Zomato Analysis Case Study Report

Great lakes institute of management

PGPDSE-FT Online April 23, Group # 3

Submitted by:

Harsh Chaudhary	AHZUJOTP1Y
Raghavendra Abhishek	Z60SKJ8ADG

Table of content:

What problem are you solving?

The problem statement is about the analysis of an Indian food delivery aggregator company called ZOMATO. The newly started companies are not able to decide the cost that would happen per two people for once. So Zomato wanted to analyse different restaurants in different locations and their reviews and votings for particular restaurant and wanted to predict the approximate cost for two people for a particular restaurant.

How did you solve the problem?

To solve the problems in the given case studies, a systematic approach was followed that involves several key steps:

1. Data Collection:

Imported data into python using libraries like 'Pandas' to create data frames.

2. Data Inspection:

Checked the first few rows of the data to get an initial sense of its structure.

3.Explore and Understand the Data:

- Performed exploratory data analysis (EDA) to understand the characteristics of the data.
- Checked for missing values, outliers, and data distributions.
- Visualized data using charts and graphs to gain insights.

4 Data Preprocessing:

- Handled missing data through imputation and removal.
- Encoded categorical variables.
- Scaled numerical features.
- Handled outliers .
- Split the dataset into training and testing sets.

5. Feature Engineering:

- Create new features or transform existing ones to better represent the underlying patterns in the data.
- Select relevant features based on domain knowledge and feature importance analysis.

6. Choose a Model:

- Based on the nature of the problem (regression), chosen a suitable machine learning algorithm.

7. Train the Model:

- Splited the data into training and validation sets.

- Trained the chosen model using the training data.
- Adjusted hyperparameters to optimize model performance.
- Validated the model using the validation set to ensure it generalizes well.

8. Evaluate the Model:

- Used metrics such as r square and adjusted r square,f1 score to evaluate the model's performance.

9. Interpret the Model:

- Understood the model's predictions and interpretability.
- Analyzed feature importance to gain insights into the model's decision-making process.

What were the steps took to solve and codes with outputs

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
# 'Scikit-learn' (sklearn) emphasizes various regression, classification and clustering algorithms
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score

# import function for ridge regression
from sklearn.linear_model import Ridge

# import function for lasso regression
from sklearn.linear_model import Lasso

# import function for elastic net regression
from sklearn.linear_model import ElasticNet

# 'Statsmodels' is used to build and analyze various statistical models
import statsmodels
import statsmodels.api as sm
import statsmodels.stats.api as sms
from statsmodels.tools.eval_measures import rmse
from statsmodels.compat import lzip
from statsmodels.graphics.gofplots import ProbPlot

# 'SciPy' is used to perform scientific computations
from scipy.stats import f_oneway
from scipy.stats import jarque_bera
from scipy import stats
```

In [2]:
```
df_zomato=pd.read_csv("C:\\Users\\abhis\\Downloads\\Project 3 (1)\\Project 3\\zomato.csv")
```

In [3]:
```
df_zomato.head()
```

Out[3]:

	url	address	name	online_order	book_table	rate	votes	phone	location
0	https://www.zomato.com/bangalore/jalsa-banasha...	942, 21st Main Road, 2nd Stage, Banashankari, ...	Jalsa	Yes	Yes	4.1/5	775	080 42297555\r\n+91 9743772233	Banashankari
1	https://www.zomato.com/bangalore/spice-elephan...	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	No	4.1/5	787	080 41714161	Banashankari
2	https://www.zomato.com/SanchurroBangalore?cont...	1112, Next to KIMS Medical College, 17th Cross...	San Churro	Yes	No	3.8/5	918	+91 9663487993	Banashankari
3	https://www.zomato.com/bangalore/addhuri-udupi...	1st Floor, Annakuteera, Banashankar...	Addhuri Udupi	No	No	3.7/5	88	+91 9620009302	Banashankari
4	https://www.zomato.com/bangalore/grand-village...	10, 3rd Floor, Lakshmi Associates, Gandhi Baza...	Grand Village	No	No	3.8/5	166	+91 8026612447\r\n+91 9901210005	Basavanagudi

In [4]:
```
df_zomato.shape
```

Out[4]:
```
(51717, 17)
```

In [5]:
```
df_zomato.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51717 entries, 0 to 51716
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   url                          51717 non-null  object
 1   address                      51717 non-null  object
 2   name                         51717 non-null  object
 3   online_order                 51717 non-null  object
 4   book_table                   51717 non-null  object
 5   rate                         43942 non-null  object
 6   votes                        51717 non-null  int64
 7   phone                        50509 non-null  object
 8   location                     51696 non-null  object
 9   rest_type                    51490 non-null  object
 10  dish_liked                   23639 non-null  object
 11  cuisines                     51672 non-null  object
 12  approx_cost(for two people)  51371 non-null  object
 13  reviews_list                 51717 non-null  object
 14  menu_item                    51717 non-null  object
 15  listed_in(type)              51717 non-null  object
 16  listed_in(city)              51717 non-null  object
dtypes: int64(1), object(16)
memory usage: 6.7+ MB
```

In [6]:
```
df_zomato.duplicated().sum()
```

Out[6]:
```
0
```

In [7]:
```
df_zomato.isnull().sum()
```

Out[7]:
```
url                              0
address                          0
name                             0
online_order                     0
book_table                       0
rate                          7775
votes                            0
phone                         1208
location                        21
rest_type                      227
dish_liked                   28078
cuisines                        45
approx_cost(for two people)    346
reviews_list                     0
menu_item                        0
listed_in(type)                  0
listed_in(city)                  0
dtype: int64
```

In [8]:
```
df_zomato['rate'] = df_zomato['rate'].str.split('/').str[0]
df_zomato.head()
```

Out[8]:

	url	address	name	online_order	book_table	rate	votes	phone	location
0	https://www.zomato.com/bangalore/jalsa-banasha...	942, 21st Main Road, 2nd Stage, Banashankari, ...	Jalsa	Yes	Yes	4.1	775	080 42297555\r\n+91 9743772233	Banashankari
1	https://www.zomato.com/bangalore/spice-elephan...	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	No	4.1	787	080 41714161	Banashankari
2	https://www.zomato.com/SanchurroBangalore?cont...	1112, Next to KIMS Medical College, 17th Cross...	San Churro	Yes	No	3.8	918	+91 9663487993	Banashankari
3	https://www.zomato.com/bangalore/addhuri-Banashankar...	1st Floor, Annakuteera,	Addhuri Udupi	No	No	3.7	88	+91 9620009302	Banashankari
4	https://www.zomato.com/bangalore/grand-village...	10, 3rd Floor, Lakshmi Associates, Gandhi Baza...	Grand Village	No	No	3.8	166	+91 8026612447\r\n+91 9901210005	Basavanagudi

Replacing Garbage Values

In [9]:
```
df_zomato['menu_item'].replace({'[]' : np.NaN, "" : np.NaN}, inplace=True)
```

```python
df_zomato['reviews_list'].replace({'[]' : np.NaN, "" : np.NaN}, inplace=True)
```

In [10]:

```python
# Define the  pattern for detecting error values
pattern = r'[^a-zA-Z0-9@&()!,.\'\s -]'

# Use the pattern to filter out error values
df_zomato[df_zomato['name'].str.contains(pattern)].head(50)
```

Out[10]:

	url	address	name	online_order	book_table	rate	votes	
10	https://www.zomato.com/bangalore/caf%C3%A9-dow...	12,29 Near PES University Back Gate, D'Souza N...	CafÃÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ© Down The A...	Yes	No	4.1	402	08
193	https://www.zomato.com/bangalore/caf%C3%A9-dow...	12,29 Near PES University Back Gate, D'Souza N...	CafÃÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ© Down The A...	Yes	No	4.1	402	08
566	https://www.zomato.com/bangalore/caf%C3%A9-dow...	12,29 Near PES University Back Gate, D'Souza N...	CafÃÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ© Down The A...	Yes	No	4.1	402	2
885	https://www.zomato.com/bangalore/refuel-banner...	7, Ground Floor, RR Commercial Complex, Akshay...	#refuel	Yes	No	3.7	37	
974	https://www.zomato.com/bangalore/bohra-bohra-c...	Shop 1, 129, 24th Main, 5th Phase, JP Nagar, B...	Bohra Bohra CafÃÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ©	Yes	No	3.6	43	9
1192	https://www.zomato.com/bangalore/refuel-banner...	7, Ground Floor, RR Commercial Complex, Akshay...	#refuel	Yes	No	3.7	37	
1256	https://www.zomato.com/bangalore/24-7-food-ser...	21/22, Saifee Apartment, 5th Cross, 22nd Main ...	24/7 Food Service	Yes	No	3.0	7	7
1303	https://www.zomato.com/bangalore/foreign-caf%C...	45, 14th Main, 7th Phase, Puttehahalli Palya, ...	Foreign CafÃÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ©	Yes	No	NaN	0	
1347	https://www.zomato.com/bangalore/dilli-bannerg...	Delivery Only	Dilli ??	Yes	No	NaN	0	
1558	https://www.zomato.com/bangalore/bella-s-kitch...	Akshay Nagar, Bannerghatta Road, Bangalore	Bella?s Kitchen	No	No	2.9	31	
1756	https://www.zomato.com/bangalore/e2-entr%C3%A9...	7th Cross, MICO Layout, BTM 2nd Stage, BTM, Ba...	E2 - EntrÃÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ©e Envoy	No	No	3.5	13	
1779	https://www.zomato.com/bangalore/cocoa-n-cr%C3...	203, Nanjundeshwara Layout, 5h Phase, JP Nagar...	Cocoa 'n' CrÃÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ¨me	No	No	3.5	11	
1936	https://www.zomato.com/bangalore/cocoa-n-cr%C3...	203, Nanjundeshwara Layout, 5h Phase, JP Nagar...	Cocoa 'n' CrÃÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ¨me	No	No	3.5	11	
1967	https://www.zomato.com/bangalore/refuel-banner...	7, Ground Floor, RR Commercial Complex, Akshay...	#refuel	Yes	No	3.7	37	
2021	https://www.zomato.com/bangalore/bella-s-kitch...	Akshay Nagar, Bannerghatta	Bella?s Kitchen	No	No	2.9	31	

		Road, Bangalore						
3790	https://www.zomato.com/bangalore/re-fuel-aloft...	Sarjapura - Marathahalli Outer Ring Road, Kadu...	Re:fuel- Aloft Bengaluru Cessna Business Park	No	No	NaN	0	
3911	https://www.zomato.com/bangalore/maia-eat-bake...	253, Green Glen Layout, Bellandur, Bangalore	Maia - Eat | Bake | Mom	Yes	Yes	4.3	996	
4062	https://www.zomato.com/bangalore/foreign-caf%C...	779/71, First Floor, Near Bata Showroom, Kaiko...	Foreign CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©	Yes	No	3.5	43	08
4321	https://www.zomato.com/bangalore/feeltheroll-b...	Opposite Mantri Commercio, Outer Ring Road, De...	#FeelTheROLL	No	No	3.4	7	
4336	https://www.zomato.com/bangalore/pi%C3%B1ata-1...	21, 6th Main,Embassy Pristine Iblur Village, B...	PiÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ±ata	No	No	3.7	19	
4385	https://www.zomato.com/bangalore/mint-spice-ca...	97/1, Green Glen Layout, Bellandur, Bangalore	Mint Spice CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©...	Yes	No	3.0	12	08
4504	https://www.zomato.com/bangalore/re-fuel-aloft...	Sarjapura - Marathahalli Outer Ring Road, Kadu...	Re:fuel- Aloft Bengaluru Cessna Business Park	No	No	NaN	0	
4512	https://www.zomato.com/bangalore/pi%C3%B1ata-1...	21, 6th Main,Embassy Pristine Iblur Village, B...	PiÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ±ata	No	No	3.7	19	
4557	https://www.zomato.com/bangalore/maia-eat-bake...	253, Green Glen Layout, Bellandur, Bangalore	Maia - Eat | Bake | Mom	Yes	Yes	4.3	996	951
4626	https://www.zomato.com/bangalore/mint-spice-ca...	97/1, Green Glen Layout, Bellandur, Bangalore	Mint Spice CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©...	Yes	No	3.0	12	08
4635	https://www.zomato.com/bangalore/re-fuel-aloft...	Sarjapura - Marathahalli Outer Ring Road, Kadu...	Re:fuel- Aloft Bengaluru Cessna Business Park	No	No	NaN	0	
5045	https://www.zomato.com/bangalore/caf%C3%A9-fel...	Level 5, 1 MG Mall, Swami Vivekananda Road, Tr...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© Felix	Yes	Yes	4.4	690	
5056	https://www.zomato.com/bangalore/caf%C3%A9-mor...	Le Parc Richmonde Building, Shantala nagar, Ri...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© Mor'ish	Yes	No	4.1	114	08
5060	https://www.zomato.com/bangalore/urban-solace-...	32, Annaswamy Mudaliar Road, Opposite Ulsoor L...	Urban Solace - CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ...	Yes	No	3.9	461	
5069	https://www.zomato.com/bangalore/caf%C3%A9-mh-...	28, Ameen House, Sulthangunta, Shivajinagar, B...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© MH-28	Yes	No	3.4	9	
5072	https://www.zomato.com/bangalore/wafl-caf%C3%A...	G1 Smart Avenue, 28, Residency Road, Bangalore	WAFL CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©	Yes	No	4.0	85	
		Saanchi, 5						

5074	https://www.zomato.com/bangalore/the-caf%C3%A9...	Setlur Street, Off Langford Road, L...	The CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© @ Saanchi	No	No	4.0	30	
5076	https://www.zomato.com/bangalore/moto-store-ca...	9, Lake Square, Tank Road, Ulsoor, Bangalore	Moto Store & CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©	Yes	No	4.2	382	
5129	https://www.zomato.com/bangalore/moto-store-ca...	9, Lake Square, Tank Road, Ulsoor, Bangalore	Moto Store & CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©	Yes	No	4.2	382	
5232	https://www.zomato.com/bangalore/wafl-caf%C3%A...	G1 Smart Avenue, 28, Residency Road, Bangalore	WAFL CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©	Yes	No	4.0	85	
5329	https://www.zomato.com/bangalore/foreign-caf%C...	4, St Mary's Building, 39/1, 2nd Stage, Indira...	Foreign CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©	Yes	No	3.1	54	080
5358	https://www.zomato.com/bangalore/caf%C3%A9-mor...	Le Parc Richmonde Building, Shantala nagar, Ri...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© Mor'ish	Yes	No	4.1	114	080
5366	https://www.zomato.com/bangalore/yauatcha-p%C3...	Level 5, 1 MG Road Mall, MG Road, Bangalore	Yauatcha PÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ¢tiss...	Yes	No	3.9	53	
5477	https://www.zomato.com/bangalore/caf%C3%A9-mh-...	28, Ameen House, Sulthangunta, Shivajinagar, B...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© MH-28	Yes	No	3.4	9	
5497	https://www.zomato.com/bangalore/urban-solace-...	32, Annaswamy Mudaliar Road, Opposite Ulsoor L...	Urban Solace - CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ...	Yes	No	3.9	463	
5586	https://www.zomato.com/bangalore/midnight-hung...	17, 17th F Main, CMH Road, Indiranagar, Bangalore	Midnight Hunger Solutions 24/7	Yes	No	2.1	252	
5652	https://www.zomato.com/bangalore/caf%C3%A9-fel...	Level 5, 1 MG Mall, Swami Vivekananda Road, Tr...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© Felix	Yes	Yes	4.4	691	
5738	https://www.zomato.com/bangalore/le-ch%C3%A9ri...	24, Nanjappa Road, Shanti Nagar, Bangalore	Le ChÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©ri	No	No	3.8	33	
5843	https://www.zomato.com/bangalore/caf%C3%A9-mor...	Le Parc Richmonde Building, Shantala nagar, Ri...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© Mor'ish	Yes	No	4.1	114	08
5861	https://www.zomato.com/bangalore/wafl-caf%C3%A...	G1 Smart Avenue, 28, Residency Road, Bangalore	WAFL CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©	Yes	No	4.0	85	
5872	https://www.zomato.com/bangalore/yauatcha-p%C3...	Level 5, 1 MG Road Mall, MG Road, Bangalore	Yauatcha PÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ¢tiss...	Yes	No	3.9	53	
5916	https://www.zomato.com/bangalore/le-ch%C3%A9ri...	24, Nanjappa Road, Shanti Nagar, Bangalore	Le ChÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©ri	No	No	3.8	33	

	url	address	name	online_order	book_table	rate	votes	phon
6044	https://www.zomato.com/bangalore/kaz%C3%A9-1-l...	909 SKAV, 21st floor, Lavelle Road, Bangalore	KazÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ©	No	Yes	4.4	858	
6092	https://www.zomato.com/bangalore/caf%C3%A9-fel...	Level 5, 1 MG Mall, Swami Vivekananda Road, Tr...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© Felix	Yes	Yes	4.4	737	
6152	https://www.zomato.com/bangalore/caf%C3%A9-mor...	Le Parc Richmonde Building, Shantala nagar, Ri...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© Mor'ish	Yes	No	4.1	114	080

Checking for some more garbage values in the rest_name column

In [11]:

```
df_zomato[df_zomato['name'].str.startswith("Caf")].loc[:].head(5)
```

Out[11]:

	url	address	name	online_order	book_table	rate	votes	phon
10	https://www.zomato.com/bangalore/caf%C3%A9-dow...	12,29 Near PES University Back Gate, D'Souza N...	CafÃÂÃÂÃÂÃÂÃÂÃÂÃÂÃÂ© Down The A...	Yes	No	4.1	402	08 26724489\r\n+9 740604898
11	https://www.zomato.com/bangalore/cafe-shuffle-...	941, 3rd FLOOR, 21st Main, Banasha...	Cafe Shuffle	Yes	Yes	4.2	150	+91 974216677
13	https://www.zomato.com/bangalore/caf-eleven-ba...	111, Sapphire Toys Building, 100 Feet Ring Roa...	Caf-Eleven	No	No	4.0	424	080 4957771
15	https://www.zomato.com/bangalore/cafe-vivacity...	2303, 21st Cross, K R Road, 2nd Stage, Banasha...	Cafe Vivacity	Yes	No	3.8	90	08 26768182\r\n+9 984570445
22	https://www.zomato.com/bangalore/cafe-coffee-d...	SRF Complex, Near BDA Complex, Kathreguppe Mai...	Cafe Coffee Day	No	No	3.6	28	080 3248629

In [12]:

```
## Removing the garbage values
import re
```

In [13]:

```
def clean_restaurant_name(name):
    cleaned_name = re.sub('[^A-Za-z0-9.!`&~@\'\s -]', '', name)
    return cleaned_name.strip()

# Clean the rest_name column
df_zomato['name'] = df_zomato['name'].apply(clean_restaurant_name)
```

In [14]:

```
df_zomato[df_zomato['name'].str.startswith("Caf")].loc[:, ['name']].head(15)
```

Out[14]:

	name
10	Caf Down The Alley
11	Cafe Shuffle
13	Caf-Eleven
15	Cafe Vivacity
22	Cafe Coffee Day
193	Caf Down The Alley
254	Cafe Coffee Day
279	Cafe Vivacity
305	Cafe Aira
312	Cafe Zone
403	Cafe Mondo
460	Cafe Shuffle
482	Cafe Ajfan
566	Caf Down The Alley
567	Cafe Shuffle

Numeric Value columns had the datatype as object, we need to convert it into the numeric format, and the non numeric values must be converted into null values

In [15]:
```
df_zomato['rate'] = pd.to_numeric(df_zomato['rate'], errors="coerce")

df_zomato['approx_cost(for two people)'] = pd.to_numeric(df_zomato['approx_cost(for two people)'], errors="coer
```

In [16]:
```
df_zomato.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51717 entries, 0 to 51716
Data columns (total 17 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   url                          51717 non-null  object 
 1   address                      51717 non-null  object 
 2   name                         51717 non-null  object 
 3   online_order                 51717 non-null  object 
 4   book_table                   51717 non-null  object 
 5   rate                         41665 non-null  float64
 6   votes                        51717 non-null  int64  
 7   phone                        50509 non-null  object 
 8   location                     51696 non-null  object 
 9   rest_type                    51490 non-null  object 
 10  dish_liked                   23639 non-null  object 
 11  cuisines                     51672 non-null  object 
 12  approx_cost(for two people)  44454 non-null  float64
 13  reviews_list                 44122 non-null  object 
 14  menu_item                    12100 non-null  object 
 15  listed_in(type)              51717 non-null  object 
 16  listed_in(city)              51717 non-null  object 
dtypes: float64(2), int64(1), object(14)
memory usage: 6.7+ MB
```

In [17]:
```
df_zomato.describe()
```

Out[17]:

	rate	votes	approx_cost(for two people)
count	41665.000000	51717.000000	44454.000000
mean	3.700449	283.697527	416.630112
std	0.440513	803.838853	194.614442
min	1.800000	0.000000	40.000000
25%	3.400000	7.000000	300.000000
50%	3.700000	41.000000	400.000000
75%	4.000000	198.000000	550.000000
max	4.900000	16832.000000	950.000000

Checking for Other missing values in non numeric columns

In [18]:
```
df_zomato.isnull().sum()
```

Out[18]:
```
url                                0
address                            0
name                               0
online_order                       0
book_table                         0
rate                           10052
votes                              0
phone                           1208
location                          21
rest_type                        227
dish_liked                     28078
cuisines                          45
approx_cost(for two people)     7263
reviews_list                    7595
menu_item                      39617
listed_in(type)                    0
listed_in(city)                    0
dtype: int64
```

In [19]:
```
df_zomato.isnull().sum()/df_zomato.shape[0]*100
```

Out[19]:
```
url                             0.000000
address                         0.000000
name                            0.000000
online_order                    0.000000
book_table                      0.000000
rate                           19.436549
votes                           0.000000
phone                           2.335789
location                        0.040606
rest_type                       0.438927
dish_liked                     54.291626
cuisines                        0.087012
approx_cost(for two people)    14.043738
reviews_list                   14.685693
menu_item                      76.603438
listed_in(type)                 0.000000
listed_in(city)                 0.000000
dtype: float64
```

In [20]:
```
df_zomato.drop('dish_liked',axis=1,inplace=True)
df_zomato.drop('menu_item',axis=1,inplace=True)
```

In [21]:
```
df_zomato['reviews_list']  =  df_zomato.groupby('name')['reviews_list'].fillna(method='ffill')
```

In [22]:
```
df_zomato['reviews_list']  =  df_zomato.groupby('name')['reviews_list'].fillna(method='bfill')
```

In [23]:
```
df_zomato['reviews_list'] = df_zomato.groupby('cuisines')['reviews_list'].fillna(method='ffill')
```

In [24]:
```
df_zomato['reviews_list']  =  df_zomato.groupby('cuisines')['reviews_list'].fillna(method='bfill')
```

In [25]:
```
df_zomato['reviews_list']  =  df_zomato.groupby('rate')['reviews_list'].fillna(method='ffill')
```

In [26]:
```
df_zomato.isnull().sum()/df_zomato.shape[0]*100
```

Out[26]:
```
url                             0.000000
address                         0.000000
name                            0.000000
online_order                    0.000000
book_table                      0.000000
rate                           19.436549
votes                           0.000000
phone                           2.335789
location                        0.040606
rest_type                       0.438927
cuisines                        0.087012
approx_cost(for two people)    14.043738
reviews_list                   19.436549
listed_in(type)                 0.000000
listed_in(city)                 0.000000
dtype: float64
```

In [27]:
```
df_zomato['approx_cost(for  two  people)'].fillna(df_zomato['approx_cost(for  two  people)'].median(),  inplace=True
```

In [28]:
```
df_zomato['rate'].fillna(df_zomato['rate'].median(), inplace=True)
```

In [29]:
```
df_zomato.isnull().sum()
```

Out[29]:
```
url                                0
address                            0
name                               0
online_order                       0
book_table                         0
rate                               0
votes                              0
phone                           1208
location                          21
rest_type                        227
cuisines                          45
approx_cost(for two people)        0
reviews_list                   10052
listed_in(type)                    0
listed_in(city)                    0
dtype: int64
```

In [30]:
```
df_zomato['rest_type']  =  df_zomato.groupby('name')['rest_type'].fillna(method='ffill')
```

In [31]:
```
df_zomato['rest_type']  =  df_zomato.groupby('name')['rest_type'].fillna(method='bfill')
```

In [32]:
```
df_zomato['rest_type']  =  df_zomato.groupby('cuisines')['rest_type'].fillna(method='ffill')
```

In [33]:
```
df_zomato['rest_type']  =  df_zomato.groupby('cuisines')['rest_type'].fillna(method='bfill')
```

In [34]:
```
df_zomato.isnull().sum()
```

Out[34]:
```
url                                0
address                            0
name                               0
online_order                       0
book_table                         0
rate                               0
votes                              0
phone                           1208
location                          21
rest_type                         72
cuisines                          45
approx_cost(for two people)        0
reviews_list                   10052
listed_in(type)                    0
listed_in(city)                    0
dtype: int64
```

Out[35]:
```
df_zomato.head()
```

	url	address	name	online_order	book_table	rate	votes	phone	location
0	https://www.zomato.com/bangalore/jalsa-banasha...	942, 21st Main Road, 2nd Stage, Banashankari, ...	Jalsa	Yes	Yes	4.1	775	080 42297555\r\n+91 9743772233	Banashankari
1	https://www.zomato.com/bangalore/spice-elephan...	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	No	4.1	787	080 41714161	Banashankari
2	https://www.zomato.com/SanchurroBangalore?cont...	1112, Next to KIMS Medical College, 17th Cross...	San Churro	Yes	No	3.8	918	+91 9663487993	Banashankari
3	https://www.zomato.com/bangalore/addhuri-Banashankar...	1st Floor, Annakuteera,	Addhuri Udupi	No	No	3.7	88	+91 9620009302	Banashankari
4	https://www.zomato.com/bangalore/grand-village...	10, 3rd Floor, Lakshmi Associates, Gandhi Baza...	Grand Village	No	No	3.8	166	+91 8026612447\r\n+91 9901210005	Basavanagudi

In [36]:
```
## Encoding the categorical data
from sklearn.preprocessing import LabelEncoder  # Import LabelEncoder

categorical_columns = df_zomato.select_dtypes(include=['object']).columns.tolist()
label_encoder = LabelEncoder()

for column in categorical_columns:
    df_zomato[column] = label_encoder.fit_transform(df_zomato[column])
```

In [37]:
```
df_zomato.head()
```

Out[37]:

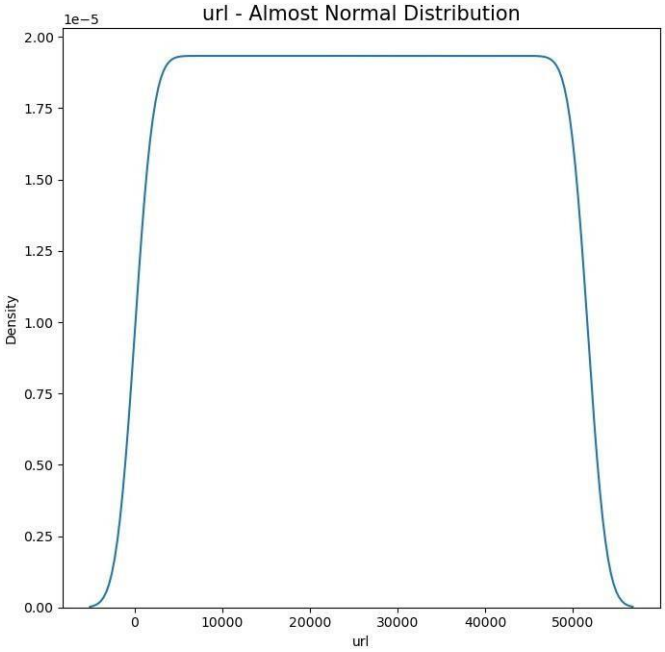
	url	address	name	online_order	book_table	rate	votes	phone	location	rest_type	cuisines	approx_cost(for two people)	reviews_list	listed_in(
0	22195	8016	3673	1	1	4.1	775	13293	1	27	2159	800.0	10313	
1	41273	3844	6995	1	0	4.1	787	13078	1	27	952	800.0	11953	
2	92	784	6473	1	0	3.8	918	7145	1	22	766	800.0	4345	
3	1160	2515	195	0	0	3.7	88	6770	1	78	2555	300.0	8805	
4	18293	340	2923	0	0	3.8	166	2006	4	27	2188	600.0	14812	

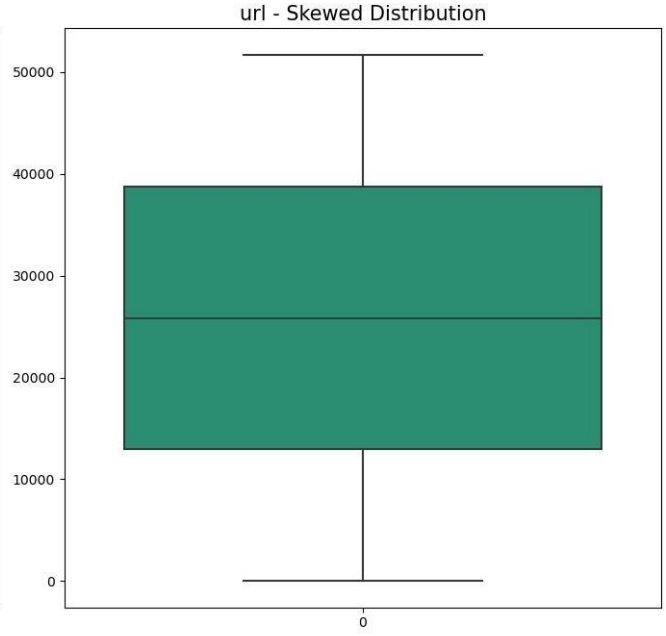
In [38]:
```
df_zomato.info()
```

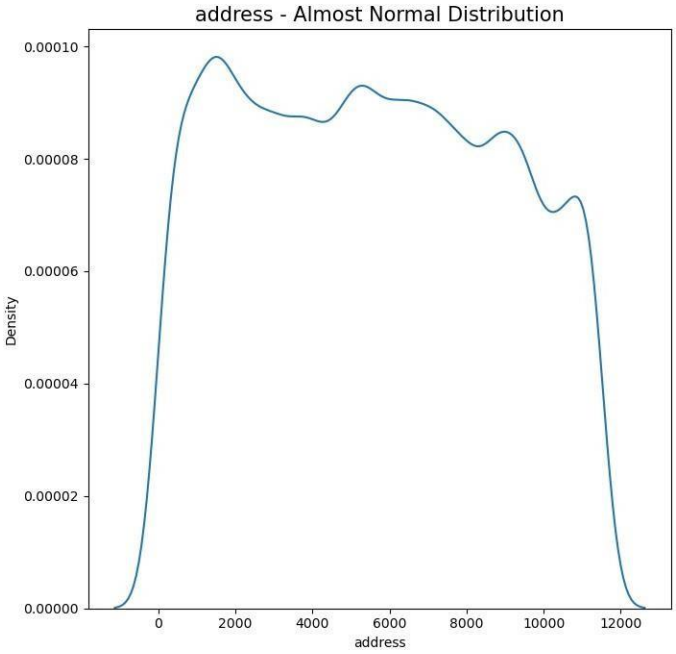
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51717 entries, 0 to 51716
Data columns (total 15 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   url                          51717 non-null  int32  
 1   address                      51717 non-null  int32  
 2   name                         51717 non-null  int32  
 3   online_order                 51717 non-null  int32  
 4   book_table                   51717 non-null  int32  
 5   rate                         51717 non-null  float64
 6   votes                        51717 non-null  int64  
 7   phone                        51717 non-null  int32  
 8   location                     51717 non-null  int32  
 9   rest_type                    51717 non-null  int32  
 10  cuisines                     51717 non-null  int32  
 11  approx_cost(for two people)  51717 non-null  float64
 12  reviews_list                 51717 non-null  int32  
 13  listed_in(type)              51717 non-null  int32  
 14  listed_in(city)              51717 non-null  int32  
dtypes: float64(2), int32(12), int64(1)
memory usage: 3.6 MB
```

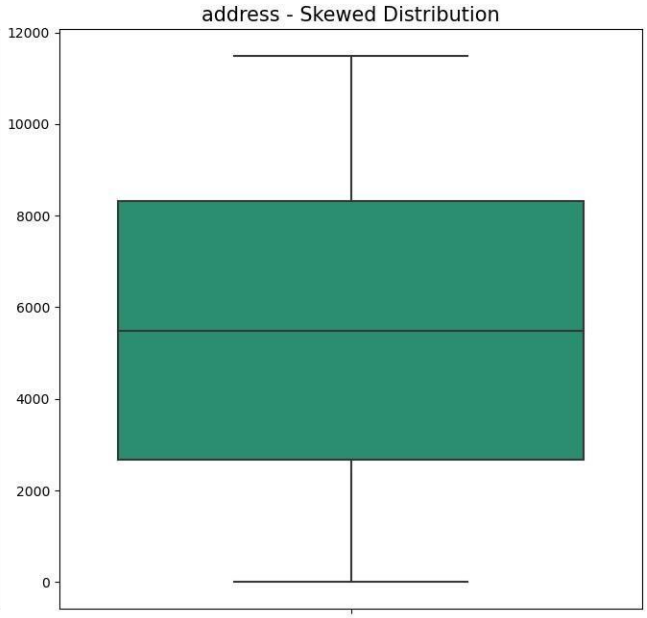
Check Distibution and Outlier

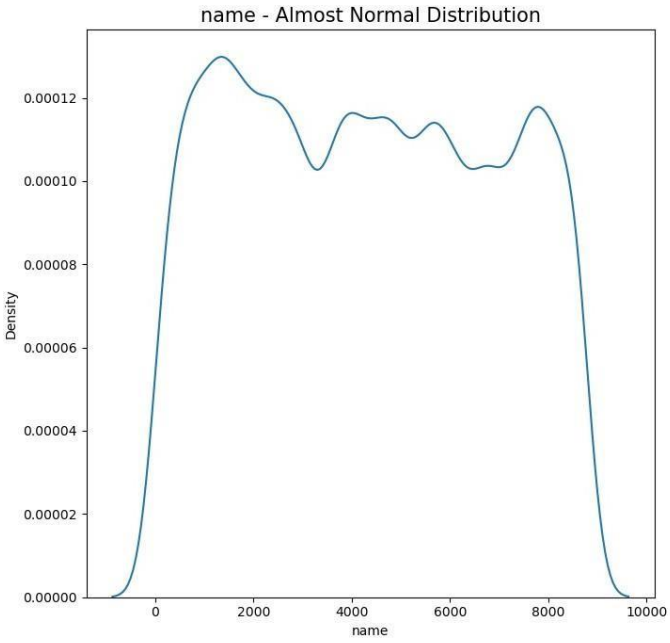
In [39]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame

# Loop through all columns
for column in df_zomato.columns:
    plt.figure(figsize=(14, 7))

    # Subplot 1: Kernel Density Estimation Plot
    plt.subplot(1, 2, 1)
    plt.title(f"{column} - Almost Normal Distribution", fontsize=15)
    sns.kdeplot(data=df_zomato[column])

    # Subplot 2: Box Plot
    plt.subplot(1, 2, 2)
    plt.title(f"{column} - Skewed Distribution", fontsize=15)
    sns.boxplot(data=df_zomato[column], palette="Dark2")

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

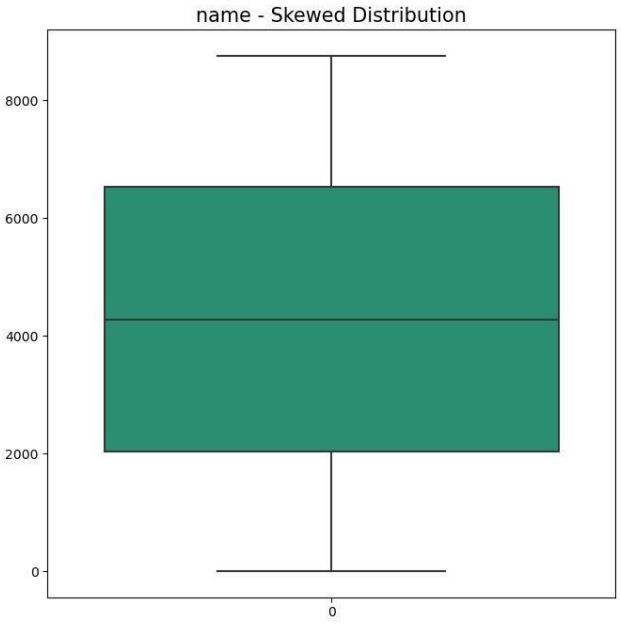

url - Almost Normal Distribution
1e−5
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Density
0
10000
20000
30000
40000
50000
url

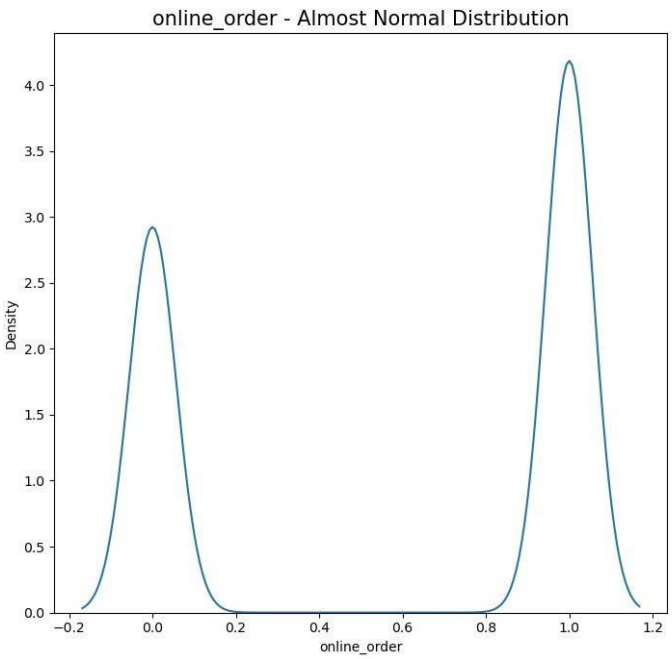

url - Skewed Distribution
50000
40000
30000
20000
10000
0
0

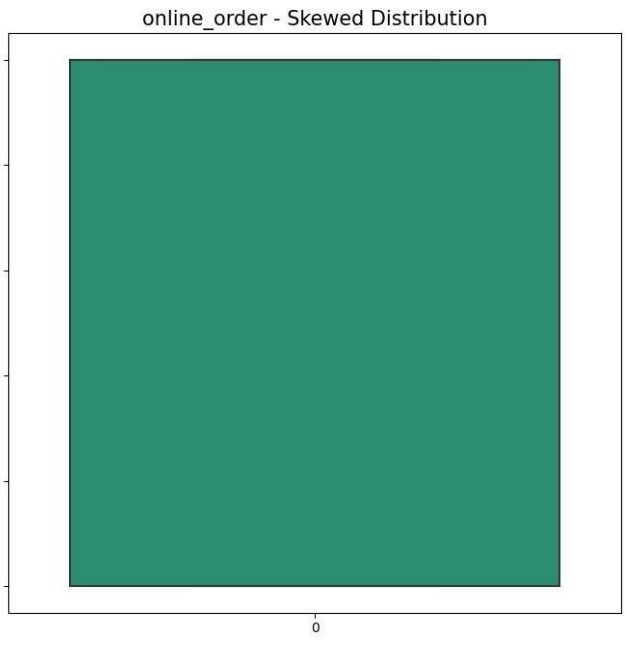

address - Almost Normal Distribution
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
Density
0
2000
4000
6000
8000
10000
12000
address

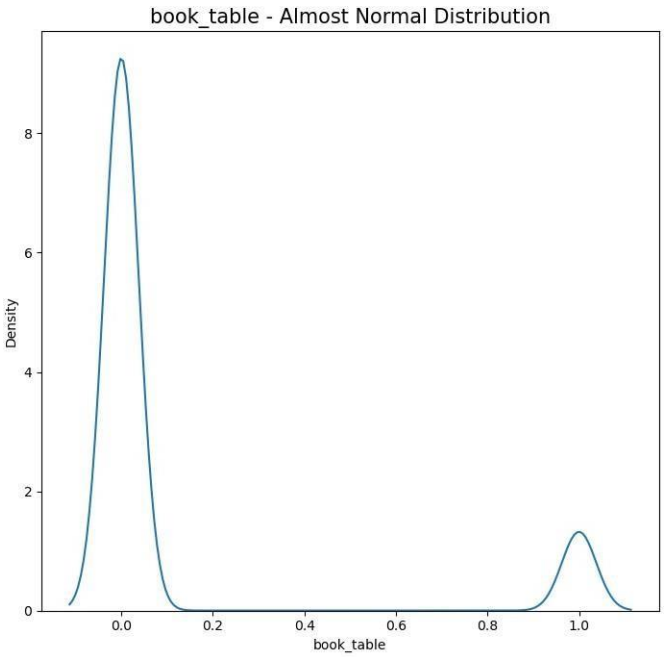

address - Skewed Distribution
12000
10000
8000
6000
4000
2000
0
0

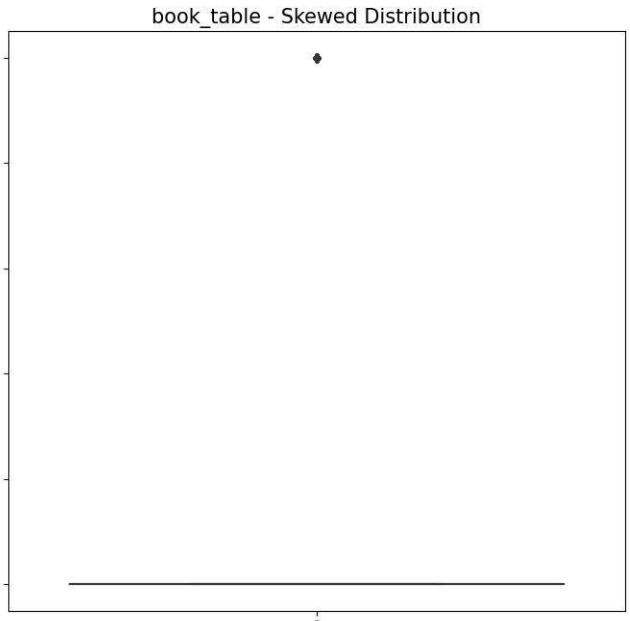

name - Almost Normal Distribution
0.00012
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
Density
0
2000
4000
6000
8000
10000
name

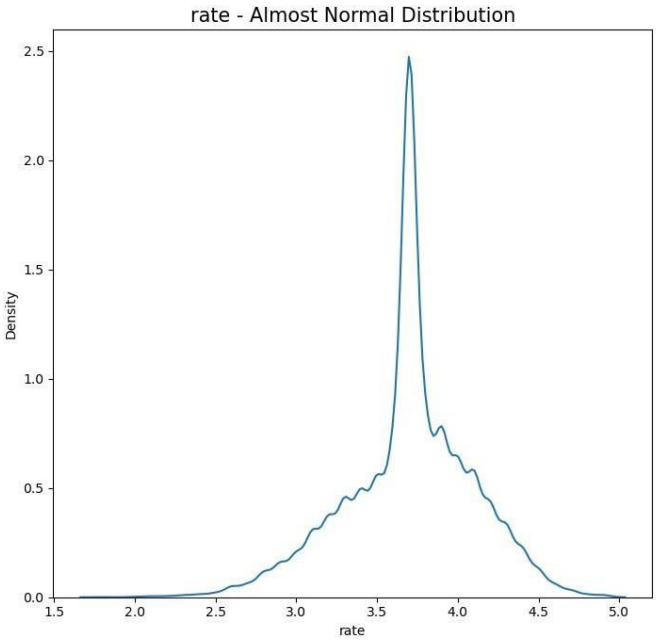

name - Skewed Distribution
8000
6000
4000
2000
0
0

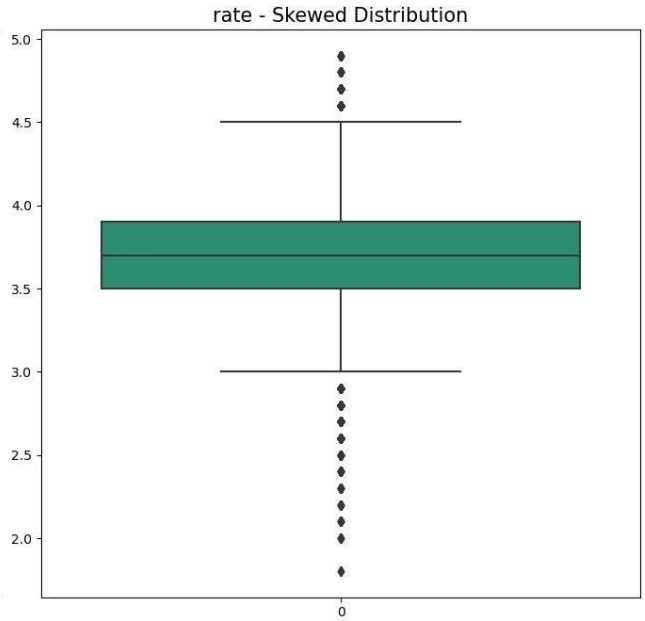

online_order - Almost Normal Distribution
Density
online_order

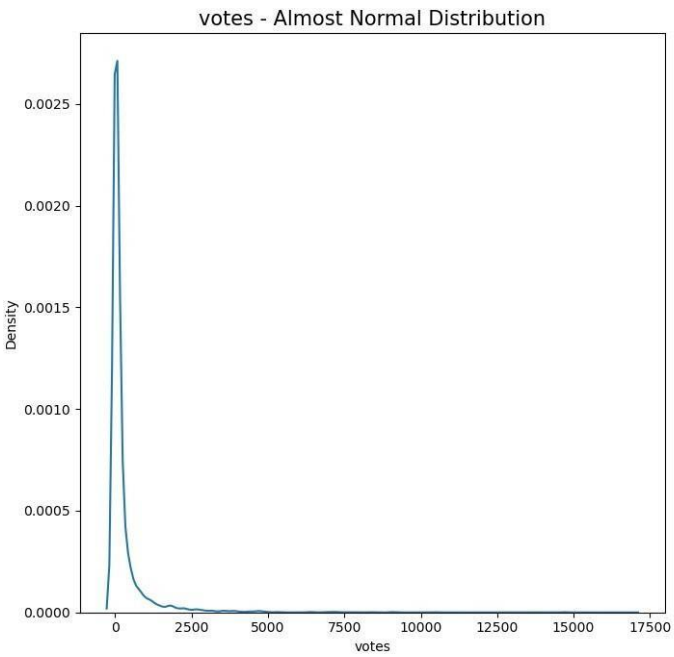

online_order - Skewed Distribution
0

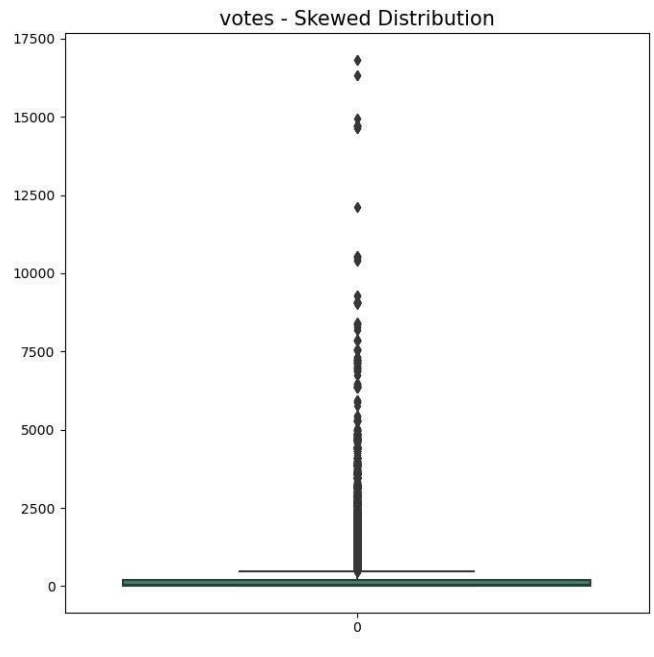

book_table - Almost Normal Distribution
Density
book_table

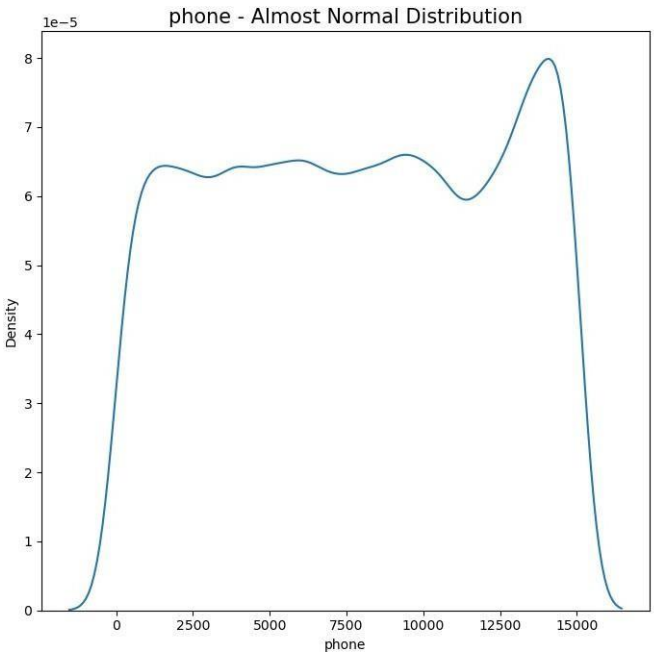

book_table - Skewed Distribution
0

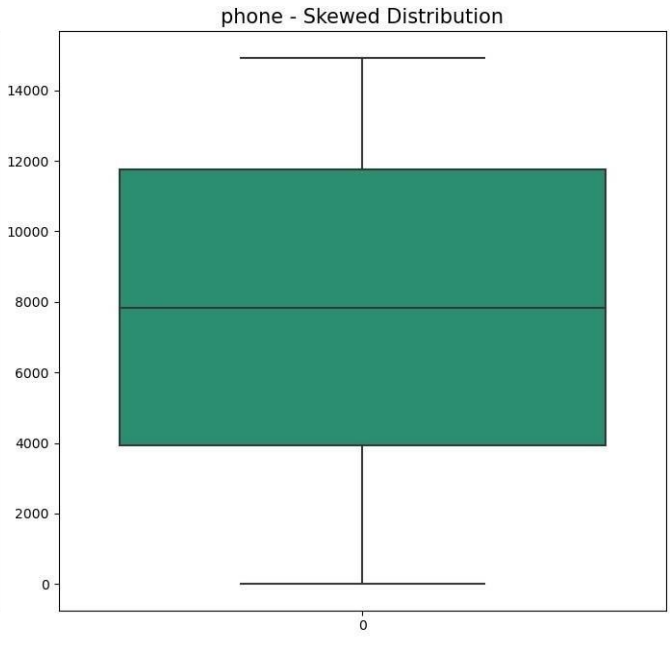

rate - Almost Normal Distribution
Density
rate

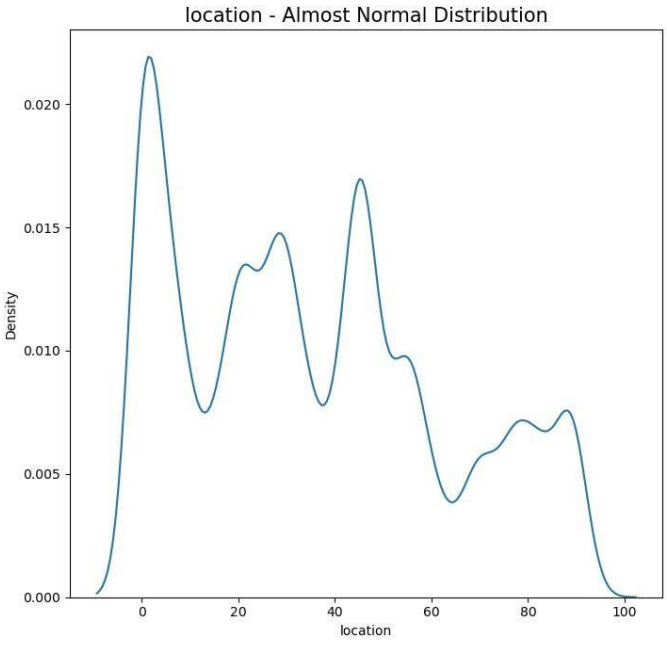

rate - Skewed Distribution
0

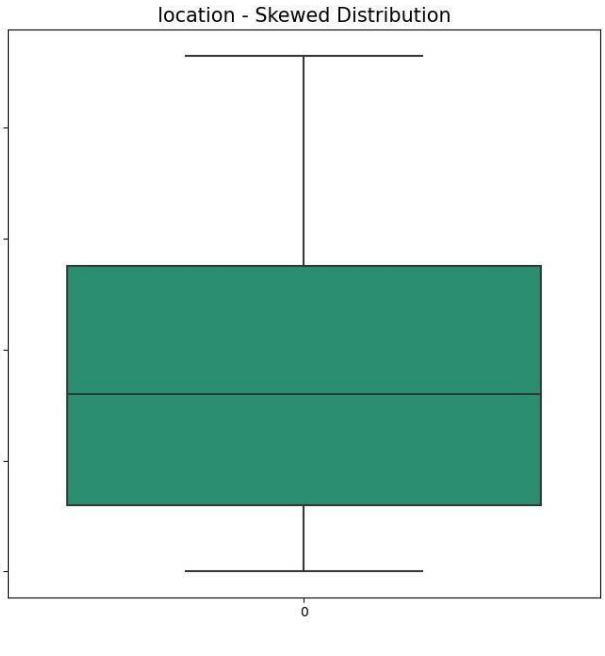

votes - Almost Normal Distribution
Density
votes
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
2500
5000
7500
10000
12500
15000
17500

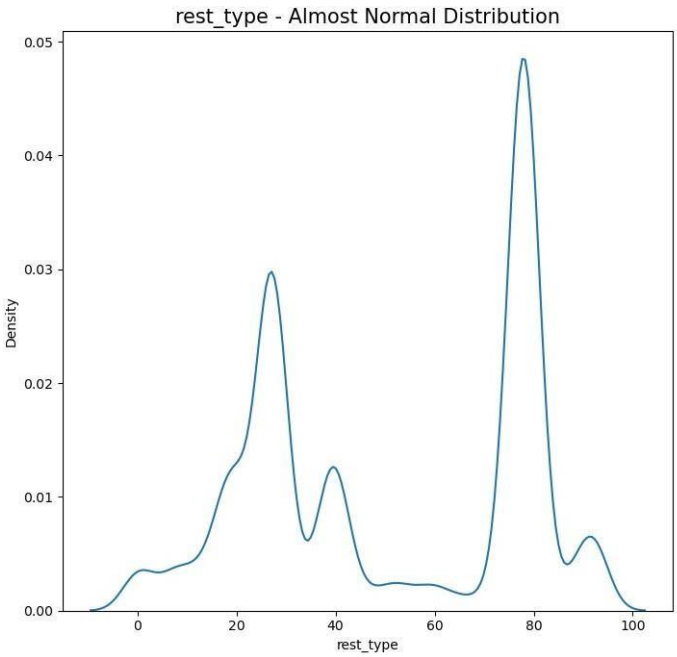

votes - Skewed Distribution
17500
15000
12500
10000
7500
5000
2500
0
0

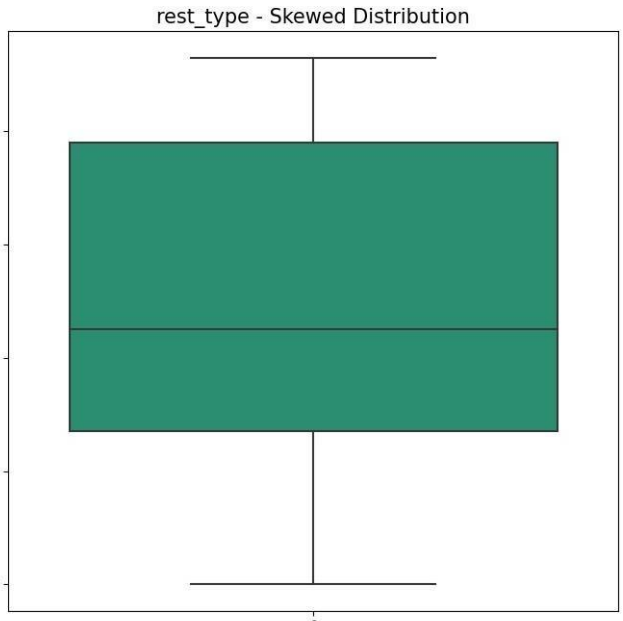

phone - Almost Normal Distribution
1e−5
Density
phone
8
7
6
5
4
3
2
1
0
0
2500
5000
7500
10000
12500
15000

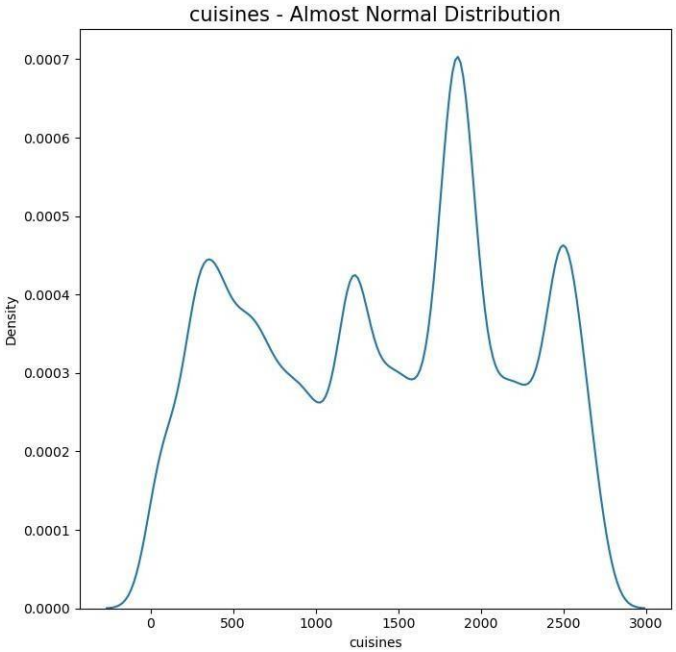

phone - Skewed Distribution
14000
12000
10000
8000
6000
4000
2000
0
0

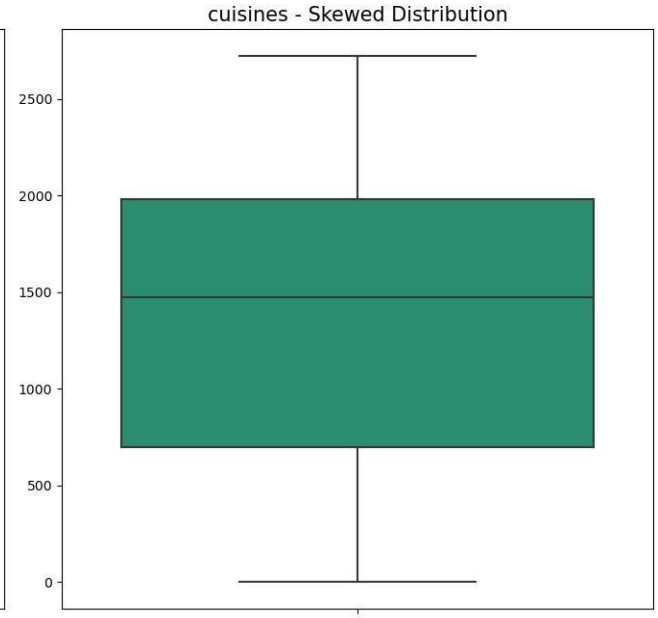

location - Almost Normal Distribution
Density
location
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100

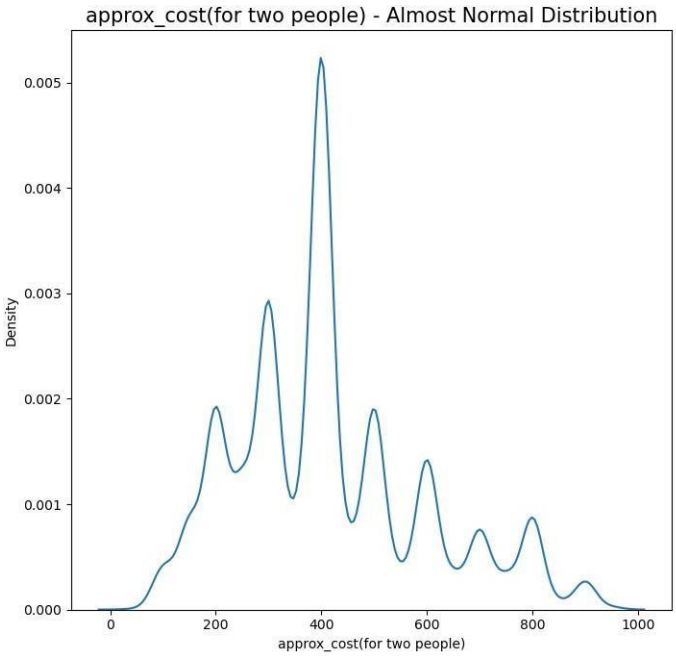

location - Skewed Distribution
80
60
40
20
0
0

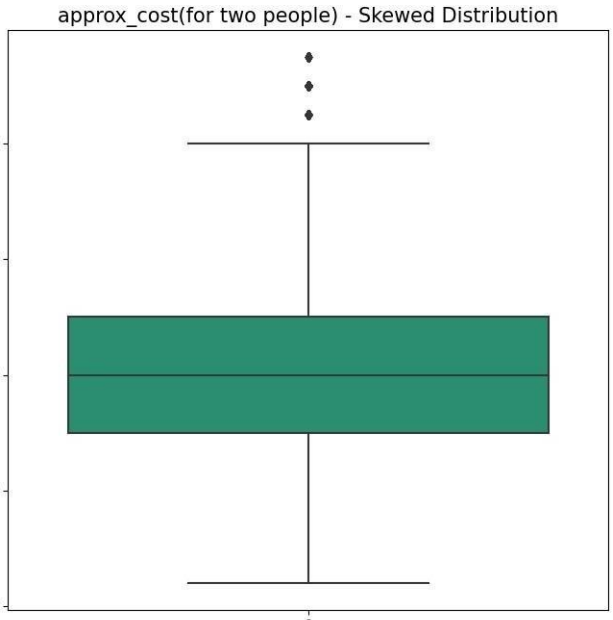

rest_type - Almost Normal Distribution
Density
rest_type

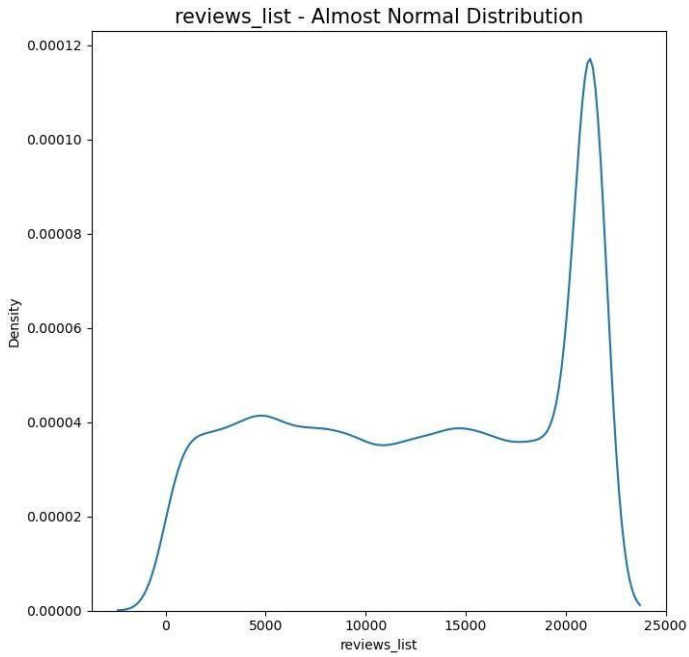

rest_type - Skewed Distribution

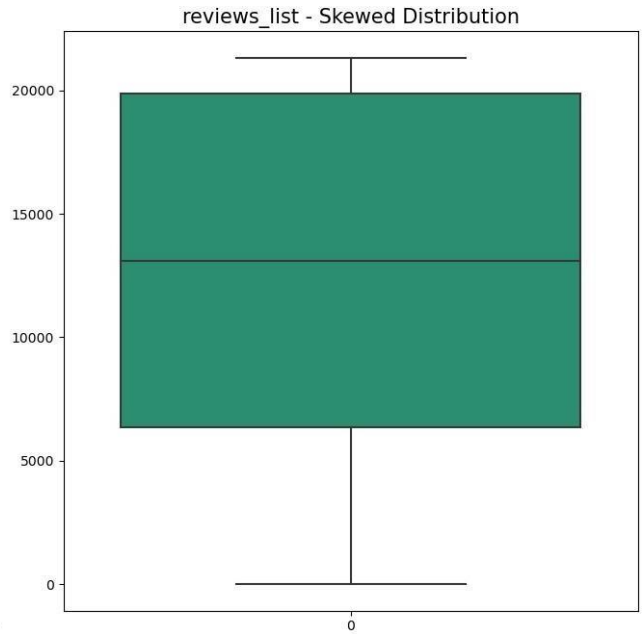

cuisines - Almost Normal Distribution
Density
cuisines

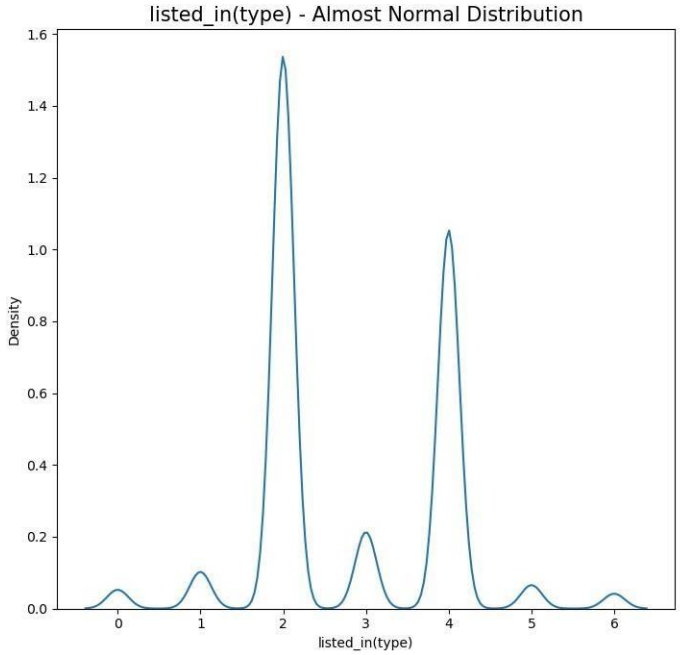

cuisines - Skewed Distribution

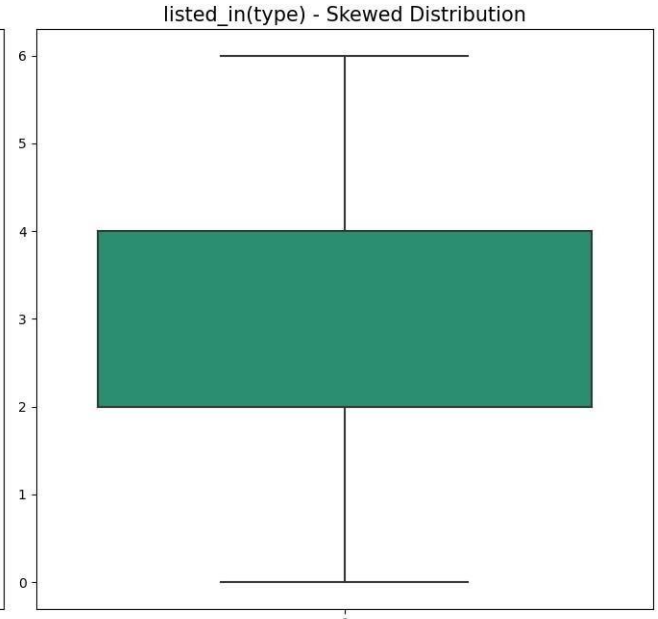

approx_cost(for two people) - Almost Normal Distribution
Density
approx_cost(for two people)

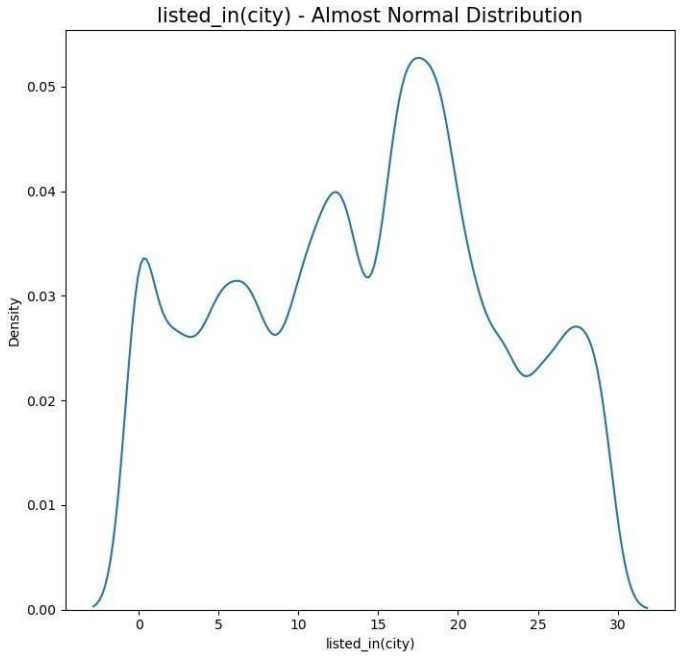

approx_cost(for two people) - Skewed Distribution

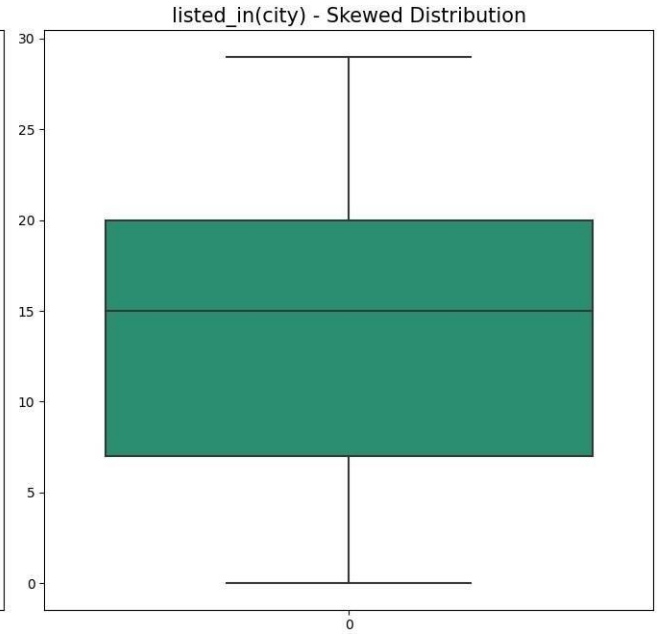




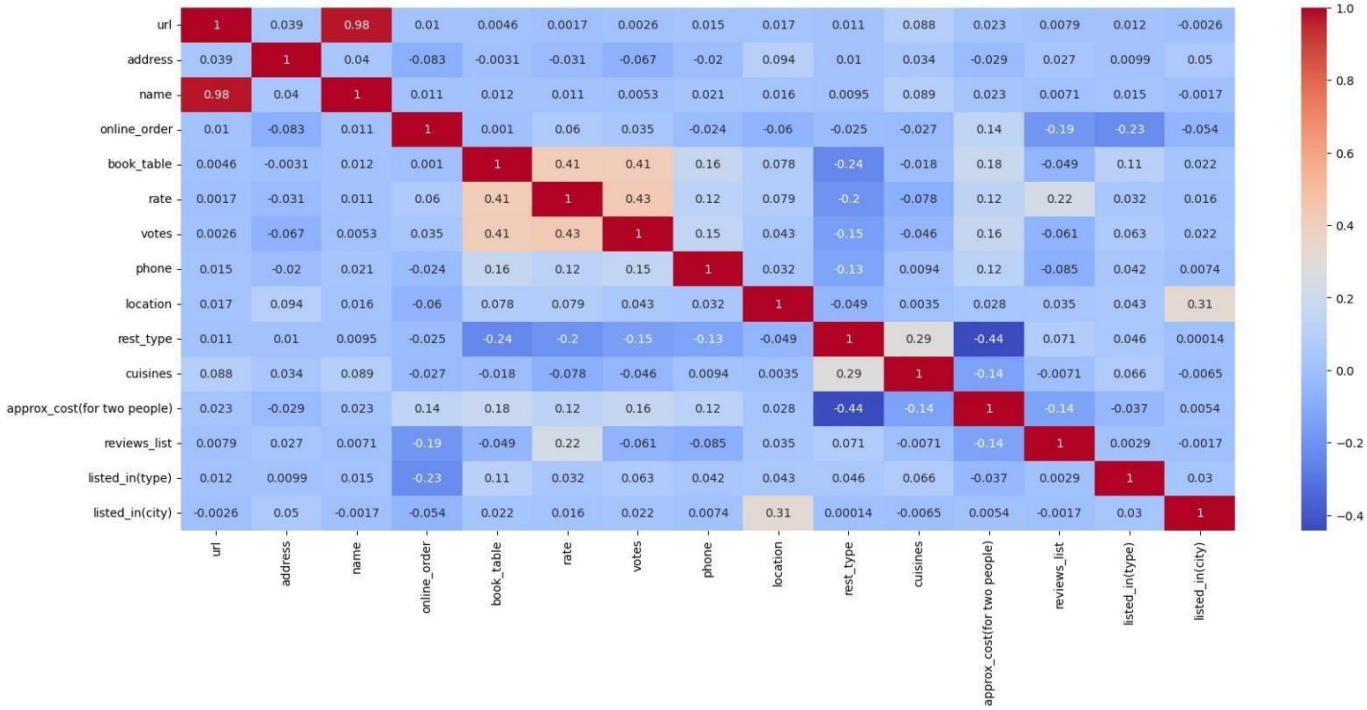








In [40]:
```
plt.figure(figsize=(20,8))
sns.heatmap(data=df_zomato.corr(),  annot=True,  cmap='coolwarm');
```

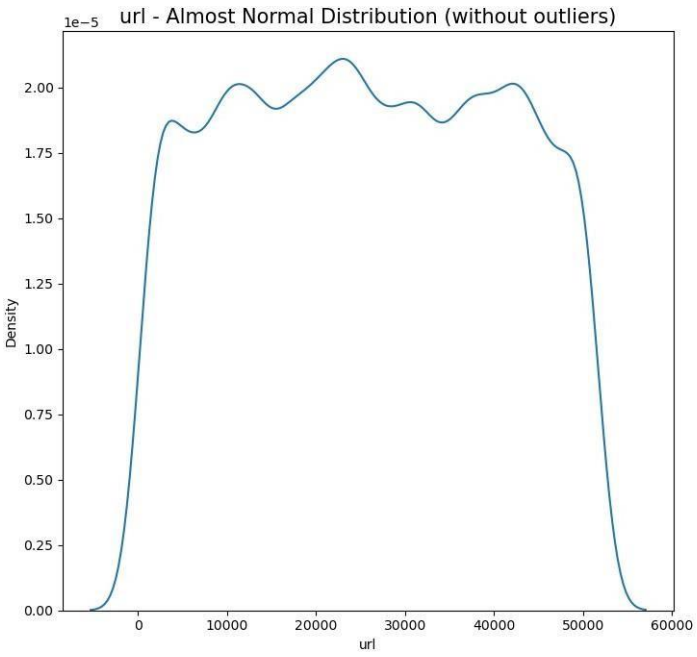


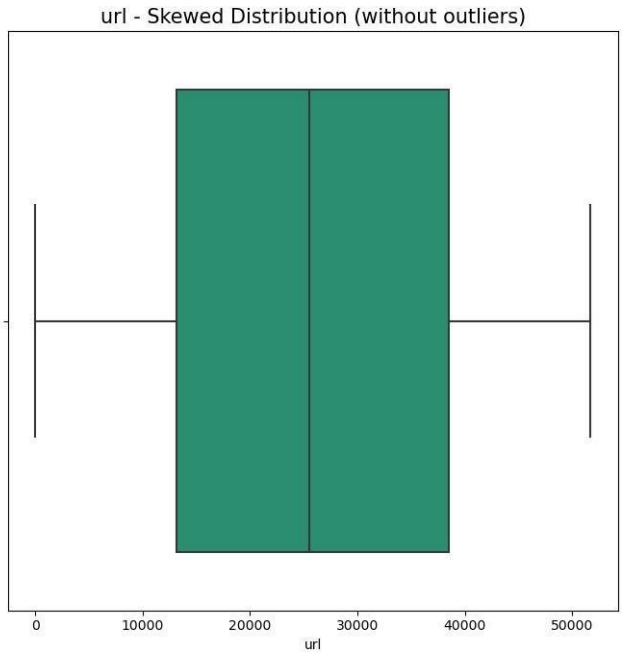
In [41]:
```
df_zomato.columns.tolist()
```

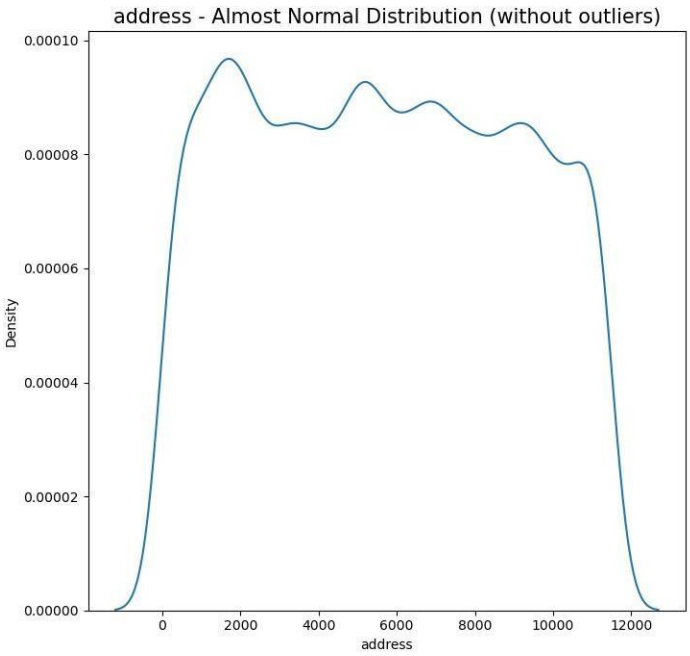
Out[41]:
```
['url',
 'address',
 'name',
 'online_order',
 'book_table',
 'rate',
 'votes',
 'phone',
 'location',
 'rest_type',
 'cuisines',
 'approx_cost(for two people)',
 'reviews_list',
 'listed_in(type)',
 'listed_in(city)']
```

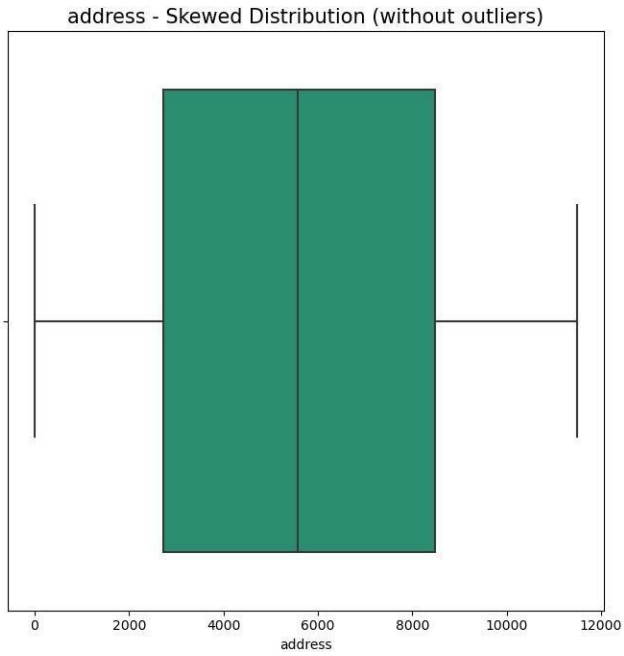
remove outliers & Scaling / fit-Trasformation

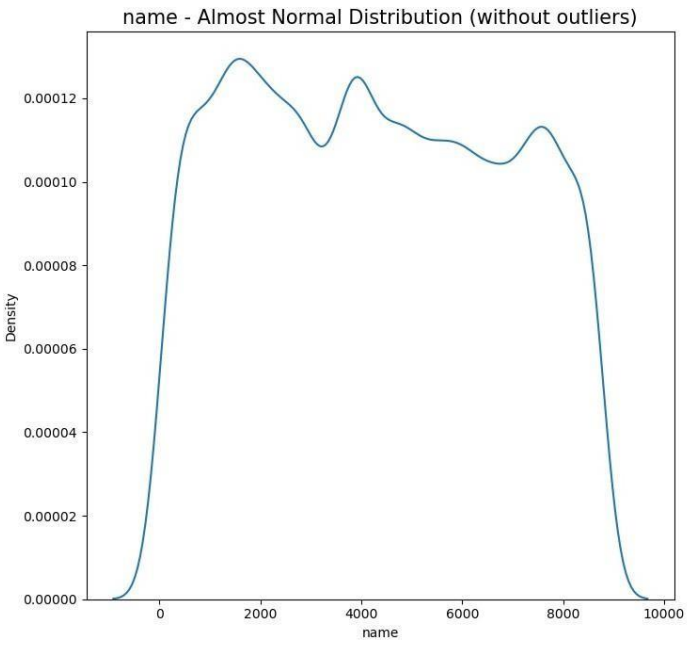
In [42]:
```
# Define a function to remove outliers using IQR for the entire DataFrame
def remove_outliers(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Removes rows where any value in a row falls below the lower bound or above the upper bound using boolean
    return df[~((df < lower_bound) | (df > upper_bound)).any(axis=1)]


# Apply outlier removal function to the entire DataFrame
cleaned_data = remove_outliers(df_zomato)

# Visualize distributions after removing outliers (example with KDE and Boxplot)
for column in cleaned_data.columns:
    plt.figure(figsize=(14, 7))

    # Subplot 1: Kernel Density Estimation Plot (without outliers)
    plt.subplot(1, 2, 1)
    plt.title(f"{column} - Almost Normal Distribution (without outliers)", fontsize=15)
    sns.kdeplot(data=cleaned_data[column])

    # Subplot 2: Box Plot (original column, but with outliers not shown)
    plt.subplot(1, 2, 2)
    plt.title(f"{column} - Skewed Distribution (without outliers)", fontsize=15)
    sns.boxplot(data=cleaned_data, x=column, palette="Dark2", showfliers=False)

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

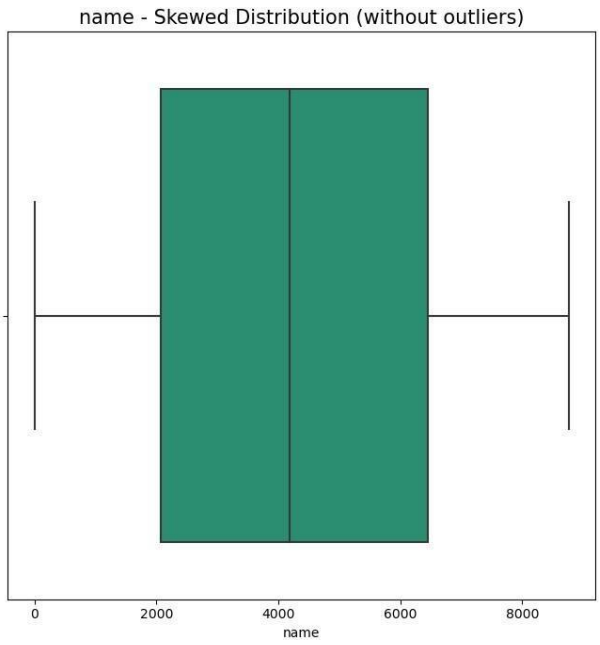

url - Almost Normal Distribution (without outliers)
1e−5
Density
url

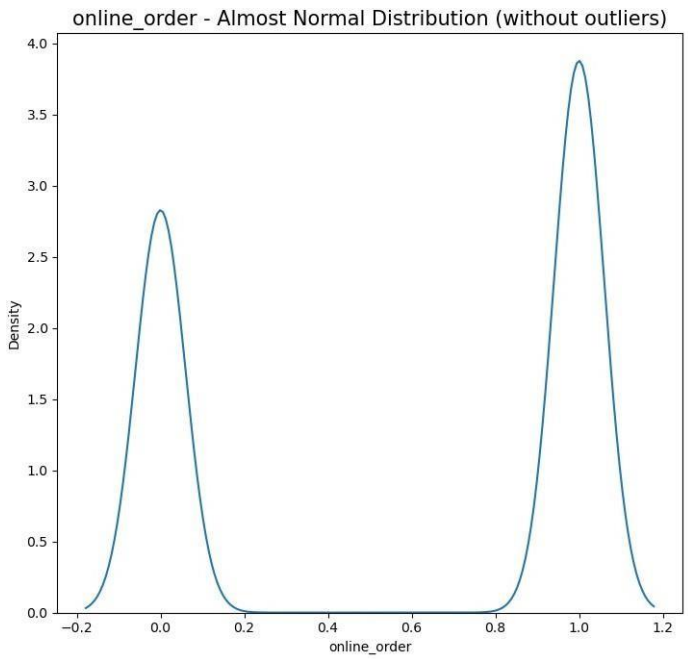

url - Skewed Distribution (without outliers)
url

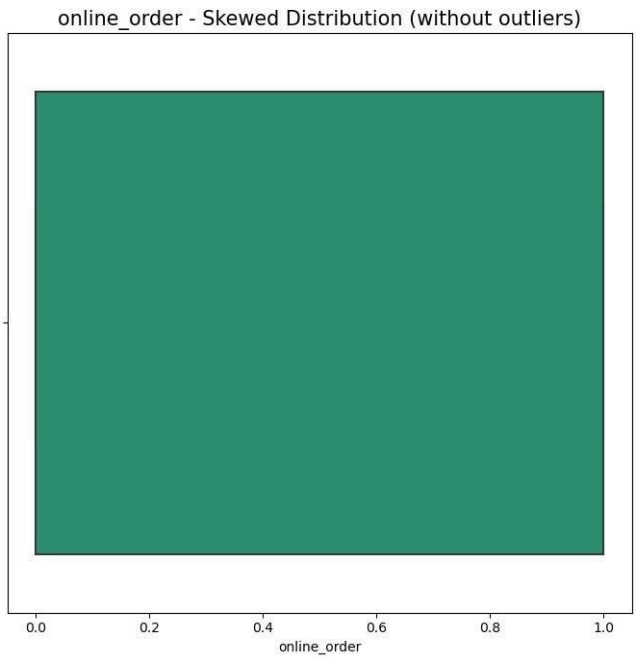

address - Almost Normal Distribution (without outliers)
Density
address

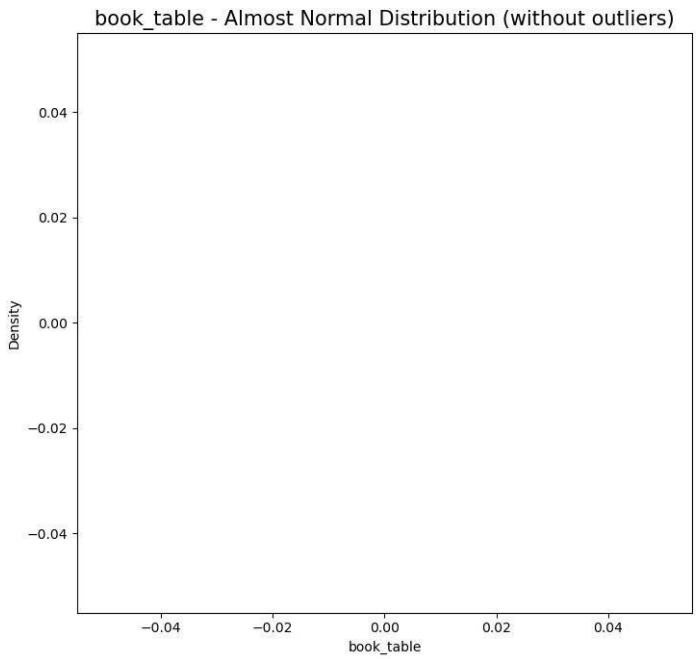

address - Skewed Distribution (without outliers)
address

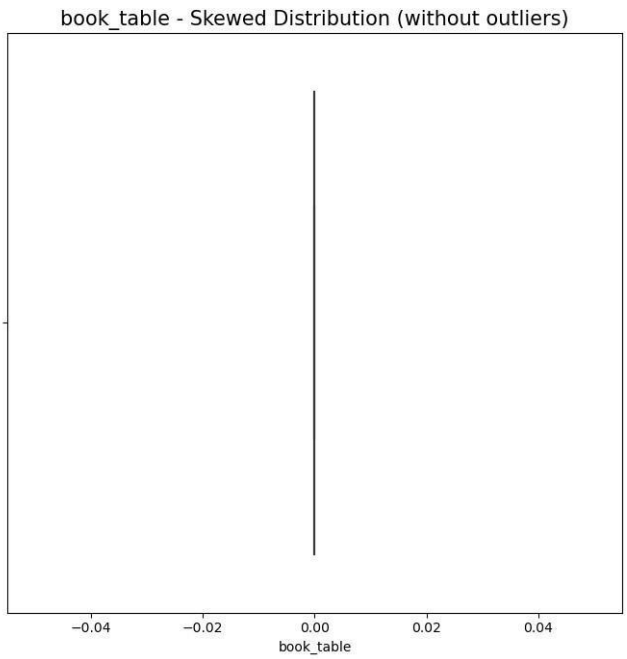

name - Almost Normal Distribution (without outliers)
Density
name

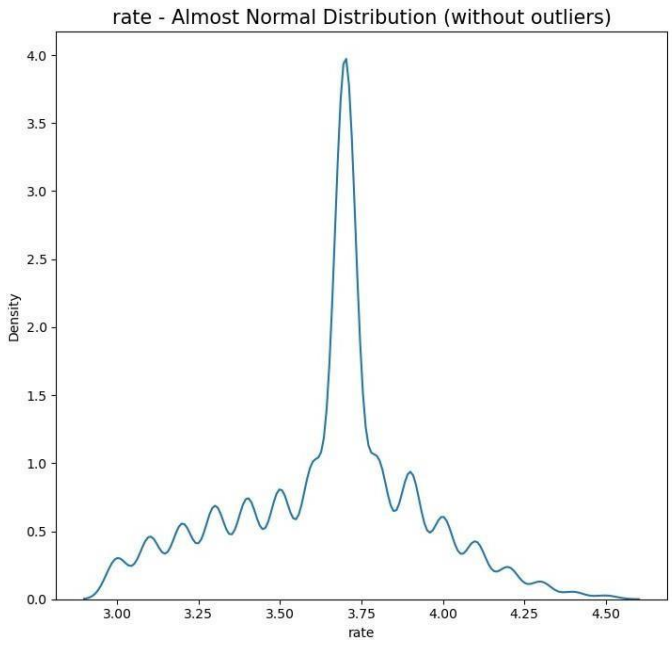

name - Skewed Distribution (without outliers)
name

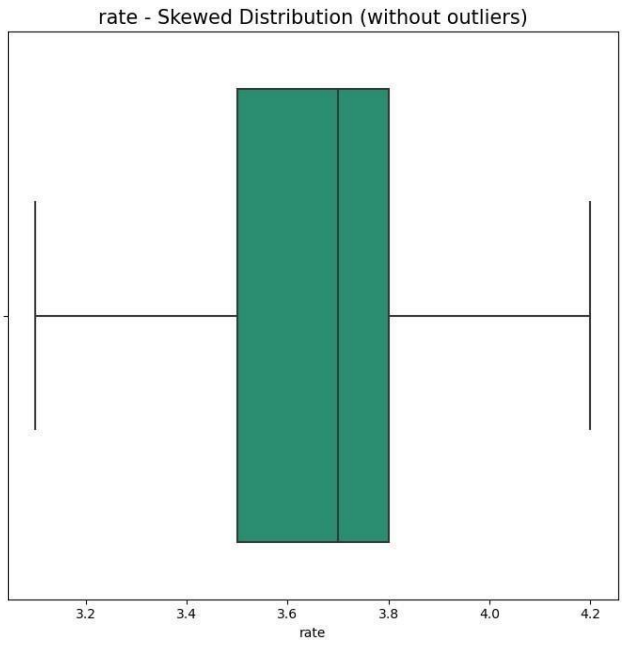

online_order - Almost Normal Distribution (without outliers)
Density
online_order

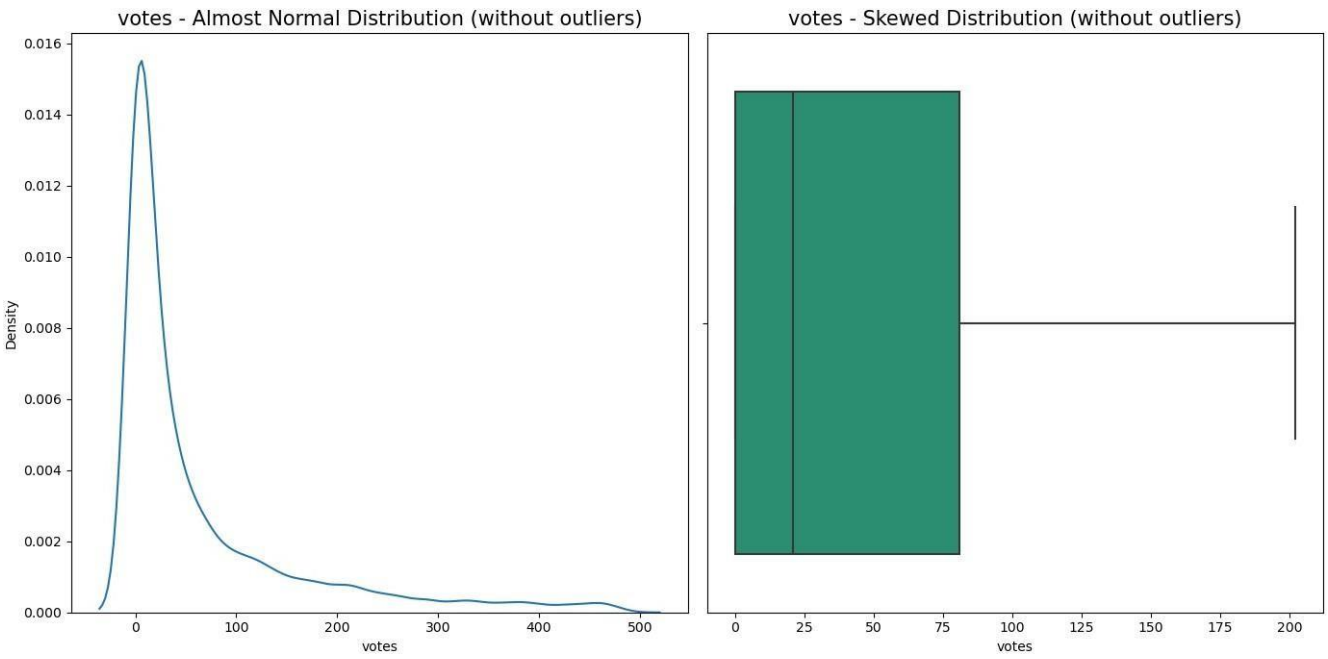

online_order - Skewed Distribution (without outliers)
online_order

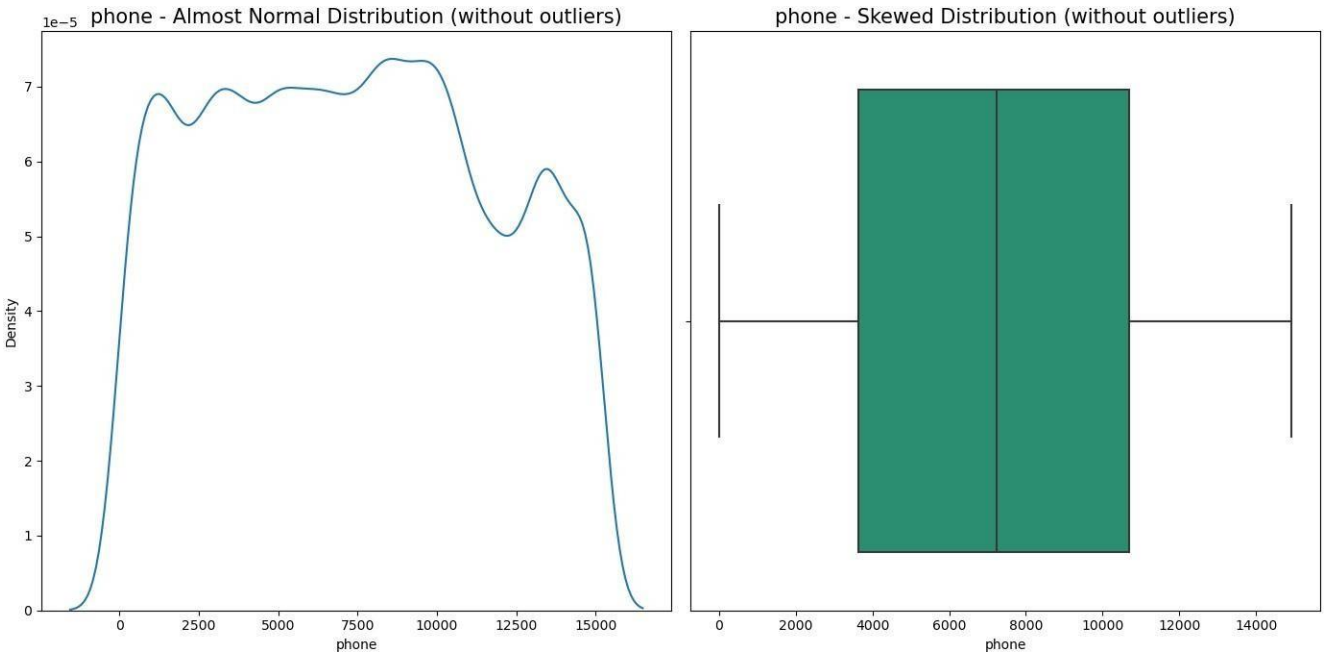

book_table - Almost Normal Distribution (without outliers)
Density
book_table

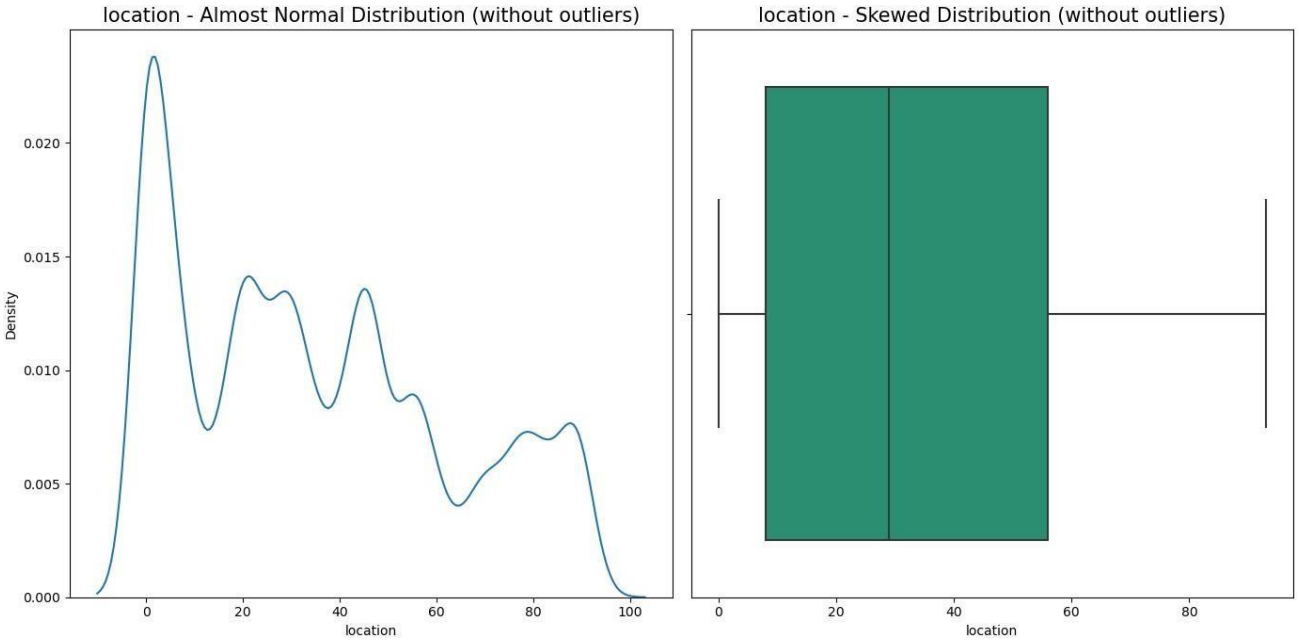

book_table - Skewed Distribution (without outliers)
book_table

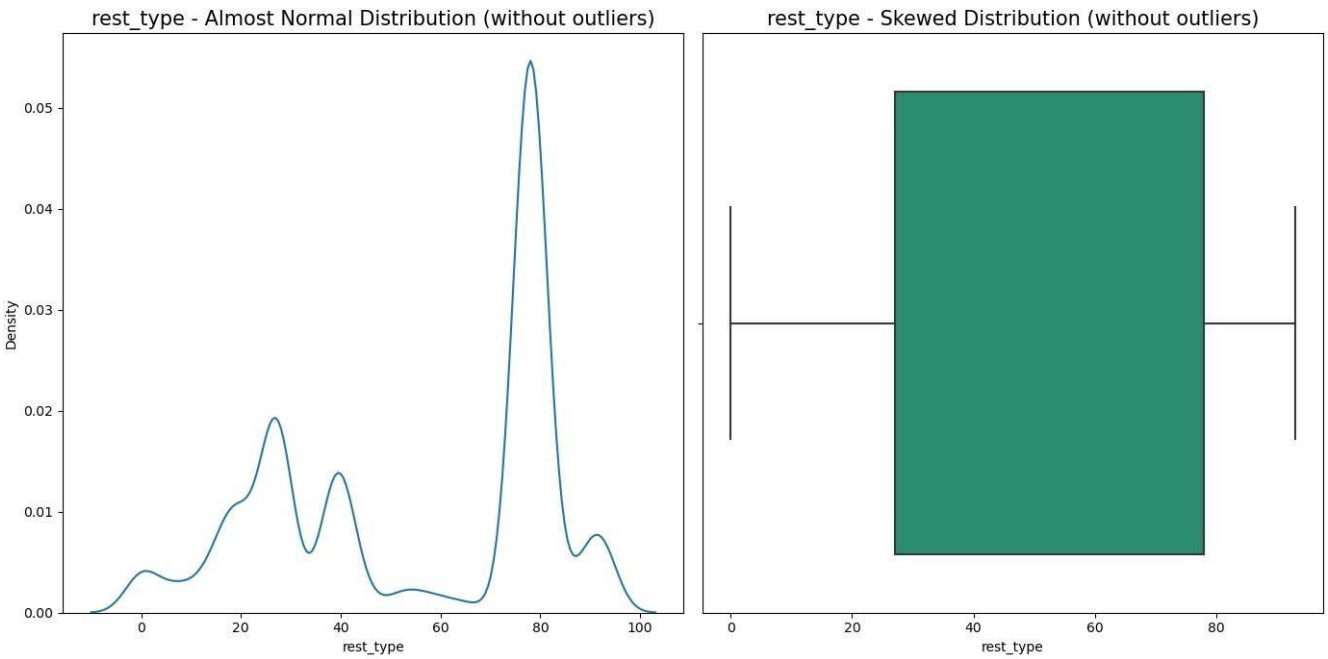

rate - Almost Normal Distribution (without outliers)
Density
rate

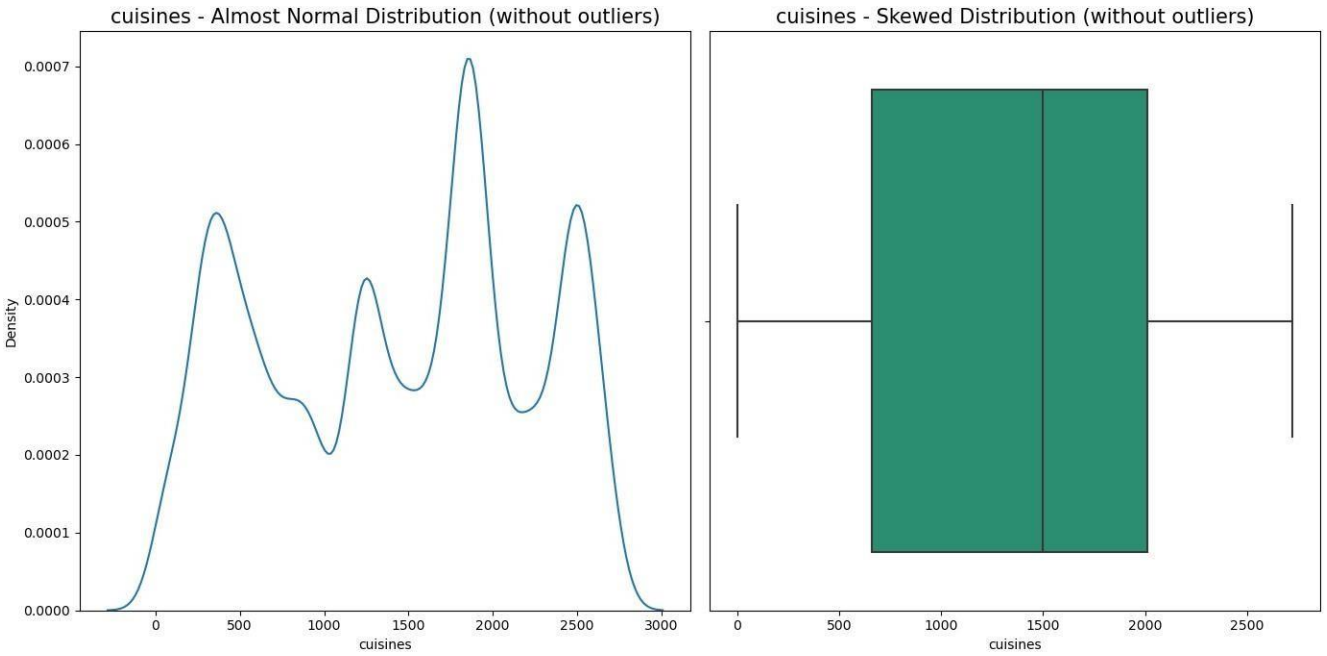

rate - Skewed Distribution (without outliers)
rate

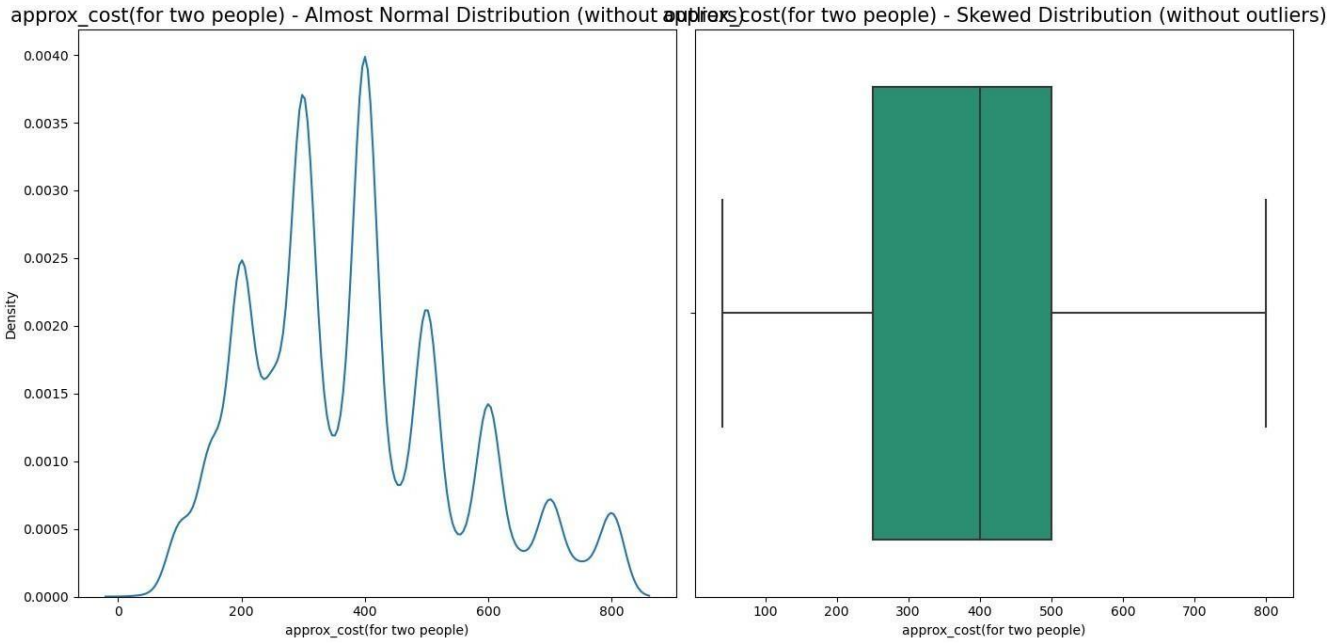

votes - Almost Normal Distribution (without outliers)
votes - Skewed Distribution (without outliers)
Density
votes

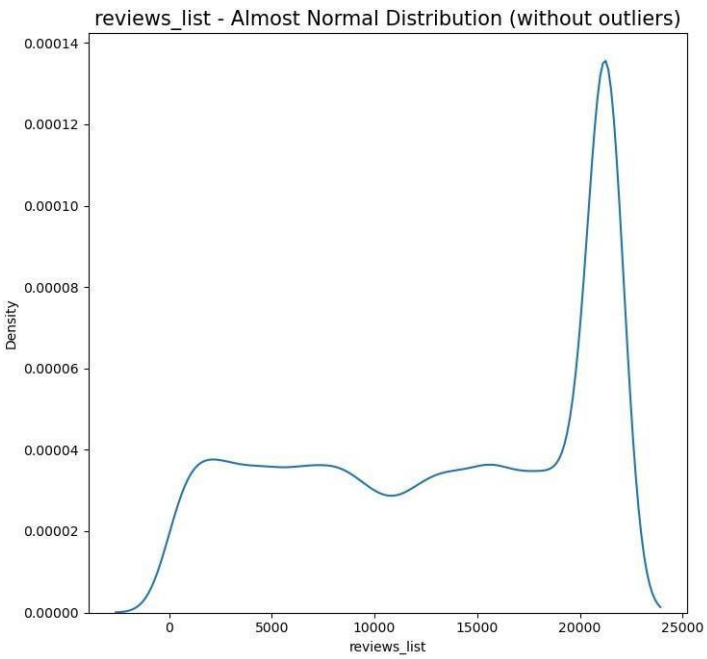

phone - Almost Normal Distribution (without outliers)
phone - Skewed Distribution (without outliers)
1e−5
Density
phone

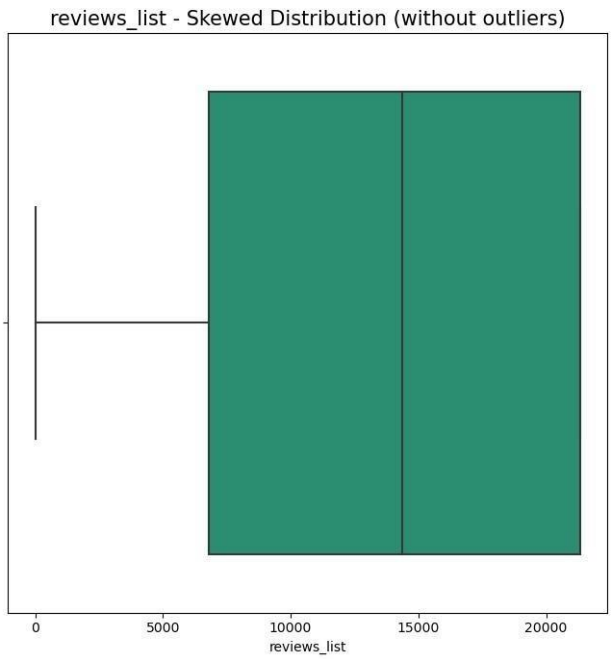

location - Almost Normal Distribution (without outliers)
location - Skewed Distribution (without outliers)
Density
location

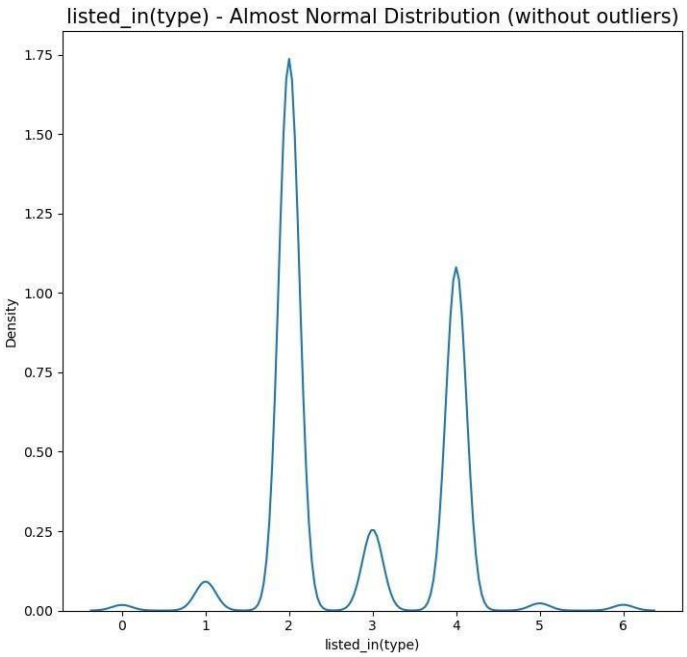

rest_type - Almost Normal Distribution (without outliers)
Density
rest_type
rest_type - Skewed Distribution (without outliers)
rest_type

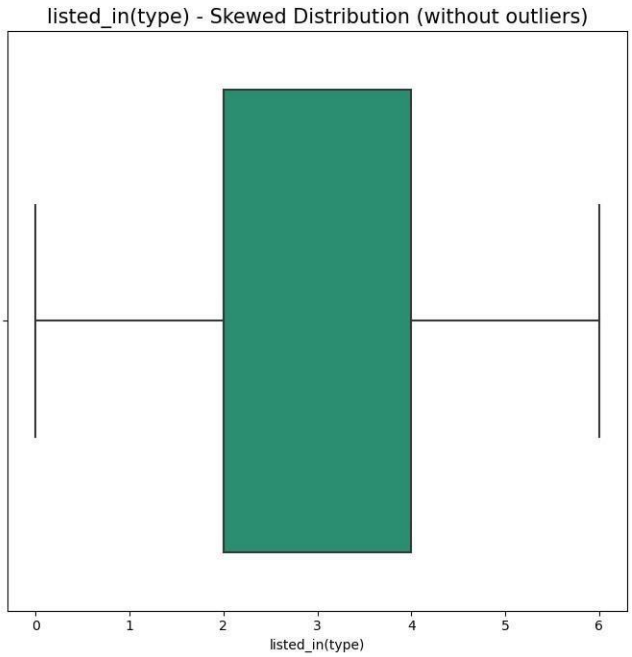

cuisines - Almost Normal Distribution (without outliers)
Density
cuisines
cuisines - Skewed Distribution (without outliers)
cuisines

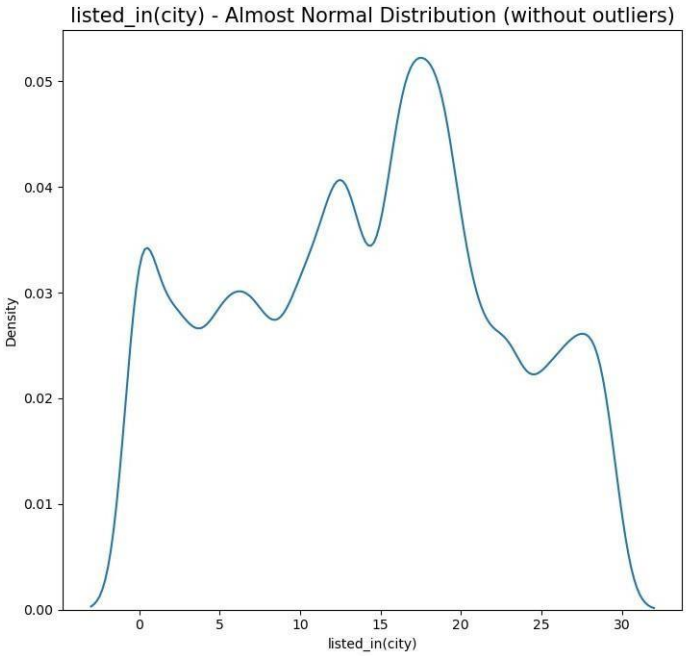

approx_cost(for two people) - Almost Normal Distribution (without outliers)
approx_cost(for two people) - Skewed Distribution (without outliers)
Density
approx_cost(for two people)
approx_cost(for two people)

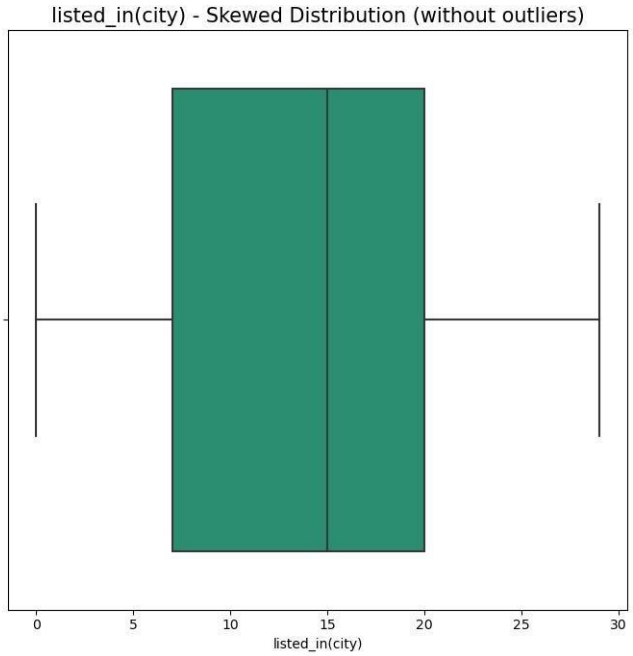












In [43]:
```
cleaned_data.head()
```

Out[43]:

	url	address	name	online_order	book_table	rate	votes	phone	location	rest_type	cuisines	approx_cost(for two people)	reviews_list	listed_in(
3	1160	2515	195	0	0	3.7	88	6770	1	78	2555	300.0	8805	
4	18293	340	2923	0	0	3.8	166	2006	4	27	2188	600.0	14812	
5	48103	4606	8187	1	0	3.8	286	10285	4	27	1823	600.0	5921	
6	37222	2396	6224	0	0	3.6	8	7432	57	27	2210	800.0	17557	
8	34390	128	5746	1	0	4.0	324	4097	1	19	742	700.0	4607	

In [44]:
```
cleaned_data.isnull().sum()
```

Out[44]:
```
url                            0
address                        0
name                           0
online_order                   0
book_table                     0
rate                           0
votes                          0
phone                          0
location                       0
rest_type                      0
cuisines                       0
approx_cost(for two people)    0
reviews_list                   0
listed_in(type)                0
listed_in(city)                0
dtype: int64
```

In [45]:
```
corr=cleaned_data.corr()
corr
```

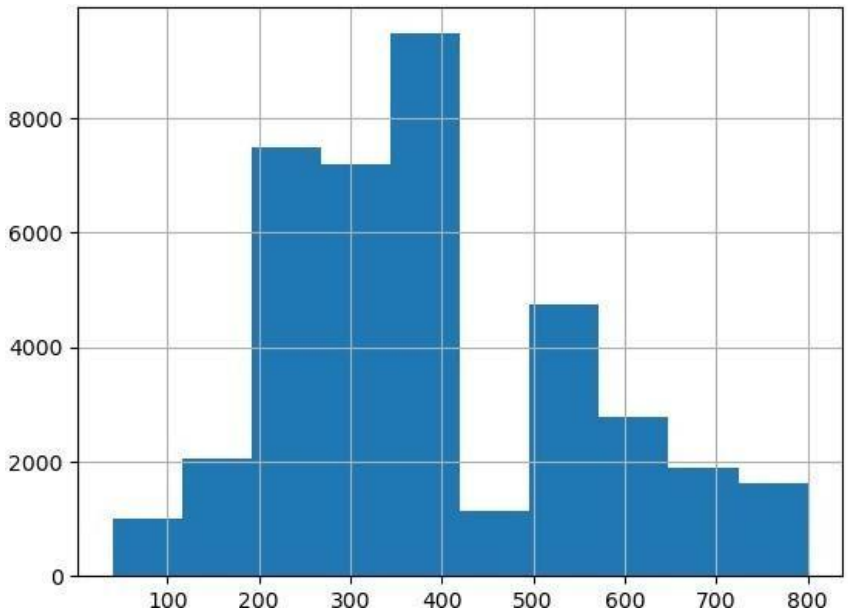
Out[45]:

	url	address	name	online_order	book_table	rate	votes	phone	location	rest_type	cuisines	ap
url	1.000000	0.034478	0.985576	0.014840	NaN	0.028441	0.009910	0.023979	0.006615	0.011905	0.085726	
address	0.034478	1.000000	0.032313	-0.066807	NaN	-0.009403	-0.054615	-0.010385	0.091069	-0.020070	0.016183	
name	0.985576	0.032313	1.000000	0.012167	NaN	0.031630	0.010579	0.026665	0.005568	0.012568	0.087673	
online_order	0.014840	-0.066807	0.012167	1.000000	NaN	0.091493	0.274433	-0.019633	-0.047366	-0.003202	-0.028225	
book_table	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
rate	0.028441	-0.009403	0.031630	0.091493	NaN	1.000000	0.421621	0.061464	0.038440	-0.132348	-0.068548	
votes	0.009910	-0.054615	0.010579	0.274433	NaN	0.421621	1.000000	0.131123	0.001046	-0.154009	-0.028314	
phone	0.023979	-0.010385	0.026665	-0.019633	NaN	0.061464	0.131123	1.000000	0.024857	-0.079213	0.028991	
location	0.006615	0.091069	0.005568	-0.047366	NaN	0.038440	0.001046	0.024857	1.000000	-0.037473	0.007758	
rest_type	0.011905	-0.020070	0.012568	-0.003202	NaN	-0.132348	-0.154009	-0.079213	-0.037473	1.000000	0.317273	
cuisines	0.085726	0.016183	0.087673	-0.028225	NaN	-0.068548	-0.028314	0.028991	0.007758	0.317273	1.000000	
approx_cost(for two people)	0.032412	-0.002947	0.031082	0.126632	NaN	0.096342	0.292744	0.078884	0.032069	-0.408389	-0.121322	
reviews_list	0.009724	0.032704	0.010401	-0.222146	NaN	0.251862	-0.253562	-0.091672	0.043590	0.050990	-0.009282	
listed_in(type)	0.018358	0.015450	0.021476	-0.229024	NaN	-0.035369	-0.031667	0.031567	0.024844	0.065276	0.104722	
listed_in(city)	-0.002362	0.053970	-0.002030	-0.054178	NaN	-0.004827	-0.007048	-0.000400	0.324379	0.005940	-0.003350	

In [46]:
```
df=cleaned_data
```

In [47]:
```
df['approx_cost(for two people)'].hist()
```

Out[47]:
```
<AxesSubplot:>
```



The null and alternate hypothesis of Jarque-Bera test are as follows:

```
H0: The data is normally distributed
H1: The data is not normally distributed
```

In [48]:
```
from scipy import stats
# normality test using jarque_bera()
# the test returns the the test statistics and the p-value of the test
stat, p = stats.shapiro(df['approx_cost(for two people)'])

# to print the numeric outputs of the Jarque-Bera test upto 3 decimal places
# %.3f: returns the a floating point with 3 decimal digit accuracy
# the '%' holds the place where the number is to be printed
print('Statistics=%.3f, p-value=%.3f' % (stat, p))

# display the conclusion
# set the level of significance to 0.05
alpha = 0.05

# if the p-value is greater than alpha print we accept alpha
# if the p-value is less than alpha print we reject alpha
if p > alpha:
    print('The data is normally distributed (fail to reject H0)')
else:
    print('The data is not normally distributed (reject H0)')
```

```
Statistics=0.949,  p-value=0.000
The data is not normally distributed (reject H0)
```

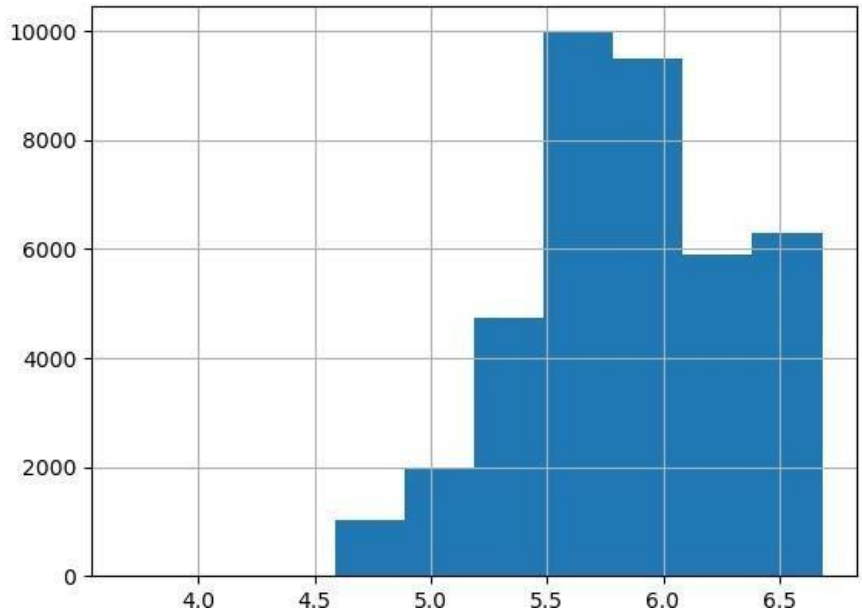
In [49]:
```
# log transformation for normality using np.log()
df['log_approx_cost_for_two_people'] = np.log(df['approx_cost(for  two  people)'])

# display first 5 rows of the data
df.head()
```

Out[49]:

	url	address	name	online_order	book_table	rate	votes	phone	location	rest_type	cuisines	approx_cost(for two people)	reviews_list	listed_in(
3	1160	2515	195	0	0	3.7	88	6770	1	78	2555	300.0	8805	
4	18293	340	2923	0	0	3.8	166	2006	4	27	2188	600.0	14812	
5	48103	4606	8187	1	0	3.8	286	10285	4	27	1823	600.0	5921	
6	37222	2396	6224	0	0	3.6	8	7432	57	27	2210	800.0	17557	
8	34390	128	5746	1	0	4.0	324	4097	1	19	742	700.0	4607	

In [50]:
```
df['log_approx_cost_for_two_people'].hist()
```

Out[50]:
```
<AxesSubplot:>
```



Jarque bera test

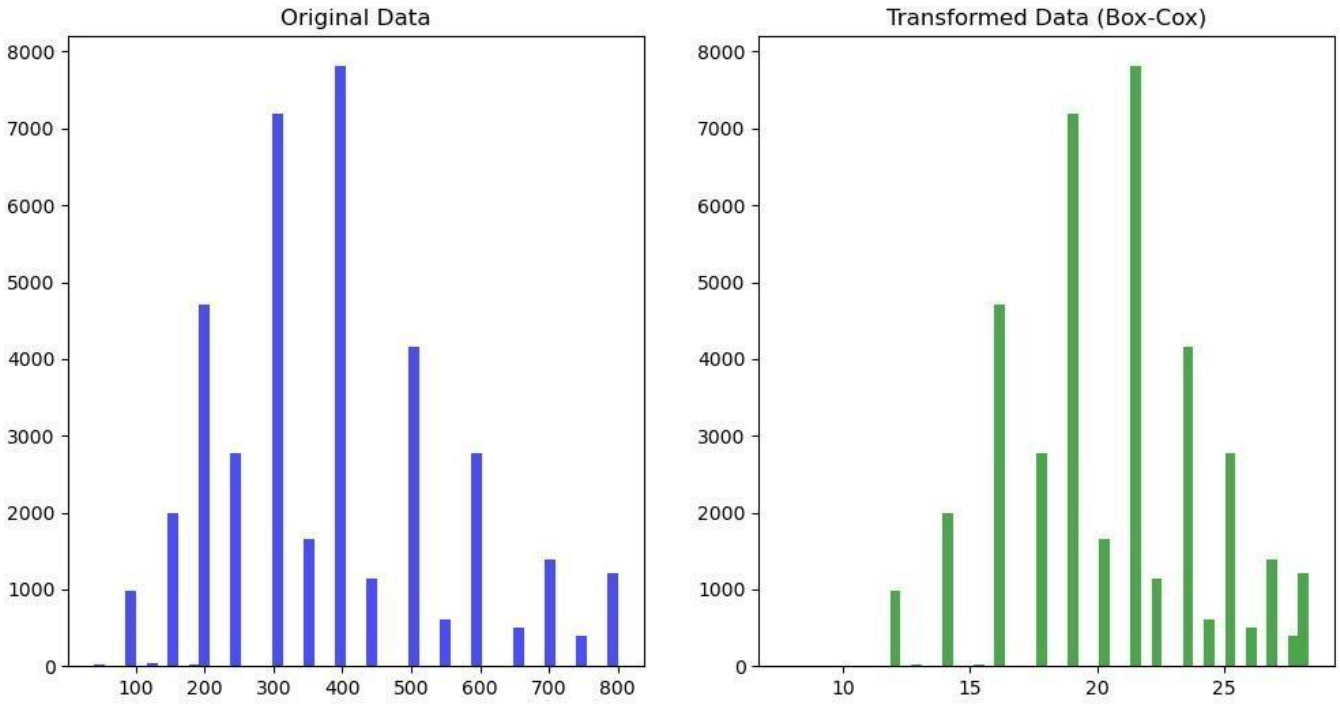
In [51]:
```
from scipy.stats import jarque_bera
# recheck normality by Jarque-Bera test
# the test returns the the test statistics and the p-value of the test
statn, pv = jarque_bera(df['log_approx_cost_for_two_people'])

# to print the numeric outputs of the Jarque-Bera test upto 3 decimal places
# %.3f: returns the a floating point with 3 decimal digit accuracy
# the '%' holds the place where the number is to be printed
print('Statistics=%.3f, p-value=%.3f' % (stat, p))

# display the conclusion
# set the level of significance to 0.05
alpha = 0.05

# if the p-value is greater than alpha print we accept alpha
# if the p-value is less than alpha print we reject alpha
if p > alpha:
    print('The data is normally distributed (fail to reject H0)')
else:
    print('The data is not normally distributed (reject H0)')
```

```
Statistics=0.949,  p-value=0.000
The data is not normally distributed (reject H0)
```

In [52]:
```
# Apply Box-Cox transformation
from scipy.stats import boxcox
transformed_data, lambda_value = boxcox(df['approx_cost(for two people)'])

# Plot original and transformed data
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(df['approx_cost(for two people)'], bins=50, color='blue', alpha=0.7)
plt.title('Original Data')

plt.subplot(1, 2, 2)
plt.hist(transformed_data, bins=50, color='green', alpha=0.7)
plt.title('Transformed Data (Box-Cox)')
```

```
Text(0.5, 1.0, 'Transformed Data (Box-Cox)')
```

Out[52]:



In [53]:
```
from scipy import stats
# normality test using jarque_bera()
# the test returns the the test statistics and the p-value of the test
stat, p = stats.shapiro(transformed_data)

# to print the numeric outputs of the Jarque-Bera test upto 3 decimal places
# %.3f: returns the a floating point with 3 decimal digit accuracy
# the '%' holds the place where the number is to be printed
print('Statistics=%.3f, p-value=%.3f' % (stat, p))

# display the conclusion
# set the level of significance to 0.05
alpha = 0.05

# if the p-value is greater than alpha print we accept alpha
# if the p-value is less than alpha print we reject alpha
if p > alpha:
    print('The data is normally distributed (fail to reject H0)')
else:
    print('The data is not normally distributed (reject H0)')
```

```
Statistics=0.976,  p-value=0.000
The data is not normally distributed (reject H0)
```

In [54]:
```
from scipy.stats import jarque_bera
# recheck normality by Jarque-Bera test
# the test returns the the test statistics and the p-value of the test
statn, pv = jarque_bera(transformed_data)

# to print the numeric outputs of the Jarque-Bera test upto 3 decimal places
# %.3f: returns the a floating point with 3 decimal digit accuracy
# the '%' holds the place where the number is to be printed
print('Statistics=%.3f, p-value=%.3f' % (stat, p))

# display the conclusion
# set the level of significance to 0.05
alpha = 0.05

# if the p-value is greater than alpha print we accept alpha
# if the p-value is less than alpha print we reject alpha
if p > alpha:
    print('The data is normally distributed (fail to reject H0)')
else:
    print('The data is not normally distributed (reject H0)')
```

```
Statistics=0.976,  p-value=0.000
The data is not normally distributed (reject H0)
```

In [55]:
```
df['transformed_approx_cost']=transformed_data
```

In [56]:
```
df.head()
```

Out[56]:

	url	address	name	online_order	book_table	rate	votes	phone	location	rest_type	cuisines	approx_cost(for two people)	reviews_list	listed_in(
3	1160	2515	195	0	0	3.7	88	6770	1	78	2555	300.0	8805	
4	18293	340	2923	0	0	3.8	166	2006	4	27	2188	600.0	14812	
5	48103	4606	8187	1	0	3.8	286	10285	4	27	1823	600.0	5921	
6	37222	2396	6224	0	0	3.6	8	7432	57	27	2210	800.0	17557	
8	34390	128	5746	1	0	4.0	324	4097	1	19	742	700.0	4607	

Multiple Linear Regression - Full Model - with Log Transformed Dependent Variable (OLS)

In [57]:
```
# add the intercept column to the dataset
df = sm.add_constant(df)

# separate the independent and dependent variables
# drop(): drops the specified columns
# axis=1: specifies that the column is to be dropped
X = df.drop(['approx_cost(for two people)','log_approx_cost_for_two_people','transformed_approx_cost'], axis=1)

# extract the target variable from the data set
y = df['log_approx_cost_for_two_people']
```

1. Split the data into training and test sets

In [58]:
```
# split data into train subset and test subset for predictor and target variables
# random_state: the seed used by the random number generator
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

Build model using sm.OLS().fit()

In [59]:
```
# build a full model using OLS()
# consider the log of charges
model_withlog = sm.OLS(y_train, X_train).fit()

# print the summary output
print(model_withlog.summary())
```

```
                                OLS Regression Results
==============================================================================================
Dep. Variable:     log_approx_cost_for_two_people   R-squared:                       0.194
Model:                                        OLS   Adj. R-squared:                  0.193
Method:                             Least Squares   F-statistic:                     545.9
Date:                            Fri, 08 Dec 2023   Prob (F-statistic):               0.00
Time:                                    07:41:17   Log-Likelihood:                -16345.
No. Observations:                           29538   AIC:                         3.272e+04
Df Residuals:                               29524   BIC:                         3.283e+04
Df Model:                                      13
Covariance Type:                        nonrobust
================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const              6.3362      0.038    165.794      0.000       6.261       6.411
url             1.758e-06   1.04e-06      1.685      0.092   -2.87e-07    3.8e-06
address         1.197e-06   7.47e-07      1.601      0.109   -2.68e-07   2.66e-06
name           -5.116e-06   6.11e-06     -0.838      0.402   -1.71e-05   6.85e-06
online_order       0.0647      0.005     11.955      0.000       0.054       0.075
book_table      1.465e-12   8.87e-15    165.222      0.000    1.45e-12    1.48e-12
rate              -0.0878      0.011     -8.334      0.000      -0.108      -0.067
votes              0.0010   3.07e-05     32.933      0.000       0.001       0.001
phone           2.491e-06    5.8e-07      4.297      0.000    1.35e-06    3.63e-06
location           0.0004   9.32e-05      4.819      0.000       0.000       0.001
rest_type         -0.0055   9.49e-05    -58.121      0.000      -0.006      -0.005
cuisines       -1.592e-05   3.32e-06     -4.790      0.000   -2.24e-05    -9.4e-06
reviews_list   -2.556e-06   3.94e-07     -6.490      0.000   -3.33e-06   -1.78e-06
listed_in(type)    0.0042      0.002      1.701      0.089      -0.001       0.009
listed_in(city)    0.0009      0.000      2.950      0.003       0.000       0.002
==============================================================================
Omnibus:                     2073.582   Durbin-Watson:                   1.991
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2638.829
Skew:                          -0.652   Prob(JB):                         0.00
Kurtosis:                       3.664   Cond. No.                     1.45e+19
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.59e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Multiple Linear Regression - Full Model - without Log Transformed Dependent Variable (OLS)

In [64]:
```
# add the intercept column to the dataset
df = sm.add_constant(df)

# separate the independent and dependent variables
# drop(): drops the specified columns
# axis=1: specifies that the column is to be dropped
X = df.drop(['approx_cost(for two people)','log_approx_cost_for_two_people','transformed_approx_cost'], axis=1)

# extract the target variable from the data set
y = df['approx_cost(for two people)']
```

In [65]:
```
# split data into train subset and test subset for predictor and target variables
# random_state: the seed used by the random number generator
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [66]:
```
# build a full model using OLS()
# consider the log of charges
model_without_log = sm.OLS(y_train, X_train).fit()

# print the summary output
print(model_without_log.summary())
```

```
                                OLS Regression Results
==============================================================================================
Dep. Variable:     approx_cost(for two people)   R-squared:                       0.229
Model:                                     OLS   Adj. R-squared:                  0.228
Method:                          Least Squares   F-statistic:                     673.4
Date:                         Fri, 08 Dec 2023   Prob (F-statistic):               0.00
Time:                                 07:49:26   Log-Likelihood:            -1.8928e+05
No. Observations:                        29538   AIC:                         3.786e+05
Df Residuals:                            29524   BIC:                         3.787e+05
Df Model:                                   13
Covariance Type:                    nonrobust
=================== ============================================ ============ ==========
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------------------
const              568.9389     13.332     42.675      0.000     542.808     595.070
url                  0.0009      0.000      2.414      0.016       0.000       0.002
address           2.687e-05      0.000      0.103      0.918      -0.000       0.001
name                -0.0029      0.002     -1.379      0.168      -0.007       0.001
online_order        21.9409      1.889     11.617      0.000      18.239      25.643
book_table       1.322e-10   3.09e-12     42.737      0.000    1.26e-10    1.38e-10
rate               -32.3068      3.676     -8.788      0.000     -39.513     -25.101
votes                0.3801      0.011     35.510      0.000       0.359       0.401
phone                0.0008      0.000      3.939      0.000       0.000       0.001
location             0.1460      0.032      4.491      0.000       0.082       0.210
rest_type           -2.2740      0.033    -68.674      0.000      -2.339      -2.209
cuisines            -0.0004      0.001     -0.351      0.725      -0.003       0.002
reviews_list        -0.0006      0.000     -4.696      0.000      -0.001      -0.000
listed_in(type)      1.9546      0.861      2.270      0.023       0.267       3.643
listed_in(city)      0.2082      0.109      1.911      0.056      -0.005       0.422
=================== ============================================ ==================
Omnibus:                      629.905   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              670.394
Skew:                           0.365   Prob(JB):                     2.67e-146
Kurtosis:                       3.107   Cond. No.                      1.45e+19
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.59e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [131]:
```
# create dataframe 'score_card'
# columns: specifies the columns to be selected
score_card = pd.DataFrame(columns=['Model_Name', 'R-Squared', 'Adj. R-Squared', 'RMSE'])

# print the score card
score_card
```

Out[131]:

Model_Name	R-Squared	Adj. R-Squared	RMSE

Calculate the p-values to know the insignificant variables

In [68]:
```
# calculate the p-values for all the variables
# create a dataframe using pd.DataFrame()
# columns: specifies the column names
linreg_full_model_withoutlog_pvalues = pd.DataFrame(model_without_log.pvalues, columns=["P-Value"])

# print the values
linreg_full_model_withoutlog_pvalues
```

Out[68]:

	P-Value
const	0.000000e+00
url	1.577346e-02
address	9.179266e-01
name	1.680561e-01
online_order	3.944961e-31
book_table	0.000000e+00
rate	1.606553e-18
votes	1.734161e-270
phone	8.200019e-05
location	7.105761e-06
rest_type	0.000000e+00
cuisines	7.254180e-01
reviews_list	2.665550e-06
listed_in(type)	2.323426e-02
listed_in(city)	5.601898e-02

In [ ]:

Let's create a list of insignificant variables

In [69]:
```
# select insignificant variables
insignificant_variables = linreg_full_model_withoutlog_pvalues[
                                                    linreg_full_model_withoutlog_pvalues['P-Value'] > 0.05

# get the position of a specified value
insigni_var = insignificant_variables.index

# convert the list of variables to 'list' type
insigni_var = insigni_var.to_list()

# get the list of insignificant variables
insigni_var
```

Out[69]:
```
['address', 'name', 'cuisines', 'listed_in(city)']
```

Building a model by dropping insignificant variables

In [70]:
```
# add the intercept column to the dataset
df = sm.add_constant(df)

# separate the independent and dependent variables
# drop(): drops the specified columns
# axis=1: specifies that the column is to be dropped
X = df.drop(['approx_cost(for two people)','log_approx_cost_for_two_people','transformed_approx_cost','address'
             'cuisines', 'listed_in(city)'], axis=1)

# extract the target variable from the data set
y = df['approx_cost(for two people)']
```

In [71]:
```
# split data into train subset and test subset for predictor and target variables
# random_state: the seed used by the random number generator
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In [72]:
```
# build a full model using OLS()
# consider the log of charges
model_without_log = sm.OLS(y_train, X_train).fit()

# print the summary output
print(model_without_log.summary())
```

OLS Regression Results

Dep. Variable:	approx_cost(for two people)	R-squared:	0.229
Model:	OLS	Adj. R-squared:	0.228
Method:	Least Squares	F-statistic:	971.9
Date:	Fri, 08 Dec 2023	Prob (F-statistic):	0.00
Time:	07:56:50	Log-Likelihood:	-1.8928e+05
No. Observations:	29538	AIC:	3.786e+05
Df Residuals:	29528	BIC:	3.787e+05
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	571.4898	13.153	43.450	0.000	545.709	597.270
url	0.0004	5.8e-05	6.580	0.000	0.000	0.000
online_order	21.8265	1.884	11.582	0.000	18.133	25.520
book_table	9.051e-11	2.17e-12	41.628	0.000	8.63e-11	9.48e-11
rate	-32.3338	3.673	-8.802	0.000	-39.534	-25.134
votes	0.3801	0.011	35.549	0.000	0.359	0.401
phone	0.0008	0.000	3.887	0.000	0.000	0.001
location	0.1664	0.031	5.428	0.000	0.106	0.227
rest_type	-2.2762	0.031	-72.479	0.000	-2.338	-2.215
reviews_list	-0.0006	0.000	-4.729	0.000	-0.001	-0.000
listed_in(type)	1.9139	0.859	2.229	0.026	0.231	3.597

Omnibus:	627.298	Durbin-Watson:	1.995
Prob(Omnibus):	0.000	Jarque-Bera (JB):	667.430
Skew:	0.365	Prob(JB):	1.17e-145
Kurtosis:	3.104	Cond. No.	1.51e+19

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.39e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

Predict the values using test set

In [76]:
```
# predict the charges using predict()
predicted_claim = model_without_log.predict(X_test)

# extract the 'charges' values from the test data
actual_claim = y_test
```

Compute model accuracy measures

Now we calculate accuray measures like Root-mean-square-error (RMSE), R-squared and Adjusted R-squared.

In [78]:
```
# calculate rmse using rmse()
linreg_full_model_withoutlog_rmse = rmse(actual_claim, predicted_claim)

# calculate R-squared using rsquared
linreg_full_model_withoutlog_rsquared  =  model_without_log.rsquared

# calculate Adjusted R-Squared using rsquared_adj
linreg_full_model_withoutlog_rsquared_adj  =  model_without_log.rsquared_adj
```

In [132…
```
# compile the required information
linreg_full_model_withoutlog_metrics = pd.Series({
                    'Model_Name': "Linreg full model without log of target variable",
                    'RMSE':linreg_full_model_withoutlog_rmse,
                    'R-Squared': linreg_full_model_withoutlog_rsquared,
                    'Adj. R-Squared': linreg_full_model_withoutlog_rsquared_adj
                   })

# append our result table using append()
# ignore_index=True: does not use the index labels
# python can only append a Series if ignore_index=True or if the Series has a name
score_card  =  score_card.append(linreg_full_model_withoutlog_metrics,  ignore_index=True)

# print the result table
score_card
```

Out[132]:

	Model_Name	R-Squared	Adj. R-Squared	RMSE
0	Linreg full model without log of target variable	0.228539	0.228304	146.71909

In [87]:
```
print('Model_Name- Linreg full model without log of target variable')
print('RMSE-',linreg_full_model_withoutlog_rmse)
print('R-Squared-',  linreg_full_model_withoutlog_rsquared)
```

```
print('Adj. R-Squared-' ,linreg_full_model_withoutlog_rsquared_adj)
```

```
Model_Name- Linreg full model without log of target variable
RMSE- 146.71909021343012
R-Squared- 0.22853867201271783
Adj. R-Squared- 0.22830353411133997
```

Detecting Heteroskedasticity

Heteroskedasticity is a systematic change in the spread of the residuals over the range of measured values. One of the assumptions of the linear regression is that there should not be Heteroskedasticity.

Breusch-Pagan is the test for detecting heteroskedasticity:

The null and alternate hypothesis of Breusch-Pagan test is as follows:

H0: The residuals are homoskedastic H1: The residuals are not homoskedastic

In [73]:
```
# create vector of result parmeters
name = ['f-value','p-value']

# perform Breusch-Pagan test using het_breushpagan()
# compute residuals using 'resid'
# 'exog' returns the independent variables in the model alng with the intercept
test  =  sms.het_breuschpagan(model_without_log.resid,  model_without_log.model.exog)

# print the output
# use 'lzip' to zip the column names and test results
lzip(name, test)
```

Out[73]:
```
[('f-value', 1702.2137677272638), ('p-value', 0.0)]
```

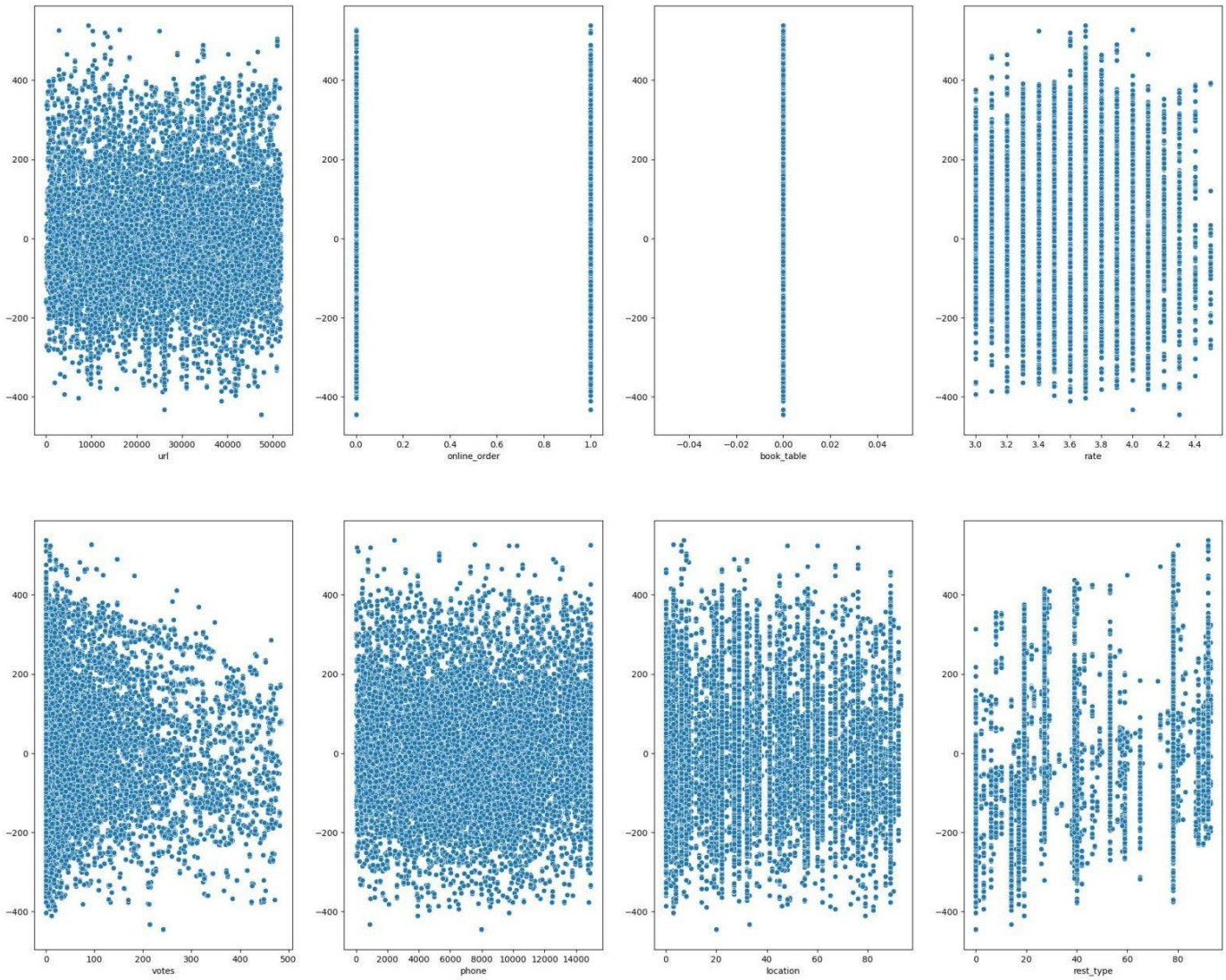
We observe that p-value is less than 0.05 and thus reject the null hypothesis. We conclude that there is heteroskedasticity present in the data.

Linearity of Residuals

Multiple linear regression requires the relationship between the independent and dependent variables to be linear, i.e. it should be linear in the parameters. The linearity assumption can best be tested with scatterplots.

The independent variables must have a linear relationship with the dependent variable.

In [89]:
```
# create subplots of scatter plots
# pass the number of rows in a subplot to 'nrows'
# pass the number of columns in a subplot to 'ncolumns'
# set plot size using 'figsize'
fig, ax = plt.subplots(nrows = 2, ncols= 4, figsize=(25, 20))

# use for loop to create scatter plot for residuals and each independent variable (do not consider the intercep
# 'ax' assigns axes object to draw the plot onto
for variable, subplot in zip(X_train.columns[1:], ax.flatten()):
    sns.scatterplot(X_train[variable],model_without_log.resid  ,  ax=subplot)

# display the plot
plt.show()
```



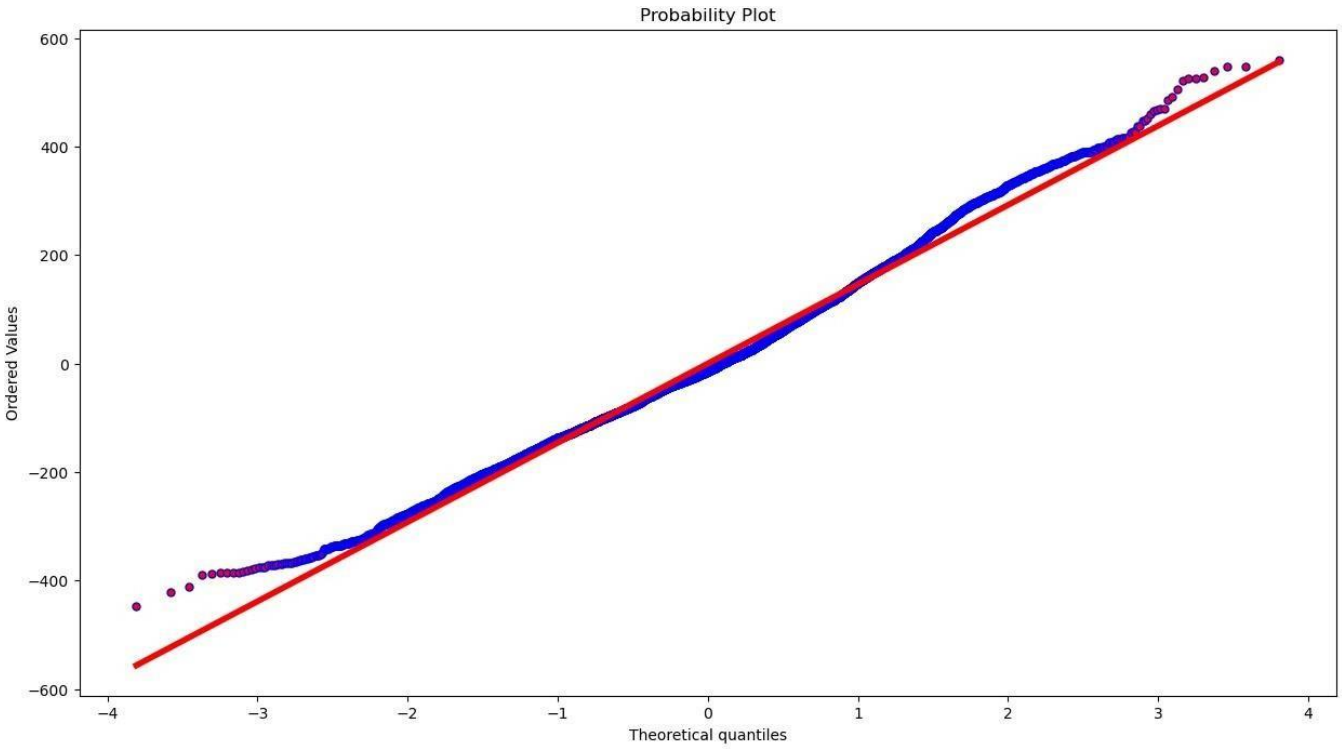
From the plots we see that none of the plots show a specific pattern much. Hence, we may conclude that the variables are linearly related to the dependent variable.

Normality of Residuals

The assumption of normality is an important assumption for many statistical tests. The normal Q-Q plot is one way to assess normality. The quantile-quantile(Q-Q) is a scatter plot that will help in validating the assumption of normal distribution in a data set.

In [90]:
```
# calculate fitted values
fitted_vals = model_without_log.predict(X_test)

# calculate residuals
resids = actual_claim - fitted_vals

# create subplots using subplots() such that there is one row having one plot
# 'figsize' sets the figure size
fig, ax = plt.subplots(1, 1, figsize=(15, 8))

# plot the probability plot to check the normality of the residuals
# plot: if specified plots the least squares fit
stats.probplot(resids, plot=plt)

# set the marker type using the set_marker() parameter
# access the line object from the axes object using ax.get_lines()
# then, the properties can be changed accordingly
# set the marker to 'o' to use circles as points
ax.get_lines()[0].set_marker('o')

# set the marker size using the set_markersize() parameter
# set the marker size to 5
ax.get_lines()[0].set_markersize(5.0)

# set the color of the trend line using set_markerfacecolor()
# set color of the trend line to red by passing 'r' to the set_markerfacecolor
ax.get_lines()[0].set_markerfacecolor('r')

# set the trend line width
ax.get_lines()[1].set_linewidth(4.0)

# display the plot
```

```
plt.show()
```



Using this plot, we can infer that the residuals do not come from a normal distribution. This is expected as our target variable is not normally distributed.

In [91]:
```
# check the mean of the residual
model_without_log.resid.mean()
```

Out[91]:
```
1.7809128958073518e-13
```

The mean of the residuals is very much closer to zero. Therefore, we can say that linearity is present.

Perform Jarque Bera test to check normality of the residuals

In [92]:
```
# normality test using 'jarque_bera'
# the test returns the the test statistics and the p-value of the test
stat, p = jarque_bera(resids)

# to print the numeric outputs of the Jarque-Bera test upto 3 decimal places
# %.3f: returns the a floating point with 3 decimal digit accuracy
# the '%' holds the place where the number is to be printed
print('Statistics=%.3f, p-value=%.3f' % (stat, p))

# display the conclusion
# set the level of significance to 0.05
alpha = 0.05

# if the p-value is greater than alpha print we accept alpha
# if the p-value is less than alpha print we reject alpha
if p > alpha:
    print('The data is normally distributed (fail to reject H0)')
else:
    print('The data is not normally distributed (reject H0)')
```

```
Statistics=193.282,  p-value=0.000
The data is not normally distributed (reject H0)
```

It is apparent that the p-value is less than 0.05. So we have enough evidence to reject the null hypothesis. It can be concluded that the residuals is not normally distributed.

computing interaction effect

In [96]:
```
# create a copy of the entire dataset to add the interaction effect using copy()
df_interaction = df.copy()

# add the interaction variable
df_interaction['rate_mul_votes'] = df_interaction['rate']*df_interaction['votes'] *df_interaction['rest_type']

# print the data with interaction
df_interaction.head()
```

Out[96]:

	const	url	address	name	online_order	book_table	rate	votes	phone	location	rest_type	cuisines	approx_cost(for two people)	reviews_list	li
3	1.0	1160	2515	195	0	0	3.7	88	6770	1	78	2555	300.0	8805	
4	1.0	18293	340	2923	0	0	3.8	166	2006	4	27	2188	600.0	14812	
5	1.0	48103	4606	8187	1	0	3.8	286	10285	4	27	1823	600.0	5921	
6	1.0	37222	2396	6224	0	0	3.6	8	7432	57	27	2210	800.0	17557	
8	1.0	34390	128	5746	1	0	4.0	324	4097	1	19	742	700.0	4607	

In [97]:

```
# separate the independent and dependent variables
# drop(): specify the variables to be dropped
# axis=1: specifies that the columns are to be dropped
X = df_interaction.drop(['approx_cost(for two people)','log_approx_cost_for_two_people','transformed_approx_cos

# extract the target variable from the train set
y = df_interaction['approx_cost(for two people)']

# split data into train subset and test subset for predictor and target variables
# random_state: the seed used by the random number generator
X_train_interaction, X_test_interaction, y_train, y_test = train_test_split( X, y, random_state=1)

# check the dimensions of the train & test subset for

# print dimension of predictors train set
print("The shape of X_train_interaction is:",X_train_interaction.shape)

# print dimension of predictors test set
print("The shape of X_test_interaction is:",X_test_interaction.shape)

# print dimension of target train set
print("The shape of y_train is:",y_train.shape)

# print dimension of target test set
print("The shape of y_test is:",y_test.shape)
```

```
The shape of X_train_interaction is: (29538, 16)
The shape of X_test_interaction is: (9846, 16)
The shape of y_train is: (29538,)
The shape of y_test is: (9846,)
```

In [98]:

```
# building a full model with an interaction term using OLS()
linreg_with_interaction = sm.OLS(y_train, X_train_interaction).fit()

# print the summary output
print(linreg_with_interaction.summary())
```

```
                                OLS Regression Results
==============================================================================
Dep. Variable:     approx_cost(for two people)   R-squared:                       0.235
Model:                                     OLS   Adj. R-squared:                  0.234
Method:                          Least Squares   F-statistic:                     646.9
Date:                         Fri, 08 Dec 2023   Prob (F-statistic):               0.00
Time:                                 11:02:01   Log-Likelihood:            -1.8916e+05
No. Observations:                        29538   AIC:                         3.784e+05
Df Residuals:                            29523   BIC:                         3.785e+05
Df Model:                                   14
Covariance Type:                    nonrobust
==============================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const            535.8806     13.454     39.830      0.000     509.510     562.252
url                0.0010      0.000      2.675      0.007       0.000       0.002
address            0.0002      0.000      0.591      0.555      -0.000       0.001
name              -0.0037      0.002     -1.720      0.085      -0.008       0.001
online_order      22.8558      1.882     12.143      0.000      19.167      26.545
book_table    -4.638e-10   1.22e-11    -37.941      0.000   -4.88e-10   -4.4e-10
rate             -27.2719      3.677     -7.418      0.000     -34.478     -20.065
votes              0.6164      0.019     32.849      0.000       0.580       0.653
phone              0.0008      0.000      3.944      0.000       0.000       0.001
location           0.1337      0.032      4.130      0.000       0.070       0.197
rest_type         -1.9773      0.038    -51.680      0.000      -2.052      -1.902
cuisines          -0.0005      0.001     -0.465      0.642      -0.003       0.002
reviews_list      -0.0008      0.000     -5.633      0.000      -0.001      -0.001
listed_in(type)    1.9782      0.858      2.306      0.021       0.297       3.660
listed_in(city)    0.2199      0.109      2.026      0.043       0.007       0.433
rate_mul_votes    -0.0013   8.39e-05    -15.302      0.000      -0.001      -0.001
==============================================================================
Omnibus:                      722.948   Durbin-Watson:                   1.994
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              776.648
Skew:                           0.392   Prob(JB):                    2.25e-169
Kurtosis:                       3.134   Cond. No.                     2.49e+17
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.23e-22. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

In [134…

```
# calculate rmse using rmse()
linreg_with_interaction_rmse = rmse(actual_claim, predicted_claim)

# calculate R-squared using rsquared
linreg_with_interaction_rsquared = linreg_with_interaction.rsquared

# calculate Adjusted R-Squared using rsquared_adj
linreg_with_interaction_rsquared_adj = linreg_with_interaction.rsquared_adj
```

In [135…

```
# compile the required information
linreg_with_interaction_metrics = pd.Series({
                    'Model_Name': "linreg_with_interaction",
                    'RMSE': linreg_with_interaction_rmse,
                    'R-Squared': linreg_with_interaction_rsquared,
                    'Adj. R-Squared': linreg with interaction rsquared adj
                   })

# append our result table using append()
# ignore_index=True: does not use the index labels
# python can only append a Series if ignore_index=True or if the Series has a name
score_card = score_card.append(linreg_with_interaction_metrics, ignore_index = True)

# print the result table
score_card
```

Out[135]:

	Model_Name	R-Squared	Adj. R-Squared	RMSE
0	Linreg full model without log of target variable	0.228539	0.228304	146.71909
1	linreg_with_interaction	0.234757	0.234395	146.71909

Scaling the data

In [108…

```
X_train_scaled,X_test_scaled,y_train_scaled,y_test_scaled = train_test_split(X,y,random_state=1)
```

In [109…

```
X_train_scaled.shape,X_test_scaled.shape
```

Out[109]:

```
((29538, 12), (9846, 12))
```

In [110…

```
y_train_scaled.shape,y_test_scaled.shape
```

Out[110]:

```
((29538,), (9846,))
```

In [111]:
```
scaler_ind = StandardScaler()
scaler_ind.fit(X_train_scaled)
```

Out[111]:
```
StandardScaler()
```

In [112]:
```
X_train_scaled = pd.DataFrame(scaler_ind.transform(X_train_scaled),columns=X_train_scaled.columns)

scaler_ind.fit(X_test_scaled)

X_test_scaled = pd.DataFrame(scaler_ind.transform(X_test_scaled),columns=X_test_scaled.columns)
```

In [113]:
```
X_train_scaled.head()
```

Out[113]:

	const	url	online_order	book_table	rate	votes	phone	location	rest_type	reviews_list	listed_in(type)	rate_mul_vo
0	0.0	-1.531673	0.852977	0.0	0.541204	0.074026	-0.805533	1.927806	-1.495196	-1.525299	-0.738393	-0.383
1	0.0	0.661879	0.852977	0.0	-1.934407	-0.534192	-1.564949	-1.041234	0.806816	-1.786718	-0.738393	-0.426
2	0.0	-1.605060	-1.172364	0.0	-1.580748	-0.615288	-0.349139	1.462776	0.806816	-0.978479	-0.738393	-0.515
3	0.0	-0.867727	0.852977	0.0	-0.166113	-0.595014	-1.484190	1.462776	1.310381	0.726673	-0.738393	-0.469
4	0.0	0.184453	0.852977	0.0	0.187545	1.209365	-0.113610	-0.969690	0.806816	-0.032210	1.208639	1.905

In [114]:
```
# scaling the target variable
scaler_target = StandardScaler()
scaler_target.fit(np.array(y_train_scaled).reshape((-1,1)))
y_train_scaled = scaler_target.fit_transform(np.array(y_train_scaled).reshape((-1,1)))


scaler_target.fit(np.array(y_test_scaled).reshape((-1,1)))
y_test_scaled = scaler_target.fit_transform(np.array(y_test_scaled).reshape((-1,1)))
```

Fit the linear regression using the SGD

In [115]:
```
# import SGDRegressor from sklearn to perform linear regression with stochastic gradient descent
from sklearn.linear_model import SGDRegressor

# build the model
linreg_with_SGD = SGDRegressor()

# we fit our model with train data
linreg_with_SGD  =  linreg_with_SGD.fit(X_train_scaled,  y_train_scaled)
```

Predict the values using test set

In [116]:
```
# we use predict() to predict our values
# Take the inverse of the prediction as we have done the transformation
linreg_with_SGD_predictions = scaler_target.inverse_transform(linreg_with_SGD.predict(X_test_scaled).reshape(-1
```

4. Compute accuracy measures

Now we calculate accuray measures Root-mean-square-error (RMSE), R-squared and Adjusted R-squared.

In [117]:
```
# calculate mse by taking inverse of the data as we did scaling
linreg_SGD_mse = mean_squared_error(y_test,linreg_with_SGD_predictions)

# calculate rmse
linreg_SGD_rmse = np.sqrt(linreg_SGD_mse)

# calculate R-squared
linreg_SGD_r_squared  =  r2_score(y_test,  linreg_with_SGD_predictions)

# calculate Adjusted R-squared
linreg_SGD_adjusted_r_squared  =  1  -  (1-linreg_SGD_r_squared)*(len(y_test)-1)/(len(y_test)-  X_test.shape[1]-1)
```

In [136]:
```
# compile the required information
linreg_full_model_SGD = pd.Series({
                        'Model_Name': "Linear Regression SGD",
                        'RMSE': linreg_SGD_rmse ,
                        'R-Squared': linreg_SGD_r_squared,
                        'Adj. R-Squared': linreg_SGD_adjusted_r_squared
                       })

# append our result table using append()
# ignore_index=True: does not use the index labels
# python can only append a Series if ignore_index=True or if the Series has a name
score_card = score_card.append(linreg_full_model_SGD, ignore_index = True)

# print the result table
```

```
score_card
```

Out[136]:

	Model_Name	R-Squared	Adj. R-Squared	RMSE
0	Linreg full model without log of target variable	0.228539	0.228304	146.71909
1	linreg_with_interaction	0.234757	0.234395	146.71909
2	Linear Regression SGD	0.242687	0.241763	146.28328

In [121…

```
print('Model_Name- Linear Regression SGD')
print('RMSE-', linreg_SGD_rmse )
print('R-Squared-',  linreg_SGD_r_squared)
print('Adj.  R-Squared-',  linreg_SGD_adjusted_r_squared)
```

```
Model_Name- Linear Regression SGD
RMSE- 146.28327985549802
R-Squared- 0.242687191198797
Adj. R-Squared- 0.24176298152671172
```

Linear Regression with SGD using GridSearchCV

Fit the linear regression using the SGD with GridSearchCV

In [124…

```
# import gridsearchcv from sklearn to optimize best parameter
from sklearn.model_selection import GridSearchCV

# Grid search

# alpha is regularization term usually in the range 10.0**-np.arange(1,7).
# this will generate different alpha values like 1.e-01, 1.e-02, 1.e-03, 1.e-04, 1.e-05, 1.e-06

# loss="squared_loss": Ordinary least squares,

# "l2": L2 norm penalty on coefficients (ridge regression),
# "l1": L1 norm penalty on coefficients(lasso regression), "elasticnet": Convex combination of L2 and L1

# The learning rate can be either constant or gradually decaying

param_grid = { 'alpha': 10.0 ** -np.arange(1, 7),

               'loss': ['squared_loss'],

               'penalty': ['l2', 'l1', 'elasticnet'],

               'learning_rate': ['constant', 'optimal', 'invscaling']
}

# using sklearn's GridSearchCV, we define our grid of parameters to search over and then run the grid search
reg = GridSearchCV(linreg_with_SGD, param_grid)

# fit the model on train data
reg.fit(X_train_scaled, y_train_scaled)
```

Out[124]:

```
GridSearchCV(estimator=SGDRegressor(),
             param_grid={'alpha': array([1.e-01, 1.e-02, 1.e-03, 1.e-04, 1.e-05, 1.e-06]),
                         'learning_rate': ['constant', 'optimal', 'invscaling'],
                         'loss': ['squared_loss'],
                         'penalty': ['l2', 'l1', 'elasticnet']})
```

Let us print the optimal parameters obtained by using GridSearchCV.

In [125…

```
# print the best parameters
# print best alpha value
print('Best alpha:', reg.best_estimator_.alpha)

# print best tolerance
print('Best tol:', reg.best_estimator_.tol)

# print best starting rate (Use eta0 to specify the starting learning)
print('Best eta0:', reg.best_estimator_.eta0)

# print best learning rate
print('Best learning rate:', reg.best_estimator_.learning_rate)
```

```
Best alpha: 0.001
Best tol: 0.001
Best eta0: 0.01
Best learning rate: invscaling
```

We have obtained the optimal parameters. Now substituting these values in SGDRegressor() we build the model.

In [126…

```
# build the model using best parameters
# squared_loss: ordinary least squares loss
# 'eta0': specify the starting learning
```

```
linreg_SGD_using_best_parameter  =  SGDRegressor(alpha=0.0001,
                                                 tol=0.001,
                                                 eta0=0.01,
                                                 learning_rate='constant')

# fit the SGD model using best parameter
linreg_SGD_using_best_parameter.fit(X_train_scaled,y_train_scaled)
```

Out[126]:
```
SGDRegressor(learning_rate='constant')
```

Predict the values using test set

In [127…
```
# predict the values on test data using predict
linreg_SGD_using_best_parameter_predictions  =    scaler_target.inverse_transform(linreg_SGD_using_best_parameter.
```

Compute accuracy measures

In [128…
```
# calculate mse
linreg_SGD_using_best_parameter_mse  =  mean_squared_error(y_test,  linreg_SGD_using_best_parameter_predictions)

# calculate rmse
linreg_SGD_using_best_parameter_rmse  =  np.sqrt(linreg_SGD_using_best_parameter_mse)

# calculate R-squared
linreg_SGD_using_best_parameter_r_squared  =  r2_score(y_test,  linreg_SGD_using_best_parameter_predictions)

# calculate Adjusted R-squared
linreg_SGD_using_best_parameter_adjusted_r_squared  =  1  -  (1-linreg_SGD_using_best_parameter_r_squared)*(len(y_t
```

In [137…
```
# compile the required information
linreg_full_model_SGD = pd.Series({
                    'Model_Name': "Linear Regression SGD",
                    'RMSE': linreg_SGD_using_best_parameter_rmse ,
                    'R-Squared': linreg_SGD_using_best_parameter_ r_squared,
                    'Adj. R-Squared': linreg_SGD_using_best_param eter_adjusted_r_squared
                   })

# append our result table using append()
# ignore_index=True: does not use the index labels
# python can only append a Series if ignore_index=True or if the S eries has a name
score_card  =  score_card.append(linreg_full_model_SGD,  ignore_index  =  True)

# print the result table
score_card
```

Out[137]:

	Model_Name	R-Squared	Adj. R-Squared	RMSE
0	Linreg full model without log of target variable	0.228539	0.228304	146.719090
1	linreg_with_interaction	0.234757	0.234395	146.719090
2	Linear Regression SGD	0.242687	0.241763	146.283280
3	Linear Regression SGD	0.219221	0.218268	148.532397

Feature Selection

Forward Selection

In [138…
```
from sklearn.metrics import accuracy_score as acc
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
```

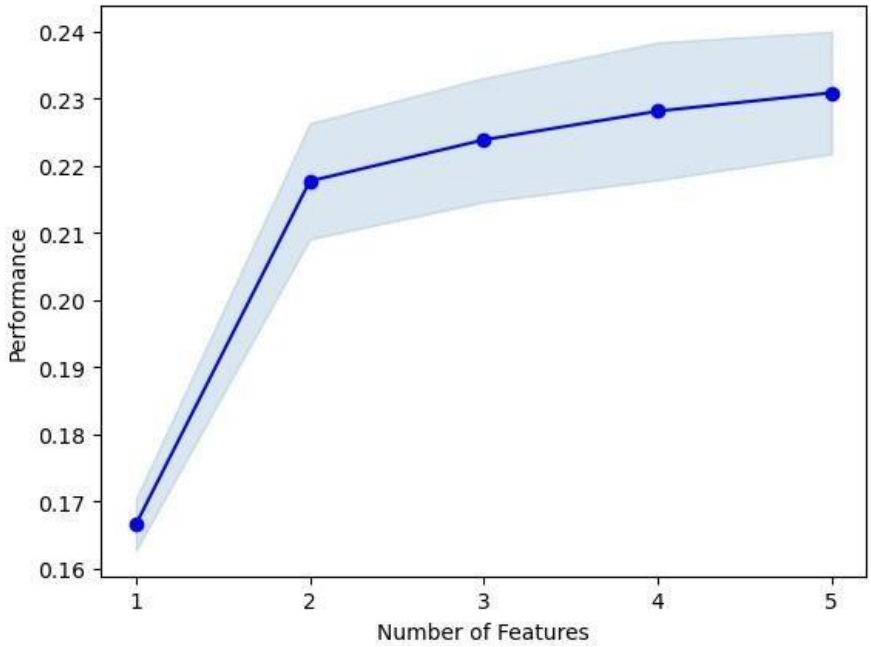
To do the feature selection we will use the interaction variables i.e X_train_interaction, and X_test_interaction which we cr eated while we came up with interaction effect features.

In [139…
```
reg = LinearRegression()

# Build step forward feature selection
sfs1 = sfs(reg, k_features = 5, forward=True,
           floating=False, scoring='r2',
           verbose=2,
           cv=5)

# Perform SFFS
sfs1 = sfs1.fit(X_train_interaction, y_train)

from mlxtend.plotting import plot_sequential_feature_selection as plt_sfs
plt.figure(figsize=(50,50))
fig1 = plt_sfs(sfs1.get_metric_dict(), kind= 'std_dev')
plt.show()
```

```
result = pd.DataFrame(sfs1.get_metric_dict()).T
result = result[['feature_idx','avg_score','feature_names']]
imp_vars_forward = list(result.feature_names[result['avg_score'] == max(result.avg_score)])
imp_vars_forward
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:    0.3s finished

[2023-12-08 13:18:57] Features: 1/5 -- score: 0.16653763354986778[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:    0.5s finished

[2023-12-08 13:18:57] Features: 2/5 -- score: 0.21768060532470254[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:    0.5s finished

[2023-12-08 13:18:58] Features: 3/5 -- score: 0.22382631978273554[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:    0.5s finished

[2023-12-08 13:18:59] Features: 4/5 -- score: 0.2281029626859823[Parallel(n_jobs=1)]: Using backend SequentialB
ackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.4s finished

[2023-12-08 13:18:59] Features: 5/5 -- score: 0.23083923463790185
```

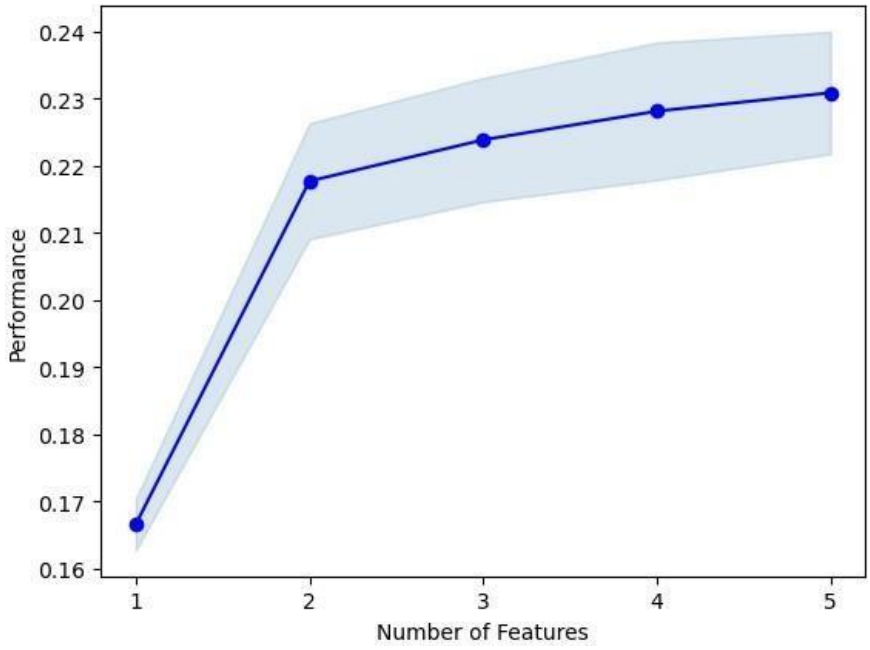
```
<Figure size 5000x5000 with 0 Axes>
```



Out[139]:
```
[('online_order', 'rate', 'votes', 'rest_type', 'rate_mul_votes')]
```

In [140…
```
from mlxtend.plotting import plot_sequential_feature_selection as plt_sfs
plt.figure(figsize=(50,50))
fig1 = plt_sfs(sfs1.get_metric_dict(), kind= 'std_dev')
plt.show()
```

```
<Figure size 5000x5000 with 0 Axes>
```



In [141]:
```
imp_vars_forward
```

Out[141]:
```
[('online_order', 'rate', 'votes', 'rest_type', 'rate_mul_votes')]
```

Backward Selection

In [142]:
```
X_train_interaction.shape
```

Out[142]:
```
(29538, 16)
```

In [143]:
```
reg = LinearRegression()

# Build step forward feature selection
sfs2 = sfs(reg, k_features = 5, forward=False,
           floating=False, scoring='r2',
           verbose=2,
           cv=5)

# Perform Backward selection
sfs2 = sfs2.fit(X_train_interaction, y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:     1.6s finished

[2023-12-08 13:21:18] Features: 15/5 -- score: 0.23375727542469638[Parallel(n_jobs=1)]: Using backend Sequentia
lBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed:     1.4s finished

[2023-12-08 13:21:20] Features: 14/5 -- score: 0.2337887090976912[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  14 out of  14 | elapsed:     1.3s finished

[2023-12-08 13:21:21] Features: 13/5 -- score: 0.23378870909767882[Parallel(n_jobs=1)]: Using backend Sequentia
lBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  13 out of  13 | elapsed:     1.1s finished

[2023-12-08 13:21:22] Features: 12/5 -- score: 0.23378870909767624[Parallel(n_jobs=1)]: Using backend Sequentia
lBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:     0.9s finished

[2023-12-08 13:21:23] Features: 11/5 -- score: 0.23375263128627846[Parallel(n_jobs=1)]: Using backend Sequentia
lBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  11 out of  11 | elapsed:     0.7s finished

[2023-12-08 13:21:24] Features: 10/5 -- score: 0.23370730392794256[Parallel(n_jobs=1)]: Using backend Sequentia
lBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done  10 out of  10 | elapsed:     0.6s finished

[2023-12-08 13:21:25] Features: 9/5 -- score: 0.23364021670387025[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.5s finished

[2023-12-08 13:21:26] Features: 8/5 -- score: 0.23335507902592595[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.4s finished

[2023-12-08 13:21:26] Features: 7/5 -- score: 0.23269190955567068[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.3s finished

[2023-12-08 13:21:27] Features: 6/5 -- score: 0.23183681423238464[Parallel(n_jobs=1)]: Using backend Sequential
Backend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.2s finished

[2023-12-08 13:21:27] Features: 5/5 -- score: 0.23083923463790185
```

In [144…
```
sfs2.k_feature_names_
```

Out[144]:
```
('online_order', 'rate', 'votes', 'rest_type', 'rate_mul_votes')
```

As we can see that Forward Feature Selection and Backward Feature Selection methods are giving the same features. So we can conclude that these features are more significant in indenticing the patterns.

ExhaustiveFeatureSelector

from mlxtend.feature_selection import ExhaustiveFeatureSelector as EFS

In [146…
```
efs = EFS(LinearRegression(),min_features=1,max_features=5,scoring='r2',cv=5,n_jobs=-1)
efs.fit(X_train_interaction,y_train)
```

```
Features: 6884/6884
```

Out[146]:
```
ExhaustiveFeatureSelector(estimator=LinearRegression(),
                          feature_groups=[[0], [1], [2], [3], [4], [5], [6],
                                          [7], [8], [9], [10], [11], [12], [13],
                                          [14], [15]],
                          max_features=5,  n_jobs=-1,  scoring='r2')
```

In [147…
```
efs.best_feature_names_
```

Out[147]:
```
('online_order', 'rate', 'votes', 'rest_type', 'rate_mul_votes')
```

In [148…
```
efs.best_score_
```

Out[148]:
```
0.2308392346379037
```

Let's build the model with these features

Let's create a variable to store the important features.

In [149…
```
imp_vars_forward = [imp_vars_forward[0][i] for i in range(len(imp_vars_forward[0]))]
```

In [150…
```
imp_vars_forward
```

Out[150]:
```
['online_order', 'rate', 'votes', 'rest_type', 'rate_mul_votes']
```

In [151…
```
linreg_model_with_forward_selection  =  sm.OLS(y_train,  X_train_interaction[imp_vars_forward]).fit()

# to print the summary output
print(linreg_model_with_forward_selection.summary())
```

```
                                 OLS Regression Results
==============================================================================================
Dep. Variable:     approx_cost(for two people)   R-squared (uncentered):                   0.868
Model:                                     OLS   Adj. R-squared (uncentered):              0.868
Method:                          Least Squares   F-statistic:                          3.868e+04
Date:                         Fri, 08 Dec 2023   Prob (F-statistic):                        0.00
Time:                                 13:27:15   Log-Likelihood:                      -1.9029e+05
No. Observations:                        29538   AIC:                                   3.806e+05
Df Residuals:                            29533   BIC:                                   3.806e+05
Df Model:                                    5
Covariance Type:                     nonrobust
================ ==========================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------
online_order      30.2895      1.859     16.297      0.000      26.647      33.932
rate             118.4032      0.695    170.333      0.000     117.041     119.766
votes              0.5957      0.019     31.077      0.000       0.558       0.633
rest_type         -1.5361      0.037    -42.074      0.000      -1.608      -1.465
rate_mul_votes    -0.0019   8.58e-05    -22.063      0.000      -0.002      -0.002
================ ==========================================================================
Omnibus:                      666.602   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              711.968
Skew:                           0.378   Prob(JB):                     2.50e-155
Kurtosis:                       3.085   Cond. No.                      5.24e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 5.24e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Predict the values using test set

In [152…
```
# predict the 'charges' using predict()
predicted_claim = linreg_model_with_forward_selection.predict(X_test_interaction[imp_vars_forward])

# extract the 'charges' values from the test data
actual_claim = y_test
```

Compute model accuracy measures

Now we calculate accuray measures like Root-mean-square-error (RMSE), R-squared and Adjusted R-squared.

In [153…
```
# calculate rmse using rmse()
linreg_model_with_forward_selection_rmse  =  rmse(actual_claim,  predicted_claim)

# calculate R-squared using rsquared
linreg_model_with_forward_selection_rsquared  =  linreg_model_with_forward_selection.rsquared

# calculate Adjusted R-Squared using rsquared_adj
linreg_model_with_forward_selection_rsquared_adj = linreg_model_with_forward_selection.rsquared_adj
```

Tabulate the results

In [154…
```
# compile the required information
linreg_with_interaction_metrics_sfs  =  pd.Series({
                     'Model_Name': "linreg_model_with_forward_selection",
                     'RMSE': linreg_model_with_forward_selection_rmse,
                     'R-Squared':  linreg_model_with_forward_selection_rsquared,
                     'Adj. R-Squared': linreg_model_with_forward_selection_rsquared_adj
                   })

# append our result table using append()
# ignore_index=True: does not use the index labels
```

```
# python can only append a Series if ignore_index=True or if the S eries has a name
score_card = score_card.append(linreg_with_interaction_metrics_sfs  , ignore_index = True)

# print the result table
score_card
```

Out[154]:

	Model_Name	R-Squared	Adj. R-Squared	RMSE
0	Linreg full model without log of target variable	0.228539	0.228304	146.719090
1	linreg_with_interaction	0.234757	0.234395	146.719090
2	Linear Regression SGD	0.242687	0.241763	146.283280
3	Linear Regression SGD	0.219221	0.218268	148.532397
4	linreg_model_with_forward_selection	0.867539	0.867517	151.117076

As we can see that R2 and Adjusted R2 scores are the same and they have gone up significantly

In [ ]:

What did you learn from solving and how do you plan on using it in the future?

- **Understanding the Problem Domain:**
  - **Learning Objective:** Gained a deep understanding of the problem solved and the domain it belongs to.
  - **Future Use:** Apply this understanding to approach and solve similar problems in the future. A strong domain knowledge foundation improves our ability to design effective models.
- **Data Handling and Preprocessing:**
  - **Learning Objective:** Learned to clean, preprocess, and transform raw data into a suitable format for modeling.
  - **Future Use:** Apply these skills to handle new datasets efficiently. Understand the importance of data quality and the impact it has on model performance.
- **Model Selection and Evaluation:**
  - **Learning Objective:** Learned to choose appropriate models, train them, and evaluated their performance using relevant metrics.
  - **Future Use:** Apply model selection techniques to different problems. Understand the trade-offs between various algorithms and how to interpret evaluation metrics.
- **Feature Engineering:**
  - **Learning Objective:** Understood the importance of feature engineering in improving model performance.
  - **Future Use:** Apply creative feature engineering techniques to enhance model predictions in future projects. Recognize the significance of selecting and creating relevant features.
- **Hyperparameter Tuning:**
  - **Learning Objective:** Learned to optimize model hyperparameters for better performance.
  - **Future Use:** Apply hyperparameter tuning to new models. Understand the impact of different hyperparameters on model behavior and performance.
- **Model Interpretability:**
  - **Learning Objective:** Understood how to interpret and explain model predictions.

- **Future Use:** Apply model interpretability techniques to new models. Communicate model insights effectively, especially in situations where model explanations are crucial.

- **Practical Problem-Solving Skills:**
  - **Learning Objective:** Develop practical problem-solving skills by working on real-world projects.
  - **Future Use:** Apply problem-solving skills to new challenges. Understand the importance of adaptability and creativity in approaching machine learning problems.
- **Communication Skills:**
  - **Learning Objective:** Learn to communicate complex technical concepts to non-technical stakeholders.
  - **Future Use:** Apply effective communication skills when presenting and explaining model results to stakeholders. Bridge the gap between technical and non-technical audiences.