Order Delivery Time Prediction

Objectives

The objective of this assignment is to build a regression model that predicts the delivery time for orders placed through Porter. The model will use various features such as the items ordered, the restaurant location, the order protocol, and the availability of delivery partners.

The key goals are:

- Predict the delivery time for an order based on multiple input features
- Improve delivery time predictions to optimiae operational efficiency
- Understand the key factors influencing delivery time to enhance the model's accuracy

Data Pipeline

The data pipeline for this assignment will involve the following steps:

1. **Data Loading**
2. **Data Preprocessing and Feature Engineering**
3. **Exploratory Data Analysis**
4. **Model Building**
5. **Model Inference**

Data Understanding

The dataset contains information on orders placed through Porter, with the following columns:

Field	Description
market_id	Integer ID representing the market where the restaurant is located.
created_at	Timestamp when the order was placed.
actual_delivery_time	Timestamp when the order was delivered.
store_primary_category	Category of the restaurant (e.g., fast food, dine-in).
order_protocol	Integer representing how the order was placed (e.g., via Porter, call to restaurant, etc.).
total_items	Total number of items in the order.
subtotal	Final price of the order.
num_distinct_items	Number of distinct items in the order.
min_item_price	Price of the cheapest item in the order.
max_item_price	Price of the most expensive item in the order.
total_onshift_dashers	Number of delivery partners on duty when the order was placed.
total_busy_dashers	Number of delivery partners already occupied with other orders.
total_outstanding_orders	Number of orders pending fulfillment at the time of the order.
distance	Total distance from the restaurant to the customer.

Importing Necessary Libraries

```
from logging import warning
# Import essential libraries for data manipulation and analysis
import pandas as pd
import numpy as np

# Import libraries for data visualization
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels
import statsmodels.api as sm
import sklearn
from sklearn.model_selection import train_test_split

import warnings
```

```
warnings.filterwarnings('ignore')
```

1. Loading the data

Load 'porter_data_1.csv' as a DataFrame

```
# Importing the file porter_data_1.csv
porter = pd.read_csv('/content/sample_data/porter_data_1.csv')
```

```
porter.head()
```

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_
0	1.0	2015-02-06 22:24:17	2015-02-06 23:11:17	4	1.0	4	3441	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:33:25	46	2.0	1	1900	1	
2	2.0	2015-02-16 00:11:35	2015-02-16 01:06:35	36	3.0	4	4771	3	
3	1.0	2015-02-12 03:36:46	2015-02-12 04:35:46	38	1.0	1	1525	1	
4	1.0	2015-01-27 02:12:36	2015-01-27 02:58:36	38	1.0	2	3620	2	

```
porter.info()
```

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

```
categorical_columns = porter.select_dtypes(include=['object']).columns
print("Categorical columns:", categorical_columns)
```

```
Categorical columns: Index(['created_at', 'actual_delivery_time'], dtype='object')
```

```
porter.isnull().sum()
```

	0
market_id	0
created_at	0
actual_delivery_time	0
store_primary_category	0
order_protocol	0
total_items	0
subtotal	0
num_distinct_items	0
min_item_price	0
max_item_price	0
total_onshift_dashers	0
total_busy_dashers	0
total_outstanding_orders	0
distance	0

dtype: int64

2. Data Preprocessing and Feature Engineering [15 marks]

2.1 Fixing the Datatypes [5 marks]

The current timestamps are in object format and need conversion to datetime format for easier handling and intended functionality

2.1.1 [2 marks]

Convert date and time fields to appropriate data type

```
# Convert 'created_at' and 'actual_delivery_time' columns to datetime format
portar = porter.copy()
portar['created_at'] = pd.to_datetime(portar['created_at'])
portar['actual_delivery_time'] = pd.to_datetime(portar['actual_delivery_time'])
```

```
portar.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 175777 entries, 0 to 175776
Data columns (total 14 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   market_id                175777 non-null  float64
 1   created_at               175777 non-null  datetime64[ns]
 2   actual_delivery_time     175777 non-null  datetime64[ns]
 3   store_primary_category   175777 non-null  int64
 4   order_protocol           175777 non-null  float64
 5   total_items              175777 non-null  int64
 6   subtotal                 175777 non-null  int64
 7   num_distinct_items       175777 non-null  int64
 8   min_item_price           175777 non-null  int64
 9   max_item_price           175777 non-null  int64
 10  total_onshift_dashers    175777 non-null  float64
 11  total_busy_dashers       175777 non-null  float64
 12  total_outstanding_orders 175777 non-null  float64
 13  distance                 175777 non-null  float64
dtypes: datetime64[ns](2), float64(6), int64(6)
memory usage: 18.8 MB
```

2.1.2 [3 marks]

Convert categorical fields to appropriate data type

```
# Convert categorical features to category type
categorical_columns = portar.select_dtypes(include=['object']).columns
print("Categorical columns:", categorical_columns)
```

```
Categorical columns: Index([], dtype='object')
```

2.2 Feature Engineering [5 marks]

Calculate the time taken to execute the delivery as well as extract the hour and day at which the order was placed

2.2.1 [2 marks]

Calculate the time taken using the features `actual_delivery_time` and `created_at`

```
from datetime import time
# Calculate time taken in minutes
time_taken = (portar['actual_delivery_time'] - portar['created_at']).dt.total_seconds() / 60
portar['time_taken'] = time_taken
```

```
portar.head()
```

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_
0	1.0	2015-02-06 22:24:17	2015-02-06 23:11:17	4	1.0	4	3441	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:33:25	46	2.0	1	1900	1	
2	2.0	2015-02-16 00:11:35	2015-02-16 01:06:35	36	3.0	4	4771	3	
3	1.0	2015-02-12 03:36:46	2015-02-12 04:35:46	38	1.0	1	1525	1	
4	1.0	2015-01-27 02:12:36	2015-01-27 02:58:36	38	1.0	2	3620	2	

2.2.2 [3 marks]

Extract the hour at which the order was placed and which day of the week it was. Drop the unnecessary columns.

```
# Extract the hour and day of week from the 'created_at' timestamp
hour_of_day = portar['created_at'].dt.hour
day_of_week = portar['created_at'].dt.dayofweek

# Create a categorical feature 'isWeekend'
is_weekend = portar['created_at'].dt.dayofweek >= 5

# Create a new DataFrame with the extracted features
df_features = pd.DataFrame({
    'hour_of_day': hour_of_day,
    'day_of_week': day_of_week,
    'is_weekend': is_weekend
})

portar = pd.concat([portar, df_features], axis=1)
portar.head()
```

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_
0	1.0	2015-02-06 22:24:17	2015-02-06 23:11:17	4	1.0	4	3441	4	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:33:25	46	2.0	1	1900	1	
2	2.0	2015-02-16 00:11:35	2015-02-16 01:06:35	36	3.0	4	4771	3	
3	1.0	2015-02-12 03:36:46	2015-02-12 04:35:46	38	1.0	1	1525	1	
4	1.0	2015-01-27 02:12:36	2015-01-27 02:58:36	38	1.0	2	3620	2	

```
# Drop unnecessary columns
portar = portar.drop(['created_at', 'actual_delivery_time'], axis=1)
portar.head()
```

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_
0	1.0	4	1.0	4	3441	4	557	1239	
1	2.0	46	2.0	1	1900	1	1400	1400	
2	2.0	36	3.0	4	4771	3	820	1604	
3	1.0	38	1.0	1	1525	1	1525	1525	
4	1.0	38	1.0	2	3620	2	1425	2195	

2.3 Creating training and validation sets [5 marks]

2.3.1 [2 marks]

Define target and input features

```
# Define target variable (y) and features (X)
y_target = portar['time_taken']
X_features = portar.drop('time_taken', axis=1)
```

2.3.2 [3 marks]

Split the data into training and test sets

```
# Split data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X_features, y_target, train_size=0.7, random_state=100)
print(X_train.shape, X_test.shape)
```

```
(123043, 15) (52734, 15)
```

```
y_train.head()
```

	time_taken
94746	41.0
173338	41.0
37592	44.0
42763	40.0
27506	44.0

dtype: float64

```
X_train.head()
```

	market_id	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	t
94746	4.0	24	5.0	2	1790	2	795	995	
173338	4.0	72	5.0	1	845	1	795	795	
37592	4.0	55	5.0	1	1900	1	1200	1200	
42763	2.0	28	4.0	6	463	3	0	299	
27506	2.0	72	1.0	3	3500	3	600	1200	

3. Exploratory Data Analysis on Training Data [20 marks]

1. Analyzing the correlation between variables to identify patterns and relationships
2. Identifying and addressing outliers to ensure the integrity of the analysis
3. Exploring the relationships between variables and examining the distribution of the data for better insights

3.1 Feature Distributions [7 marks]

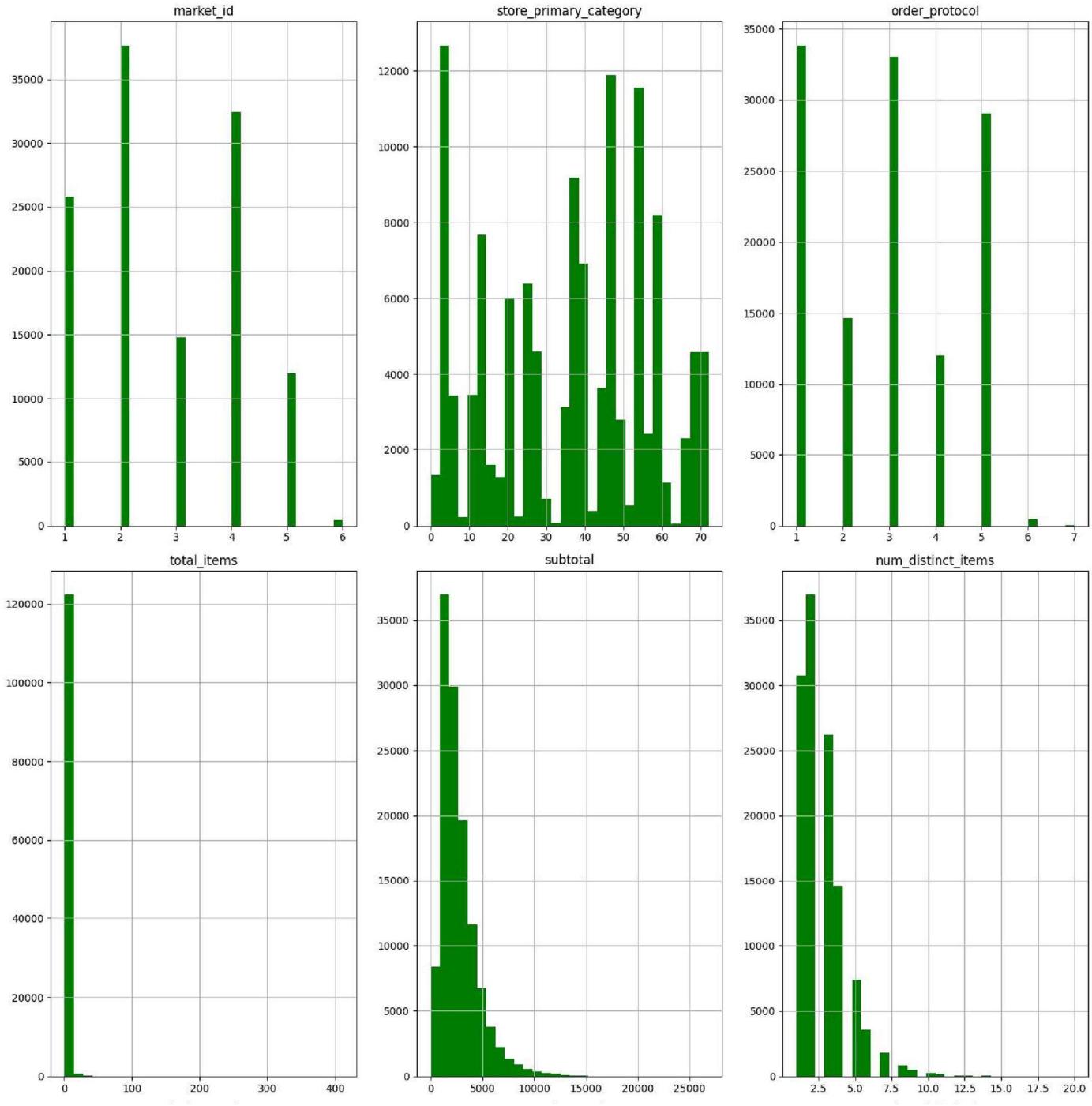
```
# Define numerical and categorical columns for easy EDA and data manipulation
numerical_columns = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_columns = X_train.select_dtypes(include=['object']).columns
print("Numerical columns:", numerical_columns)
print("Categorical columns:", categorical_columns)
```

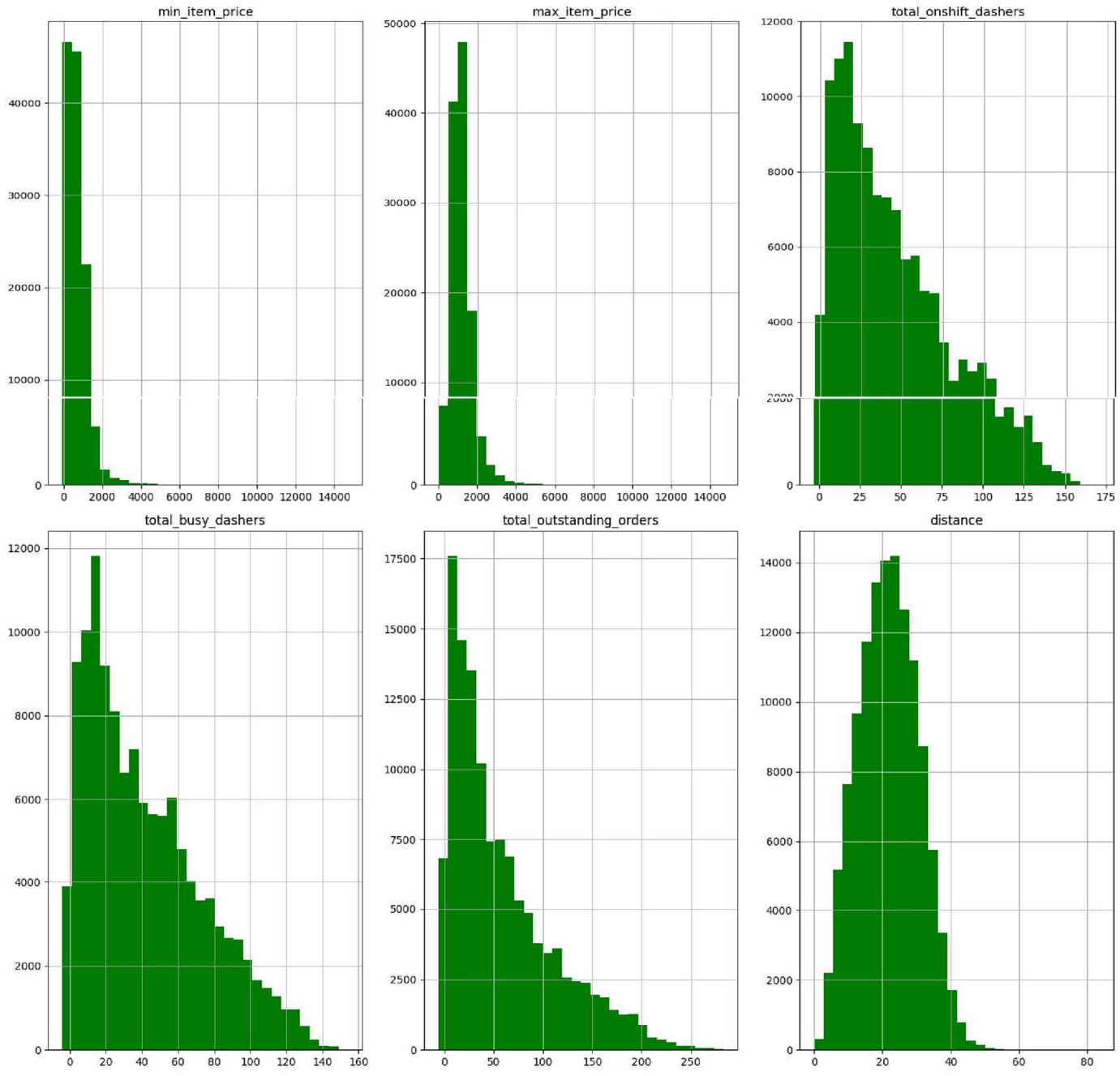
```
Numerical columns: Index(['market_id', 'store_primary_category', 'order_protocol', 'total_items',
       'subtotal', 'num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_dashers', 'total_busy_dashers',
       'total_outstanding_orders', 'distance'],
      dtype='object')
Categorical columns: Index([], dtype='object')
```

3.1.1 [3 marks]

Plot distributions for numerical columns in the training set to understand their spread and any skewness

```
# Plot distributions for all numerical columns

X_train[numerical_columns].hist(bins=30, figsize=(15, 30),color='green')
plt.tight_layout()
plt.show()
```


market_id
store_primary_category
order_protocol
total_items
subtotal
num_distinct_items


min_item_price
max_item_price
total_onshift_dashers
total_busy_dashers
total_outstanding_orders
distance

```
    sns.scatterplot(x=X_train[i], y=y_train)
    plt.show()
```

3.1.2 [2 marks]

Check the distribution of categorical features

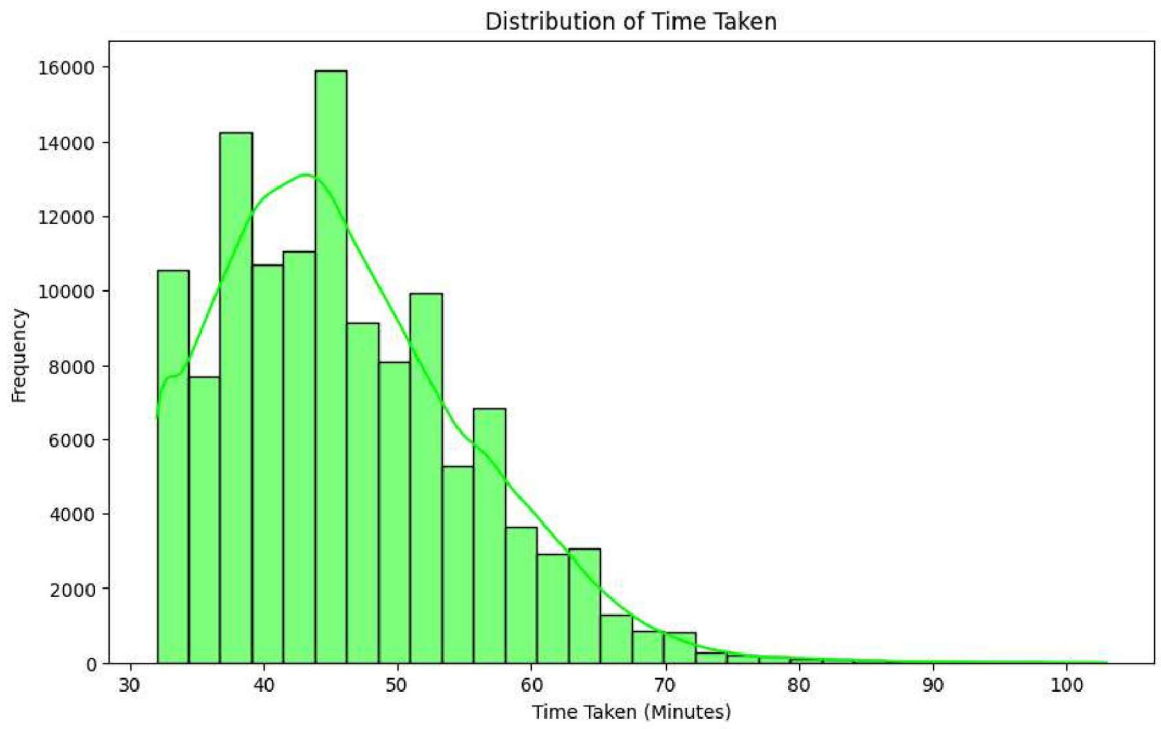
```
# Distribution of categorical columns
X_train[categorical_columns].nunique()
x_train_cat = X_train[categorical_columns]
x_train_cat.head()
```

94746	
173338	
37592	
42763	
27506	

3.1.3 [2 mark]

Visualise the distribution of the target variable to understand its spread and any skewness

```
# Distribution of time_taken
plt.figure(figsize=(10, 6))
sns.histplot(y_train, bins=30, kde=True, color='lime')
plt.title('Distribution of Time Taken')
plt.xlabel('Time Taken (Minutes)')
plt.ylabel('Frequency')
plt.show()
```

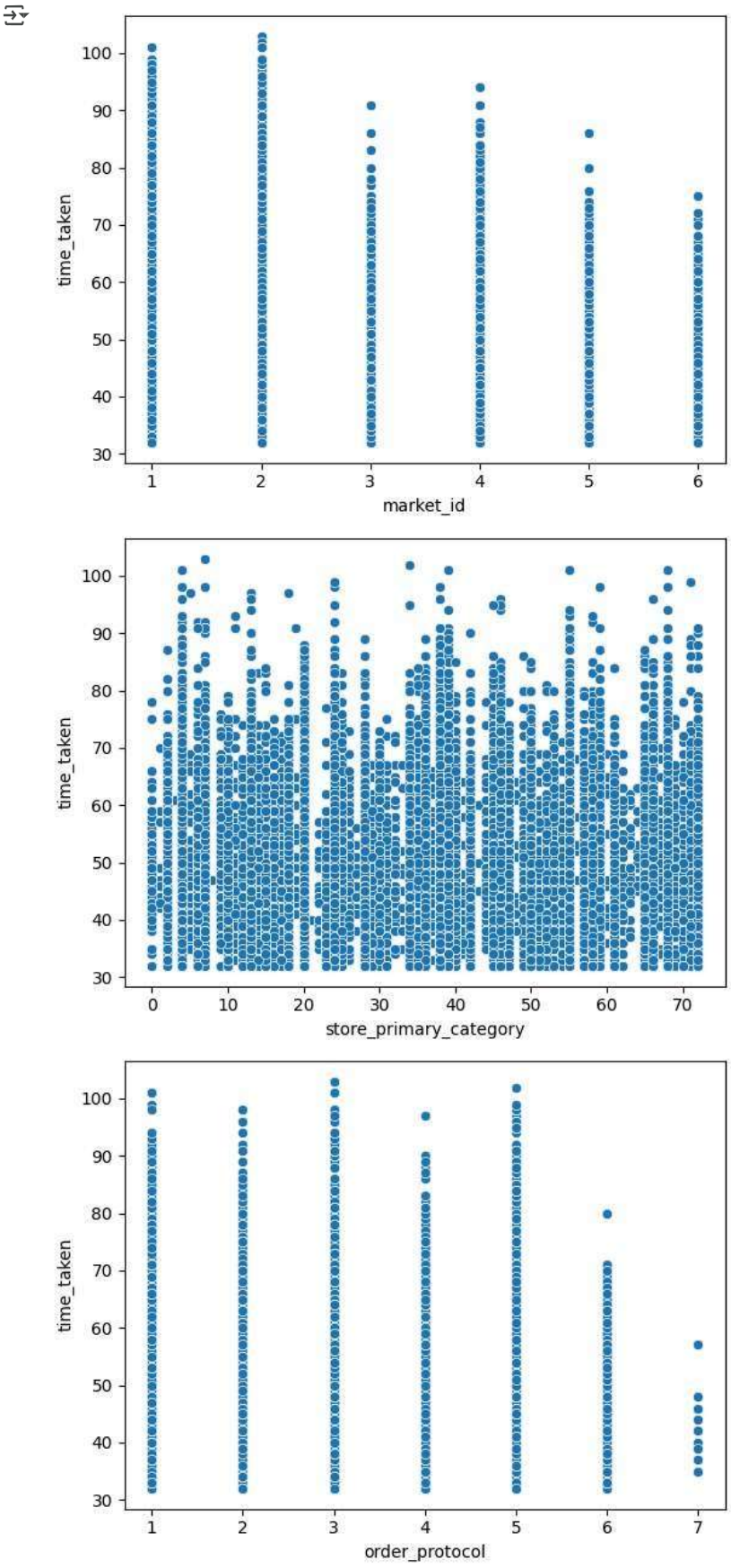


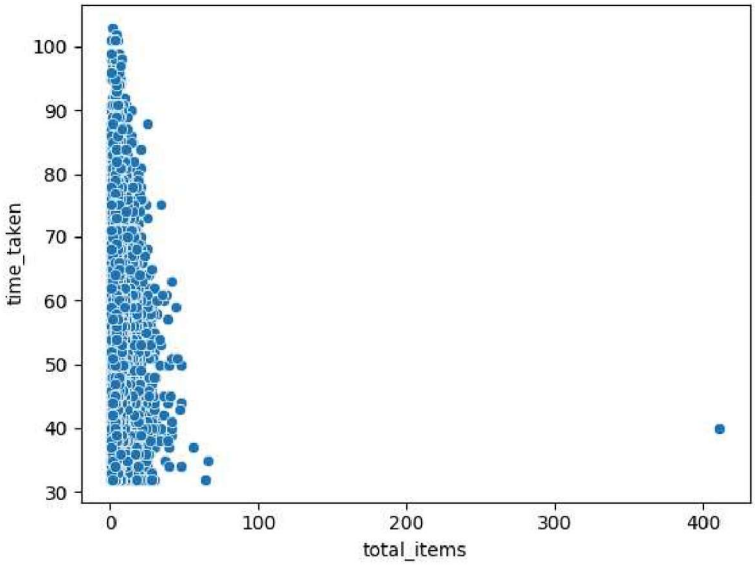
3.2 Relationships Between Features [3 marks]

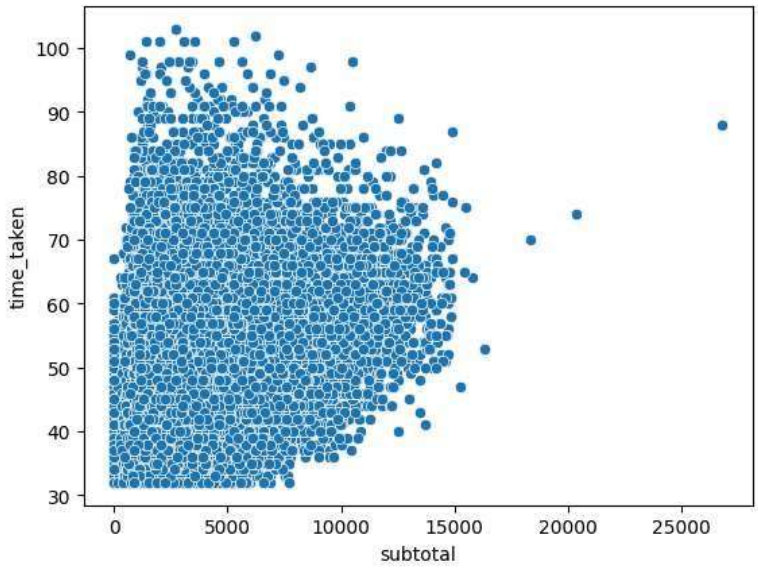
3.2.1 [3 marks]

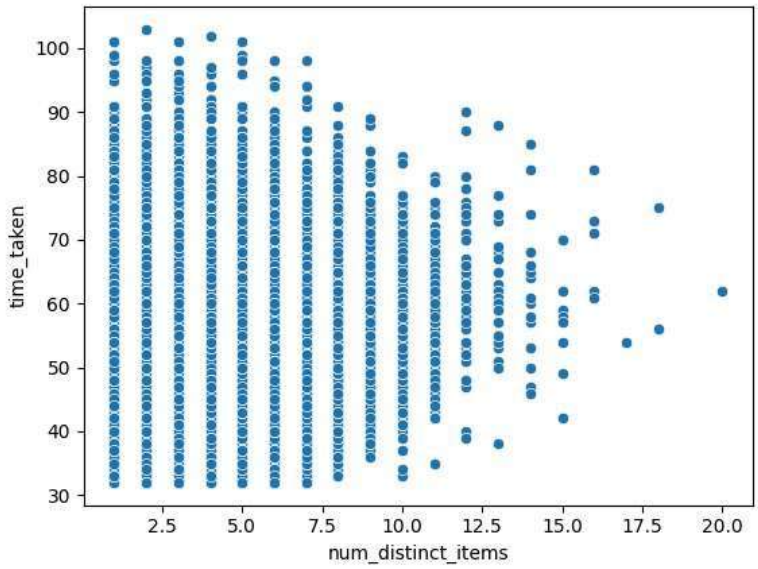
Scatter plots for important numerical and categorical features to observe how they relate to `time_taken`

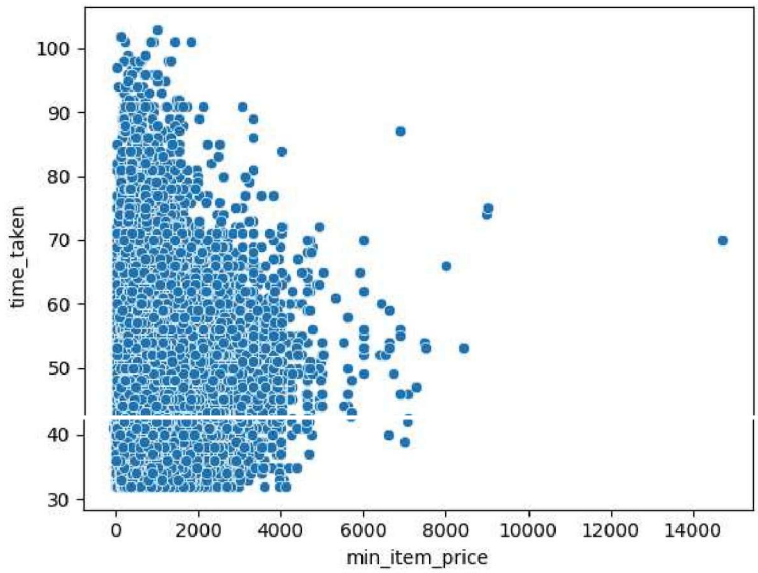

```
# Scatter plot to visualise the relationship between time_taken and other features
for i in X_train[numerical_columns]:
```

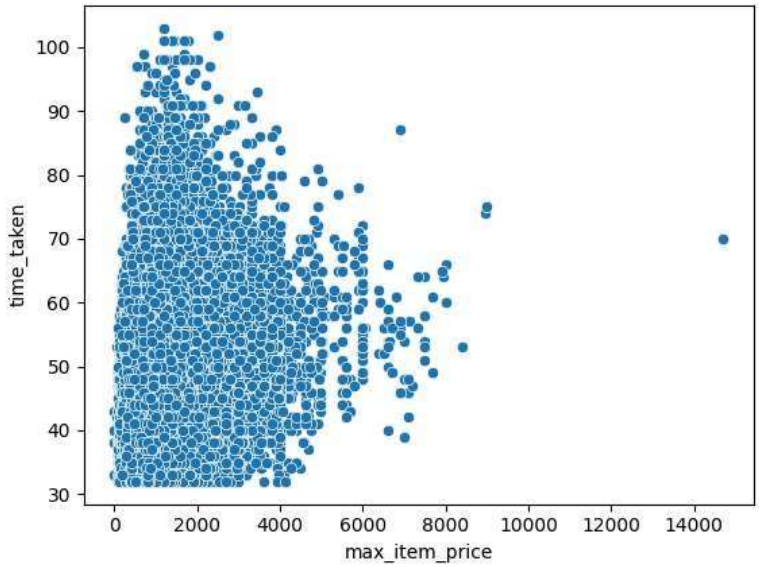

time_taken
market_id
time_taken
store_primary_category
time_taken
order_protocol

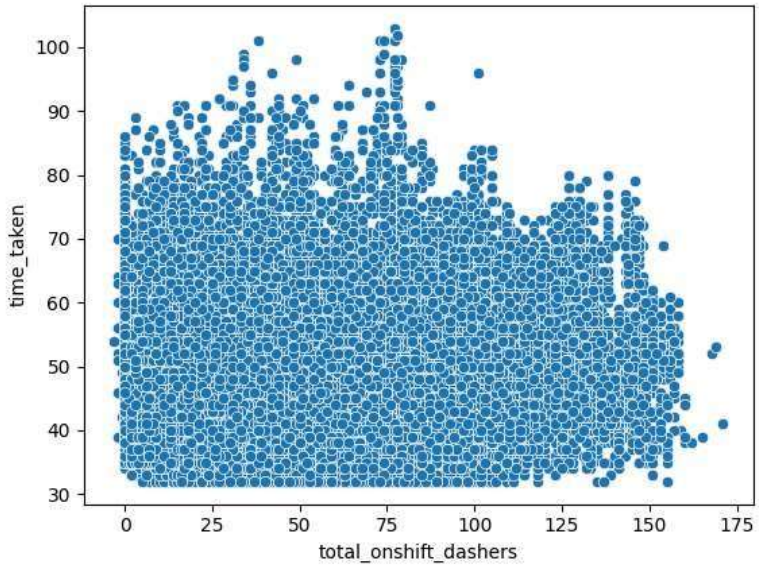

time_taken
total_items
100
90
80
70
60
50
40
30
0
100
200
300
400

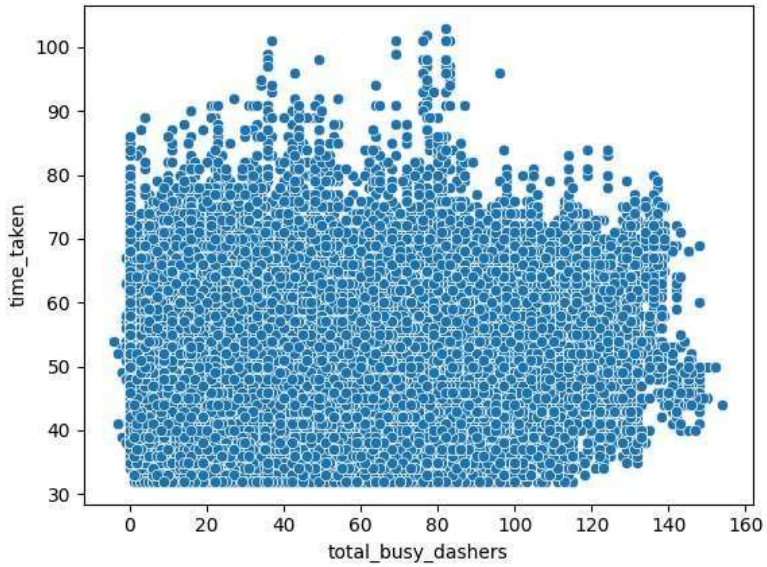

time_taken
subtotal
0
5000
10000
15000
20000
25000
30
40
50
60
70
80
90
100

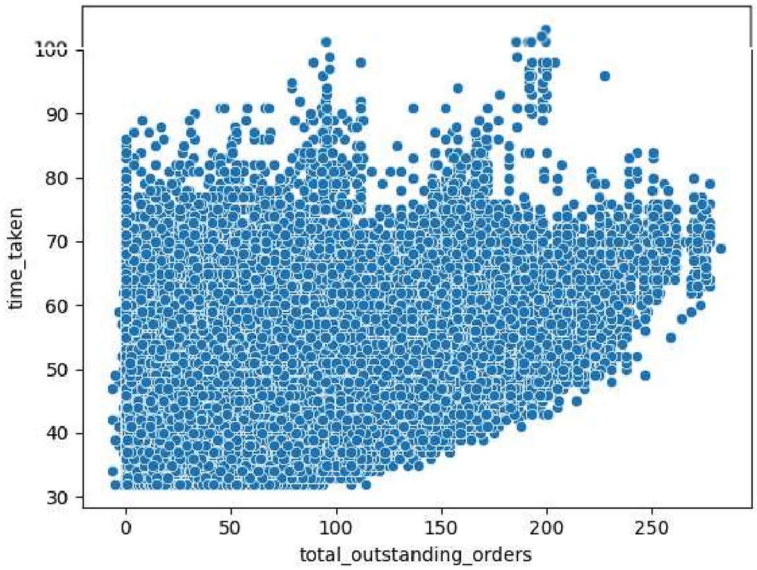

time_taken
num_distinct_items
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
30
40
50
60
70
80
90
100

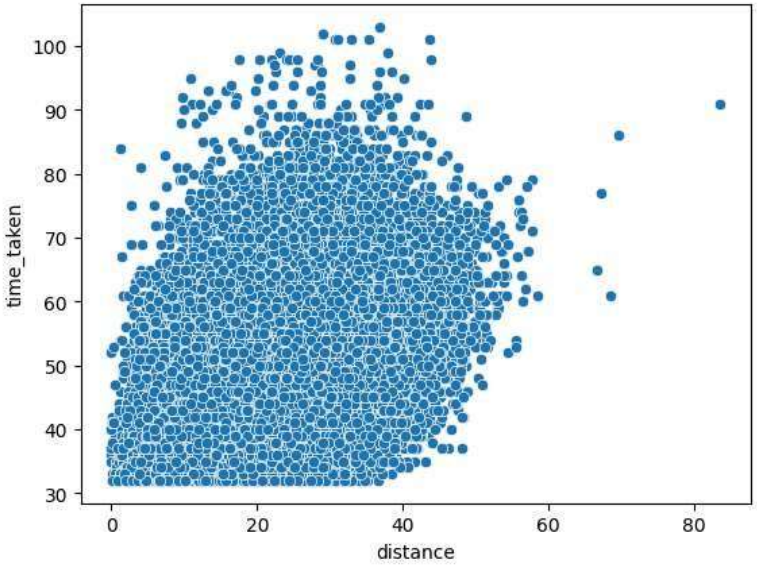

time_taken
min_item_price
0
2000
4000
6000
8000
10000
12000
14000
30
40
50
60
70
80
90
100


time_taken
max_item_price
100
90
80
70
60
50
40
30
0
2000
4000
6000
8000
10000
12000
14000

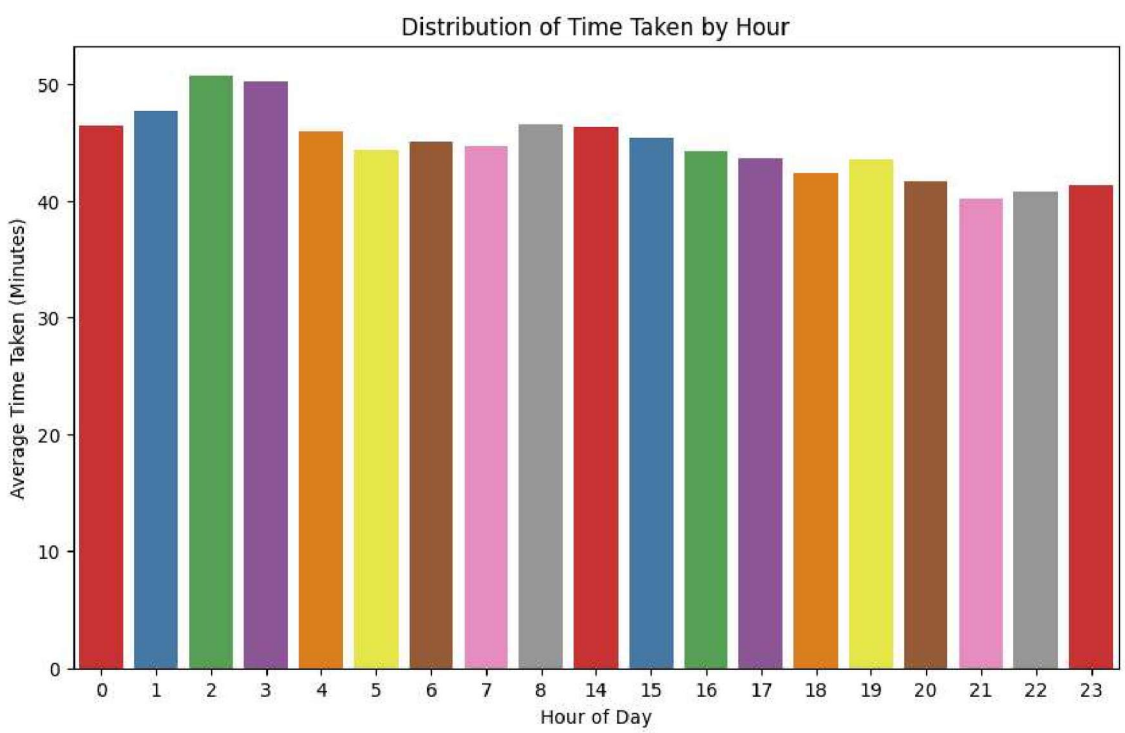

time_taken
total_onshift_dashers
100
90
80
70
60
50
40
30
0
25
50
75
100
125
150
175


time_taken
total_busy_dashers
100
90
80
70
60
50
40
30
0
20
40
60
80
100
120
140
160


time_taken
total_outstanding_orders
100
90
80
70
60
50
40
30
0
50
100
150
200
250


time_taken
distance
100
90
80
70
60
50
40
30
0
20
40
60
80

```
# Show the distribution of time_taken for different hours
time_taken_by_hour = portar.groupby('hour_of_day')['time_taken'].mean()
plt.figure(figsize=(10, 6))
sns.barplot(x=time_taken_by_hour.index, y=time_taken_by_hour.values, palette='Set1', legend=False)
plt.title('Distribution of Time Taken by Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Average Time Taken (Minutes)')
plt.show()
```



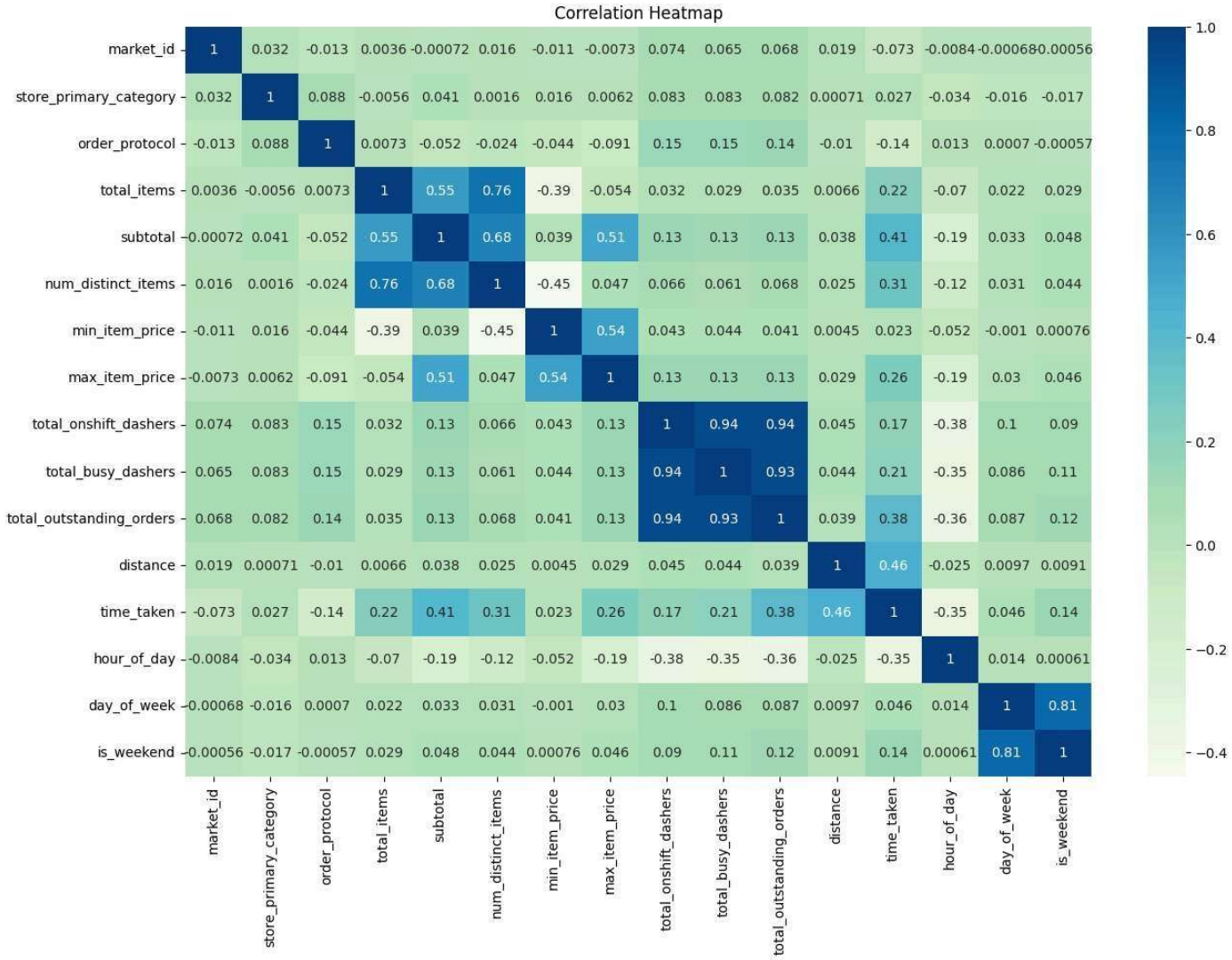
3.3 Correlation Analysis [5 marks]

Check correlations between numerical features to identify which variables are strongly related to `time_taken`

3.3.1 [3 marks]

Plot a heatmap to display correlations

```
# Plot the heatmap of the correlation matrix
plt.figure(figsize=(15, 10))
sns.heatmap(portar.corr(), annot=True, cmap="GnBu")
plt.title('Correlation Heatmap')
plt.show()
```


Correlation Heatmap
market_id 1 0.032 -0.013 0.0036 -0.00072 0.016 -0.011 -0.0073 0.074 0.065 0.068 0.019 -0.073 -0.0084 -0.00068 0.00056
store_primary_category 0.032 1 0.088 -0.0056 0.041 0.0016 0.016 0.0062 0.083 0.083 0.082 0.00071 0.027 -0.034 -0.016 -0.017
order_protocol -0.013 0.088 1 0.0073 -0.052 -0.024 -0.044 -0.091 0.15 0.15 0.14 -0.01 -0.14 0.013 0.0007 -0.00057
total_items 0.0036 -0.0056 0.0073 1 0.55 0.76 -0.39 -0.054 0.032 0.029 0.035 0.0066 0.22 -0.07 0.022 0.029
subtotal -0.00072 0.041 -0.052 0.55 1 0.68 0.039 0.51 0.13 0.13 0.13 0.038 0.41 -0.19 0.033 0.048
num_distinct_items 0.016 0.0016 -0.024 0.76 0.68 1 -0.45 0.047 0.066 0.061 0.068 0.025 0.31 -0.12 0.031 0.044
min_item_price -0.011 0.016 -0.044 -0.39 0.039 -0.45 1 0.54 0.043 0.044 0.041 0.0045 0.023 -0.052 -0.001 0.00076
max_item_price -0.0073 0.0062 -0.091 -0.054 0.51 0.047 0.54 1 0.13 0.13 0.13 0.029 0.26 -0.19 0.03 0.046
total_onshift_dashers 0.074 0.083 0.15 0.032 0.13 0.066 0.043 0.13 1 0.94 0.94 0.045 0.17 -0.38 0.1 0.09
total_busy_dashers 0.065 0.083 0.15 0.029 0.13 0.061 0.044 0.13 0.94 1 0.93 0.044 0.21 -0.35 0.086 0.11
total_outstanding_orders 0.068 0.082 0.14 0.035 0.13 0.068 0.041 0.13 0.94 0.93 1 0.039 0.38 -0.36 0.087 0.12
distance 0.019 0.00071 -0.01 0.0066 0.038 0.025 0.0045 0.029 0.045 0.044 0.039 1 0.46 -0.025 0.0097 0.0091
time_taken -0.073 0.027 -0.14 0.22 0.41 0.31 0.023 0.26 0.17 0.21 0.38 0.46 1 -0.35 0.046 0.14
hour_of_day -0.0084 -0.034 0.013 -0.07 -0.19 -0.12 -0.052 -0.19 -0.38 -0.35 -0.36 -0.025 -0.35 1 0.014 0.00061
day_of_week -0.00068 -0.016 0.0007 0.022 0.033 0.031 -0.001 0.03 0.1 0.086 0.087 0.0097 0.046 0.014 1 0.81
is_weekend -0.00056 -0.017 -0.00057 0.029 0.048 0.044 0.00076 0.046 0.09 0.11 0.12 0.0091 0.14 0.00061 0.81 1
market_id
store_primary_category
order_protocol
total_items
subtotal
num_distinct_items
min_item_price
max_item_price
total_onshift_dashers
total_busy_dashers
total_outstanding_orders
distance
time_taken
hour_of_day
day_of_week
is_weekend
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

```
X_test.head()
```

3.3.2 [2 marks]

Drop the columns with weak correlations with the target variable

```
# Drop 3-5 weakly correlated columns from training dataset
X_train = X_train.drop(['market_id', 'order_protocol'], axis=1)
X_test = X_test.drop(['market_id', 'order_protocol'], axis=1)
```

```
X_train.head()
```

	store_primary_category	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_dashers	total
94746	24	2	1790	2	795	995	10.0	
173338	72	1	845	1	795	795	134.0	
37592	55	1	1900	1	1200	1200	21.0	
42763	28	6	463	3	0	299	98.0	
27506	72	3	3500	3	600	1200	7.0	

	store_primary_category	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_dashers	total
139667	45	3	1295	2	150	995	59.0	
80077	4	2	2950	1	1225	1225	18.0	
41872	46	1	1395	1	1395	1395	28.0	
165269	24	2	2967	2	1097	1249	124.0	
151215	6	3	1250	2	375	400	39.0	

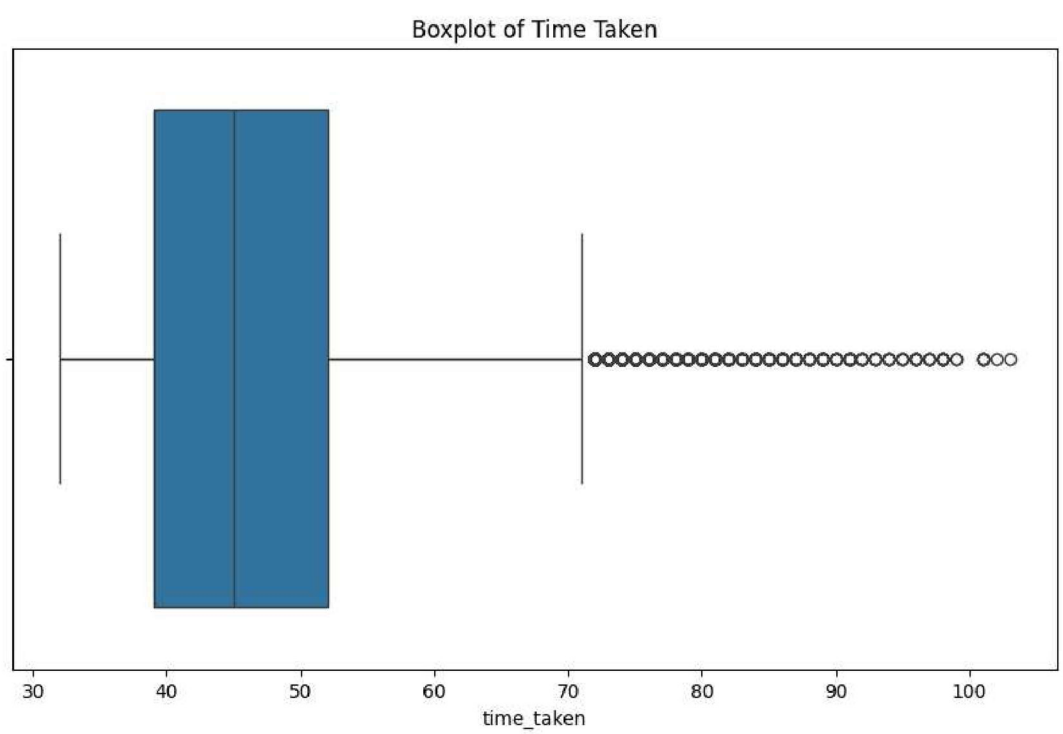
Next steps: Generate code with X_test | View recommended plots | New interactive sheet

3.4 Handling the Outliers [5 marks]

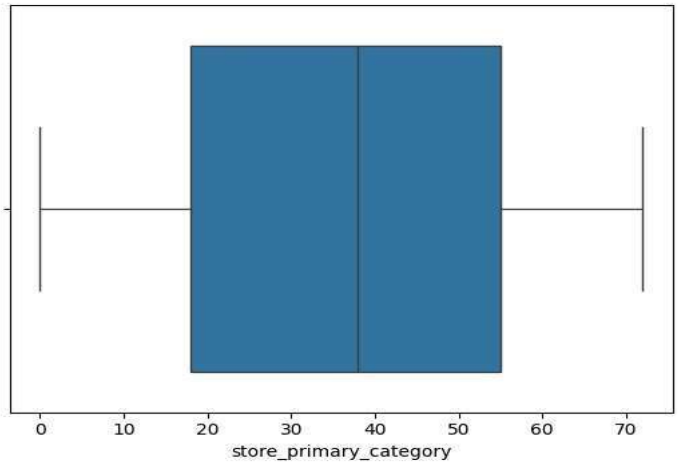
3.4.1 [2 marks]

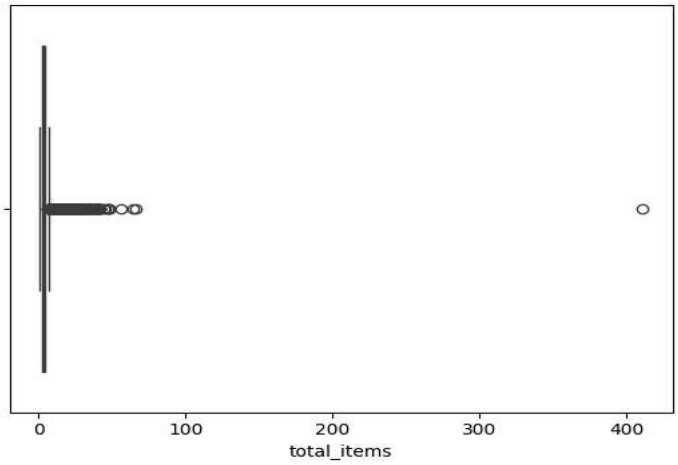
Visualise potential outliers for the target variable and other numerical features using boxplots

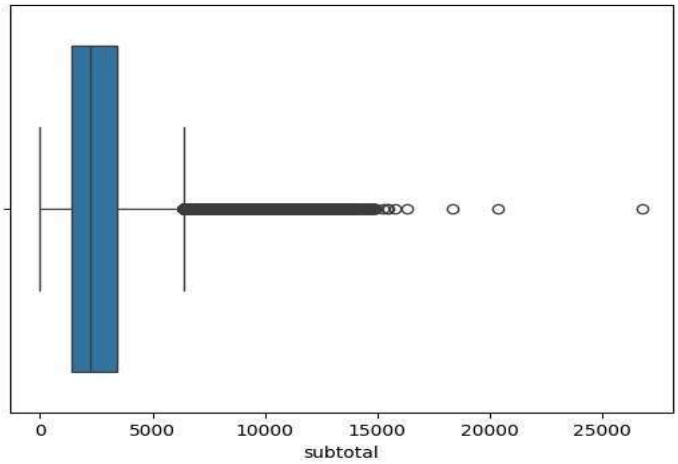
```
# Boxplot for time_taken
plt.figure(figsize=(10, 6))
sns.boxplot(x=y_train)
plt.title('Boxplot of Time Taken')
plt.show()
```

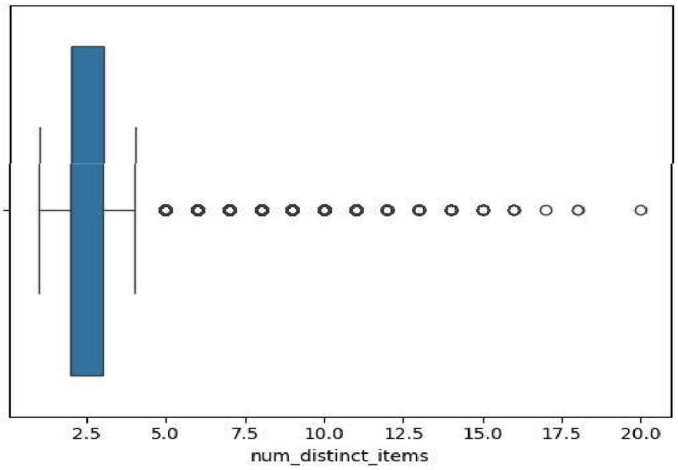


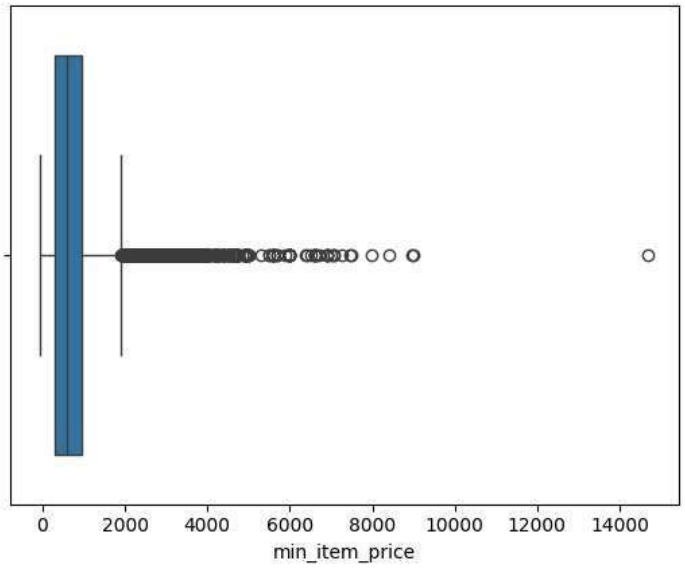

```
numerical_columns = X_train.select_dtypes(include=['int64', 'float64']).columns

for i in X_train[numerical_columns]:
    sns.boxplot(x=X_train[i])
    plt.show()
```

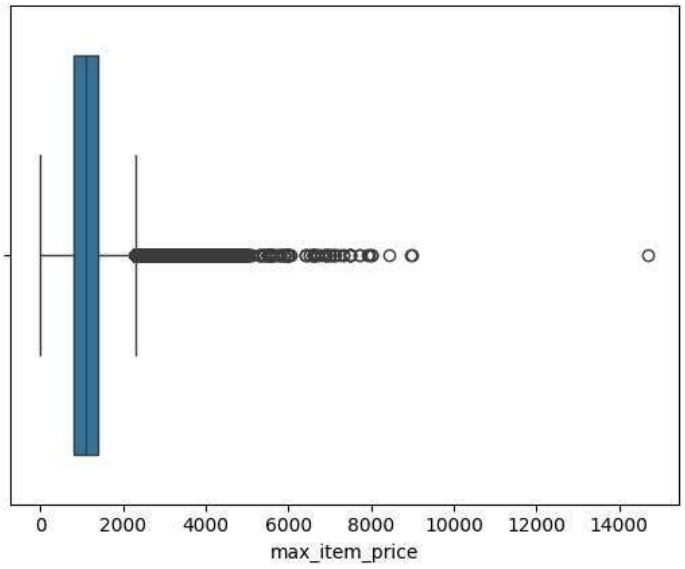

0
10
20
30
40
50
60
70
store_primary_category

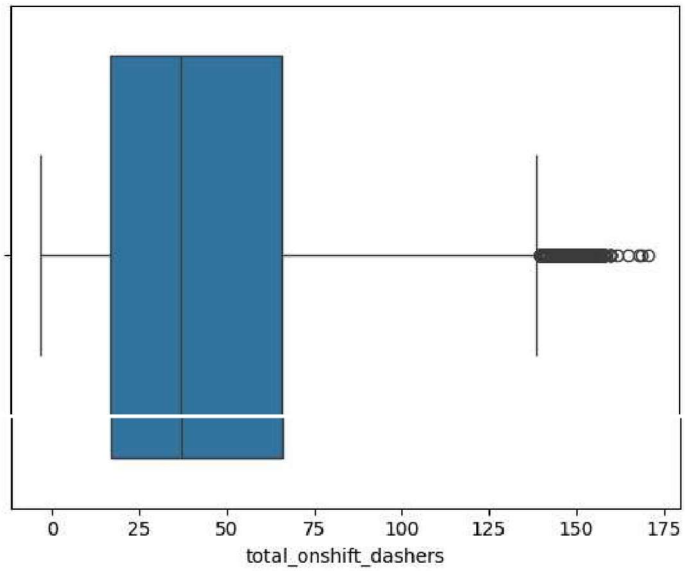

0
100
200
300
400
total_items

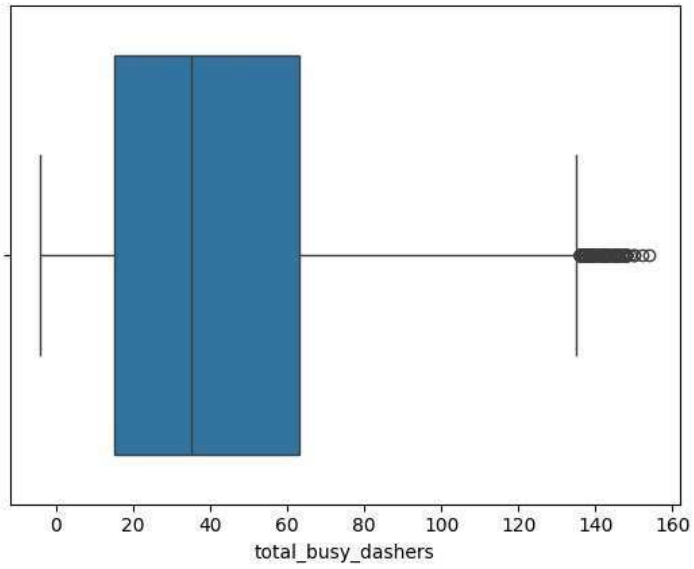

0
5000
10000
15000
20000
25000
subtotal

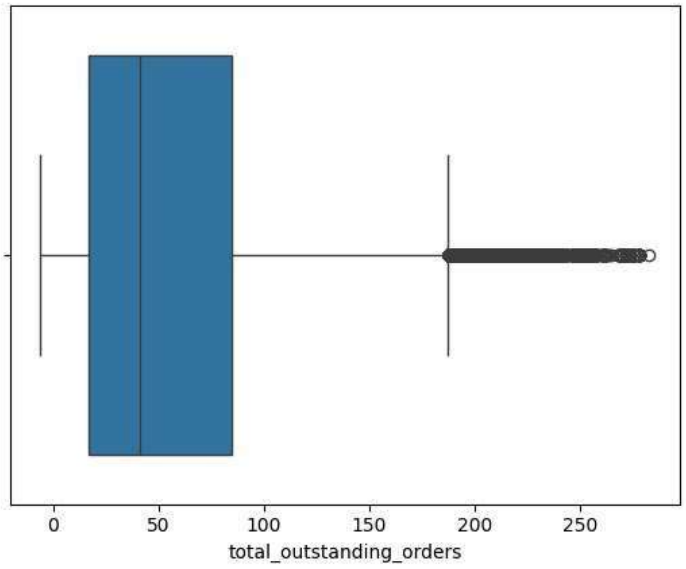

2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
num_distinct_items

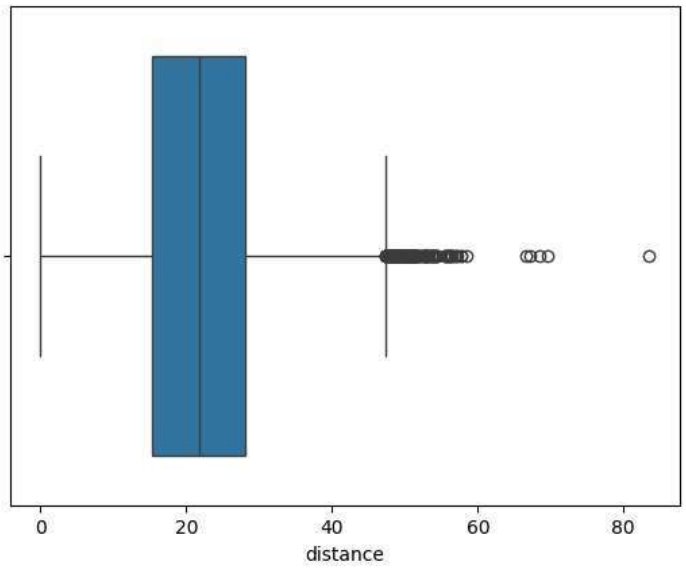

0
2000
4000
6000
8000
10000
12000
14000
min_item_price


0
2000
4000
6000
8000
10000
12000
14000
max_item_price


0
25
50
75
100
125
150
175
total_onshift_dashers


0
20
40
60
80
100
120
140
160
total_busy_dashers


0
50
100
150
200
250
total_outstanding_orders


0
20
40
60
80
distance

```
X_train.describe()
```

	store_primary_category	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_dashers
count	123043.000000	123043.000000	123043.000000	123043.000000	123043.000000	123043.000000	123043.000000
mean	35.940899	3.206082	2697.863625	2.674951	684.784506	1160.434645	44.981177
std	20.738317	2.745043	1830.338637	1.625552	520.731071	562.955073	34.568806
min	0.000000	1.000000	0.000000	1.000000	-52.000000	0.000000	-3.000000
25%	18.000000	2.000000	1417.000000	2.000000	299.000000	799.000000	17.000000
50%	38.000000	3.000000	2220.000000	2.000000	595.000000	1095.000000	37.000000
75%	55.000000	4.000000	3405.000000	3.000000	942.000000	1395.000000	66.000000
max	72.000000	411.000000	26800.000000	20.000000	14700.000000	14700.000000	171.000000

3.4.2 [3 marks]

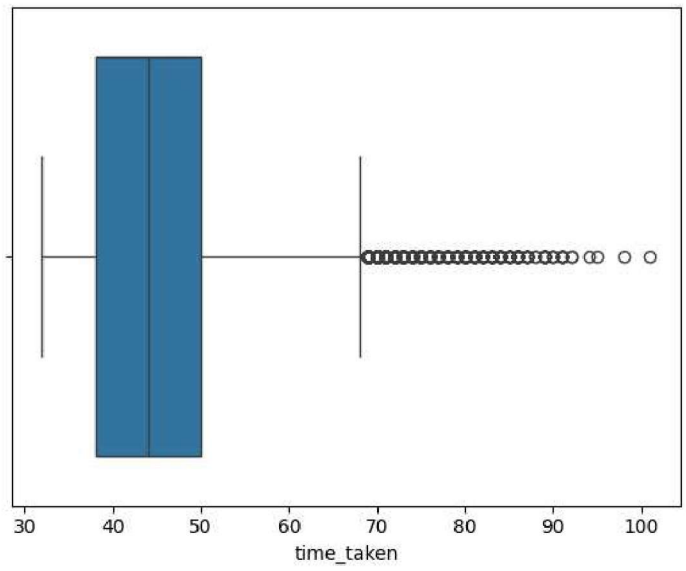
Handle outliers present in all columns

```
# Handle outliers
for i in X_train[numerical_columns]:
  Q1=X_train[i].quantile(0.25)
  Q3=X_train[i].quantile(0.75)
  IQR=Q3-Q1
  upper_limit=Q3+1.5*IQR
  lower_limit=Q1-1.5*IQR
  outlier_mask = (X_train[i] <= upper_limit) & (X_train[i] >= lower_limit)
  X_train = X_train[outlier_mask]
  y_train = y_train[outlier_mask]
```

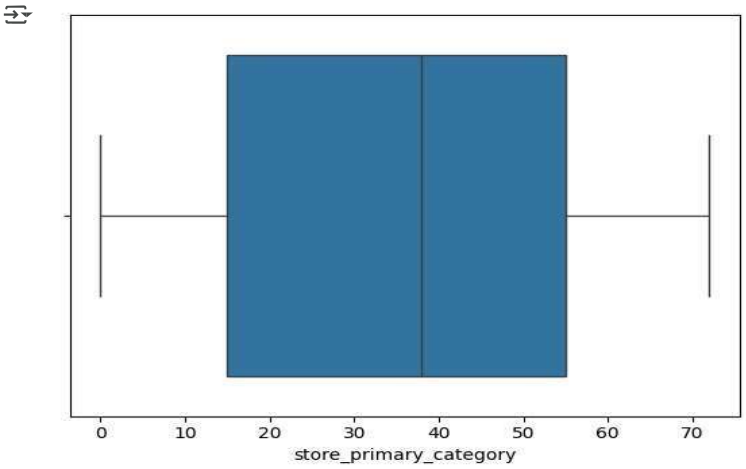
```
X_train.shape, y_train.shape
```

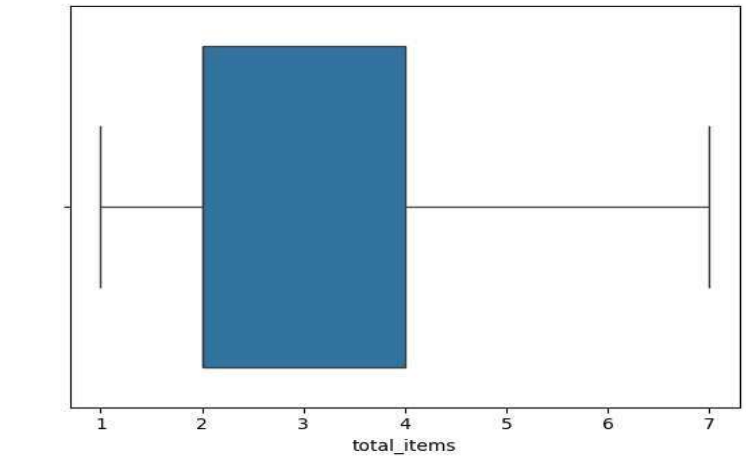
```
((102483, 13), (102483,))
```

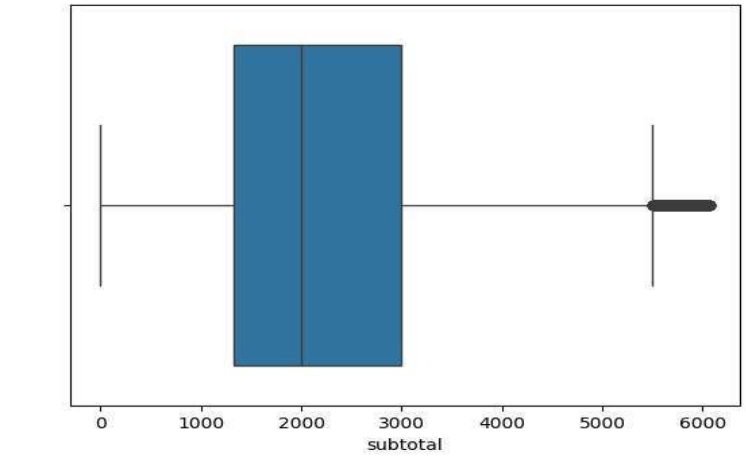
```
sns.boxplot(x=y_train)
plt.show()
```

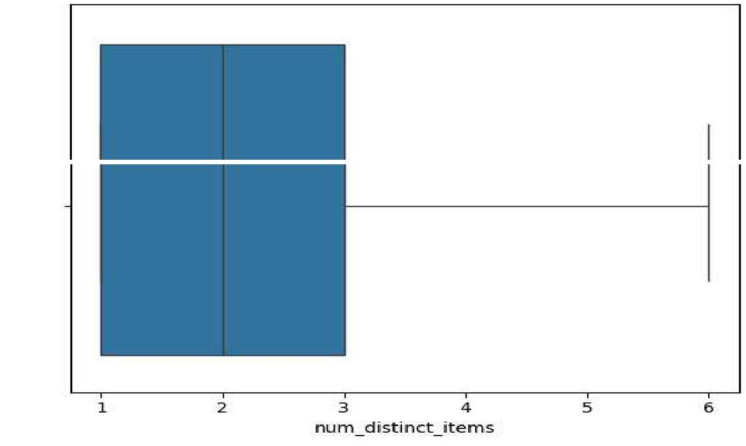


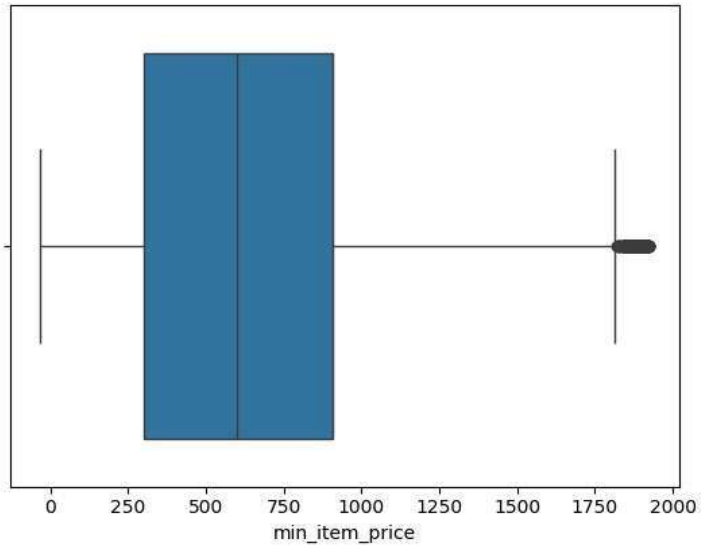

```
for i in X_train[numerical_columns]:
    sns.boxplot(x=X_train[i])
    plt.show()
```

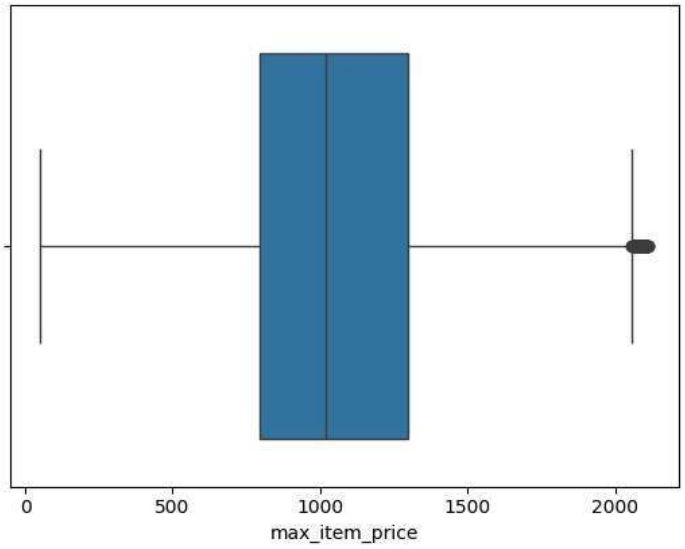

0
10
20
30
40
50
60
70
store_primary_category

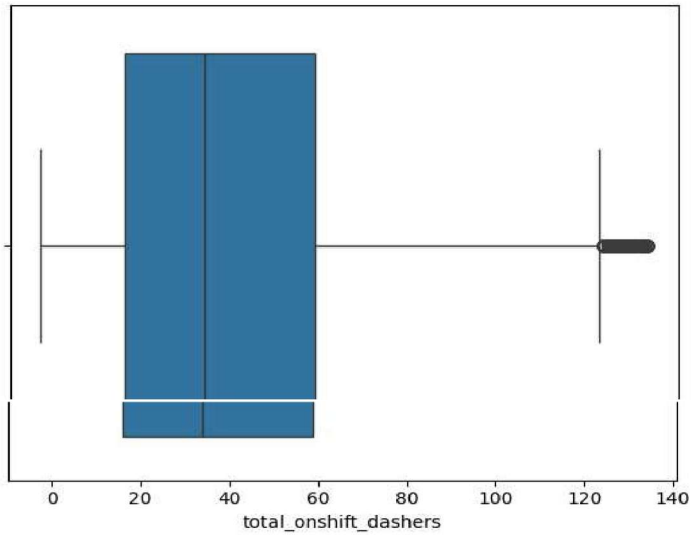

1
2
3
4
5
6
7
total_items

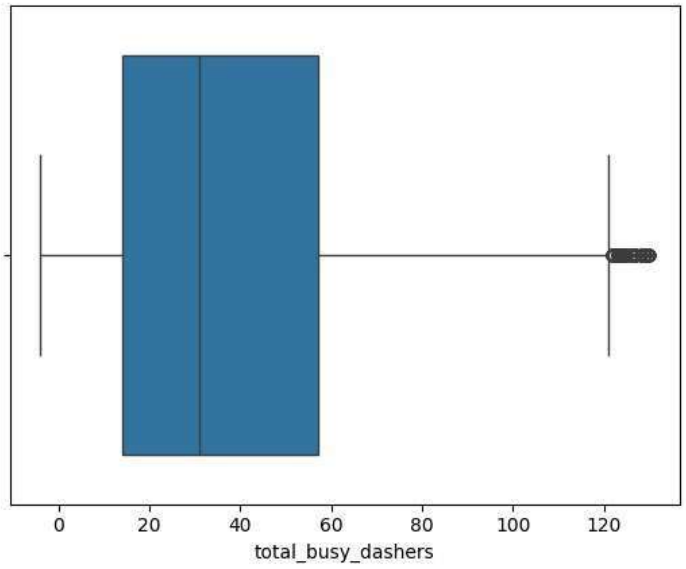

0
1000
2000
3000
4000
5000
6000
subtotal

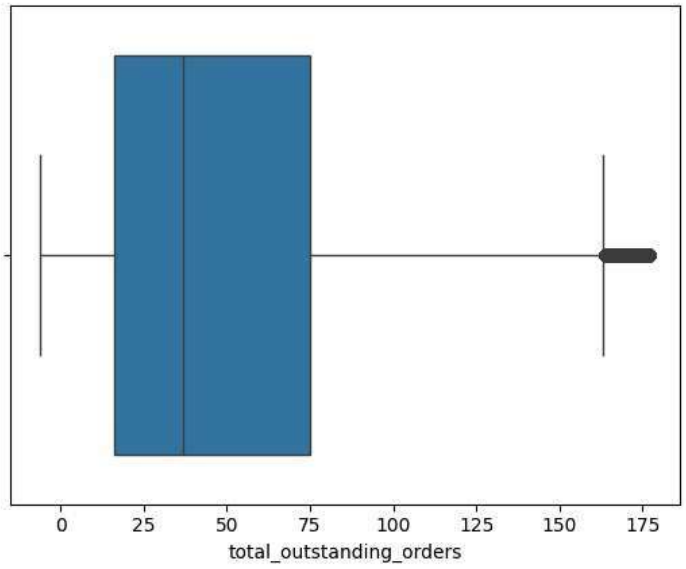

1
2
3
4
5
6
num_distinct_items

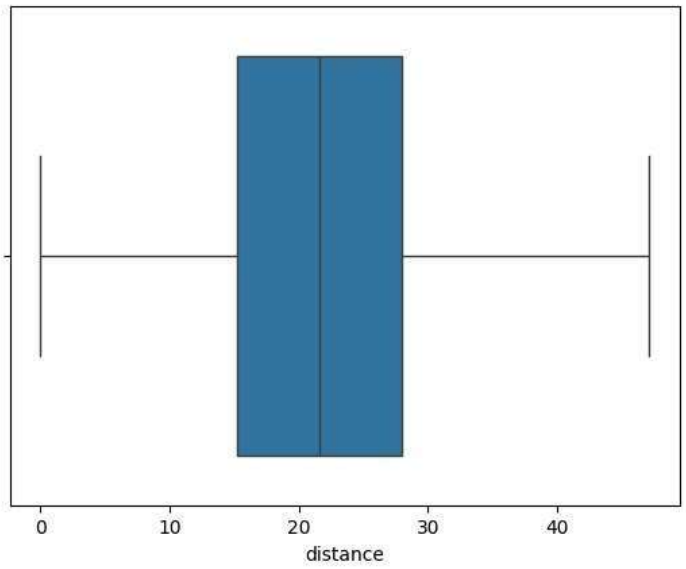

0
250
500
750
1000
1250
1500
1750
2000
min_item_price


0
500
1000
1500
2000
max_item_price


0
20
40
60
80
100
120
140
total_onshift_dashers


0
20
40
60
80
100
120
total_busy_dashers


0
25
50
75
100
125
150
175
total_outstanding_orders


0
10
20
30
40
distance

Double-click (or enter) to edit

```
X_train.store_primary_category.unique()
```

```
array([24, 72, 55, 28, 50, 38, 57, 39,  4, 25, 68, 58, 46, 47,  2, 18, 13,
       59, 20,  6, 15, 45, 34, 36, 12, 14, 66, 10, 61,  9, 65, 71, 42, 53,
       40, 52, 35,  7, 70, 54, 48, 37, 69, 29,  5, 49, 31, 26, 67, 23, 32,
       60, 16, 30, 44,  0, 27, 63, 17, 62, 51, 41, 33, 43, 11, 22, 56,  1,
        8, 19, 64,  3])
```

```
X_train.total_items.describe()
```

	total_items
count	102483.000000
mean	2.735381
std	1.459452
min	1.000000
25%	2.000000
50%	2.000000
75%	4.000000
max	7.000000

dtype: float64

```
X_train.subtotal.describe()
```

	subtotal
count	102483.000000
mean	2275.262063
std	1205.976699
min	0.000000
25%	1324.000000
50%	2000.000000
75%	2995.000000
max	6067.000000

dtype: float64

```
portar_list=X_train[X_train['subtotal']>4000].value_counts()
print (portar_list)
```

```
store_primary_category  total_items  subtotal  num_distinct_items  min_item_price  max_item_price  total_onshift_dashers  total_busy_da
72                      7            6065      5                   185             995             63.0                   60.0
0                       3            4197      3                   799             1599            27.0                   11.0
                                     4870      3                   1390            1890            11.0                   12.0
                        4            4415      4                   600             1895            78.0                   75.0
                                     4600      4                   700             1400            10.0                   12.0

2                       3            4265      2                   569             1848            28.0                   22.0
                                     4200      1                   1400            1400            12.0                   35.0
                                     4195      2                   1350            1495            38.0                   23.0
                                     4140      3                   1250            1395            68.0                   62.0
                                     4100      3                   900             1800            55.0                   31.0
Name: count, Length: 10607, dtype: int64
```

```
X_train.num_distinct_items.value_counts()
```

num_distinct_items	count
2	33514
1	27107
3	23153
4	11977
5	5032
6	1700

dtype: int64

```
X_train.min_item_price.value_counts()
```

min_item_price	count
795	2389
250	2014
150	1821
350	1757
500	1743
...	...
1344	1
1446	1
1514	1
1520	1
1597	1

1602 rows × 1 columns

dtype: int64

```
portar_list=X_train[X_train['min_item_price']<0]
print (portar_list.value_counts())
```

```
store_primary_category  total_items  subtotal  num_distinct_items  min_item_price  max_item_price  total_onshift_dashers  total_busy_da
38                      5            2585      3                   -7              1235            49.0                   47.0
55                      1            3099      1                   -31             52              22.0                   26.0
Name: count, dtype: int64
```

```
X_train['min_item_price']=X_train['min_item_price'].apply(lambda x: x if x >= 0 else (x*(-1)))
```

```
pd.DataFrame({
    'Column': X_train.columns.values,
    'Negative-Values': [X_train[col].min() if col in X_train.select_dtypes(include=[np.number]).columns else 0 for col in X_train.columns]
})
```

	Column	Negative-Values
0	store_primary_category	0.0
1	total_items	1.0
2	subtotal	0.0
3	num_distinct_items	1.0
4	min_item_price	0.0
5	max_item_price	52.0
6	total_onshift_dashers	-3.0
7	total_busy_dashers	-4.0
8	total_outstanding_orders	-6.0
9	distance	0.0
10	hour_of_day	0.0
11	day_of_week	0.0
12	is_weekend	0.0

```
X_train['min_item_price']=X_train['min_item_price'].apply(lambda x: x if x > 0 else 1)
```

```
portar_list=X_train[X_train['min_item_price']<=0]
print (portar_list.value_counts())
```

```
Series([], Name: count, dtype: int64)
```

```
X_train.describe()
```

	store_primary_category	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_dashers
count	102483.000000	102483.000000	102483.000000	102483.000000	102483.000000	102483.000000	102483.000000
mean	35.883981	2.735381	2275.262063	2.408809	653.365007	1058.596704	40.489115
std	20.919836	1.459452	1205.976699	1.223970	404.495281	378.192478	30.200106
min	0.000000	1.000000	0.000000	1.000000	1.000000	52.000000	-3.000000
25%	15.000000	2.000000	1324.000000	1.000000	300.000000	795.000000	16.000000
50%	38.000000	2.000000	2000.000000	2.000000	599.000000	1019.000000	34.000000
75%	55.000000	4.000000	2995.000000	3.000000	906.500000	1299.000000	59.000000
max	72.000000	7.000000	6067.000000	6.000000	1923.000000	2112.000000	134.000000

```
X_train[(X_train['min_item_price']==0) & (X_train['max_item_price']==0) & (X_train['total_items']!=0)].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['min_item_price']==0)].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['max_item_price']==0)].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['total_items']==0)].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['min_item_price']==0) & (X_train['max_item_price']==0)].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['min_item_price'] > X_train['max_item_price'])].value_counts()
```

store_primary_category	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_dashers	total_busy_da
72	3	2738	2	983	950	55.0	48.0
2	1	872	1	796	658	50.0	50.0
		1134	1	805	710	33.0	35.0
68	1	1197	1	1194	1187	93.0	90.0
		1145	1	971	832	51.0	63.0
...	...	...	...	...	...	...	...
4	1	1077	1	1180	1080	34.0	42.0
		1020	1	1285	1227	12.0	12.0
2	1	2295	1	1712	1707	39.0	43.0
		1560	1	1577	1498	49.0	50.0
		1369	1	672	671	68.0	59.0

419 rows × 1 columns

dtype: int64

```
X_train[(X_train['min_item_price'] > X_train['max_item_price'])].value_counts().sum()
```

```
np.int64(419)
```

```
condition = X_train['min_item_price'] > X_train['max_item_price']
X_train.loc[condition, ['min_item_price', 'max_item_price']] = X_train.loc[condition, ['max_item_price', 'min_item_price']].values
```

```
X_train[(X_train['min_item_price'] > X_train['max_item_price'])].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['min_item_price'] ==0) & (X_train['max_item_price']==0)].value_counts().sum()
```

```
np.int64(0)
```

```
pd.DataFrame({
    'Column': X_train.columns.values,
    'Negative-Values': [X_train[col].min() if col in X_train.select_dtypes(include=[np.number]).columns else 0 for col in X_train.columns]
})
```

	Column	Negative-Values
0	store_primary_category	0.0
1	total_items	1.0
2	subtotal	0.0
3	num_distinct_items	1.0
4	min_item_price	1.0
5	max_item_price	52.0
6	total_onshift_dashers	-3.0
7	total_busy_dashers	-4.0
8	total_outstanding_orders	-6.0
9	distance	0.0
10	hour_of_day	0.0
11	day_of_week	0.0
12	is_weekend	0.0

```
#total_onshift_dashers, total_busy_dashers, total_outstanding_orders,
X_train[(X_train['total_onshift_dashers'] < 0) & (X_train['total_busy_dashers']<0) & (X_train['total_outstanding_orders']<0) ].value_counts
```

```
np.int64(0)
```

```
X_train[(X_train['total_onshift_dashers'] < 0)].value_counts().sum()
```

```
np.int64(12)
```

```
X_train[(X_train['total_onshift_dashers'] < 0)].value_counts()
```

store_primary_category	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_dashers	total_busy_da
4	2	2831	2	635	681	-1.0	13.0
		3538	2	1238	1606	-2.0	2.0
18	1	986	1	827	892	-2.0	5.0
20	3	2776	3	358	1227	-2.0	6.0
28	2	1327	1	615	680	-1.0	6.0
35	1	1202	1	987	1092	-1.0	9.0
39	2	720	1	495	515	-2.0	1.0
		1862	2	488	1162	-2.0	3.0
50	1	1524	1	897	920	-2.0	3.0
57	2	1117	2	339	956	-3.0	0.0
59	3	2238	3	740	768	-1.0	5.0
	4	1874	3	110	1378	-2.0	-1.0

dtype: int64

```
X_train['total_onshift_dashers']=X_train['total_onshift_dashers'].apply(lambda x: x if x >= 0 else (x*(-1)))
X_train[(X_train['total_onshift_dashers'] < 0)].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['total_busy_dashers']<0)].value_counts().sum()
```

```
np.int64(15)
```

```
X_train['total_busy_dashers']=X_train['total_busy_dashers'].apply(lambda x: x if x >= 0 else (x*(-1)))
X_train[(X_train['total_busy_dashers']<0)].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['total_outstanding_orders']<0) ].value_counts().sum()
```

```
np.int64(28)
```

```
X_train['total_outstanding_orders']=X_train['total_outstanding_orders'].apply(lambda x: x if x >= 0 else (x*(-1)))
X_train[(X_train['total_outstanding_orders']<0) ].value_counts().sum()
```

```
np.int64(0)
```

```
X_train[(X_train['total_onshift_dashers'] < 0) & (X_train['total_busy_dashers']<0)].value_counts().sum()
```

```
np.int64(0)
```

```
pd.DataFrame({
    'Column': X_train.columns.values,
    'Negative-Values': [X_train[col].min() if col in X_train.select_dtypes(include=[np.number]).columns else 0 for col in X_train.columns]
})
```

	Column	Negative-Values
0	store_primary_category	0.0
1	total_items	1.0
2	subtotal	0.0
3	num_distinct_items	1.0
4	min_item_price	1.0
5	max_item_price	52.0
6	total_onshift_dashers	0.0
7	total_busy_dashers	0.0
8	total_outstanding_orders	0.0
9	distance	0.0
10	hour_of_day	0.0
11	day_of_week	0.0
12	is_weekend	0.0

4. Exploratory Data Analysis on Validation Data [optional]

Optionally, perform EDA on test data to see if the distribution match with the training data

```
# Define numerical and categorical columns for easy EDA and data manipulation
numerical_columns = X_test.select_dtypes(include=['int64', 'float64']).columns
categorical_columns = X_test.select_dtypes(include=['object']).columns
print("Numerical columns:", numerical_columns)
print("Categorical columns:", categorical_columns)
```

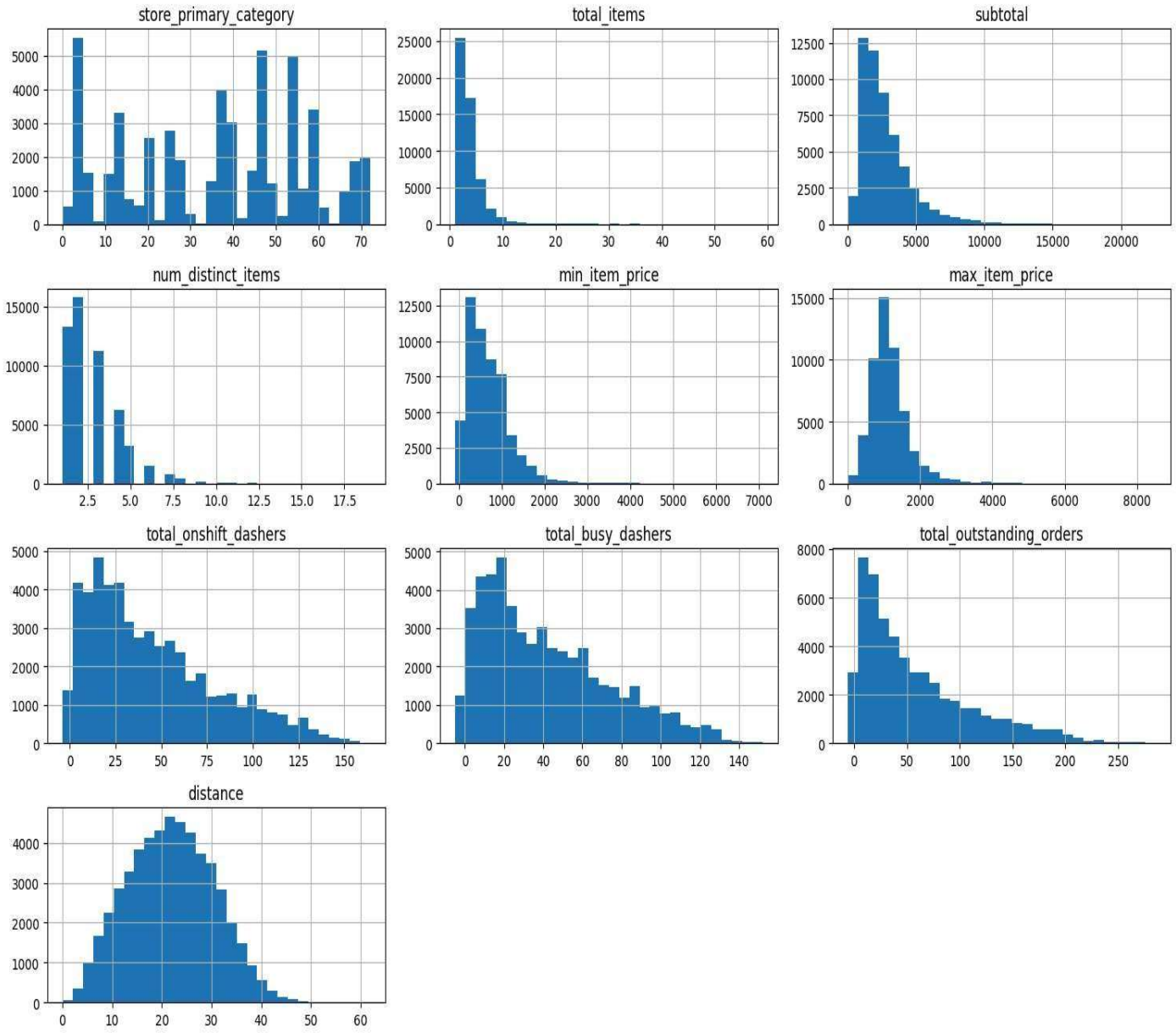
```
Numerical columns: Index(['store_primary_category', 'total_items', 'subtotal',
       'num_distinct_items', 'min_item_price', 'max_item_price',
       'total_onshift_dashers', 'total_busy_dashers',
       'total_outstanding_orders', 'distance'],
      dtype='object')
Categorical columns: Index([], dtype='object')
```

4.1 Feature Distributions

4.1.1

Plot distributions for numerical columns in the validation set to understand their spread and any skewness

```
# Plot distributions for all numerical columns
X_test[numerical_columns].hist(bins=30, figsize=(15, 10))
plt.tight_layout()
plt.show()
```


store_primary_category
total_items
subtotal
num_distinct_items
min_item_price
max_item_price
total_onshift_dashers
total_busy_dashers
total_outstanding_orders
distance

4.1.2

Check the distribution of categorical features

```
# Distribution of categorical columns
X_test[categorical_columns].nunique()
x_test_cat = X_test[categorical_columns]
x_test_cat.head()
```

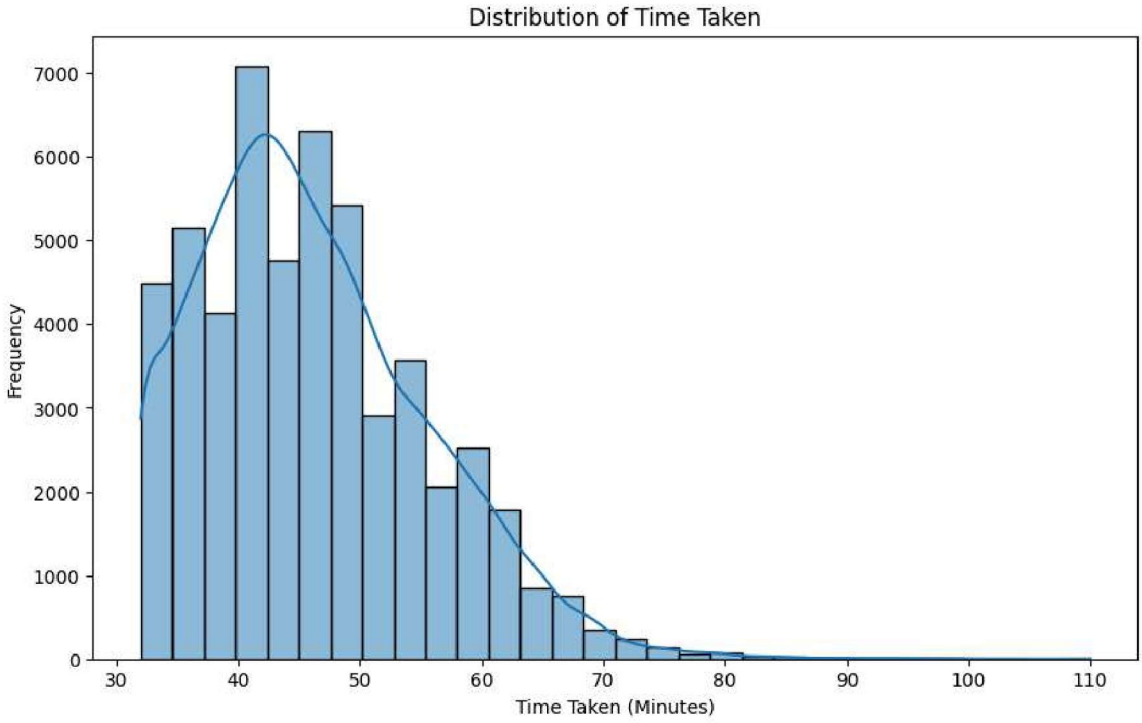
139667
80077
41872
165269
151215

4.1.3

Visualise the distribution of the target variable to understand its spread and any skewness

```
# Distribution of time_taken
plt.figure(figsize=(10, 6))
sns.histplot(y_test, bins=30, kde=True)
plt.title('Distribution of Time Taken')
plt.xlabel('Time Taken (Minutes)')
plt.ylabel('Frequency')
```

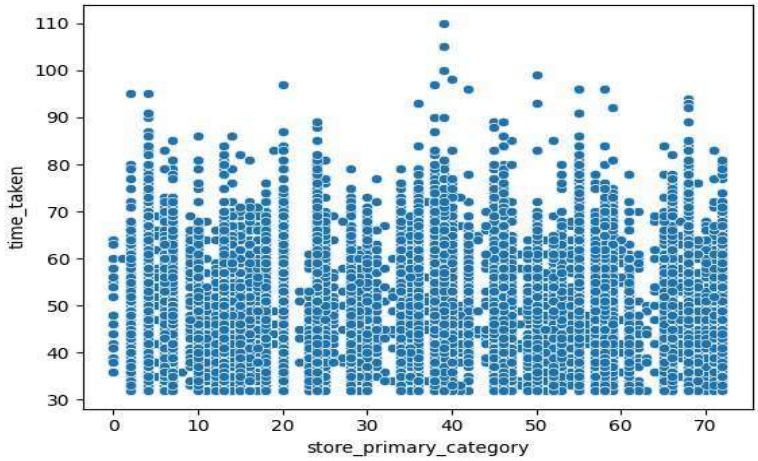
```
Text(0, 0.5, 'Frequency')
```

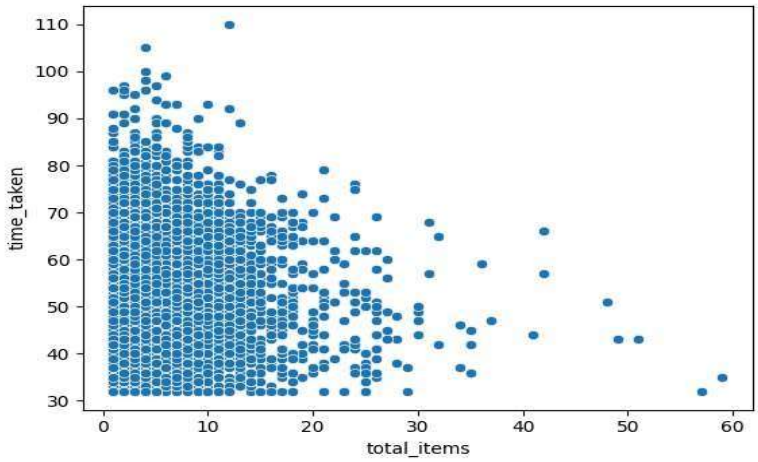


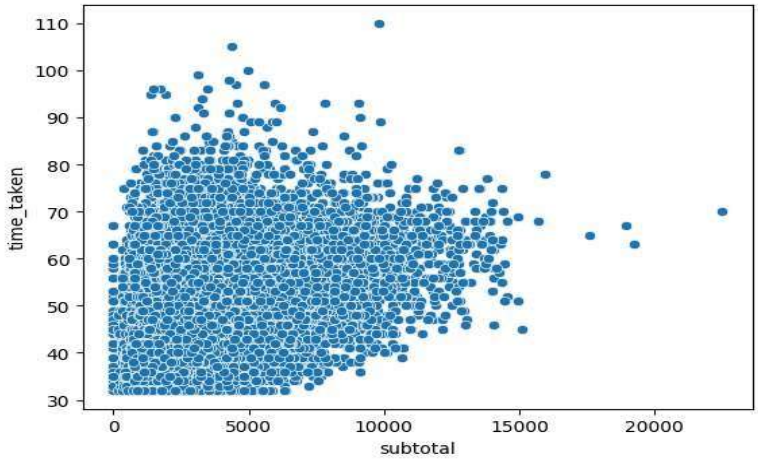
4.2 Relationships Between Features

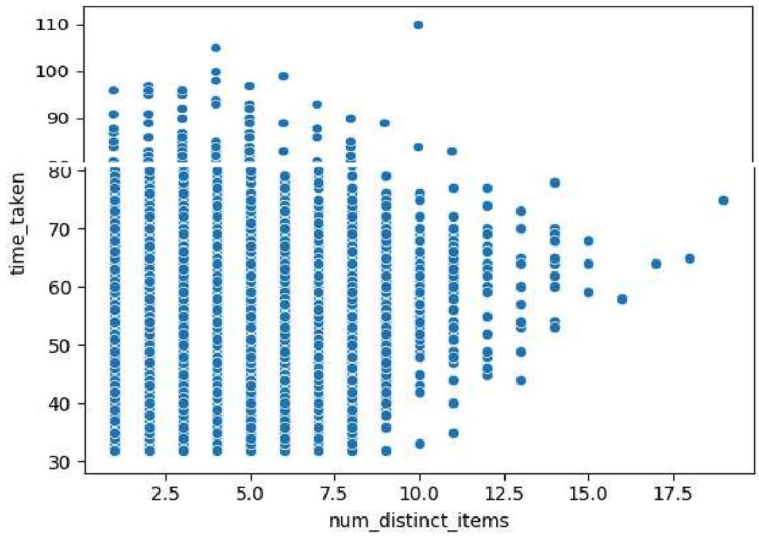
Scatter plots for numerical features to observe how they relate to each other, especially to `time_taken`

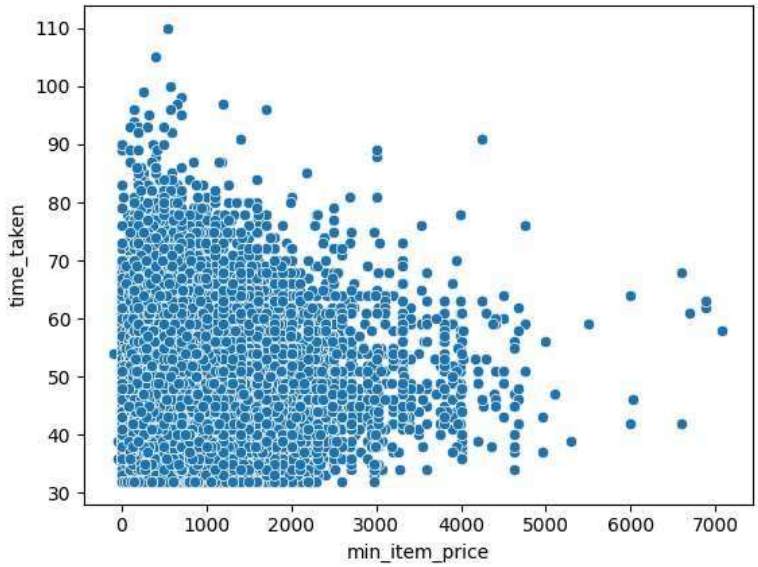
```
# Scatter plot to visualise the relationship between time_taken and other features
for i in X_test[numerical_columns]:
    sns.scatterplot(x=X_test[i], y=y_test)
    plt.show()
```

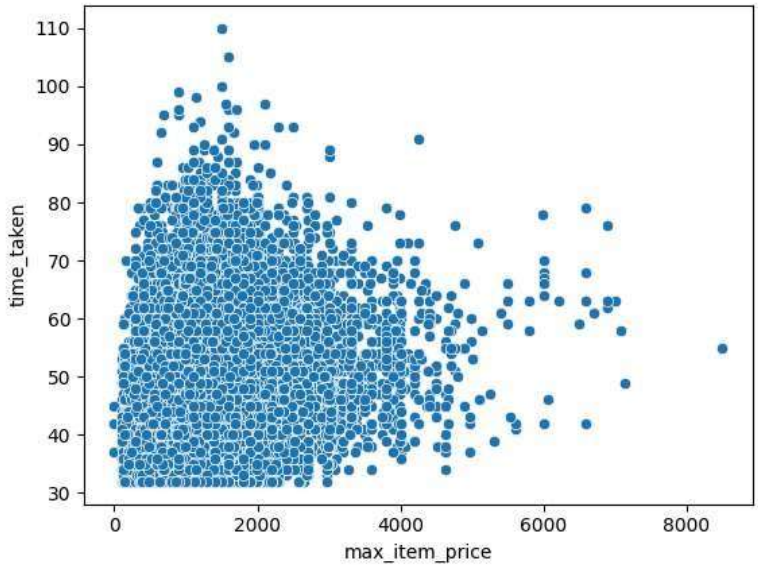

time_taken
store_primary_category

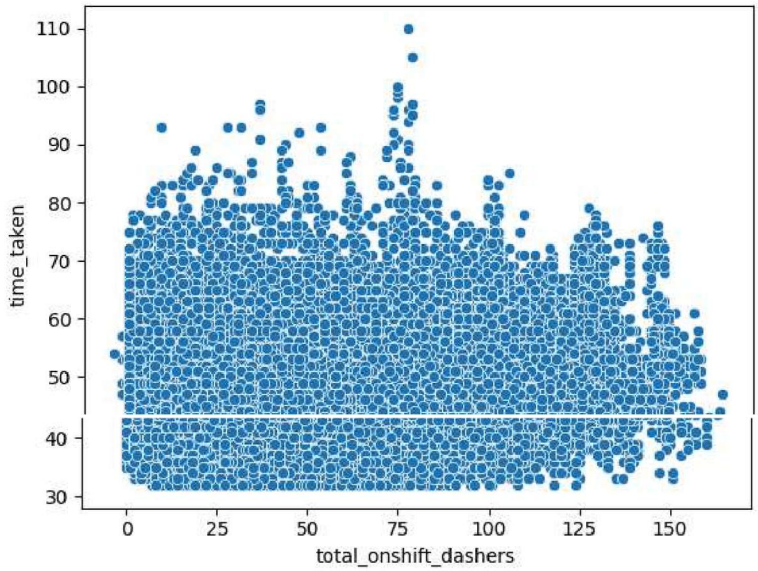

time_taken
total_items

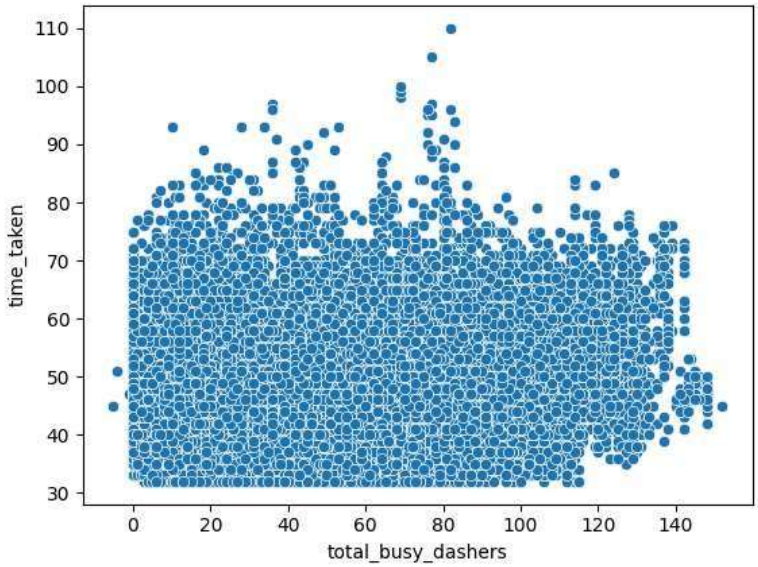

time_taken
subtotal

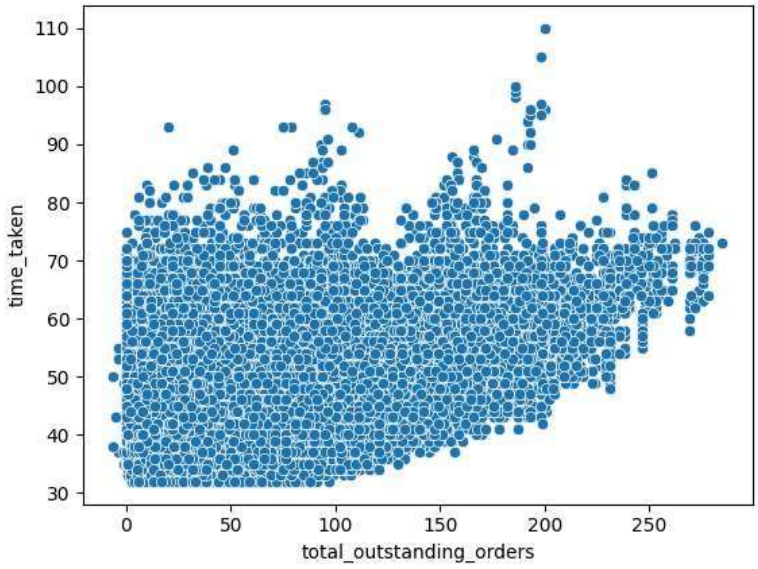

time_taken
num_distinct_items

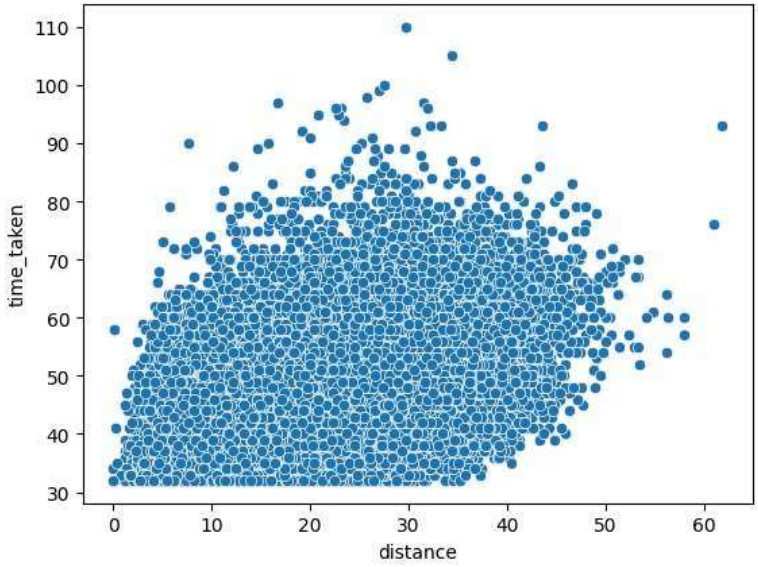

time_taken
min_item_price
110
100
90
80
70
60
50
40
30
0
1000
2000
3000
4000
5000
6000
7000


time_taken
max_item_price
110
100
90
80
70
60
50
40
30
0
2000
4000
6000
8000


time_taken
total_onshift_dashers
110
100
90
80
70
60
50
40
30
0
25
50
75
100
125
150

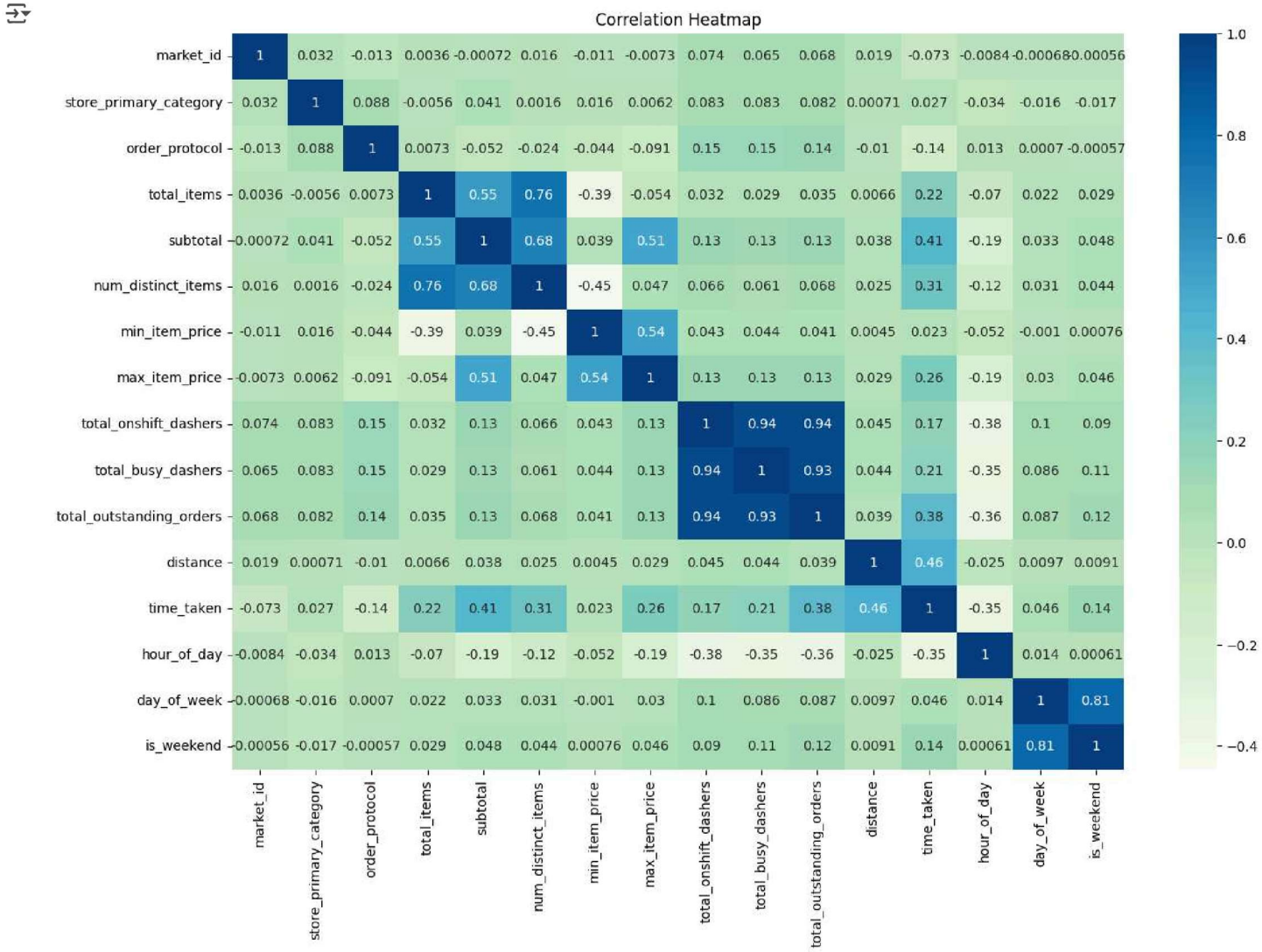

time_taken
total_busy_dashers
110
100
90
80
70
60
50
40
30
0
20
40
60
80
100
120
140


time_taken
total_outstanding_orders
110
100
90
80
70
60
50
40
30
0
50
100
150
200
250


time_taken
distance
110
100
90
80
70
60
50
40
30
0
10
20
30
40
50
60

4.3 Drop the columns with weak correlations with the target variable

```
# Drop the weakly correlated columns from training dataset
plt.figure(figsize=(15, 10))
sns.heatmap(portar.corr(), annot=True, cmap="GnBu")
plt.title('Correlation Heatmap')
plt.show()
```



5. Model Building [15 marks]

Import Necessary Libraries

```
# Import libraries
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
```

5.1 Feature Scaling [3 marks]

```
# Apply scaling to the numerical columns
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


X_train_df=pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_df=pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

Note that linear regression is agnostic to feature scaling. However, with feature scaling, we get the coefficients to be somewhat on the same scale so that it becomes easier to compare them.

5.2 Build a linear regression model [5 marks]

You can choose from the libraries *statsmodels* and *scikit-learn* to build the model.

```
# Create/Initialise the model
lm = LinearRegression()


# Train the model using the training data
rfe = RFE(estimator=lm, n_features_to_select=8)
rfe = rfe.fit(X_train_df, y_train)
```

```
feature=rfe.support_
features=X_train_df.columns[feature]
print(features)
```

```
Index(['subtotal', 'num_distinct_items', 'total_onshift_dashers',
       'total_busy_dashers', 'total_outstanding_orders', 'distance',
       'hour_of_day', 'is_weekend'],
      dtype='object')
```

```
X_train=X_train_df[features]
X_test=X_test_df[features]
```

```
X_train.shape, X_test.shape
```

```
((102483, 8), (52734, 8))
```

```
# Make predictions
lm.fit(X_train, y_train)
y_pred = lm.predict(X_test)
```

```
# Find results for evaluation metrics
re=lm.score(X_test, y_test)
print(re)
```

```
0.8586502641503106
```

Note that we have 12 (depending on how you select features) training features. However, not all of them would be useful. Let's say we want to take the most relevant 8 features.

We will use Recursive Feature Elimination (RFE) here.

For this, you can look at the coefficients / p-values of features from the model summary and perform feature elimination, or you can use the RFE module provided with *scikit-learn*.

5.3 Build the model and fit RFE to select the most important features [7 marks]

For RFE, we will start with all features and use the RFE method to recursively reduce the number of features one-by-one.

After analysing the results of these iterations, we select the one that has a good balance between performance and number of features.

```
# Loop through the number of features and test the model
from sklearn.feature_selection import RFE
for i in range(1, 13):
    selector = RFE(lm, n_features_to_select=i, step=1)
    selector = selector.fit(X_train, y_train)

    print(i, selector.support_)
    print(i, selector.ranking_)
    print(i, selector.score(X_test, y_test))

    X_train_rfe = selector.transform(X_train)
    X_test_rfe = selector.transform(X_test)
    lm.fit(X_train_rfe, y_train)
    print(i, lm.score(X_test_rfe, y_test))
```

```
1 [False False False False  True False False False]
1 [5 8 2 3 1 4 6 7]
1 0.13508783602199048
1 0.13508783602199048
2 [False False  True False  True False False False]
2 [4 7 1 2 1 3 5 6]
2 0.42873722912730605
2 0.42873722912730605
3 [False False  True  True  True False False False]
3 [3 6 1 1 1 2 4 5]
3 0.45079586674544503
3 0.45079586674544503
4 [False False  True  True  True  True False False]
4 [2 5 1 1 1 1 3 4]
4 0.6653232264448066
4 0.6653232264448066
5 [ True False  True  True  True  True False False]
5 [1 4 1 1 1 1 2 3]
5 0.8011120669606548
5 0.8011120669606548
6 [ True False  True  True  True  True  True False]
6 [1 3 1 1 1 1 1 2]
6 0.8529100280345906
6 0.8529100280345906
7 [ True False  True  True  True  True  True  True]
7 [1 2 1 1 1 1 1 1]
7 0.8571347297162277
7 0.8571347297162277
8 [ True  True  True  True  True  True  True  True]
8 [1 1 1 1 1 1 1 1]
8 0.8586502641503106
8 0.8586502641503106
9 [ True  True  True  True  True  True  True  True]
9 [1 1 1 1 1 1 1 1]
9 0.8586502641503106
9 0.8586502641503106
10 [ True  True  True  True  True  True  True  True]
10 [1 1 1 1 1 1 1 1]
10 0.8586502641503106
10 0.8586502641503106
11 [ True  True  True  True  True  True  True  True]
11 [1 1 1 1 1 1 1 1]
11 0.8586502641503106
11 0.8586502641503106
12 [ True  True  True  True  True  True  True  True]
12 [1 1 1 1 1 1 1 1]
12 0.8586502641503106
12 0.8586502641503106
```

```
# Build the final model with selected number of features
from sklearn.feature_selection import RFE
selector = RFE(lm, n_features_to_select=8, step=1)
selector = selector.fit(X_train, y_train)
lr_df=pd.DataFrame(selector.feature_names_in_, columns=['Feature Names'])
lr_df['co_efficient']=selector.estimator_.coef_
lr_df
```

	Feature Names	co_efficient
0	subtotal	1.871827
1	num_distinct_items	0.451529
2	total_onshift_dashers	-11.957092
3	total_busy_dashers	-4.092703
4	total_outstanding_orders	16.076426
5	distance	4.169394
6	hour_of_day	-2.212709
7	is_weekend	0.641108

Next steps: Generate code with lr_df | View recommended plots | New interactive sheet

```
print (selector.support_)
print (selector.ranking_)
print (selector.score(X_train, y_train))
```

```
[ True  True  True  True  True  True  True  True]
[1 1 1 1 1 1 1 1]
0.8455723007686833
```

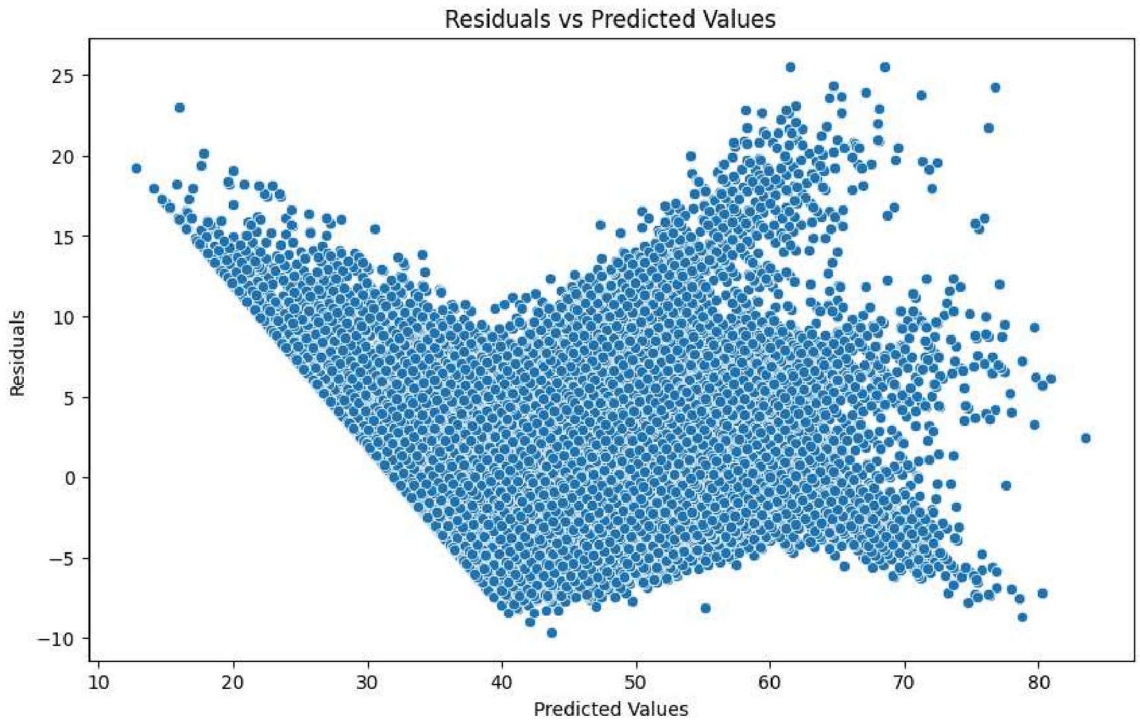
```
from sklearn.metrics import mean_squared_error, r2_score
r2_train=selector.score(X_train, y_train)
r2_test=selector.score(X_test, y_test)
print(r2_train, r2_test)
```

```
0.8455723007686833 0.8586502641503106
```

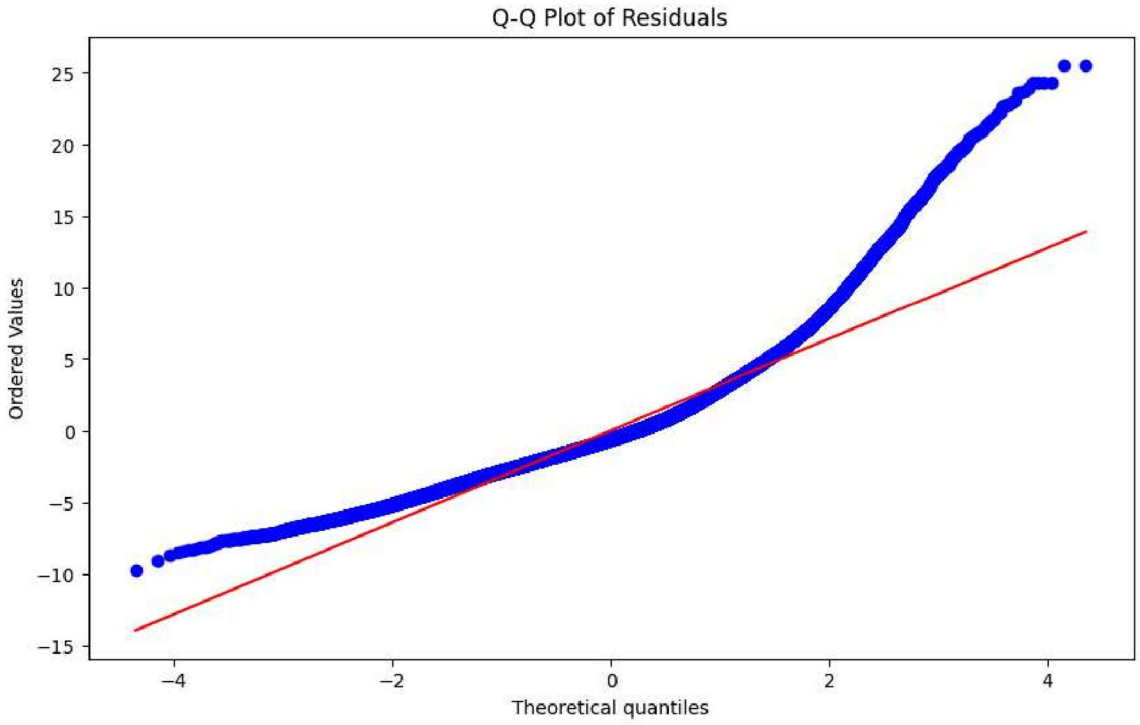
6. Results and Inference [5 marks]

6.1 Perform Residual Analysis [3 marks]

```
# Perform residual analysis using plots like residuals vs predicted values, Q-Q plot and residual histogram
residuals = y_train - selector.predict(X_train)
plt.figure(figsize=(10, 6))
sns.scatterplot(x=selector.predict(X_train), y=residuals)
plt.title('Residuals vs Predicted Values')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```

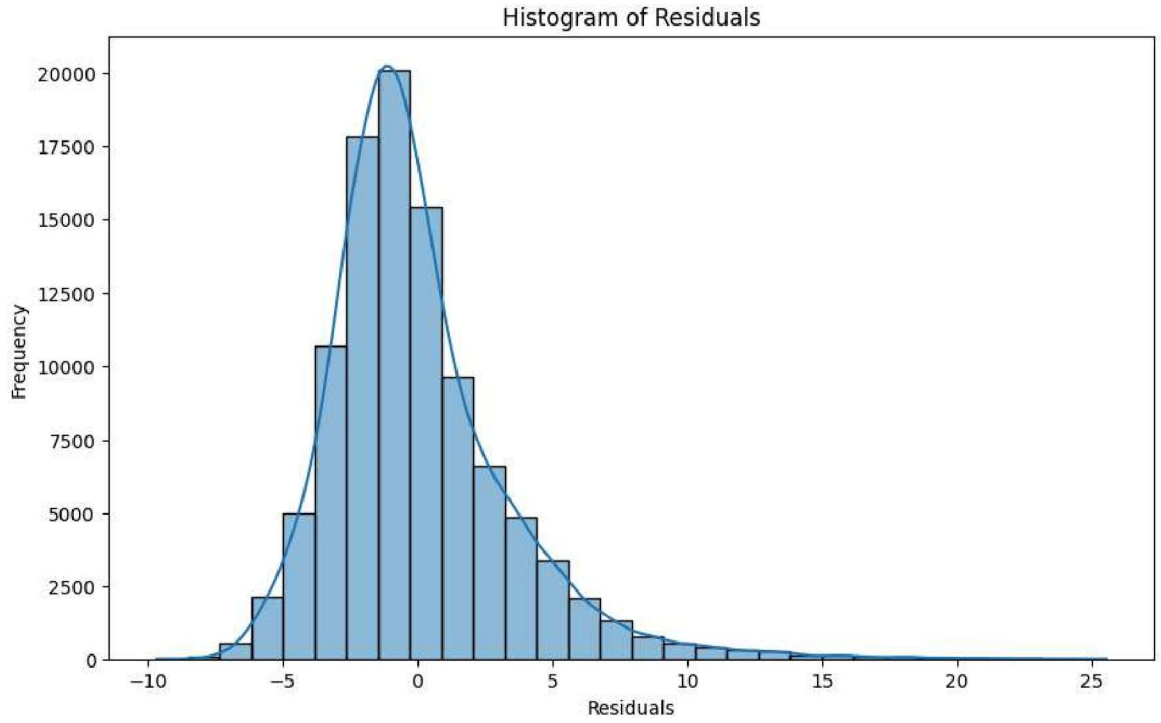



```
import scipy.stats as stats
plt.figure(figsize=(10, 6))
stats.probplot(residuals, plot=plt)
plt.title('Q-Q Plot of Residuals')
plt.show()
```




```
#Histogram of residuals
plt.figure(figsize=(10, 6))
sns.histplot(residuals, bins=30, kde=True)
plt.title('Histogram of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



[link text](https:// link text)[Your inferences here:]Residuals are normally distributed at the centre

6.2 Perform Coefficient Analysis [2 marks]

Perform coefficient analysis to find how changes in features affect the target. Also, the features were scaled, so interpret the scaled and unscaled coefficients to understand the impact of feature changes on delivery time.

```
# Compare the scaled vs unscaled features used in the final models
from sklearn.preprocessing import StandardScaler
#from sklearn import preprocessing # This import is not used
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X_train)
unscaled_features = scaler.inverse_transform(scaled_features)

print("scaled Features:", scaled_features)

print("Unscaled Features:", unscaled_features)
scaled_coefficients = selector.estimator_.coef_

# Create a new scaler and fit it on the selected features from training data
scaler_rfe = StandardScaler()
X_train_selected = X_train[[X_train.columns[i] for i in range(len(selector.support_)) if selector.support_[i]]]
scaler_rfe.fit(X_train_selected)

# Use the new scaler to inverse transform the coefficients
unscaled_coefficients = scaler_rfe.inverse_transform(scaled_coefficients.reshape(1, -1))
unscaled_coefficients = unscaled_coefficients.flatten()

print("Scaled Coefficients:", scaled_coefficients)
print("Unscaled Coefficients:", unscaled_coefficients)
```

```
scaled Features: [[-0.40238292 -0.33400434 -1.00960667 ... -0.42180497  1.01095779
  -0.69140583]
 [-1.18598398 -1.15102186  3.0964339  ...  0.29655206 -0.80017738
   1.44632855]
 [-0.3111701  -1.15102186 -0.64536113 ...  0.75243249  1.57693753
   1.44632855]
 ...
 [ 1.34226958  2.93406577  0.77850777 ...  0.1077531   1.57693753
   1.44632855]
 [ 0.60095811 -0.33400434  0.11624316 ... -0.00736822  1.12415374
  -0.69140583]
```

```
 [ 1.2228637  -0.33400434  0.0500167  ...  0.63731117 -1.02656927
  -0.69140583]]
Unscaled Features: [[-0.40238292 -0.33400434 -1.00960667 ... -0.42180497  1.01095779
  -0.69140583]
 [-1.18598398 -1.15102186  3.0964339  ...  0.29655206 -0.80017738
   1.44632855]
 [-0.3111701  -1.15102186 -0.64536113 ...  0.75243249  1.57693753
   1.44632855]
 ...
 [ 1.34226958  2.93406577  0.77850777 ...  0.1077531   1.57693753
   1.44632855]
 [ 0.60095811 -0.33400434  0.11624316 ... -0.00736822  1.12415374
  -0.69140583]
 [ 1.2228637  -0.33400434  0.0500167  ...  0.63731117 -1.02656927
  -0.69140583]]
Scaled Coefficients: [  1.87182663   0.45152895 -11.95709238  -4.09270265  16.0764257
   4.16939369  -2.21270917   0.64110796]
Unscaled Coefficients: [  1.87182663   0.45152895 -11.95709238  -4.09270265  16.0764257
   4.16939369  -2.21270917   0.64110796]
```

Additionally, we can analyse the effect of a unit change in a feature. In other words, because we have scaled the features, a unit change in the features will not translate directly to the model. Use scaled and unscaled coefficients to find how will a unit change in a feature affect the target.

```
from math import e
# Analyze the effect of a unit change in a feature, say 'total_items'
if 'total_items' in X_train.columns:
    total_items_index = X_train.columns.get_loc('total_items')
    print(total_items_index)
    effect_of_unit_change = scaled_coefficients[total_items_index]
    print("Effect of a unit change in 'total_items':", effect_of_unit_change)
    effect_of_unit_change_unscaled = unscaled_coefficients[total_items_index]
    print("Effect of a unit change in 'total_items' (unscaled):", effect_of_unit_change_unscaled)
else:
    print("'total_items' is not in the selected features.")
```

```
'total_items' is not in the selected features.
```

Note: The coefficients on the original scale might differ greatly in magnitude from the scaled coefficients, but they both describe the same relationships between variables.

Interpretation is key: Focus on the direction and magnitude of the coefficients on the original scale to understand the impact of each variable on the response variable in the original units.

Include conclusions in your report document.

Subjective Questions [20 marks]

Answer the following questions only in the notebook. Include the visualisations/methodologies/insights/outcomes from all the above steps in your report.

Subjective Questions based on Assignment

Question 1. [2 marks]

Are there any categorical variables in the data? From your analysis of the categorical variables from the dataset, what could you infer about their effect on the dependent variable?

Answer

'created_at', 'actual_delivery_time' were the categorical variable in the data.

These variables help in predicting the delivery times accurately.

While these were used to derive at features such as hour_of_day and is_weekend to predict the likely influence on the dependent variable i.e time_taken for delivery.

Question 2. [1 marks]

What does `test_size = 0.2` refer to during splitting the data into training and test sets?

ANSWER

In machine learning, Initially Models developed will be trained first on Training Data which helps the algorithm learn patterns and relationships. The trained Model i.e smaller size data is kept separate to evaluate the model's performance and ensure it doesn't just memorize the training data, but can also make accurate predictions on new inputs based on the test data available. In the context, test_size=0.2 means 20% of data will be used for testing and remaining 80% of data will be used to train the Model.

Question 3. [1 marks]

Looking at the heatmap, which one has the highest correlation with the target variable?

ANSWER

Correlation coefficient is a numerical value that ranges from -1 to +1. It indicates the strength and direction of relationship between the two variable.

As per the heatmap Distance is having highest correlation of 0.46 with the target variable i.e time_taken.

Question 4. [2 marks]

What was your approach to detect the outliers? How did you address them?

ANSWER

>Visualised potential outliers for the target variable and other numerical features using boxplots.

Checked for negative values.

Negative values in total_onshift_dashers, total_busy_dashers, total_outstadning_orders, min_item_price and Max_item_price were available.

One approach in handling inconsistent data is to eliminate the negative values.

Values of min_item_price and max_item_price were zero and subtotal were zero, min_item_price was more than max_item_price same was handled by interchanging the values, negative values were handled by multiplying the negative value by -1.

Similarly all negative values in other columns were handled.

To handle outliers, capping outliers technique, i.e Interquartile Range (IQR) method was used in data preprocessing to limit extreme values that can skew analysis.

The IQR method is effective for identifying and removing outliers, which can be seen in the significant difference in the spread and range of the data before and after outlier removal. This approach is crucial in statistical analysis for ensuring that the results are not skewed by extreme values.

Question 5. [2 marks]

Based on the final model, which are the top 3 features significantly affecting the delivery time?

ANSWER

Top 3 features affectiving the delivery time are

i)Total_outstanding_orders

ii)Distance and

iii)subtotal

General Subjective Questions

Question 6. [3 marks]

Explain the linear regression algorithm in detail

Answer:

The equation of Linear regression obtained after final model is

time_taken=(subtotal * 1.871827)+(num_distinct_items * 0.451529)-(total_onshift_dashers * 11.957092) - (total_busy_dashers * 4.092703) + (total_outstanding_orders * 16.076426) + (distance * 4.169394) - (hour_of_day*2.212709) + (is_weekend *0.641108)

Each coefficient represents the impact of an independent variable on time_taken:

Subtotal (1.87): i.e., the Larger order subtotals increase delivery time, num_distinct_items(0.45)-More unique items slightly increase delivery time,Onshift Dashers(-11.95) i.e., if more dashers are available it will reduce delivery time, Busy Dashers (-4.09) if more dashers are currently delivering reduces delivery time, Outstanding Orders (16.07)- if High pending orders increase delivery time, if Distance (4.17)is Greater, delivery distance increases delivery time,

Hour of Day (-2.21)i.e., Later hours may reduce delivery time, and

the Weekend (0.64) deliveries may take slightly longer time.

Question 7. [2 marks]

Explain the difference between simple linear regression and multiple linear regression

Answer

>The simple linear regression is the most elementary type of regression model which explains the relationship between a dependent variable and one independent variable using a straight line.

The equation for a simple linear regression is Y=mX+C

where C is the intercept

m is the slope of equation

Y is the dependent variable

and X is the independent variable.

where as in a multiple linear regression is a statistical technique to understand the relationship between one dependent variable and several independent variables. Its object is to find a linear equation that can best determine the value of dependent variable Y for different independent variables.

equation is Y = C+M1*X1+M2*X2+....+Mn*Xn

where C is the intercept

M1,M2 .. Mn is the slope of variables X1, X2 .. Xn

Y is the dependent variable

and X1, X2..Xn are the independent variables.

Simple linear regression is simple as it has only one independent variable / one relationship whereas complexity increases in Multiple linear regression.

In simple linear regression typically visualized with 2D scatter plot and a line of best fit whereas Multiple linear regression requires a 3D or multi-dimensional space, often represented using partial regression plots.

Question 8. [2 marks]

What is the role of the cost function in linear regression, and how is it minimized?

ANSWER

The Cost function in a linear regression measures how well the predictions of model aligns with the actual data. It actually measures the difference between the predicted values and actual outcomes helping in minimizing the errors by adjusting its parameters and weights of the model. It provides a way to evaluate the model perforance and helps in optimizing by finding parameters that minimizes the error. It also helps in finding the best fit line for the model

Question 9. [2 marks]

Explain the difference between overfitting and underfitting.

Answer

Overfitting in Machine Learning model happens when a model learns too much / closely follows the training data.

It offers ideal predictions when tested against training data but fail against new, unidentified data.

Overfitting is observed if model is highly complex, model overtrains on a single or a specific data set, if training data contains inapplicable information or noise.

Underfitting is the opposite of overfitting. It happens when a model is too simple to capture data or learn underlying patterns in the training data.

Underfitting model is simple and not capable of representing the complexities in the data, the features used to train model are inadquate representations of underlying factors influencing the target variable.

Size of training dataset is not enough or features are not scaled properly.>

Question 10. [3 marks]

How do residual plots help in diagnosing a linear regression model?

ANSWER

>A residual plot is a graphical representation of the residuals (errors)in a linear regression model against the predictor variables in a regression analysis. Residuals are the difference between the observed values of the dependent variable and the predicted values obtained from the linear regression model. In simple terms a residual plot shows how far off the predictions are from the actual data points. These plots help assess the assumptions and adequacy of the regression model.

In residual plots, if the residuals exhibit a random pattern around the horizontal axis, it indicates that the regression model is appropriate and adequately captures the variability in the data. However, if the residuals show a systematic pattern, such as a curve or funnel shape, it suggests that the regression model may not be the best fit for the data.

Residual plots helps in identifying the outliers or influential data points that may affect the regression analysis results.

By examining residual plots, we can make informed decisions about the validity and reliability of the regression model and make any required adjustments to the regression model to improve its accuracy.