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.**

   Experienced Data Scientist specializing in time series and forecasting. Currently working in the IoT domain, focusing on elevating consumer experience and optimizing product reliability through data-driven insights and analytics. Previously worked in various tech companies in Indonesia.

2. **What motivated you to compete in this challenge?**

   I was motivated to compete in this challenge because of its unique setup. Unlike typical data science competitions, there's no predefined training dataset provided. This means participants must not only focus on modeling but also on finding the right data to be used. Additionally, the requirement to submit operational code adds another layer of complexity and practical application. I saw this as an exciting opportunity to test and expand my machine learning skills in a practical, real-world setting.

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

   Ensembles of LightGBM models with Tweedie loss for point forecast and quantile loss for 0.10 and 0.90 quantile forecast. Data sources used are SNOTEL/CDEC SWE, USGS/USBR observed flow, gridMET PDSI, UA/SWANN SWE, CDEC SWE, ERA5-Land, and seasonal forecast products from ECMWF SEAS51. Since the dataset is small based on the number of available training years for each site and issue date, synthetic data generation is applied to increase the training sample size by 5x, which significantly improves forecast skill, prediction interval reliability, and generalizability.

   Compared to the model in the Hindcast Stage, forecast skill has improved by ~5 KAF and it is attributed to additional data sources used and better estimates of snowpack in some regions, especially in Sierra Nevada and Cascades region.

   In addition, to handle data dependency and availability in live forecasts, model ensemble members are designed with varying dependencies. The ensemble consists of nine model variants with different combinations of data source input, where the model variant with the least dependency uses SNOTEL/CDEC SWE features (equivalent to the model in the Hindcast Stage) and the model variant with the most dependency uses all data sources/features.

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

   For me, the most important chart or graph is related to the post-evaluation of the model. It gives us an idea of whether the model works as expected, which parts are harder to predict, and the potential for room for improvement. All of those charts can be accessed in the appendix section from model report.

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

   The dataset is small based on the number of available training years for each site and issue date. Non-linear models tend to overfit with small data. to mitigate the issue, synthetic data generation is applied to the training dataset which significantly improves forecast skill, prediction interval reliability, and generalizability
   src/features/base.py generate_synthetic_data function

```
def generate_synthetic_data(
    df,
    cols=["volume"],
    n_synthetic=4,
    scale_factor=[0.5, 1.5],
    filter_condition='cat == "train"',
    seed=0,
):
    df = df.copy()
    train_year = (
        df.query(filter_condition)
        .year.value_counts()
        .to_frame("count")
        .reset_index()
        .rename(columns={"index": "year"})
    )
    np.random.seed(seed)
    scale_factor = np.random.uniform(
        low=scale_factor[0], high=scale_factor[1], size=(len(train_year),
n_synthetic)
    )
    scale_factor = pd.DataFrame(scale_factor)
    scale_factor.columns = [f"f{factor}" for factor in range(n_synthetic)]
    train_year = pd.concat([train_year, pd.DataFrame(scale_factor)], axis=1)
    _df_synthetic = []
    for _, row in train_year.iterrows():
        selected_year = row["year"]
        # print(selected_year)
        for factor in range(n_synthetic):
            selected_factor = row[f"f{factor}"]
            _df = df.query("year == @selected_year").reset_index(drop=True)
            _df.loc[:, cols] = _df.loc[:, cols] * selected_factor
            _df_synthetic.append(_df)
    _df_synthetic = pd.concat(_df_synthetic)

    return _df_synthetic, train_year
```

   Applying target diff preprocessing significantly improves performance for issue dates within the seasonal month target
   src/features/base.py generate_target_diff function

```
def generate_target_diff(df_monthly, df_meta):
    """
    Generate "diff" column which is the known previous month cumulative naturalized
flow.
    This column will be used to calculate partial gt in month 5-7 in most of the
sites
    """
```

```
    df_target_diff = (
        pd.merge(
            df_monthly.assign(month_tf=lambda x: x["month"] + 1),
            df_meta[["site_id", "season_start_month", "season_end_month"]],
        )
        .query("season_start_month+1 <= month_tf <= 7")
        .assign(diff=lambda x: x.groupby(["site_id",
"year"])["volume"].transform("cumsum"))[
            ["site_id", "year", "month_tf", "diff"]
        ]
    )

    return df_target_diff
```

Flexibly evaluate various metrics based on different groupings
src/utils.py agg_error_metrics, eval_agg and eval_all functions

```
def eval_agg(pred_df, grouper=["year"], is_include_mean_std=False):
    eval_agg_df = pred_df.groupby(grouper).apply(agg_error_metrics)
    if "site_id" in grouper:
        eval_agg_df = eval_agg_df.sort_values("actual_mean", ascending=False)
    if (is_include_mean_std == True) & (len(eval_agg_df) > 1):
        eval_agg_df = pd.concat(
            [
                eval_agg_df,

eval_agg_df.mean().to_frame().T.assign(a="mean").set_index("a").rename_axis(None,
axis=0),

eval_agg_df.std().to_frame().T.assign(a="std").set_index("a").rename_axis(None,
axis=0),
            ],
            axis=0,
        )

    return eval_agg_df
```

6. **Please provide the machine specs and time you used to run your model.**

- CPU (model): Core i5-1135G7
- GPU (model or N/A): N/A
- Memory (GB): 8GB
- OS: Windows
- Training data preprocessing for all years 1981-2023: 3-4 hours (not including data download time, e.g. retrieving data from CDS API can take ~1 hour for a single date or month)
- Training duration: ~3 hours for all 720 models (20-fold years x 9 model variants x 4 losses)
- Inference duration: less than 3 minutes for a single issue date and 26 sites (not including data download time)

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.)?**

In hindcast setting, we have the ideal condition where all the data is available. In operational setting, it’s expected that we will have degraded forecast skill performance depending on data availability. Based on various simulations, forecast skill can degrade by 2-5 KAF. For example, there are ~3 KAF degradation of forecast skill for water year 2024 in the Forecast Stage.

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

   I use plotly and matplotlib to quickly explore the data and do post evaluation of the model

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

   I use other evaluation metrics such as RMSE, R2 and bias for additional internal validation. Later on, I also include relative metrics to get an idea of the model performance for specific locations and situations

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

    - Training with teleconnection indices -> higher uncertainty and instability even though it has the potential to improve long lead time forecast in January
    - Training with RCC-ACIS PRISM precipitation and temperature -> no significant improvement in forecast skill
    - Training with supplemental NRCS sites -> improvement in forecast skill but require 4x training time

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?

    - Experiment with sub-model, semi-distributed and bottom-up forecasting settings (daily and monthly forecasting, large basin divided into sub-basins)
    - Better synthetic generation approach based on simulation under physical constraints
    - Use different ensemble techniques to get more optimal result for different locations and conditions (weighted ensemble, 2nd stage model)

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

    Only use several ensemble members and adjust based on dependency and processing complexity. For example, we can exclude models with ERA5-Land data because forecast skill improvement is minimal based on CV score (~1 KAF) and processing time is longer especially if we include data download time from CDS API