Forecasting Tourism Income by Type of Expenditure of Citizens Resident Abroad

Cem Demir , Sude Halaçeli , Zeynep Taş

2023-06-02

Loading Required Packages for Data Manipulation and Visualization

The code chunk is responsible for loading the necessary packages that are required for data manipulation and visualization .

```
rm(list = ls())
options(repos = "https://cloud.r-project.org")
path.expand("/Users/cemdemir/Desktop/Courses/ECO2868_MLwR/FRESH")
```

```
## [1] "/Users/cemdemir/Desktop/Courses/ECO2868_MLwR/FRESH"
```

```
setwd("/Users/cemdemir/Desktop/Courses/ECO2868_MLwR/FRESH")
getwd()
```

```
## [1] "/Users/cemdemir/Desktop/Courses/ECO2868_MLwR/FRESH"
```

```
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
```

```
library(tidyr)
```

```
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(patchwork)
library(readr)
library(scales)

##
## Attaching package: 'scales'

## The following object is masked from 'package:readr':
##
##     col_factor

library(ggplot2)
```

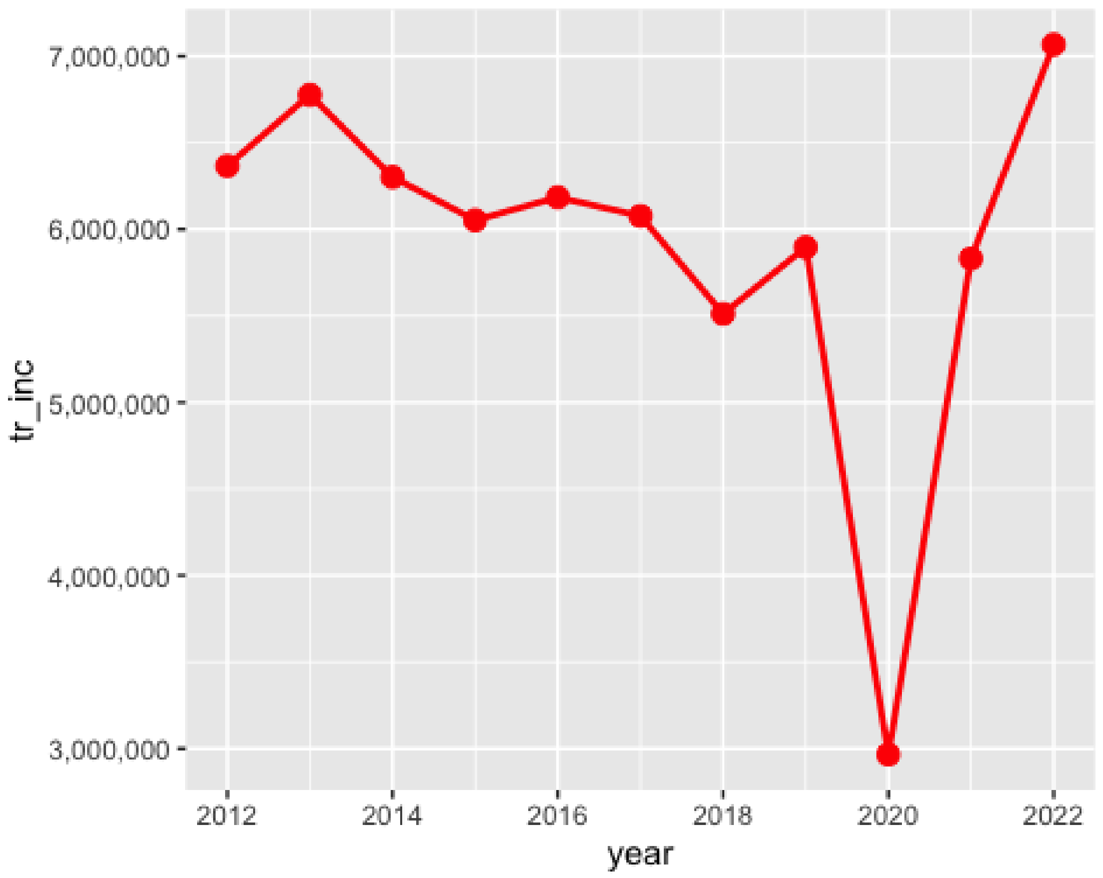
Reading Data and Selecting Annual Observations

```
data <- read.csv("dataFormatted.csv")
annualData <- subset(data, qt=="Annual")

annualData <-
  annualData |>
  select(c(-qt))

annualDataNum <- mutate_all(annualData, as.numeric)

is.numeric(annualDataNum$tr_inc)

## [1] TRUE
```

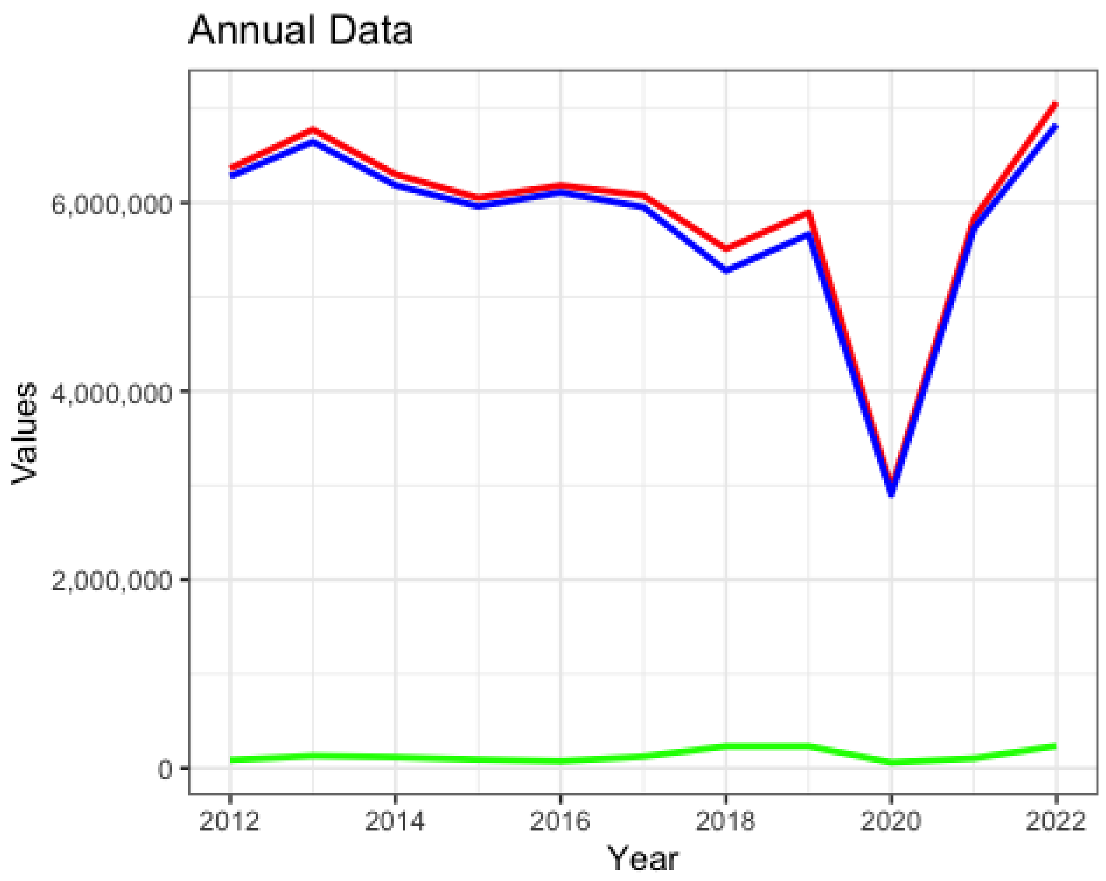
Data Visualization: Yearly Trend of Total Income

In this step, we focus on visualizing the yearly trend of the total income variable. By calling the ggplot() function and specifying the annualDataNum data frame as the data source, we set the foundation for our plot.

```
# Plots with only one variable
ggplot(annualDataNum, aes(x=year, y=tr_inc)) +
  geom_point(size = 3, col = "red") +
  geom_line(lwd = 1, col = "red") +
  scale_x_continuous(breaks= pretty_breaks()) +
  scale_y_continuous(labels = scales::comma_format())
```

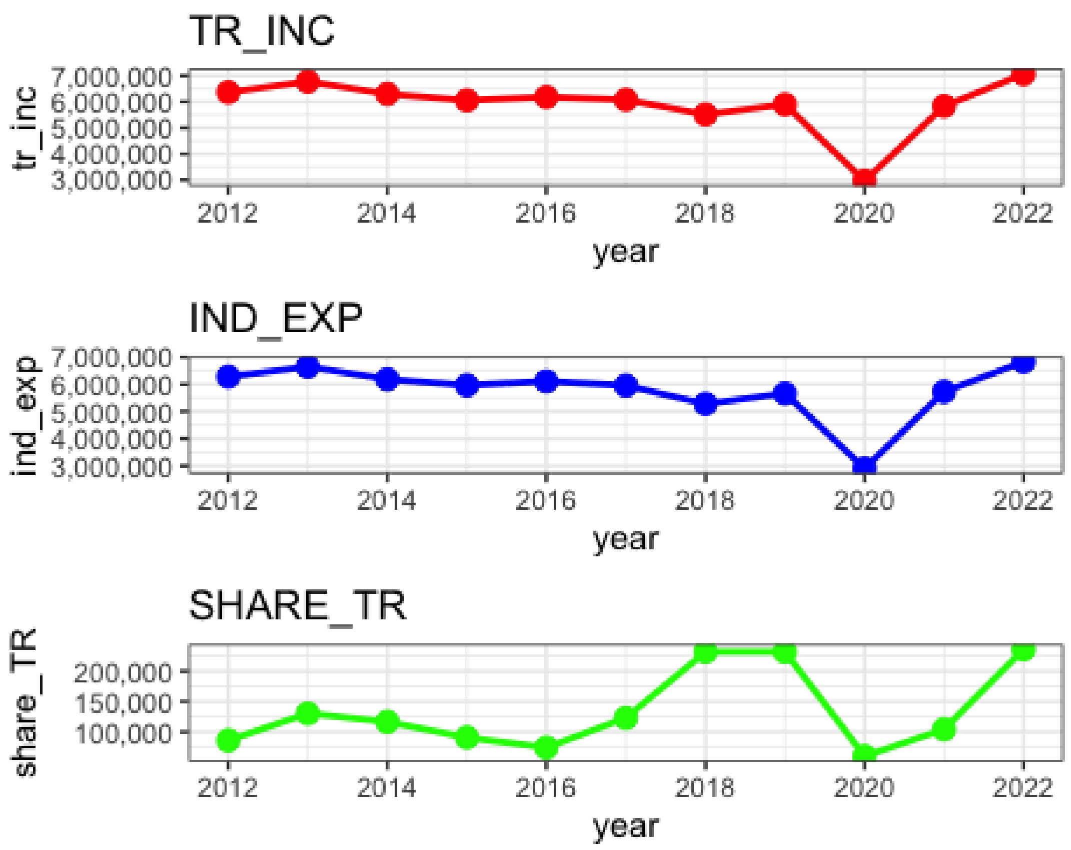


```
# Plots with multiple variables
ggplot(annualDataNum, aes(x=year)) +
  geom_line(aes(y = tr_inc), size = 1, col = "red") +
  geom_line(aes(y = ind_exp), size = 1, col = "blue") +
  geom_line(aes(y = share_TR), size = 1, col = "green") +
  scale_x_continuous(breaks= pretty_breaks()) +
  scale_y_continuous(labels = scales::comma_format()) +
  labs(title = "Annual Data", x = "Year", y = "Values") +
  theme_bw()

## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



```r
# Plots with three variables as different graphs and stacks them
p1 <- ggplot(annualDataNum, aes(x=year, y=tr_inc)) +
  geom_point(size = 3, col = "red") +
  geom_line(lwd = 1, col = "red") +
  scale_x_continuous(breaks= pretty_breaks()) +
  scale_y_continuous(labels = comma_format()) +
  ggtitle("TR_INC") + theme_bw()

p2 <- ggplot(annualDataNum, aes(x=year, y=ind_exp)) +
  geom_point(size = 3, col = "blue") +
  geom_line(lwd = 1, col = "blue") +
  scale_x_continuous(breaks= pretty_breaks()) +
  scale_y_continuous(labels = comma_format()) +
  ggtitle("IND_EXP") + theme_bw()

p3 <- ggplot(annualDataNum, aes(x=year, y=share_TR)) +
  geom_point(size = 3, col = "green") +
  geom_line(lwd = 1, col = "green") +
  scale_x_continuous(breaks= pretty_breaks()) +
  scale_y_continuous(labels = comma_format()) +
  ggtitle("SHARE_TR") + theme_bw()
```

```
(p1 / p2 / p3) + plot_layout(ncol = 1) # this lines plots all three
```



```
data <- data %>%
  mutate_at(vars(-qt), as.numeric)

## Warning: There were 14 warnings in `mutate()`.
## The first warning was:
## i In argument: `tr_inc = .Primitive("as.double")(tr_inc)`.
## Caused by warning:
## ! NAs introduced by coercion
## i Run `dplyr::last_dplyr_warnings()` to see the 13 remaining warnings.
```

Reformatting and Combining GDP Data

This code chunk demonstrates the process of reformatting and combining GDP data from separate sources, resulting in a unified and well-structured dataset ready for further analysis and visualization. First, we read two separate CSV files into variables gdp and gdpA, and then remove the “Flag.Codes” column from both datasets. We inspect the contents of gdp and gdpA to examine the data. Next, we to transform the gdp and gdpA data frames from long to wide format, creating new columns for each unique location and populating them with GDP values. The reformed data frames, reformed_gdp and reformed_gdpA, are displayed to confirm the changes. We also add a new column named “qt” with the value “Annual” to indicate the data frequency. The reformed_gdp data frame is further modified by separating the “TIME” column into “year” and “qt” columns using the separate() function. The “qt” column is converted to character values representing the quarters of the year (I, II, III, IV). Both modified data frames, reformed_gdp and reformed_gdpA, are saved as separate CSV files. We combine the reformed_gdpA and non-annual quarters of reformed_gdp into a new data frame called reformed_gdp_full using rbind(), which is then displayed using View(). The “qt” column in reformed_gdp_full is converted to a factor variable with predefined levels representing “Annual,” “I,” “II,” “III,” and “IV” quarters. Finally, the reformed_gdp_full data frame is sorted by year and quarter, and the sorted data is written to a CSV file named “reformedGdpFull.csv” using write.csv(). This code chunk demonstrates the process of reformatting and combining GDP data from separate sources, resulting in a unified and well-structured dataset ready for further analysis and visualization.

```
### GDP reformat
library(readr)
gdp <- read_csv("gdpUgly.csv")

## Rows: 135 Columns: 8
## ── Column specification
────────────────────────────────────────────────────────
## Delimiter: ","
## chr (7): LOCATION, INDICATOR, SUBJECT, MEASURE, FREQUENCY, TIME, Flag
Codes
## dbl (1): Value
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.

library(readr)
gdpA <- read_csv("gdpUglyA.csv")

## Rows: 33 Columns: 8
## ── Column specification
────────────────────────────────────────────────────────
## Delimiter: ","
## chr (6): LOCATION, INDICATOR, SUBJECT, MEASURE, FREQUENCY, Flag Codes
## dbl (2): TIME, Value
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.

names(gdp)

## [1] "LOCATION"   "INDICATOR"  "SUBJECT"    "MEASURE"    "FREQUENCY"
## [6] "TIME"       "Value"      "Flag Codes"
```

```
gdp <- gdp %>% select(-`Flag Codes`)

gdpA <- gdpA %>% select(-`Flag Codes`)

reformed_gdp <- gdp %>%
  pivot_wider(names_from = LOCATION, values_from = Value)

reformed_gdpA <- gdpA %>%
  pivot_wider(names_from = LOCATION, values_from = Value)

reformed_gdp <- reformed_gdp %>% select(TIME, GBR, DEU, FRA)
reformed_gdpA <- reformed_gdpA %>% select(TIME, GBR, DEU, FRA)

colnames(reformed_gdpA)[colnames(reformed_gdpA) == "TIME"] <- "year"
reformed_gdpA$qt <- "Annual"

reformed_gdpA <- reformed_gdpA[, c(1, 5, 2, 3, 4)]


reformed_gdp <- reformed_gdp %>%
  separate(TIME, into = c("year", "qt"), sep = "-Q")

reformed_gdp$qt <- ifelse(reformed_gdp$qt == 1, "I",
                          ifelse(reformed_gdp$qt == 2, "II",
                                 ifelse(reformed_gdp$qt == 3, "III",
                                        ifelse(reformed_gdp$qt == 4, "IV",
""))))
write.csv(reformed_gdp, file = "reformedGdp.csv", row.names = FALSE)
write.csv(reformed_gdpA, file = "reformedGdpA.csv", row.names = FALSE)

reformed_gdp_full <- rbind(reformed_gdpA, reformed_gdp[reformed_gdp$qt !=
"Annual", ])


reformed_gdp_full$qt <- factor(reformed_gdp_full$qt, levels = c("Annual",
"I", "II", "III", "IV"))
reformed_gdp_full <- reformed_gdp_full[order(reformed_gdp_full$year,
reformed_gdp_full$qt), ]
write.csv(reformed_gdp_full, file = "reformedGdpFull.csv", row.names = FALSE)
```

Reformatting Exchange Data

```
exc <- read_csv("excUgly.csv")

## Rows: 56 Columns: 17
## ── Column specification
────────────────────────────────────────────────────────
## Delimiter: ","
## chr (11): SUBJECT, Subject, LOCATION, Country, FREQUENCY, Frequency, TIME,
T...
## dbl  (2): PowerCode Code, Value
## lgl  (4): Reference Period Code, Reference Period, Flag Codes, Flags
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.

exc <- exc %>% select(TIME, Value)


write.csv(exc, file = "exc.csv", row.names = FALSE)
reformed_exc <- exc %>%
  separate(TIME, into = c("year", "qt"), sep = "-Q")

## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 11 rows [1,
2, 3, 4, 5,
## 6, 7, 8, 9, 10, 11].

reformed_exc$qt <- ifelse(is.na(reformed_exc$qt), "Annual", reformed_exc$qt)

reformed_exc$qt <- ifelse(reformed_exc$qt == 1, "I",
                          ifelse(reformed_exc$qt == 2, "II",
                                 ifelse(reformed_exc$qt == 3, "III",
                                        ifelse(reformed_exc$qt == 4, "IV",
                                               ifelse(reformed_exc$qt ==
"Annual", "Annual",""))))) 


write.csv(reformed_exc, file = "exc.csv", row.names = FALSE)

exc <- read.csv("exc.csv")

desired_order <- c(1, 12, 13, 14, 15, 2, 16, 17, 18, 19, 3, 20, 21, 22, 23 ,
4,24,25,26,27,5,28,29,30,31,6,32,33,34,35,7,36,37,38,39,8,40,41,42,43,9,44,45
,46,47,10,48,49,50,51,56)  # Update with the desired order
exc_reorder <- exc %>%
  slice(desired_order)

write.csv(exc_reorder, file="exc_reored.csv", row.names = FALSE)
```

Combining Tables for Analysis

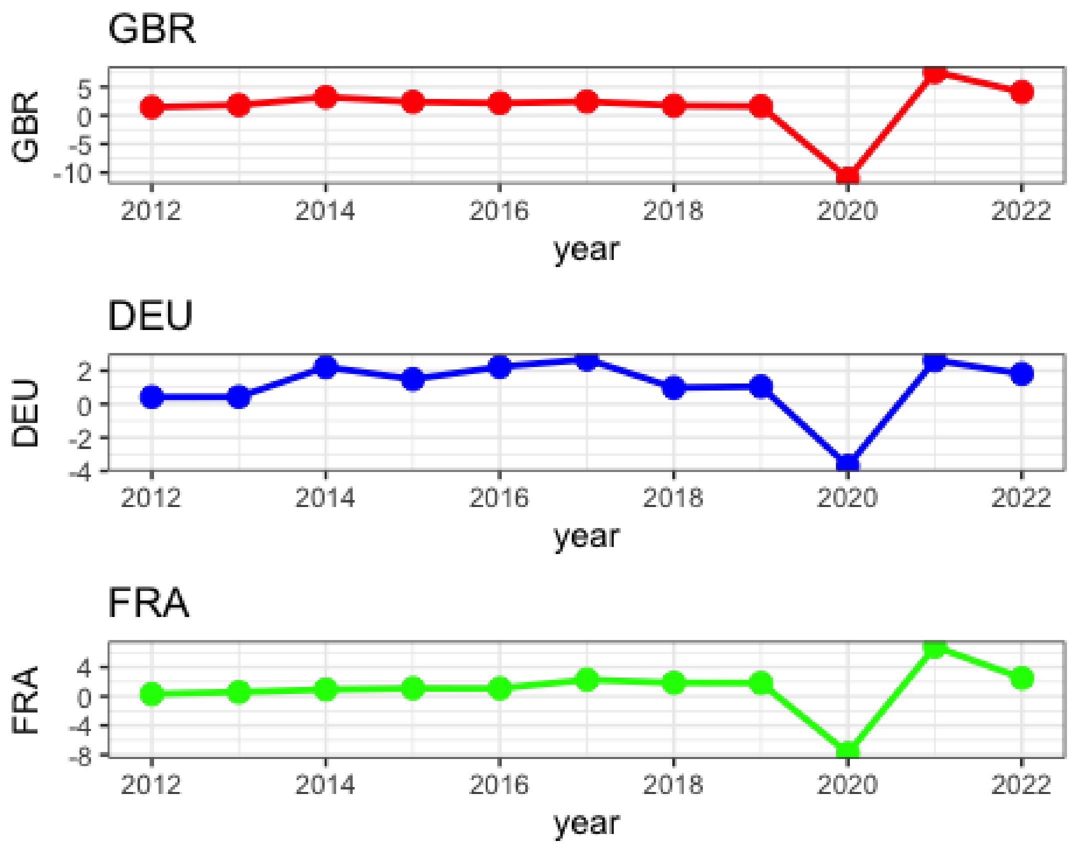
In this code chunk we focus on combining multiple tables (fullData, excC, and gdpC) based on common columns ("year" and "qt") to create a unified dataset (mergedData) for further analysis. The resulting dataset includes exchange rate information (exc_rate) and GDP data, providing a thorough dataset for subsequent analyses and visualizations.

```
### Combining the tables

excC <- exc_reorder
colnames(excC)[colnames(excC) == "Value"] <- "exc_rate"

gdpC <- reformed_gdp_full
fullData <- data
write.csv(fullData, "fullData.csv", row.names = FALSE)

mergedData <- fullData

mergedData <- merge(mergedData, excC, by = c("year", "qt"), all.x = TRUE)
mergedData <- merge(mergedData, gdpC, by = c("year", "qt"), all.x = TRUE)
```

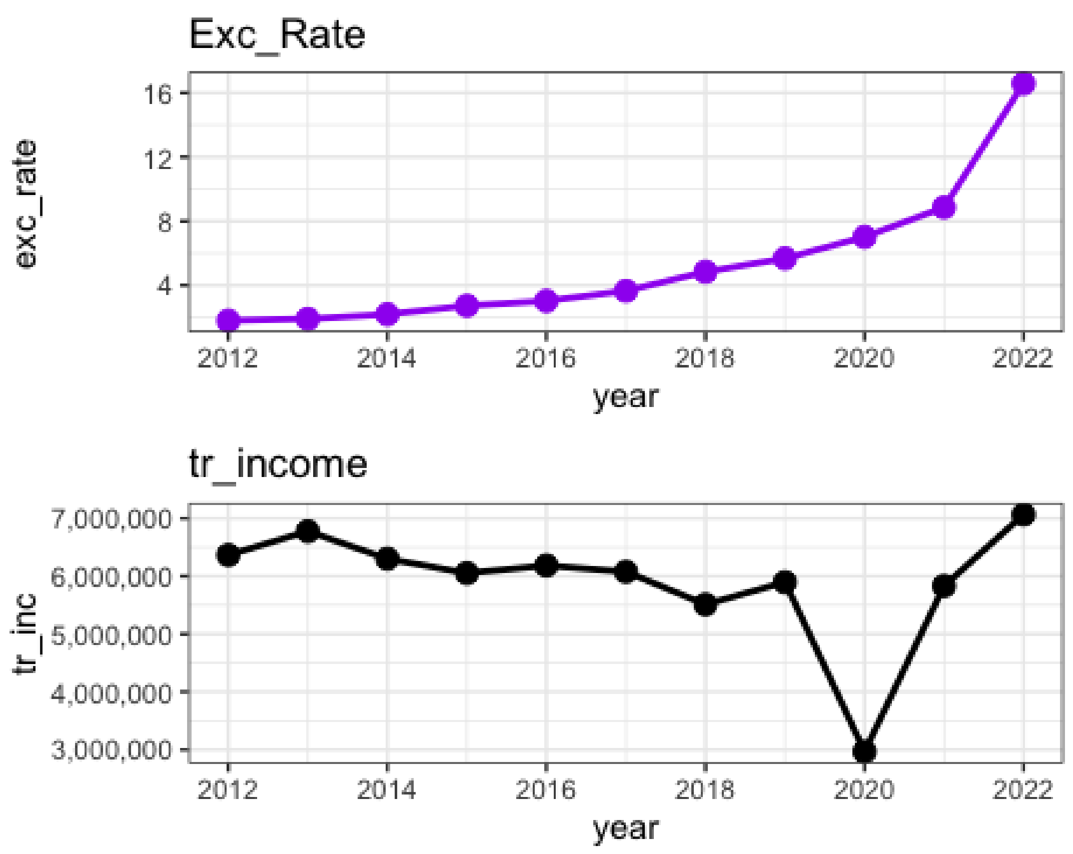
Merged data plots

```
mergedData_2_ <- read_csv("mergedData.csv")

## Rows: 55 Columns: 20
## ── Column specification
────────────────────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (1): qt
## dbl (19): year, tr_inc, ind_exp, share_TR, food_bvg, accomodation, health,
t...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.

View(mergedData_2_)
annualData <- subset(mergedData_2_, qt == "Annual")

annualData <-
  annualData |>
  select(-qt)

annualDataNum <- mutate_all(annualData, as.numeric)

is.numeric(annualDataNum$tr_inc)

## [1] TRUE

p1 <- ggplot(annualDataNum, aes(x = year, y = GBR)) +
  geom_point(size = 3, col = "red") +
  geom_line(lwd = 1, col = "red") +
  scale_x_continuous(breaks = pretty_breaks()) +
  scale_y_continuous(labels = comma_format()) +
  ggtitle("GBR") + theme_bw()
```

```r
p2 <- ggplot(annualDataNum, aes(x = year, y = DEU)) +
  geom_point(size = 3, col = "blue") +
  geom_line(lwd = 1, col = "blue") +
  scale_x_continuous(breaks = pretty_breaks()) +
  scale_y_continuous(labels = comma_format()) +
  ggtitle("DEU") + theme_bw()

p3 <- ggplot(annualDataNum, aes(x = year , y = FRA)) +
  geom_point(size = 3, col = "green") +
  geom_line(lwd = 1, col = "green") +
  scale_x_continuous(breaks = pretty_breaks()) +
  scale_y_continuous(labels = comma_format()) +
  ggtitle("FRA") + theme_bw()

p4 <- ggplot(annualDataNum, aes(x = year , y = exc_rate)) +
  geom_point(size = 3, col = "purple") +
  geom_line(lwd = 1, col = "purple") +
  scale_x_continuous(breaks = pretty_breaks()) +
  scale_y_continuous(labels = comma_format()) +
  ggtitle("Exc_Rate") + theme_bw()

p5 <- ggplot(annualDataNum, aes(x = year , y = tr_inc)) +
  geom_point(size = 3, col = "black") +
  geom_line(lwd = 1, col = "black") +
  scale_x_continuous(breaks = pretty_breaks()) +
  scale_y_continuous(labels = comma_format()) +
  ggtitle("tr_income") + theme_bw()


(p1 / p2 / p3 ) + plot_layout(ncol = 1) # this lines plots all five
```



```
(p4/p5) + plot_layout(ncol = 1)
```



Modeling and Analysis

Loading Packages and Reading Data

```
datainc<- read_csv("mergedData.csv")

## Rows: 55 Columns: 20
## ── Column specification
────────────────────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (1): qt
## dbl (19): year, tr_inc, ind_exp, share_TR, food_bvg, accomodation, health,
t...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

Data Preparation and Column Renaming

In this code chunk, we perform data preparation tasks and rename the columns of the ‘datainc’ dataset. These steps ensure consistency and clarity in the dataset for further analysis. First, we use the ‘names()’ function to assign new names to specific columns of ‘datainc’. The column names are modified using index-based assignment Next, we remove rows where the “qt” column contains the string “Annual” using the ‘subset()’ function. This step eliminates annual data records since we are focusing on quarterly data.

To ensure the correct ordering of the “qt” column as a factor variable, we convert it to a factor using the ‘factor()’ function. The levels are set as “I”, “II”, “III”, and “IV”, representing the four quarters, and the corresponding labels are set as “1”, “2”, “3”, and “4”. We then use the ‘mutate_all()’ function from the ‘dplyr’ package to convert all columns of ‘datainc’ to numeric data type using the ‘as.numeric()’ function. This step ensures that the data is in the appropriate format for subsequent calculations and modeling. Finally, the ‘write.csv()’ function is used to save the modified ‘datainc’ dataset as a CSV file. This code chunk demonstrates the necessary data preparation steps, including column renaming, filtering out unnecessary rows, and ensuring proper data types, to create a well-structured and formatted dataset for further analysis and modeling.

In this code chunk, and we also create dummy variables for the “qt” column. we modify the ‘qt’ column of ‘datainc’ to be a factor variable using the ‘factor()’ function. To create dummy variables for the ‘qt’ column, we use the ‘model.matrix()’ function.The resulting dummy variables are stored in ‘dummy_quarters’.

```r
datainc <- subset(datainc, !grepl("Annual", qt))
datainc$qt <- factor(datainc$qt, levels = c("I", "II", "III", "IV"), labels =
c("1", "2", "3", "4"))
datainc <- mutate_all(datainc, as.numeric)
is.numeric(datainc$qt)

## [1] TRUE

datainc$qt <- factor(datainc$qt, levels = c(1, 2, 3, 4), labels = c("Q1",
"Q2", "Q3", "Q4"))

dummy_quarters <- model.matrix(~ 0 + qt, data = datainc)
datainc <- cbind(datainc, dummy_quarters)


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

Data Splitting into Training and Testing Sets

In this code chunk, we split the ‘datainc’ dataset into training and testing subsets based on the years, and we also create dummy variables for the “qt” column. First, we create a training dataset named ‘data_train’ by subsetting ‘datainc’ using the condition that the ‘year’ column should be greater than or equal to 2012 and less than or equal to 2020. This ensures that the training dataset only contains data from the years 2012 to 2020. Likewise we create a testing dataset. This ensures that the testing dataset only contains data from the year 2021 onwards. Next, we modify the ‘qt’ column of ‘datainc’ to be a factor variable using the ‘factor()’ function. To create dummy variables for the ‘qt’ column, we use the ‘model.matrix()’ function.The resulting dummy variables are stored in ‘dummy_quarters’. Finally, we combine the ‘datainc’ dataset with the dummy variables by using the ‘cbind()’ function. This code chunk demonstrates the process of splitting the dataset into training and testing subsets based on years and creating dummy variables for the ‘qt’ column. These steps are crucial for further analysis and modeling tasks.

```
data_train <- datainc[datainc$year >= 2012 & datainc$year <= 2020, ]
data_test <- datainc[datainc$year >= 2021, ]
```

Linear Regression Modeling

This code snippet fits linear regression models to investigate the relationship between ‘tr_inc’ (total income) and predictor variables including ‘exc_rate’ (exchange rate), ‘GBR’ (Great Britain), ‘DEU’ (Germany), and ‘FRA’ (France). The first model, ‘LM, without Quarter Dummies’, is fitted using these predictors but does not consider the impact of different quarters on ‘tr_inc’. To account for potential quarterly variations, the second model, ‘LM, with Quarter Dummies’, includes Quarter Dummies as categorical variables representing each quarter (Q1, Q2, Q3, and Q4). This allows capturing any seasonal patterns in ‘tr_inc’. The ‘lm_model’ includes Quarter Dummies (‘qtQ1’, ‘qtQ2’, ‘qtQ3’, and ‘qtQ4’) as additional predictors. Comparing the results and model performance between the two models provides insights into the effect of incorporating Quarter Dummies on the relationship between ‘tr_inc’ and the other predictors.”

```
#LM, without Quarter Dummies

lm_modelNoQ <- lm(tr_inc ~ exc_rate + GBR + DEU + FRA, data = data_train)

#LM, with Quarter Dummies

lm_model <- lm(tr_inc ~ exc_rate + GBR + DEU + FRA + qtQ1 + qtQ2+ qtQ3 +
qtQ4, data = data_train)
```

Model Evaluation and Error Calculation

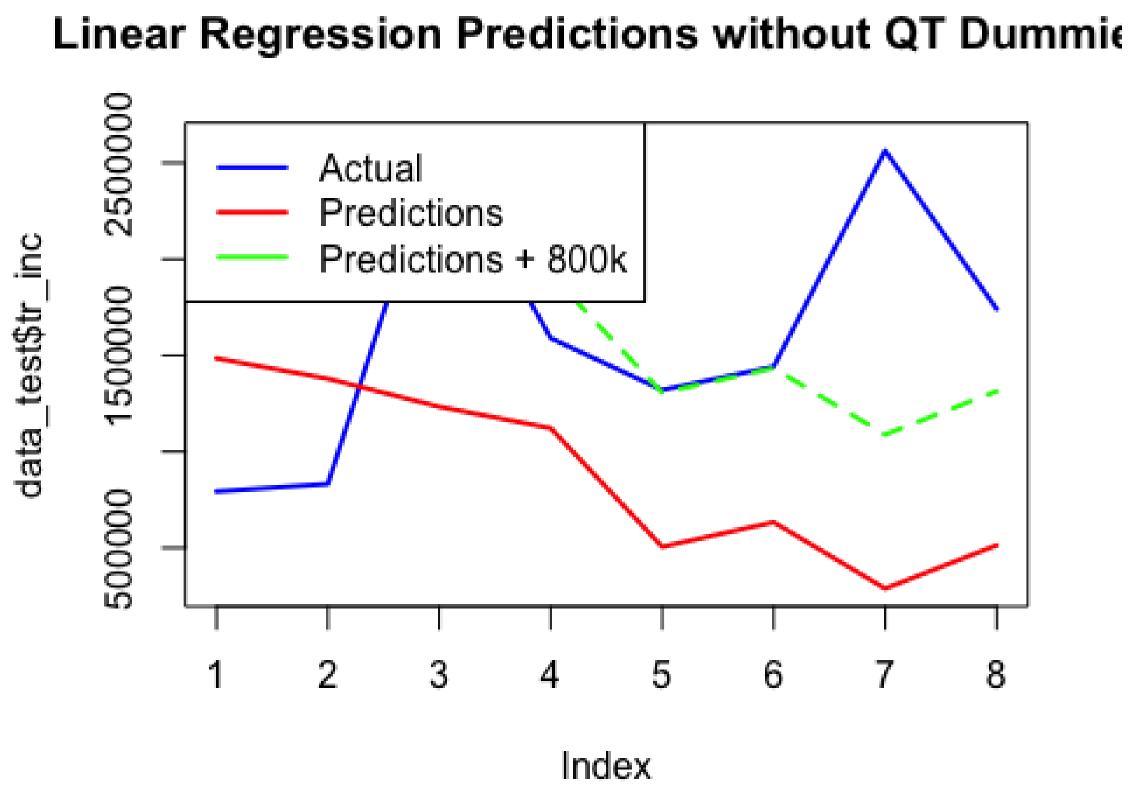
in this code chunk we show the process of fitting linear regression models to explore the relationship between the response variable ‘tr_inc’ (total income) and the predictor variables ‘exc_rate’ (exchange rate), ‘GBR’ (Great Britain), ‘DEU’ (Germany), and ‘FRA’ (France). In the first model, labeled ‘LM, without Quarter Dummies’, we fit a linear regression model using these predictor variables. This model provides a baseline understanding of the relationship between the predictors and ‘tr_inc’, but it does not consider the potential impact of different quarters on the response variable. To account for the potential influence of quarters on 'tr_inc', we have included Quarter Dummies in the second model, labeled 'LM, with Quarter Dummies'.

```
lm_predictNoQ <- predict(lm_modelNoQ, newdata = data_test)
lm_predictions <- predict(lm_model, newdata = data_test)

## Warning in predict.lm(lm_model, newdata = data_test): prediction from a
## rank-deficient fit may be misleading

lm_errors <- data_test$tr_inc - lm_predictions
lm_mse <- mean(lm_errors^2)
print("Linear Regression Model with QT Dummies MSE:")

## [1] "Linear Regression Model with QT Dummies MSE:"

print(lm_mse)

## [1] 1.364807e+12
```

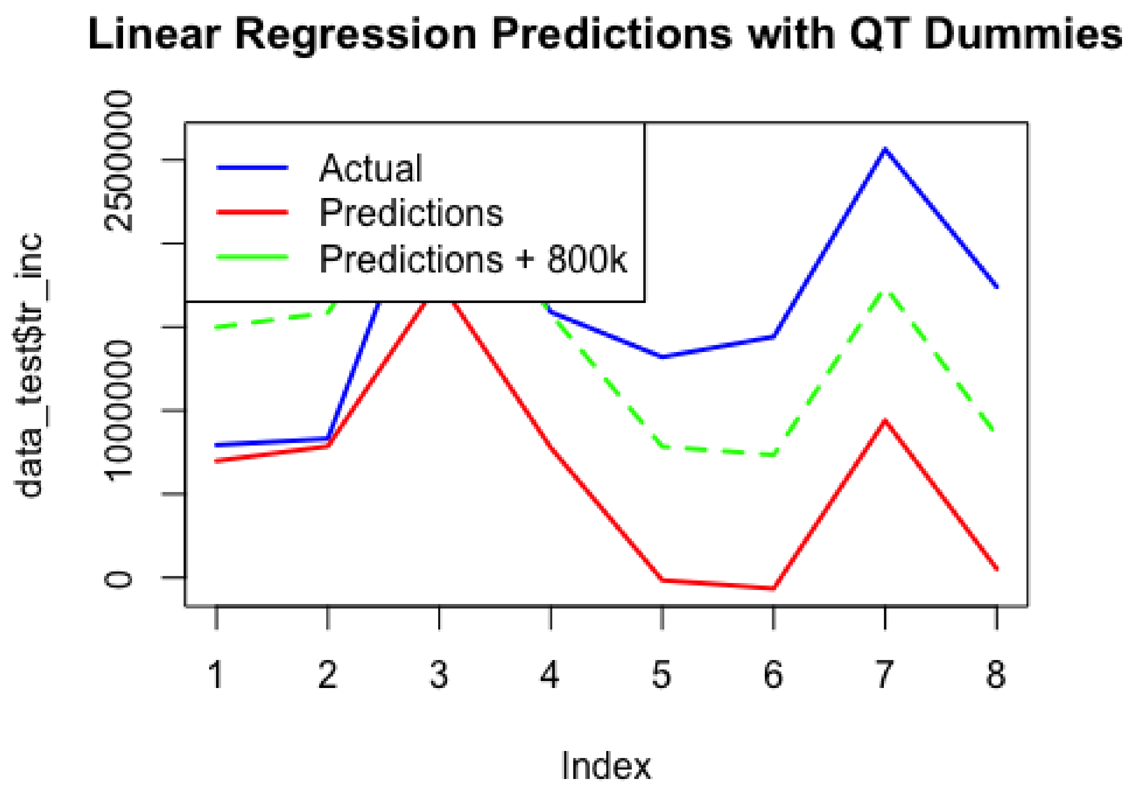
Linear Regression Model Predictions and Performance Visualization

First, we create a plot to visualize the predictions without Quarter Dummies, providing insights into the model's performance. To provide additional context, we include a dashed line representing the predicted values plus 800,000. By adding this line, we aim to approximate the predictions and compare them to the actual values in a more understandable way Following that, we repeat the process to create another plot, but this time for the linear regression model with Quarter Dummies.

```
# Linear Regression Model without QT
plot(data_test$tr_inc, type = "l", col = "blue", lwd = 2, main = "Linear
Regression Predictions without QT Dummies", ylim = c(min(lm_predictNoQ),
max(data_test$tr_inc)))
lines(lm_predictNoQ, col = "red", lwd = 2)
lines(lm_predictNoQ+800000, col = "green", lwd = 2, lty = 2)
legend("topleft", legend = c("Actual", "Predictions", "Predictions + 800k"),
col = c("blue", "red", "green"), lwd = 2)
```



```
# Linear Regression Model with QT
plot(data_test$tr_inc, type = "l", col = "blue", lwd = 2, main = "Linear
Regression Predictions with QT Dummies", ylim = c(min(lm_predictions),
max(data_test$tr_inc)))
lines(lm_predictions, col = "red", lwd = 2)
lines(lm_predictions+800000, col = "green", lwd = 2, lty = 2)
legend("topleft", legend = c("Actual", "Predictions", "Predictions + 800k"),
col = c("blue", "red", "green"), lwd = 2)
```

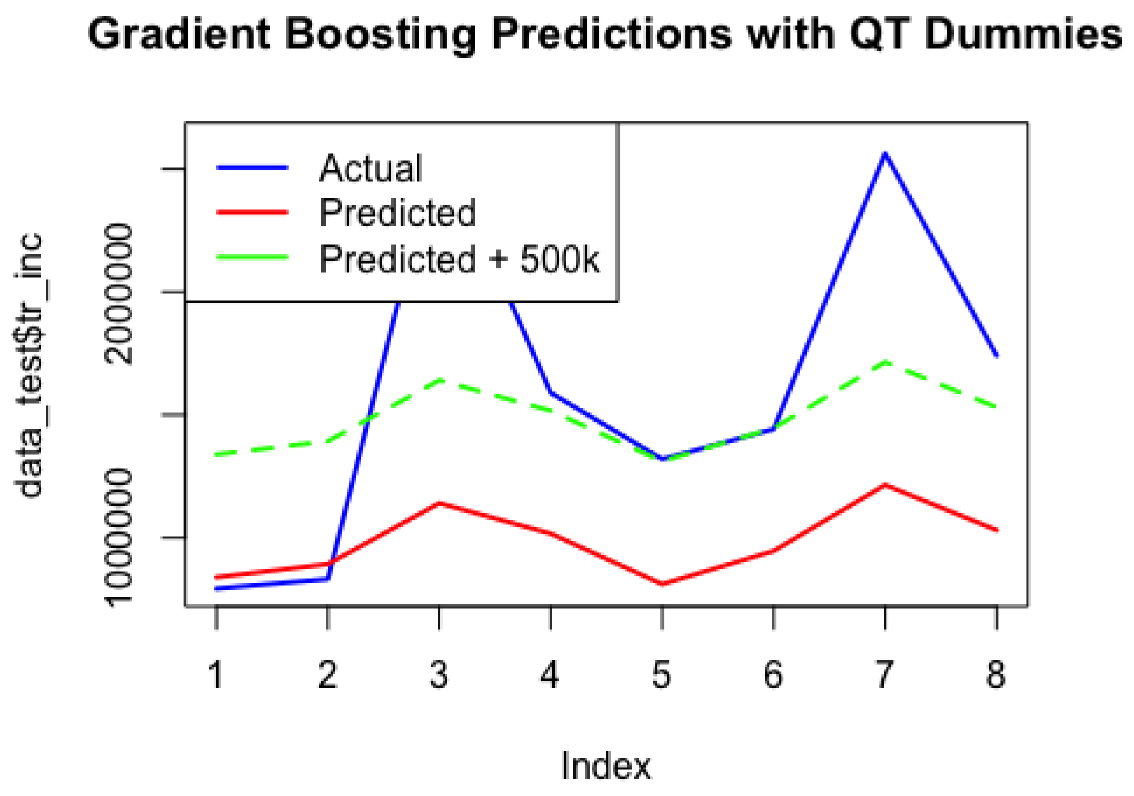


```
predicted <- predict(lm_model, newdata = data_test)+800000

## Warning in predict.lm(lm_model, newdata = data_test): prediction from a
## rank-deficient fit may be misleading

performance <- data.frame(Actual = data_test$tr_inc, Predicted = predicted)

print(performance)

##       Actual Predicted
## 37  793359.8 1498327.8
## 38  831884.3 1586164.0
## 39 2615955.0 2573566.9
## 40 1589753.5 1577593.8
## 41 1320051.7  784734.4
## 42 1441765.5  734551.6
## 43 2564155.4 1740874.5
## 44 1741179.4  853486.7
```

XGB with Quarter Dummies

In this code snippet, we train and evaluate a Gradient Boosting model using the XGBoost library with the addition of Quarter Dummies. To enhance the model's predictions, we adjust the predicted values by adding 500,000. This adjustment helps improve the alignment between the model's predictions and the actual values.

```r
library(xgboost)

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice

View(data_train)
xgb_modelQ <- xgboost(data = as.matrix(data_train[, c("exc_rate", "GBR",
"DEU", "FRA", "qtQ1", "qtQ2", "qtQ3", "qtQ4")]),
                      label = data_train$tr_inc, nrounds = 100, verbose = 0)

# Gradient Boosting Modeli Tahminleri 500k ekledik
xgb_predictionsQ <- predict(xgb_modelQ, newdata = as.matrix(data_test[,
c("exc_rate", "GBR", "DEU", "FRA", "qtQ1", "qtQ2", "qtQ3", "qtQ4")]))+500000
xgb_errorsQ <- data_test$tr_inc - xgb_predictionsQ
xgb_mseQ <- mean(xgb_errorsQ^2)
print("Gradient Boosting Model MSE:")

## [1] "Gradient Boosting Model MSE:"

print(xgb_mseQ)

## [1] 291812735625

# Gradient Boosting Modeli Tahminleri Görselleştirme
plot(data_test$tr_inc, type = "l", col = "blue", lwd = 2, main = "Gradient
Boosting Predictions with QT Dummies")
lines(xgb_predictionsQ-500000, col = "red", lwd = 2)
lines(xgb_predictionsQ, col = "green", lwd = 2, lty = 2)
legend("topleft", legend = c("Actual", "Predicted", "Predicted + 500k"), col
= c("blue", "red", "green"), lwd = 2)
```



```
predictedQ <- predict(xgb_modelQ, newdata = as.matrix(data_test[,
c("exc_rate", "GBR", "DEU", "FRA", "qtQ1", "qtQ2", "qtQ3", "qtQ4")]))
performanceQ <- data.frame(Actual = data_test$tr_inc, Predicted = predictedQ)
+ 500000
print(performanceQ)
```

```
##    Actual Predicted
## 1 1293360   1339309
## 2 1331884   1392934
## 3 3115955   1640768
## 4 2089754   1517065
## 5 1820052   1311327
## 6 1941766   1446396
## 7 3064155   1714953
## 8 2241179   1531061
```

Forecasting

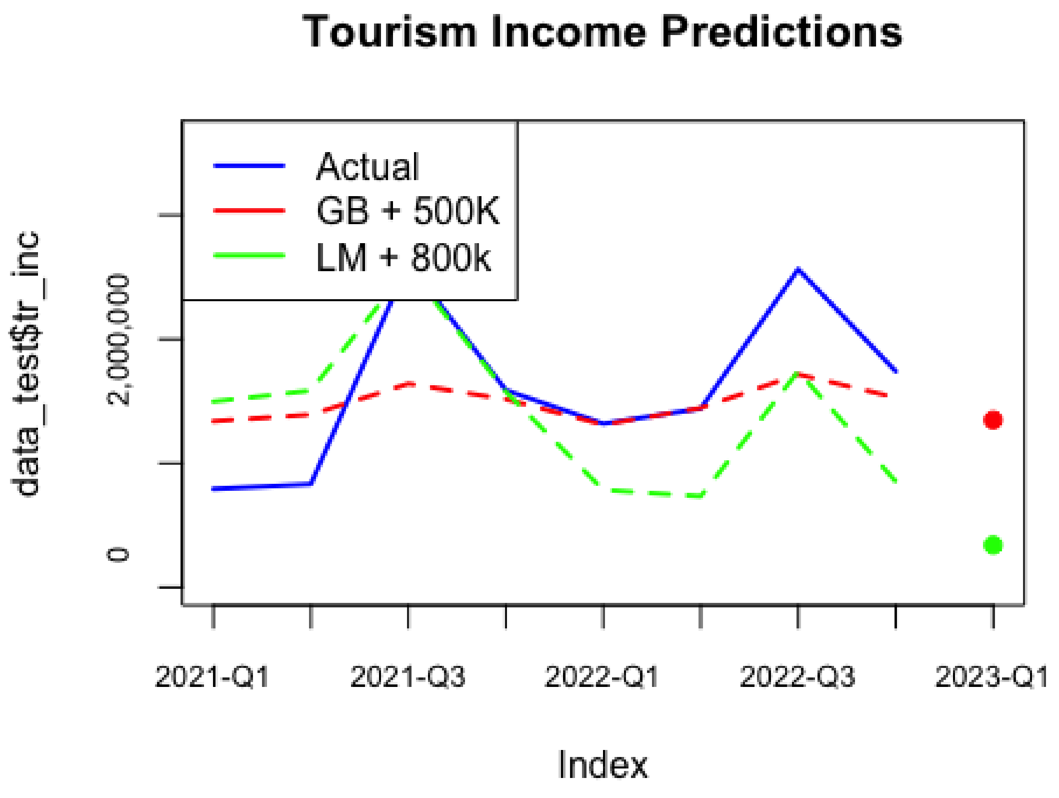
This code chunk aims to use the lm model and gradient boosting model to forecast tourism income for the first quarter of 2023 using the information provided in the “newtest.csv” file.

```
data2023q1 <- read_csv("newtest.csv") # 2023 q1 exc_rate gbr deu fra qtQ1
qtQ2 qtQ3 qtQ4

## Rows: 1 Columns: 10
## ── Column specification
────────────────────────────────────────────────────────
## Delimiter: ","
## chr (1): qt
## dbl (9): year, exc_rate, GBR, DEU, FRA, qtQ1, qtQ2, qtQ3, qtQ4
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.

data2023q1

## # A tibble: 1 × 10
##    year qt    exc_rate   GBR    DEU   FRA  qtQ1  qtQ2  qtQ3  qtQ4
##   <dbl> <chr>    <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  2023 Q1        18.9 0.125 0.0466 0.179     1     0     0     0

pred2023 <- predict(lm_model, newdata = data2023q1) + 800000

## Warning in predict.lm(lm_model, newdata = data2023q1): prediction from a
## rank-deficient fit may be misleading

pred2023

##        1
## 340520.6

# xgb23

xgb_p23 <- predict(xgb_modelQ, newdata = as.matrix(data2023q1[, c("exc_rate",
"GBR", "DEU", "FRA", "qtQ1", "qtQ2", "qtQ3", "qtQ4")])) + 500000
xgb_p23

## [1] 1349362
```

Plotting final predictions

```
# Create a vector of labels for the x-axis
x_labels <- c(paste(data_test$year, data_test$qt, sep = "-"), "2023-Q1")

# Plot the graph without default x-axis labels
plot(data_test$tr_inc, type = "l", col = "blue", lwd = 2, main = "Tourism
Income Predictions", xaxt = "n", yaxt = "n", xlim = c(1,
length(data_test$tr_inc) + 1), ylim = c(0, max(data_test$tr_inc, pred2023) +
1000000))
lines(xgb_predictionsQ, col = "red", lwd = 2, lty = 2)
lines(lm_predictions + 800000, col = "green", lwd = 2, lty = 2)
```

```
points(length(data_test$tr_inc) + 1, xgb_p23, col = "red", pch = 19)
points(length(data_test$tr_inc) + 1, pred2023, col = "green", pch = 19)
legend("topleft", legend = c("Actual", "GB + 500K", "LM + 800k"), col =
c("blue", "red", "green"), lwd = 2)

# Add custom x-axis labels
axis(1, at = 1:length(x_labels), labels = x_labels, cex.axis = 0.8, xaxt =
"s")

# Format the y-axis labels
axis(2, at = pretty(c(0, max(data_test$tr_inc, pred2023) + 1000000)), labels
= format(pretty(c(0, max(data_test$tr_inc, pred2023) + 1000000)), big.mark =
",", scientific = FALSE), cex.axis = 0.8, yaxt = "s")
```



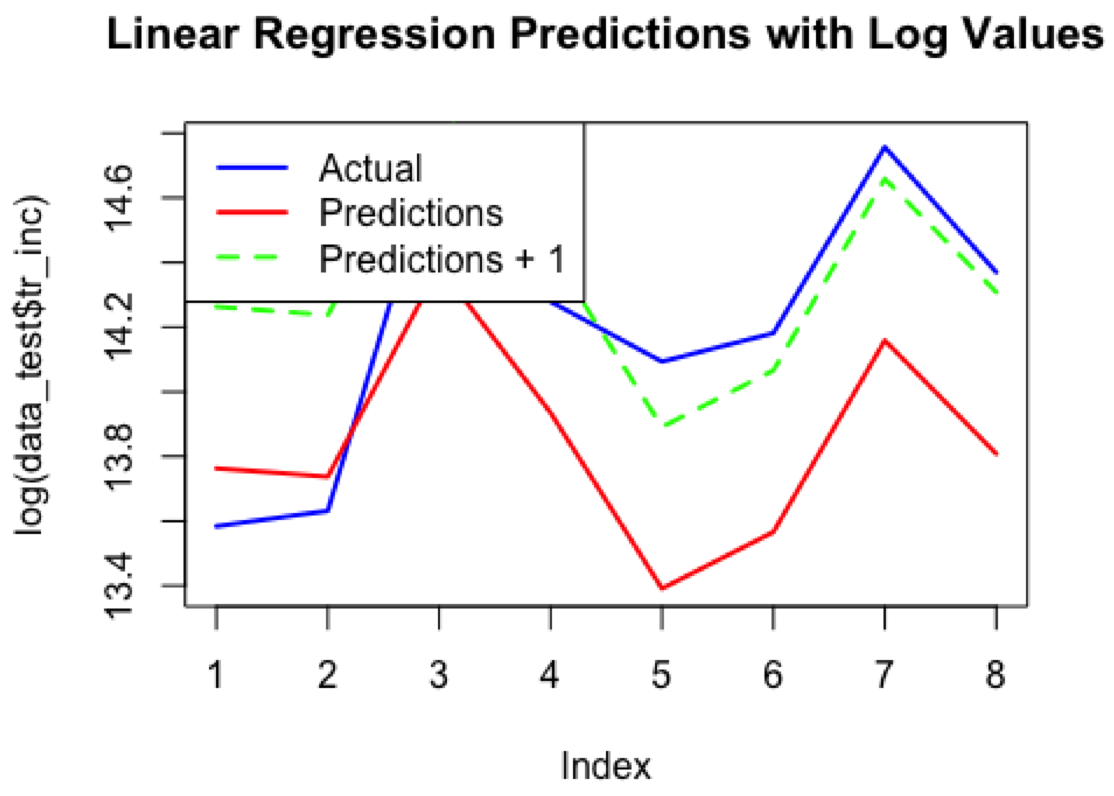
Using Log With Models

As log can not be applied to negative values (such as the ones in GDP's), offsets are defined. While in theory this could change the results, it can be observed that there were no major differences.

```r
offsetGBR <- abs(min(data_train$GBR)) + 1
offsetDEU <- abs(min(data_train$DEU)) + 1
offsetFRA <- abs(min(data_train$FRA)) + 1

# Apply log transformation with offset to variables
data_train$tr_inc_transformed <- log(data_train$tr_inc) # no addition of
offset
data_train$exc_rate_transformed <- log(data_train$exc_rate) # no addition of
offset

data_train$GBR_transformed <- log(data_train$GBR + offsetGBR)
data_train$DEU_transformed <- log(data_train$DEU + offsetDEU)
data_train$FRA_transformed <- log(data_train$FRA + offsetFRA)

data_test$exc_rate_transformed <- log(data_test$exc_rate)
data_test$GBR_transformed <- log(data_test$GBR + offsetGBR)
data_test$DEU_transformed <- log(data_test$DEU + offsetDEU)
data_test$FRA_transformed <- log(data_test$FRA + offsetFRA)

data2023q1$exc_rate_transformed <- log(data2023q1$exc_rate)
data2023q1$GBR_transformed <- log(data2023q1$GBR + offsetGBR)
data2023q1$DEU_transformed <- log(data2023q1$DEU + offsetDEU)
data2023q1$FRA_transformed <- log(data2023q1$FRA + offsetFRA)
```

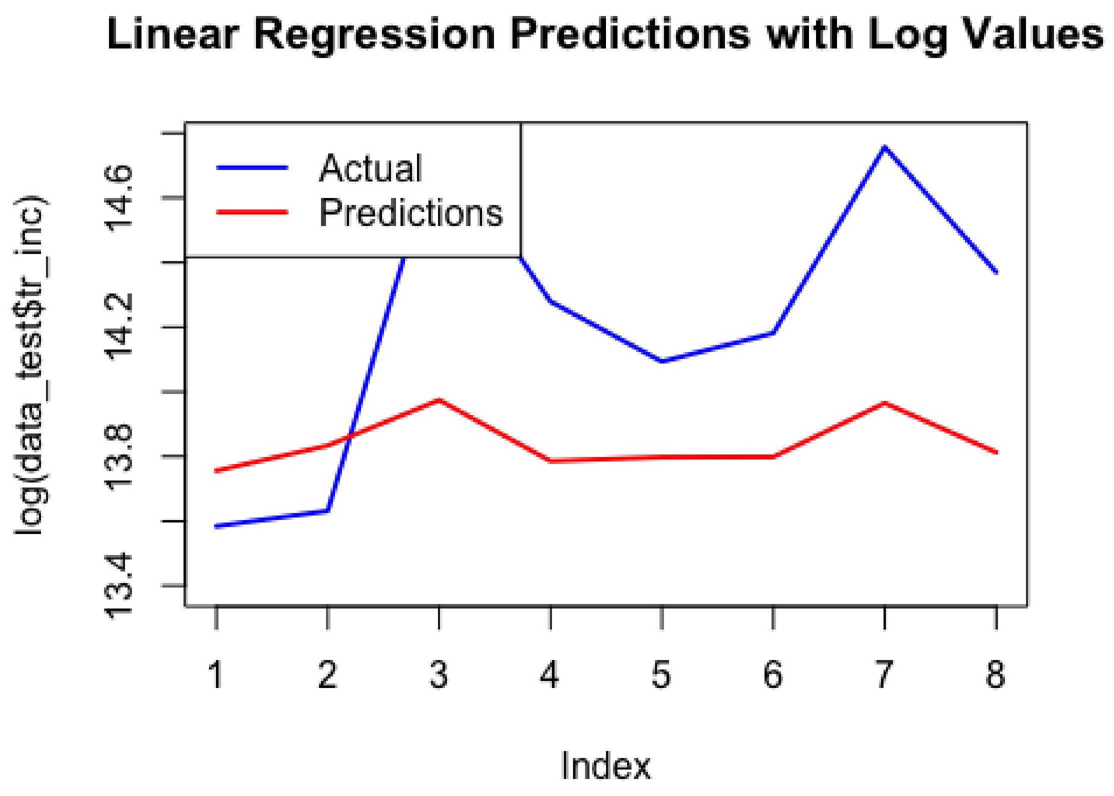
LM with Log

```r
lm_modelLog <- lm(log(tr_inc) ~ exc_rate_transformed + GBR_transformed +
DEU_transformed + FRA_transformed + qtQ1 + qtQ2 + qtQ3 + qtQ4, data =
data_train)

lm_predictionsLog <- predict(lm_modelLog, newdata = data_test)

## Warning in predict.lm(lm_modelLog, newdata = data_test): prediction from a
## rank-deficient fit may be misleading

lm_predictionsLog23 <- predict(lm_modelLog, newdata = data2023q1)

## Warning in predict.lm(lm_modelLog, newdata = data2023q1): prediction from
a
## rank-deficient fit may be misleading

plot(log(data_test$tr_inc), type = "l", col = "blue", lwd = 2, main = "Linear
Regression Predictions with Log Values", ylim = c(min(lm_predictionsLog),
max(log(data_test$tr_inc))))
lines(lm_predictionsLog, col = "red", lwd = 2)
lines(lm_predictionsLog+0.5, col = "green", lwd = 2, lty = 2)  # Add lty = 2
for dashed line
legend("topleft", legend = c("Actual", "Predictions", "Predictions + 1"), col
= c("blue", "red", "green"), lwd = 2, lty = c(1, 1, 2))
```



XGB with Log

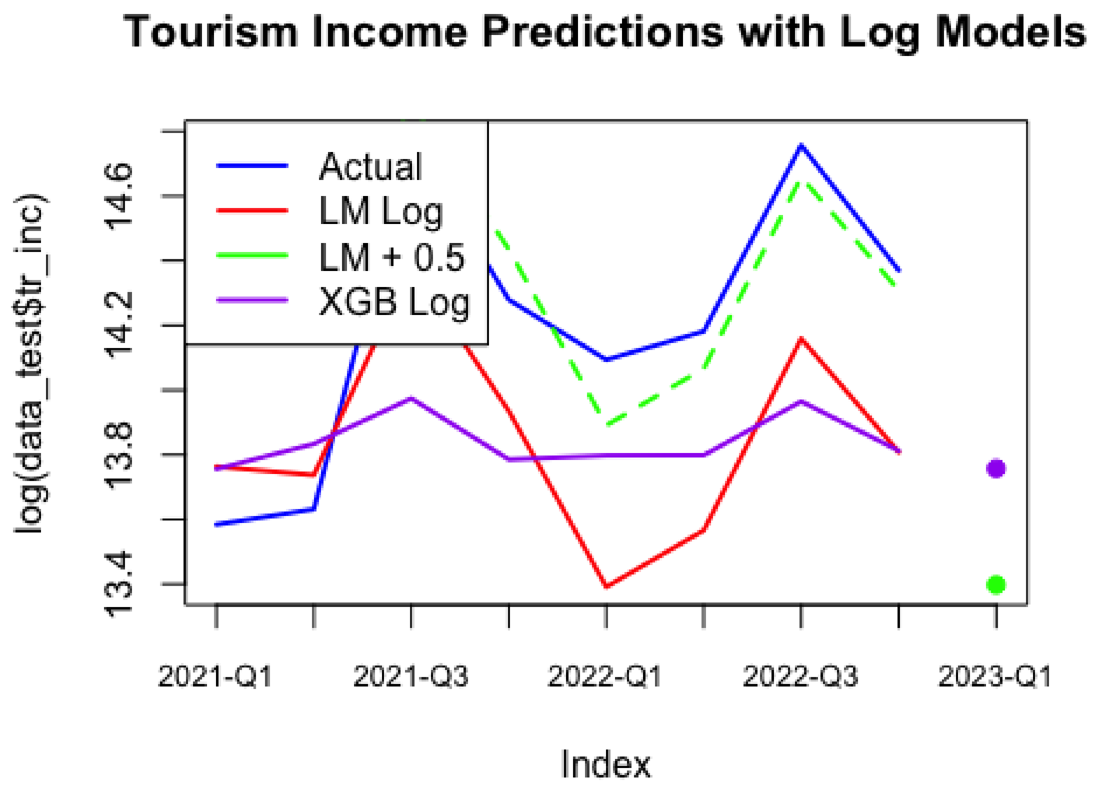
```
xgb_modelLog <- xgboost(data = as.matrix(data_train[,
c("exc_rate_transformed", "GBR_transformed", "DEU_transformed",
"FRA_transformed", "qtQ1", "qtQ2", "qtQ3", "qtQ4")]),
                      label = data_train$tr_inc_transformed, nrounds = 100,
verbose = 0)
xgb_predictionsLog <- predict(xgb_modelLog, newdata = as.matrix(data_test[,
c("exc_rate_transformed", "GBR_transformed", "DEU_transformed",
"FRA_transformed","qtQ1", "qtQ2", "qtQ3", "qtQ4")]))
xgb_predictionsLog23 <- predict(xgb_modelLog, newdata =
as.matrix(data2023q1[, c("exc_rate_transformed", "GBR_transformed",
"DEU_transformed", "FRA_transformed","qtQ1", "qtQ2", "qtQ3", "qtQ4")]))

plot(log(data_test$tr_inc), type = "l", col = "blue", lwd = 2, main = "Linear
Regression Predictions with Log Values", ylim = c(min(lm_predictionsLog),
max(log(data_test$tr_inc))))
lines(xgb_predictionsLog, col = "red", lwd = 2)
legend("topleft", legend = c("Actual", "Predictions"), col = c("blue", "red",
"green"), lwd = 2, lty = c(1, 1, 2))  # Add lty = c(1, 1, 2) for line types
```



Forecasting 2023-Q1 Using Models with Log

```
# Create a vector of labels for the x-axis
x_labels <- c(paste(data_test$year, data_test$qt, sep = "-"), "2023-Q1")

# Plot the graph without default x-axis labels
plot(log(data_test$tr_inc), type = "l", col = "blue", lwd = 2, main =
"Tourism Income Predictions with Log Models", xaxt = "n", xlim = c(1,
length(data_test$tr_inc) + 1), ylim = c(min(lm_predictionsLog),
max(log(data_test$tr_inc))))
lines(lm_predictionsLog, col = "red", lwd = 2)
lines(lm_predictionsLog + 0.5, col = "green", lwd = 2, lty = 2)
lines(xgb_predictionsLog, col = "purple", lwd = 2)
points(length(data_test$tr_inc) + 1, lm_predictionsLog23, col = "green", pch
= 19)
points(length(data_test$tr_inc) + 1, xgb_predictionsLog23, col = "purple",
pch = 19)
legend("topleft", legend = c("Actual", "LM Log", "LM + 0.5", "XGB Log"), col
= c("blue", "red", "green", "purple"), lwd = 2)

# Add custom x-axis labels
```

```
axis(1, at = 1:length(x_labels), labels = x_labels, cex.axis = 0.8, xaxt =
"s")
```



Other Trials

There have many attempts to try different models. These trials include incorporating growth rate to the models. However, no meaningful data was obtained using that method.