The Godfather of Talent | Futurense

Futurense

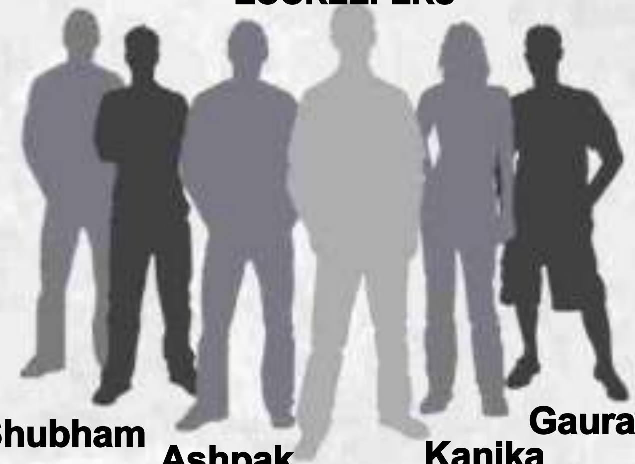
Democratizing Tech Talent
to deliver impact at scale

The Godfather of Talent

Futurense

Credit Card Analysis

The ZOOKEEPERS



Akhil

Shubham

Ashpak

Karthik

Kanika

Gaurav

GUIDE : Venkat Sir

DE Expert

(Sensei)

Business Problem/Overview

In this project, we have 2 problems – Transactions problem & Defaulters problem.

Customer Spending Behavior Analysis

The primary objective of this project is to leverage big data technologies to perform an in-depth analysis of credit card transactions and credit card defaulters datasets.

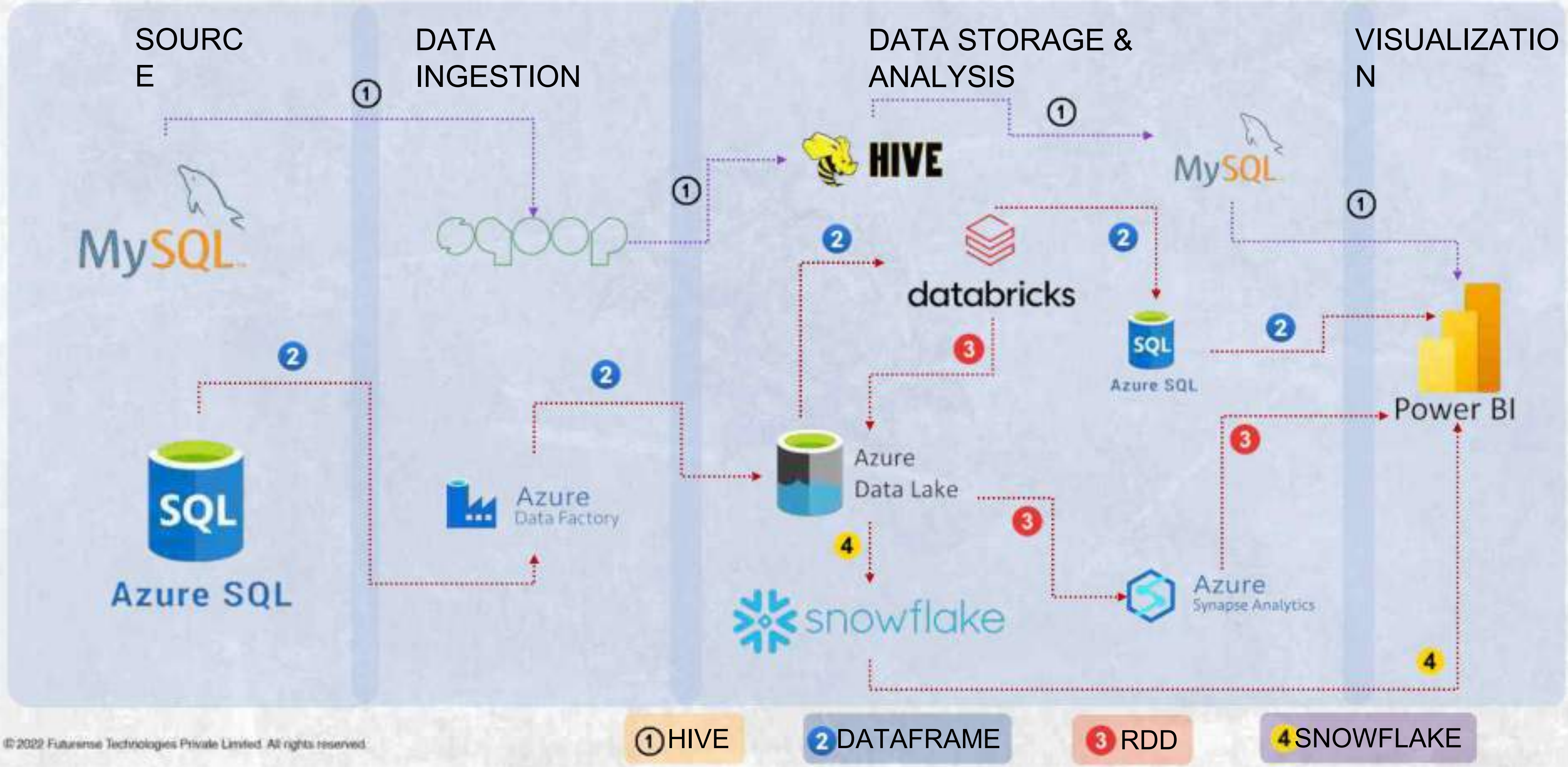
By applying advanced data processing techniques in cloud, this project aims to uncover valuable insights and patterns that can assist in making informed decisions to mitigate credit card default risks and improve overall financial strategies.

- Gain insights into customer spending patterns and behaviors.
- Identify trends and patterns in transaction data.
- Optimize marketing strategies and tailor promotions based on transaction history.
- Enhance customer experiences by understanding their preferences and behaviors.

Credit Card Defaulter Risk Analysis

The outcomes of credit card defaulter analysis empower credit issuers to make informed decisions, manage risks effectively, enhance customer relationships, and optimize their overall business strategies in the dynamic landscape of credit lending. The major goal is to reduce the risk of credit card defaults.

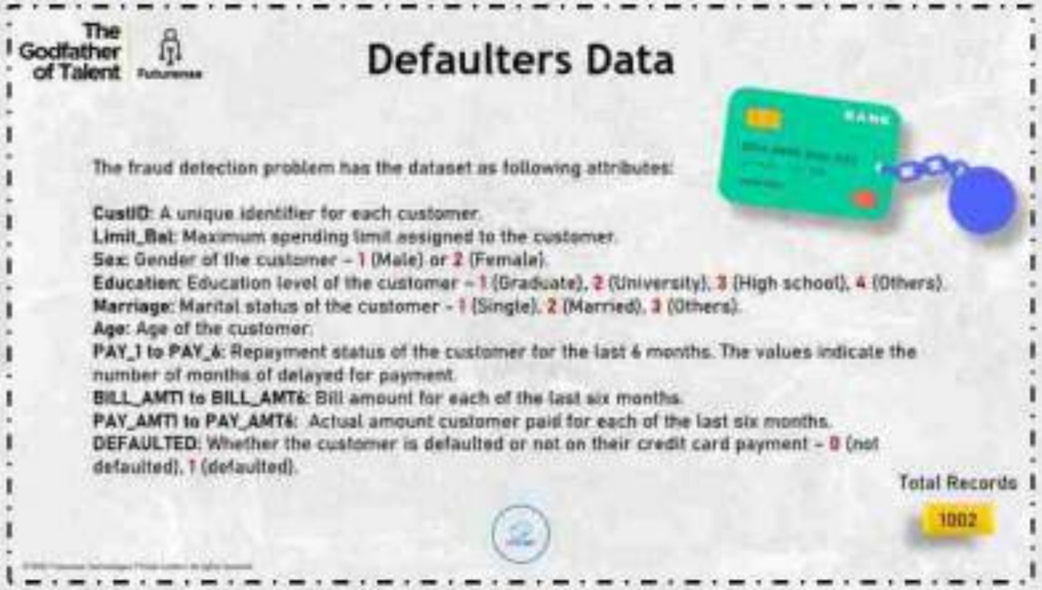
Architecture of the solution



The Godfather of Talent | Futurense

Data Representation

For the given problems, we have 2 different datasets – one for the transactions problem and another for Defaulters problem.



The Godfather of Talent | Futurense

Transactions Data

The transaction problem has the dataset as following features:

Index: A unique identifier for each record in the dataset.
City: The city where transaction was done.
Date: The date on which transaction occurred.
Card Type: Indicates the card type used for transactions – **Silver**, **Gold**, **Platinum**, **Signature**.
Exp Type: Indicates the expense type for which the card was used – **Bills, Entertainment, Food, Fuel, Grocery Travel.**
Gender: Denotes the gender of the cardholder – **Male, Female.**
Amount: The amount of transaction done by the customer.

Out of the given data, Index can be categorized as **Numerical identifier**,
City, Card Type, Exp Type and Gender as **Categorical variables**
Date as **Date variable** & Amount as **Numerical variable**.

Total Records

26052

Defaulters Data

The fraud detection problem has the dataset as following attributes:

CustID: A unique identifier for each customer.
Limit_Bal: Maximum spending limit assigned to the customer.
Sex: Gender of the customer – **1** (Male) or **2** (Female).
Education: Education level of the customer – **1** (Graduate), **2** (University), **3** (High school), **4** (Others).
Marriage: Marital status of the customer - **1** (Single), **2** (Married), **3** (Others).
Age: Age of the customer.
PAY_1 to PAY_6: Repayment status of the customer for the last 6 months. The values indicate the number of months of delayed for payment.
BILL_AMT1 to BILL_AMT6: Bill amount for each of the last six months.
PAY_AMT1 to PAY_AMT6: Actual amount customer paid for each of the last six months.
DEFAULTED: Whether the customer is defaulted or not on their credit card payment – **0** (not defaulted), **1** (defaulted).

Total Records

1002

Batch Processing using HIVE

Creating the table

```
mysql> CREATE TABLE CCD (CUSTID INT,LIMIT_BAL DECIMAL(10, 2),SEX int,EDUCATION int,MARRIAGE int,AGE INT,PAY_1 INT,PAY_2 INT,PAY_3 INT,PAY_4 INT,PAY_5 INT,PAY_6 INT,BILL_AMT1 DECIMAL(10, 2),BILL_AMT2 DECIMAL(10, 2),BILL_AMT3 DECIMAL(10, 2),BILL_AMT4 DECIMAL(10, 2),BILL_AMT5 DECIMAL(10, 2),BILL_AMT6 DECIMAL(10, 2),PAY_AMT1 DECIMAL(10, 2),PAY_AMT2 DECIMAL(10, 2),PAY_AMT3 DECIMAL(10, 2),PAY_AMT4 DECIMAL(10, 2),PAY_AMT5 DECIMAL(10, 2),PAY_AMT6 DECIMAL(10, 2),DEFAULTED INT);
```

Using the following command we will load the data in the table created

```
LOAD DATA INFILE '/home/cloudera/CCD_ProcessedData.csv' INTO TABLE CreditCardData FIELDS
TERMINATED BY ',' (@var1,@var2,@var3,@var4,@var5,@var6,@var7) SET Date=STR_TO_DATE(@var1,'%d
-%m-
%Y'),Low=@var2,Open=@var3,Volume=@var4,high=@var5,Close=@var6,Adjusted _close=@var7;
```

The Godfather of Talent | Futurense

Batch Processing using HIVE

Transfer data using Sqoop

```
[cloudera@quickstart ~]$ sqoop import --connect jdbc:mysql://localhost:3306/project --username root --password clou
dera --table CCD --target-dir /user/cloudera/Shubz/CCD.txt -m 1
```

Create the schema in Hive

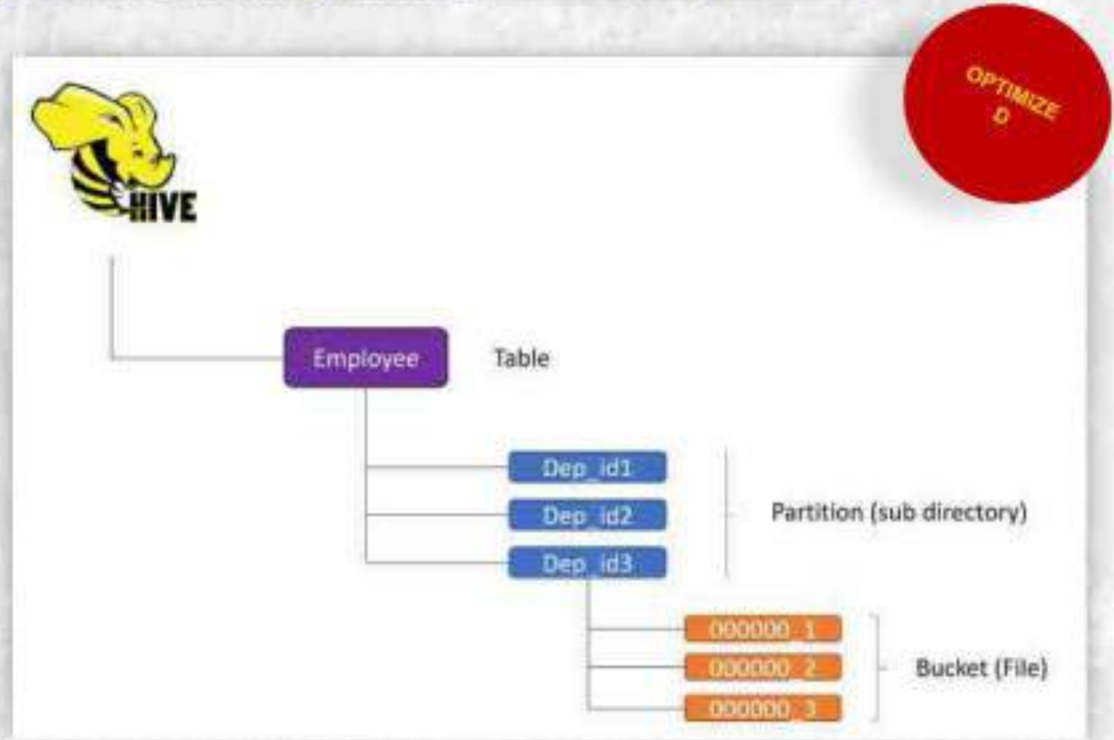
```
hive> create table CCD(CUSTID INT,LIMIT_BAL DECIMAL(10, 2),SEX int,EDUCATION int,MARRIAGE int,AGE INT,PAY_1 INT,PAY_
2 INT,PAY_3 INT,PAY_4 INT,PAY_5 INT,PAY_6 INT,BILL_AMT1 DECIMAL(10, 2),BILL_AMT2 DECIMAL(10, 2),BILL_AMT3 DECIMAL(10
, 2),BILL_AMT4 DECIMAL(10, 2),BILL_AMT5 DECIMAL(10, 2),BILL_AMT6 DECIMAL(10, 2),PAY_AMT1 DECIMAL(10, 2),PAY_AMT2 DEC
IMAL(10, 2),PAY_AMT3 DECIMAL(10, 2),PAY_AMT4 DECIMAL(10, 2),PAY_AMT5 DECIMAL(10, 2),PAY_AMT6 DECIMAL(10, 2),DEFAULTE
D INT) row format delimited fields terminated by ',';
```

Batch Processing using HIVE

Create partitioning on Hive table

```
hive> create table CCDP(CUSTID INT,LIMIT_BAL DECIMAL(10, 2),SEX int,EDUCATION int,MARRIAGE int,AGE INT,PAY_1 INT,PAY
_2 INT,PAY_3 INT,PAY_4 INT,PAY_5 INT,PAY_6 INT,BILL_AMT1 DECIMAL(10, 2),BILL_AMT2 DECIMAL(10, 2),BILL_AMT3 DECIMAL(1
0, 2),BILL_AMT4 DECIMAL(10, 2),BILL_AMT5 DECIMAL(10, 2),BILL_AMT6 DECIMAL(10, 2),PAY_AMT1 DECIMAL(10, 2),PAY_AMT2 DE
CIMAL(10, 2),PAY_AMT3 DECIMAL(10, 2),PAY_AMT4 DECIMAL(10, 2),PAY_AMT5 DECIMAL(10, 2),PAY_AMT6 DECIMAL(10, 2)) partit
ioned by (DEFAULTED INT) row format delimited fields terminated by ',';
```

Partitioning is one of the optimization techniques used in Hive to improve the performance of the query



The Godfather of Talent | Futurense

Batch Processing using HIVE

1. Write a SQL query to determine the count of defaulted (1) and non-defaulted (0) records for both males and females in the Credit Card Data table.

```
SELECT SEX,DEFAULTED,COUNT(*) AS COUNT FROM CCDP
GROUP BY SEX, DEFAULTED;
```

```
mysql> select * from statement6;
```

Sex	DEFAULTED	Total_count
2	0	373
2	1	218
0	0	1
1	1	185
1	0	224

5 rows in set (0.01 sec)

2. average billed Amount & average pay amount

```
SELECT CUSTID, SEX, EDUCATION, MARRIAGE, AGE,
AVG((BILL_AMT1 + BILL_AMT2 + BILL_AMT3 + BILL_AMT4 +
BILL_AMT5 + BILL_AMT6) / 6) AS AVERAGE_BILLED_AMOUNT,
AVG((PAY_AMT1 + PAY_AMT2 + PAY_AMT3 + PAY_AMT4 +
PAY_AMT5 + PAY_AMT6) / 6) AS AVERAGE_PAY_AMOUNT FROM
CCDP GROUP BY CUSTID,SEX,EDUCATION,MARRIAGE,AGE;
```

```
mysql> select * from statement78 limit 10;
```

CUSTID	SEX	EDUCATION	MARRIAGE	AGE	AVERAGE_BILLED_AMOUNT	AVERAGE_PAY_AMOUNT
502	2	2	1	40	54306.50	3233.33
255	2	2	2	30	10500.00	4166.67
750	2	2	2	30	1699.67	1636.33
0	0	0	0	0	0.00	0.00
256	1	2	1	30	4673.17	4654.50
257	2	2	1	50	67273.00	2201.67
258	2	2	1	30	32185.83	2862.83
259	2	3	1	40	60358.67	2300.00
260	1	1	1	50	39685.67	1295.33
261	2	1	2	30	56231.83	22051.83

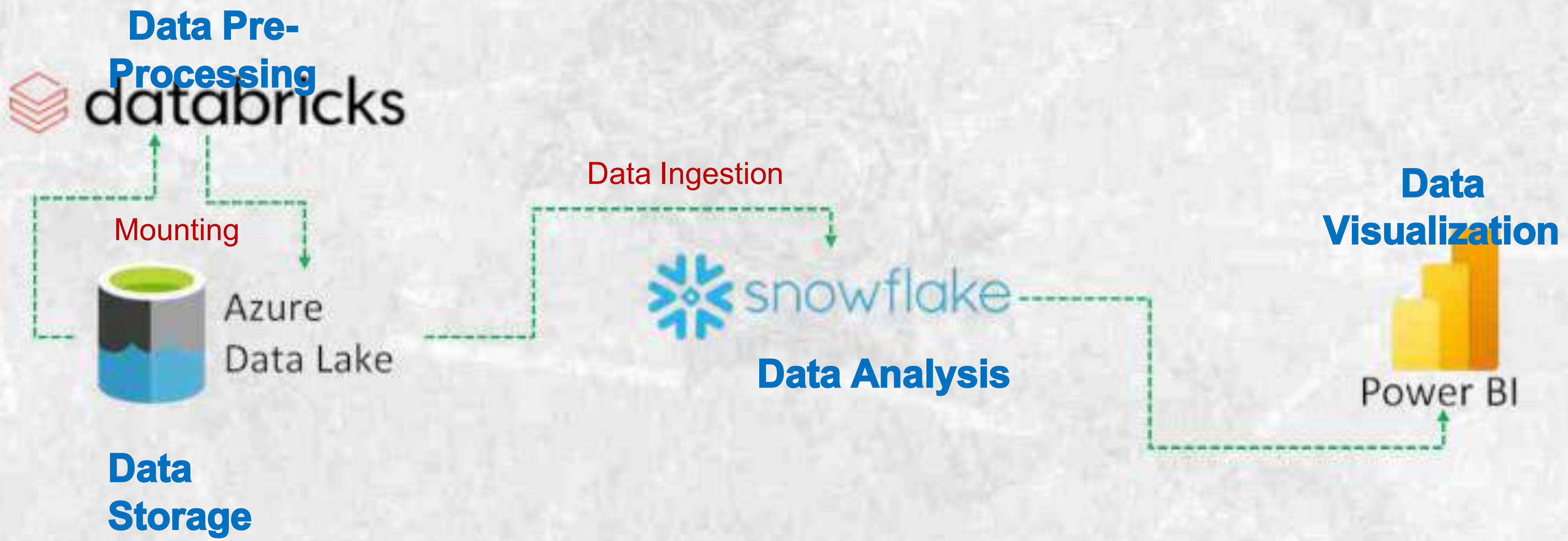
10 rows in set (0.01 sec)

The Godfather of Talent

Futurense

snowflake

Risk Analysis using snowflake



Risk Analysis using snowflake

ADLS MOUNTING TO DATABRICKS :

Connecting ADLS with Databricks

Generate SAS Token from ADLS ACCOUNT and Paste it here

```
SAS_Token = "[illegible]
```

Connect with ADLS account via mounting to load the data

```
dbutils.fs.mount(
    source = '[illegible]',
    mount_point = '/mnt/mounted_SAS',
    extra_configs={
        "[illegible]":SAS_Token
    }
)
```

Data Pre-processing

After completing the thorough data cleaning and preprocessing procedures, the processed and refined data is transferred to the Azure Data Lake Factory

Merge two part files in 1 using repartition

```
df3=ccard_df.repartition(1)
```

Save the processed data in ADLS

```
df3.write.format("csv").mode("overwrite").save("/mnt/mounted_SAS/CCdefaulter_csv",header=True)
```

```
dbutils.fs.unmount('/mnt/mounted_SAS')
```

```
/mnt/mounted_SAS has been unmounted.
Out[14]: True
```

Risk Analysis using snowflake

Data Pre-Processing : **In the initial five scenarios, the data cleaning and preprocessing tasks are executed within the Databricks environment**

Load the file into RDD

```
data = sc.textFile('/mnt/mounted_SAS/credit-card-default-1000.csv')
```

Seperating Header

```
headers= data.first()
```

```
headers1 = [name for name in headers.split(',')]
```

Convert to DataFrame

```
ccard_df = cleaned_rdd.toDF(headers1)
```

Removing header and unwanted inwerted comma , splitting columns

```
#removing header and unwanted inverted comma , splitting columns
rdd_split = data.filter(lambda x : x!=headers).map(lambda x : x.replace('"',"")).map(lambda x : x.split(','))
```

Removing Lines that are not CSV

```
rdd_partial = rdd_split.filter(lambda x : x[0].isdigit())
```

Normalize sex to only 1 and 2

```
cleaned_rdd = rdd_partial.map(lambda x : [int(i.replace('M','1').replace('F','2')) for i in x])
```

Risk Analysis using snowflake

Data Ingestion : **Data Ingestion Pipeline from Data Lake Storage to Snowflake via External Stage and COPY INTO. From ADLS, the data is then seamlessly moved to Snowflake, a data warehouse platform, in order to facilitate subsequent in-depth analysis and exploration**

Connecting with SnowSQL

```
C:\Users\Miles>snowsql -a gfsysnv-ep39977 -u KANIKAAGG
Password:
* SnowSQL * v1.2.28
Type SQL statements or !help
KANIKAAGG#COMPUTE_WH@(no database).(no schema)>use ZOODATABASE;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
1 Row(s) produced. Time Elapsed: 18.938s
KANIKAAGG#COMPUTE_WH@ZOODATABASE.PUBLIC>use ZOODATABASE.ZOOSCHEMA;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
1 Row(s) produced. Time Elapsed: 0.624s
```

Create File Format

```
KANIKAAGG#COMPUTE_WH@ZOODATABASE.ZOOSCHEMA>CREATE OR REPLACE File Format zoo_csv_format
                                          type=csv
                                          field_delimiter=','
                                          skip_header=1
                                          null_if=('NULL','null')
                                          empty_field_as_null=true;
+------------------------------------------------+
| status                                         |
|------------------------------------------------|
| File format ZOO_CSV_FORMAT successfully created. |
+------------------------------------------------+
1 Row(s) produced. Time Elapsed: 0.270s
```


[redacted]

Risk Analysis using snowflake

Create External Stage

```
KANIKAAGG#COMPUTE_[illegible]ABASE.ZOOSCHEMA>CREATE OR REPLACE STAGE zoo_azure_stage
                                            URL = '[illegible]'
                                            CREDENTIALS = (AZURE_SAS_TOKEN = '[illegible]')
                                            FILE_FORMAT = zoo_csv_format;

+-----------------------------------------------+
| status                                        |
|-----------------------------------------------|
| Stage area ZOO_AZURE_STAGE successfully created. |
+-----------------------------------------------+
1 Row(s) produced. Time Elapsed: 0.247s
```

Copy Into Snowflake Internal Table

```
KANIKAAGG#COMPUTE_WH@ZOODATA[illegible]CHEMA>COPY INTO cc_default
                                              FROM @zoo_azure_stage
                                              FILE_FORMAT = zoo_csv_format;
+--------------------------------------------------------------------------------------------------------------------------------
+--------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------+
| file
rsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|--------------------------------------------------------------------------------------------------------------------------------
--------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------|
| azure://zooadlsaccount.blob.core.windows.net/cont1/CCdefaulter_csv/part-00000-tid-739541815169323954-5a2e6ce4-b86e-4cc6-ae2d-082337fa0
1000 |        1000 |           1 |           0 | NULL        |             NULL |                  NULL | NULL                    |
+--------------------------------------------------------------------------------------------------------------------------------
+--------+-------------+-------------+-------------+-------------+------------------+-----------------------+-------------------------+
1 Row(s) produced. Time Elapsed: 2.623s
```

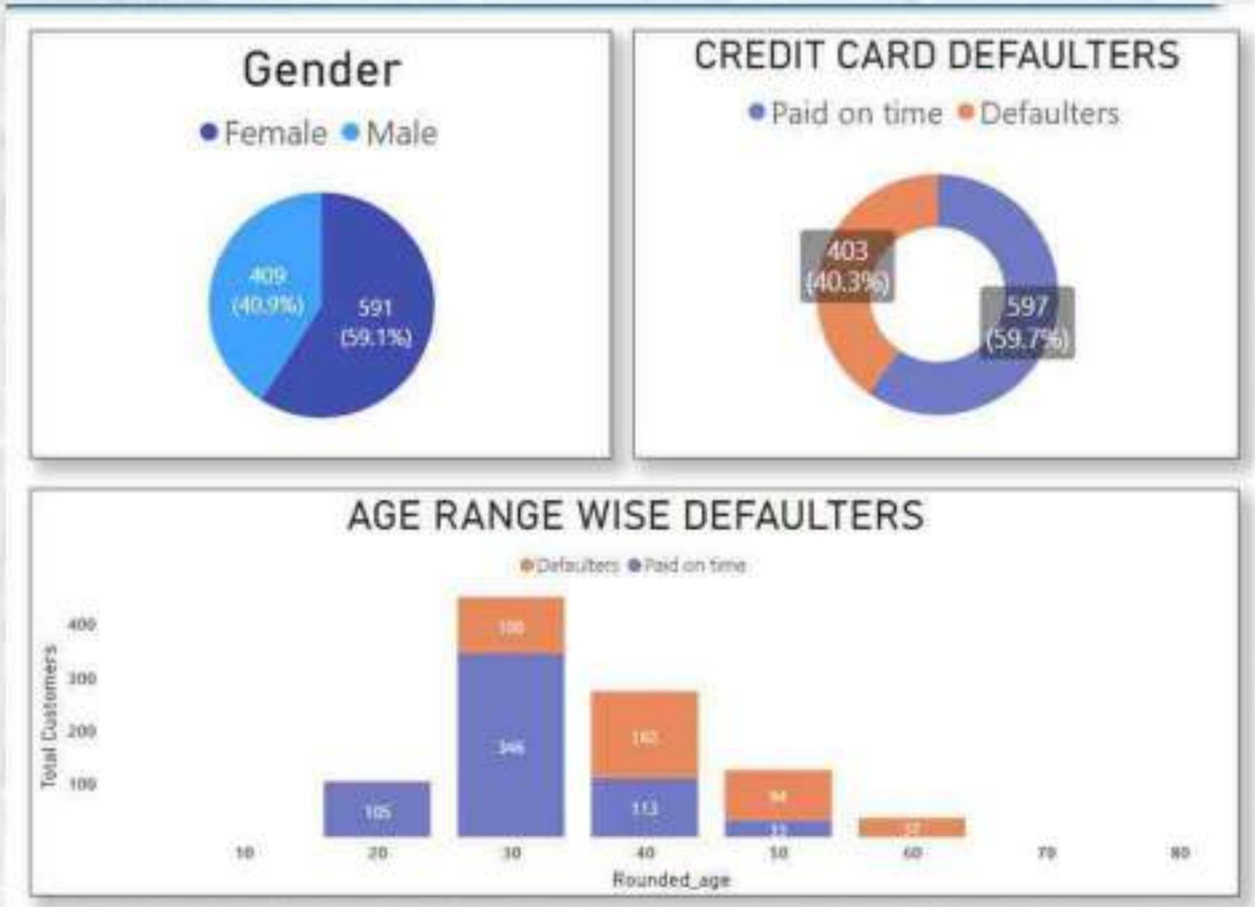
Risk Analysis using snowflake

The Godfather of Talent | Futurense

Data Processing/ Analysis : **Rounding of age to range of 10s and add SEXNAME to the data using SQL Joins.**

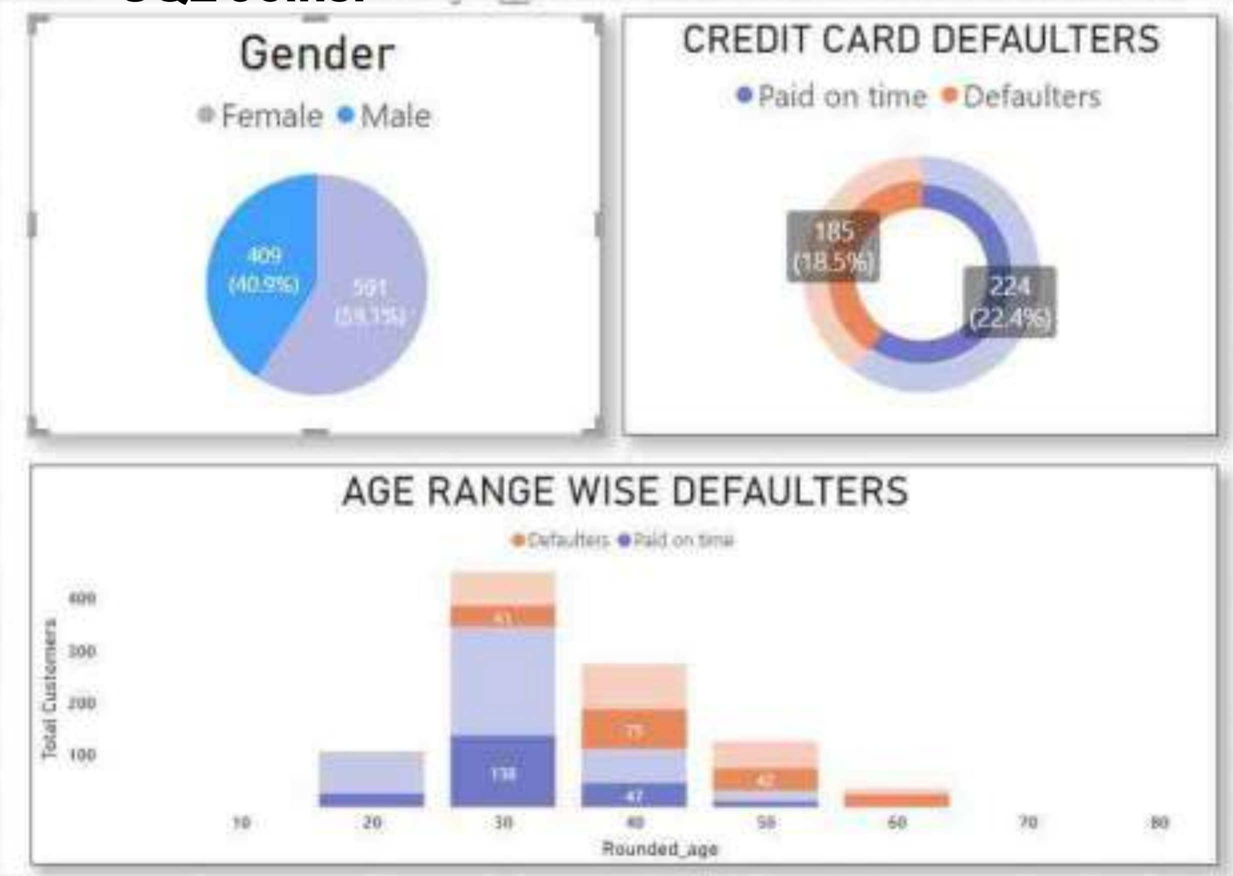
Create Gender table and insert values

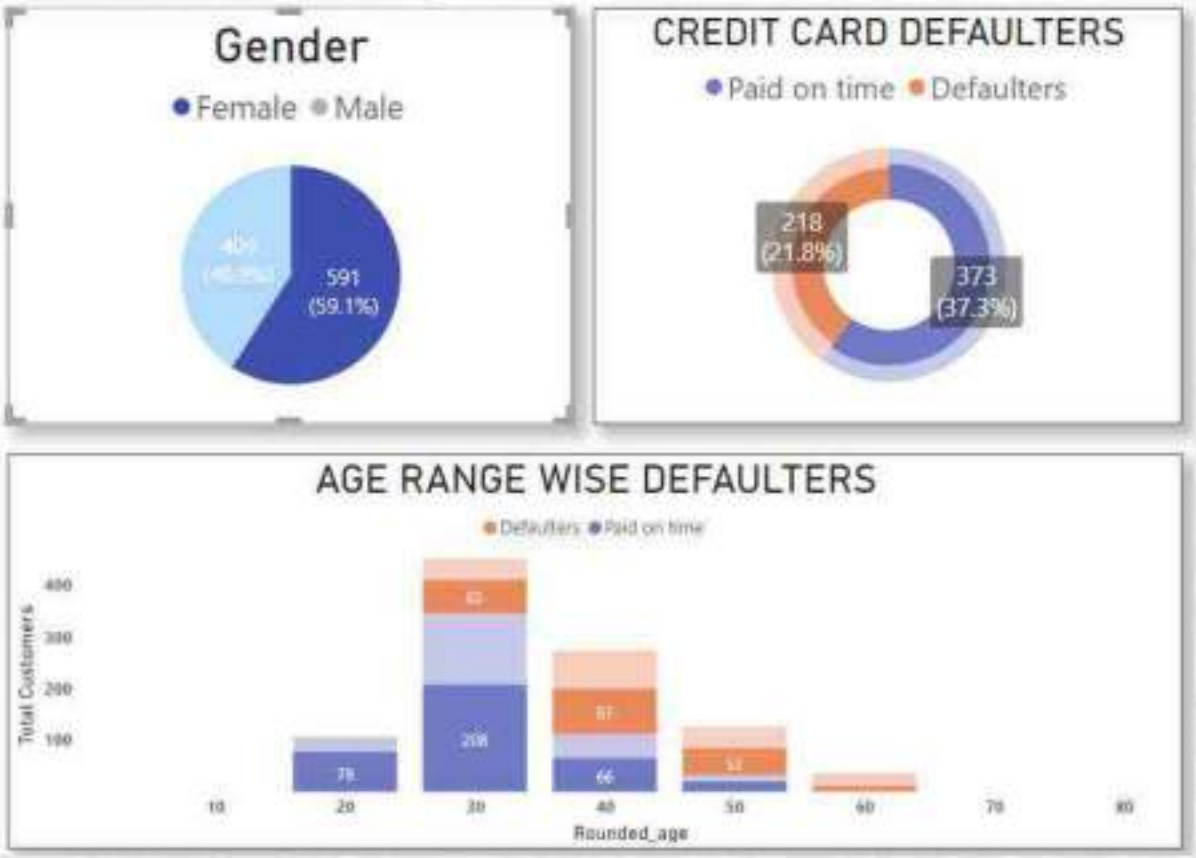
```
Create table gender (
sex_code int,
sex_name varchar(50) );


select *, FLOOR((Age + 5) / 10) * 10 AS RoundedAge
from cc_default c
inner join gender g on c.sex=g.sex_code;
```



Risk Analysis using snowflake

Data Processing/ Analysis : **Rounding of age to range of 10s and add SEXNAME to the data using SQL Joins.**

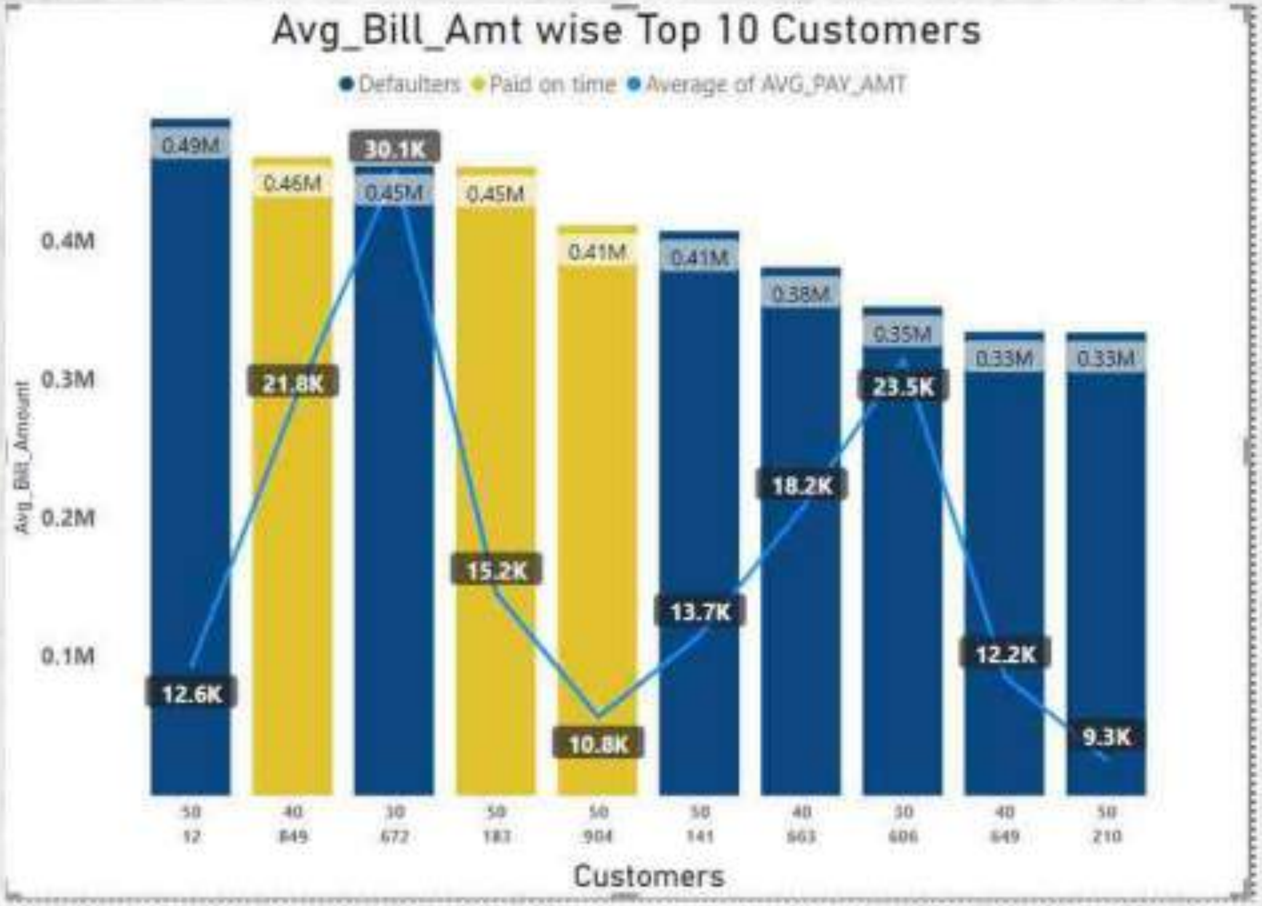




Risk Analysis using snowflake

Data Processing/ Analysis : **Find Average billed Amount and average pay amount for each customer.**

```
select *, ROUND(
(CASE WHEN BILL_AMT1 > 0 THEN BILL_AMT1 ELSE 0 END
+
CASE WHEN BILL_AMT2 > 0 THEN BILL_AMT2 ELSE 0 END
+
CASE WHEN BILL_AMT3 > 0 THEN BILL_AMT3 ELSE 0  END
+
 CASE WHEN BILL_AMT4 > 0 THEN BILL_AMT4 ELSE 0 END
+
 CASE WHEN BILL_AMT5 > 0 THEN BILL_AMT5 ELSE 0 END
+
  CASE WHEN BILL_AMT6 > 0 THEN BILL_AMT6 ELSE 0
END )/6,2)
AS AVG_BILL_AMT,
round((PAY_AMT1+PAY_AMT2+PAY_AMT3+PAY_AMT4+PAY_AMT5+PAY_AMT6)/6,2) AS AVG_PAY_AMT
FROM CC_DEFAULT;
```



Risk Analysis using snowflake

Data Processing/ Analysis : **Find average pay duration. Make sure numbers are rounded and negative values are eliminated.**

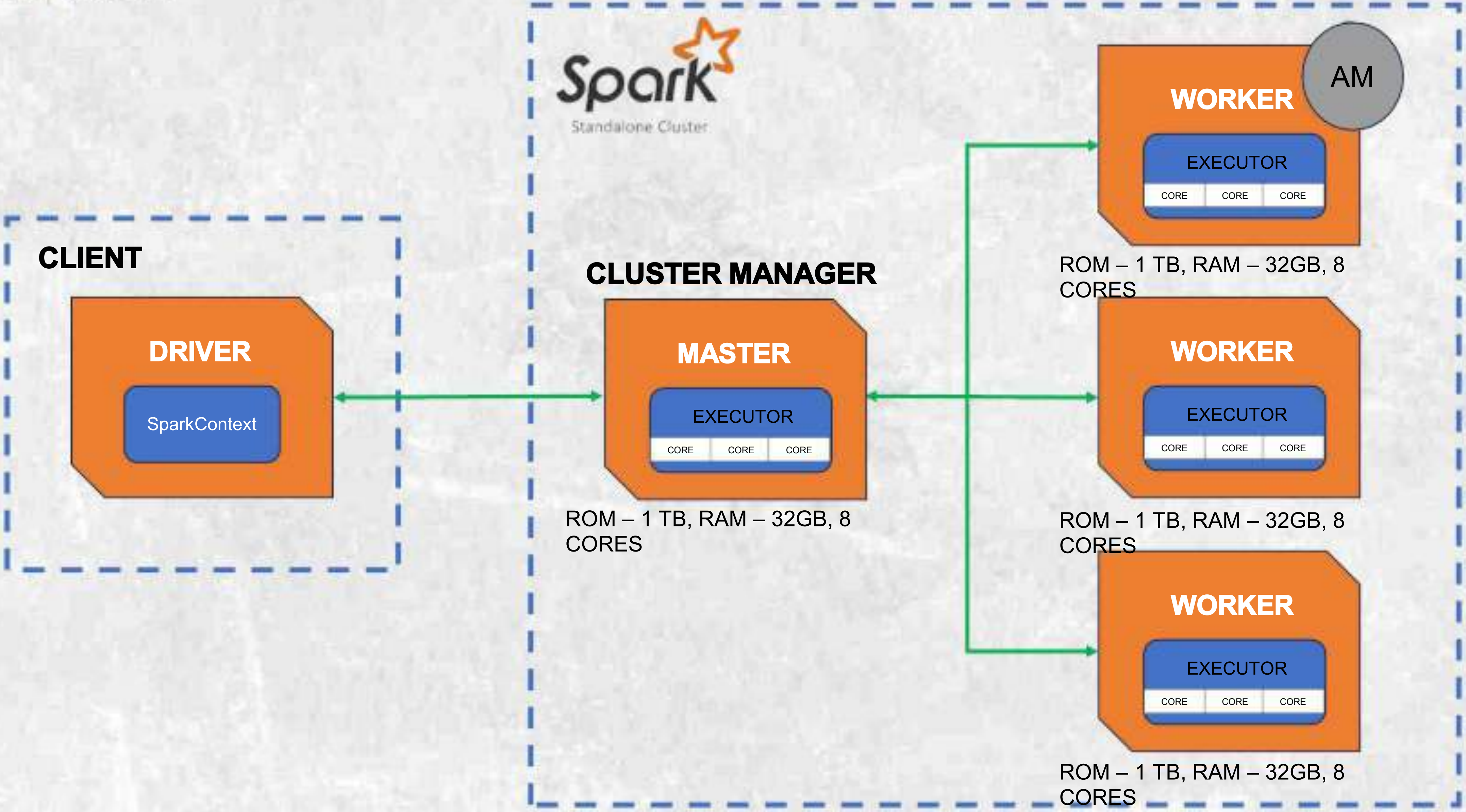
```
SELECT *, ROUND ((
CASE WHEN PAY_1 > 0 THEN PAY_1ELSE 0 END +
 CASE WHEN PAY_2 > 0 THEN PAY_2ELSE 0 END +
 CASE WHEN PAY_3 > 0 THEN PAY_3ELSE 0 END +
 CASE WHEN PAY_4 > 0 THEN PAY_4ELSE 0 END +
 CASE WHEN PAY_5 > 0 THEN PAY_5ELSE 0 END +
 CASE WHEN PAY_6 > 0 THEN PAY_6ELSE 0 END
   )/6, 0) AS AVG_PAY_DURATION
FROM
   CC_DEFAULT;
```

The Godfather of Talent | Futurense

databricks

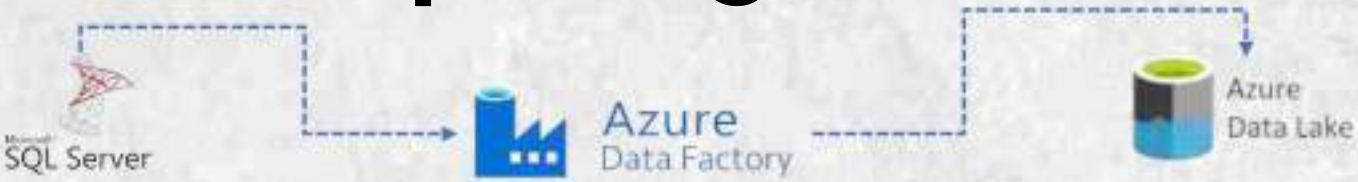
Batch Processing using RDD

Cluster Configuration

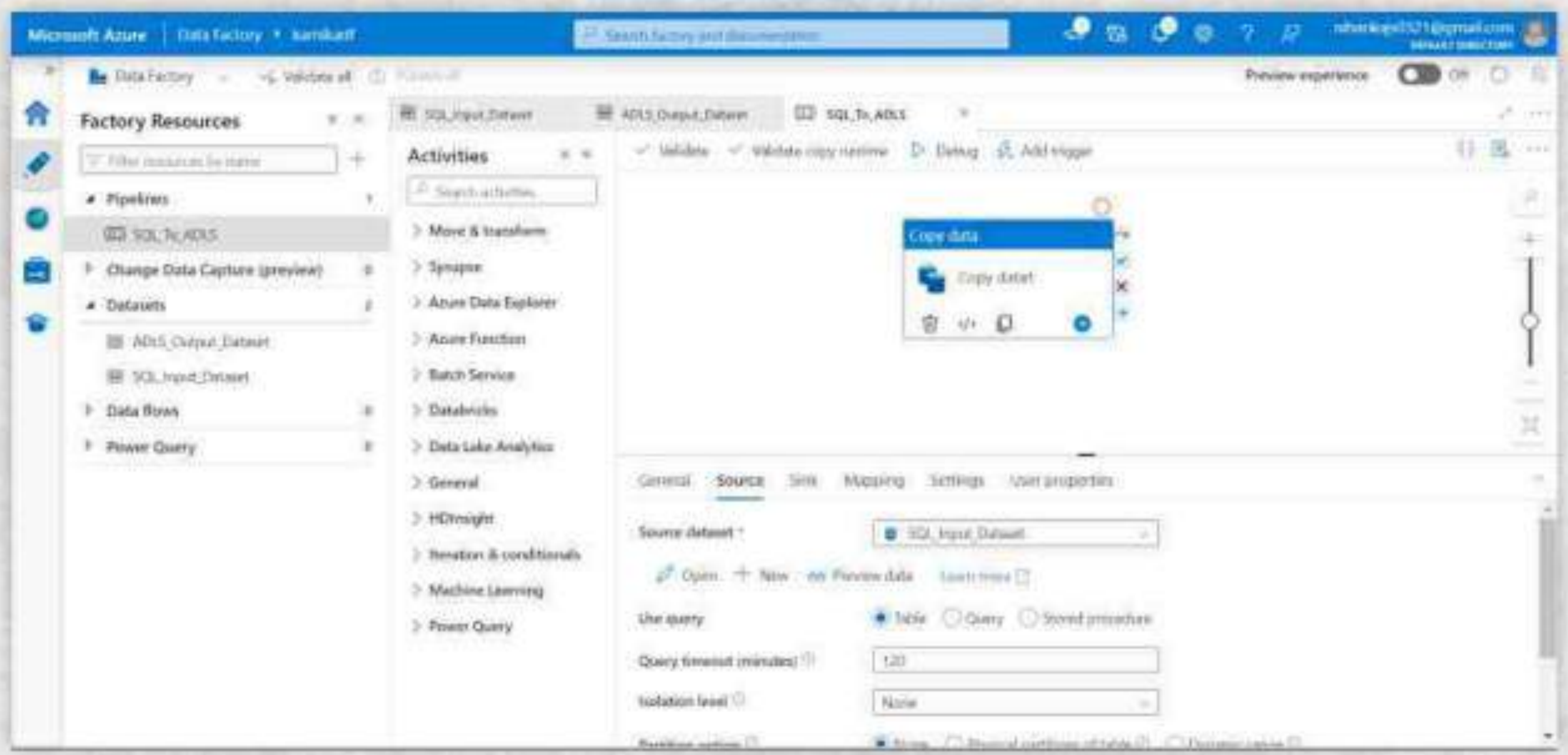


The Godfather of Talent | Futurense

Importing Data



The source for us is MS SQL Server. We are importing the data from the SQL Server into the Azure cloud, using the Azure Data Factory pipeline.



The Godfather of Talent | Futurense

Batch Processing using RDD

databricks

Mounting ADLS

Cmd 3

Python

```python
SAS_Token = "?sv=2022-11-02&ss=bfqt&srt=co&sp=rwdlacupyx&se=2023-08-20T12:48:32Z&st=2023-08-09T04:48:32Z&spr=https&sig=pZN3UpPsvxFn8Fz276yGvAd5u2x8GOl%2B7KxAjl7Ifjc%3D"
```

Command took 0.06 seconds -- by theashpak_5@live.com at 9/8/2023, 10:20:59 am on Ashpak Sheikh's Cluster

Cmd 4

```python

dbutils.fs.mount(
    source = 'wasbs://project@meraaccountdeletekarega.blob.core.windows.net',
    mount_point = '/mnt/mounted_SAS',
    extra_configs={
        "fs.azure.sas.project.meraaccountdeletekarega.blob.core.windows.net":SAS_Token
    }
)
```

Out[9]: True

Command took 10.68 seconds -- by theashpak_5@live.com at 9/8/2023, 10:21:07 am on Ashpak Sheikh's Cluster

Batch Processing using RDD

Write a query to print top 5 cities with highest spends and their percentage contribution of total credit card spends

Cmd 11

```python
# Mapping the data to (City, Amount) pairs
city_amount_rdd = rdd_split.map(lambda row: (row[1], float(row[7])))
# Total spends per city
city_total_spends_rdd = city_amount_rdd.reduceByKey(lambda a, b: a + b)

# Total spends across all cities
total_spends = city_total_spends_rdd.values().sum()

# Total spends in descending order
sorted_cities = city_total_spends_rdd.sortBy(lambda x: x[1], ascending=False)

# Top 5 cities
top_cities = sorted_cities.take(5)

# Percentage contribution of each city's spends
city_spends_percentage = [(city, (amount / total_spends) * 100,amount) for city, amount in top_cities]
# Printing the results
for city,percentage,amount in city_spends_percentage:
    print("City: {}, Amount: {}, Spends: {:.2f}%".format(city, amount,percentage))
```

▸ (4) Spark Jobs

```
City: Greater Mumbai, Amount: 576751476.0, Spends: 14.15%
City: Bengaluru, Amount: 572326739.0, Spends: 14.05%
City: Ahmedabad, Amount: 567794310.0, Spends: 13.93%
City: Delhi, Amount: 556929212.0, Spends: 13.67%
City: Kolkata, Amount: 115466943.0, Spends: 2.83%
```

Command took 1.13 seconds -- by theashpak_5@live.com at 17/8/2023, 11:48:22 am on Ashpak Shaikh's Cluster

Batch Processing using RDD

Write a query to print highest spend month and amount spent in that month for each card type

Cmd 13

Python

```
#'0', 'Delhi', ' India', '29-Oct-14', 'Gold', 'Bills', 'F', '82475'
# Splitting data monthwise
card_month_amount_rdd = rdd_split.map(lambda x: (x[3].split('-')[1], float(x[7])))

# Calculating Monthly Spent
monthly_sum = card_month_amount_rdd.reduceByKey(lambda x,y : x+y).max(lambda x : x[1])

#Getting all the data from month having maximum spent
card_month_amount_rdd = rdd_split.filter(lambda x : x[3].split('-')[1]==monthly_sum[0]).map(lambda x: ((x[4], x[3].split('-')[1]), float(x[7])))

# calculating spent by each card type
monthly_grouped = card_month_amount_rdd.reduceByKey(lambda x,y:x+y)

for card_type, max_month in monthly_grouped.collect():
    print(f"Card Type : {card_type}, Spent : {max_month}")
```

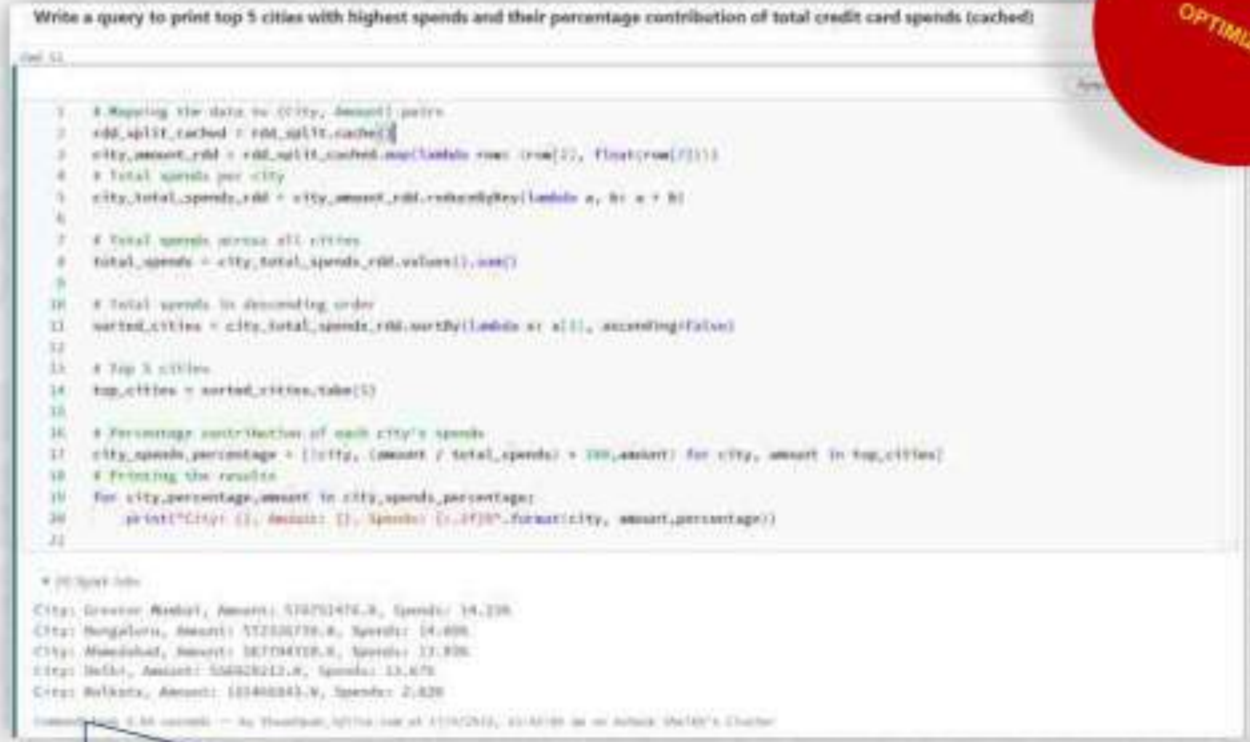
▸ (2) Spark Jobs

```
Card Type : ('Platinum', 'Jan'), Spent : 112784373.0
Card Type : ('Signature', 'Jan'), Spent : 98919381.0
Card Type : ('Silver', 'Jan'), Spent : 109359598.0
Card Type : ('Gold', 'Jan'), Spent : 110146204.0
```

Command took 0.30 seconds -- by theashpak_5@live.com at 3/3/2023, 5:35:34 pm on Ashpak Shaikh's Cluster

Optimization using RDD

OPTIMIZED

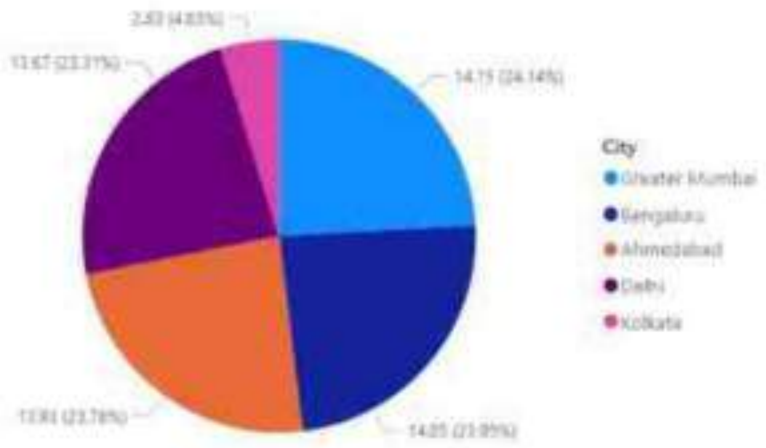


Batch Processing using RDD

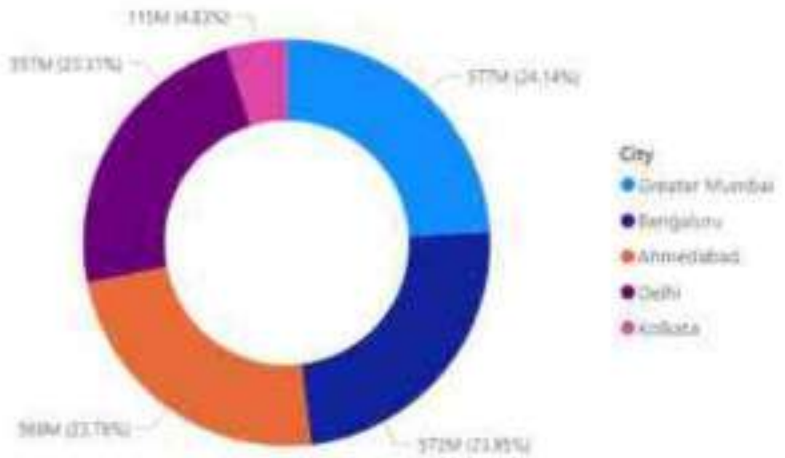
VISUALIZATION :

Write a query to print top 5 cities with highest spends and their percentage contribution of total credit card spends

Percentage Contribution by City



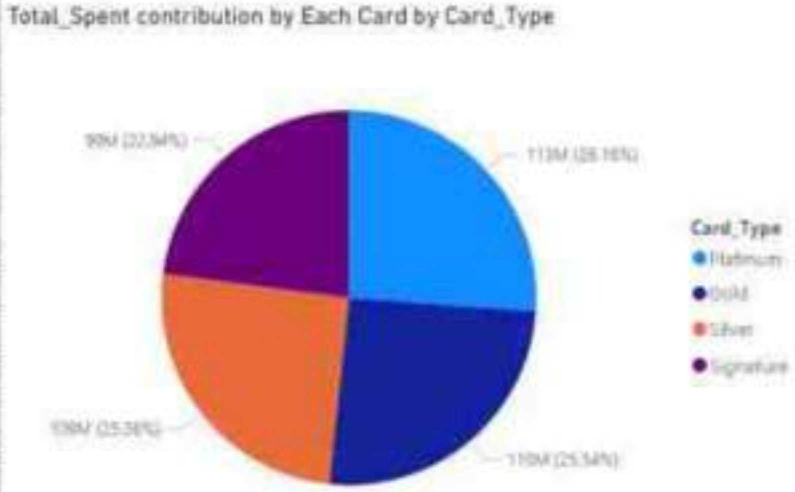
Total Spent By City by City



Batch Processing using RDD

VISUALIZATION:

Write a query to print highest spend month and amount spent in that month for each card type



Batch Processing using RDD

Scenario 6 - Write a query to find percentage contribution of spends by females for each expense type

Cmd 7

```
female_rdd = rdd2.filter(lambda x: x[6] == "F")
# Map the RDD to (expense_type, amount)
expense_type_amount_rdd = female_rdd.map(lambda x: (x[5], float(x[7])))
# Reduce by key to calculate total spend by females for each expense type
total_spend_by_expense_type = expense_type_amount_rdd.reduceByKey(lambda a, b: a + b)
# Collect the total spend by expense type as a dictionary for easy lookup
total_spend_dict = dict(total_spend_by_expense_type.collect())
# Calculate the total spend by all genders for each expense type
total_spend_all_rdd = rdd2.map(lambda x: (x[5], float(x[7]))).reduceByKey(lambda a, b: a + b)
# Calculate the percentage contribution of spends by females for each expense type
percentage_contribution_rdd = total_spend_by_expense_type.join(total_spend_all_rdd).mapValues(lambda x: (x[0] / x[1]) * 100)
# Collect and print the result
result = percentage_contribution_rdd.collect()
for expense_type, percentage in result:
    print(f"Expense Type: {expense_type}, Percentage Contribution: {percentage:.2f}%")
```

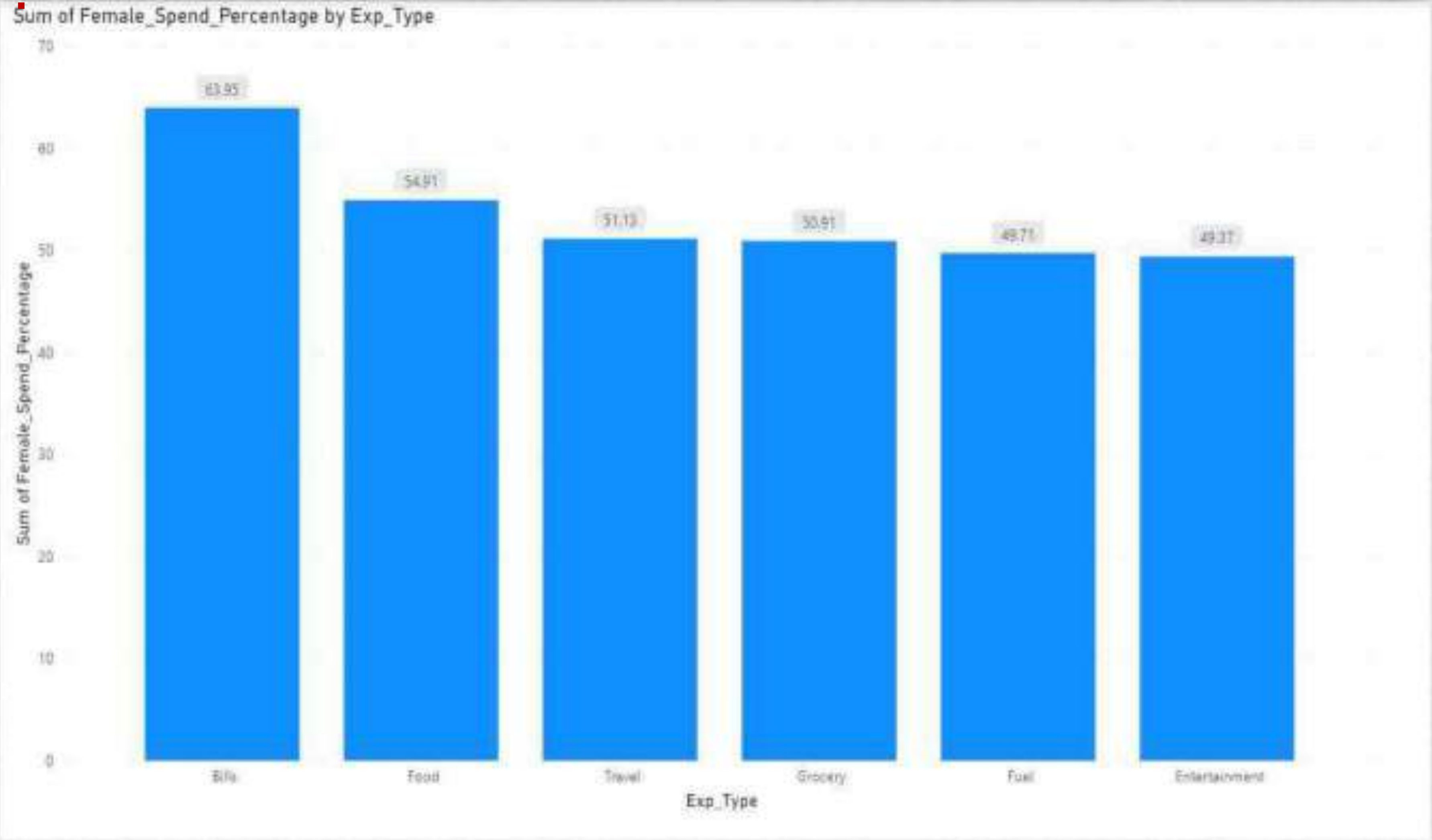
▸ (2) Spark Jobs

```
Expense Type: Bills, Percentage Contribution: 63.95%
Expense Type: Entertainment, Percentage Contribution: 49.37%
Expense Type: Grocery, Percentage Contribution: 50.91%
Expense Type: Fuel, Percentage Contribution: 49.71%
Expense Type: Food, Percentage Contribution: 54.91%
Expense Type: Travel, Percentage Contribution: 51.13%
```

Batch Processing using RDD

VISUALIZATION

:



Batch Processing using RDD

Scenario 7 - Which card and expense type combination saw highest month over month growth in Jan -2014

Cmd 12

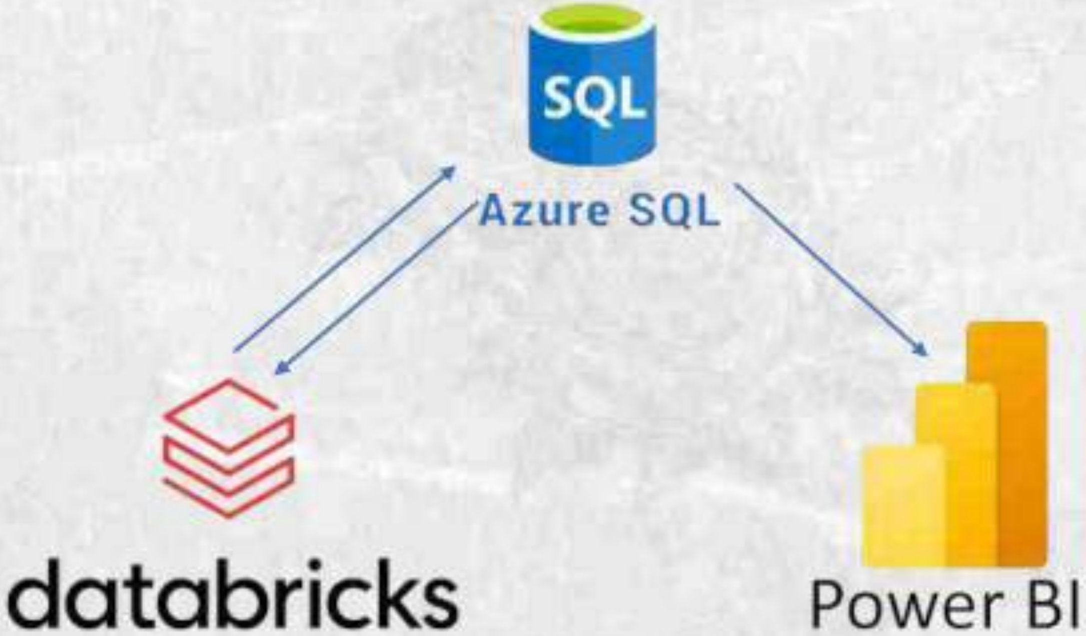
```
# Filter data for January 2014
jan_2014_data_rdd = rddy.filter(lambda x: x[3].split('-',1)[1] ==('Jan-14'))
Dec_2013_data_rdd = rddy.filter(lambda x: x[3].split('-',1)[1] ==('Dec-13'))
# Map the RDD to ((card_type, expense_type), amount)
card_expense_amount_rdd_jan = jan_2014_data_rdd.map(lambda x: ((x[4], x[5]), float(x[7])))
card_expense_amount_rdd_Dec = Dec_2013_data_rdd.map(lambda x: ((x[4], x[5]), float(x[7])))
# Reduce by key to calculate total amount spent for each card and expense type combination
total_amount_by_card_expense_jan = card_expense_amount_rdd_jan.reduceByKey(lambda a, b: a + b)
total_amount_by_card_expense_Dec = card_expense_amount_rdd_Dec.reduceByKey(lambda a, b: a + b)
final_total_amountbycardexpense=total_amount_by_card_expense_jan.join(total_amount_by_card_expense_Dec)
# Calculate the growth from the previous month (December 2013) for each card and expense type combination
growth_rdd = final_total_amountbycardexpense.map(lambda x : (x[0],100*(x[1][0]-x[1][1])/x[1][1]))
# Find the combination with the highest month-over-month growth
max_growth_combination = growth_rdd.max(lambda x: x[1])
for i in max_growth_combination:
    print(i)
```

▸ (1) Spark Jobs

```
('Gold', 'Travel')
87.92008147034576
```

Command took 0.91 seconds -- by niharikajs0321@gmail.com at 8/15/2023, 11:54:49 PM on Niharika J S's Cluster

Interactive Processing using DataFrame



Connecting to Azure SQL Database & Reading the data

Azure SQL Database Configuration

```python
jdbcHostname = "ccaserver.database.windows.net"
jdbcPort = "1433"
jdbcDatabase = "credit_analysis"

url = f"jdbc:sqlserver://{jdbcHostname}:{jdbcPort};database={jdbcDatabase}"
```

▸ data: pyspark.sql.dataframe.DataFrame = [index: short, City: string ... 5 more fields]

Reading Table from Azure SQL Database as DataFrame using Spark Read API

```python
credit_card = spark.read.format("jdbc") \
                .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
                .option("url", url) \
                .option("dbtable", "credit_card_transactions") \
                .option("user", "sqladmin") \
                .option("password", "Password@123") \
                .load()
```

▸ credit_card: pyspark.sql.dataframe.DataFrame = [index: short, City: string ... 5 more fields]

Cleaning Data and Creating Temporary Table

Removing country name from City column

```python
from pyspark.sql.functions import *
# Creating an UDF for removing the country name
func = udf(lambda x: str(x)[:-7])

# Applying UDF function on the column
credit_card = credit_card.withColumn('City', func(col('City')))
```

credit_card: pyspark.sql.dataframe.DataFrame = [index: integer, City: string ... 5 more fields]

Creating temporary table for data frame

```python
credit_card.createOrReplaceTempView('credit_tbl')
```

The Godfather of Talent | Futurense

databricks

Problem Statement

Write a query to print 3 columns: city, highest_expense_type, lowest_expense_type

Scenario 5 - Write a query to print 3 columns: city, highest_expense_type, lowest_expense_type

Cmd 38

```
sc5_out = spark.sql(""" WITH cte AS
                        (SELECT City, Exp_Type, SUM(Amount)
                         FROM credit_tbl
                         GROUP BY City, EXP_TYPE
                         ORDER BY 1, 3 DESC
                        )
                        SELECT DISTINCT City,
                               FIRST_VALUE(Exp_Type) OVER(PARTITION BY City) AS Highest_Expense_Type,
                               LAST_VALUE(Exp_Type) OVER(PARTITION BY City) AS Lowest_Expense_Type
                        FROM cte
                     """)
```

sc5_out: pyspark.sql.dataframe.DataFrame = [City: string, Highest_Expense_Type: string ... 1 more field]

Command took 1.17 seconds -- by niharikajs0321@gmail.com at 8/15/2023, 7:06:06 PM on Niharika J S's Cluster

Problem Statement

Output:

```
1   sc5_out.show()

▸ (5) Spark Jobs

+------------+--------------------+-------------------+
|        City|Highest_Expense_Type|Lowest_Expense_Type|
+------------+--------------------+-------------------+
|   Achalpur|             Grocery|      Entertainment|
|   Adilabad|               Bills|               Food|
|  Adityapur|                Food|            Grocery|
|      Adoni|               Bills|      Entertainment|
|      Adoor|                Fuel|              Bills|
|   Afzalpur|                Fuel|               Food|
|   Agartala|             Grocery|               Food|
|       Agra|               Bills|            Grocery|
|  Ahmedabad|               Bills|            Grocery|
| Ahmednagar|                Fuel|            Grocery|
|     Aizawl|                Food|            Grocery|
|      Ajmer|       Entertainment|               Fuel|
|      Akola|               Bills|               Fuel|
|       Akot|                Fuel|      Entertainment|
|  Alappuzha|                Food|      Entertainment|
|    Aligarh|               Bills|      Entertainment|
| Alipurduar|                Food|      Entertainment|
|  Alirajpur|       Entertainment|      Entertainment|
Command took 4.10 seconds -- by [illegible]
```

Catalyst Optimizer

```
sc5_out.explain(mode="extended")
```

```
== Parsed Logical Plan ==
CTE [cte]
:  +- 'SubqueryAlias cte
:     +- 'Sort [1 ASC NULLS FIRST, 3 DESC NULLS LAST], true
:        +- 'Aggregate ['City, 'EXP_TYPE], ['City, 'Exp_Type, unresolvedalias('SUM('Amount), None)]
:           +- 'UnresolvedRelation [credit_tbl], [], false
+- 'Distinct
   +- 'Project ['City, 'FIRST_VALUE('Exp_Type) windowspecdefinition('City, unspecifiedframe$()) AS Highest_Expense_Type#119, 'LAST_VALUE('Exp_Type) windowspecdefinition('C
ity, unspecifiedframe$()) AS Lowest_Expense_Type#120]
      +- 'UnresolvedRelation [cte], [], false

== Analyzed Logical Plan ==
City: string, Highest_Expense_Type: string, Lowest_Expense_Type: string
WithCTE
:- CTERelationDef 0, false
:  +- SubqueryAlias cte
:     +- Sort [City#111 ASC NULLS FIRST, sum(Amount)#122L DESC NULLS LAST], true
:        +- Aggregate [City#111, EXP_TYPE#63], [City#111, Exp_Type#63, sum(Amount#65) AS sum(Amount)#122L]
:           +- SubqueryAlias credit_tbl
:              +- View (`credit_tbl`, [index#59,City#111,Date#61,Card_Type#62,Exp_Type#63,Gender#64,Amount#65])
:                 +- Project [index#59, <lambda>(City#60)#110 AS City#111, Date#61, Card_Type#62, Exp_Type#63, Gender#64, Amount#65]
```

Command took 0.10 seconds -- by niharikajc0321@gmail.com at 8/16/2023, 11:36:09 PM on Niharika J S's Cluster

OPTIMIZED

The Godfather of Talent | Futurense

databricks

Exporting result to Azure SQL Database

Exporting result DataFrame to Azure SQL Server Table

Cmd 37

```
sc5_out.write.format("jdbc") \
              .option("driver", "com.microsoft.sqlserver.jdbc.SQLServerDriver") \
              .option("url", url) \
              .option("dbtable", "Scenario_5") \
              .option("user", "sqladmin") \
              .option("password", "Password@123") \
              .save()
```

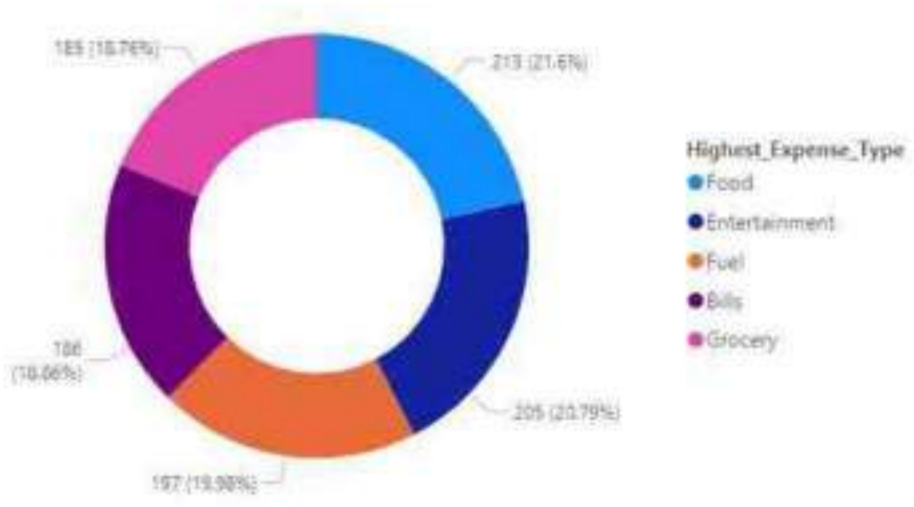
▸ (5) Spark Jobs

Command took 4.59 seconds -- by niharikajs0321@gmail.com at 8/10/2023, 11:39:28 AM on Niharika J S's Cluster

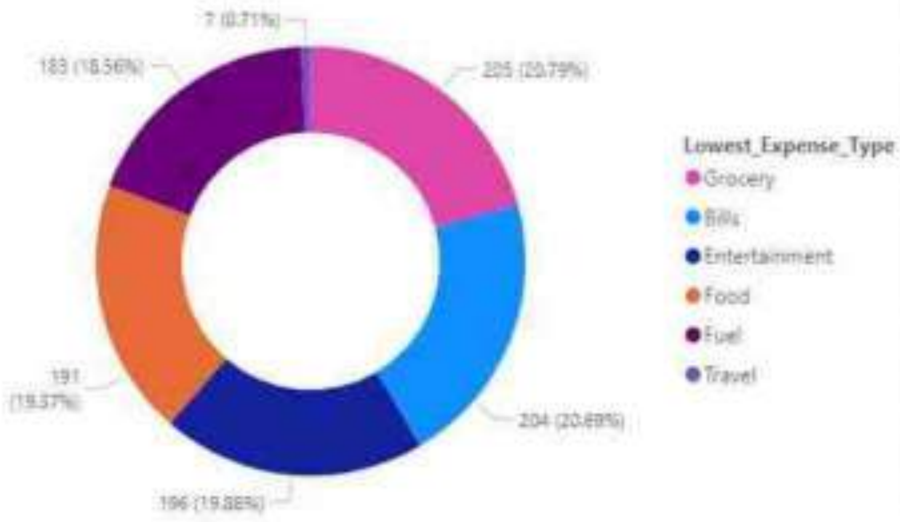
Visualization

Write a query to print 3 columns: city, highest expense type, lowest expense type

Count of City by Highest_Expense_Type



Count of City by Lowest_Expense_Type



Problem Statement

Which city took least number of days to reach its 500th transaction

Scenario 9 - Which city took least number of days to reach its 500th transaction after the first transaction in that city

```
sc9_out = spark.sql(""" WITH cte1 AS
                (SELECT City, Date, ROW_NUMBER() OVER(PARTITION BY City ORDER BY Date) AS RN
                 FROM credit_tbl
                ), cte2 AS
                (SELECT *, LAST_VALUE(RN) OVER(PARTITION BY City) AS Low
                 FROM cte1
                 WHERE RN<=500
                ), cte3 AS
                (SELECT DISTINCT City, DATEDIFF(MAX(Date) OVER(PARTITION BY City), Date) AS Difference
                 FROM cte2 WHERE Low=500
                )
                SELECT City, MAX(Difference) Days_Took
                FROM cte3
                GROUP BY City
                ORDER BY 2 LIMIT 1
              """)
```

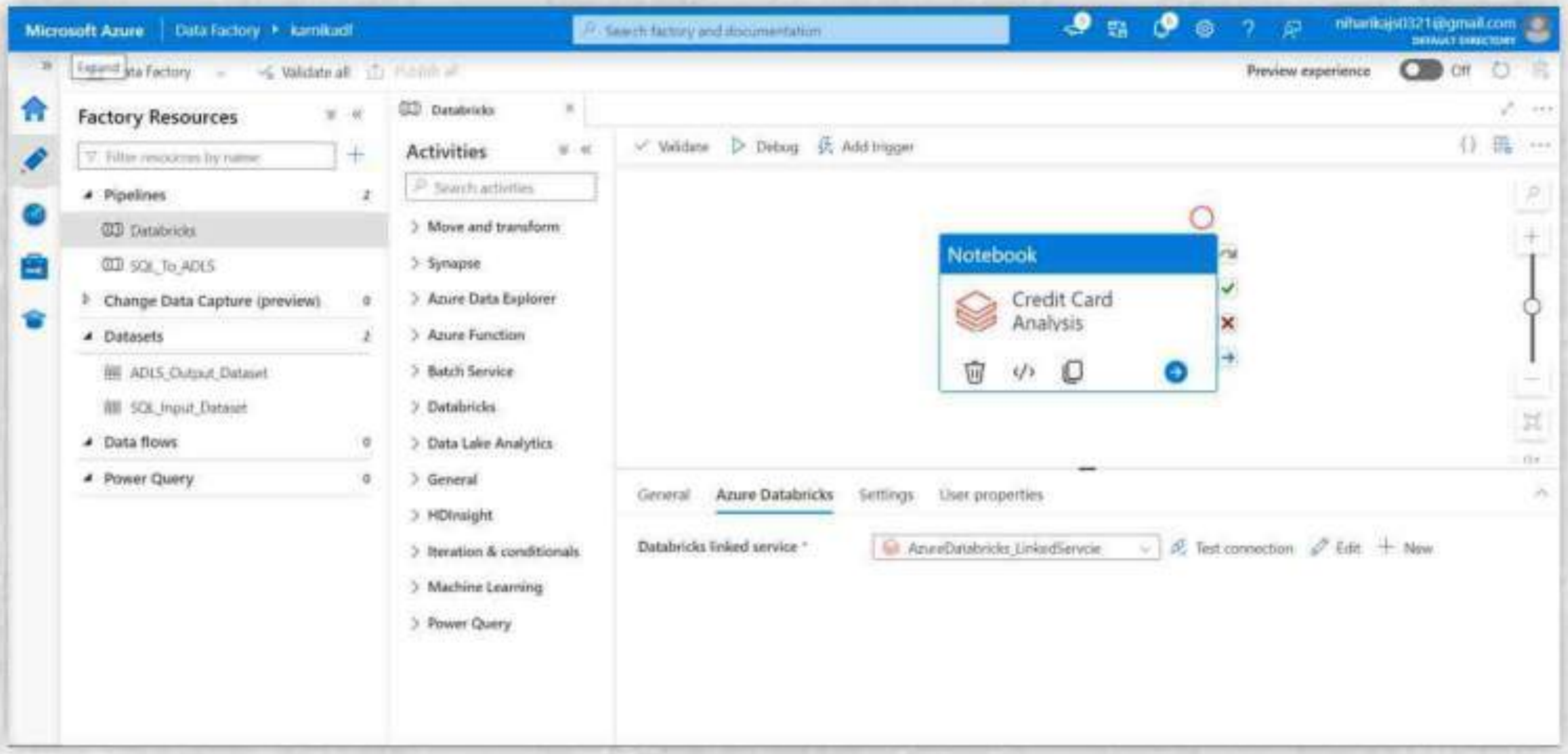
sc9_out: pyspark.sql.dataframe.DataFrame = [City: string, Days_Took: integer]

```
sc9_out.show()
```

(2) Spark Jobs

```
+---------+---------+
|     City|Days_Took|
+---------+---------+
|Bengaluru|       81|
+---------+---------+
```

Scheduling Databricks Notebook in ADF for Periodical Analysis



The Godfather of Talent | Futurense

Roadblocks

- Faced issues with data understanding and handling irrelevant data.

Resolution: Consulting with domain experts who can help you determine which features are important for credit card analysis.

- The automated data ingestion Snowflake pipeline from ADLS to Snowflake encountered challenges due to a region mismatch between the two platforms and unavailability of an enterprise version of Snowflake.

Resolution: Manually loaded the data using COPY INTO and External stage for data ingestion.

Conclusion

- **Customer Insights:** Through in-depth analysis, we gained valuable insights into how customers use their credit cards, revealing spending patterns and preferences.
- **Risk Identification:** Our analysis helped identify potential risks, enabling us to proactively manage and minimize the chances of defaults.
- **Smart Decisions:** Armed with data-driven insights, we can make informed decisions that benefit both customers and the business.
- **Financial Well-being:** By analyzing behavior, we contribute to customers' financial well-being by offering suitable credit limits and advice.

Thank You