Syed Ismail N

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

Field	Description
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

```
# Import essential libraries for data manipulation and analysis
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

1. Loading the data

Load 'porter_data_1.csv' as a DataFrame

```
# Importing the file porter_data_1.csv
df = pd.read_csv('porter_data_1.csv')
```

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

2.1.2 [3 marks]

Convert categorical fields to appropriate data type

```python
# Convert categorical features to category type
categorical_cols = ['market_id', 'store_primary_category', 'order_protocol']
for col in categorical_cols:
    df[col] = df[col].astype('category')
```

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

```python
# Calculate time taken in minutes
df['time_taken'] = (df['actual_delivery_time'] -
df['created_at']).dt.total_seconds() / 60
```

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.

```python
# Extract the hour and day of week from the 'created_at' timestamp
df['order_hour'] = df['created_at'].dt.hour.astype('category')
df['order_day_of_week'] = df['created_at'].dt.dayofweek.astype('category')

# Create a categorical feature 'isWeekend'
df['isWeekend'] = df['order_day_of_week'].apply(lambda x: 1 if x >= 5 else
0).astype('category')

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

2.3 Creating training and validation sets [5 marks]

2.3.1 [2 marks]

Define target and input features

```python
# Define target variable (y) and features (X)
y = df.pop('time_taken')
X = df
```

2.3.2 [3 marks]

Split the data into training and test sets

```python
from sklearn.model_selection import train_test_split
```

```python
# Split data into training and testing sets, preserving index
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

```python
print(X_train.shape)
print(y_train.shape)
```

```
(140621, 15)
(140621,)
```

```python
print(X_test.shape)
print(y_test.shape)
```

```
(35156, 15)
(35156,)
```

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

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

3.1 Feature Distributions [7 marks]

```python
# Define numerical and categorical columns for easy EDA and data manipulation
numerical_cols_train = X_train.select_dtypes(include=np.number).columns
categorical_cols_train = X_train.select_dtypes(include='category').columns


print("Numerical columns:", numerical_cols_train.tolist())
print("Categorical columns:", categorical_cols_train.tolist())
```

```
Numerical columns: ['total_items', 'subtotal', 'num_distinct_items',
'min_item_price', 'max_item_price', 'total_onshift_dashers',
'total_busy_dashers', 'total_outstanding_orders', 'distance']
Categorical columns: ['market_id', 'store_primary_category',
'order_protocol', 'order_hour', 'order_day_of_week', 'isWeekend']
```

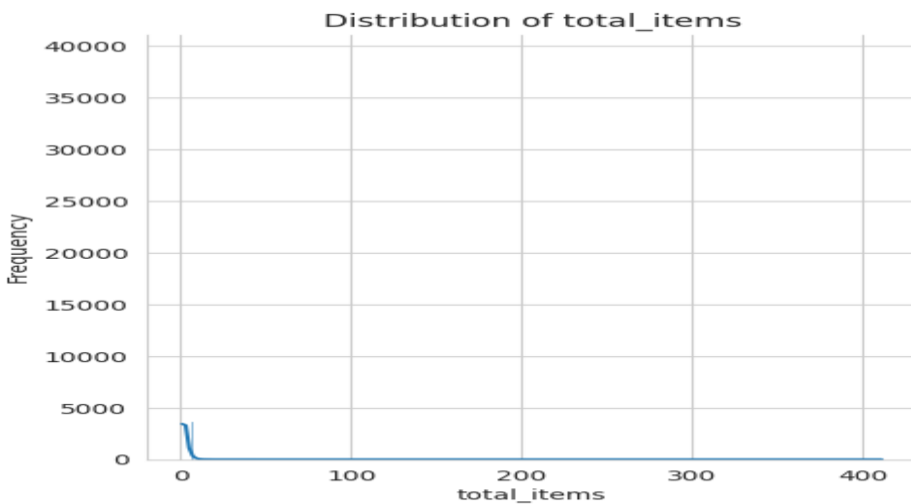
3.1.1 [3 marks]

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

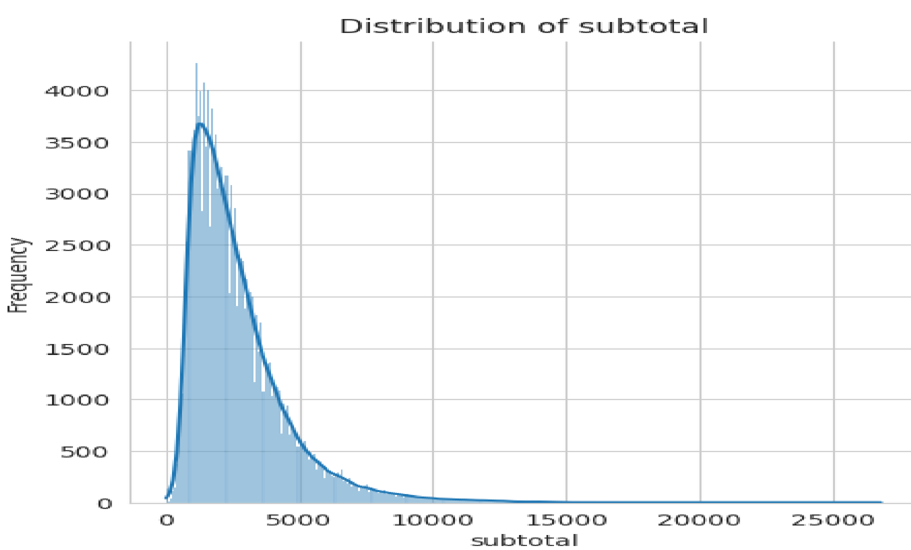
```python
# Plot distributions for all numerical columns
sns.set_style("whitegrid")
for col in numerical_cols_train:
    plt.figure(figsize=(10, 6))
    sns.displot(data=X_train, x=col, kde=True) # Using displot with kde=True
for a smooth curve
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
```

```
    plt.ylabel('Frequency')
    plt.show()
```

<Figure size 1000x600 with 0 Axes>

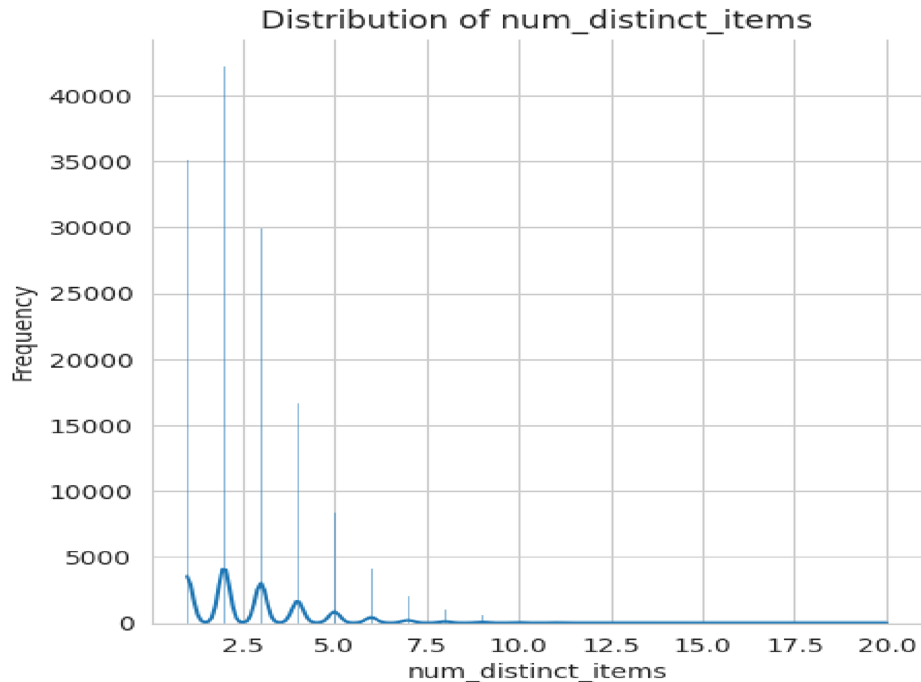


<Figure size 1000x600 with 0 Axes>

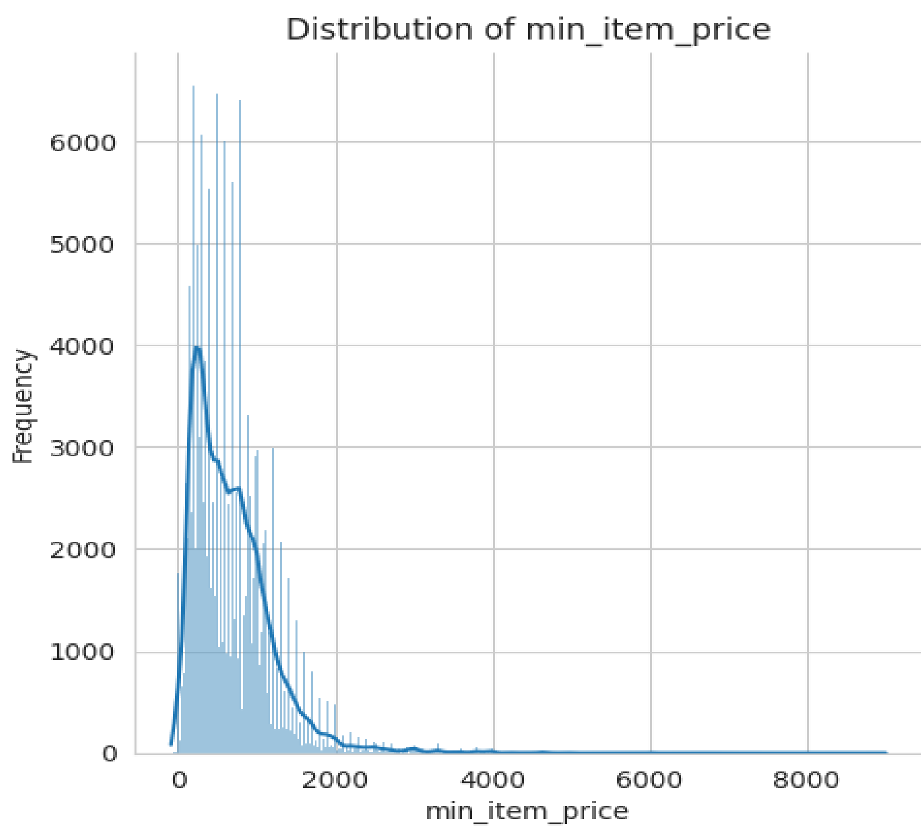


<Figure size 1000x600 with 0 Axes>

Syed Ismail N

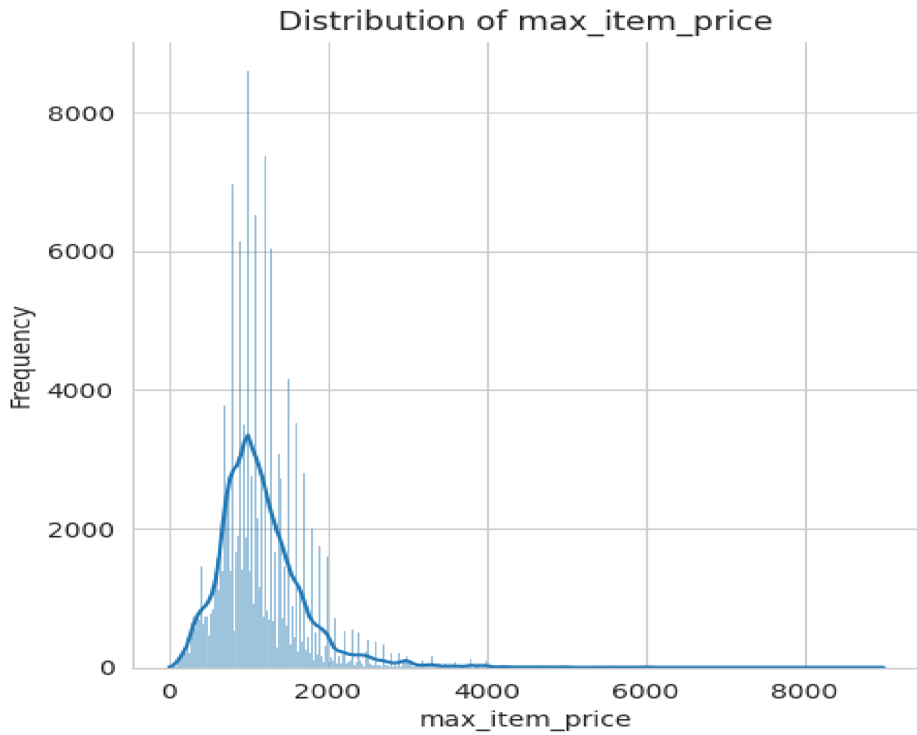


<Figure size 1000x600 with 0 Axes>

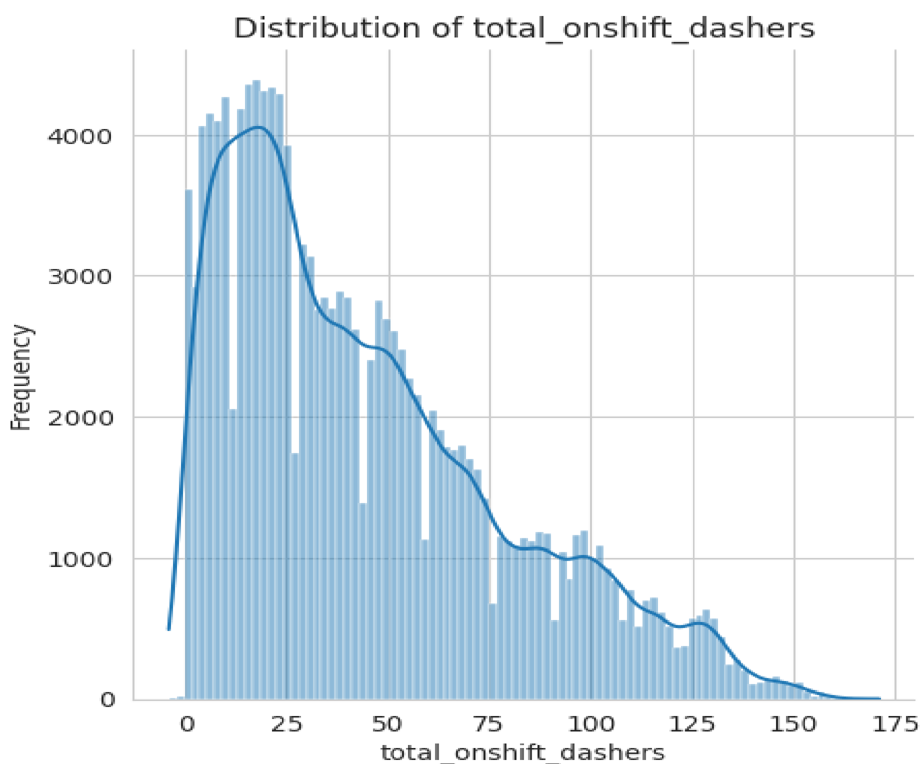


<Figure size 1000x600 with 0 Axes>

Syed Ismail N

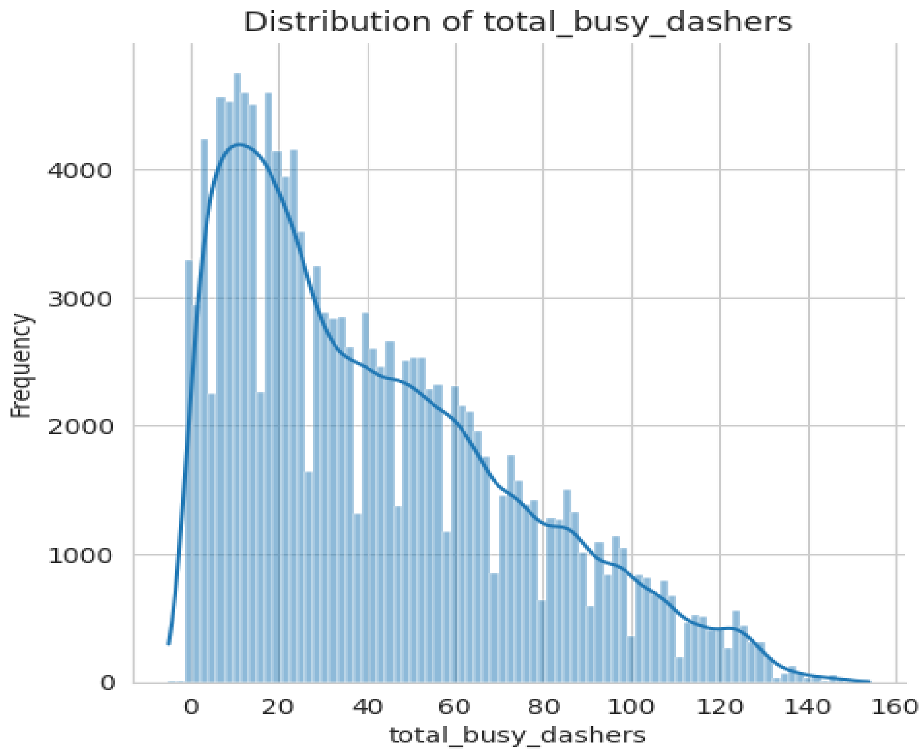


<Figure size 1000x600 with 0 Axes>

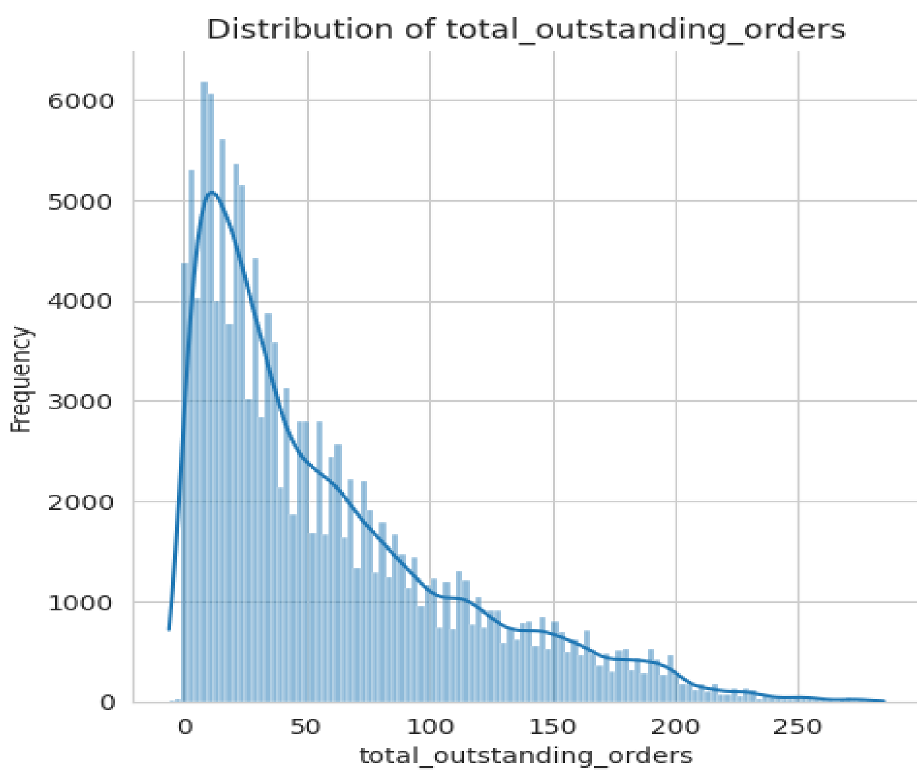


<Figure size 1000x600 with 0 Axes>

Syed Ismail N

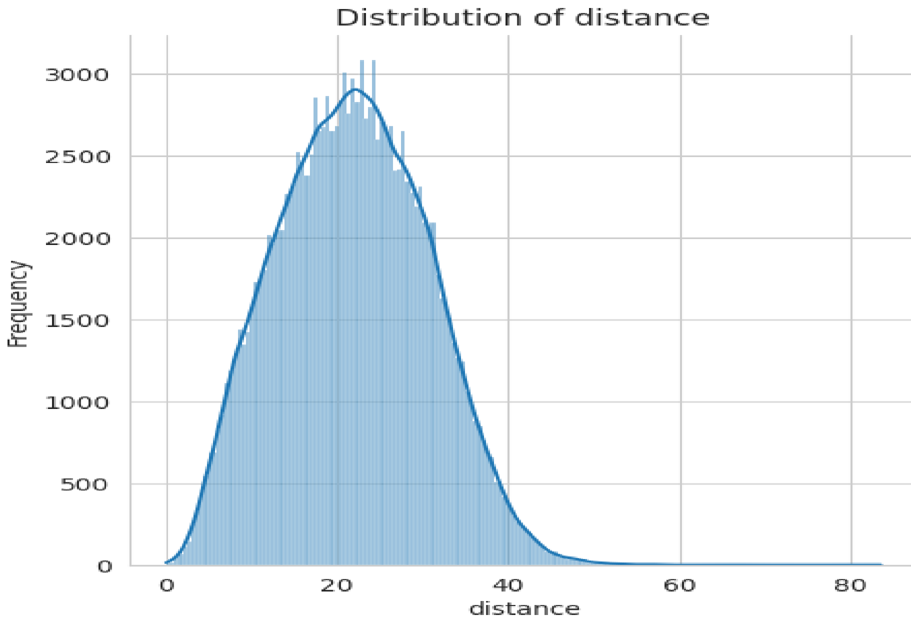


<Figure size 1000x600 with 0 Axes>



<Figure size 1000x600 with 0 Axes>

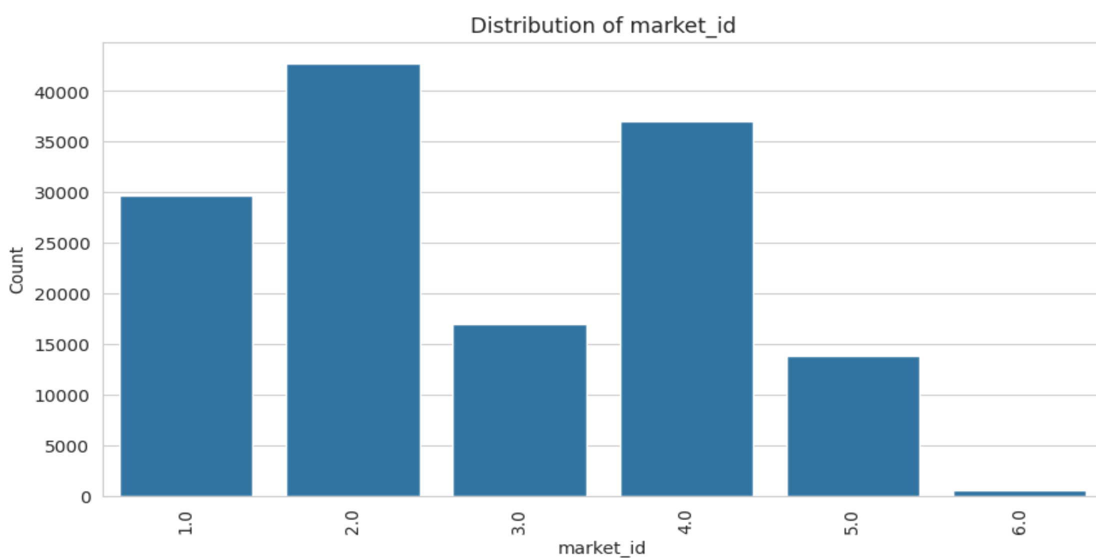
Syed Ismail N



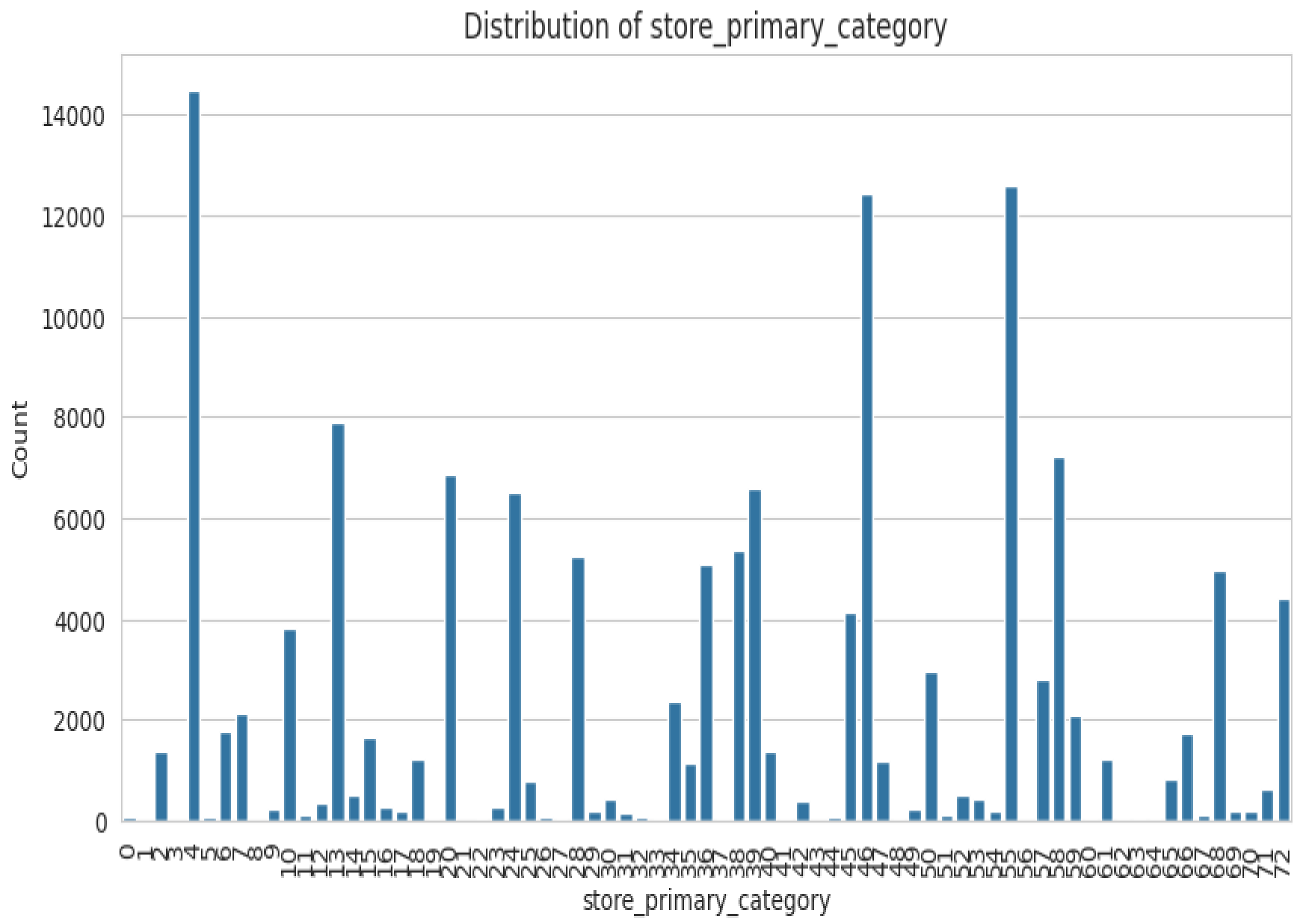
3.1.2 [2 marks]

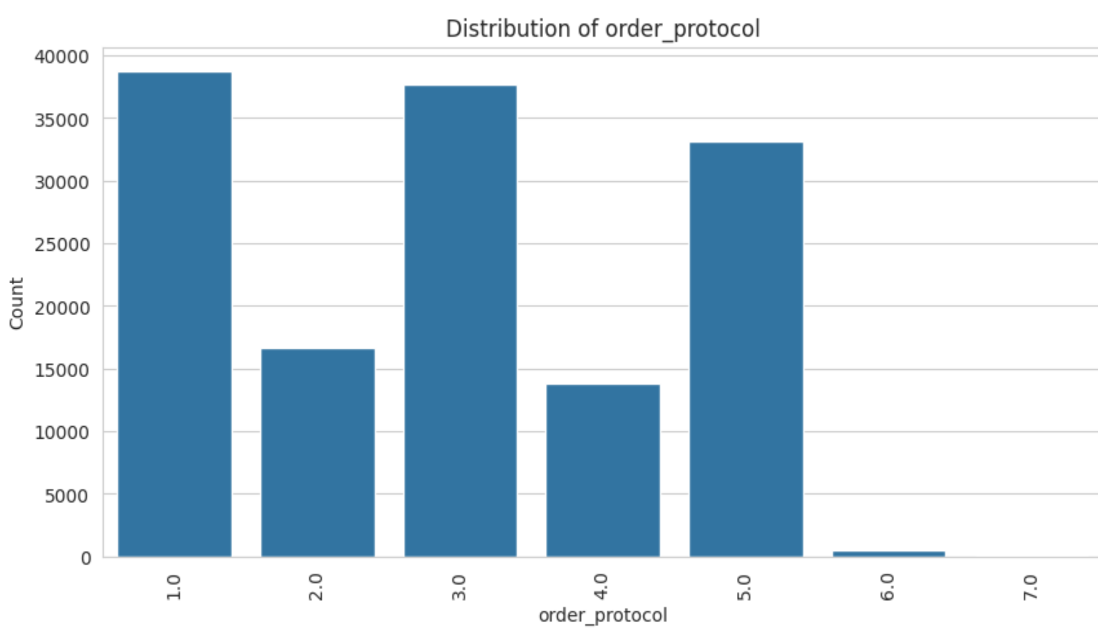
Check the distribution of categorical features

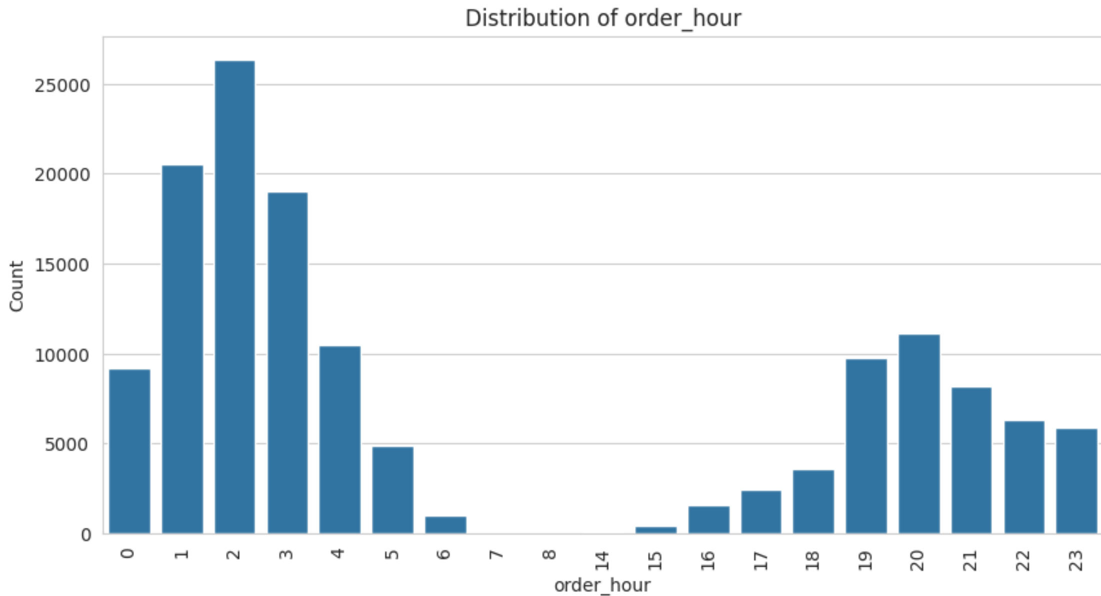
```
# Distribution of categorical columns
for col in categorical_cols_train:
    plt.figure(figsize=(10, 5))
    sns.countplot(data=X_train, x=col)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=90)
    plt.show()
```

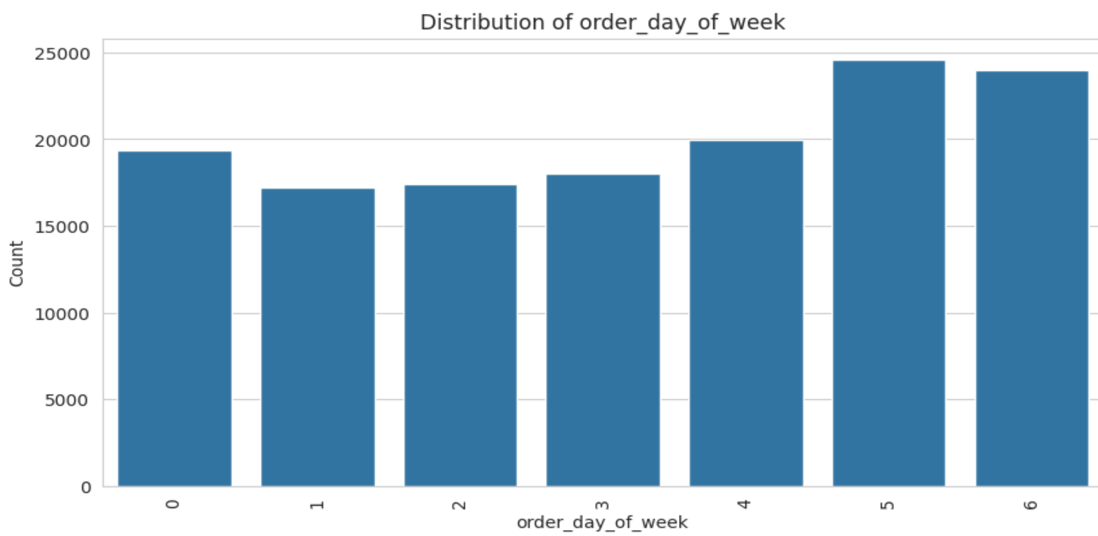


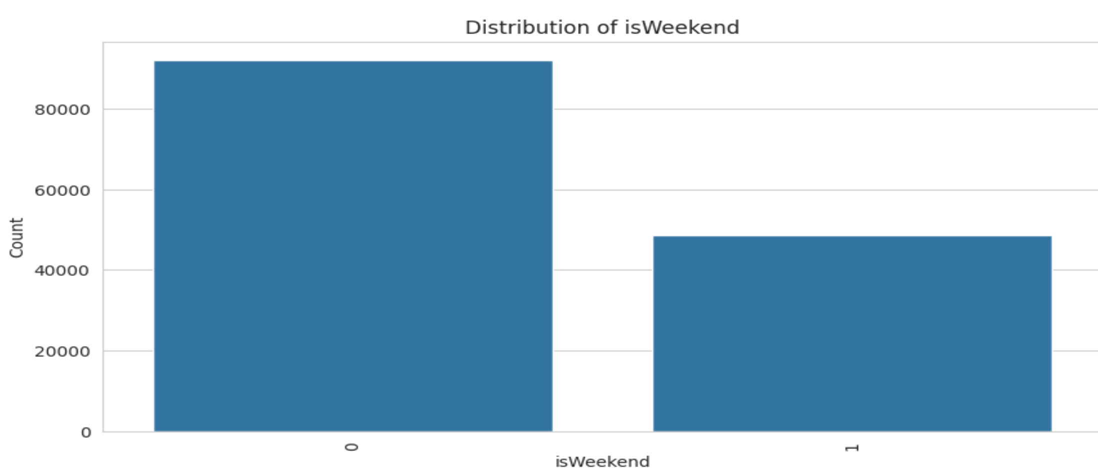
Syed Ismail N






Distribution of order_hour
Count
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
7
8
14
15
16
17
18
19
20
21
22
23
order_hour


Distribution of order_day_of_week
Count
25000
20000
15000
10000
5000
0
0
1
2
3
4
5
6
order_day_of_week

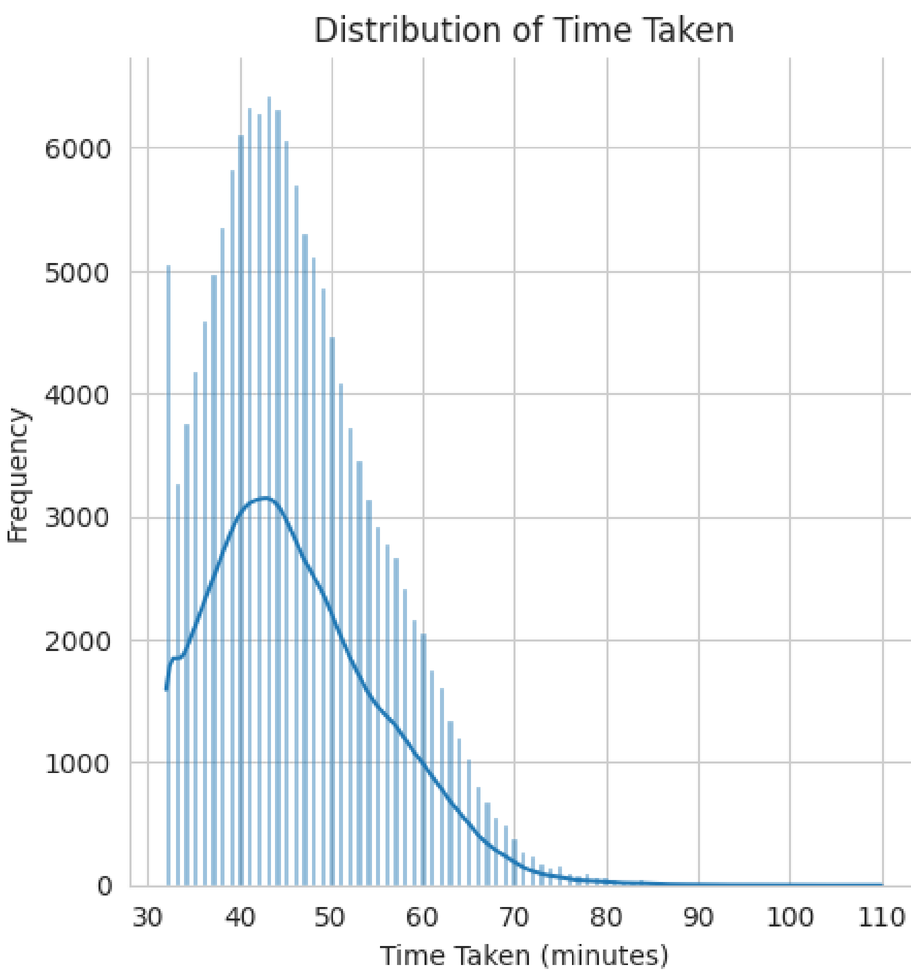

Distribution of isWeekend
Count
80000
60000
40000
20000
0
0
1
isWeekend

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, 5))
sns.displot(y_train, kde=True)
plt.title('Distribution of Time Taken')
plt.xlabel('Time Taken (minutes)')
plt.ylabel('Frequency')
plt.show()
```

```
<Figure size 1000x500 with 0 Axes>
```



***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

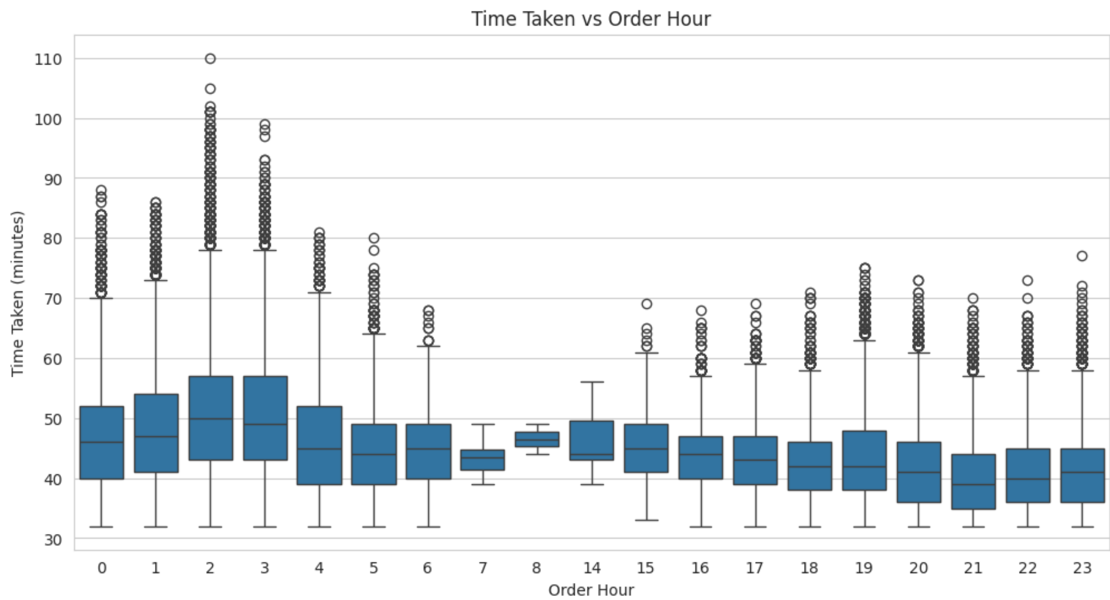
Syed Ismail N

```python
# Scatter plot to visualise the relationship between time_taken and all
numerical features

# Determine the number of plots and calculate grid dimensions
n_plots = len(X_train.columns)
n_cols = 3  # Number of columns in the grid
n_rows = (n_plots + n_cols - 1) // n_cols  # Calculate the number of rows
needed

plt.figure(figsize=(n_cols * 5, n_rows * 5))

for i, col in enumerate(X_train.columns):
    plt.subplot(n_rows, n_cols, i + 1)
    sns.scatterplot(x=col, y=y_train, data=X_train)
    plt.title(f'time_taken vs {col}')
    plt.xlabel(col)
    plt.ylabel('Time Taken (minutes)')

plt.tight_layout()
plt.show()
```

Syed Ismail N

Syed Ismail N

```python
# Show the distribution of time_taken for different hours
plt.figure(figsize=(12, 6))
sns.boxplot(x='order_hour', y=y_train, data=X_train)
plt.title('Time Taken vs Order Hour')
plt.xlabel('Order Hour')
plt.ylabel('Time Taken (minutes)')
plt.show()
```



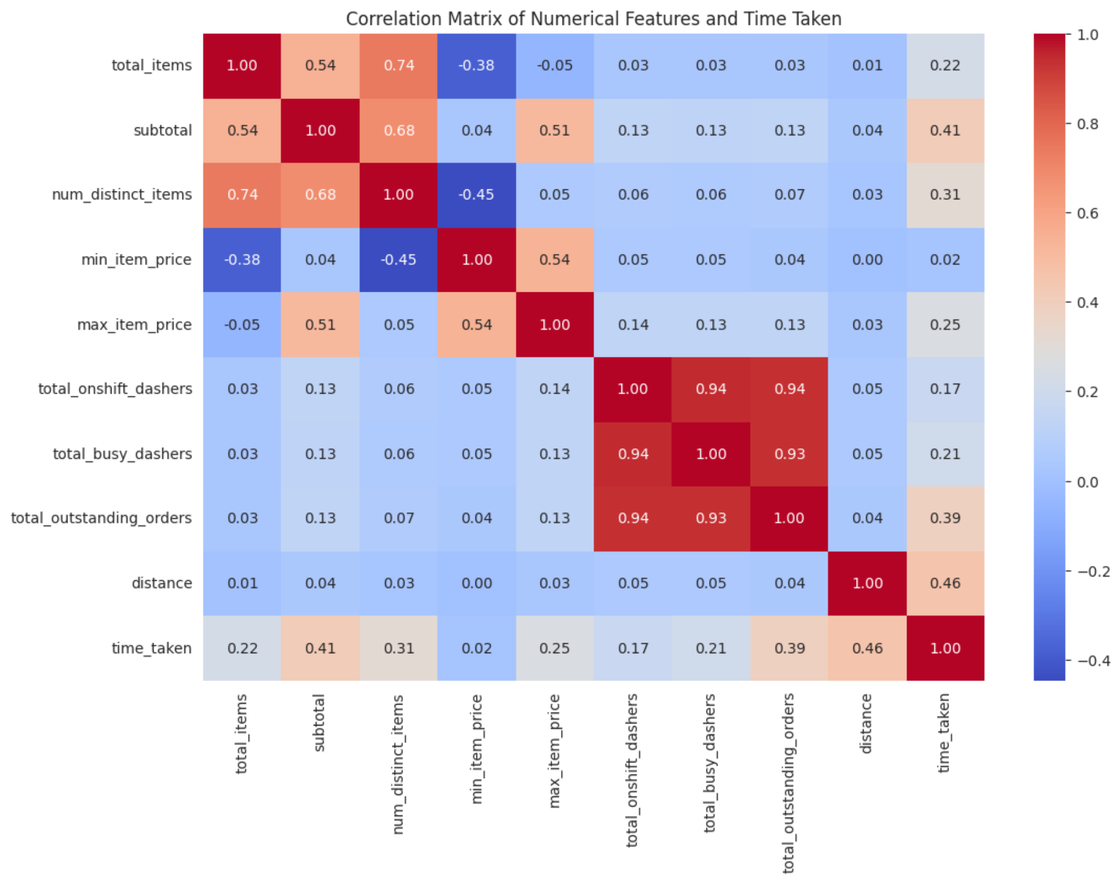
3.3 Correlation Analysis [5 marks]

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

3.3.1 [3 marks]

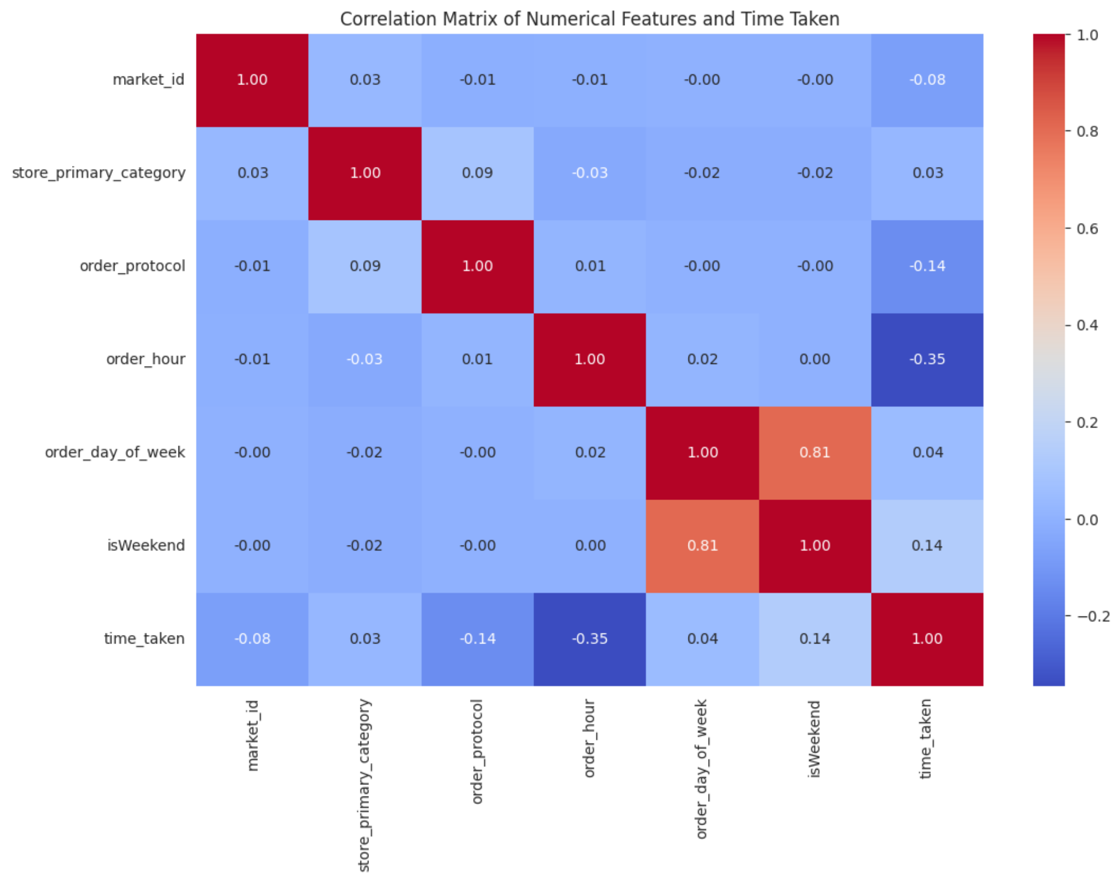
Plot a heatmap to display correlations

```python
# Concatenate X_train and y_train to calculate correlations with the target
variable
df_num_train = pd.concat([X_train[numerical_cols_train], y_train], axis=1)

# Calculate the correlation matrix
num_train_correlation_matrix = df_num_train.corr()

# Plot the heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(num_train_correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Matrix of Numerical Features and Time Taken')
plt.show()
```




```
# Concatenate X_train and y_train to calculate correlations with the target
variable
df_cat_train = pd.concat([X_train[categorical_cols_train], y_train], axis=1)

# Calculate the correlation matrix
cat_train_correlation_matrix = df_cat_train.corr()

# Plot the heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(cat_train_correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Matrix of Numerical Features and Time Taken')
plt.show()
```

Syed Ismail N



3.3.2 [2 marks]

Drop the columns with weak correlations with the target variable

```
# Drop 3-5 weakly correlated columns from training dataset
# Based on both the heatmaps; min_item_price, order_day_of_week and
story_primary_category are weakly correlated columns
weak_corr_cols_train = ['min_item_price', 'order_day_of_week',
'store_primary_category']
X_train = X_train.drop(weak_corr_cols_train, axis=1)

X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 140621 entries, 102712 to 121958
Data columns (total 12 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 140621 non-null  category
 1   order_protocol            140621 non-null  category
 2   total_items               140621 non-null  int64
 3   subtotal                  140621 non-null  int64
 4   num_distinct_items        140621 non-null  int64
 5   max_item_price            140621 non-null  int64
```

```
 6   total_onshift_dashers     140621 non-null  float64
 7   total_busy_dashers        140621 non-null  float64
 8   total_outstanding_orders  140621 non-null  float64
 9   distance                  140621 non-null  float64
 10  order_hour                140621 non-null  category
 11  isWeekend                 140621 non-null  category
dtypes: category(4), float64(4), int64(4)
memory usage: 14.2 MB
```

```
weak_corr_cols_train
```

```
['min_item_price', 'order_day_of_week', 'store_primary_category']
```

```
# Update numerical_cols by removing the weakly correlated columns
numerical_cols = numerical_cols_train.difference(weak_corr_cols_train)

# Update categorical_cols by removing the weakly correlated columns
categorical_cols = categorical_cols_train.difference(weak_corr_cols_train)
```

```
numerical_cols
```

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

```
categorical_cols
```

```
Index(['isWeekend', 'market_id', 'order_hour', 'order_protocol'],
dtype='object')
```

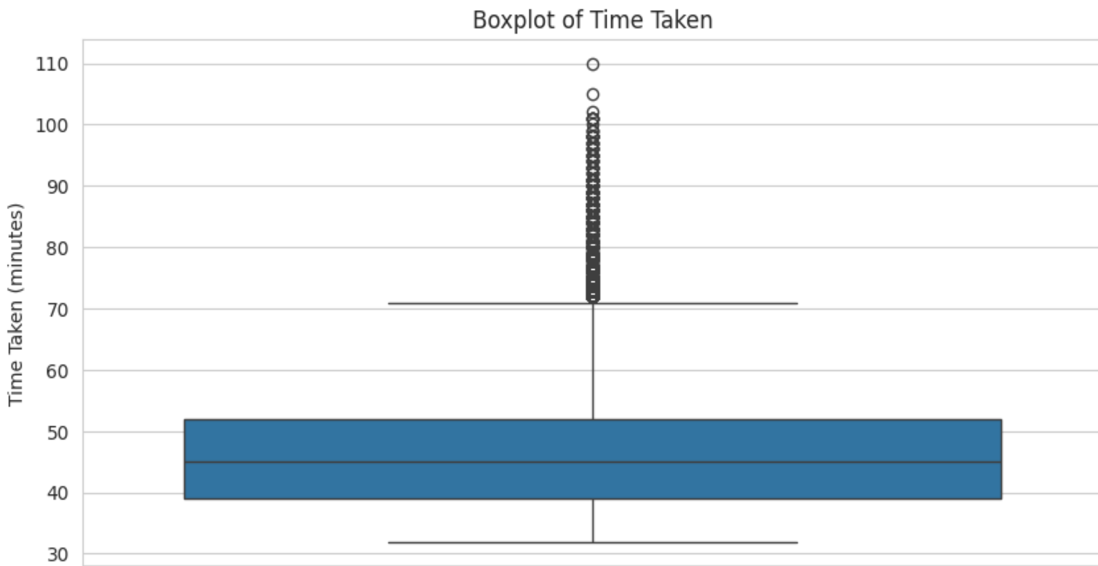
3.4 Handling the Outliers *[5 marks]*

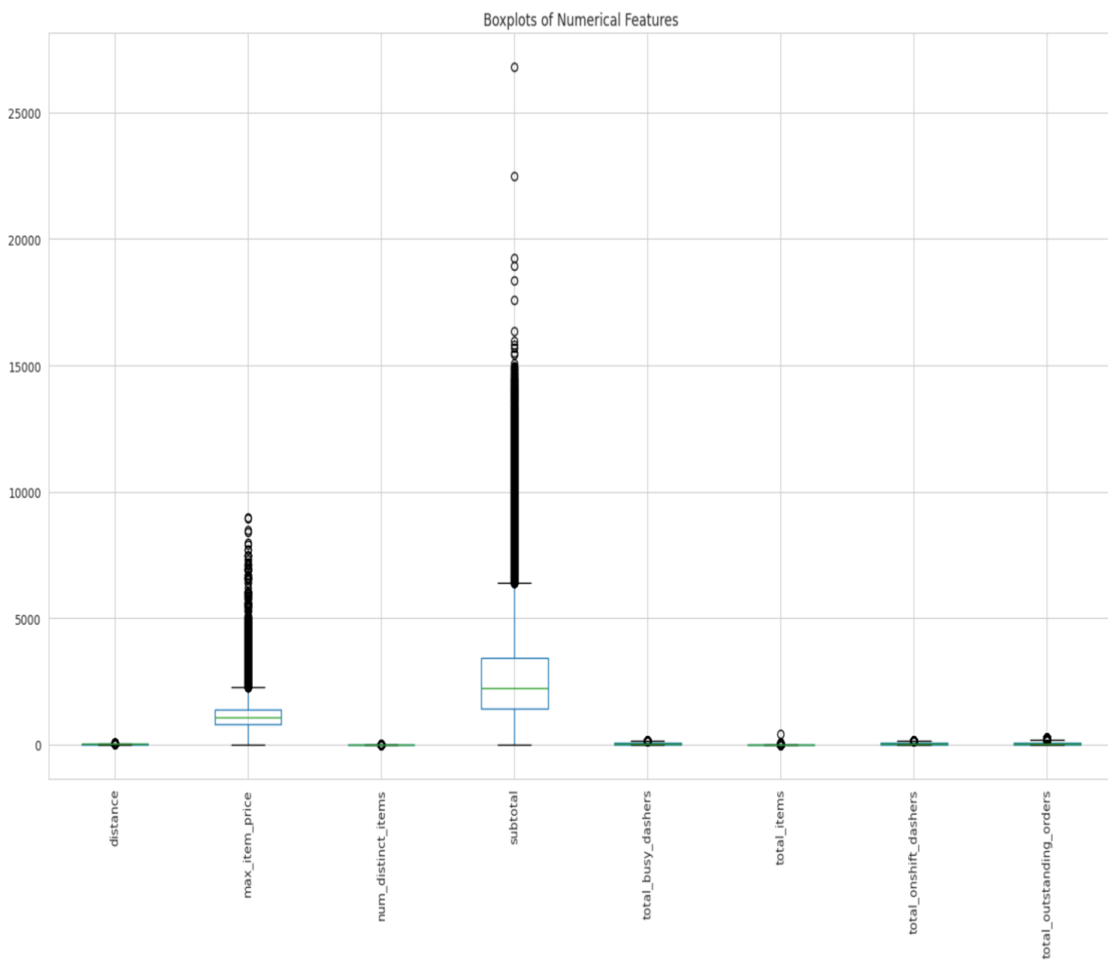
3.4.1 [2 marks]

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

```
# Boxplot for time_taken
plt.figure(figsize=(10, 5))
sns.boxplot(y=y_train)
plt.title('Boxplot of Time Taken')
plt.ylabel('Time Taken (minutes)')
plt.show()

# Boxplots for other numerical features
X_train[numerical_cols].boxplot(figsize=(15, 10))
plt.title('Boxplots of Numerical Features')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Syed Ismail N





3.4.2 [3 marks]

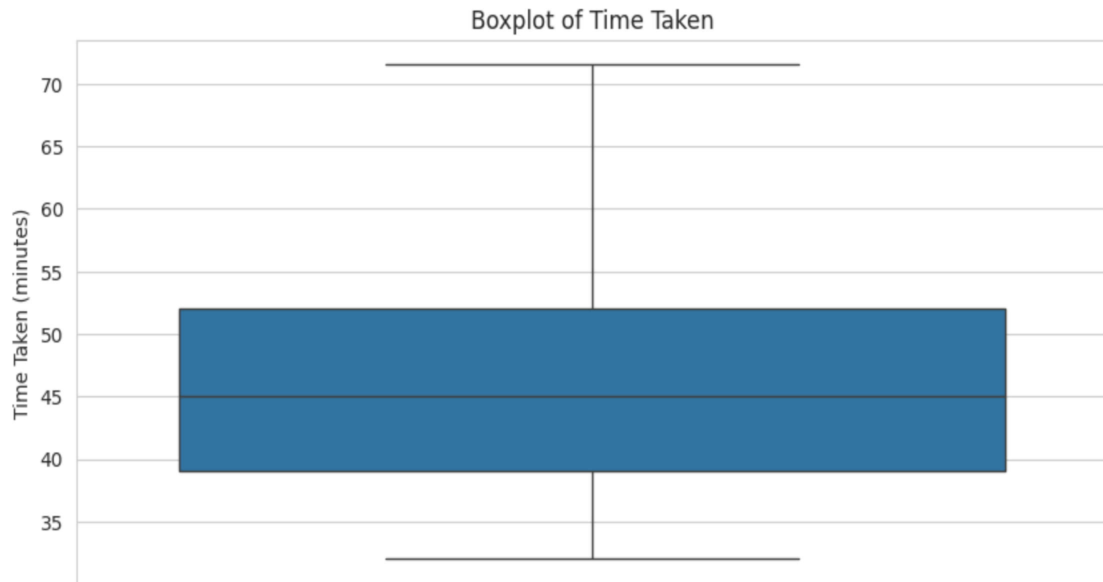
Handle outliers present in all columns

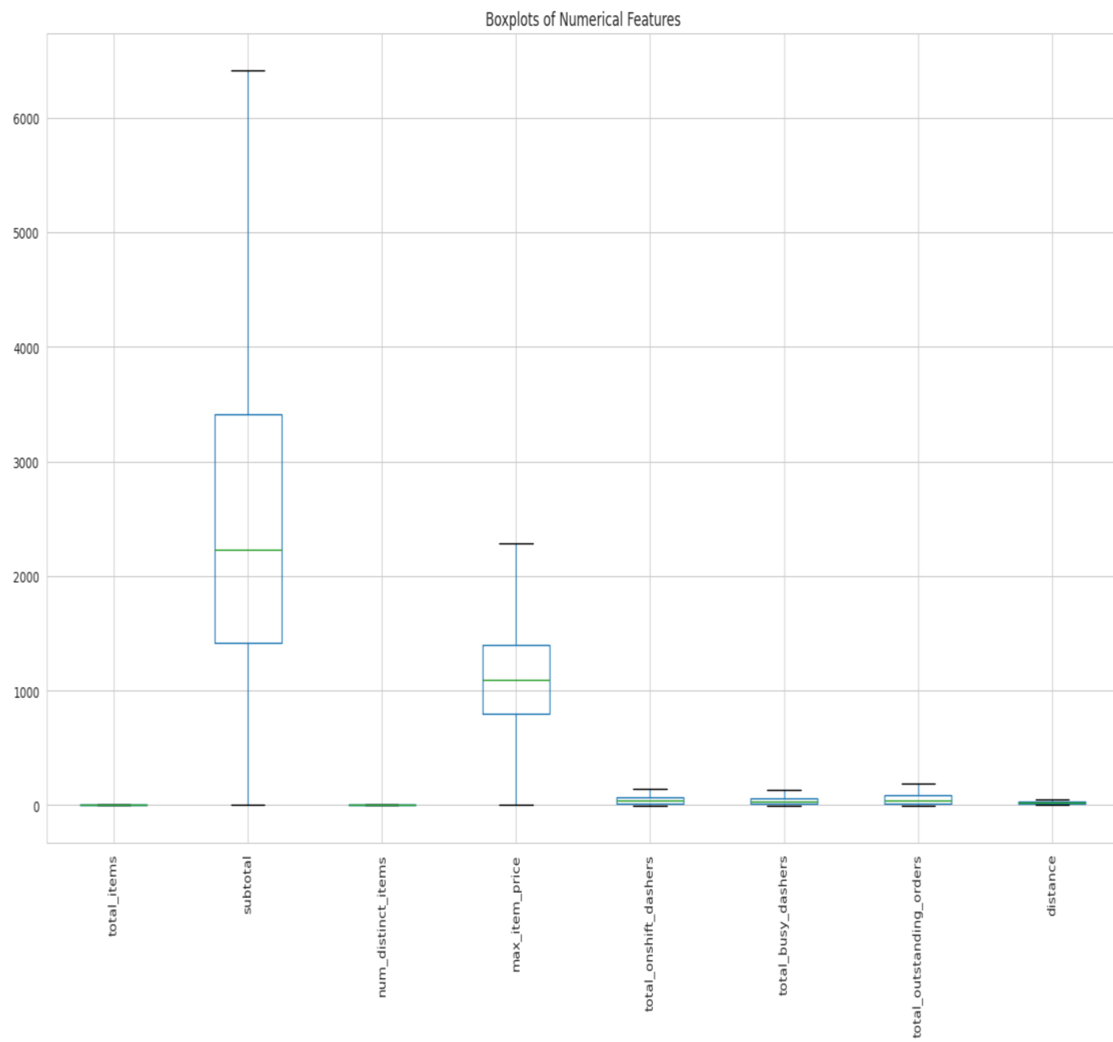
```python
# Handle outliers

for col in numerical_cols:
      Q1 = X_train[col].quantile(0.25)
      Q3 = X_train[col].quantile(0.75)
      IQR = Q3 - Q1
      lower_bound = Q1 - 1.5 * IQR
      upper_bound = Q3 + 1.5 * IQR
      X_train[col] = X_train[col].clip(lower_bound, upper_bound)

# Handle outliers in y_train without losing Series properties
Q1_y = y_train.quantile(0.25)
Q3_y = y_train.quantile(0.75)
IQR_y = Q3_y - Q1_y
lower_bound_y = Q1_y - 1.5 * IQR_y
upper_bound_y = Q3_y + 1.5 * IQR_y
y_train = y_train.clip(lower_bound_y, upper_bound_y)

print("Outliers handled in X_train and y_train.")
```

```
Outliers handled in X_train and y_train.
```

```python
# Boxplot for time_taken
plt.figure(figsize=(10, 5))
sns.boxplot(y=y_train)
plt.title('Boxplot of Time Taken')
plt.ylabel('Time Taken (minutes)')
plt.show()

# Boxplots for other numerical features
X_train[X_train.select_dtypes(include=np.number).columns].boxplot(figsize=(15
, 10))
plt.title('Boxplots of Numerical Features')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Syed Ismail N





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_cols_test = X_test.select_dtypes(include=np.number).columns
categorical_cols_test = X_test.select_dtypes(include='category').columns

print("Numerical columns (Test):", numerical_cols_test.tolist())
print("Categorical columns (Test):", categorical_cols_test.tolist())
```

```
Numerical columns (Test): ['total_items', 'subtotal', 'num_distinct_items',
'min_item_price', 'max_item_price', 'total_onshift_dashers',
'total_busy_dashers', 'total_outstanding_orders', 'distance']
Categorical columns (Test): ['market_id', 'store_primary_category',
'order_protocol', 'order_hour', 'order_day_of_week', 'isWeekend']
```

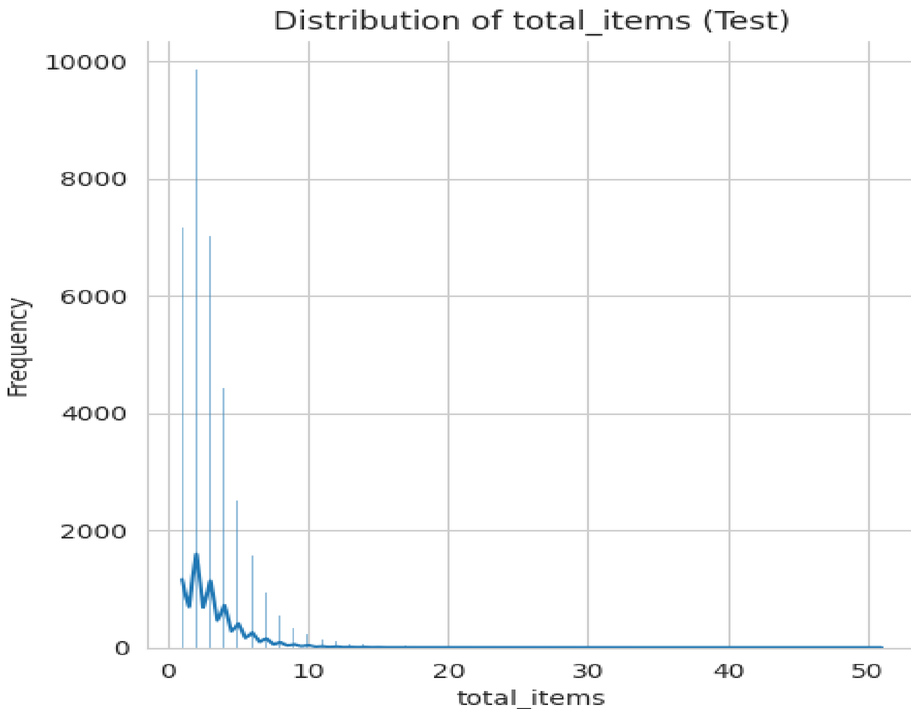
4.1 Feature Distributions

4.1.1

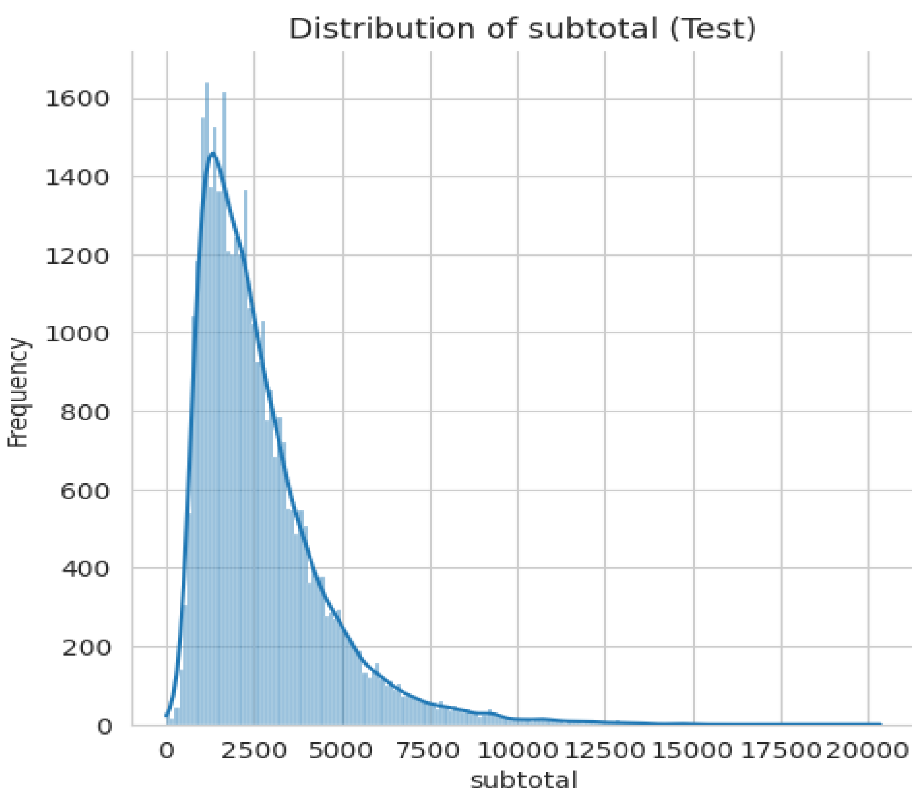
Plot distributions for numerical columns in the validation set to understand their spread and any skewness

```
# Plot distributions for all numerical columns in the test set using
sns.histplot
numerical_cols_test = X_test.select_dtypes(include=np.number).columns

for col in numerical_cols_test:
    plt.figure(figsize=(10, 6))
    sns.displot(data=X_test, x=col, kde=True)
    plt.title(f'Distribution of {col} (Test)')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

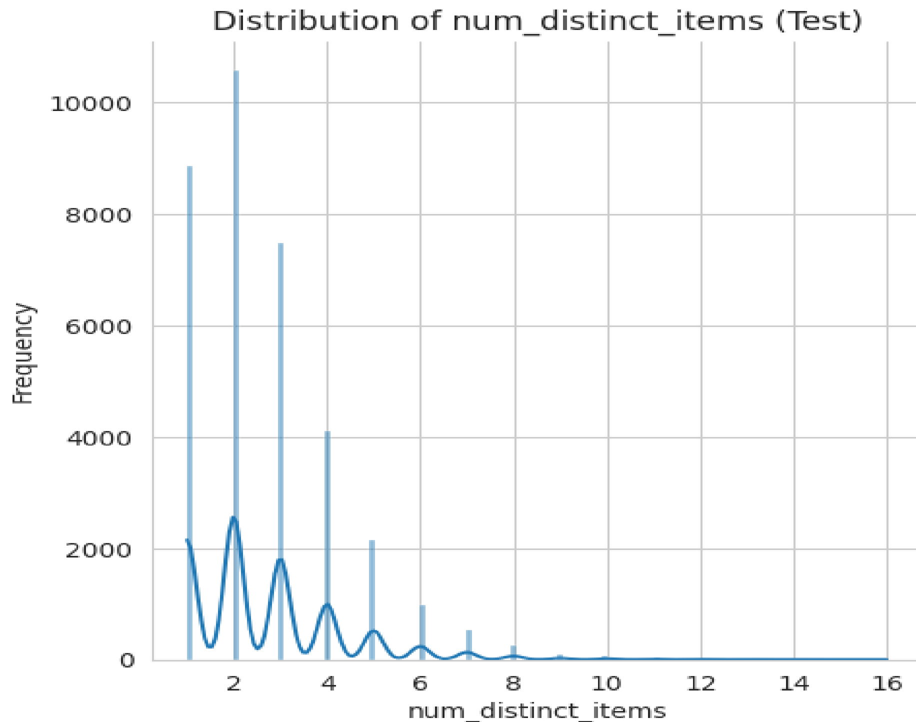


<Figure size 1000x600 with 0 Axes>

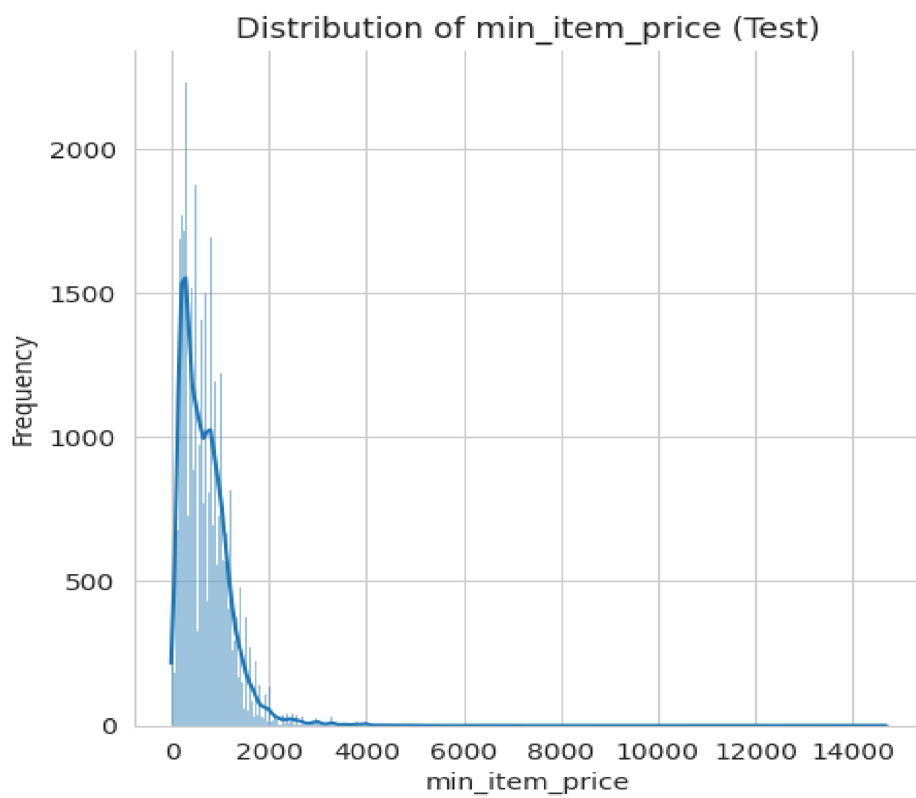


<Figure size 1000x600 with 0 Axes>

Syed Ismail N

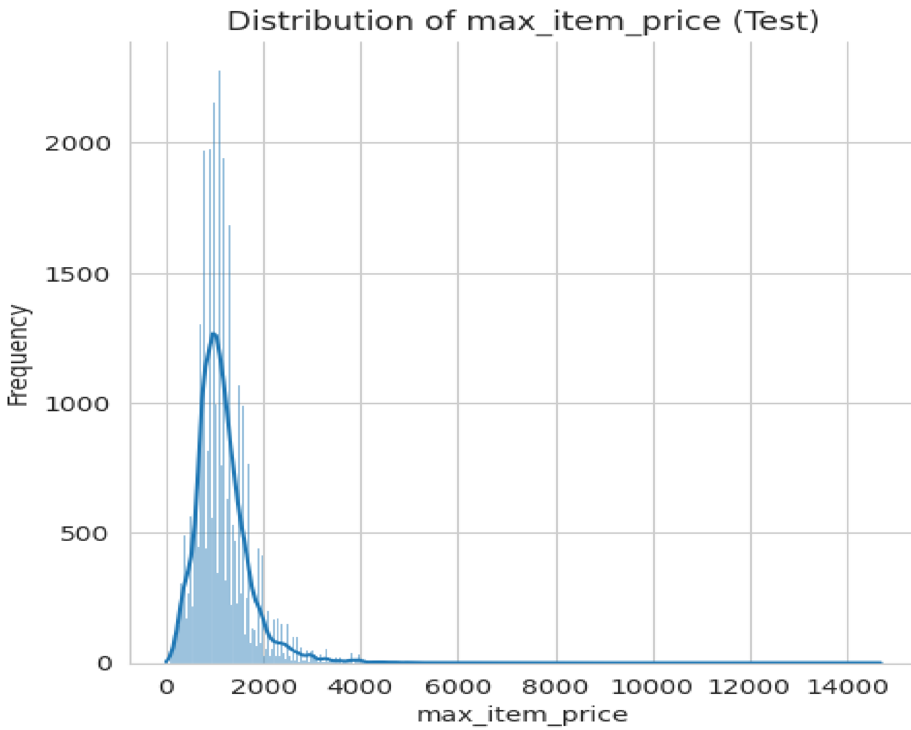


<Figure size 1000x600 with 0 Axes>

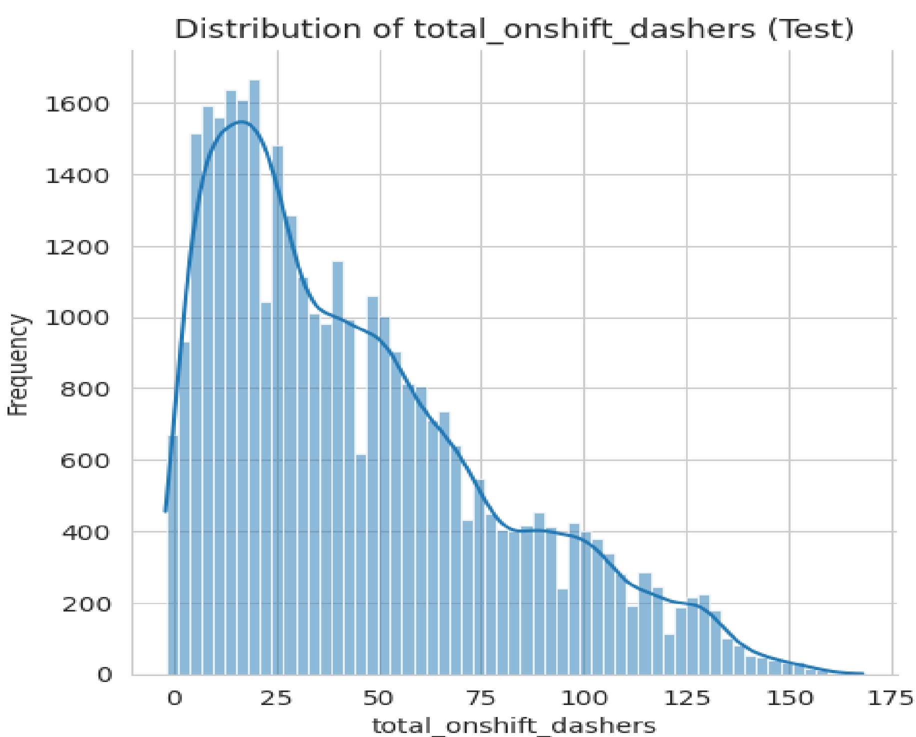


<Figure size 1000x600 with 0 Axes>

Syed Ismail N

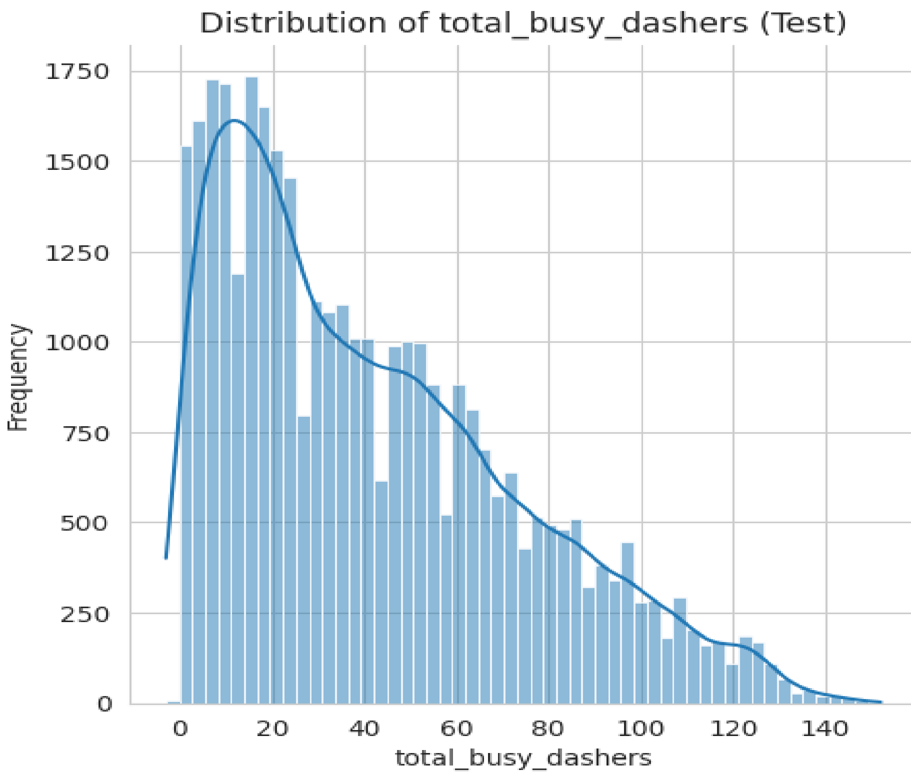



```
<Figure size 1000x600 with 0 Axes>
```

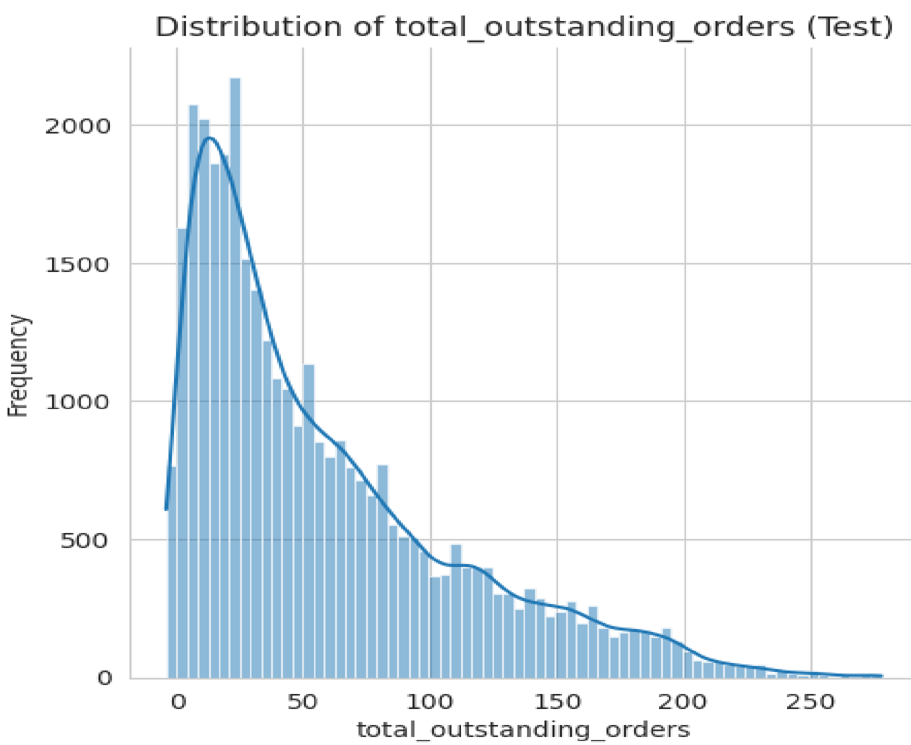



```
<Figure size 1000x600 with 0 Axes>
```

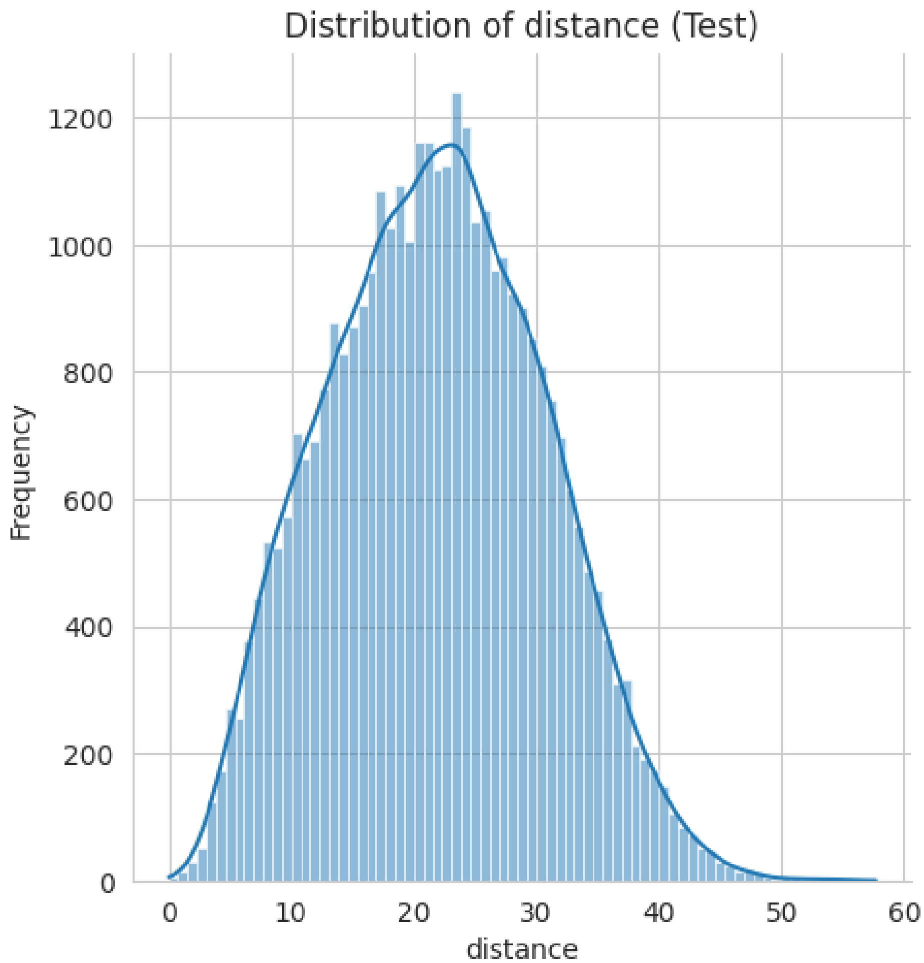
Syed Ismail N



<Figure size 1000x600 with 0 Axes>



<Figure size 1000x600 with 0 Axes>

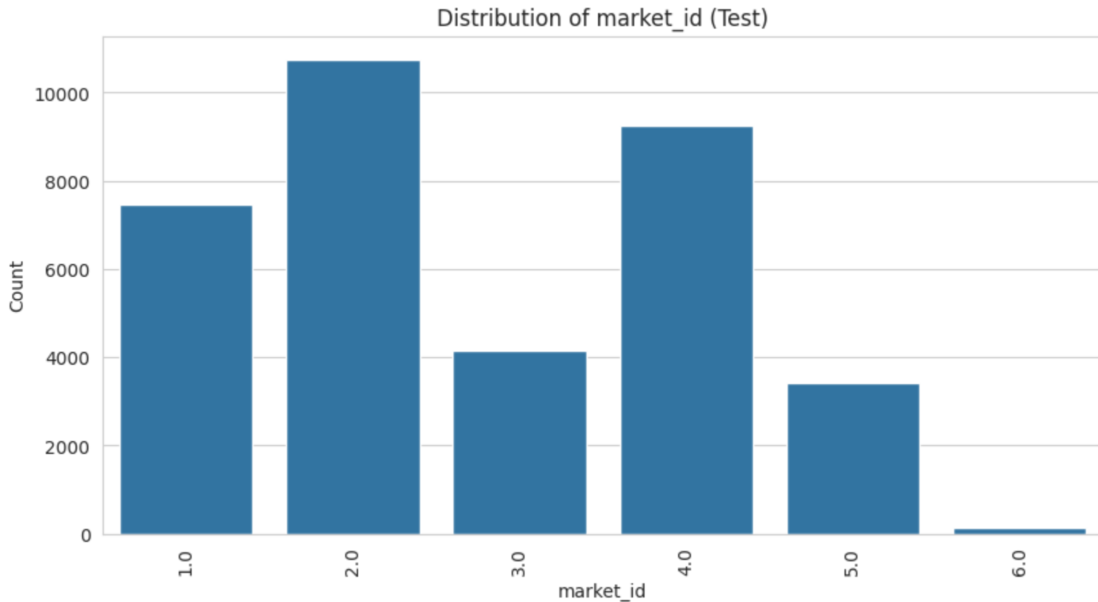


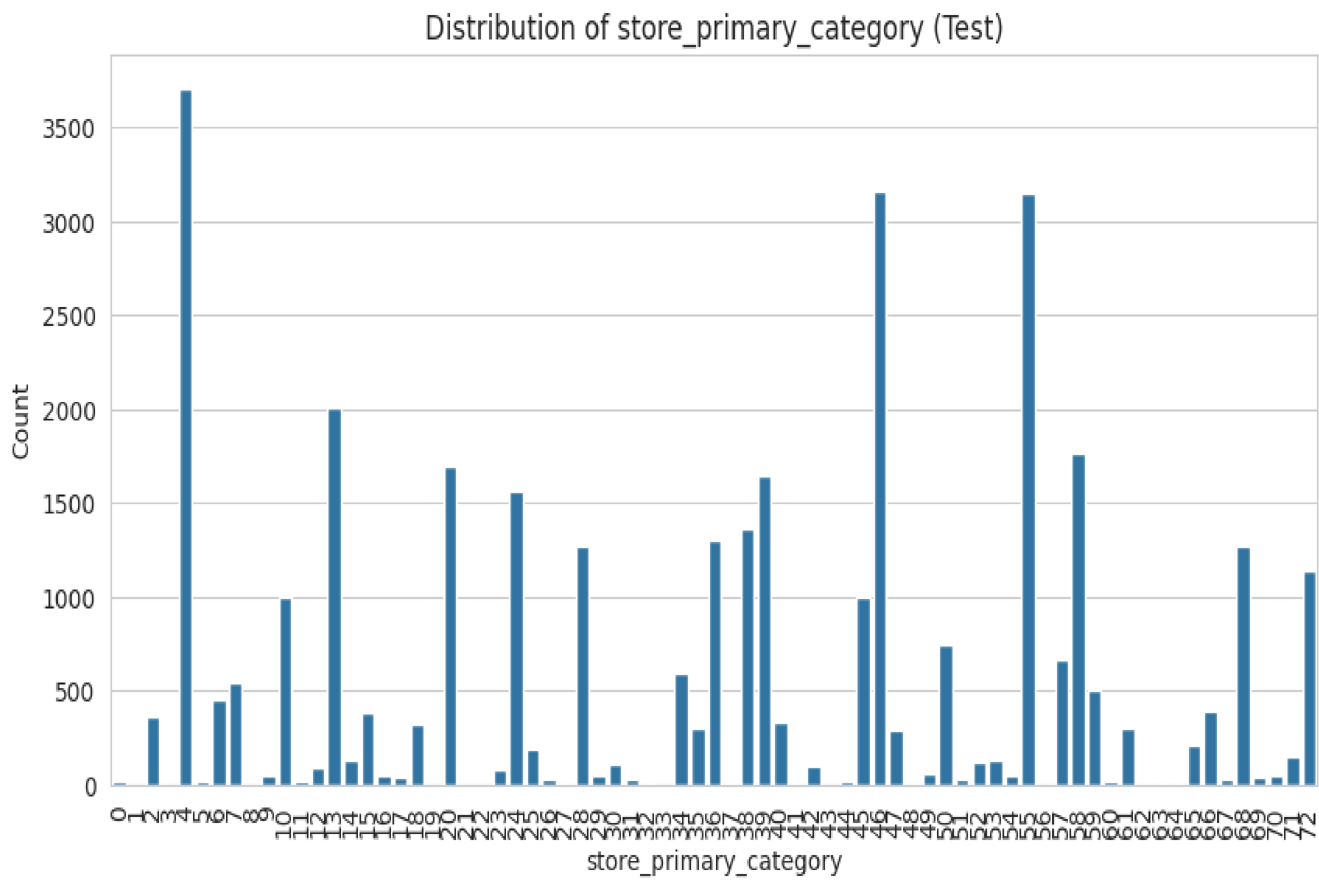
4.1.2

Check the distribution of categorical features

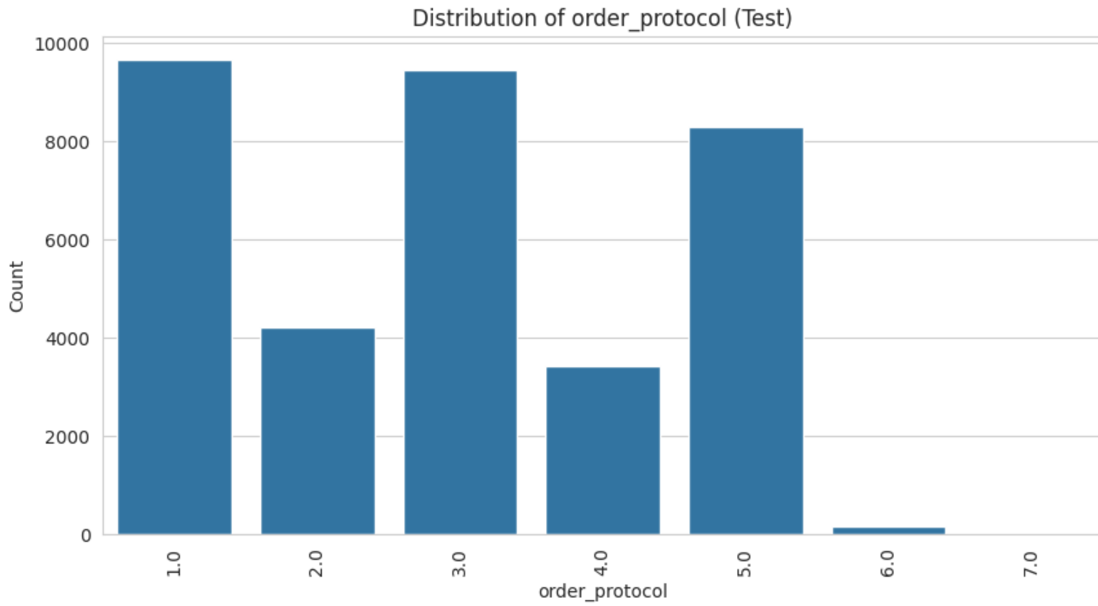
```
# Distribution of categorical columns
for col in categorical_cols_test:
    plt.figure(figsize=(10, 5))
    sns.countplot(data=X_test, x=col)
    plt.title(f'Distribution of {col} (Test)')
    plt.xlabel(col)
    plt.ylabel('Count')
    plt.xticks(rotation=90)
    plt.show()
```

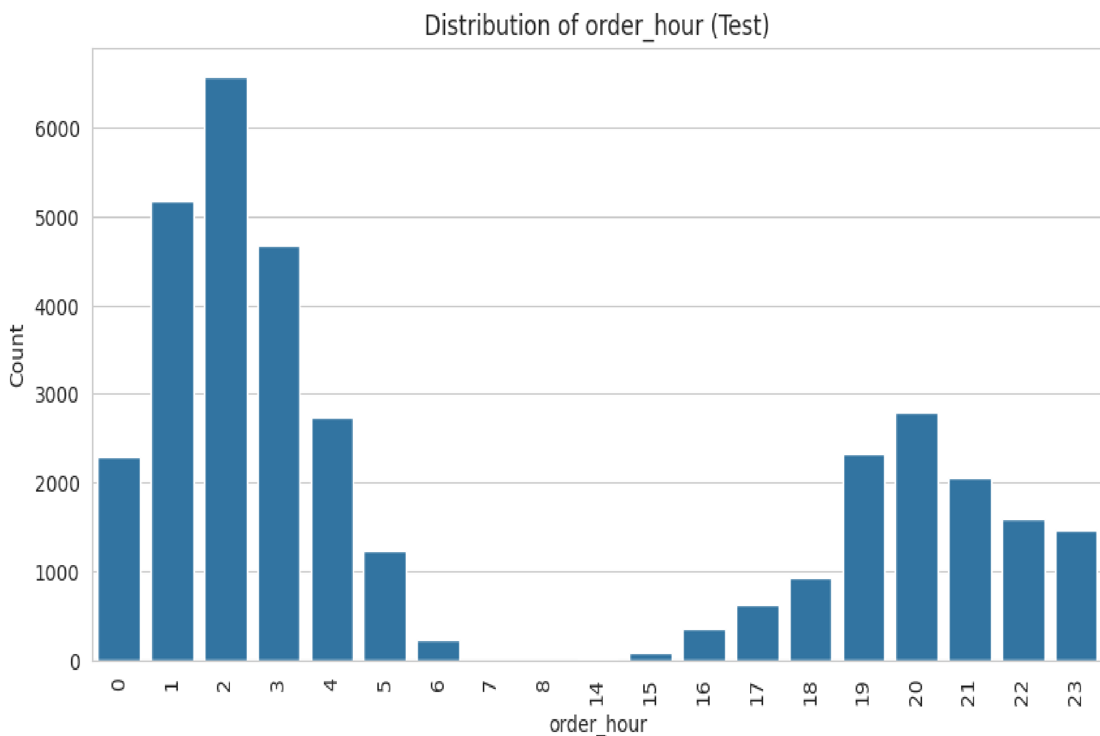
Syed Ismail N

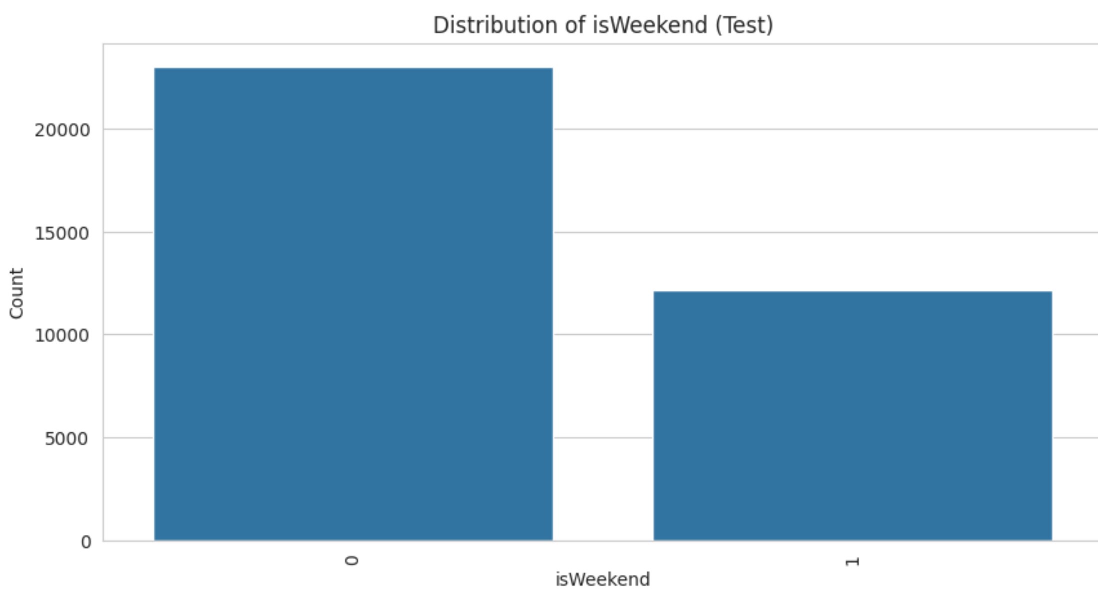




Syed Ismail N



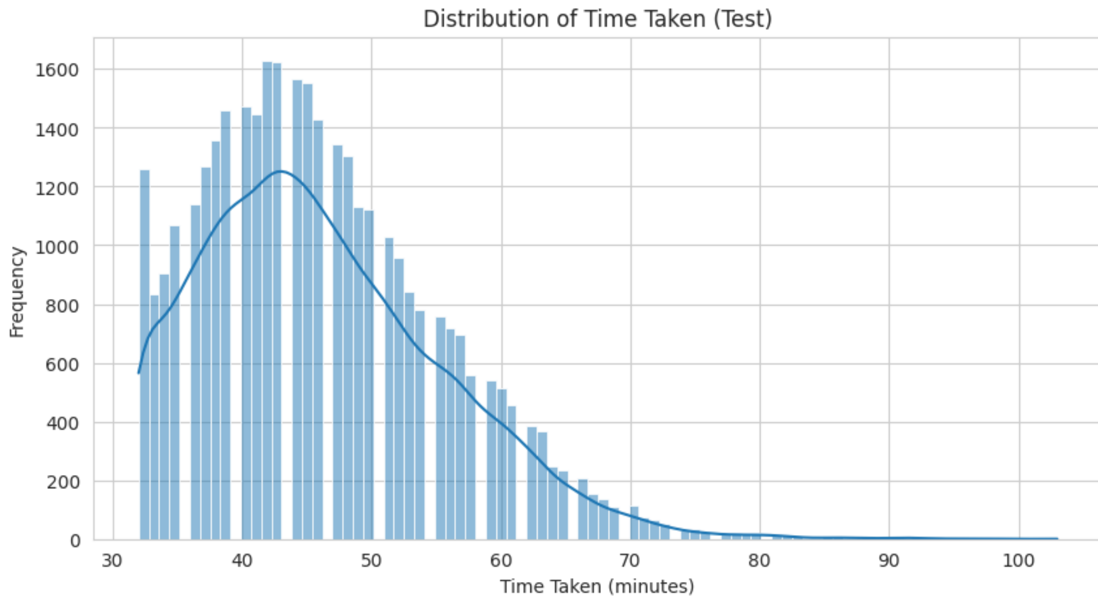




4.1.3

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

```
# Distribution of time_taken
plt.figure(figsize=(10, 5))
sns.histplot(y_test, kde=True)
plt.title('Distribution of Time Taken (Test)')
plt.xlabel('Time Taken (minutes)')
plt.ylabel('Frequency')
plt.show()
```

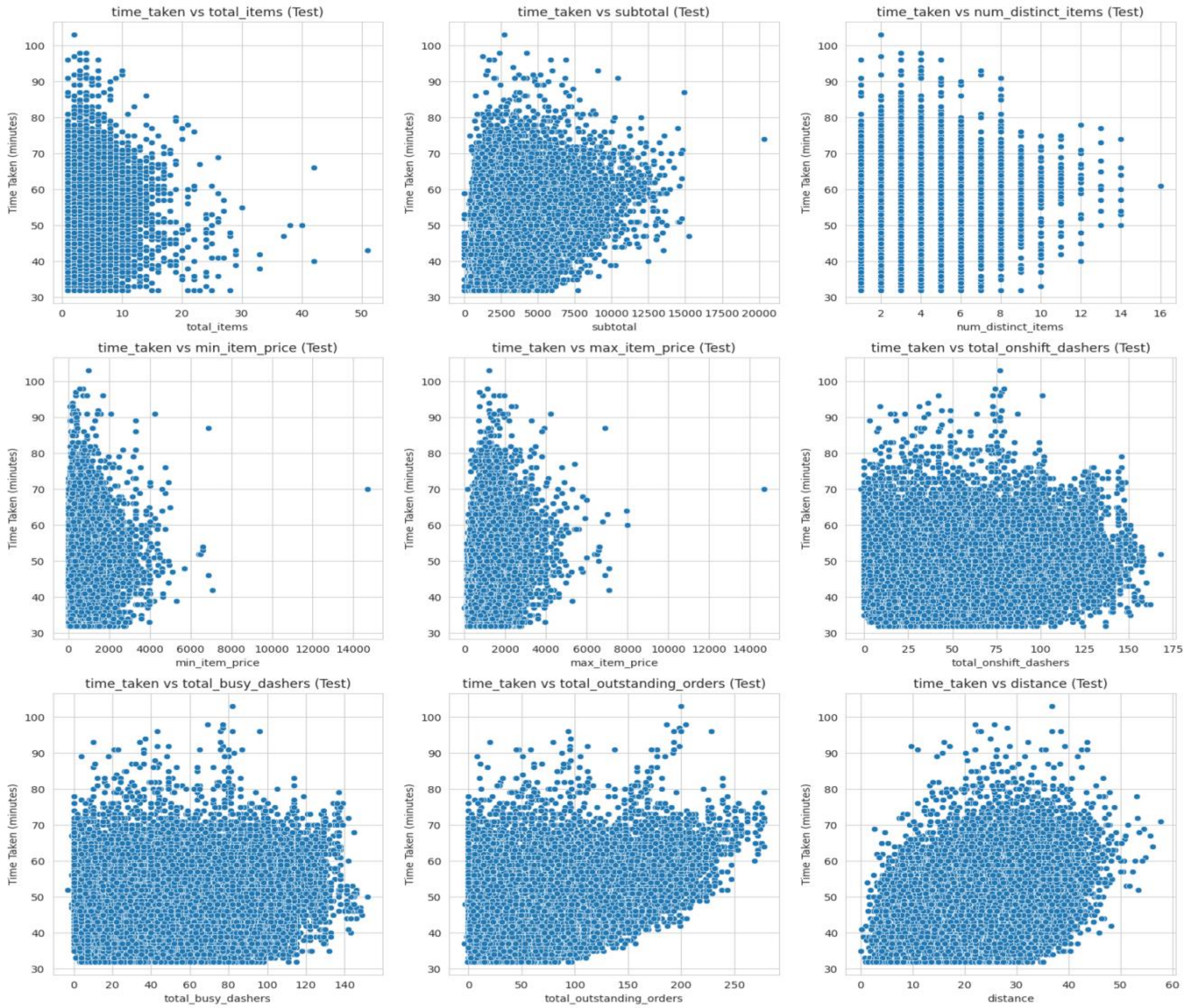


4.2 Relationships Between Features

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

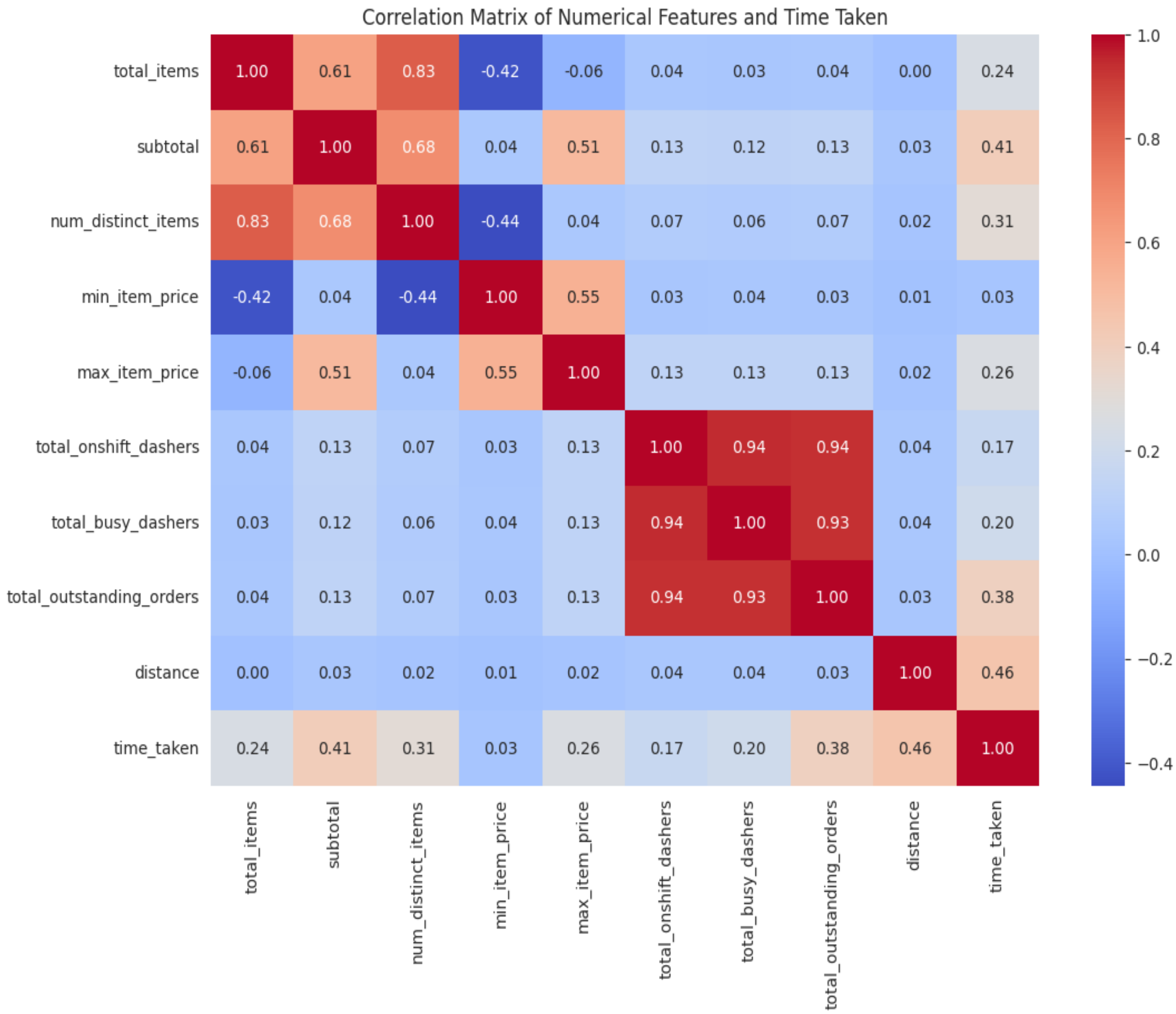
```
# Scatter plot to visualise the relationship between time_taken and all
numerical features

# Determine the number of plots and calculate grid dimensions
n_plots = len(numerical_cols_test)
n_cols = 3  # Number of columns in the grid
n_rows = (n_plots + n_cols - 1) // n_cols  # Calculate the number of rows
needed

plt.figure(figsize=(n_cols * 5, n_rows * 5))

for i, col in enumerate(numerical_cols_test):
    plt.subplot(n_rows, n_cols, i + 1)
    sns.scatterplot(x=col, y=y_test, data=X_test)
    plt.title(f'time_taken vs {col} (Test)')
    plt.xlabel(col)
    plt.ylabel('Time Taken (minutes)')

plt.tight_layout()
plt.show()
```

Syed Ismail N



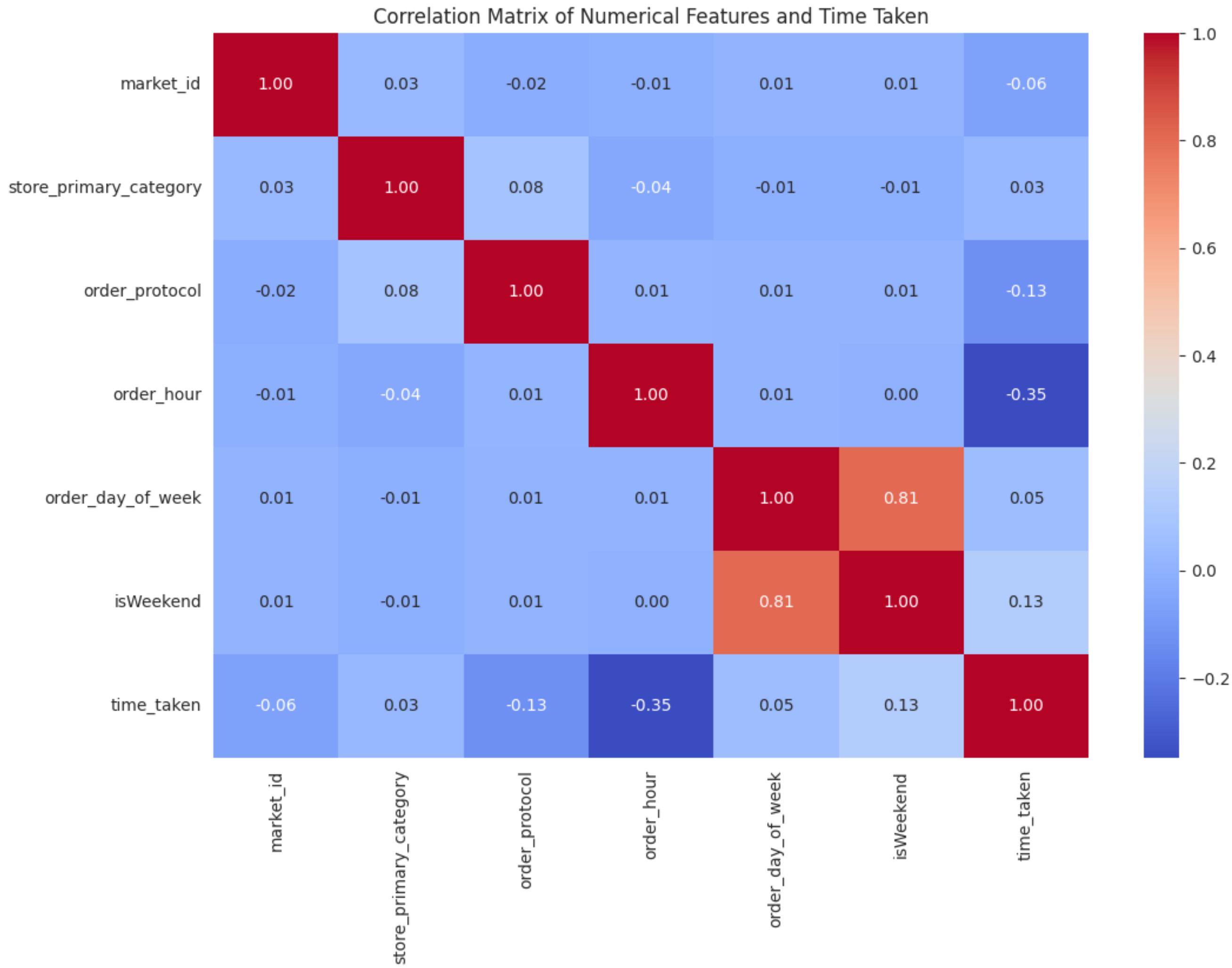
4.3 Drop the columns with weak correlations with the target variable

```
# Concatenate X_train and y_train to calculate correlations with the target
variable
df_num_test = pd.concat([X_test[numerical_cols_test], y_test], axis=1)

# Calculate the correlation matrix
num_test_correlation_matrix = df_num_test.corr()

# Plot the heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(num_test_correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Matrix of Numerical Features and Time Taken')
plt.show()
```




```python
# Concatenate X_train and y_train to calculate correlations with the target variable
df_cat_test = pd.concat([X_test[categorical_cols_test], y_test], axis=1)

# Calculate the correlation matrix
cat_test_correlation_matrix = df_cat_test.corr()

# Plot the heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(cat_test_correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features and Time Taken')
plt.show()
```

Syed Ismail N




```python
# Drop 3-5 weakly correlated columns from testing dataset
# Based on both heatmaps; min_item_price, order_day_of_week,
store_primary_category are weakly correlated columns (same as in training
dataset)
weak_corr_cols_test = ['min_item_price', 'order_day_of_week',
'store_primary_category']
X_test = X_test.drop(weak_corr_cols_test, axis=1)
```

5. Model Building [15 marks]

Import Necessary Libraries

```python
# Import libraries
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import statsmodels.api as sm
```

5.1 Feature Scaling [3 marks]

```
# Identify categorical columns in the current X_train DataFrame
categorical_cols = X_train.select_dtypes(include='category').columns

# Perform one-hot encoding on the identified categorical columns
# Keeping all columns for now to allow RFE to select the best features
X_train = pd.get_dummies(X_train, columns=categorical_cols,
drop_first=True).astype(int)
X_test = pd.get_dummies(X_test, columns=categorical_cols,
drop_first=True).astype(int)

display(X_train)
print(f"Shape of X_train after one-hot encoding: {X_train.shape}")
```

{"type":"dataframe","variable_name":"X_train"}

```
Shape of X_train after one-hot encoding: (140621, 38)
```

```
display(X_test)
print(f"Shape of X_test after one-hot encoding: {X_test.shape}")
```

{"type":"dataframe","variable_name":"X_test"}

```
Shape of X_test after one-hot encoding: (35156, 38)
```

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 140621 entries, 102712 to 121958
Data columns (total 38 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   total_items               140621 non-null  int64
 1   subtotal                  140621 non-null  int64
 2   num_distinct_items        140621 non-null  int64
 3   max_item_price            140621 non-null  int64
 4   total_onshift_dashers     140621 non-null  int64
 5   total_busy_dashers        140621 non-null  int64
 6   total_outstanding_orders  140621 non-null  int64
 7   distance                  140621 non-null  int64
 8   market_id_2.0             140621 non-null  int64
 9   market_id_3.0             140621 non-null  int64
 10  market_id_4.0             140621 non-null  int64
 11  market_id_5.0             140621 non-null  int64
 12  market_id_6.0             140621 non-null  int64
 13  order_protocol_2.0        140621 non-null  int64
 14  order_protocol_3.0        140621 non-null  int64
 15  order_protocol_4.0        140621 non-null  int64
 16  order_protocol_5.0        140621 non-null  int64
 17  order_protocol_6.0        140621 non-null  int64
 18  order_protocol_7.0        140621 non-null  int64
```

Syed Ismail N

```
 19  order_hour_1              140621 non-null  int64
 20  order_hour_2              140621 non-null  int64
 21  order_hour_3              140621 non-null  int64
 22  order_hour_4              140621 non-null  int64
 23  order_hour_5              140621 non-null  int64
 24  order_hour_6              140621 non-null  int64
 25  order_hour_7              140621 non-null  int64
 26  order_hour_8              140621 non-null  int64
 27  order_hour_14             140621 non-null  int64
 28  order_hour_15             140621 non-null  int64
 29  order_hour_16             140621 non-null  int64
 30  order_hour_17             140621 non-null  int64
 31  order_hour_18             140621 non-null  int64
 32  order_hour_19             140621 non-null  int64
 33  order_hour_20             140621 non-null  int64
 34  order_hour_21             140621 non-null  int64
 35  order_hour_22             140621 non-null  int64
 36  order_hour_23             140621 non-null  int64
 37  isWeekend_1               140621 non-null  int64
dtypes: int64(38)
memory usage: 45.9 MB
```

```
X_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 35156 entries, 50609 to 49931
Data columns (total 38 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   total_items               35156 non-null  int64
 1   subtotal                  35156 non-null  int64
 2   num_distinct_items        35156 non-null  int64
 3   max_item_price            35156 non-null  int64
 4   total_onshift_dashers     35156 non-null  int64
 5   total_busy_dashers        35156 non-null  int64
 6   total_outstanding_orders  35156 non-null  int64
 7   distance                  35156 non-null  int64
 8   market_id_2.0             35156 non-null  int64
 9   market_id_3.0             35156 non-null  int64
 10  market_id_4.0             35156 non-null  int64
 11  market_id_5.0             35156 non-null  int64
 12  market_id_6.0             35156 non-null  int64
 13  order_protocol_2.0        35156 non-null  int64
 14  order_protocol_3.0        35156 non-null  int64
 15  order_protocol_4.0        35156 non-null  int64
 16  order_protocol_5.0        35156 non-null  int64
 17  order_protocol_6.0        35156 non-null  int64
 18  order_protocol_7.0        35156 non-null  int64
 19  order_hour_1              35156 non-null  int64
 20  order_hour_2              35156 non-null  int64
```

```
 21  order_hour_3                35156 non-null  int64
 22  order_hour_4                35156 non-null  int64
 23  order_hour_5                35156 non-null  int64
 24  order_hour_6                35156 non-null  int64
 25  order_hour_7                35156 non-null  int64
 26  order_hour_8                35156 non-null  int64
 27  order_hour_14               35156 non-null  int64
 28  order_hour_15               35156 non-null  int64
 29  order_hour_16               35156 non-null  int64
 30  order_hour_17               35156 non-null  int64
 31  order_hour_18               35156 non-null  int64
 32  order_hour_19               35156 non-null  int64
 33  order_hour_20               35156 non-null  int64
 34  order_hour_21               35156 non-null  int64
 35  order_hour_22               35156 non-null  int64
 36  order_hour_23               35156 non-null  int64
 37  isWeekend_1                 35156 non-null  int64
dtypes: int64(38)
memory usage: 11.5 MB
```

```python
from sklearn.preprocessing import MinMaxScaler

# Apply MinMaxScaler to all selected columns
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert scaled numpy arrays back to pandas DataFrames, preserving column
names and index
X_train = pd.DataFrame(X_train_scaled, columns=X_train.columns,
index=X_train.index)
X_test = pd.DataFrame(X_test_scaled, columns=X_test.columns,
index=X_test.index)

display(X_train.head())
print(f"Number of columns in X_train after selection and scaling:
{X_train.shape[1]}")
```

{"type":"dataframe"}

```
Number of columns in X_train after selection and scaling: 38
```

```python
display(X_train.describe())
```

{"type":"dataframe"}

```python
display(X_test.describe())
```

{"type":"dataframe"}

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 using statsmodels
# Add a constant to the training features
X_train_sm = sm.add_constant(X_train)
lr = sm.OLS(y_train, X_train_sm)

# Train the model using the training data
lr_model = lr.fit()

# Display the model summary
print("\nModel Summary:")
print(lr_model.summary())
```

```
Model Summary:
                            OLS Regression Results
==============================================================================
=
Dep. Variable:             time_taken   R-squared:                       0.891
Model:                            OLS   Adj. R-squared:                  0.890
Method:                 Least Squares   F-statistic:                 3.009e+04
Date:                Tue, 04 Nov 2025   Prob (F-statistic):               0.00
Time:                        14:53:42   Log-Likelihood:            3.5489e+05
No. Observations:              140621   AIC:                         7.099e+05
Df Residuals:                  140582   BIC:                         7.102e+05
Df Model:                          38
Covariance Type:            nonrobust
==============================================================================
===============
                               coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
---------------
const                       37.3759      0.050    749.866      0.000
37.278      37.474
total_items                  1.4670      0.067     21.917      0.000
1.336       1.598
subtotal                     9.4077      0.073    128.783      0.000
9.265       9.551
num_distinct_items           0.4634      0.043     10.825      0.000
0.379       0.547
max_item_price               1.3822      0.064     21.597      0.000
1.257       1.508
total_onshift_dashers      -49.7992      0.122   -409.001      0.000      -
```

```
50.038     -49.561
total_busy_dashers          -17.0583       0.121   -141.236       0.000     -
17.295     -16.822
total_outstanding_orders     67.9800       0.104    654.689       0.000
67.776      68.183
distance                     22.5254       0.044    516.976       0.000
22.440      22.611
market_id_2.0                -4.8436       0.027   -176.918       0.000      -
4.897      -4.790
market_id_3.0                -4.0543       0.029   -138.475       0.000      -
4.112      -3.997
market_id_4.0                -4.1620       0.028   -150.135       0.000      -
4.216      -4.108
market_id_5.0                -3.3394       0.031   -106.175       0.000      -
3.401      -3.278
market_id_6.0                -2.8156       0.136    -20.719       0.000      -
3.082      -2.549
order_protocol_2.0           -0.7258       0.029    -25.376       0.000      -
0.782      -0.670
order_protocol_3.0           -1.4325       0.023    -62.890       0.000      -
1.477      -1.388
order_protocol_4.0           -1.8500       0.031    -59.020       0.000      -
1.911      -1.789
order_protocol_5.0           -2.7764       0.023   -118.390       0.000      -
2.822      -2.730
order_protocol_6.0           -1.3836       0.132    -10.488       0.000      -
1.642      -1.125
order_protocol_7.0           -1.3330       0.712     -1.872       0.061      -
2.729       0.063
order_hour_1                 -0.3786       0.039     -9.678       0.000      -
0.455      -0.302
order_hour_2                 -0.8528       0.040    -21.284       0.000      -
0.931      -0.774
order_hour_3                 -1.0903       0.041    -26.277       0.000      -
1.172      -1.009
order_hour_4                 -2.0741       0.044    -47.194       0.000      -
2.160      -1.988
order_hour_5                 -2.0125       0.054    -37.376       0.000      -
2.118      -1.907
order_hour_6                 -1.8396       0.102    -18.063       0.000      -
2.039      -1.640
order_hour_7                 -2.3500       1.233     -1.906       0.057      -
4.767       0.067
order_hour_8                 -0.6318       2.135     -0.296       0.767      -
4.817       3.553
order_hour_14                -2.0429       0.552     -3.698       0.000      -
3.126      -0.960
order_hour_15                -2.2354       0.153    -14.628       0.000      -
2.535      -1.936
order_hour_16                -2.7153       0.083    -32.632       0.000      -
```

```
2.878      -2.552
order_hour_17                 -3.1690      0.070    -45.452      0.000      -
3.306      -3.032
order_hour_18                 -3.8816      0.060    -64.747      0.000      -
3.999      -3.764
order_hour_19                 -4.6399      0.044   -104.541      0.000      -
4.727      -4.553
order_hour_20                 -4.9100      0.043   -114.065      0.000      -
4.994      -4.826
order_hour_21                 -4.6845      0.046   -101.565      0.000      -
4.775      -4.594
order_hour_22                 -4.6339      0.050    -93.337      0.000      -
4.731      -4.537
order_hour_23                 -4.5487      0.051    -89.695      0.000      -
4.648      -4.449
isWeekend_1                    1.4821      0.017     85.182      0.000
1.448       1.516
==============================================================================
=
Omnibus:                    46509.522   Durbin-Watson:
1.992
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
233624.701
Skew:                           1.526   Prob(JB):
0.00
Kurtosis:                       8.528   Cond. No.
459.
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

```
lr_model.params
```

```
const                        37.375877
total_items                   1.466953
subtotal                      9.407730
num_distinct_items            0.463358
max_item_price                1.382201
total_onshift_dashers       -49.799186
total_busy_dashers          -17.058253
total_outstanding_orders     67.979956
distance                     22.525442
market_id_2.0                -4.843635
market_id_3.0                -4.054339
market_id_4.0                -4.161970
market_id_5.0                -3.339441
market_id_6.0                -2.815591
```

```
order_protocol_2.0          -0.725810
order_protocol_3.0          -1.432535
order_protocol_4.0          -1.850022
order_protocol_5.0          -2.776395
order_protocol_6.0          -1.383617
order_protocol_7.0          -1.333000
order_hour_1                -0.378621
order_hour_2                -0.852806
order_hour_3                -1.090344
order_hour_4                -2.074100
order_hour_5                -2.012528
order_hour_6                -1.839619
order_hour_7                -2.349997
order_hour_8                -0.631785
order_hour_14               -2.042875
order_hour_15               -2.235352
order_hour_16               -2.715280
order_hour_17               -3.168977
order_hour_18               -3.881605
order_hour_19               -4.639887
order_hour_20               -4.909970
order_hour_21               -4.684456
order_hour_22               -4.633922
order_hour_23               -4.548728
isWeekend_1                  1.482118
dtype: float64
```

```
y_train_pred = lr_model.predict(X_train_sm)

# Plotting the regression plot for the training data using seaborn.regplot
plt.figure(figsize=(8, 6))
sns.regplot(x=y_train, y=y_train_pred, scatter_kws={'alpha': 0.3},
line_kws={'color': 'red'}) # Use regplot for scatter and fitted line
plt.title('Regression Plot: Actual vs Predicted Delivery Time (Training
Data)')
plt.xlabel('Actual Time Taken (minutes)')
plt.ylabel('Predicted Time Taken (minutes)')
plt.show()
```


```python
print(f"Root Mean Squared Error (RMSE): {np.sqrt(mean_squared_error(y_train, y_train_pred)):.2f}")
print(f"R-Squared Error (R2) : {r2_score(y_train, y_train_pred):.2f}")
```

```
Root Mean Squared Error (RMSE): 3.02
R-Squared Error (R2) : 0.89
```

Explanation for the appearance of the regression plot:

The regression plot shows the actual delivery times on the x-axis and the predicted delivery times on the y-axis. It appears as dense lines or areas instead of individual scattered dots due to the very large number of data points in the training set (over 140,000). When a large number of points are plotted, especially when they are clustered or follow a strong linear trend, they tend to overlap significantly, creating the visual effect of solid lines.

The `scatter_kws={'alpha': 0.3}` argument in `seaborn.regplot` adds transparency to the points, which helps to indicate areas of higher data density. However, with such a substantial dataset, even with transparency, the overlapping points will form what looks like lines.

This plot is intended to show the overall relationship between actual and predicted values and how closely they follow the ideal regression line (where actual equals predicted). It is different from a residual plot, which specifically visualizes the difference between actual and predicted values (the errors or residuals).

```
# Make predictions using the statsmodels results
# Add a constant to the test features
X_test_sm = sm.add_constant(X_test)
y_pred = lr_model.predict(X_test_sm)

# Plotting the regression plot for the test data using seaborn.regplot
plt.figure(figsize=(8, 6))
sns.regplot(x=y_test, y=y_pred, scatter_kws={'alpha': 0.3},
line_kws={'color': 'red'}) # Use regplot for scatter and fitted line
plt.title('Regression Plot: Actual vs Predicted Delivery Time (Test Data)')
plt.xlabel('Actual Time Taken (minutes)')
plt.ylabel('Predicted Time Taken (minutes)')
plt.show()
```


```
# Find results for evaluation metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R2): {r2:.2f}")
```

```
Root Mean Squared Error (RMSE): 3.05
R-squared (R2): 0.89
```

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.

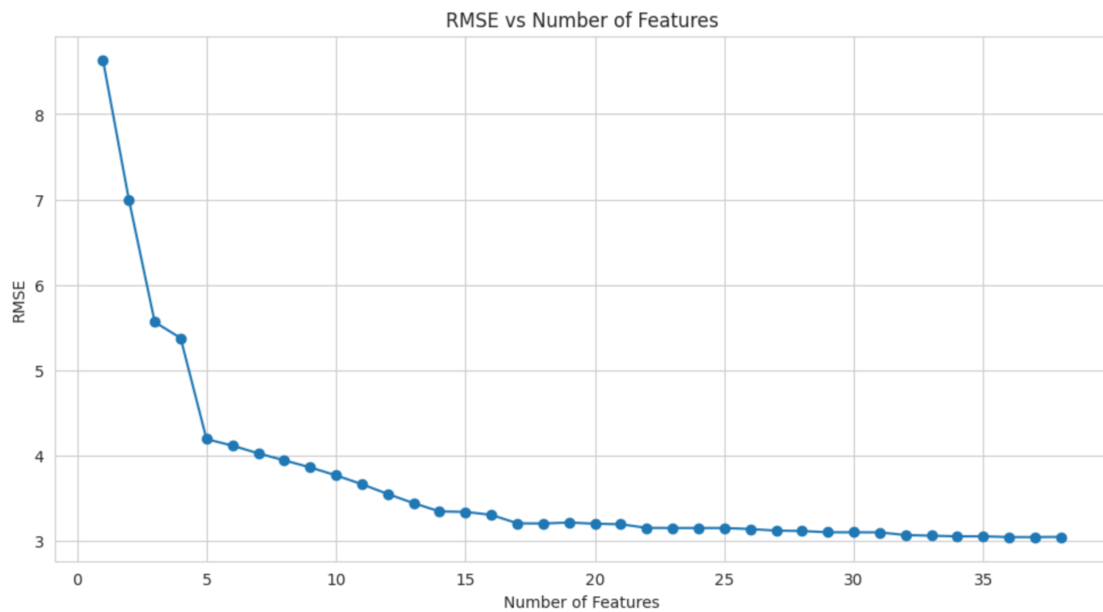
After analysing the results of these iterations, we select the one that has a good balance between performance and number of features.

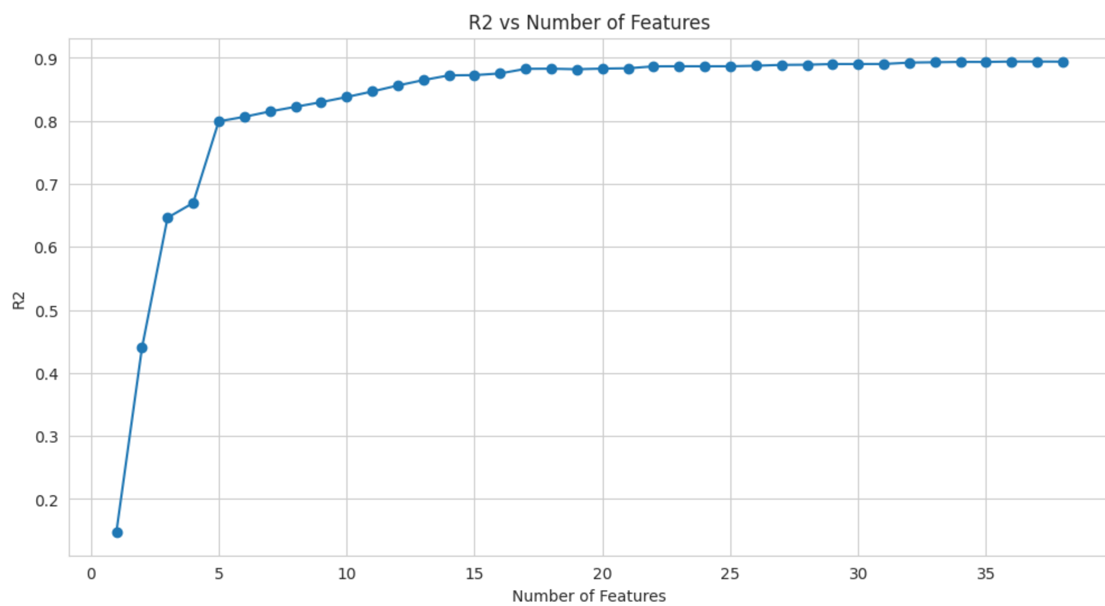
```python
# Create a Linear Regression model to use with RFE
lm = LinearRegression()

# Store results for analysis
results = []

# Iterate through different numbers of features to select (from all features down to 1)
# X_train.shape[1] is the total number of features after one-hot encoding
for n_features in range(1, X_train.shape[1] + 1):
    rfe = RFE(lm, n_features_to_select=n_features)

    # Fit RFE to the training data
    rfe = rfe.fit(X_train, y_train.ravel())

    # Get the list of selected features
    selected_rfe_features = X_train.columns[rfe.support_]

    # Build a new OLS model with the selected features
    X_train_rfe = X_train[selected_rfe_features]
    X_train_rfe_sm = sm.add_constant(X_train_rfe)

    lr_model_rfe = sm.OLS(y_train, X_train_rfe_sm).fit()

    # Make predictions on the test set using the RFE selected features
    X_test_rfe = X_test[selected_rfe_features]
    X_test_rfe_sm = sm.add_constant(X_test_rfe)
    y_pred_rfe = lr_model_rfe.predict(X_test_rfe_sm)

    # Evaluate the model with RFE selected features
```

```
    rmse_rfe = np.sqrt(mean_squared_error(y_test, y_pred_rfe))
    r2_rfe = r2_score(y_test, y_pred_rfe)

    # Store the results
    results.append({'n_features': n_features, 'rmse': rmse_rfe, 'r2': r2_rfe,
'selected_features': selected_rfe_features.tolist()})

# Convert results to a DataFrame for easier analysis
results_df = pd.DataFrame(results)

# Display the results
print("\nRFE Iteration Results:")
display(results_df.sort_values(by='rmse'))

# Optional: Plot RMSE and R2 vs number of features to visualize the trade-off
plt.figure(figsize=(12, 6))
plt.plot(results_df['n_features'], results_df['rmse'], marker='o')
plt.title('RMSE vs Number of Features')
plt.xlabel('Number of Features')
plt.ylabel('RMSE')
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(results_df['n_features'], results_df['r2'], marker='o')
plt.title('R2 vs Number of Features')
plt.xlabel('Number of Features')
plt.ylabel('R2')
plt.show()
```

```
RFE Iteration Results:
```

Syed Ismail N





Based on the RFE iteration results, we can observe the trade-off between the number of features and the model's performance (RMSE and R2). The performance generally improves as more features are added, but the gains become marginal after a certain point.

Analyzing the results from the table, a good balance between performance and the number of features is achieved when selecting around **35 to 37 features**. The improvement in RMSE and R2 is minimal beyond this point.

For instance, if we select **37 features**, the results are:

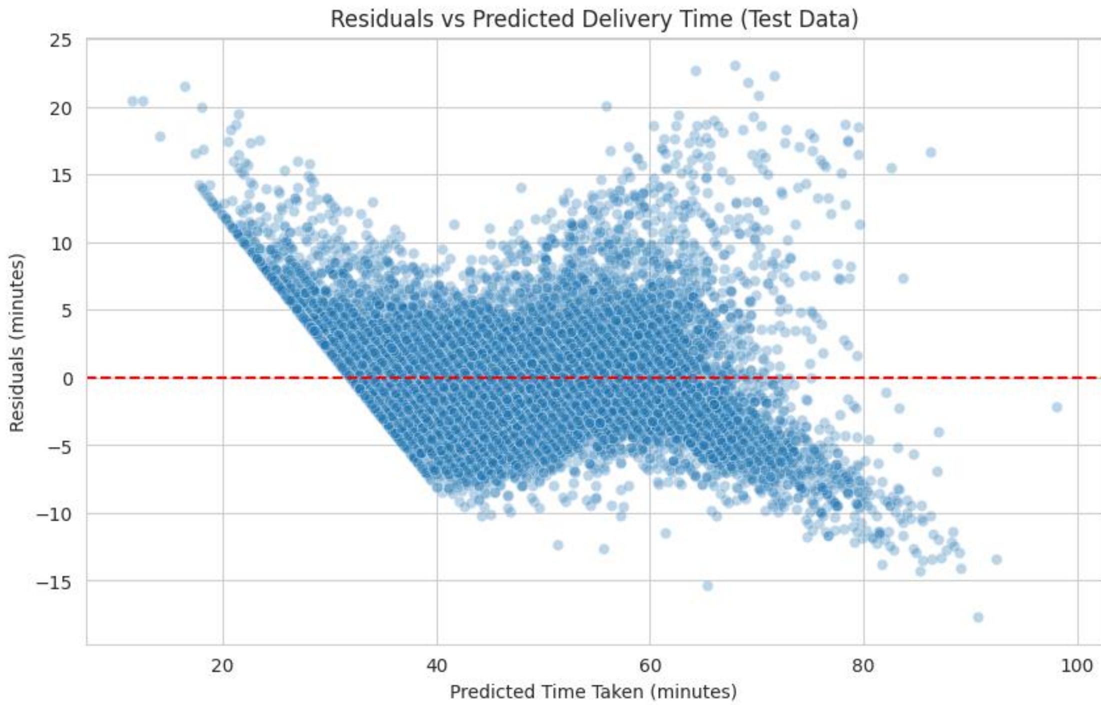
- **RMSE:** 3.04
- **R2:** 0.89

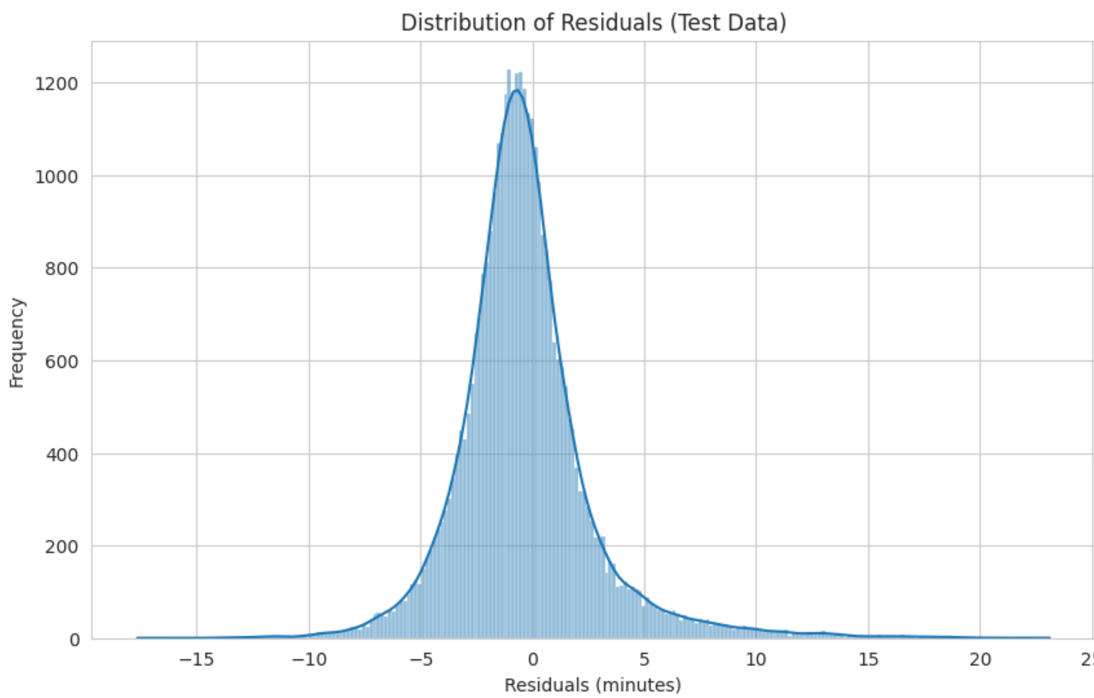
This suggests that a model with 37 features captures most of the relevant information for predicting delivery time while being slightly simpler than the full model with 38 features.

6. Results and Inference [5 marks]

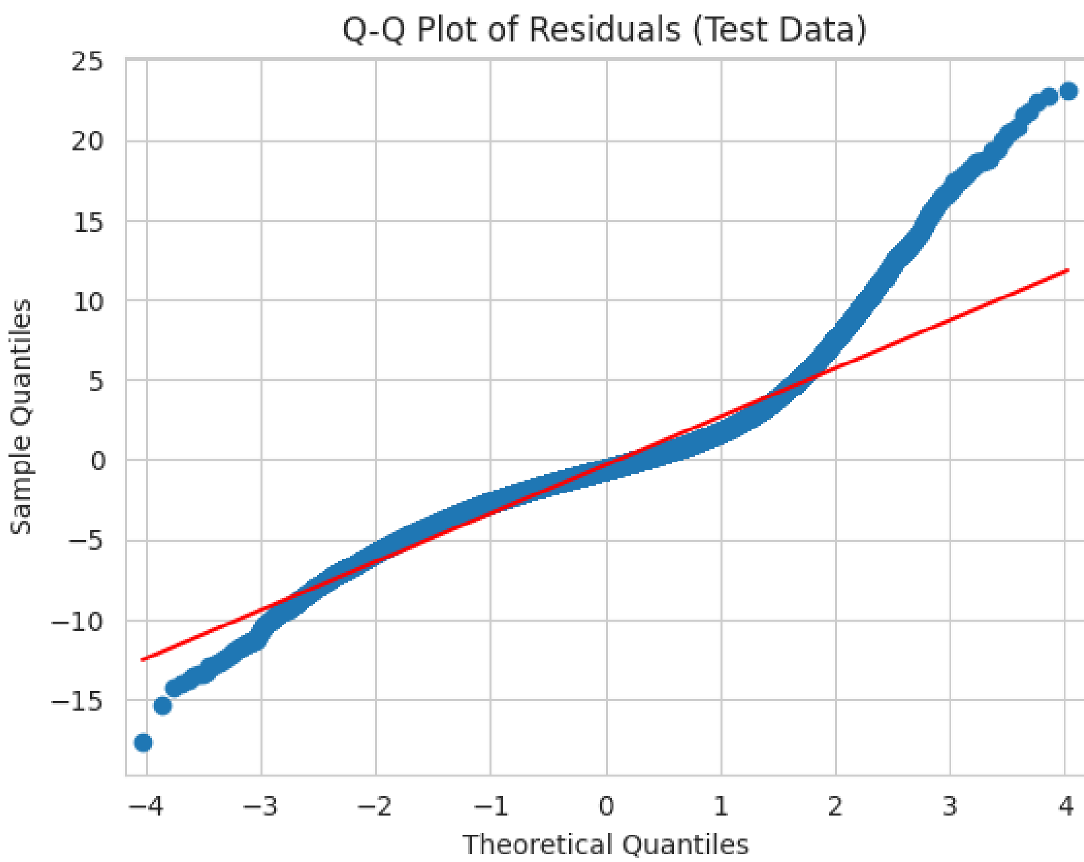
6.1 Perform Residual Analysis [3 marks]

```
# Calculate residuals
residuals = y_test - y_pred_rfe

# Plot residuals vs predicted values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred_rfe, y=residuals, alpha=0.3)
plt.axhline(y=0, color='r', linestyle='--')
plt.title('Residuals vs Predicted Delivery Time (Test Data)')
plt.xlabel('Predicted Time Taken (minutes)')
plt.ylabel('Residuals (minutes)')
plt.show()

# Plot distribution of residuals
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.title('Distribution of Residuals (Test Data)')
plt.xlabel('Residuals (minutes)')
plt.ylabel('Frequency')
plt.show()

# Q-Q plot
sm.qqplot(residuals, line='s')
plt.title('Q-Q Plot of Residuals (Test Data)')
plt.show()
```


Residuals vs Predicted Delivery Time (Test Data)
Residuals (minutes)
Predicted Time Taken (minutes)
25
20
15
10
5
0
−5
−10
−15
20
40
60
80
100


Distribution of Residuals (Test Data)
Frequency
Residuals (minutes)
1200
1000
800
600
400
200
0
−15
−10
−5
0
5
10
15
20
25

Syed Ismail N



[Your inferences here:]

Based on the residual analysis plots:

- **Residuals vs Predicted plot:** The scatter plot shows a general band of residuals around zero, but there appears to be a "fanning out" effect, where the spread of residuals increases as the predicted delivery time increases. This suggests that the assumption of homoscedasticity (constant variance of errors) might be violated. The variance of the errors is not constant across all predicted values.
- **Distribution of Residuals histogram:** The histogram of residuals is somewhat bell-shaped and centered around zero, indicating that the residuals are approximately normally distributed. However, there might be slight skewness or heavier tails than a perfect normal distribution.
- **Q-Q plot:** The points on the Q-Q plot generally follow the straight line, particularly in the center. There are some deviations at the tails, suggesting that the residuals might not be perfectly normally distributed, which is consistent with the histogram.

Inference: The residual analysis suggests that while the model's errors are roughly normally distributed and centered around zero, the assumption of homoscedasticity is likely violated due to the increasing spread of residuals with higher predicted values. This indicates that the model's predictions might be less reliable for longer delivery times.

Syed Ismail N

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.

```
# Analyze the effect of a unit change in a feature, say 'total_items'

# To understand the effect of a unit change in a feature on the original
scale,
# we need to use the unscaled coefficients.
# We will train a separate OLS model using the original, unscaled data with
the RFE selected features.

# First, get the original unscaled data. We need to reload the original data
to ensure 'time_taken' is present.
original_df = pd.read_csv('porter_data_1.csv')

# Convert 'created_at' and 'actual_delivery_time' columns to datetime format
in the original_df
original_df['created_at'] = pd.to_datetime(original_df['created_at'])
original_df['actual_delivery_time'] =
pd.to_datetime(original_df['actual_delivery_time'])

# Calculate time taken in minutes in the original_df
original_df['time_taken'] = (original_df['actual_delivery_time'] -
original_df['created_at']).dt.total_seconds() / 60

# Extract order_hour, order_day_of_week, and isWeekend from original_df
original_df['order_hour'] =
original_df['created_at'].dt.hour.astype('category')
original_df['order_day_of_week'] =
original_df['created_at'].dt.dayofweek.astype('category')
original_df['isWeekend'] = original_df['order_day_of_week'].apply(lambda x: 1
if x >= 5 else 0).astype('category')

# Drop the original timestamp columns from original_df
original_df = original_df.drop(['created_at', 'actual_delivery_time'],
axis=1)

# Convert categorical features to category type in original_df
original_categorical_cols = ['market_id', 'store_primary_category',
'order_protocol', 'order_hour', 'order_day_of_week', 'isWeekend']
for col in original_categorical_cols:
    original_df[col] = original_df[col].astype('category')

# Perform one-hot encoding on the original_df
df_encoded_unscaled = pd.get_dummies(original_df,
columns=original_categorical_cols, drop_first=True).astype(int)
```

```
# Separate features (X) and target (y) from the unscaled encoded dataframe
y_unscaled = df_encoded_unscaled.pop('time_taken')
X_unscaled = df_encoded_unscaled

# Split the unscaled data, ensuring the same split as before (using the same
random_state)
# We will use the indices from the X_train and X_test DataFrames (which
retain their original indices)
X_train_unscaled = X_unscaled.loc[X_train.index]
X_test_unscaled = X_unscaled.loc[X_test.index]
y_train_unscaled = y_unscaled.loc[y_train.index]
y_test_unscaled = y_unscaled.loc[y_test.index]

# Get the list of selected features from the RFE results (assuming 37
features were selected as discussed)
# You can choose a different number of features based on your analysis of the
RFE results_df
# For example, to get the selected features for 37 features:
selected_rfe_features = results_df[results_df['n_features'] ==
37]['selected_features'].iloc[0]


# Filter the unscaled dataframes to keep only the RFE selected features
X_train_unscaled_rfe = X_train_unscaled[selected_rfe_features]
X_test_unscaled_rfe = X_test_unscaled[selected_rfe_features]

# Add a constant to the unscaled RFE features for statsmodels
X_train_unscaled_rfe_sm = sm.add_constant(X_train_unscaled_rfe)

# Build and fit the OLS model on unscaled data
lr_model_unscaled = sm.OLS(y_train_unscaled, X_train_unscaled_rfe_sm).fit()

print("\nModel Summary with Unscaled Features (RFE Selected):")
print(lr_model_unscaled.summary())
```

```
Model Summary with Unscaled Features (RFE Selected):
                            OLS Regression Results
==============================================================================
=
Dep. Variable:             time_taken   R-squared:
0.905
Model:                            OLS   Adj. R-squared:
0.905
Method:                 Least Squares   F-statistic:
3.604e+04
Date:                Tue, 04 Nov 2025   Prob (F-statistic):
0.00
```

```
Time:                        15:44:27   Log-Likelihood:            -
3.4820e+05
No. Observations:              140621   AIC:
6.965e+05
Df Residuals:                  140583   BIC:
6.969e+05
Df Model:                          37
Covariance Type:            nonrobust
==============================================================================
===============
                                 coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
---------------
const                         37.8211      0.040    934.649      0.000
37.742      37.900
total_items                   -0.0378      0.004     -8.581      0.000      -
0.046      -0.029
subtotal                       0.0012   7.85e-06    155.930      0.000
0.001       0.001
num_distinct_items             0.5199      0.008     61.370      0.000
0.503       0.537
max_item_price                 0.0006   1.89e-05     34.039      0.000
0.001       0.001
total_onshift_dashers         -0.3467      0.001   -438.601      0.000      -
0.348      -0.345
total_busy_dashers            -0.1303      0.001   -160.063      0.000      -
0.132      -0.129
total_outstanding_orders       0.3421      0.000    718.890      0.000
0.341       0.343
distance                       0.4844      0.001    549.242      0.000
0.483       0.486
market_id_2.0                 -4.6094      0.026   -177.851      0.000      -
4.660      -4.559
market_id_3.0                 -4.2883      0.028   -153.679      0.000      -
4.343      -4.234
market_id_4.0                 -3.7727      0.026   -143.736      0.000      -
3.824      -3.721
market_id_5.0                 -3.5076      0.030   -116.973      0.000      -
3.566      -3.449
market_id_6.0                 -2.7754      0.130    -21.420      0.000      -
3.029      -2.521
order_protocol_2.0            -0.6963      0.027    -25.543      0.000      -
0.750      -0.643
order_protocol_3.0            -1.4100      0.022    -64.911      0.000      -
1.453      -1.367
order_protocol_4.0            -1.8535      0.030    -62.505      0.000      -
1.912      -1.795
order_protocol_5.0            -2.7014      0.022   -120.798      0.000      -
2.745      -2.658
```

```
order_protocol_6.0          -1.5302      0.126    -12.169      0.000      -
1.777      -1.284
order_protocol_7.0          -1.7494      0.679     -2.576      0.010      -
3.081      -0.418
order_hour_2                -0.3588      0.026    -13.769      0.000      -
0.410      -0.308
order_hour_3                -1.0181      0.028    -35.859      0.000      -
1.074      -0.962
order_hour_4                -1.7170      0.033    -52.020      0.000      -
1.782      -1.652
order_hour_5                -2.0885      0.045    -45.918      0.000      -
2.178      -1.999
order_hour_6                -1.9864      0.094    -21.039      0.000      -
2.171      -1.801
order_hour_7                -2.4840      1.176     -2.113      0.035      -
4.788      -0.180
order_hour_8                -1.5151      2.036     -0.744      0.457      -
5.505       2.475
order_hour_14               -2.2938      0.526     -4.359      0.000      -
3.325      -1.262
order_hour_15               -2.6228      0.144    -18.224      0.000      -
2.905      -2.341
order_hour_16               -3.1672      0.076    -41.582      0.000      -
3.317      -3.018
order_hour_17               -3.5744      0.063    -57.140      0.000      -
3.697      -3.452
order_hour_18               -4.2781      0.052    -81.817      0.000      -
4.381      -4.176
order_hour_19               -4.6996      0.034   -136.746      0.000      -
4.767      -4.632
order_hour_20               -4.8156      0.033   -148.163      0.000      -
4.879      -4.752
order_hour_21               -4.7419      0.037   -129.454      0.000      -
4.814      -4.670
order_hour_22               -4.7936      0.041   -116.994      0.000      -
4.874      -4.713
order_hour_23               -4.7583      0.042   -112.580      0.000      -
4.841      -4.675
isWeekend_1                  1.1444      0.017     68.970      0.000
1.112       1.177
==============================================================================
=
Omnibus:                    50023.252   Durbin-Watson:
1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
306210.162
Skew:                           1.585   Prob(JB):
0.00
Kurtosis:                       9.497   Cond. No.
9.16e+05
```

```
================================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
[2] The condition number is large, 9.16e+05. This might indicate that there
are
strong multicollinearity or other numerical problems.
```

```
# Compare the scaled vs unscaled features used in the final model

# Get the scaled coefficients from the RFE model
scaled_coefficients = pd.DataFrame(lr_model_rfe.params, columns=['Scaled
Coefficient'])
scaled_coefficients = scaled_coefficients.drop('const') # Drop the constant
term for interpretation

print("Scaled Coefficients:")
display(scaled_coefficients)

# To compare with unscaled coefficients, we need the model trained on
unscaled data
# with the same features selected by RFE.
# We already trained this model in the previous cell (dMHN7r-x-Lp5).

# Get the unscaled coefficients from the unscaled OLS model
unscaled_coefficients = pd.DataFrame(lr_model_unscaled.params,
columns=['Unscaled Coefficient'])
unscaled_coefficients = unscaled_coefficients.drop('const') # Drop the
constant term

print("\nUnscaled Coefficients (RFE Selected Features):")
display(unscaled_coefficients)

# Combine scaled and unscaled coefficients for comparison
combined_coefficients = scaled_coefficients.join(unscaled_coefficients)
print("\nComparison of Scaled and Unscaled Coefficients:")
display(combined_coefficients)
```

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.

```
unscaled_coefficient_total_items = lr_model_unscaled.params['total_items']
print(f"\nEffect of a unit change in original 'total_items' on 'time_taken':
{unscaled_coefficient_total_items:.4f} minutes")
```

```
Effect of a unit change in original 'total_items' on 'time_taken': -0.0378
minutes
```

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.

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: Yes, the dataset initially contained `market_id`, `store_primary_category`, and `order_protocol` as categorical variables. During feature engineering, `order_hour`, `order_day_of_week`, and `isWeekend` were also created and treated as categorical features.

From the correlation analysis, `order_hour` showed strongest relationship with `time_taken` (correlation of -0.35). Other features like order_protocol (-0.14) and isWeekend (0.14) also showed a mild correlation. The analysis determined that store_primary_category and order_day_of_week had very weak correlations and were dropped from the model. This suggests that the time of day an order is placed, the market, and the protocol used have a measurable influence on the delivery time, while the specific day of the week (beyond it being a weekend or not) and the restaurant category were less impactful.

Question 2. [1 marks]

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

Answer: During the splitting of the data into training and test sets, `test_size = 0.2` refers to the proportion of the dataset that will be allocated to the test set. In this case, 20% of the data will be used for testing the model, and the remaining 80% will be used for training the model. This split is a common practice to evaluate the model's performance on unseen data.

Question 3. [1 marks]

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

Answer: Looking at the "Correlation Matrix of Numerical Features and Time Taken" heatmap, the numerical feature with the highest absolute correlation with the target variable `time_taken` is `distance`, which has a correlation coefficient of **0.46**.

Question 4. [2 marks]

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

Answer: My approach to detect outliers involved using visualizations like boxplots for both the target variable (`time_taken`) and other numerical features in the training set. Boxplots visually represent the distribution of data and clearly show points that fall outside the whiskers, which are typically considered outliers.

To address the outliers, I used the Interquartile Range (IQR) method. For each numerical column and the target variable, I calculated the first quartile (Q1) and the third quartile (Q3). The IQR is the difference between Q3 and Q1. Outliers were defined as values falling below `Q1 - 1.5 * IQR` or above `Q3 + 1.5 * IQR`. Instead of removing these outliers, I chose to cap and floor them using the calculated lower and upper bounds. This means that any value below the lower bound was set to the lower bound, and any value above the upper bound was set to the upper bound. This method helps to reduce the impact of extreme values without removing data points, which can be beneficial, especially with a large dataset. After applying the capping and flooring, I re-plotted the boxplots to confirm that the outliers were handled.

Question 5. [2 marks]

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

Answer: Based on the "Model Summary with Unscaled Features (RFE Selected)", the top 3 features with the largest absolute coefficient values (and thus the most significant impact on delivery time in minutes) are:

1. **`distance`**: This feature has the largest positive coefficient (**0.4844**), indicating that for each unit increase in distance, the delivery time increases by approximately 0.48 minutes, holding other features constant.

1. **`total_onshift_dashers`:** This feature has a large negative coefficient (**-0.3467**), indicating that for each additional dasher on shift, the delivery time *decreases* by approximately 0.35 minutes.

1. **`total_outstanding_orders`:** This feature has a large positive coefficient (**0.3421**), indicating that for each additional outstanding order, the delivery time *increases* by approximately 0.34 minutes.

General Subjective Questions

Question 6. [3 marks]

Explain the linear regression algorithm in detail

Answer: Linear regression is a fundamental supervised learning algorithm used for predicting a continuous target variable based on one or more input features (predictor variables). The goal of linear regression is to find a linear relationship between the input features and the target variable.

In simple linear regression, there is only one input feature, and the relationship is represented by a straight line:

$$y = \beta_0 + \beta_1 x + \epsilon$$

where:

- y is the target variable.
- x is the input feature.
- β_0 is the y-intercept (the value of y when x is 0).
- β_1 is the slope of the line (the change in y for a one-unit change in x).
- ϵ is the error term, representing the part of y that cannot be explained by the linear relationship.

In multiple linear regression, there are two or more input features, and the relationship is represented by a hyperplane:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$$

where:

- y is the target variable.
- $x_1, x_2, \ldots, x_n$ are the input features.
- β_0 is the y-intercept.

- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for each input feature, representing the change in y for a one-unit change in the corresponding feature, holding other features constant.
- ϵ is the error term.

The algorithm learns the optimal values for the coefficients ($\beta_0, \beta_1, \ldots, \beta_n$) by minimizing a cost function. The most common cost function is the Mean Squared Error (MSE), which calculates the average of the squared differences between the actual target values and the predicted target values.

The minimization of the cost function is typically done using optimization techniques like Gradient Descent or by using the Ordinary Least Squares (OLS) method, which provides a closed-form solution for the coefficients. The goal is to find the coefficients that result in the line or hyperplane that best fits the training data, minimizing the errors between the predicted and actual values. Once the model is trained, it can be used to predict the target variable for new, unseen data.

Question 7. [2 marks]

`Explain the difference between simple linear regression and multiple linear regression`

Answer: The main difference between simple linear regression and multiple linear regression lies in the number of independent (predictor) variables used to predict the dependent (target) variable:

- **Simple Linear Regression:** Uses only **one** independent variable to predict the dependent variable. The relationship is modeled as a straight line. The equation is $y = \beta_0 + \beta_1 x + \epsilon$.
- **Multiple Linear Regression:** Uses **two or more** independent variables to predict the dependent variable. The relationship is modeled as a hyperplane in a multi-dimensional space. The equation is $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$.

In essence, multiple linear regression is an extension of simple linear regression to handle scenarios where the target variable is influenced by multiple factors.

Question 8. [2 marks]

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

Answer: The role of the cost function in linear regression is to quantify the error between the predicted values of the model and the actual target values from the training data. It serves as a measure of how well the linear model is fitting the data. A lower cost function value indicates a better fit.

The most common cost function for linear regression is the **Mean Squared Error (MSE)**. It is calculated as the average of the squared differences between the actual values (y_i) and the predicted values ($\hat{y}_i$) for all data points:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Minimizing the cost function means finding the values of the model coefficients ($\beta_0, \beta_1, \ldots, \beta_n$) that result in the smallest possible MSE. This is typically achieved through optimization algorithms:

1. **Ordinary Least Squares (OLS):** This is a closed-form solution that directly calculates the optimal coefficients by minimizing the sum of the squared residuals. It's computationally efficient for smaller datasets but can be less practical for very large datasets or when dealing with complex models.

1. **Gradient Descent:** This is an iterative optimization algorithm. It starts with initial values for the coefficients and then repeatedly adjusts them in the direction that decreases the cost function most steeply. The "gradient" refers to the slope of the cost function with respect to each coefficient. The learning rate parameter controls the size of the steps taken during each iteration. Gradient Descent is widely used, especially for large datasets and in more complex machine learning models.

By minimizing the cost function, the linear regression algorithm finds the line (or hyperplane) that best represents the linear relationship between the features and the target variable, thereby providing the best possible predictions based on the given data and the linear model assumption.

Question 9. [2 marks]

Explain the difference between overfitting and underfitting.

Answer: Overfitting and underfitting are two common problems in machine learning that describe how well a model generalizes to new, unseen data:

- **Overfitting:** This occurs when a model learns the training data too well, including the noise and outliers. An overfitted model performs exceptionally well on the training data but poorly on the test data (and new data). It is too complex and has essentially memorized the training examples rather than learning the underlying patterns. Visualizing an overfitted model's performance often shows a training error that is very low, while the test error is significantly higher.

- **Underfitting:** This occurs when a model is too simple to capture the underlying patterns in the training data. An underfitted model performs poorly on both the training data and the test data. It fails to learn the relationships between the features and the target variable. Visualizing an underfitted model's performance often shows both training and test errors that are high and relatively close to each other.

The goal in model building is to find a model that achieves a good balance between underfitting and overfitting, generalizing well to new data. This is often achieved through techniques like cross-validation, regularization, and adjusting model complexity.

Question 10. [3 marks]

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

Answer: Residual plots are valuable diagnostic tools in linear regression for assessing whether the assumptions of the linear model are met and identifying potential issues. Residuals are the differences between the observed (actual) values and the predicted values from the regression model ($e_i = y_i - \hat{y}_i$). Plotting these residuals in various ways can reveal patterns that the model has not captured.

Here's how residual plots help in diagnosing a linear regression model:

1. **Checking for Linearity:** A residual plot where residuals are plotted against the predicted values (or against an independent variable) should show a random scatter of points around the horizontal line at zero. If there's a discernible pattern (e.g., a curve), it suggests that the relationship between the variables is not linear, and a linear model may not be appropriate.

1. **Checking for Homoscedasticity (Constant Variance):** Homoscedasticity assumes that the variance of the residuals is constant across all levels of the independent variables. In a residual plot, this is indicated by a roughly equal spread of points above and below the zero line across the range of predicted values. If the spread of residuals increases or decreases as the predicted values change (creating a funnel or cone shape), it suggests heteroscedasticity, which violates a key assumption of linear regression and can lead to biased standard errors.

2. **Checking for Independence of Errors:** The assumption of independent errors means that the residuals are not correlated with each other. While a standard residual vs. predicted plot doesn't directly show independence over time or sequence, plotting residuals against the order of data collection (if applicable) can reveal patterns like autocorrelation.

3. **Identifying Outliers and Influential Points:** Residual plots can help spot outliers (points with large residuals) and influential points (points that significantly impact the regression line). Points that fall far from the cluster of other residuals on the plot are potential outliers.

4. **Checking for Normality of Residuals:** A histogram or a Q-Q plot of the residuals can help assess whether the residuals are normally distributed. While linear regression doesn't strictly require normally distributed residuals for estimating

coefficients, it is important for valid statistical inference (e.g., hypothesis testing, confidence intervals). Deviations from normality can suggest issues with the model or data.

In summary, residual plots provide a visual way to examine the errors of a linear regression model and identify deviations from the model's assumptions, guiding further model refinement or indicating the need for a different modeling approach.

Conclusion

Based on the linear regression model built to predict delivery time, here are the key takeaways:

Insights:

- The model achieved an R-squared of approximately 0.89 on the test set, indicating that about 89% of the variance in delivery time can be explained by the selected features. This suggests the model has a strong predictive capability.
- The residual analysis showed a relatively normal distribution of errors centered around zero, which is a positive sign. However, the scatter plot of residuals versus predicted values shows some fanning out, suggesting potential heteroscedasticity (non-constant variance of errors), especially at higher predicted delivery times.
- Coefficient analysis on the unscaled data revealed the significant impact of certain features on delivery time:
- distance has the largest positive coefficient (0.4844), indicating that longer delivery distances result in significantly increased delivery time. This is a fundamental factor in delivery logistics.
- total_onshift_dashers has a large negative coefficient (-0.3467), suggesting that a higher number of available dashers leads to significantly shorter delivery times. This is expected, as more available resources can handle orders more efficiently.
- total_outstanding_orders has a substantial positive coefficient (0.3421), indicating that as the number of pending orders increases, delivery time also significantly increases. This aligns with intuition, as more outstanding orders likely lead to increased workload and delays.
- Categorical features related to order_hour show significant negative coefficients, suggesting orders placed during evening and late-night hours (19:00 to 23:00) have shorter delivery times compared to the baseline.
- market_id and order_protocol dummy variables also show significant negative coefficients, suggesting certain markets and order methods are associated with shorter delivery times compared to the baseline.

Assumptions:

- **Linearity:** The model assumes a linear relationship between the selected features and the target variable. While the residual plot did not show a strong curved pattern, the fanning out suggests that the linear assumption might not hold perfectly across the entire range of predicted values.
- **Independence of Errors:** The model assumes that the errors are independent. We did not explicitly check for temporal autocorrelation, which could be relevant for time-series data like this.
- **Homoscedasticity:** The model assumes constant variance of errors. The residual plot indicates that this assumption might be violated, as the spread of residuals appears to increase with higher predicted values.
- **Normality of Residuals:** While not strictly required for coefficient estimation, the model assumes normally distributed residuals for valid statistical inference. The histogram and Q-Q plot of residuals show a distribution that is close to normal but with some skewness and heavier tails, suggesting potential deviations.
- **No Multicollinearity:** The model assumes that the independent variables are not highly correlated with each other. We performed a correlation analysis and used RFE, which helps to mitigate multicollinearity by selecting a subset of features. The condition number in the unscaled model summary is large, which can indicate multicollinearity or other numerical issues, suggesting that some correlation might still exist among the selected features on the original scale.

Recommendations:

- **Address Heteroscedasticity:** Investigate methods to address the observed heteroscedasticity, such as transforming the target variable (e.g., using a logarithmic transformation) or using robust standard errors in the regression model.
- **Explore Non-Linear Relationships:** Consider exploring non-linear regression models or incorporating interaction terms between features to capture more complex relationships if the residual analysis strongly suggests non-linearity.
- **Time Series Analysis:** Given the time-stamped nature of the data, explore time series analysis techniques to account for potential temporal patterns or autocorrelation in delivery times.
- **Feature Engineering:** Further investigate feature engineering possibilities, such as creating features related to peak hours, weather conditions (if available), or historical delivery times for specific restaurants or areas.
- **Model Evaluation Metrics:** While RMSE and R-squared are good metrics, consider using other metrics like Mean Absolute Error (MAE) which is less sensitive to outliers, or metrics relevant to business goals (e.g., percentage of deliveries within a target time).

- **Validation on New Data:** Continuously validate the model's performance on new, unseen data to ensure it maintains its predictive accuracy over time and retrain the model periodically with updated data.
- **Investigate Outliers:** Although outliers were handled by capping and flooring, it might be beneficial to investigate the characteristics of the capped orders to understand the underlying reasons for extremely long or short delivery times.
- **Business Interpretation:** Work closely with business stakeholders to interpret the coefficients and model insights in the context of real-world delivery operations to identify actionable strategies for optimizing delivery times.