Food Delivery time prediction

Table of Contents

Objectives of Food Delivery Time Prediction

- Make an accurate estimate of when the food will arrive, thus increasing customer con dence.
- Plan delivery routes and driver schedules more ef ciently by predicting how many orders will arrive so that delivery providers can use their resources better.
- Make deliveries faster by looking at past delivery data and determining the attributes that affect them.
- Grow business because of buyer satisfaction with the speed of delivery.

Based on these goals, we will use the LSTM Neural Network to develop a model that can estimate the delivery time of orders accurately based on the age of the delivery partner, the partner's rating, and the distance between the restaurant and the buyer's place. This article will guide you on predicting food delivery time using LSTM. Now, let's make the prediction through the steps in the article.

Step 1: Import Library

```
import pandas as pd import numpy as np import
plotly.express as px from sklearn.model_selection
import train_test_split from keras.models import
Sequential from keras.layers import Dense, LSTM
```

Pandas and NumPy libraries are used together for data analysis. NumPy provides fast mathematical functions for multidimensional arrays, while Pandas makes it easier to analyze and manipulate data with more complex data structures like DataFrame and Series. Meanwhile, the Plotly Express library makes it easy for users to create interactive visualizations in Python. It can use minimal code to create various charts, such as scatter plots, line charts, bar charts, and maps. The Sequential class is a type of model in Keras that allows users to create a neural network by adding layers to it in sequential order. Then, Dense and LSTM are to create layers in the Keras model and also customize their con gurations.

Step 2: Read the Data

The availability of data is crucial to any data analysis task. It is essential to have a dataset that contains all the required features and variables for the particular task at hand. And for this particular case, the appropriate dataset is on my github. The dataset given here is a cleaned version of the original dataset submitted by Gaurav Malik on Kaggle.

```
#reading dataset url =
'https://raw.githubusercontent.com/ataislucky/ data
= pd.read_csv(url)
data.sample(5)
```

ID	Delivery_person_ID	Delivery_person_Age	Delivery_person_Ratings	Restaurant_latitude	Restaurant_longitude	Delivery_location_latitude
9D04	MYSRES11DEL02	28	4.5	12.323225	76.630028	12.343225
3D45	SURRES19DEL02	26	4.7	21.149669	72.772629	21.169669
72A1	MUMRES08DEL01	36	4.8	19.065838	72.832658	19.075838
3C06	MUMRES14DEL02	27	4.5	19.181300	72.836191	19.261300
663F	JAPRES11DEL03	22	3.1	26.902940	75.793007	26.992940

Let's see detailed information about the dataset we use with the info() command.

```
RangeIndex: 45593 entries, 0 to 45592
Data columns (total 11 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   ID                           45593 non-null  object
 1   Delivery_person_ID           45593 non-null  object
 2   Delivery_person_Age          45593 non-null  int64
 3   Delivery_person_Ratings      45593 non-null  float64
 4   Restaurant_latitude          45593 non-null  float64
 5   Restaurant_longitude         45593 non-null  float64
 6   Delivery_location_latitude   45593 non-null  float64
 7   Delivery_location_longitude  45593 non-null  float64
 8   Type_of_order                45593 non-null  object
 9   Type_of_vehicle              45593 non-null  object
 10  Time_taken(min)              45593 non-null  int64
dtypes: float64(5), int64(2), object(4)
```

dataset overview

Checking a dataset's columns and null values is essential in any data analysis project. Let's do it.

```
data.isnull().sum()
```

```
ID                             0
Delivery_person_ID             0
Delivery_person_Age            0
Delivery_person_Ratings        0
Restaurant_latitude            0
Restaurant_longitude           0
Delivery_location_latitude     0
Delivery_location_longitude    0
Type_of_order                  0
Type_of_vehicle                0
Time_taken(min)                0
dtype: int64
```

The dataset is complete with no null values, so let's proceed!

Step 3: Haversine Formula

The Haversine formula is used to nd the distance between two geographical locations. The formula refers to this Wikipedia page as follows:

The Haversine formula:

a = sin²(Δlat/2) + cos(lat1) * cos(lat2) * sin²(Δlon/2)
c = 2 * atan2(√a, √(1−a))
d = R * c

where:

- lat1 and lat2 are the latitudes of the two points
- Δlat means the difference between the latitudes
- Δlon means the difference between the longitudes
- R means the radius of the sphere
- d means the distance between the two points in the same units as R

It takes the latitude and longitude of two points and converts the angles to radians to perform the necessary calculations. We use this formula because the dataset doesn't provide the distance between the restaurant and the delivery location. There are only latitude and longitude. So, let's calculate it and then create a distance column in the dataset.

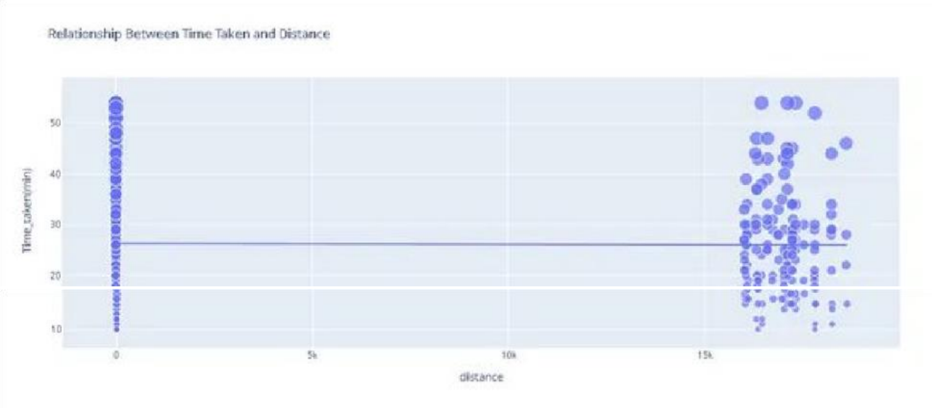
```
R = 6371  ##The earth's radius (in km)

def deg_to_rad(degrees):
    return degrees * (np.pi/180)

## The haversine formula def
distcalculate(lat1, lon1, lat2, lon2):
    d_lat = deg_to_rad(lat2-lat1)
    d_lon = deg_to_rad(lon2-lon1)
    a1 = np.sin(d_lat/2)**2 +
np.cos(deg_to_rad(lat1
    a2 = np.cos(deg_to_rad(lat2)) *
np.sin(d_lon/2)*
    a = a1 * a2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    return R * c

# Create distance column & calculate the distance
data['distance'] = np.nan
```

```
for i in range(len(data)):
  data.loc[i, 'distance'] =
distcalculate(data.loc[i

data.loc[i

data.loc[i

data.loc[i
```

The parameter “lat” means latitude, and “lon” means longitude. The deg_to_rad function is helpful for converting degrees to radians. At the same time, calculate the distance between two location points using the variables a1 and a2. The variable stores the result of multiplying a1 and a2, while the c variable stores the result of the Haversine formula calculation, which produces the distance between the two location points.

We have added a distance column to the dataset. Now, we will analyze the effect of distance and delivery time.

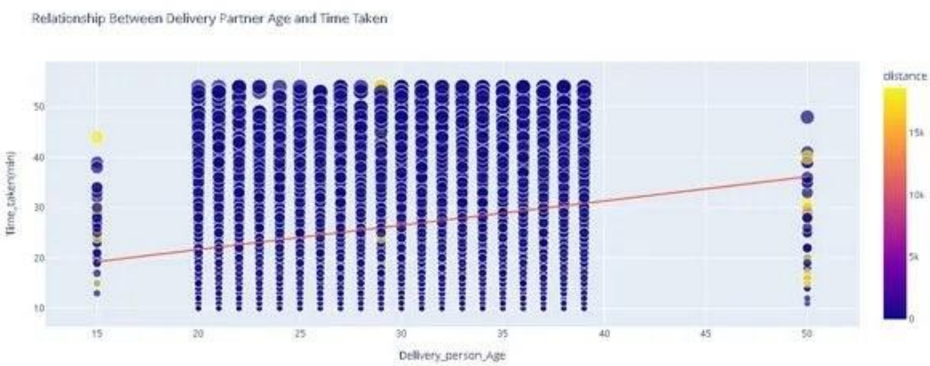
```
figure = px.scatter(data_frame = data,
x="distance",
                    y="Time_taken(min)",
                    size="Time_taken(min)",
                    trendline="ols",
                    title = "Relationship Between
Ti figure.show()
```



The graph shows that there is a consistent relationship between the time taken and the distance traveled for food delivery. This means that the majority of delivery partners deliver food within a range of 25–30 minutes, regardless of the distance.

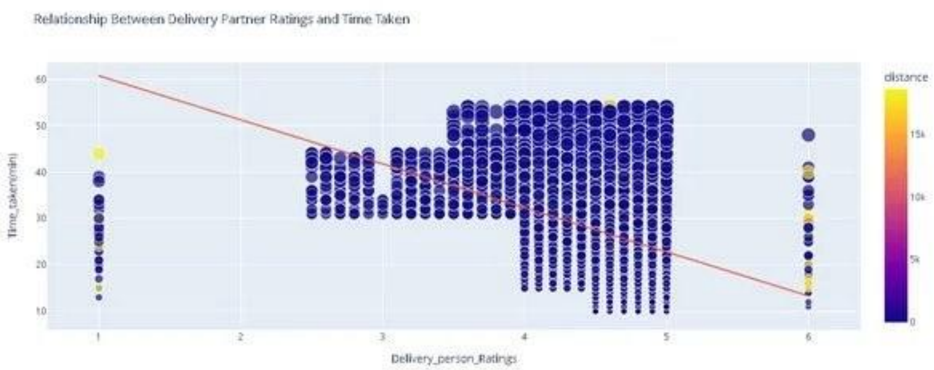
Next, we will explore whether the delivery partner's age affects delivery time or not.

```
figure = px.scatter(data_frame = data,
x="Delivery_person_Age",
y="Time_taken(min)",
                    size="Time_taken(min)",
                    color = "distance",
                    trendline="ols",
                    title = "Relationship Between
De figure.show()
```



The graph shows faster food delivery when partners are younger than their older counterparts. Now let's explore the correlation between delivery time and delivery partner ratings.

```
figure = px.scatter(data_frame = data,
                    x="Delivery_person_Ratings",
                    y="Time_taken(min)",
                    size="Time_taken(min)",
                    color = "distance",
                    trendline="ols",
                    title = "Relationship Between
De figure.show()
```



The graph shows an inverse linear relationship. The higher the rating partner, the faster the time needed to deliver food, and vice versa.

The next step will be to see whether the delivery partner’s vehicle affects the delivery time or not.

```
fig = px.box(data,
             x="Type_of_vehicle",
             y="Time_taken(min)",
             color="Type_of_order",
             title = "Relationship Between Type of
V fig.show()
```

The graph shows that the type of delivery partner’s vehicle and the type of food delivered do not signi cantly affect delivery time.

Through the analysis above, we can determine that the delivery partner’s age, the delivery partner’s rating, and the distance between the restaurant and the delivery location are the features that have the most signi cant impact on food delivery time.

Step 4: Build an LSTM Model and Make Predictions

Previously, we have determined three features that signi cantly affect the time taken, namely the delivery partner’s age, the delivery partner’s rating, and distance. So the three features will become independent variables (x), while the time taken will become the dependent variable (y).

```
x = np.array(data[["Delivery_person_Age",
                   "Delivery_person_Ratings",
                   "distance"]]) y =
np.array(data[["Time_taken(min)"]]) xtrain, xtest,
ytrain, ytest = train_test_split(x, y
test

rand
```

Now, we need to train an LSTM neural network to predict food delivery time. The aim is to create a precise model that uses features like distance, delivery partner age, and rating to estimate food delivery time. The trained model can then be used to predict new data points or unseen scenarios.

```
model = Sequential() model.add(LSTM(128,
return_sequences=True, input_sha model.add(LSTM(64,
return_sequences=False)) model.add(Dense(25))
model.add(Dense(1)) model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 3, 128)            66560

 lstm_1 (LSTM)               (None, 64)                49408

 dense (Dense)               (None, 25)                1625

 dense_1 (Dense)             (None, 1)                 26

=================================================================
Total params: 117,619
Trainable params: 117,619
Non-trainable params: 0
_________________________________________________________________
```

The code block above explains:

The rst line starts building the model architecture by creating an instance of the Sequential class. The following three lines de ne the layers of the model. The rst layer is an LSTM layer with 128 units, which returns sequences and takes input for shape (xtrain.shape[1], 1). Here, xtrain is the input training data, and shape[1] represents the number of features in the input data. The return_sequences parameter is set to True because there will be more layers after this one. The second layer is also an LSTM layer, but with 64 units and return_sequences set to False, indicating that this is the last layer. The third line adds a dense layer with 25 units, which reduces the output of the LSTM layers to a more manageable size. Finally, the fourth line adds a dense layer with one unit, which is the output layer of the model.

Now let's train the previously created model.

```
model.compile(optimizer='adam',
loss='mean_squared_e model.fit(xtrain, ytrain,
batch_size=1, epochs=9)
```

```
Epoch 1/9
36474/36474 [==============================] - 194s 5ms/step - loss: 69.1111
Epoch 2/9
36474/36474 [==============================] - 170s 5ms/step - loss: 64.1346
Epoch 3/9
36474/36474 [==============================] - 169s 5ms/step - loss: 61.7873
Epoch 4/9
36474/36474 [==============================] - 169s 5ms/step - loss: 60.9130
Epoch 5/9
36474/36474 [==============================] - 168s 5ms/step - loss: 59.9716
Epoch 6/9
36474/36474 [==============================] - 170s 5ms/step - loss: 59.3918
Epoch 7/9
36474/36474 [==============================] - 168s 5ms/step - loss: 59.6685
Epoch 8/9
36474/36474 [==============================] - 169s 5ms/step - loss: 59.3368
Epoch 9/9
36474/36474 [==============================] - 169s 5ms/step - loss: 59.0784
<keras.callbacks.History at 0x7f1a9c08b250>
```

The 'adam' parameter is a popular optimization algorithm for deep learning models, and the 'mean_squared_error' parameter is a common loss function used in regression problems. The parameter batch_size = 1 means that the model will update its weights after each sample is processed during training. The epochs parameter is set to 9, meaning the model will be trained on the entire dataset for nine iterations.

Finally, let’s test the model’s performance for predicting food delivery times given three input parameters (delivery partner age, delivery rating, and distance).

```
print("Food Delivery Time Prediction using
LSTM") a = int(input("Delivery Partner Age: "))
b = float(input("Previous Delivery Ratings: "))
c = int(input("Total Distance: "))

features = np.array([[a, b, c]])
print("Delivery Time Prediction in Minutes = ",
mode
```

```
Food Delivery Time Prediction using LSTM
Delivery Partner Age: 33
Previous Deliverie Ratings: 4.0
Total Distance: 7
1/1 [==============================] - 1s 814ms/step
Delivery Time Prediction in Minutes =  [[36.913715]]
```

The given result is a prediction of the delivery time for a hypothetical food delivery order based on the trained LSTM neural network model using the following input features:

- Delivery Partner’s Age: 33
- Previous Delivery Ratings: 4.0
- Total distance: 7

The output of the prediction is shown as “Delivery Time Prediction in Minutes = [[36.913715]],” which means that the model has estimated that the food delivery will take approximately 36.91 minutes to reach the destination.

Conclusion

This article starts by calculating the distance between the restaurant and the delivery location. Then, it analyzes previous delivery times for the same distance before predicting food delivery times in real-time using LSTM. Broadly speaking, in this post, we have discussed the following:

- How to calculate the distance using the haversine formula?
- How to nd the features that affect the food delivery time prediction?
- How to use LSTM neural network model to predict the food delivery time?