Doordash 2014 – Delivery duration prediction

[Overview 1](#_Toc188439416)

[Data Preprocessing and Feature Selection 1](#_Toc188439417)

[Scaling Techniques 4](#_Toc188439418)

[Regression Models 5](#_Toc188439419)

[Evaluation Process 7](#_Toc188439420)

[Conclusion 9](#_Toc188439421)

# Overview

This script implements a comprehensive machine learning pipeline for regression, comparing various models and preprocessing techniques to predict a target variable which the total duration of delivery. The analysis includes feature selection, dimensionality reduction, model training, and performance evaluation and aims to predict total duration of delivery and identify the key factors influencing it.

# Data Preprocessing and Feature Selection

First, we used a Random Forest to identify which input variables have the strongest predictive power for the total delivery duration (i.e., the target variable). The Random Forest algorithm assigns importance scores to each feature based on how much they contribute to reducing prediction error, based on the Gini Importance (i.e., Mean Decrease in Impurity). It measures how much each feature contributes to decreasing impurity across all trees in the random forest.

1. At each split in a decision tree, the algorithm chooses the feature that best separates the data into more homogeneous groups
2. The reduction in impurity (measured by the Gini index) is calculated for that split
3. For each feature, its importance is the sum of all impurity reductions where that feature was used, weighted by the number of samples it helped to classify
4. This is averaged across all trees in the forest

Based on the Gini importance one can determine the most important features (i.e., higher values) meaning that the model relies most heavily on these features to make predictions. The Gini importance is computationally efficient as it is calculated during the training process, no additional computation is needed after the model training phase, and it scales well with large datasets. Moreover, it is not too much affected by random variations in the data, provides consistent results across multiple runs and is suitable for initial feature selection. Finally, it provides a single, clear number for each feature, it makes easy to rank and compare features and is simple to visualize and explain to stakeholders.

*Notes: Alternative measures exist such as*

* *The permutation importance which works by randomly shuffling the values of each feature and measuring how much the model’s performance decreases. It is more intuitive to interpret, less biased towards high cardinality features and can be calculated for any model, not just tree-based ones. However, it is more computationally more expensive, can be affected by feature correlation and thus may give misleading results with highly correlated features.*
* *SHAP (Shapley Additive Explanations) which comes from game theory and measures each feature’s contribution to each individual prediction. It provides local and global importance, has a theoretically sound foundation and can show how features interact. On the other hand, it is very computationally intensive, more complex to interpret than the Gini importance or the permutation importance and thus may be an overkill for simple applications.*
* *The drop-column importance which involves removing each feature one at a time and measuring the impact on model performance. It is very intuitive, captures real impact on model performance and works with any model. However, it is computationally expensive for large datasets, it doesn’t capture feature interactions well and it may need multiple runs due to randomness.*

*These alternatives are good to use when one has highly correlated features, needs feature importance for non-tree models, needs detailed insights into feature interactions and/or is working with categorical variables with many levels.*

While the Gini importance help to select features, a PCA allows to reduce dimensionality without explicitly removing features.

1. Finding Directions of Maximum Variance: PCA looks for the direction where your data varies the most. This is like finding the angle that shows the biggest difference between data points. This becomes your first principal component.
2. Finding Additional Components: It then looks for the next most important direction, perpendicular to the first one. This process continues until you have as many components as original features.
3. The Cumulative Explained Variance Plot: This plot shows how much information (variance) you retain as you add more components. It typically looks like a curve that starts steep and then levels off. The point where it levels off suggests how many components you need to keep most of the important information.

Using both PCA and Gini importance provides complementary insights:

* Gini importance shows which individual features matter most
* PCA shows how features work together and might be redundant

If both methods suggest similar features are important (either directly or as major contributors to principal components), it strengthens your confidence in the feature selection.

This dual approach is particularly valuable when:

* You have many features and want to understand their relationships
* You need both feature selection and dimensionality reduction
* You want to validate your feature selection through different methods
* You need to balance model interpretability with performance

# Scaling Techniques

The script compares three different scaling approaches.

* Standard Scaling: This method transforms your data so that it has an average of zero and a standard deviation of one. It takes each value, subtracts the mean, and divides by the standard deviation. This is particularly useful when your data follows a bell curve distribution. For example, if you have a feature measuring age (ranging from 0 to 100) and another measuring income (ranging from 0 to millions), standard scaling will bring both these features to comparable ranges.
* Min-Max Scaling: This approach transforms your data to fit within a specific range, usually between 0 and 1. It preserves the shape of the original distribution while compressing all values into this new range. This is particularly useful when you're working with image data or when you need all positive values.
* No Scaling: This means using the original values without any transformation. Some models can work perfectly fine with unscaled data because they don't make assumptions about the relative sizes of features.

Some models perform better after scaling such as linear and logistic regressions, neural networks, support vector machines, K-nearest neighbours and principal component analysis. On the other hand, tree-based models (decision trees, random forests, gradient boosting machines) can handle raw data.

Other scaling methods exist and can be considered depending on the case at hand:

* Robust Scaling: Uses statistics that are robust to outliers (median and quartiles instead of mean and standard deviation). Useful when your data has many outliers.
* Maximum Absolute Scaling: Similar to min-max scaling but scales the data relative to the maximum absolute value. Useful when dealing with sparse data.
* Quantile Transformation: Transforms features to follow a uniform or normal distribution. Useful for data with outliers or when you want to force a specific distribution.
* Power Transformation (like Box-Cox or Yeo-Johnson): Helps stabilize variance and make data more normal distribution-like. Useful when dealing with skewed data.
* Log Transformation: Not technically a scaler but often used to handle skewed data or data with multiplicative relationships. Useful for data that spans several orders of magnitude.

The choice of a scaling method often depends on the distribution of the data, model requirements, feature characteristics, and the specific output needs of the task. When data roughly follows a normal distribution, standard scaling is typically sufficient. However, if the data is skewed, a log or power transformation may be more effective at normalising its distribution. In cases where outliers are present, robust scaling is a good option, as it focuses on the median and the interquartile range, thereby limiting the influence of extreme values.

Certain models also have specific scaling needs. Neural networks often benefit from standard or min-max scaling to speed up training and help the network converge more reliably. In contrast, tree-based methods such as Random Forest or gradient boosting typically do not require any scaling, as the split criteria rely on feature ordering rather than absolute values. Distance-based models like K-Nearest Neighbours do require feature comparability, so any consistent scaling technique can be applied.

The nature of the features themselves is equally important. Features prone to outliers may be better served by robust scaling, while those that are sparse might fit well with min-max or maximum absolute scaling. If particular features exhibit a high degree of skew, applying a log or power transformation can produce more balanced feature distributions.

Lastly, the required output range may influence which scaling method is selected. If a bounded output is essential, min-max scaling can ensure all values fall within a predefined interval. When preserving zero values in the data is important, min-max scaling again proves useful, as it maintains the position of zero within the new scaled range. However, for data containing outliers, robust scaling remains a strong option due to its resilience to extreme values.

# Regression Models

The script compares six different regression models, each with its own characteristics and, representing a spectrum from simple to complex.

Ridge regression (i.e., linear model) serves as a baseline model and is particularly valuable as is help to understand if linear relationships exist in the data, its coefficients directly show the impact of each feature, it handles correlated features better than standard linear regression trough L2 regularization by preventing overfitting (i.e., when the model is learning from the noise in the training data rather than from the underlying pattern leading to poor performance on new, unseen data) and might be enough, when it performs well, compared to more complex models.

Tree-based models (decision trees ➜ random forests ➜ gradient boosting) represent an evolution in complexity and capability. Decision trees are like a series of it-then rules, making them excellent for understanding how decisions are made, good at handling non-linear relationship, useful for identifying feature interactions but also limited by their tendency to overfit when grown too deep. On the other hand, random forests build on trees by creating multiple trees with different random subsets of data, averaging their predictions to reduce overfitting, maintaining the ability to handle non-linear relationships and providing more stable predictions than single trees. Finally, Gradient Boosting (XGBoost and LightGBM) take this further by building trees sequentially, each focusing on correcting previous trees’ mistakes, usually achieving better performance than random forests, offering different implementation trade-offs (speed vs. memory usage) but require more careful tuning. XGBoost is good at handling missing values while LightGBM is really memory efficient.

Neural networks (Multi-Layer Perceptron – MLP) represent a different approach entirely. They can learn very complex patterns but require more data to perform well, need more careful tuning of hyperparameters and are less interpretable than other models.

When choosing an appropriate model, several practical factors should be considered, including data size, the nature of the features, the available computational resources, and the level of interpretability required.

Data size plays a significant role in model selection. With smaller datasets (fewer than 1,000 samples), it is often best to begin with simpler models such as Ridge regression or Random Forest, as these are less prone to overfitting and are relatively straightforward to tune. For medium-sized datasets (1,000 to 10,000 samples), any of the commonly used models can be viable, offering a balance of complexity and performance. When working with very large datasets (more than 10,000 samples), models like LightGBM become appealing for their speed and efficiency, while neural networks can be considered if capturing complex patterns is a priority.

Feature characteristics also matter. When a dataset contains many categorical variables, tree-based models such as Random Forest or gradient boosting methods typically handle them effectively. If features are highly correlated, Ridge regression or gradient boosting can help control this multicollinearity. In cases where data is missing, certain models like XGBoost can naturally incorporate missing values into their learning process without extensive imputation.

Computational resources should be evaluated before finalizing a choice. For environments with limited CPU or memory, models like LightGBM or Random Forest are preferred due to their relatively efficient memory usage and faster training times. If resources are ample, training multiple models and creating an ensemble can often yield the best predictive performance.

Lastly, interpretability requirements can determine which model is most suitable. If transparency is paramount, methods like Decision Trees or Ridge regression offer relatively straightforward insights into how predictions are made. For moderate interpretability, Random Forest provides a balance of decent performance and partial explainability. When interpretability is less critical, neural networks or advanced gradient boosting methods can be used to capture complex patterns with minimal constraints on how these patterns are internally represented.

# Evaluation Process

To evaluate the fitted model, a first step is to split the dataset into a training subset (80%) and a test subset (20%) used to assess how the model has learnt. If the dataset is large enough, one might use more data for the training set (90-10) although the 80-20 ratio is usually a good balance.

The Root Mean Squared Error (RMSE) is a key statistical metric used to evaluate the accuracy of predictive models by quantifying the average difference between predicted and actual values. It penalised more heavily large errors than small ones meaning that it can be heavily influenced by extreme errors. RMSE represents the standard deviation of the residuals (prediction errors) in a statistical model. It provides a single number that indicates how well a model's predictions match the observed data. A lower RMSE indicates more precise predictions, with a perfect model theoretically achieving an RMSE of 0. Yet, a good score depends on the target variable and its variation range. Moreover, because RMSE uses the same units as the target variable, it is easy to interpret but should be only carefully compared across models predicting different target variables.

We also try different ways of preparing our data through scaling. Some models (like neural networks) really need this standardization, while others (like random forests) can perform well without it. We try different scaling methods to see what works best.

In our script, we also accounted for the inverse transformation in the estimation of RMSE. Indeed, prior to running the tested models, we transformed our data to allow better fit of the machine learning algorithms. However, if the transformations are better for modes, it complicates the interpretation of the outputs. After the model makes its predictions in the scaled space, we need to inverse the scaling to get back to the original units. When we calculate our error metrics (like RMSE) after this inverse transformation, we get numbers that are meaningful to business stakeholders.

This whole process is essential for three main reasons:

1. It makes our results interpretable to everyone, not just data scientists
2. It allows us to judge whether the model's performance is good enough for our business needs
3. It helps us compare different models fairly in terms that matter to the business

It is a crucial step in bridging the gap between the mathematical world where our models operate and the real world where their predictions will be used. Without this transformation, we'd be stuck trying to interpret abstract numbers that don't connect directly to our business metrics and goals.

# Conclusion

When we put all this together, we're looking for a few key things:

* Is the model's performance on the test data similar to its performance on the training data? If it did much better on the practice test, it might have just memorized the answers instead of learning the underlying patterns.
* Can we get good predictions with fewer features? Simpler models are usually easier to maintain and understand.
* Is our error rate (RMSE) acceptable for our specific problem? This depends entirely on what we're predicting and what accuracy level we need.
* Is the model consistent, or does its performance vary a lot when we make small changes? We want reliable, stable predictions.

The script's structured approach to model comparison makes it valuable for both model selection and understanding the relative importance of different preprocessing steps in the machine learning pipeline.