**-----------How to work on pyspark SQL-------­----**

**Few important urls:-**

<https://hackersandslackers.com/join-aggregate-pyspark-dataframes/>

<http://allaboutscala.com/big-data/spark/>

<https://hackersandslackers.com/transforming-pyspark-dataframes/>

<https://sparkbyexamples.com/spark/using-lit-and-typedlit-to-add-a-literal-or-constant-to-spark-dataframe/>

<http://www.learnbymarketing.com/1100/pyspark-joins-by-example/>

<https://spark.apache.org/docs/2.2.0/sql-programming-guide.html>

**--------------------sparksql interview urls---------------------------------**

<https://www.interviewgrid.com/interview_questions/bigdata_spark/bigdata_spark_sql>

<https://data-flair.training/blogs/pyspark-interview-questions/>

**note:- comments are written after # in different color, codes are in black color.**

**Step 1) first create a data frame using csv**

**df = spark.read.csv('C:/Users/Linux/Desktop/RDD/msisdn.csv',header=True,inferSchema=True) ##reading csv file from your local machine and directory where your data has been placed, copy complete path and paste into it. df is the variable used to store read data from csv file.**

**df.printSchema() ##to display all columns and their datatype of the csv file.**

**df.show() ##show data of the csv file.**

**step 2) create a temporary view/table by using data frame (df).**

**df.createOrReplaceTempView("table1") ##put temp view name inside brackets.**

**spark.sql('select \* from table1').show() ##to display data from temp view**

**step 3) run below sql query to extract data as you want**

**df2=spark.sql("select \* from table1 limit 10") ##will display on 10 rows from temp view.**

**df2.show() ##will display data which is stored in df2 variable.**

**---------------------------reading fine from csv and making dataframe(df)-------------------------**

**df2 = spark.read.csv('C:/Users/Linux/Desktop/RDD/msisdn.csv',header=True,inferSchema=True) ##header :- used to display your csv file column header, if you will not make header True then it will not print your csv file column header.**

**##inferSchema:- to make inferSchema true it will show you accurate data type of your csv file column.**

**note:= creating a table using tem view as below to run all SQL queries on it as like Oracle/SQL second option you can write table name in double cotes also like below**

**note:= df2.createOrReplaceTempView(“subscriber”) ##to create temp view. temp view always write into double code.**

**Note:= all SQL query can write in single cotes or double cotes, but try to use always double cotes.**

**all SQL oprations can be perform on it.**

* Arithmetic operators
* Comparison operators
* Logical operators
* Operators used to negate conditions

**df3 = spark.sql('select distinct(final\_status) from subscriber') ##to select a particular columns**

**df3 = spark.sql('select \* from subscriber')**

**df3 = spark.sql('select col1,col2,col3 from subscriber')**

**df3=spark.sql("select col1,col2,col3 from subscriber where ......... ")**

**#df3.show() ##display selected columns.**

**df4 = spark.sql("select \* from subscriber where final\_status='APPROVED'")**

**#df4.show()**

**df5 = spark.sql("select \* from subscriber where salary<1000")**

**df5.show()**

**--------------------------------------------work on DataFrame------------------------------------------**

**To create a data frame**

**df = spark.read.csv ("C:/Users/Linux/Desktop/RDD/msisdn2.csv",header=True,inferSchema=True)**

**to count data frmae**

**df.count()**

**to distinct any data**

**df.distinct()**

**to show data from a dataframe**

**df.show() , df.collect()**

**to show the number of columns**

**df.head(3)**

**To rename a single dataframe column**

**df = df.withColumnRenamed("first name","first\_name") ##to rename single column, old column then new column.**

**df2=df.withColumnRenamed("final status","final\_status").withColumnRenamed("first name","first\_name").withColumnRenamed("registration type","registration\_type")##To rename multiple column in single step**

**df2=df.toDF(\*(c.replace(' ','\_') for c in df.columns)) ## to rename multiple columns here the columns containing space replacing those space with underscore.**

**df3=df.toDF(\*(x.replace(' ','\_') for x in df.columns))**

**-------------How to summerize a dataframe--------------------**

**df.describe().show()**

**df4.describe('salary').show() ##summerize single columns**

**df4.describe('salary','tax').show() ##summerize multiple columns**

**---------To summerize a single columns or multiple columns.-------------**

**df.describe(‘salary’).show() , df.describe(‘salary’,’tax’,’bonus’).show()**

**select single column or multi columns**

**df4 =df3.select("salary","tax","age") , df4.describe().show()**

**df4.select(“first\_name).show(**

**Selecting Distinct Multiple Columns**

**Way:- first select columns then apply distinct method and then show()**

**to find distinct records from single columns or multiple columns.**

**df.select(‘city).distinct().show() ,,, df.select(‘first\_name’,’city’,’bonus’).distinct().show()**

**spark.sql("select salary from df").distinct().count()**

**To select single column from dataframe or multiple columns**

**df.select('first\_name','city','salary').show()**

**## apply distinct on dataframe to get unique records**

**df4.select('first\_name','city').distinct().count() ###apply distinct on multiple columns**

**df4.select(df4.first\_name).distinct().show() ##apply distinct on a single column**

**sdf.select(sdf.first\_name,sdf.city,sdf.salary+10).show() Note:- here is working all math calculation like +,-,\*,/**

**Filter with a SQL expression. Note the double and single quotes as I’m passing a SQL where clause into filter().**

**df4.filter("final\_status='APPROVED'").show()**

**creating new columns**

**I can create new columns in Spark using .withColumn(). I have yet found a convenient way to create multiple columns at once without chaining multiple .withColumn() methods.**

**df4.withColumn('road\_tax',df4.tax+df4.bonus).show() ##road\_tax is the new column name to be create**

**df4.withColumn('road\_tax',df4.bonus\*df4.tax).show() ##road\_tax is the new column name to be create**

**sdf = df4.select("city",'salary','tax','bonus')**

**total\_income = sdf.withColumn('income',sdf.salary+sdf.tax+sdf.bonus) ##first new column name, then use the column name with action to create new columns.**

**How to drop a column in DataFrame?**

**To drop a column from the DataFrame we can use drop operation. Let’s drop**

**df.drop(‘column\_name’).show() ##drop a column**

**df.drop(df.column\_name’).show() #second method to drop a column**

**Write data on specified location**

**df.write.csv("C:/Users/Linux/Desktop/RDD/out.csv",header=True) ##write output into csv file to another locationwith header**

**df.write.csv("C:/Users/Linux/Desktop/RDD/out.csv") ##write output into csv file to another location without header**

**df.wirte.json(“C:/Users/Linux/Desktop/RDD/out.json”) ##write output into json file to defined location**

**df5.coalesce(1).write.csv("C:/Users/Linux/Desktop/RDD/ottt.csv",header=True) ##writing file into a single file.**

**Simple Filtering Data**

**to filter a particular data on a condition**

**df.filter(df.city==’pune’) ##filter by string value**

**df.filter(df.first\_name=='suraj').count()**

**df.filter(df.salary=='500').show() ##filter by integer value**

**df.filter(df.city=='pune').count() ##count filtered columns**

**df.filter(df.salary>='4000').show() ##filter using comparison operator**

**df.filter(df.salary<=450).show() ##filter using comparison operator**

**df.filter(df.salary!='500').show() ##not equl to condition.**

**To filter data by date columns**

**df4.filter(df4.created\_on<='04-10-2019').show() ##use all aggregate function like >,<,==,!=**

**Filtering Data (Multiple Parameters)**

**We can filter our data based on multiple conditions (AND or OR)**

**df.filter((df.first\_name=='sandeep') & (df.final\_status=='APPROVED')).show() ## Single AND operator**

**df=df3.filter((df3.first\_name=='suraj') & (df3.city=='delhi'))**

**df.write.json("C:/Users/Linux/Desktop/RDD/result.json") ##writing ouput into json file on another location.**

**df.filter((df.first\_name=='rahul') & (df.city=='chennai') & (df.salary=='670')).show() ##multiple AND operator**

**df.filter((df.first\_name=='kamal') & (df.city=='pune')).count() ##single AND operator with value count.**

**## OR operator too**

**df.filter((df.city=='delhi') | (df.salary=='2500')).show() ##OR operator**

**df.filter((df.first\_name=='suraj') | (df.tech=='unix') | (df.salary>='3000')).show() ##multiple OR operator**

**df.filter((df.first\_name=='sunil') | (df.final\_status=='APPROVED') |(df.salary>='5000')).count() ###count value on multiple OR operator**

**df3.filter((df3.first\_name=='suraj')& (df3.city=='delhi')& (df3.salary>='2000')).show()**

**df3.select(df3.first\_name,df3.city,df3.salary).filter("salary==2500 and salary <=3000").show(3) ##important AND operator**

**df3.select(df3.first\_name,df3.salary,df3.age,df3.city).filter("first\_name=='suraj' or salary<2000").show() ##OR operator**

**use of isNull or IsNotNull operator**

**df3.select(df3.first\_name,df3.salary,df3.city,df3.age).filter(df3.salary.isNull()).show()##select and filter columns with IsNull**

**df3.select(df3.first\_name,df3.salary,df3.city,df3.age).filter(df3.salary.isNotNull()).show() ##isNotNull operator**

**USE OF LIKE OPERATOR**

**df3.select(df3.first\_name,df3.city,df3.salary).filter(df3.first\_name.like('s%')).show() ##use of LIKE operator**

**df3.select(df3.first\_name,df3.city,df3.salary,df3.tax).filter(df3.first\_name.like('r%')).show() ##LIKE**

**df3.select(df3.first\_name,df3.city,df3.salary,df3.age).filter("first\_name like '%s%'").show(3) ##use of like operator**

**df3.select(df3.first\_name,df3.salary,df3.city,df3.final\_status,df3.age).filter((df3.salary<=5000)&(df3.city=='delhi')).show(5).filter("first\_name like '%r%'").show(5) ##use multiple filter with LIKE operator**

**df3.select(df3.first\_name,df3.salary,df3.city).filter("first\_name like 'r%'").show(6) ##search starting character**

**df3.select(df3.first\_name,df3.city,df3.salary,df3.age).filter("age in(10,11,12)").filter("first\_name like 's%'").show() ##use of both IN and LIKE operator**

**DataFrame Query: SQL IN clause**

**df3.select(df3.first\_name,df3.salary,df3.age,df3.city).filter(df3.age.isin(10,11,12)).show() ##like IN operator isin is also working same way**

**df3.select(df3.first\_name,df3.salary,df3.age,df3.city).filter(df3.city.isin('pune','delhi')).show() ##use of isin select string values**

**df3.select(df3.first\_name,df3.salary,df3.age).filter("age in(10,11,12,13)").show() ##In clause to filter data**

**df3.select(df3.first\_name,df3.city,df3.salary).filter("salary in(400,450,500)").show(30) ##use IN caluse to filter data**

**df3.select(df3.first\_name,df3.city,df3.msisdn,df3.salary,df3.age).filter("first\_name like '%s%'").filter("age in(10,11,12)").show(4) ##use select,filter and in clause to select data from dataframe.**

**DataFrame Query: SQLNOT IN clause**

**df2.select(df2.first\_name,df2.msisdn,df2.city,df2.salary).filter(df2.salary.isin(2500,4000)==False).show()##filter then NOT IN caluse.**

**df2.select(df2.first\_name,df2.city,df2.salary,df2.age).filter(df2.age.isin(10,11)==False).groupBy('city').count().show() ##filter then NOT IN then grupBy.**

**Sorting Data (OrderBy/sort)**

**To sort the data we use the OrderBy method. By default, it sorts in ascending order, but we can change it to descending order as well.**

**df.orderBy(df.salary).show() ##default order is ascending order**

**df.sort(df.first\_name).show() ##use SORT key also to sorting data, default is ascending**

**df.sort(df.first\_name,ascending=False/True).show() ##descending columnsS**

**df4.orderBy('salary').show(6) ##select number of columns, pass number in show() method in ascending order**

**df4.orderBy('salary',ascending=False/True).show(6) ##select number or columns in descending order.**

**df.orderBy(df.salary,ascending=False/True).show() ##descending columns**

**#df4.orderBy(df4['salary'].desc()).show()**

**df2.groupBy("city").count().sort('count',ascending=False).show() ##sorting by count in descending order**

**What if I want to calculate pair wise frequency of categorical columns?**

**We can use crosstab operation on DataFrame to calculate the pair wise frequency of columns. Let’s apply crosstab operation on ‘Age’ and ‘Gender’ columns of train DataFrame.**

**df3.crosstab('age','salary').show()**

**What If I want to get the DataFrame which won’t have duplicate rows of given DataFrame?**

**We can use dropDuplicates operation to drop the duplicate rows of a DataFrame and get the DataFrame which won’t have duplicate rows. To demonstrate that I am performing this on two columns Age and Gender of train and get the all unique rows for these columns.**

**df4.select(df4.first\_name).dropDuplicates().show()**

**df3.select(df3.first\_name,df3.salary,df3.city).dropDuplicates().show()**

**Note:-dropDuplicates() functions will use on one columns or multiple columns, if it will apply on one columns then give the unique records, or if apply on multiple columns then dropDuplicates() functions will apply on all columns data and result will get unique data in every columns of data**

**What if I want to drop the all rows with null value?**

**The dropna operation can be use here. To drop row from the DataFrame it consider three options.**

**how– ‘any’ or ‘all’. If ‘any’, drop a row if it contains any nulls. If ‘all’, drop a row only if all its values are null.**

**thresh – int, default None If specified, drop rows that have less than thresh non-null values. This overwrites the how parameter.**

**subset – optional list of column names to consider.**

**Let’t drop null rows in train with default parameters and count the rows in output DataFrame. Default options are any, None, None for how, thresh, subset respectively.**

**df4.dropna().count()**

**What if I want to fill the null values in DataFrame with constant number?**

**Use fillna operation here. The fillna will take two parameters to fill the null values.**

**value:**

**It will take a dictionary to specify which column will replace with which value.**

**A value (int , float, string) for all columns.**

**subset: Specify some selected columns.**

**df.fillna(‘a’).show()##fill the columns with any values on the place of ‘null’ values**

**How to find the mean of each age group in train?**

**The groupby operation can be used here to find the mean of Purchase for each age group in train. Let’s see how can we get the mean purchase for the ‘Age’ column train.**

**df6.groupBy('city').agg({'salary':'max'}).show()# agg() ##function used always with groupBy clause.**

**df6.groupBy('first\_name').agg({'salary':'avg'}).show() ## you can use avg or mean for average salary**

**df4.groupBy('city').agg({'salary':'std'}).show() ##calculate std**

**df4.groupBy('city').agg({'salary':'min'}).show() ##group by min records**

**df3.select(df3.first\_name,df3.city,df3.salary,df3.age).groupBy('city').count().show()##use of simple select and group By clause**

**df3.select(df3.first\_name,df3.salary,df3.city,df3.age).groupBy('city').count().filter("count==5").show() ##use groupBy clause with filter**

**-------important and complex DataFrame query for practice……………………**

**df4.select(df4.first\_name,df4.city,df4.salary,df4.tax).filter(df4.first\_name=='suraj').show() ##query with selection and filter records**

**df4.select(df4.first\_name,df4.msisdn,df4.city,df4.salary).groupBy('city').max().show() ##selection with group By clouse**

**df4.select(df4.first\_name,df4.city,df4.salary,df4.tax).filter((df4.city=='delhi')&(df4.salary>=5000)).show() ##select columns and filter them, AND operator**

**df4.select(df4.first\_name,df4.city,df4.salary,df4.tax).filter((df4.salary<=5000)|(df4.city=='delhi')).orderBy(df4.tax,ascending=False).show() ##use OR operator for filter**

**df4.select(df4.first\_name,df4.city,df4.salary,df4.tax,df4.age).filter(df4.age<=20).count() ##select columns then filter and count the data**

**df4.select(df4.first\_name,df4.salary,df4.city,df4.age).filter(df4.age>=10).orderBy(df4.age,ascending=False).show() ##select columns then filter then use order By for sorting the data**

**df4.select(df4.first\_name,df4.salary,df4.city,df4.age).filter(df4.age>=10).orderBy(df4.age,ascending=False).count() ##select,filter,order by and count records**

**df4.select(df4.first\_name,df4.city,df4.salary).filter((df4.city=='delhi')&(df4.salary>=5000)).dropDuplicates().show() ##select columns then filter then drop Duplicates records**

**df5.select(df5.first\_name,df5.salary,df5.city,df5.salary+100).filter((df5.salary==2500)|(df5.city=='delhi')).show() ##how to use multiple OR operator in select query**

**df5.select(df5.first\_name,df5.msisdn,df5.city,df5.salary,df5.age).filter((df5.city=='delhi')&(df5.salary=='2500')&(df5.age=='10')).show() ##use multiple AND operator.**

**df3.select(df3.first\_name,df3.salary,df3.tax,df3.age).groupBy('first\_name').count().filter("count >5").show()##use of select with gorupBy with count.**

**df2.select(df2.first\_name,df2.salary,df2.city).filter((df2.city.isin('pune','delhi','noida'))).groupBy('city').count().show() ##select then use IN with filter then GroupBy columns**

**jn=df5.join(data3,df5.subscriber==data3.subscriber,how='inner').select(df5.first\_name,data3.msisdn,df5.salary,data3.city).filter((df5.salary<5000)|(df5.city=='delhi')).orderBy(df5.salary,ascending=False).show() ##use join to join two table then select columns then apply filter to filter data then apply orderBy to sort the data then show**

**df2.select(df2.first\_name,df2.salary,df2.age,df2.tech,df2.tax,df2.city).filter(df2.city.like('n%')).groupBy('city').count().show() ##select,filter with like operator,thengroupBy then count**

**inner\_join=data3.join(df3,data3.subscriber==df3.subscriber).select(df3.first\_name,df3.salary,data3.city,data3.company).groupBy('city').count().show() ##join two table and use groupBycaluse with filter**

**jn =df3.join(data3,df3.subscriber==data3.subscriber,how='inner').select(df3.first\_name,df3.salary,data3.company,data3.city).groupBy('city').agg({'salary':'max'}).show()##join tables with group by caluse and aggregate functions**

**df2.select(df2.first\_name,df2.city,df2.salary,df2.age).groupBy('city').count().filter(df2.city.isNull()).show() ##select columns then groupBy then count then filter then isNull used.**

**in\_join=df2.join(data2,df2.subscriber==data2.subscriber).join(sales2,df2.subscriber==sales2.subscriber).select(df2.first\_name,data2.age,sales2.growth).show() ##multiple table joins in dataframe**

**f\_join=df2.join(dt,df2.subscriber==dt.subscriber,how='full').join(sa,df2.subscriber==sa.subscriber,how='full').select(df2.first\_name,dt.city,sa.company,sa.growth,dt.salary,df2.age).filter(df2.first\_name.isNotNull()).filter(dt.city.isin('delhi','pune','noida')).show() ##multiple tables join with alias and with filter**

**-------------Pyspark Join Example-------------------**

**An inner join is the default join type used.**

**inner\_join=df4.join(data4,df4.subscriber==data4.subscriber,how='inner')**

**inner\_join.show(2)##join two tables using inner join**

**inner\_join=df4.join(data4,df4.subscriber==data4.subscriber,how='inner').select(df4.first\_name,df4.city,df4.salary,data4.company)##join tables and select few columns from both tables.**

**inner\_join.show(3)**

**inner\_join = df4.join(data4,df4.subscriber==data4.subscriber,how='inner').select(df4.first\_name,df4.msisdn,df4.salary,df4.age,data4.company, df4.tech,df4.city).filter(df4.city=='delhi') ##join both tables and select few columns from both tables and apply filter condition.**

**inner\_join.show()**

**inner\_join = df4.join(data4,df4.subscriber==data4.subscriber,how='full').select(df4.first\_name,data4.salary,data4.company,df4.city).filter(df4.city=='noida').orderBy(df4.salary,ascending=False) ##join two tables and select particular columns then filter the result and order the result on behalf of any columns**

**inner\_join.show(100)**

**inner\_join=df3.join(data3,df3.subscriber==data3.subscriber).select(df3.first\_name,df3.salary,data3.city).groupBy('city').agg({'salary':'max'}) ##join the columns and then groupBy data then after apply group base aggregate functions.**

**inner\_join.show()**

**Pyspark Left Join Example**

**FROM table is the left-hand side in the join.**

**left\_join=df4.join(data4,df4.subscriber==data4.subscriber,how='left').select(df4.first\_name,df4.salary,data4.company)**

**left\_join.show()**

**Pyspark Right Join Example**

**FROM table is the right-hand side in the join.**

**right\_join=df4.join(data4,df4.subscriber==data4.subscriber,how='right').select(df4.first\_name,df4.salary,data4.company)**

**right\_join.show()**

**Pyspark Full Outer Join Example**

**Finally, we get to the full outer join. This shows all records from the left table and all the records from the right table and nulls where the two do not match.**

**full\_join=df4.join(data4,df4.subscriber==data4.subscriber,how='full').select(df4.first\_name,df4.salary,data4.company) ##join both table and get matched data un-mattached data**

**full\_join.show()**

**Aggregating Data**

**Spark allows us to perform powerful aggregate functions on our data, similar to what you're probably already used to in either SQL or Pandas**

**We're going to become familiar with two functions here: agg() and groupBy(). These are typically used in tandem, but agg() can be used on a dataset without groupBy():**

**Dataframe Aggregation**

**A set of methods for aggregations on a DataFrame:**

**agg**

**avg**

**count**

**max**

**mean**

**min**

**pivot**

**sum**

**Grouping the Data**

**GroupBy is used to group the dataframe based on the column specified**

**df.groupBy(sdf.city).count().show() ##*Count city***

**df3.groupBy("salary").count().show() ##first count then show will display number of group**

**df4.groupBy('salary').max().show() ##max value in every group**

**df4.groupBy('salary').min().show() ##min value in every group**

**df4.groupBy('salary').mean().show() ##average of every group**

**df4.groupBy('salary').sum().show() ##sum of every group**

**2nd method for work**

**gdf = df4.groupBy("city")**

**gdf.max().show()**

**sdf = df4.select('city','salary','tax')**

**#gdf = sdf.groupBy('city').sum().show()**

**gdf = sdf.groupBy('city')**

**gdf.sum().show()**

**sdf = df4.select('city','salary','bonus','tax')**

**gdf = sdf.groupBy('city')**

**gdf.avg().show()**

**sdf = df4.select('city','salary','bonus','tax')**

**sdf.groupBy('city').min().show()**

**df3.agg({'\*':'count'}).show() ##Count the number of rows**

**df3.count() ##count the number of rows**

**df3.agg({"salary":"count"}).show() ##count any single column values**

**df3.agg({'salary':'sum'}).show() ###sum the given column values**

**df3.agg({'age':'max'}).show() ##get the max value from given dataset**

**df3.agg({'salary':'min'}).show() ##get the min value from given dataset**

**df3.agg({'salary':'mean'}).show() ##get the average value from given dataset**

**aggregating data with group By clause**

**df3.groupBy('first\_name').agg({'salary':'sum'}).show()##groupBy with agg ()**

**df3.groupBy('city').agg({'salary':'sum'}).orderBy('city',ascending=False).show() ##groupBy with agg() with orderBy ()**

**df3.groupBy('final\_status').agg({'city':'min'}).show()##groupBy with agg () use on string value data also**

**Use aggregate with group by functions**

**df5.select(df5.city,df5.salary,df5.bonus,df5.tax).groupBy('city').sum().show() ##sum all columns data**

**df5.select(df5.first\_name,df5.salary,df5.bonus,df5.tax).groupBy('first\_name').mean().show() ##mean of every columns with groupBy**

**-----------------DataFrame String Functions------------------------------**

We've already seen a few **String** functions such as **split()**, **format\_string()**, **upper()** and **lower()** from the previous examples. In this section, I thought of presenting some of the additional built-in functions that Spark provides when you have to work with **textual** data points. As a matter of fact, these can be handy if you have a need to **normalise** and run **feature extractions** over textual datasets in a **Machine Learning** pipeline.

We will enrich our dataframe with the following columns:

* Contains plain: This column uses the **instr()** function and outputs the index of the given substring in a column.
* Length: This column uses the **length()** function and outputs the lengths for the string values in a column.
* Trim: This column uses the **trim()** function and removes spaces on both sides of the text in a column.
* LTrim: This column uses the **ltrim()** function and removes spaces from the left side of the text in a column.
* RTrim: This column uses the **rtrim()** function and removes spaces from the right side of the text in a column.
* Reverse: This column uses the **reverse()** function and outputs the text in reverse order in a column.
* Substring: This column uses the **substring()** function and outputs the text in a column for the given from and to character indices.
* IsNull: This column uses the **isnull()** function and outputs true or false if the text in a column is null or not.
* Concat: This column uses the **concat\_ws()** function and outputs a String representation of columns being concatenated. Note also that the **concat\_ws()** function also allows you to provide a given textual separator.
* InitCap: This column uses the **initcap()** function and converts the first letter for each word in the column into uppercase.