Atrium Drive Me to the Doctor

- Maria-Elena Gorini
- Santhosh Ramaraj
- Jacob Menchak
- Sudarshan Mahanubhav

Our Question

What are the demographics surrounding atrial fibrillation in Medicare patients?

- Affects 2.7 million - 6.1 million people in the US [1,2]
- Affects 10-15% people over 80 years old [3]
- Costs the US health system $26 billion [4]
- Causes 5 increased risk to have a stroke [4,5]

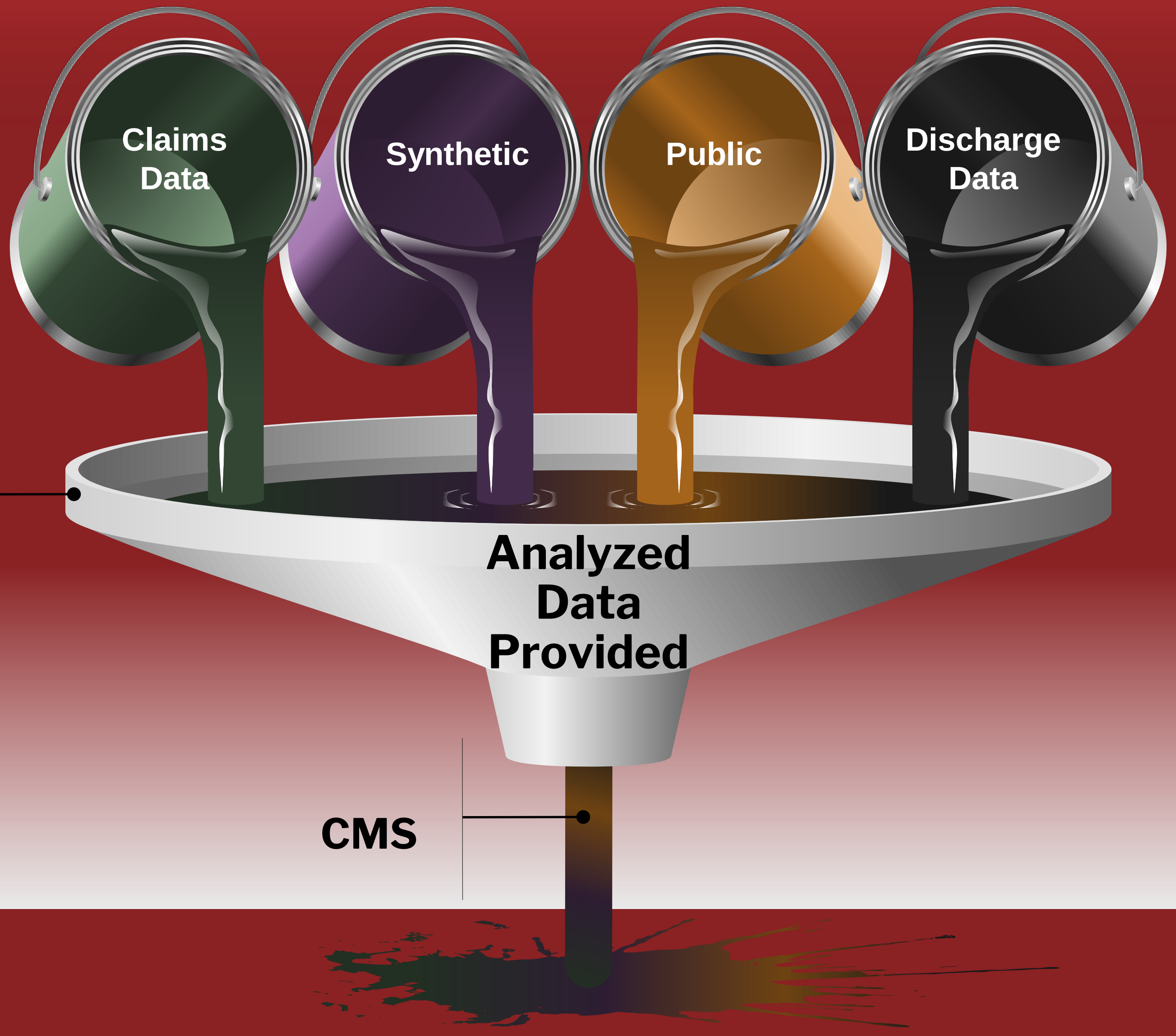
1. Miyasaka Y, Barnes ME, Gersh BJ, Cha SS, Bailey KR, Abhayaratna WP, et al. Secular trends in incidence of atrial fibrillation in Olmsted County, Minnesota, 1980 to 2000, and implications on the projections for future prevalence. ***Circulation***. 2006;114(2):199–25.
2. Go AS, Hylek EM, Phillips KA, Chang Y, Henault LE, Selby JV, Singer DE. Prevalence of diagnosed atrial fibrillation in adults: national implications for rhythm management and stroke prevention: the AnTicoagulation and Risk Factors in Atrial Fibrillation (ATRIA) Study. ***JAMA***. 2001;285(18):2370–5.
3. http://newsroom.heart.org/pr/aha/1329.aspx?ncid=36436
4. https://www.healthline.com/health/living-with-atrial-fibrillation/facts-statistics-infographic#8
5. https://www.heart.org/en/health-topics/atrial-fibrillation/why-atrial-fibrillation-af-or-afib-matters/high-blood-pressure-afib-and-your-risk-of-stroke

Our Data

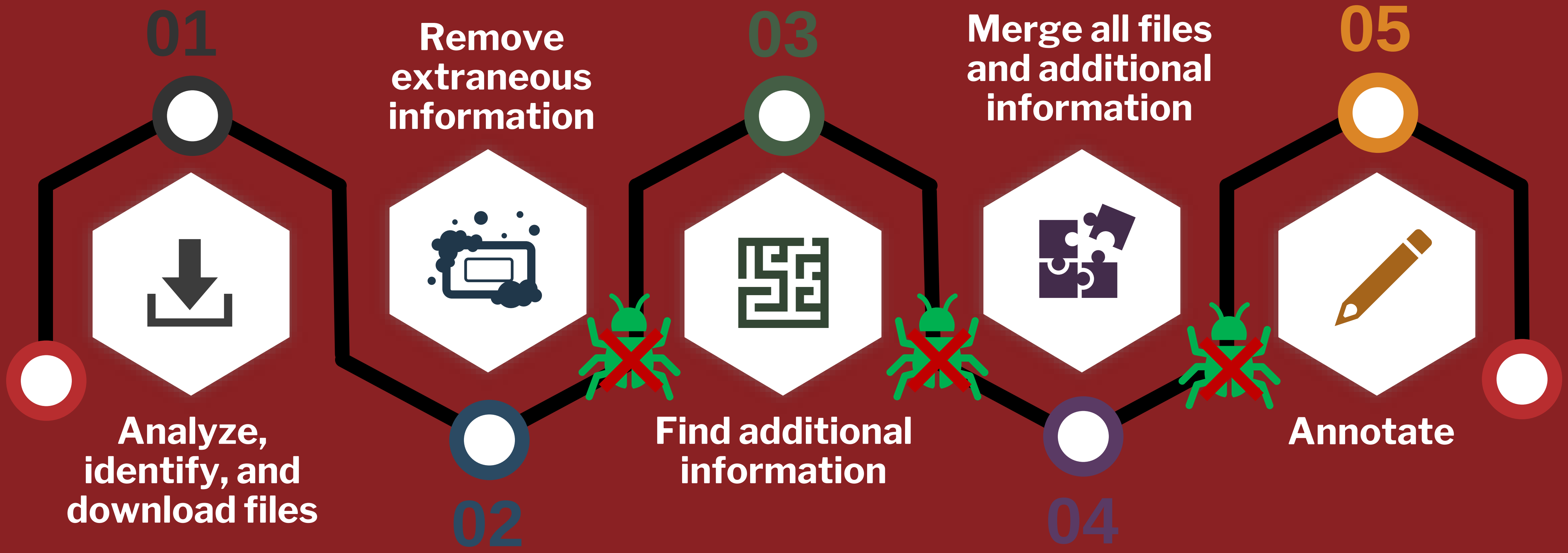
Synthia:
Models the Medical History of Synthetic Patients

Texas Hospital:
Discharge Data Public Use Data File

CMS:
Claims Synthetic Public Use Files (SynPUFs)



Data Clean Up



01

Analyze, identify, and download files

02

Remove extraneous information

03

Find additional information

04

Merge all files and additional information

05

Annotate

Centers for Medicare and Medicaid Services (CMS): Converting County & State Codes to Names

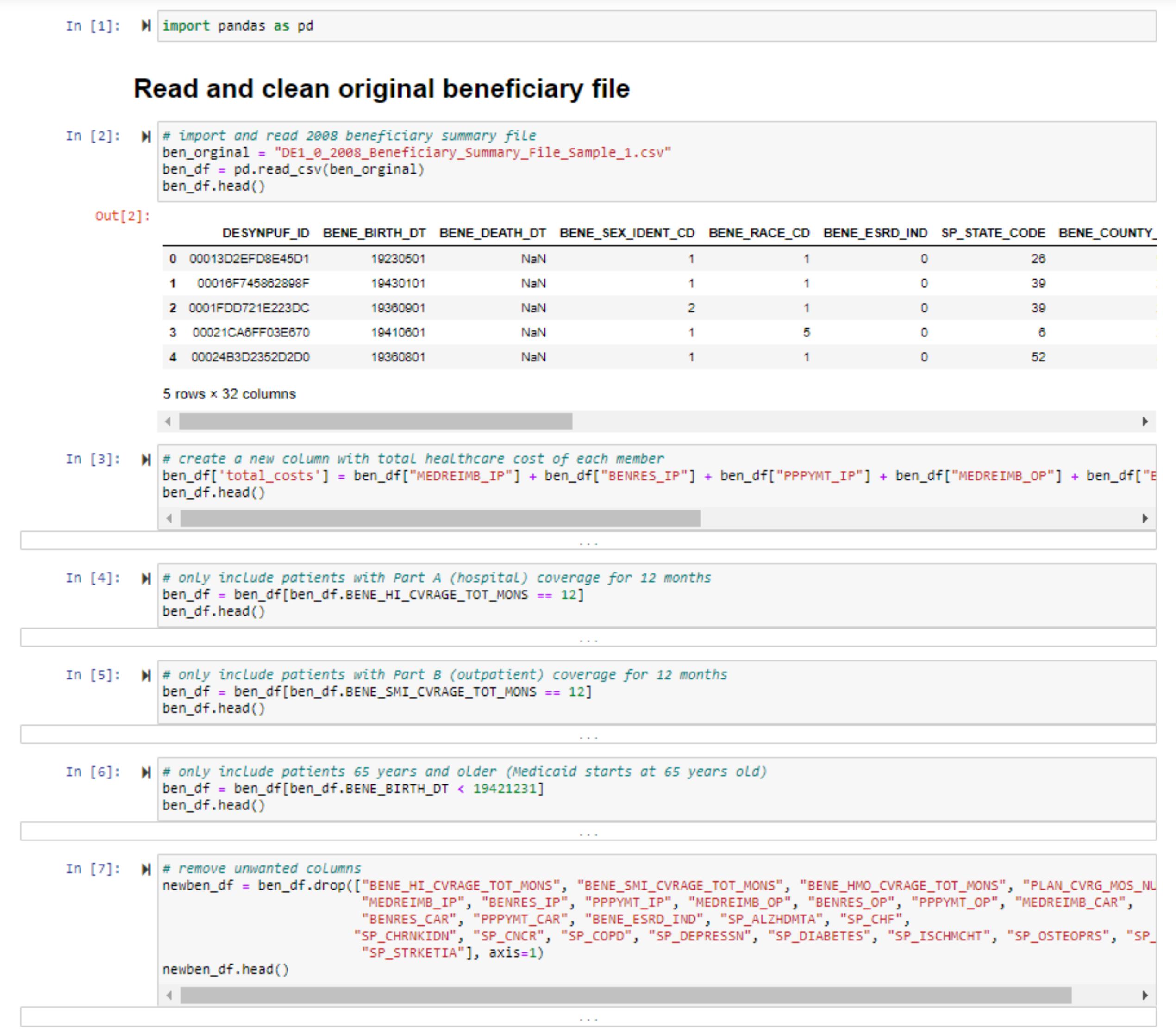
- CMS “Claims Synthetic Public Use Files” (SynPUFs) data samples (csv files) provide county codes & state codes
- Need county names and state names for Google Maps API (to produce heatmaps of AFIB patients highest claims count and highest total claims cost)
- Created a MS Excel VBA Macro Tool to find and match each county code & state code to its respective county name & state name (for each patient record in CMS csv data samples)

MS Excel VBA Macro Tool using CMS Data Sample: County & State – Codes to Names

MS Excel VBA Macros:

- 1 – Find State Code (integer) & Replace with Text
    - Uses an integer & text lookup table
    - Necessary to perform State Code Index & Match operation
- 2 – Find County Name for each State
    - Uses a state code, county code, & county name lookup table
    - Locates and stores correct county name for a given state
- 3 – Collect Data for CSV Output
    - Copies county & state code and name columns to “CSV output” sheet
- 4 – Save “CSV Output” sheet data as a new CSV file

Data Clean Up



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

Read and clean original beneficiary file

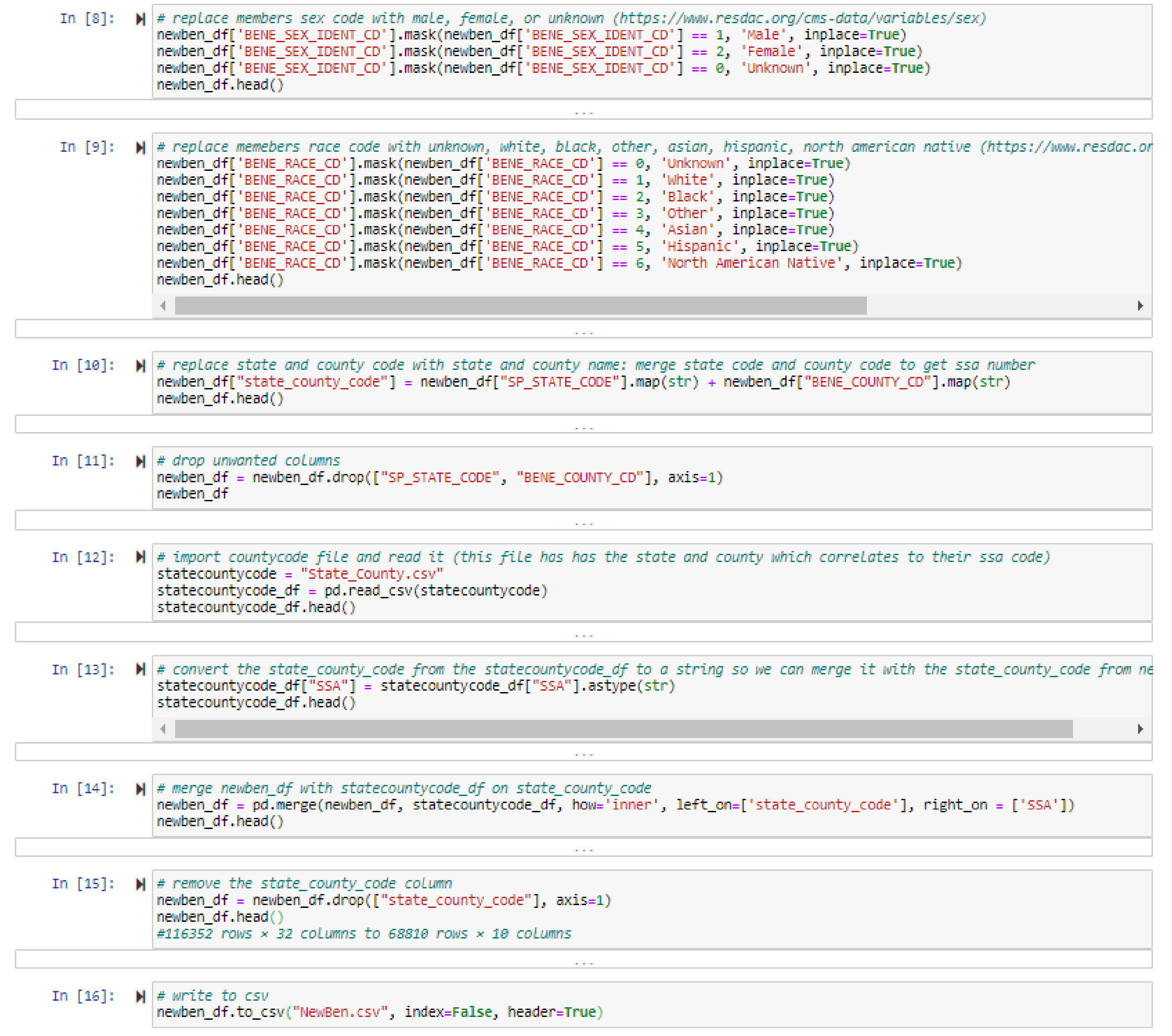
```
In [2]:  # import and read 2008 beneficiary summary file
         ben_orginal = "DE1_0_2008_Beneficiary_Summary_File_Sample_1.csv"
         ben_df = pd.read_csv(ben_orginal)
         ben_df.head()
```

Out[2]:

	DESYNPUF_ID	BENE_BIRTH_DT	BENE_DEATH_DT	BENE_SEX_IDENT_CD	BENE_RACE_CD	BENE_ESRD_IND	SP_STATE_CODE	BENE_COUNTY_
0	00013D2EFD8E45D1	19230501	NaN	1	1	0	26	
1	00016F745862898F	19430101	NaN	1	1	0	39	
2	0001FDD721E223DC	19360901	NaN	2	1	0	39	
3	00021CA6FF03E670	19410601	NaN	1	5	0	6	
4	00024B3D2352D2D0	19360801	NaN	1	1	0	52	

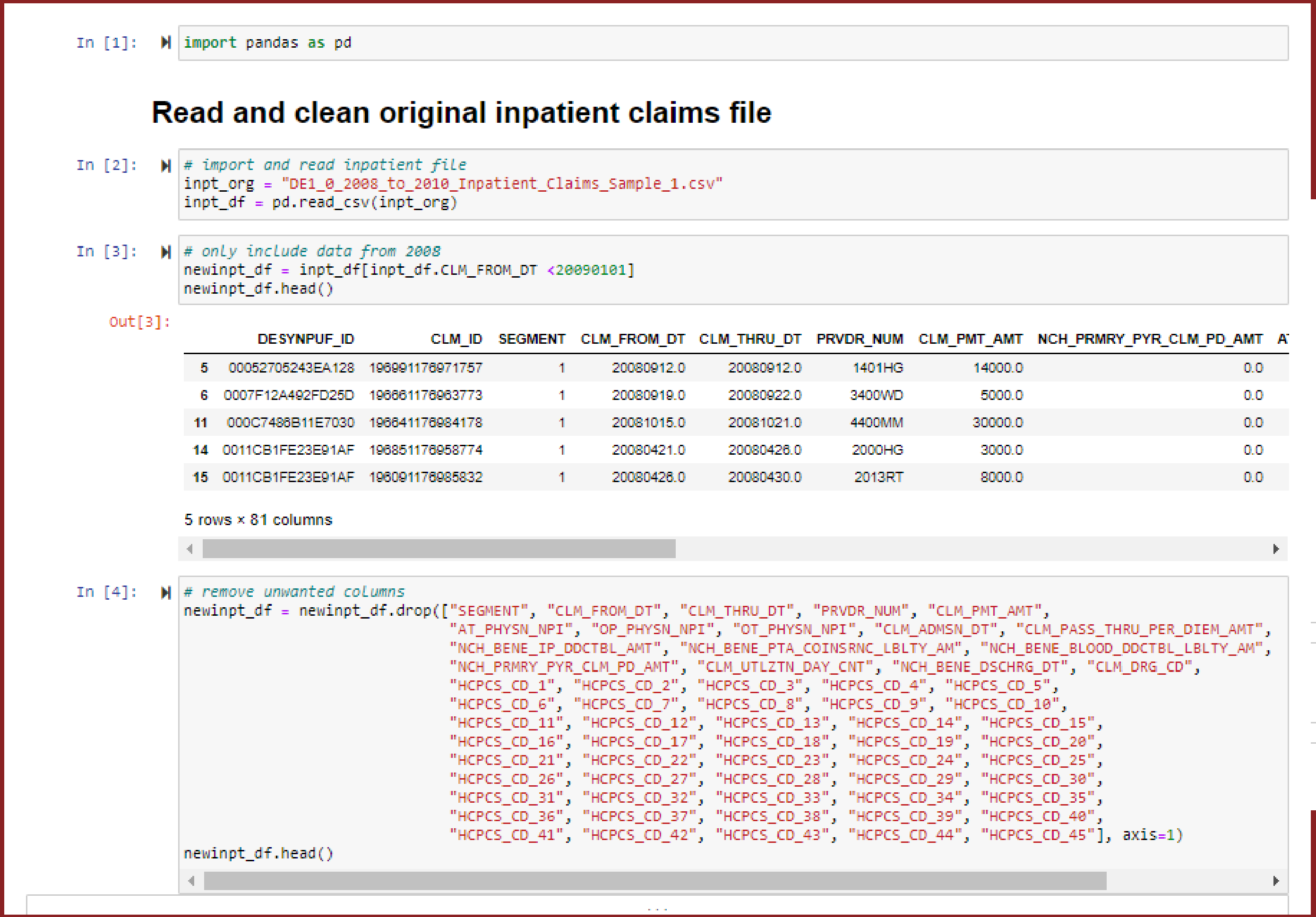
5 rows × 32 columns

```
In [3]:  # create a new column with total healthcare cost of each member
         ben_df['total_costs'] = ben_df["MEDREIMB_IP"] + ben_df["BENRES_IP"] + ben_df["PPPYMT_IP"] + ben_df["MEDREIMB_OP"] + ben_df["E
         ben_df.head()

In [4]:  # only include patients with Part A (hospital) coverage for 12 months
         ben_df = ben_df[ben_df.BENE_HI_CVRAGE_TOT_MONS == 12]
         ben_df.head()

In [5]:  # only include patients with Part B (outpatient) coverage for 12 months
         ben_df = ben_df[ben_df.BENE_SMI_CVRAGE_TOT_MONS == 12]
         ben_df.head()

In [6]:  # only include patients 65 years and older (Medicaid starts at 65 years old)
         ben_df = ben_df[ben_df.BENE_BIRTH_DT < 19421231]
         ben_df.head()

In [7]:  # remove unwanted columns
         newben_df = ben_df.drop(["BENE_HI_CVRAGE_TOT_MONS", "BENE_SMI_CVRAGE_TOT_MONS", "BENE_HMO_CVRAGE_TOT_MONS", "PLAN_CVRG_MOS_NU
                                  "MEDREIMB_IP", "BENRES_IP", "PPPYMT_IP", "MEDREIMB_OP", "BENRES_OP", "PPPYMT_OP", "MEDREIMB_CAR",
                                  "BENRES_CAR", "PPPYMT_CAR", "BENE_ESRD_IND", "SP_ALZHDMTA", "SP_CHF",
                                 "SP_CHRNKIDN", "SP_CNCR", "SP_COPD", "SP_DEPRESSN", "SP_DIABETES", "SP_ISCHMCHT", "SP_OSTEOPRS", "SP_
                                  "SP_STRKETIA"], axis=1)
         newben_df.head()
```



```
In [8]:  # replace members sex code with male, female, or unknown (https://www.resdac.org/cms-data/variables/sex)
         newben_df['BENE_SEX_IDENT_CD'].mask(newben_df['BENE_SEX_IDENT_CD'] == 1, 'Male', inplace=True)
         newben_df['BENE_SEX_IDENT_CD'].mask(newben_df['BENE_SEX_IDENT_CD'] == 2, 'Female', inplace=True)
         newben_df['BENE_SEX_IDENT_CD'].mask(newben_df['BENE_SEX_IDENT_CD'] == 0, 'Unknown', inplace=True)
         newben_df.head()

In [9]:  # replace memebers race code with unknown, white, black, other, asian, hispanic, north american native (https://www.resdac.or
         newben_df['BENE_RACE_CD'].mask(newben_df['BENE_RACE_CD'] == 0, 'Unknown', inplace=True)
         newben_df['BENE_RACE_CD'].mask(newben_df['BENE_RACE_CD'] == 1, 'White', inplace=True)
         newben_df['BENE_RACE_CD'].mask(newben_df['BENE_RACE_CD'] == 2, 'Black', inplace=True)
         newben_df['BENE_RACE_CD'].mask(newben_df['BENE_RACE_CD'] == 3, 'Other', inplace=True)
         newben_df['BENE_RACE_CD'].mask(newben_df['BENE_RACE_CD'] == 4, 'Asian', inplace=True)
         newben_df['BENE_RACE_CD'].mask(newben_df['BENE_RACE_CD'] == 5, 'Hispanic', inplace=True)
         newben_df['BENE_RACE_CD'].mask(newben_df['BENE_RACE_CD'] == 6, 'North American Native', inplace=True)
         newben_df.head()

In [10]: # replace state and county code with state and county name: merge state code and county code to get ssa number
         newben_df["state_county_code"] = newben_df["SP_STATE_CODE"].map(str) + newben_df["BENE_COUNTY_CD"].map(str)
         newben_df.head()

In [11]: # drop unwanted columns
         newben_df = newben_df.drop(["SP_STATE_CODE", "BENE_COUNTY_CD"], axis=1)
         newben_df

In [12]: # import countycode file and read it (this file has has the state and county which correlates to their ssa code)
         statecountycode = "State_County.csv"
         statecountycode_df = pd.read_csv(statecountycode)
         statecountycode_df.head()

In [13]: # convert the state_county_code from the statecountycode_df to a string so we can merge it with the state_county_code from ne
         statecountycode_df["SSA"] = statecountycode_df["SSA"].astype(str)
         statecountycode_df.head()

In [14]: # merge newben_df with statecountycode_df on state_county_code
         newben_df = pd.merge(newben_df, statecountycode_df, how='inner', left_on=['state_county_code'], right_on = ['SSA'])
         newben_df.head()

In [15]: # remove the state_county_code column
         newben_df = newben_df.drop(["state_county_code"], axis=1)
         newben_df.head()
         #116352 rows × 32 columns to 68810 rows × 10 columns

In [16]: # write to csv
         newben_df.to_csv("NewBen.csv", index=False, header=True)
```

Data Clean Up



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

Read and clean original inpatient claims file

```
In [2]:  # import and read inpatient file
         inpt_org = "DE1_0_2008_to_2010_Inpatient_Claims_Sample_1.csv"
         inpt_df = pd.read_csv(inpt_org)
```

```
In [3]:  # only include data from 2008
         newinpt_df = inpt_df[inpt_df.CLM_FROM_DT <20090101]
         newinpt_df.head()
```

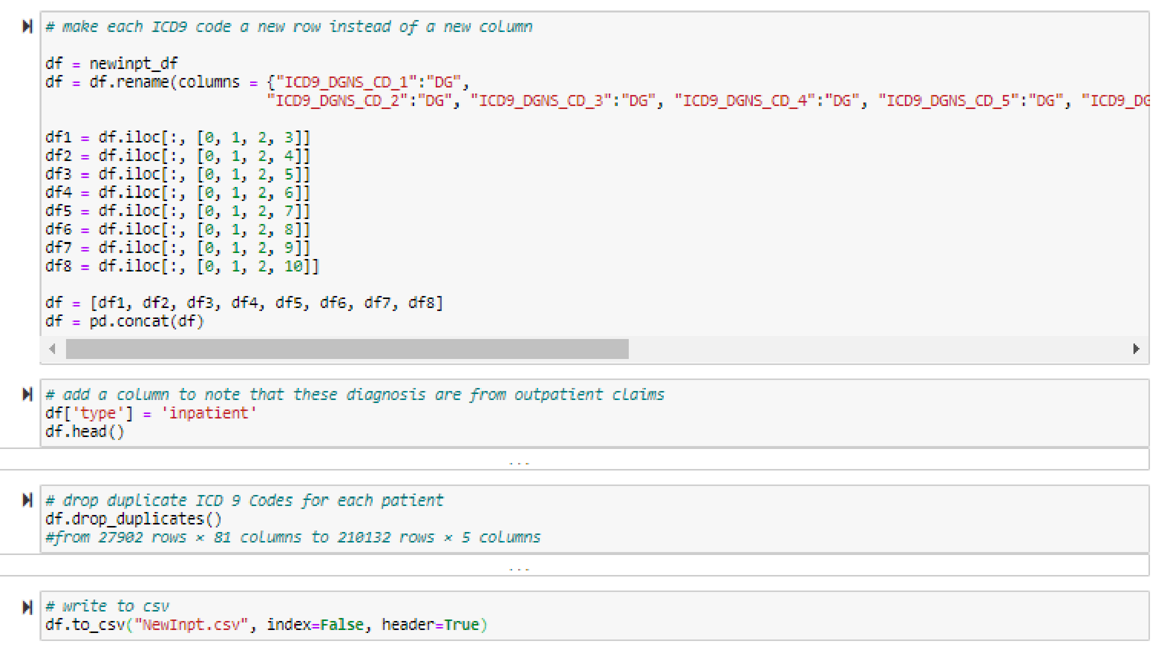
Out[3]:

	DESYNPUF_ID	CLM_ID	SEGMENT	CLM_FROM_DT	CLM_THRU_DT	PRVDR_NUM	CLM_PMT_AMT	NCH_PRMRY_PYR_CLM_PD_AMT	A
5	00052705243EA128	196991176971757	1	20080912.0	20080912.0	1401HG	14000.0	0.0	
6	0007F12A492FD25D	196661176963773	1	20080919.0	20080922.0	3400WD	5000.0	0.0	
11	000C7486B11E7030	196641176984178	1	20081015.0	20081021.0	4400MM	30000.0	0.0	
14	0011CB1FE23E91AF	196851176958774	1	20080421.0	20080426.0	2000HG	3000.0	0.0	
15	0011CB1FE23E91AF	196091176985832	1	20080426.0	20080430.0	2013RT	8000.0	0.0	

5 rows × 81 columns

```
In [4]:  # remove unwanted columns
         newinpt_df = newinpt_df.drop(["SEGMENT", "CLM_FROM_DT", "CLM_THRU_DT", "PRVDR_NUM", "CLM_PMT_AMT",
                                       "AT_PHYSN_NPI", "OP_PHYSN_NPI", "OT_PHYSN_NPI", "CLM_ADMSN_DT", "CLM_PASS_THRU_PER_DIEM_AMT",
                                       "NCH_BENE_IP_DDCTBL_AMT", "NCH_BENE_PTA_COINSRNC_LBLTY_AM", "NCH_BENE_BLOOD_DDCTBL_LBLTY_AM",
                                       "NCH_PRMRY_PYR_CLM_PD_AMT", "CLM_UTLZTN_DAY_CNT", "NCH_BENE_DSCHRG_DT", "CLM_DRG_CD",
                                       "HCPCS_CD_1", "HCPCS_CD_2", "HCPCS_CD_3", "HCPCS_CD_4", "HCPCS_CD_5",
                                       "HCPCS_CD_6", "HCPCS_CD_7", "HCPCS_CD_8", "HCPCS_CD_9", "HCPCS_CD_10",
                                       "HCPCS_CD_11", "HCPCS_CD_12", "HCPCS_CD_13", "HCPCS_CD_14", "HCPCS_CD_15",
                                       "HCPCS_CD_16", "HCPCS_CD_17", "HCPCS_CD_18", "HCPCS_CD_19", "HCPCS_CD_20",
                                       "HCPCS_CD_21", "HCPCS_CD_22", "HCPCS_CD_23", "HCPCS_CD_24", "HCPCS_CD_25",
                                       "HCPCS_CD_26", "HCPCS_CD_27", "HCPCS_CD_28", "HCPCS_CD_29", "HCPCS_CD_30",
                                       "HCPCS_CD_31", "HCPCS_CD_32", "HCPCS_CD_33", "HCPCS_CD_34", "HCPCS_CD_35",
                                       "HCPCS_CD_36", "HCPCS_CD_37", "HCPCS_CD_38", "HCPCS_CD_39", "HCPCS_CD_40",
                                       "HCPCS_CD_41", "HCPCS_CD_42", "HCPCS_CD_43", "HCPCS_CD_44", "HCPCS_CD_45"], axis=1)
         newinpt_df.head()
```

...



```
# make each ICD9 code a new row instead of a new column

df = newinpt_df
df = df.rename(columns = {"ICD9_DGNS_CD_1":"DG",
                          "ICD9_DGNS_CD_2":"DG", "ICD9_DGNS_CD_3":"DG", "ICD9_DGNS_CD_4":"DG", "ICD9_DGNS_CD_5":"DG", "ICD9_DG

df1 = df.iloc[:, [0, 1, 2, 3]]
df2 = df.iloc[:, [0, 1, 2, 4]]
df3 = df.iloc[:, [0, 1, 2, 5]]
df4 = df.iloc[:, [0, 1, 2, 6]]
df5 = df.iloc[:, [0, 1, 2, 7]]
df6 = df.iloc[:, [0, 1, 2, 8]]
df7 = df.iloc[:, [0, 1, 2, 9]]
df8 = df.iloc[:, [0, 1, 2, 10]]

df = [df1, df2, df3, df4, df5, df6, df7, df8]
df = pd.concat(df)
```

```
# add a column to note that these diagnosis are from outpatient claims
df['type'] = 'inpatient'
df.head()
```

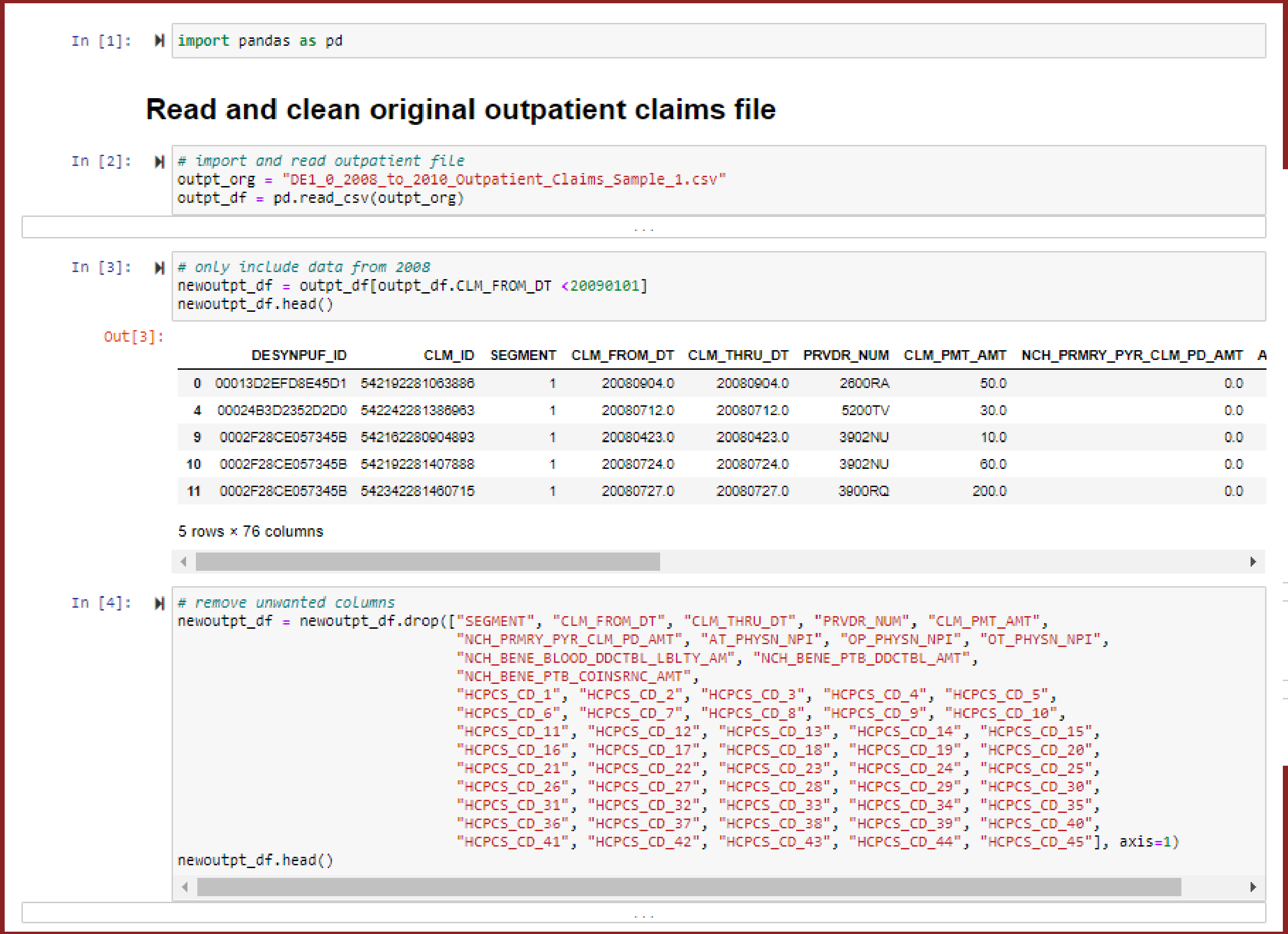
...

```
# drop duplicate ICD 9 Codes for each patient
df.drop_duplicates()
#from 27902 rows × 81 columns to 210132 rows × 5 columns
```

...

```
# write to csv
df.to_csv("NewInpt.csv", index=False, header=True)
```

Data Clean Up



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

Read and clean original outpatient claims file

```
In [2]:  # import and read outpatient file
         outpt_org = "DE1_0_2008_to_2010_Outpatient_Claims_Sample_1.csv"
         outpt_df = pd.read_csv(outpt_org)
```

...

```
In [3]:  # only include data from 2008
         newoutpt_df = outpt_df[outpt_df.CLM_FROM_DT <20090101]
         newoutpt_df.head()
```

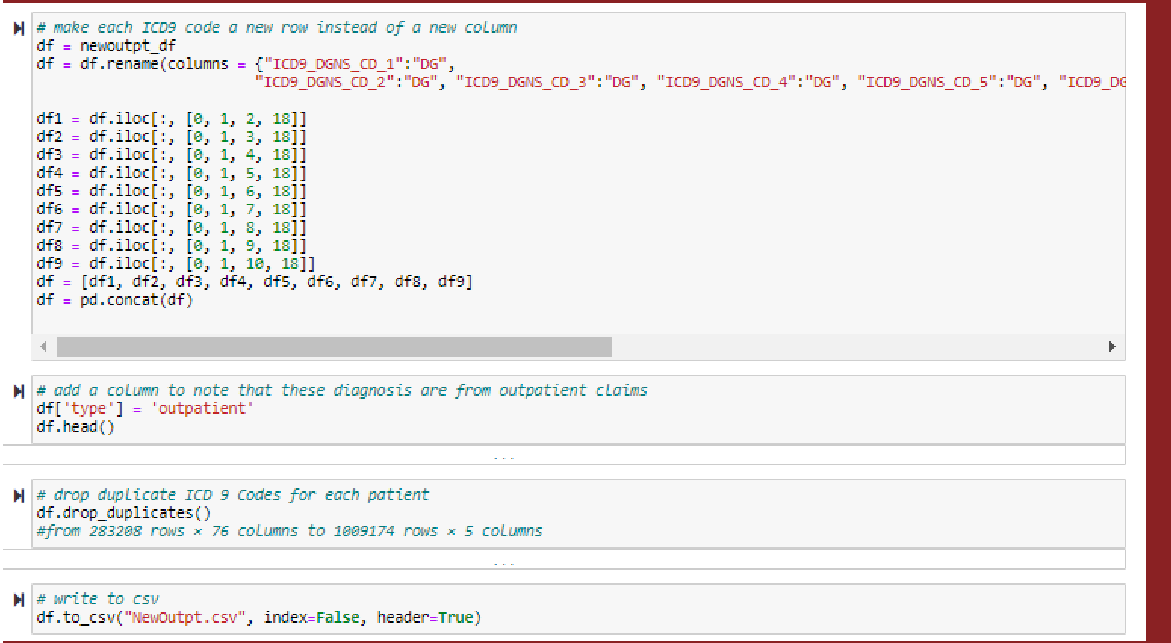
Out[3]:

	DESYNPUF_ID	CLM_ID	SEGMENT	CLM_FROM_DT	CLM_THRU_DT	PRVDR_NUM	CLM_PMT_AMT	NCH_PRMRY_PYR_CLM_PD_AMT	A
0	00013D2EFD8E45D1	542192281063886	1	20080904.0	20080904.0	2600RA	50.0	0.0	
4	00024B3D2352D2D0	542242281386963	1	20080712.0	20080712.0	5200TV	30.0	0.0	
9	0002F28CE057345B	542162280904893	1	20080423.0	20080423.0	3902NU	10.0	0.0	
10	0002F28CE057345B	542192281407888	1	20080724.0	20080724.0	3902NU	60.0	0.0	
11	0002F28CE057345B	542342281460715	1	20080727.0	20080727.0	3900RQ	200.0	0.0	

5 rows × 76 columns

```
In [4]:  # remove unwanted columns
         newoutpt_df = newoutpt_df.drop(["SEGMENT", "CLM_FROM_DT", "CLM_THRU_DT", "PRVDR_NUM", "CLM_PMT_AMT",
                                         "NCH_PRMRY_PYR_CLM_PD_AMT", "AT_PHYSN_NPI", "OP_PHYSN_NPI", "OT_PHYSN_NPI",
                                         "NCH_BENE_BLOOD_DDCTBL_LBLTY_AM", "NCH_BENE_PTB_DDCTBL_AMT",
                                         "NCH_BENE_PTB_COINSRNC_AMT",
                                         "HCPCS_CD_1", "HCPCS_CD_2", "HCPCS_CD_3", "HCPCS_CD_4", "HCPCS_CD_5",
                                         "HCPCS_CD_6", "HCPCS_CD_7", "HCPCS_CD_8", "HCPCS_CD_9", "HCPCS_CD_10",
                                         "HCPCS_CD_11", "HCPCS_CD_12", "HCPCS_CD_13", "HCPCS_CD_14", "HCPCS_CD_15",
                                         "HCPCS_CD_16", "HCPCS_CD_17", "HCPCS_CD_18", "HCPCS_CD_19", "HCPCS_CD_20",
                                         "HCPCS_CD_21", "HCPCS_CD_22", "HCPCS_CD_23", "HCPCS_CD_24", "HCPCS_CD_25",
                                         "HCPCS_CD_26", "HCPCS_CD_27", "HCPCS_CD_28", "HCPCS_CD_29", "HCPCS_CD_30",
                                         "HCPCS_CD_31", "HCPCS_CD_32", "HCPCS_CD_33", "HCPCS_CD_34", "HCPCS_CD_35",
                                         "HCPCS_CD_36", "HCPCS_CD_37", "HCPCS_CD_38", "HCPCS_CD_39", "HCPCS_CD_40",
                                         "HCPCS_CD_41", "HCPCS_CD_42", "HCPCS_CD_43", "HCPCS_CD_44", "HCPCS_CD_45"], axis=1)

         newoutpt_df.head()
```

...



```
# make each ICD9 code a new row instead of a new column
df = newoutpt_df
df = df.rename(columns = {"ICD9_DGNS_CD_1":"DG",
                          "ICD9_DGNS_CD_2":"DG", "ICD9_DGNS_CD_3":"DG", "ICD9_DGNS_CD_4":"DG", "ICD9_DGNS_CD_5":"DG", "ICD9_DG

df1 = df.iloc[:, [0, 1, 2, 18]]
df2 = df.iloc[:, [0, 1, 3, 18]]
df3 = df.iloc[:, [0, 1, 4, 18]]
df4 = df.iloc[:, [0, 1, 5, 18]]
df5 = df.iloc[:, [0, 1, 6, 18]]
df6 = df.iloc[:, [0, 1, 7, 18]]
df7 = df.iloc[:, [0, 1, 8, 18]]
df8 = df.iloc[:, [0, 1, 9, 18]]
df9 = df.iloc[:, [0, 1, 10, 18]]
df = [df1, df2, df3, df4, df5, df6, df7, df8, df9]
df = pd.concat(df)
```

```
# add a column to note that these diagnosis are from outpatient claims
df['type'] = 'outpatient'
df.head()
```

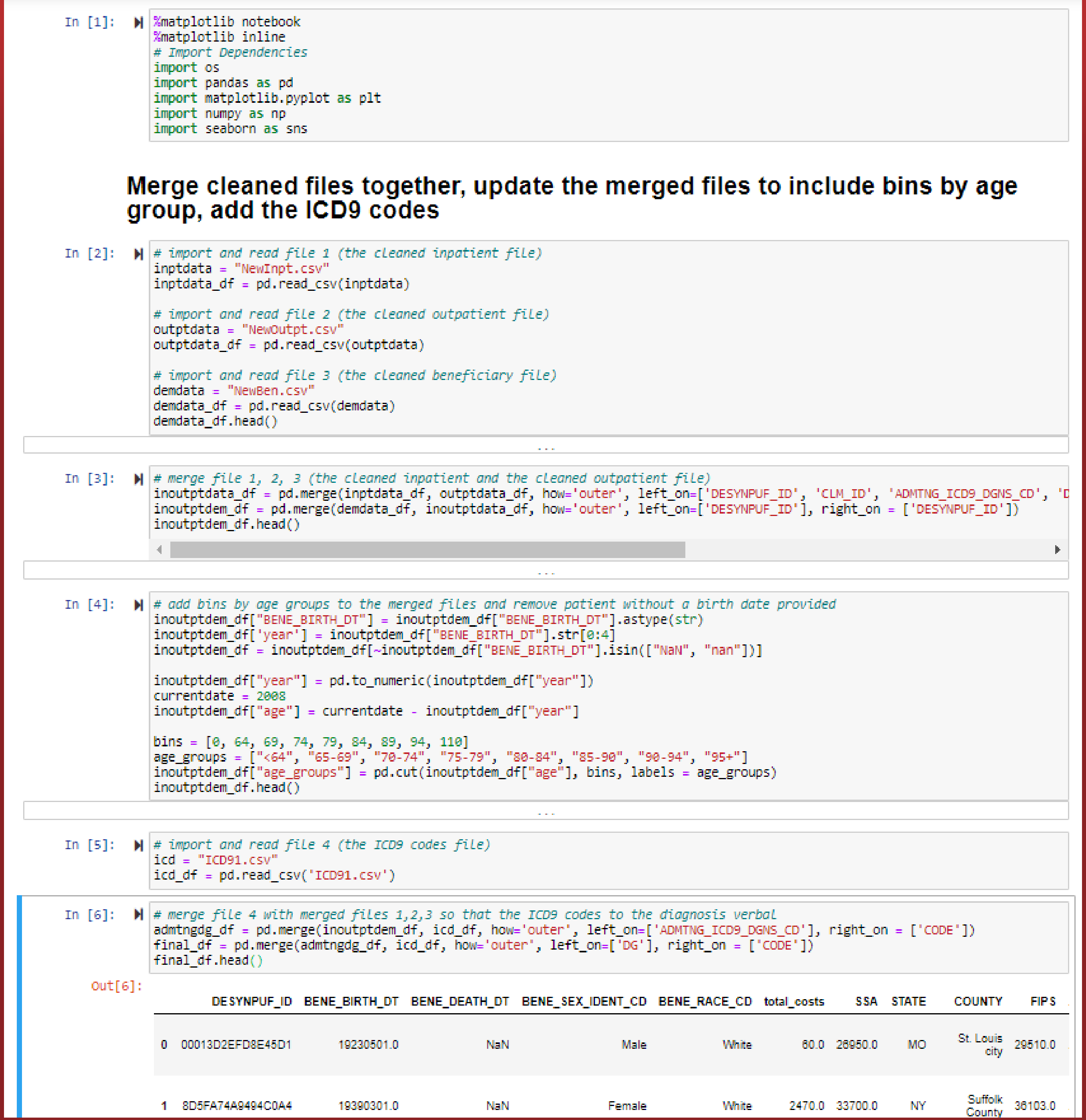
...

```
# drop duplicate ICD 9 Codes for each patient
df.drop_duplicates()
#from 283208 rows × 76 columns to 1009174 rows × 5 columns
```

...

```
# write to csv
df.to_csv("NewOutpt.csv", index=False, header=True)
```

Data Clean Up



```
In [1]:
%matplotlib notebook
%matplotlib inline
# Import Dependencies
import os
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

Merge cleaned files together, update the merged files to include bins by age group, add the ICD9 codes

```
In [2]:
# import and read file 1 (the cleaned inpatient file)
inptdata = "NewInpt.csv"
inptdata_df = pd.read_csv(inptdata)

# import and read file 2 (the cleaned outpatient file)
outptdata = "NewOutpt.csv"
outptdata_df = pd.read_csv(outptdata)

# import and read file 3 (the cleaned beneficiary file)
demdata = "NewBen.csv"
demdata_df = pd.read_csv(demdata)
demdata_df.head()
```

```
In [3]:
# merge file 1, 2, 3 (the cleaned inpatient and the cleaned outpatient file)
inoutptdata_df = pd.merge(inptdata_df, outptdata_df, how='outer', left_on=['DESYNPUF_ID', 'CLM_ID', 'ADMTNG_ICD9_DGNS_CD', 'D
inoutptdem_df = pd.merge(demdata_df, inoutptdata_df, how='outer', left_on=['DESYNPUF_ID'], right_on = ['DESYNPUF_ID'])
inoutptdem_df.head()
```

```
In [4]:
# add bins by age groups to the merged files and remove patient without a birth date provided
inoutptdem_df["BENE_BIRTH_DT"] = inoutptdem_df["BENE_BIRTH_DT"].astype(str)
inoutptdem_df['year'] = inoutptdem_df["BENE_BIRTH_DT"].str[0:4]
inoutptdem_df = inoutptdem_df[~inoutptdem_df["BENE_BIRTH_DT"].isin(["NaN", "nan"])]

inoutptdem_df["year"] = pd.to_numeric(inoutptdem_df["year"])
currentdate = 2008
inoutptdem_df["age"] = currentdate - inoutptdem_df["year"]

bins = [0, 64, 69, 74, 79, 84, 89, 94, 110]
age_groups = ["<64", "65-69", "70-74", "75-79", "80-84", "85-90", "90-94", "95+"]
inoutptdem_df["age_groups"] = pd.cut(inoutptdem_df["age"], bins, labels = age_groups)
inoutptdem_df.head()
```

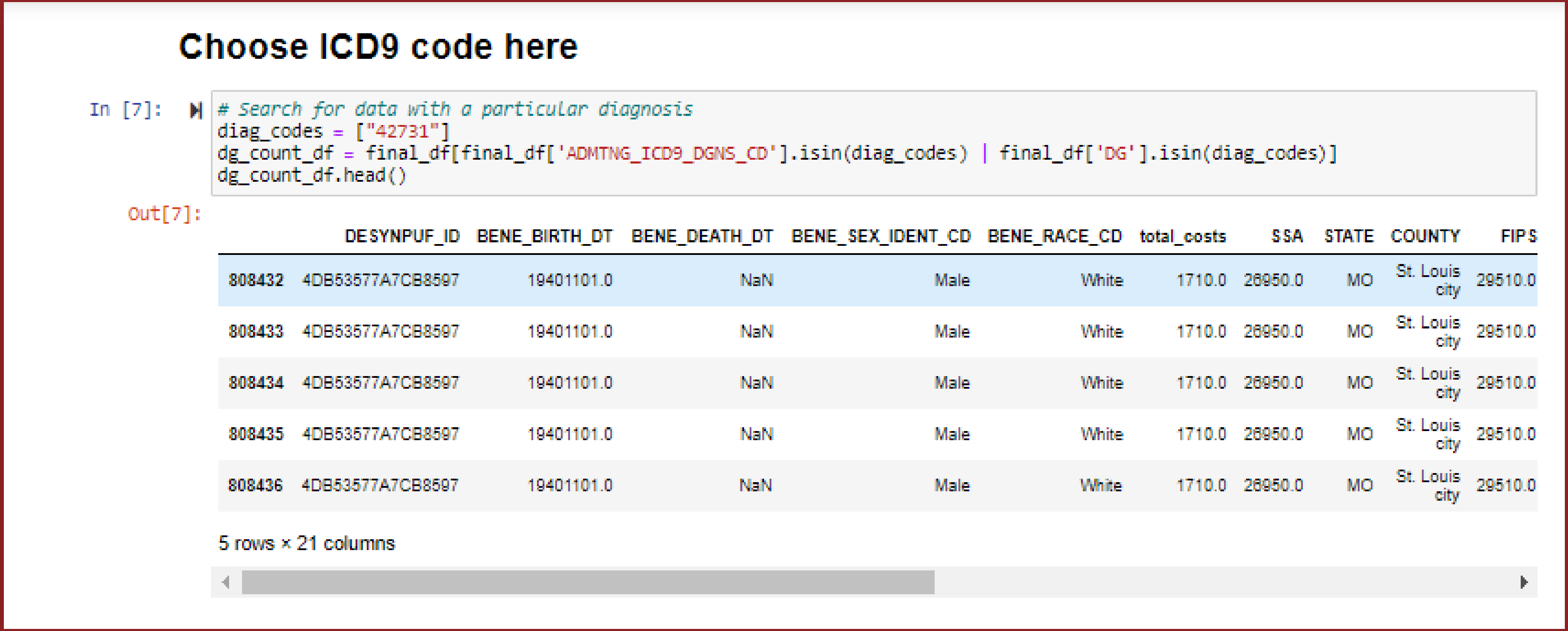
```
In [5]:
# import and read file 4 (the ICD9 codes file)
icd = "ICD91.csv"
icd_df = pd.read_csv('ICD91.csv')
```

```
In [6]:
# merge file 4 with merged files 1,2,3 so that the ICD9 codes to the diagnosis verbal
admtngdg_df = pd.merge(inoutptdem_df, icd_df, how='outer', left_on=['ADMTNG_ICD9_DGNS_CD'], right_on = ['CODE'])
final_df = pd.merge(admtngdg_df, icd_df, how='outer', left_on=['DG'], right_on = ['CODE'])
final_df.head()
```

Out[6]:

	DESYNPUF_ID	BENE_BIRTH_DT	BENE_DEATH_DT	BENE_SEX_IDENT_CD	BENE_RACE_CD	total_costs	SSA	STATE	COUNTY	FIPS
0	00013D2EFD8E45D1	19230501.0	NaN	Male	White	60.0	26950.0	MO	St. Louis city	29510.0
1	8D5FA74A9494C0A4	19390301.0	NaN	Female	White	2470.0	33700.0	NY	Suffolk County	36103.0

Analysis



Choose ICD9 code here

In [7]:

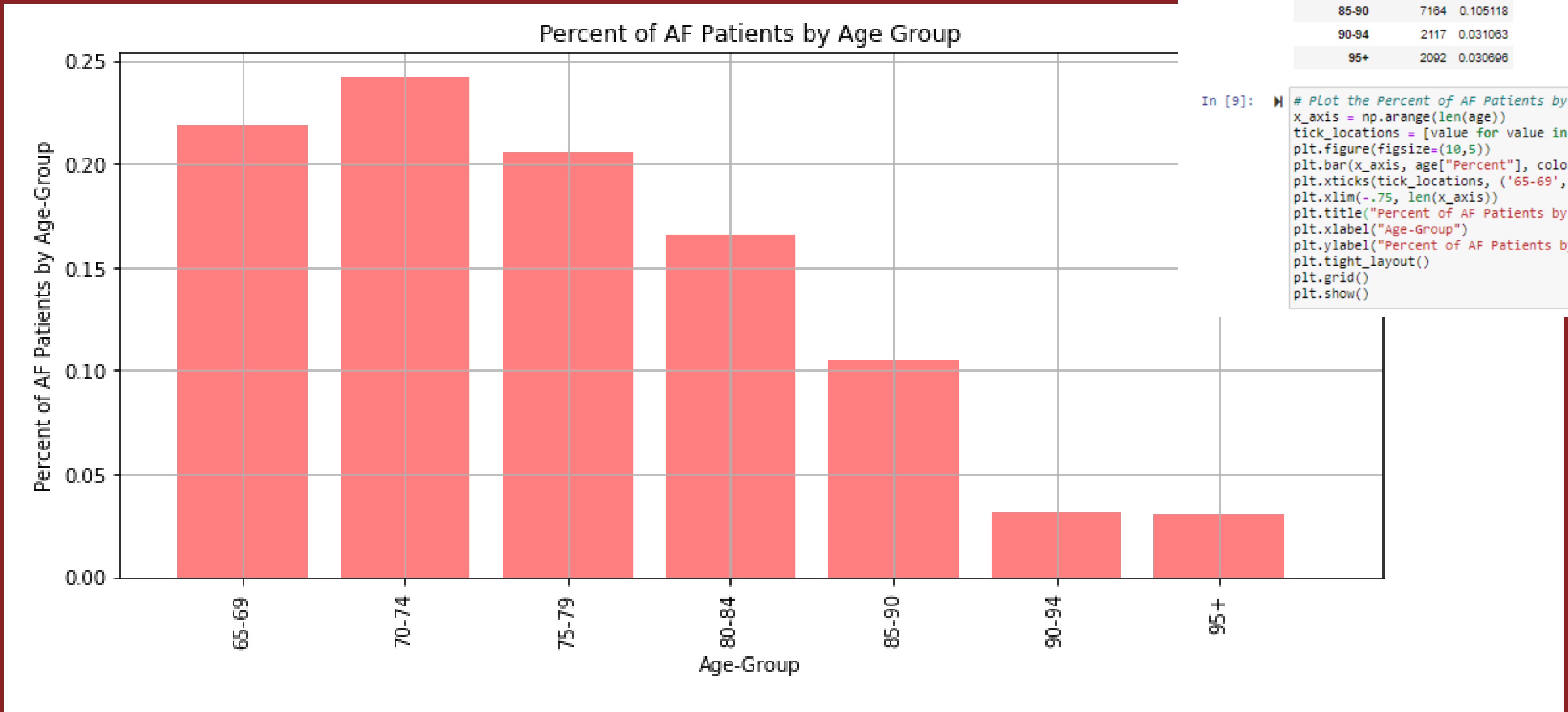
```
# Search for data with a particular diagnosis
diag_codes = ["42731"]
dg_count_df = final_df[final_df['ADMTNG_ICD9_DGNS_CD'].isin(diag_codes) | final_df['DG'].isin(diag_codes)]
dg_count_df.head()
```

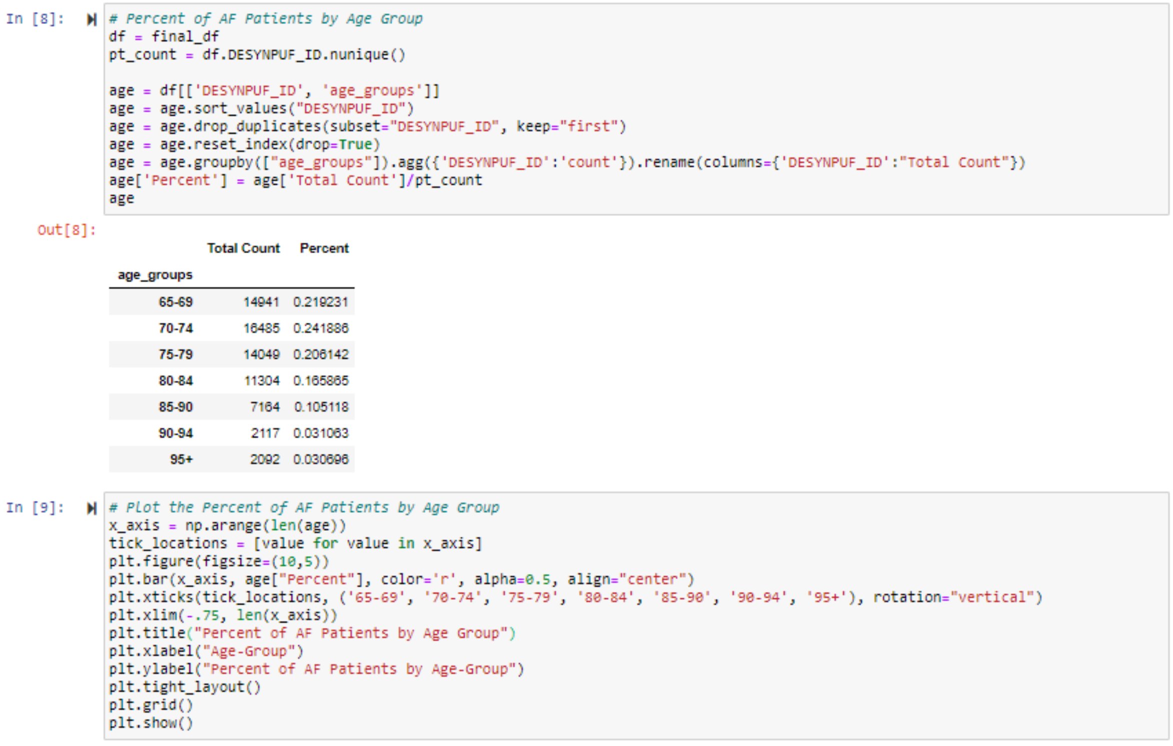
Out[7]:

	DESYNPUF_ID	BENE_BIRTH_DT	BENE_DEATH_DT	BENE_SEX_IDENT_CD	BENE_RACE_CD	total_costs	SSA	STATE	COUNTY	FIPS
808432	4DB53577A7CB8597	19401101.0	NaN	Male	White	1710.0	26950.0	MO	St. Louis city	29510.0
808433	4DB53577A7CB8597	19401101.0	NaN	Male	White	1710.0	26950.0	MO	St. Louis city	29510.0
808434	4DB53577A7CB8597	19401101.0	NaN	Male	White	1710.0	26950.0	MO	St. Louis city	29510.0
808435	4DB53577A7CB8597	19401101.0	NaN	Male	White	1710.0	26950.0	MO	St. Louis city	29510.0
808436	4DB53577A7CB8597	19401101.0	NaN	Male	White	1710.0	26950.0	MO	St. Louis city	29510.0

5 rows × 21 columns

Analysis:
Percent of AF Patients by Age Group





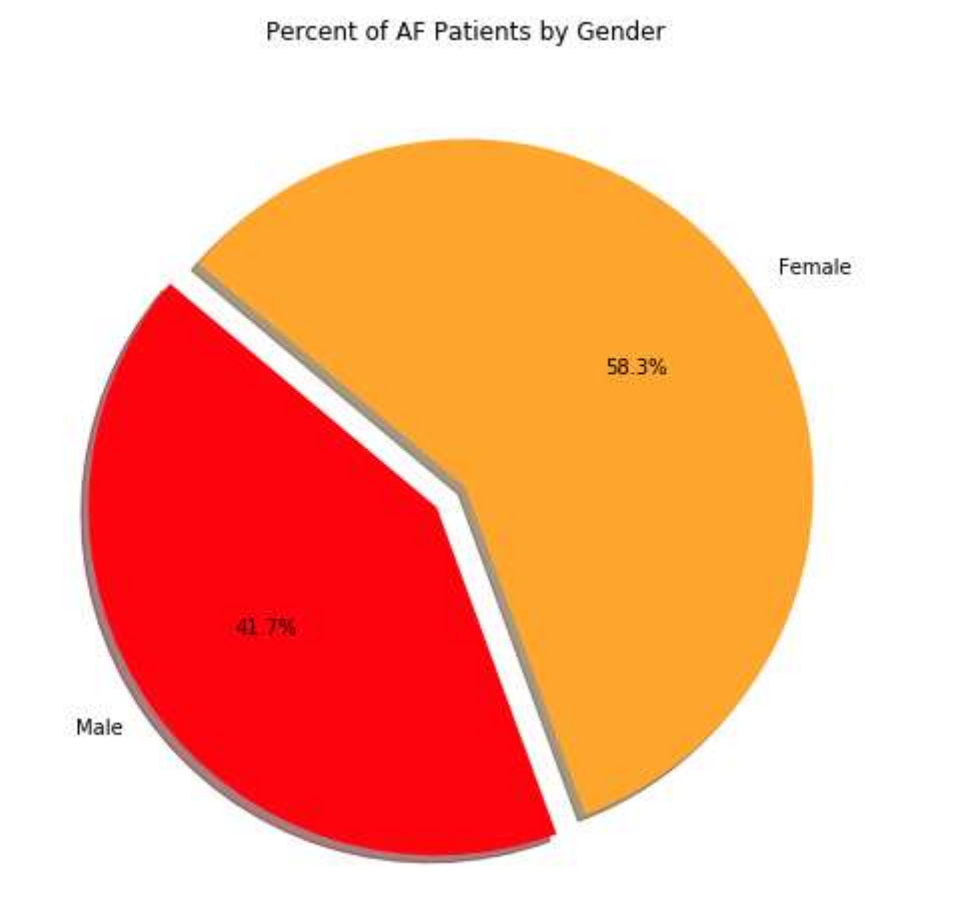
```
In [8]:  # Percent of AF Patients by Age Group
         df = final_df
         pt_count = df.DESYNPUF_ID.nunique()

         age = df[['DESYNPUF_ID', 'age_groups']]
         age = age.sort_values("DESYNPUF_ID")
         age = age.drop_duplicates(subset="DESYNPUF_ID", keep="first")
         age = age.reset_index(drop=True)
         age = age.groupby(["age_groups"]).agg({'DESYNPUF_ID':'count'}).rename(columns={'DESYNPUF_ID':"Total Count"})
         age['Percent'] = age['Total Count']/pt_count
         age
```

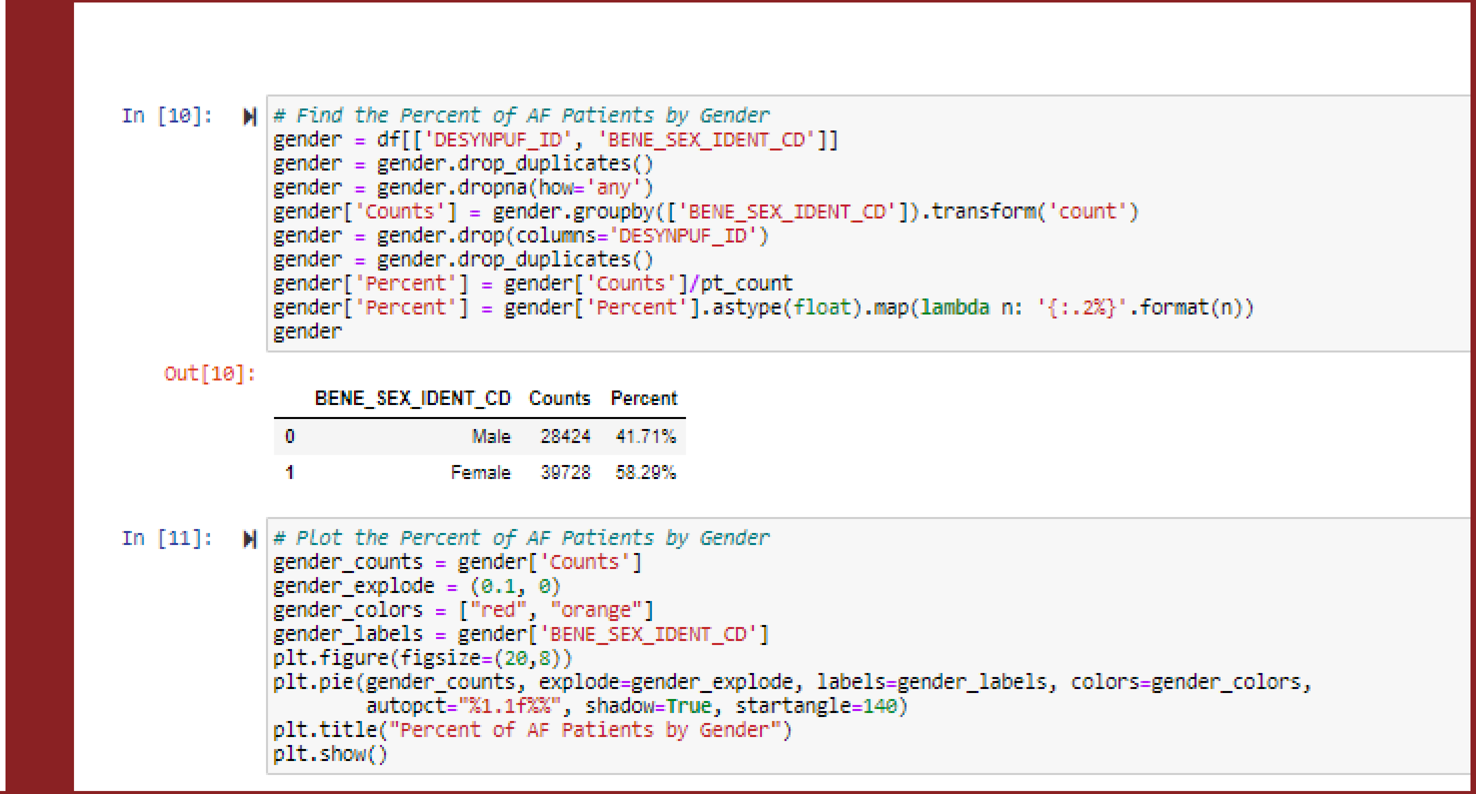
```
Out[8]:
                    Total Count   Percent
         age_groups
              65-69       14941  0.219231
              70-74       16485  0.241886
              75-79       14049  0.206142
              80-84       11304  0.165865
              85-90        7164  0.105118
              90-94        2117  0.031063
                95+        2092  0.030696
```

```
In [9]:  # Plot the Percent of AF Patients by Age Group
         x_axis = np.arange(len(age))
         tick_locations = [value for value in x_axis]
         plt.figure(figsize=(10,5))
         plt.bar(x_axis, age["Percent"], color='r', alpha=0.5, align="center")
         plt.xticks(tick_locations, ('65-69', '70-74', '75-79', '80-84', '85-90', '90-94', '95+'), rotation="vertical")
         plt.xlim(-.75, len(x_axis))
         plt.title("Percent of AF Patients by Age Group")
         plt.xlabel("Age-Group")
         plt.ylabel("Percent of AF Patients by Age-Group")
         plt.tight_layout()
         plt.grid()
         plt.show()
```

The largest percent of patients with AF are those between the ages of 70-74 years old.

Analysis:
Percent of AF Patients by Gender





```
In [10]:  # Find the Percent of AF Patients by Gender
          gender = df[['DESYNPUF_ID', 'BENE_SEX_IDENT_CD']]
          gender = gender.drop_duplicates()
          gender = gender.dropna(how='any')
          gender['Counts'] = gender.groupby(['BENE_SEX_IDENT_CD']).transform('count')
          gender = gender.drop(columns='DESYNPUF_ID')
          gender = gender.drop_duplicates()
          gender['Percent'] = gender['Counts']/pt_count
          gender['Percent'] = gender['Percent'].astype(float).map(lambda n: '{:.2%}'.format(n))
          gender
```

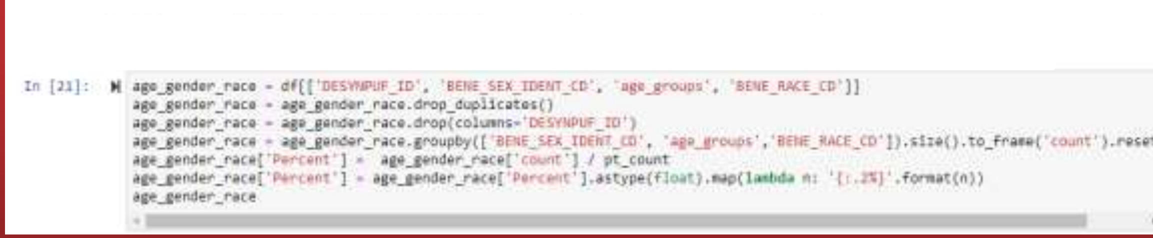
Out[10]:

	BENE_SEX_IDENT_CD	Counts	Percent
0	Male	28424	41.71%
1	Female	39728	58.29%

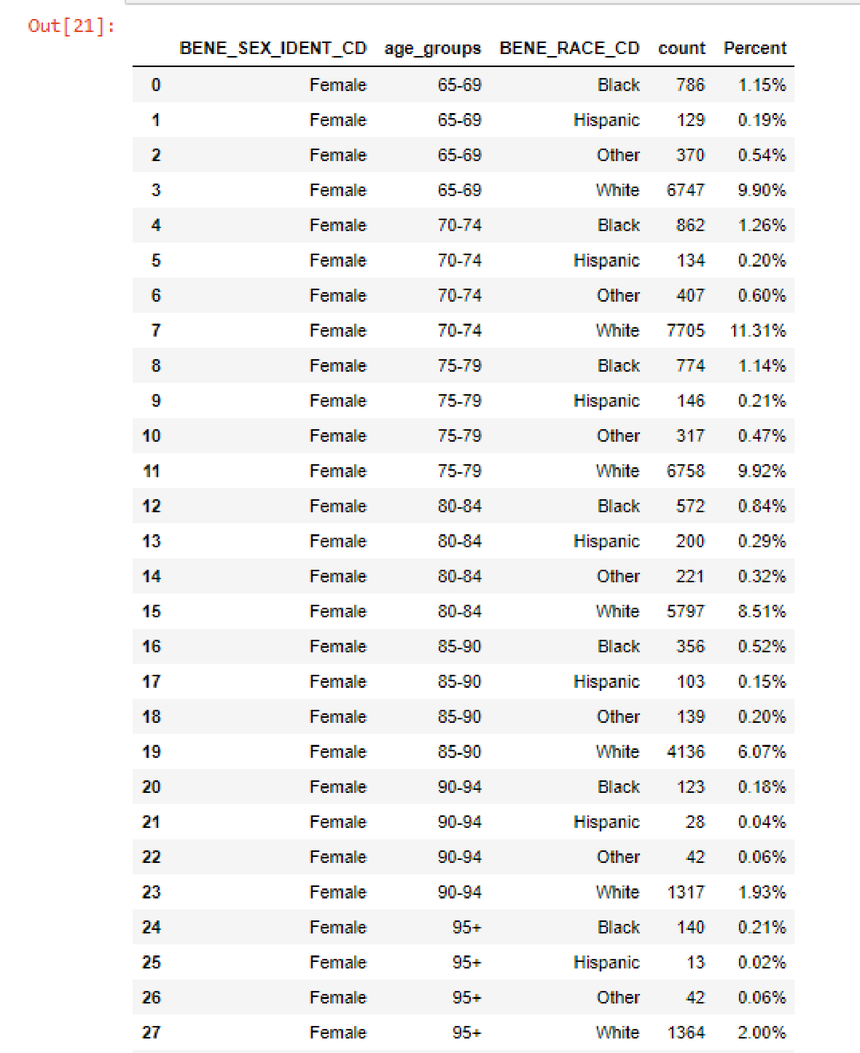
```
In [11]:  # Plot the Percent of AF Patients by Gender
          gender_counts = gender['Counts']
          gender_explode = (0.1, 0)
          gender_colors = ["red", "orange"]
          gender_labels = gender['BENE_SEX_IDENT_CD']
          plt.figure(figsize=(20,8))
          plt.pie(gender_counts, explode=gender_explode, labels=gender_labels, colors=gender_colors,
                  autopct="%1.1f%%", shadow=True, startangle=140)
          plt.title("Percent of AF Patients by Gender")
          plt.show()
```

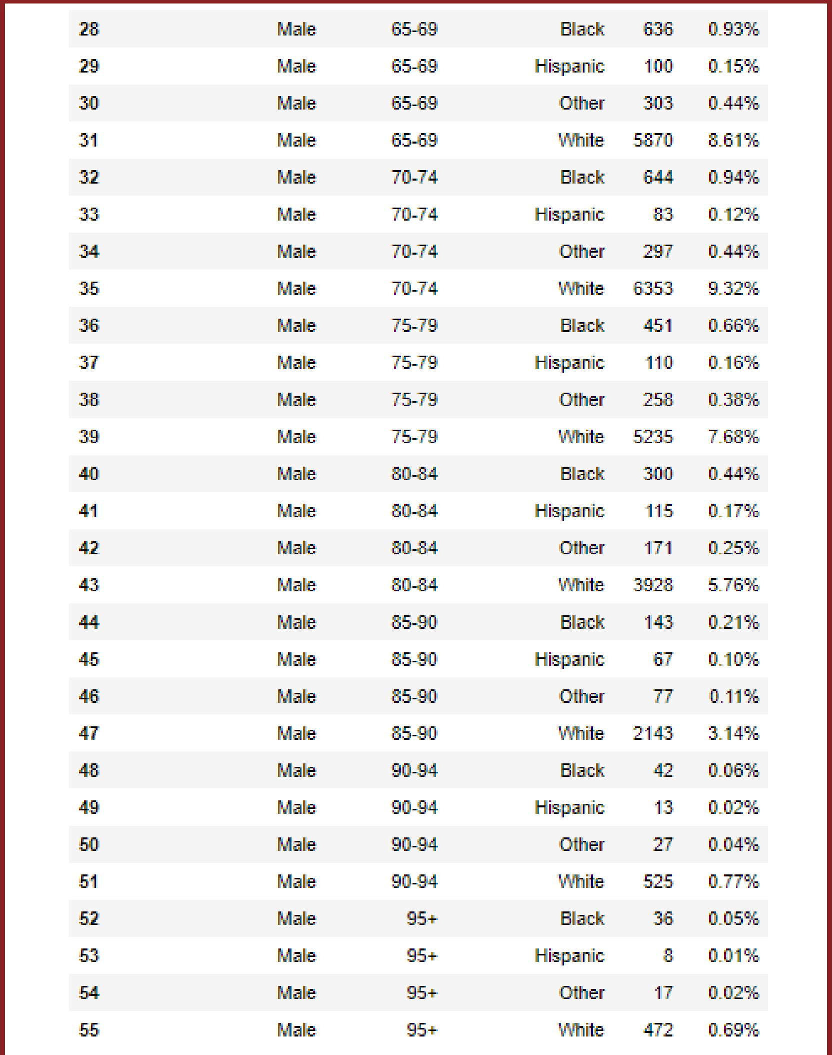
More females than males have AF.

Analysis:
Percent of AF Patients
by Gender, Age-Group, & Race



```
In [21]:  age_gender_race = df[['DESYNPUF_ID', 'BENE_SEX_IDENT_CD', 'age_groups', 'BENE_RACE_CD']]
          age_gender_race = age_gender_race.drop_duplicates()
          age_gender_race = age_gender_race.drop(columns='DESYNPUF_ID')
          age_gender_race = age_gender_race.groupby(['BENE_SEX_IDENT_CD', 'age_groups','BENE_RACE_CD']).size().to_frame('count').rese
          age_gender_race['Percent'] =  age_gender_race['count'] / pt_count
          age_gender_race['Percent'] = age_gender_race['Percent'].astype(float).map(lambda n: '{:.2%}'.format(n))
          age_gender_race
```



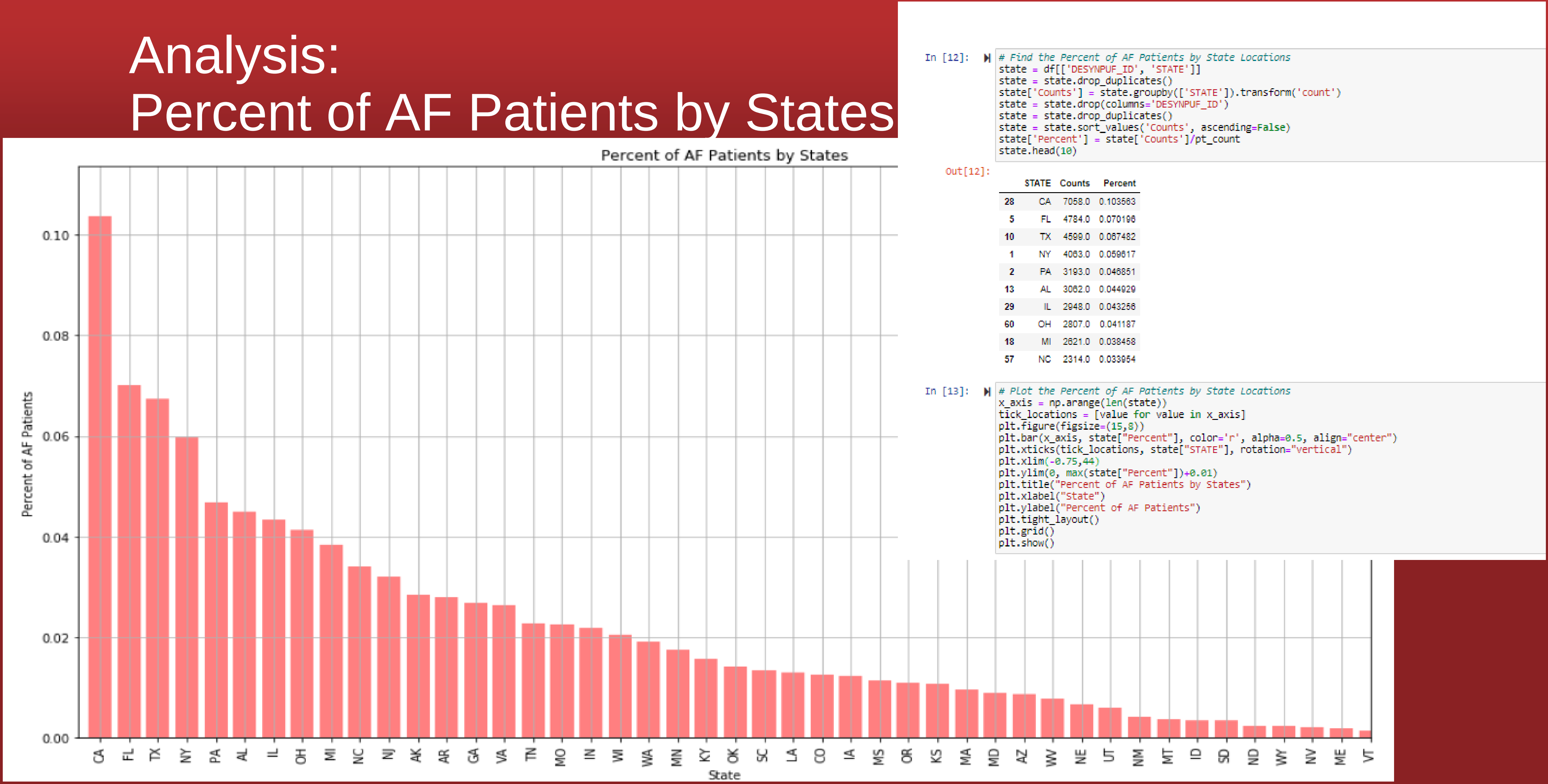


Out[21]:

	BENE_SEX_IDENT_CD	age_groups	BENE_RACE_CD	count	Percent
0	Female	65-69	Black	786	1.15%
1	Female	65-69	Hispanic	129	0.19%
2	Female	65-69	Other	370	0.54%
3	Female	65-69	White	6747	9.90%
4	Female	70-74	Black	862	1.26%
5	Female	70-74	Hispanic	134	0.20%
6	Female	70-74	Other	407	0.60%
7	Female	70-74	White	7705	11.31%
8	Female	75-79	Black	774	1.14%
9	Female	75-79	Hispanic	146	0.21%
10	Female	75-79	Other	317	0.47%
11	Female	75-79	White	6758	9.92%
12	Female	80-84	Black	572	0.84%
13	Female	80-84	Hispanic	200	0.29%
14	Female	80-84	Other	221	0.32%
15	Female	80-84	White	5797	8.51%
16	Female	85-90	Black	356	0.52%
17	Female	85-90	Hispanic	103	0.15%
18	Female	85-90	Other	139	0.20%
19	Female	85-90	White	4136	6.07%
20	Female	90-94	Black	123	0.18%
21	Female	90-94	Hispanic	28	0.04%
22	Female	90-94	Other	42	0.06%
23	Female	90-94	White	1317	1.93%
24	Female	95+	Black	140	0.21%
25	Female	95+	Hispanic	13	0.02%
26	Female	95+	Other	42	0.06%
27	Female	95+	White	1364	2.00%
28	Male	65-69	Black	636	0.93%
29	Male	65-69	Hispanic	100	0.15%
30	Male	65-69	Other	303	0.44%
31	Male	65-69	White	5870	8.61%
32	Male	70-74	Black	644	0.94%
33	Male	70-74	Hispanic	83	0.12%
34	Male	70-74	Other	297	0.44%
35	Male	70-74	White	6353	9.32%
36	Male	75-79	Black	451	0.66%
37	Male	75-79	Hispanic	110	0.16%
38	Male	75-79	Other	258	0.38%
39	Male	75-79	White	5235	7.68%
40	Male	80-84	Black	300	0.44%
41	Male	80-84	Hispanic	115	0.17%
42	Male	80-84	Other	171	0.25%
43	Male	80-84	White	3928	5.76%
44	Male	85-90	Black	143	0.21%
45	Male	85-90	Hispanic	67	0.10%
46	Male	85-90	Other	77	0.11%
47	Male	85-90	White	2143	3.14%
48	Male	90-94	Black	42	0.06%
49	Male	90-94	Hispanic	13	0.02%
50	Male	90-94	Other	27	0.04%
51	Male	90-94	White	525	0.77%
52	Male	95+	Black	36	0.05%
53	Male	95+	Hispanic	8	0.01%
54	Male	95+	Other	17	0.02%
55	Male	95+	White	472	0.69%

Slightly more females than males between the age of 65-69 have AF, while many more females than over the age of 95 have AF.

Analysis:
Percent of AF Patients by States



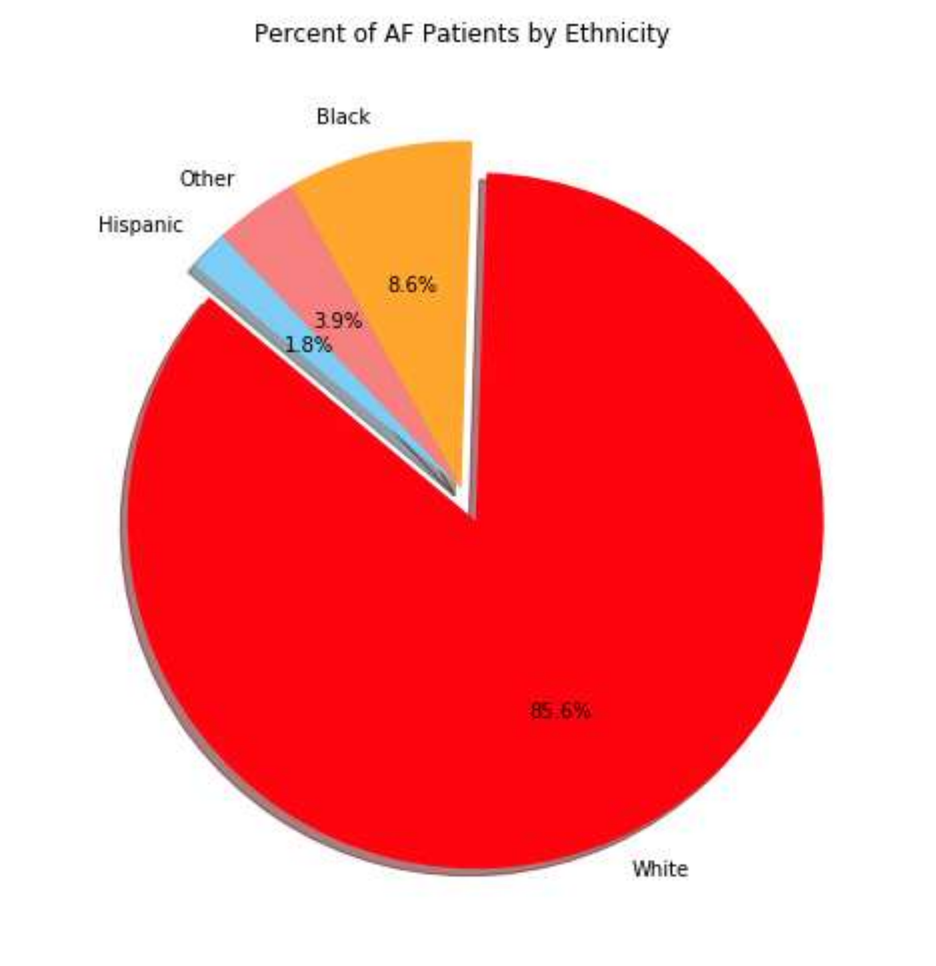
```
In [12]:  # Find the Percent of AF Patients by State Locations
          state = df[['DESYNPUF_ID', 'STATE']]
          state = state.drop_duplicates()
          state['Counts'] = state.groupby(['STATE']).transform('count')
          state = state.drop(columns='DESYNPUF_ID')
          state = state.drop_duplicates()
          state = state.sort_values('Counts', ascending=False)
          state['Percent'] = state['Counts']/pt_count
          state.head(10)
```

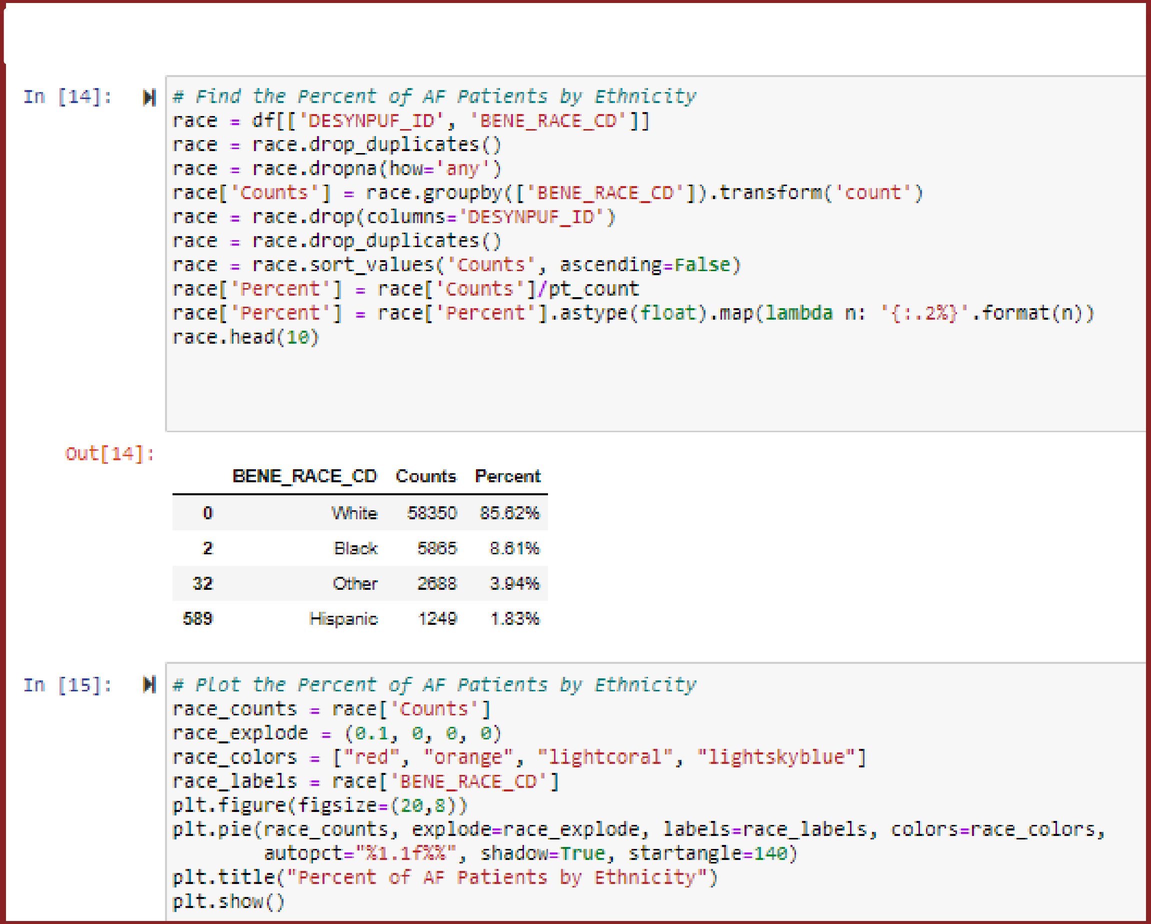
```
Out[12]:
             STATE  Counts   Percent
         28     CA  7058.0  0.103563
          5     FL  4784.0  0.070196
         10     TX  4599.0  0.067482
          1     NY  4063.0  0.059617
          2     PA  3193.0  0.046851
         13     AL  3062.0  0.044929
         29     IL  2948.0  0.043256
         60     OH  2807.0  0.041187
         18     MI  2621.0  0.038458
         57     NC  2314.0  0.033954
```

```
In [13]:  # PLot the Percent of AF Patients by State Locations
          x_axis = np.arange(len(state))
          tick_locations = [value for value in x_axis]
          plt.figure(figsize=(15,8))
          plt.bar(x_axis, state["Percent"], color='r', alpha=0.5, align="center")
          plt.xticks(tick_locations, state["STATE"], rotation="vertical")
          plt.xlim(-0.75,44)
          plt.ylim(0, max(state["Percent"])+0.01)
          plt.title("Percent of AF Patients by States")
          plt.xlabel("State")
          plt.ylabel("Percent of AF Patients")
          plt.tight_layout()
          plt.grid()
          plt.show()
```

Almost 25% of all patients with AF live in CA, FL, and TX.

Analysis:
Percent of AF Patients by Ethnicity





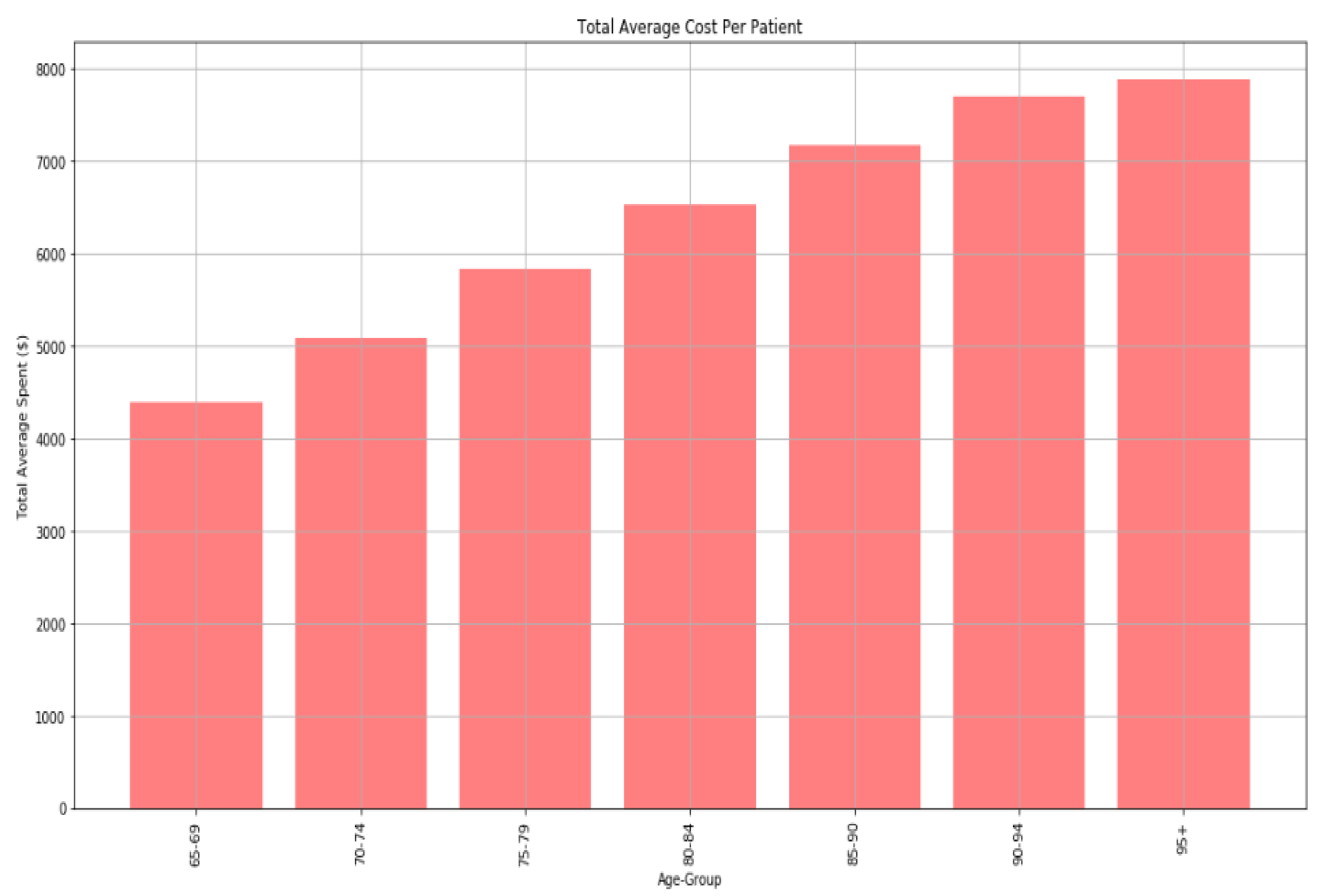
In [14]:
```
# Find the Percent of AF Patients by Ethnicity
race = df[['DESYNPUF_ID', 'BENE_RACE_CD']]
race = race.drop_duplicates()
race = race.dropna(how='any')
race['Counts'] = race.groupby(['BENE_RACE_CD']).transform('count')
race = race.drop(columns='DESYNPUF_ID')
race = race.drop_duplicates()
race = race.sort_values('Counts', ascending=False)
race['Percent'] = race['Counts']/pt_count
race['Percent'] = race['Percent'].astype(float).map(lambda n: '{:.2%}'.format(n))
race.head(10)
```

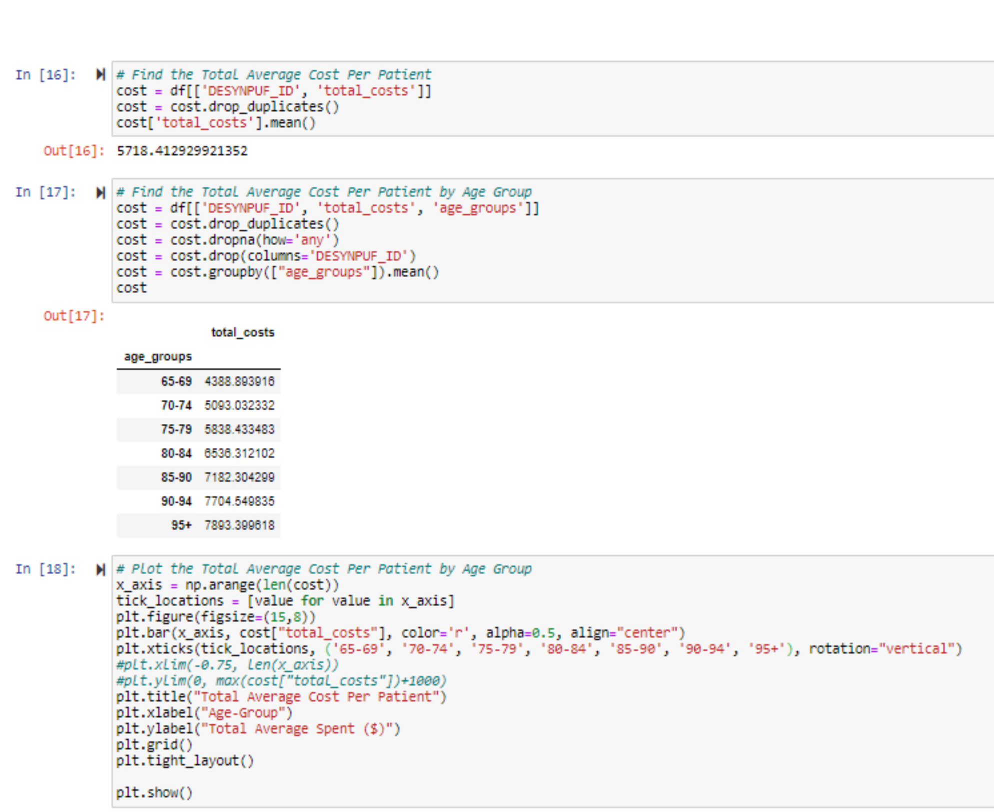
Out[14]:
```
     BENE_RACE_CD  Counts  Percent
0           White   58350   85.62%
2           Black    5865    8.61%
32          Other    2688    3.94%
589      Hispanic    1249    1.83%
```

In [15]:
```
# Plot the Percent of AF Patients by Ethnicity
race_counts = race['Counts']
race_explode = (0.1, 0, 0, 0)
race_colors = ["red", "orange", "lightcoral", "lightskyblue"]
race_labels = race['BENE_RACE_CD']
plt.figure(figsize=(20,8))
plt.pie(race_counts, explode=race_explode, labels=race_labels, colors=race_colors,
        autopct="%1.1f%%", shadow=True, startangle=140)
plt.title("Percent of AF Patients by Ethnicity")
plt.show()
```

Those of European decent are at higher risk of developing AF than those of all other ethnicities combined.

Analysis:
Total Average Cost Per Patient





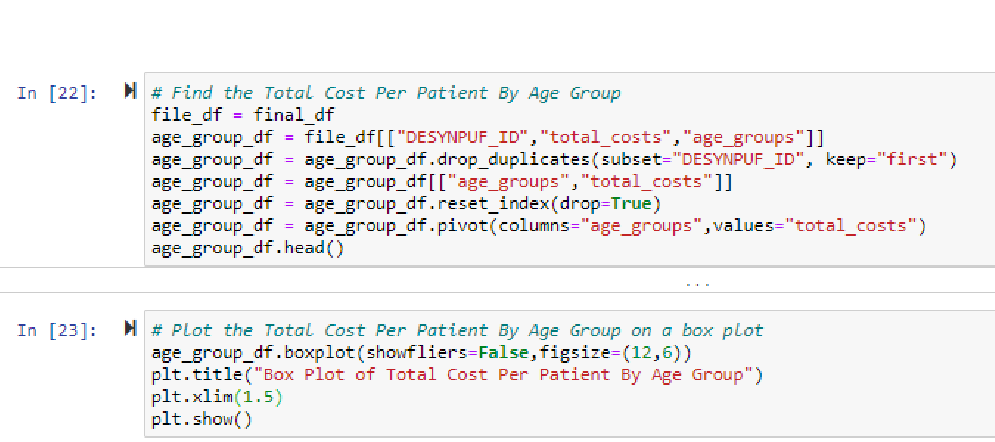
```
In [16]:  # Find the Total Average Cost Per Patient
          cost = df[['DESYNPUF_ID', 'total_costs']]
          cost = cost.drop_duplicates()
          cost['total_costs'].mean()

Out[16]:  5718.412929921352

In [17]:  # Find the Total Average Cost Per Patient by Age Group
          cost = df[['DESYNPUF_ID', 'total_costs', 'age_groups']]
          cost = cost.drop_duplicates()
          cost = cost.dropna(how='any')
          cost = cost.drop(columns='DESYNPUF_ID')
          cost = cost.groupby(["age_groups"]).mean()
          cost

Out[17]:
```

	total_costs
age_groups	
65-69	4388.893916
70-74	5093.032332
75-79	5838.433483
80-84	6536.312102
85-90	7182.304299
90-94	7704.549835
95+	7893.399618

```
In [18]:  # Plot the Total Average Cost Per Patient by Age Group
          x_axis = np.arange(len(cost))
          tick_locations = [value for value in x_axis]
          plt.figure(figsize=(15,8))
          plt.bar(x_axis, cost["total_costs"], color='r', alpha=0.5, align="center")
          plt.xticks(tick_locations, ('65-69', '70-74', '75-79', '80-84', '85-90', '90-94', '95+'), rotation="vertical")
          #plt.xlim(-0.75, len(x_axis))
          #plt.ylim(0, max(cost["total_costs"])+1000)
          plt.title("Total Average Cost Per Patient")
          plt.xlabel("Age-Group")
          plt.ylabel("Total Average Spent ($)")
          plt.grid()
          plt.tight_layout()

          plt.show()
```

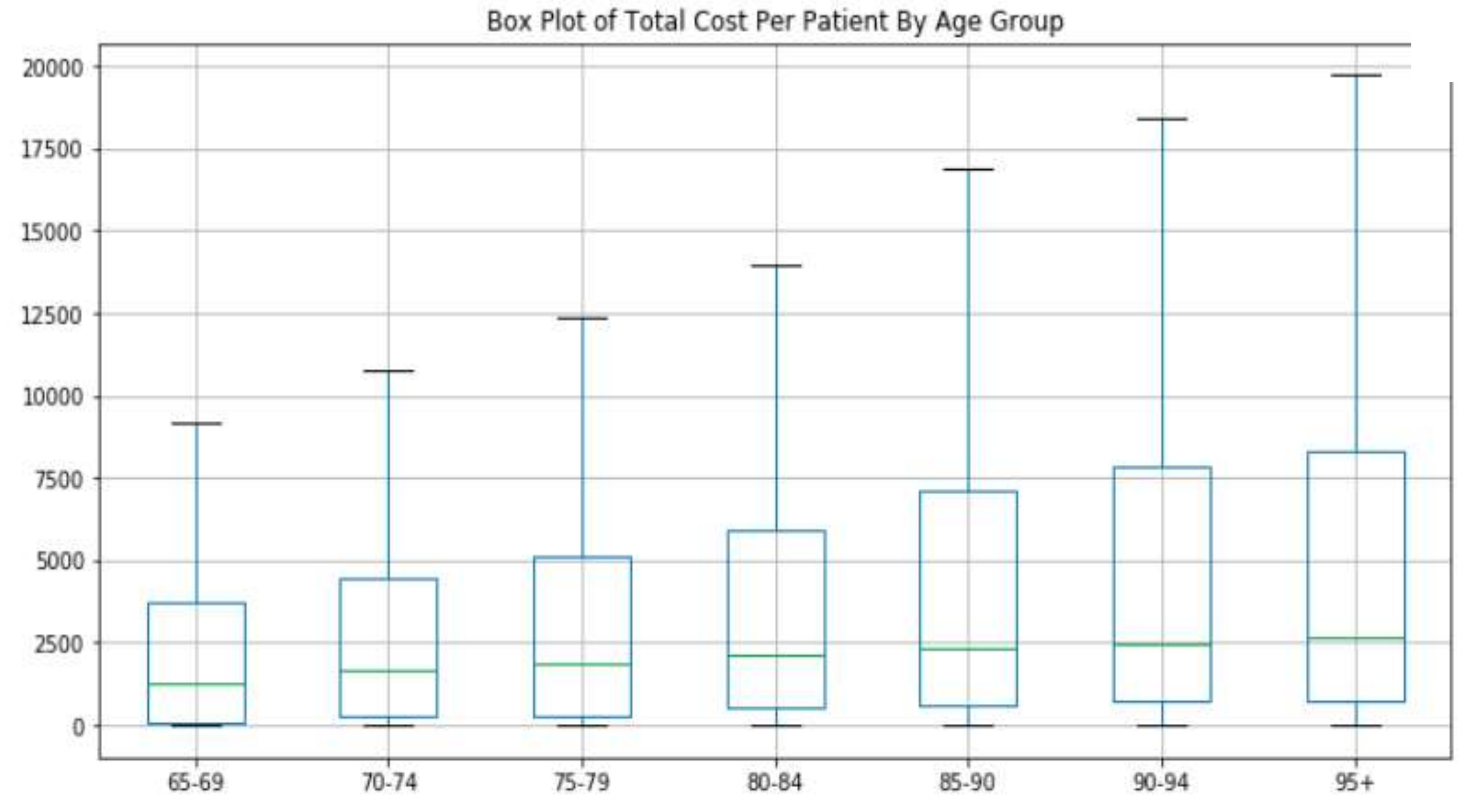
The cost of healthcare increases with ages for patients with AF.

Analysis:
Total Average Cost Per Patient



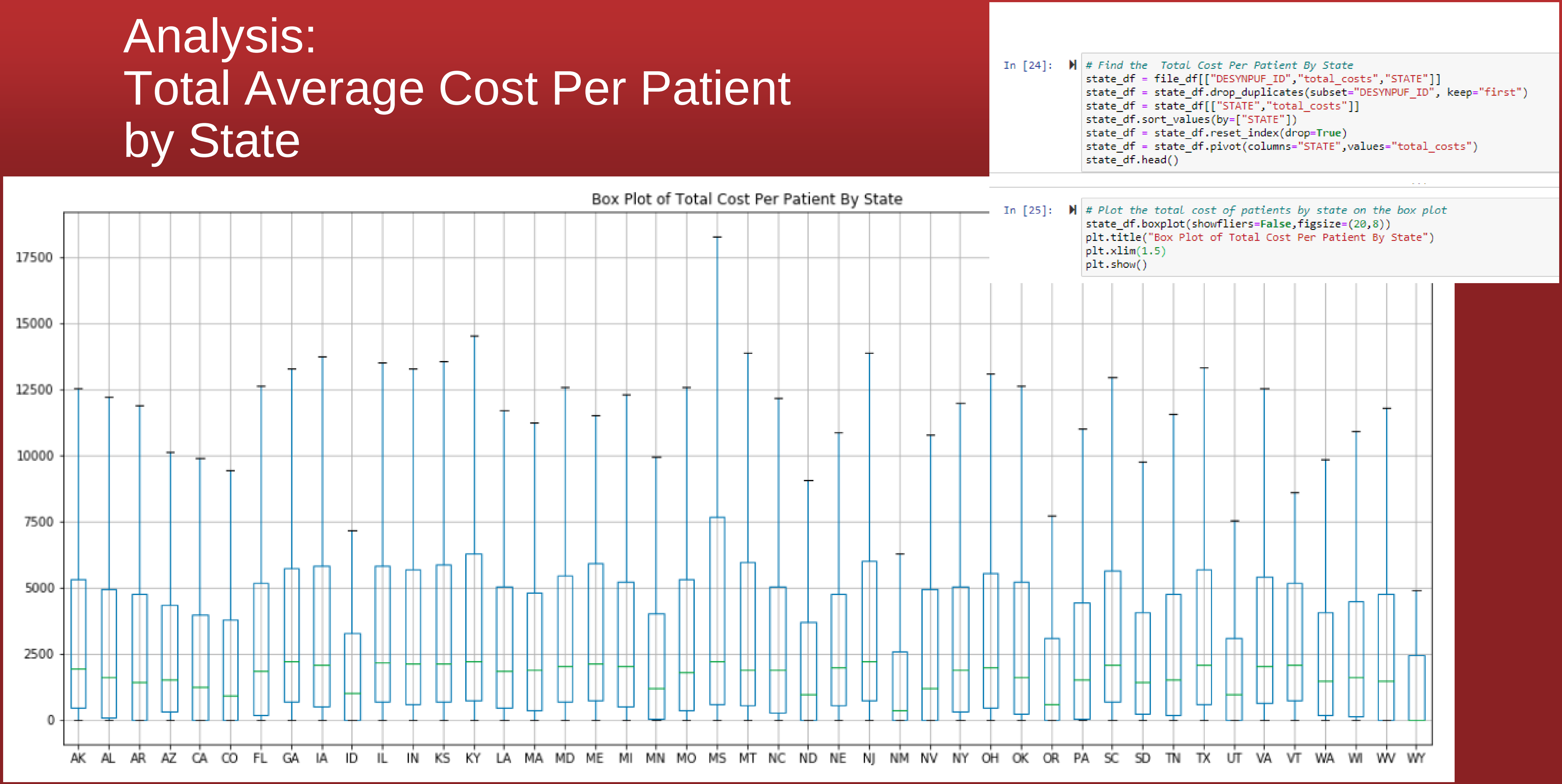
```
In [22]:  # Find the Total Cost Per Patient By Age Group
          file_df = final_df
          age_group_df = file_df[["DESYNPUF_ID","total_costs","age_groups"]]
          age_group_df = age_group_df.drop_duplicates(subset="DESYNPUF_ID", keep="first")
          age_group_df = age_group_df[["age_groups","total_costs"]]
          age_group_df = age_group_df.reset_index(drop=True)
          age_group_df = age_group_df.pivot(columns="age_groups",values="total_costs")
          age_group_df.head()
```

```
In [23]:  # Plot the Total Cost Per Patient By Age Group on a box plot
          age_group_df.boxplot(showfliers=False,figsize=(12,6))
          plt.title("Box Plot of Total Cost Per Patient By Age Group")
          plt.xlim(1.5)
          plt.show()
```



The cost of healthcare increases with ages for patients with AF.

Analysis:
Total Average Cost Per Patient by State

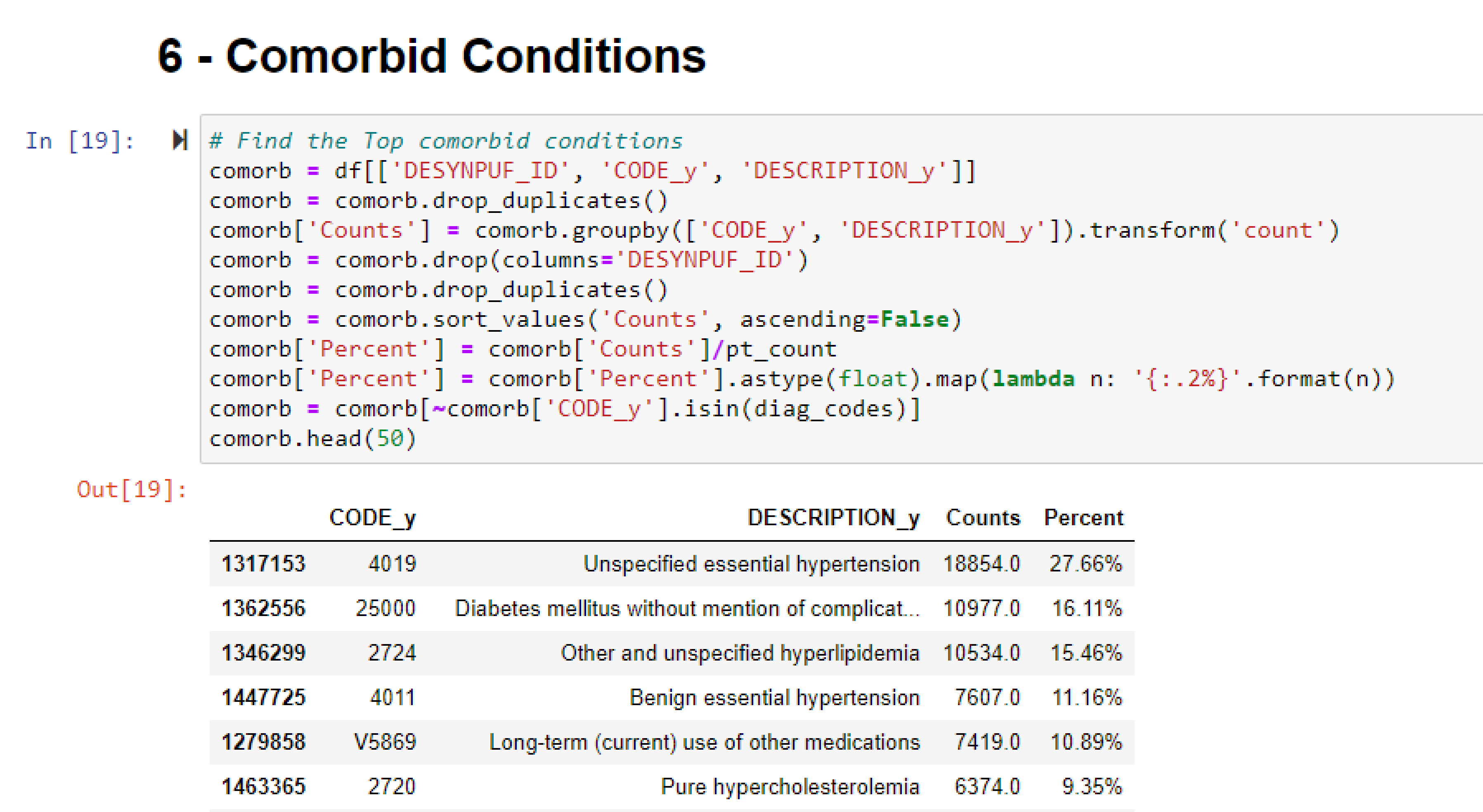


```
In [24]:  # Find the  Total Cost Per Patient By State
          state_df = file_df[["DESYNPUF_ID","total_costs","STATE"]]
          state_df = state_df.drop_duplicates(subset="DESYNPUF_ID", keep="first")
          state_df = state_df[["STATE","total_costs"]]
          state_df.sort_values(by=["STATE"])
          state_df = state_df.reset_index(drop=True)
          state_df = state_df.pivot(columns="STATE",values="total_costs")
          state_df.head()
```

```
In [25]:  # Plot the total cost of patients by state on the box plot
          state_df.boxplot(showfliers=False,figsize=(20,8))
          plt.title("Box Plot of Total Cost Per Patient By State")
          plt.xlim(1.5)
          plt.show()
```

Healthcare costs of patients with AF vary greatly by states.

Analysis



6 - Comorbid Conditions

In [19]:

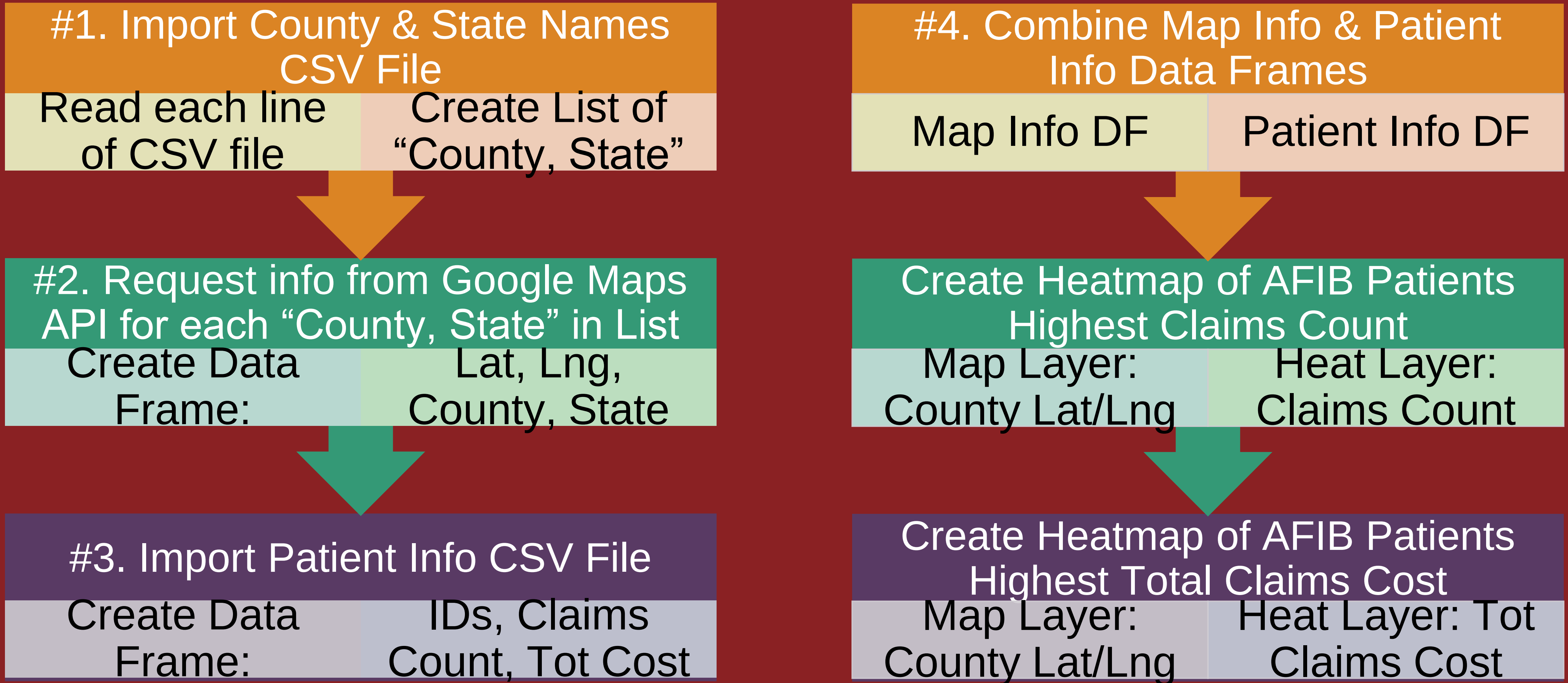
```
# Find the Top comorbid conditions
comorb = df[['DESYNPUF_ID', 'CODE_y', 'DESCRIPTION_y']]
comorb = comorb.drop_duplicates()
comorb['Counts'] = comorb.groupby(['CODE_y', 'DESCRIPTION_y']).transform('count')
comorb = comorb.drop(columns='DESYNPUF_ID')
comorb = comorb.drop_duplicates()
comorb = comorb.sort_values('Counts', ascending=False)
comorb['Percent'] = comorb['Counts']/pt_count
comorb['Percent'] = comorb['Percent'].astype(float).map(lambda n: '{:.2%}'.format(n))
comorb = comorb[~comorb['CODE_y'].isin(diag_codes)]
comorb.head(50)
```

Out[19]:

	CODE_y	DESCRIPTION_y	Counts	Percent
1317153	4019	Unspecified essential hypertension	18854.0	27.66%
1362556	25000	Diabetes mellitus without mention of complicat...	10977.0	16.11%
1346299	2724	Other and unspecified hyperlipidemia	10534.0	15.46%
1447725	4011	Benign essential hypertension	7607.0	11.16%
1279858	V5869	Long-term (current) use of other medications	7419.0	10.89%
1463365	2720	Pure hypercholesterolemia	6374.0	9.35%

Those with AF are likely to also have hypertension, diabetes mellitus, hyperlipidemia, and hypercholesterolemia.

U.S. AFIB Patient Heatmap Creation Process using Jupyter Notebook & Python Code



#1. Import County & State Names CSV File

Read each line of CSV file

Create List of “County, State”

#2. Request info from Google Maps API for each “County, State” in List

Create Data Frame:

Lat, Lng, County, State

#3. Import Patient Info CSV File

Create Data Frame:

IDs, Claims Count, Tot Cost

#4. Combine Map Info & Patient Info Data Frames

Map Info DF

Patient Info DF

Create Heatmap of AFIB Patients Highest Claims Count

Map Layer: County Lat/Lng

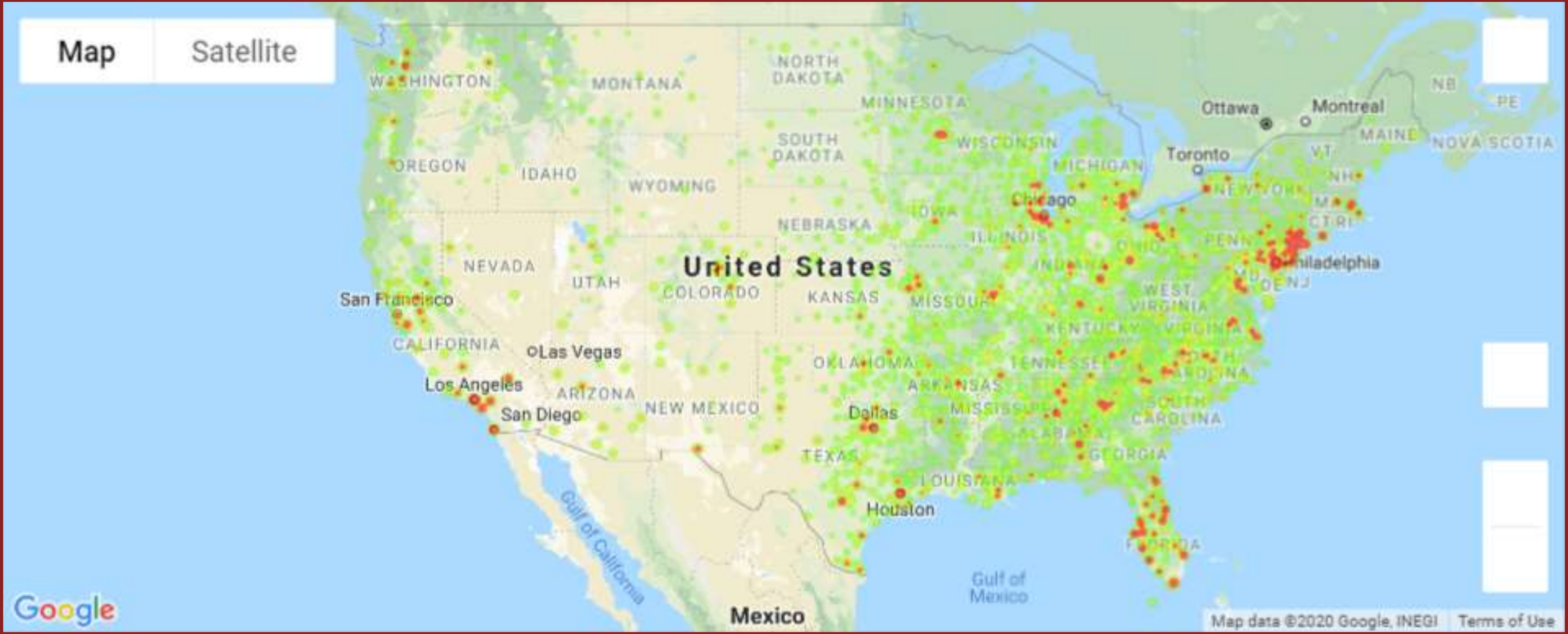
Heat Layer: Claims Count

Create Heatmap of AFIB Patients Highest Total Claims Cost

Map Layer: County Lat/Lng

Heat Layer: Tot Claims Cost

U.S. AFIB Patient Heatmap of Number of Patients, by Location, with the Highest Claims *Count*

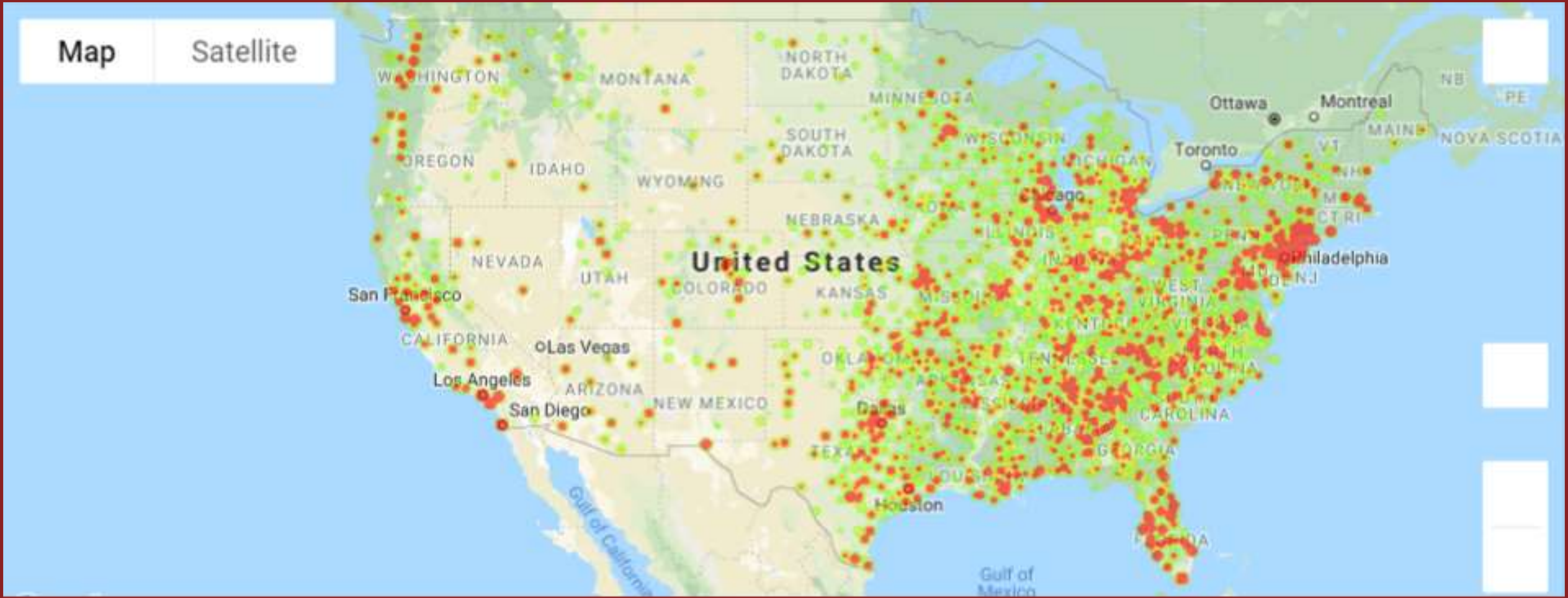


U.S. AFIB Patient Heatmap Findings

- U.S. AFIB Patient Heatmap of Number of Patients, by Location, with the Highest Claims Count:
- Los Angeles, CA
- Chicago, IL
- New York City, NY
- Philadelphia, PA
- Miami, FL

- U.S. AFIB Patient Heatmap of Number of Patients, by Location, with the Highest Total Claims Cost:
- Eastern half of the U.S.
- West Coast

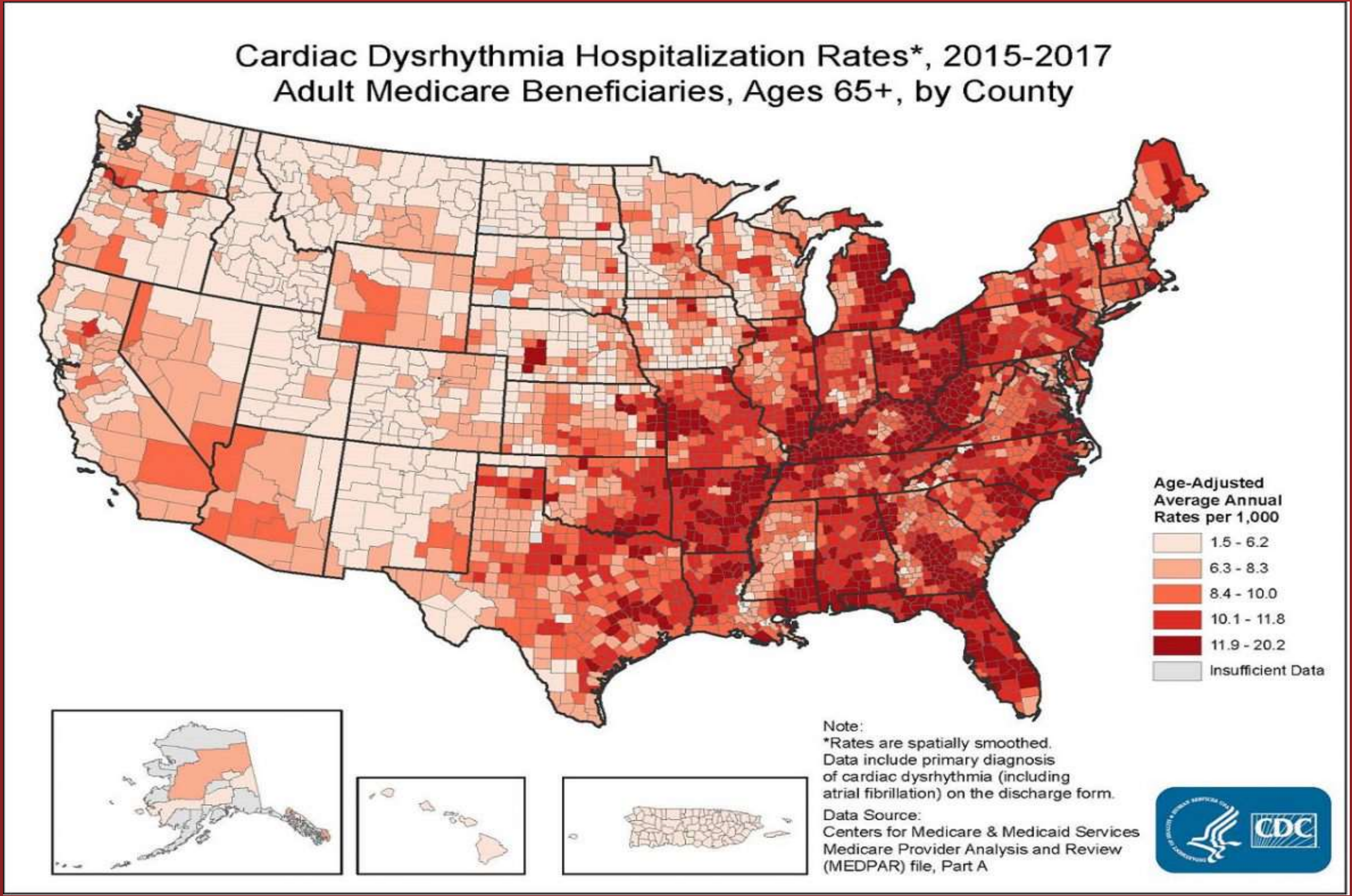
U.S. AFIB Patient Heatmap of Number of Patients, by Location, with the Highest Total Claims *Cost*



Implications of our findings: what do they mean?

- The total claims *count* for AFIB patients in highly populated cities across the U.S. is proportionate to the total claims *cost* for all AFIB patients in these regions

- The total claims *cost* for AFIB patients in the Eastern half of the U.S., (in particular, across the Midwest and Southeastern states), is disproportionately high, compared to the total claims *count* for all AFIB patients in these regions



Cardiac Dysrhythmia Hospitalization Rates*, 2015-2017
Adult Medicare Beneficiaries, Ages 65+, by County

Note:
*Rates are spatially smoothed.
Data include primary diagnosis
of cardiac dysrhythmia (including
atrial fibrillation) on the discharge form.

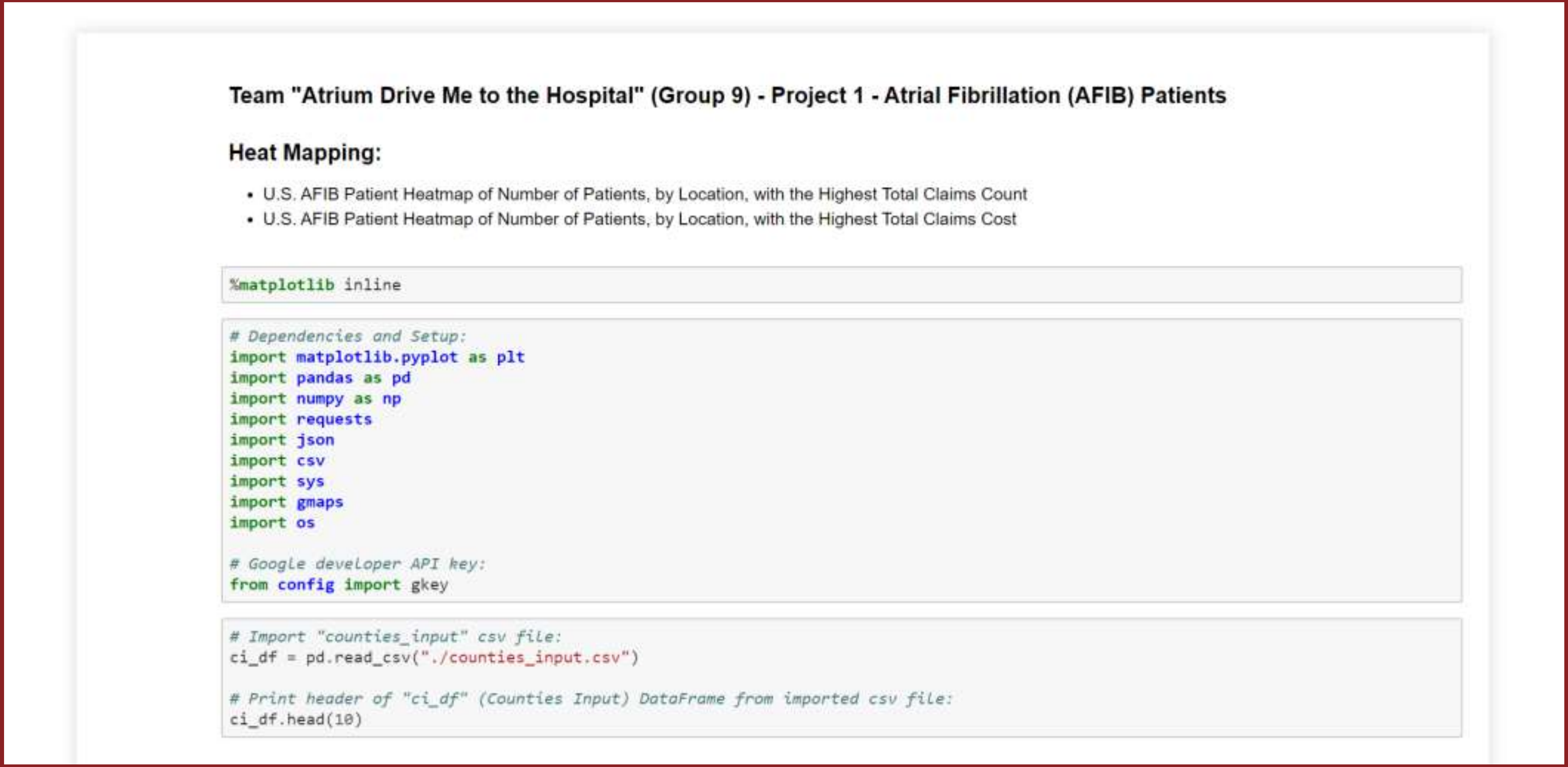
Data Source:
Centers for Medicare & Medicaid Services
Medicare Provider Analysis and Review
(MEDPAR) file, Part A

Conclusions

Based on Medicare's population:

- Those of European descent make up the largest ethnicity patients with AF
- A larger number of females have AF compared to males
- There are more patients with AF on the east cost and west coast
- Patients with AF in the Midwest pay more for healthcare costs than those living on the east coast and west coast

Team "Atrium Drive Me to the Doctor's" (Group 9) - Project 1 - Atrial Fibrillation (AFIB) Patients Code Screen Shot (1 of 7)



Team "Atrium Drive Me to the Hospital" (Group 9) - Project 1 - Atrial Fibrillation (AFIB) Patients

Heat Mapping:

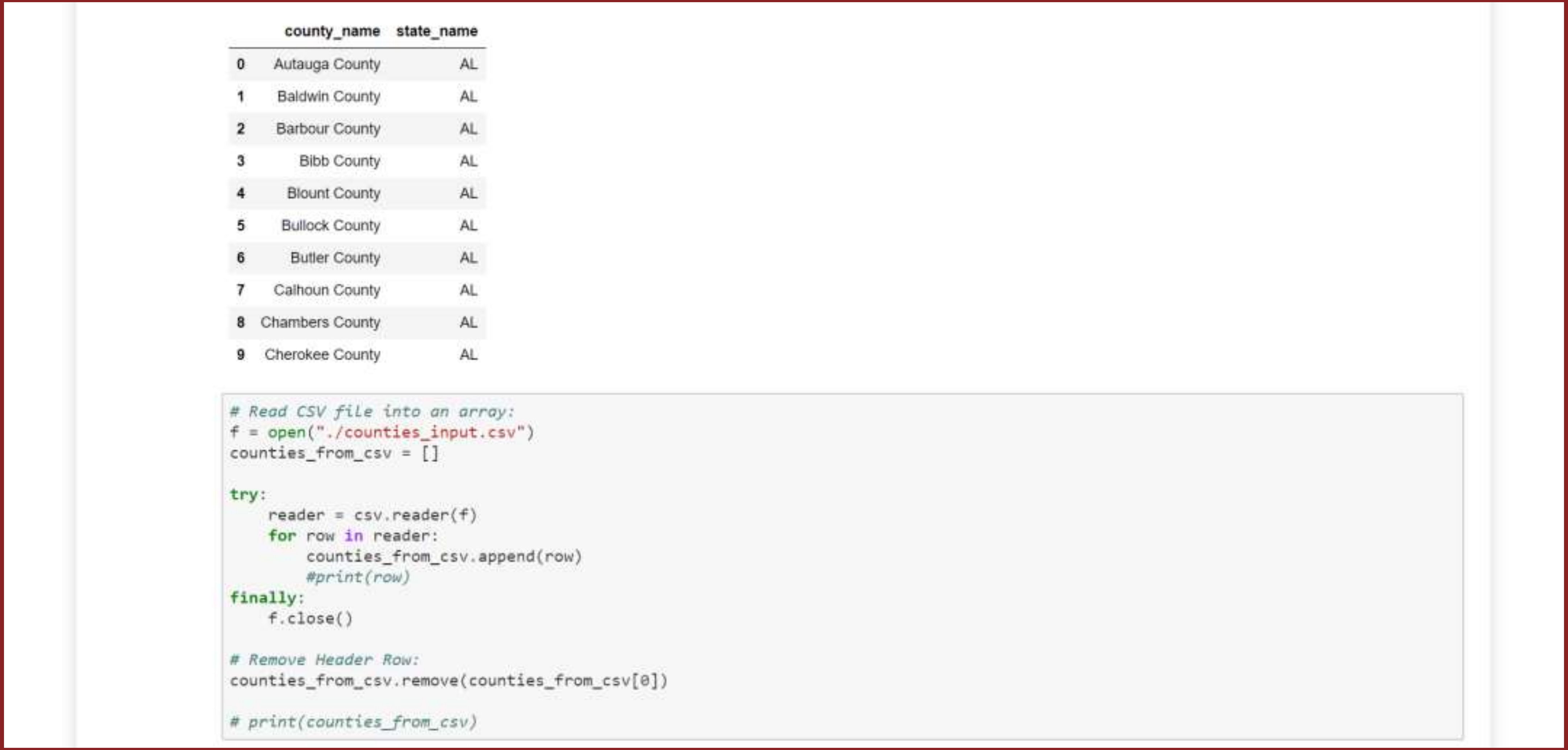
- U.S. AFIB Patient Heatmap of Number of Patients, by Location, with the Highest Total Claims Count
- U.S. AFIB Patient Heatmap of Number of Patients, by Location, with the Highest Total Claims Cost

```
%matplotlib inline
```

```
# Dependencies and Setup:
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import requests
import json
import csv
import sys
import gmaps
import os

# Google developer API key:
from config import gkey
```

```
# Import "counties_input" csv file:
ci_df = pd.read_csv("./counties_input.csv")

# Print header of "ci_df" (Counties Input) DataFrame from imported csv file:
ci_df.head(10)
```

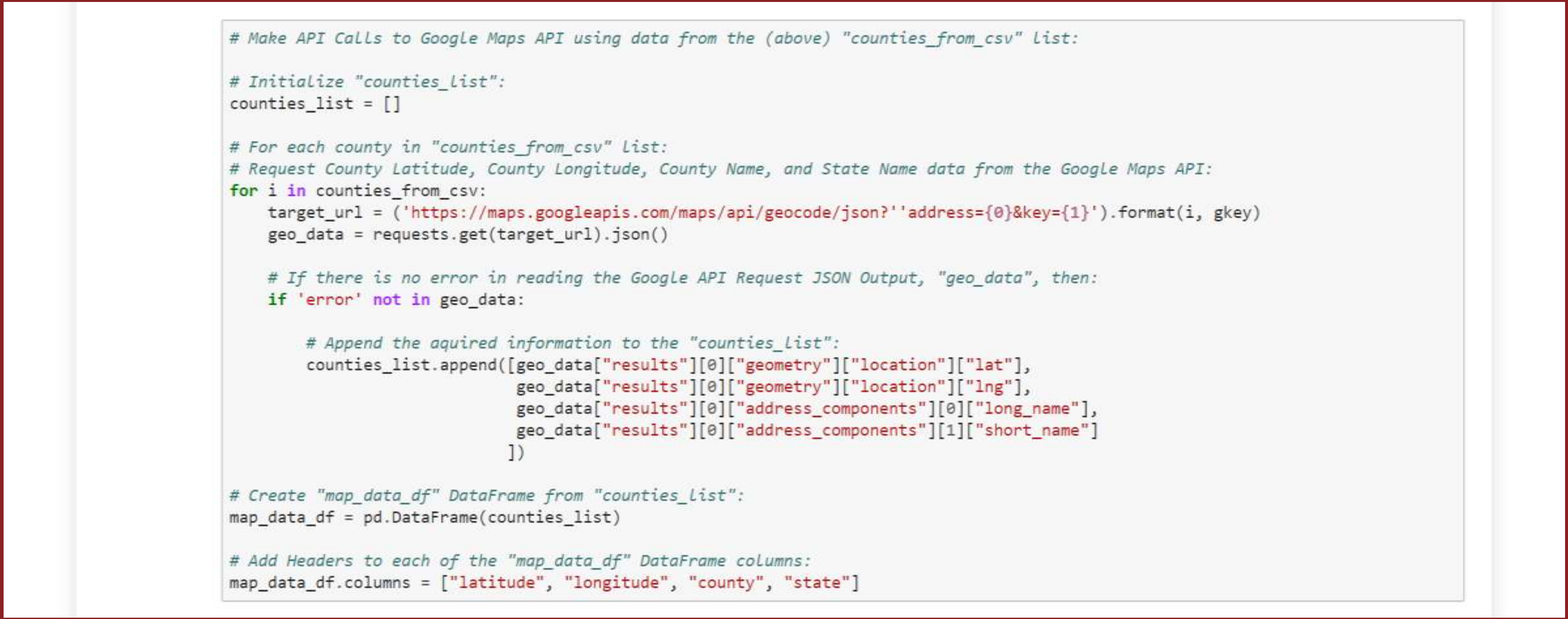
Team "Atrium Drive Me to the Doctor’s" (Group 9) - Project 1 - Atrial Fibrillation (AFIB) Patients
Code Screen Shot (2 of 7)



	county_name	state_name
0	Autauga County	AL
1	Baldwin County	AL
2	Barbour County	AL
3	Bibb County	AL
4	Blount County	AL
5	Bullock County	AL
6	Butler County	AL
7	Calhoun County	AL
8	Chambers County	AL
9	Cherokee County	AL

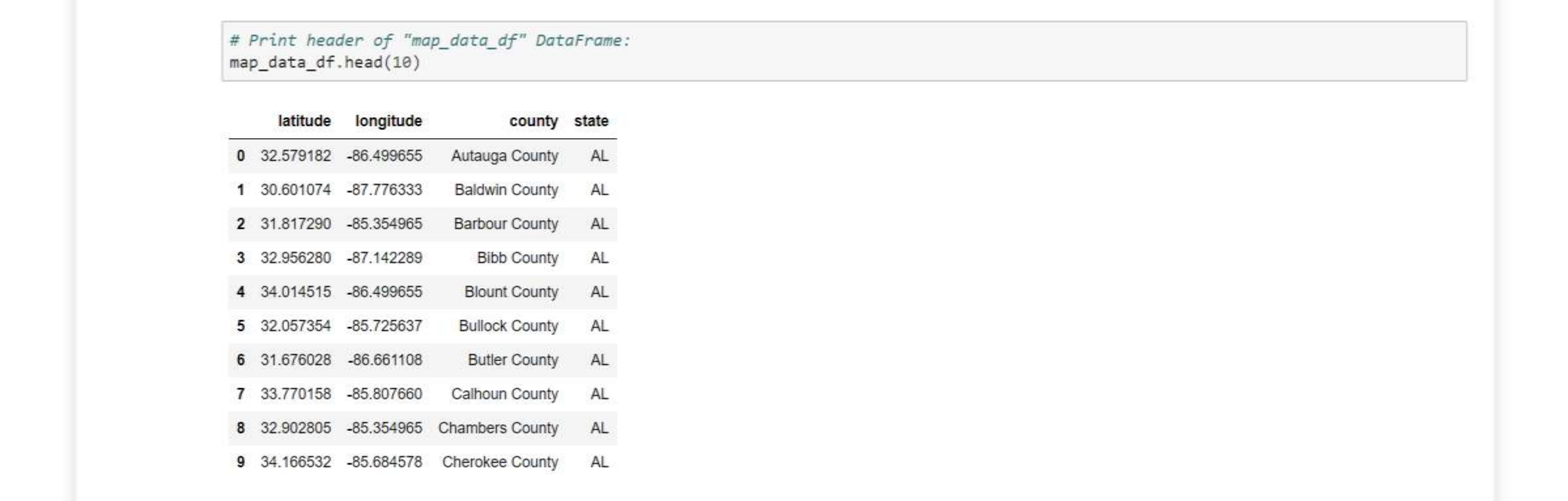
```
# Read CSV file into an array:
f = open("./counties_input.csv")
counties_from_csv = []

try:
    reader = csv.reader(f)
    for row in reader:
        counties_from_csv.append(row)
        #print(row)
finally:
    f.close()

# Remove Header Row:
counties_from_csv.remove(counties_from_csv[0])

# print(counties_from_csv)
```

Team "Atrium Drive Me to the Doctor’s" (Group 9) - Project 1 - Atrial Fibrillation (AFIB) Patients
Code Screen Shot (3 of 7)



```python
# Make API Calls to Google Maps API using data from the (above) "counties_from_csv" list:

# Initialize "counties_list":
counties_list = []

# For each county in "counties_from_csv" list:
# Request County Latitude, County Longitude, County Name, and State Name data from the Google Maps API:
for i in counties_from_csv:
    target_url = ('https://maps.googleapis.com/maps/api/geocode/json?''address={0}&key={1}').format(i, gkey)
    geo_data = requests.get(target_url).json()

    # If there is no error in reading the Google API Request JSON Output, "geo_data", then:
    if 'error' not in geo_data:

        # Append the aquired information to the "counties_list":
        counties_list.append([geo_data["results"][0]["geometry"]["location"]["lat"],
                              geo_data["results"][0]["geometry"]["location"]["lng"],
                              geo_data["results"][0]["address_components"][0]["long_name"],
                              geo_data["results"][0]["address_components"][1]["short_name"]
                             ])

# Create "map_data_df" DataFrame from "counties_list":
map_data_df = pd.DataFrame(counties_list)

# Add Headers to each of the "map_data_df" DataFrame columns:
map_data_df.columns = ["latitude", "longitude", "county", "state"]
```

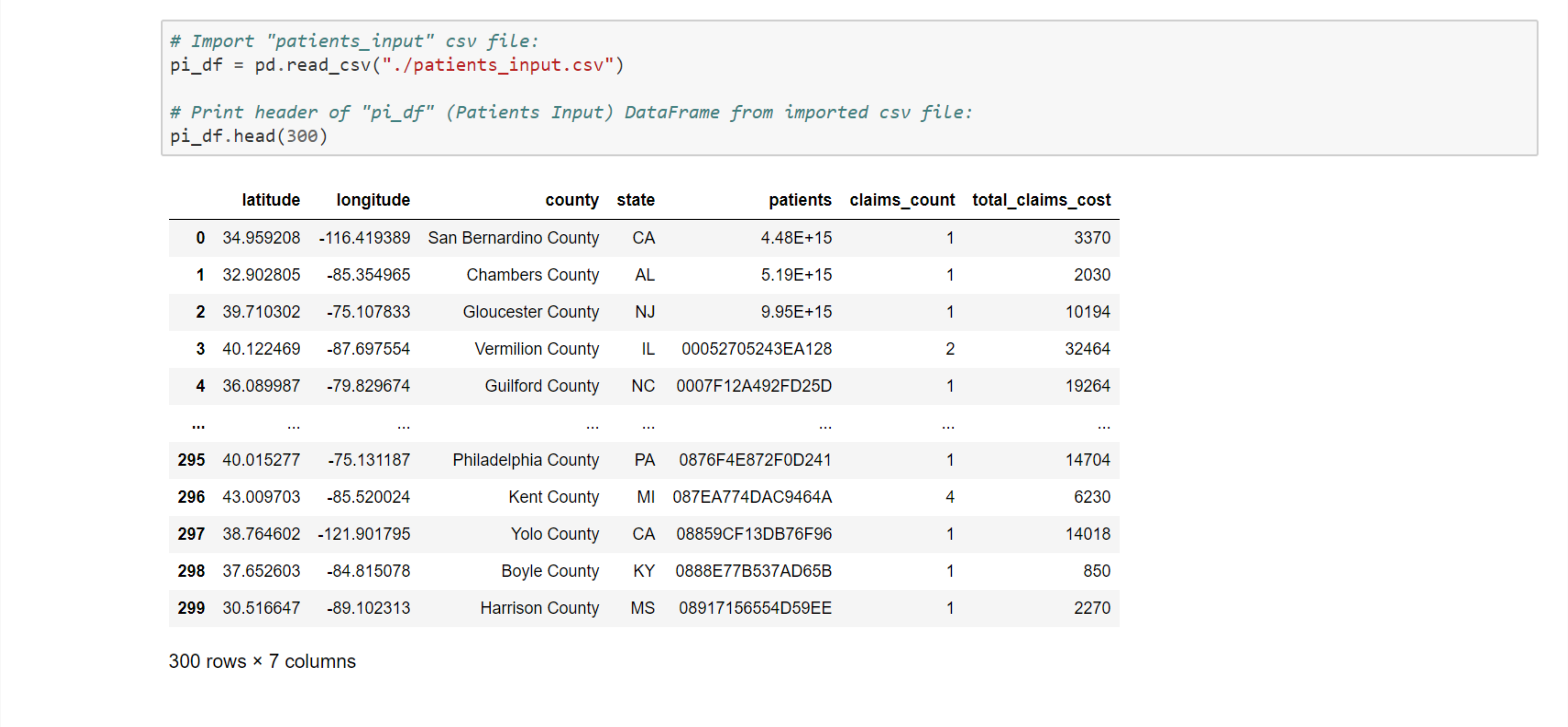
Team "Atrium Drive Me to the Doctor’s" (Group 9) - Project 1 - Atrial Fibrillation (AFIB) Patients
Code Screen Shot (4 of 7)



```
# Print header of "map_data_df" DataFrame:
map_data_df.head(10)
```

	latitude	longitude	county	state
0	32.579182	-86.499655	Autauga County	AL
1	30.601074	-87.776333	Baldwin County	AL
2	31.817290	-85.354965	Barbour County	AL
3	32.956280	-87.142289	Bibb County	AL
4	34.014515	-86.499655	Blount County	AL
5	32.057354	-85.725637	Bullock County	AL
6	31.676028	-86.661108	Butler County	AL
7	33.770158	-85.807660	Calhoun County	AL
8	32.902805	-85.354965	Chambers County	AL
9	34.166532	-85.684578	Cherokee County	AL

Team "Atrium Drive Me to the Doctor’s" (Group 9) - Project 1 - Atrial Fibrillation (AFIB) Patients Code Screen Shot (5 of 7)

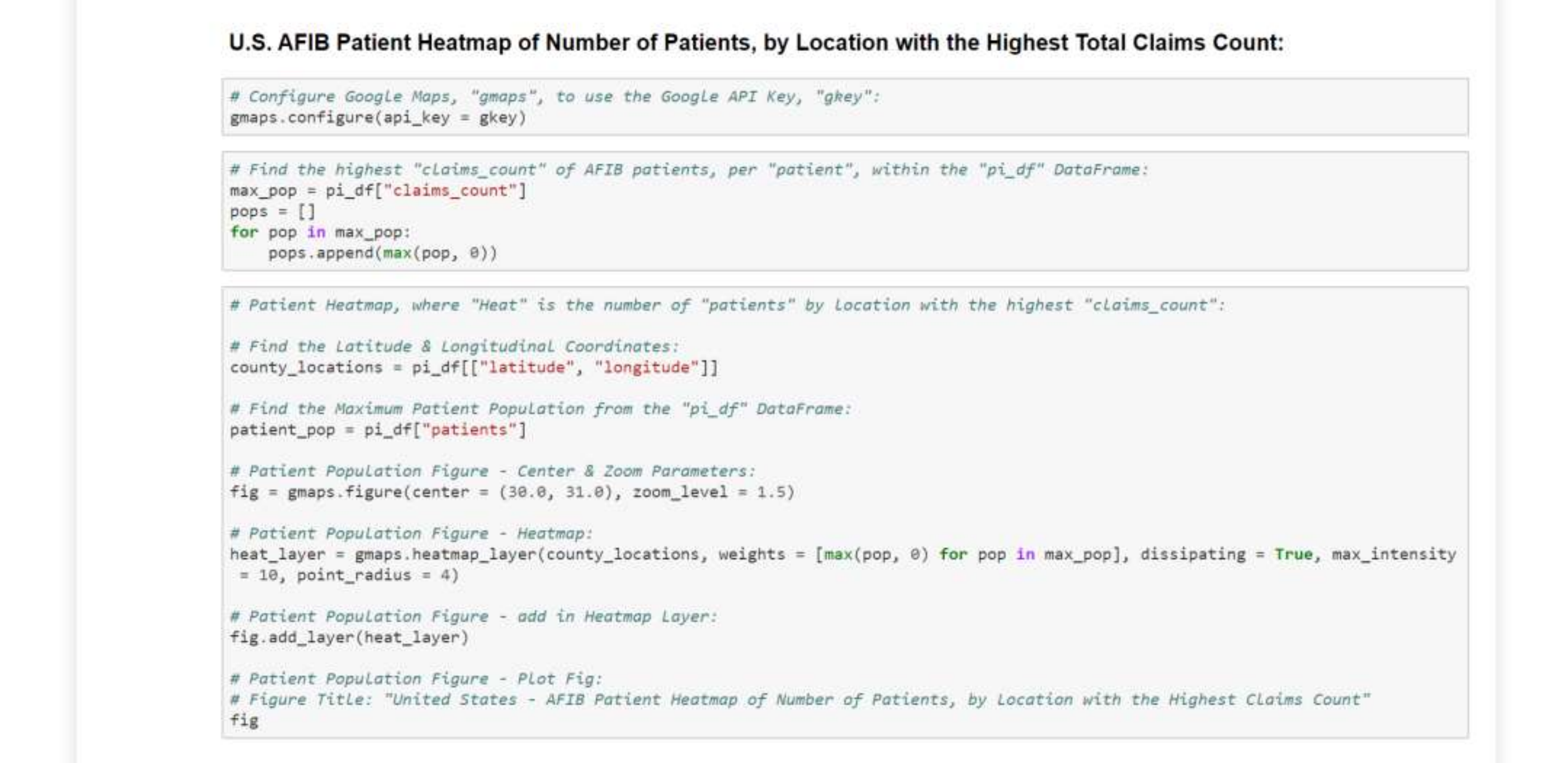


```
# Import "patients_input" csv file:
pi_df = pd.read_csv("./patients_input.csv")

# Print header of "pi_df" (Patients Input) DataFrame from imported csv file:
pi_df.head(300)
```

	latitude	longitude	county	state	patients	claims_count	total_claims_cost
0	34.959208	-116.419389	San Bernardino County	CA	4.48E+15	1	3370
1	32.902805	-85.354965	Chambers County	AL	5.19E+15	1	2030
2	39.710302	-75.107833	Gloucester County	NJ	9.95E+15	1	10194
3	40.122469	-87.697554	Vermilion County	IL	00052705243EA128	2	32464
4	36.089987	-79.829674	Guilford County	NC	0007F12A492FD25D	1	19264
...	...	...	...	...	...	...	...
295	40.015277	-75.131187	Philadelphia County	PA	0876F4E872F0D241	1	14704
296	43.009703	-85.520024	Kent County	MI	087EA774DAC9464A	4	6230
297	38.764602	-121.901795	Yolo County	CA	08859CF13DB76F96	1	14018
298	37.652603	-84.815078	Boyle County	KY	0888E77B537AD65B	1	850
299	30.516647	-89.102313	Harrison County	MS	08917156554D59EE	1	2270

300 rows × 7 columns

Team "Atrium Drive Me to the Doctor’s" (Group 9) - Project 1 - Atrial Fibrillation (AFIB) Patients Code Screen Shot (6 of 7)



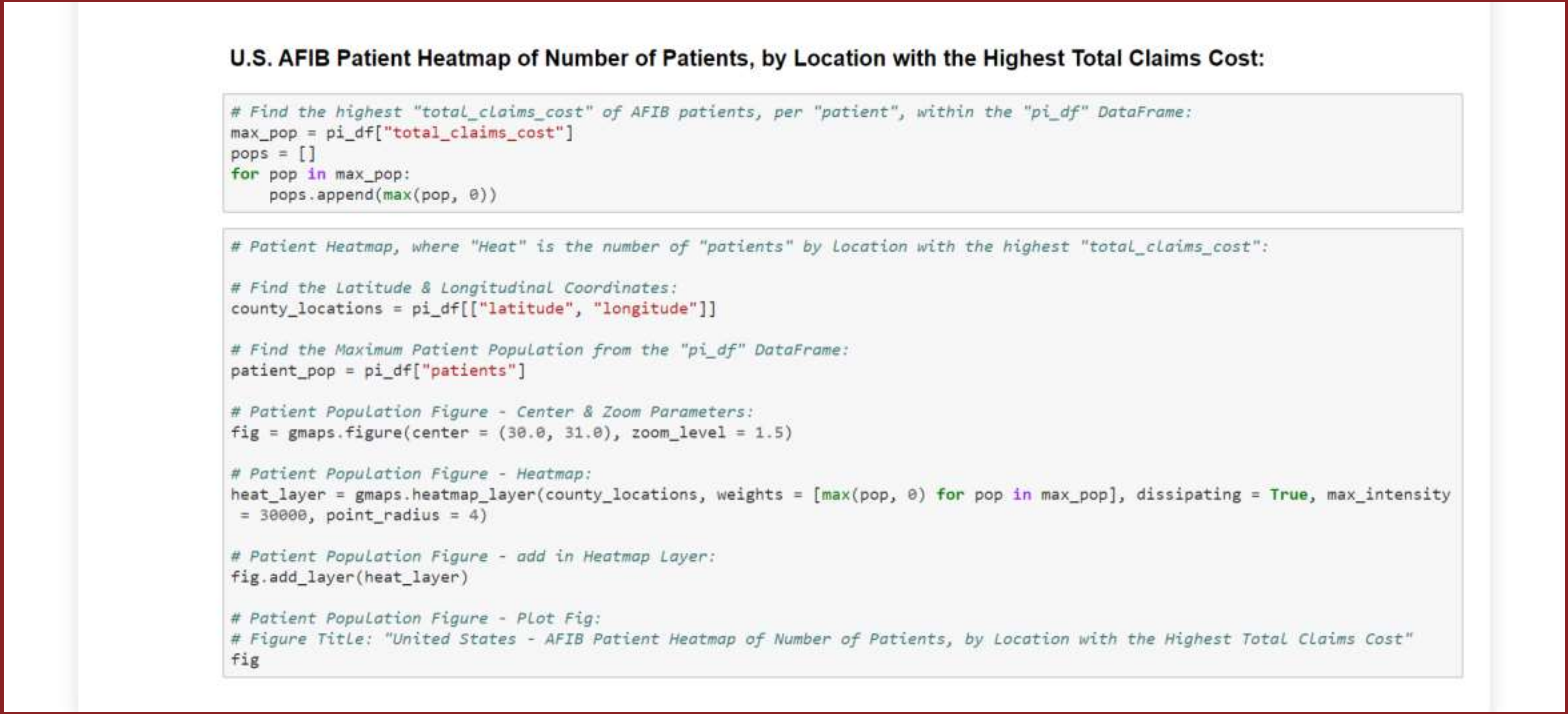
U.S. AFIB Patient Heatmap of Number of Patients, by Location with the Highest Total Claims Count:

```
# Configure Google Maps, "gmaps", to use the Google API Key, "gkey":
gmaps.configure(api_key = gkey)
```

```
# Find the highest "claims_count" of AFIB patients, per "patient", within the "pi_df" DataFrame:
max_pop = pi_df["claims_count"]
pops = []
for pop in max_pop:
    pops.append(max(pop, 0))
```

```
# Patient Heatmap, where "Heat" is the number of "patients" by location with the highest "claims_count":

# Find the Latitude & Longitudinal Coordinates:
county_locations = pi_df[["latitude", "longitude"]]

# Find the Maximum Patient Population from the "pi_df" DataFrame:
patient_pop = pi_df["patients"]

# Patient Population Figure - Center & Zoom Parameters:
fig = gmaps.figure(center = (30.0, 31.0), zoom_level = 1.5)

# Patient Population Figure - Heatmap:
heat_layer = gmaps.heatmap_layer(county_locations, weights = [max(pop, 0) for pop in max_pop], dissipating = True, max_intensity
 = 10, point_radius = 4)

# Patient Population Figure - add in Heatmap Layer:
fig.add_layer(heat_layer)

# Patient Population Figure - Plot Fig:
# Figure Title: "United States - AFIB Patient Heatmap of Number of Patients, by Location with the Highest Claims Count"
fig
```

Team "Atrium Drive Me to the Doctor’s" (Group 9) - Project 1 - Atrial Fibrillation (AFIB) Patients Code Screen Shot (7 of 7)



U.S. AFIB Patient Heatmap of Number of Patients, by Location with the Highest Total Claims Cost:

```
# Find the highest "total_claims_cost" of AFIB patients, per "patient", within the "pi_df" DataFrame:
max_pop = pi_df["total_claims_cost"]
pops = []
for pop in max_pop:
    pops.append(max(pop, 0))
```

```
# Patient Heatmap, where "Heat" is the number of "patients" by Location with the highest "total_claims_cost":

# Find the Latitude & Longitudinal Coordinates:
county_locations = pi_df[["latitude", "longitude"]]

# Find the Maximum Patient Population from the "pi_df" DataFrame:
patient_pop = pi_df["patients"]

# Patient Population Figure - Center & Zoom Parameters:
fig = gmaps.figure(center = (30.0, 31.0), zoom_level = 1.5)

# Patient Population Figure - Heatmap:
heat_layer = gmaps.heatmap_layer(county_locations, weights = [max(pop, 0) for pop in max_pop], dissipating = True, max_intensity
 = 30000, point_radius = 4)

# Patient Population Figure - add in Heatmap Layer:
fig.add_layer(heat_layer)

# Patient Population Figure - Plot Fig:
# Figure Title: "United States - AFIB Patient Heatmap of Number of Patients, by Location with the Highest Total Claims Cost"
fig
```