Traffic Accident Impact Analysis

Achille Nguessie

Contents

1 Preprocessing with PySpark on AWS EMR

Dataset Overview

The initial phase of this project involved preprocessing a dataset of car accidents in the United States, containing approximately 7.7 million records from February 2016 to March 2023 across 49 states. Given the size and complexity of the dataset, we used an AWS EMR (Elastic MapReduce) cluster with PySpark to efficiently handle data processing tasks. The preprocessing steps were designed to clean and transform raw data into a format suitable for analysis and subsequent modeling, specifically to predict accident duration and their impact on traffic flow.

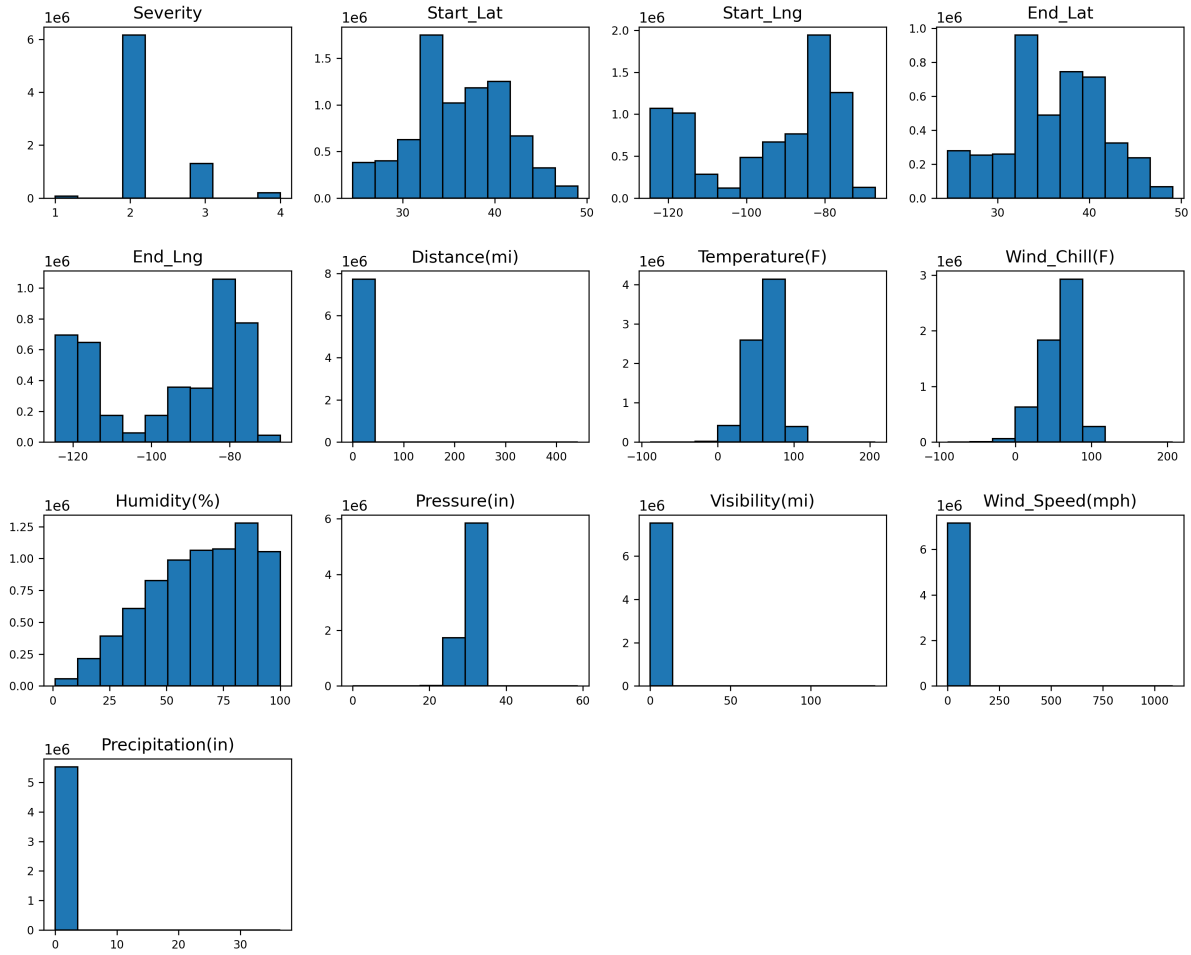

Figure 1: Raw Features Histogram

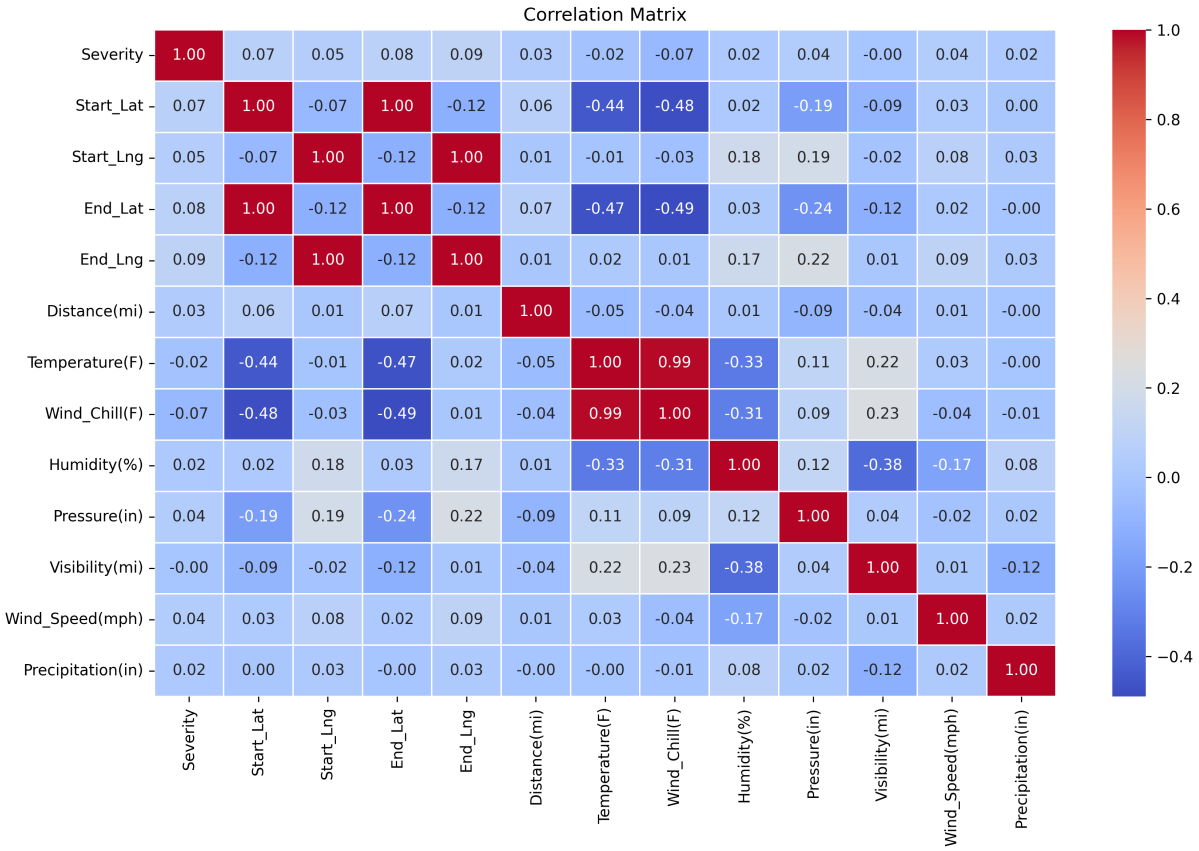

Figure 2: Raw Correlaton Matrix

Standardization

The dataset was loaded from a CSV file stored in an S3 bucket. To ensure consistency, all column names were standardized to lowercase. Unnecessary columns, such as `id`, `end_lat`, `end_lng`, and `wind_chill(f)`, were removed to streamline the dataset.

Handling Missing Values

Missing values in critical columns, such as `precipitation(in)` and `wind_speed(mph)`, were handled by filling them with monthly averages calculated based on the accident start time. This approach ensured that gaps in weather-related data were addressed contextually. Any remaining rows with null values were removed to maintain data integrity.

Temporal Feature Extraction

To prepare the dataset for temporal analysis, the `start_time` and `end_time` columns were converted to timestamps, and accident duration was calculated in minutes. Additionally, temporal features—including start hour, day, month, and year—were extracted from `start_time` to enhance the dataset with time-related information, which is essential for understanding patterns in accident duration and traffic impact.

Outlier Detection and Removal

Outliers were identified and removed to ensure dataset reliability. This process focused on key numerical columns such as `distance(mi)`, `temperature(f)`, `pressure(in)`, `visibility(mi)`, `wind_speed(mph)`, `precipitation(in)`, and `accident_duration(min)`. Whiskers (lower and upper bounds) were computed using the interquartile range (IQR) method with approximate quantiles, and extreme values falling outside these bounds were filtered out.

Saving and Logging

The final preprocessed dataset was saved as a Parquet file in an S3 bucket, merged into a single partition for simplicity. Throughout the process, logging was implemented to ensure transparency and facilitate debugging. The log file was downloaded alongside the output data for further verification.

This preprocessing effort reduced noise, improved data quality, and set the stage for accurate accident duration modeling.

2 Text Processing with Doc2Vec

Following the initial preprocessing with PySpark, the next step involved processing textual data in the dataset, specifically the `description` column containing details of each accident. Given the large volume of data—approximately 7.7 million records—using advanced embedding models like OpenAI's `text-embedding-ada-002` was deemed impractical due to computational and cost constraints. Instead, we adopted a traditional but efficient vectorization approach using `Doc2Vec`, a model from the Gensim library, to transform text into numerical representations suitable for downstream modeling.

Data Preparation

The text processing workflow began by loading the preprocessed dataset from a Parquet file generated by the PySpark job on AWS EMR. We first normalized categorical features to ensure consistency across the dataset. The `visibility(mi)` column was removed as it was deemed non-essential for this phase. Integer-based columns—such as `severity`, `start_hour`, `start_day`, `start_month`, and `start_year`—were converted to categorical type, while object and boolean columns were similarly transformed. All categorical values were then standardized by converting them to lowercase and removing spaces.

Text Cleaning and Tokenization

The core text processing focused on the `description` column. We tokenized the text by converting it to lowercase, removing punctuation, and splitting it into individual words. Non-alphabetic tokens were filtered out, and remaining words were processed by removing stopwords (common words like "the" or "and") and applying lemmatization to reduce words to their root form (e.g., "running" to "run"). This resulted in a clean set of tokens for each accident description, ready for vectorization.

Doc2Vec Embedding and Dimensionality Reduction

To convert these tokens into numerical embeddings, we trained a `Doc2Vec` model with vector size 100, window size 5, and 20 epochs, using the distributed memory (DM) approach.

The model was trained on tagged documents, where each tokenized description was associated with a unique identifier. Once trained, the model generated 100-dimensional vectors capturing the semantic meaning of each description.

To reduce dimensionality and improve computational efficiency, we applied Principal Component Analysis (PCA) to these vectors, retaining the top three principal components. These PCA-derived features—labeled `description_pca1`, `description_pca2`, and `description_pca3`—were added to the dataset, while the original `description` and `tokens` columns were discarded.

Storage

The resulting dataset, now enriched with text embeddings, was saved as a Parquet file for use in subsequent modeling steps. This approach efficiently transformed unstructured text into a structured format, enabling the inclusion of accident descriptions in predicting traffic flow impact. The process was logged throughout, with logs stored alongside the output for traceability.

3 Feature Engineering

After completing the initial preprocessing and text processing phases, an additional feature engineering phase was necessary to refine the dataset for modeling. This step aimed to enhance the predictive power of features by transforming, grouping, and reducing them, ensuring the dataset was optimized for forecasting accident duration and traffic flow impact.

Feature Selection and Outlier Handling

The feature engineering process began by loading the dataset from the Parquet file generated in the text processing step. We first simplified the dataset by removing columns deemed redundant or irrelevant for analysis. These included metadata like `weather_timestamp`, `airport_code`, `country`, and `source`, as well as location-specific details such as `street`, `city`, `county`, and `zipcode`. Additionally, binary indicators of road conditions—such as `amenity`, `bump`, `crossing`, and `traffic_signal`—and twilight-related columns were removed to reduce dimensionality. Outliers in the `accident_duration(min)` column were also filtered to ensure the target variable remained robust.

Cyclic Encoding of Temporal and Spatial Features

To capture the periodic nature of temporal and spatial features, we applied cyclic encoding using sine and cosine transformations. `start_lng` (longitude) was encoded with a 360-degree period, while `start_hour` (24-hour cycle), `start_day` (7-day week), and `start_month` (12-month year) were similarly transformed. The original columns were then removed, leaving sine and cosine feature pairs that effectively represent these cyclic patterns without assuming linear relationships. For the `wind_direction` column, text values (e.g., "north", "east", "calm") were mapped to corresponding angles (in degrees), cyclically encoded with a 360-degree period, then replaced with their sine and cosine components.

Categorical Feature Grouping

To simplify the dataset and improve interpretability, categorical features were grouped into broader categories. The `state` column, representing the 49 states in the dataset, was categorized into `urban`, `rural`, or `unknown` based on a predefined list of urban and rural states. This reduced granularity while preserving meaningful distinctions in traffic and accident patterns. Similarly, the `weather_condition` column, which contained a wide range of detailed weather descriptions, was mapped into four broader groups: `clear`, `cloudy`, `precipitation`, and `obscured`. Unmapped conditions were labeled as `unknown`, reducing noise and consolidating related conditions into more manageable categories.

Storage

The final dataset, now transformed and enriched, was saved as a Parquet file. This process resulted in a cleaner, more focused set of features, with reduced dimensionality and better representation of temporal, spatial, and environmental factors. The feature engineering phase ensured the data was well-prepared for modeling, balancing complexity and predictive potential. Logs were maintained throughout the process and stored alongside the output for reference.

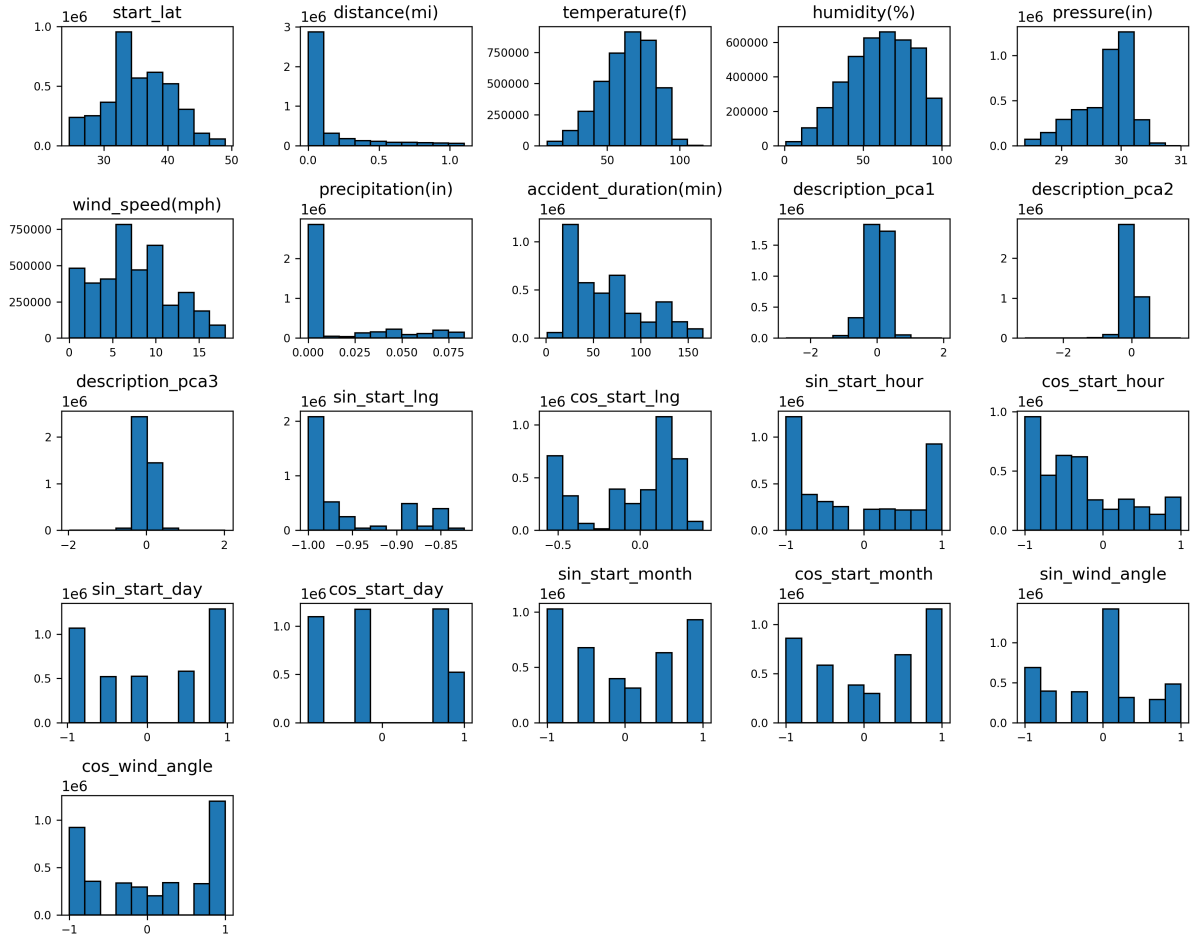

Figure 3: Final Features Histograms

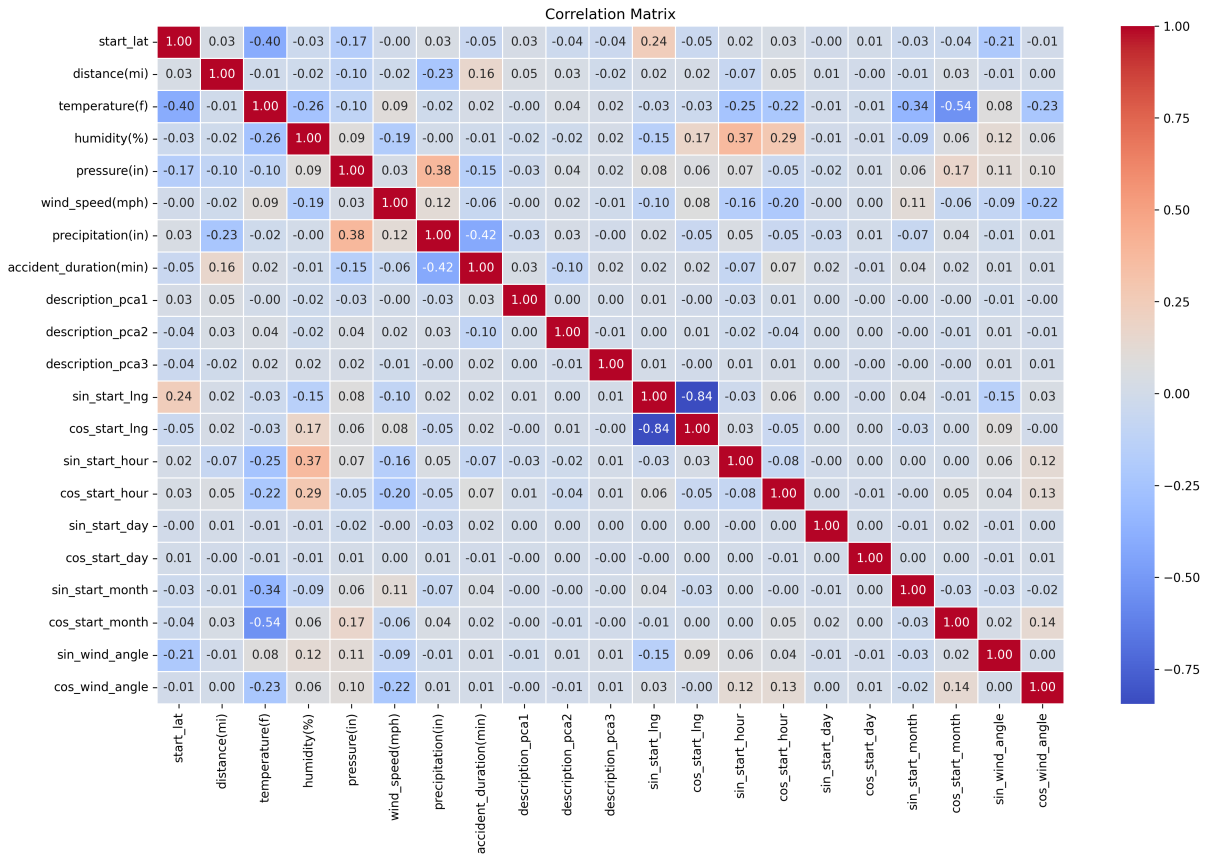

Figure 4: Final Correlaton Matrix

4 Modeling

With the dataset fully preprocessed and enriched through feature engineering, the next phase focused on training predictive models to forecast accident duration and assess their impact on traffic flow. To accomplish this, we developed training pipelines for five distinct regression models: Linear Regression, Random Forest, XGBoost, CatBoost, and LightGBM. These pipelines were designed not only to train the models but also to optimize their hyperparameters, evaluate performance, and save results - including metrics and visualizations - for further analysis.

Data Preparation and Splitting

The modeling process began by loading the final preprocessed dataset from a Parquet file. To accommodate potential computational constraints, we allowed sampling a fraction of the data (default 100%), ensuring flexibility in experimentation. The dataset was split into features (`X`) and target variable (`accident_duration(min)`), with an 80-20 train-test split applied to reserve 20% of the data for testing. Numerical features were standardized using a `StandardScaler`, while categorical features - such as `state_group` and `weather_group` - were one-hot encoded (dropping the first category to avoid multicollinearity). This preprocessing was integrated into a `ColumnTransformer` within each pipeline to ensure consistent data preparation.

Model Training and Hyperparameter Tuning

For each model, we employed `RandomizedSearchCV` to efficiently tune hyperparameters, performing a random search over predefined parameter distributions with 20 iterations and 5-fold cross-validation. The evaluation metric was set to negative mean squared error (RMSE) to prioritize minimizing prediction errors. The models and their tuned parameters were as follows:

- **Linear Regression**: No hyperparameters were tuned, serving as a baseline.
- **Random Forest**: Tuning `n_estimators` (100-500), `max_depth` (10-30 or None), and `min_samples_split` (2-10).
- **XGBoost**: Tuning `n_estimators` (100-300), `learning_rate` (0.01-0.1), and `max_depth` (4-10).
- **CatBoost**: Tuning `iterations` (100-500), `learning_rate` (0.01-0.1), and `depth` (4-10).
- **LightGBM**: Tuning `n_estimators` (100-300), `learning_rate` (0.01-0.1), and `max_depth` (6-12).

Model Evaluation and Analysis

After training, the best-performing model (based on cross-validation) was evaluated on both training and test sets. Key metrics included adjusted R^2 (accounting for the number of features) and RMSE, providing insights into model fit and prediction accuracy. These metrics were logged for each model, along with the data fraction used, to facilitate comparison.

Beyond performance metrics, we analyzed feature importance and residuals to deepen our understanding of each model's behavior. For Linear Regression, feature importance was derived from coefficient magnitudes and associated p-values (calculated via `statsmodels`), while for tree-based models (Random Forest, XGBoost, CatBoost, LightGBM), it was based on built-in feature importance scores. These results were saved as Excel files. Residual analysis included histograms, scatter plots against predicted values, and Q-Q plots to assess normality and autocorrelation, with the Durbin-Watson statistic calculated to detect residual patterns. These visualizations were saved as PNG files for further analysis.

Model Storage and Reproducibility

The trained pipelines, optimized with their best hyperparameters, were serialized and saved as `.pkl` files, named according to model type and data fraction (e.g., *RandomForest-frac-1.0.pkl*). This ensured reproducibility and easy access for future predictions or analysis. Logs captured the entire process, including preparation, training, and evaluation steps, and were stored alongside the outputs.

5 Results and analysis

5.1 Model Results

Table 1: R^2 on training set

Models / frac	**10%**	**30%**	**50%**	**80%**	**100%**
Linear	0.27	0.27	0.27	0.27	0.27
XGBoost	0.68	0.62	0.63	0.61	-
LightGBM	0.54	0.53	0.53	0.53	-
CatBoost	0.60	0.57	0.56	-	-
RandomForest	0.93	0.94	-	-	-

Table 2: R^2 on test set

Models / frac	**10%**	**30%**	**50%**	**80%**	**100%**
Linear	0.27	0.28	0.27	0.27	0.27
XGBoost	0.53	0.55	0.55	0.56	-
LightGBM	0.51	0.52	0.52	0.52	-
CatBoost	0.53	0.54	0.54	-	-
RandomForest	0.53	0.55	-	-	-

Table 3: RMSE on training set

Models / frac	**10%**	**30%**	**50%**	**80%**	**100%**
Linear	32.18	32.17	32.13	32.14	32.15
XGBoost	21.16	23.17	22.93	23.44	-
LightGBM	25.52	25.70	25.90	25.90	-
CatBoost	23.60	24.70	24.90	-	-
RandomForest	9.66	9.40	-	-	-

Table 4: RMSE on test set

Models / frac	**10%**	**30%**	**50%**	**80%**	**100%**
Linear	32.14	32.10	32.14	32.11	32.15
XGBoost	25.71	25.34	25.10	24.92	-
LightGBM	26.30	26.00	26.00	25.90	-
CatBoost	25.80	25.50	25.40	-	-
RandomForest	25.80	25.11	-	-	-

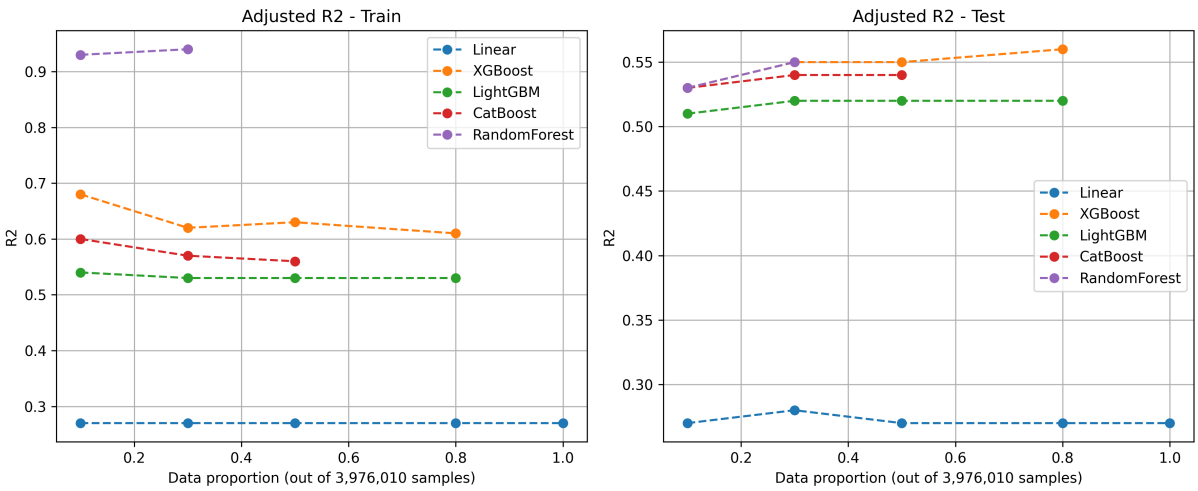

Figure 5: Adjusted R^2 results

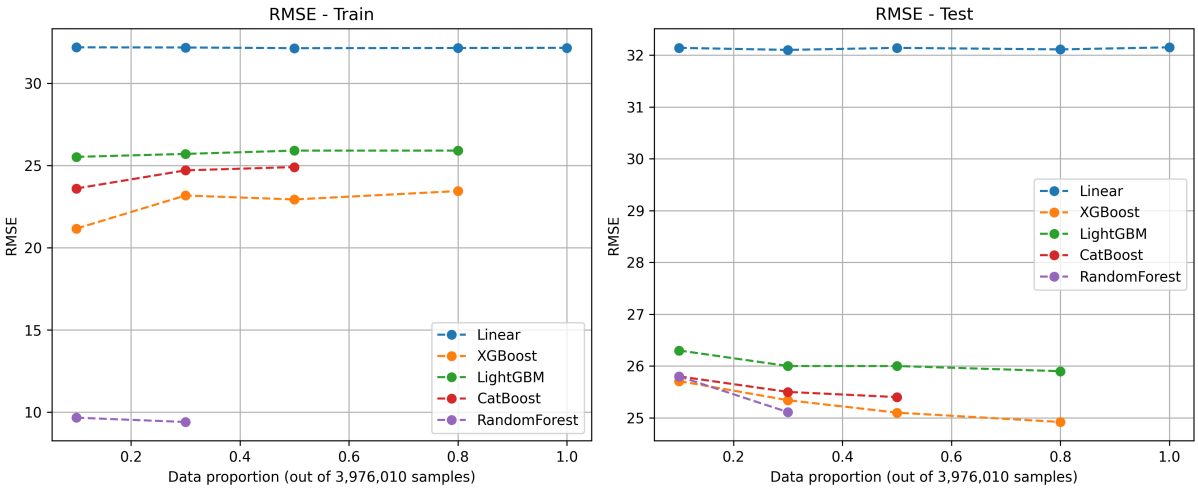

Figure 6: RMSE results

5.2 Analysis

Linear Model

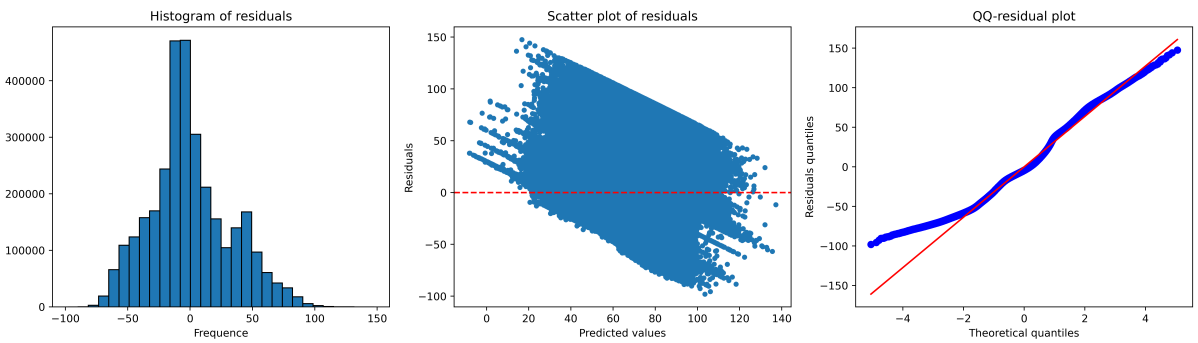

Figure 7: Final Correlaton Matrix

The Linear model exhibits remarkable consistency across all data proportions (10% to 100%). Its R^2 remains fixed at 0.27, with RMSE values tightly oscillating between 32.13–32.18 on the training set and 32.10–32.15 on the test set. This stability suggests that the model is too simplistic to benefit from additional data, showing neither overfitting nor adaptation to increased training size. The low R^2—explaining only 27% of the variance—and high RMSE, indicative of large prediction errors, confirm underfitting. The model struggles to capture the complexity of the data, regardless of dataset size.

XGBoost

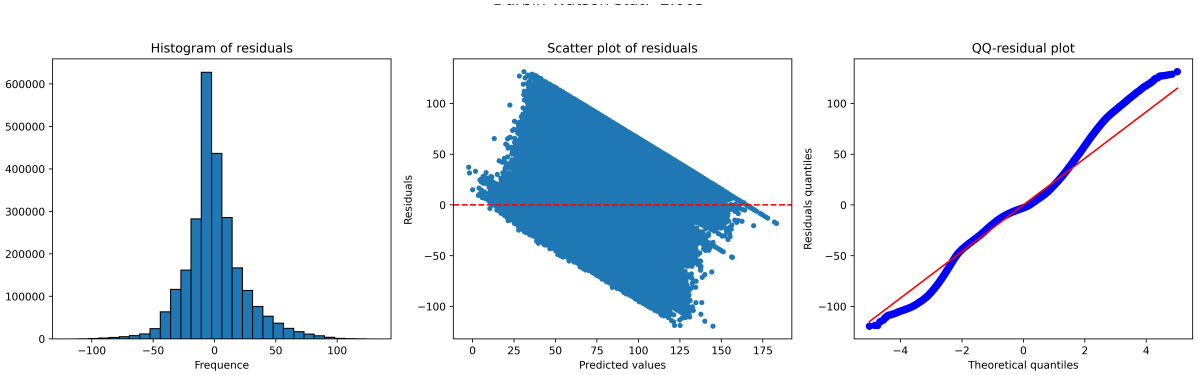

Figure 8: Residual Plots XGBoost

In contrast, XGBoost displays more dynamic behavior. On the training set, R^2 starts high at 0.68 with 10% data but declines to 0.61 at 80%, while RMSE rises from 21.16 to 23.44. This pattern suggests that with a smaller dataset (10%), XGBoost may overfit, achieving a tighter fit, but as more data is introduced, the fit weakens, possibly due to increased noise or complexity. On the test set, R^2 improves from 0.53 to 0.56 and RMSE decreases from 25.71 to 24.92 as the proportion grows from 10% to 80%. This trend indicates that XGBoost benefits from larger training sets, enhancing generalization, though the gap between training and test metrics (e.g., R^2 of 0.61 vs. 0.56 at 80%) points to moderate overfitting.

LightGBM

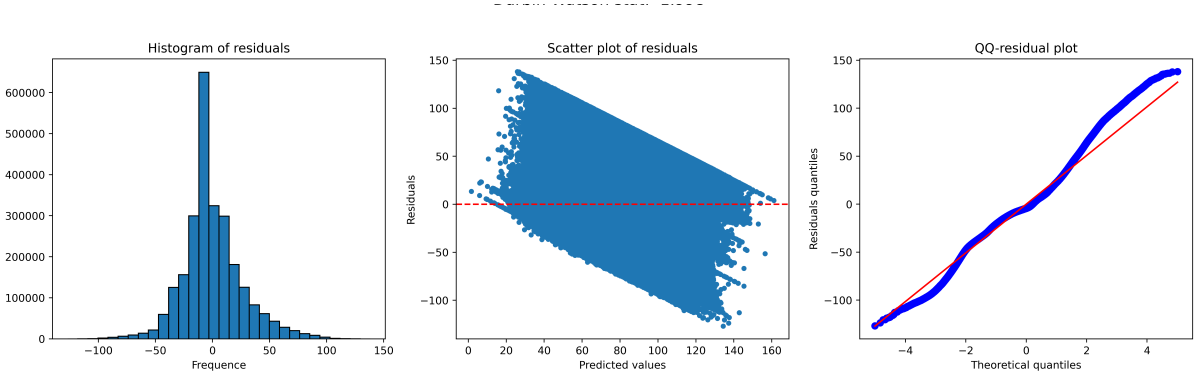

Figure 9: Residual Plots LightGBM

LightGBM presents a more stable performance profile. Its training R^2 remains steady at 0.53–0.54 across proportions from 10% to 80%, with RMSE slightly increasing from 25.52 to 25.90. On the test set, R^2 rises modestly from 0.51 to 0.52, and RMSE improves marginally from 26.30 to 25.90. The small gap between training and test results (e.g., R^2 of 0.53 vs. 0.52 at 80%) suggests minimal overfitting. While the slight improvement in test performance with more data indicates some benefit from larger training proportions, this effect is less pronounced than in XGBoost.

CatBoost

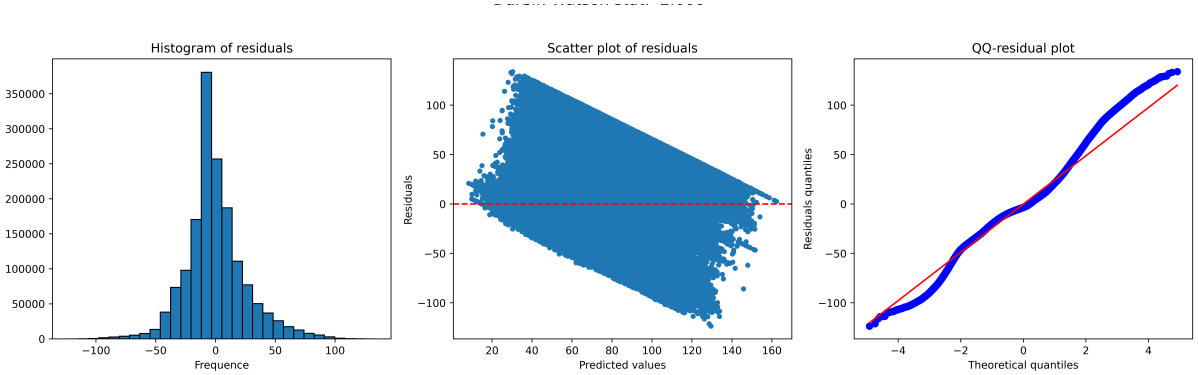

Figure 10: Residual Plots CatBoost

CatBoost shows a pattern similar to XGBoost, though with fewer data points due to training time constraints. Training R^2 decreases from 0.60 at 10% to 0.56 at 50%, with RMSE rising from 23.60 to 24.90, implying a tighter fit with less data. On the test set, R^2 improves from 0.53 to 0.54, and RMSE drops from 25.80 to 25.40 as the proportion increases to 50%. The moderate gap between training and test metrics (e.g., R^2 of 0.56 vs. 0.54 at 50%) indicates overfitting. However, the test improvement with more data aligns with XGBoost's trend, suggesting potential for better performance if trained on larger proportions.

RandomForest

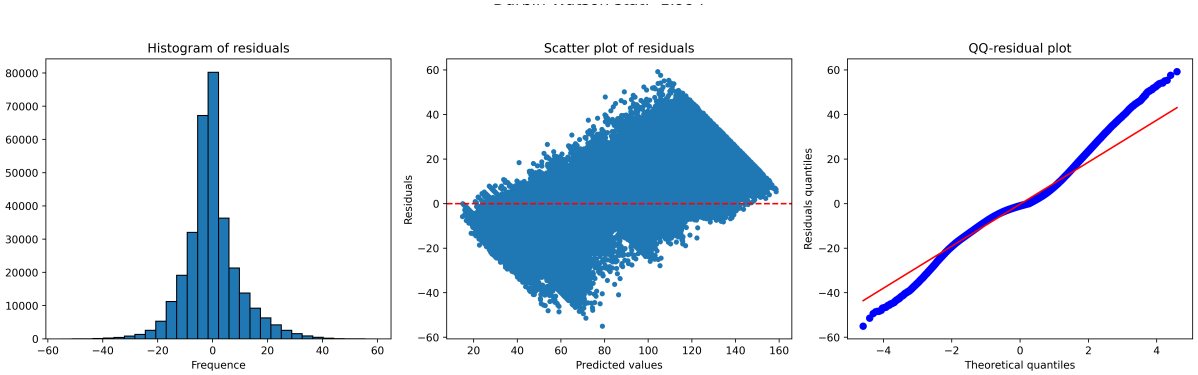

Figure 11: Residual Plots XGboost

RandomForest stands out with extreme results, limited to 10% and 30% proportions due to computational cost. On the training set, R^2 is exceptionally high (0.93–0.94) and

RMSE remarkably low (9.40–9.66), reflecting a near-perfect fit to small datasets. However, test performance is significantly worse, with R^2 ranging from 0.53–0.55 and RMSE from 25.11–25.80, revealing severe overfitting. The slight test improvement from 10% to 30% (RMSE dropping from 25.80 to 25.11) suggests that additional data helps marginally. Still, the large disparity between training and test metrics indicates that RandomForest memorizes training data rather than learning generalizable patterns.

6 Conclusion

Comparing models, Linear model's underfitting is evident in its inability to improve with more data, stuck at low R^2 and high RMSE. XGBoost, LightGBM, and CatBoost all show moderate overfitting, with training metrics outperforming test, but they benefit from larger proportions, particularly XGBoost, which achieves the best test performance (R^2 0.56, RMSE 24.92 at 80%). LightGBM's stability and lower overfitting make it a reliable alternative, while CatBoost's trajectory suggests potential if extended to higher proportions. RandomForest's severe overfitting makes it impractical despite training prowess, likely exacerbated by the small dataset sizes tested.

The trend across proportions reveals that more data generally improves test performance for tree-based models (XGBoost, LightGBM, CatBoost), reducing RMSE and increasing R^2, though training performance sometimes weakens as overfitting decreases. RandomForest might follow this trend if trained on more data, but its computational cost limited exploration. Linear model's indifference to dataset size underscores its unsuitability for this task.

XGBoost emerges as the strongest candidate, offering the best test metrics at 80% data. Its ability to leverage larger proportions suggests it might improve further at 100%, though training time was a constraint. LightGBM provides a stable, less overfitting option, while RandomForest needs regularization or more data to become viable. Given the modest test R^2 (max 0.56) and high RMSE (min 24.92), exploring feature engineering or ensemble methods could improve all models' performance, as current results suggest untapped potential in the data.

Future Improvements

Several strategies could enhance these models' performance given the current results and training time constraint with various dataset proportions. First, hyperparameter optimization for XGBoost, LightGBM, and CatBoost - such as tuning learning rates, tree depth, or regularization parameters - could reduce overfitting and improve test metrics, especially since XGBoost showed the best generalization at 80% data. Second, addressing RandomForest's severe overfitting might involve limiting tree complexity or testing it on larger proportions, potentially using distributed computing to mitigate training time issues. Finally, improving feature engineering or selection could increase all models' ability to explain more variance, as the highest test R^2 (0.56) and lowest RMSE (24.92) suggest current features leave significant patterns uncaptured.