A1_snat0021_3

January 21, 2024

1 FIT5202 Assignment 1 : Analysing eCommerce Data

1.1 Table of Contents

2 Part 1 : Working with RDDs

2.1 1.1 Working with RDD

In this section, you will need to create RDDs from the given datasets, perform partitioning in these RDDs and use various RDD operations to answer the queries for retail analysis.

2.1.1 1.1.1 Data Preparation and Loading

Write the code to create a SparkContext object using SparkSession. To create a SparkSession you first need to build a SparkConf object that contains information about your application, use Melbourne time as the session timezone. Give an appropriate name for your application and run Spark locally with as many working processors as logical cores on your machine.

[5]:
```
# Import SparkConf class into the program
from pyspark import SparkConf

# local[*]: run Spark in local mode with as many working processors as logical␣
↪cores on your machine
# If we want Spark to run locally with 'k' worker threads, we can specify as␣
↪"local[k]".
```

```python
master = "local[*]"
# The `appName` field is a name to be shown on the Spark cluster UI page
app_name = "Assignment1"
# Setup configuration parameters for Spark
spark_conf = SparkConf().setMaster(master).setAppName(app_name)

# Import SparkContext and SparkSession classes
from pyspark import SparkContext  # Spark
from pyspark.sql import SparkSession  # Spark SQL

# Method 1: Using SparkSession
spark = SparkSession.builder.config(conf=spark_conf).getOrCreate()
sc = spark.sparkContext
sc.setLogLevel('ERROR')

# Set the time zone for Spark SQL session
spark.conf.set("spark.sql.session.timeZone", "Australia/Melbourne")
```

1.1.2 Load all CSV files into RDDs.

[2]:
```python
# Load RDDs all three csv flies into RDD - Application_data,␣
↪Previous_application and Value_dict
application_data_rdd = sc.textFile('application_data.csv')
previous_application_rdd = sc.textFile('previous_application.csv')
value_dict_rdd = sc.textFile('value_dict.csv')
```

1.1.3 For each RDD, remove the header rows and display the total count and first 10 records. (Hint: You can use csv.reader to parse rows into RDDs.)

3 Application Data

[3]:
```python
# Remove the header row - Application_data_rdd
header_application = application_data_rdd.first()
# The filter method apply a function to each element. The function output is a␣
↪boolean value (TRUE or FALSE)
# Elements that have output TRUE will be kept.
map_exp_rdd_1 = application_data_rdd.filter(lambda x: x != header_application)
# Show 10 records with the Spark *action* take
map_exp_rdd_1.take(10)
```

[3]:
```
['118100,0,2,F,Y,Y,1,247500.0,667237.5,52848.0,576000.0,2,4,3,6,0.018801,-
11258,-1596,13.0,1,1,0,1,0,0,12,3.0,FRIDAY,8,28,0.60994226,0.5884348,,-
733.0,,,,,,',
 '110133,0,2,F,N,Y,2,112500.0,1374480.0,49500.0,1125000.0,8,1,3,6,0.006233,-
11044,-942,,1,1,1,1,0,0,16,4.0,MONDAY,10,42,0.7081764,0.6865754,,0.0,,,,,,',
 '110215,0,2,F,N,Y,0,166500.0,545040.0,26640.0,450000.0,2,1,6,6,0.032561,-
17115,-581,,1,1,0,1,1,0,19,1.0,MONDAY,14,22,0.49497995,0.58477587,0.47225335,-
```

```
1598.0,0.0,0.0,0.0,1.0,0.0,3.0',
 '194051,0,2,F,N,N,0,112500.0,900000.0,24750.0,900000.0,2,1,2,6,0.015221,-
17855,-5470,,1,1,0,1,1,0,8,2.0,FRIDAY,15,30,,0.59620756,0.6195277,-
734.0,0.0,0.0,0.0,0.0,0.0,1.0',
 '110368,0,2,F,N,Y,0,261000.0,1237684.5,47272.5,1138500.0,5,4,3,6,0.020713,-
22818,365243,,1,0,0,1,0,0,18,2.0,FRIDAY,10,31,,0.64156574,0.3996756,-
979.0,0.0,0.0,0.0,0.0,0.0,0.0',
 '110498,0,2,F,N,N,0,157500.0,179865.0,11133.0,148500.0,5,1,3,6,0.00496,-
21183,365243,,1,0,0,1,0,0,18,2.0,THURSDAY,14,31,,0.14626195,0.5064842,0.0,0.0,0.
0,0.0,0.0,0.0,4.0',
 '110561,0,2,F,N,Y,1,157500.0,1256400.0,36864.0,900000.0,8,4,6,6,0.018029,-
9537,-182,,1,1,0,1,0,0,10,2.0,TUESDAY,8,39,0.13320908,0.5543784,,-
1810.0,0.0,0.0,0.0,0.0,1.0,0.0',
 '110836,0,2,F,N,N,1,126000.0,454500.0,14791.5,454500.0,8,1,3,6,0.009334,-
13351,-6261,,1,1,1,1,1,0,18,3.0,TUESDAY,13,38,,0.78073716,0.5797274,-
1197.0,0.0,0.0,0.0,0.0,0.0,1.0',
 '110985,0,2,F,N,Y,0,76500.0,454500.0,14791.5,454500.0,8,4,2,6,0.007114,-16847,-
1194,,1,1,0,1,0,0,19,2.0,SUNDAY,15,50,,0.19403037,,-1150.0,,,,,,',
 '109621,0,2,F,N,N,1,67500.0,513531.0,24835.5,459000.0,2,1,3,6,0.008068,-10828,-
2693,,1,1,0,1,0,0,10,3.0,THURSDAY,12,28,0.42454174,0.17806706,0.59892625,0.0,0.0
,0.0,0.0,0.0,0.0,2.0']
```

[4]:
```python
# Count the total rows - Application_data_rdd
map_exp_rdd_1.count()
```

[4]:
```
172591
```

4 Previous Application

[5]:
```python
# Remove the header row - previous application
header_application_2 = previous_application_rdd.first()
# The filter method apply a function to each element. The function output is a
 ↪boolean value (TRUE or FALSE)
# Elements that have output TRUE will be kept.
map_exp_rdd_2 = previous_application_rdd.filter(lambda x: x !=
 ↪header_application_2)
# Show 10 records with the Spark *action* take
map_exp_rdd_2.take(10)
```

[5]:
```
['269239,3,,0.0,0.0,,,8,,,,XNA,Canceled,-
207,XNA,XAP,"",Repeater,XNA,XNA,XNA,"6",-1,XNA,,XNA,Cash,,,,,,,65',
 '221473,3,,0.0,0.0,,,8,,,,XNA,Canceled,-
317,XNA,XAP,"",Refreshed,XNA,XNA,XNA,"6",-1,XNA,,XNA,Cash,,,,,,,7817',
 '107678,4,,24480.0,24480.0,0.0,24480.0,12,0.0,,,XAP,Refused,-1252,Cash through
the bank,LIMIT,"",Repeater,Mobile,XNA,XNA,"3",92,Connectivity,,XNA,POS mobile
with interest,,,,,,,172',
```

```
 '168941,4,9580.455,41296.5,46593.0,0.0,41296.5,8,0.0,,,XAP,Approved,-633,Cash
through the bank,XAP,"",New,Mobile,POS,XNA,"3",55,Connectivity,6.0,high,POS
mobile with interest,365243.0,-602.0,-452.0,-452.0,-444.0,0.0,285',
 '204082,3,,450000.0,450000.0,0.0,450000.0,17,0.0,,,XNA,Refused,-
368,XNA,HC,"",Repeater,XNA,XNA,XNA,"3",60,Connectivity,,XNA,Cash,,,,,,,391',
 '148658,2,7875.0,0.0,157500.0,,,9,,,,XAP,Refused,-
419,XNA,HC,"",Repeater,XNA,Cards,x-sell,"8",4,XNA,0.0,XNA,Card
X-Sell,,,,,,,471',
 '190200,3,,0.0,0.0,,,12,,,,XNA,Refused,-
405,XNA,SCOFR,"",Repeater,XNA,XNA,XNA,"6",-1,XNA,,XNA,Cash,,,,,,,691',
 '152739,3,,0.0,0.0,,,6,,,,XNA,Canceled,-
413,XNA,XAP,"",Repeater,XNA,XNA,XNA,"6",-1,XNA,,XNA,Cash,,,,,,,967',
 '265668,3,,0.0,0.0,,,11,,,,XNA,Canceled,-
231,XNA,XAP,"",Repeater,XNA,XNA,XNA,"6",-1,XNA,,XNA,Cash,,,,,,,127572',
 '162831,2,4500.0,90000.0,90000.0,,90000.0,11,,,,XAP,Refused,-
23,XNA,HC,Family,Refreshed,XNA,Cards,x-sell,"4",150,Furniture,0.0,XNA,Card
X-Sell,,,,,,,1302']
```

[6]:
```python
# Count the total rows - previous application
map_exp_rdd_2.count()
```

[6]: 935037

5 Value dictationary

[7]:
```python
# Remove the header row - value_dict_rdd
header_application_3 = value_dict_rdd.first()
# The filter method apply a function to each element. The function output is a␣
↪boolean value (TRUE or FALSE)
# Elements that have output TRUE will be kept.
map_exp_rdd_3 = value_dict_rdd.filter(lambda x: x != header_application_3)
# Show 10 records with the Spark *action* take
map_exp_rdd_3.take(10)
```

[7]:
```
['4,name_type_suite,Other_B,2',
 '5,name_type_suite,Children,3',
 '55,organization_type,Business Entity Type 2,1',
 '56,organization_type,Agriculture,2',
 '57,organization_type,Industry: type 13,3',
 '58,organization_type,Religion,4',
 '59,organization_type,Construction,5',
 '60,organization_type,Police,6',
 '30,housing_type,Rented apartment,1',
 '31,housing_type,Co-op apartment,2']
```

```
[8]: #Count the total rows - value_dict_rdd
map_exp_rdd_3.count()
```

```
[8]: 126
```

1.1.4 Drop the following columns from RDDs:
previous_application: sellerplace_area, name_seller_industry
application_data: All columns start with flag_ and amt_credit_req_(except for amt_credit_req_last_year).

6 Previous Application

```
[35]: #Import the sparkconf and spark context into the program
from pyspark import SparkConf, SparkContext

# Read the CSV file into an RDD
previous_application_rdd = sc.textFile('previous_application.csv').map(lambda␣
 ↪line: line.split(','))

# Define the indices of columns to drop
# [22,23]- Sellerplace_area, name_seller_industry
columns_to_drop = [22, 23]  # Assuming 0-based indexing, replace with your␣
 ↪actual column indices



# Use the map transformation to drop specified columns from each row
previous_application_rdd2 = previous_application_rdd.map(lambda row: [element␣
 ↪for index, element in enumerate(row) if index not in columns_to_drop])
```

```
[36]: #Result of previous_application_rdd2 the drop columns
previous_application_rdd2.take(10)
```

```
[36]: [['"id_app"',
  '"contract_type"',
  '"amt_annuity"',
  '"amt_application"',
  '"amt_credit"',
  '"amt_down_payment"',
  '"amt_goods_price"',
  '"hour_appr_process_start"',
  '"rate_down_payment"',
  '"rate_interest_primary"',
  '"rate_interest_privileged"',
  '"name_cash_loan_purpose"',
  '"name_contract_status"',
  '"days_decision"',
```

```
'"name_payment_type"',
 '"code_rejection_reason"',
 '"name_type_suite"',
 '"name_client_type"',
 '"name_goods_category"',
 '"name_portfolio"',
 '"name_product_type"',
 '"channel_type"',
 '"cnt_payment"',
 '"name_yield_group"',
 '"product_combination"',
 '"days_first_drawing"',
 '"days_first_due"',
 '"days_last_due_1st_version"',
 '"days_last_due"',
 '"days_termination"',
 '"nflag_insured_on_approval"',
 '"id"'],
['269239',
 '3',
 '',
 '0.0',
 '0.0',
 '',
 '',
 '8',
 '',
 '',
 '',
 'XNA',
 'Canceled',
 '-207',
 'XNA',
 'XAP',
 '""',
 'Repeater',
 'XNA',
 'XNA',
 'XNA',
 '"6"',
 '',
 'XNA',
 'Cash',
 '',
 '',
 '',
 '',
```

```
 '',
 '',
 '65'],
['221473',
 '3',
 '',
 '0.0',
 '0.0',
 '',
 '',
 '8',
 '',
 '',
 '',
 'XNA',
 'Canceled',
 '-317',
 'XNA',
 'XAP',
 '""',
 'Refreshed',
 'XNA',
 'XNA',
 'XNA',
 '"6"',
 '',
 'XNA',
 'Cash',
 '',
 '',
 '',
 '',
 '',
 '',
 '7817'],
['107678',
 '4',
 '',
 '24480.0',
 '24480.0',
 '0.0',
 '24480.0',
 '12',
 '0.0',
 '',
 '',
 'XAP',
```

```
 'Refused',
 '-1252',
 'Cash through the bank',
 'LIMIT',
 '""',
 'Repeater',
 'Mobile',
 'XNA',
 'XNA',
 '"3"',
 '',
 'XNA',
 'POS mobile with interest',
 '',
 '',
 '',
 '',
 '',
 '',
 '172'],
['168941',
 '4',
 '9580.455',
 '41296.5',
 '46593.0',
 '0.0',
 '41296.5',
 '8',
 '0.0',
 '',
 '',
 'XAP',
 'Approved',
 '-633',
 'Cash through the bank',
 'XAP',
 '""',
 'New',
 'Mobile',
 'POS',
 'XNA',
 '"3"',
 '6.0',
 'high',
 'POS mobile with interest',
 '365243.0',
 '-602.0',
```

 '-452.0',
 '-452.0',
 '-444.0',
 '0.0',
 '285'],
['204082',
 '3',
 '',
 '450000.0',
 '450000.0',
 '0.0',
 '450000.0',
 '17',
 '0.0',
 '',
 '',
 'XNA',
 'Refused',
 '-368',
 'XNA',
 'HC',
 '""',
 'Repeater',
 'XNA',
 'XNA',
 'XNA',
 '"3"',
 '',
 'XNA',
 'Cash',
 '',
 '',
 '',
 '',
 '',
 '',
 '391'],
['148658',
 '2',
 '7875.0',
 '0.0',
 '157500.0',
 '',
 '',
 '9',
 '',
 '',

```
  '',
  'XAP',
  'Refused',
  '-419',
  'XNA',
  'HC',
  '""',
  'Repeater',
  'XNA',
  'Cards',
  'x-sell',
  '"8"',
  '0.0',
  'XNA',
  'Card X-Sell',
  '',
  '',
  '',
  '',
  '',
  '',
  '471'],
 ['190200',
  '3',
  '',
  '0.0',
  '0.0',
  '',
  '',
  '12',
  '',
  '',
  '',
  'XNA',
  'Refused',
  '-405',
  'XNA',
  'SCOFR',
  '""',
  'Repeater',
  'XNA',
  'XNA',
  'XNA',
  '"6"',
  '',
  'XNA',
  'Cash',
```

```
 '',
 '',
 '',
 '',
 '',
 '',
 '691'],
['152739',
 '3',
 '',
 '0.0',
 '0.0',
 '',
 '',
 '6',
 '',
 '',
 '',
 'XNA',
 'Canceled',
 '-413',
 'XNA',
 'XAP',
 '""',
 'Repeater',
 'XNA',
 'XNA',
 'XNA',
 '"6"',
 '',
 'XNA',
 'Cash',
 '',
 '',
 '',
 '',
 '',
 '',
 '967'],
['265668',
 '3',
 '',
 '0.0',
 '0.0',
 '',
 '',
 '11',
```

```
 '',
 '',
 '',
 'XNA',
 'Canceled',
 '-231',
 'XNA',
 'XAP',
 '""',
 'Repeater',
 'XNA',
 'XNA',
 'XNA',
 '"6"',
 '',
 'XNA',
 'Cash',
 '',
 '',
 '',
 '',
 '',
 '',
 '127572']]
```

7 Application Data

[15]:
```
# Drop columns from application_data RDD
application_data_rdd = sc.textFile('application_data.csv').map(lambda line:␣
↪line.split(','))

# Define columns to drop based on specified conditions - column that start with␣
↪flag_, amt_credit_req(except for amt_credit_req_last_year)
columns_to_drop_application_data = [col for col in header_application.
↪split(',') if col.startswith('flag_') or (col.startswith('amt_credit_req_')␣
↪and col != 'amt_credit_req_last_year')]

# Filter out specified columns from each row
application_data_rdd1 = application_data_rdd.map(lambda x: ','.join([x.
↪split(',')[i] for i in range(len(header_application.split(','))) if␣
↪header_application.split(',')[i] not in columns_to_drop_application_data]))
```

[ ]:
```
#Result of application_data_rdd1 to see the drop columns
application_data_rdd1.take(10)
```

7.0.1 1.2 Data Partitioning in RDD

1.2.1 For each RDD, print out the total number of partitions and the number of records in each partition. Answer the following questions:
How many partitions do the above RDDs have?
How is the data in these RDDs partitioned by default, when we do not explicitly specify any partitioning strategy?
Can you explain why it will be partitioned in this number? If I only have one single-core CPU on my PC, what is the default partition's number? (Hint: search the Spark source code to try to answer this question.)
Write the code and your explanation in Markdown cells.

[17]:
```
# Using Spark, we can read and load a csv file
# Read csv file and load into an RDD object - application_data
application_data_rdd = sc.textFile('application_data.csv')

## Exploring the data file, we can see that it contains different types of␣
↪information
## Some useful information is printed below
print(f"Total partitions: {application_data_rdd.getNumPartitions()}")
print(f"Number of lines: {application_data_rdd.count()}")
```

```
Total partitions: 2
Number of lines: 172592
```

Explain:the total no of partition in application data is 2 and number of lines is 172592

[18]:
```
# Using Spark, we can read and load a csv file
# Read csv file and load into an RDD object - previous_application
previous_application_rdd = sc.textFile(application_data_rdd1 .csv')

## Exploring the data file, we can see that it contains different types of␣
↪information
## Some useful information is printed below
print(f"Total partitions: {previous_application_rdd.getNumPartitions()}")
print(f"Number of lines: {previous_application_rdd.count()}")
```

```
Total partitions: 6
Number of lines: 935038
```

Explain: the total number of partition in previous application is 6 and number of lines is 935038

[19]:
```
# Using Spark, we can read and load a csv file
# Read csv file and load into an RDD object - value_dict
value_dict_rdd = sc.textFile('value_dict.csv')

## Exploring the data file, we can see that it contains different types of␣
↪information
## Some useful information is printed below
```

```
print(f"Total partitions: {value_dict_rdd.getNumPartitions()}")
print(f"Number of lines: {value_dict_rdd.count()}")
```

```
Total partitions: 2
Number of lines: 127
```

Explain:the total number of partition in value dictationary is 2 and number of line is 127

8 Default Partition

[44]:
```
# create an RDD - application_data_rdd
application_data_rdd = sc.textFile('application_data.csv')
#default partition for application_data_rdd
print('Default partitions: ',application_data_rdd.getNumPartitions())
```

```
Default partitions:  2
```

[42]:
```
# Create an RDD - previous_application
previous_application_rdd = sc.textFile('previous_application.csv')
#default partition for previous_application
print('Default partitions: ', previous_application_rdd.getNumPartitions())
```

```
Default partitions:  6
```

[43]:
```
# create an RDD - value_dict_rdd
value_dict_rdd = sc.textFile('value_dict.csv')
#default partition for value_dict_rdd
print('Default partitions: ',value_dict_rdd.getNumPartitions())
```

```
Default partitions:  2
```

1.2.2. The metadata shows that days in the dataset are stored as a relative number. For example, if the application date is 2/Jan/2024, -1 means 1/Jan/2024, -2 means 31/Dec/2023.
Create a UDF function that takes two parameters: a date and an integer value, and returns a date. (note: the integer can be either positive or negative). (3%)
Assuming all applications are made on 1/Jan/2024, create a new column named decision_date, use the UDF function to fill its values from days_decisions (3%)

[38]:
```
#Import udf function for implementating udf function in rdd
from pyspark.sql.functions import udf
#Import stringtype since we need integer values
from pyspark.sql.types import StringType
#Import date,time since that the application that are made on 1 jan 2024 as the
↪base year
from datetime import datetime, timedelta

#Define add_days_to_date function as argument base_date and days
```

```
#base_date should be as per the application date given in the question we have␣
↪formated as %Y-%m-%d
def adding_days_to_date(based_date, days):
    based_date = datetime.strptime(base_date, "%Y-%m-%d")
    new_date = based_date + timedelta(days)
    return new_date.strftime("%Y-%m-%d")

#Register the UDF with Spark
#Using string type for integer values
adding_days_to_date_udf = udf(adding_days_to_date, StringType())

# Assuming all applications are made on 1/Jan/2024
based_date = "2024-01-01"

# Use the UDF to create a new column 'decision_date' for previous_application
map_exp_rdd_1_with_decision_date = previous_application_rdd.map(lambda x: x +␣
↪',' + adding_days_to_date_udf(based_date, int(x.split(',')[2])))
```

1.2.3. Join application_data and previous_application with value_dict and replace integer values with string values from the dictionary. (5%)

[21]:
```
# RDDs for the CSV files - application_data, previous_application and value_dict
application_data_rdd = sc.textFile('application_data.csv')
previous_application_rdd = sc.textFile('previous_application.csv')
value_dict_rdd = sc.textFile('value_dict.csv')
# Join application_data_pair_rdd and previous_application_pair_rdd using the␣
↪common key
joined_rdd = application_data_rdd.join(previous_application_rdd)
# Join the result with value_dict_pair_rdd using the common key
final_joined_rdd = joined_rdd.join(value_dict_rdd)
```

8.0.1 1.3 Query/Analysis

For this part, write relevant RDD operations to answer the following queries.

1.3.1 Calculate the total approved loan amount for each year, each month. Print the results in the format of year, month, total_amount. (5%)

[37]:
```
# Assuming the structure of the CSV file
header = previous_application_rdd.first()
rdd = previous_application_rdd.filter(lambda row: row != header).map(lambda␣
↪line: line.split(','))

# Define the function extract_loan_amount variable with the argument record
def extract_loan_amount(record):
    year_month = record[8][:8]  # using the indice we can take the column␣
↪'hour_appr_process_start'
```

```
    amount = float(record[5])  # Using the indice we can take the column␣
  ↪'amt_credit'
    return (year_month, amount)

# Define the variable year_month_amount_rdd using map function to get the key
year_month_amount_rdd = previous_application_rdd.map(extract_loan_amount)

# Calculate the total approved loan amount for each year and each month
total_amount_by_year_month = year_month_amount_rdd.reduceByKey(lambda x, y: x +␣
  ↪y)
```

1.3.2 For each hour when the applications start (0-23), compute and print the percentage ratio of application cancellation. (5%)

[ ]:

```

```

8.1 Part 2. Working with DataFrames

In this section, you will need to load the given datasets into PySpark DataFrames and use DataFrame functions to answer the queries. ### 2.1 Data Preparation and Loading

2.1.1. Load CSVs into separate dataframes. When you create your dataframes, please refer to the metadata file and use appropriate data type for each column.

[12]:

```
# Read files into dataframes - application_data, previous_application and␣
  ↪value_dict datasets
df_application_data = spark.read.csv('application_data.csv',header=True)
df_previous_application = spark.read.csv('previous_application.
  ↪csv',header=True).repartition(4)
df_value_dict = spark.read.csv("value_dict.csv",header=True).repartition(4)
```

[23]:

```
# Create Views from Dataframes - application_data, previous_application and␣
  ↪value_dict datasets
df_application_data.createOrReplaceTempView("sql_application")
df_previous_application.createOrReplaceTempView("sql_previous_application")
df_value_dict.createOrReplaceTempView("sql_value_dict")
```

2.1.2 Display the schema of all dataframes.

[24]:

```
# Create Schema for all dataframes - application_data, previous_application and␣
  ↪value_dict datasets
df_application_data.printSchema()
df_previous_application.printSchema()
df_value_dict.printSchema()
```

```
root
 |-- id_app: string (nullable = true)
 |-- target: string (nullable = true)
```

```
 |-- contract_type: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- own_car: string (nullable = true)
 |-- own_property: string (nullable = true)
 |-- num_of_children: string (nullable = true)
 |-- income_total: string (nullable = true)
 |-- amt_credit: string (nullable = true)
 |-- amt_annuity: string (nullable = true)
 |-- amt_goods_price: string (nullable = true)
 |-- income_type: string (nullable = true)
 |-- education_type: string (nullable = true)
 |-- family_status: string (nullable = true)
 |-- housing_type: string (nullable = true)
 |-- region_population: string (nullable = true)
 |-- days_birth: string (nullable = true)
 |-- days_employed: string (nullable = true)
 |-- own_car_age: string (nullable = true)
 |-- flag_mobile: string (nullable = true)
 |-- flag_emp_phone: string (nullable = true)
 |-- flag_work_phone: string (nullable = true)
 |-- flag_cont_mobile: string (nullable = true)
 |-- flag_phone: string (nullable = true)
 |-- flag_email: string (nullable = true)
 |-- occupation_type: string (nullable = true)
 |-- cnt_fam_members: string (nullable = true)
 |-- weekday_app_process_start: string (nullable = true)
 |-- hour_app_process_start: string (nullable = true)
 |-- organization_type: string (nullable = true)
 |-- credit_score_1: string (nullable = true)
 |-- credit_score_2: string (nullable = true)
 |-- credit_score_3: string (nullable = true)
 |-- days_last_phone_change: string (nullable = true)
 |-- amt_credit_req_last_hour: string (nullable = true)
 |-- amt_credit_req_last_day: string (nullable = true)
 |-- amt_credit_req_last_week: string (nullable = true)
 |-- amt_credit_req_last_month: string (nullable = true)
 |-- amt_credit_req_last_quarter: string (nullable = true)
 |-- amt_credit_req_last_year: string (nullable = true)

root
 |-- id_app: string (nullable = true)
 |-- contract_type: string (nullable = true)
 |-- amt_annuity: string (nullable = true)
 |-- amt_application: string (nullable = true)
 |-- amt_credit: string (nullable = true)
 |-- amt_down_payment: string (nullable = true)
 |-- amt_goods_price: string (nullable = true)
 |-- hour_appr_process_start: string (nullable = true)
```

```
 |-- rate_down_payment: string (nullable = true)
 |-- rate_interest_primary: string (nullable = true)
 |-- rate_interest_privileged: string (nullable = true)
 |-- name_cash_loan_purpose: string (nullable = true)
 |-- name_contract_status: string (nullable = true)
 |-- days_decision: string (nullable = true)
 |-- name_payment_type: string (nullable = true)
 |-- code_rejection_reason: string (nullable = true)
 |-- name_type_suite: string (nullable = true)
 |-- name_client_type: string (nullable = true)
 |-- name_goods_category: string (nullable = true)
 |-- name_portfolio: string (nullable = true)
 |-- name_product_type: string (nullable = true)
 |-- channel_type: string (nullable = true)
 |-- sellerplace_area: string (nullable = true)
 |-- name_seller_industry: string (nullable = true)
 |-- cnt_payment: string (nullable = true)
 |-- name_yield_group: string (nullable = true)
 |-- product_combination: string (nullable = true)
 |-- days_first_drawing: string (nullable = true)
 |-- days_first_due: string (nullable = true)
 |-- days_last_due_1st_version: string (nullable = true)
 |-- days_last_due: string (nullable = true)
 |-- days_termination: string (nullable = true)
 |-- nflag_insured_on_approval: string (nullable = true)
 |-- id: string (nullable = true)

root
 |-- id: string (nullable = true)
 |-- category: string (nullable = true)
 |-- key: string (nullable = true)
 |-- value: string (nullable = true)
```

8.1.1 2.2 QueryAnalysis

Implement the following queries using dataframes. You need to be able to perform operations like filtering, sorting, joining and group by using the functions provided by the DataFrame API.

2.2.1. alculate the average income for each education_type group, and print the result. (4%)

[26]:
```python
#import functions that can used for aggregations
from pyspark.sql import functions as F

# df_application_data is your DataFrame and finding the average income for each␣
↪eduction type using groupby
```

```
avg_income_df = df_application_data.groupby('education_type').agg(F.
 ↪avg('income_total').alias('AverageIncome'))

#Showing avg_income_df
avg_income_df.show()
```

```
+--------------+------------------+
|education_type|     AverageIncome|
+--------------+------------------+
|             3| 128151.7258064516|
|             5|          227110.0|
|             1|155155.73917756812|
|             4|209339.63911283418|
|             2| 181014.4486888112|
+--------------+------------------+
```

2.2.2. Find the applicants who made credit requests last year with an average credit score of less than 0.5 from the three credit rating sources. (note: impute null value in credit score with 0.5, not 0). (4%)

[11]:
```
#import functions for column, when, average,count and year
from pyspark.sql.functions import col, when, avg, count, year
#import function for datetime and timedelta
from datetime import datetime, timedelta

# Assuming df_application_data is your DataFrame

# Impute null values in credit score columns with 0.5
#using the fill na function subsetting the credit_scores_columns - null value␣
 ↪in credit score 0.5
create_credit_scores_columns = ["credit_score_1", "credit_score_2",␣
 ↪"credit_score_3"]
df_credit_score_average = df_application_data.fillna(0.5,␣
 ↪subset=create_credit_scores_columns)

# we need to find the average credit score of less than 0.5 from three credit␣
 ↪rating sources
average_credit_score = (
    df_credit_score_average
    .filter(year(col("amt_credit_req_last_year")) >= (datetime.now() -␣
 ↪timedelta(days=365)).year)
    .groupBy("id_app")
    .agg(
        count("id_app").alias("Total"),
        (avg(
```

```
            when(col("credit_score_1").isNull(), 0.5).
↪otherwise(col("credit_score_1")) +
            when(col("credit_score_2").isNull(), 0.5).
↪otherwise(col("credit_score_2")) +
            when(col("credit_score_3").isNull(), 0.5).
↪otherwise(col("credit_score_3"))
        ) / 3.0).alias("avg_credit_score")
    )
    .filter("avg_credit_score < 0.5")
)

# show the dataframe average_credit_score
average_credit_score.show()
```

```
+------+-----+----------------+
|id_app|Total|avg_credit_score|
+------+-----+----------------+
+------+-----+----------------+
```

2.2.3. Transform the 'days_birth' column in the application_data to age(integer rounded down) and date_of_birth; then show the schema. You are allowed to use the UDF defined in part 1. (4%)

[28]:
```
#Import udf, lit,col function in the following codes
from pyspark.sql.functions import udf, lit, col
#Import IntegerType function since we need to use for day_birth column to age␣
↪so we have to use integer type as per the question
from pyspark.sql.types import IntegerType


# Load the dataframe application_data_df
application_data_df = spark.read.csv('application_data.csv', header=True,␣
↪inferSchema=True)

# Define the UDF
@udf(IntegerType())
def extract_date_year(s):
    try:
        return int(s.split(' ')[0])
    except (AttributeError, ValueError):
        return None

# Add new columns to the DataFrame - adding_new_columns as in transforming the␣
↪days_birth column to age
#by deriving the day_birth to age
adding_new_columns = application_data_df \
```

```
    .withColumn("birth_year", extract_date_year("days_birth")) \
    .withColumn("age", lit(2024) - col("birth_year")) \
    .withColumn("date_of_birth", extract_date_year("days_birth"))

# Showing whether the new column adding_new_columns has been added in the␣
↪dataframe - i mean to check whether it is included in the dataframe
adding_new_columns.show(5)

# Showing the schema of the new column defined that is adding_new_columns
adding_new_columns.printSchema()
```

```
+------+------+-------------+------+-------+------------+--------------+-------
-----+----------+-----------+---------------+-----------+--------------+--------
-----+------------+----------------+----------+------------+-----------+------
-----+--------------+---------------+----------------+----------+----------+----
-----------+---------------+------------------------+-----------------------+---
--------------+--------------+--------------+--------------+--------------------
--+------------------------+------------------------+------------------------+---
----------------------+--------------------------+------------------------+----
------+----+-------------+
|id_app|target|contract_type|gender|own_car|own_property|num_of_children|income_
total|amt_credit|amt_annuity|amt_goods_price|income_type|education_type|family_s
tatus|housing_type|region_population|days_birth|days_employed|own_car_age|flag_m
obile|flag_emp_phone|flag_work_phone|flag_cont_mobile|flag_phone|flag_email|occu
pation_type|cnt_fam_members|weekday_app_process_start|hour_app_process_start|org
anization_type|credit_score_1|credit_score_2|credit_score_3|days_last_phone_chan
ge|amt_credit_req_last_hour|amt_credit_req_last_day|amt_credit_req_last_week|amt
_credit_req_last_month|amt_credit_req_last_quarter|amt_credit_req_last_year|birt
h_year| age|date_of_birth|
+------+------+-------------+------+-------+------------+--------------+-------
-----+----------+-----------+---------------+-----------+--------------+--------
-----+------------+----------------+----------+------------+-----------+------
-----+--------------+---------------+----------------+----------+----------+----
-----------+---------------+------------------------+-----------------------+---
--------------+--------------+--------------+--------------+--------------------
--+------------------------+------------------------+------------------------+---
----------------------+--------------------------+------------------------+----
------+----+-------------+
|118100|     0|            2|     F|      Y|           Y|              1|
247500.0|  667237.5|    52848.0|       576000.0|          2|             4|
3|           6|         0.018801|    -11258|        -1596|       13.0|
1|             1|              0|               1|         0|         0|
12|            3.0|                   FRIDAY|                      8|
28|    0.60994226|     0.5884348|          NULL|                -733.0|
NULL|                       NULL|                      NULL|                   NULL|
NULL|                        NULL|       NULL|NULL|         NULL|
|110133|     0|            2|     F|      N|           Y|              2|
```

```
112500.0| 1374480.0|    49500.0|      1125000.0|          8|             1|
3|          6|         0.006233|    -11044|         -942|       NULL|
1|            1|               1|               1|         0|         0|
16|           4.0|                   MONDAY|                      10|
42|     0.7081764|     0.6865754|          NULL|                    0.0|
NULL|                   NULL|                    NULL|                     NULL|
NULL|                    NULL|      NULL|NULL|         NULL|
|110215|     0|             2|     F|       N|            Y|              0|
166500.0|  545040.0|    26640.0|       450000.0|          2|             1|
6|          6|         0.032561|    -17115|         -581|       NULL|
1|            1|               0|               1|         1|         0|
19|           1.0|                   MONDAY|                      14|
22|    0.49497995|    0.58477587|    0.47225335|               -1598.0|
0.0|                    0.0|                     0.0|                      1.0|
0.0|                     3.0|      NULL|NULL|         NULL|
|194051|     0|             2|     F|       N|            N|              0|
112500.0|  900000.0|    24750.0|       900000.0|          2|             1|
2|          6|         0.015221|    -17855|        -5470|       NULL|
1|            1|               0|               1|         1|         0|
8|           2.0|                   FRIDAY|                      15|
30|          NULL|    0.59620756|     0.6195277|                -734.0|
0.0|                    0.0|                     0.0|                      0.0|
0.0|                     1.0|      NULL|NULL|         NULL|
|110368|     0|             2|     F|       N|            Y|              0|
261000.0| 1237684.5|    47272.5|      1138500.0|          5|             4|
3|          6|         0.020713|    -22818|       365243|       NULL|
1|            0|               0|               1|         0|         0|
18|           2.0|                   FRIDAY|                      10|
31|          NULL|    0.64156574|     0.3996756|                -979.0|
0.0|                    0.0|                     0.0|                      0.0|
0.0|                     0.0|      NULL|NULL|         NULL|
+------+------+-------------+------+-------+------------+--------------+-------
-----+----------+-----------+---------------+----------+-------------+--------
-----+------------+-----------------+----------+-------------+----------+------
-----+--------------+---------------+---------------+----------+----------+----
-----------+---------------+------------------------+---------------------+---
-------------+--------------+--------------+--------------+-------------------
--+-------------------------+----------------------+-----------------------+---
---------------------+-------------------------+-----------------------+----
------+----+-------------+
only showing top 5 rows

root
 |-- id_app: integer (nullable = true)
 |-- target: integer (nullable = true)
 |-- contract_type: integer (nullable = true)
 |-- gender: string (nullable = true)
 |-- own_car: string (nullable = true)
```

```
 |-- own_property: string (nullable = true)
 |-- num_of_children: integer (nullable = true)
 |-- income_total: double (nullable = true)
 |-- amt_credit: double (nullable = true)
 |-- amt_annuity: double (nullable = true)
 |-- amt_goods_price: double (nullable = true)
 |-- income_type: integer (nullable = true)
 |-- education_type: integer (nullable = true)
 |-- family_status: integer (nullable = true)
 |-- housing_type: integer (nullable = true)
 |-- region_population: double (nullable = true)
 |-- days_birth: integer (nullable = true)
 |-- days_employed: integer (nullable = true)
 |-- own_car_age: double (nullable = true)
 |-- flag_mobile: integer (nullable = true)
 |-- flag_emp_phone: integer (nullable = true)
 |-- flag_work_phone: integer (nullable = true)
 |-- flag_cont_mobile: integer (nullable = true)
 |-- flag_phone: integer (nullable = true)
 |-- flag_email: integer (nullable = true)
 |-- occupation_type: integer (nullable = true)
 |-- cnt_fam_members: double (nullable = true)
 |-- weekday_app_process_start: string (nullable = true)
 |-- hour_app_process_start: integer (nullable = true)
 |-- organization_type: integer (nullable = true)
 |-- credit_score_1: double (nullable = true)
 |-- credit_score_2: double (nullable = true)
 |-- credit_score_3: double (nullable = true)
 |-- days_last_phone_change: double (nullable = true)
 |-- amt_credit_req_last_hour: double (nullable = true)
 |-- amt_credit_req_last_day: double (nullable = true)
 |-- amt_credit_req_last_week: double (nullable = true)
 |-- amt_credit_req_last_month: double (nullable = true)
 |-- amt_credit_req_last_quarter: double (nullable = true)
 |-- amt_credit_req_last_year: double (nullable = true)
 |-- birth_year: integer (nullable = true)
 |-- age: integer (nullable = true)
 |-- date_of_birth: integer (nullable = true)
```

2.2.4. Using an age bucket of 10(0-10, 11-20, 21-30, etc..), compute the percentage of applicants owning a car and a property. (8%)

[24]:
```
#import function expression so that we use it in the expressional conditional
 ↪case
from pyspark.sql.functions import expr

# Load the dataframe application_data
```

```python
df_application_data = spark.read.csv("application_data.csv", header=True)

# Define age ranges as per given in the dataframe
age_ranges = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Group by 'age_ranges' and compute the percentage for owning both a car and a
↪property
result = applicants_ages.groupBy("age_ranges").agg(
    expr("SUM(CASE WHEN own_car = 'Yes' AND own_property = 'Yes' THEN 1 ELSE 0
↪END) / COUNT(*) * 100").alias("percentage")
)

# Show the result
result.show()
```

```
+----------+----------+
|age_ranges|percentage|
+----------+----------+
|     40-50|       0.0|
|     20-30|       0.0|
|      NULL|       0.0|
|     60-70|       0.0|
|     50-60|       0.0|
|      0-10|       0.0|
|     10-20|       0.0|
|     30-40|       0.0|
+----------+----------+
```

2.2.5. Draw a barchart to show the total number of uncancelled applications from male/female in each year. (10%)

[ ]:

2.2.6. Draw a scatter plot of the applicants' age and their total approved credit. You may use log scales for the XY axis if necessary. (10%)

[ ]:

8.1.2 Part 3 RDDs vs DataFrame vs Spark SQL (15%)

Implement the following queries using RDDs, DataFrames in SparkSQL separately. Log the time taken for each query in each approach using the "%%time" built-in magic command in Jupyter Notebook and discuss the performance difference between these three approaches.

Complex Query (high-risk applicants): Find the top 100 applicants who are married with children and have a total approved credit that is more than five times their

incomes (regardless of any payments made), sorted by the total credit/income ratio. (hint: intermediate dataframes/tables are allowed if necessary)

3.1. RDD Implementation

[31]:
```
# Load RDDs all three csv flies into RDD - Application_data,␣
↪Previous_application and Value_dict
application_data_rdd = sc.textFile('application_data.csv')
previous_application_rdd = sc.textFile('previous_application.csv')
value_dict_rdd = sc.textFile('value_dict.csv')

# Using the parallel search function in rdd first we filter the applicants by␣
↪gender who are married and having children
filtered_applicants = application_data_rdd.filter(lambda x: x.gender ==␣
↪'married' and x.num_of_children > 0)

#Defined the variable total_ratio_credit by using the previous variable␣
↪filtered_applicants
#Using id_app and using the formula amt_credit/income_total we get the filtered␣
↪applicants
Total_ratio_credit = filtered_applicants.map(lambda x: (x.id_app, x.amt_credit/␣
↪x.income_total))

# Define the function total_amt_credit_filtered to filter the applicants id
total_amt_credit_filtered = Total_ratio_credit.filter(lambda x: x[1] > 5)
```

3.2. DataFrame Implementation

[18]:
```
import time

# Read files into DataFrames
df_application_data = spark.read.csv("application_data.csv", header=True)

# Log the time before executing the complex query
start_time = time.time()

# Perform the complex query using DataFrame API
df_application_data_1 = (
    df_application_data
    .filter((df_application_data["family_status"] == "married") &␣
↪(df_application_data["num_of_children"] > 0))
    .filter(df_application_data["amt_credit"] > 5 *␣
↪df_application_data["income_total"])
    .withColumn("credit_income_ratio", df_application_data["amt_credit"] /␣
↪df_application_data["income_total"])
    .orderBy("credit_income_ratio", ascending=False)
    .limit(100)
```

```python
)

# Log the time taken for the query
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")

# Show the result
df_application_data_1.show()
```

```
Time taken: 0.01622486114501953 seconds
+------+------+-------------+------+-------+------------+---------------+-------
-----+----------+-----------+---------------+-----------+--------------+--------
-----+------------+-----------------+----------+-------------+-----------+------
-----+--------------+---------------+----------------+----------+----------+----
-----------+---------------+-------------------------+----------------------+---
--------------+--------------+--------------+--------------+--------------------
--+------------------------+-----------------------+------------------------+---
----------------------+---------------------------+------------------------+----
---------------+
|id_app|target|contract_type|gender|own_car|own_property|num_of_children|income_
total|amt_credit|amt_annuity|amt_goods_price|income_type|education_type|family_s
tatus|housing_type|region_population|days_birth|days_employed|own_car_age|flag_m
obile|flag_emp_phone|flag_work_phone|flag_cont_mobile|flag_phone|flag_email|occu
pation_type|cnt_fam_members|weekday_app_process_start|hour_app_process_start|org
anization_type|credit_score_1|credit_score_2|credit_score_3|days_last_phone_chan
ge|amt_credit_req_last_hour|amt_credit_req_last_day|amt_credit_req_last_week|amt
_credit_req_last_month|amt_credit_req_last_quarter|amt_credit_req_last_year|cred
it_income_ratio|
+------+------+-------------+------+-------+------------+---------------+-------
-----+----------+-----------+---------------+-----------+--------------+--------
-----+------------+-----------------+----------+-------------+-----------+------
-----+--------------+---------------+----------------+----------+----------+----
-----------+---------------+-------------------------+----------------------+---
--------------+--------------+--------------+--------------+--------------------
--+------------------------+-----------------------+------------------------+---
----------------------+---------------------------+------------------------+----
---------------+
+------+------+-------------+------+-------+------------+---------------+-------
-----+----------+-----------+---------------+-----------+--------------+--------
-----+------------+-----------------+----------+-------------+-----------+------
-----+--------------+---------------+----------------+----------+----------+----
-----------+---------------+-------------------------+----------------------+---
--------------+--------------+--------------+--------------+--------------------
--+------------------------+-----------------------+------------------------+---
----------------------+---------------------------+------------------------+----
---------------+
```

3.3. Spark SQL Implementation

[23]:
```python
# Read files into DataFrames
df_application_data = spark.read.csv("application_data.csv", header=True)

# Start timing
start_time = time.time()

# Run the SQL query
df_application_date_sql = spark.sql("""
SELECT *,
amt_credit / income_total AS credit_income_ratio
FROM application_data
WHERE family_status = 'married' AND num_of_children > 0 AND amt_credit > 5
ORDER BY credit_income_ratio DESC
LIMIT 100
""")

# Log the time taken for the query
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")

# Show the result
df_application_date_sql.show()
```

```
Time taken: 0.012541532516479492 seconds
+------+------+-------------+------+-------+------------+---------------+-------
-----+----------+-----------+---------------+-----------+--------------+--------
-----+------------+-----------------+----------+-------------+-----------+------
-----+--------------+---------------+----------------+----------+----------+----
-----------+---------------+-------------------------+----------------------+---
--------------+--------------+--------------+--------------+--------------------
--+------------------------+-----------------------+------------------------+---
----------------------+---------------------------+------------------------+----
---------------+
|id_app|target|contract_type|gender|own_car|own_property|num_of_children|income_
total|amt_credit|amt_annuity|amt_goods_price|income_type|education_type|family_s
tatus|housing_type|region_population|days_birth|days_employed|own_car_age|flag_m
obile|flag_emp_phone|flag_work_phone|flag_cont_mobile|flag_phone|flag_email|occu
pation_type|cnt_fam_members|weekday_app_process_start|hour_app_process_start|org
anization_type|credit_score_1|credit_score_2|credit_score_3|days_last_phone_chan
ge|amt_credit_req_last_hour|amt_credit_req_last_day|amt_credit_req_last_week|amt
_credit_req_last_month|amt_credit_req_last_quarter|amt_credit_req_last_year|cred
it_income_ratio|
+------+------+-------------+------+-------+------------+---------------+-------
```

```
-----+----------+-----------+---------------+-----------+--------------+--------
-----+------------+-----------------+----------+-------------+-----------+------
-----+--------------+---------------+----------------+----------+----------+----
-----------+---------------+-------------------------+----------------------+---
--------------+--------------+--------------+--------------+--------------------
--+------------------------+-----------------------+------------------------+---
----------------------+---------------------------+------------------------+----
---------------+
+------+------+-------------+------+-------+------------+---------------+-------
-----+----------+-----------+---------------+-----------+--------------+--------
-----+------------+-----------------+----------+-------------+-----------+------
-----+--------------+---------------+----------------+----------+----------+----
-----------+---------------+-------------------------+----------------------+---
--------------+--------------+--------------+--------------+--------------------
--+------------------------+-----------------------+------------------------+---
----------------------+---------------------------+------------------------+----
---------------+
```

8.1.3 3.4 Observe the query execution time among RDD, DataFrame, SparkSQL, which is the fastest and why? (Maximum 500 words.)

Summary The time might differ when running on different machines. Overall, the DataFrame and SQL queries run 0.01622486114501953 seconds and 0.012541532516479492 seconds compared to RDD which says that time taken by SQL Queries are faster than the other two method which shows in fraction of time. According to Stackflow, it says that SQL Queries are faster due to exchange reuse. When we are running the sql queries in spark the systen will shuffle using aggregations and it can store copy potential nodes.

9 Reference

https://stackoverflow.com/questions/57261153/ways-to-plot-spark-dataframe-without-converting-it-to-pandas - scatter plot and bar graph for dataframes in question 5 and 6 in part 2 working with dataframe

https://spark.apache.org/docs/latest/sql-ref-functions-udf-scalar.html - UDF in dataframe for the question 3 in part 2 working with dataframe

http://127.0.0.1:5202/notebooks/Lab/FIT5202%20-%20Parallel%20Aggregation.ipynb - UDF in dataframe lab activities - parallel aggregation (UDF) and question 1 in part 2 working with dataframe - average, min,max

http://127.0.0.1:5202/notebooks/Lab/FIT5202%20-%20Parallel%20Search%20.ipynb#two - Data Partitioning in RDD for question 1.2.1 part a

https://stackoverflow.com/questions/60340360/how-to-read-a-column-from-pyspark-rdd-and-apply-udf-on-it - rdd in udf we just need to define the function

https://stackoverflow.com/questions/72523103/pyspark-analyse-execution-time-of-queries - time log query for rdd in part 3 for rdd vs dataframe vs spark sql http://127.0.0.1:5202/notebooks/Lab/FIT5202%20-%20Parallel%20Joins.ipynb - Taken parallel join for bar graph for dataframe

http://127.0.0.1:5202/notebooks/Lab/FIT5202%20-%20Python%20Refresher%20(1)%20(3).ipynb - Python Refresher

http://127.0.0.1:5202/notebooks/Lab/FIT5202%20-%20Getting%20started%20with%20Apache%20Spark%20(1)%
- RDD Operations

https://stackoverflow.com/questions/55548530/why-is-execution-time-of-spark-sql-query-different-between-first-time-and-second - Time execution summary writing

https://spark.apache.org/docs/latest/rdd-programming-guide.html - Apache Spark documentation