Analysis of the Dynamics of the U.S. Housing Price Index

Refactored Code and Workflow of STAT 1261 Final Project

Rohan Krishnan

2023-12-11

Introduction

As of November 2023, the U.S. housing market is valued at 47 trillion USD (Rosen, 2023). With an average year-over-year growth rate of 5.5%, the U.S. housing market is set to remain a key factor in calculating the overall welfare of the nation's economy (CEIC Data, 2023). On a consumer level, home ownership remains tied to the "American Dream", as working class citizens across the country strive towards purchasing, selling, or renting homes. Investors and economists rely on the Housing Price Index (referred to as the HPI from this point onwards) as a key metric for assessing investment feasibility and for determining national economic forecasts.

The HPI is a comprehensive macroeconomic measure that monitors the price fluctuations of **single-family** homes nationwide. It also serves as an analytical tool for approximating the changes in the rates of mortgage defaults, prepayments, and housing affordability (Liberto, 2023). The Federal Housing Finance Agency compiles this data by reviewing single family housing mortgages purchased or securitized by Fannie Mac or Freddie Mac. My primary goal for this statistical analysis project was to find an economic data set that I could explore to better understand the U.S. economy.

Research Question

The primary goal of this project is to examine how the HPI is affected by other macroeconomic factors. By examining how other commonly tracked measures affect the HPI, I aim to better understand the dynamics between macroeconomic measures and will create a useful basis for more focused research in the future. An improved understanding of the relationships may also prove useful in better understanding real estate and broader housing market dynamics.

In summary, this project aims to establish quantitative measures of the relationships between common macroeconomic measures and the HPI to better inform the direction of future research. Below is the main research question I aimed to answer for as part of the final project requirements:

- What variables are most useful in predicting the HPI?

    1. Which machine learning model best predicts the HPI given new data?

Statement of Purpose

In terms of a research-based objective, I created this project with the goal to broaden my knowledge about the economy, consumer behavior, and investment sectors associated within the U.S. housing market. Most importantly, however, this project aims to apply a wide range of machine learning techniques and provide statistical justifications and explanations for their use.

Methodology

This section will discuss my data sourcing, ingestion, and cleaning process.

Data Collection

To recreate the data set I used for my original STAT 1261 final project, I used two data sets of U.S. HPI influences from Kaggle. By comparing the values of variables between the two data sets and looking at the documentation of each, I chose only the variables whose values I could verify as accurate.

Data Set(s)

To create a unified data frame in R, I left-joined each individual csv file on `house_data.csv`. I chose `house_data.csv` as my base file because it only went back until 2003, unlike the other files that had data extending back to the 1970s. After left-joining the data into one data frame, I converted the `DATE` column to a "Date" type and used `dplyr` to filter the data into a data set that only contained measures taken after January 1st, 2003.

```
raw_data <- load_raw_data()
basic_describe_data(data = raw_data, data_name = "raw")
```

```
## The raw data set has 22 columns, 921 rows, spans 1947-01-01 to 2023-09-01,
## and has 12250 NA values.
```

```
preliminary_data <- load_preliminary_data()
basic_describe_data(data = preliminary_data, data_name = "preliminary")
```

```
## The preliminary data set has 22 columns, 249 rows, spans 2003-01-01 to 2023-09-01,
## and has 519 NA values.
```

As is seen above, the original raw data had 921 observations but over 12,000 missing values due to the mismatch in reporting time frames. After filtering for measures after 2003, the data set is cut down to only 249 observations, a 72% reduction in usable observations, and only 519 missing values. Unfortunately, because the measures in housing_data.csv abruptly stopped in 2003, it was not feasible to perform imputation to maintain data volume.

Variables

As the main focus of this project is to explore the U.S. HPI, we have the following features and target variable(s):

Variable Name	Definition	Target or Feature?
CPIAUSCL	U.S. Consumer Price Index	Feature
FEDFUNDS	U.S. Federal Funds rate	Feature
GDP.x	U.S. GDP reported quarterly	Feature
CSUSHPISA	**U.S. Housing Price Index**	**Target**
building_permits	Number of new building permits in the U.S.	Feature

Variable Name	Definition	Target or Feature?
const__price__index	U.S. Construction Price Index	Feature
delinquency__rate	U.S. percentage of loans overdue by more than 30 days	Feature
GDP.y	(Unreliable) U.S. GDP reported monthly	NA
house__for__sale__or__sold	(Unreliable) Number of houses for sale or sold in the U.S. (?)	NA.
housing__subsidies	Value of U.S. housing subsidies	Feature
income	Median U.S. household income	Feature
interest__rate	(Unreliable) Value of U.S. interest rates	NA
mortgage__rate	Value of U.S. home mortgage rates	Feature
construction__unit	Number of new construction units in the U.S.	Feature
total__houses	Total number of houses in the U.S.	Feature
total__const__spending	Total U.S. construction spending	Feature
unemployment__rate	(Unreliable) U.S. unemployment rate	NA
urban__population	U.S. urban population (millions)	Feature
home__price__index	(Unreliable) Measure of U.S. Housing Price Index	NA
MORTGAGE30US	(Unreliable) The average interest rates on mortgage loans in the United States	NA
UNRATE	U.S. unemployment rate	Feature

From the above table, it's clear that there are several duplicate variables whose values do not match. The variables marked "unreliable" are variables whose values I could verify using FRED data. As I move through the data cleaning process, I will remove columns which have unreliable information or have too large a number of missing values.

Below is a summary table of the preliminary data. This data is what is output after joining all csv files and filtering for values after 2003.

```
#Prelim data summary + NA by columns
stargazer(preliminary_data, summary = TRUE, type = "latex",
          title = "Preliminary Data Summary",
          header = FALSE, no.space = TRUE)
```

Table 2: Preliminary Data Summary

Statistic	N	Mean	St. Dev.	Min	Max
CPIAUCSL	249	232.681	30.280	182.600	307.481
FEDFUNDS	249	1.432	1.692	0.050	5.330
GDP.x	83	17,617.740	4,190.519	11,174.130	27,623.540
CSUSHPISA	247	184.069	45.327	128.461	306.720
building_permits	240	1,309.350	479.881	513	2,263
const_price_index	240	212.851	44.567	144.400	353.015
delinquency_rate	240	4.877	3.305	1.410	11.480
GDP.y	240	18,095.160	2,002.294	14,614.140	21,989.980
house_for_sale_or_sold	240	55.550	25.384	20	127
housing_subsidies	240	34.677	6.006	25.930	48.021
income	240	13,493.480	1,837.485	10,674.000	20,422.600
interest_rate	240	1.302	1.579	0.050	5.260
mortgage_rate	240	4.683	1.111	2.684	6.900
construction_unit	240	1,201.717	423.858	520	2,245
total_houses	240	121,344.400	6,113.869	111,278	131,202
total_const_spending	240	0.325	1.950	−5.900	5.000
unemployment_rate	240	6.012	2.034	3.500	14.700
urban_population	240	81.261	1.055	79.583	83.084
home_price_index	240	180.658	41.256	128.461	304.755
MORTGAGE30US	33	4.869	1.194	2.880	6.790
UNRATE	249	5.924	2.048	3.400	14.700

Some important points to note include the vastly different scaling between variables. For example, `GDP.x` seems to be measured in millions while `FEDFUNDS` is a measure of interest rate with a maximum value of 5.330. Also, `GDP.x` has noticeable fewer observations than the rest of the data, at only 83. This is because it is pulled directly from the FRED database, which only reports quarterly values. Since the data is monthly, there are 8 missing values for `GDP.x` per year. `MORTGAGE30US` seems to be even worse, with only 33 known values.

Below is a table showing the number of missing values per column.

```
prelim_na_vals <- extract_na_per_column(preliminary_data)

prelim_na_vals %>% rename(Feature = variable,
                          "NA Values" = na_values) %>%
  stargazer(type = "latex", summary = FALSE, flip = FALSE,
          title = "Preliminary Data NA Values by Column",
          header = FALSE, no.space = TRUE)
```

Table 3: Preliminary Data NA Values by Column

	Feature	NA Values
1	DATE	0
2	CPIAUCSL	0
3	FEDFUNDS	0
4	GDP.x	166
5	CSUSHPISA	2
6	building_permits	9
7	const_price_index	9
8	delinquency_rate	9
9	GDP.y	9
10	house_for_sale_or_sold	9
11	housing_subsidies	9
12	income	9
13	interest_rate	9
14	mortgage_rate	9
15	construction_unit	9
16	total_houses	9
17	total_const_spending	9
18	unemployment_rate	9
19	urban_population	9
20	home_price_index	9
21	MORTGAGE30US	216
22	UNRATE	0

The two clear outliers in terms of missing values are `GDP.x` and `MORTGAGE30US.` I will explore methods of dealing with these two variables in the code below.

First, I will handle the missing data in the `GDP.x` column. As noted above, the St. Louis Federal Reserve (FRED) only reports real GDP on a quarterly basis, meaning the GDP column had missing values for 8 out of the 12 months for each year. Because there were accurate measures of GDP every 4 months, I decided to use seasonal decomposition to impute its intermediate values. In this method, the time series is decomposed into its trend, and seasonal components; then, the intermediate values are imputed using only the trend; finally, the seasonality is added back into the data. This allows for imputed values that smoothly follow the trend of the time series while also adhering to any seasonality present. I also made an indicator column to keep track of which values came from the data and which were imputed.

```
#Create imputed GDP column
im_preliminary_data <- preliminary_data
im_preliminary_data$imputed_GDP <- impute_GDP(preliminary_data = preliminary_data)

im_preliminary_data <- create_imputed_column(
  imputed_preliminary_data = im_preliminary_data,
  "GDP.x", "imputed_GDP")
```

After imputing the GDP values, I worked on removing low-quality and duplicate columns. Above is the code I used, highlighting which columns I chose to remove. Because I could not verify the values in `MORTGAGE30US` and because it had so many missing values, I decided to remove it completely. I also removed duplicate columns for the federal funds rate, HPI, unemployment rate, and the `house_for_sale_or_sold` column, whose validity I could not properly confirm. After imputation and trimming, I was left with a data set of 18 columns compared to the original's 24. This data set was then saved for further use in the data visualization section.

```r
## This is the data saved for visualization
#Remove low quality columns
#Duplicate post_2003 to keep as intermediate data
trimmed_preliminary_data <- im_preliminary_data

#List of columns to delete
columns_to_delete <- c("MORTGAGE30US", "GDP.y", "interest_rate",
                       "home_price_index", "unemployment_rate",
                       "house_for_sale_or_sold")

#Loop through and set to NULL
for (i in columns_to_delete){
  trimmed_preliminary_data[,i] <- NULL
}

write(paste("Imputed data had", ncol(im_preliminary_data), "columns.",
      "\nTrimmed data has", ncol(trimmed_preliminary_data),
      "columns"), stdout())
```

```
## Imputed data had 24 columns.
## Trimmed data has 18 columns
```

At this point, I worked on creating different data sets for different parts of the modeling process. First, I removed GDP.x and the imputed indicator column to create a data set with only the variables that would be used for modeling.

```r
## This is the data saved for modeling.
# Remove GDP.x and imputed columns to create modeling data set.
modeling_preliminary_data <- trimmed_preliminary_data

#Remove GDP.x from post2003 and only leave imputeGDP, remove imputeYN
modeling_preliminary_data$GDP.x <- NULL
modeling_preliminary_data$imputed <- NULL

write(paste("The visualization data had", ncol(trimmed_preliminary_data),
            "columns.", "\nThe modeling data has",
            ncol(modeling_preliminary_data), "columns"), stdout())
```

```
## The visualization data had 18 columns.
## The modeling data has 16 columns
```

I then renamed the data for ease of reading.

```r
#Rename variables
renamed_preliminary_data <- modeling_preliminary_data

new_names = c("date", "urban_cpi", "fed_funds_rt", "hpi", "build_permits",
              "const_price_idx", "delinq_rt", "house_subsidies", "income",
              "mortgage_rt", "const_unit", "tot_house", "tot_const_spend",
              "urban_pop", "unem_rt", "imputed_gdp")

colnames(renamed_preliminary_data) <- new_names
```

After all of the above preprocessing, I was left with the following missing values:

```
# Extract NA values & display in stargazer table
final_prelim_na_vals <- extract_na_per_column(renamed_preliminary_data)

final_prelim_na_vals %>% rename(Feature = variable,
                                "NA Values" = na_values) %>%
  stargazer(type = "latex", summary = FALSE, flip = FALSE,
          title = "NA Values by Column After Preprocessing",
          header = FALSE, no.space = TRUE)
```

Table 4: NA Values by Column After Preprocessing

	Feature	NA Values
1	date	0
2	urban_cpi	0
3	fed_funds_rt	0
4	hpi	2
5	build_permits	9
6	const_price_idx	9
7	delinq_rt	9
8	house_subsidies	9
9	income	9
10	mortgage_rt	9
11	const_unit	9
12	tot_house	9
13	tot_const_spend	9
14	urban_pop	9
15	unem_rt	0
16	imputed_gdp	0

After renaming the remaining variables for ease of use and examining the remaining missing values, I decided that the were few enough missing values that it was reasonable to simply drop the problem rows. I then extracted the month and year from the `DATE` column to use in my analyses before saving the data frame to `./data/modelling/` as `modelling.csv`.

```
# Remove NAs, extract month and year
cleaned_data <- final_preprocessing(intermediate_data = renamed_preliminary_data)
```

Modeling and Analysis Plan

Description of Analysis

Analysis Plan

Results

Exploratory Data Analysis

Descriptive Statistics

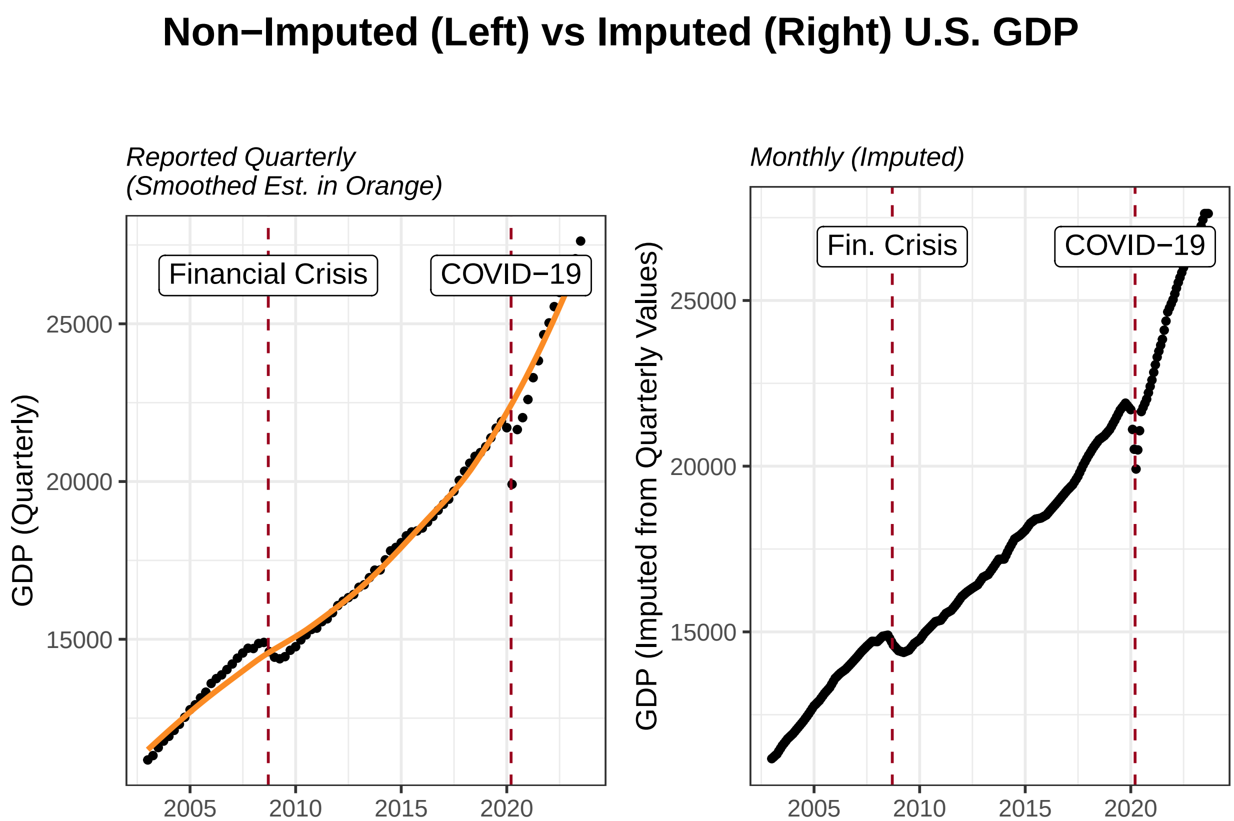
```r
#Cleaned data summary stats
stargazer(cleaned_data, type = "latex", summary = TRUE,
          flip = FALSE,  title = "Final Cleaned Data Summary",
          header = FALSE, no.space = TRUE)
```

Table 5: Final Cleaned Data Summary

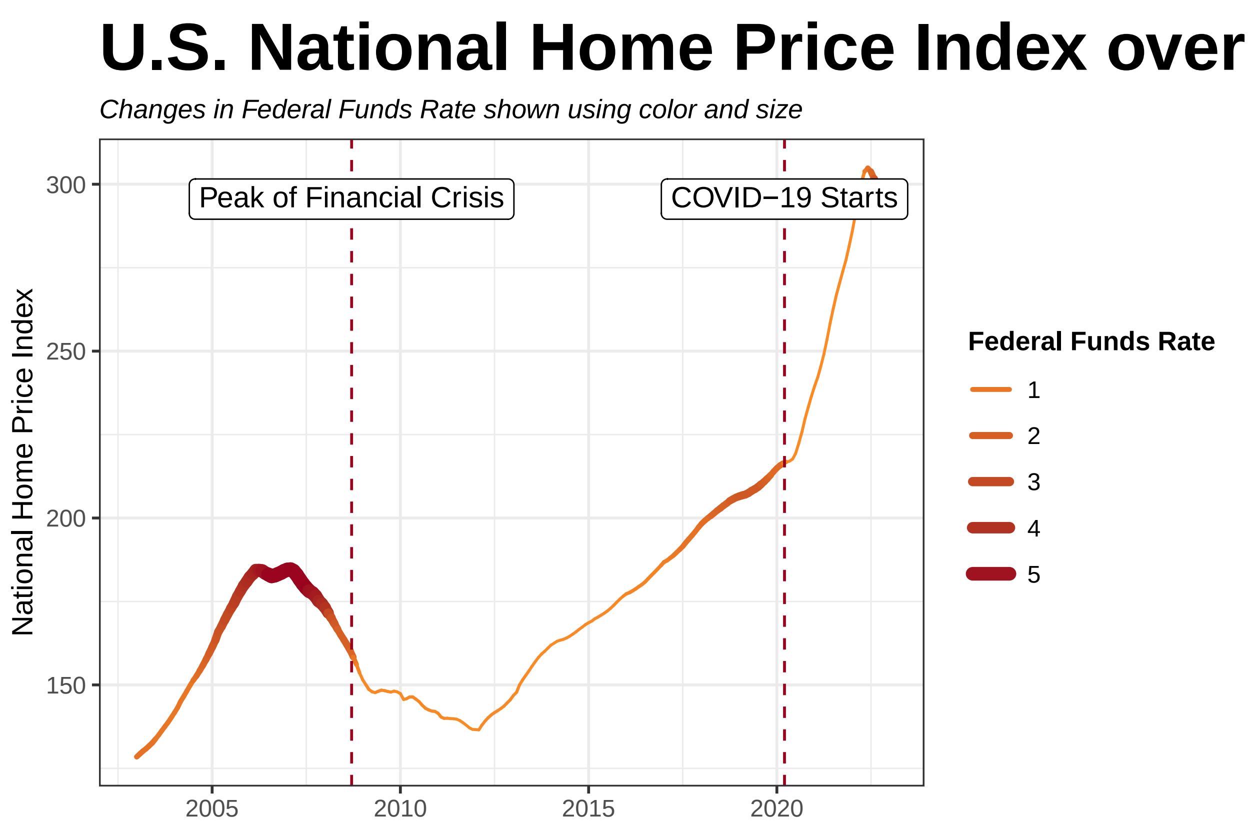
Statistic	N	Mean	St. Dev.	Min	Max
urban_cpi	240	230.023	27.475	182.600	298.990
fed_funds_rt	240	1.302	1.579	0.050	5.260
hpi	240	180.659	41.258	128.461	304.832
build_permits	240	1,309.350	479.881	513	2,263
const_price_idx	240	212.851	44.567	144.400	353.015
delinq_rt	240	4.877	3.305	1.410	11.480
house_subsidies	240	34.677	6.006	25.930	48.021
income	240	13,493.480	1,837.485	10,674.000	20,422.600
mortgage_rt	240	4.683	1.111	2.684	6.900
const_unit	240	1,201.717	423.858	520	2,245
tot_house	240	121,344.400	6,113.869	111,278	131,202
tot_const_spend	240	0.325	1.950	−5.900	5.000
urban_pop	240	81.261	1.055	79.583	83.084
unem_rt	240	6.012	2.034	3.500	14.700
imputed_gdp	240	17,324.820	3,835.256	11,174.130	26,678.540
year	240	2,012.500	5.778	2,003	2,022
month	240	6.500	3.459	1	12

Data Visualization

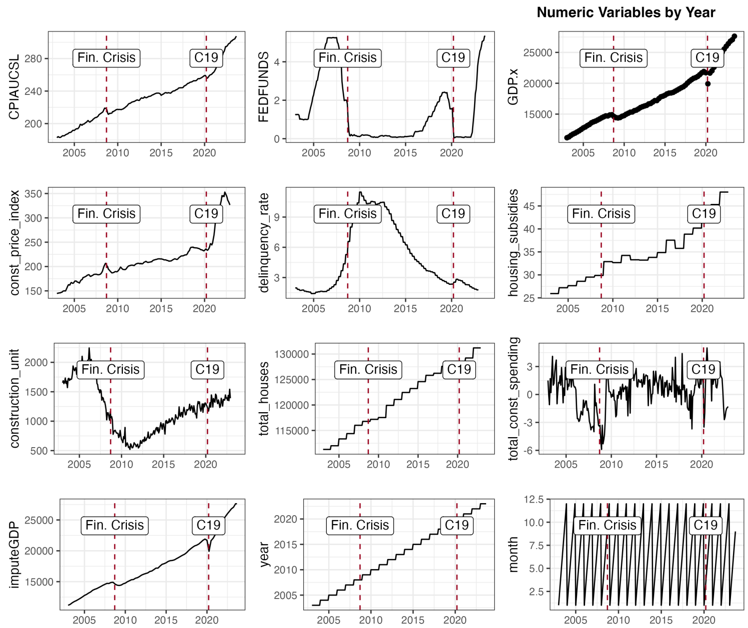
```r
#Pre vs post imputation GDP

v_data <- load_viz_data()
display <- plot_GDP_side_by_side(viz_data = v_data)
```



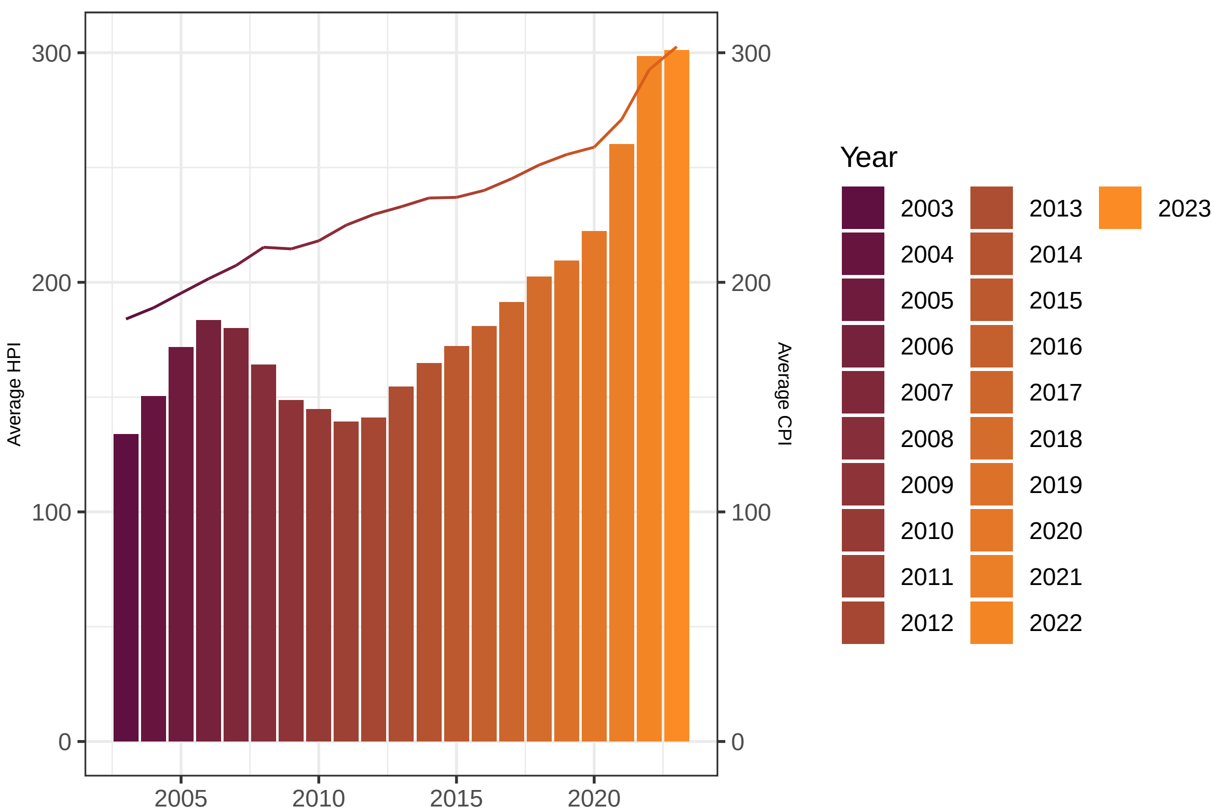

```
plot_hpi_values(viz_data = cleaned_data, date_col = "date", hpi_col = "hpi",
                ff_rt_col = "fed_funds_rt")
```



```r
## Line plot matrix
plot_line_matrix(v_data)
```

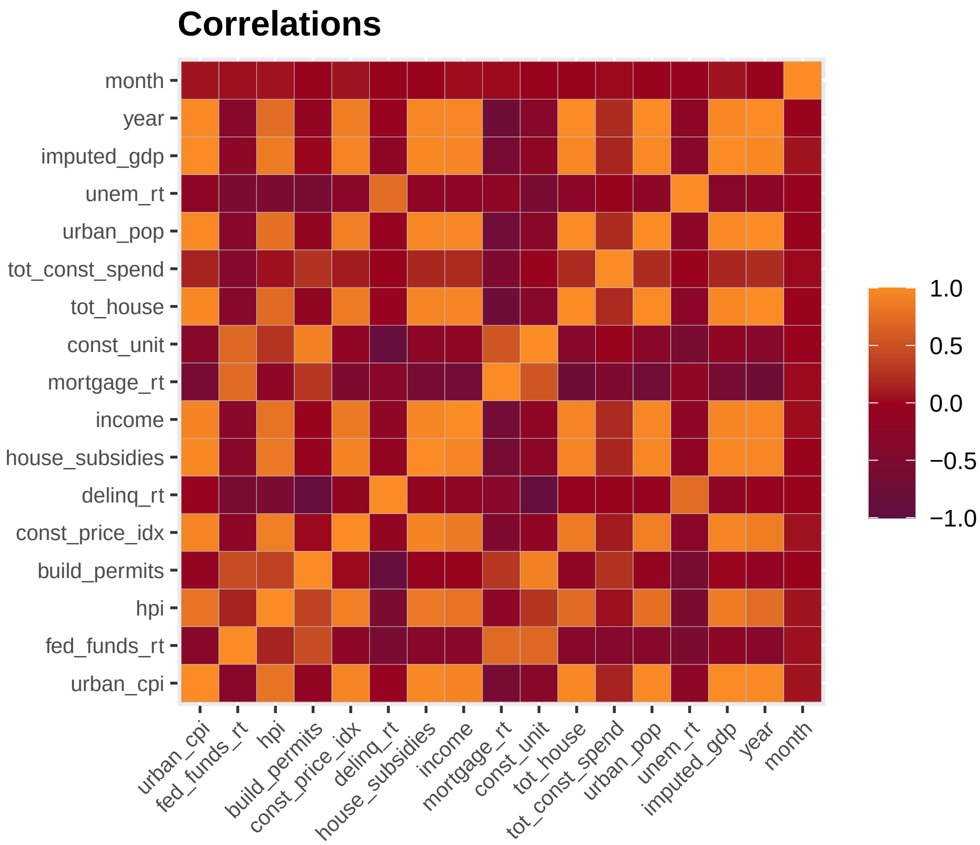



```r
## Plot hpi and cpi over time
plot_hpi_cpi(v_data)
```



Relationships

```r
plot_correlations(cleaned_data)
```

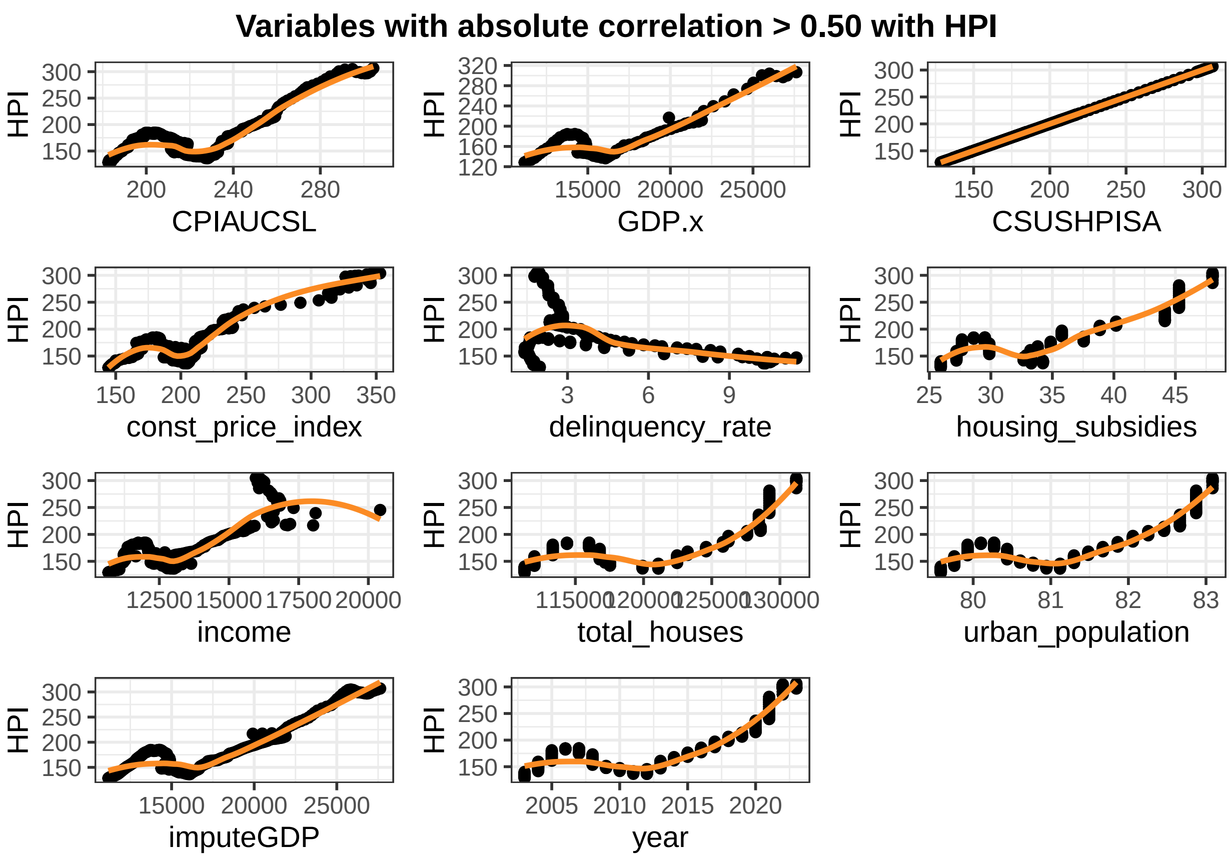



```r
imp_vars <- filter_imp_vars(viz_data = v_data)

imp_vars %>% rename(Name = name,
                    Correlation = hpi) %>%
  stargazer(type = "latex", summary = FALSE, flip = FALSE,
          title = "Variables with an Absolute Correlation >0.50 with HPI",
          header = FALSE, no.space = TRUE, rownames = FALSE)
```

```r
plot_imp_line_matrix(viz_data = v_data,
                     imp_vars = imp_vars)
```

Table 6: Variables with an Absolute Correlation >0.50 with HPI

Name	Correlation
CPIAUCSL	0.799
GDP.x	0.857
CSUSHPISA	1
const_price_index	0.887
delinquency_rate	-0.531
housing_subsidies	0.828
income	0.788
total_houses	0.720
urban_population	0.763
imputeGDP	0.857
year	0.740



Modeling

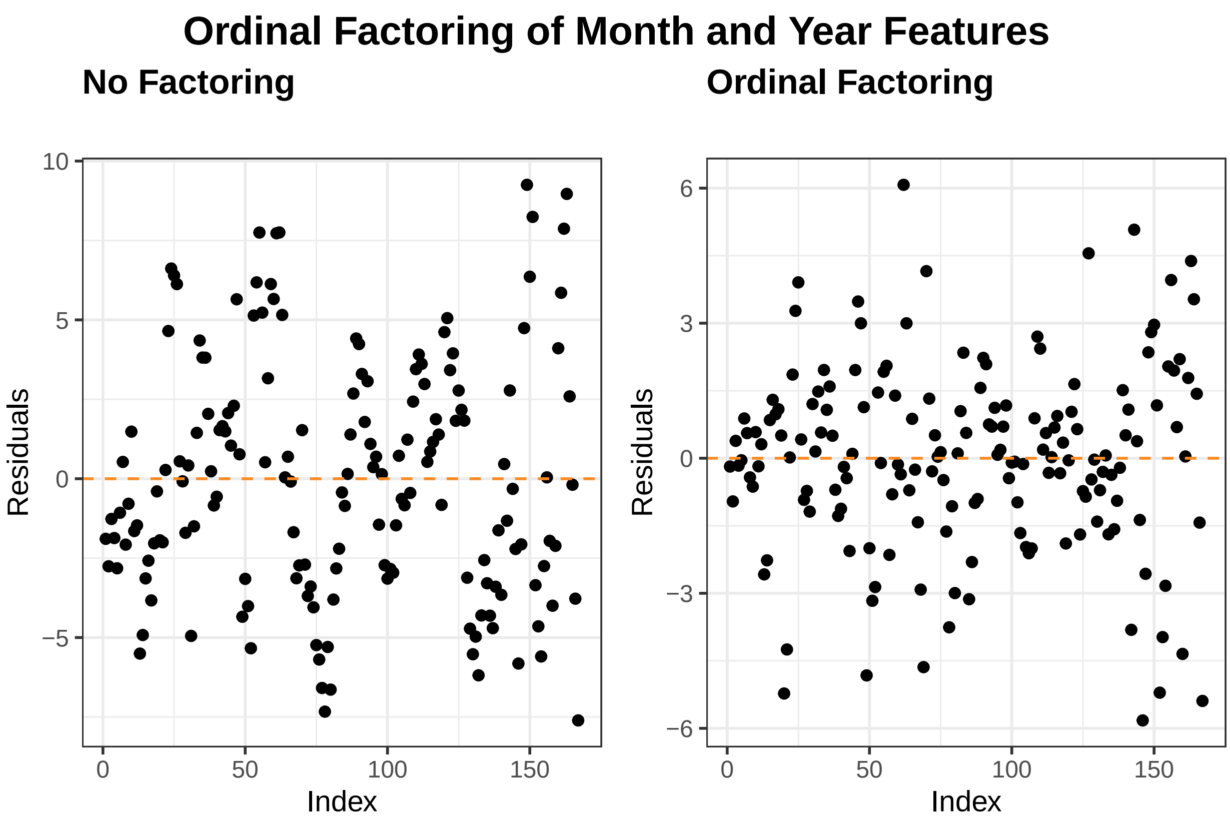
```
set.seed(100)
model_data <- read.csv("../data/modelling/modelling.csv")
train_test_split(model_data, 0.70, 0.30)
```

Multiple Linear Regression

```
model_data_lr <- model_data %>% select(-date)

model_data_lr$factor_year <- factor(model_data_lr$year, order = TRUE,
                                    levels = c(unique(model_data_lr$year)))
model_data_lr$factor_month <- factor(model_data_lr$month, order = TRUE,
                                     levels = c(unique(model_data_lr$month)))
```

```
train_test_split(model_data_lr, propTrain = 0.70, propTest = 0.30)
train_lr <- train
test_lr <- test
```

```
model_no_factor <- lm(clcsHPI ~ . -factor_month - factor_year , train_lr)
model_factor <- lm(clcsHPI ~ . -year -month, train_lr)
preds_nf <- predict(model_no_factor, test_lr)
preds_f <- predict(model_factor, test_lr)

plot_residuals_comparison(model_nf = model_no_factor, model_f = model_factor)
```



Factoring (Right) seems to look more random than no factoring (Left)

```
errors <- initialize_errors(nrows = 4, row_names = c("MSE", "RMSE", "MAE", "MAPE"))
errors <- append_errors(errors_df = errors, preds = preds_nf,
                        model_name = "lr_nf", test = test_lr,
                        target = "clcsHPI")
```

```r
errors <- append_errors(errors_df = errors, preds = preds_f,
                        model_name = "lr_f", test = test_lr,
                        target = "clcsHPI")
```

Bootstrap Regression

```r
mods <- initialize_bootstrap(df = model_data %>% select(-date),
                             n_samples = 100, target = "clcsHPI")

bootstrap_models <- generate_bootstrap_models(mods)

bootstrap_results <- calc_95_perc_int(df = model_data,
                                      exclude_cols = c("date", "clcsHPI"),
                                      mods_boot = bootstrap_models)
```

```r
bootstrap_results %>% rename(Feature = name,
                             "Lower Bound" = lower_bound,
                             "Upper Bound" = upper_bound) %>%
  mutate(`Lower Bound` = round(`Lower Bound`, 5),
         `Upper Bound` = round(`Upper Bound`, 5)) %>%
  stargazer(type = "latex", header = FALSE,
            title = "Bootstrap Regression 95 Percent Confidence Intervals",
            summary = FALSE, flip = FALSE, digits = 5)
```

Table 7: Bootstrap Regression 95 Percent Confidence Intervals

	Feature	Lower Bound	Upper Bound
1	urbanCPI	-0.21437	0.63868
2	fedFunds	2.29996	4.72484
3	buildPermits	-0.00488	0.00659
4	constructionPI	0.32249	0.58713
5	delRate	-6.29629	-3.49725
6	houseSub	1.63695	3.07629
7	income	-0.01239	-0.00054
8	mortRate	-1.27837	3.01867
9	constructionUn	0.00645	0.02046
10	totalHouse	-0.00932	-0.00564
11	totalConstSpend	-0.41983	0.49334
12	urbanPop	-262.66302	-130.90204
13	unemploymentRate	1.00138	3.70763
14	imputeGDP	0.00387	0.01169
15	year	28.38889	51.27943
16	month	-0.57833	0.12285

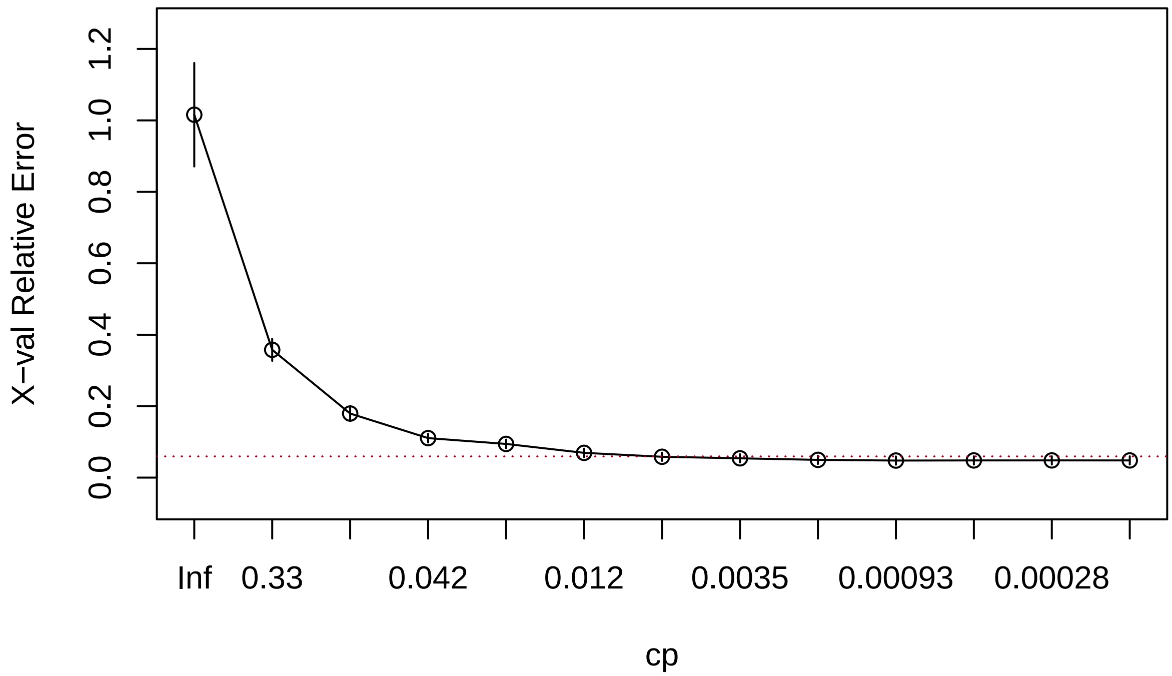
Linear Model Selection

```r
preproc_model_data <- model_df_preprocess(model_data)

train_test_split(preproc_model_data, 0.70, 0.30)
train_lms <- train
test_lms <- test

#Get x_train, y_train, x_test, y_test
x_train <- create_x_matrix(target = "clcsHPI", train_or_test = train_lms)
y_train <- create_y_vector(target = "clcsHPI", train_or_test_df = train_lms)

x_test <- create_x_matrix(target = "clcsHPI", test_lms)
y_test <- create_y_vector(target = "clcsHPI", train_or_test_df = test_lms)


#Get best lasso lambda
lasso_optimal_lambda <- get_best_lambda(x_train = x_train,
                                        y_train = y_train, alpha = 1)

preds_lasso <- predict(glmnet(x_train, y_train, alpha = 1),
                       s = lasso_optimal_lambda, newx = x_test)

errors <- append_errors(errors_df = errors, preds = preds_lasso,
                        model_name = "lasso", test = test_lms,
                        target = "clcsHPI")
```

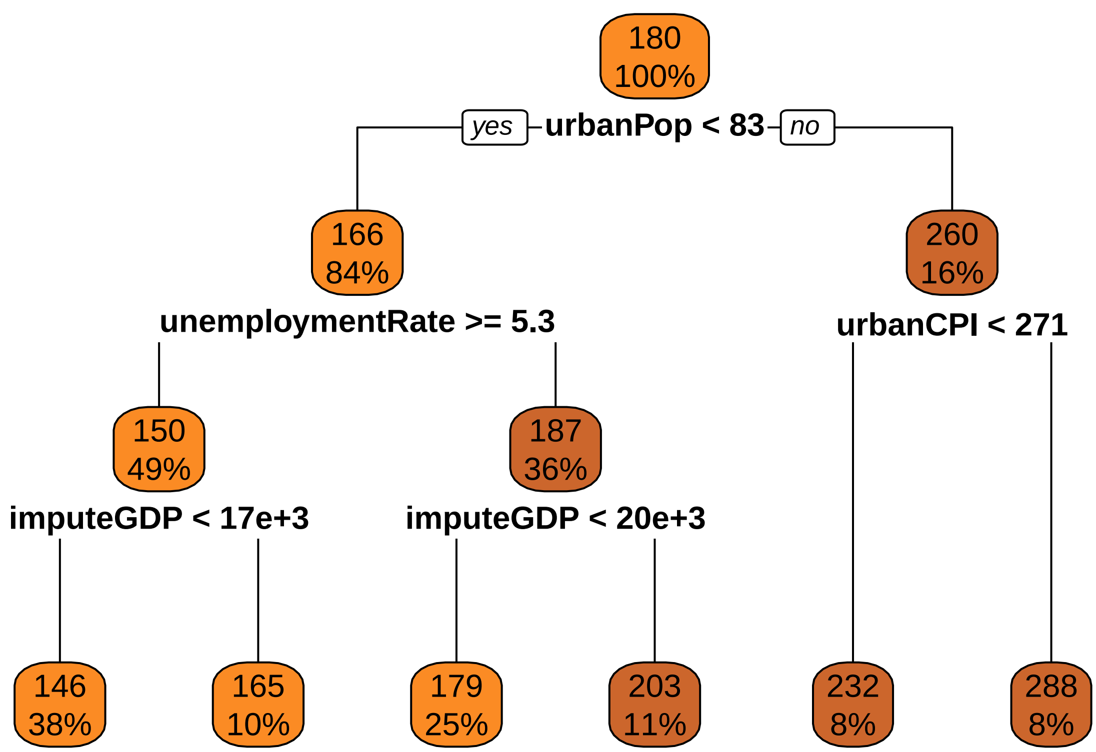
```r
#Get best ridge lambda
ridge_optimal_lambda <- get_best_lambda(x_train = x_train,
                                        y_train = y_train, alpha = 0)

preds_ridge <- predict(glmnet(x_train, y_train, alpha = 0),
                       s = ridge_optimal_lambda, newx = x_test)

errors <- append_errors(errors_df = errors, preds = preds_ridge,
                        model_name = "ridge", test = test_lms,
                        target = "clcsHPI")
```

Regression Tree

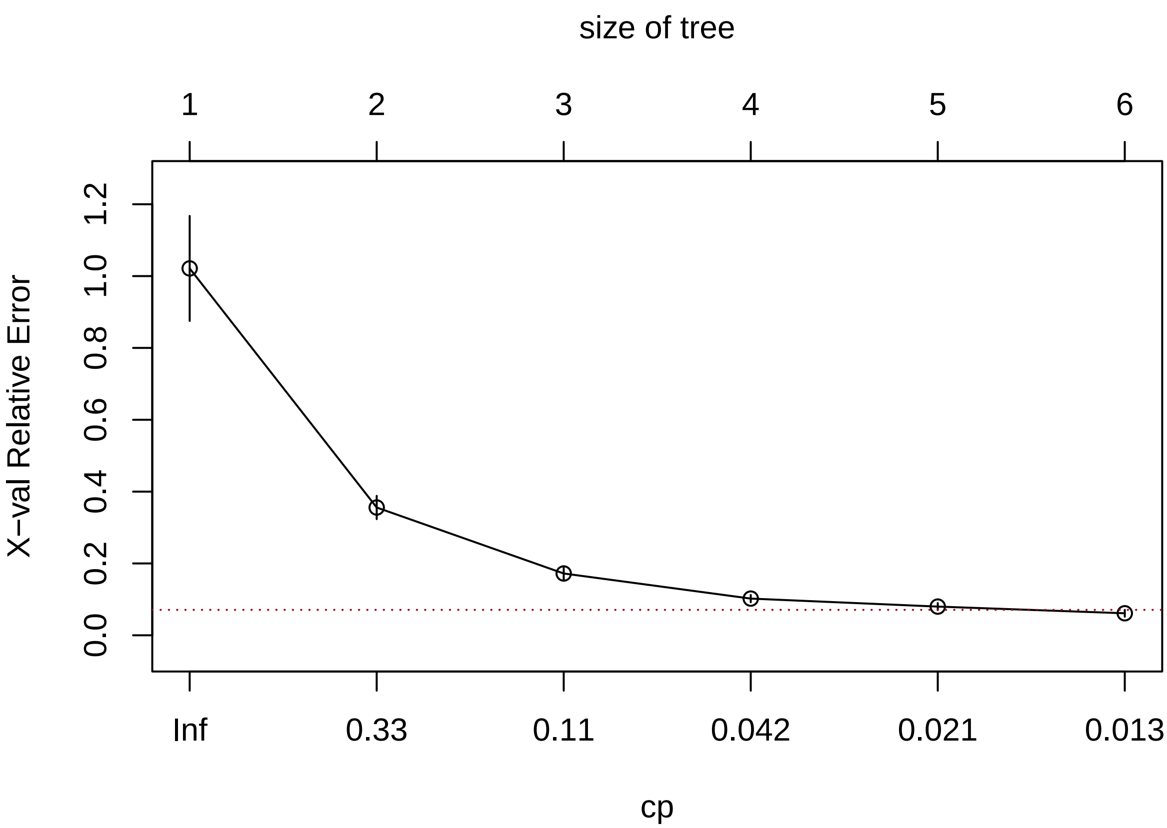
```r
train_dt <- train
test_dt <- test

#full tree
set.seed(100)
model_tree_full <- rpart(clcsHPI~., data = train, method = "anova",
                   control = list(cp = 0, xval = 16))
plotcp(model_tree_full, upper = c("none"), col = palette[2])
```




```r
#Create tree
set.seed(100)
model_tree <- rpart(clcsHPI~.,data=train, method = "anova")

#Visualize regression tree
rpart.plot(model_tree, box.palette = year_palette[c(21,15)])
```



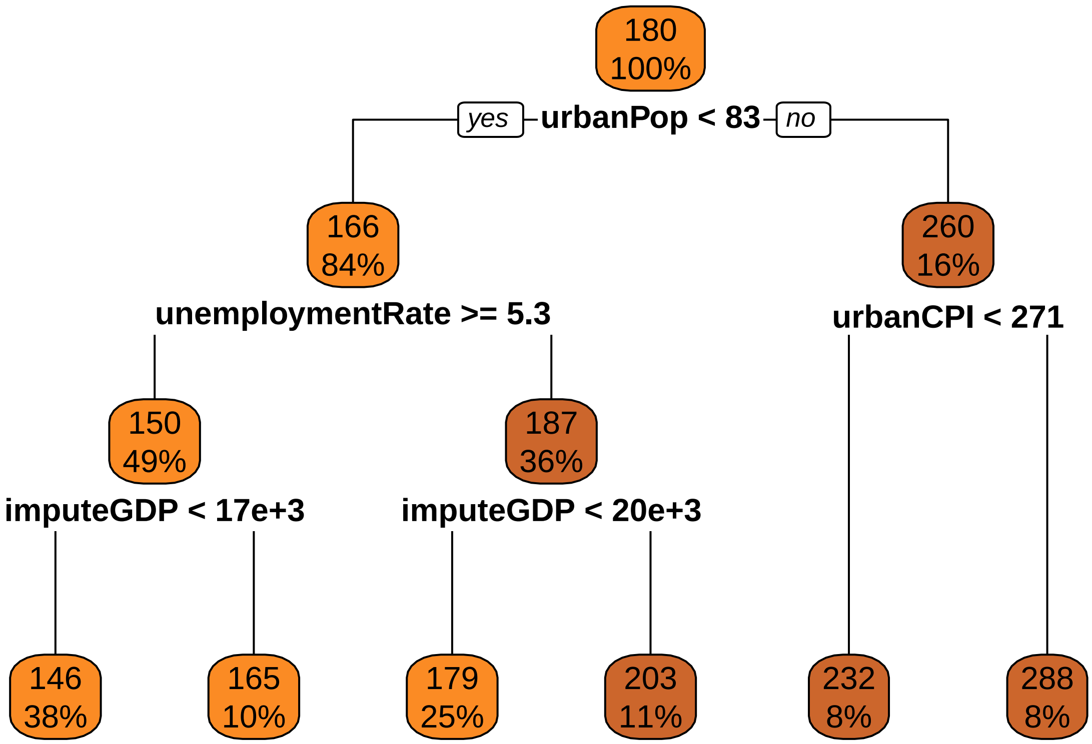
```r
#Check cp-size trade off -- maxdepth = 6
plotcp(model_tree, col = palette[2])
```




```r
#Grid search for optimal hyperparams

#Define search area
grid_tree <- expand.grid(
  minsplit = seq(5,20,1),
  maxdepth = seq(6,12,1)
)

#Search are for hyperparams
model_tree_list <- get_dt_hyperparameters(grid_tree = grid_tree,
                                          train_df = train_dt)

#Extract best minsplit and cp values
minsplit_optimal <- find_optimum_minsplit(grid_tree = grid_tree,
                                          model_list_tree = model_tree_list)

cp_optimal <- find_optimum_cp(grid_tree = grid_tree,
                              model_list_tree = model_tree_list)
```

```r
#Plot optimal tree
set.seed(100)
model_tree_optimal <- rpart(clcsHPI ~ ., data = train, method = "anova",
                            control = list(minsplit = minsplit_optimal, maxdepth = 6, cp = cp_optimal))
```

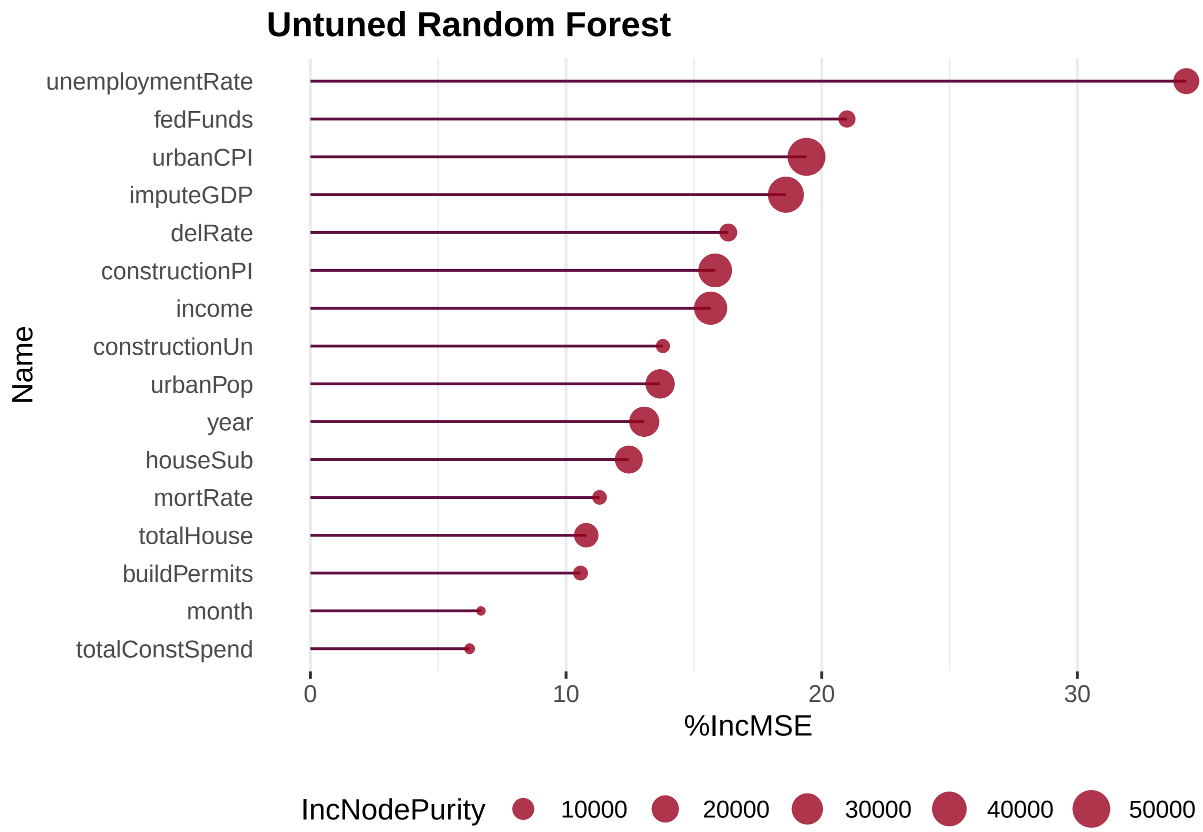
```r
#Visualize optimal regression tree
rpart.plot(model_tree_optimal, box.palette = year_palette[c(21,15)])
```




```r
preds_dt_optimal <- predict(model_tree_optimal, test_dt)


errors <- append_errors(errors_df = errors, preds = preds_dt_optimal,
                        model_name = "dt", test = test_dt,
                        target = "clcsHPI")
```

Random Forest

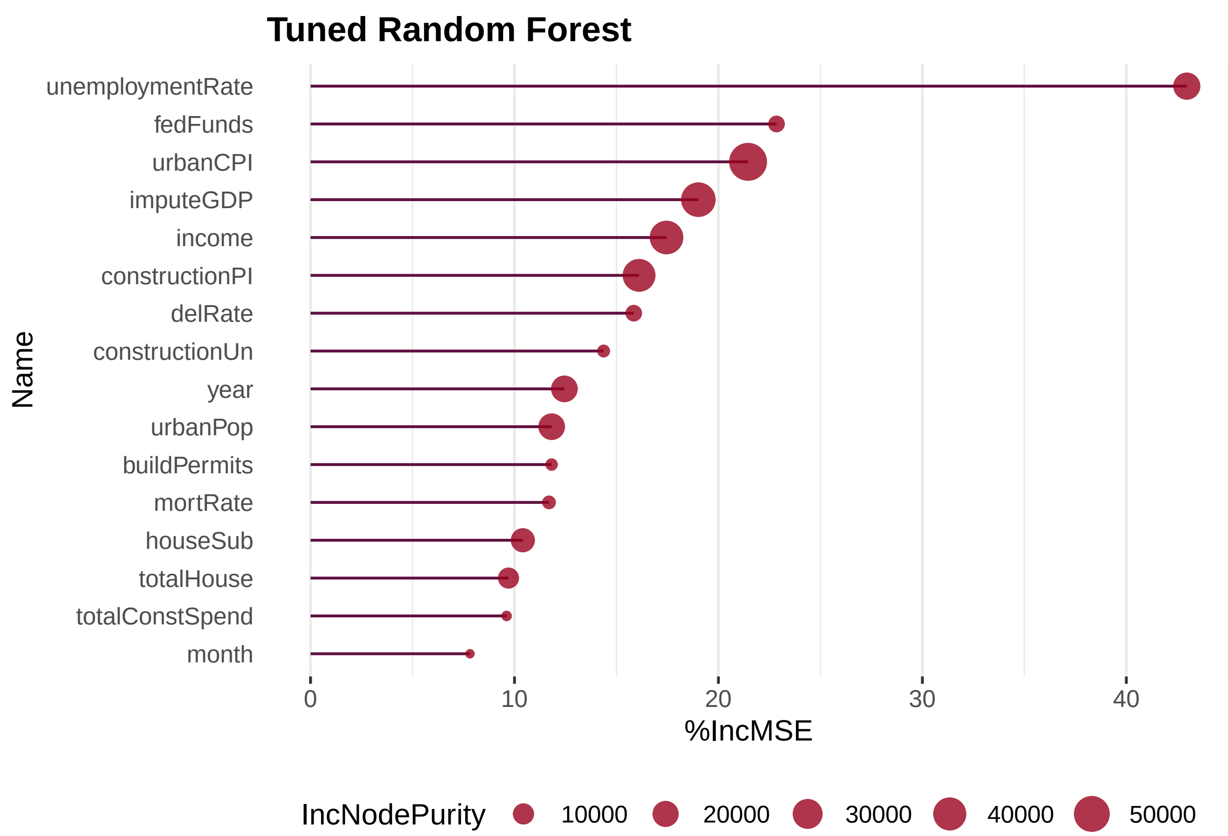
```r
train_rf <- train
test_rf <- test

# Initial model
set.seed(100)
model_forest <- randomForest(clcsHPI ~ ., data = train, ntree = 1000, importance = TRUE)

preds_forest <- predict(model_forest, test_rf)
errors <- append_errors(errors_df = errors, preds = preds_forest,
                        model_name = "untuned_rf", test = test_rf,
                        target = "clcsHPI")

plot_rf_var_imp(model_rf = model_forest, title = "Untuned Random Forest")
```




```r
#TuneRF
best_mtry_tune_rf <- tune_rf_get_best_mtry(train_df = train_rf, target_col = "clcsHPI",
                     step_factor = 1.5, improve = 1e-5, ntree = 501)
```

```r
#Grid search
best_mtry_grid <- grid_search_get_best_mtry(train_df = train_rf, num = 10, rpts = 3,
                          mtry_range = c(1:8))
```

```r
set.seed(100)
model_forest_tuned <- randomForest(clcsHPI~., data=train_rf,
                           ntree=1000, mtry=best_mtry_grid, importance = TRUE)

preds_forest_tuned <- predict(model_forest_tuned, test_rf)
errors <- append_errors(errors_df = errors, preds = preds_forest_tuned,
                        model_name = "tuned_rf", test = test_rf,
                        target = "clcsHPI")

plot_rf_var_imp(model_rf = model_forest_tuned, title = "Tuned Random Forest")
```



Interpretation of Results

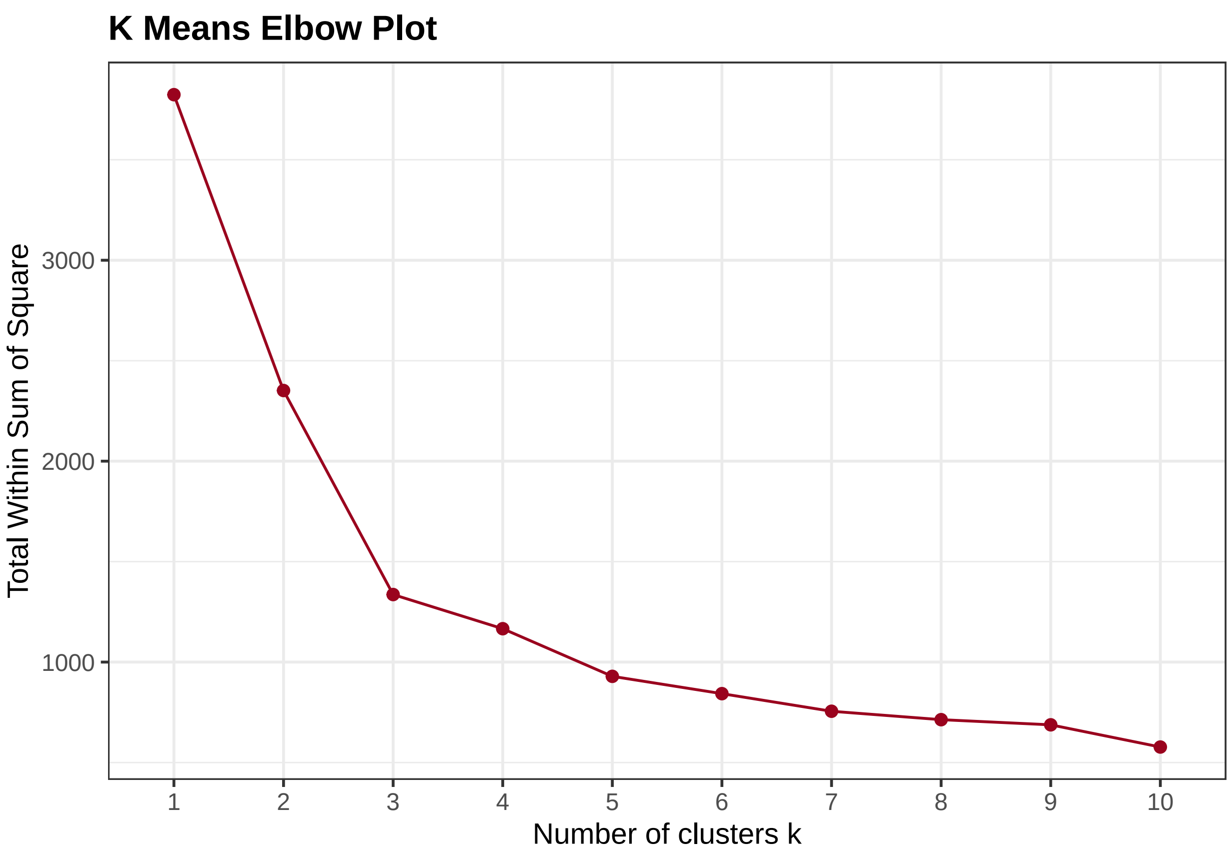
```
errors %>%
  rename("Linear Regression (No Ordinal Encoding)" = lr_nf,
         "Linear Regression (Ordinal Encoding)" = lr_f,
         "Lasso Regression" = lasso,
         "Ridge Regression" = ridge,
         "Decision Tree (Pruned)" = dt,
         "Random Forest (Untuned)" = untuned_rf,
         "Random Forest (Tuned)" = tuned_rf) %>%
  stargazer(type = "latex", flip = TRUE, summary = FALSE,
            header = FALSE,
            title = "Summary of Evaluation Metrics for all Tested Models")
```

Post-Hoc Analysis

```
df_scaled <- load_preprocess_scale_data()

plot_clustering_diagnostics(df_scaled = df_scaled, method = "wss", algorithm = "kmeans")
```

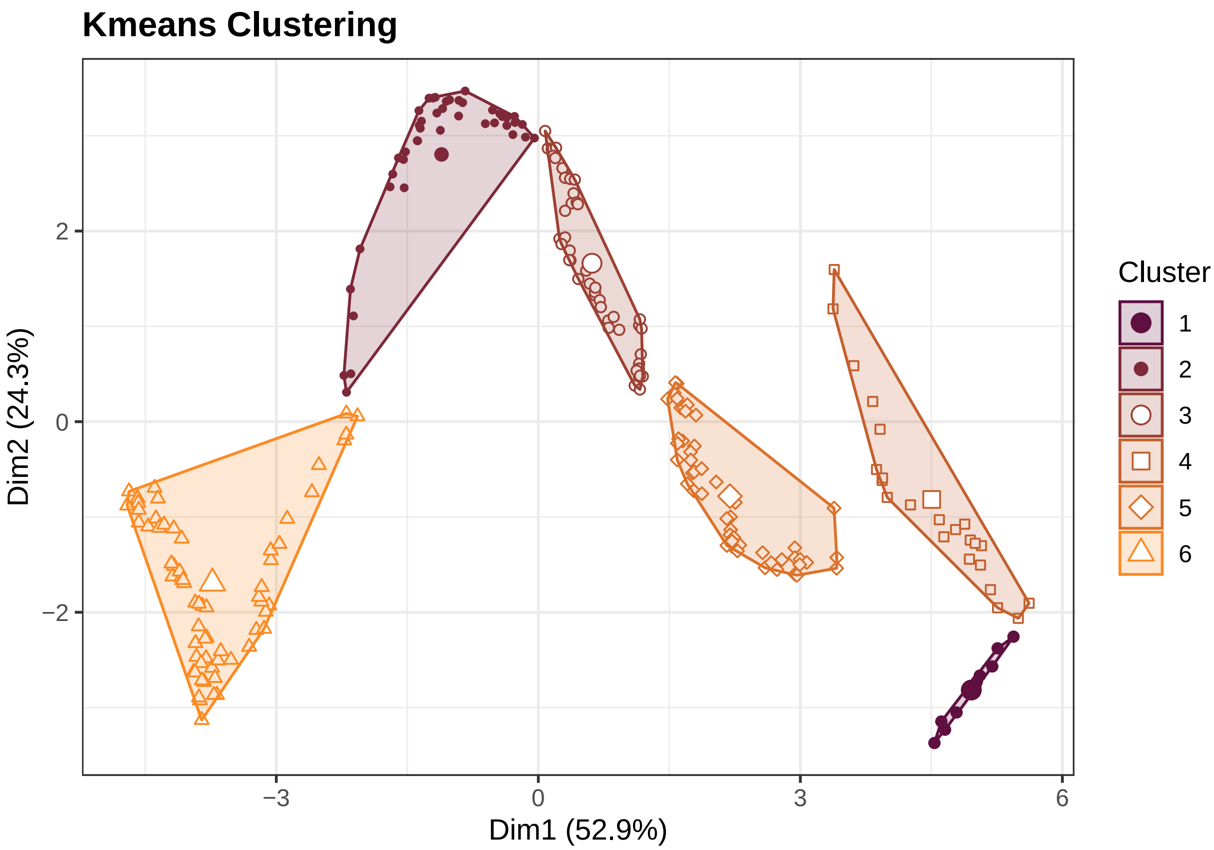
Table 8: Summary of Evaluation Metrics for all Tested Models

	MSE	RMSE	MAE	MAPE
Linear Regression (No Ordinal Encoding)	29.311	5.414	4.159	0.023
Linear Regression (Ordinal Encoding)	12.834	3.582	2.403	0.013
Lasso Regression	6.811	2.610	1.898	0.010
Ridge Regression	9.822	3.134	2.128	0.012
Decision Tree (Pruned)	59.473	7.712	6.289	0.034
Random Forest (Untuned)	2.274	1.508	1.107	0.006
Random Forest (Tuned)	2.229	1.493	1.100	0.006

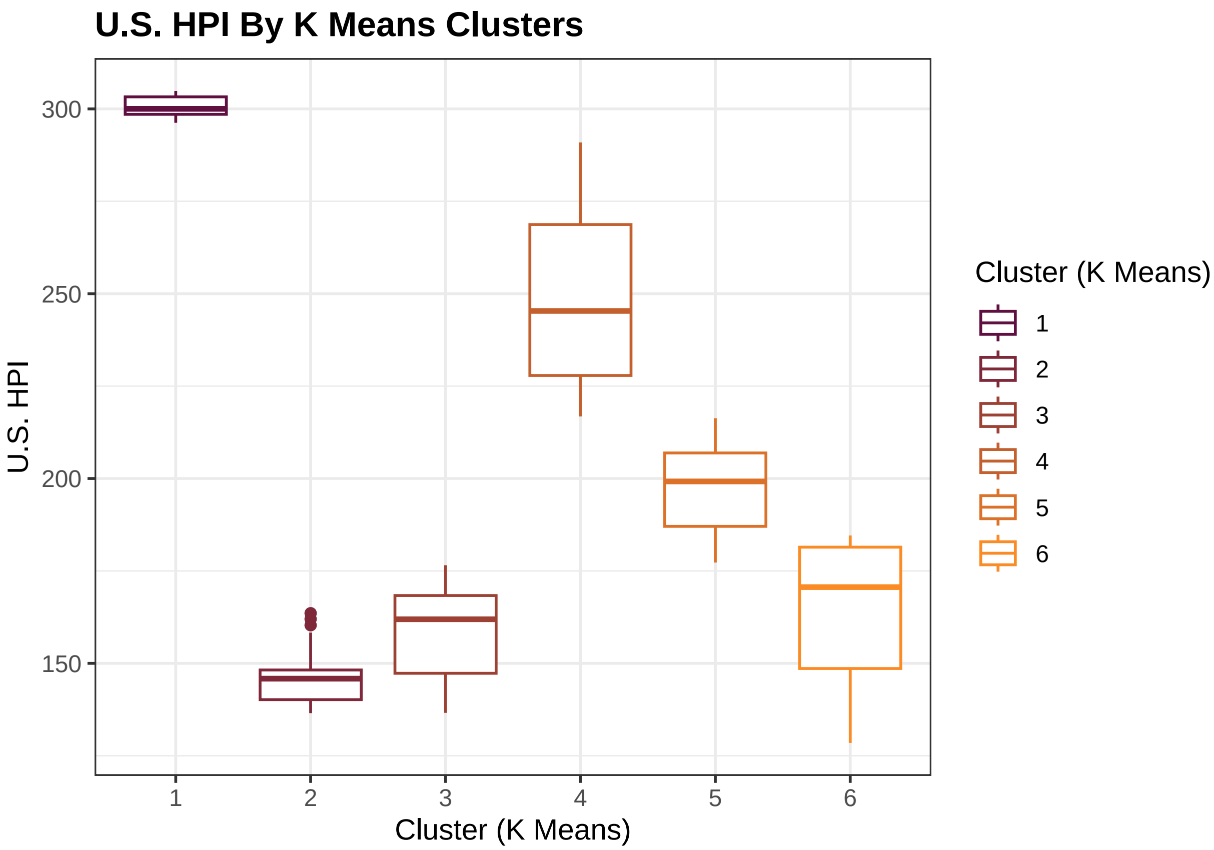



```
#Fit K Means -- use silhouette score since metrics don't agree
set.seed(100)
model_kmeans <- kmeans(df_scaled, centers = 6)
```

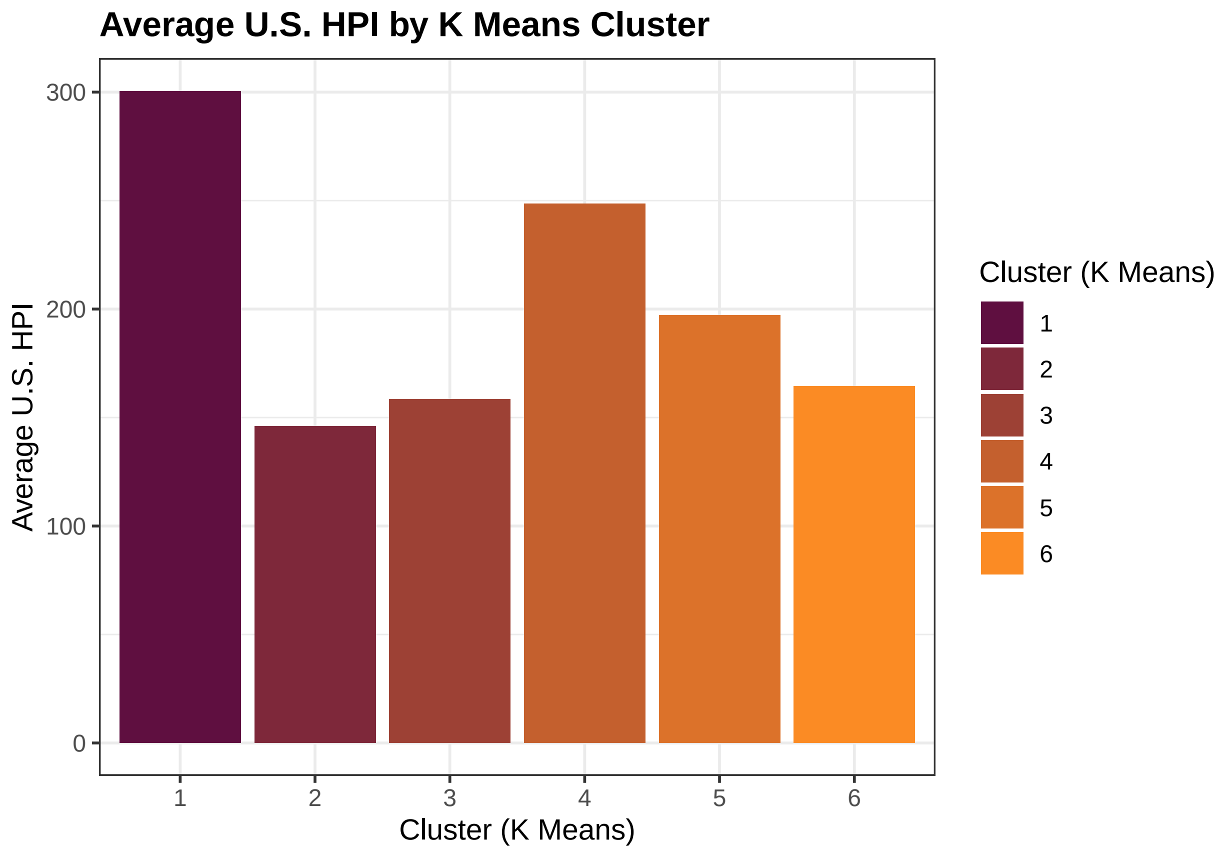
```
#Plot clusters
plot_clusters(model = model_kmeans, df_scaled = df_scaled, algorithm = "kmeans")
```



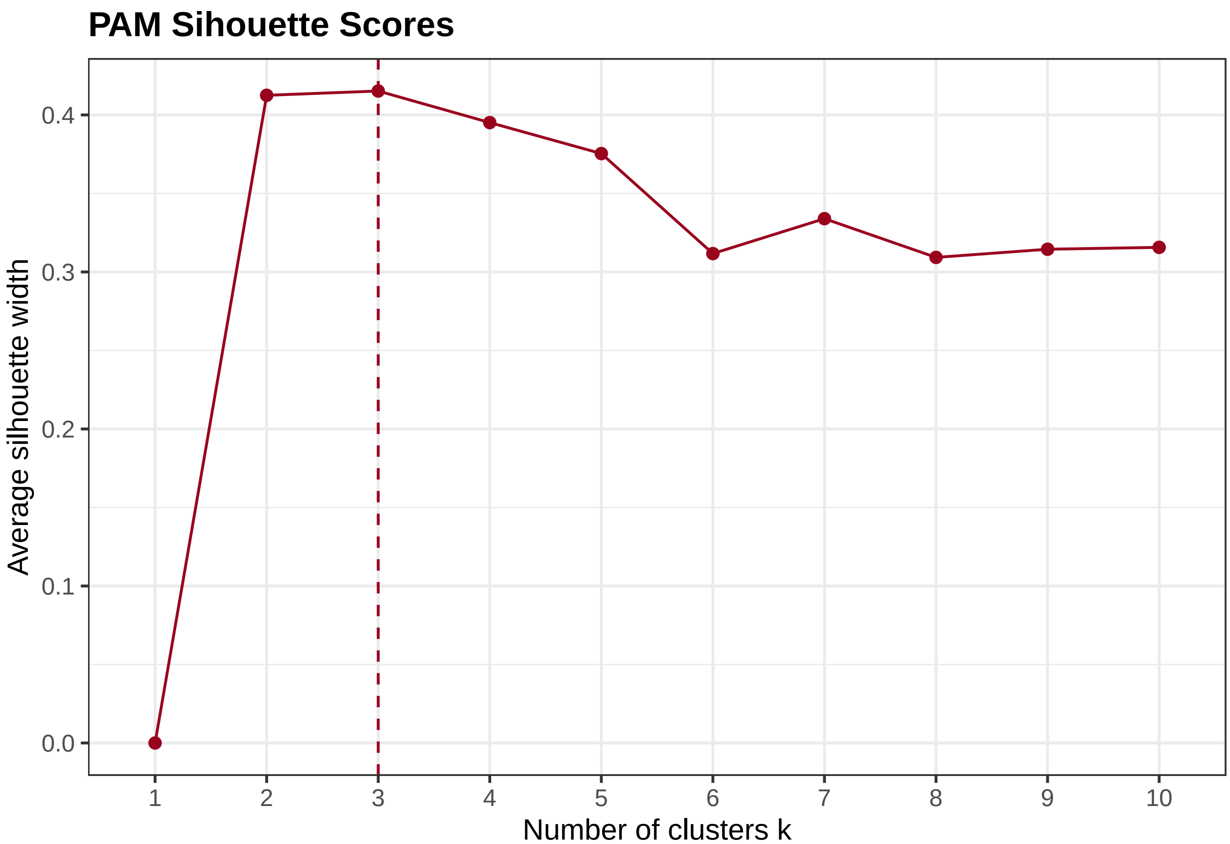

```
#Boxplots of median hpi across clusters
plot_median_hpi_across_clusters(model = model_kmeans, algorithm = "kmeans")
```




```
plot_bar_median_hpi_across_clusters(model = model_kmeans, algorithm = "kmeans")
```

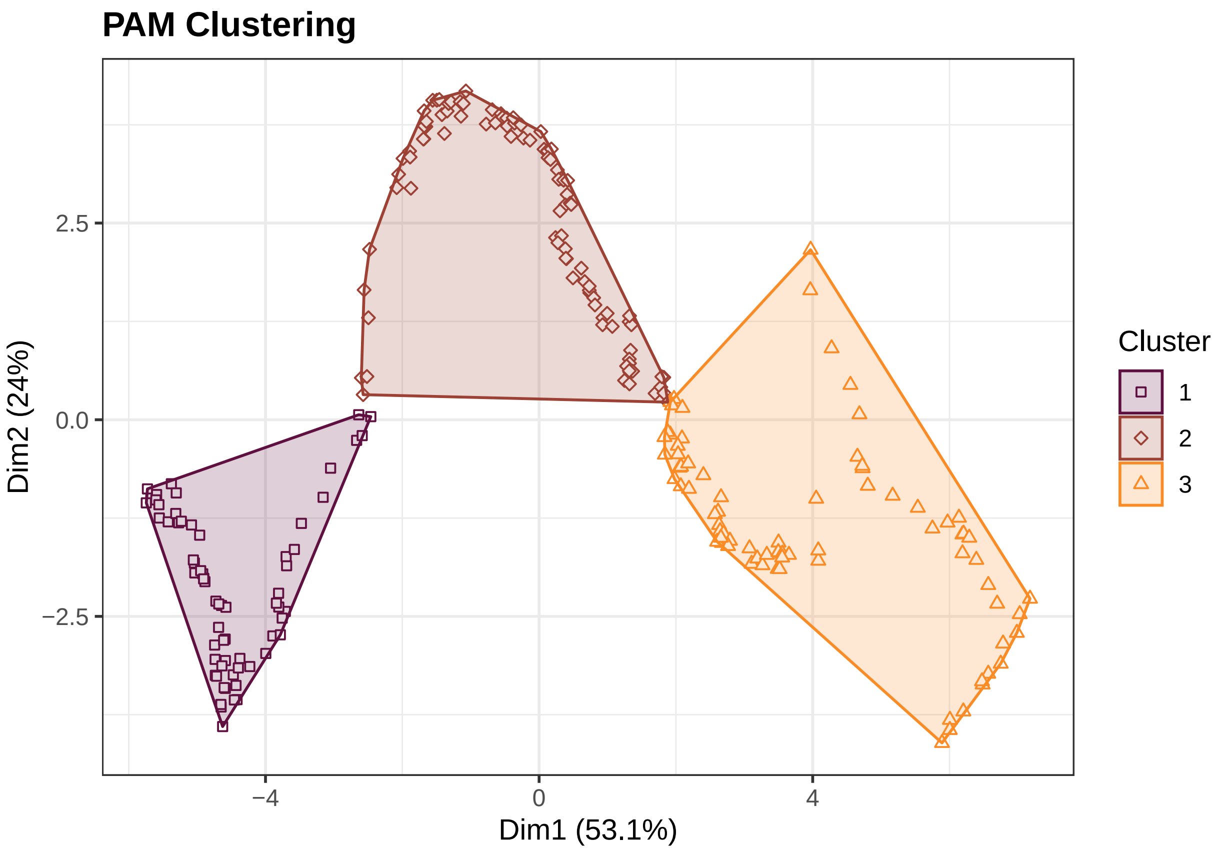



```
plot_clustering_diagnostics(df_scaled = df_scaled, method = "silhouette",
                            algorithm = "pam")
```

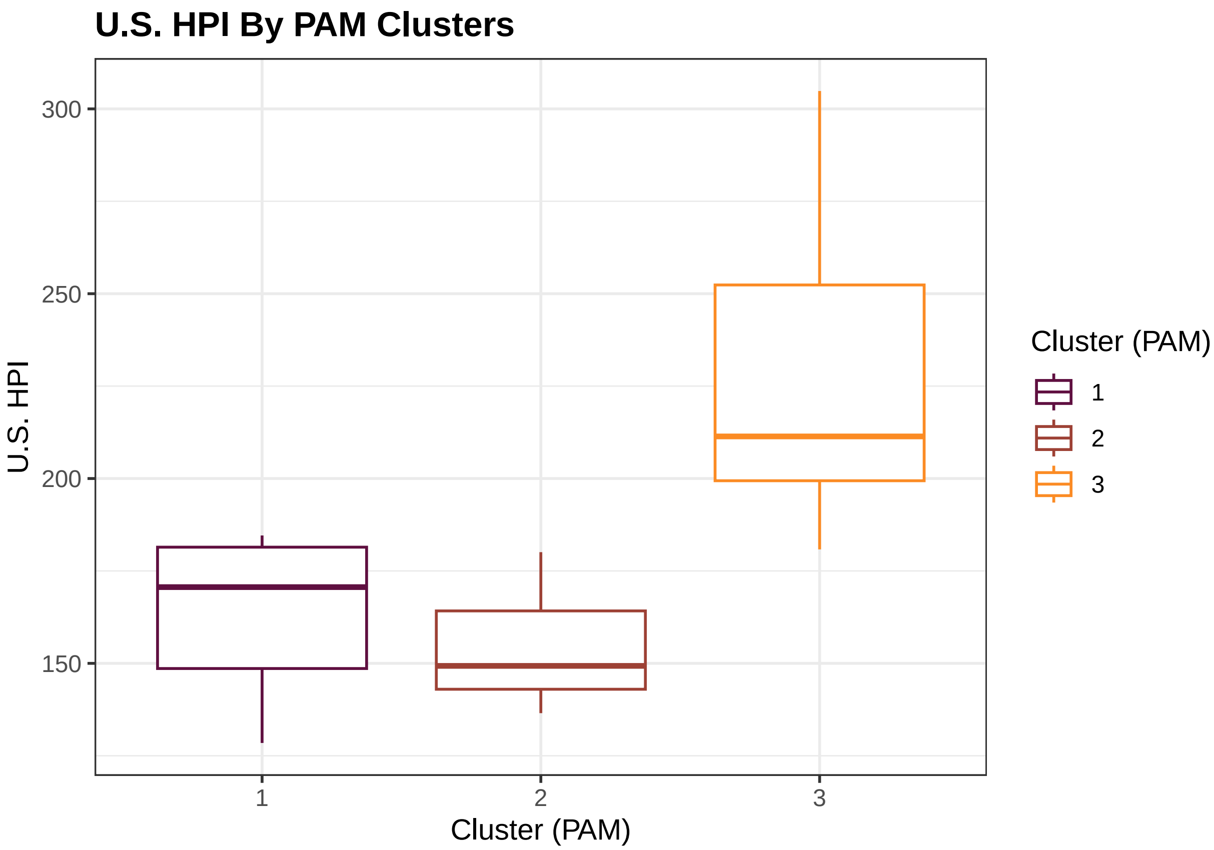



```
set.seed(100)
model_pam <- pam(df_scaled, stand = T, metric = "manhattan", k = 3)
```

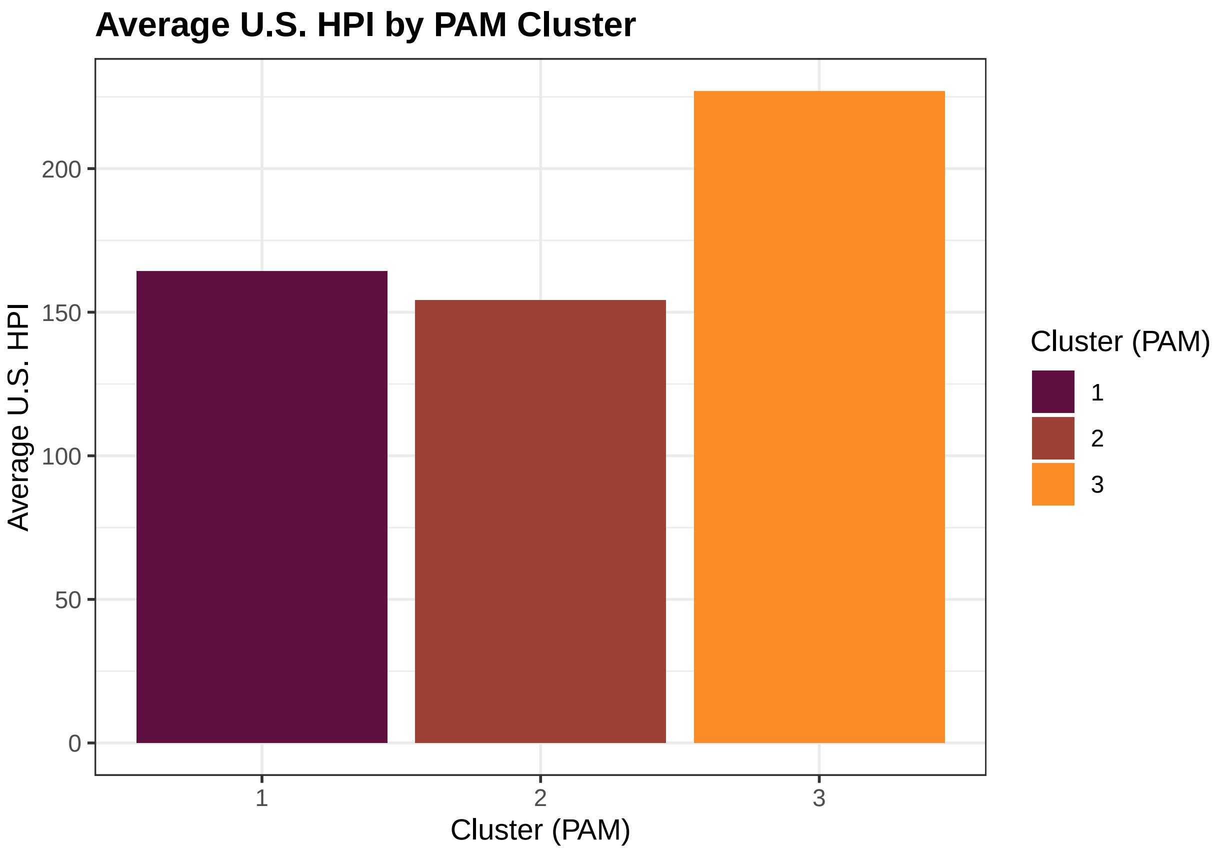
```
plot_clusters(model = model_pam, df_scaled = df_scaled, algorithm = "pam")
```




```r
plot_median_hpi_across_clusters(model = model_pam, algorithm = "pam")
```




```
plot_bar_median_hpi_across_clusters(model = model_pam, algorithm = "pam")
```



Conclusion

Limitations

Suggestions for Future Research

Works Cited

Liberto, D. (2023, August 29). Understanding the House price index (HPI) and how it is used. Understanding the House Price Index (HPI) and How It Is Used. https://www.investopedia.com/terms/h/house-price-index-hpi.asp

Rosen, P. (2023, August 11). The US housing market hits a record value of $47 trillion as the inventory shortage fuels a price boom. Business Insider. https://markets.businessinsider.com/news/commodities/housing-market-inventory-shortage-home-prices-value-real-estate-property-2023-8.

Code Appendix

Original Column Names

```
write(colnames(modeling_preliminary_data), stdout())
```

```
## DATE
## CPIAUCSL
## FEDFUNDS
## CSUSHPISA
## building_permits
## const_price_index
## delinquency_rate
## housing_subsidies
## income
## mortgage_rate
## construction_unit
## total_houses
## total_const_spending
## urban_population
## UNRATE
## imputed_GDP
```

```r
write(colnames(renamed_preliminary_data), stdout())
```

Renamed Column Names

```
## date
## urban_cpi
## fed_funds_rt
## hpi
## build_permits
## const_price_idx
## delinq_rt
## house_subsidies
## income
## mortgage_rt
## const_unit
## tot_house
## tot_const_spend
## urban_pop
## unem_rt
## imputed_gdp
```

```r
coef(glmnet(x_train, y_train, alpha = 1), s = lasso_optimal_lambda)
```

Lasso Coefficients

```
## 45 x 1 sparse Matrix of class "dgCMatrix"
##                             s1
## (Intercept)      -20.523994183
## urbanCPI           0.002599805
## fedFunds           2.748487470
```

```
## buildPermits       0.011278398
## constructionPI     0.372821451
## delRate           -2.546486339
## houseSub           0.552599952
## income             .
## mortRate           1.148669440
## constructionUn     0.006226760
## totalHouse         .
## totalConstSpend   -0.127509104
## urbanPop           .
## unemploymentRate   0.790487586
## imputeGDP          0.004534826
## year.L             .
## year.Q             1.453638527
## year.C            25.699977726
## year^4            -0.146399974
## year^5             .
## year^6             4.651611648
## year^7            -2.675087679
## year^8            -6.169210714
## year^9             .
## year^10            7.777334160
## year^11            0.322591132
## year^12            1.804398522
## year^13            6.018569346
## year^14            0.146362439
## year^15           -3.836262493
## year^16            1.955781782
## year^17            .
## year^18            .
## year^19           -0.628543017
## month.L            1.687724078
## month.Q            1.719489514
## month.C            0.780750868
## month^4           -0.912337736
## month^5           -0.698121724
## month^6           -0.249070075
## month^7           -0.217514397
## month^8           -0.023331159
## month^9            0.029186811
## month^10          -0.285266572
## month^11           0.041986505
```

```r
coef(glmnet(x_train, y_train, alpha = 0), s = ridge_optimal_lambda)
```

Ridge Coefficients

```
## 45 x 1 sparse Matrix of class "dgCMatrix"
##                               s1
## (Intercept)      -3.201699e+02
## urbanCPI          1.838138e-01
```

```
## fedFunds           1.807790e+00
## buildPermits       9.182184e-03
## constructionPI     1.760304e-01
## delRate           -1.322983e+00
## houseSub           8.710893e-01
## income             1.292202e-03
## mortRate           1.198345e+00
## constructionUn     8.111481e-03
## totalHouse         4.618046e-04
## totalConstSpend   -2.369552e-01
## urbanPop           3.376740e+00
## unemploymentRate  -7.459935e-01
## imputeGDP          1.412067e-03
## year.L             1.433803e+01
## year.Q             2.472300e+01
## year.C             3.554295e+01
## year^4            -2.281180e+00
## year^5             6.417949e+00
## year^6             4.952960e+00
## year^7            -6.340336e+00
## year^8            -5.192504e+00
## year^9            -2.109013e+00
## year^10            3.949263e+00
## year^11           -8.927535e-01
## year^12            8.003859e-01
## year^13            5.570331e+00
## year^14            1.211916e+00
## year^15           -4.093414e+00
## year^16            1.309796e+00
## year^17            8.575940e-02
## year^18           -7.044687e-01
## year^19           -6.135018e-01
## month.L            3.880782e+00
## month.Q            1.171167e+00
## month.C            5.655972e-01
## month^4           -2.291435e-01
## month^5           -5.476888e-01
## month^6           -6.344924e-01
## month^7            1.718766e-01
## month^8           -4.986018e-01
## month^9            3.983831e-01
## month^10          -7.345075e-01
## month^11           8.897317e-02
```

```r
print(model_kmeans)
```

K Means Output

```
## K-means clustering with 6 clusters of sizes 10, 43, 47, 23, 51, 66
##
## Cluster means:
```

```
##     urbanCPI    fedFunds  buildPermits constructionPI    delRate   houseSub
## 1  2.3436195  0.44496029  0.6638103272     2.87335363 -0.9048523  2.2215246
## 2 -0.4010153 -0.63777881 -1.4226417783    -0.37441284  1.4162151 -0.3144174
## 3  0.1515030 -0.75005175 -0.6085484062    -0.04393076  0.9417216 -0.1459103
## 4  1.3579647 -0.77363396  0.7190142242     1.56150290 -0.7160610  1.7327050
## 5  0.6763801  0.02530461  0.0009867641     0.32248423 -0.4498167  0.6276586
## 6 -1.1976023  1.13227824  1.0083275175    -0.95348941 -0.8590847 -1.1166715
##        income   mortRate constructionUn totalHouse totalConstSpend   urbanPop
## 1  1.43110219  0.8904030     0.47370464  1.6123260      -0.6022353  1.7270705
## 2 -0.54686439  0.1942210    -1.11329704 -0.5335842      -0.6576809 -0.4984329
## 3 -0.08065254 -0.6913514    -0.90414155  0.2506852       0.5088453  0.1304018
## 4  1.87663951 -1.4959705     0.27608907  1.2674579       0.7587086  1.4670484
## 5  0.78713906 -0.6173946    -0.06992498  0.9470017       0.1532764  0.8785723
## 6 -1.06533310  1.2292793     1.25523491 -1.2486358      -0.2254620 -1.2199436
##   unemploymentRate   imputeGDP       year       month
## 1       -1.1907291  2.27457112  1.6440730  0.28907859
## 2        1.3980166 -0.59205697 -0.4728974  0.09075723
## 3        0.3298213  0.00293697  0.1785835  0.02767774
## 4        0.4194123  1.48234850  1.4183422  0.04399022
## 5       -0.9014482  0.76728796  0.8907517 -0.07652080
## 6       -0.4148734 -1.07046971 -1.2507541 -0.07883962
##
## Clustering vector:
##   [1] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
##  [38] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
## [186] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [223] 4 4 4 4 4 4 4 4 1 1 1 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1]  27.30543 153.27363 107.37961 125.83200 134.77988 332.46574
##  (between_SS / total_SS =  77.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```r
print(model_pam)
```

PAM Output

```
## Medoids:
##       ID    urbanCPI   fedFunds buildPermits constructionPI    delRate
## [1,]  27 -1.34387949  0.8415364    1.5684095     -0.9615128 -1.0458408
## [2,] 126  0.08814872 -0.7675743   -0.7655022     -0.0909079  1.3443067
## [3,] 200  0.94762042  0.5247823    0.4556337      0.4835118 -0.7372395
##        houseSub     income   mortRate constructionUn totalHouse totalConstSpend
## [1,] -1.1698110 -1.2107204  1.1206700      1.3808488 -1.3087376      0.08994007
```

```
## [2,] -0.2438116 -0.2926722 -0.5521980     -1.0987573  0.1822986      0.24375683
## [3,]  0.9169342  1.1758566 -0.9609612      0.2177225  1.1833015      0.55139033
##        urbanPop unemploymentRate  imputeGDP        year      month
## [1,] -1.26318322       -0.3992312 -1.1613184 -1.29795241 -1.0117751
## [2,]  0.03581483        0.7314800 -0.1162979  0.08653016 -0.1445393
## [3,]  1.13489414       -1.1366516  1.1573754  1.12489209  0.4336179
## Clustering vector:
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
##  [75] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [149] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [186] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [223] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## Objective function:
##    build     swap
## 9.597077 8.635189
##
## Available components:
##  [1] "medoids"    "id.med"     "clustering" "objective"  "isolation"
##  [6] "clusinfo"   "silinfo"    "diss"       "call"       "data"
```