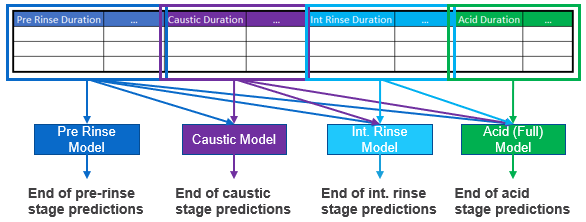
# III. Model documentation and write-up

You can respond to these questions either in an e-mail or as an attached file (any common document format is acceptable such as plain text, PDF, DOCX, etc.) **Please number your responses.**

1. Who are you (mini-bio) and what do you do professionally?  
     
   **I am currently a Data Scientist at Opex Analytics, an advanced analytics consulting firm based out of Chicago. Prior to joining Opex, I completed my Master of Science in Analytics from Northwestern University in 2017, and before that I worked in credit risk analysis for an American department store.  
     
   I am passionate about combining the technical aspects of data science, such as mathematics and programming, with business knowledge and practical considerations to create useful and impactful data-driven solutions. I also enjoy participating in data science competitions, which provide excellent opportunities to learn new machine learning techniques and apply them to real-world problems.**
2. High level summary of your approach: what did you do and why?  
     
   **I built four separate models to reflect the ends of the four different stages at which we were asked to make predictions: pre rinse, caustic, intermediate rinse, and acid. The pre rinse model was used to make predictions on the 10% of the data that was censored at the end of the pre rinse step, the caustic model on the 30% censored at the end of the caustic step, and so on. This ensured that all of the data available at the time of the prediction was used to the fullest extent.  
     
   Careful feature engineering played a large role in and making this multi-model approach possible. Each feature was generated at a phase-level or lower and “unmelted” to produce the final features at a process-level. Mid-cleaning predictions were simulated by excluding the appropriate predictors from each model. Below is a diagram that uses the ‘duration’ feature to illustrate the general principle:** **There were important practical considerations – namely, being able to explain the relationships between the predictors and the response in an easily understandable manner – that led me to choose simpler modeling approaches. For example, I felt that combining multiple techniques through stacking (e.g. neural networks + LGBM) would result in predictions that would be very difficult to explain, so I went with pure LightGBM for my models. For similar reasons centered around interpretability, I decided against dimensionality reduction techniques such as PCA and autoencoders.  
     
   To explain the output of my models, I used Shapley Additive Explanations (SHAP), a well-known technique for interpreting the output of complex models. SHAP is optimized for use with LightGBM and produced results fairly quickly. The summary plot provided an easily interpretable view of variable importance, and dependence plots clearly illustrated the effect of individual predictors (or pairs of predictors if interactions were highlighted) on each prediction.**
3. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.  
     
   **Note: I’ve removed some of the comments in my actual code for the sake of readability in the document. The original comments are all present in the source code.**

logger.info('Training ' + model\_type + ' model...')  
gbm\_train = lgb.train(params,  
 modeling\_data['train'],  
 num\_boost\_round=5000,   
 valid\_sets=modeling\_data['eval'],  
 verbose\_eval=False,  
 early\_stopping\_rounds=150,  
 keep\_training\_booster=True  
 )

**This is the code that builds the models during training and evaluates them on the validation set. A noteworthy point is the use of early stopping in conjunction with a low learning rate (set in a different script) and a large number of boosting rounds, which greatly reduced the amount of parameter tuning involved.   
  
Another useful point that may be somewhat less well-known is the “keep\_training\_booster” argument. Setting it to True allowed me to pull the validation set predictions made during the model building process directly from the resulting lightgbm.Booster object (as opposed to having to run a predict function on the validation set again after the model is trained).**

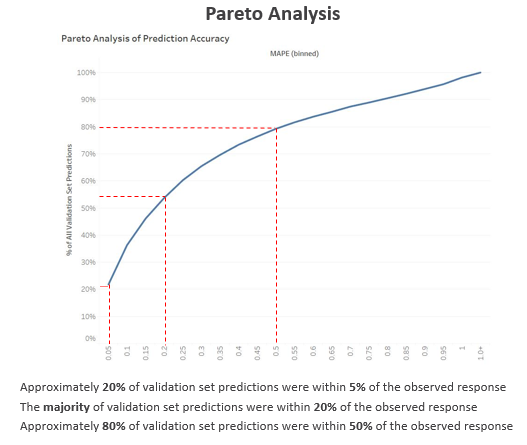
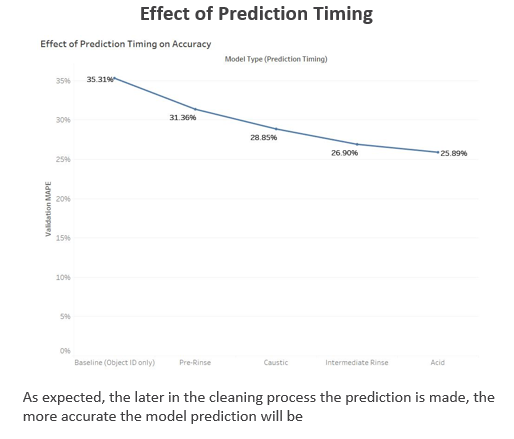
df['return\_phase'] = df.phase + '\_' + np.where(df.return\_drain == True, 'dr',  
 np.where(df.return\_caustic == True, 'cs',  
 np.where(df.return\_acid == True, 'ac',  
 np.where(df.return\_recovery\_water == True, 'rw', 'none'))))  
  
if return\_phase\_defs is None:  
 return\_phases = list(  
 df.return\_phase.value\_counts()[df.return\_phase.value\_counts() > 300000].reset\_index()['index'])  
else:  
 return\_phases = return\_phase\_defs  
df['return\_phase'] = np.where(df.return\_phase.isin(return\_phases), df.return\_phase, 'other')

**This code is used to calculate the “return-phase” for each row in the raw data. This is an important subdivision of a phase that also accounts for the return valve that is open. Infrequent return-phases (defined as occurring less than 300k times in the raw training data) are bucketed as “other”. Many features are calculated at the return-phase and supply-phase levels, which improved model performance noticeably over simply calculating them at the phase-level.**

df['return\_flow'] = np.maximum(0, df.return\_flow)  
df['supply\_flow'] = np.maximum(0, df.supply\_flow)  
df['return\_residue'] = df.return\_flow \* df.return\_turbidity  
df['phase\_elapse\_end'] = (  
 df.groupby(['process\_id', 'phase']).timestamp.transform('max') - df.timestamp).dt.seconds  
df['phase\_elapse\_start'] = (  
 df.timestamp - df.groupby(['process\_id', 'return\_phase']).timestamp.transform('min')).dt.seconds  
df['end\_turb'] = df.return\_turbidity \* (df.phase\_elapse\_end <= 40   
df['end\_residue'] = df.return\_residue \* (df.phase\_elapse\_end <= 40)

**This code engineers some simple but very important features on the raw data at a process-timestamp level. The concepts of “end turbidity” and “end residue” were particularly crucial to model performance, as they zoomed in on the amount of particulate flowing out at the end of the return-phase (last 80 seconds) rather than the entire return-phase. The results from SHAP suggested that these end-of-return-phase features often had more impact on the test set predictions than the corresponding entire-return-phase features. Ultimatley, using both end-of-return-phase and entire-return-phase features together produced the best model results.**

1. What are some other things you tried that didn’t necessarily make it into the final workflow (quick overview)?  
     
   **I tried several types of features that did not make it into the final workflow because they did not appear to improve model performance. Noteworthy examples include:  
     
   - Features that attempted to measure the variability of a predictor of time such as the standard deviation of flows/temperatures/etc. over a given phase  
   - Features derived from tank level and temperature  
   - Temporal features such day of week, hour of day, etc.  
     
   I also tried other methods for addressing outliers besides excluding them, such as clipping (i.e. setting them to the value of the highest non-outlier value). These also did not help improve performance.**
2. Did you use any tools for data preparation or exploratory data analysis that aren’t listed in your code submission?  
     
   **I used Tableau during my exploratory data analysis on the raw data.**
3. How did you evaluate performance of the model other than the provided metric, if at all?  
     
   **I did not evaluate the performance of the model on any metric other than the provided one.   
     
   However, I did highlight an important limitation of the provided metric in my modeling report: that minimizing MAPE (or variants thereof, such as the one used in this competition) will usually lead to models that systematically under-predict the response. This is an inherent limitation of MAPE due to the asymmetric costs of over- and under-prediction (under-prediction can never incur a cost of more than 100%, whereas the costs that can result from over-prediction are unbounded).   
     
   This can have important practical consequences, and I recommended as a future/next step to consider trying models with other metrics that do not have this weakness to address this potential issue.**
4. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?  
    **The main step that requires a bit of extra effort is the installation of LightGBM. I had to modify LightGBM’s source code to implement my custom objective function due to limitations of the Python interface, so LightGBM cannot just be installed with something like “conda install”. At a high level, the process is as follows:  
     
   1. Clone the git repo for LightGBM to the local machine.  
   2. Make the appropriate changes to the source code (not very involved, just changing a single number in a few different lines of code)  
   3. Run setup.py to install LightGBM with the modified source code.  
     
   More details are provided in the readme. Aside from this, as long as the directory structure is properly set up and the source data files are in the right format (train\_values and test\_values need to be in .pkl format and have all of the final\_rinse rows removed), the code should be easy to run.**
5. Do you have any useful charts, graphs, or visualizations from the process?

**I had several useful visualizations that I included in my modeling report. Two are included below as examples; please refer to the modeling report to see the others.**

1. 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?  
     
   **One area I would dedicate significantly more time to would be outlier detection. My outlier detection approach was not very sophisticated and had a lot of room for improvement. I learned about several interesting techniques such as isolation forests and local outlier factors after the competition was over, and I would be interested in seeing if these techniques revealed other outliers that my simple approach missed.  
     
   As I mentioned earlier, I would also explore other evaluation metrics to see if I could produce models without the systematic under-bias that my best model for this competition exhibited. In a similar vein, there are other responses that we could model from the given data that could provide additional actionable insights, such as how many seconds into the final phase pass before the return flow is “clean” (“clean” being defined in some rigorous manner, such as return flow < some threshold for at least 20 seconds).  
     
   Given the overwhelming importance of object\_id in all of my models, I felt that more features related to the objects themselves would be been helpful. Examples would include the category of object (i.e. a granulator, mixer, etc.), the vendor, and the company that utilizes the object. These details may have helped us make better predictions for objects that had relatively little historical data, and should increase how well the model predictions generalize to new objects that have no historical data.**