Tornado Risk & Radar Coverage Analysis Dashboard

Documentation Guide

Abstract: Project to analyze historical tornado data and evaluate radar coverage across the United States in order to forecast high-impact tornado risk and identify regions where additional radar stations may be needed, with results visualized through an interactive map interface.

Team Members:

Gennifer Gutierrez and Gayle Taylor

Our first objective was utilizing predictive modeling to forecast geographic regions with the highest probability of experiencing an EF3/F3 tornado or higher. We took data from the National Weather Service (NWS), filtered their data to only show tornados, and imported into our code. Even though data went back to the 1950s, we decided as a team to only focus on the years 1995-2025; 1995 was the first year with significant data from recently implemented NWS radars. NWS implemented additional radar stations through the remainder of the decade. One important note is that in 2008, the Enhanced Fujita Scale was created; going forward, tornados listed as F3 became EF3. In this document, we will refer to the enhanced scale.

Utilizing a for-loop, we assigned high-impact status to the historical data. EF3 and higher were categorized as "1," while under were "0." We successfully added this column to our forecasting_summary table. Following this, we utilized AI to create 16 unique regions across the United States; "Other" was used as a catch-all for earlier radar indicated storms that may not have had a beginning latitude and longitude coordinate.

We then used Tablesaw to count the amount of high-impact tornadoes in a specific region during specific months from 1995-2025. This allowed us to create a historical data set that would be utilized later with regression modeling techniques. After the program calculated the amount of high-impact tornadoes, we added it to the future Excel table. This is when we started taking the historical data and creating our own data to use. We created a Risk column that would take the total tornadoes and the total number of high-impact tornadoes in a region during each month of each year and calculate a risk value.

```
// Column for risk values
DoubleColumn risk = DoubleColumn.create("risk");



// For loop to calculate risk for each row
for (int i = 0; i < summary.rowCount(); i++) {



   double total = summary.intColumn("total_tornadoes").get(i);
   double high = summary.intColumn("high_impact_tornadoes").get(i);



   double riskValue;



   if (total == 0) {
```

```
        riskValue = 0;

    } else {

        riskValue = (high + 1.0) / (total + 2.0);

    }



    risk.append(riskValue);

}
```

We took this data and trained a RandomForest model to predict the number of high-impact tornadoes in a given region for the next five years (2026-2030). We decided to move from taking the mean to the median so that outliers wouldn't affect the data as strongly. The model included the predicted_risk score and a final_risk score to smooth any outliers from our predicted_risk column. Before adding the final_risk, there were egregious outliers; one example showed a 26% predicted risk in January in the Northwest region, which is known for not having a large number of high-impact tornadoes and was outside of the active season. This allowed smoothing to show better risk scores based on the historical data. This then became our forecasting_data file that we would base our second objective on.

```
for (int year = 2026; year <= 2030; year++) {
    for (int monthVal = 1; monthVal <= 12; monthVal++) {
        for (int regionVal = 1; regionVal <= 16; regionVal++) {

            //Filter the historical data for the same region & month
            Table history = summary.where(
                    summary.intColumn("month").isEqualTo(monthVal)

.and(summary.intColumn("int_region").isEqualTo(regionVal)));

            double averageTornados = 0;
            double averageRisk = 0;

            //Calculate the average tornado count
            if (history.rowCount() > 0) {
                averageTornados =
history.intColumn("total_tornadoes").median();//fixing from mean to median to
account for historical low months better
                averageRisk = history.doubleColumn("risk").median();
```

Our second objective was to determine if there were geographic areas where radar coverage may be less effective and where additional radar stations could improve monitoring

coverage, with constraints of current NEXRAD radar stations and the final_risk score of high-impact regions. We inputed our forecasting_data.csv file, created new columns to account for distance and gaps that would affect radar_risk_score, and then used a for-loop to go through each region.

Within this loop, we compared each region's location to all existing radar stations and calculated the distance to each one. From there, we kept the smallest distance, which gave us the nearest radar for that region. This value was added to our nearest_radar_km column so we could track how far each region is from existing coverage.

Within this loop, we calculated the distance from each region to all existing radar stations and identified the nearest one. To do this, we utilized AI to assist in implementing the Haversine helper method, which allowed us to calculate geographic distances using latitude and longitude values. We then used the smallest of those distances to determine the nearest radar for each region and added it to our nearest_radar_km column. After finding the nearest radar, we used that distance to determine if there was a coverage gap. If the distance was above our set threshold, we flagged that region as having limited coverage. This helped us identify areas where radar support may not be as strong, especially when compared to regions with shorter distances to existing stations.

After identifying coverage gaps, we built on that by adding peak season and weights into our radar_need_score. We defined peak season as March through June since those months consistently show higher tornado activity, which allowed us to treat those months differently instead of having every month count the same.

We added a seasonal weight so that regions during peak season would have more influence on the final score, while off-season months were still included but not weighted as heavily. We also applied a coverage weight based on the distance from the nearest radar. Regions that were farther away from existing radar stations were given a higher weight so that rural areas would stand out more in the results. To avoid treating all coverage gaps the same, we used multiple distance thresholds to separate moderate and more significant gaps.

To put everything together, we normalized the distance and tornado values so they could be compared more consistently. We then combined risk, tornado counts, and distance into one score and applied both the seasonal and coverage weights. This gave us a radar_need_score that reflects multiple factors at once and made it easier to compare regions overall.

```
      //adding this in to account for peak season
      risk = row.getDouble("final_risk");
      double tornadoes = row.getDouble("total_tornadoes");
      int month = row.getInt("month");
      double distance = minDistance; // adding this in for the rural area
coverage too
```

```
      // Peak season definition
      boolean peakSeason = (month >= 3 && month <= 6);
      boolean ruralGap = distance > 120; // adding it here so the booleans can
be together

      // Season weight
      double seasonWeight;
      if (peakSeason) {
         seasonWeight = 1.5;
      } else {
         seasonWeight = 0.5;
      }

      //Coverage (rural area boost)
      double coverageWeight;

      if (distance > 150) { //outside the 120 so it would be better more
noticeable.
         coverageWeight = 1.5;
      }
      else if (distance > 100) { //mid
         coverageWeight = 1.2;
      }
      else {
         coverageWeight = 1.0;
      }

      // Normalize inputs
      double normalizedDistance = minDistance / 200.0;
      double normalizedTornadoes = tornadoes / 10.0;

      // Final score
      double radarNeedScore =
             (0.5 * risk +
                    0.3 * normalizedTornadoes +
                    0.2 * normalizedDistance)
                    * seasonWeight
                    * coverageWeight;
```

From this point, our objective required coordinates for new radar placement. Based on radar_needs_score, we determined our top peak season regions were the Mid-Mississippi Valley and Central Plains, and our top coverage gap regions were the Southwest and Midwest regions. These regions stood out due to a combination of higher predicted risk and limited radar coverage.

At this point, we needed a way to actually see how our results looked across regions instead of just reading them from a table. Because both tornado risk and radar coverage are tied to geographic location, using a map made it easier to understand patterns and compare regions visually.

To show all of this, we used Vaadin to create an interactive map that displays both tornado risk and radar need. We first plotted the existing radar stations using their latitude and longitude so we could see what coverage already exists. This gave us a baseline to compare against when looking at coverage gaps and proposed placements.

```
Table radarTable = Table.read().csv("data/nexrad-stations.csv");



Icon.Options radarIconOptions = new Icon.Options();

radarIconOptions.setSrc("images/radar.png");

radarIconOptions.setScale(0.08);

Icon radarIcon = new Icon(radarIconOptions);




for (Row row : radarTable) {

   double lat = row.getDouble("lat");

   double lon = row.getDouble("lon");


   // (long, lat)

   Coordinate radar_Coordinate = new Coordinate(lon,lat);

   MarkerFeature radar = new MarkerFeature(radar_Coordinate,radarIcon);

   map.getFeatureLayer().addFeature(radar);

   existingRadars.add(radar);

}
```

We then created polygon regions using the same boundaries from our model so everything would stay consistent. These regions were stored so we could update them later depending on which view was selected, instead of having to recreate them each time. We pulled them directly from our existing data so if regions were expanded or altered, the map would still update without needing to change the code. One note to add is that we made the decision to abbreviate Mid-Mississippi Valley to MMV on the map since the full name was too long to display clearly.

```
for (Row row : radarCoverage) {

   String regionName = row.getString("region_name");



   if(regionName.equals("Mid-Mississippi Valley")) {

       regionName = "MMV";

   }



   int beginLat = row.getInt("beginLat");

   int endLat = row.getInt("endLat");

   int beginLon = row.getInt("beginLon");

   int endLon = row.getInt("endLon");




   PolygonFeature region = new PolygonFeature(List.of(

           new Coordinate(beginLon,beginLat),

           new Coordinate(endLon,beginLat),

           new Coordinate(endLon,endLat),

           new Coordinate(beginLon,endLat),

           new Coordinate(beginLon,beginLat)

   ));

   region.setText(regionName);



   regionPolygons.put(regionName, region);   // save it here

   map.getFeatureLayer().addFeature(region);

}
```

We added a dropdown that lets the user switch between Risk View and Radar Need View. When the user selects a view, the map updates using a click listener that runs the correct section of code. In Risk View, the regions are colored based on final_risk values, and in Radar Need View, they are colored based on radar_need_score.

In addition, we took data from the National Weather Service and plotted all existing radar locations in the United States. Both Home View and Risk View feature the existing radars.

```
private void applyMapType() {

   String type = mapType.getValue();



   if (type == null) {

       return;

   }

   hideExistingRadars();

   hideProposedRadars();



   if (type.equals("Risk View")) {

       showExistingRadars();

       hideProposedRadars();

       applyRiskView();

   }



   else if (type.equals("Radar Need View")) {

       hideExistingRadars();

       addProposedRadars();

       applyRadarNeedView();

   }

}
```

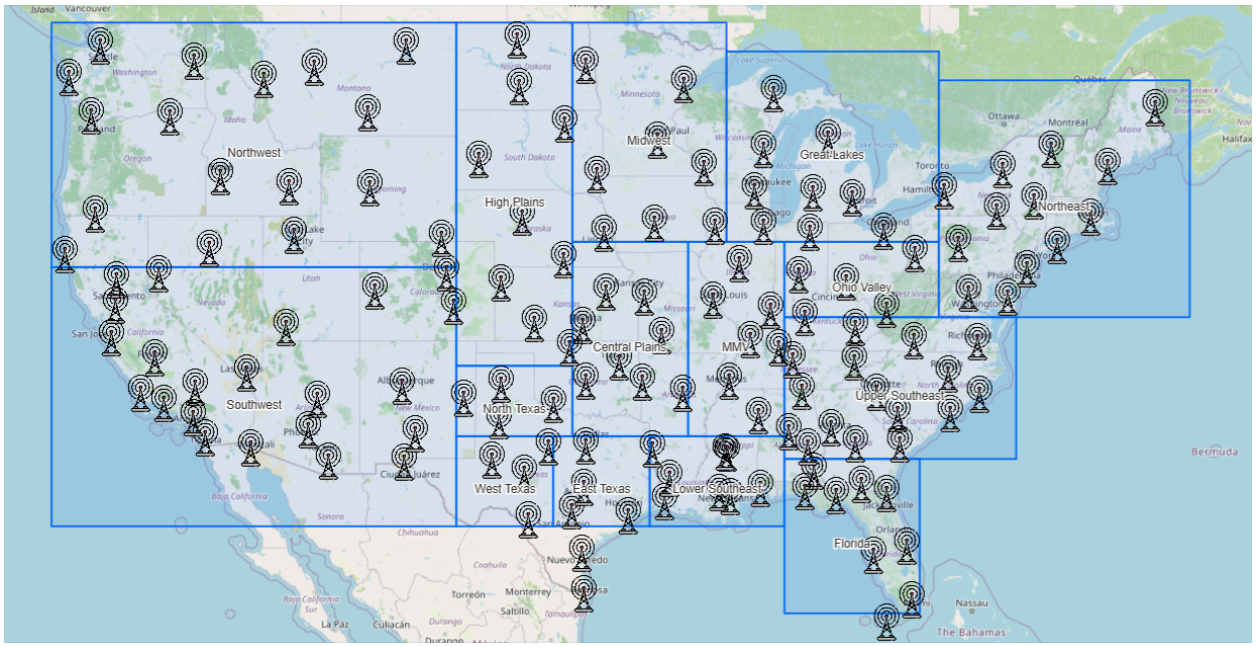
We also used RGBA color values instead of standard RGB. The “A” represents transparency, which allowed us to make the regions slightly see-through. This made it easier to still see the radar icons and map details underneath instead of completely covering them.

```
PolygonFeature polygon = regionPolygons.get(regionName);



if (polygon != null) {

   if (score < q1) {

       polygon.getStyle().setFill(new Fill("rgba(0,128,0,0.15)"));

       polygon.getStyle().setStroke(new Stroke("#66CC66", 2)); // Green

   } else if (score < q3) {

       polygon.getStyle().setFill(new Fill("rgba(255,165,0,0.15)"));

       polygon.getStyle().setStroke(new Stroke("#FFEB3B", 2)); // Yellow

   } else {

       polygon.getStyle().setFill(new Fill("rgba(255,0,0,0.15)"));

       polygon.getStyle().setStroke(new Stroke("#FF6666", 2)); // Red

   }

}
```

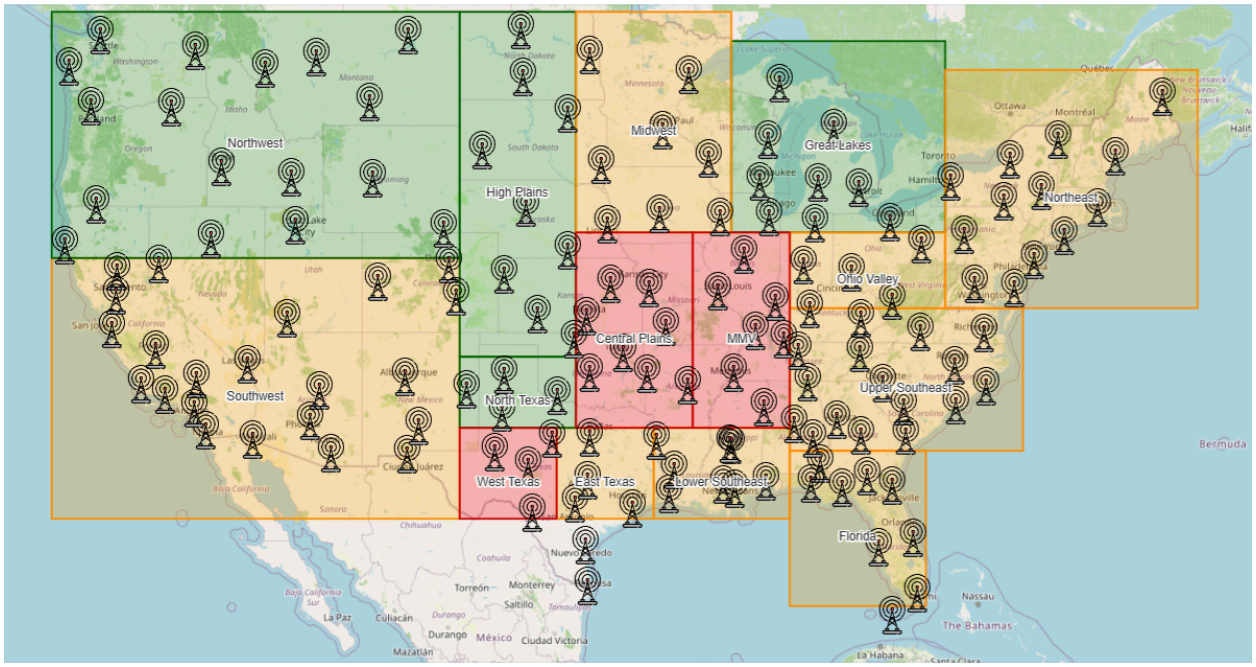
In Radar Need View, we plotted only the proposed radar locations using the coordinates from our model. We initially included both existing and proposed radar stations, but decided to focus on the proposed radars so they would stand out more clearly. We also included a legend that updates based on the selected view so users can understand what the colors represent without needing to look at the raw data.

Overall, this allowed us to bring both parts of our project together in one place and made it easier to compare tornado risk and radar coverage across regions.

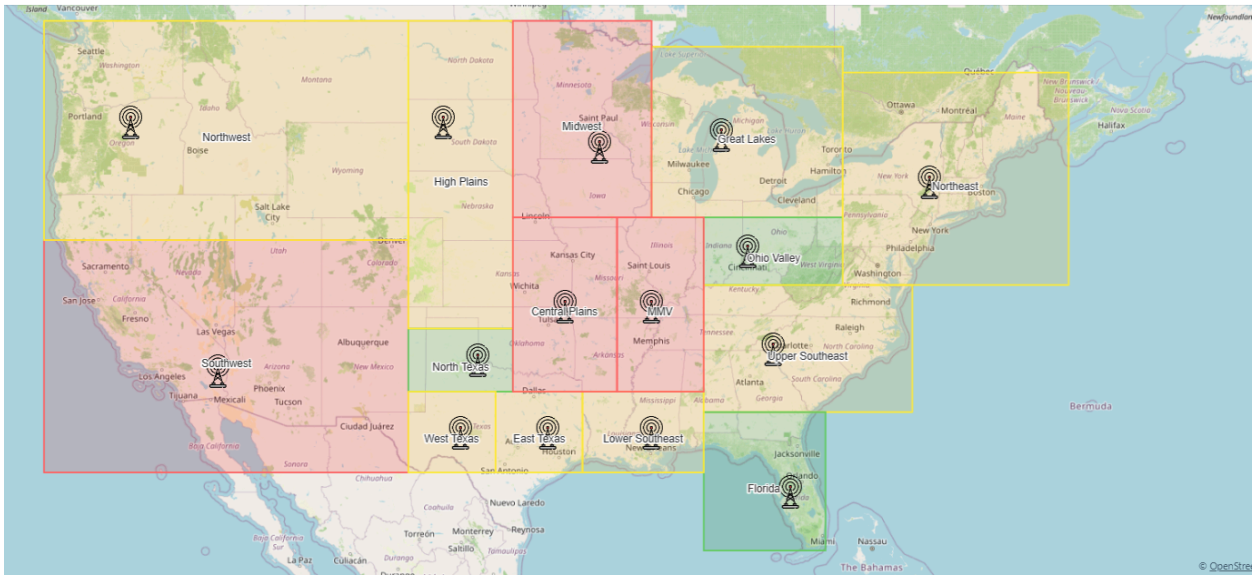
Home View



Risk Radar View



Radar Needs View



Citations:
https://chatgpt.com/share/69e97461-c2f4-83ea-9d41-9d33b601c991