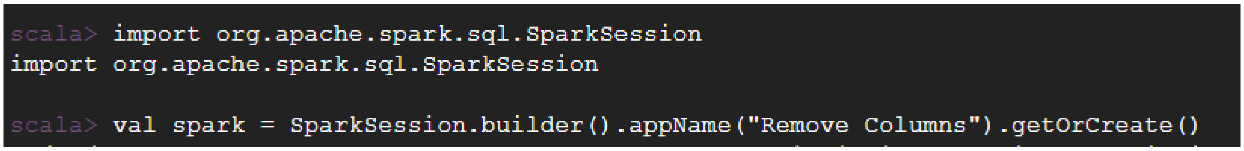
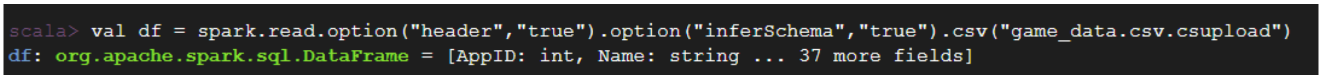
Data Cleaning:

1. Remove unnecessary columns:



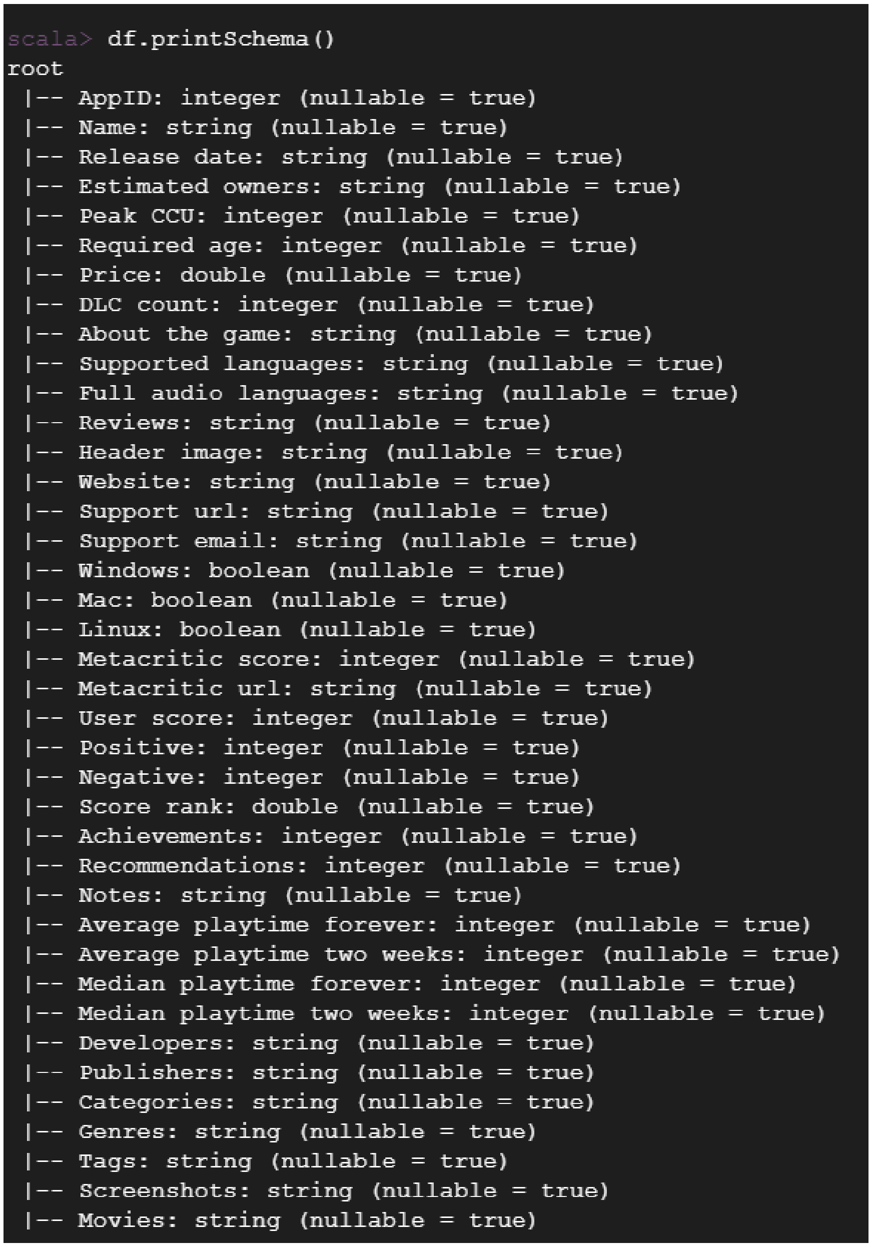
```
scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala> val spark = SparkSession.builder().appName("Remove Columns").getOrCreate()
```

- Read in the original csv file. We can see it has over 37 columns.



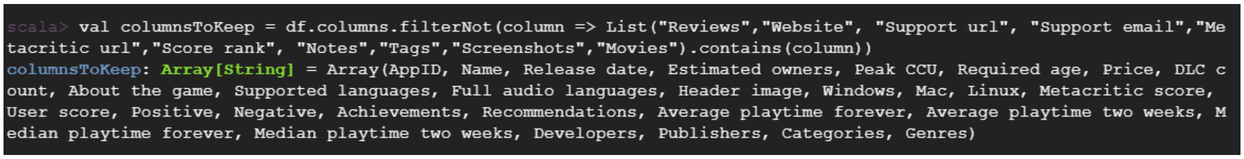
```
scala> val df = spark.read.option("header","true").option("inferSchema","true").csv("game_data.csv.csupload")
df: org.apache.spark.sql.DataFrame = [AppID: int, Name: string ... 37 more fields]
```

- If we print them, there are all the columns.



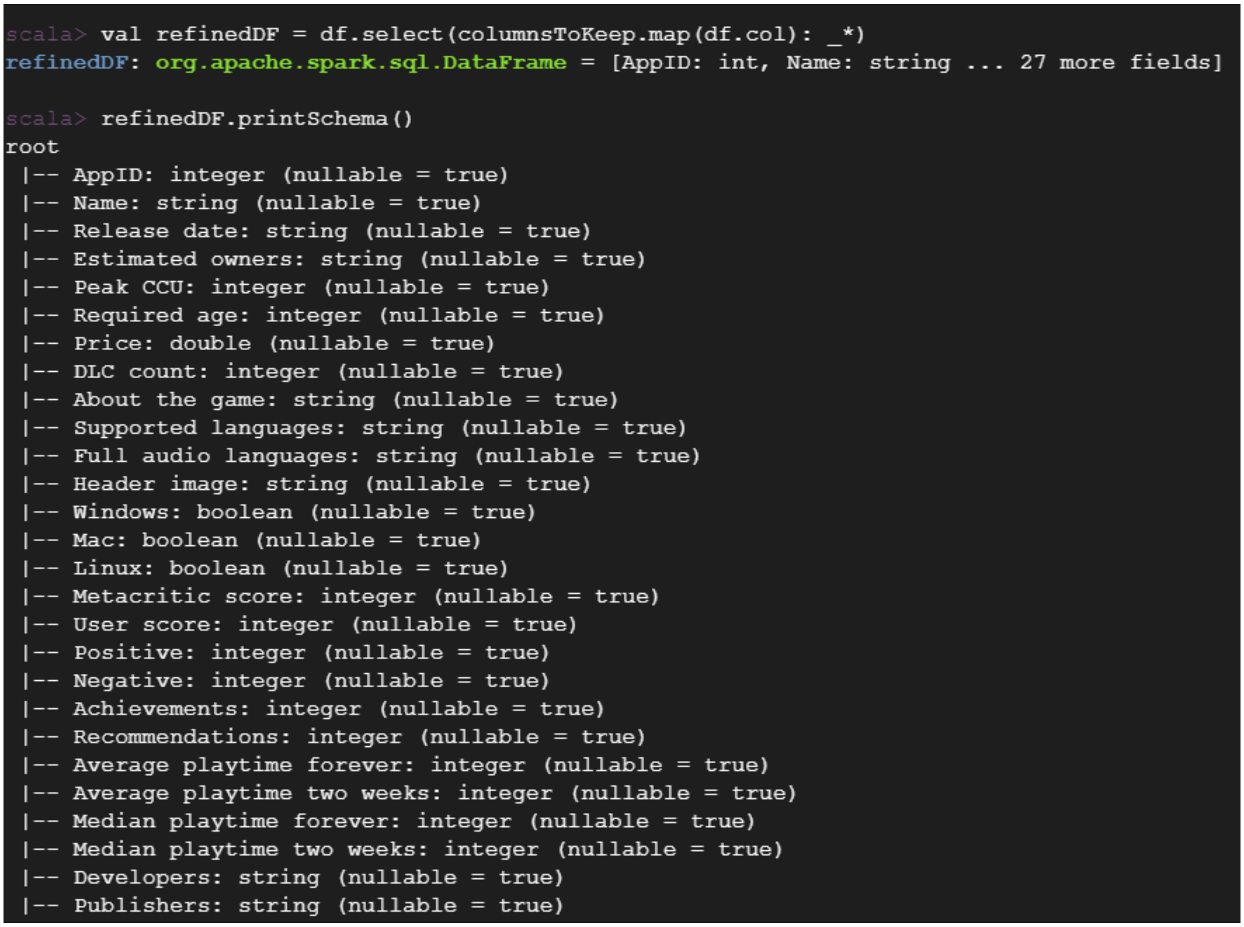
```
scala> df.printSchema()
root
 |-- AppID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Release date: string (nullable = true)
 |-- Estimated owners: string (nullable = true)
 |-- Peak CCU: integer (nullable = true)
 |-- Required age: integer (nullable = true)
 |-- Price: double (nullable = true)
 |-- DLC count: integer (nullable = true)
 |-- About the game: string (nullable = true)
 |-- Supported languages: string (nullable = true)
 |-- Full audio languages: string (nullable = true)
 |-- Reviews: string (nullable = true)
 |-- Header image: string (nullable = true)
 |-- Website: string (nullable = true)
 |-- Support url: string (nullable = true)
 |-- Support email: string (nullable = true)
 |-- Windows: boolean (nullable = true)
 |-- Mac: boolean (nullable = true)
 |-- Linux: boolean (nullable = true)
 |-- Metacritic score: integer (nullable = true)
 |-- Metacritic url: string (nullable = true)
 |-- User score: integer (nullable = true)
 |-- Positive: integer (nullable = true)
 |-- Negative: integer (nullable = true)
 |-- Score rank: double (nullable = true)
 |-- Achievements: integer (nullable = true)
 |-- Recommendations: integer (nullable = true)
 |-- Notes: string (nullable = true)
 |-- Average playtime forever: integer (nullable = true)
 |-- Average playtime two weeks: integer (nullable = true)
 |-- Median playtime forever: integer (nullable = true)
 |-- Median playtime two weeks: integer (nullable = true)
 |-- Developers: string (nullable = true)
 |-- Publishers: string (nullable = true)
 |-- Categories: string (nullable = true)
 |-- Genres: string (nullable = true)
 |-- Tags: string (nullable = true)
 |-- Screenshots: string (nullable = true)
 |-- Movies: string (nullable = true)
```

- Then I delete some columns in the csv file



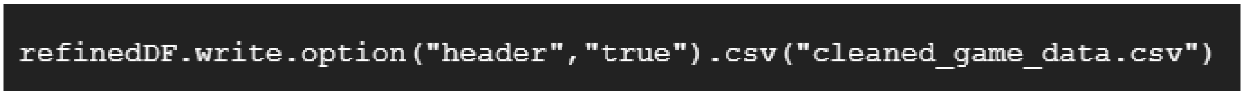
```
scala> val columnsToKeep = df.columns.filterNot(column => List("Reviews","Website", "Support url", "Support email","Me
tacritic url","Score rank", "Notes","Tags","Screenshots","Movies").contains(column))
columnsToKeep: Array[String] = Array(AppID, Name, Release date, Estimated owners, Peak CCU, Required age, Price, DLC c
ount, About the game, Supported languages, Full audio languages, Header image, Windows, Mac, Linux, Metacritic score,
User score, Positive, Negative, Achievements, Recommendations, Average playtime forever, Average playtime two weeks, M
edian playtime forever, Median playtime two weeks, Developers, Publishers, Categories, Genres)
```

- We can see there are 27 columns right now.



```
scala> val refinedDF = df.select(columnsToKeep.map(df.col): _*)
refinedDF: org.apache.spark.sql.DataFrame = [AppID: int, Name: string ... 27 more fields]

scala> refinedDF.printSchema()
root
 |-- AppID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Release date: string (nullable = true)
 |-- Estimated owners: string (nullable = true)
 |-- Peak CCU: integer (nullable = true)
 |-- Required age: integer (nullable = true)
 |-- Price: double (nullable = true)
 |-- DLC count: integer (nullable = true)
 |-- About the game: string (nullable = true)
 |-- Supported languages: string (nullable = true)
 |-- Full audio languages: string (nullable = true)
 |-- Header image: string (nullable = true)
 |-- Windows: boolean (nullable = true)
 |-- Mac: boolean (nullable = true)
 |-- Linux: boolean (nullable = true)
 |-- Metacritic score: integer (nullable = true)
 |-- User score: integer (nullable = true)
 |-- Positive: integer (nullable = true)
 |-- Negative: integer (nullable = true)
 |-- Achievements: integer (nullable = true)
 |-- Recommendations: integer (nullable = true)
 |-- Average playtime forever: integer (nullable = true)
 |-- Average playtime two weeks: integer (nullable = true)
 |-- Median playtime forever: integer (nullable = true)
 |-- Median playtime two weeks: integer (nullable = true)
 |-- Developers: string (nullable = true)
 |-- Publishers: string (nullable = true)
```

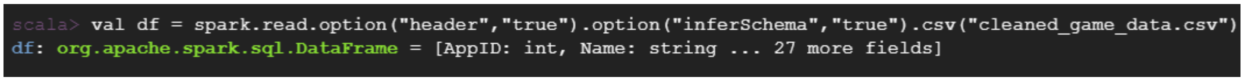
- Then I download the file back to HDFS



```
refinedDF.write.option("header","true").csv("cleaned_game_data.csv")
```

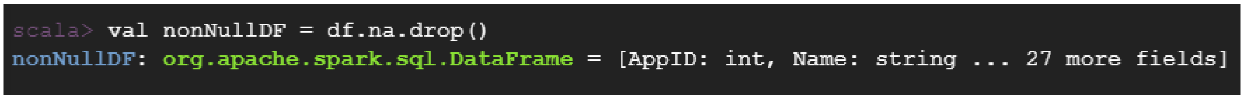
2. Remove null elements in the csv file:

- Similar to the first one, I read the csv file from the first one. Then I do



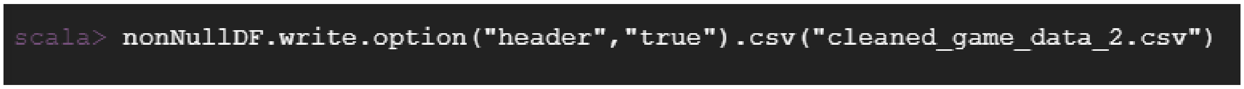
```
scala> val df = spark.read.option("header","true").option("inferSchema","true").csv("cleaned_game_data.csv")
df: org.apache.spark.sql.DataFrame = [AppID: int, Name: string ... 27 more fields]
```

- Then I just use this code to drop rows that have null values.



```
scala> val nonNullDF = df.na.drop()
nonNullDF: org.apache.spark.sql.DataFrame = [AppID: int, Name: string ... 27 more fields]
```

- Then I write the file back to HDFS, and I get a cleaned dataset, called: cleaned_game_data_2.csv

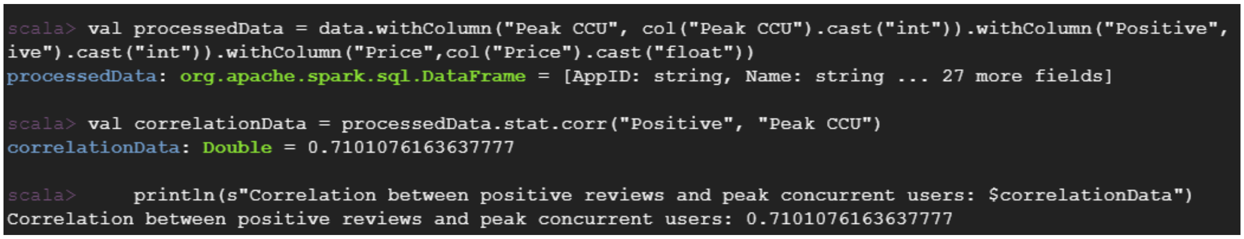


```
scala> nonNullDF.write.option("header","true").csv("cleaned_game_data_2.csv")
```

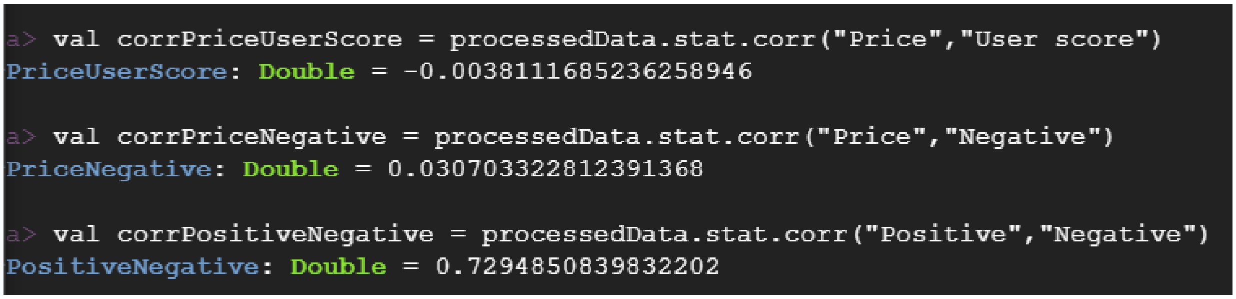
Data Analysis

1. Correlation between features

- We can see there is quite a high correlation between the “Positive” view and “Peak CCU”. There is little correlation between price and user score, price and Negative reviews.

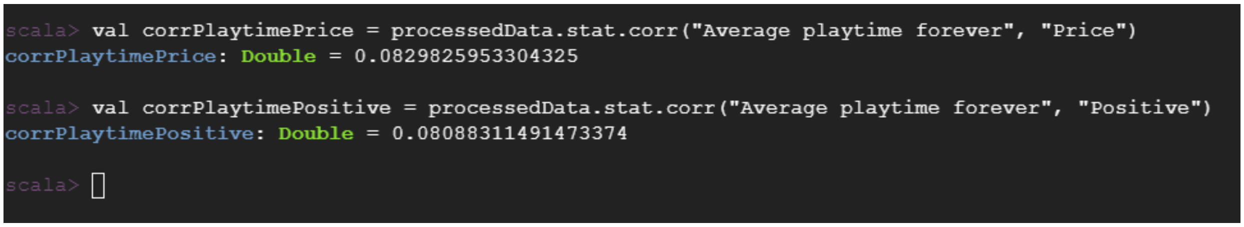


```
scala> val processedData = data.withColumn("Peak CCU", col("Peak CCU").cast("int")).withColumn("Positive",
ive").cast("int")).withColumn("Price",col("Price").cast("float"))
processedData: org.apache.spark.sql.DataFrame = [AppID: string, Name: string ... 27 more fields]

scala> val correlationData = processedData.stat.corr("Positive", "Peak CCU")
correlationData: Double = 0.7101076163637777

scala>     println(s"Correlation between positive reviews and peak concurrent users: $correlationData")
Correlation between positive reviews and peak concurrent users: 0.7101076163637777
```



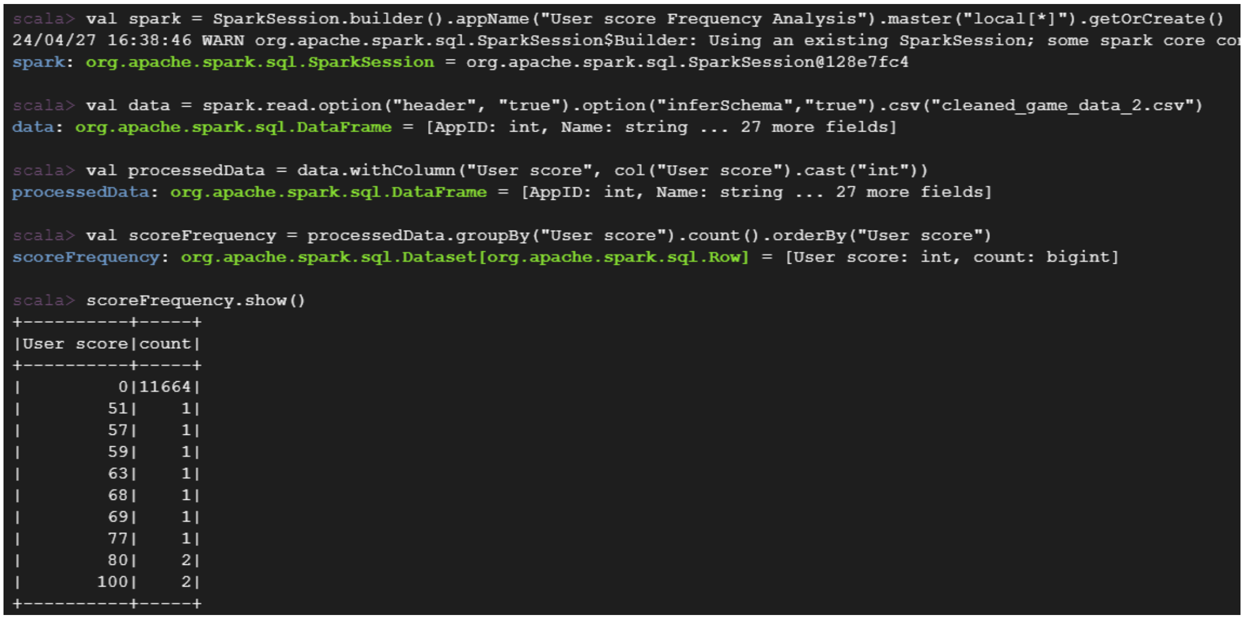
```
a> val corrPriceUserScore = processedData.stat.corr("Price","User score")
PriceUserScore: Double = -0.0038111685236258946

a> val corrPriceNegative = processedData.stat.corr("Price","Negative")
PriceNegative: Double = 0.030703322812391368

a> val corrPositiveNegative = processedData.stat.corr("Positive","Negative")
PositiveNegative: Double = 0.7294850839832202
```

- Also, there is little correlation between price and the time a player spends on a game. Also, longer a player spends on a game doesn’t mean a positive review.



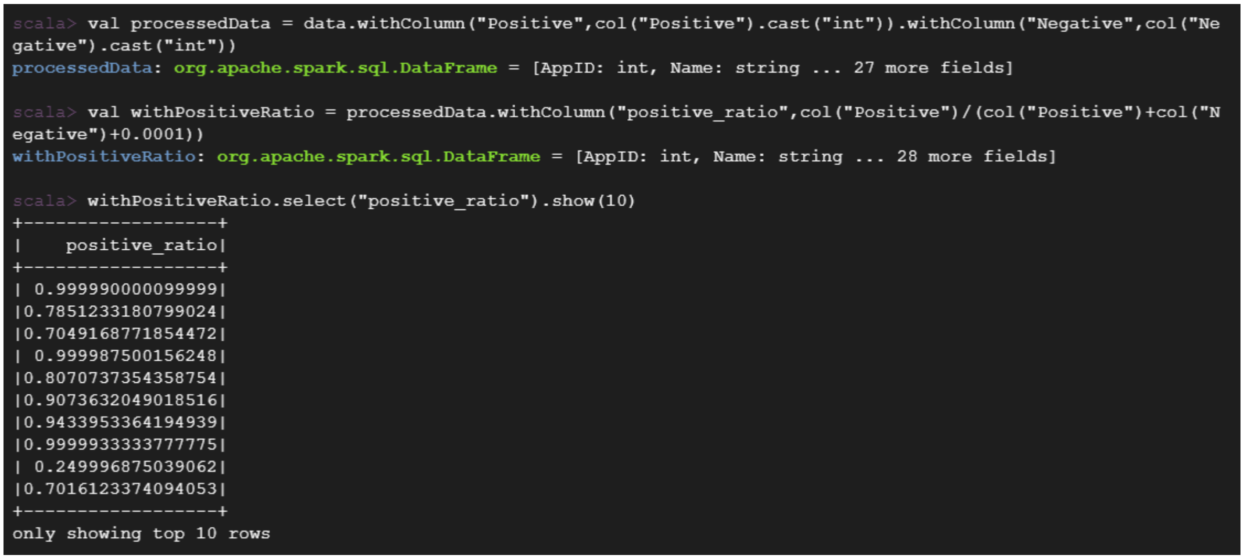
```
scala> val corrPlaytimePrice = processedData.stat.corr("Average playtime forever", "Price")
corrPlaytimePrice: Double = 0.0829825953304325

scala> val corrPlaytimePositive = processedData.stat.corr("Average playtime forever", "Positive")
corrPlaytimePositive: Double = 0.08088311491473374

scala> 
```

- I checked the “User score” column, and saw each values’ counts. Then I found that most data is zero, so this feature is also not very useful.



```
scala> val spark = SparkSession.builder().appName("User score Frequency Analysis").master("local[*]").getOrCreate()
24/04/27 16:38:46 WARN org.apache.spark.sql.SparkSession$Builder: Using an existing SparkSession; some spark core co
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@128e7fc4

scala> val data = spark.read.option("header", "true").option("inferSchema","true").csv("cleaned_game_data_2.csv")
data: org.apache.spark.sql.DataFrame = [AppID: int, Name: string ... 27 more fields]

scala> val processedData = data.withColumn("User score", col("User score").cast("int"))
processedData: org.apache.spark.sql.DataFrame = [AppID: int, Name: string ... 27 more fields]

scala> val scoreFrequency = processedData.groupBy("User score").count().orderBy("User score")
scoreFrequency: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [User score: int, count: bigint]

scala> scoreFrequency.show()
+----------+-----+
|User score|count|
+----------+-----+
|         0|11664|
|        51|    1|
|        57|    1|
|        59|    1|
|        63|    1|
|        68|    1|
|        69|    1|
|        77|    1|
|        80|    2|
|       100|    2|
+----------+-----+
```

- I created a new column called “positive_ratio”, which calculate the ratio of positive votes.



```
scala> val processedData = data.withColumn("Positive",col("Positive").cast("int")).withColumn("Negative",col("Ne
gative").cast("int"))
processedData: org.apache.spark.sql.DataFrame = [AppID: int, Name: string ... 27 more fields]

scala> val withPositiveRatio = processedData.withColumn("positive_ratio",col("Positive")/(col("Positive")+col("N
egative")+0.0001))
withPositiveRatio: org.apache.spark.sql.DataFrame = [AppID: int, Name: string ... 28 more fields]

scala> withPositiveRatio.select("positive_ratio").show(10)
+------------------+
|    positive_ratio|
+------------------+
| 0.999990000099999|
|0.7851233180799024|
|0.7049168771854472|
| 0.999987500156248|
|0.8070737354358754|
|0.9073632049018516|
|0.9433953364194939|
|0.9999933333777775|
| 0.249996875039062|
|0.7016123374094053|
+------------------+
only showing top 10 rows
```

-