Predicting delivery times for products sold on eBay

Final project for STAT 5000

Ujas Shah

December 06, 2021

GitHub: https://github.com/UjasShah/StatsProject

Abstract

In this project, I work with data provided by eBay to forecast delivery times for products sold on the website. The process of forecasting is made tougher by the fact that this is real-world data and hence there are considerable data cleaning issues. Furthermore, the sheer differentiation of buyer's and seller's geography and a multitude of different shipping methods bring in a lot of differentiation in delivery times. After cleaning the data, I implement two models, a linear regression model, and a CatBoost model, CatBoost because of the success of gradient boosting methods in previous such applications. I find that linear regression provides an R2 that provides a loss value of .42 (counted on the loss function provided by eBay). While the CatBoost model provides a loss function of 0.39, showing that it performs slightly better than the linear regression.

Keywords: Machine Leaning, Linear Regression, CatBoost, Delivery Date Prediction.

Project Introduction

Acknowledgements

This project was made possible by eBay. As part of the eBay ML challenge (hosted on eval.ai), the company has shared a dataset related to purchases made on the website in the year 2018 and 2019, in order to find the most accurate way to predict delivery times.

The data is owned by eBay and I do not have the authority to share it. Hence, I will not be a public Github link here to load the data. However, if as part of this project, you need access to the data, I am happy to show it in person.

Problem Statement

Our aim in this project is to try to accurately predict delivery times for purchases made on eBay.

There are some challenges posed by this problem. Data quality is one of the primary ones. As transactions on the website are done between business and customers, and also between customers and and other customers, the quality of data put in by these sellers and buyers suffers. Furthermore, a large list of different shipping methods (25 different ways) and large geographic variability of shipments between logistically well and not-so-well connected areas also increases challenge in making an accurate prediction.

The dataset

The dataset that I am working with has to following features:

- b2c_c2c: string variable that shows whether the seller is a business or a customer
- seller_id: integer variable with a unique integer for each seller
- declared_handling_days: integer variable for the number of days it takes for the seller to deliver the package to the carrier service (which will finally deliver the package to the buyer)
- acceptance_scan_timestamp: string variable for the datetime stamp of when the carrier recieves the package
- shipment_method_id: integer variable for the type of shipping method
- carrier_min_estimate: integer variable for the carrier's estimate of minimum delivery time
- carrier_max_estimate: integer variable for the carrier's estimate of maximum delivery time
- item_zip: string variable for the zip code of item location
- buyer_zip: string variable for the buyer's zipcode
- category_id: integer variable for the type of product purchased
- item_price: float variable for the price of item (averaged if multiple items purchased
- quantity: integer variable for the number of items purchased
- payment_datetime: string variable for the date and time of the buyer's payment
- delivery_date: string variable of the delivery date
- weight: float variable for the weight of the purchased item
- weight_units: integer variable where 1 indicates weight in lbs and 2 indicates weight in kgs
- package_size: string variable describing the package size
- record number: integer variable with unique number to identify each purchase

Recommended Hardware and Software Requirements

Processor: Intel core i5 and above
RAM: 16GB
Disk Space: 10GB

Python: version 3.7.1

Literature Review

1. In the paper "Boosting Algorithms for Delivery Time Prediction in Transportation Logistcs", written by Jihed Khiari et. al., they focus on predicting the delivery dates in the logistics industry. To be more specific, they focus on predicting longer delivery times as there are many existing solutions for a shorter time frame. They have gone ahead with boosting algorithms like Light Gradient Boosting and CatBoost. They chose these algorithms as the accuracy and runtime efficiency was better than the baseline models such as linear regression models, bagging regressor, and random forest.
2. A thesis titled "Predicting the Last Mile: Route-Free Prediction of Parcel Delivery Time with Deep Learning for Smart-City Applications", written by Arthur Cruz De Araujo and presented at Queen's University, Kingston, focuses on improving the user experience by better modeling their anticipated delivery dates and to aid the last-mile postal logistics as a whole. They have implemented many convolutional neural networks and machine learning models. After their analysis, they found the ResNet model with 8 blocks to be the one with the best performance-complexity tradeoff.

3. The paper titled "Predicting Package Delivery Time for Motorcycles in Nairobi", written by Joseph A. Magiya in the year 2020 at Kenya College of Accountancy as a thesis in the field of Data Science. The research focuses on analyzing the delivery trends when products are delivered over a motorcycle. The analysis and feature engineering done helped us to better understand our data and find the relationships between data points. They have currently implemented a basic linear regression model with very low accuracy and plan to build on it from here.

Importing Required Libraries and Loading Data

In [3]:
```
import pandas as pd
import numpy as np
from uszipcode import SearchEngine
from datetime import datetime
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.linear_model import BayesianRidge
from catboost import CatBoostRegressor
import seaborn as sns
import matplotlib.pyplot as plt

data = pd.read_csv('/Users/ujas/Downloads/eBay_ML_Challenge_Dataset_2021/eBay_ML_Cha

data
```

Out[3]:

	b2c_c2c	seller_id	declared_handling_days	acceptance_scan_timestamp	shipment_method_
0	B2C	25454	3.0	2019-03-26 15:11:00.000-07:00	
1	C2C	6727381	2.0	2018-06-02 12:53:00.000-07:00	
2	B2C	18507	1.0	2019-01-07 16:22:00.000-05:00	
3	B2C	4677	1.0	2018-12-17 16:56:00.000-08:00	
4	B2C	4677	1.0	2018-07-27 16:48:00.000-07:00	
...	...	...	...	...	
14999995	B2C	80139	5.0	2018-05-04 15:36:00.000-05:00	
14999996	C2C	452486	2.0	2018-05-02 14:53:00.000-04:00	
14999997	B2C	43307	3.0	2018-06-06 15:21:00.000-07:00	
14999998	B2C	24235	1.0	2018-12-17 10:55:00.000-05:00	
14999999	C2C	535916	3.0	2018-05-03 12:56:00.000-04:00	

15000000 rows × 19 columns

In [4]:
```
#checking data types of different columns that can help us see if any columns need t
data.dtypes
```

Out[4]:
```
b2c_c2c                      object
seller_id                     int64
declared_handling_days      float64
acceptance_scan_timestamp    object
shipment_method_id            int64
shipping_fee                float64
carrier_min_estimate          int64
carrier_max_estimate          int64
item_zip                     object
buyer_zip                    object
category_id                   int64
item_price                  float64
quantity                      int64
payment_datetime             object
delivery_date                object
weight                        int64
weight_units                  int64
package_size                 object
record_number                 int64
dtype: object
```

In [5]:
```
# printing unique values, which if less, can be used to check for data that needs to
data.nunique()
```

Out[5]:
```
b2c_c2c                         2
seller_id                 1759305
declared_handling_days         11
acceptance_scan_timestamp 2245193
shipment_method_id             25
shipping_fee                 7044
carrier_min_estimate            6
carrier_max_estimate            6
item_zip                    50939
buyer_zip                   57273
category_id                    33
item_price                  41571
quantity                      147
payment_datetime          14090416
delivery_date                 767
weight                       1298
weight_units                    2
package_size                    7
record_number            15000000
dtype: int64
```

Data Cleaning

I now start with looking for issues in data and cleaning them. I've tried to be as throrough as possible as the modelling will only be as good as the data. The fact that we are dealing with a real world dataset will make the cleaning process quite lenghty.

In [6]:
```
#I start off with checking if there are any N/A values in the dataset
data.isna().sum()
```

Out[6]:
```
b2c_c2c                         0
seller_id                       0
declared_handling_days     702886
acceptance_scan_timestamp       0
shipment_method_id              0
```

```
shipping_fee                0
carrier_min_estimate        0
carrier_max_estimate        0
item_zip                    1
buyer_zip                   1
category_id                 0
item_price                  0
quantity                    0
payment_datetime            0
delivery_date               0
weight                      0
weight_units                0
package_size                0
record_number               0
dtype: int64
```

In [7]:
```
#There are no N/A values in b2c_c2c, now let's check if the column only has the valu
data['b2c_c2c'].unique()
```

Out[7]:
```
array(['B2C', 'C2C'], dtype=object)
```

In [8]:
```
#I now onverting b2c_c2c to a dummy variable, to make it usable in the model later o

data['b2c_c2c'] = data['b2c_c2c'].replace({'B2C':0, 'C2C':1})
```

In [9]:
```
#Seller ID does not have any N/As, but to ensure there are no issues with the data,
sum(data['seller_id']<0)
```

Out[9]:
```
0
```

In [12]:
```
#Declared handling days has a considerable amount of null values, I will try to esti
#For now we check for negative values
print(sum(data['declared_handling_days']<0))

#I also check for unique values to ensure that none of the values are problematic
print(data['declared_handling_days'].unique())
```

```
0
[ 3.  2.  1.  5.  0. 10. nan  4. 30. 15. 20. 40.]
```

In [14]:
```
#Shipment method id has no N/A values, but I look into the data further to ensure it
data.shipment_method_id.value_counts()
#It does have 0 values, but that just seems to be prominent shipping method and not
```

Out[14]:
```
0     9341444
1     2955149
2      856653
3      780997
4      297472
5      253508
6      176007
7      137404
8      125093
9       39639
10      27081
11       5906
13       1545
12       1077
14        447
15        431
16         48
```

```
19         21
17         20
18         17
21         16
20         13
22          6
24          4
26          2
Name: shipment_method_id, dtype: int64
```

In [17]:
```
#For shipping fee does not have null values, but we check for negative values
print(sum(data['shipping_fee']<0))

#As negative values in shipping fee doesn't make sense, we assume it to be a data is
for i in list(data[(data['shipping_fee']<0)]['record_number']):
    data['shipping_fee'][i-1]=None
```

```
0
```

In [18]:
```
#Looking at carrier min estimate, we have no N/A values. I again check for negative
print(sum(data['carrier_min_estimate']<0))
print(data['carrier_min_estimate'].unique())
data["carrier_min_estimate"] = data["carrier_min_estimate"].replace({-1:None})
```

```
0
```

In [19]:
```
#Similarly for carrier max estimate.
print(sum(data['carrier_max_estimate']<0))
print(data['carrier_max_estimate'].unique())
data["carrier_max_estimate"] = data["carrier_max_estimate"].replace({-1:None})
```

```
1095
[ 5  8  9  1 25 -1]
```

In [20]:
```
#Now checking item zip. We see that there is a null value. We will need to remove th

#Finding the NaN value
for x in range(len(data['item_zip'])):
    try:
        x = data['item_zip'][x][0]
    except:
        to_drop=x

#Removing the NaN value
data.drop(to_drop, inplace=True)

#Resetting the Index
data.reset_index(drop=True, inplace=True)
```

In [21]:
```
#I also see that there are some 9 digit zip codes so we slice them to ensure future
data['item_zip'] = data['item_zip'].str.slice(0, 5)
```

In [22]:
```
#Similarly for buyer zip
#Finding the Nan value
for x in range(len(data['buyer_zip'])):
    try:
        x = data['buyer_zip'][x][0]
    except:
        to_drop = x
```

```
#Removing the NaN value
data.drop(to_drop, inplace=True)

#Resetting the Index
data.reset_index(drop=True, inplace=True)

#Slicing the zip to first five digits
data['buyer_zip'] = data['buyer_zip'].str.slice(0, 5)
```

In [23]:

```
#Category id does not have any null values. I check for negative values and also che
print(sum(data['category_id']<0))
print(data.category_id.value_counts())
#There are quite a bit 0 values, but from the distribution of values it seems like 0
```

```
0
0     2544488
1     1293227
2     1281052
3     1233523
4      931637
5      787817
6      772039
7      720987
8      569544
10     543662
9      538951
11     445998
12     434892
13     429443
14     366025
15     320721
16     303283
17     257026
19     239080
18     215859
20     125426
22     102039
21      97935
23      94685
24      69860
25      65579
26      64628
27      39559
28      38358
29      32863
30      28701
31       9216
32       1895
Name: category_id, dtype: int64
```

In [25]:

```
#For item price, we check for negative number.
print(sum(data['item_price']==0))
print(sum(data['item_price']<0))

#Two values are equal to zero, but we let them be as its probable that two products
```

```
2
0
```

In [26]:

```
#I now check the column quantity
print(sum(data['quantity']<=0))
```

```
0
```

In [27]:
```
#Now I'll check if any of the weights are negative values. There are 0 values in wei
print(sum(data['weight']==0))
print(sum(data['weight']<0))
```

```
4792626
0
```

In [28]:
```
#Weights are in lbs and kgs, so here we create a new column to convert them to one u
def converting_weights(row):
    return [row["weight"] if row["weight_units"]==2 else (row["weight"]*0.45359237)]

data["weight_kg"] = data.apply(converting_weights, axis=1)

#I now replace 0s to Nones
data["weight_kg"] = data["weight_kg"].replace({0:None})
```

In [29]:
```
#In package size we check for unique values to see if all the categories make sense.
print(data['package_size'].value_counts())
data["package_size"] = data["package_size"].replace({'NONE':None,'PACKAGE_THICK_ENVE
```

```
PACKAGE_THICK_ENVELOPE    12652643
NONE                       1055227
LETTER                      912894
LARGE_ENVELOPE              209218
LARGE_PACKAGE               170014
EXTRA_LARGE_PACKAGE              1
VERY_LARGE_PACKAGE               1
Name: package_size, dtype: int64
```

In [30]:
```
#We now create a carrier average estimate out of carrier min and max estimates, as w
data = data.assign(carrier_average_estimate = lambda x: (x.carrier_min_estimate + x.
```

I do not think item and buyer zip codes themselves would be an important value to enter into the model. Calculating the distance variable from it should be much better able to predict delivery times. For this I try to find distance based on the zip codes. I decide to go ahead with the 'as the crow flies' distance because we are not aware of the exact method of how the purchase will be transferred between them (by road, air, rail or water).

I use the uszipcode package for this. However, while using the package I find that it is extremely slow to search for 30 million zip codes (item and buyer) in the package database for their lat and long. Hence, I create a dictionary for all the zipcodes that I will need and store their lat and long. A dictionary because the time complexity of searching in a dictionary is O(1). This would considerably speed up the process of searching the lat and long for 30 million zip codes.

In [31]:
```
#Creating the dictionary of zipcodes and its lat and long
search = SearchEngine(simple_zipcode=True)

zipcodes_needed = set()

for i in data['item_zip']:
    zipcodes_needed.add(i)

for i in data['buyer_zip']:
    zipcodes_needed.add(i)

zipcodes_needed_list = list(zipcodes_needed)

zipcodes_dict = {}
```

```
for i in zipcodes_needed_list:
    zip1 = search.by_zipcode(i)
    zipcodes_dict[i]=[zip1.lat, zip1.lng]
```

In [32]:

```
#Searching the dictionary for item and buyer zip to get lat and long and storing the
def itemlatlong(series):
    latlong = zipcodes_dict[series['item_zip']]
    return latlong

data['item_coord'] = data.apply(itemlatlong, axis = 1)

def buyerlatlong(series):
    latlong = zipcodes_dict[series['buyer_zip']]
    return latlong

data['buyer_coord'] = data.apply(buyerlatlong, axis = 1)

data[['item_lat','item_long']] = pd.DataFrame(data.item_coord.tolist(), index= data.
data[['buyer_lat','buyer_long']] = pd.DataFrame(data.buyer_coord.tolist(), index= da
```

In [33]:

```
#Finally I use the haversine formula (which determines the great circle distance bet
def haversine_np(lon1, lat1, lon2, lat2):

    lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = np.sin(dlat/2.0)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2.0)**2

    c = 2 * np.arcsin(np.sqrt(a))
    km = 6367 * c
    return km

#Credit: https://stackoverflow.com/questions/29545704/fast-haversine-approximation-p

data['distance']=haversine_np(data['item_long'], data['item_lat'], data['buyer_long'
```

In [34]:

```
#Now I work with the three datetime columns to convert the model to a useful format
def date(string):
    return string.split()[0]

def convert_data(string):
    return datetime.strptime(str(string), '%Y-%m-%d').date()

def number_of_days(string):
    return (string['delivery_date_modified'] - string['payment_date']).days
```

In [35]:

```
#Now I use the above defined functions to convert the strings into a useable format.
data['payment_date'] = data['payment_datetime'].apply(date)
data['payment_date'] = data['payment_date'].apply(convert_data)
data['delivery_date_modified'] = data['delivery_date'].apply(convert_data)
data['no_of_days_after_payment'] = data.apply(number_of_days, axis = 1)
```

In [36]:

```
#After converting the data into a useable format, I drop rows wherein the delivery d
to_drop = data[data['no_of_days_after_payment']<0].index
```

```
data.drop(index = to_drop, inplace = True)
```

In [38]:
```
#Declared handling days has a number of null values. I create another column called
def number_of_handling_days(string):
    return (string['acceptance_date'] - string['payment_date']).days

data['acceptance_date'] = data['acceptance_scan_timestamp'].apply(date)
data['acceptance_date'] = data['acceptance_date'].apply(convert_data)
data['handling_days'] = data.apply(number_of_handling_days, axis = 1)
```

I am done with the data cleaning for the purposes of this project. I still have more ideas on how to clean the data further and better which I'll undertake in my final submission to eBay.

For now, I'll copy the data onto model_data to feed it to the models and drop null values. I decide to drop null values instead of estimating all null values as I believe we have enough data points (nearly 10 million).

In [39]:
```
#Creating model_data

model_data = data.dropna()
```

Loss Function

eBay has defined a loss function that I have to try to minimize. The loss function penalizes early deliveries less than late deliveries. This is because deliveries later than the estimated date result in a worse experience for the customer than deliveries earlier than the estimated day. The penalty for early deliveries is 0.4, while the same for late deliveries is 0.6. eBay has also provided a baseline loss score, which is 0.75.

In [40]:
```
##Loss function

def evaluate_loss(preds, actual):
    early_loss, late_loss = 0,0
    for i in range(len(preds)):
        if preds[i] < actual[i]:
            #early shipment
            early_loss += actual[i] - preds[i]
        elif preds[i] > actual[i]:
            #late shipment
            late_loss += preds[i] - actual[i]
    loss = (1/len(preds)) * (0.4 * (early_loss) + 0.6 * (late_loss))
    return loss
```

Dividing Data into Train and Test

Below I create my training and testing datasets in a 80:20 ratio.

In [44]:
```
x = model_data[['b2c_c2c', 'declared_handling_days', 'carrier_average_estimate', 'di
y = model_data['no_of_days_after_payment']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, shuffle =
```

Modelling

Multivariate Linear regression

This is the first model that I will try to train, to get an idea about what kind of baseline accuracy we can get. The linear regression tries to plot a linear equation with all the data points provided such that the error between the line and the data points is minimized. And in this way tries to predict the relationship between the predictors and the predicted variables.

In [47]:
```
model = LinearRegression()
model.fit(np.array(x_train), y_train)
predictions = model.predict(np.array(x_test))
predictions = [np.floor(x) for x in predictions]
print("RSquared: " + str(r2_score(y_test, predictions)))
print('Loss: ' + str(evaluate_loss(predictions,np.array(y_test))))
```

```
RSquared: 0.6742228623041264
Loss: 0.42817993414932787
```

CatBoost

After getting an idea about the baseline set by linear regression, I now train a CatBoost model, as this has shown to be among the best models in previous such applications. A CatBoost model is a type of gradient boosting model.

In [51]:
```
After getting an idea about the baseline set by linear regression, I now train a Cat
#Converting package size to int from float
cat_x= np.column_stack((model_data['b2c_c2c'],model_data['seller_id'], model_data['d

cat_y = np.array(model_data['no_of_days_after_payment'])

cat_x_train, cat_x_test, cat_y_train, cat_y_test = train_test_split(cat_x, cat_y, tr

cat_features = [0, 3, 7, 8, 9, 13]
model = CatBoostRegressor(iterations=100, cat_features = cat_features, early_stoppin

model.fit(cat_x_train, cat_y_train)
pred_Cat = model.predict(cat_x_test)

pred_Cat = [np.floor(x) for x in pred_Cat]
print('Loss: '+ str(evaluate_loss(pred_Cat,np.array(cat_y_test))))
```

```
0:      learn: 2.8726688        total: 3.31s    remaining: 5m 27s
1:      learn: 2.6955728        total: 5.18s    remaining: 4m 13s
2:      learn: 2.5380301        total: 6.53s    remaining: 3m 31s
3:      learn: 2.3995318        total: 8.03s    remaining: 3m 12s
4:      learn: 2.2805495        total: 9.31s    remaining: 2m 56s
5:      learn: 2.1757138        total: 10.5s    remaining: 2m 44s
6:      learn: 2.0872282        total: 11.7s    remaining: 2m 34s
7:      learn: 2.0094400        total: 12.8s    remaining: 2m 27s
8:      learn: 1.9436172        total: 14.2s    remaining: 2m 23s
9:      learn: 1.8864139        total: 15.8s    remaining: 2m 22s
10:     learn: 1.8381148        total: 17.5s    remaining: 2m 21s
11:     learn: 1.7974389        total: 18.8s    remaining: 2m 17s
12:     learn: 1.7628632        total: 19.9s    remaining: 2m 13s
13:     learn: 1.7341645        total: 21.6s    remaining: 2m 12s
14:     learn: 1.7093648        total: 22.9s    remaining: 2m 9s
15:     learn: 1.6878545        total: 24.4s    remaining: 2m 8s
16:     learn: 1.6700113        total: 25.7s    remaining: 2m 5s
17:     learn: 1.6546003        total: 26.9s    remaining: 2m 2s
18:     learn: 1.6414752        total: 28.7s    remaining: 2m 2s
19:     learn: 1.6299673        total: 30.3s    remaining: 2m 1s
20:     learn: 1.6200768        total: 31.8s    remaining: 1m 59s
21:     learn: 1.6117766        total: 34.1s    remaining: 2m 1s
22:     learn: 1.6045210        total: 35.7s    remaining: 1m 59s
```

```
23:     learn: 1.5983256        total: 37.1s    remaining: 1m 57s
24:     learn: 1.5927941        total: 38.8s    remaining: 1m 56s
25:     learn: 1.5876634        total: 40.3s    remaining: 1m 54s
26:     learn: 1.5835405        total: 41.5s    remaining: 1m 52s
27:     learn: 1.5799412        total: 42.7s    remaining: 1m 49s
28:     learn: 1.5763984        total: 43.8s    remaining: 1m 47s
29:     learn: 1.5732678        total: 45s      remaining: 1m 44s
30:     learn: 1.5706527        total: 46.1s    remaining: 1m 42s
31:     learn: 1.5683882        total: 48.9s    remaining: 1m 43s
32:     learn: 1.5662978        total: 50.3s    remaining: 1m 42s
33:     learn: 1.5633069        total: 51.9s    remaining: 1m 40s
34:     learn: 1.5608737        total: 53.4s    remaining: 1m 39s
35:     learn: 1.5585859        total: 54.9s    remaining: 1m 37s
36:     learn: 1.5567459        total: 56.2s    remaining: 1m 35s
37:     learn: 1.5543424        total: 57.7s    remaining: 1m 34s
38:     learn: 1.5524299        total: 59.1s    remaining: 1m 32s
39:     learn: 1.5505065        total: 1m       remaining: 1m 30s
40:     learn: 1.5487621        total: 1m 1s    remaining: 1m 28s
41:     learn: 1.5470418        total: 1m 2s    remaining: 1m 26s
42:     learn: 1.5457640        total: 1m 4s    remaining: 1m 25s
43:     learn: 1.5445092        total: 1m 5s    remaining: 1m 23s
44:     learn: 1.5432524        total: 1m 7s    remaining: 1m 22s
45:     learn: 1.5419993        total: 1m 8s    remaining: 1m 20s
46:     learn: 1.5411370        total: 1m 9s    remaining: 1m 18s
47:     learn: 1.5401564        total: 1m 10s   remaining: 1m 16s
48:     learn: 1.5392343        total: 1m 11s   remaining: 1m 14s
49:     learn: 1.5385159        total: 1m 13s   remaining: 1m 13s
50:     learn: 1.5378360        total: 1m 14s   remaining: 1m 11s
51:     learn: 1.5372147        total: 1m 15s   remaining: 1m 9s
52:     learn: 1.5365657        total: 1m 16s   remaining: 1m 7s
53:     learn: 1.5359591        total: 1m 17s   remaining: 1m 6s
54:     learn: 1.5354214        total: 1m 18s   remaining: 1m 4s
55:     learn: 1.5346793        total: 1m 20s   remaining: 1m 2s
56:     learn: 1.5336505        total: 1m 21s   remaining: 1m 1s
57:     learn: 1.5331233        total: 1m 22s   remaining: 59.8s
58:     learn: 1.5326468        total: 1m 23s   remaining: 58.2s
59:     learn: 1.5320565        total: 1m 24s   remaining: 56.6s
60:     learn: 1.5316845        total: 1m 26s   remaining: 55.1s
61:     learn: 1.5310724        total: 1m 27s   remaining: 53.5s
62:     learn: 1.5305555        total: 1m 28s   remaining: 52s
63:     learn: 1.5300715        total: 1m 29s   remaining: 50.4s
64:     learn: 1.5296026        total: 1m 30s   remaining: 48.9s
65:     learn: 1.5289092        total: 1m 31s   remaining: 47.3s
66:     learn: 1.5284661        total: 1m 33s   remaining: 45.8s
67:     learn: 1.5281518        total: 1m 34s   remaining: 44.3s
68:     learn: 1.5278247        total: 1m 35s   remaining: 42.8s
69:     learn: 1.5276148        total: 1m 36s   remaining: 41.4s
70:     learn: 1.5270850        total: 1m 37s   remaining: 39.9s
71:     learn: 1.5266929        total: 1m 38s   remaining: 38.4s
72:     learn: 1.5263781        total: 1m 39s   remaining: 37s
73:     learn: 1.5260017        total: 1m 41s   remaining: 35.5s
74:     learn: 1.5256389        total: 1m 42s   remaining: 34.1s
75:     learn: 1.5251253        total: 1m 43s   remaining: 32.8s
76:     learn: 1.5248380        total: 1m 45s   remaining: 31.4s
77:     learn: 1.5244788        total: 1m 46s   remaining: 30.1s
78:     learn: 1.5240483        total: 1m 47s   remaining: 28.7s
79:     learn: 1.5237579        total: 1m 49s   remaining: 27.3s
80:     learn: 1.5236411        total: 1m 50s   remaining: 25.9s
81:     learn: 1.5230413        total: 1m 51s   remaining: 24.5s
82:     learn: 1.5227168        total: 1m 52s   remaining: 23.1s
83:     learn: 1.5222556        total: 1m 54s   remaining: 21.9s
84:     learn: 1.5219290        total: 1m 56s   remaining: 20.5s
85:     learn: 1.5216679        total: 1m 57s   remaining: 19.2s
86:     learn: 1.5215520        total: 1m 59s   remaining: 17.8s
87:     learn: 1.5213988        total: 2m       remaining: 16.4s
88:     learn: 1.5211025        total: 2m 1s    remaining: 15s
89:     learn: 1.5208562        total: 2m 3s    remaining: 13.7s
90:     learn: 1.5204886        total: 2m 4s    remaining: 12.3s
91:     learn: 1.5202785        total: 2m 5s    remaining: 10.9s
```

```
92:     learn: 1.5199795        total: 2m 6s    remaining: 9.53s
93:     learn: 1.5197535        total: 2m 7s    remaining: 8.17s
94:     learn: 1.5193423        total: 2m 9s    remaining: 6.79s
95:     learn: 1.5190495        total: 2m 10s   remaining: 5.43s
96:     learn: 1.5187792        total: 2m 11s   remaining: 4.06s
97:     learn: 1.5185822        total: 2m 12s   remaining: 2.71s
98:     learn: 1.5184516        total: 2m 13s   remaining: 1.35s
99:     learn: 1.5181838        total: 2m 15s   remaining: 0us
Loss: 0.3932153689065414
```

Conclusion

After data cleaning, I ran two models, they gave loss scores of 0.42 and 0.39 for linear regression and CatBoost respectively. As described by other papers too, Gradient boosting models like the CatBoost are among the best performing models. with less loss than the linear regression.

References

1. "Boosting Algorithms for Delivery Time Prediction in Transportation Logistics", International Conference on Data Mining Workshops, 2020, DOI: 10.1109/ICDMW51313.2020.00043
2. "Predicting Shipping Time with Machine Learning", Presented at Massachusets Institute of Technology, 2012.
3. "Predicting the Last Mile: Route-Free Prediction of Parcel Delivery Time with Deep Learning for Smart-City Applications", Queen's University, Canada, 2020.
4. eBay shipping predictions: https://milliemince.github.io/eBay-shipping-predictions/
5. "A Machine Learning Approach to Delivery Time Estimation for Industrial Equipment", Purdue University.
6. "Predicting Package Delivery Time For Motorcycles In Nairobi", Kenya College of Accountancy, 2020, DOI:10.13140/RG.2.2.27105.94567
7. "DeepETA: A Spatial-Temporal Sequential Neural Network Model for Estimating Time of Arrival in Package Delivery System", Assosciation for the Advancement of Artificial Intelligence, 2019.