

# Group-09 Zomato Data Analysis

```
In [78]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [79]: df = pd.read_csv(r"D:\zomato.csv")
pd.set_option("display.max_columns", None)
df.head()
```

```
Out[79]:
```

	url	address	name	online_order	book_t
0	https://www.zomato.com/bangalore/jalsa-banasha...	942, 21st Main Road, 2nd Stage, Banashankari, ...	Jalsa	Yes	
1	https://www.zomato.com/bangalore/spice-elephan...	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	
2	https://www.zomato.com/SanchurroBangalore?cont...	1112, Next to KIMS Medical College, 17th Cross	San Churro Cafe	Yes	

## Data Cleaning

### 1.to check the no. of columns

```
In [80]: df.columns
```


```
Out[80]: Index(['url', 'address', 'name', 'online_order', 'book_table', 'rate', 'votes',
               'phone', 'location', 'rest_type', 'dish_liked', 'cuisines',
               'approx_cost(for two people)', 'reviews_list', 'menu_item',
               'listed_in(type)', 'listed_in(city)'],
              dtype='object')
```

## 2. drop the unnecessary columns from the dataset

```
In [81]: df = df.drop(['url', 'address', 'phone', 'menu_item', 'listed_in(city)', 'review'])
df.head(2)
```

```
Out[81]:
```

	name	online_order	book_table	rate	votes	location	rest_type	dish_liked	cuisin
0	Jalsa	Yes	Yes	4.1/5	775	Banashankari	Casual Dining	Pasta, Lunch Buffet, Masala Papad, Paneer Laja...	Nc Indi Mugh Chine
1	Spice Elephant	Yes	No	4.1/5	787	Banashankari	Casual Dining	Momos, Lunch Buffet, Chocolate Nirvana, Thai G...	Chine Nc Indi T



## 3.to check data

```
In [82]: df.shape
```

```
Out[82]: (51717, 11)
```

```
In [83]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51717 entries, 0 to 51716
Data columns (total 11 columns):
#   Column                                          Non-Null Count  Dtype
---  -
0   name                                           51717 non-null  object
1   online_order                                  51717 non-null  object
2   book_table                                    51717 non-null  object
3   rate                                           43942 non-null  object
4   votes                                          51717 non-null  int64
5   location                                       51696 non-null  object
6   rest_type                                     51490 non-null  object
7   dish_liked                                    23639 non-null  object
8   cuisines                                       51672 non-null  object
9   approx_cost(for two people)                  51371 non-null  object
10  listed_in(type)                               51717 non-null  object
dtypes: int64(1), object(10)
memory usage: 4.3+ MB
```

## 4.to check the null value (True as null and False as notnull)

```
In [84]: df.isna().sum()
```

```
Out[84]: name                0
online_order                0
book_table                 0
rate                    7775
votes                    0
location                 21
rest_type                227
dish_liked              28078
cuisines                 45
approx_cost(for two people)  346
listed_in(type)           0
dtype: int64
```

## 5.Calculating the missing percentage

```
In [85]: missing_percentage = df.isna().sum().sort_values(ascending = False) / len(df)
missing_percentage
```

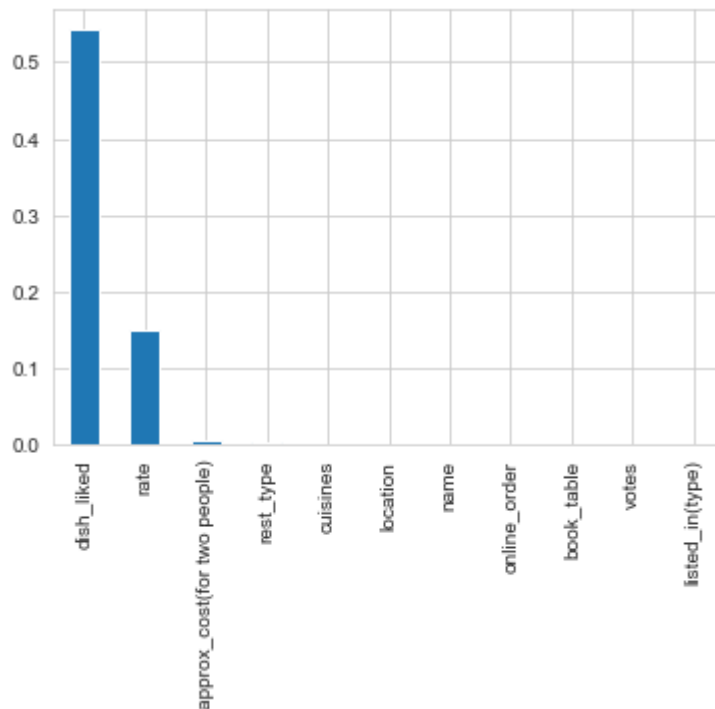
```
Out[85]: dish_liked          0.542916
rate              0.150337
approx_cost(for two people)  0.006690
rest_type         0.004389
cuisines          0.000870
location          0.000406
name              0.000000
online_order      0.000000
book_table        0.000000
votes             0.000000
listed_in(type)   0.000000
dtype: float64
```

```
In [86]: missing_percentage[missing_percentage != 0]
```

```
Out[86]: dish_liked          0.542916
rate              0.150337
approx_cost(for two people)  0.006690
rest_type         0.004389
cuisines          0.000870
location          0.000406
dtype: float64
```

```
In [87]: missing_percentage.plot(kind = 'bar')
```

```
Out[87]: <AxesSubplot:>
```



## 6. find uniuqe value

```
In [88]: df.rate.unique()
```

```
Out[88]: array(['4.1/5', '3.8/5', '3.7/5', '3.6/5', '4.6/5', '4.0/5', '4.2/5',  
                '3.9/5', '3.1/5', '3.0/5', '3.2/5', '3.3/5', '2.8/5', '4.4/5',  
                '4.3/5', 'NEW', '2.9/5', '3.5/5', nan, '2.6/5', '3.8 /5', '3.4/5',  
                '4.5/5', '2.5/5', '2.7/5', '4.7/5', '2.4/5', '2.2/5', '2.3/5',  
                '3.4 /5', '-', '3.6 /5', '4.8/5', '3.9 /5', '4.2 /5', '4.0 /5',  
                '4.1 /5', '3.7 /5', '3.1 /5', '2.9 /5', '3.3 /5', '2.8 /5',  
                '3.5 /5', '2.7 /5', '2.5 /5', '3.2 /5', '2.6 /5', '4.5 /5',  
                '4.3 /5', '4.4 /5', '4.9/5', '2.1/5', '2.0/5', '1.8/5', '4.6 /5',  
                '4.9 /5', '3.0 /5', '4.8 /5', '2.3 /5', '4.7 /5', '2.4 /5',  
                '2.1 /5', '2.2 /5', '2.0 /5', '1.8 /5'], dtype=object)
```

## 7. Dropping null values

```
In [89]: df.dropna(subset=['rate'],axis=0,inplace=True)
```

## 8.split the data

```
In [90]: def split(x):  
         return x.split('/')[0].strip()
```

```
In [91]: df['rate'] = df['rate'].apply(split)
```

```
In [92]: df['rate'].replace(['NEW', '-'],0,inplace=True)
```

```
In [93]: df['rate'] = df['rate'].astype(float)
```

## 9.checking the data after cleaning

```
In [94]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 43942 entries, 0 to 51716
Data columns (total 11 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   name                                  43942 non-null  object
1   online_order                         43942 non-null  object
2   book_table                           43942 non-null  object
3   rate                                 43942 non-null  float64
4   votes                                43942 non-null  int64
5   location                             43942 non-null  object
6   rest_type                            43791 non-null  object
7   dish_liked                           23609 non-null  object
8   cuisines                             43931 non-null  object
9   approx_cost(for two people)          43690 non-null  object
10  listed_in(type)                      43942 non-null  object
dtypes: float64(1), int64(1), object(9)
memory usage: 4.0+ MB
```

## Data Manupulation

### 1.to know the statistics summary

```
In [95]: df.describe()
```

```
Out[95]:
```

	rate	votes
count	43942.000000	43942.000000
mean	3.508698	333.430954
std	0.925639	862.180633
min	0.000000	0.000000
25%	3.300000	17.000000
50%	3.700000	64.000000
75%	4.000000	254.000000
max	4.900000	16832.000000

## 2. checking how many numeric columns

```
In [96]: numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
numeric_df = df.select_dtypes(include=numerics)
len(numeric_df.columns)
```

Out[96]: 2

## 3. Calculating the percentage of online orders

```
In [97]: online_order_percentage = (df['online_order'].value_counts(normalize=True)
print(online_order_percentage)

# Get the labels and values from the Series
labels = online_order_percentage.index.astype(str).to_list()
values = online_order_percentage.to_list()

# Create a horizontal bar chart with color
colors = ['skyblue', 'lightcoral']
plt.barh(online_order_percentage.index.astype(str), online_order_percentage)

# Add percentage labels on each bar
for index, value in enumerate(online_order_percentage):
    plt.text(value + 0.5, index, f"{value}%", va='center')

# Set labels and title
plt.xlabel("Percentage")
plt.title("Percentage of Online Orders")

# Remove y-axis ticks for cleaner look
plt.yticks([])

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

Yes 64.42  
No 35.58  
Name: online\_order, dtype: float64



## 4. Percentage of restaurants in the dataset offer online ordering

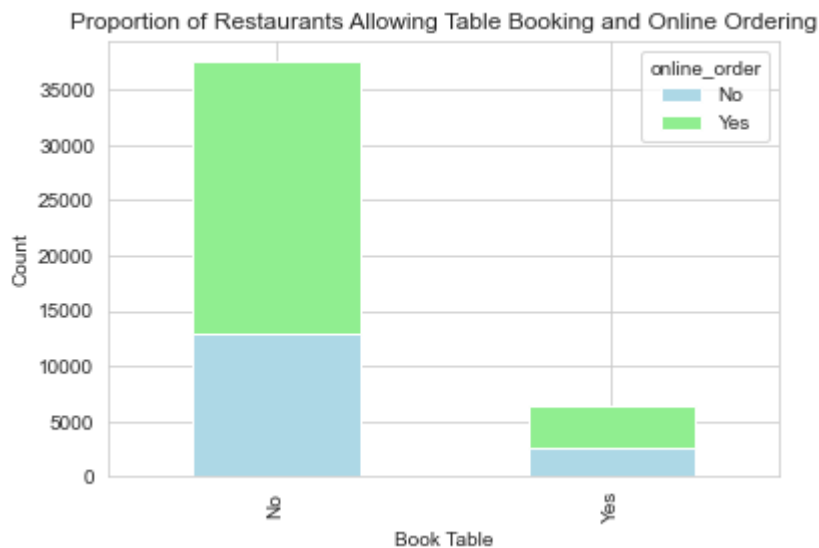
```
In [98]: online_order_percentage = (df['online_order'] == 'Yes').mean() * 100  
print(f"Percentage: {online_order_percentage:.2f}% of restaurants offer online ordering")
```

Percentage: 64.42% of restaurants offer online ordering

**Interpretation:** The percentage indicates the extent to which restaurants in the dataset have embraced online ordering, reflecting potential trends in the industry or region. A high percentage suggests a strong customer preference for online ordering, which businesses might need to prioritize to remain competitive. A low percentage could indicate potential gaps in the market for businesses to capitalize on by offering online ordering capabilities. Businesses can use this information to identify and target restaurants that have not yet adopted online ordering, offering potential partnerships or solutions.

## 5. Proportion of restaurants allowing table booking and online ordering

```
In [99]: booking_order_proportion = df.groupby(['book_table', 'online_order']).size()  
booking_order_proportion.plot(kind='bar', stacked=True, color=['lightblue', 'lightgreen'],  
plt.title('Proportion of Restaurants Allowing Table Booking and Online Ordering')  
plt.xlabel('Book Table')  
plt.ylabel('Count')  
plt.show()
```



## 6. finding the most preferred dish and order type

```
In [100]: preferred_dish_order = df.groupby('location').agg({'dish_liked': pd.Series.

# Display the result
print("Most Preferred Dish and Order Type per Location:")
print(preferred_dish_order)
```

Most Preferred Dish and Order Type per Location:

location	dish_liked \
BTM	Biryani
Banashankari	Pasta, Gelato, Garlic Bread, Mojito, Nachos, P...
Banaswadi	Mutton Biryani
Bannerghatta Road	Coffee, French Fries, Lemon Tea, Veg Sandwich
Basavanagudi	[Masala Dosa, Pizza, Fries, Anjeer Ice Cream, ...
...	...
West Bangalore	Sandwich, Salads, Pizza, Sandwiches, Salad, Pa...
Whitefield	Chicken Biryani
Wilson Garden	Friendly Staff
Yelahanka	Appam, Fish Curry, Fish Fry, Pothichoru
Yeshwantpur	[Badam Milk, Chaat, Sandwich, Noodles, Ghee Do...

location	online_order
BTM	Yes
Banashankari	Yes
Banaswadi	Yes
Bannerghatta Road	Yes
Basavanagudi	Yes
...	...
West Bangalore	No
Whitefield	Yes
Wilson Garden	Yes
Yelahanka	Yes
Yeshwantpur	Yes

[92 rows x 2 columns]

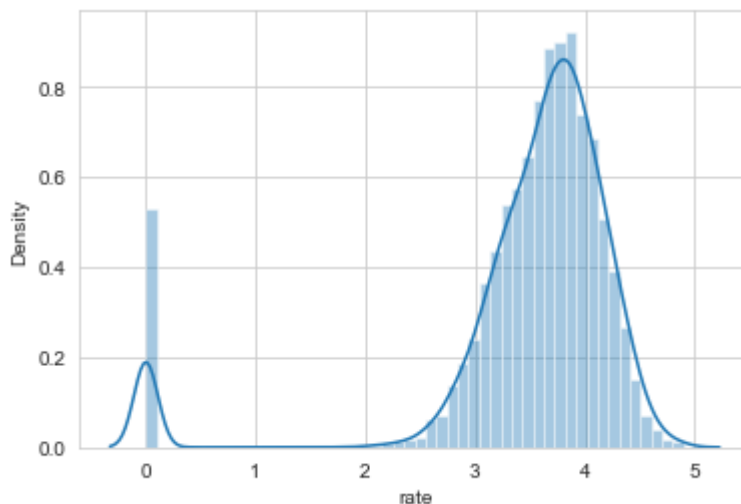


## 7. Distribution of rating

```
In [101]: sns.set_style('whitegrid')
sns.distplot(df['rate'])
plt.show
```

```
c:\Users\DELL\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
Out[101]: <function matplotlib.pyplot.show(close=None, block=None)>
```



This is a not normal distributionas we can see spike in '0's i.e the new restaurants

## 8. Top restaurant chains

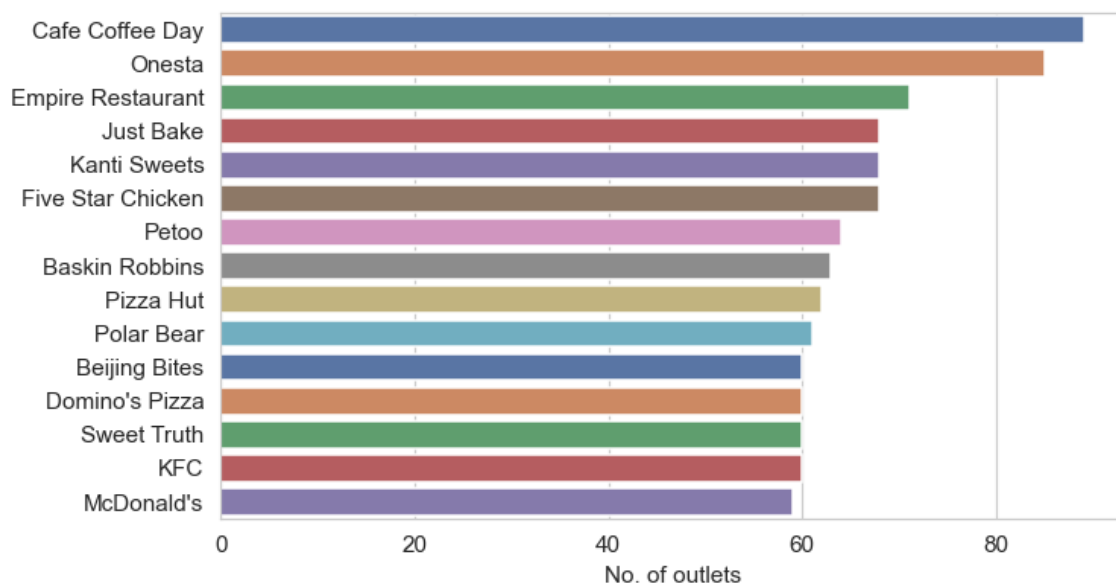
```
In [102]: df['name'].value_counts()
```

```
Out[102]: Cafe Coffee Day      89
Onesta      85
Empire Restaurant  71
Just Bake    68
Kanti Sweets  68
..
Clean Eats    1
Sumai Foods   1
Pailwan House 1
Desi Chats    1
Calicut Mess House 1
Name: name, Length: 7162, dtype: int64
```

```
In [103]: plt.figure(figsize=(7,4),dpi=110)
chains = df['name'].value_counts()[0:15]
sns.barplot(chains,chains.index,palette = 'deep')
plt.xlabel("No. of outlets")
plt.show()
```

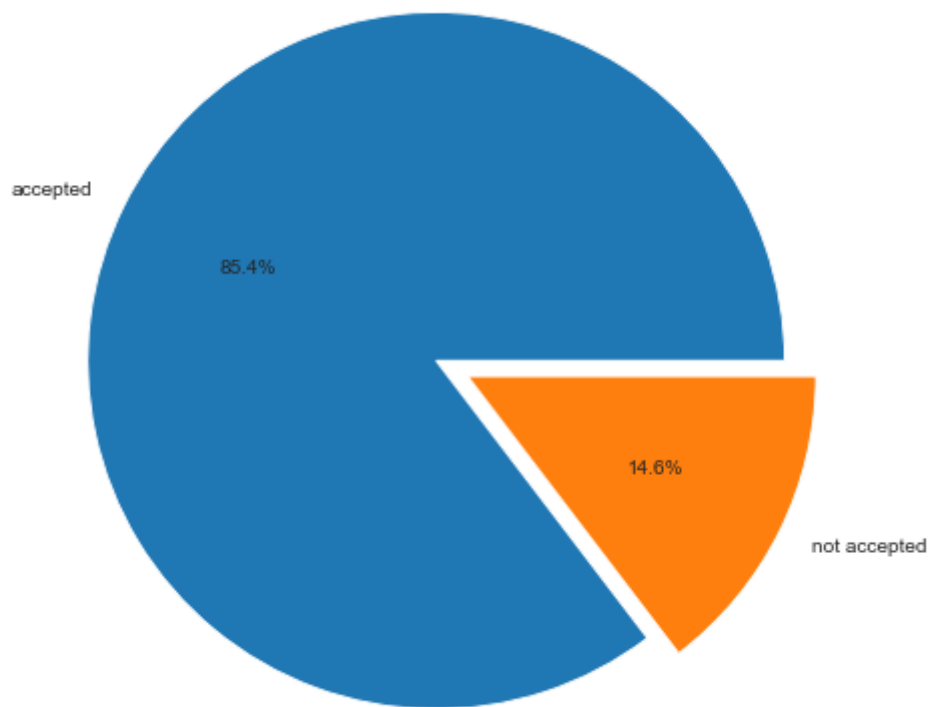
c:\Users\DELL\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```



## 9. Count the number of restaurants with and without booking tables

```
In [104]: x = df.book_table.value_counts()
labels = ['accepted', 'not accepted']
plt.figure(figsize=(8,8))
plt.pie(x, labels=labels, explode=[0.0, 0.1], autopct='%1.1f%%')
plt.show()
```



**Interpretation:** The chart will visually depict the proportion of restaurants that offer booking capabilities versus those that don't. Restaurants can use booking data to: Improve table turnover, Manage wait times, Reduce no-shows. Businesses can identify underserved niches (e.g., high-demand areas with few bookable options) or target specific customer segments who value booking capabilities.

## 10. Analysis of type of restaurants

```
In [105]: Rest_type = df.rest_type.unique()
len(Rest_type)
```

```
Out[105]: 88
```

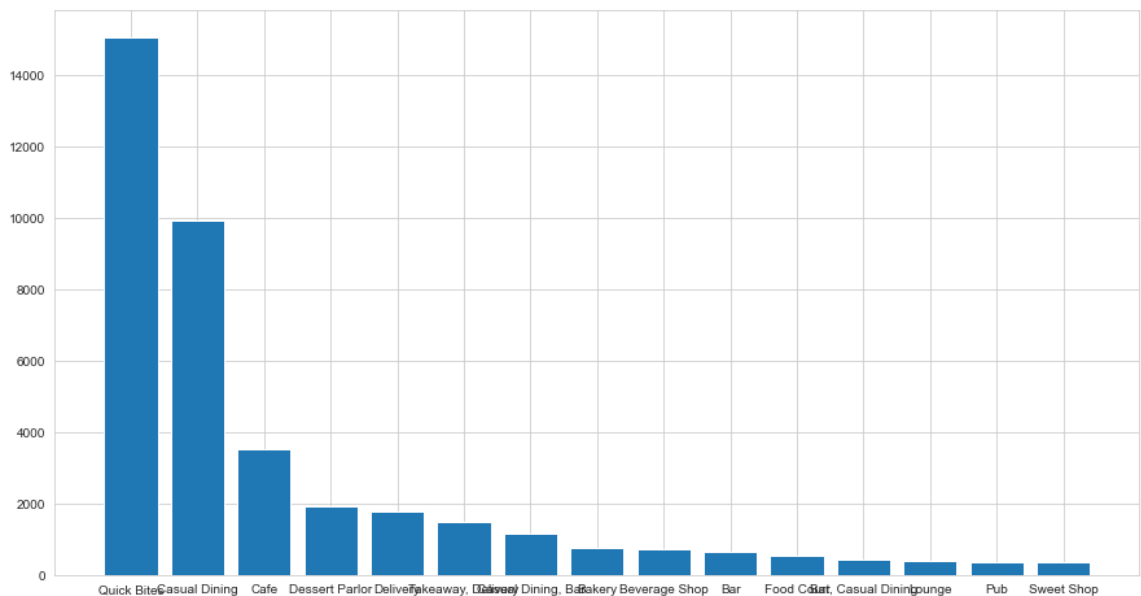
```
Out[106]: array(['Casual Dining', 'Cafe, Casual Dining', 'Quick Bites',  
                  'Casual Dining, Cafe', 'Cafe', 'Quick Bites, Cafe',  
                  'Cafe, Quick Bites', 'Delivery', 'Mess', 'Dessert Parlor',  
                  'Bakery, Dessert Parlor', 'Pub', 'Bakery', 'Takeaway, Delivery',  
                  'Fine Dining', 'Beverage Shop', 'Sweet Shop', 'Bar',  
                  'Dessert Parlor, Sweet Shop', 'Bakery, Quick Bites',  
                  'Sweet Shop, Quick Bites', 'Kiosk', 'Food Truck',  
                  'Quick Bites, Dessert Parlor', 'Beverage Shop, Quick Bites',  
                  'Beverage Shop, Dessert Parlor', 'Takeaway', 'Pub, Casual Dinin  
g',  
                  'Casual Dining, Bar', 'Dessert Parlor, Beverage Shop',  
                  'Quick Bites, Bakery', 'Microbrewery, Casual Dining', 'Lounge',  
                  'Bar, Casual Dining', 'Food Court', 'Cafe, Bakery', nan, 'Dhaba',  
                  'Quick Bites, Sweet Shop', 'Microbrewery',  
                  'Food Court, Quick Bites', 'Quick Bites, Beverage Shop',  
                  'Pub, Bar', 'Casual Dining, Pub', 'Lounge, Bar',  
                  'Dessert Parlor, Quick Bites', 'Food Court, Dessert Parlor',  
                  'Casual Dining, Sweet Shop', 'Food Court, Casual Dining',  
                  'Casual Dining, Microbrewery', 'Lounge, Casual Dining',
```

```
Insight 3: Most common restaurant types:
```

Quick Bites	15073
Casual Dining	9923
Cafe	3527
Dessert Parlor	1939
Delivery	1791
...	
Bakery, Food Court	2
Cafe, Food Court	2
Food Court, Beverage Shop	2
Dessert Parlor, Kiosk	2
Quick Bites, Kiosk	1

```
Name: rest type, Length: 87, dtype: int64
```

```
In [109]: plt.figure(figsize=(15,8))
rest_type = df.rest_type.value_counts()[0:15]
plt.bar(rest_type.index,rest_type)
plt.show()
```



This is the distribution of type of restaurants and we observe that Quick bites at the top...further this is the dataset of banglore city so most of the people living there prefer quick bites(fast food chains)

### 13.Which restaurant had the highest voting? (means count of rating given)

```
In [110]: voting = df.groupby('name')[['votes']].mean()
voting
```

Out[110]:

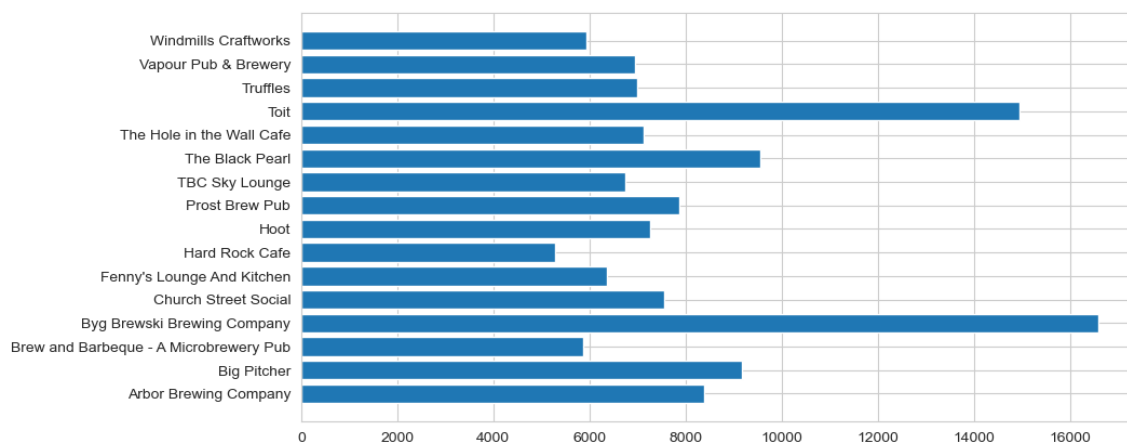
	votes
name	
#FeelTheROLL	7.000000
#L-81 Cafe	48.000000
#refuel	37.000000
1000 B.C	49.000000
1000 B.C	41.000000
...	...
i-Bar - The Park Bangalore	619.727273
iFruit Live Ice Creams	10.000000
iSpice Resto Cafe	30.142857
nu.tree	206.142857
re:cess - Hilton Bangalore Embassy GolfLinks	146.000000

7162 rows × 1 columns

```
In [111]: voting['votes'].describe()
```

```
Out[111]: count      7162.000000
mean        201.484245
std         607.015608
min          0.000000
25%         10.400000
50%         38.750000
75%        146.797414
max        16588.500000
Name: votes, dtype: float64
```

```
In [112]: high_vote = voting[voting['votes']>5000]
plt.figure(figsize=(10,5),dpi=100)
plt.barh(high_vote.index,high_vote['votes'])
plt.show()
```

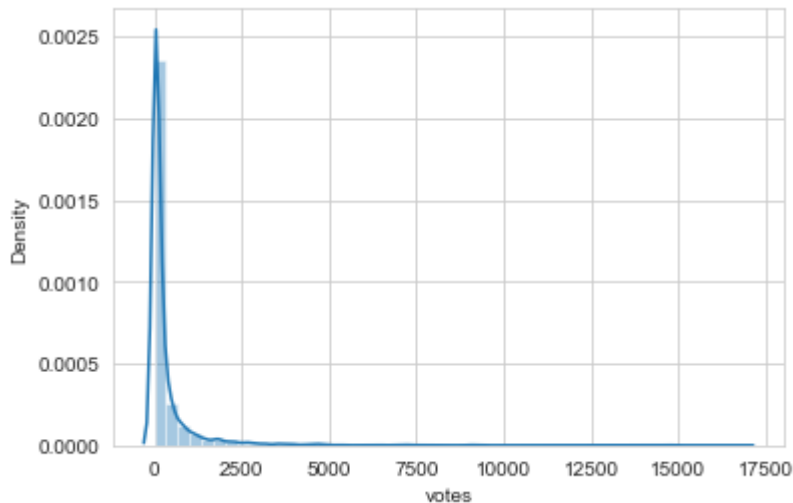


**This is a barplot for the count of ratings given by the consumers to a particular restaurants.**

## 14. analyse the highest rated restaurant and worst rated restaurants based on the number of ratings given

```
In [113]: sns.distplot(df['votes'])  
plt.show()
```

```
c:\Users\DELL\anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).  
  warnings.warn(msg, FutureWarning)
```



```
In [114]: voting_rating = df.groupby('name')[['votes']].mean().sort_values('votes', ascending=False)  
voting_rating
```

```
Out[114]:
```

	votes
Byg Brewski Brewing Company	16588.500000
Toit	14956.000000
The Black Pearl	9562.333333
Big Pitcher	9164.500000
Arbor Brewing Company	8396.545455
...	...
NutriFit	0.000000
Cross Roads Inn	0.000000
Crown Resto	0.000000
Nourich	0.000000
Lazzet Lee	0.000000

7162 rows × 1 columns

```
In [115]: voting_rating['name'] = voting_rating.index
```

```
In [116]: voting_rating = voting_rating.reset_index(drop=True)
voting_rating.head()
```

```
Out[116]:
```

	votes	name
0	16588.500000	Byg Brewski Brewing Company
1	14956.000000	Toit
2	9562.333333	The Black Pearl
3	9164.500000	Big Pitcher
4	8396.545455	Arbor Brewing Company

```
In [117]: voting_rating = pd.merge(voting_rating,df[['rate','name']])
voting_rating
```

```
Out[117]:
```

	votes	name	rate
0	16588.5	Byg Brewski Brewing Company	4.9
1	16588.5	Byg Brewski Brewing Company	4.9
2	16588.5	Byg Brewski Brewing Company	4.9
3	16588.5	Byg Brewski Brewing Company	4.9
4	16588.5	Byg Brewski Brewing Company	4.9
...	...	...	...
43937	0.0	Lazzet Lee	0.0
43938	0.0	Lazzet Lee	0.0
43939	0.0	Lazzet Lee	0.0
43940	0.0	Lazzet Lee	0.0
43941	0.0	Lazzet Lee	0.0

43942 rows × 3 columns

```
In [118]: vote_top =voting_rating[voting_rating['votes']>5000]
vote_top.head()
```

```
Out[118]:
```

	votes	name	rate
0	16588.5	Byg Brewski Brewing Company	4.9
1	16588.5	Byg Brewski Brewing Company	4.9
2	16588.5	Byg Brewski Brewing Company	4.9
3	16588.5	Byg Brewski Brewing Company	4.9
4	16588.5	Byg Brewski Brewing Company	4.9



```
In [119]: vote_top = vote_top.groupby('name')[['rate']].mean().sort_values('rate', ascending=False)
          vote_top
```

```
Out[119]:
```

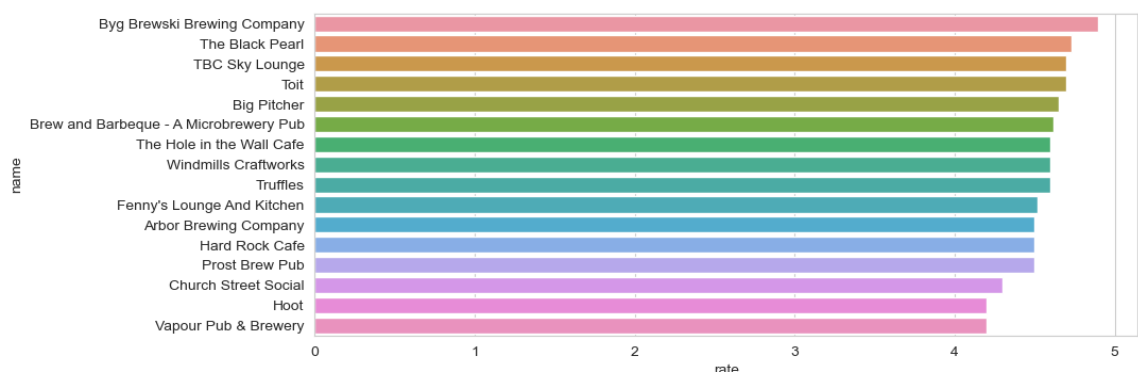
	rate
<b>Byg Brewski Brewing Company</b>	4.900000
<b>The Black Pearl</b>	4.727778
<b>TBC Sky Lounge</b>	4.700000
<b>Toit</b>	4.700000
<b>Big Pitcher</b>	4.650000
<b>Brew and Barbeque - A Microbrewery Pub</b>	4.620000
<b>The Hole in the Wall Cafe</b>	4.600000
<b>Windmills Craftworks</b>	4.600000
<b>Truffles</b>	4.595349
<b>Fenny's Lounge And Kitchen</b>	4.521429
<b>Arbor Brewing Company</b>	4.500000
<b>Hard Rock Cafe</b>	4.500000
<b>Prost Brew Pub</b>	4.500000
<b>Church Street Social</b>	4.300000
<b>Hoot</b>	4.200000
<b>Vapour Pub &amp; Brewery</b>	4.200000

## 15.highest rate based on most number of votes

```
In [120]: plt.figure(figsize = (10,4),dpi=100)
          sns.barplot(vote_top['rate'],vote_top.index,orient='h')
          plt.show()
```

c:\Users\DELL\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```



interpretation: 'Byg Brewski Brewing Comapny' is the most voted restaurant with the average rating of 4.9

```
In [121]: bad_count = voting_rating[(voting_rating['rate']<3) & (voting_rating['rate']>4.5)]
bad_count
```

```
Out[121]:
```

	votes	name	rate
3725	967.681818	Kritunga Restaurant	2.6
3726	967.681818	Kritunga Restaurant	2.9
3727	967.681818	Kritunga Restaurant	2.9
3742	967.681818	Kritunga Restaurant	2.7
3743	967.681818	Kritunga Restaurant	2.7
...	...	...	...
42110	4.000000	Special Biryani	2.9
42111	4.000000	Special Biryani	2.9
42198	4.000000	Sandesh Restaurant	2.9
42311	2.500000	Kushi Meals	2.8
42312	2.500000	Kushi Meals	2.8

2257 rows × 3 columns

## 16.Location in which have highest number of restaurants

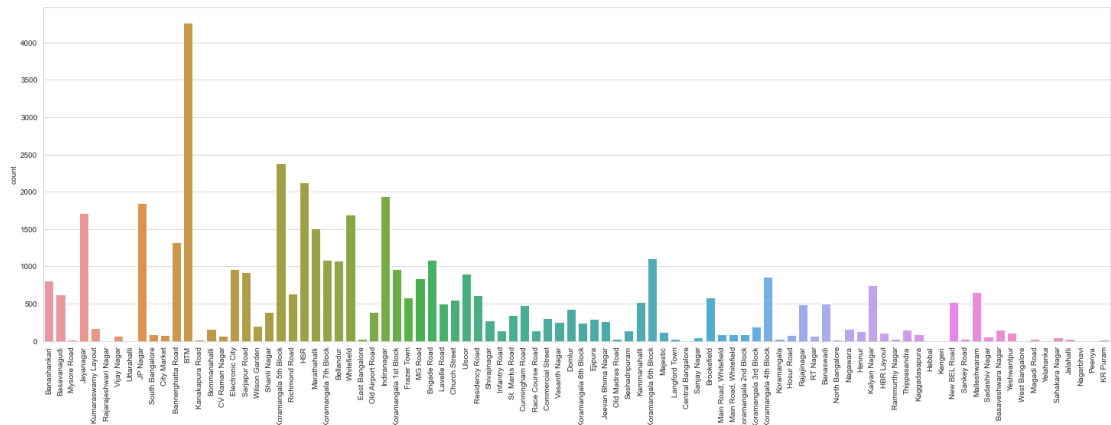
```
In [130]: location_restro = df[['location']].value_counts()[0:15]
location_restro
```

```
Out[130]: location
BTM                                4261
Koramangala 5th Block              2381
HSR                                2128
Indiranagar                        1936
JP Nagar                           1849
Jayanagar                          1718
Whitefield                         1693
Marathahalli                       1503
Bannerghatta Road                  1324
Koramangala 6th Block              1111
Koramangala 7th Block              1089
Brigade Road                       1084
Bellandur                          1078
Koramangala 1st Block               965
Electronic City                     964
dtype: int64
```

```
In [131]: plt.figure(figsize=(25,8))
sns.countplot(df['location'])
plt.xticks(rotation=90)
plt.show()
```

c:\Users\DELL\anaconda3\lib\site-packages\seaborn\\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

```
warnings.warn(
```



## 17. Average cost for two people in different locations

```
In [132]: avg_cost_by_location = df.groupby('location')['approx_cost(for two people)']
print("Insight 8: Average cost for two people in different locations:")
print(avg_cost_by_location)
```

Insight 8: Average cost for two people in different locations:

location

Sankey Road 2582.692308

Lavelle Road 1344.320487

Race Course Road 1315.328467

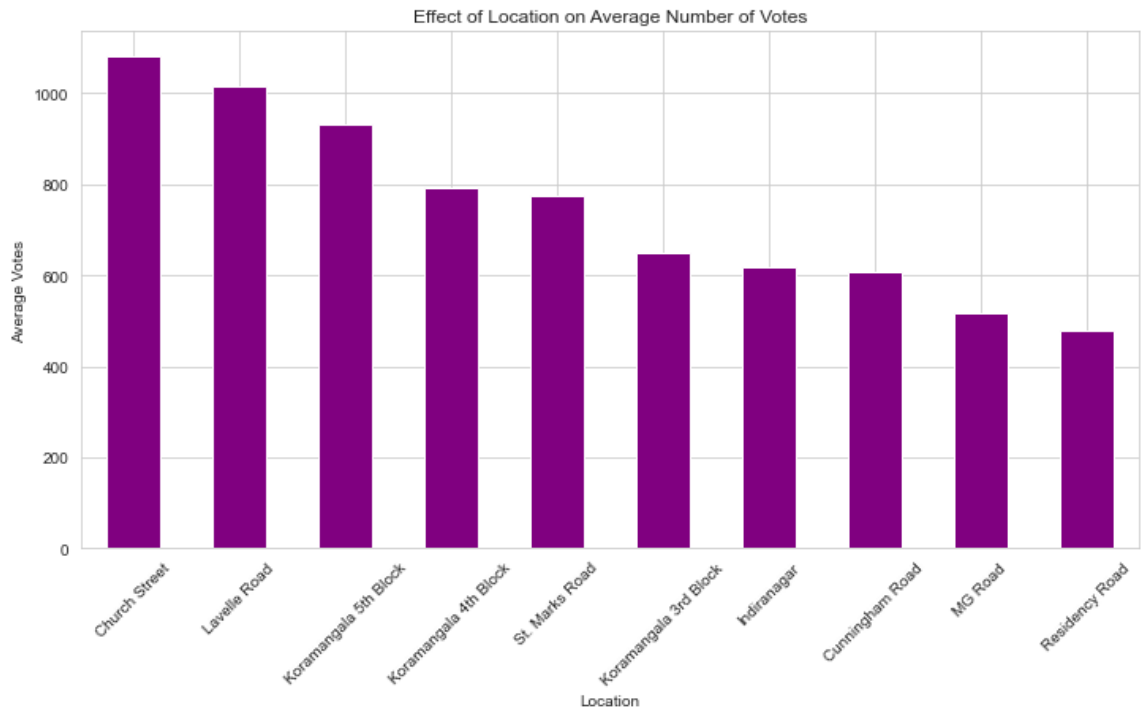
MG Road 1228.484108

Infantry Road 1072.857143

Name: approx\_cost(for two people), dtype: float64

## 18. Effect of location on the number of votes

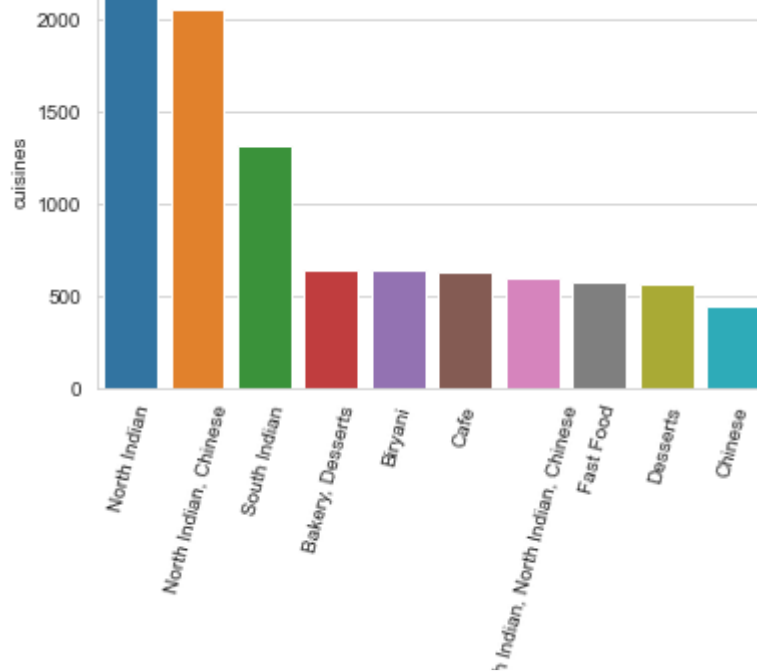
```
In [133]: avg_votes_by_location = df.groupby('location')['votes'].mean().sort_values(
plt.figure(figsize=(12, 6))
avg_votes_by_location.plot(kind='bar', color='purple')
plt.title('Effect of Location on Average Number of Votes')
plt.xlabel('Location')
plt.ylabel('Average Votes')
plt.xticks(rotation=45)
plt.show()
```



**Interpretation:** The chart will reveal which locations have the highest average ratings, suggesting potential relationships between location and popularity. Businesses can use this information to: Target expansion or marketing efforts in areas with high average ratings, Investigate factors contributing to popularity in those locations, Understand customer preferences and expectations in different areas, Optimize restaurant placement and strategies based on location insights.

## 19. Identify the top 10 cuisines

```
In [152]: cusin = df.cuisines.value_counts()[0:10]
print(cusin)
sns.barplot(cusin.index, cusin)
plt.xticks(rotation=75)
plt.show()
```



## 20. Average cost by type

```
In [136]: avg_cost_by_type = df.groupby(['listed_in(type)', 'cuisines'])['approx_cost']

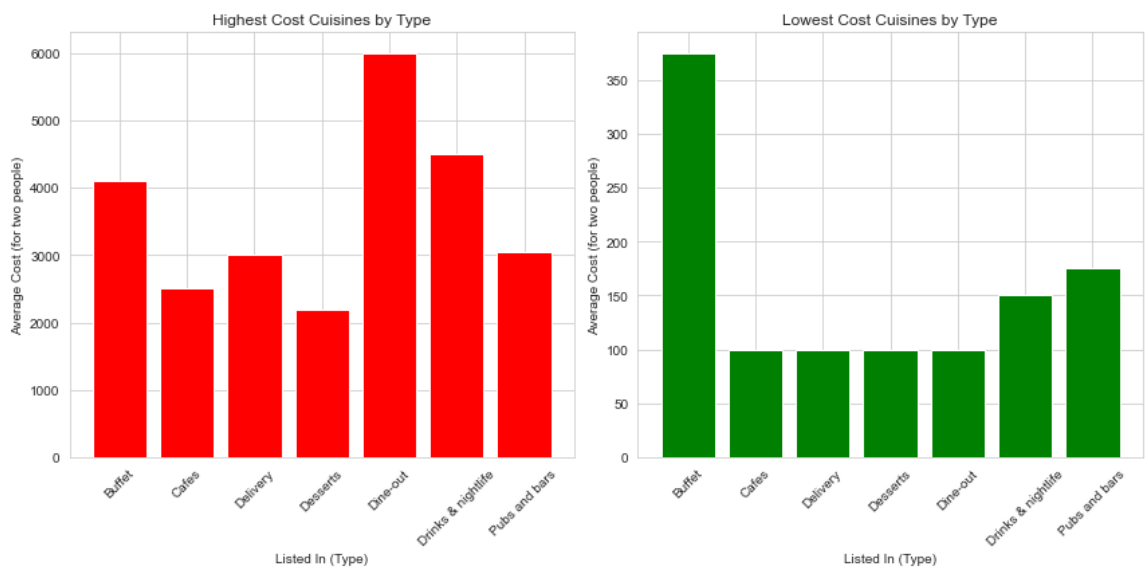
highest_cost_cuisine = avg_cost_by_type.loc[avg_cost_by_type.groupby('listed_in(type)').max()]
lowest_cost_cuisine = avg_cost_by_type.loc[avg_cost_by_type.groupby('listed_in(type)').min()]

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.bar(highest_cost_cuisine['listed_in(type)'], highest_cost_cuisine['approx_cost'])
plt.title('Highest Cost Cuisines by Type')
plt.xlabel('Listed In (Type)')
plt.ylabel('Average Cost (for two people)')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
plt.bar(lowest_cost_cuisine['listed_in(type)'], lowest_cost_cuisine['approx_cost'])
plt.title('Lowest Cost Cuisines by Type')
plt.xlabel('Listed In (Type)')
plt.ylabel('Average Cost (for two people)')
plt.xticks(rotation=45)

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



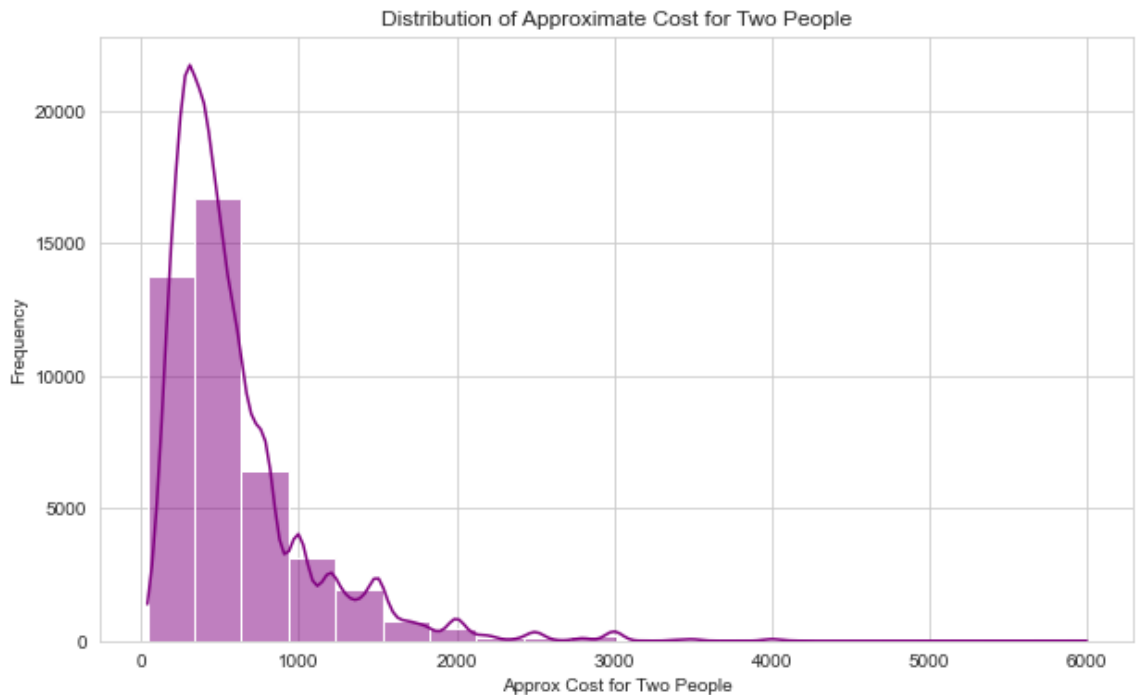
## 21. Checking the distribution for cost of 2 people

```
In [137]: df['approx_cost(for two people)'].isna().sum()
```

```
Out[137]: 252
```

```
In [139]: df.dropna(axis = 0, subset=['approx_cost(for two people)'], inplace = True)
```

```
In [145]: # Distribution of approximate cost for two people
plt.figure(figsize=(10, 6))
sns.histplot(df['approx_cost(for two people)'].dropna(), bins=20, kde=True,
plt.title('Distribution of Approximate Cost for Two People')
plt.xlabel('Approx Cost for Two People')
plt.ylabel('Frequency')
plt.show()
```



**Interpretation:** The chart will show the shape of the distribution, revealing: Common price ranges, Concentrations of restaurants at certain price points, Potential gaps or opportunities in the market. Businesses can use this information to: Target marketing or promotions based on price sensitivity, Inform menu pricing strategie, Identify opportunities for niche pricing or value offerings, Understand customer expectations and spending patterns.

In [149]: `df[df['approx_cost(for two people)'] < 2000]`

Out[149]:

	name	online_order	book_table	rate	votes	location	rest_type	dish_liked
0	Jalsa	Yes	Yes	4.1	775	Banashankari	Casual Dining	Pasta, Lunch Buffet, Masala Papad, Paneer Laja...
1	Spice Elephant	Yes	No	4.1	787	Banashankari	Casual Dining	Momos, Lunch Buffet, Chocolate Nirvana, Thai G...
2	San Churro Cafe	Yes	No	3.8	918	Banashankari	Cafe, Casual Dining	Churros, Cannelloni, Minestrone Soup, Hot Choc...
3	Addhuri Udupi Bhojana	No	No	3.7	88	Banashankari	Quick Bites	Masala Dosa
4	Grand Village	No	No	3.8	166	Basavanagudi	Casual Dining	Panipuri, Gol Gappe
...	...	...	...	...	...	...	...	...
51708	Keys Cafe - Keys Hotel	No	No	2.8	161	Whitefield	Casual Dining, Bar	Salads, Coffee, Breakfast Buffet, Halwa, Chick...
51709	The Farm House Bar n Grill	No	No	3.7	34	Whitefield	Casual Dining, Bar	NaN
51711	Bhagini	No	No	2.5	81	Whitefield	Casual Dining, Bar	Biryani, Andhra Meal
51712	Best Brews - Four Points by Sheraton Bengaluru...	No	No	3.6	27	Whitefield	Bar	NaN
51716	The Nest - The Den Bengaluru	No	No	3.4	13	ITPL Main Road, Whitefield	Bar, Casual Dining	NaN

42710 rows × 11 columns





## 22. Most liked dishes

```
In [150]: most_liked_dishes = df['dish_liked'].str.split(', ').explode().value_counts
print("Insight 9: Most liked dishes:")
print(most_liked_dishes)
```

```
Insight 9: Most liked dishes:
Pasta      3375
Burgers    3001
Cocktails  2805
Pizza      2696
Biryani    2077
Name: dish_liked, dtype: int64
```

## Conclusion

1. Asia kitchen By Mainland China, Spa Cuisine, Byg Brewski Brewing Company, Punjab Grill were the best rated restaurants.
2. Cafe Coffee Day, Onesta, Empire Restaurant, Just Bake, Kanti sweets were the top restaurant chains.
3. 64.4% of restaurants accepts online orders where 35.6% don't accept online orders.
4. Based on most voted restaurants Byg Brewski Brewing Company and The Black Pearl were the best rated restaurants.
5. Based on most voted restaurants Lazeez and Tandoor Hut were the most worst rated restaurants.
6. BTM, Koramangala 5th Block has the most numbers of restaurants.
7. North Indian and Chinese were the top favourite of the people in Bangalore.
8. Cost for 2 people for the most restaurants was varying from 500 to 1000 bucks.