◎ Data Cleaning Project - Outlook Contacts

Importing:

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
joel = pd.read_csv("Joel_Hotmail.csv",encoding='Latin-1',dtype='object')
```

We will be creating a custom column called 'Contact'. This field consists of the first and last name attributes. This will make it easier to identify duplicate contacts. The customer says his not concerned with the middle name. But if that was the case, I would add the 'Middle Name' field into our concatenation.

In [3]:
```
joel["Contact"] = joel["First Name"].fillna('') +' '+ joel["Last Name"].fillna('')
```

Identifying the fields which contains at least one entry:

In [4]:
```
joel.dropna(axis=1,thresh=1).columns
```

Out[4]:
```
Index(['Title', 'First Name', 'Middle Name', 'Last Name', 'Suffix', 'Company',
       'Department', 'Job Title', 'Business Street', 'Business City',
       'Business State', 'Business Postal Code', 'Business Country/Region',
       'Home Street', 'Home City', 'Home State', 'Home Postal Code',
       'Home Country/Region', 'Other Street', 'Other City', 'Other State',
       'Other Postal Code', 'Other Country/Region', 'Business Fax',
       'Business Phone', 'Business Phone 2', 'Car Phone', 'Company Main Phone',
       'Home Fax', 'Home Phone', 'Home Phone 2', 'Mobile Phone', 'Other Phone',
       'Pager', 'Primary Phone', 'Anniversary', 'Birthday', 'Children',
       'E-mail Address', 'E-mail Type', 'E-mail Display Name',
       'E-mail 2 Address', 'E-mail 2 Type', 'E-mail 2 Display Name',
       'E-mail 3 Address', 'E-mail 3 Type', 'E-mail 3 Display Name', 'Gender',
       'Initials', 'Notes', 'Office Location', 'Priority', 'Private',
       'Sensitivity', 'Spouse', 'Web Page', 'Contact'],
      dtype='object')
```

Stored on the variable 'imp' will be the names of the columns which are considered 'important'.

In [5]:
```
imp = list(['Contact','Business Phone','Business Phone 2','Home Phone',
            'Home Phone 2','Mobile Phone','Other Phone',
            'E-mail Address','E-mail 2 Address', 'E-mail 3 Address'])
```

'imp2' will contain the same elements found on 'imp' with the addition of the 'Notes' field.

In [6]:
```
imp2=list(['Contact','Business Phone','Business Phone 2','Home Phone','Home Phone 2',
           'Mobile Phone','Other Phone',
           'E-mail Address','E-mail 2 Address', 'E-mail 3 Address','Notes'])
```

NOTE: The 'Notes' field is of utter importance for our client, since it contains meetings histories, comments, and sometimes even phone numbers and addresses (we will be dealing with these cases on section 1)

◎ Initial number of Records:

In [7]:
```
#Initial Number of Records:
len(joel.index)
```

Out[7]:
```
12155
```

◎ Initial Number of Duplicates on Contact

In [8]:
```
len(joel[joel.duplicated(subset=['Contact'],keep=False)])
```

Out[8]:
```
3613
```

In [9]:
```
len(joel[joel.duplicated(['First Name','Last Name'],keep=False)])
```

Out[9]:
```
3613
```

---

1. Preliminary cleaning: Cleaning 'Notes' Field

Before we start, there are some unwanted data on the 'Notes' field that we wish to get rid of. We're not actually deleting then, just turning them to blank notes so it doesn't mess up with our counting of nulled cells methods later on. It is important that we do this right at the begging so that later they don't get merged when we jump to the aggregate operations.

In [10]:
```
test = joel[joel['Notes'].str.contains("Contact Imported")==True]
```

Looking through the 'Notes' field I came across some notes like this one below:

In [11]:
```
# Example: Contact 329
test.loc[329]['Notes']
```

Out[11]:
```
'------------------------------\nContact Imported:\nEmailAddress : [redacted]
ica.com\nBusinessPhone : [redacted]482\nMobilePhone : [redacted]300\nLine1 : [redacted]
Street \nSuite [redacted]\nCity : [redacted]\nState : [redacted]\nPostalCode : [redacted]0'
```

We can see that this entry has some valuable information contained within them like Business Phones, Addresses, et cetera... Information which,in their turn, are not contained by the fields which they were supposed to be,within the record itself:

```
In [12]: # Example: Contact 329
         test.loc[329][imp2]
```

```
Out[12]: Contact                                                   Marcelo [redacted]
         Business Phone                                                        NaN
         Business Phone 2                                                      NaN
         Home Phone                                                            NaN
         Home Phone 2                                                          NaN
         Mobile Phone                                                          NaN
         Other Phone                                                           NaN
         E-mail Address                                                        NaN
         E-mail 2 Address                                                      NaN
         E-mail 3 Address                                                      NaN
         Notes                    -----------------------------\nContact Importe...
         Name: 329, dtype: object
```

```
In [13]: ## Non-null entries for Business Phone on this sub-dataframe
         test['Business Phone'].count()
```

```
Out[13]: 0
```

Our objective here is to fix this by extracting only the useful information from these notes.

First, we are going to break those lenghty strings into small elements and comprised them into lists. We're going to split them by the '\n' substring. This is a special character used to describe a new line in Unix/MacOS

```
In [14]: l = test.loc[:,'Notes'].apply(lambda x: x.split('\n'))
```

```
In [15]: # Example: Contact 329
         l[329]
```

```
Out[15]: ['-----------------------------',
          'Contact Imported:',
          'EmailAddress : [redacted]ica.com',
          'BusinessPhone : [redacted]482',
          'MobilePhone : [redacted]300',
          'Line1 : [redacted]Street ',
          'Suite[redacted]',
          'City : [redacted]',
          'State : [redacted]',
          'PostalCode : [redacted]0']
```

Next, lets separate the cream from the crop. All the useful data contained within these notes are characterized by the presence of the ' : ' substring. Lets filter out our lists so that it retains only the elements that contain those characters:

```
In [16]: def WhatMatters(l):
             list1 = []
             for i in l:
                 if i.find(' : ') != -1:
                     list1.append(i)
             return list1
```

In [17]:
```
l = l.apply(WhatMatters)
l.loc[329]
```

Out[17]:
```
['EmailAddress : [redacted]ica.com',
 'BusinessPhone : [redacted]482',
 'MobilePhone : [redacted]300',
 'Line1 : [redacted]Street ',
 'City : [redacted]',
 'State : [redacted]',
 'PostalCode : [redacted]0']
```

Now, lets format the names of the variables in question so that they match the labels of our main dataframe. For example: At the notes example above, we can see the we have the label 'Line1' followed by a colon and the information about a person's Address. In the original dataset, however, the field which contains this kind of data is called 'Business Street'. That's why we will be creating and applying a function called 'FitNames', which will be formatting the labels inside our lists and replacing them with the ones that match our dataset:

In [18]:
```
def FitNames(list1):
    for i in range(0,len(list1)):
        if list1[i].find("Phone ") != -1:
            list1[i] = (list1[i][:list1[i].find("P")] + ' ' +
                        list1[i][list1[i].find("P"):])
        elif list1[i].find('Email') != -1:
            list1[i] = (list1[i][:list1[i].find('mail')] + '-' +
                        list1[i][list1[i].find('mail'):list1[i].find('A')] + ' '+
                        list1[i][list1[i].find('A'):])
        elif list1[i].find('City') != -1:
            list1[i] = 'Business' + ' ' + list1[i][list1[i].find('City'):]
        elif list1[i].find('State :') != -1:
            list1[i] = 'Business' + ' ' + list1[i][list1[i].find('State'):]
        elif list1[i].find('Line1') != -1:
            list1[i] = 'Business Street' + list1[i][list1[i].find(' '):]
        elif list1[i].find('PostalCode') != -1:
            list1[i] = 'Business Postal Code' + list1[i][list1[i].find(' '):]
        elif list1[i].find('Country :') != -1:
            list1[i] = 'Home Country/Region' + ' '+ list1[i][list1[i].find(':'):]

    return list1
```

In [19]:
```
l.apply(FitNames)
# Example: Contact 329
l.loc[329]
```

Out[19]:
```
['E-mail Address :[redacted]ica.com',
 'Business Phone : [redacted]482',
 'Mobile Phone : [redacted]300',
 'Business Street : [redacted]Street ',
 'Business City : [redacted]',
 'Business State : [redacted]',
 'Business Postal Code : [redacted]0']
```

In [20]:
```
l = l.apply(WhatMatters)
```

```
In [21]: # Example: Contact 329
         l.loc[329]
```

```
Out[21]: ['E-mail Address : [redacted]ica.com',
          'Business Phone : [redacted]482',
          'Mobile Phone : [redacted]300',
          'Business Street : [redacted]Street ',
          'Business City : [redacted]',
          'Business State : [redacted]',
          'Business Postal Code : [redacted]0']
```

With our lists finally cleaned up and containing only relevant and relatively organized information, we're ready to proceed by converting such lists into dictionaries objects...

```
In [22]: def Dictionaries(list1):
             d={}
             temp =[]
             for i in list1:
                 temp=i.split(' : ')
                 d[temp[0]]=temp[1]
             return d
```

```
In [23]: l  =l.apply(Dictionaries)
```

Once turned into dictionaries, we can easily treat our data like pandas Series as well!

```
In [24]: # Example: Contact 329
         pd.Series(l.loc[329])
```

```
Out[24]: E-mail Address          [redacted]ica.com
         Business Phone                [redacted]482
         Mobile Phone                  [redacted]300
         Business Street       [redacted]Street
         Business City                    [redacted]
         Business State                   [redacted]
         Business Postal Code            [redacted]0
         dtype: object
```

```
In [25]: len(l)
```

```
Out[25]: 490
```

Finally, we just need to run one last command in order to merge our newly-discovered info. into our old dataframe. For that, we going to be using Pandas' built-in method called 'update'. This method will be vastly used throughout our cleaning process as it is much convenient to incorporate our changes into our target dataset.

```
In [26]: # Example -Before-:
         joel.loc[329][['Contact','Business Phone','Mobile Phone','Business Street',
                         'Business City','Business State',
                        'Business Postal Code']]
```

```
Out[26]: Contact                   Marcelo [redacted]
         Business Phone                          NaN
         Mobile Phone                            NaN
         Business Street                         NaN
         Business City                           NaN
         Business State                          NaN
         Business Postal Code                    NaN
         Name: 329, dtype: object
```

```
In [27]: for i in l.index:
             joel.loc[i].update(pd.Series(l[i]))
```

```
In [28]: # Example -After-:
         joel.loc[329][['Contact','Business Phone','Mobile Phone','Business Street',
                         'Business City','Business State',
                        'Business Postal Code']]
```

```
Out[28]: Contact                      Marcelo [redacted]
         Business Phone                    [redacted]482
         Mobile Phone                      [redacted]300
         Business Street           [redacted]Street
         Business City                          [redacted]
         Business State                           [redacted]
         Business Postal Code                   [redacted]0
         Name: 329, dtype: object
```

Now that we've extracted the useful contact information out of the notes containing the 'Contact Imported' substring, we're going to go ahead and turn them into blank:

```
In [29]: ## Deleting Unwanted notes
         joel.loc[joel['Notes'].str.contains("Contact Imported")==True,'Notes'] = ''
```

---

2. Records Missing Crucial Data:

Some records consist of just the contact name or no contact name at all. After speaking with the client, we decided that the best approach would be to just to delete them altogether.

```
In [30]: #Getting rid of records with nothing but missing values on 'First Name',
         # 'Middle Name' and 'Last Name' fields...
         joel = joel.dropna(subset=['First Name','Middle Name','Last Name'],how='all')
```

```
In [31]: #Getting rid of records with nothing but missing values on all other key fields...
         joel = joel.dropna(subset=['Business Fax','Business Phone','Business Phone 2',
                          'Home Phone','Home Phone 2',
                            'Mobile Phone','Other Phone','E-mail Address','E-mail 2 Address',
                                  'Company','Notes'],how='all')
```

◎ Current Number of Records:

In [32]:
```
len(joel.index)
```
Out[32]:
```
9802
```

---

3. Converging Data:

We will be now trying to condense our data inside our dataframe. We wil be moving information from relativaly 'less' important columns like 'Home Phone 2', 'Business Phone 2' into their respective main field (in this case: 'Home Phone' and 'Business Phone'). However, we will be moving only if theres no data already sitting at the target attribute, to avoid losing info.:

In [33]:
```
## Moving phone data into primary (target) field if there's space:
transfer =joel[joel['Home Phone 2'].notnull() & (joel['Home Phone'].isnull())].index
joel.loc[transfer,'Home Phone'] = joel.loc[transfer]['Home Phone 2']
# Deleting info from source attribute:
joel.loc[transfer,'Home Phone 2'] = np.nan
```

In [34]:
```
## Moving phone data into primary (target) field if there's space:
transfer =(joel[joel['Business Phone 2'].notnull() &
                (joel['Business Phone'].isnull())].index)
joel.loc[transfer,'Business Phone'] = joel.loc[transfer]['Business Phone 2']
# Deleting info from source attribute:
joel.loc[transfer,'Business Phone 2'] = np.nan
```

In [35]:
```
## Moving phone data into primary (target) field if there's space:
transfer = (joel[joel['Home Phone 2'].notnull() &
                 (joel['Other Phone'].isnull())].index)
joel.loc[transfer,'Other Phone'] = joel.loc[transfer]['Home Phone 2']
# Deleting info from source attribute:
joel.loc[transfer,'Home Phone 2'] = np.nan
```

In [36]:
```
## Moving phone data into primary (target) field if there's space:
transfer = (joel[joel['Other Phone'].notnull() &
                 (joel['Mobile Phone'].isnull())].index)
joel.loc[transfer,'Mobile Phone'] = joel.loc[transfer]['Other Phone']
# Deleting info from source attribute:
joel.loc[transfer,'Other Phone'] = np.nan
```

In [37]:
```
## Moving phone data into primary (target) field if there's space:
transfer = (joel[joel['E-mail 2 Address'].notnull() &
                 (joel['E-mail Address'].isnull())].index)
joel.loc[transfer,'E-mail Address'] = joel.loc[transfer]['E-mail 2 Address']
# Deleting info from source attribute:
joel.loc[transfer,'E-mail 2 Address'] = np.nan
```

3.1 Creating Custom Columns:

Let's also create custom fields which concatenate phones and emails from each record so that it facilitates our job later on...

I left the counter of elements of the most inclusive column for each merged field I'm creating just to make sure I'm not leaving anything behind or unchecked

In [38]:
```
joel['Home Phone'].count()
```
Out[38]:
```
2437
```

In [39]:
```
#Creating Field 'Home Phones' for analysis' sake
joel["Home Phones"] = (joel["Home Phone"].fillna('') +' '+
                       joel["Home Phone 2"].fillna(''))

joel.loc[joel['Home Phones'] == ' ','Home Phones'] = np.nan

joel['Home Phones'].count()
```
Out[39]:
```
2437
```

In [40]:
```
joel['Business Phone'].count()
```
Out[40]:
```
2950
```

In [41]:
```
#Creating Field 'Business Phones' for analysis' sake
joel["Business Phones"] = (joel["Business Phone"].fillna('') +' '+
                           joel["Business Phone 2"].fillna(''))

joel.loc[joel['Business Phones'] == ' ','Business Phones'] = np.nan

joel['Business Phones'].count()
```
Out[41]:
```
2950
```

In [42]:
```
joel['Mobile Phone'].count()
```
Out[42]:
```
4513
```

In [43]:
```
#Creating Field 'Other Phones' for analysis' sake
joel["Other Phones"] = (joel["Mobile Phone"].fillna('') +' '+
                        joel["Other Phone"].fillna(''))

joel.loc[joel['Other Phones'] == ' ','Other Phones'] = np.nan

joel['Other Phones'].count()
```
Out[43]:
```
4513
```

In [44]:
```
phones = list(['Contact','Business Phone','Business Phone 2','Home Phone',
               'Home Phone 2','Mobile Phone','Other Phone'])
```

In [45]:
```
joel['E-mail Address'].count()
```
Out[45]:
```
5588
```

In [46]:
```
##Creating Field 'E-mail Addresses' for analysis' sake
joel["E-mail Addresses"] = (joel["E-mail Address"].fillna('') +' '+
                            joel["E-mail 2 Address"].fillna('')+' '+
                            joel["E-mail 3 Address"].fillna(''))

joel.loc[joel['E-mail Addresses'] == '  ','E-mail Addresses'] = np.nan

joel['E-mail Addresses'].count()
```
Out[46]:
```
5588
```

4. Dealing with Duplicates:

◎ ***Current Number of Duplicates on Contact***

```
In [47]: len(joel[joel.duplicated(subset=['Contact'],keep=False)])

Out[47]: 1421
```

Let's begin by subsetting our main dataframe into a smaller one containing only records which have the same 'Contact' field information:

```
In [48]: duplicates = (joel[joel.duplicated(subset=['Contact'],keep=False)].
                       sort_values('Contact'))
```

Next, let's reduce our subset even more! let's identify records with same (duplicated) information also on important fields:

```
In [49]: duplicates=duplicates[duplicates.duplicated(subset=imp,keep=False)]
```

Now that we have taken the subsetted version of our dataframe containing only records with duplicated info on'Contact' AND on important fields besides 'Notes', we will converge all notes into the first occurrence and keep that occurence.

To do that, we are first going to find which columns need to be joined, which don't, as well as the ones which need to be aggregated differently: by keeping the first of the duplicate group.

```
In [50]: # Finding columns which have at least five entries:
         relevant = joel.dropna(axis=1,thresh=5).columns
         relevant

Out[50]: Index(['Title', 'First Name', 'Middle Name', 'Last Name', 'Suffix', 'Company',
                'Department', 'Job Title', 'Business Street', 'Business City',
                'Business State', 'Business Postal Code', 'Business Country/Region',
                'Home Street', 'Home City', 'Home State', 'Home Postal Code',
                'Home Country/Region', 'Other Street', 'Other City', 'Other State',
                'Other Postal Code', 'Other Country/Region', 'Business Fax',
                'Business Phone', 'Business Phone 2', 'Car Phone', 'Company Main Phone',
                'Home Fax', 'Home Phone', 'Home Phone 2', 'Mobile Phone', 'Other Phone',
                'Pager', 'Anniversary', 'Birthday', 'Children', 'E-mail Address',
                'E-mail Type', 'E-mail Display Name', 'E-mail 2 Address',
                'E-mail 2 Type', 'E-mail 2 Display Name', 'E-mail 3 Address',
                'E-mail 3 Type', 'E-mail 3 Display Name', 'Gender', 'Initials', 'Notes',
                'Office Location', 'Priority', 'Private', 'Sensitivity', 'Spouse',
                'Web Page', 'Contact', 'Home Phones', 'Business Phones', 'Other Phones',
                'E-mail Addresses'],
               dtype='object')
```

In [51]:
```
# These are the fields which consist of duplicated information in our current
# subset. They won't need to be joined since they only contain redundant
# info. They will be aggregated by deleting all occurences except by first inside eac
# duplicate group.

firsts = ['Business Phone','Business Phone 2','Home Phone','Home Phone 2',
          'Mobile Phone','Other Phone',
          'E-mail Address','E-mail 2 Address','E-mail 3 Address']
```

In [52]:
```
# Reffering to next three commands:
# The rest however will be aggregated by joining each entry into a single record,
# each entry will be separated by ' | ' inside the cell
join_these = list(relevant[~relevant.isin(firsts)])
```

In [53]:
```
# These, however, won't be needed to be joined, either because of their content or
# because of determinations made by client

list_to_remove=['First Name', 'Middle Name', 'Last Name','Contact',
 'Home Phones',
 'Business Phones',
 'Other Phones',
 'E-mail Addresses','Anniversary','Birthday','E-mail Type',
    'Gender','Priority ','Private ','Sensitivity','Priority',
                'Private','E-mail Display Name','Initials']
```

In [54]:
```
join_these= list(set(join_these).difference(set(list_to_remove)))
```

In [55]:
```
## Lets create a python dictionary which will give directions to our
# .agg method as to which type of aggregation ('first' or 'join')
# should be used to each attribute.

dicts = {}
value_1 = 'first'
value_2 = lambda x: ' | '.join(x.fillna('').astype(str))

for i in firsts:
    dicts[i] = value_1

for j in join_these:
    dicts[j] = value_2
```

In [56]:
```
duplicates.reset_index(inplace=True)

dicts['index'] = 'first'

# The aggregation happens here! Let's cast our dictionary inside the .agg method so
# it does its magic!
#                                                     \/
a= duplicates.groupby('Contact',as_index=False).agg(dicts)

a.set_index('index',inplace=True)
```

In [57]:
```
duplicates.set_index('index',inplace=True)
```

In [58]:
```
# duplcates Left JOIN a
duplicates.update(a, join='left', overwrite=True, filter_func=None,
                  raise_conflict=False)
```

Let's take a look at our 'duplicates' dataframe

```
In [61]: # EXAMPLE:
         duplicates[duplicates.Contact == ' Center'].dropna(axis=1)
```

Out[61]:

index	Middle Name	Last Name	Company	Business Street	Anniversary	Birthday	Gender	Notes	Pr
3943	Service	Center	U.S.C.I.S. | Immigration & Naturalization Serv...	[redacted]	0/0/00	0/0/00	Unspecified	[redacted]	N
4716	Forms	Center	Immigration & Naturalization Service	[redacted]	0/0/00	0/0/00	Unspecified	[redacted]	N
2671	Backlog Processing	Center	Employment & Training Administration	[redacted]	0/0/00	0/0/00	Unspecified	[redacted]	N

Let's take the 'Company' field for instance:

```
In [62]: duplicates.loc[3943]['Company']  # All info was joined to a single cell!
```

```
Out[62]: 'U.S.C.I.S. | Immigration & Naturalization Service | Employment & Training Adminis
         tration'
```

We can see that the data were joined, but it looks like we ended up with some unwanted '|' character. let's create a function that uses regular expressions to raise 'True' if a cell contains only '|' character.

```
In [63]: import re
         >>> def special_match(strg, search=re.compile(r'[^| .]').search):
         ...     return not bool(search(strg))
```

Now lets get rid of those unnecessary entries by applying the following lambda expression:

```
In [64]: duplicates = (duplicates.applymap(lambda x: np.nan if (isinstance(x,str)
                                         and special_match(x) == True) else x))
```

```
In [71]: # EXAMPLE:
         duplicates[duplicates.Contact == ' Center'][join_these].dropna(axis=1,thresh=1)
```

Out[71]:

index	Business Fax	Business Street	Business Postal Code	Notes	Company	Business State	Business City
3943	[redacted]	[redacted] | P.O. ...	[redacted]	[redacted]	U.S.C.I.S. | Immigration & Naturalization Serv...	[redacted]	[redacted]
4716	NaN	P.O. [redacted]	NaN	[redacted]	Immigration & Naturalization Service	NaN	NaN
2671	[redacted]	[redacted]	[redacted]	[redacted]	Employment & Training Administration	[redacted]	[redacted]

```
In [72]: # Let's also create a lambda function that uses a regular expression in order to
         # remove repeated character pattern
         # in a string (for preventing redundant info after the aggregation)
         import re
         duplicates[join_these] = (duplicates[join_these].
                                  applymap(lambda x: re.sub(r'(.+?)\1+', r'\1',
                                 x) if isinstance(x,str)  else x))
```

OK! Now, let's see how we got rid of many duplicates like these ones:

```
In [73]: ## Example -Before- :
         joel[joel['Contact']=='Tatiana [redacted]'][imp2].dropna(axis=1)
```

Out[73]:

	Contact	Mobile Phone	E-mail Address	Notes
721	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
722	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
723	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
724	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
725	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
726	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
727	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
728	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
729	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with Urbano.\r\n\r\n
730	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
731	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
732	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n
733	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with Urbano.\r\n\r\n
734	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	[redacted].
735	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n

Let's use the '.update' method to carry our changes to our main df 'joel'

```
In [74]: # test Left JOIN nodupes_phone
         joel.update(duplicates, join='left', overwrite=True, filter_func=None,
                     raise_conflict=False)
```

```
In [75]: # Drop the duplicates using column 'Contact' as reference but only keep the
         # most filled rows
         duplicates_2= duplicates.drop_duplicates(subset=['Contact'],keep='first')

         joel.drop(duplicates.index.difference(duplicates_2.index),inplace=True)
```

```
In [76]: ## Example -After- :
         joel[joel['Contact']=='Tatiana [redacted]'][imp2].dropna(axis=1)
```

Out[76]:

	Contact	Mobile Phone	E-mail Address	Notes
726	Tatiana [redacted]	[redacted]417	[redacted]@hotmail.com	Works with\r\n\r\n | Works with\r\n\r\n | Work...

◎ Current Number of Duplicates on 'Contact':

```
In [77]: len(joel[joel.duplicated(subset=['Contact'],keep=False)])
```

```
Out[77]: 926
```

As I began a deeper exploration of this dataset, I came across lots of almost identical duplicated records. That is, they had similar data on Contact names but presented divergent information on other fields such as E-mails, Business Phones, Home Phones and Mobile Phone. In the following subsections, we will be performing a series of methods to promote condensation of inforrmation into a single record within a group of 'duplicated' data.

Typical Example of this phenomena:

```
In [78]: joel[joel.Contact.str.contains('Matthew [redacted]')][imp2].dropna(axis=1)
```

Out[78]:

	Contact	Mobile Phone	E-mail Address
5529	Matthew [redacted]	[redacted]288	[redacted]@hotmail.com
6406	Matthew [redacted]	[redacted]720	[redacted]@gmail.com

4.1 Records with non-duplicated Mails and with duplicated name:

```
In [79]: nodupes_mail = (joel[((~joel.duplicated(subset=['E-mail Address'],
                           keep='last')) |(~joel.duplicated(subset=['E-mail Address'],
                           keep='first')) | (joel['E-mail Address'].isnull()))
                            &((joel.duplicated(subset=['Contact'], keep=False)))])
```

4.1.1 Dealing with Pairs

After identifying the overall redundant Names, we're going to first filter out those records which have only pairs of duplictates, so that we can move not similar e-mails in order to condense all information into a single record.

In [80]:
```
pair = nodupes_mail.groupby(nodupes_mail['Contact'].values.tolist(),
                            as_index=False).size() ==2
lista = list(pair[pair==True].index)


# Function to identify pairs of duplicated records
def pairs(x):
    if x in lista:
        return True
```

In [81]:
```
# Applying the function into the 'Contact' column and subsequently filtering
# our current subset:

nodupes_mail = nodupes_mail[nodupes_mail['Contact'].apply(pairs) == True]
```

Since we now have only pairs of records which have duplicated names and different e-mail addresses, we can safely move data from 'E-mail' to 'E-mail 2'.

In [82]:
```
## Moving those e-mails From 'E-mail' Field to 'E-mail 2' field

nodupes_mail['E-mail 2 Address'] = (nodupes_mail['E-mail Address']
        [nodupes_mail.duplicated(subset=['Contact'], keep=False)])
```

We're taking the approach of sorting by a 'count' column which counts the number of cells with no information on importante fields (imp2) for each row. That way, we make sure we're merging information into the most filled row of each duplicate group.

In [83]:
```
#sorting:
nodupes_mail['count'] = pd.isnull(nodupes_mail[imp2]).sum(1)
nodupes_mail= nodupes_mail.sort_values(['Contact','count','E-mail Address'],
                                       na_position='last')
```

Now we're going to go ahead and use the shift method to move up one cell on the E-mail 2 Address. That way we get the second address into our first record, which is the most filled one since we previously sorted by our 'count' column.

In [84]:
```
## Part 2.Moving  those e-mails From 'E-mail 2 Address' to the upper records
# (same Person)
### DISCLAIMER: MUST RUN THE FOLLOWING CODE ONLY ONCE!!

nodupes_mail['E-mail 2 Address'] = (nodupes_mail.groupby(['Contact'])
                                    ['E-mail 2 Address'].shift(-1))
```

Joining back to original dataset:

In [85]:
```
#test -Before-:
joel[joel['Contact']=='Miguel [redacted]'][imp2].dropna(axis=1)
```

Out[85]:

	Contact	Mobile Phone	E-mail Address	Notes
24	Miguel [redacted]	[redacted]843	[redacted]link.net	Amigo de [redacted]\r\n\r\n[redacted]r\n\r\nNot ...
25	Miguel [redacted]	[redacted]901	[redacted]ol.com.br	[redacted]a\n\nVeiu ...

In [86]:
```
# act Left JOIN nodupes_phone
joel.update(nodupes_mail, join='left', overwrite=True, filter_func=None,
            raise_conflict=False)
```

In [87]:
```
#test -After-: (We are currently concerned only with E-mail addresses)
joel[joel['Contact']=='Miguel [redacted]'][imp2].dropna(axis=1)
```

Out[87]:

	Contact	Mobile Phone	E-mail Address	Notes
24	Miguel [redacted]	[redacted]843	[redacted]link.net	Amigo de [redacted]o\r\n\r\n[redacted]r\n\r\nNot ...
25	Miguel [redacted]	[redacted]901	[redacted]ol.com.br	[redacted]\n\nVeiu ...

4.1.2 Dealing with trios

Now we're going to be dealing with trios of redundant records (on Contact name). I've taken this approach due to limitations of Outlook itself as it only offers three fields for E-mails. For bigger groups, We will be dealing with them on the later sections...

In [90]:
```
nodupes_mail = (joel[((~joel.duplicated(subset=['E-mail Addresses'],
                    keep='last')) | (~joel.duplicated(subset=['E-mail Addresses'],
                    keep='first')) | (joel['E-mail Addresses'].isnull()))  &
                     ((joel.duplicated(subset=['Contact'],keep=False)))])
```

In [91]:
```
pair = (nodupes_mail.groupby(nodupes_mail['Contact'].tolist(),
                             as_index=False).size() ==3)
lista = list(pair[pair==True].index)
nodupes_mail = nodupes_mail[nodupes_mail['Contact'].apply(pairs) == True]
```

We're going to be pretty much taking the same steps that we've taken for dealing with duplicates. The only difference being that now we have to include 'E-mail 3 Address' into the equation!

In [92]:
```
## Moving those emails From 'E-mail Address' Field to 'E-mail 2 Address' field
nodupes_mail['E-mail 2 Address'] = (nodupes_mail['E-mail Address'][nodupes_mail.
                                    duplicated(subset=['Contact'], keep=False)])
## Moving those emails From 'E-mail 2 Address' Field to 'E-mail 3 Address' field
nodupes_mail['E-mail 3 Address'] = nodupes_mail['E-mail 2 Address']
#sorting:
nodupes_mail['count'] = pd.isnull(nodupes_mail[imp2]).sum(1)
nodupes_mail= nodupes_mail.sort_values(['Contact','count','E-mail Address'],
                                       na_position='last')
nodupes_mail.drop('count',1,inplace=True)
#Shifting Values:
### DISCLAIMER: MUST RUN THE FOLLOWING CODE ONLY ONCE!!
nodupes_mail['E-mail 2 Address'] = (nodupes_mail.groupby(['Contact'])
                                    ['E-mail 2 Address'].shift(-1))
nodupes_mail['E-mail 3 Address'] = (nodupes_mail.groupby(['Contact'])
                                    ['E-mail 3 Address'].shift(-2))
```

Joining back to original dataset:

In [94]:
```
#test -Before-:
joel[joel['Contact']=='Selma [redacted]'][imp].dropna(axis=1,thresh=1)
```

Out[94]:

	Contact	E-mail Address
8758	Selma [redacted]	[redacted]lma@yahoo.com
10342	Selma [redacted]	[redacted]rini@msn.com
11465	Selma [redacted]	[redacted]66@yahoo.com

In [95]:
```
# act Left JOIN nodupes_mail
joel.update(nodupes_mail, join='left', overwrite=True, filter_func=None,
            raise_conflict=False)
```

In [96]:
```
#test -After-:
joel[joel['Contact']=='Selma [redacted]'][imp2].dropna(axis=1,thresh=1)
```

Out[96]:

	Contact	E-mail Address	E-mail 2 Address	E-mail 3 Address
8758	Selma [redacted]	[redacted]lma@yahoo.com	[redacted]rini@msn.com	[redacted]66@yahoo.com
10342	Selma [redacted]	[redacted]rini@msn.com	[redacted]66@yahoo.com	NaN
11465	Selma [redacted]	[redacted]66@yahoo.com	NaN	NaN

4.2 Records with non-duplicated Phones and with duplicated name:

Now we're going to be doing the same but with phone information. This will be a long subsection since we have many attributes for phone numbers:

In [97]:
```
phones
```

Out[97]:
```
['Contact',
 'Business Phone',
 'Business Phone 2',
 'Home Phone',
 'Home Phone 2',
 'Mobile Phone',
 'Other Phone']
```

4.2.1 Dealing with Pairs

In [98]:
```
# Function to identify pairs of duplicated records
def pairs(x):
    if x in lista:
        return True
```

Due to the particularities of the dataset, we will be merging non-similar phone information of paired duplicates by moving and spreading data into the corresponding pair of each phone field in a single contact record.

Each phone field has its correspondent equivalent for a second entry, for instance: 'Business Phone' has 'Business Phone 2', 'Home Phone' has 'Home Phone 2', 'Mobile Phone' has 'Other Phone'. 'Other Phone' will be treated as a "jack-of-all-trades" type of field".

Examples will be provided all the way through so this doesn't get overwhelming.

Business Phones

```
In [99]: nodupes_phone_bus = (joel[((~joel.duplicated(subset=['Business Phones'],
                             keep='last'))  |(~joel.duplicated(subset=['Business Phones'],
                             keep='first')) | (joel['Business Phones'].isnull())) &
                                  ((joel.duplicated(subset=['Contact'],keep=False)))])
```

```
In [100]: pair = nodupes_phone_bus.groupby(nodupes_phone_bus['Contact'].tolist(),
                                     as_index=False).size() ==2
          lista = list(pair[pair==True].index)
          nodupes_phone_bus = (nodupes_phone_bus[nodupes_phone_bus['Contact'].
                                            apply(pairs) == True])
```

```
In [101]: ## Moving those Phones From 'Business Phone' Field to 'Business Phone 2' field
          nodupes_phone_bus['Business Phone 2'] = (nodupes_phone_bus['Business Phone']
                          [nodupes_phone_bus.duplicated(subset=['Contact'], keep=False)])
```

We're taking the approach of sorting by a 'count' column which counts the number of cells with no information for each row. That way, we make sure we're merging information into the most filled row of each duplicate group

```
In [102]: #sorting:
          nodupes_phone_bus['count'] = pd.isnull(nodupes_phone_bus[imp2]).sum(1)
          nodupes_phone_bus= nodupes_phone_bus.sort_values(['Contact','count',
                                       'Business Phone'],na_position='last')
```

```
In [103]: ## Part 2.Moving  those Phones From 'M-Phone' to the upper records (same Person)
          ### DISCLAIMER: MUST RUN THE FOLLOWING CODE ONLY ONCE!!
          nodupes_phone_bus['Business Phone 2'] = (nodupes_phone_bus.groupby(['Contact'])
                                              ['Business Phone 2'].shift(-1))
```

```
In [106]: #Example -before-:
          joel[joel['Contact']==' Bezerra'][['Contact','Business Phone',
                                       'Business Phone 2']].dropna(axis=1,thresh=1)
```

Out[106]:

	Contact	Business Phone
2604	Bezerra	[redacted]979
3566	Bezerra	[redacted]672

```
In [107]: # joel Left JOIN nodupes_phone_bus
          joel.update(nodupes_phone_bus, join='left', overwrite=True, filter_func=None,
                      raise_conflict=False)
```

```
In [108]: #Example -after- (concerned only with 'Business Phone' at the moment):
          joel[joel['Contact']==' Bezerra'][['Contact','Business Phone',
                                'Business Phone 2']].dropna(axis=1,thresh=1)
```

Out[108]:

	Contact	Business Phone	Business Phone 2
2604	Bezerra	[redacted]979	NaN
3566	Bezerra	[redacted]672	[redacted]979

Home Phones

```
In [109]: nodupes_phone_hom = (joel[((~joel.duplicated(subset=['Home Phones'],
                                keep='last')) | (~joel.duplicated(subset=['Home Phones'],
                                keep='first')) | (joel['Home Phones'].isnull())) &
                                  ((joel.duplicated(subset=['Contact'], keep=False)))])
```

```
In [110]: pair = nodupes_phone_hom.groupby(nodupes_phone_hom['Contact'].tolist(),
                                           as_index=False).size() ==2
          lista = list(pair[pair==True].index)
          nodupes_phone_hom = (nodupes_phone_hom[nodupes_phone_hom['Contact'].
                                                 apply(pairs) == True])
```

```
In [111]: ## Moving those Phones From 'Home Phone' Field to 'Home Phone 2' field
          nodupes_phone_hom['Home Phone 2'] = (nodupes_phone_hom['Home Phone']
                          [nodupes_phone_hom.duplicated(subset=['Contact'], keep=False)])
```

```
In [112]: #sorting:
          nodupes_phone_hom['count'] = pd.isnull(nodupes_phone_hom[imp2]).sum(1)
          nodupes_phone_hom= nodupes_phone_hom.sort_values(['Contact','count',
                                                 'Home Phone'],na_position='last')
```

```
In [113]: ## Part 2.Moving  those Phones From 'M-Phone' to the upper records (same Person)
          ### DISCLAIMER: MUST RUN THE FOLLOWING CODE ONLY ONCE!!
          nodupes_phone_hom['Home Phone 2'] = (nodupes_phone_hom.groupby(['Contact'])
                                               ['Home Phone 2'].shift(-1))
```

```
In [114]: #Example -before-:
          joel[joel['Contact']=='Victor [redacted]'][phones].dropna(axis=1,thresh=1)
```

Out[114]:

	Contact	Home Phone
5427	Victor [redacted]	[redacted]101
6055	Victor [redacted]	[redacted]410

```
In [115]: # joel Left JOIN nodupes_phone_hom
          joel.update(nodupes_phone_hom, join='left', overwrite=True, filter_func=None,
                      raise_conflict=False)
```

```
In [116]: #Example -after-:
          joel[joel['Contact']=='Victor [redacted]'][phones].dropna(axis=1,thresh=1)
```

Out[116]:

	Contact	Home Phone	Home Phone 2
5427	Victor [redacted]	[redacted]101	[redacted]410
6055	Victor [redacted]	[redacted]410	NaN

Mobile and Other Phones

In [117]:
```
nodupes_phone_other = (joel[((~joel.duplicated(subset=['Other Phones'],
                        keep='last')) | (~joel.duplicated(subset=['Other Phones'],
                        keep='first')) | (joel['Other Phones'].isnull())) &
                            ((joel.duplicated(subset=['Contact'],keep=False)))]
                      )
```

In [118]:
```
pair = nodupes_phone_other.groupby(nodupes_phone_other['Contact'].tolist(),
                                   as_index=False).size() ==2
lista = list(pair[pair==True].index)
nodupes_phone_other = (nodupes_phone_other[nodupes_phone_other['Contact'].
                                        apply(pairs) == True])
```

In [119]:
```
## Moving those Phones From 'Mobile Phone' Field to 'Other Phone' field
nodupes_phone_other['Other Phone'] = (nodupes_phone_other['Mobile Phone']
            [nodupes_phone_other.duplicated(subset=['Contact'], keep=False)])
```

In [120]:
```
#sorting:
nodupes_phone_other['count'] = pd.isnull(nodupes_phone_other[imp2]).sum(1)
nodupes_phone_other= nodupes_phone_other.sort_values(['Contact','count',
                                    'Mobile Phone'],na_position='last')
```

In [121]:
```
## Part 2.Moving  those Phones From 'M-Phone' to the upper records (same Person)
### DISCLAIMER: MUST RUN THE FOLLOWING CODE ONLY ONCE!!
nodupes_phone_other['Other Phone'] = (nodupes_phone_other.groupby(['Contact'])
                                    ['Other Phone'].shift(-1))
```

In [122]:
```
#Example -Before-:
joel[joel['Contact']=='Maria [redacted]'][phones].dropna(axis=1,thresh=1)
```

Out[122]:

	Contact	Mobile Phone
706	Maria [redacted]	[redacted]994
7685	Maria [redacted]	[redacted]598

In [123]:
```
# joel Left JOIN nodupes_phone_other
joel.update(nodupes_phone_other, join='left', overwrite=True, filter_func=None,
            raise_conflict=False)
```

In [124]:
```
#Example -After-:
joel[joel['Contact']=='Maria [redacted]'][phones].dropna(axis=1,thresh=1)
```

Out[124]:

	Contact	Mobile Phone	Other Phone
706	Maria [redacted]	[redacted]994	[redacted]598
7685	Maria [redacted]	[redacted]598	NaN

Result of complete condensing:

In [125]:
```
#Example -after-:
joel[joel['Contact']==' Bezerra'][phones].dropna(axis=1,thresh=1)
```

Out[125]:

	Contact	Business Phone	Business Phone 2	Home Phone	Home Phone 2	Mobile Phone	Other Phone
2604	Bezerra	[redacted]979	NaN	[redacted]874	NaN	NaN	NaN
3566	Bezerra	[redacted]672	[redacted]979	[redacted]672	[redacted]874	[redacted]691	[redacted]672

In order to accomodate trios of different numbers for redundant contacts, we're going to merge the numbers into trios of non-conflicted fields. It will be the same as the pairs approach but with the addition of dumping the third phone into the 'Other Phone' attribute. Save for the last field 'Mobile Phone'. On this one we're going to have to improvise! We're going to move the second number into 'Other Phone' and then we will be dumping the third phone into 'Business 2 Phone'

4.2.2 Exceptions: Dealing with trios

Business Phone to Business Phone 2 and Other Phone

In [126]:
```
nodupes_phone_bus = (joel[((~joel.duplicated(subset=['Business Phones'],
                        keep='last')) | (~joel.duplicated(subset=['Business Phones'],
                        keep='first')) | (joel['Business Phones'].isnull())) &
                        ((joel.duplicated(subset=['Contact'],  keep=False)))])
```

In [127]:
```
# Function to identify pairs of duplicated records
def pairs(x):
    if x in lista:
        return True
```

In [128]:
```
pair = nodupes_phone_bus.groupby(nodupes_phone_bus['Contact'].tolist(),
                                 as_index=False).size() ==3
lista = list(pair[pair==True].index)
nodupes_phone_bus = (nodupes_phone_bus[nodupes_phone_bus['Contact'].
                                       apply(pairs) == True])
```

In [129]:
```python
## Moving those Phones From 'Business Phone' Field to 'Business Phone 2' field
nodupes_phone_bus['Business Phone 2'] = (nodupes_phone_bus['Business Phone']
                [nodupes_phone_bus.duplicated(subset=['Contact'], keep=False)])
## Moving those Phones From 'Business Phone 2' Field to 'Other Phone' field
nodupes_phone_bus['Other Phone'] = nodupes_phone_bus['Business Phone 2']
#sorting:
nodupes_phone_bus['count'] = pd.isnull(nodupes_phone_bus[imp2]).sum(1)
nodupes_phone_bus= nodupes_phone_bus.sort_values(['Contact','count',
                                'Business Phone'],na_position='last')
nodupes_phone_bus.drop('count',1,inplace=True)
#Shifting Values:
### DISCLAIMER: MUST RUN THE FOLLOWING CODE ONLY ONCE!!
nodupes_phone_bus['Business Phone 2'] = (nodupes_phone_bus.groupby(['Contact'])
                                    ['Business Phone 2'].shift(-1))
nodupes_phone_bus['Other Phone'] = (nodupes_phone_bus.groupby(['Contact'])
                                   ['Other Phone'].shift(-2))
```

Joining back to original Data set and deleting less useful rows:

In [130]:
```python
#test -Before-:
joel[joel['Contact']==' Arruda'][['Contact','Business Phone',
                                'Business Phone 2','Other Phone','E-mail Address',
                                  'E-mail 2 Address','E-mail 3 Address']]
```

Out[130]:

	Contact	Business Phone	Business Phone 2	Other Phone	E-mail Address	E-mail 2 Address	
2167	Arruda	[redacted] [redacted]035	NaN	NaN	[redacted]da@i[redacted]	[redacted]dante@yahoo.com.br	[redacted]
2642	Arruda	[redacted] [redacted]222	NaN	NaN	[redacted]ruda@i[redacted]	[redacted]da@[redacted]	[redacted]
4872	Arruda	[redacted] [redacted]256	NaN	NaN	[redacted]dante@yahoo.com.br	NaN	

In [131]:
```python
# joel Left JOIN nodupes_phone_bus
joel.update(nodupes_phone_bus, join='left', overwrite=True, filter_func=None,
            raise_conflict=False)
```

In [132]:
```python
#test -After-:
joel[joel['Contact']==' Arruda'][['Contact','Business Phone',
                                'Business Phone 2','Other Phone','E-mail Address',
                                  'E-mail 2 Address','E-mail 3 Address']]
```

Out[132]:

	Contact	Business Phone	Business Phone 2	Other Phone	E-mail Address	E-mail 2 Address	
2167	Arruda	[redacted] [redacted]035	[redacted] [redacted]256	NaN	[redacted]da@[redacted]r	[redacted]dante@yahoo.com.br	fe
2642	Arruda	[redacted] [redacted]222	[redacted] [redacted]035	[redacted] [redacted]256	[redacted]ruda@[redacted]	[redacted]da@[redacted]	[redacted]
4872	Arruda	[redacted] [redacted]256	NaN	NaN	[redacted]dante@yahoo.com.br	NaN	

Home Phone to Home Phone 2 and Other Phone

In [133]:
```
nodupes_phone_hom = (joel[((~joel.duplicated(subset=['Home Phones'],
                        keep='last')) | (~joel.duplicated(subset=['Home Phones'],
                               keep='first')) | (joel['Home Phones'].isnull())) &
                      ((joel.duplicated(subset=['Contact'], keep=False)))])
```

In [134]:
```
pair = nodupes_phone_hom.groupby(nodupes_phone_hom['Contact'].tolist(),
                                 as_index=False).size() ==3
lista = list(pair[pair==True].index)
nodupes_phone_hom = (nodupes_phone_hom[nodupes_phone_hom['Contact'].
                                       apply(pairs) == True])
```

In [135]:
```
## Moving those Phones From 'Home Phone' Field to 'Home Phone 2' field
nodupes_phone_hom['Home Phone 2'] = (nodupes_phone_hom['Home Phone']
            [nodupes_phone_hom.duplicated(subset=['Contact'], keep=False)])
## Moving those Phones From 'Home Phone 2' Field to 'Other Phone' field
nodupes_phone_hom['Other Phone'] = nodupes_phone_hom['Home Phone 2']
#sorting:
nodupes_phone_hom['count'] = pd.isnull(nodupes_phone_hom[imp2]).sum(1)
nodupes_phone_hom= nodupes_phone_hom.sort_values(['Contact','count','Home Phone'],
                                                 na_position='last')
nodupes_phone_hom.drop('count',1,inplace=True)
#Shifting Values:
### DISCLAIMER: MUST RUN THE FOLLOWING CODE ONLY ONCE!!
nodupes_phone_hom['Home Phone 2'] = (nodupes_phone_hom.groupby(['Contact'])
                                     ['Home Phone 2'].shift(-1))
nodupes_phone_hom['Other Phone'] = (nodupes_phone_hom.groupby(['Contact'])
                                    ['Other Phone'].shift(-2))
```

In [136]:
```
#Test - Before:
joel[joel.Contact==' Maia'][['Contact','Home Phone','Home Phone 2','Other Phone',
                             'E-mail Address','E-mail 2 Address','E-mail 3 Address']]
```

Out[136]:

	Contact	Home Phone	Home Phone 2	Other Phone	E-mail Address	E-mail 2 Address	E-mail 3 Address
2603	Maia	[redacted] [redacted]219	NaN	NaN	[redacted]@terra.com.br	[redacted]@cibt.com	NaN
3107	Maia	[redacted]428	NaN	NaN	l[redacted]@cibt.com	NaN	NaN
3802	Maia	NaN	NaN	NaN	NaN	NaN	NaN

In [137]:
```
# joel Left JOIN nodupes_phone_hom
joel.update(nodupes_phone_hom, join='left', overwrite=True, filter_func=None,
            raise_conflict=False)
```

In [138]:
```
#Testa - After:
joel[joel.Contact==' Maia'][['Contact','Home Phone','Home Phone 2','Other Phone',
                             'E-mail Address','E-mail 2 Address','E-mail 3 Address']]
```

Out[138]:

	Contact	Home Phone	Home Phone 2	Other Phone	E-mail Address	E-mail 2 Address	E-mail 3 Address
2603	Maia	[redacted] [redacted]219	[redacted]428	NaN	[redacted]@terra.com.br	[redacted]@cibt.com	NaN
3107	Maia	[redacted]428	NaN	NaN	[redacted]@cibt.com	NaN	NaN
3802	Maia	NaN	NaN	NaN	NaN	NaN	NaN

Mobile Phone to Other Phone and Business Phone 2

In [139]:
```
nodupes_phone_other = (joel[((~joel.duplicated(subset=['Other Phones'],
                           keep='last')) | (~joel.duplicated(subset=['Other Phones'],
                           keep='first')) | (joel['Other Phones'].isnull())) &
                              ((joel.duplicated(subset=['Contact'], keep=False)))])
```

In [140]:
```
pair = nodupes_phone_other.groupby(nodupes_phone_other['Contact'].tolist(),
                                    as_index=False).size() ==3
lista = list(pair[pair==True].index)
nodupes_phone_other = (nodupes_phone_other[nodupes_phone_other['Contact'].
                                            apply(pairs) == True])
```

In [141]:
```
## Moving those Phones From 'Mobile Phone' Field to 'Other Phone' field
nodupes_phone_other['Other Phone'] = (nodupes_phone_other['Mobile Phone']
                [nodupes_phone_other.duplicated(subset=['Contact'], keep=False)])
## Moving those Phones From 'Other Phone' Field to 'Business Phone 2' field
nodupes_phone_other['Business Phone 2'] = nodupes_phone_other['Other Phone']
#sorting:
nodupes_phone_other['count'] = pd.isnull(nodupes_phone_other[imp2]).sum(1)
nodupes_phone_other= nodupes_phone_other.sort_values(['Contact','count',
                                    'Mobile Phone'],na_position='last')
nodupes_phone_other.drop('count',1,inplace=True)
#Shifting Values:
### DISCLAIMER: MUST RUN THE FOLLOWING CODE ONLY ONCE!!
nodupes_phone_other['Other Phone'] = (nodupes_phone_other.groupby(['Contact'])
                                      ['Other Phone'].shift(-1))
nodupes_phone_other['Business Phone 2'] = (nodupes_phone_other.groupby(['Contact'])
                                        ['Business Phone 2'].shift(-2))
```

In [142]:
```
# Test: Before
joel[joel['Contact'].isin(['Valentin [redacted]'])][['Contact','Mobile Phone'
                ,'Other Phone','Business Phone 2']].sort_values('Contact')
```

Out[142]:

	Contact	Mobile Phone	Other Phone	Business Phone 2
146	Valentin [redacted]	[redacted]9-10	NaN	NaN
5658	Valentin [redacted]	[redacted]671	NaN	NaN
6706	Valentin [redacted]	[redacted]169	NaN	NaN

Joining back to original Data set and deleting less useful rows:

In [143]:
```
# joel Left JOIN nodupes_phone_bus
joel.update(nodupes_phone_other, join='left', overwrite=True, filter_func=None,
            raise_conflict=False)
```

In [144]:
```
# Test: After
joel[joel['Contact'].isin(['Valentin [redacted]'])][['Contact','Mobile Phone'
                ,'Other Phone','Business Phone 2','E-mail Address',
                'E-mail 2 Address','E-mail 3 Address']].sort_values('Contact')
```

Out[144]:

	Contact	Mobile Phone	Other Phone	Business Phone 2	E-mail Address	E-mail 2 Address	E-ma Addre
146	Valentin [redacted]	[redacted]9-10	NaN	NaN	[redacted]@gmail.com	NaN	N
5658	Valentin [redacted]	[redacted]671	[redacted]9-10	NaN	NaN	NaN	N
6706	Valentin [redacted]	[redacted]169	[redacted]671	[redacted]9-10	[redacted]@mail.ru	[redacted]@gmail.com	N

4.3 Excluding Records with duplicated names (pairs and trios):

Until now, we haven't deleted a single duplicate record. However, we will be doing this in the upcoming section. In order to prevent data loss, we will be using our well-known methodology which counts the number of blank fields for each records. Sorting the records by this counter in ascending order will prove to be convenient after grouping the duplicates as we end up with the most filled records as the first occurence inside the group. After that, we will proceed by excluding the ones underneath the first occurence.

In [145]:
```
## Finding once again records with duplicated contact name:

dupes = joel[(joel.duplicated(subset=['Contact'], keep=False))]
```

In [146]:
```
# Function to identify pairs and trios of duplicated records

pair = dupes.groupby(dupes['Contact'].tolist(),as_index=False).size() <4
lista = list(pair[pair==True].index)
dupes = dupes[dupes['Contact'].apply(pairs) == True]
```

We can see that out of the 826 records on this subset, only 384 are unique in their name. That means that more than half of the records are somewhat redundant!

In [147]:
```
len(dupes)
```

Out[147]:
```
826
```

In [148]:
```
dupes['Contact'].nunique()
```

Out[148]:
```
384
```

In [149]:
```
dupes['count'] = pd.isnull(dupes[imp2]).sum(1)
dupes = dupes.sort_values(['count'])
```

But before we delete our records, We first need to merge all notes from each record group into that first, more-filled observation.

In [150]:
```
dupes.reset_index(inplace=True)

a= dupes.groupby('Contact',as_index=False).agg(dicts)

a.set_index('index',inplace=True)
```

In [151]:
```
dupes.set_index('index',inplace=True)
```

In [152]:
```
# dupes Left JOIN a
dupes.update(a, join='left', overwrite=True, filter_func=None, raise_conflict=False)
```

In [153]:
```
dupes = (dupes.applymap(lambda x: np.nan if (isinstance(x,str) and
                                  special_match(x) == True) else x))
```

In [154]:
```
# Remove repeated character pattern in a string (for preventing redundant info after
# the aggregation)
import re
dupes[join_these] = dupes[join_these].applymap(lambda x: re.sub(r'(.+?)\1+',
                                     r'\1', x) if isinstance(x,str)  else x)
```

In [155]:
```
# Test -Before-:
joel[joel['Contact']=='Miguel [redacted]'][imp2].dropna(axis=1,
                                                     thresh=1)
```

Out[155]:

	Contact	Mobile Phone	Other Phone	E-mail Address	E-mail 2 Address	Notes
24	Miguel [redacted] [redacted]	[redacted]843	[redacted]901	[redacted]ink.net	[redacted]ol.com.br	Amigo de [redacted] [redacted]\r \n\r\n[redacted] \r\n\r\nNot ...
25	Miguel [redacted] [redacted]	[redacted]901	NaN	[redacted]ol.com.br	NaN	[redacted]r [redacted] [redacted]\n\nVeiu ...

In [156]:
```
# Test -Before-:
joel[joel['Contact']==' Alcantara'][imp2].dropna(axis=1,
                                           thresh=1)
```

Out[156]:

	Contact	Mobile Phone	E-mail Address	E-mail 2 Address	Notes
2403	Alcantara	NaN	[redacted]togo.com	[redacted]maiol.com	[redacted] [redacted]...
4624	Alcantara	[redacted] [redacted]478	[redacted]maiol.com	NaN	NaN

In [157]:
```
# joel Left JOIN dupes
joel.update(dupes, join='left', overwrite=True, filter_func=None, raise_conflict=Fals
```

In [158]:
```
#drop the duplicates using column 'Contact' and 'E-mail' as reference but only
# keep the most filled rows
dupes_2= dupes.drop_duplicates(subset=['Contact'],keep='first')

joel.drop(dupes.index.difference(dupes_2.index),inplace=True)
```

In [159]:
```
# Test -After-
joel[joel['Contact']=='Miguel [redacted]'][imp2].dropna(axis=1,
                                                   thresh=1)
```

Out[159]:

	Contact	Mobile Phone	Other Phone	E-mail Address	E-mail 2 Address	Notes
24	Miguel [redacted]	[redacted]843	[redacted]901	[redacted]ink.net	[redacted]ol.com.br	Amigo de [redacted]\r\n\r\n[redacted]\r\n\r\nNot o...

In [160]:
```
joel.loc[joel[joel['Contact']=='Miguel [redacted]'].index.item()]['Notes']
```

Out[160]:
```
'Amigo de [redacted]\r\n\r\n[redacted]\r\n\r\nNot on Linked In\r\n\r\nCalvario City Ch
urch\r\nCcorl.org\r\n[redacted] & [redacted]\r\nIntercesion [redacted]
Portuguese [redacted]\r\n[redacted]\r\n[redacted], [redacted] [redacted]\r\n\r\n\r\n | [redacted]
[redacted]\n\nVeiu de [redacted]\n\nFilho de [redacted]
. \n\nFilho [redacted], [redacted], [redacted] venda compra de caros, pais
agradáveis, talvez querem investir [redacted], pode ser em nome do filho?\n'
```

In [161]:
```
# Test -After-:
joel[joel['Contact']==' Alcantara'][imp2].dropna(axis=1,
                                                   thresh=1)
```

Out[161]:

	Contact	Mobile Phone	E-mail Address	E-mail 2 Address	Notes
2403	Alcantara	[redacted]478	[redacted]togo.com	[redacted]maiol.com	[redacted]y...

◎ Current Number of Records:

In [162]:
```
len(joel.index)
```

Out[162]:
```
9074
```

5. Data on wrong Fields:

In this section, we will be moving information that somehow got mistakenly inputted into another field to the one that it is supposed to be.

Functions for finding non-alphabetical data on supposedly alphabetical fields and non-numeric data on supposedly Numerical fields

```
In [163]: ## Function for finding if string is numerical or not:
          def numeric(x):
              if pd.isna(x) == False:
                  x = (x.replace(' ','').replace('+','').replace('(','').
                       replace(')','').replace('-',''))
                  return x.isnumeric()


          ## Function for finding if string is aplhabetical or not
          def alpha(x):
              if pd.isna(x) == False:
                  x= x.replace(".","").replace(' ',"").replace("'","").replace("`","")
                  x= x.replace("-","").replace('?',"").replace('(',
                                                              '').replace(')','').replace('"','')
                  x= x.replace(':','').replace('&','').replace(",",'').replace('/','')
                  return x.isalpha()
```

5.1 Non-alphabetical Data on 'Contact'

```
In [164]: #Non alphabetical on 'Contact'
          nonalpha = joel[joel['Contact'].apply(alpha) == False]

          imp = ['Contact','Business Phone','Business Phone 2','Home Phone','Home Phone 2',
                 'Mobile Phone','Other Phone',
           'E-mail Address','E-mail 2 Address']
```

```
In [165]: len(nonalpha)

Out[165]: 116
```

```
In [166]: ## Filtering only emails on 'Contact' field
          ContactMail = joel[joel['Contact'].str.contains("@")==True]
          len(ContactMail)

Out[166]: 40
```

```
In [167]: ## Moving those mails From 'Contact' Field to 'E-mail 2 Address' field
          ContactMail.loc[:]['E-mail 2 Address'] = ContactMail['Contact']
```

```
In [168]: ContactMail[['Contact','E-mail 2 Address']].head()

Out[168]:
```

	Contact	E-mail 2 Address
164	[redacted]@aa.com	[redacted]@aa.com
175	[redacted]@gmail.com	[redacted]@gmail.com
195	[redacted]@bberry.com	[redacted]@bberry.com
219	[redacted]@gmail.com	[redacted]@gmail.com
260	[redacted]@bells[redacted]	[redacted]@bells[redacted]

```
In [169]: # act Left JOIN PhoneMail
          joel.update(ContactMail, join='left', overwrite=True, filter_func=None,
                      raise_conflict=False)
```

In [170]:
```python
# Excluding Contacts that don't have at least one Phone Number:
ContactMail_2 = (ContactMail.drop(
               ContactMail.loc[(ContactMail['Business Phones'].isnull()) &
               (ContactMail['Home Phones'].isnull()) &
                    (ContactMail['Other Phones'].isnull())].index))
```

In [171]:
```python
joel.drop(ContactMail.index.difference(ContactMail_2.index),inplace=True)
```

Go over these with client and see the best way to deal with those Records

In [172]:
```python
nonalpha = joel[joel['Contact'].apply(alpha) == False]
nonalpha[imp2].head()
```

Out[172]:

	Contact	Business Phone	Business Phone 2	Home Phone	Home Phone 2	Mobile Phone	Other Phone	E-mail Address
8	[redacted] 12/15	NaN	NaN	NaN	NaN	NaN	NaN	Na
42	Marcia [redacted]866	NaN	NaN	NaN	NaN	[redacted]866	NaN	Na
124	[redacted]«Bob ' [redacted]	[redacted]559	NaN	NaN	NaN	NaN	NaN	bob@[redacted]
151	????? ??????	NaN	NaN	NaN	NaN	NaN	NaN	[redacted]t@rambler.ru
202	[redacted]2 Mindy	NaN	NaN	[redacted]696	NaN	NaN	NaN	Na

Fixing the encoding:

In [173]:
```python
def decoder(name):
    if name[0]==' ':
        name = name.replace(' ','',1)
    else:
        name = name
    name= name.encode('Latin-1',errors='ignore').decode('utf-8',errors='ignore')
    return name

```

In [174]:
```python
nonalpha.loc[:]['Contact'] = nonalpha['Contact'].apply(decoder)
```

In [175]:
```python
# act Left JOIN nonalpha
joel.update(nonalpha, join='left', overwrite=True, filter_func=None,
             raise_conflict=False)
```

In [176]:
```python
## Test -After-:
joel.loc[[1139,895,2595,3419,2257,2639,7536,8959],'Contact']
```

Out[176]:
```
1139     d[redacted] Gomes
895                Brandão
2595               Rancaño
3419                 Núñez
2257        d[redacted] Galvão
2639          [redacted]руков
7536            Noué [redacted]
8959           [redacted]ений
Name: Contact, dtype: object
```

6 Duplicate Exceptions

We've successfully taken care of the fully redundant records in section 4. We then proceeded by dealing with doubles and trios of almost-duplicated rows in subsection 4.1 , 4.2 and 4.3.

At last, we are now going to be dealing with the almost-duplicate records which surpassed the trio status regarding their number of instances inside their 'Contact' name group.

◎ Current Number of Duplicates on Contact

```
In [177]: len(joel[joel.duplicated(subset=['Contact'],keep=False)])

Out[177]: 102
```

```
In [178]: imp = ['Contact',
           'Business Phone',
           'Business Phone 2',
           'Home Phone',
              'Home Phone 2',
           'Mobile Phone',
                 'Other Phone',
           'E-mail Address',
           'E-mail 2 Address',
                'count']
```

```
In [179]: remaining = joel[joel.duplicated(subset=['Contact'],
                                          keep=False)].sort_values('Contact')
```

```
In [180]: remaining = joel[joel.duplicated(subset=['Contact'],
                                          keep=False)].sort_values('Contact')

          # Let's get our 'count' in place once again:

          remaining['count'] = pd.isnull(remaining).sum(1)
```

The next sets of duplicated contacts will have its relevant information joined/aggregated into the first occurrence (most filled one). note that we are not able to 'spread' the data between multiple fields simply because, in this particular case, Outlook doesn't offer that many fields.

```
In [181]: save =remaining
          save['count'] = pd.isnull(save).sum(1)
          save = save.sort_values(['Contact','count'])
```

```
In [182]: dicts = {}
          value_1 = lambda x: '| '.join(x.fillna('').astype(str))

          for i in firsts + join_these:
              dicts[i] = value_1
```

In [183]:
```
save.reset_index(inplace=True)

dicts['index'] = 'first'

# Aggregation:

a= save.groupby('Contact',as_index=False).agg(dicts)

a.set_index('index',inplace=True)
```

In [184]:
```
save.set_index('index',inplace=True)
```

In [185]:
```
# save Left JOIN a
save.update(a, join='left', overwrite=True, filter_func=None, raise_conflict=False)
```

In [186]:
```
save = (save.applymap(lambda x:
       np.nan if (isinstance(x,str) and special_match(x) == True) else x))
```

In [187]:
```
# Remove repeated character pattern in a string (for preventing redundant
# info after the aggregation)

import re
save[join_these] = save[join_these].applymap(lambda x: re.sub(r'(.+?)\1+',
                                       r'\1', x) if isinstance(x,str)  else x)
```

In [188]:
```
## Example - Before -
# (Note that it would be impossible to distribute all emails accordingly).:
joel[joel['Contact']==' Carvalho'][imp2].dropna(axis=1,
                                                  thresh=1)
```

Out[188]:

	Contact	Business Phone	Home Phone	Mobile Phone	E-mail Address	E-mail 2 Address	Note
1473	Carvalho	[redacted]430	[redacted]391	NaN	NaN	NaN	[redacted]
1741	Carvalho	[redacted]172	[redacted]172	[redacted]879	NaN	NaN	[redacted]
1978	Carvalho	[redacted]999	[redacted]275	NaN	[redacted]@go[redacted]	[redacted]@remess[redacted]	[redacted]
2934	Carvalho	[redacted]426	NaN	[redacted]020	NaN	NaN	[redacted]
3815	Carvalho	[redacted]086	NaN	[redacted]479	[redacted]@[redacted]uptechnology.com	NaN	[redacted]
5233	Carvalho	[redacted]114	NaN	NaN	[redacted]@[redacted]asolutions.com	NaN	Nal
5269	Carvalho	NaN	[redacted]575	NaN	NaN	NaN	Nal

In [189]:
```
# joel Left JOIN dupes
joel.update(save, join='left', overwrite=True, filter_func=None,
                    raise_conflict=False)
```

```
In [190]: #drop the duplicates using column 'Contact' and 'E-mail' as reference but only keep
          # the most filled rows
          save_2= save.drop_duplicates(subset=['Contact'],keep='first')

          joel.drop(save.index.difference(save_2.index),inplace=True)
```

```
In [192]: ## Example - After - :
          joel[joel['Contact']==' Carvalho'][imp2].dropna(axis=1,
                                                           thresh=1)
```

Out[192]:

	Contact	Business Phone	Home Phone	Mobile Phone	E-mail Address	E-mail 2 Address	Notes
1978	Carvalho	[redacted] [redacted]999| [redacted] [redacted]172| [redacted] [redacted]426...	[redacted] [redacted]275| [redacted] [redacted]172| ||[redacted] [redacted]...	|[redacted] [redacted]879| [redacted] [redacted]020| ||[redacted] [redacted]...	[redacted]@go[redacted]||| [redacted]solutions....	[redacted]@remess[redacted] |||||	[redacted] [redacted] [redacted]...

◎ Final Number of Duplicates on 'Contact'

```
In [193]: len(joel[joel.duplicated(subset=['Contact'],keep=False)])

Out[193]: 0
```

◎ Final Number of Records

```
In [194]: len(joel)

Out[194]: 8963
```

Eliminating Created custom columns:

```
In [195]: joel.drop(['Contact','Home Phones','Business Phones', 'Other Phones',
                   'E-mail Addresses'],1,inplace=True)
```

In [196]:
```
joel.columns.values
```

Out[196]:
```
array(['Title', 'First Name', 'Middle Name', 'Last Name', 'Suffix',
       'Company', 'Department', 'Job Title', 'Business Street',
       'Business Street 2', 'Business Street 3', 'Business City',
       'Business State', 'Business Postal Code',
       'Business Country/Region', 'Home Street', 'Home Street 2',
       'Home Street 3', 'Home City', 'Home State', 'Home Postal Code',
       'Home Country/Region', 'Other Street', 'Other Street 2',
       'Other Street 3', 'Other City', 'Other State', 'Other Postal Code',
       'Other Country/Region', "Assistant's Phone", 'Business Fax',
       'Business Phone', 'Business Phone 2', 'Callback', 'Car Phone',
       'Company Main Phone', 'Home Fax', 'Home Phone', 'Home Phone 2',
       'ISDN', 'Mobile Phone', 'Other Fax', 'Other Phone', 'Pager',
       'Primary Phone', 'Radio Phone', 'TTY/TDD Phone', 'Telex',
       'Account', 'Anniversary', "Assistant's Name",
       'Billing Information', 'Birthday', 'Business Address PO Box',
       'Categories', 'Children', 'Directory Server', 'E-mail Address',
       'E-mail Type', 'E-mail Display Name', 'E-mail 2 Address',
       'E-mail 2 Type', 'E-mail 2 Display Name', 'E-mail 3 Address',
       'E-mail 3 Type', 'E-mail 3 Display Name', 'Gender',
       'Government ID Number', 'Hobby', 'Home Address PO Box', 'Initials',
       'Internet Free Busy', 'Keywords', 'Language', 'Location',
       "Manager's Name", 'Mileage', 'Notes', 'Office Location',
       'Organizational ID Number', 'Other Address PO Box', 'Priority',
       'Private', 'Profession', 'Referred By', 'Sensitivity', 'Spouse',
       'User 1', 'User 2', 'User 3', 'User 4', 'Web Page'], dtype=object)
```

Exporting

In [197]:
```
len(joel)
```

Out[197]:
```
8963
```

In [ ]:
```
writer = pd.ExcelWriter('Joel_dec212018-cleaned.xlsx', engine='xlsxwriter',
                        options={'strings_to_urls': False})
```

In [391]:
```
joel.to_excel(writer,'Sheet1')
```

In [ ]:
```
writer.save()
```