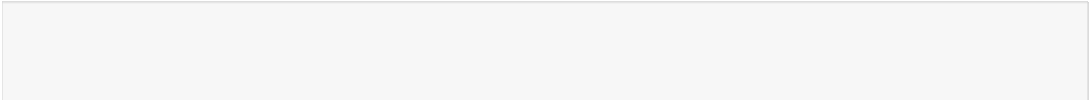
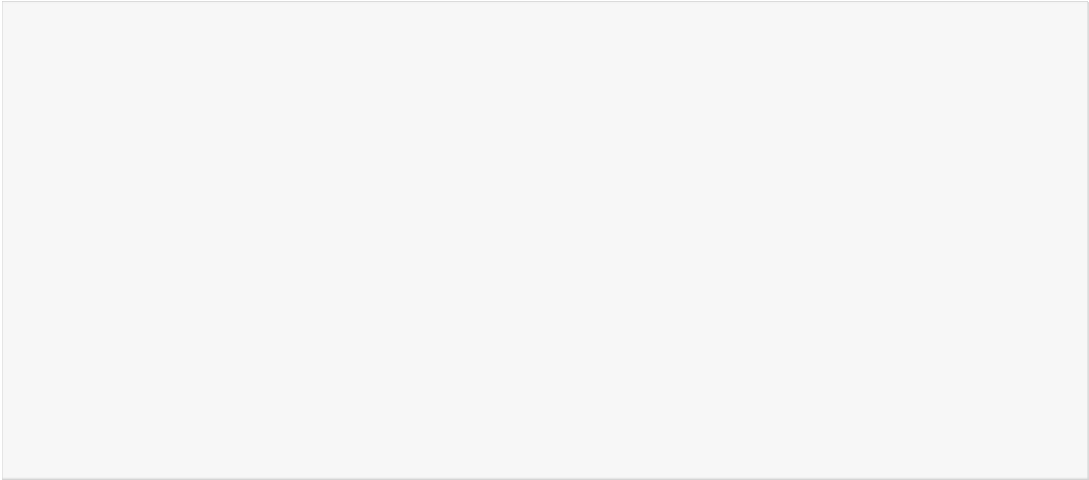
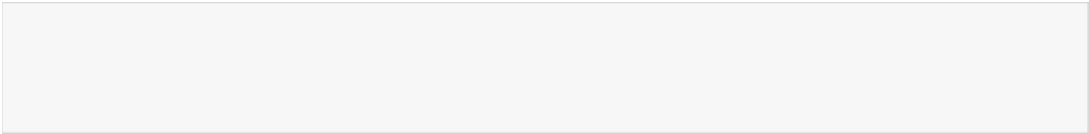
In [0]:



from pyspark.sql.functions import \* from pyspark.sql.types import \* from pyspark.sql.window import \* from collections import defaultdict from datetime import date

In [0]:

# # File location and type

file\_location = "/FileStore/tables/Zomato\_reduced.csv" file\_type = "csv"

# # CSV options

infer\_schema = "false" first\_row\_is\_header = "true" delimiter = ","

# # The applied options are for CSV files. For other file types, these will be ignored.

df = spark.read.format(file\_type) \

.option("inferSchema", infer\_schema) \

.option("header", first\_row\_is\_header) \

.option("sep", delimiter) \

.option("multiline","true") \

.option("quote", "\"")\

.option("escape", "\"")\

.load(file\_location) display(df)

In [0]:

df.count()

Out[6]: 1159

Requirement 1

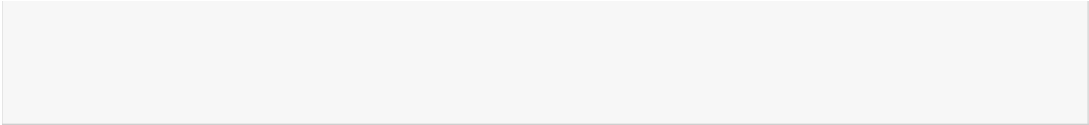
## 1)Remove all non-ascii characters from all columns : Name, Location etc

In [0]:

# ## creating udf

def ascii\_ignore(x): if x is None:



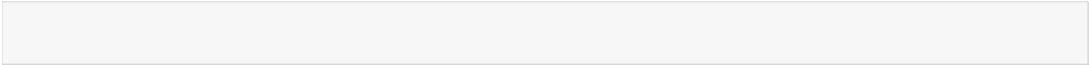


return x else:

return x.encode('ascii', 'ignore').decode('ascii') ascii\_udf = udf(ascii\_ignore)

# df1.withColumn('name',ascii\_udf('name')).filter("name != ''").show()

In [0]:



## getting all the column names

df.columns

Out[9]: ['url', 'address', 'name', 'online\_order', 'book\_table', 'rate',

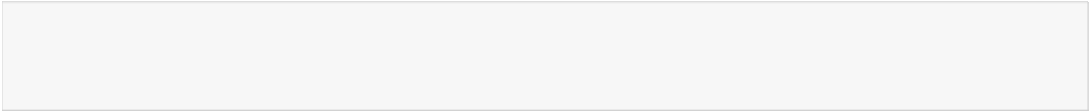
'votes',

'phone', 'location', 'rest\_type', 'dish\_liked', 'cuisines',

'approx\_cost(for two people)', 'reviews\_list',

'menu\_item', 'listed\_in(type)', 'listed\_in(city)', 'liked\_food\_from\_review', 'menus\_combined', 'location\_latitude', 'location\_longitude']

In [0]:



for i in df.columns: print(i)

df=df.withColumn(i,ascii\_udf(i)) display(df)

url address name

online\_order book\_table rate

votes phone location rest\_type dish\_liked cuisines

approx\_cost(for two people) reviews\_list

menu\_item listed\_in(type) listed\_in(city) liked\_food\_from\_review menus\_combined location\_latitude location\_longitude







**req 2 , Remove restaurants with no ratings.**

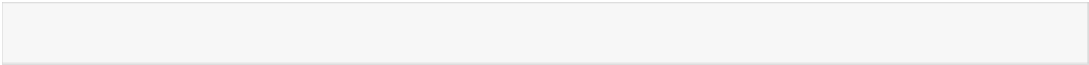
In [0]:



df=df.na.drop(subset=["rate"])

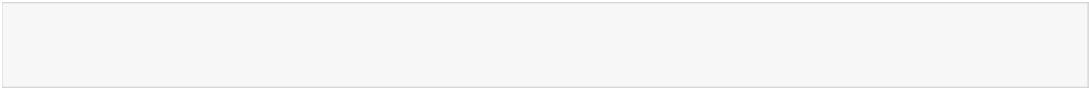
**ETL , gonna cache this dataset as it may be used multiple times**

In [0]:



df.cache() df.createOrReplaceTempView("table")

In [0]:



df\_inval\_url=spark.sql("select \* from table where url not rlike '^(http|https)\\://\*'") #just checking http or https:// is present or not. pl form the complex regex to validate complicated url pattern

In [0]:



df\_inval\_url.count()

Out[85]: 0

In [0]:



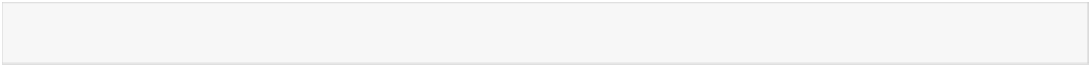
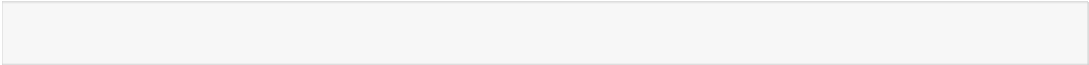
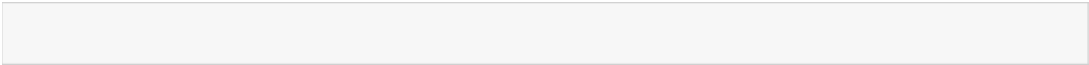
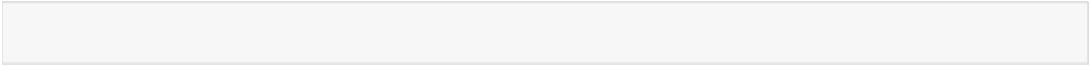
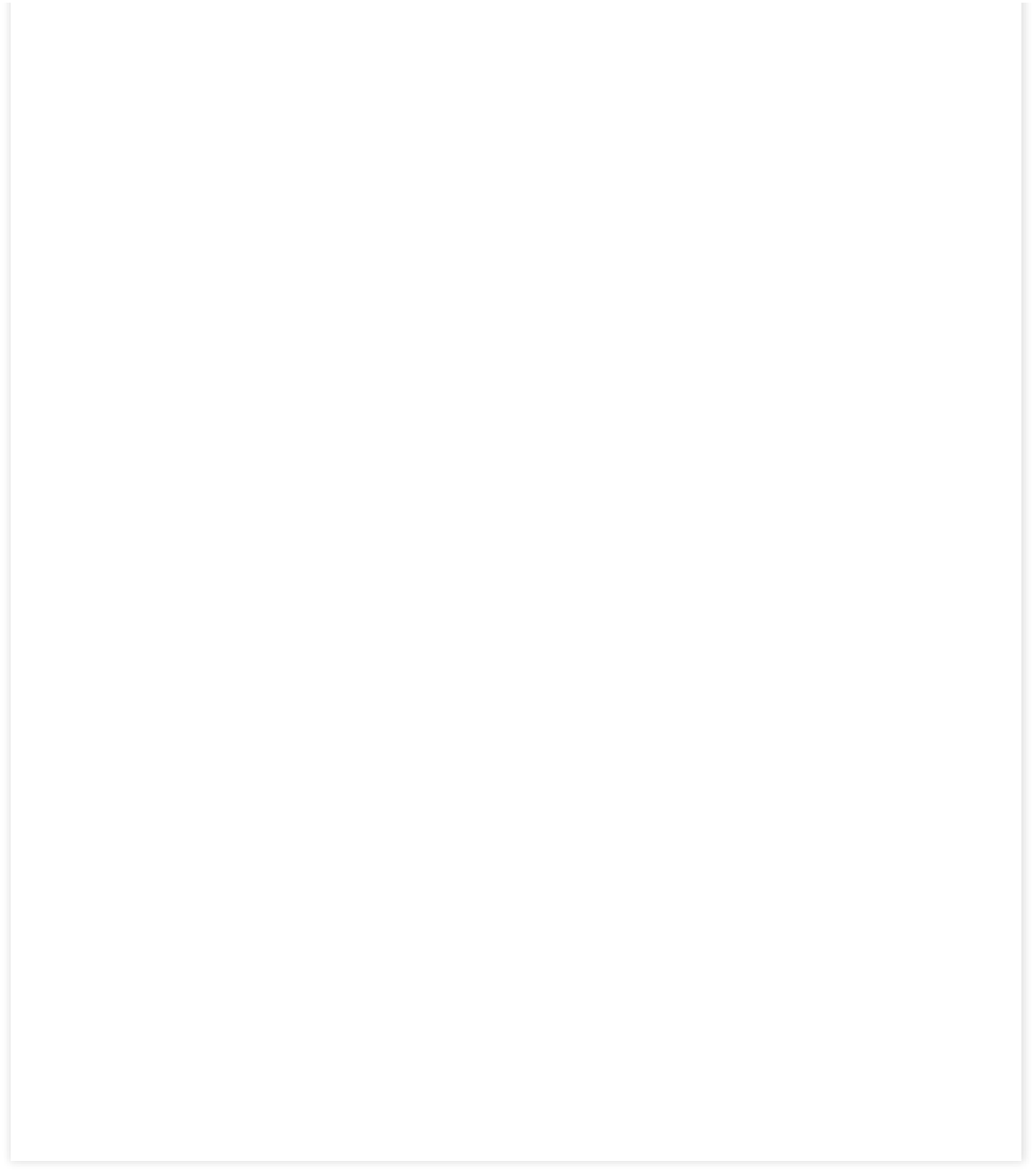
display(df\_val\_url)



In [0]:



# so all urls in dataset are valid



## for q2 we need closed res

**as all urls are valid, so we wont get any closed res in dataset. however for logic purpose lets take the 1st 100 rows from our dataset and assume those are closed**

In [0]:

df\_closed=spark.sql("select \* from table limit 100") df\_closed.count()

Out[90]: 100

In [0]:

df\_closed.createOrReplaceTempView("closed\_table")

In [0]:

spark.sql("select location,count(\*) as closed\_res from closed\_table group by location").s how()

+ + +

location|closed\_res|

+ + +

|  |  |
| --- | --- |
| Banashankari| | 78| |
| Basavanagudi| | 10| |
| Mysore Road| | 1| |
| Jayanagar| | 4| |
| Kumaraswamy Layout| | 4| |
| Rajarajeshwari Nagar| | 1| |
| Vijay Nagar| | 2| |

+ + +

## pl try out the other ones. just form the sql and place it inside spark.sql("") to save any result to parquet, do the following

In [0]:

df\_to\_be\_stored=spark.sql("select location,count(\*) as closed\_res from closed\_table group by location")

In [0]:

df\_to\_be\_stored.coalesce(1).write.mode("overwrite").format("parquet").save("FileStore/par uqet/")