Comprehensive Analysis & Delivery Time Estimation Strategy for Porter

Problem Statement:

Porter, India's largest intra-city logistics marketplace, needs to **predict accurate food delivery times** for orders placed through its platform.

The company partners with restaurants to deliver food directly to customers, and timely deliveries are critical for:

Customer satisfaction (avoiding frustration from delayed orders)

Operational efficiency (optimal delivery partner allocation)

Competitive advantage (transparent ETAs(Estimated Time of Arrivals) can differentiate Porter from rivals)

Understanding The Data

Start by importing necessary Python libraries like pandas, numpy, matplotlib, seaborn, etc. These help load the dataset, explore the structure, and perform analysis.

```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# importing data and store in variable
PorterData = pd.read_csv("Porter_dataset.csv")

# check dataset dimensions (rows, columns)
f"Shape: {PorterData.shape}"
```

```
'Shape: (197428, 14)'
```

The dataset contains over 1.9 lakh + rows and 14 columns. It holds large-scale delivery/order information.

View the first few rows to understand how the data looks and what kind of values are stored.

```
# top 5 rows of data
PorterData.head()
```

```
   market_id           created_at actual_delivery_time  \
0        1.0  2015-02-06 22:24:17  2015-02-06 23:27:16
1        2.0  2015-02-10 21:49:25  2015-02-10 22:56:29
2        3.0  2015-01-22 20:39:28  2015-01-22 21:09:09
```

```
3        3.0  2015-02-03 21:21:45  2015-02-03 22:13:00
4        3.0  2015-02-15 02:40:36  2015-02-15 03:20:26

                           store_id store_primary_category
order_protocol  \
0  df263d996281d984952c07998dc54358               american
1.0
1  f0ade77b43923b38237db569b016ba25                mexican
2.0
2  f0ade77b43923b38237db569b016ba25                    NaN
1.0
3  f0ade77b43923b38237db569b016ba25                    NaN
1.0
4  f0ade77b43923b38237db569b016ba25                    NaN
1.0

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

   total_onshift_partners  total_busy_partners
total_outstanding_orders
0                    33.0                 14.0
21.0
1                     1.0                  2.0
2.0
2                     1.0                  0.0
0.0
3                     1.0                  1.0
2.0
4                     6.0                  6.0
9.0
```

Column Description,

1. market_id: Integer ID for the market where the restaurant is located
2. created_at: Timestamp at which the order was placed
3. actual_delivery_time: Timestamp when the order was delivered
4. store_primary_category: Category of the restaurant

5. order_protocol: Integer code value for the order protocol (e.g., through Porter,call to restaurant, pre-booked, third-party, etc.)
6. total_items: Total number of items in the order
7. subtotal: Final price of the order
8. num_distinct_items: Number of distinct items in the order
9. min_item_price: Price of the cheapest item in the order
10. max_item_price: Price of the most expensive item in the order
11. total_onshift_partners: Number of delivery partners on duty when the order was placed
12. total_busy_partners: Number of delivery partners attending to other tasks
13. total_outstanding_orders: Total number of orders to be fulfilled at that moment

Checking Column Variables and Types.

```
# printing information about the data
PorterData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   market_id               196441 non-null  float64
 1   created_at              197428 non-null  object 
 2   actual_delivery_time    197421 non-null  object 
 3   store_id                197428 non-null  object 
 4   store_primary_category  192668 non-null  object 
 5   order_protocol          196433 non-null  float64
 6   total_items             197428 non-null  int64  
 7   subtotal                197428 non-null  int64  
 8   num_distinct_items      197428 non-null  int64  
 9   min_item_price          197428 non-null  int64  
 10  max_item_price          197428 non-null  int64  
 11  total_onshift_partners  181166 non-null  float64
 12  total_busy_partners     181166 non-null  float64
 13  total_outstanding_orders  181166 non-null  float64
dtypes: float64(5), int64(5), object(4)
memory usage: 21.1+ MB
```

Look at the names of all columns and their current data types (like object, int, float). Some columns like created_at need to be converted to date format for better analysis.

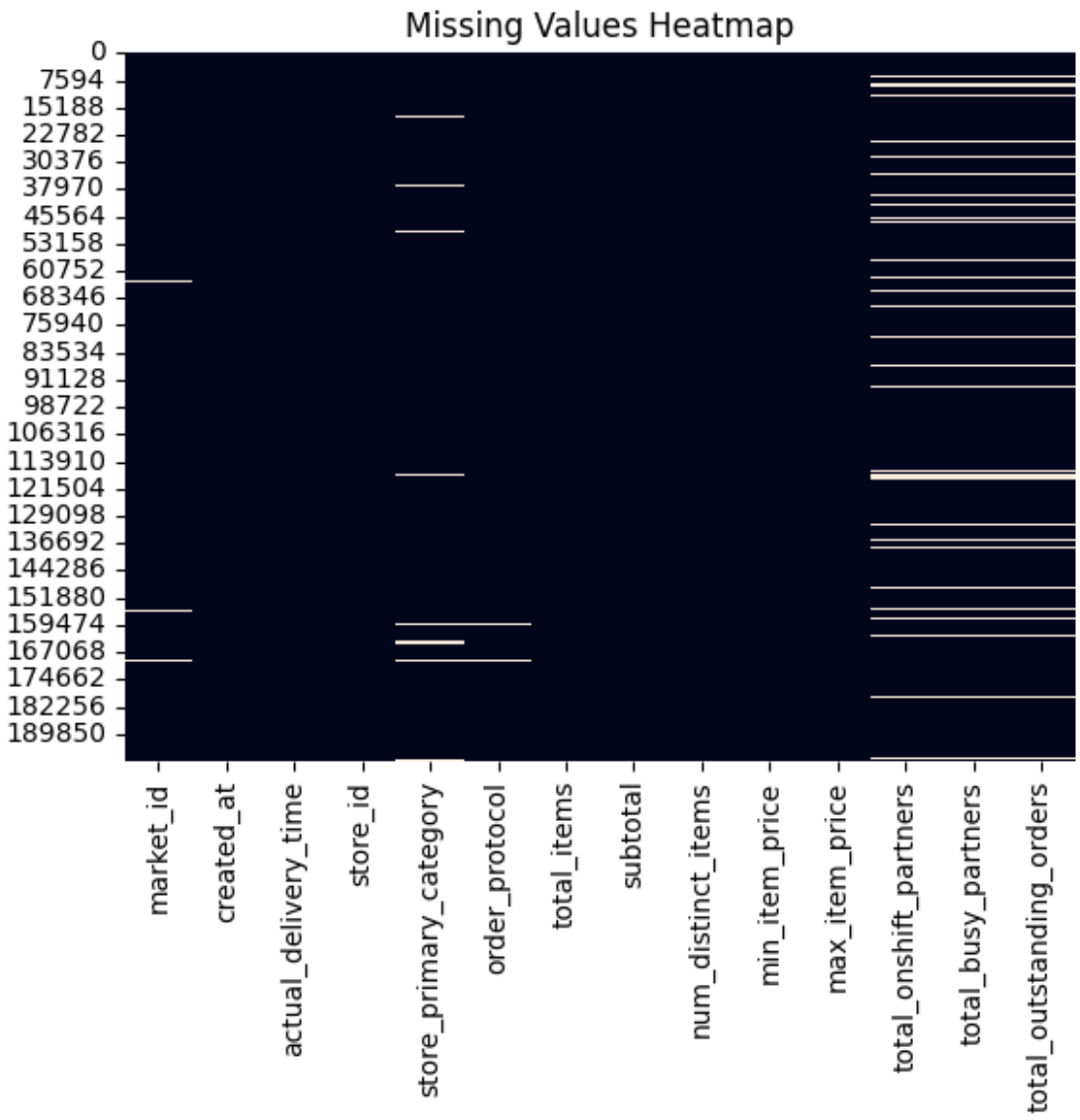
Check for missing values in each column to understand where data is incomplete.

```
# total missing values per column
PorterData.isnull().sum()
```

```
market_id                     987
created_at                      0
actual_delivery_time            7
```

```
store_id                        0
store_primary_category       4760
order_protocol                995
total_items                     0
subtotal                        0
num_distinct_items              0
min_item_price                  0
max_item_price                  0
total_onshift_partners      16262
total_busy_partners         16262
total_outstanding_orders    16262
dtype: int64
```

```
# visualize missing data
sns.heatmap(PorterData.isnull(), cbar=False)
plt.title("Missing Values Heatmap")
plt.show()
```




```
# check null percentage
PorterData.isnull().mean() * 100
```

```
market_id                     0.499929
created_at                    0.000000
actual_delivery_time          0.003546
store_id                      0.000000
store_primary_category        2.411006
order_protocol                0.503981
total_items                   0.000000
subtotal                      0.000000
num_distinct_items            0.000000
min_item_price                0.000000
max_item_price                0.000000
total_onshift_partners        8.236927
```

```
total_busy_partners         8.236927
total_outstanding_orders    8.236927
dtype: float64
```

Columns like total_onshift_partners, total_busy_partners, and total_outstanding_orders have a large number of nulls.

Other columns like market_id, actual_delivery_time, store_delivery_time, store_primary_category, and order_protocol have fewer null values.

Exploratory Data Analysis (EDA)

View summary statistics (mean, median, max, min, etc.) for each column to get a quick understanding of the dataset

```
# generating descriptive statistics
PorterData.describe()
```

```
           market_id  order_protocol    total_items       subtotal  \
count  196441.000000   196433.000000  197428.000000  197428.000000
mean        2.978706        2.882352       3.196391    2682.331402
std         1.524867        1.503771       2.666546    1823.093688
min         1.000000        1.000000       1.000000       0.000000
25%         2.000000        1.000000       2.000000    1400.000000
50%         3.000000        3.000000       3.000000    2200.000000
75%         4.000000        4.000000       4.000000    3395.000000
max         6.000000        7.000000     411.000000   27100.000000

       num_distinct_items  min_item_price  max_item_price  \
count       197428.000000   197428.000000   197428.000000
mean             2.670791      686.218470     1159.588630
std              1.630255      522.038648      558.411377
min              1.000000      -86.000000        0.000000
25%              1.000000      299.000000      800.000000
50%              2.000000      595.000000     1095.000000
75%              3.000000      949.000000     1395.000000
max             20.000000    14700.000000    14700.000000

       total_onshift_partners  total_busy_partners
total_outstanding_orders
count           181166.000000        181166.000000
181166.000000
mean                44.808093            41.739747
58.050065
std                 34.526783            32.145733
52.661830
min                 -4.000000            -5.000000                  -
6.000000
25%                 17.000000            15.000000
17.000000
```

```
50%                   37.000000             34.000000
41.000000
75%                   65.000000             62.000000
85.000000
max                  171.000000            154.000000
285.000000
```

Total Items Column:

- 75% of orders have 4 or fewer items.
- Maximum items go above 400 – likely outliers.

Item Price Columns (Min & Max):

- Both have unusually high values (up to 14,700), which seems unrealistic for a single item.
- Some values are negative, suggesting data entry mistakes.

Subtotal Column:

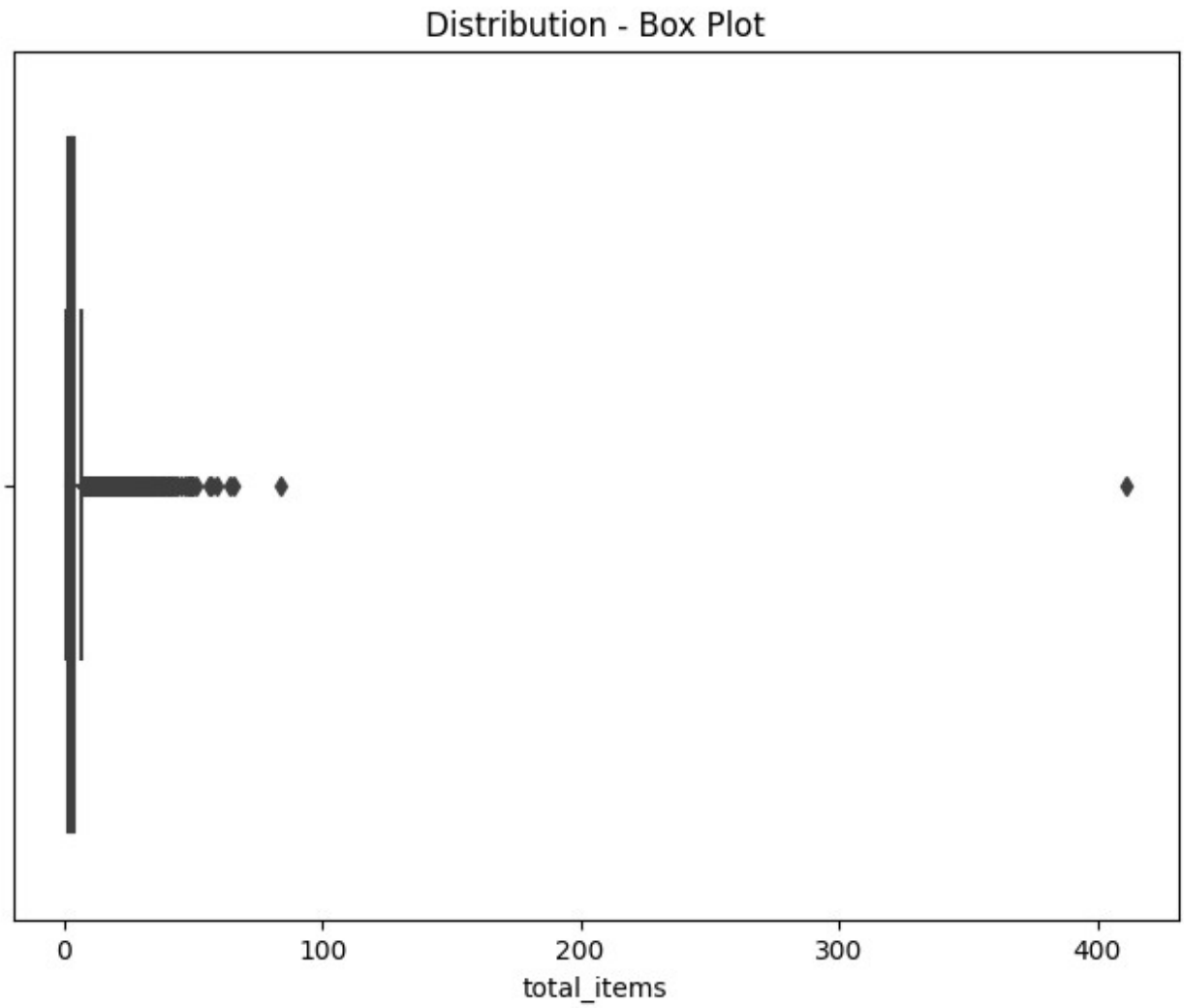
- Some subtotal values are zero or unusually high (up to 27,100).
- Could be due to missing transaction values or calculation errors.

Logical Inconsistencies:

- In a few rows, total_busy_partners is greater than total_onshift_partners, which shouldn't happen.
- In some rows, all three columns (onshift, busy, outstanding_orders) are zero, but still, there's a delivery recorded. That's not possible.

Outlier & Pattern Detection

```
# boxplot for total_items column
plt.figure(figsize=(8, 6))
sns.boxplot(x=PorterData['total_items'])
plt.title('Distribution - Box Plot')
plt.show()
```




```
PorterData[(PorterData['total_items'] > 50)]
```

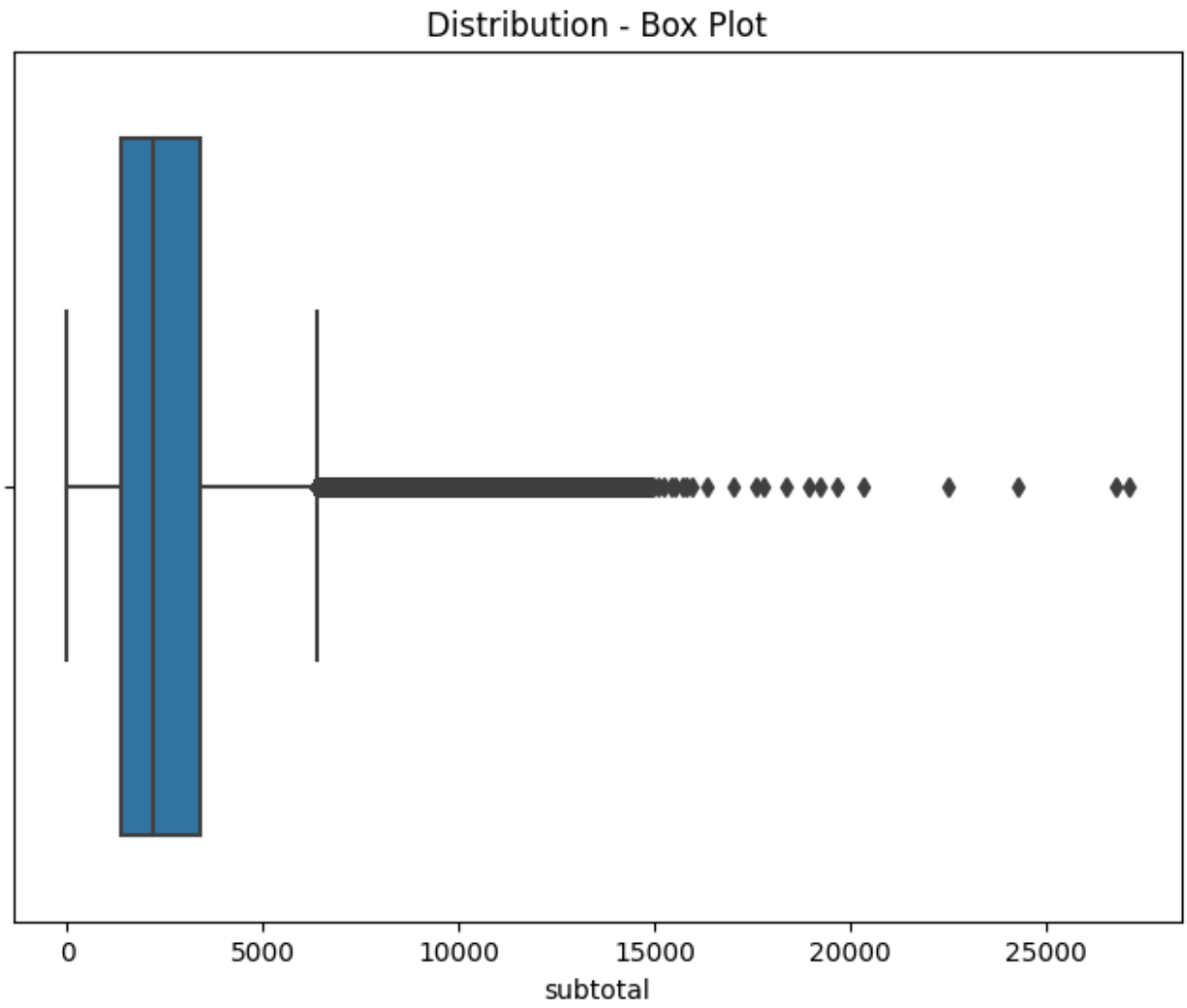
	market_id	created_at	actual_delivery_time	\
11055	2.0	2015-01-22 01:07:03	2015-01-22 01:55:43	
15053	4.0	2015-01-26 22:48:56	2015-01-26 23:28:02	
47231	2.0	2015-02-06 00:42:39	2015-02-06 01:33:34	
75577	1.0	2015-01-23 04:33:00	2015-01-23 05:07:45	
105348	3.0	2015-01-23 15:46:35	2015-01-23 16:56:32	
182223	6.0	2015-02-15 19:39:32	2015-02-15 20:54:10	
182796	2.0	2015-02-17 05:45:05	2015-02-17 06:14:01	
182800	2.0	2015-02-18 05:42:03	2015-02-18 06:11:25	

	store_id	store_primary_category	\
11055	c8512d142a2d849725f31a9a7a361ab9	fast	
15053	4a308d84cdd04aa2015bbe13622d5d7c	fast	
47231	c156cea027720c227089e679b3ae9d1b	fast	
75577	452e91de642a8e9c43121664d5d3c05c	fast	
105348	d14c2267d848abeb81fd590f371d39bd	fast	

```
182223  e50372d3fee4eadec9c42aa6528097cc                    fast
182796  42a3ddf2e1df611a280d556f1c81996a                    fast
182800  42a3ddf2e1df611a280d556f1c81996a                    fast

        order_protocol  total_items  subtotal  num_distinct_items  \
11055              4.0           56      2317                   7
15053              4.0           57       858                   5
47231              4.0          411      3115                   5
75577              4.0           59      2911                   3
105348             4.0           51      1343                   4
182223             4.0           84      1016                   4
182796             4.0           64      1166                   4
182800             4.0           66      1634                   6

        min_item_price  max_item_price  total_onshift_partners  \
11055                0             289                    35.0
15053                0             229                    25.0
47231                0             299                    35.0
75577                0             329                    25.0
105348               0             169                     3.0
182223               0             289                     NaN
182796               0             499                    28.0
182800               0             499                    30.0

        total_busy_partners  total_outstanding_orders
11055                  27.0                      26.0
15053                  54.0                      19.0
47231                  35.0                      39.0
75577                  19.0                      20.0
105348                  1.0                       2.0
182223                  NaN                       NaN
182796                 19.0                      21.0
182800                 28.0                      32.0
```

Orders with more than 50 items usually come from stores with 'fast' primary category and order_protocol = 4.

```
# boxplot for subtotal column
plt.figure(figsize=(8, 6))
sns.boxplot(x=PorterData['subtotal'])
plt.title('Distribution - Box Plot')
plt.show()
```




```
PorterData[(PorterData['subtotal'] > 15000)]
```

	market_id	created_at	actual_delivery_time	\
8994	1.0	2015-01-27 20:08:26	2015-01-27 21:21:37	
19966	4.0	2015-02-16 01:12:30	2015-02-16 02:19:05	
37806	4.0	2015-02-13 21:20:15	2015-02-13 22:23:24	
44705	4.0	2015-01-29 21:02:27	2015-01-29 21:42:20	
53072	1.0	2015-02-05 01:02:41	2015-02-05 02:25:20	
54486	4.0	2015-01-22 23:59:18	2015-01-23 00:39:52	
69243	1.0	2015-01-29 02:33:00	2015-01-29 04:03:25	
71316	6.0	2015-02-12 00:50:03	2015-02-12 02:07:01	
71702	6.0	2015-01-27 02:22:26	2015-01-27 03:47:09	
73129	4.0	2015-02-10 01:42:46	2015-02-10 02:36:05	
82763	6.0	2015-02-12 01:28:39	2015-02-12 03:39:57	
83330	6.0	2015-01-28 02:12:01	2015-01-28 03:50:48	
96456	1.0	2015-02-04 19:21:43	2015-02-04 20:20:25	
140458	4.0	2015-02-18 01:16:47	2015-02-18 02:29:14	
152185	4.0	2015-01-22 00:09:59	2015-01-22 01:29:21	

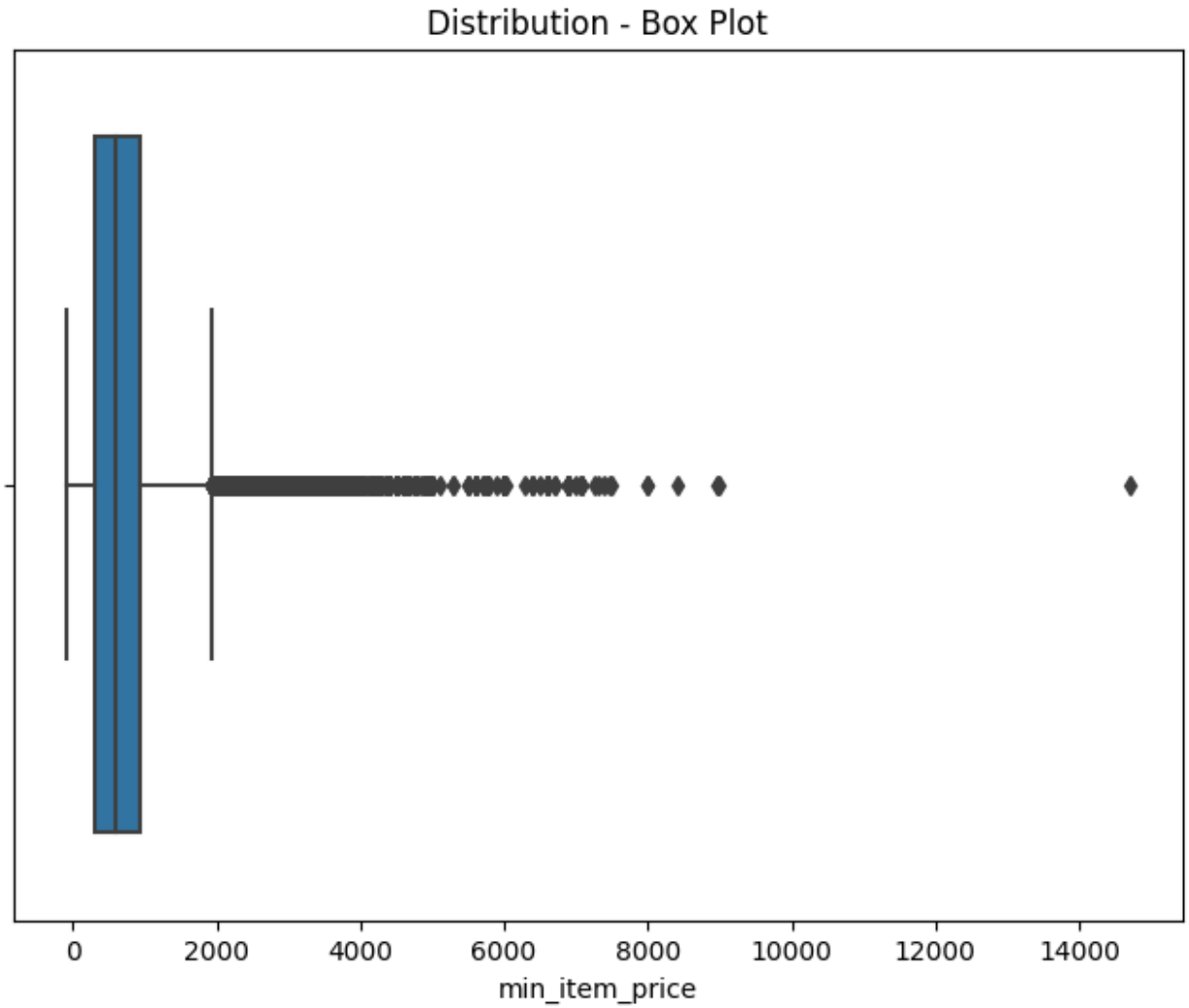
```
171276        4.0  2015-02-02 17:46:21  2015-02-02 18:46:36
173912        6.0  2015-02-13 16:29:21  2015-02-13 18:27:58
176874        4.0  2015-02-13 20:44:02  2015-02-13 22:00:09
181734        2.0  2015-02-16 02:22:47  2015-02-16 03:42:16
188378        1.0  2015-01-27 02:11:51  2015-01-27 03:35:26

                                 store_id store_primary_category  \
8994    652cf38361a209088302ba2b8b7f51e0                   salad
19966   03afdbd66e7929b125f8597834fa83a4                   sushi
37806   b9a25e422ba96f7572089a00b838c3f8                  indian
44705   09fb05dd477d4ae6479985ca56c5a12d                 italian
53072   28b60a16b55fd531047c0c958ce14b95                   pizza
54486   ccc0aa1b81bf81e16c676ddb977c5881                 seafood
69243   7f6b550688de8db0290009ba6abd673c                     NaN
71316   9cea886b9f44a3c2df1163730ab64994                   greek
71702   c900fe92840c527a0c54f28640c2f254                catering
73129   dbe40ccd11b1fb869099e58e00076027              vietnamese
82763   f58c9875ac84dfe1fbe91b918773d050                japanese
83330   f58c9875ac84dfe1fbe91b918773d050                japanese
96456   748ba69d3e8d1af87f84fee909eef339                 italian
140458  e2ef524fbf3d9fe611d5a8e90fefdc9c                american
152185  9cea10c7ff109c6e61727a0d45492ead              vietnamese
171276  3dd48ab31d016ffcbf3314df2b3cb9ce                   greek
173912  4607f7fff0dce694258e1c637512aa9d                sandwich
176874  f1b6f2857fb6d44dd73c7041e0aa0f19         middle-eastern
181734  91665c93b72f55b2e4f1048fc8289d04                smoothie
188378  fd2c5e4680d9a01dba3aada5ece22270              vietnamese

        order_protocol  total_items  subtotal  num_distinct_items  \
8994               1.0           21     18370                  11
19966              3.0           19     15505                  12
37806              3.0           28     15238                   9
44705              3.0           10     15089                   8
53072              5.0           20     20350                  14
54486              5.0           14     16350                  11
69243              4.0           24     17008                  13
71316              5.0           11     15800                   9
71702              3.0           18     24300                  14
73129              1.0           11     19250                  11
82763              1.0           27     27100                  19
83330              1.0           20     19650                  12
96456              5.0           17     22500                  14
140458             3.0            6     15435                   6
152185             5.0           18     15710                  14
171276             3.0           16     18920                   4
173912             4.0           17     17810                  14
176874             5.0           14     17600                  13
181734             2.0            4     15960                   2
188378             5.0           25     26800                  13
```

```
        min_item_price  max_item_price  total_onshift_partners  \
8994               350            1380                    13.0
19966              310            2400                    84.0
37806              245            1399                    38.0
44705             1130            1710                    80.0
53072              450            1250                     7.0
54486              300            1650                    37.0
69243               61            1556                    55.0
71316              350            2000                    16.0
71702              600            2200                     NaN
73129              200            3100                    75.0
82763              200            3000                     NaN
83330              200            2000                     NaN
96456              450            1750                    18.0
140458             795            3570                    76.0
152185             265            1275                    33.0
171276             695            1695                    10.0
173912             375            1295                     NaN
176874             400            1700                    63.0
181734            3990            3990                    56.0
188378             200            1800                    13.0

        total_busy_partners  total_outstanding_orders
8994                   13.0                      12.0
19966                  80.0                     142.0
37806                  20.0                      22.0
44705                  64.0                      70.0
53072                   8.0                      11.0
54486                  18.0                      19.0
69243                  49.0                      63.0
71316                   6.0                       5.0
71702                   NaN                       NaN
73129                  67.0                      92.0
82763                   NaN                       NaN
83330                   NaN                       NaN
96456                  17.0                      21.0
140458                 75.0                     105.0
152185                 14.0                      14.0
171276                 11.0                      14.0
173912                  NaN                       NaN
176874                 63.0                      89.0
181734                 53.0                      92.0
188378                 14.0                      17.0
```

Oddly, even when items are more, subtotal is sometimes less, which looks unrealistic.

```
# boxplot for min_item_price column
plt.figure(figsize=(8, 6))
sns.boxplot(x=PorterData['min_item_price'])
```

```
plt.title('Distribution - Box Plot')
plt.show()
```



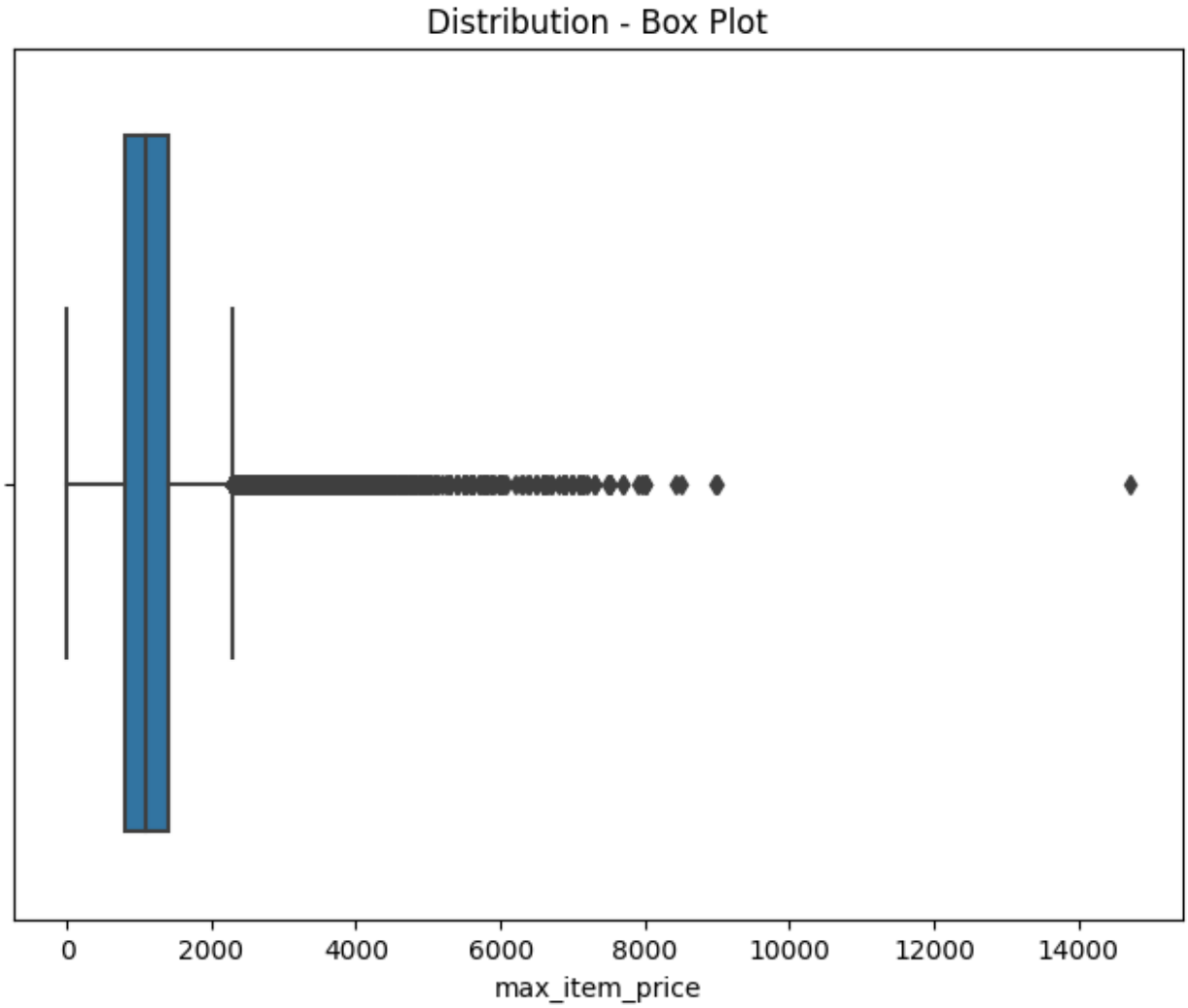

```
PorterData[(PorterData['min_item_price'] > 5000)]
```

	market_id	created_at	actual_delivery_time	\
999	2.0	2015-01-28 00:45:38	2015-01-28 01:15:26	
6814	2.0	2015-01-24 05:10:20	2015-01-24 06:05:35	
7545	6.0	2015-02-13 18:34:39	2015-02-13 19:36:52	
7899	6.0	2015-02-07 03:52:47	2015-02-07 05:11:13	
13548	4.0	2015-01-30 22:16:08	2015-01-30 23:06:16	
...	...	...	...	
184000	2.0	2015-02-03 01:23:48	2015-02-03 02:18:14	
188815	5.0	2015-01-22 02:27:31	2015-01-22 03:08:34	
190087	2.0	2015-02-01 03:46:30	2015-02-01 04:33:39	
190213	2.0	2015-01-29 02:56:02	2015-01-29 03:39:58	
195513	2.0	2015-02-14 03:38:56	2015-02-14 04:40:42	

```
                                  store_id store_primary_category  \
999     0bb0846327772451045bd30dd347821b               barbecue
6814    140f6969d5213fd0ece03148e62e461e               japanese
7545    f3a493fb0b6dc6357b9d89d6bdc1f2af                dessert
7899    8cbd005a556ccd4211ce43f309bc0eac                   thai
13548   e723e2ae3e04e8028e119ee592e81974                    NaN
...                                  ...                    ...
184000  777125b977ddc0317d0533782d3c27b5                chinese
188815  15b33319db65d343906d085ba0500783                dessert
190087  9f319422ca17b1082ea49820353f14ab               american
190213  9f319422ca17b1082ea49820353f14ab               american
195513  b783acd4479bf1b8a981bb023b363043               american

        order_protocol  total_items  subtotal  num_distinct_items  \
999                3.0            1      7475                   1
6814               2.0            1      6400                   1
7545               2.0            1      7080                   1
7899               3.0            1      9900                   1
13548              1.0            1      7299                   1
...                ...          ...       ...                 ...
184000             3.0            1      6500                   1
188815             2.0            1      6000                   1
190087             3.0            1      7029                   1
190213             3.0            1      6889                   1
195513             1.0            1      8959                   1

        min_item_price  max_item_price  total_onshift_partners  \
999               7475            7475                    15.0
6814              6400            6400                    59.0
7545              7080            7080                     NaN
7899              6000            6000                     NaN
13548             7299            7299                     7.0
...                ...             ...                     ...
184000            6500            6500                    49.0
188815            6000            6000                    18.0
190087            6889            6889                   101.0
190213            6889            6889                   103.0
195513            8959            8959                   143.0

        total_busy_partners  total_outstanding_orders
999                    14.0                      14.0
6814                   69.0                     111.0
7545                    NaN                       NaN
7899                    NaN                       NaN
13548                   7.0                      13.0
...                     ...                       ...
184000                 31.0                      55.0
188815                 13.0                      20.0
190087                102.0                     184.0
```

```
190213                  87.0                        127.0
195513                 131.0                        274.0

[72 rows x 14 columns]
```

```
# boxplot for max_item_price column
plt.figure(figsize=(8, 6))
sns.boxplot(x=PorterData['max_item_price'])
plt.title('Distribution - Box Plot')
plt.show()
```




```
PorterData[(PorterData['max_item_price'] > 6000)]
```

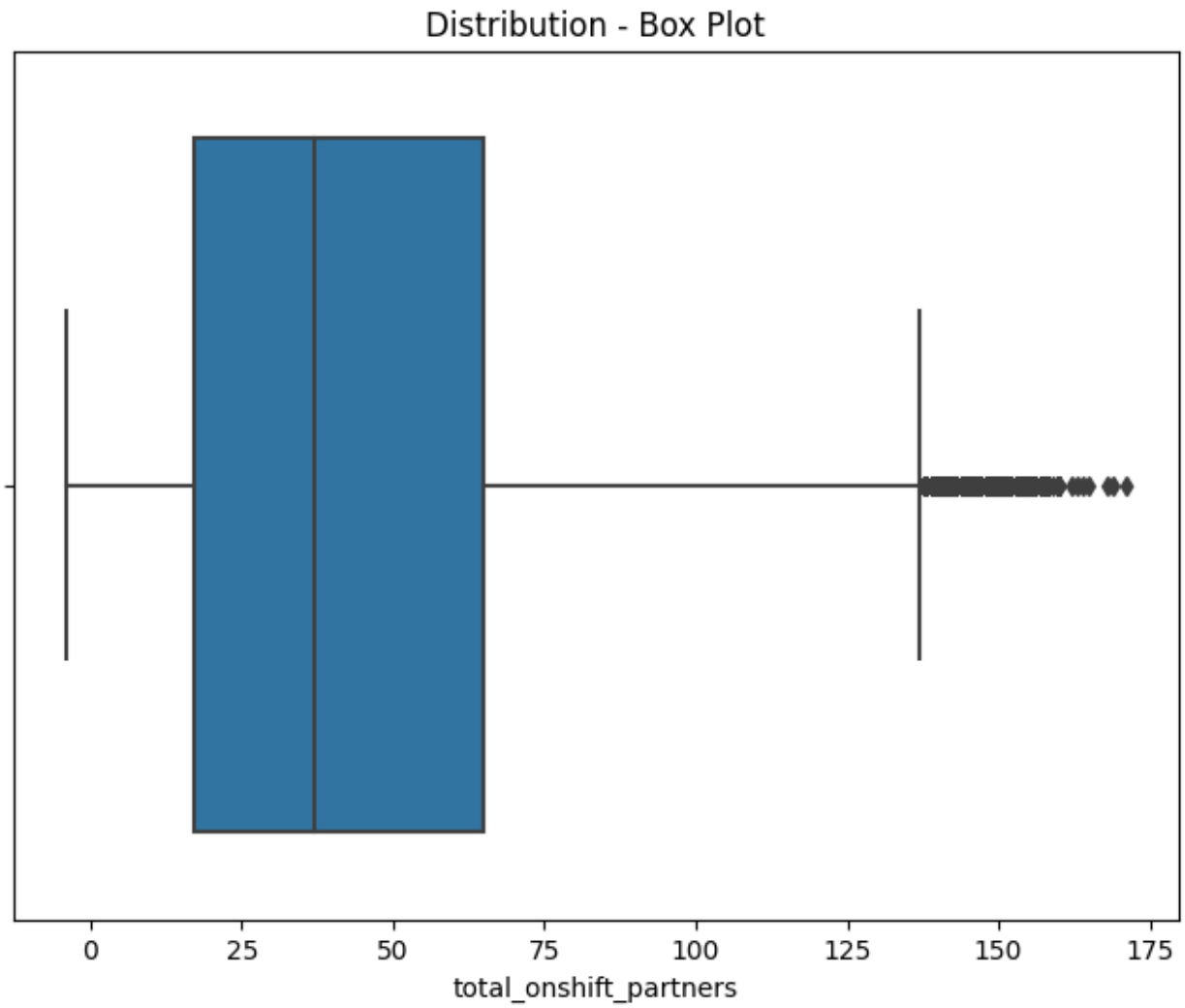
```
        market_id           created_at actual_delivery_time  \
999           2.0  2015-01-28 00:45:38  2015-01-28 01:15:26
6814          2.0  2015-01-24 05:10:20  2015-01-24 06:05:35
7545          6.0  2015-02-13 18:34:39  2015-02-13 19:36:52
```

```
12790         2.0  2015-02-14 20:17:26  2015-02-14 21:06:35
13548         4.0  2015-01-30 22:16:08  2015-01-30 23:06:16
...           ...                  ...                  ...
182045        3.0  2015-01-26 01:21:21  2015-01-26 02:38:45
184000        2.0  2015-02-03 01:23:48  2015-02-03 02:18:14
190087        2.0  2015-02-01 03:46:30  2015-02-01 04:33:39
190213        2.0  2015-01-29 02:56:02  2015-01-29 03:39:58
195513        2.0  2015-02-14 03:38:56  2015-02-14 04:40:42

                                store_id store_primary_category  \
999     0bb0846327772451045bd30dd347821b               barbecue
6814    140f6969d5213fd0ece03148e62e461e               japanese
7545    f3a493fb0b6dc6357b9d89d6bdc1f2af                dessert
12790   9854d7afce413aa13cd0a1d39d0bcec5               american
13548   e723e2ae3e04e8028e119ee592e81974                    NaN
...                                 ...                    ...
182045  38181d991caac98be8fb2ecb8bd0f166               american
184000  777125b977ddc0317d0533782d3c27b5                chinese
190087  9f319422ca17b1082ea49820353f14ab               american
190213  9f319422ca17b1082ea49820353f14ab               american
195513  b783acd4479bf1b8a981bb023b363043               american

        order_protocol  total_items  subtotal  num_distinct_items  \
999                3.0            1      7475                   1
6814               2.0            1      6400                   1
7545               2.0            1      7080                   1
12790              1.0            2      8048                   2
13548              1.0            1      7299                   1
...                ...          ...       ...                 ...
182045             4.0            1      6699                   1
184000             3.0            1      6500                   1
190087             3.0            1      7029                   1
190213             3.0            1      6889                   1
195513             1.0            1      8959                   1

        min_item_price  max_item_price  total_onshift_partners  \
999               7475            7475                    15.0
6814              6400            6400                    59.0
7545              7080            7080                     NaN
12790               79            7699                    70.0
13548             7299            7299                     7.0
...                ...             ...                     ...
182045            6699            6699                    41.0
184000            6500            6500                    49.0
190087            6889            6889                   101.0
190213            6889            6889                   103.0
195513            8959            8959                   143.0

        total_busy_partners  total_outstanding_orders
999                    14.0                      14.0
```

```
6814                    69.0                    111.0
7545                     NaN                     NaN
12790                   66.0                     71.0
13548                    7.0                     13.0
...                      ...                      ...
182045                  40.0                     47.0
184000                  31.0                     55.0
190087                 102.0                    184.0
190213                  87.0                    127.0
195513                 131.0                    274.0

[71 rows x 14 columns]
```

High max and min item prices (14,700) are seen for the breakfast store category, often with only one item in the order.

```
# boxplot for total_onshift_partners column
plt.figure(figsize=(8, 6))
sns.boxplot(x=PorterData['total_onshift_partners'])
plt.title('Distribution - Box Plot')
plt.show()
```

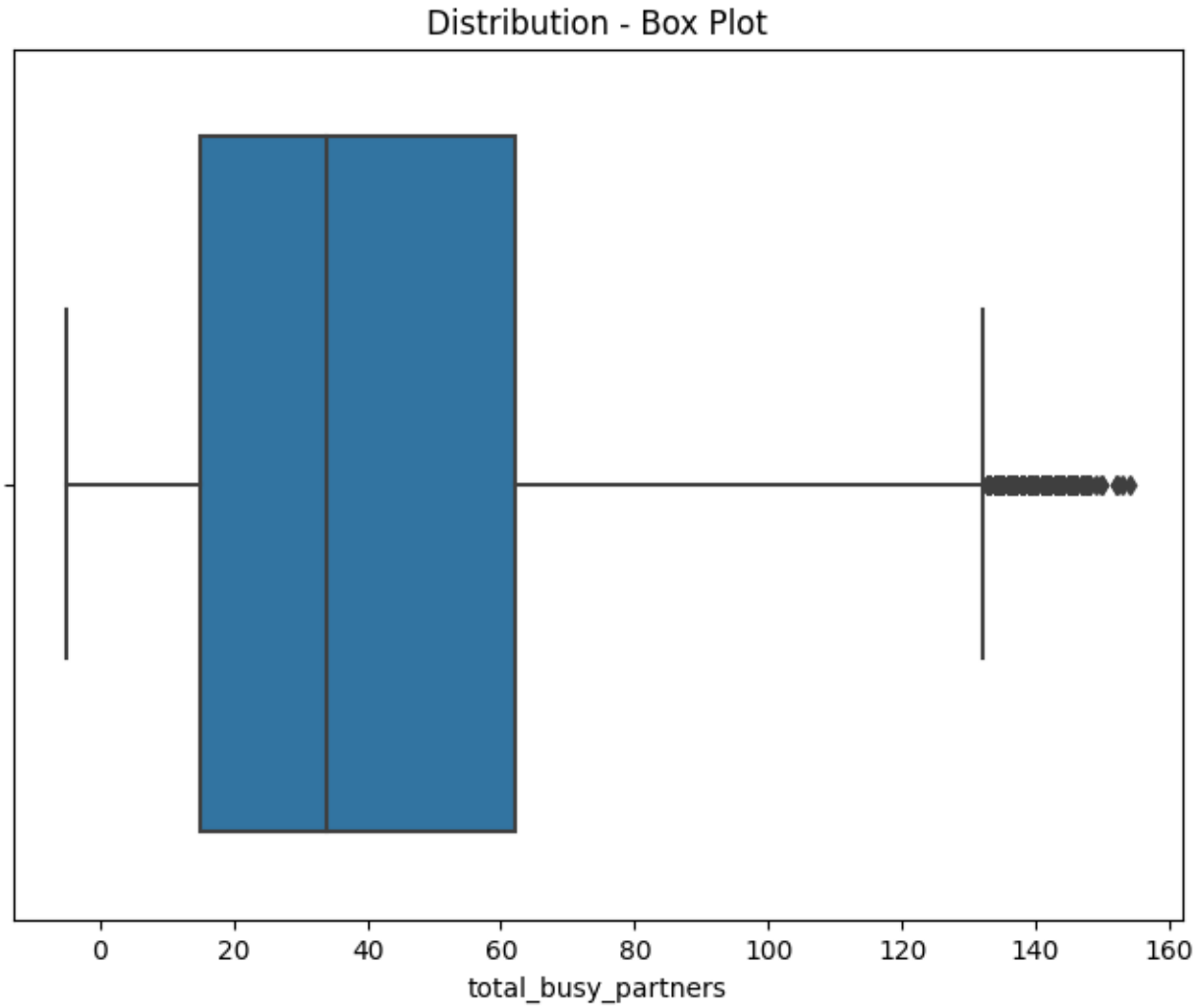



```
PorterData[(PorterData['total_onshift_partners'] > 150)]
```

	market_id	created_at	actual_delivery_time	store_id	store_primary_category
1679	2.0	2015-02-07 02:45:20	2015-02-07 03:25:44	70821a40b06f8751781d5a895357da67	chinese
3328	2.0	2015-02-06 02:27:12	2015-02-06 03:00:02	150784e5fbeb562400a0cd1111471d6a	american
4101	2.0	2015-02-14 02:56:43	2015-02-14 04:08:33		
5780	2.0	2015-02-07 02:56:37	2015-02-07 03:38:35		
5930	2.0	2015-02-07 02:46:03	2015-02-07 03:27:42		
...	...	...	...		
194419	2.0	2015-02-07 02:44:51	2015-02-07 04:05:32		
195448	2.0	2015-02-06 02:42:42	2015-02-06 03:27:28		
195971	2.0	2015-02-07 02:57:05	2015-02-07 03:36:47		
196128	2.0	2015-02-07 02:31:28	2015-02-07 03:15:44		
196180	2.0	2015-02-07 02:58:55	2015-02-07 03:42:06		

```
4101    150784e5fbeb562400a0cd1111471d6a                american
5780    57c0531e13f40b91b3b0f1a30b529a1d                   pizza
5930    115f89503138416a242f40fb7d7f338e                   pasta
...                                  ...                     ...
194419  371bce7dc83817b7893bcdeed13799b5                   pizza
195448  b783acd4479bf1b8a981bb023b363043                american
195971  84d9ee44e457ddef7f2c4f25dc8fa865                sandwich
196128  84d9ee44e457ddef7f2c4f25dc8fa865                sandwich
196180  84d9ee44e457ddef7f2c4f25dc8fa865                sandwich

        order_protocol  total_items  subtotal  num_distinct_items  \
1679               1.0            4      3460                   4
3328               1.0            1       949                   1
4101               1.0            1      1899                   1
5780               5.0            1      1645                   1
5930               3.0            4      2265                   4
...                ...          ...       ...                 ...
194419             3.0            4      4580                   2
195448             1.0            1      1619                   1
195971             5.0            4      2261                   3
196128             5.0            2      2301                   2
196180             5.0            4      2700                   3

        min_item_price  max_item_price  total_onshift_partners  \
1679               595            1275                   152.0
3328               949             949                   155.0
4101              1899            1899                   151.0
5780              1645            1645                   156.0
5930               250             925                   152.0
...                ...             ...                     ...
194419             995            1295                   154.0
195448            1619            1619                   157.0
195971             100             999                   156.0
196128             999             999                   151.0
196180             295            1111                   156.0

        total_busy_partners  total_outstanding_orders
1679                  131.0                     211.0
3328                   76.0                     114.0
4101                  127.0                     231.0
5780                  137.0                     215.0
5930                  131.0                     211.0
...                     ...                       ...
194419                129.0                     199.0
195448                 87.0                     138.0
195971                137.0                     215.0
196128                114.0                     195.0
196180                137.0                     215.0

[330 rows x 14 columns]
```

```
# boxplot for total_busy_partners column
plt.figure(figsize=(8, 6))
sns.boxplot(x=PorterData['total_busy_partners'])
plt.title('Distribution - Box Plot')
plt.show()
```



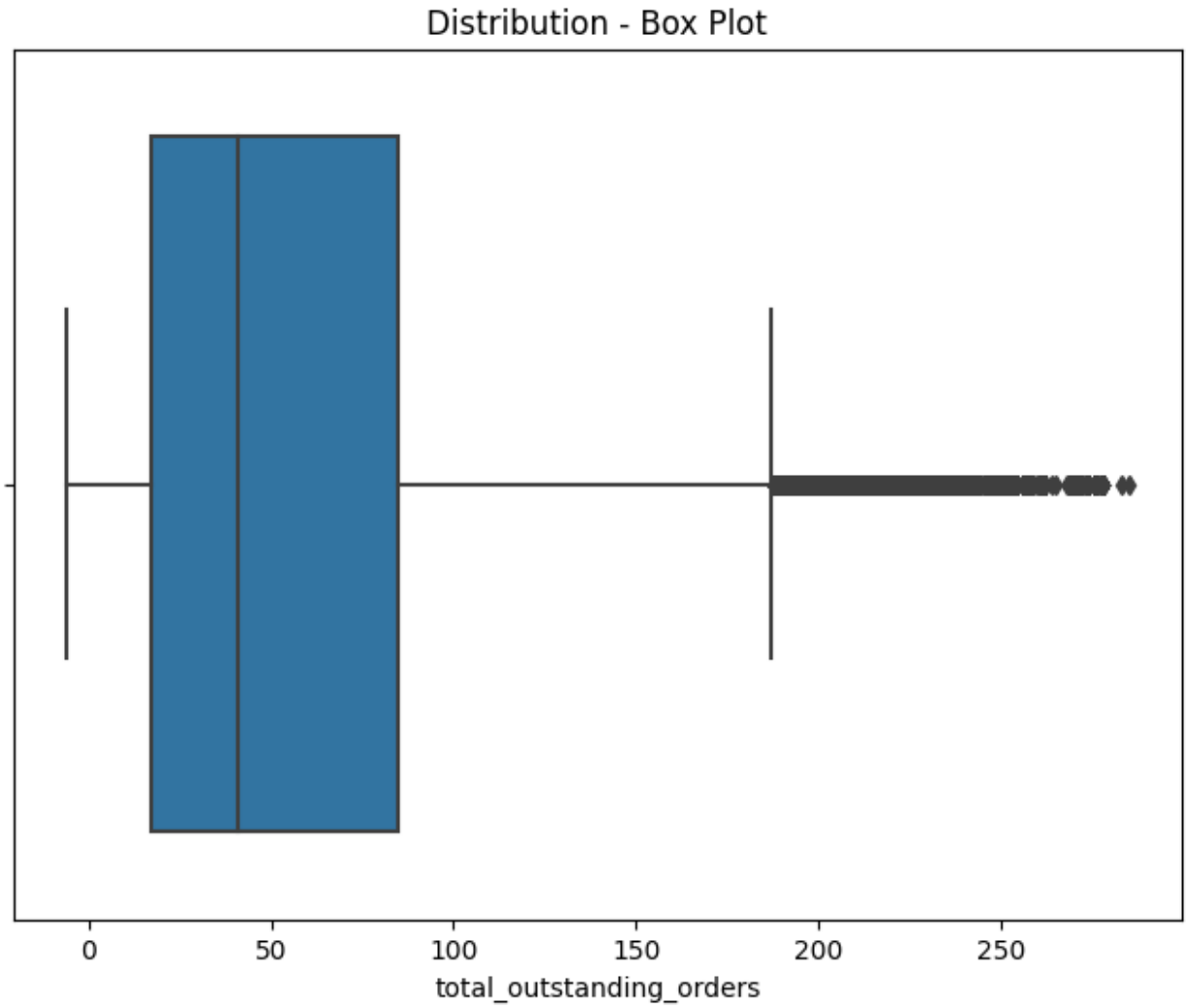

```
PorterData[(PorterData['total_busy_partners'] > 140)]
```

	market_id	created_at	actual_delivery_time	\
1266	4.0	2015-01-25 02:26:42	2015-01-25 03:07:42	
1464	4.0	2015-01-25 02:14:50	2015-01-25 02:51:20	
1843	4.0	2015-01-25 03:00:26	2015-01-25 04:14:02	
1853	4.0	2015-01-25 02:39:24	2015-01-25 03:36:16	
1905	4.0	2015-01-25 02:38:29	2015-01-25 03:39:37	
...	...	...	...	
196453	2.0	2015-02-14 03:56:01	2015-02-14 04:48:35	
197097	4.0	2015-01-25 03:14:04	2015-01-25 04:10:35	
197104	4.0	2015-01-25 02:20:30	2015-01-25 02:56:05	

```
197328        4.0  2015-01-25 02:42:49  2015-01-25 03:29:09
197338        4.0  2015-01-25 03:13:33  2015-01-25 04:24:24

                                 store_id store_primary_category  \
1266    f7177163c833dff4b38fc8d2872f1ec6                 dessert
1464    6ea9ab1baa0efb9e19094440c317e21b                 italian
1843    3f088ebeda03513be71d34d214291986                 mexican
1853    3f088ebeda03513be71d34d214291986                 mexican
1905    6ea9ab1baa0efb9e19094440c317e21b                 italian
...                                  ...                     ...
196453  0d1a9651497a38d8b1c3871c84528bd4                   other
197097  17e62166fc8586dfa4d1bc0e1742c08b              vietnamese
197104  17e62166fc8586dfa4d1bc0e1742c08b              vietnamese
197328  4b04a686b0ad13dce35fa99fa4161c65                 italian
197338  4b04a686b0ad13dce35fa99fa4161c65                 italian

        order_protocol  total_items  subtotal  num_distinct_items  \
1266               5.0            4      4035                   4
1464               3.0            2      3830                   2
1843               2.0            1      1400                   1
1853               2.0            1      1400                   1
1905               3.0            2      1970                   2
...                ...          ...       ...                 ...
196453             3.0            2      1549                   2
197097             5.0            1      1325                   1
197104             5.0            1       795                   1
197328             2.0            2      3690                   2
197338             2.0            2      4420                   2

        min_item_price  max_item_price  total_onshift_partners  \
1266               795            1295                   103.0
1464              1575            2248                   103.0
1843              1400            1400                   108.0
1853              1400            1400                   106.0
1905               575            1395                   106.0
...                ...             ...                     ...
196453             305            1042                   130.0
197097             750             750                   103.0
197104             795             795                   105.0
197328            1148            1198                   106.0
197338            1825            2595                   103.0

        total_busy_partners  total_outstanding_orders
1266                  143.0                     136.0
1464                  146.0                     134.0
1843                  148.0                     124.0
1853                  146.0                     138.0
1905                  146.0                     138.0
...                     ...                       ...
196453                141.0                     260.0
```

```
197097                  147.0                         120.0
197104                  142.0                         122.0
197328                  144.0                         137.0
197338                  147.0                         120.0

[219 rows x 14 columns]
```

```
# boxplot for total_outstanding_orders column
plt.figure(figsize=(8, 6))
sns.boxplot(x=PorterData['total_outstanding_orders'])
plt.title('Distribution - Box Plot')
plt.show()
```




```
PorterData[(PorterData['total_outstanding_orders'] > 250)]
```

```
        market_id           created_at actual_delivery_time  \
170           4.0  2015-02-07 02:34:41  2015-02-07 03:20:31
```

```
1680          2.0  2015-02-14 03:26:43  2015-02-14 04:08:55
2319          2.0  2015-02-14 03:44:27  2015-02-14 04:51:51
3000          4.0  2015-02-07 02:39:13  2015-02-07 03:51:31
3004          4.0  2015-02-07 02:31:31  2015-02-07 04:03:10
...           ...                  ...                  ...
195905        2.0  2015-01-31 03:05:11  2015-01-31 04:14:18
196141        2.0  2015-01-24 03:09:38  2015-01-24 04:27:04
196453        2.0  2015-02-14 03:56:01  2015-02-14 04:48:35
196469        2.0  2015-02-14 03:18:25  2015-02-14 04:01:40
196506        2.0  2015-02-14 03:54:01  2015-02-14 05:14:27

                                store_id store_primary_category  \
170     a87ff679a2f3e71d9181a67b7542122c          mediterranean
1680    70821a40b06f8751781d5a895357da67                chinese
2319    f806c5d2707545d718717be03e69a8d4                   fast
3000    d0cbf1a1aa1726784df15a81ead214f7                italian
3004    d0cbf1a1aa1726784df15a81ead214f7                  greek
...                                  ...                    ...
195905  84d9ee44e457ddef7f2c4f25dc8fa865               sandwich
196141  84d9ee44e457ddef7f2c4f25dc8fa865               sandwich
196453  0d1a9651497a38d8b1c3871c84528bd4                  other
196469  0d1a9651497a38d8b1c3871c84528bd4                  other
196506  0d1a9651497a38d8b1c3871c84528bd4                  other

        order_protocol  total_items  subtotal  num_distinct_items  \
170                3.0            4      2945                   3
1680               1.0            2      1545                   2
2319               4.0            6      1555                   6
3000               5.0            4      4396                   4
3004               5.0            3      4396                   3
...                ...          ...       ...                 ...
195905             3.0            3      3219                   3
196141             5.0            2      2290                   2
196453             3.0            2      1549                   2
196469             3.0            3      1947                   3
196506             3.0            7      3193                   6

        min_item_price  max_item_price  total_onshift_partners  \
170                425            1245                   129.0
1680               695             825                   147.0
2319                 0             600                   144.0
3000               299            1699                   132.0
3004              1099            1499                   129.0
...                ...             ...                     ...
195905             999            1221                   127.0
196141             999             999                   130.0
196453             305            1042                   130.0
196469             349             899                   146.0
196506             149             899                   138.0
```

```
        total_busy_partners  total_outstanding_orders
170                   128.0                     261.0
1680                  136.0                     272.0
2319                  138.0                     276.0
3000                  129.0                     256.0
3004                  128.0                     261.0
...                     ...                       ...
195905                104.0                     251.0
196141                134.0                     253.0
196453                141.0                     260.0
196469                139.0                     272.0
196506                136.0                     253.0

[371 rows x 14 columns]
```

The three operational columns (onshift, busy, outstanding) also contain outliers.

Many rows with null or zero values occur together and show inconsistent behavior.

Data Cleaning

This stage involves detecting and addressing issues like missing values, duplicates, incorrect formatting, or irrelevant data, ensuring the data is accurate, complete, and consistent for analysis or decision-making.

Creating duplicate dataset, as it act as a backup and allows us for experimentation without altering the original data.

```
# creating porter dataframe
pdf = PorterData
```

Fix issues like nulls, wrong formats, and duplicated data before doing any analysis.

```
# dropping rows based on conditions

pdf.drop(pdf[pdf['total_items'] == 411].index, inplace=True) #
removing extreme outliers from total itmes column

pdf.drop(pdf[pdf['subtotal'] == 0].index, inplace=True) # removing
missing transaction data

pdf.drop(pdf[pdf['min_item_price'] < 0].index, inplace=True) #
removing negative values from column as price is not negative

pdf.drop(pdf[pdf['min_item_price'] > 10000].index, inplace=True) #
removing outliers from min item price column

pdf.drop(pdf[pdf['max_item_price'] == 0].index, inplace=True) #
removing missing transaction data
```

```
pdf.drop(pdf[pdf['total_onshift_partners'] < 0].index, inplace=True) #
removing negative values from column

pdf.drop(pdf[pdf['total_busy_partners'] < 0].index, inplace=True) #
removing negative values from column

pdf.drop(pdf[pdf['total_outstanding_orders'] < 0].index, inplace=True)
# removing negative values from column

pdf.dropna(subset=['actual_delivery_time'], inplace=True) # Guessing
they are canceled orders.
```

Replace nulls in categorical columns (like store type or order protocol) with a generic value like "unknown"

```
# replace all nulls with, 7/unknown category for market_id
pdf['market_id'].fillna(7, inplace=True)

# replace all nulls with 8/unknown category for order_protocol
pdf['order_protocol'].fillna(8, inplace=True)

# fill missing 'store_primary_category' with 'unknown'
pdf['store_primary_category'].fillna('unknown', inplace=True)
```

Convert the created_at column to datetime format so that time-based analysis can be done easily.

```
# convert to datetime
pdf['created_at'] = pd.to_datetime(pdf['created_at'])
pdf['actual_delivery_time'] =
pd.to_datetime(pdf['actual_delivery_time'])
```

Creating features columns like date, hour, minute, dayname, etc, using created_at column

delivery_duration_min: Calculate time taken for delivery in minutes.

```
# create a new columns for time features
pdf['date'] = pdf['created_at'].dt.date
pdf['hour'] = pdf['created_at'].dt.hour
pdf['minute'] = pdf['created_at'].dt.minute
pdf['dayname'] = pdf['created_at'].dt.day_name() # 0=Monday, 6=Sunday

# calculate delivery duration in minutes
pdf['delivery_duration_min'] = (pdf['actual_delivery_time'] -
pdf['created_at']).dt.total_seconds() / 60
```

available_partners: Calculate using onshift - busy.

partner_load: Orders divided by available partners, helps understand workload.

```
# create available partners = onshift partner - busy partners
pdf['available_partners'] = pdf['total_onshift_partners'] -
pdf['total_busy_partners']
pdf.loc[pdf['available_partners'] < 0, 'available_partners'] = np.nan
# replace negatives with NaN

# create partner load = orders / available partners
pdf['partner_load'] = pdf['total_items'] /
pdf['available_partners'].replace(0, np.nan)  # avoid divide by zero
```

Columns like total_onshift_partners, total_busy_partners, and total_outstanding_orders have about 8% missing data.

Deleting these rows would cause data loss, so we choose to fill them using imputation.

Since the data is not normally distributed, and has outliers, median is used to fill missing values for partner-related columns.

```
print(f"Onshift_Mean: {pdf['total_onshift_partners'].mean()}")
print(f"Onshift_Median: {pdf['total_onshift_partners'].median()}")

print(f"Busy_Mean: {pdf['total_busy_partners'].mean()}")
print(f"Busy_Median: {pdf['total_busy_partners'].median()}")

print(f"Outstanding_Mean: {pdf['total_outstanding_orders'].mean()}")
print(f"Outstanding_Median:
{pdf['total_outstanding_orders'].median()}")
```

```
Onshift_Mean: 44.82775910488026
Onshift_Median: 37.0
Busy_Mean: 41.7578223470358
Busy_Median: 34.0
Outstanding_Mean: 58.07873206111934
Outstanding_Median: 41.0
```

```
# replacing null values with total_onshift_parteners.median value
pdf['total_onshift_partners'].fillna(pdf['total_onshift_partners'].med
ian(), inplace=True)

# replacing null values with total_busy_parteners.median value
pdf['total_busy_partners'].fillna(pdf['total_busy_partners'].median(),
inplace=True)

# replacing null values with total_outstanding_orders.median value
pdf['total_outstanding_orders'].fillna(pdf['total_outstanding_orders']
.median(), inplace=True)

# Converting columns datatype
pdf['date'] = pd.to_datetime(pdf['date'])
pdf['market_id'] = pdf['market_id'].astype('object')
pdf['order_protocol'] = pdf['order_protocol'].astype('object')
```

```
# checking information about the data
pdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 197139 entries, 0 to 197427
Data columns (total 21 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 197139 non-null  object
 1   created_at                197139 non-null  datetime64[ns]
 2   actual_delivery_time      197139 non-null  datetime64[ns]
 3   store_id                  197139 non-null  object
 4   store_primary_category    197139 non-null  object
 5   order_protocol            197139 non-null  object
 6   total_items               197139 non-null  int64
 7   subtotal                  197139 non-null  int64
 8   num_distinct_items        197139 non-null  int64
 9   min_item_price            197139 non-null  int64
 10  max_item_price            197139 non-null  int64
 11  total_onshift_partners    197139 non-null  float64
 12  total_busy_partners       197139 non-null  float64
 13  total_outstanding_orders  197139 non-null  float64
 14  date                      197139 non-null  datetime64[ns]
 15  hour                      197139 non-null  int64
 16  minute                    197139 non-null  int64
 17  dayname                   197139 non-null  object
 18  delivery_duration_min     197139 non-null  float64
 19  available_partners        140578 non-null  float64
 20  partner_load              104717 non-null  float64
dtypes: datetime64[ns](3), float64(6), int64(7), object(5)
memory usage: 33.1+ MB
```

```
# Check null percentage
pdf.isnull().mean() * 100
```

```
market_id                   0.000000
created_at                  0.000000
actual_delivery_time        0.000000
store_id                    0.000000
store_primary_category      0.000000
order_protocol              0.000000
total_items                 0.000000
subtotal                    0.000000
num_distinct_items          0.000000
min_item_price              0.000000
max_item_price              0.000000
total_onshift_partners      0.000000
total_busy_partners         0.000000
total_outstanding_orders    0.000000
date                        0.000000
```

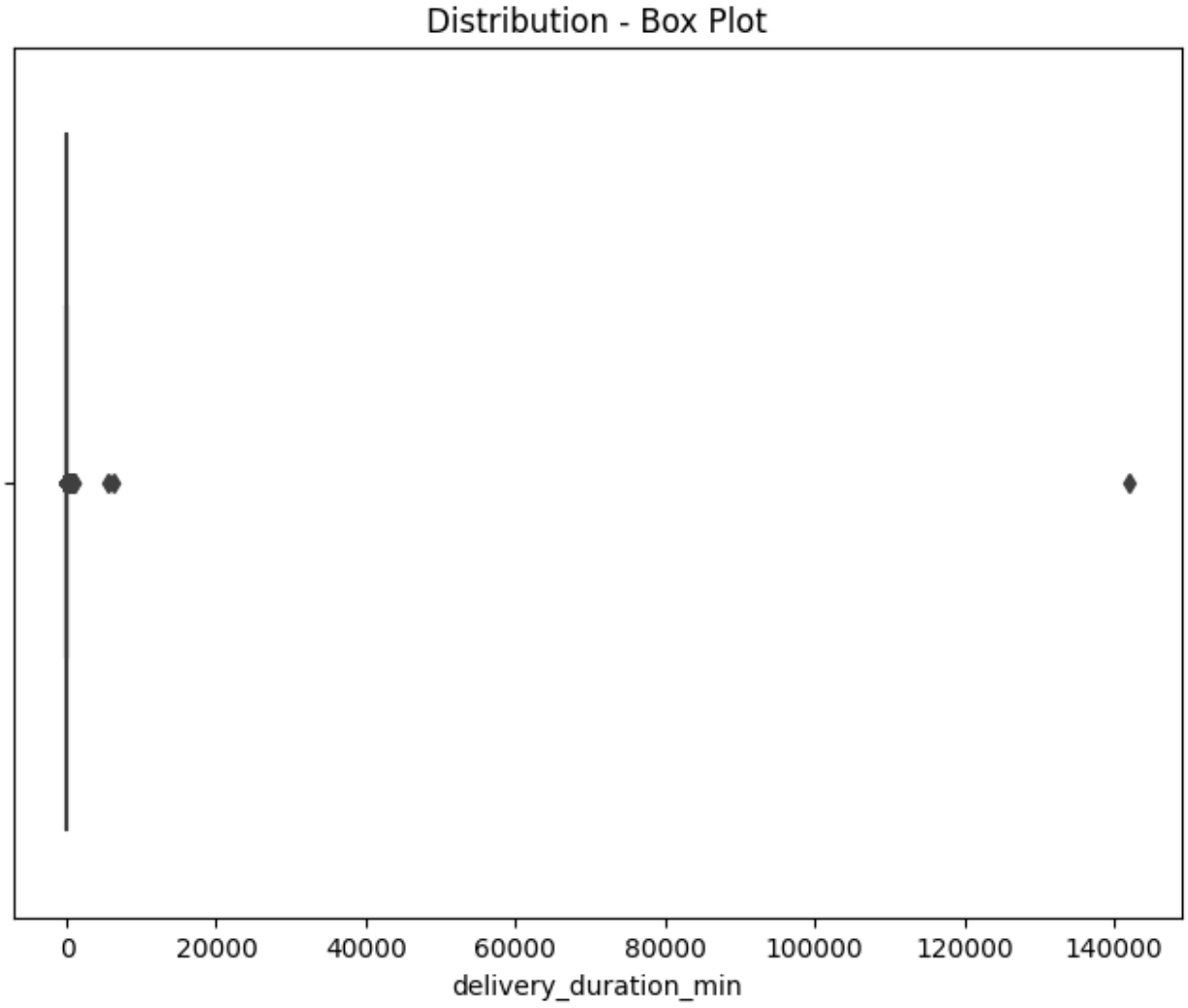
```
hour                          0.000000
minute                        0.000000
dayname                       0.000000
delivery_duration_min         0.000000
available_partners           28.690924
partner_load                 46.881642
dtype: float64
```

```
# Checking descriptive statistics
pdf.describe()
```

```
         total_items       subtotal  num_distinct_items
min_item_price  \
count  197139.000000  197139.000000       197139.000000
197139.000000
mean        3.193889    2684.840082            2.670887
686.141088
std         2.501450    1822.061562            1.630295
521.023015
min         1.000000      95.000000            1.000000
0.000000
25%         2.000000    1401.000000            1.000000
299.000000
50%         3.000000    2200.000000            2.000000
595.000000
75%         4.000000    3396.000000            3.000000
949.000000
max        84.000000   27100.000000           20.000000
8999.000000

       max_item_price  total_onshift_partners  total_busy_partners  \
count   197139.000000           197139.000000        197139.000000
mean      1159.587915               44.182643            41.118470
std        557.535031               33.141684            30.863559
min         60.000000                0.000000             0.000000
25%        800.000000               19.000000            17.000000
50%       1095.000000               37.000000            34.000000
75%       1395.000000               62.000000            59.000000
max       8999.000000              171.000000           154.000000

       total_outstanding_orders           hour         minute  \
count             197139.000000  197139.000000  197139.000000
mean                  56.671207       8.465950      29.559676
std                   50.664727       8.658617      17.277731
min                    0.000000       0.000000       0.000000
25%                   19.000000       2.000000      15.000000
50%                   41.000000       3.000000      29.000000
75%                   80.000000      19.000000      45.000000
max                  285.000000      23.000000      59.000000
```

```
       delivery_duration_min  available_partners   partner_load
count          197139.000000       140578.000000  104717.000000
mean               48.470635            6.237555       1.115262
std               320.721991            9.427962       1.608477
min                 1.683333            0.000000       0.012500
25%                35.066667            0.000000       0.222222
50%                44.333333            3.000000       0.500000
75%                56.350000            8.000000       1.333333
max            141947.650000           86.000000      48.000000
```

```
# boxplot for delivery duration column
plt.figure(figsize=(8, 6))
sns.boxplot(x=pdf['delivery_duration_min'])
plt.title('Distribution - Box Plot')
plt.show()
```



Column like available_partners and partner_load contain null values, as difference between onshit and busy partners, in some of rows is negative, and replacing this negative to null values, as the result linked to partner_load column.

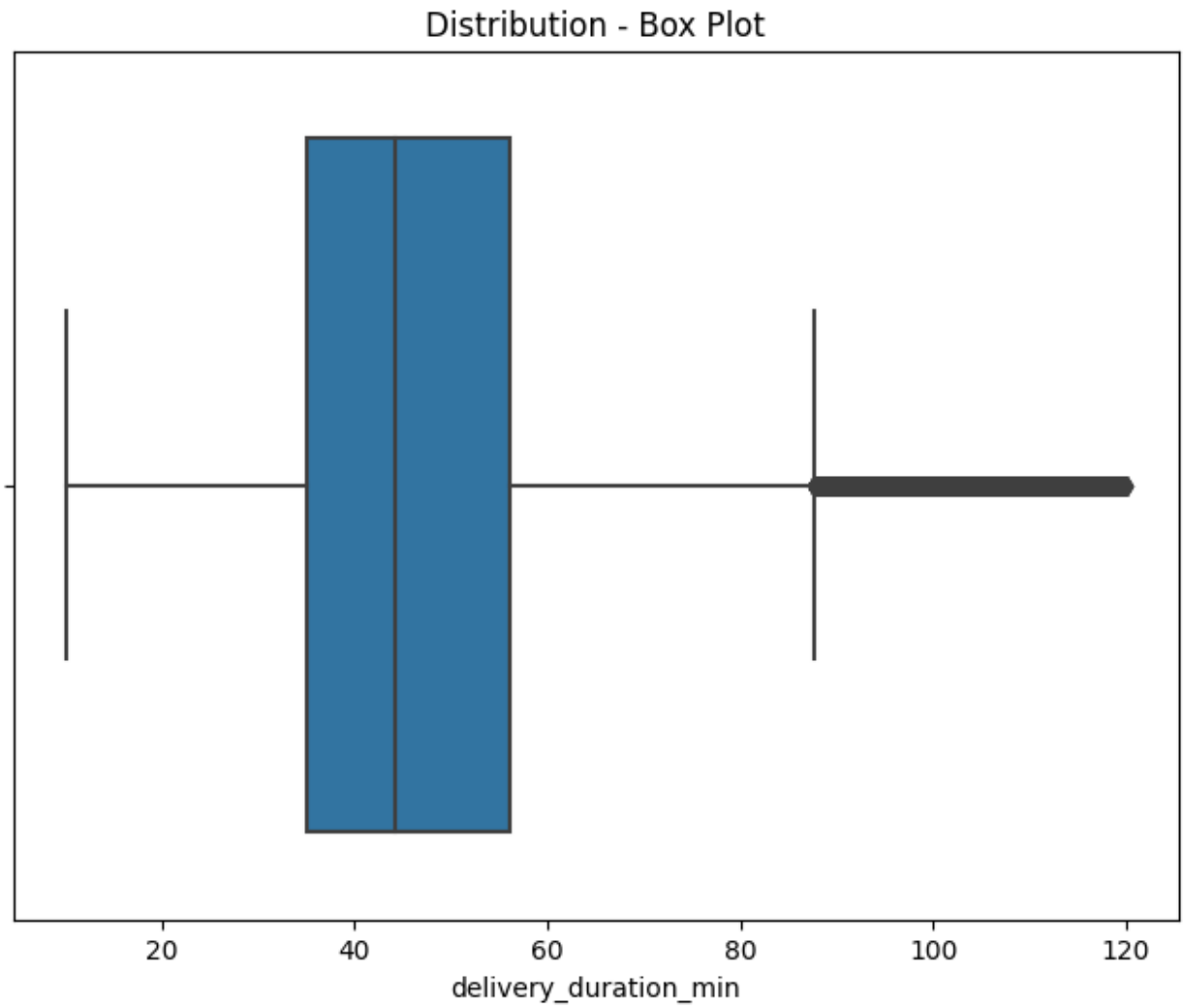
Delivery Duration contain unrealistic time i.e. outliers, so limiting the rows between 10 minutes and 2 hours

```
# outliers from delivery duration column
((pdf['delivery_duration_min'] < 10) | (pdf['delivery_duration_min'] >
120)).value_counts()
```

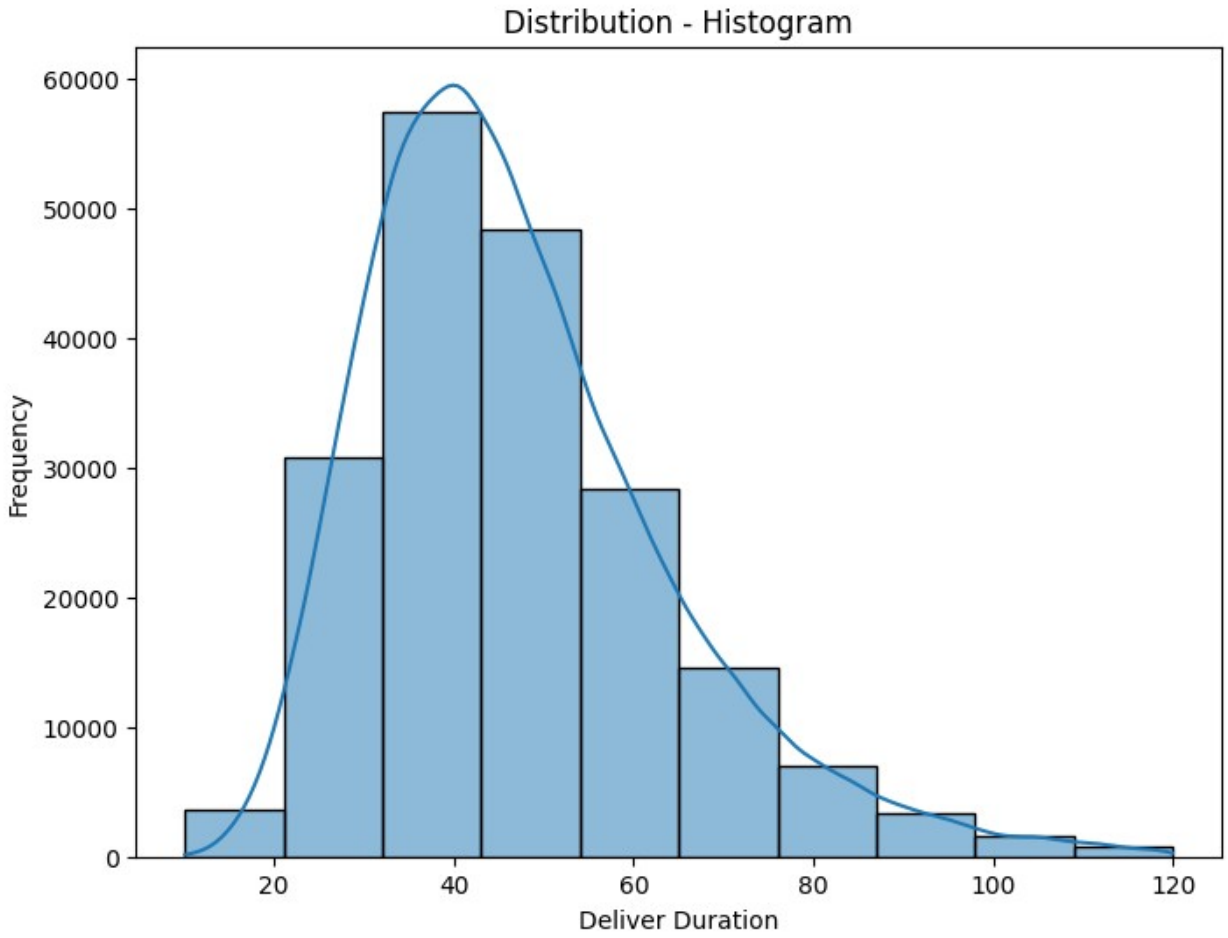
```
False    196022
True       1117
Name: delivery_duration_min, dtype: int64
```

Only keep records between 10 minutes and 2 hours to remove unrealistic values.

```
# Filter unrealistic delivery times (e.g., <10 mins or >120 mins)
pdf = pdf[(pdf['delivery_duration_min'] >= 10) &
(pdf['delivery_duration_min'] <= 120)]

# boxplot for delivery duration column
plt.figure(figsize=(8, 6))
sns.boxplot(x=pdf['delivery_duration_min'])
plt.title('Distribution - Box Plot')
plt.show()
```




```
# histogram for delivery duration column
plt.figure(figsize=(8, 6))
sns.histplot(pdf['delivery_duration_min'], kde=True, bins=10)
plt.title('Distribution - Histogram')
plt.xlabel('Deliver Duration')
plt.ylabel('Frequency')
plt.show()
```




```
pdf.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 196022 entries, 0 to 197427
Data columns (total 21 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   market_id                196022 non-null  object
 1   created_at               196022 non-null  datetime64[ns]
 2   actual_delivery_time     196022 non-null  datetime64[ns]
 3   store_id                 196022 non-null  object
 4   store_primary_category   196022 non-null  object
 5   order_protocol           196022 non-null  object
 6   total_items              196022 non-null  int64
 7   subtotal                 196022 non-null  int64
 8   num_distinct_items       196022 non-null  int64
 9   min_item_price           196022 non-null  int64
 10  max_item_price           196022 non-null  int64
 11  total_onshift_partners   196022 non-null  float64
 12  total_busy_partners      196022 non-null  float64
 13  total_outstanding_orders 196022 non-null  float64
```

```
 14  date                        196022 non-null  datetime64[ns]
 15  hour                        196022 non-null  int64
 16  minute                      196022 non-null  int64
 17  dayname                     196022 non-null  object
 18  delivery_duration_min       196022 non-null  float64
 19  available_partners          139870 non-null  float64
 20  partner_load                104345 non-null  float64
dtypes: datetime64[ns](3), float64(6), int64(7), object(5)
memory usage: 32.9+ MB

pdf.describe()

         total_items       subtotal  num_distinct_items
min_item_price  \
count  196022.000000  196022.000000       196022.000000
196022.000000
mean        3.191392    2682.255946            2.669058
686.126685
std         2.498909    1818.862710            1.627928
520.574802
min         1.000000      95.000000            1.000000
0.000000
25%         2.000000    1400.000000            1.000000
299.000000
50%         3.000000    2200.000000            2.000000
595.000000
75%         4.000000    3395.000000            3.000000
949.000000
max        84.000000   26800.000000           20.000000
8999.000000

       max_item_price  total_onshift_partners  total_busy_partners  \
count   196022.000000           196022.000000        196022.000000
mean      1159.187265               44.228796            41.156146
std        556.976319               33.144110            30.866443
min         60.000000                0.000000             0.000000
25%        800.000000               19.000000            17.000000
50%       1095.000000               37.000000            34.000000
75%       1395.000000               62.000000            60.000000
max       8999.000000              171.000000           154.000000

       total_outstanding_orders           hour         minute  \
count             196022.000000  196022.000000  196022.000000
mean                  56.671312       8.475727      29.558453
std                   50.639770       8.661893      17.278957
min                    0.000000       0.000000       0.000000
25%                   19.000000       2.000000      15.000000
50%                   41.000000       3.000000      29.000000
75%                   80.000000      19.000000      45.000000
max                  285.000000      23.000000      59.000000
```

```
       delivery_duration_min  available_partners   partner_load
count          196022.000000       139870.000000  104345.000000
mean               47.100819            6.248552       1.113847
std                16.926042            9.433390       1.606488
min                10.116667            0.000000       0.012500
25%                35.016667            0.000000       0.222222
50%                44.216667            3.000000       0.500000
75%                56.083333            8.000000       1.333333
max               119.933333           86.000000      48.000000
```

Summarizing Data

```
# earliest and latest timestamps recorded
start_datetime = pdf['created_at'].min()
end_datetime = pdf['created_at'].max()

print(f"Start Date & Time: {start_datetime}")
print(f"End Date & Time: {end_datetime}")
```

```
Start Date & Time: 2015-01-21 15:22:03
End Date & Time: 2015-02-18 06:00:44
```

```
# total orders
total_orders = pdf.shape[0]
print("Total Orders:", total_orders)
```

```
Total Orders: 196022
```

```
# total distinct values for market_id, store_id,
store_primary_category, order_protocol
distinct_counts = {
    'market_id': pdf['market_id'].nunique(),
    'store_id': pdf['store_id'].nunique(),
    'store_primary_category': pdf['store_primary_category'].nunique(),
    'order_protocol': pdf['order_protocol'].nunique()
}

print("Distinct Value Counts:")
for key, value in distinct_counts.items():
    print(f"{key}: {value}")
```

```
Distinct Value Counts:
market_id: 7
store_id: 6732
store_primary_category: 75
order_protocol: 8
```

```
#  average number_of_distinct_items per order
avg_distinct_items = round(pdf['num_distinct_items'].mean())
print(f"Average Distinct Items per Order: {avg_distinct_items:.2f}")
```

```
Average Distinct Items per Order: 3.00
```

```
# average orders per week
orders_per_week = pdf['created_at'].dt.to_period('W').value_counts()
avg_orders_per_week = round(orders_per_week.mean())

# average orders per day
orders_per_day = pdf['created_at'].dt.to_period('D').value_counts()
avg_orders_per_day = round(orders_per_day.mean())

# average orders per hour
orders_per_hour = pdf['created_at'].dt.to_period('H').value_counts()
avg_orders_per_hour = round(orders_per_hour.mean())

# average orders per minute
orders_per_minute = pdf['created_at'].dt.to_period('T').value_counts()
avg_orders_per_minute = round(orders_per_minute.mean())

print(f"Average Orders per Week: {avg_orders_per_week:.2f}")
print(f"Average Orders per Day: {avg_orders_per_day:.2f}")
print(f"Average Orders per Hour: {avg_orders_per_hour:.2f}")
print(f"Average Orders per Minute: {avg_orders_per_minute:.4f}")
```

```
Average Orders per Week: 39204.00
Average Orders per Day: 6759.00
Average Orders per Hour: 413.00
Average Orders per Minute: 8.0000
```

```
# average value for minimum price, maximum price, and subtotal per
order
avg_min_price = pdf['min_item_price'].mean()
avg_max_price = pdf['max_item_price'].mean()
avg_subtotal = pdf['subtotal'].mean()

print(f"Average Minimum Price per Order: {avg_min_price:.2f}")
print(f"Average Maximum Price per Order: {avg_max_price:.2f}")
print(f"Average Subtotal Price per Order: {avg_subtotal:.2f}")
```

```
Average Minimum Price per Order: 686.13
Average Maximum Price per Order: 1159.19
Average Subtotal Price per Order: 2682.26
```

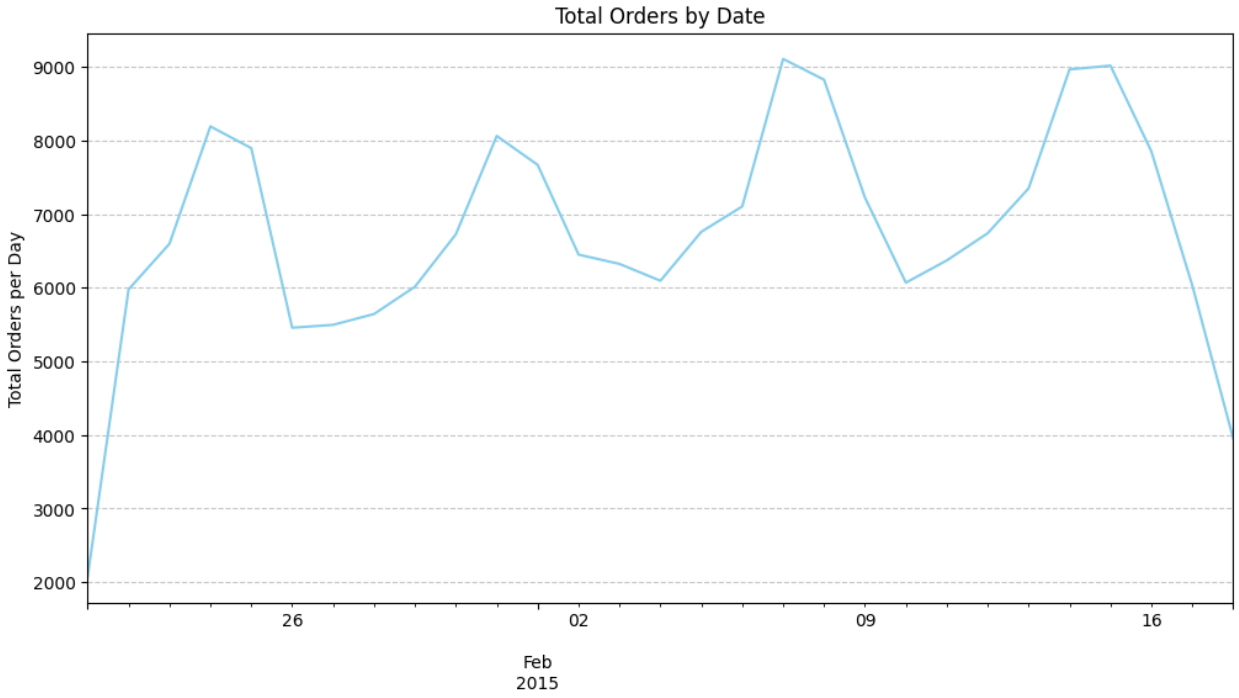
```
# average for onshift / busy partners and outstanding orders
avg_onshift = round(pdf['total_onshift_partners'].mean())
avg_busy = round(pdf['total_busy_partners'].mean())
avg_outstanding = round(pdf['total_outstanding_orders'].mean())

print(f"Average Onshift Partners per Order: {avg_onshift:.2f}")
print(f"Average Busy Partners per Order: {avg_busy:.2f}")
print(f"Average Outstanding Orders per Order: {avg_outstanding:.2f}")
```

```
Average Onshift Partners per Order: 44.00
Average Busy Partners per Order: 41.00
Average Outstanding Orders per Order: 57.00
```

Data Visualization

Total Orders by Date

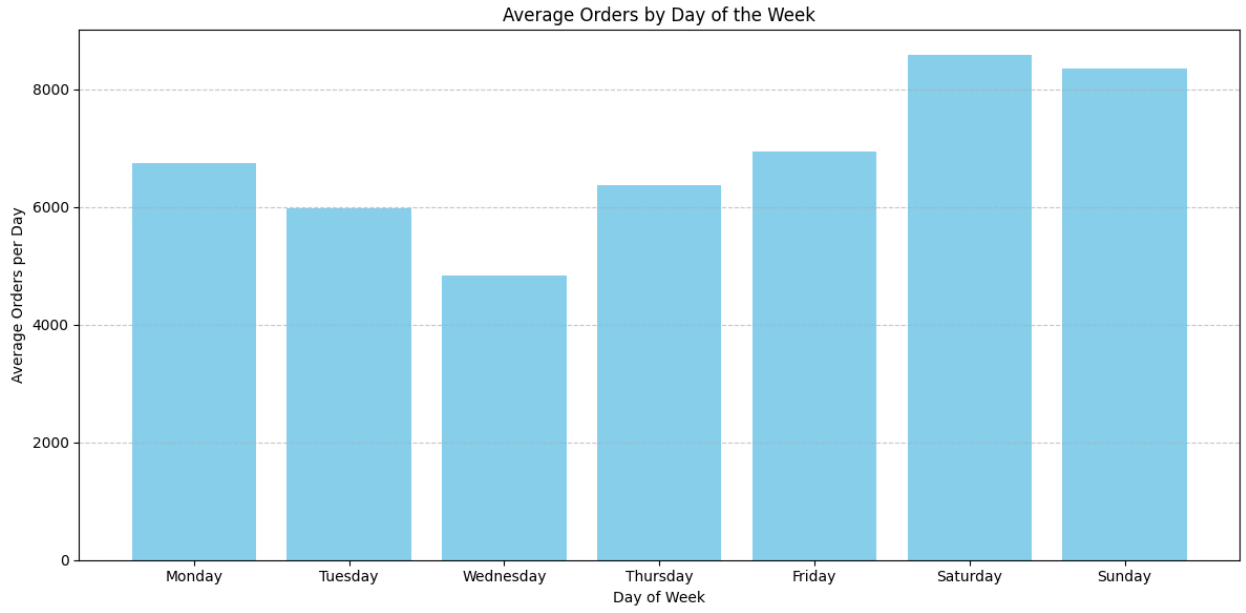
```
# total orders by date
orders_by_date = pdf.groupby('date').size()

orders_by_date.plot(kind='line', figsize=(12, 6), title='Total Orders
by Date', color='skyblue')
plt.xlabel(None)
plt.ylabel('Total Orders per Day')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- Daily orders increased steadily over time, showing rising demand for delivery services.
- Orders spike on weekends—especially on Feb 7, Feb 14 (Valentine's Day), and Feb 15, crossing 9,000+ orders.
- After Feb 15, a slight dip is seen—possibly due to reduced weekend activity or external factors.

Average Orders by Day of Week

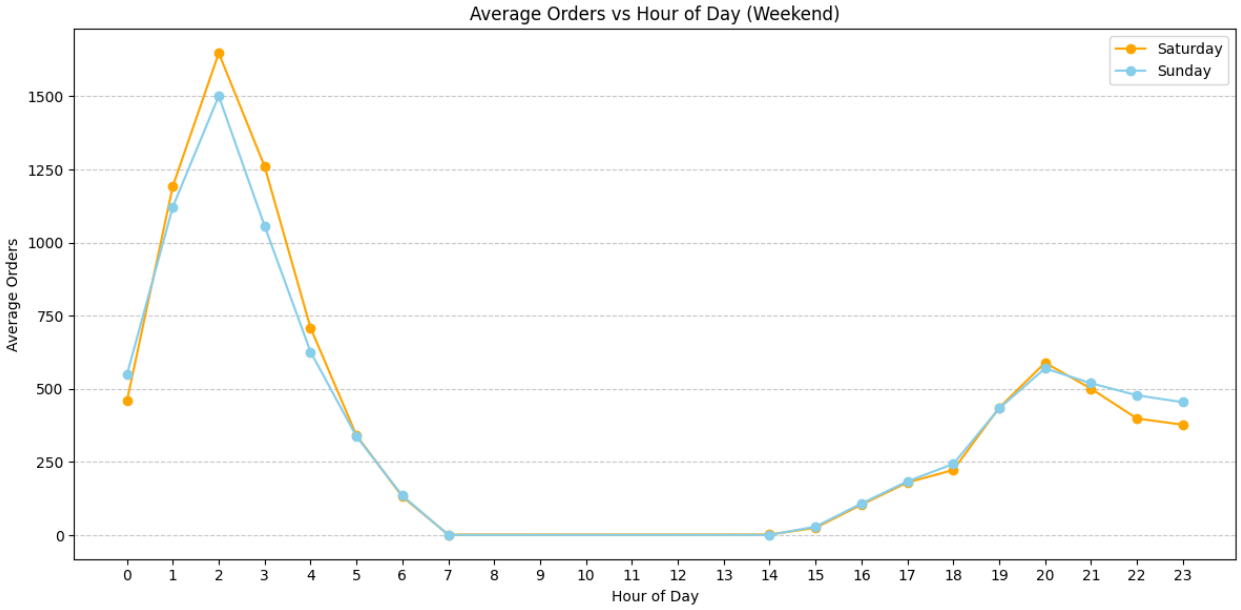
```
# average order by dayname
avg_orders_by_day = (
    pdf.groupby(['dayname', 'date'])
    .size()
    .groupby('dayname')
    .mean()
    .reindex(
        ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
    )
    .reset_index(name='avg_orders')
)


plt.figure(figsize=(12, 6))

plt.bar(avg_orders_by_day['dayname'], avg_orders_by_day['avg_orders'],
color='skyblue')
plt.title('Average Orders by Day of the Week')
plt.xlabel('Day of Week')
plt.ylabel('Average Orders per Day')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



- Saturday (8,580) and Sunday (8,351) see the highest demand, confirming weekends as peak ordering times.
- Friday (6,943) also shows a spike—likely due to weekend planning or celebrations.
- The lowest demand is on Wednesday (4,827).

Average Orders by Hour of Day for Saturday and Sunday

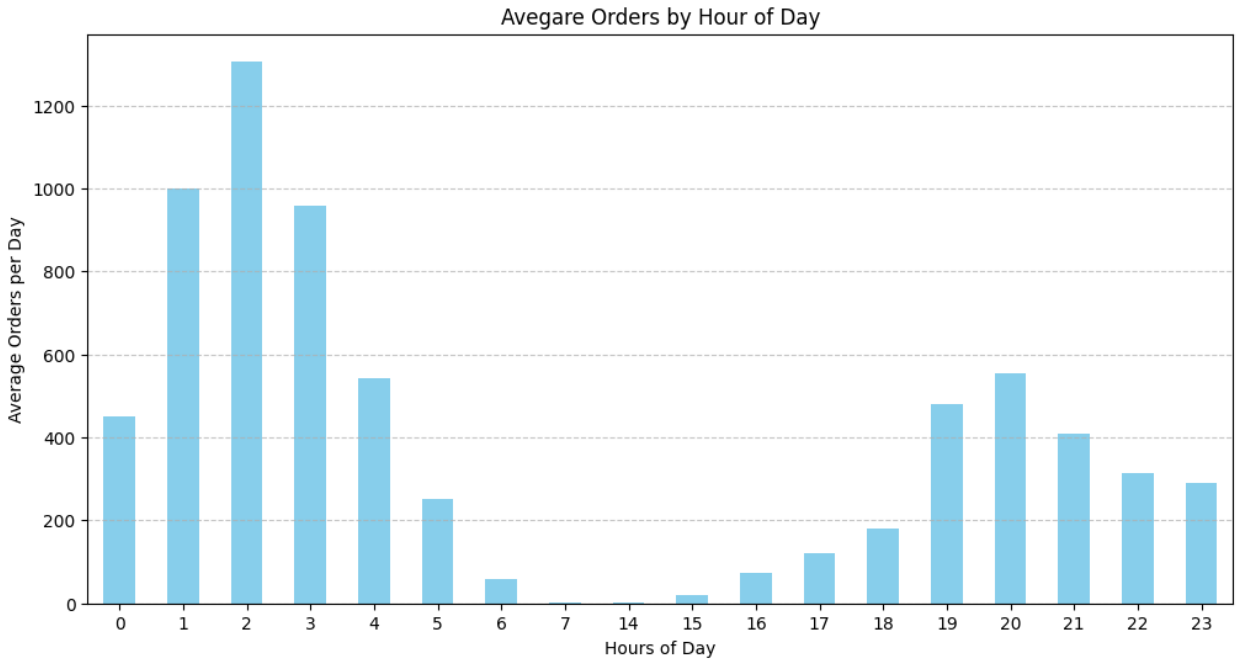
```
# average orders by hour (saturday & sunday)
saturday = pdf[pdf['dayname'] == 'Saturday']
sunday = pdf[pdf['dayname'] == 'Sunday']

saturday_avg = saturday.groupby(['date',
'hour']).size().groupby('hour').mean()
sunday_avg = sunday.groupby(['date',
'hour']).size().groupby('hour').mean()

plt.figure(figsize=(12, 6))

plt.plot(saturday_avg.index, saturday_avg.values, label='Saturday',
color='orange', marker='o')
plt.plot(sunday_avg.index, sunday_avg.values, label='Sunday',
color='skyblue', marker='o')

plt.title('Average Orders vs Hour of Day (Weekend)')
plt.xlabel('Hour of Day')
plt.ylabel('Average Orders')
plt.xticks(range(0, 24))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend()
plt.tight_layout()
plt.show()
```



- Peak hours for both days fall between 1 AM – 3 AM, again reinforcing the late-night hunger trend.
- A minor evening spike from 5 PM–9 PM, especially on Sunday evenings, shows last-minute weekend treats.

Average Orders by Hour of Day

```
# average orders by hour of day
orders_by_date_hour = pdf.groupby(['date',
'hour']).size().reset_index(name='orders')
avg_orders_by_hour = orders_by_date_hour.groupby('hour')
['orders'].mean()

avg_orders_by_hour.plot(kind='bar', figsize=(12,6), title='Avegare
Orders by Hour of Day', color='skyblue')
plt.xlabel('Hours of Day')
plt.ylabel('Average Orders per Day')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.show()
```

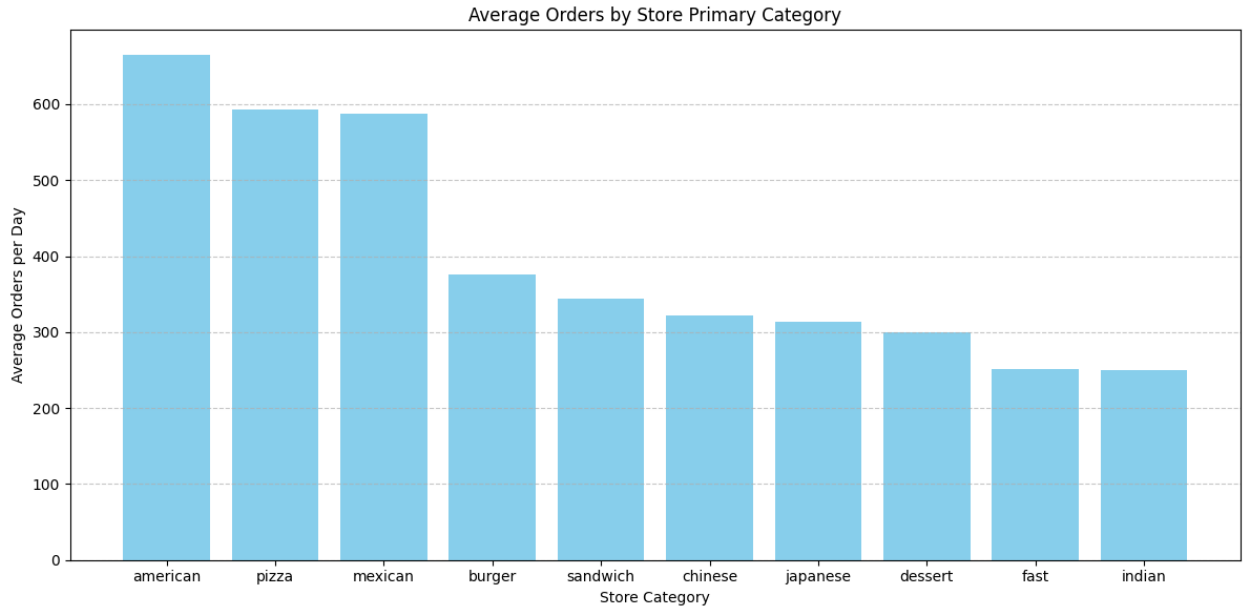


- Highest demand is during early hours (1 AM–3 AM) — peaking around 2 AM (1,306).
- Possibly due to late-night cravings or post-party orders.
- Another small pickup starts around 5 PM–8 PM, suggesting early dinner or evening snacks.
- Very few orders from 6 AM to 2 PM, with a dip to almost zero between 7 AM–2 PM.

Average Orders by Store Primary Category

```
# top 10 store category by average orders
daily_orders_by_category = pdf.groupby(['store_primary_category',
'date']).size().reset_index(name='orders')
avg_orders_by_category =
daily_orders_by_category.groupby('store_primary_category')
['orders'].mean().reset_index()
```

```
avg_orders_by_category =
avg_orders_by_category.sort_values(by='orders',
ascending=False).head(10)


plt.figure(figsize=(12, 6))

plt.bar(avg_orders_by_category['store_primary_category'],
avg_orders_by_category['orders'], color='skyblue')
plt.title('Average Orders by Store Primary Category')
plt.xlabel("Store Category")
plt.ylabel("Average Orders per Day")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```

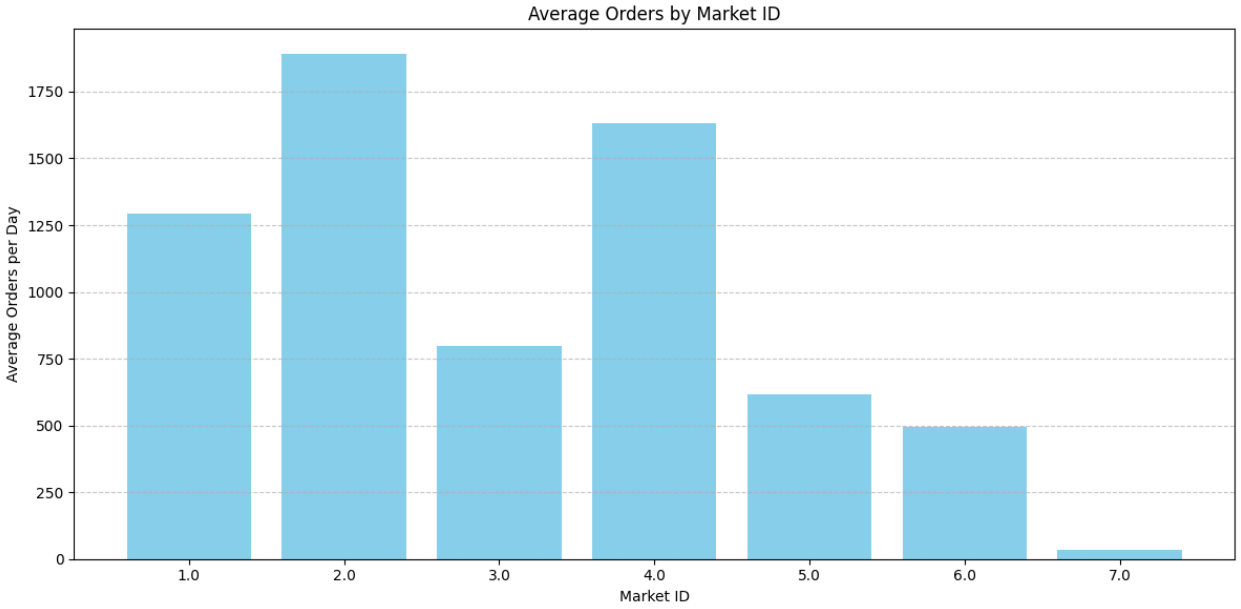


- American cuisine leads the way with the highest average number of orders (664), suggesting a strong customer preference for this food category.
- Pizza and Mexican foods follow closely, with 592 and 586 average orders respectively, indicating high demand for comfort and fast foods.
- Burgers, sandwiches, and Chinese dishes round out the mid-tier, showing consistent popularity across different customer segments.
- Indian food is in the 10th position, with a relatively lower average order volume (250), suggesting potential growth opportunity in this category depending on location.

Average Orders by Market Id

```
# average order by market id
avg_orders_by_market = (
```

```
    pdf.groupby(['market_id', 'date'])
    .size()
    .groupby('market_id')
    .mean()
    .reset_index(name='avg_orders')
)

plt.figure(figsize=(12, 6))

plt.bar(avg_orders_by_market['market_id'].astype(str),
avg_orders_by_market['avg_orders'], color='skyblue')
plt.title('Average Orders by Market ID')
plt.xlabel('Market ID')
plt.ylabel('Average Orders per Day')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

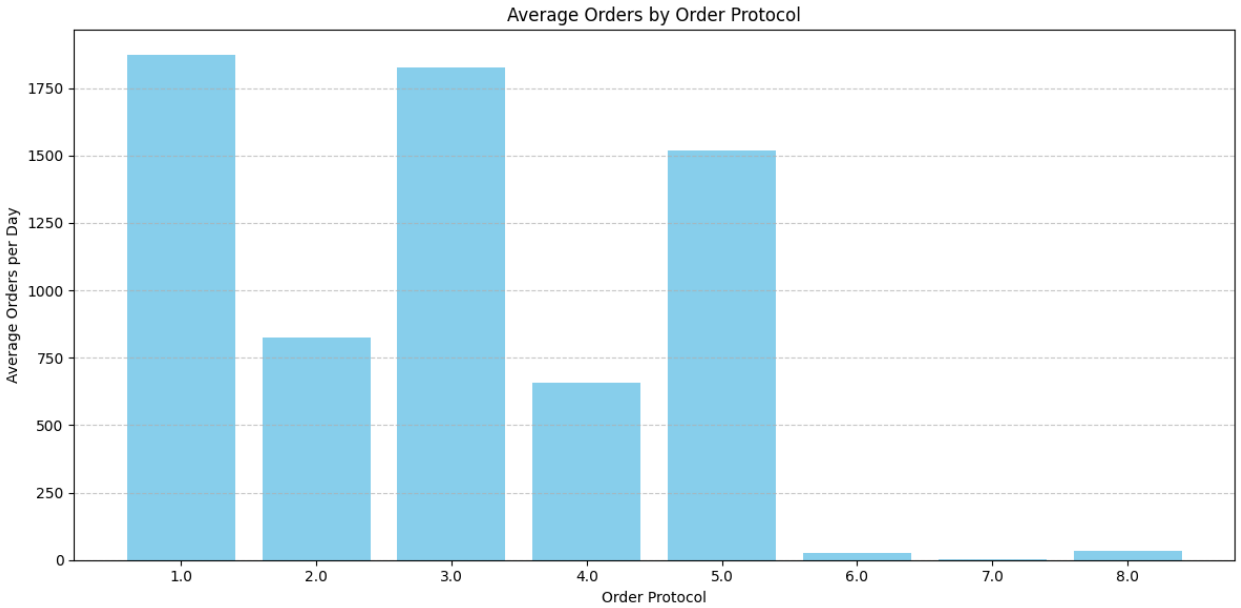


- Market 2 (1,890) and Market 4 (1,632) are the busiest.
- Market 6,comparativly, has low demand (495) — indicating a new, niche, or underserved location.

Average Orders by Order Protocol

```
# average orders by order protocol
avg_orders_by_protocol = (
    pdf.groupby(['order_protocol', 'date'])
    .size()
    .groupby('order_protocol')
```

```
    .mean()
    .reset_index(name='avg_orders')
)

plt.figure(figsize=(12, 6))

plt.bar(avg_orders_by_protocol['order_protocol'].astype(str),
avg_orders_by_protocol['avg_orders'], color='skyblue')
plt.title('Average Orders by Order Protocol')
plt.xlabel('Order Protocol')
plt.ylabel('Average Orders per Day')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

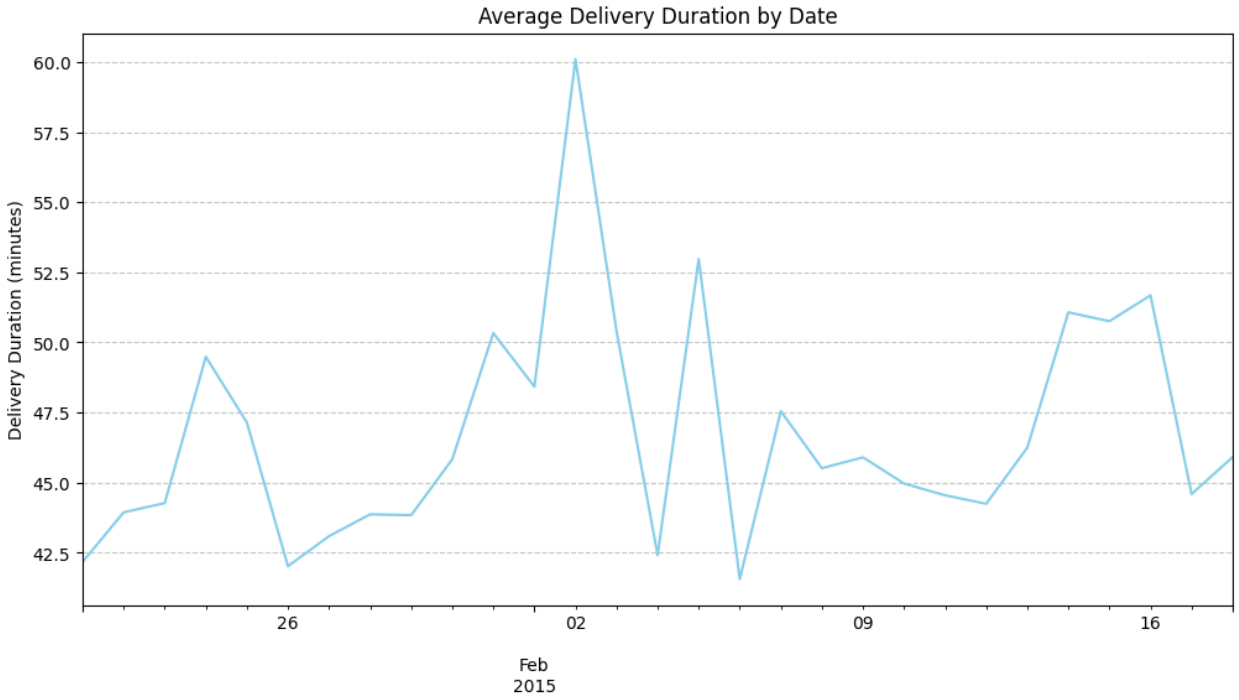


- Protocols 1.0 (1,872) and 3.0 (1,824) are most used — likely mobile app and web orders.
- Lower protocols like 6 and 7 show minimal use — could be legacy systems or internal channels.

Average Delivery Duration by Date

```
# average delivery duraiton by date
avg_duration_by_date = pdf.groupby('date')
['delivery_duration_min'].mean()

avg_duration_by_date.plot(kind='line', figsize=(12, 6), title='Average
Delivery Duration by Date', color='skyblue')
plt.xlabel(None)
plt.ylabel('Delivery Duration (minutes)')
```

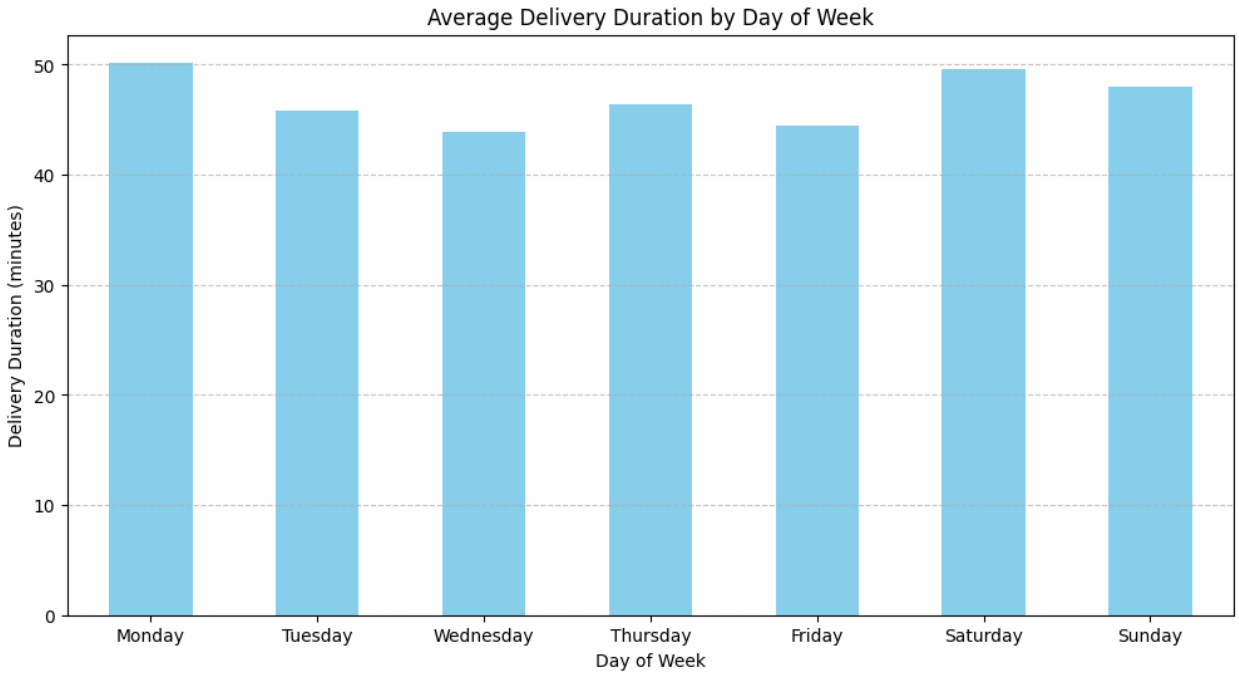
```
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- There's a noticeable spike in delivery duration around Feb 2, 2015 (60 mins).
- Weekends like Jan 31 and Feb 14-16 also show higher delivery times, possibly due to higher demand..

Average Delivery Duration by Day of Week

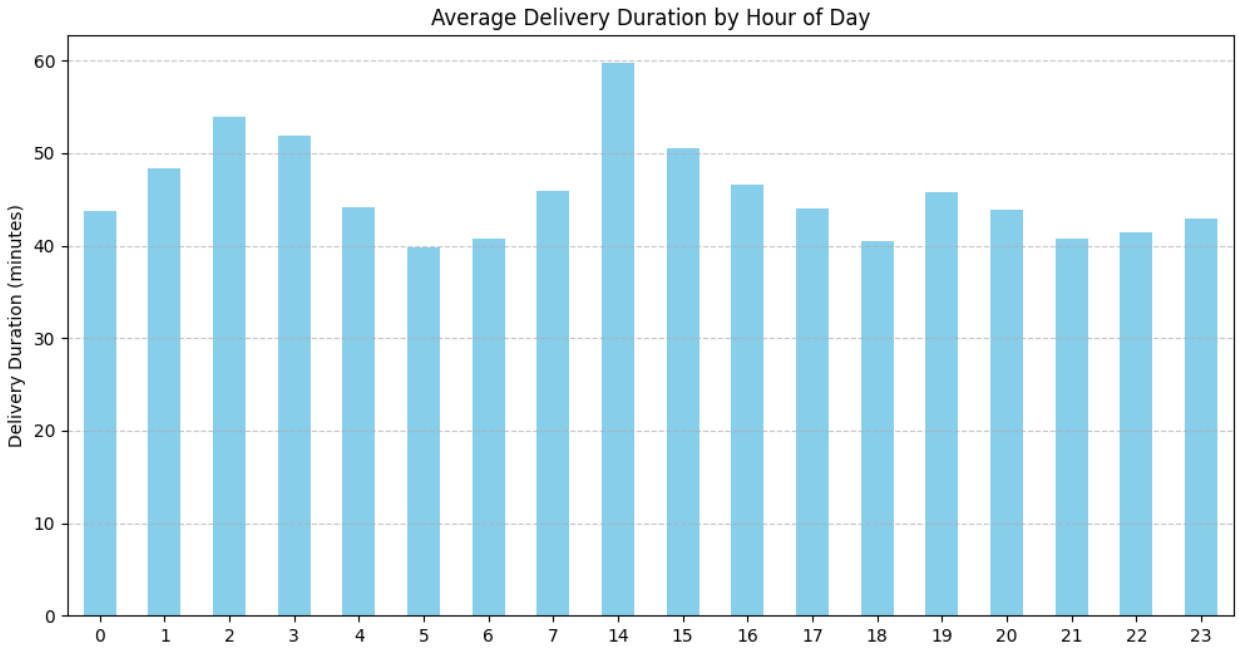
```
# average duration by dayname
avg_duration_by_day = (
    pdf.groupby('dayname')['delivery_duration_min']
    .mean()
    .reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
    )
)

avg_duration_by_day.plot(kind='bar', figsize=(12, 6), title='Average
Delivery Duration by Day of Week', color='skyblue')
plt.xlabel('Day of Week')
plt.ylabel('Delivery Duration (minutes)')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- Monday (50.2 mins) and Saturday (49.6 mins) show the highest average durations.
- Wednesday (43.9 mins) is the most efficient day on average.
- Consider allocating more resources on Mondays and Saturdays.

Average Delivery Duration by Hour of Day

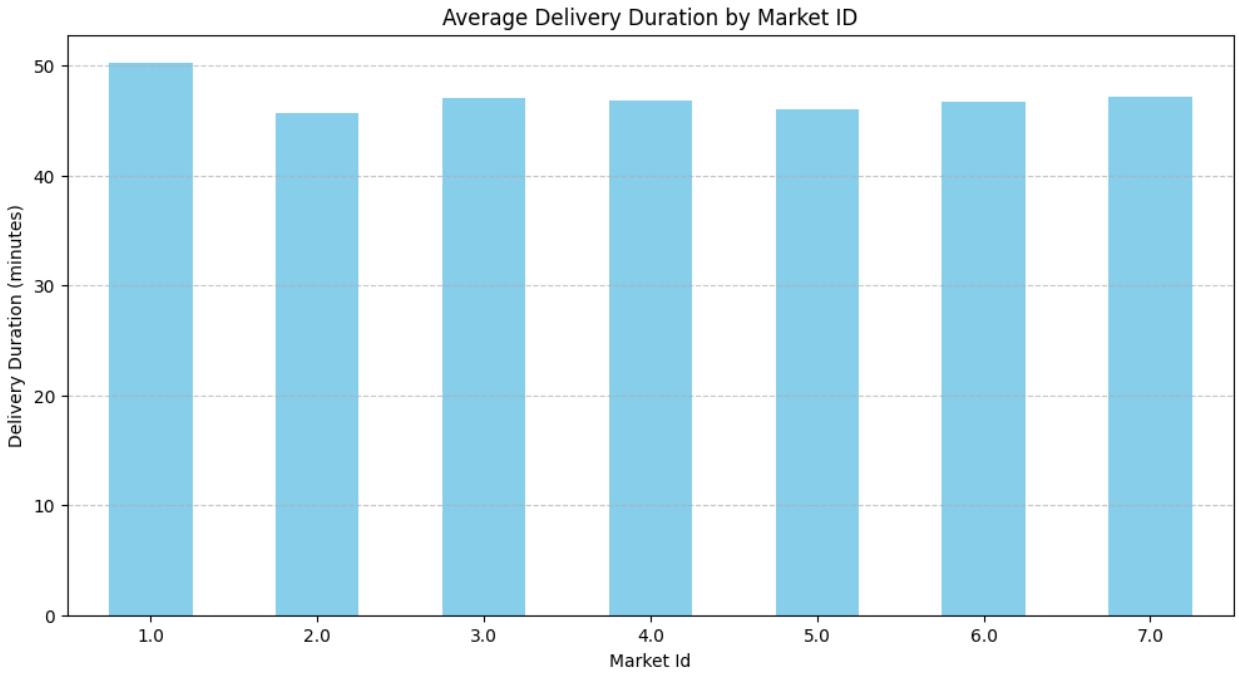
```
# average duration by hour
avg_duration_by_hour = (
    pdf.groupby('hour')['delivery_duration_min']
       .mean()
)

avg_duration_by_hour.plot(kind='bar', figsize=(12,6), title='Average
Delivery Duration by Hour of Day', color='skyblue')
plt.xlabel(None)
plt.ylabel('Delivery Duration (minutes)')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- 2 AM (54 mins) and 3 AM (52 mins) have the longest delivery durations.
- 5 AM – 6 AM (40 mins) and 9 PM – 11 PM (40-43 mins) are the most efficient periods.
- Resource allocation or order demand may vary significantly by hour—optimize late-night staffing.

Average Delivery Duration by Market Id

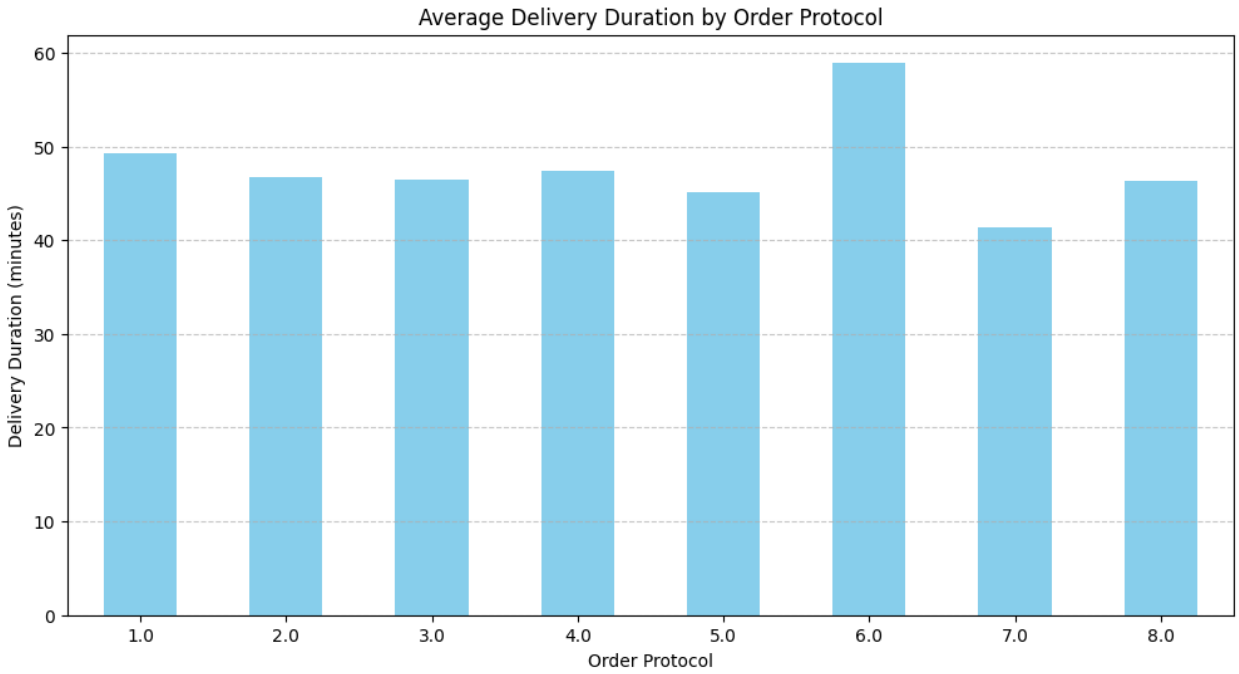
```
# average duration by market
avg_duration_by_market = pdf.groupby('market_id')
['delivery_duration_min'].mean()

avg_duration_by_market.plot(kind='bar', figsize=(12,6), title='Average
Delivery Duration by Market ID', color='skyblue')
plt.xlabel('Market Id')
plt.ylabel('Delivery Duration (minutes)')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- Market 1 has the longest average time (50.3 mins).
- Market 2 performs the best (45.7 mins).
- Investigate local market issues or traffic conditions in Market 1.

Average Delivery Duration by Order Protocol

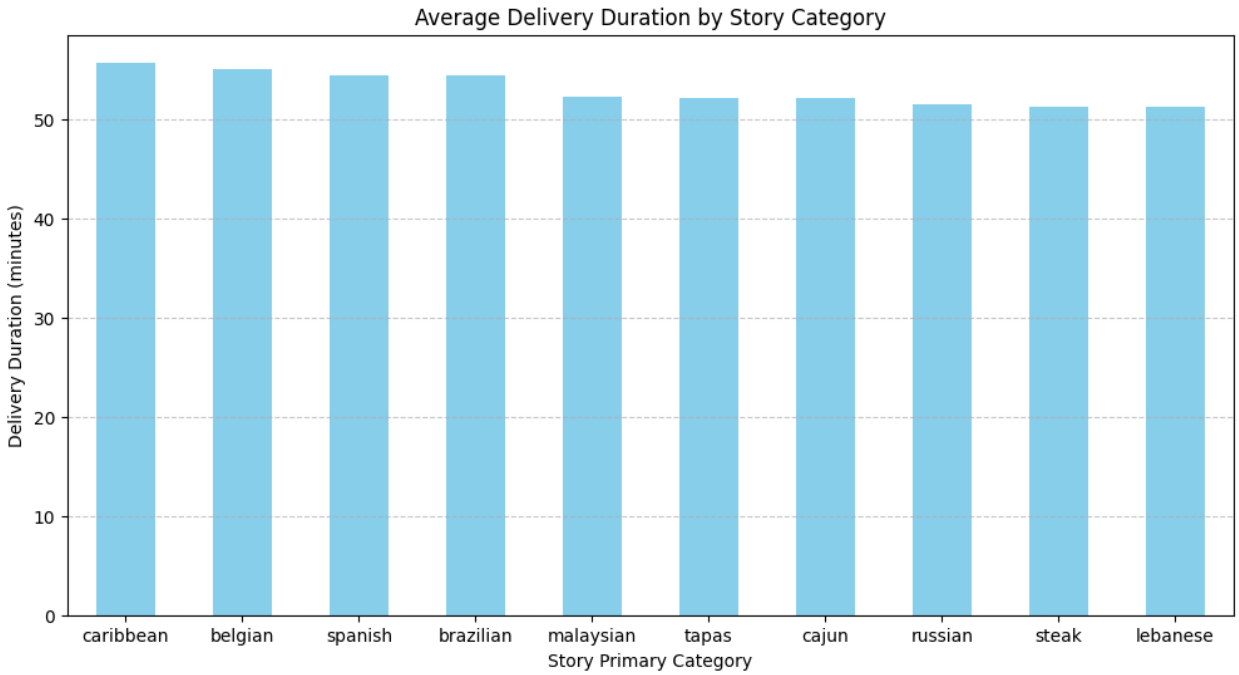
```
# average duration by protocol
avg_duration_by_protocol = pdf.groupby('order_protocol')
['delivery_duration_min'].mean()

avg_duration_by_protocol.plot(kind='bar', figsize=(12,6),
title='Average Delivery Duration by Order Protocol', color='skyblue')
plt.xlabel('Order Protocol')
plt.ylabel('Delivery Duration (minutes)')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- Protocol 6 has an exceptionally high delivery time (59 mins).
- Protocol 7 is the best performer (41 mins).
- Analyze process inefficiencies or routing strategies for Protocol 6.

Average Delivery Duration by Store Primary Category

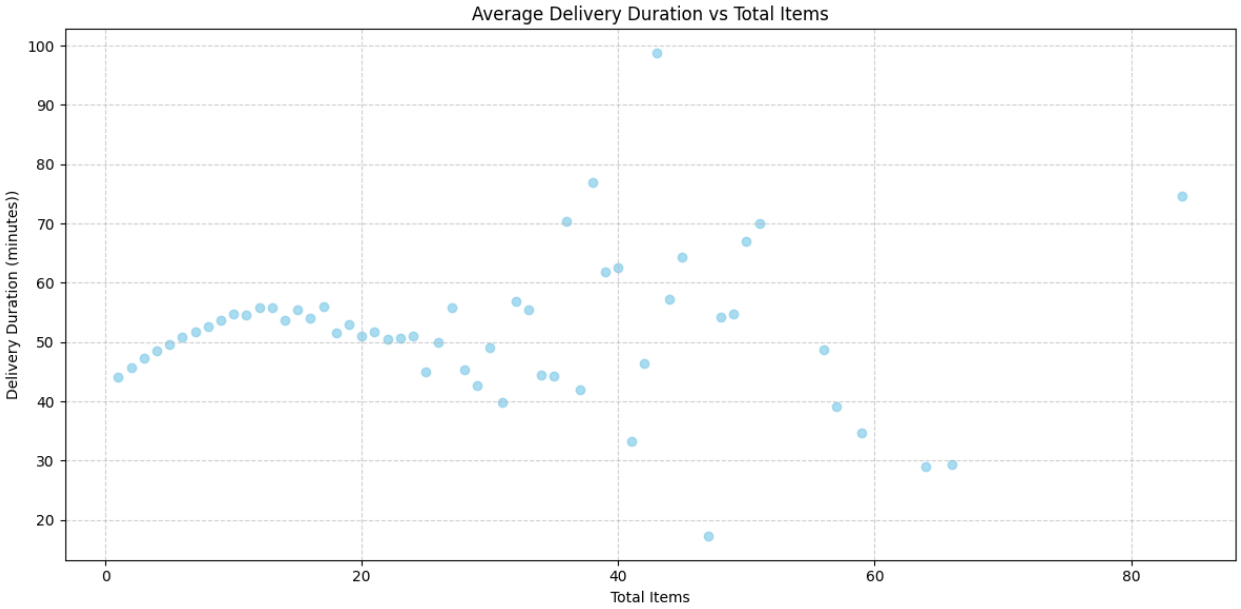
```
# average duation by store category
avg_duration_by_category = pdf.groupby('store_primary_category')
['delivery_duration_min'].mean().sort_values(ascending=False).head(10)

avg_duration_by_category.plot(kind='bar', figsize=(12,6),
title='Average Delivery Duration by Story Category', color='skyblue')
plt.xlabel('Story Primary Category')
plt.ylabel('Delivery Duration (minutes)')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- Categories like Caribbean, Belgian, Spanish, and Brazilian consistently show higher average delivery times (~54–56 mins).
- These cuisines may require more prep time. Consider prepping strategies or informing users about expected delays.

Average Delivery Duration by Total Items

```
# average duration by items
avg_duration_by_items = pdf.groupby('total_items')
['delivery_duration_min'].mean().reset_index()

plt.figure(figsize=(12, 6))
plt.scatter(avg_duration_by_items['total_items'],
avg_duration_by_items['delivery_duration_min'], color='skyblue',
alpha=0.7)
plt.title('Average Delivery Duration vs Total Items')
plt.xlabel('Total Items')
plt.ylabel('Delivery Duration (minutes))')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

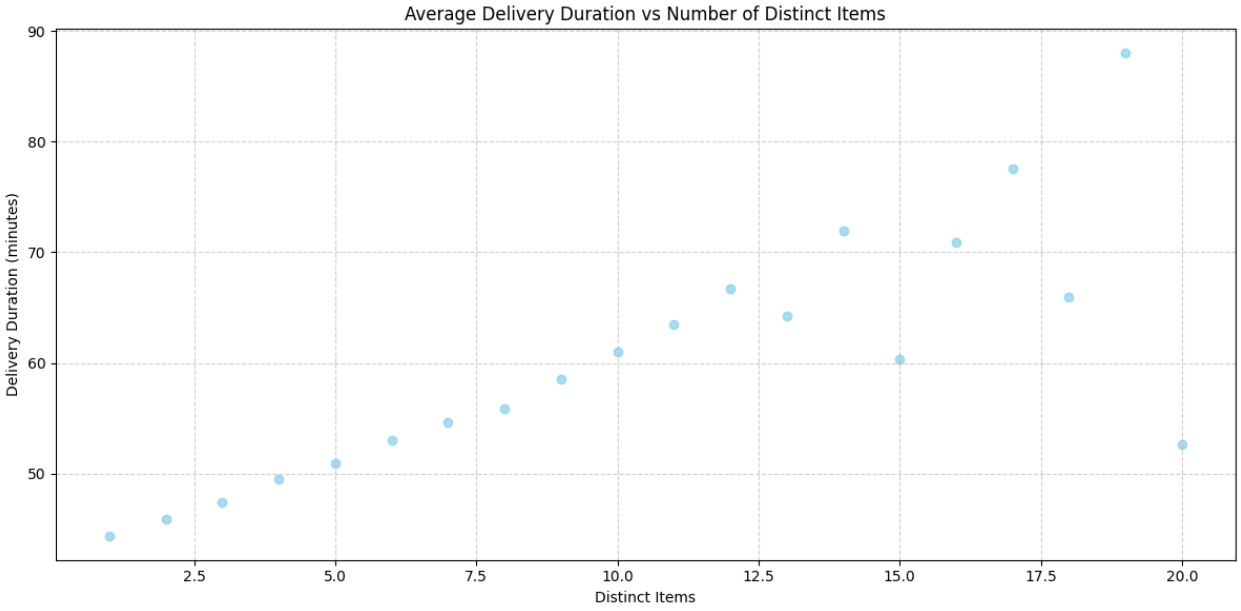


- There is a clear increasing trend in delivery time as the number of items increases.
- Orders with 36+ items see times reaching up to 98 mins.
- However, some outliers (e.g., 47 items = 17 mins) suggest inconsistencies.

- Bulk orders impact delivery time — consider batch processing or special handling for large orders.

Average Delivery Duration by Number of Distinct Items

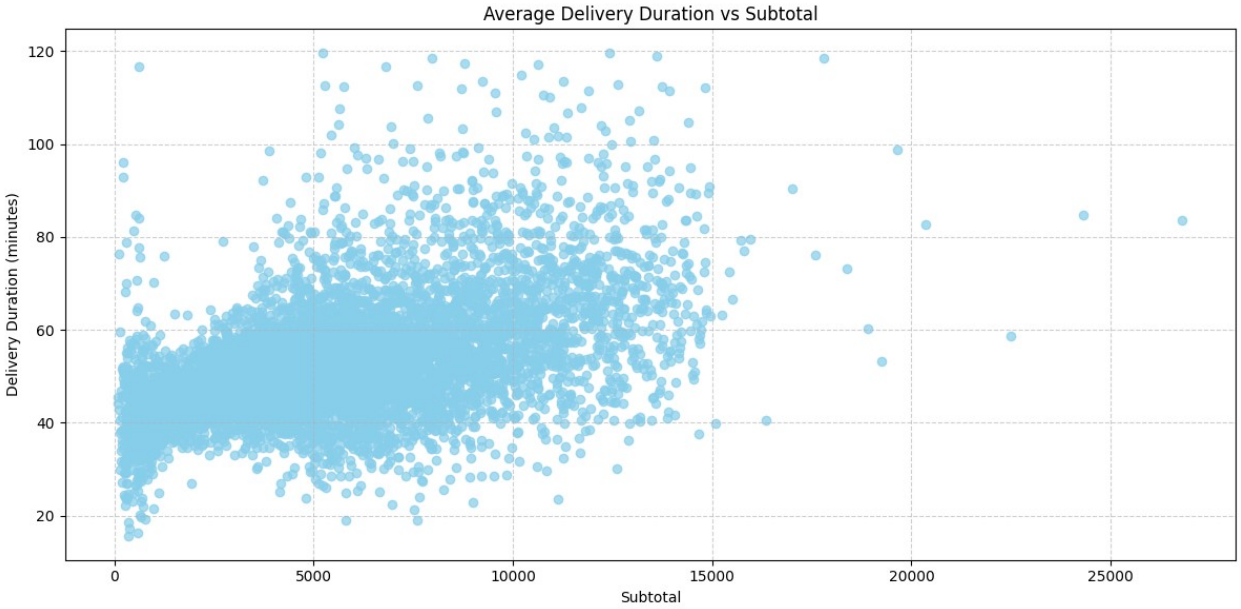
```
# average duration by distinct items
avg_duration_by_dict_items = pdf.groupby('num_distinct_items')
['delivery_duration_min'].mean().reset_index()

plt.figure(figsize=(12, 6))
plt.scatter(avg_duration_by_dict_items['num_distinct_items'],
avg_duration_by_dict_items['delivery_duration_min'], color='skyblue',
alpha=0.7)
plt.title('Average Delivery Duration vs Number of Distinct Items')
plt.xlabel('Distinct Items')
plt.ylabel('Delivery Duration (minutes)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



- There's a clear upward trend — as the number of distinct items in an order increases, average delivery duration also increases.
- This is expected, as more variety generally implies more preparation and packaging time

Average Delivery Duration by Subtotal

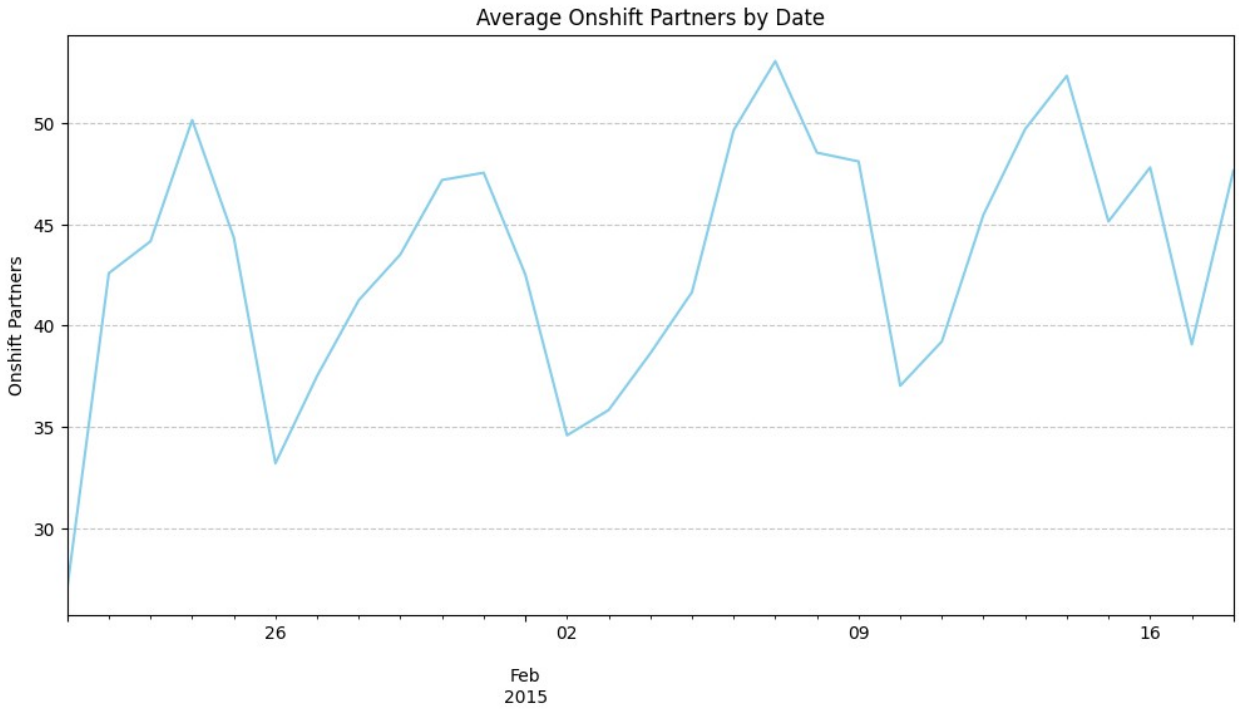
```
# average duraiton by subtotal
avg_duration_by_subtotal = pdf.groupby('subtotal')
['delivery_duration_min'].mean().reset_index()

plt.figure(figsize=(12, 6))
plt.scatter(avg_duration_by_subtotal['subtotal'],
avg_duration_by_subtotal['delivery_duration_min'], color='skyblue',
alpha=0.7)
plt.title('Average Delivery Duration vs Subtotal')
plt.xlabel('Subtotal')
plt.ylabel('Delivery Duration (minutes)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



- There is a general upward trend, indicating that as the subtotal increases, the average delivery duration also tends to increase. This could be due to larger or more complex orders requiring more preparation and delivery time.
- Some points deviate significantly from the trend. For example, orders with high subtotals but relatively low delivery durations suggest inconsistencies or exceptional cases (e.g., expedited delivery).
- The data appears to form clusters at certain subtotal ranges, possibly reflecting common order sizes or pricing tiers.
- Orders with subtotals above 20,000 show longer delivery durations, often exceeding 80 minutes. These could represent bulk or special orders.
- Orders with subtotals below 1,000 generally have shorter delivery durations, often under 50 minutes, likely due to their simplicity.

Average Onshift Partners by Date

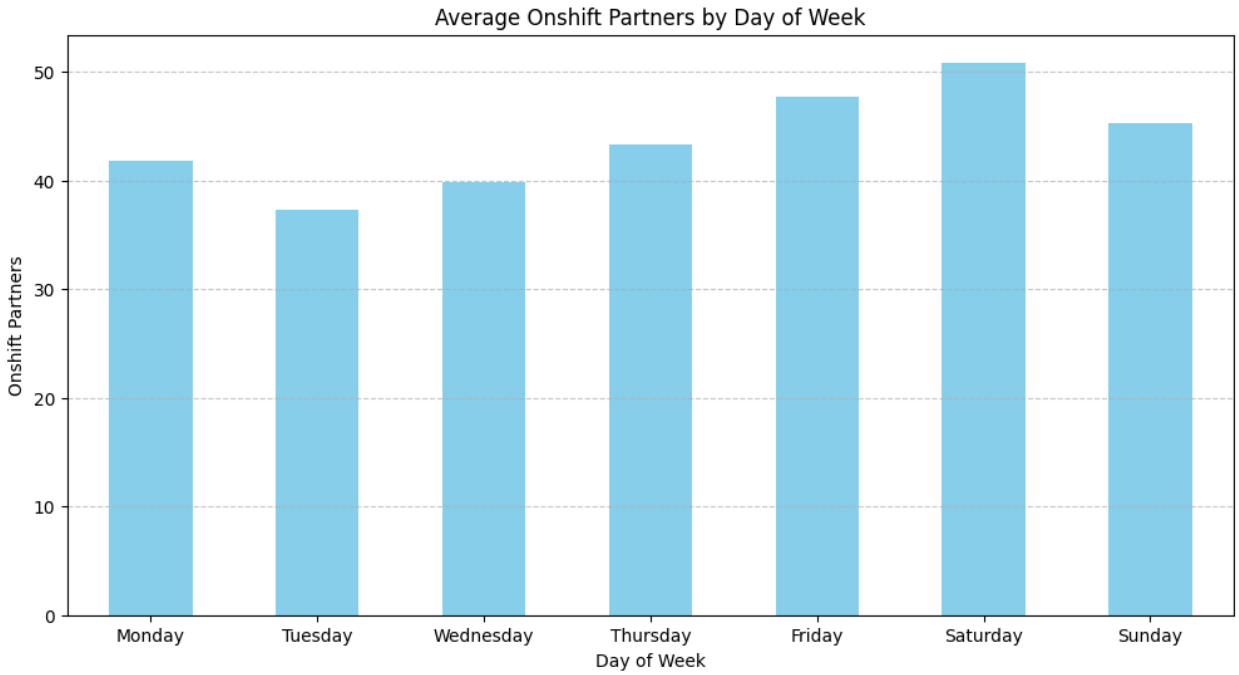
```
# average onshift partners by date
avg_onshift_by_date = pdf.groupby('date')
['total_onshift_partners'].mean()

avg_onshift_by_date.plot(kind='line', figsize=(12,6), title='Average
Onshift Partners by Date', color='skyblue')
plt.xlabel(None)
plt.ylabel('Onshift Partners')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- There is a steady rise in the number of onshift partners from Jan 21 to Feb 18, starting from around 27 and peaking over 53.
- The highest availability occurred during weekends and early February, suggesting stronger supply alignment with anticipated demand.
- Noticeable dips on Mondays and early weekdays may indicate lower partner availability or operational shifts.

Average Onshift Partners by Day of Week

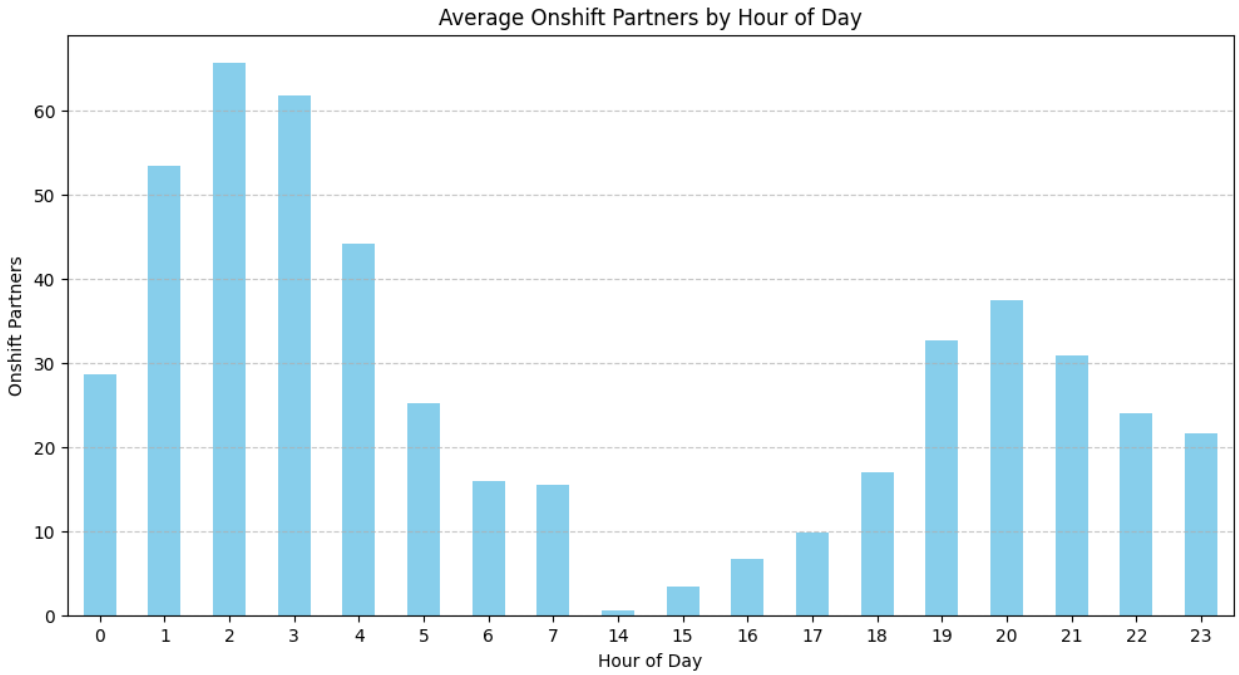
```
# average onshift partners by day
avg_onshift_by_day = (
    pdf.groupby('dayname')['total_onshift_partners']
    .mean()
    .reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
    )
)

avg_onshift_by_day.plot(kind='bar', figsize=(12, 6), title='Average
Onshift Partners by Day of Week', color='skyblue')
plt.xlabel('Day of Week')
plt.ylabel('Onshift Partners')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- Saturday (56) and Friday (48) had the highest onshift partner averages, aligning with peak demand days.
- The lowest average is on Tuesday (37), suggesting potential under-capacity midweek.
- There's a clear increase in capacity toward the weekend, likely to handle higher delivery volumes.

Average Onshift Partners by Hour of Day

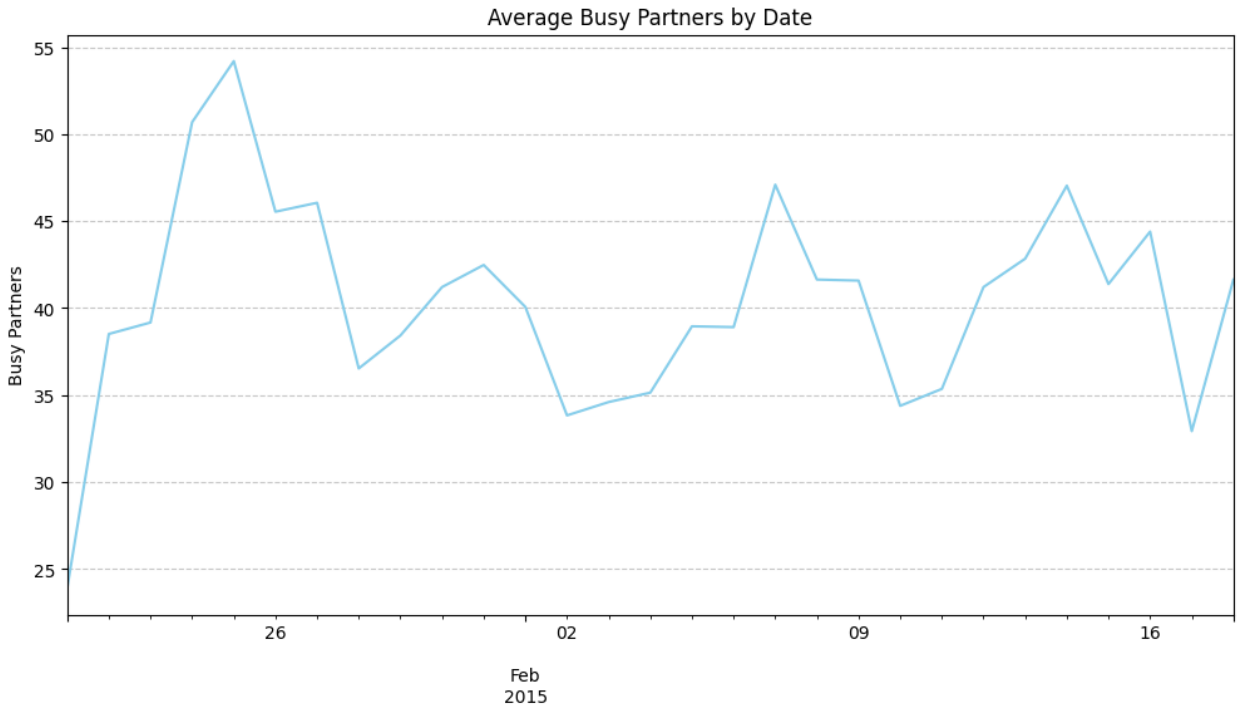
```
# average onshift partners by hour
avg_onshift_by_hour = (pdf.groupby('hour')
['total_onshift_partners'].mean())

avg_onshift_by_hour.plot(kind='bar', figsize=(12,6), title='Average
Onshift Partners by Hour of Day', color='skyblue')
plt.xlabel('Hour of Day')
plt.ylabel('Onshift Partners')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.show()
```



- Peak partner availability happens between 1 AM to 3 AM (66 at 2 AM), likely due to system or batch delivery times.
- Partner count drastically reduces post 4 AM, hitting lows around 7 AM to 8 AM.
- Evening hours (7 PM–9 PM) show a moderate rise again, reflecting increased demand during dinner-time deliveries.

Average Busy Partners by Date

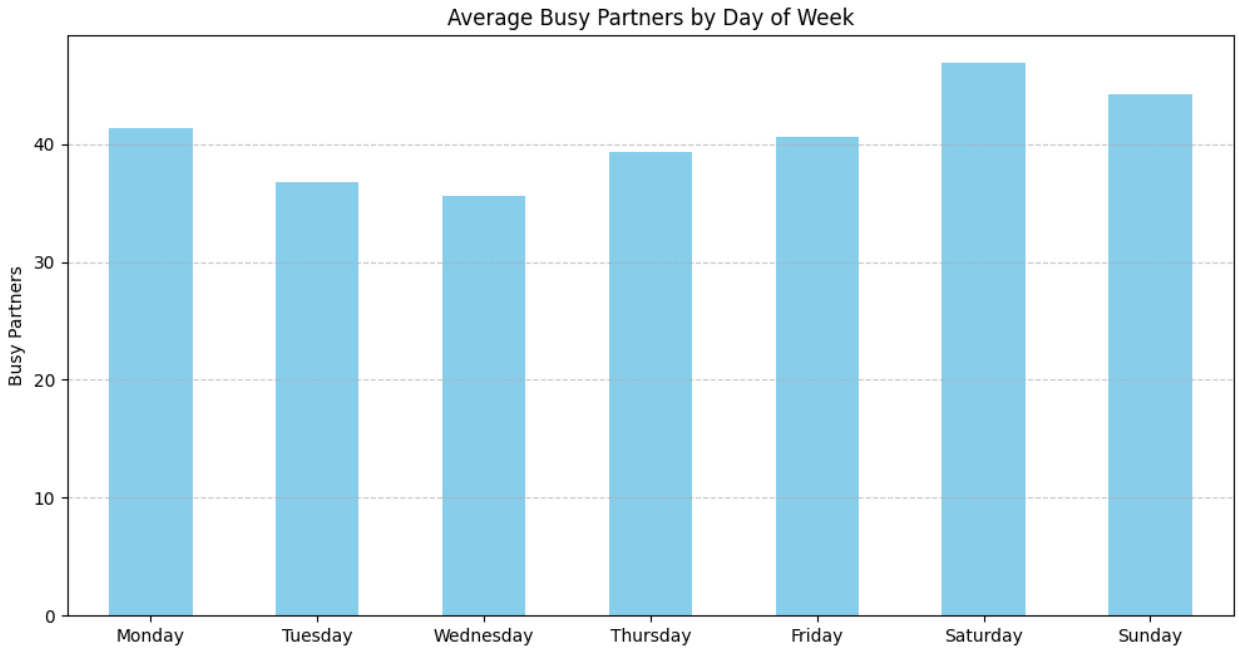
```
# average busy partners by date
avg_busy_by_date = pdf.groupby('date')['total_busy_partners'].mean()

avg_busy_by_date.plot(kind='line', figsize=(12,6), title='Average Busy
Partners by Date', color='skyblue')
plt.xlabel(None)
plt.ylabel('Busy Partners')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- The number of busy partners closely mirrors onshift trends, peaking around weekends.
- Highest busy partner count observed on Jan 25 (54) and Feb 7 (47).
- Some days like Feb 3–4 saw busy partner counts lower than onshift, hinting at potential overstaffing or lower demand.

Average Busy Partners by Day of Week

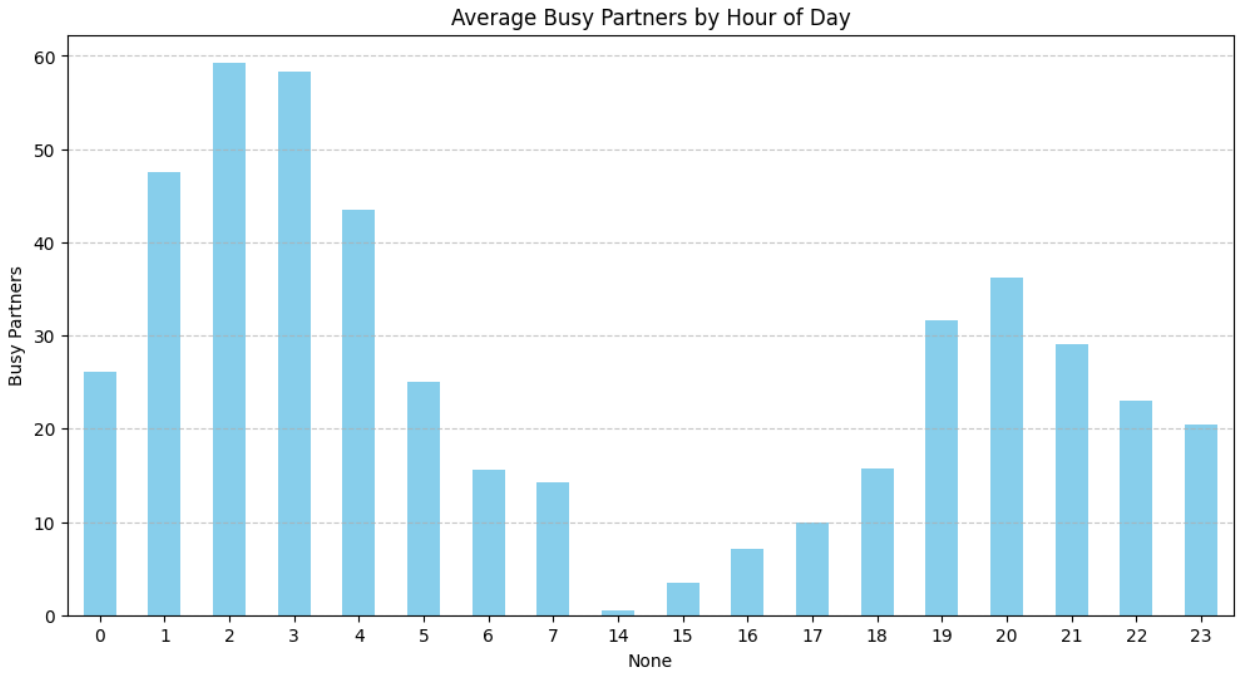
```
# average busy partners by day
avg_busy_by_day = (
    pdf.groupby('dayname')['total_busy_partners']
    .mean()
    .reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
    )
)

avg_busy_by_day.plot(kind='bar', figsize=(12, 6), title='Average Busy
Partners by Day of Week', color='skyblue')
plt.xlabel(None)
plt.ylabel('Busy Partners')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- Saturday (47) and Sunday (44) again stand out, confirming weekends as busiest.
- Wednesday (36) has the lowest average busy partners, aligned with lower delivery activity or supply-demand mismatch.
- The weekday dip suggests an opportunity to optimize resource allocation across the week.

Average Busy Partners by Hour of Day

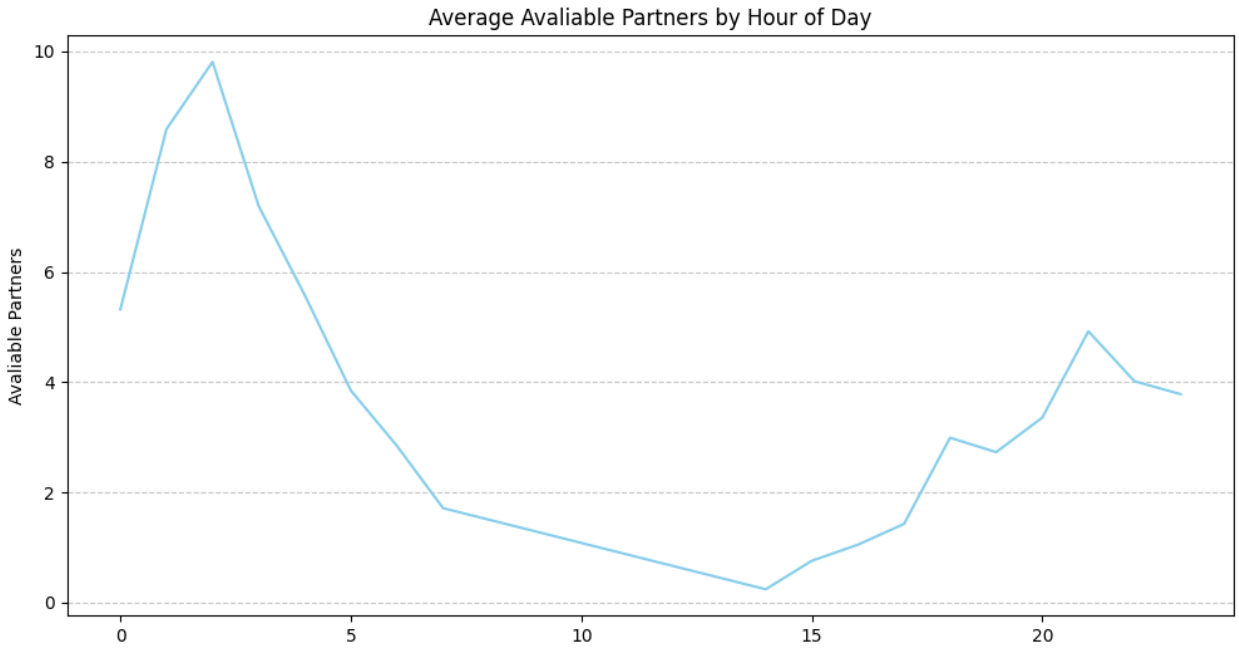
```
# average busy partnerss by hour
avg_busy_by_hour = pdf.groupby('hour')['total_busy_partners'].mean()

avg_busy_by_hour.plot(kind='bar', figsize=(12,6), title='Average Busy
Partners by Hour of Day', color='skyblue')
plt.xlabel('None')
plt.ylabel('Busy Partners')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.show()
```



- Busy partner trends follow onshift trends closely, peaking at 2 AM (59) and staying high until 4 AM.
- Drop-off starts sharply post-4 AM, with the lowest engagement around 2 PM–5 PM.
- Evenings see a small resurgence, but not as high as early morning hours.

Average Available Partner by Hour of Day

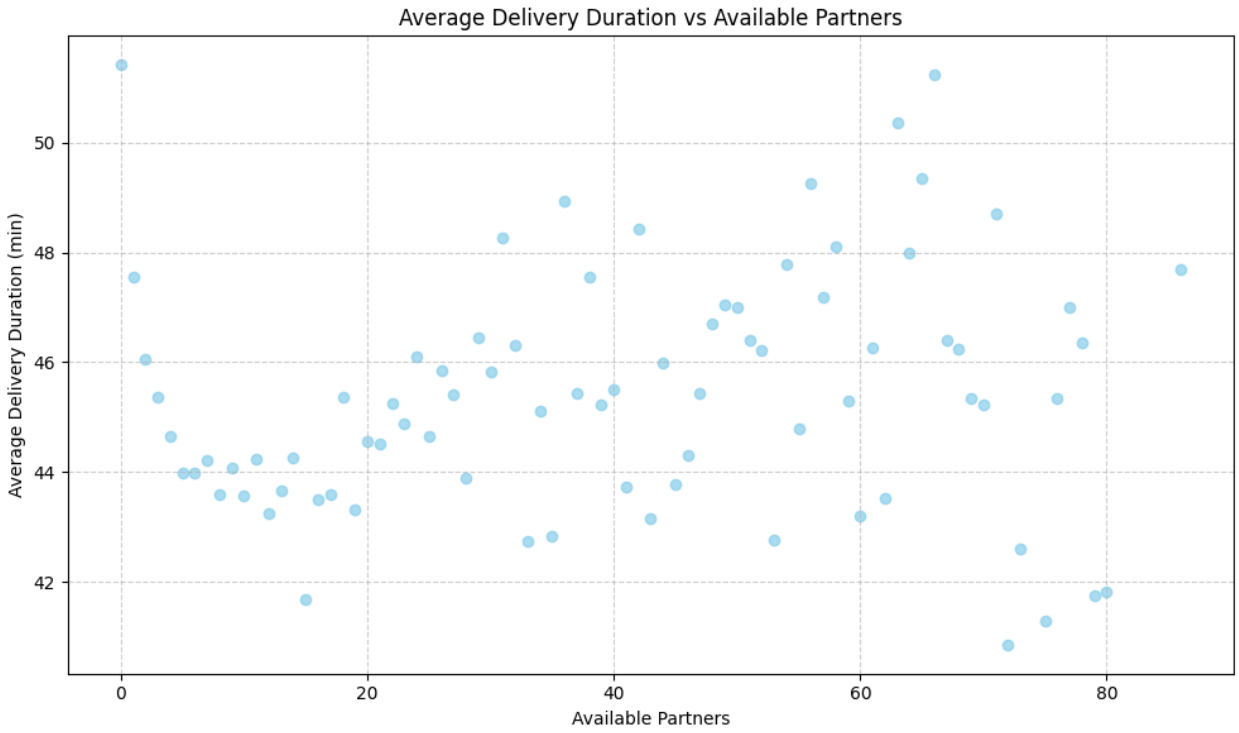
```
# average available partners (onshit partners - busy partners) by hour
available_partner_by_hour = pdf.groupby('hour')
['available_partners'].mean()

available_partner_by_hour.plot(kind='line', figsize=(12,6),
title='Average Avaliable Partners by Hour of Day', color='skyblue')
plt.xlabel(None)
plt.ylabel('Avaliable Partners')
plt.xticks(rotation=0)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



- Availability is highest between 2–3 AM, but even then, only about 9 partners are idle, showing high engagement levels.
- During evening hours (7–11 PM), available partners remain quite low (mostly 2–4), indicating a tightly stretched capacity.
- The lowest availability is seen during lunch-to-evening transition (2–6 PM), potentially a risk zone for delays.

Available Partners vs Average Delivery Duration

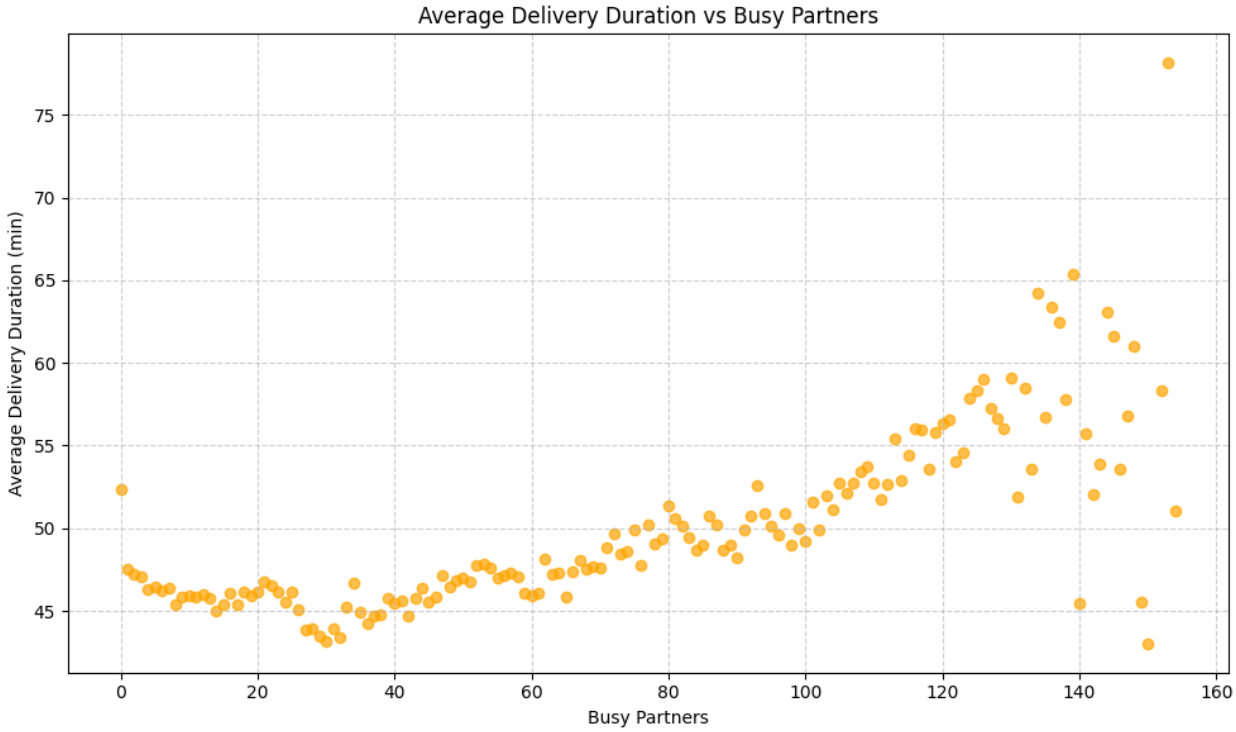
```
# available partners vs delivery duration
avg_duration_by_av_prt = pdf.groupby('available_partners')
['delivery_duration_min'].mean().reset_index()

plt.figure(figsize=(10, 6))
plt.scatter(avg_duration_by_av_prt['available_partners'],
avg_duration_by_av_prt['delivery_duration_min'], color='skyblue',
alpha=0.7)
plt.title('Average Delivery Duration vs Available Partners')
plt.xlabel('Available Partners')
plt.ylabel('Average Delivery Duration (min)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



- When no partners are available, average delivery time is the highest at ~51.4 minutes, indicating possible delays due to resource unavailability.
- As the number of available partners increases, delivery duration gradually decreases, reaching around 41–45 minutes for optimal partner availability.
- Interestingly, even with 70+ partners available, delivery times plateau or slightly increase, implying diminishing returns or potential inefficiencies in partner utilization.
- This suggests a sweet spot exists where having more partners helps, but only up to a certain point – after which system coordination matters more.

Busy Partners vs Average Delivery Duration

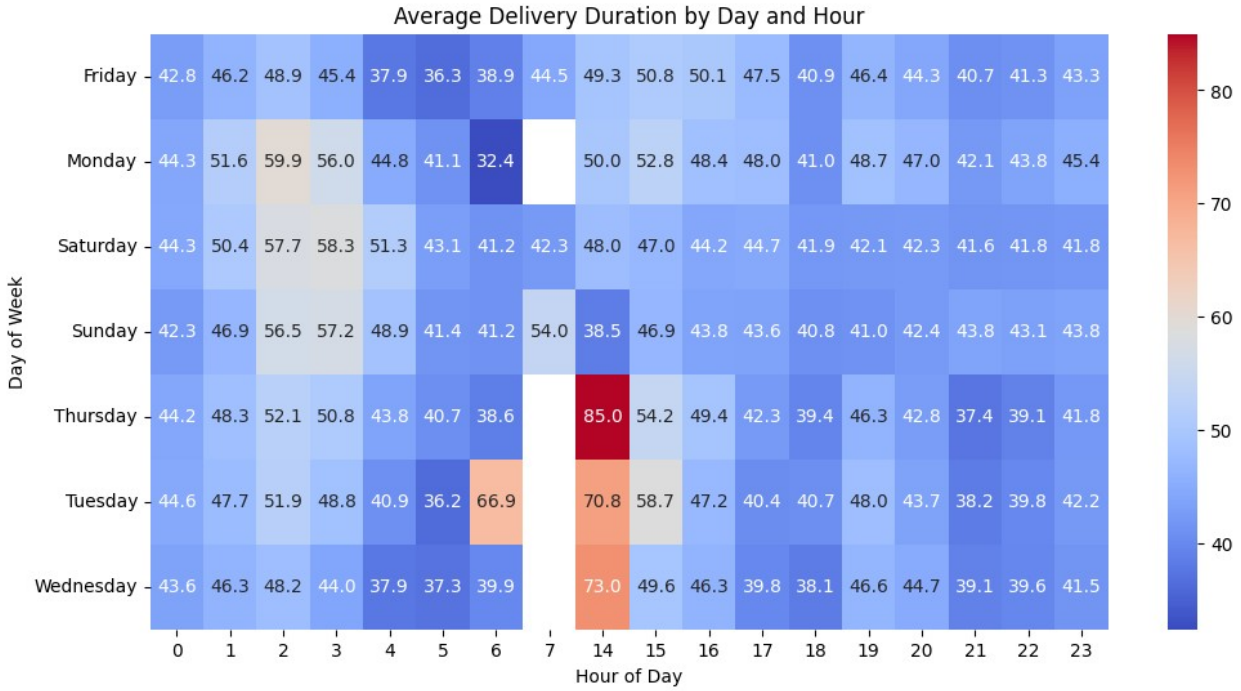
```
# busy partners vs delivery duration
avg_duration_by_bu_prt = pdf.groupby('total_busy_partners')
['delivery_duration_min'].mean().reset_index()

plt.figure(figsize=(10, 6))
plt.scatter(avg_duration_by_bu_prt['total_busy_partners'],
avg_duration_by_bu_prt['delivery_duration_min'], color='orange',
alpha=0.7)
plt.title('Average Delivery Duration vs Busy Partners')
plt.xlabel('Busy Partners')
plt.ylabel('Average Delivery Duration (min)')
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



- Similar to the available partners trend, when zero busy partners are present, delivery duration is high (~52.3 minutes), likely due to lower operational activity.
- As more partners become busy (indicating more active deliveries), average delivery time decreases steadily to the mid-40-minute range.
- At very high busy partner counts (150+), delivery times become inconsistent, with a spike at 152 partners (58 mins) and 153 partners (78 mins), hinting at overload or system strain.
- This reflects that while more activity generally means better efficiency, extreme partner busyness may cause bottlenecks, impacting delivery performance.

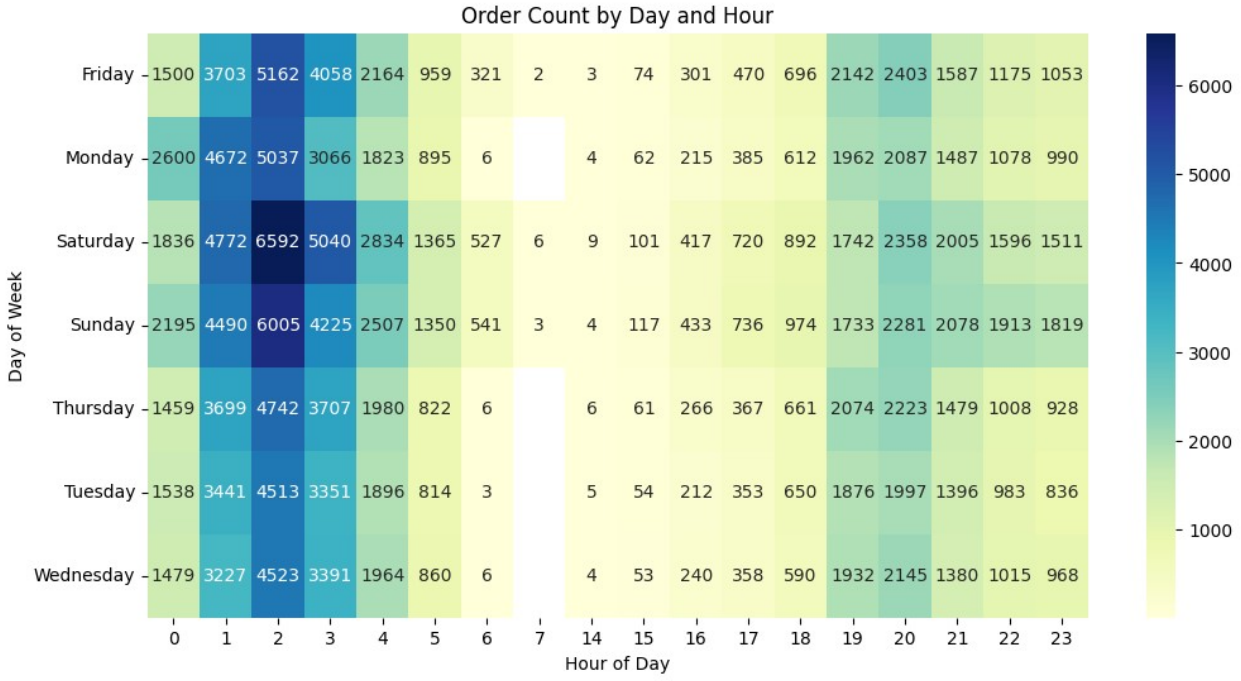
Hour of Day vs Day of Week (Avg Delivery Duration & Order Volume)

```
# pivot table for average delivery duration
pivot_duration = pdf.pivot_table(values='delivery_duration_min',
index='dayname', columns='hour', aggfunc='mean')

plt.figure(figsize=(12, 6))

sns.heatmap(pivot_duration, cmap='coolwarm', annot=True, fmt=".1f")
plt.title('Average Delivery Duration by Day and Hour')
plt.ylabel('Day of Week')
plt.xlabel('Hour of Day')
plt.show()
```



- Delivery durations are highest during these hours across all days, especially on weekends (Saturday and Sunday).
- This could be due to increased demand during late-night hours or reduced partner availability.
- Delivery durations are shortest during these hours on most days, indicating better resource allocation and lower demand variability.
- Monday and Saturday show consistently higher delivery durations across most hours, suggesting potential inefficiencies or higher demand on these days.
- Wednesday and Thursday have relatively lower delivery durations, indicating better operational efficiency midweek.
- Delivery durations spike significantly between 1 AM–4 AM, especially on weekends (Saturday and Sunday). This could be due to late-night orders or reduced partner availability.

```
# pivot table for order volume
pivot_orders = pdf.pivot_table(values='created_at', index='dayname',
columns='hour', aggfunc='count')

plt.figure(figsize=(12, 6))

sns.heatmap(pivot_orders, cmap='YlGnBu', annot=True, fmt=".0f")
plt.title('Order Count by Day and Hour')
plt.ylabel('Day of Week')
plt.xlabel('Hour of Day')
plt.show()
```

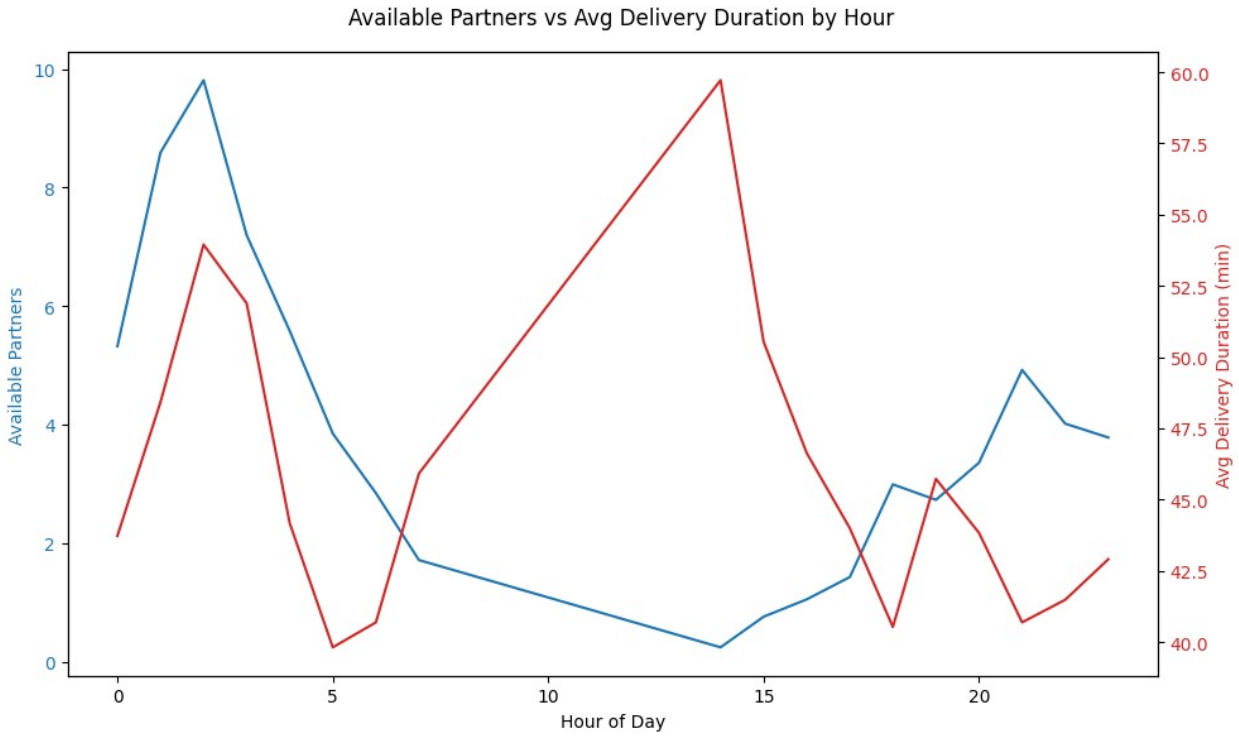


- Order volumes are highest during these hours, especially on weekends (Saturday and Sunday), reflecting late-night demand.
- Moderate order volumes are observed during these hours, likely due to dinner-time orders.
- Saturdays and Sundays generally have higher order volumes compared to weekdays, reflecting increased demand and potential strain on resources.
- Early morning hours (6 AM–2 PM) show the lowest order volumes, with a dip to almost zero between 7 AM–2 PM.

Available Partners vs Hour of Day with Delivery Duration

```
# avaliable partners vs hour with duration
hourly = pdf.groupby('hour').agg({
    'available_partners': 'mean',
    'delivery_duration_min': 'mean'
}).reset_index()


fig, ax1 = plt.subplots(figsize=(10, 6))

color1 = 'tab:blue'
ax1.set_xlabel('Hour of Day')
ax1.set_ylabel('Available Partners', color=color1)
ax1.plot(hourly['hour'], hourly['available_partners'], color=color1,
label='Available Partners')
ax1.tick_params(axis='y', labelcolor=color1)

ax2 = ax1.twinx()
color2 = 'tab:red'
```

```
ax2.set_ylabel('Avg Delivery Duration (min)', color=color2)
ax2.plot(hourly['hour'], hourly['delivery_duration_min'],
color=color2, label='Avg Delivery Duration')
ax2.tick_params(axis='y', labelcolor=color2)

fig.suptitle('Available Partners vs Avg Delivery Duration by Hour')
fig.tight_layout()
plt.show()
```



- The late-night hours (1 AM–3 AM) show high demand and partner availability, but delivery durations remain relatively high, suggesting a need for better resource allocation or process optimization.
- Delivery durations are shortest during the evening hours (7 PM–9 PM), indicating efficient operations during this time.
- During periods of high partner availability (e.g., 2 AM), delivery durations do not decrease significantly, suggesting diminishing returns or inefficiencies in partner utilization.

Recommendations

Dynamic Partner Allocation

- Late-night (1–4 AM) and weekend demand surges cause delays due to partner shortages.
- When more delivery partners are available, deliveries are faster.
- If there are very few or no partners available, delivery times increase.
- Move delivery partners to areas where more orders are expected, especially during midnight and dinner.

- In short, boost partner availability during weekends (Fri–Sun) and late nights (1–4 AM) with: Bonus pay for delivery partners, using surge pricing for customers (to manage demand)

Restaurant Level Interventions

- Some food types get more orders, like American food, Pizza, Mexican.
- Since these food types are in high demand, they may take longer to prepare.
- Slow prep times for certain cuisines (Caribbean, Belgian) and bulk orders (36+ items).
- Suggested pre-prep ingredients during peak hours.
- Add a fee for orders >10 items to compensate for extra delivery time.

Order Protocol Optimization

- Protocol 6 has 59-min avg. delivery time vs. 41 mins for Protocol 7.
- Investigate if it's a legacy system (e.g., call-in orders) and migrate users to faster channels (app/web).
- Offer discounts for customers using Protocol 3 (avg. 45 mins).

Demand Shaping & Customer Communication

- 2–4 AM orders have high demand but long delivery times (54+ mins).
- Add a 10% fee for 1–4 AM orders to smooth demand.
- Incentivize 6–10 AM orders with 15% off to balance demand.

Operational Efficiency Fixes

- Mondays/Saturdays have 50+ min deliveries despite moderate demand.
- Audit traffic routes on these days for bottlenecks.
- Assign dedicated "fast-response" partners for high-priority orders.

Data-Driven Partner Thresholds

- Delivery times spike when available partners <5 or outstanding orders >50.
- Maintain 10% idle partners during peak hours for flexibility.