Porter- Neural Networks Regression

- **Porter, India's largest marketplace for intra-city logistics, is revolutionizing the delivery sector with technology-driven solutions.**
- **Task: Given new order from customer, Help Porter to estimate delivery time, for that we build NN regression model.**

- Each row in this file corresponds to one unique delivery. Each column corresponds to a feature as explained below.

Sr. No.	Feature Name	Description
1	market_id	An integer ID indicating the market area of the restaurant.
2	created_at	Timestamp of when the order was placed.
3	actual_delivery_time	Timestamp of when the order was delivered.
4	store_id	Store ID
5	store_primary_category	Category classification of the restaurant.
6	order_protocol	Numeric code representing the mode of order placement. (how the order was placed ie: through porter, call to restaurant, pre booked, third part etc)
7	total_items	Total items in order
8	subtotal	A combined feature detailing the total number of items and final price of the order before taxes and fees.
9	num_distinct_items	Count of different items in the order.
10	min_item_price	Price of the least expensive item in the order.
11	max_item_price	Price of the most expensive item in the order.
12	total_onshift_partners	Number of delivery partners on duty when the order was placed.
13	total_busy_partners	Number of delivery partners busy with other tasks at the order placement time.
14	total_outstanding_orders	Total count of orders pending at the time.

In [1]:
```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from category_encoders import TargetEncoder
from sklearn.metrics import mean_squared_error, r2_score

import tensorflow as tf
import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
import keras_tuner
```

In [2]:
```
# data loading as df
df= pd.read_csv("Porter.csv")
```

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

Out[3]:

	market_id	created_at	actual_delivery_time	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy_pai
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	df263d996281d984952c07998dc54358	american	1.0	4	3441	4	557	1239	33.0	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	1900	1	1400	1400	1.0	
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	1900	1	1900	1900	1.0	
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	6900	5	600	1800	1.0	
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	3900	3	1100	1600	6.0	

- **Data has "actual_delivery_time" and "created_at" which is important to calculate total time taken to deliver.**
- Data also have "total_items" and "num_distinct_items" which will be helpful to determine if more number of item in single order taking significantly more time or not.

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

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

In [5]:
```
df.columns
```

Out[5]:
```
Index(['market_id', 'created_at', 'actual_delivery_time', 'store_id',
       'store_primary_category', 'order_protocol', 'total_items', 'subtotal',
       'num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_partners', 'total_busy_partners',
       'total_outstanding_orders'],
      dtype='object')
```

In [6]:

```
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
```

- created_at and actual_delivery_time given as object but it should be as timestamp.
- Some feature consist of numerical values and some are string

In [7]:

```
df["created_at"]= pd.to_datetime(df["created_at"])
df["actual_delivery_time"]= pd.to_datetime(df["actual_delivery_time"])
```

In [8]:

```
df.describe()
```

Out[8]:

	market_id	created_at	actual_delivery_time	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy_partners	total_outstanding_orders
count	196441.000000	197428	197421	196433.000000	197428.000000	197428.000000	197428.000000	197428.000000	197428.000000	181166.000000	181166.000000	181166.000000
mean	2.978706	2015-02-04 22:00:09.537962752	2015-02-04 22:48:23.348914432	2.882352	3.196391	2682.331402	2.670791	686.218470	1159.588630	44.808093	41.739747	58.050065
min	1.000000	2014-10-19 05:24:15	2015-01-21 15:58:11	1.000000	1.000000	0.000000	1.000000	-86.000000	0.000000	-4.000000	-5.000000	-6.000000
25%	2.000000	2015-01-29 02:32:42	2015-01-29 03:22:29	1.000000	2.000000	1400.000000	1.000000	299.000000	800.000000	17.000000	15.000000	17.000000
50%	3.000000	2015-02-05 03:29:09.500000	2015-02-05 04:40:41	3.000000	3.000000	2200.000000	2.000000	595.000000	1095.000000	37.000000	34.000000	41.000000
75%	4.000000	2015-02-12 01:39:18.500000	2015-02-12 02:25:26	4.000000	4.000000	3395.000000	3.000000	949.000000	1395.000000	65.000000	62.000000	85.000000
max	6.000000	2015-02-18 06:00:44	2015-02-19 22:45:31	7.000000	411.000000	27100.000000	20.000000	14700.000000	14700.000000	171.000000	154.000000	285.000000
std	1.524867	NaN	NaN	1.503771	2.666546	1823.093688	1.630255	522.038648	558.411377	34.526783	32.145733	52.661830

- we have order created data from 2015-02-04 to 2015-02-18.
- Maximum total item in single delivery is 411
- Maximum subtotal is 27100 but 75% have subtotal less than equal to 3395
- Maximum Number of distinct Item in single delivery is 20.
- 50% of the time there are less than or equal to 2 items in each delivery

In [9]:
```
df.describe(include= "object")
```

Out[9]:

	store_id	store_primary_category
count	197428	192668
unique	6743	74
top	d43ab110ab2489d6b9b2caa394bf920f	american
freq	937	19399

- Most frequent store is have store_id "d43ab110ab2489d6b9b2caa394bf920f"
- Most frequent category of restaurant is "american"

In [10]:
```
df.isna().sum()
```

Out[10]:
```
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 [11]:
```
# In percentage
df.isna().sum()/ len(df)*100
```

Out[11]:
```
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
```

- most missing values are in "total_onshift_partners","total_busy_partners", "total_outstanding_orders" have same amount of missing value i.e. 8.24%
- followed by 2nd highest is store_primary_category
- "actual_delivery_time" have lowest number of missing value (0.0035%)
- we can impute the missing values in "total_onshift_partners","total_busy_partners", "total_outstanding_orders" using group by market_id and mean values per market.

In [12]:
```python
for i in df:
    print(f" unique values in {i:<30}----- => { df[i].nunique()}")
```

```
unique values in market_id                      ----- => 6
unique values in created_at                     ----- => 180985
unique values in actual_delivery_time           ----- => 178110
unique values in store_id                       ----- => 6743
unique values in store_primary_category         ----- => 74
unique values in order_protocol                 ----- => 7
unique values in total_items                    ----- => 57
unique values in subtotal                       ----- => 8368
unique values in num_distinct_items             ----- => 20
unique values in min_item_price                 ----- => 2312
unique values in max_item_price                 ----- => 2652
unique values in total_onshift_partners         ----- => 172
unique values in total_busy_partners            ----- => 159
unique values in total_outstanding_orders       ----- => 281
```

Checking "store_primary_category" feature structure and characteristic

In [13]:
```python
df["store_primary_category"].nunique()
```

Out[13]:
```
74
```

In [14]:
```python
df["store_primary_category"].value_counts()
```

Out[14]:
```
store_primary_category
american            19399
pizza               17321
mexican             17099
burger              10958
sandwich            10060
                    ...
lebanese                9
belgian                 2
indonesian              2
chocolate               1
alcohol-plus-food       1
Name: count, Length: 74, dtype: int64
```

In [15]:
```python
df["store_primary_category"].unique()
```

Out[15]:
```
array(['american', 'mexican', nan, 'indian', 'italian', 'sandwich',
       'thai', 'cafe', 'salad', 'pizza', 'chinese', 'singaporean',
       'burger', 'breakfast', 'mediterranean', 'japanese', 'greek',
       'catering', 'filipino', 'convenience-store', 'other', 'korean',
       'vegan', 'asian', 'barbecue', 'fast', 'dessert', 'smoothie',
       'seafood', 'vietnamese', 'cajun', 'steak', 'middle-eastern',
       'soup', 'vegetarian', 'persian', 'nepalese', 'sushi',
       'latin-american', 'hawaiian', 'chocolate', 'burmese', 'british',
       'pasta', 'alcohol', 'dim-sum', 'peruvian', 'turkish', 'malaysian',
       'ethiopian', 'afghan', 'bubble-tea', 'german', 'french',
       'caribbean', 'gluten-free', 'comfort-food', 'gastropub',
       'pakistani', 'moroccan', 'spanish', 'southern', 'tapas', 'russian',
       'brazilian', 'european', 'cheese', 'african', 'argentine',
       'kosher', 'irish', 'lebanese', 'belgian', 'indonesian',
       'alcohol-plus-food'], dtype=object)
```

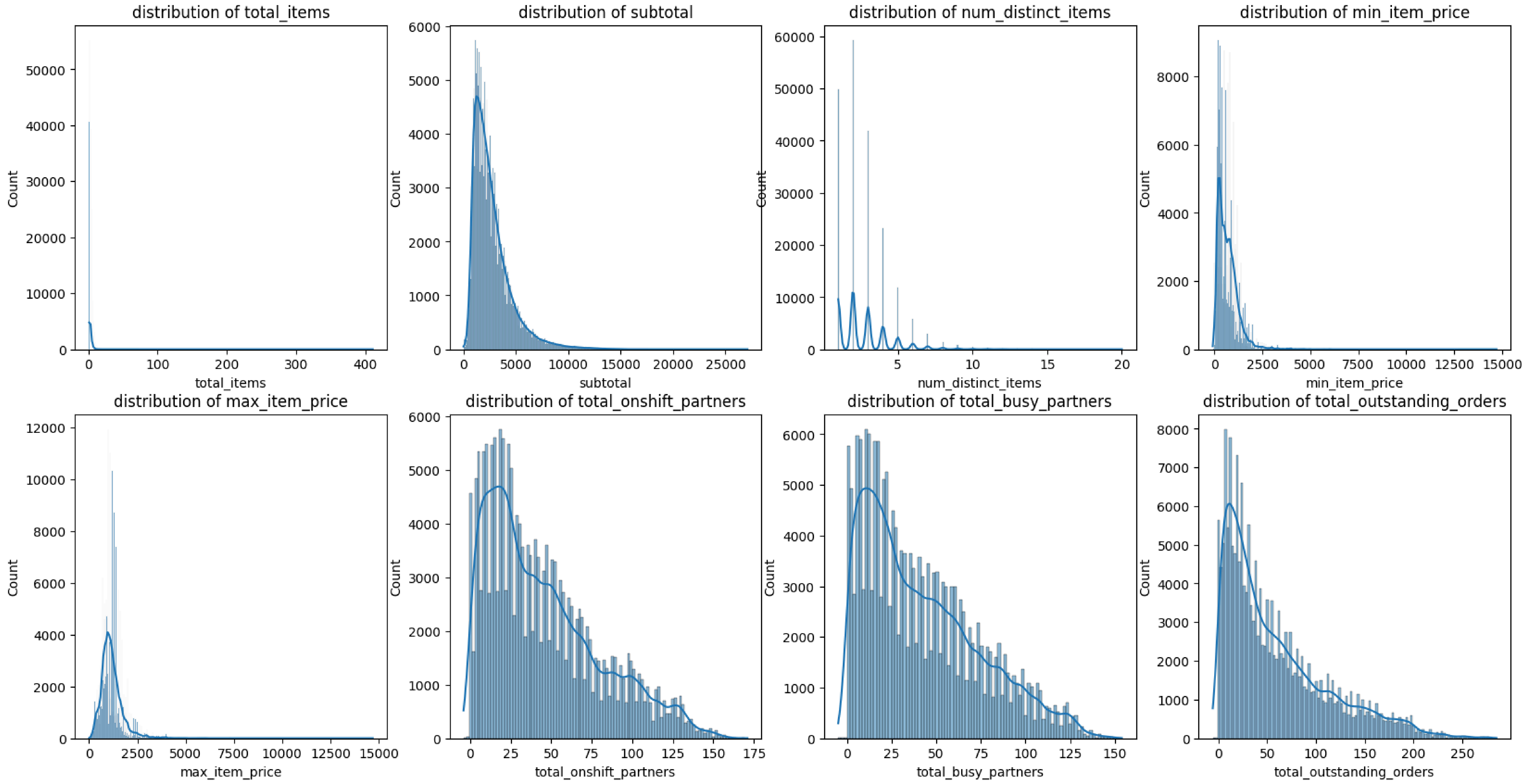
Checking distribution of all the numerical feature

In [16]:
```
numerical_columns = df.select_dtypes(include=['int', 'float']).columns.tolist()
numerical_columns= numerical_columns[2:]
numerical_columns
```

Out[16]:
```
['total_items',
 'subtotal',
 'num_distinct_items',
 'min_item_price',
 'max_item_price',
 'total_onshift_partners',
 'total_busy_partners',
 'total_outstanding_orders']
```

In [17]:

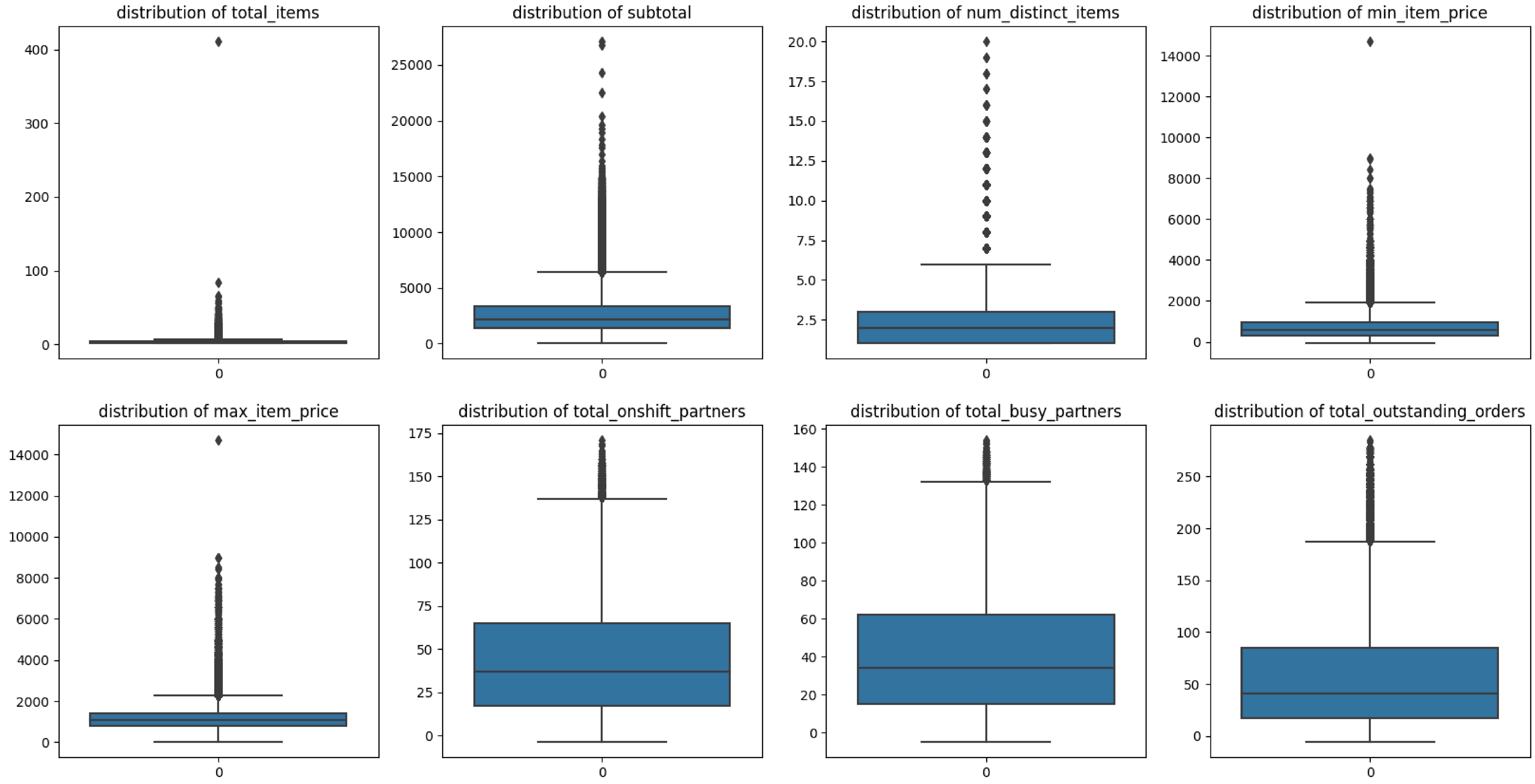
```python
fig = plt.figure(figsize=(20, 10))

for i in range(len(numerical_columns)):
    plt.subplot(2, 4, i+1)
    sns.histplot( df[numerical_columns[i]], kde= True)
    plt.title(f"distribution of {numerical_columns[i]}")
```



- All the distribution showing right skweness, indicating outliers.

In [18]:

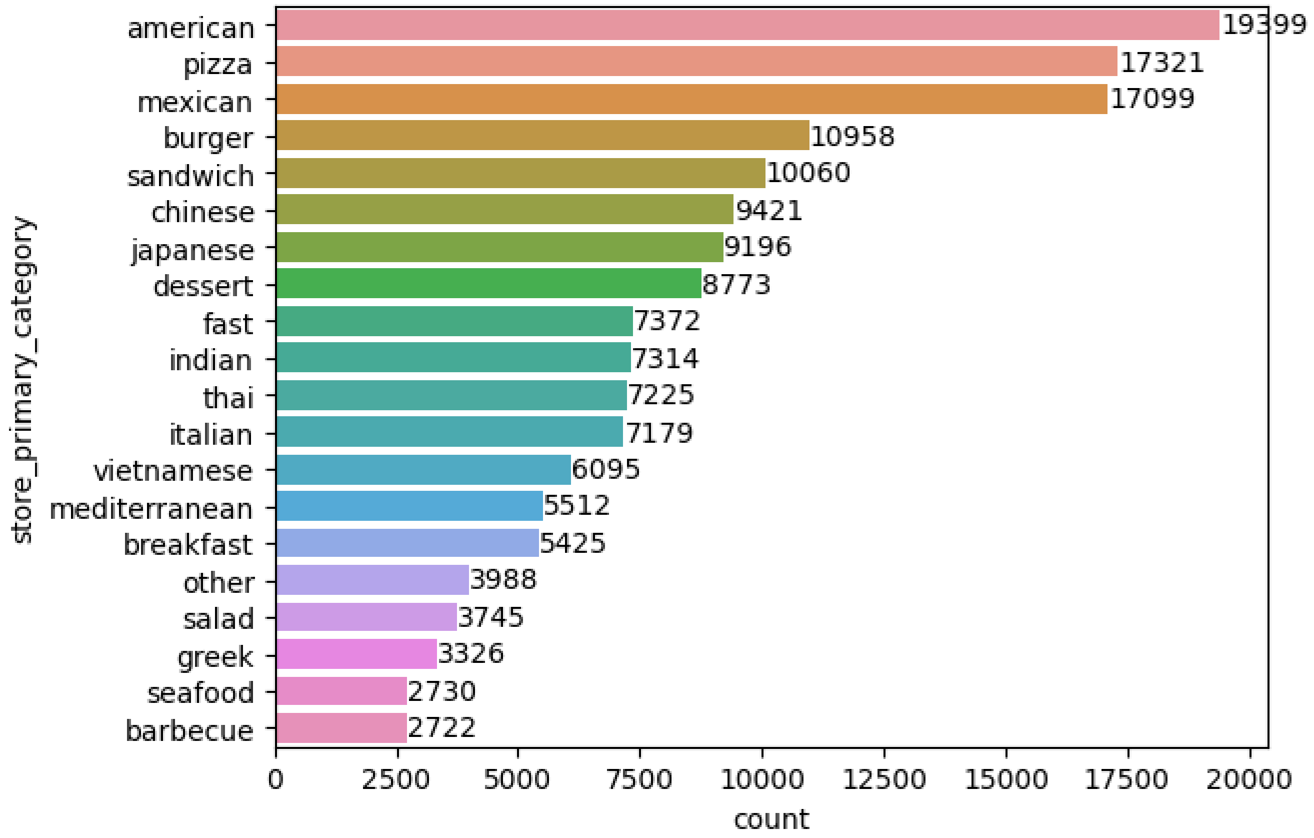
```
fig = plt.figure(figsize=(20, 10))

for i in range(len(numerical_columns)):
    plt.subplot(2, 4, i+1)
    sns.boxplot( df[numerical_columns[i]])
    plt.title(f"distribution of {numerical_columns[i]}")
```



- There are some extreme outliers which maybe not following natural pattern.

Top 20 category for the restaurant based on Number of delivery from this store

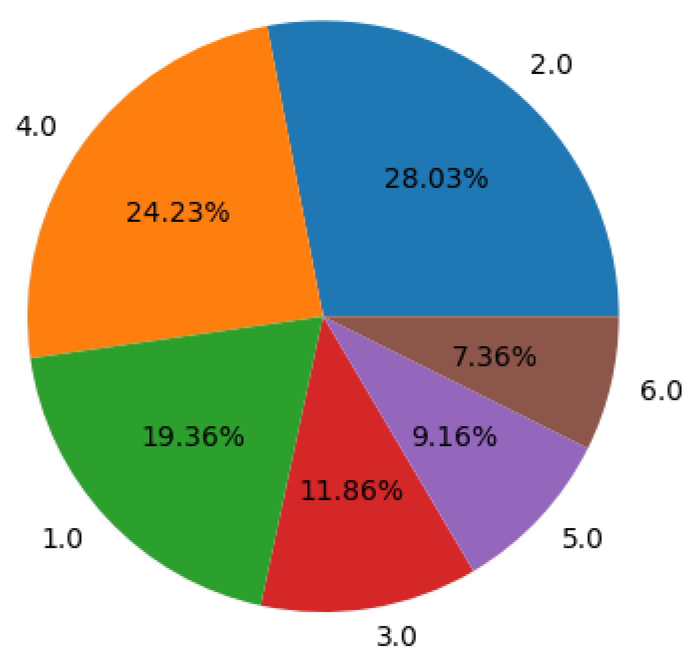
In [19]:
```python
lbl= sns.barplot(x= df["store_primary_category"].value_counts()[:20], y= df["store_primary_category"].value_counts()[:20].index)
for i in lbl.containers:
    lbl.bar_label(i)

plt.show()
```



In [20]:
```python
df["market_id"].value_counts()
```

Out[20]:
```
market_id
2.0    55058
4.0    47599
1.0    38037
3.0    23297
5.0    18000
6.0    14450
Name: count, dtype: int64
```

In [21]:
```
plt.pie(df["market_id"].value_counts(), labels= df["market_id"].value_counts().index, autopct='%.2f%%' )
plt.show()
```



- 28% order are from Market_id 2.0 followed by market_id 4.0
- Market_id 6.0 has lowest orders.

- **We have created_at and actual_delivery_time, from this we can calculate total time taken to deliver i.e. dependent variable.**
- By using timedelta we can convert time_taken into minutes.

In [22]:
```
df["time_taken"] = df["actual_delivery_time"]- df["created_at"]
```

In [23]:
```
df["time_taken_min"]= pd.to_timedelta(df["time_taken"])/pd.Timedelta("60s")
```

In [24]:
```
df.head()
```

Out[24]:

	market_id	created_at	actual_delivery_time	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy_pa
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	df263d996281d984952c07998dc54358	american	1.0	4	3441	4	557	1239	33.0	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	f0ade77b43923b38237db569b016ba25	mexican	2.0	1	1900	1	1400	1400	1.0	
2	3.0	2015-01-22 20:39:28	2015-01-22 21:09:09	f0ade77b43923b38237db569b016ba25	NaN	1.0	1	1900	1	1900	1900	1.0	
3	3.0	2015-02-03 21:21:45	2015-02-03 22:13:00	f0ade77b43923b38237db569b016ba25	NaN	1.0	6	6900	5	600	1800	1.0	
4	3.0	2015-02-15 02:40:36	2015-02-15 03:20:26	f0ade77b43923b38237db569b016ba25	NaN	1.0	3	3900	3	1100	1600	6.0	

- Deriving the feature hour_created_at and day of week at which order was created from created_at

In [25]:
```
df["hour_created"]= df["created_at"].dt.hour
df["day_of_week"]= df["created_at"].dt.dayofweek
```

checking distribution of target column i.e. time_taken_min

In [26]:
```
df["time_taken_min"].describe()
```

Out[26]:
```
count    197421.000000
mean         48.470956
std         320.493482
min           1.683333
25%          35.066667
50%          44.333333
75%          56.350000
max      141947.650000
Name: time_taken_min, dtype: float64
```

- **75% of the time time taken to deliver is less than or equal to 56 minutes.**
- mean time to deliver the order is 48.47 minutes.

checking for orders with more than 1000 minutes

In [27]:
```
df.loc[df["time_taken_min"]>1000]
```

Out[27]:

	market_id	created_at	actual_delivery_time	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy
2690	1.0	2014-10-19 05:24:15	2015-01-25 19:11:54	675f9820626f5bc0afb47b57890b466e	italian	1.0	1	1695	1	1595	1595	NaN	
27189	1.0	2015-02-16 02:24:09	2015-02-19 22:45:31	d397c2b2be2178fe6247bd50fc97cff2	indian	3.0	4	4980	4	995	1795	72.0	
185550	4.0	2015-01-28 08:34:06	2015-02-01 16:25:25	1679091c5a880faf6fb5e6087eb1b2dc	dessert	5.0	3	1520	3	220	750	0.0	

- There are only 3 orders which took more than 5000 minutes for delivery.
- we can drop this

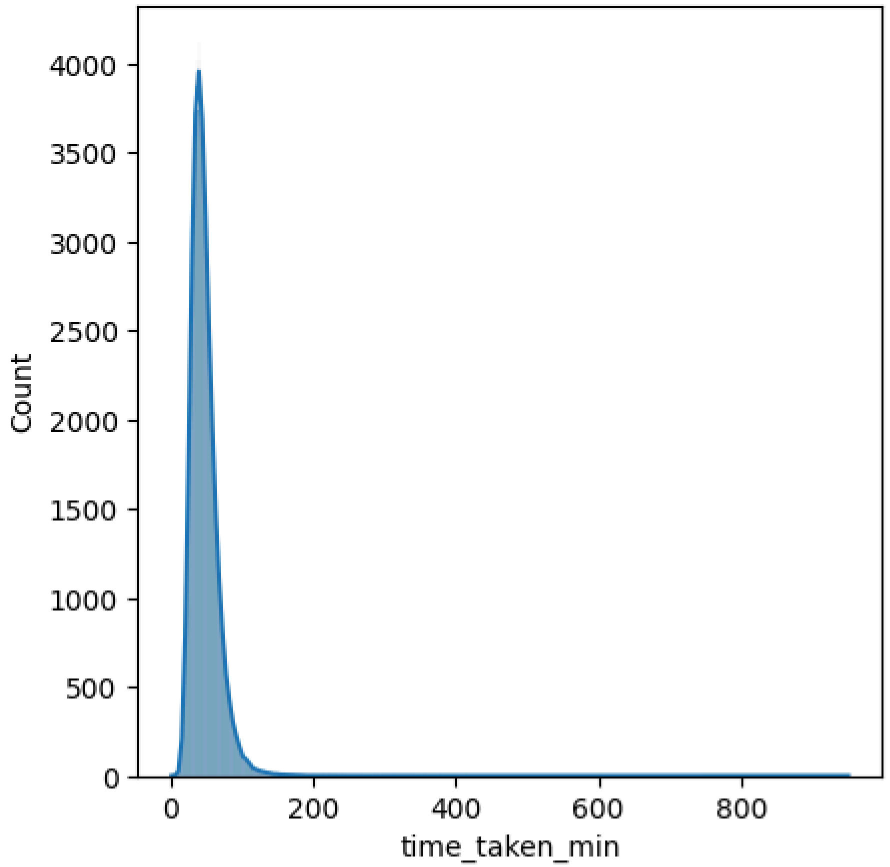
In [28]:
```
df= df.loc[df["time_taken_min"]<1000]
```

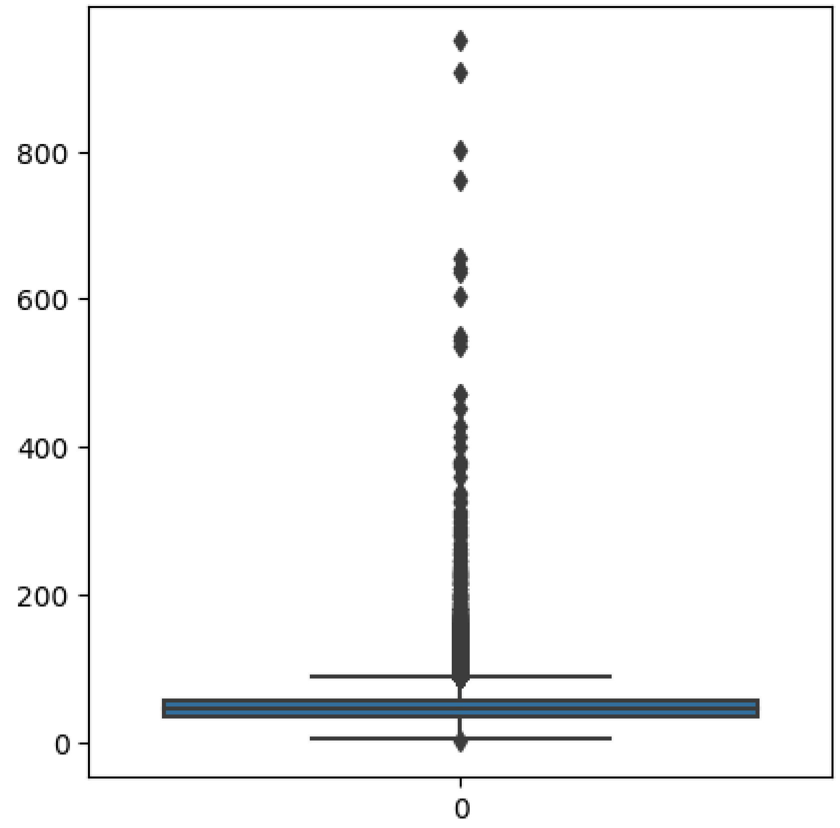
In [29]:
```
df.shape
```

Out[29]: (197418, 18)

In [30]:

```
plt.figure(figsize = (12, 5))
plt.subplot(1, 2, 1)
plt.subplots_adjust(wspace= 0.5)
sns.histplot(df['time_taken_min'], kde=True)


plt.subplot(1, 2, 2)
sns.boxplot(df['time_taken_min'])

plt.show()
```





Top stores with most orders in percentage

In [31]:

```
df["store_id"].value_counts(normalize=True)*100
```

Out[31]:

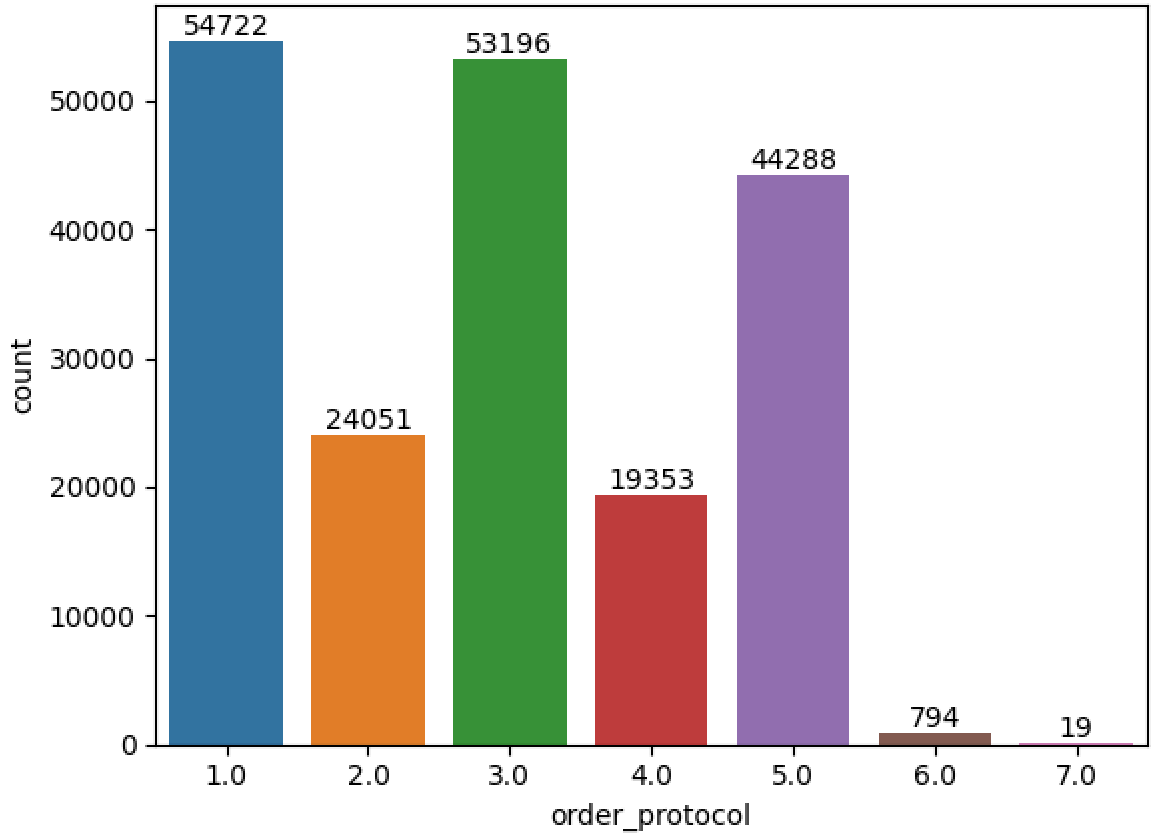
```
store_id
d43ab110ab2489d6b9b2caa394bf920f    0.474627
757b505cfd34c64c85ca5b5690ee5293    0.437144
faacbcd5bf1d018912c116bf2783e9a1    0.412323
cfecdb276f634854f3ef915e2e980c31    0.387503
45c48cce2e2d7fbdea1afc51c7c6ad26    0.365215
                                      ...
adad0f2b196a1ed3e3b9d9025c397132    0.000507
2e6d9c6052e99fcdfa61d9b9da273ca2    0.000507
25daeb9b3072e9c53f66a2196a92a011    0.000507
55285adfd78a019a3245917649e29b3c    0.000507
df263d996281d984952c07998dc54358    0.000507
Name: proportion, Length: 6743, dtype: float64
```

Order Protocol- how the order was placed.

In [32]:
```
# In percentage
df["order_protocol"].value_counts(normalize=True)*100
```

Out[32]:
```
order_protocol
1.0    27.859263
3.0    27.082368
5.0    22.547258
2.0    12.244493
4.0     9.852716
6.0     0.404230
7.0     0.009673
Name: proportion, dtype: float64
```

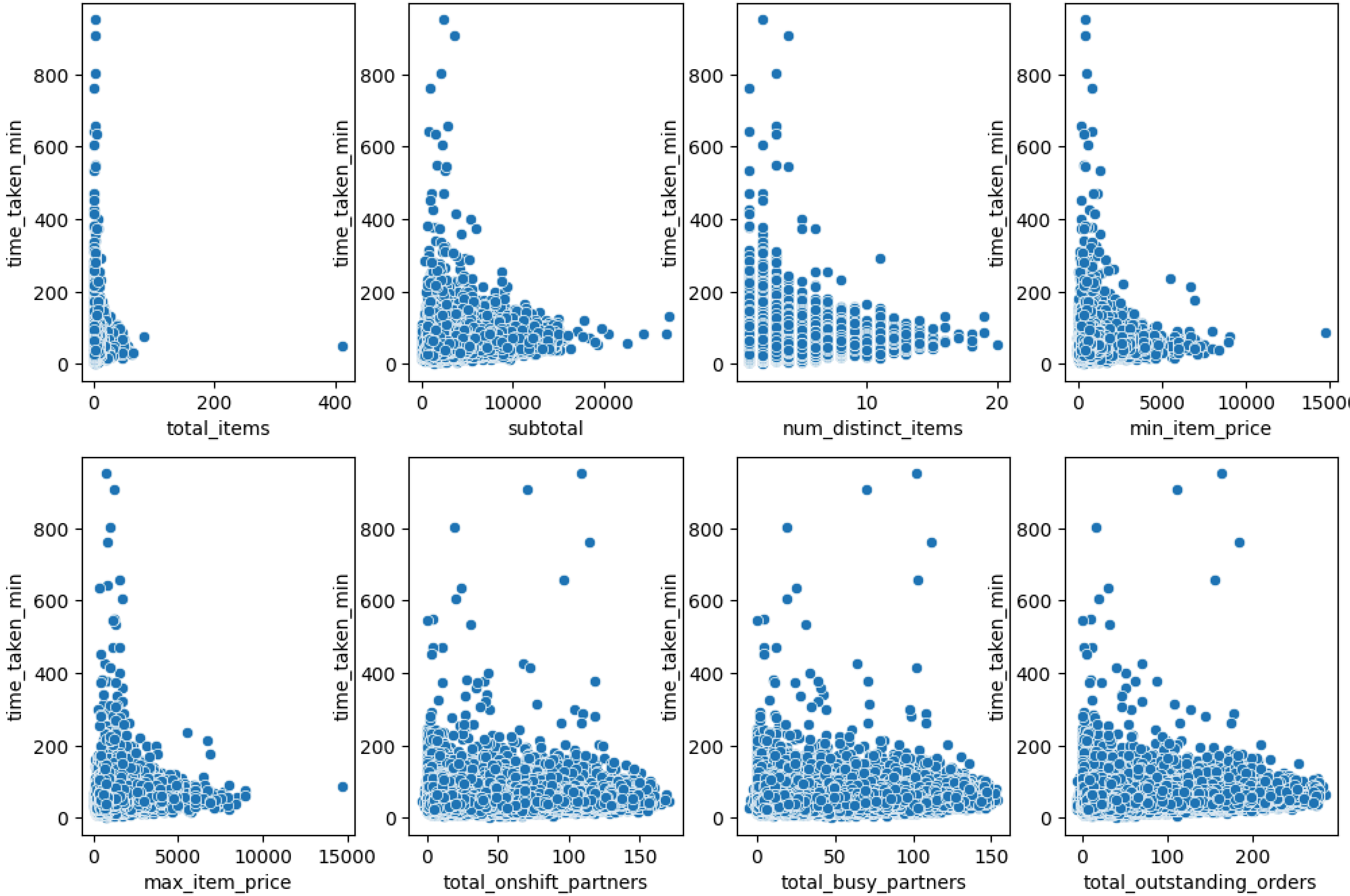
In [33]:
```
labl= sns.countplot(data=df, x=df["order_protocol"])
for i in labl.containers:
    labl.bar_label(i)
plt.show()
```



- **Most of the oder was placed through "1.0" and "3.0" type of mode.**
- Very Few order was place through "7.0"

In [34]:

```python
rows, cols = 2,4
fig, axs = plt.subplots(rows,cols, figsize=(12, 8))
index = 0
for row in range(rows):
    for col in range(cols):
        sns.scatterplot(x=numerical_columns[index], y=df["time_taken_min"], data=df, ax=axs[row, col])
        index += 1

```



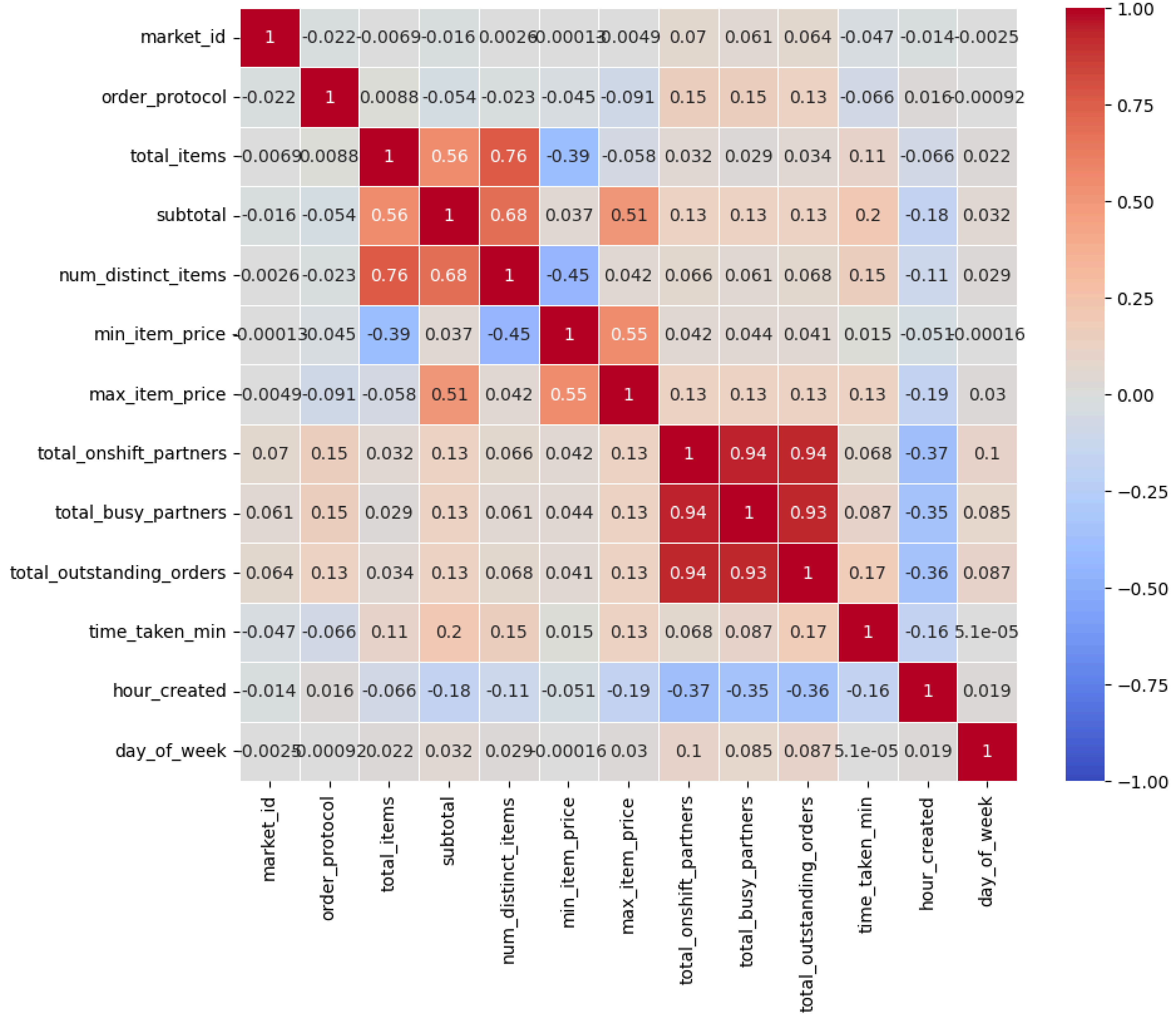
- As we can see there is not much impact on delivery of order with different variable like total_items, max_item_price, busy_partners.

In [35]:
```python
numerical_columns = df.select_dtypes(include=['int', 'float']).columns.tolist()
#numerical_columns= numerical_columns[2:]
numerical_columns
```

Out[35]:
```
['market_id',
 'order_protocol',
 'total_items',
 'subtotal',
 'num_distinct_items',
 'min_item_price',
 'max_item_price',
 'total_onshift_partners',
 'total_busy_partners',
 'total_outstanding_orders',
 'time_taken_min',
 'hour_created',
 'day_of_week']
```

In [36]:

```
plt.figure(figsize = (10,8))
dt_corr= df[numerical_columns].corr()
sns.heatmap(dt_corr, annot= True, cmap='coolwarm', linewidth=.5, vmin=-1, vmax=1)
plt.show()
```



Group by market to get min, max and Avg time taken market wise

In [37]:
```
df.groupby(["market_id"])["time_taken_min"].agg({"min", "max", "mean"}).reset_index().sort_values(by = "mean", ascending=False)
```

Out[37]:

	market_id	max	mean	min
0	1.0	907.450000	51.331649	6.766667
2	3.0	802.966667	47.695185	7.833333
5	6.0	640.950000	47.258626	9.283333
3	4.0	535.983333	47.236536	6.433333
4	5.0	381.866667	46.495435	10.316667
1	2.0	950.533333	46.073661	1.683333

In [38]:
```
#cheking duplicate record
df.duplicated().sum()
```

Out[38]: 0

Treating missing values

In [39]:
```
df.isna().sum()/len(df)*100
```

Out[39]:
```
market_id                   0.499954
created_at                  0.000000
actual_delivery_time        0.000000
store_id                    0.000000
store_primary_category      2.411128
order_protocol              0.504007
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      8.236838
total_busy_partners         8.236838
total_outstanding_orders    8.236838
time_taken                  0.000000
time_taken_min              0.000000
hour_created                0.000000
day_of_week                 0.000000
dtype: float64
```

In [40]:
```
df['total_onshift_partners'] = df.groupby('market_id')['total_onshift_partners'].transform(lambda x: x.fillna(x.mean()))
df['total_busy_partners'] = df.groupby('market_id')['total_busy_partners'].transform(lambda x: x.fillna(x.mean()))
df['total_outstanding_orders'] = df.groupby('market_id')['total_outstanding_orders'].transform(lambda x: x.fillna(x.mean()))


```

In [41]:
```
def fill_mode(col):
    mode_value = col.mode()
    if len(mode_value) > 0:
        return col.fillna(mode_value.iloc[0])
    else:
        return col
```

In [42]:
```
df['store_primary_category'] = df.groupby('store_id')['store_primary_category'].transform(fill_mode)
```

In [43]:
```
df.isna().sum()/len(df)*100
```

Out[43]:
```
market_id                   0.499954
created_at                  0.000000
actual_delivery_time        0.000000
store_id                    0.000000
store_primary_category      0.439170
order_protocol              0.504007
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      0.499954
total_busy_partners         0.499954
total_outstanding_orders    0.499954
time_taken                  0.000000
time_taken_min              0.000000
hour_created                0.000000
day_of_week                 0.000000
dtype: float64
```

- Now we can drop null values, all the null values is less than 0.5%.

In [44]:
```
df = df.dropna()
```

In [45]:
```
df.isna().sum()/len(df)*100
```

Out[45]:
```
market_id                   0.0
created_at                  0.0
actual_delivery_time        0.0
store_id                    0.0
store_primary_category      0.0
order_protocol              0.0
total_items                 0.0
subtotal                    0.0
num_distinct_items          0.0
min_item_price              0.0
max_item_price              0.0
total_onshift_partners      0.0
total_busy_partners         0.0
total_outstanding_orders    0.0
time_taken                  0.0
time_taken_min              0.0
hour_created                0.0
day_of_week                 0.0
dtype: float64
```

- Since we calculated time_taken_min we can drop related columns

In [46]:
```
df.drop(['created_at', 'actual_delivery_time', 'time_taken'], axis=1, inplace=True)
```

In [47]:
```
df.tail(2)
```

Out[47]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy_partners	total_outstanding_ord
197426	1.0	c81e155d85dae5430a8cee6f2242e82c	sandwich	1.0	1	1175	1	535	535	7.0	7.0	1
197427	1.0	c81e155d85dae5430a8cee6f2242e82c	sandwich	1.0	4	2605	4	425	750	20.0	20.0	2

- feature like market_id, order_protocol should be categorical

In [48]:
```
cat_col= ["market_id", "order_protocol"]
df[cat_col]= df[cat_col].astype("str")
```

In [49]:
```
df.info()
```

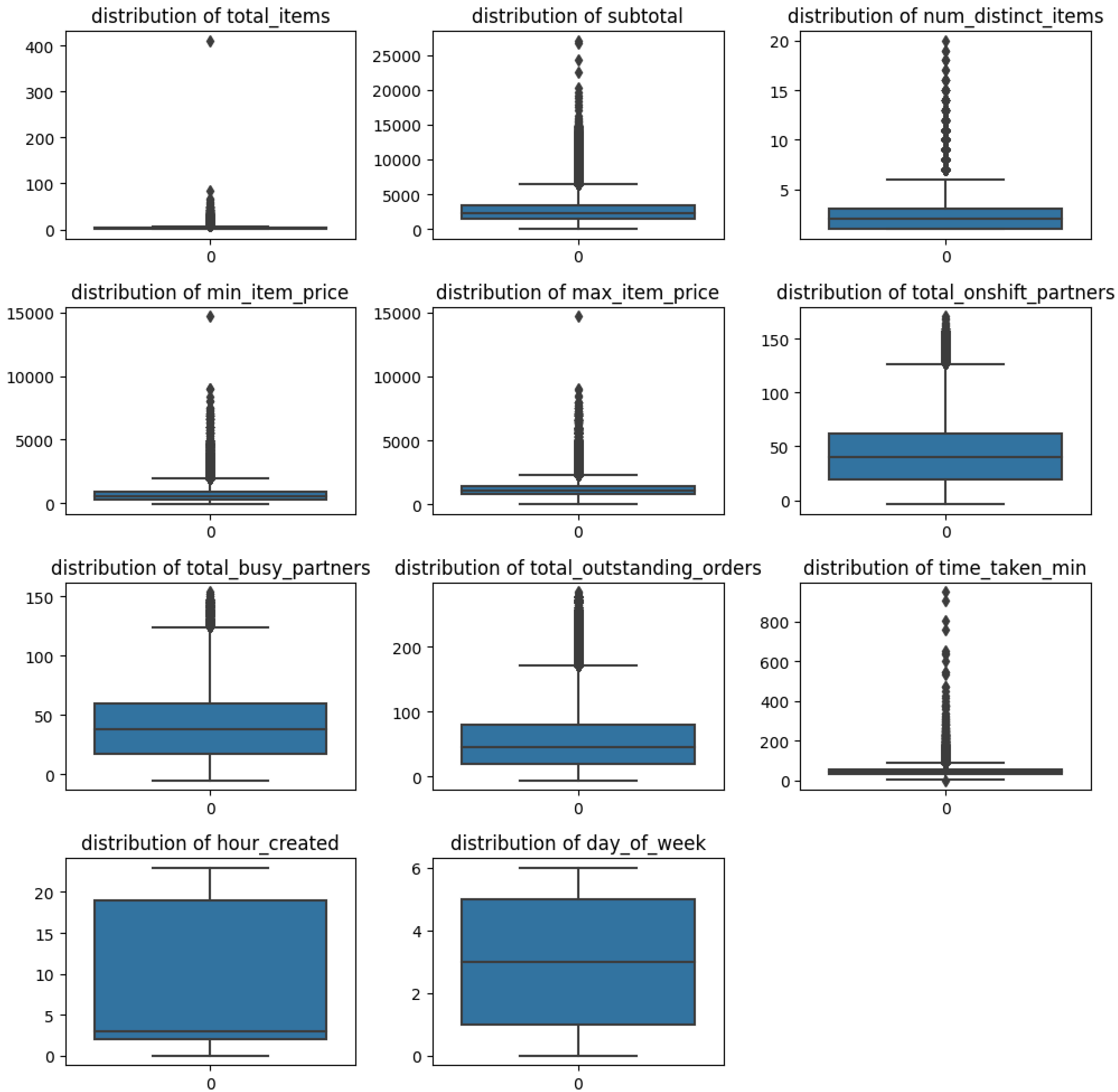
```
<class 'pandas.core.frame.DataFrame'>
Index: 195059 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype  
---  ------                    --------------   -----  
 0   market_id                 195059 non-null  object 
 1   store_id                  195059 non-null  object 
 2   store_primary_category    195059 non-null  object 
 3   order_protocol            195059 non-null  object 
 4   total_items               195059 non-null  int64  
 5   subtotal                  195059 non-null  int64  
 6   num_distinct_items        195059 non-null  int64  
 7   min_item_price            195059 non-null  int64  
 8   max_item_price            195059 non-null  int64  
 9   total_onshift_partners    195059 non-null  float64
 10  total_busy_partners       195059 non-null  float64
 11  total_outstanding_orders  195059 non-null  float64
 12  time_taken_min            195059 non-null  float64
 13  hour_created              195059 non-null  int32  
 14  day_of_week               195059 non-null  int32  
dtypes: float64(4), int32(2), int64(5), object(4)
memory usage: 22.3+ MB
```

In [50]:
```
numerical_columns = df.select_dtypes(include=['int', 'float']).columns.tolist()
#numerical_columns= numerical_columns[2:]
numerical_columns
```

Out[50]:
```
['total_items',
 'subtotal',
 'num_distinct_items',
 'min_item_price',
 'max_item_price',
 'total_onshift_partners',
 'total_busy_partners',
 'total_outstanding_orders',
 'time_taken_min',
 'hour_created',
 'day_of_week']
```

In [51]:

```
fig = plt.figure(figsize=(10, 10))

for i in range(len(numerical_columns)):
    plt.subplot(4, 3, i+1)
    fig.tight_layout()
    sns.boxplot( df[numerical_columns[i]])
    plt.title(f"distribution of {numerical_columns[i]}")
```


distribution of total_items
distribution of subtotal
distribution of num_distinct_items
distribution of min_item_price
distribution of max_item_price
distribution of total_onshift_partners
distribution of total_busy_partners
distribution of total_outstanding_orders
distribution of time_taken_min
distribution of hour_created
distribution of day_of_week

In [52]:
```
df.loc[df["total_items"]>80]
```

Out[52]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy_partners	total_outstanding_orc
47231	2.0	c156cea027720c227089e679b3ae9d1b	fast	4.0	411	3115	5	0	299	35.000000	35.000000	39.000
182223	6.0	e50372d3fee4eadec9c42aa6528097cc	fast	4.0	84	1016	4	0	289	44.929771	41.896183	58.439

- As we can see there is only one record which have total_items more than 200

In [53]:
```
df= df.loc[df["total_items"]<80]
```

In [54]:
```
df.loc[df["max_item_price"]>10000]
```

Out[54]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy_partners	total_outstanding_ord
139718	3.0	36452e720502e4da486d2f9f6b48a7bb	breakfast	5.0	1	14700	1	14700	14700	23.0	21.0	2

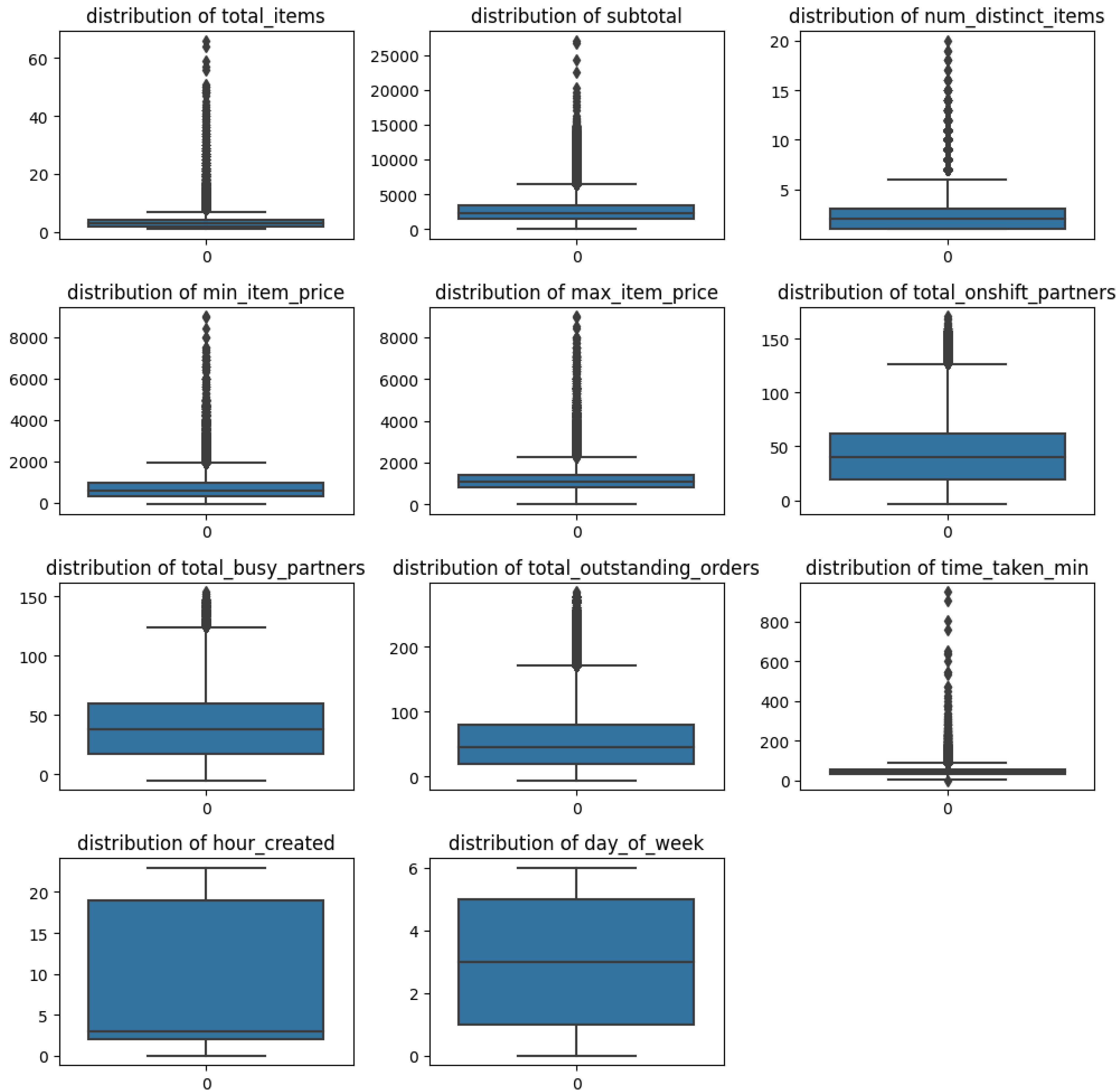
In [55]:
```
df= df.loc[df["max_item_price"]<10000]
```

In [56]:
```
df.shape
```

Out[56]: (195056, 15)

In [57]:

```
fig = plt.figure(figsize=(10, 10))

for i in range(len(numerical_columns)):
    plt.subplot(4, 3, i+1)
    fig.tight_layout()
    sns.boxplot( df[numerical_columns[i]])
    plt.title(f"distribution of {numerical_columns[i]}")
```


distribution of total_items
distribution of subtotal
distribution of num_distinct_items
distribution of min_item_price
distribution of max_item_price
distribution of total_onshift_partners
distribution of total_busy_partners
distribution of total_outstanding_orders
distribution of time_taken_min
distribution of hour_created
distribution of day_of_week

In [58]:

```
df.loc[df["time_taken_min"]>600]
```

Out[58]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy_partners	total_outstanding_or
66787	6.0	e85ca00d008a532279b798033d59a4c7	italian	1.0	1	795	1	795	795	44.929771	41.896183	58.43
76743	2.0	9380e398ee9bea45b992a3daaa6b7c4d	pizza	6.0	1	990	1	795	795	114.000000	112.000000	184.00
83055	2.0	c1502ae5a4d514baec129f72948c266e	burger	4.0	3	2379	2	389	695	109.000000	102.000000	163.00
86952	3.0	831b342d8a83408e5960e9b0c5f31f0c	thai	2.0	3	2185	3	495	995	19.000000	19.000000	16.00
105825	2.0	5fc7c9bd1fcb12799f02da8adfa4954f	alcohol	5.0	3	2850	3	200	1500	96.000000	103.000000	156.00
139989	3.0	a8e5a72192378802318bf51063153729	mediterranean	3.0	2	2300	2	600	1700	20.000000	19.000000	20.00
175971	1.0	5d616dd38211ebb5d6ec52986674b6e4	mexican	1.0	5	1530	3	300	315	24.000000	25.000000	30.00
190860	1.0	b132ecc1609bfcf302615847c1caa69a	indian	3.0	4	3660	4	375	1195	71.000000	70.000000	111.00

In [59]:

```
df= df.loc[df["time_taken_min"]<600]
```

In [60]:

```
# Independent variables
X= df.drop("time_taken_min", axis=1)
```

In [61]:

```
#target variable
y= df["time_taken_min"]
```

Splitting the data into Train, Validation and Test Data

In [62]:

```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, random_state=42)

print('Train : ', X_train.shape, y_train.shape)
print('Validation:', X_val.shape, y_val.shape)
print('Test  : ', X_test.shape, y_test.shape)
```

```
Train :  (124830, 14) (124830,)
Validation: (31208, 14) (31208,)
Test  :  (39010, 14) (39010,)
```

Encoding categorical columns

In [63]:

```
enc = TargetEncoder(cols=["market_id", "store_id", "store_primary_category", "order_protocol"])
X_train = enc.fit_transform(X_train, y_train)

X_val = enc.transform(X_val)
X_test = enc.transform(X_test)
```

In [64]:
```
X_train.head()
```

Out[64]:

	market_id	store_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_partners	total_busy_partners	total_outstanding_orders	hour_created	day_o
45046	46.099077	45.011699	47.334094	45.567002	1	1075	1	725	725	73.0	74.0	101.0	20	
87732	51.304882	49.318093	47.899950	49.918444	2	1150	2	295	855	20.0	15.0	16.0	3	
162823	46.429496	41.474057	46.279055	47.395853	5	1984	5	335	399	18.0	18.0	29.0	3	
159857	46.429496	44.946896	50.192517	48.594070	2	1143	2	279	729	14.0	12.0	12.0	23	
136409	46.099077	41.596319	45.531750	46.895346	4	1475	4	250	695	60.0	59.0	71.0	4	

Scaling the data for NN

In [65]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)

X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

Keras Sequential API

Simple model

In [66]:
```
def baseline():
    model= Sequential()
    model.add(Dense(32,kernel_initializer='normal', activation= "relu", input_shape= (X_train.shape[1],)))
    model.add(Dense(16, activation= "tanh"))
    model.add(Dense(12, activation="relu"))
    model.add(Dense(8, activation="relu"))
    model.add(Dense(4))
    model.add(Dense(1))
    return model
```

In [67]:
```
simple_model= baseline()
```

In [68]:
```
simple_model.compile(optimizer="adam", loss="mean_squared_error", metrics=["mean_absolute_error"])
```

In [69]:
```
# simple model summary
simple_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 32)	480
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 12)	204
dense_3 (Dense)	(None, 8)	104
dense_4 (Dense)	(None, 4)	36
dense_5 (Dense)	(None, 1)	5

Total params: 1,357 (5.30 KB)

Trainable params: 1,357 (5.30 KB)

Non-trainable params: 0 (0.00 B)

In [70]:
```
hist= simple_model.fit(X_train, y_train, epochs= 15, validation_data=(X_val, y_val), verbose=1)
```

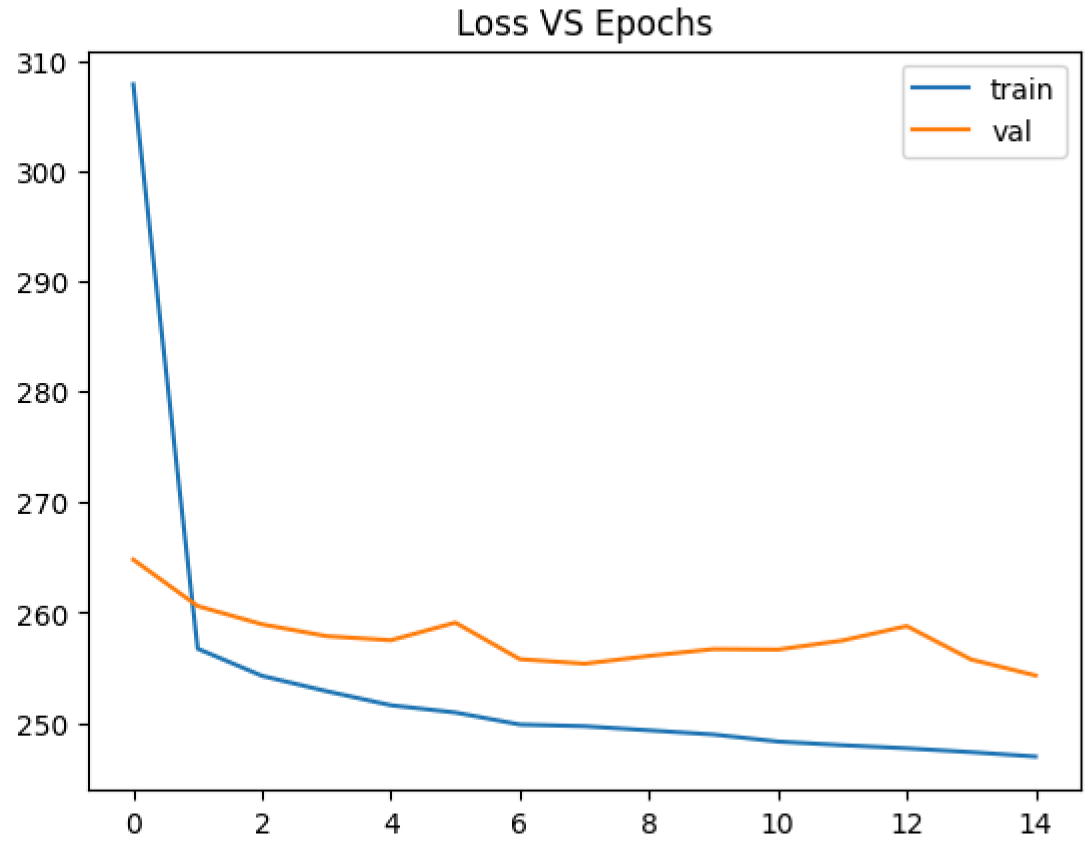
```
Epoch 1/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 469.1544 - mean_absolute_error: 15.0085 - val_loss: 264.8139 - val_mean_absolute_error: 11.2549
Epoch 2/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step - loss: 255.6663 - mean_absolute_error: 11.2893 - val_loss: 260.5962 - val_mean_absolute_error: 11.3811
Epoch 3/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 251.8255 - mean_absolute_error: 11.1864 - val_loss: 258.9371 - val_mean_absolute_error: 11.2535
Epoch 4/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 257.7316 - mean_absolute_error: 11.1421 - val_loss: 257.8633 - val_mean_absolute_error: 11.3856
Epoch 5/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 254.7227 - mean_absolute_error: 11.1492 - val_loss: 257.4925 - val_mean_absolute_error: 11.5078
Epoch 6/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 242.8393 - mean_absolute_error: 11.0477 - val_loss: 259.0905 - val_mean_absolute_error: 11.6076
Epoch 7/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 241.5830 - mean_absolute_error: 11.0278 - val_loss: 255.7721 - val_mean_absolute_error: 11.2334
Epoch 8/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 8s 2ms/step - loss: 248.9611 - mean_absolute_error: 11.0652 - val_loss: 255.3586 - val_mean_absolute_error: 11.3797
Epoch 9/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 245.3082 - mean_absolute_error: 11.0099 - val_loss: 256.0684 - val_mean_absolute_error: 11.3483
Epoch 10/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 252.9208 - mean_absolute_error: 11.0220 - val_loss: 256.6858 - val_mean_absolute_error: 11.3409
Epoch 11/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 245.4139 - mean_absolute_error: 11.0328 - val_loss: 256.6435 - val_mean_absolute_error: 11.1437
Epoch 12/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 247.9652 - mean_absolute_error: 11.0007 - val_loss: 257.4635 - val_mean_absolute_error: 11.5273
Epoch 13/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 245.4950 - mean_absolute_error: 10.9836 - val_loss: 258.7766 - val_mean_absolute_error: 11.2836
Epoch 14/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 251.4392 - mean_absolute_error: 10.9753 - val_loss: 255.7395 - val_mean_absolute_error: 11.1996
Epoch 15/15
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 244.2113 - mean_absolute_error: 10.9364 - val_loss: 254.2893 - val_mean_absolute_error: 11.1425
```

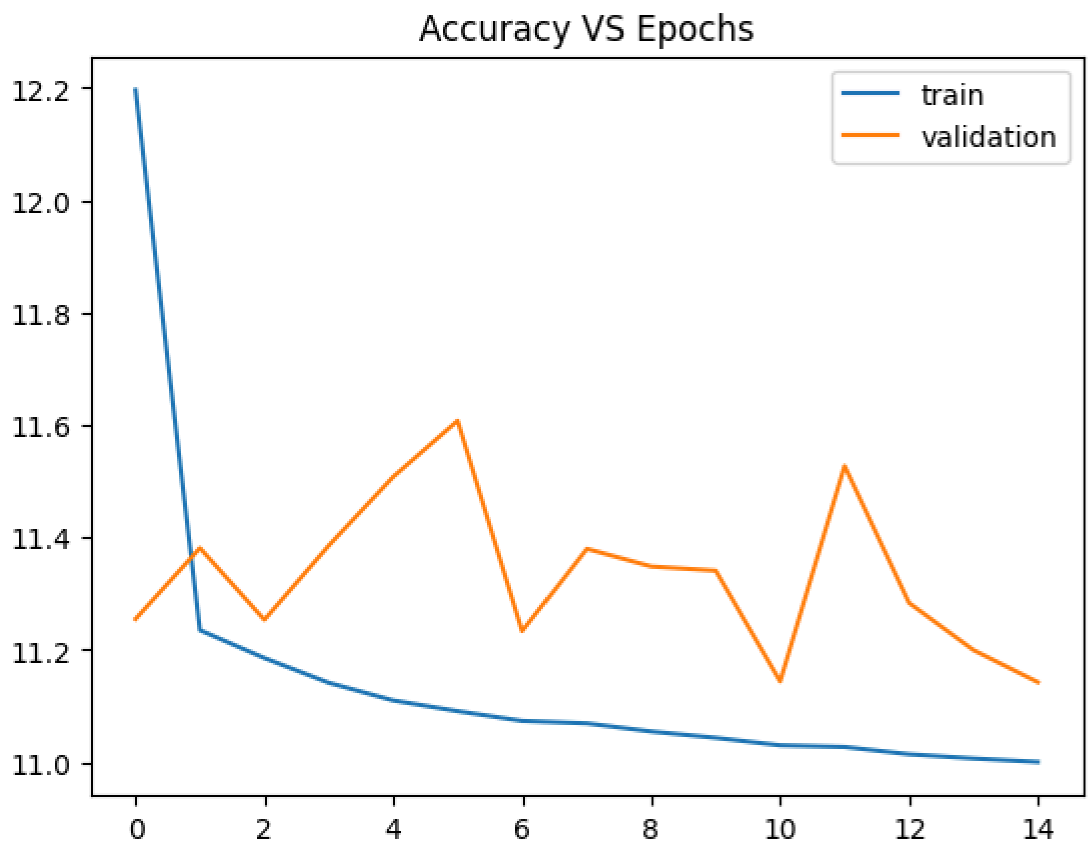
In [71]:

```
epochs = hist.epoch
train_MSE = hist.history["loss"]
train_MAE = hist.history["mean_absolute_error"]
val_MSE = hist.history["val_loss"]
val_MAE = hist.history["val_mean_absolute_error"]
```

In [72]:

```
plt.figure()
plt.plot(epochs, train_MSE, label="train")
plt.plot(epochs, val_MSE, label="val")
plt.legend()
plt.title("Loss VS Epochs")
plt.show()

plt.figure()
plt.plot(epochs, train_MAE, label="train")
plt.plot(epochs, val_MAE, label="validation")
plt.legend()
plt.title("Accuracy VS Epochs")
plt.show()
```





In [73]:
```
y_pred = simple_model.predict(X_test)
```

1220/1220 ━━━━━━━━━━━━━━━━━━━━ **2s** 2ms/step

In [74]:
```
r2_score_simple= r2_score(y_test, y_pred)
```

In [75]:
```
r2_score_simple
```

Out[75]: 0.26072215528845355

In [ ]:
```

```

Using Keras Tuner

In [102]:

```
def build_model(hp):
    model = Sequential()
    model.add(Dense(units=hp.Int("units_layer1", min_value=128, max_value=1028, step=32),
                    activation=hp.Choice("activation_layer1", ["relu", "tanh"]),  input_shape= (X_train.shape[1],)))
    model.add(BatchNormalization())


    model.add(Dense(units=hp.Int("units_layer2", min_value=100, max_value=500, step=100),
                    activation=hp.Choice("activation_layer2", ["relu", "tanh"])))

    model.add(Dense(units=hp.Int("units_layer2", min_value=50, max_value=150, step=10),
                    activation=hp.Choice("activation_layer2", ["relu", "tanh"])))

    model.add(Dense(units=hp.Int("units_layer3", min_value=16, max_value=64, step=8),
                    activation=hp.Choice("activation_layer3", ["relu", "tanh"])))
    model.add(Dense(units=hp.Int("units_layer3", min_value=4, max_value=16, step= 1),
                    activation=hp.Choice("activation_layer3", ["relu", "tanh"])))

    model.add(Dense(1))

    model.compile(
        optimizer="adam",
        loss="mean_squared_error",
        metrics=["mean_absolute_error"])


    return model


build_model(keras_tuner.HyperParameters())
```

Out[102]:

```
<Sequential name=sequential_1, built=True>
```

In [103]:

```
tuner = keras_tuner.BayesianOptimization(
    build_model,
    objective='val_mean_absolute_error',
    max_trials=10, overwrite=True)
```

In [104]:

```
tuner.search(X_train, y_train, epochs=10, validation_data= (X_val, y_val), callbacks=[EarlyStopping(patience=5, min_delta=80)], verbose= 1)
best_model = tuner.get_best_models()[0]
```

```
Trial 10 Complete [00h 02m 56s]
val_mean_absolute_error: 11.230664253234863

Best val_mean_absolute_error So Far: 11.16738224029541
Total elapsed time: 00h 14m 09s
```

In [105]:

```
best_model = tuner.get_best_models(num_models=1)[0]
```

In [106]:
```
best_model.fit(x=X_train, y= y_train, epochs=100, validation_data=(X_val, y_val), callbacks=[EarlyStopping(min_delta=80, patience=5)])
```

```
Epoch 1/100
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - loss: 249.2085 - mean_absolute_error: 11.1513 - val_loss: 261.8080 - val_mean_absolute_error: 11.8153
Epoch 2/100
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 245.4754 - mean_absolute_error: 11.1324 - val_loss: 261.8649 - val_mean_absolute_error: 11.3877
Epoch 3/100
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 258.1833 - mean_absolute_error: 11.1644 - val_loss: 256.4775 - val_mean_absolute_error: 11.3045
Epoch 4/100
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 245.5562 - mean_absolute_error: 11.0460 - val_loss: 254.7063 - val_mean_absolute_error: 11.1123
Epoch 5/100
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 250.3231 - mean_absolute_error: 11.1135 - val_loss: 254.7024 - val_mean_absolute_error: 11.3584
Epoch 6/100
3901/3901 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step - loss: 248.9978 - mean_absolute_error: 11.0769 - val_loss: 255.6751 - val_mean_absolute_error: 11.0442
```

Out[106]: <keras.src.callbacks.history.History at 0x1c981d72950>

In [107]:
```
y_Pred_2= best_model.predict(X_test)
```

```
1220/1220 ━━━━━━━━━━━━━━━━━━━━ 1s 985us/step
```

In [108]:
```
r2Score_2 = r2_score(y_test, y_Pred_2)
```

In [109]:
```
r2Score_2
```

Out[109]: 0.2563154199905652

In [110]:
```
best_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 320)	4,800
batch_normalization (BatchNormalization)	(None, 320)	1,280
dense_1 (Dense)	(None, 100)	32,100
dense_2 (Dense)	(None, 100)	10,100
dense_3 (Dense)	(None, 32)	3,232
dense_4 (Dense)	(None, 32)	1,056
dense_5 (Dense)	(None, 1)	33

Total params: 156,525 (611.43 KB)

Trainable params: 51,961 (202.97 KB)

Non-trainable params: 640 (2.50 KB)

Optimizer params: 103,924 (405.96 KB)

In [111]:

```python
# MAPE Value
from sklearn.metrics import mean_absolute_percentage_error
mean_absolute_percentage_error(y_test, y_Pred_2)
```

Out[111]: 0.2410290383908038

- A MAPE of approximately 0.2618 means, on average, predictions have an error of approximately 26.18% relative to the actual values.

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

Problem Statement: The task is to build a neural network model to predict the delivery time of parcels using various features such as parcel weight, distance, delivery location, etc. The objective is to minimize the prediction error to ensure timely and efficient parcel delivery.

Applications and Modifications:

- **E-commerce:** Predicting delivery times for products to improve customer satisfaction.
- **Logistics:** Optimizing routes and schedules for delivery trucks.
- **Food Delivery Services:** Estimating delivery times for meals to ensure food arrives hot and fresh.
- **Courier Services:** Enhancing delivery speed predictions to manage customer expectations.
- **Supply Chain Management:** Predicting lead times to optimize inventory and reduce holding costs.

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

- `pd.to_datetime()` : Converts argument to datetime.
- `pd.date_range()` : Generates a fixed frequency DatetimeIndex.
- `pd.DatetimeIndex()` : Provides an index of datetime64 data.

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

- **datetime:** A module in Python for working with dates and times. It provides classes for manipulating dates and times in both simple and complex ways.
- **timedelta:** A class in the datetime module that represents the difference between two dates or times. It is used for calculating the difference and for adding or subtracting from a date or time.
- **time span (period):** Refers to a duration of time. In pandas, a Period represents a span of time defined by a frequency, such as a month, quarter, or year.

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

Outliers can significantly affect the performance of machine learning models. They can distort statistical measures, such as mean and standard deviation, and can lead to incorrect conclusions. By identifying and handling outliers, we can ensure that our model generalizes better and is not unduly influenced by anomalous data points.

5. Name 3 outlier removal methods?

- **IQR (Interquartile Range) Method:** Removes data points that fall below Q1 - 1.5*IQR or above Q3 + 1.5*IQR.
- **Z-Score Method:** Removes data points that are a specified number of standard deviations away from the mean.
- **Isolation Forest:** An unsupervised learning algorithm that isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

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

- **Linear Regression:** To predict delivery time based on input features.
- **Random Forest Regressor:** An ensemble method that uses multiple decision trees to improve predictive performance.
- **Support Vector Regression (SVR):** A regression technique that uses support vector machines to predict continuous values.

7. Why is scaling required for neural networks?

Scaling is required for neural networks to ensure that the input data is normalized, leading to faster convergence during training. It helps in making the gradient descent algorithm more efficient and stable. Scaling also prevents features with larger magnitudes from dominating the learning process.

8. Briefly explain your choice of optimizer.

Adam Optimizer: Adam (Adaptive Moment Estimation) is chosen because it combines the advantages of two other popular optimizers: AdaGrad and RMSProp. It maintains a per-parameter learning rate that improves performance on non-stationary problems and uses the first and second moments of the gradients to adapt the learning rate. This makes it suitable for a wide range of problems and is particularly effective for large datasets and high-dimensional parameter spaces.

9. Which activation function did you use and why?

ReLU and Tanh:

- **ReLU (Rectified Linear Unit):** Used because it helps in addressing the vanishing gradient problem, allowing the network to learn faster and perform better. It is computationally efficient and introduces non-linearity, which helps the network to learn complex patterns.
- **Tanh:** Used because it outputs values between -1 and 1, making it suitable for data that has been normalized. It also centers the data, which can lead to faster convergence.

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

In [ ]:

```
1
```