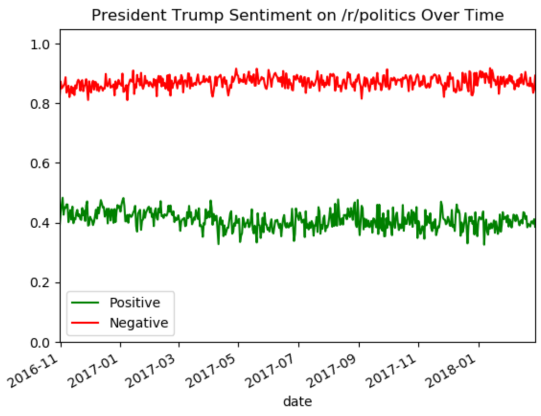
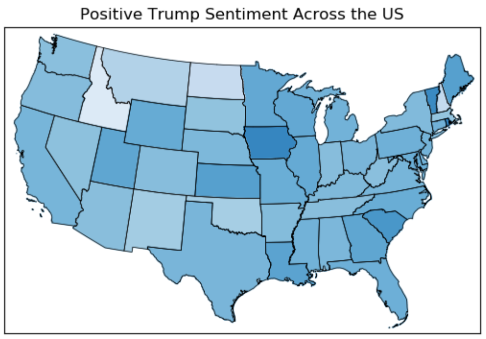
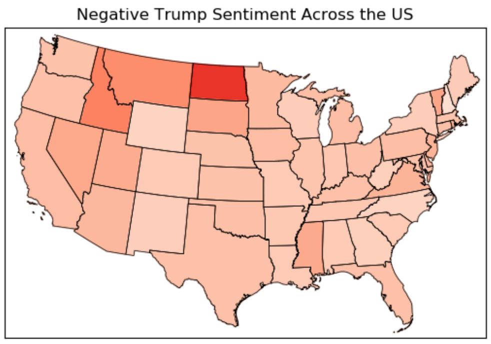
Michael Wang
Project 2 Report

1)

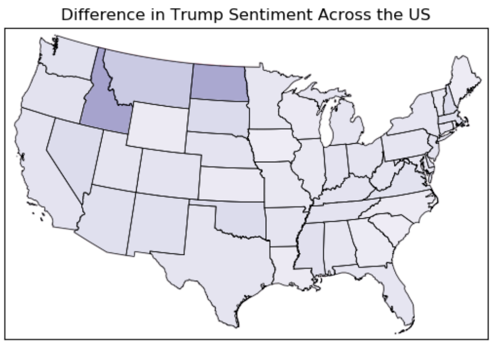


2)





3)



4)
(As I only used a small subset of the data, the following top ten lists are incredibly skewed. SID refers to the ID of the submission)

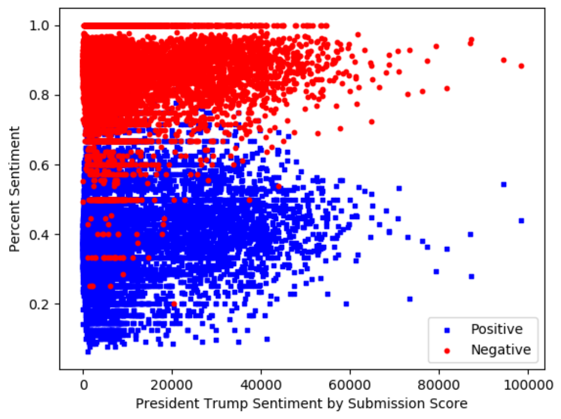
Top 10 Most Negative:

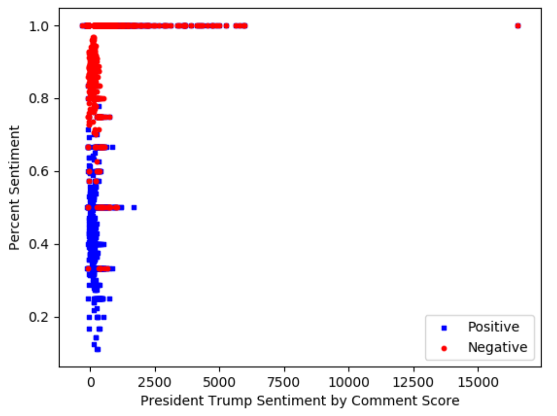
```
In [147]: ten_pos.show(10)
+------+-----+--------+--------+----+----+
|   sid|count|posCount|negCount|pos%|neg%|
+------+-----+--------+--------+----+----+
|5b1263|    1|       1|       1| 1.0| 1.0|
|5bm90b|    1|       1|       1| 1.0| 1.0|
|5c250m|    1|       1|       1| 1.0| 1.0|
|5dpe5m|    1|       1|       1| 1.0| 1.0|
|5ealhh|    1|       1|       1| 1.0| 1.0|
|5fg0k7|    2|       2|       2| 1.0| 1.0|
|5gxm6p|    1|       1|       1| 1.0| 1.0|
|5i4avy|    1|       1|       1| 1.0| 1.0|
|5i6j79|    2|       2|       1| 1.0| 0.5|
|5ijm3c|    2|       2|       2| 1.0| 1.0|
+------+-----+--------+--------+----+----+
```

Top 10 Most Negative:

```
In [148]: ten_neg.show(10)
+------+-----+--------+--------+-------------------+----+
|   sid|count|posCount|negCount|               pos%|neg%|
+------+-----+--------+--------+-------------------+----+
|5dfdqt|    2|       1|       2|                0.5| 1.0|
|5gxm6p|    1|       1|       1|                1.0| 1.0|
|5dpe5m|    1|       1|       1|                1.0| 1.0|
|5b1263|    1|       1|       1|                1.0| 1.0|
|5e2bqq|    2|       1|       2|                0.5| 1.0|
|5b3xps|    2|       1|       2|                0.5| 1.0|
|5ealhh|    1|       1|       1|                1.0| 1.0|
|5bm90b|    1|       1|       1|                1.0| 1.0|
|5eap2e|   17|       3|      17|0.17647058823529413| 1.0|
|5c250m|    1|       1|       1|                1.0| 1.0|
+------+-----+--------+--------+-------------------+----+
```

5)





6)
The most telling statistic is the sentiment over time graph which shows a consistent ~85% negative, ~40% positive sentiment over a couple of years. The rest of the data similarly shows that the subreddit is overall much more negative of Trump than positive.

While very pretty, the maps I produced are mostly misleading. Reading from the csv, every state was approximately 40% positive and 85% positive. The color variance is mostly due to me setting the vmin and vmax to be fairly close together along with the small sample size I was forced to use due to my bobo computer.

The top ten lists are also fairly misleading, with both lists being populated entirely with submissions with a small number of comments making it easy to achieve a 100% positive/negative rate. While my aggressive sampling certainly contributed, I suspect that a substantially thorough scraper would have acquired more than ten low-comment submissions without my help.

The submission score plot revealed that all submissions regardless of score attracted similar distributions of positive and negative comments. Having a higher score brought the distribution closer to the previously mentioned 85%, 40% which makes sense since a higher score implies more comments which would approach the overall distribution. On the other hand, all very high scoring comments are negative, demonstrating a clear bias towards negative comments.

1)
The functional dependencies implied are Input_id → (labelDem, labelGop, labelDJT)

2)
The data is not free of redundancies as every comment includes information about the author such as cakeday and flair which are dependent solely on the author despite the author not being a candidate key. Decomposing this relation would require moving all the author dependent information to a separate relation for which the author is the primary key. My theory for why the collector stored the information this way is because this was the format that reddit provided the data and the collector did not want to devote computing resources to parsing it. It may also be the collector figured that the time saved by not having to perform a join outweighed the cost of storing the extra data.

3)
```
In [193]: ^Icount_states = count_states.join(pos_result_states, pos_result_states.pflair ==
count_states.flair)

   ...: .select('flair', 'count', 'posCount').explain()
== Physical Plan ==
*(4) Project [flair#1, count#3047L, posCount#3062L]
+- *(4) BroadcastHashJoin [flair#1], [pflair#3061], Inner, BuildRight
   :- *(4) HashAggregate(keys=[flair#1], functions=[count(1)])
   :  +- Exchange hashpartitioning(flair#1, 200)
   :     +- *(1) HashAggregate(keys=[flair#1], functions=[partial_count(1)])
   :        +- *(1) Project [flair#1]
   :           +- *(1) Filter (flair#1 INSET (Missouri,Nevada,Rhode Island,Maryland,District of
Columbia,Vermont,Delaware,Oregon,Wisconsin,Iowa,Virginia,Idaho,North
Dakota,Illinois,Mississippi,North Carolina,Alaska,Minnesota,New
Jersey,Indiana,Massachusetts,Arkansas,Arizona,Nebraska,Maine,Kentucky,Montana,Alabama,Pennsylva
nia,Tennessee,Colorado,South Carolina,Hawaii,New
Hampshire,Utah,Ohio,Florida,Louisiana,Washington,South Dakota,New
York,Michigan,Kansas,Texas,Georgia,California,Wyoming,Connecticut,Oklahoma,New Mexico,West
Virginia) && isnotnull(flair#1))
   :              +- *(1) FileScan parquet [flair#1] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[file:/home/cs143/www/negResult.parquet], PartitionFilters: [], PushedFilters:
[In(flair, [Missouri,Nevada,Rhode Island,Maryland,District of Columbia,Vermont,Delaware,Oregon,Wi...,
ReadSchema: struct<flair:string>
   +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, true]))
      +- *(3) HashAggregate(keys=[flair#1], functions=[count(1)])
         +- Exchange hashpartitioning(flair#1, 200)
            +- *(2) HashAggregate(keys=[flair#1], functions=[partial_count(1)])
               +- *(2) Project [flair#1]
                  +- *(2) Filter (((isnotnull(pos%#5) && flair#1 INSET (Missouri,Nevada,Rhode
Island,Maryland,District of Columbia,Vermont,Delaware,Oregon,Wisconsin,Iowa,Virginia,Idaho,North
Dakota,Illinois,Mississippi,North Carolina,Alaska,Minnesota,New
Jersey,Indiana,Massachusetts,Arkansas,Arizona,Nebraska,Maine,Kentucky,Montana,Alabama,Pennsylva
nia,Tennessee,Colorado,South Carolina,Hawaii,New
Hampshire,Utah,Ohio,Florida,Louisiana,Washington,South Dakota,New
York,Michigan,Kansas,Texas,Georgia,California,Wyoming,Connecticut,Oklahoma,New Mexico,West
Virginia)) && (pos%#5 = 1)) && isnotnull(flair#1))
                  +- *(2) FileScan parquet [flair#1,pos%#5] Batched: true, Format: Parquet, Location:
InMemoryFileIndex[file:/home/cs143/www/negResult.parquet], PartitionFilters: [], PushedFilters:
[IsNotNull(pos%), In(flair, [Missouri,Nevada,Rhode Island,Maryland,District of Columbia,Vermont,D...,
ReadSchema: struct<flair:string,pos%:int>
```

The plan seems to involve hashing the flair of the outer relation, filtering the inner relation for equality on the flair, then combining on a match. This appears to be a hash join. Interestingly, the plan makes multiple references to a parquet despite neither relation being directly from one. I assume this is because Spark hasn't actually computed the to-be-joined relations and needs to read from a parquet to first compute them before performing the join.