In [130]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams
```

CSCI - 5901 - The Process of Data Science - Summer 2019 SUBMITTED BY: AAKASH PATEL (B00807065) RISHABH DHAWAN (B00826918)

ASSIGNMENT 1:

1.a.

The dataset set zomato.csv is uploaded comprising of 12000 various restaurants obtained after cleaning the raw data and have been stored inside a data frame. (accurate up till 15th march,2019). It describes the various establishments of different restaurants across the Bangalore city and plan on analyzing what are the major factors that affect the establishment of a restaurant in the city. The analysis will rely entirely on what are the various attributes involved within the dataset, and how they will affect the rating of a given restaurant. Following are the attributes which have been found in the dataset:

FEATURE LIST:

1. url: comprises of the weblink of restaurant on Zomato.
2. Address: describes the address of the restaurant in the city
3. name: name of the restaurant
4. online_order: it tell whether the restaurant has the option of ordering online.
5. book_table: its describes whether we have the option to book the table.
6. rate: signifies the rating of a given restaurants.
7. votes: defines the total number of rating of a restaurant on a stipulated date.
8. phone: phone number of the restaurant

9. location: tells the neighborhood of the restaurant which will ultimately define its location in the city.
10. rest_type: type of restaurant
11. dish_liked: favorite dishes of the people in the restaurant.
12. cuisines: type and styles of food.
13. approx_cost(for two people): estimates the cost of eating of two people in a given restaurant.
14. reviews_list: The review list comprises of reviews for a given restaurant. Here each tuple has two sections. One describing the rating and other describing the review written by the customer for that restaurant.
15. menu_item: describe the menu list of the items in the restaurant menu
16. listed_in(type): meal type
17. listed_in(city): comprises of the neighborhood where the restaurant is situated.

In [131]:
```
df=pd.read_csv("E://MACS//Term 2//5901-DSc//DS_Assignment 1//Dataset//z
omato-bangalore-restaurants//zomato.csv")
#df=pd.read_csv("C://Users//risha//Downloads//zomato.csv") #zomato.csv
 data is loaded and read from the local system.
```

2.a

The provided data set(zomato.csv) is loaded and read fro the local system.

In [132]:
```
df.head() #displays first few data records with respect to ease of visu
alising the data set.
```

Out[132]:

	url	address	name	online_order	book_table

	url	address	name	online_order	book_table	r
0	https://www.zomato.com/bangalore/jalsa-banasha...	942, 21st Main Road, 2nd Stage, Banashankari, ...	Jalsa	Yes	Yes	4
1	https://www.zomato.com/bangalore/spice-elephan...	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	No	4
2	https://www.zomato.com/SanchurroBangalore?cont...	1112, Next to KIMS Medical College, 17th Cross...	San Churro Cafe	Yes	No	3
3	https://www.zomato.com/bangalore/addhuri-udupi...	1st Floor, Annakuteera, 3rd Stage, Banashankar...	Addhuri Udupi Bhojana	No	No	3
4	https://www.zomato.com/bangalore/grand-village...	10, 3rd Floor, Lakshmi Associates, Gandhi Baza...	Grand Village	No	No	3

2.b DATA TRENDS (ATTRIBUTE SELECTION JUSTIFICATION)

Examining the data, it can be inferred that the data trends supporting certain features as the most suitable ones for selection for the designing the model:

Name This attribute will be identifying the restaurant. Along with the address, it will be used as a composite key for the data set table.

Address This attribute is very important as it will be eventually used to book the restaurant. Moreover, it will be used by people who want to visit a popular restaurant but need to know the exact location of the restaurant. Later on, name and address will be used together as a composite key for the table data.

Rate This attribute is an important one with respect to finding the ideal place for two persons to visit as it reflects the people’s analysis who have visited that restaurant.

Listed_in(city) This attribute is the next important attribute we have considered since the location also largely impact the reason why a restaurant is visited more frequently or not. This will hence replace the cuisine a particular restraunt serves.

Rest_type This determines how a particular restaurant can determine the population that visits it. Here we see that the quick bites restaurant type is the most famous one and sweet shops are the least popular among the people.

approx_cost(for two people) Since it defines the monetary aspect of the restaurant, it will be one of the most important factor in determining the type of people (depending on economic status) visiting a particular restaurant.

listed_in(type) This attribute describes how a meal is served by a restaurant. Since different people have different choice of getting access to their food, they will prefer the restaurant that satisfies their choice. It could be a buffet, delivery,

In [133]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51717 entries, 0 to 51716
Data columns (total 17 columns):
url                            51717 non-null object
address                        51717 non-null object
name                           51717 non-null object
online_order                   51717 non-null object
book_table                     51717 non-null object
rate                           43942 non-null object
votes                          51717 non-null int64
phone                          50509 non-null object
```

```
location                          51696 non-null object
rest_type                         51490 non-null object
dish_liked                        23639 non-null object
cuisines                          51672 non-null object
approx_cost(for two people)       51371 non-null object
reviews_list                      51717 non-null object
menu_item                         51717 non-null object
listed_in(type)                   51717 non-null object
listed_in(city)                   51717 non-null object
dtypes: int64(1), object(16)
memory usage: 6.7+ MB
```

In [134]:
```
print(df.describe())
```

```
              votes
count  51717.000000
mean     283.697527
std      803.838853
min        0.000000
25%        7.000000
50%       41.000000
75%      198.000000
max    16832.000000
```

BAR CHARTS OF VARIOUS ATTRIBUTES:

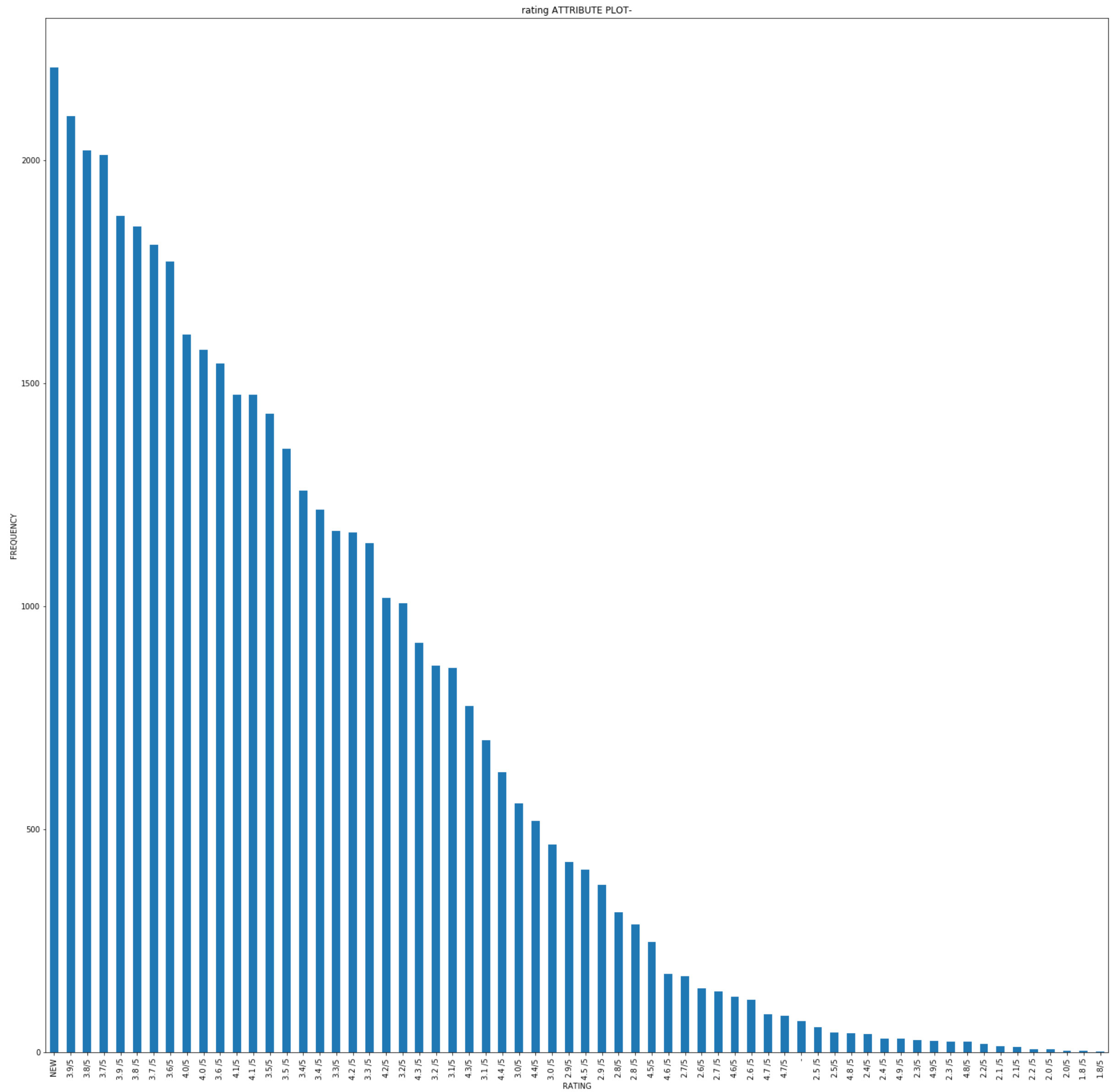
Restaurant Rating Chart-

The chart plots the frequency distribution of restaurant rating vs its occurence count(frequency).

X-axis: Customer rating. Y-axis: number/count of each rating value.

In [135]:
```
df['rate'].value_counts().plot(kind='bar',figsize=(25,25))
plt.title('rating ATTRIBUTE PLOT-')
plt.xlabel('RATING')
plt.ylabel('FREQUENCY')
```

Out[135]: Text(0, 0.5, 'FREQUENCY')



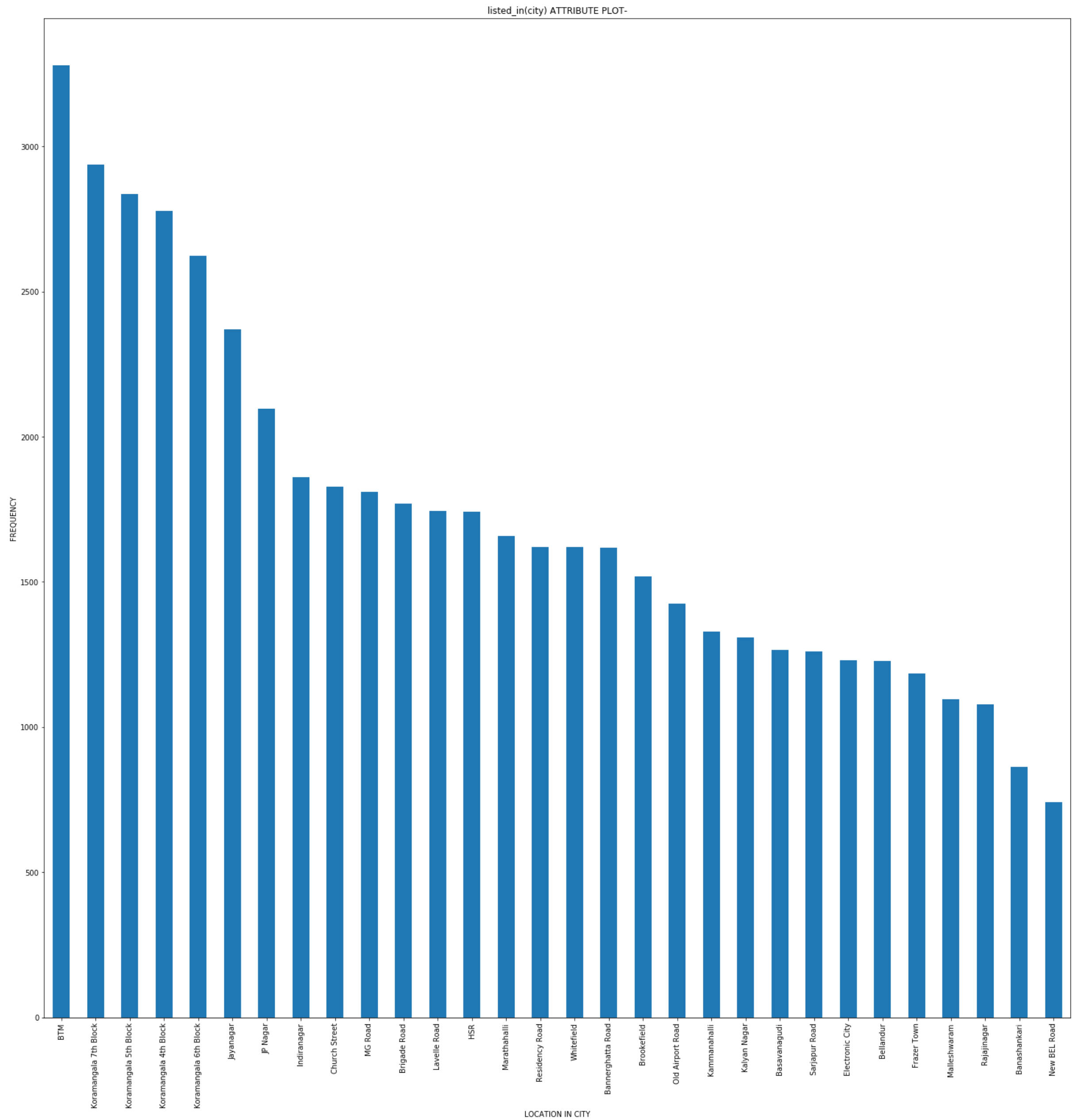
Restaurant Location Chart-

The chart plots the frequency distribution of restaurant location in the city vs its occurence count(frequency).

X-axis: where the restaurant is located in the city Y-axis: frequency

In [136]:
```
df['listed_in(city)'].value_counts().plot(kind='bar',figsize=(25,25))
plt.title('listed_in(city) ATTRIBUTE PLOT-')
plt.xlabel('LOCATION IN CITY')
plt.ylabel('FREQUENCY')
```

Out[136]:
```
Text(0, 0.5, 'FREQUENCY')
```


listed_in(city) ATTRIBUTE PLOT-
FREQUENCY
LOCATION IN CITY
0
500
1000
1500
2000
2500
3000
BTM
Koramangala 7th Block
Koramangala 5th Block
Koramangala 4th Block
Koramangala 6th Block
Jayanagar
JP Nagar
Indiranagar
Church Street
MG Road
Brigade Road
Lavelle Road
HSR
Marathahalli
Residency Road
Whitefield
Bannerghatta Road
Brookefield
Old Airport Road
Kammanahalli
Kalyan Nagar
Basavanagudi
Sarjapur Road
Electronic City
Bellandur
Frazer Town
Malleshwaram
Rajajinagar
Banashankari
New BEL Road

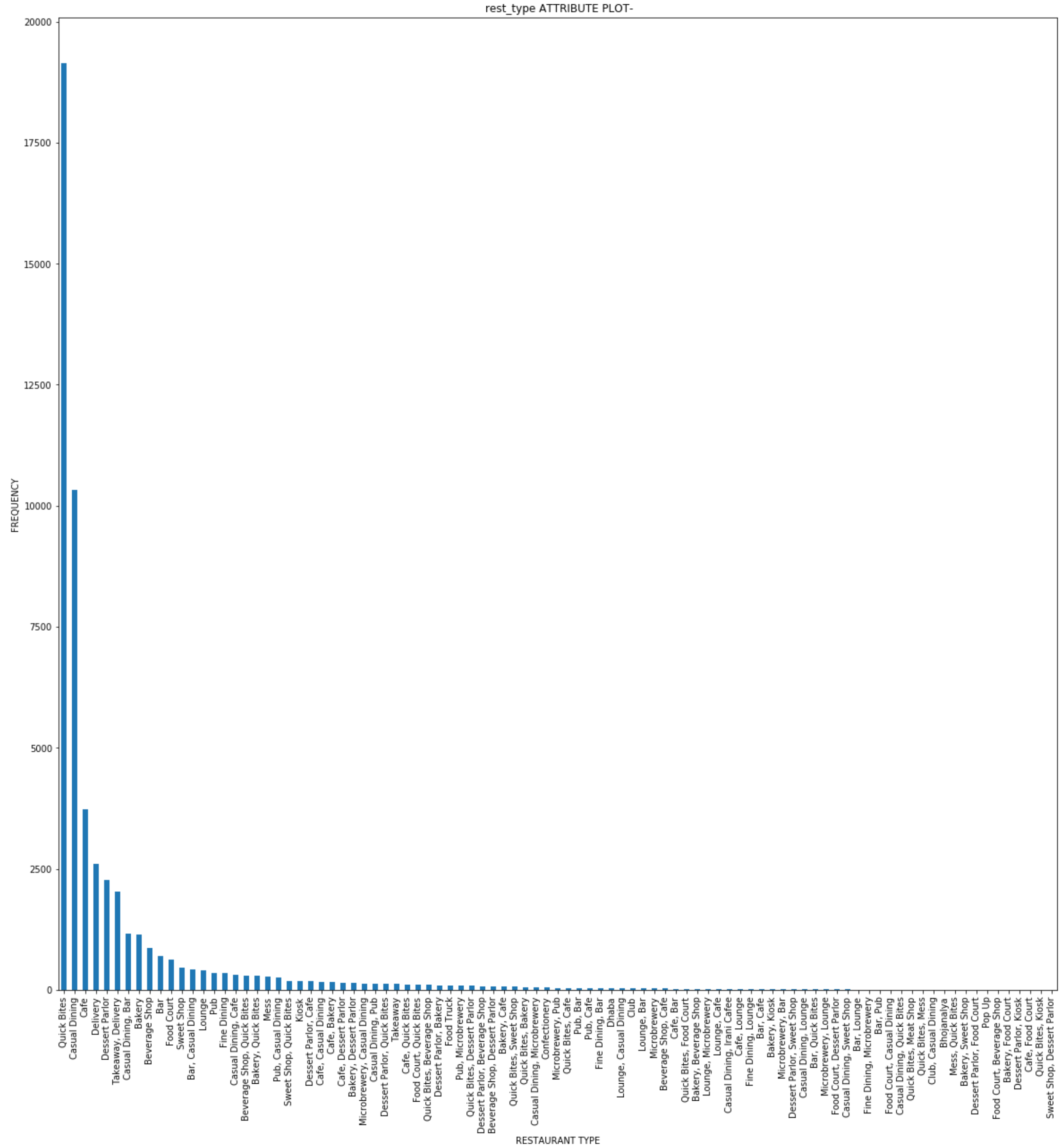
Restaurant Type/Cuisine Chart-

The chart plots the frequency distribution of type of restaurant vs its occurence count(frequency).

X-axis: restaurant type Y-axis: frequency

In [137]:
```
df['rest_type'].value_counts().plot(kind='bar',figsize=(20,20))
plt.title(' rest_type ATTRIBUTE PLOT-')
plt.xlabel('RESTAURANT TYPE')
plt.ylabel('FREQUENCY')
```

Out[137]:
```
Text(0, 0.5, 'FREQUENCY')
```


rest_type ATTRIBUTE PLOT-
FREQUENCY
20000
17500
15000
12500
10000
7500
5000
2500
0
Quick Bites
Casual Dining
Cafe
Delivery
Dessert Parlor
Takeaway, Delivery
Casual Dining, Bar
Bakery
Beverage Shop
Bar
Food Court
Sweet Shop
Bar, Casual Dining
Lounge
Pub
Fine Dining
Casual Dining, Cafe
Beverage Shop, Quick Bites
Bakery, Quick Bites
Mess
Pub, Casual Dining
Sweet Shop, Quick Bites
Kiosk
Dessert Parlor, Cafe
Cafe, Casual Dining
Cafe, Bakery
Cafe, Dessert Parlor
Bakery, Dessert Parlor
Microbrewery, Casual Dining
Casual Dining, Pub
Dessert Parlor, Quick Bites
Takeaway
Cafe, Quick Bites
Food Court, Quick Bites
Quick Bites, Beverage Shop
Dessert Parlor, Bakery
Food Truck
Pub, Microbrewery
Quick Bites, Dessert Parlor
Dessert Parlor, Beverage Shop
Beverage Shop, Dessert Parlor
Bakery, Cafe
Quick Bites, Sweet Shop
Quick Bites, Bakery
Casual Dining, Microbrewery
Confectionery
Microbrewery, Pub
Quick Bites, Cafe
Pub, Bar
Pub, Cafe
Fine Dining, Bar
Dhaba
Lounge, Casual Dining
Club
Lounge, Bar
Microbrewery
Beverage Shop, Cafe
Cafe, Bar
Quick Bites, Food Court
Bakery, Beverage Shop
Lounge, Microbrewery
Lounge, Cafe
Casual Dining, Irani Cafee
Cafe, Lounge
Fine Dining, Lounge
Bar, Cafe
Bakery, Kiosk
Microbrewery, Bar
Dessert Parlor, Sweet Shop
Casual Dining, Lounge
Bar, Quick Bites
Microbrewery, Lounge
Food Court, Dessert Parlor
Casual Dining, Sweet Shop
Bar, Lounge
Fine Dining, Microbrewery
Bar, Pub
Food Court, Casual Dining
Casual Dining, Quick Bites
Quick Bites, Meat Shop
Quick Bites, Mess
Club, Casual Dining
Bhojanalya
Mess, Quick Bites
Bakery, Sweet Shop
Dessert Parlor, Food Court
Pop Up
Food Court, Beverage Shop
Bakery, Food Court
Dessert Parlor, Kiosk
Cafe, Food Court
Quick Bites, Kiosk
Sweet Shop, Dessert Parlor
RESTAURANT TYPE

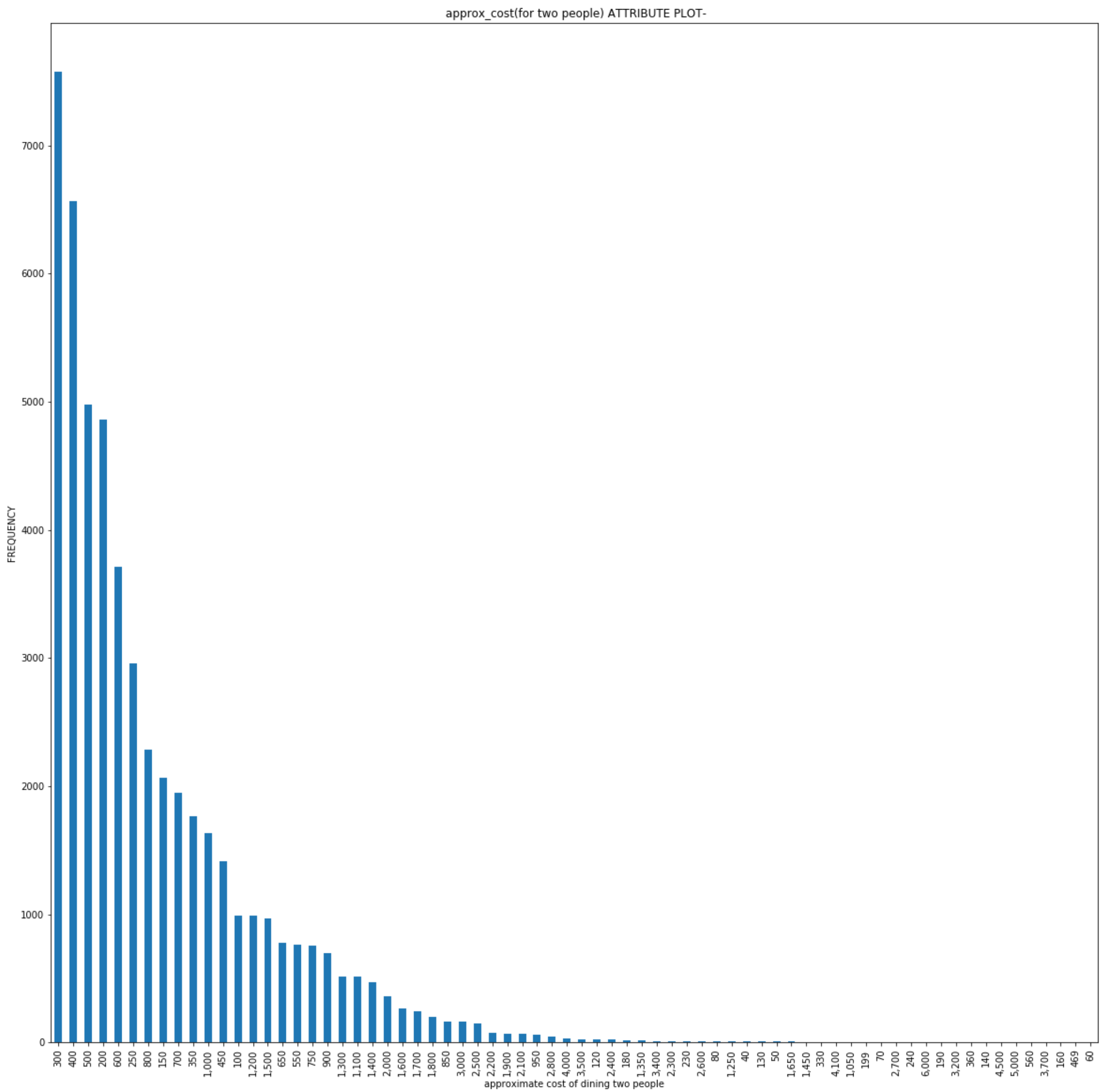
Restaurant Pricing Chart-

The chart plots the frequency distribution of cost of dining (two people) in a restaurant vs its occurence count(frequency).

X-axis: cost of dining for two persons Y-axis: frequency

In [138]:
```
df['approx_cost(for two people)'].value_counts().plot(kind='bar',figsiz
e=(20,20))
plt.title(' approx_cost(for two people) ATTRIBUTE PLOT-')
plt.xlabel('approximate cost of dining two people')
plt.ylabel('FREQUENCY')
```

Out[138]:
```
Text(0, 0.5, 'FREQUENCY')
```



Restaurant Meal Serving type Chart-

The chart plots the frequency distribution of how a meal will be served in a restaurant vs its occurence count(frequency).

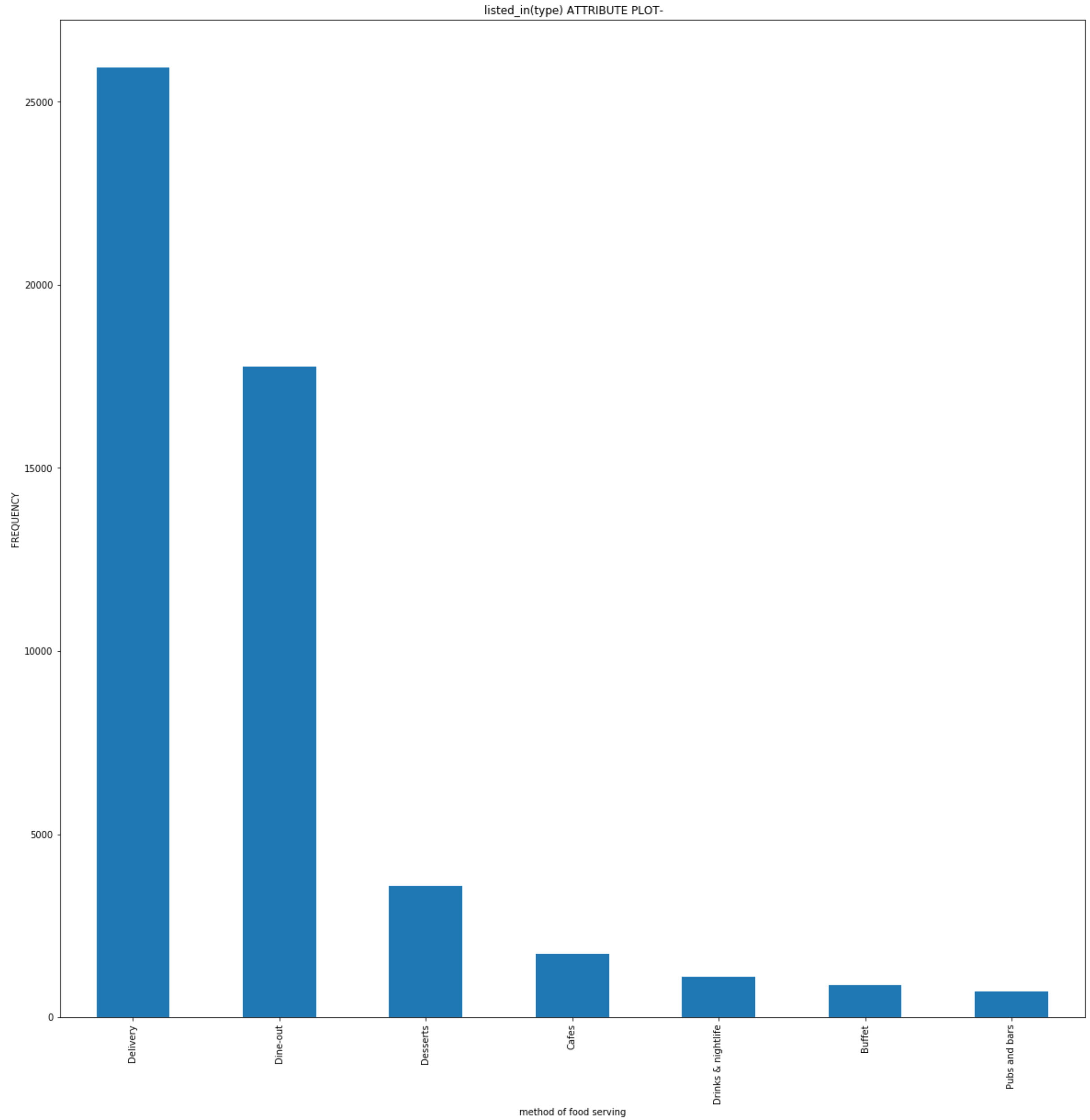
X-axis: listed Y-axis: number/count of each rating value.

In [139]:

```
df['listed_in(type)'].value_counts().plot(kind='bar',figsize=(20,20))
plt.title(' listed_in(type) ATTRIBUTE PLOT-')
plt.xlabel('method of food serving')
plt.ylabel('FREQUENCY')
```

Out[139]:

```
Text(0, 0.5, 'FREQUENCY')
```


listed_in(type) ATTRIBUTE PLOT-
FREQUENCY
25000
20000
15000
10000
5000
0
Delivery
Dine-out
Desserts
Cafes
Drinks & nightlife
Buffet
Pubs and bars
method of food serving

In [ ]:

2.c DROPPED ATTRIBUTES AND REASON FOR DROPPING-

URL attribute is dropped because in case of certain restaurants that are of same name but located in different locations in the city. They might conflict the way the whole data set is analyzed. Moreover, it increases the data’s structural complexity and due to their unnormalized nature.

Phone has been removed because with respect to the target variable of finding the ideal place to dine for two people, it won’t serve much importance and will remain irrelevant.

Location attribute is removed because of its logical conflict with the Listed in(city) attribute, both of which convey the same meaning.

Review_list attribute is dropped because most of the time people prefer to rate a given restaurant but rarely write reviews about them. This makes this attribute contain allot of blank sections which will ultimately affect the analysis’s accuracy. (However, on thing that can be done is that tagging the reviews based on their sentiment and then using that parameter in designing the models. But this has not been implemented as its out of scope of the task considering a holistic viewpoint of the attributes that will affect the target variable.

Dish_liked has not been considered as this factor can vary from person to person. Thus, it might be the case that a particular dish is liked by a set of people and then that same dish could be disliked by the other set of people. This makes the attribute vague, considering the broader perspective of the target variable.

Menu_item has been removed because of the redundancy that occurs when any two same restaurants with different locations will eventually have same menu item list. This will not be very useful with respect to the target variable being addressed.

2.d

In [ ]:

In [ ]:

In [140]:
```
#insert justification of dropping location in next step
#because "location" is part of address already present in listed_in(cit
y)
df[['location','listed_in(city)']]
```

Out[140]:

	location	listed_in(city)
0	Banashankari	Banashankari
1	Banashankari	Banashankari
2	Banashankari	Banashankari
3	Banashankari	Banashankari
4	Basavanagudi	Banashankari
5	Basavanagudi	Banashankari
6	Mysore Road	Banashankari
7	Banashankari	Banashankari
8	Banashankari	Banashankari
9	Banashankari	Banashankari
10	Banashankari	Banashankari
11	Banashankari	Banashankari
12	Banashankari	Banashankari
13	Banashankari	Banashankari
14	Banashankari	Banashankari
15	Banashankari	Banashankari
16	Banashankari	Banashankari
17	Banashankari	Banashankari
18	Banashankari	Banashankari

	location	listed_in(city)
19	Banashankari	Banashankari
20	Banashankari	Banashankari
21	Banashankari	Banashankari
22	Banashankari	Banashankari
23	Banashankari	Banashankari
24	Banashankari	Banashankari
25	Banashankari	Banashankari
26	Banashankari	Banashankari
27	Banashankari	Banashankari
28	Banashankari	Banashankari
29	Basavanagudi	Banashankari
...	...	...
51687	Whitefield	Whitefield
51688	Whitefield	Whitefield
51689	Whitefield	Whitefield
51690	Whitefield	Whitefield
51691	Whitefield	Whitefield
51692	Whitefield	Whitefield
51693	Whitefield	Whitefield
51694	Whitefield	Whitefield
51695	Whitefield	Whitefield
51696	Whitefield	Whitefield
51697	Whitefield	Whitefield
51698	Whitefield	Whitefield
51699	Whitefield	Whitefield

	location	listed_in(city)
51699	Whitefield	Whitefield

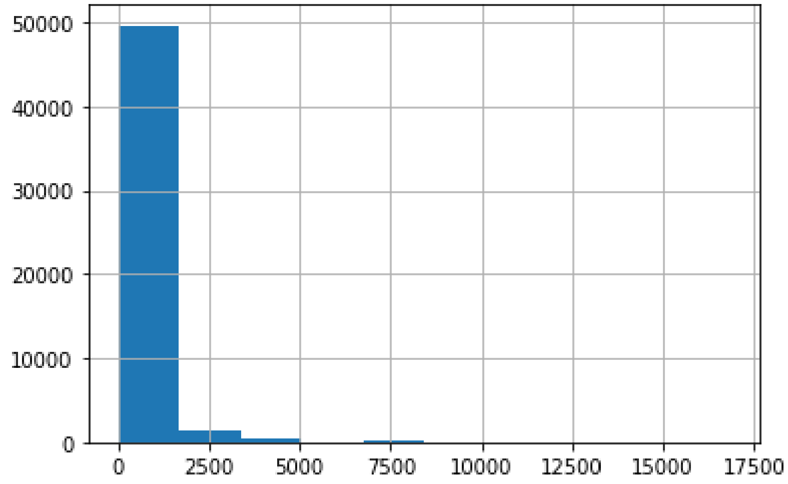
	location	listed_in(city)
51700	Whitefield	Whitefield
51701	Whitefield	Whitefield
51702	Whitefield	Whitefield
51703	Whitefield	Whitefield
51704	Whitefield	Whitefield
51705	Whitefield	Whitefield
51706	Whitefield	Whitefield
51707	Whitefield	Whitefield
51708	Whitefield	Whitefield
51709	Whitefield	Whitefield
51710	Whitefield	Whitefield
51711	Whitefield	Whitefield
51712	Whitefield	Whitefield
51713	Whitefield	Whitefield
51714	Whitefield	Whitefield
51715	ITPL Main Road, Whitefield	Whitefield
51716	ITPL Main Road, Whitefield	Whitefield

51717 rows × 2 columns

In [141]:
```
df.drop(columns=['url','phone','location'], inplace=True)
```

In [142]:
```
df['votes'].hist()
```

Out[142]: <matplotlib.axes._subplots.AxesSubplot at 0x21ff4867860>



In [143]:
```
df.rename(columns={'approx_cost(for two people)': 'average_cost', 'list
ed_in(city)':'locality', 'listed_in(type)':'meal_type'},inplace=True)
df.head()
```

Out[143]:

	address	name	online_order	book_table	rate	votes	rest_type	dish_liked	cuisine
0	942, 21st Main Road, 2nd Stage, Banashankari, ...	Jalsa	Yes	Yes	4.1/5	775	Casual Dining	Pasta, Lunch Buffet, Masala Papad, Paneer Laja...	Nor India Mughla Chines
1	2nd Floor, 80 Feet Road, Near Big Bazaar, 6th ...	Spice Elephant	Yes	No	4.1/5	787	Casual Dining	Momos, Lunch Buffet, Chocolate Nirvana, Thai G...	Chines Nor India Th
2	1112, Next to KIMS Medical College, 17th Cross...	San Churro Cafe	Yes	No	3.8/5	918	Cafe, Casual Dining	Churros, Cannelloni, Minestrone Soup, Hot Choc...	Caf Mexica Italia

	address	name	online_order	book_table	rate	votes	rest_type	dish_liked	cuisine
3	1st Floor, Annakuteera, 3rd Stage, Banashankar...	Addhuri Udupi Bhojana	No	No	3.7/5	88	Quick Bites	Masala Dosa	Sou India Nor India
4	10, 3rd Floor, Lakshmi Associates, Gandhi Baza...	Grand Village	No	No	3.8/5	166	Casual Dining	Panipuri, Gol Gappe	Nor India Rajastha

In [144]:
```
df['average_cost'].unique()
```

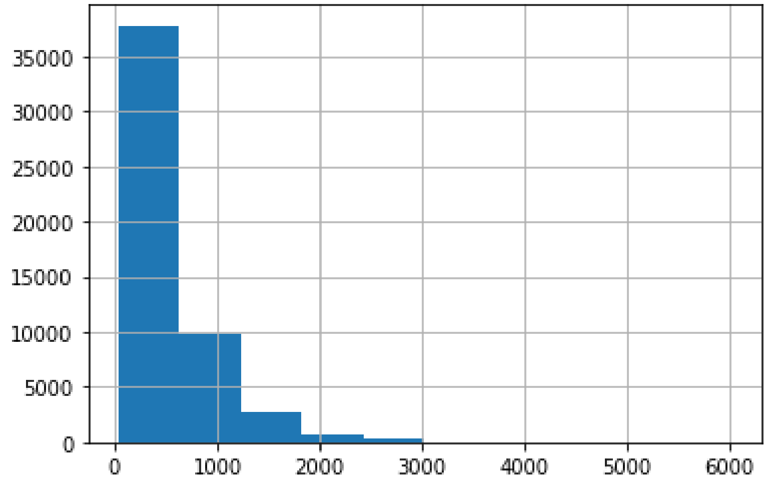
Out[144]:
```
array(['800', '300', '600', '700', '550', '500', '450', '650', '400',
       '900', '200', '750', '150', '850', '100', '1,200', '350', '250',
       '950', '1,000', '1,500', '1,300', '199', '80', '1,100', '160',
       '1,600', '230', '130', '50', '190', '1,700', nan, '1,400', '18
0',
       '1,350', '2,200', '2,000', '1,800', '1,900', '330', '2,500',
       '2,100', '3,000', '2,800', '3,400', '40', '1,250', '3,500',
       '4,000', '2,400', '2,600', '120', '1,450', '469', '70', '3,200',
       '60', '560', '240', '360', '6,000', '1,050', '2,300', '4,100',
       '5,000', '3,700', '1,650', '2,700', '4,500', '140'], dtype=objec
t)
```

In [145]:
```
df['average_cost']=df['average_cost'].str.replace(',','')
```

In [146]:
```
df['average_cost']=df['average_cost'].astype(float)
```

In [147]:
```
df['average_cost'].hist()
```

Out[147]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x21ff4958da0>
```



In [148]:
```
df['rate'].unique()
```

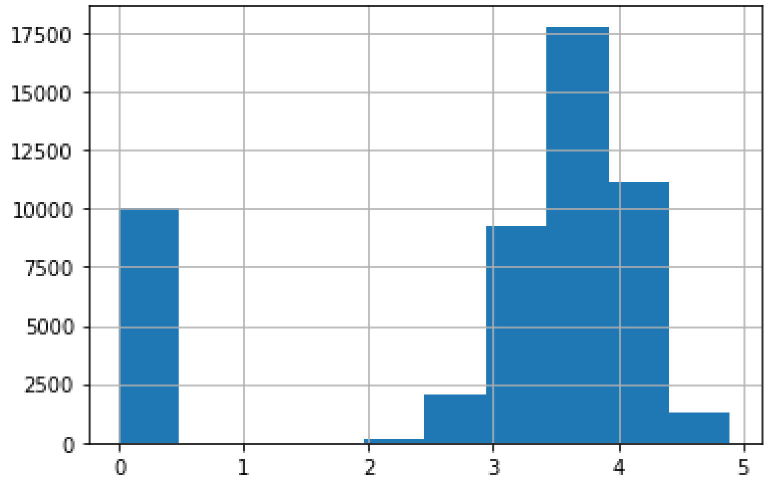
Out[148]:
```
array(['4.1/5', '3.8/5', '3.7/5', '3.6/5', '4.6/5', '4.0/5', '4.2/5',
       '3.9/5', '3.1/5', '3.0/5', '3.2/5', '3.3/5', '2.8/5', '4.4/5',
       '4.3/5', 'NEW', '2.9/5', '3.5/5', nan, '2.6/5', '3.8 /5', '3.4/
5',
       '4.5/5', '2.5/5', '2.7/5', '4.7/5', '2.4/5', '2.2/5', '2.3/5',
       '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)
```

In [149]:
```
df['rate']=df['rate'].str.strip()
df['rate']=df['rate'].str.replace('/5','')
df['rate'].fillna(0,inplace=True)
df['rate'].replace("NEW", 0,inplace=True)
df['rate'].replace("-", 0,inplace=True)
```

In [150]:
```
df.drop(columns=['reviews_list','dish_liked','menu_item'], inplace=True
)
```

In [151]:
```
df['rate']=df['rate'].astype(float)
df['rate'].hist()
```

Out[151]: <matplotlib.axes._subplots.AxesSubplot at 0x21ff7ace908>



In [152]:
```
#Name and Address uniquely identify any restaurant in the data, so remo
ving duplicates
ex=df.drop_duplicates(['name','address'])
```

In [153]:
```
ex.isna().sum()
```

Out[153]:
```
address          0
name             0
online_order     0
book_table       0
rate             0
votes            0
rest_type       63
cuisines        19
average_cost    59
meal_type        0
```

```
locality          0
dtype: int64
```

In [154]:
```python
ex.dropna(inplace=True)
```

```
c:\users\aakash patel\appdata\local\programs\python\python37\lib\site-p
ackages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [155]:
```python
ex.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 12376 entries, 0 to 51714
Data columns (total 11 columns):
address          12376 non-null object
name             12376 non-null object
online_order     12376 non-null object
book_table       12376 non-null object
rate             12376 non-null float64
votes            12376 non-null int64
rest_type        12376 non-null object
cuisines         12376 non-null object
average_cost     12376 non-null float64
meal_type        12376 non-null object
locality         12376 non-null object
dtypes: float64(2), int64(1), object(8)
memory usage: 1.1+ MB
```

In [156]:
```python
ex=ex.reset_index(drop=True)
```

In [157]:
```python
ex.groupby(['locality']).mean().sort_values(['rate'],ascending=False)
```

Out[157]:

	rate	votes	average_cost

locality	rate	votes	average_cost
locality			
MG Road	3.453846	145.615385	557.692308
Koramangala 5th Block	3.305263	86.736842	386.842105
Malleshwaram	3.088984	199.178827	508.469242
Banashankari	3.050000	170.361204	389.530100
Brigade Road	3.049713	292.830784	697.915870
Residency Road	3.022222	18.777778	522.222222
Lavelle Road	2.927344	103.078125	669.921875
Indiranagar	2.864011	335.630810	550.584329
Basavanagudi	2.838619	166.560102	422.506394
BTM	2.834671	259.963322	463.687889
Frazer Town	2.761538	148.406593	430.219780
Church Street	2.748039	160.000000	517.156863
Bellandur	2.734777	200.198726	500.458599
Koramangala 6th Block	2.716000	132.440000	478.000000
Brookefield	2.688548	160.781186	495.593047
Jayanagar	2.685816	149.070922	432.269504
Bannerghatta Road	2.627877	130.864359	426.903648
Kalyan Nagar	2.626323	109.655989	450.348189
Koramangala 7th Block	2.582143	131.178571	398.214286
Sarjapur Road	2.488889	46.090909	430.303030
Rajajinagar	2.484170	71.559846	405.135135
Kammanahalli	2.475000	57.125000	341.250000
Whitefield	2.473673	151.923567	553.651805

locality	rate	votes	average_cost
Koramangala 4th Block	2.463768	207.351449	486.739130
New BEL Road	2.438820	95.832298	405.900621
Marathahalli	2.376271	167.717514	469.915254
Old Airport Road	2.354545	70.075758	477.121212
HSR	2.345671	132.764069	424.805195
JP Nagar	2.200000	92.731707	414.024390
Electronic City	2.185241	77.151724	461.820690

2.d

To be able to find out neighbourhood with maximum rating, we grouped by locality attribute and sorted it in descending order.

In [158]:
```
q2_d=ex[ex['locality']=='MG Road']
```

We discovered that neighbourhood with maximum rating is **MG Road**

In [159]:
```
len(q2_d)
```

Out[159]: 13

We found only 13 such records

In [160]:
```
q2_d.groupby(['online_order']).count()
```

Out[160]:

	address	name	book_table	rate	votes	rest_type	cuisines	average_cost	meal_ty

	address	name	book_table	rate	votes	rest_type	cuisines	average_cost	meal_ty
online_order									
No	6	6	6	6	6	6	6	6	
Yes	7	7	7	7	7	7	7	7	

There are 7 restaurants which are "Online Order" enabled. All Others do not have online order facility.

In [161]:

```
q2_d.groupby(['rest_type']).count()
```

Out[161]:

	address	name	online_order	book_table	rate	votes	cuisines	average_cost	meal_ty
rest_type									
Cafe	4	4	4	4	4	4	4	4	
Cafe, Casual Dining	1	1	1	1	1	1	1	1	
Casual Dining	3	3	3	3	3	3	3	3	
Delivery	2	2	2	2	2	2	2	2	
Quick Bites	3	3	3	3	3	3	3	3	

Grouping with restaurant type shows number of restaurants according to restaurant type.

In [162]:

```
q2_d.groupby(['book_table']).count()
```

Out[162]:

address	name	online_order	rate	votes	rest_type	cuisines	average_cost	meal_ty

book_table	address	name	online_order	rate	votes	rest_type	cuisines	average_cost	meal_ty
book_table									
No	13	13	13	13	13	13	13	13	

Grouping by Book table shows that all the restaurants don't have booking table facility.

In [163]:
```
q2_d.groupby(['meal_type']).count()
```

Out[163]:

	address	name	online_order	book_table	rate	votes	rest_type	cuisines	average_c
meal_type									
Cafes	3	3	3	3	3	3	3	3	
Delivery	4	4	4	4	4	4	4	4	
Dine-out	6	6	6	6	6	6	6	6	

Grouping by meal type shows restaurant count for different meal types.

3.a

- The inhand task we are solving is a supervised one, as we have labelled data in the data set.Moreover, since the target varaible is to find the average cost of dining of two people in a restaurant, so the model we intend to create can use various attributes like rating, location within the city, restraunt type etc to appropriately predict the target variable.
- Secondly, since we have to qunatify the approximate cost of meal of two people. Therefore the in hand task will be a regression model which can predict the numerical value of the involved variables.
- In terms of the similarity matching, in the given context since the target variable is to predict the ideal restraunt in terms of cost, so finding similar groups who can go to a particluar

restraunt can be insighfull but it wont serve the prupose of addressing the target variable. hence similarity matching will not be used in the given context.

In [164]:
```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
one_hot=preprocessing.OneHotEncoder()
from sklearn.model_selection import train_test_split
```

In [165]:
```
Y=ex['average_cost']
ex['locality']= label_encoder.fit_transform(ex['locality'])
from sklearn.preprocessing import MultiLabelBinarizer
mlb = MultiLabelBinarizer()
ex['cuisines']=ex.cuisines.str.split(", ")
```

In [166]:
```
#ex['cuisines']=mlb.fit_transform(ex['cuisines'])
ex['rest_type']=ex.rest_type.str.split(", ")
```

In [167]:
```
ex1 = pd.DataFrame(mlb.fit_transform(ex['cuisines']),columns=mlb.classe
s_)
ex2 = pd.DataFrame(mlb.fit_transform(ex['rest_type']),columns=mlb.class
es_)
```

In [168]:
```
ex['cuisines']
```

Out[168]:
```
0                       [North Indian, Mughlai, Chinese]
1                          [Chinese, North Indian, Thai]
2                               [Cafe, Mexican, Italian]
3                           [South Indian, North Indian]
4                             [North Indian, Rajasthani]
5                                         [North Indian]
6          [North Indian, South Indian, Andhra, Chinese]
7                                 [Pizza, Cafe, Italian]
8                           [Cafe, Italian, Continental]
9             [Cafe, Mexican, Italian, Momos, Beverages]
10                                                [Cafe]
11                          [Cafe, Italian, Continental]
```

```
12                [Cafe, Chinese, Continental, Italian]
13                                  [Cafe, Continental]
14                                               [Cafe]
15         [Cafe, Fast Food, Continental, Chinese, Momos]
16                             [Chinese, Cafe, Italian]
17                            [Cafe, Italian, American]
18                [Cafe, Chinese, Continental, Italian]
19                         [Cafe, French, North Indian]
20                  [Cafe, Pizza, Fast Food, Beverages]
21                                    [Cafe, Fast Food]
22                [Italian, Fast Food, Cafe, European]
23                                               [Cafe]
24                                       [Cafe, Bakery]
25                                 [Cafe, South Indian]
26                         [Cafe, Fast Food, Beverages]
27                                    [Cafe, Fast Food]
28          [North Indian, Cafe, Chinese, Fast Food]
29                                      [Cafe, Italian]
                              ...
12346          [North Indian, South Indian, Chinese]
12347                                      [Chinese]
12348                                 [North Indian]
12349                      [South Indian, Fast Food]
12350                        [Chinese, North Indian]
12351           [North Indian, Chinese, Continental]
12352                       [Street Food, Fast Food]
12353                                 [South Indian]
12354                                 [South Indian]
12355                        [South Indian, Chinese]
12356                                  [Street Food]
12357                        [North Indian, Chinese]
12358                             [Chinese, Mughlai]
12359                             [Biryani, Chinese]
12360               [Arabian, Chinese, North Indian]
12361                                      [Arabian]
12362                             [Fast Food, Rolls]
12363                        [North Indian, Chinese]
12364                                 [North Indian]
12365                                      [Italian]
```

```
12366                                  [Finger Food]
12367          [Finger Food, North Indian, Continental]
12368                                [Chinese, Momos]
12369          [North Indian, Chinese, Arabian, Momos]
12370                          [Thai, Chinese, Momos]
12371                                  [North Indian]
12372               [North Indian, Continental, Asian]
12373                  [North Indian, Kerala, Chinese]
12374     [Andhra, South Indian, Chinese, North Indian]
12375                                  [Finger Food]
Name: cuisines, Length: 12376, dtype: object
```

In [169]:
```
X=pd.concat([ex[['rate','locality']], ex1,ex2], axis=1)
```

In [170]:
```
X = X.loc[:,~X.columns.duplicated()]
```

In [171]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
, random_state=42)
```

3.b

To implement the task of estimating the cost, we have strategised four models. These models have been implemented as an individually and then the accuracies of each model have been determined and compared. this ultimately helped us pinpoint the best model that fits the data perfectly. Models have been created for each approach as following:

3.c MODEL EVALUATION METRICS-

- For metricizing the accuracy of various models we have choosen coefficient of determination(r2 score) metrics for model evaluation.Now here ,minimum score can be of 1. but here when the model does not take into consideration any input parameter, the r2 score is 0. With each respective model, the values have been correspondingly shown in the output cell reletive to it.

- Second metric that we have used is the root mean square(RMS) value which gives the under root of the variance of residuals. It describe the absolute fit of model w.r.t data and the nearness of the seen data points to the predicted model.With each respective model, the values have been correspondingly shown in the output cell reletive to it. Here, we have not quantified the rms error to be in its ideal range. Hence, we we have taken just a general estimate of how this metric will be able to judje the model's accuracy characteristics.

DECISION TREE REGRESSOR-

- The decision tree regressor basically fits a mathematical sine curve along with additional outliers observation. Hence, the model learns the local linear regression from approximation provided by the sine curve.
- Secondly, we can obserbe that for the max depth of the tree(max_depth parameter) is kept very high, the tree overfits(because it learns very fine details of the training data and also from the noise).

In [172]:

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from math import sqrt

treemodel = DecisionTreeRegressor(random_state = 0,max_depth=10,min_imp
urity_decrease=70)
treemodel=treemodel.fit(X_train, y_train)
y_pred = treemodel.predict(X_test)
print("Decision Tree Regressor-")
print("r_2 testing and training metric:")
print("Testing accuracy is=")
print(r2_score(y_test, y_pred))
print("Training accuracy is=")
print(r2_score(y_train, treemodel.predict(X_train)))

print("RMS testing and training metric:")
```

```
print("Testing accuracy is=")
print( sqrt(mean_squared_error(y_test, y_pred)))
print("Training accuracy is=")
print(sqrt(mean_squared_error(y_train, treemodel.predict(X_train))))
```

```
Decision Tree Regressor-
r_2 testing and training metric:
Testing accuracy is=
0.7197053066027574
Training accuracy is=
0.8051509196619485
RMS testing and training metric:
Testing accuracy is=
207.40537393137066
Training accuracy is=
172.84344839194597
```

LINEAR REGRESSION MODEL-

Since the linear regression helps finding relation in two variable.So, The second model we used was linear regression where we used the method of least squares. Here the best fitting line for the given data was calculated by taking the minimum of sum of squares of vertical changes form every point of data to the line splitting the data.This was the main reson why we choose this model to predict the target variable.

In [173]:
```
from sklearn.linear_model import LinearRegression
linmodel=LinearRegression()
linmodel.fit(X_train,y_train)
y_reg_predict=linmodel.predict(X_test)
print("Linear Regression-")
print("r_2 testing and training metric:")
print("Testing accuracy is=")
print(r2_score(y_test, y_reg_predict))
print("Training accuracy is=")
print(r2_score(y_train, linmodel.predict(X_train)))

print("RMS testing and training metric:")
```

```
print("Testing accuracy is=")
print( sqrt(mean_squared_error(y_test, y_pred)))
print("Training accuracy is=")
print(sqrt(mean_squared_error(y_train, linmodel.predict(X_train))))
```

```
Linear Regression-
r_2 testing and training metric:
Testing accuracy is=
0.7273404250136796
Training accuracy is=
0.7357617027193442
RMS testing and training metric:
Testing accuracy is=
207.40537393137066
Training accuracy is=
201.28046321725284
```

RANDOM FOREST REGRESSOR-

In this, we simply made use of bagging to train each decision tree of the data sub-sample and correspondingly the replacemnts were made with every iteration. So we fit a number of various decision trees and made use of averaging to enhance predictive accuracy and reduce any overfitting. This was the reason randon forest regressor was choosen as one of the model for the purpose.

In [174]:
```
from sklearn.ensemble import RandomForestRegressor
ranmodel = RandomForestRegressor(n_estimators=40, random_state=0)
ranmodel.fit(X_train, y_train)
y_pred = ranmodel.predict(X_test)
print("Random Forest Regressor-")
print("r_2 testing and training metric:")
print("Testing accuracy is=")
print(r2_score(y_test, y_pred))
print("Training accuracy is=")
print(r2_score(y_train, ranmodel.predict(X_train)))

print("RMS testing and training metric:")
```

```
print("Testing accuracy is=")
print( sqrt(mean_squared_error(y_test, y_pred)))
print("Training accuracy is=")
print(sqrt(mean_squared_error(y_train, ranmodel.predict(X_train))))
```

```
Random Forest Regressor-
r_2 testing and training metric:
Testing accuracy is=
0.7590181971761176
Training accuracy is=
0.9555772454909043
RMS testing and training metric:
Testing accuracy is=
192.31125037861779
Training accuracy is=
82.5289812726249
```

XG BOOST-

we implemented this machine learning technique under gradient boosting framework. Since it provides with parallel tree boosting, it is accurate and fast at the same time.Moreover, its an optimisd version of gradient boosting library which makes it highly flexible,portable and efficient. Thus, we have decided to stick on with the xg-boost model , since in terms of accuracy and efficiency to work eg-boost shows the best results.

In [175]:
```
import xgboost
xgbmodel =  xgboost.XGBRegressor(colsample_bytree=0.3,
                 gamma=0,
                 learning_rate=0.08,
                 max_depth=12,
                 min_child_weight=1.5,
                 n_estimators=100,

                 reg_alpha=10,
                 reg_lambda=0.45,
                 subsample=0.6,
                 seed=42)
```

```python
xgbmodel.fit(X_train,y_train)
y_pred = xgbmodel.predict(X_test)
print("XGBoost Regressor-")
print("r_2 testing and training metric:")
print("Testing accuracy is=")
print(r2_score(y_test, y_pred))
print("Training accuracy is=")
print(r2_score(y_train, xgbmodel.predict(X_train)))

print("RMS testing and training metric:")
print("Testing accuracy is=")
print( sqrt(mean_squared_error(y_test, y_pred)))
print("Training accuracy is=")
print(sqrt(mean_squared_error(y_train, xgbmodel.predict(X_train))))
```

```
[23:39:25] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/sr
c/objective/regression_obj.cu:152: reg:linear is now deprecated in favo
r of reg:squarederror.
```

```
c:\users\aakash patel\appdata\local\programs\python\python37\lib\site-p
ackages\xgboost\core.py:587: FutureWarning: Series.base is deprecated a
nd will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
XGBoost Regressor-
r_2 testing and training metric:
Testing accuracy is=
0.7804915380213014
Training accuracy is=
0.8592333218114218
RMS testing and training metric:
Testing accuracy is=
183.54315827759189
Training accuracy is=
146.91078294915712
```

3.d. AVOIDING OVERFITTING:

DECISION TREE REGRESSOR: overfitting can be prevented pre pruning(and post pruning. Pre-pruning implies to stop growing the tree earlier, before it perfectly classifies the training set. Post-pruning works by allowing the tree to perfectly classify the training set, and then post prune the tree.

LINEAR REGRESSION: Here the overfitting can be prevented by the method of cross validation where a simple generalised estimate is made over a small data set(holdout data) and then for all the other data sets the results are evaluated and by multiple splits made across the data). the exact implementation has been done in 3.g part.

RANDOM FOREST REGRESSOR- In random forest tuming parameters could be done to avoid overfitting. In our case, making use of parameters that are not compatable with the null values must be selected to prevent the model to overfit.

XG-BOOST: Early Stopping: It prevents overfitting by trying to automatically pick the inflection point where performance on the test dataset begins to reduce whereas performance on the training dataset keep on improving as the model begin to overfit. Now cross validation can also be done to avoid overfitting in xg-boost. It has been implemented as follows.

3.e. CROSS VALIDATION

In [176]:

```
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV
results = model_selection.cross_val_score(xgbmodel, X, Y, cv=3)
print("Performing Cross Validation on XGBoost Model:")
print("Accuracy: Final mean:%.3f%%, Final standard deviation:(%.3f%%)"
% (results.mean()*100.0, results.std()*100.0))
print('Accuracies from each of the folds using kfold:',results)
print("Variance of kfold accuracies:",results.var())
objects = (['1', '2', '3'])
y_pos = np.arange(len(objects))
plt.bar(y_pos, results, align='center', alpha=1.0)
plt.xticks(y_pos, objects)
plt.ylabel('Fold Accuracy')
plt.xlabel('Folds')
```

```
plt.title('XG Boost Cross Validation ')
plt.show()
```

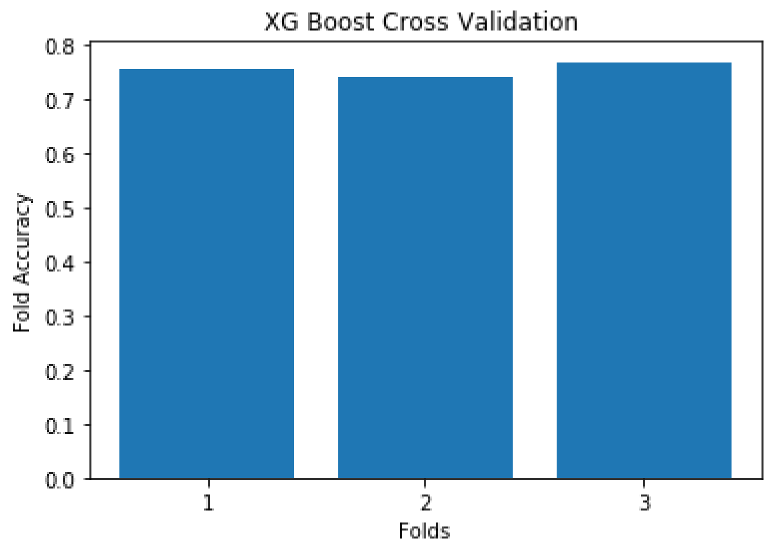
```
[23:39:34] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/sr
c/objective/regression_obj.cu:152: reg:linear is now deprecated in favo
r of reg:squarederror.
```

```
c:\users\aakash patel\appdata\local\programs\python\python37\lib\site-p
ackages\xgboost\core.py:587: FutureWarning: Series.base is deprecated a
nd will be removed in a future version
  if getattr(data, 'base', None) is not None and \
c:\users\aakash patel\appdata\local\programs\python\python37\lib\site-p
ackages\xgboost\core.py:587: FutureWarning: Series.base is deprecated a
nd will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
[23:39:41] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/sr
c/objective/regression_obj.cu:152: reg:linear is now deprecated in favo
r of reg:squarederror.
```

```
c:\users\aakash patel\appdata\local\programs\python\python37\lib\site-p
ackages\xgboost\core.py:587: FutureWarning: Series.base is deprecated a
nd will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
[23:39:48] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/sr
c/objective/regression_obj.cu:152: reg:linear is now deprecated in favo
r of reg:squarederror.
Performing Cross Validation on XGBoost Model:
Accuracy: Final mean:75.647%, Final standard deviation:(1.094%)
Accuracies from each of the folds using kfold: [0.75831316 0.74224542
0.76884321]
Variance of kfold accuracies: 0.00011961078109060164
```

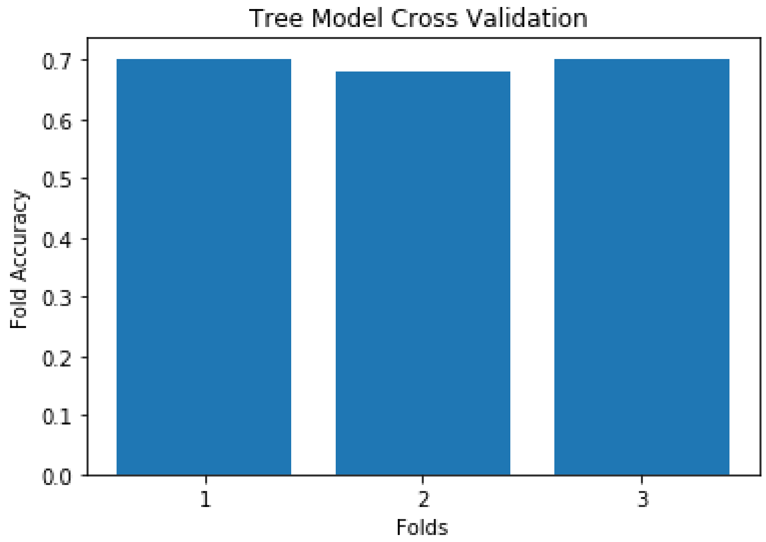


In [177]:

```
results = model_selection.cross_val_score(treemodel, X, Y, cv=3)
print("Performing Cross Validation on Tree Model:")
print("Accuracy: Final mean:%.3f%%, Final standard deviation:(%.3f%%)"
% (results.mean()*100.0, results.std()*100.0))
print('Accuracies from each of the folds using kfold:',results)
print("Variance of kfold accuracies:",results.var())

objects = (['1', '2', '3'])
y_pos = np.arange(len(objects))
plt.bar(y_pos, results, align='center', alpha=1.0)
plt.xticks(y_pos, objects)
plt.ylabel('Fold Accuracy')
plt.xlabel('Folds')
plt.title('Tree Model Cross Validation ')
```

```
plt.show()
```

```
Performing Cross Validation on Tree Model:
Accuracy: Final mean:69.502%, Final standard deviation:(0.972%)
Accuracies from each of the folds using kfold: [0.70159013 0.68128827
0.70219375]
Variance of kfold accuracies: 9.439660903321898e-05
```

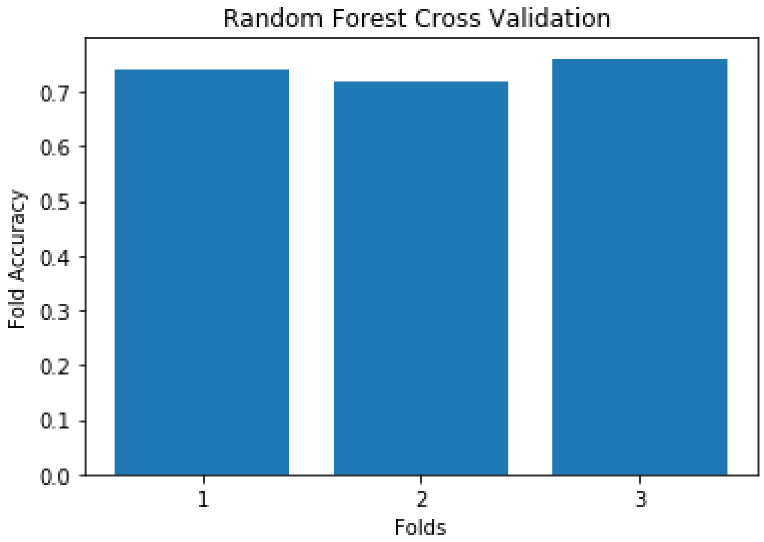


In [178]:

```
results = model_selection.cross_val_score(ranmodel, X, Y, cv=3)
print("Performing Cross Validation on Random Forest Model:")
print("Accuracy: Final mean:%.3f%%, Final standard deviation:(%.3f%%)"
% (results.mean()*100.0, results.std()*100.0))
print('Accuracies from each of the folds using kfold:',results)
print("Variance of kfold accuracies:",results.var())
objects = (['1', '2', '3'])
y_pos = np.arange(len(objects))
plt.bar(y_pos, results, align='center', alpha=1.0)
plt.xticks(y_pos, objects)
plt.ylabel('Fold Accuracy')
plt.xlabel('Folds')
```

```
plt.title('Random Forest Cross Validation ')
plt.show()
```

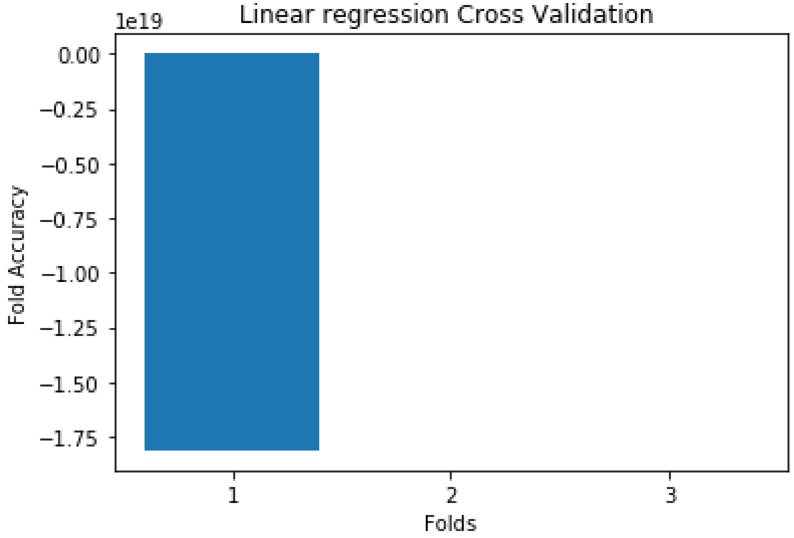
```
Performing Cross Validation on Random Forest Model:
Accuracy: Final mean:74.026%, Final standard deviation:(1.651%)
Accuracies from each of the folds using kfold: [0.74156313 0.71941552
0.75979123]
Variance of kfold accuracies: 0.0002725530608754980
```



In [179]:
```
results = model_selection.cross_val_score(linmodel, X, Y, cv=3)
print("Performing Cross Validation on Linear regression Model:")
print("Accuracy: Final mean:%.3f%%, Final standard deviation:(%.3f%%)"
% (results.mean()*100.0, results.std()*100.0))
print('Accuracies from each of the folds using kfold:',results)
print("Variance of kfold accuracies:",results.var())
objects = (['1','2', '3'])
y_pos = np.arange(len(objects))
plt.bar(y_pos, results, align='center', alpha=1.0)
plt.xticks(y_pos, objects)
plt.ylabel('Fold Accuracy')
plt.xlabel('Folds')
```

```
plt.title('Linear regression Cross Validation ')
plt.show()
```

```
Performing Cross Validation on Linear regression Model:
Accuracy: Final mean:-604111003090254954496.000%, Final standard deviat
ion:(854341973749053259776.000%)
Accuracies from each of the folds using kfold: [-1.81233301e+19  6.9027
8028e-01  7.25571902e-01]
Variance of kfold accuracies: 7.299002081094279e+37
```



Variance of linear model is significantly high. Rest all three are not too high. Since it provides with parallel tree boosting, it is accurate and fast at the same time.Moreover, its an optimisd version of gradient boosting library which makes it highly flexible,portable and efficient. Thus, we have decided to stick on with the xg-boost model , since in terms of accuracy and efficiency to work xgboost shows the best results.Thus best model is to choose is XGBoost.

3.f. Testing Training Performance comparison

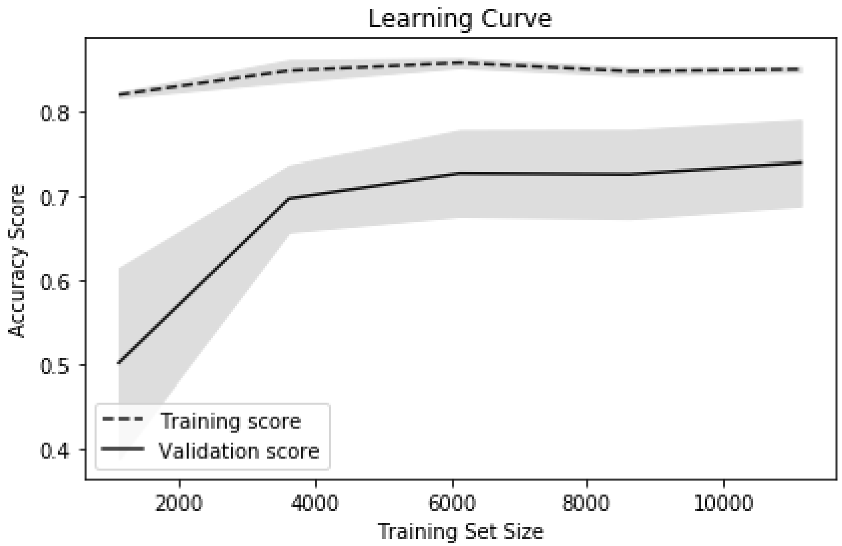
In [180]:
```
from sklearn.model_selection import learning_curve
```

```
xgbmodel.fit(X_train,y_train)
y_pred = xgbmodel.predict(X_test)
print("XGBoost Model-")
print("Testing accuracy is=")
print(r2_score(y_test, y_pred))
print("Training accuracy is=")
print(r2_score(y_train, xgbmodel.predict(X_train)))

train_sizes, train_scores, test_scores = learning_curve(
        xgbmodel, X, Y, cv=10, n_jobs=-1)

# Create means and standard deviations of training set scores
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)

# Create means and standard deviations of test set scores
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

# Draw lines
plt.plot(train_sizes, train_mean, '--', color="#111111",  label="Traini
ng score")
plt.plot(train_sizes, test_mean, color="#111111", label="Validation sco
re")

# Draw bands
plt.fill_between(train_sizes, train_mean - train_std, train_mean + trai
n_std, color="#DDDDDD")
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_st
d, color="#DDDDDD")

# Create plot
plt.title("Learning Curve")
plt.xlabel("Training Set Size"), plt.ylabel("Accuracy Score"), plt.lege
nd(loc="best")
plt.tight_layout()
plt.show()
```

```
c:\users\aakash patel\appdata\local\programs\python\python37\lib\site-p
ackages\xgboost\core.py:587: FutureWarning: Series.base is deprecated a
```

```
nd will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
[23:40:06] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/sr

c/objective/regression_obj.cu:152: reg:linear is now deprecated in favo
r of reg:squarederror.
XGBoost Model-
Testing accuracy is=
0.7804915380213014
Training accuracy is=
0.8592333218114218
```



Testing performance of XGBoost Regressor is best among all the tried regression models. Comparing the performance of trainining and testing data in the learning curve we see that accuracy of testing gradually improves as training size increases.But as the training set size becomes 400 their is almost no growth and graph is linear. In the training score however, the accuracy is highest initially and doesn't show any growth as the training set size increases.

3.g. Parameter Tuning for XGBoost Model

In [181]:
```
param_grid = {
    'max_depth': [4, 6, 7],
'colsample_bytree': [0.3, 0.2],
    'learning_rate': [0.07,0.12],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 70]
}
grid_clf_acc = GridSearchCV(estimator=xgbmodel, param_grid = param_grid
,n_jobs=-1)
grid_clf_acc.fit(X_train, y_train)
grid_clf_acc.best_score_
print(grid_clf_acc.best_score_)
print(grid_clf_acc.best_params_ )
```

```
c:\users\aakash patel\appdata\local\programs\python\python37\lib\site-p
ackages\sklearn\model_selection\_split.py:1978: FutureWarning: The defa
ult value of cv will change from 3 to 5 in version 0.22. Specify it exp
licitly to silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
c:\users\aakash patel\appdata\local\programs\python\python37\lib\site-p
ackages\xgboost\core.py:587: FutureWarning: Series.base is deprecated a
nd will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
[23:44:35] WARNING: C:/Jenkins/workspace/xgboost-win64_release_0.90/sr
c/objective/regression_obj.cu:152: reg:linear is now deprecated in favo
r of reg:squarederror.
0.7528925171082093
{'colsample_bytree': 0.3, 'learning_rate': 0.12, 'max_depth': 7, 'min_s
amples_split': 8, 'n_estimators': 70}
```

To perform parameter tuning we provided we used GridSearchCV technique. It basically needs to be provided with parameter grid. Grid contains different parameters which impact model performance. There are different values provided for each parameter. This technique checks different combination of parameter values and generates best parameter value for each parameter (which will result in highest accuracy). Here as per output best score of 75% accuracy is achieved with different provided values of parameters. Values depicted show best values among their respective provided list.

The reason why it will improve the accuracy is because, it gives the best possible value of all attributes. As the number of provided attributes increases along with provided possiblities, one can determine which attribute to tweak to achieve good accuracy. As it increasingly time complex, results take quite a while to get displayed. Hence we have not tweaked the parameters further.

In [182]:

```
# param_grid = {
#     'colsample_bytree':[0.3, 0.2],
#                    'gamma':[0,1],
#                    'learning_rate':[0.08,0.12],
#                    'max_depth':[12,7],
#                    'min_child_weight':[1.5, 2],
#                    'n_estimators':[100,70],

#                    'reg_alpha':[10,12],
#                    'reg_lambda':[0.45,0.40],
#                    'subsample':[0.6,0.4],
#                    'seed':[42, 45]
# }
# grid_clf_acc = GridSearchCV(estimator=xgbmodel, param_grid = param_gr
id,n_jobs=-1)
# grid_clf_acc.fit(X_train, y_train)
# grid_clf_acc.best_score_
# print(grid_clf_acc.best_score_)
# print(grid_clf_acc.best_params_ )
```

Above performed GridSearch Parameter Tuning would take more than stipulated time to run and but, as it logically follows, this technique of GRidSearchCV will immensely improve the performance.

REFERENCES:

- https://machinelearningmastery.com/avoid-overfitting-by-early-stopping-with-xgboost-in-python/
- https://www.saedsayad.com/decision_tree_overfitting.htm
- https://stackoverflow.com/questions/43579180/feature-names-must-be-unique-xgboost?noredirect=1&lq=1
- https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74
- https://pythonspot.com/matplotlib-bar-chart/
- https://chrisalbon.com/machine_learning/model_evaluation/plot_the_learning_curve/
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html