Final Exam FE 513-B

Lorenzo Ausiello

Problem 1

```
-- 1.1
-- Import given bank data into PostgreSQL database.
DROP TABLE IF EXISTS bank_data;
SET datestyle = 'MDY';
CREATE TABLE bank_data(
  id INTEGER,
  date DATE,
  asset INTEGER,
        liability INTEGER,
        idx INTEGER
);

COPY bank_data (id, date, asset, liability, idx)
FROM 'C:\Users\Public\bank_data-1.csv'
DELIMITER ','
CSV HEADER;

SELECT * FROM bank_data;
```

	id integer	date date	asset integer	liability integer	idx integer
1	23373	2002-09-30	95914	87304	1
2	23376	2002-12-31	95937	87453	2
3	23376	2002-03-31	83335	75939	3
4	23376	2002-06-30	84988	77125	4
5	23376	2002-09-30	90501	82248	5
6	234	2002-12-31	56866	49406	6
7	234	2002-03-31	55204	47914	7
8	234	2002-06-30	55180	47695	8
9	234	2002-09-30	56940	49249	9
10	23404	2002-12-31	78625	72580	10
11	23404	2002-03-31	72425	66709	11
12	23404	2002-06-30	73619	67798	12
13	23404	2002-09-30	73962	68002	13
14	23406	2002-12-31	2210000	1890000	14
15	23406	2002-03-31	1940000	1630000	15

```
-- 1.2
-- Create a primary key for the import table.
ALTER TABLE bank_data
ADD PRIMARY KEY (idx);


-- 1.3
-- Find the highest asset observation for each bank
-- Sort the resulting table according to asset value.
-- Report the first 10 observations of output table.
SELECT bd.id, bd.date, bank_max.max_asset AS asset, bd.liability
FROM bank_data bd
JOIN (
    SELECT id, MAX(asset) AS max_asset
    FROM bank_data
    GROUP BY id
```

```
) bank_max ON bd.id = bank_max.id AND bd.asset = bank_max.max_asset
ORDER BY bank_max.max_asset DESC
LIMIT 11;
```

	id integer	date date	asset integer	liability integer
1	628	2002-12-31	622000000	586000000
2	3510	2002-09-30	576000000	526000000
3	7213	2002-12-31	499000000	457000000
4	33869	2002-12-31	319000000	288000000
5	32633	2002-03-31	242000000	223000000
6	3618	2002-12-31	218000000	200000000
7	3511	2002-12-31	184000000	166000000
8	2558	2002-12-31	179000000	160000000
9	6548	2002-12-31	176000000	157000000
10	867	2002-12-31	115000000	106000000

```
-- 1.4
-- Show the query plan for question 1.3 using EXPLAIN tool
EXPLAIN
SELECT bd.id, bd.date, bank_max.max_asset AS asset, bd.liability
FROM bank_data bd
JOIN (
    SELECT id, MAX(asset) AS max_asset
    FROM bank_data
    GROUP BY id
) bank_max ON bd.id = bank_max.id AND bd.asset = bank_max.max_asset
ORDER BY bank_max.max_asset DESC
LIMIT 11;
```

	QUERY PLAN text
1	Limit (cost=1962.14..1962.14 rows=1 width=16)
2	-> Sort (cost=1962.14..1962.14 rows=1 width=16)
3	Sort Key: bank_max.max_asset DESC
4	-> Hash Join (cost=1144.39..1962.13 rows=1 width=16)
5	Hash Cond: ((bd.id = bank_max.id) AND (bd.asset = bank_max.max_asset))
6	-> Seq Scan on bank_data bd (cost=0.00..619.19 rows=37819 width=16)
7	-> Hash (cost=1000.35..1000.35 rows=9603 width=8)
8	-> Subquery Scan on bank_max (cost=808.29..1000.35 rows=9603 widt...
9	-> HashAggregate (cost=808.29..904.32 rows=9603 width=8)
10	Group Key: bank_data.id
11	-> Seq Scan on bank_data (cost=0.00..619.19 rows=37819 width...

-- 1.5

-- Given the highest asset table from question 1.3, count how many observations are there for

-- each quarter.

```
SELECT
    EXTRACT(QUARTER FROM bd.date) AS quarter,
    COUNT(*) AS observation_count
FROM bank_data bd
JOIN (
    SELECT id, MAX(asset) AS max_asset
    FROM bank_data
    GROUP BY id
) bank_max ON bd.id = bank_max.id AND bd.asset = bank_max.max_asset
GROUP BY quarter
ORDER BY quarter;
```

	quarter numeric	observation_count bigint
1	1	1203
2	2	763
3	3	1747
4	4	5947

-- 1.6

-- For the whole sample data, how many observations have asset value higher than 100,000 and

-- liability value smaller than 100,000.

```
SELECT COUNT(*) AS observation_count
FROM bank_data
WHERE asset > 100000 AND liability < 100000;
```

	observation_count bigint
1	1411

-- 1.7

-- Each observation was given an 'idx' number. Find the average liability of observation with

-- odd 'idx' number.

```
SELECT AVG(liability) AS average_liability_odd_idx
FROM bank_data
WHERE idx % 2 <> 0;
```

	average_liability_odd_idx numeric
1	778839.890163934426

-- 1.8

-- Find the average liability of observation with even 'idx' number. What's the difference between

-- these two average number.

```
SELECT AVG(liability) AS average_liability_even_idx
FROM bank_data
WHERE idx % 2 = 0;
```

	average_liability_even_idx numeric
1	787029.208260616638

```
SELECT
 (SELECT AVG(liability) FROM bank_data WHERE idx % 2 <> 0) -
 (SELECT AVG(liability) FROM bank_data WHERE idx % 2 = 0) AS difference;
```

	difference numeric
1	-8189.318096682212

--1.9

-- For each bank find all records with increased asset

-- Report the first 10 observation of output table.

```
SELECT b1.id, b1.date, b1.asset
FROM bank_data b1
JOIN bank_data b2 ON b1.id = b2.id
AND EXTRACT(QUARTER FROM b1.date) = EXTRACT(QUARTER FROM b2.date) + 1
AND b1.asset > b2.asset
ORDER BY b1.id, b1.date
LIMIT 10;
```

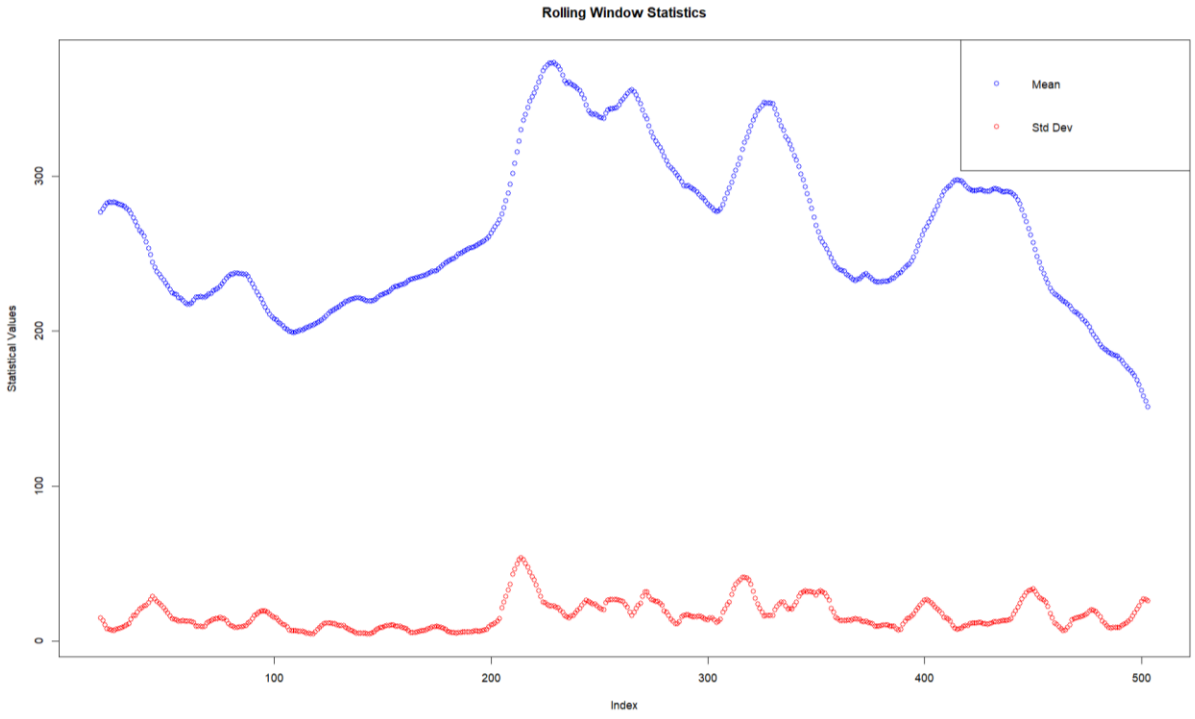
	id integer	date date	asset integer
1	9	2002-06-30	361953
2	9	2002-09-30	383246
3	14	2002-06-30	73600000
4	14	2002-12-31	79600000
5	28	2002-09-30	12474
6	35	2002-06-30	492046
7	35	2002-09-30	503401
8	39	2002-06-30	203754
9	39	2002-09-30	205211
10	39	2002-12-31	206140

Problem 2

```
library(quantmod)

get_stock_data <- function(ticker, start_time, end_time, window_size) {

  # Download daily stock data using a given stock ticker for a given time period

  stock_data <- getSymbols(ticker, from = start_time, to = end_time, auto.assign = FALSE)


  # Get the adjusted close price

  adj_close <- stock_data[,6]


  # Perform rolling window estimation for mean and standard deviation

  mean_estimates <- rollapply(adj_close, width = window_size, FUN = function(x) mean(x, na.rm = TRUE),
by.column = FALSE)

  std_estimates <- rollapply(adj_close, width = window_size, FUN = function(x) sd(x, na.rm = TRUE),
by.column = FALSE)


  # Store the statistical results of into a dataframe

  results <- data.frame(
```

```
    Index = 1:length(mean_estimates),
    Mean = mean_estimates,
    Std_Dev = std_estimates
  )
  results <- na.omit(results)

  # Plot this statistical dataframe using scatter plot
  plot(results$Index, results$Mean, col = "blue", xlab = "Index", ylab = "Statistical Values",
       ylim = c(min(c(results$Mean, results$Std_Dev)), max(c(results$Mean, results$Std_Dev))), main =
"Rolling Window Statistics")
  points(results$Index, results$Std_Dev, col = "red")
  legend("topright", legend = c("Mean", "Std Dev"), col = c("blue", "red"), pch = 1)

  # Return the statistical dataframe
  return(results)
}

# Test your function with suitable parameters
start_date <- '2021-01-01'
end_date <- '2023-01-01'
ticker <- "TSLA"
rolling_window <- 20

stock_stats <- get_stock_data(ticker, start_date, end_date, rolling_window)
head(stock_stats)
```

```
> head(stock_stats)
            Index     Mean   Std_Dev
2021-02-01    20 276.4428 14.894892
2021-02-02    21 278.8265 12.998415
2021-02-03    22 280.8195 10.325927
2021-02-04    23 282.3863  7.787053
2021-02-05    24 282.9895  7.398818
2021-02-08    25 282.7128  7.088349
>
```



Problem 3

```
library("RPostgres")

# Make a connection to your local PostgreSQL database

con <- dbConnect(RPostgres::Postgres(),

                 dbname = "postgres",

                 host = "127.0.0.1",

                 port = 5432,

                 user = "postgres",
```

password = 'Culturismo99.')

Query the PostgreSQL database via API to get the original bank data.

Store the data into a dataframe.

query <- "SELECT * FROM bank_data;"

bank_data <- dbGetQuery(con, query)

head(bank_data)

```
> head(bank_data)
     id       date asset liability idx
1 23373 2002-09-30 95914     87304   1
2 23376 2002-12-31 95937     87453   2
3 23376 2002-03-31 83335     75939   3
4 23376 2002-06-30 84988     77125   4
5 23376 2002-09-30 90501     82248   5
6   234 2002-12-31 56866     49406   6
>
```

3.3

Calculate asset growth rate for each quarter and each bank.

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.

library(dplyr)

bank_data <- bank_data %>%

arrange(id, date) %>%

group_by(id) %>%

mutate(asset_growth_rate = ifelse(row_number() == 1, NA,

(asset - lag(asset)) / lag(asset))) %>%

ungroup()

Export the dataframe to the PostgreSQL database via API

dbWriteTable(con, "asset_growth_rates", bank_data, row.names = FALSE, overwrite = TRUE)

dbDisconnect(con)

-- 3.4 (SQL)

SELECT * FROM asset_growth_rates;

	id integer	date date	asset integer	liability integer	idx integer	asset_growth_rate double precision
1	9	2002-03-31	348727	321479	20912	[null]
2	9	2002-06-30	361953	332900	20913	0.03792651558382345
3	9	2002-09-30	383246	352456	20914	0.058828079888825345
4	9	2002-12-31	371812	340365	20911	-0.029834623192414273
5	14	2002-03-31	68600000	64300000	27334	[null]
6	14	2002-06-30	73600000	69200000	27335	0.0728862973760933
7	14	2002-09-30	72800000	68200000	27336	-0.010869565217391304
8	14	2002-12-31	79600000	74500000	27333	0.09340659340659341
9	28	2002-03-31	14340	7948	3937	[null]
10	28	2002-06-30	12049	5354	3938	-0.1597629009762901
11	28	2002-09-30	12474	5543	3939	0.035272636733338865
12	35	2002-03-31	471056	438541	12623	[null]
13	35	2002-06-30	492046	457116	12624	0.04455945789884855
14	35	2002-09-30	503401	467080	12625	0.023077110676644055