PART 1: DATA ANALYSIS

First and foremost, we will explore and examine the integrity of data sets so that we won't misrepresent the results. This notebook will also involve some descriptive stats to understand the data at the high level.

The detail mechanism will be explained on each code block. Here is the summary of data transformation on each data set -

Patient

- Extract Zip Code and merge with ***ACA and ZCATA*** data set and get income data. Add additional "Income Tier" Column.
- patients.csv in new_data directory will include income, income_tier, death column (0 and 1), minority column.

Encounter

- Encounter has more rows than patient since it records all the history of patients' encounter with the hospital or clinic.
- "Emergency" or "emergency" key words will be used to filter the emergency cases.
-

Care Plan

- careplan data set has more precise on diagnosis than encounters which have a lot of null values. Hence, we merge with careplan to understand the most common reason to pay emergency visits.
- changed the name of REASONDESCRIPTION column to Diagnosis and that of DESCRIPTION column to Treatment.

In [76]:
```
#libraries
import pandas as pd
import numpy as np
import datetime as dt
import re
import math
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set(style= 'white')
sns.set(style = 'whitegrid', color_codes = True)
```

Patient Data Set

There are 1462 entries with columns such as ID,RACE and ADDRESS which will be later used to find income after connecting with ACS and zip_to_zcta data set. There are also exactly 1462 unique patients' ID - which can be used as primary keys.

In [77]:

```python
#../data since the code is stored in different folder
patients = pd.read_csv('../data/patients.csv')

#drop the unncessary columns
patients.drop(['Unnamed: 0','SSN','DRIVERS','PASSPORT','BIRTHPLACE','SUFFIX','MAIDEN'], axis= 1, inplace = True)

#Need to double check the count of unique id
print("Rows and columns of the data set {}".format(patients.shape))
print("Number of unique Patients IDs {}".format(patients.ID.nunique()))
patients.head(3)
```

```
Rows and columns of the data set (1462, 11)
Number of unique Patients IDs 1462
```

Out[77]:

	ID	BIRTHDATE	DEATHDATE	PREFIX	FIRST	LAST	MARITAL	RACI
0	71949668-1c2e-43ae-ab0a-64654608defb	5/28/88	NaN	Mrs.	Elly802	Koss811	M	hispani
1	c2caaace-9119-4b2d-a2c3-4040f5a9cf32	9/22/36	11/6/87	Mr.	Kim254	Barrows624	S	asia
2	96b24072-e1fe-49cd-a22a-6dfb92c3994c	8/9/39	NaN	Ms.	Jacquelyn868	Shanahan20	S	hispani

In [78]:

```
'''
- Get the zip code from patient data using regex.

- Adjust some mismatched values such as O instead of 0 at the front of zipcode

- Add additional Zip Code column to later match with ZCTA
'''

def get_zip_code(address):
    if address != address or not address:
        return "MA"
    else:

        if address[-8] == 'O':
            modified_string = list(address)
            modified_string[-8] = '0'
            _string = "".join(modified_string)

            return _string.split()[-2]
        else:
            zip_ = re.findall('.*(\d{5}(\-\d{4})?).*$', address)
            return zip_[0][0]
patients['Zip_Code'] = patients['ADDRESS'].apply(lambda x: get_zip_code(x))
```

In [79]:

```
#Read zip_to_zcta file
zip_to_zcta = pd.read_csv('../data/zip_to_zcta_2019.csv')
zip_to_zcta.drop(['PO_NAME','ZIP_TYPE','zip_join_type'], axis= 1 , inplace= True)
zip_to_zcta.tail()
```

Out[79]:

	ZIP_CODE	STATE	ZCTA
41102	96943	FM	No ZCTA
41103	96944	FM	No ZCTA
41104	96960	MH	No ZCTA
41105	96970	MH	No ZCTA
41106	96898	NaN	No ZCTA

In [80]:

```
#add zero to the zip to match with patients table

def add_zero_to_zip (zip_code):
    if (len(str(zip_code)) == 4):
        new_zip = '0'+str(zip_code)
        return new_zip
zip_to_zcta['ZIP_CODE'] = zip_to_zcta['ZIP_CODE'].apply(lambda x: add_ze
ro_to_zip(x))
zip_to_zcta[zip_to_zcta.STATE == 'MA'].head()
```

Out[80]:

	ZIP_CODE	STATE	ZCTA
194	01001	MA	01001
195	01002	MA	01002
196	01003	MA	01003
197	01004	MA	01002
198	01005	MA	01005

In [81]:

```
#Left Join with Zip Code from patient table
patients = pd.merge(patients, zip_to_zcta, left_on = 'Zip_Code',right_on
='ZIP_CODE', how='left')

#unique count looks fine
print(patients.shape)

#Delete additional Zip Code
patients.drop(['ZIP_CODE'], axis= 1, inplace=  True)
print(patients.columns)
```

```
(1462, 15)
Index(['ID', 'BIRTHDATE', 'DEATHDATE', 'PREFIX', 'FIRST', 'LAST', 'MARI
TAL',
       'RACE', 'GENDER', 'ADDRESS', 'AGE', 'Zip_Code', 'STATE', 'ZCT
A'],
      dtype='object')
```

In [82]:

```
#read ACS file to get income on each zip code
ACS = pd.read_csv('../data/ACS.csv')
new_header = ACS.iloc[0]
ACS = ACS[1:]
ACS.columns = new_header
ACS.columns = ['Id','Id2','Geography','Households','H_margin','Families'
,
               'F_margin','Married_couple','M_c_margin','Nonfamily_househ
old',
               'N_house_margin']

ACS.head()
```

Out[82]:

	Id	Id2	Geography	Households	H_margin	Families	F_margin	Married_coup
1	8600000US00601	00601	ZCTA5 00601	11507	1192	13283	2167	1849
2	8600000US00602	00602	ZCTA5 00602	15511	1381	18694	1752	2364
3	8600000US00603	00603	ZCTA5 00603	16681	903	20719	1164	2868
4	8600000US00606	00606	ZCTA5 00606	11648	2555	14871	2585	1785
5	8600000US00610	00610	ZCTA5 00610	17751	1326	21509	1715	2542

In [83]:

```
# join zcta to get income
patients = pd.merge(patients, ACS, left_on='ZCTA', right_on = 'Id2', how
='left')
patients.columns
```

Out[83]:

```
Index(['ID', 'BIRTHDATE', 'DEATHDATE', 'PREFIX', 'FIRST', 'LAST', 'MARI
TAL',
       'RACE', 'GENDER', 'ADDRESS', 'AGE', 'Zip_Code', 'STATE', 'ZCTA',
'Id',
       'Id2', 'Geography', 'Households', 'H_margin', 'Families', 'F_mar
gin',
       'Married_couple', 'M_c_margin', 'Nonfamily_household',
       'N_house_margin'],
      dtype='object')
```

In [84]:

```
'''
- The mismatched entries (such as - **) are replaced with NaN values.

- The string incomes such as "2500+" are striped and transformed into appropriate int values

'''
def handle_missing_income (income):

    if (income != income):

        return np.nan

    elif ("-" == income) or ("**" == income):
        return np.nan
    elif (income[-1] == '-') or (income[-1] == "+"):

        if ("," in income):
            income = income.replace(",","")
        return int(income[:-1])

    else:
        return (int(income))

patients['Households'] = patients['Households'].apply(lambda x: handle_missing_income(x))
patients['Families'] = patients['Families'].apply(lambda x: handle_missing_income(x))
patients['Married_couple'] = patients['Married_couple'].apply(lambda x: handle_missing_income(x))
patients['Nonfamily_household'] = patients['Nonfamily_household'].apply(lambda x: handle_missing_income(x))
```

In [85]:

```
# The missing income are replaced with Median Income of that state (MA)
patients['Households'].fillna((patients['Households'].median()), inplace=True)
patients['Families'].fillna((patients['Families'].median()), inplace=True)
patients['Married_couple'].fillna((patients['Married_couple'].median()), inplace=True)
patients['Nonfamily_household'].fillna((patients['Nonfamily_household'].median()), inplace=True)
```

In [86]:

```
'''
According to the Bureau Census, the household can be one or more member
s.

If the Marital status is single , the income will be household's.
If the Marital status is Married , the income will be Families'.
If the Marital status is missing , the income will be household's.
'''
def add_income (status, household,families,married_couple,non_family):

    if (status is None) or (status != status):
        return household
    elif (status == "M"):
        return families
    elif (status == 'S'):
        return non_family

patients['Income'] = patients.apply(lambda x: add_income(x['MARITAL'],
                                       x['Households'], x['Families'],
                                       x['Married_couple'],x['Nonfamily
_household']), axis = 1 )
patients.shape
```

Out[86]:

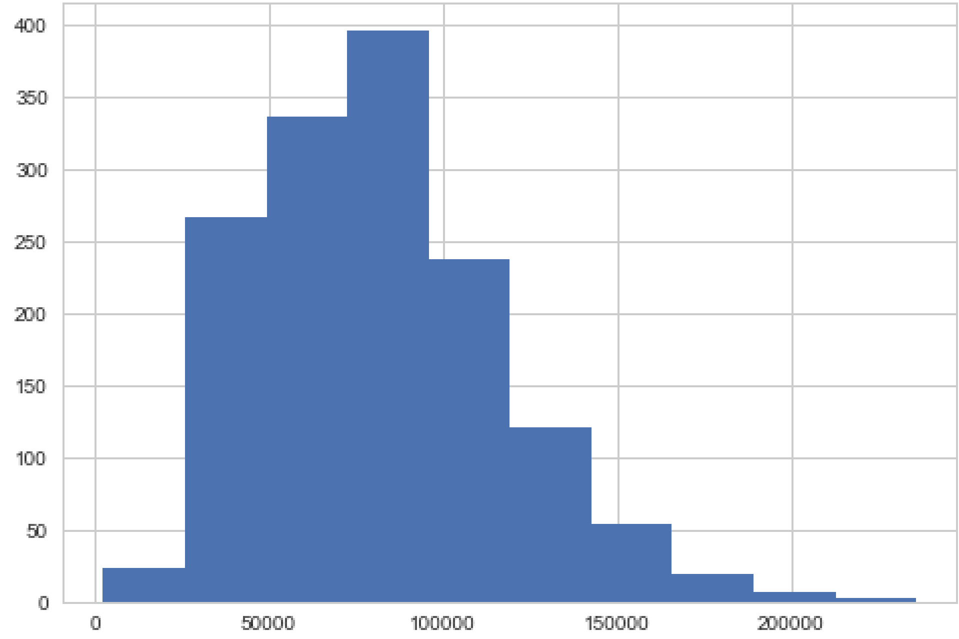
```
(1462, 26)
```

In [87]:

```
#Distribution of income
patients.Income.hist()
patients.Income.describe()
```

Out[87]:

```
count      1462.000000
mean      82027.034200
std       35207.884904
min        2500.000000
25%       53219.000000
50%       80365.500000
75%      101184.750000
max      235766.000000
Name: Income, dtype: float64
```



In [88]:

```
#based on the distribution, we divide into three income tier - lower, mi
ddle and upper.
def income_tier (income):
    if income <= 53219.0:
        return "lower"
    elif (income > 53219.0 and income <= 101184.0):
        return "middle"
    elif (income > 101184.0):
        return "upper"
patients['Income_Tier'] = patients['Income'].apply(lambda x: income_tier
(x))
patients.columns
```

Out[88]:

```
Index(['ID', 'BIRTHDATE', 'DEATHDATE', 'PREFIX', 'FIRST', 'LAST', 'MARI
TAL',
       'RACE', 'GENDER', 'ADDRESS', 'AGE', 'Zip_Code', 'STATE', 'ZCTA',
'Id',
       'Id2', 'Geography', 'Households', 'H_margin', 'Families', 'F_mar
gin',
       'Married_couple', 'M_c_margin', 'Nonfamily_household', 'N_house_
margin',
       'Income', 'Income_Tier'],
      dtype='object')
```

In [89]:

```
#Then drop the unnecessary columns
patients.drop(['Id','Id2', 'Geography', 'Households', 'H_margin', 'Famil
ies', 'F_margin',
       'Married_couple', 'M_c_margin', 'Nonfamily_household', 'N_house_m
argin'], axis = 1, inplace = True)

#And change the column name to be consistent
patients.columns = ['Patient_Id', 'Birth_Date', 'Death_Date', 'Prefix',
'First', 'Last',
                    'Marital','Race', 'Gender', 'Address', 'Age','Zip_Co
de', 'STATE', 'ZCTA',
       'Income', 'Income_Tier']
```

In [90]:

```
#Create dummy column for death
def create_dummy_for_death_date(date_):
    return 0 if (date_ != date_) else 1


patients['Death'] = patients["Death_Date"].apply(lambda x: create_dummy_
for_death_date(x) )

#group black and black or african american group
patients['Race'].replace(to_replace = 'black or african american', value
= 'black', inplace = True)

#create minority group for black and hispanic patients
patients['Minority'] = patients["Race"].map({"black":1,"hispanic":1, "wh
ite":0, "asian":0})

#Store patients with income data set in new_data directory
patients.to_csv('../new_data/patients.csv')
```

Encounter Data

In [91]:

```
#check encounter data before merging with patients - to get the emergenc
y visit from 2008-2006
encounters = pd.read_csv('../data/encounters.csv')
print("Rows and columns of the data set {}".format(encounters.shape))
print("Number of unique Patients IDs {}".format(encounters.PATIENT.nuniq
ue()))
encounters.columns
```

```
Rows and columns of the data set (20524, 8)
Number of unique Patients IDs 1461
```

Out[91]:

```
Index(['Unnamed: 0', 'ID', 'DATE', 'PATIENT', 'CODE', 'DESCRIPTION',
       'REASONCODE', 'REASONDESCRIPTION'],
      dtype='object')
```

In [92]:

```
encounters.columns = ['Unnamed: 0', 'Encounter_Id', 'Date_Visit', 'Patie
nt_Id', 'Encounter_Code', 'Encounter_Description',
       'Reason_Code', 'Reason_Description']
encounters.drop(['Unnamed: 0'], axis= 1, inplace = True)
```

Join Patient, Encounter and careplan Data Set

Next we will join patient and encounter data set. This will allow us to filter the date and the reason for the visit such as emergency cases.

In [93]:

```
# inner join the two data set and drop the unnecessary columns
patients_encounter = pd.merge(patients, encounters, on='Patient_Id', how
='inner')

#check the number of unique Patient_id again which is supposed to be the
same number from patients'
print("Rows and columns of the data set {}".format(patients_encounter.sh
ape))
print("Number of unique Patients IDs {}".format(patients_encounter.Patie
nt_Id.nunique()))
```

```
Rows and columns of the data set (20524, 24)
Number of unique Patients IDs 1461
```

Filtering the emergency visits and the time line from 2008 and 2016

Before merging with conditions, we should filter only for emergency cases

- Filtering date is simple as mentioned below.
- Filtering emergency causes is done by using regex to match "emergency" and "Emergency" cases

In [94]:

```
#change DATE to datetime format to filter between 2008 and 2016
patients_encounter['Date_Visit'] = pd.to_datetime(patients_encounter['Date_Visit'], errors='coerce')
patients_encounter['Death_Date'] = pd.to_datetime(patients_encounter['Death_Date'], errors='coerce')
patients_encounter = patients_encounter[(patients_encounter['Date_Visit'] > '01-01-2008')
                                        & (patients_encounter['Date_Visit'] < '31-12-2016')]
```

In [95]:

```
# filter for emergency visits
patients_encounter_emergency = patients_encounter[patients_encounter.Encounter_Description.str.contains('Emergency', regex= True, na=False)
                            | (patients_encounter.Encounter_Description.str.contains('emergency', regex= True, na=False))]

#save the filtered data
patients_encounter_emergency.to_csv('../new_data/patients_encounter_emergency.csv')

#There are only 569 patients left from 1461 after filtering the given timeline and for emergency reason
print("Number of rows and columns {} ".format(patients_encounter_emergency.shape))
print ("Number of unique Patient ID {}".format(patients_encounter_emergency.Patient_Id.nunique()))
```

```
Number of rows and columns (911, 24)
Number of unique Patient ID 569
```

In [96]:

```
nonduplicate_emergency = patients_encounter_emergency.drop_duplicates(subset = 'Patient_Id')


#Drop the duplicate Patient_Id for emergency case
print("Number of rows and columns {} ".format(nonduplicate_emergency.shape))
print ("Number of unique Patient ID {}".format(nonduplicate_emergency.Patient_Id.nunique()))
```

```
Number of rows and columns (569, 24)
Number of unique Patient ID 569
```

CarePlan data

In [97]:

```
careplans = pd.read_csv('../data/careplans.csv')
careplans.columns
```

Out[97]:

```
Index(['Unnamed: 0', 'ID', 'START', 'STOP', 'PATIENT', 'ENCOUNTER', 'CODE',
       'DESCRIPTION', 'REASONCODE', 'REASONDESCRIPTION'],
      dtype='object')
```

In [98]:

```
careplans.drop(['Unnamed: 0'], axis= 1, inplace = True)

#Change the column name
careplans.columns = ['Careplan_Id', 'Start', 'Stop', 'Care_Patient_Id', 'Encounter_Id'
                     , 'CODE','Treatment', 'Care_Reason_Code', 'Diagnosis']
```

In [99]:

```
emergency_care = pd.merge(nonduplicate_emergency, careplans, left_on= ['Patient_Id'], right_on = ['Care_Patient_Id'], how='inner')


#drop the duplicates of same patient and same reason
emergency_care.drop_duplicates(['Patient_Id','Diagnosis'], inplace = True)

emergency_care_unique = emergency_care.drop_duplicates(['Patient_Id'])
emergency_care_unique.to_csv('../new_data/emergency_care_unique.csv')
```

In [100]:

```
predibetes_patients = emergency_care[emergency_care.Diagnosis == 'Prediabetes'][['Patient_Id','Date_Visit',
                                                        'Death_Date','Diagnosis','Treatment','Age','Race','Death',
                                                        'Income','Income_Tier']]

predibetes_patients.to_csv('../new_data/predibetes_patients.csv')
```