Order Delivery Time Prediction Report

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 optimize 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.

1. Loading the Data

```python
# Importing the file porter_data_1.csv

porter = pd.read_csv("porter_data_1.csv")
```

```python
porter.head()
```

	market_id	created_at	actual_delivery_time	store_primary_category	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	tot
0	1.0	2015-02-06 22:24:17	2015-02-06 23:11:17	4	1.0	4	3441	4	557	1239	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:33:25	46	2.0	1	1900	1	1400	1400	
2	2.0	2015-02-16 00:11:35	2015-02-16 01:06:35	36	3.0	4	4771	3	820	1604	
3	1.0	2015-02-12 03:36:46	2015-02-12 04:35:46	38	1.0	1	1525	1	1525	1525	
4	1.0	2015-01-27 02:12:36	2015-01-27 02:58:36	38	1.0	2	3620	2	1425	2195	

2. Data preprocessing and Feature Engineering

a) Converted created_at and actual_delivery_time features into datetime.

b) Identified Categorical variables into category data type.

c) Deduced delivery time in minutes

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

```python
porter.head()
```

l_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_dashers	total_busy_dashers	total_outstanding_orders	distance	delivery_time_minutes
4	3441	4	557	1239	33.0	14.0	21.0	34.44	47.0
1	1900	1	1400	1400	1.0	2.0	2.0	27.60	44.0
4	4771	3	820	1604	8.0	6.0	18.0	11.56	55.0

d) Dropped unnecessary features such as created_at, actual_delivery_time, and store_primary_category

e) Finally created train and test data sets.

Split the data into training and test sets

```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.70, random_state=100)
```

```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((123043, 14), (52734, 14), (123043,), (52734,))
```

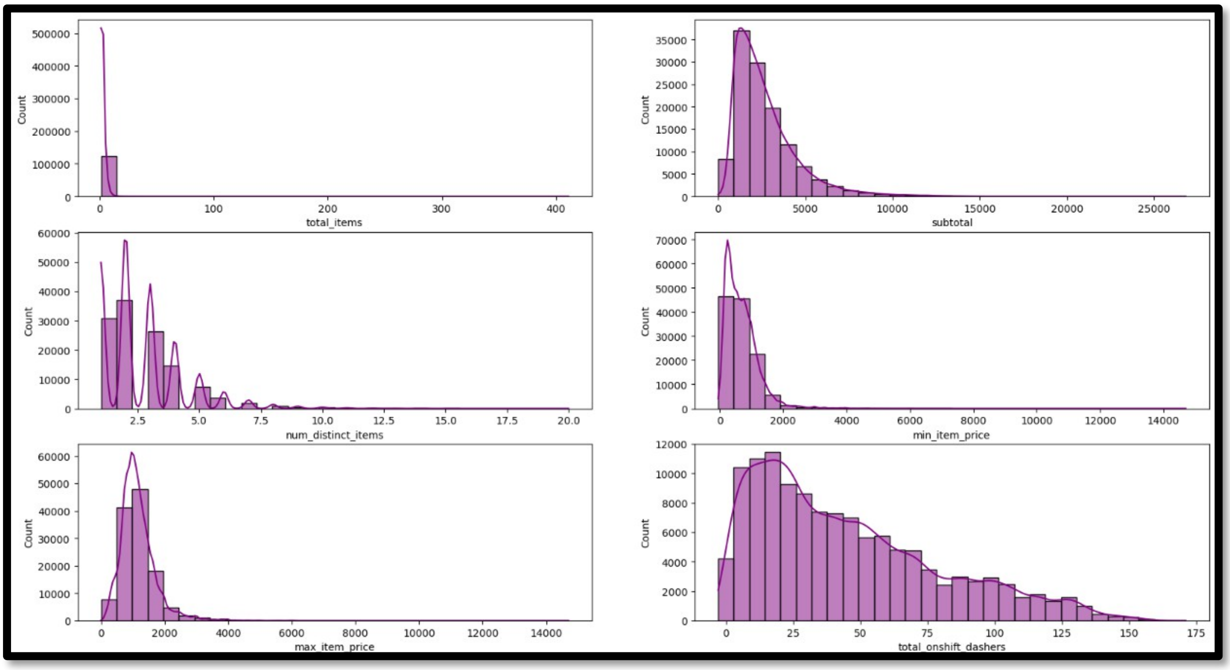
```python
X_train.head()
```

	market_id	order_protocol	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	total_onshift_dashers	total_busy_dashers	total_ou
94746	4.0	5.0	2	1790	2	795	995	10.0	9.0	
173338	4.0	5.0	1	845	1	795	795	134.0	76.0	

3. Exploratory Data Analysis

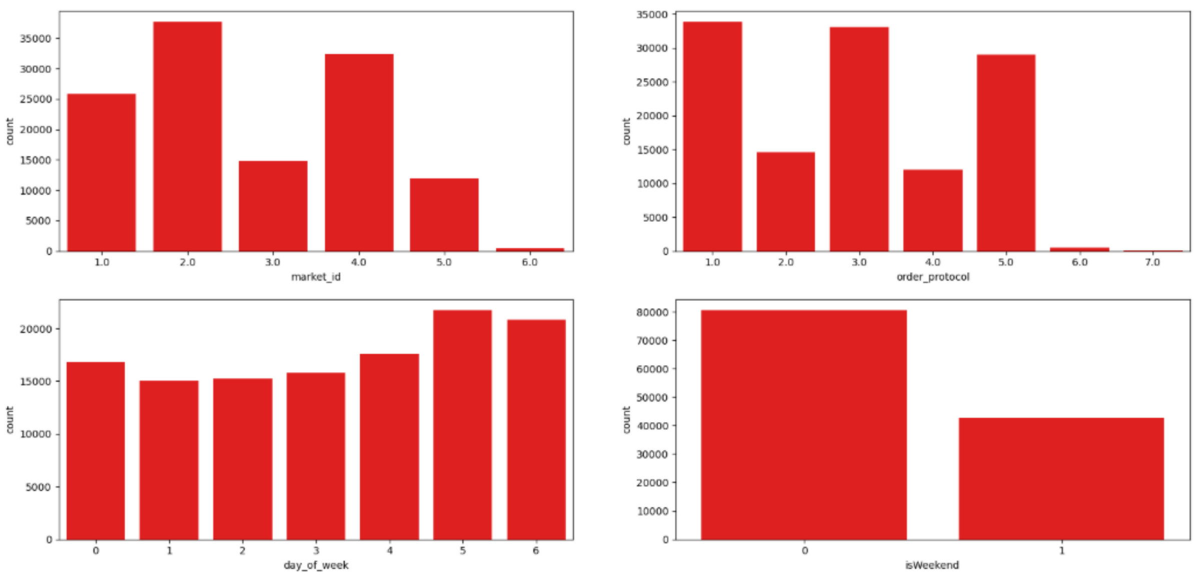
a) Separated out numerical and categorical features.

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



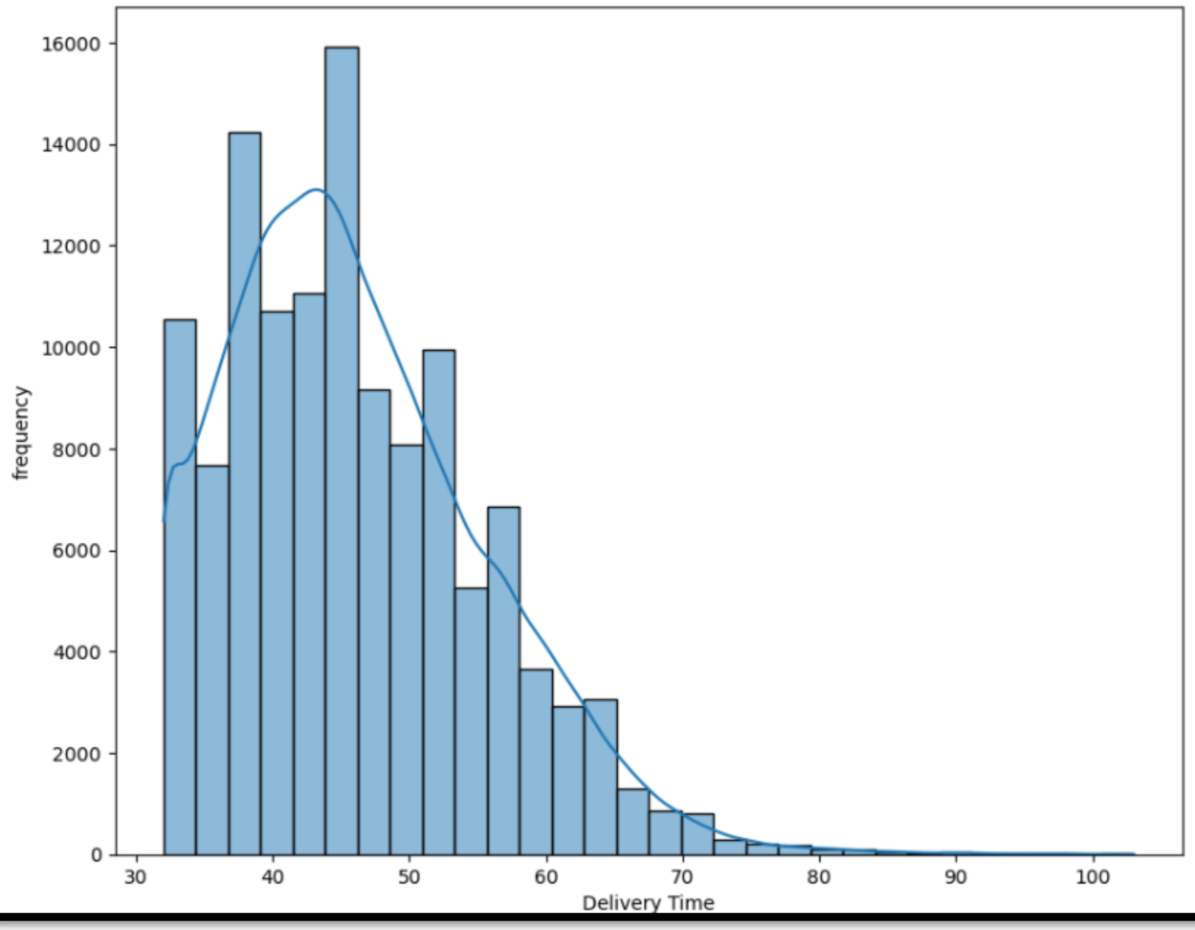
c) Plot distributions for categorical columns in the training set to understand their spread.

```
# Distribution of categorical columns

plt.figure(figsize=(20, 30))
i = 1
for c_col in cat_col:
    plt.subplot(6,2,i)
    sns.countplot(data=X_train, x=c_col, color='red')
    i += 1
plt.show()
```



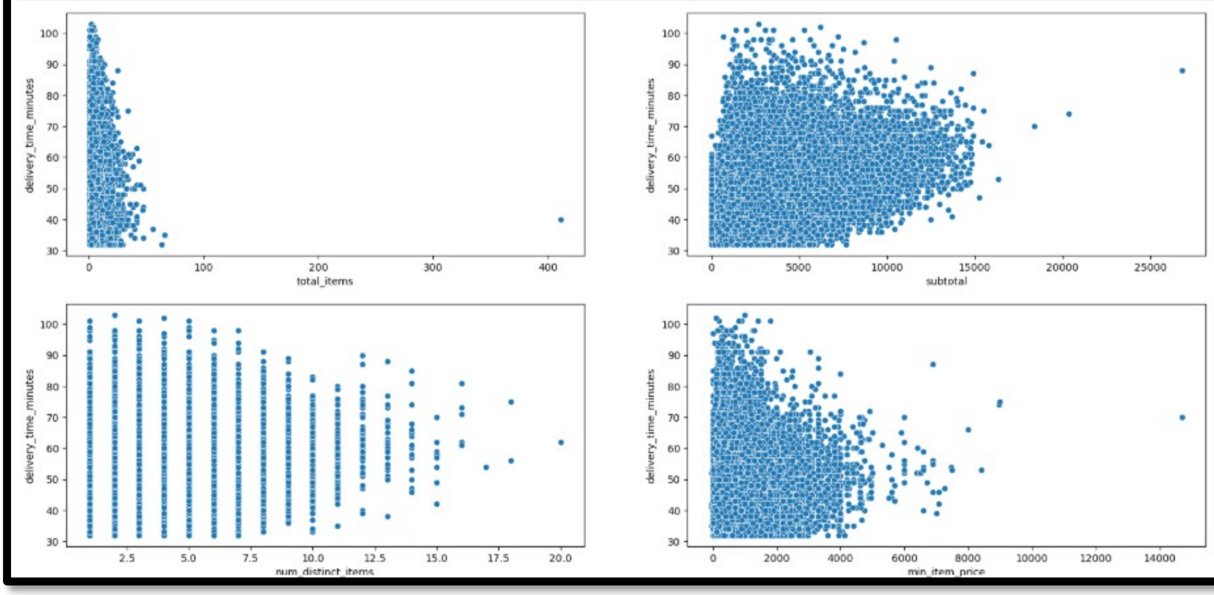
d) Plot distributions for target variable.

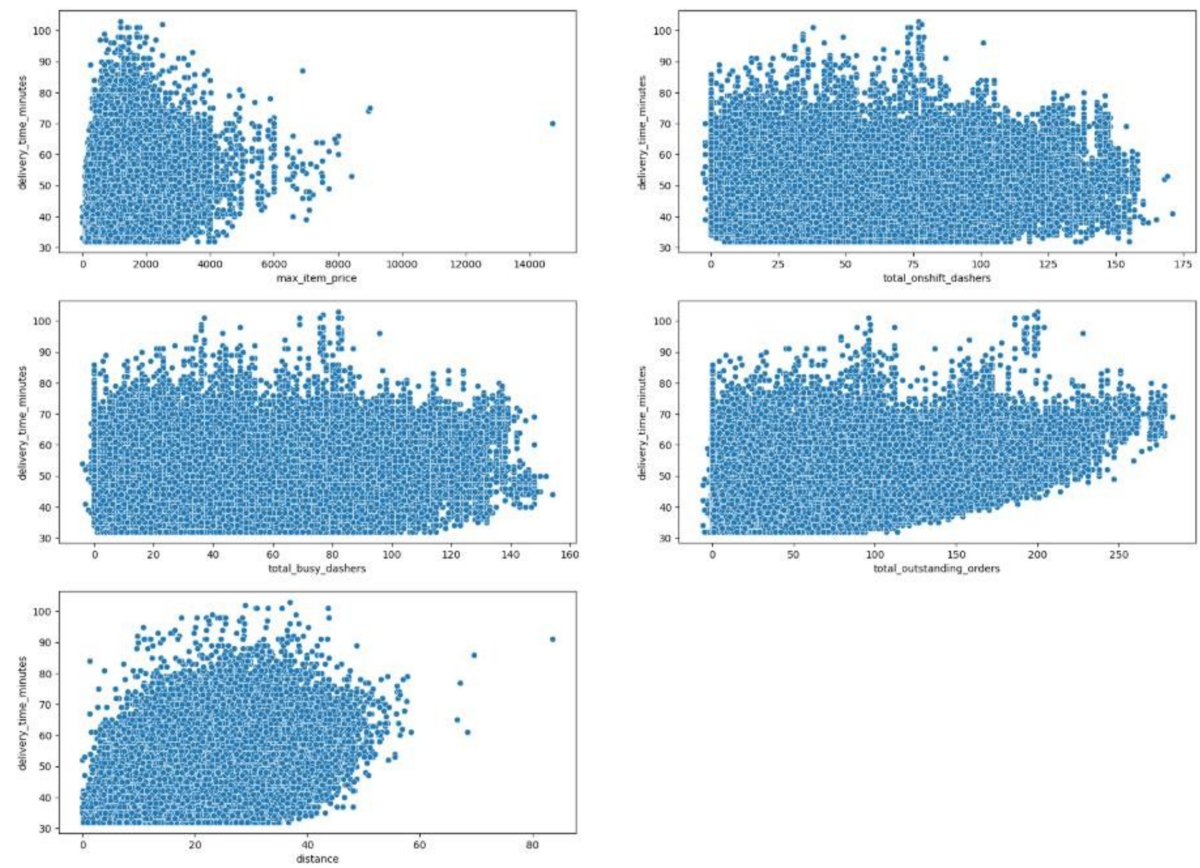
```
# Distribution of time_taken
plt.figure(figsize=(10, 8))
sns.histplot(y_train, kde=True, bins=30)
plt.xlabel("Delivery Time")
plt.ylabel("frequency")
plt.show()
```



e) Scatter plot for numeric and categorical features.

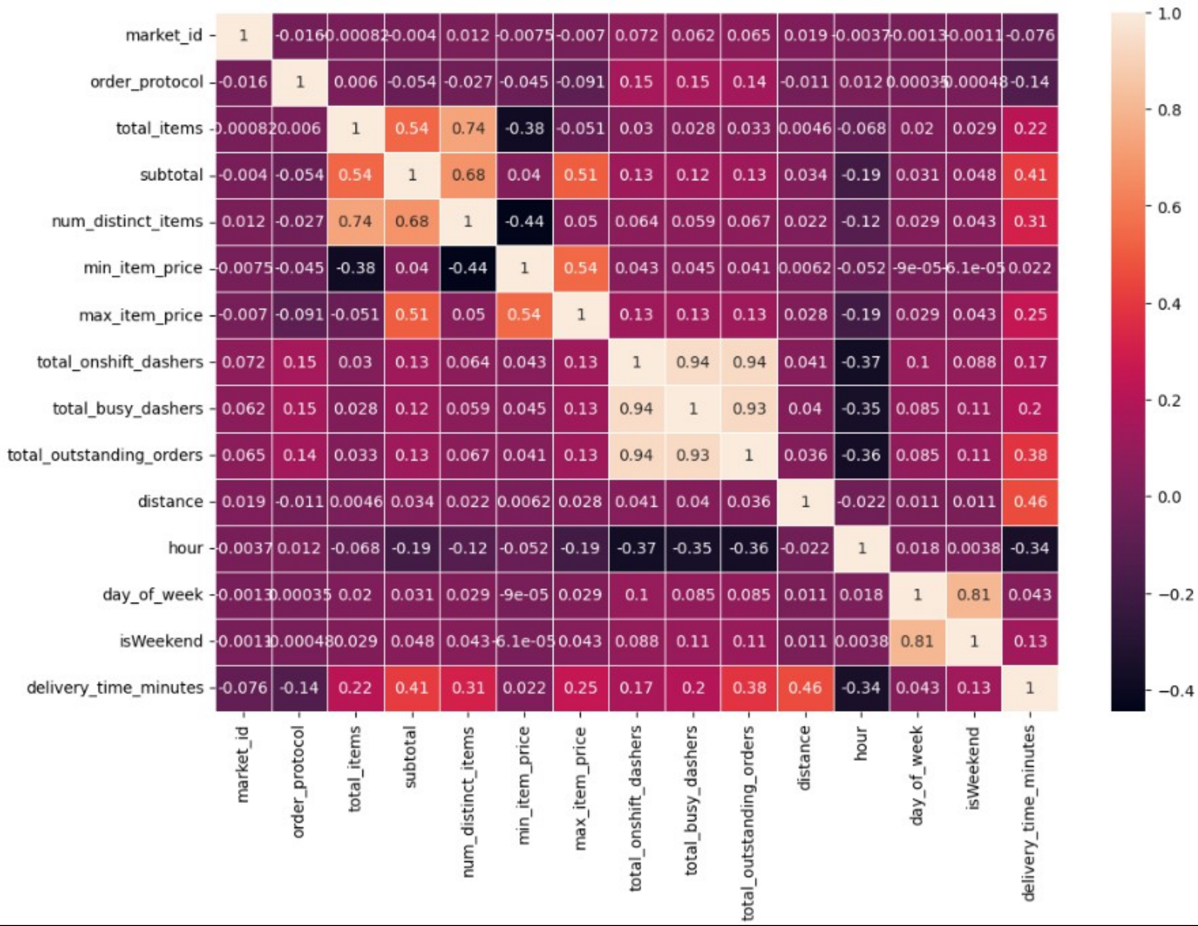
```python
# Scatter plot to visualise the relationship between time_taken and other features
plt.figure(figsize=(20, 30))
i = 1
for var in num_col:
    plt.subplot(6,2,i)
    sns.scatterplot(x=X_train[var], y=y_train)
    i += 1
plt.show()
```





f) Created a heatmap of entire features against delivery time to see the relationship among the variables and removed the features with low correlation.

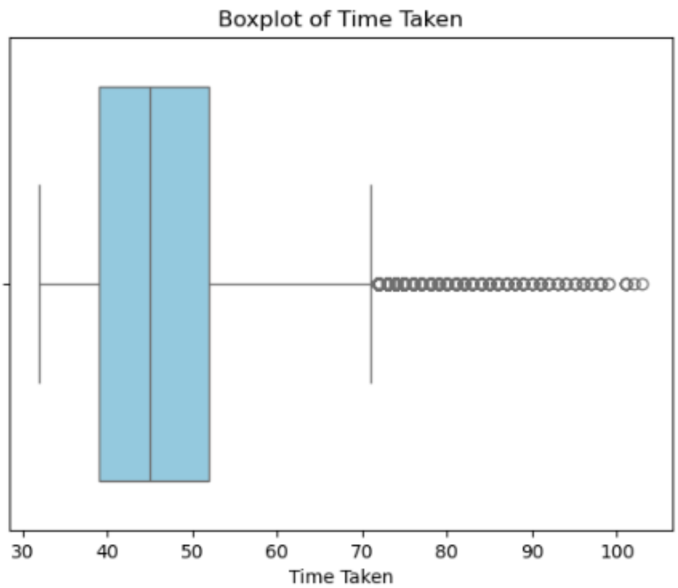
```python
# Plot the heatmap of the correlation matrix
traindata = X_train.copy()
traindata['delivery_time_minutes'] = y_train

plt.figure(figsize=(12,8))
sns.heatmap(traindata.corr(), annot=True, linewidth=0.5)
plt.show()
```



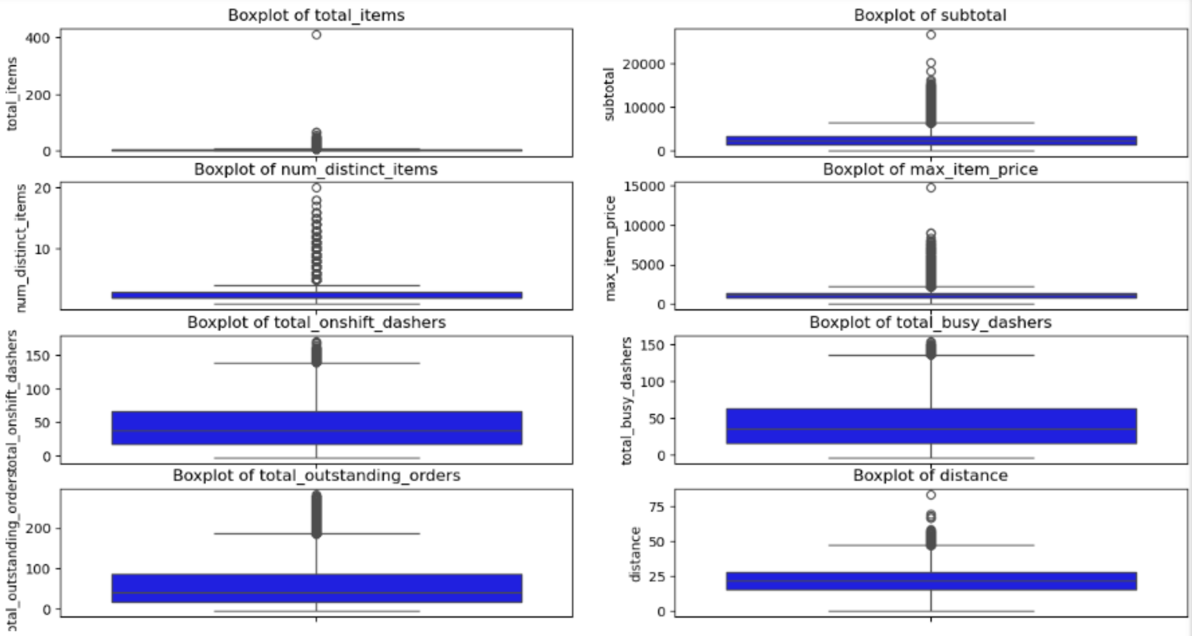

```python
# Drop 3-5 weakly correlated columns from training dataset
# By looking at the heatmap, it's clearly visible that market_id', min_item_price' have weak correlation.
X_train = X_train.drop(['market_id','min_item_price'], axis=1)
```

g) Handling of outliers was performed thereafter. I used boxplot to understand whether there are any outliers in the variables or not. IQR method was also used to find the outliers. Once outliers were found, the rows were dropped accordingly.

```python
# Boxplot for time_taken
sns.boxplot(data=X_train, x=y_train, color='skyblue')
plt.title('Boxplot of Time Taken')
plt.xlabel('Time Taken')
plt.show()
```




```python
plt.figure(figsize=(15,12))
i = 1
for var in num_col:
    plt.subplot(6,2,i)
    sns.boxplot(y=X_train[var], color="blue")
    plt.title(f"Boxplot of {var}")
    i += 1
plt.show()
```



```python
df = pd.DataFrame(X_train)

# Function to find outlier range for each column
def find_outliers(df):
    outlier_info = {}

    for col in num_col:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]

        outlier_info[col] = {
            'Q1': Q1,
            'Q3': Q3,
            'IQR': IQR,
            'Lower Bound': lower_bound,
            'Upper Bound': upper_bound,
            'Outliers': outliers[col].values.tolist()
        }

    return outlier_info

outliers_info = find_outliers(df)

for col, info in outliers_info.items():
    print(f"Outlier Info for Column '{col}':")
    print(f"  Q1: {info['Q1']}")
    print(f"  Q3: {info['Q3']}")
    print(f"  IQR: {info['IQR']}")
    print(f"  Lower Bound: {info['Lower Bound']}")
    print(f"  Upper Bound: {info['Upper Bound']}")
```

```
Outlier Info for Column 'total_items':
  Q1: 2.0
  Q3: 4.0
  IQR: 2.0
  Lower Bound: -1.0
  Upper Bound: 7.0
Outlier Info for Column 'subtotal':
  Q1: 1417.0
  Q3: 3405.0
  IQR: 1988.0
  Lower Bound: -1565.0
  Upper Bound: 6387.0
Outlier Info for Column 'num_distinct_items':
  Q1: 2.0
  Q3: 3.0
  IQR: 1.0
  Lower Bound: 0.5
  Upper Bound: 4.5
Outlier Info for Column 'max_item_price':
  Q1: 799.0
  Q3: 1395.0
  IQR: 596.0
  Lower Bound: -95.0
  Upper Bound: 2289.0
Outlier Info for Column 'total_onshift_dashers':
  Q1: 17.0
  Q3: 66.0
  IQR: 49.0
  Lower Bound: -56.5
  Upper Bound: 139.5
Outlier Info for Column 'total_busy_dashers':
  Q1: 15.0
  Q3: 63.0
  IQR: 48.0
  Lower Bound: -57.0
  Upper Bound: 135.0
Outlier Info for Column 'total_outstanding_orders':
  Q1: 17.0
  Q3: 85.0
  IQR: 68.0
  Lower Bound: -85.0
  Upper Bound: 187.0
```

```python
# Drop the data above the upper range in all numerical variables
y_train.drop(index=X_train[X_train['total_items']>7].index, inplace=True)
X_train.drop(index=X_train[X_train['total_items']>7].index, inplace=True)

y_train.drop(index=X_train[X_train['subtotal']>6387].index, inplace=True)
X_train.drop(index=X_train[X_train['subtotal']>6387].index, inplace=True)

y_train.drop(index=X_train[X_train['num_distinct_items']>4.5].index, inplace=True)
X_train.drop(index=X_train[X_train['num_distinct_items']>4.5].index, inplace=True)

y_train.drop(index=X_train[X_train['max_item_price']>2289].index, inplace=True)
X_train.drop(index=X_train[X_train['max_item_price']>2289].index, inplace=True)

y_train.drop(index=X_train[X_train['total_onshift_dashers']>139.5].index, inplace=True)
X_train.drop(index=X_train[X_train['total_onshift_dashers']>139.5].index, inplace=True)

y_train.drop(index=X_train[X_train['total_busy_dashers']>135].index, inplace=True)
X_train.drop(index=X_train[X_train['total_busy_dashers']>135].index, inplace=True)

y_train.drop(index=X_train[X_train['total_outstanding_orders']>187].index, inplace=True)
X_train.drop(index=X_train[X_train['total_outstanding_orders']>187].index, inplace=True)

y_train.drop(index=X_train[X_train['distance']>47.32].index, inplace=True)
X_train.drop(index=X_train[X_train['distance']>47.32].index, inplace=True)

X_train.shape, y_train.shape
```

```
((98775, 12), (98775,))
```

```python
# Handle outliers
condition = (y_train <= 69)
X_train = X_train[condition]
y_train = y_train[condition]
```

4. Model Building

a) Imported the necessary libraries first followed feature scaling.

Import Necessary Libraries

```python
# Import libraries
import statsmodels

import statsmodels.api as sm

from sklearn.preprocessing import MinMaxScaler

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn.feature_selection import RFE

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

from sklearn.feature_selection import RFE
```

```python
# Apply scaling to the numerical columns
scaler = MinMaxScaler()

num_vars = ['order_protocol','total_items', 'subtotal', 'num_distinct_items',
       'max_item_price', 'total_onshift_dashers', 'total_busy_dashers',
       'total_outstanding_orders', 'distance', 'hour', 'day_of_week', 'isWeekend']

X_train[num_vars] = scaler.fit_transform(X_train[num_vars])
X_train.head()
```

	order_protocol	total_items	subtotal	num_distinct_items	max_item_price	total_onshift_dashers	total_busy_dashers	total_outstanding_orders	distance	hou
94746	0.666667	0.166667	0.280564	0.333333	0.435068	0.091549	0.093525	0.108808	0.381234	0.78260
173338	0.666667	0.000000	0.132445	0.000000	0.347617	0.964789	0.575540	0.735751	0.513102	0.08695
37592	0.666667	0.000000	0.297806	0.000000	0.524705	0.169014	0.179856	0.176166	0.596788	1.00000
42763	0.500000	0.833333	0.072571	0.666667	0.130739	0.711268	0.654676	0.699482	0.293322	0.04347
27506	0.000000	0.333333	0.548589	0.666667	0.524705	0.070423	0.122302	0.119171	0.169062	0.21739

b) Built the very first model using the train set and model came out to be decent with high R2 (~86) and 0 p-values for all features except one “day_of_week”.

1st Model

```python
# Create/Initialise the model
X_train_sm = sm.add_constant(X_train)
```

```python
X_train_sm.head()
```

	const	order_protocol	total_items	subtotal	num_distinct_items	max_item_price	total_onshift_dashers	total_busy_dashers	total_outstanding_orders	distance
94746	1.0	0.666667	0.166667	0.280564	0.333333	0.435068	0.091549	0.093525	0.108808	0.381234
173338	1.0	0.666667	0.000000	0.132445	0.000000	0.347617	0.964789	0.575540	0.735751	0.513102
37592	1.0	0.666667	0.000000	0.297806	0.000000	0.524705	0.169014	0.179856	0.176166	0.596788
42763	1.0	0.500000	0.833333	0.072571	0.666667	0.130739	0.711268	0.654676	0.699482	0.293322
27506	1.0	0.000000	0.333333	0.548589	0.666667	0.524705	0.070423	0.122302	0.119171	0.169062

```python
# Create/Initialise the model for test set
X_test_sm = sm.add_constant(X_test)
```

```python
X_test_sm.head()
```

	const	order_protocol	total_items	subtotal	num_distinct_items	max_item_price	total_onshift_dashers	total_busy_dashers	total_outstanding_orders	distance
139667	1.0	0.000000	0.333333	0.202978	0.333333	0.435068	0.436620	0.453237	0.347150	0.434489
80077	1.0	0.000000	0.166667	0.462382	0.000000	0.535636	0.147887	0.151079	0.113990	0.271344
41872	1.0	0.500000	0.000000	0.218652	0.000000	0.609969	0.218310	0.223022	0.264249	0.355030
165269	1.0	0.666667	0.166667	0.465047	0.333333	0.546130	0.894366	0.942446	1.067358	0.401522
151215	1.0	0.166667	0.333333	0.195925	0.333333	0.174902	0.295775	0.244604	0.202073	0.564666

```python
# Create Model
lr = sm.OLS(y_train, X_train_sm)
```

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

lr_model.summary()
```

Dep. Variable:	delivery_time_minutes	**R-squared:**	0.859
Model:	OLS	**Adj. R-squared:**	0.859
Method:	Least Squares	**F-statistic:**	4.994e+04
Date:	Sat, 29 Mar 2025	**Prob (F-statistic):**	0.00
Time:	01:21:32	**Log-Likelihood:**	-2.4708e+05
No. Observations:	98026	**AIC:**	4.942e+05
Df Residuals:	98013	**BIC:**	4.943e+05
Df Model:	12		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	35.0565	0.050	705.202	0.000	34.959	35.154
order_protocol	-3.8925	0.039	-100.646	0.000	-3.968	-3.817
total_items	-0.4830	0.099	-4.871	0.000	-0.677	-0.289
subtotal	8.4841	0.102	82.830	0.000	8.283	8.685
num_distinct_items	1.4569	0.055	26.644	0.000	1.350	1.564
max_item_price	1.1808	0.085	13.841	0.000	1.014	1.348
total_onshift_dashers	-52.2586	0.149	-350.305	0.000	-52.551	-51.966
total_busy_dashers	-18.9793	0.146	-129.785	0.000	-19.266	-18.693
total_outstanding_orders	67.9804	0.129	528.129	0.000	67.728	68.233
distance	22.0129	0.053	418.304	0.000	21.910	22.116
hour	-5.3490	0.027	-194.960	0.000	-5.403	-5.295
day_of_week	0.0284	0.049	0.585	0.559	-0.067	0.124
isWeekend	1.4863	0.035	42.565	0.000	1.418	1.555

```
# Make predictions
y_test_pred = lr_model.predict(X_test_sm)
y_test_pred
```

```
139667    38.682745
80077     43.935203
41872     45.139708
165269    55.042130
151215    39.062535
            ...
46607     51.588530
159653    41.783802
78090     37.372877
98746     62.324136
3735      49.880708
Length: 52734, dtype: float64
```

```
# Find results for evaluation metrics

mae = mean_absolute_error(y_test, y_test_pred)
mse = mean_squared_error(y_test, y_test_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_test_pred)

print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.4f}")
```

```
Mean Absolute Error (MAE): 2.34
Mean Squared Error (MSE): 11.05
Root Mean Squared Error (RMSE): 3.32
R² Score: 0.8727
```

c) Built the model and fit RFE to select the most important features. We used RFE to reduce less significant features one-by-one and then chose the best model among all of them.
All the models came out to be significant and quite close. It was a difficult choice to choose any one model. Finally, I chose the last model which had only relevant features to determine whether delivery time will increase or decrease.

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

model = LinearRegression()

rfe = RFE(estimator=model, n_features_to_select=(X_train.shape[1]-1))
```

```
# Fit RFE on training data
rfe.fit_transform(X_train, y_train)

list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

```
[('order_protocol', True, 1),
 ('total_items', True, 1),
 ('subtotal', True, 1),
 ('num_distinct_items', True, 1),
 ('max_item_price', True, 1),
 ('total_onshift_dashers', True, 1),
 ('total_busy_dashers', True, 1),
 ('total_outstanding_orders', True, 1),
 ('distance', True, 1),
 ('hour', True, 1),
 ('day_of_week', False, 2),
 ('isWeekend', True, 1)]
```

2nd Model

```
# Creating the model again after removing day_of_week

# Add a constant
X_train_sm = sm.add_constant(X_train)

# Create Model
lr = sm.OLS(y_train, X_train_sm)

# Fit the model
lr_model = lr.fit()

lr_model.summary()
```

OLS Regression Results

Dep. Variable:	delivery_time_minutes	**R-squared:**	0.859
Model:	OLS	**Adj. R-squared:**	0.859
Method:	Least Squares	**F-statistic:**	5.448e+04
Date:	Sat, 29 Mar 2025	**Prob (F-statistic):**	0.00
Time:	01:21:33	**Log-Likelihood:**	-2.4708e+05
No. Observations:	98026	**AIC:**	4.942e+05
Df Residuals:	98014	**BIC:**	4.943e+05
Df Model:	11		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	35.0655	0.047	741.754	0.000	34.973	35.158
order_protocol	-3.8927	0.039	-100.656	0.000	-3.969	-3.817
total_items	-0.4830	0.099	-4.872	0.000	-0.677	-0.289
subtotal	8.4840	0.102	82.830	0.000	8.283	8.685
num_distinct_items	1.4569	0.055	26.644	0.000	1.350	1.564
max_item_price	1.1807	0.085	13.840	0.000	1.013	1.348
total_onshift_dashers	-52.2425	0.147	-356.290	0.000	-52.530	-51.955
total_busy_dashers	-18.9896	0.145	-130.807	0.000	-19.274	-18.705
total_outstanding_orders	67.9759	0.128	529.019	0.000	67.724	68.228

3rd Model

```
# Creating the model again after removing day_of_week

# Add a constant
X_train_sm = sm.add_constant(X_train)

# Create Model
lr = sm.OLS(y_train, X_train_sm)

# Fit the model
lr_model = lr.fit()

lr_model.summary()
```

OLS Regression Results

Dep. Variable:	delivery_time_minutes	**R-squared:**	0.858
Model:	OLS	**Adj. R-squared:**	0.858
Method:	Least Squares	**F-statistic:**	6.574e+04
Date:	Sat, 29 Mar 2025	**Prob (F-statistic):**	0.00
Time:	01:21:35	**Log-Likelihood:**	-2.4762e+05
No. Observations:	98026	**AIC:**	4.953e+05
Df Residuals:	98016	**BIC:**	4.953e+05
Df Model:	9		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	35.3429	0.045	788.443	0.000	35.255	35.431
order_protocol	-3.9299	0.039	-101.103	0.000	-4.006	-3.854
subtotal	10.1274	0.067	151.115	0.000	9.996	10.259
max_item_price	0.4163	0.068	6.088	0.000	0.282	0.550
total_onshift_dashers	-52.2017	0.147	-354.074	0.000	-52.491	-51.913
total_busy_dashers	-19.0456	0.146	-130.482	0.000	-19.332	-18.760
total_outstanding_orders	67.9779	0.129	526.146	0.000	67.725	68.231
distance	22.0200	0.053	416.160	0.000	21.916	22.124
hour	-5.3788	0.028	-195.228	0.000	-5.433	-5.325
isWeekend	1.5219	0.021	73.440	0.000	1.481	1.563

Finally, I chose the latest model as I mentioned and below are the results of my final model:

All the models are almost similar with just minor difference in the R2, so we are considering the last model as final model.

```python
# Build the final model with selected number of features

X_train.sm = sm.add_constant(X_train)
```

```python
# Create Model
lr = sm.OLS(y_train, X_train_sm)

# Fit the model
lr_model = lr.fit()

lr_model.summary()
```

OLS Regression Results

Dep. Variable:	delivery_time_minutes	**R-squared:**	0.835
Model:	OLS	**Adj. R-squared:**	0.835
Method:	Least Squares	**F-statistic:**	8.255e+04
Date:	Sat, 29 Mar 2025	**Prob (F-statistic):**	0.00
Time:	01:21:36	**Log-Likelihood:**	-2.5499e+05
No. Observations:	98026	**AIC:**	5.100e+05
Df Residuals:	98019	**BIC:**	5.101e+05
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	34.6984	0.041	836.576	0.000	34.617	34.780
subtotal	11.0125	0.060	183.306	0.000	10.895	11.130
total_onshift_dashers	-53.2507	0.159	-335.670	0.000	-53.562	-52.940
total_busy_dashers	-19.1701	0.157	-122.087	0.000	-19.478	-18.862
total_outstanding_orders	68.3947	0.139	491.188	0.000	68.122	68.668
distance	22.1283	0.057	387.949	0.000	22.017	22.240
hour	-5.4608	0.030	-184.732	0.000	-5.519	-5.403

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

```
X_test_sm = sm.add_constant(X_test)
```

```
# Make predictions
y_test_pred = lr_model.predict(X_test_sm)
```

```
final_coef=lr_model.params
final_coef
```

```
const                        34.698378
subtotal                     11.012462
total_onshift_dashers       -53.250700
total_busy_dashers          -19.170141
total_outstanding_orders     68.394668
distance                     22.128334
hour                         -5.460835
dtype: float64
```

```
# Find results for evaluation metrics

mae = mean_absolute_error(y_test, y_test_pred)
mse = mean_squared_error(y_test, y_test_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_test_pred)

# Print results
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R² Score: {r2:.4f}")
```

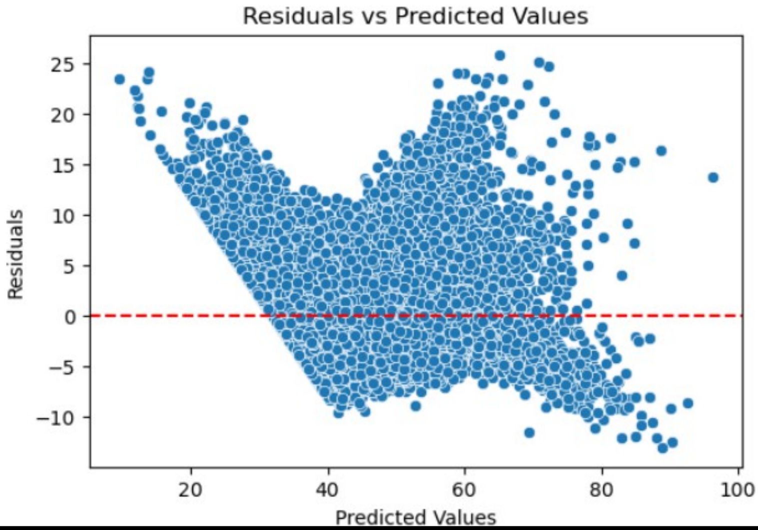
```
Mean Absolute Error (MAE): 2.59
Mean Squared Error (MSE): 12.66
Root Mean Squared Error (RMSE): 3.56
R² Score: 0.8542
```

5. Results and Inference

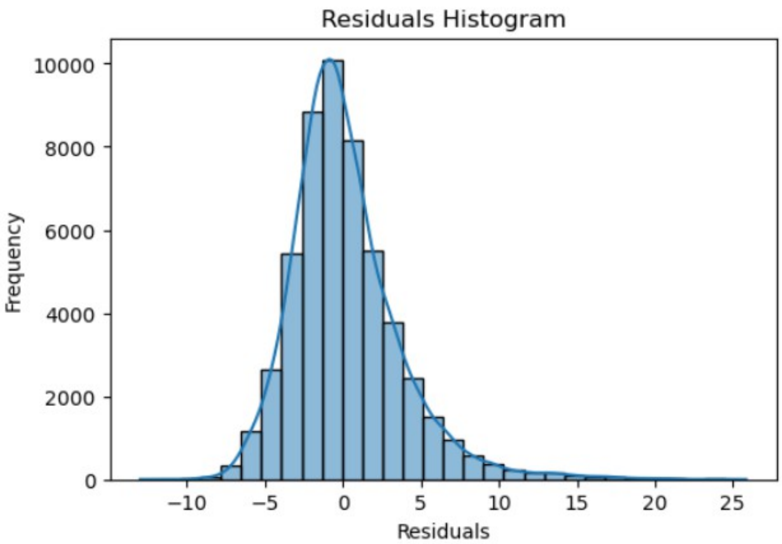
a) Performed Residual analysis and created error term distribution which has mean at 0 which means our assumption is satisfied.

```python
# Perform residual analysis using plots like residuals vs predicted values, Q-Q plot and residual histogram
residuals = y_test - y_test_pred
```

```python
#residuals vs predicted values
plt.figure(figsize=(6,4))
sns.scatterplot(x=y_test_pred, y=residuals)
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Predicted Values")
plt.show()
```




```python
# Residual Histogram
plt.figure(figsize=(6,4))
sns.histplot(residuals, bins=30, kde=True)
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.title("Residuals Histogram")
plt.show()
```



b) Performed coefficient analysis to check how changes in the features affect the target.

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

#Extracting the Coef
coef_scaled = final_coef
coef_scaled = coef_scaled.drop('const')

# Extracting the features
final_features = X_train.columns

# Create dataframe
coef_df = pd.DataFrame({"Feature": final_features, "Scaled_Coef": coef_scaled})

# Separating numerical features
num_features = ['subtotal', 'total_onshift_dashers', 'total_busy_dashers',
       'total_outstanding_orders', 'distance', 'hour']

# Calculate standard deviation
feature_sd = X_train[num_features].std()

#  Compute unscaled coefficients for numerical features
coef_df["Unscaled_Coef"] = np.nan
coef_df.loc[coef_df["Feature"].isin(num_features), "Unscaled_Coef"] = (
    coef_df.loc[coef_df["Feature"].isin(num_features), "Scaled_Coef"].values
    / feature_sd.values
)

# Sort by absolute impact
#coef_df["Absolute Impact"] = np.abs(coef_df["Unscaled Coefficient"])
#coef_df = coef_df.sort_values(by="Absolute Impact", ascending=False)

# Display the coefficient comparison
print(coef_df[["Feature", "Scaled_Coef", "Unscaled_Coef"]])
```

```
                                           Feature  Scaled_Coef  Unscaled_Coef
subtotal                                  subtotal    11.012462      61.880985
total_onshift_dashers        total_onshift_dashers   -53.250700    -241.748818
total_busy_dashers              total_busy_dashers   -19.170141     -90.953517
total_outstanding_orders  total_outstanding_orders    68.394668     293.029990
distance                                  distance    22.128334     120.960194
hour                                          hour    -5.460835     -14.202989
```

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

unit_change = 1  #  define unit change total_items

# Get the coefficient for 'total_items'
coef_total_outstanding_orders = coef_df.loc[coef_df["Feature"] == "total_outstanding_orders", "Scaled_Coef"].values[0]

impact_on_time_taken = round(unit_change * coef_total_outstanding_orders,2)

print("Impact of unit change in total_items is :", impact_on_time_taken)
```

```
Impact of unit change in total_items is : 68.39
```

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.

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

# I am doing this analysis on total_outstanding_orders as it has huge impact on the delivery time.
unit_change = 1  #  define unit change in total_outstanding_orders

# Get the coefficient for 'total_outstanding_orders'
coef_total_outstanding_orders = coef_df.loc[coef_df["Feature"] == "total_outstanding_orders", "Scaled_Coef"].values[0]

impact_on_time_taken = round(unit_change * coef_total_outstanding_orders,2)

print("Impact of unit change in total_items is :", impact_on_time_taken)
```

```
Impact of unit change in total_items is : 68.39
```

6. Conclusion:

a) total_outstanding_orders has the highest impact on the target variable/
b) total_onshift_dashers and distance have the significant impact on the model
c) Outlier handling improved the overall model performance. d)