Problem 1 SQL Querying

All SQL script may be found in the **SQL_Problem** folder

A.

```
SELECT (SELECT count(*) FROM driver WHERE NOT is_test_account) * 100 / count(*) AS
not_test_driver_percentage
FROM driver
```

B.

```
SELECT count(*)
FROM trips
WHERE 1 = 1
AND status = 'completed'
AND completed_at >= '2016-01-01 00:00:00'
AND completed_at <= '2016-12-31 23:59:59'
AND driver_uuid NOT IN (SELECT uuid FROM driver WHERE is_test_account)
```

C.

```
SELECT 1.0 * count(tc_filter.uuid) / count(distinct(driver_uuid)) AS average_trip_per_driver,
tc_filter.city_name
FROM
(SELECT * FROM
(SELECT t.uuid, t.driver_uuid, t.request_at, c.timezone, c.city_name, c.country_name
FROM trips AS t
INNER JOIN city AS c ON t.city_uuid = c.uuid
WHERE 1 = 1
AND c.country_name = 'United State') AS tc
WHERE 1 = 1
AND CONVERT(timestamp, SWITCHOFFSET(tc.request_at, DATENAME(TzOffset, tc.timezone))) >= '2017-01-01
00:00:00'
AND CONVERT(timestamp, SWITCHOFFSET(tc.request_at, DATENAME(TzOffset, tc.timezone))) <= '2017-01-31
23:59:59') AS tc_filter
GROUP BY tc_filter.city_name
HAVING count(tc_filter.uuid) > 100000
```

D.

```
CREATE TABLE cancellation_rate AS
SELECT t.driver_uuid, t.uuid AS trip_uuid,
1.0 *
(SELECT count(*)
FROM trips
WHERE 1 = 1
AND t.driver_uuid = trips.driver_uuid
AND t.request_at >= trips.request_at
AND status = 'cancelled' )
/
(SELECT count(*)
FROM trips
WHERE 1 = 1
AND t.driver_uuid = trips.driver_uuid
AND t.request_at >= trips.request_at) AS pct_cancelled,

1.0 *
(SELECT count(*)
FROM (SELECT *
FROM trips
WHERE 1 = 1
AND t.driver_uuid = trips.driver_uuid
AND t.request_at >= trips.request_at
ORDER BY trips.request_at
LIMIT 100) AS t1
WHERE t1.status = 'cancelled')
/
(SELECT count(*)
FROM (SELECT *
FROM trips
where 1 = 1
AND t.driver_uuid = trips.driver_uuid
AND t.request_at >= trips.request_at
ORDER BY trips.request_at
LIMIT 100) AS t2) AS pct_cancelled_last100

FROM trips AS t
WHERE t.driver_uuid NOT IN (SELECT uuid FROM driver WHERE is_test_account)
```

Problem 2 Data Quality / Data Analysis

Language: Python
Package: pandas, numpy, matplotlib, statsmodels, sklearn, seaborn
Source code file: **Problem2.py**

A.

Data cleaning is performed by function ***filter_raw_data(data)***. The following steps have been done:

Step 1 Reformat Data and Transform Data Type: 1) Reformat data in 'Miles' column, i.e. number string '19,380' is reformatted to '19380'; 2) Transform data type in 'Miles' from string to integer type. Transform 'Month_Ending' column from string to pandas.Timestamp; 3) Using integer type to represent categorical data in 'Product' and 'City' column for better sorting and modeling purposes.

Step 2 Filter Invalid Values: Remove rows has **zero values** in 'Miles' column and **negative values** in 'Reported_Accidents' column

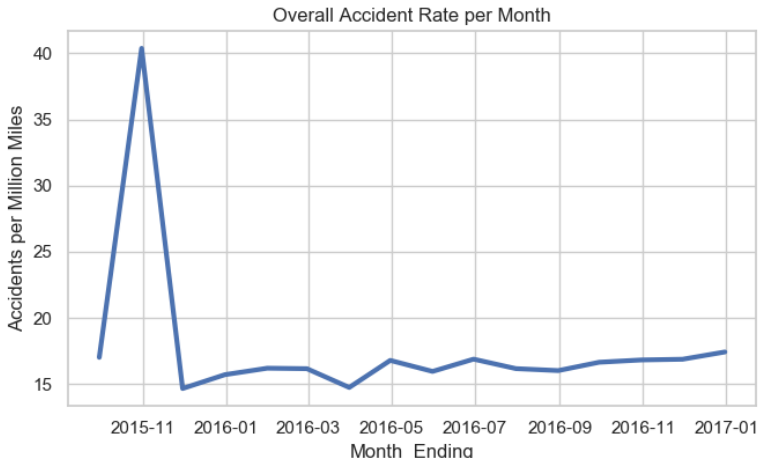
Step 3 Remove Outliers: Using quantile, remove rows has more than 2000 accidents per million miles, since 2000 accidents are too large to be true. However, these rows need to be double checked for correctness, because some of them might be true value. For example, row 6541

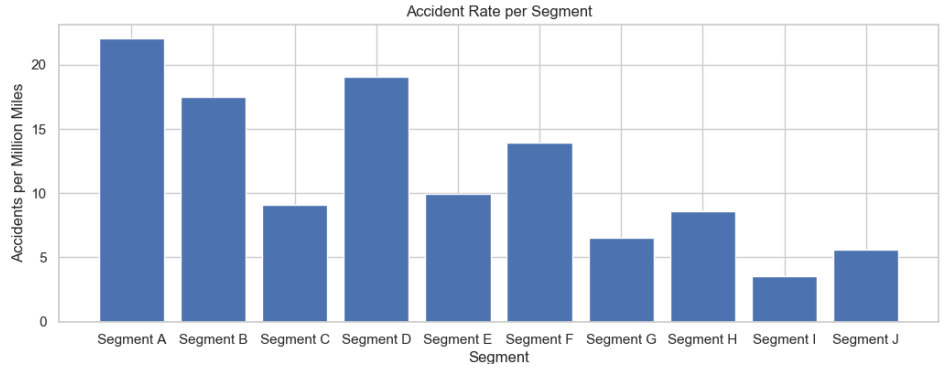
11/30/16	Segment F	City 57	Product 9	8,885	255

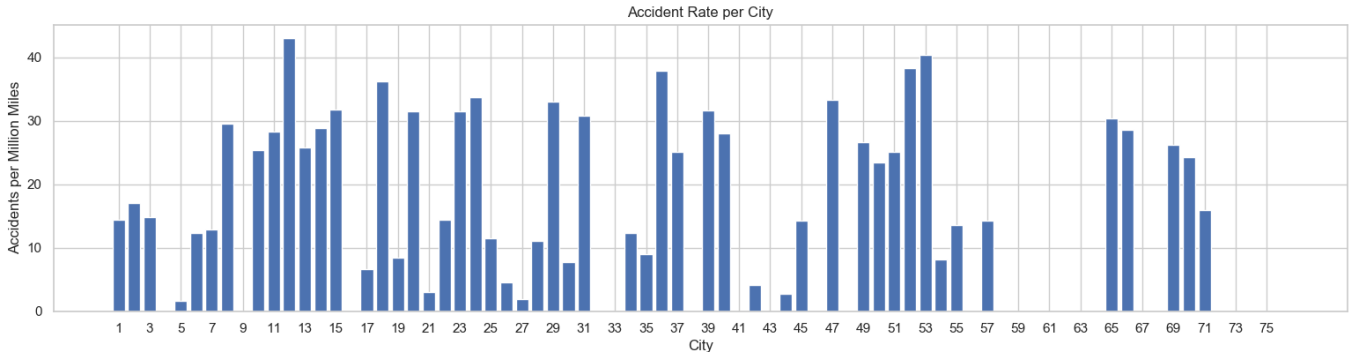
has too many accidents to be believed in.

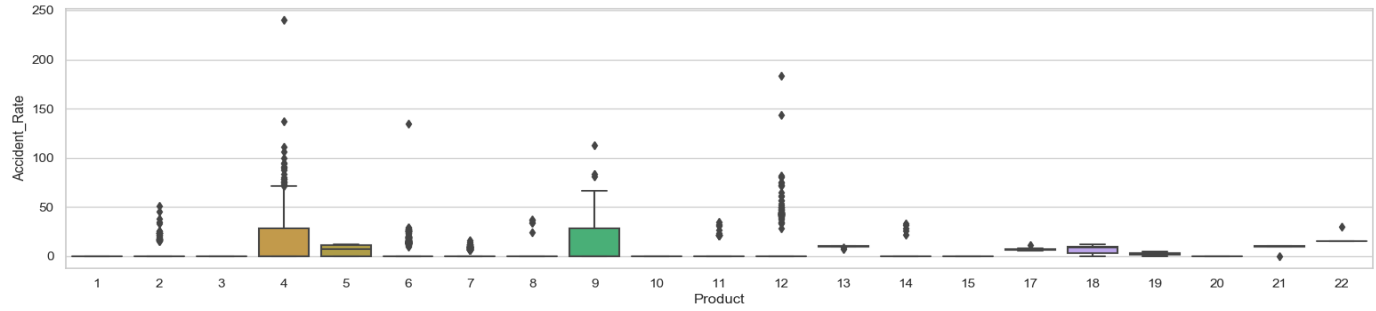
B.

Four figures are provided as follows. Figure 1 is the overall accident rate per million miles per month. Figure 2 shows accident rate by each segment. Figure 3 is the accident rate by city. Figure 4 is a box plot showing the distribution of accident rate across all product. All figures could be reproduced by **Problem2.py.**







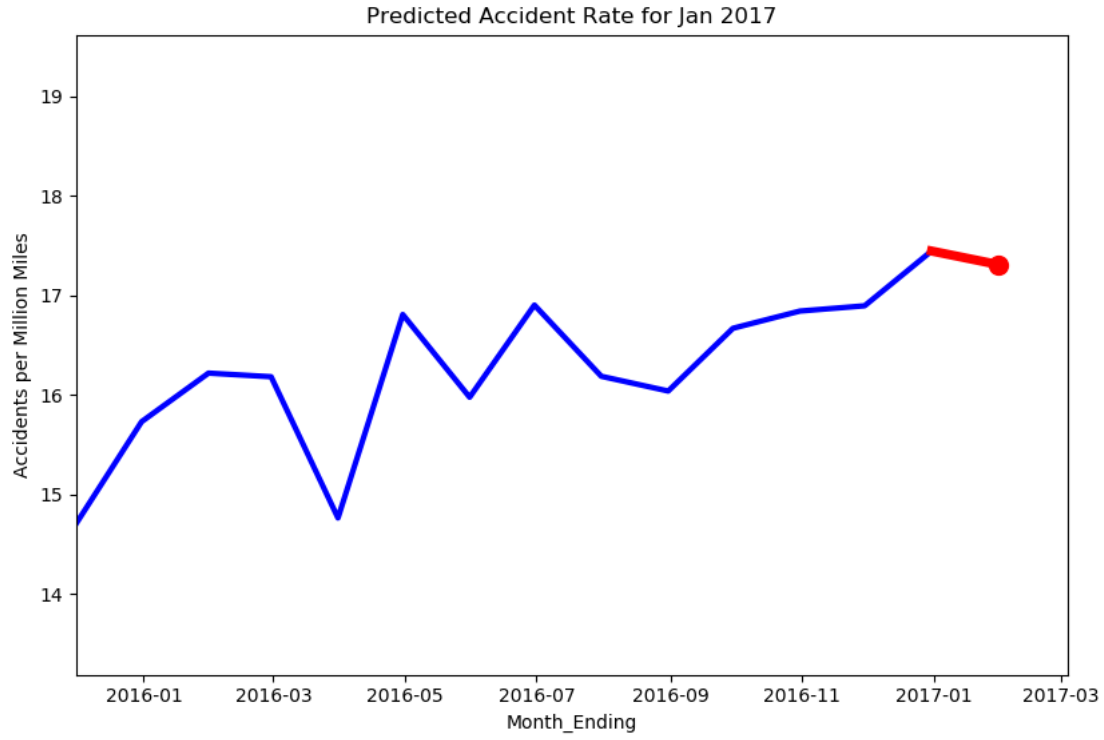


C.

An Autoregressive model is developed to predict the accident rate for Jan 2017. The prediction function in the code is ***auto_regressive(data, p = 6)***. The prediction is made using the accident rate in the past 6 months.

Predicted accident rate of Jan 2017 is: 17.3079 accidents per million miles

Plotting the predicted data together with the previous month:



D.

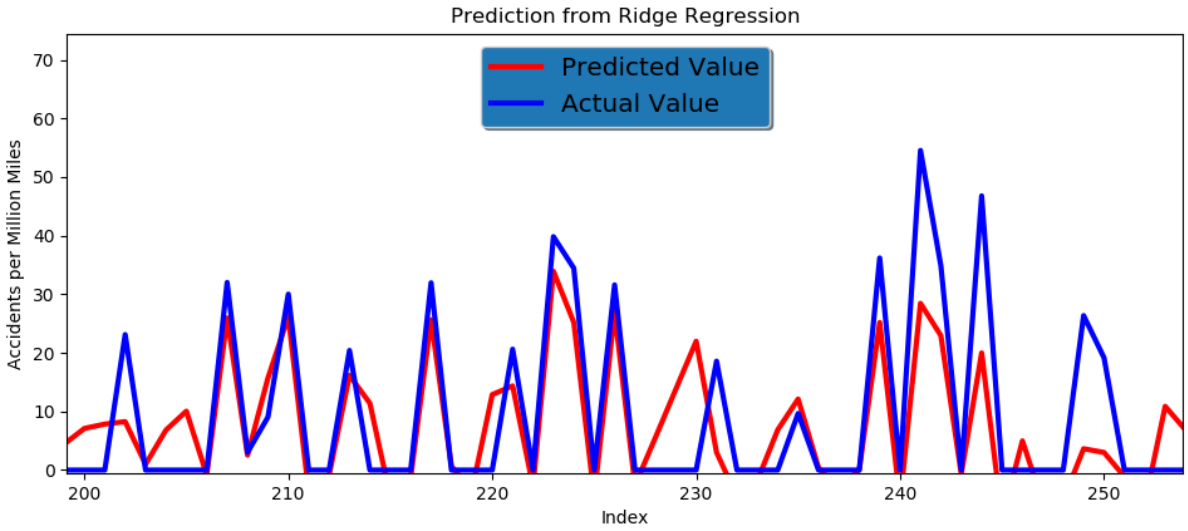
A machine learning model is built to fit the data. 85% of the data is used for training and k-fold cross validation, while 15% of data is used for testing. Scikit-learn package is applied for the modeling, training, and testing. The model **evaluation metric is R^2**.

a. A Ridge Regression with 3-fold cross validation is built in **Problem2.py** from line **125 to line 166**. Since the volume of data and number of features are small, a first order linear model is sufficient.

b. **1) Feature Selection:** Selecting features that are independent, relevant to the output, and have a greater variance. My approach is to start with 'City' as the first feature. Once adding or reducing a new feature, re-train and re-evaluate the model using R^2. If the R^2 value increases by adding the new feature, thus this new feature could increase the predicting power. For example, by adding and removing 'Month_Ending', R^2 does not change across the testing data, then 'Month_Ending' is not a good feature. The final features are: 'Segment', 'City', and 'Product'.

2) Feature Encoding: Since all selected features are categorical features, they must be encoded using one-hot encoding. One-hot encoding is performed by scikit-learn preprocessing.OneHotEncoder.

c. The model is evaluated by R^2. By shuffling the data, **R^2 of the testing data is ranging from: 0.40 – 0.49.** Meaning that 40% to 49% of the information in the data could be represented by the model. Because there are only 3 categorical features without any continuous features, 0.49 is the best R^2 result I could get. The predicting power could be better if there are more features. The following figure is a segment from the testing data showing the predicting power.



E.

The initial problem was asking for testing accident **per mile**. Since the number is so small, I am still using accident **per million mile** in here.

a. For Segment G: $\bar{X}_G = 6.9717, s_G = 1.3036, n_G = 16$
For Segment A: $\bar{X}_A = 7.1405, s_A = 16.093,\ n_A = 5535$
Our hypothesis is: H_0: $\mu_A = \mu_G$, H_1: $\mu_A \neq \mu_G$

Assuming H_0 is true, the level of significance is $\alpha = 0.05$, so the critical values of z are -1.96 and +1.96.

Z score is: $z = \frac{(\bar{X}_A - \bar{X}_G) - (\mu_A - \mu_G)}{\sqrt{\sigma_A^2/n_A + \sigma_G^2/n_G}} = \frac{(7.1405 - 6.9717) - 0}{\sqrt{16.093^2/5535 + 1.3036^2/16}} = 0.43154$

-1.96 < 0.43154 < 1.96. Fail to reject H_0. Accident rate of Segment G and Segment A is the same (no difference).

b. For Segment G: $\bar{X}_G = 6.9717, s_G = 1.3036, n_G = 16$
For Segment A: $\bar{X}_B = 4.2798, s_B = 9.2424,\ n_A = 596$
Our hypothesis is: $H_0: \mu_B = \mu_G,\ H_1: \mu_B \neq \mu_G$

Assuming H_0 is true, the level of significance is $\alpha = 0.05$, so the critical values of z are -1.96 and +1.96.
Z score is: $z = \frac{(\bar{X}_G - \bar{X}_B) - (\mu_B - \mu_G)}{\sqrt{\sigma_G^2/n_G + \sigma_B^2/n_B}} = \frac{(4.2798 - 6.9717) - 0}{\sqrt{1.3036^2/16 + 9.2424^2/596}} = -5.3888$

-5.3888 < -1.96. Reject H_0. Accident rate of Segment G and Segment B is different.

c. For Segment G: $\bar{X}_G = 6.9717, s_G = 1.3036, n_G = 16$
For Segment A: $\bar{X}_A = 7.1405, s_A = 16.093,\ n_A = 5535$
Our hypothesis is: $H_0: 0.6\mu_A > \mu_G,\ H_1: 0.6\mu_A \leq \mu_G$

Assuming H_0 is true, the level of significance is $\alpha = 0.05$, so the critical values of z are -1.96 and +1.96.
Z score is: $z = \frac{(0.6\bar{X}_A - \bar{X}_G) - (\mu_A - \mu_G)}{\sqrt{\sigma_A^2/n_A + \sigma_G^2/n_G}} = \frac{(0.6 * 7.1405 - 6.9717) - 0}{\sqrt{16.093^2/5535 + 1.3036^2/16}} = -6.8704$

-6.8704 < -1.96. Reject H_0. Accident rate of Segment G is not 40% lower than Segment A.

Problem 3 Spatial Analysis

Language: Python
Package: pandas, googlemaps, numpy, folium, geopandas, multiprocessing
Source code file: **Problem3.py**

A.

For **raw data**:
Mean: **1446.825**
Median: **872.002**
Absolute difference of 75th and 25th percentile: **1330.085**

For **filtered data**:
Mean: **1393.443**
Median: **866.585**
Absolute difference of 75th and 25th percentile: **1308.206**

B.

The data is filtered based on the widgets quantile. The **maximum** value of the widgets is **532006.925**. However, the **99.5%** quantile is only **9288.831**, meaning the data is long tailed and majority of the values are below **9288.831**. So, the widgets values above **99.5%** quantile could be discarded.

Similarly, the **0.05%** quantile appears at **0.10538**, while the **minimum** widgets value is **-0.888099**. Considering the mean value is **1393** and median value is **866**, values below **0.05%** quantile have negligible contribution in Problem C and the negative widgets values could also be noises as well.

Therefore, the filtered data only preserve widgets values between **0.05% (0.10538)** and **99.5% (9288.831)** quantile. Deleting data from the head and tail will have counter affects towards mean and medium. However, the large values at the tail contributes more to the mean and median. Both mean and medium shift to smaller values in the filtered data.

C.

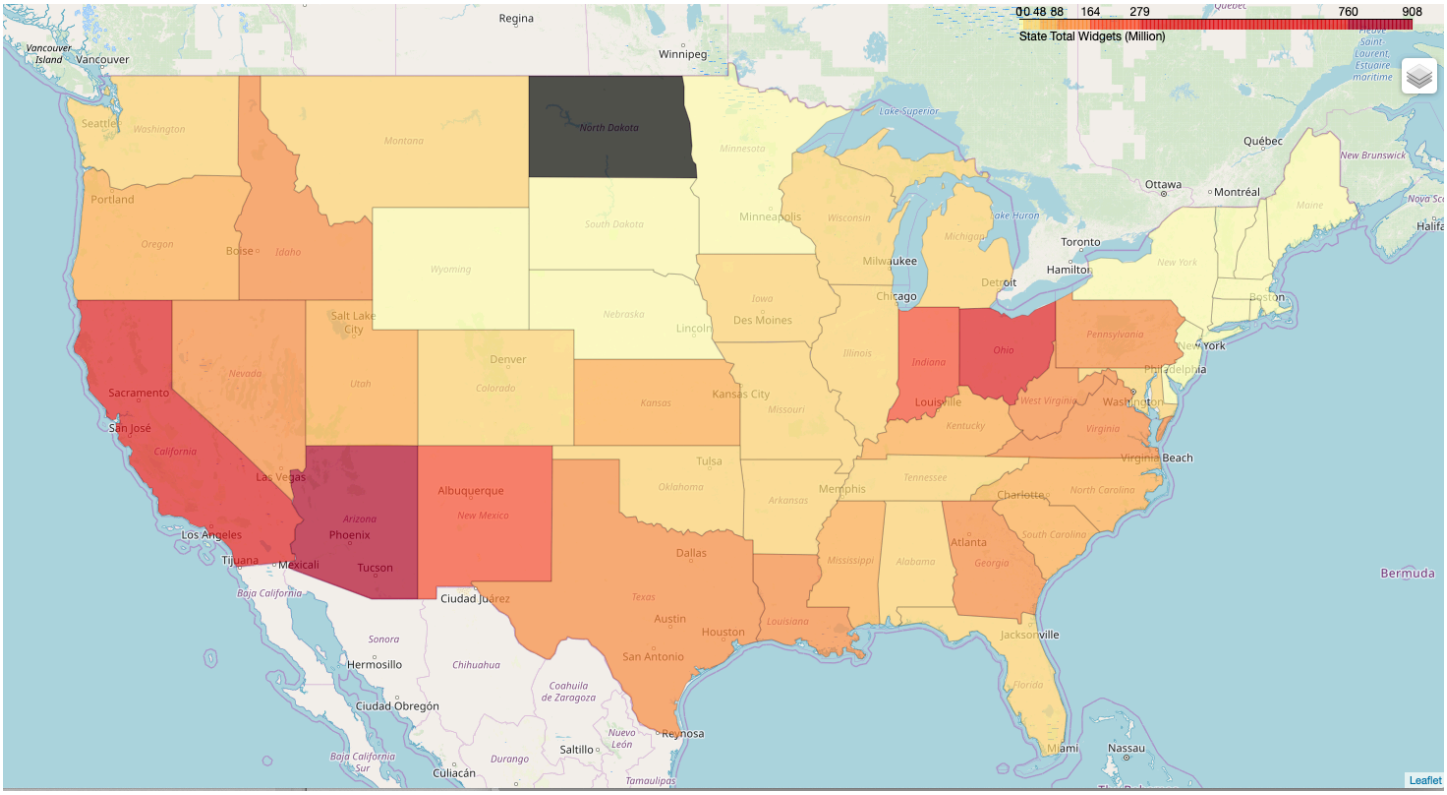
Big Data Approach:
Since the data volume is very large, a **MapReduce** design pattern is employed. Data set is split into **16** chunks and run by **16** parallel processes in function ***parallelize_dataframe(df, func, num_partitions, num_threads)***. Each process works on a chunk of data and calculate the state total widgets on that single chunk. Finally, the master merger summarizes the 16 results into the final result.

To accelerate the process, the code was deployed to an AWS EC2 cloud computing server with 16 cores and 128GB memory.

Geo Location Querying:
I implemented two methods for coordinate-to-state query. (1) Using Reverse Geocoding from Google Map in function ***get_state_count_google(data)***; (2) Using point-in-polygon search in function ***get_state_count(data)***.

Both (1) and (2) works for this problem, however, I finally applied method (2) because method (1) was IO bounded, and required constant Internet traffic to Google server.

a. **South Dakota** has the minimum total widgets at **9903.68**.
b. **New Mexico** has the 4th highest total widgets at **213280521.41**.
c. The table to total widgets by state is as below. It could also be found in: **state_widgets_count.csv.** A choropleth is attached as below. The interactive choropleth map is in: **State_Total_Widgets.html**



state	total_widgets
Alabama	21516533.2
Arizona	908227710
Arkansas	15670532
California	600450851
Colorado	38266531.4
Connecticut	262525.819
Delaware	8829184.28
District of Columbia	219152.492
Florida	41727423.6
Georgia	124275730
Hawaii	367774.546
Idaho	87536593.3
Illinois	33568196.6
Indiana	189354517
Iowa	12286172.8
Kansas	83483735.7

Kentucky	71199648
Louisiana	107832139
Maine	55980.2564
Maryland	42117649.3
Massachusetts	5049437.46
Michigan	25175804.5
Minnesota	2872221.36
Mississippi	60188112.4
Missouri	16936523.7
Montana	40169063.5
Nebraska	560656.387
Nevada	158101728
New Hampshire	67912.876
New Jersey	8824739.47
New Mexico	213280521
New York	6638641.02
North Carolina	80763532.7
Ohio	322087816
Oklahoma	29186074.1
Oregon	71474928.7
Pennsylvania	93920668.4
Rhode Island	292112.95
South Carolina	79306536.1
South Dakota	9903.68379
Tennessee	21964961.1
Texas	148795191
Utah	48899767.5
Vermont	32415.7107
Virginia	107499817
Washington	29574436.7
West Virginia	133604484
Wisconsin	17502276.7
Wyoming	2601794.57