PROBLEM STATEMENT

Context:

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

Data Dictionary

Each row in this file corresponds to one unique delivery. 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 part etc)

total_items 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_partners : number of delivery partners on duty at the time order was placed

total_busy_partners : number of delivery partners attending to other tasks

total_outstanding_orders : total number of orders to be fulfilled at the moment

Importing libraries

In [1]:
```
# Importing necessary libraries for data manipulation and analysis
import pandas as pd
import numpy as np
# Importing libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
# Importing libraries for handling datetime operations
from datetime import datetime
# Importing libraries for preprocessing and encoding
from sklearn.preprocessing import StandardScaler, OneHotEncoder
# Importing libraries for splitting data
from sklearn.model_selection import train_test_split
# Importing libraries for neural network
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
# Importing libraries for evaluation metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error
# Setting up visualization styles
sns.set(style='whitegrid')
plt.rcParams['figure.figsize'] = (10, 6)
# Ignoring warnings
import warnings
warnings.filterwarnings('ignore')
```

Importing the data

In [2]:
```
df = pd.read_csv(r"H:\Scaler\Deep learning\Porter NN Project\dataset.csv\dataset.csv")
```

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

Out[3]:

	market_id	created_at	actual_delivery_time	store_id	store_primary_category	order_protocol	total_items	su
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	df263d996281d984952c07998dc54358	american	1.0	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	

In [4]:
```
df.shape
```

Out[4]:
```
(197428, 14)
```

The dataset has 197428 rows and 14 columns

Checking info of the features in the dataset

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   market_id                 196441 non-null  float64
 1   created_at                197428 non-null  object 
 2   actual_delivery_time      197421 non-null  object 
 3   store_id                  197428 non-null  object 
 4   store_primary_category    192668 non-null  object 
 5   order_protocol            196433 non-null  float64
 6   total_items               197428 non-null  int64  
 7   subtotal                  197428 non-null  int64  
 8   num_distinct_items        197428 non-null  int64  
 9   min_item_price            197428 non-null  int64  
 10  max_item_price            197428 non-null  int64  
 11  total_onshift_partners    181166 non-null  float64
 12  total_busy_partners       181166 non-null  float64
 13  total_outstanding_orders  181166 non-null  float64
dtypes: float64(5), int64(5), object(4)
memory usage: 21.1+ MB
```

Checking for null values

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
market_id                      987
created_at                       0
actual_delivery_time             7
store_id                         0
store_primary_category        4760
order_protocol                 995
total_items                      0
subtotal                         0
num_distinct_items               0
min_item_price                   0
max_item_price                   0
total_onshift_partners       16262
total_busy_partners          16262
total_outstanding_orders     16262
dtype: int64
```

In [7]:
```
df.isnull().sum().sum()
```

Out[7]:
```
55535
```

There are a total of 55535 null values in the dataset

The market id column has 987 null values

The actual delivery time has 7 null values

The store_primary_category has 4760 null values

The orider_protocol has 995 null values

The total_onshift_partners , total_busy_partners , total_outstanding_orders has 16262 null values each respectively

In [8]:
```
df.isna().sum()/df.shape[0]*100
```

Out[8]:
```
market_id                   0.499929
created_at                  0.000000
actual_delivery_time        0.003546
store_id                    0.000000
store_primary_category      2.411006
order_protocol              0.503981
total_items                 0.000000
subtotal                    0.000000
num_distinct_items          0.000000
min_item_price              0.000000
max_item_price              0.000000
total_onshift_partners      8.236927
total_busy_partners         8.236927
total_outstanding_orders    8.236927
dtype: float64
```

Converting data type of columns created_at and actual_delivery_time to date time

In [9]:
```
DFDT = ['created_at','actual_delivery_time']

for i in DFDT:
    df[i] = pd.to_datetime(df[i])
```

In [10]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 196441 non-null  float64
 1   created_at                197428 non-null  datetime64[ns]
 2   actual_delivery_time      197421 non-null  datetime64[ns]
 3   store_id                  197428 non-null  object
 4   store_primary_category    192668 non-null  object
 5   order_protocol            196433 non-null  float64
 6   total_items               197428 non-null  int64
 7   subtotal                  197428 non-null  int64
 8   num_distinct_items        197428 non-null  int64
 9   min_item_price            197428 non-null  int64
 10  max_item_price            197428 non-null  int64
 11  total_onshift_partners    181166 non-null  float64
 12  total_busy_partners       181166 non-null  float64
 13  total_outstanding_orders  181166 non-null  float64
dtypes: datetime64[ns](2), float64(5), int64(5), object(2)
memory usage: 21.1+ MB
```

Creating target column(Time taken)

In [11]:
```
# Create a new column named 'time_taken' to store the difference in minutes
df['time_taken'] = (df['actual_delivery_time'] - df['created_at'])
```

In [12]:
```
df.head()
```

Out[12]:

	market_id	created_at	actual_delivery_time	store_id	store_primary_category	order_protocol	total_items	su
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	df263d996281d984952c07998dc54358	american	1.0	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	

In [13]:
```
# Extracting the total minutes from the 'time_taken' column
df['time_taken_minutes'] = df['time_taken'].dt.total_seconds() // 60
```

In [14]:
```
df.head()
```

Out[14]:

	market_id	created_at	actual_delivery_time	store_id	store_primary_category	order_protocol	total_items	su
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	df263d996281d984952c07998dc54358	american	1.0	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	

Feature Engineering and Data Preprocessing

In [15]:
```
# Extracting hour and day of the week from 'created_at'
df['order_hour'] = df['created_at'].dt.hour
df['order_day_of_week'] = df['created_at'].dt.dayofweek # Monday=0, Sunday=6
```

In [16]:
```
df.head()
```

Out[16]:

	market_id	created_at	actual_delivery_time	store_id	store_primary_category	order_protocol	total_items	su
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	df263d996281d984952c07998dc54358	american	1.0	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	

Dropping columns that arent useful anymore

In [18]:
```
df.drop(['time_taken','created_at','actual_delivery_time'],axis=1,inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[18], line 1
----> 1 df.drop(['time_taken','created_at','actual_delivery_time'],axis=1,inplace=True)
      3 df.info()

File D:\Users\india\anaconda3\lib\site-packages\pandas\util\_decorators.py:331, in deprecate_nonkeyword_a
rguments.<locals>.decorate.<locals>.wrapper(*args, **kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File D:\Users\india\anaconda3\lib\site-packages\pandas\core\frame.py:5399, in DataFrame.drop(self, label
s, axis, index, columns, level, inplace, errors)
   5251 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"])
```

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   market_id                 196441 non-null  float64
 1   store_id                  197428 non-null  object 
 2   store_primary_category    192668 non-null  object 
 3   order_protocol            196433 non-null  float64
 4   total_items               197428 non-null  int64  
 5   subtotal                  197428 non-null  int64  
 6   num_distinct_items        197428 non-null  int64  
 7   min_item_price            197428 non-null  int64  
 8   max_item_price            197428 non-null  int64  
 9   total_onshift_partners    181166 non-null  float64
 10  total_busy_partners       181166 non-null  float64
 11  total_outstanding_orders  181166 non-null  float64
 12  time_taken_minutes        197421 non-null  float64
 13  order_hour                197428 non-null  int64  
 14  order_day_of_week         197428 non-null  int64  
dtypes: float64(6), int64(7), object(2)
memory usage: 22.6+ MB
```

Handling Null values

In [20]:
```
df.isna().sum()
```

Out[20]:
```
market_id                    987
store_id                       0
store_primary_category      4760
order_protocol               995
total_items                    0
subtotal                       0
num_distinct_items             0
min_item_price                 0
max_item_price                 0
total_onshift_partners     16262
total_busy_partners        16262
total_outstanding_orders   16262
time_taken_minutes             7
order_hour                     0
order_day_of_week              0
dtype: int64
```

In [22]:
```
# Finding the number of unique values in each column
unique_values = {column: df[column].nunique() for column in df.columns}
# Displaying the unique values count for each column
for column, unique_count in unique_values.items():
    print(f"{column}: {unique_count}")
```

```
market_id: 6
store_id: 6743
store_primary_category: 74
order_protocol: 7
total_items: 57
subtotal: 8368
num_distinct_items: 20
min_item_price: 2312
max_item_price: 2652
total_onshift_partners: 172
total_busy_partners: 159
total_outstanding_orders: 281
time_taken_minutes: 274
order_hour: 19
order_day_of_week: 7
```

In [23]:
```
df1=df.dropna()
```

In [24]:
```
df[df["store_id"]=="252a3dbaeb32e7690242ad3b556e626b"]
```

Out[24]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items
52018	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	4	5950	3
52019	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	1	2735	1
52020	2.0	252a3dbaeb32e7690242ad3b556e626b	burger	3.0	2	2515	2
52021	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	2	3915	2
52022	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	1	2064	1
...	...	...	...	...	...	...	...
63432	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	1	1828	1
63433	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	3	4055	2
63434	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	1	1510	1
63435	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	1	1890	1
63436	6.0	252a3dbaeb32e7690242ad3b556e626b	american	5.0	2	2235	2

350 rows × 15 columns

Checking whether mean or median is the right choice for Null

imputation

In [25]:
```
df.groupby("market_id")["total_onshift_partners"].mean()
```

Out[25]:
```
market_id
1.0    24.208854
2.0    62.590695
3.0    18.847580
4.0    60.464482
5.0    23.911045
6.0    44.929771
Name: total_onshift_partners, dtype: float64
```

In [26]:
```
df.groupby("market_id")["total_onshift_partners"].median()
```

Out[26]:
```
market_id
1.0    19.0
2.0    55.0
3.0    15.0
4.0    60.0
5.0    20.0
6.0    36.0
Name: total_onshift_partners, dtype: float64
```

In [27]:
```
df.groupby("order_hour")["total_onshift_partners"].mean()
```

Out[27]:
```
order_hour
0     27.933751
1     54.325601
2     67.995169
3     64.205588
4     44.996112
5     23.589613
6     13.421094
7     10.777778
8      0.000000
14     0.550000
15     2.141473
16     4.965949
17     7.757729
18    15.092275
19    32.199487
20    37.353387
21    30.325540
22    22.749043
23    20.274580
Name: total_onshift_partners, dtype: float64
```

In [28]:
```
df.groupby("order_day_of_week")["total_onshift_partners"].mean()
```

Out[28]:
```
order_day_of_week
0    42.084044
1    37.333062
2    40.067352
3    43.746503
4    48.602855
5    52.111917
6    45.943654
Name: total_onshift_partners, dtype: float64
```

In [29]:
```
df.groupby(["market_id","order_hour"])["total_onshift_partners"].mean()
```

Out[29]:
```
market_id  order_hour
1.0        0             14.437811
           1             26.014145
           2             36.809734
           3             37.072227
           4             27.385254
                           ...
6.0        19            30.744186
           20            40.627907
           21            31.200000
           22            23.806452
           23            18.000000
Name: total_onshift_partners, Length: 106, dtype: float64
```

Mean Imputation

In [32]:
```
# List of columns to impute
columns_to_impute = ['total_outstanding_orders', 'total_busy_partners', 'total_onshift_partners']
# Group by 'market_id' and 'order_hour'
grouped = df.groupby(['market_id', 'order_hour'])
# Impute missing values
for column in columns_to_impute:
    # Calculate the mean for each group and transform to align with the original Da
    df[column] = grouped[column].transform(lambda x: x.fillna(x.mean()))
```

In [33]:
```
df
```

Out[33]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items
0	1.0	df263d996281d984952c07998dc54358	american	1.0	4	3441	4
1	2.0	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	1900	1
2	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	1900	1
3	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	6900	5
4	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	3900	3
...	...	...	...	...	...	...	...
197423	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	3	1389	3
197424	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	6	3010	4
197425	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	5	1836	3
197426	1.0	c81e155d85dae5430a8cee6f2242e82c	sandwich	1.0	1	1175	1
197427	1.0	c81e155d85dae5430a8cee6f2242e82c	sandwich	1.0	4	2605	4

197428 rows × 15 columns

In [34]:
```
df.isna().sum()
```

Out[34]:
```
market_id                    987
store_id                       0
store_primary_category      4760
order_protocol               995
total_items                    0
subtotal                       0
num_distinct_items             0
min_item_price                 0
max_item_price                 0
total_onshift_partners       989
total_busy_partners          989
total_outstanding_orders     989
time_taken_minutes             7
order_hour                     0
order_day_of_week              0
dtype: int64
```

In [35]:
```
df[df["total_onshift_partners"].isnull()].dropna(inplace=True)
```

In [36]:
```
df.isna().sum()
```

Out[36]:
```
market_id                    987
store_id                       0
store_primary_category      4760
order_protocol               995
total_items                    0
subtotal                       0
num_distinct_items             0
min_item_price                 0
max_item_price                 0
total_onshift_partners       989
total_busy_partners          989
total_outstanding_orders     989
time_taken_minutes             7
order_hour                     0
order_day_of_week              0
dtype: int64
```

In [37]:
```
df= df[~df['total_onshift_partners'].isnull()]
```

In [38]:
```
df.isna().sum()
```

Out[38]:
```
market_id                      0
store_id                       0
store_primary_category      4268
order_protocol               508
total_items                    0
subtotal                       0
num_distinct_items             0
min_item_price                 0
max_item_price                 0
total_onshift_partners         0
total_busy_partners            0
total_outstanding_orders       0
time_taken_minutes             7
order_hour                     0
order_day_of_week              0
dtype: int64
```

In [39]:
```
df= df[~df['order_protocol'].isnull()]
```

In [40]:
```
df.isna().sum()
```

Out[40]:
```
market_id                      0
store_id                       0
store_primary_category      4005
order_protocol                 0
total_items                    0
subtotal                       0
num_distinct_items             0
min_item_price                 0
max_item_price                 0
total_onshift_partners         0
total_busy_partners            0
total_outstanding_orders       0
time_taken_minutes             7
order_hour                     0
order_day_of_week              0
dtype: int64
```

In [41]:
```
df= df[~df['time_taken_minutes'].isnull()]
```

In [42]:
```
df.isna().sum()
```

Out[42]:
```
market_id                      0
store_id                       0
store_primary_category      4005
order_protocol                 0
total_items                    0
subtotal                       0
num_distinct_items             0
min_item_price                 0
max_item_price                 0
total_onshift_partners         0
total_busy_partners            0
total_outstanding_orders       0
time_taken_minutes             0
order_hour                     0
order_day_of_week              0
dtype: int64
```

In [43]:
```
df[df["store_primary_category"].isna()]
```

Out[43]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items
2	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	1900	1
3	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	6900	5
4	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	3900	3
5	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	5000	3
6	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	2	3900	2
...	...	...	...	...	...	...	...
197208	1.0	77c493ec14246d748db3ee8fce0092db	NaN	1.0	7	5100	6
197209	1.0	77c493ec14246d748db3ee8fce0092db	NaN	1.0	7	7200	6
197210	1.0	77c493ec14246d748db3ee8fce0092db	NaN	1.0	3	2800	3
197211	1.0	77c493ec14246d748db3ee8fce0092db	NaN	1.0	2	1400	2
197212	1.0	77c493ec14246d748db3ee8fce0092db	NaN	1.0	5	2800	5

4005 rows × 15 columns

In [44]:
```
df[df["store_id"]=='f0ade77b43923b38237db569b016ba25']
```

Out[44]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_
1	2.0	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	1900	1	
2	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	1900	1	
3	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	6900	5	
4	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	3900	3	
5	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	5000	3	
6	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	2	3900	2	
7	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	4	4850	4	
8	2.0	f0ade77b43923b38237db569b016ba25	indian	3.0	4	4771	3	
9	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	2	2100	2	
10	3.0	f0ade77b43923b38237db569b016ba25	NaN	4.0	4	4300	4	
11	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	2	2200	2	
12	3.0	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	1900	1	
13	3.0	f0ade77b43923b38237db569b016ba25	NaN	4.0	4	4986	4	

In [45]:
```
df[df["store_primary_category"].isna()]["store_id"].nunique()
```

Out[45]: 632

In [46]:
```
df2=df[df["store_primary_category"].isna()]["store_id"].unique()
```

Imputing store_primary_category by mode

In [48]:
```python
# Function to impute missing values by mode, handling ties randomly
def impute_by_mode(df, column):
    # Get the mode(s)
    modes = df[column].mode()
    if len(modes) > 1:
    # If there are ties, choose one randomly with equal probability
        chosen_mode = np.random.choice(modes)
    else:
    # If no tie, use the single mode
        chosen_mode = modes[0]
    # Impute missing values with the chosen mode
    df[column].fillna(chosen_mode, inplace=True)
# List of columns to impute
columns_to_impute = ['store_primary_category']
# Apply the function to each column
for column in columns_to_impute:
    impute_by_mode(df, column)
```

In [49]:
```python
df.isna().sum()
```

Out[49]:
```
market_id                   0
store_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_partners      0
total_busy_partners         0
total_outstanding_orders    0
time_taken_minutes          0
order_hour                  0
order_day_of_week           0
dtype: int64
```

In [50]:
```python
df.shape
```

Out[50]:
```
(195924, 15)
```

In [51]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 195924 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 195924 non-null  float64
 1   store_id                  195924 non-null  object
 2   store_primary_category    195924 non-null  object
 3   order_protocol            195924 non-null  float64
 4   total_items               195924 non-null  int64
 5   subtotal                  195924 non-null  int64
 6   num_distinct_items        195924 non-null  int64
 7   min_item_price            195924 non-null  int64
 8   max_item_price            195924 non-null  int64
 9   total_onshift_partners    195924 non-null  float64
 10  total_busy_partners       195924 non-null  float64
 11  total_outstanding_orders  195924 non-null  float64
 12  time_taken_minutes        195924 non-null  float64
 13  order_hour                195924 non-null  int64
 14  order_day_of_week         195924 non-null  int64
dtypes: float64(6), int64(7), object(2)
memory usage: 23.9+ MB
```

In [52]:
```python
store_name_counts = df['store_id'].value_counts()
df['store_name_enc'] = df['store_id'].map(store_name_counts)
```

In [54]:
```python
df = df.drop('store_name_enc', axis=1)
```

In [55]:
```
df
```

Out[55]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items
0	1.0	df263d996281d984952c07998dc54358	american	1.0	4	3441	4
1	2.0	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	1900	1
2	3.0	f0ade77b43923b38237db569b016ba25	american	1.0	1	1900	1
3	3.0	f0ade77b43923b38237db569b016ba25	american	1.0	6	6900	5
4	3.0	f0ade77b43923b38237db569b016ba25	american	1.0	3	3900	3
...	...	...	...	...	...	...	...
197423	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	3	1389	3
197424	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	6	3010	4
197425	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	5	1836	3
197426	1.0	c81e155d85dae5430a8cee6f2242e82c	sandwich	1.0	1	1175	1
197427	1.0	c81e155d85dae5430a8cee6f2242e82c	sandwich	1.0	4	2605	4

195924 rows × 15 columns

Using Label Encoding for store name

In [56]:
```
from sklearn.preprocessing import LabelEncoder
```

In [57]:
```
label_encoder = LabelEncoder()
df['store_name_encoded'] = label_encoder.fit_transform(df['store_id'])
```

In [58]:
```
df
```

Out[58]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items
0	1.0	df263d996281d984952c07998dc54358	american	1.0	4	3441	4
1	2.0	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	1900	1
2	3.0	f0ade77b43923b38237db569b016ba25	american	1.0	1	1900	1
3	3.0	f0ade77b43923b38237db569b016ba25	american	1.0	6	6900	5
4	3.0	f0ade77b43923b38237db569b016ba25	american	1.0	3	3900	3
...	...	...	...	...	...	...	...
197423	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	3	1389	3
197424	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	6	3010	4
197425	1.0	a914ecef9c12ffdb9bede64bb703d877	fast	4.0	5	1836	3
197426	1.0	c81e155d85dae5430a8cee6f2242e82c	sandwich	1.0	1	1175	1
197427	1.0	c81e155d85dae5430a8cee6f2242e82c	sandwich	1.0	4	2605	4

195924 rows × 16 columns

In [59]:
```
df=df.drop("store_id",axis=1)
```

In [61]:
```
df
```

Out[61]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total
0	1.0	american	1.0	4	3441	4	557	1239	
1	2.0	mexican	2.0	1	1900	1	1400	1400	
2	3.0	american	1.0	1	1900	1	1900	1900	
3	3.0	american	1.0	6	6900	5	600	1800	
4	3.0	american	1.0	3	3900	3	1100	1600	
...	...	...	...	...	...	...	...	...	
197423	1.0	fast	4.0	3	1389	3	345	649	
197424	1.0	fast	4.0	6	3010	4	405	825	
197425	1.0	fast	4.0	5	1836	3	300	399	
197426	1.0	sandwich	1.0	1	1175	1	535	535	
197427	1.0	sandwich	1.0	4	2605	4	425	750	

195924 rows × 15 columns

In [62]:
```
duplicates = df.duplicated()
# Print the original DataFrame with a marker for duplicates
print(df.loc[duplicates])
```

```
        market_id store_primary_category  order_protocol  total_items  \
139263        6.0                 indian             3.0            2
166281        6.0                   cafe             4.0            1

        subtotal  num_distinct_items  min_item_price  max_item_price  \
139263      1650                   1             825             825
166281       350                   1             350             350

        total_onshift_partners  total_busy_partners  total_outstanding_orders  \
139263               39.813559             40.40678                 51.135593
166281               39.813559             40.40678                 51.135593

        time_taken_minutes  order_hour  order_day_of_week  store_name_encoded
139263                24.0           4                  1                2637
166281                39.0           4                  4                1501
```

In [63]:
```
df=df.drop_duplicates()
```

In [64]:
```
df
```

Out[64]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total
0	1.0	american	1.0	4	3441	4	557	1239	
1	2.0	mexican	2.0	1	1900	1	1400	1400	
2	3.0	american	1.0	1	1900	1	1900	1900	
3	3.0	american	1.0	6	6900	5	600	1800	
4	3.0	american	1.0	3	3900	3	1100	1600	
...	...	...	...	...	...	...	...	...	
197423	1.0	fast	4.0	3	1389	3	345	649	
197424	1.0	fast	4.0	6	3010	4	405	825	
197425	1.0	fast	4.0	5	1836	3	300	399	
197426	1.0	sandwich	1.0	1	1175	1	535	535	
197427	1.0	sandwich	1.0	4	2605	4	425	750	

195922 rows × 15 columns

In [65]:
```
df.isna().sum()
```

Out[65]:
```
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_partners     0
total_busy_partners        0
total_outstanding_orders   0
time_taken_minutes         0
order_hour                 0
order_day_of_week          0
store_name_encoded         0
dtype: int64
```

In [66]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 195922 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   market_id                 195922 non-null  float64
 1   store_primary_category    195922 non-null  object 
 2   order_protocol            195922 non-null  float64
 3   total_items               195922 non-null  int64  
 4   subtotal                  195922 non-null  int64  
 5   num_distinct_items        195922 non-null  int64  
 6   min_item_price            195922 non-null  int64  
 7   max_item_price            195922 non-null  int64  
 8   total_onshift_partners    195922 non-null  float64
 9   total_busy_partners       195922 non-null  float64
 10  total_outstanding_orders  195922 non-null  float64
 11  time_taken_minutes        195922 non-null  float64
 12  order_hour                195922 non-null  int64  
 13  order_day_of_week         195922 non-null  int64  
 14  store_name_encoded        195922 non-null  int32  
dtypes: float64(6), int32(1), int64(7), object(1)
memory usage: 23.2+ MB
```

label Encoding store_primary_category

In [69]:
```
label_encoder = LabelEncoder()
df['store_primary_category_enc'] = label_encoder.fit_transform(df['store_primary_category'])
```

In [70]:
```
df=df.drop("store_primary_category",axis=1)
```

In [71]:
```
df
```

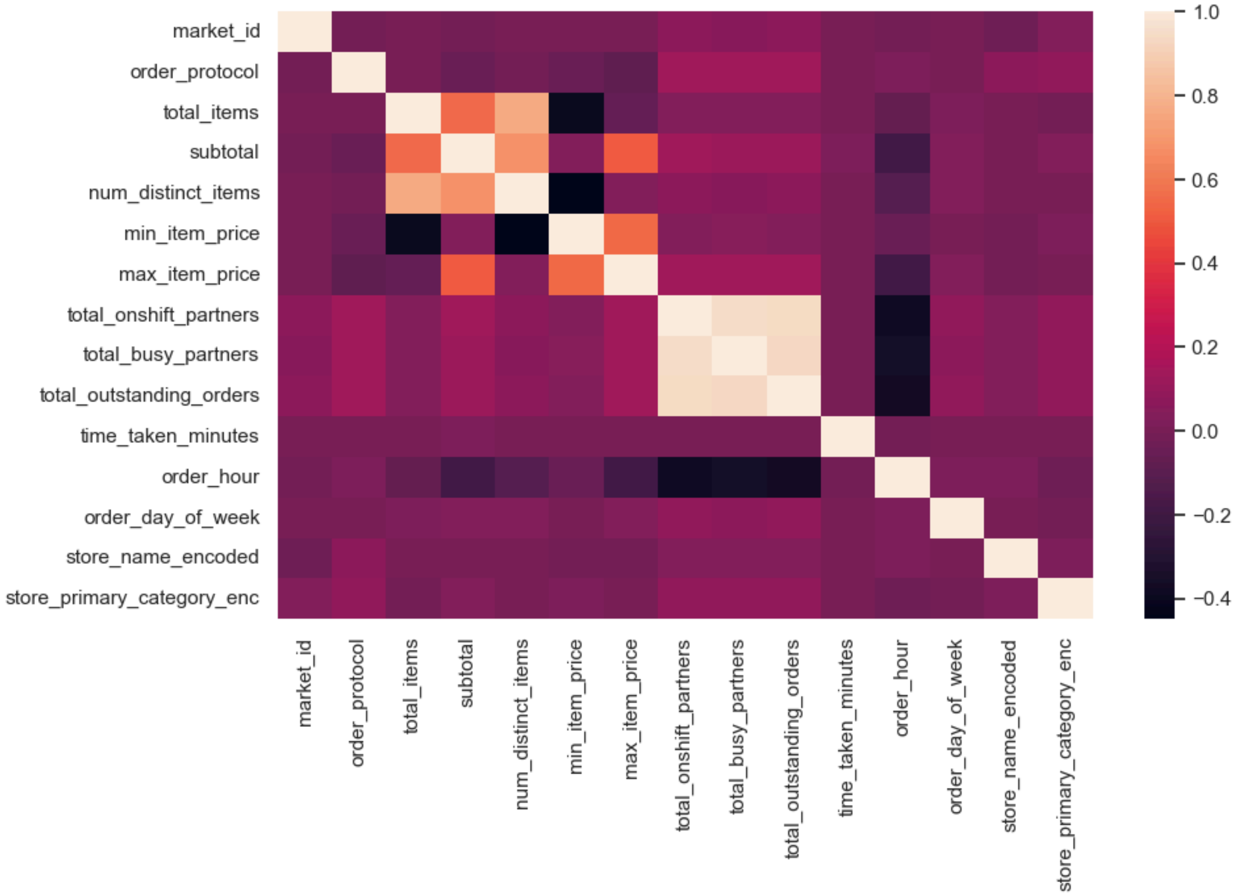
Out[71]:

	market_id	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_l
0	1.0	1.0	4	3441	4	557	1239	33.0	
1	2.0	2.0	1	1900	1	1400	1400	1.0	
2	3.0	1.0	1	1900	1	1900	1900	1.0	
3	3.0	1.0	6	6900	5	600	1800	1.0	
4	3.0	1.0	3	3900	3	1100	1600	6.0	
...	...	...	...	...	...	...	...	...	
197423	1.0	4.0	3	1389	3	345	649	17.0	
197424	1.0	4.0	6	3010	4	405	825	12.0	
197425	1.0	4.0	5	1836	3	300	399	39.0	
197426	1.0	1.0	1	1175	1	535	535	7.0	
197427	1.0	1.0	4	2605	4	425	750	20.0	

195922 rows × 15 columns

Data Vizualisation

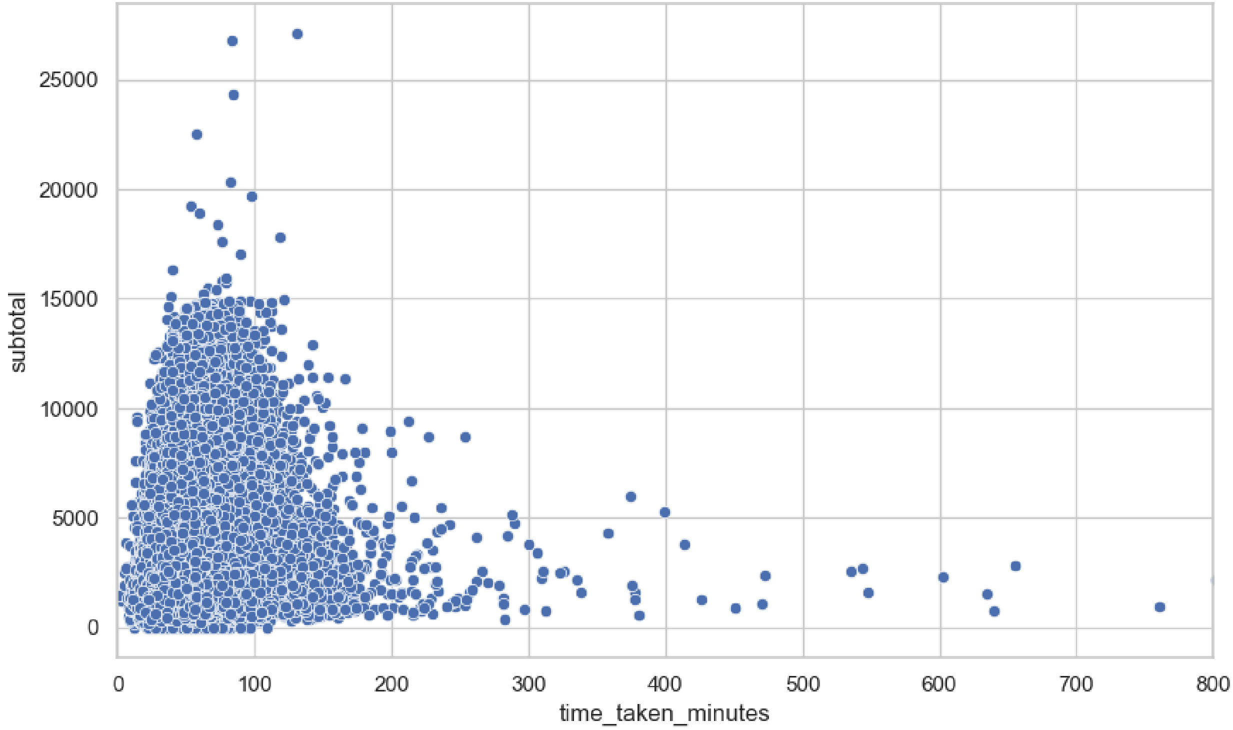
In [74]:
```
sns.heatmap(df.corr())
plt.show()
```



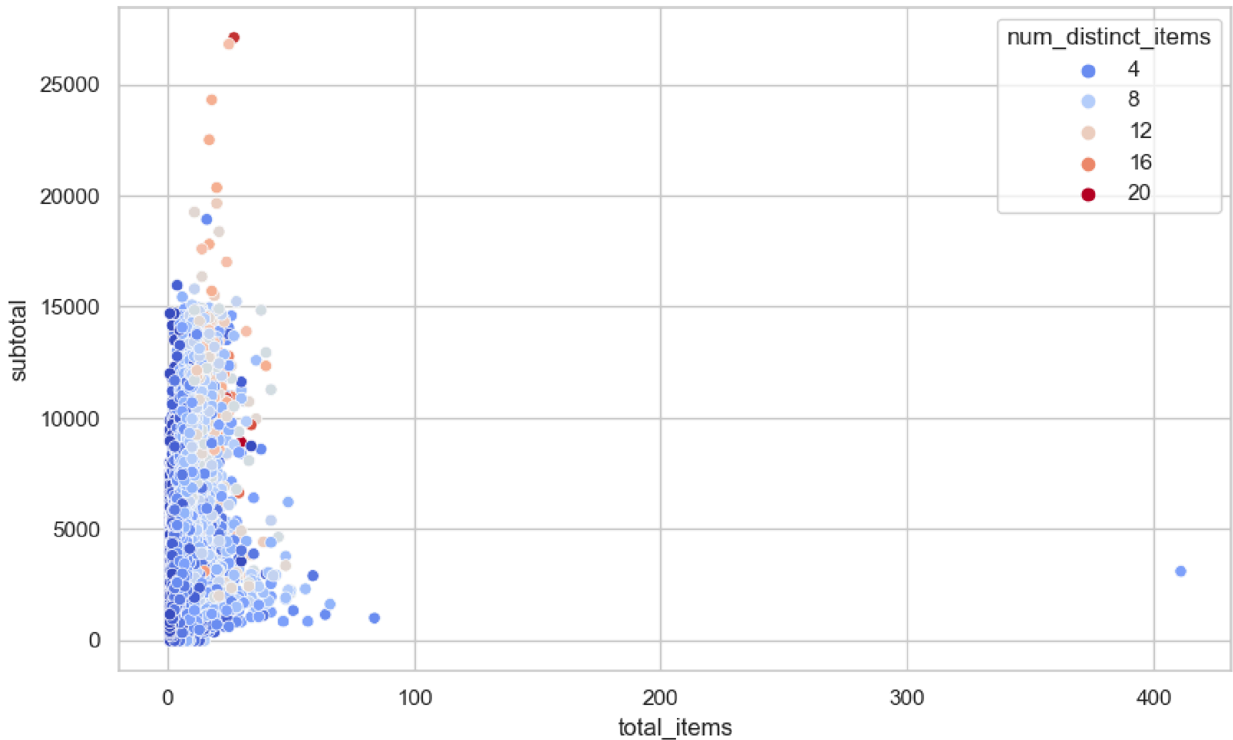
In [75]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 195922 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   market_id                   195922 non-null  float64
 1   order_protocol              195922 non-null  float64
 2   total_items                 195922 non-null  int64  
 3   subtotal                    195922 non-null  int64  
 4   num_distinct_items          195922 non-null  int64  
 5   min_item_price              195922 non-null  int64  
 6   max_item_price              195922 non-null  int64  
 7   total_onshift_partners      195922 non-null  float64
 8   total_busy_partners         195922 non-null  float64
 9   total_outstanding_orders    195922 non-null  float64
 10  time_taken_minutes          195922 non-null  float64
 11  order_hour                  195922 non-null  int64  
 12  order_day_of_week           195922 non-null  int64  
 13  store_name_encoded          195922 non-null  int32  
 14  store_primary_category_enc  195922 non-null  int32  
dtypes: float64(6), int32(2), int64(7)
memory usage: 22.4 MB
```

In [77]:
```
# Create the scatter plot
sns.scatterplot(x='time_taken_minutes', y='subtotal', data=df)
# Set the x-axis limit
plt.xlim(0, 800)
plt.show()
```



In [80]:
```
sns.scatterplot(x='total_items', y='subtotal', hue='num_distinct_items', palette='coolwarm', data=df)
plt.show()
```



In [81]:
```
df3=df.copy()
```

In [82]:
```
df3.shape
```

Out[82]:
```
(195922, 15)
```

In [83]:
```
df3=df3.drop("store_name_encoded",axis=1)
```

Removing outliers using LOF

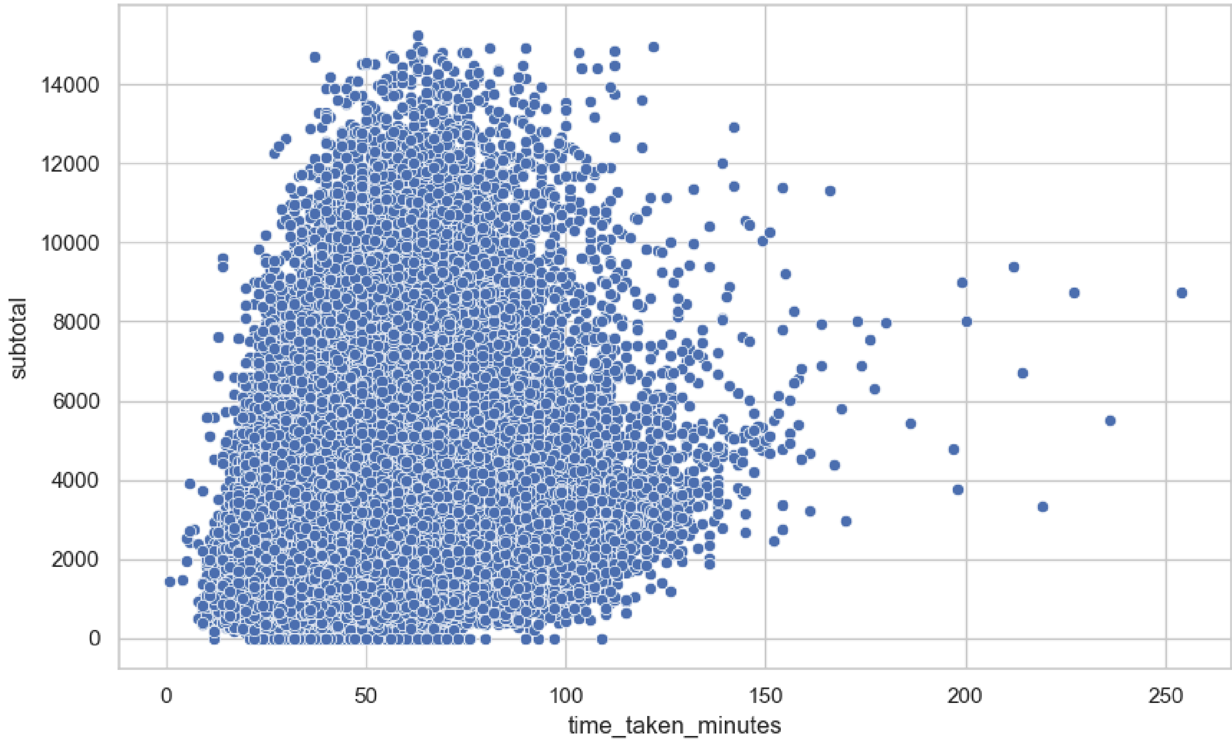
In [84]:
```
from sklearn.neighbors import LocalOutlierFactor
import matplotlib.pyplot as plt
model1 = LocalOutlierFactor(contamination=0.05)
df3['lof_anomaly_score'] = model1.fit_predict(df3)
```

In [85]:
```
print("number of outliers : ",(len(df3.loc[(df3['lof_anomaly_score'] == -1)])))
df3=df3.loc[(df3['lof_anomaly_score'] == 1)]
```

```
number of outliers :  9797
```

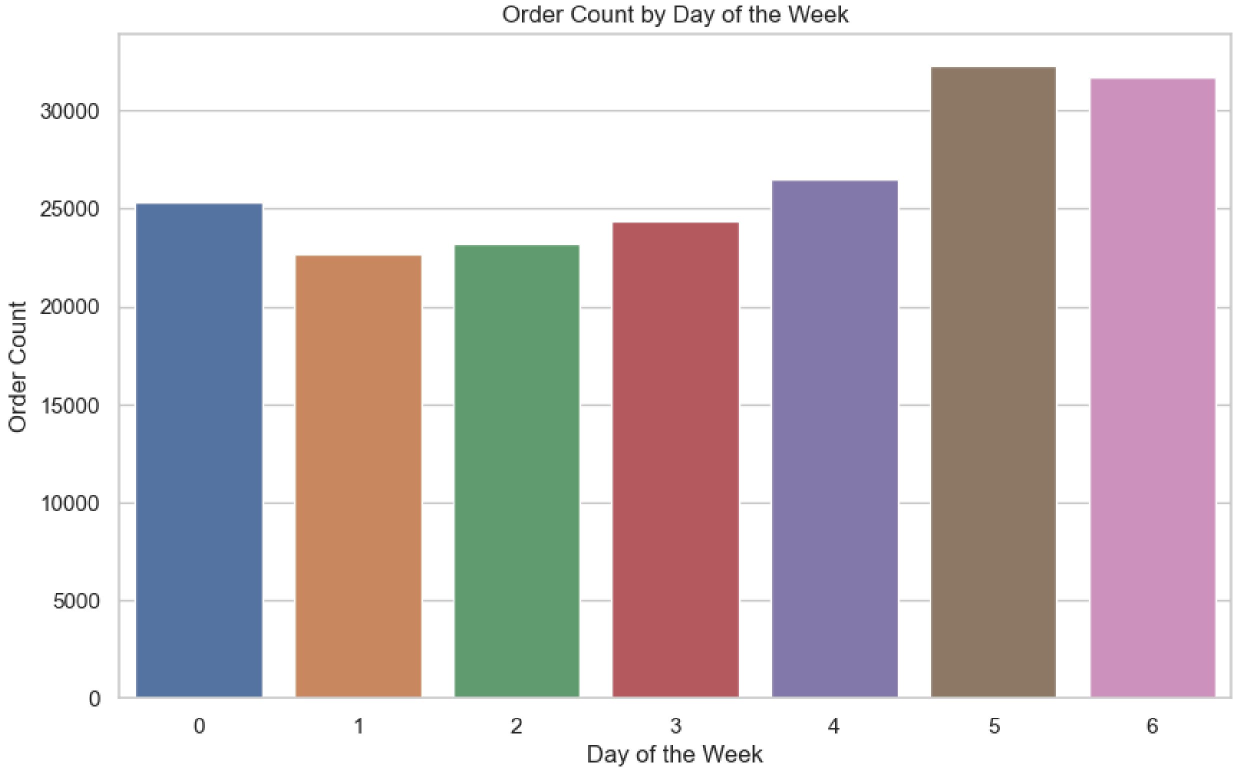
In [86]:
```
df3.drop(['lof_anomaly_score'],axis=1,inplace=True)
```

In [88]:
```
# Create the scatter plot
sns.scatterplot(x='time_taken_minutes', y='subtotal', data=df3)
plt.show()
```

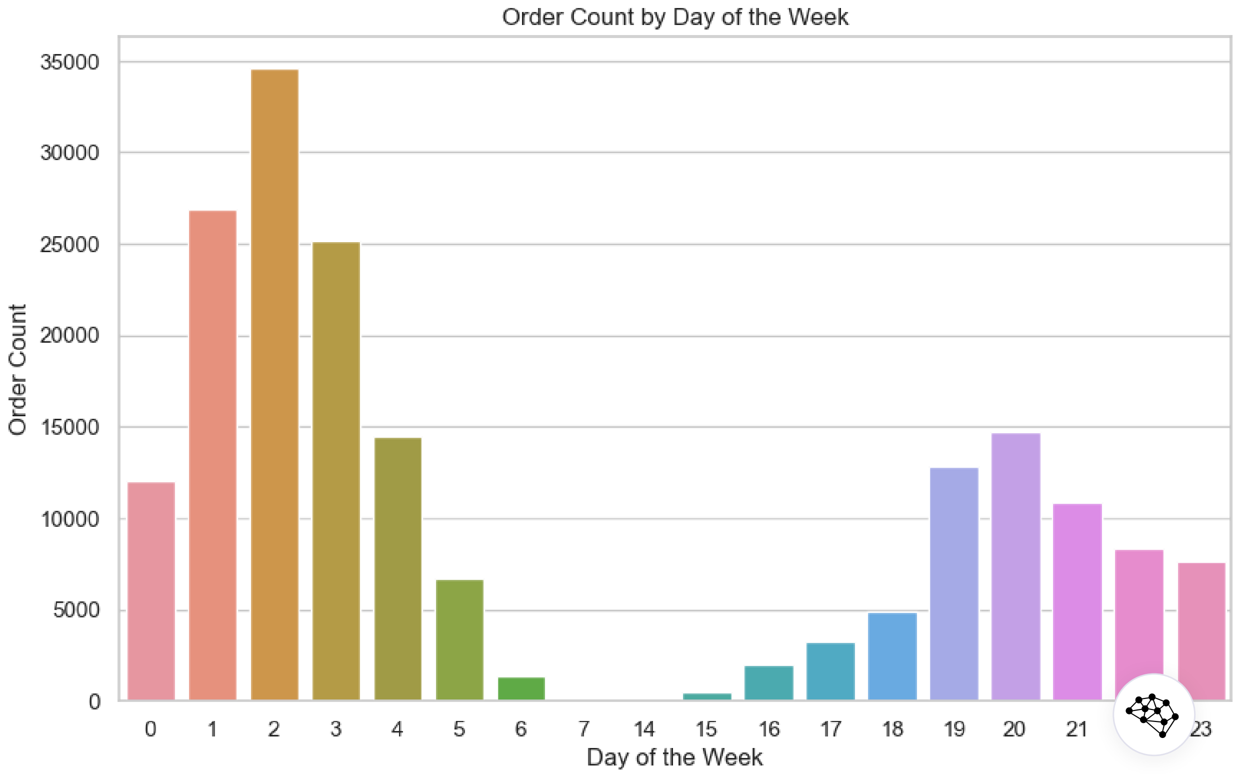


Making various plots from features

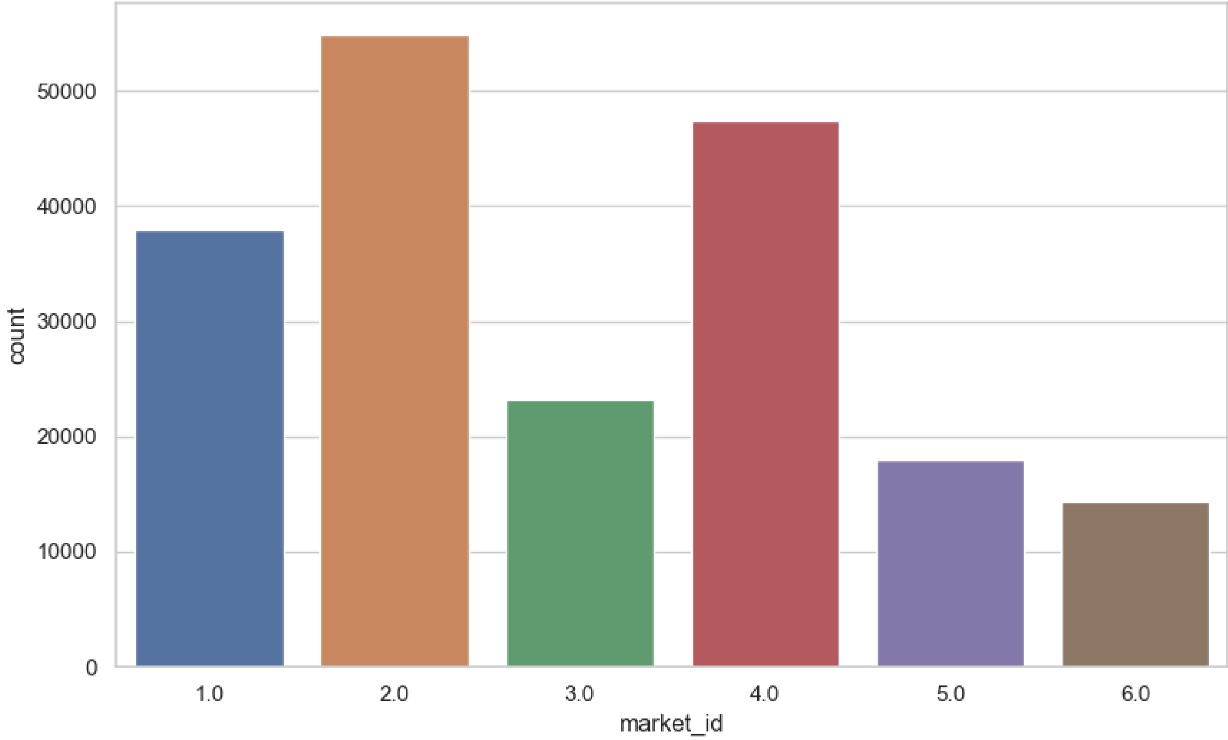
In [89]:
```
# Create a countplot for the 'order_day_of_week' column
sns.countplot(x='order_day_of_week', data=df3)
# Set the title and labels
plt.title('Order Count by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Order Count')
# Show the plot
plt.show()
```



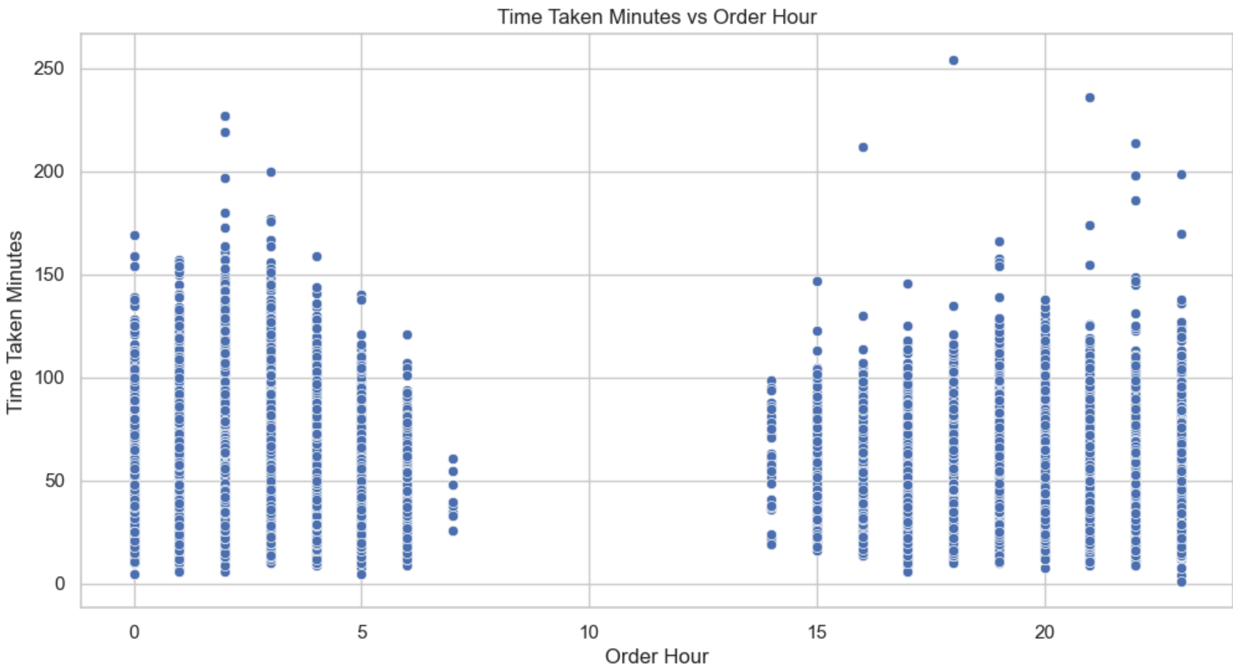
In [90]:
```
# Create a countplot for the 'order_day_of_week' column
sns.countplot(x='order_hour', data=df3)
# Set the title and labels
plt.title('Order Count by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Order Count')
# Show the plot
plt.show()
```



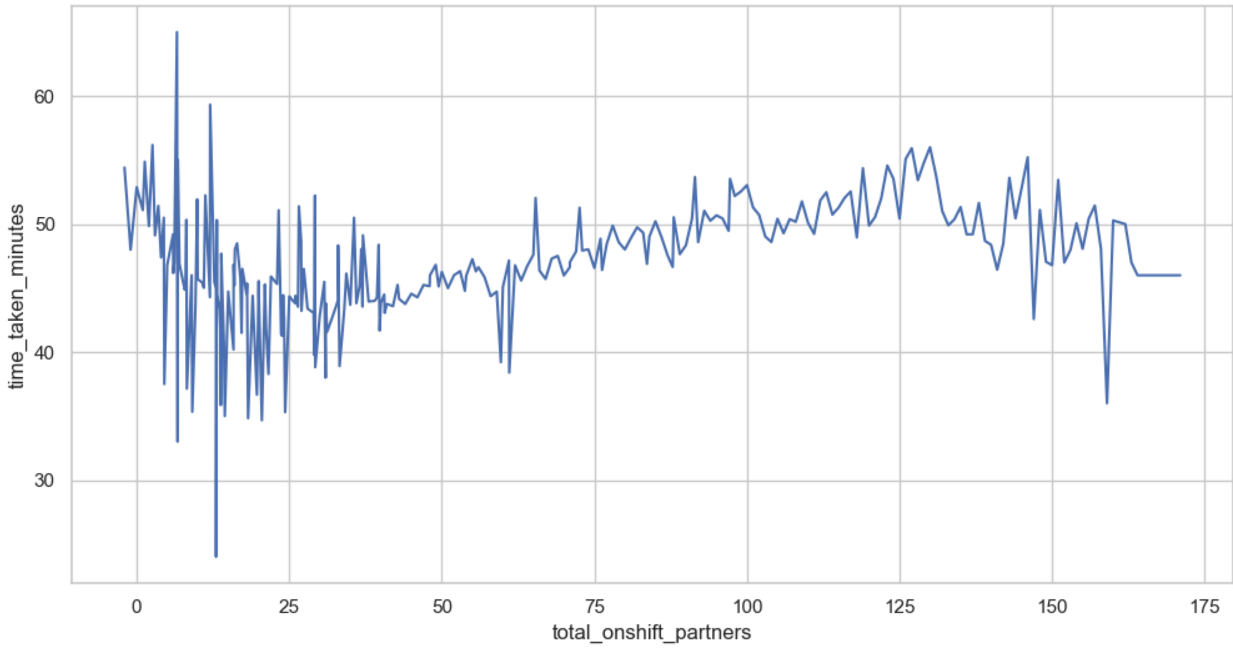
In [91]:
```
sns.countplot(x=df.market_id)
plt.show()
```



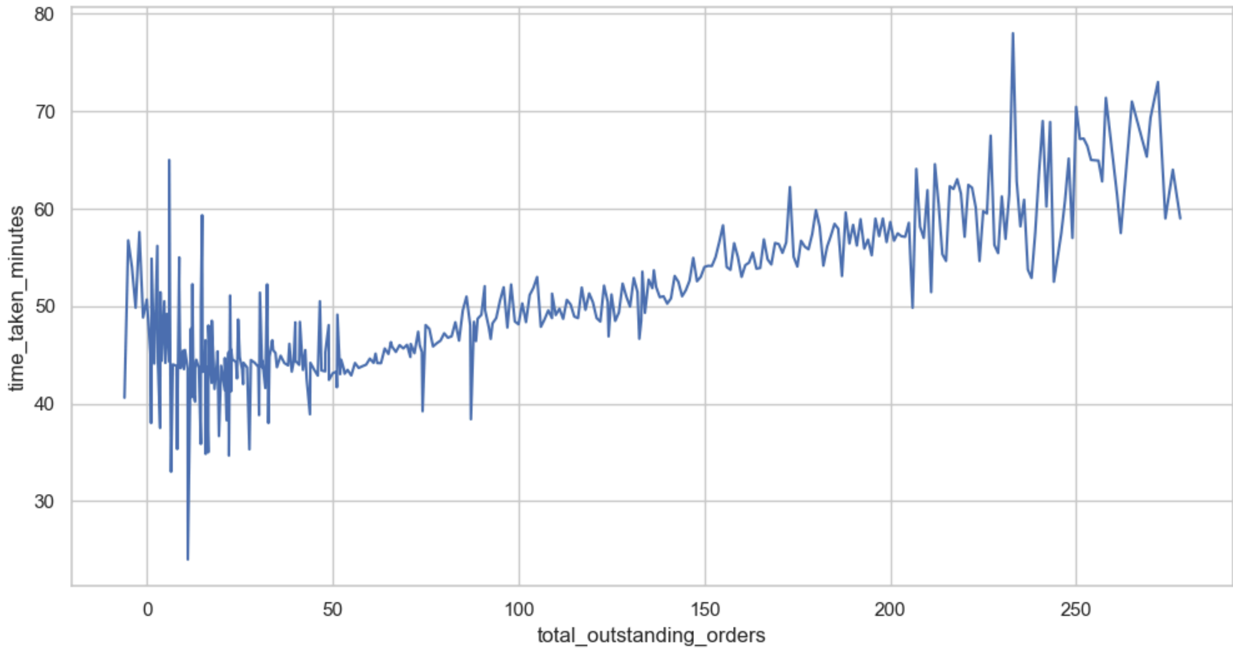
In [92]:
```
# Create a scatter plot for 'order_hour' vs 'time_taken_minutes'
plt.figure(figsize=(12, 6))
sns.scatterplot(x='order_hour', y='time_taken_minutes', data=df3)
# Set the title and labels
plt.title('Time Taken Minutes vs Order Hour')
plt.xlabel('Order Hour')
plt.ylabel('Time Taken Minutes')
# Show the plot
plt.show()
```



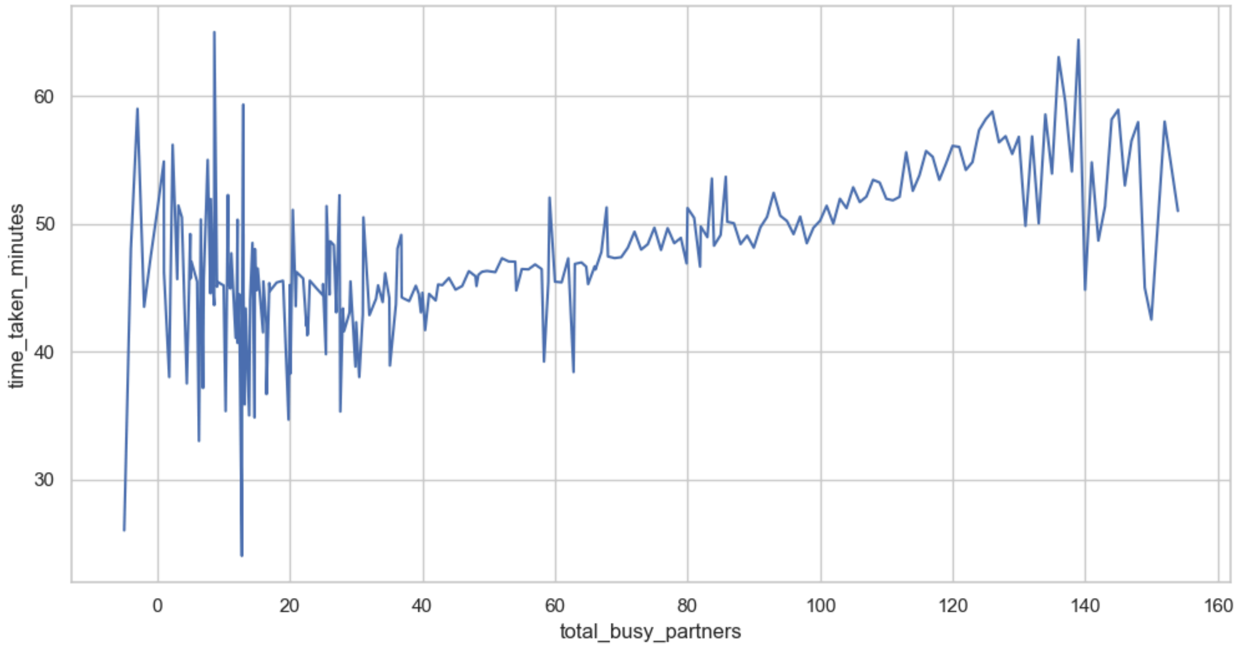
In [93]:
```
plt.figure(figsize=(12, 6))
sns.lineplot(x='total_onshift_partners', y='time_taken_minutes', data=df3, ci=None)
plt.show()
```



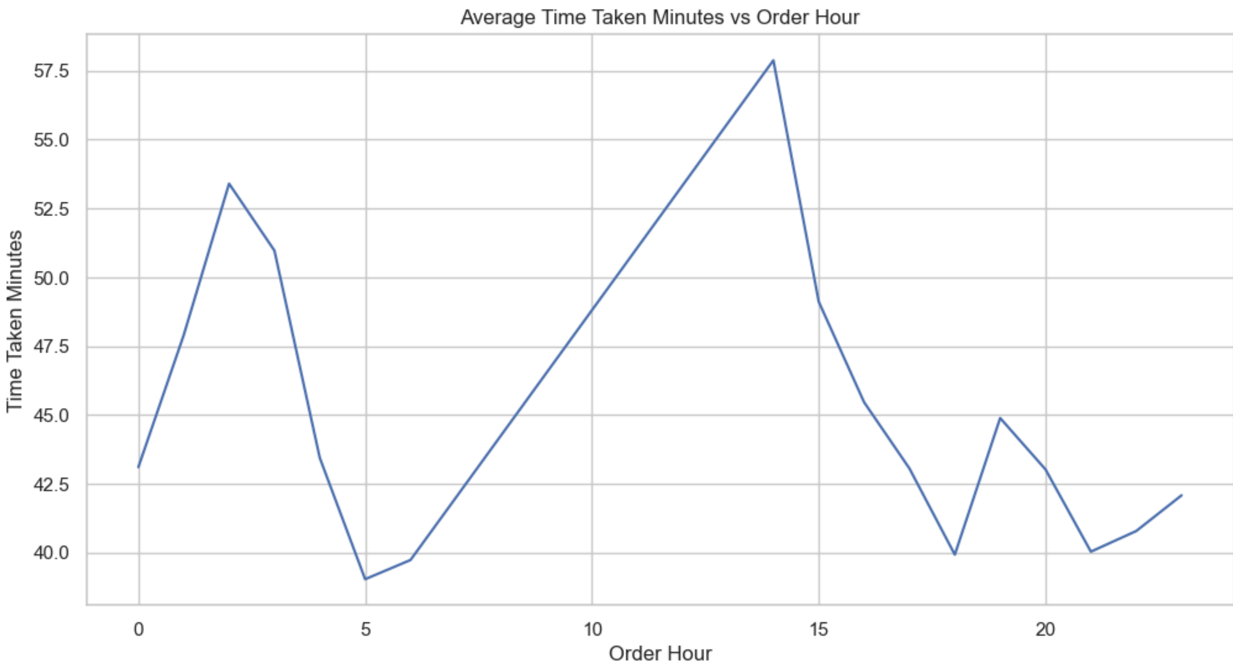
In [95]:
```
plt.figure(figsize=(12, 6))
sns.lineplot(x='total_outstanding_orders', y='time_taken_minutes', data=df3, ci=None)
plt.show()
```



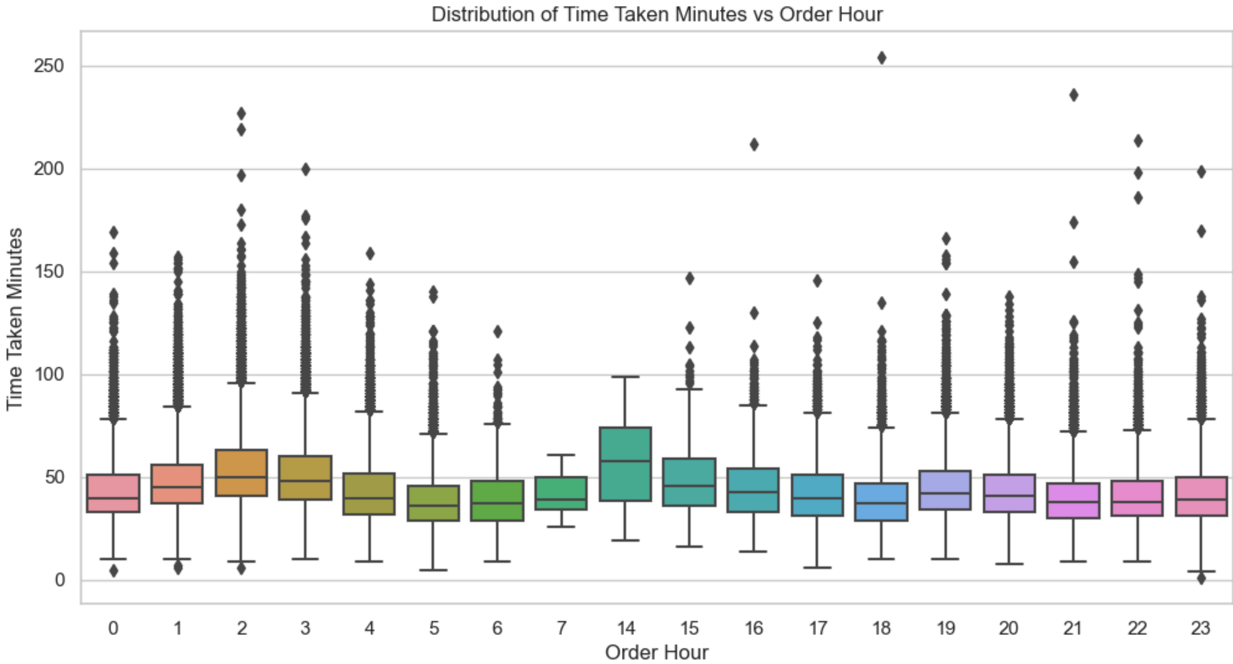
In [96]:
```
plt.figure(figsize=(12, 6))
sns.lineplot(x='total_busy_partners', y='time_taken_minutes', data=df3, ci=None)
plt.show()
```



In [97]:
```
plt.figure(figsize=(12, 6))
sns.lineplot(x='order_hour', y='time_taken_minutes', data=df3, ci=None)
# Set the title and labels
plt.title('Average Time Taken Minutes vs Order Hour')
plt.xlabel('Order Hour')
plt.ylabel('Time Taken Minutes')
# Show the plot
plt.show()
```



In [98]:
```
# Create a box plot for 'order_hour' vs 'time_taken_minutes'
plt.figure(figsize=(12, 6))
sns.boxplot(x='order_hour', y='time_taken_minutes', data=df3)
# Set the title and labels
plt.title('Distribution of Time Taken Minutes vs Order Hour')
plt.xlabel('Order Hour')
plt.ylabel('Time Taken Minutes')
# Show the plot
plt.show()
```



In [99]:
```
y = df3['time_taken_minutes']
x = df3.drop(['time_taken_minutes'], axis=1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

In [100]:
```
x
```

Out[100]:

	market_id	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_I
0	1.0	1.0	4	3441	4	557	1239	33.0	
1	2.0	2.0	1	1900	1	1400	1400	1.0	
2	3.0	1.0	1	1900	1	1900	1900	1.0	
3	3.0	1.0	6	6900	5	600	1800	1.0	
4	3.0	1.0	3	3900	3	1100	1600	6.0	
...	...	...	...	...	...	...	...	...	
197423	1.0	4.0	3	1389	3	345	649	17.0	
197424	1.0	4.0	6	3010	4	405	825	12.0	
197425	1.0	4.0	5	1836	3	300	399	39.0	
197426	1.0	1.0	1	1175	1	535	535	7.0	
197427	1.0	1.0	4	2605	4	425	750	20.0	

186125 rows × 13 columns

In [101]:
```
y
```

Out[101]:
```
0         62.0
1         67.0
2         29.0
3         51.0
4         39.0
          ...
197423    65.0
197424    56.0
197425    50.0
197426    65.0
197427    37.0
Name: time_taken_minutes, Length: 186125, dtype: float64
```

In [102]:
```
#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
```

Creating baseline model RF to compare with Neural Networks

In [103]:
```
regressor = RandomForestRegressor()
regressor.fit(X_train, y_train)
```

Out[103]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [104]:
```
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 :  203.29293151608346
rmse :  14.25808302388801
mae: 10.898218183291227
```

In [105]:
```
r2_score(y_test, prediction)
```

Out[105]:
```
0.2670855268818707
```

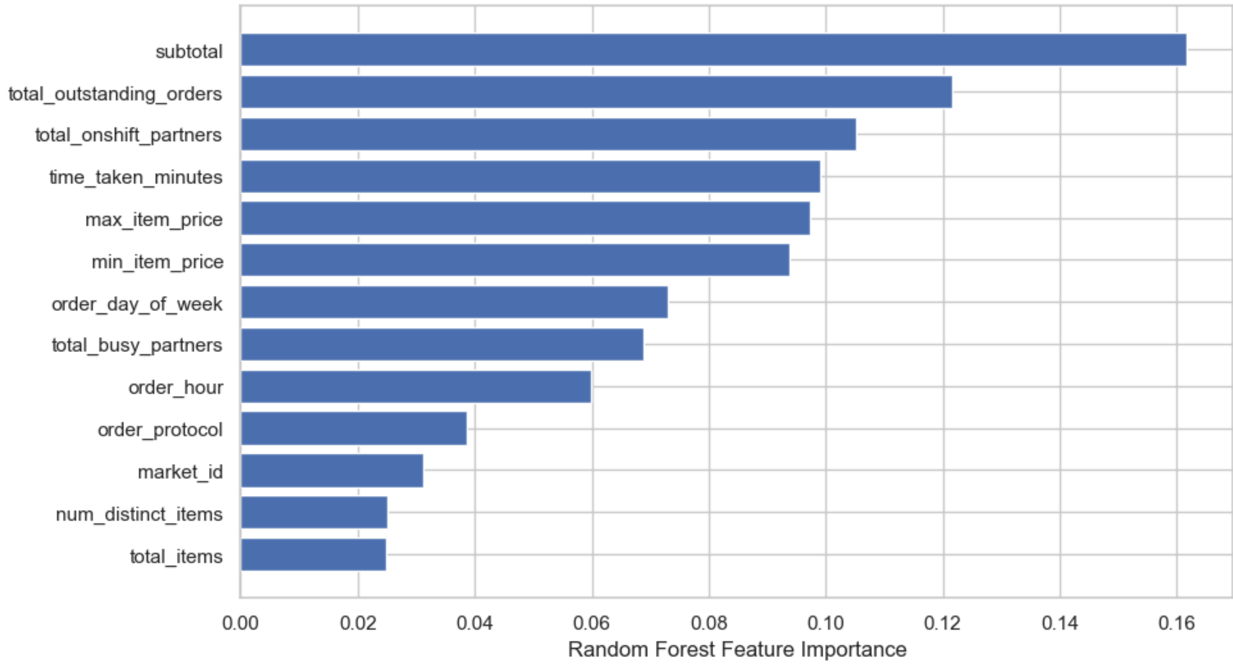
In [107]:
```
def MAPE(Y_actual,Y_Predicted):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
```

In [108]:
```
print("mape : ",MAPE(y_test, prediction))
```

```
mape :  26.222978362229632
```

In [109]:
```
sorted_idx = regressor.feature_importances_.argsort()
plt.barh(df3.columns[sorted_idx], regressor.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Out[109]: Text(0.5, 0, 'Random Forest Feature Importance')



Train-Test Splitting Standard Scaling

In [110]:
```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

# Initialize the MinMaxScaler
scaler = preprocessing.MinMaxScaler()

# Fit and transform the data
x_scaled = scaler.fit_transform(x)

# Split the scaled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x_scaled, y, test_size=0.2, random_state=42)
```

Creating Neural Network Architecture

In [111]:
```
model = Sequential()
model.add(Dense(11, kernel_initializer='normal'))
model.add(Dense(256, activation='relu'))
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='linear'))
```

Model Training

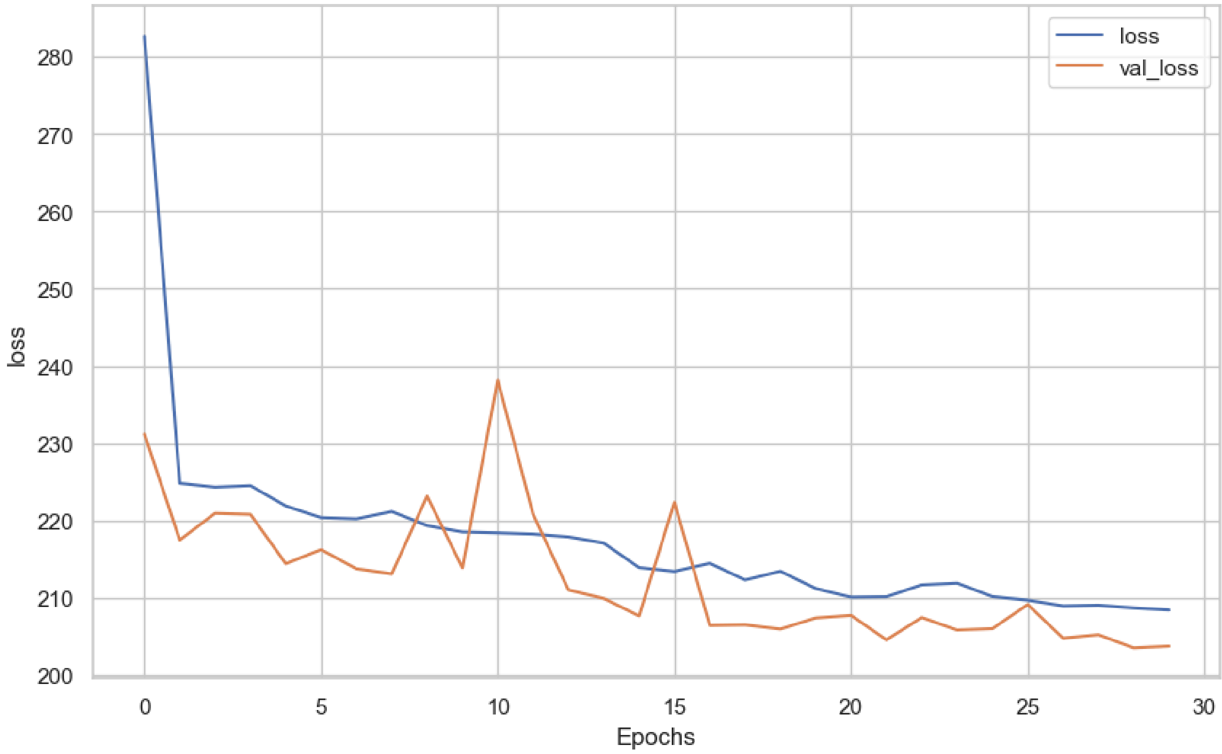
In [112]:
```
from tensorflow.keras.optimizers import Adam

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_data=(X_test, y_test)
```

```
Epoch 1/30
291/291 [==============================] - 6s 10ms/step - loss: 282.6307 - mse: 282.6307 - mae: 12.6141 - v
al_loss: 231.1850 - val_mse: 231.1850 - val_mae: 11.1970
Epoch 2/30
291/291 [==============================] - 3s 9ms/step - loss: 224.8160 - mse: 224.8160 - mae: 11.4173 - va
l_loss: 217.4656 - val_mse: 217.4656 - val_mae: 11.2839
Epoch 3/30
291/291 [==============================] - 3s 10ms/step - loss: 224.3059 - mse: 224.3059 - mae: 11.3947 - v
al_loss: 220.9363 - val_mse: 220.9363 - val_mae: 11.6191
Epoch 4/30
291/291 [==============================] - 3s 10ms/step - loss: 224.5054 - mse: 224.5054 - mae: 11.3996 - v
al_loss: 220.7947 - val_mse: 220.7947 - val_mae: 11.6464
Epoch 5/30
291/291 [==============================] - 3s 10ms/step - loss: 221.8747 - mse: 221.8747 - mae: 11.3358 - v
al_loss: 214.4131 - val_mse: 214.4131 - val_mae: 11.0570
Epoch 6/30
291/291 [==============================] - 3s 10ms/step - loss: 220.3652 - mse: 220.3652 - mae: 11.2856 - v
al_loss: 216.2168 - val_mse: 216.2168 - val_mae: 11.0922
Epoch 7/30
291/291 [==============================] - 3s 9ms/step - loss: 220.2037 - mse: 220.2037 - mae: 11.2854 - va
l_loss: 213.7458 - val_mse: 213.7458 - val_mae: 11.2511
Epoch 8/30
291/291 [==============================] - 3s 9ms/step - loss: 221.1623 - mse: 221.1623 - mae: 11.3061 - va
l_loss: 213.0859 - val_mse: 213.0859 - val_mae: 11.1524
Epoch 9/30
291/291 [==============================] - 3s 10ms/step - loss: 219.3359 - mse: 219.3359 - mae: 11.2506 - v
al_loss: 223.1522 - val_mse: 223.1522 - val_mae: 11.7961
Epoch 10/30
291/291 [==============================] - 3s 10ms/step - loss: 218.5172 - mse: 218.5172 - mae: 11.2343 - v
al_loss: 213.8713 - val_mse: 213.8713 - val_mae: 11.3567
Epoch 11/30
291/291 [==============================] - 3s 10ms/step - loss: 218.3957 - mse: 218.3957 - mae: 11.2331 - v
al_loss: 238.1378 - val_mse: 238.1378 - val_mae: 12.4312
Epoch 12/30
291/291 [==============================] - 3s 11ms/step - loss: 218.2340 - mse: 218.2340 - mae: 11.2343 - v
al_loss: 220.8149 - val_mse: 220.8149 - val_mae: 11.7375
Epoch 13/30
291/291 [==============================] - 3s 9ms/step - loss: 217.8660 - mse: 217.8660 - mae: 11.2131 - va
l_loss: 211.0166 - val_mse: 211.0166 - val_mae: 10.9998
Epoch 14/30
291/291 [==============================] - 3s 9ms/step - loss: 217.0804 - mse: 217.0804 - mae: 11.1905 - va
l_loss: 209.9097 - val_mse: 209.9097 - val_mae: 11.1757
Epoch 15/30
291/291 [==============================] - 3s 10ms/step - loss: 213.8961 - mse: 213.8961 - mae: 11.0969 - v
al_loss: 207.6500 - val_mse: 207.6500 - val_mae: 10.9583
Epoch 16/30
291/291 [==============================] - 3s 10ms/step - loss: 213.3959 - mse: 213.3959 - mae: 11.0918 - v
al_loss: 222.3456 - val_mse: 222.3456 - val_mae: 11.8455
Epoch 17/30
291/291 [==============================] - 3s 11ms/step - loss: 214.4584 - mse: 214.4584 - mae: 11.1235 - v
al_loss: 206.4343 - val_mse: 206.4343 - val_mae: 10.8411
Epoch 18/30
291/291 [==============================] - 3s 11ms/step - loss: 212.3076 - mse: 212.3076 - mae: 11.0493 - v
al_loss: 206.4986 - val_mse: 206.4986 - val_mae: 10.8529
Epoch 19/30
291/291 [==============================] - 3s 9ms/step - loss: 213.4192 - mse: 213.4192 - mae: 11.0920 - va
l_loss: 205.9552 - val_mse: 205.9552 - val_mae: 10.7876
Epoch 20/30
291/291 [==============================] - 3s 10ms/step - loss: 211.1879 - mse: 211.1879 - mae: 11.0295 - v
al_loss: 207.3801 - val_mse: 207.3801 - val_mae: 11.1824
Epoch 21/30
291/291 [==============================] - 3s 10ms/step - loss: 210.0776 - mse: 210.0776 - mae: 11.0004 - v
al_loss: 207.7153 - val_mse: 207.7153 - val_mae: 11.1545
Epoch 22/30
291/291 [==============================] - 3s 10ms/step - loss: 210.1328 - mse: 210.1328 - mae: 10.9997 - v
al_loss: 204.5477 - val_mse: 204.5477 - val_mae: 10.9081
Epoch 23/30
291/291 [==============================] - 3s 10ms/step - loss: 211.6617 - mse: 211.6617 - mae: 11.0298 - v
al_loss: 207.4345 - val_mse: 207.4345 - val_mae: 10.8461
Epoch 24/30
291/291 [==============================] - 3s 10ms/step - loss: 211.8860 - mse: 211.8860 - mae: 11.0475 - v
al_loss: 205.8262 - val_mse: 205.8262 - val_mae: 11.0192
Epoch 25/30
291/291 [==============================] - 3s 10ms/step - loss: 210.1552 - mse: 210.1552 - mae: 10.9974 - v
al_loss: 206.0043 - val_mse: 206.0043 - val_mae: 10.8366
Epoch 26/30
291/291 [==============================] - 3s 10ms/step - loss: 209.6816 - mse: 209.6816 - mae: 10.9877 - v
al_loss: 209.1212 - val_mse: 209.1212 - val_mae: 10.7518
Epoch 27/30
291/291 [==============================] - 3s 10ms/step - loss: 208.9299 - mse: 208.9299 - mae: 1[illegible]5 - v
al_loss: 204.7607 - val_mse: 204.7607 - val_mae: 10.7839
Epoch 28/30
291/291 [==============================] - 3s 10ms/step - loss: 209.0205 - mse: 209.0205 - mae: 10.9707 - v
al_loss: 205.2039 - val_mse: 205.2039 - val_mae: 11.0781
```

```
Epoch 29/30
291/291 [==============================] - 3s 10ms/step - loss: 208.6797 - mse: 208.6797 - mae: 10.9603 - v
al_loss: 203.5120 - val_mse: 203.5120 - val_mae: 10.8419
Epoch 30/30
291/291 [==============================] - 3s 9ms/step - loss: 208.4299 - mse: 208.4299 - mae: 10.9457 - va
l_loss: 203.7377 - val_mse: 203.7377 - val_mae: 10.8553
```

Comparing losses with epochs

In [113]:
```
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')
```



In [114]:
```
z= model.predict(X_test)
```

```
1164/1164 [==============================] - 2s 2ms/step
```

In [115]:
```
r2_score(y_test, z)
```

Out[115]: `0.26548217263663154`

MAE RMSE MSE values for Neural Networks

In [116]:
```
mse = mean_squared_error(y_test, z)
rmse = mse**.5
print("mse : ",mse)
print("rmse : ",rmse)
mae = mean_absolute_error(y_test, z)
print("mae : ",mae)
```

```
mse :  203.7376636051998
rmse :  14.27367029201669
mae :  10.855294681751463
```

Leading Questions:

Defining the problem statements and where can this and modifications of this be used?

List 3 functions the pandas datetime provides with one line explanation.

Short note on datetime, timedelta, time span (period)

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

Name 3 outlier removal methods?

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

Why is scaling required for neural networks?

Briefly explain your choice of optimizer.

Which activation function did you use and why?

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

List 3 functions the pandas datetime provides with one line explanation.

Pandas datetime provides several functions for handling date and time data. Three key functions include:

pd.to_datetime(): This function is used to convert input to datetime. It can parse a wide variety of formats and return a datetime object.

dt.hour: This function extracts the hour component from a datetime object, allowing for easy manipulation and analysis of time-based data.

dt.dayofweek: This function returns the day of the week for a given datetime object, where Monday is represented as 0 and Sunday as 6. It is useful for analyzing patterns based on the day of the week.

These functions are essential for manipulating and extracting meaningful insights from date and time data within a pandas DataFrame.

Short note on datetime, timedelta, time span (period)

Datetime, timedelta, and time span (period) are essential concepts in handling date and time data within the context of data analysis and machine learning.

Datetime refers to a specific point in time and is represented by the datetime data type in Python. It includes both date and time components, allowing for precise temporal calculations and comparisons. In the provided context, the datetime data type is used to represent the 'created_at' and 'actual_delivery_time' columns, enabling the analysis of time-based patterns and the calculation of time differences.

Timedelta represents a duration of time, such as a difference between two datetimes. It allows for the manipulation and arithmetic operations on time durations, such as adding or subtracting time intervals. In the context of the project, timedelta can be used to calculate the time taken for delivery by subtracting the 'created_at' from the 'actual_delivery_time'.

Time span, also known as a period, refers to a specific range of time, such as a day, month, or year. It is useful for analyzing data over specific time intervals and performing time-based aggregations. In the project, time spans can be utilized for grouping and aggregating delivery time data based on specific time periods, enabling insights into temporal trends and patterns.

These concepts are fundamental for effectively handling and analyzing date and time data, and they play a crucial role in the development of the regression model for estimating delivery time within the context of the Porter Neural Networks Regression project.

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

Checking for outliers in the data is essential for several reasons. Outliers, which are data points that significantly differ from other observations, can have a substantial impact on statistical analyses and machine learning models. By identifying and addressing outliers, we can ensure the accuracy and reliability of our analyses and models. Outliers can skew the distribution of the data, leading to biased estimates of statistical parameters such as the mean and standard deviation. In the context of machine learning, outliers can adversely affect the performance of models, particularly regression models, by influencing the estimation of coefficients and predictions. Therefore, detecting and handling outliers is crucial for producing robust and accurate analyses and models. In the provided context, the use of Local Outlier Factor (LOF) is a method for identifying and handling outliers in the dataset, which is a critical step in preparing the data for regression analysis and model training.

Name 3 outlier removal methods?

Three common outlier removal methods include:

Z-Score Method: This method involves calculating the z-score for each data point and removing those that fall outside a specified threshold, typically set at a z-score of 3 or -3.

Interquartile Range (IQR) Method: The IQR method involves determining the IQR for the dataset and removing data points below Q1 - 1.5 * IQR or above Q3 + 1.5 * IQR, where Q1 and Q3 represent the first and third quartiles, respectively.

Local Outlier Factor (LOF): LOF is a method that identifies outliers by comparing the local density of a data point to the density of its neighbors. Data points with significantly lower density compared to their neighbors are considered outliers and can be removed.

In the provided context, the document mentions the use of the Local Outlier Factor (LOF) method for removing outliers from the dataset. This method involves assigning an anomaly score to each data point and removing those with scores indicating outlier behavior

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

Based on the context provided, the problem at hand involves training a regression model to estimate delivery time based on various features related to orders, restaurants, and delivery partners within the context of Porter's intra-city logistics operations. Classical machine learning methods suitable for addressing this problem include:

Linear Regression: This method is a fundamental and widely used approach for modeling the relationship between independent variables and a continuous dependent variable, making it suitable for estimating delivery time based on the given features.

Decision Trees: Decision tree algorithms can be employed to predict delivery time by recursively partitioning the data based on the features and creating a tree-like model to make predictions.

Random Forest: Random Forest is an ensemble learning method that utilizes multiple decision trees to improve predictive accuracy and can be effective for estimating delivery time by leveraging the collective predictions of multiple trees.

These classical machine learning methods can be applied to the dataset to develop regression models for estimating delivery time, aligning with the problem statement outlined in the context. Additionally, other methods such as Support Vector Machines (SVM) and Gradient Boosting can also be considered based on the specific characteristics of the dataset and the nature of the problem.

Why is scaling required for neural networks?

Scaling is essential for neural networks due to the sensitivity of their performance to the scale of input features. Neural networks, particularly those utilizing gradient-based optimization algorithms, are sensitive to the magnitude of input features. When features are not scaled, those with larger scales can disproportionately influence the model's learning process, leading to slower convergence and suboptimal performance.

By scaling the input features, we ensure that all features contribute equally to the model's learning process. This helps in achieving faster convergence during training and prevents certain features from dominating the learning process solely based on their scale. Additionally, scaling can also aid in preventing numerical instability and improving the overall generalization of the neural network model.

In the context of the Porter Neural Networks Regression project, the use of the MinMaxScaler from the sklearn library indicates the application of feature scaling to the input data before training the neural network model. This step is crucial for ensuring the effective learning and performance of the neural network in estimating delivery time based on the provided features.

Briefly explain your choice of optimizer.

The choice of optimizer is a critical decision in training neural network models. In the provided context, the Adam optimizer is utilized for model training. Adam, short for Adaptive Moment Estimation, is a popular optimization algorithm that combines the benefits of both AdaGrad and RMSProp. It is well-suited for training deep learning models due to its ability to adapt learning rates for each parameter, leading to efficient convergence and improved performance.

Adam optimizer maintains separate learning rates for each parameter and adjusts them based on the first and second moments of the gradients. This adaptive learning rate mechanism allows Adam to handle sparse gradients and noisy data effectively, making it suitable for a wide range of neural network architectures and datasets.

In summary, the choice of the Adam optimizer in the context of training the neural network model for estimating delivery time is driven by its adaptive learning rate capabilities, which can lead to efficient convergence and improved model performance.

Which activation function did you use and why?

Based on the provided context, the activation function used in the neural network model is the Rectified Linear Unit (ReLU). This is evident from the code snippet on page 24, which specifies the activation function as 'relu' for the hidden layers of the neural network model. The ReLU activation function is commonly used in deep learning models due to its ability to introduce non-linearity and address the vanishing gradient problem, making it an effective choice for improving the learning capacity of neural networks.

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

A neural network can perform well on a large dataset due to its ability to learn complex patterns and representations from a vast amount of data. The depth and capacity of neural networks allow them to capture intricate relationships and features within the data, which can be beneficial when dealing with a large and diverse dataset. Additionally, the hierarchical nature of neural networks enables automatically extract relevant features and representations from the input data, making them well-suited for handling the co and richness of large datasets.

Furthermore, the scalability of neural networks allows them to effectively process and learn from large volumes of data, leveraging parallel processing and distributed computing to handle the computational demands of extensive datasets. This capability enables neural networks to effectively generalize from large datasets, leading to improved performance and predictive accuracy.

In the context of the Porter Neural Networks Regression project, the use of a neural network for estimating delivery time is advantageous when dealing with a large dataset containing diverse features related to orders, restaurants, and delivery partners. The neural network's capacity to learn from the extensive dataset and capture complex relationships aligns with the requirements of the problem statement.

In [ ]: