Practice - group_by

Python is used a lot in business, and people who know R might find it hard to switch to Python, and vice versa. So, I will start with an easy example and show you, step-by-step, how to change R code into Python. Following the example, I will also provide descriptions of each function of the code for learning purposes.

First and foremost, it would be best to organize all the packages used in the code in one place at the beginning.

```
library(dplyr)
```

- **dplyr :** This library is used for data manipulation, including filtering, arranging, selecting, and summarizing data. Since I am going to use the `summarize` function here, it is necessary to load the dplyr library.

1. Load data

As an example, the data.frame function is used to store data. Each column is defined with a name on the left and rows of the corresponding data. Using the `<-` left arrow, you can assign a data frame to a variable name.

```
# example
tmp <- data.frame(
    date = c('2021-01-01', '2021-01-01', '2021-01-02'),
    id = c('A', 'B', 'A'),
    return = c(0.5, 0.3, 0.2))
print(tmp)
```

- `data.frame`: this is used to create a table-like structure with rows and columns, where each column can contain different types of data.

- Inside the `data.frame` function, you can define the columns and their corresponding data. columns can be separated by commas (,).
- In R, the `c()` function is used to combine values into a vector. You can use it to create vectors of various types, such as numeric, character, or logical.
    - When you create a vector of strings, you should enclose each string in single quotes (') or double quotes (").

RESULT

As you can see from the below description, there are three different columns with three rows. If you want to see how many IDs are present on each day or find out the average return for each ID over the given two days, you need to write code in R to calculate these values. Although there are numerous ways to solve these, I wrote the code in a simplistic way here. The shorter is better.

Description: df [3 × 3]

date <chr>	id <chr>	return <dbl>
2021-01-01	A	0.5
2021-01-01	B	0.3
2021-01-02	A	0.2

3 rows

2. Group by Date

To group the `id` data by date, we can use `group_by()` and then create a new column to summarize the grouped data.

```
# group by date
grouped_date <- tmp %>%
    group_by(date) %>%
```

```
    summarize(count_date = n_distinct(id))
print(grouped_date)
```

- `%>%` (pipe operator): this operator allows you to pass the output of one function directly into the next function as input.
- `group_by(column name)`: this function allows you to group the data by each unique "column name" value.
- `summarize(new column name = n_distinct(id))`: this function is used to create a new summary of your data.
- `n_distinct`: this function counts the number of unique values in a specified column. This is helpful for identifying the number of unique items.

RESULT

The table below shows the number of unique IDs present on each day. Since there are two entries for the date 2021-01-01, the `count_date` column reflects this with a value of 2.

A tibble: 2 × 2

date <chr>	id <int>
2021-01-01	2
2021-01-02	1

2 rows

3. Group by Id

To group the return data by id, we can follow the same process as above. However, if we want to average the return for each id, we need to use the `mean` function inside the `summarize()` function.

```
# group by id
grouped_id <- tmp %>%
    group_by(id) %>%
    summarize(return = mean(return, na.rm = TRUE))
print(grouped_id
```

- `na.rm = TRUE` : this function tells the function to ignore or remove any `NA` values in the data before carrying out the operation.
- For example, see the below example.

```
data <- c(1, 2, NA, 4, 5)
mean(data, na.rm = TRUE)  # Result: 3
```

RESULT

The table below shows the average of return per each id. Since there are two entries for id 1, the average column displays the calculation (0.5 + 0.2)/ 2 = 0.35

A tibble: 2 × 2

id <chr>	return <dbl>
A	0.35
B	0.30

2 rows