Water injection wells and earthquakes in Oklahoma

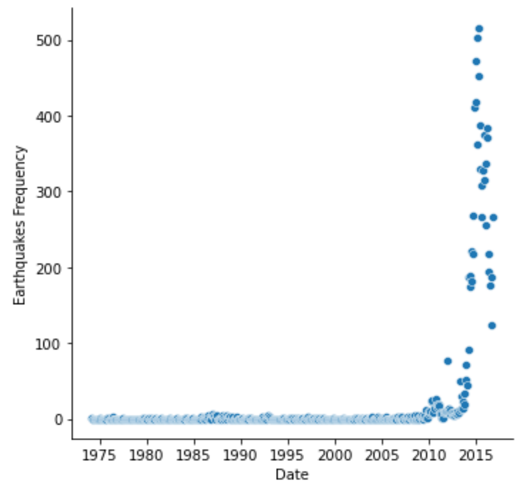
(From Kaggle)

https://www.kaggle.com/ksuchris2000/oklahoma-earthquakes-and-saltwater-injection-wells

Introduction

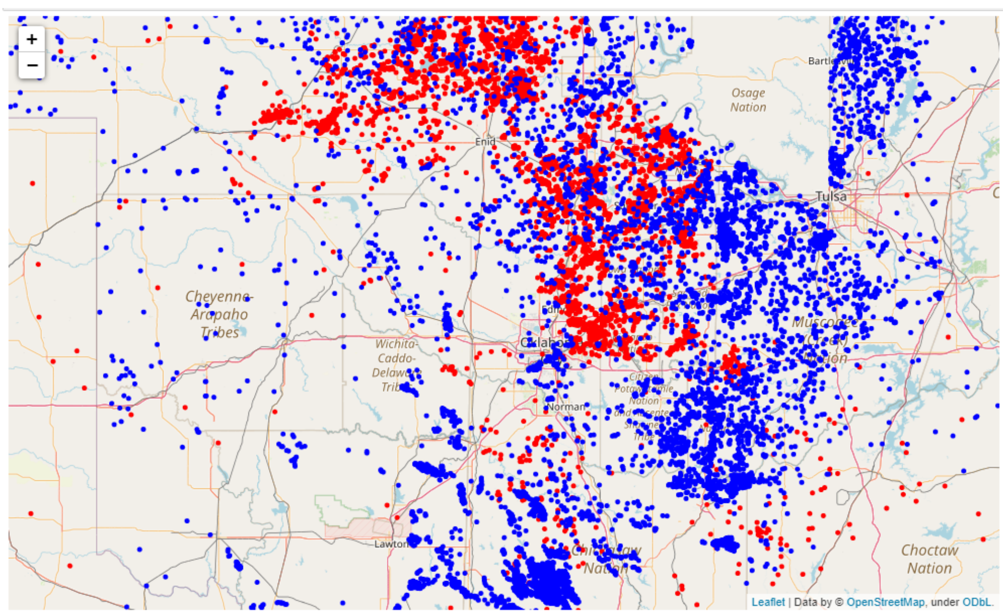
Beginning in 2009, the frequency of earthquakes in the U.S. State of Oklahoma rapidly increased from an average of fewer than two 3.0+ magnitude earthquakes per year since 1978 to hundreds per year in 2014, 2015, and 2016. Thousands of earthquakes have occurred in Oklahoma and surrounding areas in southern Kansas and North Texas since 2009. Scientific studies attribute the rise in earthquakes to the disposal of wastewater produced during oil extraction that has been injected deeply into the ground. (Wikipedia)

Injection wells are utilized to dispose of fluid created as a byproduct of oil and gas production activities. Likewise, hydraulic fracturing, ie "fracking", produces large byproducts of water. This byproduct is then injected deep back into the earth via disposal/injection wells.



Volume of Earthquakes per Month

Locations of wells and earthquakes



Red dots = Earthquakes

Blue dots = Active Injection Wells

Can the future volume of earthquakes be predicted from the number of injection wells and the volume of wastewater injected back into the subsurface?

Data Wrangling

Wells Dataframe:

```
Int64Index: 11125 entries, 0 to 11124
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   API#           11125 non-null  float64
 1   Operator       11125 non-null  object
 2   Operator ID    11125 non-null  float64
 3   WellType       11125 non-null  object
 4   WellName       11124 non-null  object
 5   WellNumber     11124 non-null  object
 6   OrderNumbers   11124 non-null  float64
 7   Approval Date  11125 non-null  object
 8   County         11125 non-null  object
 9   Sec            11125 non-null  object
 10  Twp            11125 non-null  object
 11  Rng            11125 non-null  object
 12  QQQQ           11125 non-null  object
 13  LAT            11125 non-null  float64
 14  LONG           11125 non-null  float64
 15  PSI            9689 non-null   object
 16  BBLS           9689 non-null   object
 17  ZONE           11125 non-null  object
 18  Unnamed: 18    0 non-null      float64
 19  Unnamed: 19    0 non-null      float64
 20  Unnamed: 20    0 non-null      float64
dtypes: float64(8), object(13)
```

Earthquakes Dataframe:

```
RangeIndex: 13954 entries, 0 to 13953
Data columns (total 23 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   time             13954 non-null  object
 1   latitude         13954 non-null  float64
 2   longitude        13954 non-null  float64
 3   depth            13954 non-null  float64
 4   mag              13948 non-null  float64
 5   magType          13933 non-null  object
 6   nst              5389 non-null   float64
 7   gap              12433 non-null  float64
 8   dmin             5621 non-null   float64
 9   rms              12749 non-null  float64
 10  net              13954 non-null  object
 11  id               13954 non-null  object
 12  updated          13954 non-null  object
 13  place            13954 non-null  object
 14  type             13954 non-null  object
 15  horizontalError  10756 non-null  float64
 16  depthError       12144 non-null  float64
 17  magError         6055 non-null   float64
 18  magNst           6133 non-null   float64
 19  status           13954 non-null  object
 20  locationSource   13954 non-null  object
 21  magSource        13954 non-null  object
 22  marker_color     13948 non-null  category
dtypes: category(1), float64(12), object(10)
```

The approach for data wrangling was to make each remove all unnecessary features and investigate and fix all null values. After data wrangling, every row for each feature should have a relevant value and each feature should have the correct Dtype for analysis. When looking at the Wells dataframe on the left, the ‘PSI’ and ‘BBLS’ features were the only features that had to have a creative solution to fix, not only are there several thousand null values, but the dtype is object instead of float.

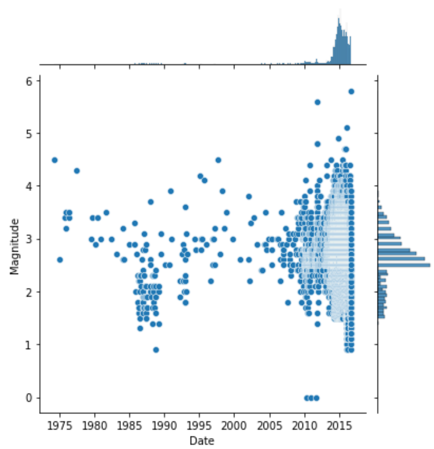
```
#There were several wells with PSI values that didn't make sense or were an incorrect format
#I saved these to a temp dataset so that they could be replaced
temp = wells.loc[wells['PSI'].isin(['224B/475G','226B/479G', '226B/479G','222B/470G', '235B/498G', '197B/416G',
       '225B/476G', '225B/475G', '223B/472G', '204B/432G', '223B/471G'
    '228B/483G', '229B/484G', '229B/486G', '236B/499G', '230B/487G',
       '202B/428G', '226B/478G', '227B/481G', '218B/461G', '227B/480G','219B/464G','223B/471G', '228B/483G'])]

#Removed the odd wells from the main dataset
wells =  wells.loc[~wells['PSI'].isin(['224B/475G','226B/479G', '226B/479G','222B/470G', '235B/498G', '197B/416G',
       '225B/476G', '225B/475G', '223B/472G', '204B/432G', '223B/471G'
    '228B/483G', '229B/484G', '229B/486G', '236B/499G', '230B/487G',
       '202B/428G', '226B/478G', '227B/481G', '218B/461G', '227B/480G','219B/464G','223B/471G', '228B/483G',])]

#Converted the PSI column in the main dataset to type float and found the median (not shown)
wells['PSI'] = wells['PSI'].astype(float)

#Set the PSI in the temp dataset to the median and set to type float so it could be combined back to the main dataset
temp['PSI1'] = '750'
temp = temp.drop(['PSI'], axis=1)
temp.rename(columns = {'PSI1':'PSI'}, inplace = True)
temp['PSI'] = temp['PSI'].astype(float)

#Concatinated the temp dataset with the main dataset
wells = pd.concat([wells, temp])

#Set BBLS (volume) column to type float
wells['BBLS'] = wells['BBLS'].astype(float)
```

Code Snippet Showing Interesting Data Wrangling

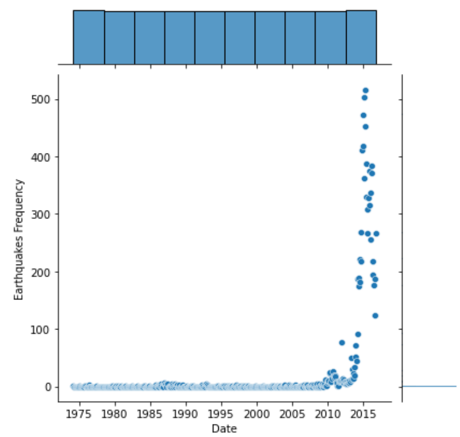
In order to take care of this it was necessary to remove all commas from the values so that it could be converted to float. Even after commas were removed it still wouldn’t convert so I looked at the unique values. There were quite a few odd values with this format: '202B/428G', '226B/478G','227B/481G', '218B/461G'. In order to fix this I removed those values to a temporary dataframe, changed the type of the rest of the remaining rows to float and found the median. Then in the temporary dataframe I replaced the odd values with the median, converted it to float, and merged it with the main dataset. Now there were no null values and it was the correct dtype.

Exploratory Data Analysis

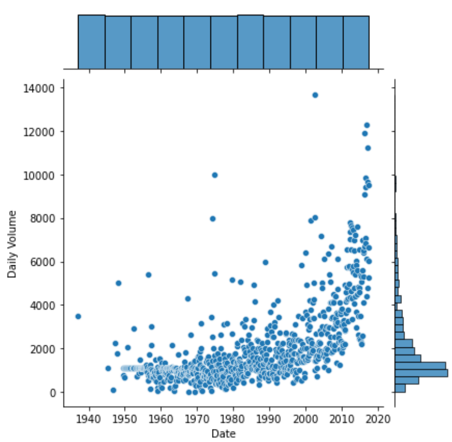
For exploratory data analysis I did preliminary plotting to see the relationships between the features and the two dataframes. This is very easily done with pandas profiling to see histograms of each feature as well to see if you have any remaining null or extreme values. After exploring with pandas profiling I made several plots exploring the features changing over time.



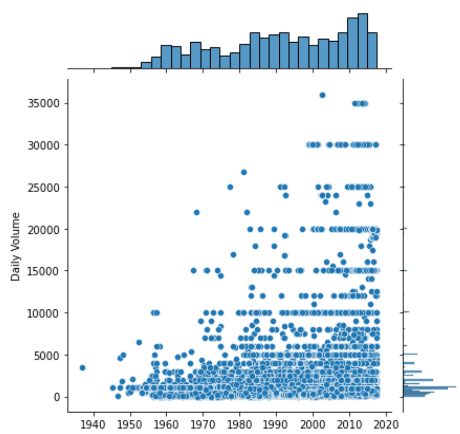
Earthquake Magnitude Over Time



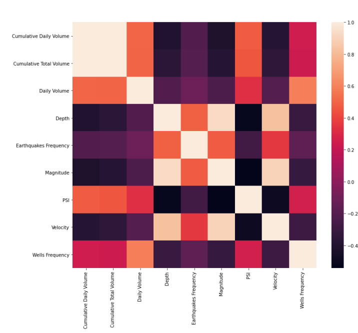
Earthquake Frequency Over Time



Daily Volume per Month Over Time



Daily Volume Rate Over Time



Pearson Correlation Heatmap

Feature Engineering

I suspected that part of the reason for the increase in earthquakes was not just more wells, but the cumulative effect of the water injection. To take account of this I had to create a few features. I had to calculate the cumulative volume injected over time based on each injection well's daily volume rate.

```
#Resample to Days so there are no duplicate dates in order to calculate new features, Wells Frequecy and Daily Volume are summed
wells_day = wells.resample('D').mean()
wells_day['Wells Frequency'] = wells['Wells Frequency'].resample('D').sum()
wells_day['Daily Volume'] = wells['Daily Volume'].resample('D').sum()
wells_day = wells_day.dropna()
wells_day.head()
```

Date	Daily Volume	Latitude	Longitude	PSI	Wells Frequency
1936-12-18	3500.0	34.199067	-97.399092	1100.0	1
1945-04-22	1100.0	36.901903	-95.900888	750.0	1
1946-10-19	100.0	35.511472	-96.767417	0.0	1
1947-03-18	2200.0	36.166273	-96.720024	750.0	2
1947-03-28	4600.0	36.718930	-95.546017	0.0	1

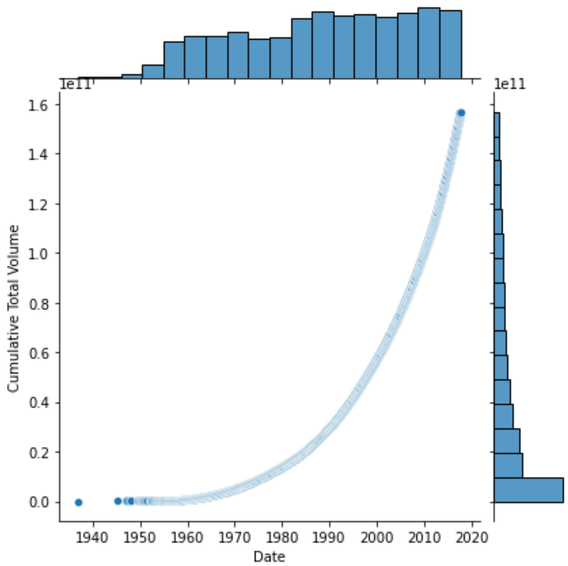
Calculating Total Daily Volume

```
#New features are calculated
    #Cumulative Daily Volume: Total daily voluume of all previous wells
        #It is shifted down 1 row to show the volume at the time that well comes on line
    #Cumulative Total Volume: Total daily volume multiplied by days since last well then summed
wells_day['Cumulative Daily Volume'] = wells_day['Daily Volume']
wells_day['Cumulative Daily Volume'] = wells_day['Cumulative Daily Volume'].cumsum().shift(1).fillna(0)
wells_day['Cumulative Total Volume'] = (wells_day['Cumulative Daily Volume'] * wells_day['Days Since Last Well']).cumsum()
wells_day.head(5)
```

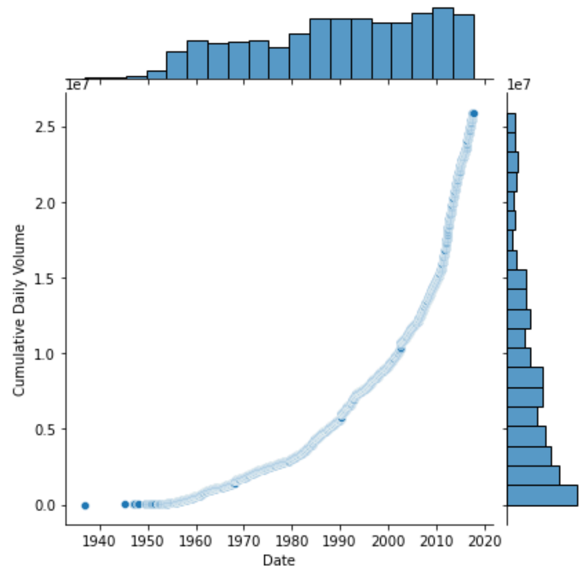
Date	Daily Volume	Latitude	Longitude	PSI	Wells Frequency	Days Since Last Well	Cumulative Daily Volume	Cumulative Total Volume
1936-12-18	3500.0	34.199067	-97.399092	1100.0	1	0.0	0.0	0.0
1945-04-22	1100.0	36.901903	-95.900888	750.0	1	3047.0	3500.0	10664500.0
1946-10-19	100.0	35.511472	-96.767417	0.0	1	545.0	4600.0	13171500.0
1947-03-18	2200.0	36.166273	-96.720024	750.0	2	150.0	4700.0	13876500.0
1947-03-28	4600.0	36.718930	-95.546017	0.0	1	10.0	6900.0	13945500.0

Calculating Cumulative Volume Over Time Code Snippet

In order to calculate the cumulative total over time, I summed the individual daily rates and shifted them down a row. This made it so 'Cumulative Daily Volume' was the total rate of all previous wells. I also calculated a 'Days Since Last Well' feature to take care of the time component. Then all I had to do was multiply these together and sum them down the column for the 'Cumulative Total Volume' over time.



Cumulative Total Volume



Cumulative Daily Volume

Preprocessing

In order to prepare the datasets for modeling there were several steps taken. The first was to resample the dataset into larger regularized bins. After the bins were created all columns present in the other dataset were created and populated with zeros to make the dataset merge easier.

```
#Data sets resampled to weeks for modeling, counter features and volumes were summed
wells_resamp = wells_day.resample('W').mean()
earthquakes_resamp = earthquakes.resample('W').mean()
wells_resamp['Wells Frequency'] = wells_day['Wells Frequency'].resample('W').sum()
wells_resamp['Daily Volume'] = wells_day['Daily Volume'].resample('W').sum()
#wells_resamp['Cumulative Daily Volume'] = wells_day['Cumulative Daily Volume'].resample('W').sum()
#wells_resamp['Cumulative Total Volume'] = wells_day['Cumulative Total Volume'].resample('W').sum()
earthquakes_resamp['Earthquakes Frequency'] = earthquakes['Earthquakes Frequency'].resample('W').sum()

#Latitude and Longitude features were dropped since after resampling they were not representative of of the rows
    #I work for a long time creating lat/long bins but once you start lumping by area you lose the time component
    #This is a focus of future work to figure out how to use time and location (Geopandas?)
wells_resamp = wells_resamp.drop(['Latitude', 'Longitude', 'Days Since Last Well'], axis=1)
wells_resamp['Type'] = 'Injection Well'
earthquakes_resamp = earthquakes_resamp.drop(['Latitude', 'Longitude'], axis=1)
earthquakes_resamp['Type'] = 'Earthquake'

#Features of the other dataset are created and set to 0 to prepare for merge
wells_resamp['Depth'] = 0
wells_resamp['Magnitude'] = 0
wells_resamp['Velocity'] = 0

earthquakes_resamp['PSI'] = 0
earthquakes_resamp['Daily Volume'] = 0
earthquakes_resamp['Days Since Last Well'] = 0
earthquakes_resamp['Cumulative Daily Volume'] = 0
earthquakes_resamp['Cumulative Total Volume'] = 0

wells_resamp = wells_resamp.dropna()
earthquakes_resamp = earthquakes_resamp.dropna()
```

Code Snippet Showing Timing Resample

This resample was redone several times during modeling to find the best size of bin. The options that were tried were: 1 day, 3 days, 5 days and 1 week. Anything larger than one week provided too few samples. The binning that gave the best solution was 5 days. After the resample, the two datasets were combined into one. The next process was to one hot encode the 'Type' features. The types differentiate each row as an earthquake or an injection well.

The final step before modeling was to split into train/test sets and scale the features. Since I was trying to use the features to predict the future I couldn't use a standard train/test split so I manually chose dates for a split point. This was also extensively tested for optimal results. I wanted to make sure that the training data saw at least a portion of the ramp up in earthquakes so it would know what to look for in the other features, but not so much that the testing data was too small. Once I closed in on the optimal split point the differences in even a month were rather drastic. Some plot displays of the split results are presented below.

```
#Manually split into train and test datasets
train = merge_onehot.loc[merge_onehot.index < '2014-03-01' ]
test = merge_onehot.loc[merge_onehot.index >= '2014-03-01' ]

#Create X and y datasets
X_train = train.drop(['Earthquakes Frequency'], axis=1)
y_train = train[['Earthquakes Frequency']]
X_test = test.drop(['Earthquakes Frequency'], axis=1)
y_test = test[['Earthquakes Frequency']]

index1 = train.index
index2 = test.index
columns1 = X_test.columns
columns2 = y_test.columns

scale1 = StandardScaler()
scale2 = StandardScaler()

X_train = scale1.fit_transform(X_train)
X_test = scale1.transform(X_test)
scaletemp = scale2.fit(y_train)
```

Code Snippet of Test/Train Split and Scaling

Modeling

For the modeling portion I planned to use a dense neural network. I tested several versions, 2 vs 3 hidden layers, 500 vs 750 vs 1000 nodes, dropout vs no dropout and rate, and settled on a 2 hidden layer, 500 node with 25% dropout model. This model ended up giving about the same results as a 2 or 3 layer 1000 node model but ran in ¼ the time.

```
model2 = Sequential()
model2.add(Dense(500,activation='relu',input_shape=(10,)))
model2.add(Dropout(.25))
model2.add(Dense(500,activation='relu'))
model2.add(Dropout(.25))
model2.add(Dense(1))
model2.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
```

```
start = time.time()
model2.fit(X_train,y_train, epochs=500)
print('It takes %s minutes' % ((time.time() - start)/60))
```

```
...
```

```
# Evaluate the best model.
loss2, accuracy2 = model2.evaluate(X_test, y_test)
print(loss2)
print(accuracy2)
```

```
335/335 [==============================] - 0s 212us/step
1268.0577449768364
0.5343283414840698
```

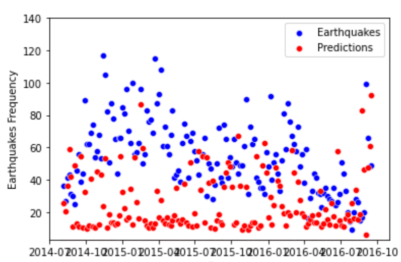
Code Snippet Showing Optimized Dense Neural Network

Date	Predictions	Earthquakes Frequency
2014-08-01	0.990913	0.0
2014-08-04	25.605185	36.0
2014-08-06	0.898451	0.0
2014-08-09	20.239346	27.0
2014-08-11	0.951540	0.0
2014-08-14	36.266479	41.0
2014-08-19	58.872772	43.0
2014-08-21	0.993143	0.0
2014-08-24	42.063889	31.0
2014-08-26	1.000559	0.0
2014-08-29	10.789319	30.0
2014-08-31	0.958262	0.0
2014-09-03	49.174908	25.0
2014-09-05	0.913515	0.0
2014-09-08	13.253757	46.0

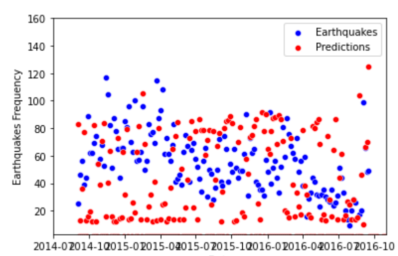
Neural Network Predictions vs Ground Truth

The accuracy output of the model gave an accuracy score of 0.5384615659713745 but it seems to be performing better than this. If you look at the test split results the model never actually predicts a zero but the output is around 1 or below. I believe this indicates that the model quite accurately predicts whether there will be a burst of earthquakes or not.

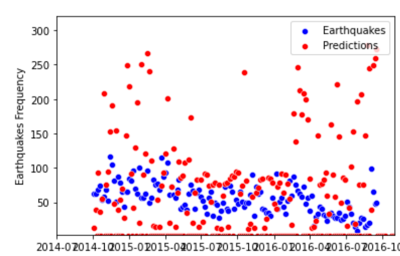
As mentioned above, the train/test split dates made quite a difference on the test predictions. When the split was August 1st, 2014 the model seems to drastically underestimate the results and is very hesitant to predict a frequency of greater than 60 earthquakes per period. I believe this is due to the split barely being on the ramp up and the training not having enough information. When you move forward two months to October 1st, 2014 it appears that the model is given too much information and it expects the ramp to continually go upward. It drastically overestimates the number of earthquakes, you'll notice that the vertical scale is double that of the other plots to be able to see the predictions. It appears that the month right in the middle, September 1st, 2014 is the perfect split. It sees enough of the ramp up to make more accurate predictions but doesn't way over predict.



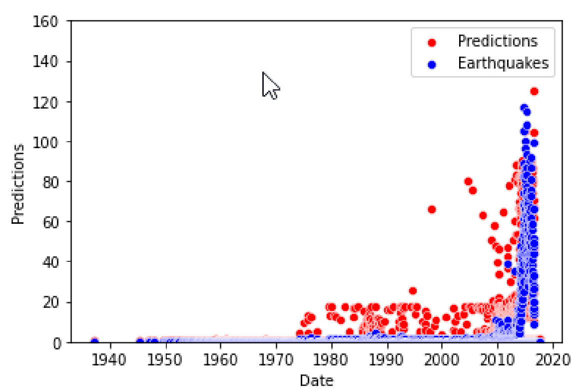
Test Preds August 1st, 2014



Test Preds September 1st, 2014



Test Preds October 1st, 2014



Full Dataset Predictions vs Earthquakes

Future Work
-Figure out what happened ~1975 to cause instability in the training data
-Figure out a spatial component, multiple groupbys so you can have geographical bins but also save the time component
-During the modeling process I spent a lot of time trying to create a LSTM model without much success. This would be a step up in accuracy due to it’s stepwise nature of predictions.