porter-delivery-time-estimation

March 12, 2025

[105]:
```
#!pip list | findstr numpy

#!pip uninstall numpy -y
#!pip install numpy==1.21.6

# !pip install tensorflow
# !pip install numpy==1.24.3

# !pip install xgboost

# %load_ext tensorboard
# %tensorboard --logdir=./logs

# !pip install scikeras

# !pip install hyperopt

import pandas as pd
import numpy as  np
import matplotlib.pyplot as plt
import seaborn as sns

from IPython.display import Image

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

Problem Statement

India's largest intra-city logistics marketplace, aims to revolutionize delivery services by leveraging technology to provide accurate delivery time estimates, a key driver of customer satisfaction. This project focuses on predicting delivery times using a dataset of order and delivery details, including market areas, restaurant categories, order volumes, and partner availability.

1. To develop a predictive model that delivers precise delivery time estimates, reducing customer churn and improving satisfaction
2. To optimize the allocation of delivery partners by understanding demand patterns and operational capacity
3. To identify bottlenecks causing delays, enabling operational efficiency gains

By employing ***exploratory data analysis, feature engineering, and machine learning models (Random Forest, XGBoost, and Neural Networks)***, this analysis seeks to provide actionable insights that enhance Porter's ability to ensure timely deliveries and streamline logistics operations

[106]:
```
data = pd.read_csv(r"C:\Users\akash.
↪bana\Desktop\Akash_backup\Akash\Scaler\Jobs\Data Scientist\Porter delivery␣
↪time estimation\dataset.csv")
df = data.copy()
df.head(2)
```

[106]:

	market_id	created_at	actual_delivery_time	\
0	1.0	2015-02-06 22:24:17	2015-02-06 23:27:16	
1	2.0	2015-02-10 21:49:25	2015-02-10 22:56:29	

	store_id	store_primary_category	order_protocol	\
0	df263d996281d984952c07998dc54358	american	1.0	
1	f0ade77b43923b38237db569b016ba25	mexican	2.0	

	total_items	subtotal	num_distinct_items	min_item_price	max_item_price	\
0	4	3441	4	557	1239	
1	1	1900	1	1400	1400	

	total_onshift_partners	total_busy_partners	total_outstanding_orders
0	33.0	14.0	21.0
1	1.0	2.0	2.0

[107]:
```
Image(filename =r"C:\Users\akash.
↪bana\Desktop\Akash_backup\Akash\Scaler\Jobs\Data Scientist\Porter delivery␣
↪time estimation\features.png",width = 500)
```

[107]:

1. market_id: An integer ID indicating the market area of the restaurant.
2. created_at: Timestamp of when the order was placed.
3. actual_delivery_time: Timestamp of when the order was delivered.
4. store_primary_category: Category classification of the restaurant.
5. order_protocol: Numeric code representing the mode of order placement (e.g., through Porter, direct call, pre-booking, third-party platform).
6. total_items_subtotal: A combined feature detailing the total number of items in the order and the final price of the order before taxes and fees.
7. num_distinct_items: Count of different items in the order.
8. min_item_price: Price of the least expensive item in the order.
9. max_item_price: Price of the most expensive item in the order.
10. total_onshift_partners: Number of delivery partners on duty when the order was placed.
11. total_busy_partners: Number of delivery partners busy with other tasks at the order placement time.
12. total_outstanding_orders: Total count of orders pending at the time.

Assumptions:

1. Since the number of on-shift partners is less than the number of busy partners, I assume that on-shift partners represent the available delivery partners at the time of the order, while busy partners refer to those who are currently offline
2. The metrics for delivery partners and outstanding orders are indicative at the platform level rather than being specific to individual markets

1 EDA

[108]:
```
df.shape
```

[108]:
```
(197428, 14)
```

[109]:
```
# basic info

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 197428 entries, 0 to 197427
Data columns (total 14 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 196441 non-null  float64
 1   created_at                197428 non-null  object
 2   actual_delivery_time      197421 non-null  object
```

```
 3   store_id                  197428 non-null  object
 4   store_primary_category    192668 non-null  object
 5   order_protocol            196433 non-null  float64
 6   total_items               197428 non-null  int64
 7   subtotal                  197428 non-null  int64
 8   num_distinct_items        197428 non-null  int64
 9   min_item_price            197428 non-null  int64
 10  max_item_price            197428 non-null  int64
 11  total_onshift_partners    181166 non-null  float64
 12  total_busy_partners       181166 non-null  float64
 13  total_outstanding_orders  181166 non-null  float64
dtypes: float64(5), int64(5), object(4)
memory usage: 21.1+ MB
```

[110]:
```
# market id - area where it is situated

df['market_id'].value_counts()
```

[110]:
```
market_id
2.0    55058
4.0    47599
1.0    38037
3.0    23297
5.0    18000
6.0    14450
Name: count, dtype: int64
```

[111]:
```
# booking mode

df = df.rename(columns = {'order_protocol':'mode_of_booking'}) # rename a␣
 ↪feature
df['mode_of_booking'].value_counts()
```

[111]:
```
mode_of_booking
1.0    54725
3.0    53199
5.0    44290
2.0    24052
4.0    19354
6.0      794
7.0       19
Name: count, dtype: int64
```

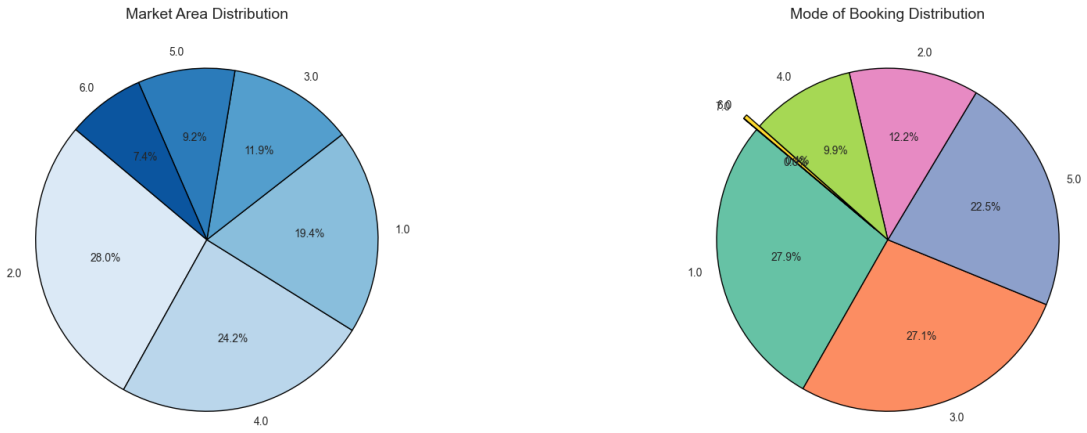
[112]:
```
# no. of unique restaurants

print(df['store_id'].nunique())
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['store'] = le.fit_transform(df['store_id'])
```

6743

Uni-variate Analysis

[113]:
```
# Set figure size
fig, axes = plt.subplots(1, 2, figsize=(20, 6))

# Market Area Pie Chart
area = df['market_id'].value_counts()
colors1 = sns.color_palette('Blues', len(area))  # Improved distinct colors
axes[0].pie(area, labels=area.index, autopct='%1.1f%%', colors=colors1,
            wedgeprops={'edgecolor': 'black', 'linewidth': 1}, startangle=140)
axes[0].set_title('Market Area Distribution', fontsize=14)

# Mode of Booking Pie Chart
mode = df['mode_of_booking'].value_counts()
colors2 = sns.color_palette('Set2', len(mode))

# Improved explode logic: Highlight segments with < 5% share
explode = [0.1 if count / mode.sum() < 0.05 else 0 for count in mode]
axes[1].pie(mode, labels=mode.index, autopct='%1.1f%%', colors=colors2,
            explode=explode, wedgeprops={'edgecolor': 'black', 'linewidth': 1},␣
 ↪startangle=140)
axes[1].set_title('Mode of Booking Distribution', fontsize=14)

# Adjust layout
plt.tight_layout()
plt.show()
```



Market Area Distribution

1. The majority of restaurants (around 70%) are located in market areas 1.0, 2.0, and 4.0.
2. The distribution is visualized using a pie chart, where different shades of blue represent different market areas.

Mode of Payment Distribution

1. About 75% of orders are paid through modes 1.0, 3.0, and 5.0.
2. Modes 6.0 and 7.0 have a negligible share of payments.
3. The pie chart highlights this distribution with pastel colors, and less frequent payment modes are slightly separated (exploded) for better visibility.

These insights help in understanding restaurant distribution and customer payment preferences, which can be valuable for strategic decision-making

[114]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Get top 20 categories
cat_20 = df['store_primary_category'].value_counts().nlargest(20)
top_20 = df[df['store_primary_category'].isin(cat_20.index)]

# Set figure size
plt.figure(figsize=(20,6))

# Improved barplot with a better color palette
ax = sns.barplot(x=cat_20.index, y=cat_20.values, palette='viridis')

# Add title and labels
plt.title('Top 20 Store Categories with Most Orders', fontsize=16,␣
 ↪fontweight='bold')
plt.xlabel('Store Category', fontsize=14)
plt.ylabel('Number of Orders', fontsize=14)
plt.xticks(rotation=45, ha='right')

# Add count labels on top of bars
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width()/2, p.
 ↪get_height()),
                ha='center', va='bottom', fontsize=12, fontweight='bold',␣
 ↪color='black')

# Improve layout
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Data Exploration:

- The dataset contains various restaurant categories, ranging from American, Mexican, and Italian to niche cuisines like Afghan, Peruvian, and Moroccan.
- The store_primary_category column provides a classification of restaurants based on their cuisine type.

Top 20 Most Ordered Restaurant Categories:

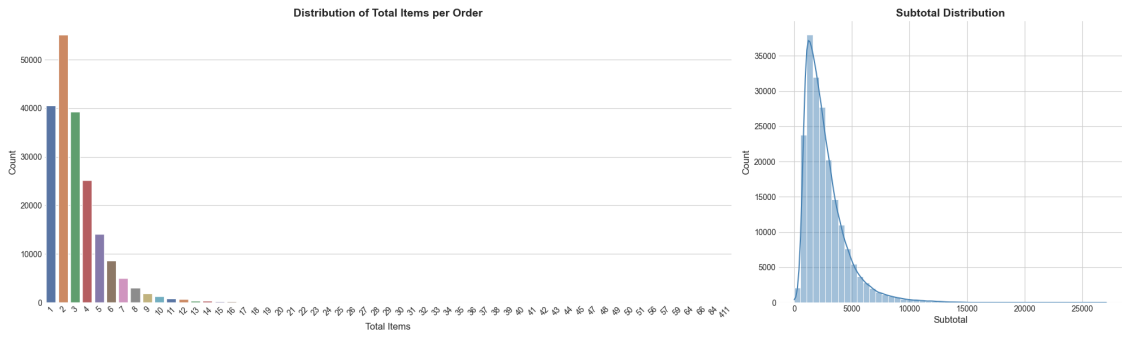
- A count plot was generated to visualize the most popular restaurant categories based on order volume.
- The highest number of orders are from American, Pizza, and Mexican restaurants, indicating strong customer demand for these cuisines.
- Other popular categories include burgers, sandwiches, and fast food, reinforcing the preference for quick-service and comfort food.

Business Insights:

- The findings can help restaurant chains optimize their menu offerings to cater to high-demand cuisines.
- Food delivery platforms can prioritize onboarding restaurants from these categories to maximize customer engagement.
- Further analysis can explore regional variations in cuisine preference, influencing location-based marketing strategies.

[115]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.gridspec as gridspec

# Set the style for better aesthetics
sns.set_style("whitegrid")

# Create figure
fig = plt.figure(figsize=(20, 6))
gs = gridspec.GridSpec(1, 2, width_ratios=[2, 1])

# Total Items Countplot
ax1 = plt.subplot(gs[0])
```

```
sns.countplot(x=df['total_items'], ax=ax1, palette="deep")
ax1.set_title("Distribution of Total Items per Order", fontsize=14,␣
 ↪fontweight='bold')
ax1.set_xlabel("Total Items", fontsize=12)
ax1.set_ylabel("Count", fontsize=12)
ax1.tick_params(axis='x', rotation=45)

# Subtotal Histogram with KDE
ax2 = plt.subplot(gs[1])
sns.histplot(df['subtotal'], bins=50, kde=True, ax=ax2, color='steelblue')
ax2.set_title("Subtotal Distribution", fontsize=14, fontweight='bold')
ax2.set_xlabel("Subtotal", fontsize=12)
ax2.set_ylabel("Count", fontsize=12)

# Improve layout
plt.tight_layout()
plt.show()
```



Total Items per Order:

- The majority of orders contain 5 or fewer items, indicating that most customers place small orders.
- A few large orders exist, likely from corporate or bulk purchases.

Subtotal Distribution:

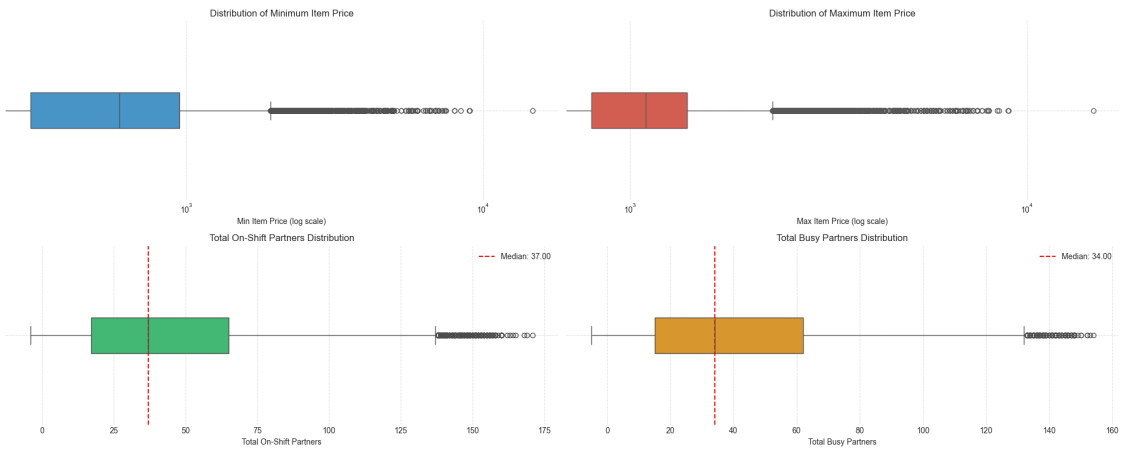
- Most orders have a subtotal below 5000, showing that lower-value transactions dominate.
- The right-skewed distribution suggests the presence of high-value bulk or corporate orders.

Potential Business Insights:

- Since most customers place small orders, strategies like bundling, discounts on larger orders, or free delivery thresholds may encourage higher spending.
- Large corporate/party orders form a minority but significantly impact revenue—loyalty programs or special deals for bulk buyers could be beneficial.

[116]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

plt.figure(figsize=(20, 8))  # Set figure size

# Define color palette
box_colors = ["#3498db", "#e74c3c", "#2ecc71", "#f39c12"]

# Min item price with log scale
plt.subplot(2, 2, 1)
sns.boxplot(x=df['min_item_price'], width=0.2, color=box_colors[0])
plt.xscale("log")  # Log scale to improve visualization
plt.title("Distribution of Minimum Item Price")
plt.xlabel("Min Item Price (log scale)")
plt.grid(True, linestyle="--", alpha=0.6)

# Max item price with log scale
plt.subplot(2, 2, 2)
sns.boxplot(x=df['max_item_price'], width=0.2, color=box_colors[1])
plt.xscale("log")
plt.title("Distribution of Maximum Item Price")
plt.xlabel("Max Item Price (log scale)")
plt.grid(True, linestyle="--", alpha=0.6)

# Compute medians
onshift_median = df['total_onshift_partners'].median()
busy_median = df['total_busy_partners'].median()

# Total on-shift partners
plt.subplot(2, 2, 3)
sns.boxplot(x=df['total_onshift_partners'], width=0.2, color=box_colors[2])
plt.axvline(onshift_median, color='red', linestyle='dashed', label=f'Median:␣
 ↪{onshift_median:.2f}')
plt.legend()
plt.title("Total On-Shift Partners Distribution")
plt.xlabel("Total On-Shift Partners")
plt.grid(True, linestyle="--", alpha=0.6)

# Total busy partners
plt.subplot(2, 2, 4)
sns.boxplot(x=df['total_busy_partners'], width=0.2, color=box_colors[3])
plt.axvline(busy_median, color='red', linestyle='dashed', label=f'Median:␣
 ↪{busy_median:.2f}')
plt.legend()
plt.title("Total Busy Partners Distribution")
plt.xlabel("Total Busy Partners")
```

```
plt.grid(True, linestyle="--", alpha=0.6)

plt.tight_layout()  # Adjust layout to avoid overlap
plt.show()
```



Item Price Distributions:

- The minimum item price is generally low, but there are a few outliers indicating certain exceptionally high-priced items.
- The maximum item price shows a wider spread, with some extreme values that may need further investigation (e.g., premium or bulk items).

Partner Availability Trends:

- The median number of on-shift delivery partners at the time of an order is 37, meaning these partners are actively available for deliveries.
- The median number of busy delivery partners (i.e., offline or unavailable) is 34, indicating that a significant portion of the workforce is not actively fulfilling orders at a given time.
- The wide range (0 to 180) suggests fluctuations, possibly influenced by demand patterns, time of day, or operational constraints.

[117]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Print summary statistics
print(df['total_outstanding_orders'].describe())

# Define figure size
plt.figure(figsize=(20,3))

# Create boxplot with improved aesthetics
ax = sns.boxplot(x=df['total_outstanding_orders'], width=0.1, color='skyblue',␣
 ↪showfliers=True)
```

```
# Highlight median with a dashed red line
median_value = df['total_outstanding_orders'].median()
plt.axvline(median_value, color='red', linestyle='dashed', label=f'Median:␣
 ↪{median_value:.2f}')

# Add title and labels
plt.title('Distribution of Total Outstanding Orders', fontsize=16,␣
 ↪fontweight='bold')
plt.xlabel('Total Outstanding Orders', fontsize=14)

# Improve layout
plt.legend()
plt.grid(axis='x', linestyle='--', alpha=0.6)
plt.show()
```

```
count    181166.000000
mean         58.050065
std          52.661830
min          -6.000000
25%          17.000000
50%          41.000000
75%          85.000000
max         285.000000
Name: total_outstanding_orders, dtype: float64
```

- The median number of outstanding orders across the platform is 41, meaning that at any given time, half of the outstanding orders are below this level
- While most values are within 0 to 100, there are extreme cases where outstanding orders exceed 200+. This suggests that the platform experiences occasional demand surges

Potential Operational Impacts:

- **Peak Load Management:** The platform needs to handle sudden spikes in outstanding orders, possibly due to high-demand hours or large-scale promotions.
- **Service Optimization:** Outlier cases where orders exceed 200+ outstanding may indicate capacity constraints, leading to potential delays.
- **Scalability Consideration:** If high outstanding order counts are frequent, the platform may need to scale resources (e.g., more delivery partners, server capacity, or optimized dispatching

algorithms).

Extracting Features

[118]:
```
# extracting time, day of the week

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

df['time_diff'] = df['actual_delivery_time'] - df['created_at']
df['order_placed'] = df['created_at'].dt.strftime('%H:%M') # order created
df['order_delivered'] = df['actual_delivery_time'].dt.strftime('%H:%M') #␣
 ↪ordered delivered
df['hour_of_the_day'] = df['created_at'].dt.hour
df['delivery_time_mins'] = round(df['time_diff'].dt.total_seconds()/60) #␣
 ↪delivery time in minutes
df['day_of_the_week'] = df['actual_delivery_time'].dt.day_name() # day of the␣
 ↪week

df.head(1)
```

[118]:
```
   market_id          created_at actual_delivery_time  \
0        1.0 2015-02-06 22:24:17  2015-02-06 23:27:16

                           store_id store_primary_category  mode_of_booking  \
0  df263d996281d984952c07998dc54358               american              1.0

   total_items  subtotal  num_distinct_items  min_item_price  …  \
0            4      3441                   4             557  …

   total_onshift_partners  total_busy_partners  total_outstanding_orders  \
0                    33.0                 14.0                      21.0

   store       time_diff order_placed order_delivered hour_of_the_day  \
0   5913 0 days 01:02:59        22:24           23:27              22

   delivery_time_mins  day_of_the_week
0                63.0           Friday

[1 rows x 21 columns]
```

[119]:
```
plt.figure(figsize=(20,5))

# Orders per hour of the day
plt.subplot(1,3,1)
sns.countplot(x=df['hour_of_the_day'], palette='coolwarm', alpha=0.8)
plt.title('Order Volume by Hour of the Day')
plt.xlabel('Hour of the Day')
```

```
plt.ylabel('Number of Orders')

# Distribution of Delivery Time
plt.subplot(1,3,2)
sns.boxplot(x=df['delivery_time_mins'], color='lightblue', showfliers=True)
plt.title('Distribution of Delivery Time')
plt.xlabel('Delivery Time (minutes)')

# Orders per Day of the Week
plt.subplot(1,3,3)
order = df['day_of_the_week'].value_counts().index
sns.countplot(x=df['day_of_the_week'], order=order, palette='Set2', alpha=0.8)
plt.title('Order Volume by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Orders')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

print(df['delivery_time_mins'].describe())
```


```
count    197421.000000
mean         48.470492
std         320.494384
min           2.000000
25%          35.000000
50%          44.000000
75%          56.000000
max      141948.000000
Name: delivery_time_mins, dtype: float64
```

Delivery Time Analysis: Hourly and Daily Trends

1. **Peak Ordering Hours**
   - High order volume observed between 1 AM - 3 AM and 7 PM - 9 PM.
   - The 2 AM order count is double that of 8 PM, indicating a late-night rush.

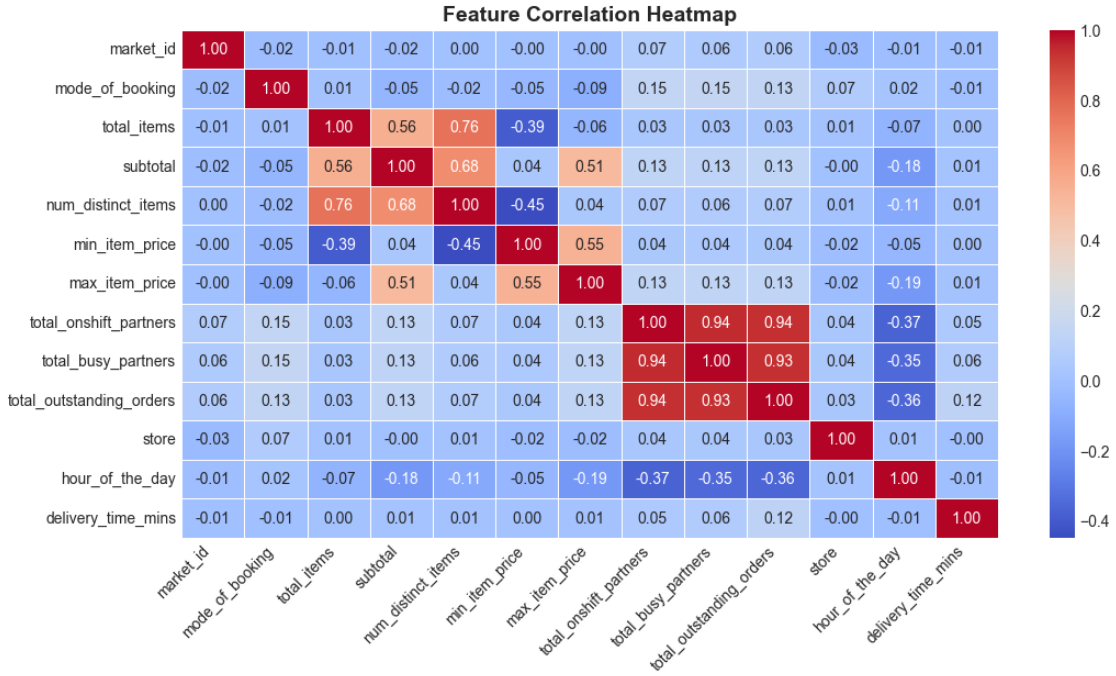
2. **Delivery Time Distribution**
    - Average delivery time: ~48 minutes
    - Median delivery time: 44 minutes
    - Max delivery time (~141,948 mins) is an extreme outlier.
3. **Order Volume by Day of the Week**
    - Higher order volume on weekends (Saturday & Sunday) compared to weekdays.
    - Order count gradually decreases from Monday to Wednesday.

Key Takeaways:

The late-night and evening peak hours suggest staffing adjustments may be needed for better efficiency. The presence of extreme delivery time outliers highlights the importance of data cleaning. Weekends drive higher demand, indicating a potential opportunity for promotions or increased workforce allocation

Bi-variate Analysis

[120]:
```
# Compute correlation matrix for numerical features
df_corr = df.select_dtypes(include=['int32', 'int64', 'float64']).corr()

# Set up the figure
plt.figure(figsize=(12,6))

# Create heatmap with improved aesthetics
sns.heatmap(df_corr, annot=True, fmt=".2f", cmap="coolwarm", linewidths=0.5,␣
 ↪annot_kws={"size": 10})

# Improve readability of x-axis labels
plt.xticks(rotation=45, ha='right')
plt.title("Feature Correlation Heatmap", fontsize=14, fontweight='bold')

plt.show()
```



Strong Correlation Between:

- total_items & subtotal (0.76) → Higher item count leads to a higher subtotal.
- total_onshift_partners, total_busy_partners, and total_outstanding_orders (>0.90) → More outstanding orders relate to a high number of busy partners.

Negative Correlation:

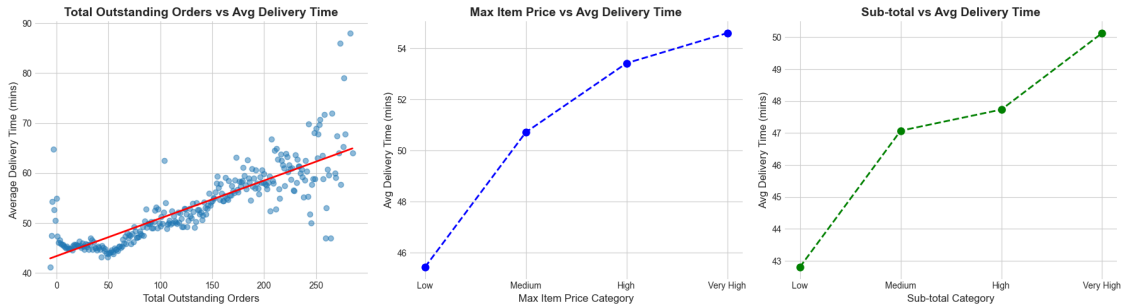
- num_distinct_items & total_items (-0.39) → Orders with diverse items tend to have fewer total items.
- hour_of_the_day & total_busy_partners (-0.35) → Fewer busy partners at certain hours, likely late night.

Weak Correlation with Delivery Time:

- delivery_time_mins has low correlation with most variables, indicating other external factors (e.g., traffic, weather) might impact delivery time.

[121]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(18,5))

# **1. Total Outstanding Orders vs Avg Delivery Time**
plt.subplot(1,3,1)
z = df.groupby('total_outstanding_orders')['delivery_time_mins'].mean()
sns.regplot(x=z.index, y=z.values, order=1, ci=None,
```

```
            scatter_kws={'alpha':0.5}, line_kws={'color':'r', 'linewidth':2})
plt.xlabel('Total Outstanding Orders', fontsize=12)
plt.ylabel('Average Delivery Time (mins)', fontsize=12)
plt.title('Total Outstanding Orders vs Avg Delivery Time', fontsize=14,␣
 ↪fontweight='bold')

# **2. Max Item Price vs Avg Delivery Time**
bin_edges = [0, 1000, 2000, 3000, 4000]
df["max_item_price_bins"] = pd.cut(df["max_item_price"],
                                   bins=bin_edges, labels=["Low", "Medium",␣
 ↪"High", "Very High"], include_lowest=True)
s = df.groupby(['max_item_price_bins'])['delivery_time_mins'].mean()

plt.subplot(1,3,2)
plt.plot(s, marker='o', linestyle='--', color='blue', markersize=8, linewidth=2)
plt.xlabel('Max Item Price Category', fontsize=12)
plt.ylabel('Avg Delivery Time (mins)', fontsize=12)
plt.title('Max Item Price vs Avg Delivery Time', fontsize=14, fontweight='bold')
plt.grid(True)

# **3. Sub-total vs Avg Delivery Time**
df["total_bins"] = pd.cut(df["subtotal"], bins=bin_edges, labels=["Low",␣
 ↪"Medium", "High", "Very High"], include_lowest=True)
s = df.groupby(['total_bins'])['delivery_time_mins'].mean()

plt.subplot(1,3,3)
plt.plot(s, marker='o', linestyle='--', color='green', markersize=8,␣
 ↪linewidth=2)
plt.xlabel('Sub-total Category', fontsize=12)
plt.ylabel('Avg Delivery Time (mins)', fontsize=12)
plt.title('Sub-total vs Avg Delivery Time', fontsize=14, fontweight='bold')
plt.grid(True)

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



Total Outstanding Orders vs. Average Delivery Time

- Delivery time remains stable for outstanding orders up to 50.
- Beyond 50 orders, delivery time increases linearly.
- When outstanding orders exceed 200, delivery time rises non-linearly, indicating congestion effects

Max Item Price vs. Average Delivery Time

- Higher-priced items tend to have longer delivery times.
- The relationship follows a gradual upward trend, showing a linear increase in delivery time with max item price

Sub-total vs. Average Delivery Time

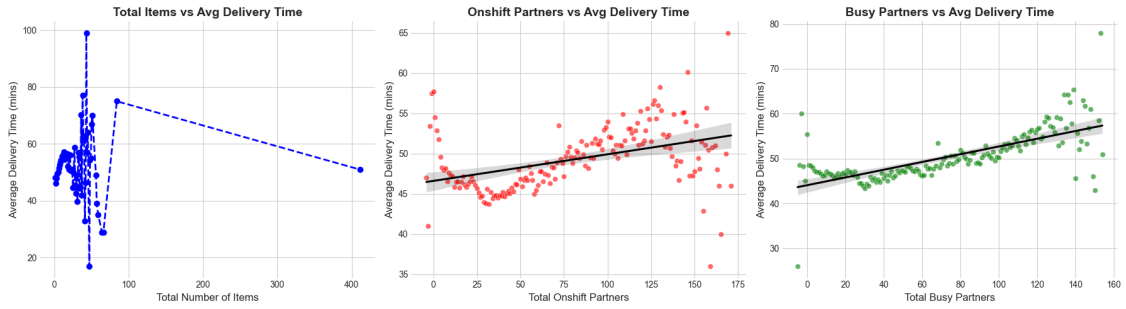
- Low sub-totals have the shortest delivery times.
- A significant jump in delivery time occurs from low to medium sub-totals.
- As sub-total increases further, delivery time continues rising linearly

Insights & Implications:

- High outstanding orders lead to delivery delays, suggesting the need for dynamic fleet allocation.
- Orders with expensive items or higher sub-totals take longer, potentially due to complex preparation or packaging.
- Businesses can optimize logistics by prioritizing order fulfillment based on outstanding orders and pricing factors.

[122]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(18,5))

# **1. Total Items vs Avg Delivery Time**
plt.subplot(1,3,1)
a = df.groupby('total_items')['delivery_time_mins'].mean()
plt.plot(a, marker='o', linestyle='--', color='blue', markersize=6, linewidth=2)
plt.xlabel('Total Number of Items', fontsize=12)
plt.ylabel('Average Delivery Time (mins)', fontsize=12)
plt.title('Total Items vs Avg Delivery Time', fontsize=14, fontweight='bold')
plt.grid(True)

# **2. On-Shift Partners vs Avg Delivery Time**
plt.subplot(1,3,2)
avg_delivery_time_onshift = df.
 ↪groupby('total_onshift_partners')['delivery_time_mins'].mean()
sns.scatterplot(x=avg_delivery_time_onshift.index, y=avg_delivery_time_onshift.
 ↪values, color='red', alpha=0.6)
sns.regplot(x=avg_delivery_time_onshift.index, y=avg_delivery_time_onshift.
 ↪values, scatter=False, color='black')
```

```
plt.xlabel('Total Onshift Partners', fontsize=12)
plt.ylabel('Average Delivery Time (mins)', fontsize=12)
plt.title('Onshift Partners vs Avg Delivery Time', fontsize=14,␣
 ↪fontweight='bold')

# **3. Busy Partners vs Avg Delivery Time**
plt.subplot(1,3,3)
avg_delivery_time_busy = df.
 ↪groupby('total_busy_partners')['delivery_time_mins'].mean()
sns.scatterplot(x=avg_delivery_time_busy.index, y=avg_delivery_time_busy.
 ↪values, color='green', alpha=0.6)
sns.regplot(x=avg_delivery_time_busy.index, y=avg_delivery_time_busy.values,␣
 ↪scatter=False, color='black')
plt.xlabel('Total Busy Partners', fontsize=12)
plt.ylabel('Average Delivery Time (mins)', fontsize=12)
plt.title('Busy Partners vs Avg Delivery Time', fontsize=14, fontweight='bold')

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



Total Items vs Avg Delivery Time:

- The delivery time fluctuates with the number of items.
- Large or complex orders may take longer to process.

On-Shift Partners vs Avg Delivery Time:

- When the number of on-shift partners increases up to 30, delivery time decreases (more availability).
- Beyond 30 on-shift partners, delivery time increases, possibly due to inefficient allocation, delays in batching, or longer wait times at restaurants.

Busy (Offline) Partners vs Avg Delivery Time:

- More busy partners (offline) → longer delivery time.
- As the number of offline partners increases, the available fleet shrinks, leading to longer delivery times.

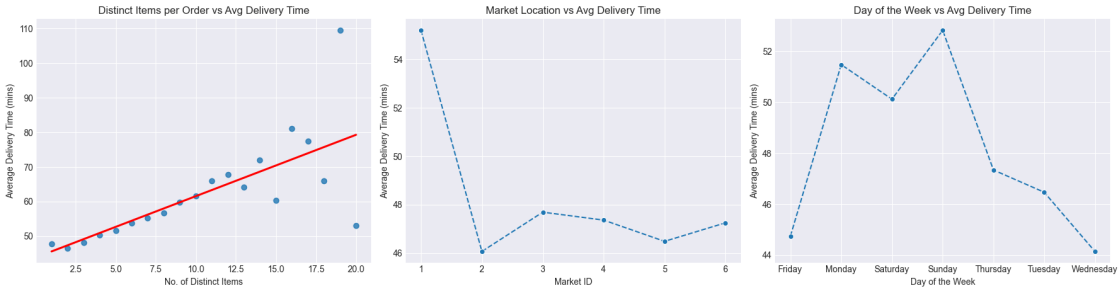
- If too many partners are offline, it can indicate high demand with insufficient supply, causing delays.

Actionable Steps for Optimization:

- Monitor busy (offline) partners in real-time to balance demand-supply.
- Incentivize drivers to stay online during peak hours.
- Improve dispatching logic to allocate orders more efficiently when many drivers are offline.

Categorical features

[123]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

plt.style.use("seaborn-v0_8-darkgrid")  # Apply consistent style

# Set up the figure
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Distinct items vs Delivery Time
z = df.groupby("num_distinct_items")["delivery_time_mins"].mean()
sns.regplot(x=z.index, y=z.values, order=1, ci=None, line_kws={"color": "red"},␣
 ↪ax=axes[0])
axes[0].set_title("Distinct Items per Order vs Avg Delivery Time", fontsize=12)
axes[0].set_xlabel("No. of Distinct Items")
axes[0].set_ylabel("Average Delivery Time (mins)")

# Market Location vs Delivery Time
z = df.groupby("market_id")["delivery_time_mins"].mean()
sns.lineplot(x=z.index, y=z.values, marker="o", linestyle="--", ax=axes[1])
axes[1].set_title("Market Location vs Avg Delivery Time", fontsize=12)
axes[1].set_xlabel("Market ID")
axes[1].set_ylabel("Average Delivery Time (mins)")

# Day of the Week vs Delivery Time
z = df.groupby("day_of_the_week")["delivery_time_mins"].mean()
sns.lineplot(x=z.index, y=z.values, marker="o", linestyle="--", ax=axes[2])
axes[2].set_title("Day of the Week vs Avg Delivery Time", fontsize=12)
axes[2].set_xlabel("Day of the Week")
axes[2].set_ylabel("Average Delivery Time (mins)")

plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout
plt.show()
```



Distinct Number of Items vs Avg Delivery Time:

- Linear increase: More distinct items in an order → longer delivery time.
- Possible reason: Increased complexity in order preparation.

Market Location (Market ID) vs Avg Delivery Time:

- Market ID 1 has the highest avg delivery time (~54 mins).
- Other market locations have a more balanced delivery time (~46 mins).
- Suggests possible inefficiencies in Market ID 1 (e.g., traffic, order volume, partner availability).

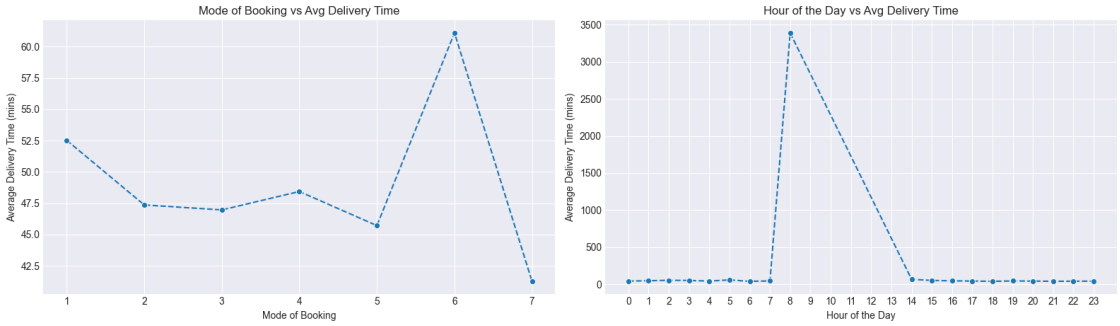
Day of the Week vs Avg Delivery Time:

- Peak delay on Sundays (~53 mins). Likely due to high order volume & fewer drivers.
- Weekday efficiency: Avg delivery time improves from Monday to Wednesday.
- Thursday → Sunday trend: Increase in delivery time, likely due to weekend rush.

Optimization Suggestions: * For Market ID 1: Investigate traffic, staffing, and partner availability. * For weekends: Increase driver incentives or dynamic dispatching.

[124]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas as pd

plt.style.use("seaborn-v0_8-darkgrid")  # Apply consistent style

# Set up figure
fig, axes = plt.subplots(1, 2, figsize=(16, 5))

# Mode of Booking vs Avg Delivery Time
z = df.groupby("mode_of_booking")["delivery_time_mins"].mean()
sns.lineplot(x=z.index, y=z.values, marker="o", linestyle="--", ax=axes[0])
axes[0].set_title("Mode of Booking vs Avg Delivery Time", fontsize=12)
axes[0].set_xlabel("Mode of Booking")
axes[0].set_ylabel("Average Delivery Time (mins)")
axes[0].grid(True)

# Hour of the Day vs Avg Delivery Time
```

```
a = df.groupby("hour_of_the_day")["delivery_time_mins"].mean()
sns.lineplot(x=a.index, y=a.values, marker="o", linestyle="--", ax=axes[1])
axes[1].set_title("Hour of the Day vs Avg Delivery Time", fontsize=12)
axes[1].set_xlabel("Hour of the Day")
axes[1].set_ylabel("Average Delivery Time (mins)")
axes[1].set_xticks(np.arange(0, df["hour_of_the_day"].max() + 1, 1))
axes[1].grid(True)

plt.tight_layout(rect=[0, 0, 1, 0.95])  # Adjust layout
plt.show()
```



Mode of Booking vs Avg Delivery Time:

- Some booking modes may be slower than others.
- Possible reasons: Different processing times or platform efficiency.

Hour of the Day vs Avg Delivery Time:

- Peak delays observed early morning (6-7 AM).
- Possible reasons: Limited delivery partners, high demand, or batch processing.
- Delivery times reduce significantly after early hours.

2 Data Pre-Processing

[125]:
```
# making a copy of the dataframe

df1 = df.copy()
df1 = df1.drop(columns = ['created_at', 'store_id',␣
 ↪'actual_delivery_time','time_diff','order_placed',
                           'order_delivered','max_item_price_bins','total_bins'])
```

Missing values

[126]:
```
# percentage of missing values

round(df1.isna().sum()/df1.shape[0] * 100,3)
```

[126]:
```
market_id                   0.500
store_primary_category      2.411
mode_of_booking             0.504
total_items                 0.000
subtotal                    0.000
num_distinct_items          0.000
min_item_price              0.000
max_item_price              0.000
total_onshift_partners      8.237
total_busy_partners         8.237
total_outstanding_orders    8.237
store                       0.000
hour_of_the_day             0.000
delivery_time_mins          0.004
day_of_the_week             0.004
dtype: float64
```

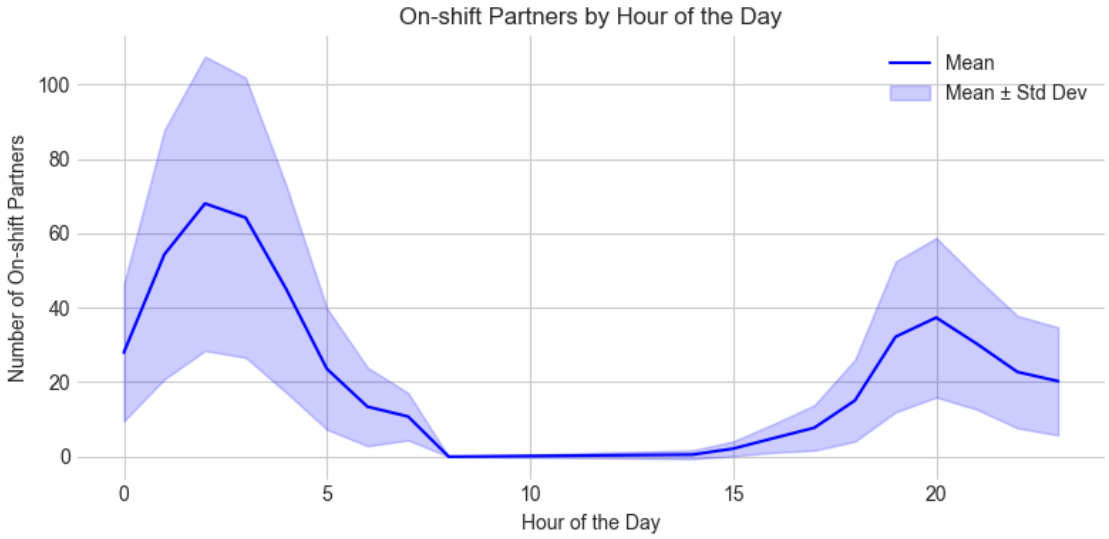
Analysis of On-Shift Partners by Hour of the Day

Objective: To understand the variation in the number of on-shift delivery partners throughout the day and determine an appropriate imputation strategy for missing values.

Visualization - The graph below represents: Mean number of on-shift partners (blue line) at each hour of the day. Standard deviation (±1) shaded region, indicating the variance in the number of on-shift partners.

[127]:
```
# getting mean & std dev

df1_stat = df1.groupby(['hour_of_the_day']).agg({'total_onshift_partners':
 ↪['mean','std'], 'total_busy_partners':['mean','std'],␣
 ↪'total_outstanding_orders':['mean','std']}).reset_index()
df1_stat.columns = ['_'.join(col) for col in df1_stat.columns]

df1 = df1.merge(df1_stat, left_on = 'hour_of_the_day', right_on =␣
 ↪'hour_of_the_day_', how = 'left')

# variance in data
sns.set_style("whitegrid")

plt.figure(figsize=(8, 4))

mean = df1_stat['total_onshift_partners_mean']
std = df1_stat['total_onshift_partners_std']

# Plot the mean
plt.plot(df1_stat['hour_of_the_day_'], mean, color="blue", label="Mean")

# Fill between (mean - std) and (mean + std)
```

```
plt.fill_between(df1_stat['hour_of_the_day_'], mean - std, mean + std,
                 color="blue", alpha=0.2, label="Mean ± Std Dev")

# Labels and title
plt.title("On-shift Partners by Hour of the Day")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of On-shift Partners")
plt.legend()

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



Key Observations

The number of on-shift partners peaks during early morning and evening hours, suggesting high demand during these times. A significant drop in on-shift partners is observed between 10 AM - 3 PM, likely due to lower demand during non-peak hours. Variance is high during peak hours, indicating fluctuations in workforce availability

Imputation Strategy

Assuming the number of on-shift partners follows a normal distribution, missing values will be imputed using the ***mean ± standard deviation***. This ensures that missing values retain realistic variance and align with expected workforce availability trends

[128]:
```
# drawing random value from derived distribution

def impute_values(row, col):
    mean_col = f'{col}_mean'
    std_col = f'{col}_std'
```

```
    if pd.isna(row[col]):
        mean_val = row[mean_col]
        std_val = row[std_col]
        if pd.isna(std_val) or std_val == 0:
            return mean_val
        else:
            return int(np.random.normal(mean_val, std_val))
    else:
        return row[col]

cols_to_impute =␣
 ↪['total_onshift_partners','total_busy_partners','total_outstanding_orders']
for col in cols_to_impute:
    df1[col] = df1.apply(lambda row: impute_values(row, col), axis = 1)
```

[129]:
```
# dropping rows

df1 = df1.dropna()
df1 = df1.drop(columns =␣
 ↪['hour_of_the_day_','total_onshift_partners_mean','total_onshift_partners_std',
                          ␣
 ↪'total_busy_partners_mean','total_busy_partners_std','total_outstanding_orders_mean',
                           'total_outstanding_orders_std'])
df1.isna().sum()
```

[129]:
```
market_id                 0
store_primary_category    0
mode_of_booking           0
total_items               0
subtotal                  0
num_distinct_items        0
min_item_price            0
max_item_price            0
total_onshift_partners    0
total_busy_partners       0
total_outstanding_orders  0
store                     0
hour_of_the_day           0
delivery_time_mins        0
day_of_the_week           0
dtype: int64
```

Analysis of Zero Values in the Dataset

To ensure data integrity and improve model performance, we analyzed the percentage of zero values in each feature

[130]:
```
(df1==0).sum()/df1.shape[0]*100
```

[130]:
```
market_id                 0.000000
store_primary_category    0.000000
mode_of_booking           0.000000
total_items               0.000000
subtotal                  0.091183
num_distinct_items        0.000000
min_item_price            1.303661
max_item_price            0.003647
total_onshift_partners    1.941424
total_busy_partners       2.229563
total_outstanding_orders  2.191006
store                     0.060963
hour_of_the_day           6.409408
delivery_time_mins        0.000000
day_of_the_week           0.000000
dtype: float64
```

- Features That Should Not Contain Zero Values: **sub total, min item price, max item price, on shift partners, busy partners, total outstanding orders**
- These features can have zero values due to their nature: **hour of the day & store - categorical variable**

Assumptions for Data Cleaning

To ensure data consistency, we assume:

1. subtotal, min_item_price, and max_item_price must be greater than 0 (they cannot be zero or negative)
2. total_onshift_partners must be greater than 0 (there must be at least one delivery partner available)
3. total_busy_partners must be non-negative (a negative value would be invalid)
4. total_outstanding_orders must be non-negative (orders cannot be negative)

[131]:
```
# removing rows which contain zero

df1 = df1[(df1['total_onshift_partners'] > 0) & (df1['total_busy_partners'] >␣
 ↪0) & (df1['total_outstanding_orders'] > 0)]
df1 = df1[(df1['subtotal'] > 0) & (df1['min_item_price'] > 0) &␣
 ↪(df1['max_item_price'] > 0)]
print((df1==0).sum()/df1.shape[0]*100)
```

```
market_id                 0.000000
store_primary_category    0.000000
mode_of_booking           0.000000
total_items               0.000000
subtotal                  0.000000
num_distinct_items        0.000000
```

```
min_item_price              0.000000
max_item_price              0.000000
total_onshift_partners      0.000000
total_busy_partners         0.000000
total_outstanding_orders    0.000000
store                       0.063811
hour_of_the_day             6.411385
delivery_time_mins          0.000000
day_of_the_week             0.000000
dtype: float64
```

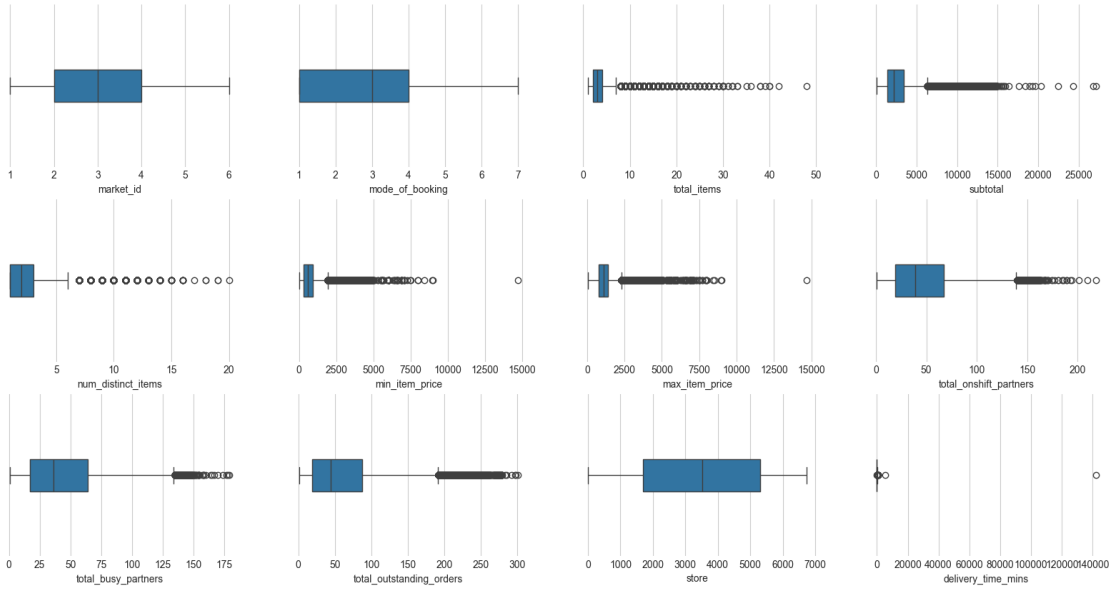
Outlier detection

[132]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 181786 entries, 0 to 197427
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   market_id                 181786 non-null  float64
 1   store_primary_category    181786 non-null  object
 2   mode_of_booking           181786 non-null  float64
 3   total_items               181786 non-null  int64
 4   subtotal                  181786 non-null  int64
 5   num_distinct_items        181786 non-null  int64
 6   min_item_price            181786 non-null  int64
 7   max_item_price            181786 non-null  int64
 8   total_onshift_partners    181786 non-null  float64
 9   total_busy_partners       181786 non-null  float64
 10  total_outstanding_orders  181786 non-null  float64
 11  store                     181786 non-null  int64
 12  hour_of_the_day           181786 non-null  int32
 13  delivery_time_mins        181786 non-null  float64
 14  day_of_the_week           181786 non-null  object
dtypes: float64(6), int32(1), int64(6), object(2)
memory usage: 21.5+ MB
```

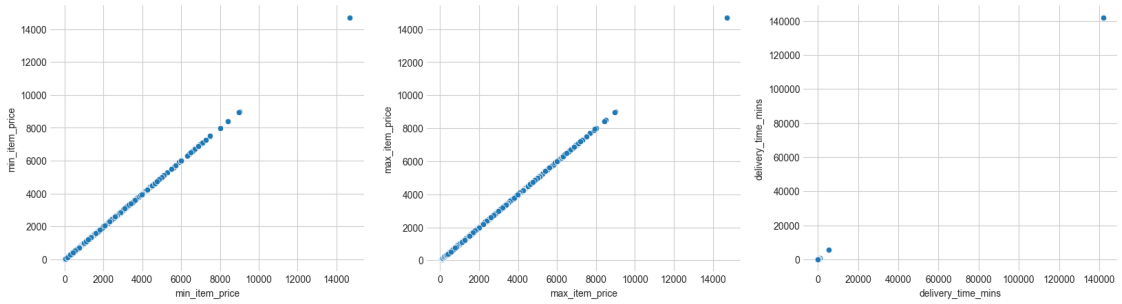
[133]:
```
# looking for outliers in features

import math

plt.figure(figsize=(20,10))

num_feat = df1.select_dtypes(include = ['float64', 'int64']).columns

# grid structure
num_plots = len(num_feat)
```

```
cols = 4
rows = math.ceil(num_plots/cols)

for j,i in enumerate(num_feat, 1):
    plt.subplot(rows, cols, j)
    sns.boxplot(x = df1[i], width = 0.2)

plt.show()
```



- Following features could have outliers:
  1. Minimum item price
  2. Maximum item price
  3. Delivery time

[134]:
```
# outliers in min & max item price

plt.figure(figsize = (20,5))

plt.subplot(1,3,1)
sns.scatterplot(x = df1['min_item_price'], y = df1['min_item_price'])

plt.subplot(1,3,2)
sns.scatterplot(x = df1['max_item_price'], y = df1['max_item_price'])

plt.subplot(1,3,3)
sns.scatterplot(x = df1['delivery_time_mins'], y = df1['delivery_time_mins'])

plt.show()
```



[135]:
```
# treating outliers

df1 = df1[df1['min_item_price'] < df1['min_item_price'].max()]

df1 = df1[df1['min_item_price'] < df1['min_item_price'].max()]

df1 = df1[df1['delivery_time_mins'] < df1['delivery_time_mins'].max()]
```

Negative values in data

[136]:
```
# rows with -ve values

df1_negative = df1.select_dtypes(include=['int32', 'int64','float64'])
df1_negative[df1_negative < 0].sum()
```

[136]:
```
market_id                 0.0
mode_of_booking           0.0
total_items               0.0
subtotal                  0.0
num_distinct_items        0.0
min_item_price            0.0
max_item_price            0.0
total_onshift_partners    0.0
total_busy_partners       0.0
total_outstanding_orders  0.0
store                     0.0
hour_of_the_day           0.0
delivery_time_mins        0.0
dtype: float64
```

3 Model Building

[77]:
```
df1.head(2)
```

[77]:
```
   market_id store_primary_category  mode_of_booking  total_items  subtotal  \
0        1.0                american              1.0            4      3441
1        2.0                 mexican              2.0            1      1900

   num_distinct_items  min_item_price  max_item_price  total_onshift_partners  \
0                   4             557            1239                    33.0
1                   1            1400            1400                     1.0

   total_busy_partners  total_outstanding_orders  store  hour_of_the_day  \
0                 14.0                      21.0   5913               22
1                  2.0                       2.0   6346               21

   delivery_time_mins day_of_the_week
0                63.0          Friday
1                67.0         Tuesday
```

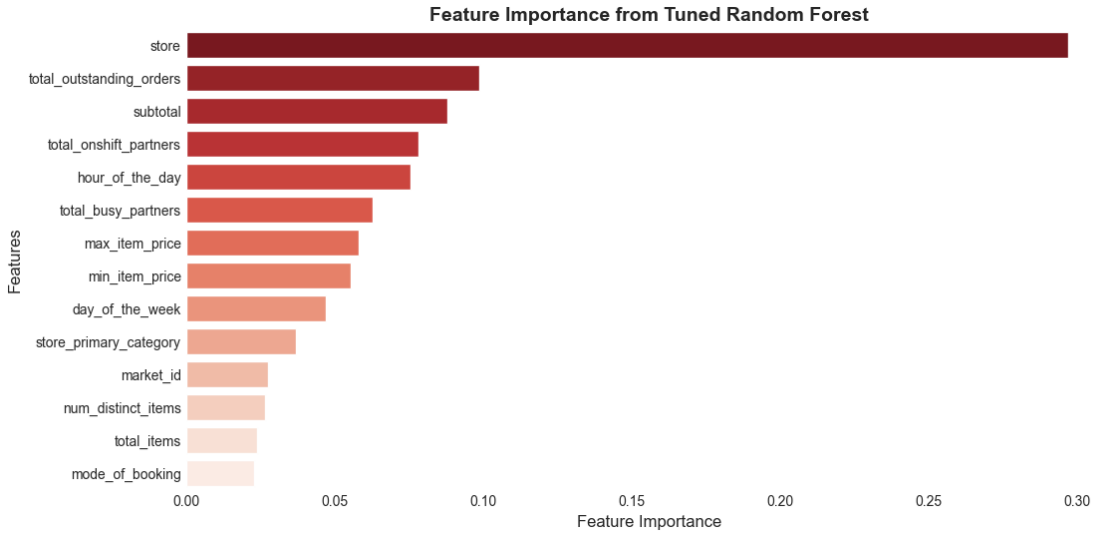
[78]:
```
# train - test split
from sklearn.model_selection import train_test_split

x = df1.drop(columns = ['delivery_time_mins'])
y = df1['delivery_time_mins']

x_tr,x_test, y_tr, y_test = train_test_split(x,y,test_size = 0.2, random_state␣
 ↪= 42)
x_train,x_val, y_train, y_val = train_test_split(x_tr,y_tr,test_size = 0.2,␣
 ↪random_state = 42)

# target encoding categorical columns: 'market id', 'mode of booking', 'hour of␣
 ↪the day', 'day of the week'

import category_encoders as ce

cat_cols = ["market_id", "store_primary_category", "mode_of_booking", "store",␣
 ↪"hour_of_the_day","day_of_the_week"]

encoder = ce.TargetEncoder(cols= cat_cols)

# fit the encoder
encoder.fit(x_train[cat_cols], y_train)

# apply transformation
x_train[cat_cols] = encoder.transform(x_train[cat_cols]).round(2)
x_val[cat_cols] = encoder.transform(x_val[cat_cols]).round(2)
x_test[cat_cols] = encoder.transform(x_test[cat_cols]).round(2)
```

Baseline Model

[79]:
```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Define hyperparameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [50, 100, 200, 300],  # Number of trees
    'max_depth': [None, 10, 20, 30, 40],  # Depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples required at each leaf node
    'max_features': ['sqrt', 'log2'],  # Number of features considered for␣
 ↪splitting
    'bootstrap': [True, False]  # Whether to use bootstrapping
}

# Initialize Random Forest model
rf = RandomForestRegressor(random_state=42)

# Initialize Randomized Search CV
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=12,  # Number of random combinations to try
    scoring='neg_mean_squared_error',  # Scoring metric
    cv=3,  # 5-fold cross-validation
    verbose=2,  # Show search progress
    n_jobs=-1,  # Use all available cores
    random_state=42
)

# Perform hyperparameter tuning
random_search.fit(x_train, y_train)

# Print best parameters
print("Best Hyperparameters:", random_search.best_params_)

# Train model with best parameters
best_rf = random_search.best_estimator_

# Make predictions
y_pred_val = best_rf.predict(x_val)

# Evaluate performance
mae = mean_absolute_error(y_val, y_pred_val)
print(f"Mean Absolute Error: {mae}")

mse = mean_squared_error(y_val, y_pred_val)
```

```
print(f"Mean Squared Error: {mse}")

r2 = r2_score(y_val, y_pred_val)
print(f"R2 score: {r2}")

# Feature Importance Visualization
feature_importances = pd.DataFrame({'Feature': x_train.columns, 'Importance':␣
 ↪best_rf.feature_importances_})
feature_importances = feature_importances.sort_values(by="Importance",␣
 ↪ascending=False)

# Plot feature importance
plt.figure(figsize=(12, 6))
sns.barplot(
    x=feature_importances["Importance"],
    y=feature_importances["Feature"],
    palette="Reds_r",
)

# Labels and Title
plt.xlabel("Feature Importance", fontsize=12)
plt.ylabel("Features", fontsize=12)
plt.title("Feature Importance from Tuned Random Forest", fontsize=14,␣
 ↪fontweight="bold")

# Remove grid for a cleaner look
plt.grid(False)

plt.show()
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best Hyperparameters: {'n_estimators': 300, 'min_samples_split': 2,
'min_samples_leaf': 4, 'max_features': 'log2', 'max_depth': None, 'bootstrap':
True}
Mean Absolute Error: 11.132566784962455
Mean Squared Error: 269.7879291837616
R2 score: 0.25370498289213705
```



Most Influential Features

- store: The most important feature, indicating that the store location has a strong influence on delivery time. This could be due to store-specific processing times or logistics.
- total_outstanding_orders: A significant factor—suggests that backlog at the store directly impacts delivery times.
- subtotal: The total order value has a notable impact, possibly affecting prioritization or batch processing.
- total_onshift_partners & total_busy_partners: The number of available and busy delivery partners plays a role in estimating delivery time.
- hour_of_the_day & day_of_the_week: Time-based effects indicate that peak hours likely result in longer delivery times.

Less Important Features

- mode_of_booking & total_items: These have lower importance, suggesting that how an order is booked (e.g., via app or website) and the total item count are not strong predictors of delivery time.

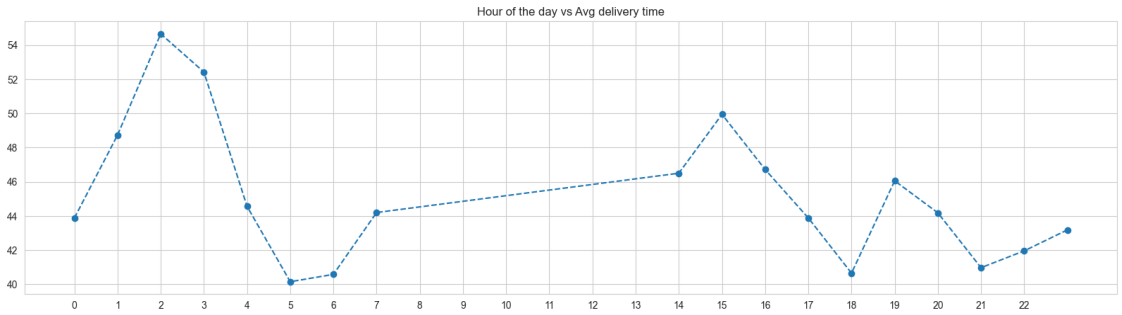
Model Performance Observations

- Mean Absolute Error (MAE): ~11.13 mins
- Mean Squared Error (MSE): ~269.79
- R^2 Score: ~0.2537 (low, meaning the model explains only ~25% of the variance)

Advance Model

[44]:
```
# peak hours

plt.figure(figsize = (20,5))

a = df1.groupby(['hour_of_the_day'])['delivery_time_mins'].mean()
```

```
plt.plot( a.index, a.values, marker='o', linestyle='--')
plt.title('Hour of the day vs Avg delivery time')
x_ticks = np.arange(0,df['hour_of_the_day'].max(),1)
plt.xticks(ticks = x_ticks)
plt.grid(True)

plt.show()
```



[81]:
```
# creating flags
peak_hours = [1, 2, 3, 15]
df1['peak'] = np.where(df1['hour_of_the_day'].isin(peak_hours), 'Peak',␣
 ↪'Non-Peak') # peak / non peak hours

weekend = ['Saturday','Sunday']
df1['weekend'] = np.where(df1['day_of_the_week'].isin(weekend), 'weekend',␣
 ↪'weekday') # weekend / weekday

# train - test split
from sklearn.model_selection import train_test_split

x = df1.drop(columns = ['delivery_time_mins'])
y = df1['delivery_time_mins']

x_tr,x_test, y_tr, y_test = train_test_split(x,y,test_size = 0.2, random_state␣
 ↪= 42)
x_train,x_val, y_train, y_val = train_test_split(x_tr,y_tr,test_size = 0.2,␣
 ↪random_state = 42)

import category_encoders as ce

cat_cols = ["market_id", "store_primary_category",␣
 ↪"mode_of_booking","hour_of_the_day","day_of_the_week", "store", "peak",␣
 ↪"weekend"]

encoder = ce.TargetEncoder(cols= cat_cols) # target encoding
```

```
# fit the encoder
encoder.fit(x_train[cat_cols], y_train)

# apply transformation
x_train[cat_cols] = encoder.transform(x_train[cat_cols]).round(2)
x_val[cat_cols] = encoder.transform(x_val[cat_cols]).round(2)
x_test[cat_cols] = encoder.transform(x_test[cat_cols]).round(2)

# creating new features
def feat_engg(df1_encoded):
    df1_encoded['required_partners'] = df1_encoded['total_outstanding_orders']␣
 ↪- df1_encoded['total_onshift_partners'] # required partners to fullfil␣
 ↪outstanding orders
    df1_encoded['partner_available_ratio'] =␣
 ↪round(df1_encoded['total_onshift_partners'] / (␣
 ↪df1_encoded['total_onshift_partners'] + df1_encoded['total_busy_partners']),␣
 ↪2)
    df1_encoded['available_ratio_and_outstanding'] =␣
 ↪df1_encoded['partner_available_ratio'] *␣
 ↪df1_encoded['total_outstanding_orders']
    df1_encoded['avg_price'] = round((df1_encoded['subtotal']/
 ↪df1_encoded['total_items'])*100,2)
    df1_encoded['price_range'] = df1_encoded['max_item_price'] -␣
 ↪df1_encoded['min_item_price']
    df1_encoded['price_ratio'] = round(df1_encoded['max_item_price'] /␣
 ↪df1_encoded['min_item_price'],2)
    df1_encoded['items_ratio'] = round(df1_encoded['num_distinct_items'] /␣
 ↪df1_encoded['total_items'],2)
    df1_encoded['store_and_its_category'] =␣
 ↪df1_encoded['store_primary_category'] * df1_encoded['store'] # type of␣
 ↪restaurant x individual store ( each store would have different efficiency )

    return df1_encoded

df_list = [x_train, x_val, x_test]
df_list = [feat_engg(df_new) for df_new in df_list]
x_train_updated, x_val_updated, x_test_updated = df_list
```

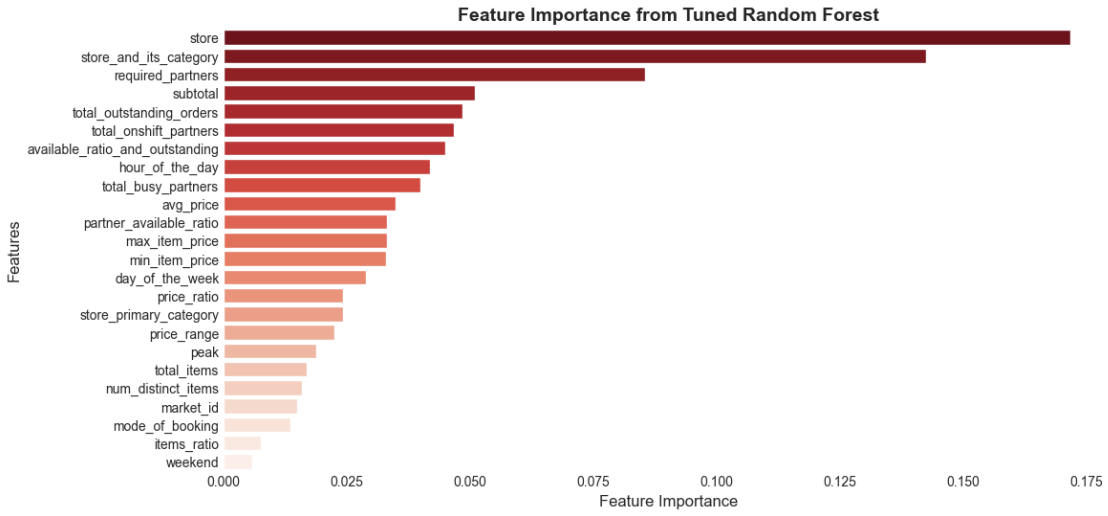
Random Forest

[82]:
```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor

# Define hyperparameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [50, 100, 200, 300],  # Number of trees
```

```
    'max_depth': [None, 10, 20, 30, 40],  # Depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples required at each leaf node
    'max_features': ['sqrt', 'log2'],  # Number of features considered for␣
 ↪splitting
    'bootstrap': [True, False]  # Whether to use bootstrapping
}

# Initialize Random Forest model
rf = RandomForestRegressor(random_state=42)

# Initialize Randomized Search CV
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=12,  # Number of random combinations to try
    scoring='neg_mean_squared_error',  # Scoring metric
    cv=3,  # 5-fold cross-validation
    verbose=2,  # Show search progress
    n_jobs=-1,  # Use all available cores
    random_state=42
)

# Perform hyperparameter tuning
random_search.fit(x_train_updated, y_train)

# Print best parameters
print("Best Hyperparameters:", random_search.best_params_)

# Train model with best parameters
best_rf = random_search.best_estimator_

# Make predictions
y_pred_train = best_rf.predict(x_train_updated)
y_pred_val = best_rf.predict(x_val_updated)

# Evaluate performance
mae = mean_absolute_error(y_val, y_pred_val)
print(f"Mean Absolute Error: {mae}")

mse = mean_squared_error(y_val, y_pred_val)
print(f"Mean Squared Error: {mse}")

r2 = r2_score(y_val, y_pred_val)
print(f"R2 score: {r2}")

# Feature Importance Visualization
```

```
feature_importances = pd.DataFrame({'Feature': x_train_updated.columns,␣
 ↪'Importance': best_rf.feature_importances_})
feature_importances = feature_importances.sort_values(by="Importance",␣
 ↪ascending=False)

# Plot feature importance
plt.figure(figsize=(12, 6))
sns.barplot(
    x=feature_importances["Importance"],
    y=feature_importances["Feature"],
    palette="Reds_r",
)

# Labels and Title
plt.xlabel("Feature Importance", fontsize=12)
plt.ylabel("Features", fontsize=12)
plt.title("Feature Importance from Tuned Random Forest", fontsize=14,␣
 ↪fontweight="bold")

# Remove grid for a cleaner look
plt.grid(False)

plt.show()
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best Hyperparameters: {'n_estimators': 300, 'min_samples_split': 2,
'min_samples_leaf': 4, 'max_features': 'log2', 'max_depth': None, 'bootstrap':
True}
Mean Absolute Error: 10.938656207246188
Mean Squared Error: 262.1552925455674
R2 score: 0.2748185987151762
```



Model Performance:

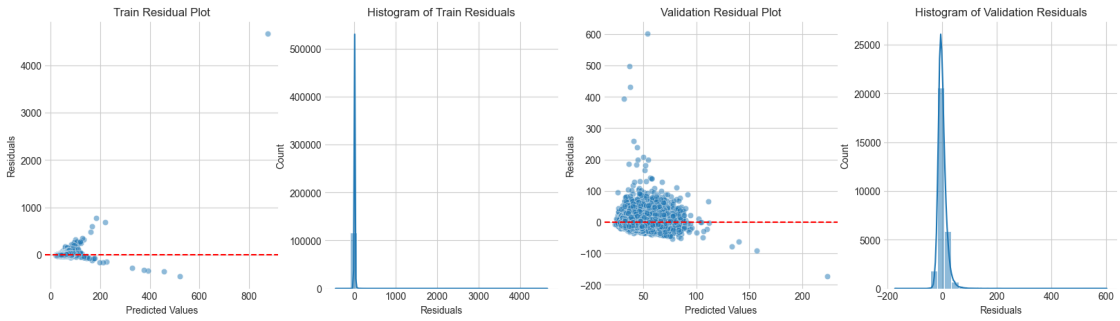
1. Random forest validation MAE - 11
2. Random forest validation MSE - 262
3. Random forest validation R2 - 0.27

Models are not performing well as they are unable to capture the variance in the data

Residual Analysis

[83]:
```
# train residuals

residuals_train = y_train - y_pred_train # training residuals

# Plot residuals
plt.figure(figsize=(20, 5))

plt.subplot(1,4,1)
sns.scatterplot(x=y_pred_train, y=residuals_train, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')  # Zero residual line
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Train Residual Plot")

plt.subplot(1,4,2)
sns.histplot(residuals_train, bins=30, kde=True)
plt.xlabel("Residuals")
plt.title("Histogram of Train Residuals")

residuals_val = y_val - y_pred_val # validation residuals

# Plot residuals
# plt.figure(figsize=(20, 5))

plt.subplot(1,4,3)
sns.scatterplot(x=y_pred_val, y=residuals_val, alpha=0.5)
plt.axhline(y=0, color='r', linestyle='--')  # Zero residual line
plt.xlabel("Predicted Values")
plt.ylabel("Residuals")
plt.title("Validation Residual Plot")

plt.subplot(1,4,4)
sns.histplot(residuals_val, bins=30, kde=True)
plt.xlabel("Residuals")
plt.title("Histogram of Validation Residuals")
```

```
plt.show()
```



Insights:

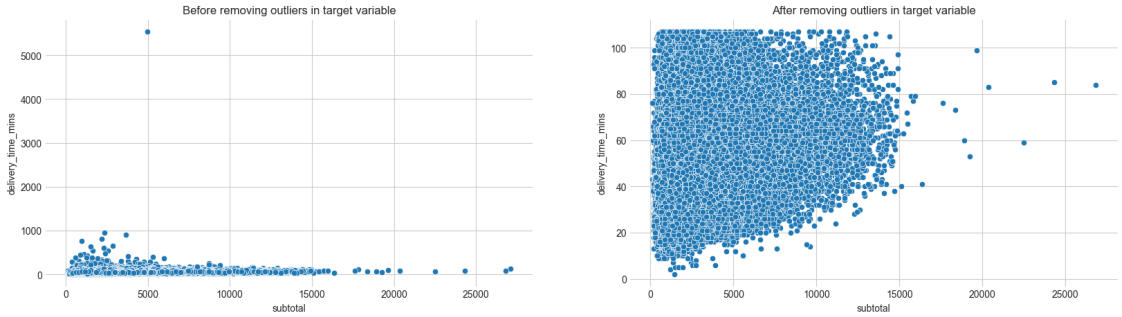
1. Residual Plot:
   - Residuals are mostly clustered around 0 but show a funnel shape (increasing variance for higher predicted values)
   - This suggests heteroscedasticity, meaning the model struggles to predict larger values accurately
2. Histogram Plot:
   - The residuals are highly skewed with a long right tail
   - Indicates the model underestimates certain data points, leading to large positive residuals

Outliers in target variable

[84]:
```
# outliers in target variable

plt.figure(figsize = (20,5))

plt.subplot(1,2,1)
sns.scatterplot(x = df1['subtotal'], y = df1['delivery_time_mins'])
plt.title('Before removing outliers in target variable')

# outliers
perc_99 = np.percentile(df1['delivery_time_mins'], 99)
above_99_perc = df1[df1['delivery_time_mins'] > perc_99].shape[0]/df1.shape[0]␣
 ↪* 100
print(f'Percentage of data above 99th percentile: {round(above_99_perc)}%')

# data after dropping outliers
df1_updated = df1[df1['delivery_time_mins'] <= perc_99] # new dataframe after␣
 ↪removing outliers
plt.subplot(1,2,2)
sns.scatterplot(x = df1_updated['subtotal'], y =␣
 ↪df1_updated['delivery_time_mins'])
plt.title('After removing outliers in target variable')
```

```
plt.show()
```

```
Percentage of data above 99th percentile: 1%
```



Before Outlier Removal (Left Plot):

- There are extreme outliers in delivery time, with some values above 5000 minutes.
- Most data points are clustered around lower delivery times, but the presence of long tail outliers skews the data.
- Such extreme values likely distort model predictions and impact performance.

After Outlier Removal (Right Plot):

- The majority of data falls within a reasonable range (0–100 minutes).
- Removing top 1% extreme values has cleaned the distribution.
- The density of points suggests a more structured pattern, improving model learnability.

[85]:
```
# train - test split
from sklearn.model_selection import train_test_split

x = df1_updated.drop(columns = ['delivery_time_mins'])
y = df1_updated['delivery_time_mins']

x_tr,x_test, y_tr, y_test = train_test_split(x,y,test_size = 0.2, random_state␣
 ↪= 42)
x_train,x_val, y_train, y_val = train_test_split(x_tr,y_tr,test_size = 0.2,␣
 ↪random_state = 42)

import category_encoders as ce

cat_cols = ["market_id", "store_primary_category",␣
 ↪"mode_of_booking","hour_of_the_day","day_of_the_week", "store", "peak",␣
 ↪"weekend"]

encoder = ce.TargetEncoder(cols= cat_cols) # target encoding
```

```
# fit the encoder
encoder.fit(x_train[cat_cols], y_train)

# apply transformation
x_train[cat_cols] = encoder.transform(x_train[cat_cols]).round(2)
x_val[cat_cols] = encoder.transform(x_val[cat_cols]).round(2)
x_test[cat_cols] = encoder.transform(x_test[cat_cols]).round(2)

df_list = [x_train, x_val, x_test]
df_list = [feat_engg(df_new) for df_new in df_list]
x_train_updated, x_val_updated, x_test_updated = df_list

# scaling the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

x_train_scaled = scaler.fit_transform(x_train_updated)
x_val_scaled = scaler.transform(x_val_updated)
x_test_scaled = scaler.transform(x_test_updated)
```

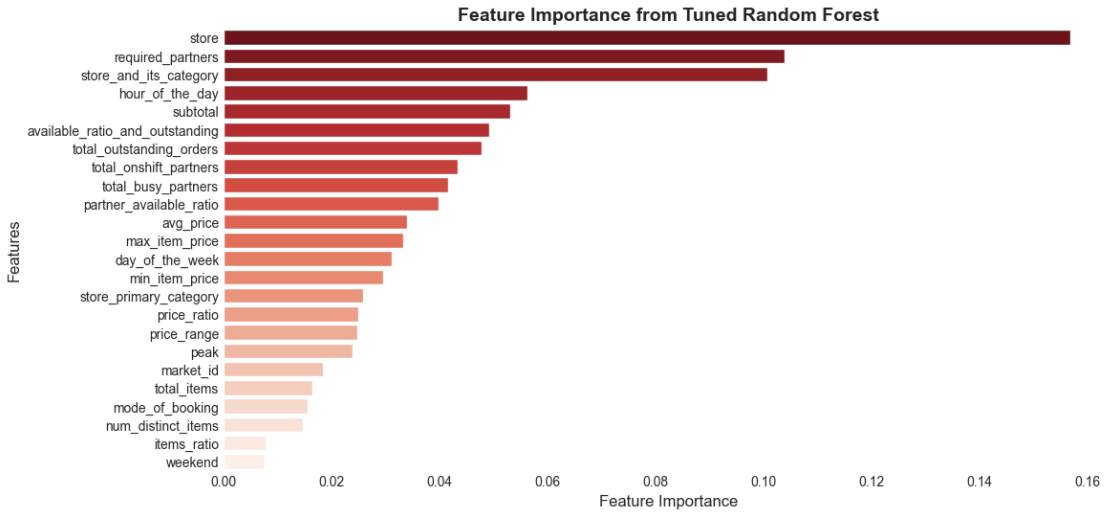
Random Forest

[88]:
```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Define hyperparameter grid for RandomizedSearchCV
param_dist = {
    'n_estimators': [50, 100, 200, 300],  # Number of trees
    'max_depth': [None, 10, 20, 30, 40],  # Depth of trees
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],  # Minimum samples required at each leaf node
    'max_features': ['sqrt', 'log2'],  # Number of features considered for␣
 ↪splitting
    'bootstrap': [True, False]  # Whether to use bootstrapping
}

# Initialize Random Forest model
rf = RandomForestRegressor(random_state=42)

# Initialize Randomized Search CV
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
```

```
    n_iter=12,  # Number of random combinations to try
    scoring='neg_mean_squared_error',  # Scoring metric
    cv=3,  # 5-fold cross-validation
    verbose=2,  # Show search progress
    n_jobs=-1,  # Use all available cores
    random_state=42
)

# Perform hyperparameter tuning
random_search.fit(x_train_updated, y_train)

# Print best parameters
print("Best Hyperparameters:", random_search.best_params_)

# Train model with best parameters
best_rf = random_search.best_estimator_

# Make predictions
y_pred_train = best_rf.predict(x_train_updated)
y_pred_val = best_rf.predict(x_val_updated)

# Evaluate performance
mae = mean_absolute_error(y_val, y_pred_val)
print(f"Mean Absolute Error: {mae}")

mse = mean_squared_error(y_val, y_pred_val)
print(f"Mean Squared Error: {mse}")

r2 = r2_score(y_val, y_pred_val)
print(f"R2 score: {r2}")

# Feature Importance Visualization
feature_importances = pd.DataFrame({'Feature': x_train_updated.columns,␣
 ↪'Importance': best_rf.feature_importances_})
feature_importances = feature_importances.sort_values(by="Importance",␣
 ↪ascending=False)

# Plot feature importance
plt.figure(figsize=(12, 6))
sns.barplot(
    x=feature_importances["Importance"],
    y=feature_importances["Feature"],
    palette="Reds_r",
)

# Labels and Title
plt.xlabel("Feature Importance", fontsize=12)
```

```
plt.ylabel("Features", fontsize=12)
plt.title("Feature Importance from Tuned Random Forest", fontsize=14,␣
 ↪fontweight="bold")

# Remove grid for a cleaner look
plt.grid(False)

plt.show()
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best Hyperparameters: {'n_estimators': 200, 'min_samples_split': 5,
'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': None, 'bootstrap':
False}
Mean Absolute Error: 10.20144547208886
Mean Squared Error: 176.6603730902782
R2 score: 0.32327755037751604
```



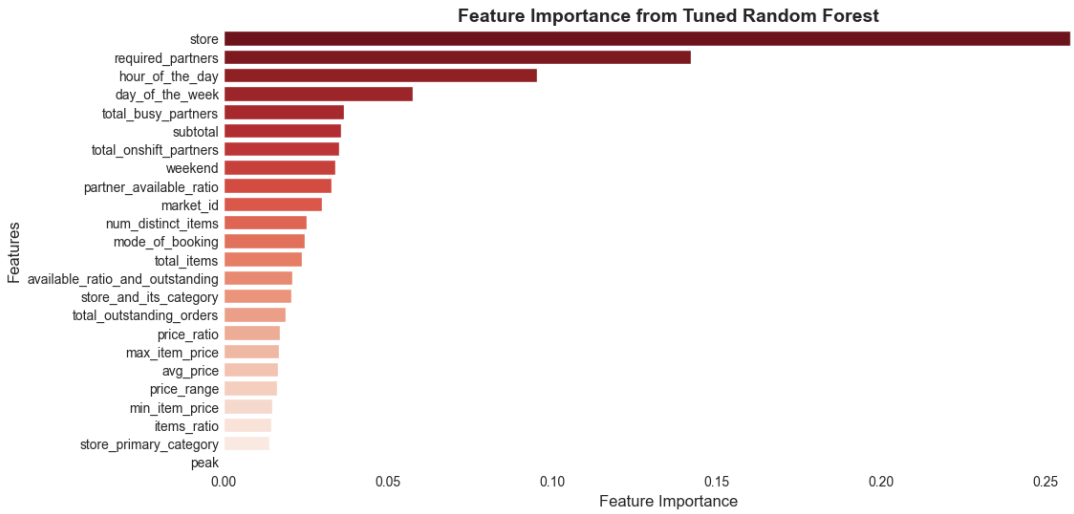
XGBoost

[89]:
```
# Randomized Search with Cross Validation

import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV

from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

param_grid = {
    'n_estimators': [250,300,350,400, 500, 600],
```

```
    'learning_rate': [0.001,0.01,0.1,0.2,0.5],
    'max_depth': [5,6,7,8,9,10],
    'subsample': [0.7, 0.8, 0.9, 1],
    'colsample_bytree': [0.6, 0.8, 1],
    'gamma': [0, 0.1, 0.2, 0.3, 0.5],
    'reg_alpha': [0, 0.01, 0.1, 1, 2, 3],
    'reg_lambda': [1, 2, 3, 4]}

xgb_reg = xgb.XGBRegressor(objective='reg:squarederror', random_state = 42)

random_search = RandomizedSearchCV(
    estimator = xgb_reg,
    param_distributions = param_grid,
    n_iter = 12,
    scoring='neg_mean_squared_error',
    cv=3,  # 5-fold cross-validation
    verbose=2,
    n_jobs=-1,
    random_state=42)

# fit the model
random_search.fit(x_train_updated, y_train)

# best model
print('Best parameters:', random_search.best_params_)

best_xgb = random_search.best_estimator_

y_pred_train = best_xgb.predict(x_train_updated)
y_pred_val = best_xgb.predict(x_val_updated)

from sklearn.metrics import mean_absolute_error, mean_squared_error

# Evaluate performance
mae = mean_absolute_error(y_val, y_pred_val)
print(f"Mean Absolute Error: {mae}")

mse = mean_squared_error(y_val, y_pred_val)
print(f"Mean Squared Error: {mse}")

r2 = r2_score(y_val, y_pred_val)
print(f"R2 score: {r2}")

importance = best_xgb.feature_importances_

df_importance = pd.DataFrame({'Feature':x_train_updated.columns, 'Importance':
 ↪importance})
```

```
df_importance = df_importance.sort_values(by = 'Importance', ascending = False)

# Plot feature importance
plt.figure(figsize=(12, 6))
sns.barplot(
    x=df_importance["Importance"],
    y=df_importance["Feature"],
    palette="Reds_r",
)

# Labels and Title
plt.xlabel("Feature Importance", fontsize=12)
plt.ylabel("Features", fontsize=12)
plt.title("Feature Importance from Tuned Random Forest", fontsize=14,␣
 ↪fontweight="bold")

# Remove grid for a cleaner look
plt.grid(False)

plt.show()
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best parameters: {'subsample': 0.9, 'reg_lambda': 1, 'reg_alpha': 1,
'n_estimators': 400, 'max_depth': 9, 'learning_rate': 0.01, 'gamma': 0.2,
'colsample_bytree': 1}
Mean Absolute Error: 10.140121988677132
Mean Squared Error: 174.84690759269319
R2 score: 0.33022428547358895
```



Neural Network

[57]:
```
from hyperopt import hp
from hyperopt import fmin, tpe, Trials
import tensorflow as tf
from tensorflow import keras
import numpy as np

space = {
    'learning_rate': hp.loguniform('learning_rate', -5, -1),
    'batch_size': hp.choice('batch_size', [8, 16, 32]),
    'num_layers': hp.choice('num_layers', [2,3]),
    'num_neurons': hp.quniform('num_neurons', 32, 256, 16),
    'dropout': hp.uniform('dropout', 0.1, 0.4),
    'activation': hp.choice('activation', ['relu','tanh'])
}

def objective(params):

    model = keras.Sequential()
    model.add(keras.layers.InputLayer(input_shape=(x_train_scaled.shape[1],)))
    model.add(keras.layers.Dense(24, activation = 'relu'))

    for _ in range(int(params['num_layers'])):
        model.add(keras.layers.Dense(int(params['num_neurons']), activation =␣
 ↪params['activation']))
        model.add(keras.layers.Dropout(params['dropout']))

    model.add(keras.layers.Dense(1))

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=params['learning_rate']),
        loss='mse',
        metrics=['mae'])

    # Train the model
    history = model.fit(x_train_scaled, y_train, validation_data=(x_val_scaled,␣
 ↪y_val),
                        epochs=5, batch_size=int(params['batch_size']),␣
 ↪verbose=0)

    # Get the validation loss
    val_loss = min(history.history['val_loss'])
    return val_loss

trials = Trials()

hyperparams = fmin(
    fn=objective,          # Objective function
```

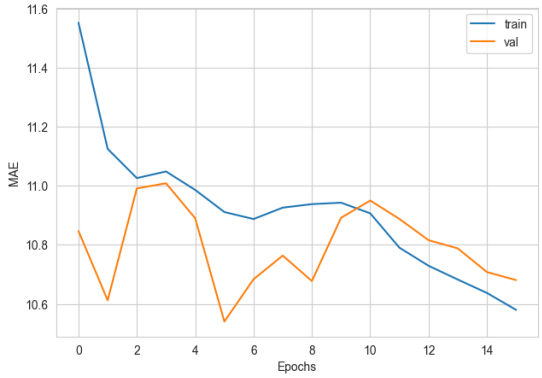
```
    space=space,            # Search space
    algo=tpe.suggest,       # Optimization algorithm
    max_evals=10,            # Number of evaluations
    trials=trials           # Store results
)

# Extract the best hyperparameters
best_hyperparams = {
    'learning_rate': hyperparams['learning_rate'],
    'batch_size': [16, 32, 64, 128][hyperparams['batch_size']],  # Because we␣
 ↪used hp.choice
    'num_layers': [2, 3, 4][hyperparams['num_layers']],
    'num_neurons': hyperparams['num_neurons'],
    'dropout': hyperparams['dropout'],
    'activation': ['relu', 'tanh'][hyperparams['activation']],
}

print("Best Hyperparameters:", best_hyperparams)
```

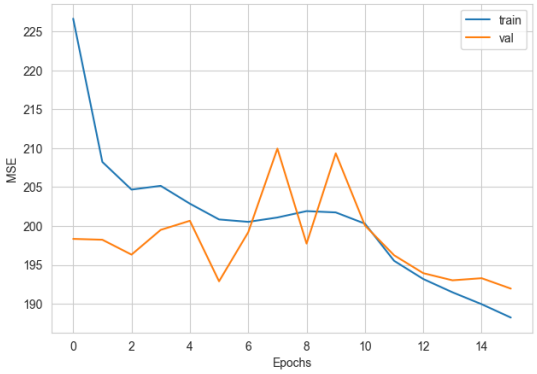
```
100%|                          | 10/10 [15:34<00:00,
93.49s/trial, best loss: 216.59530639648438]
Best Hyperparameters: {'activation': np.int64(0), 'batch_size': np.int64(0),
'dropout': np.float64(0.29437501356516926), 'learning_rate':
np.float64(0.008106509642631026), 'num_layers': np.int64(1), 'num_neurons':
np.float64(240.0)}
```

[70]:
```
# Build the final model with best hyperparameters
final_model = keras.Sequential()
final_model.add(keras.layers.InputLayer(input_shape=(x_train_scaled.shape[1],)))
final_model.add(keras.layers.Dense(24, activation = 'relu'))

for _ in range(int(best_hyperparams['num_layers'])):
    final_model.add(keras.layers.Dense(int(best_hyperparams['num_neurons']),␣
 ↪activation=best_hyperparams['activation']))
    final_model.add(keras.layers.Dropout(best_hyperparams['dropout']))

final_model.add(keras.layers.Dense(1))

# Compile with best learning rate
final_model.compile(
    optimizer=keras.optimizers.
 ↪Adam(learning_rate=best_hyperparams['learning_rate']),
    loss='mse',
    metrics=['mae'] )

# callbacks
```

```
reduce_lr = keras.callbacks.ReduceLROnPlateau(monitor = 'val_mae', factor = 0.
 ↪1, patience = 5)
model_checkpoint = keras.callbacks.ModelCheckpoint('best_final_model.keras',␣
 ↪monitor = 'val_mae', save_best_only = True)
early_stopping = keras.callbacks.EarlyStopping(monitor = 'val_mae', patience =␣
 ↪10, restore_best_weights = True )

# Train the final model on full training set
history = final_model.fit(x_train_scaled, y_train,
                epochs=20,  # More epochs if needed
                batch_size=int(best_hyperparams['batch_size']),
                validation_data=(x_val_scaled, y_val),
                callbacks = [reduce_lr,model_checkpoint,early_stopping],
                verbose=0)

# Load the full model (architecture + weights)
final_model_NN = keras.models.load_model("best_final_model.keras")
y_pred_val = final_model_NN.predict(x_val_scaled)

# Evaluate performance
mae = mean_absolute_error(y_val, y_pred_val)
print(f"Mean Absolute Error: {mae}")

mse = mean_squared_error(y_val, y_pred_val)
print(f"Mean Squared Error: {mse}")

r2 = r2_score(y_val, y_pred_val)
print(f"R2 score: {r2}")

# Visualization
plt.figure(figsize = (16,5))

plt.subplot(1,2,1)
plt.plot(history.epoch, history.history['mae'], label = 'train' )
plt.plot(history.epoch, history.history['val_mae'], label = 'val' )
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.epoch, history.history['loss'], label = 'train' )
plt.plot(history.epoch, history.history['val_loss'], label = 'val' )
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.show()
```

```
902/902              1s 1ms/step
Mean Absolute Error: 10.540231794608799
Mean Squared Error: 192.8835063913915
R2 score: 0.26962850007263095
```





Performance on Test Data

[75]:
```python
# random forest
y_pred_test = best_rf.predict(x_test_scaled)

# Evaluate performance
mae = mean_absolute_error(y_test, y_pred_test)
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
print('Random Forest')
print(f"Mean Absolute Error: {mae}, Mean Squared Error: {mse}, R2 score: {r2}")

# xgboost
y_pred_test = best_xgb.predict(x_test_scaled)

# Evaluate performance
mae = mean_absolute_error(y_test, y_pred_test)
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
print('XGBoost')
print(f"Mean Absolute Error: {mae}, Mean Squared Error: {mse}, R2 score: {r2}")

# neural network
y_pred_test = final_model_NN.predict(x_test_scaled)

# Evaluate performance
mae = mean_absolute_error(y_test, y_pred_test)
mse = mean_squared_error(y_test, y_pred_test)
r2 = r2_score(y_test, y_pred_test)
```

```
print('Neural Network')
print(f"Mean Absolute Error: {mae}, Mean Squared Error: {mse}, R2 score: {r2}")
```

```
Random Forest
Mean Absolute Error: 14.06005010218658, Mean Squared Error: 363.3820655767734,
R2 score: -0.3830131935034462
XGBoost
Mean Absolute Error: 14.01133434018776, Mean Squared Error: 361.46775262331334,
R2 score: -0.37572741822191835
1127/1127             1s 1ms/step
Neural Network
Mean Absolute Error: 10.521328381273314, Mean Squared Error: 192.88338379345257,
R2 score: 0.26589589893347654
```

Random Forest & XGBoost

- Both models perform similarly, with MAE ~14, MSE ~360, and negative R^2 scores.
- The negative R^2 score suggests that these models perform worse than a simple mean-based prediction (they fail to capture the variance in the target variable).
- Since both models are tree-based, it indicates they might not be well-suited for this dataset.

Neural Network (NN)

- Best performing model among the three: Lowest MAE (10.52) Lowest MSE (192.88) Only positive R^2 score (0.27) → This indicates the model explains ~27% of the variance, which is still not great but significantly better than Random Forest & XGBoost.

Possible Reasons for Poor Performance

- Tree-based models struggle with the dataset (maybe due to feature interactions or high noise).
- Neural Network shows promise, but an R^2 of 0.27 suggests room for improvement.
- The dataset might have irrelevant features affecting performance

4 Approach

The goal of this project is to predict delivery time for an online delivery platform. Accurate predictions help ***optimize fleet management, reduce delays, and improve customer satisfaction***

Data Exploration & Preprocessing

1. During initial EDA, Identified missing values key features like on-shift partners, busy partners, and outstanding orders. Used hour-based imputation by calculating ***mean & standard deviation*** per hour for each feature to fill gaps realistically
2. Found extreme values in delivery time (>99th percentile) and removed them to avoid model distortion. Used ***boxplots & IQR analysis*** to detect and handle inflated values in order-related features

Modeling & Optimization

1. Implemented ***Random Forest, XGBoost, and Neural Networks***, fine-tuning hyperparameters using ***RandomizedSearchCV and Hyperopt***

2. Created new engineered features, such as partner availability ratio and time based features, to enhance model performance
3. Addressed heteroscedasticity by treating outliers in target variable

Evaluation & Business Application

1. The best model achieved an R^2 score of ***~0.27 using Neural Network***, showing moderate predictive power
2. Analyzed peak-hour staffing requirements and suggested an ***optimal delivery partner-to-outstanding order ratio*** for faster deliveries

Real-World Impact: This model can help companies:

1. Optimize fleet management by dynamically adjusting the number of delivery partners
2. Reduce customer churn by providing more reliable delivery estimates
3. Identify bottlenecks and improve overall delivery efficiency