Porter - Neural Networks Regression

January 4, 2023

0.0.1 Problem Statement

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

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

Porter has a number of delivery partners available for delivering the food, from various restaurants and wants to get an estimated delivery time that it can provide the customers on the basis of what they are ordering, from where and also the delivery partners.

We need to train a Linear Regression model that will do the delivery time estimation, based on all those features

[1]:
```
# Loading all the necessary libraries

# for multidimensional array processing
import numpy as np

# for working with structured dataset
import pandas as pd
pd.options.display.float_format = "{:.2f}".format

# for basic plotting functionalities
import matplotlib.pyplot as plt

# for plotting advanced graphs
import seaborn as sns

# for Label Encoding
from sklearn.preprocessing import LabelEncoder

# for Target Encoding
from category_encoders import TargetEncoder

# to normalize data for getting best linear regression model
from sklearn.preprocessing import StandardScaler
```

```
# dividing data into training and testing data
from sklearn.model_selection import train_test_split

# for model metrics calculations
from sklearn.metrics import mean_absolute_percentage_error as mape
from sklearn.metrics import mean_absolute_error as mae
from sklearn.metrics import mean_squared_error as mse

# for Random Forest Regression
from sklearn.ensemble import RandomForestRegressor

# loading tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.metrics import RootMeanSquaredError,␣
 ↪MeanAbsolutePercentageError
from tensorflow.keras.layers import BatchNormalization, Dropout
from tensorflow.keras.losses import MeanSquaredLogarithmicError

# to suppress any warnings coming out
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
data = pd.read_csv("dataset.csv")
data.head()
```

[2]:
```
   market_id          created_at actual_delivery_time  \
0       1.00  2015-02-06 22:24:17  2015-02-06 23:27:16
1       2.00  2015-02-10 21:49:25  2015-02-10 22:56:29
2       3.00  2015-01-22 20:39:28  2015-01-22 21:09:09
3       3.00  2015-02-03 21:21:45  2015-02-03 22:13:00
4       3.00  2015-02-15 02:40:36  2015-02-15 03:20:26

                           store_id store_primary_category  order_protocol  \
0  df263d996281d984952c07998dc54358               american            1.00
1  f0ade77b43923b38237db569b016ba25                mexican            2.00
2  f0ade77b43923b38237db569b016ba25                    NaN            1.00
3  f0ade77b43923b38237db569b016ba25                    NaN            1.00
4  f0ade77b43923b38237db569b016ba25                    NaN            1.00

   total_items  subtotal  num_distinct_items  min_item_price  max_item_price  \
0            4      3441                   4             557            1239
1            1      1900                   1            1400            1400
2            1      1900                   1            1900            1900
3            6      6900                   5             600            1800
4            3      3900                   3            1100            1600
```

```
   total_onshift_partners  total_busy_partners  total_outstanding_orders
0                   33.00                14.00                     21.00
1                    1.00                 2.00                      2.00
2                    1.00                 0.00                      0.00
3                    1.00                 1.00                      2.00
4                    6.00                 6.00                      9.00
```

0.0.2 1. Exploratory Data Analysis & Analyzing Basic Metrics

[3]:
```python
data.shape
```

[3]:
```
(197428, 14)
```

On expecting the shape of the dataframe we can see that there are 197428 rows and 14 columns. Hence, we can say that we are working with a pretty good amount of data

Attribute Information of the Porter Data

- 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

[4]:
```python
data.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
```

On checking the information of data using info() method we can see that there are some null values present in the dataset

Also, columns have mixed data types like most of them are int64 and float 64 but few are object

Range of Columns

[5]:
```python
i = 1

for col in data.columns:
    if data[col].dtype == 'object':
        print(str(i) + '. "' + col + '" is categorical column with total unique␣
 ↪values as: ' + str(len(data[col].unique())))
    else:
        print(str(i) + '. "' + col + '" is continuous column with min, max and␣
 ↪mean values as: ', data[col].min(), data[col].max(), data[col].mean())
    i += 1
```

```
1. "market_id" is continuous column with min, max and mean values as:  1.0 6.0
2.978706074597462
2. "created_at" is categorical column with total unique values as: 180985
3. "actual_delivery_time" is categorical column with total unique values as:
178111
4. "store_id" is categorical column with total unique values as: 6743
5. "store_primary_category" is categorical column with total unique values as:
75
6. "order_protocol" is continuous column with min, max and mean values as:  1.0
7.0 2.8823517433425137
7. "total_items" is continuous column with min, max and mean values as:  1 411
3.196390582896043
8. "subtotal" is continuous column with min, max and mean values as:  0 27100
2682.331401827502
9. "num_distinct_items" is continuous column with min, max and mean values as:
1 20 2.6707913771096297
10. "min_item_price" is continuous column with min, max and mean values as:  -86
14700 686.2184695180015
11. "max_item_price" is continuous column with min, max and mean values as:  0
14700 1159.5886297789573
12. "total_onshift_partners" is continuous column with min, max and mean values
as:  -4.0 171.0 44.808093130057514
13. "total_busy_partners" is continuous column with min, max and mean values as:
```

```
-5.0 154.0 41.739746972389966
14. "total_outstanding_orders" is continuous column with min, max and mean
values as:  -6.0 285.0 58.0500645816544
```

[6]:
```
data.describe().T
```

[6]:

	count	mean	std	min	25%	50%
market_id	196441.00	2.98	1.52	1.00	2.00	3.00
order_protocol	196433.00	2.88	1.50	1.00	1.00	3.00
total_items	197428.00	3.20	2.67	1.00	2.00	3.00
subtotal	197428.00	2682.33	1823.09	0.00	1400.00	2200.00
num_distinct_items	197428.00	2.67	1.63	1.00	1.00	2.00
min_item_price	197428.00	686.22	522.04	-86.00	299.00	595.00
max_item_price	197428.00	1159.59	558.41	0.00	800.00	1095.00
total_onshift_partners	181166.00	44.81	34.53	-4.00	17.00	37.00
total_busy_partners	181166.00	41.74	32.15	-5.00	15.00	34.00
total_outstanding_orders	181166.00	58.05	52.66	-6.00	17.00	41.00

	75%	max
market_id	4.00	6.00
order_protocol	4.00	7.00
total_items	4.00	411.00
subtotal	3395.00	27100.00
num_distinct_items	3.00	20.00
min_item_price	949.00	14700.00
max_item_price	1395.00	14700.00
total_onshift_partners	65.00	171.00
total_busy_partners	62.00	154.00
total_outstanding_orders	85.00	285.00

From the describe function we can see that ranges of all columns are different like subtotal, min_item_price and max_item_price are in hundreds while rest are less.

Hence, before model building we need to scale the data before applying any Machine Learning algorithm.

Also, for subtotal the mean and median are way different and also it has max value as 27100 which is way higher so we need to see for outliers and handle skewness.

Simple Feature Engineering

Since, we don't have delievery time but we can get it by subtracting created_at from actual_delievery_time. The time difference of two datetimes will give us the time taken for delievery and hence this problem can be converted to regression problem.

Also we can find hour of day from the order time and also the day of the week to see when orders are mostly placed.

[7]:
```
# for getting hour when order is created
data['hour'] = pd.to_datetime(data['created_at']).dt.hour
```

```
# for getting day name when order is created
data['dayname'] = pd.to_datetime(data['created_at']).dt.day_name()
```

[8]:
```
data['delievery_time'] = (pd.to_datetime(data['actual_delivery_time']) - pd.
 ↪to_datetime(data['created_at']))
data['delievery_time'] = data['delievery_time']/ pd.Timedelta(minutes = 1)
```

For finding delievery_time we used columns actual_delivery_time and created_at and found their difference

Next we can use that data and can convert into days / hours / minutes / seconds but I am picking minutes as most delievery times are shown in minutes so for that sake I am going with minutes

[9]:
```
data.head(2)
```

[9]:
```
   market_id           created_at actual_delivery_time  \
0       1.00  2015-02-06 22:24:17  2015-02-06 23:27:16
1       2.00  2015-02-10 21:49:25  2015-02-10 22:56:29

                           store_id store_primary_category  order_protocol  \
0  df263d996281d984952c07998dc54358               american            1.00
1  f0ade77b43923b38237db569b016ba25                mexican            2.00

   total_items  subtotal  num_distinct_items  min_item_price  max_item_price  \
0            4      3441                   4             557            1239
1            1      1900                   1            1400            1400

   total_onshift_partners  total_busy_partners  total_outstanding_orders  \
0                   33.00                14.00                     21.00
1                    1.00                 2.00                      2.00

   hour  dayname  delievery_time
0    22   Friday           62.98
1    21  Tuesday           67.07
```

[10]:
```
data.describe().T
```

[10]:
```
                            count    mean     std    min     25%     50%  \
market_id               196441.00    2.98    1.52   1.00    2.00    3.00
order_protocol          196433.00    2.88    1.50   1.00    1.00    3.00
total_items             197428.00    3.20    2.67   1.00    2.00    3.00
subtotal                197428.00 2682.33 1823.09   0.00 1400.00 2200.00
num_distinct_items      197428.00    2.67    1.63   1.00    1.00    2.00
min_item_price          197428.00  686.22  522.04 -86.00  299.00  595.00
max_item_price          197428.00 1159.59  558.41   0.00  800.00 1095.00
total_onshift_partners  181166.00   44.81   34.53  -4.00   17.00   37.00
total_busy_partners     181166.00   41.74   32.15  -5.00   15.00   34.00
```

```
total_outstanding_orders 181166.00    58.05    52.66  -6.00    17.00    41.00
hour                     197428.00     8.47     8.66   0.00     2.00     3.00
delievery_time           197421.00    48.47   320.49   1.68    35.07    44.33

                              75%       max
market_id                    4.00      6.00
order_protocol               4.00      7.00
total_items                  4.00    411.00
subtotal                  3395.00  27100.00
num_distinct_items           3.00     20.00
min_item_price             949.00  14700.00
max_item_price            1395.00  14700.00
total_onshift_partners      65.00    171.00
total_busy_partners         62.00    154.00
total_outstanding_orders    85.00    285.00
hour                        19.00     23.00
delievery_time              56.35 141947.65
```

On checking the describe function we see that delievery_time has max value as 141947.65 which is an outlier and we will remove it in later steps.

[11]:
```
data.isna().sum() * 100 / data.shape[0]
```

[11]:
```
market_id                   0.50
created_at                  0.00
actual_delivery_time        0.00
store_id                    0.00
store_primary_category      2.41
order_protocol              0.50
total_items                 0.00
subtotal                    0.00
num_distinct_items          0.00
min_item_price              0.00
max_item_price              0.00
total_onshift_partners      8.24
total_busy_partners         8.24
total_outstanding_orders    8.24
hour                        0.00
dayname                     0.00
delievery_time              0.00
dtype: float64
```

[12]:
```
data[data['total_onshift_partners'].isna()].head(5)[['total_onshift_partners',
                                                      'total_busy_partners',
                                                     ␣
  ↪'total_outstanding_orders']]
```

[12]:
	total_onshift_partners	total_busy_partners	total_outstanding_orders
160	NaN	NaN	NaN
161	NaN	NaN	NaN
162	NaN	NaN	NaN
163	NaN	NaN	NaN
164	NaN	NaN	NaN

On inspecting the Nan values we see that columns - market_id - store_primary_category - order_protocol - total_onshift_partners - total_busy_partners - total_outstanding_orders

But we cannot fill store_primary_category as we cannot fill it so it is better to drop it as it is 2.41% of complete data. One way is to use market_id and store_id to fill those null values and in this way we won't loose data.

Also, same goes with market_id as we cannot fill them so we are dropping it otherwise by using store_primary_category and store_id we can fill it.

[13]:
```
# dropping the values in column "store_primary_category"
data = data[~data['store_primary_category'].isna()]

# dropping the values in column "market_id"
data = data[~data['market_id'].isna()]

# dropping the values in actual_delivery_time
data = data[~data['actual_delivery_time'].isna()]
```

For filling null values of column order_protocol we can use mode as it is sort of categorical column hence mode is best way to fill it.

[14]:
```
data['order_protocol'].fillna(data['order_protocol'].mode().iloc[0], inplace =␣
↪True)
```

Now we are left with - total_onshift_partners - total_busy_partners - total_outstanding_orders

below columns and we can use median approaches to fill these null values as these are continuous and we don't want to loose 8% values

[15]:
```
data['total_onshift_partners'].fillna(data['total_onshift_partners'].median(),␣
↪inplace = True)
data['total_busy_partners'].fillna(data['total_busy_partners'].median(),␣
↪inplace = True)
data['total_outstanding_orders'].fillna(data['total_outstanding_orders'].
↪median(), inplace = True)
```

[16]:
```
data.isna().sum() * 100 / data.shape[0]
```

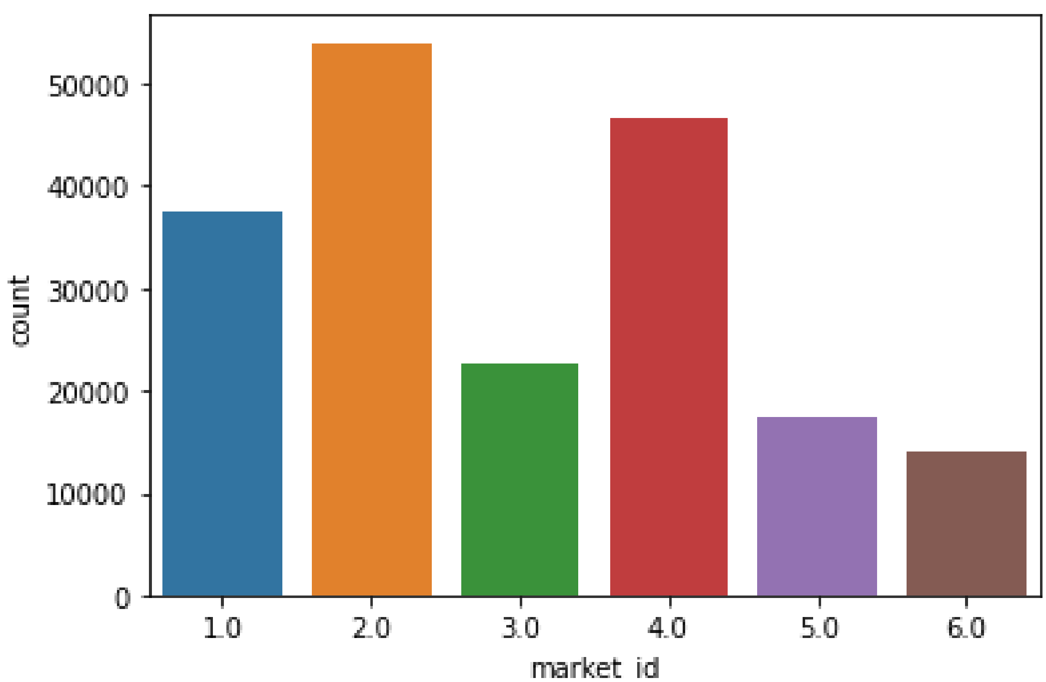
[16]:
```
market_id                  0.00
created_at                 0.00
actual_delivery_time       0.00
```

```
store_id                    0.00
store_primary_category      0.00
order_protocol              0.00
total_items                 0.00
subtotal                    0.00
num_distinct_items          0.00
min_item_price              0.00
max_item_price              0.00
total_onshift_partners      0.00
total_busy_partners         0.00
total_outstanding_orders    0.00
hour                        0.00
dayname                     0.00
delievery_time              0.00
dtype: float64
```

Now, we can see that there are no null values present in the data. Hence, we are good to go for further analysis.

0.0.3 2. Univariate Analysis

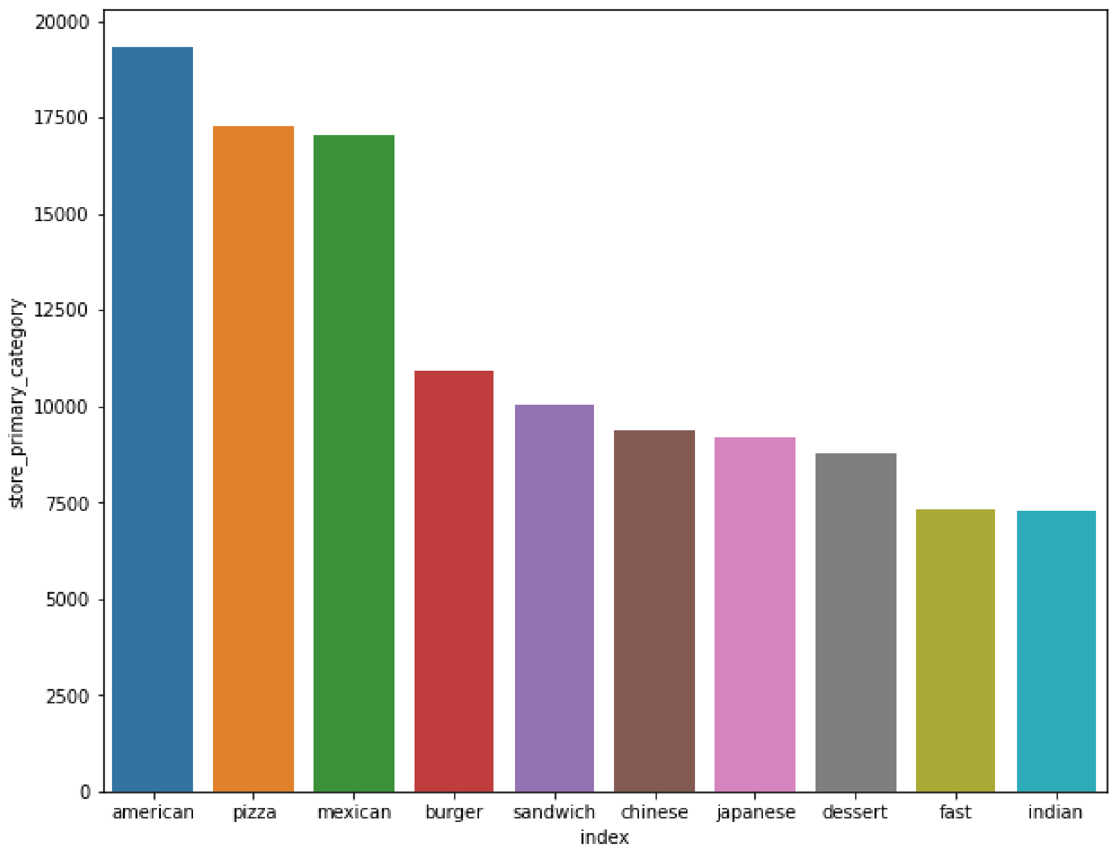
[17]:
```
sns.countplot(data['market_id'])
plt.show()
```



Observation

We can see that from above countplot mostly orders are placed from market_id 2 and least are from 6

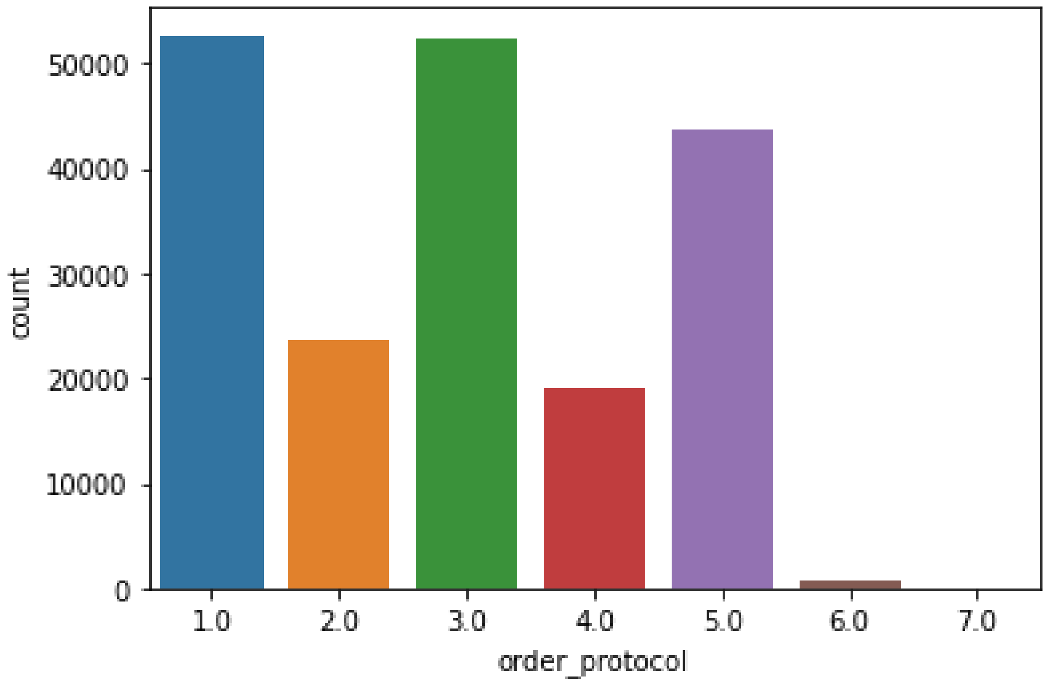
[18]:
```
top10values = pd.DataFrame(data['store_primary_category'].value_counts().
  ↪head(10)).reset_index()

fig = plt.figure(figsize = (10, 8))
sns.barplot(x = 'index', y = 'store_primary_category', data = top10values)
plt.show()
```



Observation

On plotting barplot we can see that mostly orders are placed by american store_primary_category

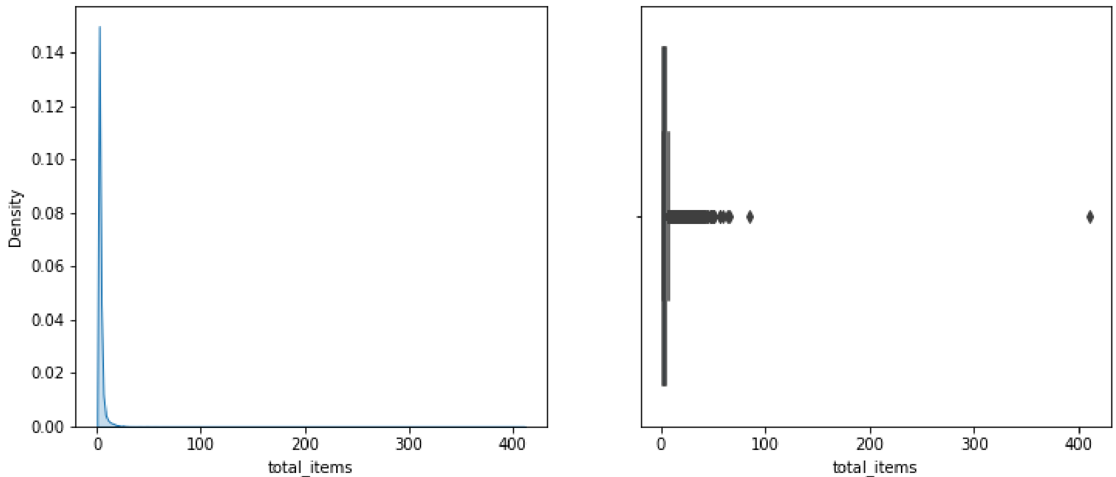
[19]:
```
sns.countplot(data['order_protocol'])
plt.show()
```



Observation

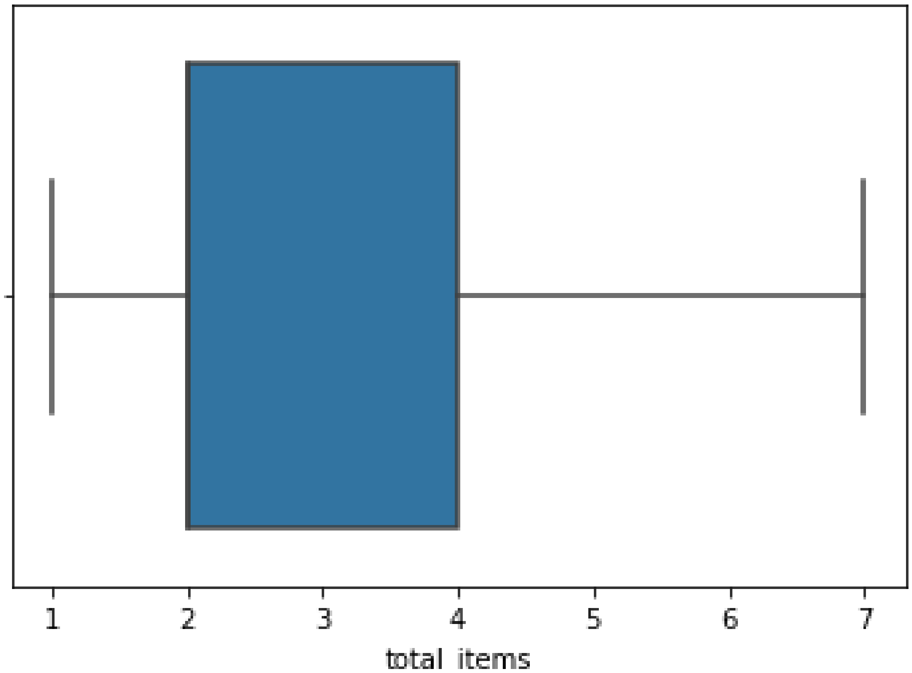
We can see that from above countplot mostly order_protocol are from id 1 and 3 and least are from 7

[20]:
```
fig = plt.figure(figsize = (12, 5))

plt.subplot(1, 2, 1)
sns.kdeplot(data['total_items'], fill = True)

plt.subplot(1, 2, 2)
sns.boxplot(data['total_items'])
plt.show()
```



On inspecting the total_items column we can see that there are outliers present in it. Also, when plotting the kdeplot we can see that it is not exactly Normal distribution. Now, we will apply IQR method to remove outliers.

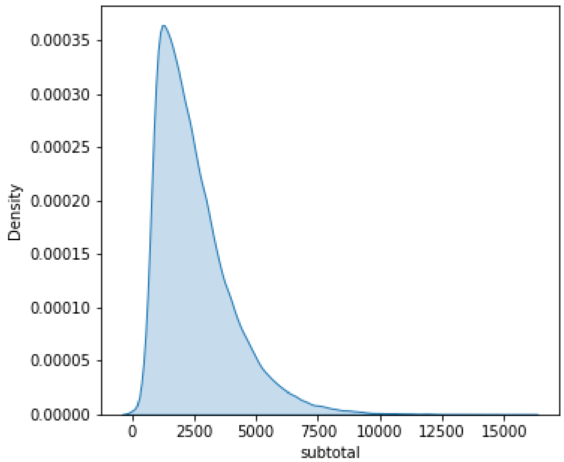
[21]:
```
q1 = np.quantile(data['total_items'], 0.25)
q3 = np.quantile(data['total_items'], 0.75)
IQR = q3 - q1
upper_bound = q3 + 1.5 * IQR
lower_bound = q1 - 1.5 * IQR

data = data[(data['total_items'] >= lower_bound) & (data['total_items'] <=␣
 ↪upper_bound)]

sns.boxplot(data['total_items'])
plt.show()
```

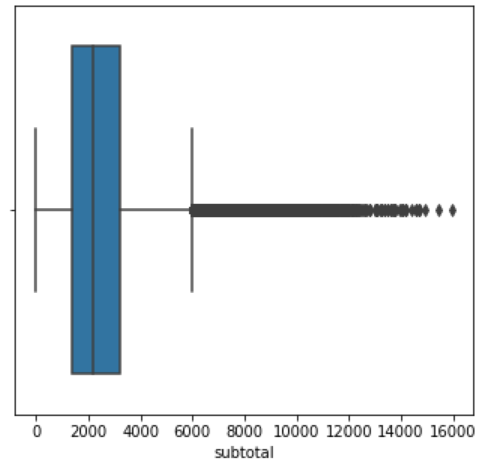


Observation

We can see that post removing outliers from total_items we are getting good data and hence we are good to go for further analysis

[22]:
```
fig = plt.figure(figsize = (12, 5))

plt.subplot(1, 2, 1)
sns.kdeplot(data['subtotal'], fill = True)

plt.subplot(1, 2, 2)
sns.boxplot(data['subtotal'])
plt.show()
```



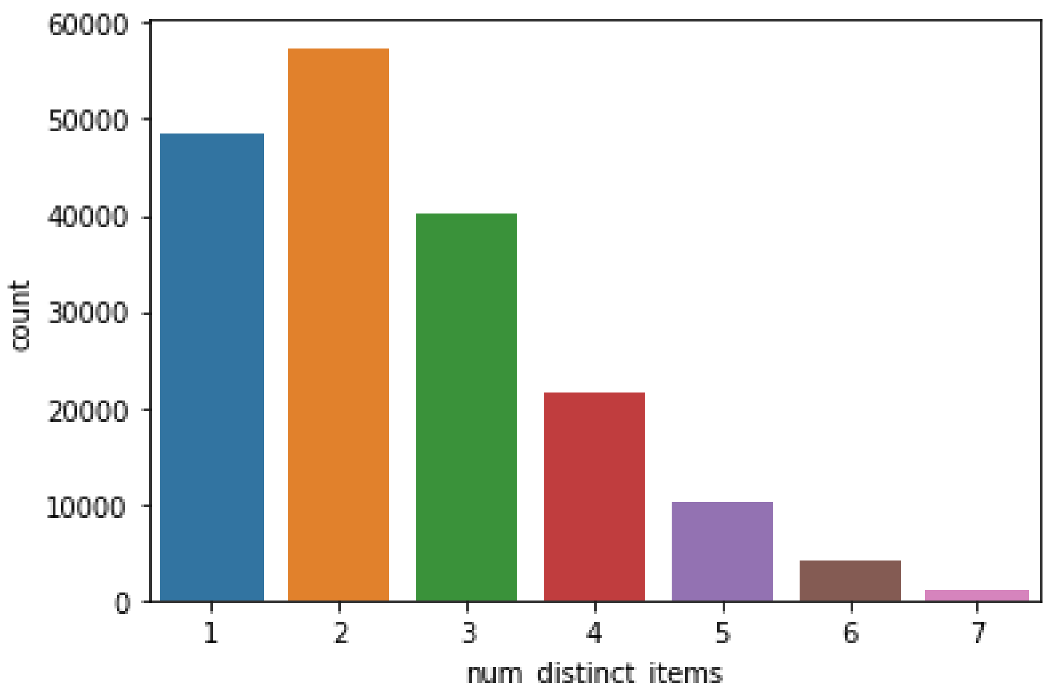


On inspecting the subtotal column we can see that there are outliers present in it. Also, when plotting the kdeplot we can see that it is not exactly Normal distribution.

Observation

But it shouldn't be removed as this is the amount paid by user and some users must have paid more amount so they must be kept as algorithm should also predict large amount orders. Hence, we will use it as it is.

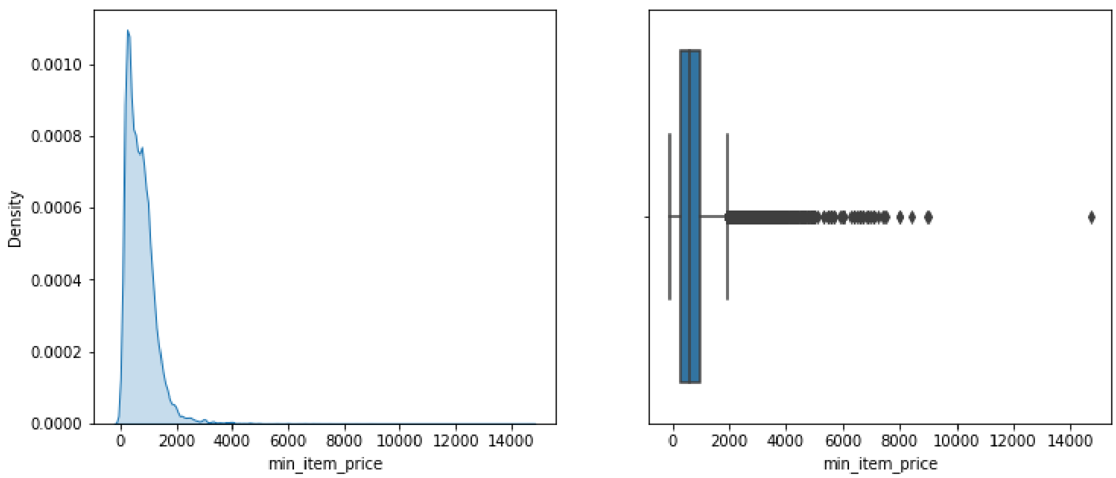
[23]:
```
sns.countplot(data['num_distinct_items'])
plt.show()
```



Observation

On plotting countplot we can see that mostly orders are placed with 2 items and then single item and very few orders are there which have 7 order

[24]:
```
fig = plt.figure(figsize = (12, 5))

plt.subplot(1, 2, 1)
sns.kdeplot(data['min_item_price'], fill = True)

plt.subplot(1, 2, 2)
sns.boxplot(data['min_item_price'])
plt.show()
```

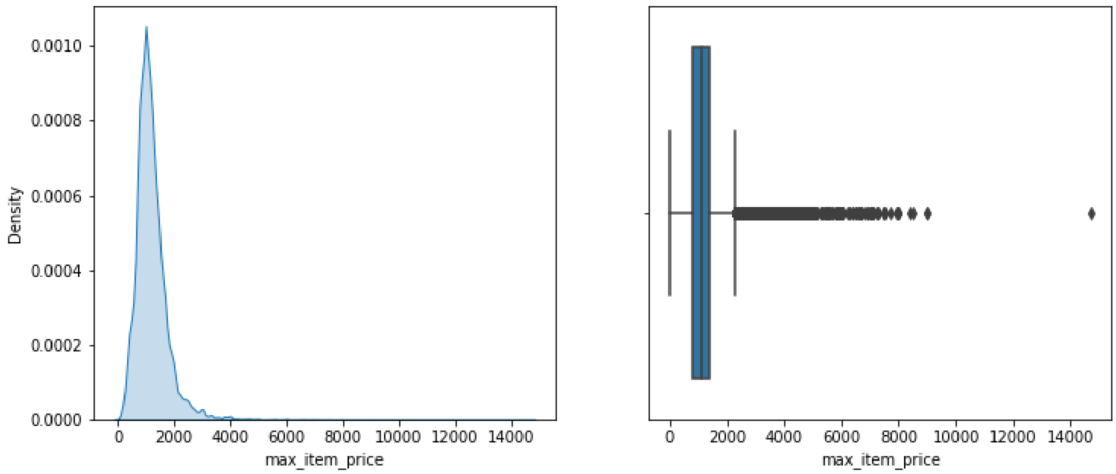


On inspecting the min_item_price column we can see that there are outliers present in it. Also, when plotting the kdeplot we can see that it is not exactly Normal distribution.

Observation

But it shouldn't be removed as this is the minimum item price for a booking made by user and model should handle these cases. Hence, we will use it as it is.

[25]:
```
fig = plt.figure(figsize = (12, 5))

plt.subplot(1, 2, 1)
sns.kdeplot(data['max_item_price'], fill = True)

plt.subplot(1, 2, 2)
sns.boxplot(data['max_item_price'])
plt.show()
```

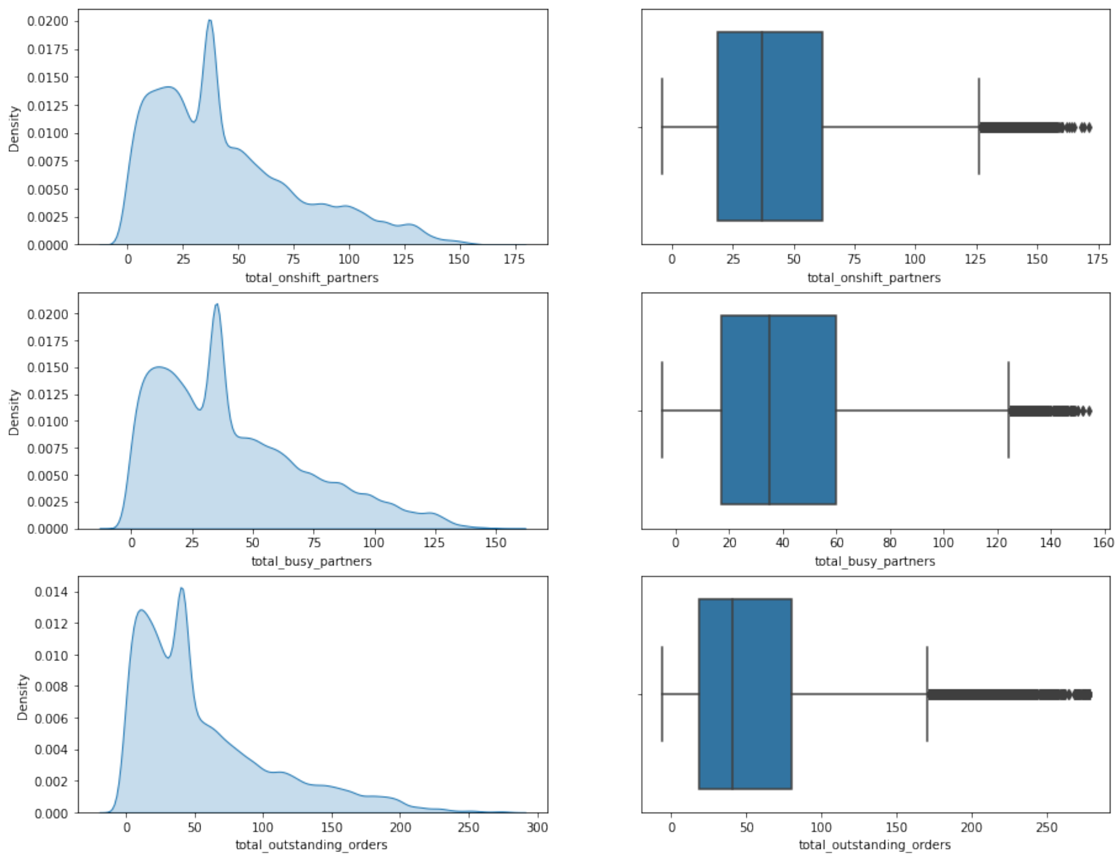


On inspecting the max_item_price column we can see that there are outliers present in it. Also, when plotting the kdeplot we can see that it is not exactly Normal distribution.

Observation

But it shouldn't be removed as this is the maximum item price for a booking made by user and model should handle these cases. Hence, we will use it as it is.

[26]:
```
fig = plt.figure(figsize = (15, 12))

plt.subplot(3, 2, 1)
sns.kdeplot(data['total_onshift_partners'], fill = True)

plt.subplot(3, 2, 2)
sns.boxplot(data['total_onshift_partners'])

plt.subplot(3, 2, 3)
sns.kdeplot(data['total_busy_partners'], fill = True)

plt.subplot(3, 2, 4)
sns.boxplot(data['total_busy_partners'])

plt.subplot(3, 2, 5)
sns.kdeplot(data['total_outstanding_orders'], fill = True)

plt.subplot(3, 2, 6)
sns.boxplot(data['total_outstanding_orders'])
plt.show()
```

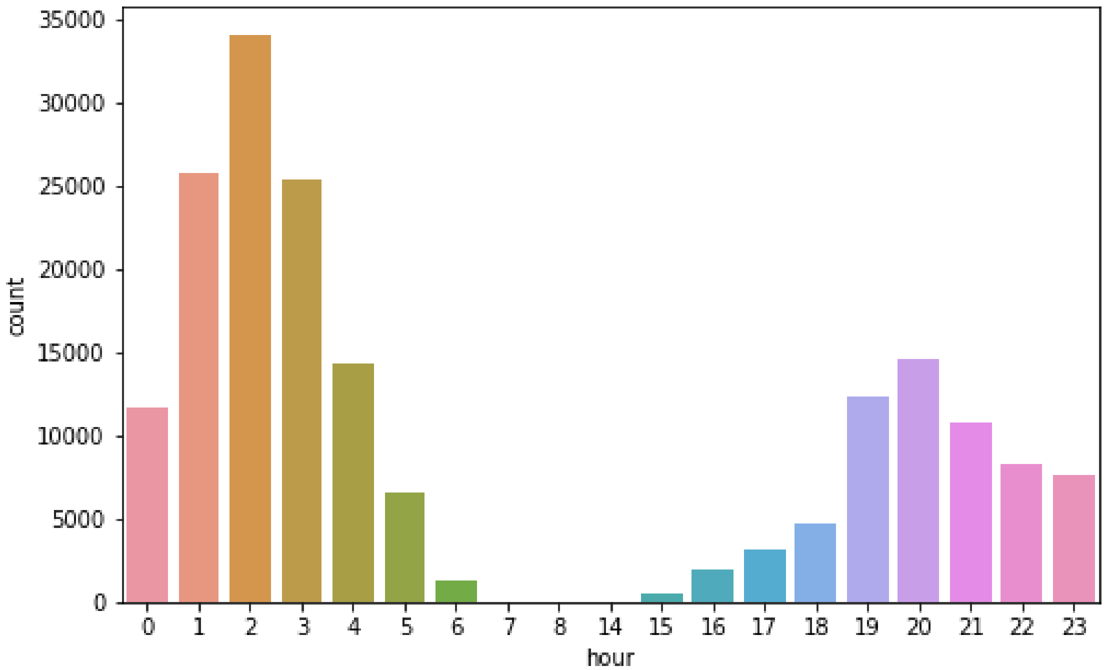


On inspecting the total_onshift_partners, total_busy_partners and total_outstanding_orders column we can see that there are outliers present in it. Also, when plotting the kdeplot we can see that it is not exactly Normal distribution.

Observation

My hypothesis is that it shouldn't be removed as there must be stores present which have high partners and high staff means that most people would be busy and it shouldn't be removed as we need to make a model which can work in all scenarios and model should be robust enough to handle this so I am ignoring them.

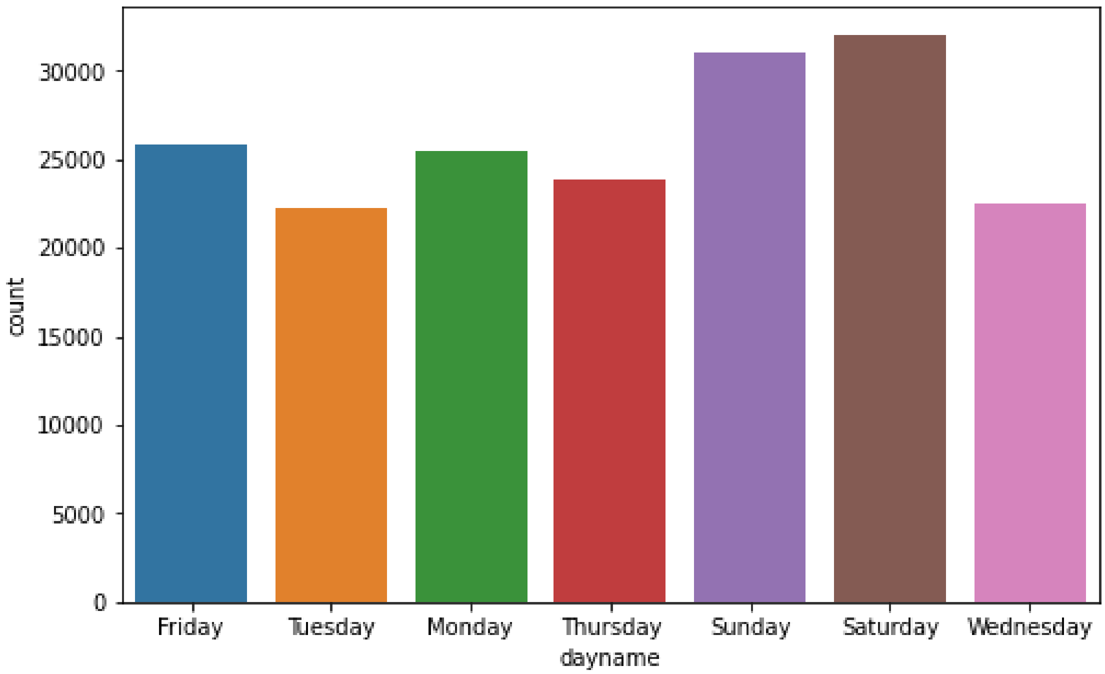
[27]:
```
fig = plt.figure(figsize = (8, 5))

sns.countplot(data['hour'])
plt.show()
```



Observation

On plotting countplot we can see that mostly orders are placed in night which is around 2 AM as mostly orders are from america so in IST time it should be around 8 - 9 AM as per my thinking

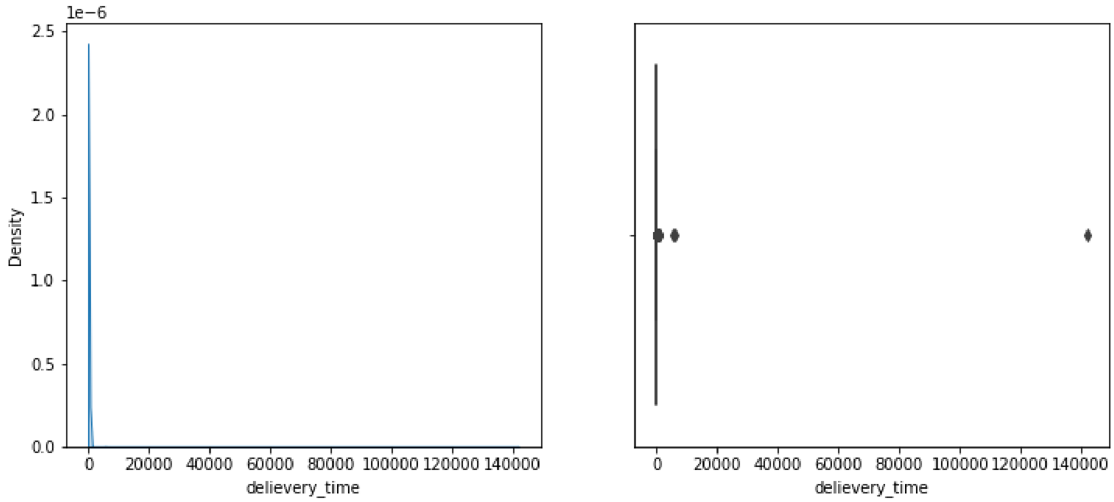
[28]:
```
fig = plt.figure(figsize = (8, 5))

sns.countplot(data['dayname'])
plt.show()
```



Observation

On plotting countplot we can see that mostly orders are placed mostly on weekends i.e. Saturday and Sunday because people get free so they might place orders here.

[29]:
```
fig = plt.figure(figsize = (12, 5))

plt.subplot(1, 2, 1)
sns.kdeplot(data['delievery_time'], fill = True)

plt.subplot(1, 2, 2)
sns.boxplot(data['delievery_time'])
plt.show()
```



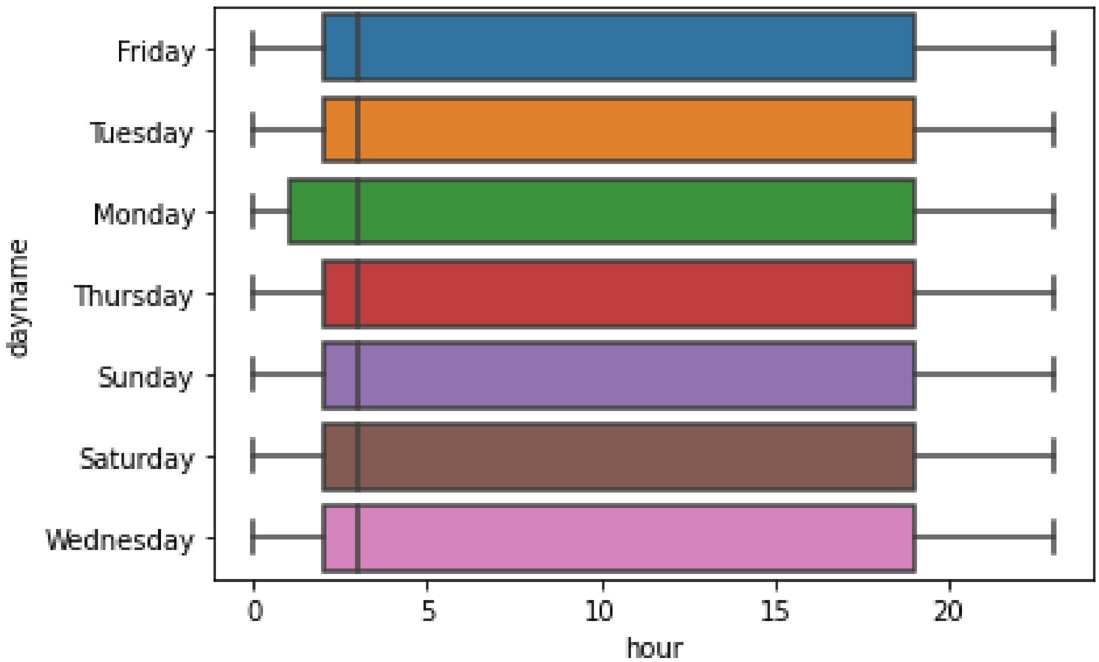
We see some values are very extreme so we will remove them so that model can learn more robustly.

Hence, on close inspection I noticed that if value is more than 5000 then it is an outlier so I am choosing values which are less than 5000

[30]:
```
data = data[data['delievery_time'] < 5000]
```

0.0.4 3. Bivariate Analysis

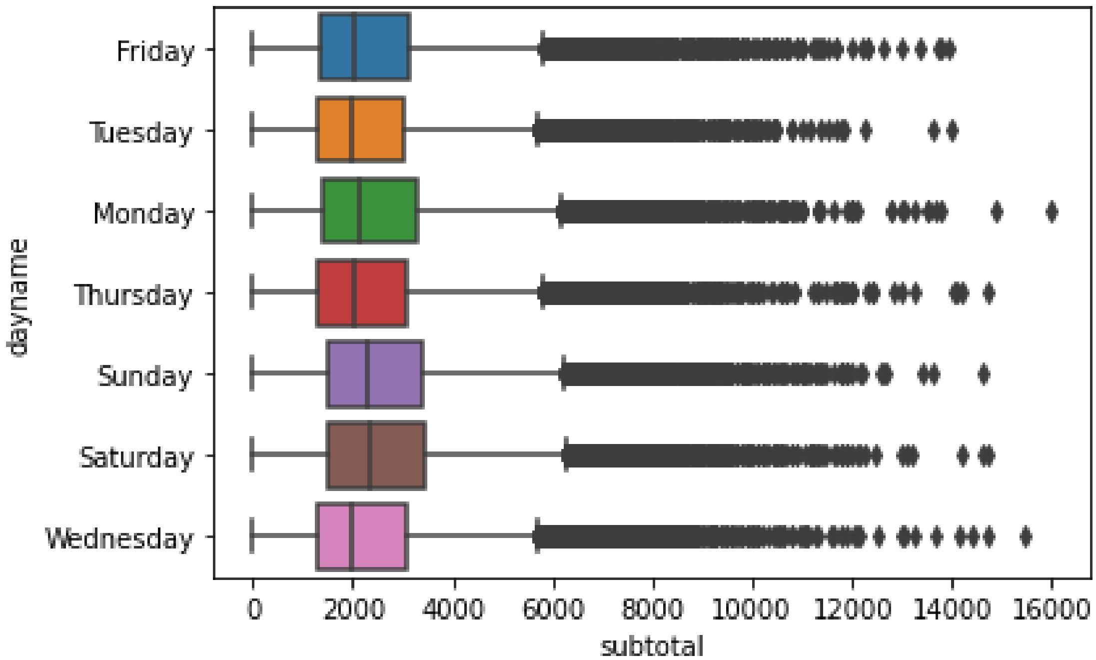
[31]:
```
sns.boxplot(x = data['hour'],
            y = data['dayname']
           )

plt.show()
```



Observation

We can see that on all days mostly orders are placed at 2 AM and there is no single case where orders were placed before or after that duration

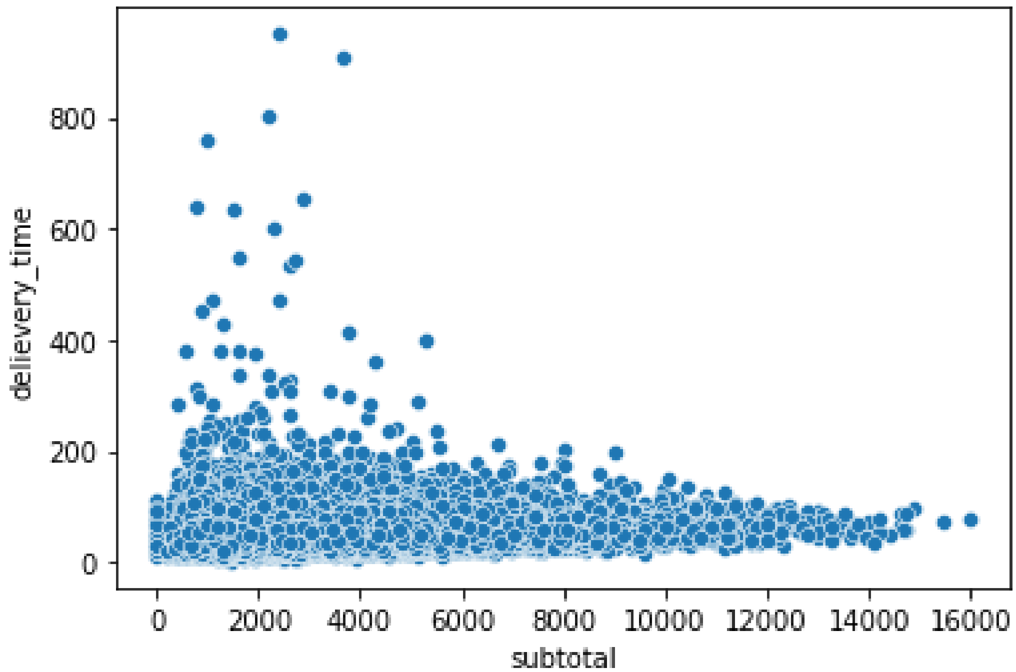
[32]:
```
sns.boxplot(x = data['subtotal'],
            y = data['dayname']
           )

plt.show()
```



Observation

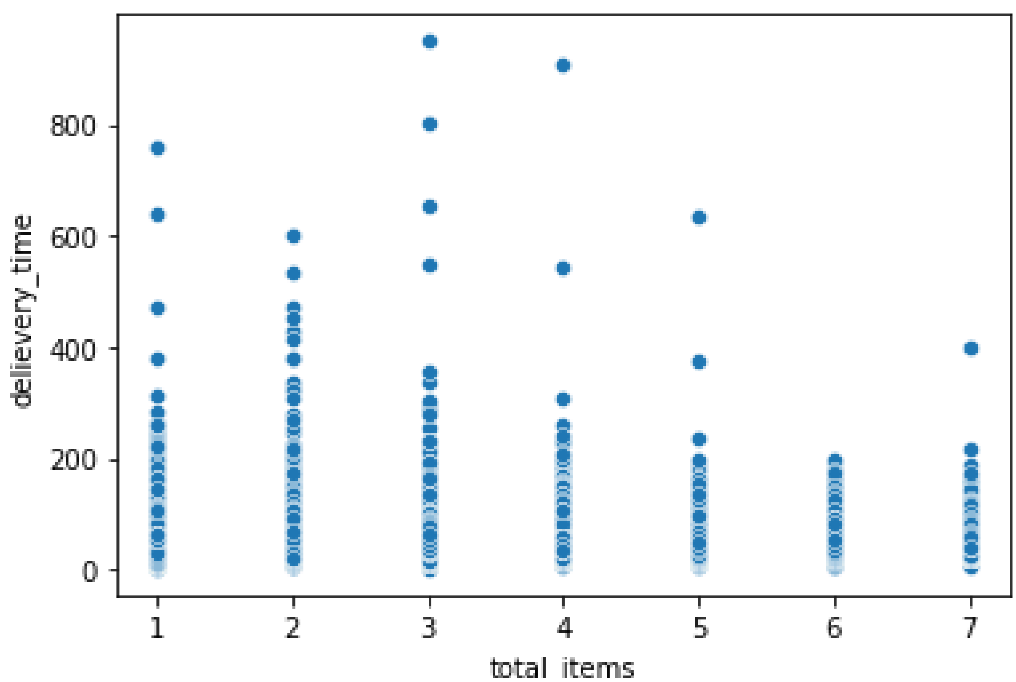
We can see that on all days mostly orders placed on Saturday's and Sunday's are little higher than other days as people get week off and they get time to do shiftings and they place little high orders.

[33]:
```
sns.scatterplot(data['subtotal'],
                data['delievery_time'])
plt.show()
```



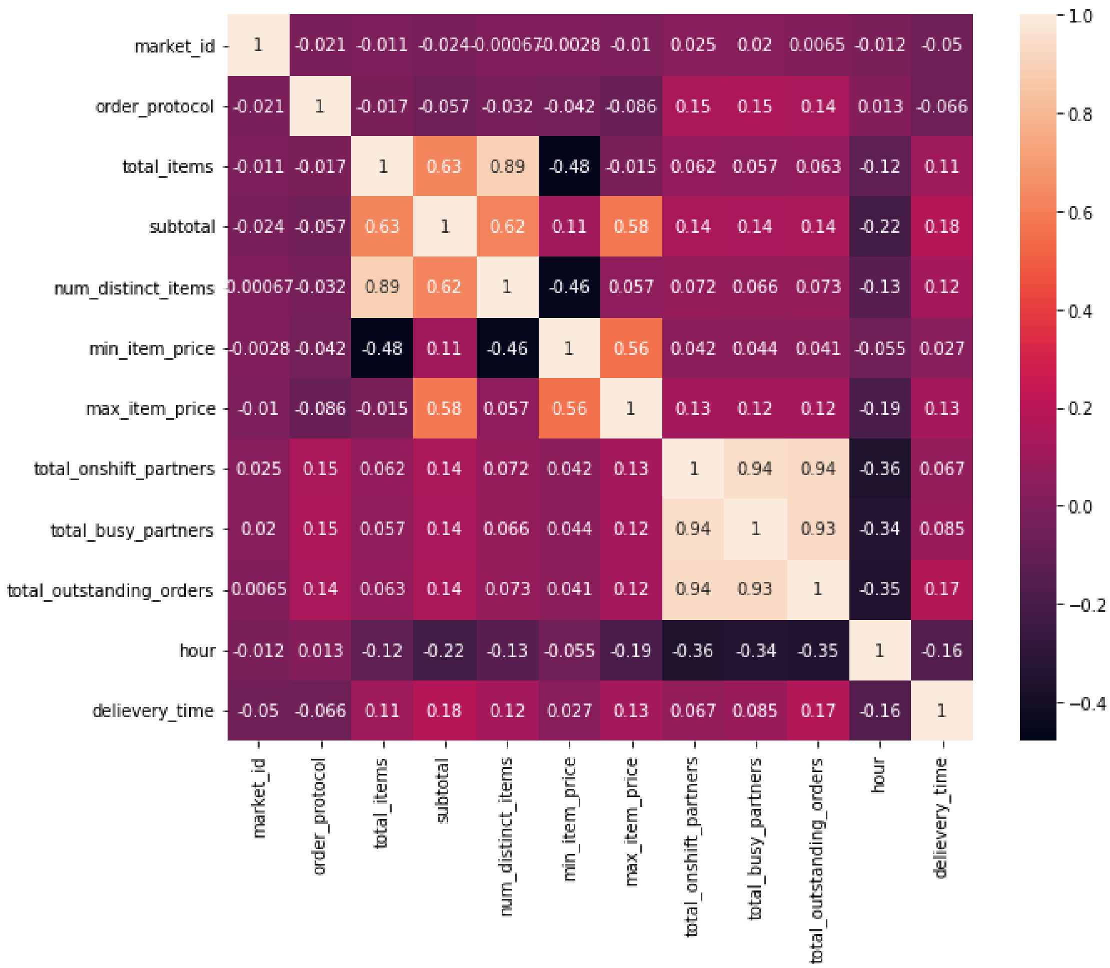
We can see that scatterplot doesn't reveal any good relation with them.

[34]:
```
sns.scatterplot(data['total_items'],
                data['delievery_time'])
plt.show()
```



We can see that scatterplot doesn't reveal any good relation with them.

[35]:
```
fig = plt.figure(figsize = (10, 8))

sns.heatmap(data.corr(), annot = True)
plt.show()
```



Observation

We can see from heatmap that for column delievery_time there is no column which has good correlation with it.

Hence, plotting pairplot won't be of much use hence we will use complete data to train our machine learning model.

0.0.5 4. Encoding Categorical Columns

[36]:
```python
data.head(2)
```

[36]:
```
   market_id           created_at actual_delivery_time  \
0       1.00  2015-02-06 22:24:17  2015-02-06 23:27:16
1       2.00  2015-02-10 21:49:25  2015-02-10 22:56:29

                           store_id store_primary_category  order_protocol  \
0  df263d996281d984952c07998dc54358               american            1.00
1  f0ade77b43923b38237db569b016ba25                mexican            2.00

   total_items  subtotal  num_distinct_items  min_item_price  max_item_price  \
0            4      3441                   4             557            1239
1            1      1900                   1            1400            1400

   total_onshift_partners  total_busy_partners  total_outstanding_orders  \
0                   33.00                14.00                     21.00
1                    1.00                 2.00                      2.00

   hour  dayname  delievery_time
0    22   Friday           62.98
1    21  Tuesday           67.07
```

- For column store_primary_category there are so many categories hence, we need target encoding
- For column dayname we can use label encoding

[37]:
```python
# target encoding the store_primary_category column
targetEncoder = TargetEncoder()

data['store_primary_category_encoded'] = targetEncoder.
 ↪fit_transform(data['store_primary_category'], data['delievery_time'])
```

[38]:
```python
# label encoding the dayname column
daynameLabelEncoder = LabelEncoder()

data['dayname_encoded'] = daynameLabelEncoder.fit_transform(data['dayname'])
```

[39]:
```python
data.head(2)
```

[39]:
```
   market_id           created_at actual_delivery_time  \
0       1.00  2015-02-06 22:24:17  2015-02-06 23:27:16
1       2.00  2015-02-10 21:49:25  2015-02-10 22:56:29

                           store_id store_primary_category  order_protocol  \
0  df263d996281d984952c07998dc54358               american            1.00
1  f0ade77b43923b38237db569b016ba25                mexican            2.00
```

```
   total_items  subtotal  num_distinct_items  min_item_price  max_item_price  \
0            4      3441                   4             557            1239
1            1      1900                   1            1400            1400

   total_onshift_partners  total_busy_partners  total_outstanding_orders  \
0                   33.00                14.00                     21.00
1                    1.00                 2.00                      2.00

   hour  dayname  delievery_time  store_primary_category_encoded  \
0    22   Friday           62.98                           47.52
1    21  Tuesday           67.07                           44.16

   dayname_encoded
0                0
1                5
```

0.0.6 5. Preparing Datset for Machine Learning

We now have to select independent variables as delievery_time is the dependent variable.

1. market_id
2. store_primary_category_encoded
3. order_protocol
4. total_items
5. subtotal
6. num_distinct_items
7. min_item_price
8. max_item_price
9. total_onshift_partners
10. total_busy_partners
11. total_outstanding_orders
12. hour
13. dayname_encoded

[40]:
```
X = data[['market_id', 'store_primary_category_encoded', 'order_protocol',
          'total_items', 'subtotal', 'num_distinct_items', 'min_item_price',
          'max_item_price', 'total_onshift_partners', 'total_busy_partners',
          'total_outstanding_orders', 'hour', 'dayname_encoded']]

y = data['delievery_time']
```

Now we have decided the independent variables so let us divide the data into testing and training.

Ideally, I would prefer to divide data first and then standard scale the training data and use the the same scaling model for test data

[41]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,
    random_state = 19)

print("Shape of Training features: ", X_train.shape)
print("Shape of Testing features: ", X_test.shape)
print("Shape of Training labels: ", y_train.shape)
print("Shape of Testing labels: ", y_test.shape)
```

```
Shape of Training features:  (146284, 13)
Shape of Testing features:  (36572, 13)
Shape of Training labels:  (146284,)
Shape of Testing labels:  (36572,)
```

Now we will fit and transform training data using standard scaler and then transform test data using that only.

[42]:
```python
standardScaler = StandardScaler()
standardScaler.fit(X_train)
```

[42]:
```
StandardScaler()
```

[43]:
```python
X_train_scaled = standardScaler.transform(X_train)
X_test_scaled = standardScaler.transform(X_test)
```

0.0.7 6. Random Forest Regressor

Before, making Neural networks let us create baseline model so that comparison can be made easily.

First, we will calculate a function modelAccuracy to calculate accuracy which will calculcate the performance metrics which is

- Mean absolute error
- Root Mean squared error
- Mean Absolute percentage error

[44]:
```python
def modelAccuracy(actual, predicted):
    print('MAE:', round(mae(actual, predicted), 2))
    print('RMSE:', round(np.sqrt(mse(actual, predicted)), 2))
    print('MAPE:', round(mape(actual, predicted), 2))
```

[45]:
```python
regression = RandomForestRegressor(max_depth = 15, n_estimators = 100,
    random_state = 0)
regression.fit(X_train_scaled, y_train)

print(regression.score(X_train_scaled, y_train))
```

```
0.5519362548291198
```

[46]:
```python
modelAccuracy(y_test, regression.predict(X_test_scaled))
```

```
MAE: 11.54
RMSE: 17.61
MAPE: 0.27
```

We can see that Random Forest is not able to perform good and it gave only 0.55 score and Mean Absolute Error is 11.54 and Root mean square is 17.61 and MAPE is around 0.27.

Hence, we need to make something better out of this.

0.0.8 7. Training Neural Networks

[47]:
```
model = Sequential([
    Dense(16, activation = "relu", input_shape = (X_train_scaled.shape[1],)),
    Dense(32, activation = "relu"),
    Dense(64, activation = "relu"),
    Dense(128, activation = "relu"),
    Dense(1, activation = "linear")
        ])
```

```
2023-01-04 01:51:07.877995: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not
creating XLA devices, tf_xla_enable_xla_devices not set
2023-01-04 01:51:07.878271: I tensorflow/core/platform/cpu_feature_guard.cc:142]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

We have created a baseline model where I have created a model with 4 hidden layers - 16 nodes in first layer - 32 nodes in second layer - 64 nodes in third layer - 128 nodes in third layer

[48]:
```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                224
_________________________________________________________________
dense_1 (Dense)              (None, 32)                544
_________________________________________________________________
dense_2 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_3 (Dense)              (None, 128)               8320
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 129
=================================================================
Total params: 11,329
Trainable params: 11,329
```

```
Non-trainable params: 0
_________________________________________________________________
None
```

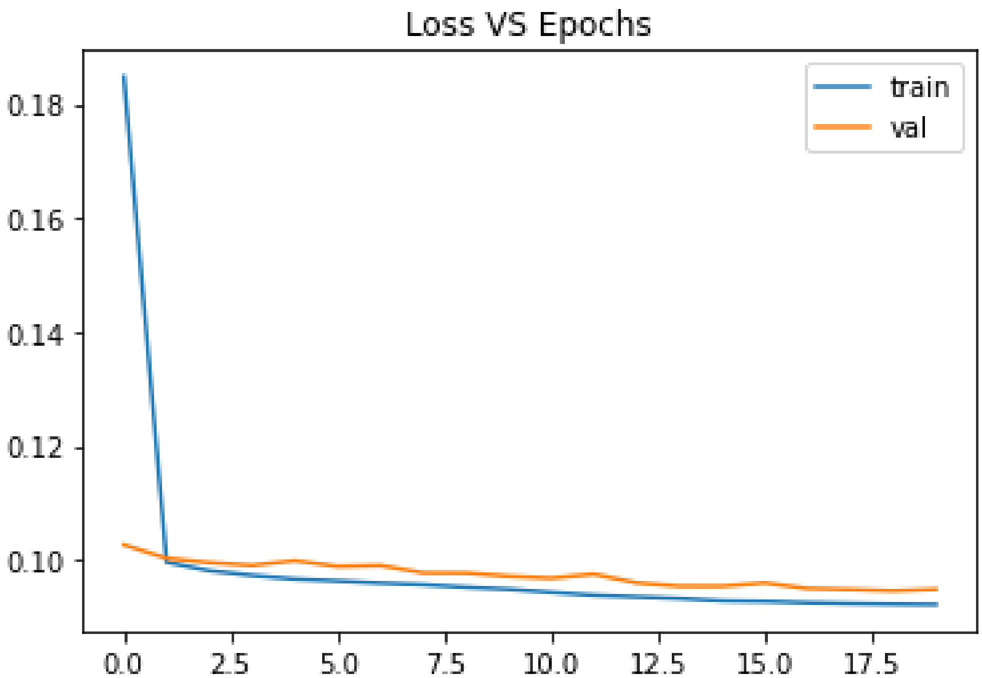
[49]:
```
model.compile(optimizer = "adam",
              loss = MeanSquaredLogarithmicError(),
              metrics = [RootMeanSquaredError(), MeanAbsolutePercentageError()])
```

[50]:
```
history1 = model.fit(X_train_scaled, y_train,
                     epochs = 20,
                     verbose = 0,
                     validation_split = 0.1)
```

```
2023-01-04 01:51:07.946829: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:116] None of the MLIR
optimization passes are enabled (registered 2)
```

[51]:
```
def plotNeuralNetworkGraph(historyModel):
    epochs = historyModel.epoch
    loss = historyModel.history["loss"]
    accuracy = historyModel.history["root_mean_squared_error"]
    val_loss = historyModel.history["val_loss"]
    val_accuracy = historyModel.history["val_root_mean_squared_error"]
    plt.figure()
    plt.plot(epochs, loss, label = "train")
    plt.plot(epochs, val_loss, label = "val")
    plt.legend()
    plt.title("Loss VS Epochs")
    plt.show()
```

[52]:
```
plotNeuralNetworkGraph(history1)
```



0.0.9 8. Tuning Neural Network

In previous model there was no BatchNormalization so by adding BatchNormalization we can further fine tune the model and increase accuracy.

Also I added normal kernel_initializer as a hyper-parameter to fine tune the model.

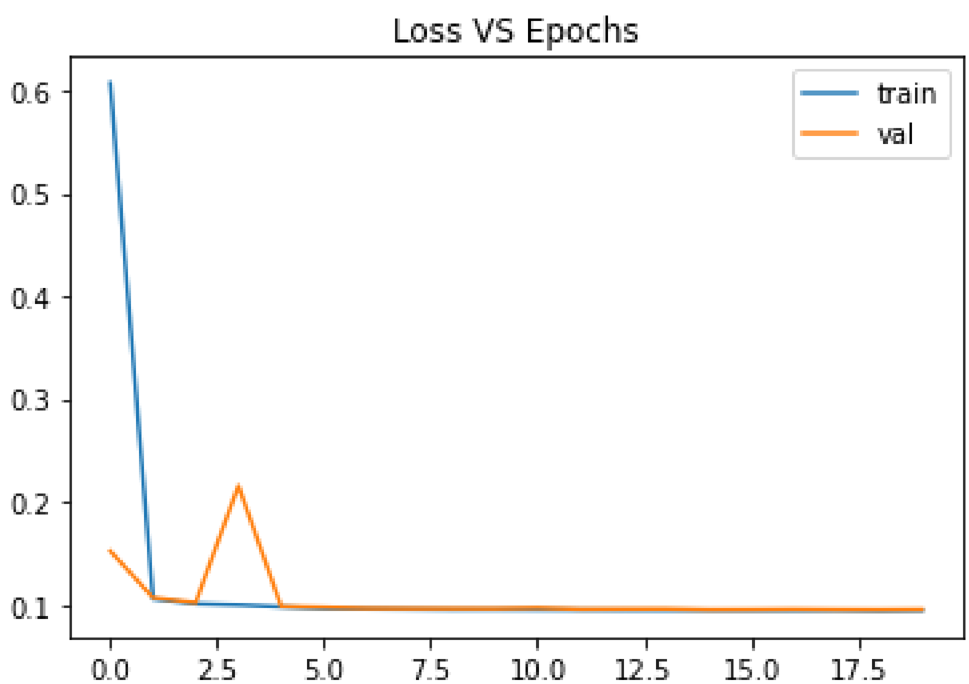
[53]:
```
modelNN2 = Sequential([
                        Dense(16,
                              activation = "relu",
                              kernel_initializer = "normal",
                              input_shape = (X_train_scaled.shape[1],)),
                        BatchNormalization(),
                        Dense(32,
                              activation = "relu",
                              kernel_initializer = "normal",),
                        BatchNormalization(),
                        Dense(64,
                              activation = "relu",
                              kernel_initializer = "normal",),
                        BatchNormalization(),
                        Dense(128,
                              activation = "relu",
                              kernel_initializer = "normal",),
                        BatchNormalization(),
```

```
                        Dense(1,
                              activation = "linear")
        ])
```

[54]:
```
modelNN2.compile(optimizer = "adam",
                 loss = MeanSquaredLogarithmicError(),
                 metrics = [RootMeanSquaredError(),␣
 ↪MeanAbsolutePercentageError()])
```

[55]:
```
history2 = modelNN2.fit(X_train_scaled, y_train,
                     epochs = 20,
                     verbose = 0,
                     validation_split = 0.1)
```

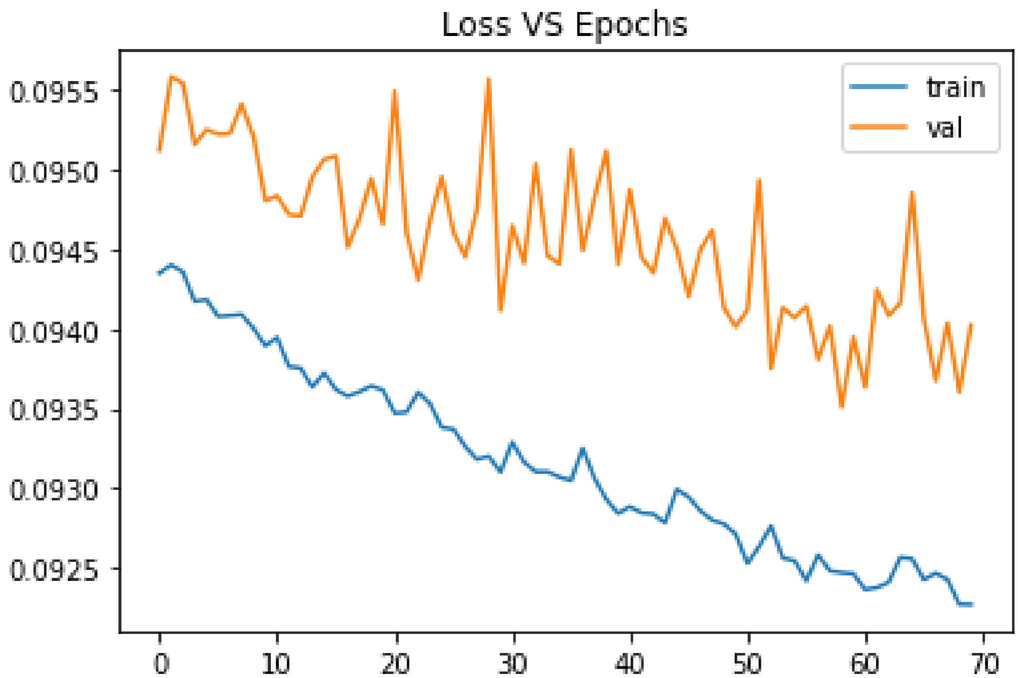
[56]:
```
plotNeuralNetworkGraph(history2)
```



One better way would be to run the model for 50 epochs and in this way we can see that in real time how the model has performed.

[57]:
```
history3 = modelNN2.fit(X_train_scaled, y_train,
                     epochs = 70,
                     verbose = 0,
                     validation_split = 0.1)
```

[58]:
```
plotNeuralNetworkGraph(history3)
```



[59]:
```
modelAccuracy(y_test, modelNN2.predict(X_test_scaled))
```

```
MAE: 11.38
RMSE: 17.64
MAPE: 0.25
```

Hence, we can see that we got a model where we got around as MAE, as RMSE and as MAPE which is far better than RandomForestRegressor

We can further improve model by increasing complexity and epochs so that more non complexity can be added and model can be improved further by more training like GPTs or Transformers

0.0.10 Questionnaire

Q1. Defining the problem statements and where can this and modifications of this be used? Ans.Porter is India's Largest Marketplace for Intra-City Logistics. Leader in the country's $40 billion intra-city logistics market, Porter strives to improve the lives of 1,50,000+ driver-partners by providing them with consistent earning & independence. Currently, the company has serviced 5+ million customers. Porter works with a wide range of restaurants for delivering their items directly to the people. Porter has a number of delivery partners available for delivering the food, from various restaurants and wants to get an estimated delivery time that it can provide the customers on the basis of what they are ordering, from where and also the delivery partners. We need to train a Linear Regression model that will do the delivery time estimation, based on all those features. The modification of this can be used for predicting delievery time for Food delievery apps like zomato, swiggy or cab booking like ola or uber

Q2. List 3 functions the pandas datetime provides with one line explanation. Ans. Pandas datetime provides several functions like

pd.to_datetime(column).dt.hour -> This will extract the hour present in the time

pd.to_datetime(column).dt.day_name -> This will give us the day name from time like Friday, Monday, Thursday etc.

pd.Timedelta(datetime2 - datetime1) -> Represents a duration, the difference between two dates or times.

Q3. Short note on datetime, timedelta, time span (period) Ans. The basic difference is:-

datetime -> The datetime is used to work with dates and times. It provides a variety of classes for representing and manipulating dates and times, as well as for formatting and parsing dates and times in a variety of formats like today's date 2023-01-04 (in format of YYYY-MM_DD) is a datetime

timedelta -> It is generally used for calculating differences in dates and also can be used for date manipulations in Python. Like if i want date after 7 days then i can use datetime and timedelta together i.e. by 2023-01-04 + timedelta(days = 7) therefore i will get 2023-01-11

time span (period) -> time span is defined as the difference between two datetimes and it can be expressed in days / hours/ minutes/seconds like 2023-01-11 - 2023-01-04 so it will give 7 days or 168 hours or 10080 minutes or 604800 seconds

Q4. Why do we need to check for outliers in our data? Ans. The outliers can disrupt the analysis as they are something which rarely occur or happened mistakenly like human error or so. Analysing data with them will skew our analysis and also while building the model since model wants to minimise something i.e. in our case mean_squared_error then the equation of hyperplane will start shifting towards outlier points thereby increasing error from normal correct points.

Q5. Name 3 outlier removal methods? Ans. The techniques are:-

IQR method

Eliptical Envelope

Isolation forest

Local Outlier Forest

Q6. What classical machine learning methods can we use other than random forest for regression? Ans. The other Machine learning techniques are:-

Linear Regression

Polynomial Regression

Decision Tree Regressor

GBDT Regressor

Q7. Why is scaling required for neural networks? Ans. Since, computations needs to be done in neural network and in case of different scales the whole multiplication would becomes bias towards feature having large values and also more time would be taken for computation. Additionally, it would lead to exploding gradients as well as memory overflow error thereby features should be scaled in neural networks.

Q8. Briefly explain your choice of optimizer. Ans. I choose ADAM as it is little efficient than SGD and RMSProp as it can apply to features when scales are different and doesn't shoot gradient in the direction of feature having little diverse range and also convergence is little better.

Q9. Which activation function did you use and why? Ans. I choose RELU as the activation function in hidden layers because it adds good non linearity and is safe as doesn't causes vanishing gradient problem. It is easy to calculate as the function is simply max(0, x), so there is no need to use expensive operations like exponentials or trigonometric functions. This makes it much faster to compute than other activation functions.

Also, in output layer i used linear because this is linear regression problem and we want single value.

Q10. Why does a neural network perform well on a large dataset? Ans. There are several reasons why a neural network might perform well on a large dataset:

A neural network is able to learn and model very complex patterns in data, so it is well-suited to handling large and complex datasets.

A larger dataset gives the neural network more examples to learn from, which can improve its accuracy and generalization to new data.

With more data, the neural network has a better chance of learning the underlying patterns in the data

A neural network is able to learn and improve over time, so as it processes more data, it can continue to improve its performance.