DoorDash Arrival time Project Report

Justin Chen

March 2023

1 Introduction

Apps like DoorDash, Uber, SkiptheDishes, etc. aim to deliver food orders to customers in a timely and efficient matter. However, a big part of the customer experience is having a somewhat accurate 'time to arrival'. The goal of this project is to predict the arrival time of DoorDash deliveries using a 200,000-row data set with 16 features. In particular, The objective of the project will be to predict the total seconds between "created at" and "actual delivery time" (both features given in the dataset). This will give us the total time it takes for a DoorDash order to reach the customer.

1.1 Motivation

Before moving on to the methodology and results I want to be a little more specific on the implications of this project. The time-to-arrival feature has implications for the overall customer experience and ecosystem of DoorDashes business. We want to tailor the customer's expectations to be as close to reality as possible. We aim to keep customers 'in the loop' when they order so they aren't wondering where their order went. If we present an arrival time thats much larger then what it actually is, this may lead to customers withdrawing orders they otherwise wouldn't have because of the inflated time commitment. If we present an arrival time much less then what it actually is, this will lead to customer dissatisfaction/disappointment upon the actual time to arrival. This also trickles down to the actual drivers since customers are likely to blame the drivers for this discrepancy leading to possibly fewer tips and fewer drivers willing to work for DoorDash. Hence why we need a 'ballpark' arrival time to close this gap between customer expectations and reality.

1.2 Details of Problem

This is a typical regression problem so we will tailor our methods accordingly. Methods like Linear Regression, Random Forests, and Boosted Trees, may be useful. We also aim to reduce the dimensions of our data using methods like Principal component analysis and checking for multicollinearity. This is a project that has lower stakes and messy data. Something like 'arrival time' has so many variables and unpredictable elements that contribute. Thus, we aim to give the client an approximate time to arrive which is within 10-20 minutes of the actual arrival time. This potential error has less impact the higher the actual arrival time is (waiting an extra 10 min from an hour is less significant than from a 5 min order). As we will see this is a fair margin of error given the average time to arrival of our data. The actual Root Mean Squared Error of the final model is within this interval (off by about 18min), which is in line with two other sources that tried to tackle this problem.

2 Methods

This is a big data project with the goal of predicting a target variable given a bunch of predictors. We need to clean/manipulate the data set, create new features to optimize our prediction, and finally fit a regression model or a deep learning model.

2.1 Cleaning/Processing the Data

First, create some features. The first feature is the 'time to arrive' this will be the target variable and is simply created by taking

$$\textbf{created at - actual delivery time = time to arrive} \tag{1}$$

The second engineered feature will represent the number of dashers available to take the order. This is of obvious importance given that the supply of drivers available will affect how fast orders are picked up and delivered.

$$\textbf{total busy dashers / total on-shift dashers = total free dashers} \tag{2}$$

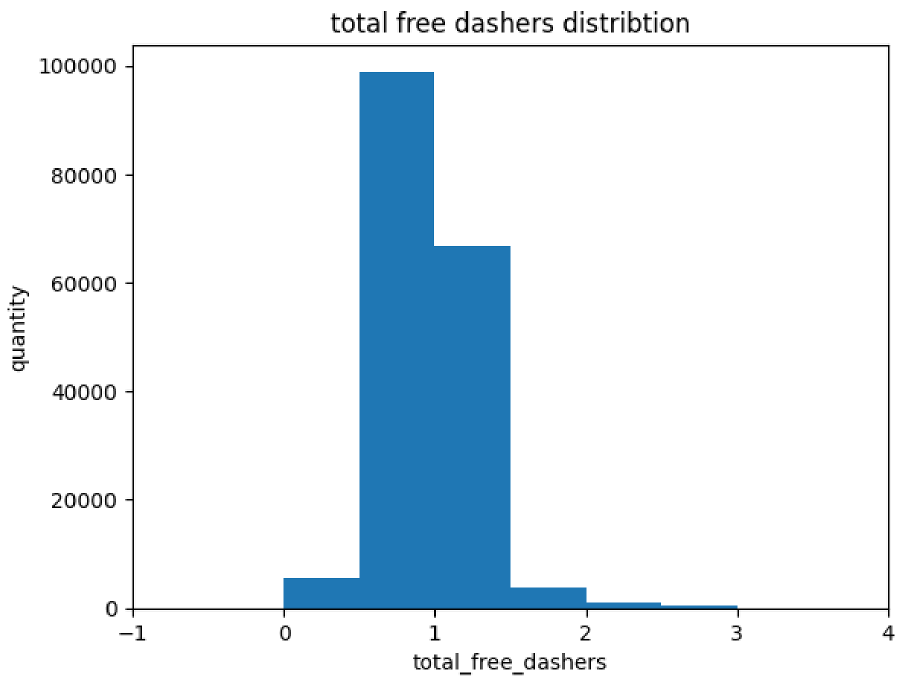
The dataset comes with a bunch of missing values that need to be dealt with. The following is a table of all the features and the corresponding null values

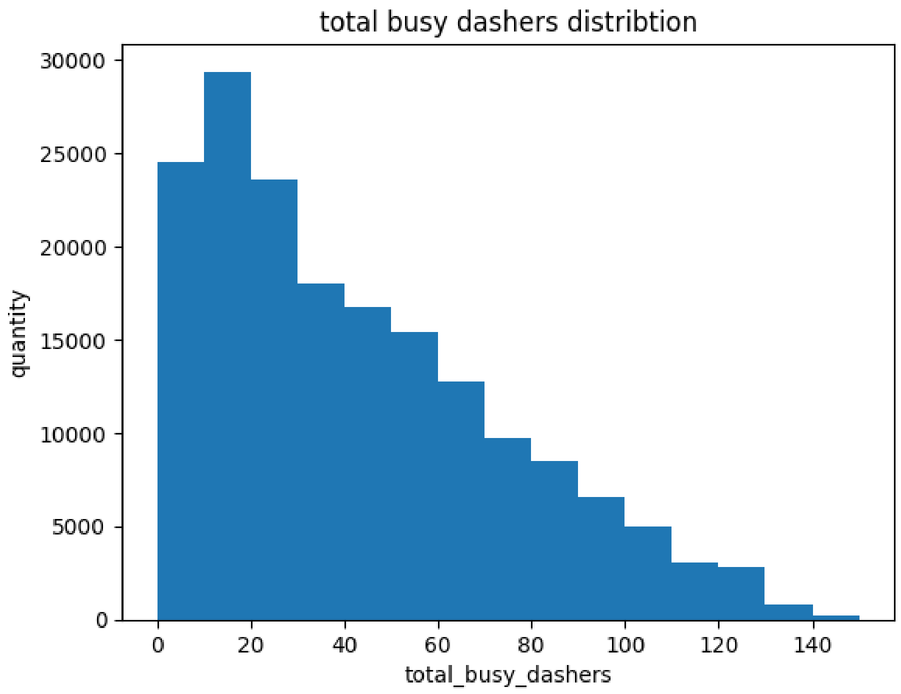
Feature	Total Null Values
market id	987
created at	0
actual delivery time	7
time to arrive	7
store id	0
store primary category	4760
order protocol	995
total items	0
subtotal	0
num distinct items	0
min item price	0
max item price	0
total on-shift dashers	16262
total busy dashers	16262
total free dashers	19838
total outstanding orders	16262
estimated order place duration	0
estimated store-to-consumer driving duration	526

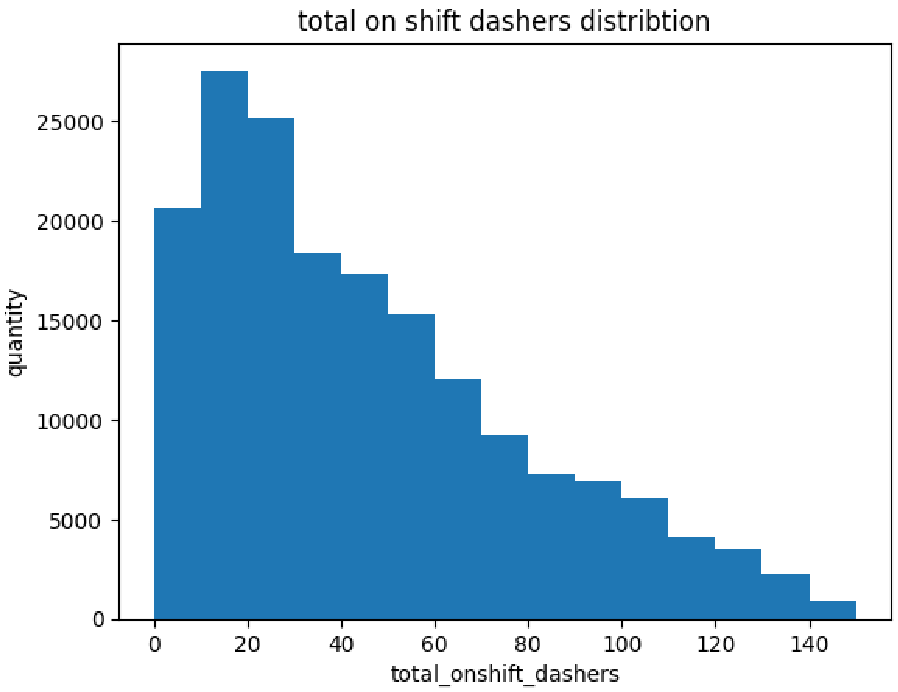
As a baseline, anything less than 1000 can be deleted. Since there is around 200,000 rows deletion of 1000 shouldn't have a huge impact on the results. So there are 5 features that need to be dealt with.

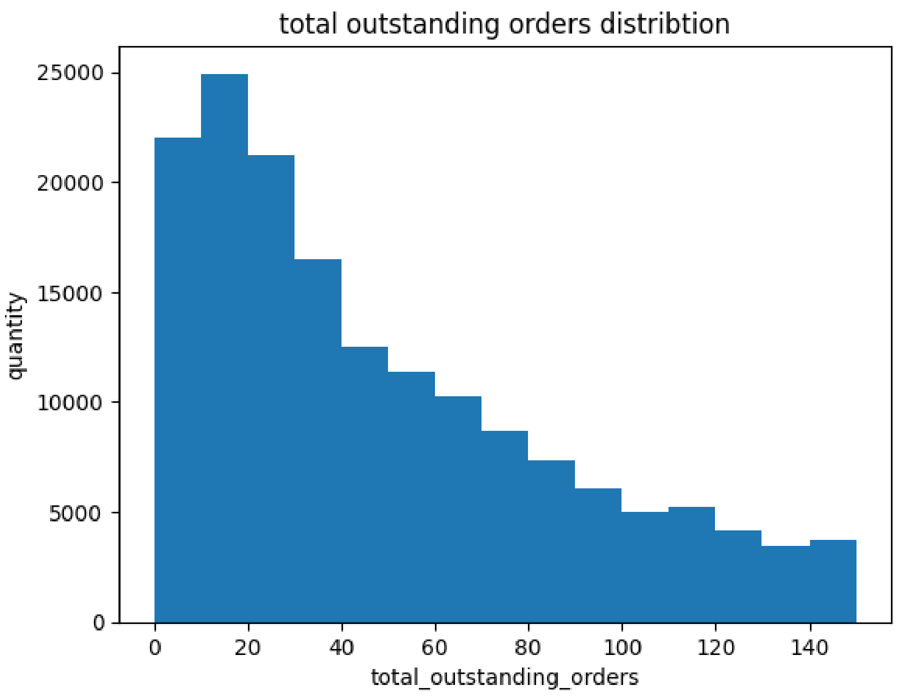
2.1.1 total busy dashers, total on-shift dashers, total outstanding orders, and total free dashers

Firstly, many of the null values from total free dashers can be attributed to the situation in which the total on-shift dashers are 0. It is of course impossible to divide by 0, hence the null. There are 4452 rows where this is the case, this value can be dealt with by dropping these rows. This may contradict the baseline we set earlier, however, 4452 rows represent about 2% of the dataset. This shouldn't throw off the estimates since this value is insignificant. The rest of the 16262 values must be dealt with. Take a look at the distribution of the values.


total free dashers distribtion
quantity
total_free_dashers
100000
80000
60000
40000
20000
0
−1
0
1
2
3
4


total busy dashers distribtion
quantity
total_busy_dashers
30000
25000
20000
15000
10000
5000
0
0
20
40
60
80
100
120
140


total on shift dashers distribtion
quantity
total_onshift_dashers
0
5000
10000
15000
20000
25000
0
20
40
60
80
100
120
140



Notice that the total free dashers ratio is the majority of the time around 1. As for the other features, we notice an obvious right skewness in the data. Impute these values with the median. In general, when we observe skewness median is a better imputation method than the mean. Since a right-skewed curve represents data that is mostly represented with lower values, the values on the positive side are seen as outliers. This is the one thing the mean is bad at accounting for and thus the median is what should be used.

2.2 Feature Selection

Before continuing with the analysis it's important to filter out certain features that will be of little importance. The following are the methods employed to detect which features are worth leaving in and which are not.

2.2.1 K-Means Clustering

K-means clustering, intuitively, aims to group the data into homogeneous clusters. This allows us to observe separations in the data i.e, how one can group the data into defined categories. Mathematically, the goal of the algorithm is to initialize the K number of centroids(centers of clusters) and match the data to these centroids based on the minimal geometric distance (Frobenius Norm for matrix A = $||A||_F = \sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|$) of the centroid to the data point. This will ideally form K clusters of the data. Mathematically the algorithm is,

$$\textbf{X = Data Matrix C = Centroid Matrix, W = Assignment Matrix} \quad (3)$$

X is the column-wise representation of the P-number of points in the dataset. C is the column-wise representation of the K centroids. W matches entries in X to the appropriate centroid. It does this by multiplying standard basis vector e_k with C. So given that a x_i matches with i-cluster the result should be:

$$[c_1, c_2, ..c_i..., c_k] * [e_i] \approx x_i \quad (4)$$

The actual algorithm has the following steps:

$$\textbf{Begin Until C converges} \quad (5)$$

Update W (assign each data point to the closest cluster)
$w_p = e_{k*}$ where $k = \arg\min_{k*=1,....,K}\{||c_k - x_p||_F^2\}$
Update C
Each centroid becomes the average of its current points
Let S_k denote the points in cluster k

$$c_k = \frac{1}{|S_k|} \sum_{p \epsilon S_k} x_p \quad (6)$$

The aforementioned algorithm will be used to detect categorical feature importance. Intuitively, if there is a true categorical separation in the data there should be minimal overlap between clusters, in other words, each point should belong in one cluster. To measure this error, use 'inertia', a within-cluster sum of squares distance between the points in the cluster and the centroid:

$$\sum_{i=0}^{n} \min_{c_k \epsilon C}(||x_i - c_k||_F^2) \quad (7)$$

The above equation is the summation of the minimum Euclidean distance between points within a cluster and a centroid. The total error of the whole algorithm calculates the above quantity for each cluster, then takes the average. This will be the y-axis of the elbow plot, the x-axis will be the number of clusters. The goal is to analyze the number of clusters that corresponds to the least error. However, dropping the categorical variables will be of interest if the error is sufficiently high for all observable values of clusters. Since this would indicate there is no clear homogeneity between clusters.

2.2.2 Principal Component Analysis

After applying the k-means clustering algorithm, one can visualize class separation between the data points by using principal component analysis. Visualizing a 10+ dimensional data set is not possible under ordinary means, we can however plot two principal components on a cartesian plane. Principal component

analysis aims to find the direction of most variance in the feature space. It then projects the points onto a lower-dimensional feature subspace that is in the direction of that variance. This preserves the geometry of the data and allows for insights to be derived visually. Formally, there are P data points $X = [x_1, ..., x_P]$ each with dimension N (represents the number of features each data point has). Choose some $K < N$. PCA aims to find K basis vectors $c_1, ..., c_k$ such that we can approximate any x_i $1 \leq i \leq P$ via a linear combination.
Represent $C = [c_1| \cdots |c_p]$ and $w_i = [w_{1,i}, ..., w_{k,i}]$
$|c_i|$ is the column-wise representation of each basis vector. C_i is thus, an NxK matrix.
w_i, in this case, represents the weights of the basis vector that approximate x_i.

$$Cw_i \approx x_i \tag{8}$$

This can be written in general for all $x_i's$. $W = [w_1| \cdots |w_p]$ where each w_i represents the KxP column of weights corresponding to each value of x_i
PCA thus tries to minimize:

$$\min_{C,W}\{||CW - X||_F^2 \tag{9}$$

Which quantifies the distance between the linear combination of the basis vectors and the actual data points. The goal is to make K=2, this will shrink the data into a two-dimensional form. Use color coding based on an existent categorical variable to identify separations (or lack thereof) in the data. The beauty of principal component analysis is the preservation of the geometry of the data. The idea would be to apply k-means clustering first, then visualize the unimportance or importance of certain categorical features using PCA. Also, it's important to normalize/scale the data before applying principal component analysis. This is to first of all ensure that all values are measured the same. Each feature has different magnitudes and thus will have different degrees of variance. This gives a more robust way to calculate direction of variance which is consistent with the data. Normalizing has the following equation: Let μ = mean, σ = standard deviation, x_i =observation.

$$\frac{x_i - \mu}{\sigma} \tag{10}$$

2.2.3 Collinearity

Another way to ensure optimal model accuracy and simplicity is to measure the collinearity of the features. First, collinearity is the linear relationship between two features say, X and Y. We measure on a scale of -1 to 1. -1 represents a perfect negative linear relationship. Meaning, as X increases Y always decreases (and vice versa for decreasing and increasing). 1 represents a perfect positive linear relationship. Meaning as X increases, Y always increases (and vice versa for decreasing). 0 being no linear relationship at all (features are completely uncorrelated). Ideally, every feature should have 0 colinearity with every other feature. Given two features that are highly correlated (close to 1 or -1), a new feature should be engineered to represent both. It also affects the standard error of the parameter estimates. If a regression coefficient is estimated from a correlated feature, the standard error from both features is inflated and influenced by each other, making prediction difficult. Mathematically, collinearity is represented by the proportion of covariance divided by the individual standard error of each feature. Take features X and Y, the collinearity (R^2)is calculated with:

$$R^2 = \frac{COV(X,Y)}{SE(X)SE(Y)} \tag{11}$$

The covariance quantifies how similar X and Y are in terms of movement. Intuitively, if X is a large number at the same time and Y is a small number, the variance between the two will be larger. This value is divided by the independent standard error of each variable, which quantifies how far X and Y deviate from the mean independently, the multiplication of the two can be seen as the direction in which both variables are going, independently. One can see that when $R^2 = 1$ this corresponds to the joint direction of X and Y equalling the independent direction of X and Y. Thus, perfect correlation.

2.3 Linear Regression

The least squares linear regression model aims to fit a typical linear line to the data by minimizing the distance between the predicted line and the actual points. The model in two dimensions has the following form: Let w = slope, x = predictor variable, y= target variable, b = y-intercept

$$y = wx + b \tag{12}$$

Of course, this idea can be extended to higher dimensions, this can be interpreted as fitting a hyperplane to the data. Formally, say we have N features One can write pth observation in the data set as:

$$x_p = \begin{bmatrix} x_{1,p} \\ x_{2,p} \\ . \\ . \\ . \\ x_{N,p} \end{bmatrix} \tag{13}$$

The slope can also be written as a column vector

$$w = \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_N \end{bmatrix} \tag{14}$$

The final model looks like

$$y_p \approx b + x_p^T w \tag{15}$$

It's important to quantify the distance between the predicted model $(b + x_p^T w)$ and the actual target variables (y_p). The distance metric used will be the sum of squares function, this will be our loss function. The goal will be to minimize the following:

$$\min_{b,w} \sum_{p}^{P} (b + x_p^T w - y_p)^2 \tag{16}$$

2.4 Decision Trees

The next model employed revolves around the concept of a decision tree. The general idea is to start with a root node and input the data into the node. Split the data conditional to one of the features of the data set. Then, the root node is split into two internal nodes. This process repeats for the internal nodes until the leaf node is reached. The goal is to group the data into homogeneous groups. This ideally, builds a model that can receive a data point, process the data through the tree, and put the data point into a defined subgroup at the leaf node. One should define how to select the feature used for splitting at a given node.

2.4.1 Information Gain

This method uses the concept of entropy applied to data. Entropy is a measure used to quantify the degree of disorder in the data. In other words, the unpredictability in our data. Given a data point, how accurate would a random guess be in determining the feature profile? Formally, entropy is calculated as:
f = feature , N = number of features

$$-\sum_{f \epsilon N}^{N} p(f) log(p(f)) \tag{17}$$

$p(f) = \frac{\textbf{Number of data points with feature f}}{Total number of points}$

Information gain is the entropy of the data set after a split occurs in the decision tree. After a split, the dataset becomes more homogenous, so the degree of disorder should decrease. A feature selected for splitting should aim to optimize this decrease in entropy. The following is the formal equation for selecting a splitting feature based on information gain: Say we select feature F to split F has subclasses denoted c

entropy$(D = data)$ **- WAVG(entropy**$(c \epsilon F)$**)**
= **entropy(D) -** $\sum_{c \epsilon F} \frac{|D_c|}{|D|} entropy(c)$
Where D_c = amount of data points that belong to subclass c
Information gain basically takes the total entropy and subtracts the entropy

given we select a feature. This quantifies how much entropy we remove given that a certain feature is selected for splitting. The higher the information gain, the better and the feature with the highest will be selected

2.4.2 Gini's index of impurity

The Gini index of impurity calculates the probability a point is misclassified given it was randomly selected. It tells us the same thing as entropy but in a different flavor. Formally, the Gini Index has the following equation (with the same notation as last question:

$$GINI = 1 - \sum_{f \epsilon N}^{N} (p(f))^2 \tag{18}$$

Select the feature with the least Gini Index value, this is because this implies there is a lower chance of misclassification given a random selection. Given the feature split, we get the most homogeneity out of all the other features.
Note: Scikit learn uses Gini's index by default because it is computationally less expensive, so the Gini's Index will be employed.
Now that the foundation is laid, consider modifications of the decision trees which will become the final models.

2.5 Random Forest

Random Forest is a variation of a bagged decision tree

2.5.1 Bagged Decision Trees

Specify N trees. Then, boot strap random sample (sample with replacement) from the data set and train each regression tree based on the separate bootstrapped samples. N Decision trees will be trained. Aggregate the trees by averaging the results to create a final decision tree.

2.5.2 Modification

To create a random forest, create a bagged decision tree but at every internal/decision node. Instead of taking into account all features, instead take a subset of the feature set. A popular choice is $\sqrt{N}$ The reason for this is to prevent overfitting. A regular decision tree create splits conditional on the intricacies of the training set that don't necessarily apply to new data. Hence why a subset of features is used instead, to prevent specific features well suited for the training set from being used repeatedly.

2.6 Boosted Tree

Another decision tree variation is the boosted tree. First, an overview of the algorithm. Train an initial decision tree. Then, train the next tree using the residuals of the previous tree, then repeat until a final model is created. The final model will be the amalgamation of weaker learners thus creating a boosted tree. Define an objective function for the tree:

$$\mathbf{obj(}\theta\mathbf{)} = \mathrm{L}(\theta) + \omega(\theta)$$

$\mathrm{L}(\theta)$ = Loss function
$\omega(\theta)$ = Regularization Term

2.6.1 Loss function

Use the typical Mean squared Error loss function

$$L(\theta) = \sum_i (y_i - \hat{y_i}) \tag{19}$$

y_i = actual observed values of target variable $\hat{y_i}$ = Values generated from model

2.6.2 Regularization

Since a boosted tree is susceptible to overfitting due to the repeated fitting of learners on the training data. The Regularization term puts a penalty on the complexity of the model and prevents overfitting. We exchange some variance in our model accuracy but ensure that the model isn't biased toward the training set.

2.7 Boosting the Tree

Define the objective function: Say we have N data points. $\hat{y_i^t}$ is the predicted value of y_i at step t (each t is each learning step or tree creation)

$$obj = \sum_{i=1}^{N} l(y_i, y_i^t) + \sum^{(} t)_{i=1} w(f_i) \tag{20}$$

Then:

$$\begin{aligned} y_i^0 \hat{=} 0 \\ \hat{y_i^1} = y_i^0 + \hat{f}_1(x_i) \\ . \\ . \\ . \\ \hat{y_i^t} = \sum_{k=1}^{t} f_k(x_i) = y_i^{t-1} \hat{+} f_t(x_i) \end{aligned}$$

Select $\hat{y_i^t}$ such that, the objective function is minimized. Notice how the loss function is updated recursively with every new tree/step in the algorithm.

3 K-Fold Cross Validation

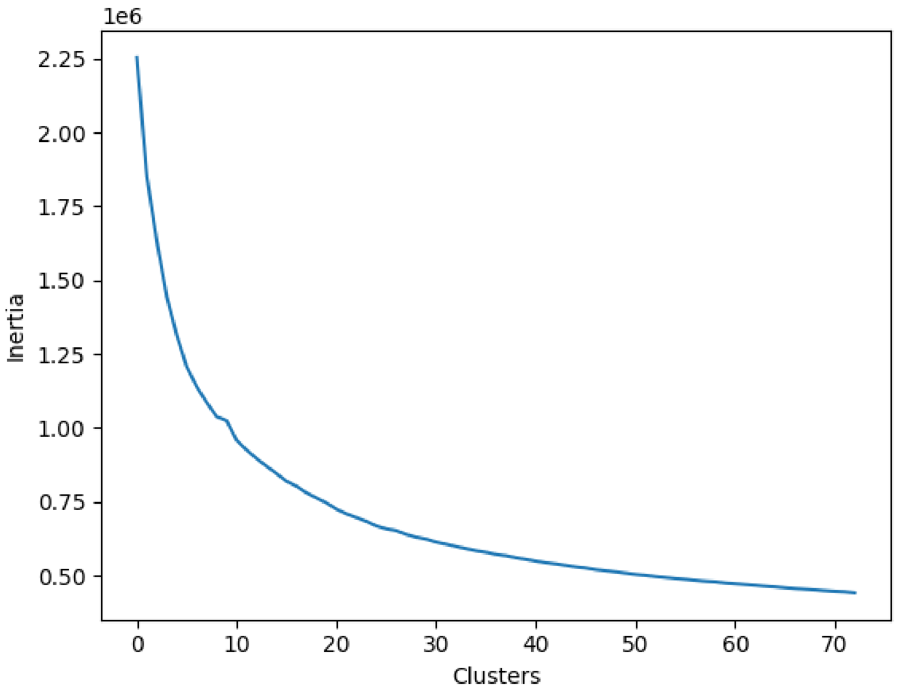
Finally, every model should be validated for consistency across datasets. One training set and one test set may not be a good representation of how well the model does. Here is where k-fold cross-validation will be useful. K-fold cross-validation bootstrap random samples k validation groups from the training dataset. In this study, there is a 80-20 split between training and test. Then, a model is trained on each of the validation datasets and applied to each of the test sets. The resulting root mean squared error out of all the models will be averaged. It may be advantageous to use metrics like median in case there are large deviations in the root mean squared error. This will ensure a robust evaluation of the model.

4 Results

4.1 Categorical Feature Importance

4.1.1 K-Means Clustering

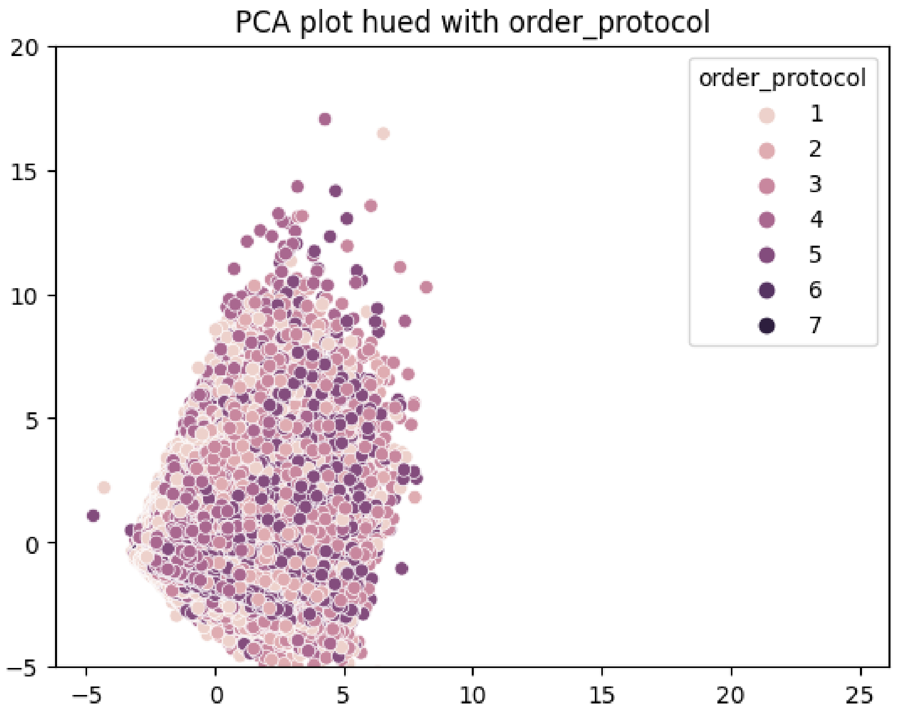
The following is the elbow plot produced from the data.

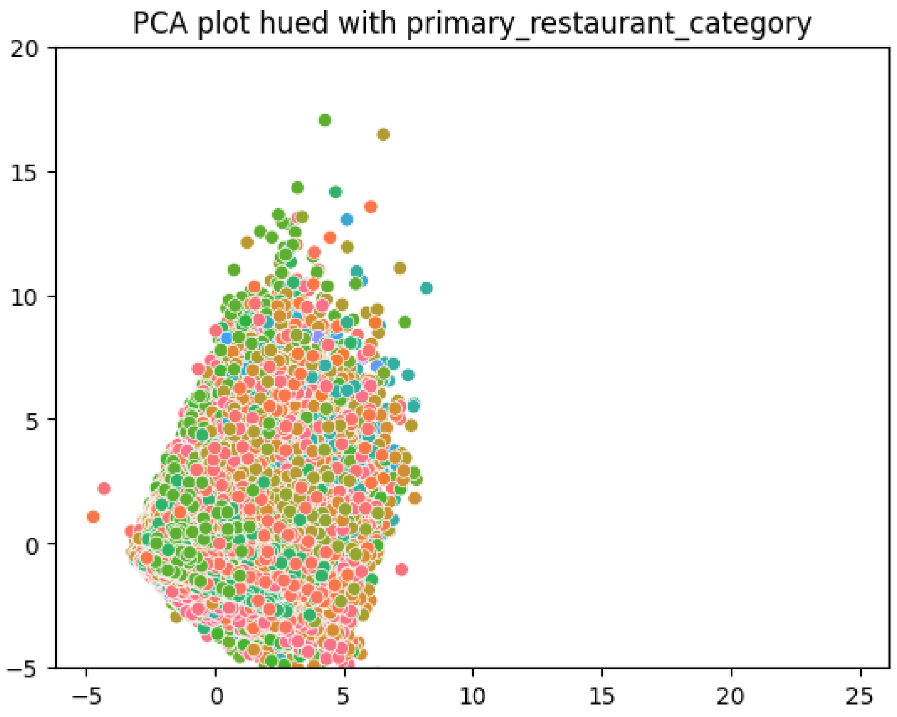


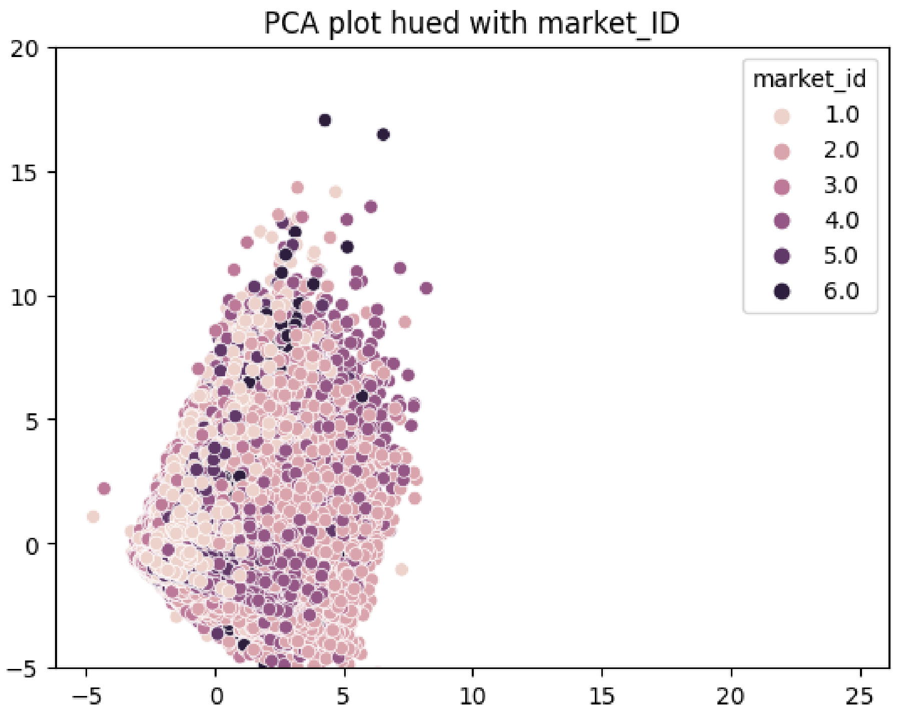
Usually, one should look for a dramatic drop-off point in the inertia given a number of clusters. There is indeed a drop-off, however, since the inertia is base 10^6 this implies at 74 classes, there is still an error of about 500,000. This implies that there is a high average minimum distance between points and the corresponding centroid. This implies the points don't really converge to homogeneous clusters. This means that the data may not need any categorical variables. One thing to note is most of the categorical variables are descriptions of the individual stores. Things like a restaurant's cuisine type, the specific market ID, etc. This result implies the actual restaurant one orders from has little impact on the time of arrival. To visualize this unimportance, visualize the categorical separation of the data using PCA

4.1.2 PCA

Below are some of the plots with different hues that represent a given categorical variable.




PCA plot hued with primary_restaurant_category
20
15
10
5
0
−5
−5
0
5
10
15
20
25



In all cases, there is no clear separation between the points. The Primary Restaurant Category and the order protocol plot seem random in their allocation of classes to points. The market ID class is interesting since there seems to be a separation of ID 1.0 and 3.0. However, embedded within these categories are the other market IDs that interfere with this separation. In general, the points are clearly clustered together and a clear separation that would be useful is not discernible.

4.1.3 Conclusion

As mentioned before, many of these categorical variables correspond to the characteristics of an individual restaurant. Based on the evidence above, it seems these attributes aren't as important. Thus, the categorical features will be deleted and more attention will be allocated to features representing processing time, contents of the order, and driving duration.

4.2 Collinearity

First, create a ranking of the correlation between variables

total onshift dashers	total busy dashers	0.942036
total onshift dashers	total outstanding orders	0.934746
total busy dashers	total outstanding orders	0.931509
total items	num distinct items	0.760825
subtotal	num distinct items	0.682125
total items	subtotal	0.555243
min item price	max item price	0.542474
subtotal	max item price	0.506313
num distinct items	min item price	0.450190
total items	min item price	0.393214

There is clearly a high correlation between total on-shift dashers, total busy dashers, and total outstanding orders. The correlation between on-shift and busy dashers is not surprising given that busy dasher is a subset of on-shift dashers. Thus, the larger the on-shift dashers are, the larger busy dashers can be. What's interesting is the total outstanding orders correlation. This implies that Doordash is able to keep up with demand and orders. The number of dashers available increases as the demand for dashers increases, with in it of itself an interesting observation into DoorDash's platform. Since a feature representing both has already been engineered (total free dashers), remove total on-shift and total busy dashers, leaving total outstanding orders. Reevaluate correlations:

total items	num distinct items	0.760825
subtotal	num distinct items	0.682125
total items	subtotal	0.555243
min item price	max item price	0.542474
subtotal	max item price	0.506313
num distinct items	min item price	0.450190
total items	min item price	0.393214
total outstanding orders	hour	0.353221

The correlation between the total items and the number of distinct items is not surprising. The more items a customer orders, the more likely there will be a high number of distinct items. Of course, the more items one orders, in general, results in a higher subtotal of the order. Feature engineer in the following way:

$$\textbf{Ratio Distinct} = \frac{\textbf{num of distinct items}}{\textbf{total items}}$$

Ratio distinct aims to account for both the number of distinct items and the total items. Leave in the subtotal. Reevaluate correlation:

min item price	max item price	0.542474
subtotal	max item price	0.506313
total outstanding orders	hour	0.353221
subtotal	ratio distinct	0.200403
max item price	hour	0.186932

The correlation between maximum and minimum is less so than in previous cases. However, this can be remedied easily by creating a new feature 'range price'

range price = max item price - min item price

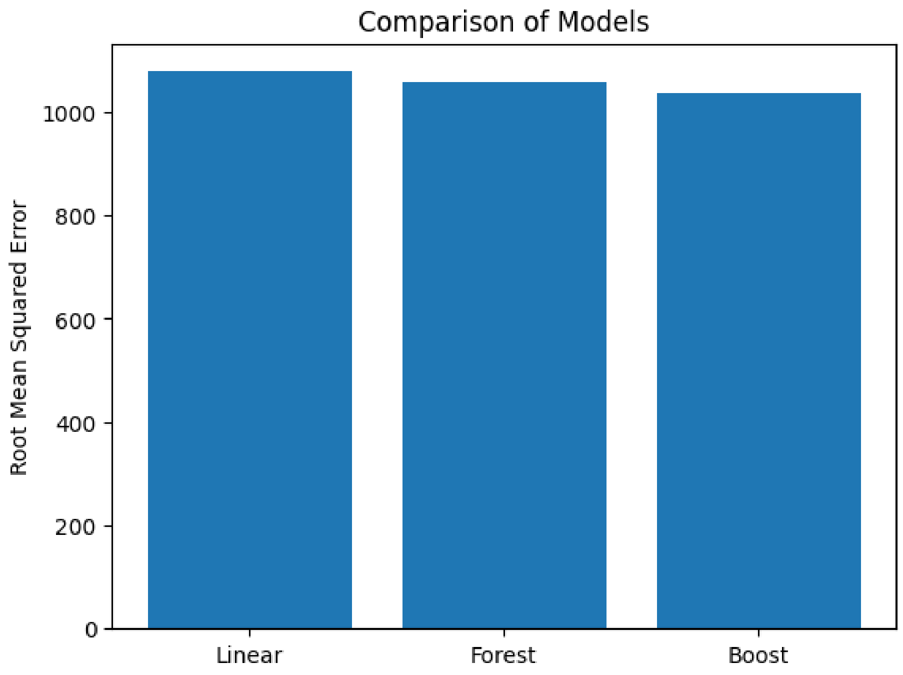
Drop max item price and min item price. Reevaluate correlation:

subtotal	range price	0.510971
total outstanding orders	hour	0.353221
subtotal	ratio distinct	0.200403
subtotal	hour	0.184285
total outstanding orders	estimated order place duration	0.164354

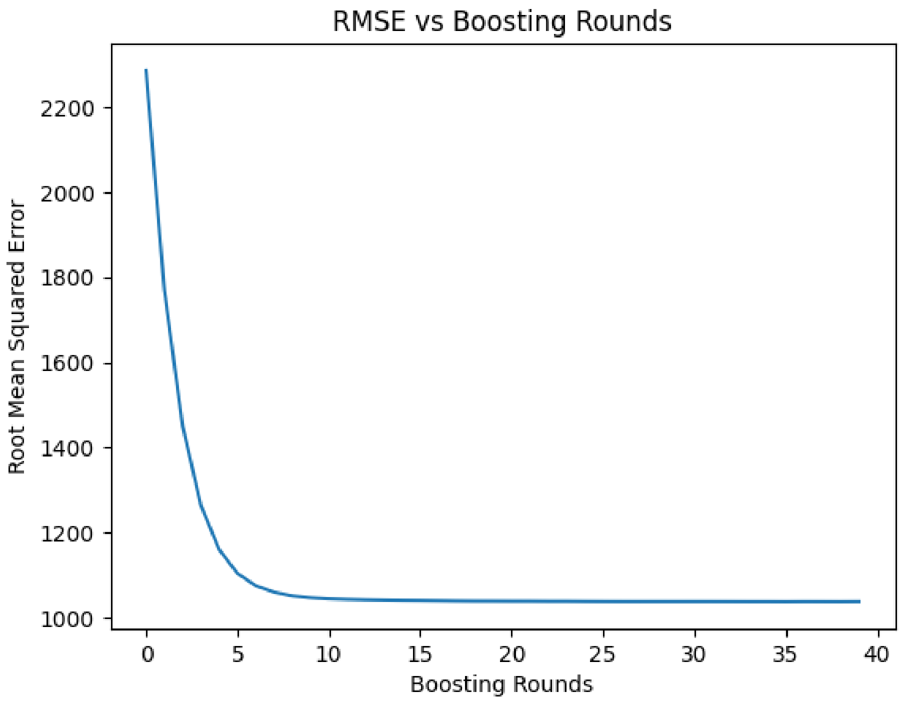
There is a borderline high correlation between subtotal and range price. However, the cut-off is usually 0.5, and it isn't significantly far off from that. Thus, have finalized the feature selection.

4.3 Model Fitting

Regression Model	Root Mean Squared Error
Linear Regression	1078.7229386131
Random Forest	1058.792527254
XG Boost	1036.741237

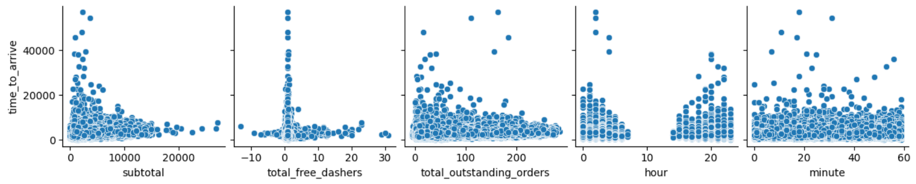


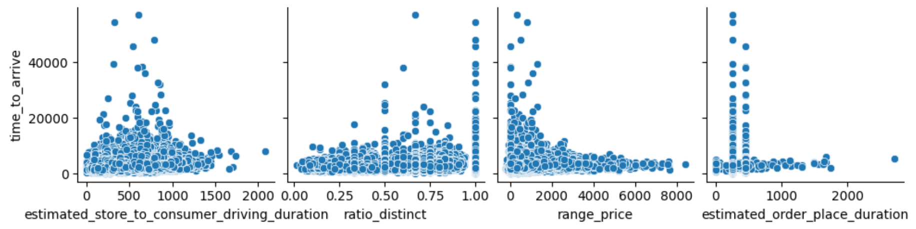
It seems the Boosted tree did the best out of all three, albeit not by much, observe how the Root Mean Squared Error decreases as the boosting rounds increase:



5 Discussion

The categorical features being unimportant was a little surprising. However, this does make sense since most people order out ’fast food’. The time to make these sorts of foods intuitively is not significantly different across restaurants. In fact, most of the time, the food is ready by the time the dasher gets to the establishment. Hence why, most of the features having to do with the actual restaurant were not important. It seems the time of arrival hinges more on the dashers or the physical transportation of the goods than anything else. Looking at the model performance, one can interpret the boosted tree to be 1036 seconds off from the original time. This is not an amazing estimate, that represents an error of about 18 minutes. However, given that the mean is 50 minutes, in context, it isn’t horrible. Taking a look at the correlation between the variables and time of arrival reveals some information on why this may be.





One can see from these plots there is no obvious correlation between the variables and the target. I found that most projects using this dataset got around the same Root Mean Squared Error (1000-1050), the data is just not sufficient. What would be helpful is specific descriptions of the location. For example, things like the weather or city may be of interest. One can also notice the performance of the models is not dramatically different. The boosted tree did have the best performance technically, in general, if the goal is to minimize the amount of error a boosted tree is well suited to that task. This is because of the cumulative learning the tree undertakes. It is prone to overfitting, but it also learns the training data very well since it continuously updates its loss function based on the residuals of the last boosting round. Something like Linear Regression may have done the worst because of this non-linear trend observed. The R^2 value was about 0.15. Hence why there will be a larger difference between the model prediction and the actual observations. The Random Forest did perform quite well, in fact, the decision tree algorithms in general performed better. This is due to the nature of the algorithm finding homogeneous splits in the data that aren't obvious when looking at a pairs plot. It doesn't necessarily rely on a linear relationship to get good results. As for why the Random Forest performed slightly worse than the boosted tree, this could be due to the fact the Random Forest is taking the aggregation of a bunch of other bootstrapped randomly sampled decision trees. So it isn't continuously learning from weaker learners, with data like this, it opens itself to higher variance estimates. However, given the difference between the two is about 20 seconds, it is nothing too significant. This problem is a classic case of 'bad data in, bad result out' to predict something as volatile as time to arrival of a door dash order, more factors need to be added and perhaps more data needs to be introduced.

6 References

Jeremy Watt, Reza Borhani, AND Aggelos K. Katsagellos. (2019). MACHINE LEARNING REFINED: foundations, algorithms, and applications. Cambridge Univ Press.

What is Random Forest? — IBM. (n.d.). What Is Random Forest? — IBM. https://www.ibm.com/topics/random-forest

Introduction to Boosted Trees — xgboost 1.5.1 documentation. (n.d.). Xgboost.readthedocs.io. https://xgboost.readthedocs.io/en/stable/tutorials/model.html

Data Source: https://platform.stratascratch.com/data-projects/delivery-duration-prediction