Prize recipient solution documentation guide

This document details the information required to submit your Phase 1 solution materials.

I. **Code submission and result reproducibility**
You package up and send us your code, documentation of dependencies, and any other assets you used. We review your package and make sure that it works and that we can fully reproduce the workflow from raw data to a submission comparable to your best submission. See the Competition prize winner submission template for more guidance on packing up and organizing your code (however, please note that the template's write-up questions differ from the questions for this challenge).

II. **Basic information for winner announcement**
Provide your preferred information for use in announcing the winners of the competition.

III. **Model documentation and write-up**
You write up answers to our questionnaire, providing important context and documentation so that the beneficiary and the community get the most out of your work.

I. Code submission and result reproducibility

You will need to submit a compressed archive of your code. You don't need to include the raw data we provided, but everything else should be included and organized. If the files are too large to be e-mailed, a Google Drive or Dropbox share (or other comparable method of transferring data) works.

Note: *Please follow these instructions carefully*. The spirit and purpose of the competition (and the reason for offering prizes) is to give our beneficiary organizations the best possible solution *along with working code they can actually use*. In accordance with the competition rules, if we can't get your code working and reproduce your results with a reasonable effort, or if your entry is too disorganized to be practically usable, then your entry may be disqualified!

The overall concept is to **set up this archive as if it were a finished open source project**, with clear instructions, dependencies and requirements identified, and code structured logically with an obvious point of entry. Check out the competition prize winner README template to get started. We also have a data science project template which may be helpful.

At a minimum, your solution must include **an extremely clear README** that details all of the steps necessary to get to your submission from a fresh system with no dependencies (e.g., a brand new Linux, Mac OS X, or Windows installation depending on what environment you choose to develop under) and no other data aside from the raw data you downloaded from us.

This will probably entail the following:

- Necessary tools and requirements (e.g. "You must install Word2Vec 0.1c" or "Install the required Python packages in `requirements.txt`").
    - **All requirements should be clearly documented**, for instance in either a requirements.txt file with versions specified or environment.yml file.
- The series of commands, in order, that would get a reasonably experienced and savvy user from your code to a finished submission.
    - **Ideally, you will have a main point of entry to your code** such as an executable script that runs all steps of the pipeline in a deterministic fashion. A well-constructed Jupyter notebook or R script meets this standard.
    - **The next best thing is a list of specific, manual steps** outlining what to do. For example, "First, open Word2Vec and set these parameters. [...] Take the output file and run the script `src/make_preds.R` with the following parameters [...]". *(The limitations of this approach should be clear to any experienced data scientist!)*
- **Make sure to provide access to all trained model weights necessary to generate predictions from new data samples** without needing to retrain your model from scratch. Note that model weights can be contained in your archive or shared via a cloud storage service. The solution should provide clear instructions to perform inference on a new data point, whether or not it is included in the test set.
- Any other instructions necessary to end up with your winning submission file (or comparable — we understand that certain parts of model fitting are stochastic and won't result in exactly the same parameters every time).

II. Basic information for winner announcement

Please provide your preferred information for use in announcing the winners of the competition.

- **Name**: Priya Dhond
- **Hometown**: Jersey City, NJ
- **A recent picture of yourself or digital avatar**:

- **Name**: Alejandro Sáez
- **Hometown**: Barcelona, Spain
- **A recent picture of yourself or digital avatar**:

III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

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.**

   - Priya Dhond: Data Scientist specialized in the streaming services industry. Currently pursuing graduate studies at NYU's center for data science.

   - Alejandro Sáez: Data Scientist with consulting experience in the banking and energy industries currently pursuing graduate studies at NYU's center for data science.

2. **What motivated you to compete in this challenge?**
   What motivated us to compete in this challenge was the possibility to interact with real-world data from NASA, improve our modelling skills, and be able to contribute to a pressing problem with significant potential impact for the well-functioning of the US aerial transportation network.

3. **High level summary of your approach: what did you do and why?**
   We approached the problem following the canonical data science workflow, that is: (1) Cleansed and documented the raw data sources, (2) Extracted 298 explanatory features from past periods that are potentially predictive of pushback time, (3) Built 4 families of gradient boosting models leveraging the Catboost implementation, (4) Fine-tuned the parameters, (5) Evaluated the goodness-of-fit of our methods individually and as an ensemble, and (6) Built the necessary functionalities to serve live predictions in the NASA runtime environment.

4. **How did knowing that you might need to federate this model change your approach to this challenge problem?**
   Being conscious that data privacy is a pressing issue in this context, and that a federated learning version of the model would need to be retrained, made our approach be modular in the way we treated features. That is, we did not leak information that is airline-specific to other flight since this information might not be available in production.

5. **Do you have any useful charts, graphs, or visualizations from the process?**
   The README.md file contains all the details of our implementation accompanied with visualizations.

6. **Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.**
   The 3 most impactful parts of our code are the following:

   - **Feature extraction**: Under /training/src/pushback_nasa/feature_extraction one can find both the nodes and the pipelines associated with the feature extraction. This is the core of the approach where in the nodes.py

```
def create_pipeline(**kwargs) -> Pipeline:
    airports = ['katl', 'kclt', 'kden', 'kdfw', 'kjfk', 'kmem',
'kmia', 'kord', 'kphx', 'ksea']

    perimeter = [node(func=compute_perimeter,
                      inputs=[f"raw_{airport}_labels",
f"sub_format", f"params:{airport}"],
                      outputs=f"perimeter_{airport}",
                      name=f"computing_perimeter_{airport}") for
airport in airports]

    mfs_features = [node(func=extract_mfs_features,
                         inputs=[f"perimeter_{airport}",
f"raw_{airport}_mfs"],
                         outputs=f"features_{airport}_mfs",
                         name=f"extract_{airport}_mfs_features")
for airport in airports]

    etd_features = [node(func=extract_etd_features,
                         inputs=[f"perimeter_{airport}",
f"raw_{airport}_etd"],
                         outputs=f"features_{airport}_etd",
                         name=f"extract_{airport}_etd_features")
for airport in airports]

    moment_features = [node(func=extract_moment_features,
                            inputs=f"perimeter_{airport}",
                            outputs=f"features_{airport}_moment",

name=f"extract_{airport}_mom_features") for airport in airports]

    config_features = [node(func=extract_config_features,
                            inputs=[f"raw_{airport}_config",
"params:start_time", "params:end_time"],
                            outputs=f"features_{airport}_config",

name=f"extract_{airport}_config_features") for airport in
airports]

    weather_features = [node(func=extract_weather_features,
                             inputs=[f"raw_{airport}_lamp",
"params:start_time", "params:end_time"],
                             outputs=f"features_{airport}_lamp",

name=f"extract_{airport}_weather_features") for airport in
airports]

    runway_features = [node(func=extract_runway_features,
```

```
                            inputs=[f"raw_{airport}_runways",
"params:start_time", "params:end_time"],
                            outputs=f"features_{airport}_runways",
                            name=f"extract_{airport}_runway"
                            ) for airport in airports]

    standtime_features = [node(func=extract_standtime_features,
                                inputs=[f"perimeter_{airport}",
f"raw_{airport}_standtimes", f"raw_{airport}_etd"],

outputs=f"features_{airport}_standtimes",

name=f"extract_{airport}_standtimes"
                                ) for airport in airports]

    tfm_features = [node(func=extract_tfm_features,
                         inputs=[f"raw_{airport}_tfm"],
                         outputs=f"features_{airport}_tfm",
                         name=f"extract_{airport}_tfm"
                         ) for airport in airports]

    tbfm_features = [node(func=extract_tbfm_features,
                          inputs=[f"raw_{airport}_tbfm"],
                          outputs=f"features_{airport}_tbfm",
                          name=f"extract_{airport}_tbfm"
                          ) for airport in airports]

    return pipeline(perimeter + mfs_features + etd_features +
                    moment_features + config_features +
weather_features +
                    runway_features + standtime_features)
```

- **Modeling**: Functionalities to train the model. Given a data set x containing the features created earlier, a target name, and an end train date, trains and stores a catboost model.

```
def train_catboost_diff(x: pd.DataFrame, target_name: str,
end_train: str):
    x = x[(x['timestamp'] > '2020-11-02') & (x[target_name] != 0)]

    feat_names = list(x.columns)
    feat_names.remove("gufi")
    feat_names.remove("timestamp")
    feat_names.remove(target_name)

    cat_features = [i for i in range(len(feat_names)) if '_cat_'
in feat_names[i]]

    x_train = x[x["timestamp"] < end_train][feat_names]
    x_val = x[x["timestamp"] >= end_train][feat_names]

    y_train = x[x["timestamp"] < end_train][target_name] -
x_train['etd_time_till_est_dep']
    y_val = x[x["timestamp"] >= end_train][target_name] -
x_val['etd_time_till_est_dep']

    print("#" * 20 + ' ' * 5 + "training with ", x_train.shape, '
```

```
' * 5 + '#' * 20)
    print("#" * 20 + ' ' * 5 + "validating with ", x_val.shape, '
' * 5 + '#' * 20)

    best_score = 1e9
    best_model = None

    for depth in [7]:
        for eta in [0.01]:

            model = CatBoostRegressor(eta=eta,
                                      depth=depth,
                                      rsm=1,
                                      subsample=0.8,
                                      max_leaves=21,
                                      l2_leaf_reg=3,
                                      min_data_in_leaf=5000,
                                      n_estimators=20000,
                                      task_type='CPU',
                                      thread_count=-1,
                                      grow_policy='Lossguide',
                                      has_time=True,
                                      random_seed=4,
                                      loss_function='MAE',
                                      boosting_type='Plain',
                                      max_ctr_complexity=12,
                                      bootstrap_type='Bernoulli')

            model.fit(x_train, y_train,
                      eval_set=(x_val, y_val),
                      use_best_model=True,
                      verbose=200,
                      cat_features=cat_features,
                      early_stopping_rounds=60)

            score =
abs(pd.Series(model.predict(x_val)).astype(int).clip(1,
299).values -
                          y_val.values).mean()
            print("Error in test:", score)

            if score < best_score:
                print('BEST SCORE', score)
                best_score = score
                best_model = model

    # Retrieve feature contributions
    feat_imps = pd.DataFrame({"features": feat_names, "imp":
best_model.feature_importances_})
    feat_imps["imp"] = feat_imps["imp"] / feat_imps["imp"].sum()
    feat_imps.sort_values("imp", ascending=False, inplace=True)

    return best_model, best_model, feat_imps
```

- **Inference / prediction serving**: Under /submission one can find the necessary code in solution.py to execute the inference pipeline end-to-end in the containerized NASA environment

```
def predict(config: pd.DataFrame, etd: pd.DataFrame,
first_position: pd.DataFrame,
            lamp: pd.DataFrame, mfs: pd.DataFrame, runways:
pd.DataFrame, standtimes: pd.DataFrame,
            tbfm: pd.DataFrame, tfm: pd.DataFrame, airport: str,
prediction_time: pd.Timestamp,
            partial_submission_format: pd.DataFrame, model: Any,
solution_directory: Path) -> pd.DataFrame:

    try:
        predictions = catboost_predictions(prediction_time,
partial_submission_format,
                                           mfs, config, etd, lamp,
runways,
                                           standtimes,
model[airport], airport)
    except:
        try:
            logger.info(f"Defaulting to baseline predictions")
            predictions =
baseline_predictions(partial_submission_format, etd)
        except:
            logger.info(f"Defaulting to manual predictions")
            predictions = partial_submission_format
            predictions["minutes_until_pushback"] = 30

    return predictions
```

7. **Please provide the machine specs and time you used to run your model.**
   - **CPU (model):** Intel Core i7 CPU @1.9GHz
   - **GPU (model or N/A):** N/A
   - **Memory (GB):** 32Gb of local RAM memory
   - **OS**: Windows 10
   - **Train duration**: 5h for each airport and model, totaling XXX after parameter fine-tuning
   - **Inference duration**: 7h for the 1800 timestamps in the held-out test set

8. **Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?**
   Following the usage instructions in the README.md file should suffice to reproduce the training and inferencing pipeline

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

No, the code submission is self-contained in terms of python packages used and scripts utilized

10. **How did you evaluate performance of the model other than the provided metric, if at all?**
    MAE was used to assess the goodness of fit of the models, both at an entire dataset level as well as broken down by month-of-year, and airport.

11. **What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?**
    Several additional approaches were attempted but deprioritized or entirely eliminated from the final workflow due to lack of positive impact on the validation MAE. Among them range the following:
    - **TBFM/TFM features**: These did not improve the performance of the models
    - **Time based cross validation**: Averaging across folds based on time splits did not improve the performance of the models either