PYSPARK GOLD LEVEL PRACTICE RESOURCE.

Step - 2 : Identifying The Input Data And Expected

INPUT

INPUT		
ACTOR_ID	DIRECTOR_ID	TIMESTAMP
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5
2	1	6

OUTPUT

OUTPUT	
ACTOR_ID	DIRECTOR_ID
1	1

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField,IntegerType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()


schema = StructType([
            StructField("ActorId",IntegerType(),True),
            StructField("DirectorId",IntegerType(),True),
            StructField("timestamp",IntegerType(),True)
])

data = [
            (1, 1, 0),
            (1, 1, 1),
            (1, 1, 2),
            (1, 2, 3),
            (1, 2, 4),
            (2, 1, 5),
            (2, 1, 6)
]
```

```
df=spark.createDataFrame(data,schema)
df.show()
```

```
+-------+----------+---------+
|ActorId|DirectorId|timestamp|
+-------+----------+---------+
|      1|         1|        0|
|      1|         1|        1|
|      1|         1|        2|
|      1|         2|        3|
|      1|         2|        4|
|      2|         1|        5|
|      2|         1|        6|
+-------+----------+---------+
```

```
df_group=df.groupBy('ActorId','DirectorId').count()
df_group.show()
```

```
+-------+----------+-----+
|ActorId|DirectorId|count|
+-------+----------+-----+
|      1|         2|    2|
|      1|         1|    3|
|      2|         1|    2|
+-------+----------+-----+
```

```
df_group.filter(df_group['count'] >= 3).show()
```

```
+-------+----------+-----+
|ActorId|DirectorId|count|
+-------+----------+-----+
|      1|         1|    3|
+-------+----------+-----+
```

Step - 1 : Problem Statement

Ads Performance

Write an pyspark code to find the ctr of each Ad.Round ctr to 2 decimal points. Order the result table by ctr in descending order and by ad_id in ascending order in case of a tie.

Ctr=Clicked/(Clicked+Viewed)

Difficult Level : EASY

DataFrame:

```
# Define the schema for the Ads table
schema=StructType([
        StructField('AD_ID',IntegerType(),True)
        ,StructField('USER_ID',IntegerType(),True)
        ,StructField('ACTION',StringType(),True)
])

# Define the data for the Ads table
data = [
        (1, 1, 'Clicked'),
        (2, 2, 'Clicked'),
        (3, 3, 'Viewed'),
        (5, 5, 'Ignored'),
        (1, 7, 'Ignored'),
        (2, 7, 'Viewed'),
        (3, 5, 'Clicked'),
        (1, 4, 'Viewed'),
        (2, 11, 'Viewed'),
        (1, 2, 'Clicked')
]
```

Step - 2 : Identifying The Input Data And Expected

INPUT

INPUT		
AD_ID	USER_ID	ACTION
1	1	Clicked
2	2	Clicked
3	3	Viewed
5	5	Ignored
1	7	Ignored
2	7	Viewed
3	5	Clicked
1	4	Viewed
2	11	Viewed
1	2	Clicked

OUTPUT

OUTPUT	
AD_ID	CTR
1	0.67
3	0.5
2	0.33
5	0

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session

from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType
from pyspark.sql.functions import when
from pyspark.sql import functions as F
from pyspark.sql.window import Window

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()


# Define the schema for the Ads table
schema=StructType([
        StructField('AD_ID',IntegerType(),True)
        ,StructField('USER_ID',IntegerType(),True)
        ,StructField('ACTION',StringType(),True)
])
```

```
# Define the data for the Ads table
data = [
        (1, 1, 'Clicked'),
        (2, 2, 'Clicked'),
        (3, 3, 'Viewed'),
        (5, 5, 'Ignored'),
        (1, 7, 'Ignored'),
        (2, 7, 'Viewed'),
        (3, 5, 'Clicked'),
        (1, 4, 'Viewed'),
        (2, 11, 'Viewed'),
        (1, 2, 'Clicked')
]

# Create a PySpark DataFrame
df=spark.createDataFrame(data,schema)
df.show()
```

```
+-----+-------+-------+
|AD_ID|USER_ID| ACTION|
+-----+-------+-------+
|    1|      1|Clicked|
|    2|      2|Clicked|
|    3|      3| Viewed|
|    5|      5|Ignored|
|    1|      7|Ignored|
|    2|      7| Viewed|
|    3|      5|Clicked|
|    1|      4| Viewed|
|    2|     11| Viewed|
|    1|      2|Clicked|
+-----+-------+-------+
```

```
ctr_df = (
        ads_df.groupBy("ad_id")
        .agg(
        F.sum(F.when(ads_df["action"] == "Clicked",
1).otherwise(0)).alias("click_count"),
        F.sum(F.when(ads_df["action"] == "Viewed",
1).otherwise(0)).alias("view_count")
        )
        .withColumn("ctr", F.round(F.col("click_count") /
(F.col("click_count") + F.col("view_count")), 2))
)

# Order the result table by CTR in descending order and by ad_id in
ascending order
window_spec = Window.orderBy(F.col("ctr").desc(),
F.col("ad_id").asc())
result_df = ctr_df.withColumn("rank", F.rank().over(window_spec))

# Show the result DataFrame
result_df.select('ad_id','ctr').show()
```

```
+-----+----+
|ad_id| ctr|
+-----+----+
|    1|0.67|
|    3| 0.5|
|    5|null|
|    2|0.33|
+-----+----+
```

Step - 1 : Problem Statement

Combine Two DF

Write a Pyspark program to report the first name, last name, city, and state of each person in the Person dataframe. If the address of a personId is not present in the Address dataframe, report null instead.

Difficult Level : EASY

DataFrame:

```
# Define schema for the 'persons' table
persons_schema = StructType([
        StructField("personId", IntegerType(), True),
        StructField("lastName", StringType(), True),
        StructField("firstName", StringType(), True)
])

# Define schema for the 'addresses' table
addresses_schema = StructType([
        StructField("addressId", IntegerType(), True),
        StructField("personId", IntegerType(), True),
        StructField("city", StringType(), True),
        StructField("state", StringType(), True)
])

# Define data for the 'persons' table
persons_data = [
        (1, 'Wang', 'Allen'),
        (2, 'Alice', 'Bob')
]

# Define data for the 'addresses' table
addresses_data = [
        (1, 2, 'New York City', 'New York'),
        (2, 3, 'Leetcode', 'California')
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT-1 persons		
PERSONID	LASTNAME	FIRSTNAME
1	Wang	Allen
2	Alice	Bob

INPUT-2 addresses			
ADDRESSID	PERSONID	CITY	STATE
1	2	New York City	New York
2	3	Leetcode	California

OUTPUT

OUTPUT			
FIRSTNAME	LASTNAME	CITY	STATE
Bob	Alice	New York City	New York
Allen	Wang		

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session

from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType
from pyspark.sql.functions import when
from pyspark.sql import functions as F
from pyspark.sql.window import Window

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
# Define schema for the 'persons' table
persons_schema = StructType([
        StructField("personId", IntegerType(), True),
        StructField("lastName", StringType(), True),
        StructField("firstName", StringType(), True)
])

# Define schema for the 'addresses' table
addresses_schema = StructType([
        StructField("addressId", IntegerType(), True),
        StructField("personId", IntegerType(), True),
        StructField("city", StringType(), True),
        StructField("state", StringType(), True)
])
```

```
# Define data for the 'persons' table
persons_data = [
        (1, 'Wang', 'Allen'),
        (2, 'Alice', 'Bob')
]

# Define data for the 'addresses' table
addresses_data = [
        (1, 2, 'New York City', 'New York'),
        (2, 3, 'Leetcode', 'California')
]
```

```
# Create a PySpark DataFrame
person_df=spark.createDataFrame(persons_data,persons_schema)
address_df=spark.createDataFrame(addresses_data,addresses_schema)

person_df.show()
address_df.show()
```

```
+--------+--------+---------+
|personId|lastName|firstName|
+--------+--------+---------+
|       1|    Wang|    Allen|
|       2|   Alice|      Bob|
+--------+--------+---------+

+---------+--------+-------------+----------+
|addressId|personId|         city|     state|
+---------+--------+-------------+----------+
|        1|       2|New York City|  New York|
|        2|       3|     Leetcode|California|
+---------+--------+-------------+----------+
```

```
person_df.join(address_df,person_df.personId==address_df.personId,'left')\
        .select('firstName','lastName','city','state')\
        .show()

# Show the result DataFrame
```

```
+---------+--------+-------------+--------+
|firstName|lastName|         city|   state|
+---------+--------+-------------+--------+
|    Allen|    Wang|         null|    null|
|      Bob|   Alice|New York City|New York|
+---------+--------+-------------+--------+
```

Step - 1 : Problem Statement

04_Employees Earning More Than Their Managers

Write a Pyspark program to find Employees Earning More Than Their Managers

Difficult Level : EASY

DataFrame:

```
# Define the schema for the "employees"
employees_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("name", StringType(), True),
        StructField("salary", IntegerType(), True),
        StructField("managerId", IntegerType(), True)
])

# Define data for the "employees"
employees_data = [
        (1, 'Joe', 70000, 3),
        (2, 'Henry', 80000, 4),
        (3, 'Sam', 60000, None),
        (4, 'Max', 90000, None)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT			
ID	NAME	SALARY	MANAGERID
1	Joe	70,000	3
2	Henry	80,000	4
3	Sam	60,000	
4	Max	90,000	

OUTPUT

OUTPUT
NAME
Joe

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session

from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType
from pyspark.sql.functions import when
from pyspark.sql import functions as F
from pyspark.sql.window import Window

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
# Define the schema for the "employees"
employees_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("name", StringType(), True),
        StructField("salary", IntegerType(), True),
        StructField("managerId", IntegerType(), True)
])

# Define data for the "employees"
employees_data = [
        (1, 'Joe', 70000, 3),
        (2, 'Henry', 80000, 4),
        (3, 'Sam', 60000, None),
        (4, 'Max', 90000, None)
]
```

```
# Create a PySpark DataFrame
emp_df=spark.createDataFrame(employees_data,employees_schema)
emp_df.show()


 emp_df1 = emp_df.alias("e1")
 emp_df2 = emp_df.alias("e2")

 self_joined_df = emp_df1.join(emp_df2, col("e1.id") ==
 col("e2.managerId"),
 "inner")            .select(col("e2.name"),col("e2.salary"),col("e1.salary").alias("msal"))


self_joined_df.filter(self_joined_df.salary>self_joined_df.msal).select("name").show()
```

```
+----+
|name|
+----+
| Joe|
+----+
```

Step - 1 : Problem Statement

Duplicate Emails

Write a Pyspark program to report all the duplicate emails.
Note that it's guaranteed that the email field is not NULL.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the "employees"
employees_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("name", StringType(), True),
        StructField("salary", IntegerType(), True),
        StructField("managerId", IntegerType(), True)
])

# Define data for the "employees"
employees_data = [
        (1, 'Joe', 70000, 3),
        (2, 'Henry', 80000, 4),
        (3, 'Sam', 60000, None),
        (4, 'Max', 90000, None)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT	
ID	EMAIL
1	a@b.com
2	c@d.com
3	a@b.com

OUTPUT

OUTPUT
EMAIL
a@b.com

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session

from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType
from pyspark.sql.functions import when
from pyspark.sql import functions as F
from pyspark.sql.window import Window

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Define the schema for the "emails" table
emails_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("email", StringType(), True)
])

# Define data for the "emails" table
emails_data = [
        (1, 'a@b.com'),
        (2, 'c@d.com'),
        (3, 'a@b.com')
]
```

```
# Create a PySpark DataFrame
df=spark.createDataFrame(emails_data,emails_schema)
df.show()


df_group=df.groupby("email").count()
df_group.filter(df_group["count"] > 1).show()
```

```
+-------+-----+
|  email|count|
+-------+-----+
|a@b.com|    2|
+-------+-----+
```

Step - 1 : Problem Statement

06_Customers Who Never Order

Write a Pyspark program to find all customers who never order anything.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the "Customers"
customers_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("name", StringType(), True)
])

# Define data for the "Customers"
customers_data = [
        (1, 'Joe'),
        (2, 'Henry'),
        (3, 'Sam'),
        (4, 'Max')
]

# Define the schema for the "Orders"
orders_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("customerId", IntegerType(), True)
])

# Define data for the "Orders"
orders_data = [
        (1, 3),
        (2, 1)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT -1 customers	
ID	NAME
1	Joe
2	Henry
3	Sam
4	Max

INPUT - 2 orders	
ID	CUSTOMERID
1	3
2	1

OUTPUT

OUTPUT
NAME
Max
Henry

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

customers_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("name", StringType(), True)
])

# Define data for the "Customers"
customers_data = [
        (1, 'Joe'),
        (2, 'Henry'),
        (3, 'Sam'),
        (4, 'Max')
]

# Define the schema for the "Orders"
orders_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("customerId", IntegerType(), True)
])
```

```
# Define data for the "Orders"
orders_data = [
        (1, 3),
        (2, 1)
]


# Create a PySpark DataFrame
cus_df=spark.createDataFrame(customers_data,customers_schem
a)
ord_df=spark.createDataFrame(orders_data,orders_schema)

cus_df.show()
ord_df.show()
```

```
+---+------+
| id|  name|
+---+------+
|  1|   Joe|
|  2|Henry|
|  3|   Sam|
|  4|   Max|
+---+------+
```

```
+---+----------+
| id|customerId|
+---+----------+
|  1|         3|
|  2|         1|
+---+----------+
```

```
cus_df.join(ord_df,cus_df.id == ord_df.customerId,"left_anti")\
        .select("name").show()
```

```
+-----+
| name|
+-----+
|  Max|
|Henry|
+-----+
```

Step - 1 : Problem Statement

07_Rising Temperature

Write a solution to find all dates' Id with higher temperatures compared to its previous dates (yesterday).

Return the result table in any order.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the "Weather" table
weather_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("recordDate", StringType(), True),
        StructField("temperature", IntegerType(), True)
])

# Define data for the "Weather" table
weather_data = [
        (1, '2015-01-01', 10),
        (2, '2015-01-02', 25),
        (3, '2015-01-03', 20),
        (4, '2015-01-04', 30)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT		
ID	RECORDDATE	TEMPERATURE
1	2015-01-01	10
2	2015-01-02	25
3	2015-01-03	20
4	2015-01-04	30

OUTPUT

OUTPUT
ID
2
4

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
```

```
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()


# Define the schema for the "Weather" table
weather_schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("recordDate", StringType(), True),
        StructField("temperature", IntegerType(), True)
])

# Define data for the "Weather" table
weather_data = [
        (1, '2015-01-01', 10),
        (2, '2015-01-02', 25),
        (3, '2015-01-03', 20),
        (4, '2015-01-04', 30)
]


# Create a PySpark DataFrame

temp_df=spark.createDataFrame(weather_data,weather_schema)
temp_df.show()
```

```
+---+----------+-----------+
| id|recordDate|temperature|
+---+----------+-----------+
|  1|2015-01-01|         10|
|  2|2015-01-02|         25|
|  3|2015-01-03|         20|
|  4|2015-01-04|         30|
+---+----------+-----------+
```

```
lag_df=temp_df.withColumn("prev_day",lag(temp_df.temperature).
over(Window.orderBy(temp_df.recordDate)))
lag_df.show()
```

```
+---+----------+-----------+--------+
| id|recordDate|temperature|prev_day|
+---+----------+-----------+--------+
|  1|2015-01-01|         10|    null|
|  2|2015-01-02|         25|      10|
|  3|2015-01-03|         20|      25|
|  4|2015-01-04|         30|      20|
+---+----------+-----------+--------+
```

```
lag_df.filter(lag_df["temperature"] >
lag_df["prev_day"] ).select("id").show()
```

```
+---+
| id|
+---+
|  2|
|  4|
+---+
```

Step - 1 : Problem Statement

08_Game Play Analysis I

Write a solution to find the first login date for each player.

Return the result table in any order.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the "Activity"
activity_schema = StructType([
        StructField("player_id", IntegerType(), True),
        StructField("device_id", IntegerType(), True),
        StructField("event_date", StringType(), True),
        StructField("games_played", IntegerType(), True)
])

# Define data for the "Activity"
activity_data = [
        (1, 2, '2016-03-01', 5),
        (1, 2, '2016-05-02', 6),
        (2, 3, '2017-06-25', 1),
        (3, 1, '2016-03-02', 0),
        (3, 4, '2018-07-03', 5)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT			
PLAYER_ID	DEVICE_ID	EVENT_DATE	GAMES_PLAYED
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

OUTPUT

OUTPUT	
PLAYER_ID	FISRT_LOGIN
1	2016-03-01
2	2017-06-25
3	2016-03-02

Step - 3 : Writing the pyspark code to solve

PYSPARK LEARNING HUB : DAY - 8

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Define the schema for the "Activity"
activity_schema = StructType([
        StructField("player_id", IntegerType(), True),
        StructField("device_id", IntegerType(), True),
        StructField("event_date", StringType(), True),
        StructField("games_played", IntegerType(), True)
])

# Define data for the "Activity"
activity_data = [
        (1, 2, '2016-03-01', 5),
        (1, 2, '2016-05-02', 6),
        (2, 3, '2017-06-25', 1),
        (3, 1, '2016-03-02', 0),
        (3, 4, '2018-07-03', 5)
]



# Create a PySpark DataFrame

activity_df=spark.createDataFrame(activity_data,activity_schema)
activity_df.show()
```

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

```
rank_df=activity_df.withColumn("RK",rank().over(Window.partitionBy(activity_df['player_id']).orderBy(activity_df['event_date'])))
rank_df.show()
```

id	recordDate	temperature	prev_day
1	2015-01-01	10	null
2	2015-01-02	25	10
3	2015-01-03	20	25
4	2015-01-04	30	20

```
rank_df.filter(rank_df["RK"] ==
1).select("player_id",rank_df["event_date"].alias("First_Login")).sh
ow()
```

```
+---------+-----------+
|player_id|First_Login|
+---------+-----------+
|        1| 2016-03-01|
|        3| 2016-03-02|
|        2| 2017-06-25|
+---------+-----------+
```

Step - 1 : Problem Statement

09_Game Play Analysis II

Write a pyspark code that reports the device that is first logged in for each player.

Return the result table in any order.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the "Activity"
activity_schema = StructType([
        StructField("player_id", IntegerType(), True),
        StructField("device_id", IntegerType(), True),
        StructField("event_date", StringType(), True),
        StructField("games_played", IntegerType(), True)
])

# Define data for the "Activity"
activity_data = [
        (1, 2, '2016-03-01', 5),
        (1, 2, '2016-05-02', 6),
        (2, 3, '2017-06-25', 1),
        (3, 1, '2016-03-02', 0),
        (3, 4, '2018-07-03', 5)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT			
PLAYER_ID	DEVICE_ID	EVENT_DATE	GAMES_PLAYED
1	2	2016-03-01	5
1	2	2016-05-02	6
2	3	2017-06-25	1
3	1	2016-03-02	0
3	4	2018-07-03	5

OUTPUT

OUTPUT	
PLAYER_ID	DEVICE_ID
1	2
2	3
3	1

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Define the schema for the "Activity"
activity_schema = StructType([
        StructField("player_id", IntegerType(), True),
        StructField("device_id", IntegerType(), True),
        StructField("event_date", StringType(), True),
        StructField("games_played", IntegerType(), True)
])

# Define data for the "Activity"
activity_data = [
        (1, 2, '2016-03-01', 5),
        (1, 2, '2016-05-02', 6),
        (2, 3, '2017-06-25', 1),
        (3, 1, '2016-03-02', 0),
        (3, 4, '2018-07-03', 5)
]
```

```
# Create a PySpark DataFrame
df=spark.createDataFrame(activity_data,activity_schema)
df.show()
```

```
+---------+---------+----------+------------+
|player_id|device_id|event_date|games_played|
+---------+---------+----------+------------+
|        1|        2|2016-03-01|           5|
|        1|        2|2016-05-02|           6|
|        2|        3|2017-06-25|           1|
|        3|        1|2016-03-02|           0|
|        3|        4|2018-07-03|           5|
+---------+---------+----------+------------+
```

```
rank_df=df.withColumn("rk",rank().over(Window.partitionBy(df["player_id"]).orderBy(df["event_date"])))
rank_df.show()
```

```
+---------+---------+----------+------------+---+
|player_id|device_id|event_date|games_played| rk|
+---------+---------+----------+------------+---+
|        1|        2|2016-03-01|           5|  1|
|        1|        2|2016-05-02|           6|  2|
|        3|        1|2016-03-02|           0|  1|
|        3|        4|2018-07-03|           5|  2|
|        2|        3|2017-06-25|           1|  1|
+---------+---------+----------+------------+---+
```

```
rank_df.filter(rank_df["rk"] ==
1).select("player_id","device_id").show()
```

```
+---------+---------+
|player_id|device_id|
+---------+---------+
|        1|        2|
|        3|        1|
|        2|        3|
+---------+---------+
```

Step - 1 : Problem Statement

10_Employee Bonus

Write a solution to report the name and bonus amount of each employee with a bonus less than 1000.

Return the result table in any order

Difficult Level : EASY

DataFrame:

```
# Define the schema for the "Employee"
employee_schema = StructType([
        StructField("empId", IntegerType(), True),
        StructField("name", StringType(), True),
        StructField("supervisor", IntegerType(), True),
        StructField("salary", IntegerType(), True)
])

# Define data for the "Employee"
employee_data = [
        (3, 'Brad', None, 4000),
        (1, 'John', 3, 1000),
        (2, 'Dan', 3, 2000),
        (4, 'Thomas', 3, 4000)
]

# Define the schema for the "Bonus"
bonus_schema = StructType([
        StructField("empId", IntegerType(), True),
        StructField("bonus", IntegerType(), True)
])
```

```
# Define data for the "Bonus"
bonus_data = [
        (2, 500),
        (4, 2000)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT-1 EMPLOYEE			
EMPID	NAME	SUPERVISOR	SALARY
3	Brad		4,000
1	John	3	1,000
2	Dan	3	2,000
4	Thomas	3	4,000

INPUT-2 BONUS	
EMPID	BONUS
2	500
4	2,000

OUTPUT

OUTPUT	
NAME	BONUS
Brad	
John	
Dan	500

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Define the schema for the "Employee"
employee_schema = StructType([
        StructField("empId", IntegerType(), True),
        StructField("name", StringType(), True),
        StructField("supervisor", IntegerType(), True),
        StructField("salary", IntegerType(), True)
])

# Define data for the "Employee"
employee_data = [
        (3, 'Brad', None, 4000),
        (1, 'John', 3, 1000),
        (2, 'Dan', 3, 2000),
        (4, 'Thomas', 3, 4000)
]


# Define the schema for the "Bonus"
```

```
bonus_schema = StructType([
        StructField("empId", IntegerType(), True),
        StructField("bonus", IntegerType(), True)
])

# Define data for the "Bonus"
bonus_data = [
        (2, 500),
        (4, 2000)
]

# Create a PySpark DataFrame

emp_df =
spark.createDataFrame(employee_data,employee_schema)
bonus_df = spark.createDataFrame(bonus_data,bonus_schema)
emp_df.show()
bonus_df.show()
```

```
+-----+------+----------+------+
|empId|  name|supervisor|salary|
+-----+------+----------+------+
|    3|  Brad|      null|  4000|
|    1|  John|         3|  1000|
|    2|   Dan|         3|  2000|
|    4|Thomas|         3|  4000|
+-----+------+----------+------+

+-----+-----+
|empId|bonus|
+-----+-----+
|    2|  500|
|    4| 2000|
+-----+-----+
```

```
join_df=emp_df.join(bonus_df,emp_df.empId==bonus_df.empId,"left")
join_df.show()
```

```
+-----+------+----------+------+-----+-----+
|empId|  name|supervisor|salary|empId|bonus|
+-----+------+----------+------+-----+-----+
|    1|  John|         3|  1000| null| null|
|    3|  Brad|      null|  4000| null| null|
|    4|Thomas|         3|  4000|    4| 2000|
|    2|   Dan|         3|  2000|    2|  500|
+-----+------+----------+------+-----+-----+
```

```
join_df.filter( (join_df.bonus < 1000) | col("bonus").isNull() ).show()
```

```
+-----+----+----------+------+-----+-----+
|empId|name|supervisor|salary|empId|bonus|
+-----+----+----------+------+-----+-----+
|    1|John|         3|  1000| null| null|
|    3|Brad|      null|  4000| null| null|
|    2| Dan|         3|  2000|    2|  500|
+-----+----+----------+------+-----+-----+
```

Step - 1 : Problem Statement

11_Find Customer Referee

Find the names of the customer that are not referred by the customer with id = 2.

Return the result table in any order

Difficult Level : EASY

DataFrame:

```
# Define the schema for the Customer table
schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("name", StringType(), True),
        StructField("referee_id", IntegerType(), True)
])

# Create an RDD with the data
data = [
        (1, 'Will', None),
        (2, 'Jane', None),
        (3, 'Alex', 2),
        (4, 'Bill', None),
        (5, 'Zack', 1),
        (6, 'Mark', 2)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT		
ID	NAME	REFEREE_ID
1	Will	
2	Jane	
3	Alex	2
4	Bill	
5	Zack	1
6	Mark	2

OUTPUT

OUTPUT
NAME
Will
Jane
Bill
Zack

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Define the schema for the Customer table
schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("name", StringType(), True),
        StructField("referee_id", IntegerType(), True)
])

# Create an RDD with the data
data = [
        (1, 'Will', None),
        (2, 'Jane', None),
        (3, 'Alex', 2),
        (4, 'Bill', None),
        (5, 'Zack', 1),
        (6, 'Mark', 2)
]
```

```
# Create a PySpark DataFrame
customer_df = spark.createDataFrame(data ,schema )
```

```
# Filter customers not referred by customer with id = 2
result_df = customer_df.filter((col("referee_id").isNull()) |
(col("referee_id") != 2))

# Select only the 'name' column
result_df = result_df.select("name")
```

```
+-----+
| name|
+-----+
| Will|
| Jane|
| Bill|
| Zack|
+-----+
```

Step - 1 : Problem Statement

12_Cities With Completed Trades

Write a pypsark code to retrieve the top three cities that have the highest number of completed trade orders listed in descending order. Output the city name and the corresponding number of completed trade orders.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the trades
trades_schema = StructType([
    StructField("order_id", IntegerType(), True),
    StructField("user_id", IntegerType(), True),
    StructField("price", FloatType(), True),
    StructField("quantity", IntegerType(), True),
    StructField("status", StringType(), True),
    StructField("timestamp", StringType(), True)
])

# Define the schema for the users
users_schema = StructType([
    StructField("user_id", IntegerType(), True),
    StructField("city", StringType(), True),
    StructField("email", StringType(), True),
    StructField("signup_date", StringType(), True)
])

# Create an RDD with the data for trades
trades_data = [
    (100101, 111, 9.80, 10, 'Cancelled', '2022-08-17 12:00:00'),
    (100102, 111, 10.00, 10, 'Completed', '2022-08-17 12:00:00'),
    (100259, 148, 5.10, 35, 'Completed', '2022-08-25 12:00:00'),
    (100264, 148, 4.80, 40, 'Completed', '2022-08-26 12:00:00'),
    (100305, 300, 10.00, 15, 'Completed', '2022-09-05 12:00:00'),
    (100400, 178, 9.90, 15, 'Completed', '2022-09-09 12:00:00'),
    (100565, 265, 25.60, 5, 'Completed', '2022-12-19 12:00:00')
]
```

```
# Create an RDD with the data for users
users_data = [
    (111, 'San Francisco', 'rrok10@gmail.com', '2021-08-03 12:00:00'),
    (148, 'Boston', 'sailor9820@gmail.com', '2021-08-20 12:00:00'),
    (178, 'San Francisco', 'harrypotterfan182@gmail.com', '2022-01-05
12:00:00'),
    (265, 'Denver', 'shadower_@hotmail.com', '2022-02-26 12:00:00'),
    (300, 'San Francisco', 'houstoncowboy1122@hotmail.com', '2022-06-30
12:00:00')
]
```

Step - 2 : Identifying The Input Data And Expected

INPUT

INPUT-1 trade					
ORDER_ID	USER_ID	PRICE	QUANTITY	STATUS	TIMESTAMP
100101	111	9.8	10	Cancelled	2022-08-17 12:00:00
100102	111	10	10	Completed	2022-08-17 12:00:00
100259	148	5.1	35	Completed	2022-08-25 12:00:00
100264	148	4.8	40	Completed	2022-08-26 12:00:00
100305	300	10	15	Completed	2022-09-05 12:00:00
100400	178	9.9	15	Completed	2022-09-09 12:00:00
100565	265	25.6	5	Completed	2022-12-19 12:00:00

INPUT - 2 user			
USER_ID	CITY	EMAIL	SIGNUP_DATE
111	San Francisco	rrok10@gmail.com	2021-08-03 12:00:00
148	Boston	sailor9820@gmail.com	2021-08-20 12:00:00
178	San Francisco	harrypotterfan182@gmail.com	2022-01-05 12:00:00
265	Denver	shadower_@hotmail.com	2022-02-26 12:00:00
300	San Francisco	houstoncowboy1122@hotmail.com	2022-06-30 12:00:00

OUTPUT

OUTPUT	
CITY	COUNT()
San Francisco	3

Boston	2
Denver	1

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Define the schema for the trades
trades_schema = StructType([
        StructField("order_id", IntegerType(), True),
        StructField("user_id", IntegerType(), True),
        StructField("price", FloatType(), True),
        StructField("quantity", IntegerType(), True),
        StructField("status", StringType(), True),
        StructField("timestamp", StringType(), True)
])
# Define the schema for the users
users_schema = StructType([
        StructField("user_id", IntegerType(), True),
```

```
        StructField("city", StringType(), True),
        StructField("email", StringType(), True),
        StructField("signup_date", StringType(), True)
])



# Create an RDD with the data for trades
trades_data = [
        (100101, 111, 9.80, 10, 'Cancelled', '2022-08-17 12:00:00'),
        (100102, 111, 10.00, 10, 'Completed', '2022-08-17 12:00:00'),
        (100259, 148, 5.10, 35, 'Completed', '2022-08-25 12:00:00'),
        (100264, 148, 4.80, 40, 'Completed', '2022-08-26 12:00:00'),
        (100305, 300, 10.00, 15, 'Completed', '2022-09-05 12:00:00'),
        (100400, 178, 9.90, 15, 'Completed', '2022-09-09 12:00:00'),
        (100565, 265, 25.60, 5, 'Completed', '2022-12-19 12:00:00')
]

# Create an RDD with the data for users
users_data = [
        (111, 'San Francisco', 'rrok10@gmail.com', '2021-08-03
12:00:00'),
        (148, 'Boston', 'sailor9820@gmail.com', '2021-08-20 12:00:00'),
        (178, 'San Francisco', 'harrypotterfan182@gmail.com', '2022-
01-05 12:00:00'),
        (265, 'Denver', 'shadower_@hotmail.com', '2022-02-26
12:00:00'),
        (300, 'San Francisco', 'houstoncowboy1122@hotmail.com',
'2022-06-30 12:00:00')
]


Trade_df=spark.createDataFrame(trades_data,trades_schema)
User_df=spark.createDataFrame(users_data,users_schema)
Trade_df.show()
User_df.show()
```

order_id	user_id	price	quantity	status	timestamp
100101	111	9.8	10	Cancelled	2022-08-17 12:00:00
100102	111	10.0	10	Completed	2022-08-17 12:00:00
100259	148	5.1	35	Completed	2022-08-25 12:00:00
100264	148	4.8	40	Completed	2022-08-26 12:00:00
100305	300	10.0	15	Completed	2022-09-05 12:00:00
100400	178	9.9	15	Completed	2022-09-09 12:00:00
100565	265	25.6	5	Completed	2022-12-19 12:00:00

user_id	city	email	signup_date
111	San Francisco	rrok10@gmail.com	2021-08-03 12:00:00
148	Boston	sailor9820@gmail.com	2021-08-20 12:00:00
178	San Francisco	harrypotterfan182...	2022-01-05 12:00:00
265	Denver	shadower_@hotmail...	2022-02-26 12:00:00
300	San Francisco	houstoncowboy1122...	2022-06-30 12:00:00

```
join_df=Trade_df.join(User_df,Trade_df['user_id']==User_df['user_id'],"inner")
join_df.show()
```

order_id	user_id	price	quantity	status	timestamp	user_id	city	email	signup_date
100259	148	5.1	35	Completed	2022-08-25 12:00:00	148	Boston	sailor9820@gmail.com	2021-08-20 12:00:00
100264	148	4.8	40	Completed	2022-08-26 12:00:00	148	Boston	sailor9820@gmail.com	2021-08-20 12:00:00
100305	300	10.0	15	Completed	2022-09-05 12:00:00	300	San Francisco	houstoncowboy1122...	2022-06-30 12:00:00
100101	111	9.8	10	Cancelled	2022-08-17 12:00:00	111	San Francisco	rrok10@gmail.com	2021-08-03 12:00:00
100102	111	10.0	10	Completed	2022-08-17 12:00:00	111	San Francisco	rrok10@gmail.com	2021-08-03 12:00:00
100400	178	9.9	15	Completed	2022-09-09 12:00:00	178	San Francisco	harrypotterfan182...	2022-01-05 12:00:00
100565	265	25.6	5	Completed	2022-12-19 12:00:00	265	Denver	shadower_@hotmail...	2022-02-26 12:00:00

```
join_df.filter(join_df['status'] == 'Completed').groupby(join_df['city']).count()
```

```
join_df.filter(join_df['status'] == 'Completed').groupby(join_df['city']).count()
```

city	count
San Francisco	3
Denver	1
Boston	2

Step - 1 : Problem Statement

13_Page With No Likes

Write a pyspark code to return the IDs of the Facebook pages that have zero likes. The output should be sorted in ascending order based on the page IDs.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the pages
pages_schema = StructType([
        StructField("page_id", IntegerType(), True),
        StructField("page_name", StringType(), True)
])
# Define the schema for the page_likes table
page_likes_schema = StructType([
        StructField("user_id", IntegerType(), True),
        StructField("page_id", IntegerType(), True),
        StructField("liked_date", StringType(), True)
])
# Create an RDD with the data for pages
pages_data = [
        (20001, 'SQL Solutions'),
        (20045, 'Brain Exercises'),
        (20701, 'Tips for Data Analysts')
]
# Create an RDD with the data for page_likes table
page_likes_data = [
        (111, 20001, '2022-04-08 00:00:00'),
        (121, 20045, '2022-03-12 00:00:00'),
        (156, 20001, '2022-07-25 00:00:00')
]
```

Step - 2 : Identifying The Input Data And Expected

INPUT

INPUT - 1 PAGES	
PAGE_ID	PAGE_NAME
20001	SQL Solutions
20045	Brain Exercises
20701	Tips for Data Analysts

INPUT - 2 PAGES_LIEKS		
USER_ID	PAGE_ID	LIKED_DATE
111	20001	2022-04-08 0:00:00
121	20045	2022-03-12 0:00:00
156	20001	2022-07-25 0:00:00

OUTPUT

OUTPUT
PAGE_ID
20701

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Define the schema for the pages
pages_schema = StructType([
      StructField("page_id", IntegerType(), True),
      StructField("page_name", StringType(), True)
])

# Define the schema for the page_likes table
page_likes_schema = StructType([
      StructField("user_id", IntegerType(), True),
      StructField("page_id", IntegerType(), True),
      StructField("liked_date", StringType(), True)
])

# Create an RDD with the data for pages
pages_data = [
      (20001, 'SQL Solutions'),
      (20045, 'Brain Exercises'),
      (20701, 'Tips for Data Analysts')
]
# Create an RDD with the data for page_likes table
page_likes_data = [
      (111, 20001, '2022-04-08 00:00:00'),
      (121, 20045, '2022-03-12 00:00:00'),
```

```
    (156, 20001, '2022-07-25 00:00:00')
    ]
```

```
page_df=spark.createDataFrame(pages_data,pages_schema)
page_like_df=spark.createDataFrame(page_likes_data,page_likes_schema)
page_df.show()
page_like_df.show()
```

```
+-------+--------------------+
|page_id|           page_name|
+-------+--------------------+
|  20001|       SQL Solutions|
|  20045|     Brain Exercises|
|  20701|Tips for Data Ana...|
+-------+--------------------+

+-------+-------+-------------------+
|user_id|page_id|         liked_date|
+-------+-------+-------------------+
|    111|  20001|2022-04-08 00:00:00|
|    121|  20045|2022-03-12 00:00:00|
|    156|  20001|2022-07-25 00:00:00|
+-------+-------+-------------------+
```

```
# Perform a left anti join to get pages with zero likes
zero_likes_pages = page_df.join(page_like_df, 'page_id', 'left_anti')
# Select and sort the result
result = zero_likes_pages.select("page_id").orderBy("page_id")
# Show the result
result.show()
```

```
+-------+
|page_id|
+-------+
|  20701|
```

Step - 1 : Problem Statement

14_Purchasing Activity by Product Type

We have been given purchasing activity DF and we need to find out cumulative purchases of each product over time.

Difficult Level : EASY

DataFrame:

```
# Define schema for the DataFrame
schema = StructType([
        StructField("order_id", IntegerType(), True),
        StructField("product_type", StringType(), True),
        StructField("quantity", IntegerType(), True),
        StructField("order_date", StringType(), True),
])

# Define data
# Define data
data = [
        (213824, 'printer', 20, "2022-06-27 "),
        (212312, 'hair dryer', 5, "2022-06-28 "),
        (132842, 'printer', 18, "2022-06-28 "),
        (284730, 'standing lamp', 8, "2022-07-05 ")
]
```

Step - 2 : Identifying The Input Data And Expected

INPUT

INPUT			
ORDER_ID	PRODUCT_TYPE	QUANTITY	ORDER_DATE
213824	printer	20	2022-06-27 12:00:00
212312	hair dryer	5	2022-06-28 12:00:00
132842	printer	18	2022-06-28 12:00:00
284730	standing lamp	8	2022-07-05 12:00:00

OUTPUT

OUTPUT		
ORDER_DATE	PRODUCT_TYPE	CUM_PURCHASED
2022-06-27 12:00:00	printer	20
2022-06-28 12:00:00	hair dryer	5
2022-06-28 12:00:00	printer	38
2022-07-05 12:00:00	standing lamp	8

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Define schema for the DataFrame
schema = StructType([
        StructField("order_id", IntegerType(), True),
        StructField("product_type", StringType(), True),
        StructField("quantity", IntegerType(), True),
        StructField("order_date", StringType(), True),
])

# Define data
# Define data
data = [
        (213824, 'printer', 20, "2022-06-27 "),
        (212312, 'hair dryer', 5, "2022-06-28 "),
        (132842, 'printer', 18, "2022-06-28 "),
        (284730, 'standing lamp', 8, "2022-07-05 ")
]
```

```
order_df=spark.createDataFrame(data,schema)
order_df.show()
```

```
+--------+-------------+--------+-----------+
|order_id| product_type|quantity| order_date|
+--------+-------------+--------+-----------+
|  213824|      printer|      20|2022-06-27 |
|  212312|   hair dryer|       5|2022-06-28 |
|  132842|      printer|      18|2022-06-28 |
|  284730|standing lamp|       8|2022-07-05 |
+--------+-------------+--------+-----------+
```

```
# Define a Window specification based on the 'order_date' column
window_spec =
Window.partitionBy("product_type").orderBy("order_date").rowsBe
tween(Window.unboundedPreceding, 0)

# Add a new column 'cumulative_purchases' representing the
cumulative sum
result_df = order_df.withColumn("cumulative_purchases",
F.sum("quantity").over(window_spec))
result_df.show()
```

```
+--------+-------------+--------+-----------+-------------------+
|order_id| product_type|quantity| order_date|cumulative_purchases|
+--------+-------------+--------+-----------+-------------------+
|  284730|standing lamp|       8|2022-07-05 |                  8|
|  212312|   hair dryer|       5|2022-06-28 |                  5|
|  213824|      printer|      20|2022-06-27 |                 20|
|  132842|      printer|      18|2022-06-28 |                 38|
+--------+-------------+--------+-----------+-------------------+
```

Step - 1 : Problem Statement

15_Teams Power Users

Write a pyspark code to identify the top 2 Power Users who sent the highest number of messages on Microsoft Teams in August 2022. Display the IDs of these 2 users along with the total number of messages they sent. Output the results in descending order based on the count of the messages.

Difficult Level : EASY

DataFrame:

```
schema = StructType([
        StructField("message_id", IntegerType(), True),
        StructField("sender_id", IntegerType(), True),
        StructField("receiver_id", IntegerType(), True),
        StructField("content", StringType(), True),
        StructField("sent_date", StringType(), True),
])

# Define the data
data = [
        (901, 3601, 4500, 'You up?', '2022-08-03 00:00:00'),
        (902, 4500, 3601, 'Only if you\'re buying', '2022-08-03 00:00:00'),
        (743, 3601, 8752, 'Let\'s take this offline', '2022-06-14 00:00:00'),
        (922, 3601, 4500, 'Get on the call', '2022-08-10 00:00:00'),
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT				
MESSAGE_ID	SENDER_ID	RECEIVER_ID	CONTENT	SENT_DATE
901	3601	4500	You up?	2022-08-03 0:00:00
902	4500	3601	Only if you're buying	2022-08-03 0:00:00
743	3601	8752	Let's take this offline	2022-06-14 0:00:00
922	3601	4500	Get on the call	2022-08-10 0:00:00

OUTPUT

OUTPUT	
SENDER_ID	COUNT(*)
3601	2
4500	1

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

schema = StructType([
        StructField("message_id", IntegerType(), True),
        StructField("sender_id", IntegerType(), True),
        StructField("receiver_id", IntegerType(), True),
        StructField("content", StringType(), True),
        StructField("sent_date", StringType(), True),
])

# Define the data
data = [
        (901, 3601, 4500, 'You up?', '2022-08-03 00:00:00'),
        (902, 4500, 3601, 'Only if you\'re buying', '2022-08-03 00:00:00'),
        (743, 3601, 8752, 'Let\'s take this offline', '2022-06-14 00:00:00'),
        (922, 3601, 4500, 'Get on the call', '2022-08-10 00:00:00'),
]
```

```
teams_df = spark.createDataFrame(data,schema)
teams_df.show()
```

```
+----------+---------+-----------+--------------------+-------------------+
|message_id|sender_id|receiver_id|             content|          sent_date|
+----------+---------+-----------+--------------------+-------------------+
|       901|     3601|       4500|            You up?|2022-08-03 00:00:00|
|       902|     4500|       3601|Only if you're bu...|2022-08-03 00:00:00|
|       743|     3601|       8752|Let's take this o...|2022-06-14 00:00:00|
|       922|     3601|       4500|    Get on the call|2022-08-10 00:00:00|
+----------+---------+-----------+--------------------+-------------------+
```

```
filter_df=teams_df.filter(teams_df['sent_date'].like("2022-08%"))
filter_df.show()
```

```
+----------+---------+-----------+--------------------+-------------------+
|message_id|sender_id|receiver_id|             content|          sent_date|
+----------+---------+-----------+--------------------+-------------------+
|       901|     3601|       4500|            You up?|2022-08-03 00:00:00|
|       902|     4500|       3601|Only if you're bu...|2022-08-03 00:00:00|
|       922|     3601|       4500|    Get on the call|2022-08-10 00:00:00|
+----------+---------+-----------+--------------------+-------------------+
```

```
result_df=filter_df.groupby(filter_df['sender_id']).count()
result_df=result_df.orderBy(desc(result_df['count'])).limit(2)
result_df.show()
```

```
+---------+-----+
|sender_id|count|
+---------+-----+
|     3601|    2|
|     4500|    1|
+---------+-----+
```

Step - 1 : Problem Statement

16_Select in pyspark

Write a pyspark code perform below function

- Write a pyspark code to get all employee detail.
- Write a query to get only "FirstName" column from emp_df
- Write a Pyspark code to get FirstName in upper case as "First Name".
- Write a pyspark code to get FirstName in lower case

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT",
"Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT",
"Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR",
"Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll",
"Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290",
"IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR",
"Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793",
"IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793",
"HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793",
"Payroll", "Male"],
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
```

```
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

emp_df=spark.createDataFrame(data,schema)
```

#1. Write a pyspark code to get all employee detail

```
emp_df.show()
```

```
+----------+----------+---------+---------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+-------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+-------------------+----------+------+
```

 # 2. Write a query to get only "FirstName" column from emp_df

```
# Method 1
emp_df.select("First_Name").show()

# Method 2
emp_df.select(col("First_Name")).show()

# Method 3
emp_df.createOrReplaceTempView("emp_table")
spark.sql("select First_Name from emp_table").show()
```

```
+----------+
|First_Name|
+----------+
|     Vikas|
|    nikita|
|    Ashish|
|    Nikhil|
|     anish|
+----------+

+----------+
|First_Name|
+----------+
|     Vikas|
|    nikita|
|    Ashish|
|    Nikhil|
|     anish|
+----------+

+----------+
|First_Name|
+----------+
|     Vikas|
|    nikita|
|    Ashish|
|    Nikhil|
|     anish|
+----------+
```

 # 3. Write a Pyspark code to get FirstName in upper case as "First Name".

```
emp_df.select(upper("First_Name")).show()
```

```
+-----------------+
|upper(First_Name)|
+-----------------+
|            VIKAS|
|           NIKITA|
|           ASHISH|
|           NIKHIL|
|            ANISH|
+-----------------+
```

#4. Write a pyspark code to get FirstName in lower case

```
from pyspark.sql.functions import lower
emp_df.select(lower("First_Name")).show()
```

```
+-----------------+
|lower(First_Name)|
+-----------------+
|            vikas|
|           nikita|
|           ashish|
|           nikhil|
|            anish|
+-----------------+
```

Step - 1 : Problem Statement

17_Select in pyspark

Write a pyspark code perform below function

- Write a pyspark code for combine FirstName and LastName and display it as "Name" (also include white space between first name & last name)
- Select employee detail whose name is "Vikas"
- Get all employee detail from EmployeeDetail table whose "FirstName" start with letter 'a'.

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT",
"Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT",
"Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR",
"Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll",
"Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
```

```
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290",
"IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR",
"Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793",
"IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793",
"HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793",
"Payroll", "Male"],
]
```

```
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

emp_df=spark.createDataFrame(data,schema)
```

#1. Write a pyspark code for combine FirstName and LastName and display it as "Name" (also include white space between first name & last name)

```
from pyspark.sql.functions import concat_ws
emp_df.select(concat_ws(" ","First_Name","Last_Name")\
        .alias("Name")).show()
```

```
+--------------+
|          Name|
+--------------+
|Vikas Ahlawat|
|  nikita Jain|
| Ashish Kumar|
|Nikhil Sharma|
| anish kadian|
+--------------+
```

2. Select employee detail whose name is "Vikas"

```
# Methos 1
from pyspark.sql.functions import col
emp_df.filter(col("First_Name") == 'Vikas' ).show(truncate=False)

# Methos 2
emp_df.filter(emp_df.First_Name == 'Vikas'  ).show(truncate=False)

# Methos 3
emp_df.filter(emp_df['First_Name'] == 'Vikas'  ).show(truncate=False)


# Methos 4
emp_df.where(emp_df['First_Name'] == 'Vikas'  ).show(truncate=False)
```

```
+----------+----------+---------+--------+-----------------------+----------+------+
|EmployeeID|First_Name|Last_Name|Salary  |Joining_Date           |Department|Gender|
+----------+----------+---------+--------+-----------------------+----------+------+
|1         |Vikas     |Ahlawat  |600000.0|2013-02-15 11:16:28.290|IT        |Male  |
+----------+----------+---------+--------+-----------------------+----------+------+

+----------+----------+---------+--------+-----------------------+----------+------+
|EmployeeID|First_Name|Last_Name|Salary  |Joining_Date           |Department|Gender|
+----------+----------+---------+--------+-----------------------+----------+------+
|1         |Vikas     |Ahlawat  |600000.0|2013-02-15 11:16:28.290|IT        |Male  |
+----------+----------+---------+--------+-----------------------+----------+------+

+----------+----------+---------+--------+-----------------------+----------+------+
|EmployeeID|First_Name|Last_Name|Salary  |Joining_Date           |Department|Gender|
+----------+----------+---------+--------+-----------------------+----------+------+
|1         |Vikas     |Ahlawat  |600000.0|2013-02-15 11:16:28.290|IT        |Male  |
+----------+----------+---------+--------+-----------------------+----------+------+

+----------+----------+---------+--------+-----------------------+----------+------+
|EmployeeID|First_Name|Last_Name|Salary  |Joining_Date           |Department|Gender|
+----------+----------+---------+--------+-----------------------+----------+------+
|1         |Vikas     |Ahlawat  |600000.0|2013-02-15 11:16:28.290|IT        |Male  |
+----------+----------+---------+--------+-----------------------+----------+------+
```

- Get all employee detail from EmployeeDetail table whose "FirstName" start with letter 'a'.

```
# Method 1
from pyspark.sql.functions import lower
emp_df.filter(lower(emp_df['First_Name']).like("a%")).show()

# Method 2

emp_df.filter((emp_df['First_Name'].like("a%")) |
(emp_df['First_Name'].like("A%")) ).show()
```

```
+----------+----------+---------+---------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+-------------------+----------+------+
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+-------------------+----------+------+

+----------+----------+---------+---------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+-------------------+----------+------+
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+-------------------+----------+------+
```

Step - 1 : Problem Statement

18_Select in pyspark

Write a pyspark code perform below function

- Get all employee details from EmployeeDetail table whose "FirstName" contains 'k'
- Get all employee details from EmployeeDetail table whose "FirstName" end with 'h'
- Get all employee detail from EmployeeDetail table whose "FirstName" start with
- Get all employee detail from EmployeeDetail table whose "FirstName" start with any single character between 'a-p'

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT",
"Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT",
"Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR",
"Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll",
"Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
```

```
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290",
"IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR",
"Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793",
"IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793",
"HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793",
"Payroll", "Male"],
```

```
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

emp_df=spark.createDataFrame(data,schema)
```

#1. Get all employee details from EmployeeDetail table whose "FirstName" contains 'k'

```
from pyspark.sql.functions import col
emp_df.filter(emp_df["First_Name"].like("%k%")).show(
)
```

```
+----------+----------+---------+--------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+--------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat|600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain|530000.0|2014-01-09 17:31:...|        HR|Female|
|         4|    Nikhil|   Sharma|480000.0|2014-01-09 09:00:...|        HR|  Male|
+----------+----------+---------+--------+--------------------+----------+------+
```

Get all employee details from EmployeeDetail table whose "FirstName" end with 'h'

```
emp_df.filter(emp_df["First_Name"].like("%h")).show()
```

```
+----------+----------+---------+---------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+-------------------+----------+------+
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+-------------------+----------+------+
```

 # Get all employee detail from EmployeeDetail table whose "FirstName" start with any single character between 'a-p'

```
emp_df.filter(emp_df["First_Name"].rlike("[^a-pA-P%]")).show()
```

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

 # Get all employee detail from EmployeeDetail table whose "FirstName" start with
 # any single character between 'a-p'

```
emp_df.filter(~(emp_df["First_Name"].rlike("[^a-pA-P%]"))).show()
```

```
+----------+----------+---------+--------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+--------+-------------------+----------+------+
|         4|    Nikhil|   Sharma|480000.0|2014-01-09 09:00:...|        HR|  Male|
+----------+----------+---------+--------+-------------------+----------+------+
```

Step - 1 : Problem Statement

19_Select in pyspark

Write a pyspark code perform below function

- Get all employee detail from emp_df whose "Gender" end with 'le' and contain 4 letters. The Underscore(_) Wildcard Character represents any single character.
- Get all employee detail from EmployeeDetail table whose "FirstName" start with # 'A' and contain 5 letters.
- Get all unique "Department" from EmployeeDetail table.
- Get the highest "Salary" from EmployeeDetail table.

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

emp_df=spark.createDataFrame(data,schema)

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

Get all employee detail from emp_df whose "Gender" end with 'le'and contain 4 letters. The Underscore(_) Wildcard Character represents any single character.

```
# Get all employee detail from emp_df  whose "Gender" end with 'le'
# and contain 4 letters. The Underscore(_) Wildcard Character represents any single
# character.

emp_df.filter(emp_df["Gender"].like("__le")).show()
```

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

 # Get all employee detail from EmployeeDetail table whose "FirstName" start with

 # 'A' and contain 5 letters.

```
# Get all employee detail from EmployeeDetail table whose "FirstName" start with
# 'A' and contain 5 letters.

emp_df.filter(emp_df["First_NamE"].like("a____")).show()
```

```
+----------+----------+---------+--------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+--------+--------------------+----------+------+
|         5|     anish|   kadian|500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+--------+--------------------+----------+------+
```

 # Get all unique "Department" from EmployeeDetail table.

```
# Get all unique "Department" from EmployeeDetail table.


emp_df.select("Department").distinct().show()
```

```
+-----------+
|Department|
+-----------+
|         HR|
|    Payroll|
|         IT|
+-----------+
```

 # Get the highest "Salary" from EmployeeDetail table.

```
# Get the highest "Salary" from EmployeeDetail table.


emp_df.agg(max("Salary")).show()
```

```
1000000.0
```

Step - 1 : Problem Statement

20_Date in pyspark

Write a pyspark code perform below function

- Get the lowest "Salary" from EmployeeDetail table.
- Show "JoiningDate" in "dd mmm yyyy" format, ex- "15 Feb 2013"
- Show "JoiningDate" in "yyyy/mm/dd" format, ex- "2013/02/15"

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

emp_df=spark.createDataFrame(data,schema)
```

PYSPARK LEARNING HUB : DAY - 20

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```
# 17). Get the lowest "Salary" from EmployeeDetail table.
emp_df.agg(min("Salary")).show()

#18)Show "JoiningDate" in "dd mmm yyyy" format, ex- "15 Feb 2013"

formatted_df = emp_df.withColumn("Joining_Date", date_format(to_timestamp(col("Joining_Date")), "dd MMM yyyy"))
formatted_df.show()

# 19)Show "JoiningDate" in "yyyy/mm/dd" format, ex- "2013/02/15"

formatted_df = emp_df.withColumn("Joining_Date", date_format(to_timestamp(col("Joining_Date")), "yyyy/mm/dd"))
formatted_df.show()
```

```
+-----------+
|min(Salary)|
+-----------+
|   480000.0|
+-----------+
```

```
+----------+----------+---------+---------+------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|Joining_Date|Department|Gender|
+----------+----------+---------+---------+------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0| 15 Feb 2013|        IT|  Male|
|         2|    nikita|     Jain| 530000.0| 09 Jan 2014|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0| 09 Jan 2014|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0| 09 Jan 2014|        HR|  Male|
|         5|     anish|   kadian| 500000.0| 09 Jan 2014|   Payroll|  Male|
+----------+----------+---------+---------+------------+----------+------+
```

```
+----------+----------+---------+---------+------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|Joining_Date|Department|Gender|
+----------+----------+---------+---------+------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|  2013/16/15|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|  2014/31/09|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|  2014/05/09|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|  2014/00/09|        HR|  Male|
|         5|     anish|   kadian| 500000.0|  2014/31/09|   Payroll|  Male|
+----------+----------+---------+---------+------------+----------+------+
```

Step - 1 : Problem Statement

21_Date in pyspark

Write a pyspark code perform below function

- Get only Year part of "JoiningDate"
- Get only Month part of "JoiningDate".
- Get only date part of "JoiningDate".
- Get the current system date using DataFrame API
- Get the current UTC date and time using DataFrame API

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

PYSPARK LEARNING HUB : DAY - 21

emp_df=spark.createDataFrame(data,schema)

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```python
# Get only Year part of "JoiningDate"
emp_df.select(date_format(to_timestamp(col("joining_date")),"yyyy")).show(truncate=False)

# Get only Month part of "JoiningDate".
emp_df.select(date_format(to_timestamp(col("joining_date")),"MM")).show(truncate=False)
emp_df.select(date_format(to_timestamp(col("joining_date")),"MMM")).show(truncate=False)

# Get only date part of "JoiningDate".
emp_df.select(date_format(to_timestamp(col("joining_date")),"dd")).show(truncate=False)

# Get the current system date using DataFrame API
system_date_df = spark.range(1).select(current_date().alias("system_date"))
system_date_df.show(truncate=False)

# Get the current UTC date and time using DataFrame API
utc_date_time_df = spark.range(1).select(current_timestamp().alias("utc_date_time"))
utc_date_time_df.show(truncate=False)
```

```
+-----------------------------------------------+
|date_format(to_timestamp(joining_date), yyyy)|
+-----------------------------------------------+
|2013                                           |
|2014                                           |
|2014                                           |
|2014                                           |
|2014                                           |
+-----------------------------------------------+

+-----------------------------------------------+
|date_format(to_timestamp(joining_date), MM)|
+-----------------------------------------------+
|02                                             |
|01                                             |
|01                                             |
|01                                             |
|01                                             |
+-----------------------------------------------+

+-----------------------------------------------+
|date_format(to_timestamp(joining_date), MMM)|
+-----------------------------------------------+
|Feb                                            |
|Jan                                            |
|Jan                                            |
|Jan                                            |
|Jan                                            |
+-----------------------------------------------+
```

```
+-------------------------------------------+
|date_format(to_timestamp(joining_date), dd)|
+-------------------------------------------+
|15                                         |
|09                                         |
|09                                         |
|09                                         |
|09                                         |
+-------------------------------------------+

+-----------+
|system_date|
+-----------+
|2024-01-09 |
+-----------+

+-----------------------+
|utc_date_time          |
+-----------------------+
|2024-01-09 21:14:36.535|
+-----------------------+
```

Step - 1 : Problem Statement

22_Date in pyspark

Write a pyspark code perform below function

- Get the first name, current date, joiningdate and diff between current date and joining date in months.
- Get the first name, current date, joiningdate and diff between current date and joining date in days.
- Get all employee details from EmployeeDetail table whose joining year is 2013

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve the

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

PYSPARK LEARNING HUB : DAY - 22

emp_df=spark.createDataFrame(data,schema)

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```python
# 25). Get the first name, current date, joiningdate and diff between current date and
# joining date in months.

from pyspark.sql.functions import months_between
emp_df.select("First_Name"\
              ,"Joining_Date"\
              ,current_date()\
              ,months_between(current_date(),col("Joining_Date")).alias("Total month"))\
             .show(truncate=False)
```

```
+----------+-----------------------+--------------+------------+
|First_Name|Joining_Date           |current_date()|Total month |
+----------+-----------------------+--------------+------------+
|Vikas     |2013-02-15 11:16:28.290|2024-01-10    |130.82355585|
|nikita    |2014-01-09 17:31:07.793|2024-01-10    |120.00871154|
|Ashish    |2014-01-09 10:05:07.793|2024-01-10    |120.01870258|
|Nikhil    |2014-01-09 09:00:07.793|2024-01-10    |120.02015868|
|anish     |2014-01-09 09:31:07.793|2024-01-10    |120.01946423|
+----------+-----------------------+--------------+------------+
```

PYSPARK LEARNING HUB : DAY - 22

```
# 26). Get the first name, current date, joiningdate and diff between current date
and
# joining date in days.

from pyspark.sql.functions import datediff
emp_df.select("First_Name"\
              ,"Joining_Date"\
              ,current_date()\
              ,datediff(current_date(),col("Joining_Date")).alias("Totsl days"))\
             .show(truncate=False)
```

```
+----------+-----------------------+--------------+----------+
|First_Name|Joining_Date           |current_date()|Totsl days|
+----------+-----------------------+--------------+----------+
|Vikas     |2013-02-15 11:16:28.290|2024-01-10    |3981      |
|nikita    |2014-01-09 17:31:07.793|2024-01-10    |3653      |
|Ashish    |2014-01-09 10:05:07.793|2024-01-10    |3653      |
|Nikhil    |2014-01-09 09:00:07.793|2024-01-10    |3653      |
|anish     |2014-01-09 09:31:07.793|2024-01-10    |3653      |
+----------+-----------------------+--------------+----------+
```

PYSPARK LEARNING HUB : DAY - 22

```
# 27). Get all employee details from EmployeeDetail table
whose joining year is 2013
from pyspark.sql.functions import year

#method 1
emp_df.filter(col("Joining_Date").like("2013%") ).show()

#method 2
emp_df.filter(year("Joining_Date")==2013).show()
```

```
+----------+----------+---------+--------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+--------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat|600000.0|2013-02-15 11:16:...|        IT|  Male|
+----------+----------+---------+--------+--------------------+----------+------+
```


```
+----------+----------+---------+--------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+--------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat|600000.0|2013-02-15 11:16:...|        IT|  Male|
+----------+----------+---------+--------+--------------------+----------+------+
```

Step - 1 : Problem Statement

23_Date in pyspark

Write a pyspark code perform below function

- Get all employee details from EmployeeDetail table whose joining month is Jan(1).
- Get all employee details from EmployeeDetail table whose joining date between 2013-01-01" and "2013-12-01".
- Get how many employee exist in "EmployeeDetail" table.
- Select all employee detail with First name "Vikas","Ashish", and "Nikhil".

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve the

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

PYSPARK LEARNING HUB : DAY - 23

emp_df=spark.createDataFrame(data,schema)

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```python
# 28). Get all employee details from EmployeeDetail table whose joining month is
# Jan(1).
from pyspark.sql.functions import month
emp_df.filter(month(col("Joining_Date"))== 1) .show()


# 29). Get all employee details from EmployeeDetail table whose joining
# date between "2013-01-01" and "2013-12-01".

emp_df.filter(col("Joining_Date").between("2013-01-01","2013-12-01")).show()
```

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+

+----------+----------+---------+--------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+--------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat|600000.0|2013-02-15 11:16:...|        IT|  Male|
+----------+----------+---------+--------+--------------------+----------+------+
```

PYSPARK LEARNING HUB : DAY - 23

```
# 30). Get how many employee exist in "EmployeeDetail" table.
emp_df.count()

# 32. Select all employee detail with First name "Vikas","Ashish", and
"Nikhil".

from pyspark.sql.functions import lower
emp_df.filter(lower(col("First_Name"))\
      .isin("vikas","ashish","nikhil")).show()
```

```
Number of employees: 5
+----------+----------+---------+---------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+-------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
+----------+----------+---------+---------+-------------------+----------+------+
```

Step - 1 : Problem Statement

24_Trim and case in pyspark

Write a pyspark code perform below function

- Select all employee detail with First name not in "Vikas","Ashish", and "Nikhil".
- Select first name from "EmployeeDetail" df after removing white spaces from right side
- Select first name from "EmployeeDetail" table after removing white spaces from left side
- Display first name and Gender as M/F.(if male then M, if Female then F)

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
```

```
])
```

Step - 2 : Writing the pyspark code to solve the

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
```

```
        StructField("Gender", StringType(), True)
])

emp_df=spark.createDataFrame(data,schema)
```

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```python
# 33. Select all employee detail with First name not in "Vikas","Ashish", and "Nikhil".

from pyspark.sql.functions import col,lower
emp_df.filter(~lower(col("First_Name")).isin("vikas","ashish","nikhil")).show()

# 34. Select first name from "EmployeeDetail" df after removing white spaces from
# right side

from pyspark.sql.functions import rtrim
emp_df.select(rtrim(col("First_Name"))).show()
```

```
+----------+----------+---------+-------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+-------+--------------------+----------+------+
|         2|    nikita|     Jain|530000.0|2014-01-09 17:31:...|        HR|Female|
|         5|     anish|   kadian|500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+-------+--------------------+----------+------+

+-----------------+
|rtrim(First_Name)|
+-----------------+
|            Vikas|
|           nikita|
|           Ashish|
|           Nikhil|
|            anish|
+-----------------+
```

PYSPARK LEARNING HUB : DAY - 24

```python
# 35. Select first name from "EmployeeDetail" table after removing white spaces from
# left side

from pyspark.sql.functions import ltrim
emp_df.select(ltrim(col("First_Name"))).show()

# 36. Display first name and Gender as M/F.(if male then M, if Female then F)

from pyspark.sql.functions import when
emp_df.withColumn("Gen",when(col("Gender")=="Female","F")\
                        .when(col("Gender")=="Male","M")
                ).select("First_Name","Gen").show()
```

```
+------------------+
|ltrim(First_Name)|
+------------------+
|             Vikas|
|            nikita|
|            Ashish|
|            Nikhil|
|             anish|
+------------------+
```

```
+----------+---+
|First_Name|Gen|
+----------+---+
|     Vikas|  M|
|    nikita|  F|
|    Ashish|  M|
|    Nikhil|  M|
|     anish|  M|
+----------+---+
```

Step - 1 : Problem Statement

25_operator in pyspark

Write a pyspark code perform below function

- Select first name from "EmployeeDetail" table prifixed with "Hello "
- Get employee details from "EmployeeDetail" table whose Salary greater than 600000
- Get employee details from "EmployeeDetail" table whose Salary less than 700000
- Get employee details from "EmployeeDetail" table whose Salary between 500000 than 600000
- Select second highest salary from "EmployeeDetail" table

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
```

```
# Create a schema for the DataFrame
```

```
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve the

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]

# Create a schema for the DataFrame
```

```
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

emp_df=spark.createDataFrame(data,schema)
```

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```
# 37. Select first name from "EmployeeDetail" table prifixed with "Hello "
from pyspark.sql.functions import concat,lit
emp_df.withColumn("prefix_firstname",concat(lit('hello '),col('First_Name')))\
        .select("prefix_firstname").show()

# 38. Get employee details from "EmployeeDetail" table whose Salary greater than
# 600000

emp_df.filter(emp_df['Salary'] > 600000 ).show()
```

```
+----------------+
|prefix_firstname|
+----------------+
|     hello Vikas|
|    hello nikita|
|    hello Ashish|
|    hello Nikhil|
|     hello anish|
+----------------+
```

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```
# 39. Get employee details from "EmployeeDetail" table whose Salary less than 700000
emp_df.filter(emp_df['Salary'] < 700000 ).show()

# 40. Get employee details from "EmployeeDetail" table whose Salary between 500000
# than 600000

emp_df.filter(col("Salary").between(500000,600000)).show()

# 41. Select second highest salary from "EmployeeDetail" table
emp_df.select("Salary").distinct().orderBy(col('Salary').desc())\
        .limit(2).collect()[1][0]
```

```
+----------+----------+---------+--------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+--------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat|600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain|530000.0|2014-01-09 17:31:...|        HR|Female|
|         4|    Nikhil|   Sharma|480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian|500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+--------+--------------------+----------+------+

+----------+----------+---------+--------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|  Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+--------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat|600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain|530000.0|2014-01-09 17:31:...|        HR|Female|
|         5|     anish|   kadian|500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+--------+--------------------+----------+------+

600000.0
```

Step - 1 : Problem Statement

26_groupby in pyspark

Write a pyspark code perform below function

- Write the query to get the department and department wise total(sum) salary from "EmployeeDetail" table.
- Write the query to get the department and department wise total(sum) salary, display it in ascending order according to salary.
- Write the query to get the department and department wise total(sum) salary, display it in descending order according to salary.
- Write the query to get the department, total no. of departments, total(sum) salary with respect to department from "EmployeeDetail" table.

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
```

```
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve the

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
```

```
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

emp_df=spark.createDataFrame(data,schema)
```

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```
# 42. Write the query to get the department and department wise
#  total(sum) salary from "EmployeeDetail" table.

from pyspark.sql.functions import sum

emp_df.groupby(col('Department'))\
      .agg(sum('Salary').alias("sum_of_salary")).show()
```

```
+----------+-------------+
|Department|sum_of_salary|
+----------+-------------+
|        HR|    1010000.0|
|   Payroll|     500000.0|
|        IT|    1600000.0|
+----------+-------------+
```

```
# 43. Write the query to get the department and department wise total(sum) salary, display it in ascending order according to salary.

emp_df.groupby(col("Department"))\
      .agg(sum("Salary").alias("sum_of_salary"))\
      .orderBy(col('sum_of_salary').asc()).show()
```

```
+----------+-------------+
|Department|sum_of_salary|
+----------+-------------+
|   Payroll|     500000.0|
|        HR|    1010000.0|
|        IT|    1600000.0|
+----------+-------------+
```

```
# 44. Write the query to get the department and department wise total(sum)
salary,display it in descending order according to salary.

emp_df.groupby(col("Department"))\
      .agg(sum("Salary").alias("sum_of_salary"))\
      .orderBy(col('sum_of_salary').desc()).show()
```

```
+----------+--------------+
|Department|sum_of_salary|
+----------+--------------+
|        IT|     1600000.0|
|        HR|     1010000.0|
|   Payroll|      500000.0|
+----------+--------------+
```

```
# 45. Write the query to get the department, total no. of
departments,total(sum) salary with respect to department from
"EmployeeDetail" table.

from pyspark.sql.functions import count
emp_df.groupby(col("Department"))\
        .agg(count("Salary").alias("count"),\
         sum("Salary").alias("sum_od_Salary")).show()
```

```
+----------+-----+--------------+
|Department|count|sum_od_Salary|
+----------+-----+--------------+
|        HR|    2|     1010000.0|
|   Payroll|    1|      500000.0|
|        IT|    2|     1600000.0|
+----------+-----+--------------+
```

Step - 1 : Problem Statement

27_groupby in pyspark

Write a pyspark code perform below function

- 46. Get department wise average salary from "EmployeeDetail" table order by salary ascending
- 47. Get department wise maximum salary from "EmployeeDetail" table order by salary ascending
- 48. Get department wise minimum salary from "EmployeeDetail" table order by salary ascending

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]


# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])
```

Step - 2 : Writing the pyspark code to solve the

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()

# Create a list of rows from the image
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]

# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
```

```
        StructField("Gender", StringType(), True)
])

emp_df=spark.createDataFrame(data,schema)
```

```
+----------+----------+---------+---------+--------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|        Joining_Date|Department|Gender|
+----------+----------+---------+---------+--------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+--------------------+----------+------+
```

```python
# 46. Get department wise average salary from
"EmployeeDetail" table order by salary ascending
from pyspark.sql.functions import avg

emp_df.groupby("Department")\
      .agg(avg(col('Salary'))).show()
```

```
+----------+-----------+
|Department|avg(Salary)|
+----------+-----------+
|        HR|   505000.0|
|   Payroll|   500000.0|
|        IT|   800000.0|
+----------+-----------+
```

PYSPARK LEARNING HUB : DAY - 27

```python
# 47. Get department wise maximum salary from "EmployeeDetail" table order by salary
# ascending
from pyspark.sql.functions import max
emp_df.groupby("Department")\
      .agg(max(col('Salary')).alias("max_salary"))\
      .orderBy(col('max_salary').asc()).show()

# 48. Get department wise minimum salary from "EmployeeDetail" table order by
# salary ascending
from pyspark.sql.functions import min
emp_df.groupby("Department")\
    .agg(min(col('Salary')).alias("max_salary"))\
    .orderBy(col('max_salary').asc()).show()
```

```
+----------+----------+
|Department|max_salary|
+----------+----------+
|   Payroll|  500000.0|
|        HR|  530000.0|
|        IT| 1000000.0|
+----------+----------+

+----------+----------+
|Department|max_salary|
+----------+----------+
|        HR|  480000.0|
|   Payroll|  500000.0|
|        IT|  600000.0|
+----------+----------+
```

Step - 1 : Problem Statement

28_Join_in_pyspark

Write a pyspark code perform below function

- Write down the query to fetch Project name assign to more than one Employee
- Get employee name, project name order by firstname from "EmployeeDetail" and"ProjectDetail" for those employee which have assigned project already.

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
```

```
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

pro_schema = StructType([
        StructField("Project_DetailID", IntegerType(), True),
        StructField("Employee_DetailID", IntegerType(), True),
        StructField("Project_Name", StringType(), True)
])

# Create the data as a list of tuples
pro_data = [
        (1, 1, "Task Track"),
        (2, 1, "CLP"),
        (3, 1, "Survey Management"),
        (4, 2, "HR Management"),
        (5, 3, "Task Track"),
        (6, 3, "GRS"),
        (7, 3, "DDS"),
        (8, 4, "HR Management"),
        (9, 6, "GL Management")
]
```

Step - 2 : Writing the pyspark code to solve the

```
# import packages
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType,DoubleType,TimestampType
from pyspark.sql.functions import col

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
pro_schema = StructType([
    StructField("Project_DetailID", IntegerType(), True),
    StructField("Employee_DetailID", IntegerType(), True),
    StructField("Project_Name", StringType(), True)
])
# Create the data as a list of tuples
pro_data = [
    (1, 1, "Task Track"),
    (2, 1, "CLP"),
    (3, 1, "Survey Management"),
    (4, 2, "HR Management"),
    (5, 3, "Task Track"),
    (6, 3, "GRS"),
    (7, 3, "DDS"),
    (8, 4, "HR Management"),
    (9, 6, "GL Management")
]
pro_df=spark.createDataFrame(pro_data,pro_schema)
pro_df.show()
```

PYSPARK LEARNING HUB : DAY - 28

Project_DetailID	Employee_DetailID	Project_Name
1	1	Task Track
2	1	CLP
3	1	Survey Management
4	2	HR Management
5	3	Task Track
6	3	GRS
7	3	DDS
8	4	HR Management
9	6	GL Management

```
# Create a list of rows from the image
emp_data = [
[1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
[2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
[3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
[4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
[5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"]
]
# Create a schema for the DataFrame
emp_schema = StructType([
StructField("EmployeeID", IntegerType(), True),
StructField("First_Name", StringType(), True),
StructField("Last_Name", StringType(), True),
StructField("Salary", DoubleType(), True),
StructField("Joining_Date", StringType(), True),
StructField("Department", StringType(), True),
StructField("Gender", StringType(), True)
])
emp_df=spark.createDataFrame(emp_data,emp_schema)
emp_df.show()
```

```
+----------+----------+---------+---------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+-------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+-------------------+----------+------+
```

```python
# 49. Write down the query to fetch Project name assign to
#more than one Employee

from pyspark.sql.functions import count,col

pro_df.groupby(col("Project_Name"))\
      .agg(count("*").alias("count_pro"))\
      .filter(col("count_pro") > 1).show()
```

```
+-------------+---------+
| Project_Name|count_pro|
+-------------+---------+
|HR Management|        2|
|   Task Track|        2|
+-------------+---------+
```

PYSPARK LEARNING HUB : DAY - 28

```
# 51. Get employee name, project name order by firstname from "EmployeeDetail"
# and "ProjectDetail" for those employee which have assigned project already.

from pyspark.sql.functions import lower
emp_df.join(pro_df,emp_df['EmployeeID']==pro_df['Employee_DetailID'],"inner")\
      .orderBy(lower(col("First_Name")).asc())\
      .select("First_Name","Project_Name").show()
```

```
+----------+------------------+
|First_Name|      Project_Name|
+----------+------------------+
|    Ashish|               GRS|
|    Ashish|               DDS|
|    Ashish|        Task Track|
|    Nikhil|     HR Management|
|    nikita|     HR Management|
|     Vikas|        Task Track|
|     Vikas|               CLP|
|     Vikas|Survey Management|
+----------+------------------+
```

PYSPARK LEARNING HUB : DAY - 29

Step - 1 : Problem Statement

29_Join_in_pyspark

Write a pyspark code perform below function

- 52. Get employee name, project name order by firstname from "EmployeeDetail" and "ProjectDetail" for all employee even they have not assigned project.
- 53 Get employee name, project name order by firstname from "EmployeeDetail" and "ProjectDetail" for all employee if project is not assigned then display "-No Project Assigned".

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
```

```
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

pro_schema = StructType([
        StructField("Project_DetailID", IntegerType(), True),
        StructField("Employee_DetailID", IntegerType(), True),
        StructField("Project_Name", StringType(), True)
])

# Create the data as a list of tuples
pro_data = [
        (1, 1, "Task Track"),
        (2, 1, "CLP"),
        (3, 1, "Survey Management"),
        (4, 2, "HR Management"),
        (5, 3, "Task Track"),
        (6, 3, "GRS"),
        (7, 3, "DDS"),
        (8, 4, "HR Management"),
        (9, 6, "GL Management")
]
```


Step - 2 : Writing the pyspark code to solve the

PYSPARK LEARNING HUB : DAY - 29

```python
# import packages
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType,DoubleType,TimestampType
from pyspark.sql.functions import col

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```python
pro_schema = StructType([
    StructField("Project_DetailID", IntegerType(), True),
    StructField("Employee_DetailID", IntegerType(), True),
    StructField("Project_Name", StringType(), True)
])
# Create the data as a list of tuples
pro_data = [
    (1, 1, "Task Track"),
    (2, 1, "CLP"),
    (3, 1, "Survey Management"),
    (4, 2, "HR Management"),
    (5, 3, "Task Track"),
    (6, 3, "GRS"),
    (7, 3, "DDS"),
    (8, 4, "HR Management"),
    (9, 6, "GL Management")
]
pro_df=spark.createDataFrame(pro_data,pro_schema)
pro_df.show()
```

PYSPARK LEARNING HUB : DAY - 29

Project_DetailID	Employee_DetailID	Project_Name
1	1	Task Track
2	1	CLP
3	1	Survey Management
4	2	HR Management
5	3	Task Track
6	3	GRS
7	3	DDS
8	4	HR Management
9	6	GL Management

```
# Create a list of rows from the image
emp_data = [
[1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
[2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
[3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
[4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
[5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"]
]
# Create a schema for the DataFrame
emp_schema = StructType([
StructField("EmployeeID", IntegerType(), True),
StructField("First_Name", StringType(), True),
StructField("Last_Name", StringType(), True),
StructField("Salary", DoubleType(), True),
StructField("Joining_Date", StringType(), True),
StructField("Department", StringType(), True),
StructField("Gender", StringType(), True)
])
emp_df=spark.createDataFrame(emp_data,emp_schema)
emp_df.show()
```

```
+----------+----------+---------+---------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+-------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+-------------------+----------+------+
```

```python
# 52. Get employee name, project name order by firstname from "EmployeeDetail" and
# "ProjectDetail" for all employee even they have not assigned project.

emp_df.join(pro_df,emp_df['EmployeeID'] == pro_df['Employee_DetailID'] ,"left")\
      .select("First_Name","Project_Name")\
      .orderBy(lower(col("First_Name")))\
      .show()
```

```
+----------+------------------+
|First_Name|      Project_Name|
+----------+------------------+
|     anish|              null|
|    Ashish|               GRS|
|    Ashish|               DDS|
|    Ashish|        Task Track|
|    Nikhil|    HR Management|
|    nikita|    HR Management|
|     Vikas|        Task Track|
|     Vikas|Survey Management|
|     Vikas|               CLP|
+----------+------------------+
```

PYSPARK LEARNING HUB : DAY - 29

```python
#  Get employee name, project name order by firstname from
# "EmployeeDetail" and "ProjectDetail" for all employee if project is not assigned then
# display "-No Project Assigned".
from pyspark.sql.functions import when,coalesce,lower,lit

left_df=emp_df.join(pro_df,emp_df['EmployeeID'] == pro_df['Employee_DetailID'] ,"left")\
      .select("EmployeeID","First_Name","Last_Name","Project_Name")\
      .orderBy(lower(col("First_Name")))

left_df.withColumn("Project_Name",coalesce(col("Project_Name"), lit("-No Project
Assigned"))).show()
```

```
+----------+----------+---------+--------------------+
|EmployeeID|First_Name|Last_Name|        Project_Name|
+----------+----------+---------+--------------------+
|         5|     anish|   kadian|-No Project Assigned|
|         3|    Ashish|    Kumar|                 GRS|
|         3|    Ashish|    Kumar|                 DDS|
|         3|    Ashish|    Kumar|          Task Track|
|         4|    Nikhil|   Sharma|       HR Management|
|         2|    nikita|     Jain|       HR Management|
|         1|     Vikas|  Ahlawat|          Task Track|
|         1|     Vikas|  Ahlawat|   Survey Management|
|         1|     Vikas|  Ahlawat|                 CLP|
+----------+----------+---------+--------------------+
```

Step - 1 : Problem Statement

30_Join_in_pyspark

Write a pyspark code perform below function

- 56. Write a pyspark code to find out the employeename who has not assigned any project, and display "-No Project Assigned"(tables :- [EmployeeDetail],[ProjectDetail]).
- 57. Write a pyspark code to find out the project name which is not assigned to any employee(tables :- [EmployeeDetail],[ProjectDetail]).

Difficult Level : EASY

DataFrame:

```
data = [
        [1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
        [2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
        [3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
        [4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
        [5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"],
]
```

```
# Create a schema for the DataFrame
schema = StructType([
        StructField("EmployeeID", IntegerType(), True),
        StructField("First_Name", StringType(), True),
        StructField("Last_Name", StringType(), True),
        StructField("Salary", DoubleType(), True),
        StructField("Joining_Date", StringType(), True),
        StructField("Department", StringType(), True),
        StructField("Gender", StringType(), True)
])

pro_schema = StructType([
        StructField("Project_DetailID", IntegerType(), True),
        StructField("Employee_DetailID", IntegerType(), True),
        StructField("Project_Name", StringType(), True)
])

# Create the data as a list of tuples
pro_data = [
        (1, 1, "Task Track"),
        (2, 1, "CLP"),
        (3, 1, "Survey Management"),
        (4, 2, "HR Management"),
        (5, 3, "Task Track"),
        (6, 3, "GRS"),
        (7, 3, "DDS"),
        (8, 4, "HR Management"),
        (9, 6, "GL Management")
]
```

Step - 2 : Writing the pyspark code to solve the

```
# import packages
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType,DoubleType,TimestampType
from pyspark.sql.functions import col

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
pro_schema = StructType([
    StructField("Project_DetailID", IntegerType(), True),
    StructField("Employee_DetailID", IntegerType(), True),
    StructField("Project_Name", StringType(), True)
])
# Create the data as a list of tuples
pro_data = [
    (1, 1, "Task Track"),
    (2, 1, "CLP"),
    (3, 1, "Survey Management"),
    (4, 2, "HR Management"),
    (5, 3, "Task Track"),
    (6, 3, "GRS"),
    (7, 3, "DDS"),
    (8, 4, "HR Management"),
    (9, 6, "GL Management")
]
pro_df=spark.createDataFrame(pro_data,pro_schema)
pro_df.show()
```

PYSPARK LEARNING HUB : DAY - 30

Project_DetailID	Employee_DetailID	Project_Name
1	1	Task Track
2	1	CLP
3	1	Survey Management
4	2	HR Management
5	3	Task Track
6	3	GRS
7	3	DDS
8	4	HR Management
9	6	GL Management

```
# Create a list of rows from the image
emp_data = [
[1, "Vikas", "Ahlawat", 600000.0, "2013-02-15 11:16:28.290", "IT", "Male"],
[2, "nikita", "Jain", 530000.0, "2014-01-09 17:31:07.793", "HR", "Female"],
[3, "Ashish", "Kumar", 1000000.0, "2014-01-09 10:05:07.793", "IT", "Male"],
[4, "Nikhil", "Sharma", 480000.0, "2014-01-09 09:00:07.793", "HR", "Male"],
[5, "anish", "kadian", 500000.0, "2014-01-09 09:31:07.793", "Payroll", "Male"]
]
# Create a schema for the DataFrame
emp_schema = StructType([
StructField("EmployeeID", IntegerType(), True),
StructField("First_Name", StringType(), True),
StructField("Last_Name", StringType(), True),
StructField("Salary", DoubleType(), True),
StructField("Joining_Date", StringType(), True),
StructField("Department", StringType(), True),
StructField("Gender", StringType(), True)
])
emp_df=spark.createDataFrame(emp_data,emp_schema)
emp_df.show()
```

```
+----------+----------+---------+---------+-------------------+----------+------+
|EmployeeID|First_Name|Last_Name|   Salary|       Joining_Date|Department|Gender|
+----------+----------+---------+---------+-------------------+----------+------+
|         1|     Vikas|  Ahlawat| 600000.0|2013-02-15 11:16:...|        IT|  Male|
|         2|    nikita|     Jain| 530000.0|2014-01-09 17:31:...|        HR|Female|
|         3|    Ashish|    Kumar|1000000.0|2014-01-09 10:05:...|        IT|  Male|
|         4|    Nikhil|   Sharma| 480000.0|2014-01-09 09:00:...|        HR|  Male|
|         5|     anish|   kadian| 500000.0|2014-01-09 09:31:...|   Payroll|  Male|
+----------+----------+---------+---------+-------------------+----------+------+
```

```python
# 56. Write a query to find out the employeename who has not assigned any project,
# and display "-No Project Assigned"( tables :- [EmployeeDetail],[ProjectDetail]).
from pyspark.sql.functions import lower,coalesce,lit

emp_df.join(pro_df,emp_df['EmployeeID'] == pro_df['Employee_DetailID'] ,"left")\
      .filter(col("Project_Name").isNull())\
      .select("First_Name",coalesce(col("Project_Name"), lit("-No Project Assigned")).alias("Project_Name"))\
      .orderBy(lower(col("First_Name")))\
        .show()
```

```
+----------+--------------------+
|First_Name|        Project_Name|
+----------+--------------------+
|     anish|-No Project Assigned|
+----------+--------------------+
```

PYSPARK LEARNING HUB : DAY - 30

```
# 57. Write a query to find out the project name which is not assigned to any employee(
# tables :- [EmployeeDetail],[ProjectDetail]).

from pyspark.sql.functions import col
result_df=emp_df.join(pro_df,emp_df["EmployeeID"]=pro_df["Employee_DetailID"],"right")\
      .select("EmployeeID","Project_Name")

result_df.filter(col("EmployeeID").isNull()).select("Project_Name").show()
```

```
+--------------+
| Project_Name |
+--------------+
|GL Management |
+--------------+
```

Step - 1 : Problem Statement

31_Histogram of Tweets

write a query to obtain a histogram of tweets posted per user in 2022. Output the tweet count per user as the bucket and the number of Twitter users who fall into that bucket.

In other words, group the users by the number of tweets they posted in 2022 and count the number of users in each group.

Difficult Level : EASY

DataFrame:

```
schema = StructType([
        StructField("tweet_id", IntegerType(), True),
        StructField("user_id", IntegerType(), True),
        StructField("msg", StringType(), True),
        StructField("tweet_date", StringType(), True)
])

# Define the data
data = [
        (214252, 111, 'Am considering taking Tesla private at $420. Funding secured.', '2021-12-30 00:00:00'),
        (739252, 111, 'Despite the constant negative press covfefe', '2022-01-01 00:00:00'),
        (846402, 111, 'Following @NickSinghTech on Twitter changed my life!', '2022-02-14 00:00:00'),
        (241425, 254, 'If the salary is so competitive why won’t you tell me what it is?', '2022-03-01 00:00:00'),
        (231574, 148, 'I no longer have a manager. I can\'t be managed', '2022-03-23 00:00:00')
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT			
TWEET_ID	USER_ID	MSG	TWEET_DATE
214252	111	Am considering taking Tesla private at $420. Funding secured.	2021-12-30 0:00:00
739252	111	Despite the constant negative press covfefe	2022-01-01 0:00:00
846402	111	Following @NickSinghTech on Twitter changed my life!	2022-02-14 0:00:00
241425	254	If the salary is so competitive why won’t you tell me what it is?	2022-03-01 0:00:00
231574	148	I no longer have a manager. I can't be managed	2022-03-23 0:00:00

OUTPUT

OUTPUT	
BUCKET	USER_NUM
1	2
2	1

PYSPARK LEARNING HUB : DAY - 31

Step - 3 : Writing the pyspark code to solve

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import
StructType,StructField,IntegerType,StringType,DoubleType,TimestampType
from pyspark.sql.functions import col,max,min,to_timestamp,date_format,current_date,
current_timestamp,datediff

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
# Define the schema for the DataFrame
schema = StructType([
    StructField("tweet_id", IntegerType(), True),
    StructField("user_id", IntegerType(), True),
    StructField("msg", StringType(), True),
    StructField("tweet_date", StringType(), True)
])
# Define the data
data = [
    (214252, 111, 'Am considering taking Tesla private at $420. Funding secured.', '2021-12-30
00:00:00'),
    (739252, 111, 'Despite the constant negative press covfefe', '2022-01-01 00:00:00'),
    (846402, 111, 'Following @NickSinghTech on Twitter changed my life!', '2022-02-14 00:00:00'),
    (241425, 254, 'If the salary is so competitive why won't you tell me what it is?', '2022-03-01
00:00:00'),
    (231574, 148, 'I no longer have a manager. I can\'t be managed', '2022-03-23 00:00:00')
]

df=spark.createDataFrame(data,schema)
df.show()
```

```
+--------+-------+--------------------+-------------------+
|tweet_id|user_id|                 msg|         tweet_date|
+--------+-------+--------------------+-------------------+
|  214252|    111|Am considering ta...|2021-12-30 00:00:00|
|  739252|    111|Despite the const...|2022-01-01 00:00:00|
|  846402|    111|Following @NickSi...|2022-02-14 00:00:00|
|  241425|    254|If the salary is ...|2022-03-01 00:00:00|
|  231574|    148|I no longer have ...|2022-03-23 00:00:00|
+--------+-------+--------------------+-------------------+
```

```python
from pyspark.sql.functions import count

df=df.filter(df["tweet_date"].like('2022%'))
df.show()

df2=df.groupby("user_id").agg(count("user_id").alias("cnt"))
df2.show()

df2.groupBy("cnt").agg(count("user_id").alias("user")).show()
```

```
+--------+-------+--------------------+-------------------+
|tweet_id|user_id|                 msg|         tweet_date|
+--------+-------+--------------------+-------------------+
|  739252|    111|Despite the const...|2022-01-01 00:00:00|
|  846402|    111|Following @NickSi...|2022-02-14 00:00:00|
|  241425|    254|If the salary is ...|2022-03-01 00:00:00|
|  231574|    148|I no longer have ...|2022-03-23 00:00:00|
+--------+-------+--------------------+-------------------+

+-------+---+
|user_id|cnt|
+-------+---+
|    148|  1|
|    111|  2|
|    254|  1|
+-------+---+

+---+----+
|cnt|user|
+---+----+
|  1|   2|
|  2|   1|
+---+----+
```

Step - 1 : Problem Statement

32_pyspark_transformation

Write a pyspark code to transform the DataFrame to display each student's marks in Math and English as separate columns.

Difficult Level : EASY

DataFrame:

```
data=[
('Rudra','math',79),
('Rudra','eng',60),
('Shivu','math', 68),
('Shivu','eng', 59),
('Anu','math', 65),
('Anu','eng',80)
]

schema = StructType([
        StructField("Name", StringType(), True),
        StructField("Sub", StringType(), True),
        StructField("Marks", IntegerType(), True)
])
```


Step - 2 : Identifying The Input Data And Expected

INPUT

```
+-----+----+-----+
| Name| Sub|Marks|
+-----+----+-----+
|Rudra|math|   79|
|Rudra| eng|   60|
|Shivu|math|   68|
|Shivu| eng|   59|
|  Anu|math|   65|
|  Anu| eng|   80|
+-----+----+-----+
```

OUTPUT

```
+-----+----+---+
| name|Math|eng|
+-----+----+---+
|Shivu|  68| 59|
|Rudra|  79| 60|
|  Anu|  65| 80|
+-----+----+---+
```

Step - 3 : Writing the pyspark code to solve

PYSPARK LEARNING HUB : DAY - 32

```
# Creating Spark Session
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType,StructField,IntegerType,StringType


#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
data=[
('Rudra','math',79),
('Rudra','eng',60),
('Shivu','math', 68),
('Shivu','eng', 59),
('Anu','math', 65),
('Anu','eng',80)
]
schema = StructType([
    StructField("Name", StringType(), True),
    StructField("Sub", StringType(), True),
    StructField("Marks", IntegerType(), True)
])
df=spark.createDataFrame(data,schema)
df.show()
```

PYSPARK LEARNING HUB : DAY - 32

```
+-----+----+-----+
| Name| Sub|Marks|
+-----+----+-----+
|Rudra|math|   79|
|Rudra| eng|   60|
|Shivu|math|   68|
|Shivu| eng|   59|
|  Anu|math|   65|
|  Anu| eng|   80|
+-----+----+-----+
```



```python
from pyspark.sql.functions import *

df=df.groupby("Name").agg(collect_list("Marks").alias("marks_new"))
df.show()

df=df.withColumn("Math",col('marks_new')[0])\
  .withColumn("eng",col("marks_new")[1])
df.show()

df.select("name","Math",'eng').show()
```

```
+-----+---------+
| Name|marks_new|
+-----+---------+
|Shivu| [68, 59]|
|Rudra| [79, 60]|
|  Anu| [65, 80]|
+-----+---------+
```

```
+-----+---------+----+---+
| Name|marks_new|Math|eng|
+-----+---------+----+---+
|Shivu| [59, 68]|  59| 68|
|Rudra| [79, 60]|  79| 60|
|  Anu| [65, 80]|  65| 80|
+-----+---------+----+---+
```

```
+-----+----+---+
| name|Math|eng|
+-----+----+---+
|Shivu|  68| 59|
|Rudra|  79| 60|
|  Anu|  65| 80|
+-----+----+---+
```

Step - 1 : Problem Statement

33_Hobbies Data Transformation

Problem Statement:

Transform a dataset with individuals' names and associated hobbies into a new format using PySpark. Convert the comma-separated hobbies into separate rows, creating a DataFrame with individual rows for each person and their respective hobbies.

Difficult Level : EASY

DataFrame:

```
# Sample input data
data = [("Alice", "badminton,tennis"),
        ("Bob", "tennis,cricket"),
        ("Julie", "cricket,carroms")]

# Create a DataFrame
df = spark.createDataFrame(data, ["name", "hobbies"])
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

```
+-----+-----------------+
| name|          hobbies|
+-----+-----------------+
|Alice|badminton,tennis|
|  Bob|  tennis,cricket|
|Julie| cricket,carroms|
+-----+-----------------+
```

OUTPUT

```
+-----+---------+
| name|    hobby|
+-----+---------+
|Alice|badminton|
|Alice|   tennis|
|  Bob|   tennis|
|  Bob|  cricket|
|Julie|  cricket|
|Julie|  carroms|
+-----+---------+
```

Step - 3 : Writing the pyspark code to solve

```
from pyspark.sql import SparkSession

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
# Sample input data
data = [("Alice", "badminton,tennis"),
        ("Bob", "tennis,cricket"),
        ("Julie", "cricket,carroms")]

# Create a DataFrame
df = spark.createDataFrame(data, ["name", "hobbies"])

df.show()
```

```
+-----+----------------+
| name|         hobbies|
+-----+----------------+
|Alice|badminton,tennis|
|  Bob|  tennis,cricket|
|Julie| cricket,carroms|
+-----+----------------+
```

```python
from pyspark.sql.functions import explode,col,split

df_transformed = df.withColumn("hobbies", split("hobbies", ",")) \
                   .withColumn("hobby", explode("hobbies")) \
                   .select("name", "hobby").show()
```

```
+-----+---------+
| name|    hobby|
+-----+---------+
|Alice|badminton|
|Alice|   tennis|
|  Bob|   tennis|
|  Bob|  cricket|
|Julie|  cricket|
|Julie|  carroms|
+-----+---------+
```

```
+-----+---------+
| Name|marks_new|
+-----+---------+
|Shivu| [68, 59]|
|Rudra| [79, 60]|
|  Anu| [65, 80]|
+-----+---------+
```

```
+-----+---------+----+---+
| Name|marks_new|Math|eng|
+-----+---------+----+---+
|Shivu| [59, 68]|  59| 68|
|Rudra| [79, 60]|  79| 60|
|  Anu| [65, 80]|  65| 80|
+-----+---------+----+---+
```

```
+-----+----+---+
| name|Math|eng|
+-----+----+---+
|Shivu|  68| 59|
|Rudra|  79| 60|
|  Anu|  65| 80|
+-----+----+---+
```

Step - 1 : Problem Statement

34_ Histogram of Tweets

write a pyspark code to obtain a histogram of tweets posted per user in 2022. Output the tweet count per user as the bucket and the number of Twitter users who fall into that bucket.In other words, group the users by the number of tweets they posted in 2022 and count the number of users in each group.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the tweets DataFrame
schema = StructType([
        StructField("tweet_id", IntegerType(), True),
        StructField("user_id", IntegerType(), True),
        StructField("msg", StringType(), True),
        StructField("tweet_date", StringType(), True)
])

# Create the tweets DataFrame
data = [
        (214252, 111, 'Am considering taking Tesla private at $420. Funding secured.', '2021-12-30 00:00:00'),
        (739252, 111, 'Despite the constant negative press covfefe', '2022-01-01 00:00:00'),
        (846402, 111, 'Following @NickSinghTech on Twitter changed my life!', '2022-02-14 00:00:00'),
        (241425, 254, 'If the salary is so competitive why won’t you tell me what it is?', '2022-03-01 00:00:00'),
        (231574, 148, 'I no longer have a manager. I can\'t be managed', '2022-03-23 00:00:00')
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

INPUT			
TWEET_ID	USER_ID	MSG	TWEET_DATE
214252	111	Am considering taking Tesla private at $420. Funding secured.	2021-12-30 0:00:00
739252	111	Despite the constant negative press covfefe	2022-01-01 0:00:00
846402	111	Following @NickSinghTech on Twitter changed my life!	2022-02-14 0:00:00
241425	254	If the salary is so competitive why won’t you tell me what it is?	2022-03-01 0:00:00
231574	148	I no longer have a manager. I can't be managed	2022-03-23 0:00:00

OUTPUT

OUTPUT	
BUCKET	USER_NUM
1	2
2	1

Step - 3 : Writing the pyspark code to solve

```
from pyspark.sql import SparkSession

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

PYSPARK LEARNING HUB : DAY - 34

```
# Define the schema for the tweets DataFrame
schema = StructType([
    StructField("tweet_id", IntegerType(), True),
    StructField("user_id", IntegerType(), True),
    StructField("msg", StringType(), True),
    StructField("tweet_date", StringType(), True)
])
# Create the tweets DataFrame
data = [
    (214252, 111, 'Am considering taking Tesla private at $420. Funding secured.', '2021-12-30
00:00:00'),
    (739252, 111, 'Despite the constant negative press covfefe', '2022-01-01 00:00:00'),
    (846402, 111, 'Following @NickSinghTech on Twitter changed my life!', '2022-02-14 00:00:00'),
    (241425, 254, 'If the salary is so competitive why won’t you tell me what it is?', '2022-03-01
00:00:00'),
    (231574, 148, 'I no longer have a manager. I can\'t be managed', '2022-03-23 00:00:00')
]
df=spark.createDataFrame(data,schema)
df.show(truncate=False)
```

```
+--------+-------+-----------------------------------------------------------------+-------------------+
|tweet_id|user_id|msg                                                              |tweet_date         |
+--------+-------+-----------------------------------------------------------------+-------------------+
|214252  |111    |Am considering taking Tesla private at $420. Funding secured.    |2021-12-30 00:00:00|
|739252  |111    |Despite the constant negative press covfefe                      |2022-01-01 00:00:00|
|846402  |111    |Following @NickSinghTech on Twitter changed my life!             |2022-02-14 00:00:00|
|241425  |254    |If the salary is so competitive why won’t you tell me what it is?|2022-03-01 00:00:00|
|231574  |148    |I no longer have a manager. I can't be managed                   |2022-03-23 00:00:00|
+--------+-------+-----------------------------------------------------------------+-------------------+
```

PYSPARK LEARNING HUB : DAY - 34

```
from pyspark.sql.functions import col,count
df=df.filter(col('tweet_date').like('2022%'))
df.show()

df_2=df.groupby("user_id").agg(count("msg").alias("cnt"))
df_2.show()

result_df=df_2.groupby(col('cnt')).agg(count("user_id").alias("bucket"))
result_df.select(col("cnt").alias("user_num"),"bucket").show()
```

```
+--------+-------+--------------------+-------------------+
|tweet_id|user_id|                 msg|         tweet_date|
+--------+-------+--------------------+-------------------+
|  739252|    111|Despite the const...|2022-01-01 00:00:00|
|  846402|    111|Following @NickSi...|2022-02-14 00:00:00|
|  241425|    254|If the salary is ...|2022-03-01 00:00:00|
|  231574|    148|I no longer have ...|2022-03-23 00:00:00|
+--------+-------+--------------------+-------------------+

+-------+---+
|user_id|cnt|
+-------+---+
|    148|  1|
|    111|  2|
|    254|  1|
+-------+---+

+--------+------+
|user_num|bucket|
+--------+------+
|       1|     2|
|       2|     1|
+--------+------+
```

PYSPARK LEARNING HUB : DAY - 34

Step - 1 : Problem Statement

35_Classes More Than 5 Students

Write a pyspark code to find all the classes that have at least five students.Return the result table in any order.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the DataFrame
schema = StructType([
        StructField("StudentID", StringType(), True),
        StructField("ClassName", StringType(), True)
])
# Data to be inserted into the DataFrame
data = [
        ('A', 'Math'),
        ('B', 'English'),
        ('C', 'Math'),
        ('D', 'Biology'),
        ('E', 'Math'),
        ('F', 'Computer'),
        ('G', 'Math'),
        ('H', 'Math'),
        ('I', 'Math')
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

```
+---------+---------+
|StudentID|ClassName|
+---------+---------+
|        A|     Math|
|        B|  English|
|        C|     Math|
|        D|  Biology|
|        E|     Math|
|        F| Computer|
|        G|     Math|
|        H|     Math|
|        I|     Math|
+---------+---------+
```

OUTPUT

```
+---------+---+
|ClassName|cnt|
+---------+---+
|     Math|  6|
+---------+---+
```

Step - 3 : Writing the pyspark code to solve

PYSPARK LEARNING HUB : DAY - 35

```python
from pyspark.sql import SparkSession

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```python
# Define the schema for the DataFrame
schema = StructType([
    StructField("StudentID", StringType(), True),
    StructField("ClassName", StringType(), True)
])
# Data to be inserted into the DataFrame
data = [
    ('A', 'Math'),
    ('B', 'English'),
    ('C', 'Math'),
    ('D', 'Biology'),
    ('E', 'Math'),
    ('F', 'Computer'),
    ('G', 'Math'),
    ('H', 'Math'),
    ('I', 'Math')
]
# Create a PySpark DataFrame
student_class_df = spark.createDataFrame(data, schema=schema)
# Show the DataFrame
student_class_df.show()
```

```
+---------+---------+
|StudentID|ClassName|
+---------+---------+
|        A|     Math|
|        B|  English|
|        C|     Math|
|        D|  Biology|
|        E|     Math|
|        F| Computer|
|        G|     Math|
|        H|     Math|
|        I|     Math|
+---------+---------+
```

```
from pyspark.sql.functions import col,count
df= student_class_df.groupby(col("ClassName"))\
                .agg(count("StudentID").alias("cnt"))
df.show()

df.filter(col("cnt")>=5).show()
```

```
+---------+---+
|ClassName|cnt|
+---------+---+
|     Math|  6|
|  English|  1|
| Computer|  1|
|  Biology|  1|
+---------+---+

+---------+---+
|ClassName|cnt|
+---------+---+
|     Math|  6|
+---------+---+
```

Step - 1 : Problem Statement

36_Rank Scores Problem

Write a pyspark code to rank scores. If there is a tie between two scores, both should have the same ranking. Note that after a tie, the next ranking number should be the next consecutive integer value.In other words, there should be no “holes” between ranks.

Difficult Level : MED

DataFrame:

```
# Define the schema for the DataFrame
schema = StructType([
        StructField("Id", IntegerType(), True),
        StructField("Score", FloatType(), True)
])

# Data to be inserted into the DataFrame
data = [
        (1, 3.50),
        (2, 3.65),
        (3, 4.00),
        (4, 3.85),
        (5, 4.00),
        (6, 3.65)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

```
+---+-----+
| Id|Score|
+---+-----+
|  1|  3.5|
|  2| 3.65|
|  3|  4.0|
|  4| 3.85|
|  5|  4.0|
|  6| 3.65|
+---+-----+
```

OUTPUT

```
+---+-----+----------+
| Id|Score|dense_rank|
+---+-----+----------+
|  5|  4.0|         1|
|  3|  4.0|         1|
|  4| 3.85|         2|
|  6| 3.65|         3|
|  2| 3.65|         3|
|  1|  3.5|         4|
+---+-----+----------+
```

Step - 3 : Writing the pyspark code to solve

```
from pyspark.sql import SparkSession

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
# Define the schema for the DataFrame
schema = StructType([
    StructField("Id", IntegerType(), True),
    StructField("Score", FloatType(), True)
])
# Data to be inserted into the DataFrame
data = [
    (1, 3.50),
    (2, 3.65),
    (3, 4.00),
    (4, 3.85),
    (5, 4.00),
    (6, 3.65)
]
# Create a PySpark DataFrame
score_df = spark.createDataFrame(data, schema=schema)
# Show the DataFrame
score_df.show()
```

```
+---+-----+
| Id|Score|
+---+-----+
|  1|  3.5|
|  2| 3.65|
|  3|  4.0|
|  4| 3.85|
|  5|  4.0|
|  6| 3.65|
+---+-----+
```

```python
from pyspark.sql import Window
from pyspark.sql.functions import dense_rank,desc


# Define a window specification
window_spec = Window.orderBy(desc("Score"))

# Add a row number column using the window function
score_df = score_df.withColumn("dense_rank",
dense_rank().over(window_spec))

# Show the DataFrame with row numbers
score_df.show()
```

```
+---+-----+----------+
| Id|Score|dense_rank|
+---+-----+----------+
|  5|  4.0|         1|
|  3|  4.0|         1|
|  4| 3.85|         2|
|  6| 3.65|         3|
|  2| 3.65|         3|
|  1|  3.5|         4|
+---+-----+----------+
```

Step - 1 : Problem Statement

37_Triangle Judgement Problem

A pupil Tim gets homework to identify whether three line segments could possibly form a triangle.
However, this assignment is very heavy because there are hundreds of records to calculate.
Could you help Tim by writing a pyspark code to judge whether these three sides can form a triangle,
assuming df triangle holds the length of the three sides x, y and z.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the DataFrame
schema = StructType([
        StructField("x", IntegerType(), True),
        StructField("y", IntegerType(), True),
        StructField("z", IntegerType(), True)
])

# Data to be inserted into the DataFrame
data = [
        (13, 15, 30),
        (10, 20, 15)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

```
+---+---+---+
|  x|  y|  z|
+---+---+---+
| 13| 15| 30|
| 10| 20| 15|
+---+---+---+
```

OUTPUT

```
+---+---+---+--------+
|  x|  y|  z|triangle|
+---+---+---+--------+
| 13| 15| 30|      No|
| 10| 20| 15|     Yes|
+---+---+---+--------+
```

PYSPARK LEARNING HUB : DAY - 37

Step - 3 : Writing the pyspark code to solve

```
from pyspark.sql import SparkSession

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
# Define the schema for the DataFrame
schema = StructType([
    StructField("x", IntegerType(), True),
    StructField("y", IntegerType(), True),
    StructField("z", IntegerType(), True)
])

# Data to be inserted into the DataFrame
data = [
    (13, 15, 30),
    (10, 20, 15)
]

# Create a PySpark DataFrame
df = spark.createDataFrame(data, schema=schema)

# Show the DataFrame
df.show()
```

```
+---+---+---+
|  x|  y|  z|
+---+---+---+
| 13| 15| 30|
| 10| 20| 15|
+---+---+---+
```

```
from pyspark.sql.functions import col, when

#Add a new column 'triangle' using when and col functions
result_df = df.withColumn(
    "triangle",
    when((col("x") + col("y") > col("z")) & (col("x") + col("z")
> col("y")) & (col("y") + col("z") > col("x")), "Yes")
    .otherwise("No")
)

result_df.show()
```

```
+---+---+---+--------+
|  x|  y|  z|triangle|
+---+---+---+--------+
| 13| 15| 30|      No|
| 10| 20| 15|     Yes|
+---+---+---+--------+
```

Step - 1 : Problem Statement

38_Biggest Single Number Problem

Df contains many numbers in column num including duplicated ones. Can you write a pyspark code to find the biggest number, which only appears once.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the DataFrame
schema = StructType([StructField("num", IntegerType(), True)])

# Your data
data = [(8,), (8,), (3,), (3,), (1,), (4,), (5,), (6,)]

# Create a PySpark DataFrame
df = spark.createDataFrame(data, schema=schema)
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

```
+---+
|num|
+---+
|  8|
|  8|
|  3|
|  3|
|  1|
|  4|
|  5|
|  6|
+---+
```

OUTPUT

```
+--------+
|max(num)|
+--------+
|       6|
+--------+
```

Step - 3 : Writing the pyspark code to solve

```
from pyspark.sql import SparkSession

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
# Define the schema for the DataFrame
schema = StructType([StructField("num", IntegerType(), True)])

# Your data
data = [(8,), (8,), (3,), (3,), (1,), (4,), (5,), (6,)]

# Create a PySpark DataFrame
df = spark.createDataFrame(data, schema=schema)
df.show()
```

```
+---+
|num|
+---+
|  8|
|  8|
|  3|
|  3|
|  1|
|  4|
|  5|
|  6|
+---+
```

```python
from pyspark.sql.functions import count,col

result_df=df.groupby("num").agg(count('num').alias('cnt'))\
            .filter(col('cnt') == 1)

result_df.show()
```

```
+---+---+
|num|cnt|
+---+---+
|  1|  1|
|  6|  1|
|  5|  1|
|  4|  1|
+---+---+
```

```
from pyspark.sql.functions import max

result_df.select(max(col('num'))).show()
```

```
+--------+
|max(num)|
+--------+
|       6|
+--------+
```

Step - 1 : Problem Statement

39_Not Boring Movies Problem

X city opened a new cinema, many people would like to go to this cinema. The cinema also gives out a poster indicating the movies' ratings and descriptions. Please write a Pyspark Code to output movies with an odd numbered ID and a description that is not 'boring'. Order the result by rating.

Difficult Level : EASY

DataFrame:

```
# Define the schema for the DataFrame
schema = StructType([
        StructField("id", IntegerType(), True),
        StructField("movie", StringType(), True),
        StructField("description", StringType(), True),
        StructField("rating", FloatType(), True)
])

# Your data
data = [
        (1, "War", "great 3D", 8.9),
        (2, "Science", "fiction", 8.5),
        (3, "Irish", "boring", 6.2),
        (4, "Ice song", "Fantasy", 8.6),
        (5, "House card", "Interesting", 9.1)
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

```
+---+----------+-----------+------+
| id|     movie|description|rating|
+---+----------+-----------+------+
|  1|       War|   great 3D|   8.9|
|  2|   Science|    fiction|   8.5|
|  3|     Irish|     boring|   6.2|
|  4|  Ice song|    Fantasy|   8.6|
|  5|House card|Interesting|   9.1|
+---+----------+-----------+------+
```

OUTPUT

```
+---+----------+-----------+------+
| id|     movie|description|rating|
+---+----------+-----------+------+
|  1|       War|   great 3D|   8.9|
|  5|House card|Interesting|   9.1|
+---+----------+-----------+------+
```

PYSPARK LEARNING HUB : DAY - 39

Step - 3 : Writing the pyspark code to solve

```
from pyspark.sql import SparkSession

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
# Define the schema for the DataFrame
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("movie", StringType(), True),
    StructField("description", StringType(), True),
    StructField("rating", FloatType(), True)
])
# Your data
data = [
    (1, "War", "great 3D", 8.9),
    (2, "Science", "fiction", 8.5),
    (3, "Irish", "boring", 6.2),
    (4, "Ice song", "Fantasy", 8.6),
    (5, "House card", "Interesting", 9.1)
]
# Create a PySpark DataFrame
df = spark.createDataFrame(data, schema=schema)
# Show the DataFrame
df.show()
```

```
+---+----------+-----------+------+
| id|     movie|description|rating|
+---+----------+-----------+------+
|  1|       War|   great 3D|   8.9|
|  2|   Science|    fiction|   8.5|
|  3|     Irish|     boring|   6.2|
|  4|  Ice song|    Fantasy|   8.6|
|  5|House card|Interesting|   9.1|
+---+----------+-----------+------+
```

```python
from pyspark.sql.functions import col

odd_id_df=df.filter( col('id') % 2 == 1 )
odd_id_df.show()

result_df = odd_id_df.filter(col('description') != 'boring')
result_df.show()
```

```
+---+----------+-----------+------+
| id|     movie|description|rating|
+---+----------+-----------+------+
|  1|       War|   great 3D|   8.9|
|  5|House card|Interesting|   9.1|
+---+----------+-----------+------+
```

Step - 1 : Problem Statement

40_Swap Gender Problem

Given a df salary, such as the one below, that has m=male and f=female values. Swap all f and m values (i.e., change all f values to m and vice versa)

Difficult Level : EASY

DataFrame:

```
# Define the schema for the DataFrame
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("sex", StringType(), True),
    StructField("salary", IntegerType(), True),
])

# Define the data
data = [
    (1, "A", "m", 2500),
    (2, "B", "f", 1500),
    (3, "C", "m", 5500),
    (4, "D", "f", 500),
]
```

Step - 2 : Identifying The Input Data And Expected Output

INPUT

```
+---+----+---+------+
| id|name|sex|salary|
+---+----+---+------+
|  1|   A|  m|  2500|
|  2|   B|  f|  1500|
|  3|   C|  m|  5500|
|  4|   D|  f|   500|
+---+----+---+------+
```

OUTPUT

```
+---+----+---+------+
| id|name|sex|salary|
+---+----+---+------+
|  1|   A|  f|  2500|
|  2|   B|  m|  1500|
|  3|   C|  f|  5500|
|  4|   D|  m|   500|
+---+----+---+------+
```

Step - 3 : Writing the pyspark code to solve

```
from pyspark.sql import SparkSession

#creating spark session
spark = SparkSession. \
builder. \
config('spark.shuffle.useOldFetchProtocol', 'true'). \
config('spark.ui.port','0'). \
config("spark.sql.warehouse.dir", "/user/itv008042/warehouse"). \
enableHiveSupport(). \
master('yarn'). \
getOrCreate()
```

```
# Define the schema for the DataFrame
schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("name", StringType(), True),
    StructField("sex", StringType(), True),
    StructField("salary", IntegerType(), True),
])
# Define the data
data = [
    (1, "A", "m", 2500),
    (2, "B", "f", 1500),
    (3, "C", "m", 5500),
    (4, "D", "f", 500),
]
# Create the DataFrame
df = spark.createDataFrame(data, schema=schema)
# Show the DataFrame
df.show()
```

```
+---+----+---+------+
| id|name|sex|salary|
+---+----+---+------+
|  1|   A|  m|  2500|
|  2|   B|  f|  1500|
|  3|   C|  m|  5500|
|  4|   D|  f|   500|
+---+----+---+------+
```

```pyspark
from pyspark.sql.functions import when

result_df=df.withColumn('sex', when(df['sex'] == "m","f").otherwise("m"))

result_df.show()
```

```
+---+----+---+------+
| id|name|sex|salary|
+---+----+---+------+
|  1|   A|  f|  2500|
|  2|   B|  m|  1500|
|  3|   C|  f|  5500|
|  4|   D|  m|   500|
+---+----+---+------+
```