Business case

by

Lohith Kumar Kasula

Context:

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

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

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

This dataset has the required data to train a regression model that will do the delivery time estimation, based on all those features

Data Dictionary

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

market_id : integer id for the market where the restaurant lies

created_at : the timestamp at which the order was placed

actual_delivery_time : the timestamp when the order was delivered

store_primary_category : category for the restaurant

order_protocol : integer code value for order protocol (how the order was placed ie through porter, call to restaurant, pre booked, third part etc)

total_items subtotal : final price of the order

num_distinct_items : the number of distinct items in the order

min_item_price : price of the cheapest item in the order

max_item_price : price of the costliest item in order

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

total_busy_partners : number of delivery partners attending to other tasks

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

estimated_store_to_consumer_driving_duration : approximate travel time from restaurant to customer

Process and Techniques used

- Import the data and understand the structure of the data:
  - usual exploratory analysis steps like checking the structure & characteristics of the dataset
- Data preprocessing
  - Cleaning of data
  - Feature engineering: Creating the target column time taken in each delivery from order timestamp (created_at) and delivery timestamp (actual_delivery_time)
  - Getting hour of day from the order time and also the day of the week
  - Understanding pandas datetime data type and what function it provides by default
  - Get delivery time in minutes
- Handling null values
- Encoding categorical columns
- Data visualization and cleaning
  - Visualize various columns for better understanding Countplots, scatterplots
- Check if the data contains outliers
  - Removing outliers by any method
  - Plotting the data again to see if anything has improved
- Split the data into train and test
- Scaling the data for neural networks.
- Creating a simple neural network
  - Trying different configurations
  - Understanding different activation functions, optimizers and other hyperparameters.
- Training the neural network for required amount of epochs
- Plotting the losses and checking the accuracy of the model
- Checking its various metrics like MSE, RMSE, MAE

Importing all the required packages

In [1]:
```
!pip install colorama
!pip install scikit-learn
!pip install tensorflow==2.18.0
!pip install keras-tuner
```

```
Requirement already satisfied: colorama in /usr/local/lib/python3.11/dist-package
s (0.4.6)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-pac
kages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-pa
ckages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-pac
kages (from scikit-learn) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-pa
ckages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/
dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: tensorflow==2.18.0 in /usr/local/lib/python3.11/di
st-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-p
ackages (from tensorflow==2.18.0) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.11/dis
t-packages (from tensorflow==2.18.0) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.11/
dist-packages (from tensorflow==2.18.0) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/
lib/python3.11/dist-packages (from tensorflow==2.18.0) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.11/d
ist-packages (from tensorflow==2.18.0) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.11/dist
-packages (from tensorflow==2.18.0) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.11/dis
t-packages (from tensorflow==2.18.0) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packag
es (from tensorflow==2.18.0) (24.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.2
1.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.11/dist-packages (from
tensorflow==2.18.0) (5.29.5)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/d
ist-packages (from tensorflow==2.18.0) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packa
ges (from tensorflow==2.18.0) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-pack
ages (from tensorflow==2.18.0) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist
-packages (from tensorflow==2.18.0) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python
3.11/dist-packages (from tensorflow==2.18.0) (4.14.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-pa
ckages (from tensorflow==2.18.0) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/d
ist-packages (from tensorflow==2.18.0) (1.73.0)
Requirement already satisfied: tensorboard<2.19,>=2.18 in /usr/local/lib/python3.
11/dist-packages (from tensorflow==2.18.0) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-pac
kages (from tensorflow==2.18.0) (3.8.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /usr/local/lib/python3.11/
dist-packages (from tensorflow==2.18.0) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-pac
kages (from tensorflow==2.18.0) (3.14.0)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /usr/local/lib/python3.
11/dist-packages (from tensorflow==2.18.0) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/loca
l/lib/python3.11/dist-packages (from tensorflow==2.18.0) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.11/di
```

```
st-packages (from astunparse>=1.6.0->tensorflow==2.18.0) (0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (f
rom keras>=3.5.0->tensorflow==2.18.0) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow==2.18.0) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow==2.18.0) (0.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.11/dist-packages (from requests<3,>=2.21.0->tensorflow==2.18.0) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pac
kages (from requests<3,>=2.21.0->tensorflow==2.18.0) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/di
st-packages (from requests<3,>=2.21.0->tensorflow==2.18.0) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/di
st-packages (from requests<3,>=2.21.0->tensorflow==2.18.0) (2025.6.15)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.11/dist-
packages (from tensorboard<2.19,>=2.18->tensorflow==2.18.0) (3.8)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/loca
l/lib/python3.11/dist-packages (from tensorboard<2.19,>=2.18->tensorflow==2.18.0)
(0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.11/dist-
packages (from tensorboard<2.19,>=2.18->tensorflow==2.18.0) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dis
t-packages (from werkzeug>=1.0.1->tensorboard<2.19,>=2.18->tensorflow==2.18.0)
(3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.1
1/dist-packages (from rich->keras>=3.5.0->tensorflow==2.18.0) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.
11/dist-packages (from rich->keras>=3.5.0->tensorflow==2.18.0) (2.19.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packa
ges (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow==2.18.0) (0.1.2)
Requirement already satisfied: keras-tuner in /usr/local/lib/python3.11/dist-pack
ages (1.4.7)
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages
(from keras-tuner) (3.8.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packag
es (from keras-tuner) (24.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-package
s (from keras-tuner) (2.32.3)
Requirement already satisfied: kt-legacy in /usr/local/lib/python3.11/dist-packag
es (from keras-tuner) (1.0.5)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages
(from keras->keras-tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from keras->keras-tuner) (2.0.2)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (f
rom keras->keras-tuner) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages
(from keras->keras-tuner) (0.1.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (f
rom keras->keras-tuner) (3.14.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages
(from keras->keras-tuner) (0.16.0)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-packag
es (from keras->keras-tuner) (0.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.11/dist-packages (from requests->keras-tuner) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pac
kages (from requests->keras-tuner) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/di
```

```
st-packages (from requests->keras-tuner) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/di
st-packages (from requests->keras-tuner) (2025.6.15)
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python
3.11/dist-packages (from optree->keras->keras-tuner) (4.14.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.1
1/dist-packages (from rich->keras->keras-tuner) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.
11/dist-packages (from rich->keras->keras-tuner) (2.19.1)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packa
ges (from markdown-it-py>=2.2.0->rich->keras->keras-tuner) (0.1.2)
```

In [2]:

```
#for reding and handling the data
import pandas as pd
import numpy as np
import os


import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import drive
from colorama import Fore, Back, Style
from sklearn.neighbors import LocalOutlierFactor



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

#random forest model training
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

#Ann training
from tensorflow.keras import Model
from tensorflow.keras import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LeakyReL
from sklearn.model_selection import train_test_split
from tensorflow.keras.losses import MeanSquaredLogarithmicError
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.losses import MeanAbsolutePercentageError
from tensorflow.keras.callbacks import EarlyStopping
import keras_tuner as kt


# from tensorflow.keras.metrics import  mean_absolute_percentage_error
# from tensorflow.keras.metrics import RootMeanSquaredError
# from tensorflow.keras.metrics import MeanAbsoluteError
from tensorflow.keras.optimizers import SGD, Adam
```

Mounting Google Drive

In [3]:
```
drive.mount('/content/drive')
df = pd.read_csv('/content/drive/MyDrive/Scaler_DSML_Digital_Notes/BusinessCases
df.head()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
```

Out[3]:

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol
0	1.0	2015-02-06 22:24:17	2015-02-06 23:11:17	4	1.0
1	2.0	2015-02-10 21:49:25	2015-02-10 22:33:25	46	2.0
2	2.0	2015-02-16 00:11:35	2015-02-16 01:06:35	36	3.0
3	1.0	2015-02-12 03:36:46	2015-02-12 04:35:46	38	1.0
4	1.0	2015-01-27 02:12:36	2015-01-27 02:58:36	38	1.0

EDA

Structure of the data

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

Out[4]: (175777, 14)

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

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

Observations:

- Identifiifed incorrect Data Types:
  - **market_id :** should be category (nominal) and it is wrongly typecasted as float
  - **created_at :** should be in datetime format but it is in object type
  - **actual_delivery_time :** should be in datetime format but it is in object type
  - **order_protocol :** This should be category type but it is wrong typecasted as float
  - **total_onshift_dashers:** This should be integer but not the float
  - **total_busy_dasher:** This should be integer but not the float
  - **total_outstanding_orders** This should also be the integer but not the float value
  - **estimated_store_to_consumer_driving_duration** Since the time here is calculate in seconds so this should be integer

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

Out[6]:

	0
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

Observations:

- There are no null values found in the dataset

Handing the incorrect datatypes in the given data

Converting the columns which are related to datetime to datetime object

In [7]:
```
df['created_at'] = pd.to_datetime(df['created_at'])
df['actual_delivery_time'] = pd.to_datetime(df['actual_delivery_time'])
```

Converitng the missclassified features from 'float' to 'int' data type and some categorical feature to category data type

In [8]:
```
df = df.astype({'market_id':'int', 'order_protocol':'int', 'total_onshift_dasher
```

Target feature creation using 'actual_delivery_time' and 'created_at' features

In [9]:
```
df['Total_Time_in_Minutes'] = (df['actual_delivery_time'] - df['created_at']).dt
df['estimated_store_to_consumer_driving_duration'] = np.round(df['estimated_stor
```

In [10]:
```
df= df.astype({'Total_Time_in_Minutes':'float'})
```

In [11]:
```
df.head(2)
```

Out[11]:

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol
0	1	2015-02-06 22:24:17	2015-02-06 23:11:17	4	1
1	2	2015-02-10 21:49:25	2015-02-10 22:33:25	46	2

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 15 columns):
 #   Column                                        Non-Null Count   Dtype
---  ------                                        --------------   -----
 0   market_id                                     175777 non-null  category
 1   created_at                                    175777 non-null  datetime64[n
s]
 2   actual_delivery_time                          175777 non-null  datetime64[n
s]
 3   store_primary_category                        175777 non-null  category
 4   order_protocol                                175777 non-null  category
 5   total_items                                   175777 non-null  int64
 6   subtotal                                      175777 non-null  int64
 7   num_distinct_items                            175777 non-null  int64
 8   min_item_price                                175777 non-null  int64
 9   max_item_price                                175777 non-null  int64
 10  total_onshift_dashers                         175777 non-null  int64
 11  total_busy_dashers                            175777 non-null  int64
 12  total_outstanding_orders                      175777 non-null  int64
 13  estimated_store_to_consumer_driving_duration  175777 non-null  float64
 14  Total_Time_in_Minutes                         175777 non-null  float64
dtypes: category(3), datetime64[ns](2), float64(2), int64(8)
memory usage: 16.6 MB
```

Data Frame statistics

Stats of Categorical columns

In [13]:
```
df.describe(include='category')
```

Out[13]:

	market_id	store_primary_category	order_protocol
count	175777	175777	175777
unique	6	73	7
top	2	4	1
freq	53469	18183	48404

Observation:

- 53469 orders were placed to market id 2 which is the highest among all the other market id's
- 48404 orders were placed with the protocol of 1 which is the highest among all the other order protocols

Stats of Numerical columns

In [14]:
```
df.describe(include='int')
```

Out[14]:

	total_items	subtotal	num_distinct_items	min_item_price	max_item_pric
count	175777.000000	175777.000000	175777.000000	175777.000000	175777.00000
mean	3.204976	2697.111147	2.675060	684.965433	1160.15861
std	2.674055	1828.554893	1.625681	519.882924	560.82857
min	1.000000	0.000000	1.000000	-86.000000	0.00000
25%	2.000000	1412.000000	1.000000	299.000000	799.00000
50%	3.000000	2224.000000	2.000000	595.000000	1095.00000
75%	4.000000	3410.000000	3.000000	942.000000	1395.00000
max	411.000000	26800.000000	20.000000	14700.000000	14700.00000

Stats of DateTime columns

In [15]:
```
df.describe(include='datetime')
```

Out[15]:

	created_at	actual_delivery_time
count	175777	175777
mean	2015-02-04 19:57:50.009631744	2015-02-04 20:44:02.190406144
min	2015-01-21 15:22:03	2015-01-21 16:07:03
25%	2015-01-29 01:31:19	2015-01-29 02:16:30
50%	2015-02-05 02:41:26	2015-02-05 03:34:33
75%	2015-02-12 01:04:32	2015-02-12 01:48:24
max	2015-02-18 06:00:44	2015-02-18 06:51:10

Handling the Redundant samples in the data

Filtering and Removing the Redunant samples in the data

In [16]:
```
df[df['subtotal'] <= 0].head(5)
```

Out[16]:

	market_id	created_at	actual_delivery_time	store_primary_category	order_protoc
988	4	2015-01-25 17:28:32	2015-01-25 18:05:32	4	
1694	2	2015-02-07 17:25:00	2015-02-07 18:00:00	28	
3652	1	2015-01-31 01:42:46	2015-01-31 02:19:46	39	
3909	1	2015-02-12 19:15:20	2015-02-12 19:58:20	10	
6617	4	2015-01-25 02:09:31	2015-01-25 02:53:31	34	

In [17]:
```
wrongly_filled_data_sample = df[(df['min_item_price'] < 0) | (df['total_onshift_
```

In [18]:
```
print('Total Redundant Samples in dataset :' , len(wrongly_filled_data_sample))
print('Percentage of Redudant Samples in the dataset :', round((len(wrongly_fill
```

```
Total Redundant Samples in dataset : 251
Percentage of Redudant Samples in the dataset : 0.14
```

In [19]:
```
df.drop(index=wrongly_filled_data_sample.index, inplace=True)
```

Structure of the data samples after removing the redundant samples

In [20]:
```
df.shape
```

Out[20]:
```
(175526, 15)
```

In [21]:
```
df.describe(include='int')
```

Out[21]:

	total_items	subtotal	num_distinct_items	min_item_price	max_item_pric
count	175526.000000	175526.000000	175526.000000	175526.000000	175526.00000
mean	3.204568	2699.656336	2.675148	684.902812	1160.16849
std	2.672512	1827.692887	1.625710	519.808712	560.80622
min	1.000000	95.000000	1.000000	0.000000	0.00000
25%	2.000000	1418.000000	1.000000	299.000000	799.00000
50%	3.000000	2225.000000	2.000000	595.000000	1095.00000
75%	4.000000	3413.000000	3.000000	942.000000	1395.00000
max	411.000000	26800.000000	20.000000	14700.000000	14700.00000

Feature Engineering

Features Extraction

Following are features extracted form the 'created_at' feature

- Created_Year
- Created_Day
- Created_Month
- Created_Hour
- Created_Date
- Time_of_Day

Feature createation by using the 'Total_Time_in_Minutes' and 'estimated_store_to_consumer_driving_duration'

- food_preparation_and_wait_time

In [22]:
```
df['Created_Year'] = df['created_at'].dt.year
df['Created_Day'] = df['created_at'].dt.strftime('%a')
df['Created_Month'] = df['created_at'].dt.strftime('%b')
df['Created_Hour'] = pd.to_datetime(df['created_at'], format='%Y-%m-%d %H:%M:%S'
df['Created_Hour'] = df['Created_Hour'].astype('int')
df['Created_Date'] = pd.to_datetime(df['created_at'], format='%Y-%m-%d %H:%M:%S'
df['food_preparation_and_wait_time'] = df['Total_Time_in_Minutes'] - df['estimat
```

In [23]:
```
def time_of_day(hour):
    if 0 <= hour < 5:
        return 'Midnight (12AM-5AM)'
    elif 5 <= hour < 8:
        return 'Early Morning (5AM-8AM)'
    elif 8 <= hour < 12:
        return 'Morning (8AM-12PM)'
    elif 12 <= hour < 15:
        return 'Afternoon (12PM-3PM)'
    elif 15 <= hour < 18:
        return 'Late Afternoon (3PM-6PM)'
    elif 18 <= hour < 21:
        return 'Evening (6PM-9PM)'
    else:
        return 'Night (9PM-12AM)'
df['Time_of_Day'] = df['Created_Hour'].apply(time_of_day)
```

Features Drop

The below features are nomore useful for our analysis, Hence these features can be removed

- created_at
- actual_delivery_time
- Created_Hour
- Created_Year

In [24]:
```
df.drop(columns=['created_at', 'actual_delivery_time', 'Created_Hour'], inplace=
df.drop(columns=['Created_Year'], inplace=True)
```

Features distinct values summary

In [25]:
```
def print_unique_values(dataframe, features):
  for _ in features:
    print(f'{Fore.GREEN}+-'*50)
    print(f'{Fore.RED}{_}:')
    if dataframe[_].dtype == 'category':
      print(f'{Fore.MAGENTA}Length : {Fore.BLUE} {len(dataframe[_].cat.categorie
      print(f'{Fore.MAGENTA}Values : {Fore.BLUE} {dataframe[_].cat.categories.va
    else:
      print(f'{Fore.MAGENTA}Length : {Fore.BLUE} {dataframe[_].nunique()}')
      print(f'{Fore.MAGENTA}Values : {Fore.BLUE} {dataframe[_].unique()}')
```

In [26]:
```
print_unique_values(df, df.columns)
```

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
market_id:
Length :  6
Values :  [1 2 3 4 5 6]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
store_primary_category:
Length :  73
Values :  [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 2
3
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71
 72]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
order_protocol:
Length :  7
Values :  [1 2 3 4 5 6 7]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
total_items:
Length :  54
Values :  [  4   1   2   3   5   7  10   6   9   8  13  16  12  25  17  11  24  3
0
  14  31  40  26  28  21  20  15  19  18  23  29  34  35  42  22  56  36
  57  39  47  45  38  33  27  41 411  32  59  51  37  48  44  49  64  66]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
subtotal:
Length :  8179
Values :  [ 3441  1900  4771 ... 12145   348  9317]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
num_distinct_items:
Length :  20
Values :  [ 4  1  3  2  5  7  6 10  8 14  9 11 12 13 15 20 18 16 17 19]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
min_item_price:
Length :  2233
Values :  [ 557 1400  820 ... 8959 2047 1213]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
max_item_price:
Length :  2576
Values :  [1239 1400 1604 ... 3879 2868  448]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
total_onshift_dashers:
Length :  168
Values :  [ 33   1   8   5   4   6  24  12  19  21  22  16  27  13 117 130 121  3
4
  18  17  38  20  15  39  26  30  35  11  14  32  29  25  23  61  41  58
  51  57  60   7   3   2   9 108  72  89 112  55 126  80  69 125 113 105
  98  86  90  87  74  84  91  96 123 101 119 129  66  81  75  43  70  65
  77 114  62 116 102  76  67  64  88  73  63  46  28  36  47 110  97  42
  79  49  50  37  92 103  44  68  56   0 107  94  78  71 128 104  45  31
 100  53  40 115 109  99  85  59 118  10  54  52  48 132 127 133 131 149
  93 111  95  82 136 120 134 124 106 137  83 152 147 144 150 145 122 155
```

```
 135 151 140 141 143 138 139 156 146 160 158 153 148 142 157 171 154 162
 165 163 164 159 169 168]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
total_busy_dashers:
Length :  153
Values :  [ 14   2   6   5   1   4   9  24  13  19  17  11  21  25  16  18 112 12
9
 119  30  15  33  22  37  29  12  23  31  34  20  62  28  44  39  32  53
  52  59  57   7   3   8  10 100  72  81 106  84 122 109  54  89 103  91
  83  75 107 102  38  96  80  94  64 117 125 110 128  27  35  70  74  73
  40  97  77  61  58 116 123  42  26  95  68 108  55  78  67  69  99  45
  63  71  50  48  41  66 127   0  79  85  88  98 118  90  36 121  65  60
  47  49  43  46  93 132 104 114 124  56  51  76 101  92 130  86 138 136
  82  87 143 120 146 131 113 115 148 105 126 141 111 135 137 133 145 139
 134 142 150 144 154 147 140 149 152]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
total_outstanding_orders:
Length :  275
Values :  [ 21   2  18   8   7   1   3  12  26  11  30  16  39  24  27  20  13 17
8
 230 205  28  15  42  43  33   5  37  46  36   9  14  29  32  72  41  54
  49  23  56  60  10   4   6 154 144  34 100 173  80 202 124 130 181 188
 148 150 127 160 107 123 159 186 143 190 171 261 132  67 120 158  38  47
  68  92  99 175 102 184 121  17  96 111  22 131 141  89  48  65 194  73
  45 203 134 126 183  64 167  31 153 119  75  95  52 118 191  61 106  19
  58 152 101  35 151  83   0 179  97  44  25 147 169 135 142 165 172  91
  40 237 198 220 219  78  90 138 168  85  81 212 192 137 189  51  74  59
  62  63  50 122  93  66 207 113 114  69  57  86  71  79  87  88  70  84
 108  76  53  55  82 110  98 185 149 225 104 145 247 232  77 133 103 174
 164 193 208 231 163  94 146 115 129 112 136 176 246 199 116 109 161 105
 125 177 196 221 217 211 272 156 242 117 162 140 139 187 166 236 276 229
 213 180 155 256 254 243 204 157 200 128 195 170 201 197 215 210 182 222
 226 223 250 238 218 239 234 262 235 214 216 251 252 270 240 253 228 258
 209 274 206 224 285 248 269 257 241 227 278 244 265 249 268 277 233 264
 259 283 245 273 260]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
estimated_store_to_consumer_driving_duration:
Length :  1318
Values :  [14.35 11.5   4.82 ... 21.25  0.12 21.65]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
Total_Time_in_Minutes:
Length :  74
Values :  [ 47.  44.  55.  59.  46.  56.  63.  58.  37.  41.  45.  87.  67.  43.
  42.  40.  64.  34.  32.  48.  57.  38.  52.  49.  36.  39.  35.  53.
  50.  51.  61.  62.  65.  69.  68.  66.  33.  70.  60.  71.  54.  86.
  74.  93.  92.  85.  76.  90.  89.  77.  72.  75.  73.  79.  91.  88.
  78.  80.  81.  84.  83.  82.  98.  94. 101.  96. 103.  97.  99. 102.
  95. 100. 110. 105.]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
Created_Day:
Length :  7
Values :  ['Fri' 'Tue' 'Mon' 'Thu' 'Sun' 'Sat' 'Wed']
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
Created_Month:
```

```
Length :  2
Values :  ['Feb' 'Jan']
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
Created_Date:
Length :  29
Values :  [datetime.date(2015, 2, 6) datetime.date(2015, 2, 10)
 datetime.date(2015, 2, 16) datetime.date(2015, 2, 12)
 datetime.date(2015, 1, 27) datetime.date(2015, 2, 8)
 datetime.date(2015, 1, 31) datetime.date(2015, 2, 17)
 datetime.date(2015, 2, 15) datetime.date(2015, 2, 2)
 datetime.date(2015, 2, 13) datetime.date(2015, 1, 24)
 datetime.date(2015, 1, 26) datetime.date(2015, 2, 7)
 datetime.date(2015, 1, 30) datetime.date(2015, 2, 9)
 datetime.date(2015, 2, 14) datetime.date(2015, 2, 18)
 datetime.date(2015, 2, 1) datetime.date(2015, 1, 23)
 datetime.date(2015, 1, 22) datetime.date(2015, 1, 25)
 datetime.date(2015, 1, 28) datetime.date(2015, 2, 5)
 datetime.date(2015, 2, 4) datetime.date(2015, 2, 11)
 datetime.date(2015, 1, 29) datetime.date(2015, 2, 3)
 datetime.date(2015, 1, 21)]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
food_preparation_and_wait_time:
Length :  3754
Values :  [32.65 32.5  50.18 ... 63.55 64.63 69.3 ]
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
-+-+-+-+-+-+-+-+-+-
Time_of_Day:
Length :  7
Values :  ['Night (9PM-12AM)' 'Midnight (12AM-5AM)' 'Early Morning (5AM-8AM)'
 'Evening (6PM-9PM)' 'Late Afternoon (3PM-6PM)' 'Afternoon (12PM-3PM)'
 'Morning (8AM-12PM)']
```

Univariate Analysis

In [27]:
```python
def annotate(ax, rotation = False):
   for patch in ax.patches:  # Loop through each bar
        if rotation:  # For horizontal bars
            x = patch.get_width()  # Get the width (value of the bar)
            y = patch.get_y() + patch.get_height() / 2  # Center the annotation
            ax.annotate(f"{x:.2f}", (x + 0.5, y), ha='left', va='center')  # Adj
        else:  # For vertical bars
            x = patch.get_x() + patch.get_width() / 2  # Center the annotation h
            y = patch.get_height()  # Get the height (value of the bar)
            ax.annotate(f"{y:.2f}", (x, y + 0.5), ha='center', va='bottom')  # A
```

In [28]:
```python
dependent_feature = 'Total_Time_in_Minutes'
indepedent_fetures = df.columns[df.columns != dependent_feature].to_list()
indepedent_fetures.append(dependent_feature)
df = df[indepedent_fetures]
df.head(2)
```

Out[28]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinc
0	1	4	1	4	3441	
1	2	46	2	1	1900	

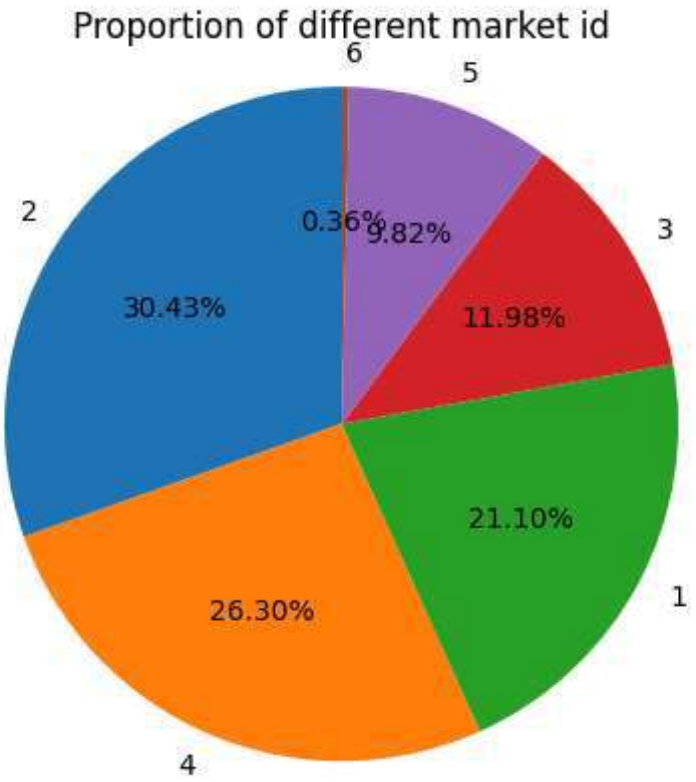
Categorical Features

Pie Plots

Proportion of Different Market where The Restaurant Lies

In [29]:
```
market_count_df = df['market_id'].value_counts().reset_index().sort_values(by='c
display(market_count_df)
plt.pie(x=market_count_df['count'], labels=market_count_df['market_id'], autopct
plt.axis('equal')
plt.title('Proportion of different market id')
plt.show()
```

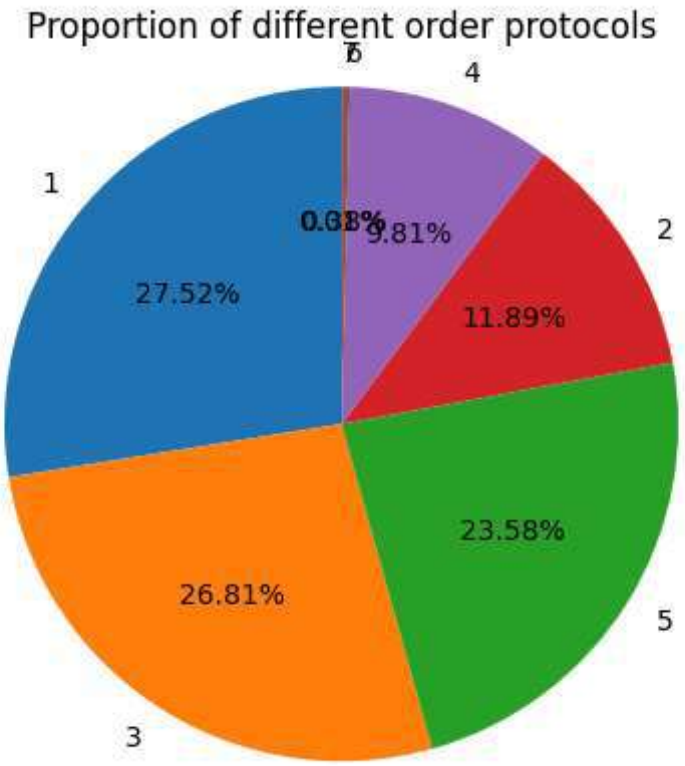
	market_id	count
0	2	53420
1	4	46170
2	1	37034
3	3	21027
4	5	17238
5	6	637



Proportion of How The Order Was Placed ie: Through Porter, Call to Restaurant, Pre Booked, Third Party etc

In [30]:
```
protocol_count_df = df['order_protocol'].value_counts().reset_index().sort_value
display(protocol_count_df)
plt.pie(x=protocol_count_df['count'], labels=protocol_count_df['order_protocol']
plt.axis('equal')
plt.title('Proportion of different order protocols')
plt.show()
```

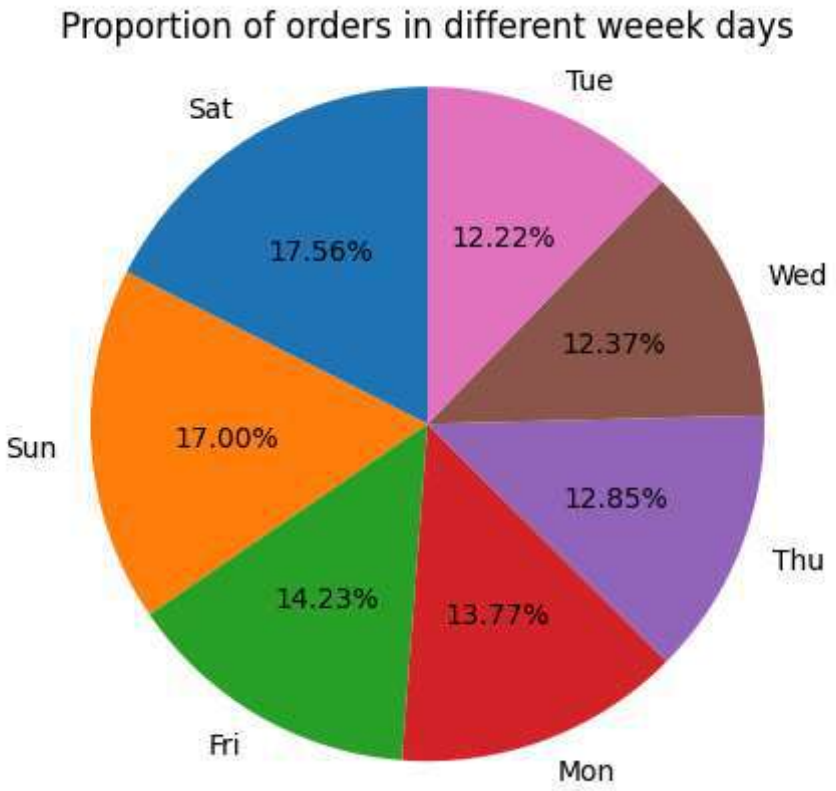
	order_protocol	count
0	1	48310
1	3	47064
2	5	41385
3	2	20862
4	4	17211
5	6	675
6	7	19



Proportion Of Orders in Different Weeek Days

In [31]:
```
created_day_df = df['Created_Day'].value_counts().reset_index().sort_values(by='
display(created_day_df)
plt.pie(x=created_day_df['count'], labels=created_day_df['Created_Day'], autopct
plt.axis('equal')
plt.title('Proportion of orders in different weeek days')
plt.show()
```

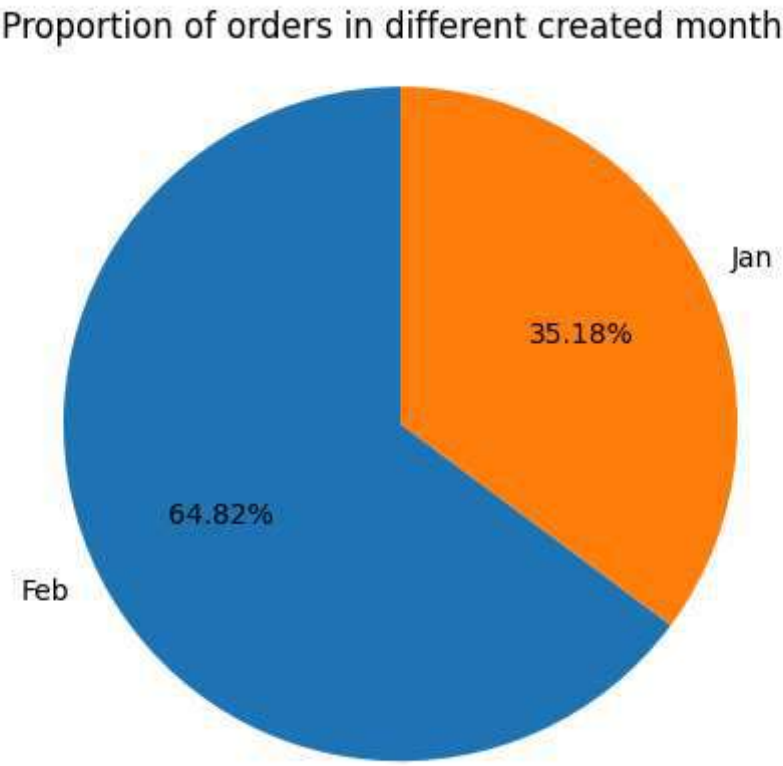
	Created_Day	count
0	Sat	30814
1	Sun	29846
2	Fri	24976
3	Mon	24171
4	Thu	22558
5	Wed	21711
6	Tue	21450



Proportion of Orders In Different Created Months

In [32]:
```
created_day_df = df['Created_Month'].value_counts().reset_index().sort_values(by
display(created_day_df)
plt.pie(x=created_day_df['count'], labels=created_day_df['Created_Month'], autop
plt.axis('equal')
plt.title('Proportion of orders in different created months')
plt.show()
```

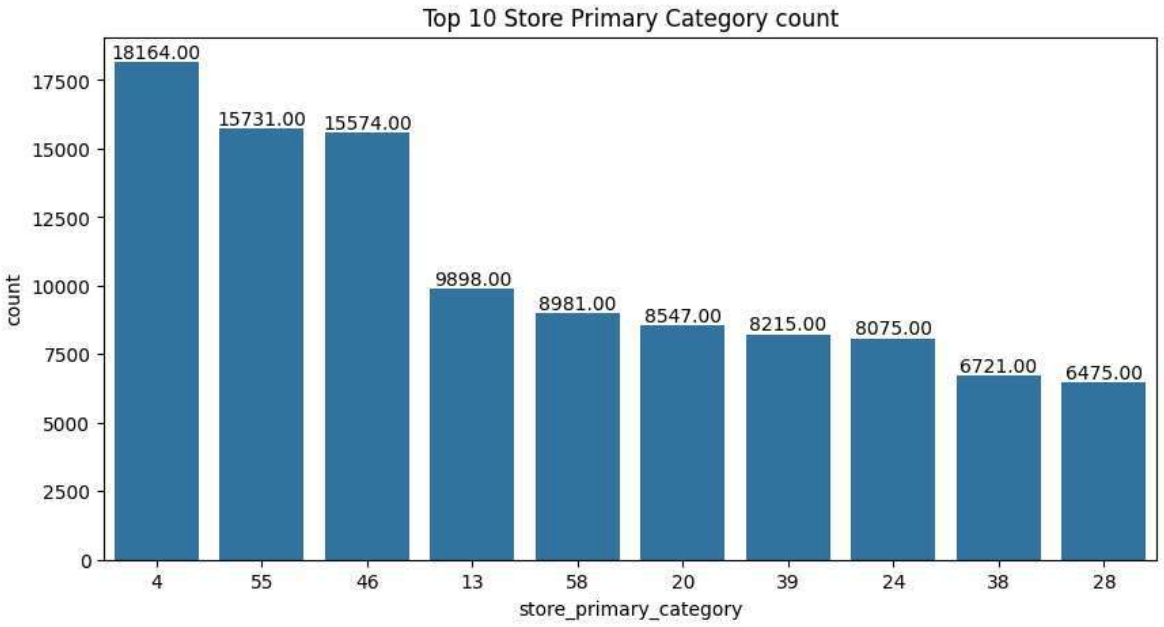
	Created_Month	count
0	Feb	113781
1	Jan	61745



Bar plots

Count Top 10 Restaurant Category Type

In [33]:
```
plt.figure(figsize=(10,5))
plt.title('Top 10 Store Primary Category count')
store_primary_category_value_counts = df['store_primary_category'].value_counts(
store_primary_category_value_counts = store_primary_category_value_counts.reset_
store_primary_category_value_counts_sorted = store_primary_category_value_counts
ax = sns.barplot(data=store_primary_category_value_counts_sorted, x='store_prima
annotate(ax)
plt.show()
store_primary_category_value_counts.head(10)
```

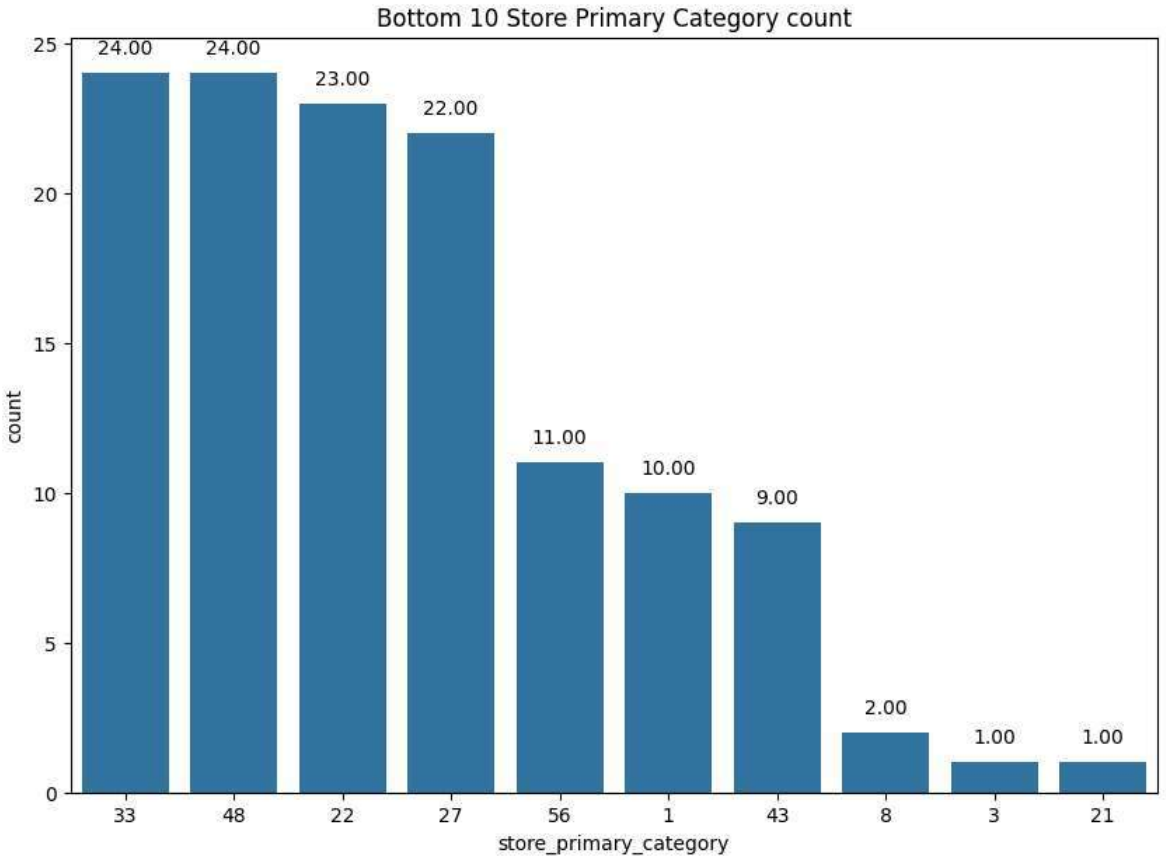


Out[33]:

	store_primary_category	count
0	4	18164
1	55	15731
2	46	15574
3	13	9898
4	58	8981
5	20	8547
6	39	8215
7	24	8075
8	38	6721
9	28	6475

Count Bottom 10 Restaurant Category Type

In [34]:
```
plt.figure(figsize=(10,7))
plt.title('Bottom 10 Store Primary Category count')
store_primary_category_value_counts = df['store_primary_category'].value_counts(
store_primary_category_value_counts = store_primary_category_value_counts.reset_
store_primary_category_value_counts_sorted = store_primary_category_value_counts
ax = sns.barplot(data=store_primary_category_value_counts_sorted, x='store_prima
annotate(ax)
plt.show()
store_primary_category_value_counts.tail(10)
```



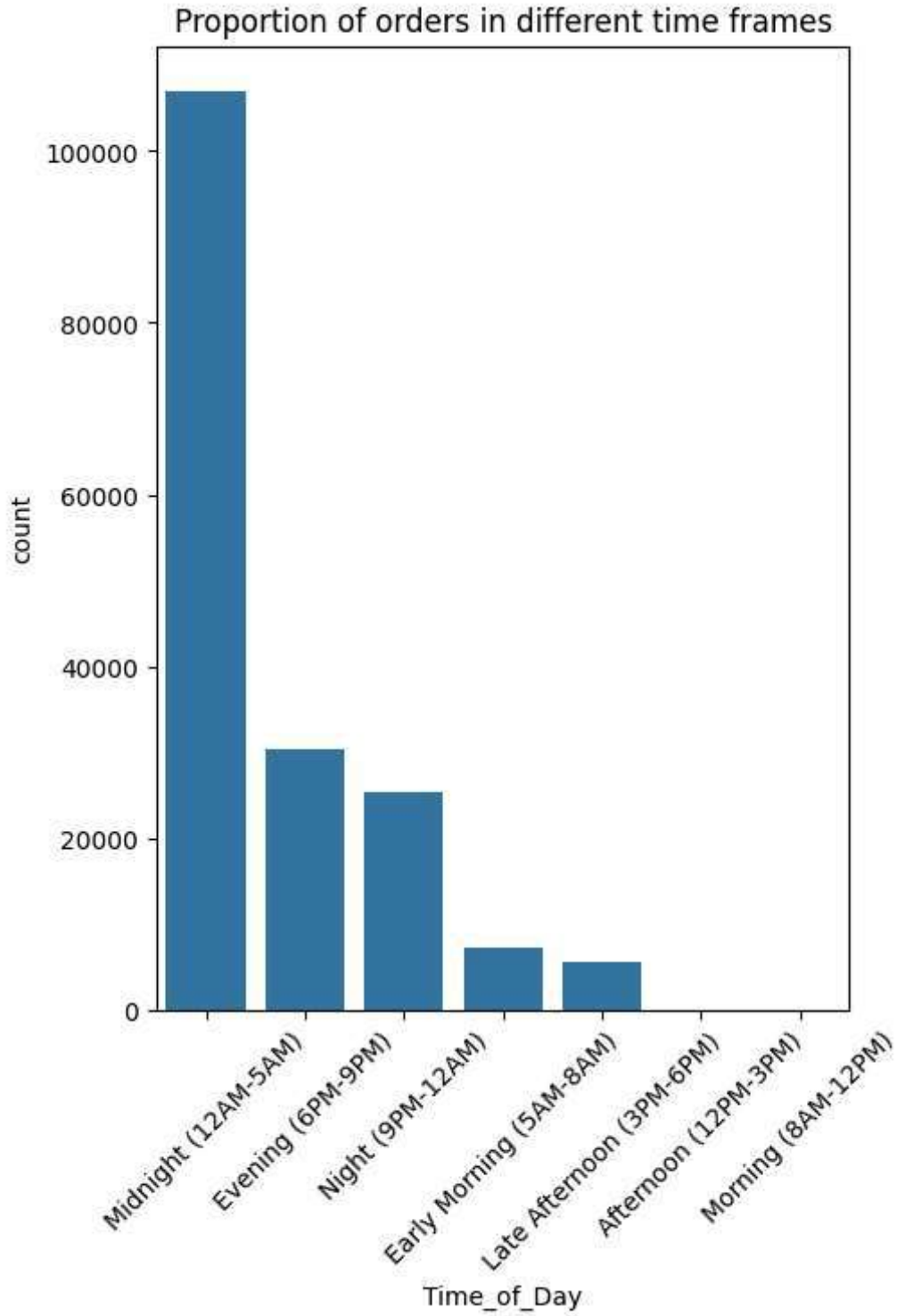
Out[34]:

	store_primary_category	count
63	33	24
64	48	24
65	22	23
66	27	22
67	56	11
68	1	10
69	43	9
70	8	2
71	3	1
72	21	1

Proportion of orders in different time frames

In [35]:

```
plt.figure(figsize=(5,7))
plt.title('Proportion of orders in different time frames')
created_time_of_day_df = df['Time_of_Day'].value_counts().reset_index().sort_val
sns.barplot(data=created_time_of_day_df, x='Time_of_Day', y='count', order = cre
plt.xticks(rotation=45)
plt.show()
display(created_time_of_day_df)
```



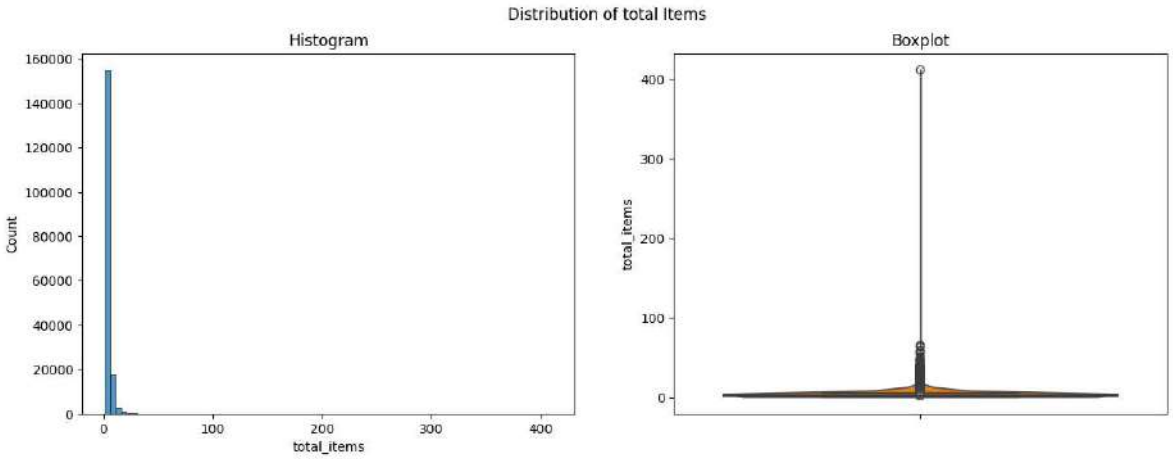
	Time_of_Day	count
0	Midnight (12AM-5AM)	106888
1	Evening (6PM-9PM)	30428
2	Night (9PM-12AM)	25392
3	Early Morning (5AM-8AM)	7298
4	Late Afternoon (3PM-6PM)	5481
5	Afternoon (12PM-3PM)	37
6	Morning (8AM-12PM)	2

Numerical Features

Histogram and boxplot Analysis

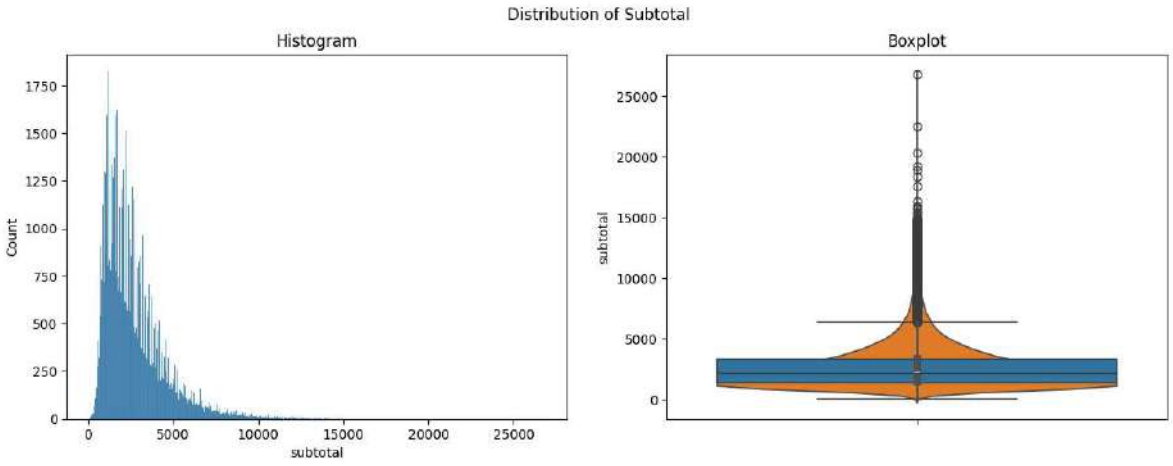
Distribution of total Items

In [36]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of total Items')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['total_items'], binwidth=5)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['total_items'])
sns.violinplot(df['total_items'])
plt.show()
```



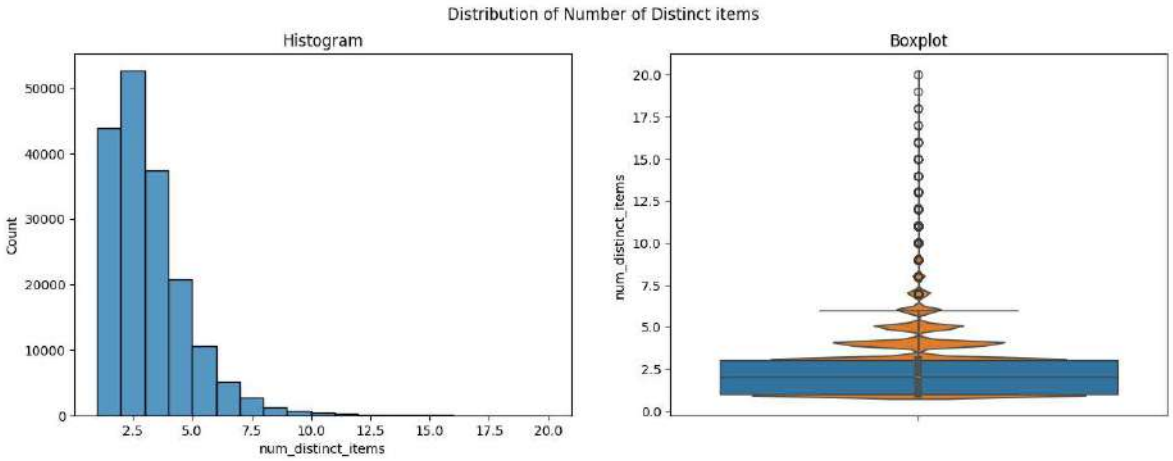
Distribution of Subtotal

In [37]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Subtotal')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['subtotal'], binwidth=20)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['subtotal'])
sns.violinplot(df['subtotal'])
plt.show()
```



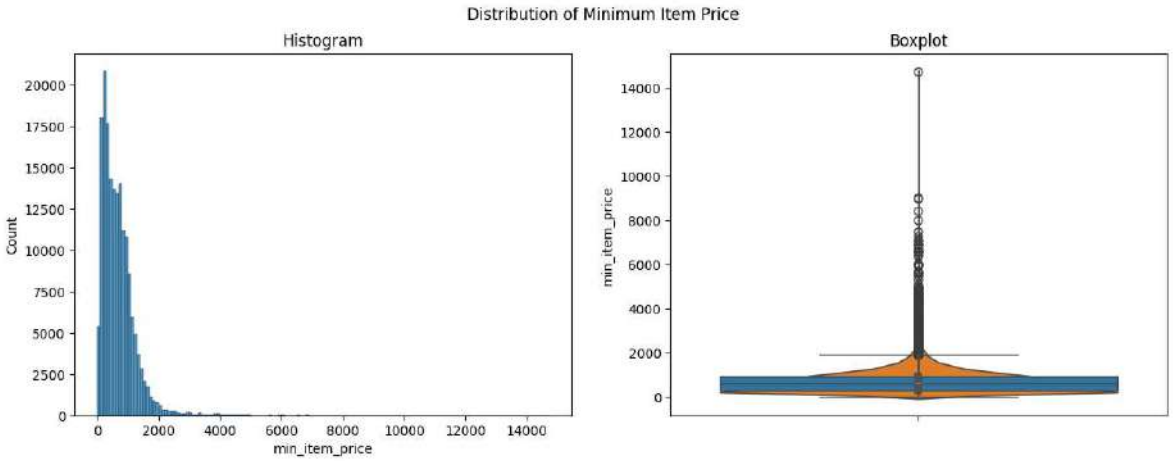
Distribution of Number of Distinct items

In [38]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Number of Distinct items')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['num_distinct_items'], binwidth=1)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['num_distinct_items'])
sns.violinplot(df['num_distinct_items'])
plt.show()
```



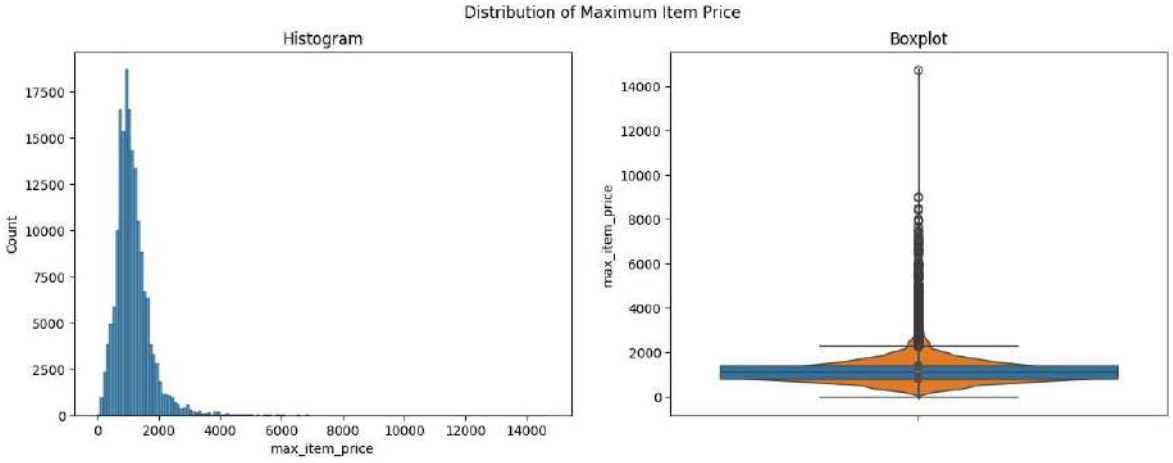
Distribution of Minimum Item Price

In [39]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Minimum Item Price')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['min_item_price'], binwidth=100)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['min_item_price'])
sns.violinplot(df['min_item_price'])
plt.show()
```



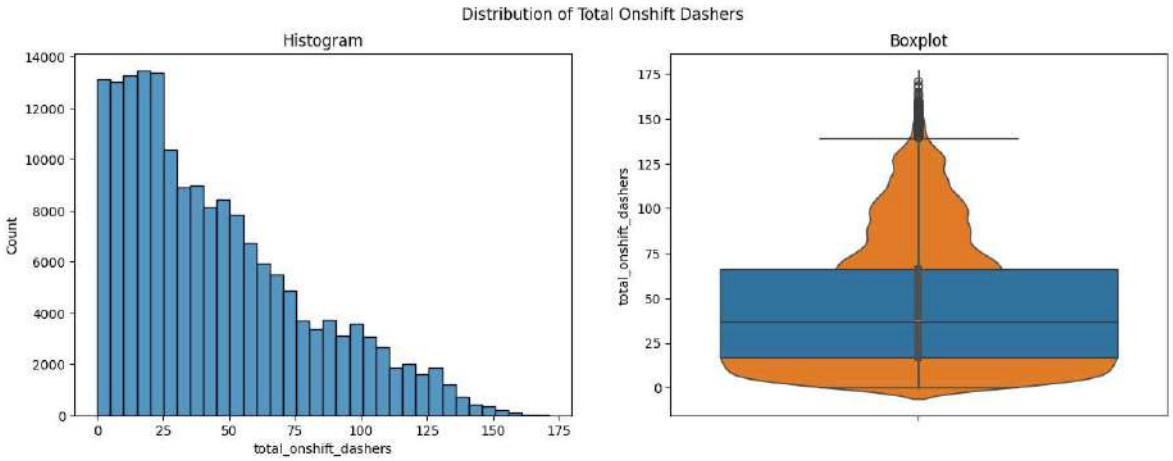
Distribution of Maximum Item Price

In [40]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Maximum Item Price')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['max_item_price'], binwidth=100)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['max_item_price'])
sns.violinplot(df['max_item_price'])
plt.show()
```



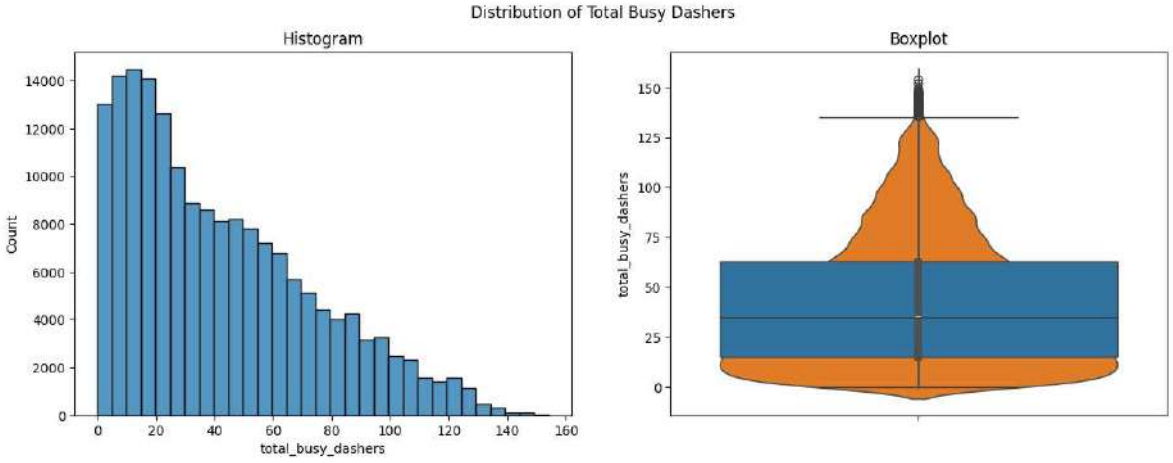
Distribution of Total Onshift Dashers

In [41]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Total Onshift Dashers')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['total_onshift_dashers'], binwidth=5)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['total_onshift_dashers'])
sns.violinplot(df['total_onshift_dashers'])
plt.show()
```



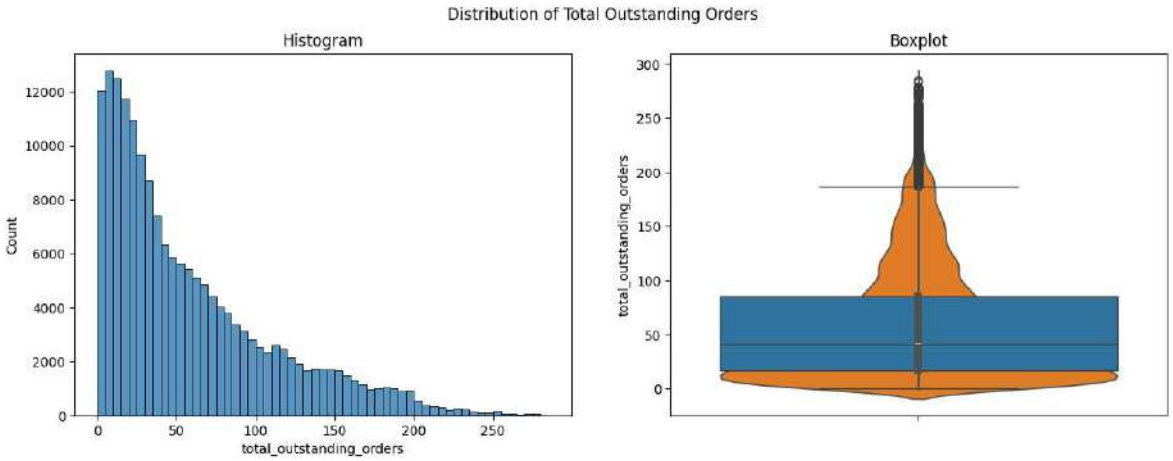
Distribution of Total Busy Dashers

In [42]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Total Busy Dashers')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['total_busy_dashers'], binwidth=5)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['total_busy_dashers'])
sns.violinplot(df['total_busy_dashers'])
plt.show()
```



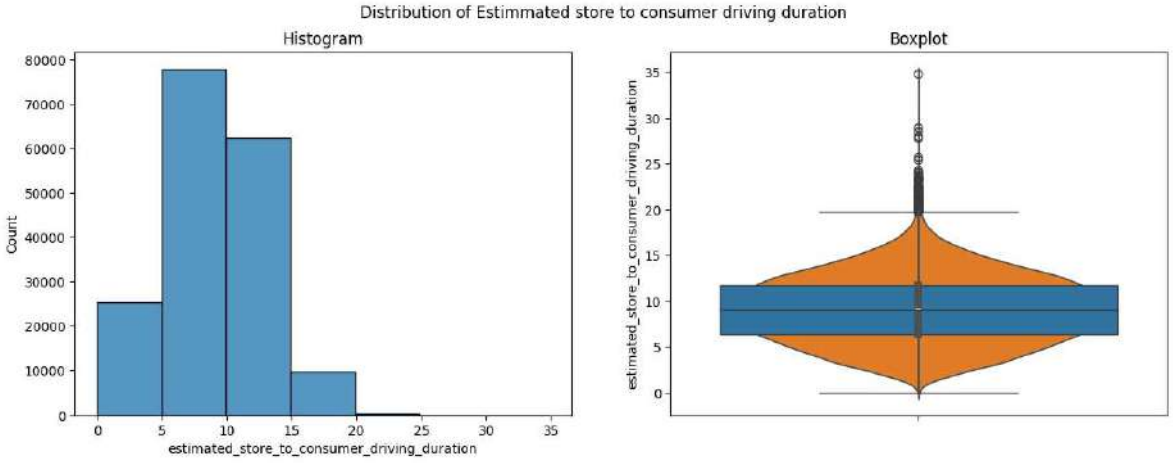
Distribution of Total Outstanding Orders

In [43]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Total Outstanding Orders')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['total_outstanding_orders'], binwidth=5)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['total_outstanding_orders'])
sns.violinplot(df['total_outstanding_orders'])
plt.show()
```



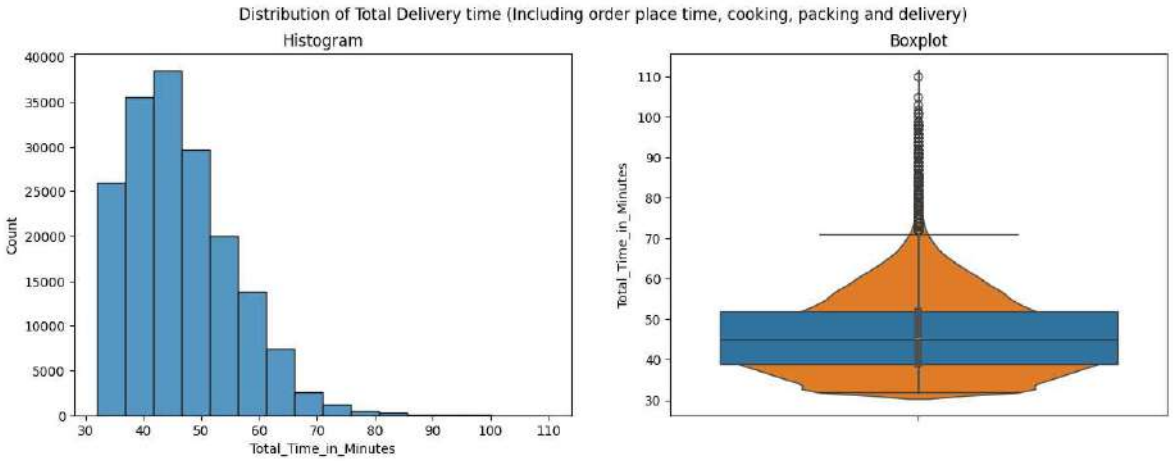
Distribution of Estimmated store to consumer driving duration

In [44]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Estimmated store to consumer driving duration')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['estimated_store_to_consumer_driving_duration'], binwidth=5)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['estimated_store_to_consumer_driving_duration'])
sns.violinplot(df['estimated_store_to_consumer_driving_duration'])
plt.show()
```



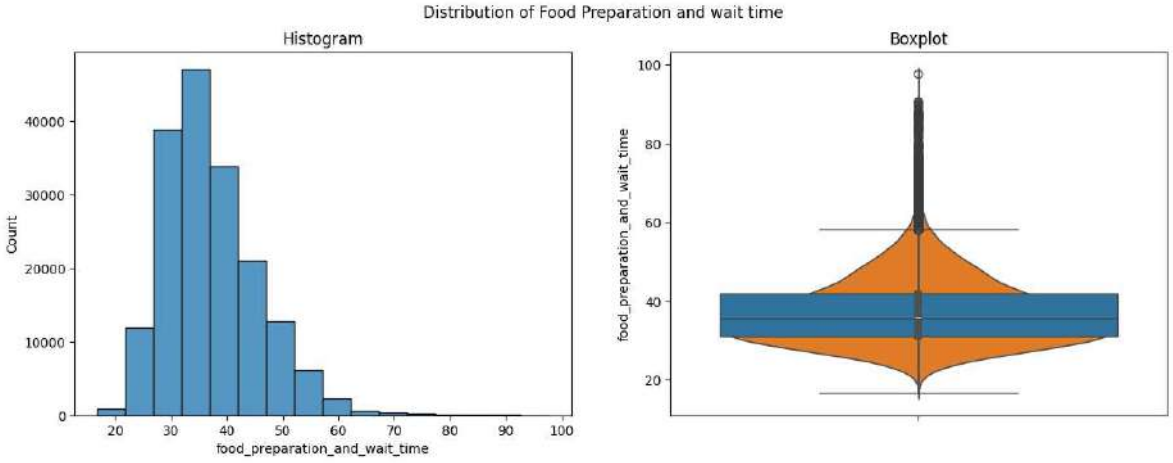
Distribution of Total Delivery time (Including order place time, cooking, packing and delivery)

In [45]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Total Delivery time (Including order place time, c

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['Total_Time_in_Minutes'], binwidth=5)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['Total_Time_in_Minutes'])
sns.violinplot(df['Total_Time_in_Minutes'])
plt.show()
```



Distribution of Food Preparation and wait time

In [46]:
```
plt.figure(figsize=(15,5))
plt.suptitle('Distribution of Food Preparation and wait time')

plt.subplot(1,2,1)
plt.title('Histogram')
sns.histplot(df['food_preparation_and_wait_time'], binwidth=5)

plt.subplot(1,2,2)
plt.title('Boxplot')
sns.boxplot(df['food_preparation_and_wait_time'])
sns.violinplot(df['food_preparation_and_wait_time'])
plt.show()
```



Bivariate Analysis

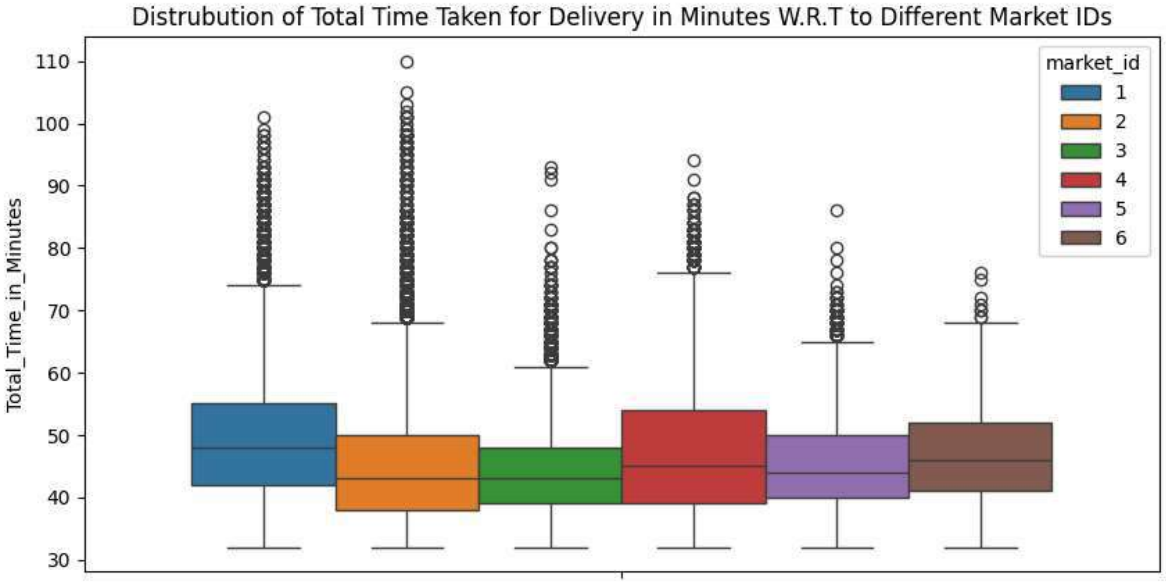
Caterocial-Numerical Feature Analysis

Box Plot Analysis

Distrubution of Total Time Taken for Delivery in Minutes W.R.T to Different Market IDs

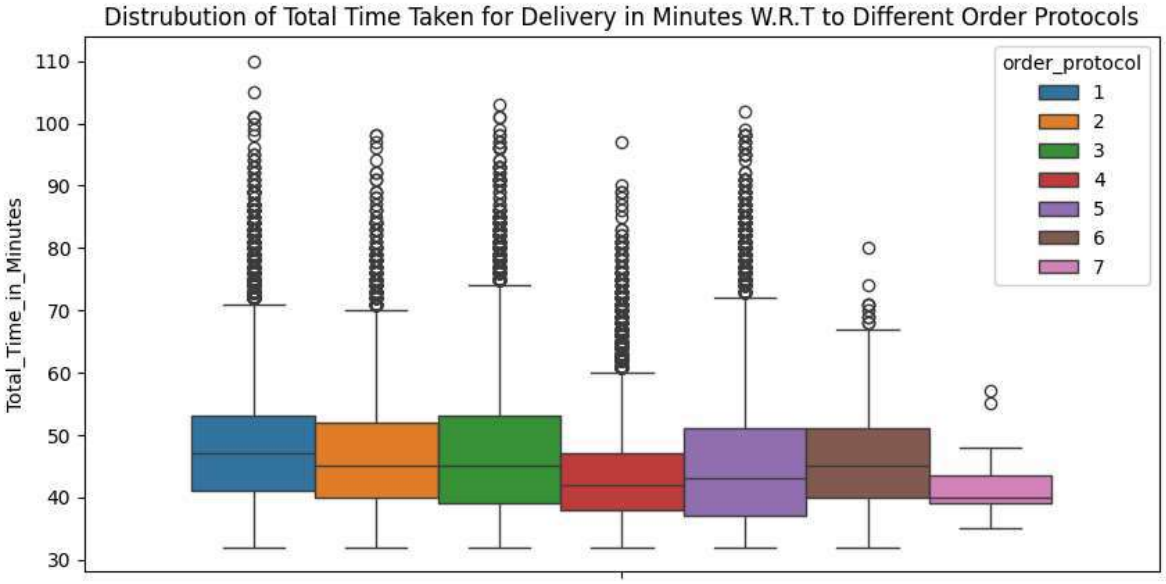
In [47]:
```
plt.figure(figsize=(10,5))
market_id_time_df = df[['market_id', 'Total_Time_in_Minutes']]
sns.boxplot(data=market_id_time_df, y='Total_Time_in_Minutes', hue='market_id')
```

```
plt.title('Distrubution of Total Time Taken for Delivery in Minutes W.R.T to Dif
plt.show()
```



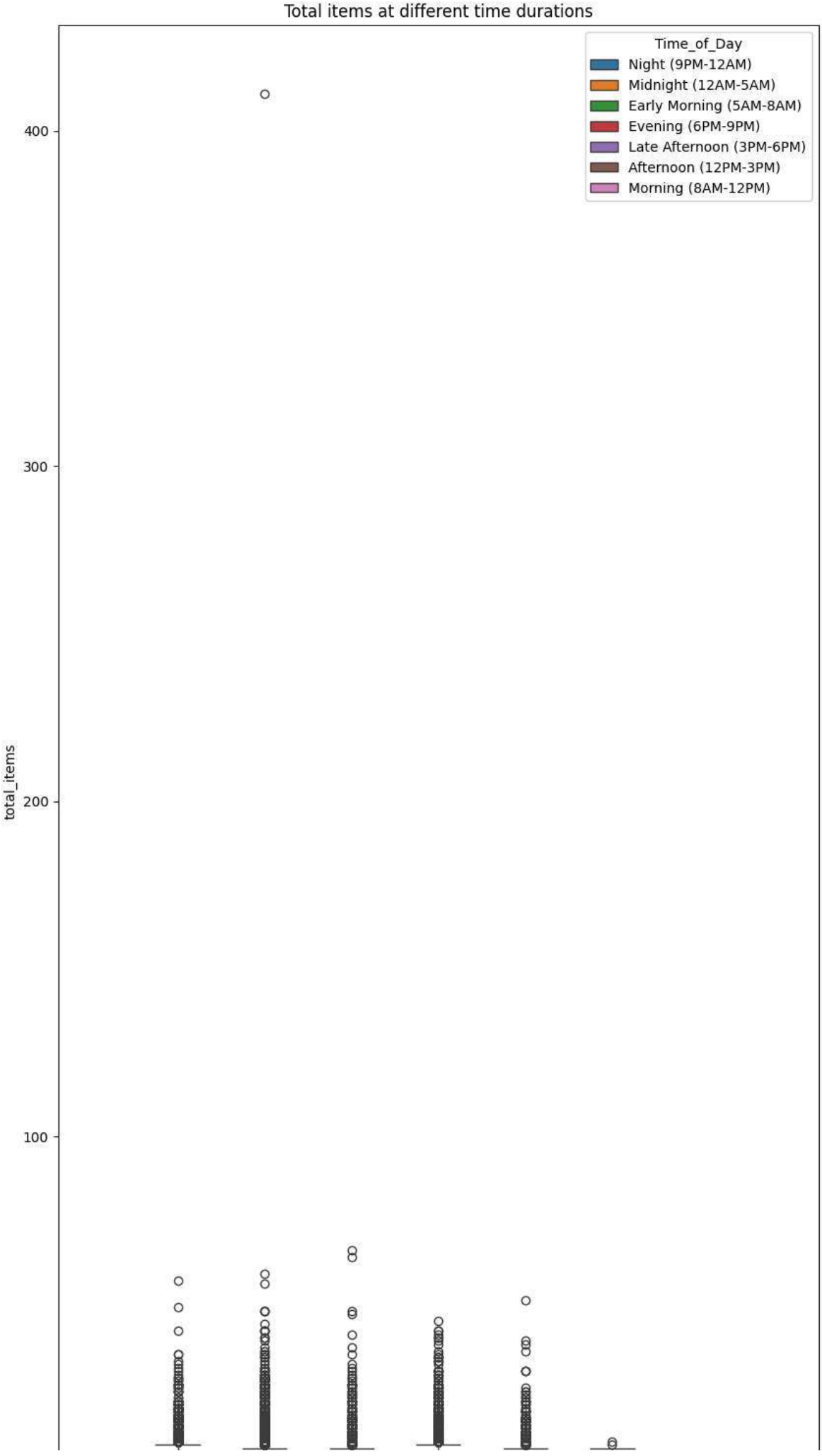
Distrubution of Total Time Taken for Delivery in Minutes W.R.T to Different Order Protocols

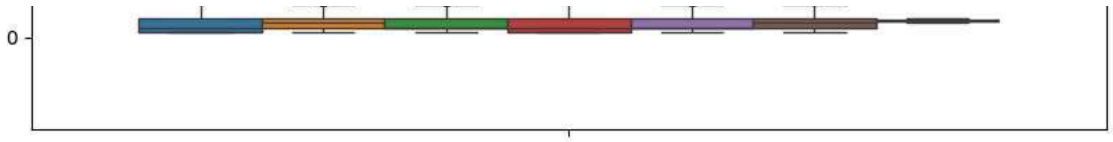
In [48]:
```
plt.figure(figsize=(10,5))
plt.title('Time taken by orders with different protocols')
st_time_df = df[['order_protocol', 'Total_Time_in_Minutes']]
sns.boxplot(data=st_time_df, y='Total_Time_in_Minutes', hue='order_protocol')
plt.title('Distrubution of Total Time Taken for Delivery in Minutes W.R.T to Dif
plt.show()
```



Distribution of Total Items at Different Time Durations

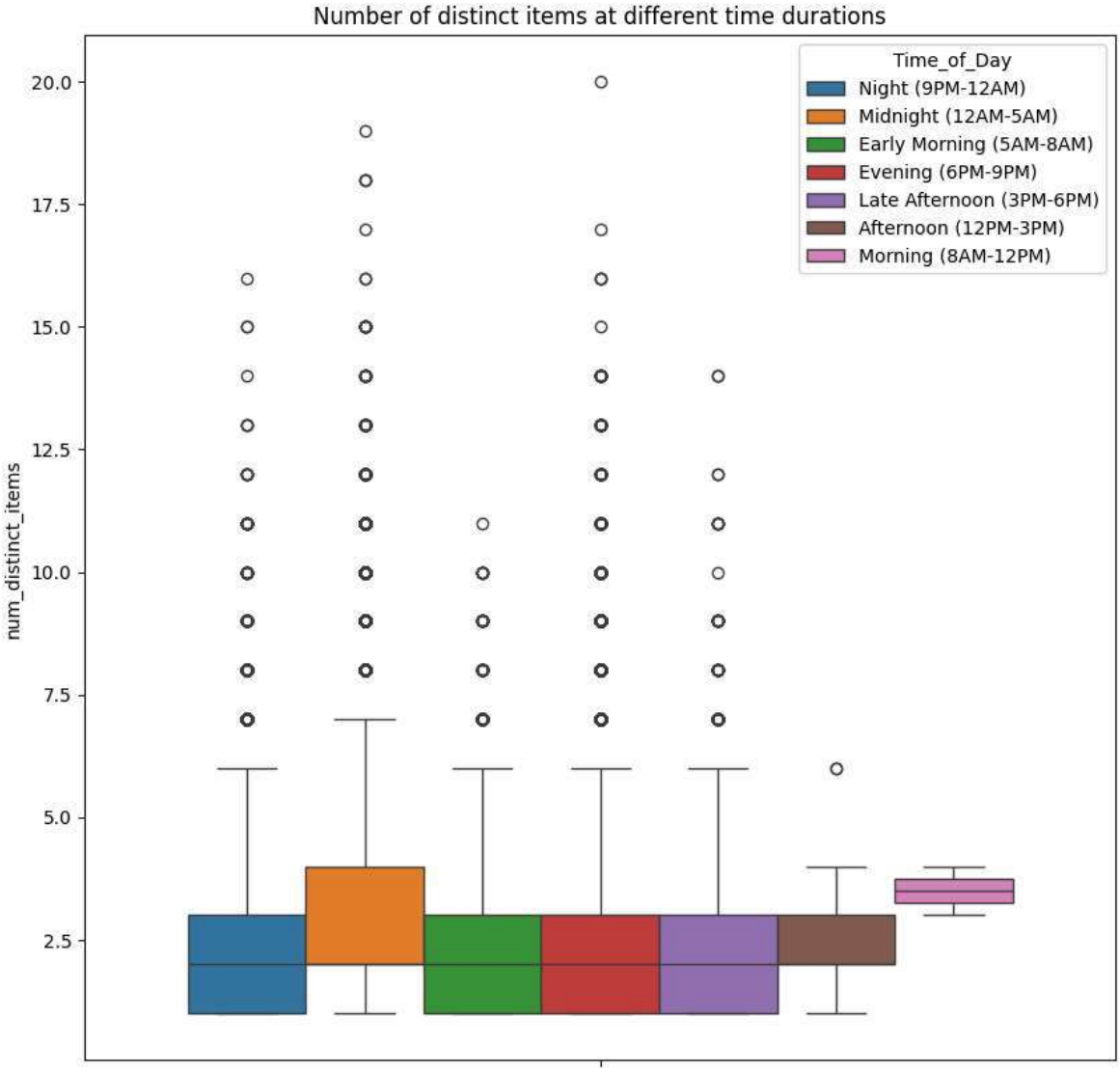
In [49]:
```
plt.figure(figsize=(10,20))
plt.title('Total items at different time durations')
total_items_time_df = df[['Time_of_Day', 'total_items']]
sns.boxplot(data=total_items_time_df, y='total_items', hue='Time_of_Day')
plt.show()
```


Total items at different time durations
Time_of_Day
Night (9PM-12AM)
Midnight (12AM-5AM)
Early Morning (5AM-8AM)
Evening (6PM-9PM)
Late Afternoon (3PM-6PM)
Afternoon (12PM-3PM)
Morning (8AM-12PM)
total_items
400
300
200
100



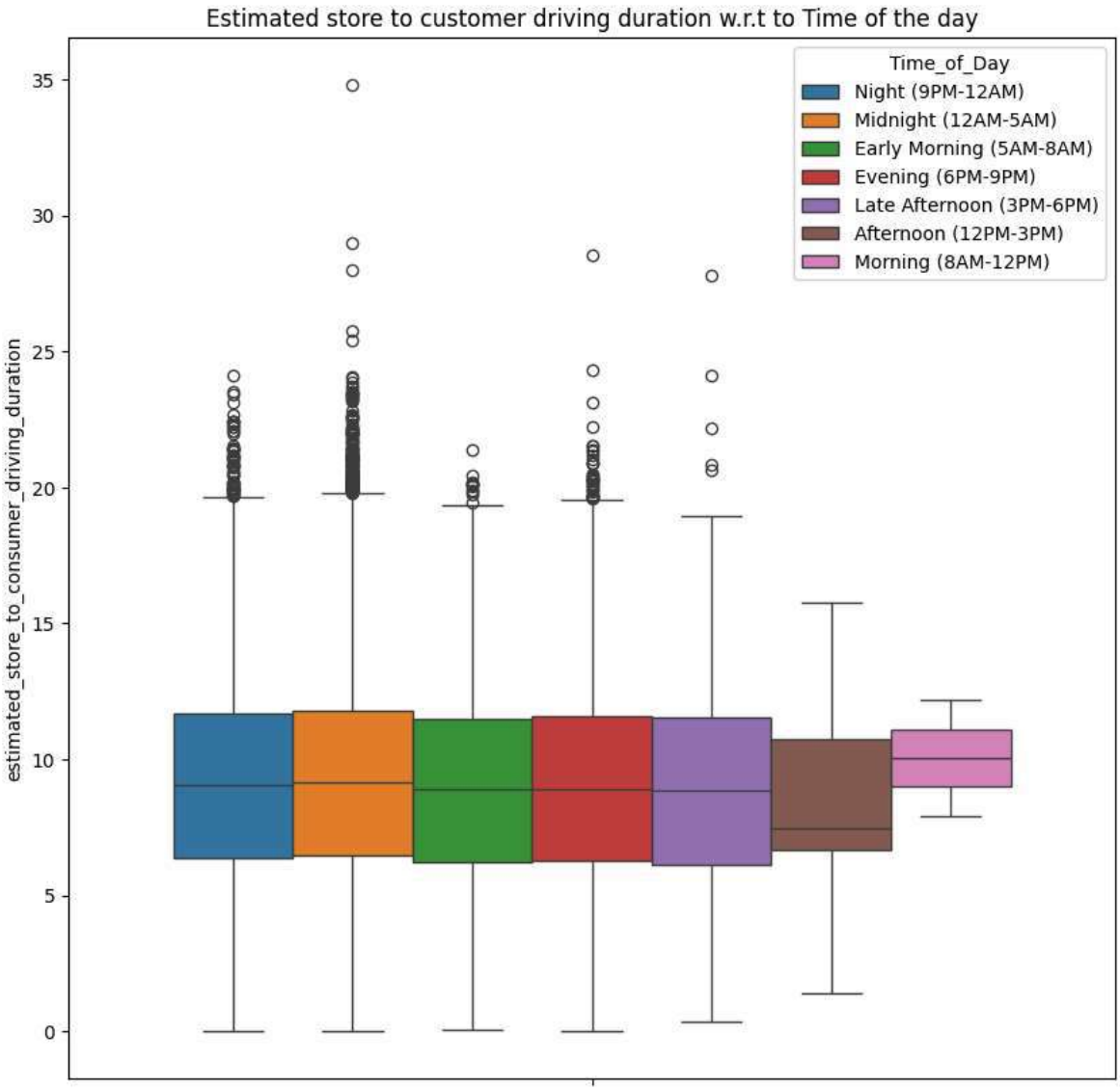
Distribution of Total Items at Different Time Frames

In [50]:
```
plt.figure(figsize=(10,10))
plt.title('Number of distinct items at different time durations')
distinct_items_time_df = df[['Time_of_Day', 'num_distinct_items']]
sns.boxplot(data=distinct_items_time_df, y='num_distinct_items', hue='Time_of_Da
plt.show()
```



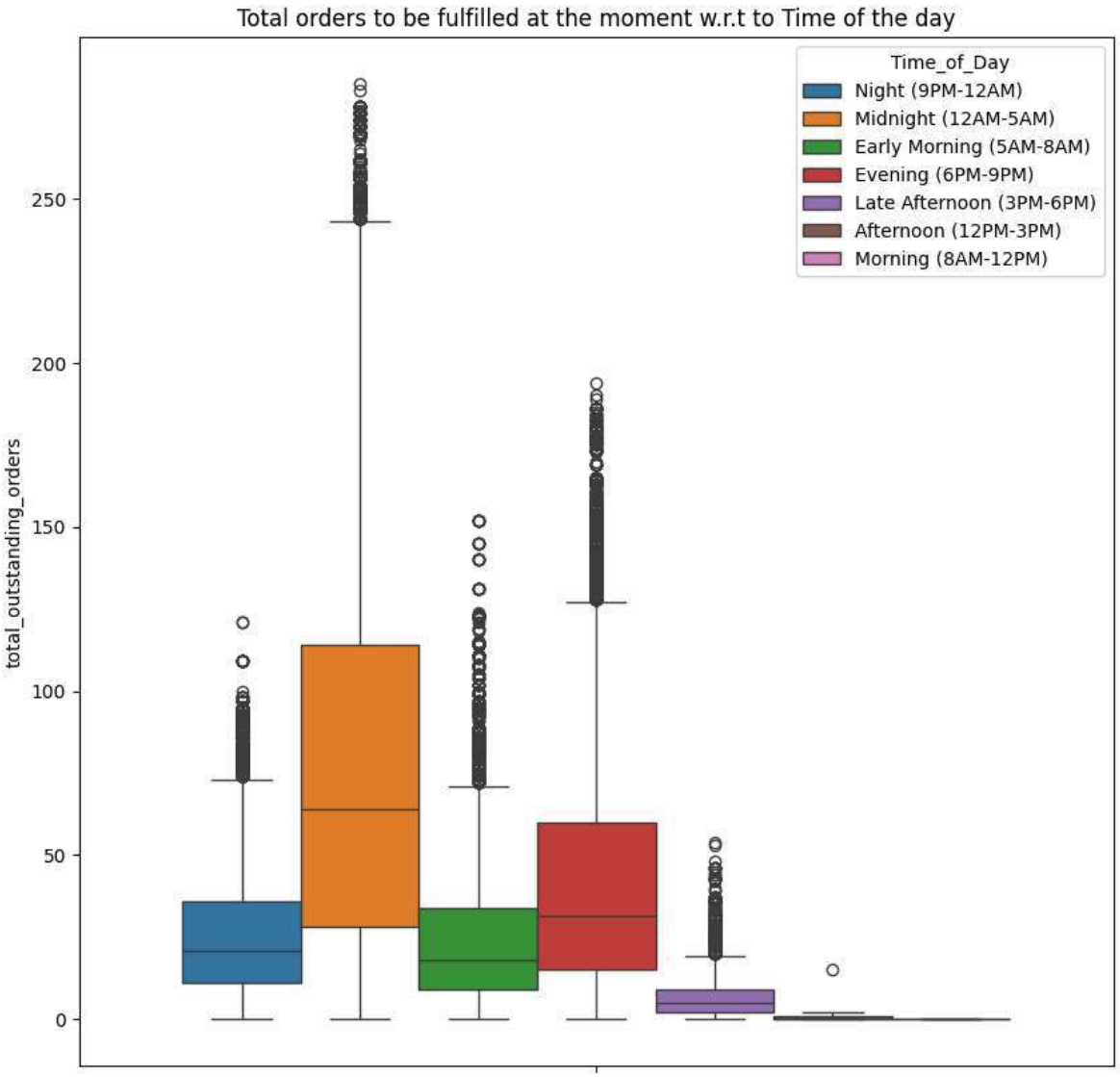
Distibution of Estimated store to customer driving duration w.r.t to Time of the day

In [51]:
```
plt.figure(figsize=(10,10))
plt.title('Estimated store to customer driving duration w.r.t to Time of the day
estcd_time_df = df[['Time_of_Day', 'estimated_store_to_consumer_driving_duration
sns.boxplot(data=estcd_time_df, y='estimated_store_to_consumer_driving_duration'
plt.show()
```



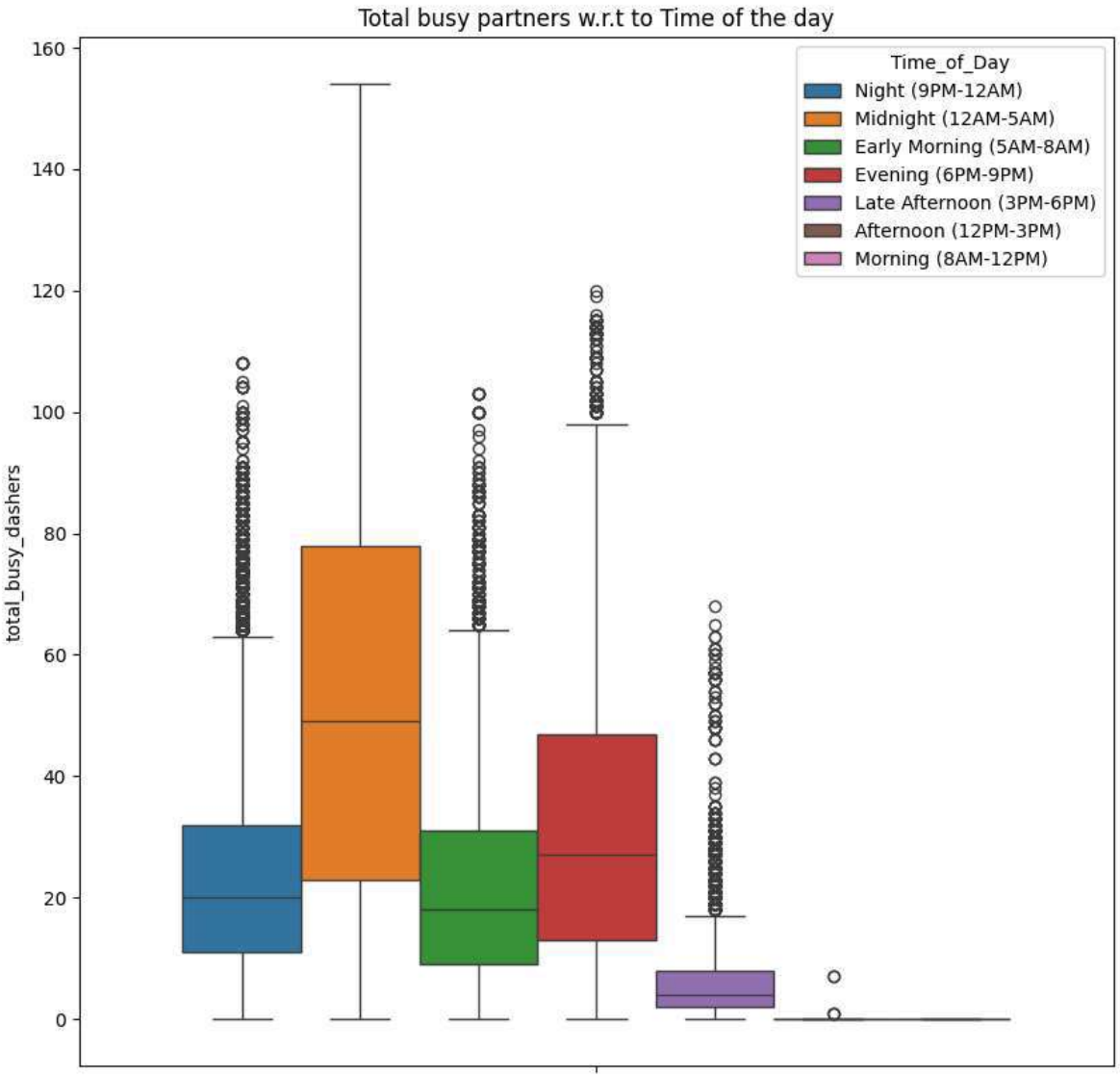
Total Orders To be Fulfilled At The Moment W.R.T to Time of The Day

In [52]:
```
plt.figure(figsize=(10,10))
plt.title('Total orders to be fulfilled at the moment w.r.t to Time of the day')
too_time_df = df[['Time_of_Day', 'total_outstanding_orders']]
sns.boxplot(data=too_time_df, y='total_outstanding_orders', hue='Time_of_Day')
plt.show()
```



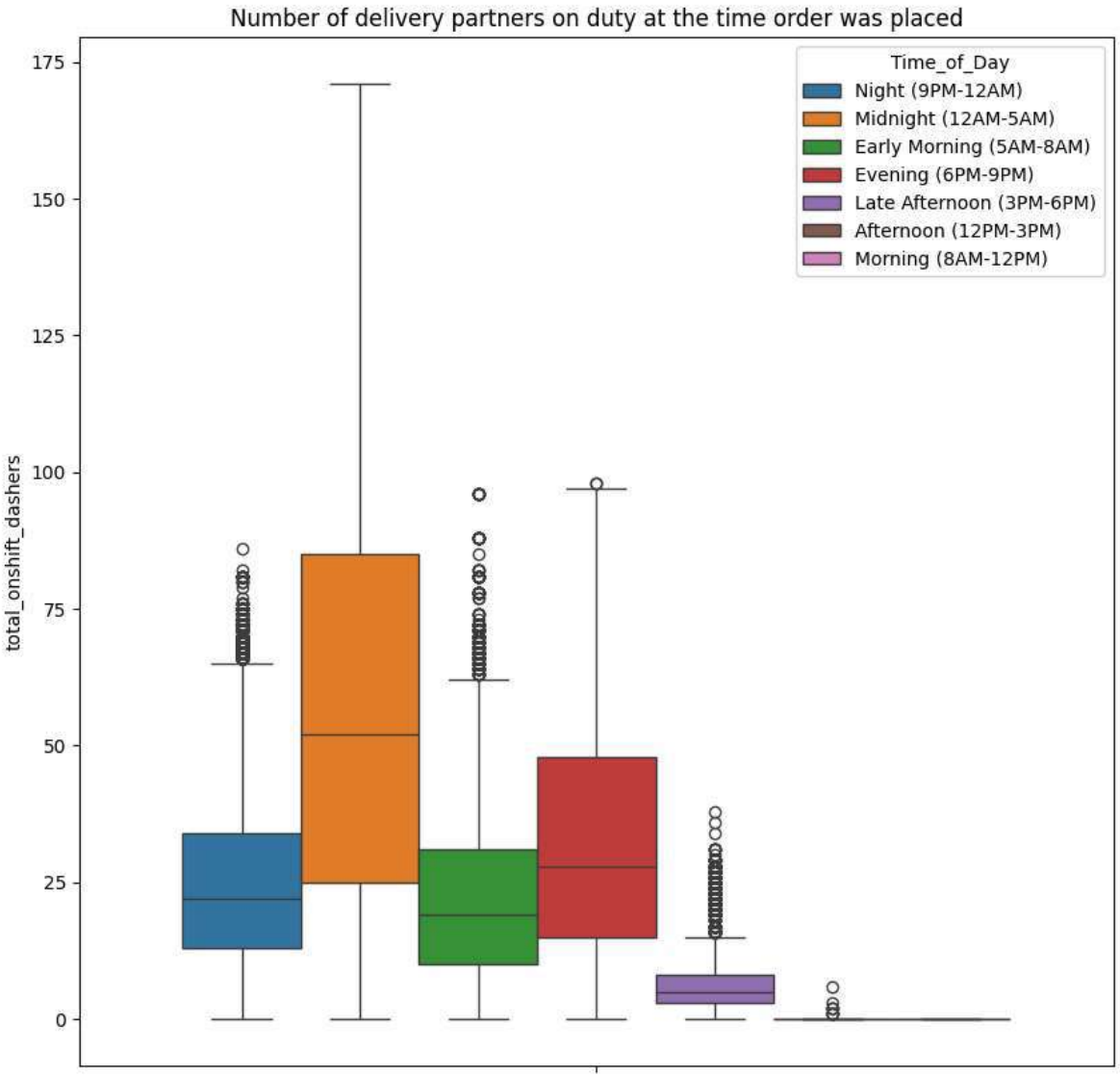
Total Busy Partners W.R.T to Time Of The Day

In [53]:
```
plt.figure(figsize=(10,10))
plt.title('Total busy partners w.r.t to Time of the day')
tbd_time_df = df[['Time_of_Day', 'total_busy_dashers']]
sns.boxplot(data=tbd_time_df, y='total_busy_dashers', hue='Time_of_Day')
plt.show()
```



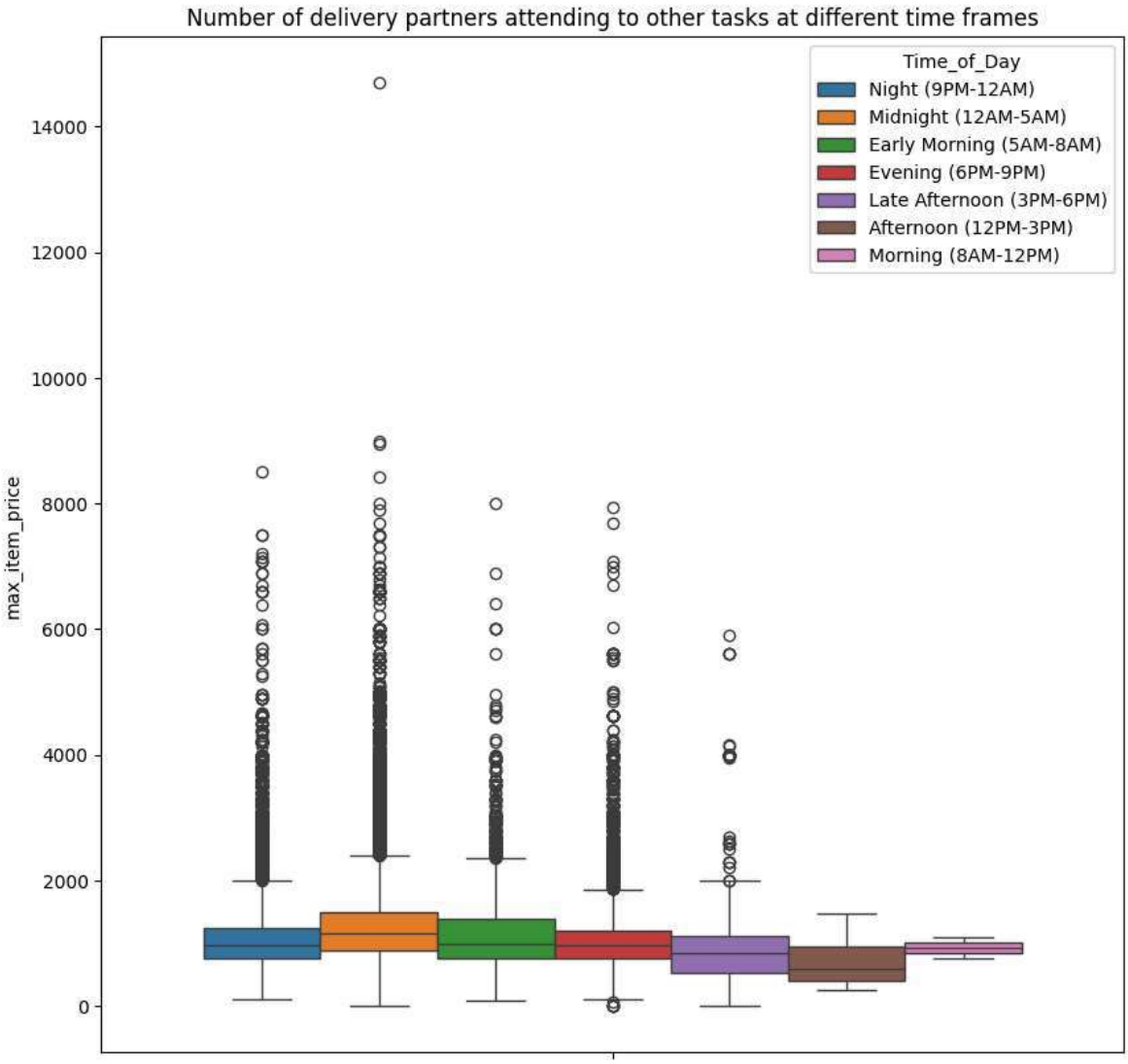
Distribution of Number of delivery partners on duty at the time order was placed

In [54]:
```
plt.figure(figsize=(10,10))
plt.title('Number of delivery partners on duty at the time order was placed')
tosd_time_df = df[['Time_of_Day', 'total_onshift_dashers']]
sns.boxplot(data=tosd_time_df, y='total_onshift_dashers', hue='Time_of_Day')
plt.show()
```



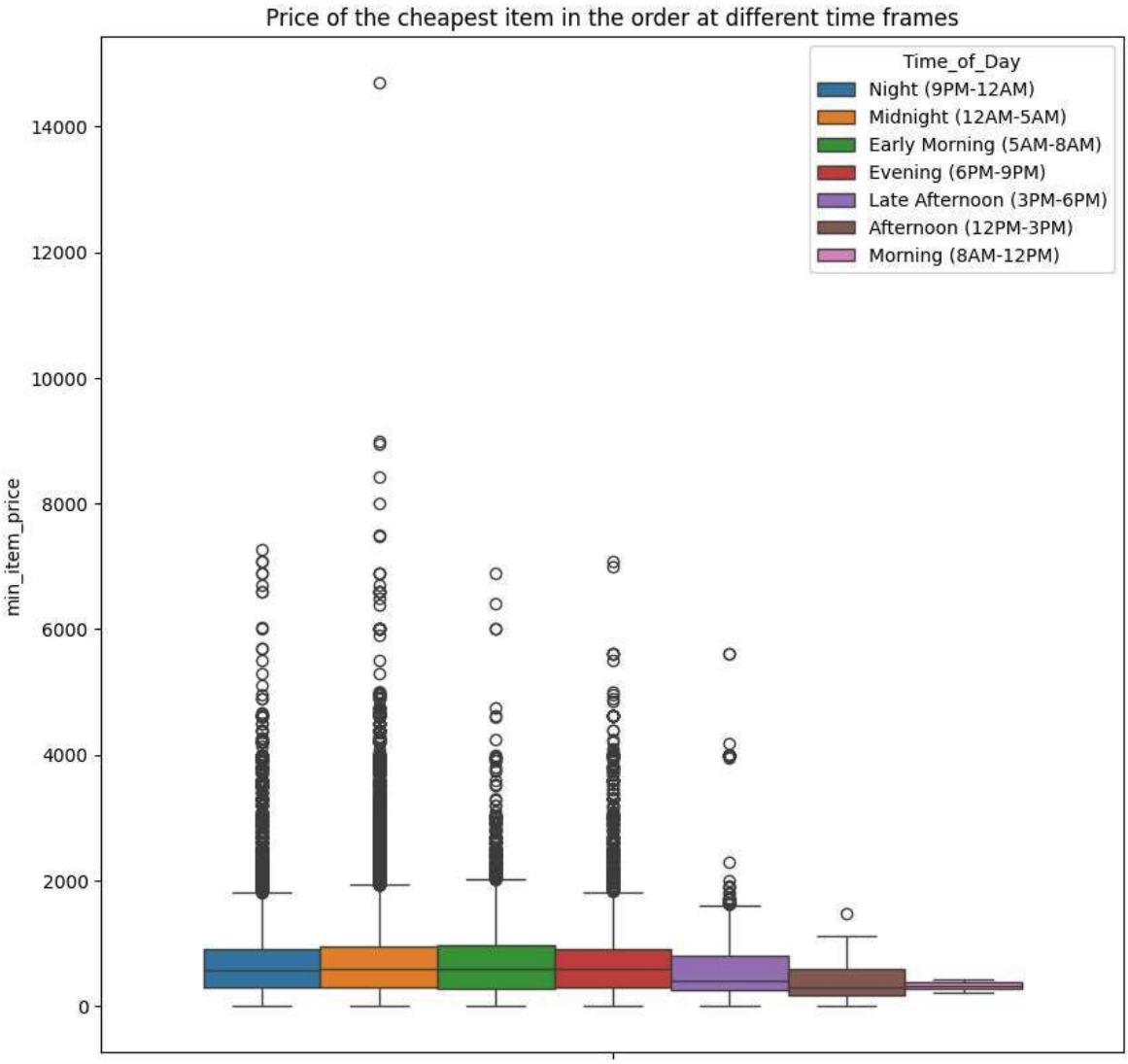
Distribution Of Number Of Delivery Partners Attending To Other Tasks At Different Time Frames

In [55]:
```
plt.figure(figsize=(10,10))
plt.title('Number of delivery partners attending to other tasks at different tim
maxp_time_df = df[['Time_of_Day', 'max_item_price']]
sns.boxplot(data=maxp_time_df, y='max_item_price', hue='Time_of_Day')
plt.show()
```



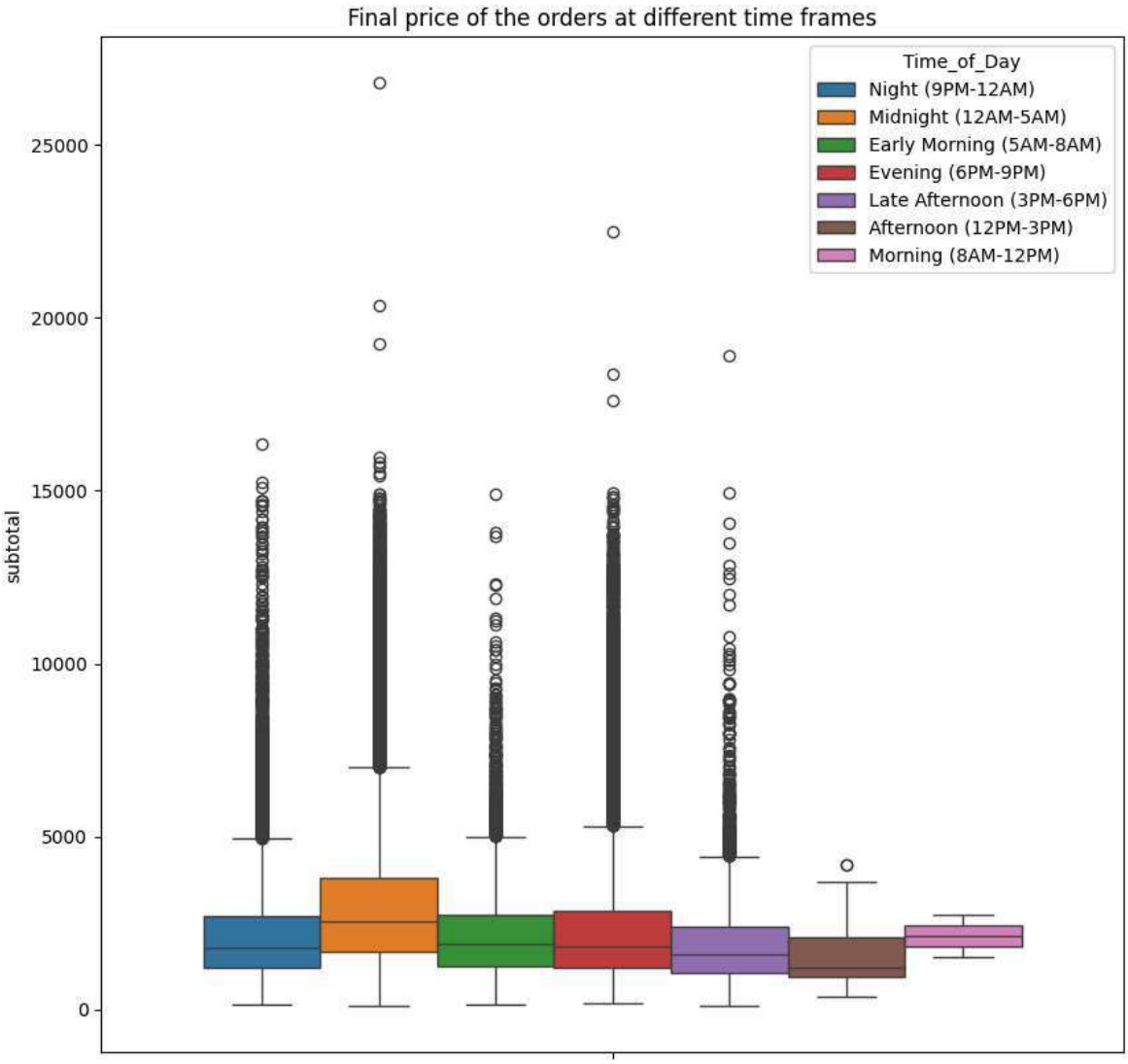
Distribution of Price Of The Cheapest Item In The Order At Different Time Frames

In [56]:
```
plt.figure(figsize=(10,10))
plt.title('Price of the cheapest item in the order at different time frames')
minp_time_df = df[['Time_of_Day', 'min_item_price']]
sns.boxplot(data=minp_time_df, y='min_item_price', hue='Time_of_Day')
plt.show()
```



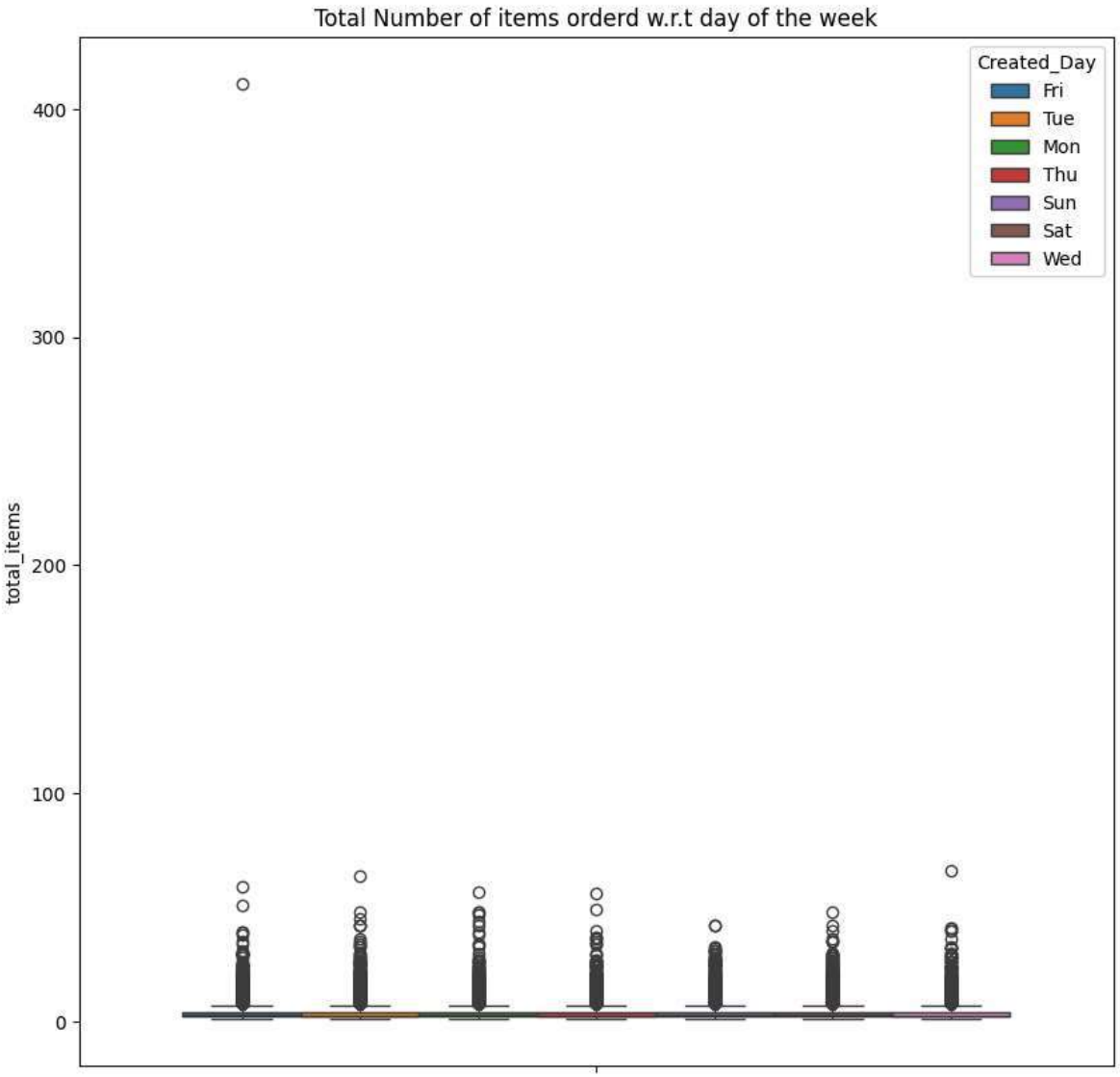
Distribution Of Final Price Of The Orders At Different Time Frames

In [57]:
```
plt.figure(figsize=(10,10))
plt.title('Final price of the orders at different time frames')
st_time_df = df[['Time_of_Day', 'subtotal']]
sns.boxplot(data=st_time_df, y='subtotal', hue='Time_of_Day')
plt.show()
```



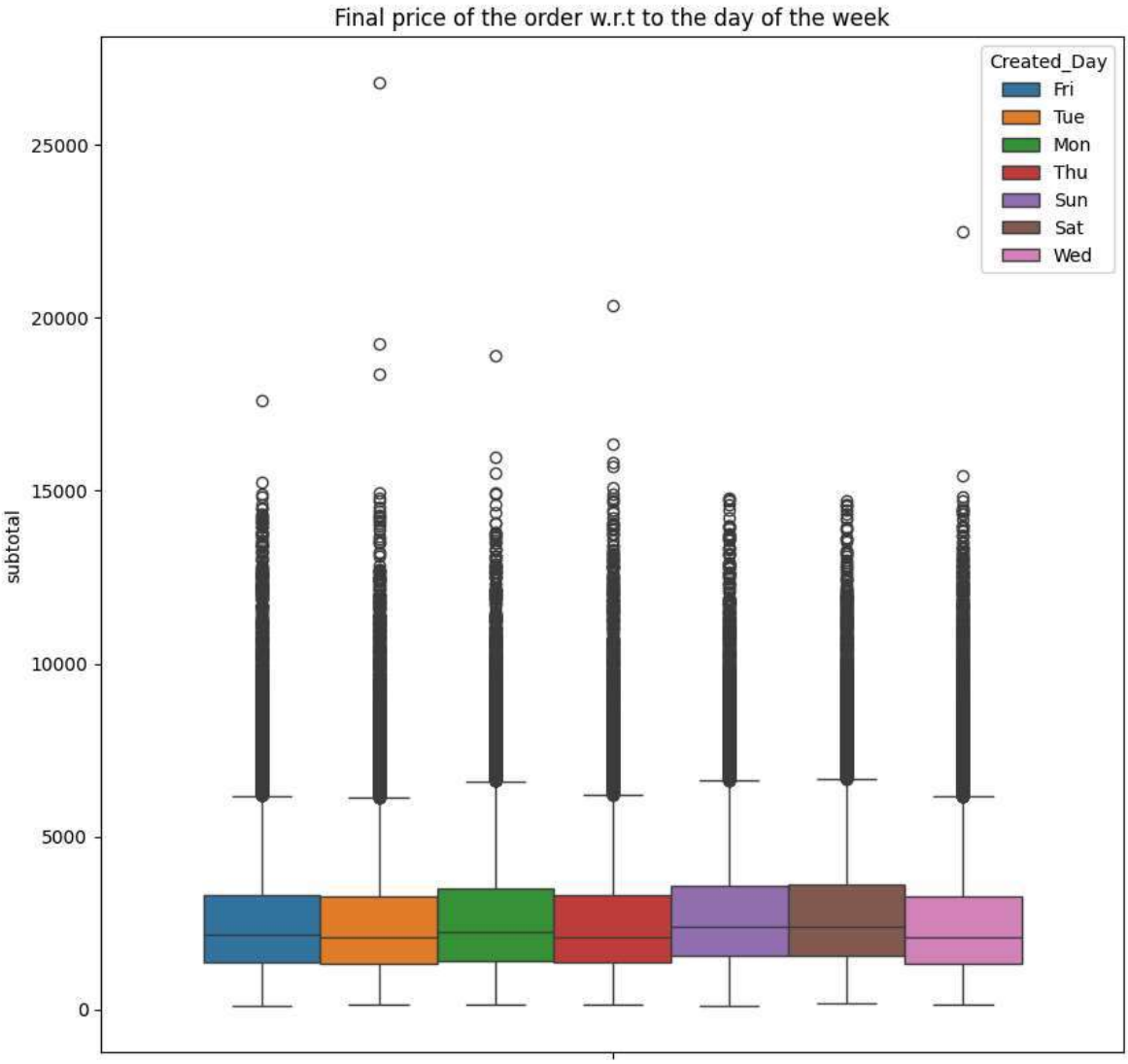
Total Number Of Items Orderd W.R.T Day Of The Week

In [58]:
```
plt.figure(figsize=(10,10))
plt.title('Total Number of items orderd w.r.t day of the week')
st_time_df = df[['Created_Day', 'total_items']]
sns.boxplot(data=st_time_df, y='total_items', hue='Created_Day')
plt.show()
```



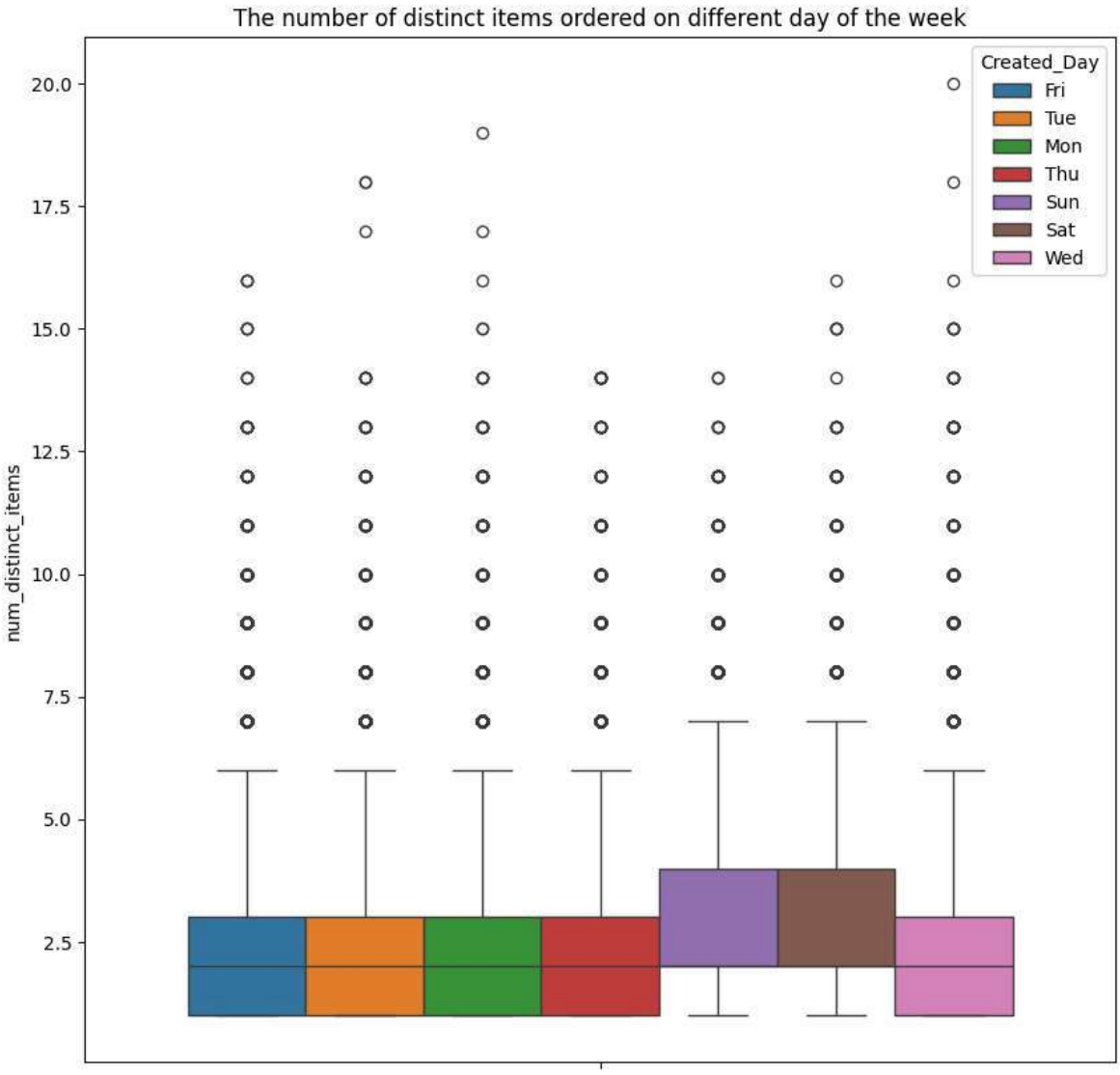
Distribution Of Final Price Of The Order W.R.T The Day Of The Week

In [59]:
```
plt.figure(figsize=(10,10))
plt.title('Final price of the order w.r.t to the day of the week')
st_time_df = df[['Created_Day', 'subtotal']]
sns.boxplot(data=st_time_df, y='subtotal', hue='Created_Day')
plt.show()
```



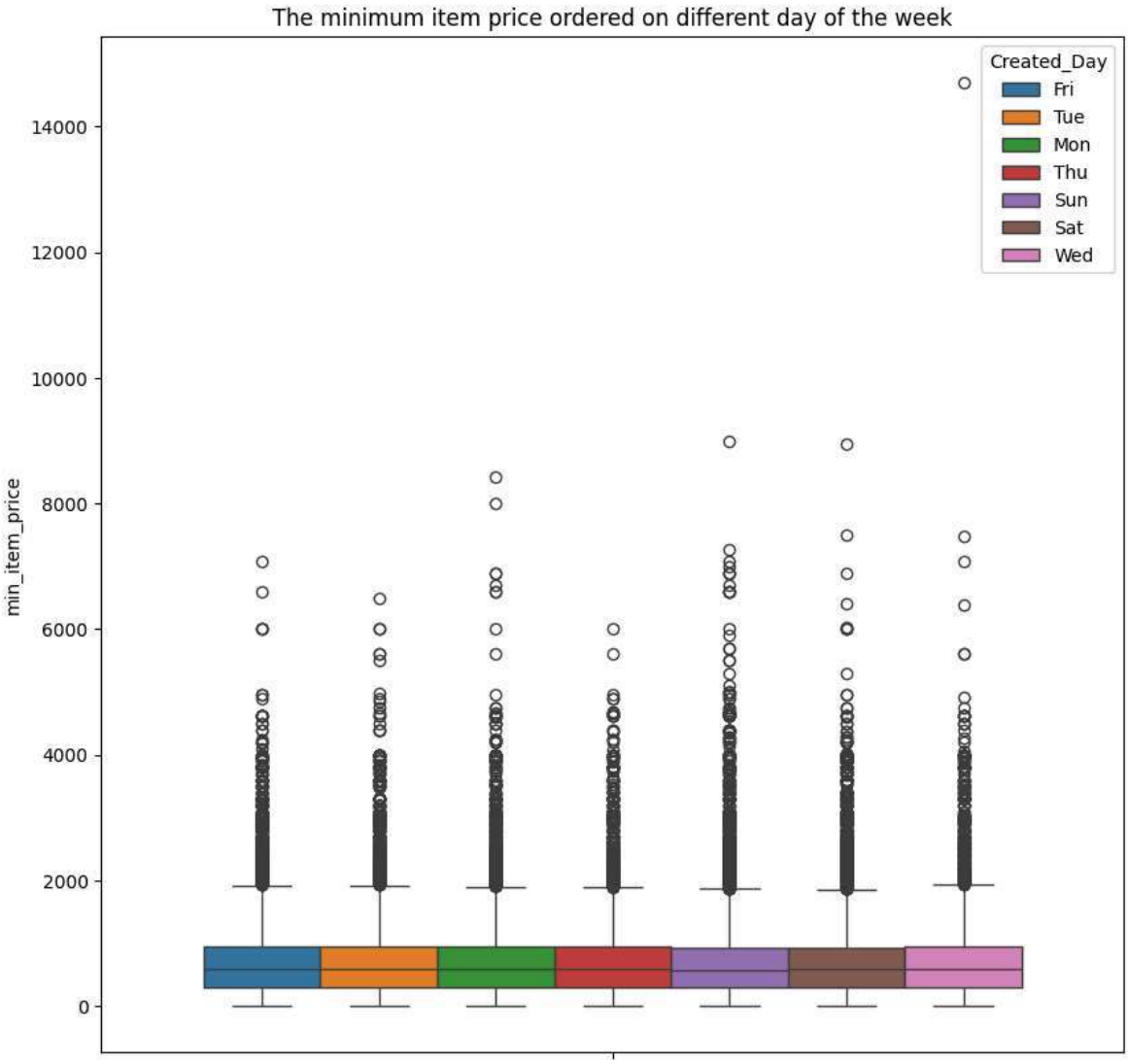
Distribution of The number of distinct items ordered on different day of the week

In [60]:
```
plt.figure(figsize=(10,10))
plt.title('The number of distinct items ordered on different day of the week')
st_time_df = df[['Created_Day', 'num_distinct_items']]
sns.boxplot(data=st_time_df, y='num_distinct_items', hue='Created_Day')
plt.show()
```



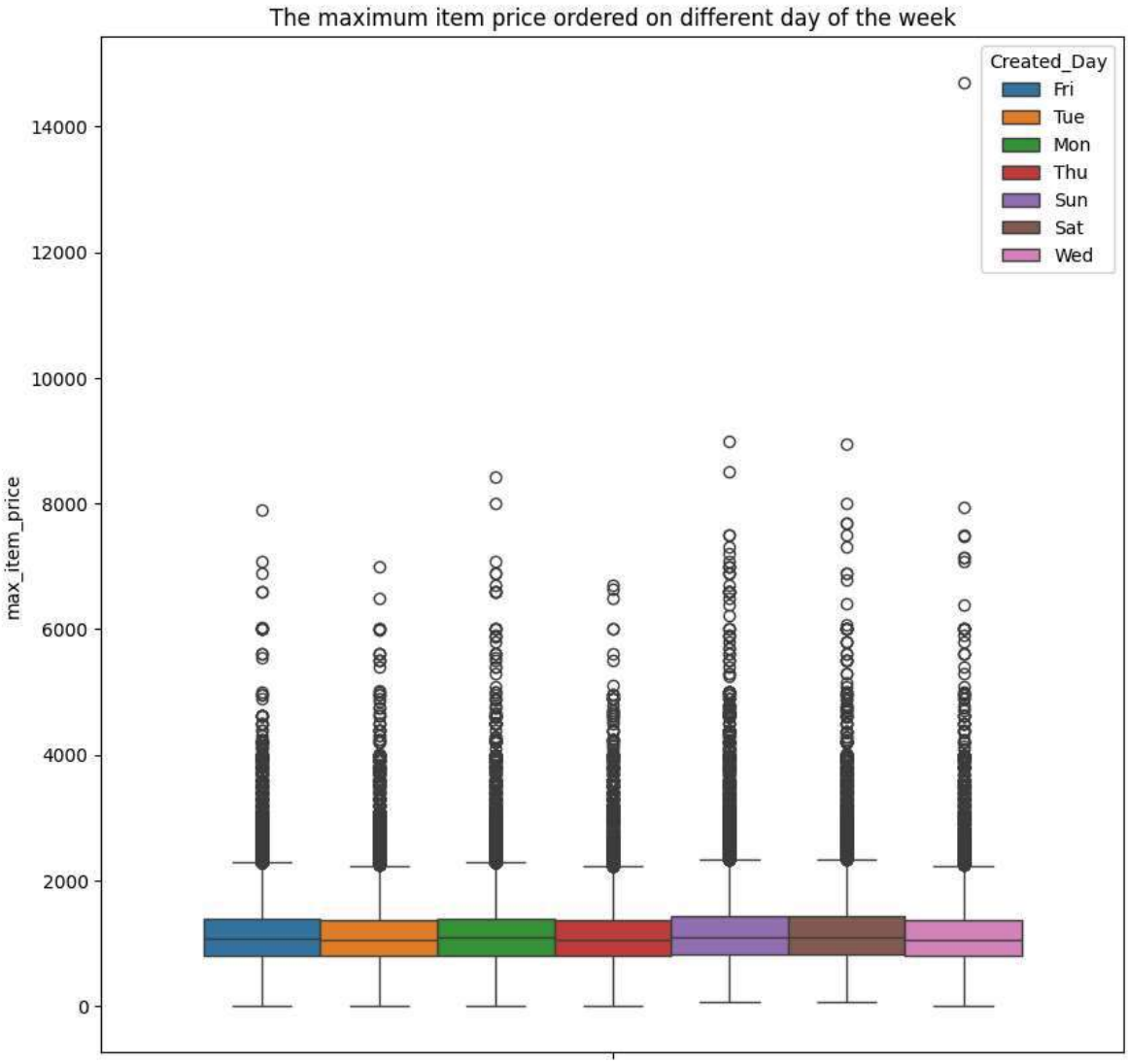
Distribution Of The Minimum Item Price Ordered On Different Day Of The Week

In [61]:
```
plt.figure(figsize=(10,10))
plt.title('The minimum item price ordered on different day of the week')
st_time_df = df[['Created_Day', 'min_item_price']]
sns.boxplot(data=st_time_df, y='min_item_price', hue='Created_Day')
plt.show()
```



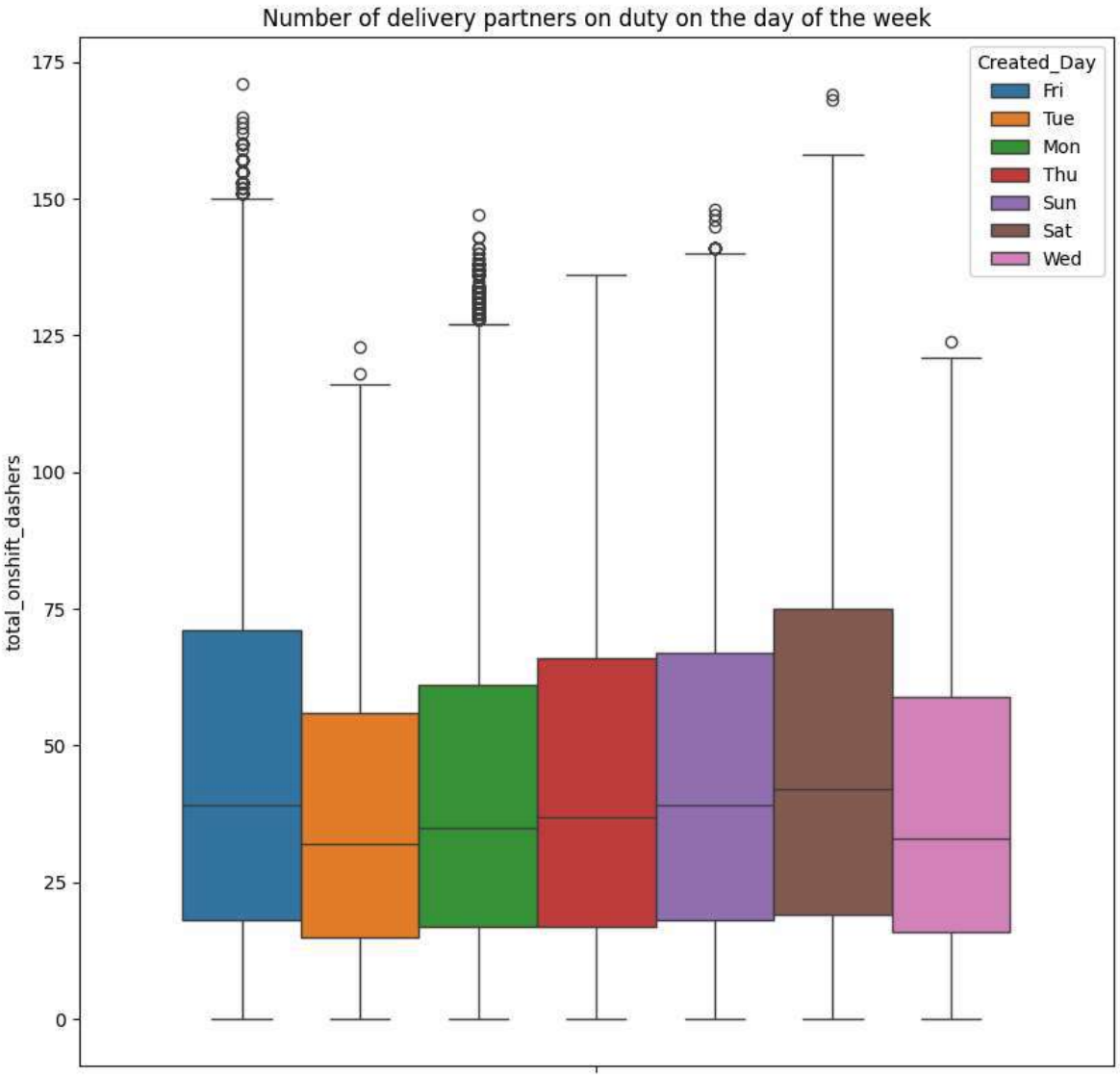
Distribution Of The Maxmium Item Price Ordered On Different Day Of The Week

In [62]:
```
plt.figure(figsize=(10,10))
plt.title('The maximum item price ordered on different day of the week')
st_time_df = df[['Created_Day', 'max_item_price']]
sns.boxplot(data=st_time_df, y='max_item_price', hue='Created_Day')
plt.show()
```



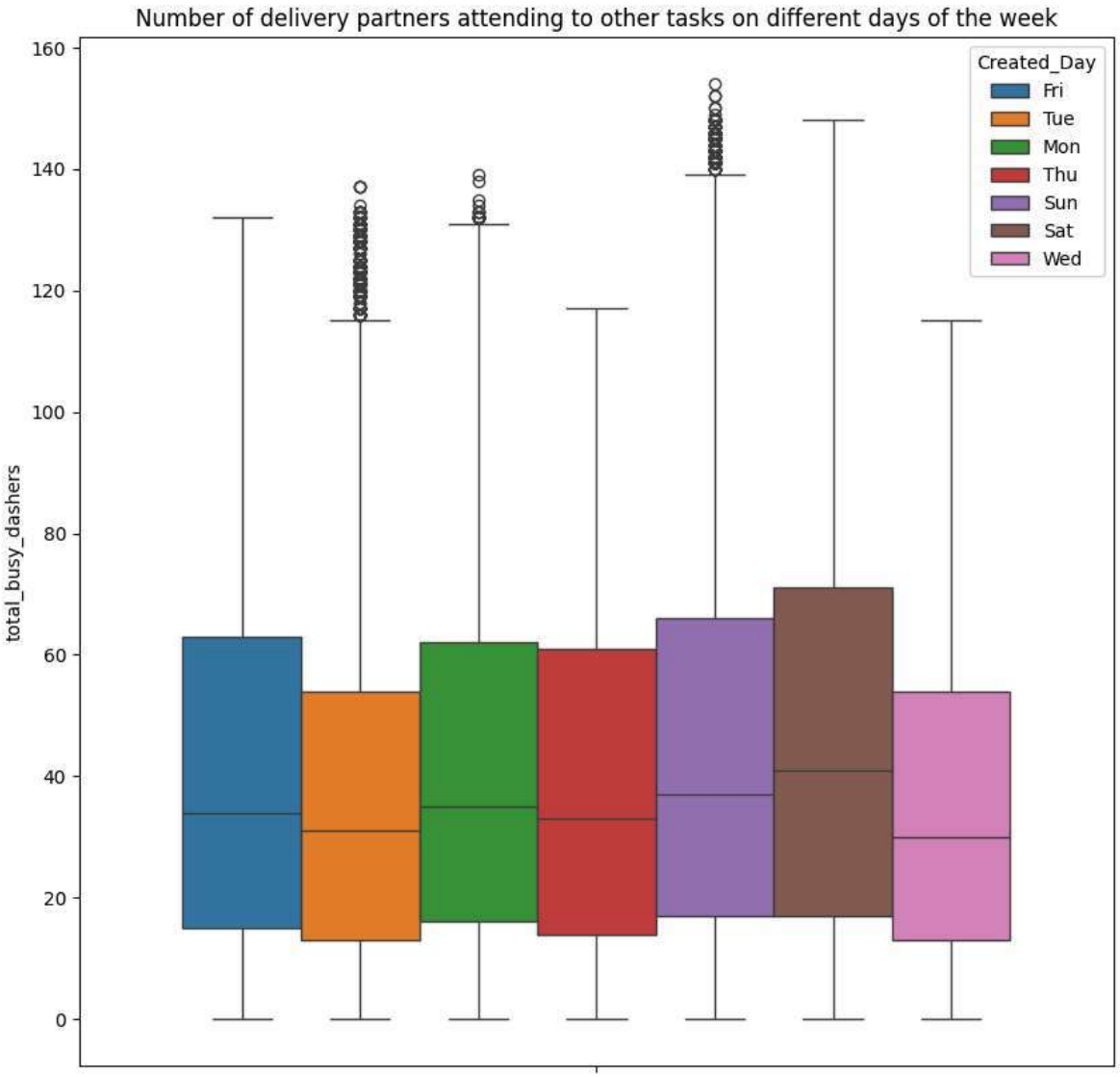
Distibution Of Number Of Delivery Partners On Duty On The Day Of The Week

In [63]:
```
plt.figure(figsize=(10,10))
plt.title('Number of delivery partners on duty on the day of the week')
st_time_df = df[['Created_Day', 'total_onshift_dashers']]
sns.boxplot(data=st_time_df, y='total_onshift_dashers', hue='Created_Day')
plt.show()
```



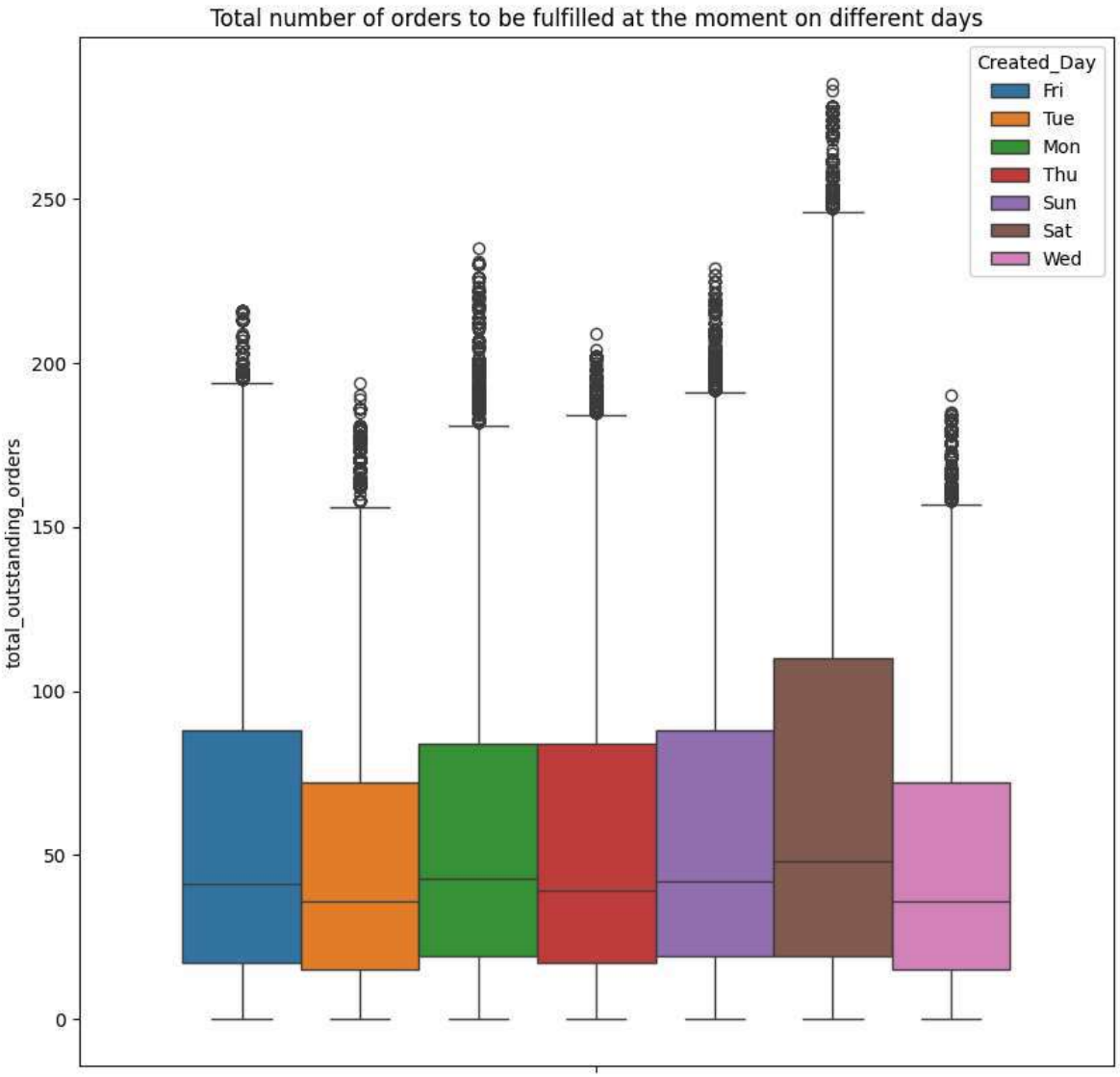
Distibution Of Number Of Delivery Partners Attending To Other Tasks On Different Days Of The Week

In [64]:
```
plt.figure(figsize=(10,10))
plt.title('Number of delivery partners attending to other tasks on different day
st_time_df = df[['Created_Day', 'total_busy_dashers']]
sns.boxplot(data=st_time_df, y='total_busy_dashers', hue='Created_Day')
plt.show()
```



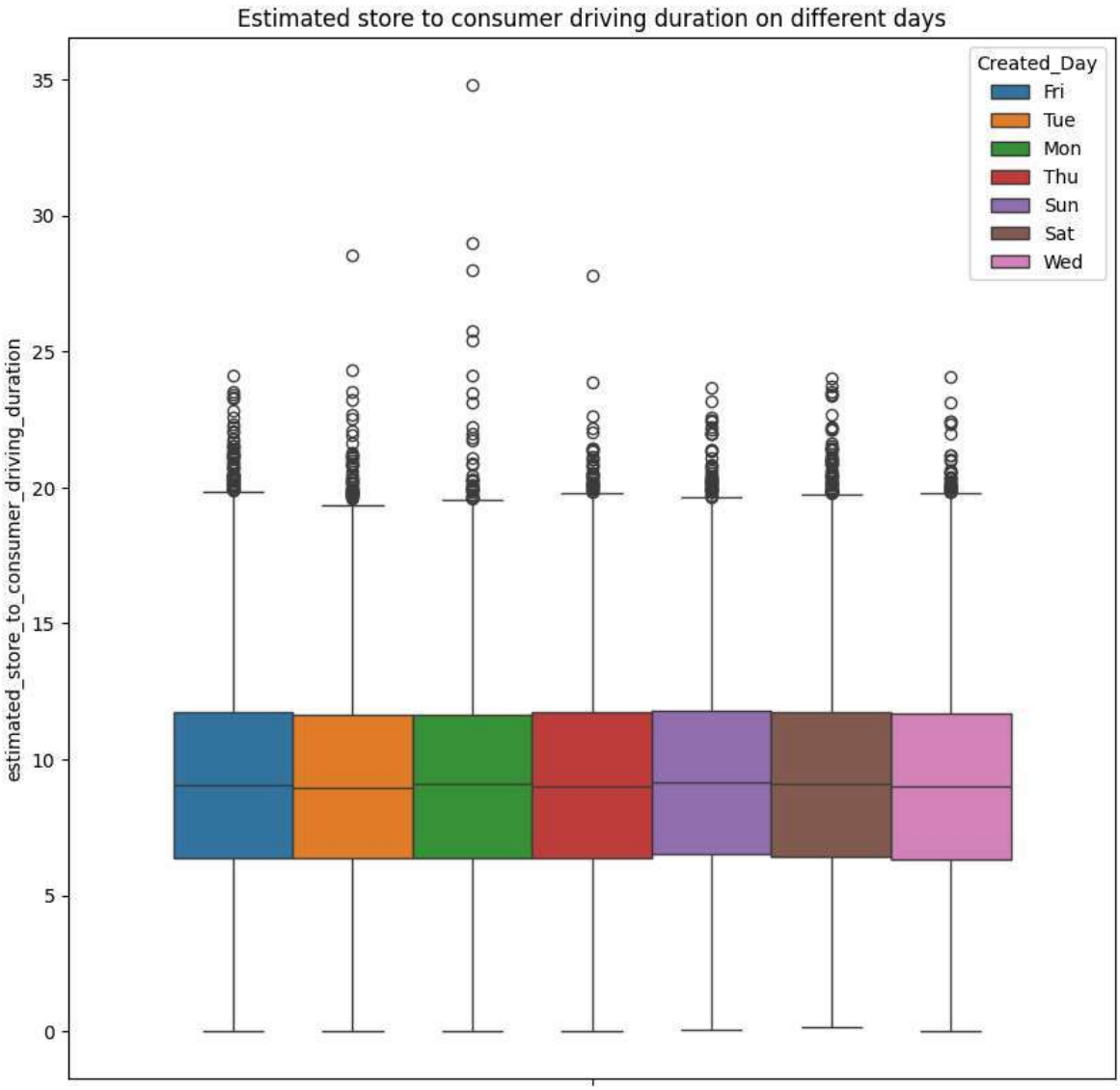
Distribution Of Total Number Of Orders To Be Fulfilled At The Moment On Different Days

In [65]:
```
plt.figure(figsize=(10,10))
plt.title('Total number of orders to be fulfilled at the moment on different day
st_time_df = df[['Created_Day', 'total_outstanding_orders']]
sns.boxplot(data=st_time_df, y='total_outstanding_orders', hue='Created_Day')
plt.show()
```



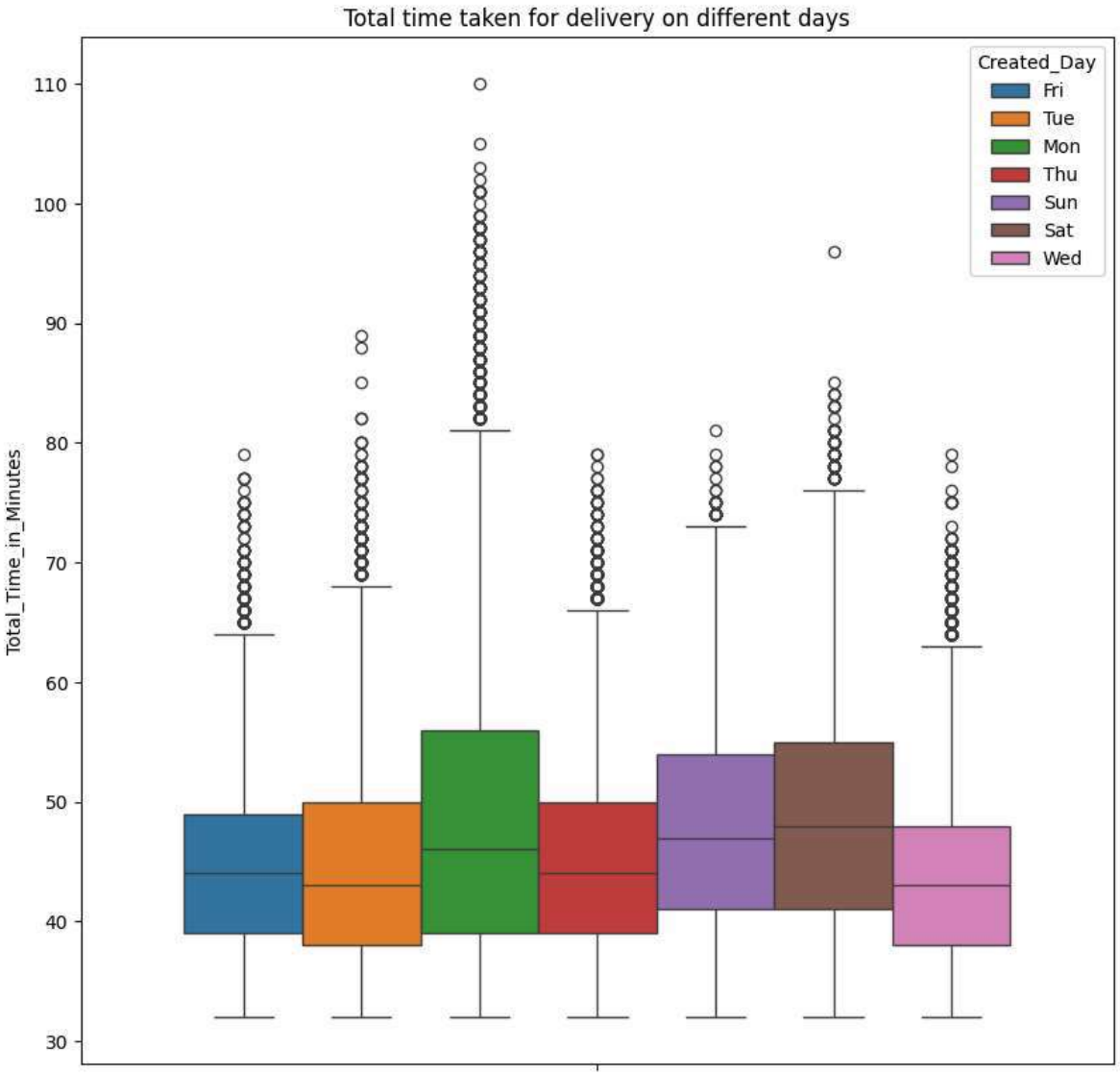
Distribution Of Estimated Store To Consumer Driving Duration On Different Days

In [66]:
```
plt.figure(figsize=(10,10))
plt.title('Estimated store to consumer driving duration on different days')
st_time_df = df[['Created_Day', 'estimated_store_to_consumer_driving_duration']]
sns.boxplot(data=st_time_df, y='estimated_store_to_consumer_driving_duration', h
plt.show()
```



Distibution Of Total Time Taken For Delivery On Different Days

In [67]:
```
plt.figure(figsize=(10,10))
plt.title('Total time taken for delivery on different days')
st_time_df = df[['Created_Day', 'Total_Time_in_Minutes']]
sns.boxplot(data=st_time_df, y='Total_Time_in_Minutes', hue='Created_Day')
plt.show()
```

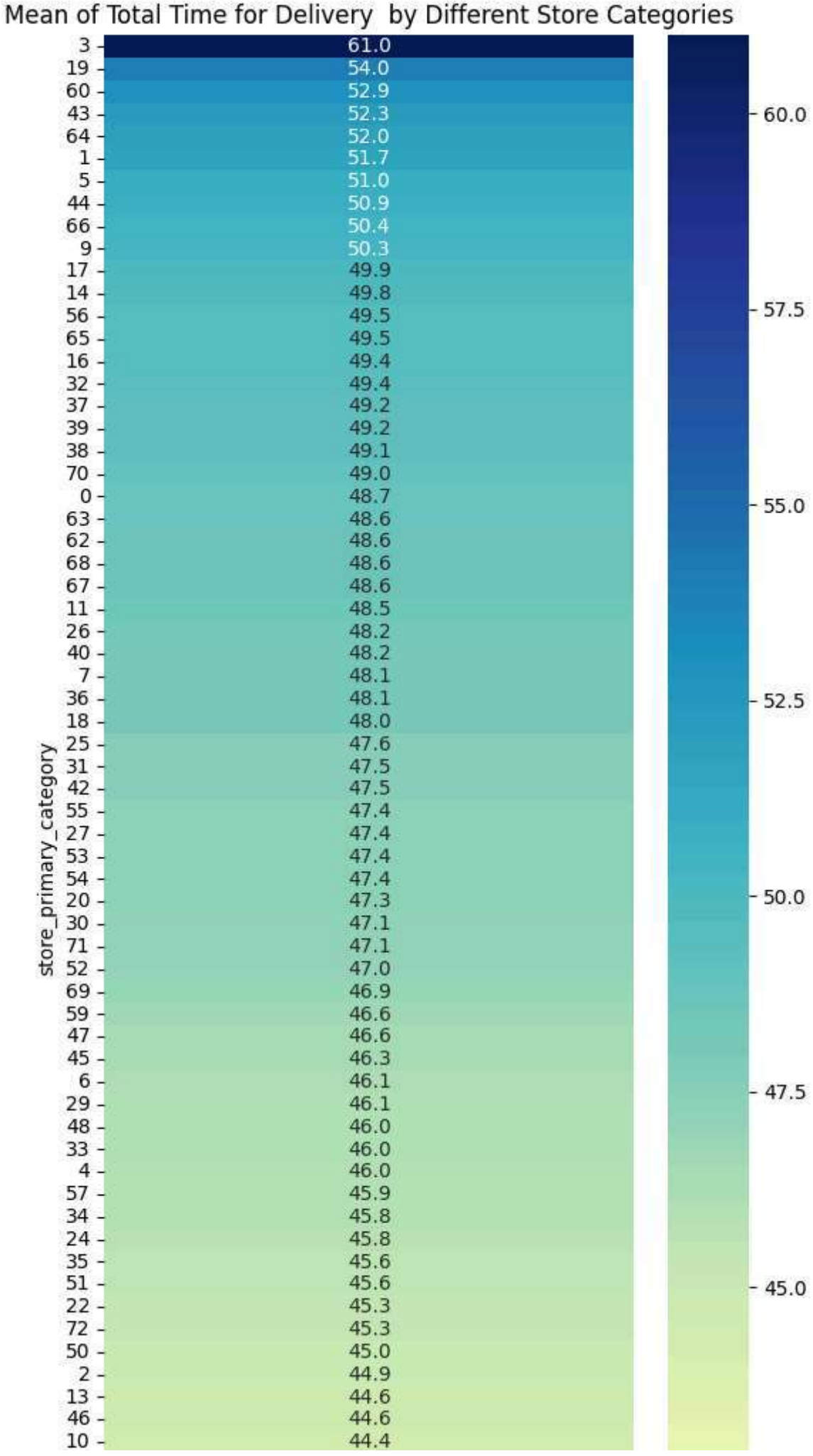


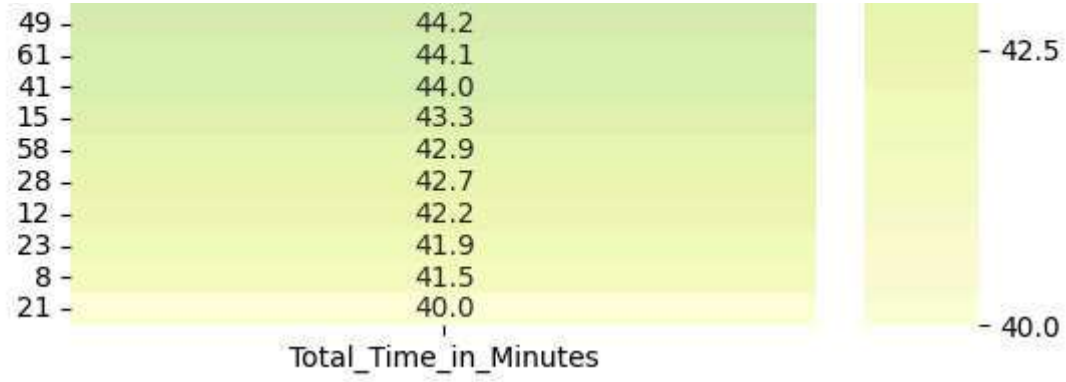
HeatMap

Mean of Total Time for Delivery by Different Store Categories

In [68]:
```
summary = df.groupby('store_primary_category')['Total_Time_in_Minutes'].mean().r
summary = summary.pivot_table(index='store_primary_category', values='Total_Time

plt.figure(figsize=(6, 15))
sns.heatmap(summary.sort_values(by='Total_Time_in_Minutes', ascending=False), an
plt.title('Mean of Total Time for Delivery  by Different Store Categories')
plt.show()
```

```
/tmp/ipython-input-68-3666839962.py:1: FutureWarning: The default of observed=Fal
se is deprecated and will be changed to True in a future version of pandas. Pass
observed=False to retain current behavior or observed=True to adopt the future de
fault and silence this warning.
  summary = df.groupby('store_primary_category')['Total_Time_in_Minutes'].mean().
reset_index()
/tmp/ipython-input-68-3666839962.py:2: FutureWarning: The default value of observ
ed=False is deprecated and will change to observed=True in a future version of pa
ndas. Specify observed=False to silence this warning and retain the current behav
ior
  summary = summary.pivot_table(index='store_primary_category', values='Total_Tim
e_in_Minutes')
```

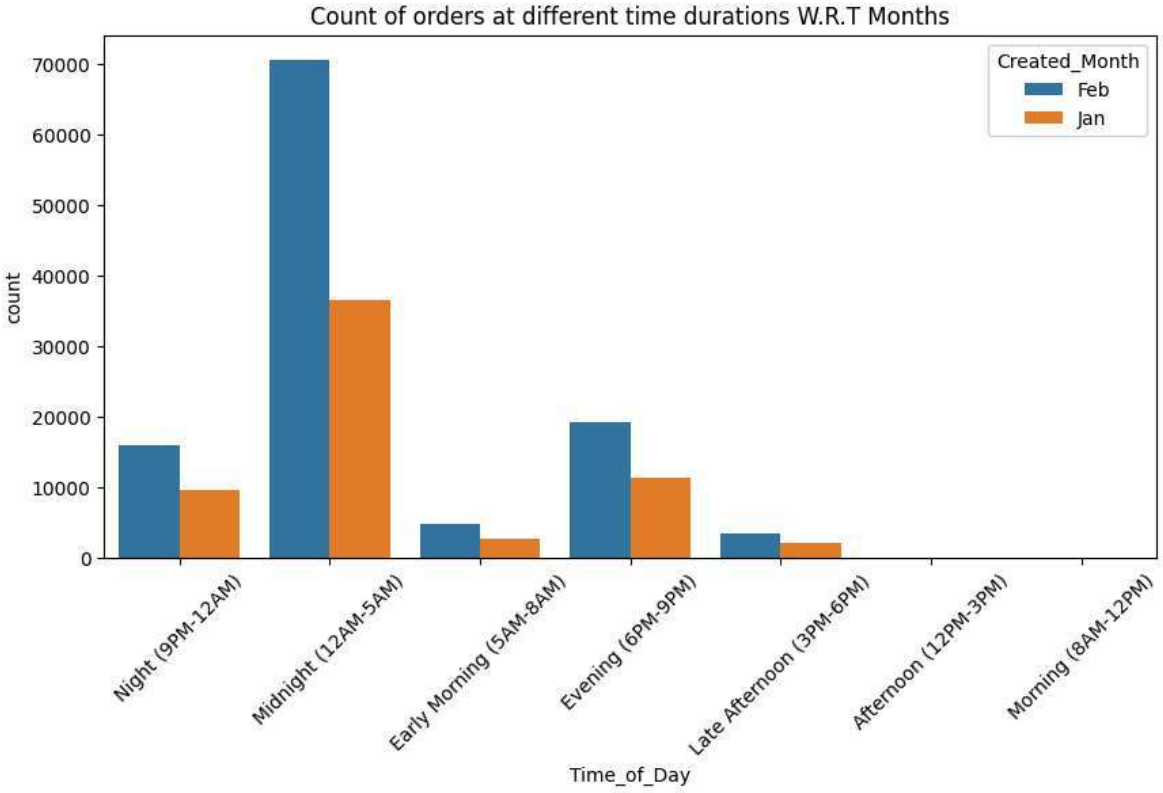

Mean of Total Time for Delivery by Different Store Categories
store_primary_category
3 61.0
19 54.0
60 52.9
43 52.3
64 52.0
1 51.7
5 51.0
44 50.9
66 50.4
9 50.3
17 49.9
14 49.8
56 49.5
65 49.5
16 49.4
32 49.4
37 49.2
39 49.2
38 49.1
70 49.0
0 48.7
63 48.6
62 48.6
68 48.6
67 48.6
11 48.5
26 48.2
40 48.2
7 48.1
36 48.1
18 48.0
25 47.6
31 47.5
42 47.5
55 47.4
27 47.4
53 47.4
54 47.4
20 47.3
30 47.1
71 47.1
52 47.0
69 46.9
59 46.6
47 46.6
45 46.3
6 46.1
29 46.1
48 46.0
33 46.0
4 46.0
57 45.9
34 45.8
24 45.8
35 45.6
51 45.6
22 45.3
72 45.3
50 45.0
2 44.9
13 44.6
46 44.6
10 44.4
60.0
57.5
55.0
52.5
50.0
47.5
45.0



Dodged Bar Plot

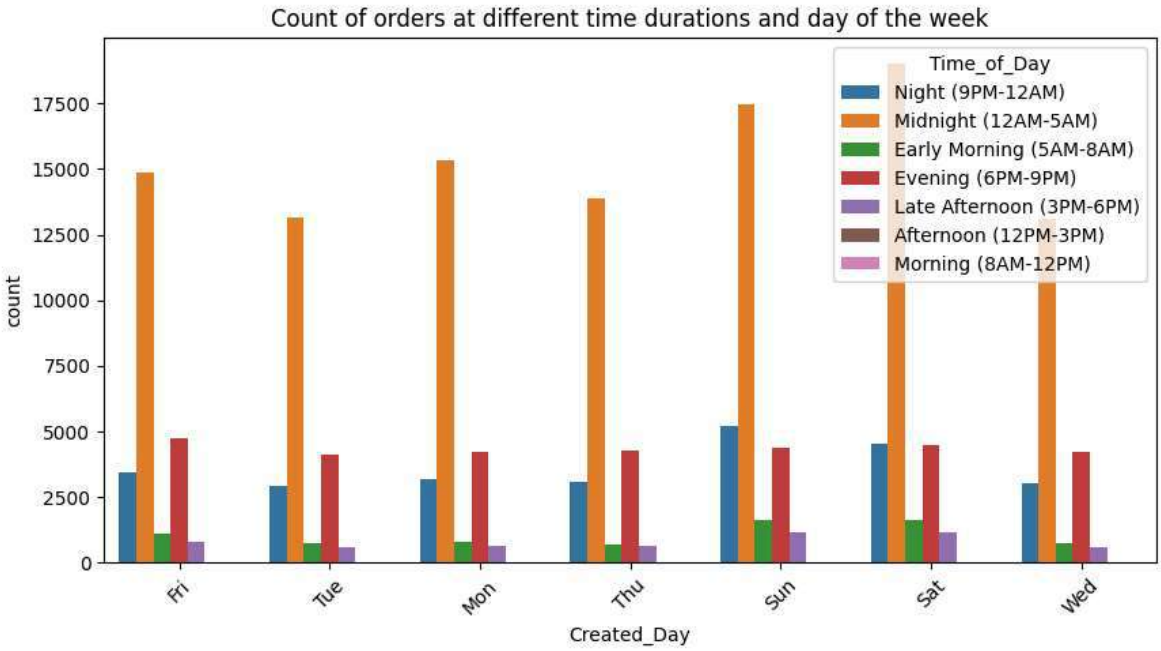
Count of orders at different time durations W.R.T Months

In [69]:
```
plt.figure(figsize=(10,5))
plt.title('Count of orders at different time durations W.R.T Months')
month_time_df = df[['Created_Month', 'Time_of_Day']]
sns.countplot(data=month_time_df, x='Time_of_Day', hue='Created_Month')
plt.xticks(rotation=45)
plt.show()
```



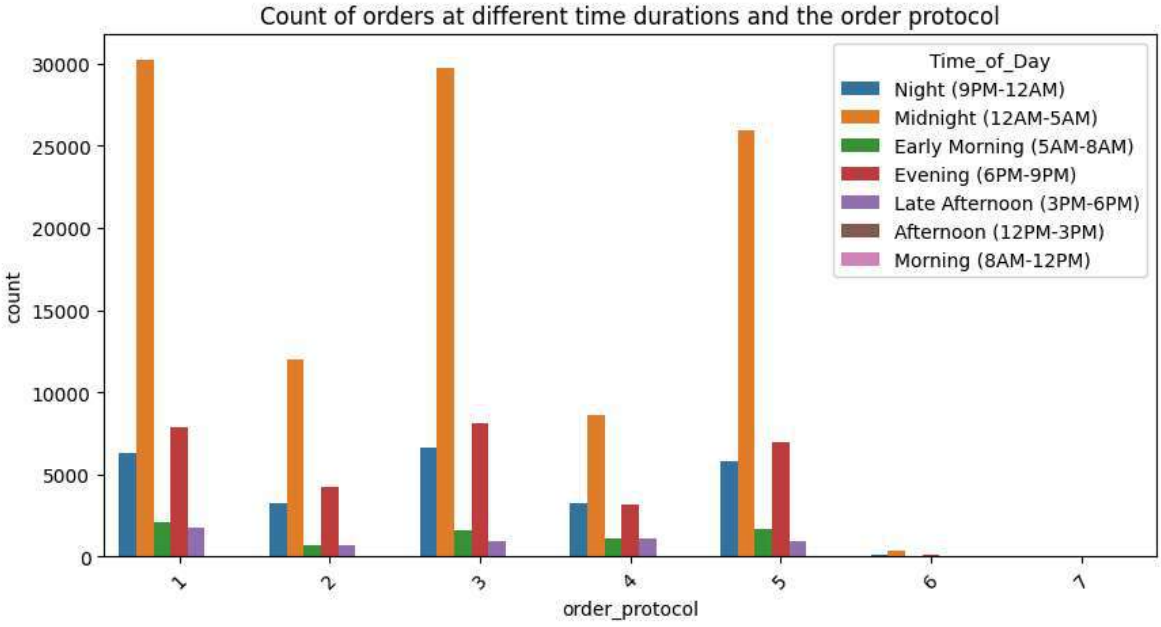
Count Of Orders At Different Time Durations W.R.T Day Of The Week

In [70]:
```
plt.figure(figsize=(10,5))
plt.title('Count of orders at different time durations and day of the week')
day_time_df = df[['Time_of_Day', 'Created_Day']]
sns.countplot(data=day_time_df, x='Created_Day', hue='Time_of_Day')
plt.xticks(rotation=45)
plt.show()
```



Count Of Orders at Different Time Durations W.R.T Order Protocol

In [71]:
```
plt.figure(figsize=(10,5))
plt.title('Count of orders at different time durations and the order protocol')
order_protocol_time_df = df[['Time_of_Day', 'order_protocol']]
sns.countplot(data=order_protocol_time_df, x='order_protocol', hue='Time_of_Day'
plt.xticks(rotation=45)
plt.show()
```



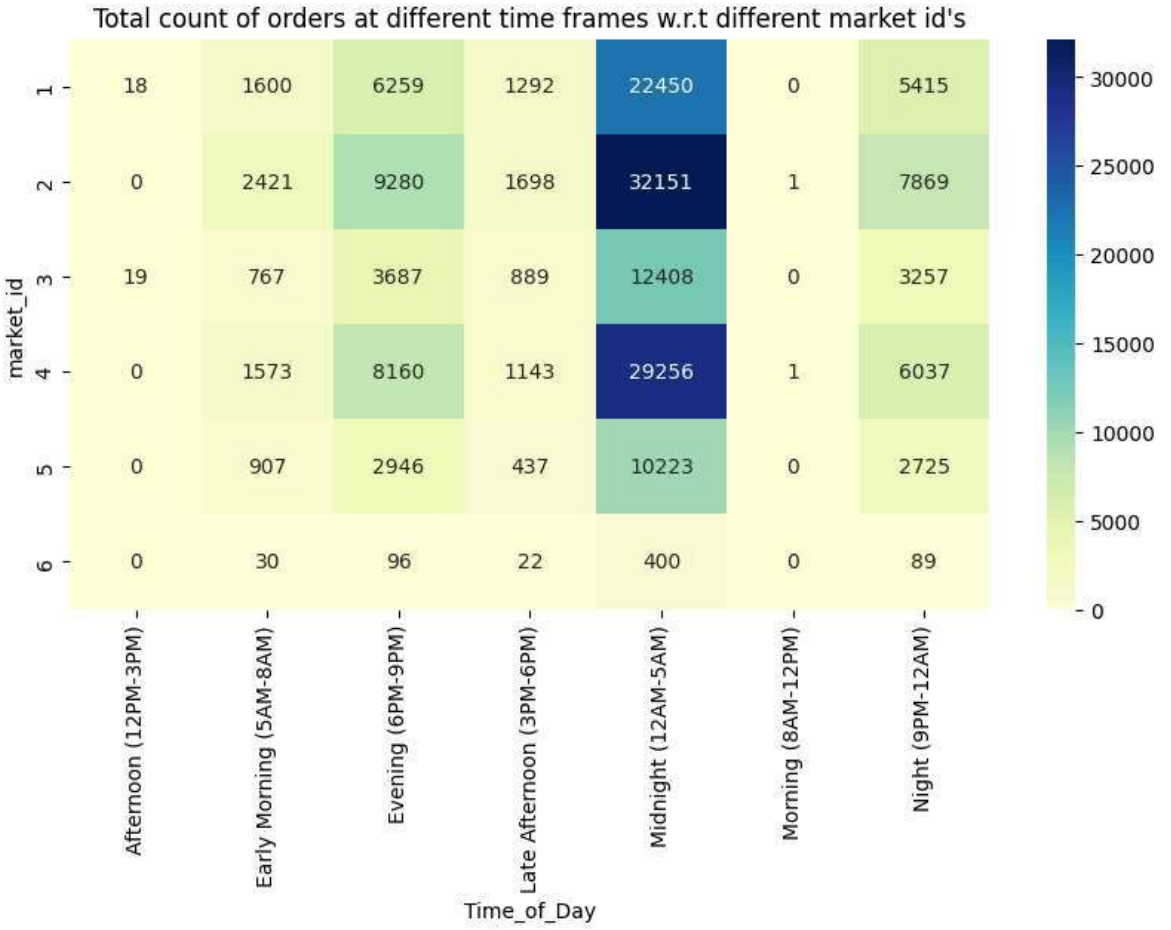
Categorical-Categorical Analysis

HeatMap

Total Count Of Orders At Different Time Frames W.R.T To Different Market IDs

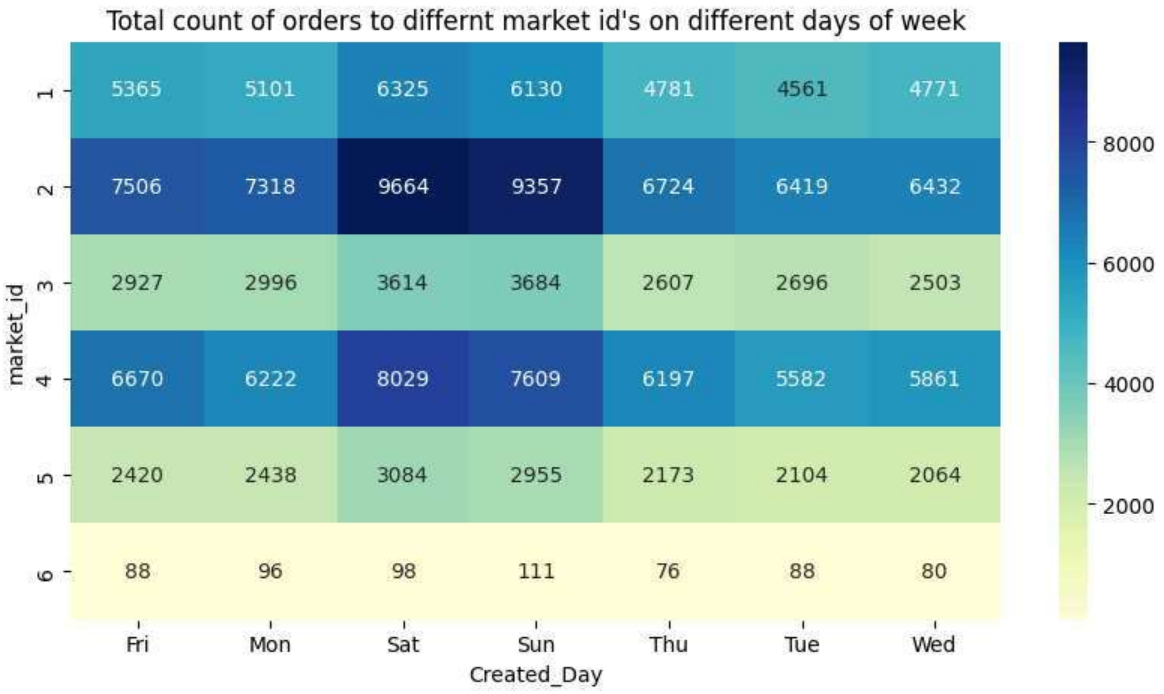
In [72]:
```
plt.figure(figsize=(10,5))
plt.title("Total count of orders at different time frames w.r.t different market
market_id_time_of_the_day = df[['market_id','Time_of_Day']].value_counts()
```

```
market_id_time_of_day_value_counts = market_id_time_of_the_day.reset_index()
market_id_time_of_day_value_pivort_df = market_id_time_of_day_value_counts.pivot
market_id_time_of_day_value_pivort_df.fillna(0, inplace=True)
market_id_time_of_day_value_pivort_df = market_id_time_of_day_value_pivort_df.as
sns.heatmap(data=market_id_time_of_day_value_pivort_df,annot=True, cmap='YlGnBu'
plt.show()
```



Total Count Of Orders At Different Week days W.R.T To Different Market IDs

In [73]:
```
plt.figure(figsize=(10,5))
plt.title("Total count of orders to differnt market id's on different days of we
market_id_created_day_value_counts = df[['market_id','Created_Day']].value_count
market_id_created_day_value_counts = market_id_created_day_value_counts.reset_in
market_id_created_day_pivort_df = market_id_created_day_value_counts.pivot(index
market_id_created_day_pivort_df.fillna(0, inplace=True)
market_id_created_day_pivort_df = market_id_created_day_pivort_df.astype(int)
sns.heatmap(data=market_id_created_day_pivort_df,annot=True, cmap='YlGnBu', fmt=
plt.show()
```

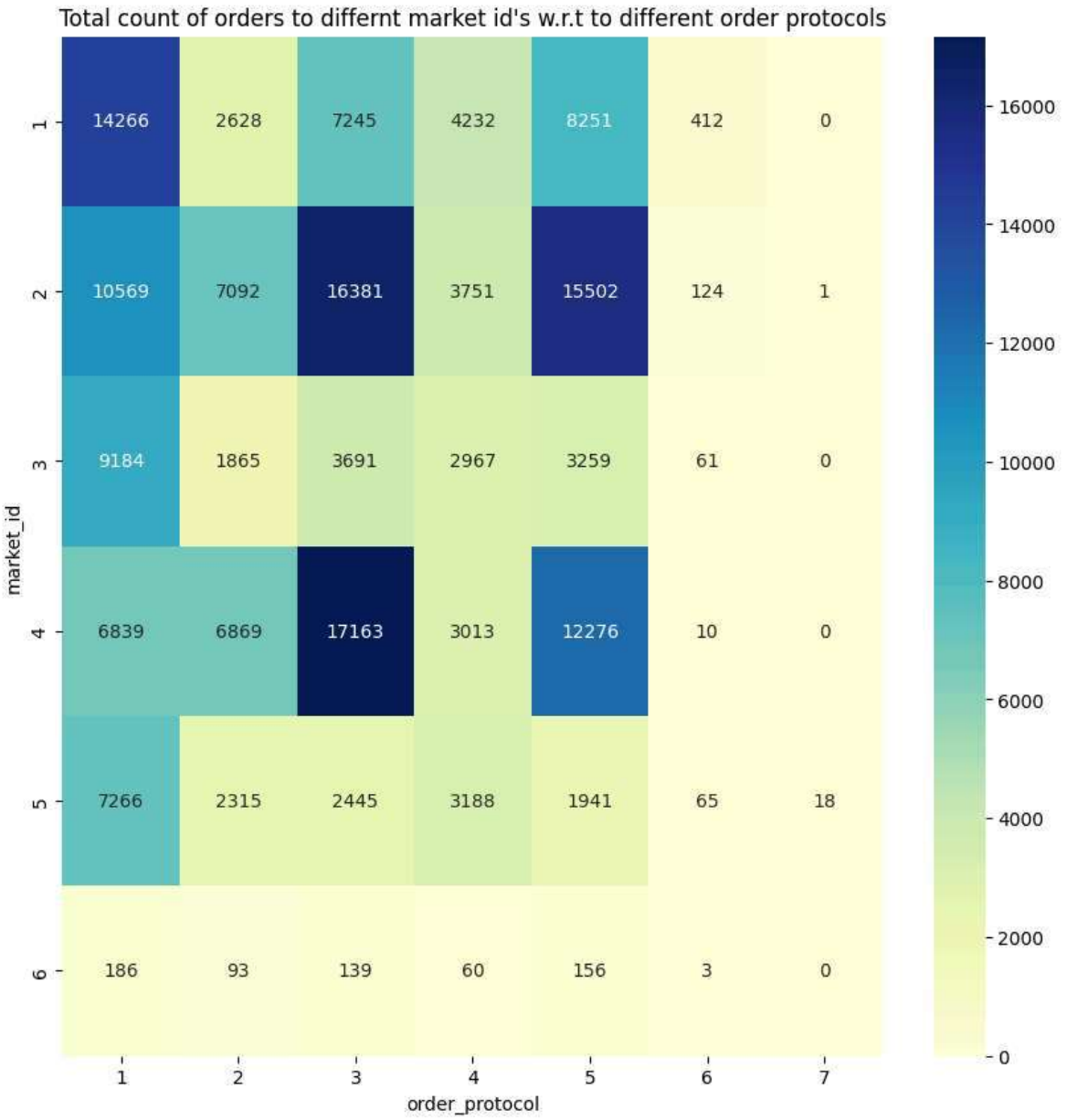


Total Count Of orders Recieved To Different Market IDs W.R.T To Different Store Primary Category

In [74]:
```
plt.figure(figsize=(100,10))
plt.title("Total count of orders recieved to different market id's w.r.t to diff
market_id_store_primary_category_value_counts = df[['market_id','store_primary_c
market_id_store_primary_category_value_counts = market_id_store_primary_category
market_id_store_primary_category_pivort_df = market_id_store_primary_category_va
market_id_store_primary_category_pivort_df.fillna(0, inplace=True)
market_id_store_primary_category_pivort_df = market_id_store_primary_category_pi
sns.heatmap(data=market_id_store_primary_category_pivort_df,annot=True, cmap='Yl
plt.show()
```

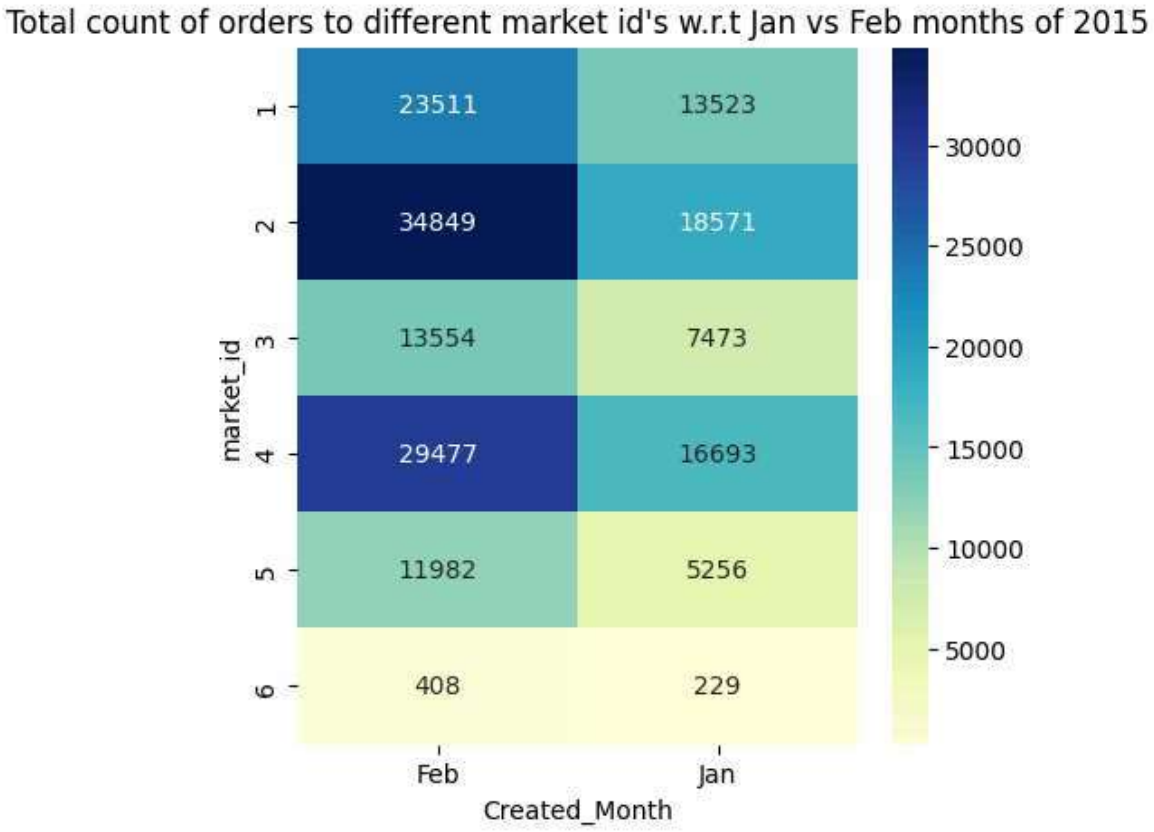
Total Count Of Orders To Differnt Market IDs W.R.T To Different Order Protocols

In [75]:
```
plt.figure(figsize=(10,10))
plt.title("Total count of orders to differnt market id's w.r.t to different orde
market_id_order_protocol_value_counts = df[['market_id','order_protocol']].value
market_id_order_protocol_value_counts = market_id_order_protocol_value_counts.re
market_id_order_protocol_pivort_df = market_id_order_protocol_value_counts.pivot
market_id_order_protocol_pivort_df.fillna(0, inplace=True)
market_id_order_protocol_pivort_df = market_id_order_protocol_pivort_df.astype(i
sns.heatmap(data=market_id_order_protocol_pivort_df,annot=True, cmap='YlGnBu', f
plt.show()
```



Total Count Of Orders To Different Market IDs W.R.T Jan vs Feb months of 2015

In [76]:
```
plt.figure(figsize=(5,5))
plt.title("Total count of orders to different market id's w.r.t Jan vs Feb month
market_id_created_month_value_counts = df[['market_id','Created_Month']].value_c
market_id_created_month_value_counts = market_id_created_month_value_counts.rese
market_id_created_month_pivoted = market_id_created_month_value_counts.pivot(ind
market_id_created_month_pivoted.fillna(0, inplace=True)
market_id_created_month_pivoted = market_id_created_month_pivoted.astype(int)
sns.heatmap(data=market_id_created_month_pivoted,annot=True, cmap='YlGnBu', fmt=
plt.show()
```



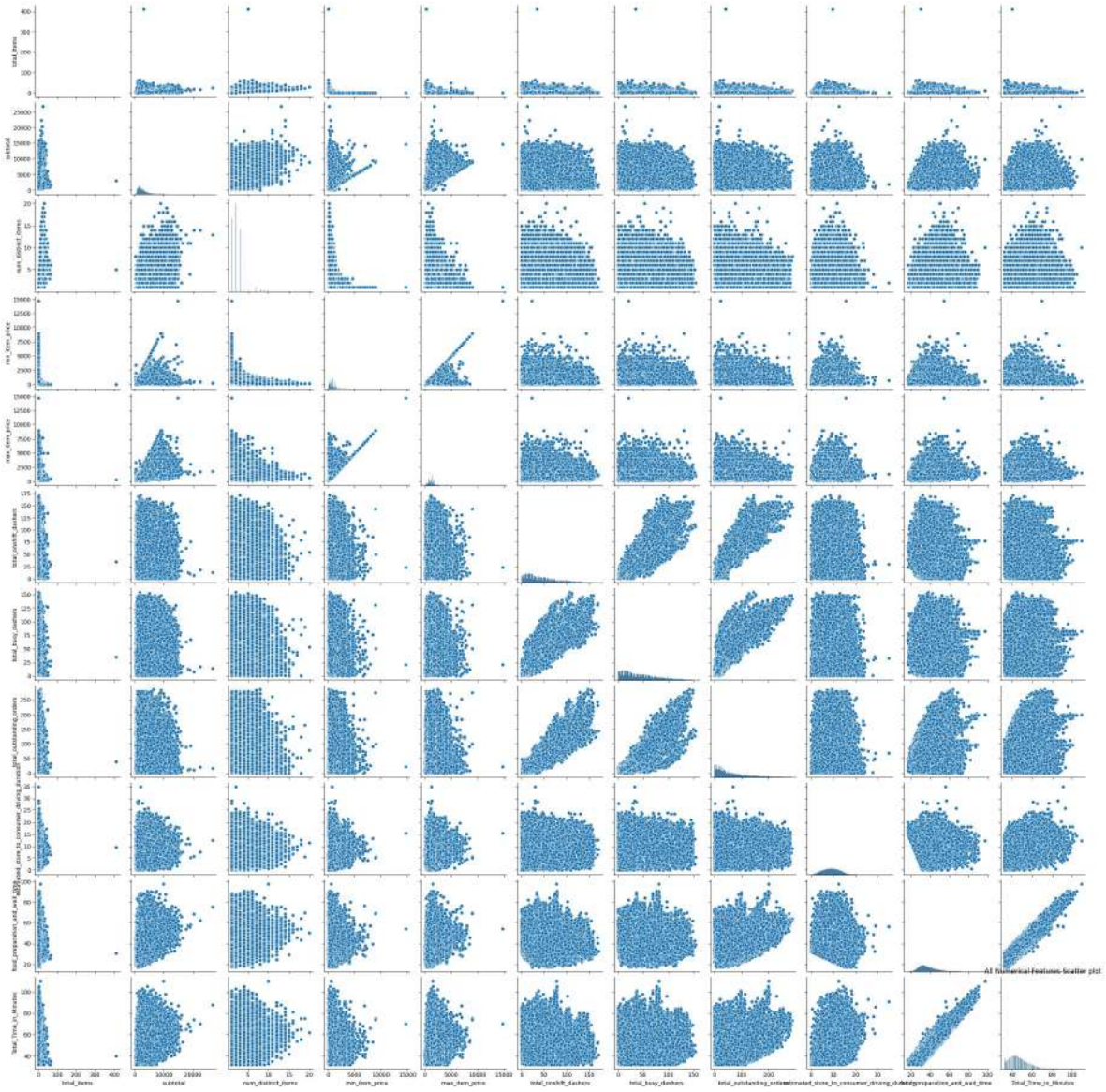
Numerical-Numerical Features Analysis

Scatter Plot

All Numerical Features Scatter plot

In [77]:
```
plt.figure('Correlation of all the numerical features')
numerical_columns = ['total_items', 'subtotal', 'num_distinct_items', 'min_item_
plt.figure(figsize=(20,50))
sns.pairplot(data=df, y_vars=numerical_columns, x_vars=numerical_columns)
plt.title('All Numerical Features Scatter plot')
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
<Figure size 2000x5000 with 0 Axes>
```



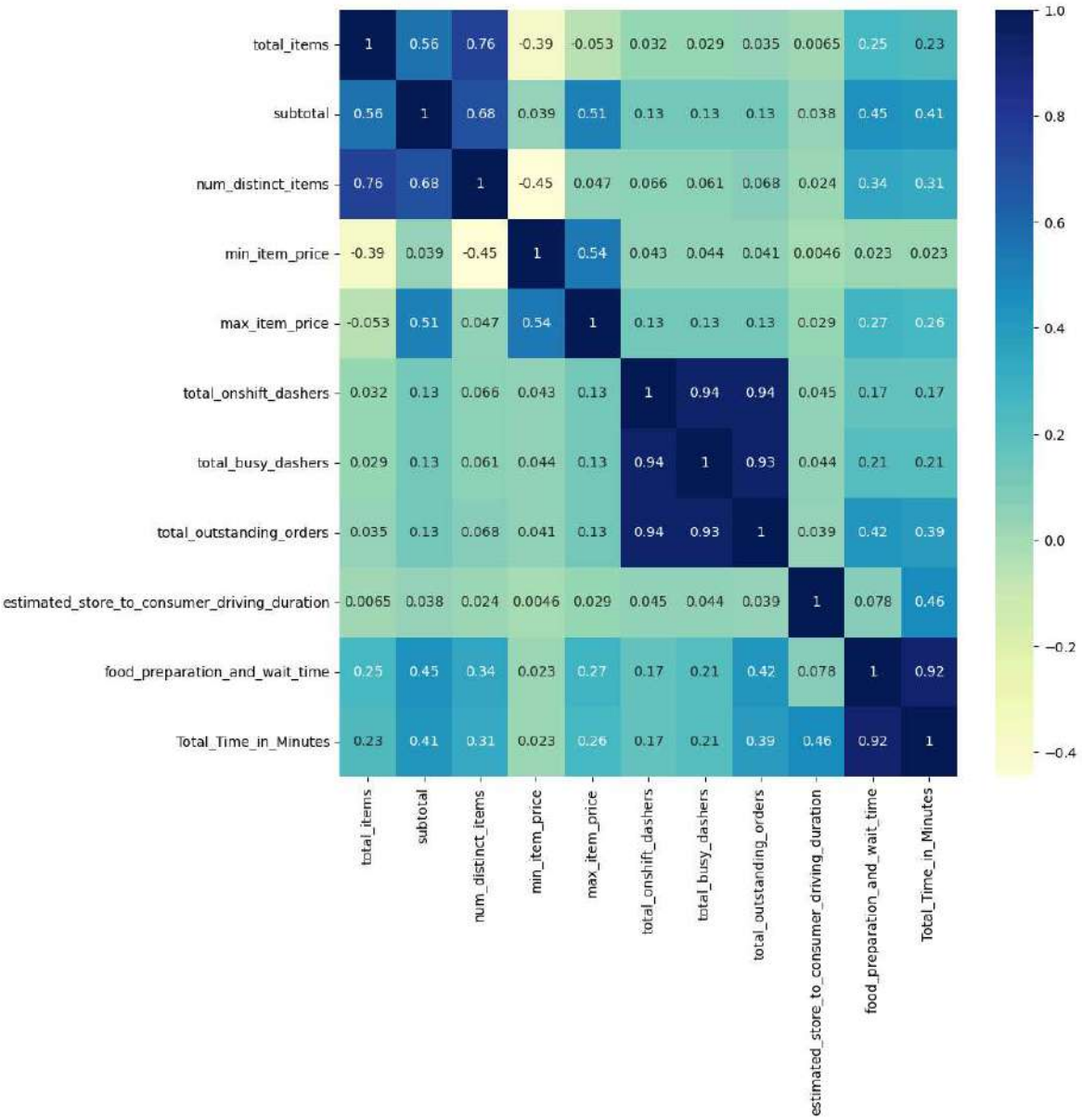
Correcation/HeatMap

All Numerical Features Correlation plot

In [78]:
```
plt.figure('Correlation of all the numerical features')
plt.figure(figsize=(10,10))
sns.heatmap(data=df[numerical_columns].corr(),annot=True, cmap='YlGnBu')
```

Out[78]:
```
<Axes: >
```

<Figure size 640x480 with 0 Axes>



Multivariate Analysis

Line Plot

The Total Number Of Orders W.R.T Time Range from 2015-01-21 to 2015-02-17 on Different Week Days

In [79]:
```
df.head()
```

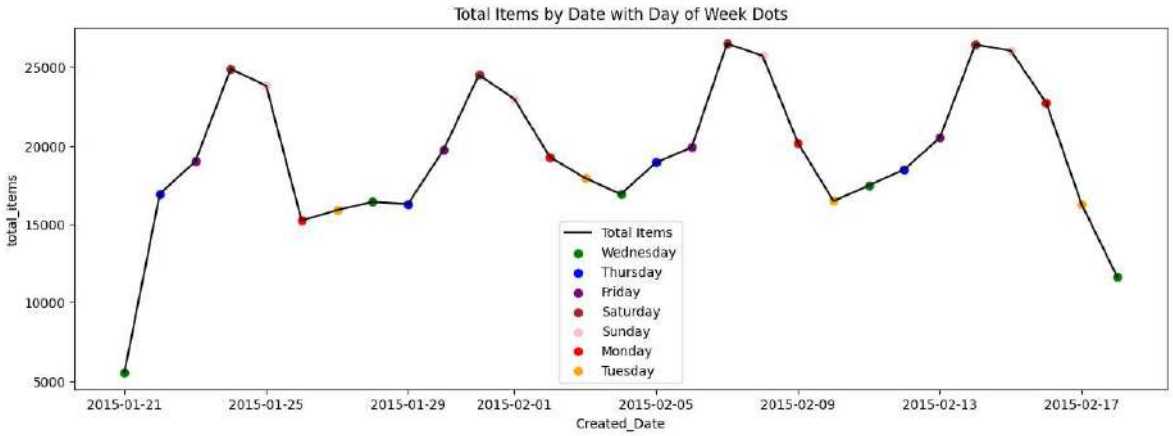
Out[79]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinc
0	1	4	1	4	3441	
1	2	46	2	1	1900	
2	2	36	3	4	4771	
3	1	38	1	1	1525	
4	1	38	1	2	3620	

In [80]:

```
plt.figure(figsize=(15,5))
total_items_sum = df.groupby(by='Created_Date')['total_items'].sum()
total_items_sum = total_items_sum.reset_index()
total_items_sum['day'] = pd.to_datetime(total_items_sum['Created_Date']).dt.day_

day_colors = {
    'Monday': 'red',
    'Tuesday': 'orange',
    'Wednesday': 'green',
    'Thursday': 'blue',
    'Friday': 'purple',
    'Saturday': 'brown',
    'Sunday': 'pink'
}

# Plot the line
plt.figure(figsize=(15,5))
plt.plot(total_items_sum['Created_Date'], total_items_sum['total_items'], label=

# Overlay dots with day-based color
for i, row in total_items_sum.iterrows():
    plt.scatter(row['Created_Date'], row['total_items'], color=day_colors[row['d

# To prevent duplicate legend entries
handles, labels = plt.gca().get_legend_handles_labels()
by_label = dict(zip(labels, handles))
plt.legend(by_label.values(), by_label.keys())

plt.xlabel('Created_Date')
plt.ylabel('total_items')
plt.title('Total Items by Date with Day of Week Dots')
plt.show()
```

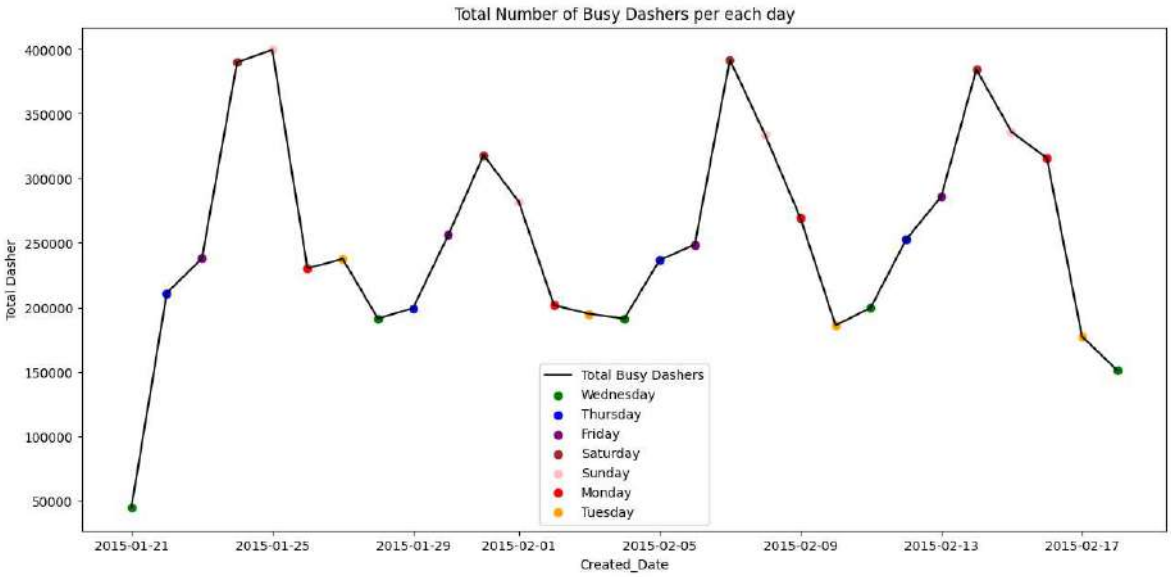
```
<Figure size 1500x500 with 0 Axes>
```



The Total Number Of Busy Dasher W.R.T Time Range from 2015-01-21 to 2015-02-17 on Different Week Days

In [81]:
```
plt.figure(figsize=(15,5))
total_busy_dashers_sum = df.groupby(by='Created_Date')['total_busy_dashers'].sum
total_busy_dashers_sum = total_busy_dashers_sum.reset_index()
total_busy_dashers_sum['day'] = pd.to_datetime(total_busy_dashers_sum['Created_D

day_colors = {
    'Monday': 'red',
    'Tuesday': 'orange',
    'Wednesday': 'green',
    'Thursday': 'blue',
    'Friday': 'purple',
    'Saturday': 'brown',
    'Sunday': 'pink'
}

# Plot the line
plt.figure(figsize=(15,7))
plt.plot(total_busy_dashers_sum['Created_Date'], total_busy_dashers_sum['total_b

# Overlay dots with day-based color
for i, row in total_busy_dashers_sum.iterrows():
    plt.scatter(row['Created_Date'], row['total_busy_dashers'], color=day_colors

# To prevent duplicate legend entries
handles, labels = plt.gca().get_legend_handles_labels()
by_label = dict(zip(labels, handles))
plt.legend(by_label.values(), by_label.keys(), loc='lower center')
plt.xlabel('Created_Date')
plt.ylabel('Total Dasher')
plt.title('Total Number of Busy Dashers per each day')
plt.show()
```

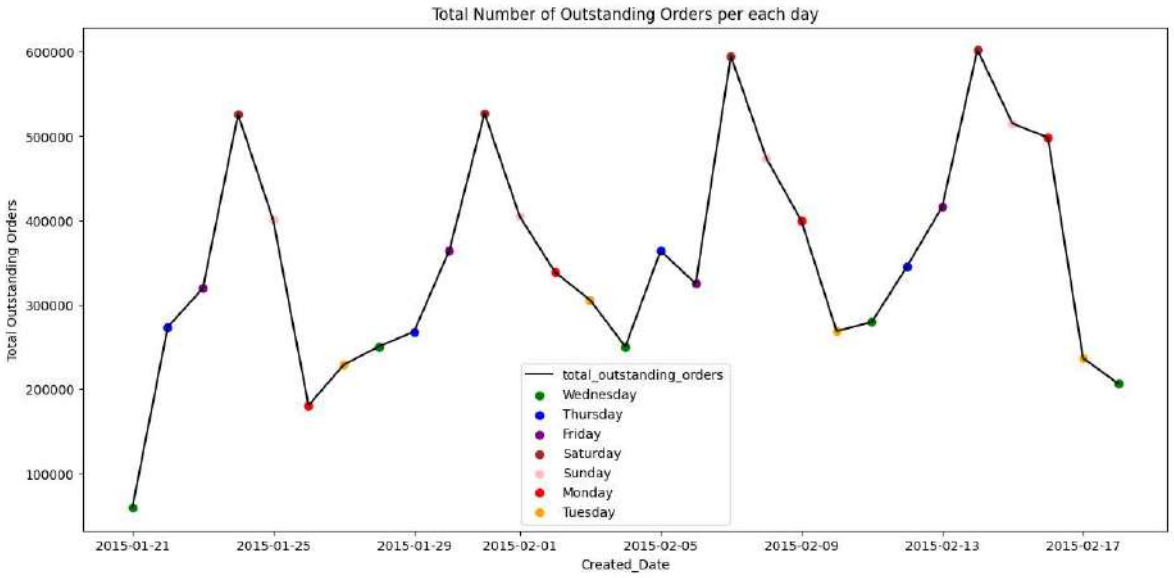
```
<Figure size 1500x500 with 0 Axes>
```



The Total Number Of Outstanding Orders W.R.T Time Range from 2015-01-21 to 2015-02-17 on Different Week Days

In [82]:
```
plt.figure(figsize=(15,5))
total_outstanding_order = df.groupby(by='Created_Date')['total_outstanding_order
total_outstanding_order = total_outstanding_order.reset_index()
total_outstanding_order['day'] = pd.to_datetime(total_outstanding_order['Created

day_colors = {
    'Monday': 'red',
    'Tuesday': 'orange',
    'Wednesday': 'green',
    'Thursday': 'blue',
    'Friday': 'purple',
    'Saturday': 'brown',
    'Sunday': 'pink'
}

# Plot the line
plt.figure(figsize=(15,7))
plt.plot(total_outstanding_order['Created_Date'], total_outstanding_order['total

# Overlay dots with day-based color
for i, row in total_outstanding_order.iterrows():
    plt.scatter(row['Created_Date'], row['total_outstanding_orders'], color=day_

# To prevent duplicate legend entries
handles, labels = plt.gca().get_legend_handles_labels()
by_label = dict(zip(labels, handles))
plt.legend(by_label.values(), by_label.keys(), loc='lower center')
plt.xlabel('Created_Date')
plt.ylabel('Total Outstanding Orders')
plt.title('Total Number of Outstanding Orders per each day')
plt.show()
```

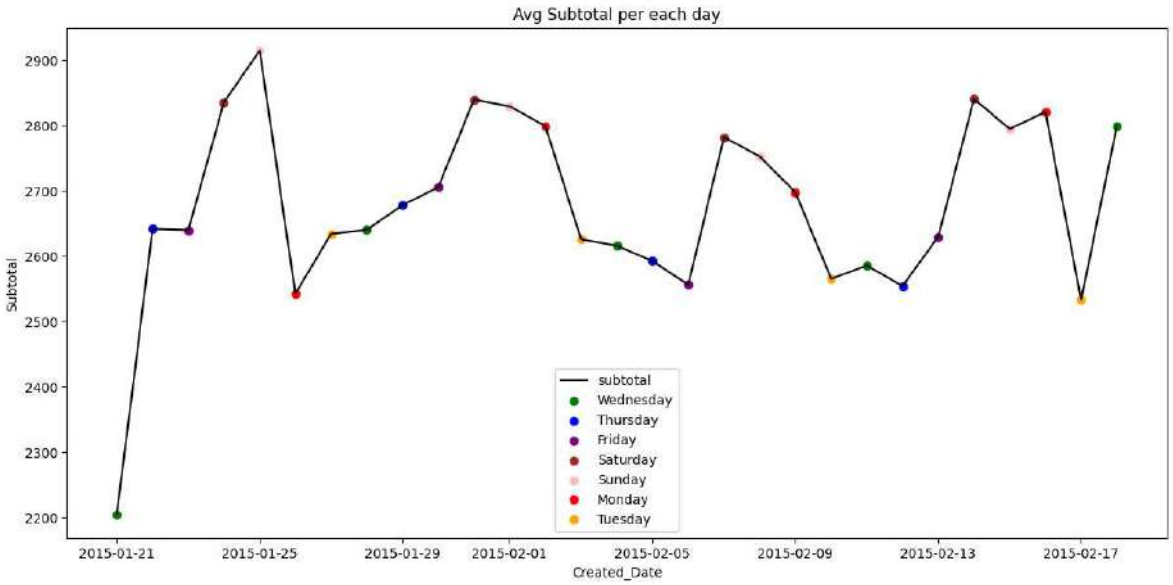
```
<Figure size 1500x500 with 0 Axes>
```



The Final Price Of Orders W.R.T Time Range from 2015-01-21 to 2015-02-17 on Different Week Days

In [83]:
```
plt.figure(figsize=(15,5))
avg_subtotal_per_day = df.groupby(by='Created_Date')['subtotal'].mean()
avg_subtotal_per_day = avg_subtotal_per_day.reset_index()
avg_subtotal_per_day['day'] = pd.to_datetime(avg_subtotal_per_day['Created_Date'

day_colors = {
    'Monday': 'red',
    'Tuesday': 'orange',
    'Wednesday': 'green',
    'Thursday': 'blue',
    'Friday': 'purple',
    'Saturday': 'brown',
    'Sunday': 'pink'
}

# Plot the line
plt.figure(figsize=(15,7))
plt.plot(avg_subtotal_per_day['Created_Date'], avg_subtotal_per_day['subtotal'],

# Overlay dots with day-based color
for i, row in avg_subtotal_per_day.iterrows():
    plt.scatter(row['Created_Date'], row['subtotal'], color=day_colors[row['day'

# To prevent duplicate legend entries
handles, labels = plt.gca().get_legend_handles_labels()
by_label = dict(zip(labels, handles))
plt.legend(by_label.values(), by_label.keys(), loc='lower center')
plt.xlabel('Created_Date')
plt.ylabel('Subtotal')
plt.title('Avg Subtotal per each day')
plt.show()
```

```
<Figure size 1500x500 with 0 Axes>
```



EDA Insights:

- The dataset has dates ranging from **2015-01-21 to 2015-02-18.**
- On an average people are spending **Rs. 2697/-** per order.
- Average Onshift drivers are **44.91**, where as total busy dashers are **41.86** this means that dashers who wait for preparation and ready for delivering the order are more in avg compared to actual busy dashers who is yet to deliver.
- Most of the restaurants are located in regions **2, 4, and 1** (in descending order), accounting for **77.83%** of the total. In contrast, only **22.17%** of restaurants are found in regions **3, 5, and 6** (also in descending order).
- Most of orders were placed with the protocol **1,3 & 5** (in descending order), accounting for **77.91%** of the total. In contrast, only few orders were placed with the protocol **2,4,6 & 7** (in descending order).
- Most of the orders are placed on saturday and sunday (in descending order) compared to other days Monday, thursday, wednesday, tuesday (in descending order).
- We have 65.82% of data of February month and only 35.18 of January month.
- Restaurants with categories **4,55, 46, 13, 58, 20, 39, 24, 38, 28** (in descending order) are in **top 10** w.r.t to order count.
- Restaurants with categories **33, 48, 22, 27, 56, 1, 43, 8, 3, 21** (in descending order) are in **bottom 10** w.r.t to order count.
- **Most of the orders** are placed at **Midnight (12AM to 5AM)**, in contrast **very few orders** are placed during **8AM to 6PM**.
- Distribution of **Total Items** is **right skewed (75% of order items are less than 4 items)** this means that most of the order items are very less compared to orders with most **extreme** count i.e **411 items. **
- Distribution of **Total amount spent** per order is also **right skewed (75% orders are less Rs. 3413/-)**, where as the extreme amount spent per order is **Rs. 26800/-**
- The Distribution of total number of **distinct items** are also **right skewed(75% of order distinct items are less than 4 items)**, where as the **extreme** value is **20**

items.

- **Both minimum and maximum item prices per order are right-skewed,** indicating that most orders contain reasonably priced items, but a small number of orders include high-priced items.
- The distibution of number of delivery **partners on duty** at the time order was placed is also **right skewed with (75% of dashers are less than 66)**, where as the **extreme** value is **171 dashers.**
- The distibution of number of delivery **partners attending to other tasks** is also right **skewed with **(75% of dashers are less than 63)**, where as the **extreme value is 154 dashers.**
- The distribution of **total number of orders to be fulfilled** at the moment is also right skewed with **(75% of orders to be fullflled are less than 85)**, where as the **extreme value is 285 orders**
- The distribution of **estimmated store to consumer driving duration** is also **right skewed (75% times the travelling time is less than 20 mins)** this means that most of the orders are delivered quickly and **few orders took more than 20 mins** for travelling
- The distribution of **total Delivery time (Including order place time, cooking, packing and delivery) is also right skewed** where most of the orders are delivered qucikly where as few orders took around 2 hours.
- The total time in minutes (target variable) is highly correlated with the food preparation and wait time than the estimated store to consumer driving duration.

Rcommendations:

- Since the food preparation and wait time is taking more time so need to estimate the daily orders and the parcels should be made quicker for delivery.
- Need to add more staff at the cooking department and more containers and vessels and stoves should be used for faster preparation.
- The delivey partners waiting time in the retaurants should be reduced by handling the parcels quickly.
- More delivery partners should be operated during the peak hours i.e after 12AM to 5AM.
- The delivery charges should also be increased during the peak hours for faster delivery compared to no peak hours.
- More delivery partners should be on duty on saturdays and sundays.

Modelling

Data Preprocessing

In [84]:
```
final_df = df.copy()
final_df.columns
```

Out[84]:
```
Index(['market_id', 'store_primary_category', 'order_protocol', 'total_items',
       'subtotal', 'num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_dashers', 'total_busy_dashers',
       'total_outstanding_orders',
       'estimated_store_to_consumer_driving_duration', 'Created_Day',
       'Created_Month', 'Created_Date', 'food_preparation_and_wait_time',
       'Time_of_Day', 'Total_Time_in_Minutes'],
      dtype='object')
```

In [85]:
```
final_df.dtypes
```

Out[85]:

	0
market_id	category
store_primary_category	category
order_protocol	category
total_items	int64
subtotal	int64
num_distinct_items	int64
min_item_price	int64
max_item_price	int64
total_onshift_dashers	int64
total_busy_dashers	int64
total_outstanding_orders	int64
estimated_store_to_consumer_driving_duration	float64
Created_Day	object
Created_Month	object
Created_Date	object
food_preparation_and_wait_time	float64
Time_of_Day	object
Total_Time_in_Minutes	float64

dtype: object

In [86]:
```
numerical_columns
```

Out[86]:
```
['total_items',
 'subtotal',
 'num_distinct_items',
 'min_item_price',
 'max_item_price',
 'total_onshift_dashers',
 'total_busy_dashers',
 'total_outstanding_orders',
 'estimated_store_to_consumer_driving_duration',
 'food_preparation_and_wait_time',
 'Total_Time_in_Minutes']
```

In [87]:
```
final_df.columns[~final_df.columns.isin(numerical_columns)].to_list()
```

Out[87]:
```
['market_id',
 'store_primary_category',
 'order_protocol',
 'Created_Day',
 'Created_Month',
 'Created_Date',
 'Time_of_Day']
```

Encoding Categorical Columns

In [88]:
```
cat_columns = final_df.columns[~final_df.columns.isin(numerical_columns)].to_lis
for _cat in cat_columns:
  final_df[_cat] = final_df[_cat].astype('category').cat.codes
```

In [89]:
```
final_df.head(2)
```

Out[89]:

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinc
0	0	4	0	4	3441	
1	1	46	1	1	1900	

Handling/Removal of Outliers using LOF

In [90]:
```
samples_before = final_df.shape[0]
samples_before
```

Out[90]:
```
175526
```

In [91]:
```
num_df = final_df.loc[:, numerical_columns]
_model = LocalOutlierFactor()
labels = _model.fit_predict(num_df)
final_df = final_df[labels == 1]
```

In [92]:
```
samples_after = final_df.shape[0]
samples_after
```

Out[92]:
```
172312
```

In [93]:
```
samples_before - samples_after
```

Out[93]: 3214

Splitting the Data into Train and Test Samples

In [94]:
```
y=final_df['Total_Time_in_Minutes']
X = final_df.drop(['Total_Time_in_Minutes'], axis=1)
final_df.drop(['Total_Time_in_Minutes'], axis=1,inplace=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,  random
```

```
/tmp/ipython-input-94-1533797870.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  final_df.drop(['Total_Time_in_Minutes'], axis=1,inplace=True)
```

In [95]:
```
print('Shape of X_train :', X_train.shape)
print('Shape of y_train :', y_train.shape)
print('Shape of X_test :', X_test.shape)
print('Shape of y_test :', y_test.shape)
```

```
Shape of X_train : (137849, 17)
Shape of y_train : (137849,)
Shape of X_test : (34463, 17)
Shape of y_test : (34463,)
```

Random Forest Modelling

RF model Training

In [96]:
```
regressor = RandomForestRegressor()

regressor.fit(X_train, y_train)
```

Out[96]:
```
RandomForestRegressor
RandomForestRegressor()
```

RF model Performance Metrics

MSE, RMSE & MAE

In [97]:
```
prediction = regressor.predict(X_train)
mse = mean_squared_error(y_train, prediction)
mae = mean_absolute_error(y_train, prediction)
rmse = mse**.5

print('Train Accuracy\n')
print("mse : ", f'{round(mse,3)} minutes => {round(mse * 60, 3)} seconds')
print("rmse : ",f'{round(rmse,3)} minutes => {round(rmse * 60, 3)} seconds')
print('mae:' ,f'{round(mae,3)} minutes => {round(mae * 60 ,3)} seconds')
print('\n')

prediction = regressor.predict(X_test)
```

```
mse = mean_squared_error(y_test, prediction)
mae = mean_absolute_error(y_test, prediction)
rmse = mse**.5

print('Test Accuracy\n')
print("mse : ", f'{round(mse,3)} minutes => {round(mse * 60, 3)} seconds')
print("rmse : ",f'{round(rmse,3)} minutes => {round(rmse * 60 ,3)} seconds')
print('mae:' ,f'{round(mae,3)} minutes => {round(mae * 60,3)} seconds')
```

```
Train Accuracy

mse :  0.001 minutes => 0.047 seconds
rmse :  0.028 minutes => 1.685 seconds
mae: 0.003 minutes => 0.175 seconds


Test Accuracy

mse :  0.007 minutes => 0.395 seconds
rmse :  0.081 minutes => 4.865 seconds
mae: 0.008 minutes => 0.478 seconds
```

RF model R-Square Score

In [98]:
```
r2_score(y_test, prediction)
```

Out[98]:
```
0.9999233404585227
```

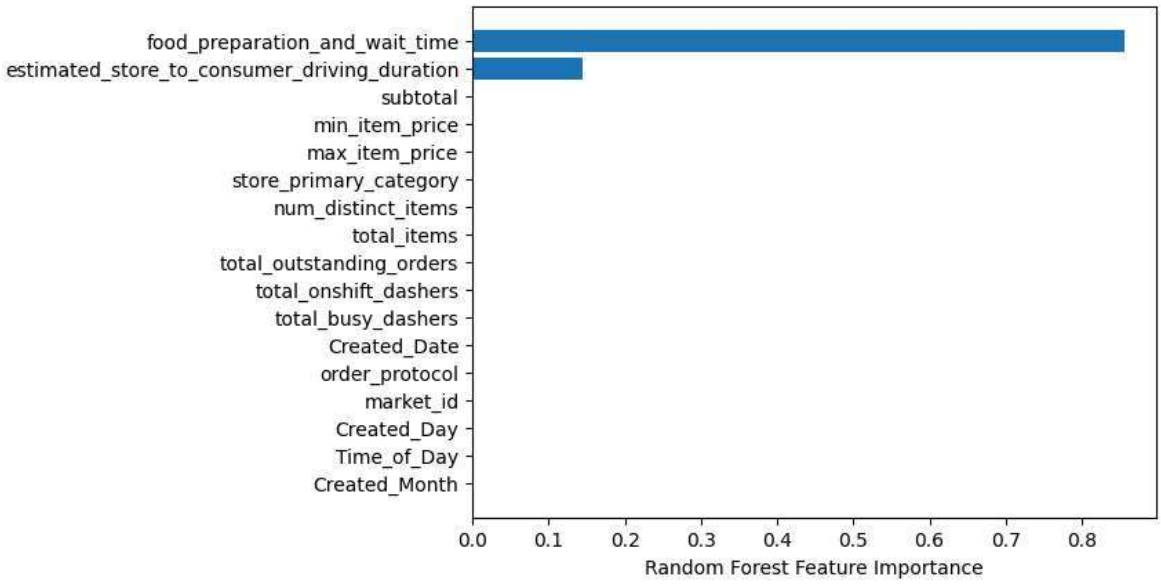
RF model MAPE

In [99]:
```
def MAPE(Y_actual,Y_Predicted):
    mape = np.mean(np.abs((Y_actual - Y_Predicted)/Y_actual))*100
    return mape

print("mape : ",MAPE(y_test, prediction))
```

```
mape :  0.013174935058472823
```

RF model Feature Importance

In [100…
```
sorted_idx = regressor.feature_importances_.argsort()
plt.barh(df.columns[sorted_idx], regressor.feature_importances_[sorted_idx])
plt.xlabel("Random Forest Feature Importance")
plt.show()
```



Neural Networks

Feature Scaling

In [101…
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

NN Hyperparameter tuning

In [107…
```
mean_absolute_percentage_error(y_test, prediction)
```

Out[107…
```
0.0005595739274562917
```

In [108…
```
def build_model(hp):
    model = Sequential()

    # Input Layer
    model.add(Dense(
        units=hp.Int('units_input', min_value=64, max_value=1024, step=64),
        activation='relu',
        input_shape=(X_train.shape[1],),
        kernel_initializer='he_uniform'
    ))

    # Hidden Layers
    for i in range(hp.Int('num_hidden_layers', 1, 4)):
        model.add(Dense(
            units=hp.Int(f'units_{i}', min_value=64, max_value=1024, step=64),
            activation='relu'
        ))

    # Output Layer
    model.add(Dense(1, activation='linear'))

    # Optimizer
    optimizer = Adam(learning_rate=hp.Choice('learning_rate', [0.01, 0.001, 0.00
```

```
    model.compile(optimizer=optimizer, loss='mse', metrics=['mse', 'mae'])
    return model
```

In [109…
```
tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=10,              # number of combinations to try
    executions_per_trial=1,     # how many times to train each model
    directory='tuner_dir',
    project_name='nn_regression'
)
```

```
Reloading Tuner from tuner_dir/nn_regression/tuner0.json
```

In [110…
```
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=

tuner.search(
    X_train_scaled, y_train,
    epochs=30,
    batch_size=512,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
)
```

In [111…
```
best_model = tuner.get_best_models(num_models=1)[0]
best_hyperparams = tuner.get_best_hyperparameters(num_trials=1)[0]

print("Best Hyperparameters:")
print(best_hyperparams.values)
```

```
Best Hyperparameters:
{'units_input': 128, 'num_hidden_layers': 1, 'units_0': 768, 'learning_rate': 0.0
01, 'units_1': 832, 'units_2': 448, 'units_3': 512}
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWa
rning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using S
equential models, prefer using an `Input(shape)` object as the first layer in the
model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
/usr/local/lib/python3.11/dist-packages/keras/src/saving/saving_lib.py:757: UserW
arning: Skipping variable loading for optimizer 'adam', because it has 2 variable
s whereas the saved optimizer has 14 variables.
  saveable.load_own_variables(weights_store.get(inner_path))
```

In [112…
```
loss, mse, mae = best_model.evaluate(X_test_scaled, y_test)
print(f"Test MSE: {mse}, Test MAE: {mae}")
```

```
1077/1077 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - loss: 0.0088 - mae: 0.0581 - mse: 0.
0088
Test MSE: 0.0083652688190341, Test MAE: 0.05779466778039932
```

Defining NN architecture

In [113…
```
model = Sequential()
model.add(Dense(17,  kernel_initializer='normal', activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(768, activation='relu'))
model.add(Dense(1, activation='linear'))
```

```
adam=Adam(learning_rate=0.001)
model.compile(loss='mse', optimizer=adam, metrics=['mse','mae'])
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

NN Model training

In [114…

```
history=model.fit(X_train_scaled, y_train, epochs=30, batch_size=512, verbose=1,
```

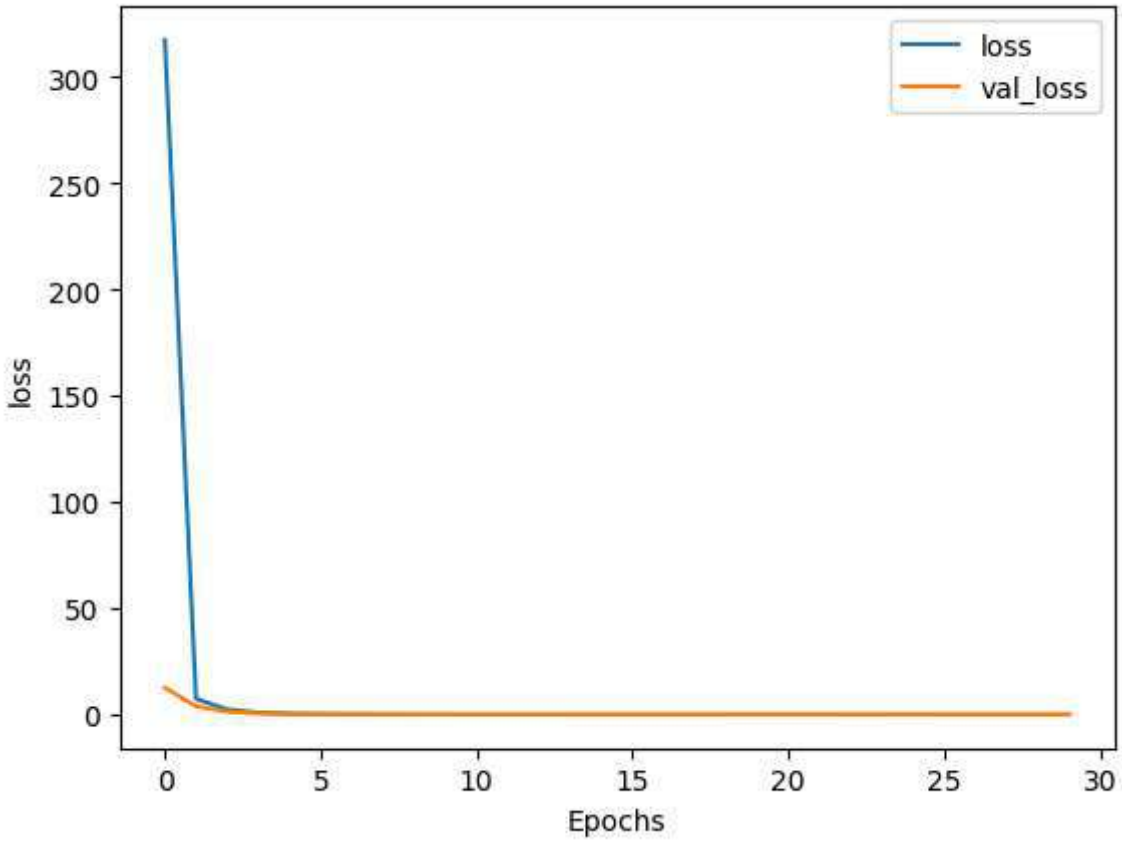
```
Epoch 1/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 6s 20ms/step - loss: 870.8058 - mae: 22.3775 - mse:
870.8058 - val_loss: 12.5684 - val_mae: 2.7835 - val_mse: 12.5684
Epoch 2/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 9.4819 - mae: 2.4024 - mse: 9.4
819 - val_loss: 3.7858 - val_mae: 1.5191 - val_mse: 3.7858
Epoch 3/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 2.9165 - mae: 1.3187 - mse: 2.9
165 - val_loss: 1.2474 - val_mae: 0.8517 - val_mse: 1.2474
Epoch 4/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 1.0001 - mae: 0.7557 - mse: 1.0
001 - val_loss: 0.4844 - val_mae: 0.5201 - val_mse: 0.4844
Epoch 5/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 15ms/step - loss: 0.4086 - mae: 0.4736 - mse: 0.4
086 - val_loss: 0.2271 - val_mae: 0.3497 - val_mse: 0.2271
Epoch 6/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 4s 16ms/step - loss: 0.1922 - mae: 0.3217 - mse: 0.1
922 - val_loss: 0.1258 - val_mae: 0.2564 - val_mse: 0.1258
Epoch 7/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.1118 - mae: 0.2411 - mse: 0.1
118 - val_loss: 0.0810 - val_mae: 0.2006 - val_mse: 0.0810
Epoch 8/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - loss: 0.0721 - mae: 0.1883 - mse: 0.0
721 - val_loss: 0.0548 - val_mae: 0.1594 - val_mse: 0.0548
Epoch 9/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - loss: 0.0487 - mae: 0.1518 - mse: 0.0
487 - val_loss: 0.0382 - val_mae: 0.1313 - val_mse: 0.0382
Epoch 10/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 5s 11ms/step - loss: 0.0351 - mae: 0.1267 - mse: 0.0
351 - val_loss: 0.0287 - val_mae: 0.1129 - val_mse: 0.0287
Epoch 11/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 14ms/step - loss: 0.0266 - mae: 0.1079 - mse: 0.0
266 - val_loss: 0.0220 - val_mae: 0.0962 - val_mse: 0.0220
Epoch 12/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 4s 17ms/step - loss: 0.0201 - mae: 0.0948 - mse: 0.0
201 - val_loss: 0.0178 - val_mae: 0.0875 - val_mse: 0.0178
Epoch 13/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.0163 - mae: 0.0831 - mse: 0.0
163 - val_loss: 0.0136 - val_mae: 0.0778 - val_mse: 0.0136
Epoch 14/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0126 - mae: 0.0732 - mse: 0.0
126 - val_loss: 0.0109 - val_mae: 0.0675 - val_mse: 0.0109
Epoch 15/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 7s 19ms/step - loss: 0.0100 - mae: 0.0651 - mse: 0.0
100 - val_loss: 0.0090 - val_mae: 0.0642 - val_mse: 0.0090
Epoch 16/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0076 - mae: 0.0586 - mse: 0.0
076 - val_loss: 0.0068 - val_mae: 0.0537 - val_mse: 0.0068
Epoch 17/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0065 - mae: 0.0526 - mse: 0.0
065 - val_loss: 0.0057 - val_mae: 0.0500 - val_mse: 0.0057
Epoch 18/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0052 - mae: 0.0483 - mse: 0.0
052 - val_loss: 0.0049 - val_mae: 0.0465 - val_mse: 0.0049
Epoch 19/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 7s 19ms/step - loss: 0.0046 - mae: 0.0450 - mse: 0.0
046 - val_loss: 0.0043 - val_mae: 0.0430 - val_mse: 0.0043
Epoch 20/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 11ms/step - loss: 0.0034 - mae: 0.0394 - mse: 0.0
034 - val_loss: 0.0031 - val_mae: 0.0360 - val_mse: 0.0031
```

```
Epoch 21/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0029 - mae: 0.0365 - mse: 0.0
029 - val_loss: 0.0028 - val_mae: 0.0343 - val_mse: 0.0028
Epoch 22/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - loss: 0.0026 - mae: 0.0349 - mse: 0.0
026 - val_loss: 0.0024 - val_mae: 0.0334 - val_mse: 0.0024
Epoch 23/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.0022 - mae: 0.0312 - mse: 0.0
022 - val_loss: 0.0018 - val_mae: 0.0280 - val_mse: 0.0018
Epoch 24/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0020 - mae: 0.0309 - mse: 0.0
020 - val_loss: 0.0018 - val_mae: 0.0289 - val_mse: 0.0018
Epoch 25/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0016 - mae: 0.0278 - mse: 0.0
016 - val_loss: 0.0039 - val_mae: 0.0492 - val_mse: 0.0039
Epoch 26/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step - loss: 0.0017 - mae: 0.0294 - mse: 0.0
017 - val_loss: 0.0012 - val_mae: 0.0238 - val_mse: 0.0012
Epoch 27/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 4s 12ms/step - loss: 0.0015 - mae: 0.0278 - mse: 0.0
015 - val_loss: 0.0013 - val_mae: 0.0239 - val_mse: 0.0013
Epoch 28/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - loss: 0.0011 - mae: 0.0239 - mse: 0.0
011 - val_loss: 0.0011 - val_mae: 0.0229 - val_mse: 0.0011
Epoch 29/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.0018 - mae: 0.0315 - mse: 0.0
018 - val_loss: 0.0018 - val_mae: 0.0322 - val_mse: 0.0018
Epoch 30/30
216/216 ━━━━━━━━━━━━━━━━━━━━ 4s 19ms/step - loss: 0.0020 - mae: 0.0324 - mse: 0.0
020 - val_loss: 0.0010 - val_mae: 0.0232 - val_mse: 0.0010
```

NN Loss Vs Epochs Plot

In [115…

```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_history(history, 'loss')
```



NN Model Performance Metreics

MSE,RMSE & MAE

In [116…

```
prediction = model.predict(X_train_scaled)
mse = mean_squared_error(y_train, prediction)
mae = mean_absolute_error(y_train, prediction)
rmse = mse**.5

print('Train Accuracy\n')
print("mse : ", f'{round(mse,3)} minutes => {round(mse * 60, 3)} seconds')
print("rmse : ",f'{round(rmse,3)} minutes => {round(rmse * 60, 3)} seconds')
print('mae:' ,f'{round(mae,3)} minutes => {round(mae * 60 ,3)} seconds')
print('\n')

prediction = model.predict(X_test_scaled)
mse = mean_squared_error(y_test, prediction)
mae = mean_absolute_error(y_test, prediction)
rmse = mse**.5

print('Test Accuracy\n')
print("mse : ", f'{round(mse,3)} minutes => {round(mse * 60, 3)} seconds')
print("rmse : ",f'{round(rmse,3)} minutes => {round(rmse * 60 ,3)} seconds')
print('mae:' ,f'{round(mae,3)} minutes => {round(mae * 60,3)} seconds')
```

```
4308/4308 ━━━━━━━━━━━━━━━━━━━━ 7s 2ms/step
Train Accuracy

mse :  0.001 minutes => 0.059 seconds
rmse :  0.031 minutes => 1.887 seconds
mae: 0.023 minutes => 1.391 seconds


1077/1077 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step
Test Accuracy

mse :  0.001 minutes => 0.068 seconds
rmse :  0.034 minutes => 2.016 seconds
mae: 0.023 minutes => 1.396 seconds
```

NN Model R-Square Score

```
In [106… r2_score(y_test, prediction)

Out[106… 0.9999879677706759
```

NN Model MAPE

```
In [117… mean_absolute_percentage_error(y_test, prediction)

Out[117… 0.0005022063038024766
```

Random forest vs Neural Network models performance metrics comparision

Metric	Neural Network (NN)	Random Forest (RF)
Train MSE	0.001 minutes (≈ 0.059 sec)	0.001 minutes (≈ 0.047 sec)
Train RMSE	0.031 minutes (≈ 1.887 sec)	0.028 minutes (≈ 1.685 sec)
Train MAE	0.023 minutes (≈ 1.391 sec)	0.003 minutes (≈ 0.175 sec)
Test MSE	0.001 minutes (≈ 0.068 sec)	0.007 minutes (≈ 0.395 sec)
Test RMSE	0.034 minutes (≈ 2.016 sec)	0.081 minutes (≈ 4.865 sec)
Test MAE	0.023 minutes (≈ 1.396 sec)	0.008 minutes (≈ 0.478 sec)
R^2 Score	0.9999879677	0.9999233405
MAPE	0.0005020 (0.05%)	0.0131749 (1.31%)

Conclusion

✅ R^2 Score is very high in both models (>0.999), indicating excellent model performance and that both are fitting the data very well.

🧠 NN model has lower test RMSE and lower MAPE (0.05% vs 1.31%), indicating higher prediction accuracy and better generalization.

🌳 RF model has lower training error (especially in MAE), but higher test error, suggesting possible overfitting.

⏱️ In terms of MAE, RF model's test MAE is lower (0.478 sec) compared to NN's (1.396 sec), but this conflicts with MAPE and RMSE — so be cautious.

💡 MAPE is the most intuitive for business — and here, NN clearly wins with significantly lower percentage error.