**FE 513: Final Exam**

Rishikesh Yadav

CWID: 20007668

1. **SQL**
2. Import given bank data into PostgreSQL database.

-- Creating table “bank” and importing the data from the csv file in it.

**QUERY:**

**DROP TABLE IF EXISTS bank ;**

Graphical user interface, text, application

Description automatically generated

CREATE TABLE IF NOT EXISTS bank (

id INTEGER NOT NULL,

date DATE NOT NULL,

asset INTEGER NOT NULL,

liability INTEGER NOT NULL,

idx INTEGER NOT NULL

) ;

Graphical user interface, text, application

Description automatically generated

COPY bank (id, date, asset, liability, idx) FROM 'C:\Users\psyad\Desktop\Stevens\Sem 4\FE 513\Final Exam\bank\_data-1.csv' DELIMITER ',' CSV HEADER ;

Graphical user interface, text, application

Description automatically generated

SELECT \* FROM bank ;

Table

Description automatically generated

1. Create a primary key for the import table.

-- On importing the data into “bank” table, since the idx column has unique values, we make that the primary key of the table.

**QUERY:**

ALTER TABLE bank ADD PRIMARY KEY (idx) ;

Table

Description automatically generated

1. Find the highest asset observation for each bank (i.e., There are 4 observations for each bank in a year. If there’re 100 unique banks in total, then the result contains 100 observations. Each observation may belong to different quarters.) Sort the resulting table according to asset value. Report the first 10 observations of output table.

-- In order to find the highest asset observation, we first group the bank data by id and sort it by id ascending and asset descending, and then take the first row for each id. We will achieve this by having 2 sub queries in our query – first subquery being responsible for grouping by id and ordering the id ascendingly, while the second subquery will be responsible for grouping by id and sorting the asset in descending order. Then we will join these 2 sub queries on id and their first row will give us the highest asset observations for each bank.

**QUERY:**

SELECT q1.id, q2.quarter, q2.date, q2.asset FROM

(SELECT id, EXTRACT(quarter FROM date) AS quarter, date, asset,

ROW\_NUMBER() OVER (Partition by id ORDER BY id ASC) AS b

from bank

order by id asc) q1

INNER JOIN

(SELECT id, extract(quarter from date) as quarter, date, asset,

ROW\_NUMBER() OVER (Partition by id ORDER BY asset DESC) AS b

FROM bank

ORDER BY id ASC) q2

ON q1.id = q2.id

AND q1.b = 1 AND q2.b = 1

ORDER BY q2.asset DESC

LIMIT 10;

Table

Description automatically generated

1. Show the query plan for question 1.3 using EXPLAIN tool.

-- EXPLAIN tool helps us understand the detailed step by step execution of any query

**QUERY:**

**EXPLAIN ANALYSE**

SELECT q1.id, q2.quarter, q2.date, q2.asset FROM

(SELECT id, EXTRACT(quarter FROM date) AS quarter, date, asset,

ROW\_NUMBER() OVER (Partition by id ORDER BY id ASC) AS b

from bank

order by id asc) q1

INNER JOIN

(SELECT id, extract(quarter from date) as quarter, date, asset,

ROW\_NUMBER() OVER (Partition by id ORDER BY asset DESC) AS b

FROM bank

ORDER BY id ASC) q2

ON q1.id = q2.id

AND q1.b = 1 AND q2.b = 1

ORDER BY q2.asset DESC

LIMIT 10 ;

Graphical user interface, text, application

Description automatically generated

A picture containing text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. Given the highest asset table from question 1.3, count how many observations are there for each quarter.

-- We simply achieve this by using the highest asset result table from question 1.3, grouping it by quarter and then taking a count of each quarter

**QUERY:**

SELECT obsv.quarter, count(\*) FROM

(SELECT q1.id, q2.quarter, q2.date, q2.asset FROM

(SELECT id, EXTRACT(quarter FROM date) AS quarter, date, asset,

ROW\_NUMBER() OVER (Partition by id ORDER BY id ASC) AS b

from bank

order by id asc) q1

INNER JOIN

(SELECT id, extract(quarter from date) as quarter, date, asset,

ROW\_NUMBER() OVER (Partition by id ORDER BY asset DESC) AS b

FROM bank

ORDER BY id ASC) q2

ON q1.id = q2.id

AND q1.b = 1 AND q2.b = 1

ORDER BY q2.asset DESC) obsv

GROUP BY obsv.quarter ;

**Table

Description automatically generated**

1. For the whole sample data, how many observations have asset value higher than 100,000 and liability value smaller than 100,000.

-- This can be achieved with the help of a simple count query by applying the given conditions, i.e., asset value higher than 100,000 and liability value smaller than 100,000.

**QUERY:**

SELECT COUNT(\*) FROM bank WHERE asset > 100000 AND liability < 100000 ;

Graphical user interface, text, application, chat or text message, email

Description automatically generated

1. Each observation was given an ’idx’ number. Find the average liability of observation with odd ‘idx’ number.

-- This can again be achieved simply by taking the average of every observation that has an odd idx value.

**QUERY:**

SELECT AVG( liability ) AS Odd\_Average\_Liability FROM bank WHERE idx % 2 = 1 ;

Graphical user interface, text, application, chat or text message

Description automatically generated

1. Find the average liability of observation with even ’idx’ number. What’s the difference between these two average numbers.

-- Like 1.7, we will achieve this by taking the average of every observation that has an even idx value. Then, we will subtract the smaller average from the greater average to find their difference.

**QUERY:**

SELECT AVG( liability ) AS Even\_Average\_Liability FROM bank WHERE idx % 2 = 0 ;

Graphical user interface, text, application, chat or text message

Description automatically generated

SELECT even.Even\_Average\_Liability - odd.Odd\_Average\_Liability AS Liability\_Difference

FROM

(SELECT AVG( liability) AS Even\_Average\_Liability FROM bank WHERE idx % 2 = 0) even,

(SELECT AVG( liability) AS Odd\_Average\_Liability FROM bank WHERE idx % 2 = 1) odd ;

Graphical user interface, text, application, chat or text message

Description automatically generated

1. For each bank find all records with increased asset. The record with increase asset means one record’s asset value is larger than the one of previous quarter. (For instance, a bank (id: 123) has asset 30,000 in 3/31/02, asset 20,000 in 6/30/02 and asset 25,000 in 9/30/02. Then the record with bank id (123), asset value (25,000) and date (9/30/02) is recorded. Because its asset value is larger than asset value in 6/30/02.) Report the first 10 observations of output table.

-- To achieve this, we will group the data by id and then order it ascendingly by quarter/ date. Then we will use lag function to find the asset value for previous quarter and compare it to the asset value for current quarter. From this we will only take those records that show an increase in the asset value from previous quarter for every bank.

**QUERY:**

SELECT q1.id, q1.quarter, q1.date, q1.asset

FROM

(SELECT id, EXTRACT(quarter FROM date) AS quarter, date, asset,

LAG(asset) OVER w AS prev\_asset

FROM bank

WINDOW w AS (PARTITION BY id ORDER BY EXTRACT(quarter FROM date) ASC)) q1

WHERE q1.asset > q1.prev\_asset

LIMIT 10 ;

Table

Description automatically generated

1. **User-defined Function in R**
2. Download daily stock data using a given stock ticker for a given time period (set starting

time, ending time and stock ticker as input variable).

1. Get the adjusted close price and consider this price data only for following tasks.
2. Perform a rolling window estimation on stock price vector to calculate the mean and standard deviation. For each time, you keep using a fixed size of data to calculate the mean and the standard deviation. The window size needs to be set as a input variable.
3. Store the statistical result of Q 2.3 into a dataframe. Plot this statistical dataframe using scatter plot. In the plot, x axis represents the index for each rolling window and y axis represents the statistical values. (Make sure you use different colors to differentiate between mean value and std value. Don’t forget to include an legend.)
4. Return the statistical dataframe.

**SCRIPT:**

library(quantmod)

library(roll)

library(ggplot2)

library(data.table)

stock\_data <- function(stock.ticker, start.date, end.date, rolling.size) {

*# 2.1*

*# Here we download the daily stock data and convert it to a dataframe for further*

*# processing*

stock.data <- as.data.frame(getSymbols

(stock.ticker, src = 'yahoo',

from = start.date,

to = end.date,

warnings = FALSE,

auto.assign = TRUE,env = NULL))

*# 2.2*

*# Here we get the adjusted close price*

adjusted.Close.Price <- stock.data[tail(names(stock.data), 1)]

*# 2.3*

*# Here we calculate the the mean and standard deviation by performing a rolling*

*# window estimation on stock price vector.*

mean <- rollapply

(adjusted.Close.Price, rolling.size,

by = 1, FUN = mean, by.column = FALSE)

std.Dev <- rollapply

(adjusted.Close.Price, rolling.size,

FUN = sd, fill=0,

align="r", by.column = FALSE)

std.Dev <- tail(std.Dev, -(rolling.size - 1))

*# 2.4*

*# Here we store the statistical result of Q 2.3 into a dataframe*

statistical <- do.call(rbind, Map(data.frame, A = mean, B = std.Dev))

colnames(statistical) <- c('Mean','Standard Deviation')

*# Here we transform the dataframe for our plot*

statistical.Transpose <- transpose(statistical)

colnames(statistical.Transpose) <- rownames(statistical)

rownames(statistical.Transpose) <- colnames(statistical)

*# Here we pot statistical dataframe using scatter plot*

statistical.plot <- ggplot() + geom\_point(data = stack(statistical.Transpose[1,]),

aes(x = ind, y = values, color = "Mean")) +

geom\_point(data = stack(statistical.Transpose[2,]),

aes(x = ind, y = values, color = "Standard Deviation"))

+ labs(x = "Index", y = "Statistical Values",

title = "Statistical Result") +

theme(axis.text.x = element\_text(angle = 90,

vjust = 0.5, hjust=1)) +

scale\_color\_manual(name = "Statistical Values",

values = c("Mean" = "yellow",

"Standard Deviation" = "green"))

print(statistical.plot)

*# 2.5*

*# Here we return the statistical dataframe result*

return(statistical)

}

1. Test your function with suitable parameters.(1 or 2-year data and a rolling window size of 20). If you got a huge dataframe for the output, do NOT print the whole sample. Showing a part of it is enough

**SCRIPT:**

*# 2.6*

*# Testing the above created function*

stock.ticker <- 'SNAP'

start.date <- as.Date('2019-12-31')

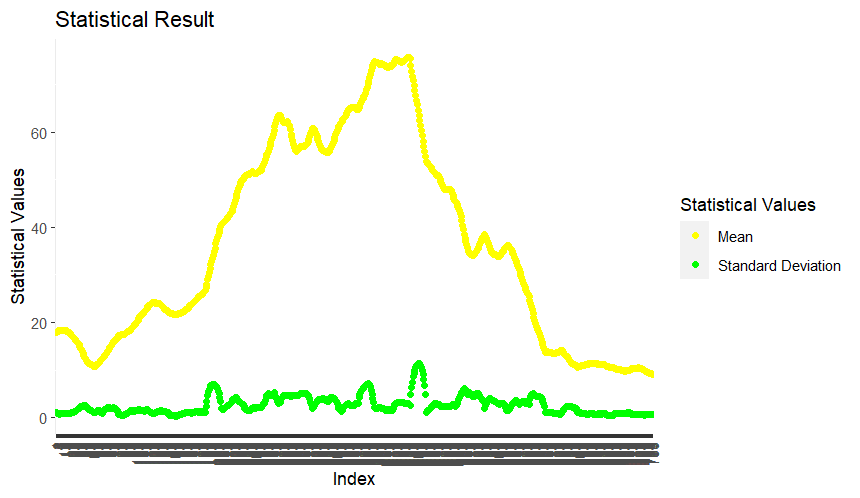
end.date <- as.Date('2022-12-31')

rolling.size <- 20

statistical.stock.dataframe <- stock\_data(stock.ticker, start.date, end.date, rolling.size)

print(statistical.stock.dataframe)

**RESULTS:**

****

**Graphical user interface, text, application, email

Description automatically generated**

**Text

Description automatically generated**

**Table

Description automatically generated**

1. **PostgreSQL API in R**
2. Make a connection to your local PostgreSQL database.

**SCRIPT:**

*# 3.1 - Here we make a connection to your local PostgreSQL database*

library(RPostgreSQL)

db\_name <- "FE\_513"

username <- "postgres"

driver <- dbDriver("PostgreSQL")

conn <-dbConnect(driver, dbname = db\_name, user = username, password = "root")

**RESULT:**

**Graphical user interface, text, application

Description automatically generated**

**Graphical user interface, text, application, email

Description automatically generated**

1. Query the PostgreSQL database via API to get the original bank data. (The bank data that you import to PostgreSQL database in Q 1.1) Store the data into a dataframe.

**SCRIPT:**

*# 3.2 - Here we query the PostgreSQL database via API to get the original bank data*

result <- dbGetQuery(conn, "SELECT \* FROM bank;")

**RESULT:**

**Graphical user interface, application, table

Description automatically generated with medium confidence**

1. Calculate asset growth rate for each quarter and each bank. (asset growth rate = (current quarter value - previous quarter value) / previous quarter value). The result start from second quarter, since we don’t have all necessary data for first quarter calculation. Store the calculation result in a data frame.

**SCRIPT:**

*# 3.3 - Here we calculate asset growth rate for each quarter and each bank with the*

*# given formula and store the result in a data frame.*

library(dplyr)

result <- result %>% group\_by(id) %>% arrange(id, date) %>% mutate(asset.growth.rate = (asset - lag(asset, n = 1))/ lag(asset, n = 1))

**RESULT:**

**Table

Description automatically generated**

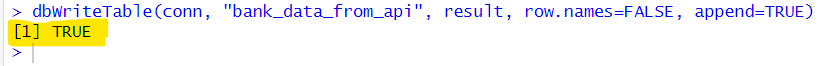
1. Export the dataframe of Q 3.3 to the PostgreSQL database via API

**SCRIPT:**

*#3.4 - Here we export the data frame of Q 3.3 to the PostgreSQL database via API*

dbWriteTable(conn, " bank\_data\_from\_api", result, row.names=FALSE, append=TRUE)

**RESULT:**



Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated