Notebook

February 25, 2024

1 Import Libraries

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

!pip install pycaret
from pycaret.regression import RegressionExperiment

from sklearn.model_selection import train_test_split
import lightgbm as lgb
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

2 Load and Understand Data

[ ]:
```
from google.colab import files
files.upload()
```

[3]:
```
df_train = pd.read_csv('train.csv')
df_train.head()
```

[3]:
```
        ID Delivery_person_ID Delivery_person_Age Delivery_person_Ratings  \
0  0x4607     INDORES13DEL02                  37                     4.9
1  0xb379     BANGRES18DEL02                  34                     4.5
2  0x5d6d     BANGRES19DEL01                  23                     4.4
3  0x7a6a    COIMBRES13DEL02                  38                     4.7
4  0x70a2     CHENRES12DEL01                  32                     4.6

   Restaurant_latitude  Restaurant_longitude  Delivery_location_latitude  \
0            22.745049             75.892471                   22.765049
1            12.913041             77.683237                   13.043041
```

```
2            12.914264            77.678400                  12.924264
3            11.003669            76.976494                  11.053669
4            12.972793            80.249982                  13.012793

   Delivery_location_longitude  Order_Date Time_Orderd Time_Order_picked  \
0                    75.912471  19-03-2022    11:30:00          11:45:00
1                    77.813237  25-03-2022    19:45:00          19:50:00
2                    77.688400  19-03-2022    08:30:00          08:45:00
3                    77.026494  05-04-2022    18:00:00          18:10:00
4                    80.289982  26-03-2022    13:30:00          13:45:00

       Weatherconditions Road_traffic_density  Vehicle_condition  \
0       conditions Sunny                High                  2
1      conditions Stormy                 Jam                  2
2  conditions Sandstorms                 Low                  0
3       conditions Sunny              Medium                  0
4      conditions Cloudy                High                  1

  Type_of_order Type_of_vehicle multiple_deliveries Festival            City  \
0        Snack      motorcycle                   0       No           Urban
1        Snack         scooter                   1       No    Metropolitian
2       Drinks      motorcycle                   1       No           Urban
3       Buffet      motorcycle                   1       No    Metropolitian
4        Snack         scooter                   1       No    Metropolitian

  Time_taken(min)
0        (min) 24
1        (min) 33
2        (min) 26
3        (min) 21
4        (min) 30
```

[4]:
```python
df_train.columns
```

[4]:
```
Index(['ID', 'Delivery_person_ID', 'Delivery_person_Age',
       'Delivery_person_Ratings', 'Restaurant_latitude',
       'Restaurant_longitude', 'Delivery_location_latitude',
       'Delivery_location_longitude', 'Order_Date', 'Time_Orderd',
       'Time_Order_picked', 'Weatherconditions', 'Road_traffic_density',
       'Vehicle_condition', 'Type_of_order', 'Type_of_vehicle',
       'multiple_deliveries', 'Festival', 'City', 'Time_taken(min)'],
      dtype='object')
```

[5]:
```python
df_train.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
```

[6]:
```python
df_train.shape
```

[6]:
```
(45593, 20)
```

[7]:
```python
df_train.describe().T
```

[7]:
```
                               count       mean        std        min  \
Restaurant_latitude          45593.0  17.017729   8.185109 -30.905562
Restaurant_longitude         45593.0  70.231332  22.883647 -88.366217
Delivery_location_latitude   45593.0  17.465186   7.335122   0.010000
Delivery_location_longitude  45593.0  70.845702  21.118812   0.010000
Vehicle_condition            45593.0   1.023359   0.839065   0.000000

                                   25%        50%        75%        max
Restaurant_latitude          12.933284  18.546947  22.728163  30.914057
Restaurant_longitude         73.170000  75.898497  78.044095  88.433452
Delivery_location_latitude   12.988453  18.633934  22.785049  31.054057
Delivery_location_longitude  73.280000  76.002574  78.107044  88.563452
Vehicle_condition             0.000000   1.000000   2.000000   3.000000
```

[8]:
```python
df_train.describe(exclude=np.number).T
```

[8]:

```
                          count unique            top   freq
ID                        45593  45593         0x4607      1
Delivery_person_ID        45593   1320  PUNERES01DEL01     67
Delivery_person_Age       45593     23              35   2262
Delivery_person_Ratings   45593     29             4.8   7148
Order_Date                45593     44      15-03-2022   1192
Time_Orderd               45593    177            NaN    1731
Time_Order_picked         45593    193        21:30:00    496
Weatherconditions         45593      7  conditions Fog   7654
Road_traffic_density      45593      5            Low   15477
Type_of_order             45593      4          Snack   11533
Type_of_vehicle           45593      4     motorcycle   26435
multiple_deliveries       45593      5              1   28159
Festival                  45593      3             No   44469
City                      45593      4   Metropolitian   34093
Time_taken(min)           45593     45        (min) 26   2123
```

[9]:

```
for column in df_train.columns:
    print(column)
    print(df_train[column].value_counts())
    print("------------------------------------")
```

```
ID
0x4607     1
0x1f3e     1
0xe251     1
0x3f31     1
0x4a78     1
          ..
0xc3f1     1
0x5db7     1
0x1985     1
0xceda     1
0x5fb2     1
Name: ID, Length: 45593, dtype: int64
------------------------------------
Delivery_person_ID
PUNERES01DEL01     67
JAPRES11DEL02      67
HYDRES04DEL02      66
JAPRES03DEL01      66
VADRES11DEL02      66
                   ..
DEHRES18DEL03       7
AURGRES11DEL03      7
KOLRES09DEL03       6
KOCRES16DEL03       6
```

```
BHPRES010DEL03      5
Name: Delivery_person_ID, Length: 1320, dtype: int64
------------------------------------
Delivery_person_Age
35      2262
36      2260
37      2227
30      2226
38      2219
24      2210
32      2202
22      2196
29      2191
33      2187
28      2179
25      2174
34      2166
26      2159
21      2153
27      2150
39      2144
20      2136
31      2120
23      2087
NaN     1854
50        53
15        38
Name: Delivery_person_Age, dtype: int64
------------------------------------
Delivery_person_Ratings
4.8     7148
4.7     7142
4.9     7041
4.6     6940
5       3996
4.5     3303
NaN     1908
4.1     1430
4.2     1418
4.3     1409
4.4     1361
4       1077
3.5      249
3.8      228
3.7      225
3.6      207
3.9      197
6         53
```

```
1          38
3.4        32
3.1        29
3.2        29
3.3        25
2.6        22
2.7        22
2.5        20
2.8        19
2.9        19
3           6
Name: Delivery_person_Ratings, dtype: int64
------------------------------------
Restaurant_latitude
 0.000000     3640
 26.911378     182
 26.914142     180
 26.892312     176
 26.902940     176
              …
-23.355164       1
-15.513150       1
-22.311358       1
-27.161661       1
-12.978453       1
Name: Restaurant_latitude, Length: 657, dtype: int64
------------------------------------
Restaurant_longitude
 0.000000     3640
 75.789034     182
 75.805704     181
 75.793007     177
 75.806896     176
              …
-76.626167       1
-85.316842       1
-76.643622       1
-72.814492       1
-77.643685       1
Name: Restaurant_longitude, Length: 518, dtype: int64
------------------------------------
Delivery_location_latitude
0.130000     341
0.020000     337
0.090000     336
0.060000     336
0.070000     335
            …
```

```
19.976969      1
19.916219      1
26.562001      1
23.324249      1
20.005337      1
Name: Delivery_location_latitude, Length: 4373, dtype: int64
------------------------------------
Delivery_location_longitude
0.130000     341
0.020000     337
0.090000     336
0.060000     336
0.070000     335
            …
75.428894      1
75.386017      1
80.444002      1
77.524007      1
75.446722      1
Name: Delivery_location_longitude, Length: 4373, dtype: int64
------------------------------------
Order_Date
15-03-2022    1192
03-04-2022    1178
13-03-2022    1169
26-03-2022    1166
24-03-2022    1162
09-03-2022    1159
05-04-2022    1157
05-03-2022    1154
07-03-2022    1153
03-03-2022    1150
19-03-2022    1150
21-03-2022    1149
11-03-2022    1149
30-03-2022    1141
01-03-2022    1140
28-03-2022    1139
17-03-2022    1134
01-04-2022    1133
02-03-2022    1012
10-03-2022     996
16-03-2022     995
20-03-2022     994
02-04-2022     992
06-03-2022     986
04-03-2022     981
29-03-2022     977
```

```
25-03-2022     975
14-03-2022     974
11-02-2022     970
18-03-2022     968
31-03-2022     967
27-03-2022     965
12-03-2022     964
08-03-2022     964
23-03-2022     964
06-04-2022     961
13-02-2022     957
15-02-2022     945
04-04-2022     941
17-02-2022     939
12-02-2022     864
16-02-2022     861
18-02-2022     855
14-02-2022     851
Name: Order_Date, dtype: int64
------------------------------------
Time_Orderd
NaN         1731
21:55:00     461
17:55:00     456
20:00:00     449
22:20:00     448
            …
12:25:00      57
14:15:00      56
16:00:00      53
13:20:00      52
16:30:00      51
Name: Time_Orderd, Length: 177, dtype: int64
------------------------------------
Time_Order_picked
21:30:00    496
22:50:00    474
22:40:00    458
18:40:00    457
17:55:00    456
           …
15:10:00     48
16:15:00     46
16:10:00     43
17:10:00     39
16:20:00     38
Name: Time_Order_picked, Length: 193, dtype: int64
------------------------------------
```

```
Weatherconditions
conditions Fog           7654
conditions Stormy        7586
conditions Cloudy        7536
conditions Sandstorms    7495
conditions Windy         7422
conditions Sunny         7284
conditions NaN            616
Name: Weatherconditions, dtype: int64
------------------------------------
Road_traffic_density
Low        15477
Jam        14143
Medium     10947
High        4425
NaN          601
Name: Road_traffic_density, dtype: int64
------------------------------------
Vehicle_condition
2    15034
1    15030
0    15009
3      520
Name: Vehicle_condition, dtype: int64
------------------------------------
Type_of_order
Snack      11533
Meal       11458
Drinks     11322
Buffet     11280
Name: Type_of_order, dtype: int64
------------------------------------
Type_of_vehicle
motorcycle          26435
scooter             15276
electric_scooter     3814
bicycle                68
Name: Type_of_vehicle, dtype: int64
------------------------------------
multiple_deliveries
1       28159
0       14095
2        1985
NaN       993
3         361
Name: multiple_deliveries, dtype: int64
------------------------------------
Festival
```

```
No      44469
Yes       896
NaN       228
Name: Festival, dtype: int64
------------------------------------
City
Metropolitian     34093
Urban             10136
NaN                1200
Semi-Urban          164
Name: City, dtype: int64
------------------------------------
Time_taken(min)
(min) 26    2123
(min) 25    2050
(min) 27    1976
(min) 28    1965
(min) 29    1956
(min) 19    1824
(min) 15    1810
(min) 18    1765
(min) 16    1706
(min) 17    1696
(min) 24    1680
(min) 23    1643
(min) 20    1640
(min) 22    1626
(min) 21    1601
(min) 33    1259
(min) 30    1218
(min) 31    1213
(min) 34    1172
(min) 32    1124
(min) 38     887
(min) 36     852
(min) 39     847
(min) 35     832
(min) 37     828
(min) 11     757
(min) 10     750
(min) 12     746
(min) 14     739
(min) 13     716
(min) 43     567
(min) 42     561
(min) 40     555
(min) 41     553
(min) 44     553
```

```
(min) 47     295
(min) 49     280
(min) 48     277
(min) 46     274
(min) 45     241
(min) 53     100
(min) 51      94
(min) 54      91
(min) 52      79
(min) 50      72
Name: Time_taken(min), dtype: int64
------------------------------------
```

[10]:
```
df_train.isnull().sum()
```

[10]:
```
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
```

[11]:
```
df_train.duplicated().sum()
```

[11]:
```
0
```

Column Explanation

- **ID**: Unique key of deliveries
- **Delivery_person_ID**: Code of the delivery person
- **Delivery_person_Ratings**: Ratings of the delivery person, which reflects the quality of his/her service
- **Restaurant_latitude**: The latitude of restaurant

- **Restaurant_longitude**: The longitude of restaurant, whose combination with Restaurant_latitude determines the location
- **Delivery_location_latitude**: The latitude of destination (customer's place)
- **Delivery_location_longitude**: The longitude of destination, whose combination with Delivery_location_latitude determines the location
- **Order_Date**: Date of order
- **Time_Orderd**: Ordered time
- **Time_Order_picked**: Picked-up time. Noted that this can be on the different date from ordered date (midnight orders)
- **Weatherconditions**: Weather conditions during the delivery time
- **Road_traffic_density**: Traffic density during the delivery time
- **Vehicle_condition**: The contemporary condition of the vehicle which reflects its quality and impacts pick-up time and delivery time
- **Type_of_order**: Which kind of food is delivered: drinks, snack, etc. This partly determines preparation time and delivery time.
- **Type_of_vehicle**: Different vehecle types have different speed
- **multiple_deliveries**: Determines whether this order is delivered with others or not. A multiple delivery takes more time than a single one
- **Festival**: Determines whether there is a festival in the delivery area or not. Festival may impact the availability of delivery service, road traffic, prepation time, etc.
- **City**: Type of city (metropolitian, urban, or semi-urban)
- **Time_taken(min)**: Delivery time - the target variable in this project

Observations

- Both **numeric and categorical features** are present
- There are some **unnecessary columns** for the Supervised Learning process: ID, Delivery_person_ID
- Some columns require data formatting: **Weatherconditions**, **Time_taken(min)**
- Some variables should be created based on available columns: **Distance from restaurant to destination** (based on Restaurant_latitude, Restaurant_longitude, Delivery_location_latitude, Delivery_location_longitude), **Preparation time** (based on Time_Ordered, Time_Order_picked)
- There are **null values** across table but they are currently a string which should be transformed for identification

3 Clean Data

[12]:
```
def transform_null(data):

  data = data.copy()

  data.replace('NaN ', pd.NA, inplace=True)
  data['Weatherconditions'].replace('conditions NaN', pd.NA, inplace=True)

  return data

df_train2 = transform_null(df_train)
```

[13]:
```
df_train2.isna().sum()
```

[13]:
```
ID                                0
Delivery_person_ID                0
Delivery_person_Age            1854
Delivery_person_Ratings        1908
Restaurant_latitude               0
Restaurant_longitude              0
Delivery_location_latitude        0
Delivery_location_longitude       0
Order_Date                        0
Time_Orderd                    1731
Time_Order_picked                 0
Weatherconditions               616
Road_traffic_density            601
Vehicle_condition                 0
Type_of_order                     0
Type_of_vehicle                   0
multiple_deliveries             993
Festival                        228
City                           1200
Time_taken(min)                   0
dtype: int64
```

Define a function for data transformation

[14]:
```
def transform_dataframe(data):

  # Convert necessary columns to numeric format
  data['Delivery_person_Age'] = pd.to_numeric(data['Delivery_person_Age'],␣
↪errors='coerce')
  data['Delivery_person_Ratings'] = pd.
↪to_numeric(data['Delivery_person_Ratings'], errors='coerce')
  data['Vehicle_condition'] = pd.to_numeric(data['Vehicle_condition'],␣
↪errors='coerce')
  data['multiple_deliveries'] = pd.to_numeric(data['multiple_deliveries'],␣
↪errors='coerce')

  data = data.rename(columns={'Time_Orderd': 'Time_Ordered'})

  #Convert necessary columns to datetime format
  data['Order_Date'] = pd.to_datetime(data['Order_Date']).dt.date
  data['Time_Ordered'] = pd.to_datetime(data['Time_Ordered']).dt.time
  data['Time_Order_picked'] = pd.to_datetime(data['Time_Order_picked']).dt.time

  # Remove necessary part of columns
```

```
data['Weatherconditions'] = data['Weatherconditions'].str.replace('conditions ','', regex=False)
data['Time_taken(min)'] = pd.to_numeric(data['Time_taken(min)'].str.extract(r'(\d+)', expand=False), errors='coerce')

# Calculate the distance between restaurant and destination
from geopy.distance import geodesic
def calculate_distance(row):
  restaurant_coords = (row['Restaurant_latitude'], row['Restaurant_longitude'])
  delivery_coords = (row['Delivery_location_latitude'], row['Delivery_location_longitude'])
  distance = geodesic(restaurant_coords, delivery_coords).kilometers
  return distance

data['distance(km)'] = data.apply(calculate_distance, axis=1)

# Drop rows with null values in 'Time_Ordered' column
data.dropna(subset=['Time_Ordered'], inplace=True)

# Get the Picked-up date as it can be different from the Ordered date
data['Pick_date'] = data.apply(
    lambda row: row['Order_Date'] + pd.DateOffset(1)
    if pd.notna(row['Time_Ordered']) > pd.notna(row['Time_Order_picked'])
    else row['Order_Date'], axis=1)

data['Datetime_Ordered'] = pd.to_datetime(data['Order_Date'].astype(str) + ' ' + data['Time_Ordered'].astype(str))
data['Datetime_Picked'] = pd.to_datetime(data['Pick_date'].astype(str) + ' ' + data['Time_Order_picked'].astype(str))

# Calculate the Preparation Time of the order
data['Time_Order_prepared'] = (data['Datetime_Picked'] - data['Datetime_Ordered']).dt.total_seconds() / 60.0

# Get the hour and minute
data['Ordered_hour'] = data['Datetime_Ordered'].apply(lambda x: x.hour)
data['Ordered_minute'] = data['Datetime_Ordered'].apply(lambda x: x.minute)
data['Picked_hour'] = data['Datetime_Picked'].apply(lambda x: x.hour)
data['Picked_minute'] = data['Datetime_Picked'].apply(lambda x: x.minute)

# Get the day, month, and weekdate
data['Order_day'] = data['Datetime_Ordered'].dt.day
data['Order_month'] = data['Datetime_Ordered'].dt.month
data['Order_weekdate'] = data['Datetime_Ordered'].dt.day_name()
```

```python
    return data

df_train3 = transform_dataframe(df_train2)
df_train3.head()
```

[14]:

```
       ID Delivery_person_ID  Delivery_person_Age  Delivery_person_Ratings  \
0  0x4607     INDORES13DEL02                 37.0                      4.9
1  0xb379     BANGRES18DEL02                 34.0                      4.5
2  0x5d6d     BANGRES19DEL01                 23.0                      4.4
3  0x7a6a    COIMBRES13DEL02                 38.0                      4.7
4  0x70a2     CHENRES12DEL01                 32.0                      4.6

   Restaurant_latitude  Restaurant_longitude  Delivery_location_latitude  \
0            22.745049             75.892471                   22.765049
1            12.913041             77.683237                   13.043041
2            12.914264             77.678400                   12.924264
3            11.003669             76.976494                   11.053669
4            12.972793             80.249982                   13.012793

   Delivery_location_longitude  Order_Date Time_Ordered  …  \
0                    75.912471  2022-03-19     11:30:00  …
1                    77.813237  2022-03-25     19:45:00  …
2                    77.688400  2022-03-19     08:30:00  …
3                    77.026494  2022-05-04     18:00:00  …
4                    80.289982  2022-03-26     13:30:00  …

     Datetime_Ordered     Datetime_Picked Time_Order_prepared  Ordered_hour  \
0 2022-03-19 11:30:00 2022-03-19 11:45:00                15.0            11
1 2022-03-25 19:45:00 2022-03-25 19:50:00                 5.0            19
2 2022-03-19 08:30:00 2022-03-19 08:45:00                15.0             8
3 2022-05-04 18:00:00 2022-05-04 18:10:00                10.0            18
4 2022-03-26 13:30:00 2022-03-26 13:45:00                15.0            13

  Ordered_minute Picked_hour  Picked_minute Order_day Order_month  \
0             30          11             45        19           3
1             45          19             50        25           3
2             30           8             45        19           3
3              0          18             10         4           5
4             30          13             45        26           3

   Order_weekdate
0        Saturday
1          Friday
2        Saturday
3       Wednesday
4        Saturday
```

```
[5 rows x 32 columns]
```

Define another function to transform the predict data (without the target - Time_taken(min))

```
[15]: def transform_dataframe_without_target(data):

        data['Delivery_person_Age'] = pd.to_numeric(data['Delivery_person_Age'],␣
      ↪errors='coerce')
        data['Delivery_person_Ratings'] = pd.
      ↪to_numeric(data['Delivery_person_Ratings'], errors='coerce')
        data['Vehicle_condition'] = pd.to_numeric(data['Vehicle_condition'],␣
      ↪errors='coerce')
        data['multiple_deliveries'] = pd.to_numeric(data['multiple_deliveries'],␣
      ↪errors='coerce')

        data = data.rename(columns={'Time_Orderd': 'Time_Ordered'})

        data['Order_Date'] = pd.to_datetime(data['Order_Date']).dt.date
        data['Time_Ordered'] = pd.to_datetime(data['Time_Ordered']).dt.time
        data['Time_Order_picked'] = pd.to_datetime(data['Time_Order_picked']).dt.time

        data['Weatherconditions'] = data['Weatherconditions'].str.replace('conditions␣
      ↪','', regex=False)

        from geopy.distance import geodesic
        def calculate_distance(row):
          restaurant_coords = (row['Restaurant_latitude'],␣
      ↪row['Restaurant_longitude'])
          delivery_coords = (row['Delivery_location_latitude'],␣
      ↪row['Delivery_location_longitude'])
          distance = geodesic(restaurant_coords, delivery_coords).kilometers
          return distance

        data['distance(km)'] = data.apply(calculate_distance, axis=1)

        data.dropna(subset=['Time_Ordered'], inplace=True)

        data['Pick_date'] = data.apply(
            lambda row: row['Order_Date'] + pd.DateOffset(1)
            if pd.notna(row['Time_Ordered']) > pd.notna(row['Time_Order_picked'])
            else row['Order_Date'], axis=1)

        data['Datetime_Ordered'] = pd.to_datetime(data['Order_Date'].astype(str) + '␣
      ↪' + data['Time_Ordered'].astype(str))
        data['Datetime_Picked'] = pd.to_datetime(data['Pick_date'].astype(str) + ' '␣
      ↪+ data['Time_Order_picked'].astype(str))
```

```
    data['Time_Order_prepared'] = (data['Datetime_Picked'] -␣
↪data['Datetime_Ordered']).dt.total_seconds() / 60.0


    data['Ordered_hour'] = data['Datetime_Ordered'].apply(lambda x: x.hour)
    data['Ordered_minute'] = data['Datetime_Ordered'].apply(lambda x: x.minute)
    data['Picked_hour'] = data['Datetime_Picked'].apply(lambda x: x.hour)
    data['Picked_minute'] = data['Datetime_Picked'].apply(lambda x: x.minute)

    data['Order_day'] = data['Datetime_Ordered'].dt.day
    data['Order_month'] = data['Datetime_Ordered'].dt.month
    data['Order_weekdate'] = data['Datetime_Ordered'].dt.day_name()

    return data
```

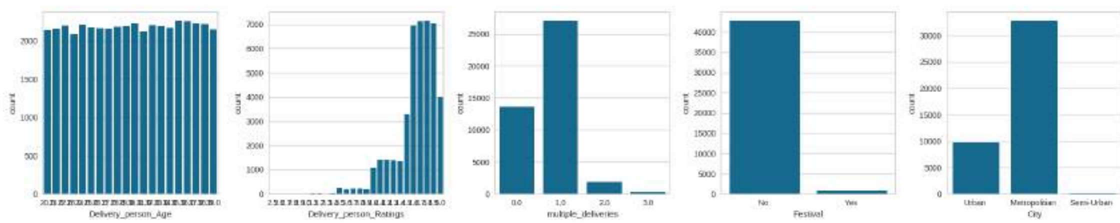
[16]:
```
df_train3.isna().sum()
```

[16]:
```
ID                               0
Delivery_person_ID               0
Delivery_person_Age            214
Delivery_person_Ratings        268
Restaurant_latitude              0
Restaurant_longitude             0
Delivery_location_latitude       0
Delivery_location_longitude      0
Order_Date                       0
Time_Ordered                     0
Time_Order_picked                0
Weatherconditions                0
Road_traffic_density             0
Vehicle_condition                0
Type_of_order                    0
Type_of_vehicle                  0
multiple_deliveries            943
Festival                       219
City                          1144
Time_taken(min)                  0
distance(km)                     0
Pick_date                        0
Datetime_Ordered                 0
Datetime_Picked                  0
Time_Order_prepared              0
Ordered_hour                     0
Ordered_minute                   0
Picked_hour                      0
Picked_minute                    0
```

```
Order_day                          0
Order_month                        0
Order_weekdate                     0
dtype: int64
```

Visualize columns containing nulls

[17]:
```python
fig, axes = plt.subplots(1, 5, figsize=(20, 4))

sns.countplot(data=df_train3, x='Delivery_person_Age', ax=axes[0])
sns.countplot(data=df_train3, x='Delivery_person_Ratings', ax=axes[1])
sns.countplot(data=df_train3, x='multiple_deliveries', ax=axes[2])
sns.countplot(data=df_train3, x='Festival', ax=axes[3])
sns.countplot(data=df_train3, x='City', ax=axes[4])

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



Define a function for null filling

[18]:
```python
columns_fill_mean = ['Delivery_person_Age', 'Delivery_person_Ratings']
columns_fill_mode = ['multiple_deliveries', 'Festival', 'City']

def transform_fill_null(data):

  data = data.copy()

  for column in columns_fill_mean:
    mean_value = data[column].mean()
    data[column].fillna(mean_value, inplace=True)

  for column in columns_fill_mode:
    mode_value = data[column].mode().iloc[0]
    data[column].fillna(mode_value, inplace=True)

  return data

df_train4 = transform_fill_null(df_train3)
```

```
df_train4.isna().sum()
```

[18]:
```
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_Ordered                   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
distance(km)                   0
Pick_date                      0
Datetime_Ordered               0
Datetime_Picked                0
Time_Order_prepared            0
Ordered_hour                   0
Ordered_minute                 0
Picked_hour                    0
Picked_minute                  0
Order_day                      0
Order_month                    0
Order_weekdate                 0
dtype: int64
```

[19]:
```
columns_to_keep = [
    'Delivery_person_Age'
    , 'Delivery_person_Ratings'
    , 'Order_day'
    , 'Order_month'
    , 'Order_weekdate'
    , 'Ordered_hour'
    , 'Ordered_minute'
    , 'Picked_hour'
    , 'Picked_minute'
    , 'Time_Order_prepared'
```

```
    , 'distance(km)'
    , 'Type_of_order'
    , 'Type_of_vehicle'
    , 'multiple_deliveries'
    , 'City'
    , 'Festival'
    , 'Weatherconditions'
    , 'Road_traffic_density'
    , 'Vehicle_condition'
    , 'Time_taken(min)'
]

df_train5 = df_train4[columns_to_keep]
df_train5.head()
```

[19]:
```
   Delivery_person_Age  Delivery_person_Ratings  Order_day  Order_month  \
0                 37.0                      4.9         19            3
1                 34.0                      4.5         25            3
2                 23.0                      4.4         19            3
3                 38.0                      4.7          4            5
4                 32.0                      4.6         26            3

  Order_weekdate  Ordered_hour  Ordered_minute  Picked_hour  Picked_minute  \
0       Saturday            11              30           11             45
1         Friday            19              45           19             50
2       Saturday             8              30            8             45
3      Wednesday            18               0           18             10
4       Saturday            13              30           13             45

   Time_Order_prepared  distance(km) Type_of_order Type_of_vehicle  \
0                 15.0      3.020737        Snack      motorcycle
1                  5.0     20.143737        Snack         scooter
2                 15.0      1.549693       Drinks      motorcycle
3                 10.0      7.774497       Buffet      motorcycle
4                 15.0      6.197898        Snack         scooter

   multiple_deliveries           City Festival Weatherconditions  \
0                  0.0          Urban       No             Sunny
1                  1.0   Metropolitian       No            Stormy
2                  1.0          Urban       No        Sandstorms
3                  1.0   Metropolitian       No             Sunny
4                  1.0   Metropolitian       No            Cloudy

  Road_traffic_density  Vehicle_condition  Time_taken(min)
0                 High                  2               24
1                  Jam                  2               33
2                  Low                  0               26
```

```
3              Medium                   0               21
4                High                   1               30
```

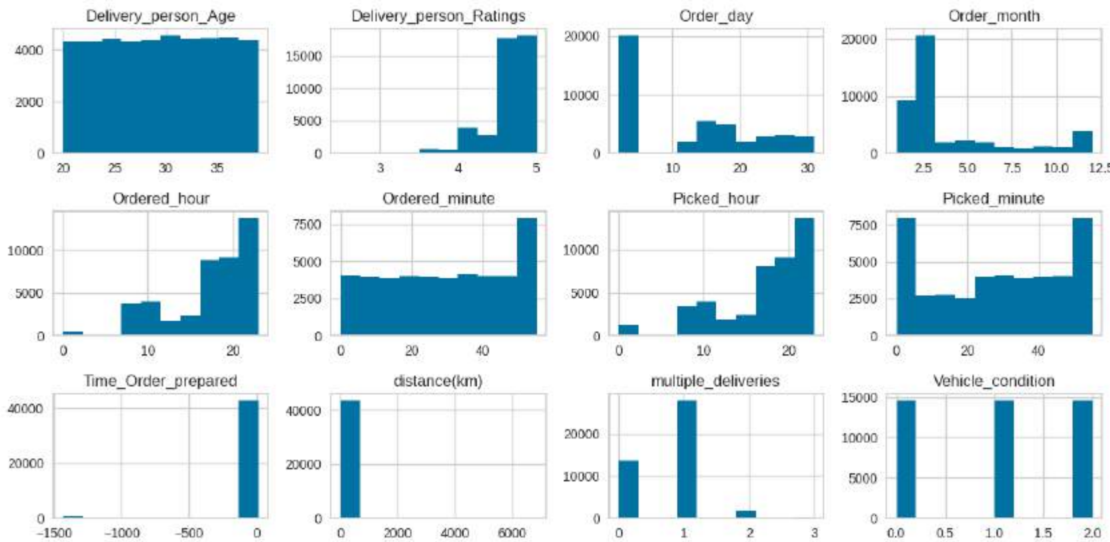
[20]:
```
columns_to_keep_without_target = [
    'Delivery_person_Age'
    , 'Delivery_person_Ratings'
    , 'Order_day'
    , 'Order_month'
    , 'Order_weekdate'
    , 'Ordered_hour'
    , 'Ordered_minute'
    , 'Picked_hour'
    , 'Picked_minute'
    , 'Time_Order_prepared'
    , 'distance(km)'
    , 'Type_of_order'
    , 'Type_of_vehicle'
    , 'multiple_deliveries'
    , 'City'
    , 'Festival'
    , 'Weatherconditions'
    , 'Road_traffic_density'
    , 'Vehicle_condition'
]
```

[21]:
```
target = 'Time_taken(min)'

numeric_features = [
    'Delivery_person_Age'
    , 'Delivery_person_Ratings'
    , 'Order_day'
    , 'Order_month'
    , 'Ordered_hour'
    , 'Ordered_minute'
    , 'Picked_hour'
    , 'Picked_minute'
    , 'Time_Order_prepared'
    , 'distance(km)'
    , 'multiple_deliveries'
    , 'Vehicle_condition'
]

categorical_features = list(df_train5.drop(columns=numeric_features + [target],␣
 ↪axis=1).columns)
```

4 Quick EDA (Exploratory Data Analysis)

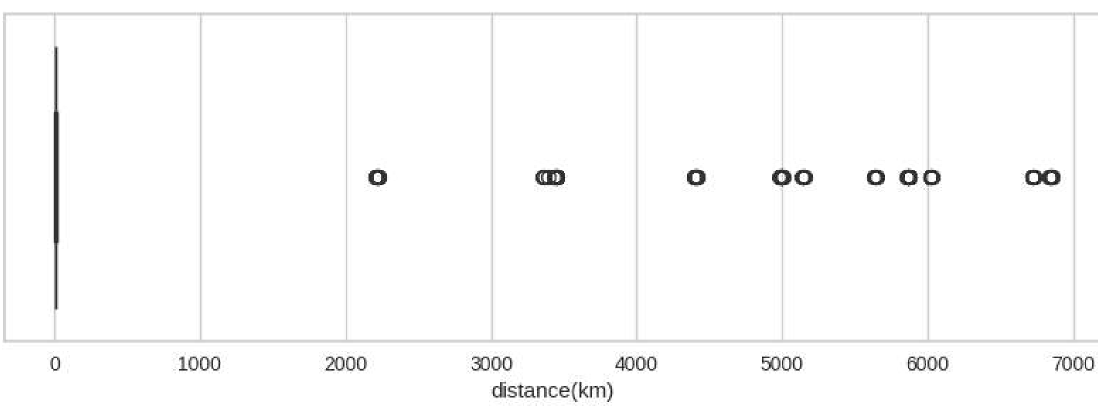
Plotting histogram of numeric variables

[22]:
```
df_train5[numeric_features].hist(layout=(3,4), figsize=(12,6))
plt.tight_layout()
```



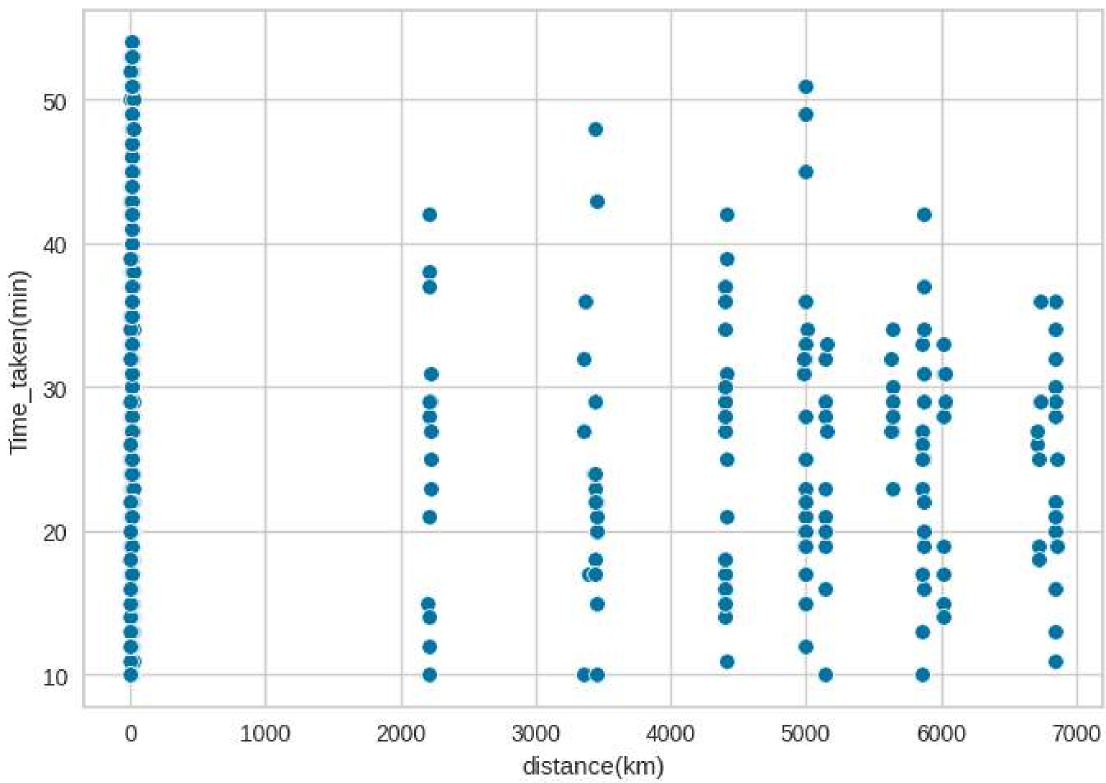
[23]:
```
plt.figure(figsize=(10,3))
sns.boxplot(x=df_train5['distance(km)'])
```

[23]: <Axes: xlabel='distance(km)'>



[24]:
```
sns.scatterplot(x='distance(km)', y='Time_taken(min)', data=df_train5)
```

[24]: <Axes: xlabel='distance(km)', ylabel='Time_taken(min)'>



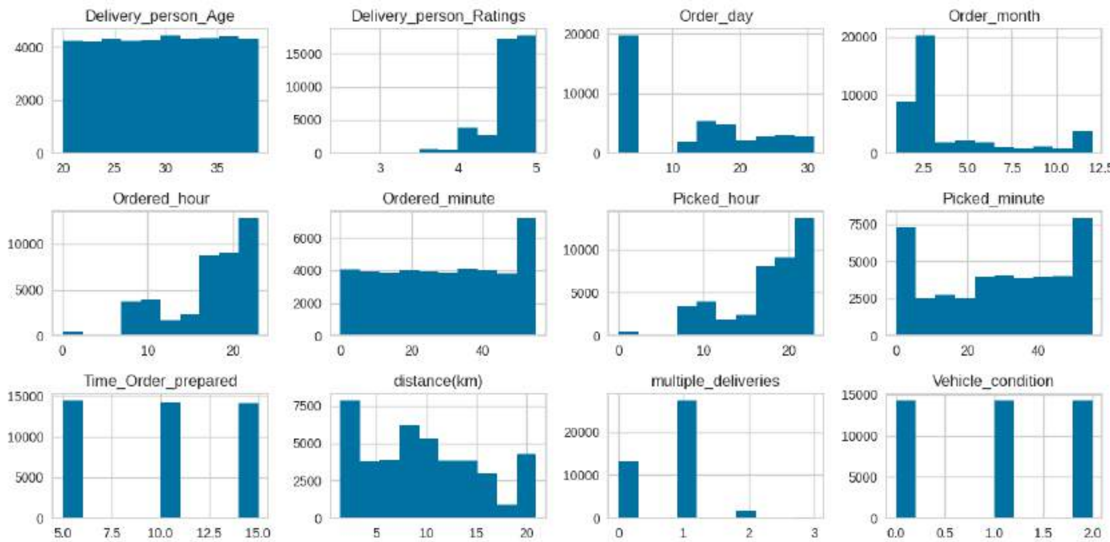
Delivering food over a distance of 2000 km in less than an hour using two-wheelers is an impractical and unrealistic proposition

There are some negative values in Time_Order_Prapared

Define a function to deal with ouliers

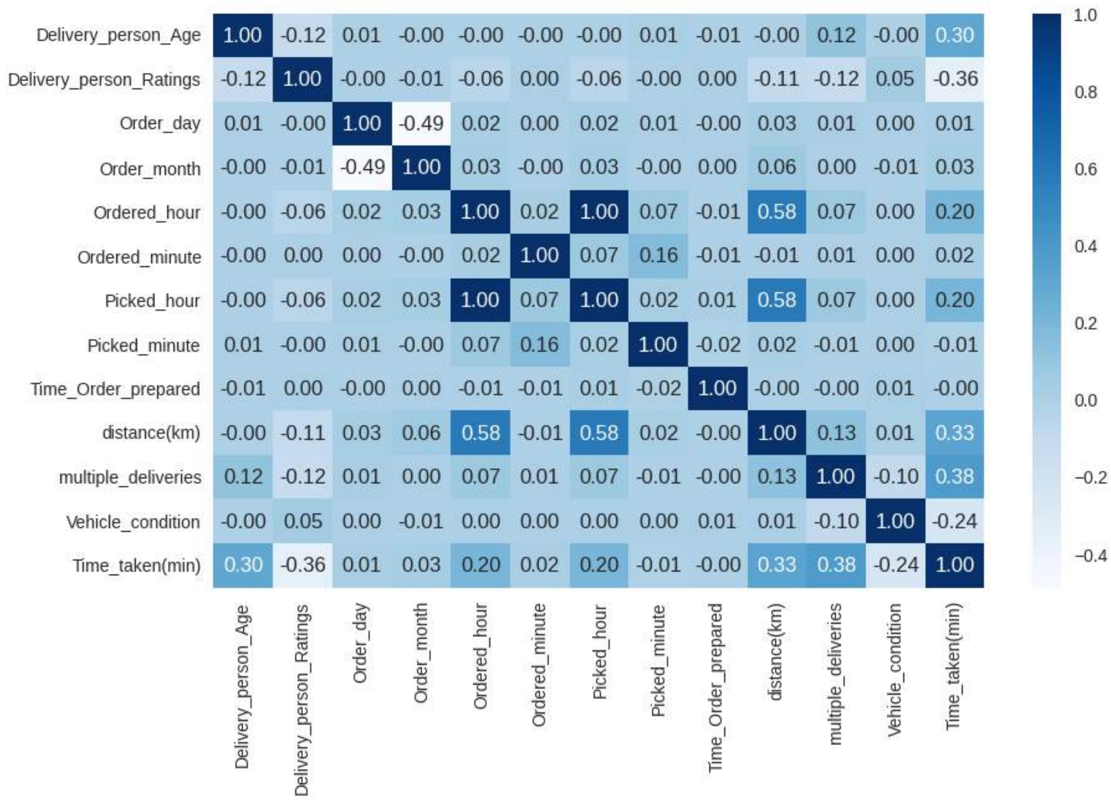
[25]:
```
def transform_outliers(data):
  data = data[(data['distance(km)'] < 1000)&(data['Time_Order_prepared'] > 0)]
  return data

df_train6 = transform_outliers(df_train5)
```

[26]:
```
df_train6[numeric_features].hist(layout=(3,4), figsize=(12,6))
plt.tight_layout()
```



Checking the correlation between numeric variables and target

[27]:
```
data_for_heatmap = df_train6[numeric_features + [target]]

correlation_matrix = data_for_heatmap.corr()

plt.figure(figsize=(10,6))
sns.heatmap(correlation_matrix, annot=True, cmap='Blues', fmt='.2f')
plt.show()
```



Order_day, Order_month, Ordered_minute, Picked_minute, Time_Order_prepared are likely to have no significant impact on target

Update columns

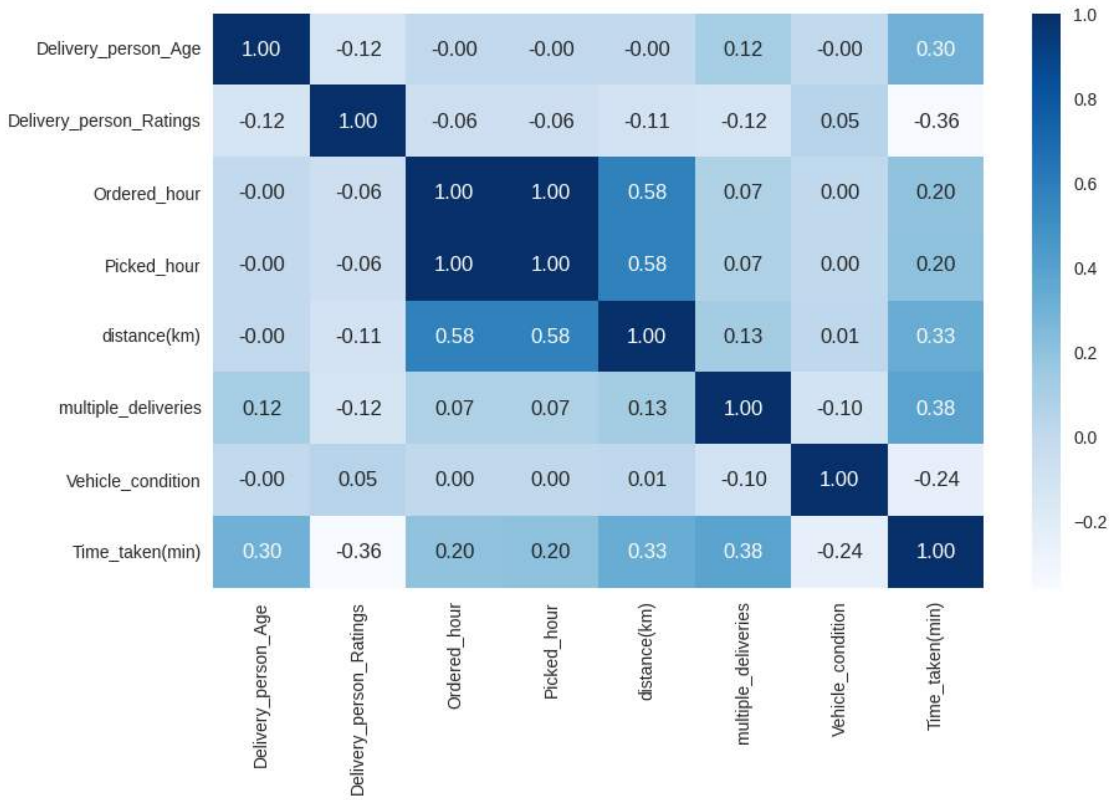
[28]:
```
numeric_features = [
    'Delivery_person_Age'
    , 'Delivery_person_Ratings'
    , 'Ordered_hour'
    , 'Picked_hour'
    , 'distance(km)'
    , 'multiple_deliveries'
    , 'Vehicle_condition'
]

df_train6 = df_train6.drop(columns=['Order_day', 'Order_month',␣
↪'Ordered_minute', 'Picked_minute', 'Time_Order_prepared'], axis=1)
```

[29]:
```
columns_to_keep_without_target = [
    'Delivery_person_Age'
    , 'Delivery_person_Ratings'
    , 'Order_day'
```

```
    , 'Order_month'
    , 'Order_weekdate'
    , 'Ordered_hour'
    , 'Ordered_minute'
    , 'Picked_hour'
    , 'Picked_minute'
    , 'Time_Order_prepared'
    , 'distance(km)'
    , 'Type_of_order'
    , 'Type_of_vehicle'
    , 'multiple_deliveries'
    , 'City'
    , 'Festival'
    , 'Weatherconditions'
    , 'Road_traffic_density'
    , 'Vehicle_condition'
]
```

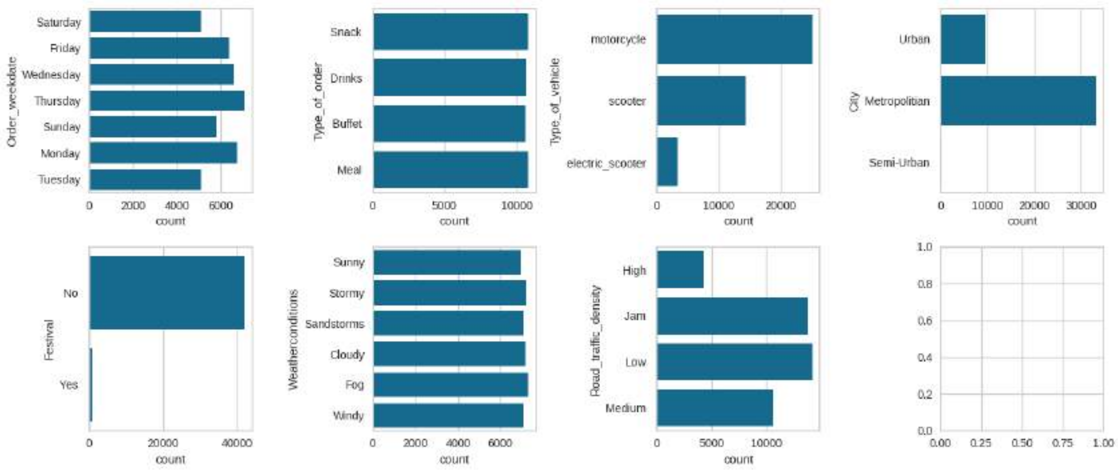
[30]:

```
data_for_heatmap = df_train6[numeric_features + [target]]

correlation_matrix = data_for_heatmap.corr()

plt.figure(figsize=(10,6))
sns.heatmap(correlation_matrix, annot=True, cmap='Blues', fmt='.2f')
plt.show()
```



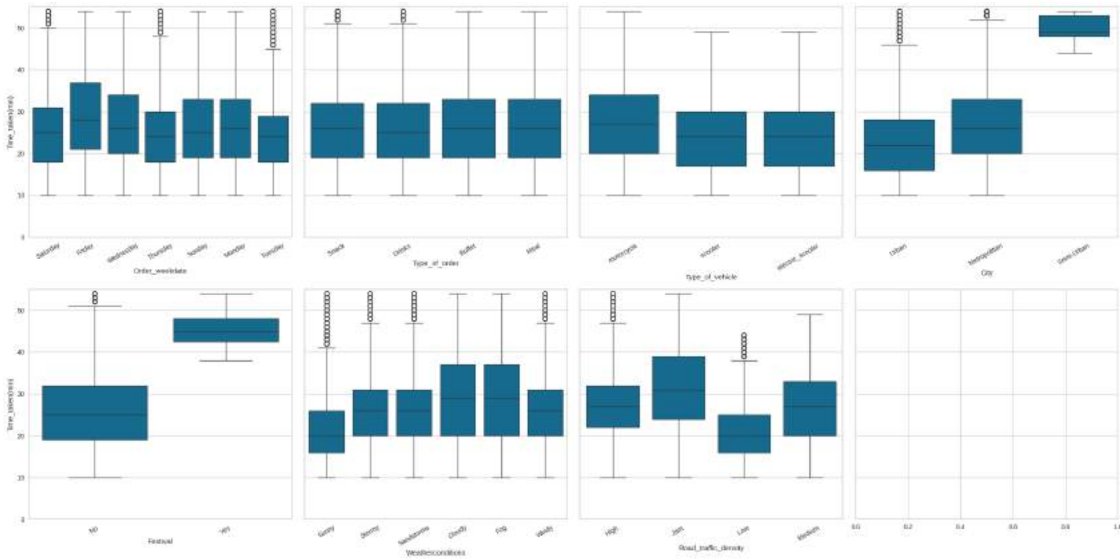
Plotting bar chart of categorical variables

[31]:
```
num_rows = (len(categorical_features) + 1) // 3

fig, axes = plt.subplots(num_rows, 4, figsize=(14, 3 * num_rows))

axes = axes.flatten()

for i, feature in enumerate(categorical_features):
    sns.countplot(data=df_train6, y=feature, ax=axes[i])

plt.tight_layout()
```



Plotting distribution of target across categorical variables

[32]:
```python
fig, ax = plt.subplots(2,4, figsize=(24,12), sharey=True)

row,col = 0,0

for feature in categorical_features:
    sns.boxplot(data=df_train6, x=feature, y=target, ax=ax[row,col])
    ax[row,col].set_ylim([0,55])
    xlabels = ax[row,col].get_xticklabels()
    ax[row,col].set_xticklabels(xlabels, rotation=30)

    if col < 3:
        col += 1
    else:
        row += 1
        col = 0

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



5 Work with Pycaret

[33]:
```
exp = RegressionExperiment()
type(exp)
```

[33]:
```
pycaret.regression.oop.RegressionExperiment
```

[34]:
```
# initiate setup on exp

exp.setup(df_train6, target=target, numeric_features=numeric_features,␣
↪categorical_features=categorical_features, session_id=123)
```

```
<pandas.io.formats.style.Styler at 0x7e9752774820>
```

[34]:
```
<pycaret.regression.oop.RegressionExperiment at 0x7e974ff57df0>
```

[35]:
```
# compare baseline models

best = exp.compare_models()
```

```
<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x7e975c643670>

Processing:   0%|          | 0/81 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[36]:
```
lightgbm_model = exp.create_model('lightgbm')
```

```
<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x7e9752e61d50>

Processing:   0%|          | 0/4 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[39]:
```
holdout_pred = exp.predict_model(lightgbm_model)
```

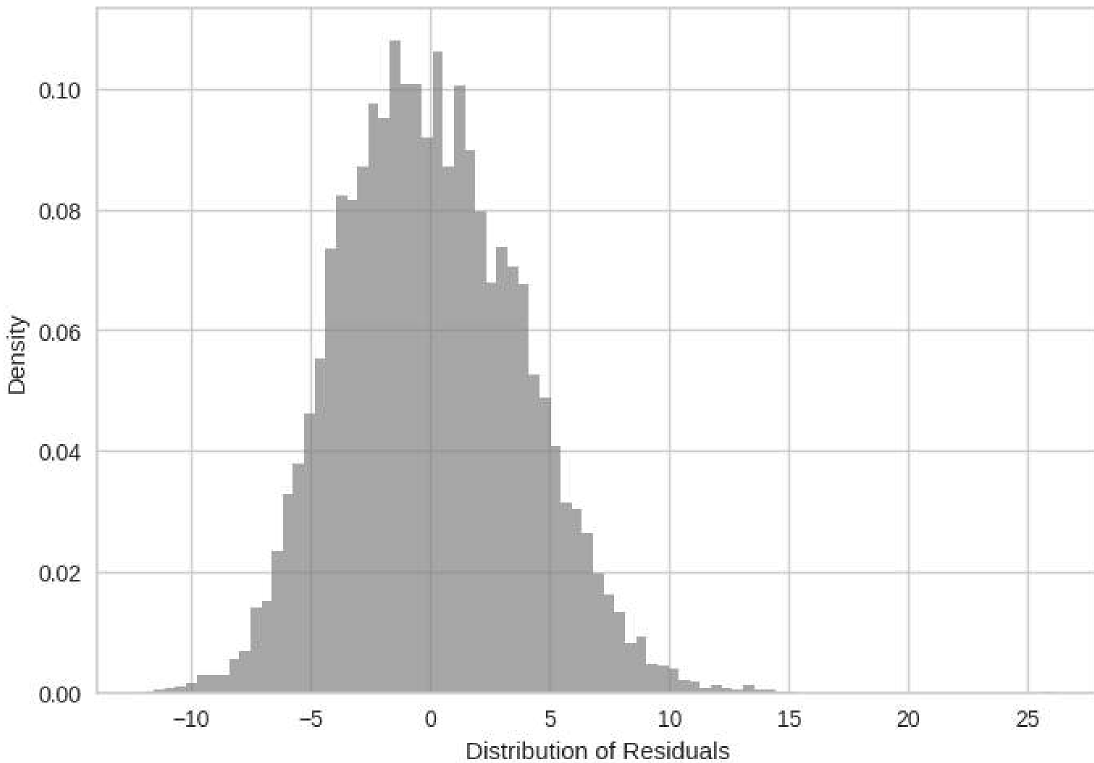
```
<pandas.io.formats.style.Styler at 0x7e975326e230>

<IPython.core.display.HTML object>
```

[40]:
```
holdout_pred.head()
```

[40]:
```
       Delivery_person_Age  Delivery_person_Ratings Order_weekdate  \
40829                 27.0                      5.0         Monday
40574                 37.0                      4.7       Thursday
31838                 32.0                      4.0         Friday
6782                  26.0                      4.8        Tuesday
35634                 39.0                      4.4       Saturday

       Ordered_hour  Picked_hour  distance(km) Type_of_order  \
40829            23           23      4.464402        Snack
40574            19           19      9.043193       Buffet
31838            18           18     16.577705         Meal
6782             17           17      7.747886         Meal
35634            17           18     12.438828        Snack

         Type_of_vehicle  multiple_deliveries           City Festival  \
40829            scooter                  0.0          Urban       No
40574            scooter                  1.0  Metropolitian       No
31838         motorcycle                  2.0  Metropolitian       No
6782             scooter                  1.0  Metropolitian       No
35634   electric_scooter                  0.0  Metropolitian       No

      Weatherconditions Road_traffic_density  Vehicle_condition  \
40829             Sunny                  Low                  1
40574            Stormy                  Jam                  2
31838               Fog               Medium                  1
6782              Windy               Medium                  2
35634        Sandstorms               Medium                  2

       Time_taken(min)  prediction_label
40829               23         17.062429
40574               25         29.111102
31838               39         39.191517
6782                26         23.972451
```

```
35634                  31          35.812686
```

[41]:
```
holdout_pred['residuals'] = holdout_pred['Time_taken(min)'] -␣
 ↪holdout_pred['prediction_label']

import matplotlib.pyplot as plt

plt.hist(holdout_pred['residuals'], bins='auto', density=True, color='grey',␣
 ↪alpha=0.7)
plt.xlabel('Distribution of Residuals')
plt.ylabel('Density')
plt.show()
```



Load and clean the predict data

[ ]:
```
# import predict data
files.upload()
```

[43]:
```
df_predict = pd.read_csv('predict.csv')
df_predict.head()
```

[43]:
```
        ID Delivery_person_ID Delivery_person_Age Delivery_person_Ratings  \
0  0x2318    COIMBRES13DEL01                 NaN                     NaN
1  0x3474     BANGRES15DEL01                  28                     4.6
2  0x9420      JAPRES09DEL03                  23                     4.5
3  0x72ee      JAPRES07DEL03                  21                     4.8
4  0xa759     CHENRES19DEL01                  31                     4.6

   Restaurant_latitude  Restaurant_longitude  Delivery_location_latitude  \
0            11.003669             76.976494                   11.043669
1            12.975377             77.696664                   13.085377
2            26.911378             75.789034                   27.001378
3            26.766536             75.837333                   26.856536
4            12.986047             80.218114                   13.096047

   Delivery_location_longitude  Order_Date Time_Orderd Time_Order_picked  \
0                    77.016494  30-03-2022         NaN          15:05:00
1                    77.806664  29-03-2022    20:30:00          20:35:00
2                    75.879034  10-03-2022    19:35:00          19:45:00
3                    75.927333  02-04-2022    17:15:00          17:20:00
4                    80.328114  27-03-2022    18:25:00          18:40:00

   Weatherconditions Road_traffic_density  Vehicle_condition Type_of_order  \
0     conditions NaN                  NaN                  3        Drinks
1   conditions Windy                  Jam                  0         Snack
2  conditions Stormy                  Jam                  0        Drinks
3     conditions Fog               Medium                  1          Meal
4   conditions Sunny               Medium                  2        Drinks

     Type_of_vehicle multiple_deliveries Festival            City
0  electric_scooter                    1       No    Metropolitian
1        motorcycle                    1       No    Metropolitian
2        motorcycle                    1       No    Metropolitian
3           scooter                    1       No    Metropolitian
4           scooter                    1       No    Metropolitian
```

[44]:
```python
def transform_outliers_new(data):
  data = data[data['distance(km)'] < 1000]
  return data
```

[45]:
```python
df_predict2 = transform_null(df_predict)
df_predict3 = transform_dataframe_without_target(df_predict2)
df_predict4 = transform_fill_null(df_predict3)
df_predict5 = df_predict4[columns_to_keep_without_target]
df_predict6 = transform_outliers(df_predict5)
df_predict6.head()
df_predict6 = df_predict6.drop(columns=['Order_day', 'Order_month',␣
↪'Ordered_minute', 'Picked_minute', 'Time_Order_prepared'], axis=1)
```

Make Prediction

[46]:
```
predictions_pycaret = exp.predict_model(lightgbm_model, data = df_predict6)
predictions_pycaret.head()
```

```
<IPython.core.display.HTML object>
```

[46]:
```
   Delivery_person_Age  Delivery_person_Ratings Order_weekdate  Ordered_hour  \
1                 28.0                      4.6        Tuesday            20
2                 23.0                      4.5         Monday            19
3                 21.0                      4.8         Friday            17
4                 31.0                      4.6         Sunday            18
5                 26.0                      4.7        Tuesday             9

   Picked_hour  distance(km) Type_of_order Type_of_vehicle  \
1           20     17.042984         Snack      motorcycle
2           19     13.390474        Drinks      motorcycle
3           17     13.397932          Meal         scooter
4           18     17.042633        Drinks         scooter
5            9      1.541060        Drinks      motorcycle

   multiple_deliveries            City Festival Weatherconditions  \
1                  1.0  Metropolitian       No             Windy
2                  1.0  Metropolitian       No            Stormy
3                  1.0  Metropolitian       No               Fog
4                  1.0  Metropolitian       No             Sunny
5                  1.0  Metropolitian       No               Fog

  Road_traffic_density  Vehicle_condition  prediction_label
1                  Jam                  0         30.658779
2                  Jam                  0         30.461126
3               Medium                  1         32.041284
4               Medium                  2         22.505894
5                  Low                  0         19.096164
```

Save the Pycaret Experiment pipeline

[47]:
```
# Save model (pipeline)
exp.save_model(best, 'time_delivery_pred_pipeline')
```

```
Transformation Pipeline and Model Successfully Saved
```

[47]:
```
(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(include=['Delivery_person_Age',
                                              'Delivery_person_Ratings',
                                              'Ordered_hour', 'Picked_hour',
                                              'distance(km)',
```

```
                                                    'multiple_deliveries',
                                                    'Vehicle_condition'],
                                        transformer=SimpleImputer())),
                    ('categorical_imputer',
                     TransformerWrapper(include=['Order_weekdate', 'Type_of_order',
                                                 'Type_o…
                                                 'Weatherconditions',
                                                 'Road_traffic_density'],
transformer=OneHotEncoder(cols=['Order_weekdate',
'Type_of_order',
'Type_of_vehicle',
                                                                          'City',
'Weatherconditions',
'Road_traffic_density'],
handle_missing='return_nan',
use_cat_names=True))),
                    ('clean_column_names',
                     TransformerWrapper(transformer=CleanColumnNames())),
                    ('trained_model', LGBMRegressor(n_jobs=-1,
random_state=123))]),
 'time_delivery_pred_pipeline.pkl')
```

[48]:
```
# Load pipeline
exp.load_model('time_delivery_pred_pipeline')
```

```
Transformation Pipeline and Model Successfully Loaded
```

[48]:
```
Pipeline(memory=FastMemory(location=/tmp/joblib),
          steps=[('numerical_imputer',
                  TransformerWrapper(include=['Delivery_person_Age',
                                               'Delivery_person_Ratings',
                                               'Ordered_hour', 'Picked_hour',
                                               'distance(km)',
                                               'multiple_deliveries',
                                               'Vehicle_condition'],
                                      transformer=SimpleImputer())),
                 ('categorical_imputer',
                  TransformerWrapper(include=['Order_weekdate', 'Type_of_ord…
                                               'Weatherconditions',
                                               'Road_traffic_density'],
transformer=OneHotEncoder(cols=['Order_weekdate',
'Type_of_order',
'Type_of_vehicle',
                                                                         'City',
'Weatherconditions',
'Road_traffic_density'],
handle_missing='return_nan',
```

```
use_cat_names=True))),
                ('clean_column_names',
                 TransformerWrapper(transformer=CleanColumnNames())),
                ('trained_model', LGBMRegressor(n_jobs=-1, random_state=123))])
```

6 Build a LightGBM Model

[49]:
```
df_train6.head()
```

[49]:
```
   Delivery_person_Age  Delivery_person_Ratings Order_weekdate  Ordered_hour  \
0                 37.0                      4.9       Saturday            11
1                 34.0                      4.5         Friday            19
2                 23.0                      4.4       Saturday             8
3                 38.0                      4.7      Wednesday            18
4                 32.0                      4.6       Saturday            13

   Picked_hour  distance(km) Type_of_order Type_of_vehicle  \
0           11      3.020737         Snack      motorcycle
1           19     20.143737         Snack         scooter
2            8      1.549693        Drinks      motorcycle
3           18      7.774497        Buffet      motorcycle
4           13      6.197898         Snack         scooter

   multiple_deliveries           City Festival Weatherconditions  \
0                  0.0          Urban       No             Sunny
1                  1.0  Metropolitian       No            Stormy
2                  1.0          Urban       No        Sandstorms
3                  1.0  Metropolitian       No             Sunny
4                  1.0  Metropolitian       No            Cloudy

  Road_traffic_density  Vehicle_condition  Time_taken(min)
0                 High                  2               24
1                  Jam                  2               33
2                  Low                  0               26
3               Medium                  0               21
4                 High                  1               30
```

1. Data Preprocessing

[50]:
```
df_train6_copy = df_train6.copy()

# scaling numeric features

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
```

```
df_train6_copy[numeric_features] = scaler.
 ↪fit_transform(df_train6_copy[numeric_features])

# encoding categorical features
df_train6_copy = pd.get_dummies(df_train6_copy, columns=categorical_features)

df_train6_copy.describe().T
```

[50]:

```
                                  count          mean       std        min  \
Delivery_person_Age             42877.0  1.218265e-15  1.000012  -1.661235
Delivery_person_Ratings         42877.0 -1.290517e-15  1.000012  -6.864116
Ordered_hour                    42877.0  3.562905e-17  1.000012  -3.606809
Picked_hour                     42877.0  1.915683e-16  1.000012  -3.641611
distance(km)                    42877.0  2.320031e-18  1.000012  -1.465951
multiple_deliveries             42877.0  2.883468e-17  1.000012  -1.321716
Vehicle_condition               42877.0 -7.440672e-17  1.000012  -1.226754
Time_taken(min)                 42877.0  2.637374e+01  9.391675  10.000000
Order_weekdate_Friday           42877.0  1.484479e-01  0.355548   0.000000
Order_weekdate_Monday           42877.0  1.573804e-01  0.364163   0.000000
Order_weekdate_Saturday         42877.0  1.196212e-01  0.324522   0.000000
Order_weekdate_Sunday           42877.0  1.360170e-01  0.342811   0.000000
Order_weekdate_Thursday         42877.0  1.651235e-01  0.371296   0.000000
Order_weekdate_Tuesday          42877.0  1.192248e-01  0.324057   0.000000
Order_weekdate_Wednesday        42877.0  1.541852e-01  0.361130   0.000000
Type_of_order_Buffet            42877.0  2.473587e-01  0.431482   0.000000
Type_of_order_Drinks            42877.0  2.492945e-01  0.432610   0.000000
Type_of_order_Meal              42877.0  2.510903e-01  0.433645   0.000000
Type_of_order_Snack             42877.0  2.522565e-01  0.434313   0.000000
Type_of_vehicle_electric_scooter 42877.0  8.050936e-02  0.272083   0.000000
Type_of_vehicle_motorcycle      42877.0  5.838795e-01  0.492920   0.000000
Type_of_vehicle_scooter         42877.0  3.356112e-01  0.472209   0.000000
City_Metropolitian              42877.0  7.743312e-01  0.418027   0.000000
City_Semi-Urban                 42877.0  3.638314e-03  0.060209   0.000000
City_Urban                      42877.0  2.220305e-01  0.415616   0.000000
Festival_No                     42877.0  9.800592e-01  0.139798   0.000000
Festival_Yes                    42877.0  1.994076e-02  0.139798   0.000000
Weatherconditions_Cloudy        42877.0  1.675024e-01  0.373428   0.000000
Weatherconditions_Fog           42877.0  1.704410e-01  0.376024   0.000000
Weatherconditions_Sandstorms    42877.0  1.656366e-01  0.371758   0.000000
Weatherconditions_Stormy        42877.0  1.684819e-01  0.374298   0.000000
Weatherconditions_Sunny         42877.0  1.620916e-01  0.368539   0.000000
Weatherconditions_Windy         42877.0  1.658465e-01  0.371947   0.000000
Road_traffic_density_High       42877.0  1.004035e-01  0.300541   0.000000
Road_traffic_density_Jam        42877.0  3.207547e-01  0.466772   0.000000
Road_traffic_density_Low        42877.0  3.305735e-01  0.470425   0.000000
Road_traffic_density_Medium     42877.0  2.482683e-01  0.432013   0.000000
```

	25%	50%	75%	max
Delivery_person_Age	-0.791928	0.077379	0.946686	1.642131
Delivery_person_Ratings	-0.435890	0.206933	0.849755	1.171167
Ordered_hour	-0.482310	0.350889	0.767489	1.184089
Picked_hour	-0.516963	0.316276	0.732896	1.149515
distance(km)	-0.897291	-0.088756	0.711540	2.011925
multiple_deliveries	-1.321716	0.437222	0.437222	3.955098
Vehicle_condition	-1.226754	-0.002142	1.222470	1.222470
Time_taken(min)	19.000000	26.000000	32.000000	54.000000
Order_weekdate_Friday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Monday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Saturday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Sunday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Thursday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Tuesday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Wednesday	0.000000	0.000000	0.000000	1.000000
Type_of_order_Buffet	0.000000	0.000000	0.000000	1.000000
Type_of_order_Drinks	0.000000	0.000000	0.000000	1.000000
Type_of_order_Meal	0.000000	0.000000	1.000000	1.000000
Type_of_order_Snack	0.000000	0.000000	1.000000	1.000000
Type_of_vehicle_electric_scooter	0.000000	0.000000	0.000000	1.000000
Type_of_vehicle_motorcycle	0.000000	1.000000	1.000000	1.000000
Type_of_vehicle_scooter	0.000000	0.000000	1.000000	1.000000
City_Metropolitian	1.000000	1.000000	1.000000	1.000000
City_Semi-Urban	0.000000	0.000000	0.000000	1.000000
City_Urban	0.000000	0.000000	0.000000	1.000000
Festival_No	1.000000	1.000000	1.000000	1.000000
Festival_Yes	0.000000	0.000000	0.000000	1.000000
Weatherconditions_Cloudy	0.000000	0.000000	0.000000	1.000000
Weatherconditions_Fog	0.000000	0.000000	0.000000	1.000000
Weatherconditions_Sandstorms	0.000000	0.000000	0.000000	1.000000
Weatherconditions_Stormy	0.000000	0.000000	0.000000	1.000000
Weatherconditions_Sunny	0.000000	0.000000	0.000000	1.000000
Weatherconditions_Windy	0.000000	0.000000	0.000000	1.000000
Road_traffic_density_High	0.000000	0.000000	0.000000	1.000000
Road_traffic_density_Jam	0.000000	0.000000	1.000000	1.000000
Road_traffic_density_Low	0.000000	0.000000	1.000000	1.000000
Road_traffic_density_Medium	0.000000	0.000000	0.000000	1.000000

2. Feature Engineering

[51]:
```
# Splitting the Data:

from sklearn.model_selection import train_test_split

X = df_train6_copy.drop('Time_taken(min)', axis=1)
y = df_train6_copy['Time_taken(min)']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

3. Training Model: LightGBM

[52]:
```
# Building and Training the LightGBM Model

X_train.columns = X_train.columns.str.replace(' ', '_')
X_test.columns = X_test.columns.str.replace(' ', '_')

# Create a LightGBM dataset
train_data = lgb.Dataset(X_train, label=y_train)

# Define model parameters
params = {
    'objective': 'regression',
    'metric': 'rmse',
    'boosting_type': 'gbdt',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': 0
}

# Train the model
model = lgb.train(params, train_data, num_boost_round=100)
```

[53]:
```
# Making Predictions

y_pred = model.predict(X_test)
```

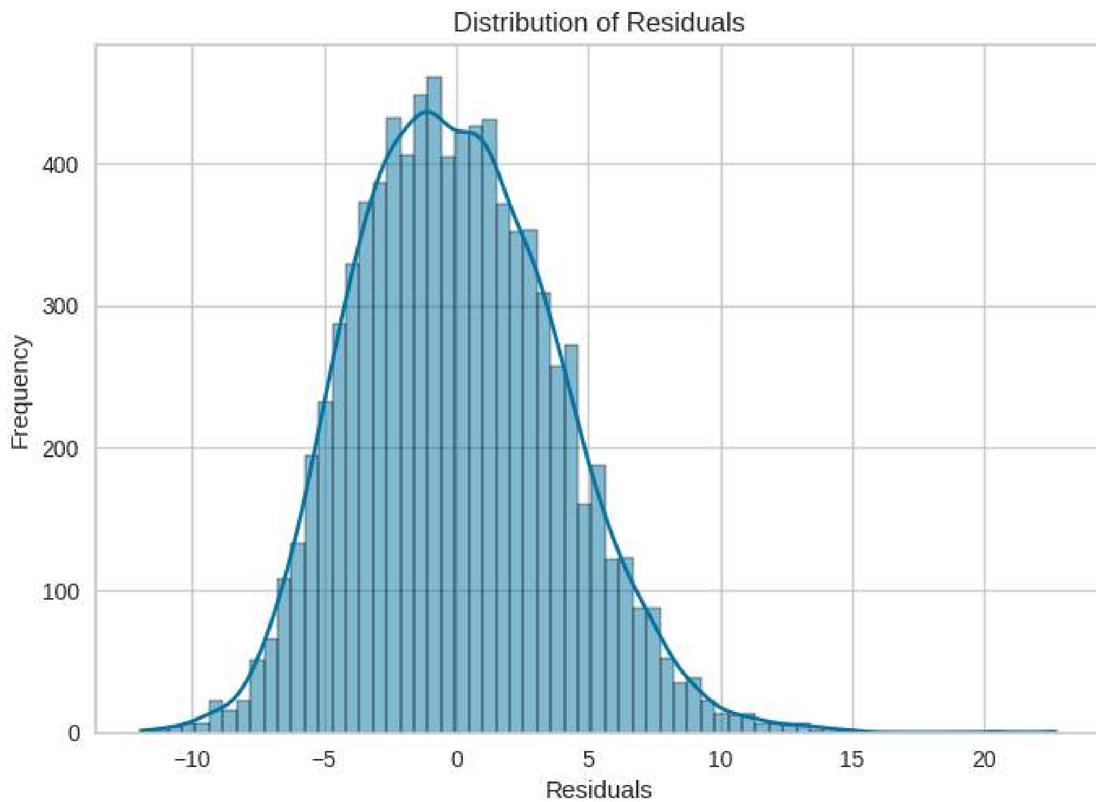
4. Evaluating Model Performance

[54]:
```
# Calculate MAE, MSE, RMSE, and R2
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f'Mean Absolute Error (MAE): {mae}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R-squared (R2): {r2}')
```

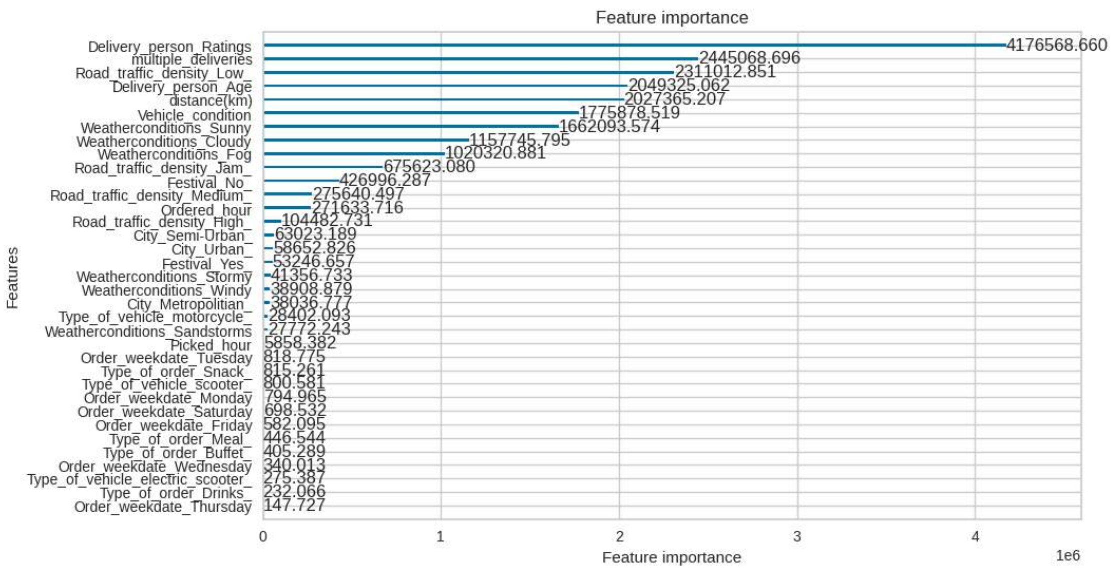
```
Mean Absolute Error (MAE): 3.0777658847022016
```

```
Mean Squared Error (MSE): 14.603112079363889
Root Mean Squared Error (RMSE): 3.8214018474067717
R-squared (R2): 0.8377692206485734
```

[55]:
```
residuals = y_test - y_pred
sns.histplot(residuals, kde=True)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Distribution of Residuals')
plt.show()
```



[59]:
```
lgb.plot_importance(model, importance_type='gain', figsize=(10,6))
plt.show()
```



5. Fine-Tuning Model

[ ]:
```
param_grid = {
    'num_leaves': [20, 31, 40],
    'learning_rate': [0.01, 0.05, 0.1],
}

# Create a LightGBM estimator (not a trained model)
base_model = lgb.LGBMRegressor()

# Create GridSearchCV with the LightGBM estimator
grid_search = GridSearchCV(estimator=base_model, param_grid=param_grid,␣
 ↪scoring='neg_mean_squared_error', cv=10)

# Fit the model
grid_search.fit(X_train, y_train)

# Get the best parameters and the best model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```

[62]:
```
# Print the best parameters
print("Best Parameters:", best_params)
```

```
Best Parameters: {'learning_rate': 0.1, 'num_leaves': 40}
```

[63]:
```
# Making Predictions
```

```
best_y_pred = best_model.predict(X_test)
```

[64]:
```
# Evaluating Model Performance

# Calculate MAE, MSE, RMSE, and R2
mae = mean_absolute_error(y_test, best_y_pred)
mse = mean_squared_error(y_test, best_y_pred)
rmse = mean_squared_error(y_test, best_y_pred, squared=False)
r2 = r2_score(y_test, best_y_pred)

print(f'Mean Absolute Error (MAE): {mae}')
print(f'Mean Squared Error (MSE): {mse}')
print(f'Root Mean Squared Error (RMSE): {rmse}')
print(f'R-squared (R2): {r2}')
```

```
Mean Absolute Error (MAE): 3.0055422432045207
Mean Squared Error (MSE): 13.895239521062162
Root Mean Squared Error (RMSE): 3.7276318918399336
R-squared (R2): 0.8456332099263835
```

6. Feature Engineering (Predict Data)

[65]:
```
# Preprocess predict data

df_predict6_copy = df_predict6.copy()

# scaling numeric features

scaler = StandardScaler()
df_predict6_copy[numeric_features] = scaler.
↪fit_transform(df_predict6_copy[numeric_features])

# encoding categorical features
df_predict6_copy = pd.get_dummies(df_predict6_copy,␣
↪columns=categorical_features)

df_predict6_copy.describe().T
```

[65]:
```
                                count          mean       std       min  \
Delivery_person_Age           10716.0  8.641422e-16  1.000047 -1.656649
Delivery_person_Ratings       10716.0  1.664298e-16  1.000047 -6.638265
Ordered_hour                  10716.0  1.644406e-16  1.000047 -3.598931
Picked_hour                   10716.0 -2.738467e-16  1.000047 -3.635631
distance(km)                  10716.0 -3.514256e-17  1.000047 -1.469664
multiple_deliveries           10716.0  1.292981e-17  1.000047 -1.326532
Vehicle_condition             10716.0 -7.293738e-17  1.000047 -1.233224
Order_weekdate_Friday         10716.0  1.432437e-01  0.350338  0.000000
Order_weekdate_Monday         10716.0  1.645203e-01  0.370764  0.000000
```

Order_weekdate_Saturday	10716.0	1.199141e-01	0.324876	0.000000
Order_weekdate_Sunday	10716.0	1.330720e-01	0.339668	0.000000
Order_weekdate_Thursday	10716.0	1.648936e-01	0.371102	0.000000
Order_weekdate_Tuesday	10716.0	1.173946e-01	0.321905	0.000000
Order_weekdate_Wednesday	10716.0	1.569616e-01	0.363781	0.000000
Type_of_order_Buffet	10716.0	2.538261e-01	0.435220	0.000000
Type_of_order_Drinks	10716.0	2.547592e-01	0.435746	0.000000
Type_of_order_Meal	10716.0	2.443076e-01	0.429696	0.000000
Type_of_order_Snack	10716.0	2.471071e-01	0.431350	0.000000
Type_of_vehicle_electric_scooter	10716.0	7.857409e-02	0.269085	0.000000
Type_of_vehicle_motorcycle	10716.0	5.863195e-01	0.492516	0.000000
Type_of_vehicle_scooter	10716.0	3.351064e-01	0.472050	0.000000
City_Metropolitian	10716.0	7.740761e-01	0.418209	0.000000
City_Semi-Urban	10716.0	4.199328e-03	0.064669	0.000000
City_Urban	10716.0	2.217245e-01	0.415426	0.000000
Festival_No	10716.0	9.813363e-01	0.135341	0.000000
Festival_Yes	10716.0	1.866368e-02	0.135341	0.000000
Weatherconditions_Cloudy	10716.0	1.650803e-01	0.371270	0.000000
Weatherconditions_Fog	10716.0	1.592945e-01	0.365968	0.000000
Weatherconditions_Sandstorms	10716.0	1.671333e-01	0.373112	0.000000
Weatherconditions_Stormy	10716.0	1.606010e-01	0.367180	0.000000
Weatherconditions_Sunny	10716.0	1.756252e-01	0.380519	0.000000
Weatherconditions_Windy	10716.0	1.722658e-01	0.377629	0.000000
Road_traffic_density_High	10716.0	9.994401e-02	0.299939	0.000000
Road_traffic_density_Jam	10716.0	3.178425e-01	0.465660	0.000000
Road_traffic_density_Low	10716.0	3.318402e-01	0.470896	0.000000
Road_traffic_density_Medium	10716.0	2.503733e-01	0.433248	0.000000

	25%	50%	75%	max
Delivery_person_Age	-0.785009	0.001285	0.783945	1.655586
Delivery_person_Ratings	-0.419695	0.202162	0.824019	1.134947
Ordered_hour	-0.485561	0.344671	0.759787	1.174903
Picked_hour	-0.520107	0.310700	0.726103	1.141506
distance(km)	-0.899874	-0.089761	0.703485	2.015117
multiple_deliveries	-1.326532	0.428421	0.428421	3.938326
Vehicle_condition	-1.233224	-0.009705	1.213814	1.213814
Order_weekdate_Friday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Monday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Saturday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Sunday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Thursday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Tuesday	0.000000	0.000000	0.000000	1.000000
Order_weekdate_Wednesday	0.000000	0.000000	0.000000	1.000000
Type_of_order_Buffet	0.000000	0.000000	1.000000	1.000000
Type_of_order_Drinks	0.000000	0.000000	1.000000	1.000000
Type_of_order_Meal	0.000000	0.000000	0.000000	1.000000
Type_of_order_Snack	0.000000	0.000000	0.000000	1.000000

```
Type_of_vehicle_electric_scooter   0.000000  0.000000  0.000000  1.000000
Type_of_vehicle_motorcycle         0.000000  1.000000  1.000000  1.000000
Type_of_vehicle_scooter            0.000000  0.000000  1.000000  1.000000
City_Metropolitian                 1.000000  1.000000  1.000000  1.000000
City_Semi-Urban                    0.000000  0.000000  0.000000  1.000000
City_Urban                         0.000000  0.000000  0.000000  1.000000
Festival_No                        1.000000  1.000000  1.000000  1.000000
Festival_Yes                       0.000000  0.000000  0.000000  1.000000
Weatherconditions_Cloudy           0.000000  0.000000  0.000000  1.000000
Weatherconditions_Fog              0.000000  0.000000  0.000000  1.000000
Weatherconditions_Sandstorms       0.000000  0.000000  0.000000  1.000000
Weatherconditions_Stormy           0.000000  0.000000  0.000000  1.000000
Weatherconditions_Sunny            0.000000  0.000000  0.000000  1.000000
Weatherconditions_Windy            0.000000  0.000000  0.000000  1.000000
Road_traffic_density_High          0.000000  0.000000  0.000000  1.000000
Road_traffic_density_Jam           0.000000  0.000000  1.000000  1.000000
Road_traffic_density_Low           0.000000  0.000000  1.000000  1.000000
Road_traffic_density_Medium        0.000000  0.000000  1.000000  1.000000
```

7. Making Prediction

[66]:
```python
prediction_normal = best_model.predict(df_predict6_copy)

prediction_normal
```

[66]:
```
array([30.43646533, 29.778379  , 30.93426622, …, 29.03624186,
       26.72663756, 23.49186358])
```

8. Save Model

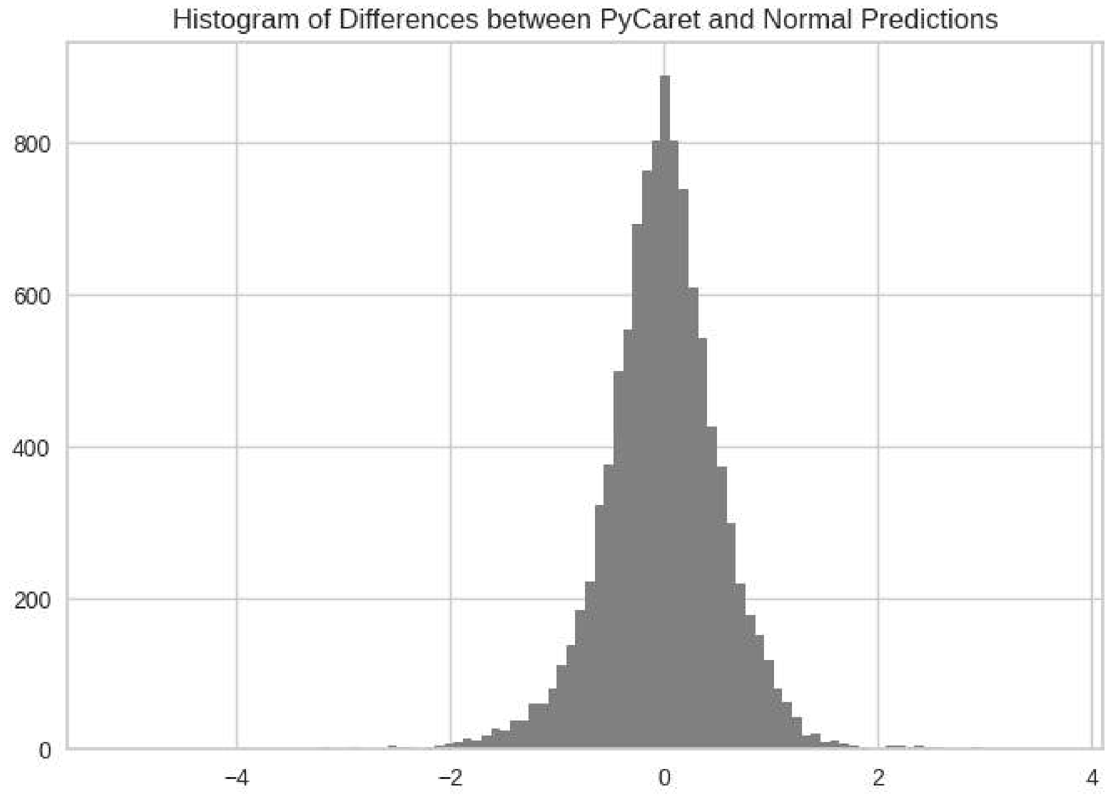
[67]:
```python
# save model
import joblib

# Save the best_model
joblib.dump(best_model, 'best_model.joblib')

# Save the best_params dictionary for reference
joblib.dump(grid_search.best_params_, 'best_params.joblib')
```

[67]:
```
['best_params.joblib']
```

[70]:
```python
# Load the saved model
loaded_model = joblib.load('best_model.joblib')

# Load the best_params dictionary
loaded_best_params = joblib.load('best_params.joblib')
```

7 Compare 2 Approaches: Pycaret and Model Building

[68]:
```
compare_result_pycaret_normal = predictions_pycaret['prediction_label'] -␣
↪prediction_normal

plt.hist(compare_result_pycaret_normal, bins=100, color='grey')
plt.title('Histogram of Differences between PyCaret and Normal Predictions')
plt.show()
```



Export to a PDF file

[ ]:
```
!pip install nbconvert
!apt-get install texlive-xetex
```

[ ]:
```
from google.colab import drive
import nbformat
from nbconvert import PDFExporter

# Mount Google Drive
drive.mount('/content/drive')

# Get the notebook name
```

```python
notebook_name = 'Delivery_Time_Prediction.ipynb'

# Load the notebook
notebook_path = f'/content/drive/My Drive/Colab Notebooks/{notebook_name}'
with open(notebook_path) as f:
    notebook = nbformat.read(f, as_version=4)

# Configure PDF export
pdf_exporter = PDFExporter()
pdf_data, resources = pdf_exporter.from_notebook_node(notebook)

# Save PDF to Google Drive
pdf_path = f'/content/drive/My Drive/Colab Notebooks/{notebook_name.replace(".
↪ipynb", ".pdf")}'
with open(pdf_path, 'wb') as f:
    f.write(pdf_data)

print(f'PDF saved to: {pdf_path}')
```