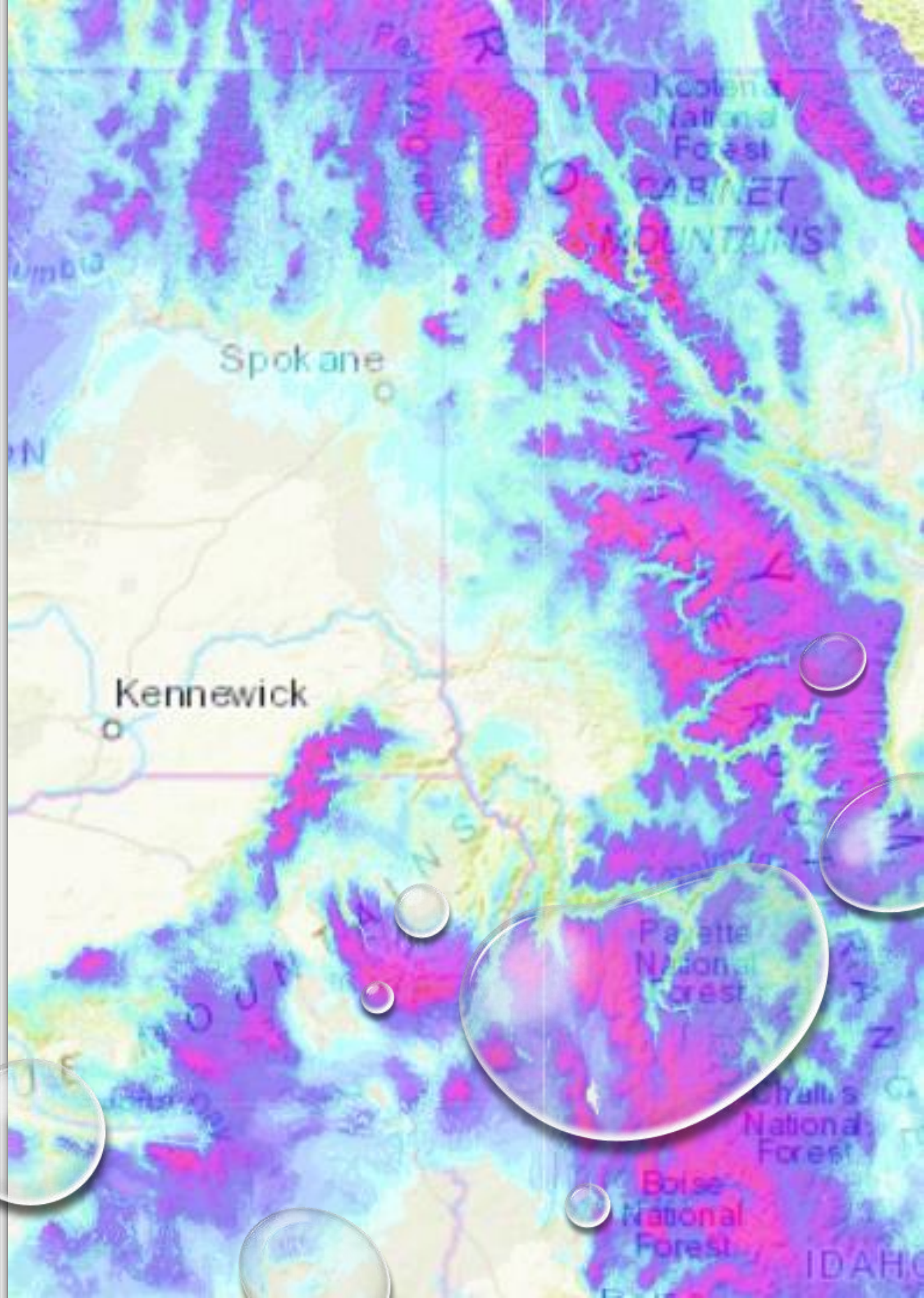
Snowpack Prediction Challenge

By Git Forked - Team 2

Sean Hodgson, Adam Caudle, Emily West, Shane Ganz, David Tran, Cole Wilson



Our Goal

- Create a spatio-temporal neural network trained on SNOTEL and meteorological data
- Would allow users to predict Snowpack trends in a given location throughout the year

Drive Behind the Snowpack challenge

- Deploying an application to see Snowpack trends that provides reliable and accurate insights into the water cycle of the Western US to inform decisions on water storage, irrigation, and agriculture.
- Monitor the effects of climate change on the local environment
- Gain experience with neural networking

Tools

- PyTorch: Train our Neural Network model
- GeoPandas: Used to handle our large geospatial data set
- Scikit-learn: Used to pre-process data before modeling

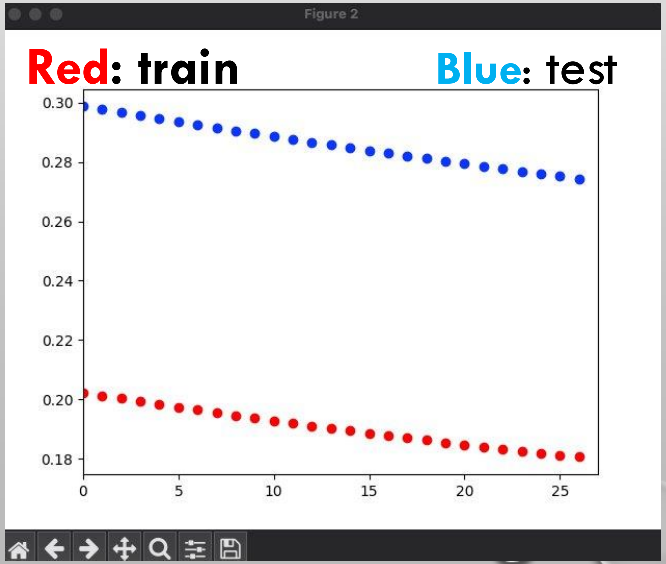
Data Cleanup & Research

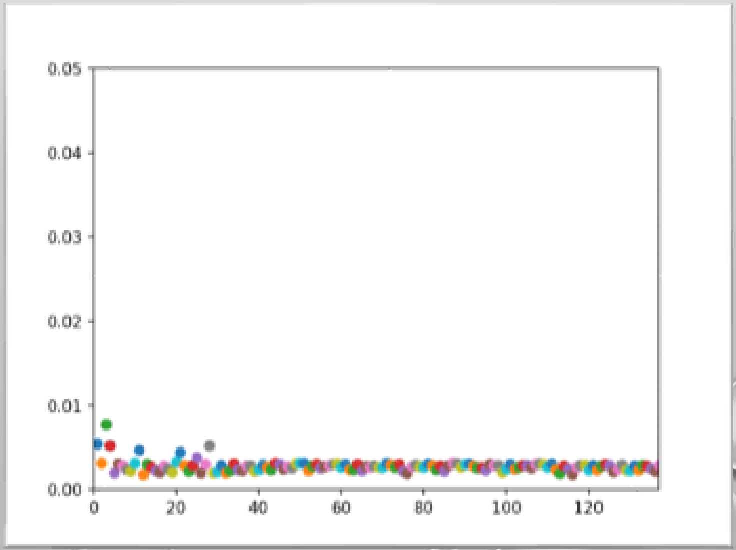
- We first used GeoPandas to format the data, before using a spatial join to combine the SNOTEL Locations with related meteorological data based on their coordinates and date
- Once combined, we identified empty values and used mean imputation to fill them with an estimate based on recent data from the same location
- Pulled site number data from the NRCS and added them to our database for reference

MODEL DESIGN

- LSTM MODEL FOR MAKING PREDICTIONS BASED ON PAST DATA
  - CUSTOM DATASET CALLED SWE DATASET CREATED TO LOAD WEATHER DATA
  - MAKES PREDICTIONS OF FUTURE SWE VALUES
  - NORMALIZE DATA IN SWE DATASET

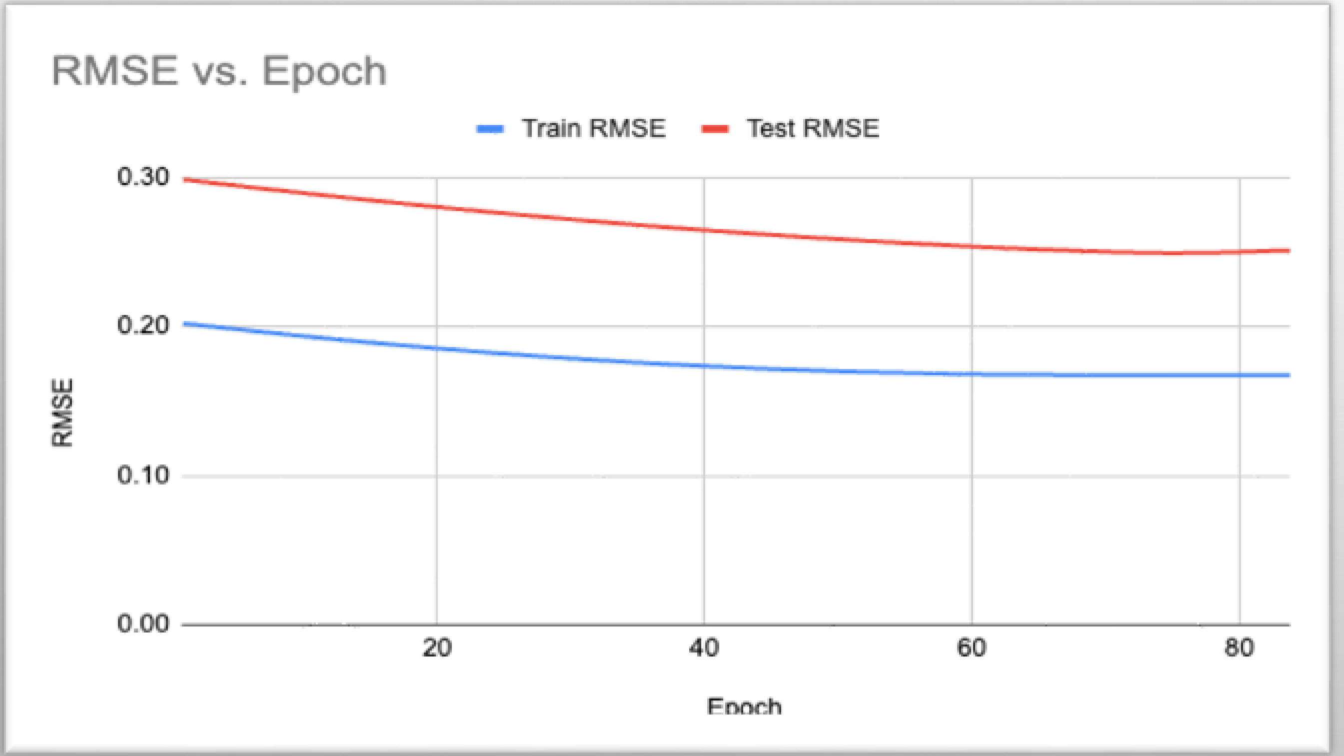
```python
class WeatherLSTM(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(WeatherLSTM, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Sequential(
            nn.Linear(hidden_size, self.hidden_size, bias=True),
            nn.ReLU(),
            nn.Linear(self.hidden_size, output_size, bias=True),
        )

    def forward(self, x, h0=None, c0=None):
        if h0 is None or c0 is None:
            h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)
            c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(x.device)

        out, (hn, cn) = self.lstm(x)  # lstm_out will have shape (batch_size, seq_len, hidden_size)
        # print(out)
        out = self.fc(out)
        # out = self.fc(out[-1, :])

        return out, hn, cn
```

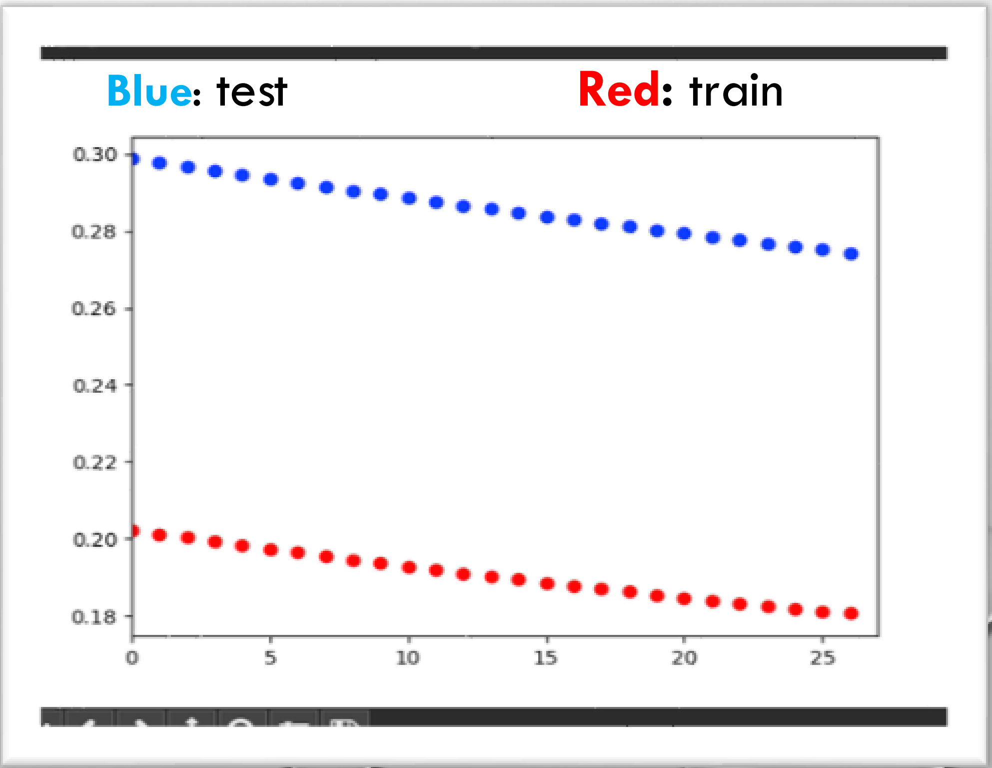




Model info

- Preprocessing: MinMaxScaler() from SciKitLearn: normalize each feature into a fixed range [0,1] to allow for faster training and predictions.
- Root Mean Square Error (RMSC)
  - Accuracy results inconclusive
  - Tested on Kamiak HPC

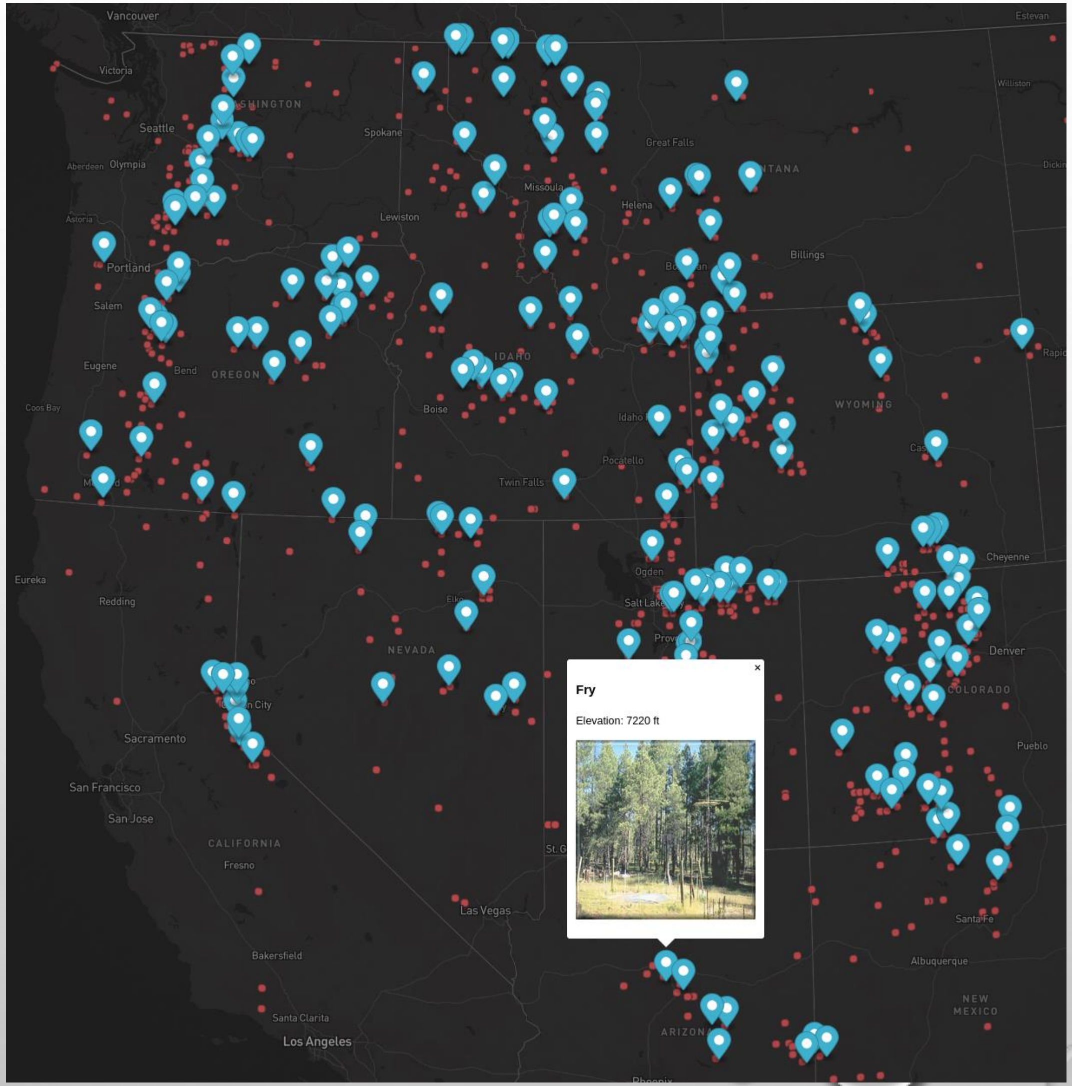




Results

- Data Pre-processing pipeline
- Trained Models
  - Mixed Accuracy results
- Pipeline needs fine-tuning then ready for app implementation

INTERACTIVE MAP



CONCLUSION