BIG DATA FINAL PROJECT REPORT

Prepared By
Ahmed Eid Abdullah 201701264
Ahmed Kamal Elmelegy 201700295

Submitted To
Dr. ElSayed Hemeyed
Eng. Ahmed Sameh
Eng. Mohamed ElAref

Table of content

Technical Part

Dataset

Name of dataset: Death in the United State.

Description of dataset: This dataset is a collection of CSV files each containing one year's worth of data and paired JSON files containing the code mappings.

Code and our methodology

1- Firstly, we uploaded the dataset on drive. Then we installed pyspark

```
# Import PyDrive and associated libraries.
# This only needs to be done once per notebook.
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

# Authenticate and create the PyDrive client.
# This only needs to be done once per notebook.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# Download a file based on its file ID.
#
# A file ID looks like: laggVyWshwcyP6kEI-y_W3P8D26sz
file_id = '1FU6NJFNr4x_7Sohk-kAFGb3uvlemOdsY'
downloaded = drive.CreateFile({'id': file_id})
downloaded.GetContentFile('Death_Dataset.zip')
```

```
[ ] !mkdir /content/Death_Dataset
    !unzip -q /content/Death_Dataset.zip -d /content/Death_Dataset
```

```
[ ] !pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.2.1.tar.gz (281.4 MB)
     |████████████████████████████████| 281.4 MB 27 kB/s
```

```
[ ] import pyspark
    from pyspark.sql import SparkSession
    from pyspark.sql import functions as F
    import json
    import numpy as np
```

```
[ ] spark = SparkSession.builder.appName("death").getOrCreate()
    df = spark.read.csv("/content/Death_Dataset/2005_data.csv")
```

2- Then we read each file from dataset (CSV file)

```
[ ]  death_2005_data = spark.read.option("header",True).csv("/content/Death_Dataset/2005_data.csv")
     death_2006_data = spark.read.option("header",True).csv("/content/Death_Dataset/2006_data.csv")
     death_2007_data = spark.read.option("header",True).csv("/content/Death_Dataset/2007_data.csv")
     death_2008_data = spark.read.option("header",True).csv("/content/Death_Dataset/2008_data.csv")
     death_2009_data = spark.read.option("header",True).csv("/content/Death_Dataset/2009_data.csv")
     death_2010_data = spark.read.option("header",True).csv("/content/Death_Dataset/2010_data.csv")
     death_2011_data = spark.read.option("header",True).csv("/content/Death_Dataset/2011_data.csv")
     death_2012_data = spark.read.option("header",True).csv("/content/Death_Dataset/2012_data.csv")
     death_2013_data = spark.read.option("header",True).csv("/content/Death_Dataset/2013_data.csv")
     death_2014_data = spark.read.option("header",True).csv("/content/Death_Dataset/2014_data.csv")
     death_2015_data = spark.read.option("header",True).csv("/content/Death_Dataset/2015_data.csv")
```

3- We implemented a function to do word count on CSV file

```
[ ]  from pyspark.sql.functions import col

     def Get_wordcount_specific_column (path,column_name):
       read_data = spark.read.option("header",True).csv(path)
       specific_column = read_data.select(col(column_name))
       wordcount_column = (specific_column.groupBy(column_name).count())

       return wordcount_column
```

4- We implemented a read json function

```
     def read_json(path):
       with open(path, 'r') as f:
             code=json.load(f)
             return code
```

Reading json data

```
[ ]  code_2005=read_json('/content/Death_Dataset/2005_codes.json')
     code_2006=read_json('/content/Death_Dataset/2006_codes.json')
     code_2007=read_json('/content/Death_Dataset/2007_codes.json')
     code_2008=read_json('/content/Death_Dataset/2008_codes.json')
     code_2009=read_json('/content/Death_Dataset/2009_codes.json')
     code_2010=read_json('/content/Death_Dataset/2010_codes.json')
     code_2011=read_json('/content/Death_Dataset/2011_codes.json')
     code_2012=read_json('/content/Death_Dataset/2012_codes.json')
     code_2013=read_json('/content/Death_Dataset/2013_codes.json')
     code_2014=read_json('/content/Death_Dataset/2014_codes.json')
     code_2015=read_json('/content/Death_Dataset/2015_codes.json')
```

5- Then we called word count function on single variables.

▾ Manner of death word count

```
mannerofdeath_2005_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2005_data.csv","manner_of_death")
mannerofdeath_2006_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2006_data.csv","manner_of_death")
mannerofdeath_2007_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2007_data.csv","manner_of_death")
mannerofdeath_2008_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2008_data.csv","manner_of_death")
mannerofdeath_2009_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2009_data.csv","manner_of_death")
mannerofdeath_2010_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2010_data.csv","manner_of_death")
mannerofdeath_2011_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2011_data.csv","manner_of_death")
mannerofdeath_2012_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2012_data.csv","manner_of_death")
mannerofdeath_2013_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2013_data.csv","manner_of_death")
mannerofdeath_2014_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2014_data.csv","manner_of_death")
mannerofdeath_2015_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2015_data.csv","manner_of_death")
```

▾ race word count 2005 2010 2015

```
[ ] race_2005_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2005_data.csv","race")
    race_2006_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2006_data.csv","race")
    race_2007_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2007_data.csv","race")
    race_2008_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2008_data.csv","race")
    race_2009_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2009_data.csv","race")
    race_2010_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2010_data.csv","race")
    race_2011_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2011_data.csv","race")
    race_2012_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2012_data.csv","race")
    race_2013_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2013_data.csv","race")
    race_2014_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2014_data.csv","race")
    race_2015_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2015_data.csv","race")
```

▾ resident status word count 2005 2010 2015

```
[ ] resident_status_2005_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2005_data.csv","resident_status")
    resident_status_2006_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2006_data.csv","resident_status")
    resident_status_2007_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2007_data.csv","resident_status")
    resident_status_2008_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2008_data.csv","resident_status")
    resident_status_2009_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2009_data.csv","resident_status")
    resident_status_2010_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2010_data.csv","resident_status")
    resident_status_2011_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2011_data.csv","resident_status")
    resident_status_2012_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2012_data.csv","resident_status")
    resident_status_2013_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2013_data.csv","resident_status")
    resident_status_2014_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2014_data.csv","resident_status")
    resident_status_2015_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2015_data.csv","resident_status")
```

▾ place of death word count

```
[ ] place_death_2005_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2005_data.csv","place_of_death_and_decedents_status")
    place_death_2006_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2006_data.csv","place_of_death_and_decedents_status")
    place_death_2007_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2007_data.csv","place_of_death_and_decedents_status")
    place_death_2008_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2008_data.csv","place_of_death_and_decedents_status")
    place_death_2009_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2009_data.csv","place_of_death_and_decedents_status")
    place_death_2010_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2010_data.csv","place_of_death_and_decedents_status")
    place_death_2011_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2011_data.csv","place_of_death_and_decedents_status")
    place_death_2012_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2012_data.csv","place_of_death_and_decedents_status")
    place_death_2013_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2013_data.csv","place_of_death_and_decedents_status")
    place_death_2014_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2014_data.csv","place_of_death_and_decedents_status")
    place_death_2015_word_count = Get_wordcount_specific_column ("/content/Death_Dataset/2015_data.csv","place_of_death_and_decedents_status")
```

6- To solve the problem that each file has its own map in json files. We implemented a function to convert from data frame into array

Function to convert dataframe to array

```
[ ]  def df_to_arr(df,column_name):
       count=df.toPandas()['count'].values
       arr=df.toPandas()[column_name].values
       return count,arr
```

Then we applied this function on all counting of all single variables

▾ Manner of death word count dataframe to array

```
[ ] mannerofdeath_count_array2005,mannerofdeath_values_array_2005=df_to_arr(mannerofdeath_2005_word_count,'manner_of_death')
    mannerofdeath_count_array2006,mannerofdeath_values_array_2006=df_to_arr(mannerofdeath_2006_word_count,'manner_of_death')
    mannerofdeath_count_array2007,mannerofdeath_values_array_2007=df_to_arr(mannerofdeath_2007_word_count,'manner_of_death')
    mannerofdeath_count_array2008,mannerofdeath_values_array_2008=df_to_arr(mannerofdeath_2008_word_count,'manner_of_death')
    mannerofdeath_count_array2009,mannerofdeath_values_array_2009=df_to_arr(mannerofdeath_2009_word_count,'manner_of_death')
    mannerofdeath_count_array2010,mannerofdeath_values_array_2010=df_to_arr(mannerofdeath_2010_word_count,'manner_of_death')
    mannerofdeath_count_array2011,mannerofdeath_values_array_2011=df_to_arr(mannerofdeath_2011_word_count,'manner_of_death')
    mannerofdeath_count_array2012,mannerofdeath_values_array_2012=df_to_arr(mannerofdeath_2012_word_count,'manner_of_death')
    mannerofdeath_count_array2013,mannerofdeath_values_array_2013=df_to_arr(mannerofdeath_2013_word_count,'manner_of_death')
    mannerofdeath_count_array2014,mannerofdeath_values_array_2014=df_to_arr(mannerofdeath_2014_word_count,'manner_of_death')
    mannerofdeath_count_array2015,mannerofdeath_values_array_2015=df_to_arr(mannerofdeath_2015_word_count,'manner_of_death')
```

▾ race word count dataframe to array

```
[ ] race_count_array2005,race_values_array_2005=df_to_arr(race_2005_word_count,'race')
    race_count_array2006,race_values_array_2006=df_to_arr(race_2006_word_count,'race')
    race_count_array2007,race_values_array_2007=df_to_arr(race_2007_word_count,'race')
    race_count_array2008,race_values_array_2008=df_to_arr(race_2008_word_count,'race')
    race_count_array2009,race_values_array_2009=df_to_arr(race_2009_word_count,'race')
    race_count_array2010,race_values_array_2010=df_to_arr(race_2010_word_count,'race')
    race_count_array2011,race_values_array_2011=df_to_arr(race_2011_word_count,'race')
    race_count_array2012,race_values_array_2012=df_to_arr(race_2012_word_count,'race')
    race_count_array2013,race_values_array_2013=df_to_arr(race_2013_word_count,'race')
    race_count_array2014,race_values_array_2014=df_to_arr(race_2014_word_count,'race')
    race_count_array2015,race_values_array_2015=df_to_arr(race_2015_word_count,'race')
```

▾ resident status word count dataframe to array

```
[ ] resident_status_count_array2005,resident_status_values_array_2005=df_to_arr(resident_status_2005_word_count,'resident_status')
    resident_status_count_array2006,resident_status_values_array_2006=df_to_arr(resident_status_2006_word_count,'resident_status')
    resident_status_count_array2007,resident_status_values_array_2007=df_to_arr(resident_status_2007_word_count,'resident_status')
    resident_status_count_array2008,resident_status_values_array_2008=df_to_arr(resident_status_2008_word_count,'resident_status')
    resident_status_count_array2009,resident_status_values_array_2009=df_to_arr(resident_status_2009_word_count,'resident_status')
    resident_status_count_array2010,resident_status_values_array_2010=df_to_arr(resident_status_2010_word_count,'resident_status')
    resident_status_count_array2011,resident_status_values_array_2011=df_to_arr(resident_status_2011_word_count,'resident_status')
    resident_status_count_array2012,resident_status_values_array_2012=df_to_arr(resident_status_2012_word_count,'resident_status')
    resident_status_count_array2013,resident_status_values_array_2013=df_to_arr(resident_status_2013_word_count,'resident_status')
    resident_status_count_array2014,resident_status_values_array_2014=df_to_arr(resident_status_2014_word_count,'resident_status')
    resident_status_count_array2015,resident_status_values_array_2015=df_to_arr(resident_status_2015_word_count,'resident_status')
```

▾ place of death word count dataframe to array

```
[ ] place_death_count_array2005,place_death_values_array_2005=df_to_arr(place_death_2005_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2006,place_death_values_array_2006=df_to_arr(place_death_2006_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2007,place_death_values_array_2007=df_to_arr(place_death_2007_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2008,place_death_values_array_2008=df_to_arr(place_death_2008_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2009,place_death_values_array_2009=df_to_arr(place_death_2009_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2010,place_death_values_array_2010=df_to_arr(place_death_2010_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2011,place_death_values_array_2011=df_to_arr(place_death_2011_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2012,place_death_values_array_2012=df_to_arr(place_death_2012_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2013,place_death_values_array_2013=df_to_arr(place_death_2013_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2014,place_death_values_array_2014=df_to_arr(place_death_2014_word_count,'place_of_death_and_decedents_status')
    place_death_count_array2015,place_death_values_array_2015=df_to_arr(place_death_2015_word_count,'place_of_death_and_decedents_status')
```

7- Then we created a function to map each csv file into its own json file

▾ Mapping from json and convert into CSV file

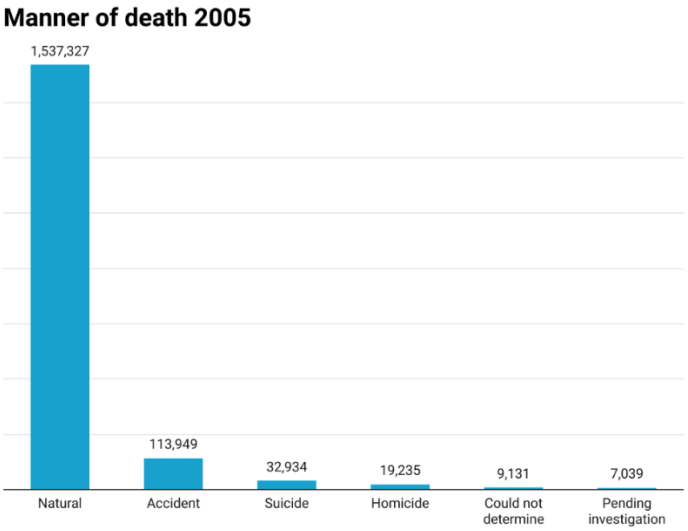
```
import csv
def convert_to_csv (json_file,name_of_column,count_of_this_column,data_before_mapping,outputted_csv):
  Mapped_array = []
  for i in data_before_mapping:
    for key in json_file[name_of_column]:
      if key == i:
        Mapped_array.append(json_file[name_of_column][key])

  header = [name_of_column,'count']
  with open(outputted_csv, 'w') as f:
    writer = csv.writer(f)
    writer.writerow(header)
    for i in range(len(Mapped_array)):
      row = [Mapped_array[i],count_of_this_column[i]]
      writer.writerow(row)
      row =[]
    f.close()
```

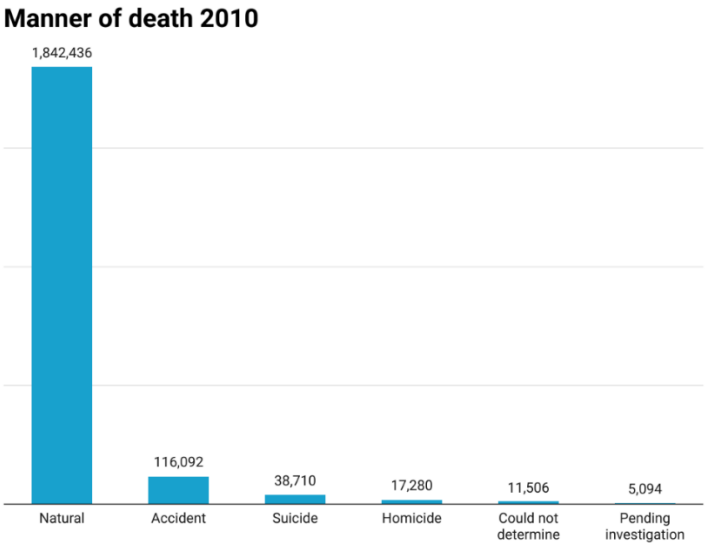
And then convert it into excel file to draw its chart later

Results

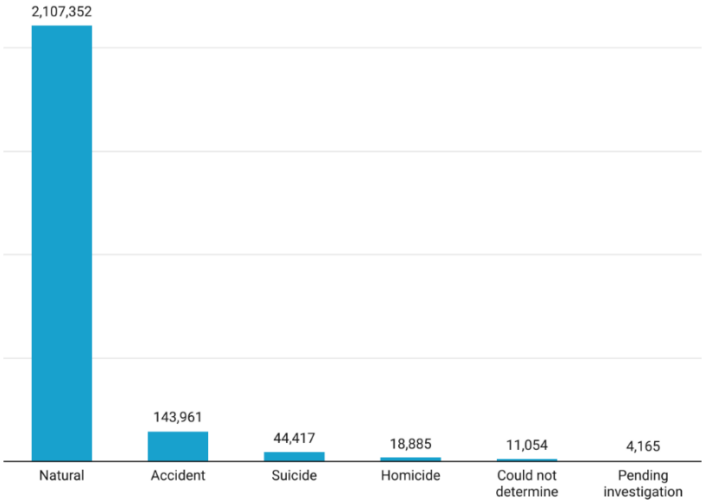
Single variable analysis and results

Manner of death variable



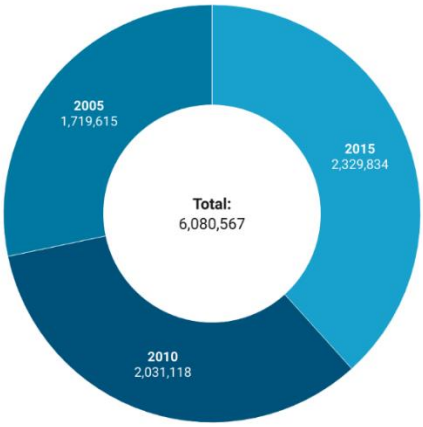


Manner of death 2015



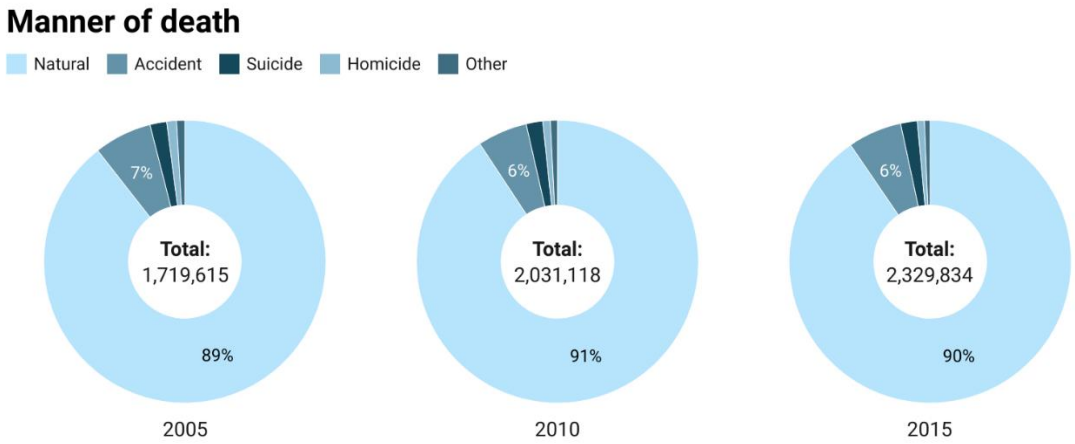
Created with Datawrapper

Manner of death count



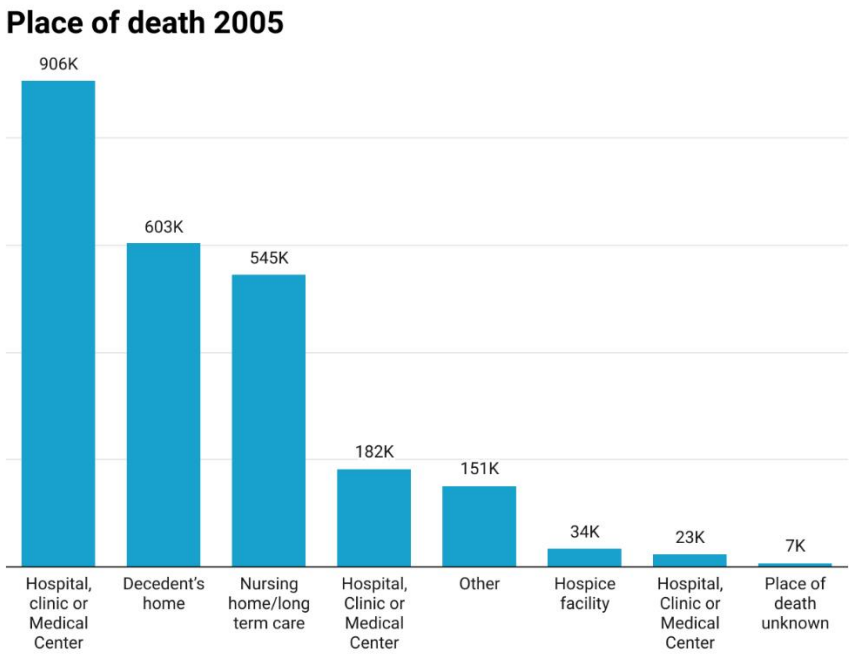
Created with Datawrapper

Race variable



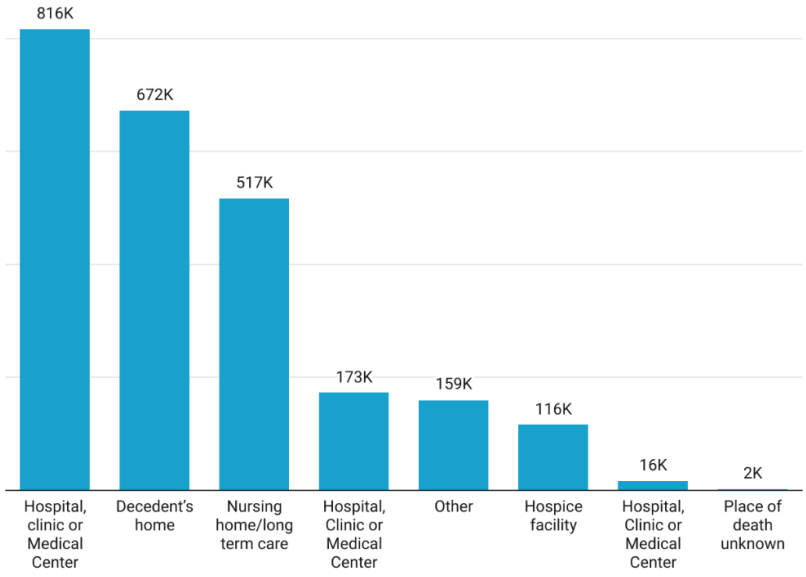
Created with Datawrapper

Place of death



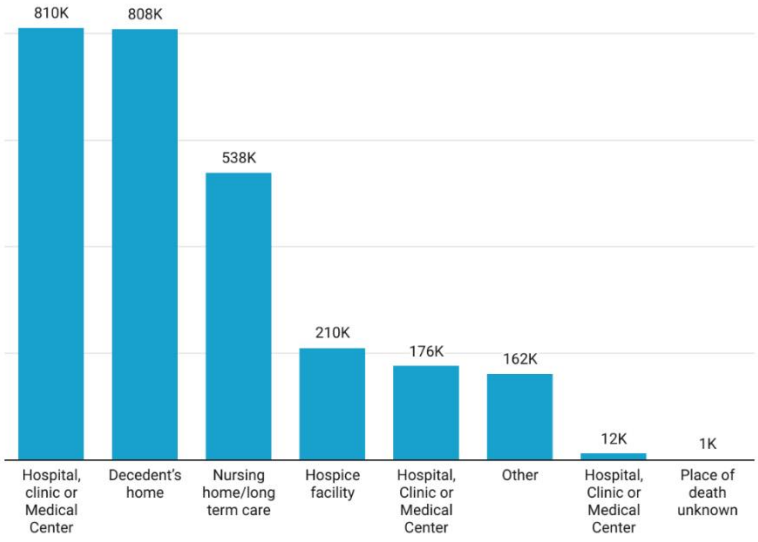
Created with Datawrapper

Place of death 2010



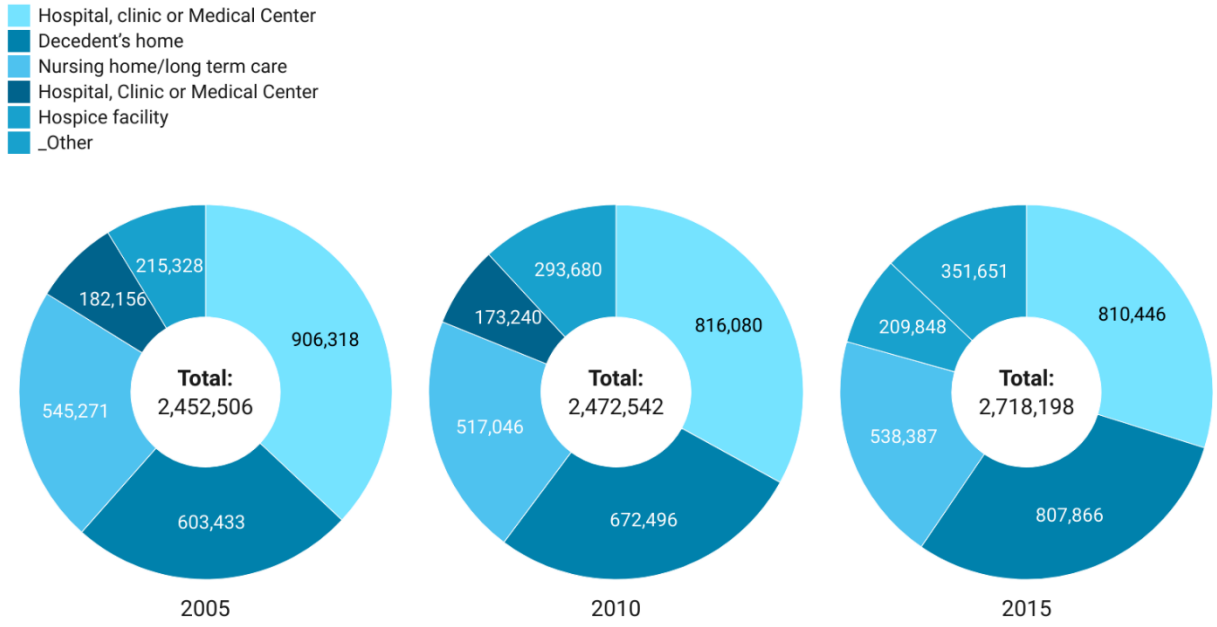
Created with Datawrapper

Place of death 2015



Created with Datawrapper

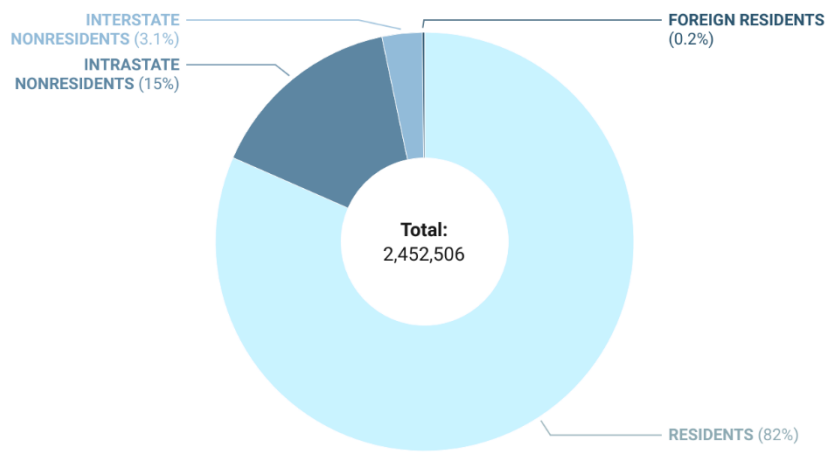
Place of death



Created with Datawrapper

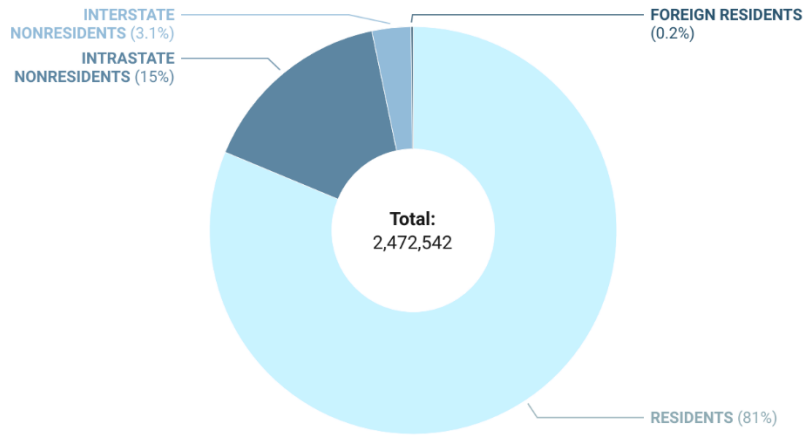
Resident status

Resident status 2005

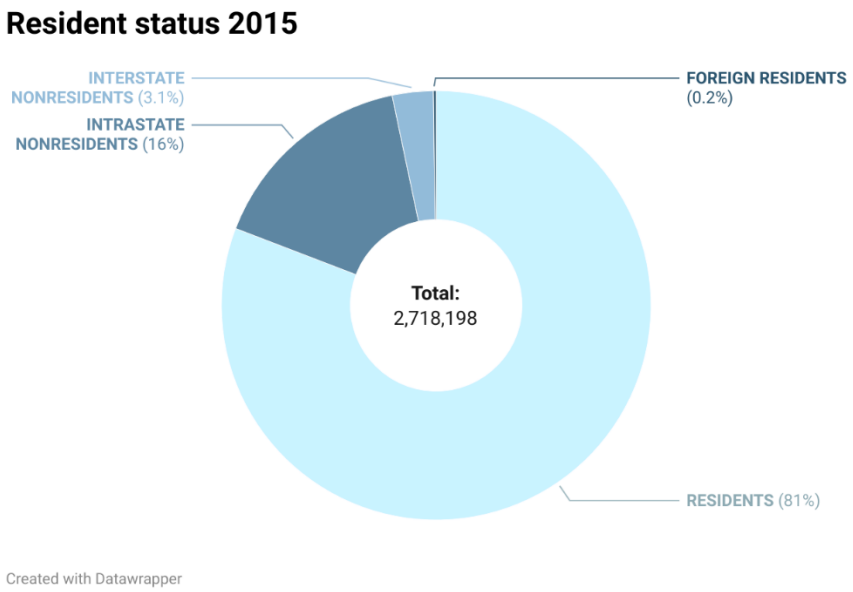


Created with Datawrapper

Resident status 2010

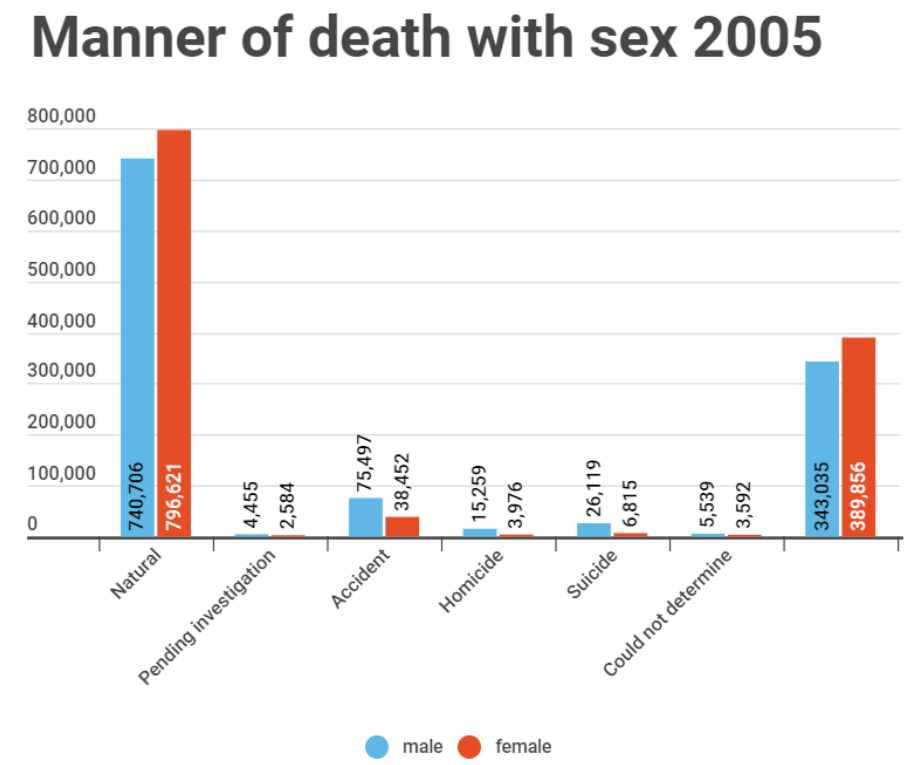


Created with Datawrapper

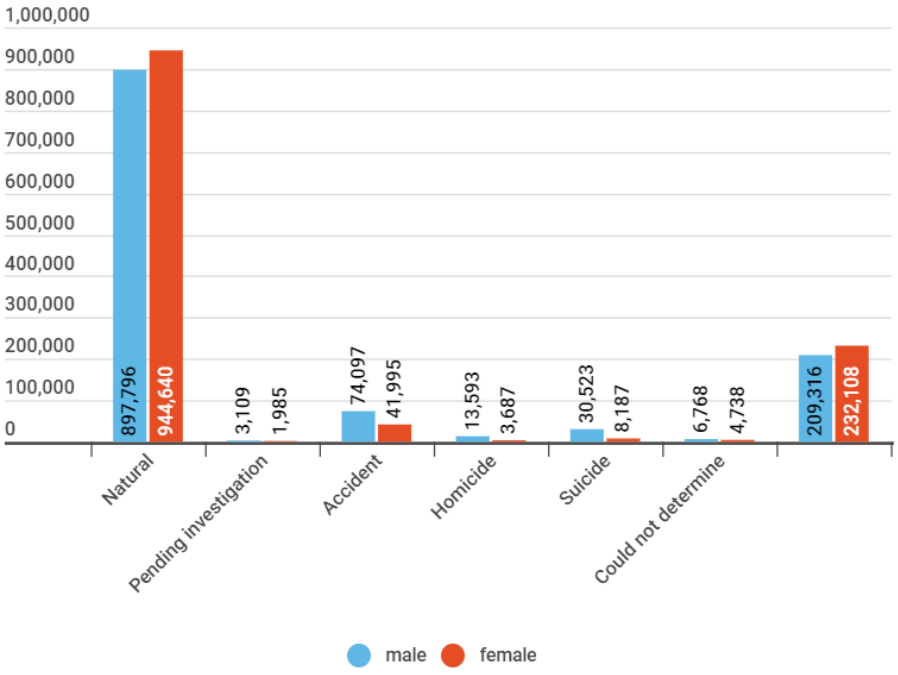


Multi variable analysis and results

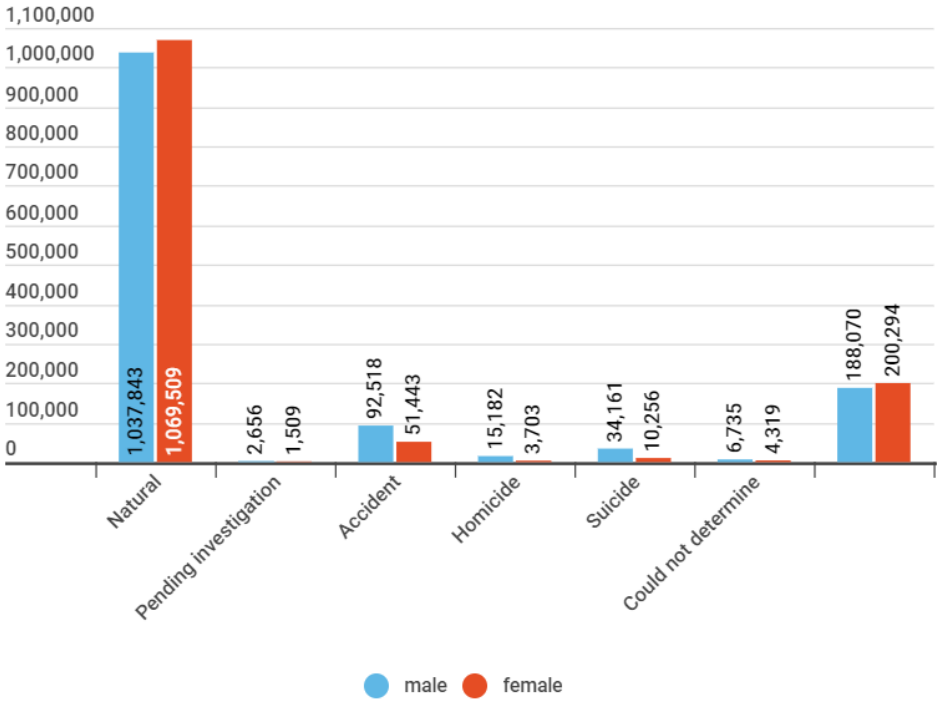
Manner of death with sex



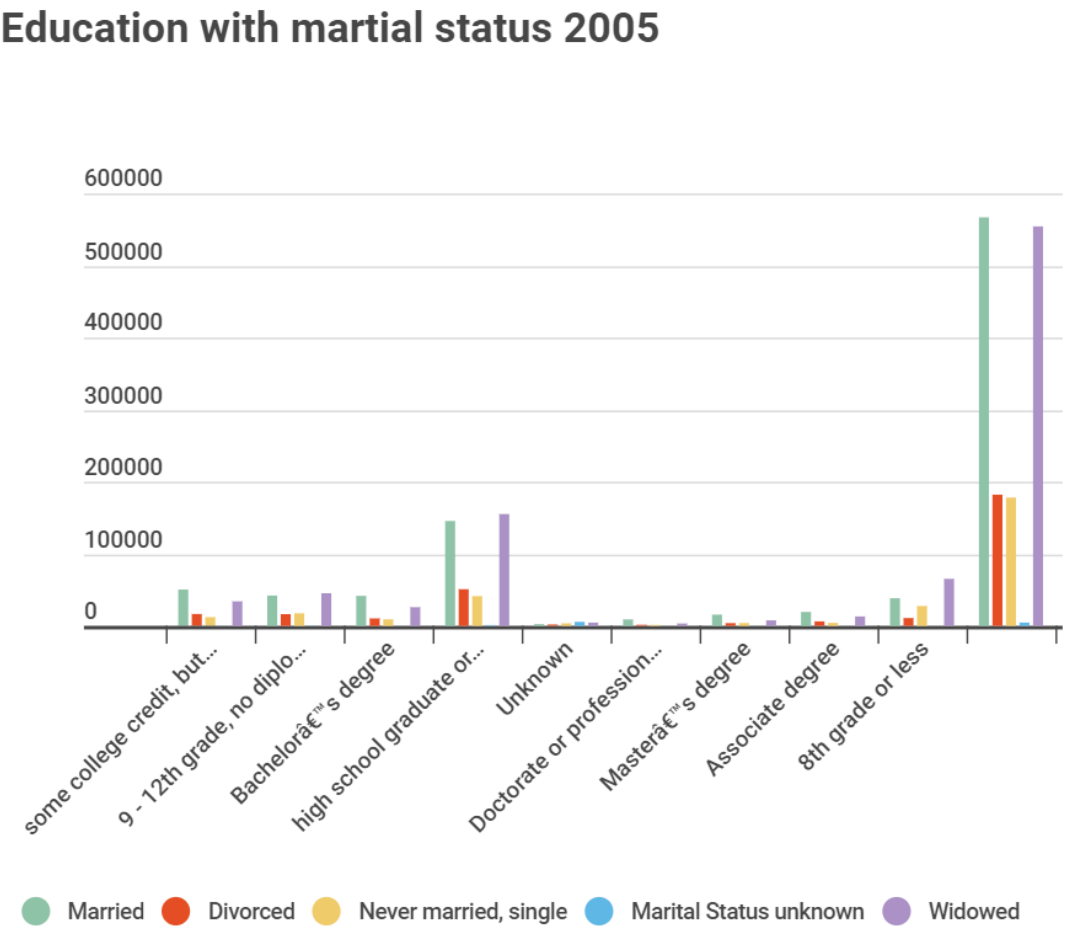
Manner of death with sex 2010



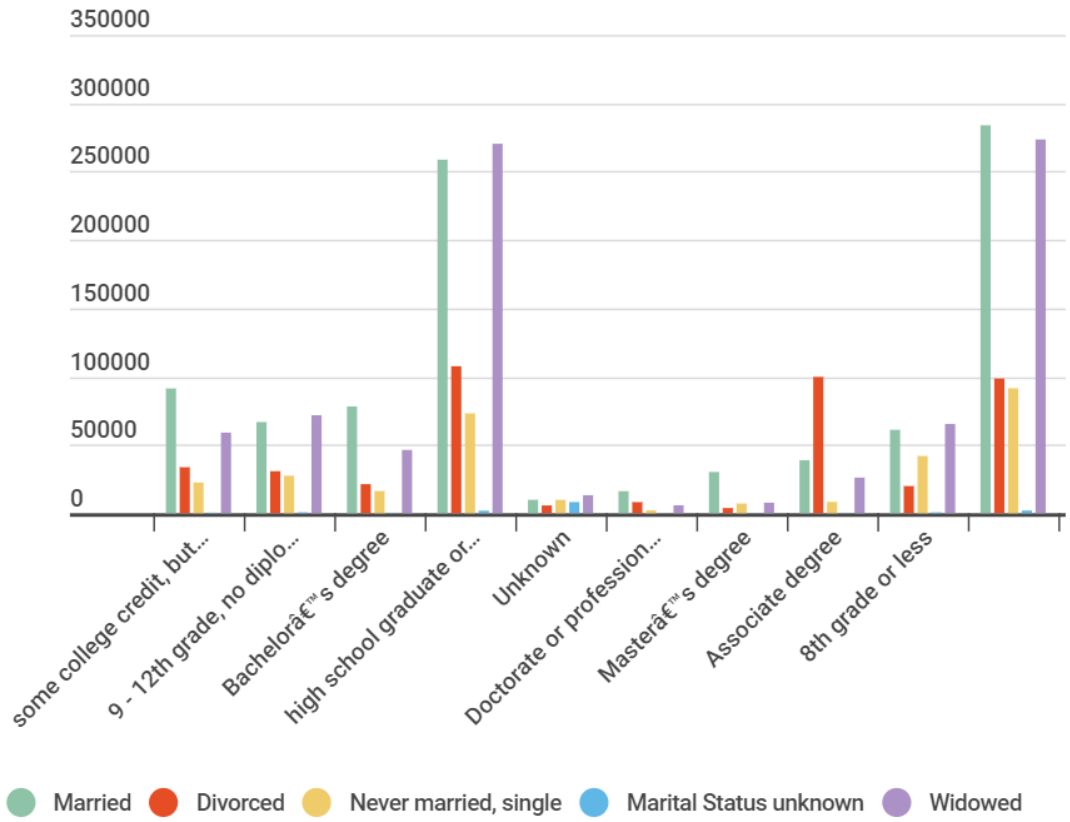
Manner of death with sex 2015



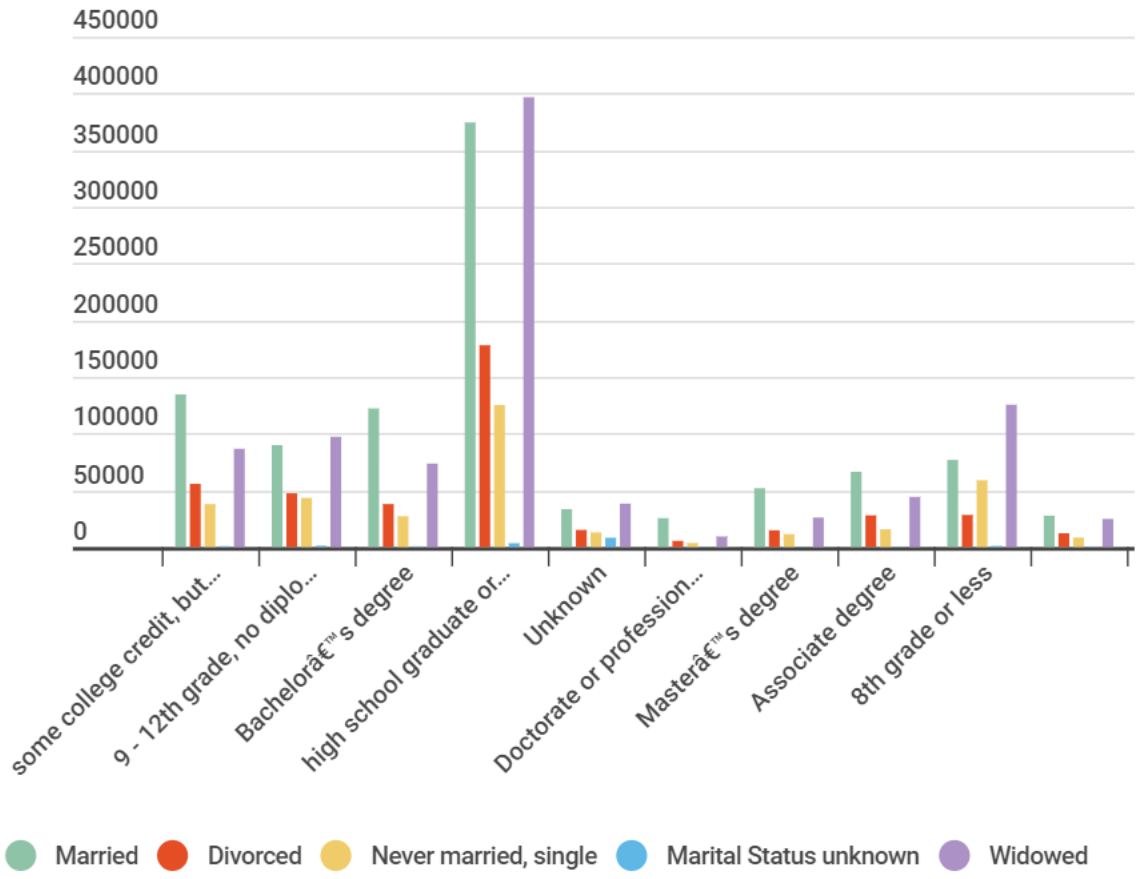
Education with marital status



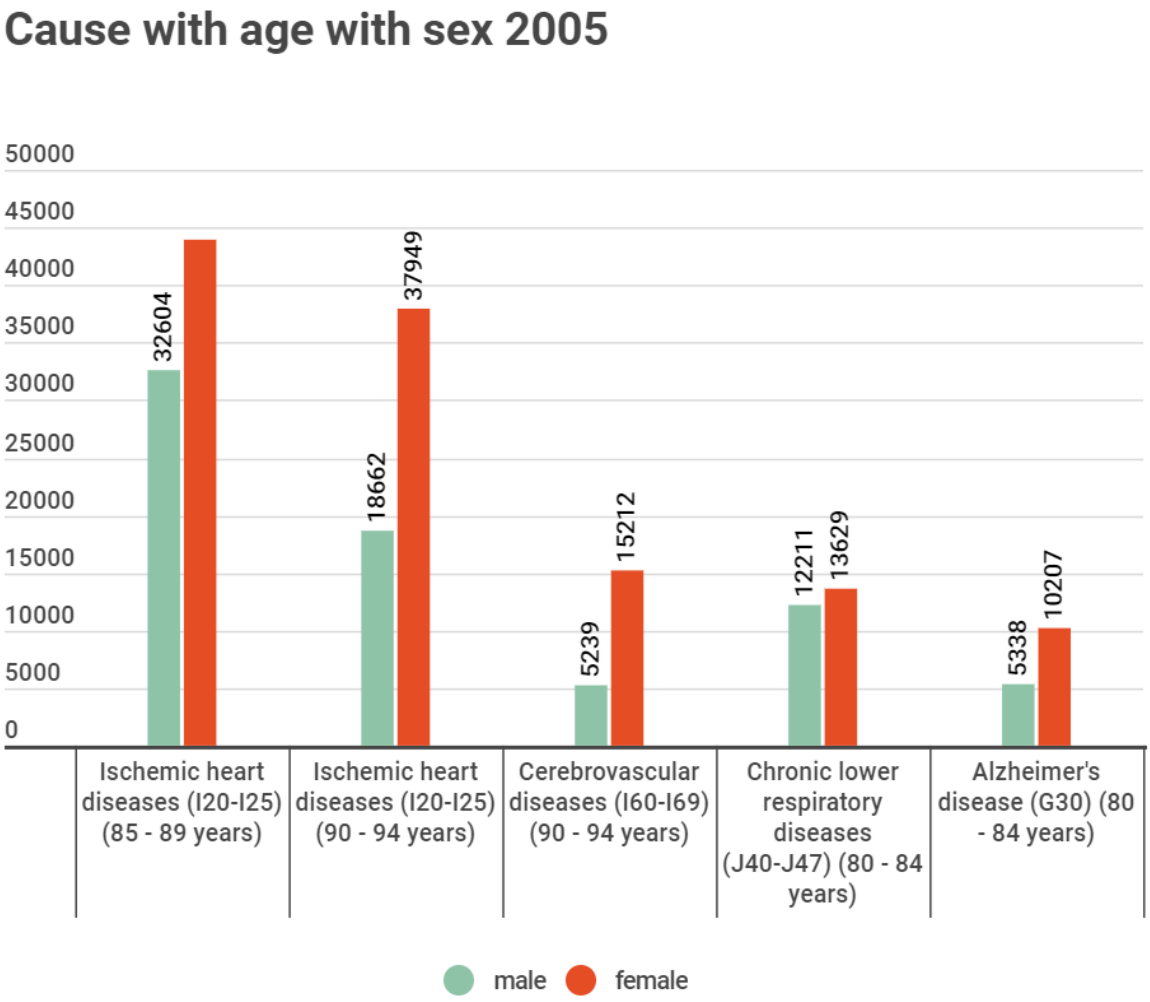
Education with martial status 2010



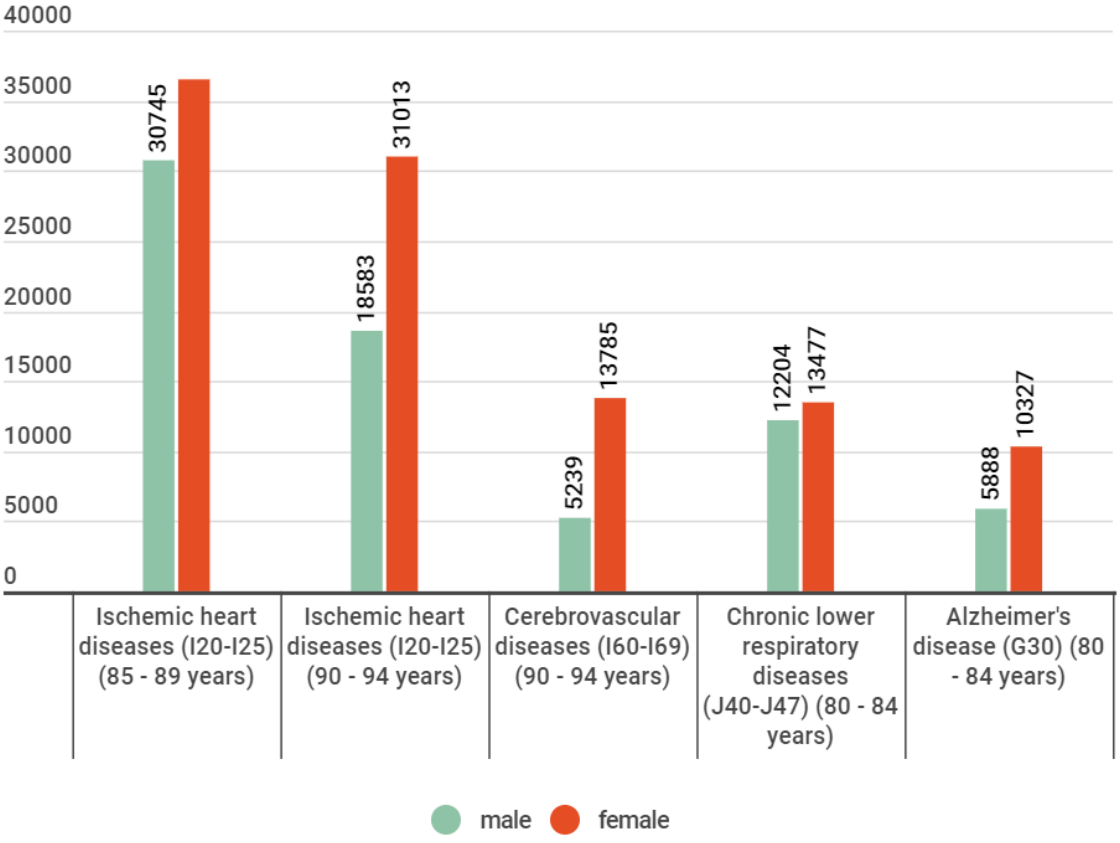
Education with martial status 2015

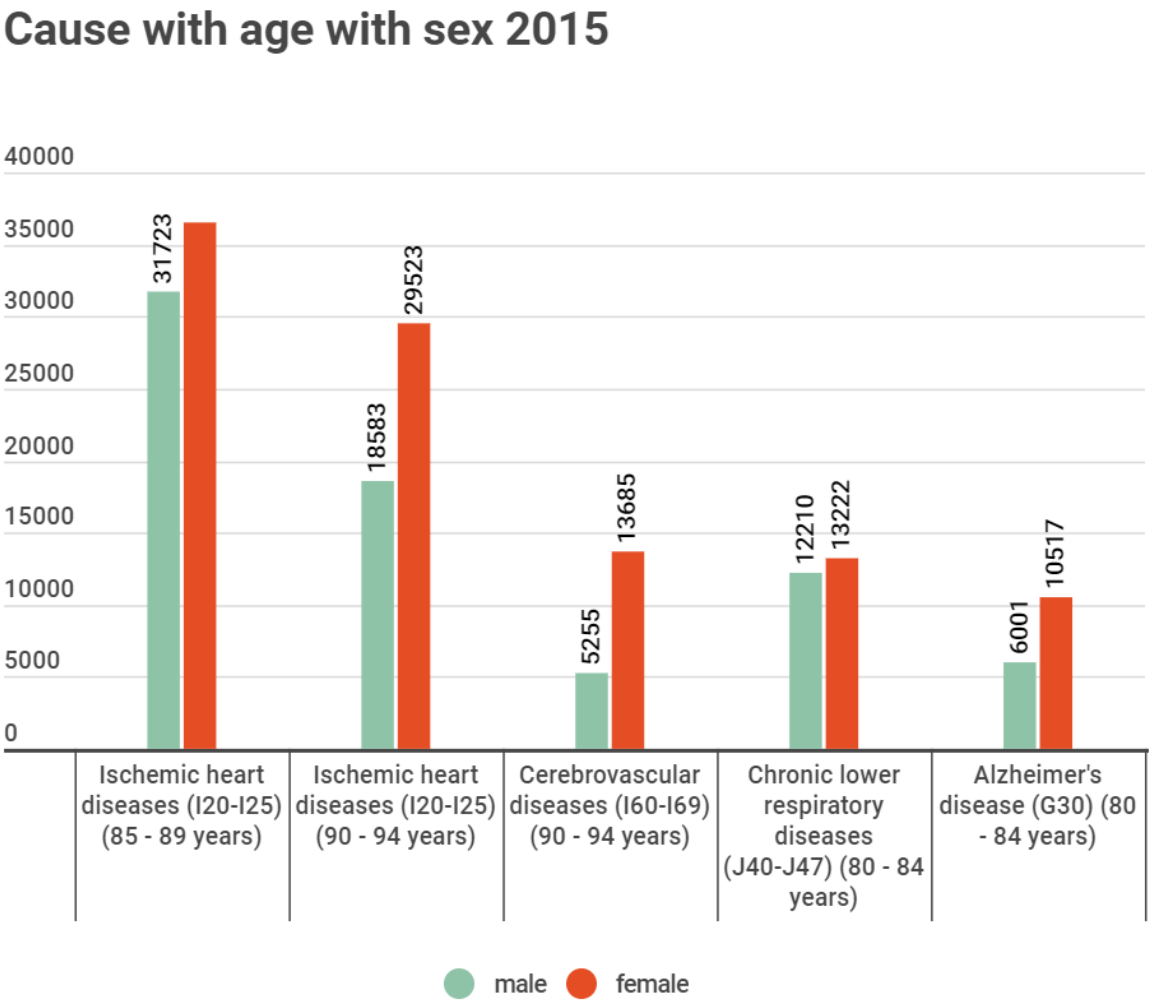


Causes with age with sex



Cause with age with sex 2010





Machine learning part

The feature columns: education, age, place of death, race cat, autopsy, marital status, resident status

Predicated column is Accident

1 means death with Accident 0 mean death with no Accident

Step 1:

Read the data for each and concatenate the desired years

```
import pandas as pd

cols2 = ['detail_age','race_recode_3','resident_status','sex','marital_status'
         ,'education_2003_revision',
           'place_of_death_and_decedents_status','injury_at_work',
        'autopsy','39_cause_recode']
data2_2015 = pd.read_csv('/content/Death_Dataset/2015_data.csv',usecols=cols2)
data2_2014 = pd.read_csv('/content/Death_Dataset/2014_data.csv',usecols=cols2)
data2_2013 = pd.read_csv('/content/Death_Dataset/2013_data.csv',usecols=cols2)
data2_2012 = pd.read_csv('/content/Death_Dataset/2012_data.csv',usecols=cols2)
data2_2011 = pd.read_csv('/content/Death_Dataset/2011_data.csv',usecols=cols2)
data2_2010 = pd.read_csv('/content/Death_Dataset/2010_data.csv',usecols=cols2)

data_df2 = pd.concat([data2_2015,data2_2014])
data_df2 = data_df2.dropna(how = 'any')
```

```
59] cols_df2 = data_df2.copy()
   #Rename columns.
   cols_df2.columns = ['resident_status','education','sex','age','place_of_death'
                       ,'marital_status','injury_at_work','autopsy','Accidents','race_CAT']
```

Step2:

Make the coding manually so we could be input to the classifier

```
+ Code  + Text

[60] #----------Brief cleansing of data.
     #Manipulate target value.
     y_Ac = cols_df2['Accidents']
     y_Ac.loc[(y_Ac == 38)|(y_Ac == 39)] = 1.0
     y_Ac.loc[y_Ac != 1] = 0.0
     cols_df2 = cols_df2.drop(['Accidents'],axis = 1)
     cols_df2 = pd.concat([cols_df2,y_Ac],axis = 1)

     #Recode resident status to resident or not (1/0).
     cols_df2['resident_status'].loc[cols_df2['resident_status'] == 1] = '1'
     cols_df2['resident_status'].loc[cols_df2['resident_status'] != '1'] = '0'

     #Recode autopsy to (1/0).
     cols_df2['autopsy'].loc[cols_df2['autopsy'] == 'Y'] = '1'
     cols_df2['autopsy'].loc[cols_df2['autopsy'] == 'N'] = '0'
     '''
     #Recode Sex to (1/0).
     cols_df2['sex'].loc[cols_df2['sex'] == 'M'] = '1'
     cols_df2['sex'].loc[cols_df2['sex'] == 'F'] = '0'

     #Recode martial status to (1/0).
     cols_df2['marital_status'].loc[cols_df2['marital_status'] == 'M'] = '1'
     cols_df2['sex'].loc[cols_df2['sex'] == 'F'] = '0'
     '''


     #Recode injury_at_work to (1/0).
     cols_df2['injury_at_work'].loc[cols_df2['injury_at_work'] == 'Y'] = '1'
     cols_df2['injury_at_work'].loc[cols_df2['injury_at_work'] == 'N'] = '0'
     '''
able
```

Step3:

Make the pipeline and fit on the data after the transformation (create feature vectors)

```
[64] from pyspark.ml.feature import VectorAssembler

numericCols = ['resident_status','injury_at_work','autopsy','education','age','place_of_death','race_CAT']
assembler = VectorAssembler(inputCols=numericCols, outputCol="features")
df = assembler.transform(sparkDF)
```

```
from pyspark.ml.feature import StringIndexer
label_stringIdx = StringIndexer(inputCol = 'Accidents', outputCol = 'labelIndex')
df = label_stringIdx.fit(df).transform(df)
```

Step4:

Use randomForest classifier to fit on training data and start classify

```
43] from pyspark.ml.classification import RandomForestClassifier
    train, test = df.randomSplit([0.7, 0.3], seed = 2018)
    rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'labelIndex')
    rfModel = rf.fit(train)
    predictions = rfModel.transform(test)
```

```
44] predictions.select('education', 'age', 'place_of_death', 'race_CAT', 'labelIndex', 'rawPrediction', 'prediction', 'probability').show(25)
```

education	age	place_of_death	race_CAT	labelIndex	rawPrediction	prediction	probability
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	1.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	0.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	0.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	0.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	0.0	[13.2106743432348...	0.0	[0.66053371716174...
1.0	1	1	1	0.0	[13.2106743432348...	0.0	[0.66053371716174...

Step5:

Evualution of randomForest on our data

```
[45] from pyspark.ml.evaluation import MulticlassClassificationEvaluator
     evaluator = MulticlassClassificationEvaluator(labelCol="labelIndex", predictionCol="prediction")
     accuracy = evaluator.evaluate(predictions)
     print("Accuracy = %s" % (accuracy))
     print("Test Error = %s" % (1.0 - accuracy))

Accuracy = 0.6334206944135473
Test Error = 0.3665793055864527
```