CYO Project Report - Delivery Prediction System

Asham Vohra

6/21/2021

Overview

Prediction systems play a crucial role in the modern world. With advent of technology and availability of data, these systems are being leveraged by retail chains, banks, colleges among other businesses. Depending on the use case, these systems are used to estimate delivery time, sales figures, carry out inventory management, predict occurrence of a disease etc.

Inspired by what all problems these systems can solve, we have attempted to build a delivery prediction system based on publicaly available dataset of orders made at Olist Store, a Brazilian e-commerce.

The dataset used here is hosted at Kaggle website. The dataset is provided by Olist itself, which is the largest department store in Brazilian marketplaces. Olist connects sellers to the customers. It is the sellers, who are responsible to fulfil any orders placed on the platform using Olist's logistic partners.

The dataset consists of **100,000+** orders and for each order the dataset contains related details like product bought and from which category, customer who bought and his/her location, seller involved and many more attributes.

The delivery prediction system developed as part of the project, leverages the past orders, product category, delivery distance, customer location, actual delivery time and related attributes to predict delivery estimate for the new orders.

The goal of the project was to build a delivery prediction system which can estimate number of hours required for delivery keeping RMSE(Root mean square error) minimal.

In order to achieve this goal, we started by cleaning data, identifying right predictors, deriving insights from data already available to create new predictors. Once we had the dataset with us, we incrementally took a predictor, analysed its relationship with the number of hours of delivery, incorporated the predictor in our model after encoding, scaling it as required and tested our updated model against the test data. To ensure that the model performance was not due to random split of dataset, the model was trained and tuned using k fold cross validation and then only tested against the test dataset. Only if the predictor helped improve our metric i.e. RMSE, the predictor under analysis and evaluation was added to the model and the steps were repeated with new predictor. Otherwise the predictor was ignored

This report walks through the approach, analysis and evaluation carried out to achieve our delivery prediction system

Analysis

Data Wrangling

An important step in using any public data set is converting it to usable form. In our case, the data set was available in multiple csv files, each file dedicated to a specific kind of information. Example:

- There was a file for orders which had information about product bought, purchase time, delivery timestamp and other attributes.
- Then there was a file with product details which had information about product category, product weight and other attributes.

Each of these files due to the varied dataset required different data wrangling steps to achieve a usable form which could then be unified for further analysis.

Data wrangling for products data set

The data set here had product details like category name, dimensions and other attributes. Here the category name was in Portuguese. Some important attributes like product dimensions and weight were not populated for few of the products. To analyse this data properly, we carried out below steps:

- Read the concerned csv file and removed irrelevant columns for the analysis.
- Removed products where dimensions and related attributes were not available.
- Assigned default category name of UNCATEGORIZED for products where product_category_name was not available.
- Replaced product category names with English names where-ever mapping was available. The product categories for which equivalent English name mapping was not available were left untouched.

The wrangled data looked like below:

Table 1: Product data set

	Example Value1	Example Value2
product_id	1e9e8ef04dbcff4541ed26657ea517e5	3aa071139cb16b67ca9e5dea641aaa2f
product_category_name	perfumery	art
product_weight_g	225	1000
product_length_cm	16	30
product_height_cm	10	18
product_width_cm	14	20

Data wrangling for orders data set

The data set here had orders details like order id, associated customer, order status, purchase time, when was the order handed over to carrier, when was it delivered to customer and estimated delivery date and few other attributes.

This was a crucial dataset not only for us to build our delivery estimation system but to even identify the current performance metric i.e. RMSE for the approach Olist was using.

In order to leverage this dataset properly, we carried out below steps:

- Read the concerned csv file and removed irrelevant columns for the analysis.
- Parsed columns with date as characters and converted them to desired date type.
- Filtered out any orders where delivery was not made or where delivery was made but required delivery time columns were not available.

The wrangled data looks like below:

Table 2: Orders data set

	Example Value1	Example Value2
order_id	e481f51cbdc54678b7cc49136f2d6af7	53cdb2fc8bc7dce0b6741e2150273451
customer_id	9ef432eb6251297304e76186b10a928db0830fb4747a6c6d20dea0b8c802d7ef	
order_status	delivered	delivered
order_purchase_timestamp	2017-10-02 10:56:33	2018-07-24 20:41:37
payment_approved_at	2017-10-02 11:07:15	2018-07-26 03:24:27
handed_over_to_carrier_date	2017-10-04 19:55:00	2018-07-26 14:31:00
order_delivered_customer_date	2017-10-10 21:25:13	2018-08-07 15:27:45
order_estimated_delivery_date	2017-10-18	2018-08-13

Data wrangling for orders items data set

The data set here had order item details like which product(s) were bought for a given order, who the seller was, price and freight value and shipping limit date. While most of the attributes were used to connect with other data sets, it was the shipping limit date which could be used along with product handed over to carrier date from orders dataset to identify any delays in the shipping process.

In order to leverage this dataset properly, we carried out below steps:

- Read the concerned csv file and removed irrelevant columns for the analysis.
- Parsed columns with date as characters and converted them to desired date type.

The wrangled data looks like below:

Table 3: Order Items' data set

	Example Value1	Example Value2
order_id	00010242fe8c5a6d1ba2dd792cb16214	00018f77f2f0320c557190d7a144bdd3
product_id	4244733e06e7ecb4970a6e2683c13e61	e5f2d52b802189ee658865ca93d83a8f
seller_id	48436dade18ac8b2bce089ec2a041202	dd7ddc04e1b6c2c614352b383efe2d36
shipping_limit_date	2017-09-19 09:45:35	2017-05-03 11:05:13
price	58.9	239.9
freight_value	13.29	19.93

Data wrangling for geolocation data set

The data set here had geolocation data. It had zip code prefix, lat, lang, city and state details. However, there was no unique identifier in the dataset.

It is important to note that Brazil uses Postal Code Address(CEP) which is 8-digit number created mainly from mail distribution objective. It is split into two parts of 5 and 3 digits resp. separated by a hyphen('-'). The first part helps identify the rough delivery area, while it is the second part which provides precision and helps one reach the desired street, neighbourhood.

So in the geolocation data set, zip code prefix referred to the first part of the CEP and it varied from 1 to 5 digits in the dataset available to us. Because the zip code prefix does not represent the precise value, it was not unique and had many duplicate values with varying value of lat, lng for same zip code prefix.

In order to be able to use this data with other data sets available like customer location and seller location datasets, which are discussed later, we carried out below steps:

- Read the concerned csv file and removed irrelevant columns for the analysis.
- Calculated the mean of lat, lng for given zip code prefix.

The wrangled data looks like below:

Table 4: Geolocation data set

	Example Value1	Example Value2
geolocation_zip_code_prefix	1001	1002
geolocation_lat	-23.5501897765518	-23.5481457317636
geolocation_lng	-46.6340235559042	-46.634979210745

Data wrangling for customers and sellers data set

There were two data sets one for customers and one for sellers. Apart from identifier ids like customer_unique_id, seller_id, both the datasets had similar columns i.e. zip code prefix, for which details were in geolocation data set and city and state.

To be able to leverage either of these datasets for analysis later, we carried out below steps:

- Read the concerned csv file and removed irrelevant columns for the analysis.
- Combined sellers dataset data individually with geolocation dataset in order to capture seller lat,lng. Similar operation was carried out for customers data set.
- Remove sellers or customers for which geolocation details are not available as delivery or even its estimation is not possible without the critical fields.

The wrangled sellers dataset looks like below:

Table 5: Sellers data set

	Example Value1	Example Value2
seller_id	3442f8959a84dea7ee197c632cb2df15	d1b65fc7debc3361ea86b5f14c68d2e2
seller_zip_code_prefix	13023	13844
seller_city	campinas	mogi guacu
seller_state	SP	SP
geolocation_lat_seller	-22.89385	-22.38344
geolocation_lng_seller	-47.06134	-46.94793

The wrangled customers dataset looks like below:

Table 6: Customers data set

	Example Value1	Example Value2
customer_id	06b8999e2fba1a1fbc88172c00ba8bc7	18955e83d337fd6b2def6b18a428ac77
customer_unique_id	861eff4711a542e4b93843c6dd7febb0	290c77bc529b7ac935b93aa66c333dc3
customer_zip_code_prefix	14409	9790
customer_city	franca	sao bernardo do campo
customer_state	SP	SP
geolocation_lat_customer	-20.49849	-23.72799
geolocation_lng_customer	-47.39693	-46.54285

Having wrangled the datasets individually we combined them into a single dataset. While the dataset had many attributes, based on their frequency and our intuition shared earlier we zeroed on the below fields for further consideration. In addition, it is important to note that for model development we created outcome column number_of_hours based on existing fields i.e. order_delivered_customer_date and order_purchase_timestamp as this is what we would want to predict whenever we get a new order i.e. given this order, how many number of hours are required to deliver the product to the customer.

The wrangled dataset after incorporating various datasets available to us looked like below:

Table 7: Final wrangled data set

	Example Value1	Example Value2
order_purchase_timestamp	1506941793	1532464897
payment_approved_at	2017-10-02 11:07:15	2018-07-26 03:24:27
handed_over_to_carrier_date	2017-10-04 19:55:00	2018-07-26 14:31:00
seller_id	3504c0cb71d7fa48d967e0e4c94d59d9	289cdb325fb7e7f891c38608bf9e0962
shipping_limit_date	2017-10-06 11:07:15	2018-07-30 03:24:27
price	29.99	118.70
freight_value	8.72	22.76
geolocation_lat_seller	-23.68073	-19.80768
geolocation_lng_seller	-46.44424	-43.98043
customer_zip_code_prefix	03149	47813
customer_state	SP	BA
geolocation_lat_customer	-23.57698	-12.17792
geolocation_lng_customer	-46.58716	-44.66071
product_category_name	housewares	perfumery
product_weight_g	500	400
product_length_cm	19	19
product_height_cm	8	13
product_width_cm	13	19
number_of_hours	202.4778	330.7689

Performance Metric

Before we dived into building our delivery prediction system, we identified our metric of choice i.e. RMSE for evaluating current delivery estimates being made by Olist and for comparing our delivery prediction system against that, in addition to improving our system as we built it.

- RMSE is the root mean square error and that's what was used for this project to evaluate models and to optimize them for fitment.
- As we know, $RMSE = sqrt(1/N * sum((\hat{y} - y)^2))$
  - Here, $\hat{y}$ is the prediction. In context of this project, predicted number of hours for delivering the order item,
  - and y is the actual outcome. In context of this project, actual number of hours taken for delivering the order item.

Current RMSE

Before we dived into building our delivery prediction system, we looked into current RMSE. To calculate this, we leveraged order_estimated_delivery_date i.e. estimated delivery date and order_delivered_customer_date i.e. actual delivery timestamp, which was available in orders data set.

While order_estimated_delivery_date did not have time details, order_delivered_customer_date did. Since order_delivered_customer_date could be rounded off to same day or next day, we computed range of RMSE which can be calculated using either of the approach.

This RMSE i.e. our performance metric was later used as benchmark for modelling. The delivery prediction system being built attempted to improve on this metric.

Below were the current RMSE calculations based on how we interpret the actual delivery timestamp.

Table 8: Performance Metric(RMSE) Summary

type	value
Current RMSE Max	359.9466
Current RMSE Min	377.9515

Partitioning dataset for training and validation

Before using the wrangled data for analysis and data modelling, we partitioned the data into training and validation data sets. The idea was to keep validation data set aside and use only training data for any analysis, model development and tuning.

A split of 80/20 was made here between training and validation set to ensure we have enough data points to train and to validate the results. Secondly, the split was done such that distribution of outcome i.e. number_of_hours was similar in both training and test data.

The data sets were named as below for future reference:

- training data set as **training_dataset**
- validation data set as **validation_dataset**

Below were the details about the elements in each of the partitioned datasets.

data_set	name	count
Training data set	training_dataset	87695
Validation data set	validation_dataset	21004

Feature Analysis and Selection

Since we needed to build delivery prediction system which could estimate number of hours required for delivery for new orders, while maintaining minimum RMSE, we used our intuition and understanding of the domain to guide us in deriving insights from available predictors. The same are shared below:

- ***Distance between seller and customer*** could likely influence delivery estimate.
- ***Delay in payment approval*** could influence delivery time as it would delay shipment from the seller. This could be related to customer zip code as often some customer locations require fraud analysis before payment is approved.
- ***Delay in handover of product from seller to carrier*** would influence delivery time as any time spent there is wasted. This could be seller related as some sellers maybe known to take more time in processing order compared to other sellers.
- Some ***customer locations*** maybe difficult to reach or need special delivery mechanism due to their remote nature. Such locations may need more delivery time compared to other locations irrespective of the distance involved. We need not analyse for each customer location but a rough area could be used based on combination of zip codes.

- ***Weight of the product*** could influence delivery time as heavy items say bed, furniture may be transported slowly while lighter items maybe shipped more quickly.
- ***Price of the product*** could influence delivery time. Expensive items may require considerable more attention and could influence delivery like priority shipping or longer delivery time due to caution.
- ***Volume of the product*** could influence delivery time as smaller products can be shipped together while others may need separate truck and therefore more time for dedicated truck to be available.
- ***Product category*** could be used for analysis as different product categories would have different packing, assembly, processing requirements and shelf life would could lead to impact in delivery time.
- ***Freight value*** could indicate complexity of shipping like fragility of the product, distance, toll booth charges on the way among others. This could be considered for analysis.
- ***Customer state*** could influence the delivery estimate due to state border checks, paper work before goods delivery among others.
- Different ***sellers*** may take different amount of time or sell specific type of products which may require different shipping time. So products from a seller selling apparels may be delivered quickly while products from one selling customised products or say furniture may take relatively more time. This could be useful information for predicting delivery time.

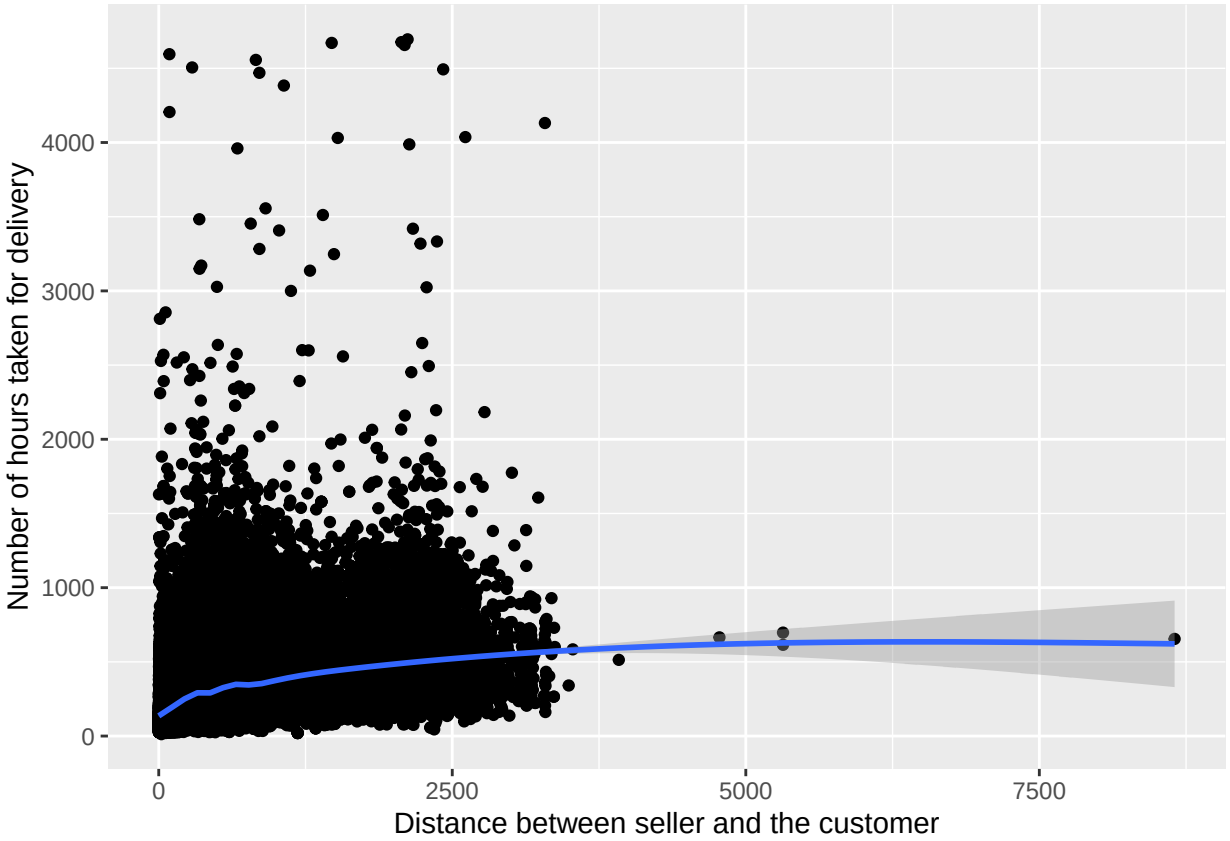
For analysis, we used only the training_dataset or subset of it, ensuring validation_dataset was left untouched.

Derived features

Distance between seller and customer feature

As part of the dataset we had customer and seller location details available in the form of latitude(lat), longitude(lng). We used this crucial knowledge of their location to determine shortest distance between the two on an ellipsoid.

We decided to look more into this intuition. For this we plotted numbers of hours taken for delivery against the distance between seller and the customer. In order to see the trend in this over-plotted graph, gam method based smoothing curve was also plotted on the graph.



In above plot, we noted that number of hours taken for delivery showed stronger correlation for smaller distances. However, once delivery distance increased beyond a threshold, the correlation weakened between the fields. Though, we could see broader confidence interval for larger distances.

Table 10: Correlation analysis between distance between seller & carrier and number of hours taken for delivery

delay_limit	correlation
>=0 & <1000	0.3347641
>=1000 & <2000	0.1214192
>=2000 & <3000	0.03644873
>=3000	0.0446932

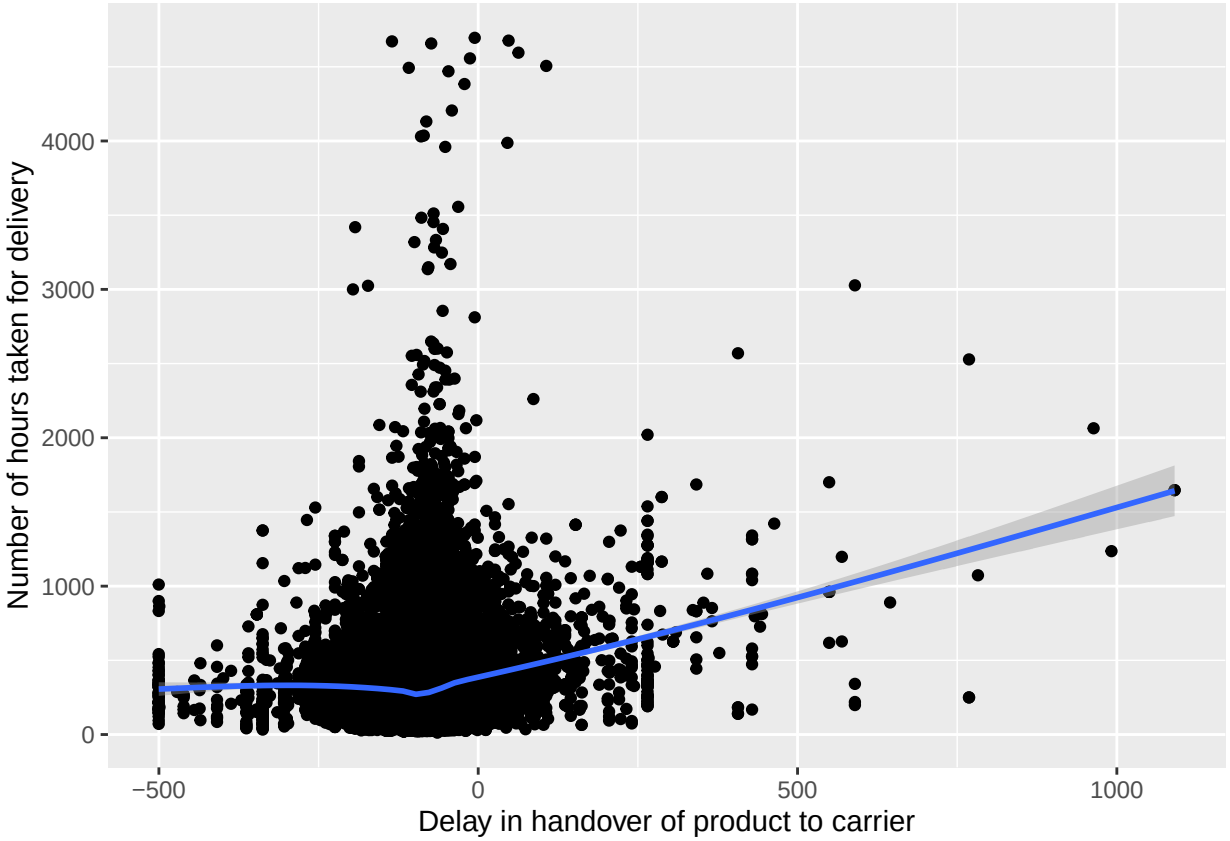
This behavior could be useful in estimating delivery timelines. We decided to consider this as a feature for further analysis for inclusion in the model.

Delay in handover of product from seller to carrier feature

As part of the dataset, there were two important attributes i.e. handed_over_to_carrier_date and shipping_limit_date. Here, handed_over_to_carrier_date is the actual timestamp when product ordered was handed over the carrier for delivery. While, shipping_limit_date is the date by which seller should handover the order to the carrier.

Together these fields were considered useful as they could be used to understand if there was a delay or not in handing over product ordered by seller to the carrier. Also, this delay or early handover, could influence delivery timeline even if everything else was to take the same time.

We decided to look more into this intuition. For this, the feature i.e mean_delay_in_handover was computed for each seller. The computed data was then joined with training_dataset and the below graph was used to visualize the relation between delay in handover to carrier and number of hours taken for delivery. In order to see the trend in this over-plotted graph, gam method based smoothing curve was also plotted on the graph.



In above plot, we noted that number of hours taken for delivery shows weaker correlation with delay in handover by seller to the carrier as long as the ordered product is handed over to carried before the shipping_limit_date. However, post the date, we could see relatively stronger correlation between the fields.

Table 11: Correlation analysis between delay in handover to carrier and number of hours taken for delivery

delay_limit	correlation
<= 0	0.0423349
> 0	0.3469383

This behavior could be useful in estimating delivery timelines. We decided to consider this as a feature for further analysis for inclusion in the model. However, as for new orders this information would not be available at time of placing the order, we were required to compute this feature using training data and apply the captured learnings later on test data.

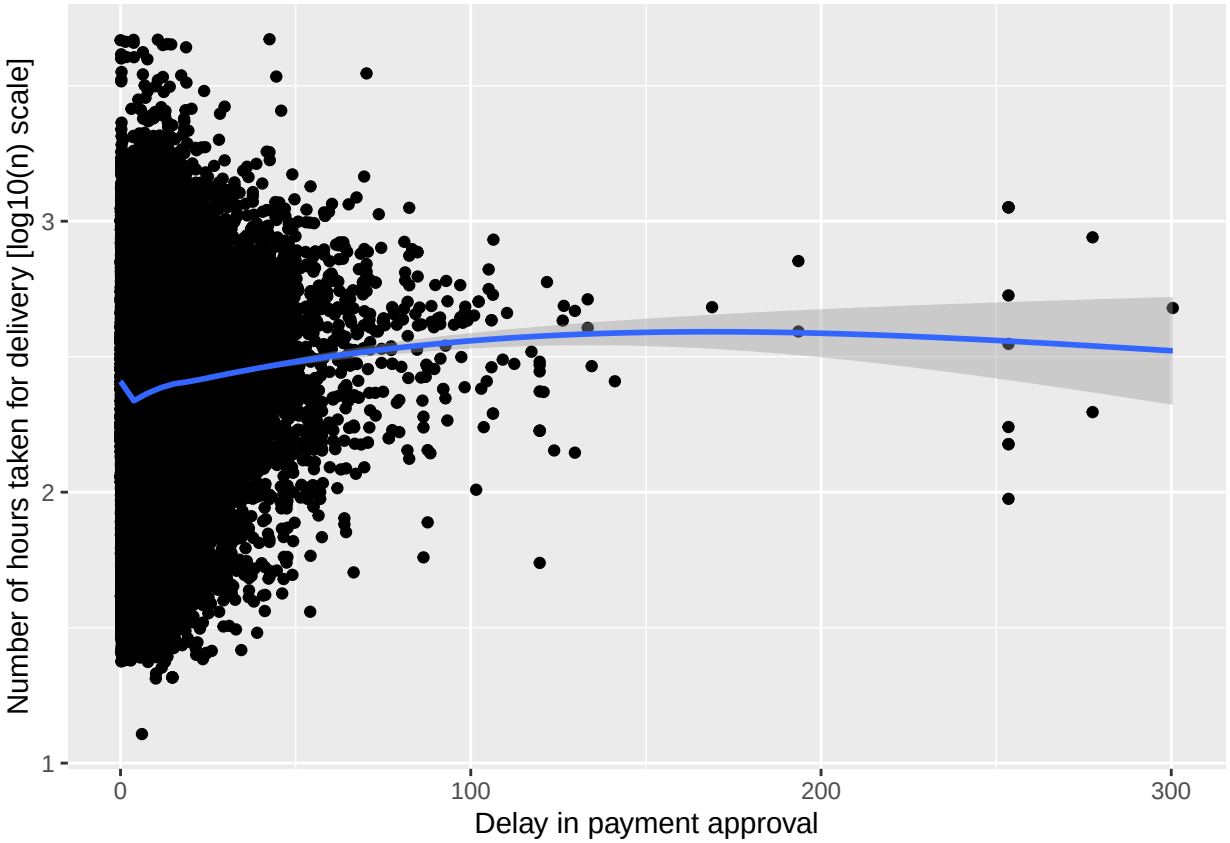
To be able to do this, as shared earlier, the feature i.e. mean_delay_in_handover was computed for each seller based on training dataset and then applied to test dataset.

Delay in payment approval feature

In addition to above, as part of the dataset, there were two other important attributes i.e. order_purchase_timestamp and payment_approved_at. Here, payment_approved_at is the actual timestamp when payment is approved for a given order. While, order_purchase_timestamp is the timestamp at which product was ordered.

Together these fields were considered useful as they could be used to understand if there was a delay or not in approving the payment. Also, this delay or early handover, could influence delivery timeline even if everything else was to take the same time.

We decided to look more into this intuition. For this, the feature i.e mean_delay_in_payment_approval was computed for each customer_zip_code_prefix. The computed data was then joined with training_dataset and the below graph was used to visualize the relation between delay in payment approval and number of hours taken for delivery. In order to see the trend in this over-plotted graph, gam method based smoothing curve was also plotted on the graph.



In above plot, we noted that number of hours taken for delivery follow a non linear pattern.

Table 12: Correlation analysis between delay in payment approval and number of hours taken for delivery

delay_in_approval	correlation
>=0 & <25	0.03740626
>=25 & <50	0.07246898
>=50 & <75	0.1573909
>=75 & <100	0.04214726
>=100 & <125	-0.3564631
>=125 & <150	-0.3695339

delay_in_approval	correlation
>=150	-0.0036209

This behavior could be useful in estimating delivery timelines. We decided to consider this as a feature for further analysis for inclusion in the model. However, as for new orders this information would not be available at time of placing the order, we were required to compute this feature using training data and apply the captured learnings later on test data.

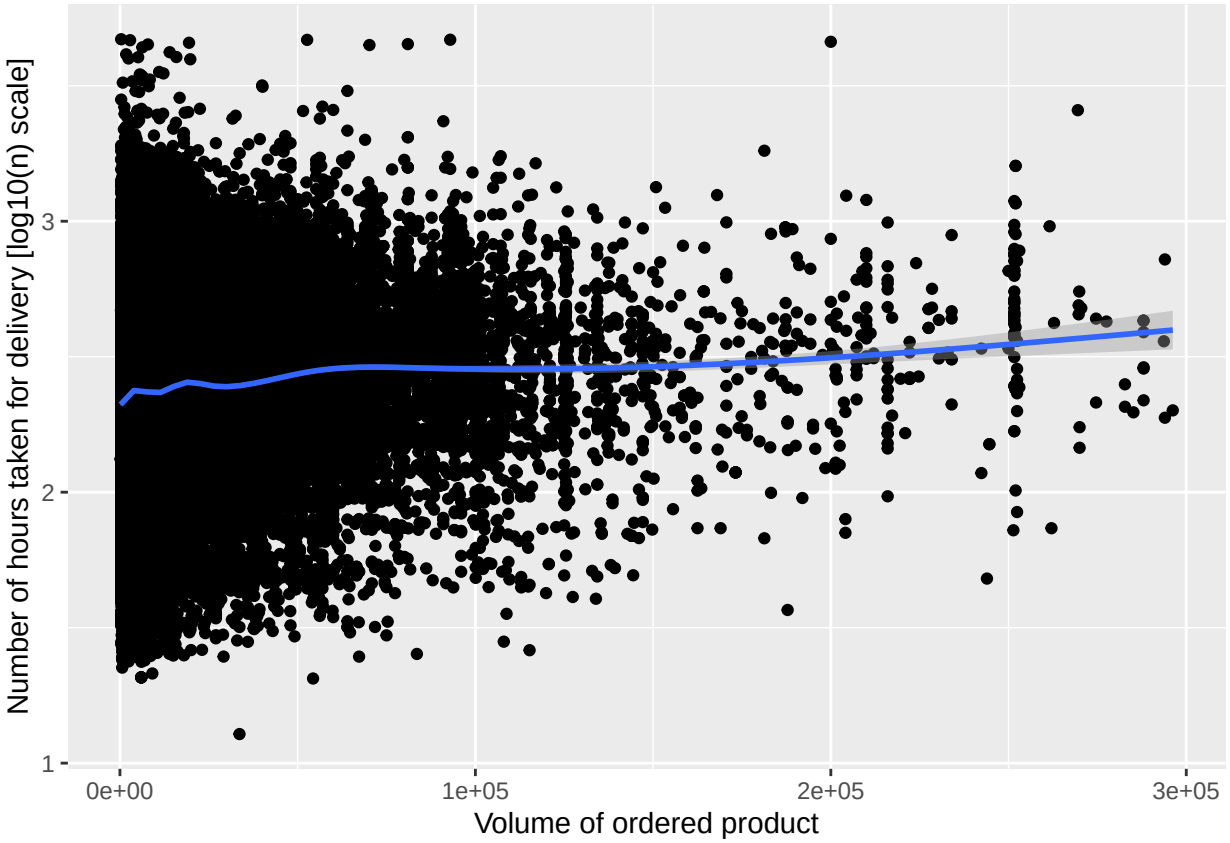
To be able to do this, as shared earlier, the feature i.e. mean_delay_in_payment_approval was computed for each customer_zip_code_prefix based on training dataset and then applied to test dataset.

Volume feature

As part of the dataset, there were three attributes about product dimensions i.e. product_length_cm, product_height_cm and product_width_cm. Together these fields were considered useful as they could be used to derive volume of the product.

We looked into the intuition that product volume could influence the delivery timelines as larger products need special delivery mechanism and even dedicated trucks at times for shipment, which may not be readily available.

To understand its influence on the delivery timelines, we plotted numbers of hours taken for delivery against the volume of the product ordered in cubic centimeter(cm^3). In order to see the trend in this over-plotted graph, gam method based smoothing curve was also plotted on the graph.



In above plot, we noted that number of hours taken for delivery vary as the product volume changes. There appeared to be stronger influence on delivery timelines for bulkier products beyond a limit.

Table 13: Correlation analysis between volume and number of hours taken for delivery

delay_limit	correlation
>=0 & <70000	0.0596262
>=70000 & <140000	-0.07120218
>=140000 & <210000	0.1057813
>=210000 & <280000	0.1414786
>=280000	0.2806246

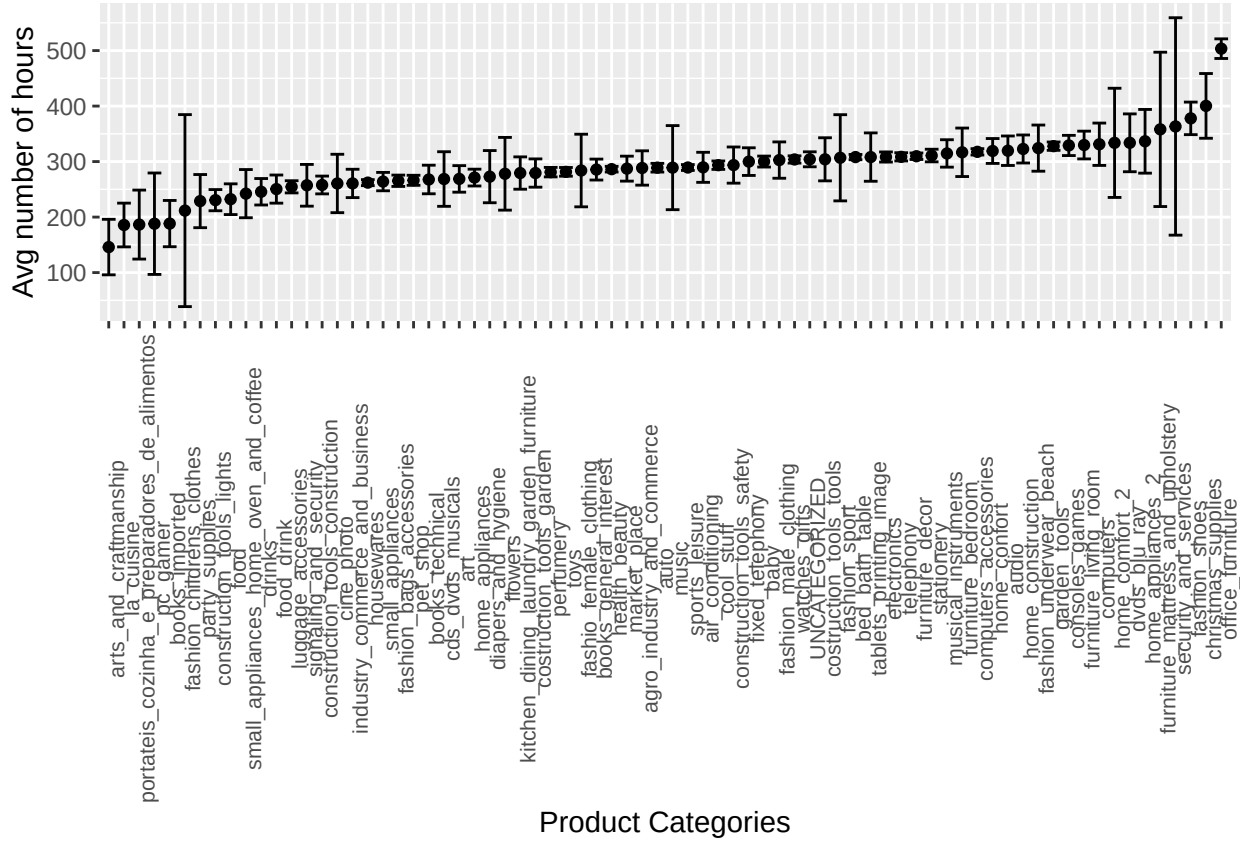
This behavior could be useful in estimating delivery timelines. We decided to consider this as a feature for further analysis for inclusion in the model.

Categorical features

Product category effects

We decided to look into our intuition that different product categories may have different delivery timelines and explore its influence on the prediction results.

Below errorbar plot depicts how average number of hours taken for delivery vary across product categories while at the same time capturing the confidence interval of number of hours taken for delivery for each product category.



In above plot, we noted that average number of hours taken for delivery vary considerably from one product category to another. So our intuition that product categories influence delivery time was confirmed.

Given the influence, we decided to consider this as a feature for further analysis for inclusion in the model. However, product categories are categorical in nature and they are quite a few categories, incorporating them directly would not have worked for many machine learning models which require numerical values. Additionally in some cases, each category could end up being treated as a predictor which is also the case with one-hot encoding.

However, to convert categorical variables to numeric values and still be able to limit features and make model training and evaluation less expensive, we applied additive smoothing based **target encoding** on the product categories attribute.

Additive smoothing based target encoding for a categorical variable $= (global\_mean * weight + N * u_t, c)/(weight + N)$

where $u_{t,c}$ is mean of target/outcome for a given category and weight is the smoothing factor or amount of regularization

But to do that, first we checked number of observations per product category, to understand the range of weights to consider. The target encoding could be erroneous if done with larger weight or against the random dataset.

To solve these, we identified the smoothing factor(or weight) to use by carrying out target encoding for different datasets for a set of weights, selecting single weight per dataset based on performance metric i.e. RMSE and then we took a mean of weights identified across the datasets.

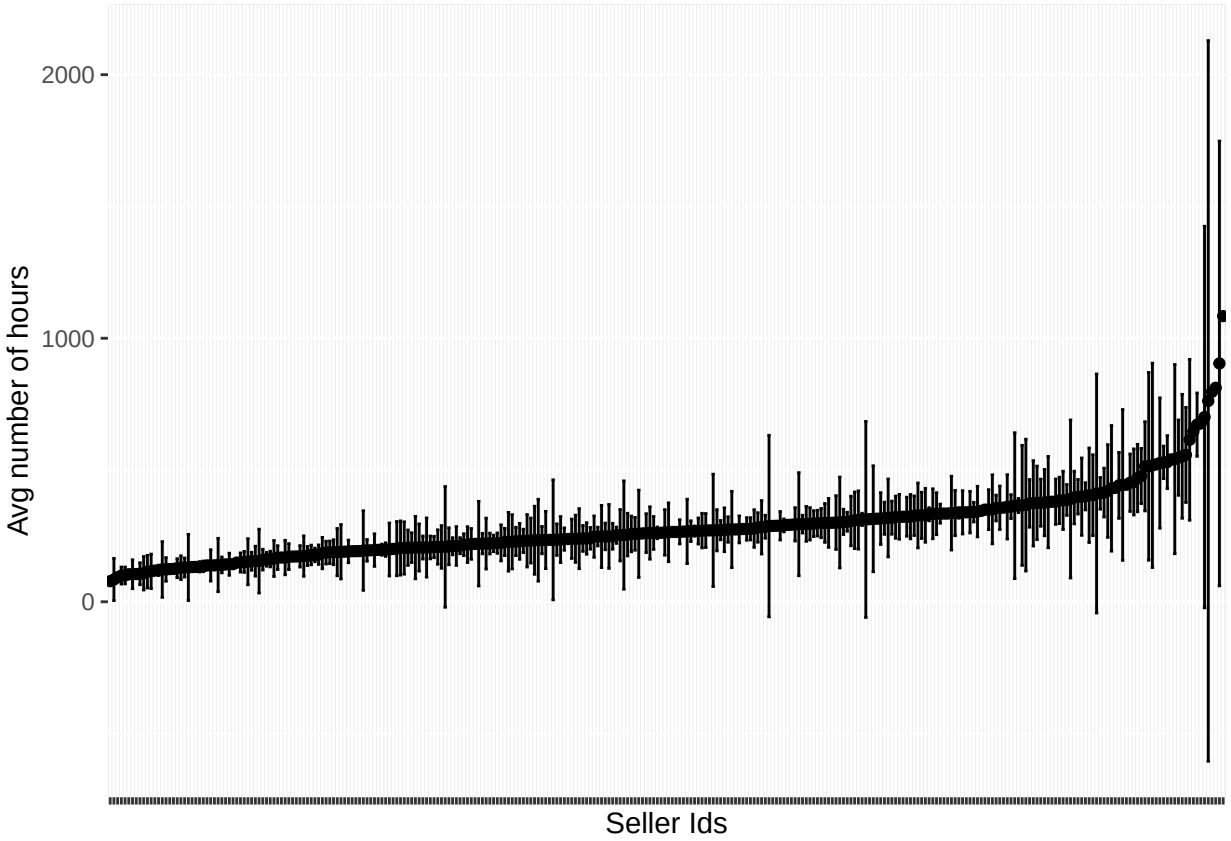
Once weight was identified, to avoid any randomness due to choice of the dataset, we computed additive smoothing based target encoding for product category attribute across 10 datasets and then took a mean for them to settle at the final target encoded values to use for the concerned feature.

For the above analysis, knn model was used, given that it does not make assumptions about the dataset, which was useful in the early phase of analysis. However, to use knn model, we **scaled and normalized** the features as knn model uses Euclidean distance. Scaling and normalization was carried our using training data and then used for test data.

Seller id effects

Next we looked into the intuition that orders for products from different sellers based on what they sell, how they operate could influence delivery timelines.

Below errorbar plot depicts how average number of hours taken for delivery vary across sellers while at the same time capturing the confidence interval of number of hours taken for delivery for each seller. Since there were **2854** sellers, we sampled **300** of them to represent their influence effectively in the graph.



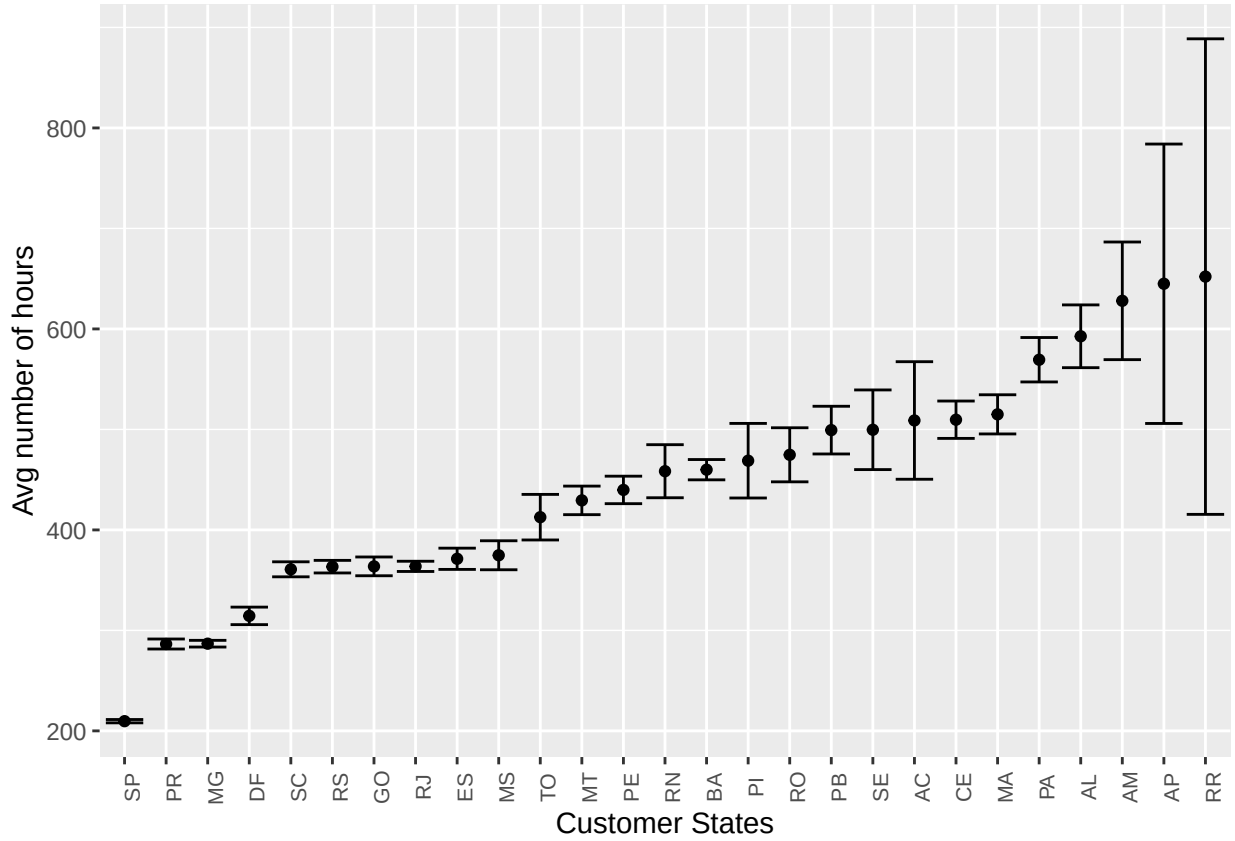
In above plot, we noted that average number of hours taken for delivery vary considerably from one seller to another. So our intuition that sellers influence delivery time was confirmed.

Given the influence, we decided to consider this as a feature for further analysis for inclusion in the model. However, seller ids similar to product categories are categorical in nature and there are **2854** sellers, which are a lot. Incorporating them as it is would not have worked for many machine learning models which require numerical values. As a result, for seller ids we applied additive smoothing based target encoding and followed strategy used for product categories.

Customer state effects

Next we looked into the intuition that customer states can have different regulations, paper work, approvals which could influence delivery timelines.

Below errorbar plot depicts how average number of hours taken for delivery vary across customer state while at the same time capturing the confidence interval of number of hours taken for delivery for each customer state.



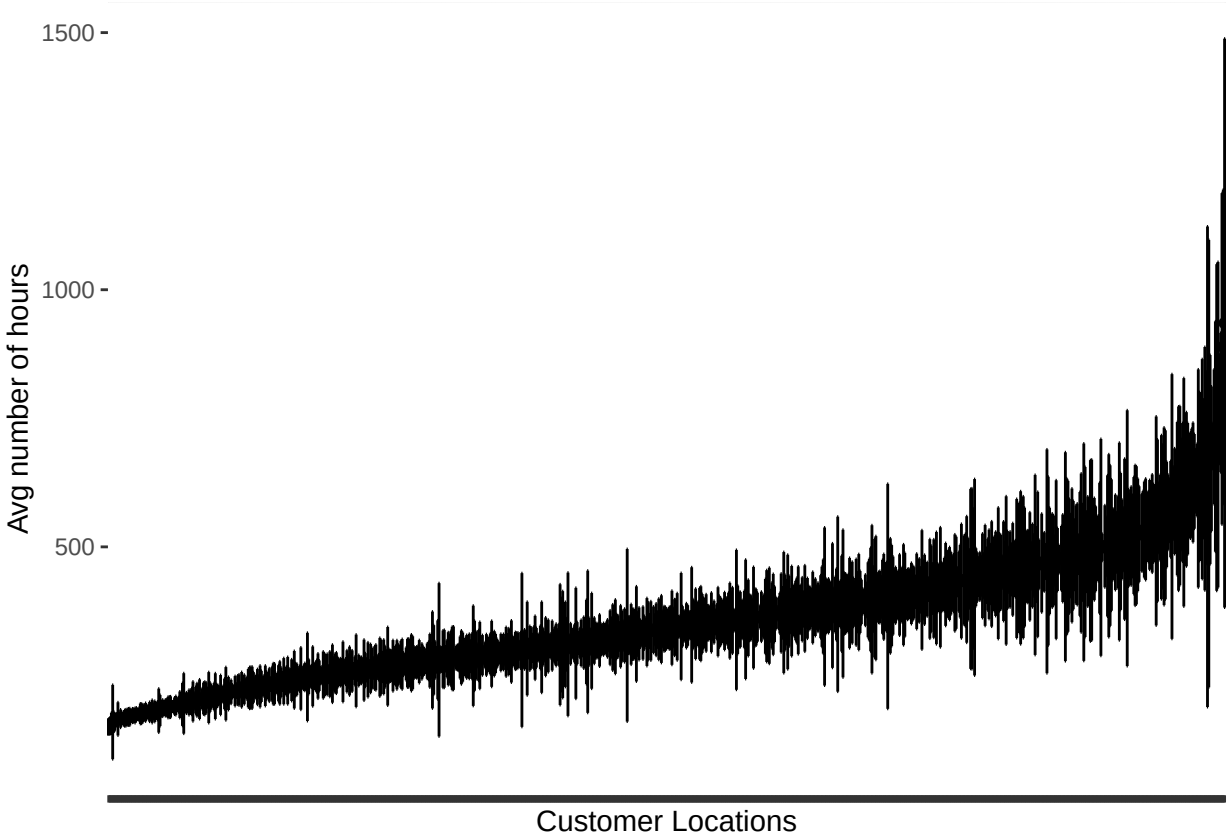
In above plot, we noted that average number of hours taken for delivery vary considerably from one customer state to another. So our intuition that customer state influences delivery time was confirmed.

Given the influence, we decided to consider this as a feature for further analysis for inclusion in the model. However, customer states similar to product categories are categorical in nature and there are **27** customer states. Incorporating them as it is would not have worked for many machine learning models which require numerical values. As a result, for customer states we applied additive smoothing based target encoding and followed strategy used for product categories.

Customer location(loc) effects

Next we looked into the intuition that rough information about customer location i.e. combination of co-located zip codes can influence the delivery timelines as customer location can be remote or require different means of transport, among others.

Below errorbar plot depicts how average number of hours taken for delivery vary across customer locations while at the same time capturing the confidence interval of number of hours taken for delivery for each customer location.



In above plot, we noted that average number of hours taken for delivery vary considerably from one customer location to another. So our intuition that customer location influences delivery time was confirmed.

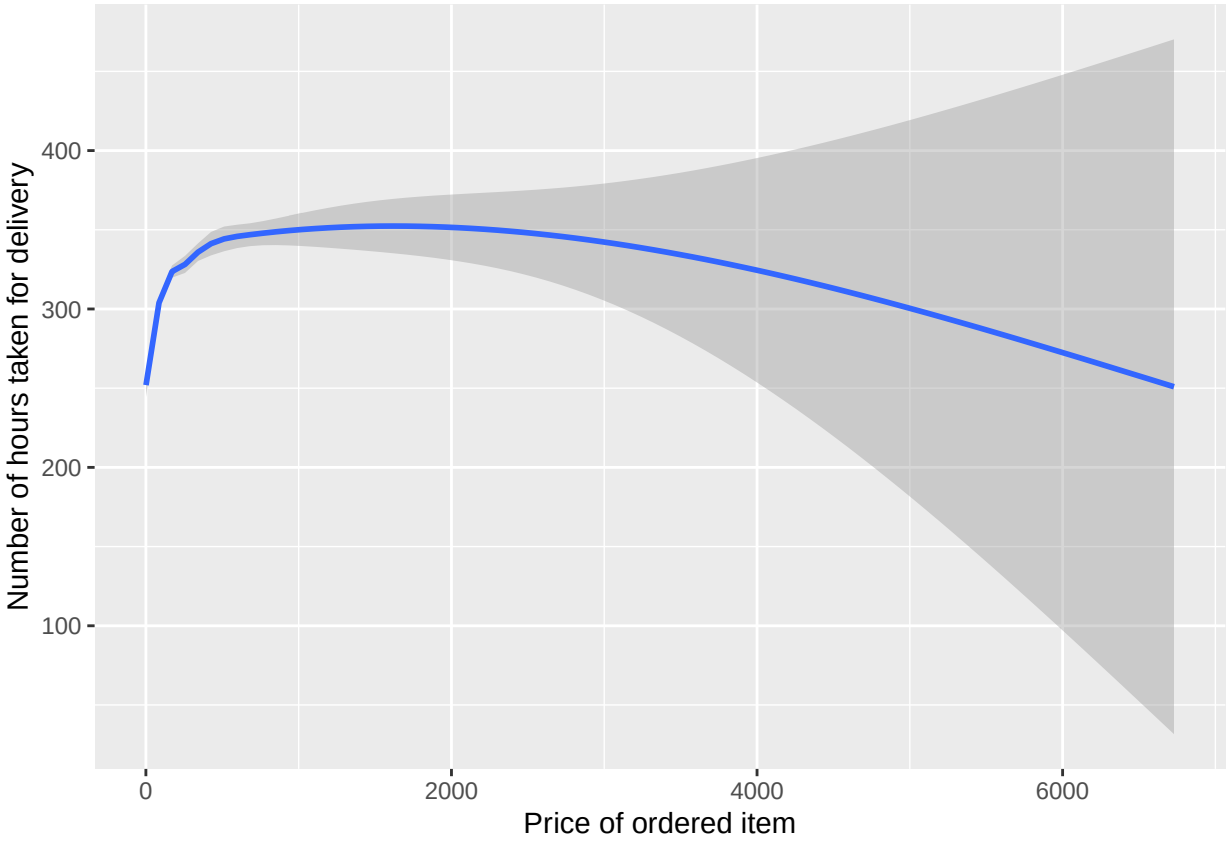
Given the influence, we decided to consider this as a feature for further analysis for inclusion in the model. However, customer locations similar to product categories are categorical in nature and there are **849** customer locations. Incorporating them as it is would not have worked for many machine learning models which require numerical values. As a result, for customer states we applied additive smoothing based target encoding and followed strategy used for product categories.

Other features

Price feature

We looked into the intuition that order item's price could influence the delivery timelines as expensive items ordered may have priority shipping or be delivered with caution.

To understand its influence on the delivery timelines, we plotted numbers of hours taken for delivery against the price of the ordered item. In order to see the trend, gam method based smoothing curve was plotted on the graph.



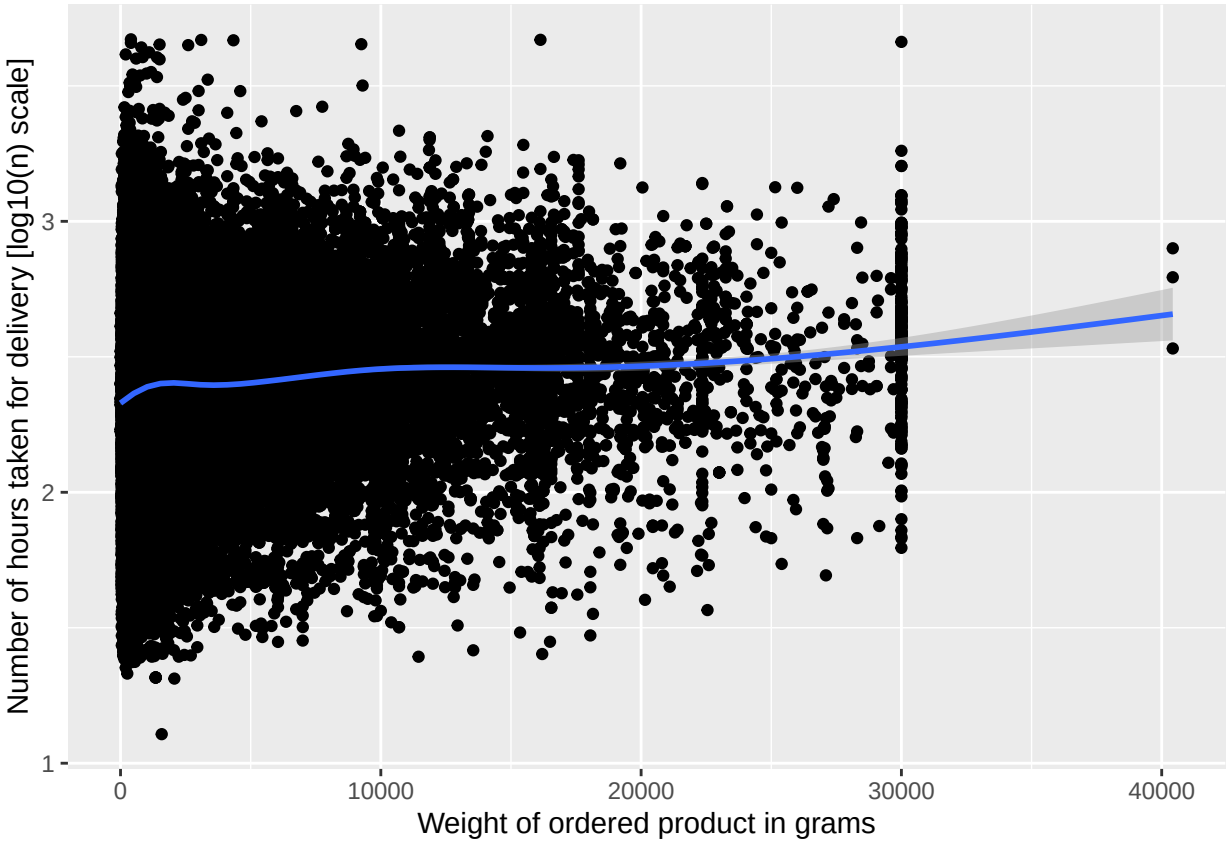
In above plot, we noted that number of hours taken for delivery vary as the price of the ordered item changes. It appeared to increase significantly as the price of ordered item increased and decreased when the price reached towards higher limits. Though confidence interval indicated higher variability in delivery timelines at higher prices.

This behavior could be useful in estimating delivery timelines. We decided to consider this as a feature for further analysis for inclusion in the model.

Product weight feature

Next we looked into the intuition that product weight can influence the delivery timelines as heavier items like furniture, bed may be transported slowly compared to lighter items.

For this we plotted numbers of hours taken for delivery against the weight of the product ordered in grams. In order to see the trend in this over-plotted graph, gam method based smoothing curve was also plotted on the graph.

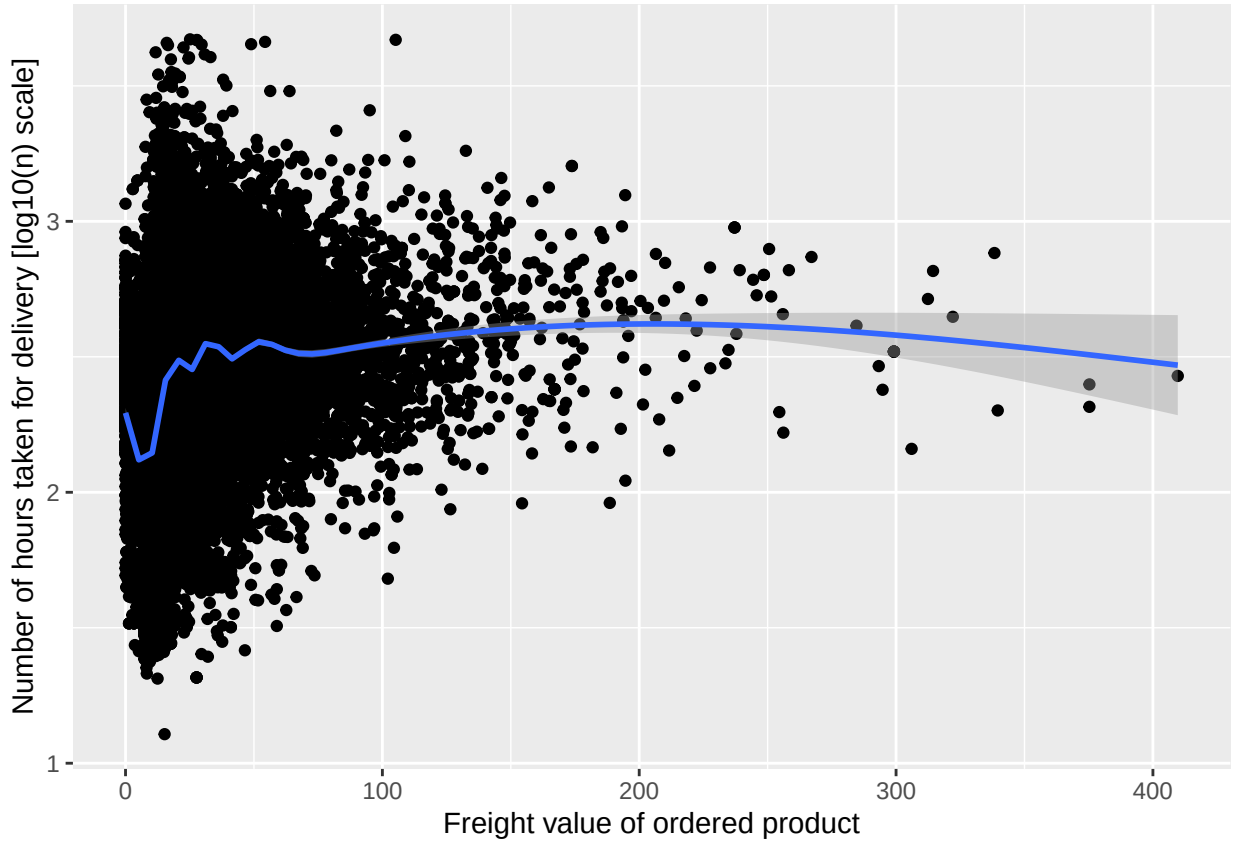


In above plot, we noted that number of hours taken for delivery vary as the product weight changes. This behavior could be useful in estimating delivery timelines. We decided to consider this as a feature for further analysis for inclusion in the model.

Freight value feature

Next we looked into the intuition that freight value can help determine the delivery timelines as higher freight value could indicate complexity in shipment, extra processing which would need time, among others.

For this we plotted numbers of hours taken for delivery against the freight value of the product ordered. In order to see the trend in this over-plotted graph, gam method based smoothing curve was also plotted on the graph.



In above plot, we noted that number of hours taken for delivery vary as the freight value of product changes. It appeared to increase as freight value increased for smaller freight values, then plateaued and decreased as freight value reached higher limits The decrease could be due to priority shipping coming at a cost. This behavior could be useful in estimating delivery timelines. We decided to consider this as a feature for further analysis for inclusion in the model.

Model Selection

Post evaluation with multiple features, the final set of features identified for model selection were as below::

Table 14: Features List

Feature
price
freight_value
product_weight_g
number_of_hours
volume
distance_between_two_points
mean_delay_in_handover
mean_delay_in_payment_approval
product_category_effect
seller_id_effect
customer_state_effect
customer_loc_effect

To start building our model, training and test subset were prepared. For this, we further partitioned our training data set i.e. training_dataset into a split of 80/20 between train_subset and test_subset to ensure we use only the train_subset for training and tuning the model and test_subset to evaluate the model and measure RMSE. This would ensure there is no data leak. The split like earlier was done such that distribution of outcome i.e. number_of_hours was similar in both training and test data.

The derived features selected earlier were prepared based on train_subset and applied over test_subset, to ensure we maintain the environmental condition where test data is not known to us.

In addition, as earlier shared, categorical features were converted to numeric form with the help of additive smoothing based target encoding with a smoothing factor/weight term computed earlier to avoid overfitting. Also, any undesired features were removed, reducing train_subset and test_subset to only the desired features.

k-nearest neighbors(knn) model

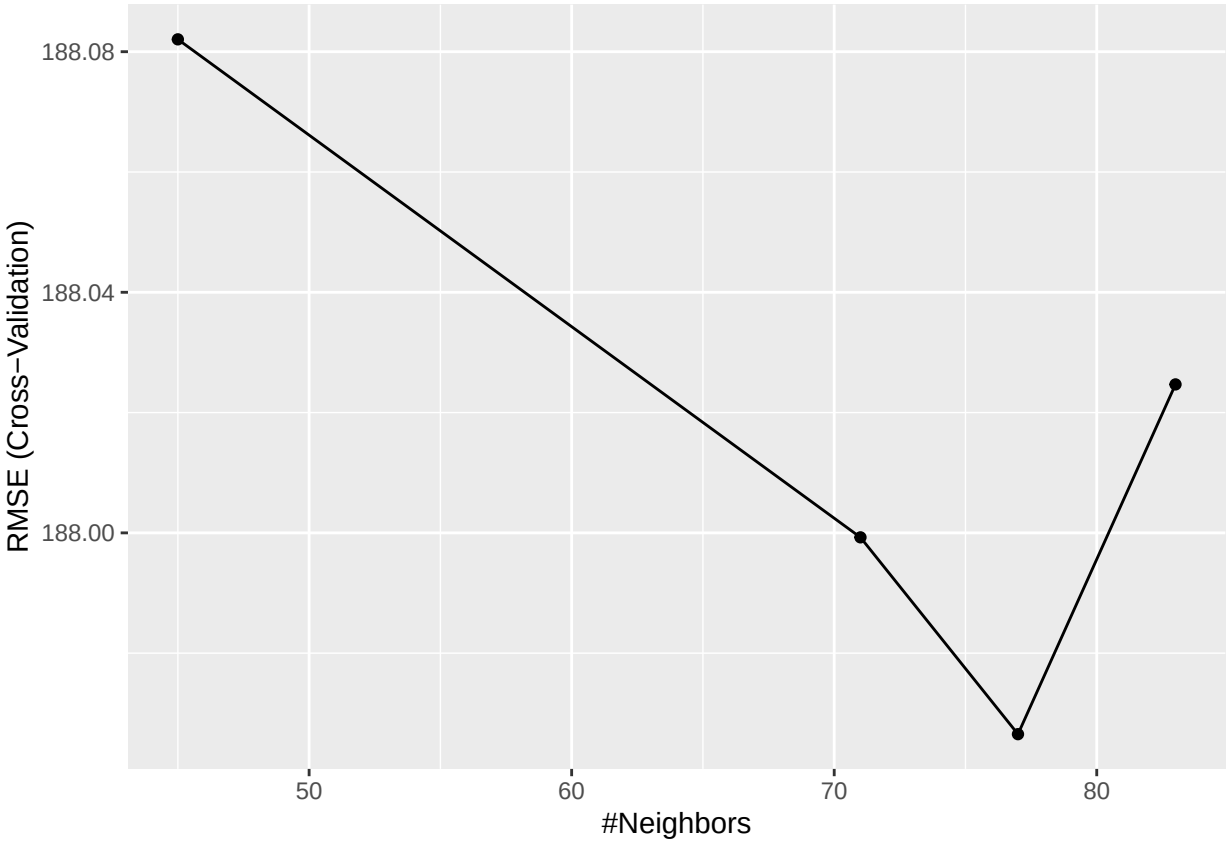
k-nearest neighbors(knn) relies on finding nearest neighbors and uses average of the nearest neigbbors for prediction. In addition, it does not make assumptions about the dataset. In our case limited dataset and that the alogrithm does not make any assumptions, made it a good candidate for model development and performance evaluation.

To use knn model, features were scaled and normalized to prevent bias towards any features, as knn internally calculates Euclidean distance for finding the nearest data-points.

In addition, for knn model training and tuning, we carried our 5-fold cross validation providing a range of values of k to use for training the model. Here, k means number of neighbors to be considered. The cross validation helped in removing any randomness from our tuning parameters. In addition, this helped reduce the probability of overfitting.

Details of the trained knn model can be found below:

```
## k-Nearest Neighbors
##
## 70155 samples
##    11 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 56124, 56125, 56124, 56123, 56124
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##   45  188.0821  0.3029086  117.3435
##   71  187.9993  0.3042030  117.2367
##   77  187.9665  0.3045986  117.2025
##   83  188.0247  0.3042821  117.2253
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 77.
```



Based on model results, k value of 77 was identified to be the one with lowest RMSE.

Below is the summary of each model and knn model(which used partial dataset) against the performance metric i.e. RMSE

Table 15: Performance Metric(RMSE) Summary

type	value
Current RMSE Max	359.9466
Current RMSE Min	377.9515
Knn model(partial dataset)	183.0288

As we can see the performance metric i.e. RMSE is significantly better for knn model (though with partial dataset at this point of time), than the one being used by Olist store.

Random forest model

Next, we used Random forest model, which is basically ensemble of multiple decision trees. The training dataset is split into multiple subsets. For each such subset, decision tree is created. However, randomness is introduced in creation of decision trees as at each node, the tree leverages a number of random features to determine the best split. Once the various models are ready, they are used to prepare predictions. The final prediction is reached by taking average of all the trees.

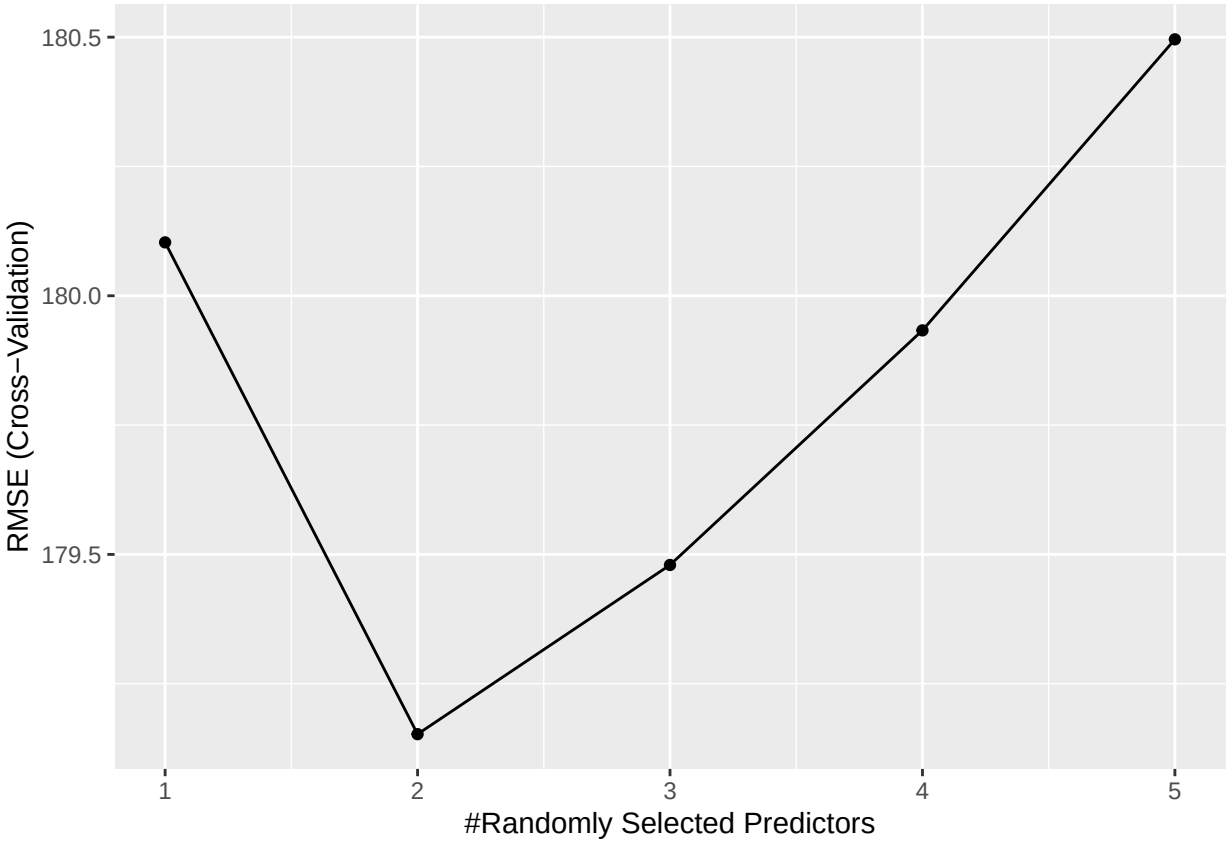
The ensemble of multiple decision trees helps prevent overfitting and creates a more stable prediction. While, the number of random features at each node of decision tree to determine best split, helps reduce bias and

variance in the model. It is these qualities of random forest, which prompted us to consider this for model development and evaluation.

To train and tune random forest model, we carried our 4-fold cross validation providing a range of values of mtry to use for training the model. Here, mtry means number of random features used to determine the best split at each node of the decision tree. In addition, we configured the number of decision trees to create as 200. Here, the cross validation not only helped to reduce the probability of overfitting but also helped in identifying the tuning parameter i.e. mtry which would work across randomly selected datasets.

Details of the trained random forest model can be found below:

```
## Random Forest
##
## 70155 samples
##    11 predictor
##
## No pre-processing
## Resampling: Cross-Validated (4 fold)
## Summary of sample sizes: 52616, 52615, 52615, 52619
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   1     180.1032  0.3633030  109.9591
##   2     179.1525  0.3679767  108.7860
##   3     179.4796  0.3658839  108.9956
##   4     179.9331  0.3631803  109.4209
##   5     180.4957  0.3596871  109.7970
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```



Based on model results, mtry value of 2 was identified to be the one with lowest RMSE.

Below is the summary of each model and random forest model(which used partial dataset) against the performance metric i.e. RMSE

Table 16: Performance Metric(RMSE) Summary

type	value
Current RMSE Max	359.9466
Current RMSE Min	377.9515
Knn model(partial dataset)	183.0288
Random forest model(partial dataset)	173.3481

As we can see the performance metric i.e. RMSE was significantly better for random forest model (though with partial dataset at this point of time), than the one being used by Olist store. In addition, we could see improvement in RMSE when compared to knn model for the same dataset.

Ensemble model

The ensemble models use multiple algorithms and if properly developed, they can be built to take learnings from the different models and generate better predictions. Since we had knn and random forest models tuned and prepared, we used them to create variety of ensemble models.

Average based ensemble model

We took previously trained and tuned knn and random forest model and created ensemble of both. For this, predictions using both knn and random forest models were computed. Then mean of the predicitions was taken. This was used to calculate performance metric i.e. RMSE.

Below is the summary of each model and ensemble model(which used partial dataset) against the performance metric i.e. RMSE

Table 17: Performance Metric(RMSE) Summary

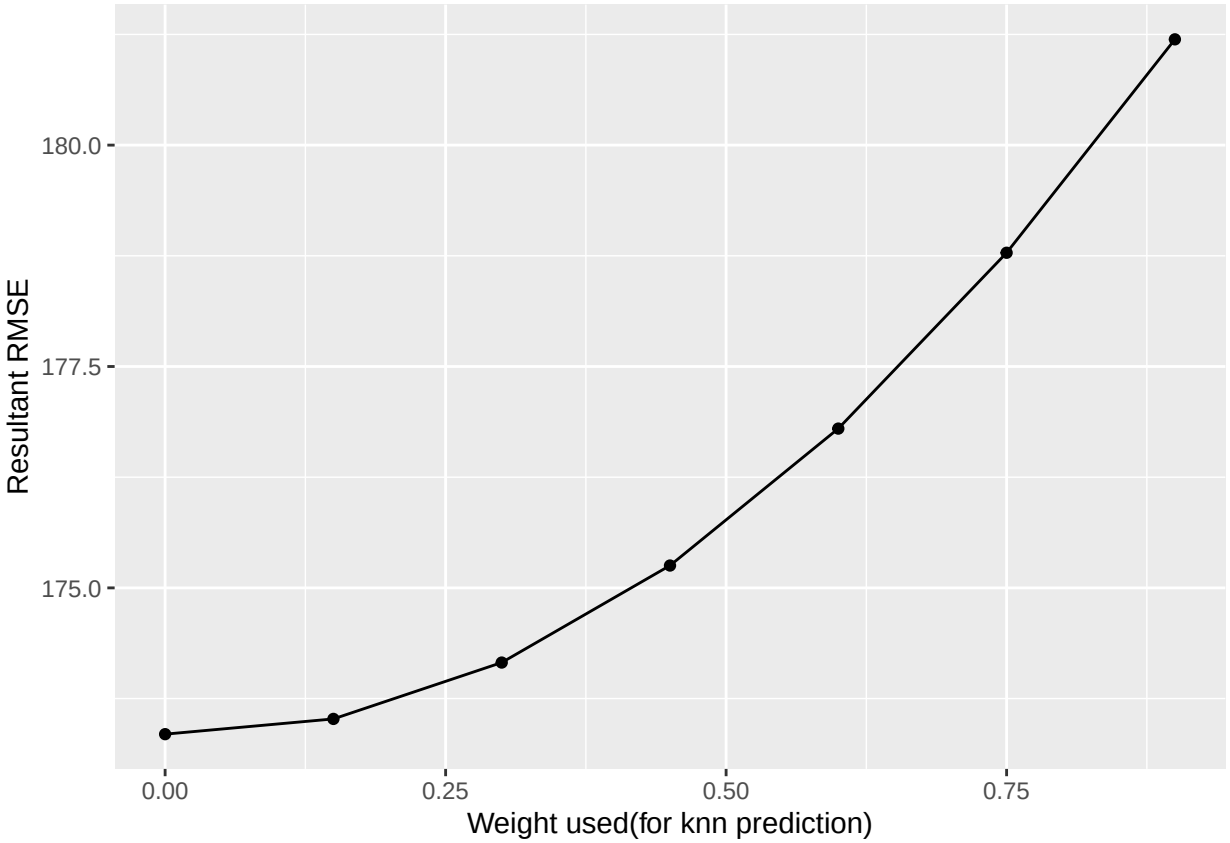
type	value
Current RMSE Max	359.9466
Current RMSE Min	377.9515
Knn model(partial dataset)	183.0288
Random forest model(partial dataset)	173.3481
Avg based ensemble model(partial dataset)	175.7183

As we can see the performance metric i.e. RMSE was significantly better for ensemble model (though with partial dataset at this point of time), than the one being used by Olist store. However, it was not better than the performance metric captured for random forest model.

Weighted average based ensemble model

Other alternative ensemble technique like weighted average approach was explored. Again, we took previously trained and tuned knn and random forest model and created ensemble of both. However, this time, final predictions were based on weighted average of the predictions of the constituient models.

The results of performance metric i.e. RMSE for the ensemble model against increasing weights for knn model based predictions are plotted on the graph.



We noted that weighted ensemble model had lowest RMSE of 173.348081 at weight=0

In addition, other ensembling techniques like stacking were explored, however they could not improve the RMSE.

Selected model

Given above results of performance metric i.e. RMSE across knn, random forest and ensemble models, we selected random forest as the final model for our delivery prediction system.

Results

Post model tuning and evaluation, random forest model emerged as the best fit for our data among the models considered for selection. The tuning parameters identified for the final random forest algorithm are shared below:

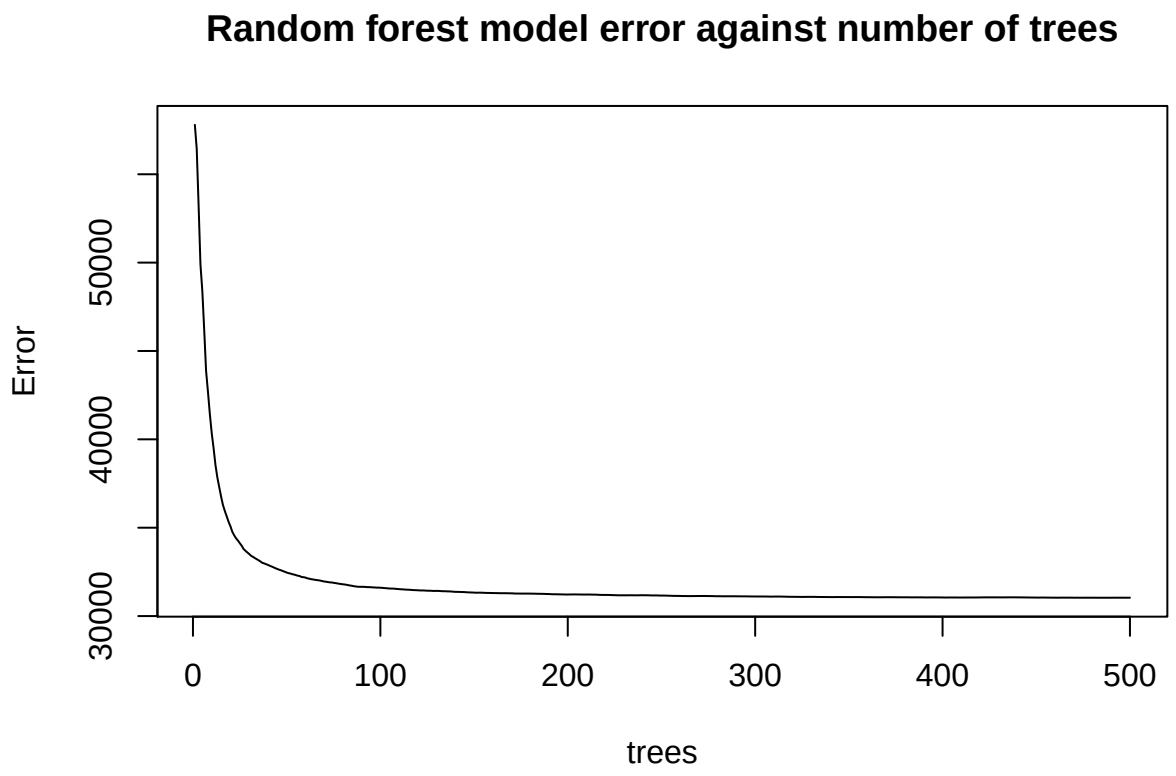
Table 18: Random forest tuning parameters

tuning_parameter	value
mtry	2
number_of_trees	500

Till this point, we had used only training data set i.e. training_dataset to identify the features, evaluate various models and tune them. Having selected the final model, we prepared first the derived features, which

as shared earlier were prepared on training_dataset. The features were then applied over validation_dataset, which was the dataset kept aside till now.

In addition, categorical features were converted to numeric form with the help of additive smoothing based target encoding with a weight term computed earlier for purpose of smoothing and to avoid overfitting. Also, any undesired features were removed, reducing training_dataset and validation_dataset to only the desired features.



Below is the summary of models against the performance metric i.e. RMSE

Table 19: Performance Metric(RMSE) Summary

type	value
Current RMSE Max	359.9466
Current RMSE Min	377.9515
Knn model(partial dataset)	183.0288
Random forest model(partial dataset)	173.3481
Avg based ensemble model(partial dataset)	175.7183
Final model	188.5793

For the **final model**, the performance metric i.e. RMSE = **188.5793107** captured against **validation_dataset** was much better than what Olist store had as identified from the dataset. The newly built model can be used for predicting number of hours required for delivery of a given ordered item.

Conclusion

As part of this project, the goal was to build a delivery prediction system which could predict the number of hours required to delivered the ordered item rating with minimal RMSE.

During the course of project, we carried out data cleaning, identified features and derived features from existing information available. In addition, converted categorical features to numeric form and applied various techniques like additive smoothing based target encoding, scaling and normalization as required.

Once the feature analysis and selection was completed, we used the features to tune models like k-nearest neighbors, random forest. We carried out k-fold cross validation to remove any randomness from the model training and tuning process. This additionally helped to reduce probability of overfitting. Post this, we explored multiple ensemble models using techniques like averaging, weighted average, stacking, among others. Finally we selected random forest based model and the model so trained performs within desired range of RMSE and can be useful for predicting number of hours required for delivery of a given ordered item. Due care was taken to prevent overfitting from influencing our model.

While the model is a significant improvement over what Olist store had, there is lot of room to improve it. It can be improved for incremental learning so that it can learn from new orders and their delivery details once they become available. In addition, it could be improved to handle new sellers or product categories.

Having said that the delivery prediction system built as part of the project is performant and can be leveraged to develop further.