# Advanced Data Analytics - Individual Assignment

[MARTINET Samuel - 12-603-726]

2025-02-17

**The dataset for the self-study is available from December 18, 2024 on Canvas. You can achieve up to 25 points for this self-study, which has a weight of 25% for the final grade.**

## Instructions:

You work for an environmental agency in the US. The environmental agency wants to develop a model to predict $CO_2$ emissions based on vehicle and fuel consumption specifications to have a better idea how to set regulations on future car models.

The file `co2_predict.Rdata` contains cars for which you are asked to deliver predictions of $CO_2$ emissions.

## Report:

### Analyzing the Data

*Are there any issues with the data (missing values, problematic labels, small groups in some categories, useless predictors, some values need to be recoded to capture what most probably happened, etc.)? Describe your data cleaning procedure.*

#### Exec summary

| Variable | Type | Remark |
|---|---|---|
| **CO2.Emissions.g.km** | **Numerical, continuous** | **Dependent variable** |
| Fuel.Consumption (x4) | Numerical, continuous | Keep only 1 of these, as there is near perfect colinearity. |
| Engine.Size.L | Numerical, continuous | Seems correlated to Cylinders, but kept it for now. |
| Cylinders | Numerical, discrete | Seems correlated to Engine.Size.L, but kept it for now.<br>Use Label Encoding, since the "categories" have a clear order and we want to preserve that ordering, yet some regrouping seems desirable.<br>Reduce number of categories to 4: x <=4, 4< x<= 6, 6<x<=8, x>8 |
| Make | Categorical | Eliminated because brand seems like a bad criteria for our goal to "set regulation on future car models". |
| Vehicle.Class | Categorical | Keep as is. Regrouping was considered but abandonned due to lack of domain expertise. |
| Transmission | Categorical | Reduced number of categories after performing domain research, due to very small number of observations on a number of categories, and loss of interpretability linked with having too many categories. |
| Fuel.Type | Categorical | Same. |

- **Missing Values:** I found no missing values (0 NA).
- **Duplicate Values:** I found 1493 duplicated rows.
- **Faulty data:** I found no evidence of faulty data. Although there are outliers for the numerical variables, these outliers do not seem inconsistent with the variance one would expect regarding eg. fuel consumption or engine size (some cars are just much bigger than others).
- **Irrelevant predictors - Predictors I eliminated:**
  - The variables Fuel.Consumption.Comb.L.100.km, Fuel.Consumption.Hwy.L.100.km, and Fuel.Consumption.City.L.100.km display a very high correlation. This nears perfect colinarity.
  - I considered making a generic combination of the three, or just keeping one. For interpretability, I decided to just keep Fuel.Consumption.Comb.L.100.km and discard the two others.
  - I then realised that Miles Per Gallon and Liters per 100 kilometers express the same thing, only in different units. I chose to discard the keep the Miles Per Gallon one since we work for an american agency.
  - Brand was eliminated: the stated aim of this analysis is to to "set regulation on future car models", and regulation is usually based on brand-agnostic attributes. Besides, a same brand can have models with very different emission levels, and this can change every year.
- **Categorical variables**
  - As detailed below, several categorical variables display very uneven frequency between categories. In order to avoid model instability, this asks for some countermeasures.
  - Since resampling was not an option, I decided to do some feature engineering and merged some categories together.
  - Transmission. For Manual, it's easy, the value 6 accounts for ± 80% of all observations. For other categories, there was more judgment involved. Easier for some categories than for others. A was harder, and I went for 2 categories.
  - I applied the "one-hot encoding" technique to the categorical variable after I defined these new categories.
- **Other remarks concerning the predictors:**
  - Engine size and number of cylinders are highly correlated (R^2 .92) but the dependency doesn't appear to be linear, so I decided to keep these two variables, at least for the first iterations of my model development.
  - By and large, the data appears very skewed ("data imbalance"): very few observations for some values, and a large number of observation for other values. This creates a tricky trade-off between taking advantage of the richness of the data and providing some generalizability and intepretability.
  - When it comes to cylinders, there are very few cars with more 10/12/16 cylinders. I lumped these into a new category: cars with more than 8 cylinders, and used the technique of Label Encoding.
- **Outliers: the numerical variables have outlier values.**
  - These do not seem to be erroneous values: the values are far from the Inter Quantile Range (IQR) but seem consistent with the values we would expect to see.
  - While outliers can cause issues (eg. distortion of results), I kept a conservative approach and only removed the most extreme outliers, while keeping the one nearest to the IQR.
  - We may consider additional measures (eg. use robust methods, more agressive censoring, transforming by eg. taking log transformation) later, if we have evidence that the presence of these remaining outliers is detrimental to our models' performance.

### Inspection of the data set

```r
# Load data and libraries
load("co2.Rdata")
library(ggplot2)
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
set.seed(123)  # for reproducibility
```

### Missing values & duplicated rows

```r
# Check for missing values in each column
missing_values_report <- co2 %>%
  summarise(across(everything(), ~ sum(is.na(.))))
# Duplicates
sum(duplicated(co2))
```

```
## [1] 1493
```

No missing values, 1493 duplicates.

### Faulty Data and Outliers

I used summary statistics (range, mean, . . . ), visualization via Box Plot charts.

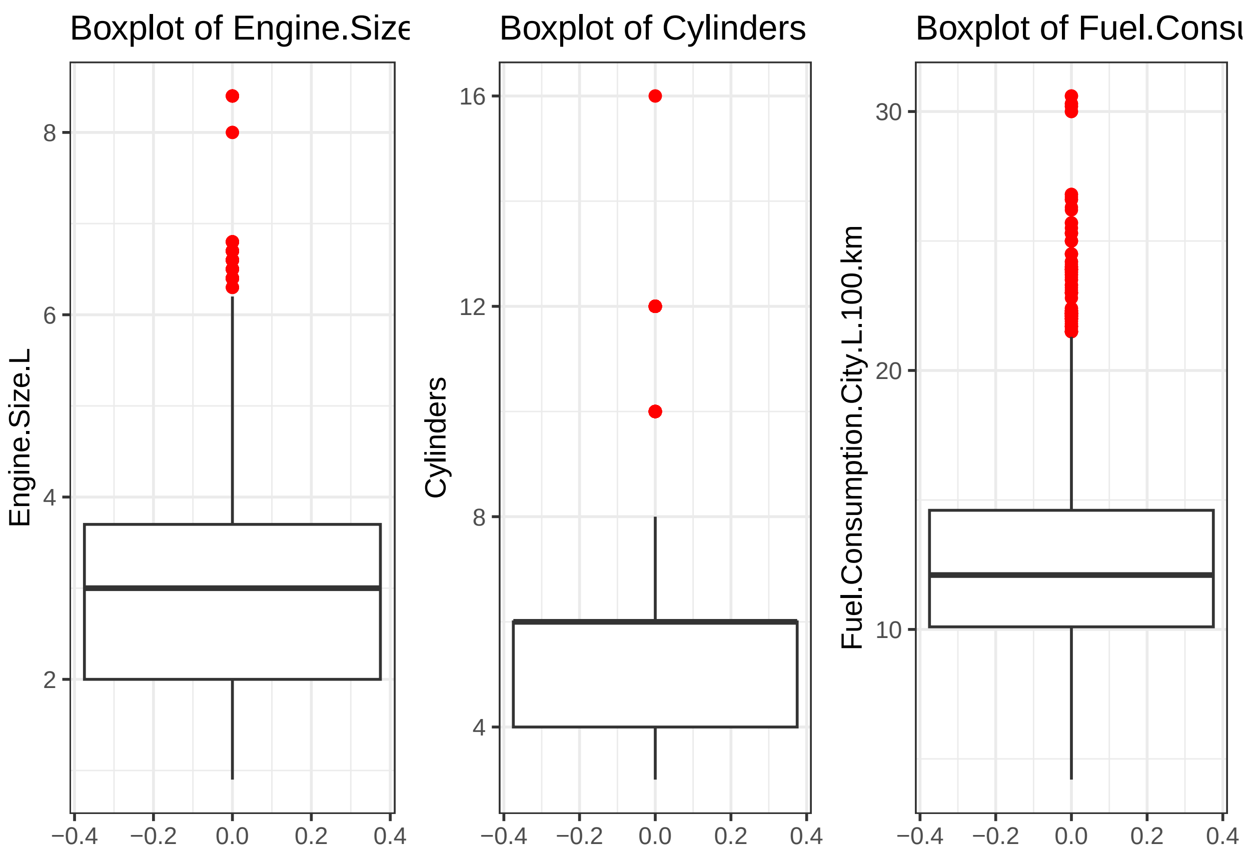
```r
str(co2)  # Check data types
summary(co2)
```

For example:

```r
if (!requireNamespace("gridExtra", quietly = TRUE)) {
  install.packages("gridExtra")
}
library("gridExtra")

num_cols <- sapply(co2, is.numeric)
# Function to create a boxplot with outliers highlighted
create_boxplot <- function(df, col_name) {
  ggplot(df, aes_string(y = col_name)) +
    geom_boxplot(outlier.colour = "red", outlier.shape = 16,
                    outlier.size = 2) +
    labs(title = paste("Boxplot of", col_name), y = col_name) +
    theme_bw() # Add a clean theme
}

# Create boxplots using the function and arrange them horizontally
library(gridExtra)
plot1 <- create_boxplot(co2, "Engine.Size.L")
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
plot2 <- create_boxplot(co2, "Cylinders")
plot3 <- create_boxplot(co2, "Fuel.Consumption.City.L.100.km")
grid.arrange(plot1, plot2, plot3, ncol=3)
```



We see some outliers in red (defined as points outside the range defined by the whiskers, 1.5 times the interquartile range above the third quartile or below the first quartile), but they are no evidence of faulty data.

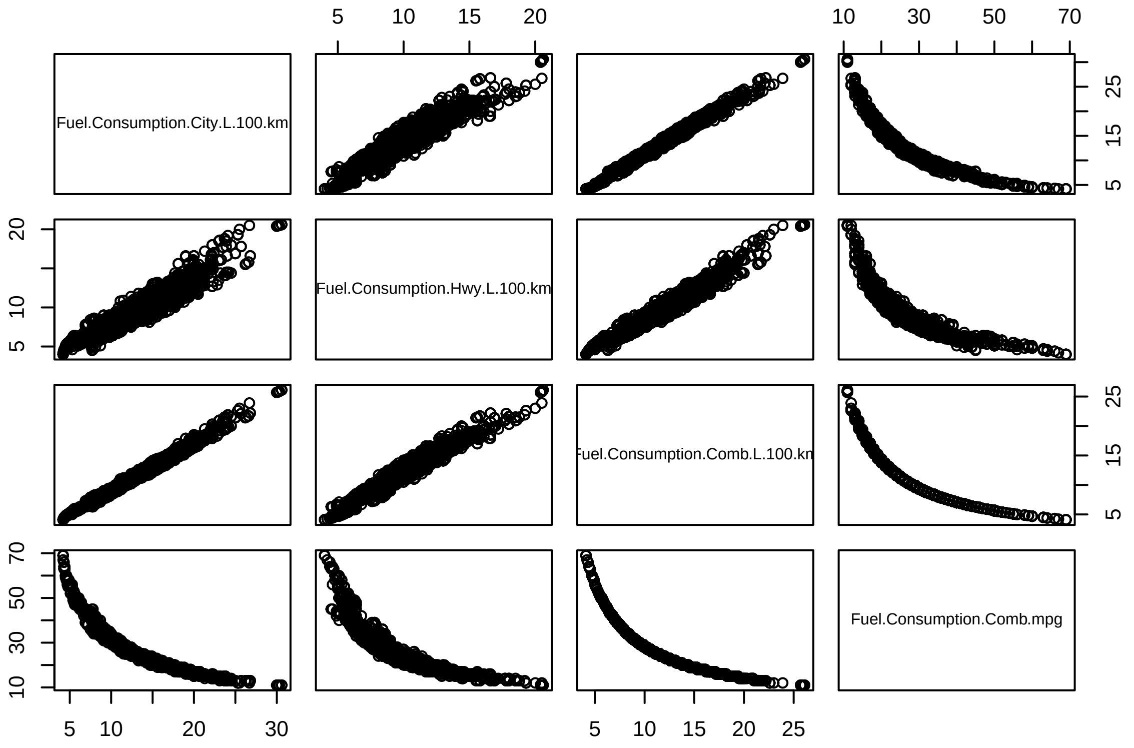
These outliers are dealt with in the next section via two methods:

- grouping (for Cylinders)
- censoring.

### Colinearity

The 4 consumption variable were very closely correlated.

```
consumption_cols <- names(co2)[grepl("Consumption", names(co2))]
pairs(co2[, consumption_cols])
```



Cylinders and engine size are correlated, and there are very few observations for Cylinders>8.

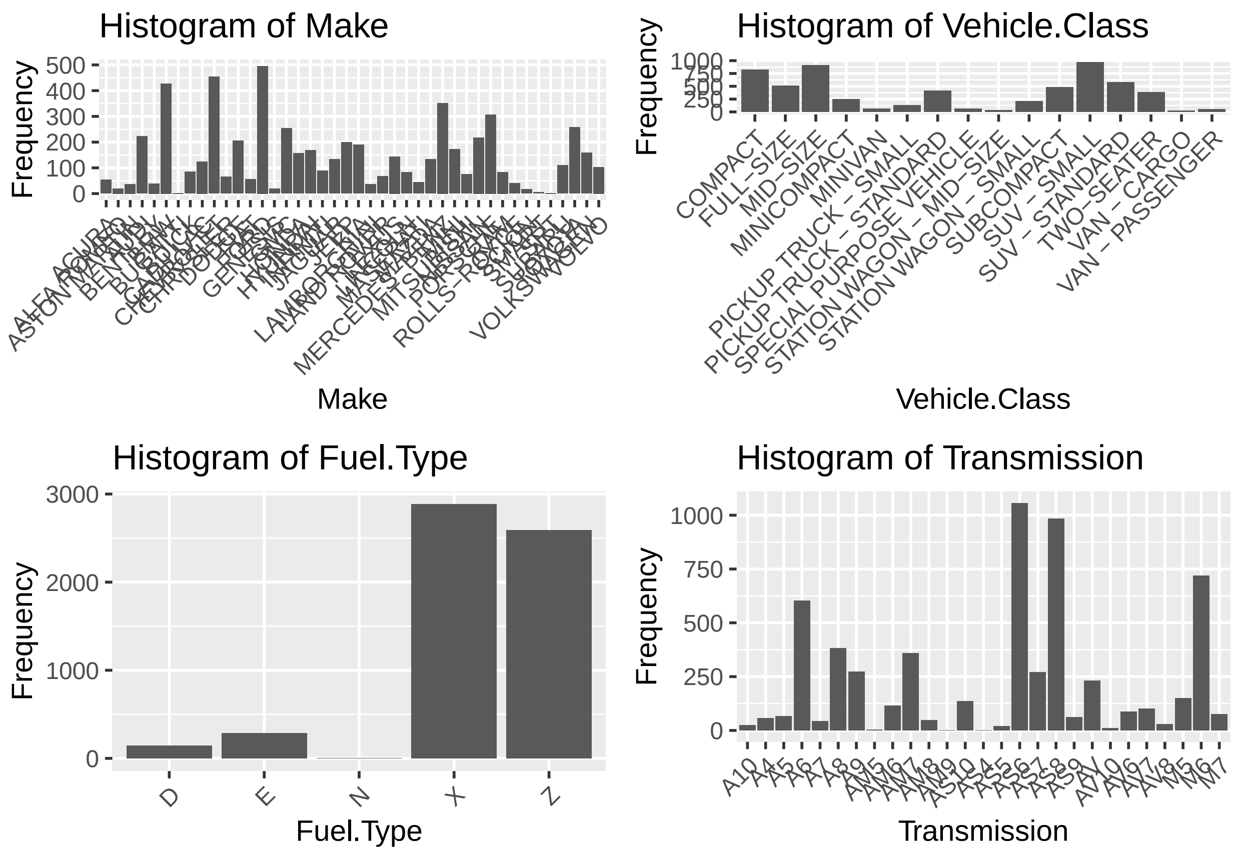
```
ggplot(co2, aes(x = Engine.Size.L, y = Cylinders)) + geom_point() + labs(title = "Relationship between
```

### Categorical variables

Using the next fragment, we could see that the categorical variables have very uneven distribution

```
# Categorial variables frequency distributions

create_histogram <- function(data, column_name) {
  ggplot(data, aes(x = .data[[column_name]])) +  # Use .data to access column by name
    geom_bar(stat = "count") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    labs(title = paste("Histogram of", column_name), x = column_name, y = "Frequency")
}

# Columns to create histograms for
columns <- c("Make", "Vehicle.Class", "Fuel.Type", "Transmission")

# Create a list to store the plots
plots <- list()

# Loop through the columns and create histograms
for (col in columns) {
  plots[[col]] <- create_histogram(co2, col)
```

```r
}

# Arrange the plots in a grid
do.call(grid.arrange, c(plots, ncol = 2, nrow = 2))
```



Fuel is an extreme example, with >5000 observations with gazoline (categories Z + X), but only 1 observation for natural gaz.

It's also hard to interprete the values taken by eg. transmission, given there are so many of them. There is a trade off between the richness of our data and its intepretability. I decided to keep flexibility to adapt our data to different statistical approaches (eg. regression vs. deep neural network) and not be too aggressive here in terms of regrouping categories.

**Data cleaning**

The steps we took are, in order towards cleaning the data:

- remove duplicates
- remove the variables Make and 3 fuel Consumption variables
- remove outliers for the variable Engine.Size.L
- shuffle the data

All the code follows.

**Data transformation**

We went a step further and performed some feature engineering thought to be beneficial for all the models we look at in the next section. Specifically, we did

- some feature engineering by redefining categories for the variables Cylinders, Fuel type; based on:
  - the frequency within each category and,
  - some semantic interpretation of each category
- use one-hot encoding for categorical variables
- standardize numerical variables

Finally, while redesigning the grouping of Transmission and Vehicle class, lack of domain expertise made this difficult. We therefore kept these variable as is for now.

**Code**

```r
# Load libraries and data
if (!requireNamespace("ggplot2", quietly = TRUE)) {
  install.packages("ggplot2")
}
library(ggplot2)
if (!requireNamespace("tidyr", quietly = TRUE)) {
  install.packages("tidyr")
}
library(tidyr)
if (!requireNamespace("dplyr", quietly = TRUE)) {
  install.packages("dplyr")
}
library(dplyr)
load("co2.Rdata")

# Remove duplicated rows in the dataframe co2_data
co2_data <- co2 |> distinct()

# New grouping for Fuel.Type
co2_data <- co2_data %>%
  mutate(Fuel_group = case_when(
    Fuel.Type == "D" ~ "Diesel",
    Fuel.Type == "X" ~ "Regular",
    Fuel.Type == "Z" ~ "Premium",
    TRUE ~ "Other"
  ))

# New grouping for cylinders
co2_data <- co2_data %>%
  mutate(Cylinder_group = case_when(
    Cylinders <= 4 ~ 1,
    Cylinders > 4 & Cylinders <= 6 ~ 2,
    Cylinders > 6 & Cylinders <= 8 ~ 3,
    TRUE ~ 4
  ))

# New grouping for transmission
co2_data <- co2_data %>%
  mutate(Transmission_group = case_when(
    Transmission %in% c("A10", "A4", "A5", "A6", "A7") ~ "A1",
    Transmission %in% c("A8", "A9") ~ "A2",
    Transmission %in% c("AM5", "AM6", "AM7", "AM8", "AM9") ~ "AM",
    Transmission %in% c("AS4", "AS5", "AS6") ~ "AS1",
    Transmission %in% c("AS7", "AS8", "AS9", "AS10") ~ "AS2",
    Transmission %in% c("AV10", "AV6", "AV7", "AV8") ~ "AV",
    Transmission %in% c("M5", "M6", "M7") ~ "M",
    TRUE ~ "OTHER"
  ))

# Remove the following variables: Fuel.Consumption.City.L.100.km, Fuel.Consumption.Hwy.L.100.km, Fuel.C
co2_data <- co2_data |>
  select(-Fuel.Consumption.City.L.100.km, -Fuel.Consumption.Hwy.L.100.km, -Fuel.Consumption.Comb.L.100.

# Convenience function
one_hot_encoding <- function(df, columns = "season") {
  # Create a copy of the original data.frame to avoid modifying it
  df_copy <- df %>% as.data.frame()
  # Ensure columns is a character vector
  columns <- as.character(columns)
  # Loop through each specified column
  for (column in columns) {
    # Get unique values and remove the first one for reference
    unique_values <- unique(df_copy[[column]])
    non_reference_values <- unique_values[-1]
    # Loop through non-reference values to create dummy variables
    for (value in non_reference_values) {
      new_col_name <- paste0(column, ".", value)
      df_copy[[new_col_name]] <- as.numeric(df_copy[[column]] == value)
    }
    # Remove the original column
    df_copy[[column]] <- NULL
  }
  return(df_copy)
}

# One-hot encoding of categorical variables Transmission_group and Fuel_group
co2_data <- one_hot_encoding(co2_data, c("Transmission_group", "Fuel_group", "Vehicle.Class"))

# Remove outliers from the variable Engine.Size.L
Q1 <- quantile(co2_data$Engine.Size.L, 0.25)
Q3 <- quantile(co2_data$Engine.Size.L, 0.75)
IQR <- IQR(co2_data$Engine.Size.L)
lower_bound <- Q1 - 1.5 * IQR
upper_bound <- Q3 + 1.5 * IQR
co2_data <- co2_data[co2_data$Engine.Size.L >= lower_bound & co2_data$Engine.Size.L <= upper_bound, ]

# standardization of numerical variable Emissions, Consumption
co2_data[,c("Fuel.Consumption.Comb.mpg")] <- scale(co2_data[,c("Fuel.Consumption.Comb.mpg")])

# shuffle
co2_data <- co2_data[sample(nrow(co2_data)), ]
```

### Grouping

I grouped variables whenever I assessed that doing so could reduce model complexity or improve model generalization.

Several categorical variables in the data set have very uneven frequency between categories. To tackle this, since resampling was not an option, I decided to do some feature engineering and merged some categories together. I grouped categories by considering similarity and relative frequency.

Let's take the example of Transmission. For Manual, it's easy, the distribution is centered around the value 6 (which accounts for ± 80% of all observations). For other categories, there was more judgment involved. Easier for some categories than for others. Transmission A was harder, and I went for 2 categories.

Despite having analyzed the US EPA Size Class (https://en.wikipedia.org/wiki/Car_classification) classification, lack of domain expertise made it difficult to group the Class categorical variable, so I kept it as is. This categorical varialbe is difficult to interpret because the mapping to weight/displacement is ambiguous.

### One hot encoding

I applied the "one-hot encoding" technique to the categorical variable after I defined these new categories.

## Applied Approaches/Methods

*Code the 3 approaches, find the best predictive model among them and answer the following questions such that the R code is well-explained. Did you partition the data? Did you further transform variables in training and test data (beyond the steps in the data cleaning section)? Which estimation methods did you use? Describe the methods briefly. How did you select the tuning parameters (if applicable)?*

- *Which estimation methods did you use? Describe the methods briefly.*
- *How did you select the tuning parameters (if applicable)?*

### Variable transformation and Data partition

Variable transformation was done in the previous section.

We partition the data into a training and a testing set, with a 90/10 ratio. Having shuffled the data earlier, we can assert that this split was random.

```r
split_value <- 0.9
# Split data into training and testing sets
index <- sample(1:nrow(co2_data), split_value * nrow(co2_data))
train_data <- co2_data[index, ]
test_data <- co2_data[-index, ]
```

### Approach 1 - Multiple linear regression

We started with a multiple linear regression (MLR). Multiple linear assumes a linear relationship between the variables and aims to find the best-fitting line (or hyperplane) that describes this relationship.

### Build the model

```r
# Multiple linear regression
model <- lm(CO2.Emissions.g.km ~ ., data = train_data)

# Make predictions
train_pred_mlr <- predict(model, newdata = train_data)
test_pred_mlr <- predict(model, newdata = test_data)

# Calculate MSE
```

```r
train_mse_mlr <- mean((train_data$CO2.Emissions.g.km - train_pred_mlr)^2)
test_mse_mlr <- mean((test_data$CO2.Emissions.g.km - test_pred_mlr)^2)
```

**Analysis**

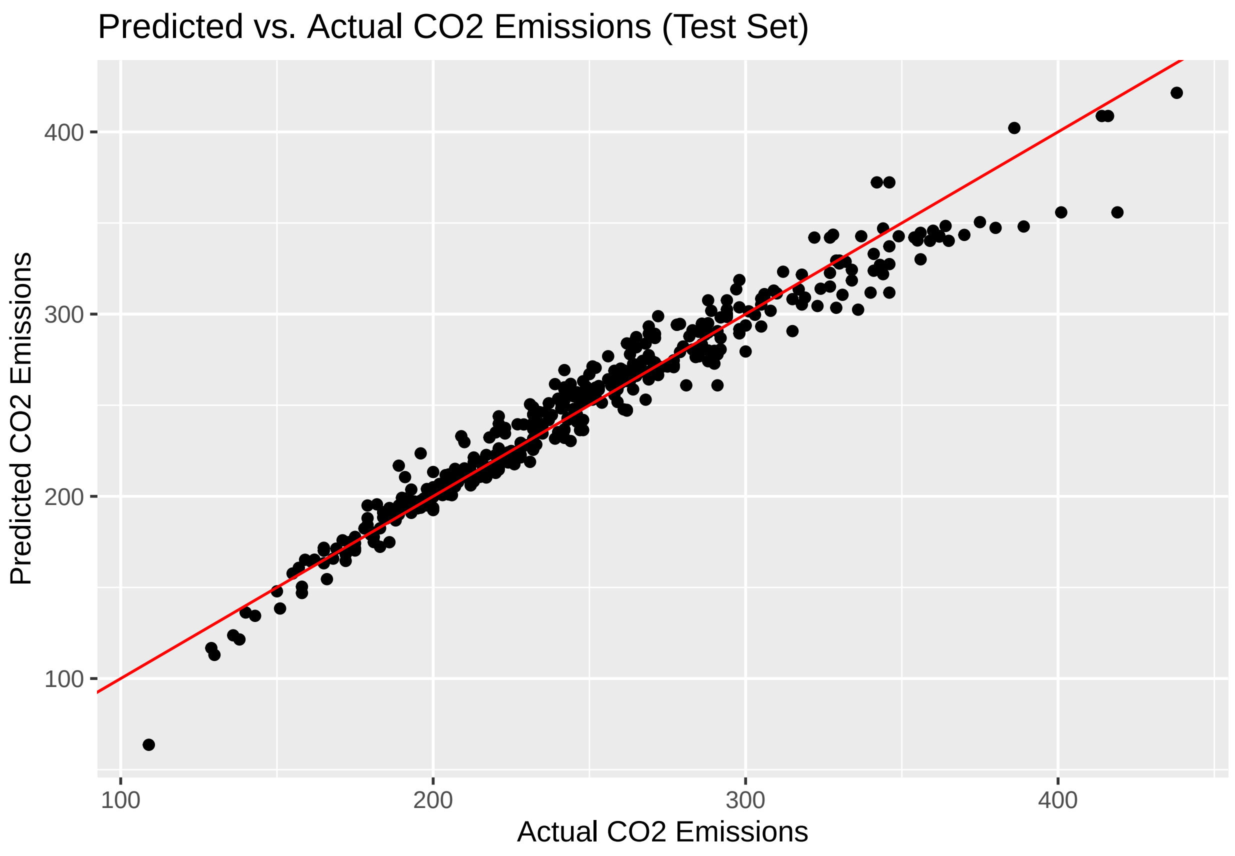
```r
cat("Training MSE:", train_mse_mlr, "\n")
```

```
## Training MSE: 169.0374
```

```r
cat("Testing MSE:", test_mse_mlr, "\n")
```

```
## Testing MSE: 132.5514
```

```r
# Create a plot of predictions vs. actual values for the test set
plot_data <- data.frame(
  Predicted = test_pred_mlr,
  Actual = test_data$CO2.Emissions.g.km
)
ggplot(plot_data, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +  # Add a 45-degree line
  labs(title = "Predicted vs. Actual CO2 Emissions (Test Set)",
       x = "Actual CO2 Emissions",
       y = "Predicted CO2 Emissions")
```



For the bulk of observations, MLR seems do be doing a very good job. There is more distance at the extremities, but there are very few data points there anyways. So for most data point, the MLR does a good job.

The MSEs are:

- Training MSE: 164,4
- **Testing MSE: 174.8**

MLR has the advantage of being easy to interprete.

```
summary(model)
```

This assignment is about predicting, not explaining, but from the output of this function, we notice nevertheless that fuel consumption and type of fuel seem to matter a lot. Also, a number of coefficients are not statistically significant. This suggest trying to use a regularization procedure.

### Approach 2 - LASSO

We went for the LASSO regularization procedure.

```
# Load necessary library
if(!require(glmnet)){install.packages("glmnet")}
```

```
## Loading required package: glmnet

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-8
```

```
library(glmnet)
# Prepare the data for glmnet
x <- model.matrix(CO2.Emissions.g.km ~ ., data = train_data)[,-1] #Exclude intercept
y <- train_data$CO2.Emissions.g.km
# Fit the lasso model
lasso_model <- glmnet(x, y, alpha = 1) #alpha = 1 for lasso
# Cross-validation to find optimal lambda
cv_lasso <- cv.glmnet(x, y, alpha = 1)
# Best lambda
best_lambda <- cv_lasso$lambda.min
# Fit lasso model with best lambda
final_lasso <- glmnet(x, y, alpha = 1, lambda = best_lambda)
# Coefficients of the final lasso model
coef(final_lasso)
```

```
## 29 x 1 sparse Matrix of class "dgCMatrix"
##                                             s0
## (Intercept)                        207.4402849
## Engine.Size.L                        6.2068105
## Fuel.Consumption.Comb.mpg          -39.1254706
## Cylinder_group                      12.4374545
## Transmission_group.AS2              -3.8294556
## Transmission_group.A2               -6.2585446
## Transmission_group.AM                2.5795871
## Transmission_group.A1               -0.5016524
## Transmission_group.AS1              -0.5780159
## Transmission_group.OTHER                       7.5793320
## Transmission_group.AV                         -7.8327771
## Fuel_group.Regular                            -2.3138058
## Fuel_group.Other                             -54.1005099
## Fuel_group.Diesel                             18.6541495
## `Vehicle.Class.FULL-SIZE`                      1.3822802
## `Vehicle.Class.SUV - SMALL`                    3.6432084
## Vehicle.Class.COMPACT                         -0.3077670
## `Vehicle.Class.MID-SIZE`                      -0.1030729
## `Vehicle.Class.STATION WAGON - SMALL`          1.4290766
## `Vehicle.Class.SPECIAL PURPOSE VEHICLE`        5.9959929
## `Vehicle.Class.PICKUP TRUCK - SMALL`          14.9385842
## `Vehicle.Class.SUV - STANDARD`                17.4490142
## Vehicle.Class.MINICOMPACT                     -6.7803195
## `Vehicle.Class.PICKUP TRUCK - STANDARD`       12.9885174
## `Vehicle.Class.TWO-SEATER`                     3.2636951
## `Vehicle.Class.VAN - PASSENGER`               73.5877023
## Vehicle.Class.MINIVAN                          2.0002507
## `Vehicle.Class.STATION WAGON - MID-SIZE`       .
## `Vehicle.Class.VAN - CARGO`                   44.3635567
```

```
# Make predictions on the test set using the lasso model
x_test <- model.matrix(CO2.Emissions.g.km ~ ., data = test_data)[,-1]
lasso_pred <- predict(final_lasso, newx = x_test)
# Calculate MSE for the lasso model
lasso_mse <- mean((test_data$CO2.Emissions.g.km - lasso_pred)^2)
cat("Lasso Testing MSE:", lasso_mse, "\n")
```

```
## Lasso Testing MSE: 132.6378
```

**The test MSE after doing LASSO is: 174.9.** We see that we are able to remove a number of parameters at virtually no cost.

We could have optimized the procedure's parameters even more, but decided to use a non-linear approach instead.

### Approach 3 - Neural Network

Given the potential for non-linearities, we decided to use a neural network.

#### Shallow network

```
if (!requireNamespace("devtools", quietly = TRUE)) {
  install.packages("devtools")
}
library(devtools)
```

```
## Loading required package: usethis
```

```
if (!requireNamespace("keras", quietly = TRUE)) {
  devtools::install_github("rstudio/keras")
}
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v forcats   1.0.0     v readr     2.1.5
## v lubridate 1.9.4     v stringr   1.5.1
## v purrr     1.0.4     v tibble    3.2.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x gridExtra::combine() masks dplyr::combine()
## x Matrix::expand()     masks tidyr::expand()
## x dplyr::filter()      masks stats::filter()
## x dplyr::lag()         masks stats::lag()
## x Matrix::pack()       masks tidyr::pack()
## x Matrix::unpack()     masks tidyr::unpack()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(tensorflow)
```

```
##
## Attaching package: 'tensorflow'
##
## The following object is masked from 'package:caret':
##
##     train
```

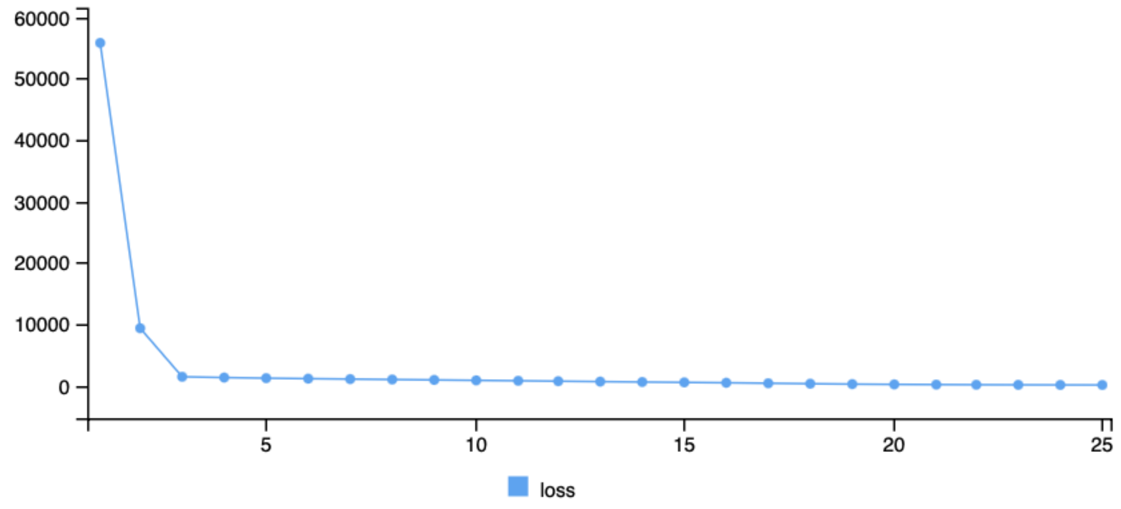
```r
library(keras)
install_keras()
```

```
## Virtual environment 'r-tensorflow' removed.
## Using Python: /opt/python/3.10.12/bin/python3.10
## Creating virtual environment 'r-tensorflow' ...

## + /opt/python/3.10.12/bin/python3.10 -m venv /cloud/project/r-tensorflow

## Done!
## Installing packages: pip, wheel, setuptools

## + /cloud/project/r-tensorflow/bin/python -m pip install --upgrade pip wheel setuptools

## Virtual environment 'r-tensorflow' successfully created.
## Using virtual environment 'r-tensorflow' ...

## + /cloud/project/r-tensorflow/bin/python -m pip install --upgrade --no-user 'tensorflow==2.15.*' ten

##
## Installation complete.
```

```r
library(mlbench)
```

```r
# Prepare the data
x <- as.matrix(train_data[, -which(names(train_data) == "CO2.Emissions.g.km")])
y <- train_data$CO2.Emissions.g.km

# Define the model
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = ncol(x)) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1)

# Compile the model
model %>% compile(
  loss = "mse",
  optimizer = "adam",
  metrics = "mse"
)

# Train the model
model %>% fit(x, y, epochs = 15, batch_size = 32)
```

```
## Epoch 1/15
## 122/122 - 2s - loss: 52778.1797 - mse: 52778.1797 - 2s/epoch - 12ms/step
## Epoch 2/15
## 122/122 - 0s - loss: 5686.9058 - mse: 5686.9058 - 213ms/epoch - 2ms/step
## Epoch 3/15
## 122/122 - 0s - loss: 1366.3390 - mse: 1366.3390 - 200ms/epoch - 2ms/step
## Epoch 4/15
## 122/122 - 0s - loss: 1227.9166 - mse: 1227.9166 - 186ms/epoch - 2ms/step
## Epoch 5/15
## 122/122 - 0s - loss: 1121.5164 - mse: 1121.5164 - 183ms/epoch - 1ms/step
## Epoch 6/15
## 122/122 - 0s - loss: 1033.8992 - mse: 1033.8992 - 187ms/epoch - 2ms/step
## Epoch 7/15
## 122/122 - 0s - loss: 952.4622 - mse: 952.4622 - 189ms/epoch - 2ms/step
## Epoch 8/15
## 122/122 - 0s - loss: 877.9473 - mse: 877.9473 - 152ms/epoch - 1ms/step
## Epoch 9/15
## 122/122 - 0s - loss: 799.2244 - mse: 799.2244 - 158ms/epoch - 1ms/step
## Epoch 10/15
## 122/122 - 0s - loss: 723.4434 - mse: 723.4434 - 183ms/epoch - 1ms/step
## Epoch 11/15
## 122/122 - 0s - loss: 647.7353 - mse: 647.7353 - 187ms/epoch - 2ms/step
## Epoch 12/15
## 122/122 - 0s - loss: 576.2727 - mse: 576.2727 - 180ms/epoch - 1ms/step
## Epoch 13/15
## 122/122 - 0s - loss: 508.5963 - mse: 508.5963 - 147ms/epoch - 1ms/step
## Epoch 14/15
## 122/122 - 0s - loss: 446.1802 - mse: 446.1802 - 189ms/epoch - 2ms/step
## Epoch 15/15
## 122/122 - 0s - loss: 390.3623 - mse: 390.3623 - 187ms/epoch - 2ms/step
```



The fitting process converges after 10 epochs, so we reduced this number from 100 to 15 above.

```
# Make predictions on the training and test data
train_predictions <- model %>% predict(as.matrix(train_data[, -which(names(train_data) == "CO2.Emission
```

```
## 122/122 - 0s - 150ms/epoch - 1ms/step
```

```
test_predictions <- model %>% predict(as.matrix(test_data[, -which(names(test_data) == "CO2.Emissions.g
```

```
## 14/14 - 0s - 22ms/epoch - 2ms/step
```

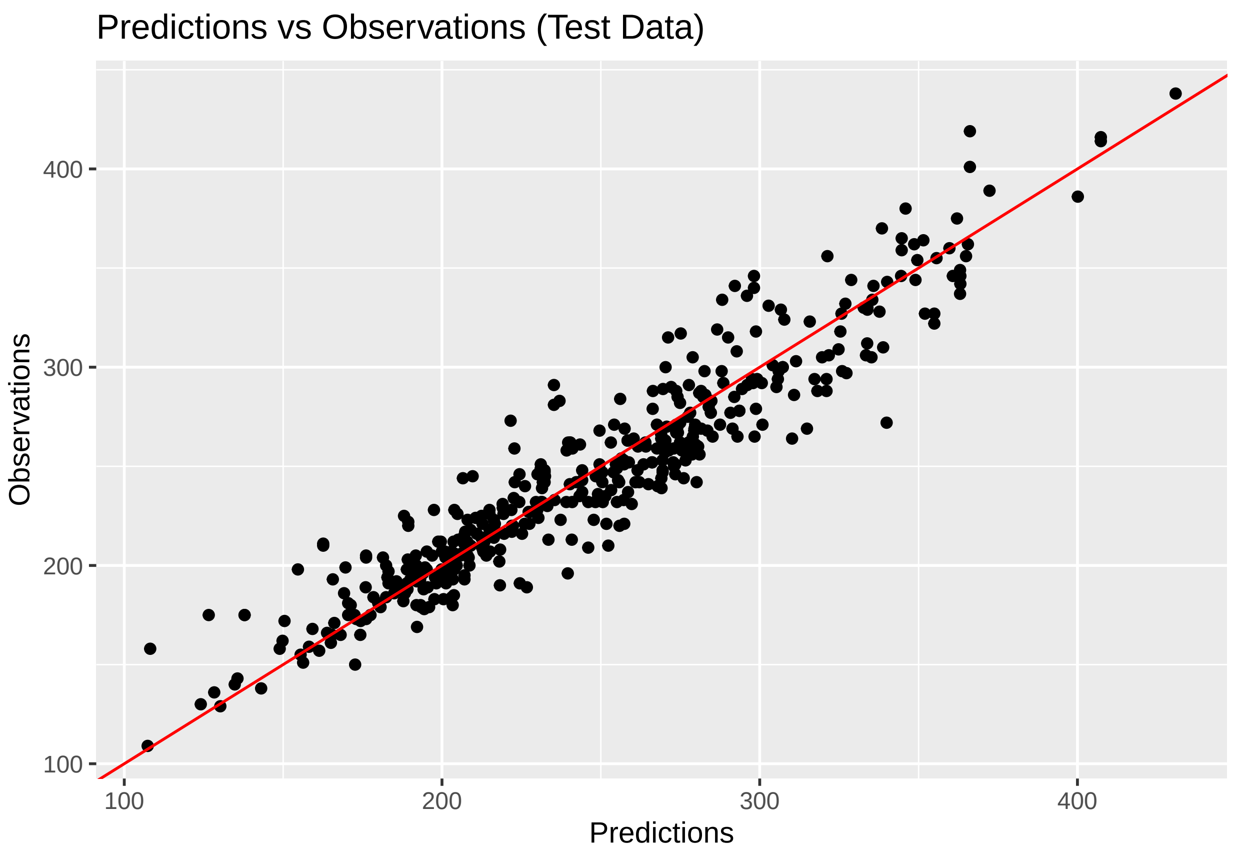
```
# Calculate the MSE for the training data
train_mse <- mean((train_predictions - train_data$CO2.Emissions.g.km)^2)
cat("Training MSE:", train_mse, "\n")
```

```
## Training MSE: 363.0716
```

```
# Calculate the MSE for the test data
test_mse <- mean((test_predictions - test_data$CO2.Emissions.g.km)^2)
cat("Test MSE:", test_mse, "\n")
```

```
## Test MSE: 349.1505
```

```
# Create a data frame for plotting
plot_data <- data.frame(
  Prediction = as.vector(test_predictions),
  Observation = test_data$CO2.Emissions.g.km
)

# Plot the predictions vs observations
ggplot(plot_data, aes(x = Prediction, y = Observation)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red") +
  labs(title = "Predictions vs Observations (Test Data)", x = "Predictions", y = "Observations")
```



This time, we get what seems to be excellent results despite the very simple architecture of our network:

- **Training MSE: 45.1**
- **Test MSE: 52.7**

We tried more complex architectures but with very small gains only.

## Summary - Preferred Approach

*What is the preferred approach? How did you select the method/approach for your preferred specification?*

It's a trade-off between interpretability and predictive power. The MLR is performing well and gives us interesting insight about the importance of each variable. But the neural network is even better at predicting.

My preferred approach has been the simple Neural Network used above. It performed best at predicting. I selected it because I think there was no big reason to assume linearity, and because we had enough data points.

*Have you tried additional ways to select the tuning parameters and/or partition the sample to confirm the best predictive model?*

I tried four things:

- make the NN deeper (adding layers)
- make each layer bigger
- try other optimizing methods (gradient descent)
- change the activation function away from RELU

But the gains do not seem significant, so I decided to keep the model as "standard" as it gets.

**Strangely, what really made the biggest difference is how I preprocessed the variables (esp. how I grouped them), for both generalizability and speed of convergence.**

## Save predictions

The code used for making the predictions. Compared to code above, when training the model, we use the entirety of data available to us (no more test/dev split)

**Note that I have 1456, not 1477 predictions as indicated in the assignment handout.**

**Preprocessing:**

```r
process_co2_data <- function(co2_data) {
  # New grouping for Fuel.Type
  co2_data <- co2_data %>%
    mutate(Fuel_group = case_when(
      Fuel.Type == "D" ~ "Diesel",
      Fuel.Type == "X" ~ "Regular",
      Fuel.Type == "Z" ~ "Premium",
      TRUE ~ "Other"
    ))

  # New grouping for cylinders
  co2_data <- co2_data %>%
    mutate(Cylinder_group = case_when(
      Cylinders <= 4 ~ 1,
      Cylinders > 4 & Cylinders <= 6 ~ 2,
      Cylinders > 6 & Cylinders <= 8 ~ 3,
      TRUE ~ 4
    ))

  # New grouping for transmission
  co2_data <- co2_data %>%
    mutate(Transmission_group = case_when(
      Transmission %in% c("A10", "A4", "A5", "A6", "A7") ~ "A1",
      Transmission %in% c("A8", "A9") ~ "A2",
      Transmission %in% c("AM5", "AM6", "AM7", "AM8", "AM9") ~ "AM",
      Transmission %in% c("AS4", "AS5", "AS6") ~ "AS1",
      Transmission %in% c("AS7", "AS8", "AS9", "AS10") ~ "AS2",
      Transmission %in% c("AV10", "AV6", "AV7", "AV8") ~ "AV",
      Transmission %in% c("M5", "M6", "M7") ~ "M",
      TRUE ~ "OTHER"
    ))

  # Remove the following variables: Fuel.Consumption.City.L.100.km,
  # Fuel.Consumption.Hwy.L.100.km, Fuel.Consumption.Comb.L.100.km , Make,
  # Cylinders, Fuel.Type
  co2_data <- co2_data |>
    select(-Fuel.Consumption.City.L.100.km, -Fuel.Consumption.Hwy.L.100.km,
           -Fuel.Consumption.Comb.L.100.km, -Make, -Cylinders, -Fuel.Type, -Transmission)

  # One-hot encoding of categorical variables Transmission_group and Fuel_group
  co2_data <- one_hot_encoding(co2_data, c("Transmission_group", "Fuel_group", "Vehicle.Class"))

  # Remove outliers from the variable Engine.Size.L
  Q1 <- quantile(co2_data$Engine.Size.L, 0.25)
  Q3 <- quantile(co2_data$Engine.Size.L, 0.75)
  IQR <- IQR(co2_data$Engine.Size.L)
  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR
  co2_data <- co2_data[co2_data$Engine.Size.L >= lower_bound & co2_data$Engine.Size.L <= upper_bound, ]

  # standardization of numerical variable Emissions, Consumption
  co2_data[, c("Fuel.Consumption.Comb.mpg")] <- scale(co2_data[, c("Fuel.Consumption.Comb.mpg")])

  return(co2_data)
}

# Convenience function (as provided in the original code)
one_hot_encoding <- function(df, columns = "season") {
  # Create a copy of the original data.frame to avoid modifying it
  df_copy <- df %>% as.data.frame()
  # Ensure columns is a character vector
  columns <- as.character(columns)
  # Loop through each specified column
  for (column in columns) {
    # Get unique values and remove the first one for reference
    unique_values <- unique(df_copy[[column]])
    non_reference_values <- unique_values[-1]
    # Loop through non-reference values to create dummy variables
    for (value in non_reference_values) {
      new_col_name <- paste0(column, ".", value)
      df_copy[[new_col_name]] <- as.numeric(df_copy[[column]] == value)
    }
    # Remove the original column
    df_copy[[column]] <- NULL
  }
  return(df_copy)
}
```

**Data import and pre-processing:**

```r
load("co2_predict.Rdata")
load("co2.Rdata")

co2_predict_data <-process_co2_data(co2_predict)
co2_data <-process_co2_data(co2)
```

**Model:**

```r
# Prepare the data
x <- as.matrix(co2_data[, -which(names(co2_data) == "CO2.Emissions.g.km")])
y <- co2_data$CO2.Emissions.g.km

# Define the model
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = ncol(x)) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1)
```

```
# Compile the model
model %>% compile(
  loss = "mse",
  optimizer = "adam",
  metrics = "mse"
)

# Train the model
model %>% fit(x, y, epochs = 15, batch_size = 32)
```

**Predict and save:**

```
predict_set_predictions <- model %>% predict(as.matrix(co2_predict_data))

if (!requireNamespace("openxlsx", quietly = TRUE)) {
  install.packages("openxlsx")
}

write.csv(predict_set_predictions,"samuel_martinet.csv", row.names = FALSE)
```