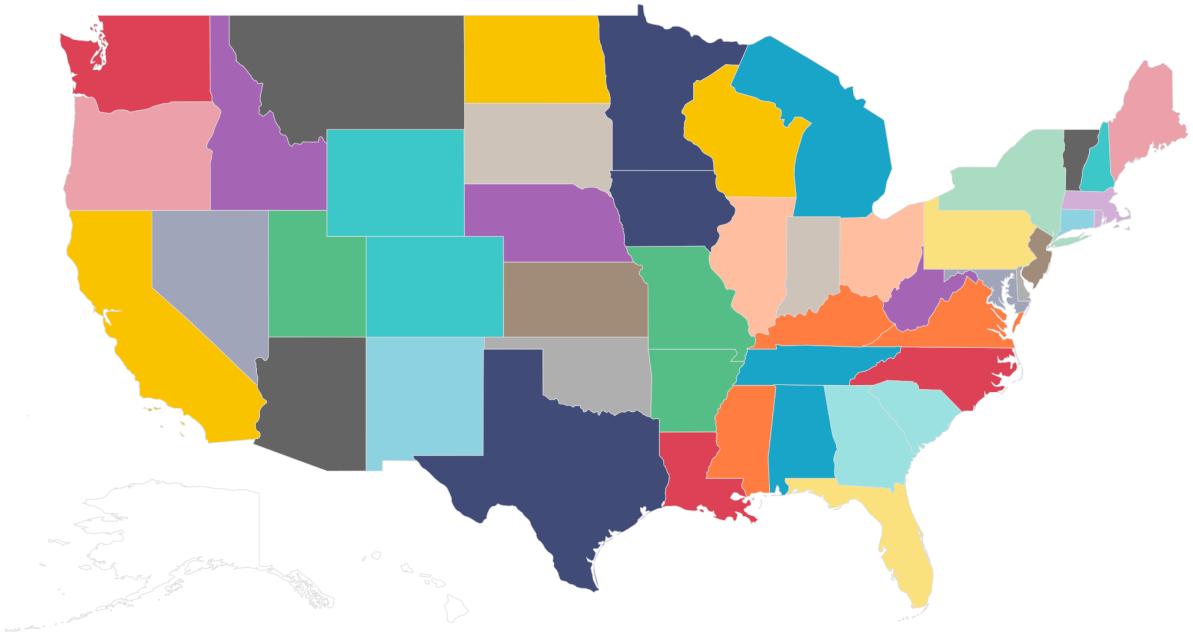
Predicting Traffic Delays Using U.S. Congestion and Weather Data

Team 26

Name	Role	Innomail
Alexey Tkachenko	Data engineer	a.tkachenko@innopolis.university
Daniil Abrosimov	ML specialist	d.abrosimov@innopolis.university
Egor Machnev	Data scientist	e.machnev@innopolis.university
Apollinaria Chernikova	Tester and Technical writer	a.chernikova@innopolis.university



Dashboard • Slides • GitHub • Dataset

Introduction

Traffic congestion represents a persistent and growing challenge across the United States, affecting millions of commuters on a daily basis. This phenomenon leads to increased travel times, fuel consumption, and environmental costs, while also introducing complexities into urban planning and logistics operations.

The objective of this project is to develop a scalable, end-to-end data pipeline that transforms raw traffic data into actionable insights using machine learning and interactive visualization. To achieve this, we leverage the publicly available US Traffic Congestion 2016–2022 dataset, which contains over **33 million** congestion records from **49 U.S. states**, spanning the period from **February 2016 to September 2022**.

The project is guided by three fundamental goals:

1. **Construct a big data pipeline** capable of handling real-world scale, starting from raw CSV files and progressing through data storage, transformation, modeling, and presentation in an analytical dashboard.
2. **Address key applied questions**, such as:
    a. Which factors (e.g., weather, time of day, location) most influence the severity and duration of traffic delays?
    b. Can the duration of a traffic jam be predicted with reasonable accuracy in advance?
3. **Support decision-making** for urban planners, transportation authorities, and logistics companies by offering a clear, intuitive dashboard that highlights congestion hotspots, temporal patterns, and model-based delay forecasts.

Throughout the course of this project, we addressed a range of technical challenges — including the management of missing data, engineering of temporal and spatial features, and the complexity of developing distributed machine learning models. The final system integrates **PostgreSQL**, **Hive**, **Spark MLlib**, and **Apache Superset**, forming a dependable and extensible analytics platform for both real-time and historical traffic congestion data.

Data Description

For this project, we selected the **US Traffic Congestion 2016–2022** dataset, publicly available on Kaggle. The dataset contains over **33 million** records describing congestion events reported across **49 U.S. states** over a period of more than six years.

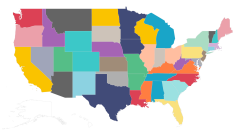
Each record captures multiple dimensions of a congestion event:

- **Spatial features**: precise latitude and longitude coordinates;
- **Temporal features**: event start and end timestamps;
- **Descriptive attributes**: severity, event type, delay, etc.;
- **Contextual data**: weather conditions, street and city names, and location metadata.

Below is a summary of the dataset's core characteristics:

Source	Kaggle – US Traffic Congestions 2016–2022
Coverage	49 U.S. states
Time Period	February 2016 – September 2022
Total Records	33,000,000+
File Format	CSV (≈11 GB uncompressed)
Fields	30 attributes
Key Features	Latitude, Longitude, Start/End Time, Severity, Weather, Location
Original Sources	Real-time traffic APIs (e.g., MapQuest, INRIX), road sensors, police & DOT reports
License	CC BY-NC-SA 4.0 (for non-commercial research use only)

This dataset serves as the foundation for both the analytical and predictive components of the project, offering rich multidimensional data suitable for large-scale machine learning and visual exploration.



Architecture of data pipeline

To manage the large-scale and multidimensional nature of the dataset, the project was designed around a modular, four-stage data pipeline. Each stage handles a distinct part of the data lifecycle — from ingestion and storage to analysis, modeling, and visualization. All scripts used throughout the pipeline are idempotent, ensuring that the system can be re-run reliably, with staging areas and temporary files cleaned at each step.

Pipeline Overview

Stage	Input	Processing	Output
Stage I	Kaggle ZIP archive (CSV, ~12.8 GB)	- Load and clean data - Import to PostgreSQL - Export to HDFS using Sqoop → convert to Parquet (Snappy-compressed)	Raw lake: /project/warehouse/traffic (Parquet)
Stage II	Parquet files from Stage I	- Create external Hive table traffic - Create optimized partitioned & bucketed table traffic_partitioned - Auto-ingest per-state - Generate 14 EDA views (CSV + Hive)	Hive warehouse + /output/dashboard/qX.csv
Stage III	50k-row-per-state sample from traffic_partitioned Hive table. ~2mil rows in total	- Spark MLlib pipeline with custom transformers: - Cyclical time encoder - GeoToECEF spatial transformer - Word2Vec embedding - OneHotEncoding for low cardinality categorical features - Feature hashing for high cardinality features - Grid search for LR and RF models	Trained models + evaluation metrics (RMSE, R^2, MAE)
Stage IV	Analytical results + model outputs	- Load into Apache Superset - Configure dashboards, filters, drilldowns	Interactive BI dashboard

Key Technologies Used

- **PostgreSQL** for initial data integrity and sampling
- **Sqoop + HDFS (Parquet)** for scalable storage
- **Hive** for structured querying and EDA view generation
- **Spark MLlib** for distributed modeling
- **Apache Superset** for live data visualization

This architecture ensures full traceability and modular execution, enabling repeatable analytics on a dataset of over 33 million records.

Development Automation ✨

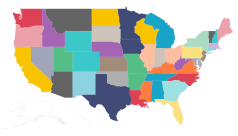
To streamline development and ensure continuous integration throughout the project lifecycle, we implemented a custom CI script tailored for the cluster environment. This Bash-based utility continuously monitors the deploy branch of the project repository and automatically:

- Fetches and resets to the latest commit;
- Executes the main pipeline script (main.sh);
- Sends logs and outputs to a Telegram chat for real-time team visibility.

The implementation of *ci-watch.sh* is as follows:

```
#!/bin/bash

set -euo pipefail
exec </dev/null

readonly CI_REPO_URL="https://github.com/da-the-dev/big-data-final-project"
readonly CI_REPO_DIR="$HOME/project/big-data-final-project"
readonly CI_BRANCH="deploy"
readonly CI_ENTRYPOINT="main.sh"
readonly CI_OUTPUT_DIR="output"
readonly CI_INTERVAL=30
readonly CI_TMP_DIR="$HOME/.ci-watch"

readonly TG_TOKEN="<your_token>"
readonly TG_CHAT_ID="<your_chat_id>"

mkdir -p "${CI_TMP_DIR}"

send_message() {
   local text="$1"
   local response
   response=$(curl -s -X POST
"https://api.telegram.org/bot${TG_TOKEN}/sendMessage" \
       -d "chat_id=${TG_CHAT_ID}" \
       -d "parse_mode=Markdown" \
       -d "disable_web_page_preview=true" \
```

```
        -d "text=${text}")
    MESSAGE_ID=$(grep -o '"message_id":[0-9]*' <<<"$response" | cut -d':' -f2)
}

edit_message() {
    local text="$1"
    curl -s -X POST "https://api.telegram.org/bot${TG_TOKEN}/editMessageText" \
        -d "chat_id=${TG_CHAT_ID}" \
        -d "message_id=${MESSAGE_ID}" \
        -d "parse_mode=Markdown" \
        -d "disable_web_page_preview=true" \
        -d "text=${text}" >/dev/null
}

send_document() {
    local file="$1"
    [ -f "$file" ] && curl -s -X POST
"https://api.telegram.org/bot${TG_TOKEN}/sendDocument" \
        -F "chat_id=${TG_CHAT_ID}" \
        -F "document=@${file}" >/dev/null
}

fail_ci() {
    local reason="$1"
    send_message "😡 CI error on \`$(hostname)\`: _${reason}_"
    exit 1
}

if [ ! -d "${CI_REPO_DIR}" ]; then
    git clone "${CI_REPO_URL}" "${CI_REPO_DIR}" || fail_ci "failed to clone
repository"
fi

cd "${CI_REPO_DIR}" || fail_ci "failed to change directory"
git fetch origin "${CI_BRANCH}" || fail_ci "failed to fetch branch"
git checkout -B "${CI_BRANCH}" "origin/${CI_BRANCH}" || fail_ci "failed to
checkout branch"

send_message "🛰️ CI up on \`$(hostname)\` — watching
[${CI_BRANCH}](${CI_REPO_URL}/tree/${CI_BRANCH})"

while true; do
    git fetch --all --prune

    local_hash=$(git rev-parse "${CI_BRANCH}")
    remote_hash=$(git rev-parse "origin/${CI_BRANCH}")

    if [ "${local_hash}" = "${remote_hash}" ]; then
        sleep "${CI_INTERVAL}"
        continue
    fi

    commit_info=$(git log -1 "origin/${CI_BRANCH}" --pretty=format:"%H;%an;%s")
    IFS=';' read -r commit_id commit_author commit_msg <<<"${commit_info}"

    short_hash="${commit_id:0:7}"
    timestamp=$(TZ="Europe/Moscow" date '+%Y-%m-%dT%H-%M-%S')
    logfile="${CI_TMP_DIR}/${timestamp}_${short_hash}.log"
    zipfile="${CI_TMP_DIR}/${timestamp}_${short_hash}.zip"

    url_commit="${CI_REPO_URL}/commit/${commit_id}"
```

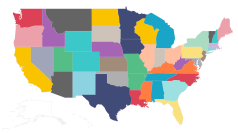
```
    url_branch="${CI_REPO_URL}/tree/${CI_BRANCH}"
    url_entrypoint="${CI_REPO_URL}/blob/${commit_id}/${CI_ENTRYPOINT}"

    header="👀 [${short_hash}](${url_commit}) on [${CI_BRANCH}](${url_branch}) by
${commit_author}: _${commit_msg}_"
    send_message "${header}"

    git reset --hard "origin/${CI_BRANCH}" || fail_ci "failed to reset"

    status="🚀 running [${CI_ENTRYPOINT}](${url_entrypoint}) on \`$(hostname)\`"
    edit_message "${header}"$'\n\n'"${status}"

    chmod +x "${CI_ENTRYPOINT}"
    ./"${CI_ENTRYPOINT}" 2>&1 | tee "${logfile}"
    exit_code=${PIPESTATUS[0]}

    if [ "${exit_code}" -eq 0 ]; then
        status="✅ finished [${CI_ENTRYPOINT}](${url_entrypoint}) on
\`$(hostname)\`"
    else
        status="❌ failed [${CI_ENTRYPOINT}](${url_entrypoint}) on
\`$(hostname)\`"
    fi

    edit_message "${header}"$'\n\n'"${status}"
    send_document "${logfile}"
    rm -f "${logfile}"

    if [ -d "${CI_OUTPUT_DIR}" ]; then
        zip -r "${zipfile}" "${CI_OUTPUT_DIR}" >/dev/null && send_document
"${zipfile}"
        rm -f "${zipfile}"
    fi
done
```

The *ci-watch.sh* script is located in the home directory of the *team26* user on the cluster. We excluded it from GitHub to avoid confusing users who clone and run the project.

The following commands are of particular utility:

Launch	`setsid bash ~/ci-watch.sh > /dev/null 2>&1 &`
Check status	`ps aux | grep ci-watch.sh`
Terminate	`pkill -f ci-watch.sh`

The script is idempotent and robust, handling all network and execution failures gracefully. It was actively used during all phases of development and became an essential part of our workflow — enabling rapid iteration, transparent debugging, and reproducible execution across team members.



Data preparation

ER Diagram

Our ERD is quite simple, because we only have one table with all of the data already in place. We can describe our data fully with only one entity:

Sample of the dataset

Here is a sample of our dataset:

Dataset sample table view

id	start_time	end_time	weather_time_stamp	severity	start_lat	start_lng	distance	delay_from_typical_traffic	delay_from_free_flow_speed	congestion_speed	description	street	city	county	state	country	zip_code
C-9999999	2021-01-20	2021-01-20	N/A	2	39.309849	-76.647362	3.130000114440918	0		3	Moderate	Delays of three minutes and delays easing on US-1 North Ave Westbound between Barclay St and Monroe St. Average speed 15 mph.	W North Ave	Baltimore	Baltimore City	MD	US	21217-1610
C-9999998	2021-01-20	2021-01-20	2021-01-20	0	38.892101	-77.04888199999998	1.8600000143051147	0		2	Slow	Delays of two minutes and delays easing on US-50 Constitution Ave Westbound between US-1 Constitution Ave and Potomac River Fwy. Average speed 15 mph.	Constitution Ave NW	Washington	District of Columbia	DC	US	20037
C-9999997	2021-01-20	2021-01-20	2021-01-20	1	38.890179	-77.421181	1.2699999809265137	1		2	Moderate	Delays of two minutes on US-50 Lee Jackson Memorial Hwy Eastbound between US-50 Lee Jackson Memorial Hwy and US-50 Lee Jackson Memorial Hwy / Lee Jackson Memorial Hwy. Average speed 15 mph.	Lee Jackson Memorial Hwy	Chantilly	Fairfax County	VA	US	20151
C-9999996	2021-01-20	2021-01-20	2021-01-20	0	38.926693	-77.0355	2.880000114440918	0		4	Slow	Delays of four minutes and delays increasing on 15th St Nw Northbound	15th St NW	Washington	District of Columbia	DC	US	20009-8317

This is a screenshot of one of our Superset charts.

Our process

We first download the raw csv file of our data, which we then store in a Postgres DB. After that, we save a compressed version of the data in Parquet format. We chose this format since it is efficient for read operations, which is our primary way of interacting with the data. Having low read times and being able to handle complex data analysis queries was important for this project primarily due to its size.

After EDA, we chose to partition our data by state and bucketing by cities. This sped up training by improving sampling. In addition, these columns were the most “shallow” columns. That is, the number of unique values was quite small, and partitioning/bucketing was applicable without much hardware ramifications.

Data analysis

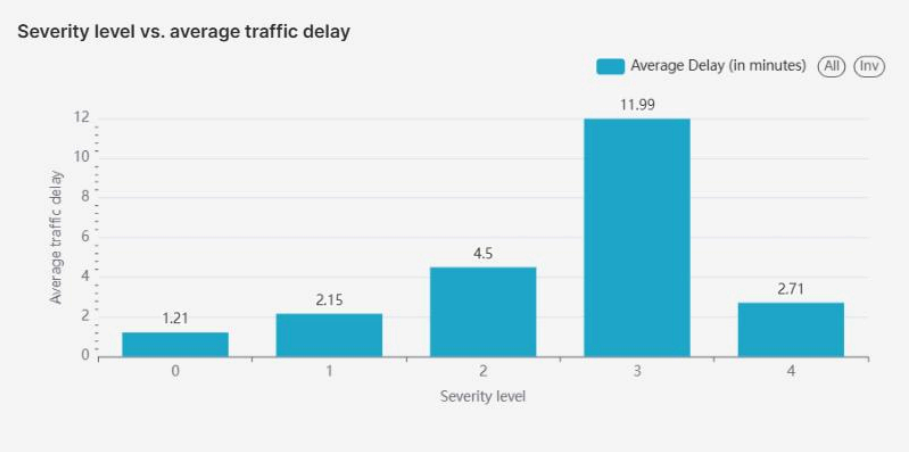
Before feeding the data into the model, it was necessary to perform a thorough exploratory data analysis (EDA). Since our target variable is *delay_from_typical_traffic*, most of the analysis focused on examining its relationships with features that could be useful for prediction. This included assessing how delays vary depending on event severity, weather conditions, geographic location, and temporal patterns such as hour of day, day of week, and seasonality across years.

The primary goals of EDA were to:

- Identify informative and predictive patterns in the data;
- Detect and handle anomalies, outliers, or sparsity issues;
- Guide feature selection and transformation for the downstream ML-pipeline.

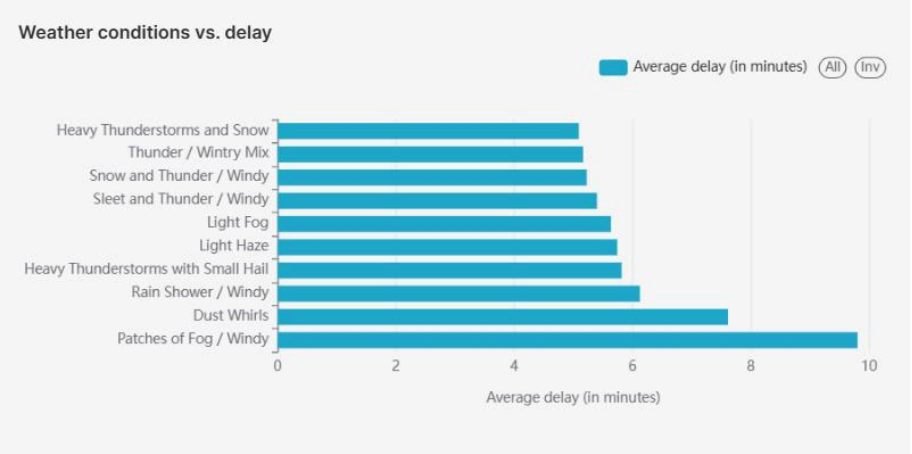
To support this process, our team developed a dedicated set of Hive SQL scripts, which we organized into a separate directory: sql/dashboard. Each script corresponds to a specific analytical query (e.g., severity vs. delay, top congested cities, weather impact, etc.) and was used to generate intermediate datasets as input for dashboard charts. Scripts are automatically detected and executed by our pipeline via the prepare_dashboard.sh script.

Severity level vs. average traffic delay



The chart illustrates the relationship between traffic congestion severity levels and the average traffic delay in minutes. Overall, there is a clear positive correlation: as the severity level increases, the average delay also tends to rise. This trend is especially noticeable at severity level 3, which shows the highest average delay of approximately 12 minutes. Interestingly, severity level 4 breaks this pattern with a significant drop in delay to around 2.7 minutes. This unexpected result can be because level 4 events are rare.

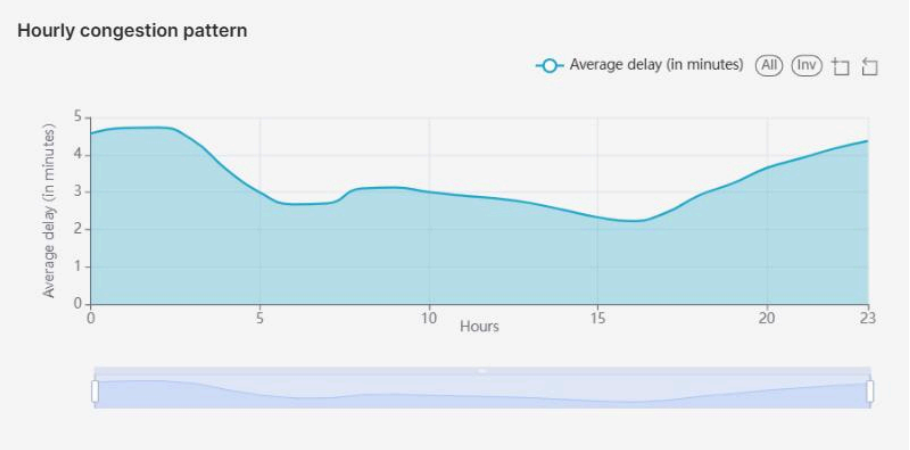
Weather conditions vs. delay



The chart displays the top weather conditions associated with the highest average traffic delays. Most of the conditions involve a combination of precipitation and reduced visibility, such as thunderstorms, snow, fog, and wind. Interestingly, the weather condition that causes the longest delays is "Patches of Fog / Windy", with an average delay close to 10 minutes, significantly higher than the others.

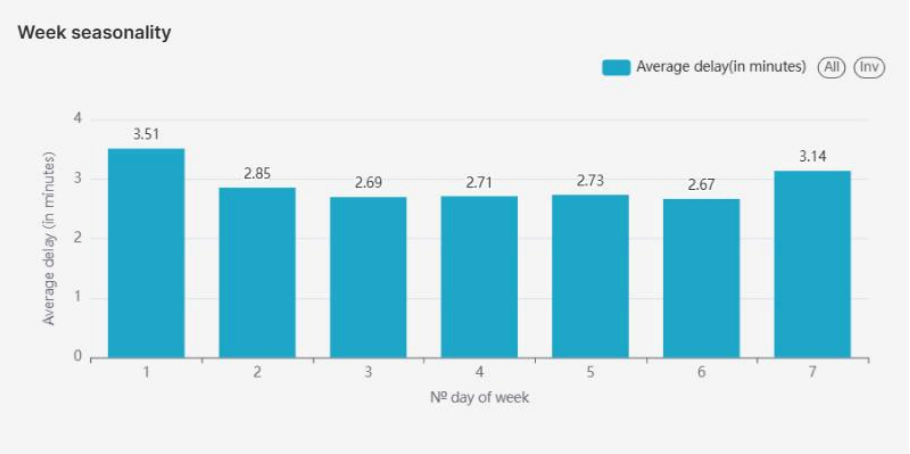
Other high-impact conditions include "Dust Whirls" and "Rain Shower / Windy", suggesting that strong winds and reduced visibility play a critical role in traffic disruption. While more severe-sounding events like "Heavy Thunderstorms and Snow" are on the list, their average delays are actually lower, around 5 minutes, which may indicate better preparedness or quicker response under more expected severe weather conditions.

Hourly congestion pattern



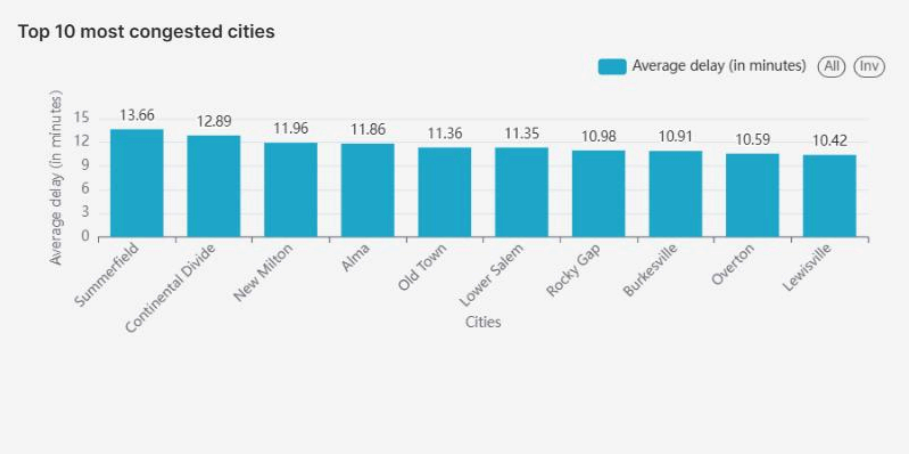
This chart illustrates the average traffic delay by hour of the day. The pattern reveals clear temporal dynamics in congestion, likely linked to daily human activity cycles. Delays are highest just after midnight, remaining elevated until around 2 a.m., which may reflect residual traffic from late-night events or road work.

Week seasonality



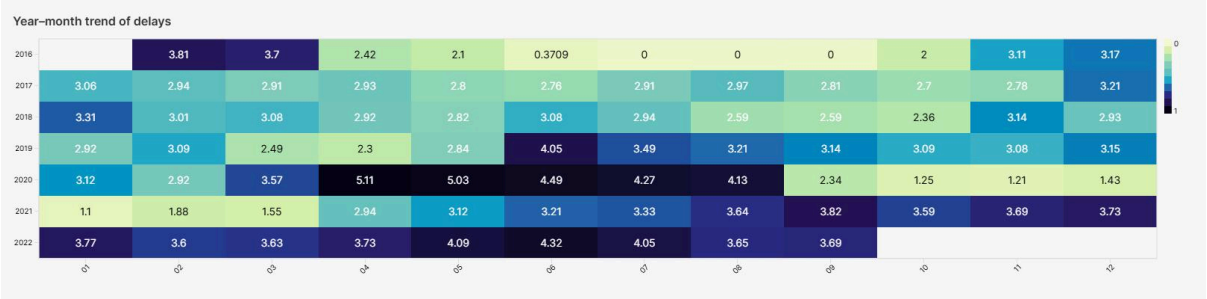
The chart illustrates average traffic delays across the days of the week, and while there are some variations, no single day stands out as an extreme outlier. The most notable is Day 1 (Monday) with the highest average delay of 3.51 minutes, which aligns with expectations — Monday often marks the transition back to routine commuting, potentially leading to higher congestion. Other days, especially Tuesday through Saturday (Days 2–6), show relatively stable and similar delay values, ranging between 2.67 and 2.85 minutes, indicating a consistent traffic pattern during the workweek.

Top 10 most congested cities



The chart presents the top 10 cities with the highest average traffic delays. Summerfield leads the list with an average delay of 13.66 minutes, followed closely by Continental Divide at 12.89 minutes. Although all these cities experience significant delays, the difference between the top and bottom of the list is relatively modest (just over 3 minutes). This suggests that while certain cities like Summerfield are clearly more congested, traffic conditions in the others are comparably strained, pointing to possible shared characteristics such as infrastructure limitations, high travel demand, or recurring local disruptions.

Year–month trend of delays

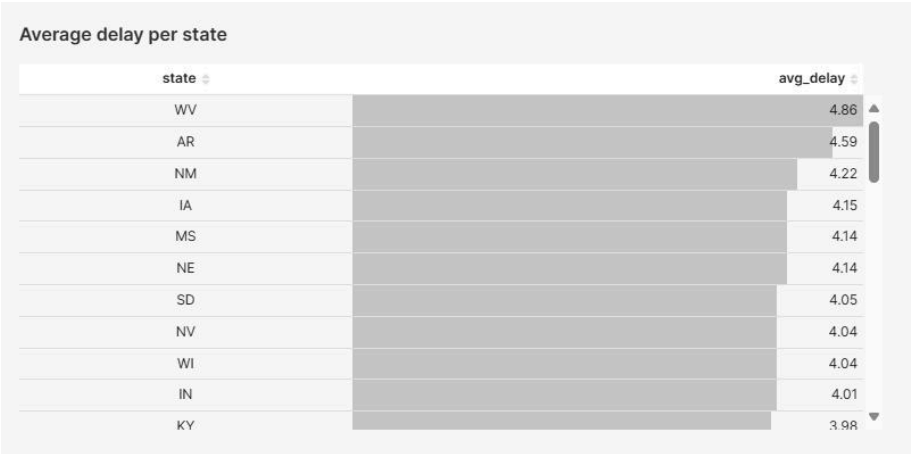


This heatmap visualizes the average traffic delay by month (x-axis) and year (y-axis), revealing temporal patterns and anomalies over time.

A clear standout is 2020, particularly in the months from March to August, where delays peaked across multiple months — with May (05) reaching a high of 5.11 minutes, the highest value on the entire chart. This spike likely reflects the erratic traffic patterns during the early phases of the COVID-19 pandemic, when some regions experienced sudden traffic surges due to policy shifts or road closures.

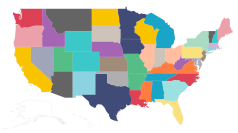
In contrast, 2016 shows several months with zero or near-zero delays (especially June to August), which could be due to missing data or early dataset sparsity. Another anomaly appears in 2021, where early months (January to March) exhibit unusually low delays (as low as 1.1 minutes), potentially reflecting residual pandemic effects or data inconsistencies.

Average delay per state



This chart shows the average traffic delay by U.S. state, with West Virginia (WV) leading at 4.86 minutes, followed by Arkansas (AR) at 4.59 minutes, and New Mexico (NM) at 4.22 minutes. All of the top 10 states listed show average delays above 4 minutes, indicating consistently higher congestion compared to other regions.

Most of the states in this list—such as Iowa (IA), Mississippi (MS), and South Dakota (SD)—are not traditionally associated with high urban traffic density, which suggests that delays may be driven less by city congestion and more by factors like road conditions, weather variability, or limited infrastructure in rural or semi-rural areas.



ML Modeling

To predict traffic delays, we developed a modular Spark ML pipeline optimized for distributed execution on a YARN cluster, leveraging Hive integration for scalable data access.

Data preparation

Data Loading and Sampling

The pipeline reads the Parquet table team26_projectdb.traffic_partitioned from the Hive warehouse. To ensure representative and unbiased subsets for modeling, it samples up to 50,000 rows per state using a window function with random ordering.

Custom Transformers

We implemented several custom PySpark ML transformers for domain-specific feature engineering:

- **CyclicalEncoder**: Encodes temporal features (e.g., month, day, hour, minute) into sine and cosine components to capture cyclical patterns.
- **GeoToECEF**: Converts geodetic coordinates (latitude, longitude, altitude) into Earth-Centered Earth-Fixed (ECEF) Cartesian coordinates for precise spatial representation.
- **TimeDeltaTransformer**: Calculates time differences (in hours) between pairs of timestamp columns (e.g., start time vs. weather timestamp).
- **TimeFeatureExtractor**: Extracts temporal components (year, month, day, hour, minute) from timestamp columns.
- **NAIndicator**: Creates binary indicators for missing values in specified columns, allowing models to account for missingness explicitly.

Data Cleaning and Feature Selection

We removed columns with high missingness, low variability, or potential data leakage, including delay_from_free_flow_speed, end_time, weather_event, zip_code, id, and country. Rows with null values in the target column delay_from_typical_traffic were excluded. Missing text in the description field was replaced with empty strings to avoid null handling issues.

Feature Engineering

- **Missing Value Indicators**: Binary flags for missing values in wind_chill and precipitation.
- **Categorical Features**:

    - Low cardinality (e.g., severity, congestion_speed, state, wind_dir) → StringIndexer + OneHotEncoder.
    - High cardinality (e.g., county, city, weather_station_airport_code, weather_conditions) → FeatureHasher (16 hashed features).
    - Ultra-high cardinality (street) → FeatureHasher (32 hashed features).
- **Temporal Features**: Extracted components from start_time (year, month, day, hour, minute) and applied cyclical encoding to periodic components.
- **Geospatial Features**: Converted latitude and longitude to ECEF coordinates.
- **Text Features**: Processed description field via tokenization, stopword removal, and Word2Vec embedding (vector size 64, minimum count 100).

Imputation

Continuous features were imputed using the mean or median, depending on the extent of missingness. The start_year was imputed using mode imputation.

Feature Assembly

All engineered features — including missing indicators, encoded categorical variables, temporal and geospatial features, text embeddings, and imputed continuous variables — were combined into a single feature vector using Spark's VectorAssembler.

Pipeline Construction and Execution

The full modeling pipeline was constructed by combining all preprocessing, transformation, and feature assembly steps into a unified Pipeline object. The pipeline was fitted on the cleaned dataset, then the transformed data was split into training (80%) and testing (20%) subsets. Final feature and target datasets were saved in JSON format for downstream modeling and evaluation.

ML Modeling Pipeline Details

The Spark ML pipeline was designed to train, tune, and evaluate two regression models — Random Forest and Linear Regression — on the preprocessed traffic delay data.

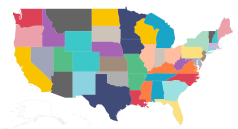
Spark Session Setup

The system initializes a SparkSession with Hive support, configuring adaptive query execution, shuffle optimizations, and dynamic resource allocation for efficient cluster use.

Data Loading

Training and test datasets are loaded from JSON files, each containing:

- features: dense feature vectors from prior engineering.



- delay_from_typical_traffic: continuous target variable to predict.

Modeling and Tuning

- **Random Forest Regressor**:
  - Parameter grid: maxDepth [5, 10]; numTrees [20, 50].
  - Trained using 5-fold cross-validation (CrossValidator), parallelized across 4 threads.
- **Linear Regression**:
  - Parameter grid: regParam [0.01, 0.1]; elasticNetParam [0.0, 0.5].
  - Also tuned with 5-fold cross-validation.

Model Selection and Saving

During the model training phase, we performed cross-validation to identify the best-performing hyperparameters for each algorithm. The best model from each run was saved to the `project/models` directory.

- **Random Forest Regressor**:
  - maxDepth 10;
  - numTrees 50.

- **Linear Regression**:
  - regParam 0.01;
  - elasticNetParam 0.0 (pure L2 regularization)

Prediction and Output

Each best model is applied to the test dataset. Predictions (true vs. predicted values) are saved as CSV files in project/output.

Evaluation Metrics

We compute standard regression metrics, including:

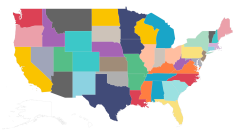
- Root Mean Square Error (RMSE)
- Coefficient of Determination (R^2)
- Mean Absolute Error (MAE)

Results are summarized in a comparison table stored in the project/output/evaluation directory.

Cluster Efficiency

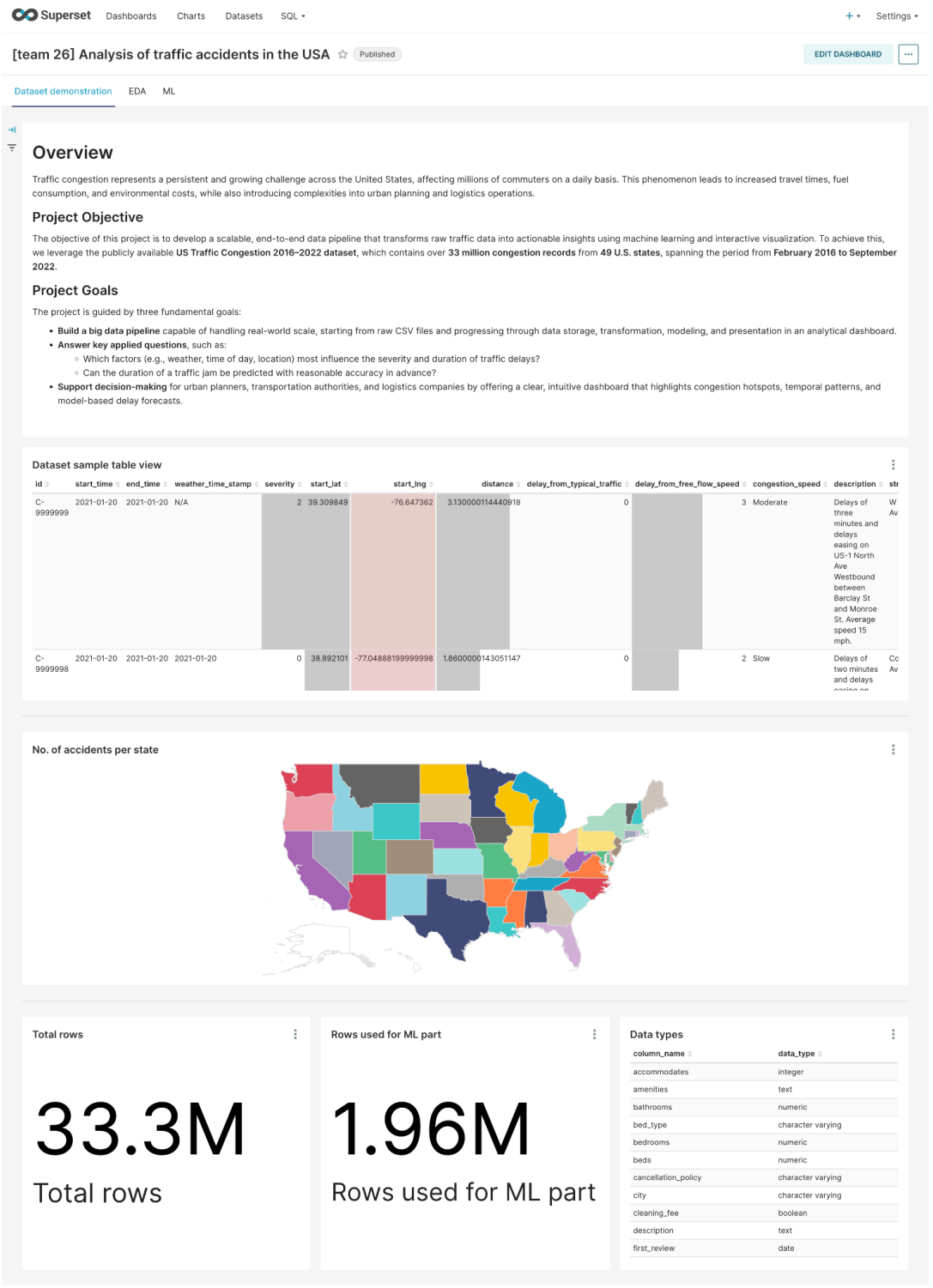
The system leverages Spark's parallelism, cross-validation parallelism, adaptive query execution, and optimized storage formats to maximize resource efficiency and accelerate model selection.

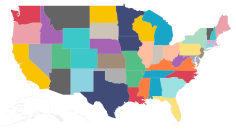
Model	RMSE	R^2	MAE
Random Forest	2.9146	0.5787	1.5438
Linear Regression	3.3102	0.4566	1.8117



Data presentation

Dataset demonstration

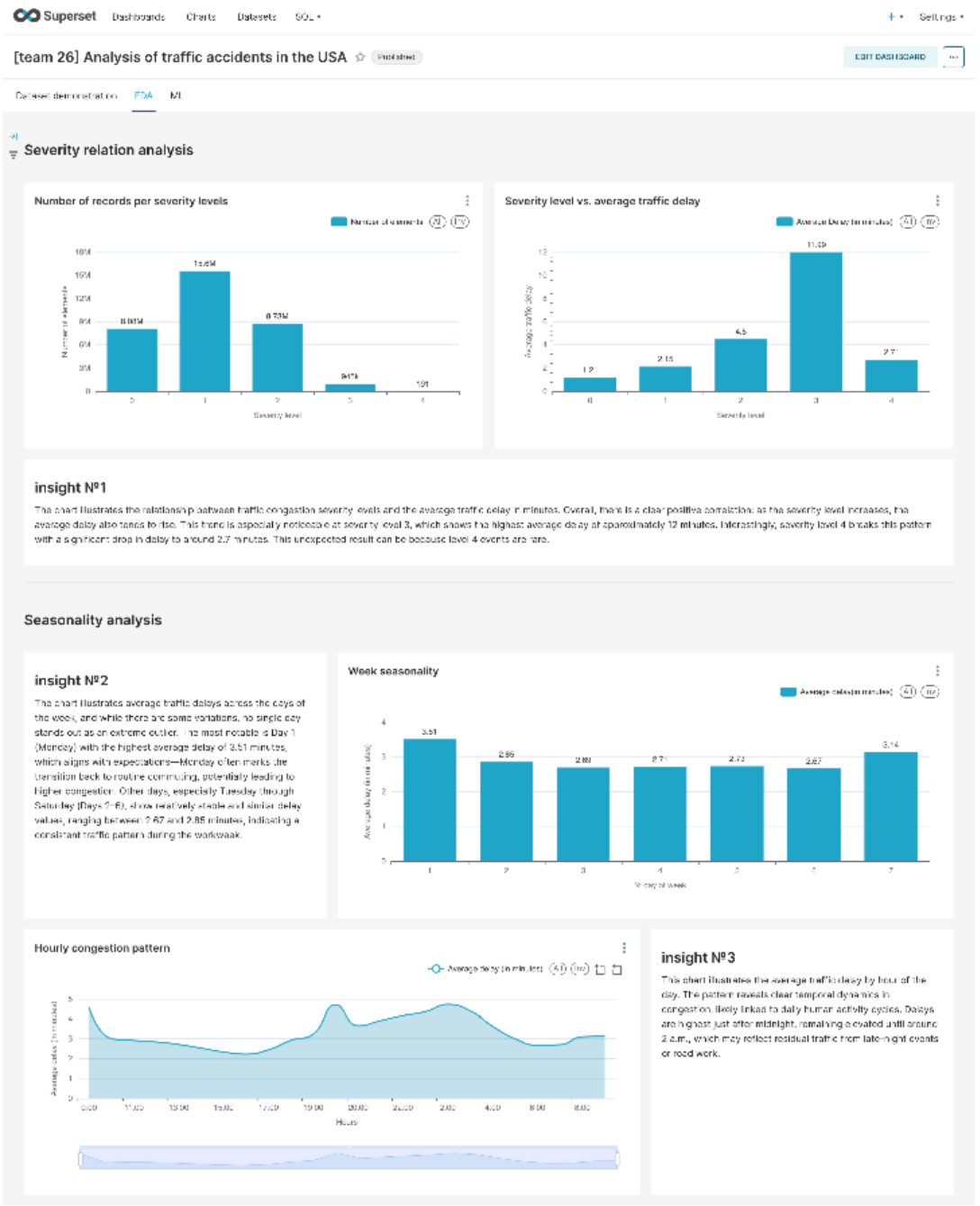


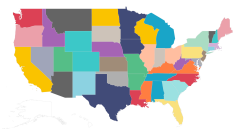


This section provides a high-level overview of the dataset. It includes:

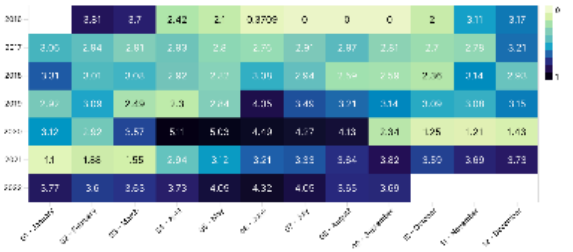
- A sample table view of the dataset to understand its structure and contents.
- A heatmap of the number of accidents per state, visualizing geographic distribution.
- Key metrics such as the total number of records, number of records used for training, and a summary of data types present in the dataset.

EDA





Year–month trend of delays

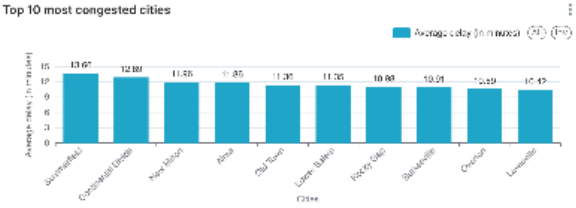


insight №4

This heatmap visualizes the average traffic delay by month (x-axis) and year (y-axis), revealing temporal patterns and anomalies over time. A clear standout is 2023, particularly in the months from March to August, where delays peaked across multiple months — with May (05) reaching a high of 5.1 minutes, the highest value on the entire chart. This spike likely reflects the erratic traffic patterns during the early phases of the COVID-19 pandemic, when some regions experienced sudden traffic surges due to policy shifts or road closures. In contrast, 2016 shows several months with zero or near-zero delays (especially June to August), which could be due to missing data or early dataset sparsity. Another anomaly appears in 2021, where early months (January to March) exhibit unusually low delays (as low as 1.1 minutes), potentially reflecting residual pandemic effects or data inconsistencies.

Location relation analysis

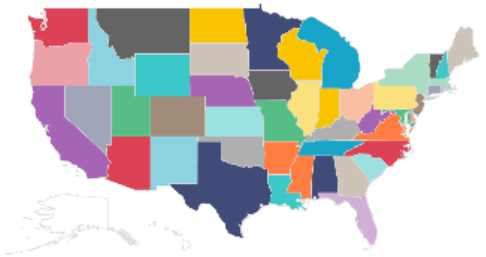
Top 10 most congested cities



insight №5

The chart presents the top 10 cities with the highest average traffic delays. Summerfield leads the list with an average delay of 13.88 minutes, followed closely by Centennial Divide at 12.88 minutes. Although all these cities experience significant delays, the difference between the top and bottom of the list is relatively modest (just over 3 minutes). This suggests that while certain cities like Summerfield are clearly more congested, traffic conditions in the others are comparably strained, pointing to possible shared characteristics such as infrastructure limitations, high travel demand, or recurring local disruptions.

Average delay per state



Average delay per state

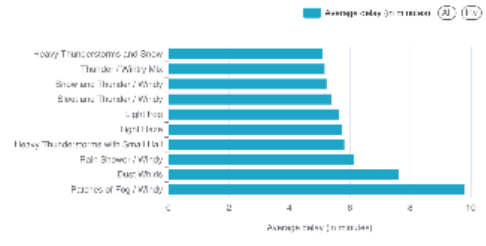
state	avg_delay
WV	4.88
AR	4.59
NM	4.22
IA	4.15
MS	4.14
NE	4.14
SD	4.05
NV	4.00
WI	4.00
IN	4.0
KY	3.98

insight №6

This chart shows the average traffic delay by U.S. state, with West Virginia (WV) leading at 4.88 minutes, followed by Arkansas (AR) at 4.59 minutes, and New Mexico (NM) at 4.22 minutes. All of the top 10 states listed show average delays above 4 minutes, indicating consistently higher congestion compared to other regions. Most of the states in this list — such as Iowa (IA), Mississippi (MS), and South Dakota (SD) — are not traditionally associated with high urban traffic density, which suggests that delays may be driven less by city congestion and more by factors like road conditions, weather variability, or limited infrastructure in rural or semi-rural areas.

Weather conditions relation analysis

Weather conditions vs. delay



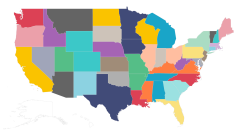
Average delay per weather condition

Search 175 records...

weather_conditions	avg_delay
Patches of Fog / Windy	9.8
Dust Whirls	7.82
Rain Shower / Windy	6.15
Heavy Thunderstorms with Small Hail	5.82
Light Snow	5.75
Light Fog	5.64
Blowing Snow	5.4
Snow and Thunder / Windy	5.29
Thunder / Wintry Mix	5.16
Heavy Thunderstorms and Snow	5.08

insight №7

The chart displays the top weather conditions associated with the highest average traffic delays. Most of the conditions involve a combination of precipitation and reduced visibility, such as thunderstorms, snow, fog, and wind. Interestingly, the weather condition that causes the longest delays is "Patches of Fog / Windy", with an average delay close to 10 minutes, significantly higher than the others. Other high-impact conditions include "Dust Whirls" and "Rain Shower / Windy", suggesting that strong winds and reduced visibility play a critical role in traffic disruption. While more severe-sounding events like "Heavy Thunderstorms and Snow" are on the list, their average delays are actually lower, around 5 minutes, which may indicate better preparedness or quicker response under more expected severe weather conditions.



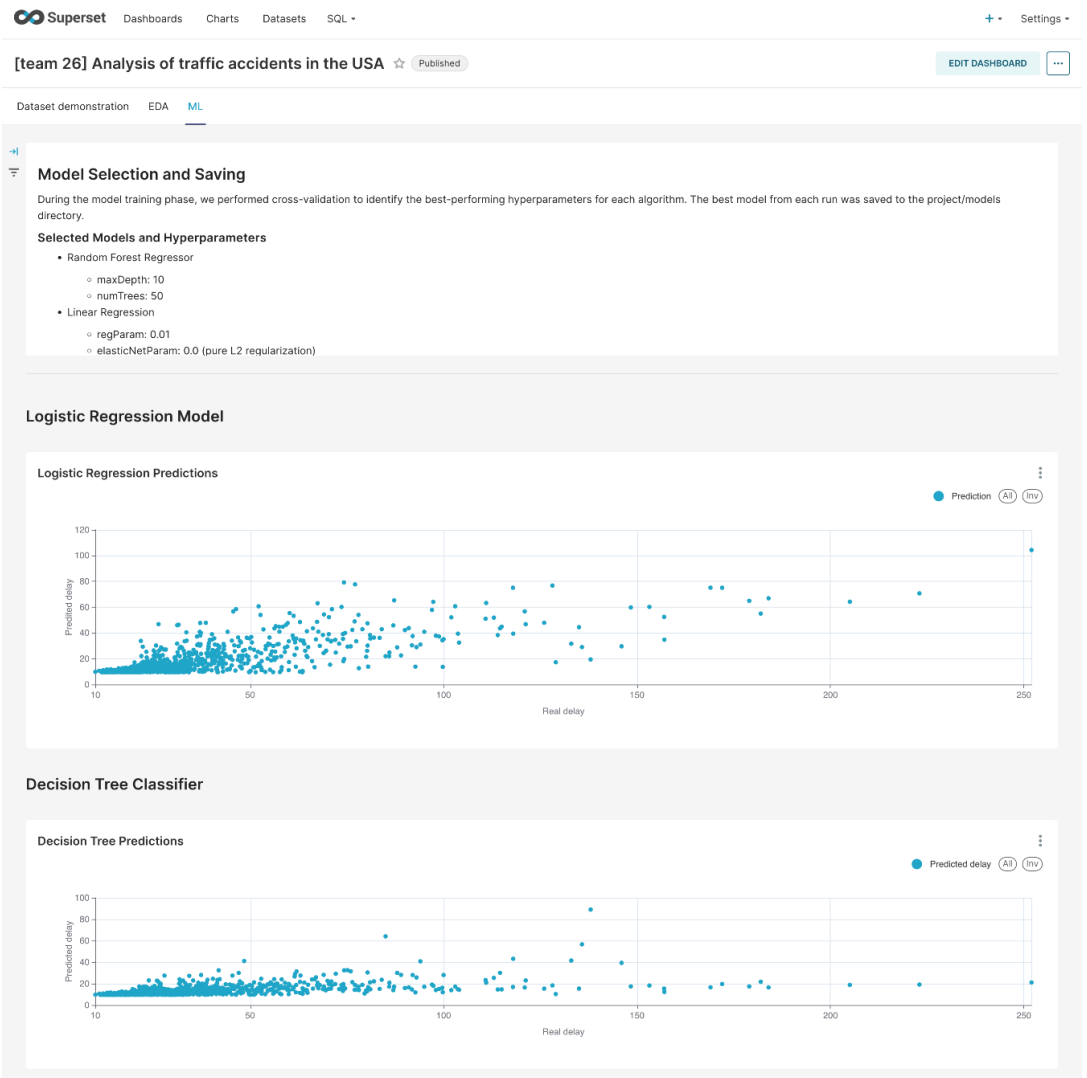
This section contains all charts developed during the data analysis phase. The goal is to uncover patterns, correlations, and trends that can inform feature engineering and model building. For better readability and structure, this section is further grouped into thematic blocks such as:

- Time-based analysis
- Weather impact
- Severity levels
- Geospatial distribution

Detailed descriptions of each chart are provided in the corresponding Data Analysis section of the report.

ML

ML dashboard showcases predictions for both of our models:



Comparison of Models

team26 ml results

model	rmse	r2	mae
Random Forest	2.91	0.5787	1.54
Linear Regression	3.31	0.4566	1.81

Results analysis

Model Evaluation Metrics

RMSE and MAE (in minutes)

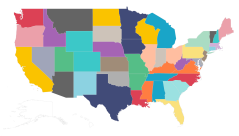
- RMSE
  - ~2.91 for Random Forest
  - ~3.31 for Linear Regression

  This means that, on average, the model's predictions deviate from the actual delay by approximately 3 minutes.
- MAE
  - ~1.54 for Random Forest
  - ~1.81 for Linear Regression

  This represents the average absolute error, offering a more intuitive measure of typical prediction error, around 1.5 to 1.8 minutes.

R^2 (Explained Variance)

- Random Forest explains about 58% of the variability in traffic delay.
  This is a solid result for a traffic prediction task, considering that delays can be influenced by unpredictable factors like accidents, weather, and local events.
- Linear Regression explains approximately 46% of the variance, indicating it captures less of the underlying complexity in the data.



Conclusion

This project delivered a complete data pipeline that ingests, processes, analyzes, and models large-scale traffic congestion data from across the United States. We transformed over **33 million** raw traffic records into structured analytical insights using distributed technologies such as PostgreSQL, Hive, Spark MLlib, and Apache Superset.

Our contributions include:

- End-to-end automation for data ingestion, partitioning, and transformation;
- Targeted HQL queries for exploratory data analysis (EDA);
- A trained machine learning model predicting traffic delays using geographic, temporal, and weather-related features;
- An interactive Superset dashboard for intuitive data visualization and decision support.

By combining robust engineering with practical analytics and visualization, we created a scalable platform for understanding and forecasting traffic congestion. This solution provides city planners, traffic authorities, and transportation analysts with tools to identify critical congestion zones, study temporal patterns, and anticipate delays more accurately.

Reflections on own work

Beyond the technical implementation, the project exemplified effective collaboration: team members contributed across domains, supported each other in debugging and integration, and collectively built a reproducible, insightful traffic analytics platform.

Challenges and Difficulties

- **Cluster resource limitations**: Due to the size of our dataset (the largest among all teams), memory and CPU constraints often led to job failures and timeouts — especially during model training on full data. As a workaround, we implemented per-state sampling (50,000 rows per state) to preserve representativeness while ensuring performance.
- **Access restrictions to the cluster**: Off-campus access was not possible due to firewall rules. To address this, we created a custom CI script (ci-watch.sh) that ran directly on the cluster, continuously monitoring the deploy branch and running pipeline jobs automatically, with live updates delivered to Telegram.
- **Data transfer bottlenecks**: Transferring data between PostgreSQL (Stage I) and Hive (Stage II) proved to be non-trivial. Ensuring compatibility in types, partitioning logic, and avoiding data duplication required careful tuning and debugging.

Recommendations

- **Optimize for scale early**: Teams working with large datasets should plan their data partitioning, compression, and sampling strategies from the beginning — especially when memory resources are limited.
- **Automate everything**: Introducing a custom CI/CD solution drastically improved our feedback loop, enabling safe testing, deployment, and log inspection without manual effort. We strongly recommend CI integration even for academic projects.
- **Collaborate beyond roles**: Despite clear role distribution, we frequently helped each other across tasks — from setting up dashboards to debugging Hive scripts. This cross-functional support proved essential for maintaining momentum and quality throughout the project.

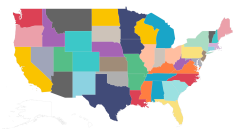
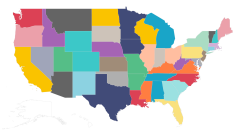


Table of contribution

Project Tasks	Task description	Alexey Tkachenko	Daniil Abronisimov	Egor Machnev	Apollinaria Chernikova	Deliverables	Total hours spent
Data collection	Collect the data from the Kaggle and extract it on the cluster	80%	0%	10%	10%	data_collection.sh + tests for reliability + lints	3
Data storage	Copy the data to Postgres on the cluster	80%	0%	10%	10%	data_storage.sh create_tables.sql import_database.sql test_database.sql + tests for reliability + lints	2
Data ingestion	Copy the data from Postgres into HDFS as compressed PARQUET	60%	30%	10%	10%	data_ingestion.sh + tests for reliability + lints	5
Data partitioning	Partition the data for ML part to speed up training	40%	60%	0%	0%	load_state.hql + bug fixes all around	7
CI/CD	Create a CI/CD pipeline to run the stages on every push to a branch on GitHub	0%	0%	100%	0%	Telegram bot that runs the pipeline and sends the results and outputs in TG group chat	10
Hypothesis formulation	Create initial hypothesis about the data from dataset description from Kaggle	0%	0%	75%	25%	Lineup of possible hypothesis	3
Hypothesis testing	Explore the data by executing hql queries	0%	0%	60%	40%	q1-14.sql q1-14.hql + results	10
Result analysis	Analyze the results of EDA	0%	0%	60%	40%	ML part understood what features to use	1
ML feature engineering	Apply EDA results for feature engineering	30%	70%	0%	0%	Feature transformations	10



Project Tasks	Task description	Alexey Tkachenko	Daniil Abronisimov	Egor Machnev	Apollinaria Chernikova	Deliverables	Total hours spent
ML model selection	Knowing the features, the hardware limitations, and the time constraints, select the appropriate ML models	0%	100%	0%	0%	Two model were selected: Linear regression and Random forest	2
Hyperparameter grid search setup	Find the best hyperparameters for all models	0%	100%	0%	0%	Hyperparameter grid search is done and integrated into the code	10
Chart creation	Create charts for the dashboard	30%	10%	0%	50%	Superset charts	6
Dashboard setup	Create the dashboard with charts displaying all of the necessary info	30%	0%	0%	70%	Final Superset charts and dashboards	1
Report writing	Write a comprehensive report, detailing our work during this project	25%	25%	25%	25%	The report you are viewing ;)	4