Forecasting GDP Growth

Is Accounting Information Useful to Predicting US GDP Growth?

AUTHOR
Chea Oussa, Du Qinyi, Luo Zeming, Zang Linlin, Zhang Qiqi

PUBLISHED
March 17, 2024

Predicting US GDP Growth:
Is Accounting Information Useful to Predicting US GDP Growth?

Students' Names:

CHEA Oussa (01473752)

Du Qinyi (01496375)

Luo Zeming (01476755)

Zang Linlin (01496340)

Zhang Qiqi (01475317)

Class Section: ACCT653

Instructor's Name: Professor Wang Jiwei

Date of Submission: March 2024

I. Introduction

In the past few decades, technological and financial advancements in various fields and industries have reshaped our globalized economy, bringing about substantial transformations to the economic health in nations around the world. Among these countries, the United States of America has stood out as a significant player; not only are they often the first to be affected by disruptive innovations, but they are often the source of said disruptions as well. With the ability to cause widespread economic effects between its own borders, the US serves as an interesting research topic of an ever-evolving economy that is more than prone to disruptions. Case in point, the 2008 financial crisis which saw economic fluctuations that not only rocked the United States, but the rest of the world as well.

Since the United States plays such a key role in the economy of the world, as well as the millions of its own citizens, economists and businesses alike strive to comprehend the country's ever changing economic welfare to adapt to it. The ability to forecast the economic health of the United States would therefore allow both policymakers and business owners alike to make sound strategic decisions regarding fiscal policies and investment activities respectively.

To this end, the forecasting of one key economic indicator springs to mind - the Gross Domestic Product (GDP) - or more specifically, the GDP Growth of the United States. Since GDP by itself is a very intuitive indicator to understand, the growth rate of real GDP is often used as a proxy to gauge the health of the US economy1; whether through its association with employment rate (Sánchez & Liborio, 2012) or relation to the overall welfare of the population (Dynan & Sheiner, 2018). As such, despite its rather basic and common usage in economic studies across the globe, the ubiquity of GDP growth in said studies is a reminder of just how useful the indicator is to economists worldwide and in helping various stakeholders in their decision-making. And towards this, two 'types' of predictor variables have been utilized to forecast the GDP growth of the US economy – the more widely known macroeconomic indicators, and the less utilized, yet still as informational, accounting information.

A. Literature Review

Historically, attempts to forecast GDP growth have incorporated a myriad of macroeconomic, environmental, and market factors to properly represent the various components of a country's economy. Since GDP reflects the total value of goods and services produced within a nation, it follows that examining the income level per capita can offer insights into the general trajectory of GDP growth. As such, research conducted in China has indicated that per capita income level serves as a reliable predictor of future economic trends (Feng Xuebin et al. 2018, Zheng Yi 2020). This conclusion was corroborated by additional studies, which suggest a close relationship between GDP growth rate, population size, and productivity levels within a country (Li & Wei, 2023). Specifically, these studies highlight a positive correlation between a nation's per capita output and income and its GDP growth.

In addition to the factors mentioned above, the influence of household consumption and consumer demand on GDP growth has been the subject of numerous studies as well, particularly examining both the short-term and long-term effects on the economies of developed nations like China and the United States. Research by Du Junli (2016), Wang Weiwei (2016), and Wan Jiejie (2013) all indicated that while household consumption is positively correlated with a nation's long-term economic growth, this impact becomes amplified when the focus is on short-term economic growth instead. Moreover, there exists a complex relationship between consumer perceptions and economic conditions, including the pressures of inflation. Pazzanese (2021) delves into this phenomenon, highlighting the influence of changes in consumer sentiment on economic health. Factors such as consumer confidence, income levels, and credit conditions can alter spending behaviors, with fluctuations in consumer sentiments directly affecting economic performance in noticeable ways.

Considering the significant impact that household consumption and consumer behavior have on GDP growth, it is crucial to address the potential effects that inflation would have as well. Generally, the dynamics between inflation and economic growth are complex. Classical theories tend to downplay the impact of inflation on real GDP, whereas New Keynesian theories suggest that moderate inflation might spur short-term economic growth given specific circumstances. However, the concept of stagflation—characterized by high inflation alongside sluggish growth—poses a challenge to these theories. Additionally, inflation can result in losses for the banking sector, subsequently leading to reduced lending and a negative impact on economic activities (Exploring the Link between Rising Inflation and Economic Growth, 2023; Agarwal & Baron, 2023).

Furthermore, the significance of employment on a country's economic growth cannot be overstated, given the close correlation between the two (Li Yanfu, 2019). While one school of thought suggests that the employment structure within the population is influenced by economic growth, it can also be argued that, to a certain extent, a country's employment rate positively influences its development and stability instead. This is further supported by Okun's Law, coined by economist Arthur Okun in the 1960s, which posits an inverse relationship between changes in aggregate economic output and changes in the unemployment rate. Empirical evidence also underscores the substantial negative impact of high unemployment on GDP. As such, policies aimed at reducing unemployment, particularly among youth, are vital for fostering economic growth (Wen & Chen, 2012; Shiferaw, 2023).

Continuing the focus on employment, debates persist regarding the relationship between real wages and GDP growth, particularly concerning the challenges in translating economic progress into improved living standards. Despite GDP in the US doubling from the period 1972 to 2014, real wages generally remained stagnant, only increasing by 17%, indicating difficulties for workers to benefit from economic expansion2. Moreover, it can be

argued that stagnant or declining real wages may constrain household consumption, thereby hindering GDP growth.

Another pivotal factor to consider is international trade and investment. According to Osano and Koine (2016), foreign direct investment (FDI) and international trade play critical roles for the transfer of technology from developed to developing countries. They stimulate domestic investment and improve human capital and institutional frameworks within host countries. Moreover, research on the impact of international trade suggests that compared to the significant influence that consumption exerts on economic growth, imports and exports have relatively weaker effects (Wan Jiejie, 2013). However, it's worth noting that this could be attributed to the repercussions of events like the European debt crisis, which led to trade setbacks between countries in the European Union during the period studied.

Moreover, extensive research has been conducted on the variation in economic growth among the different states in the United States, with studies by Barro & Sala-i-Martin (1992), Razzolini & Shughart (1997), and Jayaratne & Strahan (1996) focusing on the effects of state-level fiscal policies. These studies collectively suggest that state-level fiscal policies, encompassing government size, spending, and tax burdens, wield significant influence over GDP growth.

Additionally, monetary policies, overseen and adjusted by central banks, also impact economic growth. Specifically, research on interest rates indicates that fluctuations in interest rates affect the cost of borrowing, thereby influencing investment decisions and aggregate demand. Higher interest rates can suppress investment activity and consumption, potentially leading to a decline in economic growth. Conversely, higher interest rates might attract foreign capital inflows, impacting currency exchange rates. Empirical studies demonstrate that interest rate hikes have a moderately negative impact on real growth across European countries (Davcev et al., 2018; di Giovanni & Levchenko, 2012). Adding on to this, adjustments of currency in circulation through monetary policies changes can also affect GDP, as evidenced by Cox (2021), who found that a surge in currency in circulation may indicate increased economic activities.

In addition to financial factors from a macroeconomic standpoint, another crucial aspect to consider is the economic structure of a country. Khan (2010) asserts that the economic structure plays a pivotal role in determining GDP growth. Sectors with increasing returns, such as technology industries, are particularly influential in supporting robust growth due to their potential for innovation, high income, and price elasticity, and positive externalities. Victor Mulas, an ICT Policies Specialist at the World Bank, supported this notion by underscoring the importance of a region's economic structure in forecasting its GDP. He suggests that the diversification and complexity of economic activities, especially those showing increasing returns, are essential for maintaining sustained economic growth.

And yet, another often overlooked macroeconomic factor to consider is a country's energy consumption. Despite being non-financial in nature, the relationship between energy consumption and economic growth has undergone significant changes over time. Recent trends suggest a decoupling of GDP growth from energy consumption due to improving efficiencies and transitions towards alternative energy sources, allowing growth in a country's economy with reduced reliance on energy consumption. However, it is important to note that energy shortages can still impede GDP growth, and wealthier nations typically demonstrate higher energy consumption per capita.

However, despite this wealth of research into finding the macroeconomic variables with the best predictive power, one aspect that is often overlooked in forecasting a country's GDP growth is the role of accounting information. A growing body of research seeks to explore the effectiveness of this type of variable, such as Yaniv Konchitchki and Panos N. Patatoukas, who investigated the informativeness of accounting earnings for GDP growth (Konchitchki & Patatoukas, 2014) and discovered that aggregate accounting earnings growth serves as a leading indicator of the U.S. economy. Their study revealed that aggregate accounting earnings growth predicts GDP growth, particularly for the one-quarter-ahead forecast horizon.

Furthermore, Sumiyana examined the association between aggregate accounting earnings and future GDP growth across 39 Asia-Pacific and African countries from 1989 to 2015 (Sumiyana, 2020), and found that aggregate accounting earnings only forecast future GDP growth in eight developed countries where earnings growth is positive. Building on this research, Srikant Datar explored the extent to which accounting variables contribute to improving the accuracy of GDP growth forecasts (Datar et al., 2020). The study revealed that while accounting information doesn't significantly enhance the out-of-sample accuracy of predictions for near-term GDP growth (current and next quarter), it becomes more useful when forecasting more distant-term GDP growth (three and four-quarters-ahead). Specifically, four categories of accounting variables, relating to Profits, Accrual Estimates, Capital Raises or Distributions, and Capital Allocation decisions, are identified as the most informative for the longer-term economic outlook.

B. Research Question and Hypotheses

With all the information and past studies about the prediction of GDP growth laid out above, this report's main research question will focus the additive predictive power of accounting information to predictive models that only consist of macroeconomic variables. Simply put, the main hypothesis of this report is that the machine learning model that includes both macroeconomic and accounting variables will have better predictive power compared to a model that only has macroeconomic variables.

The hypothesis is as follows:

Ha: The inclusion of accounting information into a machine learning model will improve the predictive power[1] *when forecasting the current-quarter's GDP Growth of the United States*

C. Report Structure

This report will consist of the follow structure:

I. Introduction

II. Methodology

  i. Data Collection

  ii. Variable and Features Selection

III. Data Analysis

  A. Exploratory Data Analysis

  B. Regression Analysis

  C. Models' Evaluations

IV. Discussion

  A. Interpretation

  B. Sensitivity Analysis

  C. Limitations of Research

V. Conclusion

II. Methodology

A. Data Collection

a. Sources of Data

The 13 different datasets used in this analysis are extracted from various sources to account for both macroeconomic and accounting-based information. The dependent variables, consisting of Year-Quarter groups and their respective GDP growth for the year, were given by Kaggle.

All the datasets containing accounting information are sourced from Wharton Research Data Services, specifically from its CRSP database. To facilitate ease of data cleaning, feature engineering, and analysis, a pre-merged quarterly dataset provided in the CRSP database was used, consisting of US-based firms' accounting information (from Compustat database) and US-based firms' stock information (from CRSP database).

As for the datasets containing macroeconomic information, due to the varied sources and varying levels of usefulness, a table has been provided in the appendix, summarizing the information regarding the datasets[2].

b. Data Cleaning and Merging

Due to the number of datasets that have been sourced by the team, this data cleaning process first starts through by selecting the data that is relevant to the or has been shown to have predictive strength when forecasting GDP through the above literature review. Afterwards, datasets are then filtered based on whether they cover the timespan of this research (2015 to 2020), have enough variability between each observation (Yearly data is the limit for variability), and are relevant to the United States of America.

Furthermore, due to the difference in each dataset's format, each dataset concerning macroeconomic variables is cleaned and their formats standardized before being aggregated into a quarterly basis (Year-Quarter grouping).

As for the dataset containing accounting information, feature engineering is performed on the dataset in order to get the appropriate accounting measures. Afterwards, if any features in the dataset display more than 50% missing values, we discard those features as we assumed that imputing those missing data will lead to an imbalance in the dataset. Then winsorization at the 1% level is performed for all the remaining features to reduce effects from outliers. And finally, any data still missing from the features are progressively filled in with the average observations of that feature through the below groupings:

- Company (Gvkey)-Year group
- Year-Quarter group
- Year
- Company (Gvkey)
- Standard Industry Classification group (sic)

Then the accounting information data is aggregated on a quarterly basis (Year-Quarter grouping) before.

Finally, both the macroeconomic and the accounting information datasets are joined together with the provided training data (consisting of Year-Quarter grouping and GDP growth for the year) by Year-Quarter or Year where applicable

Please note that running the entirety of this report's code chunks will take more than 3 minutes.

Data Loading, Cleaning, and Preparation

Part 1: Loading in packages and custom functions:

```
#1. Load libraries and environment
library(tidyverse)
library(readr)
library(dplyr)
library(psych)
library(ggplot2)
library(caret)
library(lfe)
library(broom)
library(stargazer)
library(data.table)
library(corrplot)
library(car)
library(boot)
library(leaps)
library(readxl)
library(useful)
library(Matrix)
library(vtable)
library(lubridate)
library(zoo)
library(ROCR)
library(glmnet)
library(coefplot)
library(xgboost)
library(xgboostExplainer)
library(ParBayesianOptimization)
library(caret)
library(glmnetUtils)
library(corrplot)
library(randomForest)

options(scipen=999, digits=4) # avoid scientific display, keep 4 digits in display

# # Please set your own directory here
# dir_data <- "./Data_Code/"
# setwd(dir_data)
# dir()

#3. Load common functions

## Convert infinite cases to NA
finitize <- function(x) ifelse(is.finite(x), x, NA)
###e.g. denominator is zero

## Merge variables listed in ... by index_suffix
mergify <- function(x, index, suffix, ...){
  require(dplyr)
  x_vars <- x[c(index, ...)]
  colnames(x_vars) <- paste(colnames(x_vars), suffix, sep = "_")
  left_join(x, x_vars, by = paste(index, suffix, sep = "_"))
}

## Reduce dataframe to complete cases
completify <- function(x, ...)  x[complete.cases(x[c(...)]),]

## Replace NA observations with zeroes
zerofy <- function(x) ifelse(is.na(x), 0, x)

## An ifelse function that does not cause dates to lose their class

safe_ifelse <- function(cond, yes, no) {
  structure(ifelse(cond, yes, no), class = class(yes))
}

## A function that computes certain quantiles and other useful statistics
inspectify <- function(x, d = 3){
  qt = quantile(x, c(0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99), na.rm = TRUE)
  nums = c(min(x, na.rm = TRUE), qt, max(x, na.rm = TRUE), sum(is.na(x)))
  names(nums) = c("Min", names(qt), "Max", "NA")
  round(nums, digits = d)
}

## Trims (sets to NA) observations beyond quantiles [trim, 1-trim]
```

```r
trimify <- function(x, trim = 0.01) {
  ifelse(
    x > quantile(x, 1 - trim, na.rm = TRUE, type = 3)
    | x < quantile(x, trim, na.rm = TRUE, type = 3),
    NA, x
  )
}

## Converts characters to dates with specified format
datify <- function(x, f) as.Date(as.character(x), format = f)

## Define html_df() function for displaying small tables in html format
libs <- c("knitr", "kableExtra")  # table styling
invisible(lapply(libs, library, character.only = TRUE))
html_df <- function(text, cols = NULL, col1 = FALSE, full = FALSE) {
  if(!length(cols)) {
    cols = colnames(text)
  }
  if(!col1) {
    kable(text,"html", col.names = cols, align = c("l", rep('c', length(cols)-1))) %>%
      kable_styling(bootstrap_options = c("striped", "hover"), full_width = full)
  } else {
    kable(text, "html", col.names = cols, align = c("l", rep('c', length(cols) - 1))) %>%
      kable_styling(bootstrap_options = c("striped", "hover"), full_width = full) %>%
      column_spec(1, bold = TRUE)
  }
}

# Calculate missing percentage and display using html_df
calculate_missing_percentage <- function(df, vars_of_interest) {
  # Calculate the percentage of missing values in each column
  missing_percentage <- colMeans(is.na(df)) * 100

  # Create a new dataframe
  missing_df <- data.frame(Column = names(missing_percentage), Missing_Percentage = missing_perce

  # Subset the dataframe for variables of interest
  missing_df <- missing_df[missing_df$Column %in% vars_of_interest, ]

  # Format the dataframe as HTML
  formatted_df <- missing_df %>%
    arrange(Missing_Percentage) %>%
    html_df()

  # Return the formatted dataframe
  return(formatted_df)
}

# Min-Max scaling function
min_max_scaling <- function(x) {
  (x - min(x)) / (max(x) - min(x))
}

# Z-score normalization function
z_score_normalization <- function(x) {
  (x - mean(x)) / sd(x)
}

# Calculation of RMSE
cal_rmse = function(pred,y){
  rmse = sqrt(mean((pred - y)^2))
  return(rmse)
}
```

Part 2: Loading then saving data to RData format for easier loading

```r
# set eval=TRUE to run this chunk, it will take more than 1 minute

#4. Load data

## Load Training and Testing GDP data
train_df <- fread("Data/GDP_Forecast_train.csv", data.table = FALSE)
test_df <- fread("Data/GDP_Forecast_test.csv", data.table = FALSE)


## Load independent variables data

#Fundamental annual data merged with stock information
funda_merged_q <- fread("Data/merged_quarterly.csv", data.table = FALSE)

#gdp forecasts
#gdp_forecast_indv <- read_xlsx("Data/gdp_forecast_indv.xlsx")
gdp_forecast_mean <- read_xlsx("Data/gdp_forecast_mean.xlsx")
```

```r
#net income
net_income <- read_xls("Data/adj_net_income_per_capita.xls")
#consumer spending
spending <- fread("Data/consumer_spending.csv", data.table = FALSE)
cpi <- fread("Data/CPI.csv", data.table = FALSE)
#energy consumption
energy_residential <- fread("Data/energy_consumption_residential_commercial_industrial.csv", data
energy_transport <- fread("Data/energy_consumption_transportation.csv", data.table = FALSE)
#federal interest rates
fed_rates <- fread("Data/fed_funda_rates.csv", data.table = FALSE)
# real earnings
weekly_earnings <- fread("Data/median_weekly_real_earnings.csv", data.table = FALSE)
#unemployment
unemployment <- fread("Data/unemployment_rate.csv", data.table = FALSE)
#demographics
demographic <- fread("Data/usa_demographic_by_region.csv", data.table = FALSE)
#household consumption
household_consumption <- read_xls("Data/usa_household_consumption_per_capita.xls")
#gross savings
gross_saving <- read_xls("Data/gross_savings_investments.xls")

#m1 and m2 is the measure of cash/liquity of USA, m2 is more broad, including investments
#while m1 only include cash circulation
liquidity <- fread("Data/M2.csv", data.table = FALSE)

# Import and Export data
import <- read_xls("Data/USA_Imports.xls")
export <- read_xls("Data/USA_Export.xls")

# save files for easier loading
save(train_df, file = "RData/train_df.RData")
save(funda_merged_q, file = "RData/funda_merged_q.RData")
save(gdp_forecast_mean, file = "RData/gdp_forecast_mean.RData")
save(net_income, file = "RData/net_income.RData")
save(cpi, file = "RData/cpi.RData")
save(energy_residential, file = "RData/energy_residential.RData")
save(energy_transport, file = "RData/energy_transport.RData")
save(fed_rates, file = "RData/fed_rates.RData")
save(weekly_earnings, file = "RData/weekly_earnings.RData")
save(unemployment, file = "RData/unemployment.RData")
save(household_consumption, file = "RData/household_consumption.RData")
save(liquidity, file = "RData/liquidity.RData")
save(gross_saving, file = "RData/gross_saving.RData")
save(import, file = "RData/import.RData")
save(export, file = "RData/export.RData")

save(test_df, file = "RData/test_df.RData")
```

Part 3: Perform Data cleaning on macroeconomic variables

This step is separate from the accounting variables data cleaning because the macro economics variable are easier to clean up and often has 0 missing values.

```r
# set eval=TRUE to run this chunk

## 1. Load GDP training and testing data
load("RData/train_df.RData")

#Extract Year from dataset
train_df <- train_df %>%
  mutate(YEAR = as.numeric(substr(YQ, 1, 4)))
# Final obs = 104 obs
save(train_df, file = "RData/train_df.RData")


## 2. Load mean GDP forecast for cleaning

load("RData/gdp_forecast_mean.RData")
nrow(gdp_forecast_mean) # 221 obs
# Create YQ column for later merge
gdp_forecast_mean <- gdp_forecast_mean %>% mutate(YQ = paste0(YEAR, "Q", QUARTER))
sum(duplicated(gdp_forecast_mean$YQ)) #0 duplicates
# Column NGDP2 is the forecast for the current Quarter
gdp_forecast_mean <- gdp_forecast_mean %>% select(YEAR, QUARTER, YQ, NGDP2)
# Calculate percentage growth of gdp forecast
gdp_forecast_mean <- gdp_forecast_mean %>%
  mutate(forecast_growth = (NGDP2/lag(NGDP2) - 1) * 100)
# Final obs = 221 obs
save(gdp_forecast_mean, file = "RData/gdp_forecast_mean.RData")

## 3. Load net income data for cleaning
load("RData/net_income.RData")
```

```r
# Convert net_income wide form into long form then only select for USA data while cleaning up col
net_income <- dplyr::slice(net_income,3:n())
colnames(net_income) <- unlist(net_income[1, ])
net_income <- net_income[-1, ]
net_income <- net_income %>% filter(`Country Code` == "USA" )
net_income <- net_income %>% rename(CountryName = `Country Name`, CountryCode = `Country Code`)
net_income <- pivot_longer(net_income, cols = -c(CountryName,CountryCode),
                           names_to = "YEAR", values_to = "income_per_capita")
net_income <- dplyr::slice(net_income, 3:n())
net_income$CountryName <- NULL
net_income <- net_income %>%
  mutate(income_per_capita = as.numeric(income_per_capita),
         YEAR = as.numeric(YEAR))
# net_income data has no quarter data so special caution will be taken when merging

# Calculate quarter on quarter net income growth
net_income <- net_income %>%
  mutate(income_per_capita_growth = (income_per_capita/lag(income_per_capita) - 1) * 100) %>%
  select(-CountryCode)
# Final obs = 63 obs
save(net_income, file = "RData/net_income.RData")

## 4. Load CPI data for cleaning
load("RData/cpi.RData")
# Create Year, Month, and Quarter data
cpi <- cpi %>% mutate(YEAR = year(DATE), MONTH = month(DATE))
cpi <- cpi %>%
  mutate(QUARTER = case_when(
    MONTH <= 3 ~ 1,
    MONTH <= 6 ~ 2,
    MONTH <= 9 ~ 3,
    TRUE ~ 4
  ))
# Create Year-Quarter column for later merging
cpi <- cpi %>% mutate(YQ = paste0(YEAR, "Q", QUARTER))
# 912 obs

# Group by Year-Quarter and calculate the average CPI for eachq uarter
cpi <- cpi %>% group_by(YQ) %>%
  mutate(CPI = mean(CPIAUCSL_PC1)) %>%
  dplyr::slice(1) %>%
  ungroup()
# Select only YQ and CPI data
cpi <- cpi %>% select(YQ, YEAR, CPI)
# Final obs = 304 obs
save(cpi, file = "RData/cpi.RData")

## 5. Load energy data for cleaning
load("RData/energy_residential.RData") #10,260 rows
load("RData/energy_transport.RData") #8,208 rows
# Cleaning and setting up residential energy consumption data

# Only take total energy for Residential, Commercial and Industrial usage
energy_residential <- energy_residential %>%
  filter(Column_Order == 5 | Column_Order == 10 | Column_Order == 15) %>%
  pivot_wider(names_from = Column_Order, values_from = Value, values_fill = NA) %>%
  rename(residential = "5", commercial = "10", industrial = "15") #2052 rows

energy_residential <- energy_residential %>%
  group_by(YYYYMM) %>%
  mutate(residential = sum(residential, na.rm = TRUE),
         commercial = sum(commercial, na.rm = TRUE),
         industrial = sum(industrial, na.rm = TRUE)) %>%
  dplyr::slice(1) %>%
  ungroup() # 684 rows

# Create Year, Month, and Quarter columns for later merging
energy_residential <- energy_residential %>%
  mutate(YEAR = as.double(substr(YYYYMM, 1, 4)),
         MONTH = as.double(substr(YYYYMM, 5, 6)),
         QUARTER = case_when(
           MONTH <= 3 ~ 1,
           MONTH <= 6 ~ 2,
           MONTH <= 9 ~ 3,
           TRUE ~ 4)) %>%
  mutate(YQ = paste0(YEAR, "Q", QUARTER))

# Remove rows in the 13th month and aggregate the energy consumption quarterly
energy_residential <- energy_residential %>%
  select(!c(MSN, Description, Unit, YYYYMM)) %>%
  filter(MONTH != 13) %>%
  group_by(YQ) %>%
  mutate(residential = mean(residential),
         commercial = mean(commercial),
```

```r
        industrial = mean(industrial)) %>%
  dplyr::slice(1) %>%
  ungroup()
# 204 rows

# Clean up transport energy consumption in same manner
energy_transport <- energy_transport %>%
  filter(Column_Order == 5) %>% # Get only column 5 because it is energy use in transport sector
  pivot_wider(names_from = Column_Order, values_from = Value, values_fill = NA) %>%
  rename(transport = "5") #684 rows

# Create Year, Month, and Quarter columns for later merging
energy_transport <- energy_transport %>%
  mutate(YEAR = as.double(substr(YYYYMM, 1, 4)),
         MONTH = as.double(substr(YYYYMM, 5, 6)),
         QUARTER = case_when(
           MONTH <= 3 ~ 1,
           MONTH <= 6 ~ 2,
           MONTH <= 9 ~ 3,
           TRUE ~ 4)) %>%
  mutate(YQ = paste0(YEAR, "Q", QUARTER))

# Remove rows in the 13th month and aggregate the energy consumption quarterly
energy_transport <- energy_transport %>%
  select(!c(MSN, Description, Unit, YYYYMM)) %>%
  filter(MONTH != 13) %>%
  group_by(YQ) %>%
  mutate(transport = mean(transport)) %>%
  dplyr::slice(1) %>%
  ungroup() #204 obs

# Join both data
energy_consumption <- left_join(energy_residential, energy_transport, by = "YQ") %>%
  select(YQ, YEAR.x, residential, commercial, industrial, transport) %>%
  rename(YEAR = "YEAR.x") %>%
  mutate(total = residential + commercial + industrial + transport)

# Calculate percentage changes quarter on quarter for each type of energy consumption

energy_consumption <- energy_consumption %>%
  mutate(across(
    c(residential, commercial, industrial, transport, total),
    ~ (./lag(.) - 1) * 100,
    .names = "{.col}_energy_change"
  ))

# Final obs = 204 obs
save(energy_consumption, file = "RData/energy_consumption.RData")

## 6. Load federal interest rates data
# FEDFUNDS is column is in %
load("RData/fed_rates.RData")

# Extract Year, Month, and Quarter from the dataframe
fed_rates <- fed_rates %>%
  mutate(YEAR = year(DATE),
         MONTH = month(DATE),
         QUARTER = case_when(
           MONTH <= 3 ~ 1,
           MONTH <= 6 ~ 2,
           MONTH <= 9 ~ 3,
           TRUE ~ 4)) %>%
  mutate(YQ = paste0(YEAR, "Q", QUARTER)) #835 obs

# Aggregate average federal interest rates by quarter
fed_rates <- fed_rates %>%
  group_by(YQ) %>%
  mutate(rate = mean(FEDFUNDS)) %>%
  dplyr::slice(1) %>%
  ungroup() #279 obs

# Select relevant columns and changes through each quarter
fed_rates <- fed_rates %>%
  select(YQ, YEAR, rate) %>%
  mutate(rate_change = rate - lag(rate)) %>%
  rename(fed_rate = "rate", fed_rate_change = "rate_change")
# Final obs = 279 obs
save(fed_rates, file = "RData/fed_rates.RData")

## 7. Load weekly earnings data
load("RData/weekly_earnings.RData")

# Extract Year, Month, and Quarter from dataframe
weekly_earnings <- weekly_earnings %>%
```

```r
  mutate(YEAR = year(DATE),
         MONTH = month(DATE),
         QUARTER = case_when(
           MONTH <= 3 ~ 1,
           MONTH <= 6 ~ 2,
           MONTH <= 9 ~ 3,
           TRUE ~ 4)) %>%
  mutate(YQ = paste0(YEAR, "Q", QUARTER)) %>%
  rename(weekly_earnings = LES1252881600Q) #180 obs

# Calculate percentage change quarter on quarter and select relevant columns
weekly_earnings <- weekly_earnings %>%
  mutate(earnings_change = (weekly_earnings/lag(weekly_earnings) - 1) * 100) %>%
  select(YQ, YEAR, weekly_earnings, earnings_change)

# Final obs = 180 obs
save(weekly_earnings, file = "RData/weekly_earnings.RData")

## 8. Load unemployment data
# UNRATE is in %
load("RData/unemployment.RData")

# Extract Year, Month, and Quarter from dataframe
unemployment <- unemployment %>%
  mutate(YEAR = year(DATE),
         MONTH = month(DATE),
         QUARTER = case_when(
           MONTH <= 3 ~ 1,
           MONTH <= 6 ~ 2,
           MONTH <= 9 ~ 3,
           TRUE ~ 4)) %>%
  mutate(YQ = paste0(YEAR, "Q", QUARTER)) #913 obs

# Aggregate average unemployment rate by quarter
unemployment <- unemployment %>%
  group_by(YQ) %>%
  mutate(unemployment_rate = mean(UNRATE)) %>%
  dplyr::slice(1) %>%
  ungroup() #305 obs

# Calculate percentage change quarter on quarter and select relevant columns
unemployment <- unemployment %>%
  mutate(unemployment_rate_change = round(unemployment_rate - lag(unemployment_rate))) %>%
  select(YQ, YEAR, unemployment_rate, unemployment_rate_change)

save(unemployment, file = "RData/unemployment.RData")

## 9. Load household consumption data
load("RData/household_consumption.RData")

# Clean up format
household_consumption <- dplyr::slice(household_consumption,3:n())
colnames(household_consumption) <- unlist(household_consumption[1, ])
household_consumption <- household_consumption[-1, ]
household_consumption <- household_consumption %>% filter(`Country Code` == "USA" )
household_consumption <- household_consumption %>% rename(CountryName = `Country Name`, CountryCo
household_consumption <- pivot_longer(household_consumption, cols = -c(CountryName,CountryCode),
                          names_to = "YEAR", values_to = "household_consumption")
household_consumption <- dplyr::slice(household_consumption, 3:n())
household_consumption <- household_consumption %>%
  mutate(household_consumption = as.numeric(household_consumption),
         YEAR = as.numeric(YEAR))

# Calculate percentage change year on year
household_consumption <- household_consumption %>%
  mutate(household_consumption_change = (household_consumption/lag(household_consumption) - 1) *
  select(-c(CountryName,CountryCode))

save(household_consumption, file = "RData/household_consumption.RData")

## 10. Load liquidity data
load("RData/liquidity.RData") #780 obs

# Extract Year, Month, and Quarter from data
liquidity <- liquidity %>%
  mutate(YEAR = year(DATE),
         MONTH = month(DATE),
         QUARTER = case_when(
           MONTH <= 3 ~ 1,
           MONTH <= 6 ~ 2,
           MONTH <= 9 ~ 3,
           TRUE ~ 4)) %>%
  mutate(YQ = paste0(YEAR, "Q", QUARTER)) %>%
  rename(liquidity = "M2SL")
```

```r
# Aggregate average of liquidity for each quarter and calculate percentage change
liquidity <- liquidity %>%
  group_by(YQ) %>%
  mutate(avg_liquidity = mean(liquidity)) %>%
  dplyr::slice(1) %>%
  ungroup() %>%
  mutate(avg_liquidity_change = (avg_liquidity/lag(avg_liquidity) - 1) * 100) #260 obs

# Select relevant columns
liquidity <- liquidity %>%
  select(YQ, YEAR, avg_liquidity, avg_liquidity_change)

# Final obs = 260 obs
save(liquidity, file = "RData/liquidity.RData")


## 11. Load gross saving data
load("RData/gross_saving.RData") #77 obs

# Select relevant columns
gross_saving <- gross_saving %>%
  select(Year, `Gross saving`, `Gross domestic investment, capital account transactions, and net
  rename(Gross_Saving = "Gross saving",
         Gross_Investment = "Gross domestic investment, capital account transactions, and net le
  rename(YEAR = "Year") %>%
  mutate(YEAR = as.numeric(YEAR))

# Because the year ends at 2015, we need to assume the next 5 years of data

# Calculating using CAGR
gs_changes <- diff(gross_saving$Gross_Saving)
Gross_Saving_CAGR <- mean((gs_changes / gross_saving$Gross_Saving[1:length(gs_changes)]) * 100, r

gi_changes <- diff(gross_saving$Gross_Investment)
Gross_Investment_CAGR <- mean((gi_changes / gross_saving$Gross_Investment[1:length(gi_changes)])

# Add new rows for years 2007 to 2020
for (i in 1:15) {
  # Calculate the new year
  new_year <- max(gross_saving$YEAR) + 1

  # Calculate non-year columns based on each previous row
  Gross_Saving <- gross_saving$Gross_Saving[nrow(gross_saving)] * (Gross_Saving_CAGR/100 + 1)
  Gross_Investment <- gross_saving$Gross_Investment[nrow(gross_saving)] * (Gross_Investment_CAGR/

  # Create a new row
  new_row <- data.frame(YEAR = new_year, Gross_Saving = Gross_Saving, Gross_Investment = Gross_In

  # Add the new row to the dataframe
  gross_saving <- rbind(gross_saving, new_row)
}

# Calculate gross saving and gross investment change year on year
gross_saving <- gross_saving %>%
  mutate(across(
    c(Gross_Saving, Gross_Investment),
    ~(./lag(.) - 1) * 100,
    .names = "{.col}_change"))

save(gross_saving, file = "RData/gross_saving.RData")

## 12. Load import data
load("RData/import.RData") #8208 obs

# Filter for grand total rows then select relevant columns then pivot the dataset into correct fo
import <- import %>%
  filter(str_detect(`EU Description`, "Grand")) %>%
  select(Year, Qtr1, Qtr2, Qtr3, Qtr4) %>%
  pivot_longer(cols = -Year, names_to = "Quarter", values_to = "import") #136 obs

# Properly format Year and Quarter column
import <- import %>%
  mutate(Quarter = paste0("Q", substr(Quarter,4,4))) %>%
  mutate(YQ = paste0(Year, Quarter))

# Select relevant columns
import <- import %>%
  select(YQ, Year, import) %>%
  rename(YEAR = "Year")

# Calculate import changes in percentage
import <- import %>%
  mutate(import_change = (import/lag(import) - 1) * 100)
```

```r
# Final obs = 136 obs
save(import, file = "RData/import.RData")

## 13. Load export data
load("RData/export.RData") #7978 obs

# Filter for grand total rows then select relevant columns then pivot the dataset into correct fo
export <- export %>%
  filter(str_detect(`EU Description`, "Grand")) %>%
  select(Year, Qtr1, Qtr2, Qtr3, Qtr4) %>%
  pivot_longer(cols = -Year, names_to = "Quarter", values_to = "export") #136 obs

# Properly format Year and Quarter column
export <- export %>%
  mutate(Quarter = paste0("Q", substr(Quarter,4,4))) %>%
  mutate(YQ = paste0(Year, Quarter))

# Select relevant columns
export <- export %>%
  select(YQ, Year, export) %>%
  rename(YEAR = "Year")

# Calculate export changes in percentage
export <- export %>%
  mutate(export_change = (export/lag(export) - 1) * 100)

# Final obs = 136 obs
save(export, file = "RData/export.RData")
```

Part 4: Prepping and cleaning accounting information data

```r
# set eval=TRUE to run this chunk, it will take more than 3 minutes

## 1. Load funda_merged data for cleaning

load("RData/funda_merged_q.RData")
nrow(funda_merged_q) #1,217,457 records
sum(duplicated(as.data.table(funda_merged_q))) #0 duplicated rows if we try to find duplicate on
funda_merged_q <- funda_merged_q %>% mutate(YQ = paste0(fyearq, "Q", fqtr)) #create YQ column fo
funda_merged_q <- funda_merged_q %>% mutate(gvkey_YQ = paste0(GVKEY, YQ))
sum(duplicated(as.data.table(funda_merged_q$gvkey_YQ))) # 11,022 duplicated rows
funda_merged_q <- funda_merged_q %>% filter(duplicated(gvkey_YQ) == FALSE)
sum(duplicated(as.data.table(funda_merged_q$gvkey_YQ))) # 0 duplicate based on Gvkey-Year-Quarte
#797,879 records


## 2. Create financial ratios/variables for regression analysis based on formulas from Financial

funda_merged_q <- funda_merged_q %>%
  mutate(net_op_asset = ((lag(ppentq+actq-lctq)+(ppentq+actq-lctq))/2)) %>%
  mutate(provision_change = 100 * finitize( (pllq/lag(pllq) - 1)),
         writeoff_change = 100 * finitize( (wdaq/lag(wdaq) - 1)),
         p_dividend_change = 100 * finitize( (dvpq/lag(dvpq) - 1)),
         common_share_change = 100 * finitize( (cshiq/lag(cshiq) - 1)),
         shareholder_equity_change = 100 * finitize( (seqq/lag(seqq) - 1)),
         dpr = 100 * finitize(dvpq / ibadjq),
         capital_ratio = 100 * finitize(dlttq/(dlttq + sum(ceqq, pstkq))),
         debt_invcap = 100 * finitize(dlttq/icaptq),
         at_turn = 100 * finitize(saleq/((atq+lag(atq))/2)),
         inv_turn = 100 * finitize(cogsq/invtq),
         asset_change = 100 * finitize( (atq/lag(atq) - 1)),
         current_asset_change = 100 * finitize( (actq/lag(actq) - 1)),
         liabilities_change = 100 * finitize( (ltq/lag(ltq) - 1)),
         current_liabilities_change = 100 * finitize( (lctq/lag(lctq) - 1)),
         cash_ce_change = 100 * finitize( (cheq/lag(cheq) - 1)),
         profit_lct = 100 * finitize(coalesce(oibdpq, saleq - xoprq)/lctq),
         cash_ratio = 100 * finitize(cheq/lctq),
         quick_ratio = 100 * finitize(coalesce(actq - invtq, cheq + rectq)/lctq),
         cf_to_asset = 100 * finitize((ibcy+dpq)/atq),
         share_growth = 100 * finitize( (prccq/lag(prccq) - 1)),
         book_to_market = 100 * finitize(sum(seqq, txditcq, -pstkq)/(prccq*cshoq)),
         price_sales = 100 * finitize(prccq/saleq),
         net_profit_change = 100 * finitize( (niq/lag(niq) - 1)),
         depr_change = 100 * finitize( (dpq/lag(dpq) - 1)),
         net_op_asset_change = 100 * finitize( (net_op_asset/lag(net_op_asset) - 1)),
         revenue_change = 100 * finitize( (revtq/lag(revtq) - 1)),
         roa = 100 * finitize(coalesce(oibdpq, saleq - xoprq, revtq - xoprq)/((atq + lag(atq)) /
         net_profit_margin = 100 * finitize(ibq/saleq),
         opm_bd = 100 * finitize(coalesce(oibdpq, saleq-xoprq, revtq-xoprq)/saleq)
         )

## 3. Check proportion of missing values for each variables
```

```r
acct_variables <- c(
  "provision_change", "writeoff_change", "p_dividend_change", "common_share_change",
  "shareholder_equity_change", "dpr", "capital_ratio", "debt_invcap", "at_turn",
  "inv_turn", "asset_change", "current_asset_change", "liabilities_change",
  "current_liabilities_change", "cash_ce_change", "profit_lct", "cash_ratio",
  "quick_ratio", "cf_to_asset", "share_growth", "book_to_market", "price_sales",
  "net_profit_change", "revenue_change", "roa", "net_profit_margin", "opm_bd", "depr_change", "ne
)

# Calculate the percentage of null values in each column
calculate_missing_percentage(funda_merged_q, acct_variables)

# Eliminate variables with more than 50% missing values
null_percentage <- colMeans(is.na(funda_merged_q)) * 100
null_df <- data.frame(Column = names(null_percentage), Null_Percentage = null_percentage)
null_df_subset <- null_df[null_df$Column %in% acct_variables, ]
cols_to_delete <- null_df_subset$Column[null_df_subset$Null_Percentage > 50]
funda_df <- funda_merged_q[, !names(funda_merged_q) %in% cols_to_delete]

## 4. Winsorize at 2% level for all the accounting variables
# Redefine relevant acct. variables
acct_variables <- c("share_growth", "net_profit_margin", "price_sales", "dpr", "net_profit_change
                    "opm_bd", "shareholder_equity_change", "asset_change", "at_turn", "liabilitie
                    "revenue_change", "cash_ce_change", "roa", "inv_turn", "cash_ratio", "quick_r
                    "profit_lct", "current_liabilities_change", "current_asset_change", "cf_to_as
                    "depr_change", "net_op_asset_change")

funda_df <- funda_df %>%
  group_by(YQ) %>%
  mutate(across(
    .cols = acct_variables,
    ~winsor(., trim = 0.02)
  )) %>%
  ungroup()

# Select relevant columns
funda_df <- funda_df %>%
  select(GVKEY, sic, spcseccd, state, gvkey_YQ, YQ, datadate, fyearq, fqtr, cusip, conm, all_of(a
  rename(YEAR = "fyearq")

# Fill in with the average of the Company-Year group
funda_df <- funda_df %>%
  group_by(GVKEY, YEAR) %>%
  mutate(across(
    .cols = all_of(acct_variables),
    ~replace_na(., mean(., na.rm = TRUE))
  )) %>%
  ungroup()

#calculate_missing_percentage(funda_df, acct_variables) # Data is still missing

# Fill in with the average of the Year-Quarter group
funda_df <- funda_df %>%
  group_by(YQ) %>%
  mutate(across(
    .cols = acct_variables,
    ~replace_na(., mean(., na.rm = TRUE))
  )) %>%
  ungroup()

#calculate_missing_percentage(funda_df, acct_variables) #Decreased missing data but still missing

# Fill in with the average of the Year group
funda_df <- funda_df %>%
  group_by(YEAR) %>%
  mutate(across(
    .cols = acct_variables,
    ~replace_na(., mean(., na.rm = TRUE))
  )) %>%
  ungroup()

#calculate_missing_percentage(funda_df, acct_variables)

# Fill in with the average of the Company group
funda_df <- funda_df %>%
  group_by(GVKEY) %>%
  mutate(across(
    .cols = acct_variables,
    ~replace_na(., mean(., na.rm = TRUE))
  )) %>%
  ungroup()

#calculate_missing_percentage(funda_df, acct_variables)
```

```r
# Fill in with the average of the Standard Industry Classification group
funda_df <- funda_df %>%
  group_by(sic) %>%
  mutate(across(
    .cols = acct_variables,
    ~replace_na(., mean(., na.rm = TRUE))
  )) %>%
  ungroup()

calculate_missing_percentage(funda_df, acct_variables) #0% missing data

##5. Aggregate variables by quarter

funda_clean <- funda_df %>%
  select(YQ, YEAR,fqtr, all_of(acct_variables)) %>%
  group_by(YQ) %>%
  summarize(across(acct_variables, ~mean(., na.rm = TRUE)),
            num_companies = n())
# Final obs = 254 obs

save(funda_clean, file = "RData/funda_clean.RData")
```

Part 5: Joining features data to training and testing data

```r
## 1. Load train data to be left joined with independent variables
load("RData/train_df.RData")

load("RData/gdp_forecast_mean.RData")
load("RData/net_income.RData")
load("RData/cpi.RData")
load("RData/energy_consumption.RData")
load("RData/fed_rates.RData")
load("RData/weekly_earnings.RData")
load("RData/unemployment.RData")
load("RData/household_consumption.RData")
load("RData/liquidity.RData")
load("RData/gross_saving.RData")
load("RData/import.RData")
load("RData/export.RData")
load("RData/funda_clean.RData")


## 2. Join all the relevant data together
final_df <- train_df %>%
  left_join(gdp_forecast_mean) %>%
  left_join(net_income) %>%
  left_join(cpi) %>%
  left_join(energy_consumption) %>%
  left_join(fed_rates) %>%
  left_join(weekly_earnings) %>%
  left_join(unemployment) %>%
  left_join(household_consumption) %>%
  left_join(liquidity) %>%
  left_join(gross_saving) %>%
  left_join(import) %>%
  left_join(export) %>%
  left_join(funda_clean)
# Final obs = 104 obs

## 3. Rename, remove irrelevant columns, and perform Z-score normalization on num of companies
final_df <- final_df %>%
  rename(GDP_Growth = "NGDP",
         Forecast_GDP = "NGDP2") %>%
  mutate(num_companies_standard = z_score_normalization(num_companies))

## 4. Repeat the same process for same data
load("RData/test_df.RData")

## 5. Join all the relevant data together
final_test_df <- test_df %>%
  left_join(gdp_forecast_mean) %>%
  left_join(net_income) %>%
  left_join(cpi) %>%
  left_join(energy_consumption) %>%
  left_join(fed_rates) %>%
  left_join(weekly_earnings) %>%
  left_join(unemployment) %>%
  left_join(household_consumption) %>%
  left_join(liquidity) %>%
  left_join(gross_saving) %>%
  left_join(import) %>%
  left_join(export) %>%
  left_join(funda_clean)
```

```
# Final obs = 104 obs

## 6. Rename, remove irrelevant columns, and standardize number of companies
final_test_df <- final_test_df %>%
  #select(-c(NGDP)) %>%
  rename(Forecast_GDP = "NGDP2",
         GDP_Growth = "NGDP") %>%
  mutate(num_companies_standard = z_score_normalization(num_companies))

## 7. Save training and test data as RData

save(final_df, file = "RData/final_df.RData")
save(final_test_df, file = "RData/final_test_df.RData")
```

Variable Selection

The full list of dependent and independent variables will be provided in Appendix 2 - Table of Variables[3].

In this report, the dependent variable will be the quarterly US GDP Growth obtained from the Bureau of Economic Analysis' final estimate, available at the BEA website.

For independent variables, there are 2 types: macroeconomic and accounting. The macroeconomic are retrieved from various sources, listed in Appendix 2. As for the accounting independent variables, they are sourced from Wharton Research Data Services, specifically from the **CRSP/Compustat Merged Database - Fundamentals Quarterly**, available in the CRSP database which provides information on both fundamental accounting information and stock price information of US firms on a quarterly basis. The accounting variables used were manually calculated from the information available from this dataset based on **Financial Ratios SAS Code**, available on WRDS.

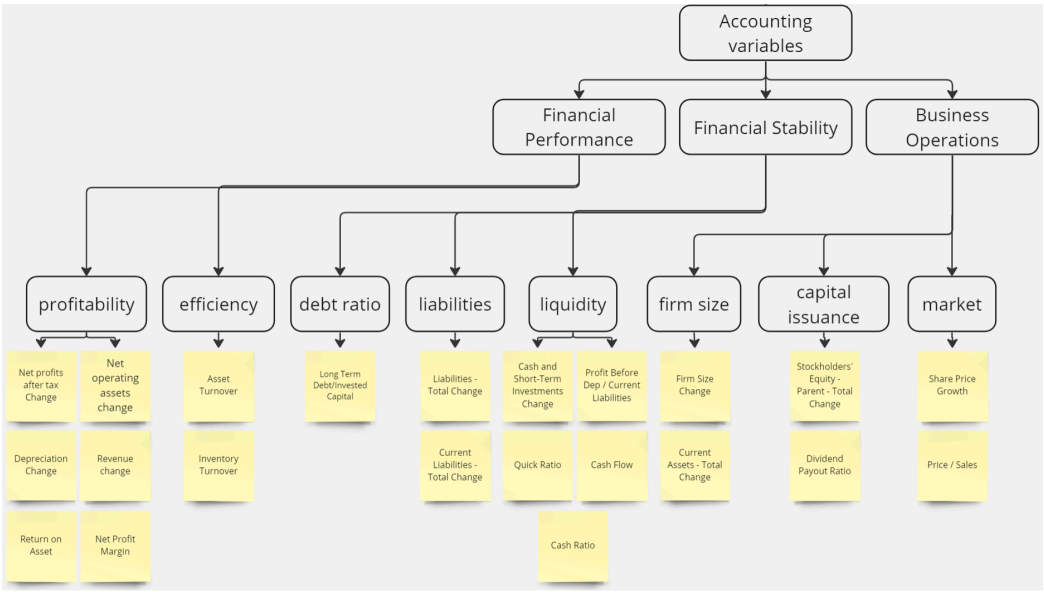
III. Data Analysis

A. Exploratory Data Analysis

Part 1: Data Overview

All of the data used in the further exploration and regression are numeric.

After joining the data, we've come down to a total of 42 variables, which include 23 "Accounting" variables and 19 "Macroeconomic" variables.



Within the "Accounting" type, we further categorized the variables into the following classes:

1.Financial Performance Metrics:

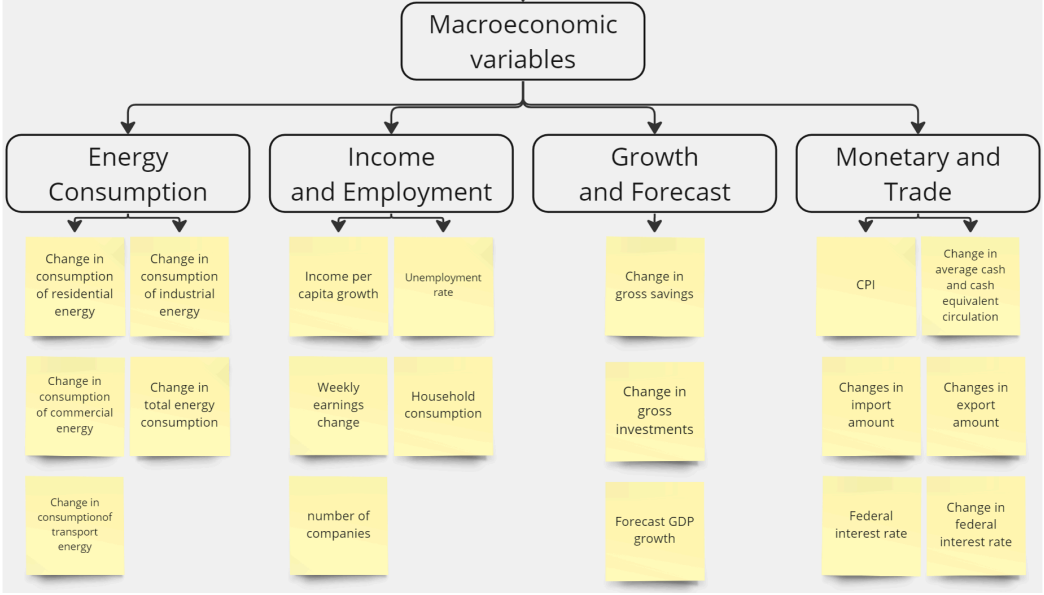
•Includes variables which gauge a company or economy's capacity to generate profits.

2.Financial Stability Indicators:

•Encompasses variables which offer insights into a firm's financial robustness, serving as metrics for societal risk assessment.

3.Business Operations:

•Consists of variables at firm level which account for industry expansion, while providing perspectives on the company's valuation and investor sentiment.



For the macroeconomic variables, we mainly choose the following variables:

1.Economic Growth and Forecasts: Encompasses variables that directly reflect the overall trend and expectations of economic activity.

2.Monetary and International Trade: assess the stability level and global influence of a nation's economy.

3.Energy Consumption: Reflect the development strategy of the national economic.

4.Employment and Income: captures the condition of the labor market and the economic well-being of the populace.

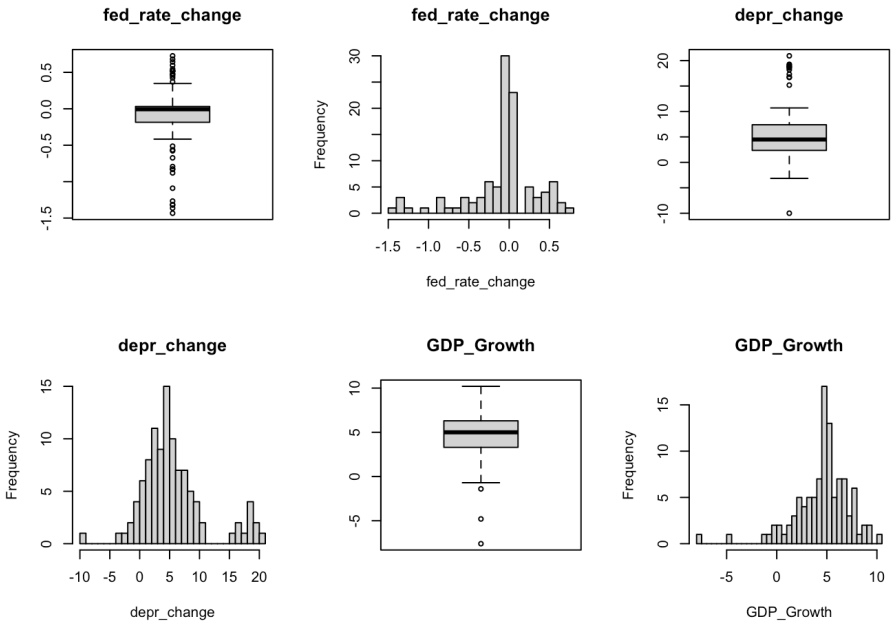
```
# Define Macroeconomic and Accounting variables

macro_vars <- c("forecast_growth", "income_per_capita_growth", "CPI",
                "residential_energy_change", "commercial_energy_change",
                "industrial_energy_change", "transport_energy_change",
                "total_energy_change", "fed_rate", "fed_rate_change",
                "earnings_change", "unemployment_rate_change",
                "household_consumption_change", "avg_liquidity_change",
                "Gross_Saving_change", "Gross_Investment_change",
                "import_change", "export_change", "num_companies_standard")

acct_vars <- c("shareholder_equity_change", "dpr", "debt_invcap", "at_turn", "inv_turn",
               "asset_change", "current_asset_change", "liabilities_change",
               "current_liabilities_change", "cash_ce_change", "profit_lct", "cash_ratio",
               "quick_ratio", "cf_to_asset", "share_growth", "price_sales", "net_profit_change",
               "net_op_asset_change", "depr_change", "revenue_change", "roa", "net_profit_margin'
               "opm_bd")

comb_vars <- c(macro_vars, acct_vars)
```

Part 2: Boxplot and histogram

Here, we utilize boxplots and histograms to help us identify outliers. Among the explanatory variables of accounting and macroeconomic type, we selected a typical outlier from each of the two categories to show the outlier. However, due to space constraints, not all variables will be shown and as such, only boxplots representing two variables (One accounting and one macroeconomic variable) which have the highest deviations will be shown, alongside boxplots of the dependent variable - GDP Growth.

```
# Define a function to calculate the number of outliers
outliers_count <- function(variable) {
  # Calculate quartiles and interquartile range
  Q1 <- quantile(variable, 0.25, na.rm = TRUE)
  Q3 <- quantile(variable, 0.75, na.rm = TRUE)
  IQR <- Q3 - Q1

  # Define lower and upper bounds for outliers
  lower_bound <- Q1 - 1.5 * IQR
  upper_bound <- Q3 + 1.5 * IQR

  # Calculate and return the number of outliers
  sum(variable < lower_bound | variable > upper_bound, na.rm = TRUE)
}

# Calculate the number of outliers for each category
macro_outliers_count <- sapply(final_df[macro_vars], outliers_count)
acct_outliers_count <- sapply(final_df[acct_vars], outliers_count)

# Identify the variable with the most outliers in each category
```

```r
macro_var_with_most_outliers <- names(which.max(macro_outliers_count))
acct_var_with_most_outliers <- names(which.max(acct_outliers_count))

# Define the variables to be plotted
vars_to_plot <- c(macro_var_with_most_outliers, acct_var_with_most_outliers, "GDP_Growth")

# Create boxplots and histograms
par(mfrow=c(2, 3))  # Arrange the graphics as 2 rows by 3 columns
for(var in vars_to_plot) {
  # Boxplot
  boxplot(final_df[[var]], main=var)

  # Histogram
  hist(final_df[[var]], main=var, xlab=var, breaks=30)
}
```



The median value of GDP_Growth is 4.6%. Most of our observing data fall in the realm of normal with only three outliers that are below Q1 - 1.5*IQR.

Tracing back to the anomalies, it is the GDP growth of Q4 2008(-7.6%), Q1 2009(-4.8%), and Q2 2009(-1.4%), symptomatic of the economic downturn during the Great Recession.

Referring back to the context, during the last quarter of 2008, the crisis reached its peak, with financial institutions collapsing or being bailed out, credit markets seizing up, global stock markets plummeting, and consumer confidence hitting record lows. The impact was so severe that it led to a drastic contraction in the GDP, which is reflected in the -7.6% growth rate in Q4 of 2008.

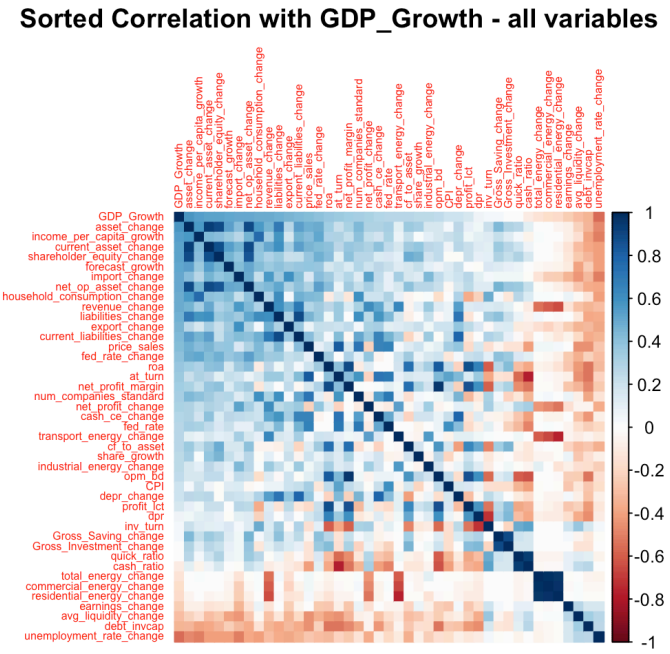
The negative trend continued into 2009, albeit with a slowing pace of contraction, indicated by the -4.8% and -1.4% GDP growth in Q1 and Q2, respectively. The economy was still reeling from the effects of the financial crisis, including high unemployment rates, sluggish firm performance and tightened credit conditions as we can see from our data.

However, with government interventions, including fiscal stimulus packages and monetary policy actions began to take effect, the negative growth rates started to ameliorate as the year 2009 progressed.

Furthermore, it can be observed that both Depreciation Change (depr_change) and Change in Federal Interest Rate (fed_rate_change) exhibit the highest number of outliers, yet the majority of observations still concentrate around the center, indicating relative stability. The outliers are not prevalent and are often responses to unexpected situations. For instance, the lowest Change in Federal Interest Rate recorded in 2008 was -1.4333333, which was a policy change made in response to the financial crisis.

```r
 # Include GDP_Growth and all other variables
  vars_for_correlation2 <- c("GDP_Growth", comb_vars)

  # Calculate the correlation matrix
  cor_matrix2 <- cor(final_df[, vars_for_correlation2], use = "complete.obs")

# Correlation coefficients with GDP_Growth
  gdp_correlation <- cor_matrix2["GDP_Growth", -1]

  # Separate positive and negative correlations and sort them by absolute value
  positive_corrs <- sort(gdp_correlation[gdp_correlation > 0], decreasing = TRUE)
  negative_corrs <- sort(gdp_correlation[gdp_correlation < 0], decreasing = TRUE)

  # Merge variable names sorted by positive and negative correlation coefficients
  sorted_vars <- c("GDP_Growth", names(positive_corrs), names(negative_corrs))
```

```r
# Retrieve the sorted correlation matrix
sorted_cor_matrix2 <- cor_matrix2[sorted_vars, sorted_vars]

# Create the heatmap on a new graphic device
 corrplot(sorted_cor_matrix2, method = "color", order = "original",
         title = paste("Sorted Correlation with GDP_Growth - all variables"),
         tl.cex = 0.5, # Adjust the font size of variable names
         mar = c(0,0,2,0))
```




```r
# Figure out which variables are at least 90% correlated
high_corr_pairs <- which(abs(sorted_cor_matrix2) > 0.9 & abs(sorted_cor_matrix2) < 1, arr.ind = T

# Extract variable names from column and row indices
variable1 <- rownames(sorted_cor_matrix2)[high_corr_pairs[, 1]]
variable2 <- colnames(sorted_cor_matrix2)[high_corr_pairs[, 2]]

# Create a data frame with Variable1 and Variable2 columns
high_corr_df <- data.frame(Variable1 = variable1, Variable2 = variable2)

# Print the resulting data frame
print(high_corr_df)
```

```
                 Variable1                Variable2
1       current_asset_change             asset_change
2              asset_change      current_asset_change
3                   opm_bd         net_profit_margin
4          net_profit_margin                   opm_bd
5                cash_ratio              quick_ratio
6               quick_ratio               cash_ratio
7   commercial_energy_change      total_energy_change
8  residential_energy_change      total_energy_change
9       total_energy_change commercial_energy_change
10 residential_energy_change commercial_energy_change
11      total_energy_change residential_energy_change
12 commercial_energy_change residential_energy_change
```

```r
# Decided to remove one of the pairs of variable with >90% correlation
var_to_remove <- c("commercial_energy_change", "residential_energy_change", "current_asset_change

comb_vars <- setdiff(comb_vars, var_to_remove)
```

We begin by drawing the heatmap illustrating the correlation between all variables and GDP growth. Through this analysis, we identified top 5 positively correlated indicators as follows: Income per capita growth, forecast growth, import change, household consumption change, and export change. Primarily, we can see all top 5 most positively relevant indicators fall in the macro realm.

The top 5 most positively relevant accounting variables are: asset turnover, return on assets, net profit margin, price/sales, and operating margin before depreciation.

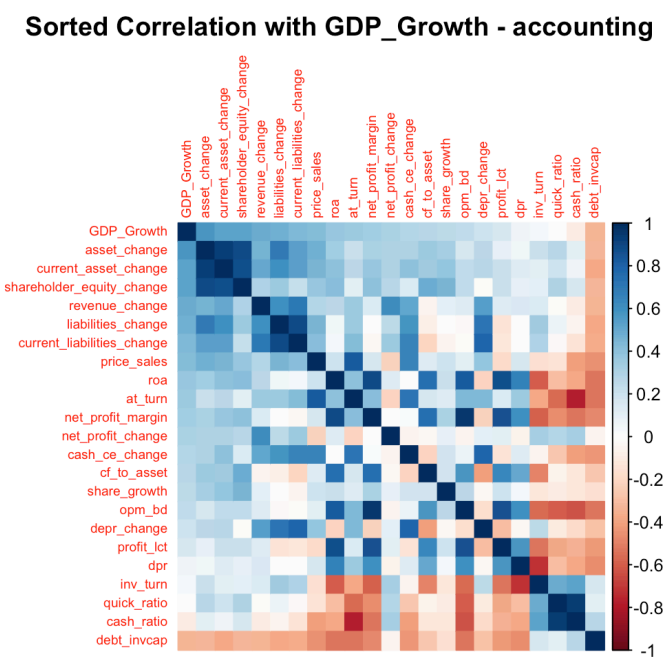
Additionally, the top 3 variables negatively correlated with GDP growth are: unemployment rate change, shareholder equity change, and debt ratio.

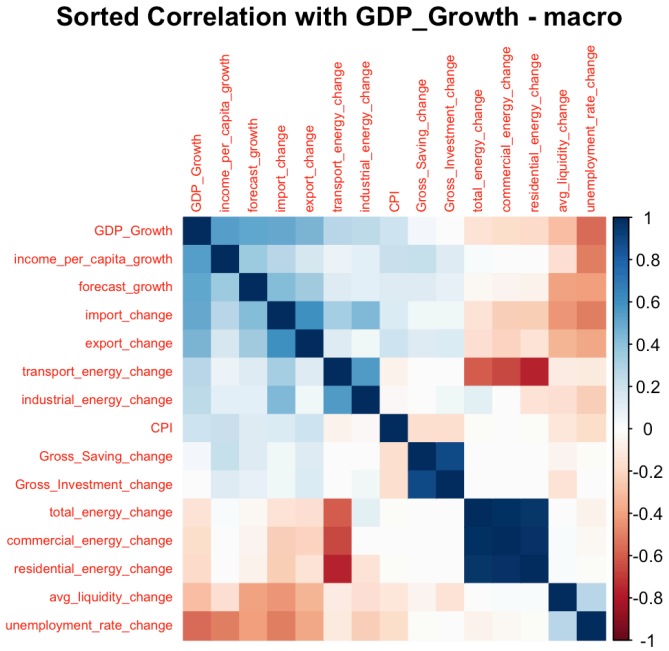
Further examination is required to ascertain the precise role these variables play in forecasting GDP.

Part 4: Sorted Correlation with GDP_Growth -accounting and GDP_Growth -macro

Then we drew the heatmap illustrating the correlation between all variables and GDP growth, segmenting by the variable types (Accounting and Macroeconomic). Through this analysis, we identified top 5 positively correlated indicators as follows: Firm Size Change, Income per capita growth, Current Assets - Total Change, shareholder equity change, forecast growth.

```
# Correlation analysis and heatmap
profitability_vars <- c("net_profit_change", "depr_change", "revenue_change", "roa", "net_profit_
efficiency_vars <- c("at_turn", "inv_turn")
debt_ratio_vars <- c("dpr", "debt_invcap")
liabilities_vars <- c("liabilities_change", "current_liabilities_change")
liquidity_vars <- c("cash_ce_change", "profit_lct", "cash_ratio", "quick_ratio", "cf_to_asset")
firm_size_vars <- c("asset_change", "current_asset_change")
capital_issuance_vars <- c("shareholder_equity_change", "share_growth")
market_vars <- c("price_sales")

# Combine categories
category_vars_no_show <- list(
  Operational_Performance = c(profitability_vars, efficiency_vars),
  Financial_Health = c(debt_ratio_vars, liabilities_vars, liquidity_vars),
  Market_Positioning = c(firm_size_vars, capital_issuance_vars, market_vars),
  Energy_Consumption = c("residential_energy_change", "commercial_energy_change",
                         "industrial_energy_change", "transport_energy_change",
                         "total_energy_change"),
  Income_Employment = c("income_per_capita_growth", "unemployment_rate_change"),
  Growth_Forecast = c("forecast_growth", "Gross_Saving_change", "Gross_Investment_change"),
  Monetary_Trade = c("CPI", "avg_liquidity_change", "import_change", "export_change")
)

# Combine categories
Operational_Performance <- c(profitability_vars, efficiency_vars)
Financial_Health <- c(debt_ratio_vars, liabilities_vars, liquidity_vars)
Market_Positioning <- c(firm_size_vars, capital_issuance_vars, market_vars)
Energy_Consumption <- c("residential_energy_change", "commercial_energy_change",
                        "industrial_energy_change", "transport_energy_change",
                        "total_energy_change")
Income_Employment <- c("income_per_capita_growth", "unemployment_rate_change")
Growth_Forecast <- c("forecast_growth", "Gross_Saving_change", "Gross_Investment_change")
Monetary_Trade <- c("CPI", "avg_liquidity_change", "import_change", "export_change")


category_vars_show1 <- list(
  accounting=c(Operational_Performance, Financial_Health, Market_Positioning),
  macro=c(Energy_Consumption, Income_Employment, Growth_Forecast, Monetary_Trade)
)


# Loop through each category and create heatmaps
for (category in names(category_vars_show1)) {
  # Include GDP_Growth and all other variables
  vars_for_correlation <- c("GDP_Growth", category_vars_show1[[category]])

  # Calculate the correlation matrix
  cor_matrix <- cor(final_df[, vars_for_correlation], use = "complete.obs")

  # Correlation coefficients with GDP_Growth
  gdp_correlation <- cor_matrix["GDP_Growth", -1]

  # Separate positive and negative correlations and sort them by absolute value
  positive_corrs <- sort(gdp_correlation[gdp_correlation > 0], decreasing = TRUE)
  negative_corrs <- sort(gdp_correlation[gdp_correlation < 0], decreasing = TRUE)

  # Merge variable names sorted by positive and negative correlation coefficients
  sorted_vars <- c("GDP_Growth", names(positive_corrs), names(negative_corrs))

  # Retrieve the sorted correlation matrix
  sorted_cor_matrix <- cor_matrix[sorted_vars, sorted_vars]

  # Create the heatmap on a new graphic device
   corrplot(sorted_cor_matrix, method = "color", order = "original",
           title = paste("Sorted Correlation with GDP_Growth -", category),
           tl.cex = 0.6, # Adjust the font size of variable names
           mar = c(0,0,2,0))
}
```





The top 5 most positively relevant accounting variables are: Firm Size Change ,Current Assets - Total Change, shareholder equity change, Revenue change and Liabilities - Total Change . The top 5 most positively relevant macroeconomic variables are: Income per capita growth , forecast growth, Changes in import amount, Changes in export amount and Change in consumption of transport energy.

Additionally, the top 3 variables negatively correlated with GDP growth are: unemployment rate change, debt ratio, and Change in average cash and cash equivalent circulation.

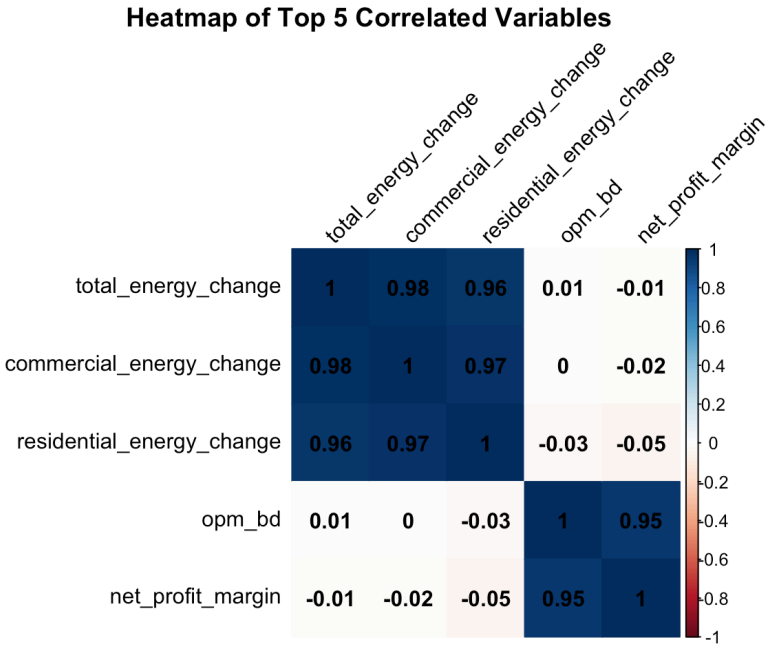
Further examination is required to ascertain the precise role these variables play in forecasting GDP.

Part 5: Heatmap of Top 5 Correlated Variables

In the meantime, we constructed another heat map to visualize just the top 5 autocorrelation of the explanatory variables.

```
category_vars_show3 <- list(
  top_5=c(Operational_Performance, Financial_Health,
          Market_Positioning,Energy_Consumption,
          Income_Employment, Growth_Forecast, Monetary_Trade))

# Calculate the correlation matrix
cor_matrix <- cor(final_df[, category_vars_show3$top_5], use = "complete.obs")

# Convert the correlation matrix to a long format and remove self-correlations
cor_long <- subset(as.data.frame(as.table(cor_matrix)), Var1 != Var2)

# Sort the correlations by absolute value
cor_long_sorted <- cor_long[order(-abs(cor_long$Freq)), ]

# Initialize a vector to store the selected variable names
selected_vars <- c()

# Select five unique variables with the highest correlations from the sorted data
for (i in 1:nrow(cor_long_sorted)) {
```

```
  vars <- c(cor_long_sorted$Var1[i], cor_long_sorted$Var2[i])

  # Add only those variables that have not yet been selected
  selected_vars <- unique(c(selected_vars, vars))

  # Exit the loop once we have enough variables
  if (length(selected_vars) >= 5) {
    break
  }
}

# Select the top five variables
top_five_vars <- selected_vars[1:5]

# Construct a new correlation matrix with these five variables
top_five_cor_matrix <- cor_matrix[top_five_vars, top_five_vars]

# Plot the heatmap for these five variables
corrplot(top_five_cor_matrix, method = "color",
         title = "Heatmap of Top 5 Correlated Variables",
         addCoef.col = "black", # Add number colors
         tl.col = "black", tl.srt = 45, # Set label color and angle
         mar = c(0,0,2,0)) # Adjust the margin to ensure the title is visible
```

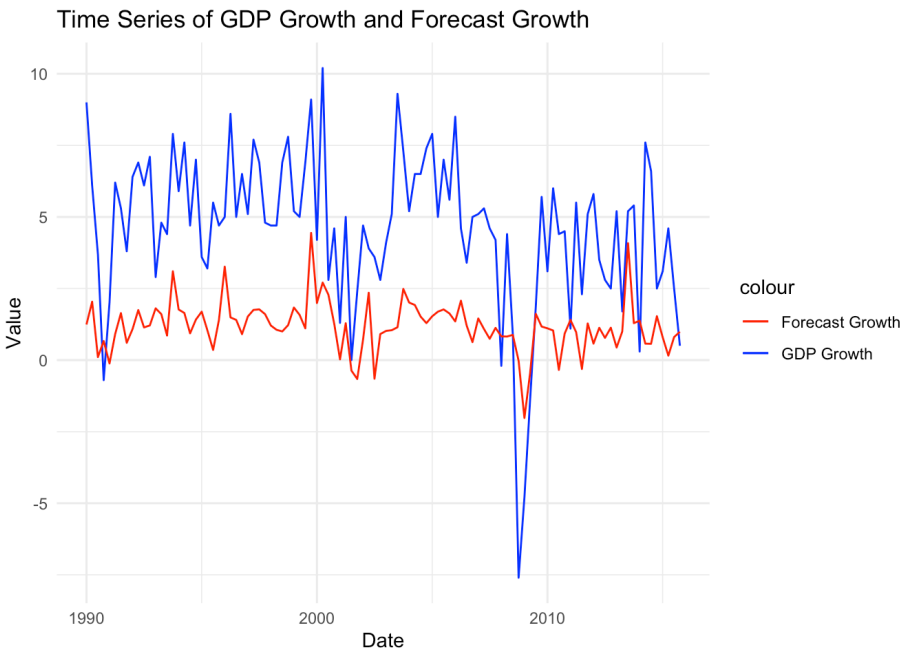


A pronounced concentration of autocorrelation is within our selected range of energy consumption variables. There exists a high degree of correlation between commercial_energy_change and total_energy_change, with a coefficient of 0.98. Commercial_energy_change and residential_energy_change, as well as residential_energy_change and total_energy_change, are correlated with coefficients of 0.97 and 0.96, respectively.

Besides, opm_bd exhibits a correlation of 0.95 with net_profit_margin. As mentioned during the previous section, to mitigate multicollinearity in our subsequent analysis, we chose to exclude one of the pair of highly correlated independent variables to improve the robustness.

Part 6: Time series plots

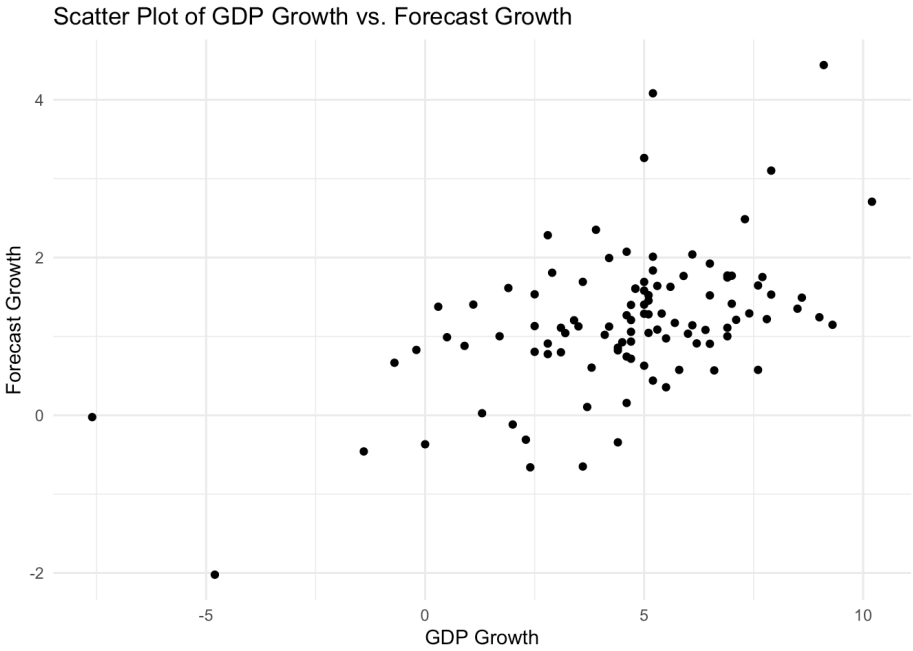
```
# Assume final_df is your data frame and it contains a column named YQ representing the year and
# Convert the YQ column to Date objects
final_df$Date <- as.Date(as.yearqtr(final_df$YQ, format = "%YQ%q"))
# Plotting GDP Growth and Forecast Growth over time using ggplot2
ggplot(final_df, aes(x = Date)) +
  geom_line(aes(y = GDP_Growth, color = "GDP Growth")) +  # Line plot for GDP Growth
  geom_line(aes(y = forecast_growth, color = "Forecast Growth")) +  # Line plot for Forecast Grov
  labs(title = "Time Series of GDP Growth and Forecast Growth", x = "Date", y = "Value") +  # Adc
  theme_minimal() +  # Minimal theme
  scale_color_manual(values = c("GDP Growth" = "blue", "Forecast Growth" = "red"))  # Manual colc
```



We do a comparative time series analysis of actual GDP growth and its forecasts over the period. The blue line represents the actual GDP growth, exhibiting the inherent volatility and cyclicality characteristic of the real economy, while the red line provides the predictive GDP growth.

Upon examination, we can several features. First, both the actual and forecasted GDP growth rates demonstrate cyclical patterns, representing the expansions and contractions of the overall economy. Second, the forecast generally follows the up-and-down trend of the actual GDP growth but is notably smoother and less pronounced with equal precision. However, the unanticipated economic shocks or crises are what models typically struggle to predict due to their outlier nature.Third, there appears to be lags in the forecasted GDP growth's response to the actual GDP growth's fluctuations, implying that the model displays the delay in reflecting rapid changes in economic indicators.

```
# Scatter plot of GDP Growth vs. Forecast Growth
ggplot(final_df, aes(x = GDP_Growth, y = forecast_growth)) +
  geom_point() +  # Scatter plot points
  labs(title = "Scatter Plot of GDP Growth vs. Forecast Growth", x = "GDP Growth", y = "Forecast
  theme_minimal()  # Minimal theme
```



The scatter plot we drew compares GDP growth with forecasted growth, revealing a positive but dispersed relationship between the two variables. Data points are clustered around the center, indicating that most forecasts are close to actual GDP growth when growth is near the mean. Outliers suggest less accurate forecasts for extreme growth rates.

B. Regression Analysis

```
# Define Macroeconomic, and Accounting formula

macro_eq <- as.formula("GDP_Growth ~ forecast_growth + income_per_capita_growth + CPI +
```

```
                       industrial_energy_change + transport_energy_change +
                       total_energy_change + fed_rate + fed_rate_change + earnings_change +
                       unemployment_rate_change + household_consumption_change +
                       avg_liquidity_change + Gross_Saving_change + Gross_Investment_change +
                       import_change + export_change + num_companies_standard + QUARTER")

acct_eq <- as.formula("GDP_Growth ~ shareholder_equity_change + dpr + debt_invcap + at_turn +
                      inv_turn + asset_change + liabilities_change +
                      current_liabilities_change + cash_ce_change + profit_lct +
                      quick_ratio + cf_to_asset + share_growth + price_sales + net_profit_change
                      net_op_asset_change + depr_change + revenue_change + roa + net_profit_margi
                      QUARTER")

# Both macroeconomic and accounting variables in formula
comb_eq <- as.formula("GDP_Growth ~ forecast_growth + income_per_capita_growth + CPI +
                       industrial_energy_change + transport_energy_change +
                       total_energy_change + fed_rate + fed_rate_change + earnings_change +
                       unemployment_rate_change + household_consumption_change +
                       avg_liquidity_change + Gross_Saving_change + Gross_Investment_change +
                       import_change + export_change + num_companies_standard + QUARTER +
                       shareholder_equity_change + dpr + debt_invcap + at_turn +
                       inv_turn + asset_change + liabilities_change +
                       current_liabilities_change + cash_ce_change + profit_lct +
                       quick_ratio + cf_to_asset + share_growth + price_sales + net_profit_change
                      net_op_asset_change + depr_change + revenue_change + roa + net_profit_margi

# init a list used to store rmse

rmse_ls = list()

# init a dataframe to store predictions
pred_test_list <- as.data.frame(final_test_df[,'YQ'])
names(pred_test_list) <- 'YQ'

# init a dataframe to store predictions
pred_train_list <- as.data.frame(final_df[,'YQ'])
pred_test_list <- as.data.frame(final_test_df[,'YQ'])
names(pred_train_list) <- 'YQ'
names(pred_test_list) <- 'YQ'
```

a. OLS

Firstly we use OLS as our prediction model to train.

Here **`caret` package is employed for cross validation.**

- **Define Train Control**:
  - `trainControl(method = "repeatedcv", number = 10, repeats = 3)`
    - This code sets up the train control parameters for cross-validation.
    - It specifies the repeated cross-validation method with 10 folds and 3 repetitions.
    - **`repeatedcv`** is a method of cross-validation that repeats the process multiple times to ensure robustness in estimating model performance.
- **Train the OLS Model using Macro and Comb Variables** separately
  - `ols_macro <- train(macro_eq, data = final_df, method = "lm", trControl = train.control)`
  - `ols_comb <- train(comb_eq, data = final_df, method = "lm", trControl = train.control)`
    - Here, the **`train()`** function from the **`caret`** package is used to train a linear regression model (**`lm`**) using only macroeconomic and combined data variables (**`macro_eq`**) separately

Macro Model

```
# Set seed for reproducibility
set.seed(2024)

train.control <- trainControl(method = "repeatedcv",
                              number = 10, repeats = 3)

# Train the model
ols_macro <- train(macro_eq, data = final_df, method = "lm",
                   trControl = train.control)
rmse_ls['ols_macro'] = cal_rmse(predict(ols_macro,final_df[all.vars(macro_eq)[-1]]), final_df$GDF

pred_train_list['ols_macro'] = predict(ols_macro,final_df[all.vars(macro_eq)[-1]])
```

Combined Model

```
# Set seed for reproducibility
set.seed(2024)

train.control <- trainControl(method = "repeatedcv",
                              number = 10, repeats = 3)

ols_comb <- train(comb_eq, data = final_df, method = "lm",
                  trControl = train.control)

rmse_ls['ols_comb'] = cal_rmse(predict(ols_comb,final_df[all.vars(comb_eq)[-1]]), final_df$GDP_Gr

pred_train_list['ols_comb'] = predict(ols_comb,final_df[all.vars(comb_eq)[-1]])
```

Prediction

```
load("RData/final_test_df.RData")
outcome_ols_comb <- as.data.frame(final_test_df[,1])
names(outcome_ols_comb)[1] <- "YQ"

# Combined model Prediction
prediction <- predict(ols_comb, newdata = final_test_df)

outcome_ols_comb$NGDP <- prediction

write.csv(outcome_ols_comb, file = "./output/ols_comb.csv", row.names = FALSE)

pred_test_list$ols_comb <- prediction

# Macro variables prediction
outcome_ols_macro <- as.data.frame(final_test_df[,1])
names(outcome_ols_macro)[1] <- "YQ"
prediction <- predict(ols_macro, newdata = final_test_df)

outcome_ols_macro$NGDP <- prediction

write.csv(outcome_ols_macro, file = "./output/ols_macro.csv", row.names = FALSE)

pred_test_list$ols_macro <- prediction
```

b. Random Forest

Then we focus on training random forest models using both macroeconomic variables (`macro_eq`) and a combination of macroeconomic and accounting variables (`comb_eq`). Tuning will be done to determine the optimal `mtry` and the optimal `number of trees`.

Similar coding logic is the part to the OLS codes due to usage of the same package `caret`.

Macro Model

```
# Running this might take more than 3 minutes
# Tuning for best mtry for Macro model

set.seed(2024)
sqrt_num_predictors <- floor(sqrt(length(macro_vars) - 1)) #define max value of mtry
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
tunegrid <- expand.grid(.mtry=c(1:sqrt_num_predictors))
rf_gridsearch <- train(macro_eq, data = final_df, method="rf", metric="RMSE", tuneGrid=tunegrid,

best_mtry_macro <- rf_gridsearch$bestTune$mtry

# Tuning for optimal number of trees
modellist <- list()
for (ntree in c(1000, 1500, 2000, 2500, 3000)) {
 set.seed(2024)
 fit <- train(macro_eq, data = final_df, method="rf", metric="RMSE", tuneGrid=tunegrid, trControl
 key <- toString(ntree)
 modellist[[key]] <- fit
}
# compare results
results <- resamples(modellist)
s <- summary(results)$statistics$RMSE[,4]
best_ntree_macro <- as.numeric(names(s)[which.min(s)]) # ntree = 1000 gives the lowest RMSE
```

```
testSet <- final_df
# With the optimal mtry and number of trees, we can now include it in the final models
```

```
# Random forest models generally do not need cross validation to prevent overfitting
# As that is done so internally when building each tree

rf_macro <- randomForest(macro_eq, data = final_df, mtry = best_mtry_macro, ntree = best_ntree_ma
print(rf_macro)
```

```
Call:
 randomForest(formula = macro_eq, data = final_df, mtry = best_mtry_macro,      ntree =
best_ntree_macro)
               Type of random forest: regression
                     Number of trees: 1000
No. of variables tried at each split: 2

          Mean of squared residuals: 4.553
                    % Var explained: 38.25
```

```
testSet$pred_rf <- predict(rf_macro, newdata = final_df)
pred_train_list['rf_macro'] = testSet$pred_rf
rmse_ls['rf_macro'] = cal_rmse(testSet$pred_rf,testSet$GDP_Growth)
```

Combined Model

```
# Running this might take more than 3 minutes
# Tuning for best mtry for Combined model

set.seed(2024)
sqrt_num_predictors <- floor(sqrt(length(comb_vars) - 1))
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
tunegrid <- expand.grid(.mtry=c(1:sqrt_num_predictors))
rf_gridsearch <- train(comb_eq, data = final_df, method="rf", metric="RMSE", tuneGrid=tunegrid, t

best_mtry_comb <- rf_gridsearch$bestTune$mtry

# Tuning for optimal number of trees
modellist <- list()
for (ntree in c(1000, 1500, 2000, 2500, 3000)) {
 set.seed(2024)
 fit <- train(comb_eq, data = final_df, method="rf", metric="RMSE", tuneGrid=tunegrid, trControl=
 key <- toString(ntree)
 modellist[[key]] <- fit
}
# compare results
results <- resamples(modellist)
s <- summary(results)$statistics$RMSE[,4]
best_ntree_comb <- as.numeric(names(s)[which.min(s)]) # ntree = 2000 gives the lowest RMSE
```

```
rf_comb <- randomForest(comb_eq, data = final_df, mtry = best_mtry_comb, ntree = best_ntree_comb)
print(rf_comb)
```

```
Call:
 randomForest(formula = comb_eq, data = final_df, mtry = best_mtry_comb,      ntree =
best_ntree_comb)
               Type of random forest: regression
                     Number of trees: 2000
No. of variables tried at each split: 2

          Mean of squared residuals: 4.498
                    % Var explained: 39
```

```
testSet$pred_rf <- predict(rf_comb, newdata = final_df)
pred_train_list['rf_comb'] = testSet$pred_rf
rmse_ls['rf_comb'] = cal_rmse(testSet$pred_rf,testSet$GDP_Growth)
```

Prediction

```
# Macro model
outcome_rf_macro <- as.data.frame(final_test_df[,1])
names(outcome_rf_macro)[1] <- "YQ"

prediction <- predict(rf_macro, newdata = final_test_df)

outcome_rf_macro$NGDP <- prediction

write.csv(outcome_rf_macro, file = "./output/outcome_rf_macro.csv", row.names = FALSE)
```

```
pred_test_list$rf_macro <- prediction


# Combined model
outcome_rf_comb <- as.data.frame(final_test_df[,1])
names(outcome_rf_comb)[1] <- "YQ"


prediction <- predict(rf_comb, newdata = final_test_df)

outcome_rf_comb$NGDP <- prediction

write.csv(outcome_rf_comb, file = "./output/outcome_rf_comb.csv", row.names = FALSE)

pred_test_list$rf_comb <- prediction
```

c. Regularization

Here Elastic Net Regression is employed, with two hyperparameters to specify which are alpha and lambda.

Firstly, to tune alpha, a parameter grid search is implemented with parallel computing from package `doParallel` and `foreach`

After securing the optimal alpha, by means of `cv.glmt` we determine two optimal lambda : lambda.min and lambda.1se, corresponding to two elastic net models which are retained: `elastic.min` and `elastic.1se` .

The reason behind is the respective advantages of two models on model complexity and generalization performance

- lambda.min aims to minimize the error on the training data, potentially leading to a more complex model,
- lambda.1se aims for a simpler model with slightly higher error but potentially better generalization performance on new data.

Macro Model

```
# Performs a loop to find the best alpha for the macro model

x <- model.matrix(macro_eq, data = final_df)[, -1]
y <- model.frame(macro_eq, data = final_df)[ , "GDP_Growth"]
xp <- model.matrix(macro_eq, data = final_df, na.action = 'na.pass')[ , -1]

set.seed(2024)
library(doParallel)
```

```
Loading required package: foreach


Attaching package: 'foreach'

The following objects are masked from 'package:purrr':

    accumulate, when

Loading required package: iterators

Loading required package: parallel
```

```
library(foreach)

list.of.fits <- list() # init a blank list

no_cores <- detectCores() -1 # reserve one core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('list.of.fits', 'x', 'y')) # import objects outside
clusterEvalQ(cl, expr= { # launch library to be used in parallel computing
  library(glmnet)
})
```

```
[[1]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[2]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[3]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"
```

```
[[4]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[5]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[6]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[7]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"
```

```
list.of.fits <- foreach(i = 0:10) %dopar% { #foreach will return a list
  set.seed(2024)
  cv.glmnet(x = x, y = y, family = "gaussian", alpha = i/10,
            type.measure = "mse")
}



stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)
  names(list.of.fits)[i+1] <- c(fit.name)
}

results <- data.frame()
for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)

  ## Use each model to predict 'y' given the Testing dataset
  pred <- predict(list.of.fits[[fit.name]], xp, type = "response",
                  s = list.of.fits[[fit.name]]$lambda.min)

  # Calculate RMSE
  rmse <- sqrt(mean((pred - final_df$GDP_Growth)^2))

  ## Store the results
  temp <- data.frame(alpha = i/10, rmse = rmse, fit.name = fit.name)
  results <- rbind(results, temp)
}
html_df(results)
```

alpha	rmse	fit.name
0.0	1.768	alpha0
0.1	1.749	alpha0.1
0.2	1.746	alpha0.2
0.3	1.745	alpha0.3
0.4	1.742	alpha0.4
0.5	1.741	alpha0.5
0.6	1.743	alpha0.6
0.7	1.743	alpha0.7
0.8	1.742	alpha0.8
0.9	1.742	alpha0.9
1.0	1.741	alpha1

```
# Find the row index corresponding to the minimum RMSE
min_rmse_row <- which.min(results$rmse)

# Extract the alpha corresponding to the minimum RMSE
best_alpha_macro <- results$alpha[min_rmse_row]

# Print the alpha corresponding to the minimum RMSE
print(best_alpha_macro)
```

```
[1] 0.5
```

```
x <- model.matrix(macro_eq, data = final_df)[, -1]
y <- model.frame(macro_eq, data = final_df)[ , "GDP_Growth"]

set.seed(2024)
```

```
fit_macro <- cv.glmnet(y = y,
                  x = x,
                  family = "gaussian",
                  alpha = best_alpha_macro,
                  type.measure = "mse")

print(fit_macro)
```

```
Call:  glmnet::cv.glmnet(x = x, y = y, family = "gaussian", alpha = best_alpha_macro,
type.measure = "mse")

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min  0.140    34    4.35 0.615      14
1se  0.745    16    4.92 1.257       8
```

```
pred_train_list['elastic_macro_min'] = predict(fit_macro, x, s = "lambda.min")
pred_train_list['elastic_macro_1se'] = predict(fit_macro, x, s = "lambda.1se")
rmse_ls['elastic_macro_lambdamin'] = cal_rmse(predict(fit_macro, x, s = "lambda.min"),y)
rmse_ls['elastic_macro_lambda1se'] = cal_rmse(predict(fit_macro, x, s = "lambda.1se"),y)
```

Combined Model

```
# Performs a loop to find the best alpha for the combined model
x <- model.matrix(comb_eq, data = final_df)[, -1]
y <- model.frame(comb_eq, data = final_df)[ , "GDP_Growth"]
xp <- model.matrix(comb_eq, data = final_df, na.action = 'na.pass')[ , -1]
#xp <- model.matrix(macro_eq, data = final_df, na.action = 'na.pass')[ , -1]

set.seed(2024)
library(doParallel)
library(foreach)

list.of.fits <- list() # init a blank list

no_cores <- detectCores() -1 # reserve one core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('list.of.fits', 'x', 'y')) # import objects outside
clusterEvalQ(cl, expr= { # launch library to be used in parallel computing
  library(glmnet)
})
```

```
[[1]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[2]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[3]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[4]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[5]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[6]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[7]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"
```

```
list.of.fits <- foreach(i = 0:10) %dopar% { #foreach will return a list
  set.seed(2024)
  cv.glmnet(x = x, y = y, family = "gaussian", alpha = i/10,
            type.measure = "mse")
}

stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing
```

```
for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)
  names(list.of.fits)[i+1] <- c(fit.name)
}

results <- data.frame()
for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)

  ## Use each model to predict 'y' given the Testing dataset
  pred <- predict(list.of.fits[[fit.name]], xp, type = "response",
                  s = list.of.fits[[fit.name]]$lambda.min)

  # Calculate RMSE
  rmse <- sqrt(mean((pred - final_df$GDP_Growth)^2))

  ## Store the results
  temp <- data.frame(alpha = i/10, rmse = rmse, fit.name = fit.name)
  results <- rbind(results, temp)
}
html_df(results)
```

alpha	rmse	fit.name
0.0	1.670	alpha0
0.1	1.626	alpha0.1
0.2	1.620	alpha0.2
0.3	1.628	alpha0.3
0.4	1.623	alpha0.4
0.5	1.620	alpha0.5
0.6	1.618	alpha0.6
0.7	1.624	alpha0.7
0.8	1.623	alpha0.8
0.9	1.622	alpha0.9
1.0	1.621	alpha1

```
# Find the row index corresponding to the minimum RMSE
min_rmse_row <- which.min(results$rmse)

# Extract the alpha corresponding to the minimum RMSE
best_alpha_comb <- results$alpha[min_rmse_row]

# Print the alpha corresponding to the minimum RMSE
print(best_alpha_comb)
```

```
[1] 0.6
```

```
x <- model.matrix(macro_eq, data = final_df)[, -1]
y <- model.frame(macro_eq, data = final_df)[ , "GDP_Growth"]

set.seed(2024)

fit_macro <- cv.glmnet(y = y,
                  x = x,
                  family = "gaussian",
                  alpha = best_alpha_macro,
                  type.measure = "mse")

print(fit_macro)
```

```
Call:  glmnet::cv.glmnet(x = x, y = y, family = "gaussian", alpha = best_alpha_macro,
type.measure = "mse")

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min  0.140    34    4.35 0.615      14
1se  0.745    16    4.92 1.257       8
```

```
# Combined equation
x <- model.matrix(comb_eq, data = final_df)[, -1]
y <- model.frame(comb_eq, data = final_df)[ , "GDP_Growth"]

fit_comb <- cv.glmnet(y = y,  # Finish this (5 lines)
                  x = x,
                  family = "gaussian",
```

```
                alpha = best_alpha_comb,
                type.measure = "mse")

print(fit_comb)
```

```
Call:  glmnet::cv.glmnet(x = x, y = y, family = "gaussian", alpha = best_alpha_comb,
type.measure = "mse")

Measure: Mean-Squared Error

    Lambda Index Measure    SE Nonzero
min  0.109    35    4.24 0.886      21
1se  0.640    16    5.07 1.589      10
```

```
pred_train_list['elastic_comb_min'] = predict(fit_comb, x, s = "lambda.min")
pred_train_list['elastic_comb_1se'] = predict(fit_comb, x, s = "lambda.1se")
rmse_ls['elastic_comb_lambdamin'] = cal_rmse(predict(fit_comb, x, s = "lambda.min"),y)
rmse_ls['elastic_comb_lambda1se'] = cal_rmse(predict(fit_comb, x, s = "lambda.1se"),y)
```

Prediction

```
# Macro model Prediction using lambda.min
outcome_elastic_macro_min <- as.data.frame(final_test_df[,1])
names(outcome_elastic_macro_min)[1] <- "YQ"

final_test_df$GDP_Growth <- 0

xp <- model.matrix(macro_eq, data = final_test_df, na.action = 'na.pass')[ , -1]

prediction <- predict(fit_macro, xp, s = "lambda.min")

outcome_elastic_macro_min$NGDP <- prediction

write.csv(outcome_elastic_macro_min, file = "./output/elastic_macro_lambda_min.csv", row.names =

pred_test_list$elastic_macro_min <- prediction

# Macro model Prediction using lambda.1se
outcome_elastic_macro_1se <- as.data.frame(final_test_df[,1])
names(outcome_elastic_macro_1se)[1] <- "YQ"
xp <- model.matrix(macro_eq, data = final_test_df, na.action = 'na.pass')[ , -1]

prediction <- predict(fit_macro, xp, s = "lambda.1se")

outcome_elastic_macro_1se$NGDP <- prediction

write.csv(outcome_elastic_macro_1se, file = "./output/elastic_macro_lambda_1se.csv", row.names =
pred_test_list$elastic_macro_1se <- prediction

# Combined model Prediction using lambda.min
outcome_elastic_comb_min <- as.data.frame(final_test_df[,1])
names(outcome_elastic_comb_min)[1] <- "YQ"

xp <- model.matrix(comb_eq, data = final_test_df, na.action = 'na.pass')[ , -1]
prediction <- predict(fit_comb, xp, s = "lambda.min")

outcome_elastic_comb_min$NGDP <- prediction

write.csv(outcome_elastic_comb_min, file = "./output/elastic_comb_lambda_min.csv", row.names = FA
pred_test_list$elastic_comb_min <- prediction

# Combined model Prediction using lambda.1se
outcome_elastic_comb_1se <- as.data.frame(final_test_df[,1])
names(outcome_elastic_comb_1se)[1] <- "YQ"

xp <- model.matrix(comb_eq, data = final_test_df, na.action = 'na.pass')[ , -1]
prediction <- predict(fit_comb, xp, s = "lambda.1se")

outcome_elastic_comb_1se$NGDP <- prediction

write.csv(outcome_elastic_comb_1se, file = "./output/elastic_comb_lambda_1se.csv", row.names = FA

pred_test_list$elastic_comb_1se <- prediction
```

d. XGBoost

Here XGBoost is adopted. Allowing for its plentiful hyperparameters to tune, `ParBayesianOptimizer` is introduced for tuning more efficiently, while `doParallel` is still employed for parallel computing to boost the tuning process.

As a compromise for time saving, hyperparameters which are optimized based on combined data will be fed into the model on macro data without additional specific tuning.

Optimizing XGBoost

```
x = as.matrix(final_df[ , all.vars(comb_eq)[-1] ])
y = as.matrix(final_df[ , all.vars(comb_eq)[1] ])
#predictors <-all.vars(comb_eq)[-1]#
#outcomeName <- all.vars(comb_eq)[1]

scoring_function <- function(
  eta, gamma, max_depth, min_child_weight, subsample, nfold) {

  dtrain <- xgb.DMatrix(x, label = y, missing = NA)

  pars <- list(
    eta = eta,
    gamma = gamma,
    max_depth = max_depth,
    min_child_weight = min_child_weight,
    subsample = subsample,

    booster = "gbtree",
    objective = "reg:squarederror",
    eval_metric = "rmse",
    verbosity = 0
  )

  xgbcv <- xgb.cv(
    params = pars,
    data = dtrain,

    nfold = nfold,

    nrounds = 200,
    prediction = TRUE,
    showsd = TRUE,
    early_stopping_rounds = 10,
    #maximize = TRUE,
    maximize = FALSE,# the smaller the better
    stratified = TRUE
  )

  # required by the package, the output must be a list
  # with at least one element of "Score", the measure to optimize
  # Score must start with capital S
  # For this case, we also report the best num of iteration
  return(
    list(
      # Score = max(xgbcv$evaluation_log$test_rmse_mean),
      #Score = min(xgbcv$evaluation_log$test_rmse_mean),# remember to change max to min! ! !
      Score = -min(xgbcv$evaluation_log$test_rmse_mean),# remember to add minus sign to maximize!
      nrounds = xgbcv$best_iteration
    )
  )
}


# define the lower and upper bounds for each function (FUN) input
bounds <- list(
  eta = c(0.01, 1),
  gamma =c(0.01, 10),
  max_depth = c(2L, 10L), # L means integers
  min_child_weight = c(1, 10),
  subsample = c(0.25, 1),
  nfold = c(3L, 10L)
)

set.seed(2021)

#library(doParallel)
no_cores <- detectCores() - 1 # 1 core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('x','y')) # import objects outside
clusterEvalQ(cl,expr= { # launch library to be used in FUN
  library(xgboost)
  set.seed(2024)
})
```

```
[[1]]
NULL
```

```
[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

[[5]]
NULL

[[6]]
NULL

[[7]]
NULL
```

```
time_withparallel <- system.time(
opt_obj <- bayesOpt(
  FUN = scoring_function,
  bounds = bounds,
  initPoints = 9,
  iters.n = 7,
  parallel = TRUE
))
```

```
stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

# again, have to use capital letters required by the package
# take a look at the output
opt_obj$scoreSummary
```

```
    Epoch Iteration     eta    gamma max_depth min_child_weight subsample nfold
    <num>     <int>   <num>    <num>     <num>            <num>     <num> <num>
 1:     0         1 0.60945  2.33211         3            9.469    0.5827     3
 2:     0         2 0.42381  7.42603         4            8.686    0.3688     7
 3:     0         3 0.13144  0.04782         3            1.034    0.7836     4
 4:     0         4 0.25313  5.47268         9            6.863    0.2639     9
 5:     0         5 0.73377  1.50139         5            2.034    0.6422     9
 6:     0         6 0.86640  3.93229         9            5.619    0.8910     6
 7:     0         7 0.89304  9.58271         6            7.418    0.4648     8
 8:     0         8 0.07007  5.59968         7            3.845    0.9390     5
 9:     0         9 0.55481  8.74252         8            4.604    0.6683     8
10:     1        10 0.27797 10.00000        10            1.000    1.0000     3
11:     2        11 0.11471 10.00000         2            1.000    1.0000    10
12:     3        12 0.01000 10.00000         2            1.000    0.2500     3
13:     4        13 0.01557  0.01000        10            1.000    1.0000    10
14:     5        14 0.24600 10.00000         5            2.111    1.0000     3
15:     6        15 0.08284  8.64792         4           10.000    1.0000     3
16:     7        16 0.09144  5.84849         5            1.962    0.2500    10
    gpUtility acqOptimum inBounds Elapsed  Score nrounds errorMessage
        <num>     <lgcl>   <lgcl>   <num>  <num>   <int>       <lgcl>
 1:        NA      FALSE     TRUE   0.024 -2.513       7           NA
 2:        NA      FALSE     TRUE   0.056 -2.393      20           NA
 3:        NA      FALSE     TRUE   0.068 -2.058      18           NA
 4:        NA      FALSE     TRUE   0.071 -2.258      22           NA
 5:        NA      FALSE     TRUE   0.093 -2.443       5           NA
 6:        NA      FALSE     TRUE   0.061 -2.516       2           NA
 7:        NA      FALSE     TRUE   0.038 -2.519       1           NA
 8:        NA      FALSE     TRUE   0.166 -2.093      92           NA
 9:        NA      FALSE     TRUE   0.035 -2.238       4           NA
10:    0.5107       TRUE     TRUE   0.023 -2.126       8           NA
11:    0.4553       TRUE     TRUE   0.080 -2.067      22           NA
12:    0.4223       TRUE     TRUE   0.092 -2.181     200           NA
13:    0.4008       TRUE     TRUE   0.886 -2.499     199           NA
14:    0.6089       TRUE     TRUE   0.017 -2.348      10           NA
15:    0.5932       TRUE     TRUE   0.097 -2.271     188           NA
16:    0.5478       TRUE     TRUE   0.080 -2.005      43           NA
```

```
# the built-in function output the FUN input arguments only.
getBestPars(opt_obj)
```

```
$eta
[1] 0.09144

$gamma
[1] 5.848

$max_depth
[1] 5
```

```
$min_child_weight
[1] 1.962

$subsample
[1] 0.25

$nfold
[1] 10
```

```
# take the optimal parameters for xgboost()
params <- list(eta = getBestPars(opt_obj)[[1]],
               gamma = getBestPars(opt_obj)[[2]],
               max_depth = getBestPars(opt_obj)[[3]],
               min_child_weight = getBestPars(opt_obj)[[4]],
               subsample = getBestPars(opt_obj)[[5]],
               nfold = getBestPars(opt_obj)[[6]],
               objective = "reg:squarederror")

# the numrounds which gives the max Score (auc)
numrounds <- opt_obj$scoreSummary$nrounds[
  which(opt_obj$scoreSummary$Score
        == max(opt_obj$scoreSummary$Score))] # still grab the max score becasue our score is nega
numrounds
```

```
[1] 43
```

Macro Model

```
# macro
x = as.matrix(final_df[ , all.vars(macro_eq)[-1] ])
y = as.matrix(final_df[ , all.vars(macro_eq)[1] ])

xgb1 <- xgboost(params = params,
                    data = x,
                    label = y,
                    nrounds = numrounds,
                    eval_metric = "rmse")
```

```
[23:26:33] WARNING: src/learner.cc:767:
Parameters: { "nfold" } are not used.

[1] train-rmse:4.619959
[2] train-rmse:4.345991
[3] train-rmse:4.019342
[4] train-rmse:3.781816
[5] train-rmse:3.513986
[6] train-rmse:3.295017
[7] train-rmse:3.107536
[8] train-rmse:2.913783
[9] train-rmse:2.769593
[10]    train-rmse:2.639194
[11]    train-rmse:2.534024
[12]    train-rmse:2.450767
[13]    train-rmse:2.355679
[14]    train-rmse:2.271198
[15]    train-rmse:2.239792
[16]    train-rmse:2.196941
[17]    train-rmse:2.116473
[18]    train-rmse:2.071280
[19]    train-rmse:2.026819
[20]    train-rmse:1.988297
[21]    train-rmse:1.966193
[22]    train-rmse:1.951384
[23]    train-rmse:1.926660
[24]    train-rmse:1.890486
[25]    train-rmse:1.863963
[26]    train-rmse:1.836303
[27]    train-rmse:1.811276
[28]    train-rmse:1.802100
[29]    train-rmse:1.798675
[30]    train-rmse:1.770690
[31]    train-rmse:1.740058
[32]    train-rmse:1.714293
[33]    train-rmse:1.706778
[34]    train-rmse:1.680621
[35]    train-rmse:1.663828
[36]    train-rmse:1.653146
[37]    train-rmse:1.637969
[38]    train-rmse:1.642164
[39]    train-rmse:1.617676
[40]    train-rmse:1.604060
```

```
[41]    train-rmse:1.590045
[42]    train-rmse:1.553778
[43]    train-rmse:1.547127
```

```
pred_train_list['xgb_macro'] = predict(xgb1,x)
rmse_ls['xgb_macro'] = cal_rmse(predict(xgb1,x),y)
```

Combined Model

```
# comb
x = as.matrix(final_df[ , all.vars(comb_eq)[-1] ])
y = as.matrix(final_df[ , all.vars(comb_eq)[1] ])

xgb2 <- xgboost(params = params,
                      data = x,
                      label = y,
                      nrounds = numrounds,
                      eval_metric = "rmse")
```

```
[23:26:33] WARNING: src/learner.cc:767:
Parameters: { "nfold" } are not used.

[1] train-rmse:4.583641
[2] train-rmse:4.317018
[3] train-rmse:4.044097
[4] train-rmse:3.773303
[5] train-rmse:3.594934
[6] train-rmse:3.418059
[7] train-rmse:3.242870
[8] train-rmse:3.084705
[9] train-rmse:2.968999
[10]    train-rmse:2.830743
[11]    train-rmse:2.646628
[12]    train-rmse:2.556321
[13]    train-rmse:2.417856
[14]    train-rmse:2.321835
[15]    train-rmse:2.253335
[16]    train-rmse:2.189009
[17]    train-rmse:2.133359
[18]    train-rmse:2.050633
[19]    train-rmse:2.002302
[20]    train-rmse:1.951965
[21]    train-rmse:1.884886
[22]    train-rmse:1.818833
[23]    train-rmse:1.780327
[24]    train-rmse:1.716599
[25]    train-rmse:1.713168
[26]    train-rmse:1.692436
[27]    train-rmse:1.680407
[28]    train-rmse:1.646621
[29]    train-rmse:1.630840
[30]    train-rmse:1.616117
[31]    train-rmse:1.570144
[32]    train-rmse:1.543709
[33]    train-rmse:1.530133
[34]    train-rmse:1.494663
[35]    train-rmse:1.482375
[36]    train-rmse:1.487492
[37]    train-rmse:1.477945
[38]    train-rmse:1.444839
[39]    train-rmse:1.418606
[40]    train-rmse:1.413600
[41]    train-rmse:1.409118
[42]    train-rmse:1.407287
[43]    train-rmse:1.388808
```

```
pred_train_list['xgb_comb'] = predict(xgb2,x)
rmse_ls['xgb_comb'] = cal_rmse(predict(xgb2,x),y)
```

Prediction

```
# Macro
x <- model.matrix(macro_eq, data = final_df)[, -1]
y <- model.frame(macro_eq, data = final_df)[ , "GDP_Growth"]

xvals <- model.matrix(macro_eq, data = final_test_df)[, -1]
yvals <- model.frame(macro_eq, data = final_test_df)[ , "GDP_Growth"]

outcome_xgb_macro <- as.data.frame(final_test_df[,1])
names(outcome_xgb_macro)[1] <- "YQ"

prediction <- predict(xgb1, xvals)
```

```
outcome_xgb_macro$NGDP <- prediction

write.csv(outcome_xgb_macro, file = "./output/outcome_xgb_macro.csv", row.names = FALSE)

pred_test_list$xgb_macro <- prediction
```

```
# Macro
x <- model.matrix(comb_eq, data = final_df)[, -1]
y <- model.frame(comb_eq, data = final_df)[ , "GDP_Growth"]

xvals <- model.matrix(comb_eq, data = final_test_df)[, -1]
yvals <- model.frame(comb_eq, data = final_test_df)[ , "GDP_Growth"]

outcome_xgb_comb <- as.data.frame(final_test_df[,1])
names(outcome_xgb_comb)[1] <- "YQ"

prediction <- predict(xgb2, xvals)

outcome_xgb_comb$NGDP <- prediction

write.csv(outcome_xgb_comb, file = "./output/outcome_xgb_comb.csv", row.names = FALSE)
pred_test_list$xgb_comb <- prediction
```

e. Ensemble

The ensemble part is aimed at creating a ensemble model based on the predictions from multiple individual models which are trained previously.

The ensemble technique is straightforward, yet theoretically, effective in improving predictive performance by leveraging the diversity of multiple models. By combining predictions from different models, it helps mitigate the risk of overfitting and can often lead to more robust and reliable predictions.

Here we adopt two ensembles:

- one simple ensemble using Simple Average
  - which calculate the average of predictions generated by base models trained previously
- one complex ensemble using XGBoost
  - which similar to base model above, `ParBayesianOptimization` and `doParallel` are employed for accelerating the hyperparameter tuning process.

Simple Average

```
y <- final_df$GDP_Growth

pred_train_list <- pred_train_list %>%
  mutate(avg_NGDP_macro = (ols_macro + rf_macro + elastic_macro_min + elastic_macro_1se + xgb_mac
         avg_NGDP_comb = (ols_comb + rf_comb + elastic_comb_min + elastic_macro_1se + xgb_macro)/

rmse_ls['ensemble_avg_macro'] <- cal_rmse(pred_train_list$avg_NGDP_macro, y)
rmse_ls['ensemble_avg_comb'] <- cal_rmse(pred_train_list$avg_NGDP_comb, y)
```

```
pred_test_list <- pred_test_list %>%
  mutate(avg_NGDP_macro = (ols_macro + rf_macro + elastic_macro_min + elastic_macro_1se + xgb_mac
         avg_NGDP_comb = (ols_comb + rf_comb + elastic_comb_min + elastic_macro_1se + xgb_macro)/

simple_average_macro <- pred_test_list[c("YQ","avg_NGDP_macro")] %>%
  rename(NGDP = "avg_NGDP_macro")
write.csv(simple_average_macro, file = "./output/simple_average_macro.csv", row.names = FALSE)

simple_average_comb <- pred_test_list[c("YQ","avg_NGDP_comb")] %>%
  rename(NGDP = "avg_NGDP_comb")
write.csv(simple_average_comb, file = "./output/simple_average_comb.csv", row.names = FALSE)
```

Ensemble XGBoost

```
#x_test = pred_test_list[2:11]
x = as.matrix(pred_train_list[c('ols_macro','rf_macro',"elastic_macro_min","elastic_macro_1se","x
y = as.matrix(final_df[ , all.vars(comb_eq)[1] ])
#predictors <-all.vars(comb_eq)[-1]#
#outcomeName <- all.vars(comb_eq)[1]

scoring_function <- function(
  eta, gamma, max_depth, min_child_weight, subsample, nfold) {
```

```
  dtrain <- xgb.DMatrix(x, label = y, missing = NA)

  pars <- list(
    eta = eta,
    gamma = gamma,
    max_depth = max_depth,
    min_child_weight = min_child_weight,
    subsample = subsample,

    booster = "gbtree",
    objective = "reg:squarederror",
    eval_metric = "rmse",
    verbosity = 0
  )

  xgbcv <- xgb.cv(
    params = pars,
    data = dtrain,

    nfold = nfold,

    nrounds = 200,
    prediction = TRUE,
    showsd = TRUE,
    early_stopping_rounds = 10,
    #maximize = TRUE,
    maximize = FALSE,# the smaller the better
    stratified = TRUE
  )

  # required by the package, the output must be a list
  # with at least one element of "Score", the measure to optimize
  # Score must start with capital S
  # For this case, we also report the best num of iteration
  return(
    list(
      # Score = max(xgbcv$evaluation_log$test_rmse_mean),
      #Score = min(xgbcv$evaluation_log$test_rmse_mean),# remember to change max to min! ! !
      Score = -min(xgbcv$evaluation_log$test_rmse_mean),# remember to add minus sign to maximize!
      nrounds = xgbcv$best_iteration
    )
  )
}


# define the lower and upper bounds for each function (FUN) input
bounds <- list(
  eta = c(0.01, 1),
  gamma =c(0.01, 10),
  max_depth = c(2L, 10L), # L means integers
  min_child_weight = c(1, 10),
  subsample = c(0.25, 1),
  nfold = c(3L, 10L)
)

set.seed(2021)

#library(doParallel)
no_cores <- detectCores() - 1 # 1 core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('x','y')) # import objects outside
clusterEvalQ(cl,expr= { # launch library to be used in FUN
  library(xgboost)
  set.seed(2024)
})
```

```
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

[[5]]
NULL

[[6]]
```

```
NULL

[[7]]
NULL
```

```
time_withparallel <- system.time(
opt_obj <- bayesOpt(
  FUN = scoring_function,
  bounds = bounds,
  initPoints = 7,
  iters.n = 5,
  parallel = TRUE
))
```

```
stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

# again, have to use capital letters required by the package
# take a look at the output
opt_obj$scoreSummary
```

```
    Epoch Iteration    eta   gamma max_depth min_child_weight subsample nfold
    <num>     <int>  <num>   <num>     <num>            <num>     <num> <num>
 1:     0         1 0.3938  0.5924         3            9.489    0.6108     9
 2:     0         2 0.2290  3.3835         9            7.534    0.9222     6
 3:     0         3 0.1439  2.8513         9            3.172    0.4921     8
 4:     0         4 0.7245  4.9805         6            7.219    0.3109     3
 5:     0         5 0.9152  8.9258         5            5.144    0.3588     5
 6:     0         6 0.7075  5.8343         3            1.824    0.8748     6
 7:     0         7 0.4549  7.7742         7            4.488    0.7575     7
 8:     1         8 1.0000  0.3543         2            1.000    1.0000     6
 9:     2         9 0.0100  0.0100         2            1.000    1.0000    10
10:     3        10 1.0000 10.0000         2           10.000    1.0000    10
11:     4        11 0.0100  0.0100        10            1.000    1.0000     6
12:     5        12 1.0000  0.0100         2            1.000    1.0000     4
    gpUtility acqOptimum inBounds Elapsed   Score nrounds errorMessage
        <num>     <lgcl>   <lgcl>   <num>   <num>   <int>       <lgcl>
 1:        NA      FALSE     TRUE   0.024 -1.1724      11           NA
 2:        NA      FALSE     TRUE   0.031 -1.2183      24           NA
 3:        NA      FALSE     TRUE   0.054 -1.0020      58           NA
 4:        NA      FALSE     TRUE   0.013 -1.6490       5           NA
 5:        NA      FALSE     TRUE   0.023 -1.5642      24           NA
 6:        NA      FALSE     TRUE   0.027 -1.0122      16           NA
 7:        NA      FALSE     TRUE   0.024 -1.2219       9           NA
 8:    0.5594       TRUE     TRUE   0.011 -0.8506      14           NA
 9:    0.6685       TRUE     TRUE   0.193 -1.2895     200           NA
10:    0.4001       TRUE     TRUE   0.013 -1.2907       6           NA
11:    0.3688       TRUE     TRUE   0.105 -1.1506     200           NA
12:    0.3323       TRUE     TRUE   0.006 -0.9053       5           NA
```

```
# the built-in function output the FUN input arguments only.
getBestPars(opt_obj)
```

```
$eta
[1] 1

$gamma
[1] 0.3543

$max_depth
[1] 2

$min_child_weight
[1] 1

$subsample
[1] 1

$nfold
[1] 6
```

```
# take the optimal parameters for xgboost()
params <- list(eta = getBestPars(opt_obj)[[1]],
               gamma = getBestPars(opt_obj)[[2]],
               max_depth = getBestPars(opt_obj)[[3]],
               min_child_weight = getBestPars(opt_obj)[[4]],
               subsample = getBestPars(opt_obj)[[5]],
               nfold = getBestPars(opt_obj)[[6]],
               objective = "reg:squarederror")

# the numrounds which gives the max Score (auc)
numrounds <- opt_obj$scoreSummary$nrounds[
  which(opt_obj$scoreSummary$Score
```

```
        == max(opt_obj$scoreSummary$Score))] # still grab the max score becasue our score is nega
numrounds
```

```
[1] 14
```

```
x = as.matrix(pred_train_list[c('ols_macro','rf_macro',"elastic_macro_min","elastic_macro_1se","
y = as.matrix(final_df[ , all.vars(comb_eq)[1] ])
xgb_ensemble1 <- xgboost(params = params,
                     data = x,
                     label = y,
                     nrounds = numrounds,
                     eval_metric = "rmse")
```

```
[23:26:48] WARNING: src/learner.cc:767:
Parameters: { "nfold" } are not used.

[1] train-rmse:1.066218
[2] train-rmse:0.889208
[3] train-rmse:0.782006
[4] train-rmse:0.664374
[5] train-rmse:0.592083
[6] train-rmse:0.491530
[7] train-rmse:0.462890
[8] train-rmse:0.421027
[9] train-rmse:0.400286
[10]    train-rmse:0.368491
[11]    train-rmse:0.330736
[12]    train-rmse:0.318368
[13]    train-rmse:0.318368
[14]    train-rmse:0.318368
```

```
#pred_train_list['xgb_comb'] = predict(xgb2,x)
rmse_ls['ensemble_xgb_macro'] = cal_rmse(predict(xgb_ensemble1,x),y)


#x_test = as.matrix(pred_test_list[2:11])
x_test = as.matrix(pred_test_list[colnames(x)])

pred_test_list['ensemble_xgb_macro'] = predict(xgb_ensemble1,x_test)
ensemble_xgb =  pred_test_list[c("YQ")] %>% mutate( NGDP = predict(xgb_ensemble1,x_test))


write.csv(ensemble_xgb, file = "./output/ensemble_xgb_macro.csv", row.names = FALSE)
```

```
x = as.matrix(pred_train_list[2:11])
y = as.matrix(final_df[ , all.vars(comb_eq)[1] ])


scoring_function <- function(
  eta, gamma, max_depth, min_child_weight, subsample, nfold) {

  dtrain <- xgb.DMatrix(x, label = y, missing = NA)

  pars <- list(
    eta = eta,
    gamma = gamma,
    max_depth = max_depth,
    min_child_weight = min_child_weight,
    subsample = subsample,

    booster = "gbtree",
    objective = "reg:squarederror",
    eval_metric = "rmse",
    verbosity = 0
  )

  xgbcv <- xgb.cv(
    params = pars,
    data = dtrain,

    nfold = nfold,

    nrounds = 200,
    prediction = TRUE,
    showsd = TRUE,
    early_stopping_rounds = 10,
    #maximize = TRUE,
    maximize = FALSE,# the smaller the better
    stratified = TRUE
  )
```

```
  # required by the package, the output must be a list
  # with at least one element of "Score", the measure to optimize
  # Score must start with capital S
  # For this case, we also report the best num of iteration
  return(
    list(
      # Score = max(xgbcv$evaluation_log$test_rmse_mean),
      #Score = min(xgbcv$evaluation_log$test_rmse_mean),# remember to change max to min! ! !
      Score = -min(xgbcv$evaluation_log$test_rmse_mean),# remember to add minus sign to maximize!
      nrounds = xgbcv$best_iteration
    )
  )
}


# define the lower and upper bounds for each function (FUN) input
bounds <- list(
  eta = c(0.01, 1),
  gamma =c(0.01, 10),
  max_depth = c(2L, 10L), # L means integers
  min_child_weight = c(1, 10),
  subsample = c(0.25, 1),
  nfold = c(3L, 10L)
)

set.seed(2021)

#library(doParallel)
no_cores <- detectCores() - 1 # 1 core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('x','y')) # import objects outside
clusterEvalQ(cl,expr= { # launch library to be used in FUN
  library(xgboost)
  set.seed(2024)
})
```

```
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

[[5]]
NULL

[[6]]
NULL

[[7]]
NULL
```

```
time_withparallel <- system.time(
opt_obj <- bayesOpt(
  FUN = scoring_function,
  bounds = bounds,
  initPoints = 7,
  iters.n = 5,
  parallel = TRUE
))
```

```
stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

# again, have to use capital letters required by the package
# take a look at the output
opt_obj$scoreSummary
```

```
   Epoch Iteration    eta   gamma max_depth min_child_weight subsample nfold
   <num>     <int>  <num>   <num>     <num>            <num>     <num> <num>
1:     0         1 0.3938  0.5924         3            9.489    0.6108     9
2:     0         2 0.2290  3.3835         9            7.534    0.9222     6
3:     0         3 0.1439  2.8513         9            3.172    0.4921     8
4:     0         4 0.7245  4.9805         6            7.219    0.3109     3
```

```
 5:     0         5 0.9152  8.9258         5            5.144    0.3588     5
 6:     0         6 0.7075  5.8343         3            1.824    0.8748     6
 7:     0         7 0.4549  7.7742         7            4.488    0.7575     7
 8:     1         8 1.0000 10.0000         8            1.000    0.2500    10
 9:     2         9 1.0000 10.0000        10            1.000    0.5612     3
10:     3        10 0.0100  2.0918         2            1.000    1.0000     8
11:     4        11 0.8659  0.5692         4            1.054    0.6391     7
12:     5        12 0.0100  0.0100         2            9.868    0.5100     7
    gpUtility acqOptimum inBounds Elapsed  Score nrounds errorMessage
        <num>     <lgcl>   <lgcl>   <num>  <num>   <int>       <lgcl>
 1:        NA      FALSE     TRUE   0.044 -1.114      15           NA
 2:        NA      FALSE     TRUE   0.046 -1.116      39           NA
 3:        NA      FALSE     TRUE   0.046 -1.054      22           NA
 4:        NA      FALSE     TRUE   0.020 -1.589       5           NA
 5:        NA      FALSE     TRUE   0.030 -1.451       6           NA
 6:        NA      FALSE     TRUE   0.026 -1.088       8           NA
 7:        NA      FALSE     TRUE   0.038 -1.229       8           NA
 8:    0.5712       TRUE     TRUE   0.028 -1.450      17           NA
 9:    0.5245       TRUE     TRUE   0.010 -1.506      11           NA
10:    0.5993       TRUE     TRUE   0.106 -1.271     200           NA
11:    0.4897       TRUE     TRUE   0.012 -1.018       3           NA
12:    0.3814       TRUE     TRUE   0.088 -1.482     200           NA
```

```r
# the built-in function output the FUN input arguments only.
getBestPars(opt_obj)
```

```
$eta
[1] 0.8659

$gamma
[1] 0.5692

$max_depth
[1] 4

$min_child_weight
[1] 1.054

$subsample
[1] 0.6391

$nfold
[1] 7
```

```r
# take the optimal parameters for xgboost()
params <- list(eta = getBestPars(opt_obj)[[1]],
               gamma = getBestPars(opt_obj)[[2]],
               max_depth = getBestPars(opt_obj)[[3]],
               min_child_weight = getBestPars(opt_obj)[[4]],
               subsample = getBestPars(opt_obj)[[5]],
               nfold = getBestPars(opt_obj)[[6]],
               objective = "reg:squarederror")

# the numrounds which gives the max Score (auc)
numrounds <- opt_obj$scoreSummary$nrounds[
  which(opt_obj$scoreSummary$Score
        == max(opt_obj$scoreSummary$Score))] # still grab the max score becasue our score is nega
numrounds
```

```
[1] 3
```

```r
x = as.matrix(pred_train_list[2:11])
y = as.matrix(final_df[ , all.vars(comb_eq)[1] ])
xgb3 <- xgboost(params = params,
                    data = x,
                    label = y,
                    nrounds = numrounds,
                    eval_metric = "rmse")
```

```
[23:27:09] WARNING: src/learner.cc:767:
Parameters: { "nfold" } are not used.

[1] train-rmse:1.518647
[2] train-rmse:0.761273
[3] train-rmse:0.599988
```

```r
#pred_train_list['xgb_comb'] = predict(xgb2,x)
rmse_ls['ensemble_xgb'] = cal_rmse(predict(xgb3,x),y)


#x_test = as.matrix(pred_test_list[2:11])
```

```
x_test = as.matrix(pred_test_list[colnames(x)])

ensemble_xgb =  pred_test_list[c("YQ")] %>% mutate( NGDP = predict(xgb3,x_test))


write.csv(ensemble_xgb, file = "./output/ensemble_xgb_comb.csv", row.names = FALSE)
```

IV. Discussion

A. Interpretation of Results

Important Features

a. OLS

Basically, important features selected vary from models to models due to the differences in algorithm which different algorithms may consider features differently, and complexity which more complex models might be able to capture intricate relationships between features, leading to different sets of important features compared to simpler models.

Overall, before further examining the important features selected by each model, the common features identified as important across models are follows:

- Macro
    - unemployment_rate_change
    - export_change
- Acct
    - shareholder_equity_change (Capital Issuance)
    - debt_invcap (Debt Ratio)
    - asset_change (Firm Size)
    - liabilities_change (Liabilities)

```
summary(ols_comb)
```

```
Call:
lm(formula = .outcome ~ ., data = dat)

Residuals:
    Min      1Q  Median      3Q     Max
-3.0767 -1.0540 -0.0132  1.1262  3.1278

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                -27.464838  16.964111   -1.62   0.1103
forecast_growth              0.319164   0.287968    1.11   0.2718
income_per_capita_growth     0.027260   0.215713    0.13   0.8998
CPI                          0.136981   0.280386    0.49   0.6268
industrial_energy_change     0.154494   0.135791    1.14   0.2594
transport_energy_change     -0.123937   0.131353   -0.94   0.3489
total_energy_change         -0.221858   0.086497   -2.56   0.0126 *
fed_rate                     0.000824   0.226098    0.00   0.9971
fed_rate_change              0.430124   0.804471    0.53   0.5947
earnings_change             -0.503663   0.278231   -1.81   0.0749 .
unemployment_rate_change    -3.898280   1.629929   -2.39   0.0197 *
household_consumption_change  0.403392   0.462424    0.87   0.3862
avg_liquidity_change         0.377041   0.419334    0.90   0.3719
Gross_Saving_change         -0.013055   0.128561   -0.10   0.9194
Gross_Investment_change     -0.020825   0.150104   -0.14   0.8901
import_change               -0.084575   0.083737   -1.01   0.3162
export_change                0.246985   0.074561    3.31   0.0015 **
num_companies_standard      -0.112155   0.780294   -0.14   0.8862
QUARTER                      1.003854   0.664746    1.51   0.1359
shareholder_equity_change    0.917992   0.330556    2.78   0.0072 **
dpr                         -1.238803   1.447505   -0.86   0.3952
debt_invcap                  0.726787   0.313010    2.32   0.0234 *
at_turn                     -0.280289   0.404145   -0.69   0.4904
inv_turn                     0.003255   0.007675    0.42   0.6729
asset_change                -2.222248   0.723178   -3.07   0.0031 **
liabilities_change           1.092600   0.370761    2.95   0.0045 **
current_liabilities_change  -0.025275   0.246363   -0.10   0.9186
cash_ce_change              -0.017072   0.034462   -0.50   0.6220
profit_lct                   0.133779   0.356123    0.38   0.7084
quick_ratio                  0.040375   0.027952    1.44   0.1534
```

```
cf_to_asset                  -0.309408   0.197320   -1.57   0.1217
share_growth                  0.030078   0.036322    0.83   0.4107
price_sales                   0.056946   0.026989    2.11   0.0387 *
net_profit_change            -0.017074   0.028455   -0.60   0.5506
net_op_asset_change           0.207523   0.321895    0.64   0.5214
depr_change                  -0.124280   0.116090   -1.07   0.2883
revenue_change               -0.228818   0.184286   -1.24   0.2188
roa                           0.516329   2.901832    0.18   0.8593
net_profit_margin             0.133046   0.061840    2.15   0.0352 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.72 on 65 degrees of freedom
Multiple R-squared:  0.748, Adjusted R-squared:  0.601
F-statistic: 5.08 on 38 and 65 DF,  p-value: 0.00000000509
```

b. Regularization

From the features selected by the regularization with alpha set as lambda.1se, important features are captured as :

- Macro
  - CPI
  - transport_energy_change
  - fed_rate_change
  - num_companies_standard
- Acct
  - at_turn (Efficiency)
  - quick_ratio (Liquidity)
  - share_growth (Market)

```
coef(fit_comb, s = fit_comb$lambda.min)
```

```
39 x 1 sparse Matrix of class "dgCMatrix"
                                    s1
(Intercept)                   1.399173
forecast_growth               0.252154
income_per_capita_growth      0.074304
CPI                           0.006219
industrial_energy_change      0.054682
transport_energy_change       0.042818
total_energy_change          -0.022025
fed_rate                      .
fed_rate_change               .
earnings_change              -0.316538
unemployment_rate_change     -1.485513
household_consumption_change  0.451175
avg_liquidity_change          0.041503
Gross_Saving_change          -0.031427
Gross_Investment_change       .
import_change                 .
export_change                 0.134554
num_companies_standard        .
QUARTER                       .
shareholder_equity_change     0.082332
dpr                          -0.709377
debt_invcap                   .
at_turn                       .
inv_turn                      .
asset_change                  .
liabilities_change            0.073845
current_liabilities_change    .
cash_ce_change                .
profit_lct                    .
quick_ratio                   .
cf_to_asset                   .
share_growth                  0.024264
price_sales                   0.005862
net_profit_change             .
net_op_asset_change           .
depr_change                  -0.012476
revenue_change                0.006230
roa                           0.910139
net_profit_margin             0.004690
```

c. XGBoost

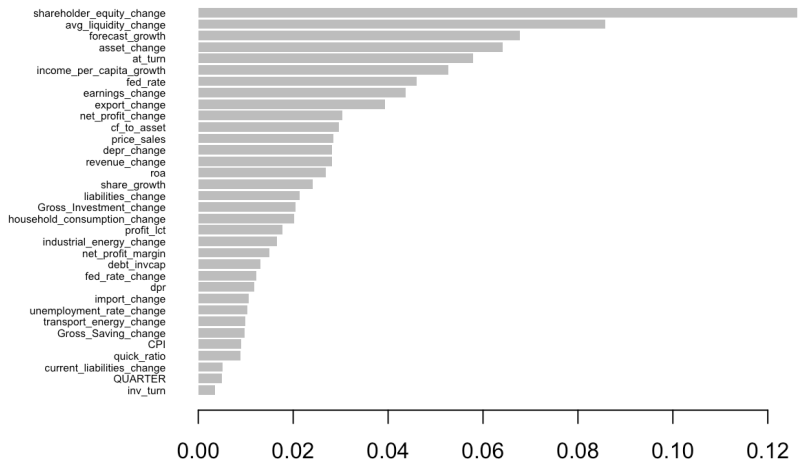
From the importance table offered by XGBoost, top 10 important features are captured as :

- Macro
  - unemployment_rate_change
  - transport_energy_change
  - export_change
  - import_change
- Acct
  - asset_change (Firm Size)
  - price_sales (Market)
  - at_turn (Efficiency)
  - net_profit_margin (Profitability)
  - share_growth (Market)
  - shareholder_equity_change (Capital Issuance)

```
# Show variable importance
importance_matrix <- xgb.importance(model = xgb2)
importance_matrix
```

```
                       Feature     Gain    Cover Frequency
                        <char>    <num>    <num>     <num>
 1:    shareholder_equity_change 0.126261 0.055067  0.055046
 2:         avg_liquidity_change 0.085795 0.070801  0.073394
 3:              forecast_growth 0.067768 0.066173  0.082569
 4:                 asset_change 0.064116 0.063397  0.045872
 5:                      at_turn 0.057908 0.022675  0.018349
 6:     income_per_capita_growth 0.052729 0.022212  0.036697
 7:                     fed_rate 0.045960 0.073577  0.073394
 8:              earnings_change 0.043630 0.028690  0.027523
 9:                export_change 0.039367 0.057381  0.064220
10:            net_profit_change 0.030366 0.037020  0.036697
11:                  cf_to_asset 0.029681 0.038408  0.027523
12:                  price_sales 0.028402 0.035169  0.027523
13:                  depr_change 0.028243 0.023137  0.018349
14:               revenue_change 0.028162 0.025914  0.018349
15:                          roa 0.026832 0.020361  0.018349
16:                share_growth 0.024063 0.032855  0.036697
17:           liabilities_change 0.021387 0.030541  0.027523
18:      Gross_Investment_change 0.020552 0.012494  0.009174
19: household_consumption_change 0.020217 0.030079  0.027523
20:                   profit_lct 0.017713 0.021749  0.027523
21:     industrial_energy_change 0.016626 0.034706  0.027523
22:            net_profit_margin 0.014964 0.019435  0.018349
23:                  debt_invcap 0.013127 0.022212  0.027523
24:              fed_rate_change 0.012227 0.021749  0.027523
25:                          dpr 0.011823 0.015733  0.018349
26:                import_change 0.010572 0.025451  0.027523
27:     unemployment_rate_change 0.010330 0.014345  0.009174
28:      transport_energy_change 0.009872 0.020361  0.018349
29:          Gross_Saving_change 0.009831 0.012031  0.009174
30:                          CPI 0.008961 0.008792  0.018349
31:                 quick_ratio 0.008859 0.016196  0.018349
32:   current_liabilities_change 0.005086 0.009255  0.009174
33:                      QUARTER 0.005042 0.001851  0.009174
34:                    inv_turn 0.003529 0.010180  0.009174
                       Feature     Gain    Cover Frequency
```

```
xgb.plot.importance(importance_matrix)
```



Model Performance

Based on the RMSE results from training and testing data, several conclusions could roughly be drawn:

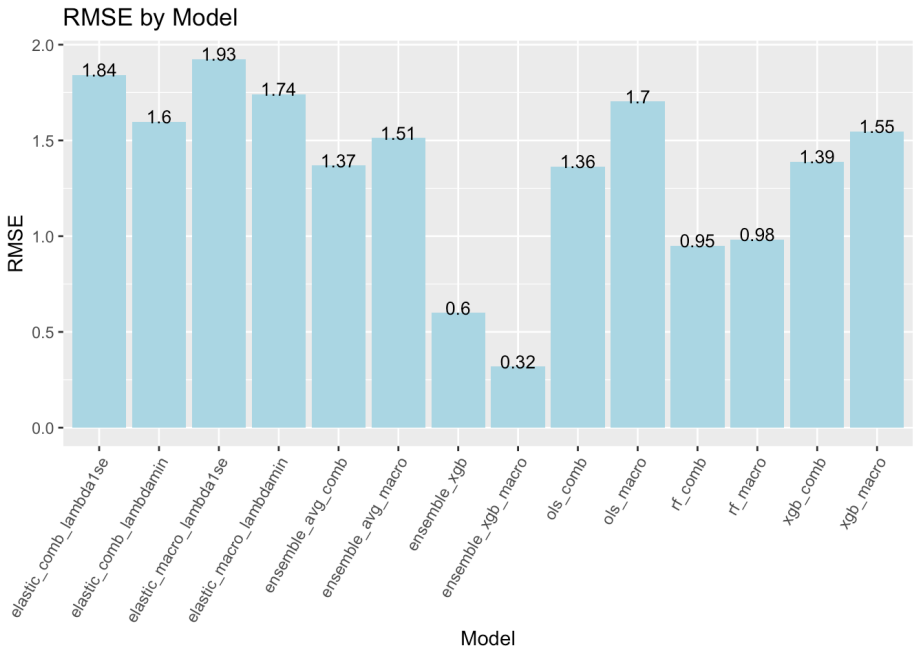
1. Across Basic Models, model fitting performance :
    - XGBoost >Random Forest > OLS > Elastic Net_lambda.min > Elastic Net_lambda.1se
    - Such sequence may result from the difference in the model complexity, implying potential overfitting problem
2. Across Base and Ensemble Models, model fitting performance:
    - Ensemble Models > Base Models
3. Within Base Models, model fitting performance:
    - Combined data models > Macro data models
    - which implies that accounting data plays a conducive role in improving models' predictivity
4. Within Ensemble Models, model fitting performance:
    - XGBosst Ensemble > Simple Average
5. Testing Dataset:
    - Across macro data and combined data
        - On testing dataset, generally models based on combined data outperform those based solely on macro data on the testing dataset.
    - Within periods of the testing dataset:
        - RMSE during COVID period is around ten times as much as non-COVID period.
            - This significant increase in RMSE during the COVID-19 period highlights the substantial impact of the unexpected COVID-19 shock on model performance.
        - The unexpected COVID shock accounts for majority of RMSE on testing dataset.
            - Details on how the unexpected COVID-19 shock contributes to the majority of RMSE on the testing dataset are as follows through visualization of dataframe `rmse_test1` and `rmse_year1`

```
rmse_ls
```

```
$ols_macro
[1] 1.703

$ols_comb
[1] 1.363

$rf_macro
[1] 0.9824

$rf_comb
[1] 0.9504

$elastic_macro_lambdamin
[1] 1.741
```

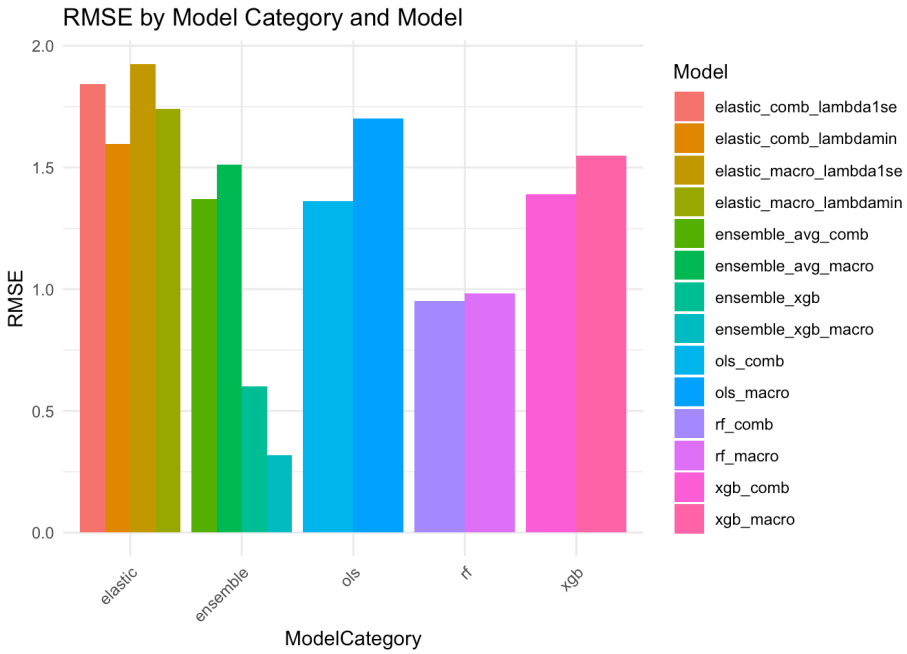
```
$elastic_macro_lambda1se
[1] 1.925

$elastic_comb_lambdamin
[1] 1.597

$elastic_comb_lambda1se
[1] 1.842

$xgb_macro
[1] 1.547

$xgb_comb
[1] 1.389

$ensemble_avg_macro
[1] 1.512

$ensemble_avg_comb
[1] 1.369

$ensemble_xgb_macro
[1] 0.3184

$ensemble_xgb
[1] 0.6
```

```r
## Convert the rmse_ls list to a data frame
rmse_df <- data.frame(Model = names(rmse_ls), RMSE = unlist(rmse_ls))

#options(repr.plot.width = 5, repr.plot.height =4)
# Create the bar chart
ggplot(rmse_df, aes(x = Model, y = RMSE)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  geom_text(aes(label = round(RMSE, 2)), vjust = 0.1, size = 3.5, color = "black") + # Add labels
  labs(title = "RMSE by Model",
       x = "Model",
       y = "RMSE") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1),
        # aspect.ratio = 3 / 2  # Adjust the aspect ratio to make the plot longer
        )
```




```r
## Convert the rmse_ls list to a data frame
rmse_df <- data.frame(Model = names(rmse_ls), RMSE = unlist(rmse_ls))
# Define a function to extract the model category
extract_model_category <- function(model_name) {
  # Split the model name by underscore
  model_parts <- strsplit(model_name, "_")[[1]]
  # Extract the first part as the category
  model_category <- model_parts[1]
  return(model_category)
}

# Apply the function to create a new column
rmse_df$model_category <- sapply(names(rmse_ls), extract_model_category)
```

```
# Create the bar chart
ggplot(rmse_df, aes(x = model_category, y = RMSE, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "RMSE by Model Category and Model",
       x = "ModelCategory",
       y = "RMSE",
       fill = "Model") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Furthermore, this analysis also utilized the GDP growth data from the year of 2016 to 2020 (The testing sample period), based on data from the Federal Reserve of St. Louis, found here.

```
test_GDP = read_csv('Data/Federal_Reserve_GDP_Test_Sample.csv')
```

```
Rows: 20 Columns: 2
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (1): YQ
dbl (1): NGDP

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.
```

```
rmse_test1 = data.frame()
rmse_year1 = data.frame()

for (i in colnames(pred_test_list)[-1]){
  #print(i)

  rmse_test1["2016-2019",i] = cal_rmse(pred_test_list[1:16, i], test_GDP$NGDP[1:16])
  rmse_test1["2020-2020",i] = cal_rmse(pred_test_list[17:20,i], test_GDP$NGDP[17:20])
  rmse_test1["2016-2020",i] = cal_rmse(pred_test_list[[i]], test_GDP$NGDP)
  #calculate RMSE by year from 2016 to 2020
  for (y in seq(1,5)){
        start_idx = 4*y-3
        end_idx = 4*y
        # rmse_year[2015+y,i] = cal_rmse(pred_test_list2[start_idx:end_idx,i], test_GDP$NGDP[star
    # Calculate RMSE for the current year
    rmse_year1[paste0("Y", 2015 + y), i] <- cal_rmse(pred_test_list[start_idx:end_idx, i], test_(
      }
}
rmse_test1
```

```
          ols_comb ols_macro rf_macro rf_comb elastic_macro_min
2016-2019    3.421     1.281    1.199   1.354             1.099
2020-2020   12.858     8.812   22.195  23.163             9.024
2016-2020    6.513     4.104    9.984  10.429             4.154
          elastic_macro_1se elastic_comb_min elastic_comb_1se xgb_macro
2016-2019             1.063            1.710            1.070     1.277
2020-2020            12.230           12.196           13.715    22.285
2016-2020             5.551            5.665            6.208    10.031
          xgb_comb avg_NGDP_macro avg_NGDP_comb ensemble_xgb_macro
2016-2019    1.209          1.127         1.323              1.566
2020-2020   23.369         14.282        14.844             21.825
2016-2020   10.507          6.466         6.743              9.860
```

```
rmse_year1
```

```
      ols_comb ols_macro rf_macro rf_comb elastic_macro_min elastic_macro_1se
Y2016    3.753    1.5171    0.953  0.6266            0.9485            0.6925
Y2017    2.542    0.9773    1.390  1.7234            0.9563            1.1911
Y2018    4.806    1.5127    1.507  1.1023            1.4420            1.3665
Y2019    1.776    1.0093    0.796  1.6609            0.9702            0.8668
Y2020   12.858    8.8117   22.195 23.1631            9.0242           12.2299
      elastic_comb_min elastic_comb_1se xgb_macro xgb_comb avg_NGDP_macro
Y2016            1.032           0.3759    0.5854   0.8738         0.8613
Y2017            1.433           1.1639    1.1880   1.2368         1.1151
Y2018            1.618           1.2500    1.8886   1.1536         1.5090
Y2019            2.441           1.2324    1.0968   1.4900         0.9041
Y2020           12.196          13.7148   22.2849  23.3695        14.2817
      avg_NGDP_comb ensemble_xgb_macro
Y2016        0.9468             1.8884
Y2017        1.3139             1.5533
Y2018        1.5371             1.7269
Y2019        1.4196             0.9213
Y2020       14.8440            21.8250
```

However, when looking at the testing data, we can see a completely different story. Below are the out-of-sample RMSE for each of the above models, alongside the ensemble models.

```
ols_macro_rmse <- 5.44
ols_comb_rmse <- 7.277
rf_macro_rmse <- 10.35
rf_comb_rmse <- 10.76
elastic_min_macro_rmse <- 5.54
elastic_min_comb_rmse <- 6.06
elastic_1se_macro_rmse <- 7.62
elastic_1se_comb_rmse <- 7.04
xgb_macro_rmse <- 10.29
xgb_comb_rmse <- 11.95
ensemble_avg_rmse_macro <- 7.32
ensemble_avg_rmse_comb <- 7.66
ensemble_xgb_rmse_macro <- 10.31
ensemble_xgb_rmse_comb <- 9.84

rmse_test <- rbind(ols_macro_rmse, ols_comb_rmse, rf_macro_rmse, rf_comb_rmse,
                   elastic_min_macro_rmse, elastic_min_comb_rmse,
                   elastic_1se_macro_rmse, elastic_1se_comb_rmse, xgb_macro_rmse,
                   xgb_comb_rmse, ensemble_avg_rmse_macro, ensemble_avg_rmse_comb,
                   ensemble_xgb_rmse_macro, ensemble_xgb_rmse_comb)

# Set the row names to be the names of each item
row.names(rmse_test) <- c("ols_macro", "ols_comb", "rf_macro", "rf_comb",
                          "elastic_min_macro", "elastic_min_comb",
                          "elastic_1se_macro", "elastic_1se_comb", "xgb_macro",
                          "xgb_comb", "ensemble_avg_rmse_macro", "ensemble_avg_rmse_comb",
                          "ensemble_xgb_rmse_macro", "ensemble_xgb_rmse_comb")
rmse_test <- data.frame(rmse_test)

# Print the resulting dataframe
html_df(rmse_test)
```

	rmse_test
ols_macro	5.440
ols_comb	7.277
rf_macro	10.350
rf_comb	10.760
elastic_min_macro	5.540
elastic_min_comb	6.060
elastic_1se_macro	7.620
elastic_1se_comb	7.040
xgb_macro	10.290
xgb_comb	11.950
ensemble_avg_rmse_macro	7.320
ensemble_avg_rmse_comb	7.660
ensemble_xgb_rmse_macro	10.310
ensemble_xgb_rmse_comb	9.840

In a complete contrast to the training RMSE, we can see that the macroeconomic model always outperform the combined model. In other words, adding accounting variables into the macroeconomic model did not improve its predicting power by all that much. In fact, it seems to lower it.

There are several reasons for this, most of which will be elaborated on in later sections of the report, but one possible main reason as to why our out-of-sample performance is worse overall and generally favors the macroeconomic model is that the macroeconomic variables - especially the GDP growth forecast from the Survey of Professional Forecasters - are more than efficient enough to account for all the necessary information to predict the actual GDP growth.

Furthermore, our testing data contains the time period of 2020, which was the peak of the Covid-19 pandemic and also the period in which the United States suffered a massive economic downturn, subsequently leading to a recession and negative GDP growth. Therefore, the models trained using the previous 25 years wouldn't be able to account for such an anomaly.

As such, adding accounting variables into the model seems to add more noise to the data rather than helping to account for that anomaly, compared to macroeconomic variables.

B. Sensitivity Analysis

From the above discussion on models' performance, it's clear that while in-sample performance is as expected, with the combined model performing better, out-of-sample performance presents a different scenario. Contrary to our hypothesis, the macroeconomic model often outperforms the combined model.

Therefore, does this imply that accounting information is essentially useless in improving the predictive power of the existing macroeconomic models?

Not quite.

Our literature review reveals that when both macroeconomic and accounting variables are employed to forecast current quarter GDP, such as using 2020Q1 data to predict 2020Q1 GDP growth, the addition of accounting variables doesn't improve out-of-sample accuracy. As mentioned, professional forecasts of GDP already captures the essential information. The same would apply for forecasting of next-quarter GDP growth due to the similar time frames.

However, when forecasting GDP growth further ahead - say four quarters, a different dynamic should occur instead, in which accounting variables add more predictive power instead of being 'useless'.

This leads us to a new hypothesis:

Hb: The inclusion of accounting information into a machine learning model will improve the predictive power when forecasting the 4-quarters-ahead's GDP Growth of the United States

To conduct this experimental analysis, we recreated our training and testing dataset, in which the GDP growth is 'led' by 4 quarters. In essence this means that we are using the current quarter's data to predict GDP growth which is 4 quarters ahead.

We then use the new training data and perform regression analysis on it by reusing the models in the previous data analysis, including OLS, Random Forest, Regularization, XGBoost, and ensembling.

```
load("RData/train_df.RData")
load("RData/test_df.RData")
# Create new training dataset
test_df <- test_df %>%
  mutate(YEAR = as.numeric(substr(YQ,1,4)))

b <- union(train_df, test_df)

new_rows_df <- data.frame(
  YQ = c("1989Q1", "1989Q2", "1989Q3", "1989Q4")
)

# Add the new rows to the dataframe
b <- bind_rows(b, new_rows_df) %>%
  arrange(YQ)


f <- b %>%
  left_join(gdp_forecast_mean) %>%
  left_join(net_income) %>%
  left_join(cpi) %>%
  left_join(energy_consumption) %>%
  left_join(fed_rates) %>%
  left_join(weekly_earnings) %>%
  left_join(unemployment) %>%
  left_join(household_consumption) %>%
  left_join(liquidity) %>%
  left_join(gross_saving) %>%
  left_join(import) %>%
  left_join(export) %>%
  left_join(funda_clean)
```

```
Joining with `by = join_by(YQ, YEAR)`
Joining with `by = join_by(YEAR)`
Joining with `by = join_by(YQ, YEAR)`
```

```
Joining with `by = join_by(YQ, YEAR)`
Joining with `by = join_by(YQ, YEAR)`
Joining with `by = join_by(YQ, YEAR)`
Joining with `by = join_by(YQ, YEAR)`
Joining with `by = join_by(YEAR)`
Joining with `by = join_by(YQ, YEAR)`
Joining with `by = join_by(YEAR)`
Joining with `by = join_by(YQ, YEAR)`
Joining with `by = join_by(YQ, YEAR)`
Joining with `by = join_by(YQ)`
```

```
f <- f %>%
  rename(GDP_Growth = "NGDP",
         Forecast_GDP = "NGDP2") %>%
  mutate(num_companies_standard = z_score_normalization(num_companies)) %>%
  arrange(YQ)

# 4 quarter
f4 <- f %>%
  arrange(YQ) %>%
  mutate(next_GDP_Growth = lead(GDP_Growth, default = NA, n = 4),
         next_YQ = lead(YQ, default = NA, n = 4)) %>%
  select(-GDP_Growth) %>%
  select(YQ,next_GDP_Growth, next_YQ, everything())

# Now the predicted/dependent value is next_GDP_Growth
# next_GDP_Growth ~ variables

# And then create a new test dataset starting from 2015Q4 to 2020Q3
# impute missing values in each column of the dataframe with the mean of that column
f4_training <- f4[1:104,] %>% rename(GDP_Growth = next_GDP_Growth) %>%
  mutate_if(is.numeric, ~ if_else(is.na(.), mean(., na.rm = TRUE), .))
f4_test <- f4[105:124,] %>% rename(GDP_Growth = next_GDP_Growth) %>%
  mutate_if(is.numeric, ~ if_else(is.na(.), mean(., na.rm = TRUE), .))


# Init a list for storing RMSE
rmse_ls2 <- list()
# Init a list to store testing predictions
pred_test_list2 <- as.data.frame(final_test_df[,'YQ'])
names(pred_test_list2) <- 'YQ'
```

```
# Init a dataframe to store predictions
pred_train_list2 <- as.data.frame(f4_training[,'YQ'])
pred_test_list2 <- as.data.frame(f4_test[,'YQ'])
names(pred_train_list2) <- 'YQ'
names(pred_test_list2) <- 'YQ'
```

a. OLS

Macro Model

```
# Set seed for reproducibility
set.seed(2024)

train.control <- trainControl(method = "repeatedcv",
                              number = 10, repeats = 3)

# Train the model
ols_macro2 <- train(macro_eq, data = f4_training, method = "lm",
               trControl = train.control)

pred_train_list2['ols_macro'] = predict(ols_macro2,f4_training[all.vars(macro_eq)[-1]])

rmse_ls2['ols_macro2'] = cal_rmse(predict(ols_macro2, f4_training[all.vars(macro_eq)[-1]]), f4_tr
```

Combined Model

```
# Set seed for reproducibility
set.seed(2024)

train.control <- trainControl(method = "repeatedcv",
                              number = 10, repeats = 3)

ols_comb2 <- train(comb_eq, data = f4_training, method = "lm",
               trControl = train.control)

pred_test_list2['ols_macro'] = predict(ols_macro2,f4_test[all.vars(macro_eq)[-1]])
```

```
rmse_ls2['ols_comb2'] = cal_rmse(predict(ols_comb2, f4_training[all.vars(comb_eq)[-1]]), f4_trair
```

Prediction

```
outcome_ols_comb <- as.data.frame(final_test_df[,1])
names(outcome_ols_comb)[1] <- "YQ"

# Combined model Prediction
prediction <- predict(ols_comb, newdata = f4_test)

outcome_ols_comb$NGDP <- prediction

write.csv(outcome_ols_comb, file = "./output/ols_comb2.csv", row.names = FALSE)

pred_test_list2$ols_comb <- prediction

# Macro variables prediction
outcome_ols_macro <- as.data.frame(final_test_df[,1])
names(outcome_ols_macro)[1] <- "YQ"
prediction <- predict(ols_macro, newdata = f4_test)

outcome_ols_macro$NGDP <- prediction

write.csv(outcome_ols_macro, file = "./output/ols_macro2.csv", row.names = FALSE)

pred_test_list2$ols_macro <- prediction
```

b. Random Forest

Macro Model

```
# Running this might take more than 3 minutes
# Tuning for best mtry for Macro model

set.seed(2024)
sqrt_num_predictors <- floor(sqrt(length(macro_vars) - 1)) #define max value of mtry
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
tunegrid <- expand.grid(.mtry=c(1:sqrt_num_predictors))
rf_gridsearch <- train(macro_eq, data = f4_training, method="rf", metric="RMSE", tuneGrid=tunegri

best_mtry_macro <- rf_gridsearch$bestTune$mtry

# Tuning for optimal number of trees
modellist <- list()
for (ntree in c(1000, 1500, 2000, 2500, 3000)) {
 set.seed(2024)
 fit <- train(macro_eq, data = f4_training, method="rf", metric="RMSE", tuneGrid=tunegrid, trCont
 key <- toString(ntree)
 modellist[[key]] <- fit
}
# compare results
results <- resamples(modellist)
s <- summary(results)$statistics$RMSE[,4]
best_ntree_macro <- as.numeric(names(s)[which.min(s)]) # ntree = 1000 gives the lowest RMSE
```

```
testSet <- f4_training
# With the optimal mtry and number of trees, we can now include it in the final models

# Random forest models generally do not need cross validation to prevent overfitting
# As that is done so internally when building each tree

rf_macro <- randomForest(macro_eq, data = f4_training, mtry = best_mtry_macro, ntree = best_ntree
print(rf_macro)
```

```
Call:
 randomForest(formula = macro_eq, data = f4_training, mtry = best_mtry_macro,      ntree =
best_ntree_macro)
               Type of random forest: regression
                     Number of trees: 1000
No. of variables tried at each split: 3

          Mean of squared residuals: 6.441
                    % Var explained: 12.66
```

```
testSet$pred_rf <- predict(rf_macro, newdata = f4_training)
pred_train_list2['rf_macro'] = testSet$pred_rf
rmse_ls2['rf_macro'] = cal_rmse(testSet$pred_rf,testSet$GDP_Growth)
```

Combined Model

```
# Running this might take more than 3 minutes
# Tuning for best mtry for Combined model

set.seed(2024)
sqrt_num_predictors <- floor(sqrt(length(comb_vars) - 1))
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")
tunegrid <- expand.grid(.mtry=c(1:sqrt_num_predictors))
rf_gridsearch <- train(comb_eq, data = f4_training, method="rf", metric="RMSE", tuneGrid=tunegric

best_mtry_comb <- rf_gridsearch$bestTune$mtry

# Tuning for optimal number of trees
modellist <- list()
for (ntree in c(1000, 1500, 2000, 2500, 3000)) {
 set.seed(2024)
 fit <- train(comb_eq, data = f4_training, method="rf", metric="RMSE", tuneGrid=tunegrid, trContr
 key <- toString(ntree)
 modellist[[key]] <- fit
}
# compare results
results <- resamples(modellist)
s <- summary(results)$statistics$RMSE[,4]
best_ntree_comb <- as.numeric(names(s)[which.min(s)]) # ntree = 1500 gives the lowest RMSE
```

```
rf_comb <- randomForest(comb_eq, data = f4_training, mtry = best_mtry_comb, ntree = best_ntree_co
print(rf_comb)
```

```
Call:
 randomForest(formula = comb_eq, data = f4_training, mtry = best_mtry_comb,      ntree =
best_ntree_comb)
               Type of random forest: regression
                     Number of trees: 1000
No. of variables tried at each split: 5

          Mean of squared residuals: 6.039
                    % Var explained: 18.11
```

```
testSet$pred_rf <- predict(rf_comb, newdata = f4_training)
pred_train_list2['rf_comb'] = testSet$pred_rf
rmse_ls2['rf_comb'] = cal_rmse(testSet$pred_rf,testSet$GDP_Growth)
```

Prediction

```
# Macro model
outcome_rf_macro <- as.data.frame(final_test_df[,1])
names(outcome_rf_macro)[1] <- "YQ"

prediction <- predict(rf_macro, newdata = f4_test)

outcome_rf_macro$NGDP <- prediction

write.csv(outcome_rf_macro, file = "./output/outcome_rf_macro2.csv", row.names = FALSE)

pred_test_list2$rf_macro <- prediction


# Combined model
outcome_rf_comb <- as.data.frame(final_test_df[,1])
names(outcome_rf_comb)[1] <- "YQ"


prediction <- predict(rf_comb, newdata = f4_test)

outcome_rf_comb$NGDP <- prediction

write.csv(outcome_rf_comb, file = "./output/outcome_rf_comb2.csv", row.names = FALSE)

pred_test_list2$rf_comb <- prediction
```

c. Regularization

Macro Model

```
# Performs a loop to find the best alpha for the macro model

x <- model.matrix(macro_eq, data = f4_training)[, -1]
y <- model.frame(macro_eq, data = f4_training)[ , "GDP_Growth"]
xp <- model.matrix(macro_eq, data = f4_training, na.action = 'na.pass')[ , -1]

set.seed(2024)
library(doParallel)
library(foreach)

list.of.fits <- list() # init a blank list

no_cores <- detectCores() -1 # reserve one core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('list.of.fits', 'x', 'y')) # import objects outside
clusterEvalQ(cl, expr= { # launch library to be used in parallel computing
  library(glmnet)
})
```

```
[[1]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[2]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[3]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[4]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[5]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[6]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[7]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"
```

```
list.of.fits <- foreach(i = 0:10) %dopar% { #foreach will return a list
  set.seed(2024)
  cv.glmnet(x = x, y = y, family = "gaussian", alpha = i/10,
            type.measure = "mse")
}



stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)
  names(list.of.fits)[i+1] <- c(fit.name)
}

results <- data.frame()
for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)

  ## Use each model to predict 'y' given the Testing dataset
  pred <- predict(list.of.fits[[fit.name]], xp, type = "response",
                  s = list.of.fits[[fit.name]]$lambda.min)

  # Calculate RMSE
  rmse <- sqrt(mean((pred - f4_training$GDP_Growth)^2))

  ## Store the results
  temp <- data.frame(alpha = i/10, rmse = rmse, fit.name = fit.name)
  results <- rbind(results, temp)
}
html_df(results)
```

alpha	rmse	fit.name
0.0	2.497	alpha0
0.1	2.194	alpha0.1
0.2	2.230	alpha0.2
0.3	2.217	alpha0.3
0.4	2.212	alpha0.4
0.5	2.210	alpha0.5
0.6	2.211	alpha0.6
0.7	2.206	alpha0.7
0.8	2.209	alpha0.8
0.9	2.206	alpha0.9
1.0	2.203	alpha1

```
# Find the row index corresponding to the minimum RMSE
min_rmse_row <- which.min(results$rmse)

# Extract the alpha corresponding to the minimum RMSE
best_alpha_macro <- results$alpha[min_rmse_row]

# Print the alpha corresponding to the minimum RMSE
print(best_alpha_macro)
```

```
[1] 0.1
```

```
x <- model.matrix(macro_eq, data = f4_training)[, -1]
y <- model.frame(macro_eq, data = f4_training)[ , "GDP_Growth"]

set.seed(2024)

fit_macro <- cv.glmnet(y = y,
                  x = x,
                  family = "gaussian",
                  alpha = best_alpha_macro,
                  type.measure = "mse")

print(fit_macro)
```

```
Call:  glmnet::cv.glmnet(x = x, y = y, family = "gaussian", alpha = best_alpha_macro,
type.measure = "mse")

Measure: Mean-Squared Error

    Lambda Index Measure   SE Nonzero
min   0.12    45    7.18 1.42      18
1se   7.00     1    7.53 2.34       0
```

```
pred_train_list2['elastic_macro_min'] = predict(fit_macro, x, s = "lambda.min")
pred_train_list2['elastic_macro_1se'] = predict(fit_macro, x, s = "lambda.1se")
rmse_ls2['elastic_macro_lambdamin'] = cal_rmse(predict(fit_macro, x, s = "lambda.min"),y)
rmse_ls2['elastic_macro_lambda1se'] = cal_rmse(predict(fit_macro, x, s = "lambda.1se"),y)
```

Combined Model

```
# Performs a loop to find the best alpha for the combined model
x <- model.matrix(comb_eq, data = f4_training)[, -1]
y <- model.frame(comb_eq, data = f4_training)[ , "GDP_Growth"]
xp <- model.matrix(comb_eq, data = f4_training, na.action = 'na.pass')[ , -1]
#xp <- model.matrix(macro_eq, data = f4_training, na.action = 'na.pass')[ , -1]

set.seed(2024)
library(doParallel)
library(foreach)

list.of.fits <- list() # init a blank list

no_cores <- detectCores() -1 # reserve one core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('list.of.fits', 'x', 'y')) # import objects outside
clusterEvalQ(cl, expr= { # launch library to be used in parallel computing
  library(glmnet)
})
```

```
[[1]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
```

```
[7] "datasets"  "methods"   "base"

[[2]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[3]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[4]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[5]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[6]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"

[[7]]
[1] "glmnet"    "Matrix"    "stats"     "graphics"  "grDevices" "utils"
[7] "datasets"  "methods"   "base"
```

```
list.of.fits <- foreach(i = 0:10) %dopar% { #foreach will return a list
  set.seed(2024)
  cv.glmnet(x = x, y = y, family = "gaussian", alpha = i/10,
            type.measure = "mse")
}

stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)
  names(list.of.fits)[i+1] <- c(fit.name)
}

results <- data.frame()
for (i in 0:10) {
  fit.name <- paste0("alpha", i/10)

  ## Use each model to predict 'y' given the Testing dataset
  pred <- predict(list.of.fits[[fit.name]], xp, type = "response",
                  s = list.of.fits[[fit.name]]$lambda.min)

  # Calculate RMSE
  rmse <- sqrt(mean((pred - f4_training$GDP_Growth)^2))

  ## Store the results
  temp <- data.frame(alpha = i/10, rmse = rmse, fit.name = fit.name)
  results <- rbind(results, temp)
}
html_df(results)
```

alpha	rmse	fit.name
0.0	2.001	alpha0
0.1	1.968	alpha0.1
0.2	1.939	alpha0.2
0.3	1.912	alpha0.3
0.4	1.910	alpha0.4
0.5	1.900	alpha0.5
0.6	1.907	alpha0.6
0.7	1.901	alpha0.7
0.8	1.910	alpha0.8
0.9	1.906	alpha0.9
1.0	1.903	alpha1

```
# Find the row index corresponding to the minimum RMSE
min_rmse_row <- which.min(results$rmse)

# Extract the alpha corresponding to the minimum RMSE
best_alpha_comb <- results$alpha[min_rmse_row]
```

```
# Print the alpha corresponding to the minimum RMSE
print(best_alpha_comb)
```

```
[1] 0.5
```

```
x <- model.matrix(macro_eq, data = f4_training)[, -1]
y <- model.frame(macro_eq, data = f4_training)[ , "GDP_Growth"]

set.seed(2024)

fit_macro <- cv.glmnet(y = y,
                       x = x,
                       family = "gaussian",
                       alpha = best_alpha_macro,
                       type.measure = "mse")

print(fit_macro)
```

```
Call:  glmnet::cv.glmnet(x = x, y = y, family = "gaussian", alpha = best_alpha_macro,
type.measure = "mse")

Measure: Mean-Squared Error

    Lambda Index Measure   SE Nonzero
min   0.12    45    7.18 1.42      18
1se   7.00     1    7.53 2.34       0
```

```
# Combined equation
x <- model.matrix(comb_eq, data = f4_training)[, -1]
y <- model.frame(comb_eq, data = f4_training)[ , "GDP_Growth"]

fit_comb <- cv.glmnet(y = y,  # Finish this (5 lines)
                      x = x,
                      family = "gaussian",
                      alpha = best_alpha_comb,
                      type.measure = "mse")

print(fit_comb)
```

```
Call:  glmnet::cv.glmnet(x = x, y = y, family = "gaussian", alpha = best_alpha_comb,
type.measure = "mse")

Measure: Mean-Squared Error

    Lambda Index Measure   SE Nonzero
min   0.31    20    6.59 2.09      14
1se   1.82     1    7.58 2.71       0
```

```
pred_train_list2['elastic_comb_min'] = predict(fit_comb, x, s = "lambda.min")
pred_train_list2['elastic_comb_1se'] = predict(fit_comb, x, s = "lambda.1se")
rmse_ls2['elastic_comb_lambdamin'] = cal_rmse(predict(fit_comb, x, s = "lambda.min"),y)
rmse_ls2['elastic_comb_lambda1se'] = cal_rmse(predict(fit_comb, x, s = "lambda.1se"),y)
```

Prediction

```
# Macro model Prediction using lambda.min
outcome_elastic_macro_min <- as.data.frame(final_test_df[,1])
names(outcome_elastic_macro_min)[1] <- "YQ"

f4_test$GDP_Growth <- 0

xp <- model.matrix(macro_eq, data = f4_test, na.action = 'na.pass')[ , -1]

prediction <- predict(fit_macro, xp, s = "lambda.min")

outcome_elastic_macro_min$NGDP <- prediction

write.csv(outcome_elastic_macro_min, file = "./output/elastic_macro_lambda_min2.csv", row.names =

pred_test_list2$elastic_macro_min <- prediction

# Macro model Prediction using lambda.1se
outcome_elastic_macro_1se <- as.data.frame(final_test_df[,1])
names(outcome_elastic_macro_1se)[1] <- "YQ"
xp <- model.matrix(macro_eq, data = f4_test, na.action = 'na.pass')[ , -1]

prediction <- predict(fit_macro, xp, s = "lambda.1se")

outcome_elastic_macro_1se$NGDP <- prediction
```

```
write.csv(outcome_elastic_macro_1se, file = "./output/elastic_macro_lambda_1se2.csv", row.names =
pred_test_list2$elastic_macro_1se <- prediction

# Combined model Prediction using lambda.min
outcome_elastic_comb_min <- as.data.frame(final_test_df[,1])
names(outcome_elastic_comb_min)[1] <- "YQ"

xp <- model.matrix(comb_eq, data = f4_test, na.action = 'na.pass')[ , -1]
prediction <- predict(fit_comb, xp, s = "lambda.min")

outcome_elastic_comb_min$NGDP <- prediction

write.csv(outcome_elastic_comb_min, file = "./output/elastic_comb_lambda_min2.csv", row.names = F
pred_test_list2$elastic_comb_min <- prediction

# Combined model Prediction using lambda.1se
outcome_elastic_comb_1se <- as.data.frame(final_test_df[,1])
names(outcome_elastic_comb_1se)[1] <- "YQ"

xp <- model.matrix(comb_eq, data = f4_test, na.action = 'na.pass')[ , -1]
prediction <- predict(fit_comb, xp, s = "lambda.1se")

outcome_elastic_comb_1se$NGDP <- prediction

write.csv(outcome_elastic_comb_1se, file = "./output/elastic_comb_lambda_1se2.csv", row.names = F

pred_test_list2$elastic_comb_1se <- prediction
```

d. XGBoost

Optimizing XGBoost

```
x = as.matrix(f4_training[ , all.vars(comb_eq)[-1] ])
y = as.matrix(f4_training[ , all.vars(comb_eq)[1] ])
#predictors <-all.vars(comb_eq)[-1]#
#outcomeName <- all.vars(comb_eq)[1]

scoring_function <- function(
  eta, gamma, max_depth, min_child_weight, subsample, nfold) {

  dtrain <- xgb.DMatrix(x, label = y, missing = NA)

  pars <- list(
    eta = eta,
    gamma = gamma,
    max_depth = max_depth,
    min_child_weight = min_child_weight,
    subsample = subsample,

    booster = "gbtree",
    objective = "reg:squarederror",
    eval_metric = "rmse",
    verbosity = 0
  )

  xgbcv <- xgb.cv(
    params = pars,
    data = dtrain,

    nfold = nfold,

    nrounds = 200,
    prediction = TRUE,
    showsd = TRUE,
    early_stopping_rounds = 10,
    #maximize = TRUE,
    maximize = FALSE,# the smaller the better
    stratified = TRUE
  )

  # required by the package, the output must be a list
  # with at least one element of "Score", the measure to optimize
  # Score must start with capital S
  # For this case, we also report the best num of iteration
  return(
    list(
      # Score = max(xgbcv$evaluation_log$test_rmse_mean),
      #Score = min(xgbcv$evaluation_log$test_rmse_mean),# remember to change max to min! ! !
      Score = -min(xgbcv$evaluation_log$test_rmse_mean),# remember to add minus sign to maximize!
      nrounds = xgbcv$best_iteration
    )
  )
```

```
}


# define the lower and upper bounds for each function (FUN) input
bounds <- list(
  eta = c(0.01, 1),
  gamma =c(0.01, 10),
  max_depth = c(2L, 10L), # L means integers
  min_child_weight = c(1, 10),
  subsample = c(0.25, 1),
  nfold = c(3L, 10L)
)

set.seed(2021)

#library(doParallel)
no_cores <- detectCores() - 1 # 1 core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('x','y')) # import objects outside
clusterEvalQ(cl,expr= { # launch library to be used in FUN
  library(xgboost)
  set.seed(2024)
})
```

```
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

[[5]]
NULL

[[6]]
NULL

[[7]]
NULL
```

```
time_withparallel <- system.time(
opt_obj <- bayesOpt(
  FUN = scoring_function,
  bounds = bounds,
  initPoints = 9,
  iters.n = 7,
  parallel = TRUE
))
```

```
stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

# again, have to use capital letters required by the package
# take a look at the output
opt_obj$scoreSummary
```

```
    Epoch Iteration     eta    gamma max_depth min_child_weight subsample nfold
    <num>     <int>   <num>    <num>     <num>            <num>     <num> <num>
 1:     0         1 0.60945  2.33211         3            9.469    0.5827     3
 2:     0         2 0.42381  7.42603         4            8.686    0.3688     7
 3:     0         3 0.13144  0.04782         3            1.034    0.7836     4
 4:     0         4 0.25313  5.47268         9            6.863    0.2639     9
 5:     0         5 0.73377  1.50139         5            2.034    0.6422     9
 6:     0         6 0.86640  3.93229         9            5.619    0.8910     6
 7:     0         7 0.89304  9.58271         6            7.418    0.4648     8
 8:     0         8 0.07007  5.59968         7            3.845    0.9390     5
 9:     0         9 0.55481  8.74252         8            4.604    0.6683     8
10:     1        10 0.01000 10.00000        10           10.000    0.2500     3
11:     2        11 0.14025  0.01000         2            6.012    1.0000    10
12:     3        12 0.12412 10.00000         2            4.732    0.2500    10
13:     4        13 0.22093 10.00000         2            5.703    1.0000     3
14:     5        14 0.01000  0.01000        10            3.408    0.8389     9
15:     6        15 0.16073  7.38043        10            3.556    1.0000     8
16:     7        16 0.13375  2.63769        10           10.000    1.0000     8
    gpUtility acqOptimum inBounds Elapsed  Score nrounds errorMessage
```

```
       <num>     <lgcl>   <lgcl>   <num>  <num>   <int>       <lgcl>
 1:        NA      FALSE     TRUE   0.038 -2.721       3           NA
 2:        NA      FALSE     TRUE   0.037 -2.594       8           NA
 3:        NA      FALSE     TRUE   0.072 -2.527      49           NA
 4:        NA      FALSE     TRUE   0.050 -2.420      13           NA
 5:        NA      FALSE     TRUE   0.106 -2.988      19           NA
 6:        NA      FALSE     TRUE   0.092 -2.717      25           NA
 7:        NA      FALSE     TRUE   0.034 -2.685       2           NA
 8:        NA      FALSE     TRUE   0.125 -2.411      54           NA
 9:        NA      FALSE     TRUE   0.040 -2.686       4           NA
10:    0.5231       TRUE     TRUE   0.041 -2.713     200           NA
11:    0.4327       TRUE     TRUE   0.068 -2.366      39           NA
12:    0.3339       TRUE     TRUE   0.042 -2.416      35           NA
13:    0.2917       TRUE     TRUE   0.009 -2.690       9           NA
14:    0.4210       TRUE     TRUE   0.555 -2.548     200           NA
15:    0.3395       TRUE     TRUE   0.091 -2.293      19           NA
16:    0.2347       TRUE     TRUE   0.118 -2.520      49           NA
```

```r
# the built-in function output the FUN input arguments only.
getBestPars(opt_obj)
```

```
$eta
[1] 0.1607

$gamma
[1] 7.38

$max_depth
[1] 10

$min_child_weight
[1] 3.556

$subsample
[1] 1

$nfold
[1] 8
```

```r
# take the optimal parameters for xgboost()
params <- list(eta = getBestPars(opt_obj)[[1]],
               gamma = getBestPars(opt_obj)[[2]],
               max_depth = getBestPars(opt_obj)[[3]],
               min_child_weight = getBestPars(opt_obj)[[4]],
               subsample = getBestPars(opt_obj)[[5]],
               nfold = getBestPars(opt_obj)[[6]],
               objective = "reg:squarederror")

# the numrounds which gives the max Score (auc)
numrounds <- opt_obj$scoreSummary$nrounds[
  which(opt_obj$scoreSummary$Score
        == max(opt_obj$scoreSummary$Score))] # still grab the max score becasue our score is nega
numrounds
```

```
[1] 19
```

Macro Model

```r
# macro
x = as.matrix(f4_training[ , all.vars(macro_eq)[-1] ])
y = as.matrix(f4_training[ , all.vars(macro_eq)[1] ])

xgb1 <- xgboost(params = params,
                    data = x,
                    label = y,
                    nrounds = numrounds,
                    eval_metric = "rmse")
```

```
[23:32:24] WARNING: src/learner.cc:767:
Parameters: { "nfold" } are not used.

[1] train-rmse:4.287556
[2] train-rmse:3.767420
[3] train-rmse:3.307701
[4] train-rmse:2.915797
[5] train-rmse:2.589380
[6] train-rmse:2.320115
[7] train-rmse:2.097107
[8] train-rmse:1.915275
[9] train-rmse:1.746485
[10]    train-rmse:1.619755
```

```
[11]    train-rmse:1.532887
[12]    train-rmse:1.443684
[13]    train-rmse:1.383133
[14]    train-rmse:1.333398
[15]    train-rmse:1.272330
[16]    train-rmse:1.233623
[17]    train-rmse:1.194947
[18]    train-rmse:1.141694
[19]    train-rmse:1.088674
```

```
pred_train_list2['xgb_macro'] = predict(xgb1,x)
rmse_ls2['xgb_macro'] = cal_rmse(predict(xgb1,x),y)
```

Combined Model

```
# comb
x = as.matrix(f4_training[ , all.vars(comb_eq)[-1] ])
y = as.matrix(f4_training[ , all.vars(comb_eq)[1] ])

xgb2 <- xgboost(params = params,
                      data = x,
                      label = y,
                      nrounds = numrounds,
                      eval_metric = "rmse")
```

```
[23:32:25] WARNING: src/learner.cc:767:
Parameters: { "nfold" } are not used.

[1] train-rmse:4.253975
[2] train-rmse:3.706773
[3] train-rmse:3.233780
[4] train-rmse:2.838523
[5] train-rmse:2.507580
[6] train-rmse:2.228375
[7] train-rmse:2.009037
[8] train-rmse:1.815651
[9] train-rmse:1.654676
[10]    train-rmse:1.518136
[11]    train-rmse:1.401601
[12]    train-rmse:1.292693
[13]    train-rmse:1.204659
[14]    train-rmse:1.148300
[15]    train-rmse:1.103609
[16]    train-rmse:1.069434
[17]    train-rmse:1.058948
[18]    train-rmse:1.051472
[19]    train-rmse:1.015069
```

```
pred_train_list2['xgb_comb'] = predict(xgb2,x)
rmse_ls2['xgb_comb'] = cal_rmse(predict(xgb2,x),y)
```

Prediction

```
# Macro
x <- model.matrix(macro_eq, data = f4_training)[, -1]
y <- model.frame(macro_eq, data = f4_training)[ , "GDP_Growth"]

xvals <- model.matrix(macro_eq, data = f4_test)[, -1]
yvals <- model.frame(macro_eq, data = f4_test)[ , "GDP_Growth"]

outcome_xgb_macro <- as.data.frame(final_test_df[,1])
names(outcome_xgb_macro)[1] <- "YQ"

prediction <- predict(xgb1, xvals)

outcome_xgb_macro$NGDP <- prediction

write.csv(outcome_xgb_macro, file = "./output/outcome_xgb_macro2.csv", row.names = FALSE)

pred_test_list2$xgb_macro <- prediction
```

```
# Macro
x <- model.matrix(comb_eq, data = f4_training)[, -1]
y <- model.frame(comb_eq, data = f4_training)[ , "GDP_Growth"]

xvals <- model.matrix(comb_eq, data = f4_test)[, -1]
yvals <- model.frame(comb_eq, data = f4_test)[ , "GDP_Growth"]

outcome_xgb_comb <- as.data.frame(final_test_df[,1])
names(outcome_xgb_comb)[1] <- "YQ"
```

```
prediction <- predict(xgb2, xvals)

outcome_xgb_comb$NGDP <- prediction

write.csv(outcome_xgb_comb, file = "./output/outcome_xgb_comb2.csv", row.names = FALSE)
pred_test_list2$xgb_comb <- prediction
```

e. Ensemble

```
pred_test_list2 <- pred_test_list2 %>%
  mutate(avg_NGDP_macro = (ols_macro + rf_macro + elastic_macro_min + elastic_macro_1se + xgb_mac
         avg_NGDP_comb = (ols_comb + rf_comb + elastic_comb_min + elastic_comb_1se + xgb_comb)/5)

# Macro predictions
outcome_ols_macro <- as.data.frame(final_test_df[,1])
names(outcome_ols_macro)[1] <- "YQ"
simple_average_macro <- cbind(outcome_ols_macro,pred_test_list2[c("avg_NGDP_macro")]) %>%
  rename(NGDP = "avg_NGDP_macro")
write.csv(simple_average_macro, file = "./output/simple_average_macro2.csv", row.names = FALSE)

# Combined predictions
outcome_ols_comb <- as.data.frame(final_test_df[,1])
names(outcome_ols_comb)[1] <- "YQ"
simple_average_comb <- cbind(outcome_ols_comb, pred_test_list2[c("avg_NGDP_comb")]) %>%
  rename(NGDP = "avg_NGDP_comb")
write.csv(simple_average_comb, file = "./output/simple_average_comb2.csv", row.names = FALSE)
```

```
x_test = pred_test_list2[c('ols_macro','rf_macro',"elastic_macro_min","elastic_macro_1se","xgb_ma
x = as.matrix(pred_train_list2[c('ols_macro','rf_macro',"elastic_macro_min","elastic_macro_1se","
y = as.matrix(f4_training[ , all.vars(comb_eq)[1] ])
#predictors <-all.vars(comb_eq)[-1]#
#outcomeName <- all.vars(comb_eq)[1]

scoring_function <- function(
  eta, gamma, max_depth, min_child_weight, subsample, nfold) {

  dtrain <- xgb.DMatrix(x, label = y, missing = NA)

  pars <- list(
    eta = eta,
    gamma = gamma,
    max_depth = max_depth,
    min_child_weight = min_child_weight,
    subsample = subsample,

    booster = "gbtree",
    objective = "reg:squarederror",
    eval_metric = "rmse",
    verbosity = 0
  )

  xgbcv <- xgb.cv(
    params = pars,
    data = dtrain,

    nfold = nfold,

    nrounds = 200,
    prediction = TRUE,
    showsd = TRUE,
    early_stopping_rounds = 10,
    #maximize = TRUE,
    maximize = FALSE,# the smaller the better
    stratified = TRUE
  )

  # required by the package, the output must be a list
  # with at least one element of "Score", the measure to optimize
  # Score must start with capital S
  # For this case, we also report the best num of iteration
  return(
    list(
      # Score = max(xgbcv$evaluation_log$test_rmse_mean),
      #Score = min(xgbcv$evaluation_log$test_rmse_mean),# remember to change max to min! ! !
      Score = -min(xgbcv$evaluation_log$test_rmse_mean),# remember to add minus sign to maximize!
      nrounds = xgbcv$best_iteration
    )
  )
}
```

```r
# define the lower and upper bounds for each function (FUN) input
bounds <- list(
  eta = c(0.01, 1),
  gamma =c(0.01, 10),
  max_depth = c(2L, 10L), # L means integers
  min_child_weight = c(1, 10),
  subsample = c(0.25, 1),
  nfold = c(3L, 10L)
)

set.seed(2021)

#library(doParallel)
no_cores <- detectCores() - 1 # 1 core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('x','y')) # import objects outside
clusterEvalQ(cl,expr= { # launch library to be used in FUN
  library(xgboost)
  set.seed(2024)
})
```

```
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

[[5]]
NULL

[[6]]
NULL

[[7]]
NULL
```

```r
time_withparallel <- system.time(
opt_obj <- bayesOpt(
  FUN = scoring_function,
  bounds = bounds,
  initPoints = 7,
  iters.n = 5,
  parallel = TRUE
))
```

```r
stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

# again, have to use capital letters required by the package
# take a look at the output
opt_obj$scoreSummary
```

```
    Epoch Iteration    eta   gamma max_depth min_child_weight subsample nfold
    <num>     <int>  <num>   <num>     <num>            <num>     <num> <num>
 1:     0         1 0.3938  0.5924         3            9.489    0.6108     9
 2:     0         2 0.2290  3.3835         9            7.534    0.9222     6
 3:     0         3 0.1439  2.8513         9            3.172    0.4921     8
 4:     0         4 0.7245  4.9805         6            7.219    0.3109     3
 5:     0         5 0.9152  8.9258         5            5.144    0.3588     5
 6:     0         6 0.7075  5.8343         3            1.824    0.8748     6
 7:     0         7 0.4549  7.7742         7            4.488    0.7575     7
 8:     1         8 0.0100  0.0100         2            1.000    0.4417     6
 9:     2         9 1.0000 10.0000         2            1.000    1.0000     5
10:     3        10 1.0000  0.0100         2            1.000    1.0000     7
11:     4        11 1.0000  0.0100        10            1.000    1.0000     8
12:     5        12 0.0100  3.3022         2            1.000    1.0000     8
    gpUtility acqOptimum inBounds Elapsed   Score nrounds errorMessage
        <num>     <lgcl>   <lgcl>   <num>   <num>   <int>       <lgcl>
 1:        NA      FALSE     TRUE   0.039 -1.2483      24           NA
 2:        NA      FALSE     TRUE   0.024 -1.1340      24           NA
 3:        NA      FALSE     TRUE   0.040 -1.0504      28           NA
 4:        NA      FALSE     TRUE   0.010 -1.8492       5           NA
 5:        NA      FALSE     TRUE   0.018 -1.3831       3           NA
 6:        NA      FALSE     TRUE   0.030 -0.9756      31           NA
```

```
 7:        NA      FALSE     TRUE   0.026 -1.1688      10           NA
 8:    0.4784       TRUE     TRUE   0.063 -1.3096     200           NA
 9:    0.4422       TRUE     TRUE   0.012 -1.1690      21           NA
10:    0.3682       TRUE     TRUE   0.012 -0.9365       7           NA
11:    0.3587       TRUE     TRUE   0.019 -0.9876       4           NA
12:    0.3317       TRUE     TRUE   0.086 -1.1843     200           NA
```

```
# the built-in function output the FUN input arguments only.
getBestPars(opt_obj)
```

```
$eta
[1] 1

$gamma
[1] 0.01

$max_depth
[1] 2

$min_child_weight
[1] 1

$subsample
[1] 1

$nfold
[1] 7
```

```
# take the optimal parameters for xgboost()
params <- list(eta = getBestPars(opt_obj)[[1]],
               gamma = getBestPars(opt_obj)[[2]],
               max_depth = getBestPars(opt_obj)[[3]],
               min_child_weight = getBestPars(opt_obj)[[4]],
               subsample = getBestPars(opt_obj)[[5]],
               nfold = getBestPars(opt_obj)[[6]],
               objective = "reg:squarederror")

# the numrounds which gives the max Score (auc)
numrounds <- opt_obj$scoreSummary$nrounds[
  which(opt_obj$scoreSummary$Score
        == max(opt_obj$scoreSummary$Score))] # still grab the max score becasue our score is nega
numrounds
```

```
[1] 7
```

```
x = as.matrix(pred_train_list2[c('ols_macro','rf_macro',"elastic_macro_min","elastic_macro_1se",'
y = as.matrix(f4_training[ , all.vars(comb_eq)[1] ])
xgb3 <- xgboost(params = params,
                    data = x,
                    label = y,
                    nrounds = numrounds,
                    eval_metric = "rmse")
```

```
[23:32:48] WARNING: src/learner.cc:767:
Parameters: { "nfold" } are not used.

[1] train-rmse:1.076324
[2] train-rmse:0.892501
[3] train-rmse:0.776172
[4] train-rmse:0.711868
[5] train-rmse:0.652924
[6] train-rmse:0.613237
[7] train-rmse:0.586419
```

```
pred_train_list2['ensemble_xgb_macro'] = predict(xgb3,x)
rmse_ls2['ensemble_xgb_macro'] = cal_rmse(predict(xgb3,x),y)


#x_test = as.matrix(pred_test_list[2:11])
x_test = as.matrix(pred_test_list2[colnames(x)])

pred_test_list2['ensemble_xgb_macro'] = predict(xgb3,x_test)

ensemble_xgb <- as.data.frame(final_test_df[,1])
names(ensemble_xgb)[1] <- "YQ"

ensemble_xgb <- ensemble_xgb %>%
  mutate(NGDP = predict(xgb3,x_test))
```

```r
write.csv(ensemble_xgb, file = "./output/ensemble_xgb_macro2.csv", row.names = FALSE)
```

```r
#x_test = pred_test_list[2:11]
x = as.matrix(pred_train_list2[-1])
y = as.matrix(f4_training[ , all.vars(comb_eq)[1] ])
#predictors <-all.vars(comb_eq)[-1]#
#outcomeName <- all.vars(comb_eq)[1]

scoring_function <- function(
  eta, gamma, max_depth, min_child_weight, subsample, nfold) {

  dtrain <- xgb.DMatrix(x, label = y, missing = NA)

  pars <- list(
    eta = eta,
    gamma = gamma,
    max_depth = max_depth,
    min_child_weight = min_child_weight,
    subsample = subsample,

    booster = "gbtree",
    objective = "reg:squarederror",
    eval_metric = "rmse",
    verbosity = 0
  )

  xgbcv <- xgb.cv(
    params = pars,
    data = dtrain,

    nfold = nfold,

    nrounds = 200,
    prediction = TRUE,
    showsd = TRUE,
    early_stopping_rounds = 10,
    #maximize = TRUE,
    maximize = FALSE,# the smaller the better
    stratified = TRUE
  )

  # required by the package, the output must be a list
  # with at least one element of "Score", the measure to optimize
  # Score must start with capital S
  # For this case, we also report the best num of iteration
  return(
    list(
      # Score = max(xgbcv$evaluation_log$test_rmse_mean),
      #Score = min(xgbcv$evaluation_log$test_rmse_mean),# remember to change max to min! ! !
      Score = -min(xgbcv$evaluation_log$test_rmse_mean),# remember to add minus sign to maximize!
      nrounds = xgbcv$best_iteration
    )
  )
}


# define the lower and upper bounds for each function (FUN) input
bounds <- list(
  eta = c(0.01, 1),
  gamma =c(0.01, 10),
  max_depth = c(2L, 10L), # L means integers
  min_child_weight = c(1, 10),
  subsample = c(0.25, 1),
  nfold = c(3L, 10L)
)

set.seed(2021)

#library(doParallel)
no_cores <- detectCores() - 1 # 1 core to avoid crash
cl <- makeCluster(no_cores) # make a cluster
registerDoParallel(cl) # register a parallel backend
clusterExport(cl, c('x','y')) # import objects outside
clusterEvalQ(cl,expr= { # launch library to be used in FUN
  library(xgboost)
  set.seed(2024)
})
```

```
[[1]]
NULL
```

```
[[2]]
NULL

[[3]]
NULL

[[4]]
NULL

[[5]]
NULL

[[6]]
NULL

[[7]]
NULL
```

```
time_withparallel <- system.time(
opt_obj <- bayesOpt(
  FUN = scoring_function,
  bounds = bounds,
  initPoints = 7,
  iters.n = 5,
  parallel = TRUE
))
```

```
stopCluster(cl) # stop the cluster
registerDoSEQ() # back to serial computing

# again, have to use capital letters required by the package
# take a look at the output
opt_obj$scoreSummary
```

```
    Epoch Iteration    eta   gamma max_depth min_child_weight subsample nfold
    <num>     <int>  <num>   <num>     <num>            <num>     <num> <num>
 1:     0         1 0.3938  0.5924         3            9.489    0.6108     9
 2:     0         2 0.2290  3.3835         9            7.534    0.9222     6
 3:     0         3 0.1439  2.8513         9            3.172    0.4921     8
 4:     0         4 0.7245  4.9805         6            7.219    0.3109     3
 5:     0         5 0.9152  8.9258         5            5.144    0.3588     5
 6:     0         6 0.7075  5.8343         3            1.824    0.8748     6
 7:     0         7 0.4549  7.7742         7            4.488    0.7575     7
 8:     1         8 1.0000  0.0100         2            1.000    1.0000     4
 9:     2         9 0.0100 10.0000        10            1.000    0.3042    10
10:     3        10 0.0100 10.0000         2            1.000    0.8826     4
11:     4        11 1.0000  0.0100         2            1.000    0.7262     6
12:     5        12 1.0000  0.0100         2            1.000    1.0000     7
    gpUtility acqOptimum inBounds Elapsed   Score nrounds errorMessage
        <num>     <lgcl>   <lgcl>   <num>   <num>   <int>       <lgcl>
 1:        NA      FALSE     TRUE   0.046 -1.1253      24           NA
 2:        NA      FALSE     TRUE   0.044 -1.0458      17           NA
 3:        NA      FALSE     TRUE   0.047 -0.9841      28           NA
 4:        NA      FALSE     TRUE   0.037 -1.6505      17           NA
 5:        NA      FALSE     TRUE   0.032 -1.3291       5           NA
 6:        NA      FALSE     TRUE   0.035 -0.9897       4           NA
 7:        NA      FALSE     TRUE   0.041 -1.1515       7           NA
 8:    0.5554       TRUE     TRUE   0.011 -0.9481      19           NA
 9:    0.5265       TRUE     TRUE   0.142 -1.3873     200           NA
10:    0.4990       TRUE     TRUE   0.053 -1.2299     200           NA
11:    0.5084       TRUE     TRUE   0.013 -0.9920      16           NA
12:    0.4533       TRUE     TRUE   0.017 -0.7761      20           NA
```

```
# the built-in function output the FUN input arguments only.
getBestPars(opt_obj)
```

```
$eta
[1] 1

$gamma
[1] 0.01

$max_depth
[1] 2

$min_child_weight
[1] 1

$subsample
[1] 1

$nfold
[1] 7
```

```
# take the optimal parameters for xgboost()
params <- list(eta = getBestPars(opt_obj)[[1]],
               gamma = getBestPars(opt_obj)[[2]],
               max_depth = getBestPars(opt_obj)[[3]],
               min_child_weight = getBestPars(opt_obj)[[4]],
               subsample = getBestPars(opt_obj)[[5]],
               nfold = getBestPars(opt_obj)[[6]],
               objective = "reg:squarederror")

# the numrounds which gives the max Score (auc)
numrounds <- opt_obj$scoreSummary$nrounds[
  which(opt_obj$scoreSummary$Score
        == max(opt_obj$scoreSummary$Score))] # still grab the max score becasue our score is nega
numrounds
```

```
[1] 20
```

```
x = as.matrix(pred_train_list2[2:10])
y = as.matrix(f4_training[ , all.vars(comb_eq)[1] ])
xgb4 <- xgboost(params = params,
                    data = x,
                    label = y,
                    nrounds = numrounds,
                    eval_metric = "rmse")
```

```
[23:33:08] WARNING: src/learner.cc:767:
Parameters: { "nfold" } are not used.

[1] train-rmse:1.045765
[2] train-rmse:0.829885
[3] train-rmse:0.703365
[4] train-rmse:0.635854
[5] train-rmse:0.571265
[6] train-rmse:0.512752
[7] train-rmse:0.444605
[8] train-rmse:0.389605
[9] train-rmse:0.365097
[10]    train-rmse:0.348961
[11]    train-rmse:0.330836
[12]    train-rmse:0.319937
[13]    train-rmse:0.297427
[14]    train-rmse:0.284113
[15]    train-rmse:0.270354
[16]    train-rmse:0.261082
[17]    train-rmse:0.246701
[18]    train-rmse:0.233333
[19]    train-rmse:0.221304
[20]    train-rmse:0.208372
```

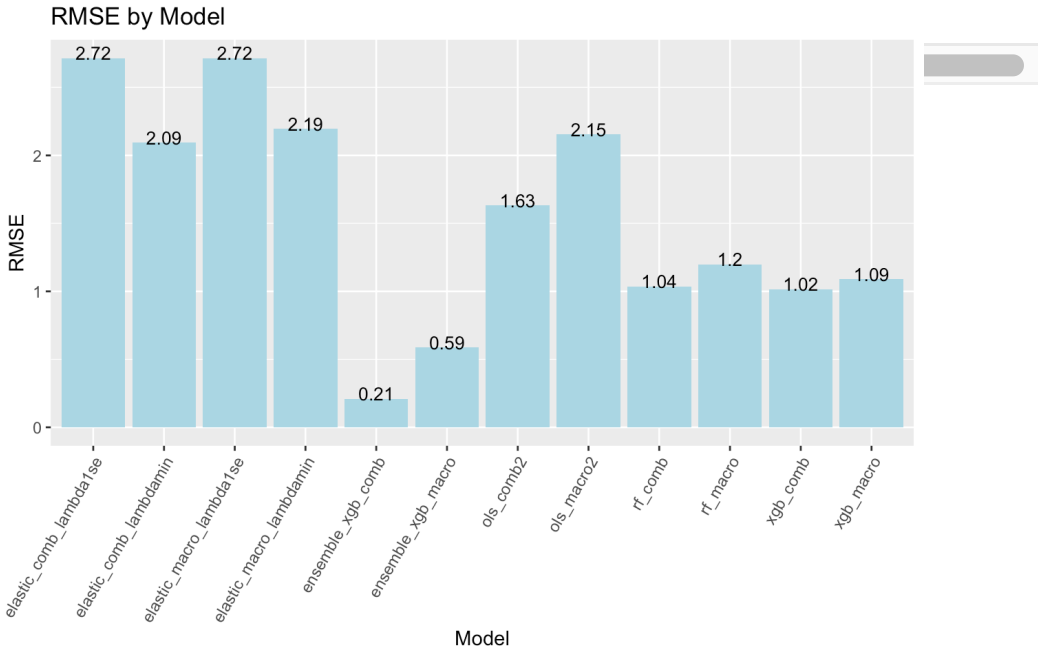
```
pred_train_list2['ensemble_xgb_comb'] = predict(xgb4,x)
rmse_ls2['ensemble_xgb_comb'] = cal_rmse(predict(xgb4,x),y)


#x_test = as.matrix(pred_test_list[2:11])
x_test = as.matrix(pred_test_list2[colnames(x)])

pred_test_list2['ensemble_xgb_comb'] = predict(xgb4,x_test)
ensemble_xgb <- as.data.frame(final_test_df[,1])
names(ensemble_xgb)[1] <- "YQ"

ensemble_xgb <- ensemble_xgb %>%
  mutate(NGDP = predict(xgb4,x_test))


write.csv(ensemble_xgb, file = "./output/ensemble_xgb_comb2.csv", row.names = FALSE)
```

Model Performance

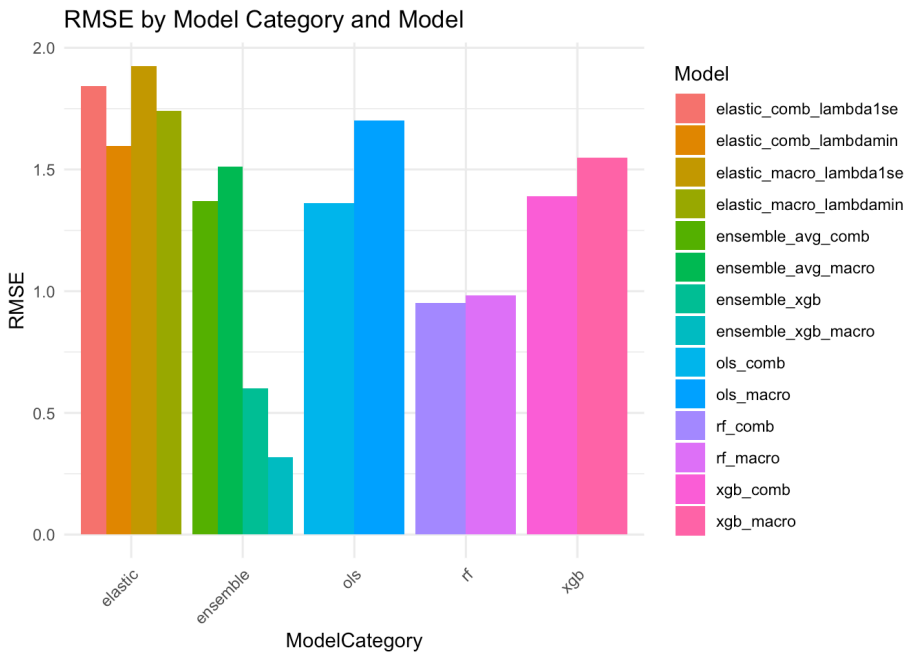
- Across macro data and combined data
    - On testing dataset, generally models based on combined data perform better than macro data.
- Within testing periods,
    - RMSE during COVID period is around ten times as much as non-COVID period.
    - The unexpected COVID shock accounts for most part of RMSE on testing dataset

```
rmse_df <- data.frame(Model = names(rmse_ls2), RMSE = unlist(rmse_ls2))

#options(repr.plot.width = 5, repr.plot.height =4)
```

```r
# Create the bar chart
ggplot(rmse_df, aes(x = Model, y = RMSE)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  geom_text(aes(label = round(RMSE, 2)), vjust = 0.1, size = 3.5, color = "black") + # Add labels
  labs(title = "RMSE by Model",
       x = "Model",
       y = "RMSE") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1),
        # aspect.ratio = 3 / 2  # Adjust the aspect ratio to make the plot longer
        )
```




```r
## Convert the rmse_ls list to a data frame
rmse_df <- data.frame(Model = names(rmse_ls), RMSE = unlist(rmse_ls))
# Define a function to extract the model category
extract_model_category <- function(model_name) {
  # Split the model name by underscore
  model_parts <- strsplit(model_name, "_")[[1]]
  # Extract the first part as the category
  model_category <- model_parts[1]
  return(model_category)
}

# Apply the function to create a new column
rmse_df$model_category <- sapply(names(rmse_ls), extract_model_category)

# Create the bar chart
ggplot(rmse_df, aes(x = model_category, y = RMSE, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "RMSE by Model Category and Model",
       x = "ModelCategory",
       y = "RMSE",
       fill = "Model") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
test_GDP = read_csv('Data/Federal_Reserve_GDP_Test_Sample.csv')
```

```
Rows: 20 Columns: 2
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
chr (1): YQ
dbl (1): NGDP

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.
```

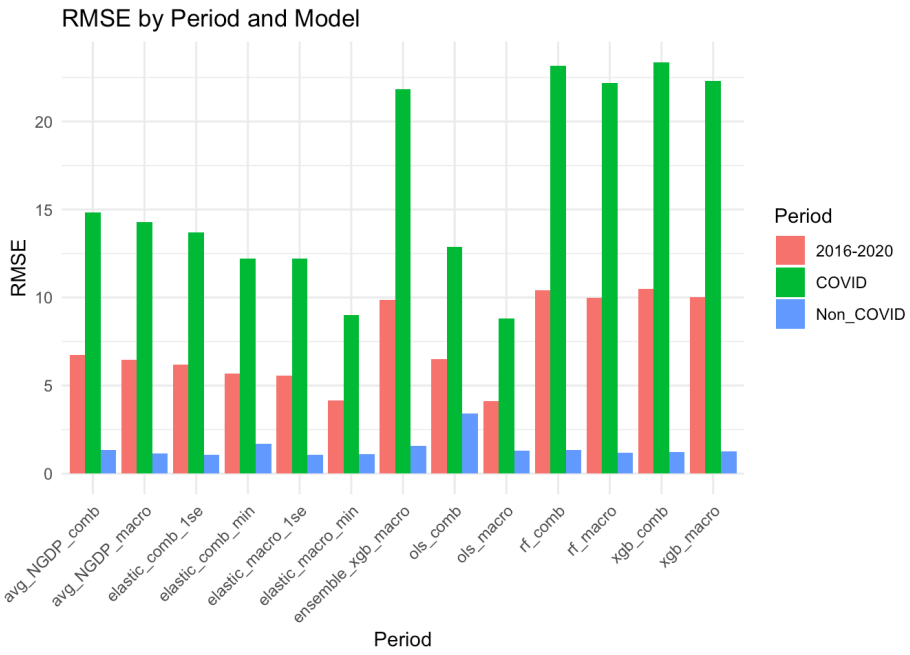
```
rmse_test2 = data.frame()
rmse_year2 = data.frame()

for (i in colnames(pred_test_list2)[-1]){
  #print(i)

  rmse_test2["Non_COVID",i] = cal_rmse(pred_test_list2[1:16, i], test_GDP$NGDP[1:16])
  rmse_test2["COVID",i] = cal_rmse(pred_test_list2[17:20,i], test_GDP$NGDP[17:20])
  rmse_test2["2016-2020",i] = cal_rmse(pred_test_list2[[i]], test_GDP$NGDP)
  #calculate RMSE by year from 2016 to 2020
  for (y in seq(1,5)){
        start_idx = 4*y-3
        end_idx = 4*y
        # rmse_year[2015+y,i] = cal_rmse(pred_test_list2[start_idx:end_idx,i], test_GDP$NGDP[star
    # Calculate RMSE for the current year
    rmse_year2[paste0("Y", 2015 + y), i] <- cal_rmse(pred_test_list2[start_idx:end_idx, i], test_
      }
}
html_df(rmse_test2)
```

	ols_macro	ols_comb	rf_macro	rf_comb	elastic_macro_min	elastic_macro_1se	elastic_comb_min	elas
Non_COVID	1.451	3.688	1.365	1.764	1.909	1.287	2.186	
COVID	24.354	24.664	24.952	24.566	24.849	24.649	24.487	
2016-2020	10.969	11.513	11.226	11.099	11.243	11.083	11.124	

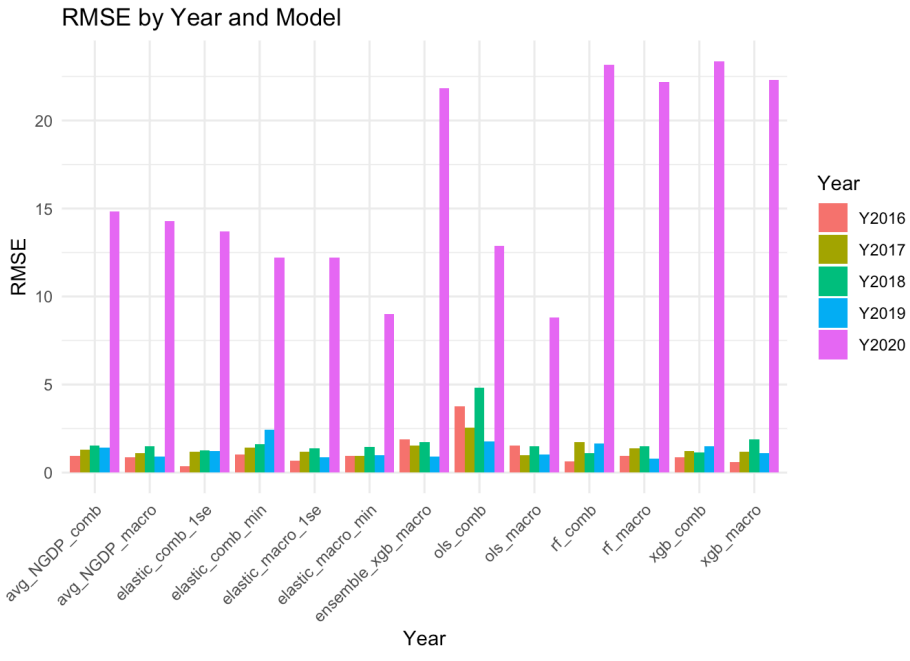
```
html_df(rmse_year2)
```

	ols_macro	ols_comb	rf_macro	rf_comb	elastic_macro_min	elastic_macro_1se	elastic_comb_min	elastic_c
Y2016	1.0164	1.232	0.7859	1.078	0.5856	1.4171	1.027	1.4
Y2017	1.8836	4.854	2.0858	2.185	3.2518	1.4111	2.690	1.4
Y2018	1.8971	2.777	1.2641	1.513	1.5386	1.3224	1.822	1.32
Y2019	0.4897	4.648	0.9444	2.055	1.1361	0.9363	2.738	0.93
Y2020	24.3541	24.664	24.9521	24.566	24.8487	24.6492	24.487	24.6

```
rmse_test2_long = pivot_longer(rmse_test1%>%mutate(Year=rownames(rmse_test2)), cols = -Year, name
rmse_year2_long = pivot_longer(rmse_year1%>%mutate(Year=rownames(rmse_year2)), cols = -Year, name


# Create the bar chart
ggplot(rmse_test2_long, aes(x = Model, y = RMSE, fill = Year)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "RMSE by Period and Model",
       x = "Period",
       y = "RMSE",
       fill = "Period") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# Create the bar chart
ggplot(rmse_year2_long, aes(x = Model, y = RMSE, fill = Year)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "RMSE by Year and Model",
       x = "Year",
       y = "RMSE",
       fill = "Year") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
ols_macro_rmse2 <- 9.11
ols_comb_rmse2 <- 7.86
rf_macro_rmse2 <- 11.63
rf_comb_rmse2 <- 11.53
```

```
elastic_min_macro_rmse2 <- 11.7
elastic_min_comb_rmse2 <- 11.56
elastic_1se_macro_rmse2 <- 11.48
elastic_1se_comb_rmse2 <- 11.48
xgb_macro_rmse2 <- 11.50
xgb_comb_rmse2 <- 11.76
ensemble_avg_rmse2_macro <- 11.58
ensemble_avg_rmse2_comb <- 11.59
ensemble_xgb_rmse2_macro <- 11.75
ensemble_xgb_rmse2_comb <- 11.78

rmse_test2 <- rbind(ols_macro_rmse2, ols_comb_rmse2, rf_macro_rmse2, rf_comb_rmse2,
                    elastic_min_macro_rmse2, elastic_min_comb_rmse2,
                    elastic_1se_macro_rmse2, elastic_1se_comb_rmse2, xgb_macro_rmse2,
                    xgb_comb_rmse2, ensemble_avg_rmse2_macro, ensemble_avg_rmse2_comb ,
                    ensemble_xgb_rmse2_macro, ensemble_xgb_rmse2_comb)

row.names(rmse_test2) <- c("ols_macro", "ols_comb", "rf_macro", "rf_comb",
                           "elastic_min_macro", "elastic_min_comb",
                           "elastic_1se_macro", "elastic_1se_comb", "xgb_macro",
                           "xgb_comb", "ensemble_avg_rmse_macro", "ensemble_avg_rmse_comb",
                           "ensemble_xgb_rmse_macro", "ensemble_xgb_rmse_comb")

rmse_test2 <- data.frame(rmse_test2)
html_df(rmse_test2)
```

	rmse_test2
ols_macro	9.11
ols_comb	7.86
rf_macro	11.63
rf_comb	11.53
elastic_min_macro	11.70
elastic_min_comb	11.56
elastic_1se_macro	11.48
elastic_1se_comb	11.48
xgb_macro	11.50
xgb_comb	11.76
ensemble_avg_rmse_macro	11.58
ensemble_avg_rmse_comb	11.59
ensemble_xgb_rmse_macro	11.75
ensemble_xgb_rmse_comb	11.78

Conclusion of Sensitivity Analysis:

From the results above, it can be observed that while the absolute RMSE is generally worse off compared to our initial models, the dynamics of the RMSE comparatively have shifted when we added the accounting variables. With this new dataset, when comparing the macroeconomic models to the combined models (Which includes the added accounting variables), it appears that the adding all those accounting variables does somewhat improve out-of-sample accuracy.

However, it should be noted that this sensitivity analysis should be taken with a grain of salt as it is nowhere near perfect, but it serves as a proof of concept for future research.

Furthermore, in line with our literature review, to an extent, we were able to prove that when predicting GDP growth that is 4 quarters ahead, adding accounting variables into existing models will improve the predicting power of those models

C. Limitations of Research

This research report is not without its flaws and can definitely be improved upon. The results shown in this report did not holistically show the intended results of our hypothesis and 'mental model' for our experiments due to several limitations. Firstly, despite the group's best efforts, there is a lack of industry experience especially in regards to the workings of the United States' GDP and economy, as well as inexperience regarding accounting - the latter of which severely impacted the feature selection and feature engineering of our accounting variables, leading to a rather lackluster collection of variables as a whole, only only 38 total independent variables compared to the literature review.

Furthermore, the lack of knowledge on machine learning models and hyperparameters is apparent as well, which led to all our models overfitting and unable to serve as practical predictive model. While some techniques were employed to make up for the low dimensionality of the training dataset (Only 104 observations), it was not enough to counteract the aforementioned overfitting issues.

V. Conclusion

When predicting future GDP growth of the United States of America, economists and researchers worldwide utilize a wide variety techniques and data in order to make the most accurate forecast possible. However, some researchers posit that the data used in these forecasting models do not accurately account for the financial information of the individual firms in the United States.

In this report, when adding those accounting variables into the pre-existing models (Which only include macroeconomic variables), the change in performance depends on the time horizon of the dependent variable.

Specifically, when forecasting current quarter's GDP growth of the US, accounting information does not add any predicting power to the pre-existing model for out-of-sample testing. In some cases, the accounting information only detracts from predicting power.

However, when forecasting 4-quarters-ahead's GDP growth of the US, accounting information does improve the predicting power of the forecasting model to a certain extent for out-of-sample testing.

In conclusion, while accounting information does not improve forecast accuracy of models when forecasting current quarter's GDP growth, it does marginally enhance the predictive power when forecasting 4-quarters-ahead's GDP growth.

References

Abrams, B. A., Clarke, M. Z., & Settlet, R. F. (1999). The impact of banking and fiscal policies on State-Level economic growth. *Southern Economic Journal*, *66*(2), 367. https://doi.org/10.2307/1061148

Agarwal, I., & Baron, M. (2018). Inflation and disintermediation. *Social Science Research Network*. https://doi.org/10.2139/ssrn.3399553

Baron, I. a. M., & Baron, I. a. M. (2023, October 2). Exploring the link between rising inflation and economic growth: The role of the banking sector. *World Bank Blogs*. https://blogs.worldbank.org/allaboutfinance/exploring-link-between-rising-inflation-and-economic-growth-role-banking-sector

Barro, R. J. (1991). Economic growth in a cross section of countries. *The Quarterly Journal of Economics*, *106*(2), 407–443. https://doi.org/10.2307/2937943

Chien, Y., & Arias, M. A. (2015). Lagging Long-Term wage growth. *Economic Synopses*, *2015*(14). https://doi.org/10.20955/es.2015.14

Cox, J. (2021, January 6). Cash in circulation is soaring, and that usually means good things for the economy. *CNBC*. https://www.cnbc.com/2021/01/05/cash-in-circulation-is-soaring-and-that-usually-means-good-things-for-the-economy.html

Cutler Cleveland. (2024, February 26). *What is the relationship between energy use and economic output?* Visualizing Energy. https://visualizingenergy.org/what-is-the-relationship-between-energy-use-and-economic-output/

Datar, S. M., Jain, A., Wang, C. C. Y., & Zhang, S. (2020). Is accounting useful for forecasting GDP growth? A Machine learning perspective. *Social Science Research Network*. https://doi.org/10.2139/ssrn.3827510

Davčev, L., Hourvouliades, N., & Komić, J. (2017). Impact of interest rate and inflation on GDP in Bulgaria, Romania and FYROM. *Journal of Balkan and Near Eastern Studies*, *20*(2), 131–147. https://doi.org/10.1080/19448953.2018.1379746

Decoupling of wages from productivity: what implications for public policies? (2018). *OECD Economic Outlook*, *2018*(2), 52. https://www.oecd.org/economy/decoupling-of-wages-from-productivity/

Di Giovanni, J., & Levchenko, A. A. (2012a). Country size, international trade, and aggregate fluctuations in granular economies. *Journal of Political Economy*, *120*(6), 1083–1132. https://doi.org/10.1086/669161

Di Giovanni, J., & Levchenko, A. A. (2012b). Country size, international trade, and aggregate fluctuations in granular economies. *Journal of Political Economy*, *120*(6), 1083–1132. https://doi.org/10.1086/669161

Di Giovanni, J., & Levchenko, A. A. (2012c). Country size, international trade, and aggregate fluctuations in granular economies. *Journal of Political Economy*, *120*(6), 1083–1132. https://doi.org/10.1086/669161

Dynan, K., & Sheiner, L. (2018). GDP as a Measure of Economic Well-being. *Hutchins Center Working Paper*. https://www.brookings.edu/articles/gdp-as-a-measure-of-economic-well-being/

Gatti, R., Lederman, D., Islam, A., Nguyen, H. H., Lotfi, R., & Mousa, M. E. (2023). Data transparency and GDP growth forecast errors. *World Bank Policy Research Working Papers*, *10406*. https://doi.org/10.1596/1813-9450-10406

GDP is growing, but workers' wages aren't. (2018, July 26). *Center for American Progress*. https://www.americanprogress.org/article/gdp-growing-workers-wages-arent/

Gross domestic product: an economy's all. (2019, June 15). IMF. https://www.imf.org/en/Publications/fandd/issues/Series/Back-to-Basics/gross-domestic-product-GDP

Heys, R., Martin, J., & Mkandawire, W. (2019). GDP and Welfare: A spectrum of opportunity. *RePEc: Research Papers in Economics*. https://ideas.repec.org/p/nsr/escoed/escoe-dp-2019-16.html

Independent Military Interests Analysis of the factors influencing GDP growth in China since 2001. (2016). *Era Finance*, *11*, DOI: CNKI: SUN: YNJR0.2016-11-006.

Jayaratne, J., & Strahan, P. E. (1996). The Finance-Growth Nexus: Evidence from Bank Branch Deregulation. *The Quarterly Journal of Economics*, *111*(3), 639–670. https://doi.org/10.2307/2946668

Jiajing, L., & Qiuyan, W. (2021). Empirical analysis of factors influencing GDP growth rate based on VAR Model Chinese Business Theory. *Chinese Business Theory*, *06*, 15–17. https://doi.org/10.19699/j.cnki.issn2096-0298.2023.06.015

Jiejie, W. (2010). Analysis of economic factors influencing GDP growth. *Modern Commerce and Industry*, *17*, 12–13. https://doi.org/10.19311/j.cnki.1672-3198.2013.17.006

Khan, M. (2010). Political settlements and the governance of Growth-Enhancing institutions. *Unpublished Manuscript*. https://eprints.soas.ac.uk/9968/1/Political_Settlements_internet.pdf

Konchitchki, Y., & Patatoukas, P. N. (2014). Accounting earnings and gross domestic product. *Journal of Accounting and Economics*, *57*(1), 76–88. https://doi.org/10.1016/j.jacceco.2013.10.001

Landefeld, J. S., Seskin, E. P., & Fraumeni, B. M. (2008). Taking the pulse of the economy: Measuring GDP. *Journal of Economic Perspectives*, *22*(2), 193–216. https://doi.org/10.1257/jep.22.2.193

Mirdala, R., Liotti, G., Canale, R. R., & Salvati, L. (2023). Inflation persistence and policy implications for central banks. In *Elsevier eBooks*. https://doi.org/10.1016/b978-0-44-313776-1.00102-1

Mulas, V. (2018). Which Countries are Better Prepared to Compete Globally in the Disruptive Technology Age?: A Rapid, Forward-Looking Analysis of Countries' Share of the Global Private Sector. In *World Bank, Washington, DC eBooks*. https://doi.org/10.1596/30615

Nocoń, A. (2023). Modern monetary policy - strategies, aims, and instruments. In *Elsevier eBooks*. https://doi.org/10.1016/b978-0-44-313776-1.00062-3

Osano, H. M., & Koine, P. W. (2016). Role of foreign direct investment on technology transfer and economic growth in Kenya: a case of the energy sector. *Journal of Innovation and Entrepreneurship*, *5*(1). https://doi.org/10.1186/s13731-016-0059-3

Pazzanese, C. (2021, July 15). *A key inflation index leaps. Getting worried?* Harvard Gazette. https://news.harvard.edu/gazette/story/2021/07/harvard-economist-examines-how-consumer-perceptions-can-affect-the-economy/

Sánchez, J. M., & Liborio, C. S. (2012). The relationships among changes in GDP, employment, and unemployment: This time, it's different. *Economic Synopses*, *2012*(13). https://doi.org/10.20955/es.2012.13

Sharma, N., Smeets, B., & Tryggestad, C. (2019, April 24). *The decoupling of GDP and energy growth: A CEO guide*. McKinsey & Company. https://www.mckinsey.com/industries/electric-power-and-natural-gas/our-insights/the-decoupling-of-gdp-and-energy-growth-a-ceo-guide

Shiferaw, Y. A. (2023). An Understanding of How GDP, Unemployment and Inflation Interact and Change across Time and Frequency. *Economies*, *11*(5), 131. https://doi.org/10.3390/economies11050131

Shughart, W. F., & Razzolini, L. (1997a). On the (relative) unimportance of a balanced budget. In *Springer eBooks* (pp. 215–233). https://doi.org/10.1007/978-94-011-5728-5_10

Shughart, W. F., & Razzolini, L. (1997b). On the (relative) unimportance of a balanced budget. *Public Choice*, *90*(1–4), 215–233. https://doi.org/10.1007/978-94-011-5728-5_10

Sumiyana, S. (2020). Different characteristics of the aggregate of accounting earnings between developed and developing countries: Evidence for predicting future GDP. *Journal of International Studies*, *13*(1), 58–80. https://doi.org/10.14254/2071-8330.2020/13-1/4

Syrquin, M. (2011). GDP as a measure of economic welfare. *Social Science Research Network*. https://doi.org/10.2139/ssrn.1808685

The Investopedia Team. (2023, October 20). *Currency in Circulation: Definition, how it works, and example*. Investopedia. https://www.investopedia.com/terms/c/currency-in-circulation.asp

Tuovila, A. (2022, June 27). *Real income, inflation, and the real wages formula*. Investopedia. https://www.investopedia.com/terms/r/realincome.asp

Weiwei, W. (2016). Analysis of the influencing factors of China's GDP growth based on multiple regression models. *China's Collective Economy*, *09*, DOI: CNKI: SUN: ZJTG.0.2016-09-031.

Xuebin, F., & Jianying, Z. (2019). Research on the Impact of population change quality on Economic Growth in Inner Mongolia Autonomous Region. *Introduction to Economic Research*, *33*, DOI: CNKI: SUN: JJYD.0.2019-33-029.

Yanfu, L. (2019). Research on the influencing factors of GDP growth in Jiangsu Province based on multiple linear regression models. *Special Economic Zone*, *04*, DOI: CNKI: SUN: TAJJ.0.2019-04-025.

Yi, Z. (2020). The source, current potential, and future growth range of China's economic growth. *Journal of Guangdong University of Finance and Economics*, *02*, 04–19.

Appendix

Appendix 1: Data Sourcing Table

Table 1: Data Sourcing Table

Dataset	Dataset_Name	Description	Source	Used?
Unicorn companies	unicorn_companies.xlsx	Information on global unicorn companies, ie. startups that are valued at $1 billion or more	CB Insights. Private Companies of Unicorn Club. Retrieved from https://www.cbinsights.com	No

Consumer Spending	consumer_spending.csv	Consumer spending by state	U.S. Bureau of Economic Analysis (BEA). Consumer Spending by State. Retrieved from https://www.bea.gov/	No
USA Population	usa_demographic_by_region.csv	Demographic information on US population by states	World Population Review. Population of USA. Retrieved from https://worldpopulationreview.com/states	No
Market Interest	fed_funda_rates.csv	Market interest rates in the US	https://fred.stlouisfed.org/series/FEDFUNDS#0	Yes
CPI	CPI.csv	Measures the average change in prices paid by consumers over a period of time for a basket of goods and services	https://fred.stlouisfed.org/series/CPIAUCSL#0	Yes
Unemployment Rate	unemployment_rate.csv	Unemployment rate in the US	https://fred.stlouisfed.org/series/UNRATE	Yes
Real Wage	median_weekly_real_earnings.csv	Median weekly real earnings in the US	https://fred.stlouisfed.org/series/LES1252881600Q	Yes
Currency in Circulation 1	M1.csv	Cash in circulation in the US	https://fred.stlouisfed.org/series/M1SL#0	No
Currency in Circulation 2	M2.csv	Cash and cash equivalents in the US	https://fred.stlouisfed.org/series/M2SL	Yes
Energy Consumption Residential	energy_consumption_residential_commercial_industrial.csv	Energy consumption for residential, industrial and commercial, industry	https://www.eia.gov/totalenergy/data/browser/index.php?tbl=T02.01B#/?f=A&start=1949&end=2022&charted=6-12-18	Yes
Energy Consumption Transport	energy_consumption_transportation.csv	Energy consumption for transport industry	https://www.eia.gov/totalenergy/data/browser/index.php?tbl=T02.01A#/?f=A&start=1949&end=2022&charted=6-12-18	Yes
Accounting Variables	merged_quarterly.csv	US firms' accounting information	WRDS - CRSP database	Yes
Import	USA_Imports.xls	USA import	Bureau of Economic Analysis, retrieved from https://www.bea.gov/data/intl-trade-investment/international-trade-goods-and-services	Yes
Export	USA_Export.xls	USA export	Bureau of Economic Analysis, retrieved from https://www.bea.gov/data/intl-trade-investment/international-trade-goods-and-services	Yes
Adjusted net national income per capita	adj_net_income_per_capita.xls	Net income per capita in the US	World Bank, retrieved from https://data.worldbank.org/indicator/NY.ADJ.NNTY.PC.CD?locations=US	Yes
Estimated Annual Sales	Estimated Annual Sales ofU.S. Retail Firms by Kind of Business 1992-2022.xlsx	Annual sales of US firms	United States Census Bureau	No
Gross Savings and Investments	gross_savings_investments.xls	Gross savings in the US	United States Census Bureau	Yes
Consumption Expenditure per capita	usa_household_consumption_per_capita.xls	Household consumption per capita in the US	World Bank, retrieved from https://data.worldbank.org/indicator/NE.CON.PRVT.PC.KD	Yes

Forecasted GDP	gdp_forecast_mean.xlsx	Analysts' forecast of GDP - through Survey of Professional Forecasters	https://www.philadelphiafed.org/surveys-and-data/real-time-data-research/survey-of-professional-forecasters	Yes

Appendix 2: Table of Variables

Table 2: Table of Variables

Variables	Mnemonic	Formula	Variable Explanation	Category	Main Type
Stockholders' Equity - Parent - Total Change	shareholder_equity_change	SEQQ / LAG(SEQQ) - 1	Changes in parent's equity	Capital Issuance	Accounting
Dividend Payout Ratio	dpr	dvpq/ibadjq	Expresses the proportion of a company's earnings distributed to shareholders as dividends.	Capital Issuance	Accounting
Long Term Debt/Invested Capital	debt_invcap	dlttq/icaptq	Ratio between long term debt and capital invested into the firm	Debt Ratio	Accounting
Asset Turnover	at_turn	saleq/((atq+lag(atq))/2)	Measures the efficiency with which a company utilizes its assets to generate revenue or sales	Efficiency	Accounting
Inventory Turnover	inv_turn	cogsq/invtq	Measures the efficiency with which a company manages its inventory	Efficiency	Accounting
Firm Size Change	asset_change	ATQ / LAG(ATQ) - 1	Changes in firm's size according to quarter	Firm Size	Accounting
Current Assets - Total Change	current_asset_change	ACTQ / LAG(ACTQ) - 1	Changes in firm's current assets	Firm Size	Accounting
Liabilities - Total Change	liabilities_change	LTQ / LAG(LTQ) - 1	Changes in firm's total liabilities	Liabilities	Accounting
Current Liabilities - Total Change	current_liabilities_change	LCTQ / LAG(LCTQ) - 1	Changes in firm's current liabilities	Liabilities	Accounting
Cash and Short-Term Investments Change	cash_ce_change	CHEQ / LAG(CHEQ) - 1	Changes in firm's cash and short term investments	Liquidity	Accounting
Profit Before Dep / Current Liabilities	profit_lct	coalesce(OIBDPQ,saleq-xoprq)/lctq	Operating Income before D&A as a fraction of	Liquidity	Accounting

			Current Liabilities		
Cash Ratio	cash_ratio	cheq/lctq	Measures a company's ability to cover its short-term liabilities with its cash and cash equivalents	Liquidity	Accounting
Quick Ratio	quick_ratio	coalesce(actq-invtq, cheq+rectq)/lctq	Also known as the acid-test ratio, is a financial metric that measures a company's ability to cover its short-term liabilities with its most liquid assets, excluding inventory	Liquidity	Accounting
Cash Flow	cf_to_asset	(ibcy + dpq)/atq	Cash flow standardized by total asset	Liquidity	Accounting
Share Price Growth	share_growth	PRCCQ / LAG(PRCCQ) - 1	Changes in share price of firm	Market	Accounting
Price / Sales	price_sales	PRCCQ/SALEQ	Closing price of firm against its sales for the quarter	Market	Accounting
Net profits after tax Change	net_profit_change	NIQ / LAG(NIQ) - 1	Net Income (Loss)	Profitability	Accounting
Net operating assets change	net_op_asset_change	log(((lag(ppentq+actq-lctq)+(ppentq+actq-lctq))/2))	Manual calculation	Profitability	
Depreciation Change	depr_change	DPQ / LAG(DPQ) - 1	Depreciation and Amortization change	Profitability	Accounting
Revenue change	revenue_change	REVTQ / LAG(REVTQ) - 1	Revenue - Total change	Profitability	Accounting
Return on Asset	roa	coalesce(oibdpq,saleq-xoprq,revtq-xoprq)/((atq+lag(atq))/2)	Measures a company's profitability relative to its total assets	Profitability	Accounting
Net Profit Margin	net_profit_margin	ibq/saleq	Measures the percentage of revenue that translates into profit after all expenses have been deducted	Profitability	Accounting
Operating Margin Before Depreciation	opm_bd	coalesce(oibdpq,saleq-xoprq,revtq-xoprq)/saleq	Operating Income Before Depreciation as a fraction of Sales	Profitability	Accounting
GDP Growth	GDP_Growth	GDP_Growth	Provided data for gdp growth, need	Dependent Variable	Dependent Variable

			to be predicted from 2015 - 2020		
Forecast GDP	Forecast_GDP	Forecast_GDP	Forecast of nominal GDP provided by the US BEA	Macroeconomic	Macroeconomic
Forecast GDP growth	forecast_growth	forecast_growth	Change in forecast of nominal GDP	Macroeconomic	Macroeconomic
Income per capita	income_per_capita	income_per_capita	USA income per capita	Macroeconomic	Macroeconomic
Income per capita growth	income_per_capita_growth	income_per_capita_growth	Change in income per capita	Macroeconomic	Macroeconomic
CPI	CPI	CPI	Consumer Price Index	Macroeconomic	Macroeconomic
Consumption of residential energy	residential	residential	Consumption of residential energy	Macroeconomic	Macroeconomic
Consumption of commercial energy	commercial	commercial	Consumption of commercial energy	Macroeconomic	Macroeconomic
Consumption of industrial energy	industrial	industrial	Consumption of industrial energy	Macroeconomic	Macroeconomic
Consumption of transport energy	transport	transport	Consumption of transport energy	Macroeconomic	Macroeconomic
Total energy consumption	total	total	Sum of energy consumption in the 4 sectors above	Macroeconomic	Macroeconomic
Change in consumption of residential energy	residential_energy_change	residential_energy_change	Change in consumption of residential energy	Macroeconomic	Macroeconomic
Change in consumption of commercial energy	commercial_energy_change	commercial_energy_change	Change in consumption of commercial energy	Macroeconomic	Macroeconomic
Change in consumption of industrial energy	industrial_energy_change	industrial_energy_change	Change in consumption of industrial energy	Macroeconomic	Macroeconomic
Change in consumptionof transport energy	transport_energy_change	transport_energy_change	Change in consumptionof transport energy	Macroeconomic	Macroeconomic
Change in total energy consumption	total_energy_change	total_energy_change	Change in total energy consumption	Macroeconomic	Macroeconomic
Federal interest rate	fed_rate	fed_rate	Federal interest rates	Macroeconomic	Macroeconomic
Change in federal interest rate	fed_rate_change	fed_rate_change	Change in federal interest rates	Macroeconomic	Macroeconomic
Weekly earnings	weekly_earnings	weekly_earnings	Weekly earnings	Macroeconomic	Macroeconomic

Weekly earnings change	earnings_change	earnings_change	Changes in weekly earnings	Macroeconomic	Macroeconomic
Unemployment rate	unemployment_rate	unemployment_rate	Unemployment rates	Macroeconomic	Macroeconomic
Change in unemployment rate	unemployment_rate_change	unemployment_rate_change	Change in unemployment rate	Macroeconomic	Macroeconomic
Household consumption	household_consumption	household_consumption	Expenditure made by US households	Macroeconomic	Macroeconomic
Change in household consumption	household_consumption_change	household_consumption_change	Change in US household expenditures	Macroeconomic	Macroeconomic
Average cash and cash equivalent circulation	avg_liquidity	avg_liquidity	Average cash and cash equivalent in circulation in the US	Macroeconomic	Macroeconomic
Change in average cash and cash equivalent circulation	avg_liquidity_change	avg_liquidity_change	Change in average cash and cash equivalent in circulation in the US	Macroeconomic	Macroeconomic
Gross savings	Gross_Saving	Gross_Saving	The difference between disposable income and consumption	Macroeconomic	Macroeconomic
Gross investments	Gross_Investment	Gross_Investment	Spending on productive physical capital such as machinery and construction of buildings, and on changes to inventories	Macroeconomic	Macroeconomic
Change in gross savings	Gross_Saving_change	Gross_Saving_Change	Changes in gross savings	Macroeconomic	Macroeconomic
Change in gross investments	Gross_Investment_change	Gross_Investment_Change	Changes in gross investment	Macroeconomic	Macroeconomic
Import	import	import	Goods and services that are purchased from the rest of the world	Macroeconomic	Macroeconomic
Changes in import amount	import_change	import_change	Changes in import amount	Macroeconomic	Macroeconomic
Export	export	export	Goods and services that are sold to the rest of the world	Macroeconomic	Macroeconomic
Changes in export amount	export_change	export_change	Changes in export amount	Macroeconomic	Macroeconomic
Total number of companies	num_companies	num_companies	Total number of companies	Macroeconomic	Macroeconomic
Variables	Mnemonic	Formula	Variable Explanation	Category	Type

Stockholders' Equity - Parent - Total Change	shareholder_equity_change	SEQQ / LAG(SEQQ) - 1	Changes in parent's equity	Capital Issuance	Accounting
Dividend Payout Ratio	dpr	dvpq/ibadjq	Expresses the proportion of a company's earnings distributed to shareholders as dividends.	Capital Issuance	Accounting
Long Term Debt/Invested Capital	debt_invcap	dlttq/icaptq	Ratio between long term debt and capital invested into the firm	Debt Ratio	Accounting
Asset Turnover	at_turn	saleq/((atq+lag(atq))/2)	Measures the efficiency with which a company utilizes its assets to generate revenue or sales	Efficiency	Accounting
Inventory Turnover	inv_turn	cogsq/invtq	Measures the efficiency with which a company manages its inventory	Efficiency	Accounting
Firm Size Change	asset_change	ATQ / LAG(ATQ) - 1	Changes in firm's size according to quarter	Firm Size	Accounting
Current Assets - Total Change	current_asset_change	ACTQ / LAG(ACTQ) - 1	Changes in firm's current assets	Firm Size	Accounting
Liabilities - Total Change	liabilities_change	LTQ / LAG(LTQ) - 1	Changes in firm's total liabilities	Liabilities	Accounting
Current Liabilities - Total Change	current_liabilities_change	LCTQ / LAG(LCTQ) - 1	Changes in firm's current liabilities	Liabilities	Accounting
Cash and Short-Term Investments Change	cash_ce_change	CHEQ / LAG(CHEQ) - 1	Changes in firm's cash and short term investments	Liquidity	Accounting
Profit Before Dep / Current Liabilities	profit_lct	coalesce(OIBDPQ,saleq-xoprq)/lctq	Operating Income before D&A as a fraction of Current Liabilities	Liquidity	Accounting
Cash Ratio	cash_ratio	cheq/lctq	Measures a company's ability to cover its short-term liabilities with its cash and cash equivalents	Liquidity	Accounting
Quick Ratio	quick_ratio	coalesce(actq-invtq, cheq+rectq)/lctq	Also known as the acid-test ratio, is a financial	Liquidity	Accounting

			metric that measures a company's ability to cover its short-term liabilities with its most liquid assets, excluding inventory		
Cash Flow	cf_to_asset	(ibcy + dpq)/atq	Cash flow standardized by total asset	Liquidity	Accounting

Footnotes

1. The predictive power mentioned will be measured by the RMSE of the models.↩
2. See Table 1↩
3. See Table 2↩