Embryo Options | Inventory Exercises - Data Analyst

Linnea Miller | 5.5.19

Load data

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

In [2]:
```
# Read in .csv files
charges = pd.read_csv('./PostCharges - Export.csv')
territories = pd.read_csv('./StateProvinceTerritories.csv')
countries = pd.read_csv('./Countries.csv')
red_rows = pd.read_csv('./red_rows.csv')
```

Overview of data

1. `charges`

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

Out[3]:

	EMR	Last Name	First Name	DOB	Partner	# Embryos	# Oocytes	Tank	Canister-Color	Donated From	...	Billing Month
0	503113.0	Last503113	First503113	11/4/72	PartnerLast503113, PartnerFirst503113	3	0.0	V	11-Blue	NaN	...	December
1	909895.0	Last909895	First909895	11/20/87	PartnerLast909895, PartnerFirst909895	3	0.0	S	7-Silver	NaN	...	December
2	93361.0	Last93361	First93361	5/13/90	PartnerLast93361, PartnerFirst93361	6	0.0	J	8-Pink	NaN	...	July

3 rows × 21 columns

In [4]:
```
# list(charges.columns)
```

In [5]:
```
# Create column called 'Full Name' --> combine First and Last name columns
charges['Full Name'] = charges[['Last Name', 'First Name']].apply(lambda x: ', '.join(x), ax
```

2. *territories*

In [6]:
```
territories.head(3)
```

Out[6]:

	ID	Country Abbreviation	State Abbreviation	State Name
0	AT1	AT	1	Burgenland
1	AT2	AT	2	Karnten
2	AT3	AT	3	Niederosterreich

3. *countries*

In [7]:
```
countries.head(3)
```

Out[7]:

	Country Abbreviation	Country Name (*This sheet should be sorted by this column for LOOKUP function to work)
0	AF	Afghanistan
1	AL	Albania
2	DZ	Algeria

In [8]:
```
# Rename 2nd column in countries to simplify
countries = countries.rename(columns={'Country Name (*This sheet should be sorted by this co
```

4. `red_rows`

In [9]:
```
1 red_rows.head(3)
```

Out[9]:

	EMR	Last Name	First Name	DOB	Partner	# Embryos	# Oocytes	Tank	Canister-Color	Donated From	...	Billing Month
0	NaN	Last915553	First212	7/12/84	PartnerLast804628, PartnerFirst748242	3	0	N	3-Green	NaN	...	September
1	NaN	Last26280	First292222	1/19/94	PartnerLast895676, PartnerFirst384420	4	0	K	7-Yellow	NaN	...	February
2	944874.0	Last944874	First944874	12/1/79	PartnerLast944874, PartnerFirst944874	4	0	S	4-Silver	NaN	...	November

3 rows × 21 columns

In [10]:
```
1 # Create column called 'Full Name' --> combine First and Last name columns
2 red_rows['Full Name'] = red_rows[['Last Name', 'First Name']].apply(lambda x: ', '.join(x),
3
```

Question 1

Question: How many unique patients are there in the dataset?

In [11]:
```
1 # Look at shape of the table to see TOTAL # of rows
2 charges.shape
```

Out[11]: (2201, 22)

In [12]:
```
1 # Using set function to find unique rows (i.e. unique patients)
2 len(set(charges['EMR']))
```

Out[12]: 2201

In [13]:
```
# Verify using 'Full Name' column
charges['Full Name'].describe()
```

Out[13]:
```
count                         2201
unique                        2201
top       Last169669, First169669
freq                             1
Name: Full Name, dtype: object
```

Answer: 2201 unique patients.

Question 2

Question: Identify all patients without an EMR number.

Answer:

In [14]:
```
charges[charges['EMR'].isnull()]
```

Out[14]:

	EMR	Last Name	First Name	DOB	Partner	# Embryos	# Oocytes	Tank	Canister-Color	Donated From	...	Comments
17	NaN	Last915553	First212	7/12/84	PartnerLast804628, PartnerFirst748242	3	0.0	N	3-Green	NaN	...	Discard
33	NaN	Last26280	First292222	1/19/94	PartnerLast895676, PartnerFirst384420	4	0.0	K	7-Yellow	NaN	...	Donate to research
55	NaN	Last260308	First982213	4/26/81	PartnerLast294652, PartnerFirst328208	3	0.0	M	20-Pink	NaN	...	Donate to research
111	NaN	Last802195	First866153	6/30/88	PartnerLast111122, PartnerFirst888945	8	0.0	N	12-Green	NaN	...	Donate to research
293	NaN	Last579476	First31068	10/27/85	PartnerLast152672, PartnerFirst647124	1	0.0	I	8-Pink	NaN	...	Discard
328	NaN	Last880108	First302075	5/4/81	PartnerLast276505, PartnerFirst184089	4	0.0	F	1-Yellow	NaN	...	Discard
352	NaN	Last356408	First886705	12/15/84	PartnerLast294027, PartnerFirst55172	1	0.0	M	19-Yellow	NaN	...	Discard

Question 3

Question: Identify all patients with an invalid Date of Birth.

In [15]:
```
# Change from string to datetime format
charges['DOB'] = pd.to_datetime(charges['DOB'])
```

In [16]:
```
charges['DOB'].sort_values().tail(45)
```

Out[16]:
```
2181   1998-05-11
330    1999-08-02
224    2016-03-20
354    2016-12-14
1631   2018-06-25
585    2019-01-21
1074   2019-03-15
2082   2055-05-11
1908   2055-05-20
2087   2058-06-17
613    2060-06-05
2021   2061-07-25
614    2061-08-06
1595   2063-04-13
1492   2063-04-18
704    2063-04-27
514    2063-12-25
1541   2064-05-03
1483   2064-08-31
1034   2064-11-05
```

Answer: When changing the datatype from string to datetime, any row with a DOB *before the year of 1969* get's incorrectly converted to future dates that aren't currently valid birthdays.

For example, `11/25/68` *incorrectly* becomes `2068-11-25`, whereas `11/20/69` *correctly* becomes `1969-11-20`.

In [17]:
```
# All invalid DOB
charges[charges['DOB'] > '2019-12-31']
```

Out[17]:

	EMR	Last Name	First Name	DOB	Partner	# Embryos	# Oocytes	Tank	Canister-Color	Donated From	...	Comments
39	472517.0	Last472517	First472517	2068-03-03	PartnerLast472517, PartnerFirst472517	17	0.0	S	6-Blue	NaN	...	NaN
310	397770.0	Last397770	First397770	2066-07-08	PartnerLast397770, PartnerFirst397770	4	0.0	S	3-Yellow	NaN	...	NaN
464	149323.0	Last149323	First149323	2066-04-04	PartnerLast149323, PartnerFirst149323	1	0.0	J	8-Yellow	NaN	...	NaN
514	267894.0	Last267894	First267894	2063-12-25	PartnerLast267894, PartnerFirst267894	8	0.0	N	3-Blue	NaN	...	NaN
613	886999.0	Last886999	First886999	2060-06-05	PartnerLast886999, PartnerFirst886999	5	0.0	E	7-Green	NaN	...	Ext. Freeze
614	859908.0	Last859908	First859908	2061-08-06	PartnerLast859908, PartnerFirst859908	2	0.0	V	2-Yellow	NaN	...	NaN
658	595532.0	Last595532	First595532	2067-10-07	PartnerLast595532, PartnerFirst595532	14	0.0	S	6-White	NaN	...	NaN

Answer: I'm also wondering if it makes sense for anyone under the age of 18 to be able to store their embryos/oocytes - I would want to verify that these birthdays are valid.

In [18]:
```
charges[(charges['DOB'] < '2019-5-5') & (charges['DOB'] > '2001-5-5')]
```

Out[18]:

	EMR	Last Name	First Name	DOB	Partner	# Embryos	# Oocytes	Tank	Canister-Color	Donated From	...	Comments
224	399824.0	Last399824	First399824	2016-03-20	PartnerLast399824, PartnerFirst399824	3	0.0	S	10-Purple	ED 2122	...	NaN
354	250054.0	Last250054	First250054	2016-12-14	PartnerLast250054, PartnerFirst250054	11	0.0	N	9-Orange	NaN	...	NaN
585	597535.0	Last597535	First597535	2019-01-21	PartnerLast597535, PartnerFirst597535	10	0.0	V	11-Yellow	NaN	...	NaN
1074	318543.0	Last318543	First318543	2019-03-15	NaN	3	0.0	M	11-Green	NaN	...	NaN
1631	684088.0	Last684088	First684088	2018-06-25	PartnerLast684088, PartnerFirst684088	9	0.0	V V	3-Green 3-Yellow	NaN	...	NaN

5 rows × 22 columns

Question 4

Q: Identify all patient addresses that do not have a valid country or state code.

1. Isolate rows where `'Address 1'` is populated.

In [19]:
```
has_address = charges[['Address 1', 'Address 2', 'City', 'State', 'Zip', 'Country']][charges
has_address.head()
```

Out[19]:

	Address 1	Address 2	City	State	Zip	Country
0	107 Main St.	NaN	West Jordan	UT	84081	USA
1	863 Main St.	NaN	Salt Lake City	UT	84106	USA
2	216 Main St.	NaN	Moroni	UT	84646	USA
3	932 Main St.	NaN	Bluffdale	UT	84065	USA
4	4 Main St.	NaN	Eagle Mountain	UT	84005	USA

2. Compare country codes in `charges` table with those in the `countries` table.

In [20]:
```
# Identify all Country codes that occur in original charges table
set(charges['Country'])
```

Out[20]:
```
{'CA', 'ESP', 'USA', nan}
```

In [21]:
```
# Set of all Country codes that exist in the has_address table
country_charges = set(has_address['Country'])
country_charges
```

Out[21]:
```
{'CA', 'USA', nan}
```

In [22]:
```
# Set of all Country codes that exist in the countries table
country_countries = set(list(countries['Country Abbreviation']))
```

In [23]:
```
# Find invalid Country codes
invalid_country = list(country_charges - country_countries)
invalid_country
```

Out[23]:
```
['USA']
```

Answer: *ALL* United States addresses in the `charges` table are **invalid** with `'USA'` instead of `'US'`. The `countries` table deems `'US'` the correct `'Country Abbreviation'` for the United States or USA.

In [24]:
```
countries[countries['Country Name'].isin(['United States', 'USA', 'United States of America
```

Out[24]:

	Country Abbreviation	Country Name
187	US	United States
188	US	USA

3. Compare state codes in `charges` table with those in the `territories` tables.

In [25]:
```
# Set of all State codes that exist in the has_address table
states_charges = set(has_address['State'])
# states_charges
```

In [26]:
```
# Set of all State codes that exist in the territories table
states_territories = set(list(territories['State Abbreviation']))
# states_territories
```

In [27]:
```
# Find invalid 'State' codes
invalid_states = list(states_charges - states_territories)
invalid_states
```

Out[27]:
```
['XY', 'UX', 'CD', 'VW', 'AW', 'AE']
```

Answer: The resulting table shows the rows in `charges` where the `State` code is **invalid**.

In [28]:
```
has_address[has_address['State'].isin(invalid_states)]
```

Out[28]:

	Address 1	Address 2	City	State	Zip	Country
210	915 Main St.	NaN	Cedar City	UX	84721	USA
401	758 Main St.	NaN	Logan	CD	84321	USA
610	177 Main St.	NaN	Perry	AW	84302	USA
1483	467 Main St.	NaN	Fpo	AE	9566	USA
1891	664 Main St.	NaN	Cedar City	VW	84720	USA
2198	601 Main St.	NaN	Vernal	XY	84078	USA

In [29]:
```
# # Get list of index values for all invalid State code rows
# invalid_1 = list(has_address[has_address['State'].isin(invalid_states)].index)
```

4. Check nulls

Answer: The resulting table shows the rows in `charges` where the `'State'` code OR `'Country'` code is **NULL**.

In [30]:
```
# Find all rows where State code OR Country code is NULL
has_address[(has_address['State'].isnull()) | (has_address['Country'].isnull())]
```

Out[30]:

	Address 1	Address 2	City	State	Zip	Country
218	173 Main St.	NaN	Alberta	NaN	NaN	CA
1720	731 Main St.	Seuzach Switzerland	NaN	NaN	NaN	NaN
1758	234 Main St.	NaN	NaN	NaN	NaN	NaN
1812	992 Main St.	NaN	Vancouver	BC	CA	NaN
2175	127 Main St.	NaN	NaN	NaN	NaN	NaN

In [31]:
```
# Get list of index values for all null rows
invalid_2 = list(has_address[(has_address['State'].isnull()) | (has_address['Country'].isnul
```

5. Final overview

All invalid rows were in one of 2 groups:

1. Invalid `'Country'` code - either **null** or **USA** which is NOT the correct country code according to the `countries` table.
2. Invalid `'State'` code - either **null** or an **incorrect code** : `['XY', 'VW', 'CD', 'AE', 'UX', 'AW']`.

6. Other observations/EDA

- Notice `has_address.iloc[1720]` which has city/state/country information in the `Address 2` column instead of in the *appropriate* columns. Based on the `countries` table, the country code for Switzerland would be `CH`.

In [32]:
```
# Verify Switzerland is valid country name & find out what the country abbreviation is for
countries[countries['Country Name'] == 'Switzerland']
```

Out[32]:

	Country Abbreviation	Country Name
169	CH	Switzerland

- `has_address.iloc[1812]` has an error. The country code appears in the `'Zip'` column instead of the `'Country'` column. If `'CA'` was in the `'Country'` column, it *would* be a valid country code!!

In [33]:
```
# Verify CA is valid country abbreviation
countries[countries['Country Abbreviation'] == 'CA']
```

Out[33]:

	Country Abbreviation	Country Name
31	CA	Canada

The rows in `charges` where `Country` == `'ESP'` have an invalid country code. The official country abbreviation is `'ES'`, not `'ESP'`.

In [34]:
```
# Look up Country Abbreviation values in countries table
countries[countries['Country Abbreviation'].isin(['ES', 'ESP', 'EP', 'SP'])]
```

Out[34]:

	Country Abbreviation	Country Name
163	ES	Spain

Question 5

Question: Identify all records where the 'Billing Month' does not match the month of the 'Date In'.

Approach: Create two new columns that contain the numeric value of the months from both original columns ('*Billing Month*' and '*Date In*'). Do this by converting each column to datetime in some form and pulling the **month** information. Finally, find the rows where the values in the two newly created columns *don't* match.

1. Nulls

In [35]:
```
# Check for null values
charges[['Date In', 'Billing Month']].isnull().sum()
```

Out[35]:
```
Date In           2
Billing Month    27
dtype: int64
```

In [36]:
```
# Replace all null cells with a string ('?') and leave all other cells as they are
charges['Billing Month'] = charges['Billing Month'].map(lambda x: '?' if str(x) == 'nan' els
charges['Date In'] = charges['Date In'].map(lambda x: '?' if str(x) == 'nan' else x)
```

2. Clean up `'Date In'` columns

In [37]:
```
import re
```

In [38]:
```
# Clean up 'Date In' column
for i in range(len(charges)):

    c = charges['Date In'][i]
    charges['Date In'][i] = re.sub('\s+', ' ', c).strip().split(' ')

    if i % 500 == 0:
        print(i)
```

In [39]:
```
1 charges['Date In'].tail(20)
```

Out[39]:
```
2181              [12/8/18]
2182    [5/12/18, 11/12/18]
2183              [4/18/18]
2184              [2/26/19]
2185             [10/27/18]
2186               [2/9/19]
2187              [8/15/18]
2188              [4/20/18]
2189              [5/29/17]
2190              [8/29/18]
2191              [5/31/17]
2192              [1/25/18]
2193      [4/7/16, 7/27/18]
2194               [3/9/18]
2195              [1/17/14]
2196             [10/28/18]
2197             [11/16/18]
2198              [12/1/18]
2199              [2/17/18]
2200              [9/30/18]
Name: Date In, dtype: object
```

In [40]:
```
1 # Create empty DataFrame (date_in_fix) with identical index to charges DataFrame
2 date_in_fix = pd.DataFrame(columns = ['Date In - 1', 'Date In - 2', 'Date In - 3', 'Date In
3
```

In [41]:

```
# Populate empty DataFrame with 'Date In' dates
for i in range(len(charges)):
    if len(charges['Date In'][i]) == 1:
        date_in_fix['Date In - 1'][i] = str(charges['Date In'][i][0])

    elif len(charges['Date In'][i]) == 2:
        date_in_fix['Date In - 1'][i] = str(charges['Date In'][i][0])
        date_in_fix['Date In - 2'][i] = str(charges['Date In'][i][1])

    elif len(charges['Date In'][i]) == 3:
        date_in_fix['Date In - 1'][i] = str(charges['Date In'][i][0])
        date_in_fix['Date In - 2'][i] = str(charges['Date In'][i][1])
        date_in_fix['Date In - 3'][i] = str(charges['Date In'][i][2])

    elif len(charges['Date In'][i]) == 4:
        date_in_fix['Date In - 1'][i] = str(charges['Date In'][i][0])
        date_in_fix['Date In - 2'][i] = str(charges['Date In'][i][1])
        date_in_fix['Date In - 3'][i] = str(charges['Date In'][i][2])
        date_in_fix['Date In - 4'][i] = str(charges['Date In'][i][3])

    elif len(charges['Date In'][i]) >= 5:
        date_in_fix['Date In - 1'][i] = str(charges['Date In'][i][0])
        date_in_fix['Date In - 2'][i] = str(charges['Date In'][i][1])
        date_in_fix['Date In - 3'][i] = str(charges['Date In'][i][2])
        date_in_fix['Date In - 4'][i] = str(charges['Date In'][i][3])
        date_in_fix['Date In - 5'][i] = str(charges['Date In'][i][4])

    else:
        pass
```

In [42]:

```
# Check for nulls
date_in_fix[date_in_fix['Date In - 1'].isnull()]
```

Out[42]:

Date In - 1	Date In - 2	Date In - 3	Date In - 4	Date In - 5

In [43]:

```
# Concatenate charges and date_in_fix
charges = pd.concat([charges, date_in_fix], axis=1)
```

In [44]:
```
1 charges[['Date In - 1', 'Date In - 2', 'Date In - 3', 'Date In - 4', 'Date In - 5']].tail(2(
```

Out[44]:

	Date In - 1	Date In - 2	Date In - 3	Date In - 4	Date In - 5
2181	12/8/18	NaN	NaN	NaN	NaN
2182	5/12/18	11/12/18	NaN	NaN	NaN
2183	4/18/18	NaN	NaN	NaN	NaN
2184	2/26/19	NaN	NaN	NaN	NaN
2185	10/27/18	NaN	NaN	NaN	NaN
2186	2/9/19	NaN	NaN	NaN	NaN
2187	8/15/18	NaN	NaN	NaN	NaN
2188	4/20/18	NaN	NaN	NaN	NaN
2189	5/29/17	NaN	NaN	NaN	NaN
2190	8/29/18	NaN	NaN	NaN	NaN
2191	5/31/17	NaN	NaN	NaN	NaN
2192	1/25/18	NaN	NaN	NaN	NaN
2193	4/7/16	7/27/18	NaN	NaN	NaN
2194	3/9/18	NaN	NaN	NaN	NaN
2195	1/17/14	NaN	NaN	NaN	NaN
2196	10/28/18	NaN	NaN	NaN	NaN
2197	11/16/18	NaN	NaN	NaN	NaN
2198	12/1/18	NaN	NaN	NaN	NaN
2199	2/17/18	NaN	NaN	NaN	NaN
2200	9/30/18	NaN	NaN	NaN	NaN

3. Get month numbers

In [45]:
```
from datetime import datetime
```

In [46]:
```
# Create function that will yield the corresponding number to the 'Billing Month'
def month_bill(x):
    d = datetime.strptime(x, '%B')
    if x != '?':
        return int(d.strftime('%m'))
    else:
        return x
```

In [47]:
```
# Create new column with 'Billing Month' converted into corresponding number (int) if doesn
# Using month_bill function from above
charges['Bill_Month_num'] = charges['Billing Month'].map(lambda x: month_bill(x) if x != '?
```

In [48]:
```
# Create function that will yield the corresponding number to the 'Date In'
def month_datein(x):
    # Using try/except here in case there are any weird strings that don't fit the format I
    try:
        d = datetime.strptime(x, '%m/%d/%y')
        if x != '?':
            return int(d.strftime('%m'))
    except:
        d = datetime.strptime(x, '%m/%d/%Y')
        if x!= '?':
            return int(d.strftime('%m'))
```

In [49]:
```
# Create new column with 'Date In' converted into corresponding number (int) if doesn't equ
# Using month2 function from above
charges['Date_In_num'] = charges['Date In - 1'].map(lambda x: month_datein(x) if x != '?' el
```

In [50]:
```
charges[['Billing Month', 'Bill_Month_num', 'Date In - 1', 'Date_In_num']].head()
```

Out[50]:

	Billing Month	Bill_Month_num	Date In - 1	Date_In_num
0	December	12	12/22/18	12
1	December	12	12/5/17	12
2	July	7	7/30/15	7
3	September	9	9/21/14	9
4	April	4	4/26/18	4

In [51]:
```
# Find the rows where the month values don't match
non_match = charges[['Billing Month', 'Bill_Month_num', 'Date In - 1', 'Date In - 2', 'Date_
```

In [52]:
```
# Find how many rows out of the whole dataset have mismatching months
len(non_match)
```

Out[52]: 39

In [53]:
```
# Show records where 'Billing Month' does not match the month of the 'Date In'
non_match.head(10)
# non_match
```

Out[53]:

	Billing Month	Bill_Month_num	Date In - 1	Date In - 2	Date_In_num
26	?	?	12/26/16	NaN	12
180	November	11	2/20/17	NaN	2
224	May	5	8/4/16	NaN	8
272	?	?	9/28/15	NaN	9
399	October	10	2/5/17	10/16/17	2
468	?	?	10/30/13	NaN	10
470	?	?	7/28/17	NaN	7
512	?	?	7/27/16	NaN	7
565	March	3	2/19/16	3/18/16	2
597	?	?	9/22/15	NaN	9

Answer: After cleaning the data to an extent, I've found two patterns concerning the rows containing months that "don't match":

1. `'Bill_Month_num'` = `'?'` when `'Date_In_num'` is some integer

Example: `charges.iloc[26]`

In [54]:
```
1 charges[['Bill_Month_num', 'Date_In_num']].iloc[26]
```

Out[54]:
```
Bill_Month_num     ?
Date_In_num       12
Name: 26, dtype: object
```

In [55]:
```
1 non_match[non_match['Billing Month'] == '?']
```

Out[55]:

	Billing Month	Bill_Month_num	Date In - 1	Date In - 2	Date_In_num
26	?	?	12/26/16	NaN	12
272	?	?	9/28/15	NaN	9
468	?	?	10/30/13	NaN	10
470	?	?	7/28/17	NaN	7
512	?	?	7/27/16	NaN	7
597	?	?	9/22/15	NaN	9
718	?	?	5/15/18	NaN	5
935	?	?	12/10/18	NaN	12
1336	?	?	11/23/14	NaN	11
1337	?	?	3/13/16	NaN	3
1341	?	?	9/28/16	NaN	9

2. `'Bill_Month_num'` and `'Date_In_num'` are both integers but don't match. *Example:* `charges.iloc[399]`

In [56]:
```
1 charges[['Bill_Month_num', 'Date_In_num']].iloc[399]
```

Out[56]:
```
Bill_Month_num    10
Date_In_num        2
Name: 399, dtype: object
```

The rows where the `'Billing Month'` does NOT match the `'Date In'` month are those in which the the `'Billing Month'` either isn't reflecting the month of the **most recent** `'Date In'` or is just straight-up incorrect!

It would obviously take additional research to figure out why some of the `'Bill_Month_num'` rows contained nulls (i.e. '?').

In [57]:
```
1 non_match[non_match['Billing Month'] != '?']
```

Out[57]:

	Billing Month	Bill_Month_num	Date In - 1	Date In - 2	Date_In_num
180	November	11	2/20/17	NaN	2
224	May	5	8/4/16	NaN	8
399	October	10	2/5/17	10/16/17	2
565	March	3	2/19/16	3/18/16	2
860	May	5	6/2/17	NaN	6
1115	October	10	2/17/18	10/29/18	2
1189	November	11	7/14/17	NaN	7
1761	November	11	?	NaN	?
1926	December	12	11/9/15	12/8/16	11
2053	December	12	2/23/14	12/20/16	2
2058	July	7	1/19/15	7/16/15	1
2165	February	2	12/16/14	2/26/14	12
2176	June	6	4/2/15	5/13/15	4

Question 6

Question: Create a combined list of all non-zero embryo and egg records (so we can track embryo and egg records separately).

1. Clean `'# Embryos'` column

In [58]:
```
# Check datatypes
charges[['# Embryos', '# Oocytes']].dtypes
```

Out[58]:
```
# Embryos     object
# Oocytes    float64
dtype: object
```

In [59]:
```
# Find any strings that aren't pure numbers
bad_string = []
for i in range(len(charges)):
    try:
        if len(charges['# Embryos'][i]) > 2:
            bad_string.append(i)
        else:
            pass
    except:
        pass
```

In [60]:
```
bad_string
```

Out[60]:
```
[2053]
```

In [61]:
```
charges[['# Embryos']].iloc[2053]
```

Out[61]:
```
# Embryos    15 (minus 5)
Name: 2053, dtype: object
```

Since this particular row has non-numeric characters, I'll change the value to `'10'` since it says 15 (minus 5). In a real-life scenario, I would double check with the specific clinic to see what the actual value should be if not 10.

In [62]:
```
charges['# Embryos'].iloc[2053] = '10'
```

```
/anaconda3/lib/python3.6/site-packages/pandas/core/indexing.py:189: 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-docs/stable/indexing.html#indexing-view-versus-copy
  self._setitem_with_indexer(indexer, value)
```

In [63]:
```
charges[['# Embryos']].iloc[2053]
```

Out[63]:
```
# Embryos    10
Name: 2053, dtype: object
```

2. Convert `'# Embryos'` column to dtype = float

In [64]:
```
charges['# Embryos'] = charges['# Embryos'].map(lambda x: float(x))
```

In [65]:
```
charges[['# Embryos', '# Oocytes']].dtypes
```

Out[65]:
```
# Embryos    float64
# Oocytes    float64
dtype: object
```

3. Create list of non-zero embryo & egg records

In [66]:
```
# # Find all rows where '# Embryos' OR '# Oocytes' are NOT 0
# charges[(charges['# Embryos'] != 0) | (charges['# Oocytes'] != 0)]
```

In [67]:
```
# Create separate lists of row ids to reflect the possibility of there being two differeny i
embryos = list(charges[charges['# Embryos'] != 0].index)
oocytes = list(charges[charges['# Oocytes'] != 0].index)
```

In [68]:
```
# Gather embryo records into DataFrame
em = charges[['EMR', 'Full Name', '# Embryos', 'Date In - 1', 'Date_In_num']].loc[embryos]
em['Type'] = 'embryo'
em.rename(columns={'# Embryos':'#'}, inplace=True)
em.reset_index(inplace=True, drop=True)
```

In [69]:
```
# Gather oocyte records into DataFrame
oo = charges[['EMR', 'Full Name', '# Oocytes', 'Date In - 1', 'Date_In_num']].loc[oocytes]
oo['Type'] = 'oocyte'
oo.rename(columns={'# Oocytes':'#'}, inplace=True)
oo.reset_index(inplace=True, drop=True)
```

In [70]:
```
# Concat the two tables
non_zero_records = pd.concat([em, oo])
non_zero_records.reset_index(inplace=True, drop=True)
```

In [71]:
```
# Reformat table
all_tissue = non_zero_records[['EMR', 'Full Name', 'Type', '#', 'Date In - 1', 'Date_In_num
```

Answer:

In [72]:
```
all_tissue.head(10)
```

Out[72]:

	EMR	Full Name	Type	#	Date In - 1	Date_In_num
0	503113.0	Last503113, First503113	embryo	3.0	12/22/18	12
1	909895.0	Last909895, First909895	embryo	3.0	12/5/17	12
2	93361.0	Last93361, First93361	embryo	6.0	7/30/15	7
3	937394.0	Last937394, First937394	embryo	4.0	9/21/14	9
4	216833.0	Last216833, First216833	embryo	5.0	4/26/18	4
5	142608.0	Last142608, First142608	embryo	13.0	6/25/17	6
6	130988.0	Last130988, First130988	embryo	6.0	7/29/18	7
7	195936.0	Last195936, First195936	embryo	5.0	8/18/15	8
8	223515.0	Last223515, First223515	embryo	2.0	11/7/18	11
9	362926.0	Last362926, First362926	embryo	5.0	4/11/18	4

In [73]:
```
# Example of two separate records for the same EMR
all_tissue[all_tissue['EMR'] == 668790.0]
```

Out[73]:

	EMR	Full Name	Type	#	Date In - 1	Date_In_num
1247	668790.0	Last668790, First668790	embryo	2.0	3/15/16	3
2152	668790.0	Last668790, First668790	oocyte	17.0	3/15/16	3

Question 7

Question: Given the list from question 6, within each specimen type, the clinic has decided to charge each patient only for their oldest record (based on the 'Date In'). In addition, some records have been highlighted in red and should not be billed.

Identify all records as billable or non-billable.

1. Check for duplicates and nulls

In [74]:
```
# # Check for null Last Name in red_rows --> no nulls
# red_rows[red_rows['Full Name'].isnull()]
```

In [75]:
```
# # Check for duplicate Last Name in red_rows --> no dups
# red_rows[red_rows['Full Name'].duplicated() == True]
```

In [76]:
```
# # Check for null Last Name in all_tissue --> no nulls
# all_tissue[all_tissue['Full Name'].isnull()]
```

In [77]:
```
# # Check for duplicate Last Name in all_tissue
# all_tissue[all_tissue['Full Name'].duplicated() == True]
```

In [78]:
```
full_name_dups = list(all_tissue[all_tissue['Full Name'].duplicated() == True]['Full Name'])

```

2. Identify/label non-billable rows

In [79]:
```
all_tissue['Billable'] = None
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

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

In [80]:
```
red = list(red_rows['Full Name'])
```

(The following code takes a minute to run)

In [81]:
```
# If corresponding to a red row, NOT billable --> 0
# If not corresponding to a red row, billable --> 1
for i in range(len(all_tissue)):
    if all_tissue['Full Name'][i] in red:
        all_tissue['Billable'].iloc[i] = 0 # i.e. not-billable
    else:
        all_tissue['Billable'].iloc[i] = 1 # i.e. billable
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:7: 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-docs/stable/indexing.html#indexing-view-versus-copy
  import sys
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:5: 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-docs/stable/indexing.html#indexing-view-versus-copy
  """
```

In [82]:
```
# Verify new 'Billable' column
all_tissue[all_tissue['Billable'] == 0].head()
```

Out[82]:

	EMR	Full Name	Type	#	Date In - 1	Date_In_num	Billable
17	NaN	Last915553, First212	embryo	3.0	9/29/16	9	0
31	NaN	Last26280, First292222	embryo	4.0	2/19/17	2	0
37	944874.0	Last944874, First944874	embryo	4.0	11/11/17	11	0
52	NaN	Last260308, First982213	embryo	3.0	4/24/17	4	0
106	NaN	Last802195, First866153	embryo	8.0	2/11/17	2	0

In [83]:
```
all_tissue.tail(10)
# all_tissue
```

Out[83]:

	EMR	Full Name	Type	#	Date In - 1	Date_In_num	Billable
2177	298586.0	Last298586, First298586	oocyte	32.0	8/25/15	8	1
2178	884447.0	Last884447, First884447	oocyte	52.0	2/7/18	2	1
2179	NaN	Last309176, First872381	oocyte	12.0	10/25/16	10	0
2180	355378.0	Last355378, First355378	oocyte	12.0	1/22/17	1	1
2181	207303.0	Last207303, First207303	oocyte	17.0	5/28/16	5	1
2182	559696.0	Last559696, First559696	oocyte	12.0	2/28/19	2	1
2183	171526.0	Last171526, First171526	oocyte	14.0	2/26/18	2	1
2184	376055.0	Last376055, First376055	oocyte	22.0	11/12/16	11	1
2185	275122.0	Last275122, First275122	oocyte	10.0	12/20/16	12	1
2186	15687.0	Last15687, First15687	oocyte	26.0	1/17/14	1	1

Question 8

Question: Given the list of billable from question 7, calculate the number of records that will be billed in each month (assuming records are billed once a year on their `'Date In'` anniversary.

In [84]:
```
import matplotlib, matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

In [85]:
```
billable = all_tissue[all_tissue['Billable'] == 1]
```

In [86]:
```
def monthly_bill_count(month_num):
    return len(billable[billable['Date_In_num'] == month_num])

```

In [87]:
```
x = ['JAN',
     'FEB',
     'MAR',
     'APR',
     'MAY',
     'JUNE',
     'JULY',
     'AUG',
     'SEP',
     'OCT',
     'NOV',
     'DEC',
     'Unknown']

y = [monthly_bill_count(1),
    monthly_bill_count(2),
    monthly_bill_count(3),
    monthly_bill_count(4),
    monthly_bill_count(5),
    monthly_bill_count(6),
    monthly_bill_count(7),
    monthly_bill_count(8),
    monthly_bill_count(9),
    monthly_bill_count(10),
    monthly_bill_count(11),
    monthly_bill_count(12),
    monthly_bill_count('?')]

```

In [88]:
```
def show_counts():
    for i in range(13):
        print(x[i], ': ', y[i])
```

Answer:

In [89]:
```
show_counts()
```

```
JAN :  195
FEB :  225
MAR :  208
APR :  141
MAY :  141
JUNE :  136
JULY :  153
AUG :  180
SEP :  184
OCT :  152
NOV :  201
DEC :  164
Unknown :  3
```

In [90]:

```
plt.figure(figsize=(10, 7))
plt.title('# of Billable Records per Month')
matplotlib.rc('font', size=11)
matplotlib.rc('axes', titlesize=14)
sns.barplot(x, y).set_xticklabels(x, rotation=40)

plt.show();
```