delivery-time-prediction

March 26, 2024

[1]:
```python
import pandas as pd
import numpy as np
from geopy.distance import geodesic
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import xgboost as xgb
```

[2]:
```python
# Load the data
df = pd.read_csv('train.csv')
df.head().T
```

[2]:
```
                                           0                  1  \
ID                                    0x4607             0xb379
Delivery_person_ID             INDORES13DEL02     BANGRES18DEL02
Delivery_person_Age                       37                 34
Delivery_person_Ratings                  4.9                4.5
Restaurant_latitude                22.745049          12.913041
Restaurant_longitude               75.892471          77.683237
Delivery_location_latitude         22.765049          13.043041
Delivery_location_longitude        75.912471          77.813237
Order_Date                        19-03-2022         25-03-2022
Time_Orderd                         11:30:00           19:45:00
Time_Order_picked                   11:45:00           19:50:00
Weatherconditions          conditions Sunny  conditions Stormy
Road_traffic_density                   High                Jam
Vehicle_condition                          2                  2
Type_of_order                         Snack              Snack
Type_of_vehicle                  motorcycle            scooter
multiple_deliveries                        0                  1
Festival                                  No                 No
City                                   Urban       Metropolitian
Time_taken(min)                     (min) 24           (min) 33
```

```
                                              2                  3  \
ID                                      0x5d6d             0x7a6a
Delivery_person_ID              BANGRES19DEL01   COIMBRES13DEL02
Delivery_person_Age                         23                 38
Delivery_person_Ratings                    4.4                4.7
Restaurant_latitude                  12.914264          11.003669
Restaurant_longitude                   77.6784          76.976494
Delivery_location_latitude           12.924264          11.053669
Delivery_location_longitude            77.6884          77.026494
Order_Date                          19-03-2022         05-04-2022
Time_Orderd                           08:30:00           18:00:00
Time_Order_picked                     08:45:00           18:10:00
Weatherconditions         conditions Sandstorms  conditions Sunny
Road_traffic_density                      Low             Medium
Vehicle_condition                            0                  0
Type_of_order                          Drinks             Buffet
Type_of_vehicle                    motorcycle         motorcycle
multiple_deliveries                          1                  1
Festival                                   No                 No
City                                    Urban       Metropolitian
Time_taken(min)                       (min) 26           (min) 21

                                          4
ID                                   0x70a2
Delivery_person_ID           CHENRES12DEL01
Delivery_person_Age                      32
Delivery_person_Ratings                 4.6
Restaurant_latitude               12.972793
Restaurant_longitude              80.249982
Delivery_location_latitude        13.012793
Delivery_location_longitude       80.289982
Order_Date                       26-03-2022
Time_Orderd                        13:30:00
Time_Order_picked                  13:45:00
Weatherconditions         conditions Cloudy
Road_traffic_density                   High
Vehicle_condition                         1
Type_of_order                         Snack
Type_of_vehicle                     scooter
multiple_deliveries                       1
Festival                                 No
City                          Metropolitian
Time_taken(min)                    (min) 30
```

[3]:
```
df.shape
```

[3]:
```
(45593, 20)
```

[4]:
```
df.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
```

[5]:
```
df.describe()
```

[5]:
```
       Restaurant_latitude  Restaurant_longitude  Delivery_location_latitude  \
count         45593.000000          45593.000000                45593.000000
mean             17.017729             70.231332                   17.465186
std               8.185109             22.883647                    7.335122
min             -30.905562            -88.366217                    0.010000
25%              12.933284             73.170000                   12.988453
50%              18.546947             75.898497                   18.633934
75%              22.728163             78.044095                   22.785049
max              30.914057             88.433452                   31.054057

       Delivery_location_longitude  Vehicle_condition
count                 45593.000000       45593.000000
mean                     70.845702           1.023359
std                      21.118812           0.839065
min                       0.010000           0.000000
```

```
25%                        73.280000           0.000000
50%                        76.002574           1.000000
75%                        78.107044           2.000000
max                        88.563452           3.000000
```

[6]:
```
for i in df.columns:
    print(i)
    print(df[i].value_counts())
    print("--------------------------------------")
```

```
ID
ID
0x4607     1
0x1f3e     1
0xe251     1
0x3f31     1
0x4a78     1
          ..
0xc3f1     1
0x5db7     1
0x1985     1
0xceda     1
0x5fb2     1
Name: count, Length: 45593, dtype: int64
--------------------------------------
Delivery_person_ID
Delivery_person_ID
PUNERES01DEL01     67
JAPRES11DEL02      67
HYDRES04DEL02      66
JAPRES03DEL01      66
VADRES11DEL02      66
                   ..
DEHRES18DEL03       7
AURGRES11DEL03      7
KOLRES09DEL03       6
KOCRES16DEL03       6
BHPRES010DEL03      5
Name: count, Length: 1320, dtype: int64
--------------------------------------
Delivery_person_Age
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: count, dtype: int64
--------------------------------------
Delivery_person_Ratings
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: count, dtype: int64
--------------------------------------
Restaurant_latitude
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: count, Length: 657, dtype: int64
--------------------------------------
Restaurant_longitude
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: count, Length: 518, dtype: int64
--------------------------------------
Delivery_location_latitude
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: count, Length: 4373, dtype: int64
--------------------------------------
Delivery_location_longitude
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: count, Length: 4373, dtype: int64
--------------------------------------
Order_Date
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: count, dtype: int64
--------------------------------------
Time_Orderd
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: count, Length: 177, dtype: int64
--------------------------------------
Time_Order_picked
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: count, Length: 193, dtype: int64
--------------------------------------
Weatherconditions
```

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

```
NaN       993
3         361
Name: count, dtype: int64
--------------------------------------
Festival
Festival
No      44469
Yes       896
NaN       228
Name: count, dtype: int64
--------------------------------------
City
City
Metropolitian     34093
Urban             10136
NaN                1200
Semi-Urban          164
Name: count, dtype: int64
--------------------------------------
Time_taken(min)
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: count, dtype: int64
--------------------------------------
```

[7]:

```
import folium
from folium.plugins import MarkerCluster

# Create a map centered at an average latitude and longitude
m = folium.Map(location=[df['Restaurant_latitude'].mean(),␣
 ↪df['Restaurant_longitude'].mean()], zoom_start=10)

# Create a MarkerCluster layer for restaurant locations
restaurant_cluster = MarkerCluster().add_to(m)

# Add markers for restaurant locations to the MarkerCluster layer
for index, row in df.iterrows():
    folium.Marker(location=[row['Restaurant_latitude'],␣
 ↪row['Restaurant_longitude']], popup=row['Type_of_order']).
 ↪add_to(restaurant_cluster)

# Create a MarkerCluster layer for delivery locations
delivery_cluster = MarkerCluster().add_to(m)

# Add markers for delivery locations to the MarkerCluster layer
for index, row in df.iterrows():
    folium.Marker(location=[row['Delivery_location_latitude'],␣
 ↪row['Delivery_location_longitude']], popup='Delivery').
 ↪add_to(delivery_cluster)

# Display the map
```

```
m
```

[7]: <folium.folium.Map at 0x11e18441850>

0.0.1 data cleaning

```
[8]: # Update column names
def update_column_name(d):
    d.rename(columns={'Weatherconditions': 'Weather_conditions'}, inplace=True)

update_column_name(df)
```

```
[9]: def extract_column_value(d):
    # Extract time and convert to int
    d['Time_taken(min)'] = d['Time_taken(min)'].apply(lambda x: int(x.split('␣
↪')[1].strip()) if isinstance(x, str) else x)
    # Extract Weather conditions with error handling
    d['Weather_conditions'] = d['Weather_conditions'].apply(lambda x: x.split('␣
↪')[1].strip() if len(x.split(' ')) > 1 else x)
    # Extract city code from Delivery person ID
    d['City_code'] = d['Delivery_person_ID'].str.split("RES", expand=True)[0]

extract_column_value(df)
df[['Time_taken(min)', 'Weather_conditions', 'City_code']].head()
```

[9]:

	Time_taken(min)	Weather_conditions	City_code
0	24	Sunny	INDO
1	33	Stormy	BANG
2	26	Sandstorms	BANG
3	21	Sunny	COIMB
4	30	Cloudy	CHEN

```
[24]: # Drop columns not needed for modeling
def drop_columns(d):
    d.drop(['ID', 'Delivery_person_ID'], axis=1, inplace=True)

drop_columns(df)
```

```
[25]: # Drop duplicate rows
df.drop_duplicates(inplace=True)
```

```
[26]: # Update data types
def update_datatype(df):
    df['Delivery_person_Age'] = df['Delivery_person_Age'].astype('float64')
    df['Delivery_person_Ratings'] = df['Delivery_person_Ratings'].
↪astype('float64')
```

```
update_datatype(df)
```

[27]:
```
# Convert string 'NaN' to np.nan
def convert_nan(d):
    d.replace('NaN', float(np.nan), regex=True, inplace=True)

convert_nan(df)
```

[29]:
```
# Extract date features
def extract_date_features(data):
    data["Order_Date"] = pd.to_datetime(data["Order_Date"])
    data["day"] = data.Order_Date.dt.day
    data["month"] = data.Order_Date.dt.month
    data["quarter"] = data.Order_Date.dt.quarter
    data["year"] = data.Order_Date.dt.year
    data['day_of_week'] = data.Order_Date.dt.dayofweek.astype(int)
    data["is_month_start"] = data.Order_Date.dt.is_month_start.astype(int)
    data["is_month_end"] = data.Order_Date.dt.is_month_end.astype(int)
    data["is_quarter_start"] = data.Order_Date.dt.is_quarter_start.astype(int)
    data["is_quarter_end"] = data.Order_Date.dt.is_quarter_end.astype(int)
    data["is_year_start"] = data.Order_Date.dt.is_year_start.astype(int)
    data["is_year_end"] = data.Order_Date.dt.is_year_end.astype(int)
    data['is_weekend'] = np.where(data['day_of_week'].isin([5,6]), 1, 0)

extract_date_features(df)
df.head()
```

[29]:
```
   Delivery_person_Age  Delivery_person_Ratings  Restaurant_latitude  \
0                 37.0                      4.9            22.745049
1                 34.0                      4.5            12.913041
2                 23.0                      4.4            12.914264
3                 38.0                      4.7            11.003669
4                 32.0                      4.6            12.972793

   Restaurant_longitude  Delivery_location_latitude  \
0             75.892471                   22.765049
1             77.683237                   13.043041
2             77.678400                   12.924264
3             76.976494                   11.053669
4             80.249982                   13.012793

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

```
4                   80.289982 2022-03-26    13:30:00          13:45:00

  Weather_conditions  … quarter  year day_of_week is_month_start  \
0              Sunny  …        1  2022           5              0
1             Stormy  …        1  2022           4              0
2         Sandstorms  …        1  2022           5              0
3              Sunny  …        2  2022           1              0
4             Cloudy  …        1  2022           5              0

  is_month_end is_quarter_start is_quarter_end  is_year_start is_year_end  \
0            0                0              0              0           0
1            0                0              0              0           0
2            0                0              0              0           0
3            0                0              0              0           0
4            0                0              0              0           0

   is_weekend
0           1
1           0
2           1
3           0
4           1

[5 rows x 31 columns]
```

[30]:
```python
# Calculate time difference
def calculate_time_diff(d):
    d['Time_Orderd'] = pd.to_timedelta(d['Time_Orderd'])
    d['Time_Order_picked'] = pd.to_timedelta(d['Time_Order_picked'])
    d['order_prepare_time'] = ((d['Time_Order_picked'] - d['Time_Orderd']).dt.
 ↪total_seconds() / 60)
    d['order_prepare_time'].fillna(d['order_prepare_time'].median(),␣
 ↪inplace=True)
    d.drop(['Time_Orderd', 'Time_Order_picked', 'Order_Date'], axis=1,␣
 ↪inplace=True)

calculate_time_diff(df)
```

[31]:
```python
# Calculate distance between restaurant location & delivery location
def calculate_distance(d):
    restaurant_coordinates = df[['Restaurant_latitude',␣
 ↪'Restaurant_longitude']].to_numpy()
    delivery_location_coordinates = df[['Delivery_location_latitude',␣
 ↪'Delivery_location_longitude']].to_numpy()
    d['distance'] = np.array([geodesic(restaurant, delivery).meters for␣
 ↪restaurant, delivery in zip(restaurant_coordinates,␣
 ↪delivery_location_coordinates)])
```

```
calculate_distance(df)
```

0.0.2 data preprocessing

[36]:
```
# Label encoding for categorical variables
from sklearn.preprocessing import LabelEncoder

def label_encoding(d):
    categorical_columns = d.select_dtypes(include=['object']).columns
    label_encoder = LabelEncoder()
    d[categorical_columns] = d[categorical_columns].apply(lambda col:␣
 ↪label_encoder.fit_transform(col))

label_encoding(df)
df.head()
```

[36]:
```
   Delivery_person_Age  Delivery_person_Ratings  Restaurant_latitude  \
0                 37.0                      4.9            22.745049
1                 34.0                      4.5            12.913041
2                 23.0                      4.4            12.914264
3                 38.0                      4.7            11.003669
4                 32.0                      4.6            12.972793

   Restaurant_longitude  Delivery_location_latitude  \
0             75.892471                   22.765049
1             77.683237                   13.043041
2             77.678400                   12.924264
3             76.976494                   11.053669
4             80.249982                   13.012793

   Delivery_location_longitude  Weather_conditions  Road_traffic_density  \
0                    75.912471                   4                     0
1                    77.813237                   3                     1
2                    77.688400                   2                     2
3                    77.026494                   4                     3
4                    80.289982                   0                     0

   Vehicle_condition  Type_of_order  …  day_of_week  is_month_start  \
0                  2              3  …            5               0
1                  2              3  …            4               0
2                  0              1  …            5               0
3                  0              0  …            1               0
4                  1              3  …            5               0

   is_month_end  is_quarter_start  is_quarter_end  is_year_start  is_year_end  \
0             0                 0               0              0            0
```

```
1          0            0           0          0          0
2          0            0           0          0          0
3          0            0           0          0          0
4          0            0           0          0          0

   is_weekend  order_prepare_time      distance
0           1                15.0   3020.736643
1           0                 5.0  20143.736910
2           1                15.0   1549.692932
3           0                10.0   7774.496620
4           1                15.0   6197.897917

[5 rows x 30 columns]
```

[35]:

```
# Drop rows with missing values
df.dropna(inplace=True)
```

[37]:

```
# Split features & label
X = df.drop('Time_taken(min)', axis=1)
y = df['Time_taken(min)']

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[38]:

```
# Standardize numeric features
numeric_columns = X_train.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
X_train[numeric_columns] = scaler.fit_transform(X_train[numeric_columns])
X_test[numeric_columns] = scaler.transform(X_test[numeric_columns])
```

0.1 Model Building

Steps

Employ cross-validation & hyper parameter tuning to determine the optimal regression model. Construct the food delivery prediction model using the identified best model. Evaluate the model's performance on the testing data to assess its accuracy and reliability.

0.2 cross validation

[39]:

```
# Define models and parameter grids
models = [
    LinearRegression(),
    Ridge(),
    Lasso(),
    DecisionTreeRegressor(),
    RandomForestRegressor(),
```

```
    GradientBoostingRegressor(),
    xgb.XGBRegressor(),
]

param_grid = [
    {},  # For LinearRegression
    {'alpha': [0.01, 0.1, 1.0, 10.0, 100.0]},  # For Ridge
    {'alpha': [0.001, 0.01, 0.1, 1.0]},  # For Lasso
    {'max_depth': [3, 5, 7, 9, 11]},  # For DecisionTreeRegressor
    {'n_estimators': [100, 200, 300, 400, 500], 'max_depth': [5, 7, 9]},  # For RandomForestRegressor
    {'n_estimators': [100, 200, 300], 'learning_rate': [0.01, 0.05, 0.1]},  # For GradientBoostingRegressor
    {'n_estimators': [50, 100, 150], 'max_depth': [3, 5, 7], 'learning_rate': [0.01, 0.05, 0.1]}  # For XGBRegressor
]

import time

# Perform grid search and print results
for i, model in enumerate(models):
    start_time = time.time()  # Start timing
    grid_search = GridSearchCV(model, param_grid[i], cv=5, scoring='r2')
    grid_search.fit(X_train, y_train)
    end_time = time.time()  # End timing

    print(f"{model.__class__.__name__}:")
    print("Best parameters:", grid_search.best_params_)
    print("Best R2 score:", grid_search.best_score_)
    print("Time taken:", end_time - start_time, "seconds")
    print()
```

```
LinearRegression:
Best parameters: {}
Best R2 score: 0.3813186769368565
Time taken: 0.11083817481994629 seconds

Ridge:
Best parameters: {'alpha': 1.0}
Best R2 score: 0.38136974988791333
Time taken: 0.39036059379577637 seconds
```

```
C:\Users\hp\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 7.538e+03, tolerance: 2.460e+02
```

```
  model = cd_fast.enet_coordinate_descent(
C:\Users\hp\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 9.880e+04, tolerance: 2.459e+02
  model = cd_fast.enet_coordinate_descent(
C:\Users\hp\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 8.714e+04, tolerance: 2.449e+02
  model = cd_fast.enet_coordinate_descent(
C:\Users\hp\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 8.423e+04, tolerance: 2.463e+02
  model = cd_fast.enet_coordinate_descent(
C:\Users\hp\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 6.325e+03, tolerance: 2.451e+02
  model = cd_fast.enet_coordinate_descent(
C:\Users\hp\anaconda3\Lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:678: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality
gap: 2.089e+04, tolerance: 3.071e+02
  model = cd_fast.enet_coordinate_descent(

Lasso:
Best parameters: {'alpha': 0.001}
Best R2 score: 0.3810205667734574
Time taken: 2.002333879470825 seconds

DecisionTreeRegressor:
Best parameters: {'max_depth': 9}
Best R2 score: 0.828121737403951
Time taken: 3.201230764389038 seconds

RandomForestRegressor:
Best parameters: {'max_depth': 9, 'n_estimators': 300}
Best R2 score: 0.8377348431487137
Time taken: 1673.3960962295532 seconds

GradientBoostingRegressor:
```

```
Best parameters: {'learning_rate': 0.1, 'n_estimators': 300}
Best R2 score: 0.7959433586925619
Time taken: 427.98803901672363 seconds

XGBRegressor:
Best parameters: {'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 150}
Best R2 score: 0.8402613854539702
Time taken: 21.436118602752686 seconds
```

0.2.1 Model Building

[40]:
```python
# create a XGB regressor model
model = xgb.XGBRegressor(n_estimators=20, max_depth=9)

# Fit the model on the training data
model.fit(X_train, y_train)
```

[40]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=9, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=20, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
```

[42]:
```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Assuming you have already defined y_test and y_pred
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error (MAE):", round(mae,2))
print("Mean Squared Error (MSE):", round(mse,2))
print("Root Mean Squared Error (RMSE):", round(rmse,2))
print("R-squared (R2) Score:", round(r2,2))
```

```
Mean Absolute Error (MAE): 3.02
Mean Squared Error (MSE): 14.14
Root Mean Squared Error (RMSE): 3.76
R-squared (R2) Score: 0.84
```

1 Conclusion

In conclusion, the food delivery prediction model was developed using XGBoost, achieving an impressive R2 score of 84%. Moving forward, potential enhancements include identifying the best features, conducting additional feature engineering, and exploring other optimization techniques to further improve the model's performance and accuracy. These steps will contribute to fine-tuning the model and unlocking its full potential in predicting food delivery timings accurately.

[ ]:
```

```