Porter : Delivery Time Estimation

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.

Problem statement

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 esimation, 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

In [ ]:

```

```

Broad steps in the notebook

- load the data and understand the features
- feature engineering creating target variable(time taken for each order)
- cleaning the data and visualization
- preparing the data for training
- random forest regression
- neural network regression
- comarision of both ways

Importing libraries

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /opt/conda/lib/python3.10/site-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (23.5.26)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /opt/conda/lib/python3.10/site-packages (from tensorflow)
(0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (3.10.0)
Requirement already satisfied: libclang>=13.0.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (16.0.6)
Requirement already satisfied: ml-dtypes~=0.2.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.26.4)
Requirement already satisfied: opt-einsum>=2.3.2 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /opt/conda/lib/python3.10/site-packages (from tensorflow) (21.3)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /opt/conda/l
ib/python3.10/site-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in /opt/conda/lib/python3.10/site-packages (from tensorflow) (69.0.3)
Requirement already satisfied: six>=1.12.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (4.9.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /opt/conda/lib/python3.10/site-packages (from tensorflo
w) (0.35.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (1.60.0)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /opt/conda/lib/python3.10/site-packages (from tensorflow) (2.15.1)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /opt/conda/lib/python3.10/site-packages (from tensorflow)
(2.15.0)
Collecting keras<2.16,>=2.15.0 (from tensorflow)
  Downloading keras-2.15.0-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /opt/conda/lib/python3.10/site-packages (from astunparse>=1.6.0->tensorflo
w) (0.42.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /opt/conda/lib/python3.10/site-packages (from tensorboard<2.16,>=2.15->
tensorflow) (2.26.1)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /opt/conda/lib/python3.10/site-packages (from tensorboard<2.16,>
=2.15->tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in /opt/conda/lib/python3.10/site-packages (from tensorboard<2.16,>=2.15->tensor
flow) (3.5.2)
Requirement already satisfied: requests<3,>=2.21.0 in /opt/conda/lib/python3.10/site-packages (from tensorboard<2.16,>=2.15->te
nsorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /opt/conda/lib/python3.10/site-packages (from tensorboa
rd<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /opt/conda/lib/python3.10/site-packages (from tensorboard<2.16,>=2.15->tensor
```

```
flow) (3.0.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/python3.10/site-packages (from packaging->tensorflow)
(3.1.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /opt/conda/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->t
ensorboard<2.16,>=2.15->tensorflow) (4.2.4)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /opt/conda/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->te
nsorboard<2.16,>=2.15->tensorflow) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /opt/conda/lib/python3.10/site-packages (from google-auth<3,>=1.6.3->tensorboar
d<2.16,>=2.15->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /opt/conda/lib/python3.10/site-packages (from google-auth-oauthlib<
2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/site-packages (from requests<3,>=2.21.0->t
ensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorboard<
2.16,>=2.15->tensorflow) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorb
oard<2.16,>=2.15->tensorflow) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorb
oard<2.16,>=2.15->tensorflow) (2024.2.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /opt/conda/lib/python3.10/site-packages (from werkzeug>=1.0.1->tensorboard<
2.16,>=2.15->tensorflow) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /opt/conda/lib/python3.10/site-packages (from pyasn1-modules>=0.2.1->goo
gle-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.5.1)
Requirement already satisfied: oauthlib>=3.0.0 in /opt/conda/lib/python3.10/site-packages (from requests-oauthlib>=0.7.0->googl
e-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (3.2.2)
Downloading keras-2.15.0-py3-none-any.whl (1.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 19.0 MB/s eta 0:00:0000:010:01
Installing collected packages: keras
  Attempting uninstall: keras
    Found existing installation: keras 3.2.1
    Uninstalling keras-3.2.1:
      Successfully uninstalled keras-3.2.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is th
e source of the following dependency conflicts.
tensorflow-decision-forests 1.8.1 requires wurlitzer, which is not installed.
Successfully installed keras-2.15.0
```

In [4]:
```
#for reading and handling the data
import pandas as pd
import numpy as np
import os
```

```python
#for visualizinng 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
```

```
2024-05-21 14:21:30.418735: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: A
ttempting to register factory for plugin cuDNN when one has already been registered
2024-05-21 14:21:30.418920: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: At
tempting to register factory for plugin cuFFT when one has already been registered
2024-05-21 14:21:30.583317: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory:
Attempting to register factory for plugin cuBLAS when one has already been registered
```

In [5]:
```python
sns.set(rc={'figure.figsize':(11.7,8.27)})
```

Loading the data from kaggle

In [6]:
```
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/porter-delivery-time-estimation-dataset/porter_data.csv
```

In [7]:
```
df=pd.read_csv('/kaggle/input/porter-delivery-time-estimation-dataset/porter_data.csv')
```

Printing the head and information of the data to get an understanding of it

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

Out[8]:

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item
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 [9]:
```
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
```

Data preprocessing

Feature engineering

We have the time at which the order was placed and time at which it was delivered, 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

The time stamps that we have now are in object format and need to be convertd to datetime format for easily working with them as intended. The **pandas** datetime function checks if the data is in correct format for it and also understands the order of the data and converts accordingly

In [10]:
```python
df['created_at']=pd.to_datetime(df['created_at'])
df['actual_delivery_time']=pd.to_datetime(df['actual_delivery_time'])
```

In [11]:
```python
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 [12]:
```python
df['time_taken']=df['actual_delivery_time'] - df['created_at']
```

In [13]:
```python
df.head()
```

Out[13]:

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item
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 [14]:

```
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 [15]:
```
df['time_taken_mins']=pd.to_timedelta(df['time_taken'])/pd.Timedelta('60s')
```

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

Out[16]:

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item
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 placed and which day of the week it was

In [17]:

```
df['hour']=df['created_at'].dt.hour
df['day']=df['created_at'].dt.dayofweek
```

In [18]:

```
df.head()
```

Out[18]:

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item
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	

Dropping the columns that are no longer required

In [19]:
```
df.drop(['time_taken','created_at','actual_delivery_time'],axis=1,inplace=True)
```

Checking null values in the data

In [20]:
```
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  int32  
 14  day                                           175777 non-null  int32  
dtypes: float64(7), int32(2), int64(6)
memory usage: 18.8 MB
```

In [21]:
```
df.isna().sum()
```

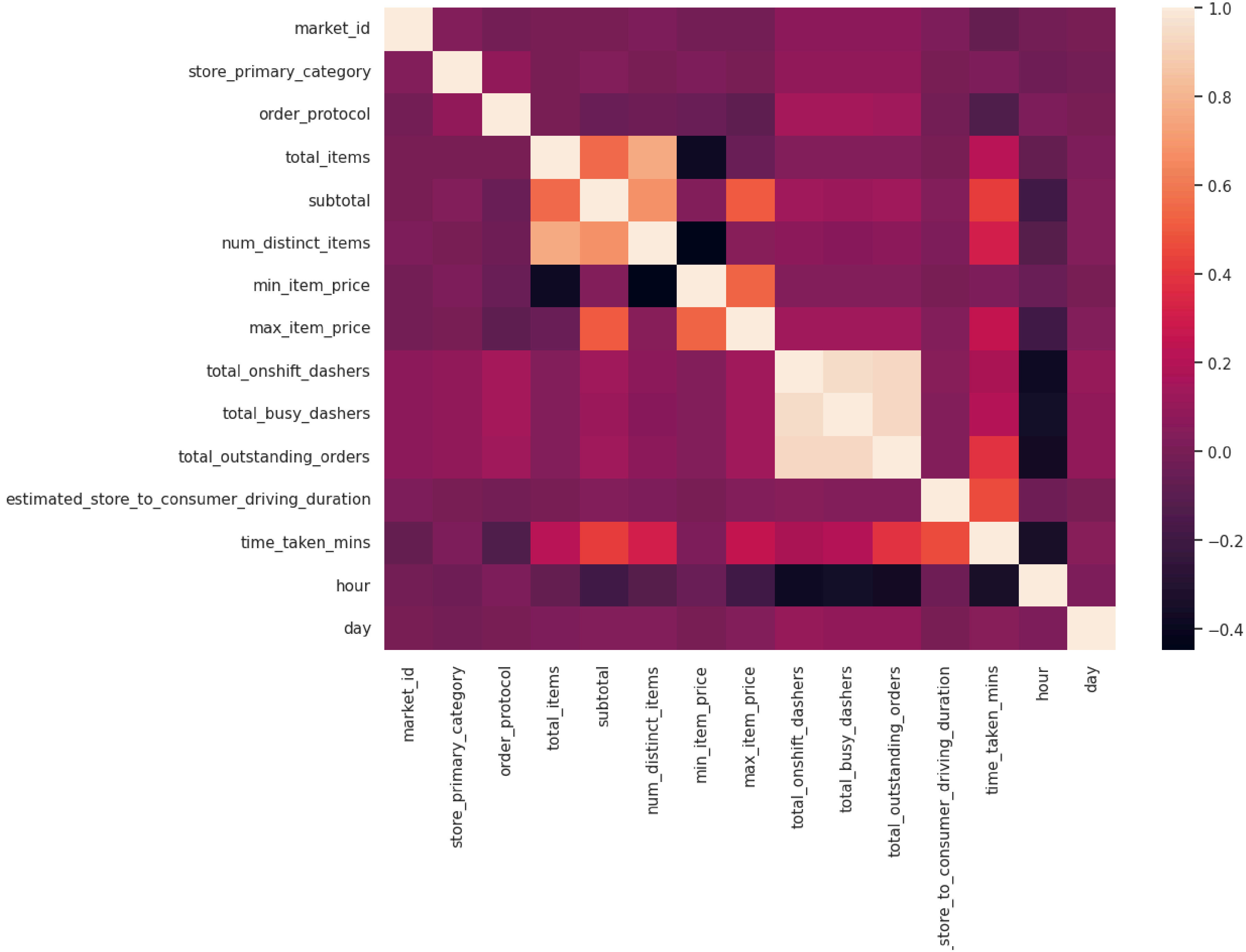
Out[21]:
```
market_id                                       0
store_primary_category                          0
order_protocol                                  0
total_items                                     0
subtotal                                        0
num_distinct_items                              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
```

dropping null values from the data(if present)

Plotting correlation to get an idea of the data

In [22]:
```
sns.heatmap(df.corr())
```

Out[22]:
```
<Axes: >
```



market_id
store_primary_category
order_protocol
total_items
subtotal
num_distinct_items
min_item_price
max_item_price
total_onshift_dashers
total_busy_dashers
total_outstanding_orders
estimated_store_to_consumer_driving_duration
time_taken_mins
hour
day
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

estimated_

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

In [23]:
```python
df['store_primary_category']=df['store_primary_category'].astype('category').cat.codes
```

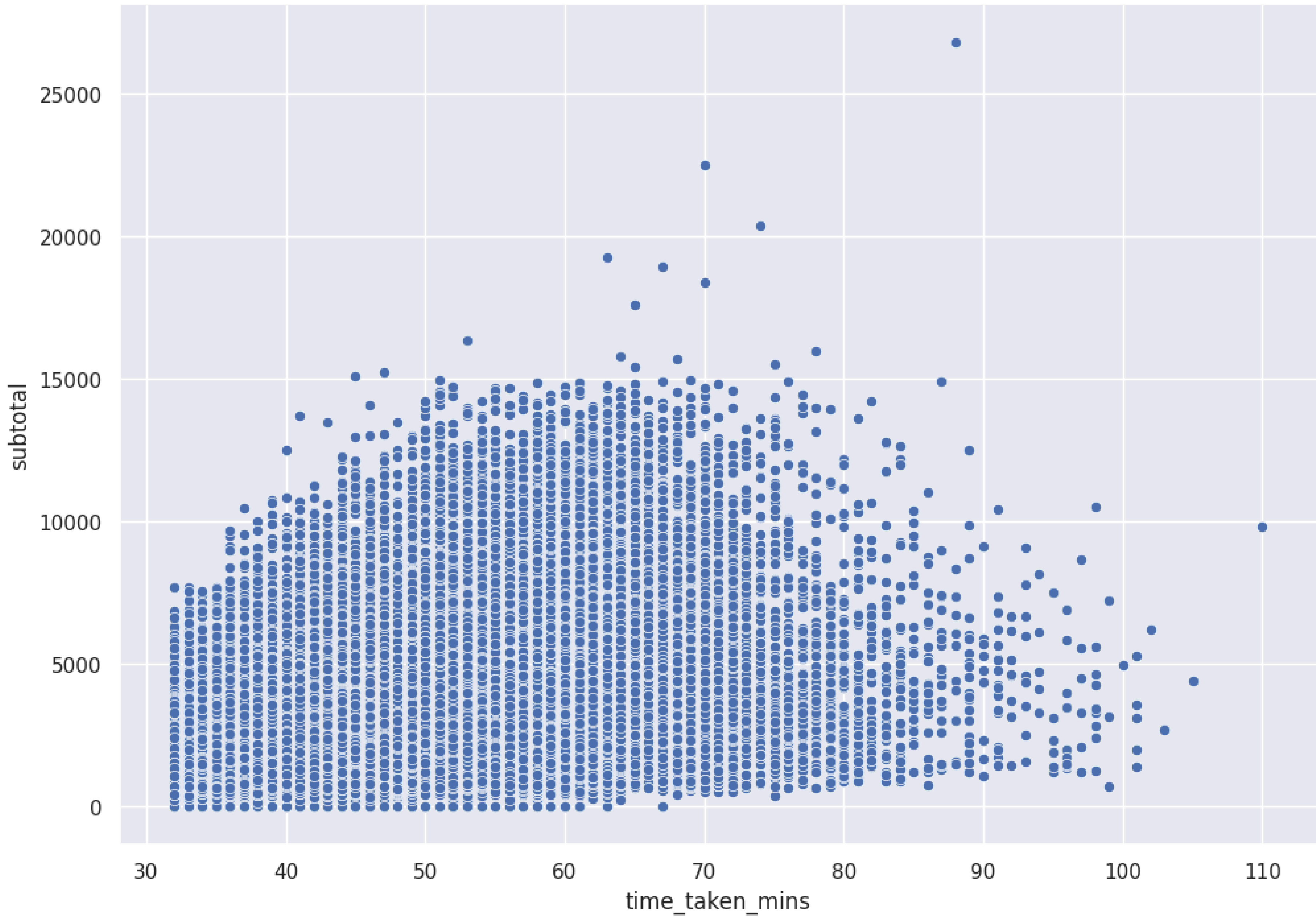
In [24]:
```python
df.info()
```

```
<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  int32  
 14  day                                           175777 non-null  int32  
dtypes: float64(7), int32(2), int64(5), int8(1)
memory usage: 17.6 MB
```

Data Visualization and Cleaning

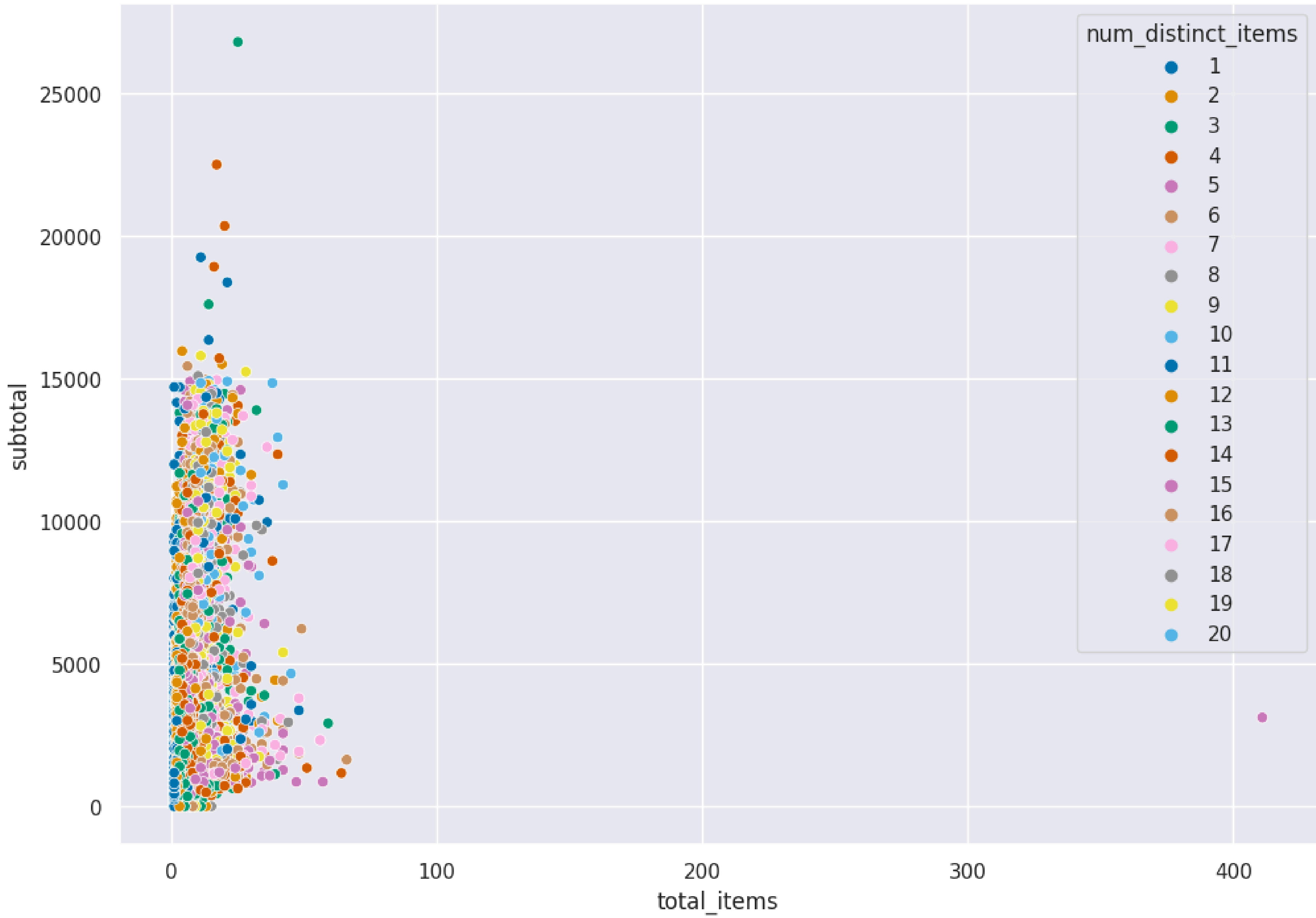
In [25]:
```python
sns.scatterplot(x='time_taken_mins',y='subtotal',data=df)
```

Out[25]:
```
<Axes: xlabel='time_taken_mins', ylabel='subtotal'>
```



In [26]:
```
sns.scatterplot(x='total_items',y='subtotal',hue='num_distinct_items',palette='colorblind',data=df)
```

Out[26]: <Axes: xlabel='total_items', ylabel='subtotal'>



In [27]:
```python
from sklearn.neighbors import LocalOutlierFactor
import matplotlib.pyplot as plt
model1=LocalOutlierFactor()
#model1.fit(df)
df['lof_anomaly_score']=model1.fit_predict(df)
```

In [28]:
```python
print("number of outliers : ",(len(df.loc[(df['lof_anomaly_score'] == -1)])))
df=df.loc[(df['lof_anomaly_score'] == 1)]
```

```
number of outliers :  831
```

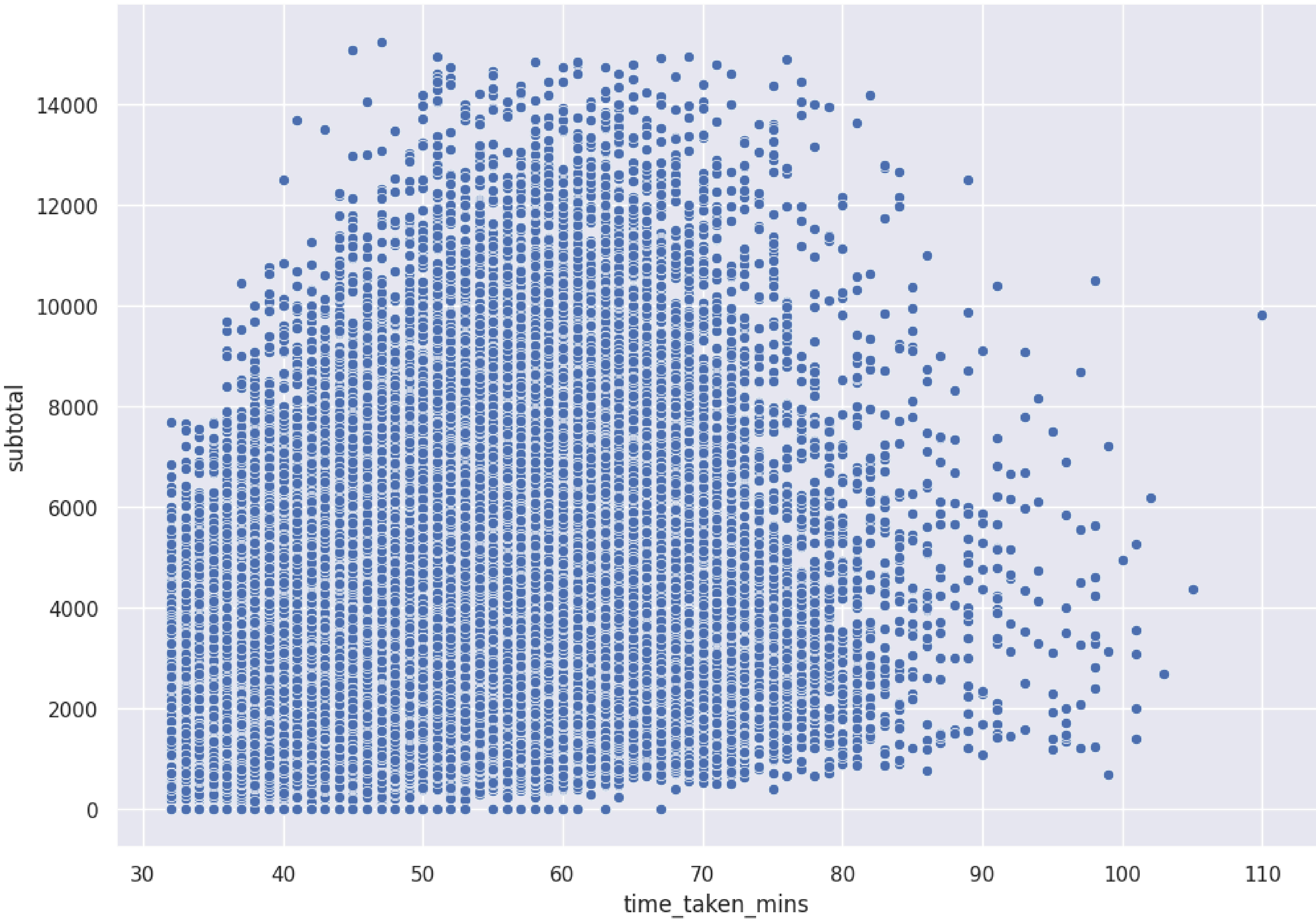
In [29]:
```python
df.drop(['lof_anomaly_score'],axis=1,inplace=True)
```

In [30]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 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  int32  
 14  day                                           174946 non-null  int32  
dtypes: float64(7), int32(2), int64(5), int8(1)
memory usage: 18.9 MB
```

In [31]:
```python
sns.scatterplot(x='time_taken_mins',y='subtotal',data=df)
```

Out[31]: <Axes: xlabel='time_taken_mins', ylabel='subtotal'>



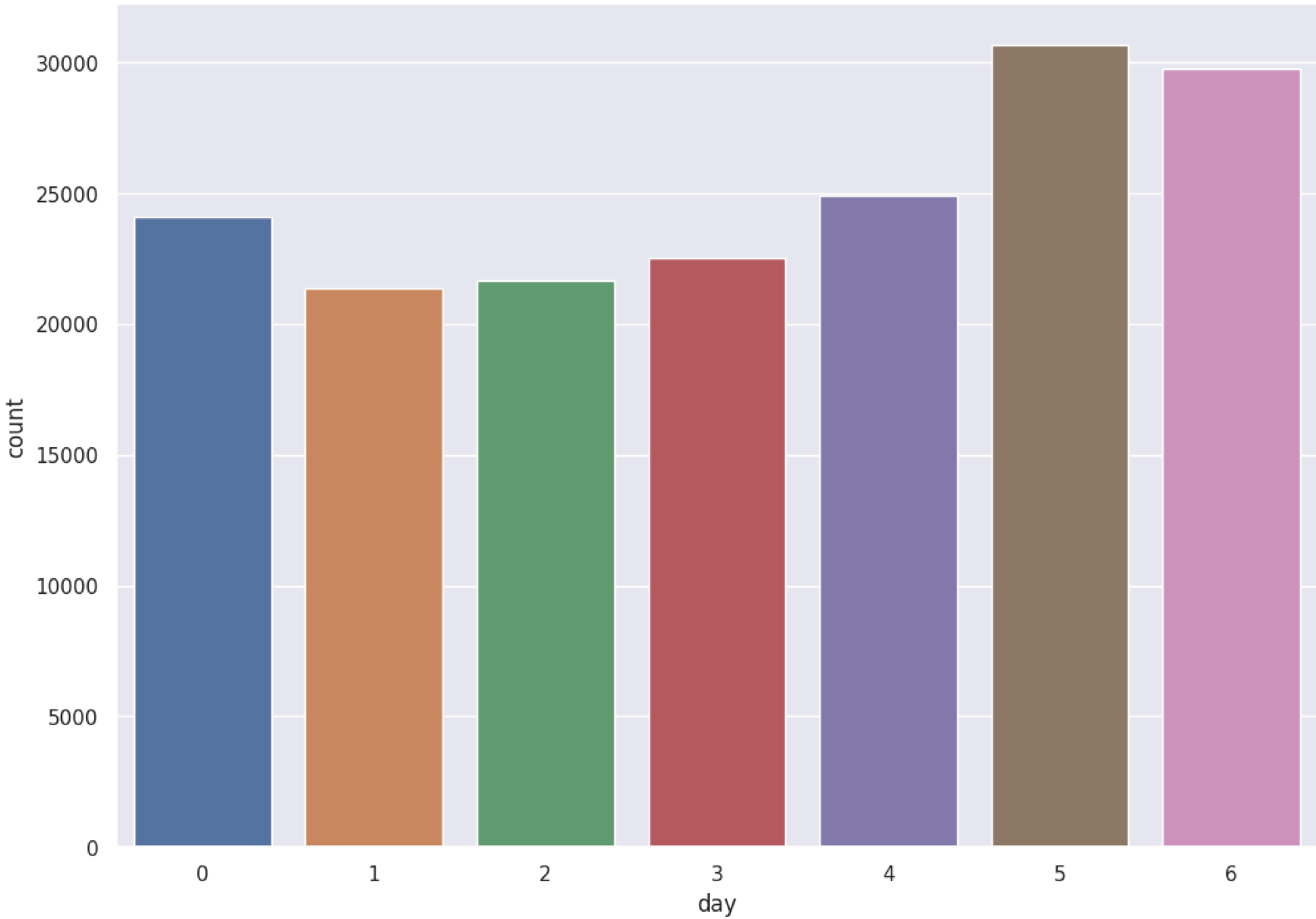
We can see that after removing outliers our data is looking better

In [32]:
```
df.columns
```

Out[32]:
```
Index(['market_id', 'store_primary_category', 'order_protocol', 'total_items',
       'subtotal', 'num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_dashers', 'total_busy_dashers',
       'total_outstanding_orders',
       'estimated_store_to_consumer_driving_duration', 'time_taken_mins',
       'hour', 'day'],
      dtype='object')
```

In [33]:
```
sns.countplot(x=df.day)
```

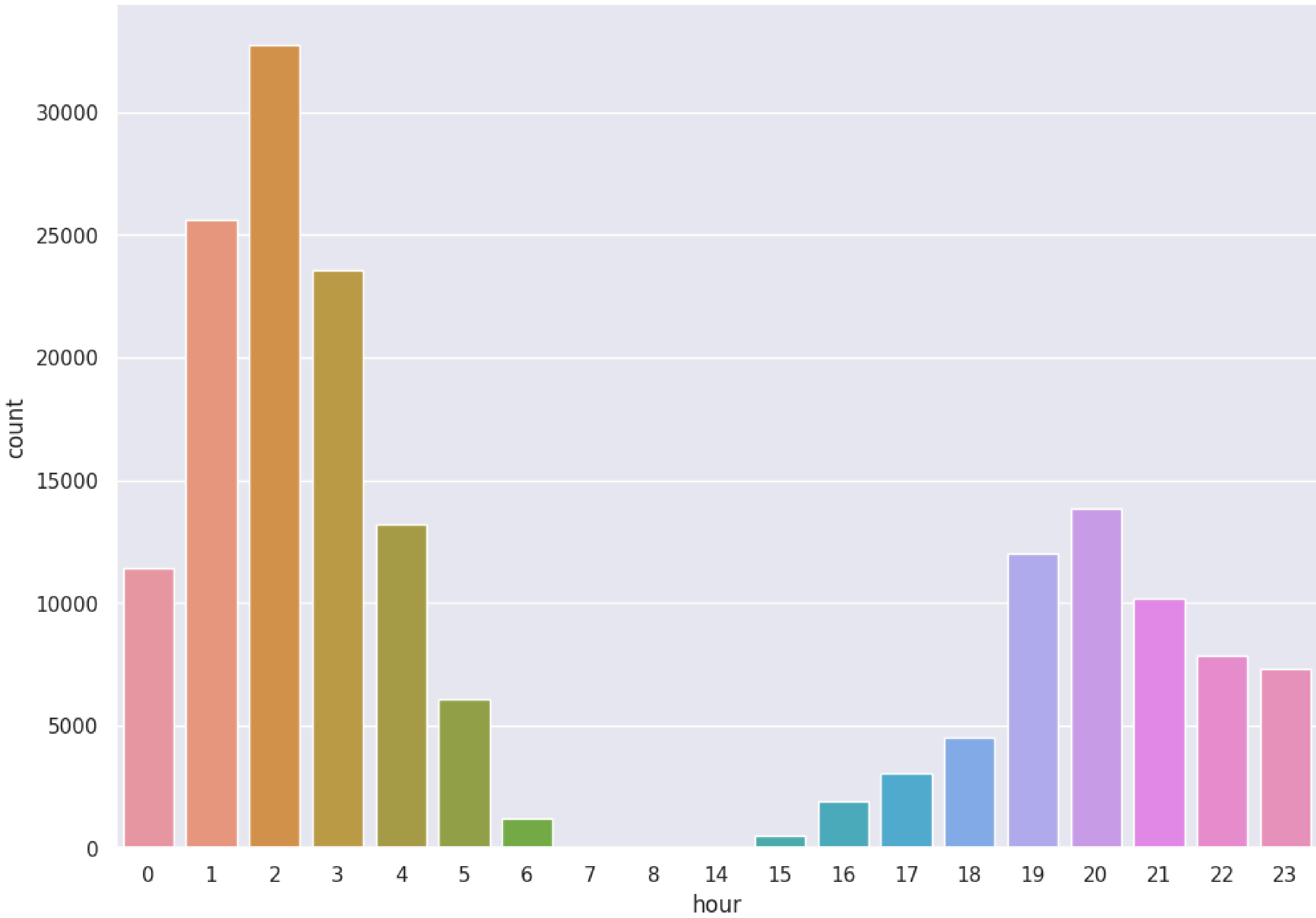
Out[33]:
```
<Axes: xlabel='day', ylabel='count'>
```



a little more orders on the weekends

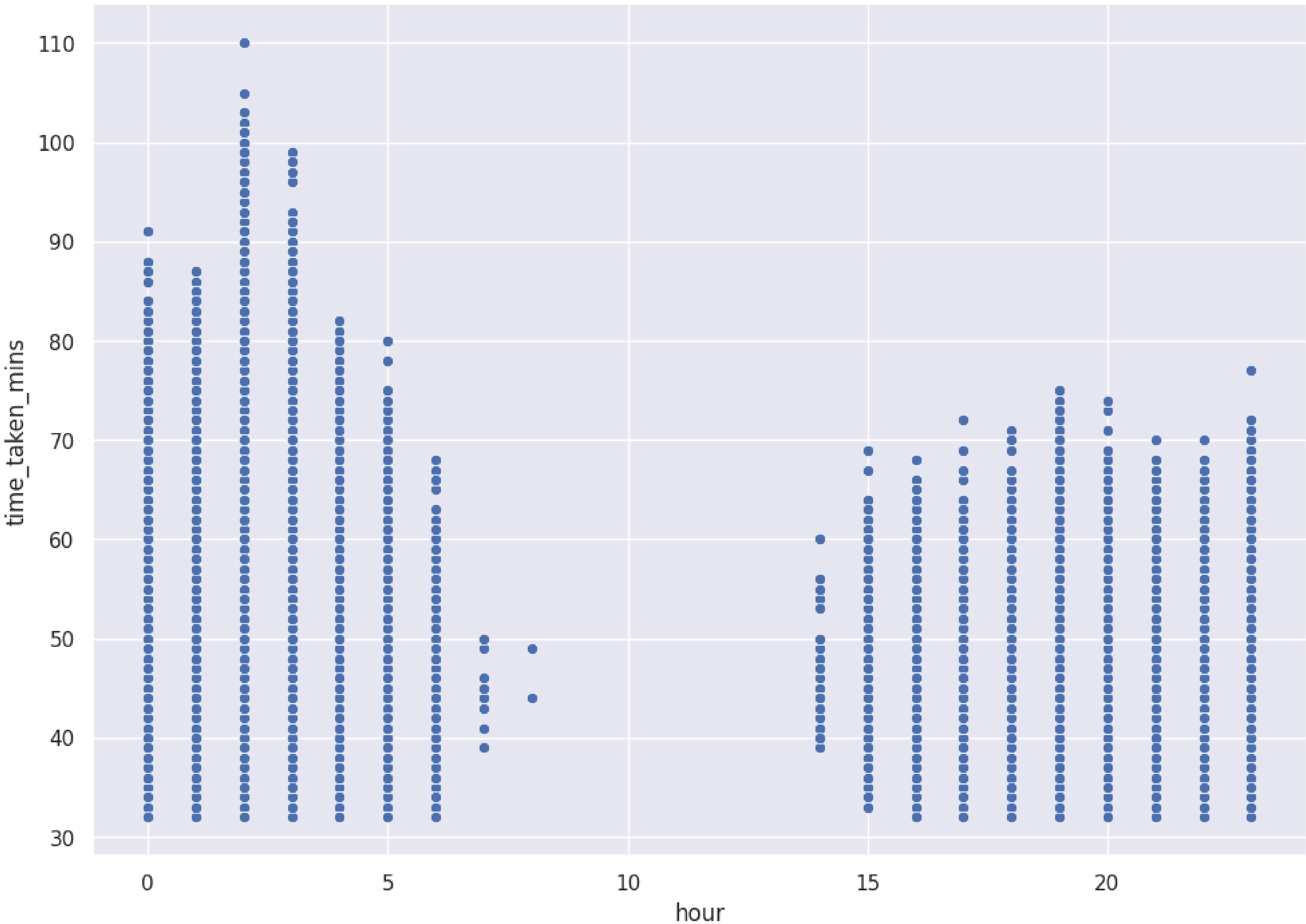
In [34]:
```
sns.countplot(x=df.hour)
```

Out[34]:
```
<Axes: xlabel='hour', ylabel='count'>
```



In [35]:
```
sns.scatterplot(x='hour',y='time_taken_mins',data=df)
```

Out[35]: <Axes: xlabel='hour', ylabel='time_taken_mins'>



Data Splitting and Modelling

In [36]:
```
y=df['time_taken_mins']
x=df.drop(['time_taken_mins'],axis=1)
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [37]:
```
x.head()
```

Out[37]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onsl
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	

Random Forest

In [38]:
```
regressor=RandomForestRegressor()
regressor.fit(X_train,y_train)
```

Out[38]:
```
▾ RandomForestRegressor
RandomForestRegressor()
```

In [39]:
```
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("mase : ",mae)
```

```
mse :  3.236834567019148
rmse :  1.7991204981932556
mase :  1.285313803943984
```

In [40]:
```
r2_score(y_test,prediction)
```

Out[40]:
```
0.9623800597608164
```

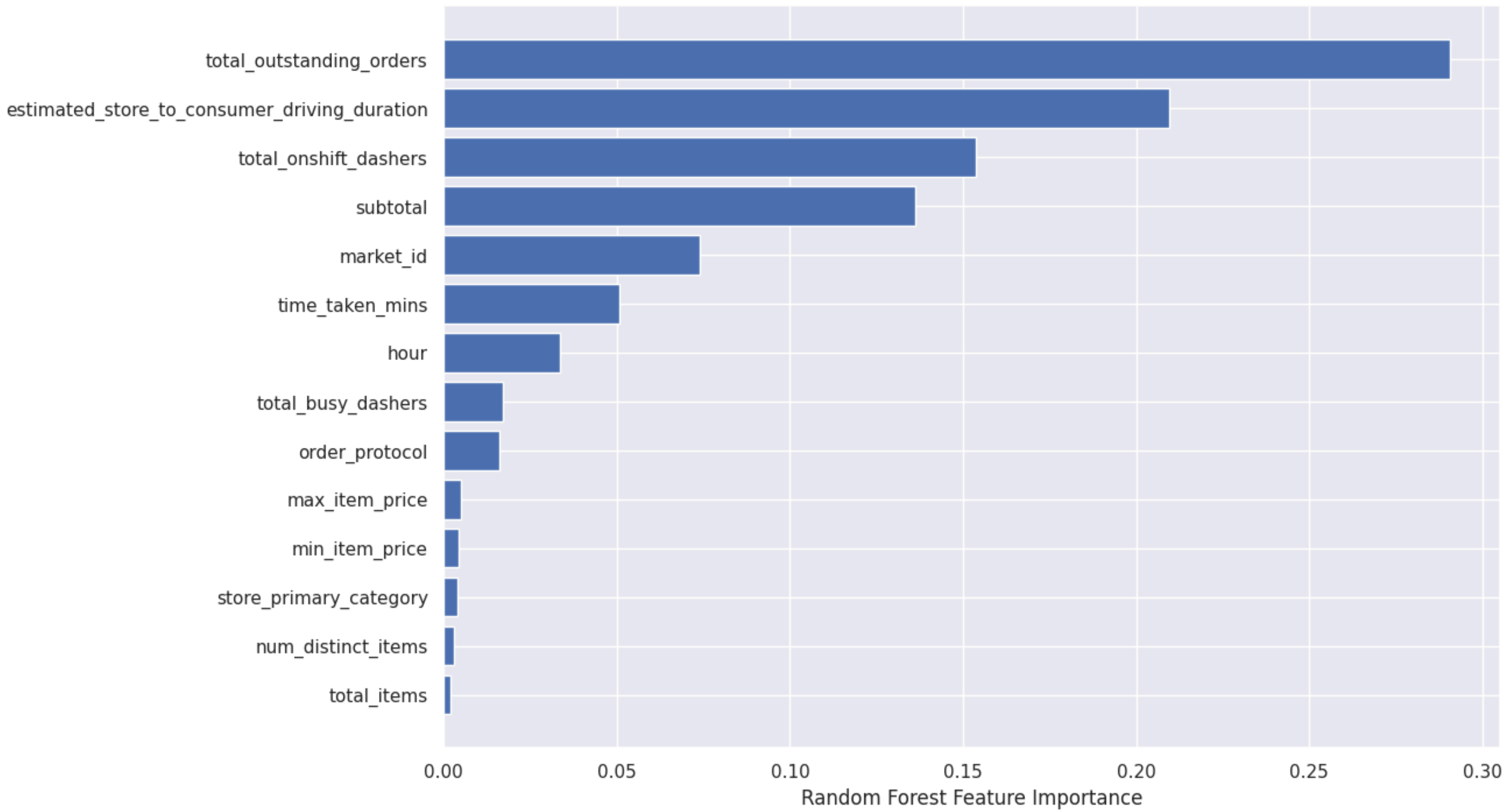
In [41]:
```
def MAPE(Y_actual,Y_Predicted):
    mape=np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
```

In [42]:
```
print("mape : ",MAPE(y_test,prediction))
```

```
mape :  2.76832678890719
```

In [43]:
```
sorted_idx=regressor.feature_importances_.argsort()
plt.barh(df.columns[sorted_idx],regressor.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
```

Out[43]:
```
Text(0.5, 0, 'Random Forest Feature Importance')
```



Neural Networks

Scalling the data to feed before neural network

In [44]:
```
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)
```

We will build a simple neural network to train our regression model it is a sequential model with three layers,

we have kept the number of nodes in the first layers equal to the number of input columns, and for the subsequent layers 512,1024,256, which can we changed or experimented with

the activation for the layers is kept as relu because it is a great non linear activation function that works for most cases, we could have used leaky relu if we see gradient vanishing.

the last layer has one node because it will give the single result that is our delivery time and the activation function for that should be linear

In [45]:
```
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'))
```

we use adam optimizer which is extention to classic schostic gradient descent(SGD) algorithm, but handles much of its drawbacks

Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training.

In [46]:
```
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 [==============================] - 9s 34ms/step - loss: 82.8328 - mse: 82.8328 - mae: 5.3642 - val_loss: 5.3444 - val_m
se: 5.3444 - val_mae: 1.7327
Epoch 2/30
219/219 [==============================] - 7s 33ms/step - loss: 4.0091 - mse: 4.0091 - mae: 1.5198 - val_loss: 8.0985 - val_ms
e: 8.0985 - val_mae: 2.5329
Epoch 3/30
219/219 [==============================] - 8s 35ms/step - loss: 2.7601 - mse: 2.7601 - mae: 1.2164 - val_loss: 0.8793 - val_ms
e: 0.8793 - val_mae: 0.7379
Epoch 4/30
219/219 [==============================] - 7s 32ms/step - loss: 2.0863 - mse: 2.0863 - mae: 1.1214 - val_loss: 0.7736 - val_ms
e: 0.7736 - val_mae: 0.6892
Epoch 5/30
219/219 [==============================] - 7s 33ms/step - loss: 1.6975 - mse: 1.6975 - mae: 1.0182 - val_loss: 0.6698 - val_ms
e: 0.6698 - val_mae: 0.6548
Epoch 6/30
219/219 [==============================] - 7s 33ms/step - loss: 1.3965 - mse: 1.3965 - mae: 0.9339 - val_loss: 0.3191 - val_ms
e: 0.3191 - val_mae: 0.4423
Epoch 7/30
219/219 [==============================] - 7s 34ms/step - loss: 1.4549 - mse: 1.4549 - mae: 0.9450 - val_loss: 3.7672 - val_ms
e: 3.7672 - val_mae: 1.8165
Epoch 8/30
219/219 [==============================] - 7s 33ms/step - loss: 1.5464 - mse: 1.5464 - mae: 0.9482 - val_loss: 2.7575 - val_ms
e: 2.7575 - val_mae: 1.4824
Epoch 9/30
219/219 [==============================] - 7s 33ms/step - loss: 0.8889 - mse: 0.8889 - mae: 0.7255 - val_loss: 0.1968 - val_ms
e: 0.1968 - val_mae: 0.3481
Epoch 10/30
219/219 [==============================] - 7s 33ms/step - loss: 1.0594 - mse: 1.0594 - mae: 0.7941 - val_loss: 0.2430 - val_ms
e: 0.2430 - val_mae: 0.3968
Epoch 11/30
219/219 [==============================] - 7s 33ms/step - loss: 1.2394 - mse: 1.2394 - mae: 0.8934 - val_loss: 2.6022 - val_ms
e: 2.6022 - val_mae: 1.5342
Epoch 12/30
219/219 [==============================] - 7s 34ms/step - loss: 0.8185 - mse: 0.8185 - mae: 0.7110 - val_loss: 0.2280 - val_ms
e: 0.2280 - val_mae: 0.3579
Epoch 13/30
219/219 [==============================] - 7s 32ms/step - loss: 0.9784 - mse: 0.9784 - mae: 0.7821 - val_loss: 0.1640 - val_ms
e: 0.1640 - val_mae: 0.3243
Epoch 14/30
219/219 [==============================] - 7s 33ms/step - loss: 0.8109 - mse: 0.8109 - mae: 0.7088 - val_loss: 0.1719 - val_ms
```

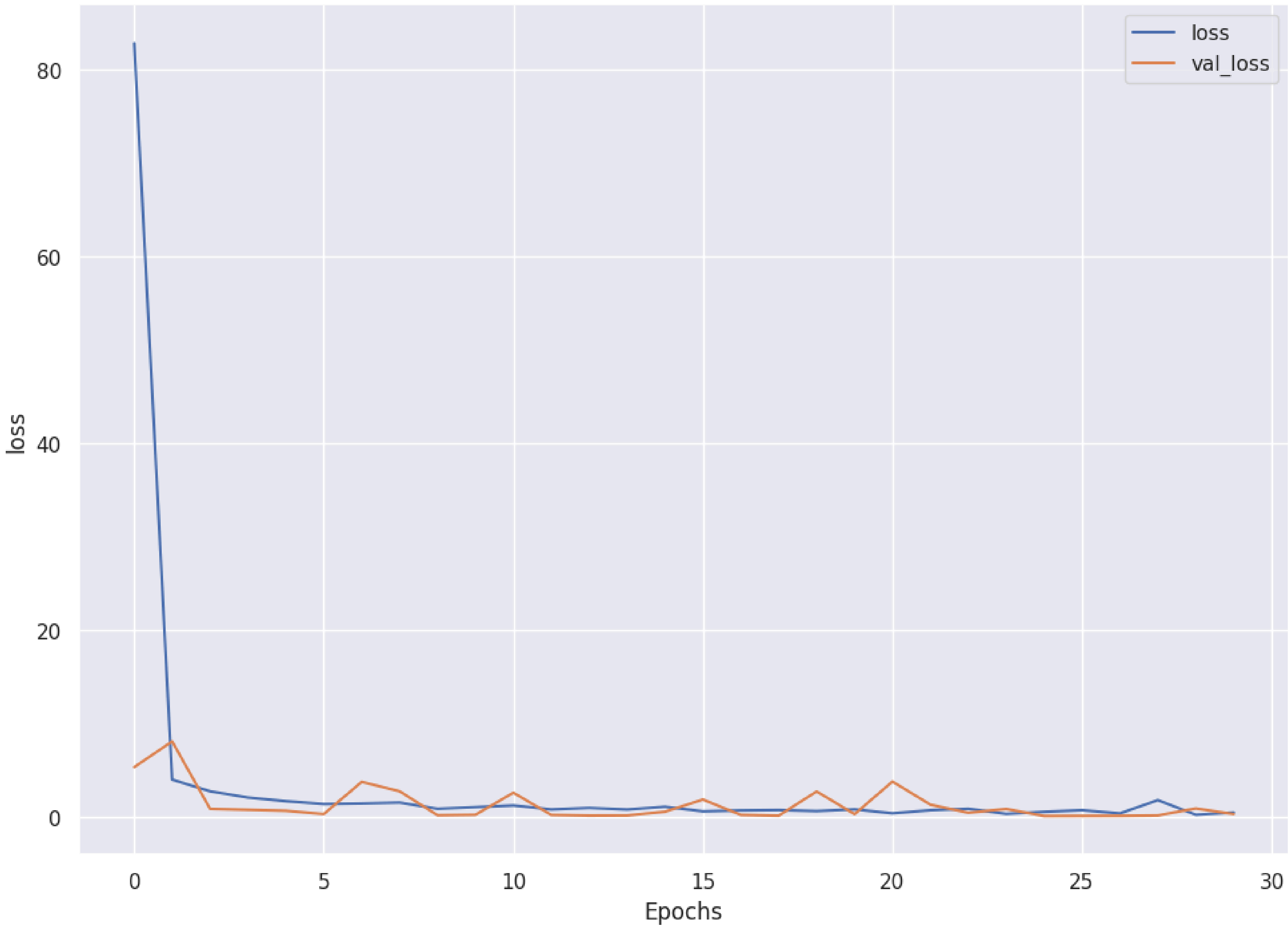
```
e: 0.1719 - val_mae: 0.3338
Epoch 15/30
219/219 [==============================] - 7s 33ms/step - loss: 1.1001 - mse: 1.1001 - mae: 0.6632 - val_loss: 0.5716 - val_ms
e: 0.5716 - val_mae: 0.6495
Epoch 16/30
219/219 [==============================] - 8s 35ms/step - loss: 0.6126 - mse: 0.6126 - mae: 0.6134 - val_loss: 1.8717 - val_ms
e: 1.8717 - val_mae: 1.2382
Epoch 17/30
219/219 [==============================] - 7s 33ms/step - loss: 0.7086 - mse: 0.7086 - mae: 0.6749 - val_loss: 0.2296 - val_ms
e: 0.2296 - val_mae: 0.3862
Epoch 18/30
219/219 [==============================] - 7s 33ms/step - loss: 0.7457 - mse: 0.7457 - mae: 0.6558 - val_loss: 0.1518 - val_ms
e: 0.1518 - val_mae: 0.3166
Epoch 19/30
219/219 [==============================] - 7s 33ms/step - loss: 0.6392 - mse: 0.6392 - mae: 0.6340 - val_loss: 2.7383 - val_ms
e: 2.7383 - val_mae: 1.4869
Epoch 20/30
219/219 [==============================] - 7s 33ms/step - loss: 0.8223 - mse: 0.8223 - mae: 0.6879 - val_loss: 0.3098 - val_ms
e: 0.3098 - val_mae: 0.4662
Epoch 21/30
219/219 [==============================] - 8s 35ms/step - loss: 0.3997 - mse: 0.3997 - mae: 0.5054 - val_loss: 3.8000 - val_ms
e: 3.8000 - val_mae: 1.8465
Epoch 22/30
219/219 [==============================] - 7s 33ms/step - loss: 0.7291 - mse: 0.7291 - mae: 0.6255 - val_loss: 1.3264 - val_ms
e: 1.3264 - val_mae: 1.0338
Epoch 23/30
219/219 [==============================] - 7s 33ms/step - loss: 0.8897 - mse: 0.8897 - mae: 0.7106 - val_loss: 0.4590 - val_ms
e: 0.4590 - val_mae: 0.5723
Epoch 24/30
219/219 [==============================] - 7s 33ms/step - loss: 0.3386 - mse: 0.3386 - mae: 0.4478 - val_loss: 0.8658 - val_ms
e: 0.8658 - val_mae: 0.8289
Epoch 25/30
219/219 [==============================] - 8s 35ms/step - loss: 0.5633 - mse: 0.5633 - mae: 0.5861 - val_loss: 0.1159 - val_ms
e: 0.1159 - val_mae: 0.2820
Epoch 26/30
219/219 [==============================] - 7s 32ms/step - loss: 0.7321 - mse: 0.7321 - mae: 0.5952 - val_loss: 0.1268 - val_ms
e: 0.1268 - val_mae: 0.2927
Epoch 27/30
219/219 [==============================] - 7s 33ms/step - loss: 0.3913 - mse: 0.3913 - mae: 0.4893 - val_loss: 0.1334 - val_ms
e: 0.1334 - val_mae: 0.2986
Epoch 28/30
```

```
219/219 [==============================] - 7s 33ms/step - loss: 1.8086 - mse: 1.8086 - mae: 0.8651 - val_loss: 0.1728 - val_ms
e: 0.1728 - val_mae: 0.3298
Epoch 29/30
219/219 [==============================] - 8s 37ms/step - loss: 0.2463 - mse: 0.2463 - mae: 0.3909 - val_loss: 0.9194 - val_ms
e: 0.9194 - val_mae: 0.8670
Epoch 30/30
219/219 [==============================] - 7s 33ms/step - loss: 0.4820 - mse: 0.4820 - mae: 0.5225 - val_loss: 0.3178 - val_ms
e: 0.3178 - val_mae: 0.4649
```

we plot train and validation loss throughout training

In [47]:

```
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 training loss so our model is not overfitting

In [48]:
```
z= model.predict(X_test)
```

```
1094/1094 [==============================] - 4s 3ms/step
```

In [49]:
```
r2_score(y_test, z)
```

Out[49]:
```
0.9962758051276646
```

In [50]:
```
mse = mean_squared_error(y_test, z)
rmse = mse**.5
print("mse : ",mse)
print("rmse : ",rmse)
print("errors for neural net")
mae = mean_absolute_error(y_test, z)
print("mae : ",mae)
```

```
mse :  0.32043120271985864
rmse :  0.5660664295997941
errors for neural net
mae :  0.46901796443968646
```

In [51]:
```
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, z)
```

Out[51]:
```
0.010093349651551706
```

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

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?

Defining the problem statement and its applications:

- The problem statement involves estimating delivery time for orders placed through Porter's intra-city logistics service. This estimation can enhance customer satisfaction and operational efficiency. Modifications of this approach could be applied to various delivery services, optimizing logistics for e-commerce, food delivery, and more.

Functions provided by pandas datetime:

1. **dt.year**: Extracts the year from datetime objects.
2. **dt.month**: Extracts the month from datetime objects.
3. **dt.day**: Extracts the day from datetime objects.

Brief note on datetime, timedelta, and time span (period):

- **Datetime**: Represents a specific point in time, including both date and time components.
- **Timedelta**: Represents the difference between two datetime objects, providing flexibility for time calculations.
- **Time span (period)**: Represents a specific duration or interval of time, typically used in time series analysis to denote a fixed frequency.

Need to check for outliers in data:

- Outliers can skew statistical analyses and machine learning models, leading to inaccurate results and reduced model performance. Identifying and handling outliers is crucial for ensuring the robustness and reliability of analyses and predictions.

Outlier removal methods:

1. **Standard deviation method**: Removing data points that fall outside a certain number of standard deviations from the mean.
2. **Interquartile range (IQR) method**: Removing data points that fall outside a specified range defined by the first and third quartiles.
3. **Z-score method**: Removing data points with z-scores above or below a certain threshold.

Classical machine learning methods for this problem:

- Classical machine learning methods such as linear regression, decision trees, random forests, and gradient boosting can be used for regression tasks like delivery time estimation. These methods offer interpretable models and can handle a variety of feature types.

Scaling requirement for neural networks:

- Scaling is required for neural networks to ensure that all input features contribute proportionally to the model's training process. Without scaling, features with larger magnitudes can dominate the learning process, leading to slower convergence and suboptimal performance.

Choice of optimizer:

- I chose the Adam optimizer for its adaptive learning rate properties, which can lead to faster convergence and better generalization performance compared to traditional gradient descent algorithms.

Activation function choice:

- I used the ReLU (Rectified Linear Unit) activation function for hidden layers due to its ability to mitigate the vanishing gradient problem and accelerate convergence through efficient gradient propagation.

Neural network performance on a large dataset:

- Neural networks perform well on large datasets due to their capacity to learn complex patterns and relationships from vast amounts of data. With more data, neural networks can better generalize to unseen examples, resulting in improved performance and robustness. Additionally, neural networks can effectively utilize parallel processing capabilities to handle large-scale training tasks efficiently.

In [ ]:

```

```

To conclude after above excersize, here are some findings and recommendations:

Findings:

1. **Data Preprocessing and Feature Engineering**:
   - The dataset contains various features such as market ID, order timestamps, total items, and more, which are crucial for regression analysis.
   - Feature engineering involved creating new features such as hour of day, day of the week, and the target variable - delivery time.
2. **Data Visualization and Cleaning**:
   - Visualization revealed insights into the distribution and relationships between different features.
   - Outliers were detected, potentially skewing the analysis and model performance.
3. **Regression with Neural Networks**:
   - The dataset was split into training and testing sets for model evaluation.
   - Data scaling was performed to ensure all features contribute proportionally to model training.
   - Different configurations and hyperparameters were explored to optimize the neural network architecture.
   - Model training involved defining the neural network architecture, selecting activation functions, optimizers, and training for multiple epochs.

Recommendations:

1. **Data Preprocessing**:
   - Further exploration of data quality issues and missing values should be conducted to ensure the reliability of the analysis.
   - Consider additional feature engineering techniques such as interaction terms or polynomial features to capture more complex relationships in the data.
2. **Outlier Handling**:
   - Outliers should be carefully examined to determine if they represent genuine data points or erroneous entries.
   - Employ robust outlier detection methods and consider domain knowledge to decide whether to remove or adjust outlier values.
3. **Model Evaluation**:
   - Besides traditional regression metrics like MSE, RMSE, and MAE, consider evaluating the model's performance using domain-specific metrics such as delivery time accuracy within a certain time window.

- Explore techniques like cross-validation to assess the model's generalization ability more effectively.

4. **Neural Network Optimization**:

   - Continue experimenting with different neural network architectures, including varying the number of layers, neurons per layer, and regularization techniques like dropout or L2 regularization.
   - Perform hyperparameter tuning using techniques like grid search or random search to find the optimal combination of parameters for model performance.

5. **Deployment and Monitoring**:

   - Once a satisfactory model is developed, deploy it in a production environment for real-time delivery time estimation.
   - Implement monitoring systems to track model performance over time and retrain the model periodically with new data to ensure its accuracy and relevance.

6. **Customer Satisfaction and Operational Efficiency**:

   - Use the estimated delivery time to provide customers with accurate delivery estimates, improving their experience and satisfaction.
   - Optimize logistics operations by efficiently allocating delivery partners based on demand and improving overall service quality.

By implementing these recommendations, Porter can enhance its delivery time estimation capabilities, leading to improved customer satisfaction, operational efficiency, and business performance.

In [ ]:

```
```

In [ ]:

```
```