Porter is India's Largest Marketplace for Intra-City Logistics. Leader in the country's $40 billion intra-city logistics market, Porter strives to improve the lives of 1,50,000+ driver-partners by providing them with consistent earning & independence. Currently, the company has serviced 5+ million customers

Porter works with a wide range of restaurants for delivering their items directly to the people.

Porter 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

In [96]:
```
import pandas as pd
import numpy as np
import os

#for visualizing and analyzing it
import matplotlib.pyplot as plt
import seaborn as sns

#data preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

#random forest model training
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.ensemble import RandomForestRegressor

#Ann training
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LeakyReLU
from sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredLogarithmicError
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.losses import MeanAbsolutePercentageError


from tensorflow.keras.metrics import  mean_absolute_percentage_error
from tensorflow.keras.metrics import RootMeanSquaredError
from tensorflow.keras.metrics import MeanAbsoluteError
from tensorflow.keras.optimizers import SGD, Adam
```

In [65]:
```
!gdown 1WFa46c7_uSZ6GSzgYrDk4TKJd3y9zv8f
```

```
Downloading...
From: https://drive.google.com/uc?id=1WFa46c7_uSZ6GSzgYrDk4TKJd3y9zv8f (https://drive.google.com/uc?id=1WFa46c7_uSZ6GSzgYrDk4TKJd3y9zv8f)
To: C:\Users\bikim\data_2.csv

  0%|          | 0.00/15.7M [00:00<?, ?B/s]
  3%|3         | 524k/15.7M [00:00<00:15, 1.00MB/s]
  7%|6         | 1.05M/15.7M [00:00<00:10, 1.37MB/s]
 10%|#         | 1.57M/15.7M [00:01<00:09, 1.43MB/s]
 13%|#3        | 2.10M/15.7M [00:01<00:10, 1.25MB/s]
 17%|#6        | 2.62M/15.7M [00:02<00:10, 1.26MB/s]
 20%|##        | 3.15M/15.7M [00:02<00:10, 1.24MB/s]
 23%|##3       | 3.67M/15.7M [00:02<00:09, 1.23MB/s]
 27%|##6       | 4.19M/15.7M [00:03<00:10, 1.13MB/s]
 30%|###       | 4.72M/15.7M [00:03<00:08, 1.27MB/s]
 33%|###3      | 5.24M/15.7M [00:04<00:08, 1.30MB/s]
 37%|###6      | 5.77M/15.7M [00:04<00:06, 1.43MB/s]
 40%|####      | 6.29M/15.7M [00:04<00:06, 1.53MB/s]
 43%|####3     | 6.82M/15.7M [00:05<00:05, 1.59MB/s]
 47%|####6     | 7.34M/15.7M [00:05<00:05, 1.57MB/s]
 50%|#####     | 7.86M/15.7M [00:05<00:05, 1.53MB/s]
 53%|#####3    | 8.39M/15.7M [00:06<00:04, 1.64MB/s]
 57%|#####6    | 8.91M/15.7M [00:06<00:04, 1.47MB/s]
 60%|######    | 9.44M/15.7M [00:06<00:03, 1.59MB/s]
 63%|######3   | 9.96M/15.7M [00:07<00:03, 1.50MB/s]
 67%|######6   | 10.5M/15.7M [00:07<00:03, 1.47MB/s]
 70%|#######   | 11.0M/15.7M [00:08<00:03, 1.27MB/s]
 73%|#######3  | 11.5M/15.7M [00:08<00:03, 1.28MB/s]
 77%|#######6  | 12.1M/15.7M [00:08<00:02, 1.52MB/s]
 80%|########  | 12.6M/15.7M [00:08<00:01, 1.68MB/s]
 83%|########3 | 13.1M/15.7M [00:09<00:01, 1.81MB/s]
 87%|########6 | 13.6M/15.7M [00:09<00:01, 1.63MB/s]
 90%|######### | 14.2M/15.7M [00:09<00:00, 1.67MB/s]
 93%|#########3| 14.7M/15.7M [00:09<00:00, 1.99MB/s]
 97%|#########6| 15.2M/15.7M [00:10<00:00, 2.31MB/s]
100%|##########| 15.7M/15.7M [00:10<00:00, 2.54MB/s]
100%|##########| 15.7M/15.7M [00:10<00:00, 1.54MB/s]
```

In [66]:
```
df=pd.read_csv('data_2.csv')#, parse_dates=[1, 2]
df.head()
```

Out[66]:

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

In [67]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 14 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
 13  estimated_store_to_consumer_driving_duration  175777 non-null  float64
dtypes: float64(6), int64(6), object(2)
memory usage: 18.8+ MB
```

Feature Engineering

we have the time at which the order was placed and time at which it was delivired, so we will create a new column for time taken in delivery and that will be our target column

calculating time taken in delivery by subtracting the order timestamp from delivery timestamp

In [68]:
```
df['created_at'] = pd.to_datetime(df['created_at'])
df['actual_delivery_time'] = pd.to_datetime(df['actual_delivery_time'])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 14 columns):
 #   Column                                        Non-Null Count   Dtype         
---  ------                                        --------------   -----         
 0   market_id                                     175777 non-null  float64       
 1   created_at                                    175777 non-null  datetime64[ns]
 2   actual_delivery_time                          175777 non-null  datetime64[ns]
 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       
 13  estimated_store_to_consumer_driving_duration  175777 non-null  float64       
dtypes: datetime64[ns](2), float64(6), int64(6)
memory usage: 18.8 MB
```

In [69]:
```
df['time_taken'] = df['actual_delivery_time'] - df['created_at']
df.head()
```

Out[69]:

s	min_item_price	max_item_price	total_onshift_dashers	total_busy_dashers	total_outstanding_orders	estimated_store_to_consume
4	557	1239	33.0	14.0	21.0	
1	1400	1400	1.0	2.0	2.0	
3	820	1604	8.0	6.0	18.0	
1	1525	1525	5.0	6.0	8.0	
2	1425	2195	5.0	5.0	7.0	

In [70]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 15 columns):
 #   Column                                        Non-Null Count   Dtype
---  ------                                        --------------   -----
 0   market_id                                     175777 non-null  float64
 1   created_at                                    175777 non-null  datetime64[ns]
 2   actual_delivery_time                          175777 non-null  datetime64[ns]
 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
 13  estimated_store_to_consumer_driving_duration  175777 non-null  float64
 14  time_taken                                    175777 non-null  timedelta64[ns]
dtypes: datetime64[ns](2), float64(6), int64(6), timedelta64[ns](1)
memory usage: 20.1 MB
```

now that we have our time taken for the delivery we can convert it to minutes and that will be our target variable to train the models

the timedelta is a datatype that stores the time difference and it is better we convert it to float and converting to minute does that as well

In [71]:
```
df['time_taken_mins'] = pd.to_timedelta(df['time_taken'])/pd.Timedelta('60s')
df.head()
```

Out[71]:

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

we can also extract the hour at which the order was places and which day of week it was

In [72]:
```
df['hour'] = df['created_at'].dt.hour
df['day'] = df['created_at'].dt.dayofweek
df.head()
```

Out[72]:

price	total_onshift_dashers	total_busy_dashers	total_outstanding_orders	estimated_store_to_consumer_driving_duration	time_taken
1239	33.0	14.0	21.0	861.0	0 days 00:47:00
1400	1.0	2.0	2.0	690.0	0 days 00:44:00
1604	8.0	6.0	18.0	289.0	0 days 00:55:00
1525	5.0	6.0	8.0	795.0	0 days 00:59:00
2195	5.0	5.0	7.0	205.0	0 days 00:46:00

Dropping the columns that are no longer required

In [73]:
```
df.drop(['created_at','time_taken','actual_delivery_time'],axis = 1,inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 15 columns):
 #   Column                                        Non-Null Count   Dtype  
---  ------                                        --------------   -----  
 0   market_id                                     175777 non-null  float64
 1   store_primary_category                        175777 non-null  int64  
 2   order_protocol                                175777 non-null  float64
 3   total_items                                   175777 non-null  int64  
 4   subtotal                                      175777 non-null  int64  
 5   num_distinct_items                            175777 non-null  int64  
 6   min_item_price                                175777 non-null  int64  
 7   max_item_price                                175777 non-null  int64  
 8   total_onshift_dashers                         175777 non-null  float64
 9   total_busy_dashers                            175777 non-null  float64
 10  total_outstanding_orders                      175777 non-null  float64
 11  estimated_store_to_consumer_driving_duration  175777 non-null  float64
 12  time_taken_mins                               175777 non-null  float64
 13  hour                                          175777 non-null  int64  
 14  day                                           175777 non-null  int64  
dtypes: float64(7), int64(8)
memory usage: 20.1 MB
```

Checking null values in the data

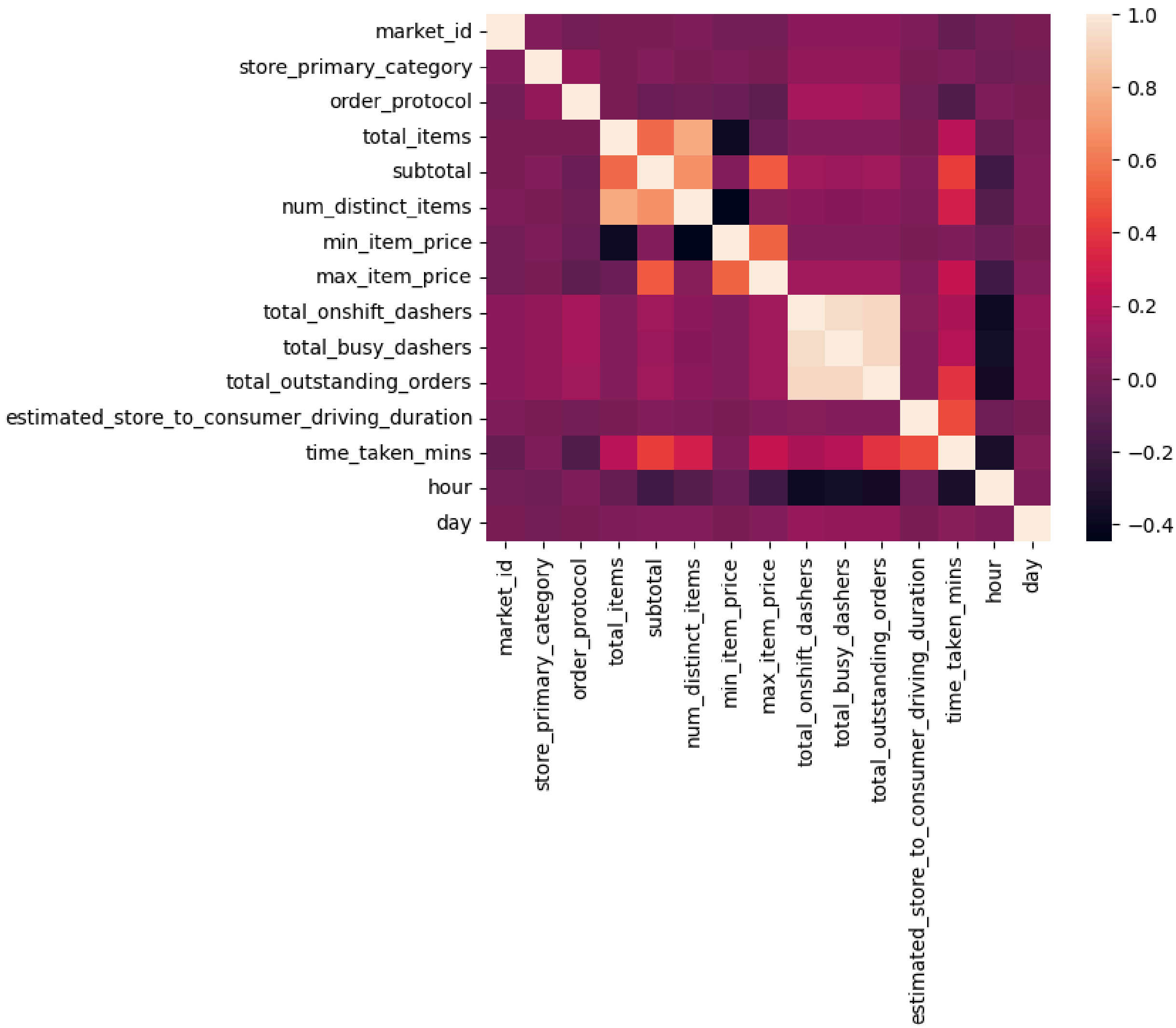
In [74]:
```
df.isna().sum()
```

Out[74]:
```
market_id                                       0
store_primary_category                          0
order_protocol                                  0
total_items                                     0
subtotal                                        0
num_distinct_items                              0
min_item_price                                  0
max_item_price                                  0
total_onshift_dashers                           0
total_busy_dashers                              0
total_outstanding_orders                        0
estimated_store_to_consumer_driving_duration    0
time_taken_mins                                 0
hour                                            0
day                                             0
dtype: int64
```

Plotting correlation to get an idea of the data

In [75]:
```
sns.heatmap(df.corr())
```

Out[75]: <Axes: >



we have one categoriacal column which we will change to integer for model

In [76]:
```
df['store_primary_category'] = df['store_primary_category'].astype('category').cat.codes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 15 columns):
 #   Column                                        Non-Null Count   Dtype  
---  ------                                        --------------   -----  
 0   market_id                                     175777 non-null  float64
 1   store_primary_category                        175777 non-null  int8   
 2   order_protocol                                175777 non-null  float64
 3   total_items                                   175777 non-null  int64  
 4   subtotal                                      175777 non-null  int64  
 5   num_distinct_items                            175777 non-null  int64  
 6   min_item_price                                175777 non-null  int64  
 7   max_item_price                                175777 non-null  int64  
 8   total_onshift_dashers                         175777 non-null  float64
 9   total_busy_dashers                            175777 non-null  float64
 10  total_outstanding_orders                      175777 non-null  float64
 11  estimated_store_to_consumer_driving_duration  175777 non-null  float64
 12  time_taken_mins                               175777 non-null  float64
 13  hour                                          175777 non-null  int64  
 14  day                                           175777 non-null  int64  
dtypes: float64(7), int64(7), int8(1)
memory usage: 18.9 MB
```

In [77]:
```
df.head()
```

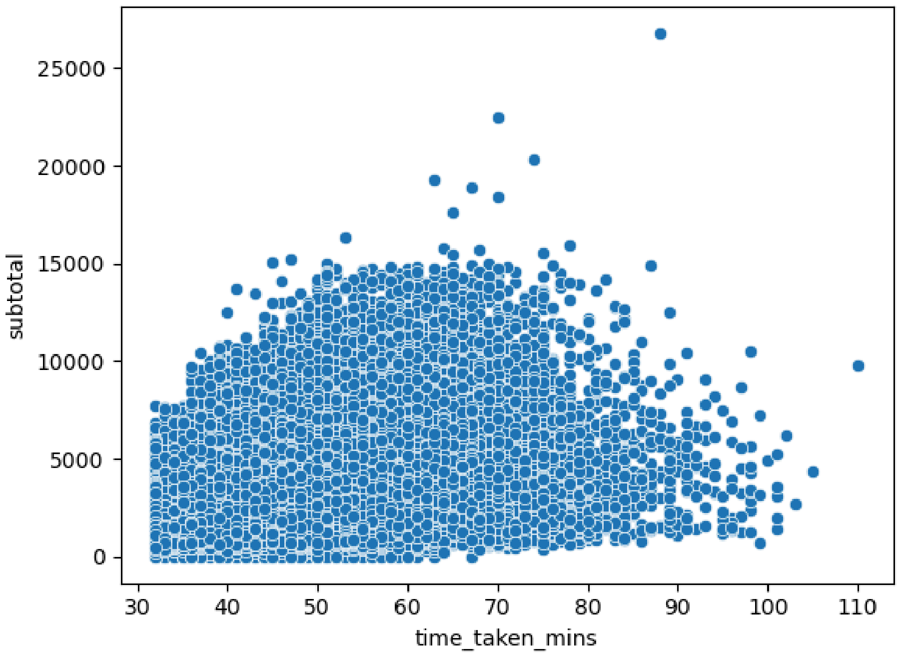
Out[77]:

max_item_price	total_onshift_dashers	total_busy_dashers	total_outstanding_orders	estimated_store_to_consumer_driving_duration
1239	33.0	14.0	21.0	861.0
1400	1.0	2.0	2.0	690.0
1604	8.0	6.0	18.0	289.0
1525	5.0	6.0	8.0	795.0
2195	5.0	5.0	7.0	205.0

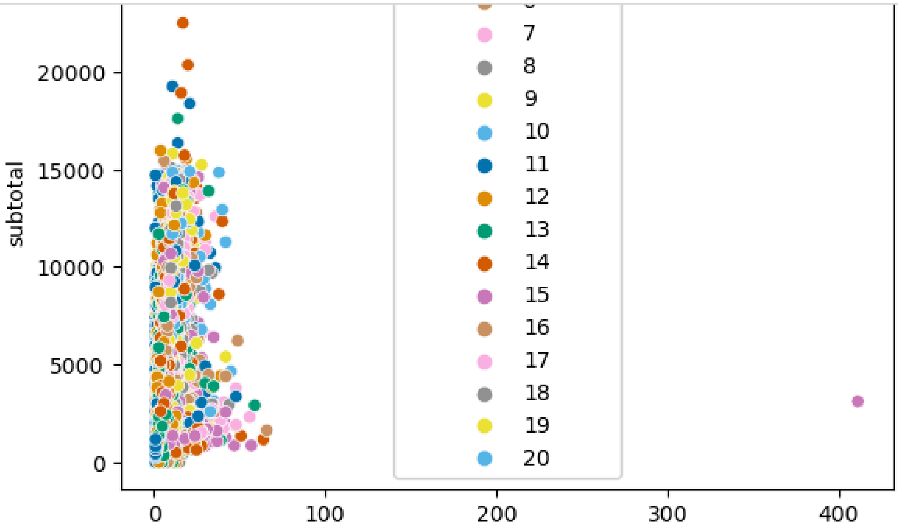
Data visualization and cleaning

In [78]:
```
sns.scatterplot(x='time_taken_mins', y='subtotal', data=df)
```

Out[78]: <Axes: xlabel='time_taken_mins', ylabel='subtotal'>



In [79]:
```
sns.scatterplot(x= 'total_items', y= 'subtotal',hue = 'num_distinct_items',palette='colorblind', data=df
```



In [80]:
```
from sklearn.neighbors import LocalOutlierFactor
import matplotlib.pyplot as plt
model1 = LocalOutlierFactor()
df['lof_anomaly_score'] = model1.fit_predict(df)
```

In [81]:
```
df.head()
```

Out[81]:

otal_onshift_dashers	total_busy_dashers	total_outstanding_orders	estimated_store_to_consumer_driving_duration	time_taken_mins
33.0	14.0	21.0	861.0	47.0
1.0	2.0	2.0	690.0	44.0
8.0	6.0	18.0	289.0	55.0
5.0	6.0	8.0	795.0	59.0
5.0	5.0	7.0	205.0	46.0

In [82]:
```
print("number of outliers :", (len(df.loc[(df['lof_anomaly_score'] == -1)])))
df = df.loc[(df['lof_anomaly_score'] == 1)]
```

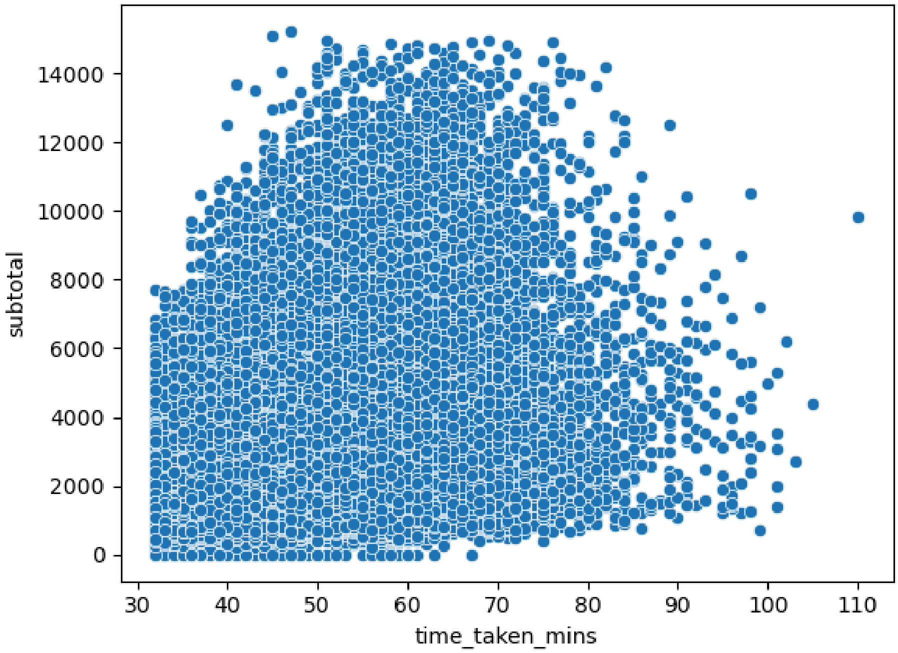
```
number of outliers : 831
```

In [83]:
```
df.drop(['lof_anomaly_score'], axis = 1, inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 174946 entries, 0 to 175776
Data columns (total 15 columns):
 #   Column                                        Non-Null Count   Dtype  
---  ------                                        --------------   -----  
 0   market_id                                     174946 non-null  float64
 1   store_primary_category                        174946 non-null  int8   
 2   order_protocol                                174946 non-null  float64
 3   total_items                                   174946 non-null  int64  
 4   subtotal                                      174946 non-null  int64  
 5   num_distinct_items                            174946 non-null  int64  
 6   min_item_price                                174946 non-null  int64  
 7   max_item_price                                174946 non-null  int64  
 8   total_onshift_dashers                         174946 non-null  float64
 9   total_busy_dashers                            174946 non-null  float64
 10  total_outstanding_orders                      174946 non-null  float64
 11  estimated_store_to_consumer_driving_duration  174946 non-null  float64
 12  time_taken_mins                               174946 non-null  float64
 13  hour                                          174946 non-null  int64  
 14  day                                           174946 non-null  int64  
dtypes: float64(7), int64(7), int8(1)
memory usage: 20.2 MB
```

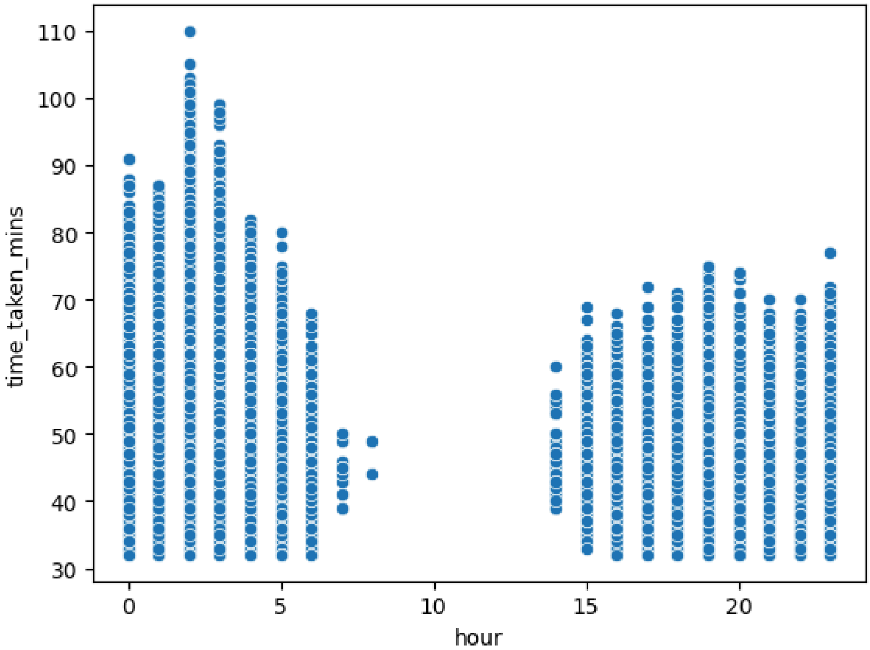
In [84]:
```
sns.scatterplot(x='time_taken_mins' , y= 'subtotal',data = df)
```

Out[84]: <Axes: xlabel='time_taken_mins', ylabel='subtotal'>



In [88]:
```
sns.scatterplot(x = 'hour', y = 'time_taken_mins', data =df)
```

Out[88]: <Axes: xlabel='hour', ylabel='time_taken_mins'>



Data Spliting and modeling

In [91]:
```
y = df['time_taken_mins']
x = df.drop(['time_taken_mins'], axis=1)
df.drop(['time_taken_mins'], axis=1, inplace=True)
X_train,X_test,y_train,y_test = train_test_split(x,y, test_size = 0.2, random_state = 42)
```

In [92]:
```
x.head()
```

Out[92]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price
0	1.0	4	1.0	4	3441	4	557	1239
1	2.0	46	2.0	1	1900	1	1400	1400
2	2.0	36	3.0	4	4771	3	820	1604
3	1.0	38	1.0	1	1525	1	1525	1525
4	1.0	38	1.0	2	3620	2	1425	2195

Neural networks

In [93]:
```
from sklearn import preprocessing
scaler = preprocessing.MinMaxScaler()
x_scaled = scaler.fit_transform(x)
X_train,X_test,y_train,y_test = train_test_split(x_scaled,y, test_size = 0.2, random_state = 42)
```

In [98]:
```
model = Sequential()
model.add(Dense(14, kernel_initializer = 'normal', activation = 'relu'))
model.add(Dense(512,  activation = 'relu'))
model.add(Dense(1024, activation = 'relu'))
model.add(Dense(256,  activation = 'relu'))
model.add(Dense(1,  activation = 'linear'))
```

```
WARNING:tensorflow:From D:\Anaconda\lib\site-packages\keras\src\backend.py:873: The name tf.get_default
_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

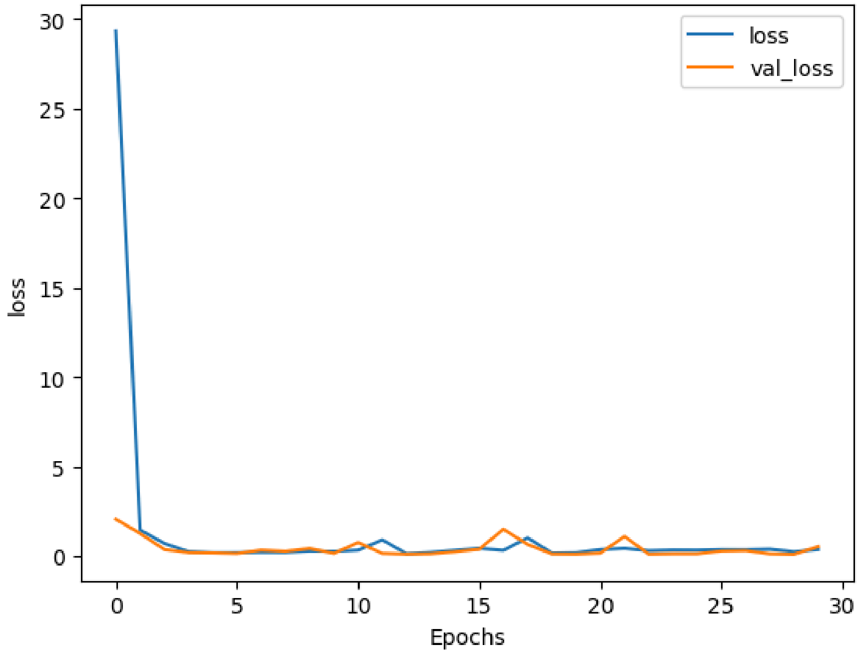
In [101]:
```
from tensorflow.keras.optimizers import Adam
adam = Adam(learning_rate=0.01)
model.compile(loss = 'mse', optimizer = adam, metrics = ['mse','mae'])
history = model.fit(X_train, y_train, epochs = 30, batch_size = 512, verbose=1, validation_split=0.2)
```

```
Epoch 1/30
219/219 [==============================] - 10s 31ms/step - loss: 29.3488 - mse: 29.3488 - mae: 2.9796 -
val_loss: 2.0558 - val_mse: 2.0558 - val_mae: 0.9578
Epoch 2/30
219/219 [==============================] - 7s 32ms/step - loss: 1.4569 - mse: 1.4569 - mae: 0.8309 - va
l_loss: 1.2616 - val_mse: 1.2616 - val_mae: 0.8158
Epoch 3/30
219/219 [==============================] - 7s 30ms/step - loss: 0.7017 - mse: 0.7017 - mae: 0.5965 - va
l_loss: 0.3646 - val_mse: 0.3646 - val_mae: 0.4544
Epoch 4/30
219/219 [==============================] - 7s 33ms/step - loss: 0.2446 - mse: 0.2446 - mae: 0.3865 - va
l_loss: 0.1922 - val_mse: 0.1922 - val_mae: 0.3467
Epoch 5/30
219/219 [==============================] - 7s 32ms/step - loss: 0.1863 - mse: 0.1863 - mae: 0.3430 - va
l_loss: 0.1746 - val_mse: 0.1746 - val_mae: 0.3343
Epoch 6/30
219/219 [==============================] - 8s 38ms/step - loss: 0.1888 - mse: 0.1888 - mae: 0.3459 - va
l_loss: 0.1433 - val_mse: 0.1433 - val_mae: 0.3033
Epoch 7/30
219/219 [==============================] - 8s 36ms/step - loss: 0.2053 - mse: 0.2053 - mae: 0.3595 - va
l_loss: 0.3286 - val_mse: 0.3286 - val_mae: 0.4756
Epoch 8/30
219/219 [==============================] - 7s 32ms/step - loss: 0.2012 - mse: 0.2012 - mae: 0.3577 - va
l_loss: 0.2629 - val_mse: 0.2629 - val_mae: 0.4117
Epoch 9/30
219/219 [==============================] - 7s 33ms/step - loss: 0.2726 - mse: 0.2726 - mae: 0.4099 - va
l_loss: 0.4255 - val_mse: 0.4255 - val_mae: 0.5509
Epoch 10/30
219/219 [==============================] - 7s 33ms/step - loss: 0.2562 - mse: 0.2562 - mae: 0.3980 - va
l_loss: 0.1476 - val_mse: 0.1476 - val_mae: 0.3108
Epoch 11/30
219/219 [==============================] - 7s 32ms/step - loss: 0.3276 - mse: 0.3276 - mae: 0.4484 - va
l_loss: 0.7515 - val_mse: 0.7515 - val_mae: 0.7610
Epoch 12/30
219/219 [==============================] - 7s 32ms/step - loss: 0.9020 - mse: 0.9020 - mae: 0.6376 - va
l_loss: 0.1448 - val_mse: 0.1448 - val_mae: 0.3057
Epoch 13/30
219/219 [==============================] - 7s 32ms/step - loss: 0.1263 - mse: 0.1263 - mae: 0.2905 - va
l_loss: 0.1039 - val_mse: 0.1039 - val_mae: 0.2683
Epoch 14/30
219/219 [==============================] - 7s 32ms/step - loss: 0.2076 - mse: 0.2076 - mae: 0.3656 - va
l_loss: 0.1247 - val_mse: 0.1247 - val_mae: 0.2892
Epoch 15/30
219/219 [==============================] - 7s 33ms/step - loss: 0.3204 - mse: 0.3204 - mae: 0.4412 - va
l_loss: 0.2363 - val_mse: 0.2363 - val_mae: 0.3997
Epoch 16/30
219/219 [==============================] - 7s 32ms/step - loss: 0.4320 - mse: 0.4320 - mae: 0.5258 - va
l_loss: 0.3829 - val_mse: 0.3829 - val_mae: 0.5351
Epoch 17/30
219/219 [==============================] - 7s 34ms/step - loss: 0.3201 - mse: 0.3201 - mae: 0.4465 - va
l_loss: 1.4968 - val_mse: 1.4968 - val_mae: 1.1258
Epoch 18/30
219/219 [==============================] - 7s 31ms/step - loss: 1.0169 - mse: 1.0169 - mae: 0.7130 - va
l_loss: 0.6498 - val_mse: 0.6498 - val_mae: 0.6497
Epoch 19/30
219/219 [==============================] - 7s 32ms/step - loss: 0.1679 - mse: 0.1679 - mae: 0.3276 - va
l_loss: 0.1162 - val_mse: 0.1162 - val_mae: 0.2806
Epoch 20/30
219/219 [==============================] - 7s 32ms/step - loss: 0.1867 - mse: 0.1867 - mae: 0.3444 - va
l_loss: 0.1048 - val_mse: 0.1048 - val_mae: 0.2693
Epoch 21/30
219/219 [==============================] - 7s 32ms/step - loss: 0.3623 - mse: 0.3623 - mae: 0.4519 - va
l_loss: 0.1631 - val_mse: 0.1631 - val_mae: 0.3269
Epoch 22/30
219/219 [==============================] - 7s 32ms/step - loss: 0.4299 - mse: 0.4299 - mae: 0.4712 - va
l_loss: 1.0950 - val_mse: 1.0950 - val_mae: 0.8902
Epoch 23/30
219/219 [==============================] - 7s 32ms/step - loss: 0.2988 - mse: 0.2988 - mae: 0.4061 - va
l_loss: 0.1048 - val_mse: 0.1048 - val_mae: 0.2685
Epoch 24/30
219/219 [==============================] - 7s 32ms/step - loss: 0.3336 - mse: 0.3336 - mae: 0.4565 - va
l_loss: 0.1239 - val_mse: 0.1239 - val_mae: 0.2875
Epoch 25/30
219/219 [==============================] - 7s 32ms/step - loss: 0.3280 - mse: 0.3280 - mae: 0.4464 - va
l_loss: 0.1245 - val_mse: 0.1245 - val_mae: 0.2889
Epoch 26/30
219/219 [==============================] - 8s 35ms/step - loss: 0.3604 - mse: 0.3604 - mae: 0.4655 - va
l_loss: 0.2743 - val_mse: 0.2743 - val_mae: 0.4310
```

```
Epoch 27/30
219/219 [==============================] - 7s 33ms/step - loss: 0.3570 - mse: 0.3570 - mae: 0.4666 - va
l_loss: 0.2981 - val_mse: 0.2981 - val_mae: 0.4582
Epoch 28/30
219/219 [==============================] - 7s 33ms/step - loss: 0.3854 - mse: 0.3854 - mae: 0.4506 - va
l_loss: 0.1207 - val_mse: 0.1207 - val_mae: 0.2848
Epoch 29/30
219/219 [==============================] - 7s 32ms/step - loss: 0.2375 - mse: 0.2375 - mae: 0.3864 - va
l_loss: 0.0960 - val_mse: 0.0960 - val_mae: 0.2592
Epoch 30/30
219/219 [==============================] - 7s 32ms/step - loss: 0.3672 - mse: 0.3672 - mae: 0.4638 - va
l_loss: 0.5284 - val_mse: 0.5284 - val_mae: 0.6343
```

we plot train and validation loss throughout training

In [102]:
```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')
```



val loss is below train loss so model is not overfitting

In [103]:
```python
z = model.predict(X_test)
```

```
1094/1094 [==============================] - 3s 3ms/step
```

In [104]:
```python
r2_score(y_test,z)
```

Out[104]: 0.993786580416008

In [105]:
```python
mse = mean_squared_error(y_test,z)
rmse = mse **0.5
print("mse :", mse)
print("rmse :", rmse)
print("errors for neural network :")
mae = mean_absolute_error(y_test, z)
print("mae : ",mae)
```

```
mse : 0.5346050833943498
rmse : 0.7311669326455825
errors for neural network :
mae :  0.6406848088166618
```

In [106]:
```
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, z)
```

Out[106]: 0.013711704036428712

Random Forest

In [107]:
```
regressor = RandomForestRegressor()

regressor.fit(X_train, y_train)
```

Out[107]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [108]:
```
prediction = regressor.predict(X_test)
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)
```

```
mse :  3.2220552472134893
rmse :  1.7950084253878835
mae: 1.2867925121463273
```

In [109]:
```
r2_score(y_test, prediction)
```

Out[109]: 0.9625518316312511

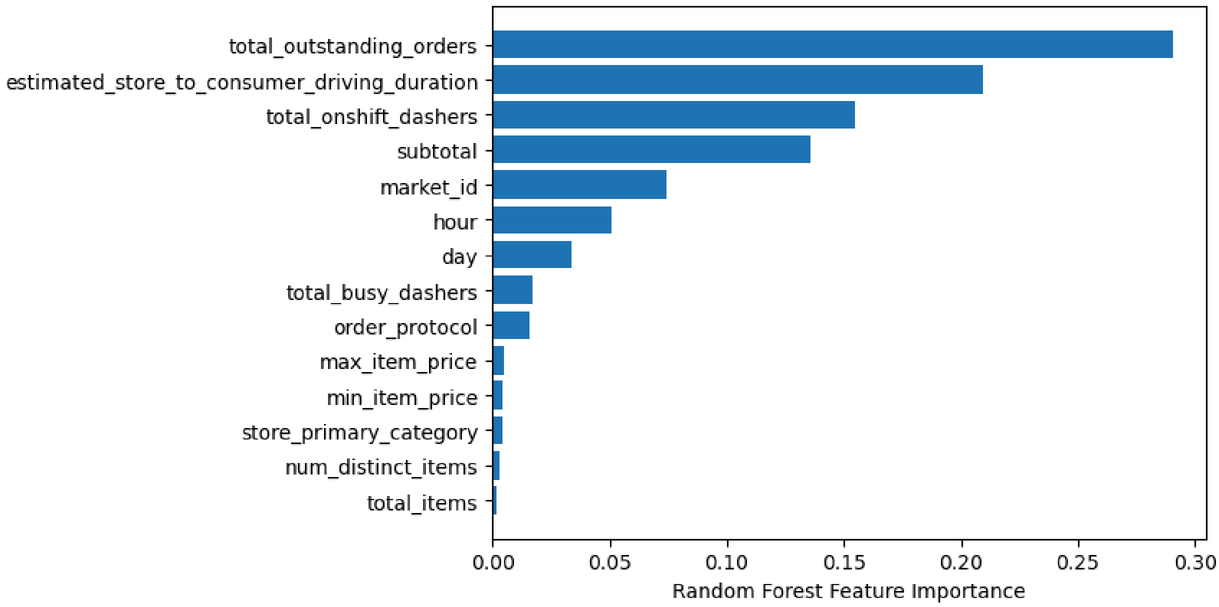
In [110]:
```
def MAPE(Y_actual, Y_Predicted):
    mape = np.mean(np.abs((Y_actual- Y_Predicted)/Y_actual))*100
    return mape
```

In [111]:
```
print("mape : ",MAPE(y_test, prediction))
```

```
mape :  2.771124878312462
```

In [112]:
```
sorted_idx = regressor.feature_importances_.argsort()
plt.barh(df.columns[sorted_idx], regressor.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Out[112]: Text(0.5, 0, 'Random Forest Feature Importance')



By comparing the results of our neural network model with the random forest model we can see that without any tuning or creating pretty complex architectures for training our model we have achieved high accuracy

In [ ]: