In [34]:

```
import py_entitymatching as em #Import megallan entity matching library
import math
import warnings
warnings.filterwarnings('ignore')
```

In [35]:

```
def phone_match(str1,str2):
    if type(str1) is float and type(str2) is float:
        if math.isnan(str1) and math.isnan(str2):
            return True
    elif type(str1) is float:
        if math.isnan(str1):
            return False
    elif type(str2) is float:
        if math.isnan(str2):
            return False
    else:
        stra = ""
        strb = ""
        for ch in str1:
            if ch.isdigit():
                stra += ch
        for ch in str2:
            if ch.isdigit():
                strb += ch
        if stra == strb:
            return True
        else:
            return False
```

In [36]:

```
matching_records = em.read_csv_metadata("before_merging.csv",key="id")
yelp_original = em.read_csv_metadata("yelp_original.csv",key="id")
zomato_original = em.read_csv_metadata("zomato_original.csv",key="id")
```

In [37]:

```
matching_records.columns
```

Out[37]:

```
Index([u'id', u'Unnamed: 1', u'ltable_Name', u'ltable_Phone',
       u'ltable_Zipcode', u'ltable_State', u'ltable_City', u'ltable_Ad
dress',
       u'ltable_Delivery', u'ltable_Takeout', u'ltable_Outdoor_seating
',
       u'rtable_Name', u'rtable_Phone', u'rtable_Zipcode', u'rtable_St
ate',
       u'rtable_City', u'rtable_Address', u'rtable_Delivery',
       u'rtable_Takeout', u'rtable_Outdoor_seating', u'Label', u'predi
cted',
       u'restaurant_name', u'phone', u'zipcode', u'state', u'city', u'
address',
       u'delivery', u'takeout', u'outdoor_seating', u'pricy', u'rating
'],
      dtype='object')
```

In [38]:

```
yelp_original.columns # Original CSV file with added Pricyness column
```

Out[38]:

```
Index([u'id', u'Name', u'Phone', u'Price', u'Zipcode', u'State', u'Cit
y',
       u'Address', u'Has Delivery', u'Has Take-out', u'Has outdoor sea
ting',
       u'Parking'],
      dtype='object')
```

In [39]:

```
zomato_original.columns # Original CSV with added Rating column
```

Out[39]:

```
Index([u'id', u'Name', u'Phone', u'Rating', u'Price', u'Zipcode', u'St
ate',
       u'City', u'Address', u'Delivery', u'Takeout', u'Outdoor', u'Par
king'],
      dtype='object')
```

Schema Merging

In [41]:

```
indexes_to_keep = set()
index = 0
```

```python
for index in range(matching_records.shape[0]):
    tuple = matching_records.iloc[index]
    if tuple['predicted'] == 1:

        # Merging the Names -
        # Picking the one that has more length
        if len(tuple['ltable_Name']) > len(tuple['rtable_Name']):
            tuple['restaurant_name'] = tuple['ltable_Name']
        else:
            tuple['restaurant_name'] = tuple['rtable_Name']

        # Merging the Phone no -
        phone1 = tuple['ltable_Phone']
        phone2 = tuple['rtable_Phone']
        if phone_match(phone1, phone2) is True: # When phone numbers are same
            tuple['phone'] = phone1
        else: # Case when phone nos are different. We keep both separated by comma.
            tuple['phone'] = phone1+ "," + phone2

        # Merging the Zipcode -
        # Since blocking was done based on exact match for ZipCode, picking the lef
        tuple['zipcode'] = tuple['ltable_Zipcode']

        # Merging the State -
        # Picking the left table attribute
        tuple['state'] = tuple['ltable_State']

        # Merging the City -
        # Picking the left table attribute
        tuple['city'] = tuple['ltable_City']

        # Merging the Address
        # Picking the one that has more length
        if len(tuple['ltable_Address']) > len(tuple['rtable_Address']):
            tuple['address'] = tuple['ltable_Address']
        else:
            tuple['address'] = tuple['rtable_Address']


        tuple['delivery'] = tuple['rtable_Delivery']
        tuple['takeout'] = tuple['rtable_Takeout']
        tuple['outdoor_seating'] = tuple['ltable_Outdoor_seating']

        # For rating
        for ind in range(zomato_original.shape[0]):
            entry = zomato_original.iloc[ind]
            if entry['Name'] == tuple['rtable_Name']:
                tuple['rating'] = entry['Rating']
                break

        # For priciness
        for ind in range(yelp_original.shape[0]):
            entry = yelp_original.iloc[ind]
            if entry['Name'] == tuple['ltable_Name']:
                tuple['pricy'] = entry['Price']
                break

        # Updating the tuple in predications table
        matching_records.iloc[index] = tuple
        indexes_to_keep.add(index)

    index += 1
```

In [42]:

```
# Print the schema
matching_records.head(1)
```

Out[42]:

	id	Unnamed: 1	ltable_Name	ltable_Phone	ltable_Zipcode	ltable_State	ltable_City	ltable_Addr
0	1	512	McSorley's Old Ale House	(212) 473-9148	10003	NY	New York	15 E 7th

1 rows × 33 columns

In [44]:

```
# Fetch only those rows where predicted = "1" => get correctly matched tuples
sliced = matching_records.take(list(indexes_to_keep))

# Drop columns before merging.
# Dropping old attributes

del sliced['ltable_Name']
del sliced['rtable_Name']
del sliced['ltable_Phone']
del sliced['rtable_Phone']
del sliced['ltable_Zipcode']
del sliced['rtable_Zipcode']
del sliced['ltable_State']
del sliced['rtable_State']
del sliced['ltable_City']
del sliced['rtable_City']
del sliced['ltable_Address']
del sliced['rtable_Address']
del sliced['ltable_Delivery']
del sliced['rtable_Delivery']
del sliced['ltable_Takeout']
del sliced['rtable_Takeout']
del sliced['ltable_Outdoor_seating']
del sliced['rtable_Outdoor_seating']
del sliced['Label'] # Dropping the column'Label'
del sliced['predicted'] # Dropping the column 'predicted

sliced.to_csv("filtered_predictions.csv") # Writing the resultant table to a CSV fi
```