

Diabetic Data Analysis

By Carlos Velasco

Executive Summary

TA hospital system wants to analyze patient data to reduce readmission rates, optimize treatment outcomes, and detect patterns in hospital operations. The existing data is large and semi-structured, and the hospital wants to build a scalable data engineering pipeline using PySpark on Databricks:

- **diabetic_data.csv**: ~100,000 patient records.
- Data Details (Selected Columns):
 - race, gender, age
 - admission_type_id, discharge_disposition_id, admission_source_id
 - time_in_hospital, num_lab_procedures, num_procedures, num_medications
 - number_outpatient, number_emergency, number_inpatient • diag_1, diag_2, diag_3 (Diagnosis codes)
 - readmitted (Yes/<30/>30/No)
 - change, diabetesMed, insulin, A1Cresult

Objectives

1. Data Ingestion: :

- Load the CSV into Databricks using PySpark.

2. Data Cleansing:

- Handle missing values like "?" in race, gender, diag_1, etc.
- Remove or analyze invalid entries (e.g., gender = "Unknown/Invalid").

3. Data Transformation:

- Convert categorical columns into meaningful values (e.g., map admission types).
- Derive new columns such as:
 - o `total_visits = number_outpatient + number_emergency + number_inpatient`
 - o `readmission_flag = 1` if readmitted in ("`<30`", "`>30`") else 0

4. Exploratory Analysis:

- Average time in hospital per age group or diagnosis.
- Identify which diagnoses have high readmission rates.
- Relationship between insulin/A1C levels and readmission.

Objectives


5. Performance Optimization:

- Implement data partitioning by age or readmission_flag.
- Cache intermediate results for repeated analysis.

6. Data Export:

- Save cleaned dataset as Parquet/Delta.
- Export analytical summaries as CSV for BI tools.

Data Ingestion

- **Methodology:** Load the CSV into Databricks using PySpark.
 - **Tools:** PySpark
 - **Process:** Data Collection
 - **Data Source:**
- 
- A screenshot of a Databricks workspace interface. The left sidebar shows a 'Workspace' tab with a file named 'enestrovaleco73@gmail.com'. The main area displays a PySpark script with the following code:
- ```
StructField("layer_code", StringType(), True),
StructField("medical_specialty", StringType(), True),
```
- The script is running in a 'Python' environment, as indicated by the 'Python' label in the top right corner of the code editor.

[illegible]

# Data Cleaning

- Handle missing values like "?" in race, gender, diag\_1, etc.
- Remove or analyze invalid entries (e.g., gender = "Unknown/Invalid").
- Data Source:

**Handling Missing Values ("?")**

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

Columns to check for "?"
columns_to_clean = ["race", "gender", "diag_1", "diag_2", "diag_3"]

Replace "?" with NA
for column in columns_to_clean:
 df = df.withColumn(column,
 when(col(column) == "?", None)
 .otherwise(col(column)))
```

**Check Missing Values After Cleaning**

```
from pyspark.sql.functions import count, isnull

Count NA's per column
df.select(count(isnull(c), c).alias(c) for c in df.columns).show()
```

**Handling Invalid Entries**

```
Check current gender distribution
df.groupby("gender").count().show()

Remove "Unknown/Invalid" entries
df_cleaned = df.filter(col("gender") != "Unknown/Invalid")

Replace "Unknown/Invalid" with "Other"
df = df.withColumn("gender",
 when(col("gender") == "Unknown/Invalid", "Other")
 .otherwise(col("gender")))
```

**Impute Missing Values**

```
from pyspark.sql.functions import most_frequent

Impute "race" with mode (most frequent category)
df.groupby("race").count().orderBy("count", ascending=False).show() # Check mode

df = df.na.fill({"race": "Caucasian"}) # Replace NA's with "Caucasian"
```

**Check Missing Values After Cleaning**

```
from pyspark.sql.functions import count, isnull

Count NA's per column
df.select(count(isnull(c), c).alias(c) for c in df.columns).show()
```

**diabetic\_data**

| encounter_id | patient_id | race     | gender | age | weight | admission_type_id | discharge_disposition_id | time_in_hospital | payer_code | medical_specialty         | num_lab_procs | num_visits | readmission_flag | readmission_type |
|--------------|------------|----------|--------|-----|--------|-------------------|--------------------------|------------------|------------|---------------------------|---------------|------------|------------------|------------------|
| 1            | 1          | White    | Male   | 45  | 180    | 1                 | 1                        | 10               | 1          | Internal Medicine         | 1             | 1          | 0                | None             |
| 2            | 2          | Black    | Female | 32  | 150    | 2                 | 2                        | 5                | 2          | Obstetrics and Gynecology | 2             | 2          | 1                | Re-admission     |
| 3            | 3          | Hispanic | Male   | 55  | 220    | 3                 | 3                        | 15               | 3          | Cardiology                | 3             | 3          | 0                | None             |
| 4            | 4          | White    | Female | 28  | 160    | 1                 | 1                        | 8                | 1          | Internal Medicine         | 1             | 1          | 0                | None             |
| 5            | 5          | Black    | Male   | 60  | 250    | 4                 | 4                        | 20               | 4          | Neurology                 | 4             | 4          | 1                | Re-admission     |

# Data Transformation

- Convert categorical columns into meaningful values (e.g., map admission types).
- Derive new columns such as:

1) `total_visits` = `number_outpatient` + `number_emergency` + `number_inpatient`  
2) `readmission_flag` = 1 if readmitted in ("`<30`", "`>30`") else 0

- Data Source:

The image displays four screenshots of a Databricks workspace, illustrating the steps of data transformation:

- Convert Categorical Columns to Meaningful Values:** This notebook shows the creation of a mapping dictionary for admission types. The dictionary maps codes like "1: 'Emergency'", "2: 'Urgent'", "3: 'Elective'", "4: 'Readmission'", "5: 'Not Available'", "6: 'N/A'", "7: 'Trauma Center'", and "8: 'Unknown'" to a common format. This dictionary is then used to create a Spark map type and apply it to the `admission_type_id` column in the `diabetic_data` table.
- Calculate total\_visits:** This notebook demonstrates how to calculate the total number of visits by summing the `number_outpatient`, `number_emergency`, and `number_inpatient` columns using the `coalesce` function.
- Create readmission\_flag:** This notebook shows how to create a new column, `readmission_flag`, based on the `readmitted` column. It uses a conditional statement to set the flag to 1 if the patient was readmitted within 30 days and 0 otherwise.
- Verifying Transformations:** This notebook shows the results of the transformations. It displays the transformed data with columns `admission_type_desc`, `number_outpatient`, `number_emergency`, `number_inpatient`, `total_visits`, and `readmission_flag`. It also shows the value counts for the `readmission_flag` column.
- Saving Transformed Data:** This notebook shows the final step of saving the transformed data to a Delta table. It uses the `df.write.mode("overwrite").parquet()` command to save the data to the path `dbfs:/Users/menestrelasco7@gmail.com/diabetic_data_processed`.

# Exploratory Analysis

- Average time in hospital per age group or diagnosis.
- Identify which diagnoses have high readmission rates.
- Relationship between insulin/A1C levels and readmission.
- Data Source:

The image displays four screenshots of a Databricks workspace, each showing a different SQL query for exploratory data analysis. The workspace interface includes a sidebar with navigation options like 'Workspace', 'Bakehouse Sales Starter Space', 'Diabetic Data Project', 'diabetic\_data.csv', and 'Workspace Usage Dashboard'.

### Average Time in Hospital per Age Group or Diagnosis

```
1 from pyspark.sql.functions import avg, round
2
3 # By age group
4 avg_time_by_age = df.groupBy("age_group")
5 .agg(round(avg("time_in_hospital"), 2).alias("avg_hospital_days")) \
6 .orderBy("avg_hospital_days", ascending=False)
7
8 print("Average Hospital Stay by Age Group:")
9 avg_time_by_age.show()
10
11 # By diagnosis category
12 avg_time_by_diag = df.groupBy("diag_1_category") \
13 .agg(round(avg("time_in_hospital"), 2).alias("avg_hospital_days")) \
14 .orderBy("avg_hospital_days", ascending=False)
15
16 print("Average Hospital Stay by Diagnosis Category:")
17 avg_time_by_diag.show()
```

● [UNSOLVED\_COLUMN\_WITH\_SUGGESTION] A column, variable, or function parameter with name 'age\_group' cannot be resolved. Did you mean one of the following? ['age', 'gender', 'weight', 'diag\_1', 'diag\_2']. SQLSTATE: 42703  
File command: 874682630250157, line 4  
1 from pyspark.sql.functions import avg, round  
2 by age\_group

### Diagnoses with High Readmission Rates

```
1 from pyspark.sql.functions import count, sum
2
3 # Calculate readmission rates by diagnosis
4 readmission_rates = df.groupBy("diag_1_category") \
5 .agg(
6 count("").alias("total_cases"),
7 sum("readmission_flag").alias("readmitted_cases"),
8 (sum("readmission_flag")/count("").alias("readmission_rate"))
9) \
10 .orderBy("readmission_rate", ascending=False)
11
12 print("Diagnoses with Highest Readmission Rates:")
13 readmission_rates.show()
14
15 # Filter for significant results (e.g., 1000 cases)
16 print("Unsignificant Diagnoses (1000 cases):")
17 readmission_rates.filter(col("total_cases") > 1000).show()
```

● [UNSOLVED\_COLUMN\_WITH\_SUGGESTION] A column, variable, or function parameter with name 'diag\_1\_category' cannot be resolved. Did you mean one of the following? ['diag\_1', 'diag\_2', 'diag\_3', 'payer\_code', 'acohouse']. SQLSTATE: 42703  
File command: 874682630250157, line 4  
1 from pyspark.sql.functions import count, sum

### Insulin/A1C Levels and Readmission Relationship

```
1 from pyspark.sql.functions import count
2
3 # Insulin analysis
4 insulin_readmission = df.groupBy("insulin", "readmission_flag") \
5 .agg(count("").alias("count")) \
6 .orderBy("insulin", "readmission_flag")
7
8 print("Readmission by Insulin Status:")
9 insulin_readmission.show()
10
11 # A1C analysis (assuming A1C results are available)
12 if "A1Cresult" in df.columns:
13 a1c_readmission = df.groupBy("A1Cresult", "readmission_flag") \
14 .agg(count("").alias("count")) \
15 .orderBy("A1Cresult", "readmission_flag")
16
17 print("Readmission by A1C Results:")
18 a1c_readmission.show()
19 else:
20 print("A1Cresult column not found in dataset")
21
22 # Statistical test example (using chi-square)
23 from pyspark.ml.stat import ChiSquareTest
24 from pyspark.ml.feature import VectorAssembler
25 from pyspark.ml import Pipeline
```

● Last execution failed 26 Python

```
25 from pyspark.ml import Pipeline
26
27 # Prepare data for insulin analysis
28 assembler = VectorAssembler()
29 inputCols = ["insulin_index"]; # Need to convert to numeric index first
30 outputCol = "features"
31
32 pipeline = Pipeline(stages=[assembler])
33 model = pipeline.fit(df)
34 data = model.transform(df)
35
36 # Run chi-square test
37 c = ChiSquareTest.test(data, ["features", "readmission_flag"].head())
38 print(f"Unk... ensure p-value for insulin relationship: {p.value}")
39
40 [UNSOLVED_COLUMN_WITH_SUGGESTION] A column, variable, or function parameter with name 'readmission_flag' cannot be resolved. Did you mean one of the following? ['admission_type_id', 'readmitted', 'admission_source_id', 'admission_type_desc', 'weightkg']. SQLSTATE: 42703
File command: 874682630250157, line 4
1 from pyspark.sql.functions import count
2 # Insulin analysis
3 insulin_readmission = df.groupBy("insulin", "readmission_flag") \
4 .agg(count("").alias("count")) \
5 .orderBy("insulin", "readmission_flag")
6 print("Readmission by Insulin Status:")
7 insulin_readmission.show()
```

File: /databricks/pyspark/llm/packages/pyspark-sql/connect/client/core.py:210, in SparkConnectClient\_handle\_rpc\_error(self, rpc\_error)



## Performance Optimization

- Implement data partitioning by age or readmission\_flag.
- Cache intermediate results for repeated analysis.
- Data Source:

The image displays two screenshots of a Jupyter Notebook interface, showing code execution for a Spark application. The top screenshot is titled "Partition by Readmission Flag" and the bottom screenshot is titled "Cache Frequently Used DataFrames".

**Top Screenshot: Partition by Readmission Flag**

The code in this section defines a function `df_partitioning` that partitions a DataFrame by the `readmission_flag` column. It then executes the function on a DataFrame named `df` and displays the result as a table with 28 rows.

```
def df_partitioning(df, readmission_flag):
 mode = "overwrite"

 spark.write \
 .partitionBy("readmission_flag") \
 .mode(mode) \
 .saveAsTable("workspace/users/mestrowelavo73@gmail.com/diabetic_data_partitioned_by_readmission")

See performance (1)
df_partitioning(df, readmission_flag)
```

The output table shows columns: `id`, `readmission_flag`, `diabetes_location_code`, `diabetes_code`, `diabetes_code_2`, `diabetes_code_3`, `diabetes_code_4`, `diabetes_code_5`, `diabetes_code_6`, `diabetes_code_7`, `diabetes_code_8`, `diabetes_code_9`, `diabetes_code_10`, `diabetes_code_11`, `diabetes_code_12`, `diabetes_code_13`, `diabetes_code_14`, `diabetes_code_15`, `diabetes_code_16`, `diabetes_code_17`, `diabetes_code_18`, `diabetes_code_19`, `diabetes_code_20`, `diabetes_code_21`, `diabetes_code_22`, `diabetes_code_23`, `diabetes_code_24`, `diabetes_code_25`, `diabetes_code_26`, `diabetes_code_27`, `diabetes_code_28`. The `readmission_flag` column has values 0, 1, and 2.

**Bottom Screenshot: Cache Frequently Used DataFrames**

The code in this section defines a function `df_cached` that caches a DataFrame. It then executes the function on a DataFrame named `df` and displays the result as a table with 30 rows.

```
def df_cached(df):
 spark.write \
 .partitionBy("readmission_flag") \
 .mode(mode) \
 .saveAsTable("workspace/users/mestrowelavo73@gmail.com/diabetic_data_partitioned_by_readmission")

See performance (1)
df_cached(df)
```

The output table shows columns: `id`, `readmission_flag`, `diabetes_location_code`, `diabetes_code`, `diabetes_code_2`, `diabetes_code_3`, `diabetes_code_4`, `diabetes_code_5`, `diabetes_code_6`, `diabetes_code_7`, `diabetes_code_8`, `diabetes_code_9`, `diabetes_code_10`, `diabetes_code_11`, `diabetes_code_12`, `diabetes_code_13`, `diabetes_code_14`, `diabetes_code_15`, `diabetes_code_16`, `diabetes_code_17`, `diabetes_code_18`, `diabetes_code_19`, `diabetes_code_20`, `diabetes_code_21`, `diabetes_code_22`, `diabetes_code_23`, `diabetes_code_24`, `diabetes_code_25`, `diabetes_code_26`, `diabetes_code_27`, `diabetes_code_28`. The `readmission_flag` column has values 0, 1, and 2.

Workspace

← enstrovlawco73@gmail.com

Workspace

Bakuhouse Sales Starter Space

Diabetic Data Project

diabetic\_data.csv

Workspace Usage Dashboard

Optimized Analysis Pipeline

Python

1 from pySpark.sql.functions import broadcast  
2  
3 def optimized\_analysis()  
4 # Load partitioned data (example for age)  
5 age\_partitioned = spark.read.parquet(  
6 "hdfs://user:enstrovlawco73@gmail.com:diabetic\_data/partitioned\_by\_age"  
7 )  
8  
9 # Cache partitioned data  
10 age\_partitioned = age\_partitioned.filter(col("age\_group") in ["minor","cache"])  
11  
12 # Broadcast small lookup tables  
13 diag\_lookup = spark.table("diagnosis\_codes") # Assuming single  
14 broadcast\_diag = broadcast(diag\_lookup)  
15  
16 # Join with enhanced data  
17 enhanced\_data = age\_partitioned.join(  
18 broadcast\_diag,  
19 age\_partitioned.diag\_1 == broadcast\_diag.code,  
20 "left"  
21 )  
22  
23 # Cache final enhanced data  
24 enhanced\_data.cache()  
25  
26 return enhanced\_data  
27  
28 # Run optimized analysis  
29 enhanced\_df = optimized\_analysis()  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
78

The screenshot shows a PyCharm IDE with the following details:

- Top Bar:** Project name "Diabetic Data Project", Python version "Python 3.7.4".
- File Explorer (Left):**
  - Workspace
  - ← enestorelaco7@gmail.com
  - Bahakouse Sales Starter Space
  - Diabetic Data Project (selected)
  - diabetic\_data.py
  - Workspace Usage Dashboard
- Main Editor:**

# Monitoring Cache Usage

```

1 # 检查存储内存占用
2 storage_level = df.storageLevel
3 print("Storage Level: {storage_level}")
4 print("Memory Used: Access to sparkContext is not supported in serverless compute.")
5
6 # 清除缓存当内存
7 df.unpersist()
8 avg_time_by_age.unpersist()

```

Storage Level: Serialized In Replicated  
Memory Used: Access to sparkContext is not supported in serverless compute.

```

9 > [NOT SUPPORTED WITH SERVERLESS] UNPERSIST TABLE is not supported on serverless compute. SQLSTATE: 08000
10
11 # 命令 414663692583987070, line 7
12 print("Memory Used: Access to sparkContext is not supported in serverless compute.")
13
14 # 清除缓存当内存
15 --> 7 df.unpersist()
16 # avg_time_by_age.unpersist()

```

File |diabetic\_data|python/lib/python3.7/site-packages/pyspark/sql/connect/client/core.py:2049, in SparkConnectClient.\_band  
In pySpark\_core(self, pySpark\_core)

```

2134 raise Exception(
2135 "Python versions in the Spark Connect client and server are different."
2136 "To execute user-defined functions, client and server should have the "

```



# Conclusion

**Completed Work:** Was able to do the Data Ingestion, Data Cleansing, Data Transformation, Exploratory Analysis, Performance Optimization, Data Export. But, I did run into some connection errors in the code.

**Moving forward:** Will need to fix the DBFS root error I kept running into and the other errors I was running into.