Daily Transaction processing with PySpark

Basic Requirements

Most of the RentSpree transactions are based on transactions and we need to develop a special method to generate new_users, dropoff_users and returning_users based on following calculation:

- Total number of **new_users** in month M:
  - The number of users who create the first transaction in month M
- Total number of **dropoff_users** in month M:
  - The number of users who didn't create transactions longer than the churn period (Days) in month M
- Total number of **returning_users** in month M:
  - The number of drop-off users who create the transactions in month M

daily_transactions.csv

- **_id**: The unique id for users
- **event_date**: The date that the user has created the transaction in our system
- **usertype**: The type of user. [A, B, C, D]. If you found the same user who has more than 1

_id	event_date	usertype
By1+rEy20nW/sgRehb+RSZe9VI8=	2021-02-10	A
x7IxSW+y6Kymzxf1yrriMXDNZWQ=	2021-02-10	B
BQ7feJAnc6ntg4PaOXShHLrd3xQ=	2021-02-24	D
xmc9CsW3Q7K9HK5+pczuRvKUCBk=	2020-12-08	D

PySpark Initialization and Data Loading

1. Import SparkSession from PySpark and the same session will be used for the whole process.

```
from pyspark.sql import SparkSession
```

2. Declare SparkSession as spark with the application name RentSpree

```
spark = SparkSession.builder.appName('RentSpree').getOrCreate()
```

3. Read the daily_transactions.csv file and convert it to spark dataframe.

```
df=spark.read.option('header','true').csv('daily_transactions/daily_transactions.csv',inferSchema=True)
```

df.printSchema()

```
|-- _id: string (nullable = true)
|-- event_date: timestamp (nullable = true)
|-- usertype: string (nullable = true)
```

df.show()

```
+--------------------+-------------------+--------+
|                 _id|         event_date|usertype|
+--------------------+-------------------+--------+
|By1+rEy20nW/sgReh...|2021-02-10 00:00:00|       A|
|x7lxSW+y6Kymzxf1y...|2021-02-10 00:00:00|       A|
|BQ7feJAnc6ntg4PaO...|2021-02-24 00:00:00|       A|
|vb6FkuOhgdOzmGpgl...|2021-01-18 00:00:00|       A|
|hginiEQhCS1lA6bns...|2020-05-02 00:00:00|       A|
|TqcP/Vc8nfJIaOG9T...|2020-11-08 00:00:00|       A|
+--------------------+-------------------+--------+
```

Data Preparation

According to the requirements, we need remove the users from the calculation who have more than 1 usertype.

Find out who has more than one user type.

In SQL:

```
with duplicate_id_lst as (
  select _id from
    (select _id, count(*) as count from (
      select distinct _id,usertype from `daily_transaction`
    )
    group by _id
  ) where count > 1
)
select * from `daily_transaction` where _id not in (select _id from duplicate_id_lst);
```

Find unique combination of **_id** and **usertype**. Get the **_id** count of each combination.

Get all the **_id** which count is more than 1. We can assume that this kind of user has more than 1 usertype.

Exclude **_id** that we get from 2nd step.

In Spark:

```
dis_df = df.select(['_id','usertype']).distinct().groupBy(['_id']).count()
duplicate_id_lst = dis_df.select('_id').filter(dis_df['count']>1).rdd.map(lambda x: x._id).collect()
df = df[~df['_id'].isin(duplicate_id_lst)]
```

Get New Users - Logic

New user thinking process:

- We can assume that one is new user if he/she doesn't have previous event in our system.
- To do that we will first generate the minimum event_date for each user.
- If the minimum event_date for specific user is the same with his event_date, we can see this is the new user.

Step1:

Generate 2 new columns:
first_event:
minimum event_date of user.
new_user:
boolean value if **first_event** date is the same with **event_date**.

x7lxSW+y6Kymzxf1yrriMXDNZWQ= new_user for 17 April 2020

_id	month	event_date	first_event	new_user
x7lxSW+y6Kymzxf1yrriMXDNZWQ=	Apr-20	17/04/2020	17/04/2020	TRUE
x7lxSW+y6Kymzxf1yrriMXDNZWQ=	Apr-20	18/04/2020	17/04/2020	FALSE
x7lxSW+y6Kymzxf1yrriMXDNZWQ=	Apr-20	22/04/2020	17/04/2020	FALSE
x7lxSW+y6Kymzxf1yrriMXDNZWQ=	Apr-20	28/04/2020	17/04/2020	FALSE
x7lxSW+y6Kymzxf1yrriMXDNZWQ=	May-20	02/05/2020	17/04/2020	FALSE
x7lxSW+y6Kymzxf1yrriMXDNZWQ=	May-20	06/05/2020	17/04/2020	FALSE

Step2:

Get unique count of user for each month who **new_user** value is **TRUE**.

month	new_users
Apr-20	15117
Apr-21	34927
Aug-19	11755
Aug-20	28019
Dec-19	8567
Dec-20	19071
Feb-20	12891
Feb-21	22676
Jan-20	12732
Jan-21	23389

Get New Users - Code In SQL

```
select month, count(*) from
(
   select m._id, FORMAT_DATE('%b %Y', m.event_date) as month,m.event_date, first_event, (m.event_date=first_event) as
new_user
   from `daily_transaction` m
   inner join
  (
      SELECT _id, min(event_date) as first_event FROM `daily_transaction` group by 1
   ) as f_date
   on m._id=f_date._id
)
where new_user is true
group by 1;
```

Get minimum event date as **first_event** from each user.

Extract only month and year as **month**.

Join with the result from first step to get **first_event** and **new_user** column.

new_user: boolean value if **first_event** date is the same with **event_date**.

Get unique count of user for each month who **new_user** value is **TRUE**.

Get New Users - Code In Spark

```
first_event_df = df.groupBy('_id').agg(F.min('event_date').alias("first_event"))

join_df = df.join(first_event_df, df['_id']==first_event_df['_id'], 'inner').drop(first_event_df._id)

newuser_df = join_df.withColumn("new_user", join_df["event_date"]==join_df["first_event"]).withColumn('month',
F.date_format(join_df["event_date"], "MMMM yyyy"))

newuser_count_df =
newuser_df.filter(newuser_df["new_user"]==True).groupBy("month").count().withColumnRenamed("count","new_users")
```

Get minimum event date as **first_event** from each user.

Join with the result from first step to get **first_event** column and drop the additional column.

Calculate **new_user** column and extract only month and year as **month**.

new_user: boolean value if **first_event** date is the same with **event_date**.

Get unique count of user for each month who **new_user** value is **TRUE**.

Get Returning Users - Logic

Returning user thinking process:

- We can assume that one is returning user if he/she come back after inactive for more than **churn period**.
- To do that, we need to get previous event of the user and compare it with current event to find how many days he's been inactive for.
- If he is inactive for more than churn period, we can assume that he/she is returning user for this current event.

_id	event_date	usertype
KolCNj3XmRRpkxvl7iguxYAT8TY=	30/05/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	05/09/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	10/09/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	11/09/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	02/10/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	07/10/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	02/03/2021	A

Date difference
512 days

KolCNj3XmRRpkxvl7iguxYAT8TY= returning_user for 02 March 2021

Step1:

Generate 2 new columns:
previous_event:
the previous event_date before the current event_date.
date_diffrent:
inactive days after previous_event. It is calculated by **event_date - previous_event**.

Note: Null values mean user doesn't have previous event before that and it's the first time he enter into system.
We have to drop that row.

Need to drop

_id	event_date	usertype	previous_event	date_different
KolCNj3XmRRpkxvl7iguxYAT8TY=	30/05/2019	A	null	null
KolCNj3XmRRpkxvl7iguxYAT8TY=	05/09/2019	A	30/05/2019	98
KolCNj3XmRRpkxvl7iguxYAT8TY=	10/09/2019	A	05/09/2019	5
KolCNj3XmRRpkxvl7iguxYAT8TY=	11/09/2019	A	10/09/2019	1
KolCNj3XmRRpkxvl7iguxYAT8TY=	02/10/2019	A	11/09/2019	21
KolCNj3XmRRpkxvl7iguxYAT8TY=	07/10/2019	A	02/10/2019	5
KolCNj3XmRRpkxvl7iguxYAT8TY=	02/03/2021	A	07/10/2019	512

event_date - previous_event

Get Returning Users - Logic

Step2:

Generate **churn_period** table for each usertype along with churn period.

usertype	period
A	360
B	360
C	120
D	260

Step3:

Attach 1 column:
period: churn period for each user type. Can get it by joining with **churn_period** table on **usertype**.

Generate 2 columns:
returning_user: boolean value if **date_different** is greater than equal with **period**.
month: extract month and year as MMMM YYYY format from event_date.

Step4:

Get unique count of user for each month who **returning_user** value is **TRUE**.

month	returning_users
Apr-21	946
Mar-21	719
Feb-21	569
Jan-21	560
Oct-20	393
Dec-20	387
Nov-20	351
Aug-20	347
Jul-20	345
Sep-20	289
Jun-20	210
May-20	157
May-21	96
Mar-20	94
Apr-20	86
Feb-20	81
Jan-20	76
Dec-19	41
Nov-19	22
Oct-19	15
Sep-19	3

_id	event_date	usertype	previous_event	date_different	period	returning_user	month
KoICNj3XmRRpkxvl7iguxYAT8TY=	05/09/2019	A	30/05/2019	98	360	FALSE	Sep-19
KoICNj3XmRRpkxvl7iguxYAT8TY=	10/09/2019	A	05/09/2019	5	360	FALSE	Sep-19
KoICNj3XmRRpkxvl7iguxYAT8TY=	11/09/2019	A	10/09/2019	1	360	FALSE	Sep-19
KoICNj3XmRRpkxvl7iguxYAT8TY=	02/10/2019	A	11/09/2019	21	360	FALSE	Oct-19
KoICNj3XmRRpkxvl7iguxYAT8TY=	07/10/2019	A	02/10/2019	5	360	FALSE	Oct-19
KoICNj3XmRRpkxvl7iguxYAT8TY=	02/03/2021	A	07/10/2019	512	360	TRUE	Mar-21

KoICNj3XmRRpkxvl7iguxYAT8TY= returning_user for 02 March 2021

Get Returning Users - Code In SQL

```sql
with churn_period as (
  select 'A' as usertype, 360 as period union all
  select 'B' as usertype, 360 as period union all
  select 'C' as usertype, 120 as period union all
  select 'D' as usertype, 260 as period
),
date_differ as (
  select _id, event_date, usertype, previous_event, date_diff(event_date, previous_event, DAY) as date_different from
  (
    select *, lag(event_date,1) over (partition by _id order by event_date) as previous_event
    from `daily_transaction`
  ) where previous_event is not null
),
return_user as (
  select _id, event_date, date_differ.usertype, previous_event, date_different, period, (date_different>=period) as returning_user, FORMAT_DATE('%b %Y',
event_date) as month
  from date_differ
  inner join churn_period
  on date_differ.usertype=churn_period.usertype
)
select month, count(*) as returning_users from (select distinct month, _id, returning_user from return_user)
where returning_user is true
group by 1;
```

Step2:
Generate **churn_period** table for each usertype along with churn period.

Step1:
Generate **previoud_event** and **date_different**. Drop the null values.

Step3:
Attach **period** column by joining with **churn_period** table on **usertype**. Generate **returning_user** and **month** columns.

Step4:
Get unique count of user for each month who **returning_user** value is **TRUE**.

Get Returning Users - Code In Spark

```
data = [("A",360), ("B",360),("C",120),("D",260)]
churn_period_df = spark.createDataFrame(data=data,schema=['usertype', 'period'])

windowSpec  = Window.partitionBy("_id").orderBy("event_date")
previous_event_df = df.withColumn("previous_event", F.lag("event_date",1).over(windowSpec)).na.drop()
date_diff_df = previous_event_df.withColumn('date_diff', F.datediff(previous_event_df['event_date'], previous_event_df['previous_event']))

join_period_df = date_diff_df.join(churn_period_df, date_diff_df['usertype']==churn_period_df['usertype'],
'inner').drop(churn_period_df['usertype'])
return_user_df = join_period_df.withColumn("return_user", join_period_df["date_diff"]>=join_period_df["period"]).withColumn('month',
F.date_format(df["event_date"], "MMMM yyyy"))

return_count_df = return_user_df.select(['month',
'_id']).distinct().filter(return_user_df["return_user"]==True).groupBy("month").count().withColumnRenamed("count","returning_users")
```

Step1:
Generate **previous_event** by doing lag window operation and dropped the null values.
Generate **date_diff** column as **event_date - previous_event.**

Step2:
Generate **churn_period_df** dataframe for each usertype along with churn period.

Step3:
Attach **period** column by joining with **churn_period_df** dataframe on **usertype**.
Generate **returning_user** and **month** columns.

Step4:
Get unique count of user for each month who **returning_user** value is **TRUE**.

Get Drop-off Users - Logic

Drop-off user thinking process:

- We can assume that one is drop-off user if he/she is inactive for **churn period**.
- To do that, we have to find if the user has next event after current event and calculate the number of days to next event.
  Note: **If the user doesn't have next event, we will have to compare it with the latest date in database to calculate how many days he's been inactive for until that point.**
- If the number of days is more than churn period, we can assume that the user dropped off in the middle and we need to find exact date he dropped off. (which is **event_date + churn period**).

_id	event_date	usertype
KolCNj3XmRRpkxvl7iguxYAT8TY=	30/05/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	05/09/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	10/09/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	11/09/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	02/10/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	07/10/2019	A
KolCNj3XmRRpkxvl7iguxYAT8TY=	02/03/2021	A

Date difference
512 days

KolCNj3XmRRpkxvl7iguxYAT8TY=

drop-off date = (07/10/2019 + 360) = 01/10/2020

Step1:

Generate 2 new columns:
next_event: the next event_date before the current event_date.
date_diffrent: number of days until next_event. It is calculated by **next_event** - **event_date**.
Note: If user doesn't have next event, that means he's still inactive and we need to count the inactive days until today.
As our database is not live, we will get lasted event_date from database instead of today and compare it with current event_date.

_id	event_date	usertype	next_event	date_different
KolCNj3XmRRpkxvl7iguxYAT8TY=	30/05/2019	A	05/09/2019	98
KolCNj3XmRRpkxvl7iguxYAT8TY=	05/09/2019	A	10/09/2019	5
KolCNj3XmRRpkxvl7iguxYAT8TY=	10/09/2019	A	11/09/2019	1
KolCNj3XmRRpkxvl7iguxYAT8TY=	11/09/2019	A	02/10/2019	21
KolCNj3XmRRpkxvl7iguxYAT8TY=	02/10/2019	A	07/10/2019	5
KolCNj3XmRRpkxvl7iguxYAT8TY=	07/10/2019	A	02/03/2021	512
KolCNj3XmRRpkxvl7iguxYAT8TY=	02/03/2021	A	04/05/2021	63

Latest date in database next_event - event_date

Get Drop-off Users - Logic

Step2:

Attach 1 column:
period: churn period for each user type. Can get it by joining with **churn_period** table on **usertype**.

Generate 1 column:
dropped: boolean value if **date_different** is greater than equal with **period**.

_id	event_date	usertype	next_event	date_different	period	dropped
KolCNj3XmRRpkxvI7iguxYAT8TY=	30/05/2019	A	05/09/2019	98	360	FALSE
KolCNj3XmRRpkxvI7iguxYAT8TY=	05/09/2019	A	10/09/2019	5	360	FALSE
KolCNj3XmRRpkxvI7iguxYAT8TY=	10/09/2019	A	11/09/2019	1	360	FALSE
KolCNj3XmRRpkxvI7iguxYAT8TY=	11/09/2019	A	02/10/2019	21	360	FALSE
KolCNj3XmRRpkxvI7iguxYAT8TY=	02/10/2019	A	07/10/2019	5	360	FALSE
KolCNj3XmRRpkxvI7iguxYAT8TY=	07/10/2019	A	02/03/2021	512	360	TRUE
KolCNj3XmRRpkxvI7iguxYAT8TY=	02/03/2021	A	04/05/2021	63	360	FALSE

KolCNj3XmRRpkxvI7iguxYAT8TY=

dropped user

Step3:

Filter only **dropped** is **TRUE.**

Generate 1 column:
dropped_date: calculated once user is inactive for churn period. (event_date + period).

_id	event_date	usertype	next_event	date_different	period	dropped	dropped_date
KolCNj3XmRRpkxvI7iguxYAT8TY=	07/10/2019	A	02/03/2021	512	360	TRUE	01/10/2020

(07/10/2019 + 360) = **01/10/2020**

Get Drop-off Users - Logic

Step4:

Generate 1 column:
month: extract month and year as MMMM YYYY format from **dropped_date**.

Get unique count of drop-off user for each **month** of **dropped_date**.

month	dropoff_users
Sep-19	269
Oct-19	516
Dec-19	534
Nov-19	550
Apr-20	828
Mar-20	964
Feb-20	1025
Jan-20	1056
May-21	2433
May-20	8111
Dec-20	9399
Nov-20	9846
Oct-20	10238
Sep-20	10312
Jun-20	10923
Aug-20	11222
Mar-21	12231
Jul-20	12456
Feb-21	12618
Jan-21	13126
Apr-21	15101

Get Drop-off Users - Code In SQL

```sql
with churn_period as (
   select 'A' as usertype, 360 as period union all
   select 'B' as usertype, 360 as period union all
   select 'C' as usertype, 120 as period union all
   select 'D' as usertype, 260 as period),
date_differ as (
   select _id,event_date,main.usertype,coalesce(next_event, m_date) as next_event, date_diff(coalesce(next_event, m_date), event_date, DAY) as
date_different from
   ( select *, lead(event_date,1) over (partition by _id order by event_date) as next_event
      from `daily_transaction` ) main cross join
      (select max(event_date) as m_date from `daily_transaction`) max_date
),
dropped_user as (
   select _id, event_date, main.usertype, next_event, date_different, period,
      (date_different>=period) as dropped
   from date_differ main  inner join churn_period
   on main.usertype=churn_period.usertype
),
dropped_dt as (
   select *, date_add(event_date, INTERVAL period DAY) as dropped_date
      from dropped_user  where dropped is true
)
select FORMAT_DATE('%b %Y', dropped_date) as month, count(*) as dropoff_users from
(select distinct dropped_date, _id, dropped from dropped_dt
group by 1;
```

Step1:
Generate **next_event** and **date_diffrent** columns.
And cross join with maximum event_date as **m_date** from database.
m_date will be used if the user has no **next_event**.

Step2:
Attach **period** column by joining with **churn_period** table on **usertype**.
Generate **dropped** column which will return **True** if **date_different** is greater than equal with **period**.

Step3:
Filter only **dropped** is **TRUE.**
Generate **dropped_date** column which is calculated once user is inactive for churn period. (event_date + period).

Step4:
Generate **month** column by extracting month and year as MMMM YYYY format from **dropped_date**.
Get unique count of drop-off user for each **month** of **dropped_date**.

Get Drop-off Users - Code In Spark

```
windowSpec = Window.partitionBy("_id").orderBy("event_date")
max_date = df.select(F.max('event_date'))
next_event_df = df.withColumn("next_event", F.lead("event_date",1).over(windowSpec)).crossJoin(max_date).withColumnRenamed('max(event_date)',
'max_date')
next_event_df = next_event_df.select(['_id','event_date','usertype',F.coalesce(next_event_df['next_event'], next_event_df['max_date']).alias('next_event')])
date_diff_df = next_event_df.withColumn('date_diff', F.datediff(next_event_df['next_event'], next_event_df['event_date']))

join_period_df = date_diff_df.join(churn_period_df, date_diff_df['usertype']==churn_period_df['usertype'], 'inner').drop(churn_period_df['usertype'])
dropped_user_df = join_period_df.withColumn("dropped", join_period_df["date_diff"]>=join_period_df["period"]).withColumn("period",
join_period_df.period.cast('int'))

dropped_dt_df = dropped_user_df.withColumn("dropped_date", F.date_add(dropped_user_df['event_date'],
dropped_user_df['period'])).filter(dropped_user_df["dropped"]==True)

dropped_mn_df = dropped_dt_df.withColumn('month', F.date_format(dropped_dt_df["dropped_date"], "MMMM yyyy"))
dropped_count_df = dropped_mn_df.select(['month', '_id', 'dropped']).distinct().groupBy("month").count().withColumnRenamed("count","dropoff_users")
```

Step1:
Get max_date from database and cross join to attach **max_date**. **max_date** will be used if the user has no **next_event**.
Generate **next_event** by doing lead window operation and **date_diff** column.

Step2:
Attach **period** column by joining with **churn_period_df** dataframe on **usertype**.
Generate **dropped** column which will return **True** if **date_different** is greater than equal with **period**.

Step3:
Generate **dropped_date** column which is calculated once user is inactive for churn period. (event_date + period).
Filter only **dropped** is **TRUE.**

Step4:
Generate **month** column by extracting month and year as MMMM YYYY format from **dropped_date**.
Get unique count of drop-off user for each **month** of **dropped_date**.

Join Everything - Code In Spark

Join all data frames together dropped_count_df + return_count_df + newuser_count_df on **month**

And if month is null for a data frame, it will get value from another data frame with month value is present.

```
result_df = dropped_count_df.join(return_count_df, dropped_count_df['month']==return_count_df['month'],
'outer').withColumn('c_month', F.coalesce(dropped_count_df['month'], return_count_df['month'])).drop('month')
result_df = result_df.join(newuser_count_df, result_df['c_month']==newuser_count_df['month'],
'outer').withColumn('c_month', F.coalesce(result_df['c_month'], newuser_count_df['month'])).drop('month')
result_df = result_df.withColumnRenamed('c_month', 'month').na.fill(0).select(['month', 'dropoff_users',
'returning_users', 'new_users'])
```

result_df.printSchema()

```
|-- month: string (nullable = true)
|-- dropoff_users: long (nullable = true)
|-- returning_users: long (nullable = true)
|-- new_users: long (nullable = true)
```

result_df.show()

month	dropoff_users	returning_users	new_users
April 2020	828	86	15117
April 2021	15101	946	34927
August 2019	0	0	11755
August 2020	11222	347	28019
December 2019	534	41	8567
December 2020	9399	387	19071
February 2020	1025	81	12891
February 2021	12618	569	22676
January 2020	1056	76	12732
January 2021	13126	560	23389

Filter Date Range - Code In Spark

According to the requirements, we have to filter date from April 2020 to April 2021.

First we need to change **month** from result_df to **date** format and do the filter using **between**.

```
from_date = 'April 2020'
to_date = 'April 2021'
from_date = datetime.strptime(from_date, "%B %Y").date()
to_date = datetime.strptime(to_date, "%B %Y").date()

result_df = result_df.withColumn('month_num',F.to_date(result_df['month'],"MMMM yyyy"))
result_df = result_df.filter(F.col('month_num').between(from_date,
to_date)).sort(result_df['month_num']).drop('month_num')
```

result_df.show()

Final Result !

month	dropoff_users	returning_users	new_users
April 2020	828	86	15117
May 2020	8111	157	20295
June 2020	10923	210	22949
July 2020	12456	345	30152
August 2020	11222	347	28019
September 2020	10312	289	21693
October 2020	10238	393	22854
November 2020	9846	351	18875
December 2020	9399	387	19071
January 2021	13126	560	23389
February 2021	12618	569	22676
March 2021	12231	719	31369
April 2021	15101	946	34927

Store the file into CSV

According to the requirements, we have to store the result file as CSV format.

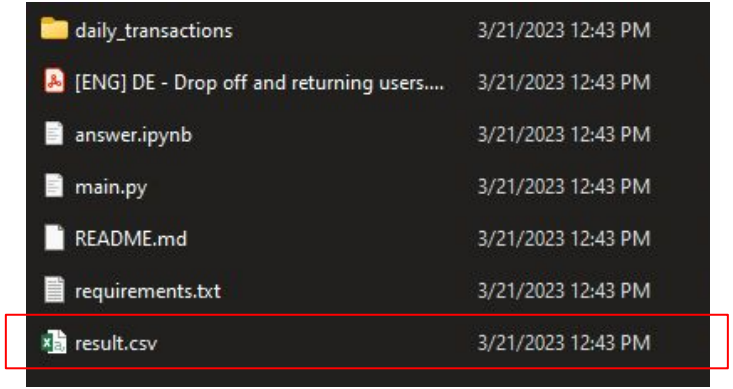
First write file to single csv file using coalesce(1) with header and replace the **result** folder if existing.

Then, move only result CSV file to main directory and remove the **result** folder.

```
result_df.coalesce(1).write.csv(path="result", header=True, mode="overwrite")

file_name=list(filter(lambda x: x.endswith('.csv'), shutil.os.listdir('result')))[0]
shutil.move(f'result/{file_name}', 'result.csv')
shutil.rmtree('result', ignore_errors=True)
```

Check result.csv file under main directory.



daily_transactions	3/21/2023 12:43 PM
[ENG] DE - Drop off and returning users....	3/21/2023 12:43 PM
answer.ipynb	3/21/2023 12:43 PM
main.py	3/21/2023 12:43 PM
README.md	3/21/2023 12:43 PM
requirements.txt	3/21/2023 12:43 PM
result.csv	3/21/2023 12:43 PM

Check github to install the requirements and run the code.
https://github.com/Thihahtoo/spark-daily-transaction

Thank You