Spark安装配置

1. **下载解压spark压缩包**

```
# user @ ubuntu in /opt/software [9:44:14] C:3
$ sudo wget https://dlcdn.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
--2023-12-24 09:44:26--  https://dlcdn.apache.org/spark/spark-3.5.0/spark-3.5.0-bin-hadoop3.tgz
正在解析主机 dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
正在连接 dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... 已连接。
已发出 HTTP 请求，正在等待回应... 200 OK
长度： 400395283 (382M) [application/x-gzip]
正在保存至: “spark-3.5.0-bin-hadoop3.tgz”

spark-3.5.0-bin-hadoop3.tgz         54%[================================>                         ] 208.31M  11.9MB/s    剩余 16s
spark-3.5.0-bin-hadoop3.tgz        100%[==========================================================>] 381.85M  9.16MB/s    用时 37s

2023-12-24 09:45:03 (10.4 MB/s) - 已保存 “spark-3.5.0-bin-hadoop3.tgz” [400395283/400395283])
```

2. **完成相关节点的配置**

```
# - SPARK_DAEMON_CLASSPATH, to set the classpath for all daemons
# - SPARK_PUBLIC_DNS, to set the public dns name of the master or workers

# Options for launcher
# - SPARK_LAUNCHER_OPTS, to set config properties and Java options for the launcher (e.g. "-Dx=y")

# Generic options for the daemons used in the standalone deploy mode
# - SPARK_CONF_DIR      Alternate conf dir. (Default: ${SPARK_HOME}/conf)
# - SPARK_LOG_DIR       Where log files are stored.  (Default: ${SPARK_HOME}/logs)
# - SPARK_LOG_MAX_FILES Max log files of Spark daemons can rotate to. Default is 5.
# - SPARK_PID_DIR       Where the pid file is stored. (Default: /tmp)
# - SPARK_IDENT_STRING  A string representing this instance of spark. (Default: $USER)
# - SPARK_NICENESS      The scheduling priority for daemons. (Default: 0)
# - SPARK_NO_DAEMONIZE  Run the proposed command in the foreground. It will not output a PID file.
# Options for native BLAS, like Intel MKL, OpenBLAS, and so on.
# You might get better performance to enable these options if using native BLAS (see SPARK-21305).
# - MKL_NUM_THREADS=1        Disable multi-threading of Intel MKL
# - OPENBLAS_NUM_THREADS=1   Disable multi-threading of OpenBLAS

# Options for beeline
# - SPARK_BEELINE_OPTS, to set config properties only for the beeline cli (e.g. "-Dx=y")
# - SPARK_BEELINE_MEMORY, Memory for beeline (e.g. 1000M, 2G) (Default: 1G)
spark-local-ip=127.0.1.1
```

3. **安装pyspark**

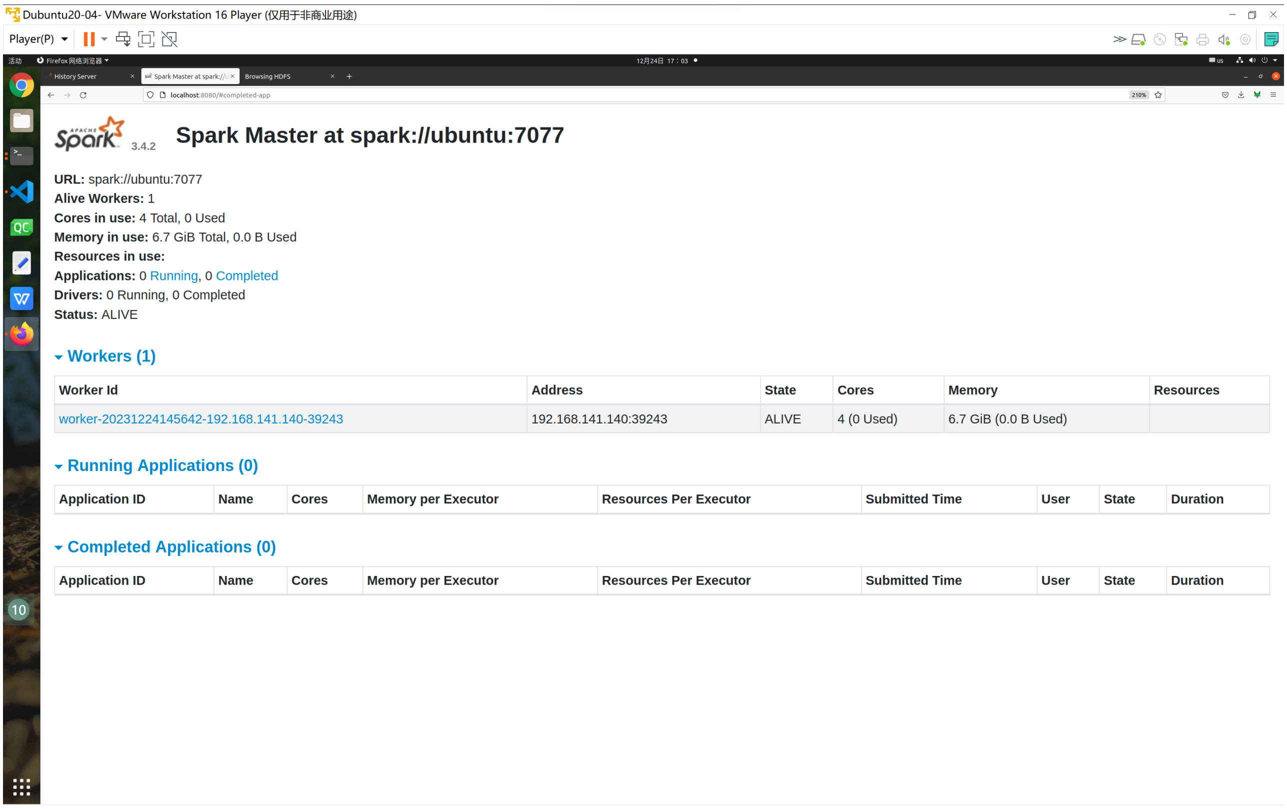
```
# user @ ubuntu in /opt/anaconda3 [10:34:36]
$ sudo pip install pyspark==2.4.8 -i https://pypi.tuna.tsinghua.edu.cn/simple
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Collecting pyspark==2.4.8
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/13/29/1a5d7c60609dfc8fd79c86965e95ea7e9e181385d839bc43913d277effa2/pyspark-2.4.8.tar.gz (220.5 MB)
     |████████████████████████████████| 220.5 MB 64 kB/s
Collecting py4j==0.10.7
  Downloading https://pypi.tuna.tsinghua.edu.cn/packages/e3/53/c737818eb9a7dc32a7cd4f1396e787bd94200c3997c72c1dbe028587bd76/py4j-0.10.7-py2.py3-none-any.whl (197 kB)
     |████████████████████████████████| 197 kB 8.0 MB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-2.4.8-py2.py3-none-any.whl size=220864377 sha256=8267d06bbdb720ed4613e999ae2d773e66f8115e77bdb443d3e549cdf38f002a
  Stored in directory: /root/.cache/pip/wheels/88/cc/f3/cbbe5f00ba594a086811118408d254235fbc551c9af61710b8
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.7 pyspark-2.4.8
(base)
```

4. **启动spark，并运行相关命令检验输出**

```
# user @ ubuntu in /opt/software/spark-3.5/conf [13:04:16]
$ spark-shell
23/12/24 13:05:05 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where a
pplicable
Spark context Web UI available at http://ubuntu:4040
Spark context available as 'sc' (master = local[*], app id = local-1703394306716).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 1.8.0_382)
Type in expressions to have them evaluated.
Type :help for more information.

scala> var r = sc.parallelize(Array(1,2,3,4))
r: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[0] at parallelize at <console>:23

scala> r.map(_*10).collect()
res0: Array[Int] = Array(10, 20, 30, 40)
```

5. **启动相应节点，查看Web UI**

```
# user @ ubuntu in /opt/software/spark-3.5 [13:28:49]
$ jps
6467 Master
6580 Worker
3926 ResourceManager
6694 Jps
3335 NameNode
3673 SecondaryNameNode
4060 NodeManager
3485 DataNode

# user @ ubuntu in /opt/software/spark-3.5 [13:29:31]
$ sbin/start-master.sh
org.apache.spark.deploy.master.Master running as process 6467.  Stop it first.
```



Spark编程

- **任务1-1**
  - 任务需求

> 编写 Spark 程序，统计application_data.csv中所有用户的贷款金额AMT_CREDIT 的分布情况。以10000 元为区间进行输出。
>
> 输出格式示例：((20000,30000),1234)表示20000到30000元之间（包括20000元，但不包括30000元）有1234条记录。

> **思路**：通过将贷款金额按照10000元为一段进行划分，然后统计每个区间内的记录数，最后以指定格式输出。关键步骤包括数据类型转换、区间划分、分组统计、排序和输出格式转换。

- 关键代码

```
  # 将贷款金额 AMT_CREDIT 转换为整数类型
  df = df.withColumn("AMT_CREDIT", df["AMT_CREDIT"].cast("int"))

  # 计算贷款金额的分布情况
  credit_distribution = df.select(expr("floor(AMT_CREDIT / 10000) * 10000 as credit_range")) \
      .groupBy("credit_range").count() \
      .orderBy("credit_range")

  # 将结果以指定格式输出
  output = credit_distribution.rdd.map(lambda row: (("({0},{1})".format(row["credit_range"], row["credit_range"] + 10000), row["count"]))).collect()

  # 输出结果
  for interval, count in output:
      print(interval, count)
```

- 结果输出

```
(40000,50000) 561
(50000,60000) 891
(60000,70000) 719
(70000,80000) 1226
(80000,90000) 668
(90000,100000) 1939
(100000,110000) 1871
(110000,120000) 1930
(120000,130000) 1323
(130000,140000) 4792
(140000,150000) 2239
(150000,160000) 3653
(160000,170000) 1919
(170000,180000) 2131
(180000,190000) 8745
(190000,200000) 1537
(200000,210000) 4017
(210000,220000) 1475
(220000,230000) 10013
(230000,240000) 3343
(240000,250000) 4206
(250000,260000) 6796
(260000,270000) 5186
(270000,280000) 10328
(280000,290000) 5728
(290000,300000) 3721
(300000,310000) 1766
(310000,320000) 6009
(320000,330000) 2248
(330000,340000) 3720
(340000,350000) 2462
(350000,360000) 1719
(360000,370000) 3200
(370000,380000) 1224
(380000,390000) 2579
```

```
(390000,400000) 1411
(400000,410000) 2789
(410000,420000) 1579
(420000,430000) 1050
(430000,440000) 1641
(440000,450000) 1471
(450000,460000) 13199
(460000,470000) 1196
(470000,480000) 2297
(480000,490000) 1655
(490000,500000) 5161
(500000,510000) 3701
(510000,520000) 2222
(520000,530000) 5496
(530000,540000) 2672
(540000,550000) 8587
(550000,560000) 2019
(560000,570000) 2139
(570000,580000) 1832
(580000,590000) 2375
(590000,600000) 3189
(600000,610000) 1581
(610000,620000) 1120
(620000,630000) 1222
(630000,640000) 1810
(640000,650000) 3265
(650000,660000) 1243
(660000,670000) 839
(670000,680000) 11388
(680000,690000) 686
(690000,700000) 895
(700000,710000) 1308
(710000,720000) 916
(720000,730000) 2678
(730000,740000) 857
(740000,750000) 1054
```

```
(750000,760000) 4424
(760000,770000) 1978
(770000,780000) 1123
(780000,790000) 3633
(790000,800000) 1222
(800000,810000) 4847
(810000,820000) 2276
(820000,830000) 799
(830000,840000) 2468
(840000,850000) 1169
(850000,860000) 1109
(860000,870000) 778
(870000,880000) 640
(880000,890000) 881
(890000,900000) 579
(900000,910000) 7230
(910000,920000) 715
(920000,930000) 627
(930000,940000) 518
(940000,950000) 2423
(950000,960000) 505
(960000,970000) 523
(970000,980000) 1157
(980000,990000) 697
(990000,1000000) 778
(1000000,1010000) 2587
(1010000,1020000) 656
(1020000,1030000) 1438
(1030000,1040000) 868
(1040000,1050000) 1463
(1050000,1060000) 905
(1060000,1070000) 725
(1070000,1080000) 3252
(1080000,1090000) 760
(1090000,1100000) 768
(1100000,1110000) 421
```

```
(1110000,1120000) 1009
(1120000,1130000) 4199
(1130000,1140000) 492
(1140000,1150000) 301
(1150000,1160000) 379
(1160000,1170000) 365
(1170000,1180000) 474
(1180000,1190000) 586
(1190000,1200000) 478
(1200000,1210000) 431
(1210000,1220000) 427
(1220000,1230000) 1224
(1230000,1240000) 599
(1240000,1250000) 268
(1250000,1260000) 1642
(1260000,1270000) 532
(1270000,1280000) 263
(1280000,1290000) 2865
(1290000,1300000) 516
(1300000,1310000) 789
(1310000,1320000) 688
(1320000,1330000) 532
(1330000,1340000) 385
(1340000,1350000) 343
(1350000,1360000) 2690
(1360000,1370000) 99
(1370000,1380000) 197
(1380000,1390000) 172
(1390000,1400000) 172
(1400000,1410000) 111
(1410000,1420000) 148
(1420000,1430000) 137
(1430000,1440000) 419
(1440000,1450000) 192
(1450000,1460000) 188
(1460000,1470000) 503
(1470000,1480000) 106
```

```
(1470000,1480000) 106
(1480000,1490000) 233
(1490000,1500000) 232
(1500000,1510000) 391
(1510000,1520000) 305
(1520000,1530000) 350
(1530000,1540000) 237
(1540000,1550000) 1430
(1550000,1560000) 96
(1560000,1570000) 148
(1570000,1580000) 974
(1580000,1590000) 72
(1590000,1600000) 55
(1600000,1610000) 51
(1610000,1620000) 87
(1620000,1630000) 107
(1630000,1640000) 106
(1640000,1650000) 143
(1650000,1660000) 27
(1660000,1670000) 119
(1670000,1680000) 61
(1680000,1690000) 160
(1690000,1700000) 47
(1700000,1710000) 35
(1710000,1720000) 126
(1720000,1730000) 408
(1730000,1740000) 60
(1740000,1750000) 47
(1750000,1760000) 103
(1760000,1770000) 672
(1770000,1780000) 78
(1780000,1790000) 44
(1790000,1800000) 38
(1800000,1810000) 931
(1810000,1820000) 33
(1820000,1830000) 68
(1830000,1840000) 23
```

```
(1840000,1850000) 35
(1850000,1860000) 20
(1860000,1870000) 29
(1870000,1880000) 36
(1880000,1890000) 165
(1890000,1900000) 25
(1900000,1910000) 20
(1910000,1920000) 27
(1920000,1930000) 143
(1930000,1940000) 27
(1940000,1950000) 31
(1950000,1960000) 19
(1960000,1970000) 47
(1970000,1980000) 394
(1980000,1990000) 60
(1990000,2000000) 17
(2000000,2010000) 6
(2010000,2020000) 481
(2020000,2030000) 72
(2030000,2040000) 21
(2040000,2050000) 11
(2050000,2060000) 5
(2060000,2070000) 13
(2070000,2080000) 12
(2080000,2090000) 133
(2090000,2100000) 7
(2100000,2110000) 11
(2110000,2120000) 18
(2120000,2130000) 7
(2130000,2140000) 5
(2140000,2150000) 8
(2150000,2160000) 100
(2160000,2170000) 23
(2170000,2180000) 12
(2180000,2190000) 7
```

```
(2190000,2200000) 8
(2200000,2210000) 20
(2210000,2220000) 18
(2220000,2230000) 45
(2230000,2240000) 3
(2240000,2250000) 8
(2250000,2260000) 375
(2260000,2270000) 21
(2270000,2280000) 2
(2280000,2290000) 5
(2290000,2300000) 14
(2300000,2310000) 12
(2310000,2320000) 12
(2320000,2330000) 3
(2330000,2340000) 6
(2340000,2350000) 5
(2350000,2360000) 12
(2360000,2370000) 9
(2370000,2380000) 15
(2380000,2390000) 4
(2390000,2400000) 3
(2400000,2410000) 3
(2410000,2420000) 30
(2420000,2430000) 7
(2440000,2450000) 22
(2450000,2460000) 5
(2460000,2470000) 41
(2470000,2480000) 7
(2480000,2490000) 1
(2510000,2520000) 227
(2520000,2530000) 1
(2540000,2550000) 1
(2570000,2580000) 2
(2580000,2590000) 2
(2600000,2610000) 9
(2610000,2620000) 1
```

```
(2610000,2620000) 1
(2680000,2690000) 3
(2690000,2700000) 62
(2700000,2710000) 9
(2730000,2740000) 1
(2920000,2930000) 3
(2930000,2940000) 6
(2960000,2970000) 1
(2980000,2990000) 1
(3020000,3030000) 1
(3060000,3070000) 1
(3070000,3080000) 1
(3150000,3160000) 9
(3290000,3300000) 1
(3310000,3320000) 1
(3370000,3380000) 4
(3600000,3610000) 2
(3860000,3870000) 1
(3950000,3960000) 1
(4020000,4030000) 1
(4030000,4040000) 1
(4050000,4060000) 8
```

- **任务1-2**
  - 任务需求

> 编写Spark程序，统计application_data.csv中客户贷款AMT_CREDIT 比客户收入AMT_INCOME_TOTAL差值最高和最低的各十条记录。 输出格式： <SK_ID_CURR><NAME_CONTRACT_TYPE><AMT_CREDIT><AMT_INCOME_TOTAL>, <差值> 差值 `=AMT_CREDIT-AMT_INCOME_TOTAL`

> **思路**：将贷款金额和客户收入转换为double类型，然后计算二者的差值。接着，按照差值升序和降序排列数据，并分别输出最高差值和最低差值的前十条记录。

  - 关键代码

```
  # 将贷款金额 AMT_CREDIT 和客户收入 AMT_INCOME_TOTAL 转换为 double 类型
  df = df.withColumn("AMT_CREDIT", col("AMT_CREDIT").cast("double"))
  df = df.withColumn("AMT_INCOME_TOTAL",
col("AMT_INCOME_TOTAL").cast("double"))

  # 计算差值，并按照差值升序和降序排列
  diff_column = df.withColumn("difference", col("AMT_CREDIT") -
col("AMT_INCOME_TOTAL"))
  sorted_diff = diff_column.orderBy("difference")

  # 输出最高差值的前十条记录
  print("Top 10 records with the highest difference:")
  sorted_diff.select("SK_ID_CURR", "NAME_CONTRACT_TYPE", "AMT_CREDIT",
"AMT_INCOME_TOTAL", "difference") \
      .orderBy("difference", ascending=False) \
```

```
        .limit(10) \
        .show()

    # 输出最低差值的前十条记录
    print("Top 10 records with the lowest difference:")
    sorted_diff.select("SK_ID_CURR", "NAME_CONTRACT_TYPE", "AMT_CREDIT",
"AMT_INCOME_TOTAL", "difference") \
        .limit(10) \
        .show()
```

- 结果输出

```
$ spark-submit task1-2.py
23/12/25 14:29:19 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where a
pplicable
Top 10 records with the highest difference:
+----------+------------------+----------+----------------+----------+
|SK_ID_CURR|NAME_CONTRACT_TYPE|AMT_CREDIT|AMT_INCOME_TOTAL|difference|
+----------+------------------+----------+----------------+----------+
|    433294|        Cash loans| 4050000.0|        405000.0| 3645000.0|
|    210956|        Cash loans| 4031032.5|        430650.0| 3600382.5|
|    434170|        Cash loans| 4050000.0|        450000.0| 3600000.0|
|    315893|        Cash loans| 4027680.0|        458550.0| 3569130.0|
|    238431|        Cash loans| 3860019.0|        292050.0| 3567969.0|
|    240007|        Cash loans| 4050000.0|        587250.0| 3462750.0|
|    117337|        Cash loans| 4050000.0|        760846.5| 3289153.5|
|    120926|        Cash loans| 4050000.0|        783000.0| 3267000.0|
|    117085|        Cash loans| 3956274.0|        749331.0| 3206943.0|
|    228135|        Cash loans| 4050000.0|        864900.0| 3185100.0|
+----------+------------------+----------+----------------+----------+

Top 10 records with the lowest difference:
+----------+------------------+----------+----------------+-------------+
|SK_ID_CURR|NAME_CONTRACT_TYPE|AMT_CREDIT|AMT_INCOME_TOTAL|   difference|
+----------+------------------+----------+----------------+-------------+
|    114967|        Cash loans|  562491.0|          1.17E8|-1.16437509E8|
|    336147|        Cash loans|  675000.0|     1.800009E7|  -1.732509E7|
|    385674|        Cash loans| 1400503.5|          1.35E7|-1.20994965E7|
|    190160|        Cash loans| 1431531.0|       9000000.0|   -7568469.0|
|    252084|        Cash loans|  790830.0|       6750000.0|   -5959170.0|
|    337151|        Cash loans|  450000.0|       4500000.0|   -4050000.0|
|    317748|        Cash loans|  835380.0|       4500000.0|   -3664620.0|
|    310601|        Cash loans|  675000.0|       3950059.5|   -3275059.5|
|    432980|        Cash loans| 1755000.0|       4500000.0|   -2745000.0|
|    157471|        Cash loans|  953460.0|       3600000.0|   -2646540.0|
+----------+------------------+----------+----------------+-------------+
```

- **任务2-1**

  - 任务需求

> 统计所有男性客户（CODE_GENDER=M）的小孩个数（CNT_CHILDREN）类型占比情况。
>
> 输出格式为：<CNT_CHILDREN>，<类型占比>

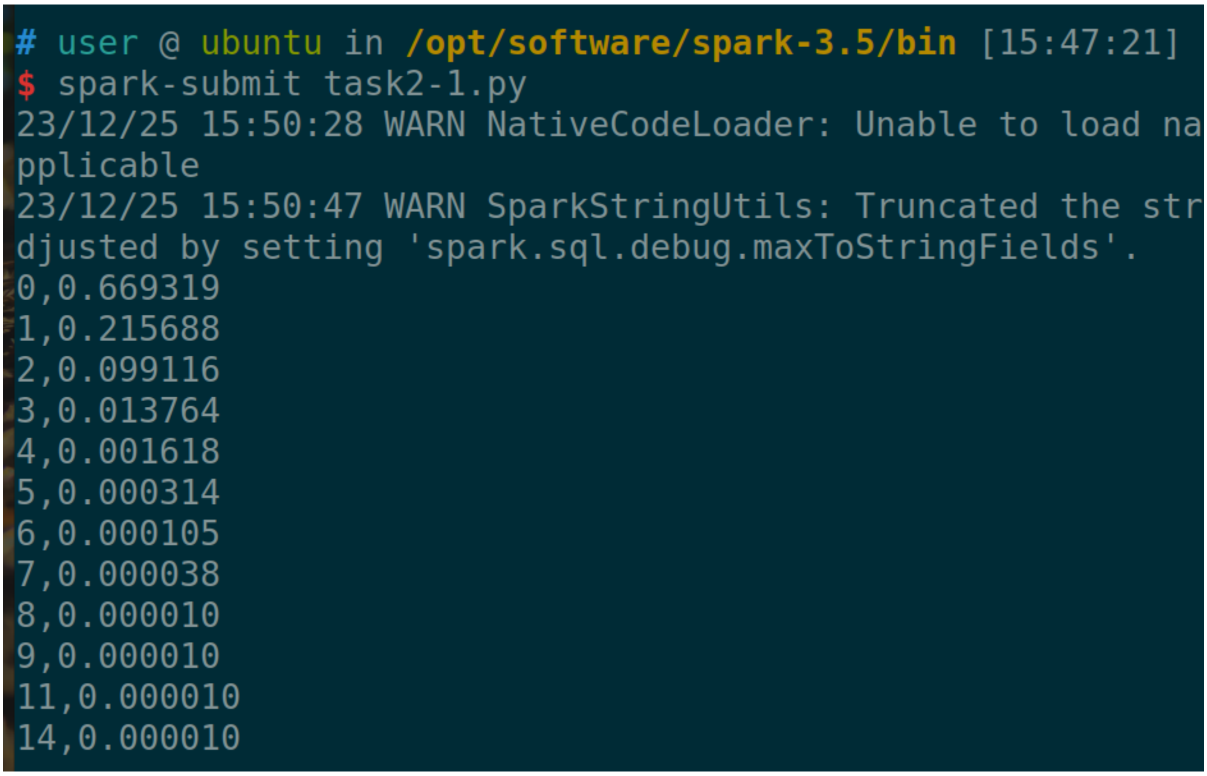
> **思路**：将数据注册为Spark SQL临时表，然后筛选出所有男性客户的小孩个数（CNT_CHILDREN）。接下来，对小孩个数进行分组统计，计算每种小孩个数类型在男性客户中的占比，并按小孩个数升序排列。

  - 关键代码

```
    # 将数据注册为 Spark SQL 临时表
    df.createOrReplaceTempView("application_data")

    # 将小孩个数和男性客户筛选出来
    filtered_data = spark.sql("SELECT CNT_CHILDREN FROM application_data WHERE
CODE_GENDER = 'M'")

    # 统计小孩个数类型占比
```

```
    result = filtered_data.groupBy("CNT_CHILDREN") \
        .agg(
            (count("*") / filtered_data.count()).alias("ratio")
        ) \
        .orderBy("CNT_CHILDREN")

    # 将结果以指定格式输出
    output = result.rdd.map(lambda row: "{0},
{1:.6f}".format(row["CNT_CHILDREN"], row["ratio"])).collect()

    # 输出结果
    for item in output:
        print(item)
```

- 结果输出（为了方便查看，代码中设置了只保留6位小数）



- **任务2-2**
  - 任务需求

> 统计每个客户出生以来每天的平均收入（avg_income）=总收入（AMT_INCOME_TOTAL）/出生天数（DAYS_BIRTH），统计每日收入大于1的客户，并按照从大到小排序，保存为csv。
>
> 输出格式：<SK_ID_CURR>, <avg_income>

> **思路**：将DataFrame注册为一个Spark SQL的临时表，然后执行Spark SQL查询，计算每个客户出生以来每天的平均收入（avg_income）。接下来，添加筛选条件保留每日收入大于1的客户，并重新计算avg_income。然后，按照avg_income从大到小进行排序。最后，选择需要的列（SK_ID_CURR和avg_income），将结果保存为CSV文件，

  - 关键代码

```
# 注册DataFrame为一个临时表
df.createOrReplaceTempView("application_data")

# 执行Spark SQL查询
result_df = spark.sql("""
    SELECT SK_ID_CURR,
        AMT_INCOME_TOTAL / ABS(DAYS_BIRTH) AS avg_income
    FROM application_data
""")

# 添加筛选条件并重新计算avg_income
result_df = result_df.filter(col("avg_income") > 1)

# 重新排序结果
result_df = result_df.orderBy(col("avg_income").desc())

# 选择需要的列
result_df = result_df.select("SK_ID_CURR", "avg_income")

# 将结果保存为CSV文件
result_df.write.csv("/user/user/avg_income.csv", header=True, mode="overwrite")
```

- 结果输出（csv文件保存在`/任务二/avg_income/avg_income.csv`）

```
# user @ ubuntu in /opt/software/spark-3.5/bin [16:20:57]
$ spark-submit task2-2.py
23/12/25 16:29:22 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
```

```
# user @ ubuntu in /hadoop_installs/hadoop-3.3.6 [16:30:16]
$ bin/hdfs dfs -get /user/user/avg_income.csv avg_income
```

	A	B
1	SK_ID_CURR	avg_income
2	114967	9274.673
3	336147	1146.211
4	385674	996.2364
5	190160	547.9452
6	219563	417.5172
7	310601	373.6341
8	157471	360.4325
9	252084	348.9995
10	199821	269.6548
11	337151	243.7571
12	141198	243.6237
13	429258	241.6594
14	196091	240.7619
15	317748	240.4488
16	432980	239.5656
17	217276	235.3205
18	445335	234.3084
19	387126	230.4653
20	304300	223.584
21	123587	207.2497
22	399467	195.9219
23	441639	192.4557
24	440768	192.041
25	225210	188.7539
26	206341	186.0017
27	134526	183.6885
28	214063	180.0827
29	336135	177.4648
30	111903	174.1351
31	269498	172.6027
32	194130	172.0052
33	431111	166.7593
34	251262	159.3202
35	422344	157.1449
36	268905	156.8272

avg_income

	A	B	C	D	E	F	G
307478	418117	1.265111					
307479	352415	1.264045					
307480	238677	1.263133					
307481	338222	1.260387					
307482	149155	1.257686					
307483	312410	1.257578					
307484	105945	1.256281					
307485	384810	1.248565					
307486	207041	1.245635					
307487	378118	1.24268					
307488	362281	1.236377					
307489	199035	1.235641					
307490	409014	1.235246					
307491	409321	1.227608					
307492	379018	1.227273					
307493	156273	1.223507					
307494	414387	1.223389					
307495	195352	1.222439					
307496	316377	1.215706					
307497	102779	1.21123					
307498	297401	1.21009					
307499	256532	1.210057					
307500	191341	1.205357					
307501	288702	1.198828					
307502	155473	1.194036					
307503	124157	1.192081					
307504	404855	1.186918					
307505	193872	1.184309					
307506	251601	1.174832					
307507	402866	1.167517					
307508	309932	1.163125					
307509	248175	1.144206					
307510	307620	1.142915					
307511	386069	1.105523					
307512	172587	1.09188					

- **任务3**
  - 任务需求

> 根据给定的数据集，基于Spark MLlib 或者Spark ML编写程序对贷款是否违约进行分类，并评估实验结果的准确率.

> **思路**：借鉴课堂上讲的鸢尾花示例，借用决策树算法。在机器学习中，决策树是一种流行的分类和回归算法。它的工作原理类似于树状结构，通过在数据集中选择最佳特征来进行分割，从而递归地构建一棵树。 结合金融相关知识，这次我选取的特征属性为`"CNT_CHILDREN", "FLAG_CONT_MOBILE", "AMT_INCOME_TOTAL", "AMT_CREDIT", "FLAG_MOBIL"`;这分别关乎客户的家庭状态、偿还能力、负债情况等相关，能较好的反应用户的经济能力。

  - 关键代码

```
  # 特征工程
  feature_cols = ["CNT_CHILDREN", "FLAG_CONT_MOBILE", "AMT_INCOME_TOTAL",
"AMT_CREDIT", "FLAG_MOBIL"]

  assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
  assembled_df = assembler.transform(df)

  # 将目标变量 "TARGET" 转换为数值型 "label"
  string_indexer = StringIndexer(inputCol="TARGET", outputCol="label")
  string_index_model = string_indexer.fit(assembled_df)
  indexed_df = string_index_model.transform(assembled_df)

  # 将数据拆分为训练集和测试集
  train_data, test_data = indexed_df.randomSplit([0.8, 0.2], seed=123)

  # 准备分类器（在此使用决策树模型）
  classifier = DecisionTreeClassifier(featuresCol="features", maxBins=16,
impurity="gini", seed=10)

  # 训练决策树模型
  dtc_model = classifier.fit(train_data)

  # 在训练集和测试集上进行预测
  train_predictions = dtc_model.transform(train_data)
  test_predictions = dtc_model.transform(test_data)

  # 使用 MulticlassClassificationEvaluator 计算指标
  evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
  accuracy = evaluator.evaluate(test_predictions)
  print(f"准确率: {accuracy}")

  # 计算 F1 分数
  evaluator_f1 = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="f1")
  f1_score = evaluator_f1.evaluate(test_predictions)
  print(f"F1 分数: {f1_score}")

  # 计算召回率
  evaluator_recall = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="weightedRecall")
  recall = evaluator_recall.evaluate(test_predictions)
  print(f"召回率: {recall}")

  # 计算精确度
  evaluator_precision = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="weightedPrecision")
  precision = evaluator_precision.evaluate(test_predictions)
  print(f"精确度: {precision}")
```

- 结果输出

```
# user @ ubuntu in /opt/software/spark-3.5/bin [22:44:15]
$ spark-submit task3.py
23/12/25 23:13:41 WARN NativeCodeLoader: Unable to load native-
pplicable
准确率： 0.9206442166910688
F1 分数： 0.8826057073568289
召回率： 0.9206442166910688
精确度： 0.8475857737267116
```

> 我们可以看出准确率、F1 Score、REcall和precision都在0.8以上，说明这次模型的预测能力较强。