Food Delivery Time Prediction

Food Delivery services like Zomato and Swiggy need to show the accurate time it will take to deliver your order to keep transparency with their customers.To predict the food delivery time based on how much time the delivery partners took for the same distance in the past.

To predict the food delivery time in real-time, we need to calculate the distance between the food preparation point and the point of food consumption. After finding the distance between the restaurant and the delivery locations, we need to find relationships between the time taken by delivery partners to deliver the food in the past for the same distance.

In [1]:
```
#lets us import the necessary Python libraries and the dataset:
import pandas as pd
import numpy as np
import plotly.express as px

data = pd.read_csv("deliverytime.txt")
print(data.head())
```

```
     ID Delivery_person_ID  Delivery_person_Age  Delivery_person_Ratings  \
0  4607     INDORES13DEL02                   37                      4.9
1  B379     BANGRES18DEL02                   34                      4.5
2  5D6D     BANGRES19DEL01                   23                      4.4
3  7A6A    COIMBRES13DEL02                   38                      4.7
4  70A2     CHENRES12DEL01                   32                      4.6

   Restaurant_latitude  Restaurant_longitude  Delivery_location_latitude  \
0            22.745049             75.892471                   22.765049
1            12.913041             77.683237                   13.043041
2            12.914264             77.678400                   12.924264
3            11.003669             76.976494                   11.053669
4            12.972793             80.249982                   13.012793

   Delivery_location_longitude Type_of_order Type_of_vehicle  Time_taken(min)
0                    75.912471        Snack      motorcycle                24
1                    77.813237        Snack         scooter                33
2                    77.688400       Drinks      motorcycle                26
3                    77.026494       Buffet      motorcycle                21
4                    80.289982        Snack         scooter                30
```

In [2]:
```
#Let’s have a look at the column insights before moving forward:
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
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)
memory usage: 3.8+ MB
```

In [3]:

```
#let’s have a look at whether this dataset contains any null values or not:
data.isnull().sum()
```

Out[3]:

```
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
```

Calculating Distance Between Two Latitudes and Longitudes

All we have are the latitude and longitude points of the restaurant and the delivery location. We can use the haversine formula to calculate the distance between two locations based on their latitudes and longitudes.

In [4]:

```
# Set the earth's radius (in kilometers)
R = 6371

# Convert degrees to radians
def deg_to_rad(degrees):
    return degrees * (np.pi/180)

# Function to calculate the distance between two points using the haversine form
def distcalculate(lat1, lon1, lat2, lon2):
    d_lat = deg_to_rad(lat2-lat1)
    d_lon = deg_to_rad(lon2-lon1)
    a = np.sin(d_lat/2)**2 + np.cos(deg_to_rad(lat1)) * np.cos(deg_to_rad(lat2))
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    return R * c
```

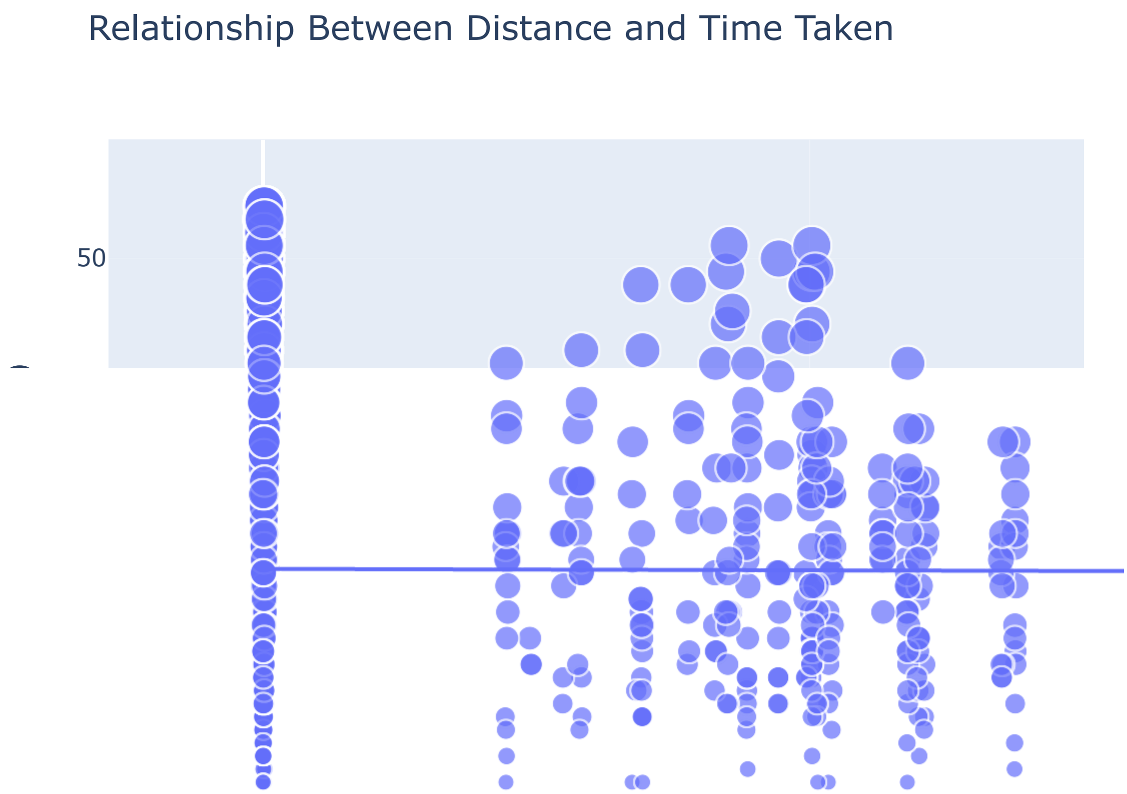
```
# Calculate the distance between each pair of points
data['distance'] = np.nan

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

In [5]:
```
#We have also added a new feature in the dataset as distance. Let’s look at the
print(data.head())
```

```
     ID Delivery_person_ID  Delivery_person_Age  Delivery_person_Ratings  \
0  4607     INDORES13DEL02                   37                      4.9
1  B379     BANGRES18DEL02                   34                      4.5
2  5D6D     BANGRES19DEL01                   23                      4.4
3  7A6A    COIMBRES13DEL02                   38                      4.7
4  70A2     CHENRES12DEL01                   32                      4.6

   Restaurant_latitude  Restaurant_longitude  Delivery_location_latitude  \
0            22.745049             75.892471                   22.765049
1            12.913041             77.683237                   13.043041
2            12.914264             77.678400                   12.924264
3            11.003669             76.976494                   11.053669
4            12.972793             80.249982                   13.012793

   Delivery_location_longitude Type_of_order Type_of_vehicle  Time_taken(min)  \
0                    75.912471        Snack      motorcycle               24
1                    77.813237        Snack         scooter               33
2                    77.688400       Drinks      motorcycle               26
3                    77.026494       Buffet      motorcycle               21
4                    80.289982        Snack         scooter               30

    distance
0   3.025149
1  20.183530
2   1.552758
3   7.790401
4   6.210138
```

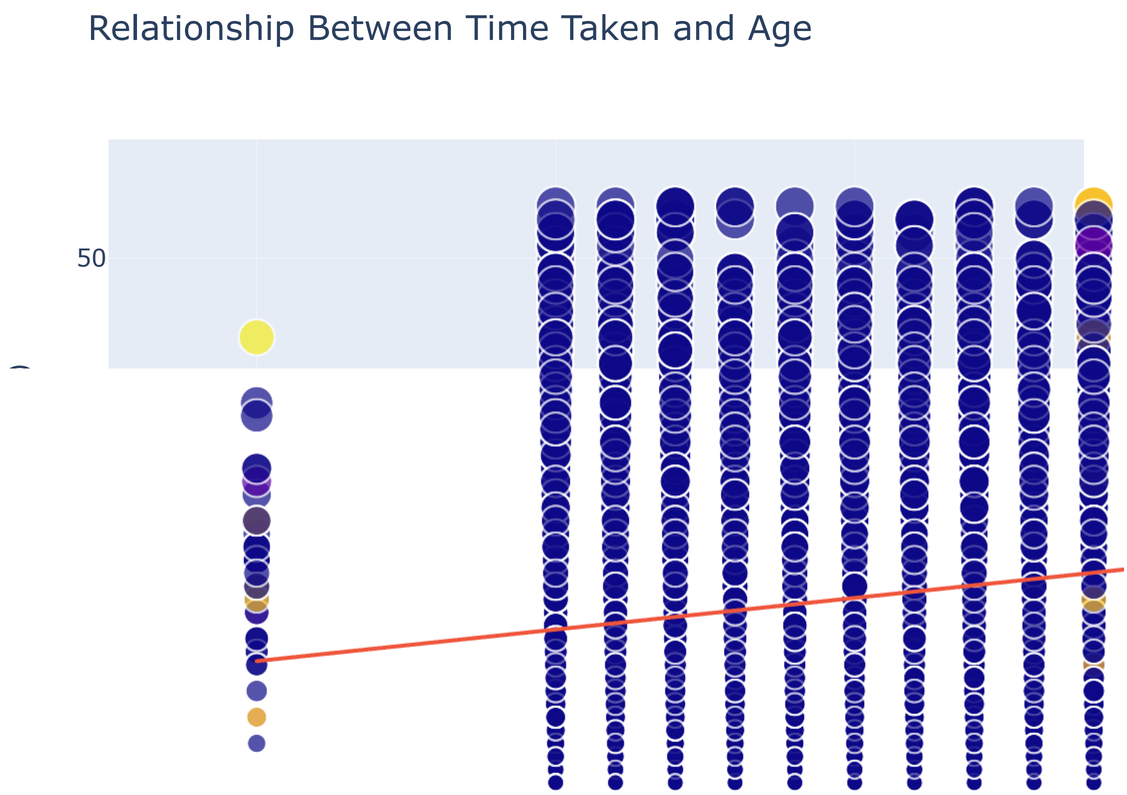
Data Exploration

In [6]:
```
#Let’s explore the data to find relationships between the features. I’ll start b
figure = px.scatter(data_frame = data,
                    x="distance",
                    y="Time_taken(min)",
                    size="Time_taken(min)",
                    trendline="ols",
                    title = "Relationship Between Distance and Time Taken")
figure.show()
```



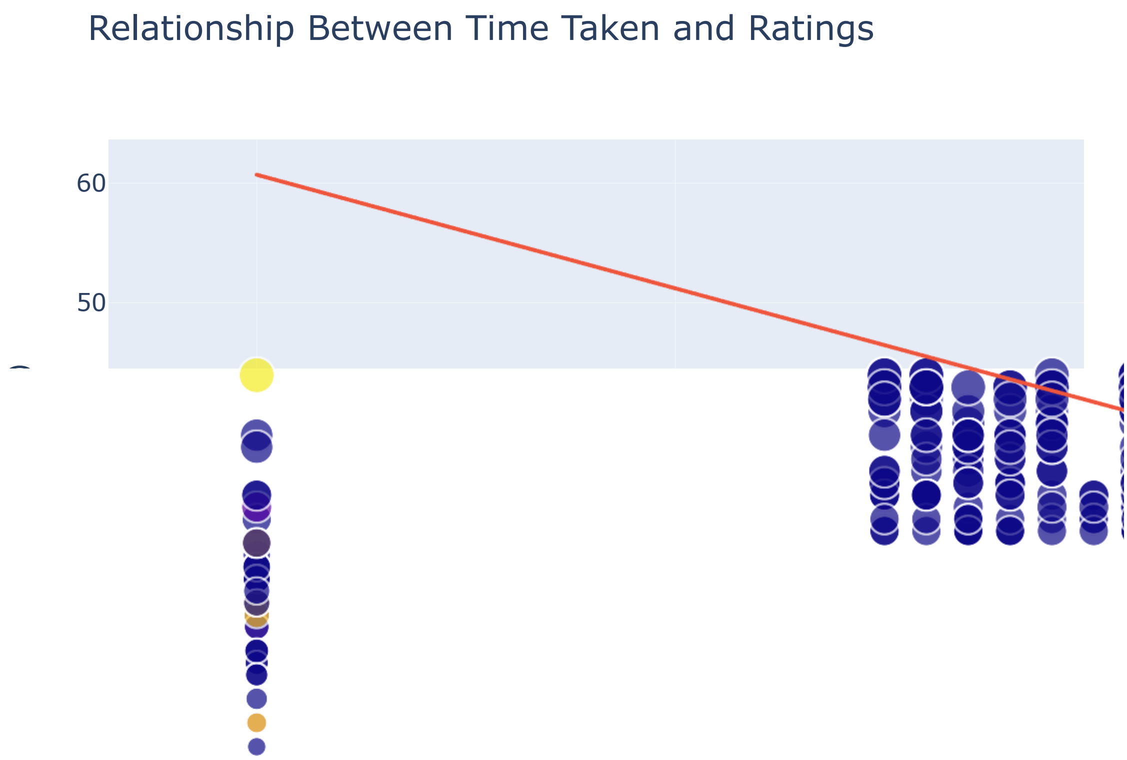
There is a consistent relationship between the time taken and the distance travelled to deliver the food. It means that most delivery partners deliver food within 25-30 minutes, regardless of distance.

In [7]:
```
#let’s have a look at the relationship between the time taken to deliver the foo
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 Time Taken and Age")
figure.show()
```



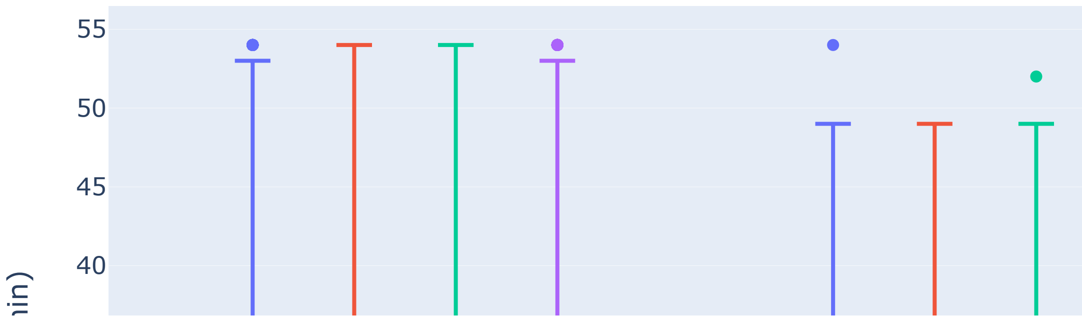
There is a linear relationship between the time taken to deliver the food and the age of the delivery partner. It means young delivery partners take less time to deliver the food compared to the elder partners.

In [8]:
```
#let’s have a look at the relationship between the time taken to deliver the foo
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 Time Taken and Ratings")
figure.show()
```



There is an inverse linear relationship between the time taken to deliver the food and the ratings of the delivery partner. It means delivery partners with higher ratings take less time to deliver the food compared to partners with low ratings.

In [9]:
```
#let’s have a look if the type of food ordered by the customer and the type of v
fig = px.box(data,
             x="Type_of_vehicle",
             y="Time_taken(min)",
             color="Type_of_order")
fig.show()
```



So there is not much difference between the time taken by delivery partners depending on the vehicle they are driving and the type of food they are delivering.

So the features that contribute most to the food delivery time based on our analysis are:

age of the delivery partner ratings of the delivery partner distance between the restaurant and the delivery location

Food Delivery Time Prediction Model

In [10]:
```
#let’s train a Machine Learning model using an LSTM neural network model for the
#splitting data
from sklearn.model_selection import train_test_split
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_size=0.10,
                                                random_state=42)

# creating the LSTM neural network model
from keras.models import Sequential
from keras.layers import Dense, LSTM
```

```
model = Sequential()
model.add(LSTM(128, return_sequences=True, input_shape= (xtrain.shape[1], 1)))
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: 117619 (459.45 KB)
Trainable params: 117619 (459.45 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [11]:

```
# training the model
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(xtrain, ytrain, batch_size=1, epochs=9)
```

```
Epoch 1/9
41033/41033 [==============================] - 406s 10ms/step - loss: 69.2657
Epoch 2/9
41033/41033 [==============================] - 402s 10ms/step - loss: 63.7669
Epoch 3/9
41033/41033 [==============================] - 400s 10ms/step - loss: 61.3817
Epoch 4/9
31801/41033 [======================>.......] - ETA: 1:19 - loss: 60.7540
```

```
IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
41033/41033 [==============================] - 254s 6ms/step - loss: 59.7667
Epoch 6/9
 5647/41033 [===>..........................] - ETA: 3:16 - loss: 58.4936
```

```
IOPub message rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_msg_rate_limit`.

Current values:
NotebookApp.iopub_msg_rate_limit=1000.0 (msgs/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
41033/41033 [==============================] - 222s 5ms/step - loss: 59.0098
Epoch 8/9
41033/41033 [==============================] - 234s 6ms/step - loss: 59.4412
Epoch 9/9
41033/41033 [==============================] - 233s 6ms/step - loss: 59.2663
```

Out[11]: <keras.src.callbacks.History at 0x188d31dd9d0>

In [12]:

```
#let’s test the performance of our model by giving inputs to predict the food d
print("Food Delivery Time Prediction")
a = int(input("Age of Delivery Partner: "))
b = float(input("Ratings of Previous Deliveries: "))
c = int(input("Total Distance: "))

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

```
Food Delivery Time Prediction
Age of Delivery Partner: 42
Ratings of Previous Deliveries: 5
Total Distance: 20
1/1 [==============================] - 1s 811ms/step
Predicted Delivery Time in Minutes =  [[30.022797]]
```

In [ ]: