“MTCARS” Statistical Analysis

Dan Mailman

2024-05-23

Contents

List of Tables

List of Figures

1 Introduction

1.1 Background

One piece of information that influences car-buyer choices is **miles per gallon** (**mpg**). So, it would be useful to be able to predict **mpg** from other car-model variables.

the **mtcars** dataset provided with r is an example dataset for variables and **mpg** data.

1.2 Objectives

This analysis attempts to find a reasonably accurate model using **mtcars** to predict the **dependent variable** (**DV**), **mpg** from available **independent variables** (**IV**s).

Figure 1 shows the distribution of **mpg**.

The goal is to:

- Distinguish among significant and insignificant IVs.
- Rank significance for IVs.
- Produce a Model/Formula for predicting **mpg** from significant IVs.

2 Data Famililiarization

Learning what each of the variables represents enables a preliminary classification that eliminates two variables from consideration as predictors for **mpg**. Table 1 indicates **mtcars** IVs may be classified as **design** (specification for final mechanical assembly) and **measurement** (used to evaluate the finished assembly). By this criterion, **hp** and **qsec** are not best candidates for inclusion in predictive models for **mpg**.

To determine a preliminary ordering for model candidates, this study hypothesizes

1. Measurement IVs are not likely to be the best candidates for final model IVs.
2. IVs highly correlated to DV are better candidates for final model IVs.

3 Data Conditioning

To facilitate correlation (via spearman), IVs with low integral counts of unique values are treated as ordinals. IVs with larger variation or real values are treated as numerics. Table 2 and Table 3 summarize ordinal and numeric IVs.

4 Model IV Candidate Ordering by Correlation

IV/DV correlation magnitudes (whether positive or negative) are used to order IV candidates for inclusion in the model. Table 4 shows positive and negative correlations to DV in diminishing degree order. Taking absolute values of correlation coefficients produces an experimental candidate order:

mpg, cyl, disp, hp, wt, vs, carb, drat, am, gear, qsec

Removing the **measurement** variables **hp** and **qsec**:

mpg, cyl, disp, wt, vs, carb, drat, am, gear

Table 5 shows conditioned, re-ordered **mtcars** data.

Table 6 and Figure 2 show correlations with **mpg** and among IVs. Table 7 and figure Figure 3 show the same information removing measurement IVs.

5 Eliminating Multicollinearity and Forming a Preliminary Hypothesis

Highly correlated IVs are candidates for multicollinearity. Table 8 shows highly correlated IV groups. The IVs in the lower two groups don't correlate highly with **mpg**, so are not candidates for the predictive model.

At this point, we are able to hypothesize that the model will not include **hp** at all, and will only include a linear combination of **cyl**, **disp**, and **wt**. A little research into these variables indicates that **disp** is to large degree a physical function of **cyl** and that **wt** is independent of **cyl**. So, before using r tools to evaluate models, we can hypothesize that the model should be based on **cyl** and **wt**.

6 Using `step()` to Select Predictors

Using `step()` with models based on no IVs and the full set of IVs results in a ***STEP_MODEL*** formula/model for optimal set of predictors:

$$\textbf{mpg} = 38.75179 - 3.16697 \cdot \textbf{wt} - 0.94162 \cdot \textbf{cyl} - 0.01804 \cdot \textbf{hp}$$

Removing **hp** from ***STEP_MODEL*** for the ***WT_CYL_MODEL***:

$$\textbf{mpg} = 39.6863 - 3.1910 \cdot \textbf{wt} - 1.5078 \cdot \textbf{cyl}$$

7 Model Summary, Comparison, Conclusion

- **Model with wt and cyl only**
  - Coefficients: Both predictors (**wt** and **cyl**) are highly significant ($p < 0.01$).
  - Multiple R-squared: 0.8302
  - Adjusted R-squared: 0.8185
  - F-statistic: 70.91 on 2 and 29 DF, p-value: 6.809e-12
- **Model with wt, cyl, and hp**
  - Coefficients:
    - **wt** and **Intercept** remain highly significant.
    - **cyl** significance drops ($p < 0.1$, less significant than in the first model).
    - **hp** is not statistically significant ($p > 0.1$).
  - Multiple R-squared: 0.8431
  - Adjusted R-squared: 0.8263
  - F-statistic: 50.17 on 3 and 28 DF, p-value: 2.184e-11

The **wt/cyl** model is preferable due to its simplicity and the statistical significance of all included predictors. The addition of **hp** in the step_model does not provide enough benefit in terms of explained variance to justify its inclusion, especially given its lack of statistical significance and the minimal increase in adjusted R-squared. Table 9 and Figure 4 summarize the comparison of the two models. Results of programmatic comparison via this study's `compare_two_models()` (see appendix) follows.

```
mdlWtCylHp = lm(mpg ~ wt + cyl + hp, data = mtcars)
mdlWtCyl   = lm(mpg ~ wt + cyl,      data = mtcars)
compare_two_models(mdlWtCylHp, mdlWtCyl)
```

```
Comparison of Two Models:
Difference in AIC:  0.5334366
Difference in BIC:  0.9322993
Difference in Adjusted R-squared:  0.007825687

Both models perform similarly based on the criteria.
```

8 Tables

Table 1: Variable Classes

Class	Variables
Target DV	mpg
Design	cyl, disp, drat, wt, vs, am, gear, carb
Measurement	hp, qsec

Table 2: Ordinal Variables Summary

NumOrds	Ords	Counts	Description
2	0, 1	18, 14	Engine (0: V-shaped, 1: straight)
2	0, 1	19, 13	Transmission (0: auto, 1: manual)
3	4, 6, 8	11, 7, 14	Number of cylinders
3	3, 4, 5	15, 12, 5	Number of forward gears
6	1, 2, 3, 4, 6, 8	7, 10, 3, 10, 1, 1	Number of carburetors

Table 3: Numeric Variables Summary

Min	Median	Mean	Max	Uniqs	Description
10.400	19.200	20.090625	33.900	25	Miles/(US) gallon
71.100	196.300	230.721875	472.000	27	Displacement (cu.in.)
52.000	123.000	146.687500	335.000	22	Gross horsepower
1.513	3.325	3.217250	5.424	29	Weight (1000 lbs)
2.760	3.695	3.596563	4.930	22	Rear axle ratio
14.500	17.710	17.848750	22.900	30	1/4 mile time

Table 4: Ordered Model Candidate IVs

Class	Variables
Target DV	mpg
Direct	vs, drat, am, gear, qsec
Inverse	cyl, disp, hp, wt, carb
Not Best IV	hp, qsec

Table 5: Conditioned Cars Data

mpg	cyl	disp	hp	wt	vs	carb	drat	am	gear	qsec
21.0	6	160.0	110	2.620	0	4	3.90	1	4	16.46
21.0	6	160.0	110	2.875	0	4	3.90	1	4	17.02
22.8	4	108.0	93	2.320	1	1	3.85	1	4	18.61
21.4	6	258.0	110	3.215	1	1	3.08	0	3	19.44
18.7	8	360.0	175	3.440	0	2	3.15	0	3	17.02
18.1	6	225.0	105	3.460	1	1	2.76	0	3	20.22
14.3	8	360.0	245	3.570	0	4	3.21	0	3	15.84
24.4	4	146.7	62	3.190	1	2	3.69	0	4	20.00
22.8	4	140.8	95	3.150	1	2	3.92	0	4	22.90
19.2	6	167.6	123	3.440	1	4	3.92	0	4	18.30
17.8	6	167.6	123	3.440	1	4	3.92	0	4	18.90
16.4	8	275.8	180	4.070	0	3	3.07	0	3	17.40
17.3	8	275.8	180	3.730	0	3	3.07	0	3	17.60
15.2	8	275.8	180	3.780	0	3	3.07	0	3	18.00
10.4	8	472.0	205	5.250	0	4	2.93	0	3	17.98
10.4	8	460.0	215	5.424	0	4	3.00	0	3	17.82
14.7	8	440.0	230	5.345	0	4	3.23	0	3	17.42
32.4	4	78.7	66	2.200	1	1	4.08	1	4	19.47
30.4	4	75.7	52	1.615	1	2	4.93	1	4	18.52
33.9	4	71.1	65	1.835	1	1	4.22	1	4	19.90
21.5	4	120.1	97	2.465	1	1	3.70	0	3	20.01
15.5	8	318.0	150	3.520	0	2	2.76	0	3	16.87
15.2	8	304.0	150	3.435	0	2	3.15	0	3	17.30
13.3	8	350.0	245	3.840	0	4	3.73	0	3	15.41
19.2	8	400.0	175	3.845	0	2	3.08	0	3	17.05
27.3	4	79.0	66	1.935	1	1	4.08	1	4	18.90
26.0	4	120.3	91	2.140	0	2	4.43	1	5	16.70
30.4	4	95.1	113	1.513	1	2	3.77	1	5	16.90
15.8	8	351.0	264	3.170	0	4	4.22	1	5	14.50
19.7	6	145.0	175	2.770	0	6	3.62	1	5	15.50
15.0	8	301.0	335	3.570	0	8	3.54	1	5	14.60
21.4	4	121.0	109	2.780	1	2	4.11	1	4	18.60

Table 6: Initial Correlation Matrix

	mpg	cyl	disp	hp	wt	vs	carb	drat	am	gear	qsec
mpg	1.0000	-0.9108	-0.9089	-0.8947	-0.8864	0.7066	-0.6575	0.6515	0.5620	0.5428	0.4669
cyl	-0.9108	1.0000	0.9277	0.9018	0.8577	-0.8138	0.5801	-0.6789	-0.5221	-0.5643	-0.5724
disp	-0.9089	0.9277	1.0000	0.8510	0.8977	-0.7237	0.5398	-0.6836	-0.6241	-0.5945	-0.4598
hp	-0.8947	0.9018	0.8510	1.0000	0.7747	-0.7516	0.7334	-0.5201	-0.3623	-0.3314	-0.6666
wt	-0.8864	0.8577	0.8977	0.7747	1.0000	-0.5870	0.4998	-0.7504	-0.7377	-0.6761	-0.2254
vs	0.7066	-0.8138	-0.7237	-0.7516	-0.5870	1.0000	-0.6337	0.4475	0.1683	0.2827	0.7916
carb	-0.6575	0.5801	0.5398	0.7334	0.4998	-0.6337	1.0000	-0.1252	-0.0644	0.1149	-0.6587
drat	0.6515	-0.6789	-0.6836	-0.5201	-0.7504	0.4475	-0.1252	1.0000	0.6866	0.7448	0.0919
am	0.5620	-0.5221	-0.6241	-0.3623	-0.7377	0.1683	-0.0644	0.6866	1.0000	0.8077	-0.2033
gear	0.5428	-0.5643	-0.5945	-0.3314	-0.6761	0.2827	0.1149	0.7448	0.8077	1.0000	-0.1482
qsec	0.4669	-0.5724	-0.4598	-0.6666	-0.2254	0.7916	-0.6587	0.0919	-0.2033	-0.1482	1.0000

Table 7: Correlation Matrix w/o hp, qsec

	mpg	cyl	disp	wt	vs	carb	drat	am	gear
mpg	1.0000	-0.9108	-0.9089	-0.8864	0.7066	-0.6575	0.6515	0.5620	0.5428
cyl	-0.9108	1.0000	0.9277	0.8577	-0.8138	0.5801	-0.6789	-0.5221	-0.5643
disp	-0.9089	0.9277	1.0000	0.8977	-0.7237	0.5398	-0.6836	-0.6241	-0.5945
wt	-0.8864	0.8577	0.8977	1.0000	-0.5870	0.4998	-0.7504	-0.7377	-0.6761
vs	0.7066	-0.8138	-0.7237	-0.5870	1.0000	-0.6337	0.4475	0.1683	0.2827
carb	-0.6575	0.5801	0.5398	0.4998	-0.6337	1.0000	-0.1252	-0.0644	0.1149
drat	0.6515	-0.6789	-0.6836	-0.7504	0.4475	-0.1252	1.0000	0.6866	0.7448
am	0.5620	-0.5221	-0.6241	-0.7377	0.1683	-0.0644	0.6866	1.0000	0.8077
gear	0.5428	-0.5643	-0.5945	-0.6761	0.2827	0.1149	0.7448	0.8077	1.0000

Table 8: Multicollinearity Analysis

members	correlations
cyl, disp, hp, wt	cyl/disp (0.93); cyl/hp (0.90); cyl/wt (0.86); disp/hp (0.85); disp/wt (0.90); hp/wt (0.77)
vs, qsec	vs/qsec (0.79)
am, gear	am/gear (0.81)

Table 9: Model Predictions Comparison

Actual	mdlWtCylHp	mdlWtCyl
21.0	22.8204	22.2791
21.0	22.0128	21.4654
22.8	25.9604	26.2520
21.4	20.9361	20.3805
18.7	17.1678	16.6470
18.1	20.2504	19.5987
14.3	15.4934	16.2321
24.4	23.7643	23.4759
22.8	23.2957	23.6035
19.2	19.9890	19.6625
17.8	19.9890	19.6625
16.4	15.0824	14.6366
17.3	16.1592	15.7216
15.2	16.0008	15.5620
10.4	10.8944	10.8713
10.4	10.1630	10.3161
14.7	10.1426	10.5682
32.4	26.8275	26.6349
30.4	28.9327	28.5017
33.9	28.0014	27.7996
21.5	25.4290	25.7893
15.5	17.3654	16.3917
15.2	17.6346	16.6629
13.3	14.6383	15.3706
19.2	15.8852	15.3546
27.3	27.6667	27.4806
26.0	26.5665	26.8264
30.4	28.1554	28.8271
15.8	16.4175	17.5085
19.7	21.1729	21.8005
15.0	13.8700	16.2321
21.4	24.2150	24.7842

9 Figures

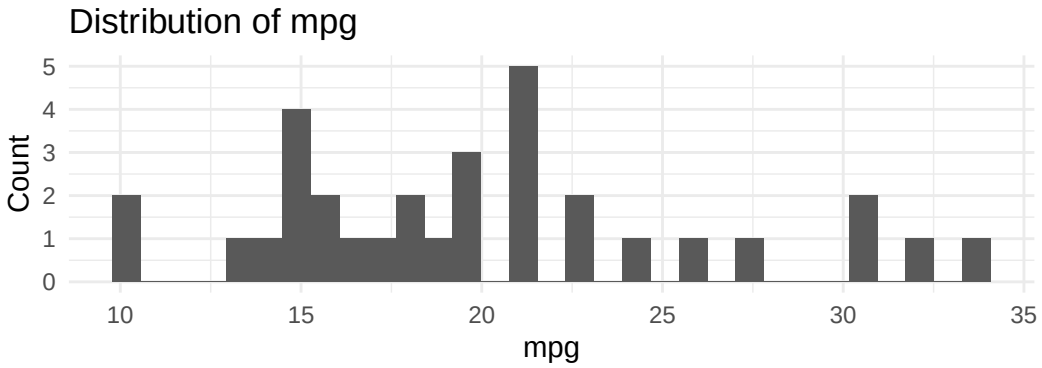

Figure 1: MPG Distribution

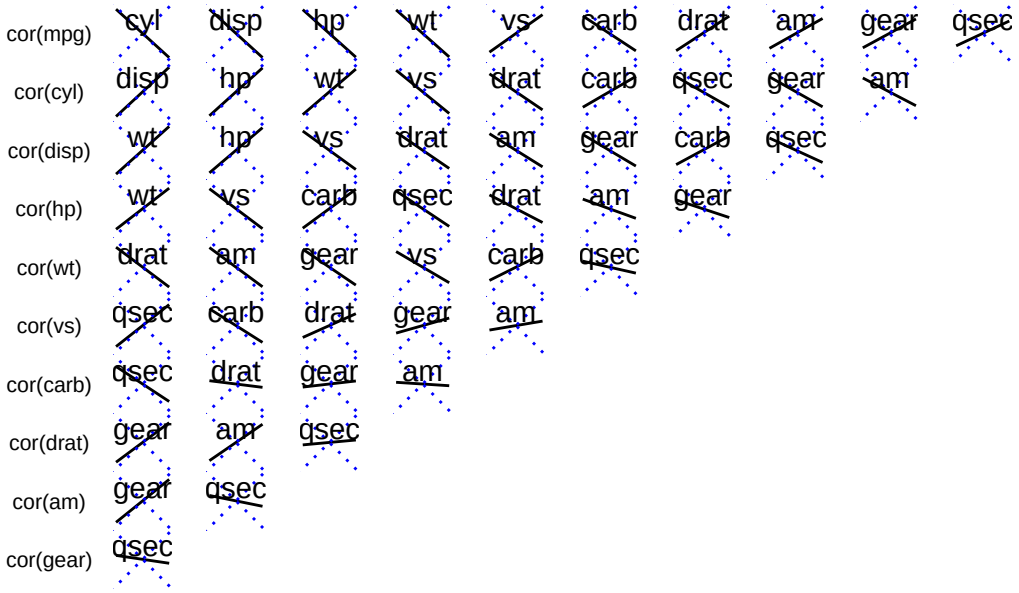

Figure 2: Correlations with MPG and Ordered IVs

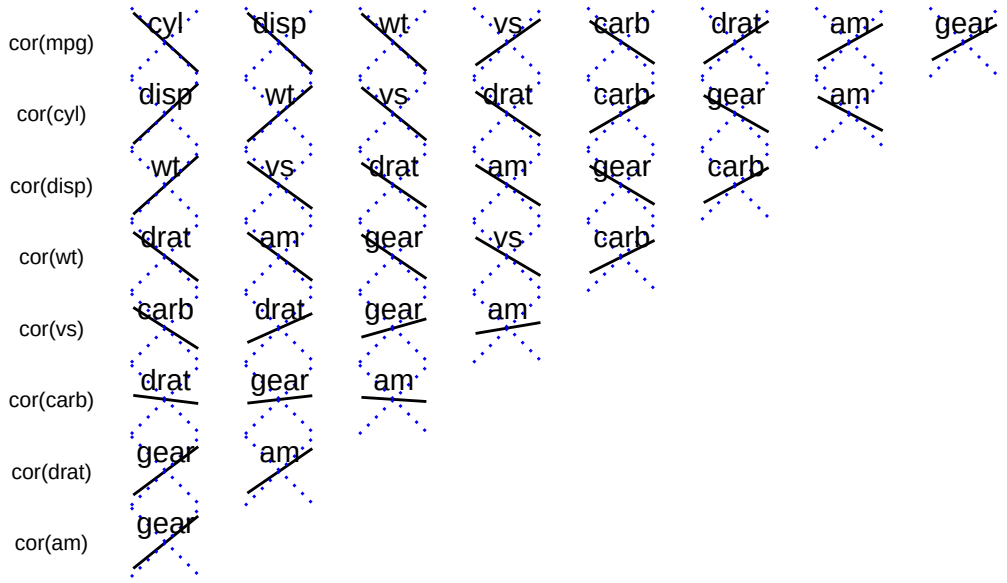

Figure 3: Ordered IV Correlations (w/o HP, QSEC)

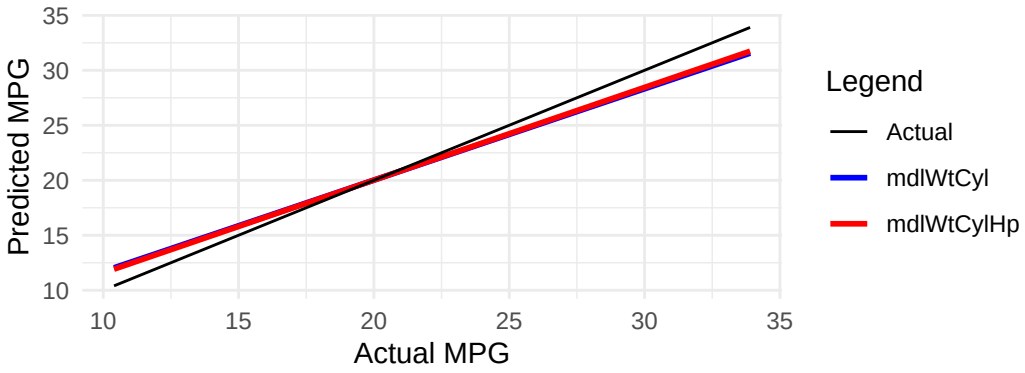

Figure 4: Predictive Model Comparison

10 Appendix - Code

```
# IMPORT/EXPORT DATA ##########################################################
dfCars = mtcars
write.csv(dfCars, "dfCars.csv", row.names = TRUE)
# CONSTANTS ###################################################################
vDesc = c(mpg  = "Miles/(US) gallon",
          cyl  = "Number of cylinders",
          disp = "Displacement (cu.in.)",
          hp   = "Gross horsepower",
          drat = "Rear axle ratio",
          wt   = "Weight (1000 lbs)",
          qsec = "1/4 mile time",
          vs   = "Engine (0: V-shaped, 1: straight)",
          am   = "Transmission (0: auto, 1: manual)",
          gear = "Number of forward gears",
          carb = "Number of carburetors")
vClass = c(mpg  = "Target DV",
           cyl  = "Design",
           disp = "Design",
           hp   = "Measurement",
           drat = "Design",
           wt   = "Design",
           qsec = "Measurement",
           vs   = "Design",
           am   = "Design",
           gear = "Design",
           carb = "Design")
# FUNCTION TO ANALYZE CORRELATION #############################################
AnalyzeCorrelation = function(nPVal, nCoeff, nDegree) {
  # Determine significance level based on p-value
  sSignif = ifelse(nPVal < 0.001, "highly significant",
                   ifelse(nPVal < 0.01, "very significant",
                          ifelse(nPVal < 0.05, "significant",
                                 "not significant")))

  # Determine the direction of the coefficient
  sDir = ifelse(nCoeff > 0, "positive",
                ifelse(nCoeff < 0, "negative", "zero"))

  # Determine the degree of correlation
  sDegree = ifelse(nDegree > 0.7, "strong",
                   ifelse(nDegree > 0.4, "moderate",
                          ifelse(nDegree > 0.1, "weak", "very weak")))

  # Return a list of the results
  return(list(Significance = sSignif, Direction = sDir, Degree = sDegree))
}


# FUNCTION TO GET CORRELATION, P-VALUE,SIGNIFICANCE,DIRECTION, DEGREE ##########
GetCorr = function(df, col1, col2) {
  # Check if columns exist in the dataframe
  if (!(col1 %in% names(df) && col2 %in% names(df))) {
    stop("One or both specified columns do not exist in the dataframe.")
  }
```

```r
  # Get data from the specified columns
  data1 = df[[col1]]
  data2 = df[[col2]]

  # Convert ordinal factors to numeric if necessary
  if (is.ordered(data1)) {
    data1 = as.numeric(levels(data1))[data1]
  }
  if (is.ordered(data2)) {
    data2 = as.numeric(levels(data2))[data2]
  }

  # Perform Spearman correlation test
  test_result = suppressWarnings(cor.test(data1, data2, method = "spearman"))

  # Extract the correlation coefficient and p-value
  correlation = test_result$estimate
  p_value = test_result$p.value

  # Calculate the degree of correlation (can be modified based on your criteria)
  nDegree = abs(correlation)

  # Analyze correlation significance, direction, and degree
  analysis_results = AnalyzeCorrelation(p_value, correlation, nDegree)

  # Combine and return all results
  return(list(
    Correlation = correlation,
    P_Value = p_value,
    Significance = analysis_results$Significance,
    Direction = analysis_results$Direction,
    Degree = analysis_results$Degree
  ))
}

# FUNCTION TO GET IV CORRELATION ORDER AND ORDERED CORRELATION MATRIX #########
GetCorrMatrix = function(df, dv) {
  # Ensure the dependent variable is in the dataframe
  if (!dv %in% names(df)) {
    stop("The dependent variable does not exist in the dataframe.")
  }

  # Create an empty matrix to store correlation values
  n = ncol(df)
  corr_matrix = matrix(nrow = n, ncol = n)
  rownames(corr_matrix) = colnames(corr_matrix) = names(df)

  # Compute pairwise correlations using GetCorr
  for (i in seq_len(n)) {
    for (j in seq_len(i)) {  # Fill lower triangle and diagonal
      corr_result = GetCorr(df, names(df)[i], names(df)[j])
      corr_matrix[i, j] = corr_matrix[j, i] = corr_result$Correlation
      # Symmetric matrix
    }
  }
```

```
  # Move the DV to the first position
  cols_order = c(dv, setdiff(names(df), dv))
  corr_matrix = corr_matrix[cols_order, cols_order]

  # Reorder the remaining IV columns by the magnitude of their correlation with the DV
  iv_cols = setdiff(cols_order, dv)
  iv_order = iv_cols[order(abs(corr_matrix[dv, iv_cols]), decreasing = TRUE)]
  final_cols_order = c(dv, iv_order)
  corr_matrix = corr_matrix[final_cols_order, final_cols_order]

  # Prepare the results to return
  results = list(
    CorrelationMatrix = corr_matrix,
    ColumnOrder = final_cols_order
  )

  return(results)
}
# CONDITION MTCARS,GENERATE CORRELATION MATRIX,GET IV CORRELATION ORDER ########
GetCondDF  = function(df, dv) {
  MakeOrdsDF = function(df, ORDS_THRESHOLD = 9) {
    # ORDINALIZE VARS WITH LOW COUNTS OF UNIQUE INTEGRAL VALUES
    dfRet = df
    for (col_name in names(df)) {
      col_data = df[[col_name]]
      if (is.numeric(col_data) && all(col_data == floor(col_data))) {
        unique_values = unique(col_data)
        num_unique_values = length(unique_values)
        if (num_unique_values <= ORDS_THRESHOLD) {
          dfRet[[col_name]] = factor(col_data,
                                     levels = sort(unique_values),
                                     ordered = TRUE)
        }
      }
    }
    return(dfRet)
  }
  dfRet = MakeOrdsDF(df)
  oGetCorrMatrix = GetCorrMatrix(dfRet, dv)
  mCorr = oGetCorrMatrix$CorrelationMatrix
  vCorrOrder = oGetCorrMatrix$ColumnOrder
  dfRet = dfRet[, vCorrOrder]
  vSummaries = list()
  for (col_name in names(dfRet)[-1]) {
    if (is.ordered(dfRet[[col_name]])) {
      levels = levels(dfRet[[col_name]])
      counts = table(factor(dfRet[[col_name]], levels = levels))
      vSummaries[[col_name]] = data.frame(
        Levels = paste(levels, collapse=", "),
        Counts = I(list(counts)))  # Use I() to keep list structure in df col
    }
  }
  if (length(vSummaries) > 0) {
    dfSumm = do.call(rbind, vSummaries)
    rownames(dfSumm) = names(vSummaries)
  } else {
```

```
    dfSumm = data.frame()  # Return an empty dataframe if no vSummaries exist
  }
  results = list(df = dfRet,
                 mat = mCorr,
                 order = vCorrOrder,
                 dfSumm = dfSumm)
  return(results)
}
oGetCondDF = GetCondDF(dfCars,'mpg')
dfCars      = oGetCondDF$df
mCorrs      = oGetCondDF$mat
vCorrOrder    = oGetCondDF$order
rm(oGetCondDF)
# FUNCTIONS, DATAFRAMES FOR TABLES ############################################
GetClassTblDF = function(vClass) {
  dfRet = data.frame(Class = unique(vClass),
                     Variables = "",
                     stringsAsFactors = FALSE)
  for (i in 1:nrow(dfRet)) {
    class_name = dfRet$Class[i]
    variables = names(vClass)[vClass == class_name]
    dfRet$Variables[i] = toString(variables)
  }
  return(dfRet)
}
dfClassTbl = GetClassTblDF(vClass)
GetSummDF = function(df, Descriptions) {
  if (!is.data.frame(df)) { stop("df must be a data frame.") }
  # Create a dataframe with variable names and their types
  dfRet = data.frame(Type = ifelse(sapply(df, is.numeric), "num",
                                   ifelse(sapply(df, is.ordered),
                                          "ord", "other")),
                     stringsAsFactors = FALSE)
  rownames(dfRet) = names(df)  # Set rownames as variable names
  vNumStats = c("Min", "Median", "Mean", "Max", "Uniqs")
  vOrdStats = c("NumOrds", "Ords", "Counts")
  for (stat in c(vNumStats, vOrdStats)) { dfRet[stat] = NA }
  for (i in seq_along(df)) {
    var_name = names(df)[i]
    var_data = df[[var_name]]
    if (is.numeric(var_data)) {
      dfRet[var_name, vNumStats] = c(min(var_data, na.rm = TRUE),
                                     median(var_data, na.rm = TRUE),
                                     mean(var_data, na.rm = TRUE),
                                     max(var_data, na.rm = TRUE),
                                     length(unique(var_data)))
    } else if (is.ordered(var_data)) {
      levels_info = levels(var_data)
      counts_info = table(var_data)
      dfRet[var_name, vOrdStats] = c(length(levels_info),
                                     toString(levels_info),
                                     toString(counts_info))
    }
  }
  dfRet$Description = Descriptions[rownames(dfRet)]
  return(dfRet)
```

```r
}
dfVarsSumm = GetSummDF(dfCars,vDesc)
GetOrderedTypeSummDF = function(df, type, order_by) {
  # Ensure df is a dataframe
  if (!is.data.frame(df)) {
    stop("Input must be a data frame.")
  }

  # Process the dataframe, filtering to include only rows of the specified type
  dfRet = df %>%
    filter(Type == type) %>%
    select(-Type) %>%
    select_if(~any(!is.na(.)))  # Remove columns that are entirely NA

  # Determine the type of order_by to decide the ordering approach
  if (is.character(order_by) && length(order_by) == 1 && order_by %in% names(df)) {
    # Order by a column within the dataframe
    dfRet = dfRet %>%
      arrange(!!sym(order_by))
  } else if (is.character(order_by)) {
    # Assume order_by is a vector of row names or similar identifiers
    dfRet = dfRet %>%
      mutate(row_name = rownames(.)) %>%
      filter(row_name %in% order_by) %>%
      mutate(order = match(row_name, order_by)) %>%
      arrange(order) %>%
      `rownames<-`(., .$row_name) %>%
      select(-row_name, -order)
  } else {
    stop("order_by must be a valid column name or a character vector for row ordering.")
  }

  return(dfRet)
}
dfOrdsSummTbl = GetOrderedTypeSummDF(dfVarsSumm, "ord", "NumOrds")
dfNumsSummTbl = GetOrderedTypeSummDF(dfVarsSumm, "num", vCorrOrder)
GetIVModelOrdTblDF = function(mCorr, vIgnore) {
  # Ensure inputs are valid
  if (!is.matrix(mCorr) || is.null(colnames(mCorr))) {
    stop("mCorr must be a named matrix.")
  }
  if (!is.vector(vIgnore)) {
    stop("vIgnore must be a vector.")
  }

  # Identify the target dependent variable (first column of the matrix)
  dv = colnames(mCorr)[1]

  # Extract the correlation coefficients for the DV, removing the DV itself
  correlations = mCorr[dv, -1]

  # Classify the variables
  direct = names(correlations[correlations > 0])
  inverse = names(correlations[correlations < 0])
  not_best_iv = intersect(vIgnore, colnames(mCorr))

  # Sort variables by absolute value of their correlations
```

```
  direct = direct[order(-abs(correlations[direct]))]
  inverse = inverse[order(-abs(correlations[inverse]))]

  # Convert lists to comma-separated strings
  direct = paste(direct, collapse = ', ')
  inverse = paste(inverse, collapse = ', ')
  not_best_iv = paste(not_best_iv, collapse = ', ')

  # Create the dataframe with complete lists
  df = data.frame(
    Class = c("Target DV", "Direct", "Inverse", "Not Best IV"),
    Variables = c(dv, direct, inverse, not_best_iv),
    stringsAsFactors = FALSE
  )
  return(df)
}
dfIVModelOrdTbl = GetIVModelOrdTblDF(mCorrs,c('hp','qsec'))
# FUNCTIONS TO MAKE PLOTS ##################################################
GetDistribPlot = function(df, col, dfName = deparse(substitute(df))) {
  # Return Distribution Plot for Col in DF
  if (!col %in% names(df)) {
    stop("Column ", sQuote(col), " not in DF ", dQuote(dfName), ".")
  }
  vCol = df[[col]] # Extract column data
  if (is.numeric(vCol)) { # Numeric distribution plot
    p = ggplot(df, aes_string(x = col)) +
      geom_histogram(binwidth = diff(range(vCol, na.rm = TRUE)) / 30) +
      theme_minimal() +
      labs(title = paste("Distribution of", col), x = col, y = "Count")
  } else if (is.factor(vCol)) { # Factor distribution plot
    p = ggplot(df, aes_string(x = col, fill = col)) +
      geom_bar() +
      labs(title = paste("Distribution of", col), x = col, y = "Frequency") +
      theme_minimal() +
      theme(axis.text.x = element_text(angle = 90, hjust = 1))
  } else {
    stop("Data type '", typeof(vCol), "' of column ",
         sQuote(col), " is not supported for distribution plotting.")
  }
  return(p)
}
# FUNCTIONS AND VARIABLES FOR CORRELATION FIGURES ##############################
remove_from_vector = function(vector, remove_values) {
  # Remove specified values from the vector
  filtered_vector = vector[!vector %in% remove_values]
  return(filtered_vector)
}
remove_from_matrix = function(matrix, remove_values) {
  # Find indices of the values to remove from the matrix row and column names
  indices_to_remove = which(rownames(matrix) %in% remove_values | colnames(matrix) %in% remove_values)

  # Remove the corresponding rows and columns
  updated_matrix = matrix[-indices_to_remove, -indices_to_remove]
  return(updated_matrix)
}
GetMaxSquarePlotDims = function(n_rows, n_cols) {
  min(inPrtW / n_cols, inPrtH / n_rows)
```

```
}
GetCorrPlot = function(nCorr, sTitle) {
  if (is.na(nCorr) || is.nan(nCorr) || abs(nCorr) > 1) {
    return(ggplot() + annotate("text", x = 0, y = 0,
                               label = "Not plotable",
                               hjust = 0.5, vjust = 0.5))
  }

  dfPlot = data.frame(x = c(-1, 1), y = c(-1, 1) * nCorr)
  pltRet = ggplot(dfPlot, aes(x = x, y = y)) +
    geom_line() +
    geom_abline(intercept = 0, slope = 1, linetype = "dotted", color = "blue") +
    geom_abline(intercept = 0, slope = -1, linetype = "dotted", color = "blue") +
    coord_fixed(ratio = 1) +
    xlim(-1, 1) +
    ylim(-1, 1) +
    annotate("text", x = 0, y = 1, label = sTitle,
             hjust = 0.5, vjust = 1.1, size = 4, color = "black") +
    theme_void() +
    theme(plot.margin = margin(t = 0, r = 0, b = 0, l = 0, unit = "mm"))

  return(pltRet)
}
GetIndVarNamesAndIndices = function(nColDV, mCorrs) {
  if (!is.matrix(mCorrs) || nColDV < 1 || nColDV >= ncol(mCorrs)) {
    stop("Invalid input: Ensure mCorrs is a matrix and nColDV is a valid column index.")
  }
  nColFirstIV = nColDV + 1
  vIdx = (nColFirstIV):ncol(mCorrs)
  vNam = colnames(mCorrs)[vIdx]
  vVal = mCorrs[nColDV,vIdx]
  mOrig = data.frame(nam = vNam, idx = vIdx, val = vVal,
                     stringsAsFactors = FALSE)
  mReord = mOrig[order(abs(mOrig$val), decreasing = TRUE), ]  # Use abs() for sorting
  list(OrigNam = mOrig$nam,
       OrigIdx = mOrig$idx,
       OrigVal = mOrig$val,
       ReordNam = mReord$nam,
       ReordIdx = mReord$idx,
       ReordVal = mReord$val)
}
CheckValidCorrMatrixIdx = function(n, m) {
  # Check if n is a valid row index
  if (n < 1 || n > nrow(m)) {
    stop(sprintf("%d: Invalid Row Idx. Must be between 1 and %d.", n, nrow(m)))
  }
  # Check if n is also a valid column index
  if (n < 1 || n > ncol(m)) {
    stop(sprintf("%d: Invalid Col Idx. Must be between 1 and %d.", n, ncol(m)))
  }
  # If both checks are passed, return TRUE
  return(TRUE)
}
GetIndVarPlotRow = function(mCorrs, nColDV) {
  if (!CheckValidCorrMatrixIdx(nColDV, mCorrs)) {
    stop("Invalid column index for the matrix.")
  }
```

```r
  oIndVarNamIdxVal = GetIndVarNamesAndIndices(nColDV, mCorrs)
  vPlots = lapply(seq_along(oIndVarNamIdxVal$ReordIdx), function(i) {
    idx = oIndVarNamIdxVal$ReordIdx[[i]]
    single_plot = GetCorrPlot(mCorrs[nColDV, idx], rownames(mCorrs)[idx])
    return(single_plot)
  })

  # Prepare the rho label as a ggplot object
  dep_var_name = rownames(mCorrs)[nColDV]  # Dependent variable name
  #rho_label = paste('$\\rho(', colnames(mCorrs)[nColDV], ')$', sep='')
  rho_label = paste("cor(", dep_var_name, ")", sep = "")
  #rho_label = expression(paste("rho(", dep_var_name, ")", sep = ""))
  #rho_label = expression(rho[.(dep_var_name)])
  rho_plot = ggplot() +
    geom_text(aes(x = 0, y = 0), label = rho_label, size = 2.9,
              hjust = 0.5, vjust = 0.5) +
    theme_void() +
    theme(plot.margin = margin(t = 0, b = 0, l = 0, r = 0))

  vPlots = c(list(rho_plot), vPlots)

  # Determine the number of empty plots needed to fill the row
  total_columns = length(colnames(mCorrs))
  num_empty_plots = total_columns - length(vPlots)
  empty_plots = replicate(num_empty_plots, ggplot() + theme_void(), simplify = FALSE)

  full_row = c(vPlots, empty_plots)
  # cat("Debug: DV =", dep_var_name,
  #     " | Number of plots:", length(vPlots) - 1,  # Exclude  rho plot
  #     " | Number of blank plots:", num_empty_plots, "\n")
  return(full_row)
}
GetAllIndVarPlotRows = function(mCorrs) {
  if (!is.matrix(mCorrs) || is.null(colnames(mCorrs))) {
    stop("mCorrs must be a named matrix.")
  }
  all_plot_rows = list()
  for (nColDV in 1:(ncol(mCorrs) - 1)) {
    current_row = GetIndVarPlotRow(mCorrs, nColDV)
    all_plot_rows = c(all_plot_rows, current_row)
  }
  return(all_plot_rows)
}
ArrangeAndResizeSquarePlots = function(plots, n_rows, n_cols) {
  # Use patchwork to combine plots
  plot_grid = patchwork::wrap_plots(plots, ncol = n_cols)
  plot_grid + patchwork::plot_layout(heights = rep(GetMaxSquarePlotDims(n_rows, n_cols), n_rows))
}
vAllDepVarPlotRows = GetAllIndVarPlotRows(mCorrs)
vNotIndVars = c('hp','qsec')
vCorrOrder01 = remove_from_vector(vCorrOrder,vNotIndVars)
mCorrsDepVarAndGoodIVs = remove_from_matrix(mCorrs, vNotIndVars)
vAllDepVarPlotRows01 = GetAllIndVarPlotRows(mCorrsDepVarAndGoodIVs)
mCorrsNoDepVar = remove_from_matrix(mCorrs, c('mpg'))

# FUNCTION TO MAKE CORRELATION GROUPS TBL DF ##################################
GetCorrGrps = function(mCorrs, Threshold = 0.7) {
```

```r
  # Convert correlation matrix to a distance matrix
  dist_mat = as.dist(1 - abs(mCorrs))
  # Perform hierarchical clustering
  hc = hclust(dist_mat, method = "complete")
  # Cut tree to form groups based on the threshold
  clusters = cutree(hc, h = 1 - Threshold)
  # Extract variable names
  vars = rownames(mCorrs)
  # Initialize a list to store each group's data
  cluster_groups = list()
  # Iterate through each unique cluster
  for (k in unique(clusters)) {
    members = vars[clusters == k]
    # Ensure there are at least two members to form correlations
    if (length(members) > 1) {
      # Create a matrix of all pairwise combinations
      combns = combn(members, 2, simplify = FALSE)
      # Extract correlations for each combination
      correlations = sapply(combns, function(pair) mCorrs[pair[1], pair[2]])
      # Collect information about pairs and their correlations
      pairs_info = paste(sapply(combns, paste, collapse = "/"),
                         sprintf("(%.2f)", correlations), collapse = "; ")
      # Create a data frame for this group
      df = data.frame(
        #group_id = paste("Group", k, sep = "_"),
        members = paste(members, collapse = ", "),
        correlations = pairs_info,
        stringsAsFactors = FALSE
      )
      # Store the data frame in the list
      cluster_groups[[k]] = df
    }
  }
  # Combine all group data frames into a single data frame
  if (length(cluster_groups) > 0) {
    final_df = do.call(rbind, cluster_groups)
  } else {
    final_df = data.frame(group_id = character(),
                          members = character(),
                          correlations = character(),
                          stringsAsFactors = FALSE)
  }
  return(final_df)
}
mCorrsNoDepVar = remove_from_matrix(mCorrs, c('mpg'))
dfCorrGrpsTbl = GetCorrGrps(mCorrsNoDepVar)
# FUNCTION TO COMPARE MODELS ##################################################
compare_two_models = function(model1, model2,
                              aic_threshold = 2,
                              bic_threshold = 6,
                              adj_r2_threshold = 0.01) {
  # Make sure input models are fitted model objects
  if (class(model1) != "lm" || class(model2) != "lm") {
    stop("Both inputs must be fitted model objects from 'lm' or similar.")
  }
  aic1 = AIC(model1)
```

```r
  aic2 = AIC(model2)
  bic1 = BIC(model1)
  bic2 = BIC(model2)
  adj_r21 = summary(model1)$adj.r.squared
  adj_r22 = summary(model2)$adj.r.squared
  aic_diff = abs(aic1 - aic2)
  bic_diff = abs(bic1 - bic2)
  adj_r2_diff = abs(adj_r21 - adj_r22)
  cat("Comparison of Two Models:\n")
  cat("Difference in AIC: ", aic_diff, "\n")
  cat("Difference in BIC: ", bic_diff, "\n")
  cat("Difference in Adjusted R-squared: ", adj_r2_diff, "\n")
  message = ifelse(
    aic_diff < aic_threshold && bic_diff < bic_threshold && adj_r2_diff < adj_r2_threshold,
    "\nBoth models perform similarly based on the criteria.\n",
    "\nThere is a significant difference between the models.\n"
  )
  cat(message)
}
# MODELS FOR COMPARISON ##################################################
full_model = lm(mpg ~ ., data = mtcars)
intercept_only = lm(mpg ~ 1, data = mtcars)
step_model = step(intercept_only,
                  direction = "both",
                  scope = formula(full_model),
                  trace = 0)
mdlWtCylHp = lm(mpg ~ wt + cyl + hp, data = mtcars)
mdlWtCyl = lm(mpg ~ wt + cyl,      data = mtcars)
predWtCylHp = predict(mdlWtCylHp, newdata = mtcars)
predWtCyl = predict(mdlWtCyl, newdata = mtcars)
comparison_df = data.frame(Car = rownames(mtcars),
                           Actual = mtcars$mpg,
                           mdlWtCylHp = predWtCylHp,
                           mdlWtCyl = predWtCyl)
rownames(comparison_df) = comparison_df$Car
comparison_df$Car = NULL  # remove Car column
```