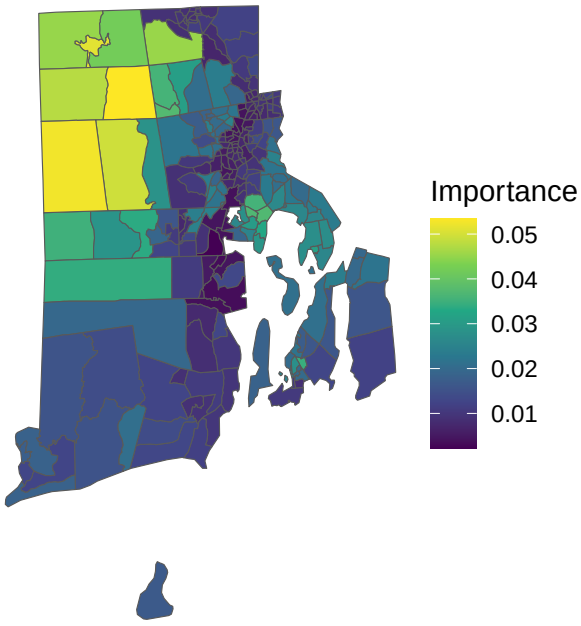
Distibution of Percentage of houses built pre 1980 (lead exposure)



No Citation

Distibution of Walkability Values

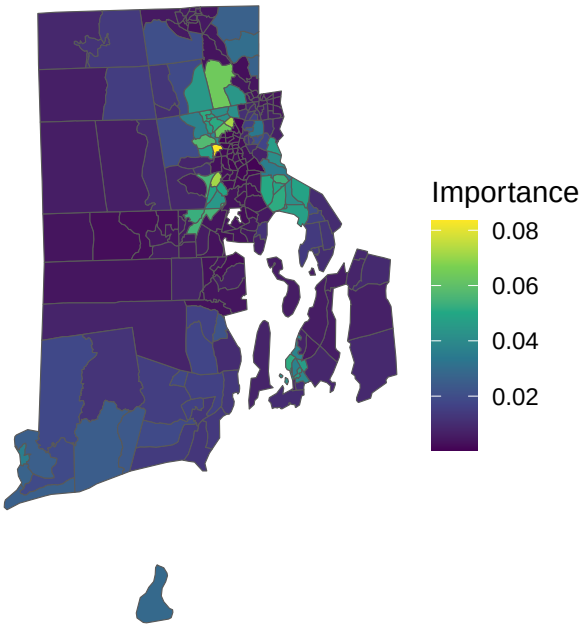
e_wlkind



EPA Walkability Inex

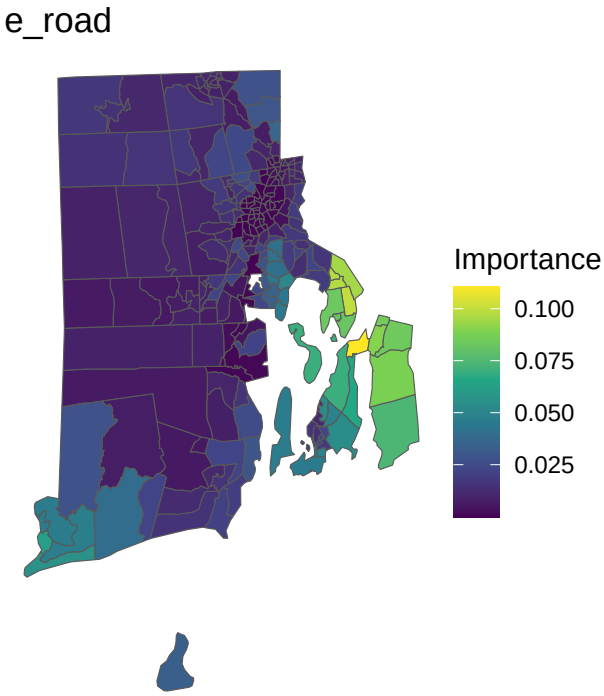
Distibution of Proportion of tract's area within 1−mi buffer of railroad

e_rail



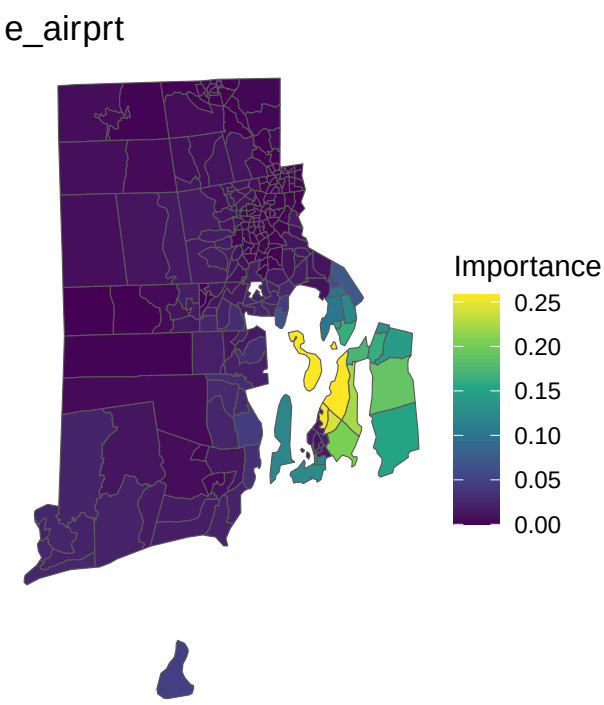
EPA Walkability Inex

Distibution of Proportion of tract's area within 1−mi buffer of high volume road or highway



EPA Walkability Inex

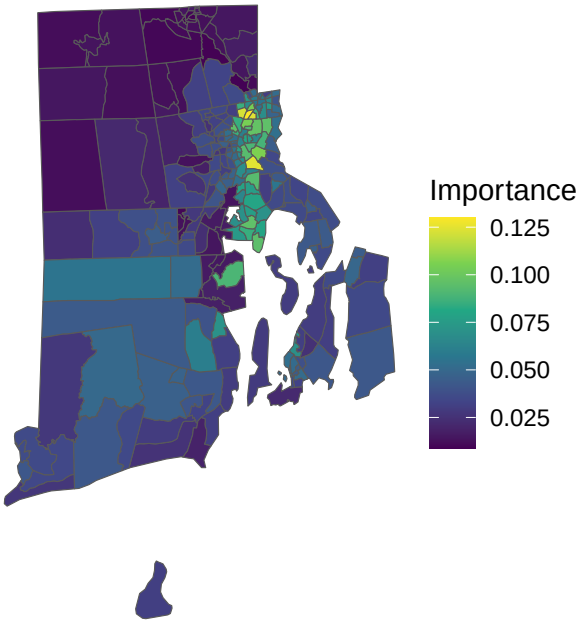
Distibution of Proportion of tract's area within 1−mi buffer of airport



EPA Walkability Inex

Distibution of Percentage of Minority Persons

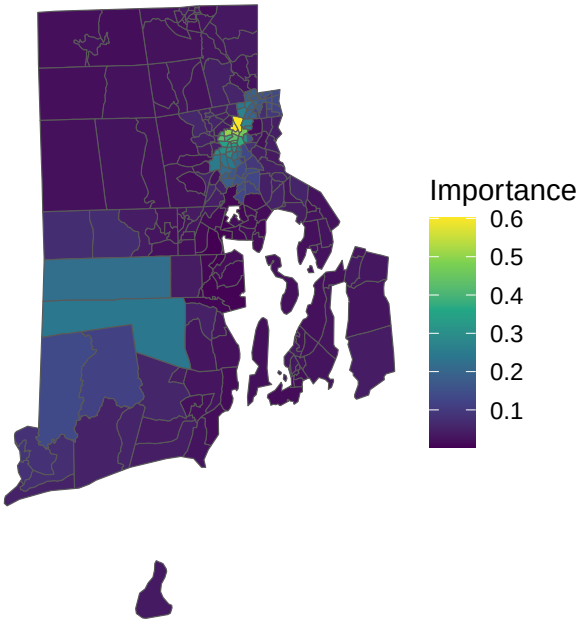
ep_minrty



A.C.S. 2015−2019

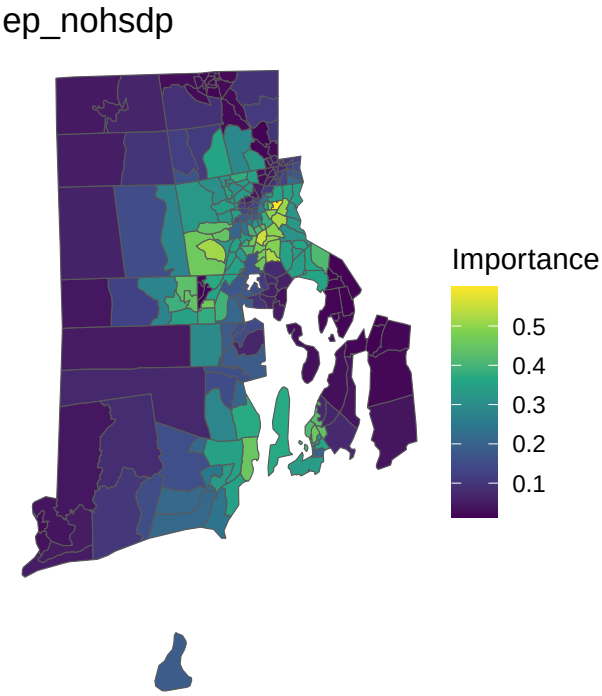
Distibution of Percentage of Persons Below 200% Poverty

ep_pov200



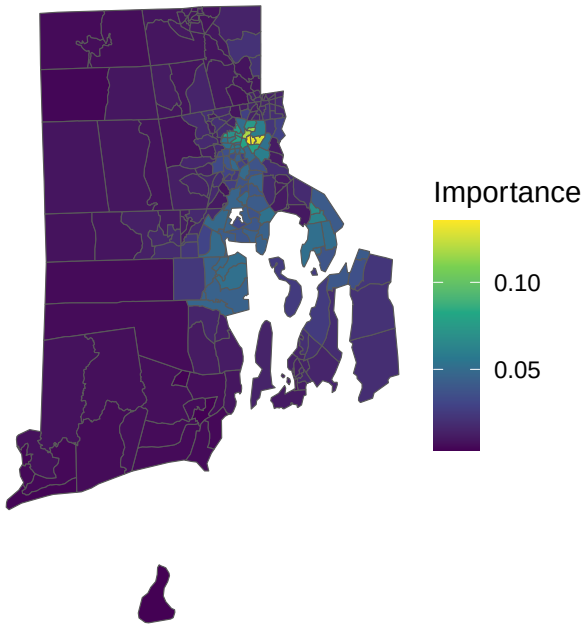
A.C.S. 2015−2019

Distibution of Percentage of Persons with no High School Diploma (age 25+)



A.C.S. 2015−2019

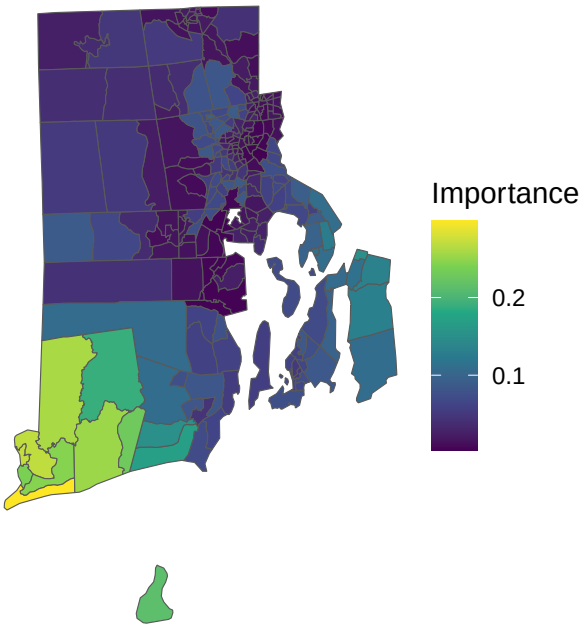
Distibution of Percentage of Unemployed Persons



A.C.S. 2015−2019

Distibution of Percentage of Renters

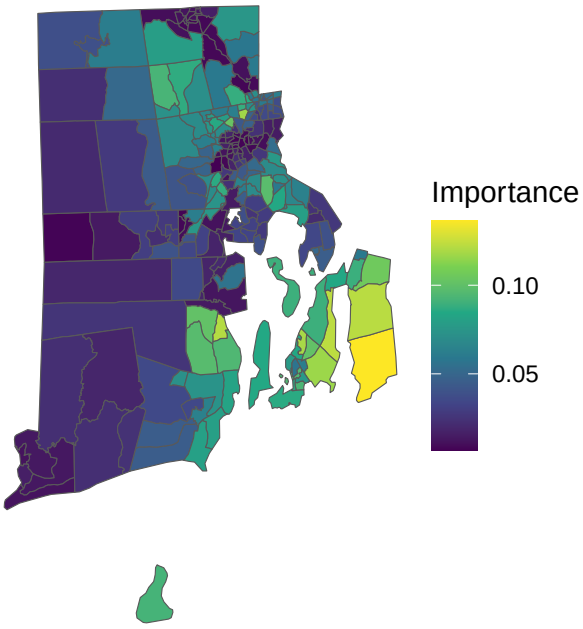
ep_renter



A.C.S. 2015−2019

Distibution of Percentage of households that make less than $75,000

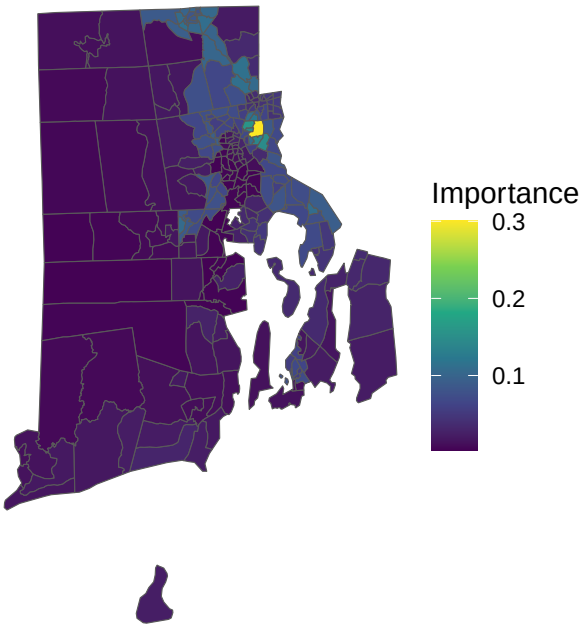
ep_houbdn



A.C.S. 2015−2019

Distibution of Percentage of Unisured Persons

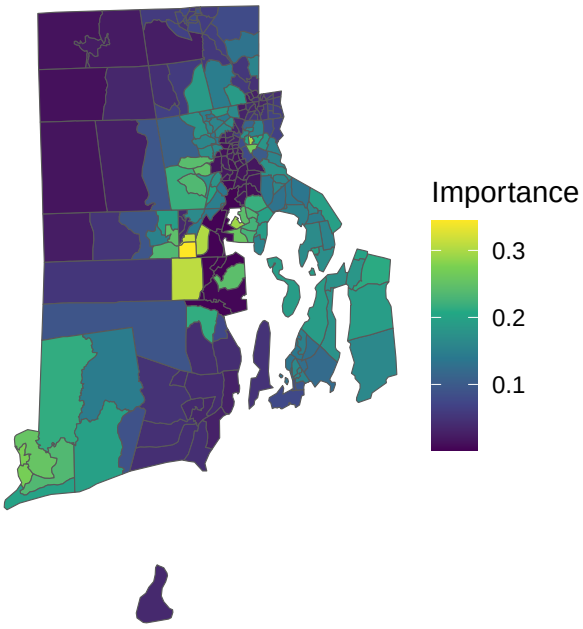
ep_uninsur



A.C.S. 2015−2019

Distibution of Percentage of Persons Without Internet

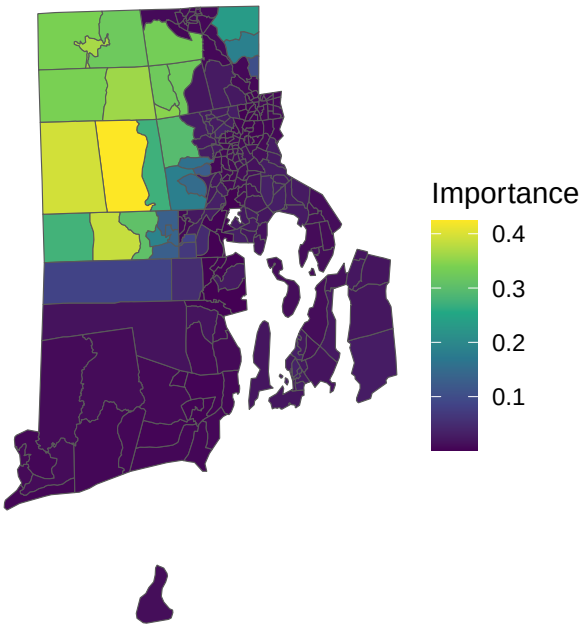
ep_noint



A.C.S. 2015−2019

Distibution of Percentage of persons aged 65 and older

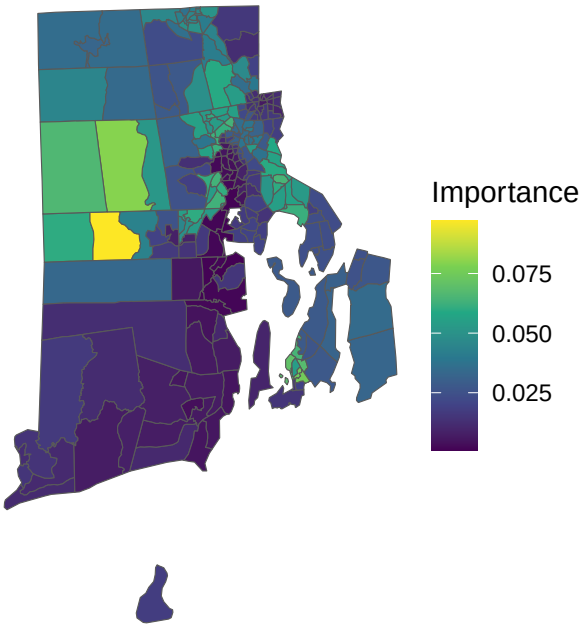
ep_age65



A.C.S. 2015−2019

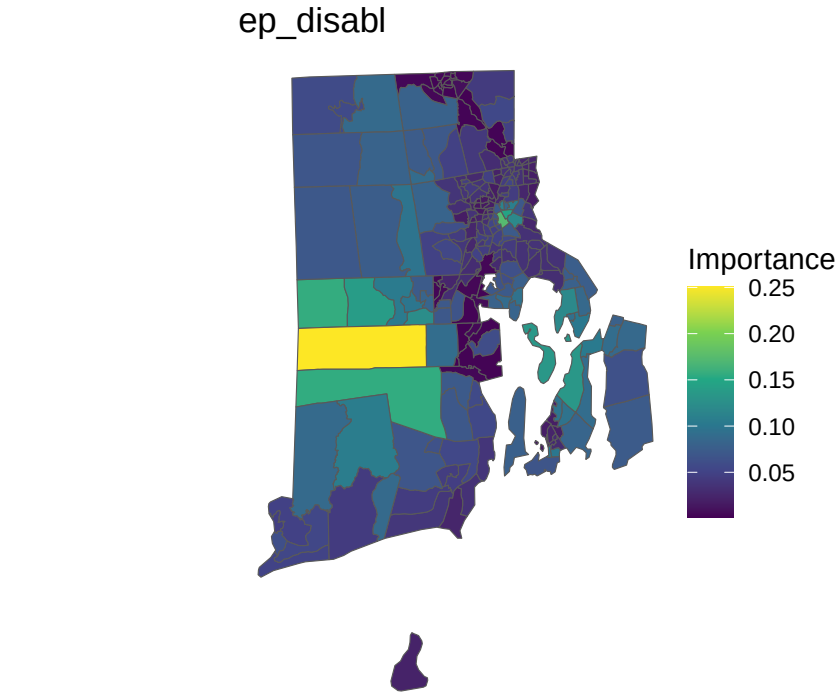
Distibution of Percentage of persons aged 17 and younger

ep_age17



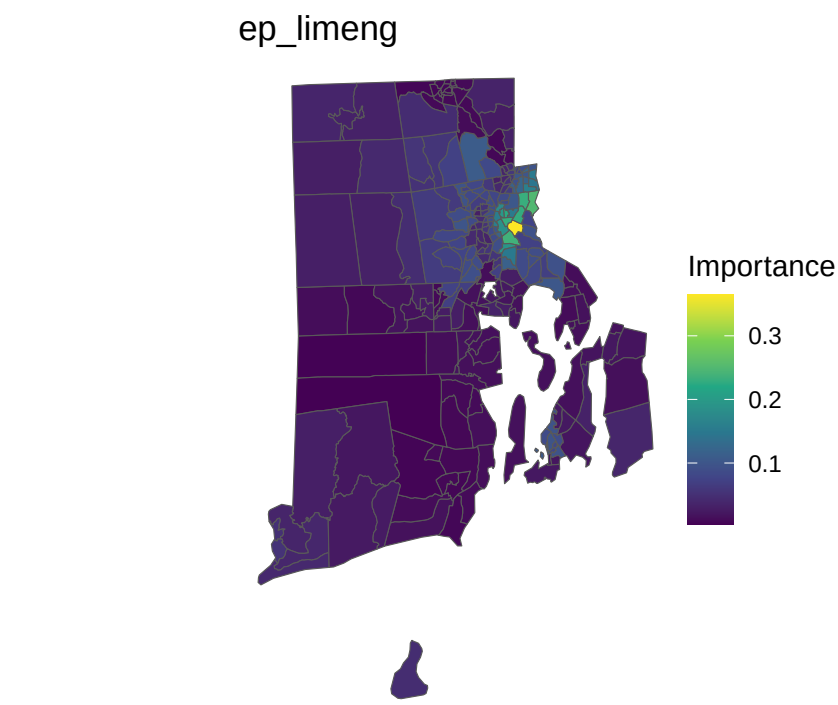
A.C.S. 2015−2019

Distibution of Percentage of civilian noninstitutionalized population with a disability



A.C.S. 2015−2019

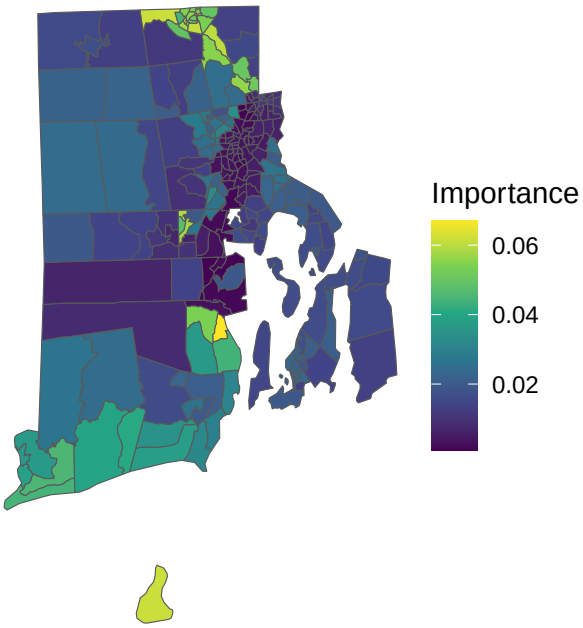
Distibution of Percentage of persons (age 5+) who speak English 'less than well'



A.C.S. 2015−2019

Distibution of Percentage of persons in group quarters estimate

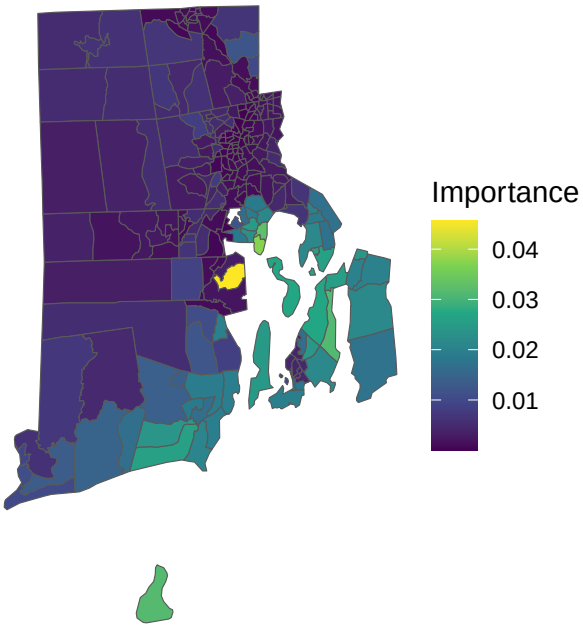
ep_groupq



A.C.S. 2015−2019

Distibution of Percentage of Mobile Homes

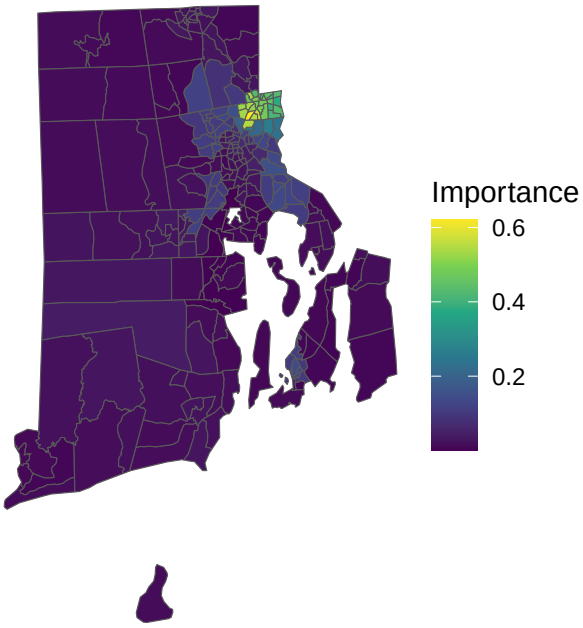
ep_mobile



A.C.S. 2015−2019

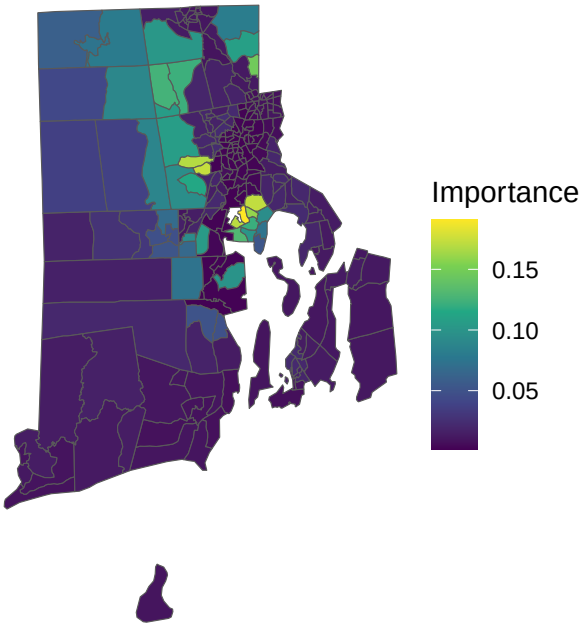
Distibution of Percentage Individuals With Asthma

ep_asthma



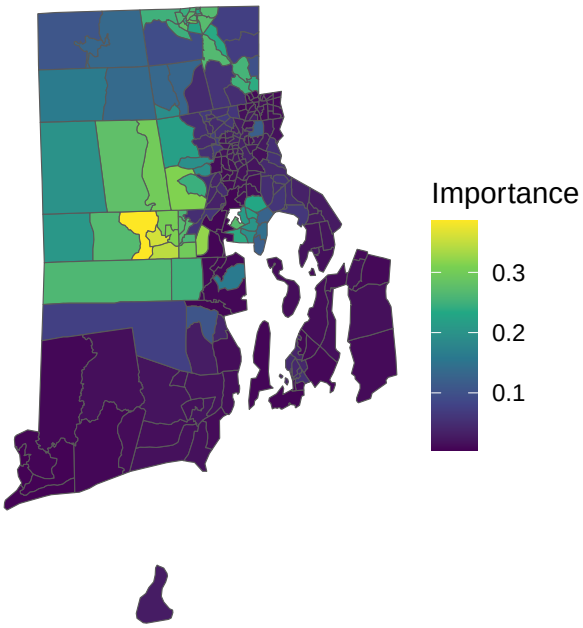
Distibution of Percentage Individuals With Cancer

ep_cancer



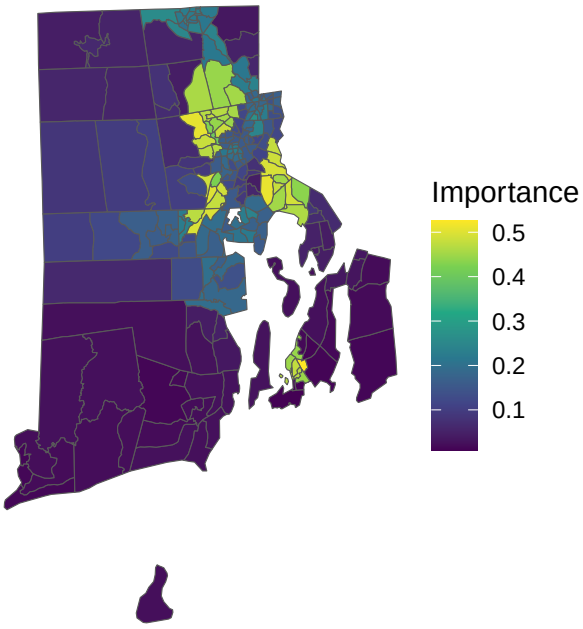
Distibution of Percentage Individuals With High Blood Pressure

ep_bphigh

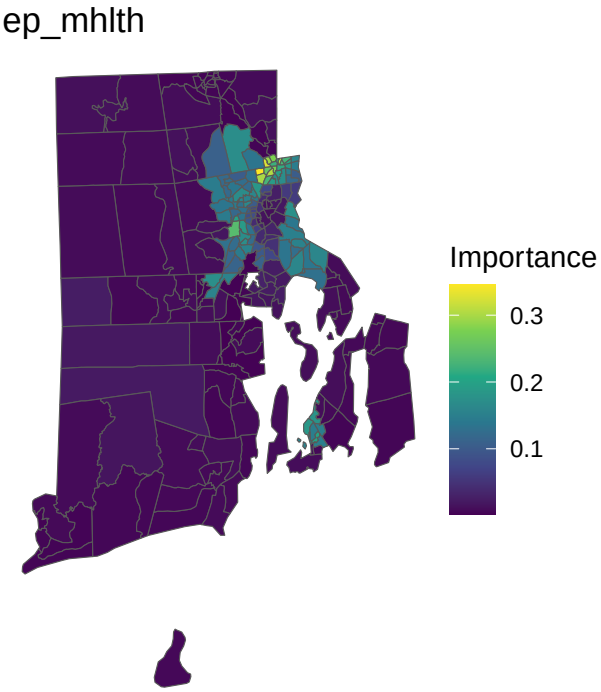


Distibution of Percentage Individuals With Diabetes

ep_diabetes



Distibution of Percentage Individuals No Reporting Good Mental Health



Code Appendix

```
knitr::opts_chunk$set(warning = FALSE, message = FALSE, echo = FALSE)
knitr::opts_chunk$set(fig.width=6, fig.height=4)
options(tigris_use_cache = TRUE)
options(repos = c(CRAN = "https://cran.r-project.org"))
knitr::include_graphics("/Users/diazg/Documents/GitHub/MPH-Thesis_GeographicalRandomForest/Nest
knitr::include_graphics("/Users/diazg/Documents/GitHub/MPH-Thesis_GeographicalRandomForest/Metl
######################
#### Preparation ####
######################

### importing packages
# define desired packages
library(tidyverse)     # general data manipulation
library(knitr)         # Rmarkdown interactions
library(here)          # define top level of project folder
                           # this allows for specification of where
                           # things live in relation to the top level
library(foreach)       # parallel execution
# spatial tasks
library(tigris)        # obtain shp files
library(spdep)         # exploratory spatial data analysis
# random forest
library(caret)         # machine learning model training
library(rsample)       # splitting testing/training data
library(randomForest)  # traditional RF model
library(SpatialML)     # spatial RF model
# others
library(doParallel)    # parallel processing
library(foreach)       # parallel processing
library(ggpubr)        # arrange multiple graphs
library(gridExtra)     # arrange multiple graphs
library(ClustGeo)      # h clustering

### setting seed
set.seed(926)

### loading data
eji_df = read_csv(here::here("01_Data", "eji_df.csv")) %>%
  mutate(geoid = as.character(geoid)) %>%
  select(-...1)

### obtaining SPH files for RI tracts
tracts = tracts(state = "RI", year = 2010, cb = TRUE)
# removing excess characters to allow for join
tracts$GEO_ID = str_remove(tracts$GEO_ID, "^1400000US")
```

```r
### joining data
eji_df = inner_join(tracts, eji_df, by = c("GEO_ID" = "geoid"))

map = eji_df
# make a second copy for later
map_map = map

### defining analytical coordinates and df
df_coords = eji_df %>%
  mutate(
    # redefines geometry to be the centroid of the polygon
    geometry = st_centroid(geometry),
    # pulls the lon and lat for the centroid
    lon = map_dbl(geometry, ~st_point_on_surface(.x)[[1]]),
    lat = map_dbl(geometry, ~st_point_on_surface(.x)[[2]])) %>%
  # removes geometry, coerce to data.frame
  st_drop_geometry() %>%
  # only select the lon and lat
  select(lon, lat)

# only obtain response and predictor variables
df = eji_df %>%
  st_drop_geometry() %>%
  select(rpl_eji , starts_with("e_"), starts_with("ep_"))

unregister_dopar <- function() {
    env <- foreach:::.foreachGlobals
    rm(list=ls(name=env), pos=env)
}
####################
##### RF Mod 1 #####
####################

### setting seed
set.seed(926)

# obtain the number of predictors
pred_num = eji_df %>%
  st_drop_geometry() %>%
  select(starts_with("e_"), starts_with("ep_")) %>%
  colnames() %>%
  length()
# determine the default number of predictors
mtry = round(pred_num / 3)

# creating the first model
# cross validated evaluation
```

```r
cl = makeCluster(detectCores() - 1)  # Use one less core than available
registerDoParallel(cl)

rf_mod1 = train(rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr +
                          e_npl + e_tri + e_tsd + e_rmp + e_coal + e_lead +
                          e_park + e_houage + e_wlkind + e_rail + e_road + e_airprt +
                          e_impwtr + ep_minrty + ep_pov200 + ep_nohsdp +
                          ep_unemp + ep_renter + ep_houbdn + ep_uninsur +
                          ep_noint + ep_age65 + ep_age17 + ep_disabl +
                          ep_limeng + ep_mobile + ep_groupq + ep_bphigh +
                          ep_asthma + ep_cancer + ep_mhlth + ep_diabetes,
                data = df,
                method = "rf",
                trControl = trainControl(method = "cv", number = 10, allowParallel = TRUE ),
                tuneGrid = expand.grid(mtry = mtry),
                ntree = 500,
                importance = TRUE)

stopCluster(cl)
unregister_dopar()

# Print the results
rf_mod1$finalModel

Best_mtry = 5
Best_ntree = 500
Test_Error = NA
RMSE = rf_mod1$results$RMSE
MAE  = rf_mod1$results$MAE
R_squared =  rf_mod1$results$Rsquared

model1_table = data.frame(Best_mtry, Best_ntree, Test_Error, RMSE, MAE, R_squared)
####################
##### RF Mod 2 #####
####################

### setting seed
set.seed(926)

### creating the custom function
customRF <- list(type = "Regression", library = "randomForest", loop = NULL)
customRF$parameters <- data.frame(parameter = c("mtry", "ntree"), class = rep("numeric", 2), la
customRF$grid <- function(x, y, len = NULL, search = "grid") {}
customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {
  randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...)
}
customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
```

```
   predict(modelFit, newdata)
customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)
   predict(modelFit, newdata, type = "prob")
customRF$sort <- function(x) x[order(x[,1]),]
customRF$levels <- function(x) x$classes
### defining the outer folds
outer_folds = createFolds(df$rpl_eji, k = 5)

df = df %>%
  mutate(outer_fold_id = case_when(
    row_number() %in% outer_folds$Fold1 ~ 1,
    row_number() %in% outer_folds$Fold2 ~ 2,
    row_number() %in% outer_folds$Fold3 ~ 3,
    row_number() %in% outer_folds$Fold4 ~ 4,
    row_number() %in% outer_folds$Fold5 ~ 5,
    TRUE ~ 999
  ))

nested_cv = function(form, data, response_var, method, trControl, tuneGrid, k) {

  # Initialize the list to store nested cross-validation results
  model_results = list()

  # Perform the nested cross-validation
  for (i in seq_len(k)) {
    train_data = data %>% filter(outer_fold_id == i)
    test_data = data %>% filter(outer_fold_id != i)

    # Perform inner cross-validation with parallel processing
    inner_model = train(
      form = form,
      data = train_data,
      method = method,
      trControl = trControl,
      tuneGrid = tuneGrid,
      importance = TRUE
    )

    # Evaluate the model on the outer test data
    predictions = predict(inner_model, newdata = test_data)
    performance_metric = postResample(pred = predictions, obs = test_data[[response_var]])

    # Store the results
    model_results[[i]] = list(
      model = inner_model,
      performance = performance_metric
    )
```

```
  }

  return(model_results)
}
### define arguments
num_cols = df %>%
  select(starts_with("e_"), starts_with("ep_")) %>%
  colnames() %>% length()

grid = expand.grid(.mtry = c(1:num_cols),
                   .ntree = c(100, 150, 200, 250,
                              300, 350, 400, 450,
                              500, 550, 600, 650,
                              700, 750, 800, 850,
                              900, 950, 1000))

ctrl = trainControl(method = "cv", number = 10)

k = length(outer_folds)

model2_results = nested_cv(
  form = rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr +
                           e_npl + e_tri + e_tsd + e_rmp + e_coal + e_lead +
                           e_park + e_houage + e_wlkind + e_rail + e_road + e_airprt +
                           e_impwtr + ep_minrty + ep_pov200 + ep_nohsdp +
                           ep_unemp + ep_renter + ep_houbdn + ep_uninsur +
                           ep_noint + ep_age65 + ep_age17 + ep_disabl +
                           ep_limeng + ep_mobile + ep_groupq + ep_bphigh +
                           ep_asthma + ep_cancer + ep_mhlth + ep_diabetes,
  response_var = "rpl_eji",
  data = df,
  method = customRF,
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = grid,
  k = k
)
Fold = c(1:5)

Tuned_mtry = c(as.numeric(model2_results[[1]]$model$bestTune[1]),
               as.numeric(model2_results[[2]]$model$bestTune[1]),
               as.numeric(model2_results[[3]]$model$bestTune[1]),
               as.numeric(model2_results[[4]]$model$bestTune[1]),
               as.numeric(model2_results[[5]]$model$bestTune[1]))

Tuned_ntree = c(as.numeric(model2_results[[1]]$model$bestTune[2]),
               as.numeric(model2_results[[2]]$model$bestTune[2]),
               as.numeric(model2_results[[3]]$model$bestTune[2]),
```

```
              as.numeric(model2_results[[4]]$model$bestTune[2]),
              as.numeric(model2_results[[5]]$model$bestTune[2]))

RMSE = c(as.numeric(model2_results[[1]]$performance[[1]]),
         as.numeric(model2_results[[2]]$performance[[1]]),
         as.numeric(model2_results[[3]]$performance[[1]]),
         as.numeric(model2_results[[4]]$performance[[1]]),
         as.numeric(model2_results[[5]]$performance[[1]]))

MAE = c(as.numeric(model2_results[[1]]$performance[[3]]),
         as.numeric(model2_results[[2]]$performance[[3]]),
         as.numeric(model2_results[[3]]$performance[[3]]),
         as.numeric(model2_results[[4]]$performance[[3]]),
         as.numeric(model2_results[[5]]$performance[[3]]))

R_squared = c(as.numeric(model2_results[[1]]$performance[[2]]),
         as.numeric(model2_results[[2]]$performance[[2]]),
         as.numeric(model2_results[[3]]$performance[[2]]),
         as.numeric(model2_results[[4]]$performance[[2]]),
         as.numeric(model2_results[[5]]$performance[[2]]))

tab = data.frame(Fold, Tuned_mtry, Tuned_ntree, RMSE, MAE, R_squared)

Best_mtry = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Tuned_mtry)

Best_ntree = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Tuned_ntree)

Test_Error = NA

RMSE = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(RMSE)

MAE  = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(MAE)

R_squared = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(R_squared)

Best_Fold = tab %>%
  filter(RMSE == min(RMSE)) %>%
```

```r
  pull(Fold)

# access the model information by preforming the following function: model2_results[[Best_Fold

model2_table = data.frame(Best_mtry, Best_ntree, Test_Error, RMSE, MAE, R_squared)

kable(tab, caption = "Model 2 - Traditional Cross Validation: Hyperparametyer Tuning and Perfo
      digits = 3,
      align = c("lllccc"))

####################
##### RF Mod 3 #####
####################

### this code c
D0 <- dist(df)
tree <- hclustgeo(D0)

"
You cut the dendrogram horizontally at a level that
represents a reasonable trade-off between the
number of clusters and the within-cluster similarity, with
the goal to  Look for large vertical gaps between
successive merges. The idea is to cut the dendrogram at
a height where the gap between clusters is largest,
indicating that merging clusters beyond that point would
result in combining distinct groups.

I am going to go with k = 4 because the trade off seems to
work well, and all groups seem similar in size.
"

k = 5

plot(tree, hang = -1, label = FALSE,
     xlab = "", sub = "",
     main = "Ward Dendrogram with D0 only")
rect.hclust(tree ,k = k, border = c(1:k))
legend("topright", legend = paste("cluster", 1:k),
       fill=1:k, bty="n", border = "white")

# taking geographical and neighorhood constraints into account
list.nb = poly2nb(map, queen=TRUE) #list of neighbours of each city
A = nb2mat(neighbours = list.nb,style="B", zero.policy = TRUE)
D1 = as.dist(1-A)
# choice of mixing parameter
range.alpha = seq(0,1,0.1)
```

```r
cr = choicealpha(D0, D1,
                 range.alpha,
                 k,
                 graph=FALSE)

# normalization if required given the characteristics
# geographic distances with other data, normalization
# might be required to balance the contributions of
# geographic and non-geographic distances. This ensures
# that neither component disproportionately influences
# the clustering result.

plot(cr, norm = TRUE)

a = 0.3

# here the plot seggust to choose alpha = 0.3
tree = hclustgeo(D0,D1,alpha=a)
P5bis = cutree(tree,k)
map$cluster_id = as.factor(P5bis)
df_coords$cluster_id = as.factor(P5bis)
# graph produced by clustering method
ggplot(data = map) +
  geom_sf(aes(fill = cluster_id), color = "grey") +
  scale_fill_viridis_d(name = "cluster_id") +
  labs(title = "Partition P5bis obtained with alpha=0.3
         and neighborhood dissimilarities") +
  theme_void() +
  theme(legend.position = "left")
# This function performs nested cross-validation with parallel processing
spatial_nested_cv = function(form, data, method, trControl, tuneGrid, cluster_col) {
  outer_folds = createFolds(data[[cluster_col]], k = length(unique(data[[cluster_col]])), retu

  outer_results = foreach(i = seq_along(outer_folds), .packages = c('caret', 'randomForest'),
    train_indices = outer_folds[[i]]
    train_data = data[train_indices, ]
    test_data = data[-train_indices, ]

    # Perform inner cross-validation
    inner_model = train(
      form = form,
      data = train_data,
      method = method,
      trControl = trControl,
      tuneGrid = tuneGrid,
      importance = TRUE
    )
```

```
    # Evaluate the model on the outer test data
    predictions <- predict(inner_model, newdata = test_data)
    performance_metric <- postResample(pred = predictions, obs = test_data$rpl_eji)

    list(
      model = inner_model,
      performance = performance_metric
    )
  }

  stopCluster(cl)  # Stop the parallel backend

  return(outer_results)
}
# implementing an exhaustive search
map = map %>% st_drop_geometry()

# Register parallel backend
num_cores <- detectCores() - 1
cl <- makeCluster(num_cores)
registerDoParallel(cl)

model3_results = spatial_nested_cv(
  form = rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr +
                           e_npl + e_tri + e_tsd + e_rmp + e_coal + e_lead +
                           e_park + e_houage + e_wlkind + e_rail + e_road + e_airprt +
                           e_impwtr + ep_minrty + ep_pov200 + ep_nohsdp +
                           ep_unemp + ep_renter + ep_houbdn + ep_uninsur +
                           ep_noint + ep_age65 + ep_age17 + ep_disabl +
                           ep_limeng + ep_mobile + ep_groupq + ep_bphigh +
                           ep_asthma + ep_cancer + ep_mhlth + ep_diabetes,
  data = map,
  method = customRF,
  trControl = ctrl,
  tuneGrid = grid,
  cluster_col = "cluster_id")

unregister_dopar()
Fold = c(1:k)

Tuned_mtry = c(as.numeric(model3_results[[1]]$model$bestTune[1]),
               as.numeric(model3_results[[2]]$model$bestTune[1]),
               as.numeric(model3_results[[3]]$model$bestTune[1]),
               as.numeric(model3_results[[4]]$model$bestTune[1]),
               as.numeric(model3_results[[5]]$model$bestTune[1]))

Tuned_ntree = c(as.numeric(model3_results[[1]]$model$bestTune[2]),
```

```
                as.numeric(model3_results[[2]]$model$bestTune[2]),
                as.numeric(model3_results[[3]]$model$bestTune[2]),
                as.numeric(model3_results[[4]]$model$bestTune[2]),
                as.numeric(model3_results[[5]]$model$bestTune[2]))

RMSE = c(as.numeric(model3_results[[1]]$performance[[1]]),
          as.numeric(model3_results[[2]]$performance[[1]]),
          as.numeric(model3_results[[3]]$performance[[1]]),
          as.numeric(model3_results[[4]]$performance[[1]]),
          as.numeric(model3_results[[5]]$performance[[1]]))

Rsquared = c(as.numeric(model3_results[[1]]$performance[[2]]),
          as.numeric(model3_results[[2]]$performance[[2]]),
          as.numeric(model3_results[[3]]$performance[[2]]),
          as.numeric(model3_results[[4]]$performance[[2]]),
          as.numeric(model3_results[[5]]$performance[[2]]))

MAE = c(as.numeric(model3_results[[1]]$performance[[3]]),
         as.numeric(model3_results[[2]]$performance[[3]]),
         as.numeric(model3_results[[3]]$performance[[3]]),
         as.numeric(model3_results[[4]]$performance[[3]]),
         as.numeric(model3_results[[5]]$performance[[3]]))

tab = data.frame(Fold, Tuned_mtry, Tuned_ntree, RMSE, MAE, Rsquared)

Best_mtry = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Tuned_mtry)

Best_ntree = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Tuned_ntree)

Test_Error = NA

RMSE = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(RMSE)

MAE  = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(MAE)

R_squared = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Rsquared)
```

```r
Best_Fold = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Fold)

# access the model information by preforming the following function: model3_results[[Best_Fold

model3_table = data.frame(Best_mtry, Best_ntree, Test_Error, RMSE, MAE, R_squared)

kable(tab, caption = "Model 3 - Partially Spatial Cross Validation: Hyperparametyer Tuning and
      digits = 3,
      align = c("lllccc"))
####################
##### RF Mod 4 #####
####################

# create an interaction function to search over different values of mtry
mtry_iter = function(from, to, stepFactor = 1.05){
  nextEl = function(){
    if (from > to) stop('StopIteration')
    i = from
    from <<- ceiling(from * stepFactor)
    i
  }
  obj = list(nextElem = nextEl)
  class(obj) = c('abstractiter', 'iter')
  obj
}

# Define the function to calculate RMSE, MAE, and R-squared
calculate_metrics <- function(predictions, actuals) {
  residuals <- predictions - actuals
  mse <- mean(residuals^2)
  rmse <- sqrt(mse)
  mae <- mean(abs(residuals))
  r_squared <- 1 - sum(residuals^2) / sum((actuals - mean(actuals))^2)

  return(c(RMSE = rmse, MAE = mae, R2 = r_squared))
}


# Nested cross-validation function with random forest
nested_cv_tune <- function(x, y, ntree = c(51, 101, 501, 1001, 1501), num_folds = 5) {

  # Create outer cross-validation folds
  outer_folds <- createFolds(y, k = num_folds, returnTrain = TRUE)

  # Initialize list to store outer fold results
```

```r
outer_results <- list()

# Initialize list to store final models
final_models <- list()

# Iterate over each outer fold
for (i in seq_along(outer_folds)) {
  train_index <- outer_folds[[i]]
  x_train <- x[train_index, ]
  y_train <- y[train_index]
  x_test <- x[-train_index, ]
  y_test <- y[-train_index]

  # Inner cross-validation for hyperparameter tuning
  inner_results <- foreach(mtry = mtry_iter(1, ncol(x_train)), .combine = 'rbind', .packages
    model <- randomForest(x_train, y_train, ntree = max(ntree), mtry = mtry, keep.forest = F
    if (is.factor(y)) {
      errors <- data.frame(ntree = ntree, mtry = mtry, error = model$err.rate[ntree, 1])
    } else {
      errors <- data.frame(ntree = ntree, mtry = mtry, error = model$mse[ntree])
    }
    return(errors)
  }

  # Find the best hyperparameters based on the inner fold results
  best_params <- inner_results[which.min(inner_results$error), ]

  # Train the final model on the entire outer training set using the best hyperparameters
  final_model <- randomForest(x_train, y_train, ntree = best_params$ntree, mtry = best_params

  # Store the final model in the list
  final_models[[i]] <- final_model

  # Test the final model on the outer test set
  final_pred <- predict(final_model, x_test)

  # Calculate performance metrics
  if (is.factor(y)) {
    test_error <- mean(final_pred != y_test)
    rmse <- NA
    mae <- NA
    rsquared <- NA
  } else {
    test_error <- mean((final_pred - y_test)^2)
    rmse <- sqrt(mean((final_pred - y_test)^2))
    mae <- mean(abs(final_pred - y_test))
    rsquared <- 1 - (sum((final_pred - y_test)^2) / sum((y_test - mean(y_test))^2))
```

```
    }

    # Store the results
    outer_results[[i]] <- data.frame(
      Fold = i,
      Best_mtry = best_params$mtry,
      Best_ntree = best_params$ntree,
      Test_Error = test_error,
      RMSE = rmse,
      MAE = mae,
      R_squared = rsquared
    )
  }

  # Combine all outer fold results
  final_results <- do.call(rbind, outer_results)

  # Stop the parallel backend
  stopCluster(cl)

  # Return both the results and the models
  return(list(Results = final_results, Models = final_models))
}

# create a vector of ntree values of interest
vntree = c(100, 150, 200, 250,
           300, 350, 400, 450,
           500, 550, 600, 650,
           700, 750, 800, 850,
           900, 950, 1000)

# specify the predictor (x) and outcome (y) object
x = df %>% select(starts_with("e_"), starts_with("ep_"))
y = df %>% pull(rpl_eji)

# Register parallel backend
num_cores <- detectCores() - 1
cl <- makeCluster(num_cores)
registerDoParallel(cl)

# call the custom function
model4_results = nested_cv_tune(x, y, ntree = vntree, num_folds = 5)

unregister_dopar()

model4_models = model4_results$Models
model4_results = model4_results$Results
```

```r
Best_mtry = model4_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(Best_mtry)

Best_ntree = model4_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(Best_ntree)

Test_Error = model4_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(Test_Error)

RMSE = model4_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(RMSE)

MAE  = model4_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(MAE)

R_squared = model4_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(R_squared)

Best_Fold = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Fold)

# access the model information by preforming the following function: model4_models[[Best_Fold]]

model4_table = data.frame(Best_mtry, Best_ntree, Test_Error, RMSE, MAE, R_squared)

kable(model4_results, caption = "Model 4 - Traditional Cross Validation: Hyperparametyer Tuning
      digits = 3,
      align = c("lllcccc"))
####################
##### RF Mod 5 #####
####################

# Nested cross-validation function with random forest
spatial_nested_cv_tune <- function(formula, data, response_var, cluster_col, num_predictors, nt

  # Create outer cross-validation folds
  outer_folds <- createFolds(data[[cluster_col]], k = length(unique(data[[cluster_col]])), retu

  # Initialize list to store outer fold results and models
  outer_results <- list()
```

```r
final_models <- list()

# Iterate over each outer fold
for (i in seq_along(outer_folds)) {
  train_index <- outer_folds[[i]]
  train_data <- data[train_index, ]
  test_data <- data[-train_index, ]

  # Inner cross-validation for hyperparameter tuning
  inner_results <- foreach(mtry = mtry_iter(1, num_predictors), .combine = 'rbind', .packages
    model <- randomForest(formula, data = train_data, ntree = max(ntree), mtry = mtry, keep.
    if (is.factor(train_data[[response_var]])) {
      errors <- data.frame(ntree = ntree, mtry = mtry, error = model$err.rate[ntree, 1])
    } else {
      errors <- data.frame(ntree = ntree, mtry = mtry, error = model$mse[ntree])
    }
    return(errors)
  }

  # Find the best hyperparameters based on the inner fold results
  best_params <- inner_results[which.min(inner_results$error), ]

  # Train the final model on the entire outer training set using the best hyperparameters
  final_model <- randomForest(formula, data = train_data, ntree = best_params$ntree, mtry =

  # Store the final model
  final_models[[i]] <- final_model

  # Test the final model on the outer test set
  final_pred <- predict(final_model, test_data)

  # Calculate performance metrics using response_var
  y_test <- test_data[[response_var]]
  if (is.factor(y_test)) {
    test_error <- mean(final_pred != y_test)
    rmse <- NA
    mae <- NA
    rsquared <- NA
  } else {
    test_error <- mean((final_pred - y_test)^2)
    rmse <- sqrt(mean((final_pred - y_test)^2))
    mae <- mean(abs(final_pred - y_test))
    rsquared <- 1 - (sum((final_pred - y_test)^2) / sum((y_test - mean(y_test))^2))
  }

  # Store the results
  outer_results[[i]] <- data.frame(
```

```
      Fold = i,
      Best_mtry = best_params$mtry,
      Best_ntree = best_params$ntree,
      Test_Error = test_error,
      RMSE = rmse,
      MAE = mae,
      R_squared = rsquared
    )
  }

  # Combine all outer fold results
  final_results <- do.call(rbind, outer_results)

  # Stop the parallel backend
  stopCluster(cl)


  return(list(Results = final_results, Models = final_models))
}
# create a vector of ntree values of interest
vntree = c(100, 150, 200, 250,
           300, 350, 400, 450,
           500, 550, 600, 650,
           700, 750, 800, 850,
           900, 950, 1000)

# specify the predictor (x) and outcome (y) object
cluster_id = map %>% select(cluster_id) %>% st_drop_geometry()
x = map %>% select(starts_with("e_"), starts_with("ep_")) %>% st_drop_geometry()
y = map %>% pull(rpl_eji) %>% st_drop_geometry()

# Register parallel backend
num_cores <- detectCores() - 1
cl <- makeCluster(num_cores)
registerDoParallel(cl)

# call the custom function
model5_results = spatial_nested_cv_tune(
  form = rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr +
                           e_npl + e_tri + e_tsd + e_rmp + e_coal + e_lead +
                           e_park + e_houage + e_wlkind + e_rail + e_road + e_airprt +
                           e_impwtr + ep_minrty + ep_pov200 + ep_nohsdp +
                           ep_unemp + ep_renter + ep_houbdn + ep_uninsur +
                           ep_noint + ep_age65 + ep_age17 + ep_disabl +
                           ep_limeng + ep_mobile + ep_groupq + ep_bphigh +
                           ep_asthma + ep_cancer + ep_mhlth + ep_diabetes,
  data = map,
```

```
  response_var = "rpl_eji",
  num_predictors = num_cols,
  cluster_col = "cluster_id", ntree = vntree)

unregister_dopar()

model5_models = model5_results$Models
model5_results = model5_results$Results
Best_mtry = model5_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(Best_mtry)

Best_ntree = model5_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(Best_ntree)

Test_Error = model5_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(Test_Error)

RMSE = model5_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(RMSE)

MAE  = model5_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(MAE)

R_squared = model5_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  pull(R_squared)

Best_Fold = tab %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Fold)

# access the model information by preforming the following function: model5_models[[Best_Fold]]

model5_table = data.frame(Best_mtry, Best_ntree, Test_Error, RMSE, MAE, R_squared)

kable(model5_results, caption = "Model 5 - Partially Spatial Cross Validation: Hyperparametyer
      digits = 3,
      align = c("lllcccc"))
#####################
##### RF Models ####
#####################
# make a list of all the model objects
```

```r
rf_model_1 = rf_mod1$finalModel
rf_model_2 = model2_results[[Best_Fold]]$model$finalModel
rf_model_3 = model3_results[[Best_Fold]]$model$finalModel
rf_model_4 = model4_models[[Best_Fold]]
rf_model_5 = model5_models[[Best_Fold]]

final_model_lst = list(rf_model_1, rf_model_2,
                       rf_model_3, rf_model_4,
                       rf_model_5)


# Create a data frame with the results
results_rf = rbind(model1_table, model2_table,
                   model3_table, model4_table,
                   model5_table)

Model = c(1, 2, 3, 4, 5)

results_rf = cbind(as.data.frame(Model), results_rf)

# Best Model Metrics
Best_Model = results_rf %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Model)

Best_ntree = results_rf %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Best_ntree)

Best_mtry = results_rf %>%
  filter(RMSE == min(RMSE)) %>%
  pull(Best_mtry)

Best_rf = results_rf %>%
  filter(RMSE == min(RMSE))

# Print the results using kable
kable(results_rf, caption = "Performance Metrics for Each Model",
      digits = 3, align = c("l", "c", "c", "c", "c", "c", "c"))

# Export the results_rf as csv file
write.csv(results_rf, "results_rf.csv", row.names = FALSE)


save(rf_model_1, file = "rf_model_1.RData")
save(rf_model_2, file = "rf_model_2.RData")
save(rf_model_3, file = "rf_model_3.RData")
```

```
save(rf_model_4, file = "rf_model_4.RData")
save(rf_model_5, file = "rf_model_5.RData")
####################
#### GWRF Mod 1 ####
####################

# testing for optimal bandwidth
temp = SpatialML::grf.bw(rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr +
                           e_npl + e_tri + e_tsd + e_rmp + e_coal + e_lead +
                           e_park + e_houage + e_wlkind + e_rail + e_road + e_airprt +
                           e_impwtr + ep_minrty + ep_pov200 + ep_nohsdp +
                           ep_unemp + ep_renter + ep_houbdn + ep_uninsur +
                           ep_noint + ep_age65 + ep_age17 + ep_disabl +
                           ep_limeng + ep_mobile + ep_groupq + ep_bphigh +
                           ep_asthma + ep_cancer + ep_mhlth + ep_diabetes,
                             dataset = df,
                             kernel = "adaptive",
                             bw.min = 20,
                             bw.max = 50,
                             coords = df_coords,
                             trees = 500,
                             mtry = 12,
                             step = 1)
best.bw_gwrf_mod1 = temp$Best.BW

# defining the spatial model with prior model hyperparameters
gwrf_mod1 = SpatialML::grf(rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr +
                           e_npl + e_tri + e_tsd + e_rmp + e_coal + e_lead +
                           e_park + e_houage + e_wlkind + e_rail + e_road + e_airprt +
                           e_impwtr + ep_minrty + ep_pov200 + ep_nohsdp +
                           ep_unemp + ep_renter + ep_houbdn + ep_uninsur +
                           ep_noint + ep_age65 + ep_age17 + ep_disabl +
                           ep_limeng + ep_mobile + ep_groupq + ep_bphigh +
                           ep_asthma + ep_cancer + ep_mhlth + ep_diabetes,
                         dframe = df,
                         kernel = "adaptive",
                         coords = df_coords,
                         bw = best.bw_gwrf_mod1,
                         ntree = 500,
                         mtry = 12)
####################
#### GWRF Mod 2 ####
####################

# testing for optimal bandwidth
temp = SpatialML::grf.bw(rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr +
                           e_npl + e_tri + e_tsd + e_rmp + e_coal + e_lead +
```

```
                            e_park + e_houage + e_wlkind + e_rail + e_road + e_airprt +
                            e_impwtr + ep_minrty + ep_pov200 + ep_nohsdp +
                            ep_unemp + ep_renter + ep_houbdn + ep_uninsur +
                            ep_noint + ep_age65 + ep_age17 + ep_disabl +
                            ep_limeng + ep_mobile + ep_groupq + ep_bphigh +
                            ep_asthma + ep_cancer + ep_mhlth + ep_diabetes,
                              dataset = df,
                              kernel = "adaptive",
                              bw.min = 20,
                              bw.max = 50,
                              coords = df_coords,
                              trees = Best_ntree,
                              mtry = Best_mtry,
                              step = 1)

best.bw_gwrf_mod2 = temp$Best.BW

# defining the spatial model with prior model hyperarameters
gwrf_mod2 = SpatialML::grf(rpl_eji ~ e_ozone + e_pm + e_dslpm + e_totcr +
                            e_npl + e_tri + e_tsd + e_rmp + e_coal + e_lead +
                            e_park + e_houage + e_wlkind + e_rail + e_road + e_airprt +
                            e_impwtr + ep_minrty + ep_pov200 + ep_nohsdp +
                            ep_unemp + ep_renter + ep_houbdn + ep_uninsur +
                            ep_noint + ep_age65 + ep_age17 + ep_disabl +
                            ep_limeng + ep_mobile + ep_groupq + ep_bphigh +
                            ep_asthma + ep_cancer + ep_mhlth + ep_diabetes,
                           dframe = df,
                           kernel = "adaptive",
                           coords = df_coords,
                           bw = best.bw_gwrf_mod2,
                           ntree = Best_ntree,
                           mtry = Best_mtry) # this is a ranger argument
                                                  # specification of this value
                                                  # corrected errors that previously appe
                                                  # no importance value specified
####################
#### GWRF Models ###
####################
# Model 6
predictions6 = gwrf_mod1$Global.Model$predictions
mse6 = mean((df$rpl_eji - predictions6)^2)
rmse6 = sqrt(mse6)
mae6 = sum(abs(df$rpl_eji - predictions6))/length(predictions6)
r_squared6 = 1 - sum((df$rpl_eji - predictions6)^2) / sum((df$rpl_eji - mean(df$rpl_eji))^2)

# Model 7
predictions7 = gwrf_mod2$Global.Model$predictions
```

```r
mse7 = mean((df$rpl_eji - predictions7)^2)
rmse7 = sqrt(mse7)
mae7 = sum(abs(df$rpl_eji - predictions7))/length(predictions7)
r_squared7 = 1 - sum((df$rpl_eji - predictions7)^2) / sum((df$rpl_eji - mean(df$rpl_eji))^2)

# Create a data frame with the results
results_grf = data.frame(
  Model = c(6, 7),
  bw = c(best.bw_gwrf_mod1, best.bw_gwrf_mod2),
  mtry = c(5, Best_mtry),
  ntree = c(500,Best_ntree),
  MAE = c(mae6, mae7),
  RMSE = c(rmse6, rmse7),
  R_Squared = c(r_squared6, r_squared7)
)

# Print the results using kable
kable(results_grf, caption = "Performance Metrics for Each Model",
      digits = 3, align = c("c", "c", "c", "c", "c", "c", "c"))

Best_gwrf =  results_grf %>% filter(Model == "7")
####################
#### All Models ####
####################
Best_rf = Best_rf %>%
  mutate(Bandwidth = NA) %>%
  select(Model, Bandwidth, Best_mtry, Best_ntree, RMSE, MAE, R_squared)
colnames(Best_gwrf) = colnames(Best_rf)

final_results = rbind(Best_rf, Best_gwrf)

# Print the results using kable
kable(final_results, caption = "Performance Metrics for Each Model",
      digits = 3, align = c("lcccccc"))
####################
#### RF Results ####
####################
# Variable Importance
temp = as.data.frame(model3_results[[Best_Fold]]$model$finalModel$importance)
colnames(temp) = c("IncMSE", "IncNodePurity")
temp = tibble::rownames_to_column(temp, "Variable")

ggplot(temp, aes(x = fct_reorder(Variable, IncMSE), y = IncMSE)) +
  geom_bar(stat = "identity", fill = "grey") +
  coord_flip() +  # Flip coordinates for better readability
  labs(title = "Variable Importance (Increase in %IncMSE)",
       subtitle = "Traditional Random Forest Model",
```

```
       x = "Variable",
       y = "Increase in MSE") +
  theme_bw()

# Generate Partial Dependence Plots and store them as ggplot objects
rf_mod = rf_model_3
a = as.data.frame(randomForest::partialPlot(rf_mod,
                                            df, e_tri, plot = FALSE))
b = as.data.frame(randomForest::partialPlot(rf_mod,
                                            df, ep_pov200, plot = FALSE))
c = as.data.frame(randomForest::partialPlot(rf_mod,
                                            df, e_totcr, plot = FALSE))
d = as.data.frame(randomForest::partialPlot(rf_mod,
                                            df, ep_diabetes, plot = FALSE))

# Convert the base R plots to ggplot objects
plot_a = ggplot(a, aes(x, y)) + geom_line() + labs(title = "Partial Dependence of e_limeng")
plot_b = ggplot(b, aes(x, y)) + geom_line() + labs(title = "Partial Dependence of e_pov150")
plot_c = ggplot(c, aes(x, y)) + geom_line() + labs(title = "Partial Dependence of e_noveh")
plot_d = ggplot(d, aes(x, y)) + geom_line() + labs(title = "Partial Dependence of e_munit")

# Arrange the plots in a grid
grid.arrange(plot_a, plot_b, plot_c, plot_d, nrow = 2, ncol = 2)
####################
### GWRF Results ###
####################

# Spatial Distribution of Prediction and Observation Summary Index Values
predictions = gwrf_mod2$Global.Model$predictions
map_map = map_map %>%
  mutate(grf_pred = predictions)

map_map = cbind(map_map, gwrf_mod2$Global.Model$predictions)

# Predicted 1:1 Plot
ggplot(map_map, aes(x = grf_pred, y = rpl_eji)) +
  geom_point() +
  theme_bw() +
  geom_abline(slope=1, intercept=0,linetype="dashed",linewidth=0.5) +
  geom_smooth(method = "lm", se = FALSE, colour="black",linewidth=0.5) +
  labs(x="Observed", y = "Predicted")

a = ggplot(map_map, aes(fill = rpl_eji)) +
  geom_sf() +
  theme_void() +
  scale_fill_viridis_c() +
  labs(title = "Observed EJI Values") +
```

```
  theme(legend.title=element_blank())

b = ggplot(map_map, aes(fill = grf_pred)) +
  geom_sf() +
  theme_void() +
  scale_fill_viridis_c() +
  labs(title = "Predicted EJI Values") +
  theme(legend.title=element_blank())

ggarrange(a,b, ncol = 2, common.legend = TRUE, legend="bottom")
# Global Variable Importance
temp = as.data.frame(gwrf_mod2$Global.Model$variable.importance)
colnames(temp) = c("IncMSE")
temp = tibble::rownames_to_column(temp, "Variable")
ggplot(temp, aes(x = reorder(Variable, IncMSE), y = IncMSE)) +
  geom_bar(stat = "identity", fill = "grey") +
  coord_flip() +  # Flip coordinates for better readability
  labs(title = "Variable Importance (Increase in %IncMSE)",
       subtitle = "Geographically Weighted Random Forest Model",
       x = "Variable",
       y = "Increase in MSE") +
  theme_bw()

# Local Variable Importance
map_variable_importance = map_map %>%
  select(-rpl_eji, -starts_with("e_"), -starts_with("ep_"), -grf_pred) %>%
  cbind(gwrf_mod2$Local.Variable.Importance)
variable_names = colnames(gwrf_mod2$Local.Variable.Importance)
plot_lst = list()

# Loop through the names and create maps
for (var_name in variable_names) {

  # Generate the map using ggplot2
  p <- ggplot(map_variable_importance, aes_string(fill = var_name)) +
    geom_sf() +
    scale_fill_viridis_c() +
    labs(title = paste(var_name),
         fill = "Importance") +
    theme_void()

  # Save the plot or print it
  plot_lst[[var_name]] = p
}
############ SES Variableeeeeeeeeeeeeeee
annotate_figure(plot_lst[["e_ozone"]],
                top = text_grob("Distibution of Annual mean days above 03 regulatory standard
```

```
                fig.lab = "EPA AQS 2014-2016", fig.lab.pos = c("bottom.right"), fig.lab.size =
annotate_figure(plot_lst[["e_pm"]],
                top = text_grob("Distibu tion of Annual mean days above PM2.5 regulatory standa
                fig.lab = "EPA AQS 2014-2016", fig.lab.pos = c("bottom.right"), fig.lab.size =
annotate_figure(plot_lst[["e_dslpm"]],
                top = text_grob("Distibution of Ambient concentrations of Diesel (PM/m3)", size
                fig.lab = "EPA AQS 2014-2016", fig.lab.pos = c("bottom.right"), fig.lab.size =
annotate_figure(plot_lst[["e_totcr"]],
                top = text_grob("Distibution of Probability of Contracting Cancer (assuming cor
                fig.lab = "EPA AQS 2014-2016", fig.lab.pos = c("bottom.right"), fig.lab.size =
annotate_figure(plot_lst[["e_npl"]],
                top = text_grob("Distibution of Proportion of Tract's Area within 1-mi buffer o
                fig.lab = "EPA Geospatial Download", fig.lab.pos = c("bottom.right"), fig.lab.s
annotate_figure(plot_lst[["e_tri"]],
                top = text_grob("Distibution of Proportion of Tract's Area within 1-mi buffer o
                fig.lab = "EPA Geospatial Download", fig.lab.pos = c("bottom.right"), fig.lab.s
annotate_figure(plot_lst[["e_tsd"]],
                top = text_grob("Distibution of Proportion of Tract's Area within 1-mi buffer o
                                face = "bold", size = 8),
                fig.lab = "EPA Geospatial Download", fig.lab.pos = c("bottom.right"), fig.lab.s
annotate_figure(plot_lst[["e_rmp"]],
                top = text_grob("Distibution of Proportion of Tract's Area within 1-mi buffer o
                fig.lab = "EPA Geospatial Download", fig.lab.pos = c("bottom.right"), fig.lab.s
annotate_figure(plot_lst[["e_coal"]],
                top = text_grob("Distibution of Proportion of tract's area within 1-mi buffer o
                fig.lab = "EPA Geospatial Download", fig.lab.pos = c("bottom.right"), fig.lab.s
annotate_figure(plot_lst[["e_lead"]],
                top = text_grob("Distibution of Proportion of tract's area within 1-mi buffer o
                fig.lab = "EPA Geospatial Download", fig.lab.pos = c("bottom.right"), fig.lab.s
annotate_figure(plot_lst[["e_park"]],
                top = text_grob("Distibution of Proportion of tract's area within 1-mi buffer o
                fig.lab = "2020 TomTom MultiNet Enterpirse Dataset", fig.lab.pos = c("bottom.ri
annotate_figure(plot_lst[["e_houage"]],
                top = text_grob("Distibution of Percentage of houses built pre 1980 (lead expos
                fig.lab = "No Citation", fig.lab.pos = c("bottom.right"), fig.lab.size = 9)
annotate_figure(plot_lst[["e_wlkind"]],
                top = text_grob("Distibution of Walkability Values", face = "bold", size = 14)
                fig.lab = "EPA Walkability Inex", fig.lab.pos = c("bottom.right"), fig.lab.size
annotate_figure(plot_lst[["e_rail"]],
                top = text_grob("Distibution of Proportion of tract's area within 1-mi buffer o
                fig.lab = "EPA Walkability Inex", fig.lab.pos = c("bottom.right"), fig.lab.size
annotate_figure(plot_lst[["e_road"]],
                top = text_grob("Distibution of Proportion of tract's area within 1-mi buffer o
                fig.lab = "EPA Walkability Inex", fig.lab.pos = c("bottom.right"), fig.lab.size
annotate_figure(plot_lst[["e_airprt"]],
                top = text_grob("Distibution of Proportion of tract's area within 1-mi buffer o
                fig.lab = "EPA Walkability Inex", fig.lab.pos = c("bottom.right"), fig.lab.size
```

```
annotate_figure(plot_lst[["ep_minrty"]],
                top = text_grob("Distibution of Percentage of Minority Persons", face = "bold"
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_pov200"]],
                top = text_grob("Distibution of Percentage of Persons Below 200% Poverty", face
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_nohsdp"]],
                top = text_grob("Distibution of Percentage of Persons with no High School Diplo
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_unemp"]],
                top = text_grob("Distibution of Percentage of Unemployed Persons", face = "bold
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_renter"]],
                top = text_grob("Distibution of Percentage of Renters", face = "bold", size = 1
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_houbdn"]],
                top = text_grob("Distibution of Percentage of households that make less than $7
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_uninsur"]],
                top = text_grob("Distibution of Percentage of Unisured Persons", face = "bold"
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_noint"]],
                top = text_grob("Distibution of Percentage of Persons Without Internet", face =
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_age65"]],
                top = text_grob("Distibution of Percentage of persons aged 65 and older", face
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_age17"]],
                top = text_grob("Distibution of Percentage of persons aged 17 and younger", fac
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_disabl"]],
                top = text_grob("Distibution of Percentage of civilian noninstitutionalized pop
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_limeng"]],
                top = text_grob("Distibution of Percentage of persons (age 5+) who speak Englis
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
```

```
annotate_figure(plot_lst[["ep_groupq"]],
                top = text_grob("Distibution of Percentage of persons in group quarters estimat
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9

annotate_figure(plot_lst[["ep_mobile"]],
                top = text_grob("Distibution of Percentage of Mobile Homes", face = "bold", siz
                fig.lab = "A.C.S. 2015-2019", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["ep_asthma"]],
                top = text_grob("Distibution of Percentage Individuals With Asthma", face = "bo

annotate_figure(plot_lst[["ep_cancer"]],
                top = text_grob("Distibution of Percentage Individuals With Cancer", face = "bo

annotate_figure(plot_lst[["ep_bphigh"]],
                top = text_grob("Distibution of Percentage Individuals With High Blood Pressure

annotate_figure(plot_lst[["ep_diabetes"]],
                top = text_grob("Distibution of Percentage Individuals With Diabetes", face = "

annotate_figure(plot_lst[["ep_mhlth"]],
                top = text_grob("Distibution of Percentage Individuals No Reporting Good Mental
```