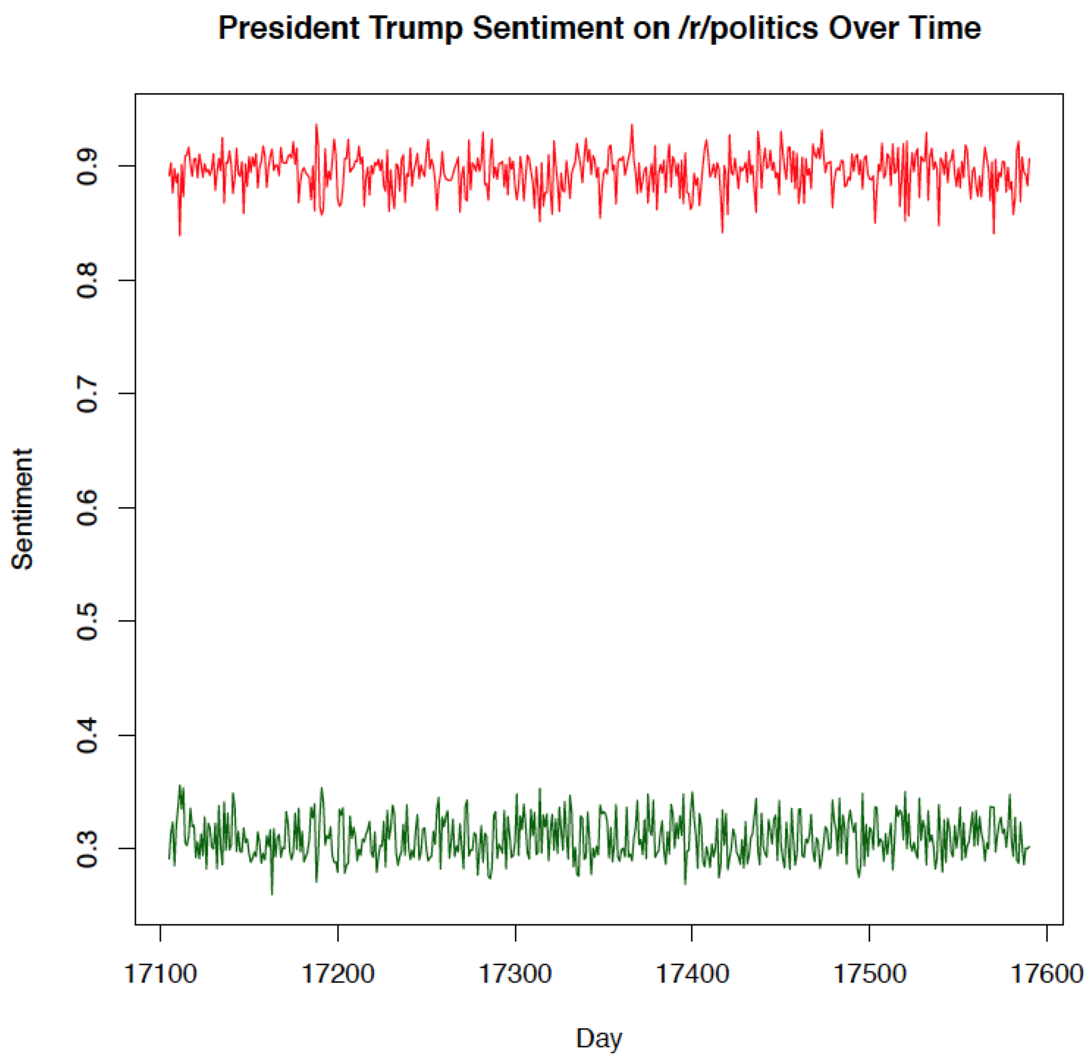
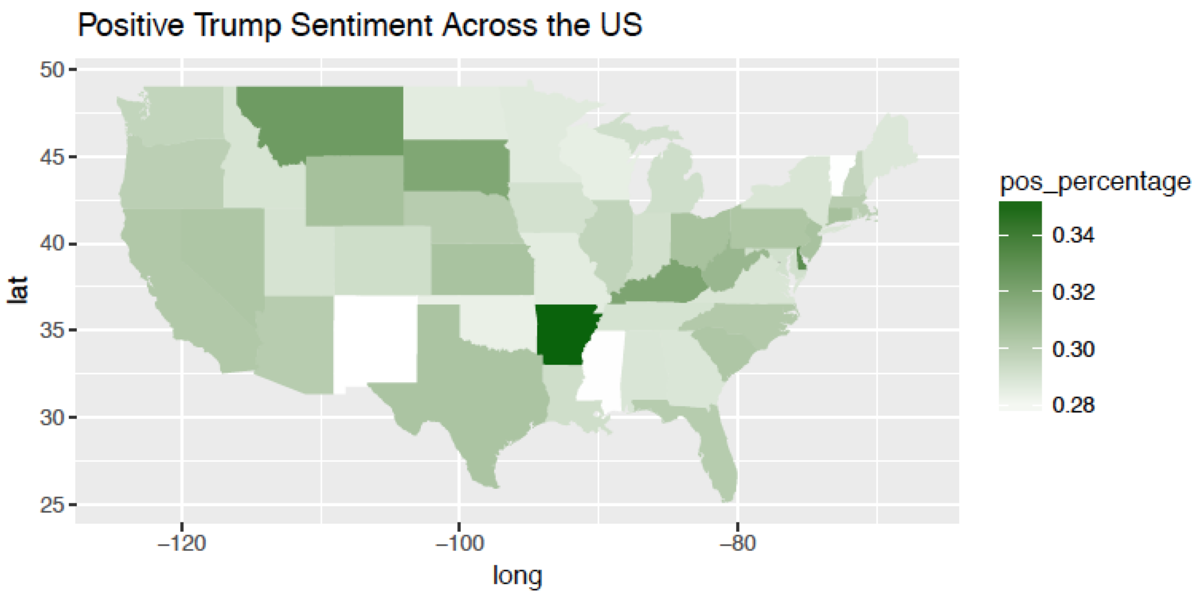
Final Deliverable

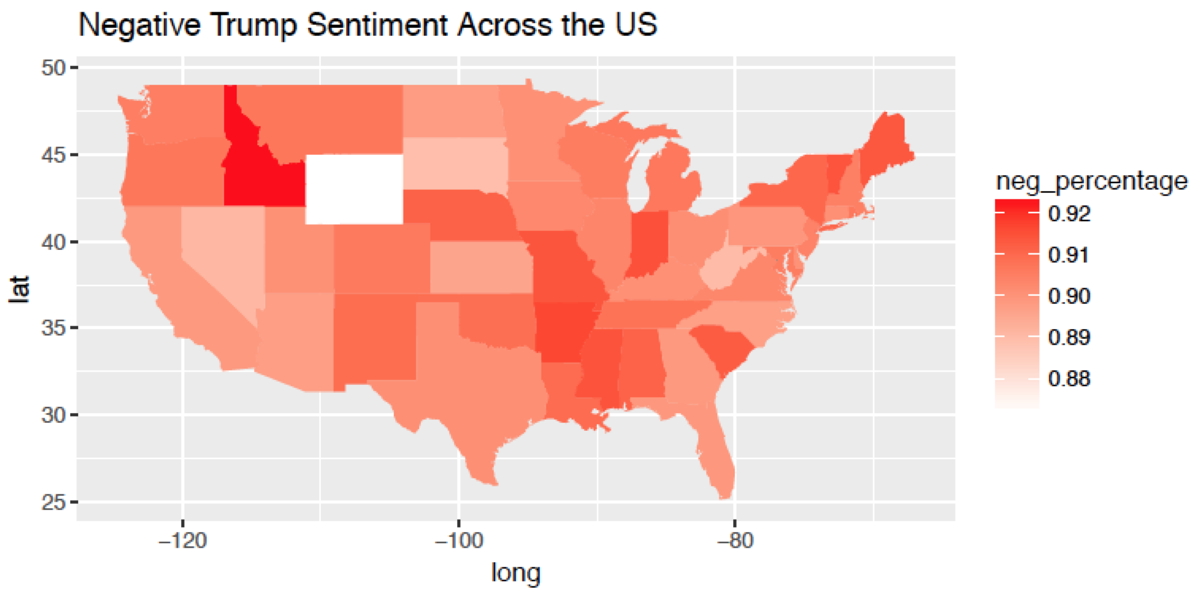
1. Create a time series plot (by day) of positive and negative sentiment. This plot should contain two lines, one for positive and one for negative. It must have data as an X axis and the percentage of comments classified as each sentiment on the Y axis.



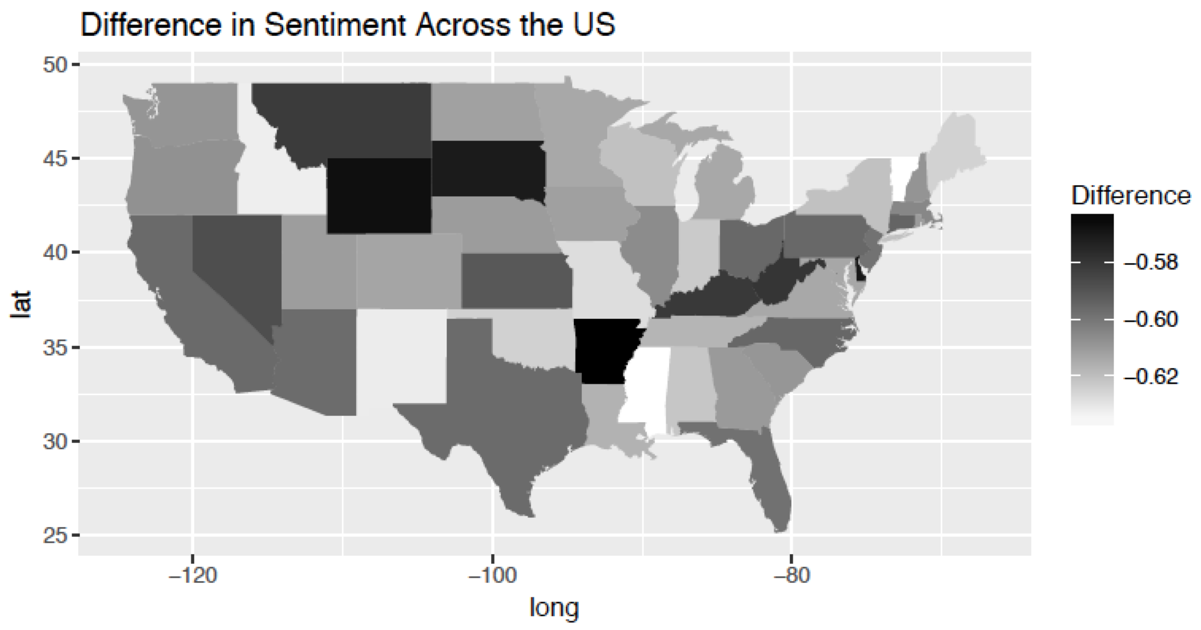
*Green lines represent Positive, Red lines represent Negative

2. Create 2 maps of the United States: one for positive sentiment and one for negative sentiment. Color the states by the percentage.

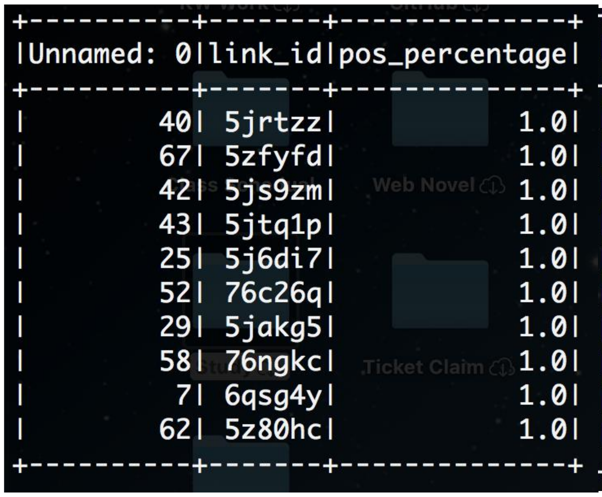




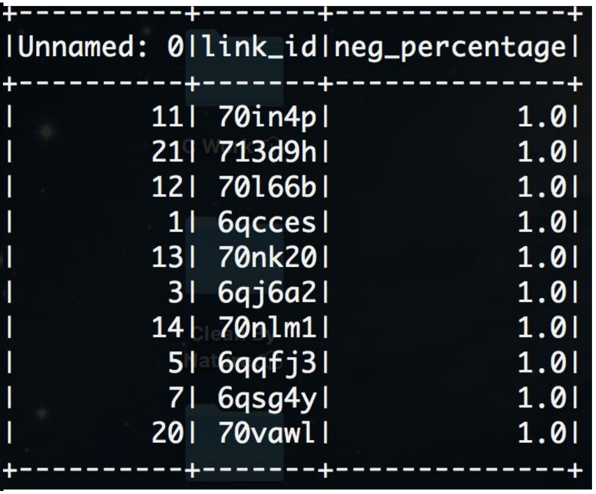
3. Create a third map of the United States that computes the *difference:* %Positive - %Negative.



4. Give a list of the top 10 positive stories (have the highest percentage of positive comments) and the top 10 negative stories (have the highest percentage of negative comments).

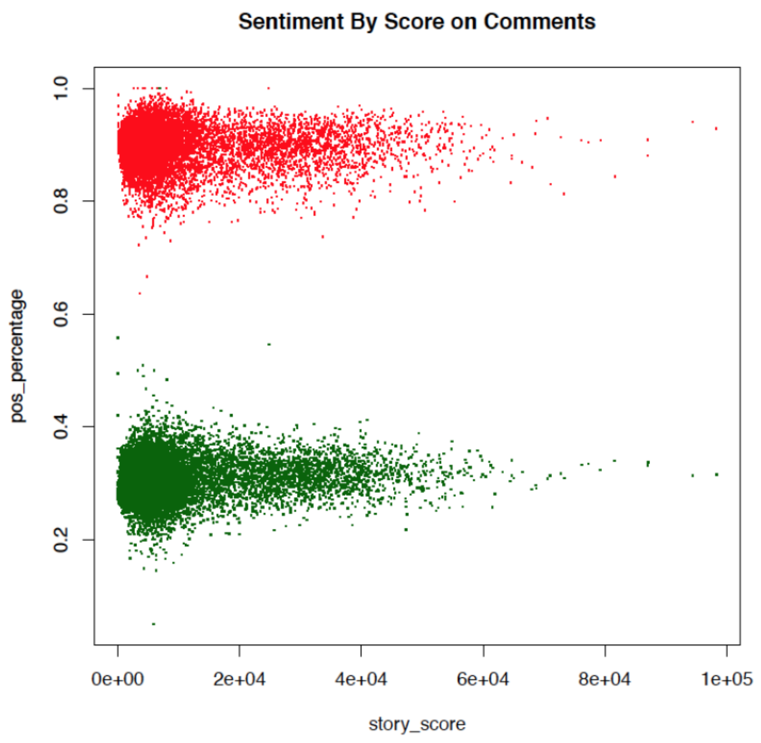


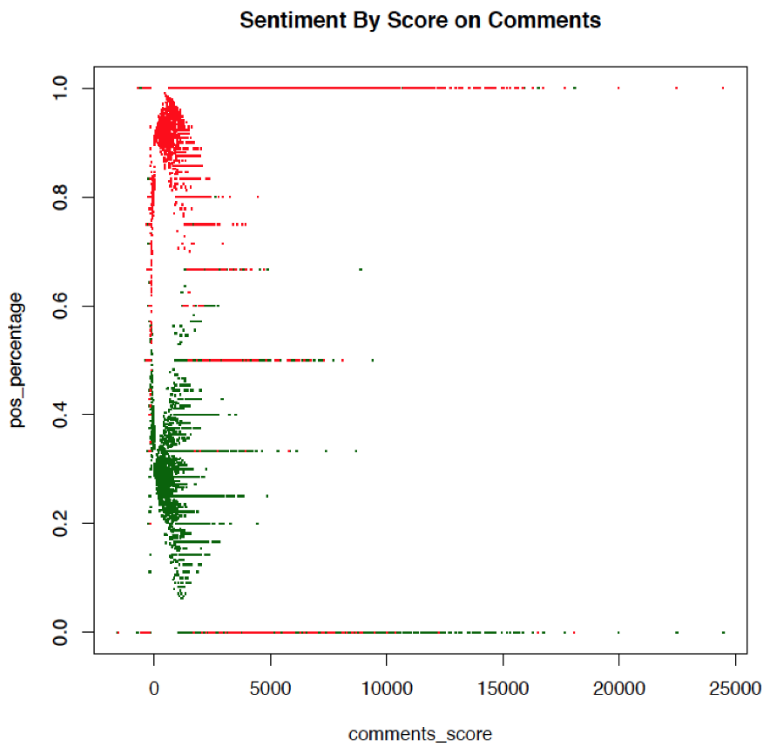
```
+----------+-------+--------------+
|Unnamed: 0|link_id|pos_percentage|
+----------+-------+--------------+
|        40| 5jrtzz|           1.0|
|        67| 5zfyfd|           1.0|
|        42| 5js9zm|           1.0|
|        43| 5jtq1p|           1.0|
|        25| 5j6di7|           1.0|
|        52| 76c26q|           1.0|
|        29| 5jakg5|           1.0|
|        58| 76ngkc|           1.0|
|         7| 6qsg4y|           1.0|
|        62| 5z80hc|           1.0|
+----------+-------+--------------+
```



```
+----------+-------+--------------+
|Unnamed: 0|link_id|neg_percentage|
+----------+-------+--------------+
|        11| 70in4p|           1.0|
|        21| 713d9h|           1.0|
|        12| 70l66b|           1.0|
|         1| 6qcces|           1.0|
|        13| 70nk20|           1.0|
|         3| 6qj6a2|           1.0|
|        14| 70nlm1|           1.0|
|         5| 6qqfj3|           1.0|
|         7| 6qsg4y|           1.0|
|        20| 70vawl|           1.0|
+----------+-------+--------------+
```

5. Create TWO scatterplots where the X axis is the submission score, and a second where the X axis is the comment score, and the Y access is the percentage positive and negative. Use two different colors for positive and negative. **This allows us to determine if submission score, or comment score can be used as a feature.**





Summary:

The result we get shows that most of the /r/politics sentiments are negative. By the result from data, we believe that most of the /r/politics think about President Trump in negative attitude.

The data shows that the result is not vary over time or submissions, but is vary on states. Negatives are much more than Positives among all states, but there are still slightly difference between states.

Questions

QUESTION 1: Take a look at `labeled_data.csv`. Write the functional dependencies implied by the data.

The functional dependencies in labeled_data.csv is that Input_id → labeldem, labelgop, labeldjt. Input_id is the key here and for each input_id, there is only one labeldem, labelgop and labeldjt associated with it.

QUESTION 2: Take a look at the schema for the comments dataframe. Forget BCNF and 3NF. Does the data frame *look* normalized? In other words, is the data frame free of redundancies that might affect insert/update integrity? If not, how would we decompose it? Why do you believe the collector of the data stored it in this way?

It doesn't look like normalized. There is irrelevant functional dependencies: author_flair_text → author_flair_css_class. To decompose it, we create another form contains only the two attributes and remove one from the original data frame. The data collector may collect it for completeness.

```
root
 |-- author: string (nullable = true)
 |-- author_cakeday: boolean (nullable = true)
 |-- author_flair_css_class: string (nullable = true)
 |-- author_flair_text: string (nullable = true)
 |-- body: string (nullable = true)
 |-- can_gild: boolean (nullable = true)
 |-- can_mod_post: boolean (nullable = true)
 |-- collapsed: boolean (nullable = true)
 |-- collapsed_reason: string (nullable = true)
 |-- controversiality: long (nullable = true)
 |-- created_utc: long (nullable = true)
 |-- distinguished: string (nullable = true)
 |-- edited: string (nullable = true)
 |-- gilded: long (nullable = true)
 |-- id: string (nullable = true)
 |-- is_submitter: boolean (nullable = true)
 |-- link_id: string (nullable = true)
 |-- parent_id: string (nullable = true)
 |-- permalink: string (nullable = true)
 |-- retrieved_on: long (nullable = true)
 |-- score: long (nullable = true)
 |-- stickied: boolean (nullable = true)
 |-- subreddit: string (nullable = true)
 |-- subreddit_id: string (nullable = true)
 |-- subreddit_type: string (nullable = true)
```

QUESTION 3: Pick one of the joins that you executed for this project. Rerun the join with .explain() attached to it. Include the output. What do you notice? Explain what Spark SQL is doing during the join. Which join algorithm does Spark seem to be using?

output:

```
== Physical Plan ==
*(7) SortMergeJoin [id#0], [id#186], Inner
:- *(3) Sort [id#0 ASC NULLS FIRST], false, 0
:   +- Exchange hashpartitioning(id#0, 200)
:      +- *(2) Project [id#0, link_id#1, body#2, created_utc#3L, author_flair_text#4, title#5,
comments_score#6L, story_score#7L, features#8, UDF(features#8) AS rawPrediction#62,
UDF(UDF(features#8)) AS probability#73, UDF(UDF(features#8)) AS prediction#85,
pythonUDF0#299 AS pos#158]
:         +- BatchEvalPython [<lambda>(UDF(UDF(features#8)))], [id#0, link_id#1, body#2,
created_utc#3L, author_flair_text#4, title#5, comments_score#6L, story_score#7L, features#8,
pythonUDF0#299]
:            +- *(1) Filter isnotnull(id#0)
:               +- *(1) FileScan parquet
[id#0,link_id#1,body#2,created_utc#3L,author_flair_text#4,title#5,comments_score#6L,story_sc
ore#7L,features#8] Batched: false, Format: Parquet, Location:
InMemoryFileIndex[file:/home/cs143/www/comments_unseen.parquet], PartitionFilters: [],
PushedFilters: [IsNotNull(id)], ReadSchema:
struct<id:string,link_id:string,body:array<string>,created_utc:bigint,author_flair_text:string,ti...
+- *(6) Sort [id#186 ASC NULLS FIRST], false, 0
   +- Exchange hashpartitioning(id#186, 200)
      +- *(5) Project [id#186, link_id#187, body#188, created_utc#189L, author_flair_text#190,
title#191, comments_score#192L, story_score#193L, features#194, UDF(features#194) AS
rawPrediction#110, UDF(UDF(features#194)) AS probability#121, UDF(UDF(features#194)) AS
prediction#133, pythonUDF0#300 AS neg#172]
```

```
      +- BatchEvalPython [<lambda>(UDF(UDF(features#194)))], [id#186, link_id#187,
body#188, created_utc#189L, author_flair_text#190, title#191, comments_score#192L,
story_score#193L, features#194, pythonUDF0#300]
        +- *(4) Filter isnotnull(id#186)
          +- *(4) FileScan parquet
[id#186,link_id#187,body#188,created_utc#189L,author_flair_text#190,title#191,comments_sco
re#192L,story_score#193L,features#194] Batched: false, Format: Parquet, Location:
InMemoryFileIndex[file:/home/cs143/www/comments_unseen.parquet], PartitionFilters: [],
PushedFilters: [IsNotNull(id)], ReadSchema:
struct<id:string,link_id:string,body:array<string>,created_utc:bigint,author_flair_text:string,ti...
```

Answer:

We have noticed that Spark SQL are passing in the UDF we defined in format: ‘UDF(UDF(features#8))’, and it also put extra label in every original label, probably use to distinguish labels have same name in different DataFrame. Spark SQL also calling some python method to help the join progress. The last thing we notice is that Spark SQL using parquet format during the join process, but not DataFrame type, probably for speeding up the progress.

In the join progrees, Spark SQL first scan two DataFrames as parquet file, and filter out the Null value of key. Next, it apply the UDFs to the original two Dataframes in “BatchEvalPython” part. Then it execute the ‘SELECT’, project out the attributes we choose as a new table(probably temporary), and hashes the key ‘id’ in ‘Exchange hashpartitioning(id#0, 200)’ step. Finally, Spark SQL sort two Dataframes by key, and Inner join two Dataframes.

In the return plan of explain(), it says ‘SortMergeJoin [id#0], [id#186], Inner’, but the implementation of it is a combination of Hash Join and Merge Join, it seems Spark SQL is using techniques of both Hash Join and Sorted Merge Join.