# Surgical Complications

Group Members: 1) Theresa Aguilar 2) Jeff Thornhill 3) JoAnn Vuong

### Import the necessary libraries

```
In [1]: import csv
        import pandas as pd
        import numpy as np
        import datetime
        import matplotlib.pyplot as plt
        import warnings
        warnings.filterwarnings('ignore')
```

### Load the files into dataframes

```
In [2]: #Inpatient

        inpatient_raw = pd.read_csv('Data_Files/inpatient_master.csv', low_memory=False)

        inpatient_raw.head(5)
```

Out[2]:

| | DESYNPUF_ID | CLM_ID | SEGMENT | CLM_FROM_DT | CLM_THRU_DT | PRVDR_NUM |
|---|---|---|---|---|---|---|
| **0** | 00013D2EFD8E45D1 | 196661176988405 | 1 | 20100312.0 | 20100313.0 | 2600GD |
| **1** | 00016F745862898F | 196201177000368 | 1 | 20090412.0 | 20090418.0 | 3900MB |
| **2** | 00016F745862898F | 196661177015632 | 1 | 20090831.0 | 20090902.0 | 3900HM |
| **3** | 00016F745862898F | 196091176981058 | 1 | 20090917.0 | 20090920.0 | 3913XU |
| **4** | 00016F745862898F | 196261176983265 | 1 | 20100626.0 | 20100701.0 | 3900MB |

5 rows × 81 columns

In [3]:
```
# Beneficiary

ben_2008_raw = pd.read_csv('Data_Files/master_2008_csv.csv', low_memory=False)
ben_2009_raw = pd.read_csv('Data_Files/master_2009_csv.csv', low_memory=False)
ben_2010_raw = pd.read_csv('Data_Files/master_2010_csv.csv', low_memory=False)

ben_raw = pd.concat([ben_2008_raw,ben_2009_raw,ben_2010_raw])

ben_raw.head(5)
```

Out[3]:
```
         DESYNPUF_ID  BENE_BIRTH_DT  BENE_DEATH_DT  BENE_SEX_IDENT_CD  BENE_RACE_C
0   0000141F2FECE9BC       19740401            NaN                  2
1   0000B27E77EE1987       19550201            NaN                  1
2   0000C1386AE2C2A2       19591101            NaN                  2
3   0000EC65FBF94AB8       19370401            NaN                  1
4   00020C9F73FD7F45       19430601            NaN                  2

5 rows × 32 columns
```

In [4]:
```
#clean up unecessary variables from the script to create more memory
del ben_2008_raw
del ben_2009_raw
del ben_2010_raw
```

## Data Profiling

In [5]:
```
# Data Profile analysis for Inpatient.
ip_dataprofile_results_df = pd.DataFrame([], columns = ["column_name", "col_count
li = inpatient_raw.columns
row_num = inpatient_raw.shape[0]

for val in li:
  if inpatient_raw[val].count() == 0:
    column_name = val
    col_count = inpatient_raw[val].count()
    unique_count = len(pd.unique(inpatient_raw[val]))
    column_mode = inpatient_raw[val].mode()
    null_count = inpatient_raw[val].isnull().sum()
    null_pct = (null_count / null_count)*100
  else:
    column_name = val
    col_count = inpatient_raw[val].count()
    unique_count = len(pd.unique(inpatient_raw[val]))
    column_mode = inpatient_raw[val].mode()
    null_count = inpatient_raw[val].isnull().sum()
    null_pct = (null_count / row_num)*100          # Needs to be tweaked.
  stage = [column_name,col_count,unique_count,column_mode,null_count,null_pct]
  a_series = pd.Series(stage, index=ip_dataprofile_results_df.columns)
  ip_dataprofile_results_df = ip_dataprofile_results_df.append(a_series, ignore_i
```

In [6]:
```
# Data Profile results inpatient.
pd.set_option("display.max_rows", None)

ip_dataprofile_results_df
```

Out[6]:

| | column_name | col_count | unique_count | column_mode | null_c |
|---|---|---|---|---|---|
| **0** | DESYNPUF_ID | 4008836 | 755214 | 0 91948E46E2DF89A1 dtype: object | |
| **1** | CLM_ID | 4008836 | 1331533 | 0 90011100088330 1 90031100088821 ... | |
| **2** | SEGMENT | 4008836 | 2 | 0 1 dtype: int64 | |
| **3** | CLM_FROM_DT | 4004965 | 1132 | 0 20080704.0 dtype: float64 | 3 |
| **4** | CLM_THRU_DT | 4004965 | 1097 | 0 20080709.0 dtype: float64 | 3 |
| **5** | PRVDR_NUM | 4008836 | 2895 | 0 23006G dtype: object | |
| **6** | CLM_PMT_AMT | 4008836 | 100 | 0 4000.0 dtype: float64 | |

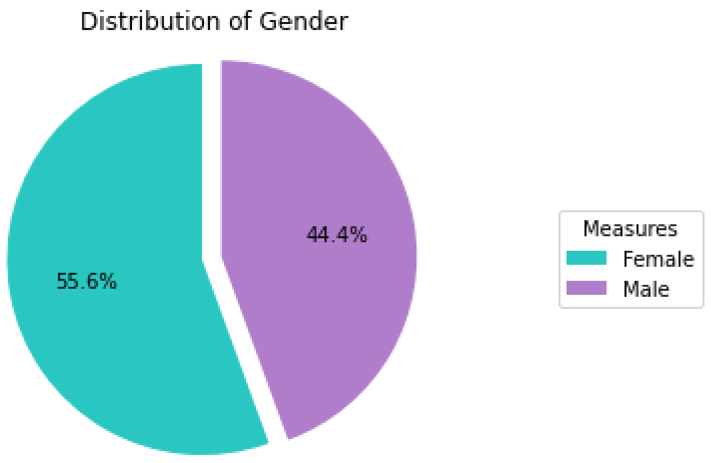
In [7]:
```
# Data Profile analysis for Beneficiary.
bene_dataprofile_results_df = pd.DataFrame([], columns = ["column_name", "col_cou
li = ben_raw.columns
row_num = ben_raw.shape[0]

for val in li:
  if ben_raw[val].count() == 0:
    column_name = val
    col_count = ben_raw[val].count()
    unique_count = len(pd.unique(ben_raw[val]))
    column_mode = ben_raw[val].mode()
    null_count = ben_raw[val].isnull().sum()
    null_pct = (null_count / null_count)*100
  else:
    column_name = val
    col_count = ben_raw[val].count()
    unique_count = len(pd.unique(ben_raw[val]))
    column_mode = ben_raw[val].mode()
    null_count = ben_raw[val].isnull().sum()
    null_pct = (null_count / row_num)*100         # Needs to be tweaked.
  stage = [column_name,col_count,unique_count,column_mode,null_count,null_pct]
  a_series = pd.Series(stage, index=bene_dataprofile_results_df.columns)
  bene_dataprofile_results_df = bene_dataprofile_results_df.append(a_series, ignc
```

In [8]:
```
# Data Profile results beneficiary.
bene_dataprofile_results_df.head(5)
```

Out[8]:

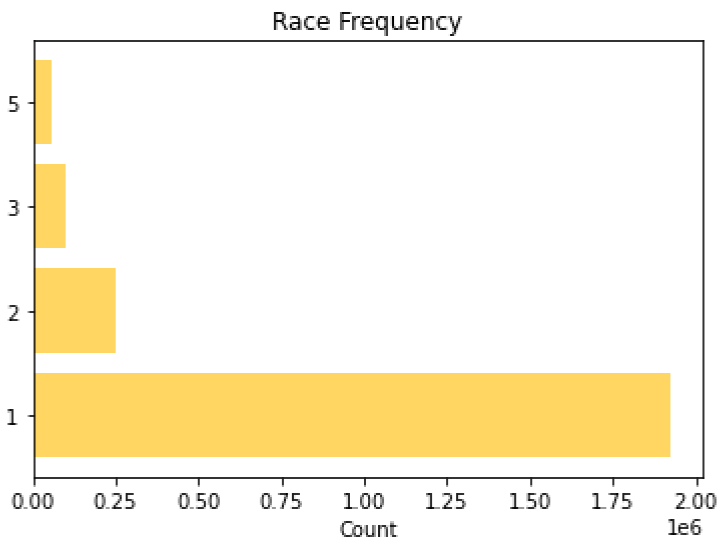
| | column_name | col_count | unique_count | column_mode | null_count | null_pct |
|---|---|---|---|---|---|---|
| **0** | DESYNPUF_ID | 15573396 | 2326856 | 0 0000438E79D01BEA 1 00015BF6509... | 0 | 0.000000 |
| **1** | BENE_BIRTH_DT | 15573396 | 900 | 0 19431001 dtype: int64 | 0 | 0.000000 |
| **2** | BENE_DEATH_DT | 242991 | 37 | 0 20080901.0 dtype: float64 | 15330405 | 98.439704 |
| **3** | BENE_SEX_IDENT_CD | 15573396 | 2 | 0 2 dtype: int64 | 0 | 0.000000 |
| **4** | BENE_RACE_CD | 15573396 | 4 | 0 1 dtype: int64 | 0 | 0.000000 |

In [9]:
```
## ADD GRAPHS -

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = ['Female','Male']
sizes = [1292861, 1033995]
explode = (0.1, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')
colors = ['#2AC7C2','#B07DCB','#EB5A5A','#FFD661','#ABC7FC']
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, colors=colors, autopct='%1.1f%%',
        shadow=False, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
ax1.legend(labels,
          title="Measures",
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1))
plt.title('Distribution of Gender')
plt.show()
```



In [10]:
```
plt.barh(range(0,4), [1926708, 247723, 97972, 54453],
         color='#FFD661',
         tick_label=['1','2','3','5'])
plt.title("Race Frequency")
plt.xlabel("Count")
```

Out[10]:
```
Text(0.5, 0, 'Count')
```



**After reviewing the data profiling results, the columns that we found to be valuable and worth keeping are kept while the others are removed.**

In [11]:
```
# Inpatient dataset after reviewing the data profiling results.
narrow_inpatient_raw = inpatient_raw[["DESYNPUF_ID","CLM_ID","SEGMENT","CLM_FROM_
"NCH_BENE_IP_DDCTBL_AMT","NCH_BENE_DSCHRG_DT","ICD9_DGNS_CD_1","ICD9_DGNS_CD_2",'
"ICD9_PRCDR_CD_2","ICD9_PRCDR_CD_3","ICD9_PRCDR_CD_4","ICD9_PRCDR_CD_5"]]

narrow_inpatient_raw.head(5)
```

Out[11]:

| | DESYNPUF_ID | CLM_ID | SEGMENT | CLM_FROM_DT | CLM_THRU_DT | CLM_PMT_AI |
|---|---|---|---|---|---|---|
| **0** | 00013D2EFD8E45D1 | 196661176988405 | 1 | 20100312.0 | 20100313.0 | 400( |
| **1** | 00016F745862898F | 196201177000368 | 1 | 20090412.0 | 20090418.0 | 2600( |
| **2** | 00016F745862898F | 196661177015632 | 1 | 20090831.0 | 20090902.0 | 500( |
| **3** | 00016F745862898F | 196091176981058 | 1 | 20090917.0 | 20090920.0 | 500( |
| **4** | 00016F745862898F | 196261176983265 | 1 | 20100626.0 | 20100701.0 | 1600( |

5 rows × 23 columns

In [12]:
```
# Beneficiary dataset after reviewing the data profiling results.

ben_master = ben_raw[["DESYNPUF_ID","BENE_BIRTH_DT","BENE_DEATH_DT","BENE_SEX_IDE

ben_master.head(5)
```

Out[12]:

| | DESYNPUF_ID | BENE_BIRTH_DT | BENE_DEATH_DT | BENE_SEX_IDENT_CD | BENE_RACE_CI |
|---|---|---|---|---|---|
| **0** | 0000141F2FECE9BC | 19740401 | NaN | 2 | |
| **1** | 0000B27E77EE1987 | 19550201 | NaN | 1 | |
| **2** | 0000C1386AE2C2A2 | 19591101 | NaN | 2 | |
| **3** | 0000EC65FBF94AB8 | 19370401 | NaN | 1 | |
| **4** | 00020C9F73FD7F45 | 19430601 | NaN | 2 | |

### Format the data

In [13]:
```
#Inpatient

narrow_inpatient_raw['NCH_BENE_DSCHRG_DT'] = pd.to_datetime(narrow_inpatient_raw[
narrow_inpatient_raw['CLM_FROM_DT'] = pd.to_datetime(narrow_inpatient_raw['CLM_FI
narrow_inpatient_raw['CLM_THRU_DT'] = pd.to_datetime(narrow_inpatient_raw['CLM_TI
narrow_inpatient_raw['CLM_ADMSN_DT'] = pd.to_datetime(narrow_inpatient_raw['CLM_A
```

In [14]:
```
# Beneficiary

ben_master['BENE_BIRTH_DT'] = pd.to_datetime(ben_master['BENE_BIRTH_DT'].astype(
ben_master['BENE_DEATH_DT'] = pd.to_datetime(ben_master['BENE_DEATH_DT'].astype(

ben_master.rename(columns = {'BENE_BIRTH_DT':'BIRTH_DT','BENE_DEATH_DT':'DEATH_DT
```

### Datasets

In [15]:
```
# Create a dataframe of unique DESYNPUF_IDs from beneficiary dataset.
ben_master_sorted = ben_master.sort_values(by=['DEATH_DT','DESYNPUF_ID'], ascendi
unique_ben = ben_master_sorted.drop_duplicates(subset = ["DESYNPUF_ID"])

# Create a dataframe of only the DESYNPUF_IDs who have a DEATH_DT.
dead_ben = unique_ben[unique_ben.DEATH_DT.notnull()]
```

### Create the general and vascular surgery procedure list for filtering.

In [16]:
```
# Load the procedure code list provided by Ram.
proc_reference = pd.read_csv('Data_Files/surgery_flags_i9_2015.csv', low_memory=F
proc_reference = proc_reference.rename(columns = {"'ICD-9-CM CODE'":'ICD9_PROCEDU

proc_reference.head(5)
```

Out[16]:

| | ICD9_PROCEDURE_CD | SURGERY_FLAG | ICD9_DX_DESCRIPTION |
|---|---|---|---|
| **0** | '0050' | '2' | IMPLA RESYNCHR PACEMAKER W/0 (Begin 2002) |
| **1** | '0051' | '2' | IMPLA RESYNCHRONIZATION DEFI (Begin 2002) |
| **2** | '0052' | '2' | IMPL/REPL TRANSVENOUS LEAD L (Begin 2002) |
| **3** | '0053' | '2' | IMPL/REPL PACEMAKER PLSE GE (Begin 2002) |
| **4** | '0054' | '2' | IMPL/REPL DEFIBRIL GENERATOR (Begin 2002) |

In [17]:
```
# Clean up procedure code list.

# Remove all quotes around the surgery flag number.
proc_reference['SURGERY_FLAG'] = proc_reference['SURGERY_FLAG'].str.replace(r"[\'
proc_reference['ICD9_PROCEDURE_CD'] = proc_reference['ICD9_PROCEDURE_CD'].str.rep


proc_reference2 = proc_reference[proc_reference['SURGERY_FLAG'] == '2']

# Creating the selected surgeries and turning them into a list.
proc_list = proc_reference2['ICD9_PROCEDURE_CD'].to_list()
rams_procedure_codes = [float(i) for i in proc_list] #converting string to float

proc_reference2.head(5)
```

Out[17]:

| | ICD9_PROCEDURE_CD | SURGERY_FLAG | ICD9_DX_DESCRIPTION |
|---|---|---|---|
| **0** | 0050 | 2 | IMPLA RESYNCHR PACEMAKER W/0 (Begin 2002) |
| **1** | 0051 | 2 | IMPLA RESYNCHRONIZATION DEFI (Begin 2002) |
| **2** | 0052 | 2 | IMPL/REPL TRANSVENOUS LEAD L (Begin 2002) |
| **3** | 0053 | 2 | IMPL/REPL PACEMAKER PLSE GE (Begin 2002) |
| **4** | 0054 | 2 | IMPL/REPL DEFIBRIL GENERATOR (Begin 2002) |

In [18]:
```
# Applicable measure procedure codes.
procedure_codes = 46.0,46.01,46.02,46.03,46.04,46.13,46.23,46.20,45.76,45.03,45.4

#preparing the list of codes for next filter
P_codes = []
for number in procedure_codes:
    code = str(number).replace('.','')
    P_codes.append(code)
print(P_codes)
```

```
['460', '4601', '4602', '4603', '4604', '4613', '4623', '462', '4576', '4503',
'4541', '4542', '4543', '4571', '4572', '4573', '4574', '4575', '4581', '4582',
'4583']
```

In [19]:
```
# Concatenate the measure procedure code list with Ram's procedure codes.
our_proc_codes = ['46.0','4601.0','4602.0','4603.0','4604.0','4613.0','4623.0','4

for i in rams_procedure_codes:
  if i in rams_procedure_codes != our_proc_codes:
    our_proc_codes.append(i)
```

In [20]:
```
del proc_reference
```

### Denominator Calculation

In [21]:
```
# Filter the inpatient file with the identified procedure codes.
P_codes = our_proc_codes

#Denominator file
inpatient_denominator = narrow_inpatient_raw[narrow_inpatient_raw['ICD9_PRCDR_CD_
                                   narrow_inpatient_raw['ICD9_PRCDR_CD_3'].isin(P
                                   narrow_inpatient_raw['ICD9_PRCDR_CD_5'].isin(P

# Add diagnosis and procedure flag if the row has a qualifying Dx or Px code.
inpatient_denominator['qualified_diagnosis'] = 0
inpatient_denominator['qualified_procedures'] = 1

inpatient_denominator.head(5)
```

Out[21]:

| | DESYNPUF_ID | CLM_ID | SEGMENT | CLM_FROM_DT | CLM_THRU_DT | CLM_PMT_A |
|---|---|---|---|---|---|---|
| **2** | 00016F745862898F | 196661177015632 | 1 | 2009-08-31 | 2009-09-02 | 500 |
| **9** | 0007F12A492FD25D | 196831176966961 | 1 | 2010-06-16 | 2010-06-19 | 2900 |
| **11** | 000C7486B11E7030 | 196641176984178 | 1 | 2008-10-15 | 2008-10-21 | 3000 |
| **19** | 00157F1570C74E09 | 196381176974293 | 1 | 2009-06-26 | 2009-06-30 | 900 |
| **26** | 001AFA59A08ABBF1 | 196331177006825 | 1 | 2008-06-21 | 2008-06-25 | 1100 |

5 rows × 25 columns

**Numerator Calculation**

In [22]:
```
# Filtering the inpatient dataset with the identified measure qualifying diagnosi
D_codes = ('480','481','482','483','485','486','487','51881','51882','51884','799

inpatient_numerator = narrow_inpatient_raw[narrow_inpatient_raw['ICD9_DGNS_CD_1']
                                   narrow_inpatient_raw['ICD9_DGNS_CD_3'].isin(D_
                                   narrow_inpatient_raw['ICD9_DGNS_CD_5'].isin(D_

inpatient_numerator['qualified_diagnosis'] = 1
inpatient_numerator['qualified_procedures'] = 0
```

In [23]:
```
#deleting unecessary variables from script to create memory
del inpatient_raw
```

**Create a combined Numerator and Denominator Dataset**

In [24]:
```
# Join the numerator and denominator.
readmission = pd.concat([inpatient_numerator,inpatient_denominator], ignore_inde

# Remove records that qualify in both the numerator and denominator datasets.
readmission = readmission.drop_duplicates(subset=['DESYNPUF_ID', 'NCH_BENE_DSCHR

readmission.head(5)
```

Out[24]:

| | DESYNPUF_ID | CLM_ID | SEGMENT | CLM_FROM_DT | CLM_THRU_DT | CLM_PMT_AI |
|---|---|---|---|---|---|---|
| **0** | 00052705243EA128 | 196991176971757 | 1 | 2008-09-12 | 2008-09-12 | 1400( |
| **1** | 00139C345A104F72 | 196691176984309 | 1 | 2009-12-07 | 2009-12-13 | 1700( |
| **2** | 001EA2F4DB30F105 | 196601176970568 | 1 | 2009-07-19 | 2009-07-22 | 500( |
| **3** | 0021D4CDAFC0609F | 196031176965445 | 1 | 2008-04-19 | 2008-04-23 | 500( |
| **4** | 00271F7DF9C2B88A | 196711177025513 | 1 | 2010-01-13 | 2010-01-14 | 2800( |

5 rows × 25 columns

**Create custom columns needed in future calculations.**

The discharge date from the previous record is put one row below it on the next most recent one claim the patient has. This pattern repeats until the patient no longer has encounters.

In [25]:
```
# Create a column to place the previous encounter discharge date by patient.
readmission['PreviousDischargeDTS'] = (readmission.sort_values(by=['DESYNPUF_ID',
                       .groupby(['DESYNPUF_ID'])['NCH_BENE_DSCHRG_DT'].shift(1, a
readmission[['DESYNPUF_ID','CLM_ID','NCH_BENE_DSCHRG_DT','PreviousDischargeDTS']]
```

Out[25]:

| | DESYNPUF_ID | CLM_ID | NCH_BENE_DSCHRG_DT | PreviousDischargeDTS |
|---|---|---|---|---|
| **0** | 00052705243EA128 | 196991176971757 | 2008-09-12 | NaT |
| **1** | 00139C345A104F72 | 196691176984309 | 2009-12-13 | NaT |
| **2** | 001EA2F4DB30F105 | 196601176970568 | 2009-07-22 | NaT |
| **3** | 0021D4CDAFC0609F | 196031176965445 | 2008-04-23 | NaT |
| **4** | 00271F7DF9C2B88A | 196711177025513 | 2010-01-14 | NaT |

Similar to discharge date, a patient's previous procedure is put on the following line (where applicable) so that it can be used to confirm if a patient qualifies as a NSQIP measure. This is determined when a qualifying diagnosis (in the list above) is on the same line as a previous qualifying procedure.

In [26]:
```
# Create a column to place the previous procedure on the next line if it is consi
readmission['previous_qualified_procedures'] = (readmission.sort_values(by=['DESY
                                .groupby(['DESYNPUF_ID'])['qualified_procedures'].shift(1,


# Review the results.
readmission[['DESYNPUF_ID','qualified_diagnosis','qualified_procedures','previous
```

Out[26]:

| | DESYNPUF_ID | qualified_diagnosis | qualified_procedures | previous_qualified_procedures |
|---|---|---|---|---|
| **0** | 00052705243EA128 | 1 | 0 | NaN |
| **1** | 00139C345A104F72 | 1 | 0 | NaN |
| **2** | 001EA2F4DB30F105 | 1 | 0 | NaN |
| **3** | 0021D4CDAFC0609F | 1 | 0 | NaN |
| **4** | 00271F7DF9C2B88A | 1 | 0 | NaN |

**Calculate the number of days that elapsed between a patient's discharge dates.**

In [27]:
```
# Creating a column with difference of days between claims.
readmission['difference_days'] = readmission['NCH_BENE_DSCHRG_DT'] - readmission[
readmission[['DESYNPUF_ID','CLM_ID','NCH_BENE_DSCHRG_DT','PreviousDischargeDTS','
```

Out[27]:

| | DESYNPUF_ID | CLM_ID | NCH_BENE_DSCHRG_DT | PreviousDischargeDTS | differen |
|---|---|---|---|---|---|
| **0** | 00052705243EA128 | 196991176971757 | 2008-09-12 | NaT | |
| **1** | 00139C345A104F72 | 196691176984309 | 2009-12-13 | NaT | |
| **2** | 001EA2F4DB30F105 | 196601176970568 | 2009-07-22 | NaT | |
| **3** | 0021D4CDAFC0609F | 196031176965445 | 2008-04-23 | NaT | |
| **4** | 00271F7DF9C2B88A | 196711177025513 | 2010-01-14 | NaT | |

In [28]:
```
# Creating a column that has 'x days' as a float (using a float rather than an in
readmission['days_float'] = readmission['difference_days'].dt.days.astype(float)
```

NSQIP is concerned about patients who had a qualifying procedure and came back with a qualifying diagnosis as long as it is within 30 days. Therefore, claims that have a gap in visits longer than 30 days are removed as well as same day claims.

In [29]:
```
# Filter the dataset by selecting the rows that have a difference in days betweer
readmission_30days = readmission[(readmission['days_float'] >= 1.0) & (readmissic

# Review the data after filtering.
readmission_30days[['DESYNPUF_ID','CLM_ID','NCH_BENE_DSCHRG_DT','PreviousDischarg
```

Out[29]:

| | DESYNPUF_ID | CLM_ID | NCH_BENE_DSCHRG_DT | PreviousDischargeDTS | differe |
|---|---|---|---|---|---|
| **7** | 002AB71D3224BE66 | 196351177026160 | 2008-09-01 | 2008-08-12 | |
| **18** | 007928DE5B0C4AA5 | 196601176961455 | 2010-01-18 | 2010-01-07 | |
| **88** | 02D6BF1BB02FAC15 | 196881176959734 | 2008-07-24 | 2008-07-23 | |
| **107** | 0342E8D226B4E3FC | 196471177009155 | 2008-05-27 | 2008-05-18 | |
| **150** | 046A3EAEF33204E1 | 196061177019627 | 2008-01-05 | 2008-01-03 | |

**Are there patients that had died within 30 days from their procedure?**

- In order to find this, we merged our beneficiary dataset with our readmission dataset. At which point, we created a column "Days before death" to count the number of days in order to see if there were any patients that died within 30 days of have a qualified surgery.

In [30]:
```
readmissions within 30 days with the beneficiary dataset of those who died.
n_death = pd.merge(readmission_30days,dead_ben, how='left', on='DESYNPUF_ID')

n_death['Days_Before_Death'] = (readmission_death['PreviousDischargeDTS'] - readm

n_death_2 = readmission_death[(readmission_death['Days_Before_Death'] >= 1.0) & (

 results. Any patients that come back would qualify as a NSQIP patient for dieing
n_death_2
```

Out[30]:

| DESYNPUF_ID | CLM_ID | SEGMENT | CLM_FROM_DT | CLM_THRU_DT | CLM_PMT_AMT | AT_PHYSN_ |
|---|---|---|---|---|---|---|

0 rows × 34 columns

**Findings:** There were no patients who died within 30 days of receiving a qualifying procedure.

Now that the datasets of those who had a qualifying diagnosis and a qualifying procedure have been merged. Filtered for only discharge dates that are within 30 days of each other, patients can be identified as a NSQIP individual if they have a 1 in the qualified_diagnosis column and a 1 in the qualified_procedure column.

In [31]:
```
# Create a NSQIP column based on the qualified diagnosis & previous qualified pro
readmission_30days['nsqip_person'] = readmission_30days['qualified_diagnosis'].as
numerator = (readmission_30days['nsqip_person'] == 2).value_counts()
denominator = readmission_30days['DESYNPUF_ID'].count()

print(numerator)
```

```
False    18181
True      5111
Name: nsqip_person, dtype: int64
```

**Calculate the overall rate of individuals who have been identified as a NSQIP patient as compared to everyone.**

In [32]:
```
# Calculate the overall rate:
rate = numerator/denominator *100
rate
```

Out[32]:
```
False    78.056844
True     21.943156
Name: nsqip_person, dtype: float64
```

**Join in the beneficiary data so that patient demographic can be used with the main dataset. This is done using the DESYNPUF_ID as a join column between the two datasets.**

In [33]:
```
# Bring in the patient information.
readmission_with_pat_info = pd.merge(readmission_30days,unique_ben, how='inner',
readmission_with_pat_info.shape[0]
```

Out[33]:
```
23292
```

**Findings:** The denominator count remained the same after joining in the beneficiary list so we know that we did not inadvertently increase our dataset size.

**Explore the NSQIP individuals**

In order to further understand the patients who were identified as a NSQIP patient, a dataset is created with just those who have a value of 2 in the nsquip_person column.

In [34]:
```
he columns of interest.
ith_pat_info[['DESYNPUF_ID','qualified_diagnosis','qualified_procedures','previou
```

Out[34]:

| | DESYNPUF_ID | qualified_diagnosis | qualified_procedures | previous_qualified_procedures | ns |
|---|---|---|---|---|---|
| **0** | 002AB71D3224BE66 | 1 | 0 | 1.0 | |
| **1** | 007928DE5B0C4AA5 | 1 | 0 | 1.0 | |
| **2** | 02D6BF1BB02FAC15 | 1 | 0 | 0.0 | |
| **3** | 0342E8D226B4E3FC | 1 | 0 | 1.0 | |
| **4** | 046A3EAEF33204E1 | 1 | 0 | 0.0 | |

In [35]:
```
# Create a dataframe of patients with a nsqip_person value equal to 2.

nsqip_people = readmission_with_pat_info[readmission_with_pat_info['nsqip_person'
```

**Classify each NSQIP patient into their qualifying measure.**

Loop through different diagnosis code columns (1 - 7) to find which measure category each individual belongs to (i.e. uti, pneumonia, etc.)

In [36]:
```
from numpy import NaN
#ICD codes that we want to find
uti = ['390','5990']
sepsis = ['99591','99592','78552']
pneumonia = ['480','481','482','483','485','486','487','51881','51882','51884','7
colon = ['4299','4280','41511','45340','9972','9971','99749','9989','56962','4349
deep_vein = ['453400','453900','453410','45342','9971','99749','99739','9972','99

#creating an empty list of our data
NSQIP_Measure = []
for i in nsqip_people['ICD9_DGNS_CD_1']:
    if i in uti:
        NSQIP_Measure.append('Urinary Tract Infection')
    elif i in sepsis:
        NSQIP_Measure.append('Sepsis')
    elif i in pneumonia:
        NSQIP_Measure.append('Pneumonia')
    elif i in colon:
        NSQIP_Measure.append('Colon')
    elif i in (deep_vein):
        NSQIP_Measure.append('deep_vein')
    else:
        NSQIP_Measure.append(NaN)

#using our list, we create a column in the dataframe with it
nsqip_people["NSQIP_Measure_1"] = NSQIP_Measure
```

In [37]:
```
#creating an empty list of our data
NSQIP_Measure = []
for i in nsqip_people['ICD9_DGNS_CD_2']:
    if i in uti:
        NSQIP_Measure.append('Urinary Tract Infection')
    elif i in sepsis:
        NSQIP_Measure.append('Sepsis')
    elif i in pneumonia:
        NSQIP_Measure.append('Pneumonia')
    elif i in colon:
        NSQIP_Measure.append('Colon')
    elif i in (deep_vein):
        NSQIP_Measure.append('deep_vein')
    else:
        NSQIP_Measure.append(NaN)

#using our list, we create a column in the dataframe with it
nsqip_people["NSQIP_Measure_2"] = NSQIP_Measure
```

In [38]:
```
NSQIP_Measure = []
for i in nsqip_people['ICD9_DGNS_CD_3']:
    if i in uti:
        NSQIP_Measure.append('Urinary Tract Infection')
    elif i in sepsis:
        NSQIP_Measure.append('Sepsis')
    elif i in pneumonia:
        NSQIP_Measure.append('Pneumonia')
    elif i in colon:
        NSQIP_Measure.append('Colon')
    elif i in (deep_vein):
        NSQIP_Measure.append('deep_vein')
    else:
        NSQIP_Measure.append(NaN)

#using our list, we create a column in the dataframe with it
nsqip_people["NSQIP_Measure_3"] = NSQIP_Measure
```

In [39]:
```
NSQIP_Measure = []
for i in nsqip_people['ICD9_DGNS_CD_4']:
    if i in uti:
        NSQIP_Measure.append('Urinary Tract Infection')
    elif i in sepsis:
        NSQIP_Measure.append('Sepsis')
    elif i in pneumonia:
        NSQIP_Measure.append('Pneumonia')
    elif i in colon:
        NSQIP_Measure.append('Colon')
    elif i in (deep_vein):
        NSQIP_Measure.append('deep_vein')
    else:
        NSQIP_Measure.append(NaN)

#using our list, we create a column in the dataframe with it
nsqip_people["NSQIP_Measure_4"] = NSQIP_Measure
```

In [40]:
```
NSQIP_Measure = []
for i in nsqip_people['ICD9_DGNS_CD_5']:
    if i in uti:
        NSQIP_Measure.append('Urinary Tract Infection')
    elif i in sepsis:
        NSQIP_Measure.append('Sepsis')
    elif i in pneumonia:
        NSQIP_Measure.append('Pneumonia')
    elif i in colon:
        NSQIP_Measure.append('Colon')
    elif i in (deep_vein):
        NSQIP_Measure.append('deep_vein')
    else:
        NSQIP_Measure.append(NaN)

#using our list, we create a column in the dataframe with it
nsqip_people["NSQIP_Measure_5"] = NSQIP_Measure
```

In [41]:
```
NSQIP_Measure = []
for i in nsqip_people['ICD9_DGNS_CD_6']:
    if i in uti:
        NSQIP_Measure.append('Urinary Tract Infection')
    elif i in sepsis:
        NSQIP_Measure.append('Sepsis')
    elif i in pneumonia:
        NSQIP_Measure.append('Pneumonia')
    elif i in colon:
        NSQIP_Measure.append('Colon')
    elif i in (deep_vein):
        NSQIP_Measure.append('deep_vein')
    else:
        NSQIP_Measure.append(NaN)

#using our list, we create a column in the dataframe with it
nsqip_people["NSQIP_Measure_6"] = NSQIP_Measure
```

In [42]:
```
NSQIP_Measure = []
for i in nsqip_people['ICD9_DGNS_CD_7']:
    if i in uti:
        NSQIP_Measure.append('Urinary Tract Infection')
    elif i in sepsis:
        NSQIP_Measure.append('Sepsis')
    elif i in pneumonia:
        NSQIP_Measure.append('Pneumonia')
    elif i in colon:
        NSQIP_Measure.append('Colon')
    elif i in (deep_vein):
        NSQIP_Measure.append('deep_vein')
    else:
        NSQIP_Measure.append(NaN)

#using our list, we create a column in the dataframe with it
nsqip_people["NSQIP_Measure_7"] = NSQIP_Measure
```

**Consolidate the newly created NSQIP_Measure_1-7 columns into 1 where all the rows will be populated.**

In [43]:
```
# Create a column that summarizes the patient's qualifying diagnosis category.
nsqip_people['diagnosis'] = np.nan
nsqip_people['diagnosis'] = nsqip_people.diagnosis.fillna(nsqip_people.NSQIP_Meas
```

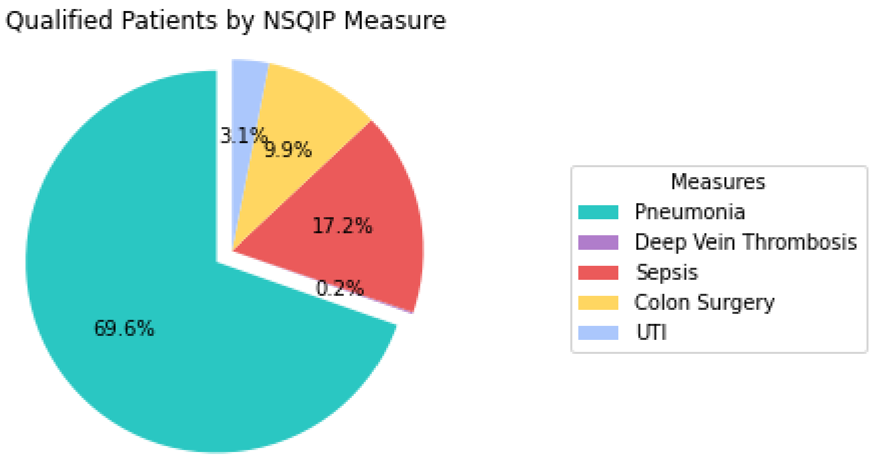
**Findings**

In [44]:
```
# Frequency count of each diagnosis.
diagnosis_freq = nsqip_people['diagnosis'].value_counts()

# Total amount of people who have a matched diagnosis.
nsqip_people.shape[0]

# Creating a dataframe of the frequencies.
diagnosis_freq.to_frame()

# Diagnosis/diagnosis_total.
diagnosis_freq['diagnosis_total'] = nsqip_people.shape[0]

# Do a ratio of diagnosis/numerator (2000ish people)
diagnosis_freq['numerator_total'] = readmission_with_pat_info.shape[0]

diagnosis_freq
```

Out[44]:
```
Pneumonia                  3542
Sepsis                      847
Colon                       515
Urinary Tract Infection     162
deep_vein                     8
diagnosis_total            5111
numerator_total           23292
Name: diagnosis, dtype: int64
```

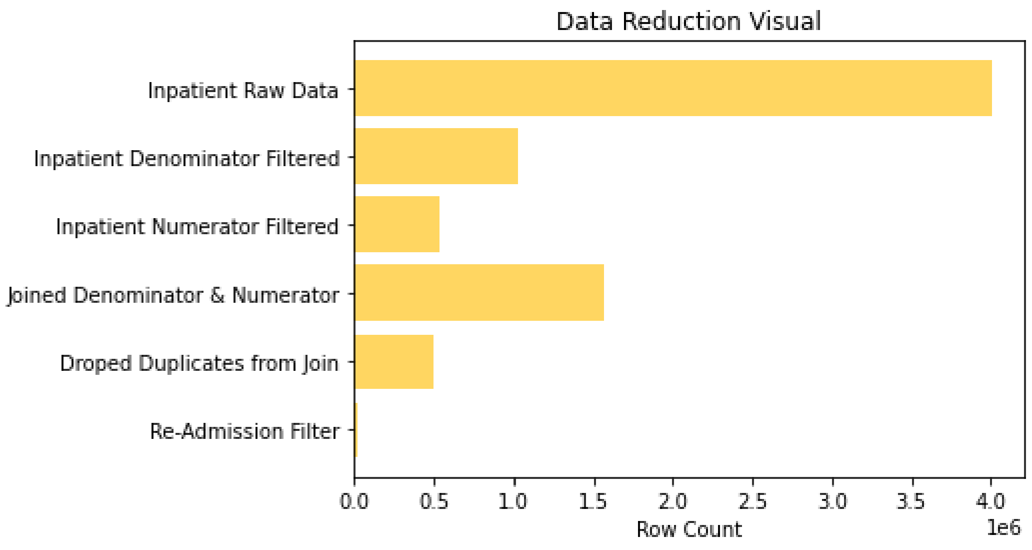
**Look at the breakdown of identified NSQIP individuals and how they are distributed across the various measures. In this particular instance, the denominator is the "numerator" and the individual measures are the numerator.**

In [45]:
```
# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = ['Pneumonia','Deep Vein Thrombosis', 'Sepsis', 'Colon Surgery', 'UTI']
sizes = [3678, 8, 908, 524, 164]
explode = (0.1, 0, 0, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')
colors = ['#2AC7C2','#B07DCB','#EB5A5A','#FFD661','#ABC7FC']
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, colors=colors, autopct='%1.1f%%',
        shadow=False, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
ax1.legend(labels,
          title="Measures",
          loc="center left",
          bbox_to_anchor=(1, 0, 0.5, 1))
plt.title('Qualified Patients by NSQIP Measure')
plt.show()
```



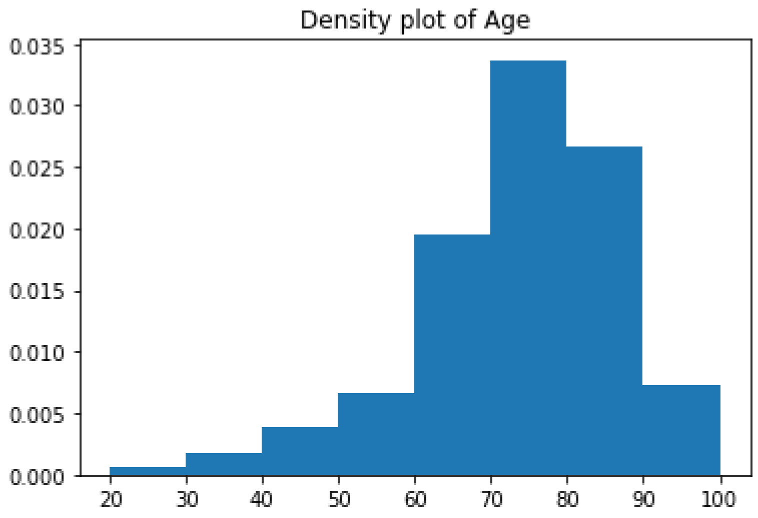
In [46]:
```
plt.barh(range(0,6), [23557,494344,1564154,533220,1030934,4008836],
         color='#FFD661',
         tick_label=["Re-Admission Filter","Droped Duplicates from Join",
                     "Joined Denominator & Numerator","Inpatient Numerator Filter
                     "Inpatient Denominator Filtered","Inpatient Raw Data"])
plt.title("Data Reduction Visual")
plt.xlabel("Row Count")
```

Out[46]:
```
Text(0.5, 0, 'Row Count')
```



In [47]:
```
#Calculating the age for patients
readmission_with_pat_info['age'] = readmission_with_pat_info['NCH_BENE_DSCHRG_DT'
readmission_with_pat_info['age'] = (readmission_with_pat_info['age'].dt.days.asty
```

In [48]:
```
#Creating histogram of age range in dataset
bins = [20,30,40,50,60,70,80,90,100]
fig = plt.figure()
ax = fig.add_subplot(111)
n, bins, rectangles = ax.hist(readmission_with_pat_info['age'], bins, density=Tru
fig.canvas.draw()
plt.title('Density plot of Age')
plt.show()
```



**Conclusion**

While we found patients who qualified for 5 of the 6 measures we sought after to find, the amount of data that is available through the CMS PUF files is not complete enough to submit to NSQIP. This project highlighted the plausiblity of building a dataset out and identifying patients who might fit the different measure defintions but that is the extent of this dataset. The other limitation with using the CMS files is that they are more than 10 years old and healthcare codes have changed from ICD9 diagnosis and procedure to ICD10 diagnosis and procedure. This means that the codes used in this analysis would not be transferrable directly but would rather need to be translated into the respective ICD-10 equivalents.