Notebook

February 3, 2024

1 FOOD DELIVERY TIME PREDICTION

1. Data Cleaning

[ ]:
```python
import pandas as pd
from google.colab import files
files.upload()
```

[4]:
```python
df_train = pd.read_csv('train.csv')
df_train.head()
```

[4]:
```
        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
```

Checking basic information of dataset

[5]:
```
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]:
```
df_train.duplicated().sum()
```

[6]: 0

[7]:
```
df_train.isnull().sum()
```

[7]:
```
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
```

[8]:
```
unique_values = {}
for column in df_train.columns:
    unique_values[column] = df_train[column].value_counts().index

# Print unique values
for column, values in unique_values.items():
    print(f"Unique values of {column}:", values)
```

```
Unique values of ID: Index(['0x4607 ', '0x1f3e ', '0xe251 ', '0x3f31 ', '0x4a78
', '0xa16d ',
       '0xa561 ', '0xa9a5 ', '0x415e ', '0x9b31 ',
       …
       '0x8b0 ', '0x8537 ', '0x47df ', '0x2947 ', '0x4102 ', '0xc3f1 ',
       '0x5db7 ', '0x1985 ', '0xceda ', '0x5fb2 '],
      dtype='object', length=45593)
Unique values of Delivery_person_ID: Index(['PUNERES01DEL01 ', 'JAPRES11DEL02 ',
'HYDRES04DEL02 ', 'JAPRES03DEL01 ',
       'VADRES11DEL02 ', 'RANCHIRES02DEL01 ', 'VADRES08DEL02 ',
       'INDORES15DEL01 ', 'RANCHIRES02DEL02 ', 'BANGRES07DEL02 ',
```

```
       …
       'DEHRES12DEL03 ', 'BHPRES11DEL03 ', 'BHPRES15DEL03 ', 'AURGRES13DEL03 ',
       'GOARES01DEL03 ', 'DEHRES18DEL03 ', 'AURGRES11DEL03 ', 'KOLRES09DEL03 ',
       'KOCRES16DEL03 ', 'BHPRES010DEL03 '],
      dtype='object', length=1320)
Unique values of Delivery_person_Age: Index(['35', '36', '37', '30', '38', '24',
'32', '22', '29', '33', '28', '25',
       '34', '26', '21', '27', '39', '20', '31', '23', 'NaN ', '50', '15'],
      dtype='object')
Unique values of Delivery_person_Ratings: Index(['4.8', '4.7', '4.9', '4.6',
'5', '4.5', 'NaN ', '4.1', '4.2', '4.3',
       '4.4', '4', '3.5', '3.8', '3.7', '3.6', '3.9', '6', '1', '3.4', '3.1',
       '3.2', '3.3', '2.6', '2.7', '2.5', '2.8', '2.9', '3'],
      dtype='object')
Unique values of Restaurant_latitude: Float64Index([       0.0,  26.911378,
26.914142,  26.892312,   26.90294,
               26.902908,   26.88842,  26.905287,  26.913726,  22.308096,
              …
              -12.933284, -30.359722, -19.875016, -22.569358, -12.284747,
              -23.355164,  -15.51315, -22.311358, -27.161661, -12.978453],
             dtype='float64', length=657)
Unique values of Restaurant_longitude: Float64Index([       0.0,  75.789034,
75.805704,  75.793007,  75.806896,
               75.792934,   75.75282,  75.800689,  75.794592,  73.167753,
              …
              -72.972281, -73.164798, -76.625861, -78.379347, -77.615428,
              -76.626167, -85.316842, -76.643622, -72.814492, -77.643685],
             dtype='float64', length=518)
Unique values of Delivery_location_latitude: Float64Index([     0.13,      0.02,
0.09,      0.06,      0.07,      0.04,
                   0.05,      0.11,      0.01,      0.08,
              …
              10.029186, 30.376994, 30.456994, 30.482509,   22.5991, 19.976969,
              19.916219, 26.562001, 23.324249, 20.005337],
             dtype='float64', length=4373)
Unique values of Delivery_location_longitude: Float64Index([     0.13,
0.02,      0.09,      0.06,      0.07,      0.04,
                   0.05,      0.11,      0.01,      0.08,
              …
              76.367361, 78.092543, 78.172543, 78.201187, 88.450467, 75.428894,
              75.386017, 80.444002, 77.524007, 75.446722],
             dtype='float64', length=4373)
Unique values of Order_Date: Index(['15-03-2022', '03-04-2022', '13-03-2022',
'26-03-2022', '24-03-2022',
       '09-03-2022', '05-04-2022', '05-03-2022', '07-03-2022', '03-03-2022',
       '19-03-2022', '21-03-2022', '11-03-2022', '30-03-2022', '01-03-2022',
       '28-03-2022', '17-03-2022', '01-04-2022', '02-03-2022', '10-03-2022',
       '16-03-2022', '20-03-2022', '02-04-2022', '06-03-2022', '04-03-2022',
```

```
       '29-03-2022', '25-03-2022', '14-03-2022', '11-02-2022', '18-03-2022',
       '31-03-2022', '27-03-2022', '12-03-2022', '08-03-2022', '23-03-2022',
       '06-04-2022', '13-02-2022', '15-02-2022', '04-04-2022', '17-02-2022',
       '12-02-2022', '16-02-2022', '18-02-2022', '14-02-2022'],
      dtype='object')
Unique values of Time_Orderd: Index(['NaN ', '21:55:00', '17:55:00', '20:00:00',
'22:20:00', '21:35:00',
       '19:50:00', '21:15:00', '22:45:00', '21:20:00',
       …
       '16:10:00', '13:25:00', '12:15:00', '16:15:00', '14:30:00', '12:25:00',
       '14:15:00', '16:00:00', '13:20:00', '16:30:00'],
      dtype='object', length=177)
Unique values of Time_Order_picked: Index(['21:30:00', '22:50:00', '22:40:00',
'18:40:00', '17:55:00', '21:45:00',
       '22:25:00', '18:05:00', '20:50:00', '23:50:00',
       …
       '14:15:00', '13:15:00', '13:10:00', '08:15:00', '14:20:00', '15:10:00',
       '16:15:00', '16:10:00', '17:10:00', '16:20:00'],
      dtype='object', length=193)
Unique values of Weatherconditions: Index(['conditions Fog', 'conditions
Stormy', 'conditions Cloudy',
       'conditions Sandstorms', 'conditions Windy', 'conditions Sunny',
       'conditions NaN'],
      dtype='object')
Unique values of Road_traffic_density: Index(['Low ', 'Jam ', 'Medium ', 'High
', 'NaN '], dtype='object')
Unique values of Vehicle_condition: Int64Index([2, 1, 0, 3], dtype='int64')
Unique values of Type_of_order: Index(['Snack ', 'Meal ', 'Drinks ', 'Buffet '],
dtype='object')
Unique values of Type_of_vehicle: Index(['motorcycle ', 'scooter ',
'electric_scooter ', 'bicycle '], dtype='object')
Unique values of multiple_deliveries: Index(['1', '0', '2', 'NaN ', '3'],
dtype='object')
Unique values of Festival: Index(['No ', 'Yes ', 'NaN '], dtype='object')
Unique values of City: Index(['Metropolitian ', 'Urban ', 'NaN ', 'Semi-Urban
'], dtype='object')
Unique values of Time_taken(min): Index(['(min) 26', '(min) 25', '(min) 27',
'(min) 28', '(min) 29', '(min) 19',
       '(min) 15', '(min) 18', '(min) 16', '(min) 17', '(min) 24', '(min) 23',
       '(min) 20', '(min) 22', '(min) 21', '(min) 33', '(min) 30', '(min) 31',
       '(min) 34', '(min) 32', '(min) 38', '(min) 36', '(min) 39', '(min) 35',
       '(min) 37', '(min) 11', '(min) 10', '(min) 12', '(min) 14', '(min) 13',
       '(min) 43', '(min) 42', '(min) 40', '(min) 41', '(min) 44', '(min) 47',
       '(min) 49', '(min) 48', '(min) 46', '(min) 45', '(min) 53', '(min) 51',
       '(min) 54', '(min) 52', '(min) 50'],
      dtype='object')
```

Some null values are identified: ‘NaN’ as a string

[9]:
```
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)
```

[10]:
```
df_train2.isna().sum()
```

[10]:
```
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 transforming data

[11]:
```
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
```

```python
    # 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()

    return data

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

[11]:

```
        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 for transforming the predict data (without the target - Time_taken(min))

[12]:
```python
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
```

[13]:
```
df_train3.isna().sum()
```

[13]:
```
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
```

[14]:

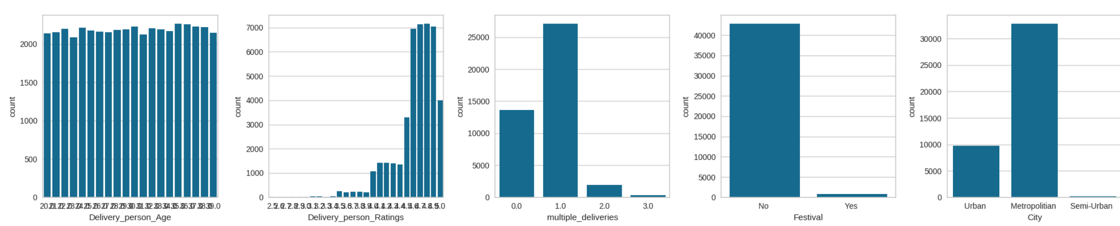
```
df_train3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43862 entries, 0 to 45592
Data columns (total 32 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   ID                           43862 non-null  object
 1   Delivery_person_ID           43862 non-null  object
 2   Delivery_person_Age          43648 non-null  float64
 3   Delivery_person_Ratings      43594 non-null  float64
 4   Restaurant_latitude          43862 non-null  float64
 5   Restaurant_longitude         43862 non-null  float64
 6   Delivery_location_latitude   43862 non-null  float64
 7   Delivery_location_longitude  43862 non-null  float64
 8   Order_Date                   43862 non-null  object
 9   Time_Ordered                 43862 non-null  object
 10  Time_Order_picked            43862 non-null  object
 11  Weatherconditions            43862 non-null  object
 12  Road_traffic_density         43862 non-null  object
 13  Vehicle_condition            43862 non-null  int64
 14  Type_of_order                43862 non-null  object
 15  Type_of_vehicle              43862 non-null  object
 16  multiple_deliveries          42919 non-null  float64
 17  Festival                     43643 non-null  object
 18  City                         42718 non-null  object
 19  Time_taken(min)              43862 non-null  int64
 20  distance(km)                 43862 non-null  float64
 21  Pick_date                    43862 non-null  object
 22  Datetime_Ordered             43862 non-null  datetime64[ns]
 23  Datetime_Picked              43862 non-null  datetime64[ns]
```

```
 24  Time_Order_prepared          43862 non-null  float64
 25  Ordered_hour                 43862 non-null  int64
 26  Ordered_minute               43862 non-null  int64
 27  Picked_hour                  43862 non-null  int64
 28  Picked_minute                43862 non-null  int64
 29  Order_day                    43862 non-null  int64
 30  Order_month                  43862 non-null  int64
 31  Order_weekdate               43862 non-null  object
dtypes: datetime64[ns](2), float64(9), int64(8), object(13)
memory usage: 11.0+ MB
```

Filling null values

[15]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set up the matplotlib figure with subplots
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()
```



[16]:
```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()
```

[16]:
```
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
```

[35]:
```
columns_to_keep = [
    'Delivery_person_Age'
    , 'Delivery_person_Ratings'
    , 'Order_weekdate'
```

```
    , 'Ordered_hour'
    , 'Picked_hour'
    , '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()
```

[35]:

```
   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
```

[27]:

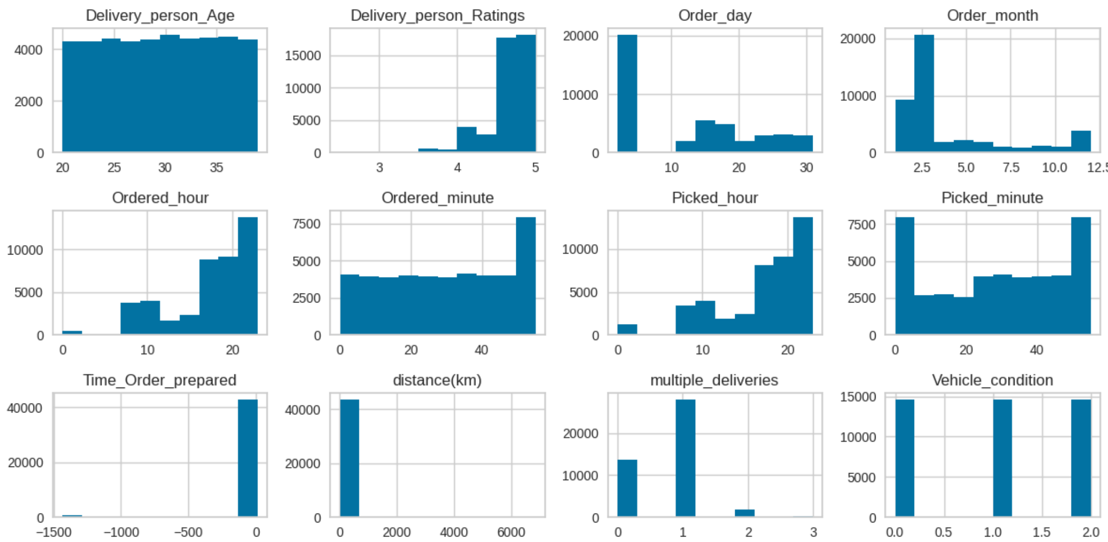
```
columns_to_keep_without_target = [
    'Delivery_person_Age'
```

```
    , 'Delivery_person_Ratings'
    , 'Order_weekdate'
    , 'Ordered_hour'
    , 'Picked_hour'
    , 'distance(km)'
    , 'Type_of_order'
    , 'Type_of_vehicle'
    , 'multiple_deliveries'
    , 'City'
    , 'Festival'
    , 'Weatherconditions'
    , 'Road_traffic_density'
    , 'Vehicle_condition'
]
```

[36]:
```
target = 'Time_taken(min)'

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

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

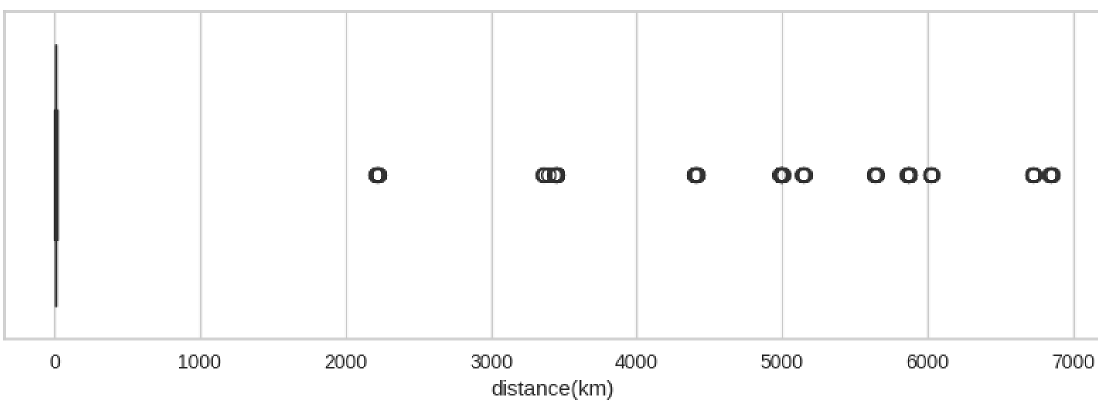
2. Exploratory Data Analysis

[21]:
```
df_train5[numeric_features].hist(layout=(3,4), figsize=(12,6))
plt.tight_layout()
```



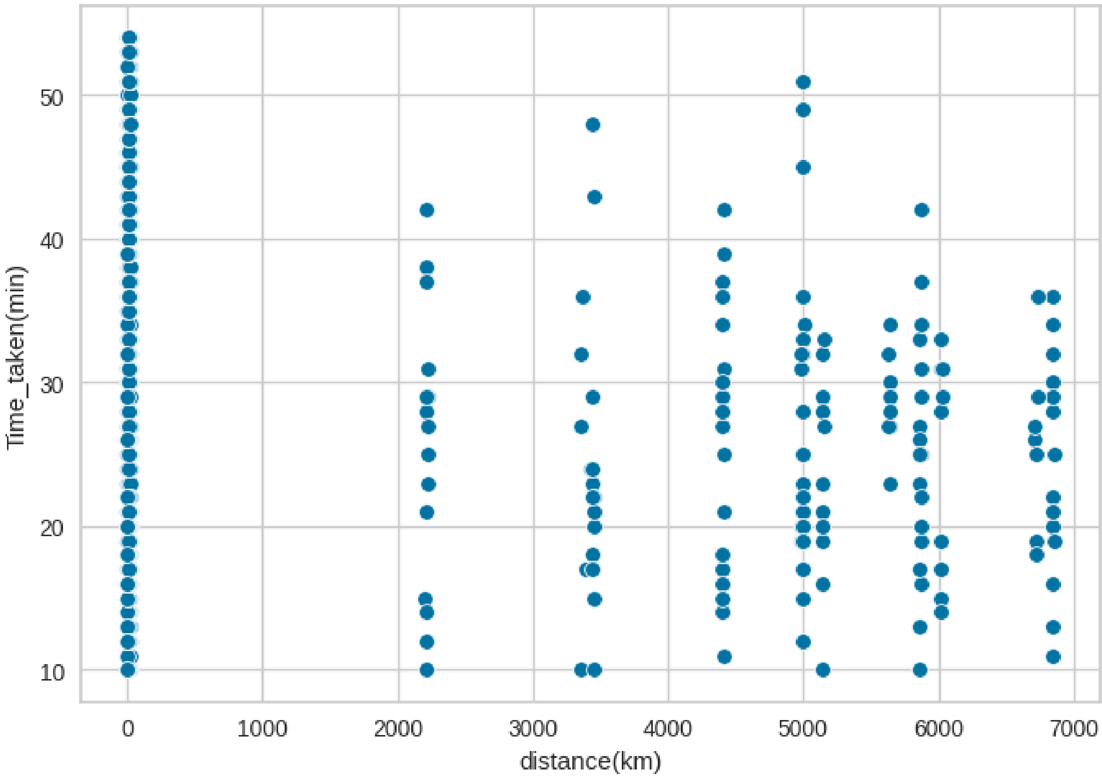
[22]:
```
plt.figure(figsize=(10,3))
sns.boxplot(x=df_train5['distance(km)'])
```

[22]: <AxesSubplot: xlabel='distance(km)'>



[23]:
```
sns.scatterplot(x='distance(km)', y='Time_taken(min)', data=df_train5)
```

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

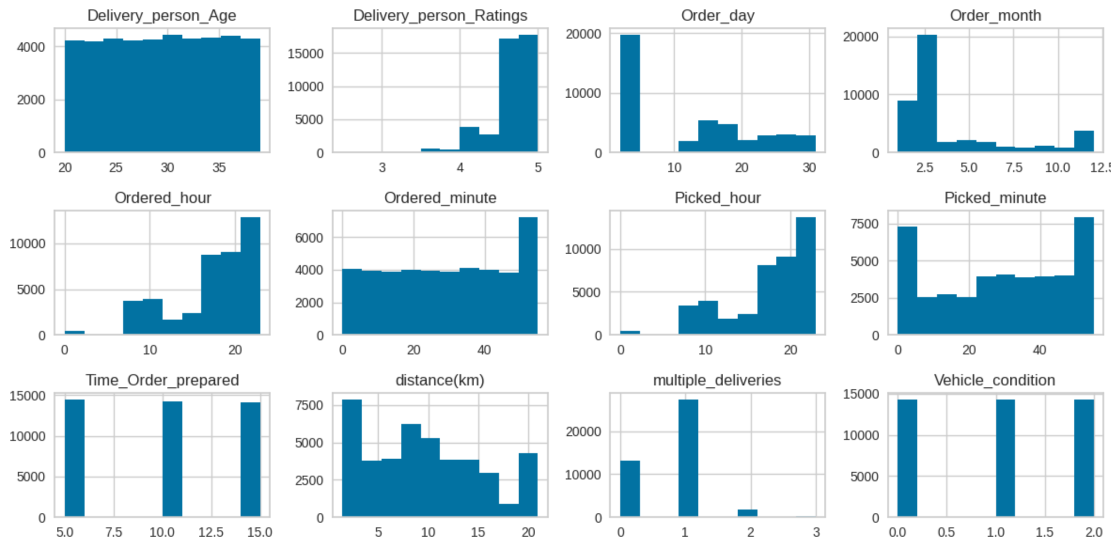


Delivering food over a distance of 2000 km in less than an hour using two-wheelers is an impractical and unrealistic proposition.

[54]:
```
def transform_outliers(data):
  data = data[data['distance(km)'] < 1000]
  return data

df_train6 = transform_outliers(df_train5)
```

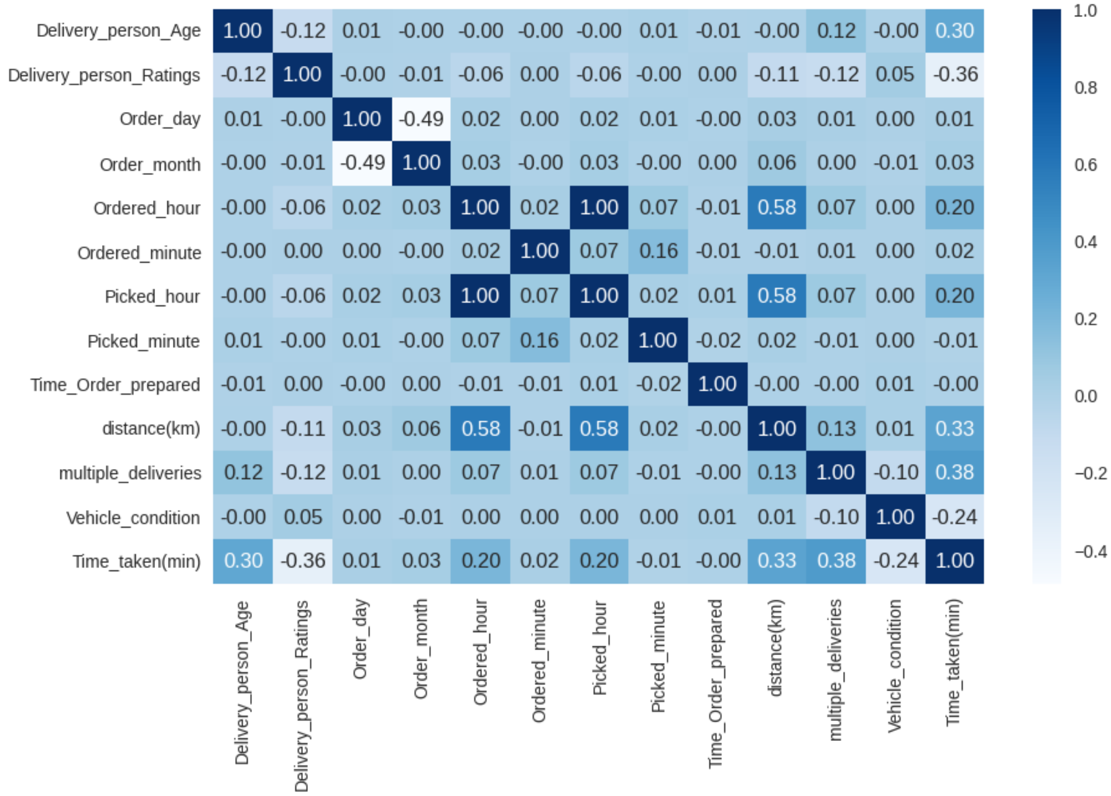
Plotting histogram of numeric variables

[25]:
```
df_train6[numeric_features].hist(layout=(3,4), figsize=(12,6))
plt.tight_layout()
```



Checking the correlation between numeric variables and target

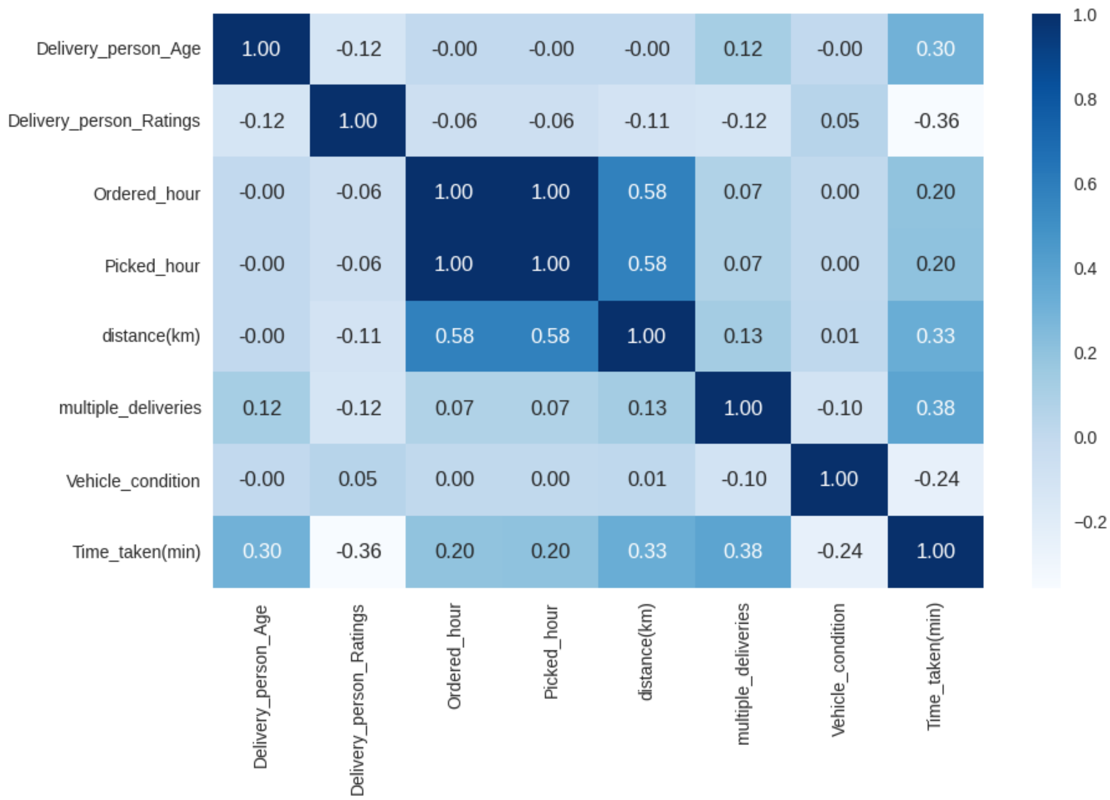
[26]:
```python
data_for_heatmap = df_train6[numeric_features + [target]]

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

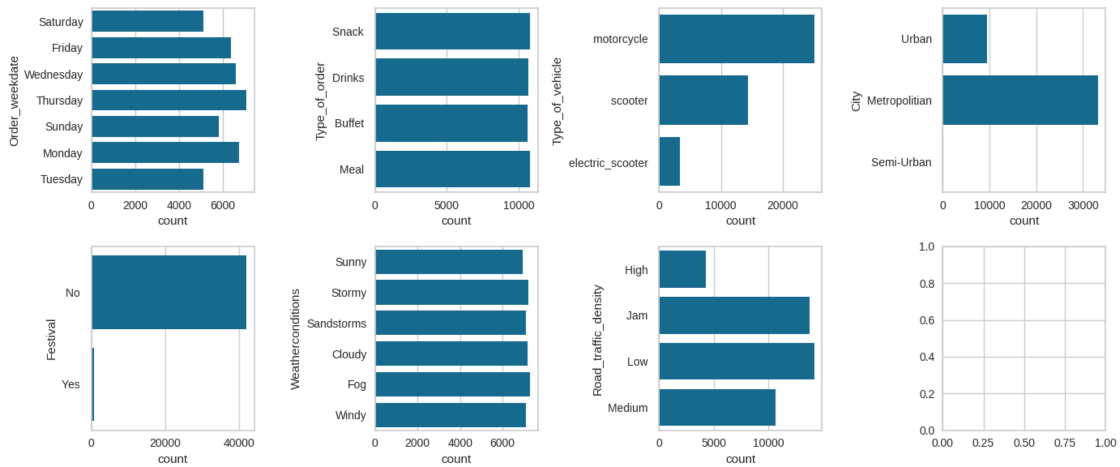
Adjust the columns_to_keep list

[29]:
```
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()
```



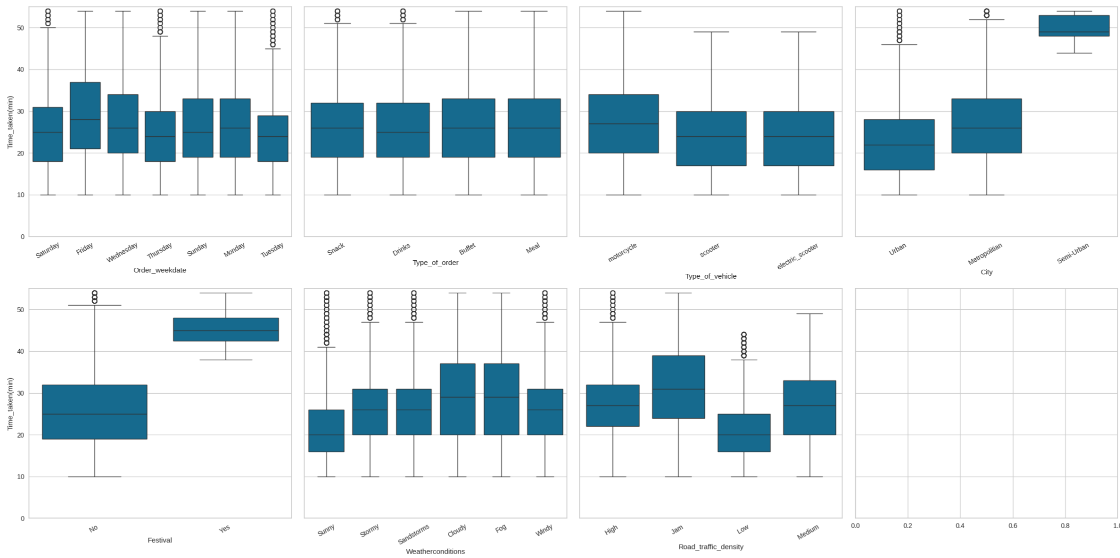
Plotting bar chart of categorical variables

[37]:
```
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

[38]:
```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()
```



3. Using Pycaret to Identify The Best Model

[ ]:
```
# Setup PYCARET

# install pycaret
!pip install pycaret

# import RegressionExperiment and initiate the class

from pycaret.regression import RegressionExperiment
exp = RegressionExperiment()

# check the type of exp
type(exp)
```

[55]:
```
# 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 0x7a2cbd0d5de0>
```

[55]:
```
<pycaret.regression.oop.RegressionExperiment at 0x7a2d02417ac0>
```

[56]:
```
# compare baseline models

best = exp.compare_models()
```

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

```
<pandas.io.formats.style.Styler at 0x7a2cb8e4ba00>

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

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

[57]:
```python
lightgbm_model = exp.create_model('lightgbm')
```

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

<pandas.io.formats.style.Styler at 0x7a2cb84badd0>

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

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

[58]:
```python
tuned_lightgbm_model = exp.tune_model(lightgbm_model)
```

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

<pandas.io.formats.style.Styler at 0x7a2cba6b9720>

Processing:   0%|          | 0/7 [00:00<?, ?it/s]

Fitting 10 folds for each of 10 candidates, totalling 100 fits

<IPython.core.display.HTML object>

Original model was better than the tuned model, hence it will be returned. NOTE:
The display metrics are for the tuned model (not the original one).
```

[59]:
```python
holdout_pred = exp.predict_model(lightgbm_model)
```

```
<pandas.io.formats.style.Styler at 0x7a2cb94d3be0>
```

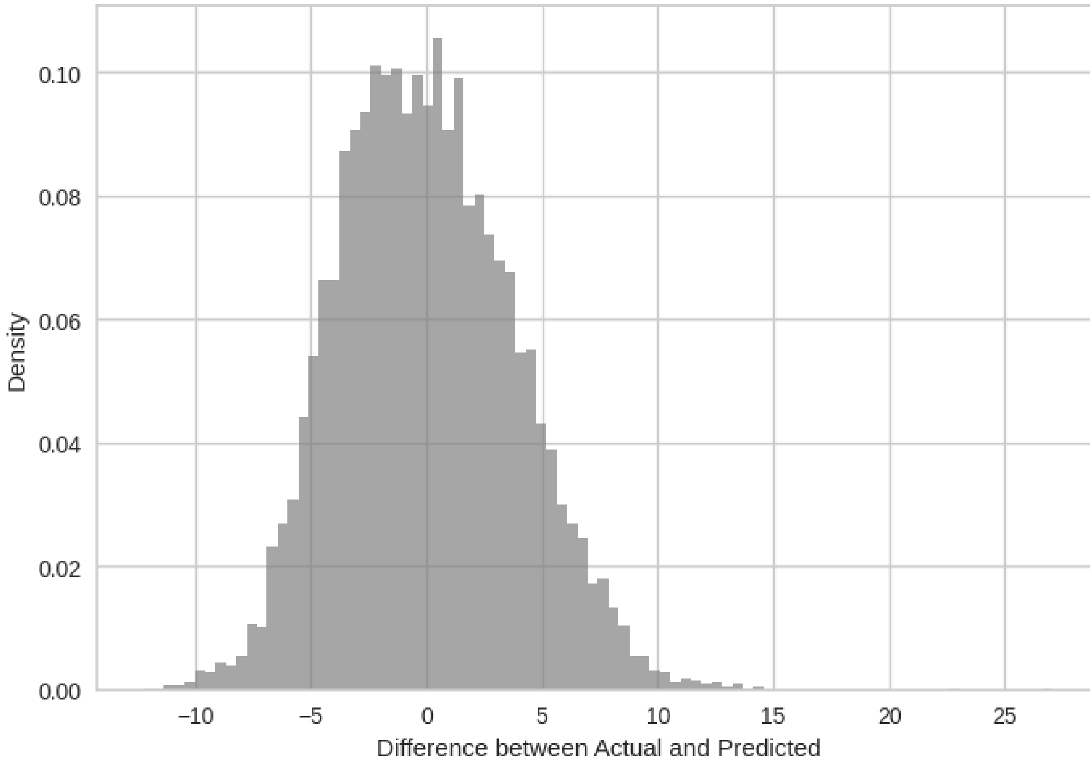
[60]:
```python
holdout_pred.head()
```

[60]:
```
       Delivery_person_Age  Delivery_person_Ratings Order_weekdate  \
25267                 30.0                      4.8      Wednesday
22207                 36.0                      5.0       Saturday
38148                 37.0                      4.9      Wednesday
40569                 24.0                      4.6       Thursday
15038                 28.0                      5.0         Friday

       Ordered_hour  Picked_hour  distance(km) Type_of_order  \
25267            22           22     12.552276       Buffet
22207            13           13      6.118387        Snack
38148             9           10      3.068343        Snack
40569            12           12      6.045846        Snack
15038            21           21     20.807497         Meal

         Type_of_vehicle  multiple_deliveries          City Festival  \
25267  electric_scooter                   0.0         Urban       No
```

```
22207         motorcycle                   0.0  Metropolitian       No
38148         motorcycle                   1.0          Urban       No
40569         motorcycle                   0.0  Metropolitian       No
15038   electric_scooter                   0.0          Urban       No

      Weatherconditions Road_traffic_density  Vehicle_condition  \
25267            Stormy                 Low                  2
22207             Windy                High                  2
38148            Cloudy                 Low                  0
40569             Windy                High                  2
15038            Stormy                 Jam                  2

       Time_taken(min)  prediction_label
25267               15         20.694558
22207               29         28.781621
38148               23         24.191445
40569               27         18.914969
15038               21         19.955154
```

[61]:

```
holdout_pred['difference'] = holdout_pred['Time_taken(min)'] -
 ↪holdout_pred['prediction_label']

import matplotlib.pyplot as plt

plt.hist(holdout_pred['difference'], bins='auto', density=True, color='grey',
 ↪alpha=0.7)
plt.xlabel('Difference between Actual and Predicted')
plt.ylabel('Density')
plt.show()
```



[ ]:
```
# import predict data

files.upload()
```

4. Cleaning the Predict Data

[46]:
```
df_predict = pd.read_csv('predict.csv')
df_predict.head()
```

[46]:
```
        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
```

[50]:
```
def transform_outliers_new(data):
  data = data[data['distance(km)'] < 1000]
  return data
```

[62]:
```
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()
```

[62]:
```
   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     17.042985         Snack      motorcycle
2           19     13.390474        Drinks      motorcycle
3           17     13.397932          Meal         scooter
4           18     17.042634        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
1                  Jam                  0
2                  Jam                  0
3               Medium                  1
4               Medium                  2
5                  Low                  0
```

5. Make Prediction with Pycaret

[63]:
```
predictions_pycaret = exp.predict_model(lightgbm_model, data = df_predict6)
predictions_pycaret.head()
```

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

[63]:
```
   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         30.514893
2                  Jam                  0         30.436237
3               Medium                  1         31.192454
4               Medium                  2         22.242044
5                  Low                  0         19.252308
```

[64]:
```python
# Save model (pipeline)

exp.save_model(best, 'time_delivery_pred_pipeline')
```

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

[64]:
```
(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')
```

[65]:
```python
# Load pipeline

exp.load_model('time_delivery_pred_pipeline')
```

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

[65]:
```
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. Make Prediction by Manually Building a Model

[66]:
```
df_train6.head()
```

[66]:
```
   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
```

6.1 Data Preprocessing

[67]:
```python
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
```

[67]:
```
                                   count          mean       std        min  \
Delivery_person_Age              43706.0 -6.527317e-17  1.000011  -1.662208
Delivery_person_Ratings          43706.0  1.622278e-16  1.000011  -6.824140
Ordered_hour                     43706.0 -4.235034e-17  1.000011  -3.616413
Picked_hour                      43706.0  2.471114e-17  1.000011  -3.224137
distance(km)                     43706.0 -3.722928e-17  1.000011  -1.475079
multiple_deliveries              43706.0  6.340358e-18  1.000011  -1.320326
Vehicle_condition                43706.0 -6.974393e-17  1.000011  -1.225663
Time_taken(min)                  43706.0  2.629815e+01  9.376591  10.000000
Order_weekdate_Friday            43706.0  1.492015e-01  0.356291   0.000000
Order_weekdate_Monday            43706.0  1.577129e-01  0.364476   0.000000
Order_weekdate_Saturday          43706.0  1.191370e-01  0.323953   0.000000
Order_weekdate_Sunday            43706.0  1.362742e-01  0.343083   0.000000
Order_weekdate_Thursday          43706.0  1.649659e-01  0.371154   0.000000
Order_weekdate_Tuesday           43706.0  1.184277e-01  0.323118   0.000000
Order_weekdate_Wednesday         43706.0  1.542809e-01  0.361222   0.000000
Type_of_order_Buffet             43706.0  2.474260e-01  0.431521   0.000000
Type_of_order_Drinks             43706.0  2.484556e-01  0.432122   0.000000
Type_of_order_Meal               43706.0  2.511097e-01  0.433656   0.000000
Type_of_order_Snack              43706.0  2.530087e-01  0.434741   0.000000
Type_of_vehicle_electric_scooter 43706.0  8.053814e-02  0.272128   0.000000
Type_of_vehicle_motorcycle       43706.0  5.842905e-01  0.492850   0.000000
Type_of_vehicle_scooter          43706.0  3.351714e-01  0.472056   0.000000
City_Metropolitian               43706.0  7.740585e-01  0.418206   0.000000
City_Semi-Urban                  43706.0  3.569304e-03  0.059638   0.000000
City_Urban                       43706.0  2.223722e-01  0.415845   0.000000
Festival_No                      43706.0  9.803917e-01  0.138652   0.000000
```

Festival_Yes	43706.0	1.960829e-02	0.138652	0.000000
Weatherconditions_Cloudy	43706.0	1.673683e-01	0.373309	0.000000
Weatherconditions_Fog	43706.0	1.704800e-01	0.376058	0.000000
Weatherconditions_Sandstorms	43706.0	1.659955e-01	0.372081	0.000000
Weatherconditions_Stormy	43706.0	1.688555e-01	0.374629	0.000000
Weatherconditions_Sunny	43706.0	1.619686e-01	0.368426	0.000000
Weatherconditions_Windy	43706.0	1.653320e-01	0.371484	0.000000
Road_traffic_density_High	43706.0	9.849906e-02	0.297992	0.000000
Road_traffic_density_Jam	43706.0	3.146708e-01	0.464390	0.000000
Road_traffic_density_Low	43706.0	3.432709e-01	0.474806	0.000000
Road_traffic_density_Medium	43706.0	2.435592e-01	0.429235	0.000000

	25%	50%	75%	max
Delivery_person_Age	-0.792378	0.077451	0.947280	1.643144
Delivery_person_Ratings	-0.432419	0.206753	0.845925	1.165511
Ordered_hour	-0.502982	0.327267	0.742391	1.157515
Picked_hour	-0.592206	0.347769	0.723759	1.099749
distance(km)	-0.904903	-0.091747	0.704684	2.006289
multiple_deliveries	-1.320326	0.441049	0.441049	3.963798
Vehicle_condition	-1.225663	-0.000869	1.223926	1.223926
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
```

6.2 Feature Engineering

[68]:
```python
# 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)
```

6.3 Training Model: LightGBM

[69]:
```python
# Building and Training the LightGBM Model

import lightgbm as lgb

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)
```

[70]:
```python
# Making Predictions

y_pred = model.predict(X_test)
```

6.4 Evaluating Model Performance

[71]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# 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): 3.0653179167740547
Mean Squared Error (MSE): 14.385000698377013
Root Mean Squared Error (RMSE): 3.7927563457697904
R-squared (R2): 0.8351687849548064
```

6.5 Fine-Tuning Model

[ ]:
```python
from sklearn.model_selection import GridSearchCV

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_
```

[73]:
```python
# Print the best parameters
print("Best Parameters:", best_params)
```

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

[74]:
```python
# Making Predictions

best_y_pred = best_model.predict(X_test)
```

[75]:
```python
# 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): 3.0133514809382573
Mean Squared Error (MSE): 13.874662643216084
Root Mean Squared Error (RMSE): 3.7248708223529152
R-squared (R2): 0.8410165178454603
```

6.6 Feature Engineering (Predict Data)

[76]:
```python
# 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
```

[76]:
```
                               count          mean       std       min  \
Delivery_person_Age          10918.0  1.181852e-15  1.000046 -1.657059
Delivery_person_Ratings      10918.0 -9.664370e-16  1.000046 -6.579854
Ordered_hour                 10918.0 -3.358125e-16  1.000046 -3.608841
Picked_hour                  10918.0 -5.206395e-17  1.000046 -3.228631
distance(km)                 10918.0  5.141315e-17  1.000046 -1.478451
multiple_deliveries          10918.0 -1.145407e-16  1.000046 -1.327104
Vehicle_condition            10918.0 -9.696910e-17  1.000046 -1.231652
```

Order_weekdate_Friday	10918.0	1.448983e-01	0.352014	0.000000
Order_weekdate_Monday	10918.0	1.644074e-01	0.370662	0.000000
Order_weekdate_Saturday	10918.0	1.199853e-01	0.324959	0.000000
Order_weekdate_Sunday	10918.0	1.328082e-01	0.339383	0.000000
Order_weekdate_Thursday	10918.0	1.643158e-01	0.370579	0.000000
Order_weekdate_Tuesday	10918.0	1.172376e-01	0.321718	0.000000
Order_weekdate_Wednesday	10918.0	1.563473e-01	0.363201	0.000000
Type_of_order_Buffet	10918.0	2.528851e-01	0.434686	0.000000
Type_of_order_Drinks	10918.0	2.558161e-01	0.436339	0.000000
Type_of_order_Meal	10918.0	2.442755e-01	0.429676	0.000000
Type_of_order_Snack	10918.0	2.470233e-01	0.431300	0.000000
Type_of_vehicle_electric_scooter	10918.0	7.876901e-02	0.269390	0.000000
Type_of_vehicle_motorcycle	10918.0	5.861879e-01	0.492538	0.000000
Type_of_vehicle_scooter	10918.0	3.350430e-01	0.472027	0.000000
City_Metropolitian	10918.0	7.734933e-01	0.418590	0.000000
City_Semi-Urban	10918.0	4.121634e-03	0.064070	0.000000
City_Urban	10918.0	2.223851e-01	0.415868	0.000000
Festival_No	10918.0	9.816816e-01	0.134106	0.000000
Festival_Yes	10918.0	1.831837e-02	0.134106	0.000000
Weatherconditions_Cloudy	10918.0	1.656897e-01	0.371819	0.000000
Weatherconditions_Fog	10918.0	1.587287e-01	0.365440	0.000000
Weatherconditions_Sandstorms	10918.0	1.664224e-01	0.372476	0.000000
Weatherconditions_Stormy	10918.0	1.610185e-01	0.367565	0.000000
Weatherconditions_Sunny	10918.0	1.755816e-01	0.380481	0.000000
Weatherconditions_Windy	10918.0	1.725591e-01	0.377883	0.000000
Road_traffic_density_High	10918.0	9.809489e-02	0.297457	0.000000
Road_traffic_density_Jam	10918.0	3.119619e-01	0.463316	0.000000
Road_traffic_density_Low	10918.0	3.442022e-01	0.475129	0.000000
Road_traffic_density_Medium	10918.0	2.457410e-01	0.430545	0.000000

	25%	50%	75%	max
Delivery_person_Age	-0.785517	0.000688	0.783259	1.654802
Delivery_person_Ratings	-0.414380	0.202167	0.818714	1.126988
Ordered_hour	-0.505641	0.321879	0.735639	1.149399
Picked_hour	-0.597655	0.341979	0.717833	1.093687
distance(km)	-0.907731	-0.093860	0.703373	2.006220
multiple_deliveries	-1.327104	0.430669	0.430669	3.946214
Vehicle_condition	-1.231652	-0.008403	1.214846	1.214846
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  0.000000  1.000000
```

6.7 Making Prediction

[77]:
```
prediction_normal = best_model.predict(df_predict6_copy)

prediction_normal
```

[77]:
```
array([30.20313861, 29.84643809, 31.14962591, …, 29.28750977,
       26.11095041, 24.51935491])
```

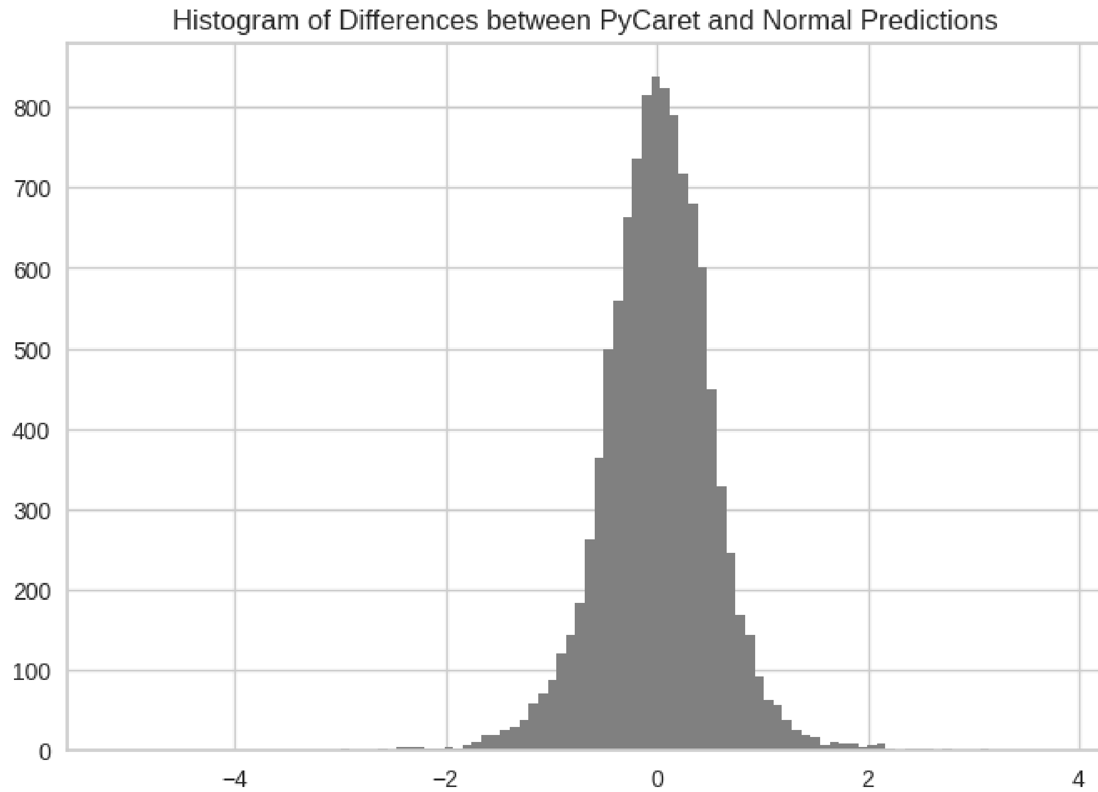
[78]:
```
# save model
import joblib

# 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')
```

[78]:
```
['best_params.joblib']
```

[79]:
```
# Load the saved model
loaded_model = joblib.load('best_model.joblib')

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

6.8 Compare Pycaret and Normal Method

[80]:
```
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()
```



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

[83]:
```
from google.colab import drive
import nbformat
from nbconvert import PDFExporter

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

# Get the notebook name
notebook_name = 'Delivery_Time_Prediction.ipynb'  # Replace with your notebook␣
    ↪name
```

```python
# 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}')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
PDF saved to: /content/drive/My Drive/Colab
Notebooks/Delivery_Time_Prediction.pdf
```