Applying Data Science Skills to Metal Content of Consumer Products Tested Dataset from the NYC Health Department

Le' Sean Roberts (NYC Ofiice of Managment and Budget College Aide)

Summary

This study presents an analysis of laboratory results for consumer products tested by the New York City Department of Health and Mental Hygiene, focusing on lead, arsenic, and mercury content. The dataset comprises products collected during investigations of lead poisonings and store surveys. Through exploratory data analysis and machine learning techniques, we aim to uncover patterns, trends, and potential risks associated with these metals in consumer products.

Introduction

Lead, arsenic, and mercury are toxic metals that can pose significant health risks, particularly when present in consumer products. The New York City Department of Health and Mental Hygiene conducts rigorous testing of such products to safeguard public health. Understanding the prevalence and distribution of these metals in consumer goods is crucial for identifying potential sources of exposure and implementing targeted interventions.

In this study, we analyze a dataset containing laboratory results of tests conducted on various consumer products. These products were collected during investigations of lead poisonings and store surveys across New York City. Our objective is to utilize exploratory data analysis (EDA) techniques and machine learning (ML) algorithms to explore the dataset, identify patterns, and develop predictive models to aid in risk assessment and mitigation strategies.

Data Cleaning

Before diving into analysis, it's imperative to observe and clean the dataset by handling missing values.

```r
library(readr)
Metal_Content_of_Consumer_Products_NYC_Health_Department <-
  read_csv("C:/Users/verlene/Downloads/Metal_Content_of_Consumer_Products_Tested_by_the_NY
```

```r
library(tidyverse)
glimpse(Metal_Content_of_Consumer_Products_NYC_Health_Department)
```

```
Rows: 6,893
Columns: 10
$ ROW_ID          <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,~
$ PRODUCT_TYPE    <chr> "Food-Spice", "Food-Spice", "Food-Spice", "Food-Spice"~
$ PRODUCT_NAME    <chr> "Turmeric powder", "Cumin powder", "Cumin seeds", "Mas~
$ METAL           <chr> "Lead", "Lead", "Lead", "Lead", "Lead", "Lead", "Lead"~
$ CONCENTRATION   <dbl> 2.9, 2.4, 2.8, 12.0, -1.0, -1.0, 580000.0, -1.0, 41000~
$ UNITS           <chr> "ppm", "ppm", "ppm", "ppm", "ppm", "ppm", "ppm", "ppm"~
$ MANUFACTURER    <chr> "UNKNOWN OR NOT STATED", "UNKNOWN OR NOT STATED", "UNK~
$ MADE_IN_COUNTRY <chr> "INDIA", "INDIA", "INDIA", "UNKNOWN OR NOT STATED", "U~
$ COLLECTION_DATE <chr> "01/04/2011 12:00:00 AM", "01/04/2011 12:00:00 AM", "0~
$ DELETED         <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
```

Observed prior are the columns and the type of variables they are classified as. After such observation will clean the data to acquire variables or columns of interest and eliminate instances of NA and "UNKOWN OR NOT STATED" throughout the variables.

```r
# Dropping rows not meaningful to development, eliminating NAs, and dropping entries of "U
relevant_columns <-
  Metal_Content_of_Consumer_Products_NYC_Health_Department |>
  select(-ROW_ID, -UNITS, -COLLECTION_DATE, -DELETED) |>
  filter(MANUFACTURER != "UNKNOWN OR NOT STATED" ,
         MADE_IN_COUNTRY != "UNKNOWN OR NOT STATED") |>
  rename("CONCENTRATION (PPM)" = CONCENTRATION) |>
  na.omit()
```

```r
glimpse(relevant_columns)
```

```
Rows: 1,965
Columns: 6
$ PRODUCT_TYPE          <chr> "Food-Candy", "Dietary Supplement/Medications/Re~
$ PRODUCT_NAME          <chr> "Lucas Muecas candy", "Pregnita (Ayurvedic)", "C~
$ METAL                 <chr> "Lead", "Lead", "Lead", "Lead", "Lead", "Lead", ~
$ `CONCENTRATION (PPM)` <dbl> -1.0e+00, 1.2e+04, 2.7e+00, -1.0e+00, -1.0e+00, ~
$ MANUFACTURER          <chr> "Lucas Muecas", "Ajmera Pharmaceuticals PVT. LTD~
$ MADE_IN_COUNTRY       <chr> "MEXICO", "INDIA", "INDIA", "DOMINICAN REPUBLIC"~
```

NYC is a major importer of products and goods from various parts of the world. Hence, for the "PRODUCT NAME" and "MANUFACTURER" features it's important to observe the unique counts since both are character features, where totals may be too abundant for such features to be practical with numerical recoding.

```
# Issue with excessive numeric instances in the MANUFACTURER variable.
unique_instances_recog_for_name <-
  unique(relevant_columns$PRODUCT_NAME)
# Count the number of unique instances
num_unique_instances_NAME <-
  length(unique_instances_recog_for_name)
print(num_unique_instances_NAME)
```

```
[1] 1082
```

```
# Issue with excessive numeric instances in the MANUFACTURER variable.
unique_instances_recog_for_manufacturer <-
  unique(relevant_columns$MANUFACTURER)
# Count the number of unique instances
num_unique_instances_MANU <-
  length(unique_instances_recog_for_manufacturer)
print(num_unique_instances_MANU)
```

```
[1] 975
```

Of, consequence, to drop such two features.

```
relevant_columns <- relevant_columns |>
  select(-PRODUCT_NAME, -MANUFACTURER)
```

Sample with a metal concentration below the laboratory reporting limit is marked "-1", meaning "Not Detected". # Then, will drop such entries so they will not contort the data with -1 values. The units of PPM identifies parts per million w.r.t. mass, which is equivalent to 1 milligram per litre.

```r
# Sample with a metal concentration below the laboratory reporting limit is marked "-1", m
# Then, will drop such entries so they will not contort the data with -1 values.
cleaned_data <- relevant_columns |>
  filter(`CONCENTRATION (PPM)` != -1)
```

```r
glimpse(cleaned_data)
```

```
Rows: 694
Columns: 4
$ PRODUCT_TYPE          <chr> "Dietary Supplement/Medications/Remedy", "Dietar~
$ METAL                 <chr> "Lead", "Lead", "Lead", "Lead", "Lead", "Lead", ~
$ `CONCENTRATION (PPM)` <dbl> 1.2e+04, 2.7e+00, 1.5e-02, 1.6e-01, 4.4e-01, 4.1~
$ MADE_IN_COUNTRY       <chr> "INDIA", "INDIA", "INDIA", "INDIA", "MEXICO", "M~
```

Now, to observe the range of instances for each feature; specifically concerned with the categorical features.

```r
length(unique(cleaned_data$PRODUCT_TYPE))
```

```
[1] 10
```

```r
length(unique(cleaned_data$METAL))
```

```
[1] 3
```

```r
length(unique(cleaned_data$MADE_IN_COUNTRY))
```

```
[1] 49
```

For the "PRODUCT_TYPE" feature there are 10 unique instances. For the "METAL" feature there are 3 unique instances. For the "MADE_IN_COUNTRY" feature there are 49 unique features. So, keeping record of uniqueness is crucial to make any proper analysis later on.

```r
# Create a factor for the product type column
cleaned_data$PRODUCT_TYPE_factor <- factor(cleaned_data$PRODUCT_TYPE)

# Create a new column for numeric codes
```

```r
cleaned_data$PRODUCT_TYPE_numeric <- as.integer(cleaned_data$PRODUCT_TYPE_factor)

# Print the original and recoded COUNTRY columns
print(cleaned_data[, c("PRODUCT_TYPE", "PRODUCT_TYPE_numeric")])
```

```
# A tibble: 694 x 2
   PRODUCT_TYPE                                     PRODUCT_TYPE_numeric
   <chr>                                                           <int>
 1 Dietary Supplement/Medications/Remedy                               2
 2 Dietary Supplement/Medications/Remedy                               2
 3 Dietary Supplement/Medications/Remedy                               2
 4 Dietary Supplement/Medications/Remedy                               2
 5 Food-Candy                                                          3
 6 Toys/Children's Products                                           10
 7 Dietary Supplement/Medications/Remedy                               2
 8 Dietary Supplement/Medications/Remedy                               2
 9 Dietary Supplement/Medications/Remedy                               2
10 Dietary Supplement/Medications/Remedy                               2
# i 684 more rows
```

```r
# Print levels of the factor
print(levels(cleaned_data$PRODUCT_TYPE_factor))
```

```
 [1] "Cosmetics"
 [2] "Dietary Supplement/Medications/Remedy"
 [3] "Food-Candy"
 [4] "Food-Spice"
 [5] "Food Other"
 [6] "Other"
 [7] "Paint Supplies"
 [8] "Religious powder"
 [9] "Tableware/Pottery"
[10] "Toys/Children's Products"
```

```r
# Create a factor for the metal type column
cleaned_data$METAL_factor <- factor(cleaned_data$METAL)

# Create a new column for numeric codes
cleaned_data$METAL_numeric <- as.integer(cleaned_data$METAL_factor)
```

```r
# Print the original and recoded METALcolumns
print(cleaned_data[, c("METAL", "METAL_numeric")])
```

```
# A tibble: 694 x 2
   METAL   METAL_numeric
   <chr>           <int>
 1 Lead                2
 2 Lead                2
 3 Lead                2
 4 Lead                2
 5 Lead                2
 6 Lead                2
 7 Lead                2
 8 Mercury             3
 9 Arsenic             1
10 Lead                2
# i 684 more rows
```

```r
# Print levels of the factor
print(levels(cleaned_data$METAL_factor))
```

```
[1] "Arsenic" "Lead"    "Mercury"
```

```r
# Create a factor for the country type column
cleaned_data$COUNTRY_factor <- factor(cleaned_data$MADE_IN_COUNTRY)

# Create a new column for numeric codes
cleaned_data$COUNTRY_numeric <- as.integer(cleaned_data$COUNTRY_factor)

# Print the original and recoded METAL columns
print(cleaned_data[, c("MADE_IN_COUNTRY", "COUNTRY_numeric")])
```

```
# A tibble: 694 x 2
   MADE_IN_COUNTRY COUNTRY_numeric
   <chr>                     <int>
 1 INDIA                        20
 2 INDIA                        20
 3 INDIA                        20
 4 INDIA                        20
 5 MEXICO                        28
 6 MACAU                         26
 7 INDIA                         20
 8 INDIA                         20
 9 INDIA                         20
10 INDIA                         20
# i 684 more rows
```

```
# Print levels of the factor
print(levels(cleaned_data$COUNTRY_factor))
```

```
 [1] "AFGHANISTAN"                    "BANGLADESH"
 [3] "BOSNIA AND HERZEGOWINA"         "BRAZIL"
 [5] "CANADA"                         "CHILE"
 [7] "CHINA"                          "COLOMBIA"
 [9] "DOMINICAN REPUBLIC"             "ECUADOR"
[11] "EGYPT"                          "FRANCE"
[13] "GEORGIA"                        "GERMANY"
[15] "GHANA"                          "GUATEMALA"
[17] "GUYANA"                         "HAITI"
[19] "HONG KONG"                      "INDIA"
[21] "INDONESIA"                      "ITALY"
[23] "IVORY COAST"                    "JAMAICA"
[25] "LEBANON"                        "MACAU"
[27] "MALAYSIA"                       "MEXICO"
[29] "MYANMAR"                        "NEPAL"
[31] "NIGERIA"                        "PAKISTAN"
[33] "PARAGUAY"                       "PERU"
[35] "RUSSIAN FEDERATION"             "SAUDI ARABIA"
[37] "SPAIN"                          "SRI LANKA"
[39] "SWEDEN"                         "SWITZERLAND"
[41] "TAIWAN - PROVINCE OF CHINA"     "TANZANIA - UNITED REPUBLIC OF"
[43] "THAILAND"                       "TRINIDAD AND TOBAGO"
[45] "UNITED ARAB EMIRATES"           "UNITED KINGDOM"
[47] "UNITED STATES"                  "UZBEKISTAN"
[49] "YEMEN"
```

```
# Observing first 6 rows.
head(cleaned_data)
```

```
# A tibble: 6 x 10
  PRODUCT_TYPE   METAL `CONCENTRATION (PPM)` MADE_IN_COUNTRY PRODUCT_TYPE_factor
  <chr>          <chr>                 <dbl> <chr>           <fct>
1 Dietary Suppl~ Lead               12000     INDIA           Dietary Supplement~
2 Dietary Suppl~ Lead                   2.7   INDIA           Dietary Supplement~
3 Dietary Suppl~ Lead                   0.015 INDIA           Dietary Supplement~
4 Dietary Suppl~ Lead                   0.16  INDIA           Dietary Supplement~
5 Food-Candy     Lead                   0.44  MEXICO          Food-Candy
6 Toys/Children~ Lead                  41     MACAU           Toys/Children's Pr~
# i 5 more variables: PRODUCT_TYPE_numeric <int>, METAL_factor <fct>,
#   METAL_numeric <int>, COUNTRY_factor <fct>, COUNTRY_numeric <int>
```

```
# Further cleaning: to visualize well the factor and numeric columns in dual.
# Such includes placing the CONCENTRATION column into the last spot.
# Move column 'A' to the last position
cleaned_data <- cleaned_data |>
  select(-`CONCENTRATION (PPM)`, everything(), `CONCENTRATION (PPM)`) |>
  select(-PRODUCT_TYPE, -METAL, -MADE_IN_COUNTRY)
# Showing first 6 rows
head(cleaned_data)
```

```
# A tibble: 6 x 7
  PRODUCT_TYPE_factor            PRODUCT_TYPE_numeric METAL_factor METAL_numeric
  <fct>                                         <int> <fct>                <int>
1 Dietary Supplement/Medication~                    2 Lead                     2
2 Dietary Supplement/Medication~                    2 Lead                     2
3 Dietary Supplement/Medication~                    2 Lead                     2
4 Dietary Supplement/Medication~                    2 Lead                     2
5 Food-Candy                                        3 Lead                     2
6 Toys/Children's Products                         10 Lead                     2
# i 3 more variables: COUNTRY_factor <fct>, COUNTRY_numeric <int>,
#   `CONCENTRATION (PPM)` <dbl>
```

Data Frame Conversion to File

You can transform a data frame into a CSV file using the **`write.csv()`** function in R. Here's how you can do it within a function:

rite_dataframe_to_csv <- function(df, filename) { write.csv(df, file = filename, row.names = FALSE) }

Example data frame

df <- data.frame(A = c(1, 2, 3), B = c(4, 5, 6), C = c(7, 8, 9))

Call the function to write the data frame to a CSV file.

write_dataframe_to_csv(df, “output.csv”)

In such code:

- We define a function `write_dataframe_to_csv()` that takes two arguments: `df` (the data frame to be written to CSV) and `filename` (the name of the CSV file).
- Within the function, we use `write.csv()` to write the data frame to the specified CSV file.
- We set `row.names = FALSE` to exclude row names from being written as the first column in the CSV file.
- We call the function with an example data frame `df` and specify the output filename `"output.csv"`.

After running this code, a CSV file named `"output.csv"` will be created in your working directory containing the data from the data frame `df`.

The cleaned data consisting of *factor* and *numeric* duals can be archived for referencing or recollection with future research or analysis.

Machine Learning Demonstration

Logistic regression and random forest are two commonly used algorithms for classification tasks, including when the target variable is categorical. The objective now is to apply logistic regression and random forest to a dataset with a categorical target and a mix of continuous and categorical features.

Random Forest

1. **Ensemble Method**: Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the class that is the mode of the classes of the individual trees.
2. **Non-Linear Model**: Random Forest can capture non-linear relationships between features and the target variable. It is capable of handling complex datasets with non-linear decision boundaries.
3. **Highly Accurate**: Random Forest tends to produce highly accurate predictions, often outperforming traditional linear models in various scenarios.
4. **Robust to Overfitting**: Random Forest is less prone to overfitting compared to individual decision trees, thanks to the randomization in feature selection and bootstrap aggregation (bagging).
5. **Feature Importance**: It provides a measure of feature importance, indicating which features contribute the most to the predictive accuracy of the model.
6. **Handles Missing Values and Outliers**: Random Forest can handle missing values and outliers in the data without requiring preprocessing.
7. **Parallelizable**: Random Forest can be easily parallelized, making it efficient for training on large datasets and taking advantage of multi-core processors.

Prepping the Data

Will focus only on the numeric label columns and the concentration column because the instances of such columns are numeric values making them meaningful to machine learning algorithms.

```r
# Data frame for ML.
# Showing first 6 rows.
ML_data <- cleaned_data |>
  select(-PRODUCT_TYPE_factor, -METAL_factor, -COUNTRY_factor)
head(ML_data)
```

```
# A tibble: 6 x 4
  PRODUCT_TYPE_numeric METAL_numeric COUNTRY_numeric `CONCENTRATION (PPM)`
                 <int>         <int>           <int>                 <dbl>
1                    2             2              20             12000
2                    2             2              20                 2.7
3                    2             2              20                 0.015
```

```
4                    2             2              20                 0.16
5                    3             2              28                 0.44
6                   10             2              26                41
```

```
# Load required libraries
library(ggplot2)     # for data visualization
library(caret)       # for machine learning models

# Set the seed for reproducibility
set.seed(123)

# Split the data into training and testing sets (80% training, 20% testing)
trainIndex <- createDataPartition(ML_data$PRODUCT_TYPE_numeric, p = .8,
                                  list = FALSE,
                                  times = 1)
train_data <- ML_data[trainIndex, ]
test_data <- ML_data[-trainIndex, ]

# Ensure that levels of PRODUCT_TYPE in training and testing data match.
# Instances in the training set may not exist in the testing set; vice versa.
levels_train <- levels(train_data$PRODUCT_TYPE_numeric)
levels_test <- levels(test_data$PRODUCT_TYPE_numeric)

# Check for unseen levels in the testing dataset
unseen_levels <- levels_test[!levels_test %in% levels_train]

# Remove rows with unseen levels from the testing dataset
test_data <- test_data[!test_data$PRODUCT_TYPE_numeric %in% unseen_levels, ]


# Random Forest model
rf_model <- train(PRODUCT_TYPE_numeric ~ ., data = train_data,
                  method = "rf",
                  trControl = trainControl(method = "cv"))
```

```
note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
# Print the results
print(rf_model)
```

```
Random Forest

556 samples
  3 predictor

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 501, 501, 500, 500, 501, 499, ...
Resampling results across tuning parameters:

  mtry  RMSE      Rsquared   MAE
  2     1.742373  0.4301459  0.9291152
  3     1.797782  0.4083818  0.9370862

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was mtry = 2.
```

```r
# Predictions
rf_pred <- predict(rf_model, newdata = test_data)


# Calculate accuracy
rf_accuracy <-
  sum(rf_pred == test_data$PRODUCT_TYPE) / length(test_data$PRODUCT_TYPE)

cat("Random Forest Accuracy:", rf_accuracy, "\n")
```

```
Random Forest Accuracy: NaN
```

NOTE: in the context of random forests, `mtry` is a tuning parameter that determines the number of variables randomly sampled as candidates at each split when constructing each tree in the forest.

Here's what `mtry` manages:

1. **Variable Sampling**: At each split of a decision tree in the random forest, `mtry` random variables (features) are selected from the total set of predictors. These selected variables are candidates for determining the best split at that node.
2. **Control Overfitting**: By limiting the number of variables considered at each split, `mtry` helps control overfitting in random forests. If `mtry` is too high, the trees may become more correlated, potentially leading to overfitting. If it's too low, the trees may become too dissimilar, reducing the benefits of averaging predictions.

3. **Default Value**: In most implementations, the default value of **`mtry`** is the square root of the total number of predictors in the dataset. This value is often a good starting point for experimentation.
4. **Tuning**: **`mtry`** is a hyperparameter that you can tune during model training to optimize the performance of the random forest. You can use techniques like grid search or random search to find the best value of **`mtry`** based on cross-validation performance.

In summary, **`mtry`** controls the randomness and diversity of individual trees in a random forest, balancing the trade-off between model variance and bias. Adjusting **`mtry`** can significantly impact the performance and generalization ability of the random forest model.

Conclusion

In the realm of sustainability, data science serves as a powerful tool for driving positive change and fostering inclusive growth. Sustainability encompasses efforts to balance economic growth, environmental preservation, and social well-being to meet the needs of the present without compromising the ability of future generations to meet their own needs. Data science plays a vital role in sustainability initiatives by analyzing environmental data, predicting future trends, and optimizing resource allocation. Through techniques like predictive modeling and machine learning, data scientists can inform sustainable practices in areas such as trade, contamination in goods and products, waste management, and urban planning.
Moreover, data science fosters transparency and accountability by enabling stakeholders to access, analyze, and interpret data relevant to equity and sustainability. Open data initiatives, data visualization tools, and citizen science projects empower individuals and communities to actively participate in decision-making processes and advocate for positive change.

References

Department of Health and Mental Hygiene. (2024, February 7). *Metal content of consumer products tested by the NYC Health Department: NYC open data.* Metal Content of Consumer Products Tested by the NYC Health Department | NYC Open Data. https://data.cityofnewyork.us/Health/Metal-Content-of-Consumer-Products-Tested-by-the-N/da9u-wz3r/about_data