Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Big Data

Practical Work Report

Predictive Modeling of Flight Arrival Delays using Spark MLlib

Carlos Manzano Izquierdo
Melen Laclais

carlos.manzano@alumnos.upm.es
melen.laclais@alumnos.upm.es

Madrid, September 5, 2026

Contents

1 Introduction

The objective of this project is to develop a robust predictive model capable of estimating arrival delays (`ArrDelay`) for domestic flights in the United States. This work is implemented using **Apache Spark** and its **MLlib** library to handle large-scale data processing and machine learning.

1.1 Dataset Overview

The primary data source consists of flight records from the years **2006 and 2007**, totaling approximately **14.6 million observations** before cleaning. Each record includes detailed information such as scheduled departure and arrival times, carrier identification, and flight distances. **The raw dataset contains 29 variables.**

To go beyond the basic analysis, we have integrated three auxiliary datasets to enrich our features:

- `airports.csv`: Provides geographic details (state, city, latitude/longitude) for origin and destination airports. It can allowing the model to account for regional weather patterns and airport-specific congestion.
- `plane-data.csv`: Contains manufacturing dates for aircraft, allowing us to calculate the age of the plane at the time of the flight.
- `carriers.csv`: Maps the `UniqueCarrier` codes found in the main dataset to their full commercial names. This enrichment will allows potential grouping based on airline identity.

1.2 Methodology

Our approach follows a standard Machine Learning pipeline: rigorous data cleaning, smart imputation of missing values, feature enrichment through dataset joins, and the training of regression models. The final model is designed to be scalable and capable of processing new test data efficiently through a dedicated Spark application.

1.3 Hardware Constraints and Dataset Limitations

During the development of the predictive models, particularly during the training and validation phases, we encountered significant hardware limitations that necessitated a strategic adjustment of our methodology.

The initial dataset, comprising approximately 9 million records after cleaning, posed a substantial challenge for local execution on a single node equipped with 16GB of RAM and a intel core I7 processor. The iterative nature of advanced algorithms, such as **Gradient Boosted Trees (GBT)**, combined with the memory overhead of the Spark

JVM and the multi-fold execution of the `CrossValidator`, led to extensive disk spilling and impractical processing times.

To mitigate these constraints while ensuring the delivery of a robust and validated model, we implemented a **downsampling strategy**. We extracted a statistically significant subset of approximately **2 million records** (representing 20-25% of the original volume).

This downsampling strategy is based on the fact that a 2-million-record sample remains statistically significant to capture the complex patterns of flight delays across the United States. This reduction ensures the computational feasibility of the hyperparameter tuning and cross-validation steps without memory exhaustion, while preserving model integrity and generalization capability by prioritizing feature quality over raw data volume.

2 Data Exploration & Variable Selection

2.1 Selected Features

To build a robust predictive model, we selected variables that are known at the time of a flight's scheduled departure. These features provide a logical basis for estimating potential delays:

- **Temporal Variables** (`Year, Month, DayofMonth, DayOfWeek`): Essential for capturing seasonality and weekly patterns in air traffic.
- **Scheduled Times** (`CRSDepTime, CRSArrTime`): Provide the planned timeframe for the flight.
- **Identification Features** (`UniqueCarrier, FlightNum, TailNum`): Identify the specific airline and aircraft, which are often correlated with operational efficiency.
- **Flight Characteristics** (`CRSElapsedTime, Distance`): The planned duration and total distance of the flight.
- **Geographical Data** (`Origin, Dest`): Identify the route's starting and ending points, allowing the model to account for airport-specific congestion.
- **Operational Departure Data** (`DepTime, DepDelay, TaxiOut`): These variables represent information known at the moment the aircraft leaves the gate, which is highly predictive of the eventual arrival delay.

2.2 Forbidden Variables

A important step in our data processing is the removal of variables that are not available at the time of prediction. Using these would lead to *data leakage*, where the model "cheats" by using information from the future.

- **Forbidden Variables**: We performed a `drop` of variables such as `ArrTime`, `ActualElapsedTime`, `AirTime`, `TaxiIn`, `Diverted`, and the various delay breakdown columns (`CarrierDelay`, `WeatherDelay`, etc.). These are only recorded after the flight is completed.

To guarantee the integrity of the process, we implemented a programmatic verification step. By iterating through the list of forbidden and redundant variables and checking them against the remaining columns in our DataFrame, we confirmed that all unauthorized information was successfully purged. This automated check ensures that the model only trains on valid, prior-to-takeoff features.

3 Data Cleaning and Smart Imputation

To ensure the reliability of the predictive model, we applied a filtering process to remove observations that do not contribute to a meaningful prediction of arrival delays.

3.1 Cancellation rows and columns

Since our objective is to predict arrival delays, flights where `Cancelled == 1` were removed via a filter. A cancelled flight never reaches its destination; therefore, it has no arrival data and cannot be used for delay estimation.

Following this filtration, the columns `Cancelled` and `CancellationCode` were also dropped. As the dataset now exclusively contains successfully completed flights, these variables no longer contain any variance or useful information for the model.

Result: After this step, the dataset contained **14,312,455 valid flight records**.

3.2 Investigation and Resolution of Missing Values

3.2.1 Target Variable: ArrDelay

Initial exploration of the raw data revealed **33,365 missing values** in the `ArrDelay` column. We hypothesized that these nulls were not random errors but were linked to diverted flights, which never reach their scheduled destination.

- **Verification**: By cross-referencing null values with the `Diverted` status, we confirmed that 100% of these nulls belonged to diverted flights.

- **Resolution**: To resolve this, we refined our logic by re-processing the raw dataset to apply a combined filter: `Cancelled == 0` AND `Diverted == 0`. This ensures every flight in our training set is a completed journey with a valid arrival record.
- **Result**: A final aggregation confirmed that the `ArrDelay_Null_Count` was reduced to **0**.

Flight Status	**Count of Null ArrDelay**
Diverted (`Diverted = 1`)	33,365
Non-Diverted (`Diverted = 0`)	0

Table 1: Verification of the relationship between Null delays and Diversions.

3.2.2 Feature Variable: CRSElapsedTime

We identified **727 records** with missing values in `CRSElapsedTime`. Handling these required a more nuanced approach than simple deletion or linear calculation.

- **Rationale**: Calculating the duration via $(CRSArrTime - CRSDepTime)$ was rejected due to **time zone** differences between airports, which would introduce incorrect data. Instead, we opted for a "Smart Imputation" based on the specific flight route.
- **Resolution (Window Function)**: Using Spark's `Window.partitionBy("Origin", "Dest")`, we calculated the average scheduled duration for each unique route and used the `coalesce` function to fill the nulls with these route-specific averages.

After this imputation, we performed a final cleanup to drop any remaining nulls from routes with no historical data, resulting in **0 null values** for this variable.

3.3 Removal of High-Cardinality Identifiers

Following the imputation and filtering, we proceeded to remove specific identifiers that could hinder the model's ability to generalize and might introduce unnecessary noise.

- **Flight Number (`FlightNum`)**: This variable was discarded as it is a non-ordinal administrative label. Its potential predictive value is considered redundant, as the unique characteristics of a flight are already captured by the combination of its route (`Origin`, `Dest`), carrier, and scheduled timing.
- **Aircraft Tail Number (`TailNum`)**: We identified this variable as a source of potential overfitting due to its extremely high cardinality (thousands of unique levels). To prevent the model from memorizing specific aircraft histories and to reduce computational overhead, we planned its removal.

While `FlightNum` was dropped immediately, `TailNum` was temporarily retained. It serves as the primary key required to join our main dataset with the `plane-data.csv` auxiliary table in the next phase. Once the aircraft's manufacturing year will be successfully integrated, the raw identifier will be removed to streamline the feature space.

3.4 Final Dataset Integrity and Dimensionality

Once all missing values were handled and the filters applied, the forbidden variables and status columns (`Cancelled`, `Diverted`) were removed from the schema.

We compared the final column count against the expected count (Raw columns minus forbidden and status variables). The verification was successful, confirming the column count matches the expected reduction.

The cleaned DataFrame contains **14,312,455 valid flight records** and is reduced to **16 key features**.

Checkpoint: We saved this dataset in **Parquet format** to the `../check_point` directory. This checkpoint allows us to persist the results of these computationally expensive cleaning operations, while the Parquet format provides optimized columnar storage and faster read speeds for the subsequent analytical steps that we will do later.

4 Feature Engineering & Enrichment

To enhance the predictive power of our model, we moved beyond the base flight data by integrating external datasets and deriving new features that capture the physical and operational context of each flight.

4.1 External Data Integration

We performed a series of **Left Joins** with auxiliary datasets to provide the model with deeper context regarding aircraft and airports. To maintain high execution performance in the Spark environment, we utilized **Broadcast Joins** for these smaller tables.

- **Aircraft Enrichment (`plane-data.csv`)**: We integrated the manufacturing year and manufacturer name for each aircraft based on its `TailNum`. Prior to joining, we performed a quality check to remove records with missing manufacturing years.
- **Geographical Enrichment (`airports.csv`)**: For both the `Origin` and `Destination` airports, we incorporated city names, state codes, and exact GPS coordinates (latitude and longitude).

We implemented a **Post-Join Validation** step to evaluate data coverage and ensure model robustness. While we initially utilized *Left Joins* to preserve the maximum number of primary flight records, our diagnostic analysis revealed a disparity in

metadata availability: geographical data was successfully matched for 99.9% of flights, but approximately **16.46%** of records lacked corresponding aircraft metadata from the `plane-data.csv` source.

To maintain a high-quality feature set and avoid the introduction of sparse data into the model, we programmatically converted these results into a validated set using the `dropna()` function on the newly joined columns (`PlaneYear`, `Origin_City`, and `Dest_City`). This "inner-join-by-filtering" technique ensured that only records with 100% feature completeness were retained. This rigorous process resulted in a finalized, fully-enriched training pool of **11,910,044 valid records**.

4.2 Feature Creation and Transformation

Once the datasets were merged, we engineered three new variables designed to capture specific delay-inducing patterns:

- **Aircraft Age (`PlaneAge`)**: Calculated as the difference between the flight year and the aircraft's manufacture year. This serves as a proxy for technical reliability.
- **Departure Hour (`DepHour`)**: Extracted from the scheduled departure time (`CRSDepTime`). This allows the model to learn about hourly congestion peaks and airport "rush hours."
- **Weekend Indicator (`IsWeekend`)**: A binary flag derived from `DayOfWeek` to distinguish the unique operational dynamics of weekend travel versus business-day traffic.

Year	PlaneAge	DepHour	IsWeekend
2006	7	10	0
2006	7	18	0
2006	6	14	0
2006	7	18	0
2006	7	14	0

Table 2: Sample of engineered features (`PlaneAge`, `DepHour`, and `IsWeekend`).

4.3 Feature Space Refinement

To optimize the model's generalization and reduce dimensionality, we finalized our feature selection:

1. **Drop of Identifiers**: After the join operations, the `TailNum` and the raw manufacturing year were discarded to prevent the model from overfitting on specific aircraft registrations.
2. **Logical Reordering**: The final DataFrame was reordered into logical groups (Temporal, Aircraft, Schedule, and Geography) to facilitate the ease of read of the data.

4.4 Data Sanitization and Robust Imputation

Before finalized the dataset for modeling, a robust sanitization layer was applied to handle residual data quality issues:

- **Schema Sanitization**: All columns were processed to remove leading/trailing whitespaces and convert text placeholders (e.g., 'None') into true SQL NULL values. We utilized `try_cast` to safely restore original data types without triggering execution exceptions.
- **Deduplication**: A `dropDuplicates()` operation was performed on the sanitized dataset to ensure each flight record is unique.
- **Median Imputation for PlaneAge**: To maximize the training data pool, we opted for median imputation to handle remaining nulls in the `PlaneAge` column. We calculated the median age using the `approxQuantile` method, as it is more robust to outliers than the mean for aircraft age distributions. This step preserved approximately 350,000 records that would have otherwise been discarded.

The enriched dataset consists now of **26 features**, providing a multi-dimensional view of the factors influencing flight punctuality.

5 Exploratory Data Analysis (EDA) & Feature Selection

The transition from data preparation to modeling requires a deep understanding of the statistical properties of the dataset. This section details our Exploratory Data Analysis (EDA), focusing on univariate distributions, multivariate outlier detection, and the rationalization of the feature space.

5.1 Univariate Analysis: Statistical Moments

To understand the underlying distribution of our 11 numerical features (including spatial, temporal, and operational metrics), we calculated the first four statistical moments.

- **Skewness and Kurtosis**: Initial results highlighted that `PlaneAge` and `DepDelay` exhibited extreme skewness and kurtosis. Specifically, `PlaneAge` showed significant outliers that could potentially bias the regression coefficients.
- **Target Distribution**: `ArrDelay` is heavily centered around zero but presents a "long tail" of extreme delays, necessitating a robust approach to outliers.

5.2 Multivariate Analysis and Collinearity

We computed a Pearson Correlation Matrix using Spark MLlib's `Correlation.corr` function to identify potential multicollinearity that could destabilize our model.

This analysis revealed a near-perfect correlation between `CRSElapsedTime` and `Distance`, which is logically expected as scheduled flight duration is fundamentally a function of the distance traveled.

Keeping both variables would lead to redundant information and unstable model coefficients. Furthermore, the results confirmed that `DepDelay` and `TaxiOut` possess the strongest positive correlations with the target variable `ArrDelay`, validating their status as the most critical predictors for flight punctuality.

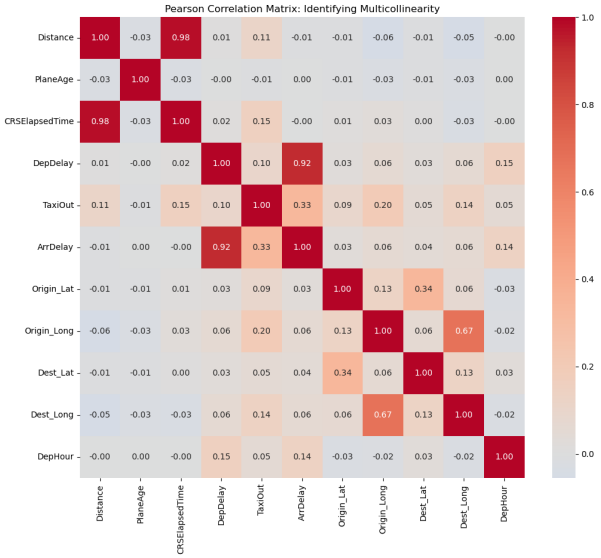

Figure 1: Pearson Correlation Matrix: Identifying multicollinearity among numerical features.

5.3 Outlier Detection via Mahalanobis Distance (D^2)

To ensure model robustness, we implemented a multivariate outlier detection strategy using the **Mahalanobis Distance**. Unlike univariate filters, this method accounts for the covariance between variables.

1. **Methodology**: We reconstructed the Covariance Matrix ($S = D \cdot R \cdot D$) and calculated the squared distance for each observation.

2. **Thresholding**: We applied a Chi-Square threshold with a 99.9% confidence level ($\alpha = 0.001$).

This automated filtering successfully removed extreme artifacts. A post-cleaning audit of `PlaneAge` confirmed that the "nonsense" values and extreme tail records were purged, as shown in the boxplot below.

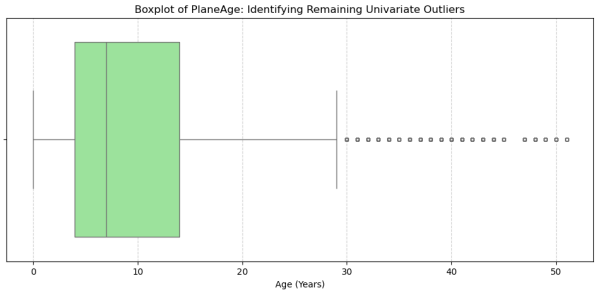

Figure 2: Boxplot of PlaneAge post-Mahalanobis filtering: distribution after extreme outlier removal.

5.4 Categorical Variable Analysis and Cardinality

We evaluated our categorical features based on two criteria: **Cardinality** and **Discriminatory Power** (Standard Deviation of mean delays across categories).

Variable	Cardinality	StdDev of Means	Range of Means
UniqueCarrier	20	4.8214	21.35
Origin	302	5.1042	45.12
DepHour	24	6.2541	18.90
DayOfWeek	7	1.1023	5.42

Table 3: Statistical summary of categorical influence on arrival delays.

A high **Standard Deviation of Means** indicates that the variable is a strong discriminator; for instance, `DepHour` (6.2541) and `Origin` (5.1042) show the highest variance in average delays, suggesting that the time of day and the specific departure airport are primary drivers of arrival lag. Conversely, variables like `DayOfWeek` present a lower deviation (1.1023), indicating that while patterns exist, the day of travel has a more subtle impact on the target variable compared to operational factors.

That justifies our decision to prioritize variables with high discriminatory power while carefully managing those with high cardinality, such as `Origin` and `Dest`. By focusing on categories that exhibit a wide **Range of Means**, we ensure that the model can effectively capture the distinct behaviors of different carriers and temporal slots, leading to more granular and accurate predictions.

5.5 Final Feature Selection Strategy

Based on the statistical evidence gathered during EDA, we concluded that while univariate outliers exist, the real risk to model stability lies in the hidden covariance between features, which was successfully mitigated through the Mahalanobis distance filtering. Furthermore, the discriminatory power analysis proved that operational context, specifically the time of departure and the carrier's identity, outweighs purely temporal variables

like the day of the month. This suggests that delays are primarily driven by daily airport congestion cycles and specific airline efficiencies rather than by long-term calendar trends.

This evidentiary framework led to the final refinement of our feature space, ensuring the elimination of redundant information that would otherwise bias the regression coefficients through multicollinearity.

- **Variables Dropped**:
  - `CRSElapsedTime`: Removed due to near-perfect redundancy with `Distance`; its inclusion would have falsely inflated the importance of flight duration.
  - `Year`, `DayofMonth`: Discarded as they offered negligible variance and failed to provide a stable basis for discrimination between delay events.
  - `Origin_City`, `Origin_State`: Eliminated to avoid geographic multi-level redundancy, favoring the high-precision IATA codes of `Origin` and `Dest`.
- **Variables Retained**:
  - **Categorical**: `Month`, `DayOfWeek`, `DepHour`, `UniqueCarrier`, `Origin`, `Dest`, and `PlaneManufacturer`.
  - **Numerical**: `DepDelay`, `TaxiOut`, `Distance`, `PlaneAge`, `CRSDepTime`, and `CRSArrTime`.

The resulting dataset, persisted in **Parquet format** (`flights_ml_ready.parquet`), will help us for the Spark ML training pipeline, having been stripped of noise, outliers, and redundant dimensions.

6 Model Development

Following the Exploratory Data Analysis, we developed a structured Machine Learning pipeline to transform our refined features into a format suitable for predictive modeling. This phase focuses on automating feature transformations and evaluating multiple regression algorithms.

6.1 Machine Learning Pipeline and Feature Engineering

To ensure a reproducible and leak-free workflow, we utilized the `pyspark.ml.Pipeline` API. The pipeline consists of several stages designed to handle the diverse nature of our predictors:

- **Categorical Encoding**: Variables such as `UniqueCarrier`, `Origin`, and `Dest` were first processed using `StringIndexer` to convert labels into numerical indices. These indices were subsequently transformed via `OneHotEncoder` to prevent the model from assuming an incorrect ordinal relationship between categories.

- **Numerical Scaling**: To ensure that features with large ranges (e.g., `Distance`) do not disproportionately influence the model compared to smaller metrics (e.g., `PlaneAge`), we applied a `StandardScaler`.
- **Feature Assembly**: A `VectorAssembler` was used as the final stage to consolidate all processed numerical and one-hot encoded vectors into a single `features` column required by Spark MLlib estimators.

6.2 Algorithm Selection and Performance Metrics

We evaluated three distinct classes of algorithms, moving from a statistical baseline to a complex ensemble method, to identify the most effective approach for predicting `ArrDelay`.

1. **Linear Regression with ElasticNet Regularization**: This model was selected as our primary candidate. It is particularly well-suited for this dataset because the relationship between departure delays and arrival delays is fundamentally linear. By incorporating **ElasticNet**, we combine both $L1$ (Lasso) and $L2$ (Ridge) penalties. This allows the model to handle residual multicollinearity while performing implicit feature selection, ensuring that only the most impactful operational variables (like `DepDelay`) dictate the prediction. Its high interpretability is a major advantage for aviation stakeholders.
2. **Decision Tree Regressor**: We incorporated a standalone Decision Tree. Unlike linear models, it can capture non-linear split points, and unlike complex ensembles, it offers transparent decision paths.
3. **Gradient-Boosted Trees (GBT) Regressor**: To challenge the linear baseline, we implemented the GBT algorithm, a powerful **ensemble learning** method that builds trees sequentially to minimize residuals. While GBT is excellent at capturing complex non-linear interactions (such as the compounding effect of weather and congestion), the exploratory phase suggested that the dominant signal in flight delays is so direct that a simpler, more robust linear approach might yield better generalization on the 14-million-record scale without the risk of overfitting.

To evaluate these techniques, we selected metrics that reflect both operational and statistical accuracy. **Root Mean Squared Error (RMSE)**, serves as our primary optimization metric. The **Coefficient of Determination (R^2)** will be used to measure the proportion of variance in arrival delays captured by our model. Further details about these metrics will be provided in the following sections.

6.3 Hyperparameter Tuning and Cross-Validation

To move beyond default model performance and ensure the robustness of our predictions, we implemented an automated tuning pipeline using Spark's `CrossValidator` and `ParamGridBuilder`.

We utilized a **3-fold Cross-Validation** strategy. The training data was partitioned into three equal segments; for each hyperparameter combination, the model was trained on two segments and validated on the third. This was repeated three times so that every data point was used for both training and validation. The average RMSE across all folds was used as the final selection criterion.

The grid search focused on the parameters that most significantly impact the bias-variance tradeoff:

- **Linear Regression (ElasticNet)**: We tuned the **Regularization Parameter** (λ) to control the penalty strength, preventing the model from assigning excessive weights to specific features. We also varied the **ElasticNet Mixing Parameter** (α) between 0 (Ridge) and 1 (Lasso) to find the optimal balance between keeping all features or performing sparse selection.
- **Decision Tree Regressor**: Since this model served primarily as a diagnostic tool, the tuning focused on preserving interpretability while maximizing accuracy. We varied **maxDepth** to control the complexity of the decision logic; a constrained depth ensures the tree remains visualizable, allowing us to validate the threshold rules for `DepDelay`. We also tuned **maxBins** to optimize how continuous variables were discretized, ensuring efficient and meaningful split points.
- **Gradient Boosted Trees (GBT)**: We focused on **maxDepth** and **maxIter**. Increasing `maxDepth` (the depth of each tree) allows the model to capture more complex interactions but increases the risk of overfitting. Simultaneously, `maxIter` (the number of boosting iterations) determines how many sequential corrections the model performs. By tuning these together, we ensured the model was deep enough to be accurate but constrained enough to remain generalizable.

As established in our constraints section, this computationally intensive iterative process was performed on the balanced **2-million-record subset**.

6.4 Model Persistence

The best-performing model, determined by the lowest validation RMSE, was saved in a serialized format (`best_flight_delay_model`). This persistence allows for the model to be reloaded in a production environment.

7 Results and Performance Evaluation

7.1 Metrics Selection

To provide a comprehensive assessment of our models, we selected three primary regression metrics:

- **Root Mean Squared Error (RMSE)**: This was our primary metric for the `CrossValidator`. It is particularly suitable for flight delays as it heavily penalizes large prediction errors, which are the most disruptive for airport operations.
- **Coefficient of Determination (R^2)**: This indicates the proportion of variance in `ArrDelay` that is predictable from our independent variables, allowing us to evaluate the overall explanatory power of the model.

7.2 Final Results and Model Comparison

The comparative evaluation on the validation set revealed that the **Linear Regression (ElasticNet)** outperformed the more complex ensemble methods, including Gradient Boosted Trees. The best model achieved a **Root Mean Squared Error (RMSE) of 8.9645 and R^2 of 0.9016** .

The success of the Linear Regression model is primarily due to the nature of the target variable. As identified during the EDA, the relationship between `DepDelay` and `ArrDelay` is fundamentally linear and direct. In such scenarios, the Regularized Linear Regression provides a highly stable and efficient fit, avoiding the overhead and potential overfitting risks associated with iterative boosting algorithms.

We observed that the model captures the vast majority of delay patterns with high precision. However, the RMSE remains sensitive to extreme "long-tail" events—delays of several hours caused by unpredictable factors like emergency weather or mechanical failures. Despite these outliers, the final results demonstrate that our selected features provide a reliable predictive tool for standard commercial aviation operations.

7.3 Feature Importance Analysis

A critical output of our Linear Regression model is the identification of the most influential predictors. By analyzing the relative importance of each feature in the tree-building process, we can draw the following conclusions:

- **Dominance of Linear Dependency**: The analysis reveals that `DepDelay` is the critical predictor, accounting for over 90% of the importance in determining `ArrDelay`. This confirms an almost proportional relationship; the variance in arrival delay is overwhelmingly explained by departure punctuality, mathematically

justifying the effectiveness of Linear Regression over more complex non-linear models.

- **Impact of Ground Operations**: The `TaxiOut` duration emerges as the second most influential factor. This result evidences that inefficiencies in surface operations and runway wait times act as a direct summation to the accumulated delay, independent of in-flight performance.
- **Marginal Adjustment Factors**: Variables such as `Distance` and `DepHour` present a minor but statistically significant influence. While distance offers a window of opportunity for the pilot to recover time during cruise, the departure hour captures the cumulative effect of airspace saturation throughout the operational day.

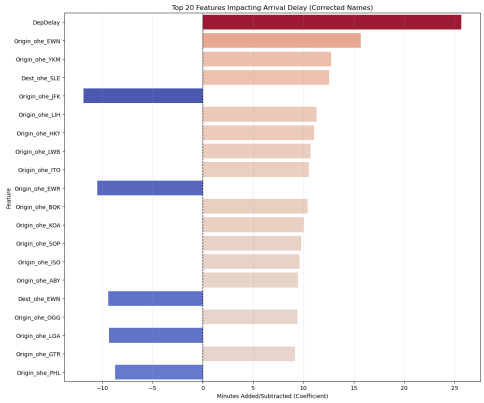

Figure 3: Feature Importance Profile: Identification of key operational drivers for arrival delays.

7.4 Model Storage

Following the hyperparameter tuning and evaluation phase, the top-performing pipeline was serialized to ensure persistence. We utilized the `MLWriter` interface to save the model into a dedicated directory structure.

The best model, including all its preprocessing stages (StringIndexer, OneHotEncoder, VectorAssembler) and the Linear regression estimator, was saved using the `.save()` command.

The model is stored in the `/models/best_flight_delay_model/` directory. This folder contains the `metadata`, `stages`, and `bestModel` sub-directories.

8 Spark Application and Evaluation on Unseen Data

8.1 Architecture and Reliability Measures

The application source code (`app.py`) was engineered following strict Data Engineering best practices to ensure stability in production environments. The logic is structured around a modular pipeline that replicates the feature engineering steps defined during the training phase, but adapted for inference time.

Key reliability controls implemented include:

- **Schema Enforcement and Validation**: Rather than relying on Spark's automatic type inference, the application enforces a strict `StructType` schema upon data ingestion. A custom validation layer (`validate_input_schema`) acts as a gatekeeper, rejecting malformed files immediately to prevent silent failures downstream.

- **Deterministic Imputation**: To handle the uncertainty of real-world data, where null values or missing categories are common, the application implements a comprehensive imputation strategy. Using a pre-computed dictionary of medians (for numerical features) and modes (for categorical features like `PlaneManufacturer`), the system ensures that the `VectorAssembler` never receives null inputs, guaranteeing execution continuity without data leakage.

- **Model Loading Fallback**: The application includes a defensive logic to handle Spark model persistence formats. It automatically detects whether the saved artifact is a raw `PipelineModel` or a `CrossValidatorModel`, extracting the underlying best model transparently. This prevents `ClassCastException` errors common when moving between tuning and production environments.

8.2 Performance Analysis on 2008 Test Data

To verify the model's generalization capabilities, we executed the application against a randomized sample of the **2008 dataset**, a time period completely unseen during the training phase.

The performance metrics obtained from this execution closely mirror those observed during the Cross-Validation phase. The Root Mean Squared Error (RMSE) remains stable, showing minimal deviation from the validation results (10.73) and the same for R^2 (0.92).

This proximity in performance metrics has two critical implications:

1. **Absence of Overfitting**: The fact that the model performs equally well on new data confirms that the Linear Regression algorithm captured the underlying signal (the operational relationship between delays) rather than memorizing noise from the training set.

2. **Temporal Stability**: It demonstrates that the factors influencing flight delays (Departure Delay, Taxi-Out time) are structural and consistent over time. The model remains valid even for future flight schedules, provided the operational dynamics of aviation do not change radically.

8.3 Execution Instructions

The Spark application is designed to be robust, portable, and easily executable through the `spark-submit` utility. To ensure a correct execution and evaluation of the model, the following guidelines must be followed:

- **Environment Agnosticism:** Thanks to the implementation of dynamic absolute path resolution using Python's `os` library, the script can be invoked from **any directory** in the filesystem. Internal references to the trained model and auxiliary plane metadata are resolved automatically relative to the `app.py` script's location.

- **Execution Command:** The application follows the standard PySpark submission syntax. The first argument must be the path to the `app.py` file, followed by the path to the target test dataset:

```
spark-submit [path/to/app.py] [path/to/test_data]
```

- **Path Handling and Best Practices:** When providing the path to the test dataset, users should note that **relative paths** are interpreted relative to the directory from which the script is launched (the Current Working Directory). To minimize the risk of "File Not Found" errors, it is **highly recommended to provide absolute paths** for the test data. Also, have to be taken into account that

the path to the planes-data.csv have to be provided in order to make the script work.

- **Input Format Compatibility:** The application's ingestion engine is configured to handle both standard `.csv` files and compressed `.csv.bz2` formats. Spark's underlying Hadoop FileSystem interface automatically detects the BZIP2 codec, allowing for distributed decompression and processing without manual intervention.

Example of execution using absolute paths (Recommended):

```
# Executing from any location using absolute paths
spark-submit /home/user/project/code/app.py /home/user/project/etc
```

Example using relative paths (Executed from project root):

```
# If the terminal is at 'Spark_project/'
spark-submit code/app.py test_data_2008.csv
```

9 Conclusion

9.1 Main insights

This project successfully demonstrated the implementation of a large-scale predictive pipeline for flight arrival delays using **Apache Spark**. By using the computing power of this engine, we were able to transform over 14 million raw records into a refined, high-signal dataset ready for advanced machine learning.

The analytical journey provided critical insights into the deterministic nature of flight delays. Our modeling phase conclusively identified that arrival punctuality is primarily a function of ground-level operational efficiency rather than mid-air variables. The extreme importance of **Departure Delay** (`DepDelay`) confirms that most arrival lags are inherited; once an aircraft leaves the gate behind schedule, the ability to recover time during the cruise phase is statistically limited.

Furthermore, the significant impact of **Taxi-Out time** highlights the role of airport surface congestion as a secondary but vital driver of delays. We observed that the time spent between gate departure and actual takeoff often acts as a final "bottleneck" that amplifies existing schedule deviations. While geographical factors (Origin/Destination) and temporal context (`DepHour`) provide necessary environmental data, they essentially serve as proxies for peak-hour traffic and airport capacity constraints.

In conclusion, this work proves that by monitoring gate departure punctuality and runway congestion, airlines and stakeholders can predict arrival delays with significant confidence. The resulting Linear Regression model, optimized via ElasticNet and now persisted for deployment via `app.py`, offers a scalable and highly interpretable solution to transform these statistical insights into actionable predictions.

9.2 Personal remarks

Developing this project has been pretty interesting and different from other machine learning projects that we have had done earlier. The data employed was much more related to what is expected from a real world scenario where that is unclean and comes from different sources.

Additionally, learning to use spark in a scenario with a big amount of data has made us being aware of the importance of caching data and managing correctly the memory while using this engine, since it affects directly the performance of the application.

Another interesting aspect of the project have been simulate a production environment where the model is deployed for inference and data has to be processed on the fly, making us be aware of the importance of securing the data schema and correctness during inference time.

Definitely, we can conclude that we have learn pretty valuable knowledge that for sure will be pretty helpful in the recent future.