Model Creation and Validation for the Social Vulnerability Index
Training Random Forest and Geographically Weighted Random Forest Models

Thesis for a Master of Public Health, Epidemiology

Nathan Garcia-Diaz

Contents

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
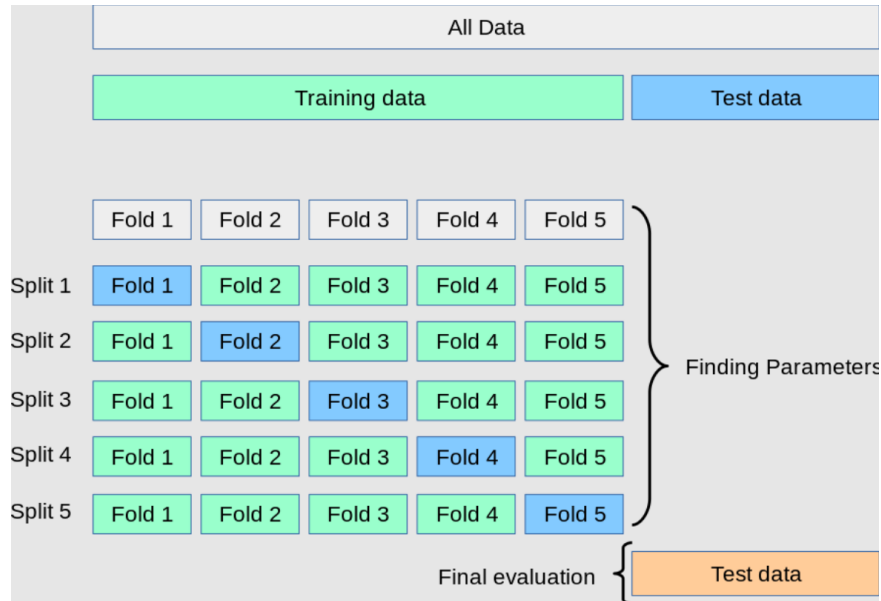
OOB Error Rate $= \frac{1}{n}\Sigma_{i=1}^{N}(y_i - \hat{y}_i^{\text{OOB}})^2$. $\hat{y}_i^{\text{OOB}}$ is the OOB prediction for the i-th observation, which is obtained by averaging the predictions from only those trees that did not include i in their bootstrap sample. To provide a high-level summary, 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:

4

- 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.

5
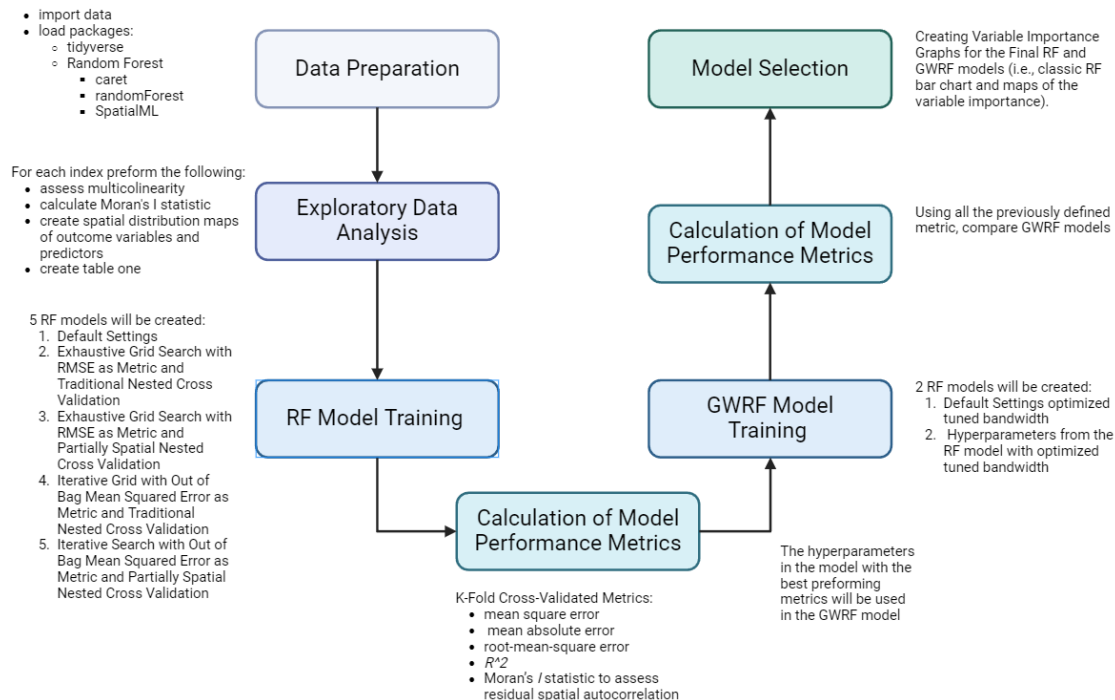
Traditional Random Forest Models: Model Building and Hyperparameter Tuning

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

Model 1: Default Settings

Background: The default settings for the RF model is $mtry = \text{p}/3 = 5$, 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: 5
##
##          Mean of squared residuals: 0.01881214
##                    % Var explained: 77.61
```

Model 2: 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

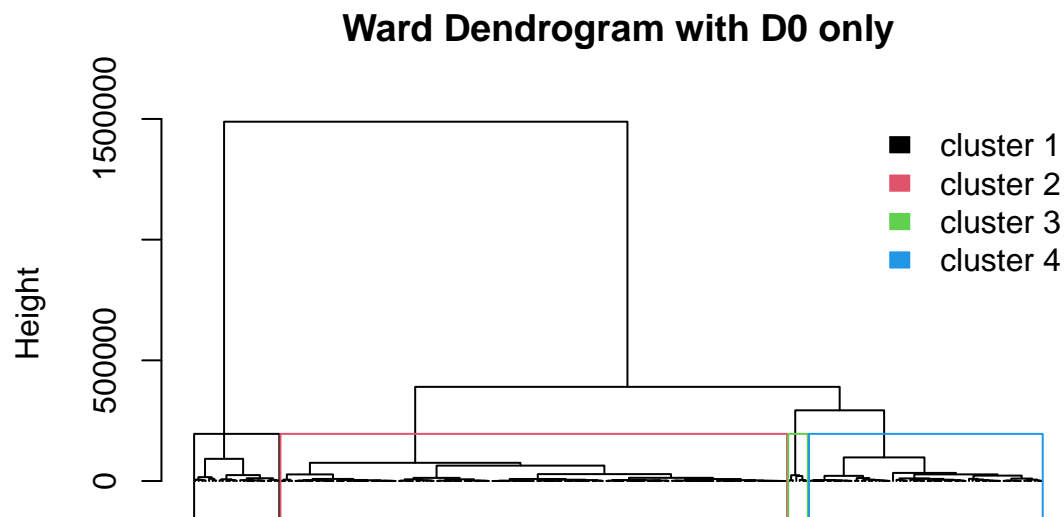| Fold	Tuned_mtry	Tuned_ntree	RMSE	MAE	R_squared
1	11	100	0.161	0.130	0.710
2	5	100	0.137	0.110	0.788
3	5	150	0.151	0.122	0.759
4	5	950	0.158	0.125	0.726
5	3	350	0.169	0.139	0.722

Model 2 has hyperparameters set to $mtry = 5$, and $ntrees = 100$.

Model 3: 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 socio-economic 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.

Ward Dendrogram with D0 only

Ward Dendrogram with D0 only

Legend:
- cluster 1 (black)
- cluster 2 (red)
- cluster 3 (green)
- cluster 4 (blue)
- cluster 5 (cyan)
- cluster 6 (black)
- cluster 7 (red)
- cluster 8 (green)

of 78%   of 16%

- based on D0
- based on D1
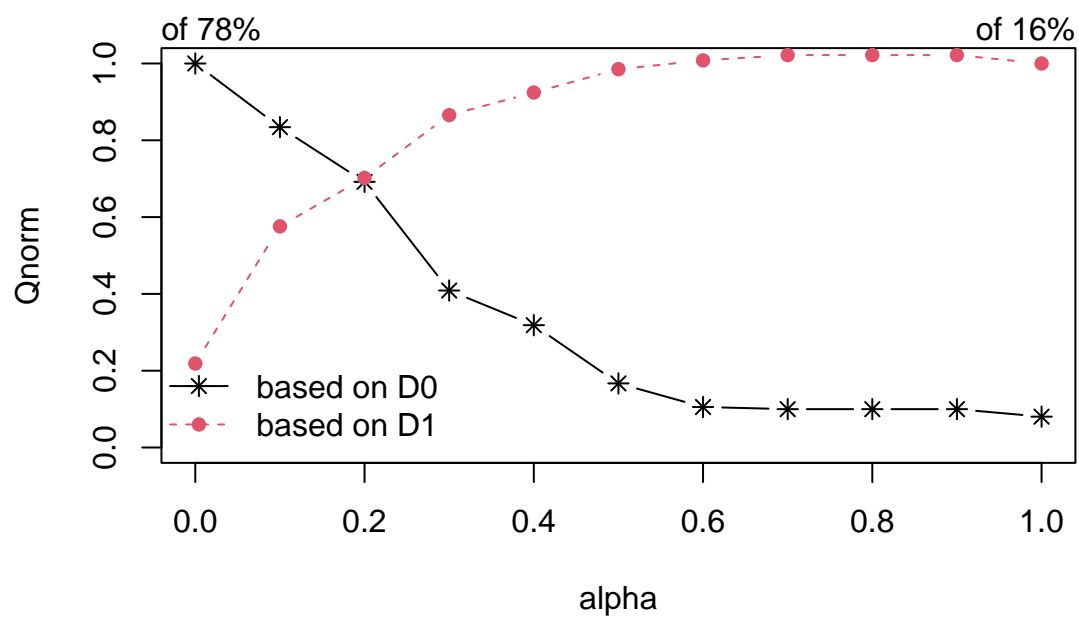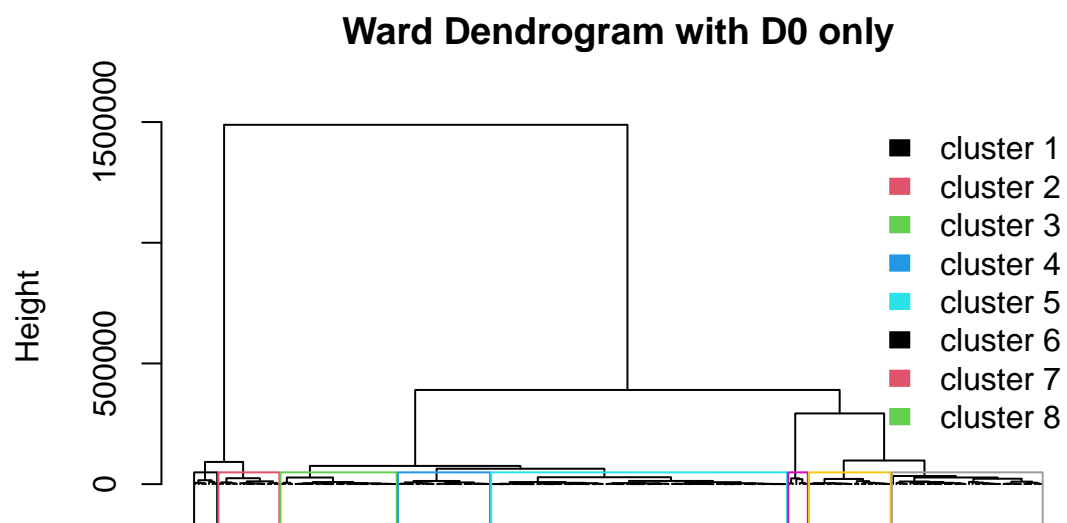
Partition P5bis obtained with alpha=0.5
and neighborhood dissimilarities
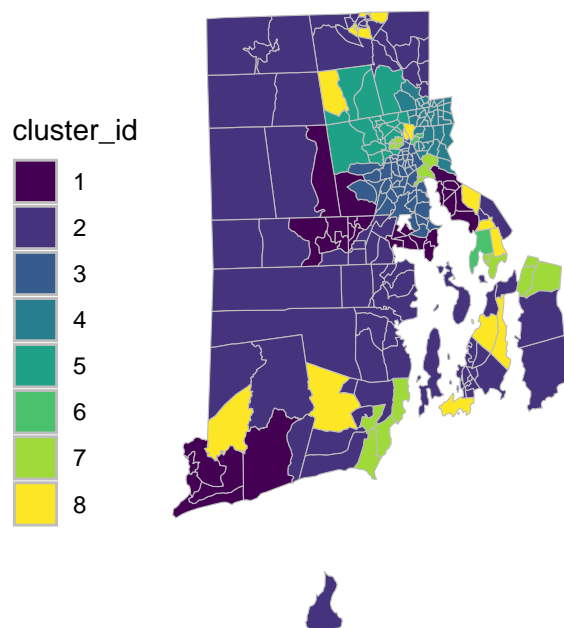
Table 2: Model 3: Partially Spatial Cross Validation: Hyperparametyer Tuning and Performance Metrics

Fold	Tuned_mtry	Tuned_ntree	RMSE	MAE	Rsquared
1	7	650	0.128	0.110	0.860
2	10	100	0.137	0.113	0.859
3	13	550	0.144	0.108	0.748
4	8	250	0.129	0.102	0.767
5	13	150	0.135	0.097	0.772
6	6	150	0.131	0.105	0.830
7	5	350	0.148	0.110	0.708
8	15	100	0.131	0.104	0.745

Model 3 has hyperparameters set to $mtry = 7$, and $ntrees = 650$.

Model 4: Grid Search 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: Hyperparameteyer Tuning and Performance Metrics

Fold	Best_mtry	Best_ntree	Test_Error	RMSE	MAE	R_squared
1	3	100	0.021	0.146	0.109	0.742
2	15	950	0.019	0.139	0.112	0.787
3	7	400	0.021	0.144	0.115	0.748
4	11	100	0.020	0.142	0.109	0.754
5	12	150	0.011	0.107	0.086	0.859

Model 4 has hyperparameters set to $mtry = 12$, and $ntrees = 150$.

Model 5: Grid 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	5	100	0.016	0.128	0.109	0.770
2	13	200	0.010	0.100	0.076	0.847
3	16	400	0.017	0.130	0.101	0.751
4	13	350	0.023	0.153	0.122	0.689
5	6	300	0.026	0.162	0.129	0.651
6	14	150	0.021	0.144	0.119	0.787
7	12	200	0.014	0.117	0.098	0.869
8	6	800	0.029	0.172	0.123	0.602

Model 5 has hyperparameters set to $mtry = 13$, and $ntrees = 200$.

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	0.137	0.111	0.787
2	5	100	NA	0.137	0.110	0.788
3	7	650	NA	0.128	0.110	0.860
4	12	150	0.011	0.107	0.086	0.859
5	13	200	0.010	0.100	0.076	0.847

Geographically Weighted Random Forest Models: Model Building and Hyperparameter Tuning

Model 6: Default Settings and Optimized Bandwidth

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

```
## [1] "Global Model Summary"

## Ranger result
##
## Call:
##  ranger(rpl_themes ~ e_pov150 + e_unemp + e_hburd + e_nohsdp +      e_uninsur + e_age65 + e_
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      246
## Number of independent variables:  16
## Mtry:                             5
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.01905588
## R squared (OOB):                  0.7741361
```

The final model hyperparameters have been set to *bandwidth* = 50, *mtry* = 5, and *ntrees* = 500.

Model 7: Best Preforming RF Parameters and Optimized Bandwidth

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

```
## [1] "Global Model Summary"

## Ranger result
##
## Call:
##  ranger(rpl_themes ~ e_pov150 + e_unemp + e_hburd + e_nohsdp +      e_uninsur + e_age65 + e
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      246
## Number of independent variables:  16
## Mtry:                             5
## Target node size:                 5
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       0.01905588
## R squared (OOB):                  0.7741361
```

The final model hyperparameters have been set to *bandwidth* = 47, *mtry* = 13, and *ntrees* = 200.

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	50	5	500	0.110	0.138	0.773
7	47	13	200	0.109	0.138	0.774

Table 7: Names for Variables

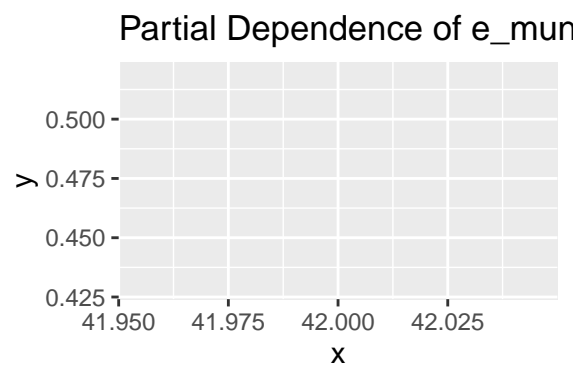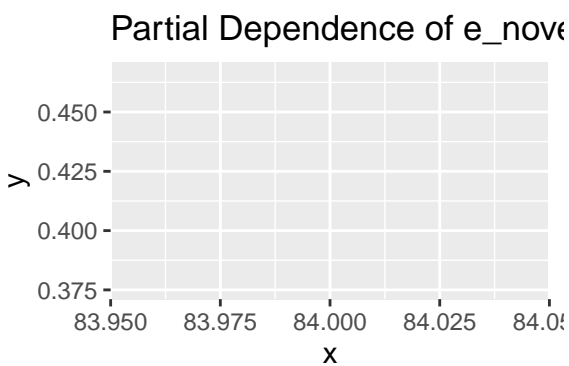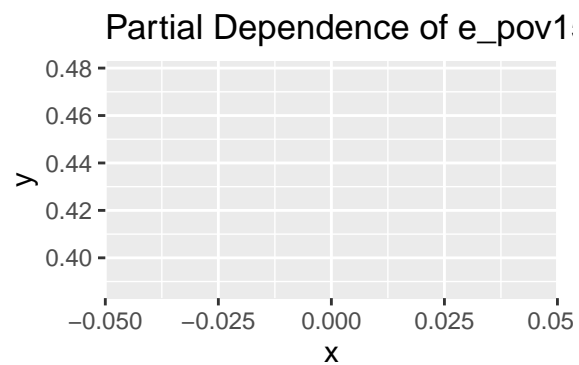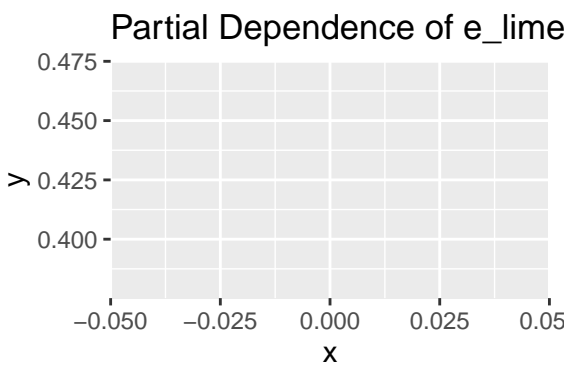| Code	Variable Name
e_pov150	Distibution of Person Below 150% Poverty Estimates
e_unemp	Distibution of Civilian (16 yrs+) Unemployed Estimates
e_hburd	Distibution of Cost Burdened Occupied Housing United ($< \$75k$)
e_nohsdp	Distibution of Persons (25 yrs+) With No High School Diploma
e_uninsur	Distibution of Uninsured in Total Civilian Non Institutionalized Population Estimate
e_age65	Distibution of Persons Aged 65 or Older Estimate
e_age17	Distibution of Persons Aged 17 or Younger Estimate
e_disabl	Distibution of Total Civilian Non Institutionalized Population with a Disability Estimate
e_sngpnt	Distribution of Single Parent Households with Children Less than 18 Estimate
e_limeng	Distribution of Person (5 yrs+) Who Speak English 'less than well' Estimate
e_minrty	Distribution of Minority Persons Estimate
e_mobile	Distribution of Mobile Homes Estimate
e_crowd	Distribution of Occupied Housing Units With More People Than Rooms Estimate
e_noveh	Distribution of Households With No Vehicles Avaiable Estimate
e_groupq	Distribution of Persons In Group Quarters Estimate

Final Random Forest Model Preformance Metrics and Results

Table 8: Performance Metrics for Each Model

Model	Bandwidth	Best_mtry	Best_ntree	RMSE	MAE	R_squared
5	NA	13	200	0.100	0.076	0.847
7	47	13	200	0.109	0.138	0.774

Results for Traditional Random Forest Model

Partial Dependency Maps are pending.

Variable Importance (Increase in %IncMSE)
Traditional Random Forest Model

Partial Dependence of e_lime

Partial Dependence of e_pov1...

Partial Dependence of e_nove...

Partial Dependence of e_mun...
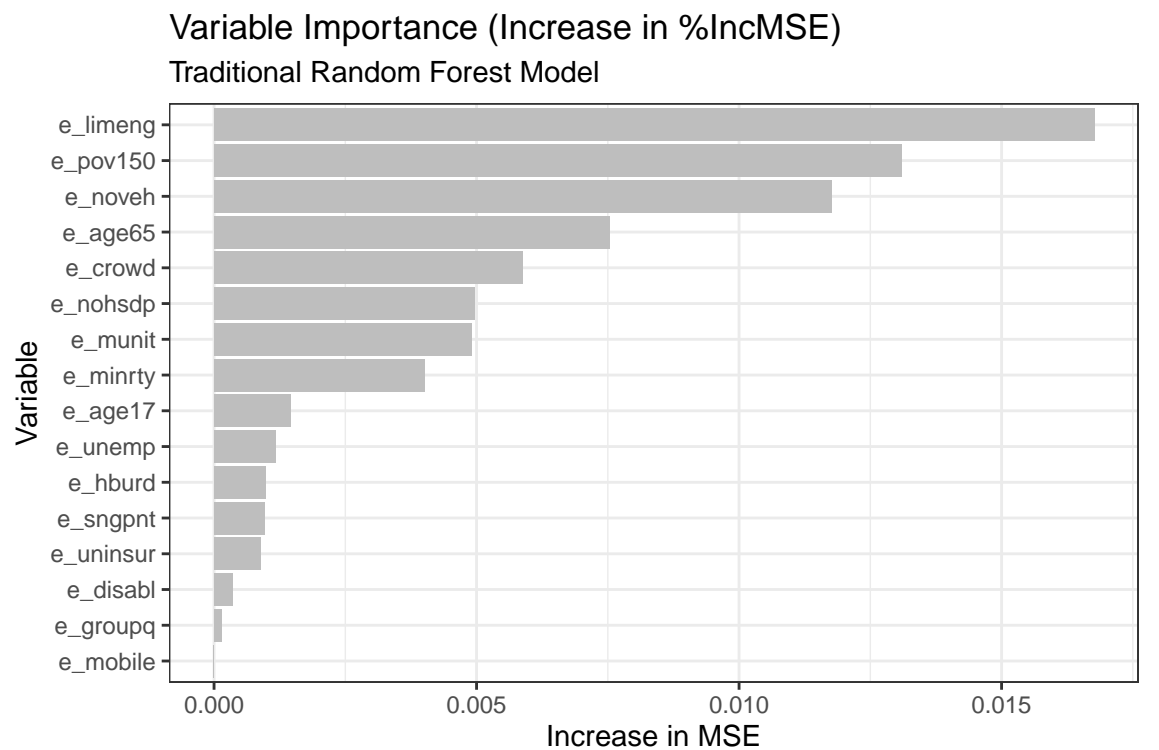
Results for Geographically Weighted Random Forest

Variable Importance (Increase in %IncMSE)

Geographically Weighted Random Forest Model

Distibution of Person Below 150% Poverty Estimates
e_pov150
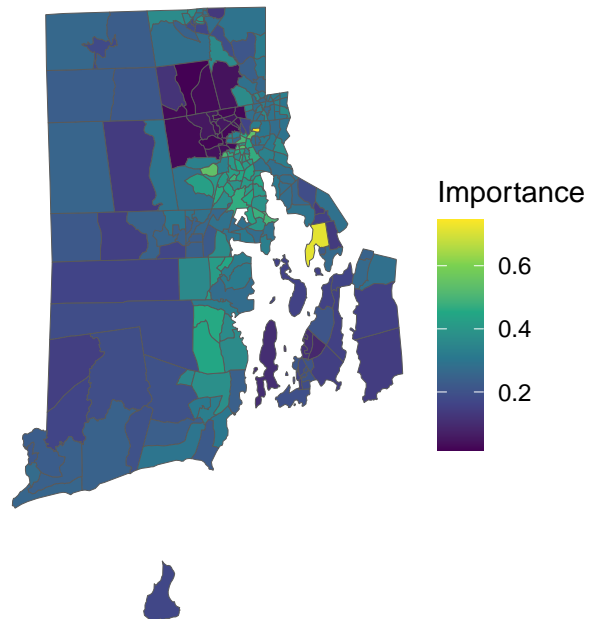
A.C.S. 2018–2022

Distibution of Civilian (16 yrs+) Unemployed Estimates
e_unemp
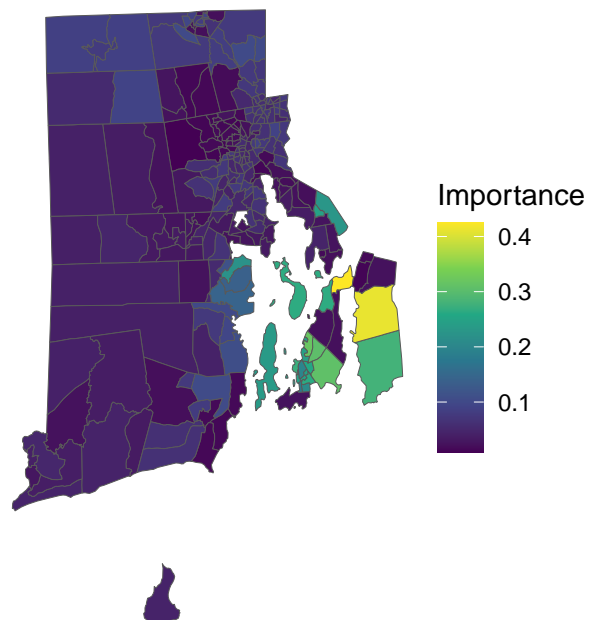
A.C.S. 2018–2022

Distibution of Cost−Burdened Occupied Housing United (< $75k)
e_hburd
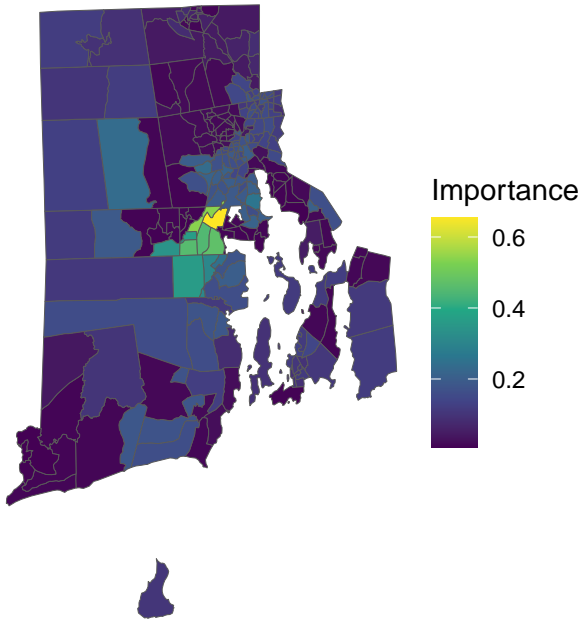
A.C.S. 2018−2022

Distibution of Persons (25 yrs+) With No High School Diploma
e_nohsdp
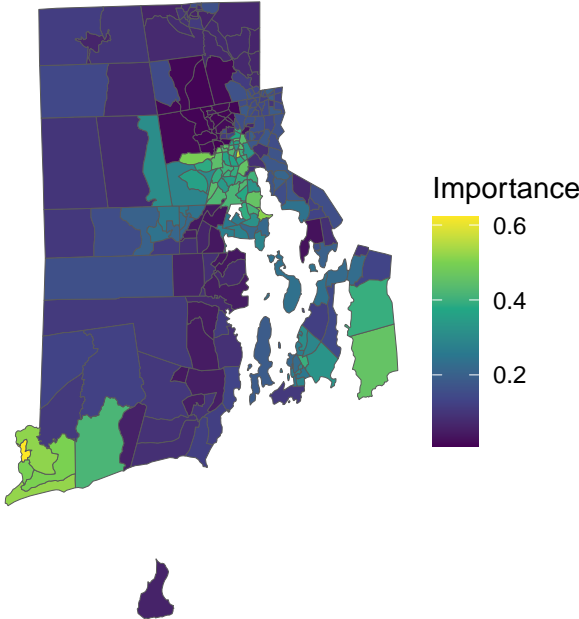
A.C.S. 2018−2022

Distibution of Uninsured in Total Civilian Non−Institutionalized Population Estimate

e_uninsur
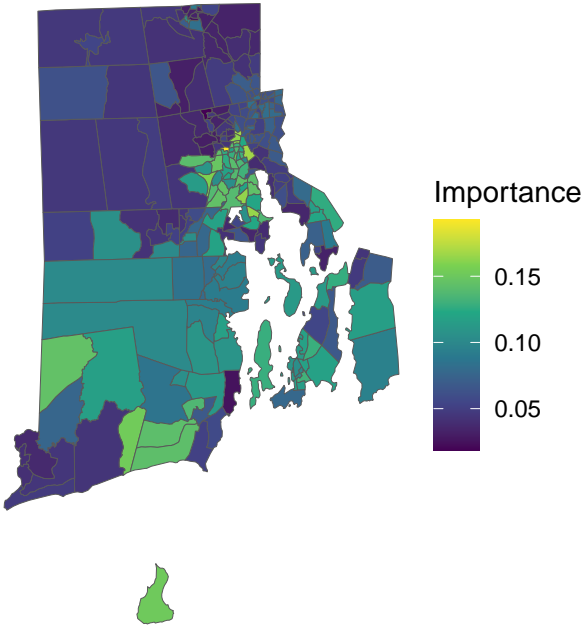
A.C.S. 2018−2022

Distibution of Persons Aged 65 or Older Estimate
e_age65
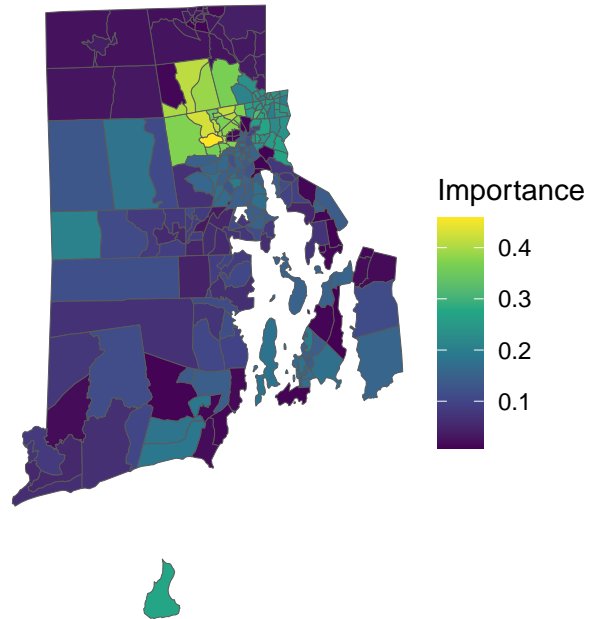
A.C.S. 2018–2022

Distibution of Persons Aged 17 or Younger Estimate
e_age17
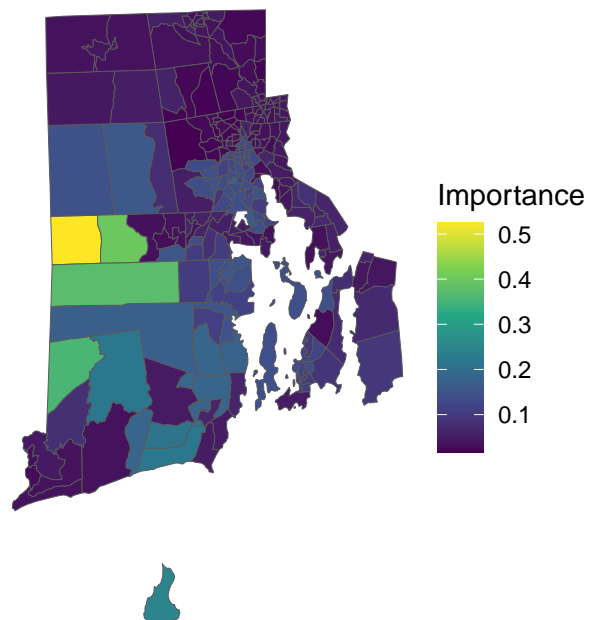
A.C.S. 2018–2022

Distibution of Total Civilian Non−Institutionalized Population with a Disability Estimate

e_disabl
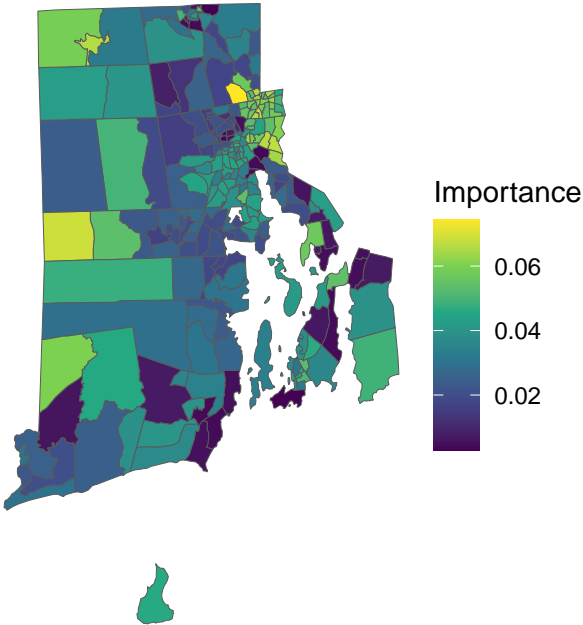
A.C.S. 2018−2022

Distribution of Single Parent Households with Children Less than 18 Estimate

e_sngpnt
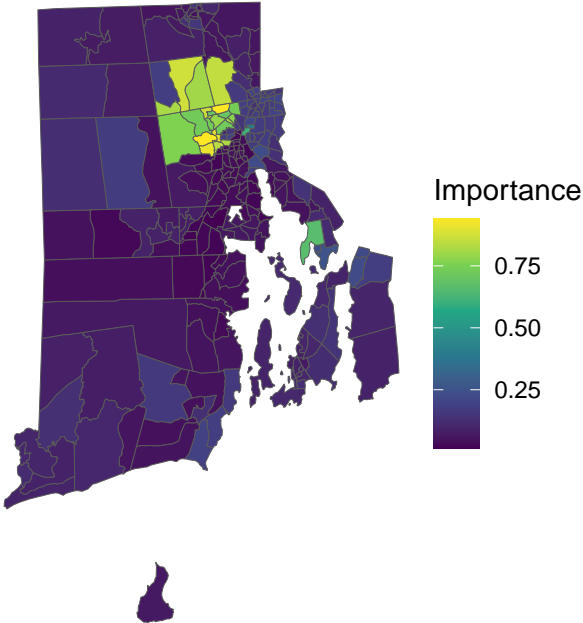
A.C.S. 2018−2022

Distribution of Person (5 yrs+) Who Speak English 'less than well' Estiamte
e_limeng
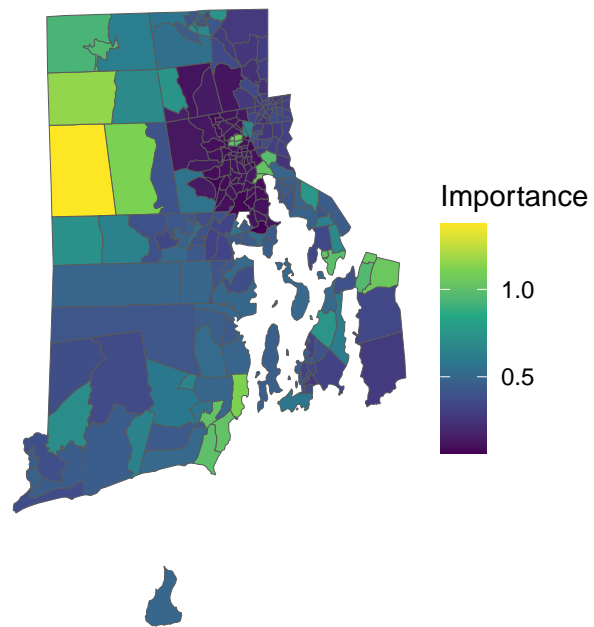
A.C.S. 2018–2022

Local Variable Importance of Racial & Ethnic Minority Status Variables

Distribution of Minority Persons Estimate
e_minrty
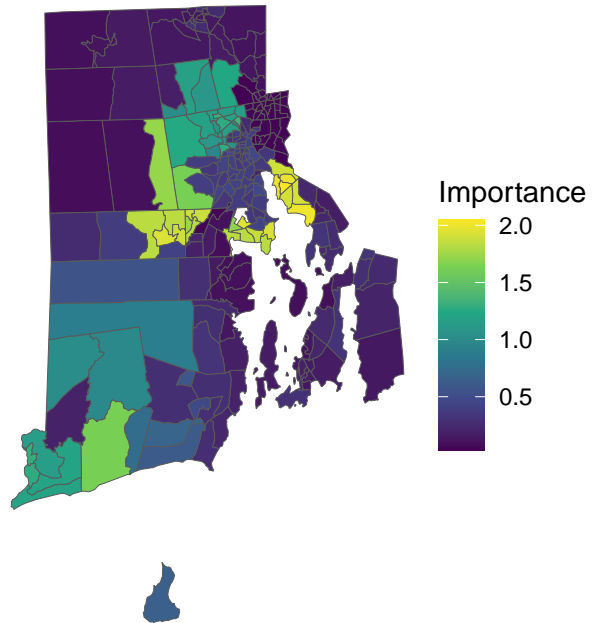
A.C.S. 2018–2022

Local Variable Importance of Housing Type & Transportation Variables

Distribution of Housing in Structures with 10 or More Units Estimate
e_munit
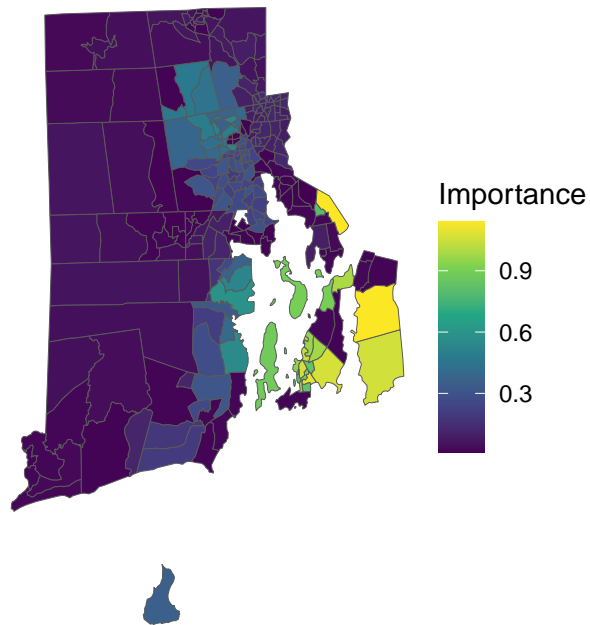
A.C.S. 2018–2022

Distribution of Mobile Homes Estimate
e_mobile
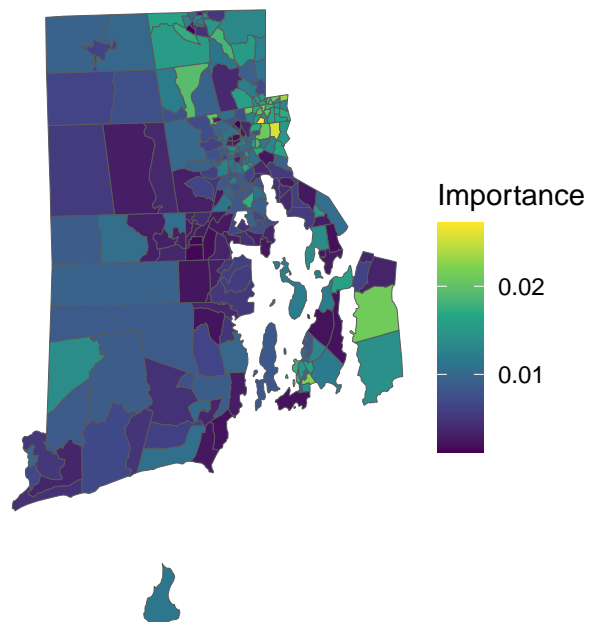
A.C.S. 2018–2022

Distribution of Occupied Housing Units With More People Than Rooms Estimate

e_crowd
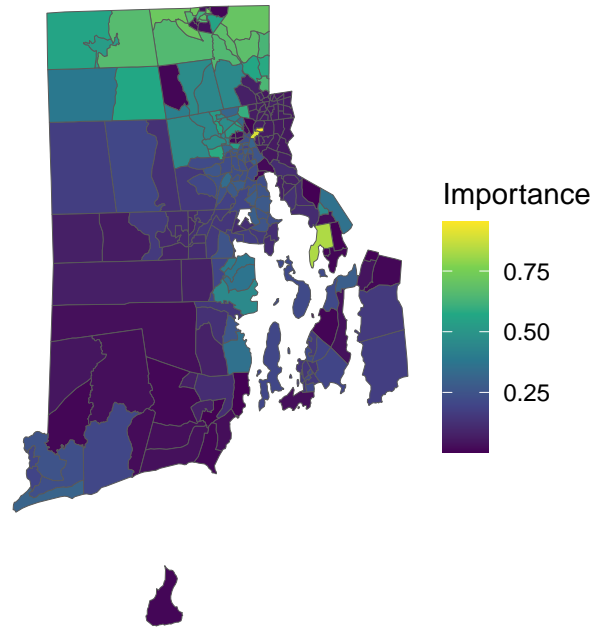
A.C.S. 2018–2022

Distribution of Households With No Vehicles Avaiable Estimate

e_noveh
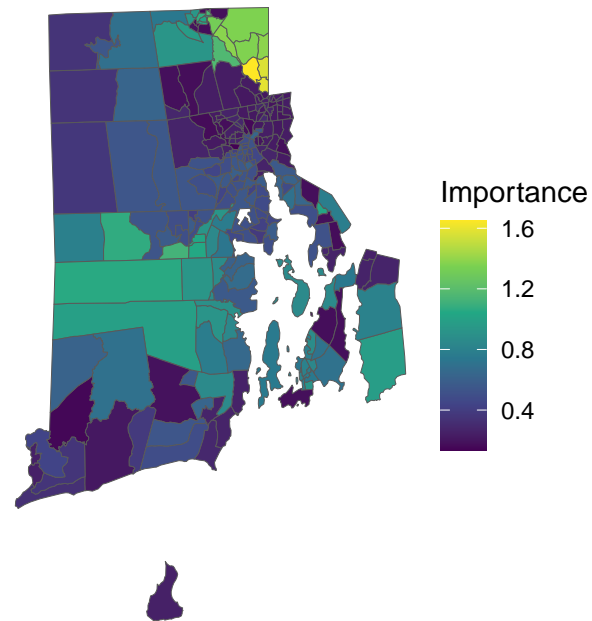
A.C.S. 2018–2022

Distribution of Persons In Group Quarters Estimate
e_groupq
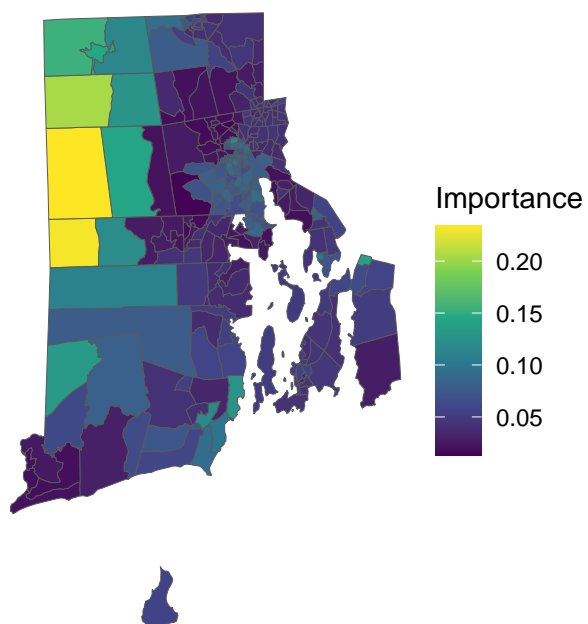
A.C.S. 2018–2022

Code Appendix

```r
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/Meth
####################
#### 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)

### setting seed
set.seed(926)

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

### obtaining SPH files for RI tracts
tracts = tracts(state = "RI", year = 2022, cb = TRUE)

### joining data
```

```r
map = inner_join(tracts, svi_df, by = c("GEOID" = "fips")) %>%
  select(rpl_themes, starts_with("e_"))
# keep a copy for later
map_map = map

### defining analytical coordinates and df
df_coords = map %>%
  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 = svi_df %>%
  st_drop_geometry() %>%
  select(rpl_themes, starts_with("e_"))

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

# creating the first model
# cross validated evaluation
cl = makeCluster(detectCores() - 1)  # Use one less core than available
registerDoParallel(cl)
```

```r
rf_mod1 = train(rpl_themes ~ e_pov150 + e_unemp + e_hburd +
    e_nohsdp + e_uninsur + e_age65 + e_age17 +
    e_disabl + e_sngpnt + e_limeng + e_minrty +
    e_munit + e_mobile + e_crowd + e_noveh + e_groupq,
                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_themes, k = 5)
```

```r
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
grid = expand.grid(.mtry = c(1:16),
                   .ntree = c(100, 150, 200, 250,
                              300, 350, 400, 450,
```

```r
                                    500, 550, 600, 650,
                                    700, 750, 800, 850,
                                    900, 950, 1000))

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

k = length(outer_folds)

# run model
model2_results = nested_cv(
  form = rpl_themes ~ e_pov150 + e_unemp + e_hburd +
    e_nohsdp + e_uninsur + e_age65 + e_age17 +
    e_disabl + e_sngpnt + e_limeng + e_minrty +
    e_munit + e_mobile + e_crowd + e_noveh + e_groupq,
  response_var = "rpl_themes",
  data = df,
  method = customRF,
  trControl = ctrl,
  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
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_Fold2 = tab %>%
  filter(RMSE == min(RMSE)) %>%
  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 Perform
      digits = 3,
      align = c("lllccc"))

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

### this code c
```

```r
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 continue with k = 8 because
#the large cluster found in k = 4 graphs
#colored in red contains the branch that is less homogeneous or
#the distance between clsutesr within the branch is smaller.
#Additionally since the the gaol is cross validation
#obtaining

#Graphs 4 are illustrated to should the largest vertical
#distance, while 8 was choosen to emphasis the equal groups


# k=4
plot(tree, hang = -1, label = FALSE,
     xlab = "", sub = "",
     main = "Ward Dendrogram with D0 only")
rect.hclust(tree ,k = 4, border = c(1:4))
legend("topright", legend = paste("cluster", 1:4),
       fill=1:5, bty="n", border = "white")
# k=8
plot(tree, hang = -1, label = FALSE,
     xlab = "", sub = "",
     main = "Ward Dendrogram with D0 only")
rect.hclust(tree ,k = 8, border = c(1:8))
legend("topright", legend = paste("cluster", 1:8),
       fill=1:5, 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)
k = 8
cr = choicealpha(D0, D1,
```

```r
                 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 = 0.2

# here the plot seggust to choose alpha = 0.2
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=0.5
          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, k) {
  outer_folds = createFolds(data[[cluster_col]], k = length(unique(data[[cluster_col]])), retur

  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
```

```r
    predictions <- predict(inner_model, newdata = test_data)
    performance_metric <- postResample(pred = predictions, obs = test_data$rpl_themes)

    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)
# run model
model3_results = spatial_nested_cv(
  form = rpl_themes ~ e_pov150 + e_unemp + e_hburd +
    e_nohsdp + e_uninsur + e_age65 + e_age17 +
    e_disabl + e_sngpnt + e_limeng + e_minrty +
    e_munit + e_mobile + e_crowd + e_noveh + e_groupq,
  data = map,
  method = customRF,
  trControl = ctrl,
  tuneGrid = grid,
  cluster_col = "cluster_id")
# parrellel clustering
unregister_dopar()
Fold = c(1:8)

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]),
              as.numeric(model3_results[[6]]$model$bestTune[1]),
              as.numeric(model3_results[[7]]$model$bestTune[1]),
              as.numeric(model3_results[[8]]$model$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]),
```

```r
                    as.numeric(model3_results[[4]]$model$bestTune[2]),
                    as.numeric(model3_results[[5]]$model$bestTune[2]),
                    as.numeric(model3_results[[6]]$model$bestTune[2]),
                    as.numeric(model3_results[[7]]$model$bestTune[2]),
                    as.numeric(model3_results[[8]]$model$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]]$performance[[1]]),
         as.numeric(model3_results[[6]]$performance[[1]]),
         as.numeric(model3_results[[7]]$performance[[1]]),
         as.numeric(model3_results[[8]]$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]]$performance[[2]]),
         as.numeric(model3_results[[6]]$performance[[2]]),
         as.numeric(model3_results[[7]]$performance[[2]]),
         as.numeric(model3_results[[8]]$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]]$performance[[3]]),
         as.numeric(model3_results[[6]]$performance[[3]]),
         as.numeric(model3_results[[7]]$performance[[3]]),
         as.numeric(model3_results[[8]]$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)) %>%
```

```r
  pull(RMSE)

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

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

Best_Fold3 = 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))
```

```r
}

# 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
  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 - out of bag error rates
    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 = FALSE,
                            importance = TRUE)
      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$ntree,
                                mtry = best_params$mtry,
                                importance = TRUE)
```

```r
    # 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)
```

```r
# specify the predictor (x) and outcome (y) object
x = df %>% select(starts_with("e_"))
y = df %>% pull(rpl_themes)

# 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
#plot(model4_models[[5]]$mse, type = "l", main = "OOB Error vs Number of Trees", xlab = "Numbe
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_Fold4 = model4_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  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()
  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.forest = FALSE, importance = TRUE)
      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$ntree,
                                mtry = best_params$mtry,
                                importance = TRUE)
```

```r
    # 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)
```

```r
# specify the predictor (x) and outcome (y) object
cluster_id = map %>% select(cluster_id) %>% st_drop_geometry()
x = map %>% select(starts_with("e_")) %>% st_drop_geometry()
y = map %>% pull(rpl_themes) %>% 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_themes ~ e_pov150 + e_unemp + e_hburd +
    e_nohsdp + e_uninsur + e_age65 + e_age17 +
    e_disabl + e_sngpnt + e_limeng + e_minrty +
    e_munit + e_mobile + e_crowd + e_noveh + e_groupq,
  data = map,
  response_var = "rpl_themes",
  num_predictors = 16,
  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)) %>%
```

```r
  pull(R_squared)

Best_Fold5 = model5_results %>%
  filter(Test_Error == min(Test_Error)) %>%
  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
rf_model_1 = rf_mod1$finalModel
rf_model_2 = model2_results[[Best_Fold2]]$model$finalModel
rf_model_3 = model3_results[[Best_Fold3]]$model$finalModel
rf_model_4 = model4_models[[Best_Fold4]]
rf_model_5 = model5_models[[Best_Fold5]]

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 %>%
```

```r
  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 = "svi_rf_model_1.RData")
save(rf_model_2, file = "svi_rf_model_2.RData")
save(rf_model_3, file = "svi_rf_model_3.RData")
save(rf_model_4, file = "svi_rf_model_4.RData")
save(rf_model_5, file = "svi_rf_model_5.RData")
####################
#### GWRF Mod 1 ####
####################

# testing for optimal bandwidth
temp = SpatialML::grf.bw(rpl_themes ~ e_pov150 + e_unemp + e_hburd + e_nohsdp +
                             e_uninsur + e_age65 + e_age17 + e_disabl +
                             e_sngpnt + e_limeng + e_minrty + e_munit +
                             e_mobile + e_crowd + e_noveh + e_groupq,
                         dataset = df,
                         kernel = "adaptive",
                         bw.min = 20,
                         bw.max = 50,
                         coords = df_coords,
                         trees = 500,
                         mtry = 5,
                         step = 1)
best.bw_gwrf_mod1 = temp$Best.BW

# defining the spatial model with prior model hyperparameters
gwrf_mod1 = SpatialML::grf(rpl_themes ~ e_pov150 + e_unemp + e_hburd + e_nohsdp +
                             e_uninsur + e_age65 + e_age17 + e_disabl +
                             e_sngpnt + e_limeng + e_minrty + e_munit +
                             e_mobile + e_crowd + e_noveh + e_groupq,
                         dframe = df,
                         kernel = "adaptive",
                         coords = df_coords,
                         bw = best.bw_gwrf_mod1,
```

```
                              ntree = 500,
                              mtry = 5)
"Global Model Summary"
gwrf_mod1$Global.Model
####################
#### GWRF Mod 2 ####
####################

# testing for optimal bandwidth
temp = SpatialML::grf.bw(rpl_themes ~ e_pov150 + e_unemp + e_hburd + e_nohsdp +
                              e_uninsur + e_age65 + e_age17 + e_disabl +
                              e_sngpnt + e_limeng + e_minrty + e_munit +
                              e_mobile + e_crowd + e_noveh + e_groupq,
                         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 hyparameters
gwrf_mod2 = SpatialML::grf(rpl_themes ~ e_pov150 + e_unemp + e_hburd + e_nohsdp +
                              e_uninsur + e_age65 + e_age17 + e_disabl +
                              e_sngpnt + e_limeng + e_minrty + e_munit +
                              e_mobile + e_crowd + e_noveh + e_groupq,
                         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
"Global Model Summary"
gwrf_mod1$Global.Model
####################
#### GWRF Models ###
####################
# Model 6
predictions6 = gwrf_mod1$Global.Model$predictions
mse6 = mean((df$rpl_themes - predictions6)^2)
rmse6 = sqrt(mse6)
```

```r
mae6 = sum(abs(df$rpl_themes - predictions6))/length(predictions6)
r_squared6 = 1 - sum((df$rpl_themes - predictions6)^2) / sum((df$rpl_themes - mean(df$rpl_theme

# Model 7
predictions7 = gwrf_mod2$Global.Model$predictions
mse7 = mean((df$rpl_themes - predictions7)^2)
rmse7 = sqrt(mse7)
mae7 = sum(abs(df$rpl_themes - predictions7))/length(predictions7)
r_squared7 = 1 - sum((df$rpl_themes - predictions7)^2) / sum((df$rpl_themes - mean(df$rpl_theme

# 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")
a = c("Code", "Variable Name")
# Local Variable Importance of SocioEconomic Status Variables
b = c("e_pov150", "Distibution of Person Below 150% Poverty Estimates")
c = c("e_unemp", "Distibution of Civilian (16 yrs+) Unemployed Estimates")
d = c("e_hburd", "Distibution of Cost Burdened Occupied Housing United (< $75k)")
e = c("e_nohsdp", "Distibution of Persons (25 yrs+) With No High School Diploma")
f = c("e_uninsur", "Distibution of Uninsured in Total Civilian Non Institutionalized Population
# Local Variable Importance of Household Characteristics Variables
g = c("e_age65", "Distibution of Persons Aged 65 or Older Estimate")
h = c("e_age17", "Distibution of Persons Aged 17 or Younger Estimate")
i = c("e_disabl", "Distibution of Total Civilian Non Institutionalized Population with a Disab
j = c("e_sngpnt", "Distribution of Single Parent Households with Children Less than 18 Estimate
k = c("e_limeng", "Distribution of Person (5 yrs+) Who Speak English 'less than well' Estimate
# Local Variable Importance of Racial & Ethnic Minority Status Variables
l = c("e_minrty", "Distribution of Minority Persons Estimate")
# Local Variable Importance of Housing Type & Transportation Variables
m = c("e_mobile", "Distribution of Mobile Homes Estimate")
n = c("e_crowd", "Distribution of Occupied Housing Units With More People Than Rooms Estimate")
o = c("e_noveh", "Distribution of Households With No Vehicles Avaiable Estimate")
p = c("e_groupq", "Distribution of Persons In Group Quarters Estimate")
temp_tab = rbind(a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p)
```

```r
rownames(temp_tab) = NULL
kable(temp_tab,
      caption = "Names for Variables")
####################
#### 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(model_results[[Best_Fold]]$model$finalModel$importance)
temp = as.data.frame(rf_model_5$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_5
a = as.data.frame(randomForest::partialPlot(rf_mod, df, e_limeng, plot = FALSE))
b = as.data.frame(randomForest::partialPlot(rf_mod, df, e_noveh, plot = FALSE))
c = as.data.frame(randomForest::partialPlot(rf_mod, df, e_pov150, plot = FALSE))
d = as.data.frame(randomForest::partialPlot(rf_mod, df, e_minrty, 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")
```

```r
# 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
map_map = map_map %>%
  mutate(grf_pred = gwrf_mod2$Global.Model$predictions)

# Predicted 1:1 Plot
ggplot(map_map, aes(x = grf_pred, y = rpl_themes)) +
  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_themes)) +
  geom_sf() +
  theme_void() +
  scale_fill_viridis_c() +
  labs(title = "Observed SVI 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 SVI 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
```

```r
#variable_codes = temp_tab[-1, 1]
#variable_descriptions = temp_tab[-1, 2]
map_variable_importance = map_map %>%
  select(-rpl_themes, -starts_with("e_"), -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) {
  #var_name = variable_codes[i]
  #var_desc = variable_descriptions[i]

  # Generate the map using ggplot2
  p = ggplot(data=map_variable_importance, aes_string(fill = var_name)) +
    geom_sf(lwd=0.01) +
    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 Variable
annotate_figure(plot_lst[["e_pov150"]],
                top = text_grob("Distibution of Person Below 150% Poverty Estimates", size = 14
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_unemp"]],
                top = text_grob("Distibution of Civilian (16 yrs+) Unemployed Estimates", size
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_hburd"]],
                top = text_grob("Distibution of Cost-Burdened Occupied Housing United (< $75k)"
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_nohsdp"]],
                top = text_grob("Distibution of Persons (25 yrs+) With No High School Diploma"
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_uninsur"]],
                top = text_grob("Distibution of Uninsured in Total Civilian Non-Institutionaliz
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
############### Household Characteristics
annotate_figure(plot_lst[["e_age65"]],
                top = text_grob("Distibution of Persons Aged 65 or Older Estimate", size = 14)
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_age17"]],
                top = text_grob("Distibution of Persons Aged 17 or Younger Estimate", size = 14
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
```

```r
annotate_figure(plot_lst[["e_disabl"]],
                top = text_grob("Distibution of Total Civilian Non-Institutionalized Population
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_sngpnt"]],
                top = text_grob("Distribution of Single Parent Households with Children Less tha
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_limeng"]],
                top = text_grob("Distribution of Person (5 yrs+) Who Speak English 'less than w
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
################ Racial & Ethnic Minority Status
annotate_figure(plot_lst[["e_minrty"]],
                top = text_grob("Distribution of Minority Persons Estimate", size = 12),
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
################ Housing Type & Transportation
annotate_figure(plot_lst[["e_munit"]],
                top = text_grob("Distribution of Housing in Structures with 10 or More Units Es
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_mobile"]],
                top = text_grob("Distribution of Mobile Homes Estimate", size = 14),
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_crowd"]],
                top = text_grob("Distribution of Occupied Housing Units With More People Than I
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_noveh"]],
                top = text_grob("Distribution of Households With No Vehicles Avaiable Estimate"
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
annotate_figure(plot_lst[["e_groupq"]],
                top = text_grob("Distribution of Persons In Group Quarters Estimate", size = 14
                fig.lab = "A.C.S. 2018-2022", fig.lab.pos = c("bottom.right"), fig.lab.size = 9
```