Food Deleivery

Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Load Data

In [2]:
```
data = pd.read_csv("train.csv")
```

In [3]:
```
data
```

Out[3]:

	ID	Delivery_person_ID	Delivery_person_Age	Delivery_person_Ratings	Restaurant_latitude	Restaurant_longitude	Deli
0	0x4607	INDORES13DEL02	37	4.9	22.745049	75.892471	
1	0xb379	BANGRES18DEL02	34	4.5	12.913041	77.683237	
2	0x5d6d	BANGRES19DEL01	23	4.4	12.914264	77.678400	
3	0x7a6a	COIMBRES13DEL02	38	4.7	11.003669	76.976494	
4	0x70a2	CHENRES12DEL01	32	4.6	12.972793	80.249982	
...	...	...	...	...	...	...	
45588	0x7c09	JAPRES04DEL01	30	4.8	26.902328	75.794257	
45589	0xd641	AGRRES16DEL01	21	4.6	0.000000	0.000000	
45590	0x4f8d	CHENRES08DEL03	30	4.9	13.022394	80.242439	
45591	0x5eee	COIMBRES11DEL01	20	4.7	11.001753	76.986241	
45592	0x5fb2	RANCHIRES09DEL02	23	4.9	23.351058	85.325731	

45593 rows × 20 columns

Head

In [4]:
```
data.head(15)
```

Out[4]:

	ID	Delivery_person_ID	Delivery_person_Age	Delivery_person_Ratings	Restaurant_latitude	Restaurant_longitude	Delivery
0	0x4607	INDORES13DEL02	37	4.9	22.745049	75.892471	
1	0xb379	BANGRES18DEL02	34	4.5	12.913041	77.683237	
2	0x5d6d	BANGRES19DEL01	23	4.4	12.914264	77.678400	
3	0x7a6a	COIMBRES13DEL02	38	4.7	11.003669	76.976494	
4	0x70a2	CHENRES12DEL01	32	4.6	12.972793	80.249982	
5	0x9bb4	HYDRES09DEL03	22	4.8	17.431668	78.408321	
6	0x95b4	RANCHIRES15DEL01	33	4.7	23.369746	85.339820	
7	0x9eb2	MYSRES15DEL02	35	4.6	12.352058	76.606650	
8	0x1102	HYDRES05DEL02	22	4.8	17.433809	78.386744	
9	0xcdcd	DEHRES17DEL01	36	4.2	30.327968	78.046106	
10	0xd987	KOCRES16DEL01	21	4.7	10.003064	76.307589	
11	0x2784	PUNERES13DEL03	23	4.7	18.562450	73.916619	
12	0xc8b6	LUDHRES15DEL02	34	4.3	30.899584	75.809346	
13	0xdb64	KNPRES14DEL02	24	4.7	26.463504	80.372929	
14	0x3af3	MUMRES15DEL03	29	4.5	19.176269	72.836721	

Information

In [5]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45593 entries, 0 to 45592
Data columns (total 20 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   ID                           45593 non-null  object 
 1   Delivery_person_ID           45593 non-null  object 
 2   Delivery_person_Age          45593 non-null  object 
 3   Delivery_person_Ratings      45593 non-null  object 
 4   Restaurant_latitude          45593 non-null  float64
 5   Restaurant_longitude         45593 non-null  float64
 6   Delivery_location_latitude   45593 non-null  float64
 7   Delivery_location_longitude  45593 non-null  float64
 8   Order_Date                   45593 non-null  object 
 9   Time_Orderd                  45593 non-null  object 
 10  Time_Order_picked            45593 non-null  object 
 11  Weatherconditions            45593 non-null  object 
 12  Road_traffic_density         45593 non-null  object 
 13  Vehicle_condition            45593 non-null  int64  
 14  Type_of_order                45593 non-null  object 
 15  Type_of_vehicle              45593 non-null  object 
 16  multiple_deliveries          45593 non-null  object 
 17  Festival                     45593 non-null  object 
 18  City                         45593 non-null  object 
 19  Time_taken(min)              45593 non-null  object 
dtypes: float64(4), int64(1), object(15)
memory usage: 7.0+ MB
```

Check Null Values

In [6]:

```
data.isnull().sum()
```

Out[6]:
```
ID                           0
Delivery_person_ID           0
Delivery_person_Age          0
Delivery_person_Ratings      0
Restaurant_latitude          0
Restaurant_longitude         0
Delivery_location_latitude   0
Delivery_location_longitude  0
Order_Date                   0
Time_Orderd                  0
Time_Order_picked            0
Weatherconditions            0
Road_traffic_density         0
Vehicle_condition            0
Type_of_order                0
Type_of_vehicle              0
multiple_deliveries          0
Festival                     0
City                         0
Time_taken(min)              0
dtype: int64
```

Check Unique and Fix NaN Values

In [7]:
```
data["Delivery_person_Age"].unique()
```

Out[7]:
```
array(['37', '34', '23', '38', '32', '22', '33', '35', '36', '21', '24',
       '29', '25', '31', '27', '26', '20', 'NaN ', '28', '39', '30', '15',
       '50'], dtype=object)
```

In [8]:
```
data['Delivery_person_Age'] = data['Delivery_person_Age'].str.strip()
data['Delivery_person_Age'] = data['Delivery_person_Age'].replace('NaN', np.nan)  # Convert 'NaN ' to actual Nal
data['Delivery_person_Age'] = pd.to_numeric(data['Delivery_person_Age'])
data['Delivery_person_Age'] = pd.to_numeric(data['Delivery_person_Age']).astype('Int64')
```

In [9]:
```
data["Delivery_person_Age"].unique()
```

Out[9]:
```
<IntegerArray>
[  37,   34,   23,   38,   32,   22,   33,   35,   36,   21,   24,   29,   25,
   31,   27,   26,   20, <NA>,   28,   39,   30,   15,   50]
Length: 23, dtype: Int64
```

In [10]:
```
data["Delivery_person_Ratings"].unique()
```

Out[10]:
```
array(['4.9', '4.5', '4.4', '4.7', '4.6', '4.8', '4.2', '4.3', '4', '4.1',
       '5', '3.5', 'NaN ', '3.8', '3.9', '3.7', '2.6', '2.5', '3.6',
       '3.1', '2.7', '1', '3.2', '3.3', '6', '3.4', '2.8', '2.9', '3'],
      dtype=object)
```

In [11]:
```
data["Delivery_person_Ratings"] = data["Delivery_person_Ratings"].str.strip()  # Remove spaces
data["Delivery_person_Ratings"] = data["Delivery_person_Ratings"].replace('NaN', np.nan)  # Convert 'NaN ' to p
data["Delivery_person_Ratings"] = pd.to_numeric(data["Delivery_person_Ratings"])
```

In [12]:
```
data["Delivery_person_Ratings"].unique()
```

Out[12]:
```
array([4.9, 4.5, 4.4, 4.7, 4.6, 4.8, 4.2, 4.3, 4. , 4.1, 5. , 3.5, nan,
       3.8, 3.9, 3.7, 2.6, 2.5, 3.6, 3.1, 2.7, 1. , 3.2, 3.3, 6. , 3.4,
       2.8, 2.9, 3. ])
```

In [13]:
```
data["Time_Orderd"].unique()
```

Out[13]:
```
array(['11:30:00', '19:45:00', '08:30:00', '18:00:00', '13:30:00',
       '21:20:00', '19:15:00', '17:25:00', '20:55:00', '21:55:00',
       '14:55:00', '17:30:00', '09:20:00', '19:50:00', '20:25:00',
       '20:30:00', '20:40:00', '21:15:00', '20:20:00', '22:30:00',
       '08:15:00', '19:30:00', '12:25:00', '18:35:00', '20:35:00',
       '23:20:00', '23:35:00', '22:35:00', '23:25:00', '13:35:00',
       '21:35:00', '18:55:00', '14:15:00', '11:00:00', '09:45:00',
       '08:40:00', '23:00:00', 'NaN ', '19:10:00', '10:55:00', '21:40:00',
       '19:00:00', '16:45:00', '15:10:00', '22:45:00', '22:10:00',
       '20:45:00', '22:50:00', '17:55:00', '09:25:00', '20:15:00',
       '22:25:00', '22:40:00', '23:50:00', '15:25:00', '10:20:00',
       '10:40:00', '15:55:00', '20:10:00', '12:10:00', '15:30:00',
       '10:35:00', '21:10:00', '20:50:00', '12:35:00', '21:00:00',
       '23:40:00', '18:15:00', '18:20:00', '11:45:00', '12:45:00',
       '23:30:00', '10:50:00', '21:25:00', '10:10:00', '17:50:00',
       '22:20:00', '12:40:00', '23:55:00', '10:25:00', '08:45:00',
       '23:45:00', '19:55:00', '22:15:00', '23:10:00', '09:15:00',
       '18:25:00', '18:45:00', '16:50:00', '00:00:00', '14:20:00',
       '10:15:00', '08:50:00', '09:00:00', '17:45:00', '16:35:00',
       '21:45:00', '19:40:00', '14:50:00', '18:10:00', '12:20:00',
       '12:50:00', '09:10:00', '12:30:00', '17:10:00', '17:20:00',
       '18:30:00', '13:10:00', '19:35:00', '09:50:00', '15:00:00',
       '20:00:00', '10:30:00', '09:40:00', '15:35:00', '16:55:00',
       '22:55:00', '16:00:00', '17:15:00', '21:30:00', '18:40:00',
       '11:10:00', '13:50:00', '10:00:00', '21:50:00', '11:50:00',
       '22:00:00', '08:25:00', '11:20:00', '11:55:00', '09:30:00',
       '08:20:00', '08:10:00', '11:40:00', '23:15:00', '19:20:00',
       '12:15:00', '11:35:00', '11:15:00', '17:35:00', '17:40:00',
       '14:40:00', '18:50:00', '11:25:00', '14:25:00', '12:00:00',
       '16:10:00', '19:25:00', '08:55:00', '13:40:00', '17:00:00',
       '09:35:00', '08:35:00', '16:15:00', '13:20:00', '15:50:00',
       '15:20:00', '16:20:00', '14:30:00', '15:45:00', '16:40:00',
       '13:00:00', '12:55:00', '10:45:00', '13:25:00', '09:55:00',
       '15:15:00', '13:15:00', '14:00:00', '15:40:00', '16:25:00',
       '14:10:00', '13:45:00', '13:55:00', '14:35:00', '16:30:00',
       '14:45:00'], dtype=object)
```

In [14]:
```
# Strip spaces and handle NaN values
data["Time_Orderd"] = data["Time_Orderd"].str.strip()
data["Time_Orderd"].replace("NaN", np.nan, inplace=True)

# Convert to datetime format (only time part)
data["Time_Orderd"] = pd.to_datetime(data["Time_Orderd"], format="%H:%M:%S", errors="coerce").dt.time
```

In [15]:
```
data["Time_Orderd"].unique()
```

Out[15]:
```
array([datetime.time(11, 30), datetime.time(19, 45), datetime.time(8, 30),
       datetime.time(18, 0), datetime.time(13, 30), datetime.time(21, 20),
       datetime.time(19, 15), datetime.time(17, 25),
       datetime.time(20, 55), datetime.time(21, 55),
       datetime.time(14, 55), datetime.time(17, 30), datetime.time(9, 20),
       datetime.time(19, 50), datetime.time(20, 25),
       datetime.time(20, 30), datetime.time(20, 40),
       datetime.time(21, 15), datetime.time(20, 20),
       datetime.time(22, 30), datetime.time(8, 15), datetime.time(19, 30),
       datetime.time(12, 25), datetime.time(18, 35),
       datetime.time(20, 35), datetime.time(23, 20),
       datetime.time(23, 35), datetime.time(22, 35),
       datetime.time(23, 25), datetime.time(13, 35),
       datetime.time(21, 35), datetime.time(18, 55),
       datetime.time(14, 15), datetime.time(11, 0), datetime.time(9, 45),
       datetime.time(8, 40), datetime.time(23, 0), NaT,
       datetime.time(19, 10), datetime.time(10, 55),
       datetime.time(21, 40), datetime.time(19, 0), datetime.time(16, 45),
       datetime.time(15, 10), datetime.time(22, 45),
       datetime.time(22, 10), datetime.time(20, 45),
       datetime.time(22, 50), datetime.time(17, 55), datetime.time(9, 25),
       datetime.time(20, 15), datetime.time(22, 25),
       datetime.time(22, 40), datetime.time(23, 50),
       datetime.time(15, 25), datetime.time(10, 20),
       datetime.time(10, 40), datetime.time(15, 55),
       datetime.time(20, 10), datetime.time(12, 10),
       datetime.time(15, 30), datetime.time(10, 35),
       datetime.time(21, 10), datetime.time(20, 50),
       datetime.time(12, 35), datetime.time(21, 0), datetime.time(23, 40),
       datetime.time(18, 15), datetime.time(18, 20),
       datetime.time(11, 45), datetime.time(12, 45),
       datetime.time(23, 30), datetime.time(10, 50),
       datetime.time(21, 25), datetime.time(10, 10),
       datetime.time(17, 50), datetime.time(22, 20),
       datetime.time(12, 40), datetime.time(23, 55),
       datetime.time(10, 25), datetime.time(8, 45), datetime.time(23, 45),
       datetime.time(19, 55), datetime.time(22, 15),
       datetime.time(23, 10), datetime.time(9, 15), datetime.time(18, 25),
       datetime.time(18, 45), datetime.time(16, 50), datetime.time(0, 0),
       datetime.time(14, 20), datetime.time(10, 15), datetime.time(8, 50),
       datetime.time(9, 0), datetime.time(17, 45), datetime.time(16, 35),
       datetime.time(21, 45), datetime.time(19, 40),
       datetime.time(14, 50), datetime.time(18, 10),
       datetime.time(12, 20), datetime.time(12, 50), datetime.time(9, 10),
       datetime.time(12, 30), datetime.time(17, 10),
       datetime.time(17, 20), datetime.time(18, 30),
       datetime.time(13, 10), datetime.time(19, 35), datetime.time(9, 50),
       datetime.time(15, 0), datetime.time(20, 0), datetime.time(10, 30),
       datetime.time(9, 40), datetime.time(15, 35), datetime.time(16, 55),
       datetime.time(22, 55), datetime.time(16, 0), datetime.time(17, 15),
       datetime.time(21, 30), datetime.time(18, 40),
       datetime.time(11, 10), datetime.time(13, 50), datetime.time(10, 0),
       datetime.time(21, 50), datetime.time(11, 50), datetime.time(22, 0),
       datetime.time(8, 25), datetime.time(11, 20), datetime.time(11, 55),
       datetime.time(9, 30), datetime.time(8, 20), datetime.time(8, 10),
       datetime.time(11, 40), datetime.time(23, 15),
       datetime.time(19, 20), datetime.time(12, 15),
       datetime.time(11, 35), datetime.time(11, 15),
       datetime.time(17, 35), datetime.time(17, 40),
       datetime.time(14, 40), datetime.time(18, 50),
       datetime.time(11, 25), datetime.time(14, 25), datetime.time(12, 0),
       datetime.time(16, 10), datetime.time(19, 25), datetime.time(8, 55),
       datetime.time(13, 40), datetime.time(17, 0), datetime.time(9, 35),
       datetime.time(8, 35), datetime.time(16, 15), datetime.time(13, 20),
       datetime.time(15, 50), datetime.time(15, 20),
       datetime.time(16, 20), datetime.time(14, 30),
       datetime.time(15, 45), datetime.time(16, 40), datetime.time(13, 0),
       datetime.time(12, 55), datetime.time(10, 45),
       datetime.time(13, 25), datetime.time(9, 55), datetime.time(15, 15),
       datetime.time(13, 15), datetime.time(14, 0), datetime.time(15, 40),
       datetime.time(16, 25), datetime.time(14, 10),
       datetime.time(13, 45), datetime.time(13, 55),
       datetime.time(14, 35), datetime.time(16, 30),
       datetime.time(14, 45)], dtype=object)
```

In [16]:
```
data["Weatherconditions"].unique()
```

Out[16]:
```
array(['conditions Sunny', 'conditions Stormy', 'conditions Sandstorms',
       'conditions Cloudy', 'conditions Fog', 'conditions Windy',
       'conditions NaN'], dtype=object)
```

In [17]:
```
data["Weatherconditions"] = data["Weatherconditions"].str.replace("conditions ", "", regex=False)
data["Weatherconditions"].replace("NaN", pd.NA, inplace=True)
```

```
In [18]: data["Weatherconditions"].unique()
```

```
Out[18]: array(['Sunny', 'Stormy', 'Sandstorms', 'Cloudy', 'Fog', 'Windy', <NA>],
               dtype=object)
```

```
In [19]: data["Road_traffic_density"].unique()
```

```
Out[19]: array(['High ', 'Jam ', 'Low ', 'Medium ', 'NaN '], dtype=object)
```

```
In [20]: data['Road_traffic_density'] = data['Road_traffic_density'].str.strip()
         data["Road_traffic_density"].replace("NaN", pd.NA, inplace=True)
```

```
In [21]: data["Road_traffic_density"].unique()
```

```
Out[21]: array(['High', 'Jam', 'Low', 'Medium', <NA>], dtype=object)
```

```
In [22]: data["Type_of_order"].unique()
```

```
Out[22]: array(['Snack ', 'Drinks ', 'Buffet ', 'Meal '], dtype=object)
```

```
In [23]: data['Type_of_order'] = data['Type_of_order'].str.strip()
```

```
In [24]: data["Type_of_order"].unique()
```

```
Out[24]: array(['Snack', 'Drinks', 'Buffet', 'Meal'], dtype=object)
```

```
In [25]: data["Type_of_vehicle"].unique()
```

```
Out[25]: array(['motorcycle ', 'scooter ', 'electric_scooter ', 'bicycle '],
               dtype=object)
```

```
In [26]: data['Type_of_vehicle'] = data['Type_of_vehicle'].str.strip()
```

```
In [27]: data["Type_of_vehicle"].unique()
```

```
Out[27]: array(['motorcycle', 'scooter', 'electric_scooter', 'bicycle'],
               dtype=object)
```

```
In [28]: data["multiple_deliveries"].unique()
```

```
Out[28]: array(['0', '1', '3', 'NaN ', '2'], dtype=object)
```

```
In [29]: data["multiple_deliveries"] = data["multiple_deliveries"].str.strip()  # Remove spaces
         data["multiple_deliveries"] = data["multiple_deliveries"].replace('NaN', np.nan)  # Convert 'NaN ' to proper NaN
         data["multiple_deliveries"] = pd.to_numeric(data["multiple_deliveries"])
         data['multiple_deliveries'] = pd.to_numeric(data['multiple_deliveries']).astype('Int64')
```

```
In [30]: data["multiple_deliveries"].unique()
```

```
Out[30]: <IntegerArray>
         [0, 1, 3, <NA>, 2]
         Length: 5, dtype: Int64
```

```
In [31]: data["Festival"].unique()
```

```
Out[31]: array(['No ', 'Yes ', 'NaN '], dtype=object)
```

```
In [32]: data['Festival'] = data['Festival'].str.strip()
         data["Festival"].replace("NaN", pd.NA, inplace=True)
```

```
In [33]: data["Festival"].unique()
```

```
Out[33]: array(['No', 'Yes', <NA>], dtype=object)
```

```
In [34]: data["City"].unique()
```

```
Out[34]: array(['Urban ', 'Metropolitian ', 'Semi-Urban ', 'NaN '], dtype=object)
```

```
In [35]: data['City'] = data['City'].str.strip()
         data["City"].replace("NaN", pd.NA, inplace=True)
```

```
In [36]: data["City"].unique()
```

```
Out[36]: array(['Urban', 'Metropolitian', 'Semi-Urban', <NA>], dtype=object)
```

```
In [37]: data["Time_taken(min)"].unique()
```

Out[37]:
```
array(['(min) 24', '(min) 33', '(min) 26', '(min) 21', '(min) 30',
       '(min) 40', '(min) 32', '(min) 34', '(min) 46', '(min) 23',
       '(min) 20', '(min) 41', '(min) 15', '(min) 36', '(min) 39',
       '(min) 18', '(min) 38', '(min) 47', '(min) 12', '(min) 22',
       '(min) 25', '(min) 35', '(min) 10', '(min) 19', '(min) 11',
       '(min) 28', '(min) 52', '(min) 16', '(min) 27', '(min) 49',
       '(min) 17', '(min) 14', '(min) 37', '(min) 44', '(min) 42',
       '(min) 31', '(min) 13', '(min) 29', '(min) 50', '(min) 43',
       '(min) 48', '(min) 54', '(min) 53', '(min) 45', '(min) 51'],
      dtype=object)
```

In [38]:
```
data["Time_taken(min)"] = data["Time_taken(min)"].str.extract(r'(\d+)')  # Extract numbers only
data["Time_taken(min)"] = pd.to_numeric(data["Time_taken(min)"])
```

In [39]:
```
data["Time_taken(min)"].unique()
```

Out[39]:
```
array([24, 33, 26, 21, 30, 40, 32, 34, 46, 23, 20, 41, 15, 36, 39, 18, 38,
       47, 12, 22, 25, 35, 10, 19, 11, 28, 52, 16, 27, 49, 17, 14, 37, 44,
       42, 31, 13, 29, 50, 43, 48, 54, 53, 45, 51], dtype=int64)
```

In [40]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45593 entries, 0 to 45592
Data columns (total 20 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   ID                           45593 non-null  object 
 1   Delivery_person_ID           45593 non-null  object 
 2   Delivery_person_Age          43739 non-null  Int64  
 3   Delivery_person_Ratings      43685 non-null  float64
 4   Restaurant_latitude          45593 non-null  float64
 5   Restaurant_longitude         45593 non-null  float64
 6   Delivery_location_latitude   45593 non-null  float64
 7   Delivery_location_longitude  45593 non-null  float64
 8   Order_Date                   45593 non-null  object 
 9   Time_Orderd                  43862 non-null  object 
 10  Time_Order_picked            45593 non-null  object 
 11  Weatherconditions            44977 non-null  object 
 12  Road_traffic_density         44992 non-null  object 
 13  Vehicle_condition            45593 non-null  int64  
 14  Type_of_order                45593 non-null  object 
 15  Type_of_vehicle              45593 non-null  object 
 16  multiple_deliveries          44600 non-null  Int64  
 17  Festival                     45365 non-null  object 
 18  City                         44393 non-null  object 
 19  Time_taken(min)              45593 non-null  int64  
dtypes: Int64(2), float64(5), int64(2), object(11)
memory usage: 7.0+ MB
```

In [41]:
```
data.isnull().sum()/len(data)*100
```

Out[41]:
```
ID                             0.000000
Delivery_person_ID             0.000000
Delivery_person_Age            4.066414
Delivery_person_Ratings        4.184853
Restaurant_latitude            0.000000
Restaurant_longitude           0.000000
Delivery_location_latitude     0.000000
Delivery_location_longitude    0.000000
Order_Date                     0.000000
Time_Orderd                    3.796635
Time_Order_picked              0.000000
Weatherconditions              1.351085
Road_traffic_density           1.318185
Vehicle_condition              0.000000
Type_of_order                  0.000000
Type_of_vehicle                0.000000
multiple_deliveries            2.177966
Festival                       0.500077
City                           2.631983
Time_taken(min)                0.000000
dtype: float64
```

In [42]:
```
data["Delivery_person_Age"] = data["Delivery_person_Age"].fillna(int(data["Delivery_person_Age"].mean()))
```

In [43]:
```
data["Delivery_person_Ratings"] = data["Delivery_person_Ratings"].fillna(int(data["Delivery_person_Ratings"].mea
```

In [44]:
```
data["Time_Orderd"] = data["Time_Orderd"].fillna(data["Time_Orderd"].mode()[0])
```

In [45]:
```
data["Weatherconditions"] = data["Weatherconditions"].fillna("unknown")
```

In [46]:
```
data["Road_traffic_density"] = data["Road_traffic_density"].fillna("unknown")
```

In [47]:
```
data["multiple_deliveries"] = data["multiple_deliveries"].fillna(int(data["multiple_deliveries"].mean()))
```

In [48]:
```
data["Festival"] = data["Festival"].fillna("unknown")
```

In [49]:
```
data["City"] = data["City"].fillna("unknown")
```

In [50]:
```
data.isnull().sum()/len(data)*100
```

Out[50]:
```
ID                             0.0
Delivery_person_ID             0.0
Delivery_person_Age            0.0
Delivery_person_Ratings        0.0
Restaurant_latitude            0.0
Restaurant_longitude           0.0
Delivery_location_latitude     0.0
Delivery_location_longitude    0.0
Order_Date                     0.0
Time_Orderd                    0.0
Time_Order_picked              0.0
Weatherconditions              0.0
Road_traffic_density           0.0
Vehicle_condition              0.0
Type_of_order                  0.0
Type_of_vehicle                0.0
multiple_deliveries            0.0
Festival                       0.0
City                           0.0
Time_taken(min)                0.0
dtype: float64
```

Description

In [51]:
```
data.describe()
```

Out[51]:

	Delivery_person_Age	Delivery_person_Ratings	Restaurant_latitude	Restaurant_longitude	Delivery_location_latitude	Delivery
count	45593.0	45593.000000	45593.000000	45593.000000	45593.000000	
mean	29.544075	4.607258	17.017729	70.231332	17.465186	
std	5.696793	0.351359	8.185109	22.883647	7.335122	
min	15.0	1.000000	-30.905562	-88.366217	0.010000	
25%	25.0	4.500000	12.933284	73.170000	12.988453	
50%	29.0	4.700000	18.546947	75.898497	18.633934	
75%	34.0	4.800000	22.728163	78.044095	22.785049	
max	50.0	6.000000	30.914057	88.433452	31.054057	

Calculate Distance and Time

In [52]:
```
from geopy.distance import geodesic
import pandas as pd

# Function to calculate Haversine distance
def haversine_distance(row):
    return round(geodesic(
        (row["Restaurant_latitude"], row["Restaurant_longitude"]),
        (row["Delivery_location_latitude"], row["Delivery_location_longitude"])
    ).kilometers, 2)  # Correct rounding

# Apply function to calculate distance
data["Distance_km"] = data.apply(haversine_distance, axis=1)

# Convert time columns to datetime
data["Time_Orderd"] = pd.to_datetime(data["Time_Orderd"], format="%H:%M:%S", errors="coerce")
data["Time_Order_picked"] = pd.to_datetime(data["Time_Order_picked"], format="%H:%M:%S", errors="coerce")

# Fix negative time differences by adding one day if needed
data.loc[data["Time_Order_picked"] < data["Time_Orderd"], "Time_Order_picked"] += pd.Timedelta(days=1)

# Compute time difference in minutes
data["Time_Difference_min"] = (data["Time_Order_picked"] - data["Time_Orderd"]).dt.total_seconds() / 60
```

In [53]:
```
# Convert to datetime format
```

```python
data["Order_Date"] = pd.to_datetime(data["Order_Date"], format="%d-%m-%Y", errors="coerce")

# Extract day, month, and year
data["Day"] = data["Order_Date"].dt.day
data["Month"] = data["Order_Date"].dt.month
data["Year"] = data["Order_Date"].dt.year
```

Drop Columns

In [54]:
```python
data = data.drop(["Restaurant_latitude", "Restaurant_longitude","Delivery_location_latitude","Delivery_location_
```

In [55]:
```python
data = data.drop(["ID","Delivery_person_ID","Order_Date","Time_Orderd","Time_Order_picked","Year"],axis=1)
```

In [56]:
```python
data
```

Out[56]:

	Delivery_person_Age	Delivery_person_Ratings	Weatherconditions	Road_traffic_density	Vehicle_condition	Type_of_order	T
0	37	4.9	Sunny	High	2	Snack	
1	34	4.5	Stormy	Jam	2	Snack	
2	23	4.4	Sandstorms	Low	0	Drinks	
3	38	4.7	Sunny	Medium	0	Buffet	
4	32	4.6	Cloudy	High	1	Snack	
...	...	...	...	...	...	...	
45588	30	4.8	Windy	High	1	Meal	
45589	21	4.6	Windy	Jam	0	Buffet	
45590	30	4.9	Cloudy	Low	1	Drinks	
45591	20	4.7	Cloudy	High	0	Snack	
45592	23	4.9	Fog	Medium	2	Snack	

45593 rows × 15 columns

In [57]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45593 entries, 0 to 45592
Data columns (total 15 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Delivery_person_Age      45593 non-null  Int64  
 1   Delivery_person_Ratings  45593 non-null  float64
 2   Weatherconditions        45593 non-null  object 
 3   Road_traffic_density     45593 non-null  object 
 4   Vehicle_condition        45593 non-null  int64  
 5   Type_of_order            45593 non-null  object 
 6   Type_of_vehicle          45593 non-null  object 
 7   multiple_deliveries      45593 non-null  Int64  
 8   Festival                 45593 non-null  object 
 9   City                     45593 non-null  object 
 10  Time_taken(min)          45593 non-null  int64  
 11  Distance_km              45593 non-null  float64
 12  Time_Difference_min      45593 non-null  float64
 13  Day                      45593 non-null  int32  
 14  Month                    45593 non-null  int32  
dtypes: Int64(2), float64(3), int32(2), int64(2), object(6)
memory usage: 5.0+ MB
```

In [58]:
```python
data.describe()
```

Out[58]:

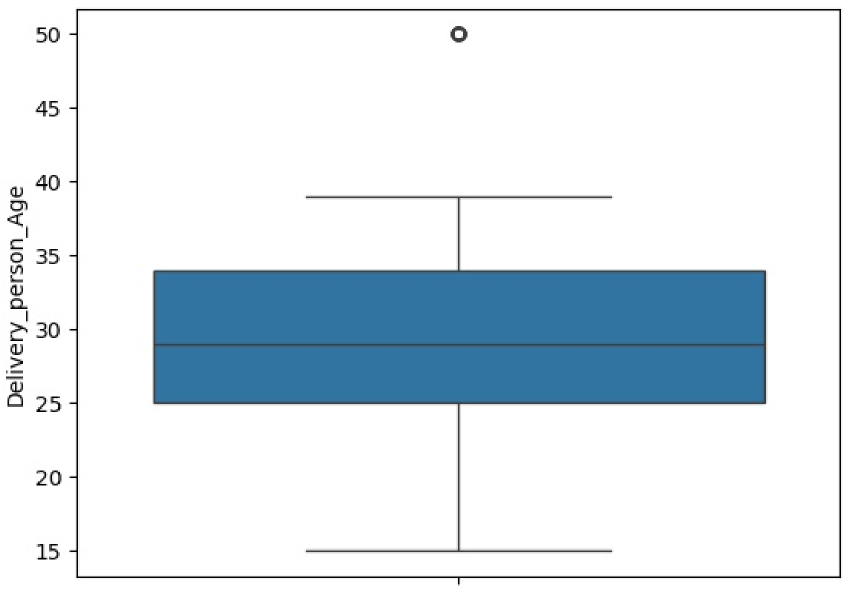
	Delivery_person_Age	Delivery_person_Ratings	Vehicle_condition	multiple_deliveries	Time_taken(min)	Distance_km	Time_I
count	45593.0	45593.000000	45593.000000	45593.0	45593.000000	45593.000000	
mean	29.544075	4.607258	1.023359	0.728445	26.294607	99.198964	
std	5.696793	0.351359	0.839065	0.576543	9.383806	1099.925177	
min	15.0	1.000000	0.000000	0.0	10.000000	1.460000	
25%	25.0	4.500000	0.000000	0.0	19.000000	4.650000	
50%	29.0	4.700000	1.000000	1.0	26.000000	9.250000	
75%	34.0	4.800000	2.000000	1.0	32.000000	13.740000	
max	50.0	6.000000	3.000000	3.0	54.000000	19709.580000	

In [59]:
```
data.isnull().sum()
```

Out[59]:
```
Delivery_person_Age        0
Delivery_person_Ratings    0
Weatherconditions          0
Road_traffic_density       0
Vehicle_condition          0
Type_of_order              0
Type_of_vehicle            0
multiple_deliveries        0
Festival                   0
City                       0
Time_taken(min)            0
Distance_km                0
Time_Difference_min        0
Day                        0
Month                      0
dtype: int64
```

Check and Handle Outliers

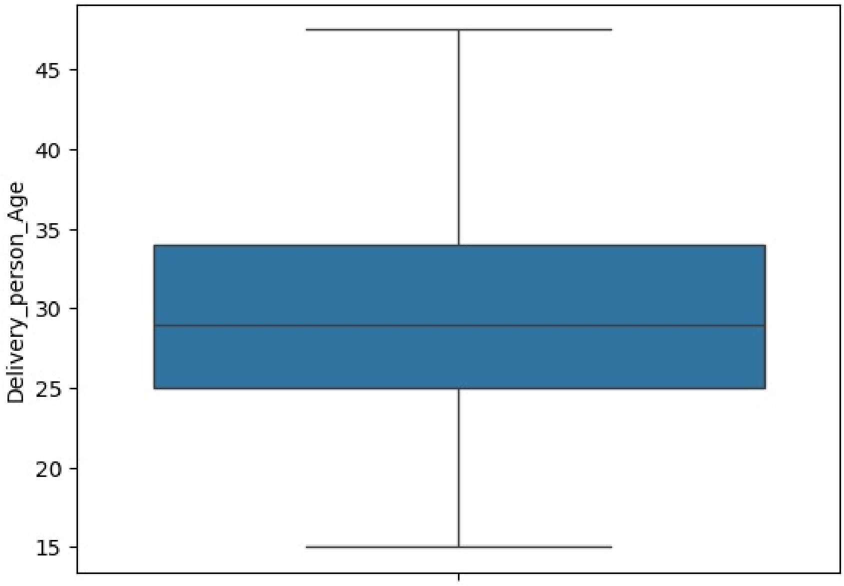
In [60]:
```
sns.boxplot(data["Delivery_person_Age"])
plt.show()
```



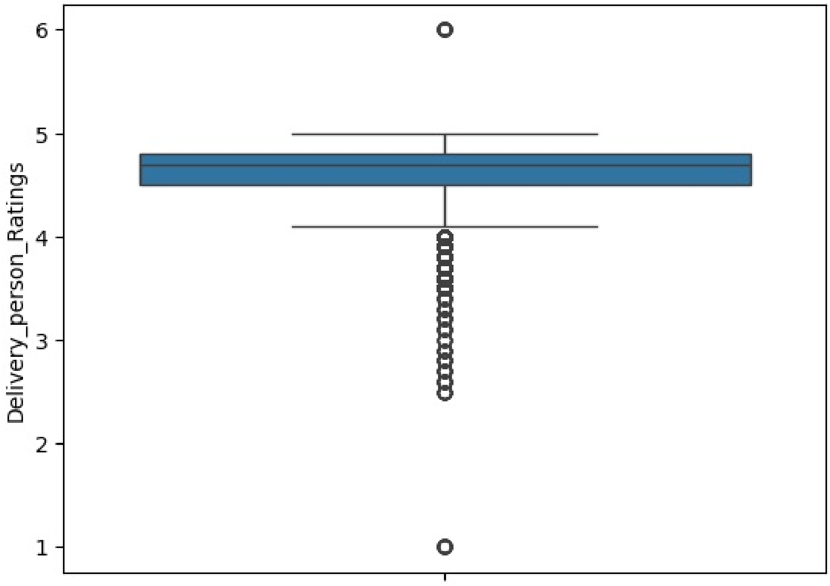
In [61]:
```
Q1 = data['Delivery_person_Age'].quantile(0.25)
Q3 = data['Delivery_person_Age'].quantile(0.75)
IQR = Q3 - Q1

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

outliers = data[(data['Delivery_person_Age'] < lower_bound) | (data['Delivery_person_Age'] > upper_bound)]
print(f"Number of outliers in amt: {len(outliers)}")

data['Delivery_person_Age'] = np.where(data['Delivery_person_Age'] < lower_bound, lower_bound, data['Delivery_pe
data['Delivery_person_Age'] = np.where(data['Delivery_person_Age'] > upper_bound, upper_bound, data['Delivery_pe
```

```
Number of outliers in amt: 53
```

In [62]:
```
sns.boxplot(data["Delivery_person_Age"])
plt.show()
```



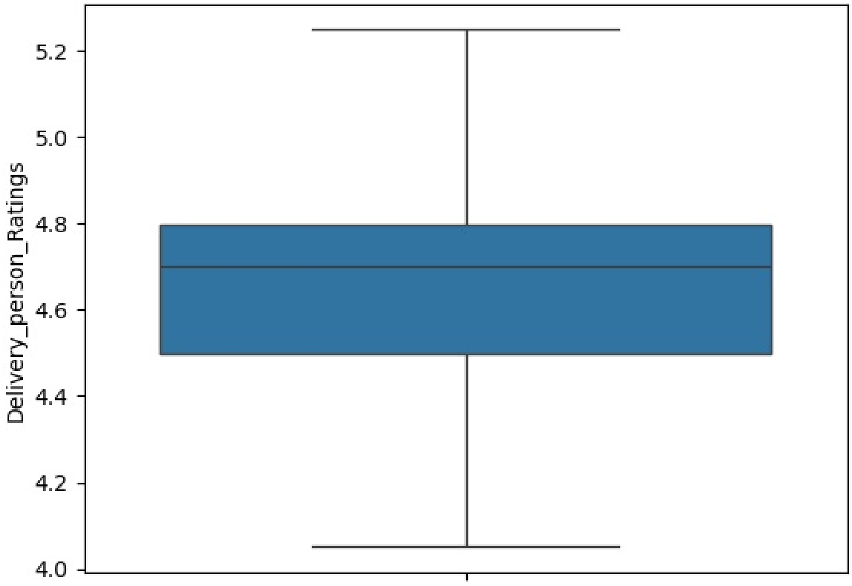
In [63]:
```
sns.boxplot(data["Delivery_person_Ratings"])
plt.show()
```



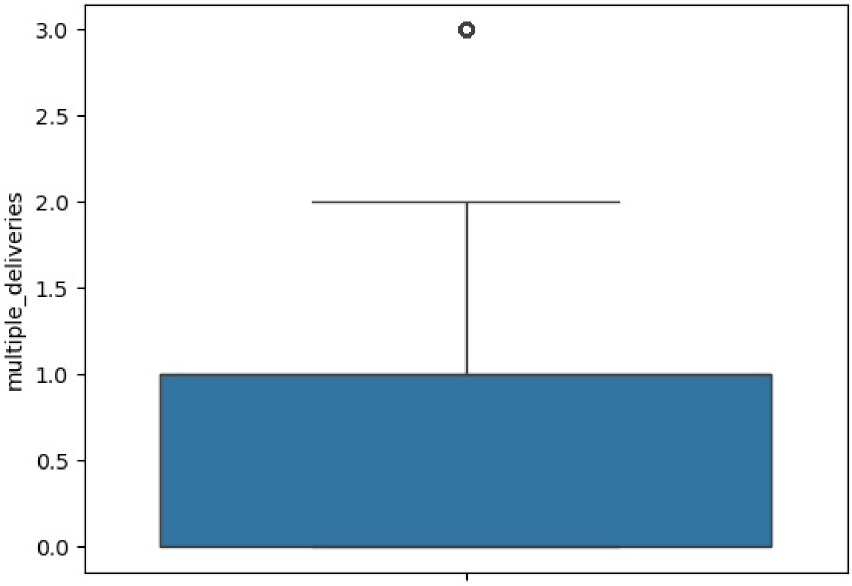
In [64]:
```
Q1 = data['Delivery_person_Ratings'].quantile(0.25)
Q3 = data['Delivery_person_Ratings'].quantile(0.75)
IQR = Q3 - Q1

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

outliers = data[(data['Delivery_person_Ratings'] < lower_bound) | (data['Delivery_person_Ratings'] > upper_bound
print(f"Number of outliers in amt: {len(outliers)}")

data['Delivery_person_Ratings'] = np.where(data['Delivery_person_Ratings'] < lower_bound, lower_bound, data['De
data['Delivery_person_Ratings'] = np.where(data['Delivery_person_Ratings'] > upper_bound, upper_bound, data['De
```

```
Number of outliers in amt: 4405
```

In [65]:
```
sns.boxplot(data["Delivery_person_Ratings"])
plt.show()
```



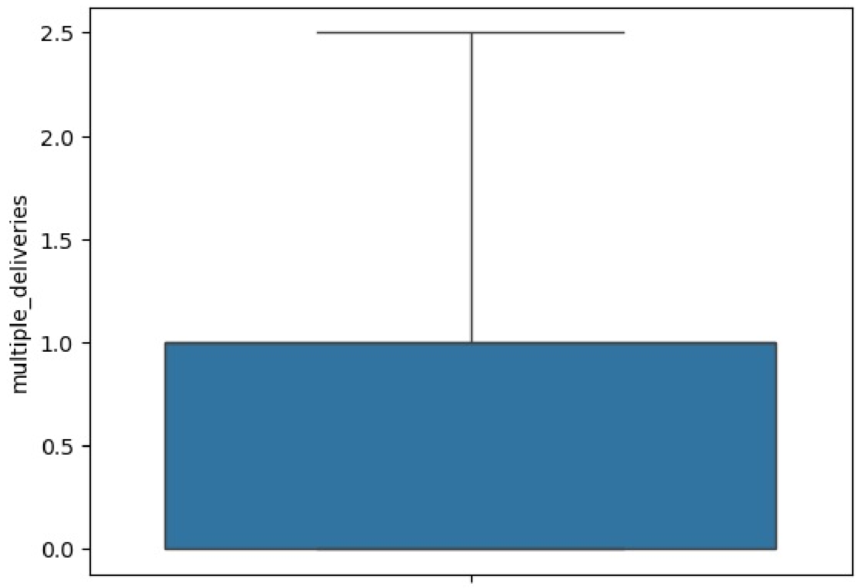

```
In [66]: sns.boxplot(data["multiple_deliveries"])
plt.show()
```



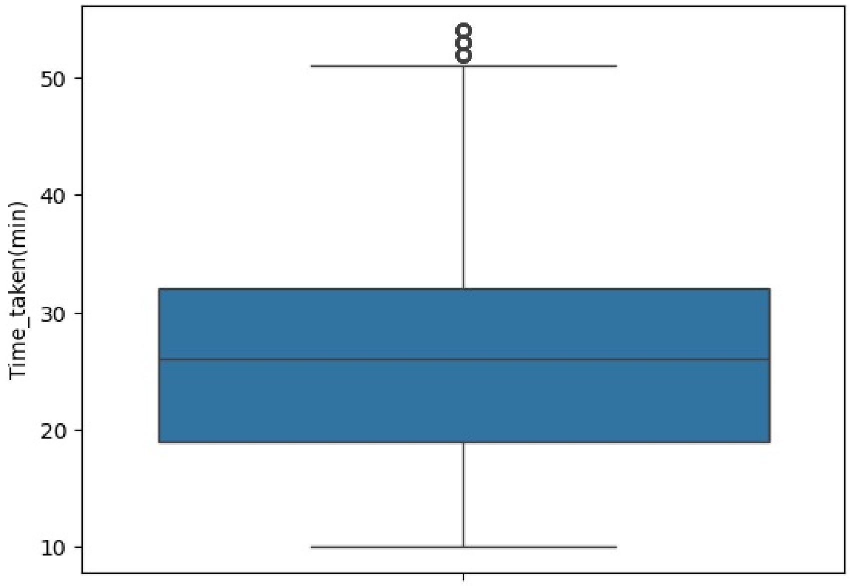

```
In [67]: Q1 = data['multiple_deliveries'].quantile(0.25)
Q3 = data['multiple_deliveries'].quantile(0.75)
IQR = Q3 - Q1

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

outliers = data[(data['multiple_deliveries'] < lower_bound) | (data['multiple_deliveries'] > upper_bound)]
print(f"Number of outliers in amt: {len(outliers)}")

data['multiple_deliveries'] = np.where(data['multiple_deliveries'] < lower_bound, lower_bound, data['multiple_de
data['multiple_deliveries'] = np.where(data['multiple_deliveries'] > upper_bound, upper_bound, data['multiple_de
```

```
Number of outliers in amt: 361
```

```
In [68]: sns.boxplot(data["multiple_deliveries"])
plt.show()
```



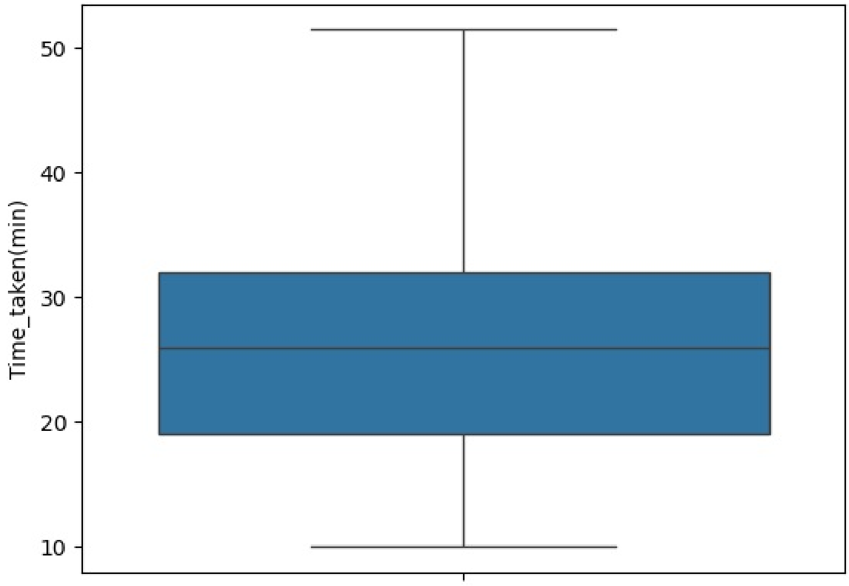
In [69]:
```
sns.boxplot(data["Time_taken(min)"])
plt.show()
```



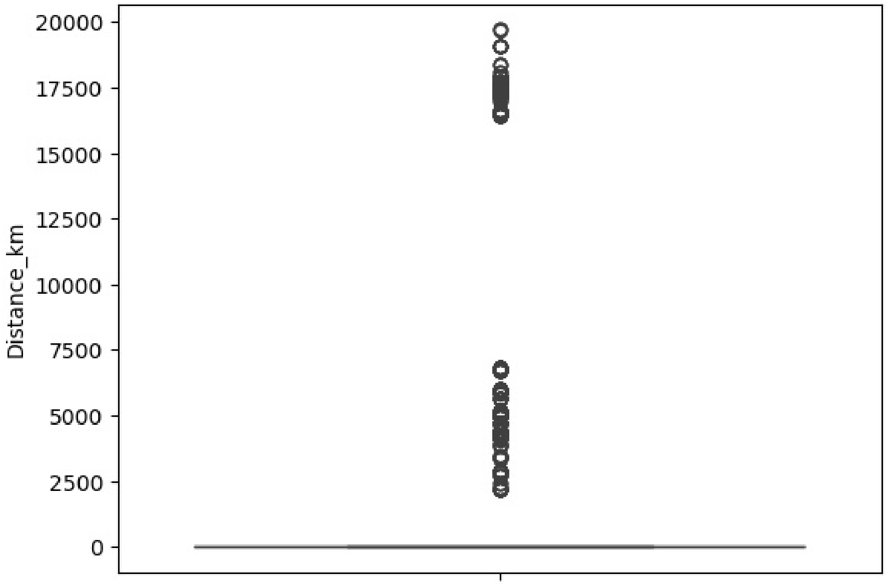
In [70]:
```
Q1 = data['Time_taken(min)'].quantile(0.25)
Q3 = data['Time_taken(min)'].quantile(0.75)
IQR = Q3 - Q1

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

outliers = data[(data['Time_taken(min)'] < lower_bound) | (data['Time_taken(min)'] > upper_bound)]
print(f"Number of outliers in amt: {len(outliers)}")

data['Time_taken(min)'] = np.where(data['Time_taken(min)'] < lower_bound, lower_bound, data['Time_taken(min)'])
data['Time_taken(min)'] = np.where(data['Time_taken(min)'] > upper_bound, upper_bound, data['Time_taken(min)'])
```

```
Number of outliers in amt: 270
```

In [71]:
```
sns.boxplot(data["Time_taken(min)"])
plt.show()
```



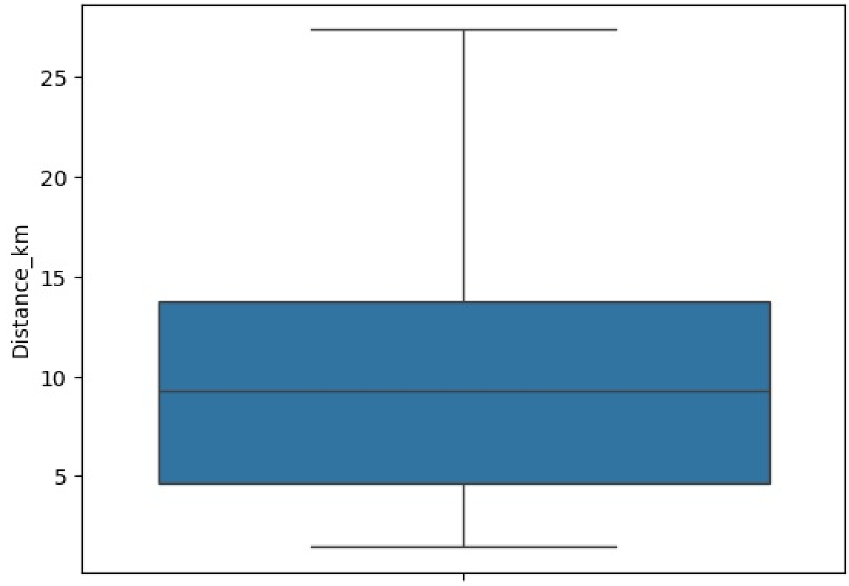
In [72]:
```
sns.boxplot(data["Distance_km"])
plt.show()
```



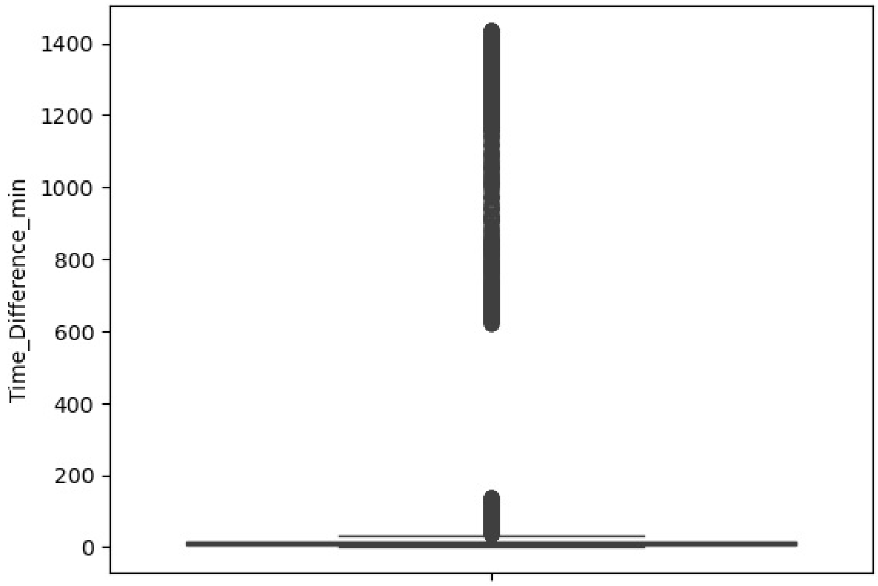
In [73]:
```
Q1 = data['Distance_km'].quantile(0.25)
Q3 = data['Distance_km'].quantile(0.75)
IQR = Q3 - Q1

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

outliers = data[(data['Distance_km'] < lower_bound) | (data['Distance_km'] > upper_bound)]
print(f"Number of outliers in amt: {len(outliers)}")

data['Distance_km'] = np.where(data['Distance_km'] < lower_bound, lower_bound, data['Distance_km'])
data['Distance_km'] = np.where(data['Distance_km'] > upper_bound, upper_bound, data['Distance_km'])
```

```
Number of outliers in amt: 431
```

In [74]:
```
sns.boxplot(data["Distance_km"])
plt.show()
```



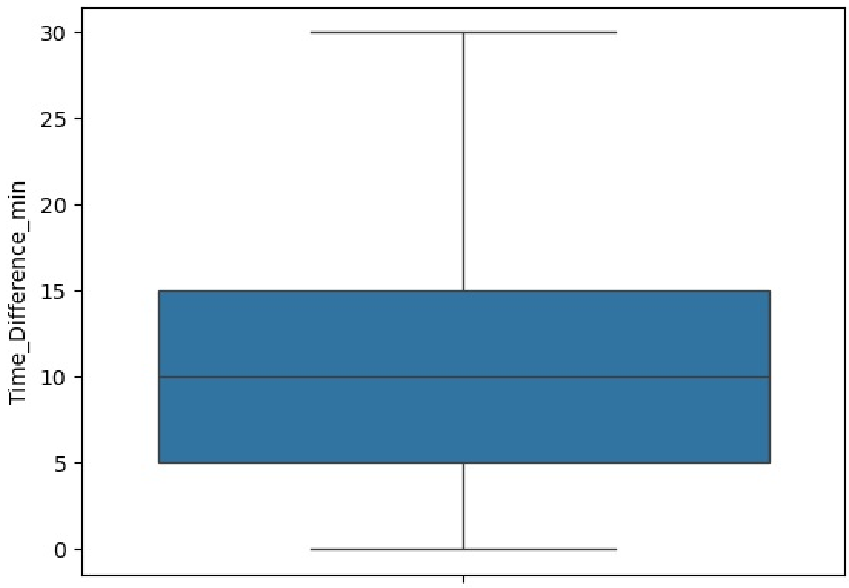
In [75]:
```
sns.boxplot(data["Time_Difference_min"])
plt.show()
```



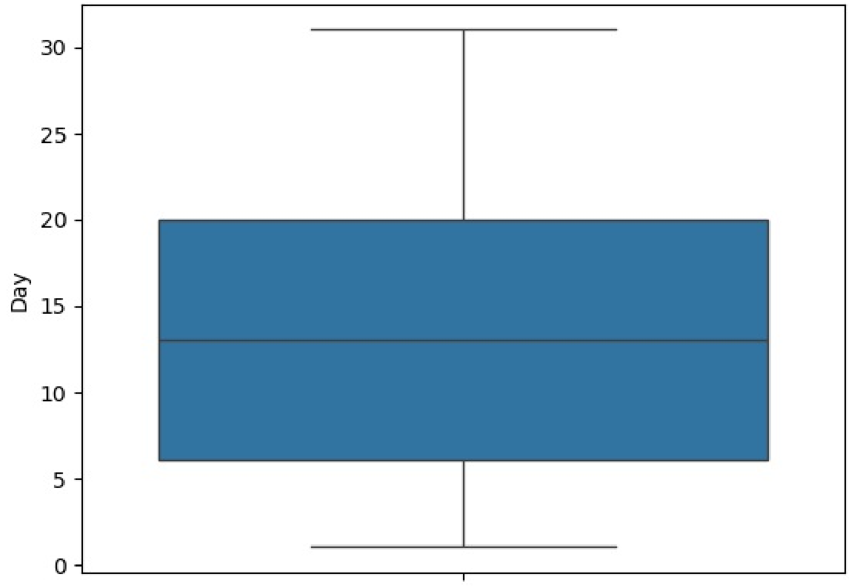
In [76]:
```
Q1 = data['Time_Difference_min'].quantile(0.25)
Q3 = data['Time_Difference_min'].quantile(0.75)
IQR = Q3 - Q1

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

outliers = data[(data['Time_Difference_min'] < lower_bound) | (data['Time_Difference_min'] > upper_bound)]
print(f"Number of outliers in amt: {len(outliers)}")

data['Time_Difference_min'] = np.where(data['Time_Difference_min'] < lower_bound, lower_bound, data['Time_Diffe
data['Time_Difference_min'] = np.where(data['Time_Difference_min'] > upper_bound, upper_bound, data['Time_Diffe
```

```
Number of outliers in amt: 1644
```

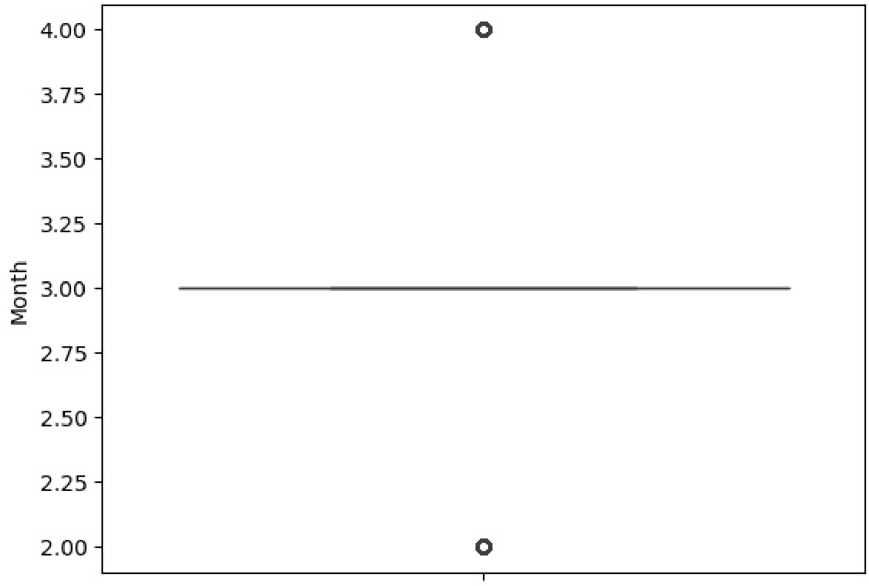
In [77]:
```
sns.boxplot(data["Time_Difference_min"])
plt.show()
```



In [78]:
```
sns.boxplot(data["Day"])
plt.show()
```



In [79]:
```
sns.boxplot(data["Month"])
plt.show()
```



In [80]:
```
Q1 = data['Month'].quantile(0.25)
Q3 = data['Month'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
```

```python
upper_bound = Q3 + 1.5 * IQR

outliers = data[(data['Month'] < lower_bound) | (data['Month'] > upper_bound)]
print(f"Number of outliers in amt: {len(outliers)}")

data['Month'] = np.where(data['Month'] < lower_bound, lower_bound, data['Month'])
data['Month'] = np.where(data['Month'] > upper_bound, upper_bound, data['Month'])
```

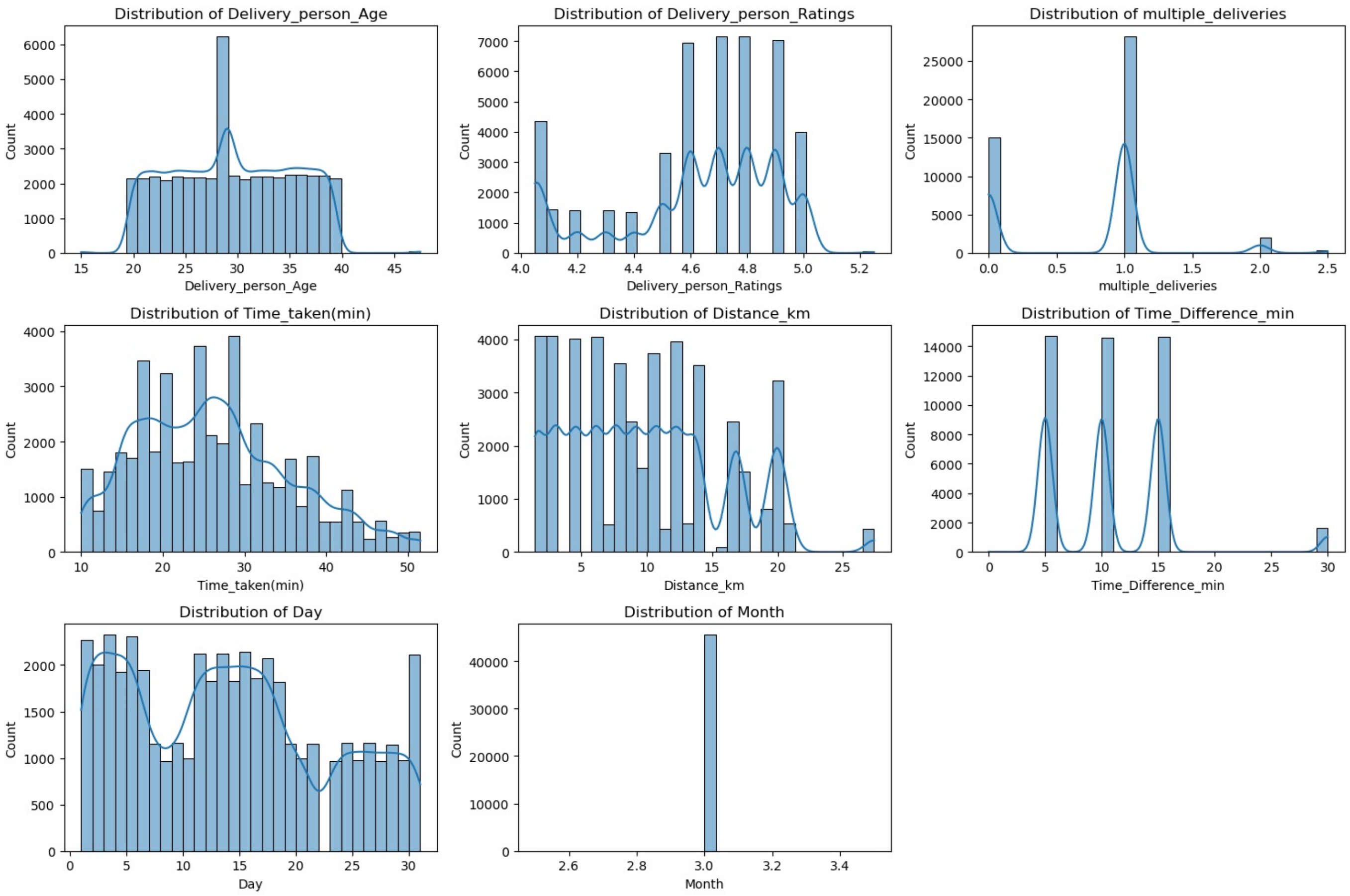
```
Number of outliers in amt: 13604
```

In [81]:
```python
sns.boxplot(data["Month"])
plt.show()
```

Visualization Plots

Distribution Plots of Numerical Values

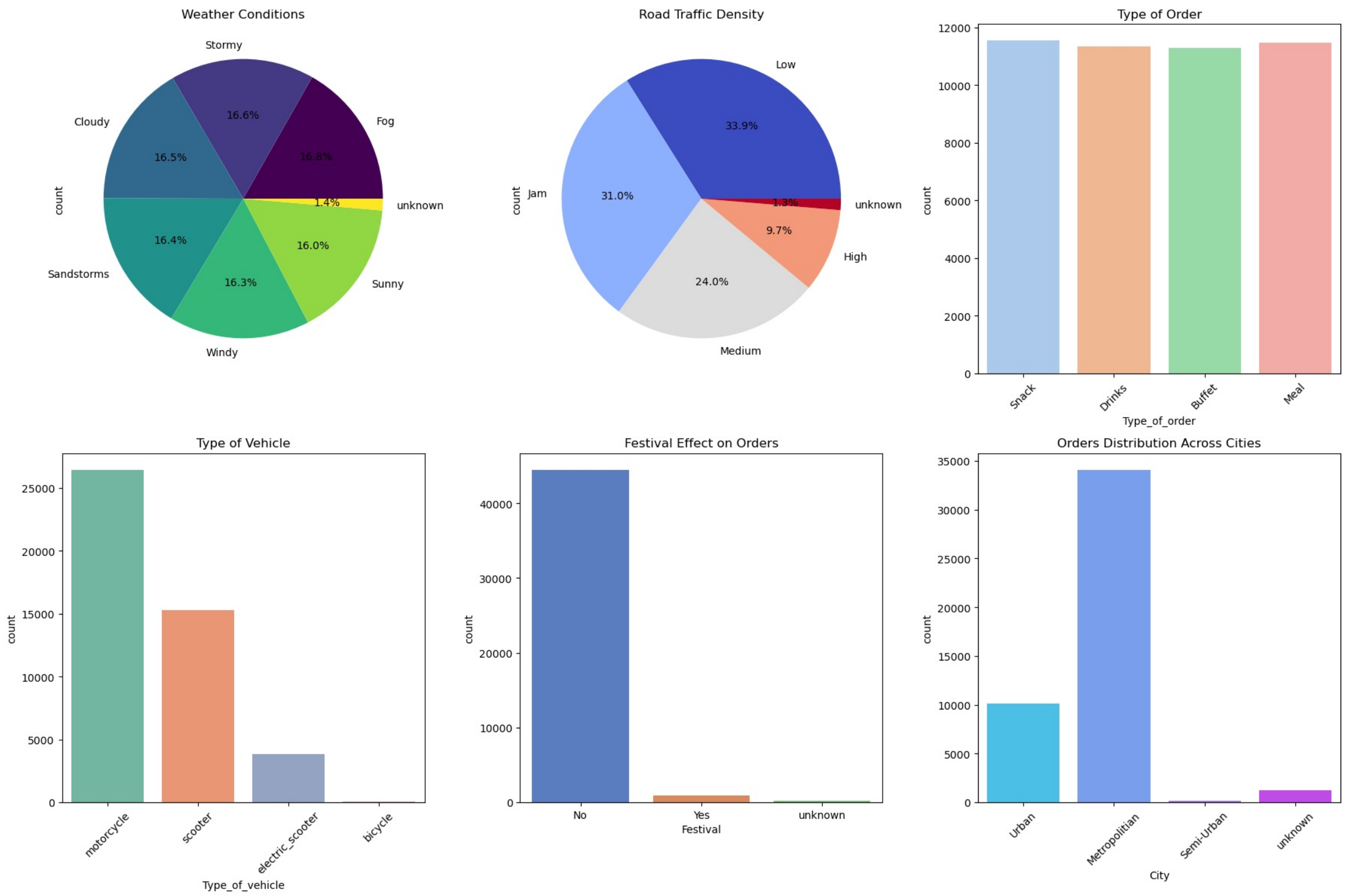
In [82]:
```python
numeric_cols = ["Delivery_person_Age", "Delivery_person_Ratings", "multiple_deliveries",
                "Time_taken(min)", "Distance_km", "Time_Difference_min","Day", "Month"]
plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(data[col], kde=True, bins=30)
    plt.title(f"Distribution of {col}")
plt.tight_layout()
plt.show()
```



Distribution Plots of Categorical Values

In [83]:

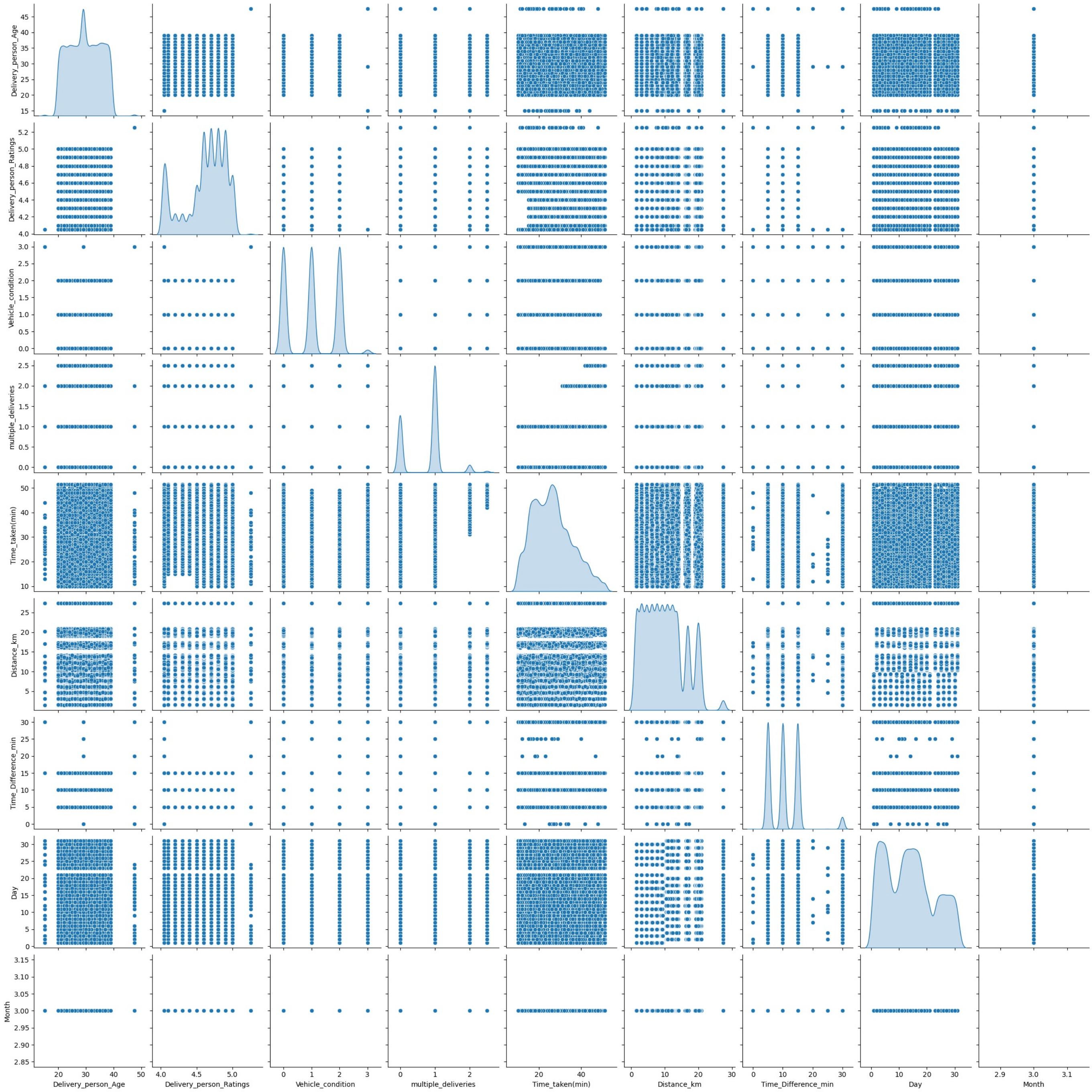
```
categorical_cols = ["Weatherconditions", "Road_traffic_density", "Vehicle_condition",
                    "Type_of_order", "Type_of_vehicle", "Festival", "City"]

# Setting figure size
plt.figure(figsize=(18, 12))

# Creating Pie Charts
plt.subplot(2, 3, 1)
data["Weatherconditions"].value_counts().plot.pie(autopct='%1.1f%%', cmap='viridis')
plt.title("Weather Conditions")

plt.subplot(2, 3, 2)
data["Road_traffic_density"].value_counts().plot.pie(autopct='%1.1f%%', cmap='coolwarm')
plt.title("Road Traffic Density")

# Creating Bar Plots
plt.subplot(2, 3, 3)
sns.countplot(x="Type_of_order", data=data, palette="pastel")
plt.xticks(rotation=45)
plt.title("Type of Order")

plt.subplot(2, 3, 4)
sns.countplot(x="Type_of_vehicle", data=data, palette="Set2")
plt.xticks(rotation=45)
plt.title("Type of Vehicle")

plt.subplot(2, 3, 5)
sns.countplot(x="Festival", data=data, palette="muted")
plt.title("Festival Effect on Orders")

plt.subplot(2, 3, 6)
sns.countplot(x="City", data=data, palette="cool")
plt.xticks(rotation=45)
plt.title("Orders Distribution Across Cities")

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



Pair Plot of All Columns

In [119…

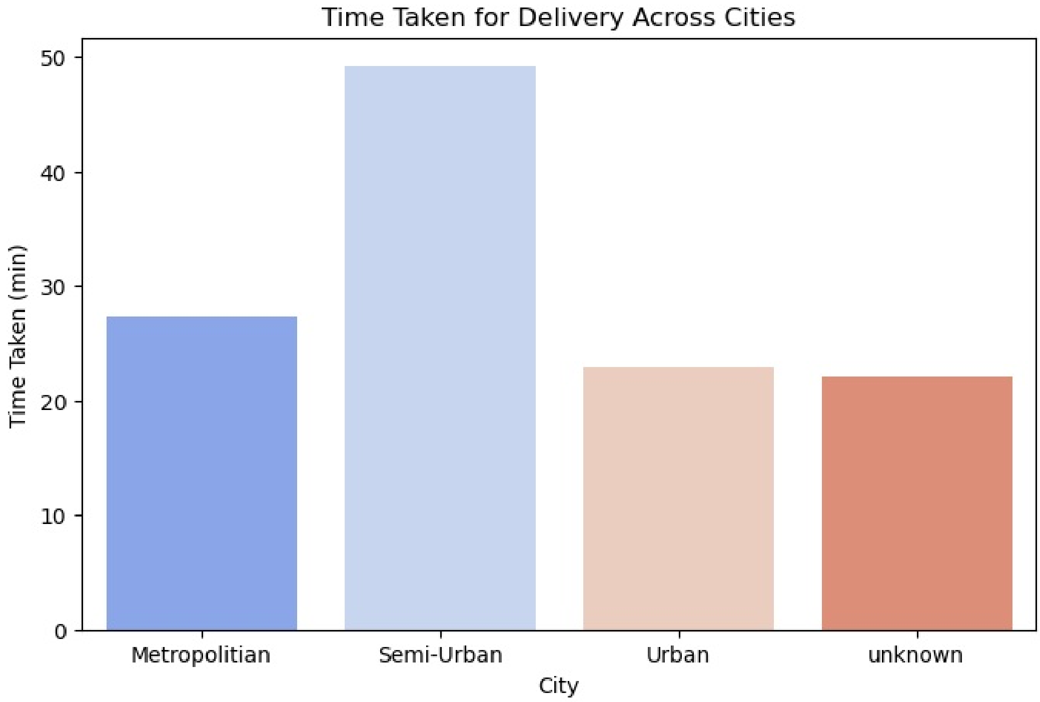
```
sns.pairplot(data, diag_kind='kde')
plt.show()
```



Bar Chart of Time Taken by City

In [120…

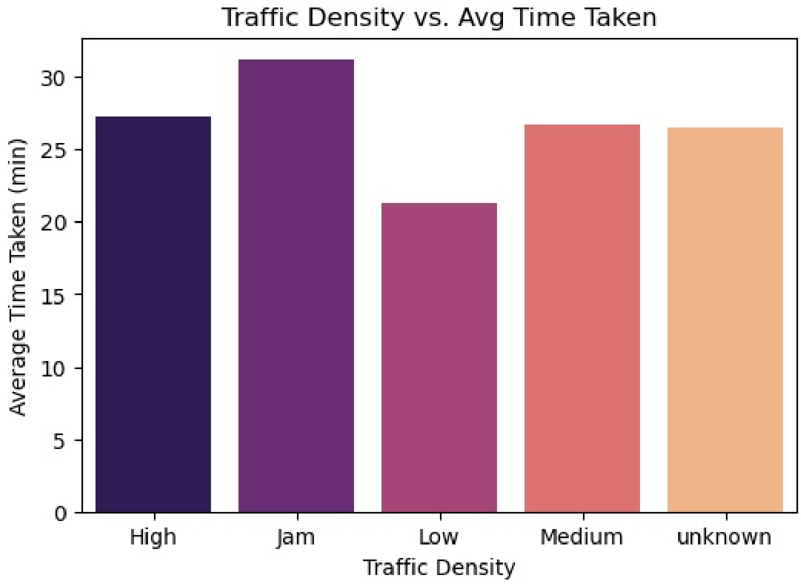
```
plt.figure(figsize=(8,5))
df_city = data.groupby("City")["Time_taken(min)"].mean().reset_index()
sns.barplot(x='City', y='Time_taken(min)', data=df_city, palette='coolwarm')
plt.title("Time Taken for Delivery Across Cities")
plt.xlabel("City")
plt.ylabel("Time Taken (min)")
plt.show()
```



Bar Chart of Traffic Density vs. Average Time Taken

In [121…

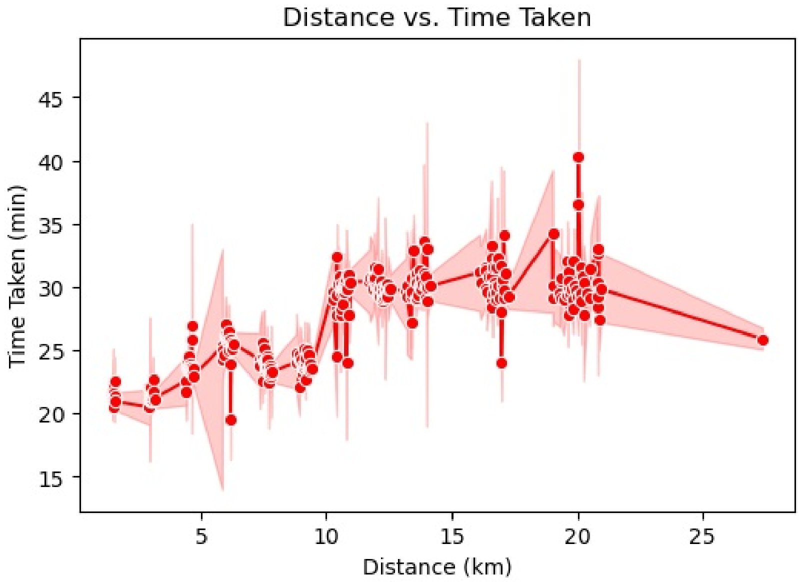
```
plt.figure(figsize=(6,4))
df_traffic = data.groupby("Road_traffic_density")["Time_taken(min)"].mean().reset_index()
sns.barplot(x='Road_traffic_density', y='Time_taken(min)', data=df_traffic, palette='magma')
plt.title("Traffic Density vs. Avg Time Taken")
plt.xlabel("Traffic Density")
plt.ylabel("Average Time Taken (min)")
plt.show()
```



Line Plot of Distance vs. Time Taken

In [122…

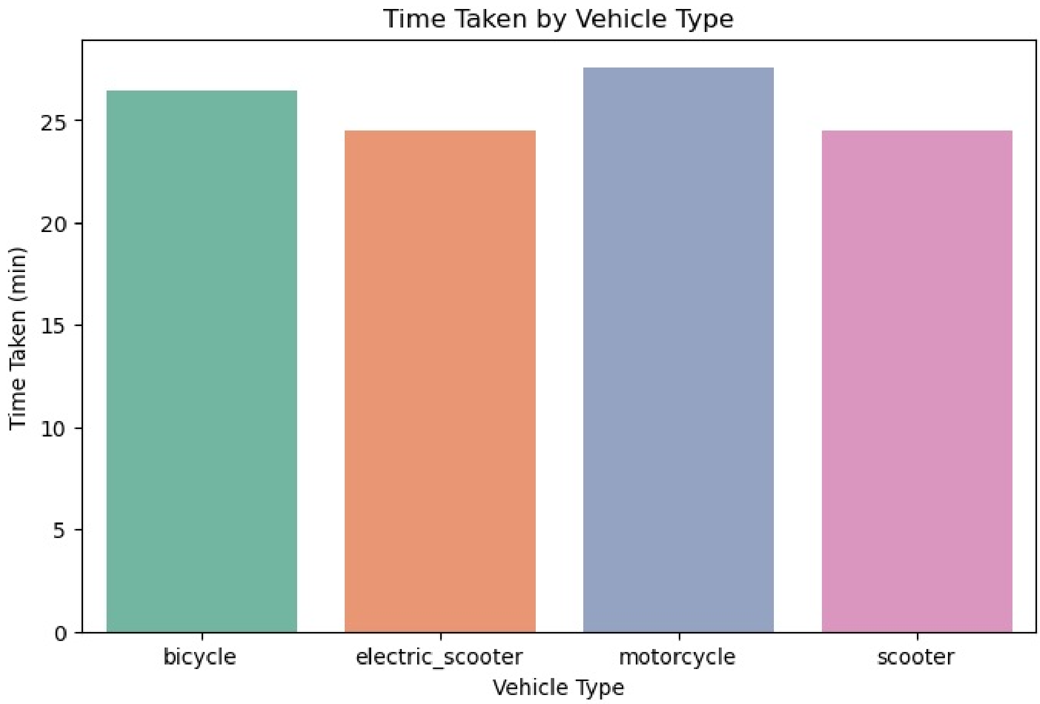
```
plt.figure(figsize=(6,4))
sns.lineplot(x='Distance_km', y='Time_taken(min)', data=data, marker='o', color='red')
plt.title("Distance vs. Time Taken")
plt.xlabel("Distance (km)")
plt.ylabel("Time Taken (min)")
plt.show()
```



Bar Plot of Time Taken by Vehicle Type

In [123…

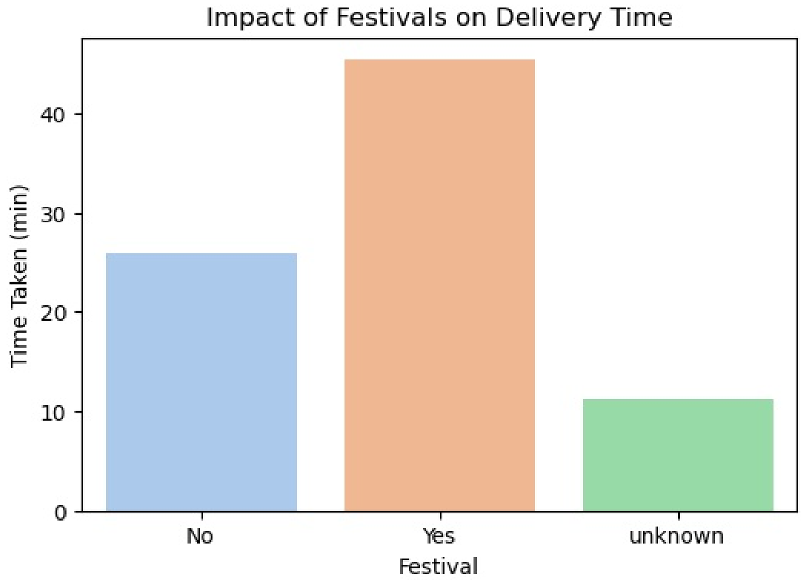
```
plt.figure(figsize=(8,5))
df_vehicle = data.groupby("Type_of_vehicle")["Time_taken(min)"].mean().reset_index()
sns.barplot(x='Type_of_vehicle', y='Time_taken(min)', data=df_vehicle, palette='Set2')
plt.title("Time Taken by Vehicle Type")
plt.xlabel("Vehicle Type")
plt.ylabel("Time Taken (min)")
plt.show()
```



Bar Chart of Festival vs. Time Taken

In [124…

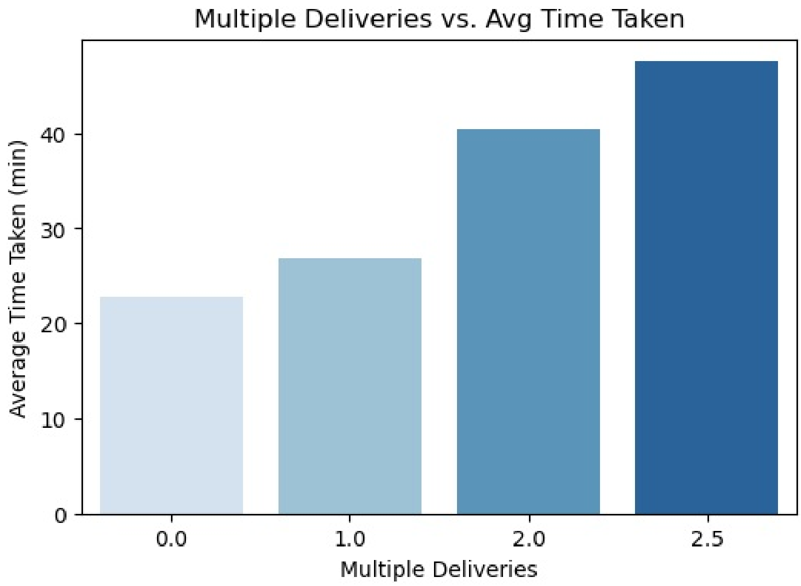
```
plt.figure(figsize=(6,4))
df_festival = data.groupby("Festival")["Time_taken(min)"].mean().reset_index()
sns.barplot(x='Festival', y='Time_taken(min)', data=df_festival, palette='pastel')
plt.title("Impact of Festivals on Delivery Time")
plt.xlabel("Festival")
plt.ylabel("Time Taken (min)")
plt.show()
```



Bar Chart of Multiple Deliveries vs. Average Time Taken

In [125…

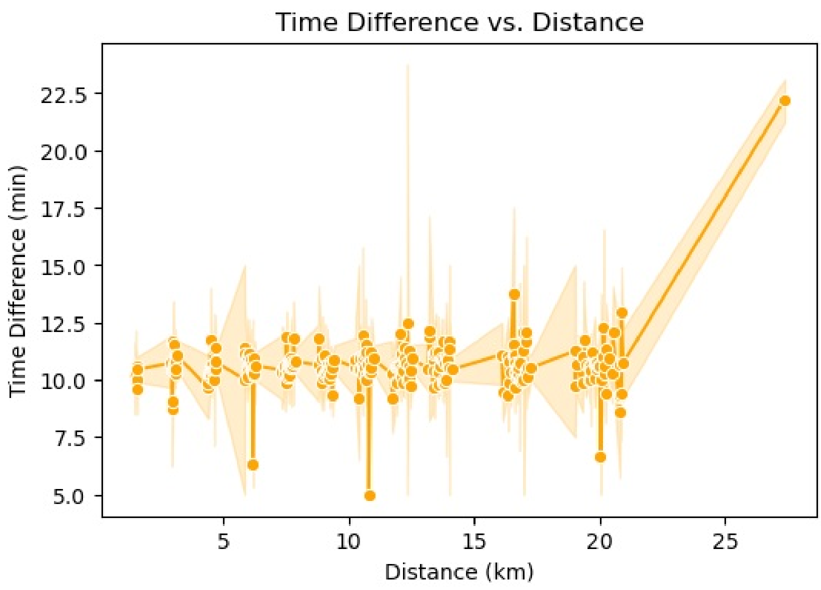
```
plt.figure(figsize=(6,4))
df_multiple = data.groupby("multiple_deliveries")["Time_taken(min)"].mean().reset_index()
sns.barplot(x='multiple_deliveries', y='Time_taken(min)', data=df_multiple, palette='Blues')
plt.title("Multiple Deliveries vs. Avg Time Taken")
plt.xlabel("Multiple Deliveries")
plt.ylabel("Average Time Taken (min)")
plt.show()
```



Line Plot of Time Difference vs. Distance

In [126…

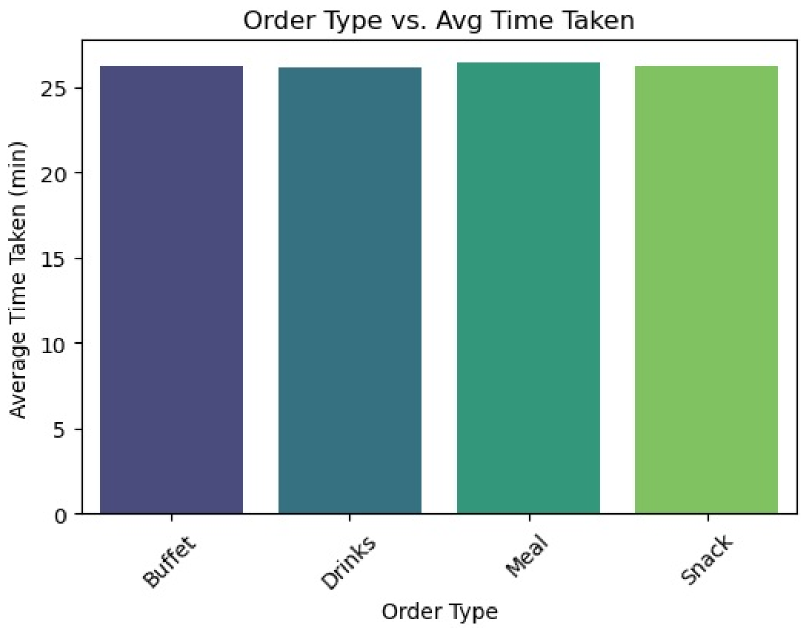
```
plt.figure(figsize=(6,4))
sns.lineplot(x='Distance_km', y='Time_Difference_min', data=data, marker='o', color='orange')
plt.title("Time Difference vs. Distance")
plt.xlabel("Distance (km)")
plt.ylabel("Time Difference (min)")
plt.show()
```



Bar Chart of Order Type vs. Average Time Taken

In [127…

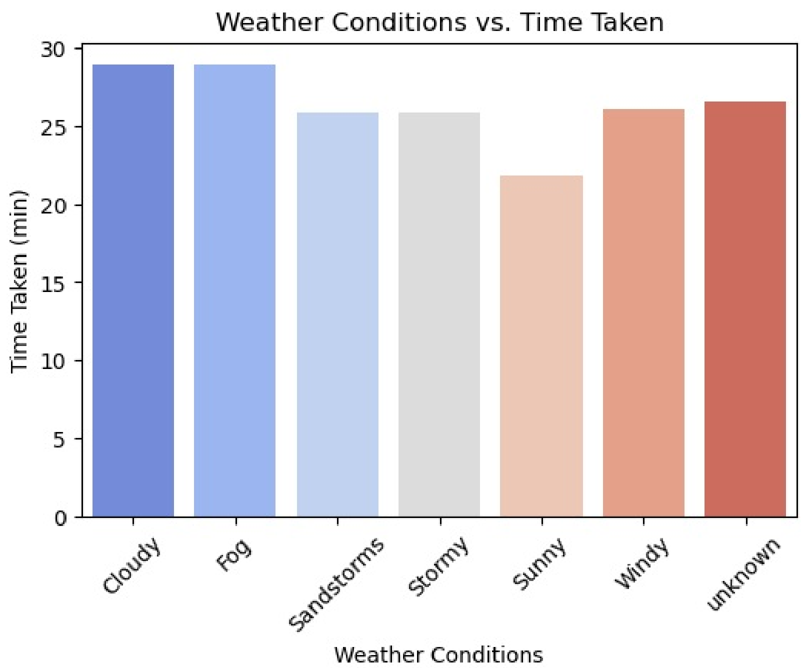
```
plt.figure(figsize=(6,4))
df_order_type = data.groupby("Type_of_order")["Time_taken(min)"].mean().reset_index()
sns.barplot(x='Type_of_order', y='Time_taken(min)', data=df_order_type, palette='viridis')
plt.title("Order Type vs. Avg Time Taken")
plt.xlabel("Order Type")
plt.ylabel("Average Time Taken (min)")
plt.xticks(rotation=45)
plt.show()
```



Bar Chart of Weather Conditions vs. Time Taken

In [128…

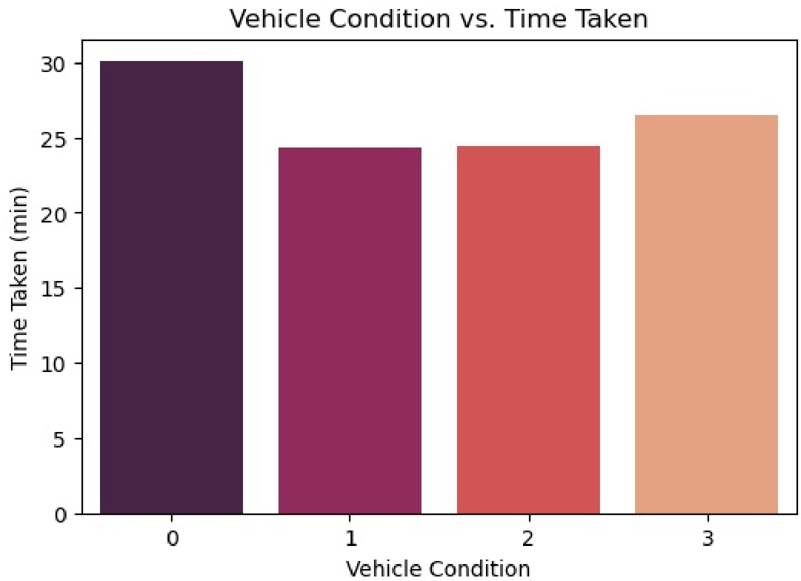
```
plt.figure(figsize=(6,4))
df_weather = data.groupby("Weatherconditions")["Time_taken(min)"].mean().reset_index()
sns.barplot(x='Weatherconditions', y='Time_taken(min)', data=df_weather, palette='coolwarm')
plt.title("Weather Conditions vs. Time Taken")
plt.xlabel("Weather Conditions")
plt.ylabel("Time Taken (min)")
plt.xticks(rotation=45)
plt.show()
```



Bar Chart of Vehicle Condition vs. Time Taken

In [129…

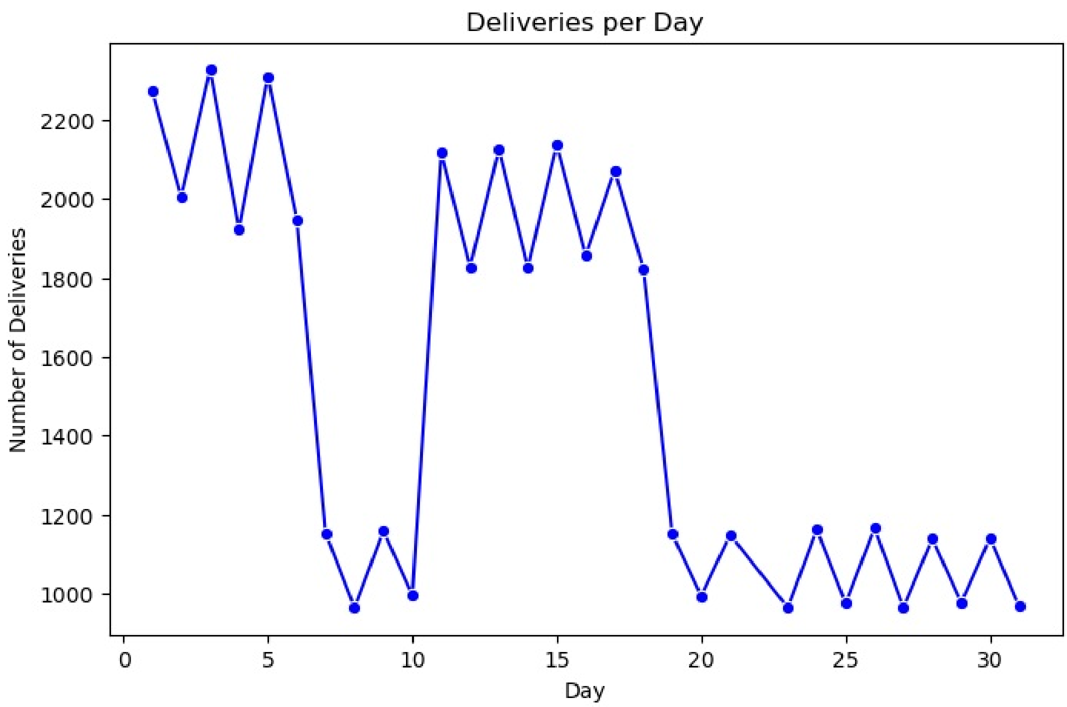
```
plt.figure(figsize=(6,4))
df_vehicle_condition = data.groupby("Vehicle_condition")["Time_taken(min)"].mean().reset_index()
sns.barplot(x='Vehicle_condition', y='Time_taken(min)', data=df_vehicle_condition, palette='rocket')
plt.title("Vehicle Condition vs. Time Taken")
plt.xlabel("Vehicle Condition")
plt.ylabel("Time Taken (min)")
plt.show()
```



Line Plot of Day vs. Deliveries

In [130…

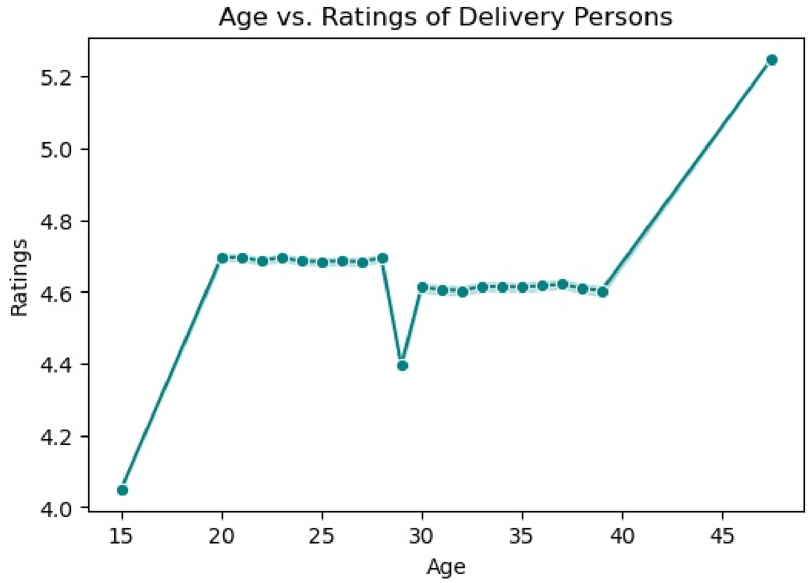
```
plt.figure(figsize=(8,5))
df_day = data.groupby("Day").size().reset_index(name='Deliveries')
sns.lineplot(x='Day', y='Deliveries', data=df_day, marker="o", color='blue')
plt.title("Deliveries per Day")
plt.xlabel("Day")
plt.ylabel("Number of Deliveries")
plt.show()
```



Line Plot of Delivery Person Age vs. Ratings

In [131…

```
plt.figure(figsize=(6,4))
sns.lineplot(x='Delivery_person_Age', y='Delivery_person_Ratings', data=data, marker='o', color='teal')
plt.title("Age vs. Ratings of Delivery Persons")
plt.xlabel("Age")
plt.ylabel("Ratings")
plt.show()
```

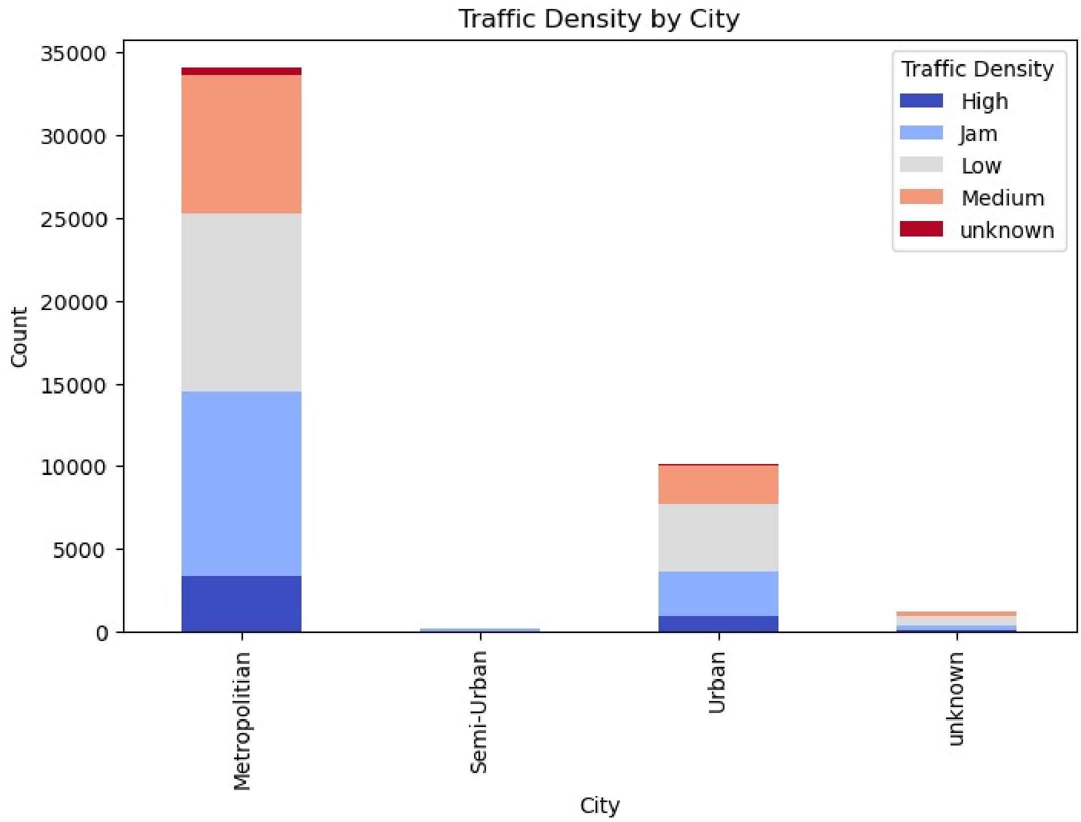


Bar Chart of City vs. Road Traffic Density

In [132…

```
plt.figure(figsize=(8,5))
df_city_traffic = data.groupby("City")["Road_traffic_density"].value_counts().unstack().fillna(0)
df_city_traffic.plot(kind='bar', stacked=True, colormap='coolwarm', figsize=(8,5))
plt.title("Traffic Density by City")
plt.xlabel("City")
plt.ylabel("Count")
plt.legend(title="Traffic Density")
plt.show()
```

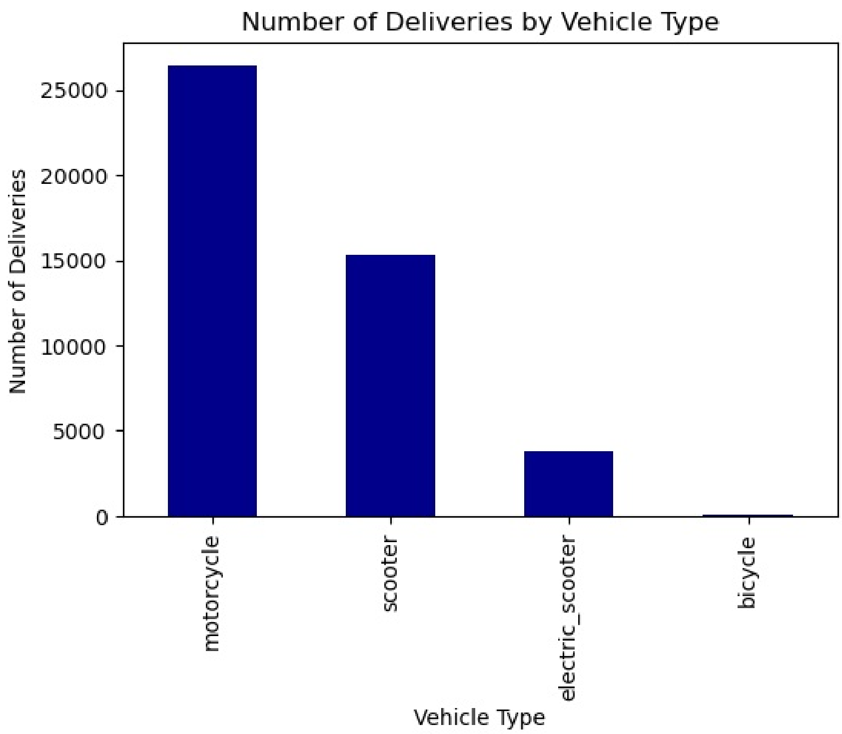
```
<Figure size 800x500 with 0 Axes>
```



Bar Chart of Vehicle Type vs. Number of Deliveries

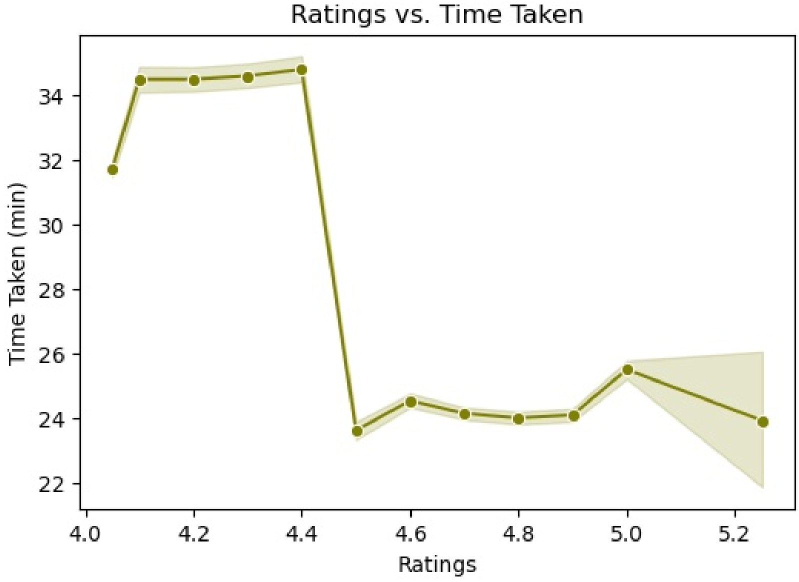
In [133…

```
plt.figure(figsize=(6,4))
df_vehicle_type = data['Type_of_vehicle'].value_counts()
df_vehicle_type.plot(kind='bar', color='darkblue')
plt.title("Number of Deliveries by Vehicle Type")
plt.xlabel("Vehicle Type")
plt.ylabel("Number of Deliveries")
plt.show()
```



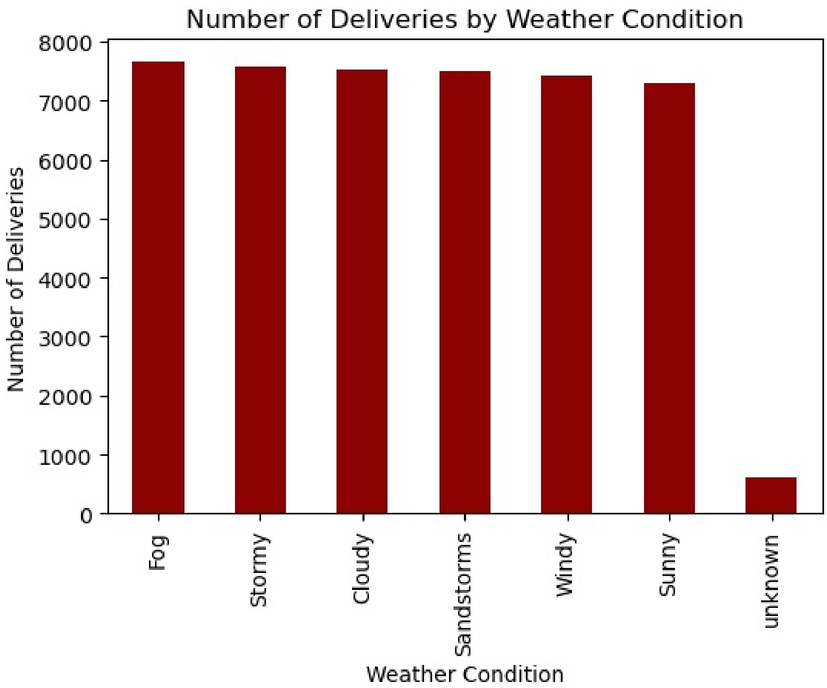
Line Plot of Delivery Person Ratings vs. Time Taken

In [134…
```
plt.figure(figsize=(6,4))
sns.lineplot(x='Delivery_person_Ratings', y='Time_taken(min)', data=data, marker='o', color='olive')
plt.title("Ratings vs. Time Taken")
plt.xlabel("Ratings")
plt.ylabel("Time Taken (min)")
plt.show()
```

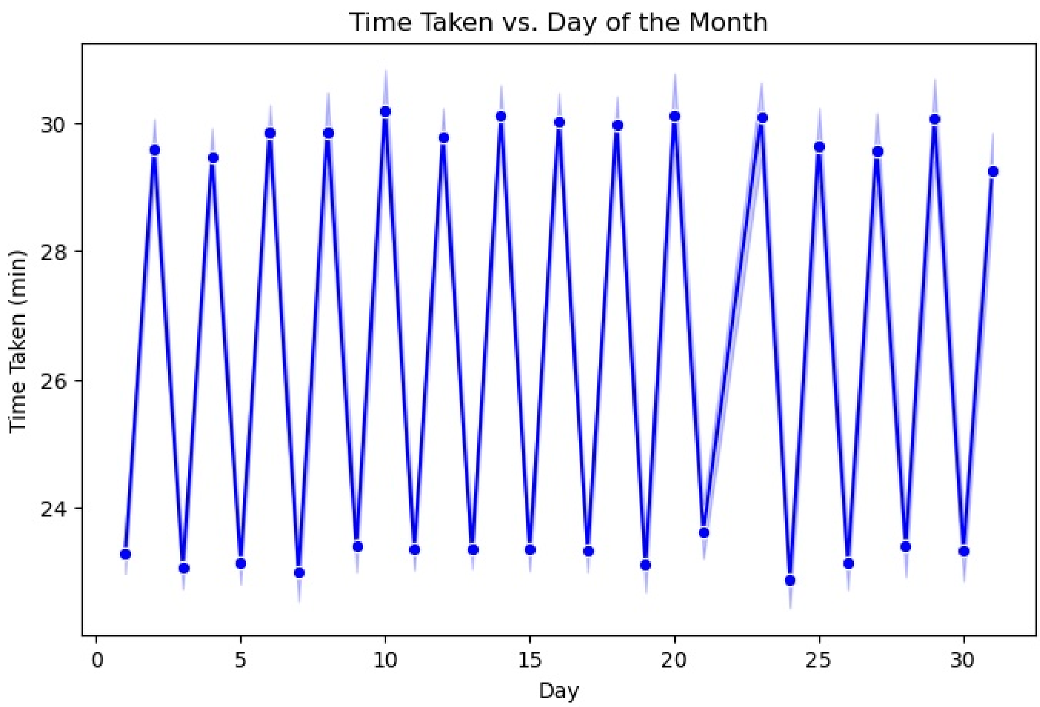


Bar Chart of Weather Conditions vs. Number of Deliveries

In [135…
```
plt.figure(figsize=(6,4))
df_weather_deliveries = data['Weatherconditions'].value_counts()
df_weather_deliveries.plot(kind='bar', color='darkred')
plt.title("Number of Deliveries by Weather Condition")
plt.xlabel("Weather Condition")
plt.ylabel("Number of Deliveries")
plt.show()
```



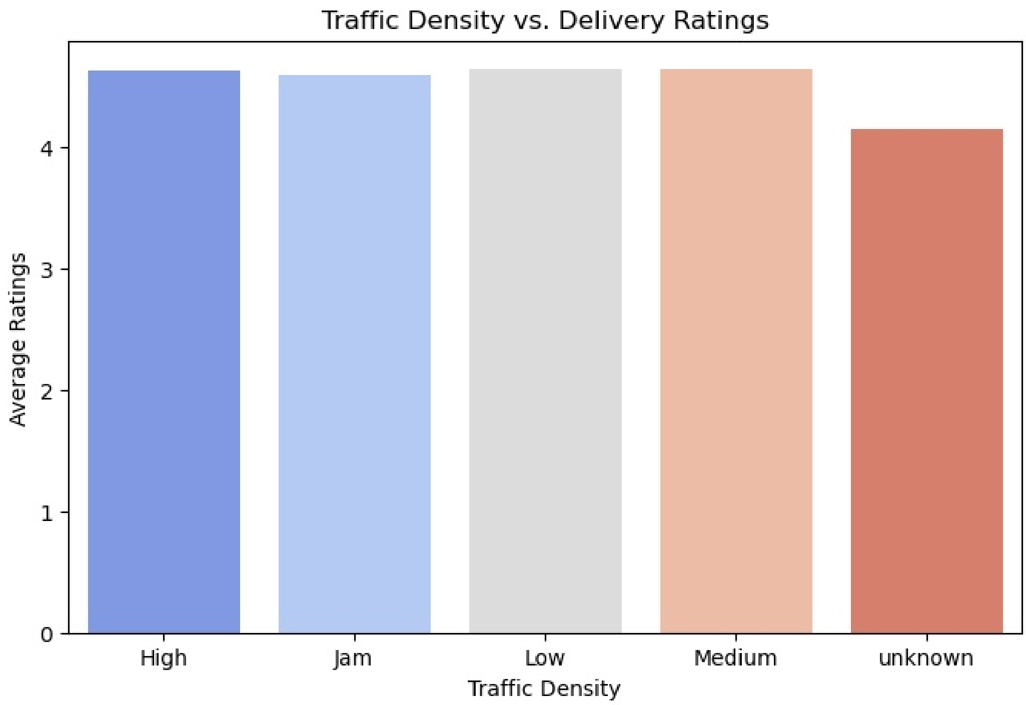
In [101…
```
# 19. Line Plot of Time Taken vs. Day of the Month
plt.figure(figsize=(8,5))
sns.lineplot(x='Day', y='Time_taken(min)', data=data, marker='o', color='blue')
plt.title("Time Taken vs. Day of the Month")
plt.xlabel("Day")
plt.ylabel("Time Taken (min)")
plt.show()
```



Bar Chart of Traffic Density vs. Delivery Ratings

In [136…

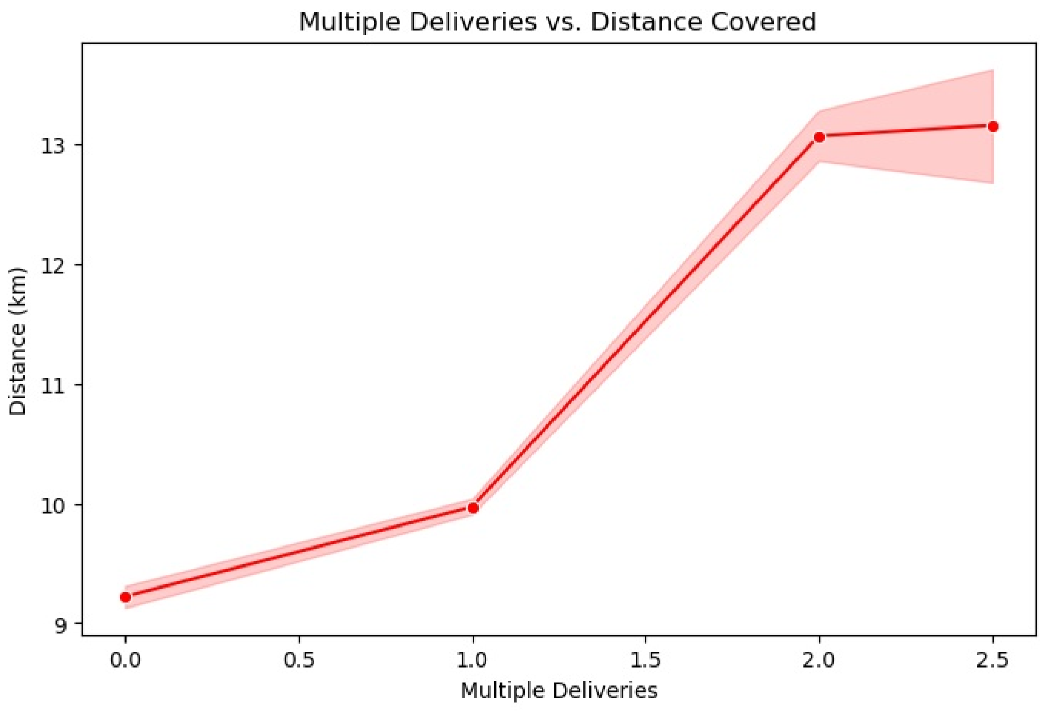
```
plt.figure(figsize=(8,5))
df_traffic_ratings = data.groupby("Road_traffic_density")["Delivery_person_Ratings"].mean().reset_index()
sns.barplot(x='Road_traffic_density', y='Delivery_person_Ratings', data=df_traffic_ratings, palette='coolwarm')
plt.title("Traffic Density vs. Delivery Ratings")
plt.xlabel("Traffic Density")
plt.ylabel("Average Ratings")
plt.show()
```



Line Plot of Multiple Deliveries vs. Distance Covered

In [137…

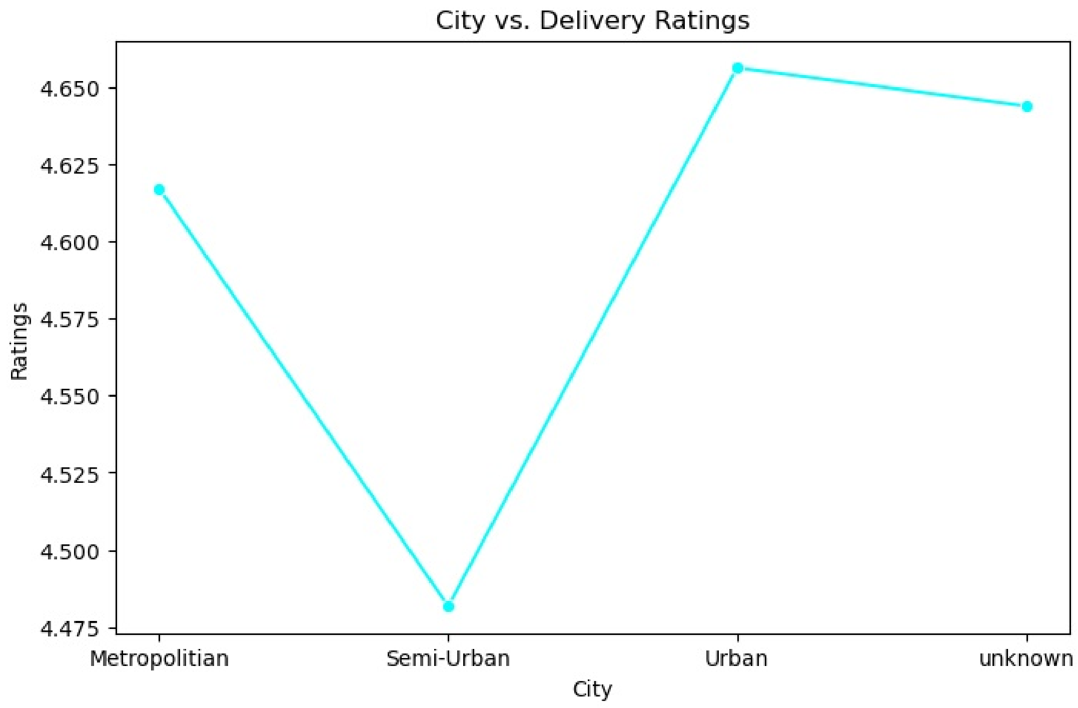
```
plt.figure(figsize=(8,5))
sns.lineplot(x='multiple_deliveries', y='Distance_km', data=data, marker='o', color='red')
plt.title("Multiple Deliveries vs. Distance Covered")
plt.xlabel("Multiple Deliveries")
plt.ylabel("Distance (km)")
plt.show()
```



Line Plot of City vs. Delivery Ratings

In [138…

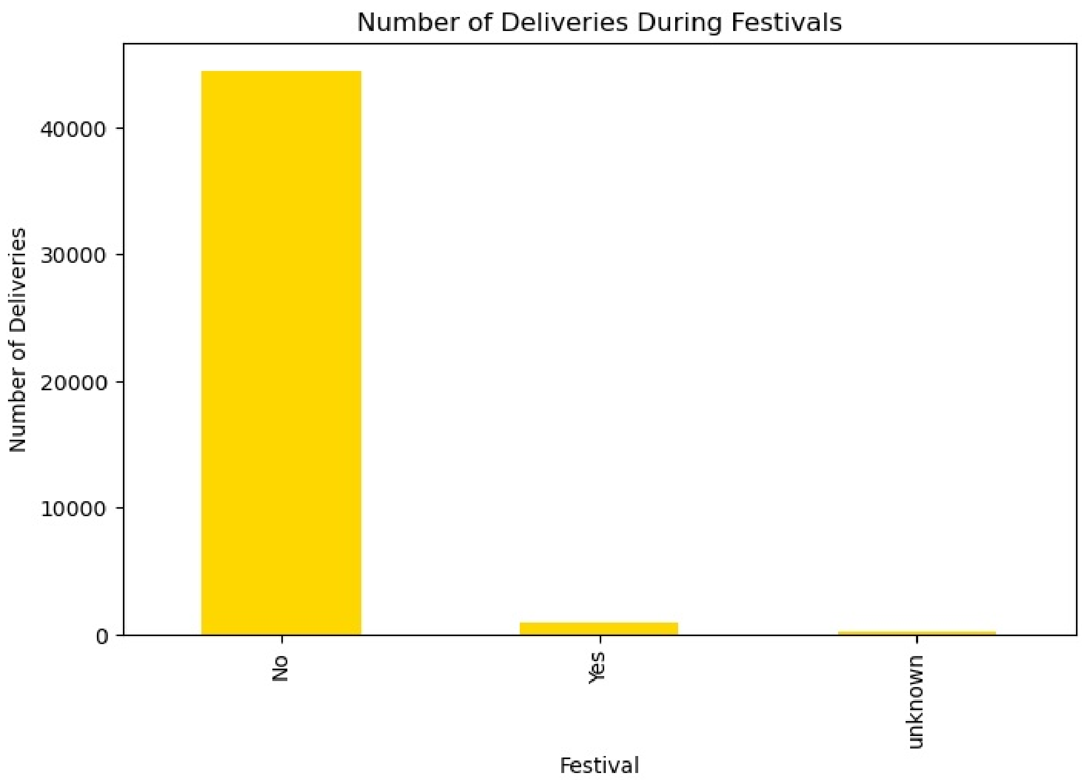
```python
plt.figure(figsize=(8,5))
df_city_ratings = data.groupby("City")["Delivery_person_Ratings"].mean().reset_index()
sns.lineplot(x='City', y='Delivery_person_Ratings', data=df_city_ratings, marker='o', color='cyan')
plt.title("City vs. Delivery Ratings")
plt.xlabel("City")
plt.ylabel("Ratings")
plt.show()
```



Bar Chart of Festival vs. Number of Deliveries

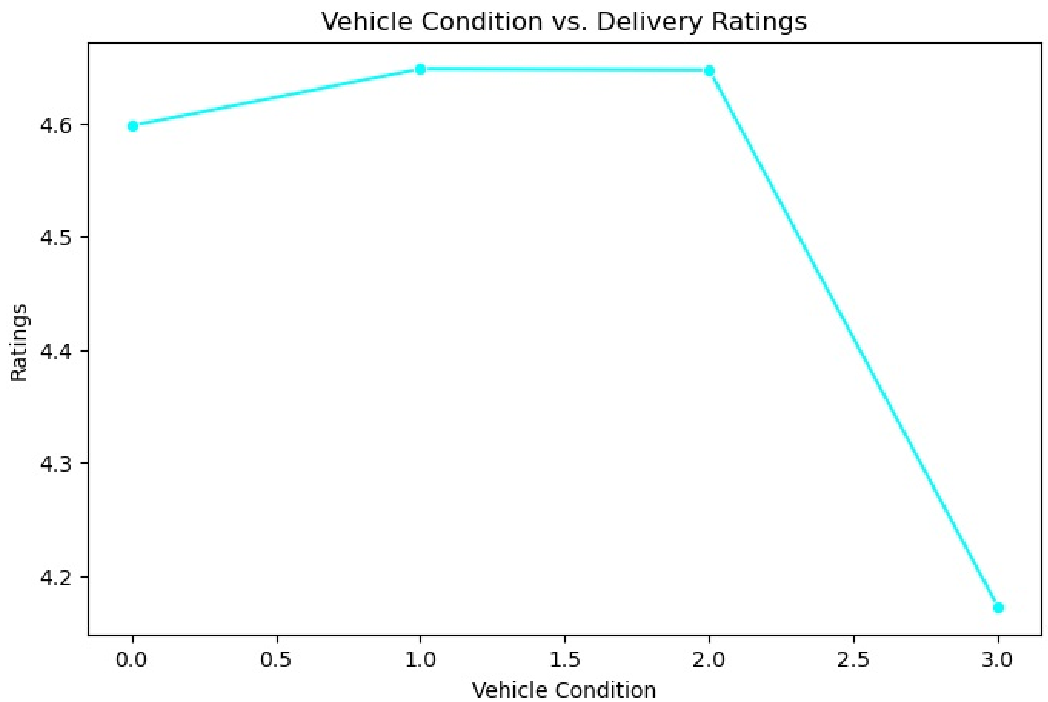
In [139…

```python
plt.figure(figsize=(8,5))
df_festival_deliveries = data['Festival'].value_counts()
df_festival_deliveries.plot(kind='bar', color='gold')
plt.title("Number of Deliveries During Festivals")
plt.xlabel("Festival")
plt.ylabel("Number of Deliveries")
plt.show()
```



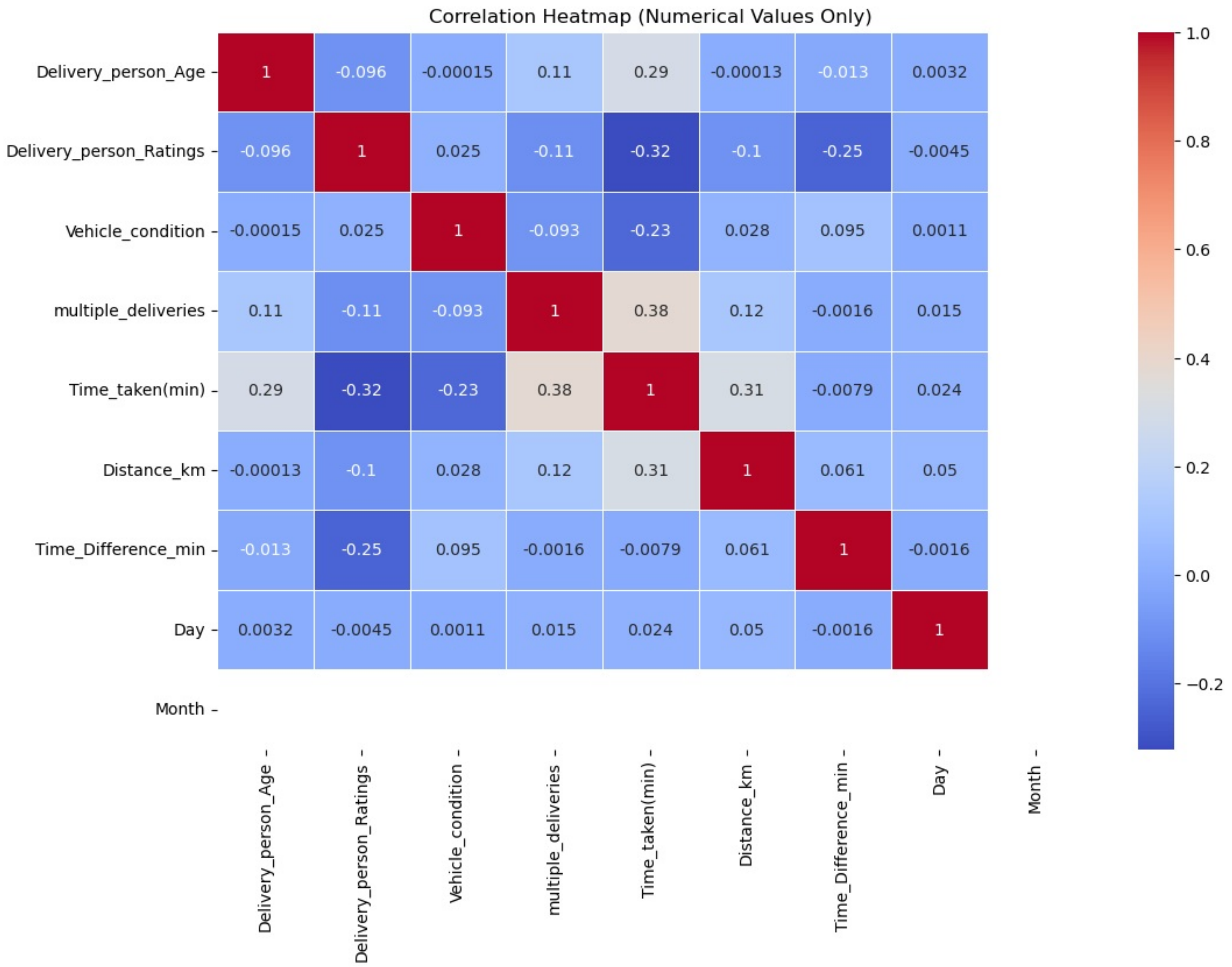
Line Plot of Vehicle Condition vs. Delivery Ratings

```
In [140…
plt.figure(figsize=(8,5))
df_vehicle_ratings = data.groupby("Vehicle_condition")["Delivery_person_Ratings"].mean().reset_index()
sns.lineplot(x='Vehicle_condition', y='Delivery_person_Ratings', data=df_vehicle_ratings, marker='o', color='cya
plt.title("Vehicle Condition vs. Delivery Ratings")
plt.xlabel("Vehicle Condition")
plt.ylabel("Ratings")
plt.show()
```



Heatmap

```
In [107…
numeric_df = data.select_dtypes(include=['number'])
plt.figure(figsize=(12,8))
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Correlation Heatmap (Numerical Values Only)")
plt.show()
```



Machine Learning

Scikit Libraries

In [108…

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Make a copy of the dataset

In [141…

```
df_copy = data.copy()
```

Encode categorical features using Label Encoding

In [146…

```
label_encoders = {}
for column in df_copy.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df_copy[column] = le.fit_transform(df_copy[column])
    label_encoders[column] = le
```

In [111…

```
df_copy
```

Out[111…

	Delivery_person_Age	Delivery_person_Ratings	Weatherconditions	Road_traffic_density	Vehicle_condition	Type_of_order	T
0	37.0	4.9	4	0	2	3	
1	34.0	4.5	3	1	2	3	
2	23.0	4.4	2	2	0	1	
3	38.0	4.7	4	3	0	0	
4	32.0	4.6	0	0	1	3	
...	...	...	...	...	...	...	
45588	30.0	4.8	5	0	1	2	
45589	21.0	4.6	5	1	0	0	
45590	30.0	4.9	0	2	1	1	
45591	20.0	4.7	0	0	0	3	
45592	23.0	4.9	1	3	2	3	

45593 rows × 15 columns

Define target variable and features

In [148…
```python
y = df_copy['Time_taken(min)']
X = df_copy.drop(columns=['Time_taken(min)'])
```

Splitting data into train and test sets

In [149…
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Standardizing numerical features

In [150…
```python
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Define regression models with optimizations to reduce overfitting

In [152…
```python
models = {
    "Simple Linear Regression": LinearRegression(),
    "Multiple Linear Regression": LinearRegression(),
    "Support Vector Regression (SVR)": SVR(kernel='rbf', C=1.0, epsilon=0.1),
    "Decision Tree": DecisionTreeRegressor(max_depth=5, min_samples_split=10),
    "Random Forest": RandomForestRegressor(n_estimators=100, max_depth=10, min_samples_split=10),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=5),
    "XGBoost": XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=5, objective='reg:squarederror')
}

# Store model results
results = {}
```

Train and evaluate models

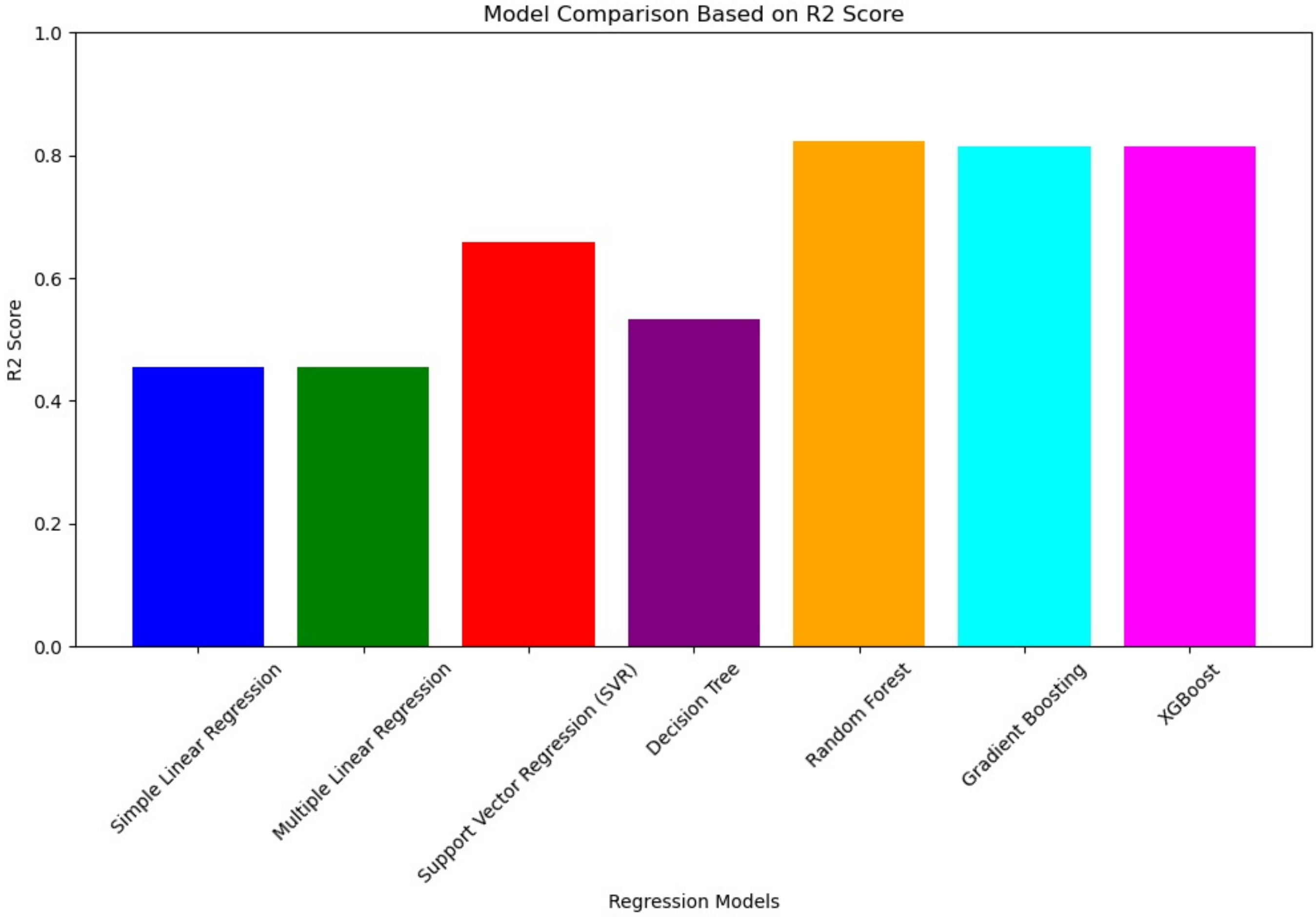
In [153…
```python
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results[name] = r2_score(y_test, y_pred)
    print(f"{name} Performance:")
    print(f"MAE: {mean_absolute_error(y_test, y_pred):.2f}")
    print(f"MSE: {mean_squared_error(y_test, y_pred):.2f}")
    print(f"R2 Score: {r2_score(y_test, y_pred):.2f}\n")
```

```
Simple Linear Regression Performance:
MAE: 5.49
MSE: 47.49
R2 Score: 0.46

Multiple Linear Regression Performance:
MAE: 5.49
MSE: 47.49
R2 Score: 0.46

Support Vector Regression (SVR) Performance:
MAE: 4.29
MSE: 29.77
R2 Score: 0.66

Decision Tree Performance:
MAE: 5.05
MSE: 40.79
R2 Score: 0.53

Random Forest Performance:
MAE: 3.14
MSE: 15.42
R2 Score: 0.82

Gradient Boosting Performance:
MAE: 3.20
MSE: 16.12
R2 Score: 0.82

XGBoost Performance:
MAE: 3.21
MSE: 16.21
R2 Score: 0.81
```

Model Comparison Graph

In [154…

```
plt.figure(figsize=(12, 6))
plt.bar(results.keys(), results.values(), color=['blue', 'green', 'red', 'purple', 'orange', 'cyan', 'magenta']
plt.xlabel("Regression Models")
plt.ylabel("R2 Score")
plt.title("Model Comparison Based on R2 Score")
plt.xticks(rotation=45)
plt.ylim(0, 1)
plt.show()
```



Train and evaluate models using cross-validation

In [155…

```
for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='r2')
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    results[name] = {"Train R2": train_r2, "Test R2": test_r2, "CV Mean R2": scores.mean()}

    print(f"{name} Performance:")
    print(f"Train R2 Score: {train_r2:.2f}")
    print(f"Test R2 Score: {test_r2:.2f}")
    print(f"Cross-Validation Mean R2 Score: {scores.mean():.2f}")
    print("-" * 40)
```

```
Simple Linear Regression Performance:
Train R2 Score: 0.46
Test R2 Score: 0.46
Cross-Validation Mean R2 Score: 0.45
----------------------------------------
Multiple Linear Regression Performance:
Train R2 Score: 0.46
Test R2 Score: 0.46
Cross-Validation Mean R2 Score: 0.45
----------------------------------------
Support Vector Regression (SVR) Performance:
Train R2 Score: 0.67
Test R2 Score: 0.66
Cross-Validation Mean R2 Score: 0.65
----------------------------------------
Decision Tree Performance:
Train R2 Score: 0.54
Test R2 Score: 0.53
Cross-Validation Mean R2 Score: 0.54
----------------------------------------
Random Forest Performance:
Train R2 Score: 0.84
Test R2 Score: 0.82
Cross-Validation Mean R2 Score: 0.82
----------------------------------------
Gradient Boosting Performance:
Train R2 Score: 0.83
Test R2 Score: 0.82
Cross-Validation Mean R2 Score: 0.81
----------------------------------------
XGBoost Performance:
Train R2 Score: 0.82
Test R2 Score: 0.81
Cross-Validation Mean R2 Score: 0.81
----------------------------------------
```

Final Recommendations

✅ Best models: Random Forest, Gradient Boosting, XGBoost (Highest R^2 and low overfitting)

In [ ]:

In [ ]: