Model documentation and write-up

1. Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

   I work as a Senior Machine Learning Engineer for Uplight, Inc. where I build production machine learning systems that help orchestrate energy assets on the electric grid.

2. What motivated you to compete in this challenge?

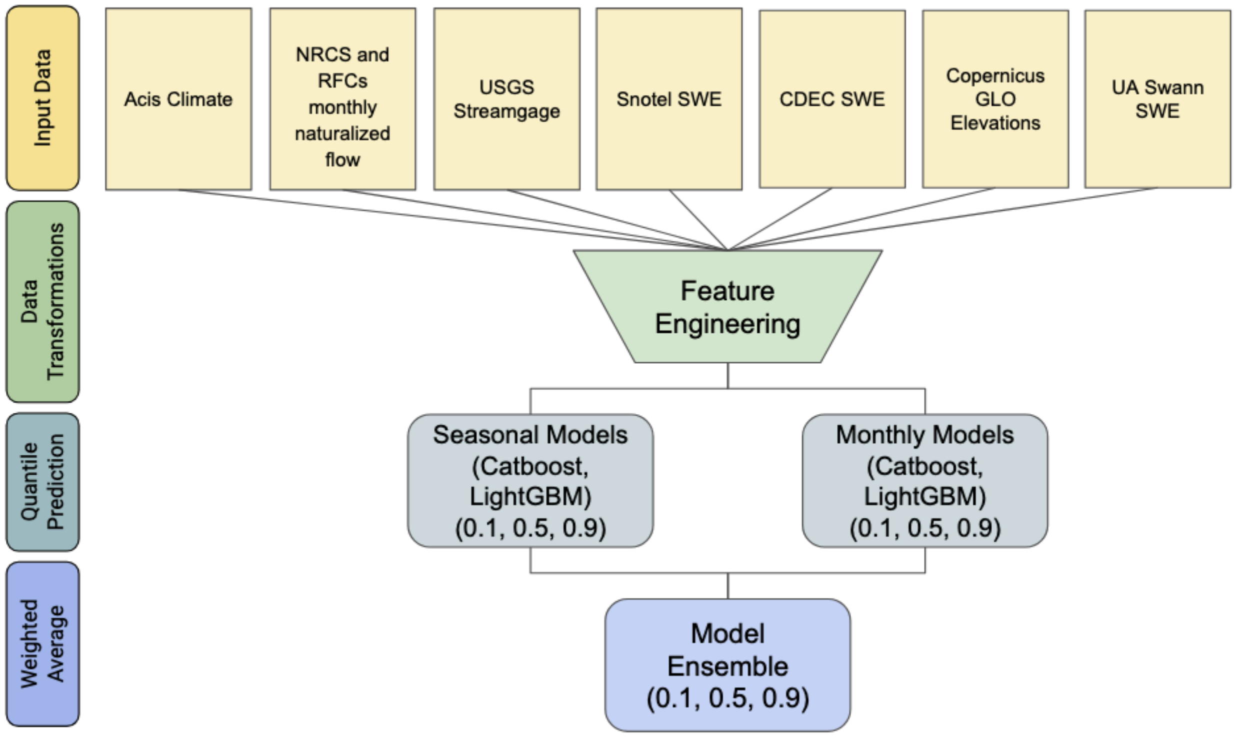
   Growing up and living in Denver, Colorado I have long been aware of the importance of seasonal streamflow to communities across the Western United States. I wanted to take this opportunity to learn more about the dynamics that shape streamflow from season to season and help give back to the community.

3. High level summary of your approach: what did you do and why?

   Using historical data from 26 different hydrologic sites we created an ensemble of gradient boosting models that provide a probabilistic forecast for the 0.10, 0.50, and 0.90 quantiles of cumulative, seasonal streamflow volume. There are two model architectures underlying the solution, both based on the Catboost implementation of gradient boosting on decision trees. The first model architecture predicts the quantiles of total cumulative streamflow volume in each season, for each site. The second model architecture predicts the quantiles of monthly cumulative streamflow volume within the season, for each site. The monthly predicted quantiles are then summed over the season to give the seasonal streamflow volume. The models use static features from each site including, the latitude, longitude, elevation, and categorical site id. The models also use dynamic features including, the day of the year, and features derived from localized precipitation, temperature, streamflow measurement from streamgages, and snow water equivalent measurements. The quantile predictions from each model are combined using a weighted average to minimize the prediction error given by the mean quantile loss across the three quantiles.

4. Do you have any useful charts, graphs, or visualizations from the process?

   Below is a chart that outlines the model architecture.


Input Data
Data Transformations
Quantile Prediction
Weighted Average
Acis Climate
NRCS and RFCs monthly naturalized flow
USGS Streamgage
Snotel SWE
CDEC SWE
Copernicus GLO Elevations
UA Swann SWE
Feature Engineering
Seasonal Models (Catboost, LightGBM) (0.1, 0.5, 0.9)
Monthly Models (Catboost, LightGBM) (0.1, 0.5, 0.9)
Model Ensemble (0.1, 0.5, 0.9)

5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

To get more out of the physical measurements (snow pack, streamflow, climate) that make up the features I calculated a z-score for the feature with respect to a geographic location and time of year. Using z-scores allowed for the aggregation of measurements across geographies and streamflow seasons and minimized the impact on a feature when a location measurement was missing data for part of the training period. Below is a section of code calculating the z-score ("deviation") for USGS streamflow measurements.

```
streamflow_deviation.to_csv(preprocessed_dir / 'train_streamflow.csv', index=False)
streamflow_deviation['datetime'] = pd.to_datetime(streamflow_deviation['datetime'])
streamflow_deviation['month_day'] = streamflow_deviation['datetime'].dt.strftime("%m%d")
grouped_streamflow = (
  streamflow_deviation.groupby(
    ['site_id', 'month_day']
  )['discharge_cfs_mean']
  .agg([np.mean, np.std]).reset_index()
)
grouped_streamflow.columns = [
    'site_id', 'month_day', 'discharge_mean', 'discharge_std'
  ]
grouped_streamflow.to_csv(preprocessed_dir / 'streamflow_parameters.csv', index=False)
streamflow_deviation = pd.merge(
    streamflow_deviation, grouped_streamflow, on=['site_id', 'month_day']
)
discharge = streamflow_deviation['discharge_cfs_mean']
discharge_mean = streamflow_deviation['discharge_mean']
discharge_std = streamflow_deviation['discharge_std']
streamflow_deviation['discharge_deviation'] = (discharge - discharge_mean) / discharge_std
```

Below is the code to ensemble the monthly and yearly models. It wasn't possible to train monthly models for every steam site because some of them did not have monthly data. But the monthly model was more accurate, especially in later months. Therefore the ensemble was set up to use the yearly prediction if there was no monthly prediction and use a weighted average of the monthly and yearly model with different weights for different quantiles and months. If the prediction was in the last month then the monthly prediction was used.

```
mth_pct_10 = 0.35
yr_pct_10 = 1.0 - mth_pct_10
mth_pct_50 = 0.45
yr_pct_50 = 1.0 - mth_pct_50
mth_pct_90 = 0.55
yr_pct_90 = 1.0 - mth_pct_90

def ensemble_monthly_yearly(rw):
    if pd.isna(rw['volume_10_mth']):
        rw['volume_10'] = rw['volume_10_yr']
    elif rw['month'] in [1, 2, 3, 4, 5, 6]:
        rw['volume_10'] = yr_pct_10 * rw['volume_10_yr'] + mth_pct_10 * rw['volume_10_mth']
    else:
        rw['volume_10'] = rw['volume_10_mth']

    if pd.isna(rw['volume_50_mth']):
        rw['volume_50'] = rw['volume_50_yr']
    elif rw['month'] in [1, 2, 3, 4, 5, 6]:
        rw['volume_50'] = yr_pct_50 * rw['volume_50_yr'] + mth_pct_50 * rw['volume_50_mth']
    else:
        rw['volume_50'] = rw['volume_50_mth']

    if pd.isna(rw['volume_90_mth']):
        rw['volume_90'] = rw['volume_90_yr']
    elif rw['month'] in [1, 2, 3, 4, 5, 6]:
        rw['volume_90'] = yr_pct_90 * rw['volume_90_yr'] + mth_pct_90 * rw['volume_90_mth']
    else:
        rw['volume_90'] = rw['volume_90_mth']

    return rw

submission_final = submission_final.apply(lambda rw: ensemble_monthly_yearly(rw), axis=1)
```

One of the tricks to getting the monthly model to work was rolling up the predictions to get a seasonal prediction without leaking naturalized monthly flow measurements. There are a couple of pieces of code required to make this work. First, I introduced a prediction month feature, in addition to a month feature, where the prediction month is the month we are predicting streamflow for and month is the month in which we are making the prediction. So for a given issue date there will be a prediction for each month in the streamflow season. If the month of the issue date is later than the prediction month then we use the observed value of streamflow for that month as the prediction, otherwise we use the predicted value. I then sum over all the predicted values for each issue date to get the total seasonal streamflow value for that issue date.

```
# For months occurring in the past use the monthly naturalized flow value if it isn't null
def use_observed_monthly_flow(rw):
  return (rw['pred_month'] < rw['month']) and (not pd.isna(rw['month_volume_observed']))

test_rw = pd.Series({'pred_month': 6, 'month': 7, 'month_volume_observed': np.nan})

print("test ", use_observed_monthly_flow(test_rw))

test_results['volume_10_mth'] = test_results.apply(
    lambda rw: rw['month_volume_observed']
    if use_observed_monthly_flow(rw)
    else rw['volume_10_mth'],
  axis=1
)
test_results['volume_50_mth'] = test_results.apply(
    lambda rw: rw['month_volume_observed']
    if use_observed_monthly_flow(rw)
    else rw['volume_50_mth'],
  axis=1
)
test_results['volume_90_mth'] = test_results.apply(
    lambda rw: rw['month_volume_observed']
    if use_observed_monthly_flow(rw)
    else rw['volume_90_mth'],
  axis=1
)

test_results[
  [
    'site_id', 'issue_date', 'pred_month', 'month_volume_observed', 'volume_10_mth',
    'volume_50_mth', 'volume_90_mth'
  ]
].groupby(['site_id', 'issue_date']).sum().reset_index()
grouped_result = test_results[
  [
    'site_id', 'issue_date', 'pred_month', 'month_volume_observed', 'volume_10_mth',
    'volume_50_mth', 'volume_90_mth'
  ]
].groupby(['site_id', 'issue_date']).sum().reset_index()
```

6. Please provide the machine specs and time you used to run your model.
   - CPU (model): 2.6 GHz 6-Core Intel Core i7
   - GPU (model or N/A): N/A
   - Memory (GB): 16 GB 2667 MHz DDR4
   - OS: Mac OS Ventura 13.6.4
   - Train duration: 2 hours to generate train features, 10 min. to train models.
   - Inference duration: 25 min. (including feature generation)

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

   Not that I can think of.

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

   I used sklearn to evaluate my model and I used matplotlib and seaborn for visualizing the data. Some analysis was done in Google Colab before migrating code back to the runtime environment.

9. How did you evaluate performance of the model other than the provided metric, if at all?

   I used cross-validation and compared models based on the quantile pinball loss metric.

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

    I tried an ensemble of linear models. I also tried incorporating different data sources, like SNODAS and the ONI teleconnections.

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

    My final submission report documents many additional techniques I experimented with after the forecasting round.

12. What simplifications could be made to run your solution faster without sacrificing significant accuracy?

    The most time-consuming portion of the code is the feature generation. Training the Catboost models and using them to predict is fast. I wouldn’t characterize it as a simplification but it is likely possible to speed up the feature engineering by pre-computing the rolling averages, outside of a Pandas apply function. Another possible speed up is using another, more performant dataframe library, like https://pola.rs/.