HW3 - Joe Risi - PLSC 597: Machine Learning

Introduction and Replication - Question 1

- Harvard Dataverse Link - https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/IOUQ5X
- Cambridge Core Stable Link - https://www-cambridge-org.ezaccess.libraries.psu.edu/core/journals/japanese-journal-of-political-science/article/explaining-variations-in-responsiveness-to-external-pressure-japans-aid-policy-and-bureaucratic-politics/EF0A1EAF5E54EB18A332C444CBE3A592
- Github Link to Code For Assignment - https://github.com/jrisi256/schoolwork/tree/main/plsc597/hw3/src

Introduction: Explaining variations in responsiveness to external pressure: Japan's aid policy and bureaucratic politics

- This paper attempts to provide an explanation as to how and in what direction the USA influence's Japan's aid policies and if the impact of USA influence varies across different types of aid.
- Results suggest the USA tends to urge Japan to complement USA aid efforts rather than to be a substitute for USA aid efforts as substitution would increase Japan's clout, reputation and influence potentially at the cost of the USA.
- Grants appear to be more receptive to USA pressure vs. loans potentially because loans are distributed in consultation with multiple parties each of whom only feels minimal USA pressure if any at all.

Data and Methods

- Data for the dependent variables comes from the Ministry of Foreign Affair's website. Each observation is a record of aid flow from Japan to another country in a given year from the years 1971 - 2009.
- Data for the independent variables come from a variety of sources including but not limited to the United Nations Statistics Division, the USA Agency for International Development, and the Global Terrorism Database.
- **Dependent Variable**: **lyenloans2**: The dependent variable of interest is the net disbursement of loans by the Japanese government in a given year (in 2015 US dollars). The natural logarithm is applied to this variable and then 1 is added because the data is highly right skewed.
- **Independent Variables**:
  - **lusaid2l**: Sum of USA economic and military assistance to the given country in the given year. The natural logarithm is applied to this variable and then 1 is added. In the OLS regression, the variable is lagged by one year.
- **Control Variables**:
  - **lgdppcl**: Natural logarithm of per capita GDP.
  - **lpopl**: Natural logarithm of population.
  - **ltradel**: Natural logarithm (plus 1) of the sum of exports and imports between Japan and a country.
  - **democracy2**: Democracy indicator variable (1 if country is a democracy, 0 otherwise).
  - **jipdistance**: Policy distance between the two countries' voting patterns (measured as the distance between ideal point estimates of Japan and the recipient country in a given year).
  - **warl**: Is the recipient country at war? (1 if yes, 0 if otherwise).
  - **ltotaldeath2l**: Natural logarithm (plus 1) of the total amount of deaths due to natural disasters.
  - **ltargetjapanl**: Natural logarithm (plus 1) of the number of terrorist attacks targeting Japanese citizens in a given country in a given year.

  - **cmember**: Is the given country in a given year a member of the United Nations Security Council? 1 if they are temporary members, 0 otherwise. Permanent members are treated as missing.
  - Country and year are included as fixed-effect dummy variables.
- **Results**
  - **US aid** and **amount of trade** are positively and highly statistically significantly associated with Japanese net disbursement of loans. USA aid being positively associated with Japanese loan disbursement confirms the hypothesis of the paper which is that the USA pressures/influences Japan to align its aid with the USA. The trade variable finding seems logical as Japan has a small domestic market and lack of natural resources which would necessitate Japan exercising all options at their disposal to expand their import/export markets. Previous research has demonstrated mixed evidence regarding the effect of this variable though on Japanese aid.
  - **Population** and **war** are negatively and highly statistically significantly associated with Japanese net disbursement of loans. Previous research has demonstrated population has a negative relationship to Japanese aid due to the fact that smaller countries are easier to *buy off* as it were during UN voting. With each country receiving one vote, it's more efficient for Japan to aid many smaller countries than a few relatively bigger ones. The war variable finding make sense given Japan's emphasis on peace and pacifism post-WWII, the decreased likelihood of a warring state to repay its debts, and the increased difficulty and costs of assuring the safety of the personnel designated as dispersing the loan.
  - All other variables are not statistically significant.

Packages

```r
library(mlr)
library(here)
library(purrr)
library(dplyr)
library(caret)
library(readr)
library(tidyr)
library(stringr)
```

Light data cleaning and Replication of Results

I turn year and country code into categorical variables to match results obtained in the paper, and I filter out all observations missing data. The coefficient results replicate. The standard errors are different due to the paper clustering standard errors by country. When I don't cluster by standard error, the substantive findings remain largely the same (one variable gains significance at the 0.05 level which we ignore because we aren't clustering).

```r
# Read in data
load(here("hw3/data/aid_data.RData"))

# Turn year and country code into categorical variables
dataClean <-
    table %>%
    select(lyenloans2, lusaid2l, lgdppcl, lpopl, ltradel, democracy2, jipdistance,
           warl, ltotaldeath2l, ltargetjapanl, cmember, year, ccode) %>%
    filter(across(everything(), ~!is.na(.x))) %>%
    mutate(year = as.factor(year),
           ccode = as.factor(ccode),
           democracy2 = as.integer(democracy2),
           warl = as.integer(warl),
           cmember = as.integer(cmember))
```

```r
# Replicate results from paper
olsPaper <- lm(lyenloans2 ~ ., data = dataClean)
summary(olsPaper)
```

```
##
## Call:
## lm(formula = lyenloans2 ~ ., data = dataClean)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.3785  -2.8165  -0.4248   2.1704  17.0750
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    39.19154   11.52168   3.402 0.000675 ***
## lusaid2l        0.11899    0.01989   5.981 2.36e-09 ***
## lgdppcl         0.57342    0.35559   1.613 0.106893
## lpopl          -3.12025    0.60373  -5.168 2.45e-07 ***
## ltradel         0.17771    0.05769   3.080 0.002079 **
## democracy2      0.22253    0.29541   0.753 0.451310
## jipdistance    -0.43795    0.21609  -2.027 0.042743 *
## warl           -1.55114    0.28664  -5.412 6.53e-08 ***
## ltotaldeath2l  -0.06408    0.04418  -1.450 0.147031
## ltargetjapanl  -0.99810    0.88855  -1.123 0.261366
## cmember         0.32628    0.30295   1.077 0.281529
## year1972        0.17805    0.75317   0.236 0.813127
## year1973        0.75007    0.74987   1.000 0.317230
## year1974        2.48718    0.74381   3.344 0.000832 ***
## year1975        3.00275    0.74090   4.053 5.13e-05 ***
## year1976        3.04577    0.73830   4.125 3.76e-05 ***
## year1977        3.30526    0.74100   4.461 8.35e-06 ***
## year1978        3.97424    0.74232   5.354 8.98e-08 ***
## year1979        4.93654    0.74659   6.612 4.17e-11 ***
## year1980        4.67354    0.75022   6.230 5.04e-10 ***
## year1981        4.60143    0.75095   6.127 9.58e-10 ***
## year1982        4.23368    0.71656   5.908 3.67e-09 ***
## year1983        4.66055    0.71864   6.485 9.67e-11 ***
## year1984        4.76738    0.72079   6.614 4.11e-11 ***
## year1985        4.29495    0.72567   5.919 3.45e-09 ***
## year1986        4.78052    0.72792   6.567 5.62e-11 ***
## year1987        5.04376    0.73254   6.885 6.44e-12 ***
## year1988        4.84764    0.73711   6.577 5.28e-11 ***
## year1989        5.90685    0.74190   7.962 2.06e-15 ***
## year1990        5.13950    0.75056   6.848 8.37e-12 ***
## year1991        5.12503    0.75883   6.754 1.60e-11 ***
## year1992        4.35645    0.75674   5.757 9.06e-09 ***
## year1993        3.80048    0.75473   5.036 4.92e-07 ***
## year1994        3.88776    0.75980   5.117 3.22e-07 ***
## year1995        4.40164    0.76652   5.742 9.86e-09 ***
## year1996        4.51211    0.76711   5.882 4.30e-09 ***
## year1997        4.61604    0.77209   5.979 2.40e-09 ***
## year1998        4.14716    0.77897   5.324 1.06e-07 ***
## year1999        4.35273    0.78599   5.538 3.21e-08 ***
```

```
## year2000        4.50175    0.79509   5.662 1.58e-08 ***
## year2001        4.11382    0.79588   5.169 2.44e-07 ***
## year2002        3.33538    0.80530   4.142 3.50e-05 ***
## year2003        3.12360    0.80638   3.874 0.000109 ***
## year2004        2.74408    0.81320   3.374 0.000745 ***
## year2005        2.94750    0.82119   3.589 0.000335 ***
## year2006        2.83486    0.83229   3.406 0.000664 ***
## year2007        2.71575    0.84119   3.228 0.001252 **
## year2008        2.50461    0.85327   2.935 0.003346 **
## year2009        3.03550    0.86388   3.514 0.000445 ***
## ccode31       -14.38290    3.14376  -4.575 4.87e-06 ***
## ccode40        -0.80409    1.83021  -0.439 0.660433
## ccode41        -2.00744    2.15033  -0.934 0.350579
## ccode42         4.46769    1.84906   2.416 0.015717 *
## ccode51        -1.22024    2.18646  -0.558 0.576806
## ccode52        -8.75228    2.42691  -3.606 0.000313 ***
## ccode53       -13.82402    3.15983  -4.375 1.24e-05 ***
## ccode54       -16.71453    4.12049  -4.056 5.05e-05 ***
## ccode55       -15.68705    3.89459  -4.028 5.71e-05 ***
## ccode56       -15.08136    3.70952  -4.066 4.86e-05 ***
## ccode57       -15.50175    3.91142  -3.963 7.49e-05 ***
## ccode60       -18.93473    4.32518  -4.378 1.22e-05 ***
## ccode70        12.29968    1.43851   8.550  < 2e-16 ***
## ccode80       -13.96865    3.56935  -3.913 9.21e-05 ***
## ccode90         4.07261    1.77278   2.297 0.021641 *
## ccode91         3.55503    2.10448   1.689 0.091227 .
## ccode92         3.67605    1.94373   1.891 0.058647 .
## ccode93        -0.72765    2.16865  -0.336 0.737239
## ccode94        -1.14131    2.07871  -0.549 0.582997
## ccode95        -5.05058    2.20163  -2.294 0.021829 *
## ccode100        7.73110    1.47675   5.235 1.71e-07 ***
## ccode101        1.47836    1.40578   1.052 0.293016
## ccode110       -8.25488    2.95412  -2.794 0.005219 **
## ccode115      -11.01582    3.09998  -3.554 0.000383 ***
## ccode130        6.85569    1.70201   4.028 5.71e-05 ***
## ccode135       15.16985    1.54956   9.790  < 2e-16 ***
## ccode140       19.17562    1.60325  11.960  < 2e-16 ***
## ccode145        6.89774    2.03037   3.397 0.000686 ***
## ccode150        7.79007    2.14824   3.626 0.000290 ***
## ccode155        4.60597    1.51729   3.036 0.002412 **
## ccode160        8.11548    1.34516   6.033 1.72e-09 ***
## ccode165       -2.69498    2.02443  -1.331 0.183172
## ccode205       -6.26997    1.71926  -3.647 0.000268 ***
## ccode210       -1.51117    1.26788  -1.192 0.233360
## ccode211       -2.93525    1.35548  -2.165 0.030396 *
## ccode212      -11.79114    3.12261  -3.776 0.000161 ***
## ccode225       -3.57158    2.39273  -1.493 0.135581
## ccode230        1.58236    1.25331   1.263 0.206809
## ccode235       -3.80287    1.42807  -2.663 0.007770 **
## ccode255        3.16365    1.60167   1.975 0.048295 *
## ccode260        3.27412    1.72159   1.902 0.057252 .
## ccode290        3.14590    1.36362   2.307 0.021092 *
## ccode305       -4.08063    1.42558  -2.862 0.004221 **
## ccode310        0.39625    1.51830   0.261 0.794114
```

```
## ccode315         0.05726    1.99184   0.029 0.977069
## ccode316        -2.65628    1.82504  -1.455 0.145600
## ccode317         2.58211    2.07028   1.247 0.212370
## ccode325         2.29672    1.29521   1.773 0.076246 .
## ccode338       -12.59744    3.01449  -4.179 2.98e-05 ***
## ccode339         0.64829    2.31163   0.280 0.779145
## ccode341       -10.08467    4.26114  -2.367 0.017986 *
## ccode343        -2.20251    2.60702  -0.845 0.398241
## ccode344        -5.51596    2.12284  -2.598 0.009392 **
## ccode345         0.96676    1.53643   0.629 0.529227
## ccode346         1.60515    2.49873   0.642 0.520650
## ccode349        -8.32641    2.39250  -3.480 0.000505 ***
## ccode350        -4.04630    1.41079  -2.868 0.004146 **
## ccode352       -11.74631    2.68547  -4.374 1.24e-05 ***
## ccode355         5.46632    1.82544   2.995 0.002761 **
## ccode359        -3.74288    2.55749  -1.463 0.143391
## ccode360         5.31641    1.45204   3.661 0.000253 ***
## ccode366        -9.03424    2.60826  -3.464 0.000537 ***
## ccode367        -6.84559    2.42107  -2.828 0.004709 **
## ccode368        -5.98810    2.24403  -2.668 0.007643 **
## ccode369         6.04702    1.83939   3.288 0.001017 **
## ccode370        -1.69674    2.06750  -0.821 0.411870
## ccode371         1.58672    2.59229   0.612 0.540505
## ccode372         4.15305    2.45733   1.690 0.091075 .
## ccode373         7.31106    2.26168   3.233 0.001234 **
## ccode375        -5.34801    1.58718  -3.369 0.000758 ***
## ccode380        -3.09821    1.39354  -2.223 0.026240 *
## ccode385        -5.48224    1.62362  -3.377 0.000739 ***
## ccode390        -4.68238    1.55582  -3.010 0.002628 **
## ccode395       -14.17632    3.03254  -4.675 3.02e-06 ***
## ccode402       -10.53371    3.37300  -3.123 0.001800 **
## ccode403       -13.56342    4.12076  -3.291 0.001003 **
## ccode404        -6.82793    3.05886  -2.232 0.025645 *
## ccode411       -10.53390    3.24567  -3.246 0.001180 **
## ccode420        -7.91651    3.12165  -2.536 0.011241 *
## ccode432         2.04830    2.23219   0.918 0.358860
## ccode433         4.02833    2.08810   1.929 0.053762 .
## ccode434        -1.86676    2.30992  -0.808 0.419042
## ccode435         0.22835    2.66850   0.086 0.931810
## ccode436         2.70939    2.27499   1.191 0.233729
## ccode437         4.92476    1.84039   2.676 0.007475 **
## ccode438         3.93801    2.44750   1.609 0.107678
## ccode439        -0.48868    2.24953  -0.217 0.828031
## ccode450        -0.85365    2.79413  -0.306 0.759987
## ccode451        -1.18893    2.48715  -0.478 0.632649
## ccode452         8.07594    1.90477   4.240 2.28e-05 ***
## ccode461        -1.04932    2.45645  -0.427 0.669274
## ccode471         3.83251    1.87592   2.043 0.041102 *
## ccode475        13.48878    1.70849   7.895 3.50e-15 ***
## ccode481        -8.36504    2.56330  -3.263 0.001108 **
## ccode482        -2.49956    2.57765  -0.970 0.332238
## ccode483        -0.23253    2.28010  -0.102 0.918776
## ccode484        -5.25383    2.38998  -2.198 0.027973 *
## ccode490         8.77984    1.87855   4.674 3.03e-06 ***
```

```
## ccode500         6.84246    2.09774   3.262 0.001114 **
## ccode501        16.00424    1.84920   8.655  < 2e-16 ***
## ccode510        10.12846    1.96973   5.142 2.82e-07 ***
## ccode516         1.61512    2.53989   0.636 0.524868
## ccode517         2.90902    2.40942   1.207 0.227350
## ccode520         3.30227    2.36101   1.399 0.161971
## ccode522        -9.47955    3.24184  -2.924 0.003469 **
## ccode530         8.92269    2.11893   4.211 2.59e-05 ***
## ccode540         1.42813    1.86471   0.766 0.443786
## ccode541         6.15122    2.25047   2.733 0.006291 **
## ccode551         7.49242    2.08615   3.592 0.000332 ***
## ccode552         8.34497    2.13961   3.900 9.73e-05 ***
## ccode553         4.45973    2.25236   1.980 0.047753 *
## ccode560         4.99602    1.60912   3.105 0.001914 **
## ccode580         9.11834    2.13894   4.263 2.05e-05 ***
## ccode581        -9.44898    3.41202  -2.769 0.005637 **
## ccode590        -3.80337    2.62300  -1.450 0.147116
## ccode591       -17.25187    4.02993  -4.281 1.89e-05 ***
## ccode600        13.57270    1.63872   8.282  < 2e-16 ***
## ccode615         6.58531    1.54185   4.271 1.98e-05 ***
## ccode616         8.73518    1.85841   4.700 2.66e-06 ***
## ccode620        -3.49290    1.90584  -1.833 0.066898 .
## ccode625         5.54809    1.84062   3.014 0.002588 **
## ccode630        10.56655    1.61460   6.544 6.54e-11 ***
## ccode640        15.01188    1.38725  10.821  < 2e-16 ***
## ccode645         6.99830    1.93962   3.608 0.000311 ***
## ccode651        16.39815    1.73883   9.431  < 2e-16 ***
## ccode652        10.44896    1.86516   5.602 2.22e-08 ***
## ccode660        -2.27659    2.14371  -1.062 0.288291
## ccode663         4.53227    2.21906   2.042 0.041159 *
## ccode666        -5.15399    1.74622  -2.952 0.003176 **
## ccode670         3.68965    1.37631   2.681 0.007367 **
## ccode678        -1.83385    2.32592  -0.788 0.430474
## ccode679         5.06025    2.12786   2.378 0.017438 *
## ccode680        -2.34640    2.84048  -0.826 0.408808
## ccode690        -7.82211    2.17611  -3.595 0.000328 ***
## ccode692       -11.87071    2.84302  -4.175 3.02e-05 ***
## ccode694       -12.18831    2.86141  -4.260 2.08e-05 ***
## ccode696        -5.90827    2.09526  -2.820 0.004823 **
## ccode698        -8.34739    2.29863  -3.631 0.000285 ***
## ccode700         3.14891    2.06468   1.525 0.127286
## ccode701         1.34164    2.32377   0.577 0.563723
## ccode702        -0.69137    2.65039  -0.261 0.794213
## ccode703        12.00746    2.66484   4.506 6.75e-06 ***
## ccode704        17.49030    2.26397   7.725 1.33e-14 ***
## ccode705        13.27069    1.92140   6.907 5.55e-12 ***
## ccode712         4.54029    2.61141   1.739 0.082157 .
## ccode731         2.68721    2.18810   1.228 0.219464
## ccode732         1.09832    1.56648   0.701 0.483247
## ccode750        31.18393    2.47351  12.607  < 2e-16 ***
## ccode770        23.72244    1.83499  12.928  < 2e-16 ***
## ccode771        15.80649    1.99375   7.928 2.70e-15 ***
## ccode775        18.81745    2.24190   8.394  < 2e-16 ***
## ccode780        18.57824    1.85450  10.018  < 2e-16 ***
```

```
## ccode781      -11.51307    3.57791  -3.218 0.001300 **
## ccode790       13.51506    2.16361   6.247 4.53e-10 ***
## ccode800       19.06467    1.58946  11.994  < 2e-16 ***
## ccode811        7.55443    2.33373   3.237 0.001215 **
## ccode812        4.25461    2.57107   1.655 0.098024 .
## ccode816       17.50127    1.99254   8.783  < 2e-16 ***
## ccode820       12.60699    1.56497   8.056 9.70e-16 ***
## ccode830       -2.44697    1.90929  -1.282 0.200035
## ccode835        0.41174    3.19816   0.129 0.897566
## ccode840       23.18948    1.68330  13.776  < 2e-16 ***
## ccode850       25.45593    1.88477  13.506  < 2e-16 ***
## ccode900       -1.24469    1.25277  -0.994 0.320485
## ccode910        7.47902    2.30221   3.249 0.001167 **
## ccode920       -5.82301    1.74883  -3.330 0.000875 ***
## ccode935      -12.32894    3.79111  -3.252 0.001153 **
## ccode940       -9.07692    3.46194  -2.622 0.008769 **
## ccode950       -6.74884    2.82448  -2.389 0.016911 *
## ccode955      -15.31988    4.33153  -3.537 0.000408 ***
## ccode990      -13.96751    3.73673  -3.738 0.000188 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.367 on 5257 degrees of freedom
## Multiple R-squared:  0.4916, Adjusted R-squared:  0.4705
## F-statistic: 23.21 on 219 and 5257 DF,  p-value: < 2.2e-16
```

Question 2

Create test and train

democracy2, warl, ltotaldeath2l, ltargetjapanl, cmember, ccode, and year are all dropped due to sparseness/categorical variable issues with GAM.

```r
# Set random seed
set.seed(420)

# Create test and train data
split <- createDataPartition(dataClean$lyenloans2, p = 0.7, list = F, times = 1)

# Drop problematic variables
dataCleanShort <-
    dataClean %>%
    select(-democracy2, -warl, -ltotaldeath2l, -ltargetjapanl, -cmember,
           -ccode, -year)

train <- dataCleanShort[split,]
test <- dataCleanShort[-split,]

# Create our list of features to hold out one-by-one (as well as full features)
features <- as.list(c(colnames(select(dataCleanShort, -lyenloans2)), "total"))
names(features) <- features

# Create new task for each new feature set
tasksTrain <- map(features, function(missingFeat) {
    if(missingFeat != "total")
```

Table 1. Results of OLS regressions

	1 Net ODA	2 Loans	3 Grants-tech	4 Grants	5 Tech assist
Constant	20.768	39.192*	−4.680	−27.639	−14.813
	(25.306)	(22.425)	(24.401)	(32.958)	(23.462)
$\ln(\text{US aid})_{t-1}$	0.194***	0.119***	0.160***	0.168***	0.138***
	(0.034)	(0.041)	(0.031)	(0.035)	(0.029)
$\ln(\text{GDPpc})_{t-1}$	−0.958	0.573	−0.233	−1.694**	0.333
	(0.708)	(0.860)	(0.573)	(0.843)	(0.598)
$\ln(\text{Population})_{t-1}$	−1.121	−3.120***	−0.019	2.290	0.222
	(1.336)	(1.087)	(1.301)	(1.743)	(1.239)
$\ln(\text{Trade})_{t-1}$	0.091	0.178**	0.081	−0.039	0.121*
	(0.066)	(0.078)	(0.062)	(0.108)	(0.069)
Democracy_{t-1}	1.205**	0.223	1.112***	0.912*	1.213***
	(0.485)	(0.555)	(0.330)	(0.517)	(0.324)
$\text{Policy distance}_{t-1}$	−1.567***	−0.438	−1.857***	−0.961**	−1.824***
	(0.390)	(0.458)	(0.364)	(0.402)	(0.359)
War_{t-1}	−1.375***	−1.551***	−0.892***	−1.432***	−0.780***
	(0.388)	(0.680)	(0.227)	(0.465)	(0.222)
$\ln(\text{Natural disasters})_{t-1}$	0.080**	−0.064	0.055**	0.111**	0.042*
	(0.037)	(0.060)	(0.025)	(0.050)	(0.023)
$\ln(\text{Attacks on Japanese})_{t-1}$	0.072	−0.998	0.493	1.724*	0.169
	(0.830)	(1.130)	(0.332)	(0.964)	(0.229)
UNSC member	−0.102	0.326	0.063	0.133	0.080
	(0.262)	(0.348)	(0.169)	(0.208)	(0.164)
Country fixed effects	Yes	Yes	Yes	Yes	Yes
Year fixed effects	Yes	Yes	Yes	Yes	Yes
Observations	5,477	5,477	5,477	5,477	5,477
R^2	0.667	0.492	0.810	0.675	0.812

Clustered standard errors are reported within parentheses. $^*p<0.1$, $^{**}p<0.05$, $^{***}p<0.01$ (two-tailed).

Figure 1: Regression Results from Paper

```r
        makeRegrTask(data = select(train, -matches(missingFeat)),
                     target = "lyenloans2")
    else
        makeRegrTask(data = train, target = "lyenloans2")
})
```

Create learners and set cross-validation strategy

We will be using 3-fold cross-validation, and we will be exploring 60 different combinations of hyperparameters for the random forest algorithm.

```r
# Create our learners
ols <- makeLearner("regr.lm")
gam <- makeLearner("regr.gamboost")
rf <- makeLearner("regr.randomForest")

# 3-fold Cross-Validation
kFold3 <- makeResampleDesc("CV", iters = 3)

# Explore 100 different random hyperparameter combinations for random forest
randSearchRf <- makeTuneControlRandom(maxit = 60)
```

Tune the parameters for random forest and cross-validate GAM

It should be noted root mean squared error is used to determine the best hyperparameters for the random forest algorithm, and it is used to determine the performance of the models trained using GAM.

```r
# Tune the hyperparameters for random forest
rfParamSpace <-
    makeParamSet(makeIntegerParam("ntree", lower = 100, upper = 100),
                 makeIntegerParam("mtry", lower = 1, upper = 4),
                 makeIntegerParam("nodesize", lower = 1, upper = 10),
                 makeIntegerParam("maxnodes", lower = 5, upper = 30))

tunedRfs <- map(tasksTrain, function(task) {
    tuneParams(rf,
               task = task,
               resampling = kFold3,
               par.set = rfParamSpace,
               control = randSearchRf,
               measures = list(rmse))
})

# There are no hyperparameters we need to tune for GAMs
gamCVs <- map(tasksTrain, function(task) {
    resample(gam, task, resampling = kFold3, measures = list(rmse))
})

tunedRfs[["total"]]
```

```
## Tune result:
## Op. pars: ntree=100; mtry=4; nodesize=4; maxnodes=30
## rmse.test.rmse=5.9401483
```

```r
gamCVs[["total"]]
```

```
## Resample Result
## Task: train
## Learner: regr.gamboost
## Aggr perf: rmse.test.rmse=6.3472633
## Runtime: 1.10982
```

Using cross-validation, it appears as if random forest performs better than GAM. However we will hold off on making any final claims until we evaluate the models on the held-out test set.

Question 3

Train our models

```
# For each learning algorithm, for each task, train a model

# train our OLS models
trainedOlss <- map(tasksTrain, function(task) {
    mlr::train(ols, task)
})

# train GAM models
trainedGams <- map(tasksTrain, function(task) {
    mlr::train(gam, task)
})

# train random forest models
tunedRfPars <- map(tunedRfs, function(hyperparams) {
    setHyperPars(rf, par.vals = hyperparams$x)
})

trainedRfs <- pmap(list(tasksTrain, tunedRfPars), function(task, tunedModel){
    mlr::train(tunedModel, task)
})
```

Run predictions

```
# Run predictions on the test set
PredictTest <- function(model, missingFeat) {
    if(missingFeat != "total")
        p <- predict(model, newdata = select(test, -lyenloans2, -missingFeat))
    else
        p <- predict(model, newdata = select(test, -lyenloans2))

    p$data["truth"] <- test[["lyenloans2"]]

    p$data <-
        p$data %>%
        mutate(error = response - truth,
               error_sq = error ^ 2,
               sum_sq_error = sum(error_sq),
               rmse = sqrt(sum_sq_error / n()))
    p
}
```

```r
predictOlss <- pmap(list(trainedOlss, names(trainedOlss)), PredictTest)
predictGams<- pmap(list(trainedGams, names(trainedGams)), PredictTest)
predictRfs <- pmap(list(trainedRfs, names(trainedRfs)), PredictTest)

cat("Root Mean Square Error For OLS: ")
```

```
## Root Mean Square Error For OLS:
```

```r
unique(predictOlss[["total"]]$data$rmse)
```

```
## [1] 6.498816
```

```r
cat("\n")
```

```r
cat("Root Mean Square Error For GAM: ")
```

```
## Root Mean Square Error For GAM:
```

```r
unique(predictGams[["total"]]$data$rmse)
```

```
## [1] 6.212928
```

```r
cat("\n")
```

```r
cat("Root Mean Square Error For Random Forest: ")
```

```
## Root Mean Square Error For Random Forest:
```

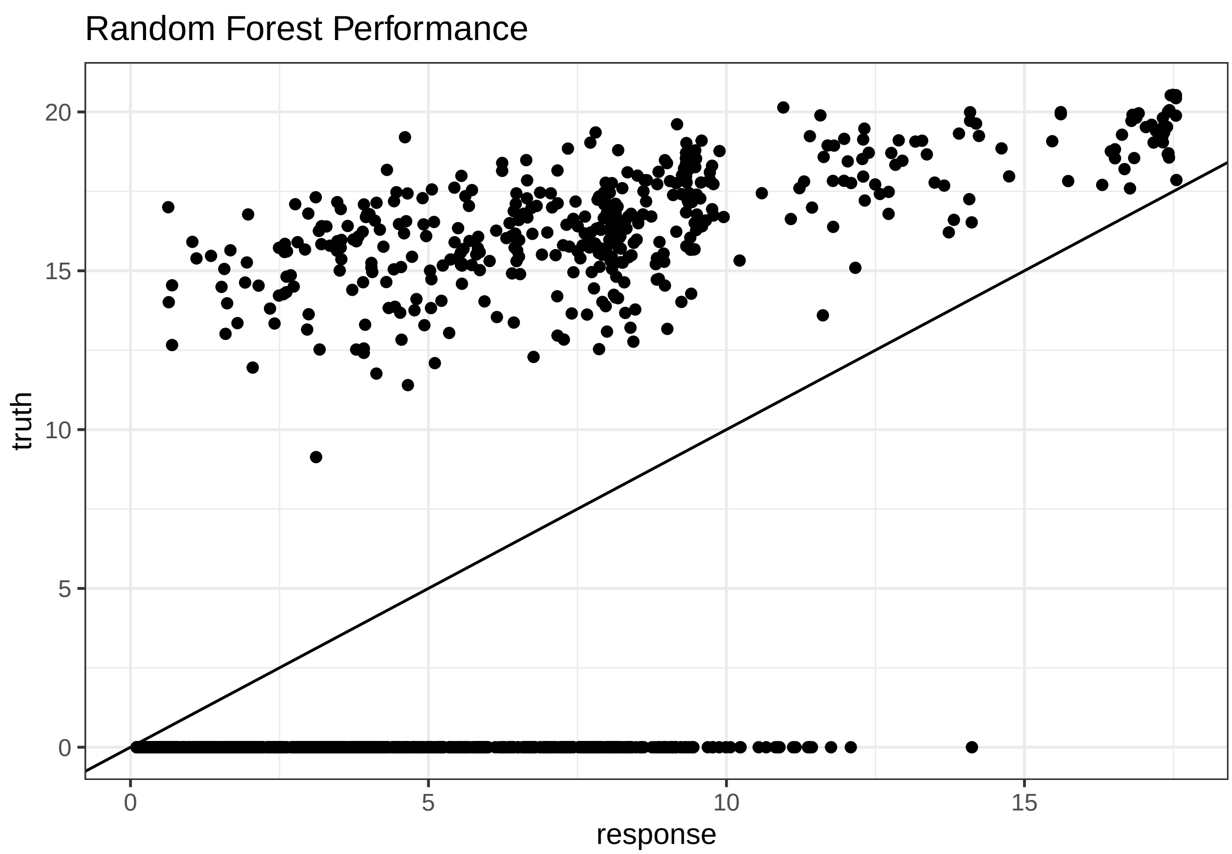
```r
unique(predictRfs[["total"]]$data$rmse)
```

```
## [1] 5.849523
```

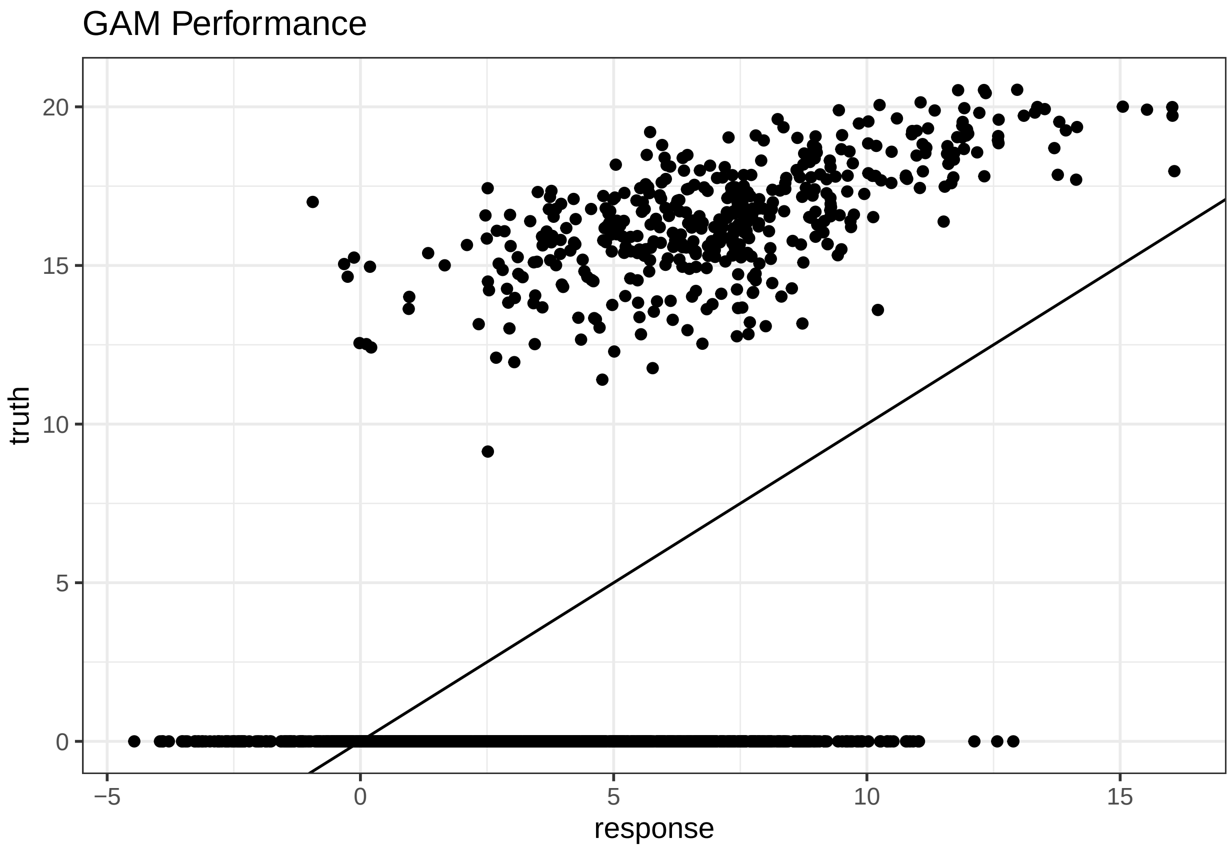
```r
cat("\n")
```

It would appear as if the results from the cross-validation hold up and Random Forest performs the best on the held-out test data.

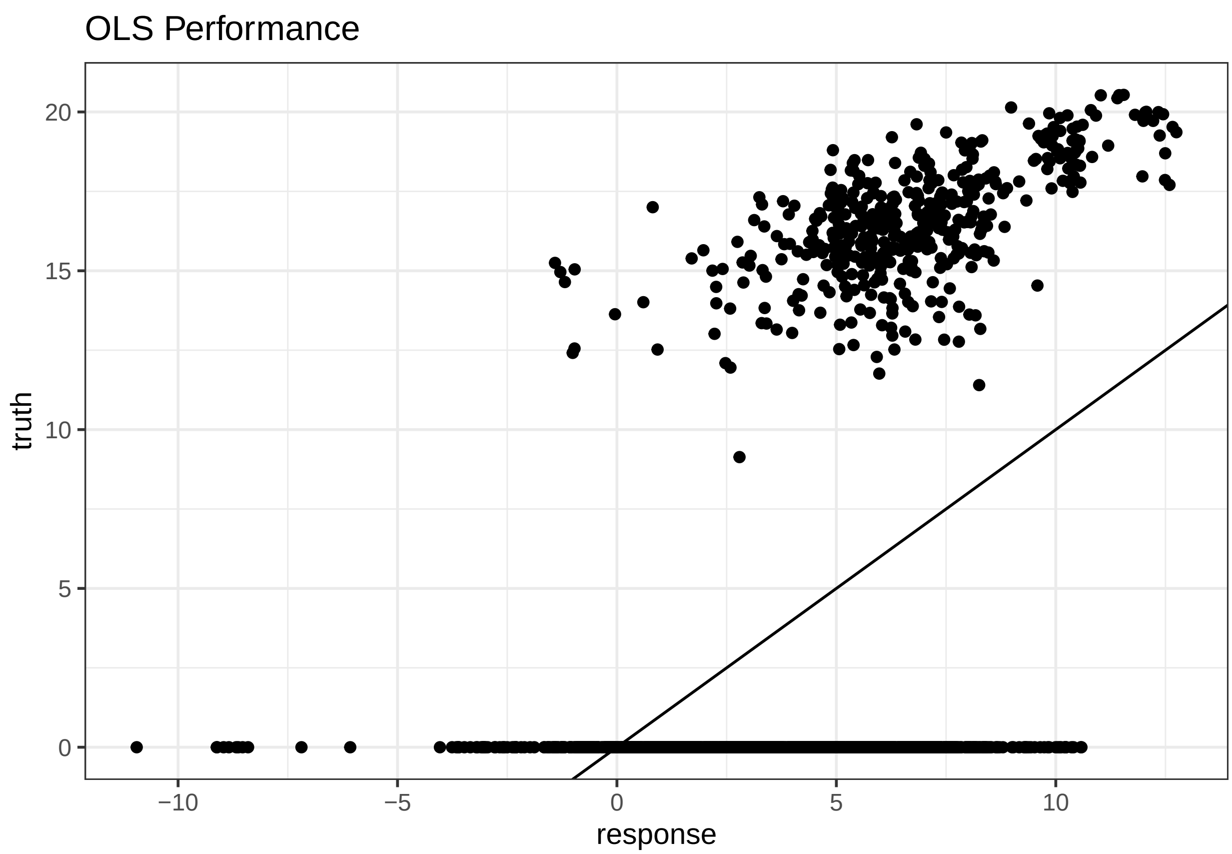
```r
ggplot(predictRfs[["total"]]$data, aes(x = response, y = truth)) +
    geom_point() + theme_bw() +
    geom_abline(intercept = 0, slope = 1) +
    labs(title = "Random Forest Performance")
```




```
ggplot(predictGams[["total"]]$data, aes(x = response, y = truth)) +
    geom_point() + theme_bw() +
    geom_abline(intercept = 0, slope = 1) +
    labs(title = "GAM Performance")
```




```r
ggplot(predictOlss[["total"]]$data, aes(x = response, y = truth)) +
    geom_point() + theme_bw() +
    geom_abline(intercept = 0, slope = 1) +
    labs(title = "OLS Performance")
```



In an absolute sense, visually inspecting the performance of the models indicates none of them did a particularly great job in fitting the data. It would appear as if the large number of 0 values is throwing all of them off.

Question 4 - Variable Importance and GAM feature interpretation

```
# Run predictions on the full data to get variable importance
PredictFull <- function(model, missingFeat, algo) {
    if(missingFeat != "total")
        p <- predict(model, newdata = select(dataCleanShort, -lyenloans2, -missingFeat))
    else
        p <- predict(model, newdata = select(dataCleanShort, -lyenloans2))

    p$data["truth"] <- dataCleanShort[["lyenloans2"]]

    p$data %>%
        mutate(error = response - truth,
               error_sq = error ^ 2,
               feature = missingFeat,
               algo = algo) %>%
        group_by(feature, algo) %>%
        summarise(sum_sq_error = sum(error_sq),
                  rmse = sqrt(sum_sq_error / n())) %>%
        ungroup() %>%
        select(-sum_sq_error)
}
```

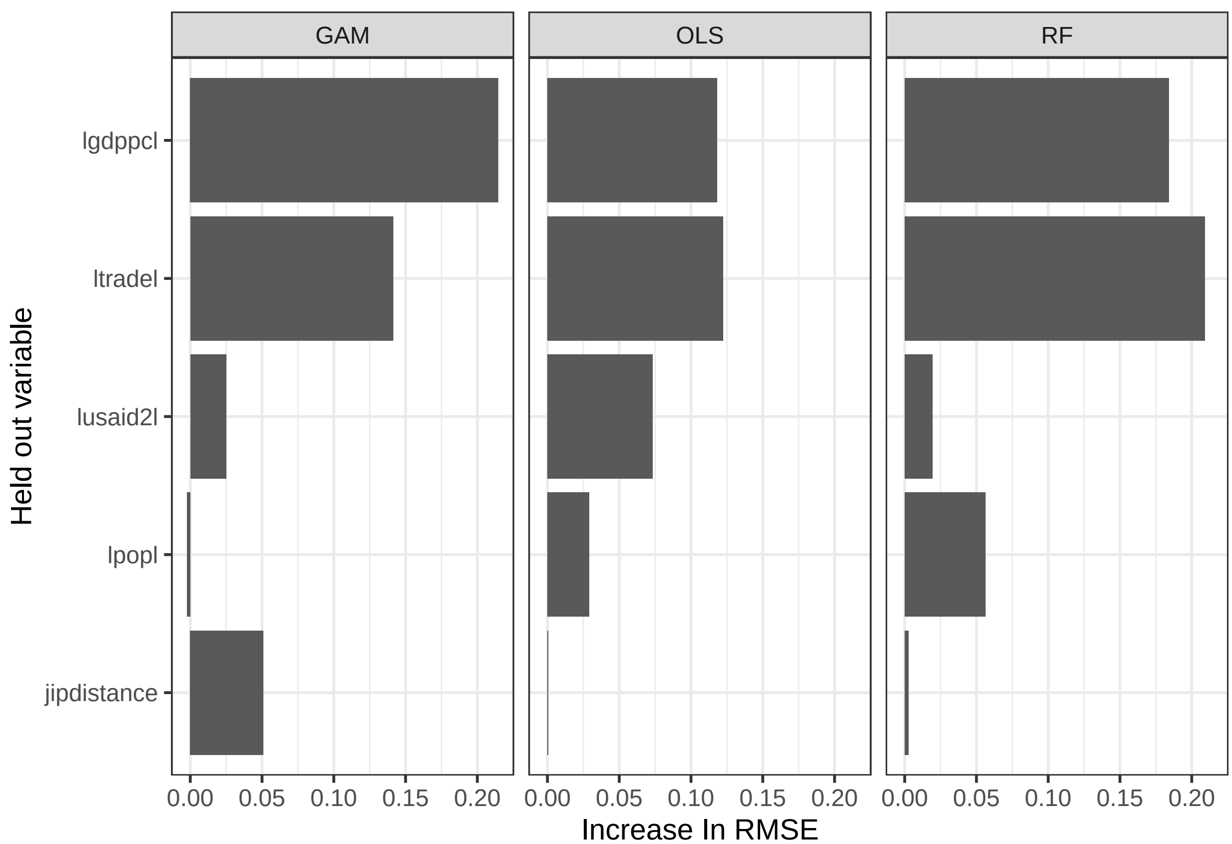
```r
predictOlssFeat <-
    pmap_dfr(list(trainedOlss, names(trainedOlss), "OLS"), PredictFull) %>%
    pivot_wider(names_from = feature, values_from = rmse) %>%
    pivot_longer(-c(algo, total)) %>%
    mutate(difference = value - total)

predictGamsFeat <-
    pmap_dfr(list(trainedGams, names(trainedGams), "GAM"), PredictFull) %>%
    pivot_wider(names_from = feature, values_from = rmse) %>%
    pivot_longer(-c(algo, total)) %>%
    mutate(difference = value - total)

predictRfsFeat <-
    pmap_dfr(list(trainedRfs, names(trainedRfs), "RF"), PredictFull) %>%
    pivot_wider(names_from = feature, values_from = rmse) %>%
    pivot_longer(-c(algo, total)) %>%
    mutate(difference = value - total)

featImportance <- bind_rows(predictOlssFeat, predictGamsFeat, predictRfsFeat)

ggplot(featImportance, aes(x = difference, y = reorder(name, difference))) +
    geom_bar(stat = "identity") +
    facet_wrap(~algo) +
    theme_bw() +
    labs(x = "Increase In RMSE", y = "Held out variable")
```

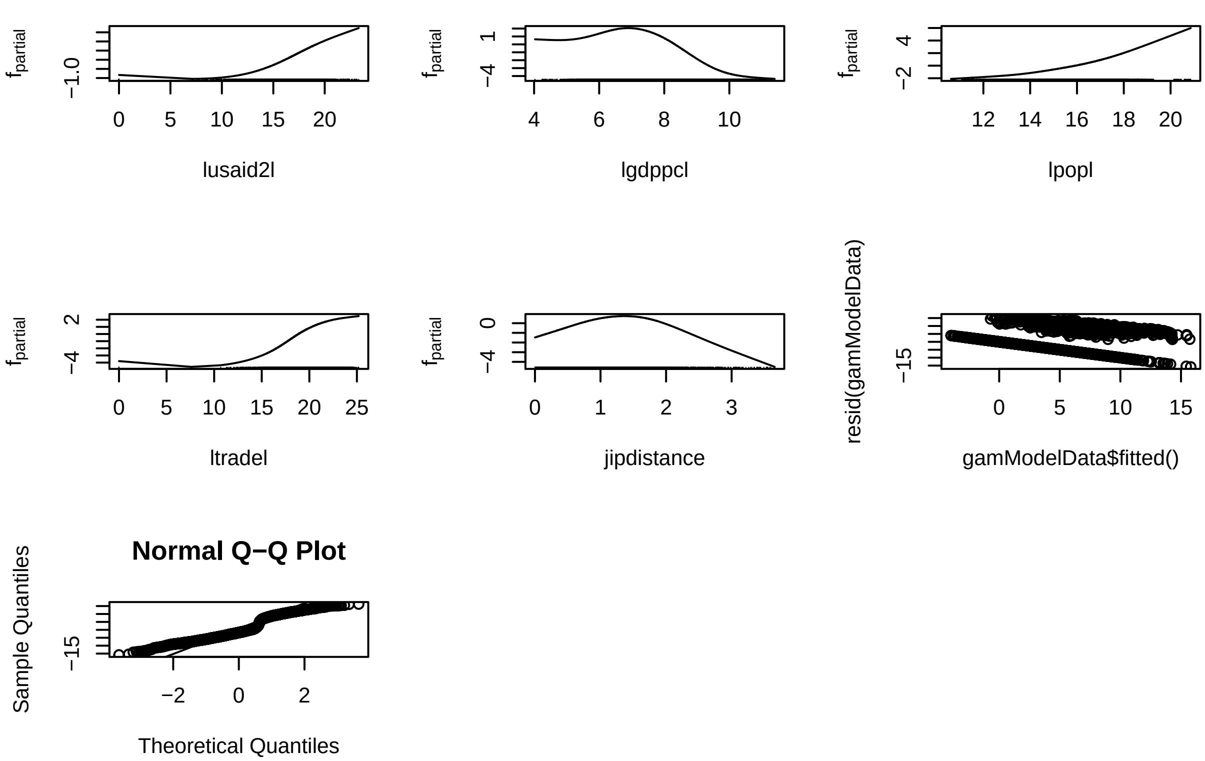


Interestingly there is a moderate amount of agreement among the models in terms of variable importance. Both lgdppcl (per capita GDP) and ltradel (amount of trade between the two countries) have the highest increases in RMSE when they're removed for OLS, GAM, and Random Forest. The GDP variable had a large coefficient in the estimated model from the paper, but it had a large standard error preventing it from achieving statistical significance. The trade variable had a moderately large coefficient, and it did achieve significance.

The other three variables vary in importance, though, across the three sets of models. The amount of aid given to the country by the USA, in general, does not add much to predictive power which is interesting because it's framed as the central variable of importance in the paper.

```r
# Obtain model data from the GAM model fitted on the full data
gamModelData <- getLearnerModel(trainedGams[["total"]])

par(mfrow = c(3, 3))
# create line plots for each function learned for each variable
# shows how much each predictor contributes to ozone estimate across its values
plot(gamModelData, type = "l")
plot(gamModelData$fitted(), resid(gamModelData)) # residuals vs. fitted values
qqnorm(resid(gamModelData)) # quantile-quantile plot
qqline(resid(gamModelData))
par(mfrow = c(1, 1))
```



Nearly every variable has a nonlinear relationship with the predictor variable (amount of loans given out by Japan) except for population. The relationship is slightly nonlinear for population, but it generally has a relatively consistently increasing relationship where as population increases so too, generally, does the loan amount given.

These results are very intriguing, but there is also cause for concern. The residuals vs. fitted plot shows a definitive pattern indicating heteroscedasticity is a problem. The Q-Q plot also indicates the residuals are likely not drawn from a random distribution. All in all these results call into question how valid regression based approaches are for modeling this data.