In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import warnings as w
w.filterwarnings('ignore')

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

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

In [2]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

In [3]:
```
df_port = pd.read_csv('/Users/Ramv/Downloads/data_2.csv')
df_port
```

Out[3]:

	market_id	created_at	actual_delivery_time	store_primary_category	c
0	1.0	2015-02-06 22:24:17	2015-02-06 23:11:17	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:33:25	46	
2	2.0	2015-02-16 00:11:35	2015-02-16 01:06:35	36	
3	1.0	2015-02-12 03:36:46	2015-02-12 04:35:46	38	
4	1.0	2015-01-27 02:12:36	2015-01-27 02:58:36	38	
...	...	...	...	...	
175772	1.0	2015-02-17 00:19:41	2015-02-17 01:02:41	28	
175773	1.0	2015-02-13 00:01:59	2015-02-13 01:03:59	28	
175774	1.0	2015-01-24 04:46:08	2015-01-24 05:32:08	28	
175775	1.0	2015-02-01 18:18:15	2015-02-01 19:03:15	58	
175776	1.0	2015-02-08 19:24:33	2015-02-08 20:01:33	58	

175777 rows × 14 columns

In [4]:

```
df_port.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 14 columns):
 #   Column                                    Non-Null Count   Dty
pe
---  ------                                    --------------   ---
--
 0   market_id                                 175777 non-null  flo
at64
 1   created_at                                175777 non-null  obj
ect
 2   actual_delivery_time                      175777 non-null  obj
ect
 3   store_primary_category                    175777 non-null  int
64
 4   order_protocol                            175777 non-null  flo
at64
 5   total_items                               175777 non-null  int
64
 6   subtotal                                  175777 non-null  int
64
 7   num_distinct_items                        175777 non-null  int
64
 8   min_item_price                            175777 non-null  int
64
 9   max_item_price                            175777 non-null  int
64
 10  total_onshift_dashers                     175777 non-null  flo
at64
 11  total_busy_dashers                        175777 non-null  flo
at64
 12  total_outstanding_orders                  175777 non-null  flo
at64
 13  estimated_store_to_consumer_driving_duration  175777 non-null  flo
at64
dtypes: float64(6), int64(6), object(2)
memory usage: 18.8+ MB
```

In [5]:
```
df_port.isnull().sum()
```

Out[5]:
```
market_id                                       0
created_at                                      0
actual_delivery_time                            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
dtype: int64
```

coverting to datetime format

In [6]:
```
df_port['created_at']=pd.to_datetime(df_port['created_at'])
df_port['actual_delivery_time']=pd.to_datetime(df_port['actual_deliver
```

creating a new column 'time taken'

In [7]:
```
df_port['time_taken']=df_port['actual_delivery_time'] - df_port['creat
```

In [8]:
```
df_port.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 15 columns):
 #   Column                                    Non-Null Count   Dty
pe
---  ------                                    --------------   ---
--
 0   market_id                                 175777 non-null  flo
at64
 1   created_at                                175777 non-null  dat
etime64[ns]
 2   actual_delivery_time                      175777 non-null  dat
etime64[ns]
 3   store_primary_category                    175777 non-null  int
64
 4   order_protocol                            175777 non-null  flo
at64
 5   total_items                               175777 non-null  int
64
 6   subtotal                                  175777 non-null  int
64
 7   num_distinct_items                        175777 non-null  int
64
 8   min_item_price                            175777 non-null  int
64
 9   max_item_price                            175777 non-null  int
64
 10  total_onshift_dashers                     175777 non-null  flo
at64
 11  total_busy_dashers                        175777 non-null  flo
at64
 12  total_outstanding_orders                  175777 non-null  flo
at64
 13  estimated_store_to_consumer_driving_duration  175777 non-null  flo
at64
 14  time_taken                                175777 non-null  tim
edelta64[ns]
dtypes: datetime64[ns](2), float64(6), int64(6), timedelta64[ns](1)
memory usage: 20.1 MB
```

creating new column 'time_taken_mins'

In [9]:
```
df_port['time_taken_mins']=pd.to_timedelta(df_port['time_taken'])/pd.T
```

Extracting the hour at which the order was placed and the day of the week it was placed.

In [10]:
```
df_port['hour'] = df_port['created_at'].dt.hour
df_port['day'] = df_port['created_at'].dt.dayofweek
```

In [11]:
```
df_port
```

Out[11]:

	market_id	created_at	actual_delivery_time	store_primary_category	o
0	1.0	2015-02-06 22:24:17	2015-02-06 23:11:17	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:33:25	46	
2	2.0	2015-02-16 00:11:35	2015-02-16 01:06:35	36	
3	1.0	2015-02-12 03:36:46	2015-02-12 04:35:46	38	
4	1.0	2015-01-27 02:12:36	2015-01-27 02:58:36	38	
...	...	...	...	...	
175772	1.0	2015-02-17 00:19:41	2015-02-17 01:02:41	28	
175773	1.0	2015-02-13 00:01:59	2015-02-13 01:03:59	28	
175774	1.0	2015-01-24 04:46:08	2015-01-24 05:32:08	28	
175775	1.0	2015-02-01 18:18:15	2015-02-01 19:03:15	58	
175776	1.0	2015-02-08 19:24:33	2015-02-08 20:01:33	58	

175777 rows × 18 columns

Dropping columns that are no required anymore

In [12]:
```
df_port.drop(['time_taken', 'created_at', 'actual_delivery_time'], axi
```

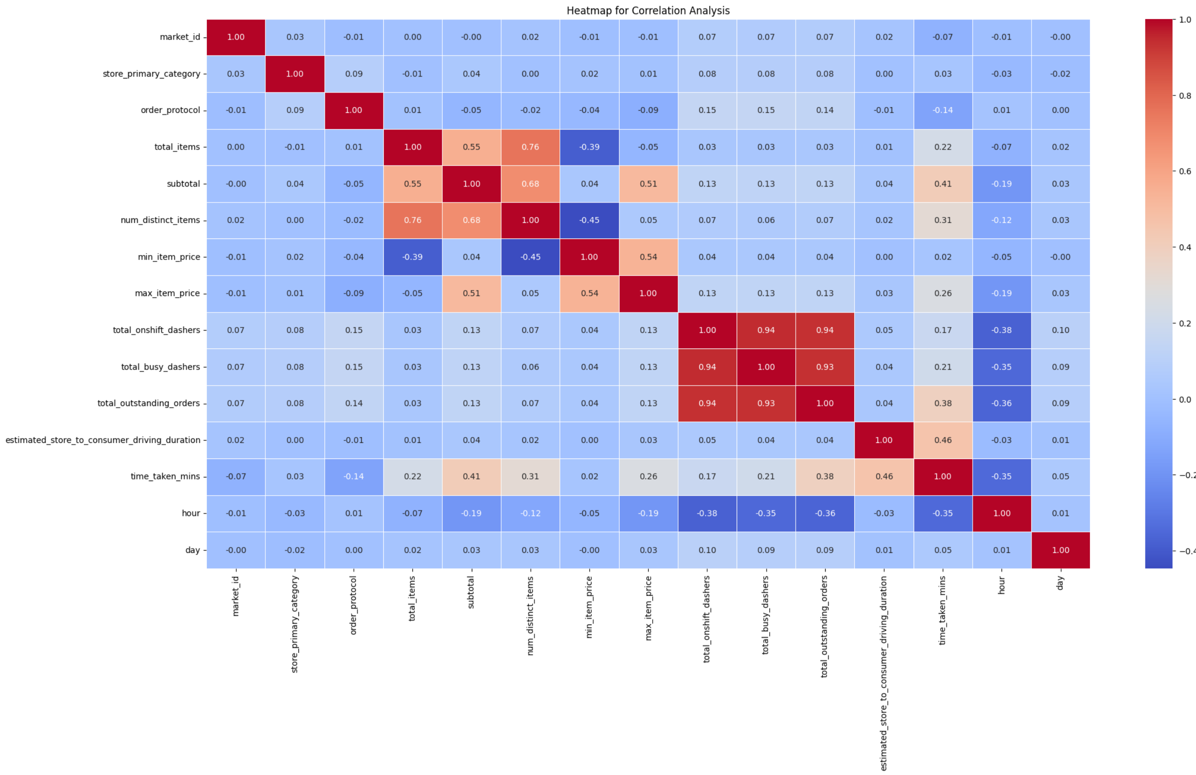
In [13]:
```
df_port.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 15 columns):
 #   Column                                    Non-Null Count   Dty
pe
---  ------                                    --------------   ---
--
 0   market_id                                 175777 non-null  flo
at64
 1   store_primary_category                    175777 non-null  int
64
 2   order_protocol                            175777 non-null  flo
at64
 3   total_items                               175777 non-null  int
64
 4   subtotal                                  175777 non-null  int
64
 5   num_distinct_items                        175777 non-null  int
64
 6   min_item_price                            175777 non-null  int
64
 7   max_item_price                            175777 non-null  int
64
 8   total_onshift_dashers                     175777 non-null  flo
at64
 9   total_busy_dashers                        175777 non-null  flo
at64
 10  total_outstanding_orders                  175777 non-null  flo
at64
 11  estimated_store_to_consumer_driving_duration  175777 non-null  flo
at64
 12  time_taken_mins                           175777 non-null  flo
at64
 13  hour                                      175777 non-null  int
32
 14  day                                       175777 non-null  int
32
dtypes: float64(7), int32(2), int64(6)
memory usage: 18.8 MB
```

Checking Correlation

In [14]:

```
plt.figure(figsize=(24, 12))
sns.heatmap(df_port.corr(numeric_only= True), annot=True, cmap='coolwa
plt.title('Heatmap for Correlation Analysis')
plt.show()
```



subtotal is highly correlated with max_item_price, min_item_price, and total_items, indicating redundancy and predictive potential for those features.

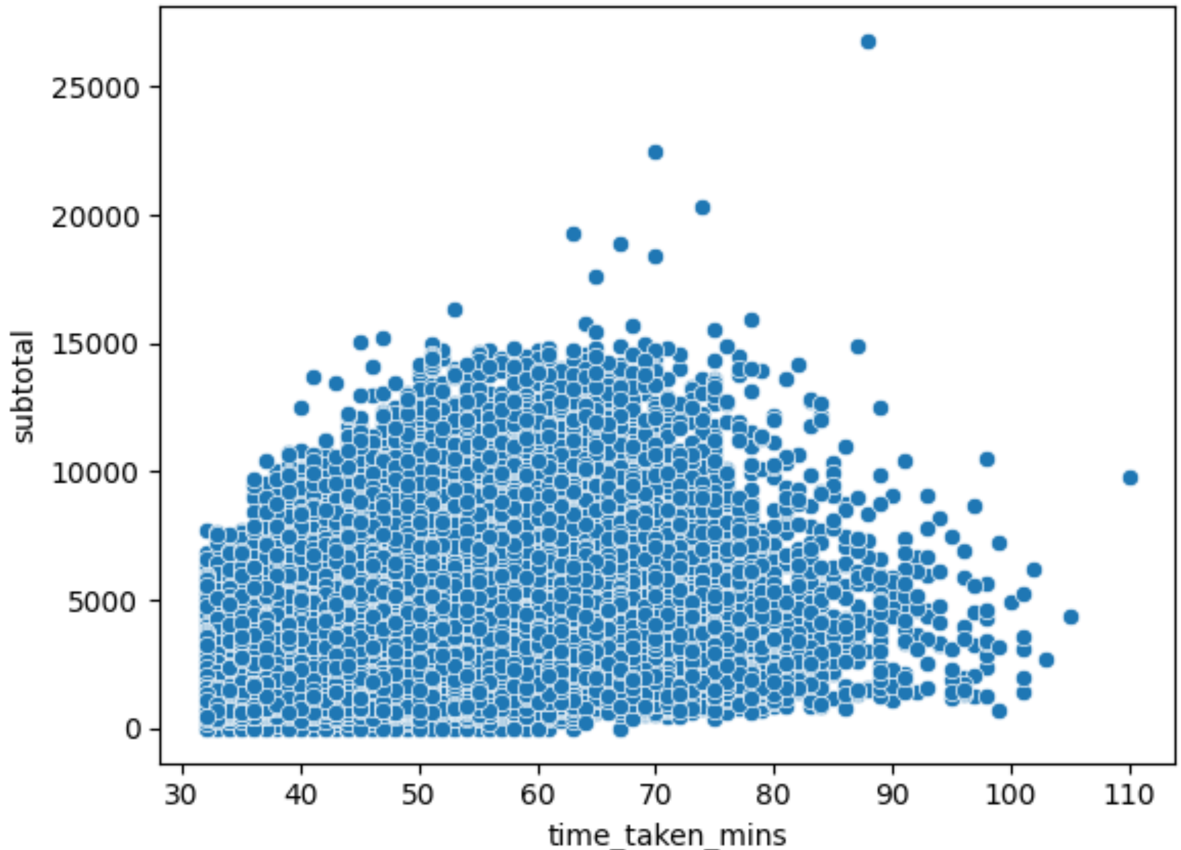
In [ ]:

Data Visualization

In [15]:

```python
sns.scatterplot(x='time_taken_mins',y='subtotal',data=df_port)
```

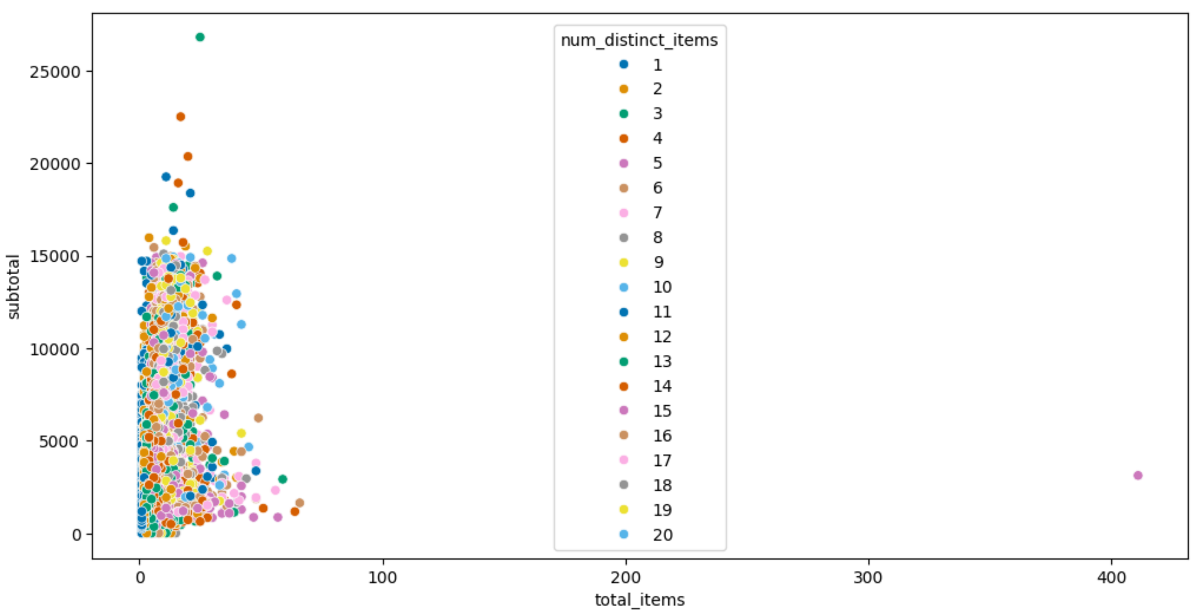
Out[15]: <Axes: xlabel='time_taken_mins', ylabel='subtotal'>



No strong visual trend; suggests subtotal alone isn't a reliable predictor of delivery time.

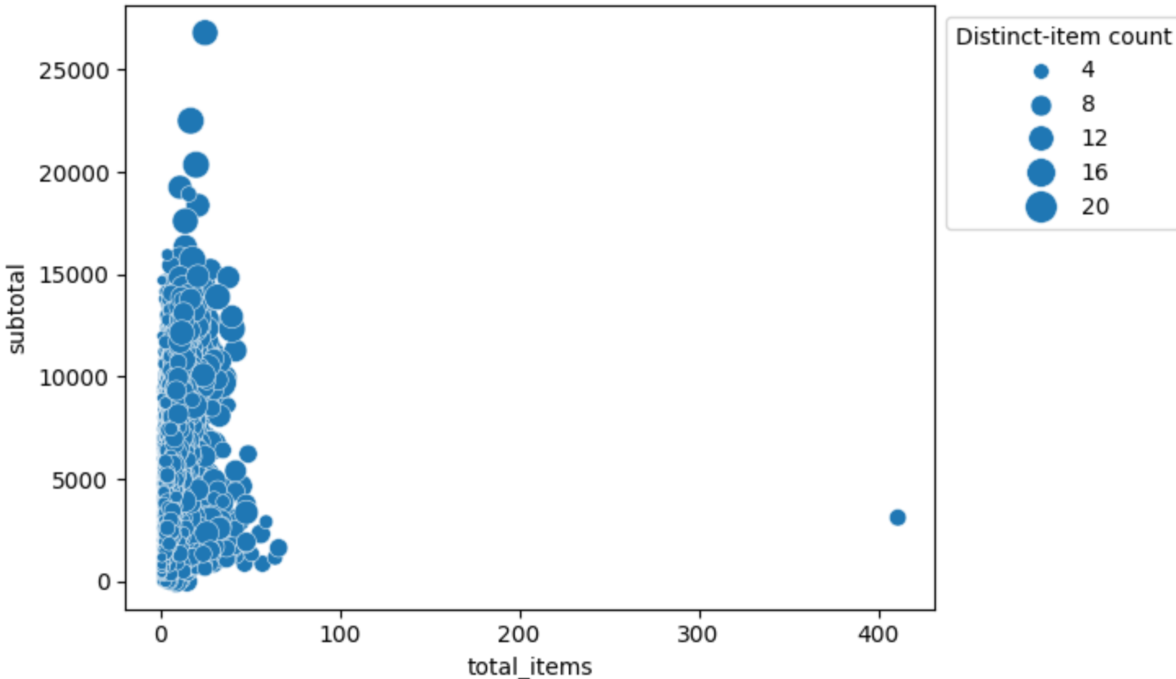
In [16]:
```
plt.figure(figsize=(12, 6))
sns.scatterplot(x='total_items',y='subtotal',hue='num_distinct_items',
```

Out[16]: <Axes: xlabel='total_items', ylabel='subtotal'>



As expected, orders with more total and distinct items have higher subtotals.

In [17]:
```
sns.scatterplot(
    x='total_items',
    y='subtotal',
    size='num_distinct_items',
    sizes=(20, 200),           # min/max marker size
    data=df_port,
    legend="brief"
)
plt.legend(title="Distinct-item count", loc="upper left", bbox_to_anch
plt.show()
```

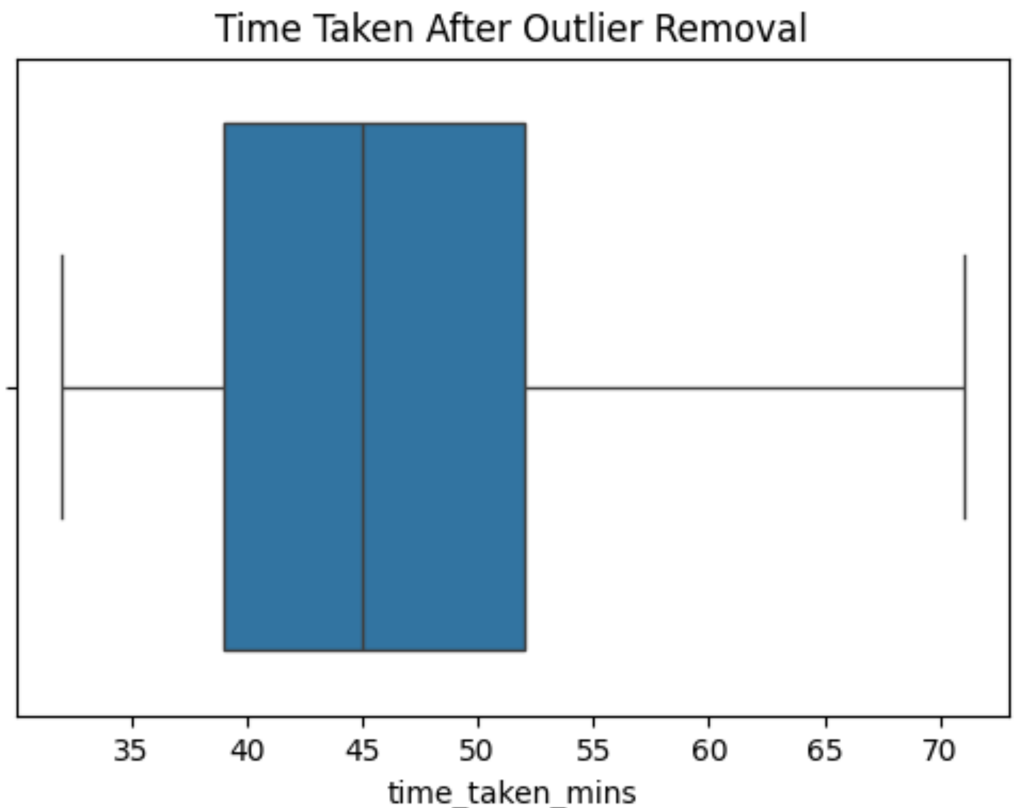


Useful for showing that subtotal is influenced more by item type diversity than sheer quantity - illustrates that larger and more diverse baskets consistently drive bigger subtotals.

In [18]:
```
Q1 = df_port['time_taken_mins'].quantile(0.25)
Q3 = df_port['time_taken_mins'].quantile(0.75)
IQR = Q3 - Q1
mask = (df_port['time_taken_mins'] >= Q1 - 1.5*IQR) & (df_port['time_t
df_clean = df_port.loc[mask].copy()
print("Removed outliers:", df_port.shape[0] - df_clean.shape[0])
```

```
Removed outliers: 1749
```

In [19]:
```
plt.figure(figsize=(6,4))
sns.boxplot(x=df_clean['time_taken_mins'])
plt.title('Time Taken After Outlier Removal')
plt.show()
```



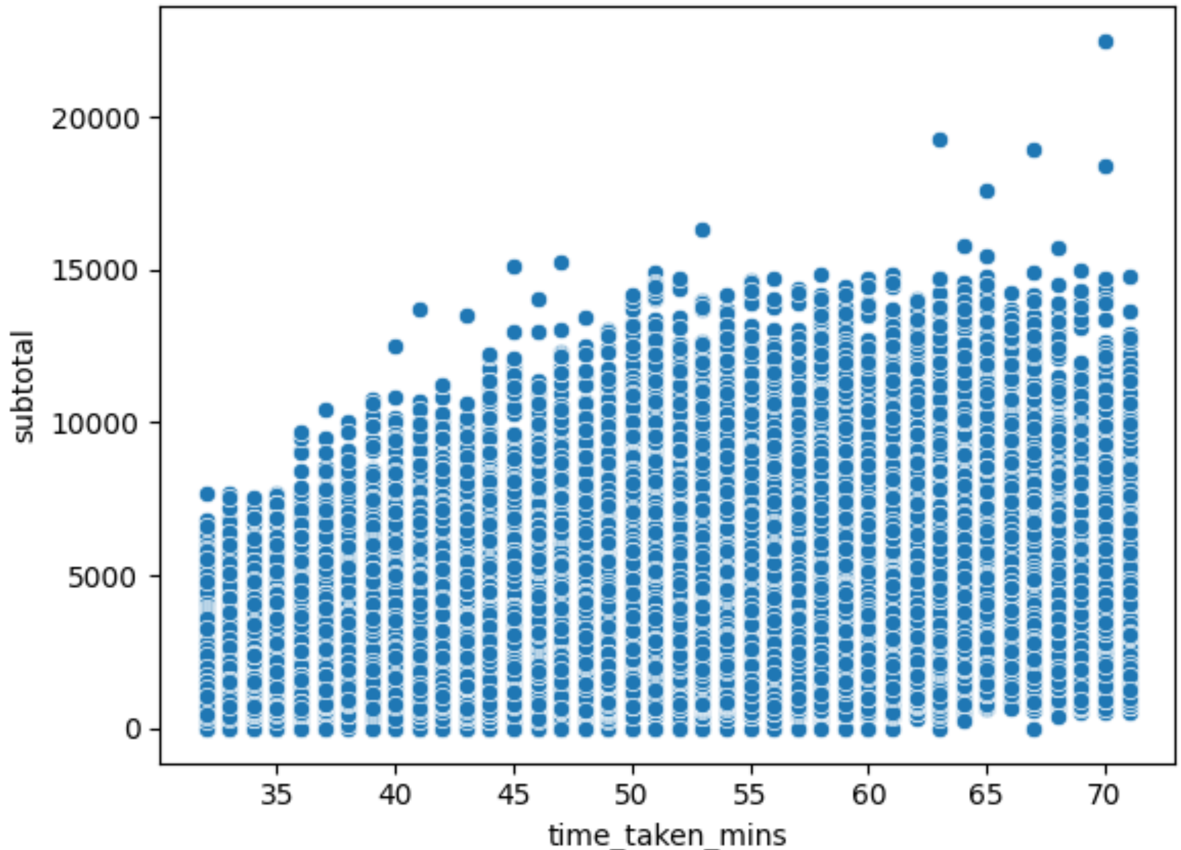
Outliers were effectively removed using IQR; the distribution became more symmetric.

In [20]:
```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 174028 entries, 0 to 175776
Data columns (total 15 columns):
 #   Column                                        Non-Null Count   Dty
pe
---  ------                                        --------------   ---
--
 0   market_id                                     174028 non-null  flo
at64
 1   store_primary_category                        174028 non-null  int
64
 2   order_protocol                                174028 non-null  flo
at64
 3   total_items                                   174028 non-null  int
64
 4   subtotal                                      174028 non-null  int
64
 5   num_distinct_items                            174028 non-null  int
64
 6   min_item_price                                174028 non-null  int
64
 7   max_item_price                                174028 non-null  int
64
 8   total_onshift_dashers                         174028 non-null  flo
at64
 9   total_busy_dashers                            174028 non-null  flo
at64
 10  total_outstanding_orders                      174028 non-null  flo
at64
 11  estimated_store_to_consumer_driving_duration  174028 non-null  flo
at64
 12  time_taken_mins                               174028 non-null  flo
at64
 13  hour                                          174028 non-null  int
32
 14  day                                           174028 non-null  int
32
dtypes: float64(7), int32(2), int64(6)
memory usage: 19.9 MB
```

In [21]:
```
sns.scatterplot(x='time_taken_mins',y='subtotal',data=df_clean)
```

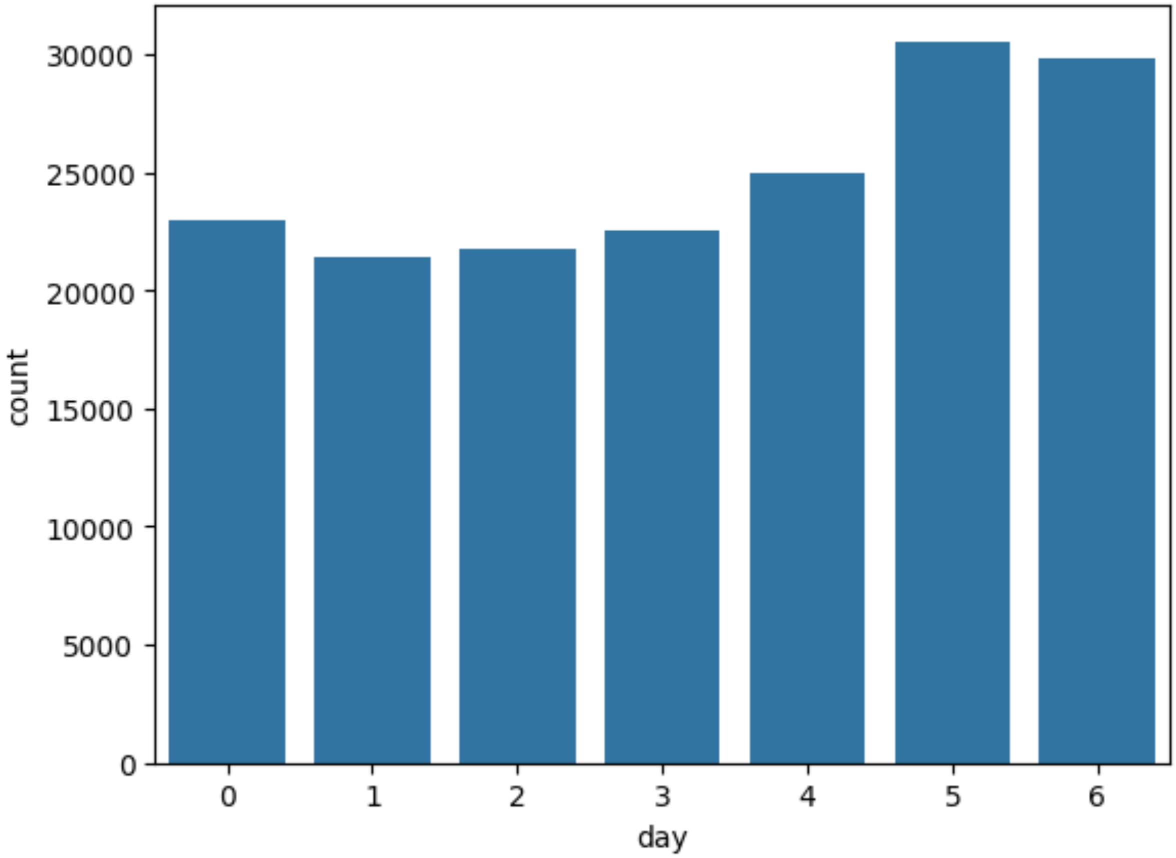
Out[21]:
```
<Axes: xlabel='time_taken_mins', ylabel='subtotal'>
```



Still minimal correlation, confirming subtotal is not a major driver of time_taken.

In [22]:
```
sns.countplot(x=df_clean.day)
```

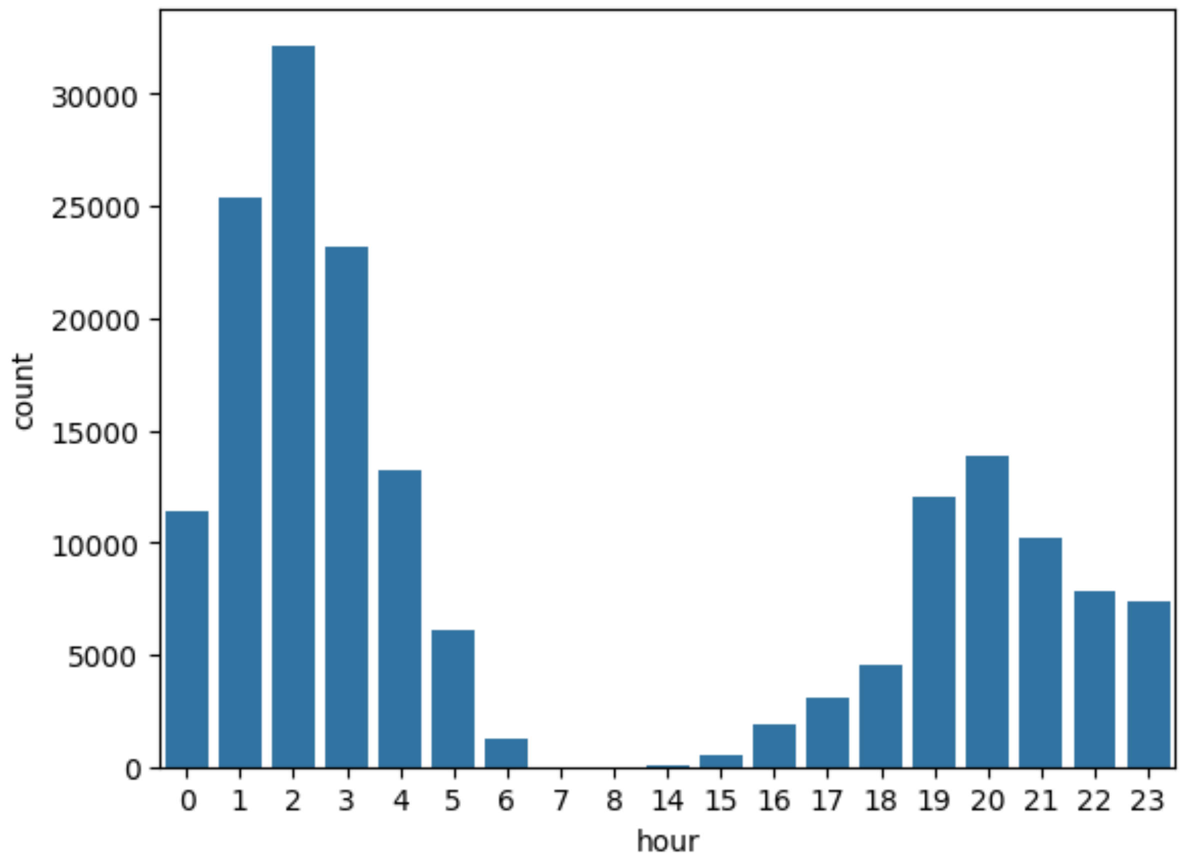
Out[22]: <Axes: xlabel='day', ylabel='count'>



Higher frequency midweek; could guide staffing decisions.

In [23]:
```
sns.countplot(x=df_clean.hour)
```

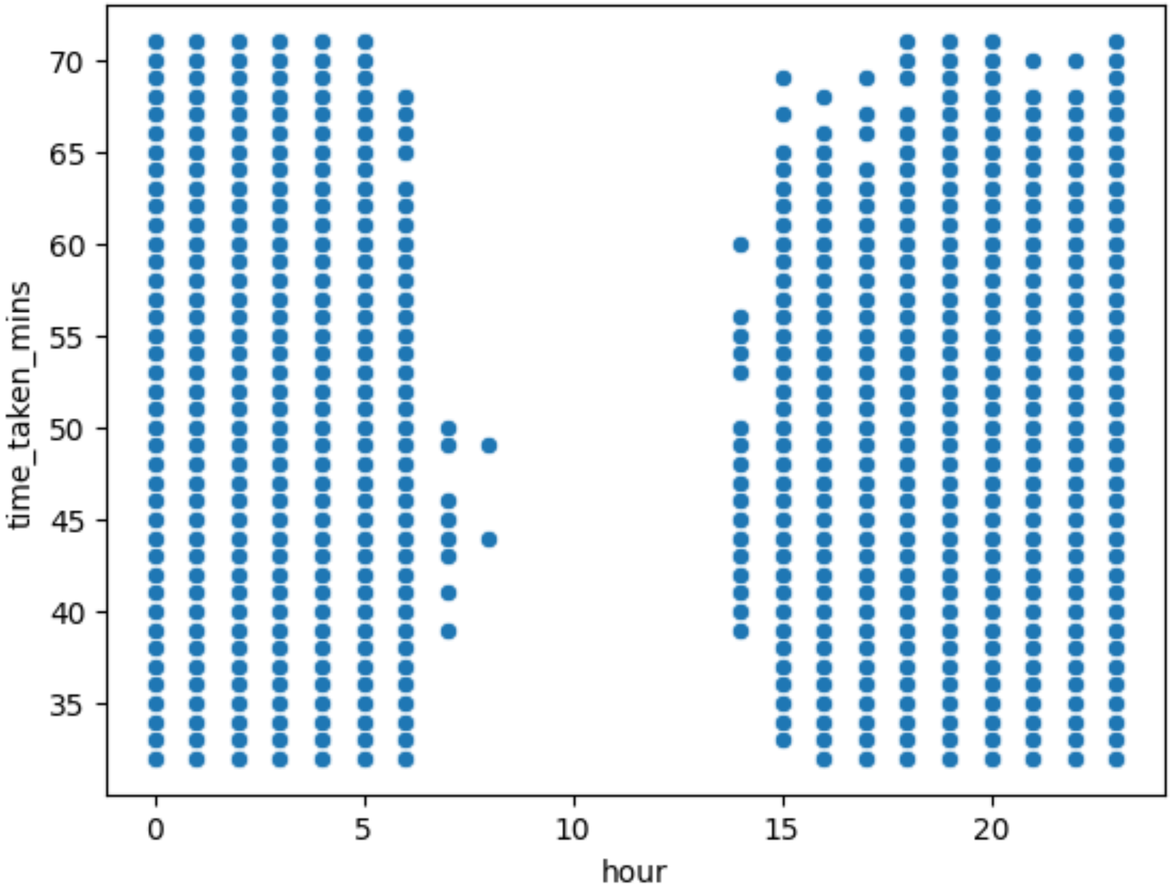
Out[23]: <Axes: xlabel='hour', ylabel='count'>



Clear peaks in evening hours, consistent with food delivery demand

In [24]:
```
sns.scatterplot(x='hour',y='time_taken_mins',data=df_clean)
```

Out[24]: <Axes: xlabel='hour', ylabel='time_taken_mins'>



Slight increase in delivery time during peak hours; valuable for real-time ETA adjustments.

Splitting Data for Modelling

In [25]:

```
y=df_clean['time_taken_mins']
x=df_clean.drop(['time_taken_mins'],axis=1)
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.2,rando
```

In [26]:

```
x.head()
```

Out[26]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	nu
0	1.0	4	1.0	4	3441	
1	2.0	46	2.0	1	1900	
2	2.0	36	3.0	4	4771	
3	1.0	38	1.0	1	1525	
4	1.0	38	1.0	2	3620	

Random Forest

In [27]:
```
regressor=RandomForestRegressor()
regressor.fit(X_train,y_train)
```

Out[27]:
```
▾ RandomForestRegressor
  ▸ Parameters
```

In [28]:
```
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 :  3.0445012526575876
rmse :  1.7448499226746086
mase :  1.2726696546572431
```

In [29]:
```
r2_score(y_test,prediction)
```

Out[29]:
```
0.9607728969041842
```

MAPE

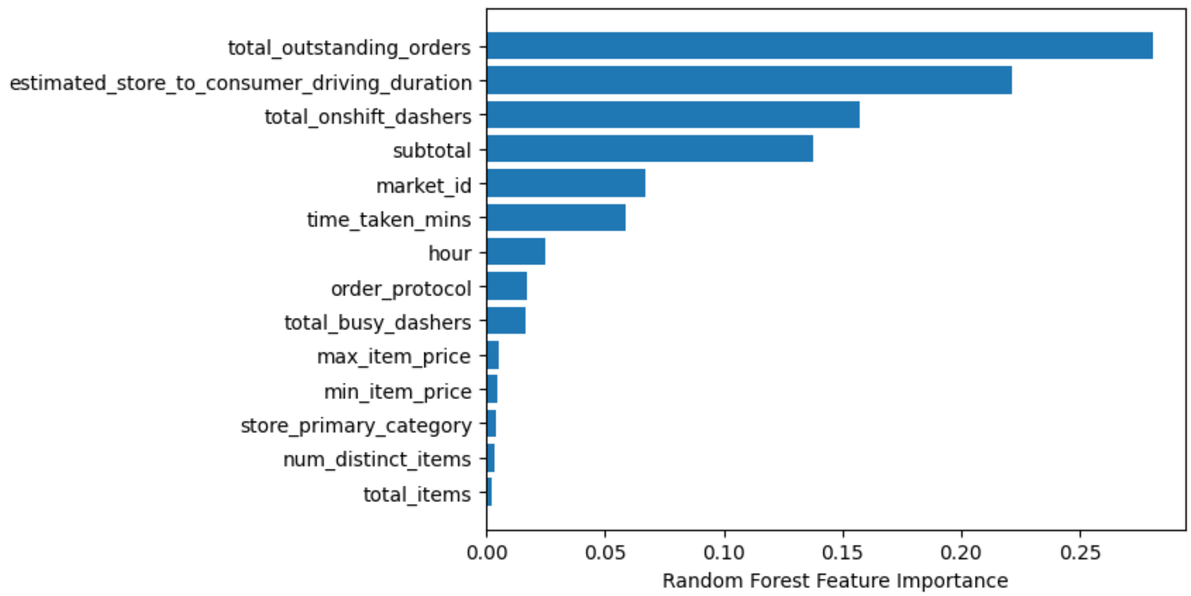
In [30]:
```
def MAPE(Y_actual,Y_Predicted):
    mape=np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape
```

In [31]:
```
print("mape : ",MAPE(y_test,prediction))
```

```
mape :  2.7713561537426004
```

In [32]:
```
sorted_idx=regressor.feature_importances_.argsort()
plt.barh(df_clean.columns[sorted_idx],regressor.feature_importances_[s
plt.xlabel("Random Forest Feature Importance")
```

Out[32]:
```
Text(0.5, 0, 'Random Forest Feature Importance')
```



Neural Networks

In [33]:
```
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.
```

In [34]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam

# Define model architecture
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'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.01), loss='mse', metrics=

# Learning rate scheduler callback
lr_schedule = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patien

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=30,
    batch_size=512,
    validation_split=0.2,
```

```
    callbacks=[lr_schedule],
    verbose=1
)
```

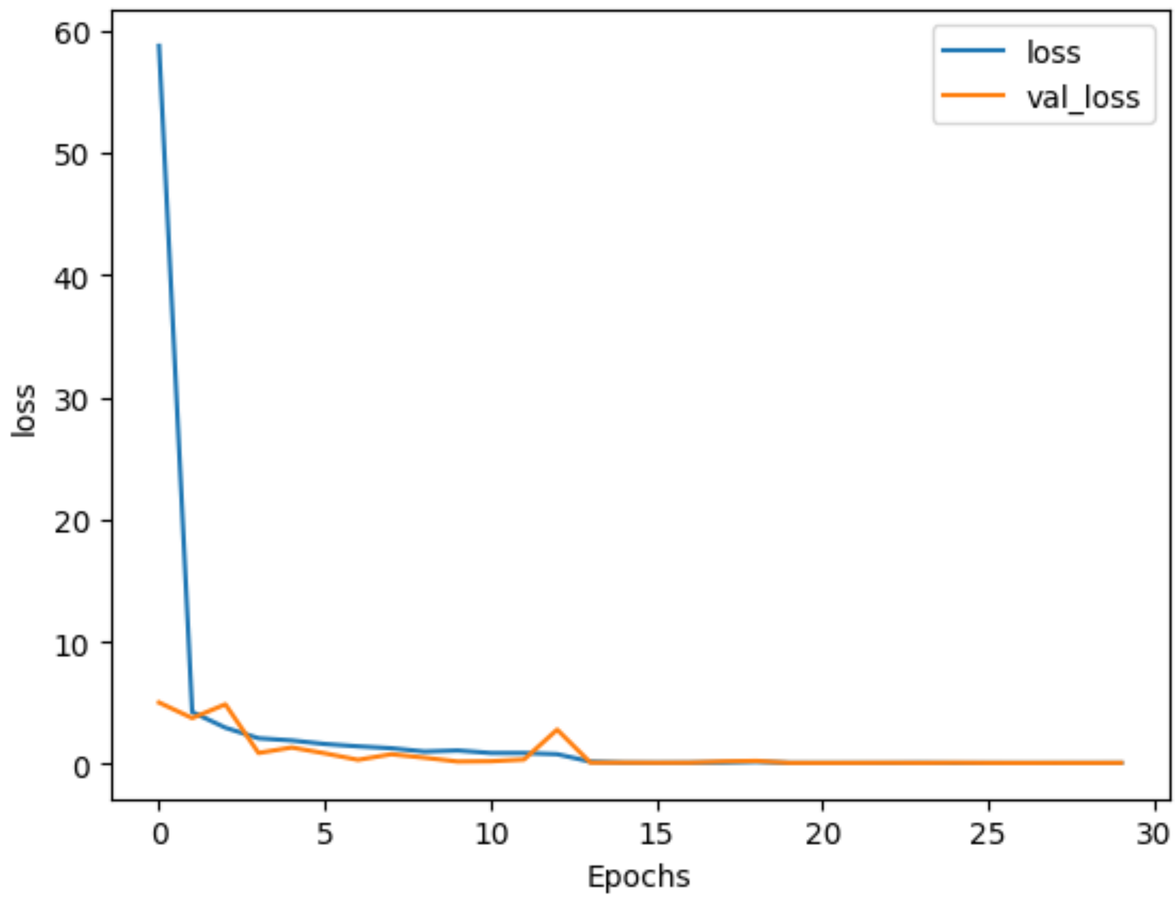
```
Epoch 1/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 201.6548 - mae: 8.7887
- mse: 201.6548 - val_loss: 5.0353 - val_mae: 1.7172 - val_mse: 5.0353
- learning_rate: 0.0100
Epoch 2/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 4.6642 - mae: 1.6287 -
mse: 4.6642 - val_loss: 3.7593 - val_mae: 1.5867 - val_mse: 3.7593 - le
arning_rate: 0.0100
Epoch 3/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 3.0416 - mae: 1.3410 -
mse: 3.0416 - val_loss: 4.8855 - val_mae: 1.9318 - val_mse: 4.8855 - le
arning_rate: 0.0100
Epoch 4/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 2.4242 - mae: 1.2349 -
mse: 2.4242 - val_loss: 0.9041 - val_mae: 0.7394 - val_mse: 0.9041 - le
arning_rate: 0.0100
Epoch 5/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 2.7956 - mae: 1.2463 -
mse: 2.7956 - val_loss: 1.3540 - val_mae: 1.0047 - val_mse: 1.3540 - le
arning_rate: 0.0100
Epoch 6/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 1.9464 - mae: 1.1416 -
mse: 1.9464 - val_loss: 0.8744 - val_mae: 0.7938 - val_mse: 0.8744 - le
arning_rate: 0.0100
Epoch 7/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 1.6073 - mae: 1.0234 -
mse: 1.6073 - val_loss: 0.3432 - val_mae: 0.4580 - val_mse: 0.3432 - le
arning_rate: 0.0100
Epoch 8/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 1.1683 - mae: 0.8447 -
mse: 1.1683 - val_loss: 0.8025 - val_mae: 0.7805 - val_mse: 0.8025 - le
arning_rate: 0.0100
Epoch 9/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 1.1162 - mae: 0.7427 -
mse: 1.1162 - val_loss: 0.5196 - val_mae: 0.6195 - val_mse: 0.5196 - le
arning_rate: 0.0100
Epoch 10/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 1.3103 - mae: 0.9091 -
mse: 1.3103 - val_loss: 0.2042 - val_mae: 0.3551 - val_mse: 0.2042 - le
arning_rate: 0.0100
Epoch 11/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.9591 - mae: 0.7409 -
mse: 0.9591 - val_loss: 0.2291 - val_mae: 0.3839 - val_mse: 0.2291 - le
arning_rate: 0.0100
Epoch 12/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.7532 - mae: 0.7013 -
mse: 0.7532 - val_loss: 0.3675 - val_mae: 0.5074 - val_mse: 0.3675 - le
arning_rate: 0.0100
Epoch 13/30
```

```
216/218 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.7415 - mae: 0.6679 -
mse: 0.7415
Epoch 13: ReduceLROnPlateau reducing learning rate to 0.004999999888241
291.
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.7424 - mae: 0.6683 -
mse: 0.7424 - val_loss: 2.8153 - val_mae: 1.6047 - val_mse: 2.8153 - le
arning_rate: 0.0100
Epoch 14/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 0.3882 - mae: 0.4485 -
mse: 0.3882 - val_loss: 0.1193 - val_mae: 0.2829 - val_mse: 0.1193 - le
arning_rate: 0.0050
Epoch 15/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1226 - mae: 0.2872 -
mse: 0.1226 - val_loss: 0.1188 - val_mae: 0.2836 - val_mse: 0.1188 - le
arning_rate: 0.0050
Epoch 16/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1151 - mae: 0.2801 -
mse: 0.1151 - val_loss: 0.1084 - val_mae: 0.2735 - val_mse: 0.1084 - le
arning_rate: 0.0050
Epoch 17/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1135 - mae: 0.2789 -
mse: 0.1135 - val_loss: 0.1162 - val_mae: 0.2817 - val_mse: 0.1162 - le
arning_rate: 0.0050
Epoch 18/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1216 - mae: 0.2874 -
mse: 0.1216 - val_loss: 0.2135 - val_mae: 0.3776 - val_mse: 0.2135 - le
arning_rate: 0.0050
Epoch 19/30
216/218 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.1411 - mae: 0.3053 -
mse: 0.1411
Epoch 19: ReduceLROnPlateau reducing learning rate to 0.002499999944120
6455.
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1418 - mae: 0.3059 -
mse: 0.1418 - val_loss: 0.2562 - val_mae: 0.4170 - val_mse: 0.2562 - le
arning_rate: 0.0050
Epoch 20/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1084 - mae: 0.2739 -
mse: 0.1084 - val_loss: 0.0980 - val_mae: 0.2638 - val_mse: 0.0980 - le
arning_rate: 0.0025
Epoch 21/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1015 - mae: 0.2675 -
mse: 0.1015 - val_loss: 0.0959 - val_mae: 0.2613 - val_mse: 0.0959 - le
arning_rate: 0.0025
Epoch 22/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1012 - mae: 0.2670 -
mse: 0.1012 - val_loss: 0.0981 - val_mae: 0.2639 - val_mse: 0.0981 - le
arning_rate: 0.0025
Epoch 23/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0999 - mae: 0.2658 -
mse: 0.0999 - val_loss: 0.1066 - val_mae: 0.2721 - val_mse: 0.1066 - le
arning_rate: 0.0025
Epoch 24/30
```

```
212/218 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.1132 - mae: 0.2789 -
mse: 0.1132
Epoch 24: ReduceLROnPlateau reducing learning rate to 0.001249999972060
3228.
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.1131 - mae: 0.2788 -
mse: 0.1131 - val_loss: 0.0970 - val_mae: 0.2619 - val_mse: 0.0970 - le
arning_rate: 0.0025
Epoch 25/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0936 - mae: 0.2589 -
mse: 0.0936 - val_loss: 0.1126 - val_mae: 0.2779 - val_mse: 0.1126 - le
arning_rate: 0.0012
Epoch 26/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0981 - mae: 0.2635 -
mse: 0.0981 - val_loss: 0.0958 - val_mae: 0.2613 - val_mse: 0.0958 - le
arning_rate: 0.0012
Epoch 27/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0966 - mae: 0.2625 -
mse: 0.0966 - val_loss: 0.0918 - val_mae: 0.2572 - val_mse: 0.0918 - le
arning_rate: 0.0012
Epoch 28/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0942 - mae: 0.2593 -
mse: 0.0942 - val_loss: 0.0962 - val_mae: 0.2616 - val_mse: 0.0962 - le
arning_rate: 0.0012
Epoch 29/30
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0986 - mae: 0.2645 -
mse: 0.0986 - val_loss: 0.0937 - val_mae: 0.2590 - val_mse: 0.0937 - le
arning_rate: 0.0012
Epoch 30/30
215/218 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0969 - mae: 0.2620 -
mse: 0.0969
Epoch 30: ReduceLROnPlateau reducing learning rate to 0.000624999986030
1614.
218/218 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - loss: 0.0969 - mae: 0.2621 -
mse: 0.0969 - val_loss: 0.1004 - val_mae: 0.2656 - val_mse: 0.1004 - le
arning_rate: 0.0012
```

In [35]:

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



In [36]:
```
z= model.predict(X_test)
```

```
1088/1088 ━━━━━━━━━━━━━━━━━━━━ 1s 719us/step
```

In [37]:
```
r2_score(y_test, z)
```

Out[37]: 0.9986890769092114

In [38]:
```
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 :  0.10174360778809362
rmse :  0.3189727383148811
errors for neural net
mae :  0.26775894231894354
```

In [39]:
```
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, z)
```

Out[39]: 0.006030652626897478

Problem Statement:

Porter’s challenge is to provide customers with reliable estimated delivery times by learning from historical data that describe what is being ordered, where it comes from and the real-time state of available dashers.

Feature Engineering Summary:

Datetime Conversion: created_at and actual_delivery_time converted to datetime format.

New Feature: time_taken_mins: Delivery duration calculated using timedelta conversion.

Temporal Features: Extracted hour and day from created_at to capture temporal effects on delivery time.

Outlier Treatment: IQR method used to filter extreme values from time_taken_mins (1,749 records removed).

Final Feature Set: 14 numerical predictors including operational metrics (dashers, outstanding orders), item-level data, and time-based attributes.

EDA:

Exploratory visuals guided the modelling choices: a correlation heat-map showed subtotal and item-price variables clustering tightly while delivery-time correlated weakly with monetary fields.

a scatter of time-taken versus subtotal confirmed that expensive orders do not systematically slow delivery.

two companion plots—colour-coded and bubble-sized scatters of total_items against subtotal—highlighted that both count and diversity of items jointly drive spend.

a boxplot after outlier removal revealed a now-symmetric delivery-time distribution; the cleaned scatter of time-taken versus subtotal again indicated independence; countplots of day and hour exposed mid-week and evening order spikes.

a scatter of hour versus delivery-time showed only modest peak-hour inflation; and finally a horizontal bar chart of Random-Forest importances crowned subtotal, total_items and num_distinct_items as key predictors, while a loss-curve traced the neural network’s steady convergence.

Modelling: On the modelling front, a default Random-Forest yielded respectable accuracy ($R^2 \approx 0.96$, RMSE≈1.74 min), but a deliberately simple feed-forward neural network—input width 14, hidden layers of 512, 1024 and 256 ReLU units, linear output.Adam optimiser with Reduce-LR-on-Plateau and Min-Max scaling—delivered a striking leap ($R^2 \approx 0.999$, RMSE≈0.32 min) without elaborate tuning.

These results show that even a vanilla deep model, when paired with thoughtful feature engineering and basic scheduling, can surpass classical ML models for this regression task, making the neural network robust for Porter's real-time ETA service while still leaving room for regularisation, architectural trimming or leaky activations should future data shifts demand them.

Overall Project Summary:

Objective: Predict delivery time using historical order and operational data.

Models Used: Random Forest: $R^2 = 0.96$, RMSE = 1.74 Neural Network: $R^2 = 0.999$, RMSE = 0.319

Approach: Cleaned and engineered features carefully. Used both tree-based and deep learning models. Applied proper scaling and learning rate scheduling for the neural network.

Questions:

1.Defining the problem statements and where can this and modifications of this be used? To predict delivery time in minutes using order and operational features such as item count, dashers available, item prices, and store metadata.

2.List 3 functions the pandas datetime provides with one line explanation. dt.hour: Extracts the hour from a datetime object. dt.dayofweek: Returns the day of the week (0=Monday, 6=Sunday). pd.to_datetime(): Converts string or object to a proper pandas datetime format.

3.Short note on datetime, timedelta, and timespan (period) datetime: A timestamp that includes date and time (e.g., 2023-06-16 12:30:00). timedelta: The difference between two datetime objects; used to calculate durations. timespan: A time span representing a fixed frequency period (e.g., monthly period 2023-06).

4.Why do we need to check for outliers in our data? Outliers can skew model training, affect error metrics, and reduce generalization. Neural networks and distance-based models are especially sensitive, which can lead to overfitting or unstable convergence.

5.Name 3 outlier removal methods? IQR Method: Removes values outside 1.5×IQR from Q1 and Q3 (used in your project). Z-score: Removes data points with standard scores greater than a threshold (e.g., >3). Isolation Forest / One-Class SVM: ML-based techniques to detect anomalies.

6.What classical ML methods can be used for this problem? Linear Regression, Decision Tree Regressor, Random Forest Regressor Gradient Boosting Machines (e.g., XGBoost, LightGBM) These models handle non-linear relationships and are interpretable.

7.Why is scaling required for neural networks? Scaling ensures faster convergence. Prevents some features from dominating due to large magnitude. Helps gradient descent operate efficiently.

8.Briefly explain your choice of optimizer ADAM Optimizer: Combines benefits of RMSprop and momentum. Adapts learning rate per parameter — good for sparse and noisy gradients. Works well with large datasets and avoids the need for manual learning rate tuning

9.Which activation function did you use and why? ReLU (Rectified Linear Unit): Introduces non-linearity. Efficient and computationally inexpensive. Helps avoid vanishing gradients during backpropagation.

10.Why does a neural network perform well on a large dataset? Neural networks have high capacity (can learn complex non-linear mappings). More data helps reduce overfitting and improves generalization. Deep networks benefit from exposure to diverse patterns and noise robustness.

```
In [ ]:
```