# WEN

September 21, 2025

# Contents

**1 Preparations** **3**

**2 Data Analysis** **3**
2.1 (1) Daily Returns . . . . . 5
2.2 (2) Overnight Returns . . . . . 5
2.3 (3) Volume Change . . . . . 5
2.4 (4) Money Flow Volume Indicator (MFV) . . . . . 5

**3 More features, metrics and dates** **6**
3.1 (5) Month & (6) Year . . . . . 6
3.2 (7) Total Trading Volume in June 2023 . . . . . 6
3.3 (8) Mean Daily Return . . . . . 6
3.4 (9) Date with the largest High Price . . . . . 6
3.5 (10) Date with the largest Daily Return . . . . . 7

# List of Figures

# List of Tables

# 1 Preparations

Before messing around with the stock data, the environment should install and load the dplyr and lubridate packages to perform easier data analysis.

[ ]:
```
# installation
install.packages("dplyr")
install.packages("lubridate")
install.packages("knitr")
install.packages("kableExtra")
```

After installing them, the packages are loaded in.

[41]:
```
#| warning: false
library(dplyr)
library(lubridate)
library(knitr)
library(kableExtra)
```

```
Attaching package: ‘kableExtra’


The following object is masked from ‘package:dplyr’:

    group_rows
```

# 2 Data Analysis

Now let's start actually processing and transforming the stock data. First of all, the data must be *read* into existence, so that it becomes a workable dataframe. Columns irrelevant to this exercise are removed for illustration purposes. But to ensure correct formatting, the `datadate` variable is adjusted to the standard date format.

[42]:
```
# load csv file as stock_data but clean the date format
data = read.csv("compustat_food_bev.csv")
stock_data = data[, -c(1, 2, 5)]
stock_data$datadate = as.Date(stock_data$datadate, format = "%d/%m/%Y")
```

With the stock data in place, only the relevant data set gets filtered out, here: Wendy's (ticker: WEN)

[43]:
```
# construct new data table with stock data exclusively tied to Wendy's
stock_data_wen = filter(stock_data, tic=="WEN")
```

Inbefore tackling the tasks, some new columns must be created. These are going to be loaded with metrics such as the daily returns, overnight returns, volume change and MFV of our stock data.

[44]:
```
# create new empty columns
stock_data_wen$retd = 0
stock_data_wen$retov = 0
stock_data_wen$volch = 0
stock_data_wen$mfv = 0
```

Next, we can actually perform the calculations for these metrics.

### 2.1 (1) Daily Returns

[ ]:
```
# calculate daily returns as: (closing price_t/closing price_t-1)-1
head(stock_data_wen <- mutate(stock_data_wen, retd = round((prccd/
 ↪lag(prccd))-1, 2)))
```

### 2.2 (2) Overnight Returns

[ ]:
```
# calculate overnight returns as: (openning price_t/closing price_t-1)-1
head(stock_data_wen <- mutate(stock_data_wen, retov = round((prcod/
 ↪lag(prccd))-1, 2)))
```

### 2.3 (3) Volume Change

[ ]:
```
# calculate volume change as: (volume_t/volume_t-1)-1
head(stock_data_wen <- mutate(stock_data_wen, volch = round((cshtrd/
 ↪lag(cshtrd))-1, 2)))
```

### 2.4 (4) Money Flow Volume Indicator (MFV)

[ ]:
```
# calculate mfv
head(stock_data_wen <- mutate(stock_data_wen,
    mfv = round((
        ((prccd-prcld)-(prchd-prcld))/(prchd-prcld)
        ), 2)
))
```

# 3 More features, metrics and dates

After completing the initial analysis, the next part will cover various kinds of data manipulation and appending new variables to the dataframe.

## 3.1 (5) Month & (6) Year

Once more, first a few more columns are added to hold the data, after which the `month()` and `year()` functions help identifying the respective data points.

[ ]:
```
# Add column that indicate the month and year
stock_data_wen$mon = 0
stock_data_wen$yr = 0
head(stock_data_wen <- mutate(stock_data_wen,
    mon = month(datadate),
    yr = year(datadate)
))
```

## 3.2 (7) Total Trading Volume in June 2023

By employing a method innate to R, the `sum()` function, together with a condition tied to the `mon` and `yr` variables lends a quick result to the task.

[50]:
```
# Calculate the total trading volume, in June 2023.
tv0623 = sum(stock_data_wen$cshtrd[
    stock_data_wen$mon == 6 & stock_data_wen$yr==2023
    ])
tv0623
```

54557454

## 3.3 (8) Mean Daily Return

Another method of R, the `mean()` function, produces an even quicker result to this task.

[51]:
```
# Calculate the mean daily return, over the entire period.
return_daily_mean = mean(stock_data_wen$retd, na.rm = TRUE)*100
print(paste0(round(return_daily_mean, 2), "%"))
```

```
[1] "0.02%"
```

## 3.4 (9) Date with the largest High Price

The desired date relates to the maximum value across the `prchd` variable, thereby hinting at applying the `which.max()` function for indexing. Note that applying just the `max()` function does not give the desired result, since it returns the actual price instead of the row index.

[52]:
```
# Calculate the date that saw the largest positive high price.
date_highest_high = stock_data_wen$datadate[
    which.max(stock_data_wen$prchd)
```

```
    ]
date_highest_high
```

2021-06-08

But it's still interesting to see the price.

[53]:
```
# Said price is how much?
price_highest = stock_data_wen$prchd[which.max(stock_data_wen$prchd)]
price_highest
```

29.46

### 3.5 (10) Date with the largest Daily Return

This desired date relates to the maximum value across the `retd` variable, again hinting at the `which.max()` function.

[54]:
```
# Calculate the date that saw the largest positive daily return.
date_highest_daily_return = stock_data_wen$datadate[
    which.max(stock_data_wen$retd)
    ]
date_highest_daily_return
```

2021-06-08

The return is..

[55]:
```
# Said return is how much?
retd_highest_p = stock_data_wen$retd[which.max(stock_data_wen$retd)]*100
print(paste0(round(retd_highest_p, 2), "%"))
```

```
[1] "26%"
```