Code Appendix

Eduardo Martinez

Contents

Important Notes

This appendix is for my main code file: `CompleteAnalysis.Rmd`.

Many code chunks are very similar with subtle changes. For instance, I often run the same code for each model, but just change the model name. In these cases, for the appendix, I will only show and explain the code for 1-2 models.

Quick comments about an entire code chunk are added at the top of the code chunk, quick comments about a specific line within a code chunk are added to the right of the line, and further details appear below/after (example below).

```r
1  # Sample Commenting Structure
2  UncommentedLine <- "Straightforward - No Comment Necessary"
3  CommentedLine <- "Commented Text"                          # Comment About Just This Line
```

Further details that are unnumbered refer to the whole chunk.

(3) Red numbers listed refer to details regarding code in the corresponding line number.

0.0.1 Section 0

```r
1  source("/Users/Eduardo/Desktop/R/MyFunctions.R")
```

I created several of my own functions in an R script (`MyFunctions.R`), which I used throughout my code.

(1) `MyFunctions.R` was loaded to the local environment of `CompleteAnalysis.Rmd`.

IMPORTANT!!! In order for my code to execute properly in your computer, change `source("/Users/Eduardo/Desktop/R/MyFunctions.R")` to the location you have the file saved on your computer.

```r
# Loading All Packages Required to Execute My Main Code File
library(scales)     # To visualize My Colors below and provides font families
library(knitr)      # For LaTeX Tables
library(kableExtra) # For LaTeX Tables with advanced customization
library(leaps)      # For finding best subset models via exhaustive search
library(car)        # For regression (VIF of Predictors)
library(modelr)     # For regression (MAE, MSE, RMSE)
library(performance)# Get model performance stats and provides additional fonts
library(tidyverse)  # Fundamental R package of libraries used for data processing, visualizations, etc.
library(ggrepel)    # Add labels to ggplots while prevent overlaps
```

```r
1 mycolors <- c("#cc3333", "#3B00fb", "#737373", "#04551f", "#782ab6", "#daa500",
2               "#753232", "#84e9ec", "#000000", "#00cd00", "#ff00c0", "#ffff00",
3               "#f98b85", "#005e7d", "#babab1", "#0d987d", "#330066", "#856601",
4               "#ff5000", "#66B0ff", "#685a4e", "#00ff99", "#ca9ec2", "#fbbb05",
5               "#a11f48", "#5b09ba", "#2c2520", "#044111", "#fe9aaa", "#e0cd67")
6 scales::show_col(mycolors, cex_label = 0.95)                       # Plot to Visually Examine Colors
```

I decided to manually define 30 colors using Hex Codes such that the colors contrast each other well for visualizations. I only ended up using 28 of these colors: 1 for Life Expectancy (LE) and 27 for each of the 27 predictors.

(1-5) Colors were inputed row by row (left to right); these colors really contrast each other well (see plots throughout).

(6) Observe the actual plot produced in `CompleteAnalysis.Rmd`. Each isolated columns (top to bottom) consist of colors coming from the same core color: red/orange, blue, gray/black, green, purple/pink, yellow.

If confusing, all you need to know is that I used these colors in order to easily distinguish variables in my visualizations.

```r
DemNames <- c("Total Population", "\\% Male", "\\% Female",
              "\\% Non-Hispanic White", "\\% Non-Hispanic Black",
              "\\% Hispanic or Latino", "\\%  Asian or Pacific Islander",
              "\\% Children (0-17 years)", "\\% Young Adults (18-39 years)",
              "\\% Middle-Aged Adults (40-64 years)", "\\% Seniors (65 and older)")
ACSNames <- c("Average Life Expectancy",                                  # 2015-2019 5-Year Averages
              "Number of Federally Qualified Health Centers",             # Total Counts
              "Median Household Income", "Per Capita Income",             # USD
              "Poverty Rate",                        # % of Residents in Families Below Federal Poverty Level
              "Unemployment Rate",                   # % of Residents >= 16 Actively Seeking Employment
              "Preschool Enrollment",                                  # % of Toddlers Ages 3-4
              "High School Graduation Rate", "College Graduation Rate",         # % of Residents >= 25
              "Limited English Proficiency",                           # % of Residents >= 5
              "Foreign Born", "Uninsured Rate",                        # % of Residents
              "Ambulatory Difficulty", "Cognitive Difficulty",         # % of Residents
              "Hearing Difficulty", "Independent Living Difficulty",   # % of Residents
              "Self Care Difficulty", "Vision Difficulty",             # % of Residents
              "Food Stamps (SNAP)", "Public Assistance Income (Cash Welfare)",  # % of Household
              "Households in Poverty Not Receiving SNAP",         # % of Households Below Federal Poverty Level
              "Rent Burdened", "Severely Rent Burdened",          # % of Renter-Occupied Households
              "Single Parent Households",                              # % of Households
              "Vacant Housing",                                        # % of Housing Units
              "Crowded Housing",                                       # % of Occupied Housing Units
              "Economic Diversity Index",                              # Score Ranging Between 0-0.83
              "Hardship Index")                                        # Score Ranging Between 0-100
```

Full variables names for each data set were stored in `DemNames` and `ACSNames` for reference, but only `DemNames` was used (once in `Section 1.1`). Units are provided for each ACS variable in the comments to the right of each line.

1 The Data

1.1 Demographics

```r
1  LoadDem <- read_csv("Data/CSV/Demographics.csv", show_col_types = F)
2  DemDF <- cbind(LoadDem[,1:5], as_tibble(apply(LoadDem[,6:15], 2, function(x) (x / LoadDem$POP) * 100)))
3  ChiDem <- DemDF[1,]
4  CADem <- DemDF[-1,]
```

(1) Loaded the data containing demographics for Chicago, IL and each of its 77 Community Areas (CAs).

(2) `DemDF` converts totals into percentages.

(3) `ChiDem` contains data for Chicago, IL only.

(4) `CADem` contains data for the 77 CAs only.

```r
1  ChiDemStats <- tibble(Demographic = DemNames, Chicago = round(unlist(select(ChiDem, POP:Seniors)), 2))
2  CADemStats <- sum.tab(CADem[,5:15], exclude=c("Var", "CV", "SD"), digits=2) %>%
3    mutate(Demographic = fDemNames) %>%
4    select(Demographic, Min:Max)
```

This is the only time I used `DemNames` defined in `Section 0`.

(1) `ChiDemStats` provides demographics for Chicago, IL overall.

(2-4) `CADemStats` stores summary statistics in a tibble (data frame).

(2) I created the function `sum.table`, which can be found in the script file, `MyFunctions.R`

```r
1  AllDemDF <- cbind(ChiDemStats, CADemStats[,-1]) %>%
2    mutate_if(.predicate = is.double,
3              function(x) ifelse(x > 1500,
4                yes = paste("$\\boldsymbol{",
5                            format(round(x), big.mark = ",",  drop0trailing = T, scientific = F),
6                            "}$", sep = ""),
7                no = paste("$\\boldsymbol{",
8                           format(x, scientific = F, nsmall = 2, trim = T),
9                           "}$", sep = "")
10               ))
```

(1) Recombined the tibbles containing statistics for Chicago and each CA.

(2-9) To fit columns in the screen, I did some fancy formatting. Also, I made each numeric value bold.

```r
1  kbl(AllDemDF, align = "lrrrrrrr", linesep = "", escape = F,
2        col.names = paste("$\\underline{\\textbf{", names(AllDemDF), "}}$")) %>%
3    kable_styling(bootstrap_options = c("hover", "striped"),
4                  htmltable_class = "lightable-classic",         # Built-In HTML Table Style
5                  html_font = "Courier", font_size = 15) %>%
6    row_spec(0, align = "c", font_size = 16, extra_css = 'font-family: "serif;"') %>%
7    column_spec(1, width = "30em", bold = T) %>%
8    column_spec(2:8, width = "7em")
```

Several chunks are similar to the one above. I will only provide details once for the code in the chunk above.

(1) `kbl()` converts the tibble `AllDemDf` into a nicely formatted LaTeX or HTML table.
`align = "lrrrrrrr"` refers to the vertical alignment for each of the 8 columns in this data table. The first column is left-aligned and the rest are right-aligned.
`linesep = ""`: Adjust spacing between rows such that no space is added.
`escape = F`: To avoid conflicts between HTML and LaTeX special characters, I always set this to FALSE.

(2) I used LaTeX to modify the column names so that they would be in bold and underlined.

(3) `"hover"` makes it so that the table rows light up when the mouse moves over them.
`"striped"` makes even row numbers have different background colors.

(6) Header row is center-aligned, contains slightly larger font, and the font-family is `serif` instead of `Courier`.

Lines `column_spec` lines and all other settings are straightforward.

1.2 Life Expectancy, FQHC, and ACS Data

```r
1 LoadACS <- read_csv("Data/CSV/Main-Data.csv", show_col_types = F) # Loading Main ACS Data Set
2 CADF <- LoadACS[-1,]                                               # 77 CAs Only (Chicago Excluded)
3 CA2 <- filter(CADF, Name != "Burnside", Name != "Fuller Park")     # `CA2` Excludes Burnside and Fuller Park
```

```r
1 MeltCADF <- select(CADF, LE:HardshipIndex) %>%
2   pivot_longer(cols = 1:28, names_to = "Variable", values_to = "Value") %>%
3   mutate(Variable = factor(Variable, levels = names(CADF)[-1:-3]))
```

I took all 28 variables and stored there names in a column labeled ”Variable” and their values in column ”Value”.

(3) `factor(Variable)` makes transforms names from ordinary strings to categorical variables. `levels` are unique categories specified in order. I set the `levels` equal to the 28 variable names in the order they are listed in the data. If levels are not specified, then the factors will be ordered alphabetically.

```r
1  # Box plots for each variable are generated.
2  ggplot(data = MeltCADF, aes(x = Variable, y = Value)) +
3
4    geom_boxplot(aes(fill = Variable), show.legend = F,
5                 color = ifelse(unique(MeltCADF$Variable) == "College", "#daa500", "black"),
6                 outlier.color = "Red3", outlier.alpha = 0.55,          # alpha range: 0 (transparent) - 1 (not transparent
7                 outlier.size = rel(1.25), outlier.stroke = rel(1)) +   # stroke is an additional size modification
8    facet_wrap(vars(Variable), scales = "free") +
9
10   labs(title = "Box Plots for Each Variable Considered", x = NULL, y = NULL) +  # x and y labs implied in plot
11   scale_fill_manual(values = mycolors) +
12   scale_y_continuous(n.breaks = 4, labels=scales::comma) +
13
14   theme_bw() +           # Built-In Black and White Theme
15   theme(plot.title = element_text(hjust = 0.5, vjust = 2, face = "bold", size = rel(1.5), family="Courier"),
16         strip.text = element_text(hjust = 0.5, face = "bold", size = rel(0.9), color = "White"),
17         strip.background = element_rect(fill = "#005e7d"),
18         panel.grid.major.y = element_line(color = "grey74"),
19         panel.grid.minor.y = element_line(color = "grey84"),
20         panel.grid.major.x = element_blank(),
21         panel.grid.minor.x = element_blank(),
22         axis.text.y = element_text(size = rel(0.95), color = "black"),
23         axis.text.x = element_blank(),
24         axis.ticks.x = element_blank())
```

The repeatedly used `rel()` function is used to make size modifications adjust based on the document environment dimensions. That is, `rel()` makes the sizes proportional to the window they are being viewed in.

(2) All my visualizations begin with `ggplot()`, which creates a blank plot window. If the data and the x and y variables to plot are specified in `ggplot()` under `data =` and `aes()`, respectively, then they are automatically applied to all subsequent functions beginning with `geom`.

(4) In addition to the aesthetics (`aes()`) specified in `ggplot()`, I added aesthetics so that each variable generate a separate box plot that is distinguished by the color used to fill the box.

(5) I like to set the box line colors to black, but one of `mycolors`, which I used to fill the boxes, is already black. Consequently, it will be impossible to see the black lines. Therefore, I used an `ifelse()` statement to make only the black filled box use different colored lines.

(8) `face_wrap()` generates a separate plot window for each variable.

(12) Each unique y-axis will have roughly 4 main breaks, and big numbers will include commas.

(13-23) `theme_bw()` and `theme()` are styling modifications that are not essential, and, thus, I will not explain them in detail.

1.3 Training and Test Set

```r
set.seed(9711) # Same random numbers sampled every time
trainRows <- sample(x = c(1:36, 38:46, 48:77), size = 45, replace = F)
```

I randomly selected 45 GEOIDs from 75 total GEOIDs. While there are 77 total CAs with GEOIDs, Fuller Park and Burnside, which have GEOIDs 37 and 47, respectively, were excluded because they are potential outliers.

```r
1  # Producing Data Frames for Training Set 1, Training Set 2, and the Test Set
2  preTrainDF1 <- filter(CADF, GEOID %in% c(37, 47, trainRows))
3  trainDF1 <- preTrainDF1[c(which(preTrainDF1$GEOID %in% c(37, 47)), which(preTrainDF1$GEOID %in% trainRows)), ]
4  trainDF2 <- trainDF1[-1:-2,]          # Selecting all 45 CAs except Burnside and Fuller Park
5  testDF1 <- filter(CADF, GEOID %notin% c(37, 47, trainRows))  # Contains all CAs not in trainDF1
6  
7  n1a <- nrow(trainDF1)     # n1a = 47 (CAs in Training Set 1)
8  n2a <- nrow(trainDF2)     # n2a = 45 (CAs in Training Set 2)
9  n1b <- nrow(testDF1)      # n1b = 30 (CAs in Test Set)
```

(2) I created a data frame containing Burnside, Fuller Park, and the 45 CAs with the GEOIDs randomly sampled in the previous chunk.

(3) `trainDF1` is an organized version of `preTrainDF1` with Burnside and Fuller Park located at rows 1 and 2, respectively, and the remaining 45 CAs are organized alphabetically in rows 3-47.

---

2 Full Model 1 (FM_1) vs Full Model 2 (FM_2)

2.1 Regression Summaries

```r
1  s.FM1 <- summary(FM1 <- lm(LE ~ ., data = trainDF1[-1:-3]))
```

I created a linear model that I called FM1 (Full Model 1), and saved its summary into a variable called s.FM1 all in one line; I did the same for FM2.

(1) Notice, I set `data = trainDF1[-1:-3]` because the first three columns are `Name`, `Label`, and `GEOID`, all of which all of which I do not want to use to predict Life Expectancy (LE).

2.1.1 FM$_1$

```r
1  # Converting the Regression Summary into a LaTeX Table
2  kbl(s.Mod2LaTeX(s.FM1, label = "\\boldsymbol{FM_1}"), align = "lrrrc", linesep = "", escape = F) %>%
3  
4    kable_styling(bootstrap_options = c("hover", "striped"), htmltable_class = "lightable-classic",
5                  html_font = "Courier", font_size = 16, full_width = F) %>%
6  
7    row_spec(0, align = "c", font_size = 17, extra_css = 'font-family: "serif;"') %>%
8  
9    column_spec(1, width = "12em") %>%
10   column_spec(2:4, width = "10em") %>%
11   column_spec(5, width = "6em")
```

(2) The function `s.Mod2LaTeX()` is defined in `MyFunctions.R`; it converts a regression summary into a nice LaTeX table.

2.2 Related Data

2.2.1 FM$_1$

```r
# Created a tibble containing key values used in the next section: 2.3 Diagnostic plots
FM1.DF <- tibble(Index = 1:47, trainDF1[,2:4],
                 Fit = FM1$fitted.values, Residual = FM1$residuals, Std_Res = rstandard(FM1),
                 Leverage = hatvalues(FM1), CooksD = cooks.distance(FM1))
as_tibble(cbind(FM1.DF[,1:3], round(FM1.DF[,-1:-3], 3)))      # Rounded numeric values to 3 decimal places
```

2.3 Diagnostic Plots

2.3.1 Residuals vs. Fits

2.3.1.1 FM$_1$

```
1  # This chunk generates a scatter plot of the Residuals (y-axis) vs Fitted Values (x-axis).
2  FM1.RvF <- ggplot(FM1.DF, aes(x = Fit, y = Residual)) +
3  
4    geom_point(shape = 1, size = rel(3), color = "black") +
5  
6    geom_abline(slope = 0, intercept = 0, color = "Red1", linetype = 2, size = rel(0.65), alpha = 0.75) +
7  
8    stat_smooth(formula = y*(2/3) ~ x, method = "loess", se = F, color = "#3B00FB", size = rel(0.5), span = 0.8) +
9  
10   geom_label_repel(aes(label = ifelse(Label == "FullerPk" | Label == "Burnside", Label,'')),
11                    nudge_x = -2, nudge_y = ifelse(FM1.DF$Label == "FullerPk", 0.75, -0.75),
12                    size = rel(3.75), box.padding = 0.3, label.padding = 0.3) +
13 
14   labs(title = "Full Model 1  -  Residuals  vs.  Fitted Values", x = "Fitted Values", y = "Residuals") +
15 
16   scale_x_continuous(limits = c(65.6, 84.4), breaks = seq(65, 85, 5)) +
17   scale_y_continuous(limits = c(-1.9, 1.9), breaks = c(-2, -1, 0, 1, 2)) +
18 
19   theme_bw() +
20   theme(plot.title = element_text(face = "bold", size = rel(1.5), color = "#3B00FB", family = "serif"),
21         panel.grid.major = element_line(color = "gray78"),
22         panel.grid.minor = element_line(color = "gray84"),
23         axis.title.y.left = element_text(size = rel(1.25), face = "bold", family = "sans",
24                                          margin = margin(t=0, r=8, b=0, l=3)),
25         axis.title.x.bottom = element_text(size = rel(1.25), face = "bold", family = "sans",
26                                            margin = margin(t=8, r=0, b=3, l=0)),
27         axis.text = element_text(size = rel(1.2), face = "bold", family = "Courier"))
```

(4) Setting `shape = 1` makes the points open circles instead of solid points, which makes it easier to see if multiple points are overlapping.

The red and blue lines are used to assess a key linear regression assumption about the residuals (e_i):

$$e_i \sim N(0, \sigma^2)$$

This means that the points should have mean equal to zero and constant variance from left to right. In other words, they should be approximately symmetric across the x-axis.

(6) I overlaid a red dashed horizontal line on the x-axis used to highlight the zero mean location.

(8) Running the generic `plot(FM1, which=1)` plots the Residuals vs Fitted Values with a red trend `loess` line. I researched how this line is generated and tried to replicate the line in a `stat_smooth()` environment. Instead of making the formula $y \sim x$, I multiplied $y * (2/3)$, which makes the `loess` line less sensitive to single y-values. The `span` setting controls the line smoothness. Overall, line (8) transform the `loess` line such that the line should be close to 0 for all x-values; otherwise, the mean is not equal to 0. Also, if the line has periods that deviate sharply from the x-axis, this suggest that the variance is not constant/symmetric.

(10) I labeled Burnside and Fuller Park because I was investigating the possibility that these CAs represent outliers. The function `geom_label_repel` is from the `ggrepel` package. It assures no labels overlap.

(11-12) These are just styling modifications.

(16) `limits = c(65.6, 84.4)` The x-axis consist of values between 65.7 and 84.4.
`breaks = seq(65, 85, 5)` Major tick marks are located at $y = 65, 70, 75, 80, 85$.

(17) Similar modifications as in line (16), but for the y-axis.

2.3.2 Normal Q-Q Plot

```r
1  # Used in all Models
2  q.x <- qnorm(c(.25, .75)) # z-scores
3  
4  # Full Model 1 Calculations
5  FM1.y <- quantile(FM1.DF$Std_Res, c(.25, .75))
6  FM1.slope <- diff(FM1.y) / diff(q.x)
7  FM1.int  <- FM1.y[1] - FM1.slope * q.x[1]
8  FM1.q <- sort(rank(FM1.DF$Std_Res) - 0.5) / nrow(FM1.DF)
9  FM1.theoQ = qnorm(FM1.q, mean = mean(FM1.DF$Std_Res), sd = sd(FM1.DF$Std_Res))
10 FM1.qq <- tibble(Std_Res = sort(FM1.DF$Std_Res), TheoQ = FM1.theoQ)
11 
12 FM1.qqDF <- inner_join(FM1.DF, FM1.qq, by = "Std_Res") %>%
13   select(Index:Std_Res, TheoQ, everything()) %>%
14   arrange(Std_Res)
15 FM1.dist <- matrix(c(FM1.slope * FM1.qqDF$TheoQ + FM1.int, FM1.qqDF$Std_Res), ncol = 2)
16 FM1.qqDF2 <- FM1.qqDF %>%
17   mutate(QQDist = apply(FM1.dist, 1, function(x) dist(x, method = "euclidean")))
```

Note, `plot(FM1, which=2)` easily a basic Normal Q-Q plot. I decided to manually generate it in order to manually label extreme points. Since there are quick alternative options for generating a Normal Q-Q plot, the computations are not very important, so I will not explain each line of code.

(17) I will explain my calculations for `QQDist` because it is an added computation that was not required. `QQDist` measures the euclidean distance between each data point and the Normal Q-Q line (blueish-purple line in the FM1 plot below). I added `QQDist` in order to identify points that are relatively far from the Normal Q-Q line. Such points are potential outliers because they deviate away from the assumed/theoretical normal distribution.

2.3.2.1 FM$_1$

```r
1  FM1.qqPlot <- ggplot(FM1.qqDF2, aes(x = TheoQ, y = Std_Res)) +
2  
3    geom_abline(slope = FM1.slope, intercept = FM1.int, linetype=1, size = rel(1.5), color = "#3B00FB", alpha=0.65) +
4  
5    geom_point(shape = 1, size = rel(4), color = "black") +
6  
7    geom_label_repel(
8      aes(label = ifelse(abs(Std_Res) > 2 | QQDist > 0.5 | Label == "FullerPk" | Label == "Burnside", Label,'')),
9      nudge_x=ifelse(FM1.qqDF2$Std_Res == max(FM1.qqDF2$Std_Res), -0.75, 0),
10     nudge_y=ifelse(FM1.qqDF2$Std_Res == min(FM1.qqDF2$Std_Res) | FM1.qqDF2$Std_Res == max(FM1.qqDF2$Std_Res), 1,-1.25),
11     size = rel(3.75), box.padding = 0.3, label.padding = 0.3) +
12 
13   labs(title = "Full Model 1  -  QQ Plot", x = "Theoretical Quantiles", y = "Standardized Residuals") +
14 
15   scale_x_continuous(limits = c(-2.9, 2.9), breaks = seq(-3, 3, 1)) +
16   scale_y_continuous(limits = c(-2.9, 2.9), breaks = seq(-3, 3, 1)) +
17 
18   theme_bw() +
19   theme(plot.title = element_text(face = "bold", size = rel(1.5), color = "#3B00FB", family = "serif"),
20         panel.grid.major = element_line(color = "gray78"),
21         panel.grid.minor = element_blank(),
22         axis.title.y.left = element_text(
23           size = rel(1.25), face = "bold", family = "sans", margin = margin(t=0, r=8, b=0, l=3)),
24         axis.title.x.bottom = element_text(
25           size = rel(1.25), face = "bold", family = "sans", margin = margin(t=8, r=0, b=3, l=0)),
26         axis.text = element_text(size = rel(1.2), face = "bold", family = "Courier"))
```

(3) Produces Normal Q-Q Line

(8) I labeled Burnside and Fuller Park, points with Standardized Residuals > 2, and/or Euclidean Distance > 0.5.

2.3.3 Point Leverage

$$\underline{\text{Point Leverage}} : \boxed{h_i = H_{ii} = diag(H)} \;, \quad \text{where} \;\; H = X(X^T X)^{-1} X^T$$

$$\underline{FM_1 - \text{Cutoff Value for Point Leverage}} = \frac{2(p+1)}{n_1} = \frac{2(28)}{47} \approx 1.19$$

$$\underline{FM_2 - \text{Cutoff Value for Point Leverage}} = \frac{2(p+1)}{n_2} = \frac{2(28)}{45} \approx 1.2\overline{4}$$

2.3.3.1 FM$_1$

```r
1  maxLevFM1 <- (2 * length(coef(FM1))) / nrow(FM1.DF)   # The Cutoff Value
2  
3  # Point Leverage Plot for FM1
4  LevPlotFM1 <- ggplot(data = FM1.DF, aes(x = Index, y = Leverage)) +
5  
6    geom_hline(yintercept = maxLevFM1, color = "Red1", linetype = 2, size = rel(0.75)) + # Cutoff Line
7  
8    geom_linerange(aes(ymin = 0, ymax = Leverage), color = "#3B00FB", size = rel(0.9)) +
9  
10   geom_point(size = rel(2), color = "black") +
11 
12   geom_label_repel(
13     aes(label = ifelse(Leverage > maxLevFM1 | Label == "FullerPk" | Label == "Burnside", Label,'')),
14     nudge_y = 0.08, nudge_x = 0.15, size = rel(3.75), label.r = 0.5, box.padding = 0.3, label.padding = 0.3) +
15 
16   geom_label_repel(
17     data = filter(FM1.DF, Index == 27),
18     aes(x = 27.5, y = maxLevFM1, label = paste0("Cutoff Value = ", round(maxLevFM1, 5), collapse = "")),
19     color = "Red1", size = rel(3.75), family = "Arial",
20     nudge_y = -0.095, nudge_x = 0, label.r = 0.5, box.padding = 0.3, label.padding = 0.3) +
21 
22   labs(title = "FM 1  -  Point Leverage", x = "Index", y = "Leverage") +
23 
24   scale_x_continuous(limits = c(1, 47), breaks = seq(5, 85, 10)) +
25   scale_y_continuous(limits = c(0, 1.25), breaks = seq(0, 1.2, 0.2)) +
26 
27   theme_bw() +
28   theme(plot.title = element_text(face = "bold", size = rel(1.5), color = "#3B00FB", family = "serif"),
29         panel.grid.major = element_line(color = "gray78"),
30         panel.grid.minor = element_line(color = "gray84"),
31         axis.title.y.left = element_text(
32           size = rel(1.25), face = "bold", family = "sans", margin = margin(t=0, r=8, b=0, l=3)),
33         axis.title.x.bottom = element_text(
34           size = rel(1.25), face = "bold", family = "sans", margin = margin(t=8, r=0, b=3, l=0)),
35         axis.text = element_text(size = rel(1.2), face = "bold", family = "Courier"))
```

(8) `geom_linerange()` is similar to a column chart except each column is a vertical line.

(13) The code labels points with leverage exceeding the cutoff value, but no points did. Thus, only Burnside and Fuller Park were labeled to assess their influence.

(16-20) I labeled the cutoff line with the actual cutoff value.

(17) To ensure the label is not repeated, I had to filter the data to only one observation located at Index 27. I selected Index 27 because I wanted the label to be placed around $x = 27$.

2.3.4 Cook's Distance

$$\underline{\text{Cook's Distance}}: \quad D_i = \frac{r_i^2}{p+1} \cdot \frac{h_i}{1-h_i}$$

$$\underline{\text{Cutoff Value for Cook's Distance}}: \quad D_i > F(\ p = 0.5, \quad df_1 = p+1, \quad df_2 = n-p-1\ )$$

$$\hookrightarrow \ \text{FM 1}: \quad D_i > F(\ p = 0.5, \quad df_1 = 28, \quad df_2 = 19\ ) \approx 0.0466$$

$$\hookrightarrow \ \text{FM 2}: \quad D_i > F(\ p = 0.5, \quad df_1 = 28, \quad df_2 = 17\ ) \approx 0.0504$$

2.3.4.1 FM$_1$

```r
# Cutoff Value
maxCookFM1 <- pf(0.5, 28,  19)

# Plot of Each Points Cook's Distance
CookPlotFM1 <- ggplot(data = FM1.DF, aes(x = Index, y = CooksD)) +

  geom_hline(yintercept = maxCookFM1, color = "Red1", linetype = 2, size = rel(0.75)) +

  geom_linerange(aes(ymin = 0, ymax = CooksD), color = "#3B00FB", size = rel(0.9)) +

  geom_point(size = rel(2), color = "black") +

  geom_label_repel(
    aes(label = ifelse(CooksD > 0.15 | Label == "FullerPk" | Label == "Burnside", Label,'')),
    nudge_x = ifelse(FM1.DF$Label == "FullerPk" | FM1.DF$Label == "Chatham", 1.25 ,0), nudge_y = 0.085,
    size = rel(3.75), label.r = 0.5, box.padding = 0.3, label.padding = 0.3) +

  geom_label_repel(
    data = filter(FM1.DF, Index == 26),
    aes(x = 29.25, y = maxCookFM1, label = paste0("Cutoff Value = ", round(maxCookFM1, 4), collapse = "")),
    color = "Red1", size = rel(3.75), family = "Arial",
    nudge_y = 0.3, nudge_x = 0, label.r = 0.5, box.padding = 0.3, label.padding = 0.3) +

  labs(title = "FM 1  -  Cook's Distance", x = "Index", y = "Cook's Distance") +

  scale_x_continuous(limits = c(1, 47), breaks = seq(5, 85, 10)) +
  scale_y_continuous(limits = c(0, 0.85), breaks = seq(0, 0.8, 0.2)) +

  theme_bw() +
  theme(plot.title = element_text(face = "bold", size = rel(1.5), color = "#3B00FB", family = "serif"),
        panel.grid.major = element_line(color = "gray78"),
        panel.grid.minor = element_line(color = "gray84"),
        axis.title.y.left = element_text(
          size = rel(1.25), face = "bold", family = "sans", margin = margin(t=0, r=8, b=0, l=3)),
        axis.title.x.bottom = element_text(
          size = rel(1.25), face = "bold", family = "sans", margin = margin(t=8, r=0, b=3, l=0)),
        axis.text = element_text(size = rel(1.2), face = "bold", family = "Courier"))
```

The code for this plot is basically the same as the previous point leverage plot. The only difference is Cook's distance is plotted and the corresponding cutoff values.

2.4 VIF

```r
1 `FM Predictors` <- names(vif(FM1))            # Storing the name of each predictor
2 `FM1 VIF` <- vif(FM1); `FM2 VIF` <- vif(FM2)  # Storing the VIF for each FM
3
4 FM.VIF.DF <- tibble(`FM Predictors`, `FM1 VIF`, `FM2 VIF`) %>%
5   arrange(desc(`FM1 VIF` + `FM2 VIF`))
```

The `car` package has a `vif()` function where the input is linear model, and it outputs the VIF values for each predictor in the model.

(1) The names for FM1 and FM2 are the same since they both are full models.

(3) The `vif()` function outputs the VIF values as a vector. I converted this to a tibble.

(4) While the magnitude of a VIF value is important, the order VIF values are displayed in the table does not matter. I chose to order then in descending order by the sum of a predictors VIF in FM1 and FM2.

Note, I omitted the code for the LaTeX version of the table because I already explained such code in `Section 2.1.1`.

---

3 Best Subset Models

3.1 Selection Statistics

3.1.1 (Skip Section)

- The code in this sub-section contains was used to extract the models via complex for loops.
- No output is generated.

```r
1 BestEx1 <- regsubsets(x = as.matrix(select(trainDF1, FQHC:HardshipIndex)), y = trainDF1$LE,
2                       method = "exhaustive", nvmax = 20, nbest = 1)
3 S.BestEx1 <- summary(BestEx1)      # Saving the Summary
4 nmods <- nrow(S.BestEx1$which)     # Since I set nvmax = 20 and nbest = 1, nmods = 20
```

`regsubsets()` is a function used to quickly find subset models that are strong candidates for the best subset model. Please refer to the following link, providing documentation on the `regsubsets()` function from the `leaps` package:

[https://www.rdocumentation.org/packages/leaps/versions/3.1/topics/regsubsets](https://www.rdocumentation.org/package

(1) The data of all potential predictors must be inputed as a matrix. The outcome variable must be specified separately, and it must be formatted as a vector.

(2) I specified that I would like the function to perform an exhaustive search. Setting `nvmax = 20` limits the exhaustive search to models with up to 20 parameters. Given that there are 27 predictors total, this limitation did not affect my final results. Primarily, I added this limitation because it takes a long time for a CPU to perform exhaustive searches. I set `nbest = 1`, which is the default value, because I only want the top model of each size.

```r
1 ModList1 <- getRegSSMods(BestEx1, df = trainDF1, y = trainDF1$LE)
2 ModStats.DF1 <- UnpackModStats(ModList1, FM1)
```

`regsubsets()` outputs characteristics of each subset model, but it does not allow for direct extraction of the model itself. I created the functions `getRegSSMods()` and `UnpackModStats()`, which can be found in `MyFunctions.R`, to extract the linear model of each subset size and compute each model selection statistic.

3.2 Statistic vs. Model Size

3.2.1 Training Set#1

```r
# Melted/Gathered Data
PlotData1 <- select(BestSubsets1, -c(IVs)) %>%
  gather(Statistic, Value, -Size)     # A function similar to pivot_longer() from Secction 1.2

# Facet Plots illustrating how each model selection statistic changes as model size increases.
ggplot(data = PlotData1, aes(Size, Value)) +

  geom_line(aes(color = Statistic), size = rel(1.25), show.legend = F, alpha = 0.9) +

  geom_point(shape = 1, color = "black", stroke=rel(0.9), size = rel(2.25), show.legend = F, alpha = 0.9)  +

  facet_wrap(~Statistic, scales = "free") +

  labs(title = "Training Set #1") +

  scale_x_continuous(name = "Number of Parameters", limits = c(0, 21), breaks = seq(0, max(PlotData1$Size), 4)) +
  scale_y_continuous(n.breaks = 6) +

  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5, face = "bold", size = rel(1.5), family = "Courier"),
        strip.text = element_text(hjust = 0.5, face = "bold", size = rel(1.35), color = "White"),
        strip.background = element_rect(fill = "#005e7d"),
        axis.text = element_text(size = rel(1.25)),
        axis.title = element_text(size = rel(1.2), face = "bold"),
        axis.title.y.left = element_blank(),
        panel.grid.major = element_line(color = "gray78"),
        panel.grid.minor = element_line(color = "gray84"))
```

---

4 Regression Analysis

This section contains code similar to the code already explained in `Section 2.1`.

---

5 Results

5.1 Model Fitted & Predicted Values

5.1.1 Training Set – Life Expectancy vs. Fitted Values

5.1.1.1 Tabulated Results (Skip This Section)

```r
1  # Training Set 1
2  trainPredsM1 <- predict(object = M1, newdata = trainDF1)   # Same as M1$fitted.values
3  trainPredsM2 <- predict(object = M2, newdata = trainDF1)
4  trainPredsM3 <- predict(object = M3, newdata = trainDF1)
5
6  TrainFits1 <- trainDF1 %>%
7    mutate(M1 = trainPredsM1, M2 = trainPredsM2, M3 = trainPredsM3) %>%
8    select(Label = Name, `Actual LE` = LE, M1, M2, M3, FQHC:HardshipIndex)
```

(2-4) Since the `newdata = trainDF1` is the same data used to train M1, M2, and M3, the corresponding values are the same as each model's fitted values.

(6-8) After saving each model's fitted values into separate variables, I appended them to the main `trainDF1`.

Then, I did the same with H1, H2, H3, except these models used `trainDF2` (code omitted).

```r
1  # Combined Fits
2  FitsDF <- TrainFits1[1:2, 1:5] %>%
3    mutate(H1 = NA, H2 = NA, H3 = NA) %>%
4    rbind(cbind(TrainFits1[3:nrow(TrainFits1), 1:5], TrainFits2[3:5])) %>%
5    mutate(Index = 1:47) %>%
6    select(Index, everything())
```

I combined the fitted values of all 6 subset models into a new tibble called `FitsDF`.

(3-4) This part is a little tricky. Recall, `trainDF1` has two more rows than `trainDF2` because the former contains Burnside and Fuller Park. In order to combine all fits in one tibble, I just set the values for Burnside and Fuller Park as NA for models H1, H2, and H3.

(5) I appended index values, which are just the row numbers of the tibble. Index values were assigned for personal tracking of the data, but I did not end up using the index values in later code.

```r
1  FitLowBound <- as.numeric(apply(select(FitsDF, `Actual LE`:H3), 1, function(x) min(x, na.rm=T)))    # Extracting the Min
2  FitHighBound <- as.numeric(apply(select(FitsDF, `Actual LE`:H3), 1, function(x) max(x, na.rm=T)))   # Extracting the Max
3
4  FitsDF$Low <- floor(FitLowBound)
5  FitsDF$High <- apply(X = tibble(High1 = round(FitHighBound) + 0.5, High2 = round(FitHighBound + 0.5)),
6                       MARGIN = 1, FUN = function(x) min(x, na.rm=T))
```

Actual Life Expectancy values ranged between 65.89 and 82.94 ($range(y_i) = max(y_i) - min(y_i) \approx 17.05$), and all fitted values ranged between 83.02 and 65.61 ($range(\hat{y}_i) = max(\hat{y}_i) - min(\hat{y}_i) \approx 17.41$).

In comparison, the absolute value of all residuals did not exceeded 2 for any model ($|e_i| = |y_i - \hat{y}_i| < 2, \ \forall \ e_i$); specifically, the residuals had values ranging between -1.76 and 1.98 ($range(e_i) = 1.98 - (-1.76) \approx 3.74$). As a result, the fitted values for each model were very accurate.

In `Section 5.1.1.2`, I plotted the Actual Life Expectancy (LE) and each models fitted values. If I had set the limits of the y-axis to the overall min and max y-value of 65.61 and 83.02 for all CAs, it would have been difficult to observe differences between Actual LE and the fitted values.

Hence, I defined lower and upper bounds for each CA defined below:

(4) The lower bounds equaled the min y-value rounded down to the nearest whole number. The lower bounds were saved in a column called `Low`.

(5-6) The upper bounds equals the max y-value rounded up to the nearest 0.5 years. The upper bounds were saved in a column called `High`.

For example, Armour Square had y-values ranging between 80.33 and 81.01, so its plot limits were 80 and 81.5.

```r
# The resulting table is outputted by this code chunk.
cbind(FitsDF[,-1], round(select(trainDF1, FQHC:HardshipIndex), 2)) %>%
  mutate_if(.predicate = is.numeric, .funs = function(x) round(x, 2)) # Numeric columns are rounded to 2 decimal places.
```

5.1.1.2 Facet Grid

```r
1  FitPlotDF1 <- FitsDF %>%
2    pivot_longer(cols = `Actual LE`:H3, names_to = "Model", values_to = "Fits")
3  TrainCAs1 <- unique(FitPlotDF1$Label)[1:24]
4  TrainCAs2 <- unique(FitPlotDF1$Label)[25:47]
```

(3-4) There were 47 CAs total in the Training Set. I could not fit all CAs in one plot window, so I made two separate plots containing 24 and 23 CAs, respectively.

```r
1  # Plot 1
2  ggplot(data = filter(FitPlotDF1, Label %in% TrainCAs1), aes(x = Model, y = Fits)) +
3    geom_linerange(aes(ymin = Low, ymax = Fits, color = Model), show.legend = T, size = rel(2.5), alpha = 0.9) +
4    geom_blank(aes(y = High), show.legend = F) +
5    facet_wrap(vars(factor(Label, levels = TrainFits1$Label)), scales = "free_y", ncol = 5) +
6    labs(title = "Actual Life Expectancy vs. Each Model's Fitted Values (Part 1)", x = NULL, y = NULL) +
7    scale_color_manual(values = mycolors) +
8    scale_y_continuous(breaks = seq(64, 85, 1)) +
9    guides(color = guide_legend(nrow=1, label.position = "top",
10                               label.theme = element_text(family = "Courier", face = "bold", hjust = 0.5) )) +
11   theme_bw() +
12   theme(plot.title = element_text(hjust = 0.5, vjust = 2, face = "bold", size = rel(1.5), family = "Courier"),
13         strip.text = element_text(hjust = 0.5, face = "bold", size = rel(0.85), color = "White"),
14         strip.background = element_rect(fill = "#005e7d"),
15         panel.grid.major.y = element_line(color = "gray65"),
16         panel.grid.minor.y = element_line(color = "gray70"),
17         panel.grid.major.x = element_blank(),
18         panel.grid.minor.x = element_blank(),
19         axis.title = element_blank(),
20         axis.text.y = element_text(size = rel(1), color = "Black", family = "sans"),
21         axis.text.x = element_blank(),
22         axis.ticks.x = element_blank(),
23         legend.position = "top",
24         legend.background = element_rect(color = "Black", fill = "gray90"),
25         legend.key.width = unit(2.7, "cm"),
26         legend.key.height= unit(0.5, "cm"),
27         legend.key = element_rect(fill = "White", color = "Black"),
28         legend.title = element_blank())
```

(2) `Label %in% TrainCAs1`: I filtered the fitted training data to only include the first 24 CAs.

(4) `geom_blank`: I expanded the y-axis limits based on my upper bounds previously defined in the previous section (5.1.1.1)

(8) `scale_y_continuous()`: To keep plot scales consistent, I set all plots to have major ticks marks every whole value. Accordingly, minor tick marks show up every 0.5 years.

(9) `guides()`: The legend will include all components in 1 row and labels will be placed above their symbol.

5.2 Model Accuracy

```
1  PredsMain <- unique(c(names(coef(M1))[-1], names(coef(M2))[-1], names(coef(M3))[-1],
2                        names(coef(H1))[-1], names(coef(H2))[-1], names(coef(H3))[-1]))
3
4  OtherTrainDF1 <- select(trainDF1, -c(all_of(PredsMain), GEOID)) %>%
5    mutate(Label = Name, Name = NULL)
6  OtherTrainDF2 <- select(trainDF2, -c(all_of(PredsMain), GEOID)) %>%
7    mutate(Label = Name, Name = NULL)
8
9  s.OM1 <- summary(OM1 <- lm(LE ~ ., data = OtherTrainDF1[-1]))
10 s.OM2 <- summary(OM2 <- lm(LE ~ ., data = OtherTrainDF2[-1]))
```

To compare subset models selected via statistical methods, I created Other Model 1 (OM1) and Other Model 2 (OM2) containing only the predictors that weren't selected by any other previous subset model.

(1-2) `PredsMain` is a vector containing every unique variable included in any previous subset model.

(4-7) Similar to the previously described `trainDF1` and `trainDF2`, `OtherTrainDF1` includes Burnside and Fuller Park while `OtherTrainDF2` does not.

5.2.1 Model Accuracy Results

```
1  Models1 <- list(FM1, M1, M2, M3, OM1)
2  Labels1 <- c("$\\boldsymbol{FM_1}$", "$\\boldsymbol{M_1}$", "$\\boldsymbol{M_2}$", "$\\boldsymbol{M_3}$",
3               "$\\boldsymbol{OM_1}$")
4  Models2 <- list(FM2, H1, H2, H3, OM2)
5  Labels2 <- c("$\\boldsymbol{FM_2}$", "$\\boldsymbol{H_1}$", "$\\boldsymbol{H_2}$", "$\\boldsymbol{H_3}$",
6               "$\\boldsymbol{OM_2}$")
7  AllLabels <- c("$\\boldsymbol{FM_1}$", "$\\boldsymbol{FM_2}$",
8                 "$\\boldsymbol{M_1}$", "$\\boldsymbol{M_2}$", "$\\boldsymbol{M_3}$",
9                 "$\\boldsymbol{H_1}$", "$\\boldsymbol{H_2}$", "$\\boldsymbol{H_3}$",
10                "$\\boldsymbol{OM_1}$", "$\\boldsymbol{OM_2}$")
11
12 FinalAcc.DF <- rbind(ModAccLaTeX2(ListOfModels = Models1, VectorOfLabels = Labels1,
13                                   TrainSet = trainDF1, TestSet = testDF1),
14                      ModAccLaTeX2(ListOfModels = Models2, VectorOfLabels = Labels2,
15                                   TrainSet = trainDF2, TestSet = testDF1))  %>%
16   arrange(factor(Model, levels = AllLabels))
```

(1) `Models1` contains all the models trained with `trainDF1`, which includes Burnside and Fuller Park.

(4) `Models2` contains all the models trained with `trainDF2`, which excludes Burnside and Fuller Park.

(12-15) The function `ModAccLaTeX()`, defined in the script file `MyFunctions.R`, computes the accuracy statistics (MAE, MSE, RMSE, MAPE) for a list of models; statistics are computed for both the training set and test set.

5.2.2 Accuracy Plots

```r
1  Labs1 <- c("FM1", "M1", "M2", "M3", "OM1")
2  Labs2 <- c("FM2", "H1", "H2", "H3", "OM2")
3  AllLabs <- c("FM1", "FM2", "M1", "M2", "M3", "H1", "H2", "H3", "OM1", "OM2")
4
5  AccDF1 <- rbind(ModAcc(ListOfModels = Models1, VectorOfLabels = Labs1, TrainSet = trainDF1, TestSet = testDF1),
6                  ModAcc(ListOfModels = Models2, VectorOfLabels = Labs2, TrainSet = trainDF2, TestSet = testDF1)) %>%
7    arrange(factor(Model, levels = AllLabs))
```

This chunk performs generates a similar output as the previous chunk in `Section 5.2.1` except it does not apply LaTeX formatting.

(5-6) The function `ModAcc()` is defined in the script file `MyFunctions.R`. As mentioned, `ModAcc()` and `ModAccLaTeX()` are similar functions except `ModAcc()` does not apply LaTeX formatting. By not applying LaTeX formatting, numeric values are interpreted as numerically; in contrast, numeric values were interpreted as strings in the previous section. In order to generate useful plots, it is necessary that values are interpreted numerically.

```r
1  TrainAccDF <- select(AccDF1, c(1,2, grep(pattern = "Training", names(AccDF1)))) %>%
2    pivot_longer(cols = 3:6, names_to = "Statistic", values_to = "Value") %>%
3    mutate(Statistic = gsub(x = Statistic, pattern = "Training ", replacement = "")) %>%
4    mutate(Model = factor(Model, levels = AllLabs),
5           Statistic = factor(Statistic, levels = c("MAE", "MSE", "RMSE", "MAPE")))
```

The data is prepared for plotting.

(1) `grep` returns all indices that contain the string "Training"

(3) `gsub` removes the word "Training" from the names of each statistic. For instance, Training MSE becomes just MSE.

(4-5) The Model and Statistic variables/columns are converted to factor variables.

5.2.2.1 Training Set

```r
1  ggplot(data = TrainAccDF) +
2    geom_col(aes(x = Statistic, y = Value, fill = Model),
3             position = "dodge", color ="black") +
4    labs(title = "Performance on the Training Set", x = NULL, y = NULL) +
5    scale_fill_manual(values = mycolors) +
6    guides(fill = guide_legend(nrow = 1, label.position = "top",
7                               label.theme = element_text(family = "Courier", face = "bold", hjust = 0.5) )) +
8    theme_bw() +
9    theme(plot.title = element_text(hjust = 0.5, face = "bold", family = "Courier", size = rel(2.25)),
10         panel.grid.major = element_line(color = "gray78"),
11         panel.grid.minor = element_line(color = "gray84"),
12         axis.title.y.left=element_text(size=rel(1.75), face="bold", family="sans", margin=margin(t=0, r=8, b=0, l=3)),
13         axis.title.x.bottom=element_text(size=rel(1.75), face="bold", family="sans", margin=margin(t=8, r=0, b=3, l=0)),
14         axis.text = element_text(size = rel(1.5), face = "bold", family = "Courier"),
15         legend.position = "bottom",
16         legend.background = element_rect(color = "Black", fill = "gray90"),
17         legend.key.width = unit(1, "cm"),
18         legend.text = element_text(size=rel(1.75), face="bold", family="Courier", margin=margin(t=0, r=8, b=0, l=3)),
19         legend.key = element_rect(fill = "White", color = "Black"),
20         legend.title = element_blank())
```

(2) `fill = Model`: Each column represents a model.

(3) `position = "dodge"`: For each x-value, column for different modes are placed next to each other rather than on top of each other.