Electronic Supplementary Material (ESI) for Environmental Science: Atmospheres.
This journal is © The Royal Society of Chemistry 2022

Supplemental Document

Application of Machine Learning and Statistical Modeling to Identify Sources of Air Pollutant Emissions in Kitchener, Ontario, Canada

Wisam Mohammed[1], Adrian Adamescu[1], Lucas Neil[2], Nicole Shantz[2], Tom Townend[3], Martin Lysy[4*], and Hind A. Al-Abadleh[1*]

[1]Department of Chemistry and Biochemistry, Wilfrid Laurier University, 75 University Ave West, Waterloo, ON N2L 3C5, Canada

[2] Ausenco, 100 – 1016B Sutton Dr, Burlington, Ontario L7L 6B8, Canada

[3] AQMesh, Environmental Instruments Ltd., Unit 5, The Mansley Centre, Timothy's Bridge Road, Stratford-upon-Avon CV37 9NQ, UK

[4] Department of Statistics and Actuarial Science, University of Waterloo, 200 University Ave West, Waterloo, Ontario N2L 3G1, Canada

Date: October 11, 2022

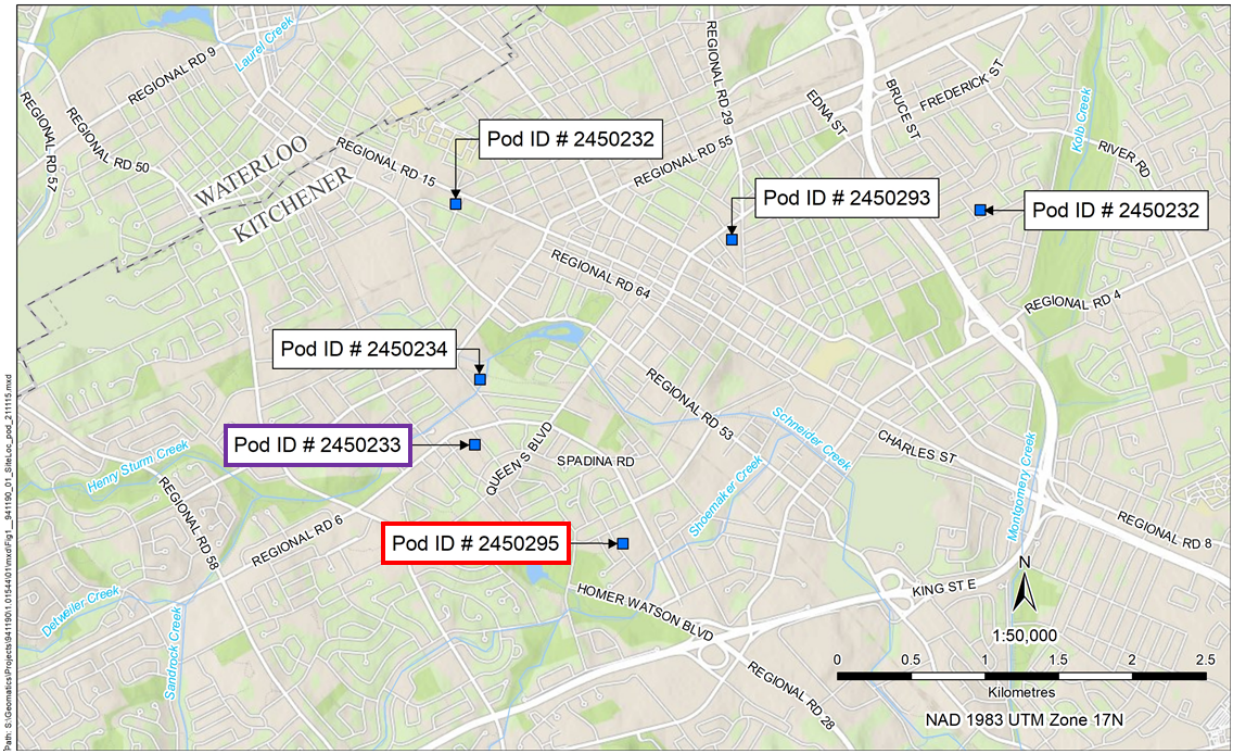


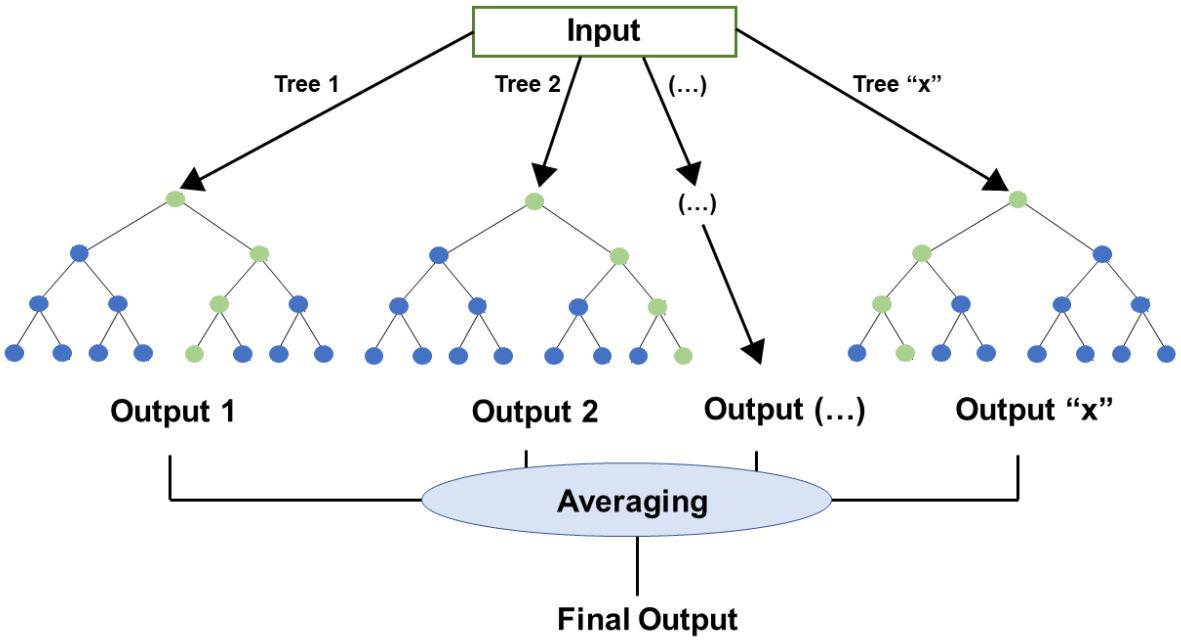
Figure S1. Spatial location of low-cost sensors in *Kitchener, ON*. Pod 2450233 (Purple) and Pod 2450293 are located near major roads. Pod 2450232 was located downwind from a factory (western location, Sept. 2020 to Dec. 2020), and was moved to near a main road (eastern location, Dec. 2020-present). Pod 2450295 (Red) is located near Highway 8, and the co-located pod (Pod 2450234) is located within 30 meters of the reference station used by the MECP. See Table S1 for additional details on the coordinates and associated location. Reproduced under the CC-BY license from reference W. Mohammed, N. Shantz, L. Neil, T. Townend, A. Adamescu and H. A. Al-Abadleh, Air Quality Measurements in Kitchener, Ontario, Canada using Multisensor Mini Monitoring Stations, *Atmosphere*, 2022, **13**, doi:10.3390/atmos13010083.

Table S1. Classification, UTM Coordinate location, and elementary school associated with each sensor pod in the low-cost sensor network. Reproduced under the CC-BY license from reference W. Mohammed, N. Shantz, L. Neil, T. Townend, A. Adamescu and H. A. Al-Abadleh, Air Quality Measurements in Kitchener, Ontario, Canada using Multisensor Mini Monitoring Stations, Atmosphere, 2022, 13, doi:10.3390/atmos13010083.

Pod Number	**Pod ID**	**UTM Coordinates**			**Associated Location**
		Zone	**Easting**	**Northing**	
Pod 1	2450295	17 T	541107	4809056	St. Bernadette (School)
Pod 2	2450233	17 T	540152	4809823	JF Carmichael (School)
Pod 3	2450293	17 T	541824	4811246	Suddaby (School)
Pod 4	2450232	17 T	540000	4811398	King Edward (School)[1]
		17 T	543590	4811325	Smithson (School)
Pod 5	2450234	17 T	540140	4810258	Victoria Park (City Park)[2]

[1] The data collected from Pod 3 (west side of the main road) showed high levels of pollutant emissions. To further validate that these observations were in fact real, Pod 4 was moved in December 2020 from near a rubber factory (King Edward) to the east side of this main road (Smithson) and remains at this second location to this day.

[2] The co-located pod (Pod 5: ID 2450234) was located within a 30-meter range of the regional reference station.



Schematic S1: Random Forest model: sample process (Recreated under the creative commons license from reference K. Guo, X. Wan, L. Liu, Z. Gao and M. Yang, Fault Diagnosis of Intelligent Production Line Based on Digital Twin and Improved Random Forest, *J. Appl. Sci.*, 2021, **11**, 7733. https://doi.org/10.3390/app11167733)

R code for fitting Random Forest model and calculating %IncMSE:

```
#--- load required packages ---------------------------------------------
---
require(tidyr)
require(dplyr)
require(readr)
require(readxl)
require(parallel)
require(randomForestSRC)

#--- helper functions ---------------------------------------------------
---

#' Helper function to standardize file names.
#'
#' @param path Path to the file.
#' @param ... Elements of the file name, to be concatenated into a single
```

```
string separated by "_".
#' @param ext File extension.
#' @return A string representing the file name.
get_filename <- function(path, ..., ext = "rds") {
  fname <- paste0(c(...), collapse = "_")
  file.path(path, paste0(fname, ".", ext))
}



#' Generate a specific parallel random seed.
#'
#' @param par_seed Positive integer specifying the parallel seed.
#' @param base_seed Integer specifying the base seed.
#'
#' @return Nothing; called for the side effect of setting `.Random.seed`.
#'
#' @details In principle, `par_seed` should be less than `2^64 ~ 1.8e+19`.
However, the calculation involves a for-loop of length `par_seed` (but with
constant storage), which takes on the order of 1 second for `par_seed =
1e+06`.
#'
#' Assumes that `RNGkind("L'Ecuyer-CMRG")` has been set.
parallel_set_seed <- function(par_seed, base_seed = 0) {
  set.seed(base_seed)
  current_seed <- .Random.seed
  for(i in 1:par_seed) {
    current_seed <- parallel::nextRNGStream(current_seed)
  }
  .Random.seed <<- current_seed
  invisible(NULL)
}

#' Random forest calculations for a given job.
#'
#' @param data Tibble where row is data and columns are Pollutant, School,
Year, Concentration, then list of meteorological and anthropomorphic
variables.
#' @param school School in `school_names`.
#' @param year Year; either "2020", "2021" or "both".
#' @param pollutant Name of pollutant variable.
#' @param ntree Number of trees for random forest.
#' @param ntree_tune Number of trees for tuning.  See 'Details'.
#'
#' @return A list with elements:
#' - `school`, `year`, `pollutant`: The input values of these variables.
#' - `rf_mtry`: The value of `mtry` selected by tuning procedure.
```

```
#' - `rf_nodezise`: The value of `nodesize` selected by tuning procedure.
#' - `rf_mspe`: The mean square prediction error for the random forest model.
#' - `lm_mspe`: The mean square prediction error for the linear model.
#' - `rf_predict`: The out-of-sample predictions for the random forest model.
#' - `lm_predict`: The out-of-sample predictions for the linear model.
#' - `test_obs`: The observed out-of-sample pollutant concentrations.
#' - `test_ind`: Logical vector indicating which of the rows were used for
testing.
#' - `rf_predict_full`: The in-sample predictions for the random forest model
on the full dataset.
#' - `rf_predict_perm`: The variable-permuted predictions for the random
forest  model on the full dataset.  A tibble with columns consisting of the
meteorological and traffic variables.
#' - `perm_ind`: The permutation index shared by all variables.
#' - `rf_importance`: The variable importance metrics for the random forest
model on the full dataset.  See 'Details'.
#'
#' @details
#'
#' The number of variables per split `mtry` and the minimum size of the
terminal node `nodesize` are determined by minimizing the out-of-bag (OOB)
MSE, as implemented by `randomForestSRC::tune()`.
#'
#' Standard errors for the variable importance metric (VIMP) are computed
using the subsampling and jacknife procedures described by Ishawan & Lu
(2019) and implemented by `randomForestSRC::subsample()`.  The VIMP for
variable j is defined as
#' ```
#' (MSPE_OOB(variable j permuted) - MSPE_OOB(full model)) / var(response) x
100 %,
#' ```
#'
#' @references Ishawan, H., Lu, M. (2019) "Standard errors and confidence
intervals for variable importance in random forest regression,
classification, and survival", Statistics & Medicine 38(4): 558-582.
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6279615/>.
#'
#' @importFrom randomForestSRC rfsrc,tune,subsample,extract.subsample
rf_calc <- function(data, school, year, pollutant, ntree, ntree_tune = 500) {
  # subset the data
  if(year != "both") {
    data_sub <- data %>%
      filter(School == school, Year == as.numeric(year), Pollutant ==
pollutant)
  } else {
    data_sub <- data %>%
```

```
    filter(School == school, Pollutant == pollutant)
}
data_sub <- data_sub %>%
  select(-School, -Year, -Pollutant) %>%
  as.data.frame() # randomForestSRC doesn't allow tibbles
# tuning of mtry and nodesize
message("Tuning mtry and nodesize ...")
tm <- system.time({
  rf_tune <- tune(Concentration ~ .,
                  data = data_sub,
                  ntreeTry = ntree_tune,
                  maxIter = 50)
  mtry_opt <- as.numeric(rf_tune$optimal["mtry"])
  nodesize_opt <- as.numeric(rf_tune$optimal["nodesize"])
})
message("Time elapsed: ", round(tm[3], 1), " seconds")
# training data split
n_obs <- nrow(data_sub)
train_ind <- rep(FALSE, n_obs)
train_ind[sample(1:n_obs, size = floor(.7 * n_obs))] <- TRUE
message("Fitting RF on training data...")
tm <- system.time({
  rf_model <- rfsrc(Concentration ~ .,
                    data = data_sub[train_ind,],
                    mtry = mtry_opt,
                    nodesize = nodesize_opt,
                    # fastest when no importance metric needed
                    block.size = NULL,
                    ntree = ntree)
})
message("Time elapsed: ", round(tm[3], 1), " seconds")
# fit a linear model as well
lm_model <- lm(Concentration ~ ., data = data_sub[train_ind,])
# model predictions
rf_predict <- predict(rf_model, newdata = data_sub[!train_ind,])$predicted
lm_predict <- predict(lm_model, newdata = data_sub[!train_ind,])
# mspe
pollutant_test <- data_sub$Concentration[!train_ind]
rf_mspe <- mean((rf_predict - pollutant_test)^2)
lm_mspe <- mean((lm_predict - pollutant_test)^2)
# rf importance metrics on full data
message("Fitting RF on full data...")
tm <- system.time({
  rf_model_full <- rfsrc(Concentration ~ .,
                         data = data_sub,
                         mtry = mtry_opt,
```

```
                            nodesize = nodesize_opt,
                            block.size = 1,
                            ntree = ntree,
                            importance = "permute")
  })
  message("Time elapsed: ", round(tm[3], 1), " seconds")
  # importance metrics
  message("Calculating importance metrics...")
  tm <- system.time({
    rf_importance <- subsample(rf_model_full, performance = TRUE)
  })
  message("Time elapsed: ", round(tm[3], 1), " seconds")
  # rf model predictions on full data, and then with each variable permuted
  message("Calculating original and permuted predictions on the full
data...")
  tm <- system.time({
    rf_predict_full <- predict(rf_model, newdata = data_sub)$predicted
    # same permutation index for each variable
    ind_perm <- sample(nrow(data_sub))
    rf_predict_perm <- sapply(names(data_sub)[2:12], function(nm) {
      newdata <- data_sub
      newdata[[nm]] <- newdata[[nm]][ind_perm]
      as.numeric(predict(rf_model, newdata = newdata)$predicted)
    })
  })
  message("Time elapsed: ", round(tm[3], 1), " seconds")
  list(school = school, year = year, pollutant = pollutant,
       rf_mtry = mtry_opt, rf_nodesize = nodesize_opt,
       rf_mspe = rf_mspe, lm_mspe = lm_mspe,
       rf_predict = rf_predict, lm_predict = lm_predict,
       test_obs = pollutant_test, test_ind = !train_ind,
       rf_predict_full = rf_predict_full,
       rf_predict_perm = rf_predict_perm,
       ind_perm = ind_perm,
       rf_importance = extract.subsample(rf_importance, standardize = TRUE))
}


#' Random forest job function for parallelizing.
#'
#' @param ii Job number.
#' @param data P-value data to process.
#' @param out_path Path to save data.
#' @param save Save to file or return directly.
rf_job <- function(ii, data, out_path, save = TRUE) {
  # set reproducible seed
```

```
  parallel_set_seed(ii)
  # extract variables
  school <- as.character(job_descr$school[ii])
  year <- unlist(job_descr$year[ii])
  pollutant <- as.character(job_descr$pollutant[ii])
  ntree <- job_descr$ntree[ii]
  ntree_tune <- job_descr$ntree_tune[ii]
  # random forest calculations
  rf_data <- tryCatch(
    rf_calc(data,
            ntree = ntree,
            ntree_tune = ntree_tune,
            school = school, year = year, pollutant = pollutant),
    error = function(e) NULL
  )
  # save data
  if(!is.null(rf_data)) {
    if(save) {
      saveRDS(rf_data,
              file = get_filename(path = out_path, "rf", ii))
    } else {
      return(rf_data)
    }
  }
  # return TRUE if p-value computation failed and FALSE otherwise
  is.null(rf_data)
}

#--- load and format the traffic data ------------------------------------
---

traffic_raw <- lapply(1:nrow(file_id), function(i) {
  school <- file_id$school[i]
  year <- file_id$year[i]
  read_csv(file.path(data_path, paste0(school, "_", year, ".csv")),
           show_col_types = FALSE) %>%
    mutate(School = school,
           Year = year) # add column
}) %>% bind_rows()
# replace spaces with underscores in variable names
names(traffic_raw) <- gsub("[ ]+", "_", names(traffic_raw))

# format data for random forest calculations
traffic_rf <- traffic_raw %>%
  pivot_longer(cols = CO:O3,
               names_to = "Pollutant", values_to = "Concentration") %>%
```

```
  select(Pollutant, School, Year, Concentration,
         Pressure:`Buses_Trucks`) %>%
  drop_na()

#--- run calculations on parallel cluster ---------------------------------
---

# Each random forest calculation takes several minutes.
# Since we had a number of them to do it was more effective to do them
# in parallel on multiple CPU cores.
#
# The code below saves the results of each random forest calculation (or job)
into a
# separate file.  After the parallel code is done, all jobs are merged into
# various csv files.

ntree <- 10000 # number of rf trees
ntree_tune <- 1000 # number of rf trees for tuning hyperparameters
out_ext <- "10k" # output extension for saving files

# data frame of job descriptions
job_name <- paste0("random_forest_", out_ext)
job_descr <- expand.grid(school = school_names,
                         pollutant = unique(traffic_rf$Pollutant),
                         year = c("2020", "2021", "both"),
                         ntree = ntree,
                         ntree_tune = ntree_tune,
                         stringsAsFactors = FALSE)
out_path <- file.path("calc", job_name) # output folder for saving
job_id <- 1:nrow(job_descr) # job list

# parallel environment
ncores <- min(length(job_id), # number of parallel cores
              detectCores(logical = FALSE))
RNGkind("L'Ecuyer-CMRG")  # parallel processing seed
cl <- makeCluster(spec = ncores)  # create the parallel cluster
clusterSetRNGStream(cl) # creates a .Random.seed on each core

# load all packages on each cluster
invisible(
  clusterEvalQ(cl, {
    require(tidyr)
    require(dplyr)
    require(readr)
    require(readxl)
    require(parallel)
```

```
    require(randomForestSRC)
  })
)

# copy all R objects required for the calculation onto each core.
# it's often simplest to just copy everything in the workspace.
clusterExport(cl, varlist = ls()[ls() != "cl"])

# actual execution of jobs
# traffic_failed is a T/F vector of whether the calculation failed.
# this is useful for restarting particular jobs without having to repeat the
# entire calculation
# also, the random seed of each job is set to the job_id.
# so even if one of the jobs fail, we can restart it later and still
# ensure completely reproducible results.
system.time({
  traffic_failed <- parLapply(cl, job_id, fun = function(ii) {
    rf_job(ii, data = traffic_rf, out_path = out_path)
  })
})

# shut down the parallel cluster
stopCluster(cl)

#--- merge jobs into various csv files -----------------------------------
---

out_path <- file.path("calc", paste0("random_forest_", out_ext))
n_rf <- 24 # number of files per dataset

# Variable importance metrics

# Create a tibble with columns School, Year, Pollutant, Variable, VIMP,
# VIMP_se, VIMP_{2.5,25,50,75,97.5}%.  We'll use the jacknife variance
# estimator for the se, but the subsampling estimator for the quantiles.

rf_importance <- lapply(1:n_rf, function(ii) {
  out <- readRDS(get_filename(out_path, "rf", ii))
  as_tibble(t(out$rf_importance$ci[,1:11]),
            .name_repair = function(x) paste0("VIMP_", x)) %>%
    mutate(School = out$school,
           Year = out$year,
           Pollutant = out$pollutant,
           Variable = names(out$rf_importance$vmp[1:11]),
           VIMP = as.numeric(out$rf_importance$vmp[1:11]),
           VIMP_se = as.numeric(out$rf_importance$se.jk.Z[1:11]),
```

```
            .before = `VIMP_2.5%`)
}) %>% bind_rows()

write_csv(rf_importance,
          file = get_filename("calc", "rf_importance",
                              "2020-2021", "10k", ext = "csv"))

# MSPE and tuning parameters

# Tibble with columns School, Year, Pollutant, mtry, nodesize,
# MSPE_rf, MSPE_lm.

rf_mspe <- lapply(1:n_rf, function(ii) {
  out <- readRDS(get_filename(out_path, "rf", ii))
  tibble(School = out$school,
         Year = out$year,
         Pollutant = out$pollutant,
         mtry = out$rf_mtry,
         nodesize = out$rf_nodesize,
         MSPE_rf = out$rf_mspe,
         MSPE_lm = out$lm_mspe)
}) %>% bind_rows() %>%
  mutate(ratio = MSPE_rf / MSPE_lm)

write_csv(rf_mspe,
          file = get_filename("calc", "rf_mspe",
                              "2020-2021", "10k", ext = "csv"))


# Out-of-sample predictions and observed data

# Tibble with columns School, Year, Pollutant, rf_predict,
# lm_predict, test_obs.

rf_predict <- lapply(1:n_rf, function(ii) {
  out <- readRDS(get_filename(out_path, "rf", ii))
  tibble(rf_predict = out$rf_predict, test_obs = out$test_obs) %>%
    mutate(School = out$school,
           Year = out$year,
           Pollutant = out$pollutant,
           .before = rf_predict)
}) %>% bind_rows() %>%
  # for some reason rf_predict is not saving properly
  mutate(rf_predict = as.numeric(rf_predict))
```

```
write_csv(rf_predict,
          file = get_filename("calc", "rf_predict",
                              "2020-2021", "10k", ext = "csv"))
```

R code used to calculate p-values:

```
#--- load required packages ---------------------------------------------
---

require(tidyr)
require(dplyr)
require(readr)
require(readxl)
require(parallel)
require(aq2020) # this package can be installed from github.com/mlysy/aq2020

#--- helper functions ---------------------------------------------------
---

# In addition to functions `get_filename()` and `parallel_seed()` defined
above:

#' Calculate the absolute difference in medians statistic.
#'
#' @param value A vector of length `n_obs` of daily median data.
#' @param group A grouping vector of length `n_obs` indicating whether the
observation is pre or post lockdown data.  Any vector for which
`length(unique(group)) == 2` will work.
#' @return The difference in medians statistic.
#' @details When `length(unique(group)) = n_group < 2`, the function returns
`NA`.  This is so the p-value calculation doesn't crash when e.g., there is
no data in one of the groups.  When `n_group > 2` an error is thrown.
med_diff <- function(value, group) {
  n_group <- length(unique(group))
  if(n_group > 2) {
    stop("group must consist of at most 2 unique groups.")
  }
  if(n_group < 2) return(NA)
  meds <- tapply(value, group, median)
  dmeds <- abs(meds[1] - meds[2])
  setNames(dmeds, nm = "med_diff")
}

#' Helper function for `summarize()` with tibbles.
set_tibble <- function(x, nm) {
  if(missing(nm)) nm <- names(x)
  tibble(Value = setNames(x, NULL),
         Stat = nm)
}

#' Calculate p-values for a given job.
#'
#' @param data Tibble where row is date and columns are meteorological
```

```r
variables.
#' @param school School in `school_names`.
#' @param year Pair of years.
#' @param variable Name of meteorological variable.
#' @param nsim Number of Monte Carlo simulations for p-value calculation.
#' @return A tibble with columns
#' - `School`
#' - `Variable`
#' - `N_year1`, `N_year2`: Sample size per year.
#' - `Median_year1`, `Median_year2`: Sample median per year.
#' - `Pval`: P-value calculation.
pval_calc <- function(data, school, variable, year, nsim) {
  out_names <- expand.grid(year, c("N", "Median"))
  out_names <- sapply(out_names, as.character)[,2:1]
  out_names <- apply(out_names, 1, paste0, collapse = "_")
  out_names <- c(out_names, "Pval")
  data %>%
    filter(School == school, Variable == variable) %>%
    summarize(
      School = school,
      Variable = variable,
      set_tibble(c(
        N_1 = sum(Year == year[1]),
        N_2 = sum(Year == year[2]),
        Median_1 = median(Value[Year == year[1]]),
        Median_2 = median(Value[Year == year[2]]),
        Pval = fisher_pv(
          group = Year,
          value = Value,
          Tfun = med_diff,
          nsim = nsim, strict = FALSE)[2]
      ), nm = out_names)
    ) %>%
    pivot_wider(names_from = Stat, values_from = Value)
}

#' P-value job function for parallelizing.
#'
#' @param ii Job number.
#' @param data P-value data to process.
#' @param out_path Path to save data.
#' @param save Save to file or return directly.
pval_job <- function(ii, data, out_path, save = TRUE) {
  # set reproducible seed
  parallel_set_seed(ii)
  # extract variables
  school <- as.character(job_descr$school[ii])
  variable <- as.character(job_descr$variable[ii])
  year <- unlist(job_descr$year[ii])
  nsim <- job_descr$nsim[ii]
```

```
  # calculate p-value in tryCatch block to catch/flag any errors
  pv_data <- tryCatch(
    pval_calc(data,
              nsim = nsim,
              school = school, variable = variable, year = year),
    error = function(e) NULL
  )
  # save data
  if(!is.null(pv_data)) {
    if(save) {
      saveRDS(pv_data,
              file = get_filename(path = out_path, "pval", ii))
    } else {
      return(pv_data)
    }
  }
  # return TRUE if p-value computation failed and FALSE otherwise
  is.null(pv_data)
}

#--- load and format the traffic data -------------------------------------
---

traffic_raw <- lapply(1:nrow(file_id), function(i) {
  school <- file_id$school[i]
  year <- file_id$year[i]
  read_csv(file.path(data_path, paste0(school, "_", year, ".csv")),
           show_col_types = FALSE) %>%
    mutate(School = school,
           Year = year) # add column
}) %>% bind_rows()
# replace spaces with underscores in variable names
names(traffic_raw) <- gsub("[ ]+", "_", names(traffic_raw))

# format data for p-value calculations
traffic_pval <- traffic_raw %>%
  select(School, Year, `CO`:`Buses Trucks`) %>%
  pivot_longer(`CO`:`Buses Trucks`,
               names_to = "Variable", values_to = "Value") %>%
  drop_na()


#--- run calculations on parallel cluster ---------------------------------
---

# Each p-value calculation takes several minutes.
# Since we had a number of them to do it was more effective to do them
# in parallel on multiple CPU cores.
#
```

```r
# The code below saves the results of each p-value calculation (or job) into
a
# separate file.  After the parallel code is done, all jobs are merged into
# a single dataset.

pval_nsim <- 200000 # number of p-value simulations
out_ext <- "200k" # output extension for saving files

job_name <- paste0("pvalues_", out_ext)
job_descr <- expand.grid(school = school_names,
                         variable = unique(traffic_pval$Variable),
                         year = list(2020:2021),
                         nsim = pval_nsim)
out_path <- file.path("calc", job_name) # output folder
job_id <- 1:nrow(job_descr) # job list

# parallel environment
ncores <- min(length(job_id), # number of parallel cores
              detectCores(logical = FALSE))
RNGkind("L'Ecuyer-CMRG")  # parallel processing seed
cl <- makeCluster(spec = ncores)  # create the parallel cluster
clusterSetRNGStream(cl) # creates a .Random.seed on each core

# load all packages on each cluster
invisible(
  clusterEvalQ(cl, {
    require(tidyr)
    require(dplyr)
    require(readr)
    require(readxl)
    require(parallel)
    require(aq2020)
  })
)

# copy all R objects required for the calculation onto each core.
# it's often simplest to just copy everything in the workspace.
clusterExport(cl, varlist = ls()[ls() != "cl"])

# actual execution of jobs
# traffic_failed is a T/F vector of whether the calculation failed.
# this is useful for restarting particular jobs without having to repeat the
# entire calculation
# also, the random seed of each job is set to the job_id.
# so even if one of the jobs fail, we can restart it later and still
# ensure completely reproducible results.
system.time({
  traffic_failed <- parLapply(cl, job_id, fun = function(ii) {
    pval_job(ii, data = traffic_pval, out_path = out_path)
```

```
  })
})

# shut down the parallel cluster
stopCluster(cl)

#--- merge jobs into a single dataset ---------------------------------------
---

out_path <- file.path("calc", paste0("pvalues_", out_ext))
n_pval <- 30 # number of p-value files to merge


pval_summary <- lapply(1:n_pval, function(ii) {
  readRDS(get_filename(out_path, "pval", ii))
}) %>% bind_rows()

write_csv(pval_summary,
          file = get_filename("calc", "pval_summary",
                              "2020-2021", "200k", ext = "csv"))
```

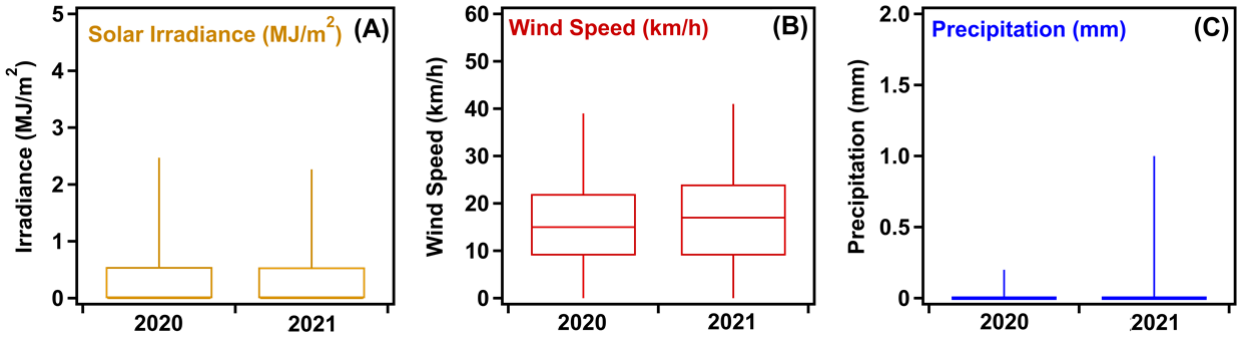


Figure S2. Box and whisker plot comparisons of Meteorology collected from the Canadian National Climate Archives website (Solar Irradiance and Precipitation), where Solar irradiance data was collected from the Ottawa station. Wind speed data was collected from the regional airport located 9 kilometers from the multisensor network. Whiskers extend from the 2nd to 98th percentile.

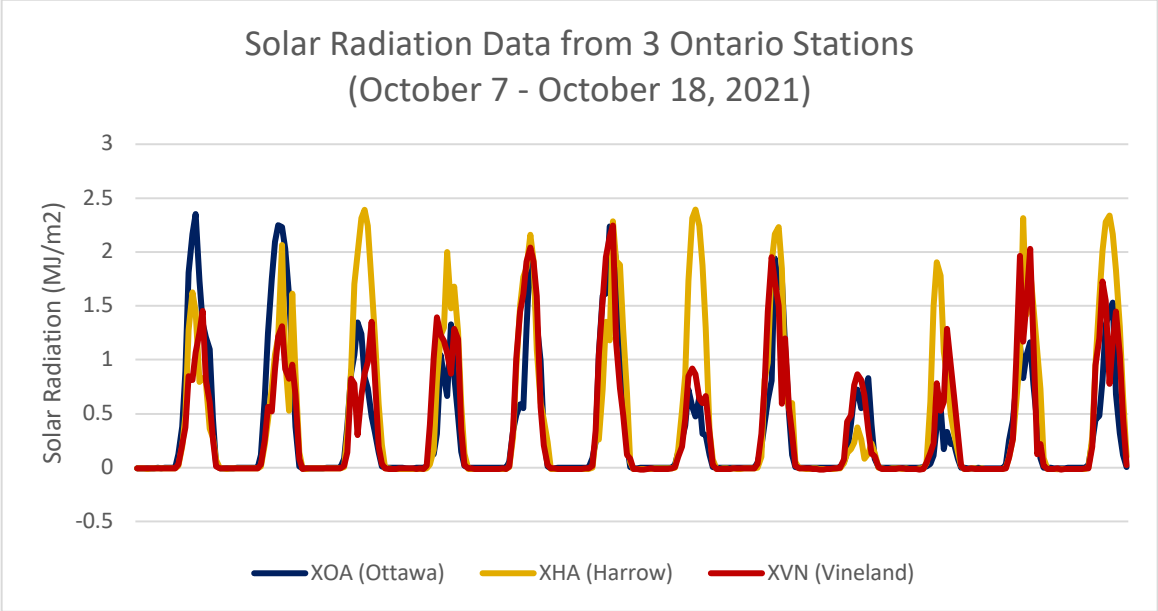


Figure S3. Comparing Solar Radiation from 3 locations in Ontario during October 7 – October 18, 2021.

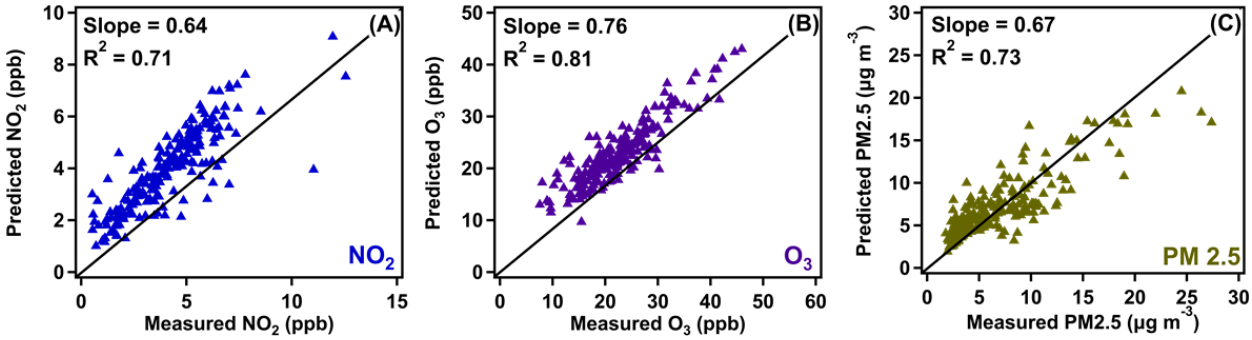


Figure S4. Hourly averaged regression plots of pollutant data comparing the observed dataset with the predicted dataset output by the RF model for both the Fall 2020 and Fall 2021 period for (A) NO_2, (B) O_3, and (C) PM2.5. Data collected from the Pod 2 location. The diagonal is the 1:1 line. The slope and R^2 values are calculated from the regression of predicted onto observed.

Table S2. Out-of-sample mean square error (MSE) for RF model and multiple linear regression model for Fall 2020, Fall 2021, and the combined dataset.

	Pod 1					Pod 2				
	Mtry	**Node Size**	**MSPE_rf**	**MSPE_lm**	**Ratio**	**Mtry**	**Node Size**	**MSPE_rf**	**MSPE_lm**	**Ratio**
Fall 2020										
CO	10	1	892.6	1601.9	0.56	10	1	986.9	1636.2	0.60
NO_2	11	1	2.3	7.6	0.31	11	1	1.6	2.9	0.54
O_3	10	1	12.8	19.8	0.64	11	1	11.3	16.7	0.67
PM2.5	11	1	25.6	40.2	0.64	11	1	7.6	15.7	0.48
Fall 2021										
CO	11	1	380.6	480.7	0.79	10	1	144.3	153.4	0.94
NO_2	11	1	1.7	1.8	0.93	10	1	0.5	0.8	0.67
O_3	11	1	10.1	11.0	0.92	11	1	7.5	13.7	0.55
PM2.5	11	1	4.9	7.3	0.67	11	1	3.4	5.2	0.66
Fall 2020 and 2021: Combined Dataset										
CO	10	1	812.6	1458.4	0.56	11	1	2317.5	2790.8	0.83
NO_2	11	1	2.5	4.4	0.56	11	1	1.2	3.3	0.36
O_3	11	1	10.8	20.1	0.54	11	1	9.8	17.2	0.57
PM2.5	11	1	15.7	30.3	0.52	11	1	5.9	16.2	0.36

Notes: 'Mtry' refers to the number of input features a decision tree has available to consider at each split. 'Node Size' is the minimum size of the terminal node in each tree. 'MSPE_rf' is the Mean Squared Prediction Error (MSPE) for the Random Forest model. 'MSPE_lm' is the MPSE for the multiple linear regression model. 'Ratio' is MSPE_rf / MSPE_lm.

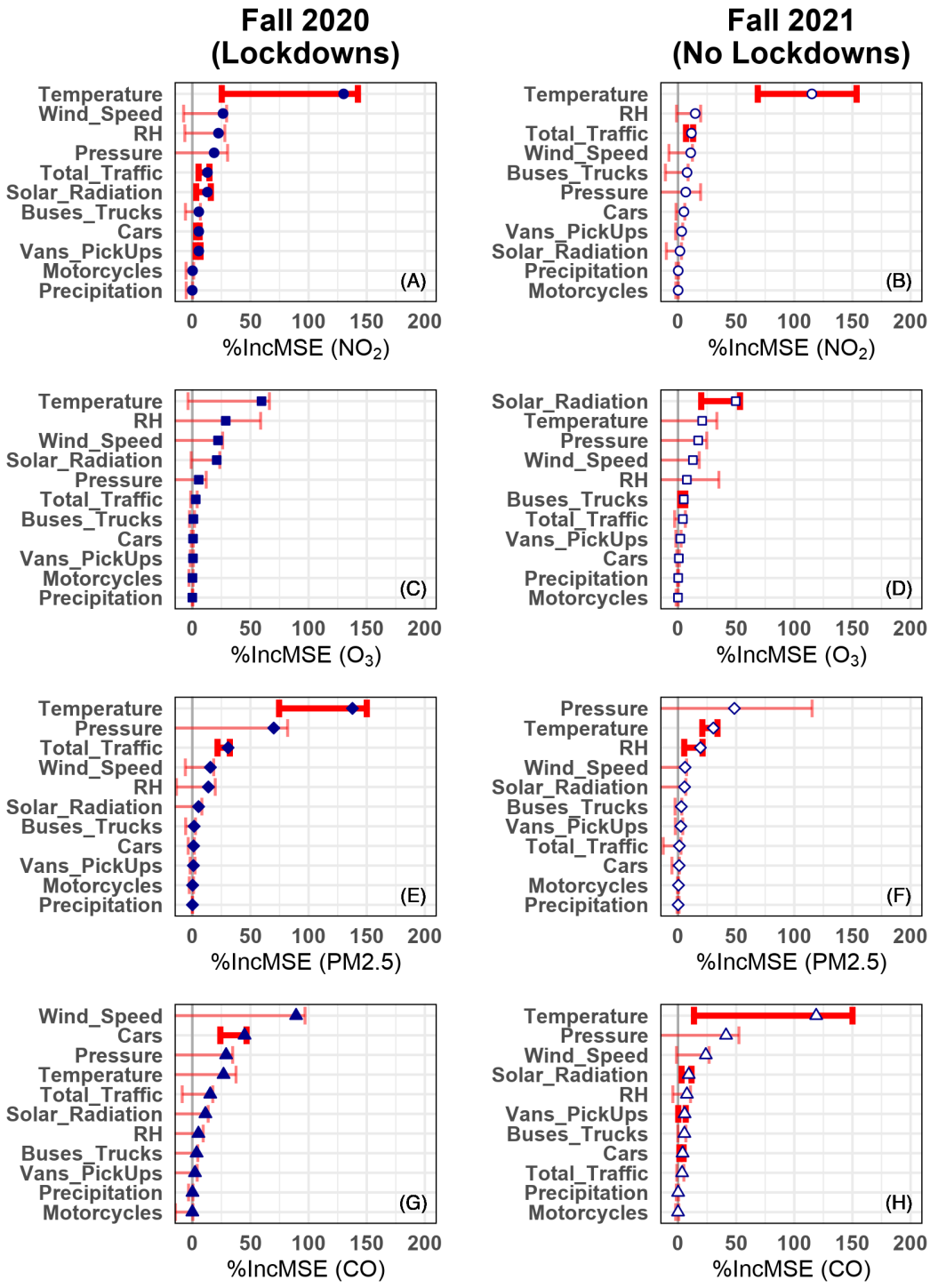


Figure S5. Variable importance plots comparing the calculated *%IncMSE* at the Pod 2 location for NO_2 (A, B), O_3 (C, D), PM2.5 (E, F), and CO (G, H). Symbols represent the estimate values and error bars, the 95% confidence intervals. Statistically significant and non-significant *%IncMSE* values have thick/dark and thin/light error bars, respectively.

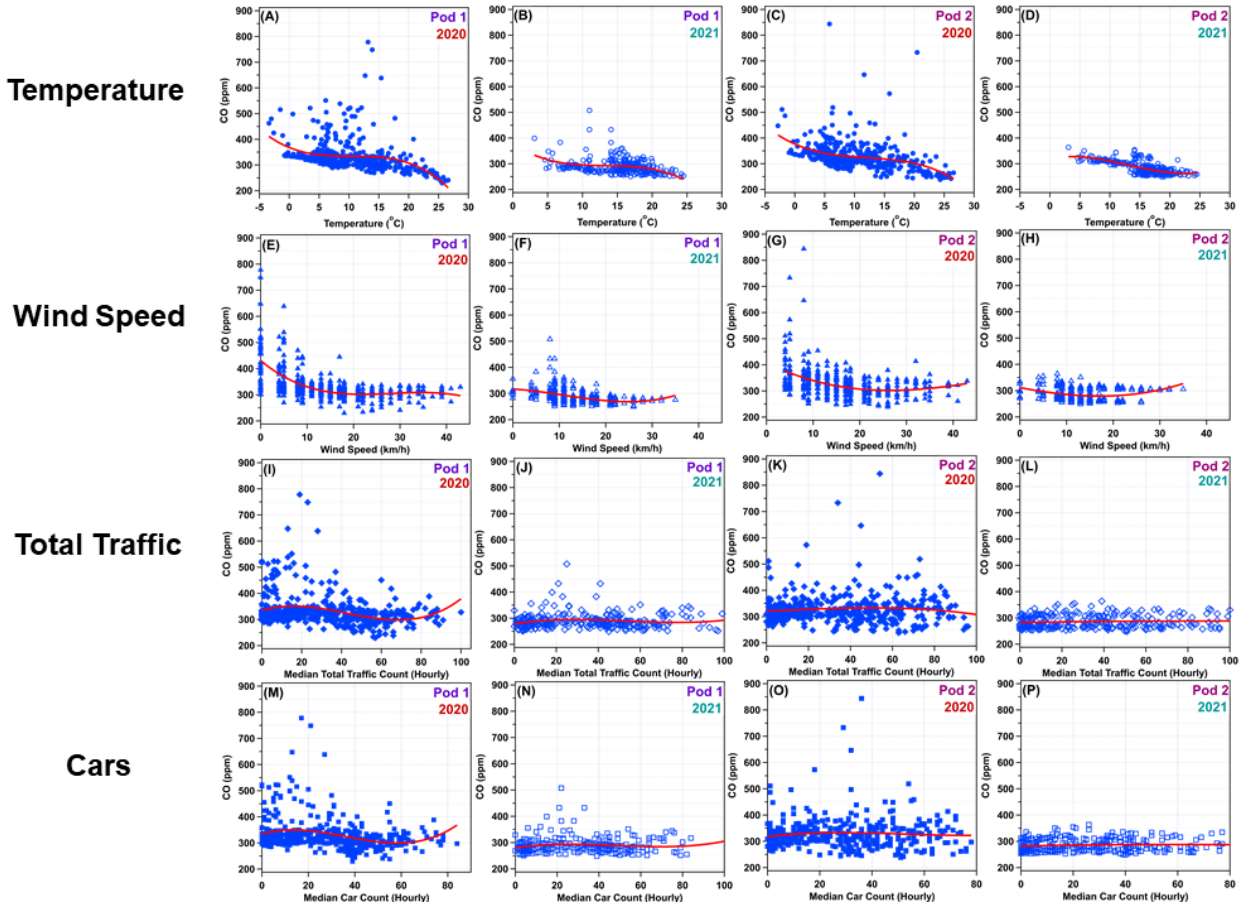


Figure S6. CO emission levels vs various meteorological and traffic variables for both Fall 2020 and Fall 2021. The solid red line corresponds to the LOESS curve of the RF model predictions.