cs4641

Missed Flight Connection Risk

1) Introduction/Background

According to the US Bureau of Transportation over 20% of flights expected delays in 2023[1]. Previous research has predicted flight delays using machine learning algorithms, using linear regression, random forests, and deep neural networks (Shao 2019). We also realized that the size of the dataset may require data sampling (Dai 2024). The data processes is further increased when weather conditions are taken into account (Kim 2024).

This project will use data from the Bureau of Transportation Statistics and their publicly available data set to estimate the probability of a passenger missing a connection flight. This data includes departure and arrival times, the carrier, and airport which can be used to predict connection-risk.

[1] “OST_R: BTS: Transtats.” BTS, Department of Transportation, www.transtats.bts.gov/homedrillchart.asp. Accessed 1 Oct. 2025.

2) Problem Definition

Problem:

We aim to predict the likelihood of a missed flight connection. Furthermore, rather than a simple boolean output, we hope to provide an accurate probability of missing the connection.

Motivation:

Current existing flight predictors are focused on giving odds of a delay; however, missed flight connections are arguably more impactful.

- Passengers need to rebook.
- Airlines need to cover costs of housing and re-routing.

A probability-based predictor enables travelers to make better booking choices and allows airlines to proactively reduce costs.

3) Methods

Data Preprocessing & Validation

- `notebooks/Data_Preprocessing.ipynb` handles ingestion of BTS On-Time records via `src/data/make_dataset.py`, giving us a reproducible, notebook-documented path from raw CSVs into the pipeline.
- `src/data/validation.py` runs schema/missing-value checks (row counts, dtype enforcement) before any modeling step so that our models can expect consistent data and we can stop bad batches early.
- All time (HHMM) fields are normalized to minutes-since-midnight, and `FL_DATE` is casted to `datetime64[ns]` for join and layover calculations, which prevents arithmetic errors when computing layovers or aggregating by calendar bins.
- One-Hot Encoding for the planned logistic-regression baseline has been implemented to convert categorical data into a numerical format that the machine algorithms we will be implementing can understand.
- Data filtering is implemented in `src/data/preprocessing.py::preprocess_flight_data`, which drops rows with missing `DEP_TIME/ARR_TIME`, imputes the timing fields, and applies IQR-based outlier removal to delays/duration columns to keep ~90% of flights so we model on reliable, high-signal observations.

Cleaning & Feature Engineering

- `src/data/preprocessing.py` converts numeric columns, imputes key timing fields (median for `ARR_DELAY`, `DEP_DELAY`, elapsed-time columns; mean for `DISTANCE`; zero for `CANCELLED`), drops rows missing `DEP_TIME/ARR_TIME`, and removes IQR outliers from `ARR_DELAY`, `DEP_DELAY`, `ACTUAL_ELAPSED_TIME`, and `DISTANCE`, retaining ~90% of rows so downstream models see dense, consistent features.
- `src/data/feature_engineering.py` creates `CRS_DEP_HOUR/MIN` and `CRS_ARR_HOUR/MIN`, casts categorical columns (`OP_UNIQUE_CARRIER`, origin/destination IDs, calendar features) to `category`, and defines the supervised target `IS_DELAYED = 1` when `ARR_DELAY ≥ 15` minutes (delay prevalence 9.6%), giving the model intuitive temporal splits and a clearly documented label.
- Cleaned datasets are saved to `/data/processed/processed_flight_data.csv` and summarized with `display_feature_summary`, which documents memory footprint and class balance for the report. This makes it easy to trace errors related to our generated dataset

Connection-Building Module

- `src/data/connection_builder.py` pairs inbound/outbound legs at the same airport within 30 minute to 6 hour layovers, computes planned vs. actual layover, flags `CONNECTION_MISSED`, and adds layover categories, hub indicators, and carrier-match features so that the prediction task mirrors real itineraries rather than isolated legs. This logic is integrated into the preprocessing notebook so we can move from per-leg delay probabilities to end-to-end connection risk easily as we develop our remaining models.

Models

- The three models that we utilized for this project was logistic regression, random forest classifier, and XGBoost
- All three models are supervised
  - This is because our goal is to predict a known target variable, which, is the delay and cancellation status of a flight.
  - To find this variable, we use labeled data, meaning each flight record in the historical dataset includes both the input features (time of flight, origin and destination airports, etc.) and the correct output (flight status).
  - These labels allow the models to learn the direct mapping between flight characteristics and the eventual outcome, optimizing its performance to accurately predict future outcomes.

Logistic Regression Model

- Why Logistic Regression?
  - It is a fundamental probabilistic model that natively outputs a score between 0 and 1, which directly fulfills the project's goal of providing an accurate probability of missing a connection.
  - Logistic regression serves as the most straightforward baseline model, allowing us to evaluate whether the complex methods (Random Forest, XGBoost) offer a performance gain significant enough to justify their complexity.
  - While slower than the tree models on this large dataset, it is computationally efficient for establishing initial model performance and interpretability before scaling up.
- Categorical Features: Requires One-Hot Encoding for categorical features (like airport and carrier IDs) to be used in the linear model.
- Numerical Features: Scaled using StandardScaler.
  - This process created a sparse model with 802 features in the sampled dataset.
- Imbalance Handling: The class_weight='balanced' parameter is used to adjust the misclassification penalty for the minority (delayed) class.
- Uses the saga solver for efficient handling of the large, sparse dataset.

Random Forest Model

- Why Random Forest?
  - Selected as an ensemble model capable of capturing non-linear interactions between features (such as the combination of a specific carrier and a specific time of day) that a linear model cannot resolve.\
  - It provides a reliable measure of feature importance, which is critical for identifying the most influential variables driving connection risk.
  - The tree structure naturally handles the mixture of the project's categorical and numerical features, and is robust to any remaining outliers that were not removed during the preprocessing step.
- Categorical Features: Processed using Label Encoding, which is efficient and appropriate for tree-based models.
- Numerical Features: Used directly as inputs without explicit scaling.
- Imbalance Handling: The class_weight='balanced' parameter is applied to give greater importance to the minority class.
- Parameters like n_estimators=100 and max_depth=20 were tuned to control model complexity and prevent overfitting.

XGBoost Model

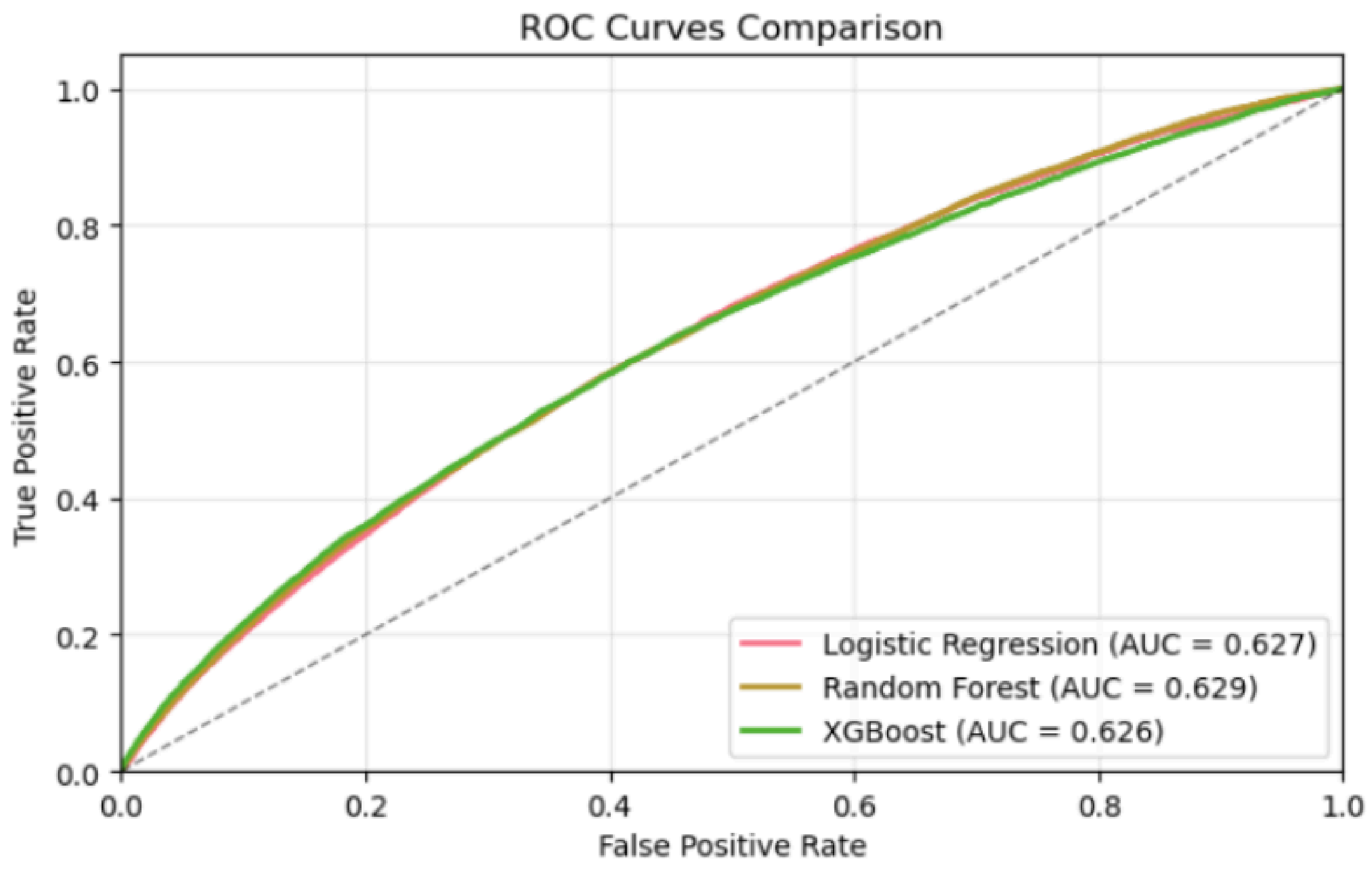
- Why XGBoost?:
  - Selected because it efficiently scales to our large dataset (hist tree method + native categorical support) and offers built-in controls for imbalance (`scale_pos_weight`) without oversampling overhead.
  - Sequential learning results in a higher prediction accuracy, maintains interpretability, and is computationally efficient.
  - Gradient-boosted trees also capture non-linear interactions between operational timing features and calendar/categorical signals.
- Implemented `FlightDelayXGBoost` in `src/models/xgboost_model.py` . The model uses 13 modeling features (carrier, origin, destination, calendar columns, CRS elapsed time, distance, departure/arrival hour & minute) and excludes leakage-prone fields like (`ARR_DELAY` , `DEP_DELAY` , `FL_DATE` , `IS_DELAYED`).
- Categorical Features: Uses Label Encoding and leverages its native categorical support.
- Numerical Features: Used directly as inputs without explicit scaling.
- Imbalance Handling: scale_pos_weight is set to the class imbalance ratio (approx. 9.47) to bias the model toward the rare "delayed" class.
- Uses extensive hyperparameters including max_depth, learning_rate, subsample, and L1/L2 regularization

- **Dataset splits: 80/20 train-test with an internal 85/15 validation cut inside `model.fit` .**

4) Results and Discussion

This section evaluates the performance of Logistic Regression, Random Forest, and XGBoost on the task of predicting flight delays. Along with visualizations, quantitative metrics, and confusion matrices, we analyze why each model behaves the way it does and what limits overall accuracy. A major goal of this section is to understand not only which model performs best but also why all three models converge to similar outcomes and what steps are needed to significantly improve prediction performance.

ROC Curve Comparison



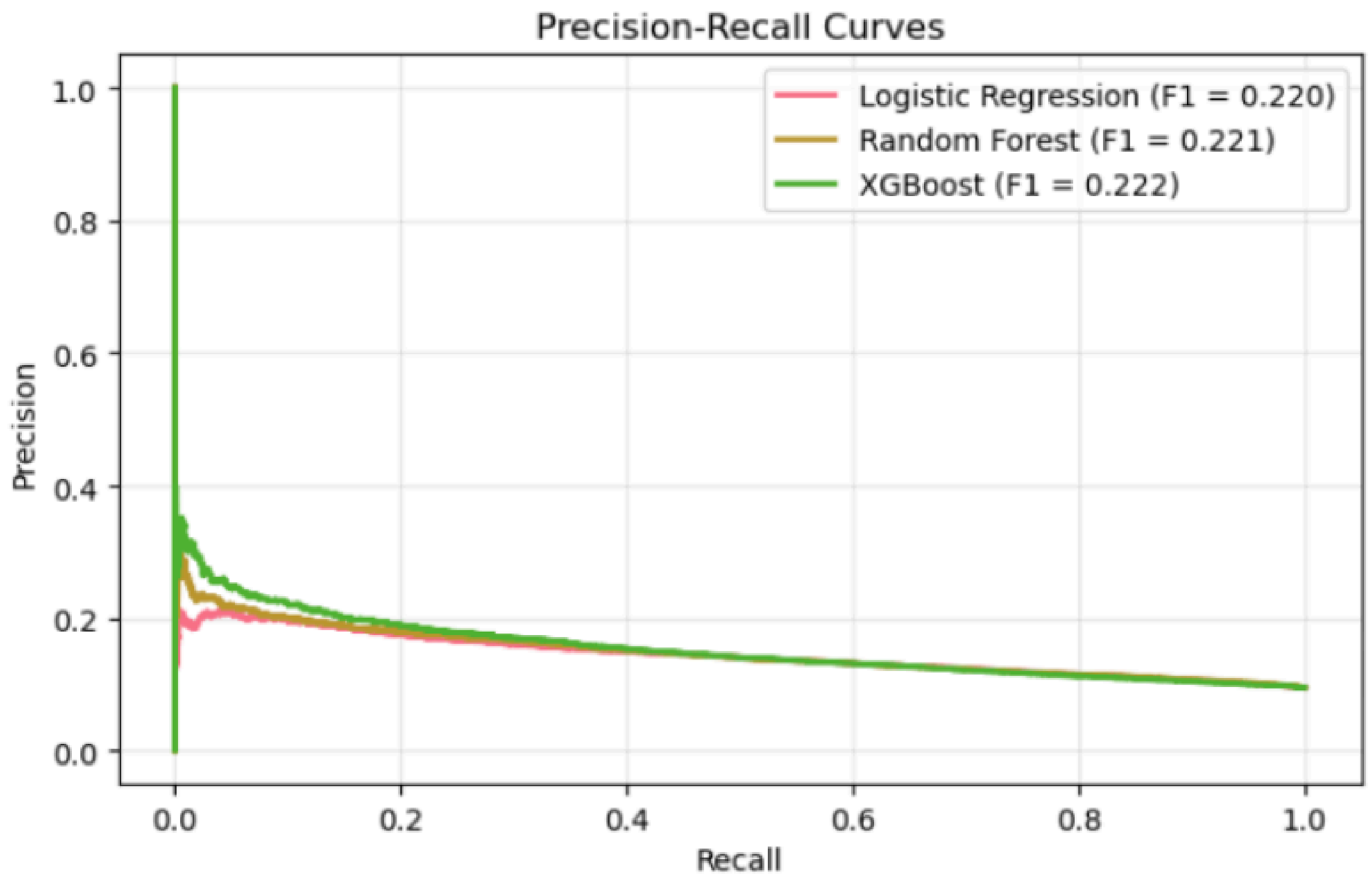
The ROC curves for all three models overlap nearly perfectly, and each model achieves an AUC score of around **0.63**:

- Logistic Regression AUC: **0.627**

- Random Forest AUC: **0.629**
- XGBoost AUC: **0.626**

These values show that the models are only slightly better than random chance. This does not reflect a failure of the models themselves but rather a limitation of the dataset. Many of the true causes of flight delays, such as weather, airport congestion, mechanical issues, and inbound aircraft delays, are not included in the training data. Without these strong predictors, the models rely on weak scheduling patterns and airport identifiers, leading to similar and limited ROC performance.

Precision and Recall Performance



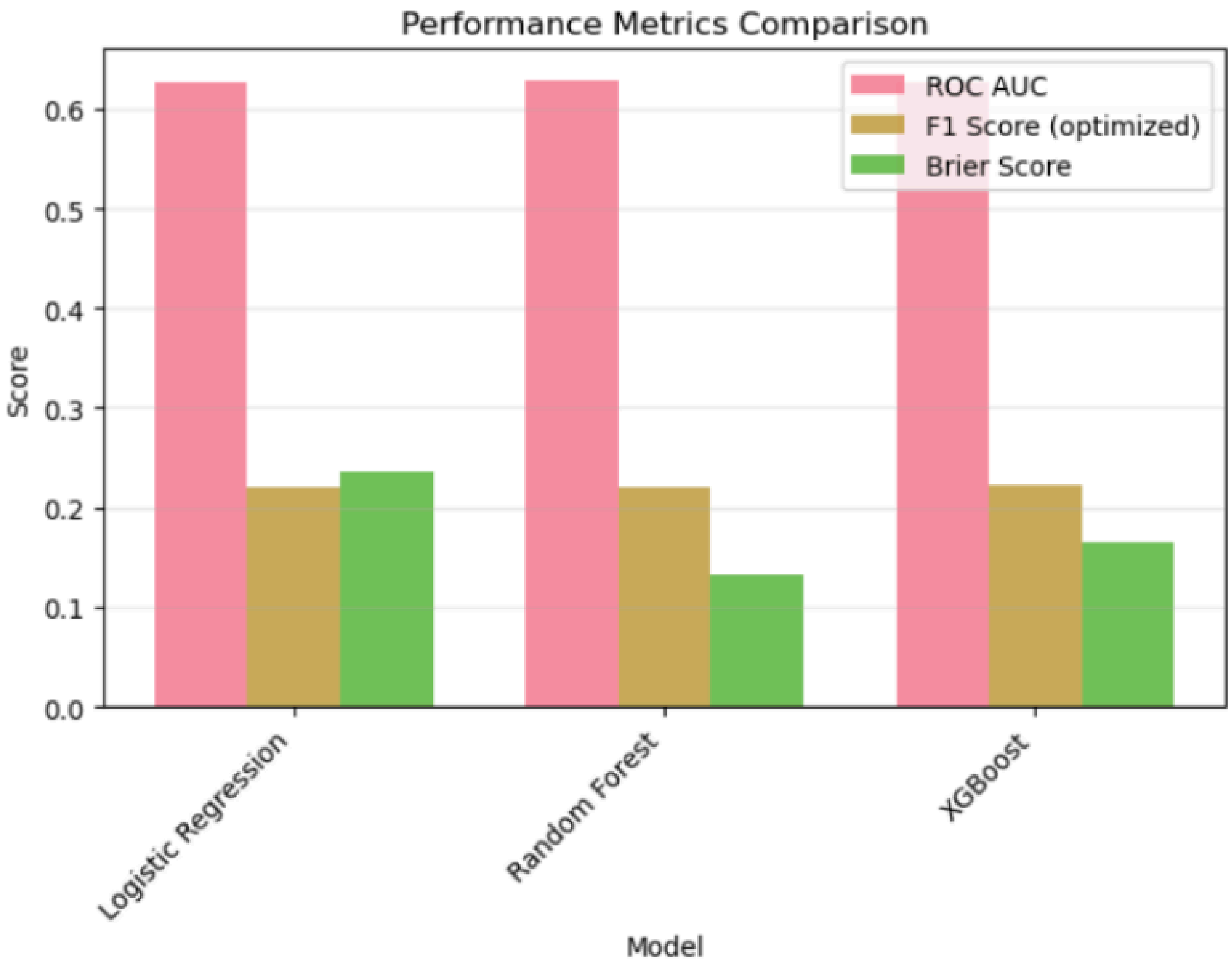
The Precision–Recall curves highlight how difficult it is for the models to identify delayed flights. Their best F1 scores are:

- Logistic Regression F1: **0.220**
- Random Forest F1: **0.221**
- XGBoost F1: **0.222**

Precision drops quickly as recall increases, which means that attempts to capture more delayed flights result in many false positives. This behavior is typical in rare-event classification problems. Since delayed flights are a small fraction of all flights, and because the dataset lacks strong predictive features, the models struggle to separate the delay class from the dominant on-time class.

Quantitative Metrics Summary

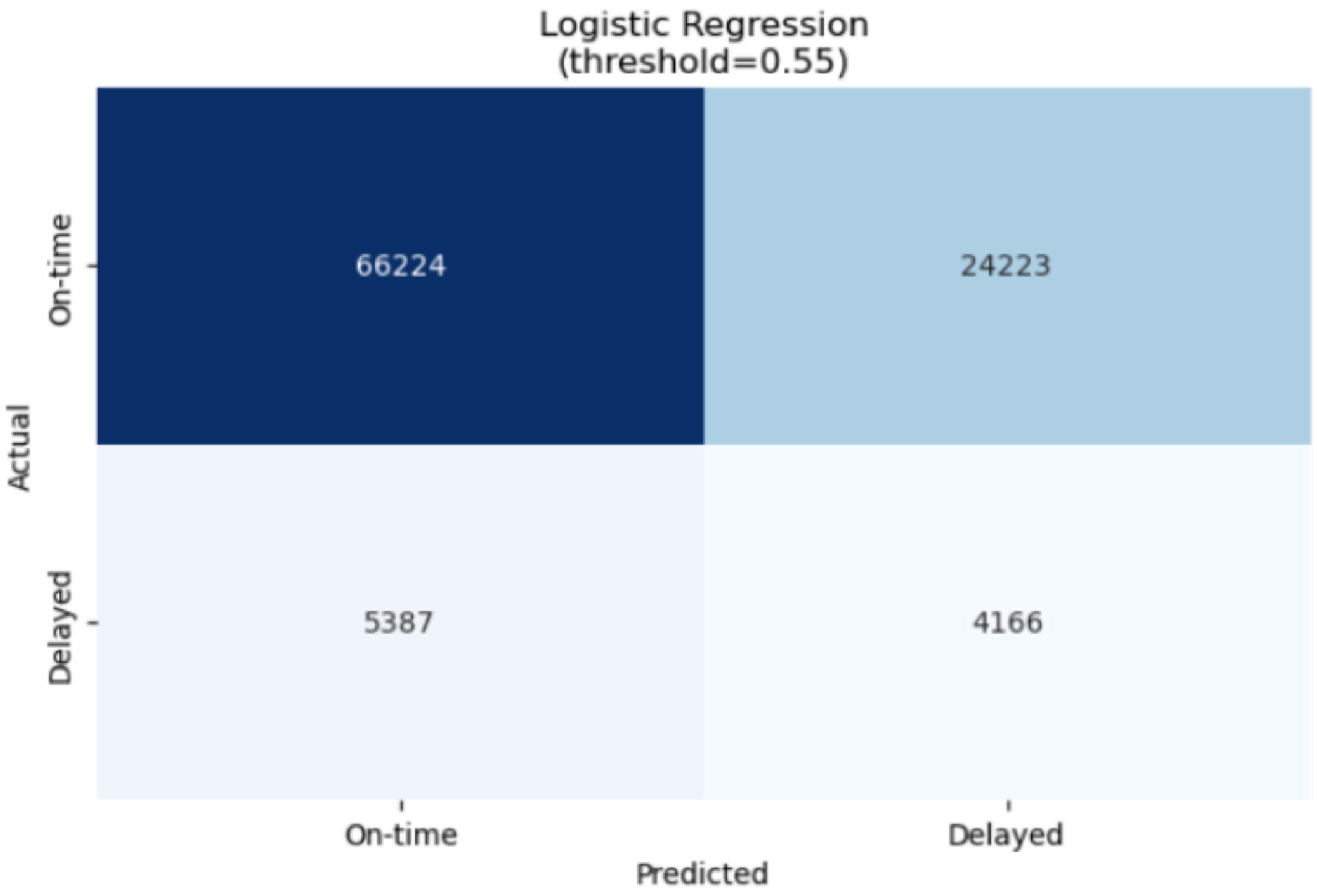
Model	AUC	F1 Score	Brier Score	Training Time (min)
Logistic Regression	**0.627**	**0.220**	**0.239**	**0.6**
Random Forest	**0.629**	**0.221**	**0.138**	**0.1**
XGBoost	**0.626**	**0.222**	**0.165**	**0.1**



These metrics show that the three models reach almost identical performance. All AUC values cluster around 0.63, and all F1 values cluster around 0.22. Random Forest provides the best-calibrated probability estimates, shown by its lower Brier score. Logistic Regression takes longer to train on this large dataset, while Random Forest and XGBoost train much faster. The most important takeaway is that **the dataset itself is the limiting factor**. Because the models lack access to meaningful causal features, even complex models like XGBoost cannot significantly outperform a simple linear classifier.

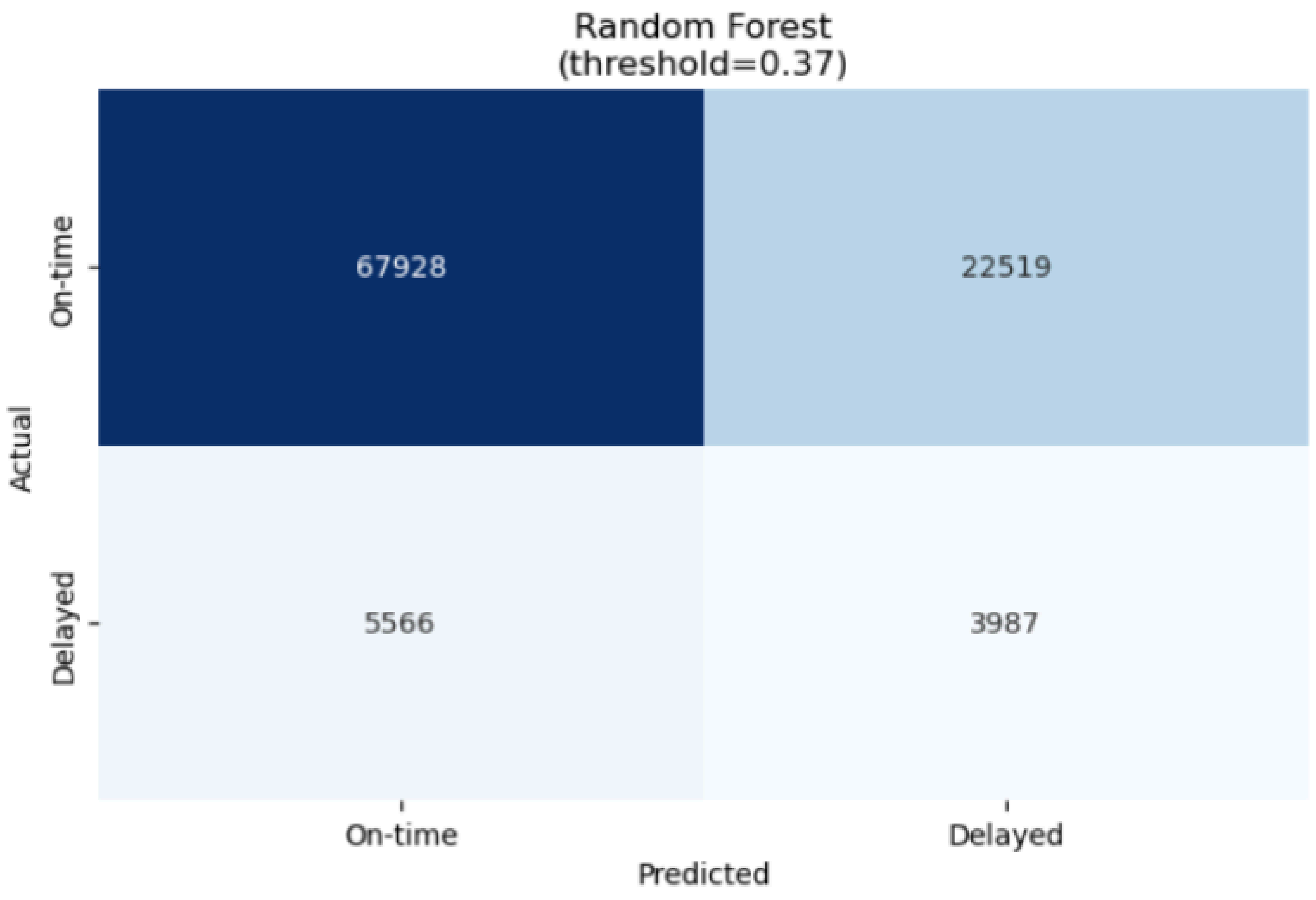
Confusion Matrix Evaluation

Logistic Regression



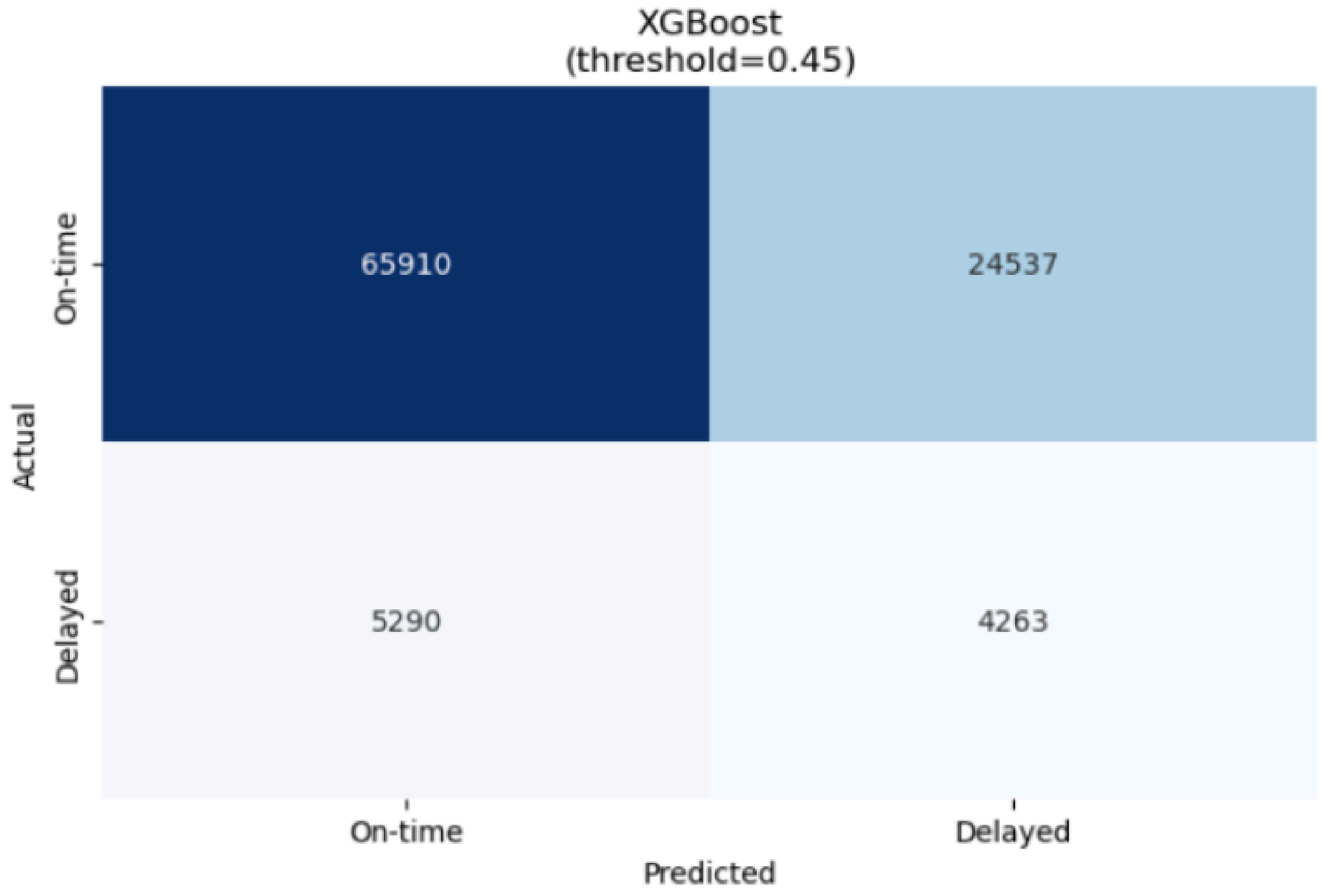
- True Positives: **4166**
- False Negatives: **5387**
- False Positives: **24223**
- True Negatives: **66224**

Random Forest



- True Positives: **3987**
- False Negatives: **5566**
- False Positives: **22519**
- True Negatives: **67928**

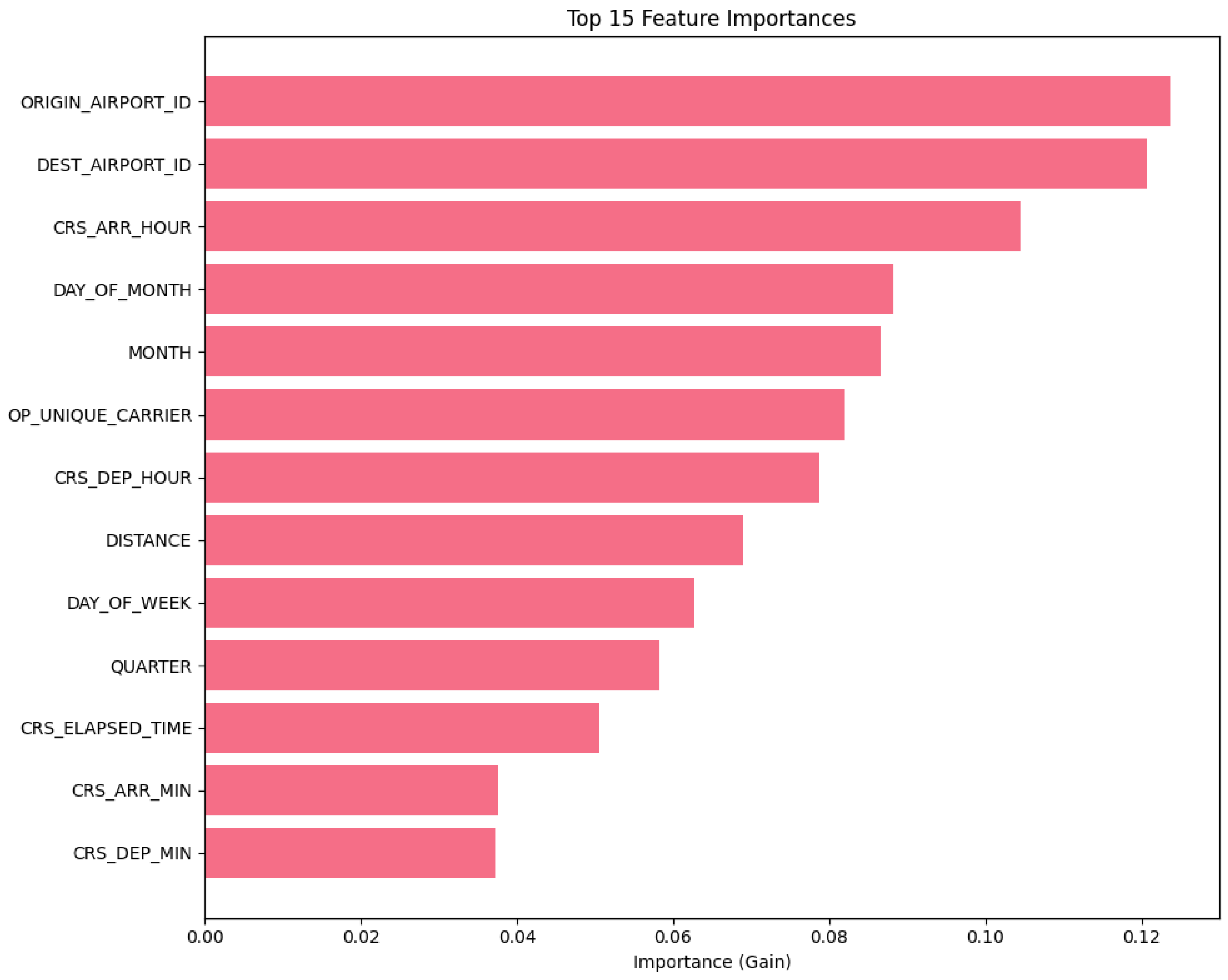
XGBoost



- True Positives: **4263**
- False Negatives: **5290**
- False Positives: **24537**
- True Negatives: **65910**

Across all three models, the confusion matrices show the same pattern. Each model predicts on-time flights well but misses many delayed flights. Adjusting thresholds can shift this balance slightly, but the underlying challenge remains. The models do not have access to the operational, environmental, and real-time indicators that typically signal whether a flight will be delayed. As a result, they cannot reliably distinguish delayed flights from on-time flights.

Feature Importance



Rank	Feature	Importance
1	ORIGIN_AIRPORT_ID	0.124
2	DEST_AIRPORT_ID	0.121
3	CRS_ARR_HOUR	0.105
4	DAY_OF_MONTH	0.088
5	MONTH	0.087

The most influential features are all related to airports and scheduling. These variables capture patterns such as airport congestion, time-of-day traffic peaks, and seasonal travel surges. Because the dataset lacks weather data and inbound aircraft information, the model uses these features as imperfect substitutes for the real drivers of delay. This explains both the moderate predictive performance and why all three models emphasize similar features.

Model Strengths and Limitations

Logistic Regression

Logistic Regression offers interpretability and stable probability estimates. It performs reasonably well despite its simplicity, which shows that the predictive relationships in this dataset are shallow. However, it cannot capture deeper nonlinear effects.

Random Forest

Random Forest models nonlinear interactions and often yields well-calibrated probabilities. It performs slightly better in calibration, but its overall accuracy does not improve much because the dataset lacks rich predictive signals.

XGBoost

XGBoost is a powerful gradient boosting method capable of modeling complex relationships, yet its advantage is minimal here. Without meaningful features related to real delay mechanisms, it cannot leverage its full potential.

Why the Models Perform Similarly

All three models produce nearly identical results because they all face the same fundamental limitation: **the dataset does not include the true causal factors behind aircraft delays**. Factors such as storms, late incoming flights, crew rotations, mechanical problems, and congestion at major hubs drive real-world delays. None of these appear in the dataset, so the models rely heavily on weaker scheduling and airport-based patterns. This leads to the convergence in metrics and prevents any model from achieving strong recall for delayed flights.

Next Steps

To meaningfully improve delay prediction accuracy, the dataset needs to be expanded with operational and environmental context. The most impactful next steps include:

1. Incorporating real weather data at departure and arrival airports
2. Adding inbound aircraft delays and aircraft turn time information
3. Integrating airport congestion and runway usage metrics
4. Training airport-specific or route-specific models

5. Adding time-series features to capture trends across hours and seasons
6. Engineering features related to holidays and peak travel periods
7. Optimizing decision thresholds for specific practical objectives

These additions would give the models access to strong, causal predictors rather than relying on indirect proxies. With richer data, advanced models like XGBoost would be able to outperform simpler methods and achieve significantly better recall for delayed flights.

5) References (IEEE)

1. Dai, M. A hybrid machine learning-based model for predicting flight delay through aviation big data. Sci Rep 14, 4603 (2024). https://doi.org/10.1038/s41598-024-55217-z
2. Kim, S., Park, E. Prediction of flight departure delays caused by weather conditions adopting data-driven approaches. J Big Data 11, 11 (2024). https://doi.org/10.1186/s40537-023-00867-5
3. Wei Shao, Arian Prabowo, Sichen Zhao, Siyu Tan, Piotr Koniusz, Jeffrey Chan, Xinhong Hei, Bradley Feest, & Flora D. Salim (2019). Flight Delay Prediction using Airport Situational Awareness Map. CoRR, abs/1911.01605.

6) Contribution Table

Name	Proposal Contributions
Jonathan Liang	Proposal: Quantitiative Metrics, Project Goals, Expected Results, Drafted Initial Proposal Outline, Drafted Presentation from Proposal Midterm: Data Scraping / Collection, Data Preprocessing Pipeline (Dataset Creation, Data Validation, Feature Engineering, Preprocessing Algorithms, Flight Connections Generation), Initial XG Boost Model Framework Final: Edited Final Presentation Slides, Polished Final Proposal
Allen Jiang	Proposal: Data Preprocessing and Algorithm Methods, Literature on Algorithms and Libraries, Drafted Slides on Methods Midterm: Data Collection, Data Preprocessing, Completed XGBoost model training/testing, xgboost_model.py and associated notebook Final: Created and Wrote Final Presentation Slides, Polished Final Proposal
Logan Tao	Proposal: Final overall proposal edits, Gantt Chart, Lit Review Midterm: XGBoost model implementation, hyperparameter tuning and training,

Name	Proposal Contributions
	report section three Final: Edited Final Presentation Slides, Polished Final Proposal
Ethan Saddler	Problem Statement, Polished Slides Midterm: Pre-Processing and Model Evaluation Final: Edited Final Presentation Slides, Polished Final Proposal
Neathan Aresh	Proposal: Introduction/Background, Video Creation, Dataset Identification Midterm: Model Training, Model Evaluation, Result and Discussion Final: Edited Final Presentation Slides, Polished Final Proposal

7) Gantt Chart

Download the Gantt Chart

8) Repository Map

/assets/: Assets forlder for items needed

/data/: Project data folder (raw → interim → processed)
/data/raw/: Monthly BTS On-Time CSVs (large; managed with Git LFS or excluded)

/notebooks/: Analysis and walkthrough notebooks
/notebooks/Data_Preprocessing.ipynb: End-to-end preprocessing demo (cleaning, join, features)
/notebooks/Machine_Learning.ipynb: Machine learning model implementation

/src/: Source code for the data pipeline
/src/data/make_dataset.py: Ingest monthly files, schema checks, write interim datasets
/src/data/preprocessing.py: Cleaning, missing values, outlier handling, encoding/scaling
/src/data/connection_builder.py: Build inbound→outbound pairs, compute layovers & labels
/src/data/feature_engineering.py: Time/operational/connection feature construction
/src/data/validation.py: Splits, leakage control, sanity checks

/requirements.txt: Python dependencies
/README.md: This file

9) How to Run

1) Create environment

```
python -m venv .venv
source .venv/bin/activate # Windows: .venv\Scripts\activate
pip install -r requirements.txt
```

2) Place monthly BTS CSVs under data/raw/

```
git lfs install (If not already installed)
git lfs fetch –all
git lfs checkout
```

3) Build datasets

```
python src/data/make_dataset.py
python src/data/preprocessing.py
python src/data/connection_builder.py
python src/data/feature_engineering.py
```

4) Inspect in notebook

```
jupyter notebook notebooks/Data_Preprocessing.ipynb
```