

Case Study 4 – Working with Sensor Data

The dataset for the case study is the following csv file.

1. inpatientCharges.csv

This csv file is loaded onto HDFS as follows.

```
hadoop fs -put inpatientCharges.csv '/user/acadgild/hadoop/Hospital/'
```

```
[acadgild@localhost Hospital]$ hadoop fs -put inpatientCharges.csv '/user/acadgild/hadoop/Hospital/'
18/07/04 23:29:55 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost Hospital]$ hadoop fs -ls '/user/acadgild/hadoop/Hospital/'
18/07/04 23:42:58 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
Found 1 items
-rw-r--r-- 1 acadgild supergroup 26870105 2018-07-04 23:29 /user/acadgild/hadoop/Hospital/inpatientCharges.csv
You have new mail in /var/spool/mail/acadgild
```

On listing the contents of the Hospital folder on HDFS, we can see that the csv file is present.

Objective 1:

Load file into spark

```
scala> val session = org.apache.spark.sql.SparkSession.builder.master("local").appName("Load CSV").getOrCreate();
18/07/04 00:10:25 WARN sql.SparkSessionBuilder: Using an existing SparkSession; some configuration may not take effect.
session: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@2b625e82

scala> val hospitaldf = session.read.format("com.databricks.spark.csv").option("header", "true").option("inferSchema", "true").
load("/user/acadgild/hadoop/Hospital/inpatientCharges.csv")
hospitaldf: org.apache.spark.sql.DataFrame = [DRGDefinition: string, ProviderId: int ... 10 more fields]

scala> hospitaldf.show
```

DRGDefinition	ProviderId	ProviderName	ProviderStreetAddress	ProviderCity	ProviderState	ProviderZipCode	Hospit
alReferralRegionDescription	TotalDischarges	AverageCoveredCharges	AverageTotalPayments	AverageMedicarePayments			
039 - EXTRACRANIA...	10001	SOUTHEAST ALABAMA...	1108 ROSS CLARK C...	DOETHAN	AL	36301	
AL - Dothan	91	32963.07	5777.24	4763.73		35957	
039 - EXTRACRANIA...	10005	MARSHALL MEDICAL ...	2505 U S HIGHWAY ...	BOAZ	AL	35957	
AL - Birmingham	14	15131.85	5787.57	4976.71		35631	
039 - EXTRACRANIA...	10006	ELIZA COFFEE MEMO...	205 MARENGO STREET	FLORENCE	AL	35631	
AL - Birmingham	24	37560.37	5434.95	4453.79		35235	
039 - EXTRACRANIA...	10011	ST VINCENT'S EAST	50 MEDICAL PARK E...	BIRMINGHAM	AL	35235	
AL - Birmingham	25	13998.28	5417.56	4129.16		35007	
039 - EXTRACRANIA...	10016	SHELBY BAPTIST ME...	1000 FIRST STREET...	ALABASTER	AL	35007	
AL - Birmingham	18	31633.27	5658.33	4851.44		36116	
039 - EXTRACRANIA...	10023	BAPTIST MEDICAL C...	2105 EAST SOUTH B...	MONTGOMERY	AL	36116	
AL - Montgomery	67	16920.79	6653.8	5374.14		36801	
039 - EXTRACRANIA...	10029	EAST ALABAMA MEDI...	2000 PEPPERELL PA...	OPELIKA	AL	36801	
AL - Birmingham	51	11977.13	5834.74	4761.41		35233	
039 - EXTRACRANIA...	10033	UNIVERSITY OF ALA...	619 SOUTH 19TH ST...	BIRMINGHAM	AL	35233	
AL - Birmingham	32	35841.09	8031.12	5858.5		35801	
039 - EXTRACRANIA...	10039	HUNTSVILLE HOSPITAL	101 SIVLEY RD	HUNTSVILLE	AL	35801	
AL - Huntsville	135	28523.39	6113.38	5228.4		35903	
039 - EXTRACRANIA...	10040	GADSDEN REGIONAL ...	1007 GOODYEAR AVENUE	GADSDEN	AL	35903	
AL - Birmingham	34	75233.38	5541.05	4386.94		35901	
039 - EXTRACRANIA...	10046	RIVERVIEW REGIONA...	600 SOUTH THIRD S...	GADSDEN	AL	35901	
AL - Birmingham	14	67327.92	5461.57	4493.57		36305	
039 - EXTRACRANIA...	10055	FLOWERS HOSPITAL	4370 WEST MAIN ST...	DOETHAN	AL	36305	
AL - Dothan	45	39607.28	5356.28	4408.2			

We have loaded all the CSV data as a DataFrame into Spark SQL. Here, we have used `inferSchema` as an option so it will automatically infer the data type of the columns.

```
val session =
org.apache.spark.sql.SparkSession.builder.master("local").appName("Spark CSV
Reader").getOrCreate;

val hospitaldf = session.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema",
"true").load("/user/acadgild/hadoop/Hospital/inpatientCharges.csv")
```

We can see the contents of the dataframe in the screenshot above using the following command.

```
hospitaldf.show
```

Objective 2:

- A. What is the average amount of AverageCoveredCharges per state
- B. Find out the AverageTotalPayments charges per state
- C. Find out the AverageMedicarePayments charges per state.

- A. What is the average amount of AverageCoveredCharges per state

```
scala>
scala> hospitaldf.groupBy("ProviderState").avg("AverageCoveredCharges").show
```

ProviderState	avg(AverageCoveredCharges)
AZ	41200.063019992995
SC	35862.49456269756
LA	33085.372791542846
MN	27894.36182060388
NJ	66125.68627434729
DC	40116.66365800864
OR	27390.111870669723
VA	29222.000487072903
RI	29942.701122448976
KY	24523.80716940223
WY	28700.59862348178
NH	27059.020801944105
MI	24124.247209817277
NV	61047.11541597337
WI	26149.325331686607
ID	25565.547041742288
CA	67508.616535517
CT	31318.4101143709
NE	31736.427824858758
MT	22670.015237154144

only showing top 20 rows

```
hospitaldf.groupBy("ProviderState").avg("AverageCoveredCharges").show
```

groupBy and avg functions are used on the dataframe to calculate the average per state. The average amount of AverageCoveredCharges per state is shown as output in the screenshot above.

B. Find out the AverageTotalPayments charges per state

```
scala>
scala> hospitaldf.groupBy("ProviderState").avg("AverageTotalPayments").show
```

ProviderState	avg(AverageTotalPayments)
AZ	10154.528211153991
SC	9132.420758693366
LA	8638.66257680871
MN	9948.236962699833
NJ	10678.98864691253
DC	12998.029415584406
OR	10436.192863741335
VA	8887.75217682364
RI	10509.566853741484
KY	8278.58884484363
WY	11398.485910931167
NH	9289.661822600248
MI	9754.420405978948
NV	10291.718028286188
WI	9270.705617501746
ID	9827.180090744107
CA	12629.668472137122
CT	11365.450671307795
NE	9331.682523540492
MT	9252.802766798422

only showing top 20 rows

```
hospitaldf.groupBy("ProviderState").avg("AverageTotalPayments").show
```

groupBy and avg functions are used on the dataframe to calculate the average per state. The average amount of AverageTotalPayments per state is shown as output in the screenshot above.

C. Find out the AverageMedicarePayments charges per state.

```
scala>
scala> hospitaldf.groupBy("ProviderState").avg("AverageMedicarePayments").show
```

ProviderState	avg(AverageMedicarePayments)
AZ	8825.717239565045
SC	7876.33152441167
LA	7387.704625041281
MN	8619.214982238007
NJ	9586.940055946912
DC	11811.967705627709
OR	9035.259961508847
VA	7538.847006001846
RI	9317.939115646255
KY	7185.227810467647
WY	9539.392024291496
NH	8124.506852976913
MI	8662.157756043543
NV	8747.602828618963
WI	8002.597911079731
ID	8461.977513611617
CA	11494.381677893474
CT	10104.592943809059
NE	7992.6272504707995
MT	7981.088063241104

only showing top 20 rows

hospitaldf.groupBy("ProviderState").avg("AverageMedicarePayments").show

groupBy and avg functions are used on the dataframe to calculate the average per state. The average amount of AverageMedicarePayments per state is shown as output in the screenshot above.

Objective 3:

- A. Find out the total number of Discharges per state and for each disease
- B. Sort the output in descending order of totalDischarges

- A. Find out the total number of Discharges per state and for each disease

```
scala>
scala> hospitaldf.groupBy(("ProviderState"),("DRGDefinition")).sum("TotalDischarges").show
18/07/05 00:35:13 WARN executor.Executor: Managed memory leak detected; size = 17039360 bytes, TID = 511
```

ProviderState	DRGDefinition	sum(TotalDischarges)
KY 065	- INTRACRANIA...	1937
NY 101	- SEIZURES W/...	4503
IN 149	- DYSEQUILIBRIUM	700
IA 178	- RESPIRATORY...	540
WI 202	- BRONCHITIS ...	338
MO 208	- RESPIRATORY...	1840
WI 251	- PERC CARDIO...	417
AR 281	- ACUTE MYOCA...	413
AZ 292	- HEART FAILU...	2643
NY 292	- HEART FAILU...	13289
NV 293	- HEART FAILU...	519
SD 303	- ATHEROSCLER...	53
TN 305	- HYPERTENSIO...	730
ME 308	- CARDIAC ARR...	312
NV 372	- MAJOR GASTR...	126
WA 392	- ESOPHAGITIS...	3148
WI 439	- DISORDERS O...	215
MN 536	- FRACTURES O...	332
DC 563	- FX, SPRN, S...	43
CO 602	- CELLULITIS ...	86

only showing top 20 rows

```
hospitaldf.groupBy(("ProviderState"),("DRGDefinition")).sum("TotalDischarges").s
how
```

Groupby is applied on a combination of state and DRG definition and then the sum is calculated for each group. The sum of totalDischarges per state and for each disease is shown in the screenshot above.

B. Sort the output in descending order of totalDischarges

```
scala>
scala> hospitaldf.groupBy(("ProviderState"),("DRGDefinition")).sum("TotalDischarges").orderBy(desc(sum("TotalDischarges").toString)).show
```

ProviderState	DRGDefinition	sum(TotalDischarges)
CA 871	- SEPTICEMIA ...	34284
TX 470	- MAJOR JOINT...	30095
FL 470	- MAJOR JOINT...	29985
CA 470	- MAJOR JOINT...	29731
TX 871	- SEPTICEMIA ...	23144
NY 871	- SEPTICEMIA ...	21970
FL 392	- ESOPHAGITIS...	21298
IL 470	- MAJOR JOINT...	20095
NY 470	- MAJOR JOINT...	19371
FL 871	- SEPTICEMIA ...	18660
TX 690	- KIDNEY & UR...	17384
NY 392	- ESOPHAGITIS...	17337
MI 470	- MAJOR JOINT...	16847
PA 470	- MAJOR JOINT...	16712
FL 292	- HEART FAILU...	16639
FL 690	- KIDNEY & UR...	16405
OH 470	- MAJOR JOINT...	16062
NC 470	- MAJOR JOINT...	15820
IL 871	- SEPTICEMIA ...	15610
MI 871	- SEPTICEMIA ...	15548

only showing top 20 rows

```
hospitaldf.groupBy(("ProviderState"),("DRGDefinition")).sum("TotalDischarges").s
how .orderBy(desc(sum("TotalDischarges").toString)).show
```

On the above query, orderBy(desc) is applied on the TotalDischarges column to sort the results in a descending order.