Problem Description:

Porter is India's Largest Marketplace for Intra-City Logistics. Leader in the country's $40 billion intra-city logistics market, it strives to improve the lives of 1,50,000+ driver-partners or dashers by providing them with consistent earning & independence. Currently, the company has serviced 5+ million customers. It works with a wide range of restaurants for delivering their items directly to the people. It has a number of delivery partners available for delivering the food, from various restaurants and wants to get an estimated delivery time that it can provide the customers on the basis of what they are ordering, from where and also the delivery partners. This dataset has the required data to train a regression model that will do the delivery time estimation, based on all those features.

Each column corresponds to a feature as explained below.

- market_id : integer id for the market where the restaurant lies
- created_at : the timestamp at which the order was placed
- actual_delivery_time : the timestamp when the order was delivered
- store_primary_category : category for the restaurant
- order_protocol : integer code value for order protocol(how the order was placed ie: through porter, call to restaurant, pre booked, third party etc)
- total_items : number of items in order
- subtotal : final price of the order
- num_distinct_items : the number of distinct items in the order
- min_item_price : price of the cheapest item in the order
- max_item_price : price of the costliest item in order
- total_onshift_dashers : number of delivery dashers on duty at the time order was placed
- total_busy_dashers : number of delivery dashers attending to other tasks
- total_outstanding_orders : total number of orders to be fulfilled at the moment

All of these data columns can give us some hints to predict future delivery times. I will create the output columns as

```
delivery_time = actual_delivery_time - created_at
```

In [2]:
```python
df = pd.read_csv('data_2.csv')
```

In [3]:
```python
df.head()
```

Out[3]:

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_
0	1.0	2015-02-06 22:24:17	2015-02-06 23:11:17	4	1.0	4	3441	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:33:25	46	2.0	1	1900	
2	2.0	2015-02-16 00:11:35	2015-02-16 01:06:35	36	3.0	4	4771	
3	1.0	2015-02-12 03:36:46	2015-02-12 04:35:46	38	1.0	1	1525	
4	1.0	2015-01-27 02:12:36	2015-01-27 02:58:36	38	1.0	2	3620	

In [5]:
```
# df.drop(['estimated_store_to_consumer_driving_duration'],axis=1,inplace=True)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 13 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   market_id                 175777 non-null  float64
 1   created_at                175777 non-null  object 
 2   actual_delivery_time      175777 non-null  object 
 3   store_primary_category    175777 non-null  int64  
 4   order_protocol            175777 non-null  float64
 5   total_items               175777 non-null  int64  
 6   subtotal                  175777 non-null  int64  
 7   num_distinct_items        175777 non-null  int64  
 8   min_item_price            175777 non-null  int64  
 9   max_item_price            175777 non-null  int64  
 10  total_onshift_dashers     175777 non-null  float64
 11  total_busy_dashers        175777 non-null  float64
 12  total_outstanding_orders  175777 non-null  float64
dtypes: float64(5), int64(6), object(2)
memory usage: 17.4+ MB
```

There are 1,75,777 rows in the dataframe, so it is a large dataset. There are no null values.

3 pandas datetime functions.

Datetime: A specific date and time with timezone support. Similar to datetime.datetime from the standard library.

Time deltas: An absolute time duration. Similar to datetime.timedelta from the standard library.

Time spans: A span of time defined by a point in time and its associated frequency.

In [7]:
```
df['created_at'] = pd.to_datetime(df['created_at'])
df['actual_delivery_time'] = pd.to_datetime(df['actual_delivery_time'])
```

In [8]:
```
df.describe()
```

Out[8]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	n
count	175777.000000	175777.000000	175777.000000	175777.000000	175777.000000	175777.000000	
mean	2.743726	35.887949	2.911752	3.204976	2697.111147	2.675060	
std	1.330963	20.728254	1.513128	2.674055	1828.554893	1.625681	
min	1.000000	0.000000	1.000000	1.000000	0.000000	1.000000	
25%	2.000000	18.000000	1.000000	2.000000	1412.000000	1.000000	
50%	2.000000	38.000000	3.000000	3.000000	2224.000000	2.000000	
75%	4.000000	55.000000	4.000000	4.000000	3410.000000	3.000000	
max	6.000000	72.000000	7.000000	411.000000	26800.000000	20.000000	

Market ID has values in the range 1-6, with mean value 2.74 and median 2. There may not be a lot of outliers.

Store primary category has range 0-72, 35.9 mean and 38 median. There maybe a few outliers.

Order protocol has range 1-7, with mean 2.9 and median 3. There maybe a few outliers.

Total items has range 1-411, with mean 3.2 and median 3. There maybe a few outliers.

Subtotal has range 0 to Rs. 26,800. Mean Rs. 2.7k and median 2.2k. There maybe a few outliers.

Number of distinct items has range 1 to 20, with mean 2.67 and median 2. There maybe a few outliers.

Minimum item price ranges from Rs.(-86) to Rs. 14,700. The negative values need to be removed. Mean 684 and median 595. There are many outliers.

Maximum item price has range Rs.0 to Rs. 14,700. Mean Rs. 1,160 and median 1,095. There maybe a few outliers.

Total onshift partners has range -4 to 171. The negative values are erroneous and need to be removed. Mean 44.9 median 37. There maybe a few outliers.

Total busy dashers range from -5 to 154. Mean 41.9 and median 35. There are many outliers.

Total outstanding orders range from -6 to 285. Mean 58.2 and median 41. There are many outliers.

Data preprocessing and feature engineering

In [9]:
```
# clipping off negative valued outliers
df['min_item_price'] = df['min_item_price'].clip(lower=0)
df['min_item_price'].describe()
```

Out[9]:
```
count    175777.000000
mean        684.967533
std         519.880057
min           0.000000
25%         299.000000
50%         595.000000
75%         942.000000
max       14700.000000
Name: min_item_price, dtype: float64
```

In [10]:
```
df['total_onshift_dashers'] = df['total_onshift_dashers'].clip(lower=0)
df['total_onshift_dashers'].describe()
```

Out[10]:
```
count    175777.000000
mean         44.918886
std          34.544429
min           0.000000
25%          17.000000
50%          37.000000
75%          66.000000
max         171.000000
Name: total_onshift_dashers, dtype: float64
```

In [11]:
```
df['total_busy_dashers'] = df['total_busy_dashers'].clip(lower=0)
df['total_busy_dashers'].describe()
```

Out[11]:
```
count    175777.000000
mean         41.861597
std          32.168215
min           0.000000
25%          15.000000
50%          35.000000
75%          63.000000
max         154.000000
Name: total_busy_dashers, dtype: float64
```

In [12]:
```
df['total_outstanding_orders'] = df['total_outstanding_orders'].clip(lower=0)
df['total_outstanding_orders'].describe()
```

Out[12]:
```
count    175777.000000
mean         58.230758
std          52.730310
min           0.000000
25%          17.000000
50%          41.000000
75%          85.000000
max         285.000000
Name: total_outstanding_orders, dtype: float64
```

In [13]:
```
df['order_hour'] = df['created_at'].dt.hour
```

In [14]:
```
df['order_hour'].value_counts()
```

Out[14]:
```
2     32896
1     25722
3     23693
20    13883
4     13248
19    12083
0     11464
21    10219
22     7875
23     7334
5      6078
18     4514
17     3058
16     1936
6      1223
15      502
14       38
7         9
8         2
Name: order_hour, dtype: int64
```

There seems to be missing values from 9 am to 1pm in order hours, or that is not a time preferred by most office going people for ordering food.

Duplicated rows checking

In [15]:
```
df.duplicated().sum()  # no duplicate rows
```

Out[15]:
```
0
```

Creating total available dashers feature

In [16]:
```
df['total_available_dashers'] = df['total_onshift_dashers'] - df['total_busy_dashers']
```

In [17]:
```
# removing useless features
df.drop(['total_onshift_dashers','total_busy_dashers'],axis=1,inplace=True)
```

Creating the target column

In [18]:
```
df['delivery_time'] = df['actual_delivery_time'] - df['created_at']
```

In [19]:
```
df['delivery_time'].head()
```

Out[19]:
```
0   0 days 00:47:00
1   0 days 00:44:00
2   0 days 00:55:00
3   0 days 00:59:00
4   0 days 00:46:00
Name: delivery_time, dtype: timedelta64[ns]
```

In [20]:
```
df['delivery_time(min)'] = df['delivery_time'].dt.total_seconds()/60  # minutes
```

In [21]:
```
df['delivery_time(min)'].head()
```

```
Out[21]: 0    47.0
1    44.0
2    55.0
3    59.0
4    46.0
Name: delivery_time(min), dtype: float64
```

```
In [22]: #### Getting the day of the week from the order time
from datetime import datetime
# get weekday name
df['order_day'] = df['created_at'].dt.strftime('%A')
```

```
In [23]: df['order_day'].head()
```

```
Out[23]: 0      Friday
1     Tuesday
2      Monday
3    Thursday
4     Tuesday
Name: order_day, dtype: object
```

```
In [24]: df.drop(['created_at','actual_delivery_time'], axis=1, inplace=True)
```

```
In [25]: df.drop(['delivery_time'], axis=1, inplace=True)
```

```
In [26]: df.head()
```

Out[26]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	m
0	1.0	4	1.0	4	3441	4	557	
1	2.0	46	2.0	1	1900	1	1400	
2	2.0	36	3.0	4	4771	3	820	
3	1.0	38	1.0	1	1525	1	1525	
4	1.0	38	1.0	2	3620	2	1425	

Handling null values

```
In [27]: df.isnull().sum()/len(df)*100
```

```
Out[27]: market_id                  0.0
store_primary_category     0.0
order_protocol             0.0
total_items                0.0
subtotal                   0.0
num_distinct_items         0.0
min_item_price             0.0
max_item_price             0.0
total_outstanding_orders   0.0
order_hour                 0.0
total_available_dashers    0.0
delivery_time(min)         0.0
order_day                  0.0
dtype: float64
```

Encoding categorical columns

In [28]:
```
df.columns[df.dtypes =="object"]
```

Out[28]:
```
Index(['order_day'], dtype='object')
```

Out of these, store_id are large strings that need to be labeled but had high cardinality.

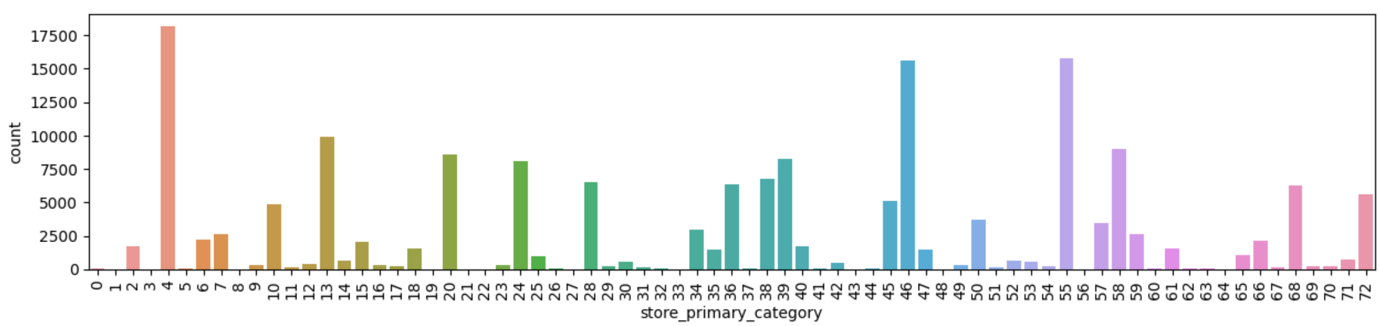
Data visualization and cleaning

Univariate analysis

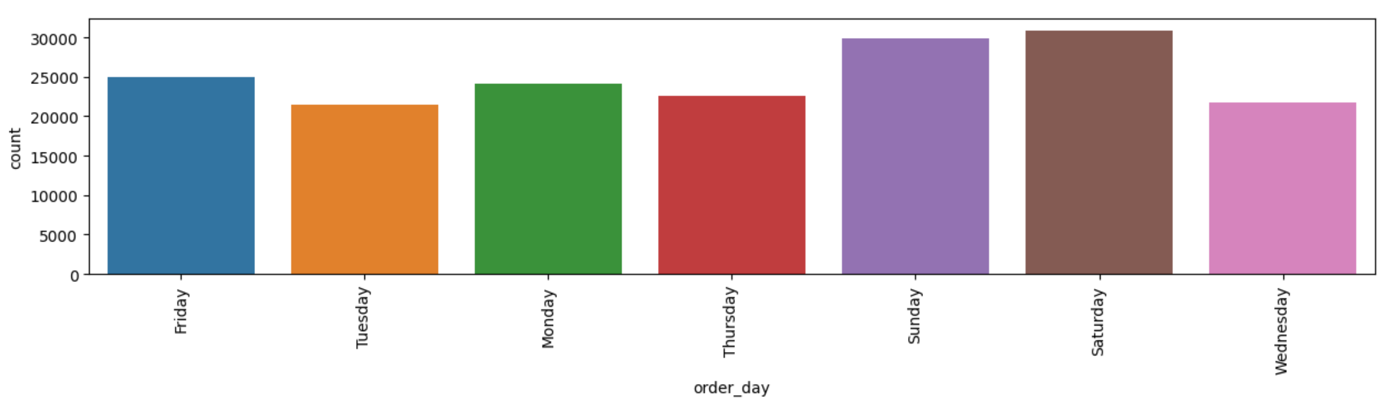
In [29]:
```
import seaborn as sns, matplotlib.pyplot as plt
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\scipy\__init__.py:169:
UserWarning: A NumPy version >=1.18.5 and <1.26.0 is required for this version of SciPy (detecte
d version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [30]:
```
f = plt.figure(figsize=(15,3))
sns.countplot(data=df, x = 'store_primary_category')
plt.xticks(rotation=90)
plt.show()
```



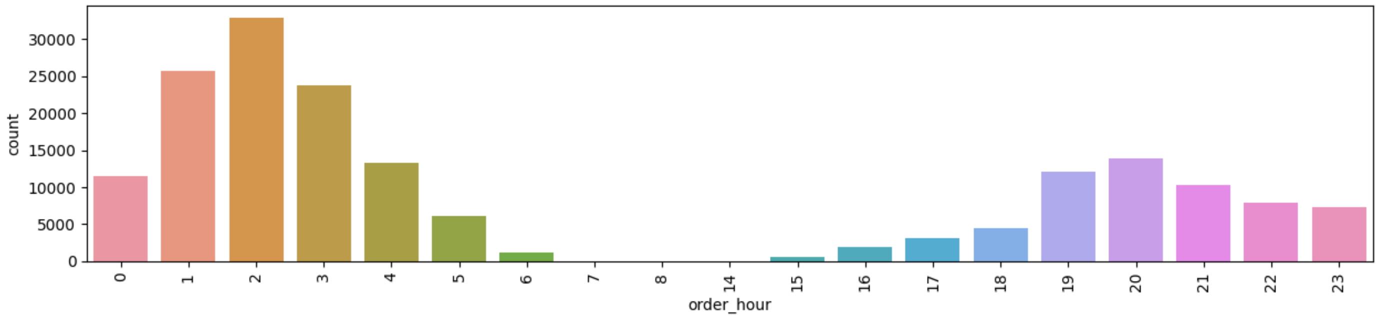
In [31]:
```
f = plt.figure(figsize=(15,3))
sns.countplot(data=df, x = 'order_day')
plt.xticks(rotation=90)
plt.show()
```



Most orders were placed on Saturday and Sunday.

In [32]:
```
f = plt.figure(figsize=(15,3))
sns.countplot(data=df, x = 'order_hour')
```

```
plt.xticks(rotation=90)
plt.show()
```

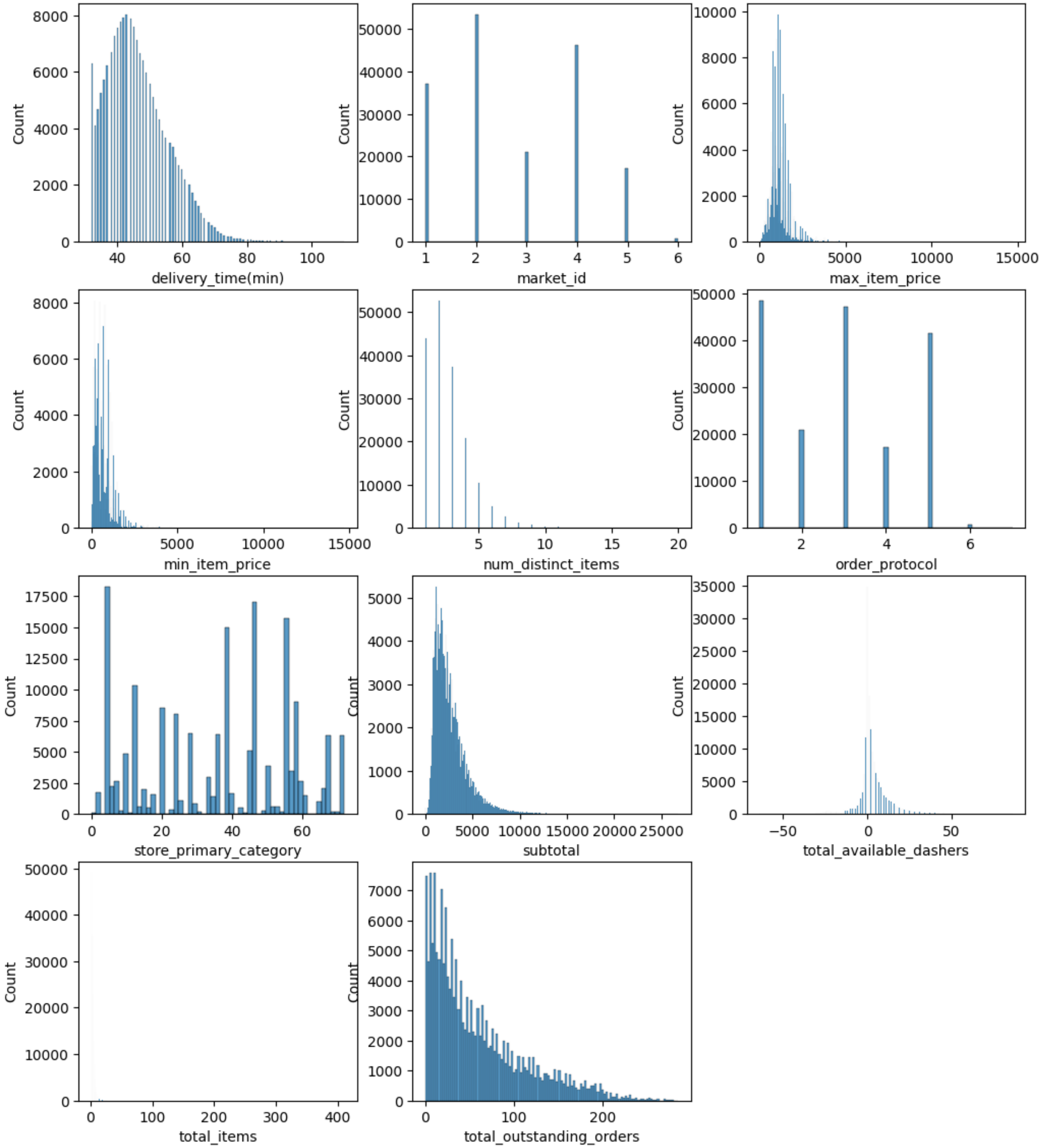


There seems to be missing values from 9 am to 1pm in order hours.

In [33]:
```
continuous_cols = df.columns[(df.dtypes == "float64") | (df.dtypes == "int64")]
continuous_cols = continuous_cols.difference(['order_hour'])
```

In [34]:
```
## histogram subplots
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(14)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(4,(n//4)+1,i+1)
    sns.histplot(data=df, x=continuous_cols[i])
plt.show()
```

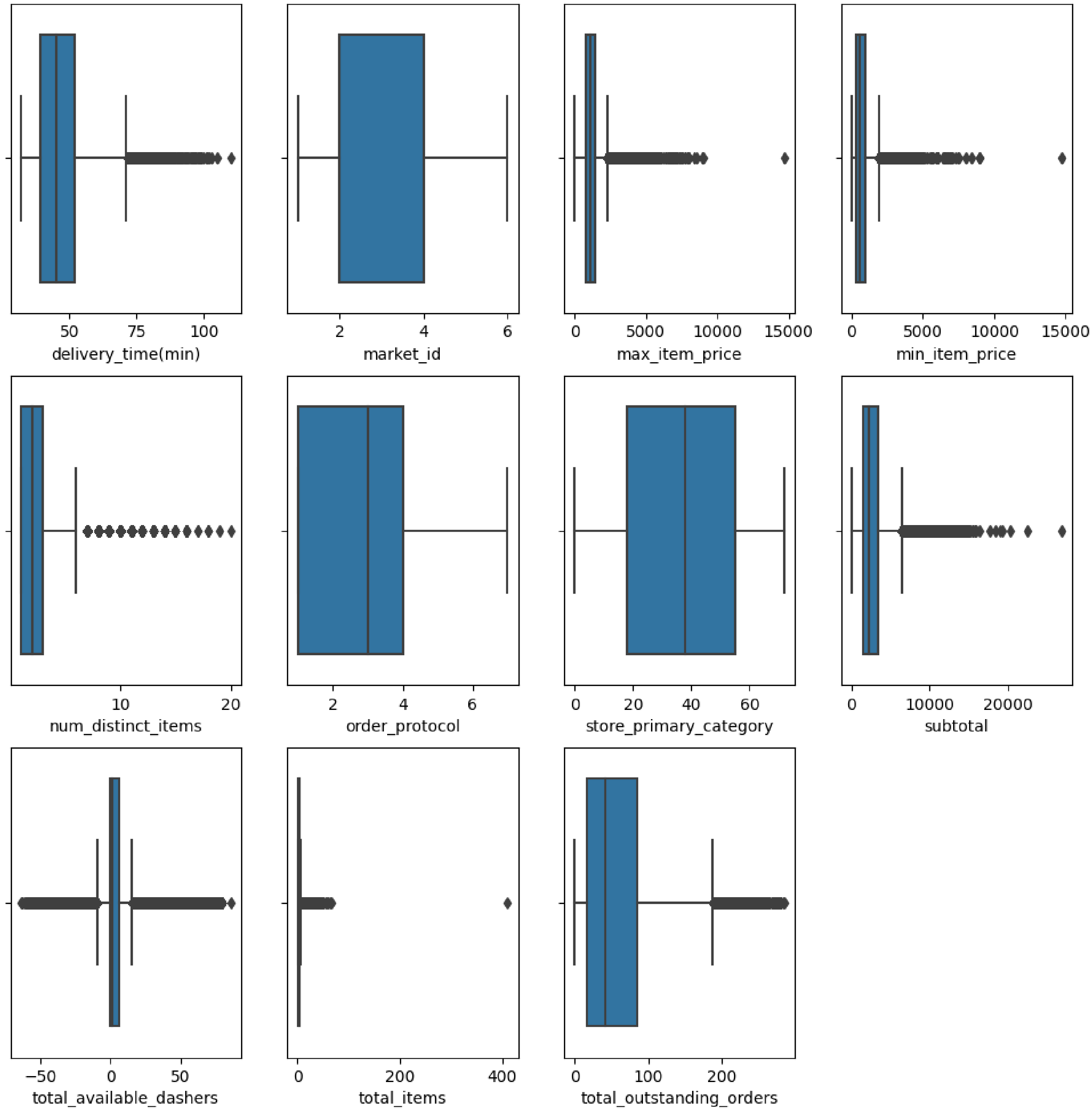


Max_item_price, min_item_price, num_distinct_items, subtotal, total_available_dashers, total_outstanding_orders all of these have unimodal distribution, all right-skewed.

In [35]:

```
## boxplot subplots
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(12)
n = len(continuous_cols)

for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df, x=continuous_cols[i])
plt.show()
```

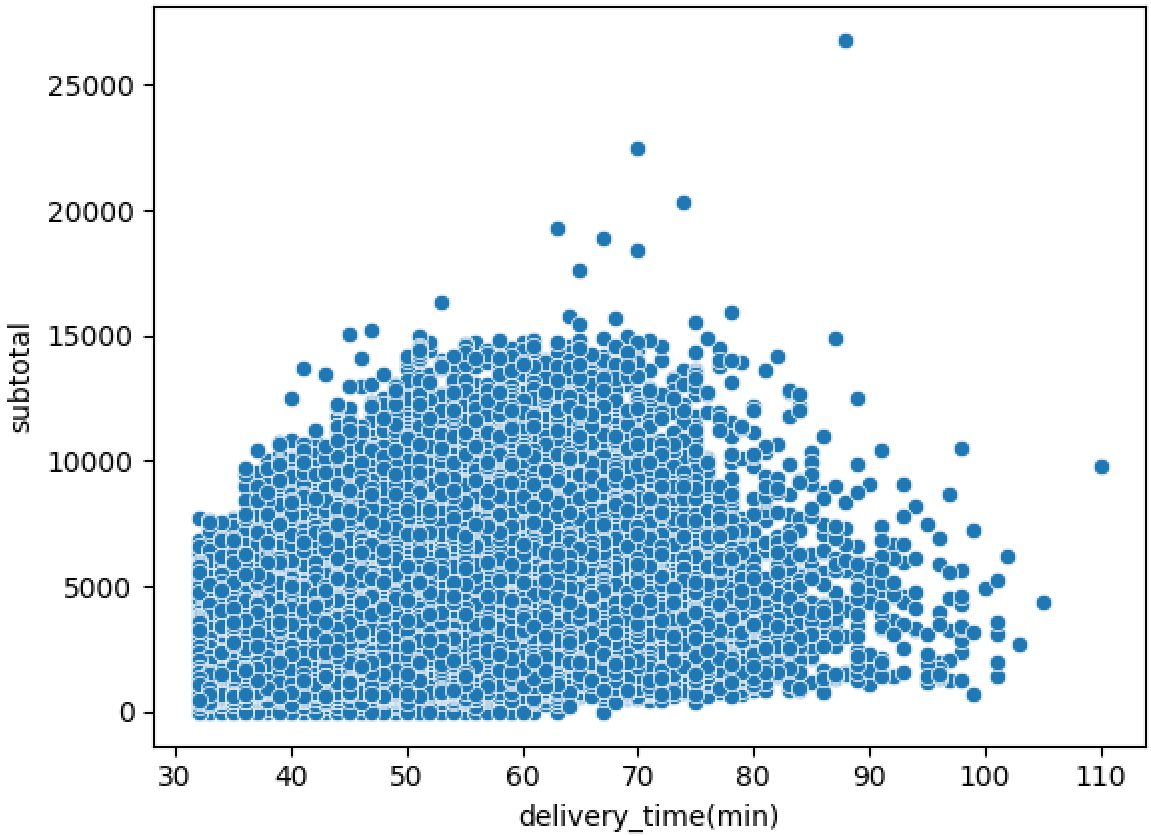


There are a lot of outliers that need to be removed.

Bivariate analysis

In [36]:
```
sns.scatterplot(x='delivery_time(min)', y='subtotal', data=df)
```

Out[36]: <AxesSubplot: xlabel='delivery_time(min)', ylabel='subtotal'>



In [37]:
```
continuous_cols = continuous_cols.difference(['delivery_time(min)'])
```

In [38]:
```
df.head()
```

Out[38]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	m
0	1.0	4	1.0	4	3441	4	557	
1	2.0	46	2.0	1	1900	1	1400	
2	2.0	36	3.0	4	4771	3	820	
3	1.0	38	1.0	1	1525	1	1525	
4	1.0	38	1.0	2	3620	2	1425	

In [39]:
```
day_label_mapping = {'Sunday':1,'Monday':2,'Tuesday':3,'Wednesday':4,'Thursday':5,'Friday':6,'Sa
df['encoded_order_day'] = df['order_day'].map(day_label_mapping)
df['encoded_order_day'].head()
```

Out[39]:
```
0    6
1    3
2    2
3    5
4    3
Name: encoded_order_day, dtype: int64
```

In [40]:
```
df['order_day'].head()
```

```
Out[40]: 0      Friday
         1     Tuesday
         2      Monday
         3    Thursday
         4     Tuesday
         Name: order_day, dtype: object
```

```
In [41]: df.drop(['order_day'], axis=1, inplace=True)
```

```
In [42]: df.head()
```

Out[42]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	m
0	1.0	4	1.0	4	3441	4	557	
1	2.0	46	2.0	1	1900	1	1400	
2	2.0	36	3.0	4	4771	3	820	
3	1.0	38	1.0	1	1525	1	1525	
4	1.0	38	1.0	2	3620	2	1425	

Why do we need to check for outliers in our data?

Outliers are extreme values that differ from most other data points in a dataset. They can have a big impact on your statistical analyses and skew the results of any hypothesis tests. It's important to carefully identify potential outliers in your dataset and deal with them in an appropriate manner for accurate results.

Name 3 outlier removal methods.

- Inter quartile range method
- Local Outlier Factor
- Percentile method

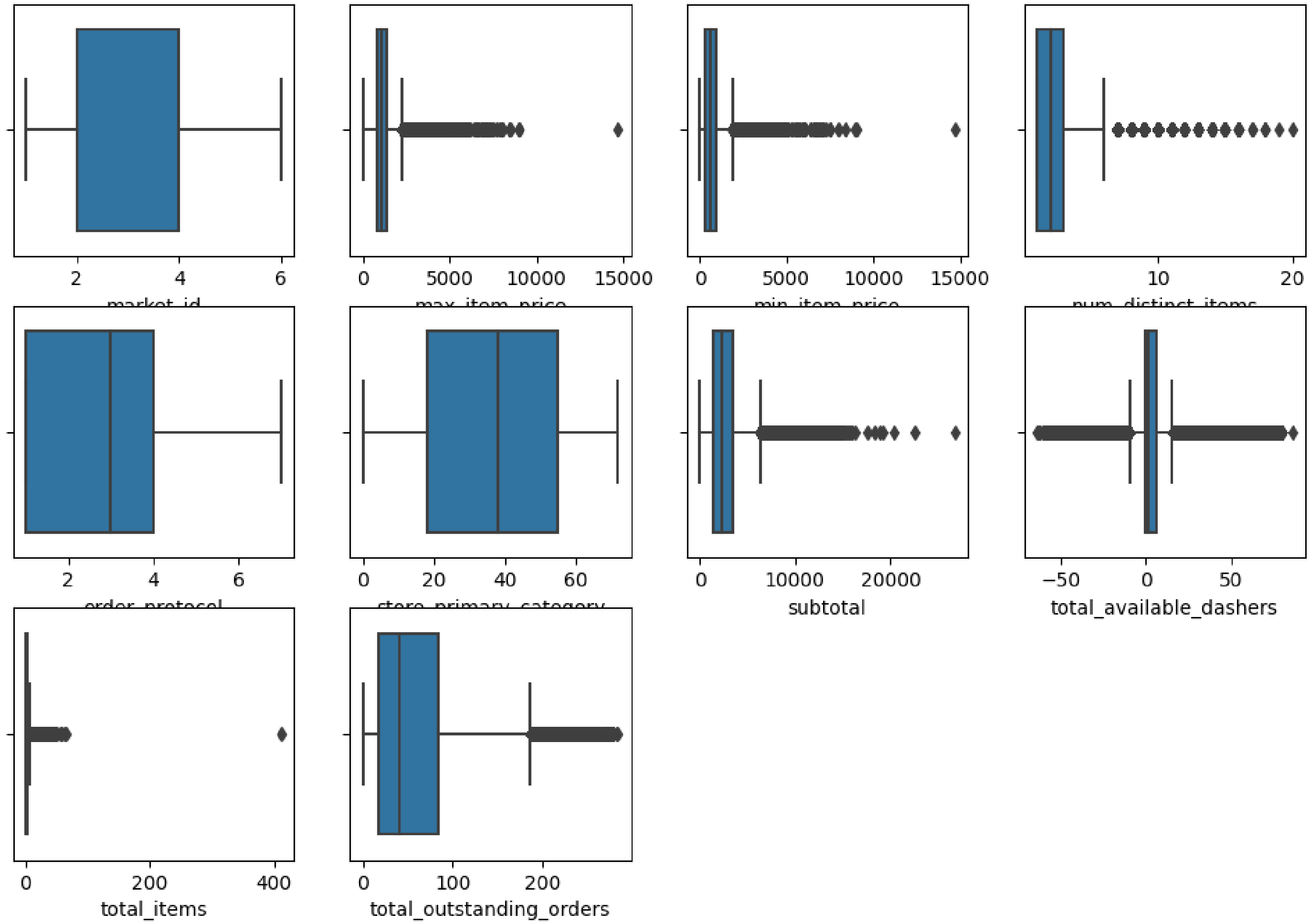
```
In [41]: # ## I am using the inter quartile method
         # n = len(continuous_cols)
         # for i in range(n):
         #     iqr = scipy.stats.iqr(df[continuous_cols[i]]) # inter quartile range
         #     q3 = np.percentile(df[continuous_cols[i]],75) # third quartile
         #     df[continuous_cols[i]] = df[continuous_cols[i]][df[continuous_cols[i]] < (q3 +iqr*1.5)] # c
```

```
In [42]: # # to remove the outliers, we look at the percentiles of datapoints
         # low = 0.1
         # high = 100
         # percentiles = np.arange(low,high,0.01)
         #  = np.percentile(df[''],q=percentiles)
         #  = pd.DataFrame(, index=percentiles)
         # print()
```

```
In [43]: ## boxplot subplots
         f = plt.figure()
         f.set_figwidth(12)
         f.set_figheight(8)
         n = len(continuous_cols)

         for i in range(n):
```

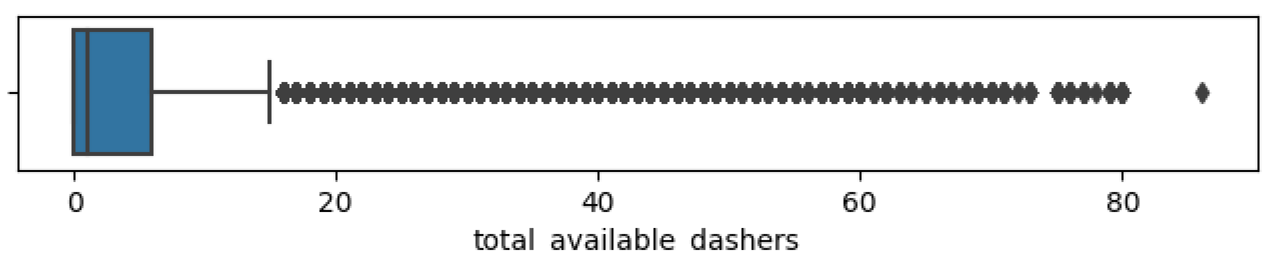
```
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df, x=continuous_cols[i])
plt.show()
```



Clipping off negative values in available dashers

In [44]:
```
df['total_available_dashers'] = df['total_available_dashers'].clip(lower=0)
f = plt.figure()
f.set_figwidth(8)
f.set_figheight(1)
sns.boxplot(data=df, x='total_available_dashers')
```

Out[44]: <AxesSubplot: xlabel='total_available_dashers'>



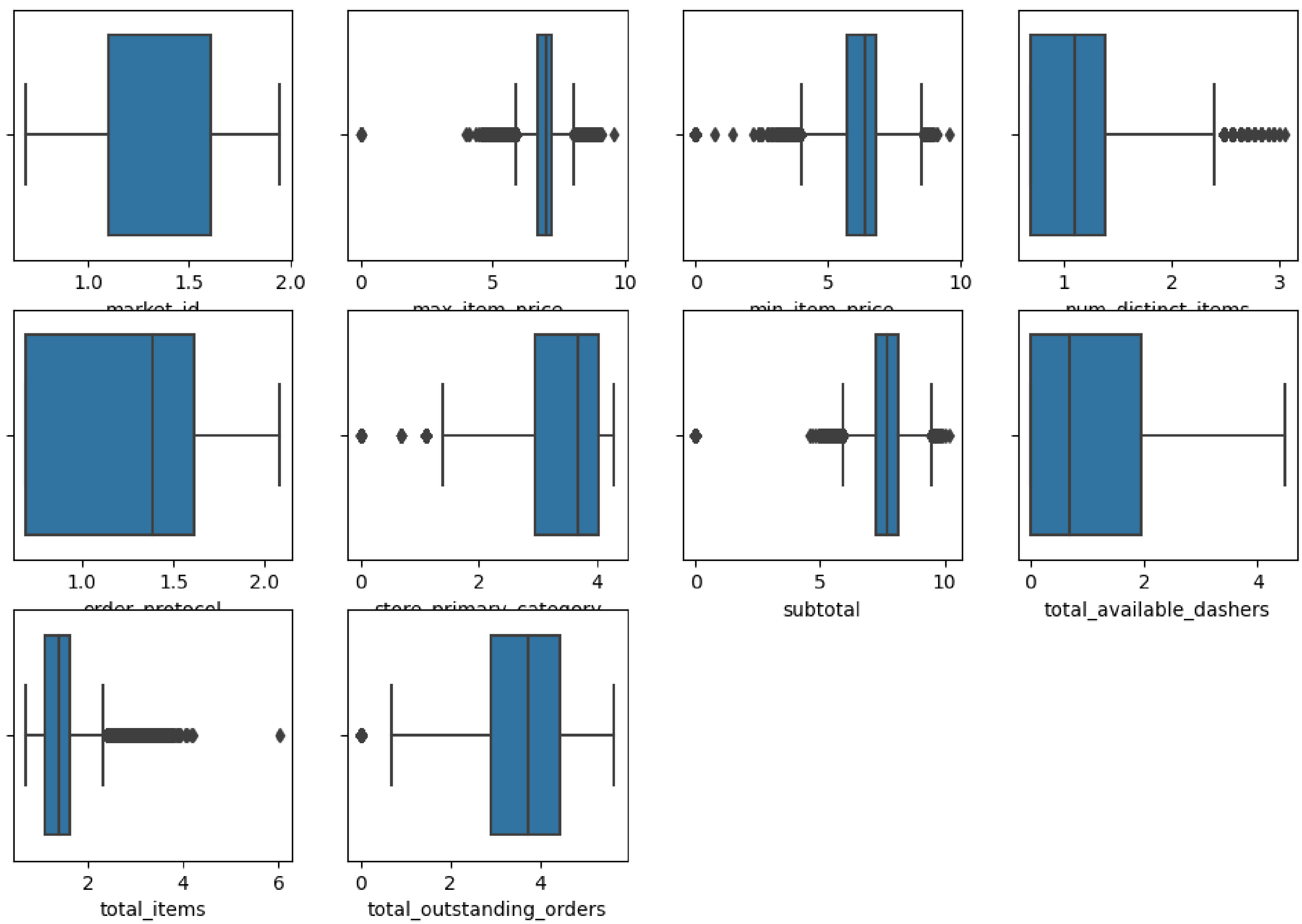
Log transformation for right-skewed data

In [45]:
```
cont_cols = continuous_cols.difference(['encoded_storeID'])
n = len(cont_cols)
for i in range(n):
    df[cont_cols[i]] = np.log1p(df[cont_cols[i]])
```

In [46]:

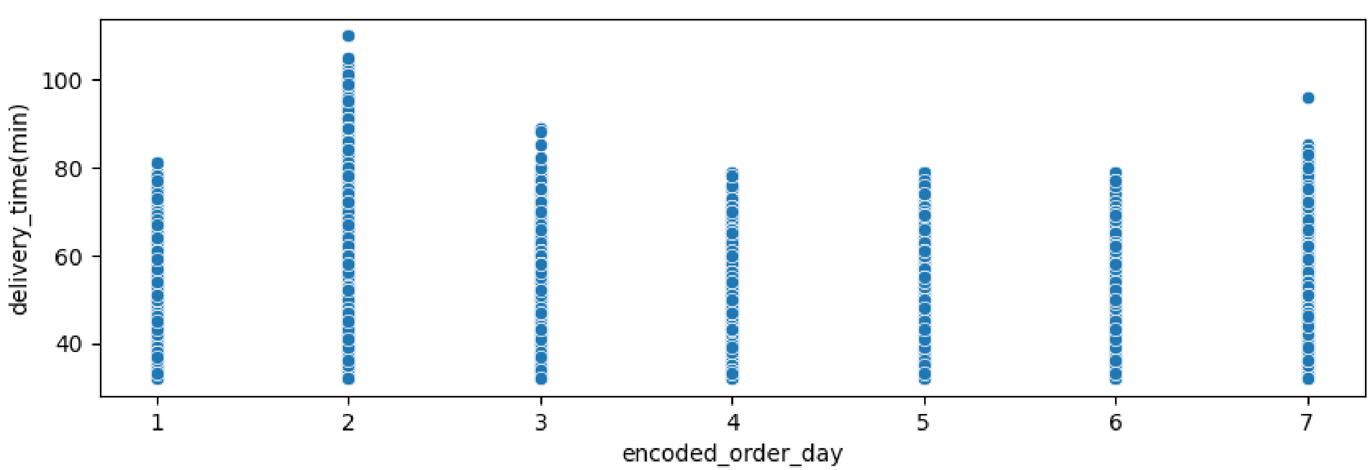
```
## boxplot subplots
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(8)
n = len(cont_cols)

for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df, x=cont_cols[i])
plt.show()
```



In [47]:

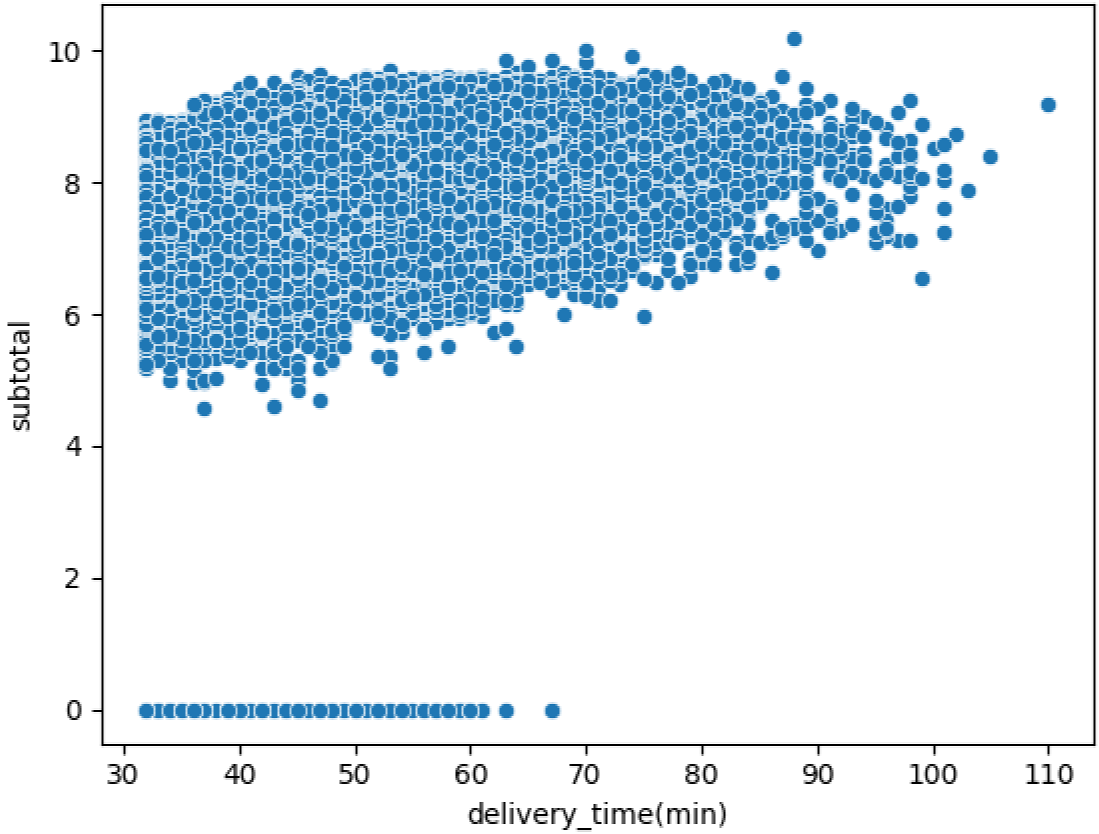
```
f = plt.figure(figsize=(10,3))
sns.scatterplot(x='encoded_order_day', y='delivery_time(min)', data=df)
plt.show()
```



The delivery time range is almost same for all weekdays - 30 to 100 minutes.

In [48]:
```
sns.scatterplot(x='delivery_time(min)', y='subtotal', data=df)
```

Out[48]: <AxesSubplot: xlabel='delivery_time(min)', ylabel='subtotal'>

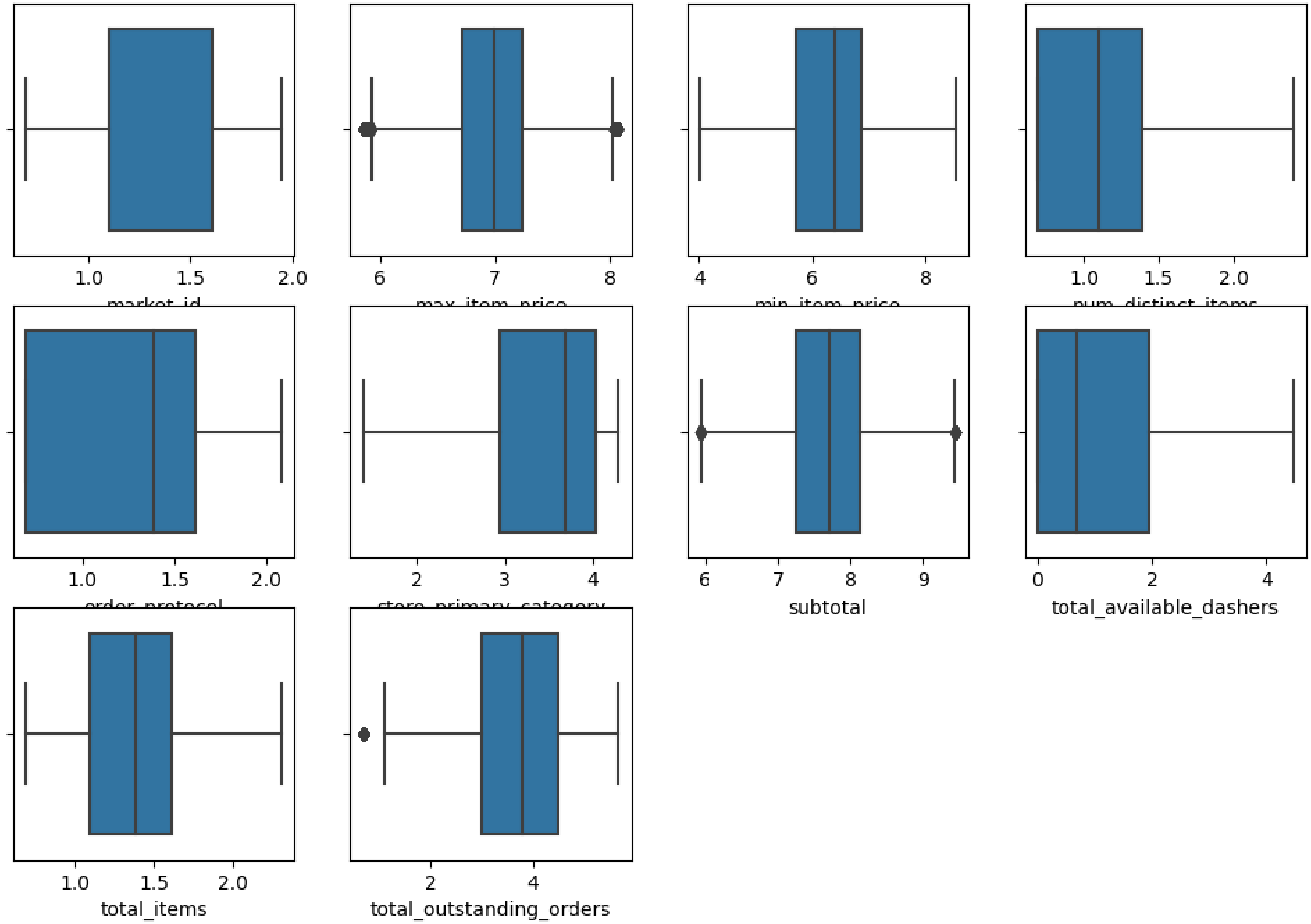


There does not seem to be a linear relationship between subtotal cost of order and delivery time.

In [49]:
```
## I am using the inter quartile method
import scipy
n = len(cont_cols)
for i in range(n):
    iqr = scipy.stats.iqr(df[cont_cols[i]]) # inter quartile range
    q1 = np.percentile(df[cont_cols[i]],25) # first quartile
    q3 = np.percentile(df[cont_cols[i]],75) # third quartile
    df[cont_cols[i]] = df[cont_cols[i]][df[cont_cols[i]] < (q3 +iqr*1.5)] # outlier points
    df[cont_cols[i]] = df[cont_cols[i]][df[cont_cols[i]] > (q1 -iqr*1.5)] # outlier points
```

In [50]:
```
## boxplot subplots
f = plt.figure()
f.set_figwidth(12)
f.set_figheight(8)
n = len(cont_cols)

for i in range(n):
    plt.subplot(3,4,i+1)
    sns.boxplot(data=df, x=cont_cols[i])
plt.show()
```



In [51]:
```
df.isnull().sum()*100/len(df)
```

Out[51]:
```
market_id                    0.000000
store_primary_category       1.057021
order_protocol               0.000000
total_items                  2.338190
subtotal                     0.429521
num_distinct_items           0.219028
min_item_price               1.784079
max_item_price               3.498751
total_outstanding_orders     2.300642
order_hour                   0.000000
total_available_dashers      0.000000
delivery_time(min)           0.000000
encoded_order_day            0.000000
dtype: float64
```

Group mean imputation

In [53]:
```
numerical_features = ['total_items','subtotal','max_item_price','min_item_price', 'total_outstand
# Impute missing values with the mean of each group
df[numerical_features] = df.groupby(['order_protocol','order_hour'])[numerical_features].transfor
```

In [54]:
```
df.isnull().sum()*100/len(df)
```

Out[54]:
```
market_id                    0.000000
store_primary_category       1.057021
order_protocol               0.000000
total_items                  0.000000
subtotal                     0.000000
num_distinct_items           0.219028
min_item_price               0.000000
max_item_price               0.000569
total_outstanding_orders     0.001707
order_hour                   0.000000
total_available_dashers      0.000000
delivery_time(min)           0.000000
encoded_order_day            0.000000
dtype: float64
```

In [55]:
```
# some NaN values still remain, if the group had all NaN values. So I will impute them by the ove
df[numerical_features] = df[numerical_features].transform(lambda x: x.fillna(x.mean()))
```

In [56]:
```
df[['store_primary_category','num_distinct_items']] = df[['store_primary_category','num_distinct_
```

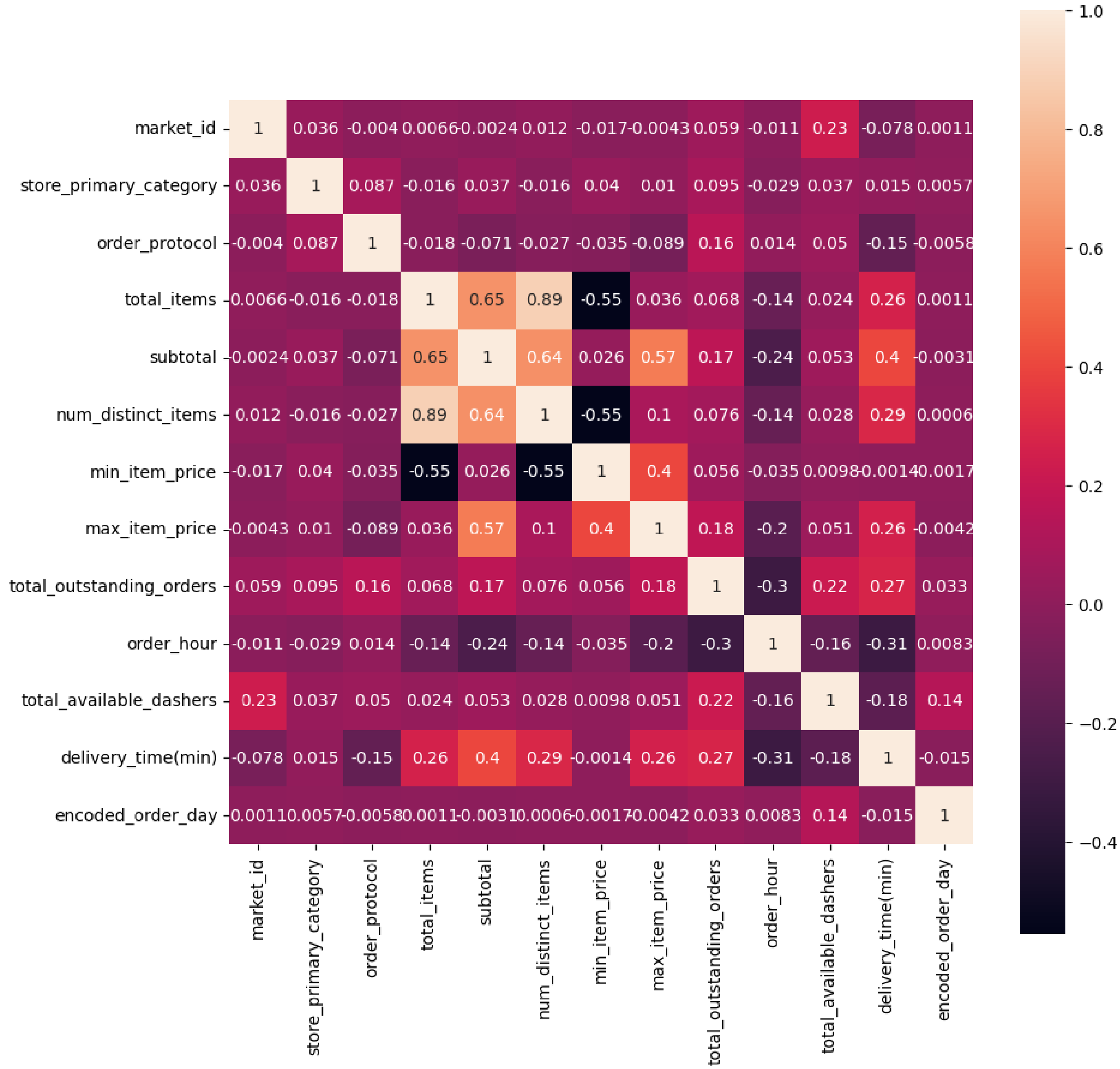
In [57]:
```
df.isnull().sum()*100/len(df)
```

Out[57]:
```
market_id                    0.0
store_primary_category       0.0
order_protocol               0.0
total_items                  0.0
subtotal                     0.0
num_distinct_items           0.0
min_item_price               0.0
max_item_price               0.0
total_outstanding_orders     0.0
order_hour                   0.0
total_available_dashers      0.0
delivery_time(min)           0.0
encoded_order_day            0.0
dtype: float64
```

Multivariate analysis

In [58]:
```
# Spearman's Rank Correlation Coefficient
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(method='spearman'), square=True,annot=True)
```

Out[58]: <AxesSubplot: >



Total items and num_distinct_items are correlated. So I only kept total_items.

In [59]:
```
df.drop(['num_distinct_items'], axis=1, inplace=True)
```

Regression with neural networks

In [60]:
```
X = df.drop(['delivery_time(min)'], axis=1)
Y = df['delivery_time(min)']
```

In [61]:
```
X.shape, Y.shape
```

Out[61]:
```
((175777, 11), (175777,))
```

In [62]:
```
from sklearn.model_selection import train_test_split
# Create training and test split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.05, random_state=4) #5% te
```

What classical machine learning methods can we use for this problem?

We can use Linear Regression, Decision Trees, Random Forests, or XGBoost algorithms for regression.

Why is scaling required for neural networks?

1. Feature scaling speeds up optimization by making the training faster. It prevents the optimization from getting stuck in local optima. Gradient descent optimization algorithms, which are commonly used for training neural networks, perform more efficiently when features are on a similar scale. If features have vastly different scales, the optimization process can take longer to converge, or it might converge to a suboptimal solution.
2. Activation functions (e.g., sigmoid, tanh, or ReLU) are sensitive to the input values, and features with large scales can dominate the activation and lead to saturation or vanishing gradients, especially in the case of sigmoid and tanh functions.
3. Scaling can result in a more spherical and well-behaved loss surface, making the optimization process more efficient. This is particularly important in high-dimensional spaces.

Minmax Normalization - because the data is not normally distributed

Otherwise for normally distributed data, I can use StandardScaler.

In [63]:
```
# Mean centering and Variance scaling
from sklearn.preprocessing import MinMaxScaler
X_columns = X_train.columns
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_columns)
X_train_scaled.head()
```

Out[63]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	min_item_price	max_item_price	total_
0	0.876951	0.766011	0.000000	0.778385	0.462219	0.218292	0.333112	
1	0.731416	0.348328	0.000000	0.000000	0.269351	0.633222	0.463696	
2	0.323657	0.076836	0.292481	0.430677	0.677252	0.698260	0.595690	
3	0.000000	0.631018	0.660964	0.682606	0.843657	0.637716	0.712151	
4	0.000000	0.631018	0.660964	0.251930	0.604126	0.655244	0.807366	

In [64]:
```
# Validation data from training data
X_train, X_val, y_train, y_val = train_test_split(X_train_scaled, y_train, test_size=0.25, randor
```

In [65]:
```
X_train.shape
```

Out[65]:
```
(125241, 11)
```

Random Forest benchmarking model

Using Random forest as a simple model to set a baseline performance metric.

In [66]:
```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor()
```

```
regressor.fit(X_train, y_train)
```

Out[66]:

```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [67]:

```
#random forest model training
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
prediction = regressor.predict(X_test_scaled)
mse = mean_squared_error(y_test, prediction)
rmse = mse**.5
print("mse : ", mse)
print("rmse : ",rmse)
mae = mean_absolute_error(y_test, prediction)
print('mae:' ,mae)
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:450: Us
erWarning: X does not have valid feature names, but RandomForestRegressor was fitted with featur
e names
  warnings.warn(
mse :  32.48861183279437
rmse :  5.699878229646171
mae: 4.414360802735021
```

In [68]:

```
r2_score(y_test, prediction)
```

Out[68]:

```
0.6194182901358741
```

In [69]:

```
def MAPE(Y_actual,Y_Predicted):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
```

In [70]:

```
print("mape : ",MAPE(y_test, prediction))
```

```
mape :  9.712284898022961
```

In [71]:

```
feature_importances = pd.Series(regressor.feature_importances_, index=X_train.columns)
print(feature_importances)
```

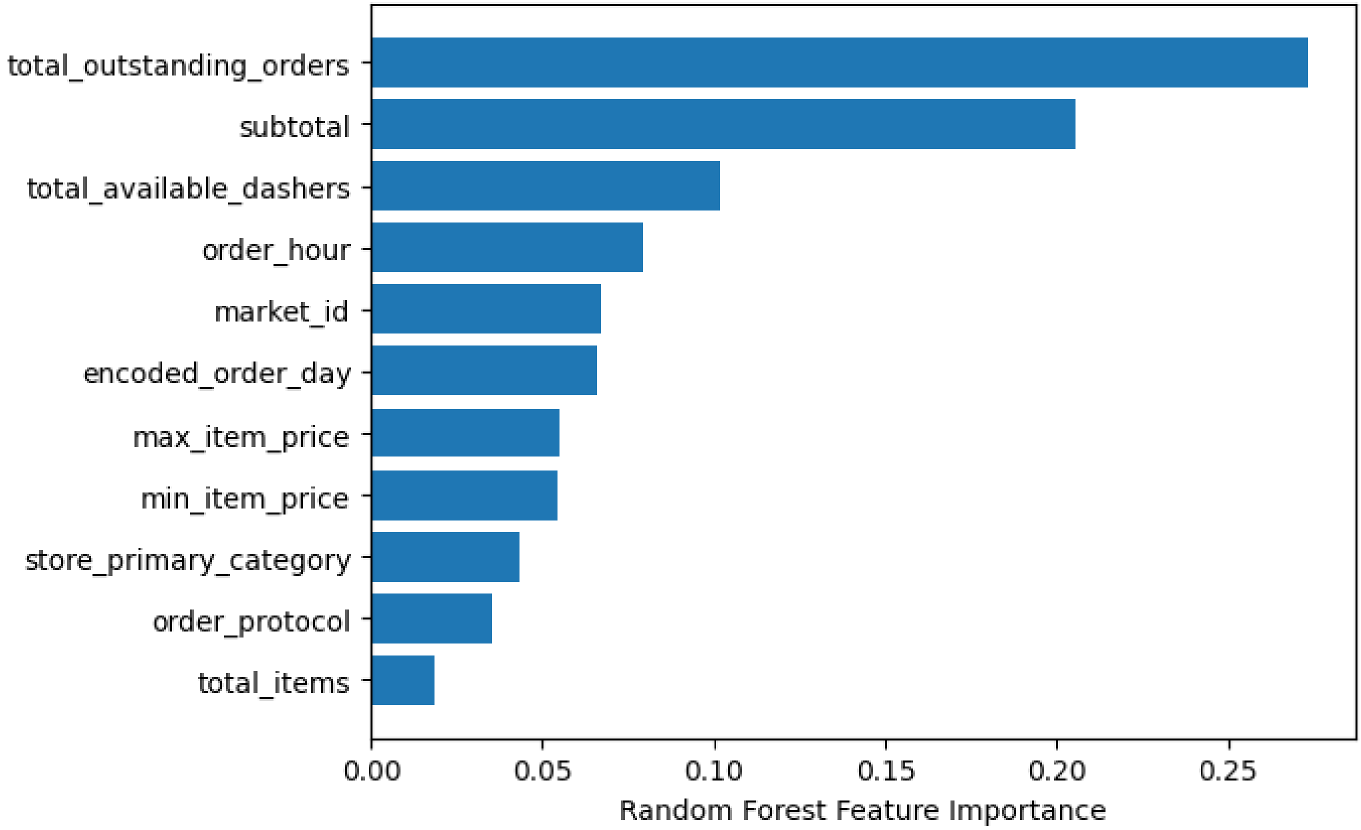
```
market_id                   0.066993
store_primary_category      0.043605
order_protocol              0.035423
total_items                 0.018232
subtotal                    0.205535
min_item_price              0.054379
max_item_price              0.054878
total_outstanding_orders    0.273534
order_hour                  0.079336
total_available_dashers     0.101995
encoded_order_day           0.066091
dtype: float64
```

In [72]:

```
sorted_idx = regressor.feature_importances_.argsort()
plt.barh(X_test.columns[sorted_idx], regressor.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Out[72]:

```
Text(0.5, 0, 'Random Forest Feature Importance')
```



The most important features affecting delivery time in the decreasing order are: total outstanding orders, total cost of order, available dashers, market ID, time of order, weekday of order, order type protocol, maximum and minimum item price, store primary category and total items.

Benchmarking is an important aspect before choosing an artificial neural network model. They are computationally costly and their value lift has to be atleast significant to justify their use. Creating simpler models such as tree based models or even linear models may help in establishing a lower bound on accuracy.

Artificial Neural Network model

Trying different combinations and hyperparameters like learnming rate, number of layers, different optimizers to improve the model.

Why does a neural network perform well on a large dataset?

1. Non-Linearity: Neural networks can model complex, non-linear relationships between input features and output targets. Linear regression, on the other hand, assumes a linear relationship, which may not capture the intricate patterns present in large and complex datasets.
2. End-to-end learning. Neural networks can perform end-to-end learning, meaning they can learn both feature extraction and prediction in a unified manner. Linear regression separates feature extraction and prediction, which might limit its ability to optimize both aspects simultaneously.
3. Automatic feature learning: Neural networks learn complex patterns or high level features by incrementally learning simpler patterns by single neurons and refining performance on that by adding decisions of more and more neurons and layers. This eliminates the need of domain expertise and hard core feature extraction. Linear regression relies on manually selecting and engineering features, which can be challenging for large and complex datasets.

4. Parallel processing: Neural networks can take advantage of parallel processing capabilities, especially when training on GPUs or TPUs. This allows for faster training times on large datasets compared to linear regression.

Using Keras tuner for hyperparameter optimization

1. search the most appropriate optimizer
2. search the number of nodes in a layer
3. search the optimum no. of layers

1. Tuning the type of optimizer

In [75]:

```
# 1. The first simplest model architecture consists of one input layer with 12 input dimensions,
# a hidden layer with 32 neurons and ReLU activation,
# and an output layer with one neuron and linear activation.
# This model is just for optimizer hyperparamter tuning.

# hyperparameter search space includes the choice of optimizer among 'adam', 'sgd', 'rmsprop', ar
# The objective for tuning is set to minimize the validation mean absolute error.

# Define model
from tensorflow import keras
from tensorflow.keras import layers
from keras import Sequential
from keras.layers import Dense
from kerastuner.tuners import RandomSearch

#  Use hp (hyperparameter) from Keras Tuner to define the search space for hyperparameters.
def build_model(hp):
    model = Sequential()
    model.add(Dense(32,activation='relu',input_dim=11))
    model.add(Dense(1,activation='linear'))
    optimizer = hp.Choice('optimizer',['adam','sgd','rmsprop','adagrad']) # keras.optimizers.SGD
    model.compile(optimizer=optimizer,
                  loss='mean_squared_error',
                  metrics=['mean_absolute_error'])
    return model
```

In [76]:

```
# Define the search space
tuner = RandomSearch(
    build_model,
    objective='val_mean_absolute_error',  # The metric to optimize
    max_trials=5,  # Total number of trials (models to test)
    directory='validation_error_tuning_results',  # Directory to store tuning results
    project_name='my_tuning_project'   # Name of the tuning project
)
tuner.search(X_train, y_train, epochs=5, validation_data=(X_val, y_val))
```

```
Trial 4 Complete [00h 00m 49s]
val_mean_absolute_error: 26.955589294433594

Best val_mean_absolute_error So Far: 5.9777021408081055
Total elapsed time: 00h 02m 05s
```

In [77]:

```
tuner.get_best_hyperparameters()[0].values
```

Out[77]: {'optimizer': 'rmsprop'}

Briefly explain your choice of optimizer.

The tuner selected RMSProp (Root Mean Square Propagation) as the best optimizer. It adapts the learning rates individually for each parameter. This adaptability is beneficial for avoiding slow convergence in dimensions with small gradients and overshooting in dimensions with large gradients. It introduces a dampening term to the running average of squared gradients, which helps to prevent the learning rates from becoming too small over time. This can be useful for avoiding getting stuck in local minima.

2. Tuning number of nodes in layer

In [84]:
```
# 2. hyperparameter search space includes the range of numbers of nodes in hidden layer and the c
from keras.layers import LeakyReLU
# alpha in leaky ReLU is the slope of the negative values.

def build_model_2(hp):
    model = Sequential()
    units = hp.Int('units',64,1024,step=64)
    activation = hp.Choice('hidden_activation',['relu','tanh','leaky_relu'])
    if activation == 'leaky_relu':
        model.add(Dense(units=units, activation=LeakyReLU(alpha=0.01), input_dim=11))
    else:
        model.add(Dense(units=units, activation=activation, input_dim=11))

    model.add(Dense(units=1,activation='linear'))
    model.compile(optimizer='rmsprop',
                  loss='mean_squared_error',
                  metrics=['mean_absolute_error'])
    return model
```

In [85]:
```
# Define the search space
tuner = RandomSearch(
    build_model_2,
    objective='val_mean_absolute_error',  # The metric to optimize
    max_trials=5 # Total number of trials (models to test)
    )
tuner.search(X_train, y_train, epochs=5, validation_data=(X_val, y_val))
```

```
Trial 5 Complete [00h 00m 51s]
val_mean_absolute_error: 5.895590782165527

Best val_mean_absolute_error So Far: 4.9125752449035645
Total elapsed time: 00h 04m 08s
```

In [86]:
```
tuner.get_best_hyperparameters()[0].values
```

Out[86]: {'units': 960, 'hidden_activation': 'relu'}

This shows that 960 nodes are the best number of nodes in a layer compared to others in the range [64,128,256,...,1024].

In [89]:
```
import shutil

# Replace 'validation_error_tuning_results' with your actual directory name
# directory_to_delete = 'validation_error_tuning_results'
directory_to_delete = 'untitled_project'
```

```
# Use shutil.rmtree to delete the entire directory
shutil.rmtree(directory_to_delete, ignore_errors=True)
```

Which activation function did you use and why?

The Rectified Linear Unit (ReLU) activation function was used for regression where the function doesn't do anything to the positive weighted sum of the input, it simply cuts out negative values. Since it is linear it is a much faster function than others.

3. Tuning number of hidden layers

In [88]:
```
# 3. Hyperparameter search space includes the range of number of hidden layers

def build_model_3(hp):
    model = Sequential()
    model.add(Dense(256,activation='relu',input_dim=11))
    for i in range(hp.Int('num_layers',min_value=1,max_value=2)):
        model.add(Dense(960,activation='relu'))
    model.add(Dense(256,activation='relu'))
    model.add(Dense(1,activation='linear'))
    model.compile(optimizer= 'rmsprop',   #keras.optimizers.SGD(learning_rate=1e-4),
                  loss='mean_squared_error',
                  metrics=['mean_absolute_error'])
    return model
```

In [90]:
```
# Define the search space
tuner = RandomSearch(
    build_model_3,
    objective='val_mean_absolute_error',  # The metric to optimize
    max_trials=3, # Total number of trials (models to test)
    )
tuner.search(X_train, y_train, epochs=5, validation_data=(X_val, y_val))
```

```
Trial 2 Complete [00h 06m 21s]
val_mean_absolute_error: 4.868793487548828

Best val_mean_absolute_error So Far: 4.746240139007568
Total elapsed time: 00h 09m 03s
```

In [111…
```
# np.isinf(X_train).any(), np.isinf(y_train).any()
```

Out[111]:
```
(market_id                                    False
 store_primary_category                       False
 order_protocol                               False
 total_items                                  False
 subtotal                                     False
 min_item_price                               False
 max_item_price                               False
 total_outstanding_orders                     False
 estimated_store_to_consumer_driving_duration  False
 order_hour                                   False
 total_available_dashers                      False
 encoded_order_day                            False
 dtype: bool,
 False)
```

In [91]:
```
tuner.get_best_hyperparameters()[0].values
```

Out[91]: {'num_layers': 1}

1 layer of 960 nodes is the best.

In [93]:
```
import shutil

# Replace 'validation_error_tuning_results' with your actual directory name
# directory_to_delete = 'validation_error_tuning_results'
directory_to_delete = 'untitled_project'

# Use shutil.rmtree to delete the entire directory
shutil.rmtree(directory_to_delete, ignore_errors=True)
```

Fitting the best model

In [129…
```
from tensorflow.keras.layers import Dropout, BatchNormalization
from tensorflow.keras import regularizers
from tensorflow.keras.callbacks import EarlyStopping, LearningRateScheduler, ModelCheckpoint

# Define a simple model
def build_model():
    model = Sequential()
    model.add(Dense(64, activation='relu', input_dim=11))    #input layer

    # Hidden layers with dropout, L2 regularization, and batch normalization
    model.add(Dense(512, activation='relu')) #, kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    model.add(Dense(960, activation='relu')) #, kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    model.add(Dense(256, activation='relu')) #, kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    model.add(Dense(64, activation='relu')) #, kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())

    model.add(Dense(1, activation='linear')) # Output layer
    return model

# Create the model
model = build_model()

# Define a learning rate schedule callback (you can adjust the learning rates as needed)
def lr_schedule(epoch):
    if epoch < 10:
        return 0.001
    elif epoch < 20:
        return 0.0001
    else:
        return 0.00001
lr_callback = LearningRateScheduler(lr_schedule)

# Define a ModelCheckpoint callback to save the best model during training
checkpoint = ModelCheckpoint('model_checkpoint.h5', save_best_only=True)
```

```python
# Compile the model with an optimizer
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mean_absolute_error'])

# Define the Early Stopping callback
early_stopping = EarlyStopping(
    monitor='val_mean_squared_loss',    # Monitor validation loss
    patience=5,             # Number of epochs with no improvement after which training will be s
    restore_best_weights=True  # Restore model weights from the epoch with the best value of the
)
 # fitting on training data
history = model.fit(
    X_train, y_train,
    batch_size=512,
    epochs=30,  # You can set a large number of epochs
    initial_epoch=0,
    validation_data=(X_val, y_val),
    callbacks=[early_stopping,lr_callback,checkpoint],  # Pass the Early Stopping, Learning rate
    verbose=1)
```

```
Epoch 1/30
244/245 [============================>.] - ETA: 0s - loss: 1880.1332 - mean_absolute_error: 42.7
223WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not
available. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 18s 62ms/step - loss: 1879.0121 - mean_absolute_erro
r: 42.7085 - val_loss: 1828.0779 - val_mean_absolute_error: 41.8437 - lr: 0.0010
Epoch 2/30
  2/245 [..............................] - ETA: 12s - loss: 1337.2246 - mean_absolute_error: 36.
0024
```

```
C:\Users\Admin\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\engine\traini
ng.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file
format is considered legacy. We recommend using instead the native Keras format, e.g. `model.sav
e('my_model.keras')`.
  saving_api.save_model(
```

```
244/245 [============================>.] - ETA: 0s - loss: 765.6693 - mean_absolute_error: 26.12
49WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not a
vailable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 18s 72ms/step - loss: 764.5119 - mean_absolute_error:
26.0992 - val_loss: 383.9266 - val_mean_absolute_error: 18.1964 - lr: 0.0010
Epoch 3/30
244/245 [============================>.] - ETA: 0s - loss: 120.4156 - mean_absolute_error: 8.886
3WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not av
ailable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 61ms/step - loss: 120.2525 - mean_absolute_error:
8.8781 - val_loss: 60.7926 - val_mean_absolute_error: 6.0867 - lr: 0.0010
Epoch 4/30
244/245 [============================>.] - ETA: 0s - loss: 44.0052 - mean_absolute_error: 5.1376
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 60ms/step - loss: 43.9864 - mean_absolute_error:
5.1368 - val_loss: 37.1950 - val_mean_absolute_error: 4.7500 - lr: 0.0010
Epoch 5/30
244/245 [============================>.] - ETA: 0s - loss: 40.8837 - mean_absolute_error: 4.9920
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 17s 68ms/step - loss: 40.8953 - mean_absolute_error:
4.9928 - val_loss: 37.5272 - val_mean_absolute_error: 4.7789 - lr: 0.0010
Epoch 6/30
244/245 [============================>.] - ETA: 0s - loss: 39.3252 - mean_absolute_error: 4.8911
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 67ms/step - loss: 39.3385 - mean_absolute_error:
4.8917 - val_loss: 36.1028 - val_mean_absolute_error: 4.6809 - lr: 0.0010
Epoch 7/30
244/245 [============================>.] - ETA: 0s - loss: 38.2887 - mean_absolute_error: 4.8345
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 64ms/step - loss: 38.2940 - mean_absolute_error:
4.8347 - val_loss: 36.4255 - val_mean_absolute_error: 4.7542 - lr: 0.0010
Epoch 8/30
244/245 [============================>.] - ETA: 0s - loss: 37.7871 - mean_absolute_error: 4.7985
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 66ms/step - loss: 37.7895 - mean_absolute_error:
4.7985 - val_loss: 36.0306 - val_mean_absolute_error: 4.6775 - lr: 0.0010
Epoch 9/30
244/245 [============================>.] - ETA: 0s - loss: 37.2553 - mean_absolute_error: 4.7588
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 14s 57ms/step - loss: 37.2538 - mean_absolute_error:
4.7588 - val_loss: 36.4316 - val_mean_absolute_error: 4.6705 - lr: 0.0010
Epoch 10/30
244/245 [============================>.] - ETA: 0s - loss: 36.7418 - mean_absolute_error: 4.7294
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 67ms/step - loss: 36.7368 - mean_absolute_error:
4.7291 - val_loss: 36.4222 - val_mean_absolute_error: 4.6802 - lr: 0.0010
Epoch 11/30
245/245 [==============================] - ETA: 0s - loss: 35.8340 - mean_absolute_error: 4.6707
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 65ms/step - loss: 35.8340 - mean_absolute_error:
4.6707 - val_loss: 34.2056 - val_mean_absolute_error: 4.5563 - lr: 1.0000e-04
Epoch 12/30
244/245 [============================>.] - ETA: 0s - loss: 35.4645 - mean_absolute_error: 4.6486
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
```

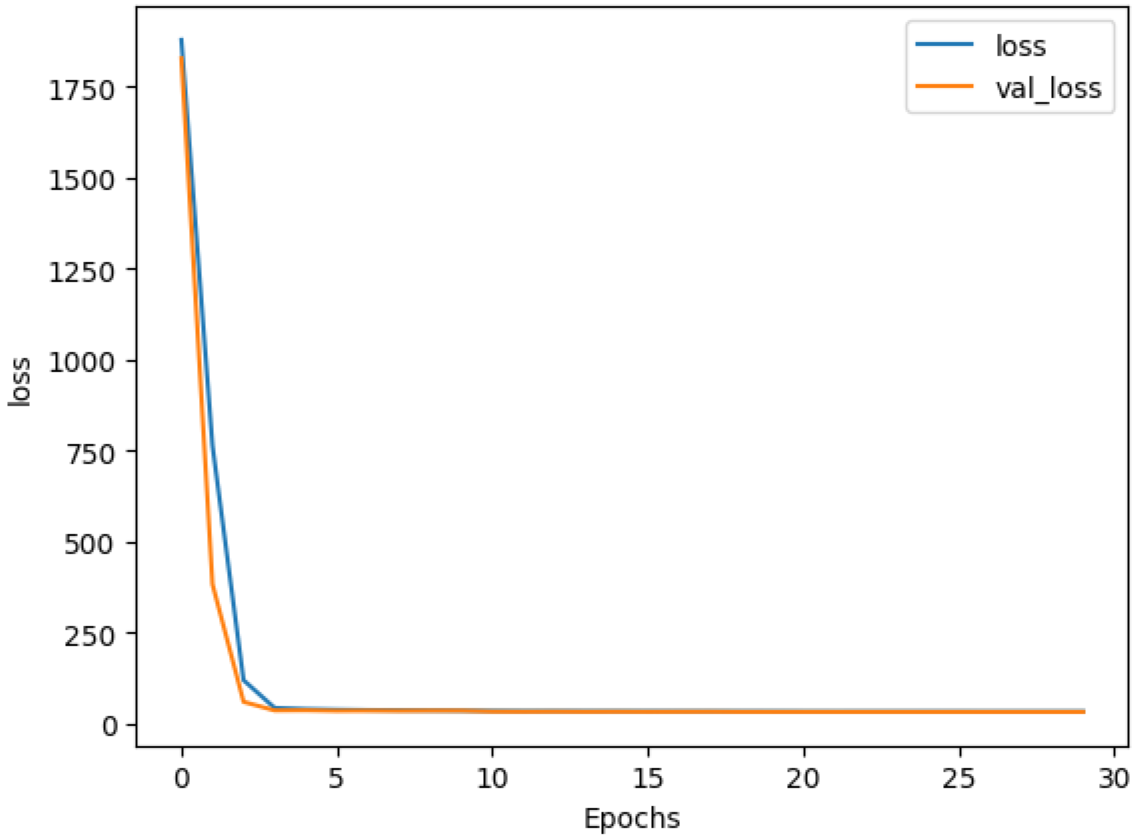
```
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 65ms/step - loss: 35.4601 - mean_absolute_error:
4.6489 - val_loss: 33.8870 - val_mean_absolute_error: 4.5342 - lr: 1.0000e-04
Epoch 13/30
244/245 [============================>.] - ETA: 0s - loss: 35.4189 - mean_absolute_error: 4.6411
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 64ms/step - loss: 35.4219 - mean_absolute_error:
4.6413 - val_loss: 34.1059 - val_mean_absolute_error: 4.5505 - lr: 1.0000e-04
Epoch 14/30
244/245 [============================>.] - ETA: 0s - loss: 35.2656 - mean_absolute_error: 4.6302
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 65ms/step - loss: 35.2634 - mean_absolute_error:
4.6300 - val_loss: 33.7507 - val_mean_absolute_error: 4.5255 - lr: 1.0000e-04
Epoch 15/30
244/245 [============================>.] - ETA: 0s - loss: 35.1656 - mean_absolute_error: 4.6276
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 64ms/step - loss: 35.1869 - mean_absolute_error:
4.6289 - val_loss: 33.9492 - val_mean_absolute_error: 4.5432 - lr: 1.0000e-04
Epoch 16/30
244/245 [============================>.] - ETA: 0s - loss: 35.0917 - mean_absolute_error: 4.6208
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 64ms/step - loss: 35.0884 - mean_absolute_error:
4.6206 - val_loss: 33.6464 - val_mean_absolute_error: 4.5154 - lr: 1.0000e-04
Epoch 17/30
244/245 [============================>.] - ETA: 0s - loss: 35.0048 - mean_absolute_error: 4.6129
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 17s 69ms/step - loss: 34.9995 - mean_absolute_error:
4.6125 - val_loss: 33.7786 - val_mean_absolute_error: 4.5264 - lr: 1.0000e-04
Epoch 18/30
244/245 [============================>.] - ETA: 0s - loss: 34.9425 - mean_absolute_error: 4.6122
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 61ms/step - loss: 34.9399 - mean_absolute_error:
4.6121 - val_loss: 33.9207 - val_mean_absolute_error: 4.5290 - lr: 1.0000e-04
Epoch 19/30
244/245 [============================>.] - ETA: 0s - loss: 34.8295 - mean_absolute_error: 4.6015
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 61ms/step - loss: 34.8383 - mean_absolute_error:
4.6023 - val_loss: 33.4975 - val_mean_absolute_error: 4.5210 - lr: 1.0000e-04
Epoch 20/30
244/245 [============================>.] - ETA: 0s - loss: 34.7756 - mean_absolute_error: 4.5980
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 66ms/step - loss: 34.7710 - mean_absolute_error:
4.5977 - val_loss: 33.4496 - val_mean_absolute_error: 4.5053 - lr: 1.0000e-04
Epoch 21/30
244/245 [============================>.] - ETA: 0s - loss: 34.5943 - mean_absolute_error: 4.5897
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 61ms/step - loss: 34.5963 - mean_absolute_error:
4.5899 - val_loss: 33.4302 - val_mean_absolute_error: 4.4974 - lr: 1.0000e-05
Epoch 22/30
244/245 [============================>.] - ETA: 0s - loss: 34.4977 - mean_absolute_error: 4.5817
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 61ms/step - loss: 34.4950 - mean_absolute_error:
```

```
4.5820 - val_loss: 33.3941 - val_mean_absolute_error: 4.4941 - lr: 1.0000e-05
Epoch 23/30
244/245 [============================>.] - ETA: 0s - loss: 34.5652 - mean_absolute_error: 4.5845
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 64ms/step - loss: 34.5462 - mean_absolute_error:
4.5835 - val_loss: 33.3753 - val_mean_absolute_error: 4.4962 - lr: 1.0000e-05
Epoch 24/30
244/245 [============================>.] - ETA: 0s - loss: 34.5112 - mean_absolute_error: 4.5834
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 64ms/step - loss: 34.5040 - mean_absolute_error:
4.5834 - val_loss: 33.3951 - val_mean_absolute_error: 4.4949 - lr: 1.0000e-05
Epoch 25/30
244/245 [============================>.] - ETA: 0s - loss: 34.5507 - mean_absolute_error: 4.5762
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 63ms/step - loss: 34.5627 - mean_absolute_error:
4.5769 - val_loss: 33.3407 - val_mean_absolute_error: 4.4925 - lr: 1.0000e-05
Epoch 26/30
244/245 [============================>.] - ETA: 0s - loss: 34.5355 - mean_absolute_error: 4.5865
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 16s 63ms/step - loss: 34.5291 - mean_absolute_error:
4.5862 - val_loss: 33.3728 - val_mean_absolute_error: 4.4923 - lr: 1.0000e-05
Epoch 27/30
244/245 [============================>.] - ETA: 0s - loss: 34.4139 - mean_absolute_error: 4.5772
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 63ms/step - loss: 34.4290 - mean_absolute_error:
4.5779 - val_loss: 33.3290 - val_mean_absolute_error: 4.4924 - lr: 1.0000e-05
Epoch 28/30
244/245 [============================>.] - ETA: 0s - loss: 34.4942 - mean_absolute_error: 4.5824
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 63ms/step - loss: 34.4846 - mean_absolute_error:
4.5817 - val_loss: 33.3375 - val_mean_absolute_error: 4.4914 - lr: 1.0000e-05
Epoch 29/30
244/245 [============================>.] - ETA: 0s - loss: 34.4764 - mean_absolute_error: 4.5798
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 63ms/step - loss: 34.4837 - mean_absolute_error:
4.5805 - val_loss: 33.3254 - val_mean_absolute_error: 4.4915 - lr: 1.0000e-05
Epoch 30/30
244/245 [============================>.] - ETA: 0s - loss: 34.4909 - mean_absolute_error: 4.5792
WARNING:tensorflow:Early stopping conditioned on metric `val_mean_squared_loss` which is not ava
ilable. Available metrics are: loss,mean_absolute_error,val_loss,val_mean_absolute_error
245/245 [==============================] - 15s 63ms/step - loss: 34.4820 - mean_absolute_error:
4.5788 - val_loss: 33.3244 - val_mean_absolute_error: 4.4916 - lr: 1.0000e-05
```

- Dropout layers with a dropout rate of 0.2 after each hidden layer.
- L2 regularization with a regularization strength of 0.01 on the kernel weights of the hidden layers.
- Batch normalization after each hidden layer.
- A learning rate schedule callback using LearningRateScheduler for dynamic learning rate adjustments during training.
- The stochastic nature of mini-batch training introduces randomness into the optimization process. This can help the model generalize better to unseen data by preventing it from getting stuck in local minima.

In [130…

```python
def plot_history(history, key):
  plt.plot(history.history[key])
  plt.plot(history.history['val_'+key])
  plt.xlabel("Epochs")
  plt.ylabel(key)
  plt.legend([key, 'val_'+key])
  plt.show()
# Plot the history
plot_history(history, 'loss')
```



The results plateau after 5 epochs.

If the training loss is steadily decreasing but the validation loss is fluctuating, it might be a sign that the model is overfitting to the training data. Overfitting occurs when the model learns the training data too well, including its noise and outliers, making it less effective on new, unseen data.

Model testing

In [131…

```python
model.evaluate(X_test_scaled, y_test)
```

```
275/275 [==============================] - 1s 3ms/step - loss: 33.1457 - mean_absolute_error: 4.
4701
```

Out[131]:

```
[33.145687103271484, 4.4700775146484375]
```

In [132…

```python
z = model.predict(X_test_scaled)
r2_score(y_test, z)
```

```
275/275 [==============================] - 1s 3ms/step
```

Out[132]:

```
0.6117211107251497
```

In [133…

```python
y_test.shape, z.shape
```

Out[133]: ((8789,), (8789, 1))

In [134…
```
z = z.flatten()
def MAPE(Y_actual,Y_Predicted):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
print("mape : ",MAPE(y_test, z))
```

```
mape :  9.772735524091148
```

Using Random Forest, the Mean absolute percentage error was 9.7% and with neural network it is also 9.7%.

Inference:

The most important features affecting delivery time in the decreasing order are: total outstanding orders, total cost of order, available dashers, market ID, time of order, weekday of order, order type protocol, maximum and minimum item price, store primary category and total items.

In [ ]: