Python Programming – Capstone Project

Capstone Project Overview

Banks pay interest to their customers based on Average Monthly Balance (or Average Quarterly Balance). Many banks charge their customer with Non-Maintenance Charges if the customer does not maintain sufficient Average Balance in their account. As such it is essential for banks to accurately calculate the Average Balance.

The Average Balance for a period is calculated by taking the Sum of End of Day Balance divided by Number of Days.

From programming standpoint, the Average Balance calculation is a complex calculation. It involves knowledge of various programming functionalities like:

- Connecting to Database
- Extracting Required Data
- Data Manipulation
    - Rename Columns
    - Merge (Join)
    - Union (Append)
    - Sort
    - Date Transformation
    - New Field Creation
    - Dropping Unwanted Columns
    - Partitioning Data
    - Referencing Values of Previous Row
    - Aggregation (Group By)
    - Summary Functions like sum, sum product
    - Storing Aggregated Data
- Storing Aggregated Data in Database.

Capstone Project Objective

The objective of the Capstone Project is to test your Python Programming Skills.

The Output expected is the month-wise Average Monthly Balance (AMB) calculation. The detailed calculation for AMB is explained later.

The project will be evaluated based on:

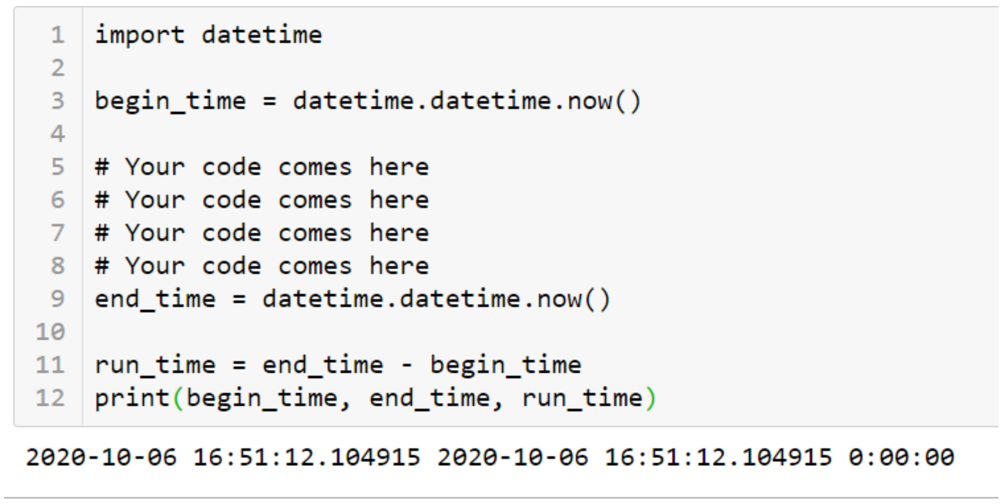
1. Correctness of the Output. Note – The code will be tested on a dataset which is not shared with the participant.
2. Time taken to complete the Capstone Project, i.e., How early you submit your code
3. Programming Logic. How easy it is to understand your logic and comments provided in the code
4. Code Efficiency as measure in terms of time taken to get the final output

Capstone Project Submission

The participants of the Capstone Project are supposed to submit their code in TWO Jupyter Notebook file:

1. Jupyter Notebook containing sample output of the intermediate steps.

2. Jupyter Notebook file where the entire code is written in One Cell as shown below:



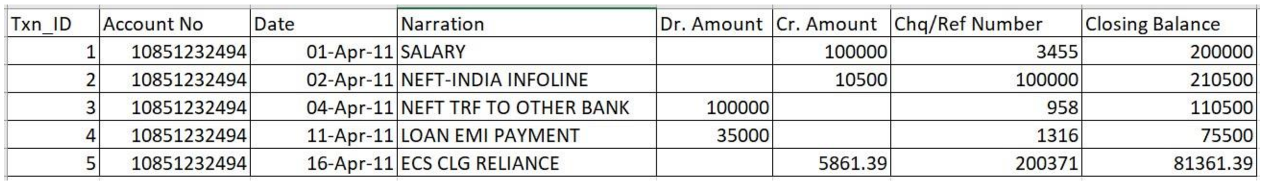
```python
import datetime

begin_time = datetime.datetime.now()

# Your code comes here
# Your code comes here
# Your code comes here
# Your code comes here
end_time = datetime.datetime.now()

run_time = end_time - begin_time
print(begin_time, end_time, run_time)
```

```
2020-10-06 16:51:12.104915 2020-10-06 16:51:12.104915 0:00:00
```

Data

The data is from Banking Domain. The data is in two tables/files.

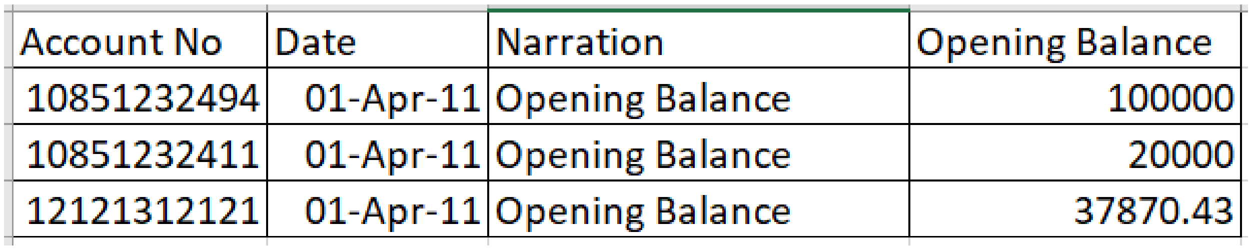
1. **CASA_TXNS**: The table contains the transaction of Current Account & Saving Account holder. The data is given for the period Apr 2011 to Sep 2011. **The data has three distinct account numbers** – 10851232494, 12121312121, and 10851232411. Sample few records are shown below:



Txn_ID	Account No	Date	Narration	Dr. Amount	Cr. Amount	Chq/Ref Number	Closing Balance
1	10851232494	01-Apr-11	SALARY		100000	3455	200000
2	10851232494	02-Apr-11	NEFT-INDIA INFOLINE		10500	100000	210500
3	10851232494	04-Apr-11	NEFT TRF TO OTHER BANK	100000		958	110500
4	10851232494	11-Apr-11	LOAN EMI PAYMENT	35000		1316	75500
5	10851232494	16-Apr-11	ECS CLG RELIANCE		5861.39	200371	81361.39

Metadata of CASA_TXNS table	
Column	**Description**
Txn_ID	Transaction ID. It is unique sequential number at Account No level. For each account the transaction id starts from serial number 1.
Account No	Account Number (Self-explanatory)
Date	Transaction Date
Narration	Description of the transaction
Dr. Amount	The amount debited from account, if it is debit transaction
Cr. Amount	The amount credited to account, if it is credit transaction
Chq/Ref Number	Transaction Reference Number
Closing Balance	The Balance after accounting for the Debit / Credit Transaction

2. **CASA_OPENING_BALANCE**: The opening balance of the account as on 1st April 2011. The fields are all self-explanatory



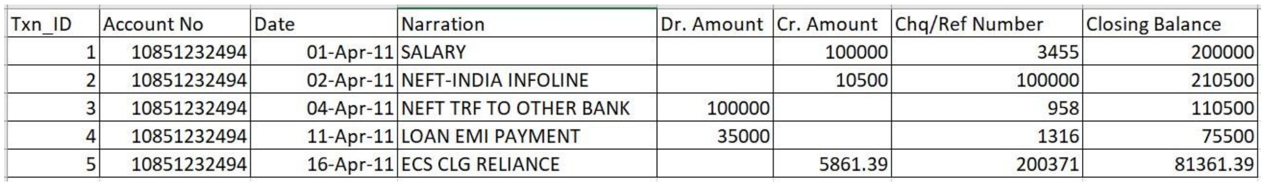
Account No	Date	Narration	Opening Balance
10851232494	01-Apr-11	Opening Balance	100000
10851232411	01-Apr-11	Opening Balance	20000
12121312121	01-Apr-11	Opening Balance	37870.43

Detailed Pseudo Logic

A detailed step-by-step pseudo logic to get the expected output is explained below. Note: The steps are just guidelines. The end output required is a working code which gives the desired output. The participant is free to write his/her own logic to get the required final output.

Step 1: Connect to the database / file system

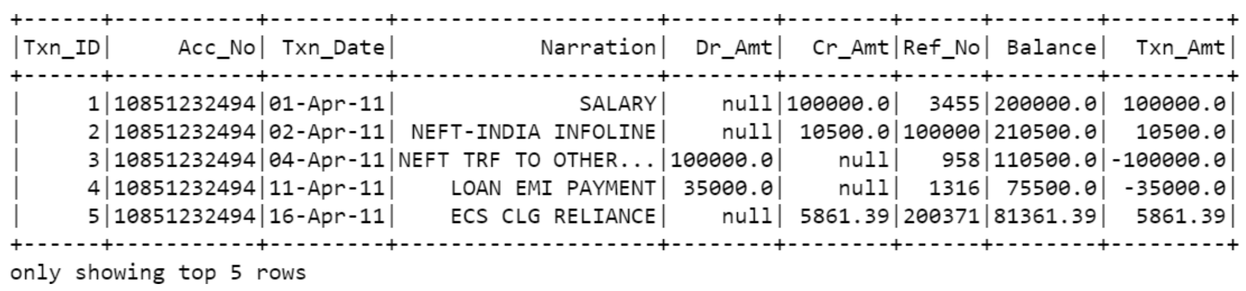
Step 2: Extract the data of CASA_TXNS and get it in Python Environment



Txn_ID	Account No	Date	Narration	Dr. Amount	Cr. Amount	Chq/Ref Number	Closing Balance
1	10851232494	01-Apr-11	SALARY		100000	3455	200000
2	10851232494	02-Apr-11	NEFT-INDIA INFOLINE		10500	100000	210500
3	10851232494	04-Apr-11	NEFT TRF TO OTHER BANK	100000		958	110500
4	10851232494	11-Apr-11	LOAN EMI PAYMENT	35000		1316	75500
5	10851232494	16-Apr-11	ECS CLG RELIANCE		5861.39	200371	81361.39

Step 2A: Rename the columns where necessary

Step 2B: Create a column Txn_Amt. If the transaction Cr. Amount is not null then populate the Txn_Amt = Cr. Amount. For Dr. Amount populate the Txn_Amt = -Dr. Amount

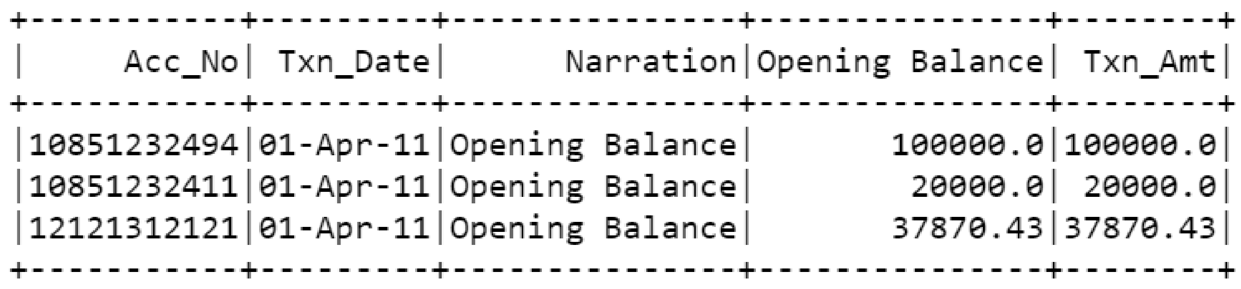


```
+------+-----------+---------+--------------------+--------+--------+------+--------+---------+
|Txn_ID|     Acc_No| Txn_Date|           Narration|  Dr_Amt|  Cr_Amt|Ref_No| Balance|  Txn_Amt|
+------+-----------+---------+--------------------+--------+--------+------+--------+---------+
|     1|10851232494|01-Apr-11|              SALARY|    null|100000.0|  3455|200000.0| 100000.0|
|     2|10851232494|02-Apr-11| NEFT-INDIA INFOLINE|    null| 10500.0|100000|210500.0|  10500.0|
|     3|10851232494|04-Apr-11|NEFT TRF TO OTHER...|100000.0|    null|   958|110500.0|-100000.0|
|     4|10851232494|11-Apr-11|    LOAN EMI PAYMENT| 35000.0|    null|  1316| 75500.0| -35000.0|
|     5|10851232494|16-Apr-11|    ECS CLG RELIANCE|    null| 5861.39|200371|81361.39|  5861.39|
+------+-----------+---------+--------------------+--------+--------+------+--------+---------+
only showing top 5 rows
```

Step 3: Extract the data of CASA_OPENING_BALANCE and get it in Python Environment

Step 3A: Rename the column where necessary

Step 3B: Create a column Txn_Amt. If the Opening Balance is positive the populate the Txn_Amt = Balance else populate the Txn_Amt = -Balance



```
+-----------+---------+-----------------+---------------+--------+
|     Acc_No| Txn_Date|        Narration|Opening Balance| Txn_Amt|
+-----------+---------+-----------------+---------------+--------+
|10851232494|01-Apr-11|Opening Balance|       100000.0|100000.0|
|10851232411|01-Apr-11|Opening Balance|        20000.0| 20000.0|
|12121312121|01-Apr-11|Opening Balance|       37870.43|37870.43|
+-----------+---------+-----------------+---------------+--------+
```

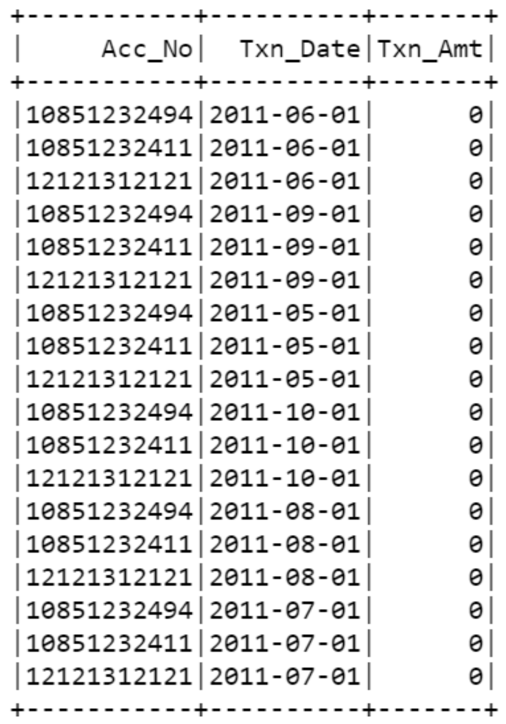
Step 4: Get distinct YYYY-MM from CASA_TXNS

Step 5: Get distinct Account No from CASA_TXNS

Step 6: Join (Merge) the output of the Step 4 and Step 5. There will be no common column to join as such the join will lead to cartesian product. (Note: There will be 18 rows after the join). **This step is required to create Dummy Transaction.**

Step 7: Do the following transformations on the Dummy Transactions created in above step:

a) The YYYY-MM dates be converted to 1st Day of the next month. For example, if the date is 2011-04 then it should be changed to 2011-05-01 (i.e. 01-May-2011)
b) Add Txn_Amt field and populate it with 0. Note: You should give appropriate names to the fields.



```
+-----------+----------+-------+
|     Acc_No|  Txn_Date|Txn_Amt|
+-----------+----------+-------+
|10851232494|2011-06-01|      0|
|10851232411|2011-06-01|      0|
|12121312121|2011-06-01|      0|
|10851232494|2011-09-01|      0|
|10851232411|2011-09-01|      0|
|12121312121|2011-09-01|      0|
|10851232494|2011-05-01|      0|
|10851232411|2011-05-01|      0|
|12121312121|2011-05-01|      0|
|10851232494|2011-10-01|      0|
|10851232411|2011-10-01|      0|
|12121312121|2011-10-01|      0|
|10851232494|2011-08-01|      0|
|10851232411|2011-08-01|      0|
|12121312121|2011-08-01|      0|
|10851232494|2011-07-01|      0|
|10851232411|2011-07-01|      0|
|12121312121|2011-07-01|      0|
+-----------+----------+-------+
```

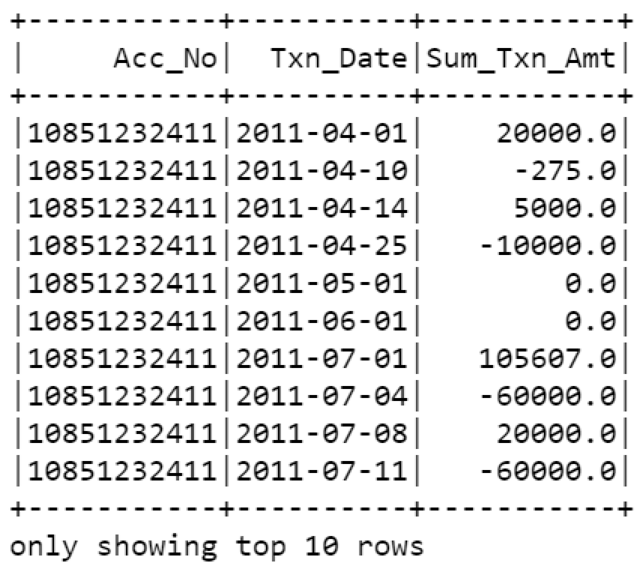
Step 8: Union (Append) the CASA_TXNS dataframe as prepared in step 1 with the Dummy Transactions. Note – The number of columns in CASA_TXNS and Dummy Transactions will not be same. **You require only three columns – Account No, Txn Date, Txn Amount.**

You will have to ensure you keep the required columns and the matching columns have the same column names

Step 9: Union (Append) the output of Step 8 with Output of Step 3

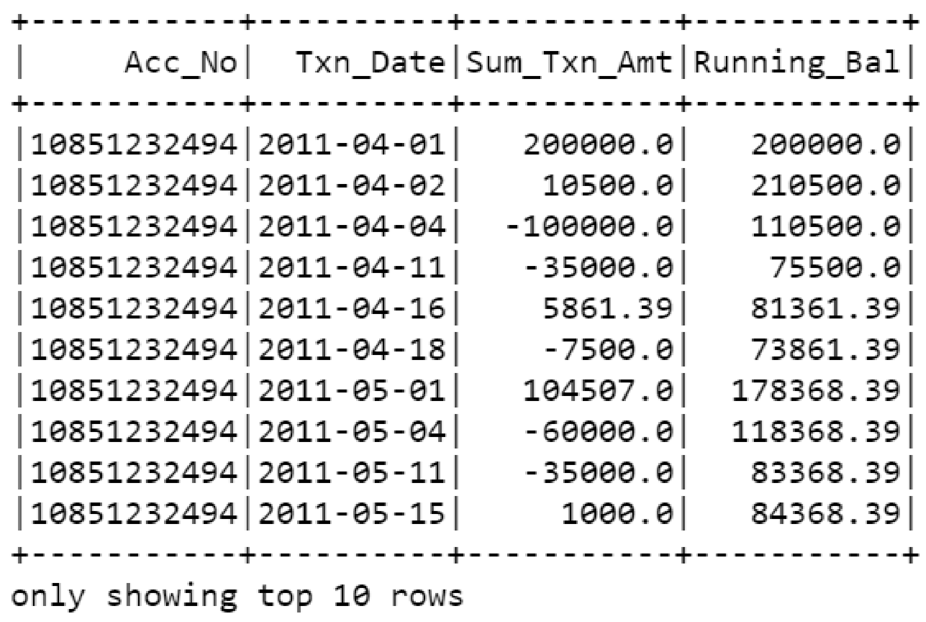
Step 10: Aggregate the Txn_Amt values, group by Account No & Txn Date level.

Step 10A: Sort the data by Account No and Txn Date



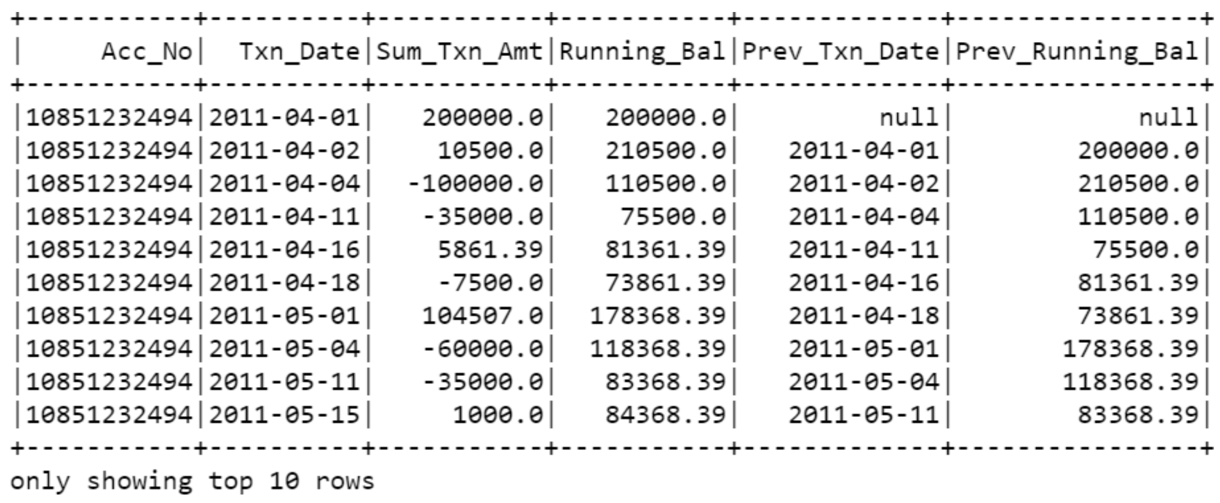
```
+-----------+----------+-----------+
|     Acc_No|  Txn_Date|Sum_Txn_Amt|
+-----------+----------+-----------+
|10851232411|2011-04-01|    20000.0|
|10851232411|2011-04-10|     -275.0|
|10851232411|2011-04-14|     5000.0|
|10851232411|2011-04-25|   -10000.0|
|10851232411|2011-05-01|        0.0|
|10851232411|2011-06-01|        0.0|
|10851232411|2011-07-01|   105607.0|
|10851232411|2011-07-04|   -60000.0|
|10851232411|2011-07-08|    20000.0|
|10851232411|2011-07-11|   -60000.0|
+-----------+----------+-----------+
only showing top 10 rows
```

Step 11: Use the output of step 10 (10A). Create a Running Balance (Cumulative Sum) column by Account Level. For the first transaction of a given account, the value in the Running Balance field will be equal to the Txn Amount. For subsequent rows, the Running Balance = Previous Row Running Balance + Txn Amount value.



```
+-----------+----------+-----------+-----------+
|     Acc_No|  Txn_Date|Sum_Txn_Amt|Running_Bal|
+-----------+----------+-----------+-----------+
|10851232494|2011-04-01|   200000.0|   200000.0|
|10851232494|2011-04-02|    10500.0|   210500.0|
|10851232494|2011-04-04|  -100000.0|   110500.0|
|10851232494|2011-04-11|   -35000.0|    75500.0|
|10851232494|2011-04-16|    5861.39|   81361.39|
|10851232494|2011-04-18|    -7500.0|   73861.39|
|10851232494|2011-05-01|   104507.0|  178368.39|
|10851232494|2011-05-04|   -60000.0|  118368.39|
|10851232494|2011-05-11|   -35000.0|   83368.39|
|10851232494|2011-05-15|     1000.0|   84368.39|
+-----------+----------+-----------+-----------+
only showing top 10 rows
```

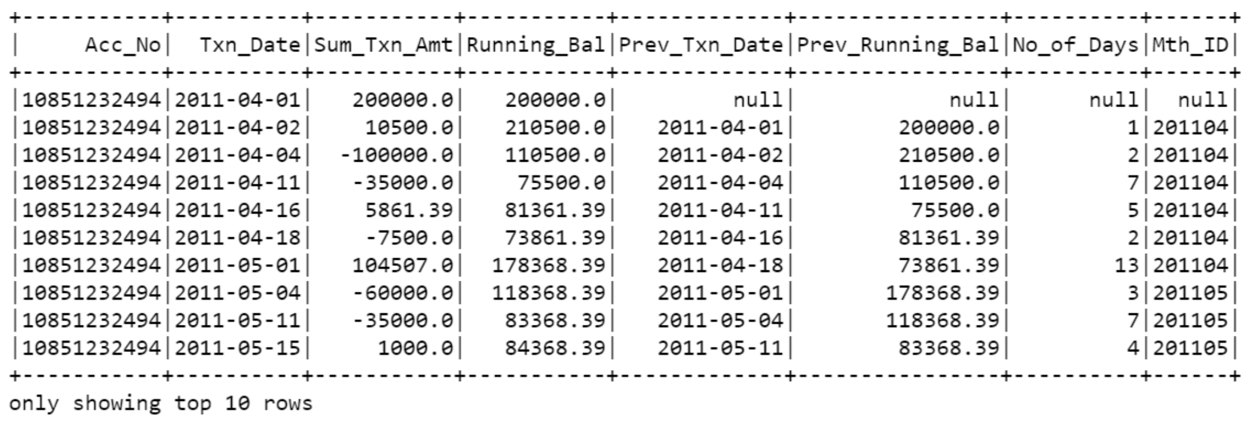
Step 12: To the output of Step 11, add two columns **– Prev_Txn_Date** and **Prev_Running_Bal**



```
+-----------+----------+-----------+-----------+-------------+----------------+
|     Acc_No|  Txn_Date|Sum_Txn_Amt|Running_Bal|Prev_Txn_Date|Prev_Running_Bal|
+-----------+----------+-----------+-----------+-------------+----------------+
|10851232494|2011-04-01|   200000.0|   200000.0|         null|            null|
|10851232494|2011-04-02|    10500.0|   210500.0|   2011-04-01|        200000.0|
|10851232494|2011-04-04|  -100000.0|   110500.0|   2011-04-02|        210500.0|
|10851232494|2011-04-11|   -35000.0|    75500.0|   2011-04-04|        110500.0|
|10851232494|2011-04-16|    5861.39|   81361.39|   2011-04-11|         75500.0|
|10851232494|2011-04-18|    -7500.0|   73861.39|   2011-04-16|        81361.39|
|10851232494|2011-05-01|   104507.0|  178368.39|   2011-04-18|        73861.39|
|10851232494|2011-05-04|   -60000.0|  118368.39|   2011-05-01|       178368.39|
|10851232494|2011-05-11|   -35000.0|   83368.39|   2011-05-04|       118368.39|
|10851232494|2011-05-15|     1000.0|   84368.39|   2011-05-11|        83368.39|
+-----------+----------+-----------+-----------+-------------+----------------+
only showing top 10 rows
```

Step 13: Calculate Date Difference between **Txn_Date** and **Prev_Txn_Date** and store in some column (say No_Of_Days). The number of days is the period for which the previous balance was maintained in the account at each End of Day.

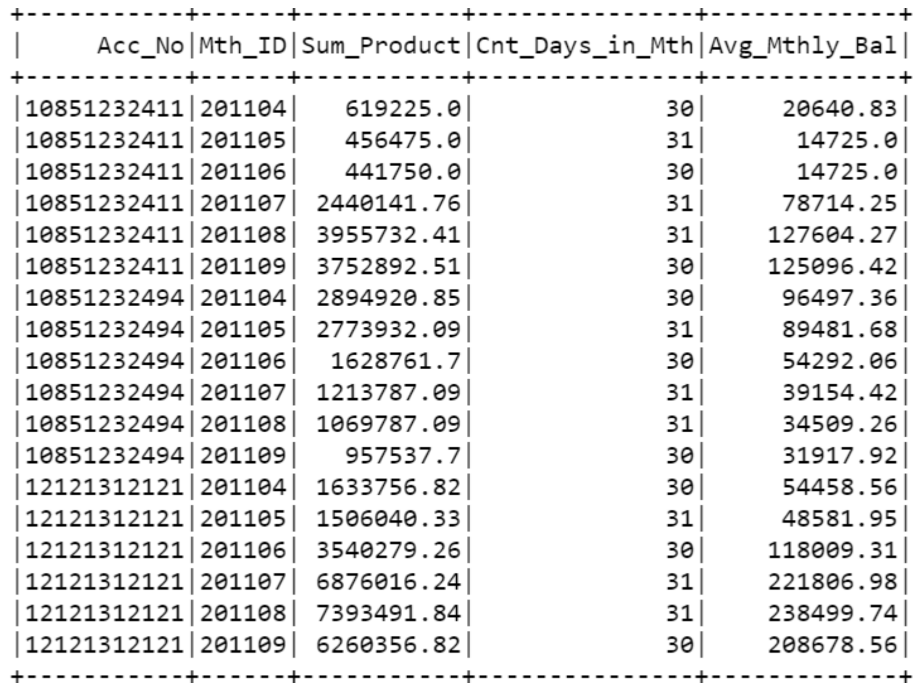
Step 14: Add a column Mth_ID having value in YYYYMM format. The value for this column should be computed from **Prev_Txn_Date.**



```
+-----------+----------+-----------+-----------+-------------+----------------+----------+------+
|     Acc_No|  Txn_Date|Sum_Txn_Amt|Running_Bal|Prev_Txn_Date|Prev_Running_Bal|No_of_Days|Mth_ID|
+-----------+----------+-----------+-----------+-------------+----------------+----------+------+
|10851232494|2011-04-01|   200000.0|   200000.0|         null|            null|      null|  null|
|10851232494|2011-04-02|    10500.0|   210500.0|   2011-04-01|        200000.0|         1|201104|
|10851232494|2011-04-04|  -100000.0|   110500.0|   2011-04-02|        210500.0|         2|201104|
|10851232494|2011-04-11|   -35000.0|    75500.0|   2011-04-04|        110500.0|         7|201104|
|10851232494|2011-04-16|    5861.39|   81361.39|   2011-04-11|         75500.0|         5|201104|
|10851232494|2011-04-18|    -7500.0|   73861.39|   2011-04-16|        81361.39|         2|201104|
|10851232494|2011-05-01|   104507.0|  178368.39|   2011-04-18|        73861.39|        13|201104|
|10851232494|2011-05-04|   -60000.0|  118368.39|   2011-05-01|       178368.39|         3|201105|
|10851232494|2011-05-11|   -35000.0|   83368.39|   2011-05-04|       118368.39|         7|201105|
|10851232494|2011-05-15|     1000.0|   84368.39|   2011-05-11|        83368.39|         4|201105|
+-----------+----------+-----------+-----------+-------------+----------------+----------+------+
only showing top 10 rows
```

Step 15: Calculate Month wise Average Monthly Balance by aggregating the data at Account No, YYYYMM level. While aggregating filter out the rows where Mth_ID is null.

AMB = sum(Prev_Running_Bal * No_Of_Days) / sum(No_Of_Days)



```
+-----------+------+-----------+---------------+------------+
|     Acc_No|Mth_ID|Sum_Product|Cnt_Days_in_Mth|Avg_Mthly_Bal|
+-----------+------+-----------+---------------+------------+
|10851232411|201104|   619225.0|             30|    20640.83|
|10851232411|201105|   456475.0|             31|     14725.0|
|10851232411|201106|   441750.0|             30|     14725.0|
|10851232411|201107| 2440141.76|             31|    78714.25|
|10851232411|201108| 3955732.41|             31|   127604.27|
|10851232411|201109| 3752892.51|             30|   125096.42|
|10851232494|201104| 2894920.85|             30|    96497.36|
|10851232494|201105| 2773932.09|             31|    89481.68|
|10851232494|201106|  1628761.7|             30|    54292.06|
|10851232494|201107| 1213787.09|             31|    39154.42|
|10851232494|201108| 1069787.09|             31|    34509.26|
|10851232494|201109|   957537.7|             30|    31917.92|
|12121312121|201104| 1633756.82|             30|    54458.56|
|12121312121|201105| 1506040.33|             31|    48581.95|
|12121312121|201106| 3540279.26|             30|   118009.31|
|12121312121|201107| 6876016.24|             31|   221806.98|
|12121312121|201108| 7393491.84|             31|   238499.74|
|12121312121|201109| 6260356.82|             30|   208678.56|
+-----------+------+-----------+---------------+------------+
```

Step 16: Cross Check your output with the above output

Step 17: Push the output of above step in database/file system

Step 18: WOW… WOW…WOW. You have done it!!! Congratulations.

A) Send your Jupyter Code file having all intermediate output to the training coordinator.
B) Create a Jupyter File where the whole code is written in one Cell.