CS109A Introduction to Data Science

Final Project

Harvard University
Fall 2023
Instructors: Pavlos Protopapas and Kevin Rader

Project Title: Predicting US Population General, Physical and Mental Health Outcomes

Members:
Luning (Alice) Wang (luningwang1022@gmail.com)
Ji Soo Han (hjs2050@gmail.com)
Christophe Sterckx (sterckx.c@gmail.com)
Brendon Gory (brg294@g.harvard.edu)

Table of Contents

Abstract

What has been the trend of general health in the US Population from 2018 to 2022? Obvious factors like COVID-19 lockdown in 2020-2021 would decrease overall general

health as it affected millions physically and mentally. However, over the span of five years, medical advances have improved people's lives in the U.S. Notably, the U.S. has instituted a mental health crisis hotline in 2022[1]. Though this was launched in 2022 and may not influence the survey results, it is a recent example of possible improvements.

U.S. alcohol consumption has remained constant over the past few decades[2], however U.S. obesity has been consistenly increasing over the years[3]. We will study the several variables on their impact on general health. These factors include but are not limited to alcohol consumption, BMI, gender, age, location and income. If the general health has fluctuated through the period of 2018 to 2022, what factors contributed the most based on the chosen variables. Can these variables then be used to predict an individual's general health if a model is trained on the dataset?

This project aims to answer if a trend can be seen regarding general health over the five years. We hope to show which factors have the most impact on general health. If the repsonse variable General Health spikes or dips for some years - holding all variables constant - perhaps there is an exogeneous factor, such as elections, global conflict or COVID, that contribute the general health over the questions asked in the survey.

[1]The new 988 mental health hotline is live

[2]What Percentage of Americans Drink Alcohol?

[3]NIH: Overweight & Obesity Statistics

Introduction

The data were collected from https://www.cdc.gov/brfss (CDC: Behavioral Risk Factor Surveillance System). The agency polls about 400,000 U.S. residents each year asking them questions over a landline or cellular phone. The agency records the age, gender at birth, location, race and other personal information. All participants' data are anonymous; they cannot be traced to a specific individual. Not every question received a response. This could be because the individual does not know the answer, they refuse to answer the question or they did not understand the answer. The questions are asked in English and Spanish, so it is conceivable the participant does not speak either language well enough for an informed answer.

Each feature in the dataset is an answer to a question. The majority of the answers are categorical. For example, income is a level indicator and not the actual income. Some values, like BMI, are calculated by the CDC since is a number not frequently known by someone. The BMI is calculated by height and weight. Again, this is put into a category without the numerical data stored. For the years starting from 2018 to 2022, there are 2,118,822 records. Each record is an individual who answered questions over a phone

interview.

We will evaluate the following features from the dataset: `Date, State, Gender, Age Group, Race, BMI Category, Physical Activity, Pregnant, Drinking, Smoking, Education Category, Employment, Income Group, Veteran, Marital Status, Rent Home, Insurance`. Some feature names have changed over the years, but the information they captured remained the same. A script preprocessed the years to find the common features and renamed what are the same features with different labels. The appendix gives a description and range of the features' values.

Analysis

Load libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import py7zr
import os
from sklearn.utils import resample

import warnings
warnings.filterwarnings("ignore")

def pretty_df(title, in_dict, col_names, cols=3, rows=0):
    print(f'{title}', end='')

    df = pd.DataFrame(columns=col_names * int(cols / 2))

    # If number of rows isn't specfied, print all rows
    print_last = False
    if rows == 0:
        print_last = True
        rows = len(in_dict.keys())

    val_list = []
    row_count = 0
    for idx, val in enumerate(in_dict):
        if (idx * 2) % cols == 0 and idx > 0:
            df.loc[row_count] = val_list
            val_list = []
            row_count += 1
            if row_count == rows:
                break

        val_list.extend([str(val), str(in_dict[val])])
```

```
    if val_list != '' and print_last == True:
        while len(val_list) < cols:
            # Pad list length to match column length in data frame
            val_list.extend(['', ''])
        df.loc[row_count] = val_list

    display(df.style.hide(subset=None))
```

EDA

Load data file

In [2]:
```
if not os.path.exists('full_dataset.csv'):
    with py7zr.SevenZipFile('full_dataset.7z', 'r') as z:
        z.extract()

df = pd.read_csv('full_dataset.csv')
print(f'Data size: {df.shape}')
```

```
Data size: (2141487, 142)
```

Format date column to `yyyy-mm-dd`

For calculations or reshaping data, the date needs to be defined in the data frame. This allows for filtering on specific years or times of the year.

In [3]:
```
df['IDATE'] = pd.to_datetime(df['IDATE'])
display(df['IDATE'].head(10))
```

```
0   2018-01-05
1   2018-01-12
2   2018-01-08
3   2018-01-03
4   2018-01-12
5   2018-01-11
6   2018-01-10
7   2018-01-13
8   2018-01-09
9   2018-01-10
Name: IDATE, dtype: datetime64[ns]
```

Data types

In [4]:
```
dtypes = {}
for idx, col in enumerate(df.columns):
    dtypes[col] = df[col].dtype

dtypes = dict(sorted(dtypes.items()))
pretty_df('Features and Data Types:', dtypes, ['Feature', 'Data Type'], 10)
```

Features and Data Types:

Feature	Data Type	Feature	Data Type	Feature	Data Type	Feature	Dat Typ
ASTHMA3	float64	ASTHNOW	float64	BLIND	float64	CAREGIV1	float6
CASTHNO2	float64	CCLGHOUS	float64	CDASSIST	float64	CDDISCUS	float6
CDHOUSE	float64	CDSOCIAL	float64	CELLFON5	float64	CHECKUP1	float6
CHKHEMO3	float64	CIMEMLOS	float64	CNCRAGE	float64	CNCRDIFF	float6
CRGVEXPT	float64	CRGVHRS1	float64	CRGVLNG1	float64	CSRVCLIN	float6
CSRVDOC1	float64	CSRVINSR	float64	CSRVINST	float64	CSRVPAIN	float6
CSRVSUM	float64	CSTATE1	float64	CTELENM1	float64	CTELNUM1	float6
CVDINFR4	float64	CVDSTRK3	float64	DEAF	float64	DECIDE	float6
DIFFDRES	float64	DIFFWALK	float64	DISPCODE	float64	DRNK3GE5	float6
EMPLOY1	float64	EXERANY2	float64	EYEEXAM1	float64	FMONTH	float6
GENHLTH	float64	HADHYST2	float64	HADMAM	float64	HEIGHT3	float6
HIVTSTD3	float64	HOWLONG	float64	HPVADSHT	float64	HTIN4	float6
IDATE	datetime64[ns]	IDAY	object	IMONTH	object	INCOME3	float6
LANDLINE	float64	LASTSMK2	float64	LCSFIRST	float64	LCSLAST	float6
MARIJAN1	float64	MARITAL	float64	MAXDRNKS	float64	MENTHLTH	float6
NUMADULT	float64	NUMMEN	float64	NUMWOMEN	float64	PHYSHLTH	float6
POORHLTH	float64	PREGNANT	float64	PSATEST1	float64	PVTRESD1	float6
QSTLANG	float64	QSTVER	float64	RCSRLTN2	float64	RENTHOM1	float6
SEQNO	object	SHINGLE2	float64	SMOKDAY2	float64	SMOKE100	float6
SOMALE	float64	STATERE1	float64	STOPSMK2	float64	TETANUS1	float6
USENOW3	float64	VETERAN3	float64	WEIGHT2	float64	WTKG3	float6
_AGE80	float64	_AGEG5YR	float64	_AGE_G	float64	_ASTHMS1	float6
_BMI5CAT	float64	_CASTHM1	float64	_CHISPNC	float64	_CHLDCNT	float6
_DUALCOR	float64	_DUALUSE	float64	_EDUCAG	float64	_HISPANC	float6
_IMPRACE	float64	_LLCPWT	float64	_LLCPWT2	float64	_LTASTH1	float6
_METSTAT	float64	_MICHD	float64	_PHYS14D	float64	_PNEUMO3	float6
_RAWRAKE	float64	_RFBMI5	float64	_RFDRHV8	float64	_RFHLTH	float6
_SMOKER3	float64	_STATE	float64	_STRWT	float64	_STSTR	float6
_URBSTAT	float64	_WT2RAKE	float64				

Drop all features except for the variables used in modeling

Of the the 142 total features, we have determined some to be highly correlated and some to have too many missing values to be of use.

In [5]:
```
column_mapping = {
    'IDATE':'Date',
    '_STATE':'State',
    'GENDER':'Gender(Male)',
    '_AGE_G':'Age Group',
    '_IMPRACE':'Race',
    '_BMI5CAT':'BMI Category',
    '_TOTINDA':'Physical Activity',
    'PREGNANT':'Pregnant',
    '_RFDRHV8':'Drinking',
    '_RFSMOK3':'Smoking',
    '_EDUCAG':'Education Category',
    'EMPLOY1':'Employment',
    'INCOME3':'Income Group',
    'VETERAN3':'Veteran',
    'MARITAL':'Marital Status',
    'RENTHOM1':'Rent Home',
    '_HLTHPLN':'Insurance',
    'GENHLTH':'General Health'
    }
# Rename columns using the column_mapping dictionary
df.rename(columns=column_mapping, inplace=True)

# Select only the columns to keep
columns_to_keep = column_mapping.values()  # Use values instead of keys
df = df[list(columns_to_keep)]

# Display the first 10 rows of the modified DataFrame
display(df.head(10))
```

	Date	State	Gender(Male)	Age Group	Race	BMI Category	Physical Activity	Pregnant	Drinking	S
0	2018-01-05	1.0	2.0	6.0	1.0	2.0	2.0	NaN	1.0	
1	2018-01-12	1.0	2.0	2.0	2.0	4.0	1.0	2.0	1.0	
2	2018-01-08	1.0	2.0	6.0	1.0	3.0	1.0	NaN	1.0	
3	2018-01-03	1.0	1.0	6.0	1.0	3.0	1.0	NaN	1.0	
4	2018-01-12	1.0	2.0	3.0	1.0	NaN	2.0	2.0	1.0	
5	2018-01-11	1.0	2.0	6.0	1.0	4.0	2.0	NaN	1.0	
6	2018-01-10	1.0	2.0	6.0	1.0	2.0	1.0	NaN	1.0	
7	2018-01-13	1.0	2.0	4.0	1.0	2.0	1.0	2.0	1.0	
8	2018-01-09	1.0	1.0	5.0	1.0	3.0	2.0	NaN	1.0	
9	2018-01-10	1.0	2.0	4.0	2.0	3.0	1.0	2.0	1.0	

High-level descriptions

In [6]:
```
df.describe()
```

Out[6]:

	Date	State	Gender(Male)	Age Group	Race
count	2141481	2.141487e+06	2.141487e+06	2.141487e+06	2.141487e+06
mean	2020-07-25 03:11:42.094485248	3.023119e+01	1.543624e+00	4.386690e+00	1.706860e+00
min	2018-01-02 00:00:00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
25%	2019-04-16 00:00:00	1.800000e+01	1.000000e+00	3.000000e+00	1.000000e+00
50%	2020-07-22 00:00:00	2.900000e+01	2.000000e+00	5.000000e+00	1.000000e+00
75%	2021-11-04 00:00:00	4.200000e+01	2.000000e+00	6.000000e+00	1.000000e+00
max	2023-02-21 00:00:00	7.800000e+01	9.000000e+00	6.000000e+00	6.000000e+00
std	NaN	1.592214e+01	5.215067e-01	1.605545e+00	1.450179e+00

In [7]:
```
df.shape
```

Out[7]: (2141487, 18)

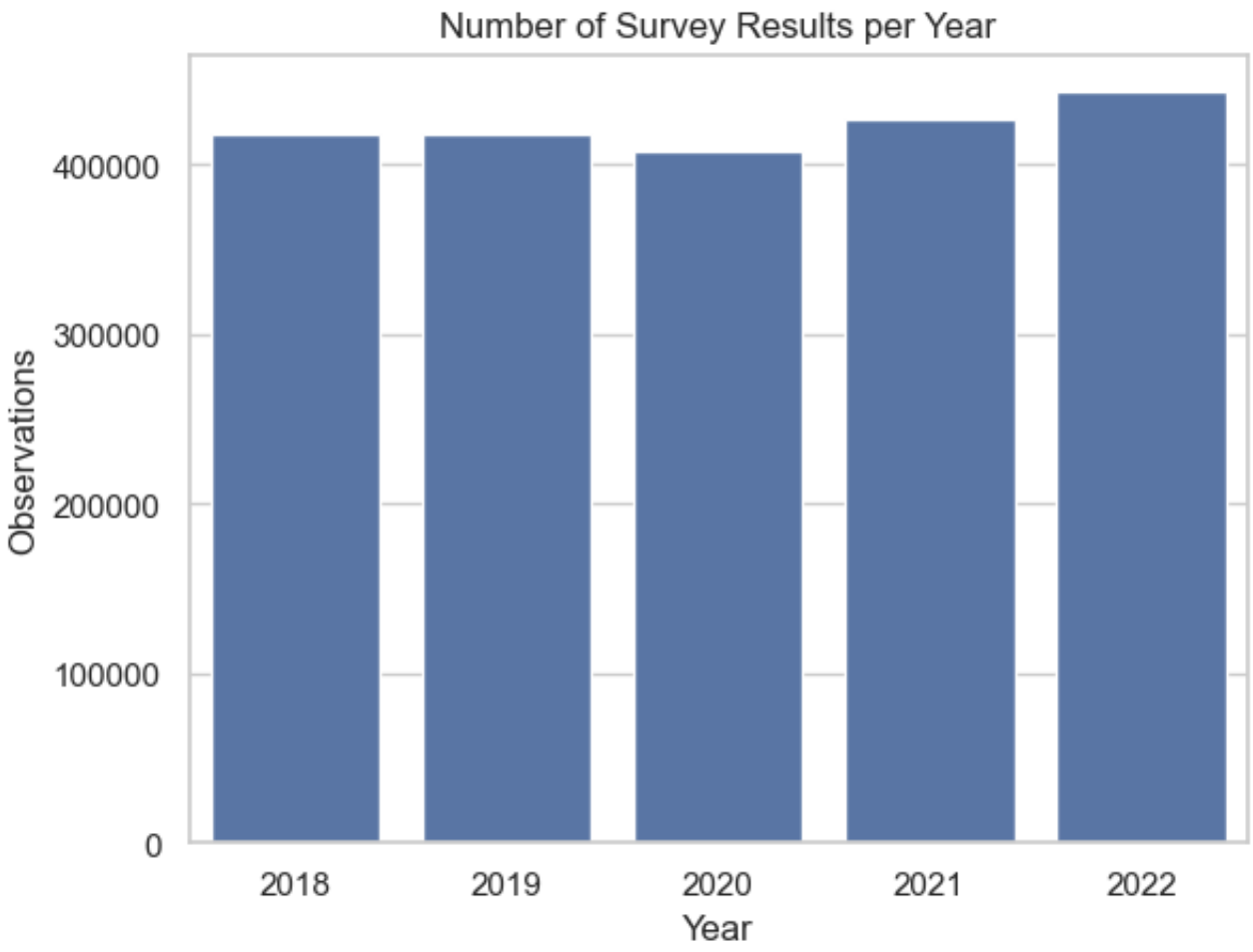
Drop observations with year = 2023

In [8]:
```
df = df[df['Date'].dt.year != 2023]
df.shape
```

Out[8]: (2115828, 18)

Observations per year

In [9]:
```
df_years = pd.DataFrame()
df_years['Year'] = df['Date'].dt.year
df_years = df_years.dropna()
df_years['Year'] = df_years['Year'].astype(int)

years, counts = np.unique(df_years['Year'], return_counts=True)

sns.set(style='whitegrid')
sns.barplot(x=years, y=counts)
plt.xlabel('Year')
plt.ylabel('Observations')
plt.title('Number of Survey Results per Year')
plt.show()
```

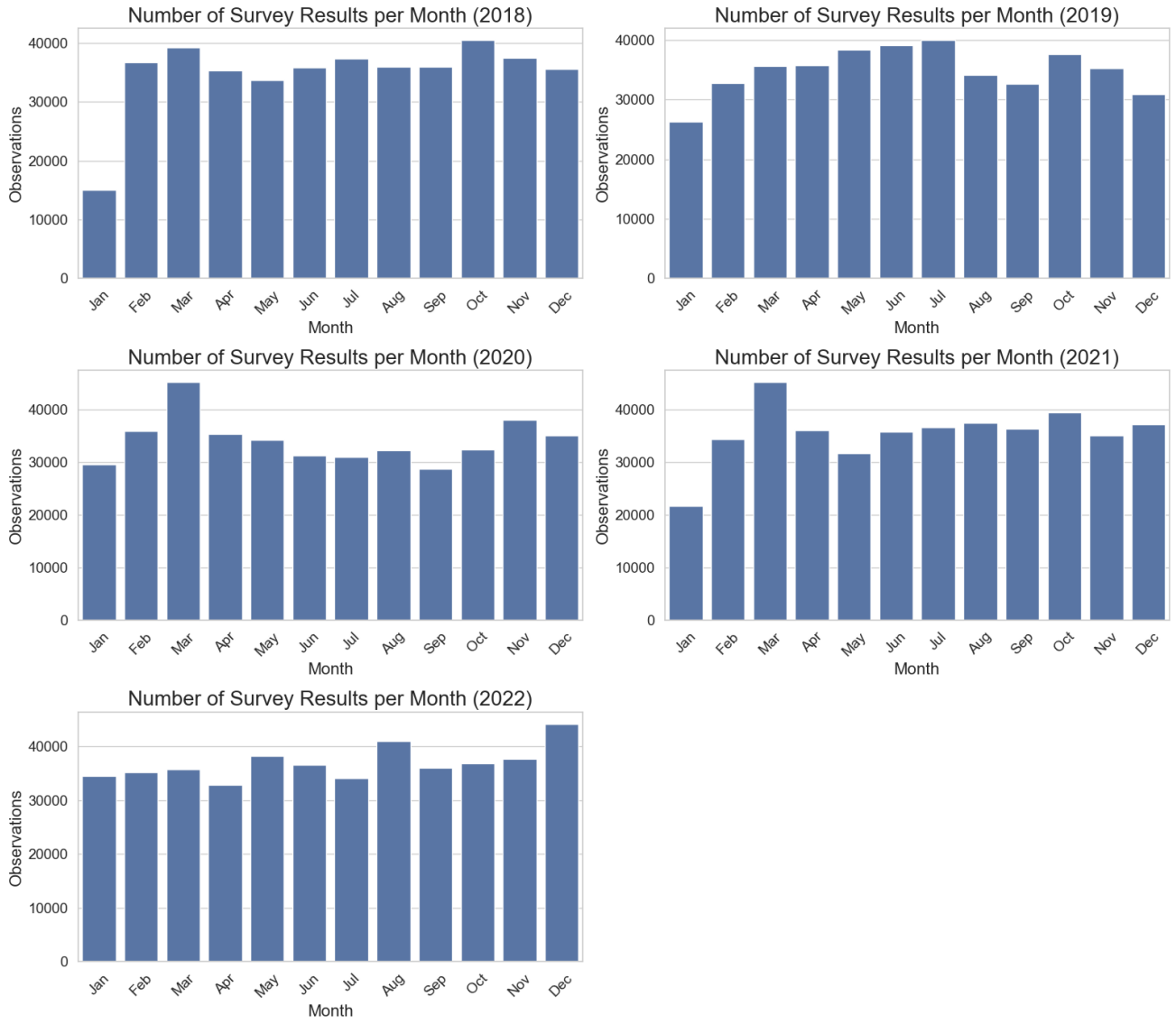


The CDC has a pretty even distribtuion of responses over the years. It seems to be a few percentage points difference among the years. Not a concern for training models on specific years and predicting on any chosen year.

Surveys by month per year

In [10]:
```
month_labels = {
    1: 'Jan', 2: 'Feb', 3: 'Mar',
    4: 'Apr', 5: 'May', 6: 'Jun',
    7: 'Jul', 8: 'Aug', 9: 'Sep',
    10: 'Oct', 11: 'Nov', 12: 'Dec'
}
fig, axs = plt.subplots(3, 2, figsize=(16, 14))
axs = axs.flatten()

for idx, year in enumerate(years):
    months, counts = np.unique(df['Date'][df['Date'].dt.year == year].dt.mor
    sns.set(style='whitegrid')
    sns.barplot(x=months, y=counts, ax=axs[idx])
    axs[idx].set_xlabel('Month', fontsize=16)
    axs[idx].set_ylabel('Observations', fontsize=16)
    axs[idx].set_title(f'Number of Survey Results per Month ({year})', fonts
    axs[idx].set_xticklabels([month_labels[month] for month in months], rota
    axs[idx].tick_params(axis='y', labelsize=14)
```

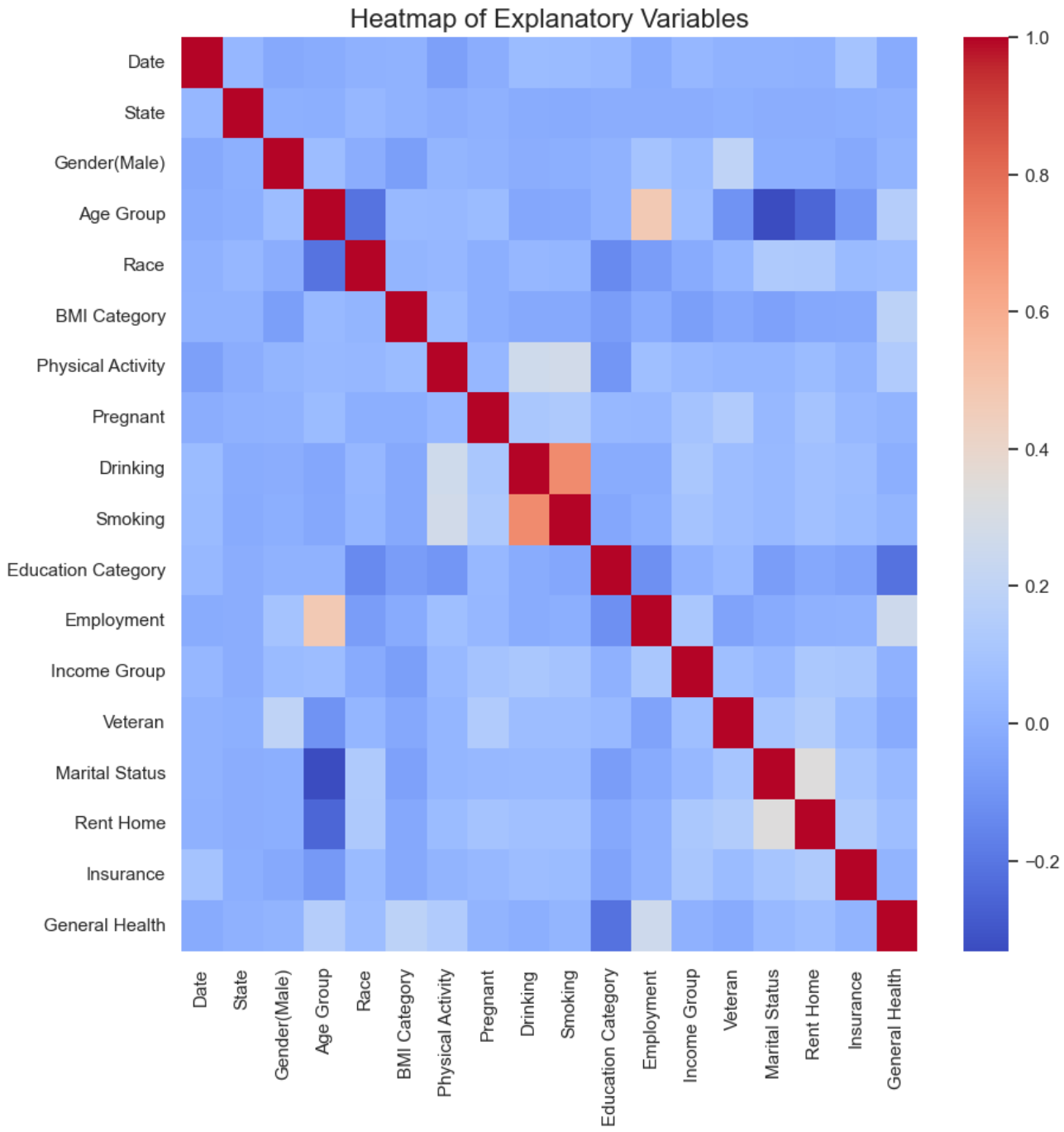
```
fig.delaxes(axs[-1])
plt.tight_layout()
plt.show()
```



These plots show even distribution of surveys conducted throughout the year. Important for consideration during 2020 when the US lockdown began in mid-March. The survey results should not be skewed to one season or time of the year where people may be in less or more general health.

Heatmap and pairplot

In [11]:
```
plt.figure(figsize=(10, 10))
corr = df.corr()
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, cmap='
plt.title('Heatmap of Explanatory Variables', fontsize=16)
plt.show()
```



We see a high correlation between `Smoking` and `Drinking` - the highest correlation between all features. The next strongest correlations are negative with `Marital Status` and `Rent Home` to `Age Group`. By the way the data were categorized with numeric values, younger people did not own homes as much as their older counterparts and the younger someone is, the more likely they will not be married. There are no other relationships between features to note.

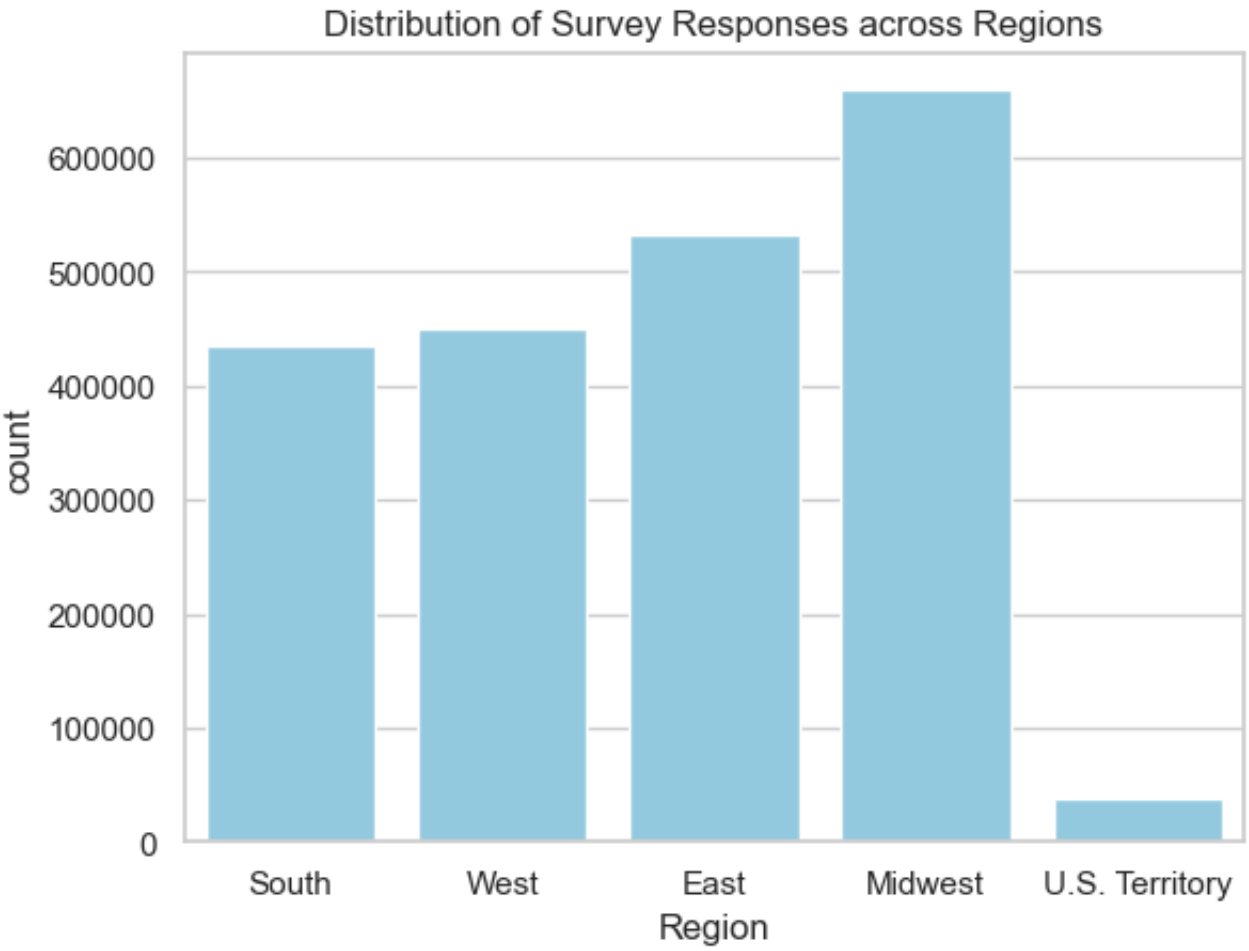
Creating regions for states

Too many values for states will not provide much insight from the models. By grouping 53 unique values into regional groups, we can reduce complexity in the model's interpretations. Understanding impacts of a region on genreal health, or general health in

a region suffices without diving into specific states.

In [12]:

```
"""
  1. Alabama, 2. Alaska,  4. Arizona,  5. Arkansas,  6. California,  8. Colo
  11. District of Columbia,  12. Florida,  13. Georgia,  15. Hawaii,  16. Id
  19. Iowa,  20. Kansas,  21. Kentucky,  22. Louisiana,  23. Maine,  24. Mar
  27. Minnesota,  28. Mississippi,  29. Missouri,  30. Montana,  31. Nebrask
  34. New Jersey,  35. New Mexico,  36. New York,  37. North Carolina,  38.
  41. Oregon,  42. Pennsylvania,  44. Rhode Island,  45. South Carolina,  46
  49. Utah,  50. Vermont,  51. Virginia,  53. Washington,  54. West Virginia
  66. Guam, 72. Puerto Rico,  78. Virgin Islands
"""
dict_regions = {
    1: 'South', 2: 'West', 4: 'West', 5: 'South', 6: 'West', 8: 'West', 9: '
    10: 'East', 11: 'East', 12: 'South', 13: 'South', 15: 'West', 16: 'West'
    17: 'Midwest', 18: 'Midwest', 19: 'Midwest', 20: 'Midwest', 21: 'South',
    23: 'East', 24: 'East', 25: 'East', 26: 'Midwest', 27: 'Midwest', 28: 'S
    29: 'Midwest', 30: 'Midwest', 31: 'Midwest', 32: 'West', 33: 'East', 34:
    35: 'West', 36: 'East', 37: 'South', 38: 'Midwest', 39: 'Midwest', 40: '
    41: 'West', 42: 'East', 44: 'East', 45: 'South', 46: 'Midwest', 47: 'Sou
    48: 'South', 49: 'West', 50: 'East', 51: 'South', 53: 'West', 54: 'Midwe
    55: 'Midwest', 56: 'West', 66: 'U.S. Territory', 72: 'U.S. Territory', 7
}
regions = df['State'].map(dict_regions)
df['Region'] = regions
sns.countplot(data=df, x='Region', color='skyblue')
plt.title('Distribution of Survey Responses across Regions')
plt.show()
```

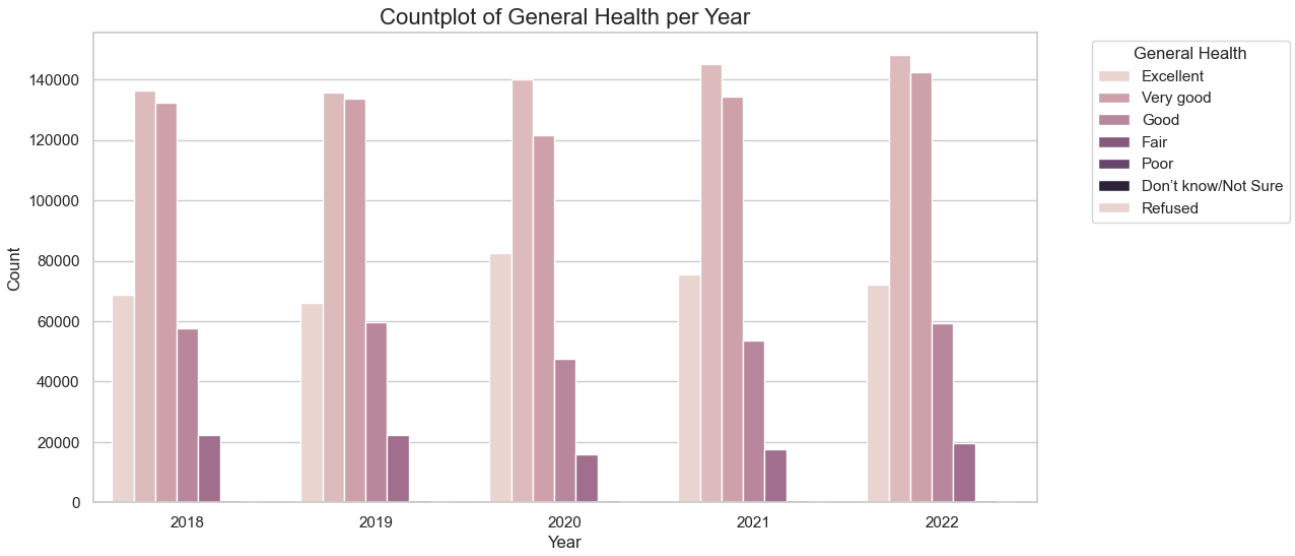


The highest populations in the US are in the Northeast (Philadelphia, New York, Washington, DC., Baltimore and Boston). The other large population location is California. However, the majority of the repsondents over the five year period was in the Midwest. There is no clear-cut defintion for these regions, i.e. which state belongs in which region. Interesting to note that the Midwest had about 50% more respondents than the West region.

General health values from 2018 to 2022

In [13]:
```
health_labels = {
    1: 'Excellent',
    2: 'Very good',
    3: 'Good',
    4: 'Fair',
    5: 'Poor',
    7: 'Don’t know/Not Sure',
    9: 'Refused'
}

df_health_year = pd.DataFrame(columns=['Year', 'General Health'])
df_health_year['Year'] = df['Date'].dt.year
df_health_year['General Health'] = df['General Health']
df_health_year = df_health_year.dropna()
```

```
df_health_year['Year'] = df_health_year['Year'].astype(int)
df_health_year['General Health'] = df_health_year['General Health'].astype(i
np.unique(df_health_year['General Health'])

plt.figure(figsize=(12, 6))
sns.countplot(x='Year', hue='General Health', data=df_health_year)
plt.title('Countplot of General Health per Year', fontsize=16)
plt.xlabel('Year', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.legend(title='General Health', labels=health_labels.values(), bbox_to_ar
plt.show()
```

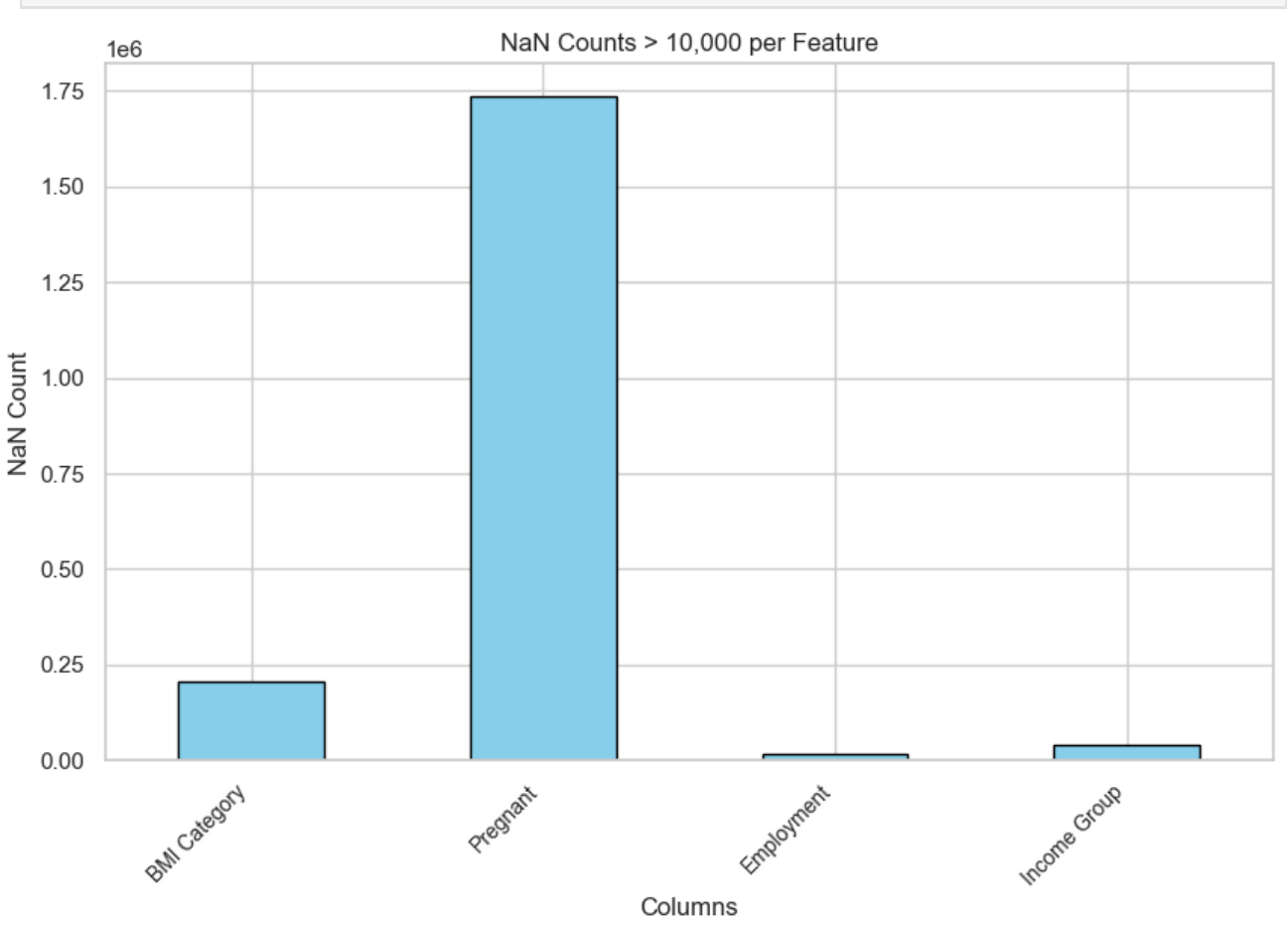


Contrary to intuition, we see less people resonded `Poor` health in 2020. Equally surprising, more people responded `Excellent` and `Very good` than the two previous years in the study. The highest coount of `excellent` General Health was reported in 2020. The response `Very good` increased each year from 2018 to 2022.

Missing values

Possible imbalance with missing data

In [14]:
```
nan_counts = df.isna().sum()

plt.figure(figsize=(10, 6))
nan_counts[nan_counts > 10_000].plot(kind='bar', color='skyblue', edgecolor=

# Customize the plot
plt.title('NaN Counts > 10,000 per Feature')
plt.xlabel('Columns')
plt.ylabel('NaN Count')
plt.xticks(rotation=45, ha='right', fontsize=10)

# Show the plot
plt.show()
```



Missing value summary

We first take a look at all selected columns, and calculate how many missing values each attribute has(from both count and percentage perspective):

In [15]:
```
df_na_count = df.isna().sum()
df_na_pct = round((df.isna().sum())/len(df)*100,3)
pd.concat([df_na_count, df_na_pct], keys = ['NA Count','NA Percent'], axis=1
```

Out[15]:

	NA Count	NA Percent
Date	6	0.000
State	0	0.000
Gender(Male)	0	0.000
Age Group	0	0.000
Race	0	0.000
BMI Category	205297	9.703
Physical Activity	0	0.000
Pregnant	1733740	81.941
Drinking	0	0.000
Smoking	0	0.000
Education Category	0	0.000
Employment	16338	0.772
Income Group	39423	1.863
Veteran	8747	0.413
Marital Status	108	0.005
Rent Home	94	0.004
Insurance	35	0.002
General Health	66	0.003
Region	0	0.000

Removing rows with missing values in certain features

For the following set of predictors: survey date, employment status, income group, veteran flag, marital status, rent/own home flag, health insurance, mental health, physical health: only a small percentage(<2%) of observations have missing values. Similarly, for response variable general health, only a tiny proportion of observations are blank. Therefore, we'll just remove records with missing values in these columns.

In [16]:
```
df.dropna(subset=['Date', 'Employment', 'Income Group', 'Veteran', 'Marital
df.shape
```

Out[16]:
```
(2076224, 19)
```

Imputing missing values for certain features

Two predictors with a lot of missing values are: **BMI category(10% missing)** and **pregnant flag(82% missing)**. We need to impute missing values for these two columns before predictive modelling.

BMI category is a categorical variable describing Body Mass Index. To fill in the blanks, we'll impute missing values as most common category(category 3). However, we cannot tell whether people really don't know their BMI, or they're reluctant to report an unhealthy BMI score. Therefore, we'll create a flag indicating missing values.

For pregnant flag, the question was not asked to everyone. Males and aged females will be imputed as non-pregnant, with missing value flag as False; other missing values will be imputed as non-pregnant as well, but the missing value flag will be True.

In [17]:
```
# Impute missing values for BMI category
df['BMI_missing'] = df['BMI Category'].isna().astype(int)
df['BMI Category'] = df['BMI Category'].fillna(3)
```

In [18]:
```
# Impute missing values for pregancy flag
df['Pregnant'] = np.where(df['Gender(Male)']==1, 2, df['Pregnant'])
df['Pregnant'] = np.where(df['Age Group'].isin([5,6]), 2, df['Pregnant'])
df['Pregnant_missing'] = df['Pregnant'].isna().astype(int)
df['Pregnant'] = df['Pregnant'].fillna(2)
```

In [19]:
```
df.describe()
```

Out[19]:

	Date	State	Gender(Male)	Age Group	Race
count	2076224	2.076224e+06	2.076224e+06	2.076224e+06	2.076224e+06
mean	2020-07-11 00:29:41.329183232	3.021882e+01	1.544075e+00	4.394180e+00	1.699239e+00
min	2018-01-02 00:00:00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
25%	2019-04-09 00:00:00	1.900000e+01	1.000000e+00	3.000000e+00	1.000000e+00
50%	2020-07-06 00:00:00	2.900000e+01	2.000000e+00	5.000000e+00	1.000000e+00
75%	2021-10-17 00:00:00	4.200000e+01	2.000000e+00	6.000000e+00	1.000000e+00
max	2022-12-31 00:00:00	7.800000e+01	9.000000e+00	6.000000e+00	6.000000e+00
std	NaN	1.587326e+01	5.210429e-01	1.604383e+00	1.444939e+00

Dealing with unknown/refused responses

Sometimes there are not missing values in the predictor, but there is a group indicating the respondent was unsure of the answer, or refused to answer a particular question. We'll do a count on refused/unknown responses and see if we need to drop them as well. Normally, 7 and 9 means Unknown or Refused, but there are a few exceptions which we'll also discuss.

In [20]:
```
count_7_per_col = df.apply(lambda col: round((col==7).sum()/len(df)*100,3))
count_9_per_col = df.apply(lambda col: round((col==9).sum()/len(df)*100,3))

pd.concat([count_7_per_col, count_9_per_col], keys = ['7 Percentage', '9 Per
```

Out[20]:

	7 Percentage	9 Percentage
Date	0.000	0.000
State	0.000	2.162
Gender(Male)	0.020	0.031
Age Group	0.000	0.000
Race	0.000	0.000
BMI Category	0.000	0.000
Physical Activity	0.000	0.882
Pregnant	0.046	0.074
Drinking	0.000	6.113
Smoking	0.000	3.619
Education Category	0.000	0.441
Employment	30.524	0.959
Income Group	13.490	4.582
Veteran	0.038	0.227
Marital Status	0.000	0.897
Rent Home	0.213	0.635
Insurance	0.169	1.681
General Health	0.175	0.076
Region	0.000	0.000
BMI_missing	0.000	0.000
Pregnant_missing	0.000	0.000

Here are the exceptions for predictors(if 7 or 9 has specific meanings):

State: 9 refers to a state
Employment: 7 means retired
Income: 7 and 9 are both valid groups

Also, drinking and smoking have a lot of unknown/refused responses(6% and 4% respectively). So we'll not remove them - we'll keep them in the dataset as the original category.

We'll remove the rest observations having attribute values in 7 or 9.

In [21]:
```
for col in (['Gender(Male)','Pregnant','Veteran','Rent Home','Insurance','Ge
    df = df[df[col] != 7]

for col in (['Gender(Male)','Pregnant','Physical Activity','Education Categc
    df = df[df[col] != 9]

df.shape
```

Out[21]: (1968123, 21)

For income category, 77 and 99 means Unknown and Refused respectively. Given the large number of observations in these two categories, we also imputed it using median group(group 8), and added income unknown/refused flag.

In [22]:
```
# Impute income category
df['Income_missing'] = np.where(df['Income Group'].isin([77,99]), 1, 0)
df['Income Group'] = np.where(df['Income Group'].isin([77,99]), 8, df['Incom
```

Comparing to the original dataset, we removed about 7% of the observations with missing value/unknown response/refused resonses.

Feature engineering

Encoding binary categorical varibles

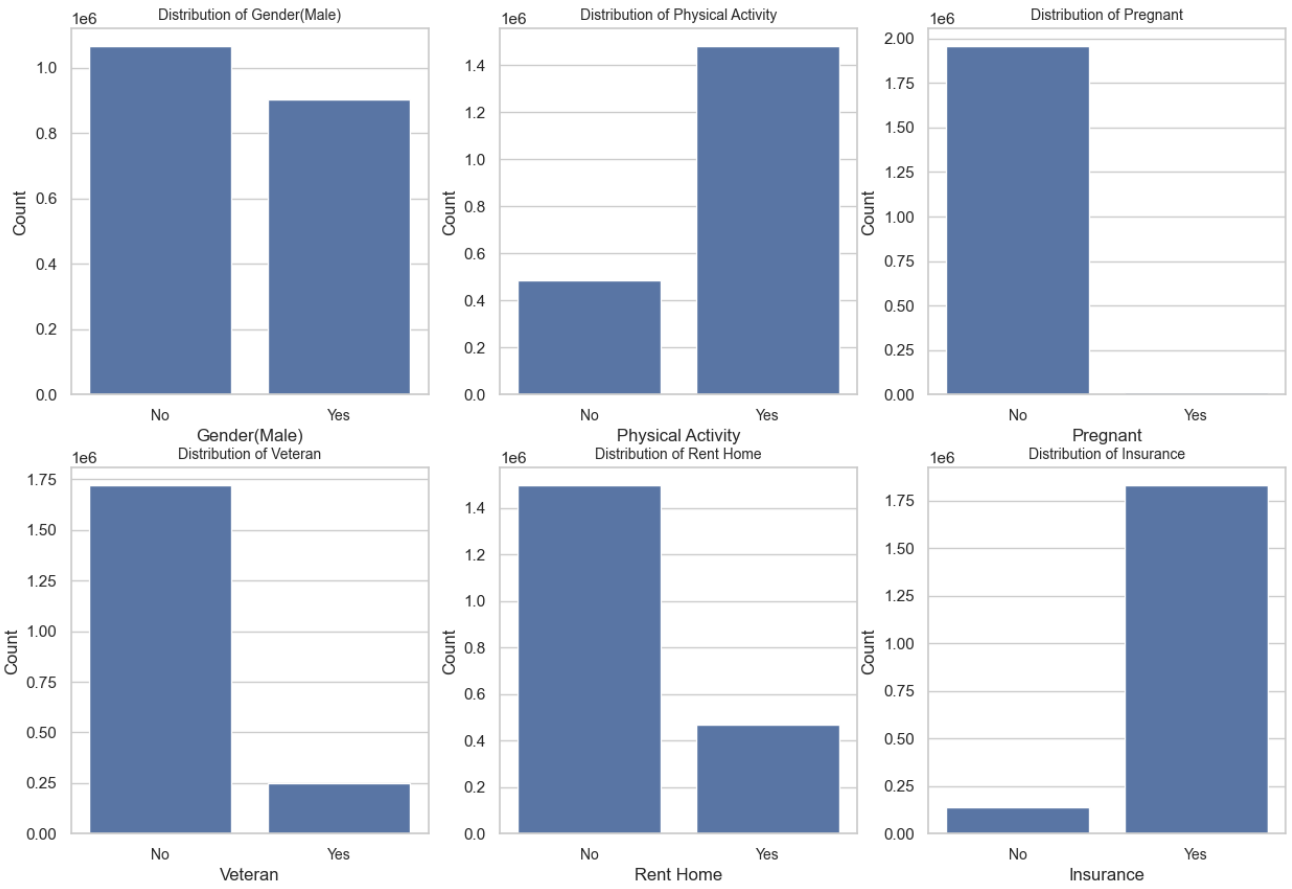
The below list of variables are binary. However, they're not on 0-1 scale. We need to recode them to 0-1 scale for modeling purpose.

Gender: 1 is male, 0 is female
Physical activity flag: 1 is yes, 0 is no
Pregnacy flag: 1 is yes, 0 is no
Veteran flag: 1 is yes, 0 is no
Renting home flag: 1 is yes, 0 is no
Insurance flag: 1 is yes, 0 is no

In [23]:
```
df['Gender(Male)'] = np.where(df['Gender(Male)']==1, 1, 0)
df['Physical Activity'] = np.where(df['Physical Activity']==1, 1, 0)
df['Pregnant'] = np.where(df['Pregnant']==1, 1, 0)
df['Veteran'] = np.where(df['Veteran']==1, 1, 0)
df['Rent Home'] = np.where(df['Rent Home']==2, 1, 0)
df['Insurance'] = np.where(df['Insurance']==1, 1, 0)
```

The below barplots show the distribution of the above six binary variables:

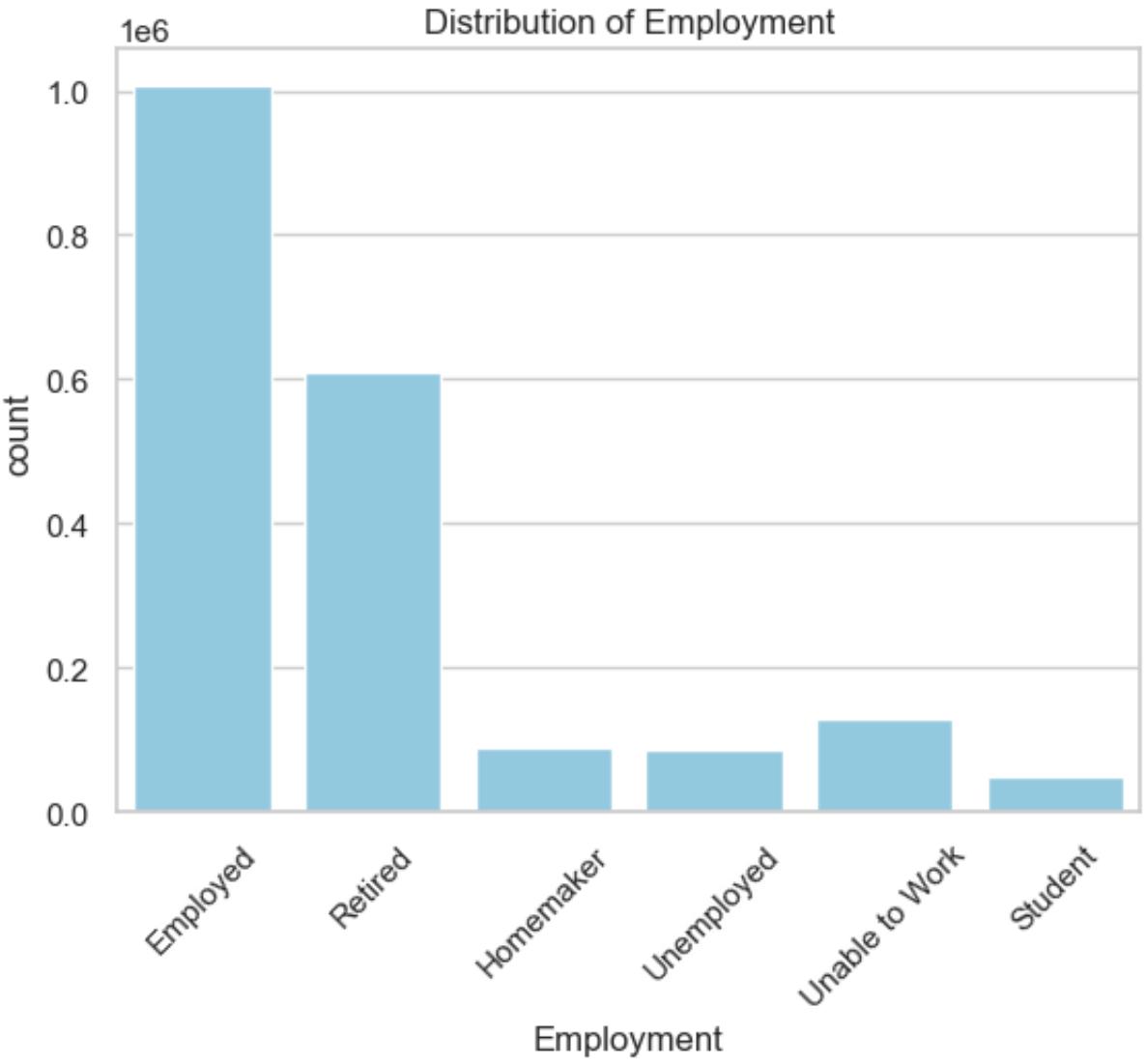
In [24]:
```
binary_cols = ["Gender(Male)", "Physical Activity", "Pregnant", "Veteran", "

fig, axs =  plt.subplots(2,3, figsize=(15, 10))
axs = axs.flatten()

# plot countplots for binary variables
for col, ax in zip(binary_cols, axs):
    sns.countplot(x=df[col], ax=ax)
    ax.set_ylabel('Count')
    ax.set_xlabel(col)
    ax.set_xticks([0,1], ['No', 'Yes'], fontsize=10)
    ax.set_title(f'Distribution of {col}', fontsize=10)


plt.tight_layout;
```



Creating Employment category

There are too many employment categories. We can group some of them together to get a cleaner version of data to put into the modeling process.

In [25]:
```
dict_employment = {1: 'Employed', 2: 'Employed', 3: 'Unemployed', 4: 'Unempl
           7: 'Retired', 8: 'Unable to Work'}

df['Employment'] = df['Employment'].map(dict_employment)
sns.countplot(data=df, x='Employment', color='skyblue')
plt.xticks(rotation=45)
plt.title('Distribution of Employment')
plt.show()
```



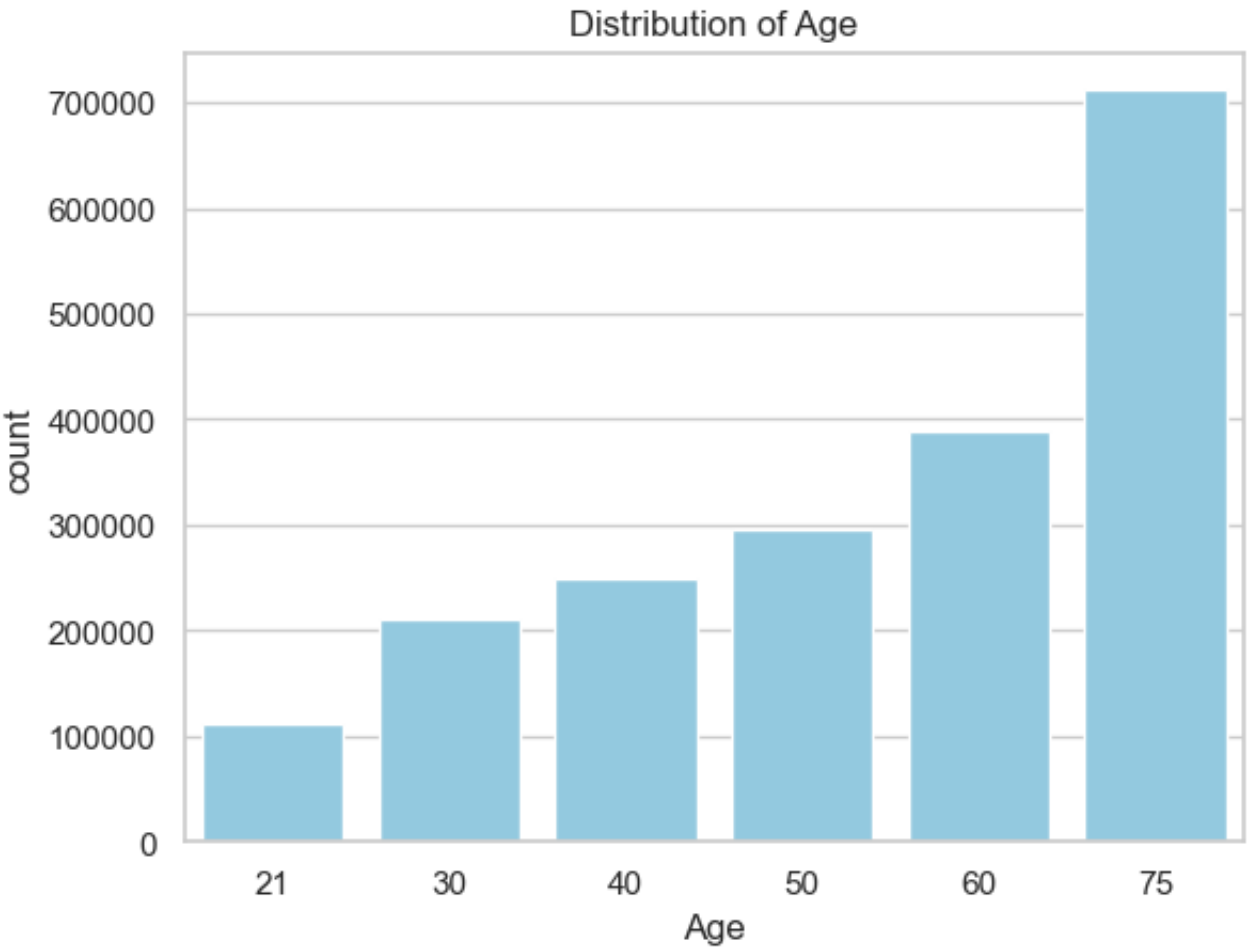
Turning categories into numerical values

Age group and income category are both categorical variables which group numerical values together. Numerical values can show a better scale than categories, which makes it easier to quantify relationships between predictors and response variable. Thus, we'll transform both these two variables back to numerical(using median values per group), and scale them properly:

Creating Age from Age Group

We'll calculate the age per group using median number. For the last group(age over 65), since age range is wider for this category, we use 75 as a proxy.

In [26]:
```
dict_age = {1: 21, 2: 30, 3: 40, 4: 50, 5: 60, 6: 75}

df['Age'] = df['Age Group'].map(dict_age)
sns.countplot(data=df, x='Age', color='skyblue')
plt.title('Distribution of Age')
plt.show()
```

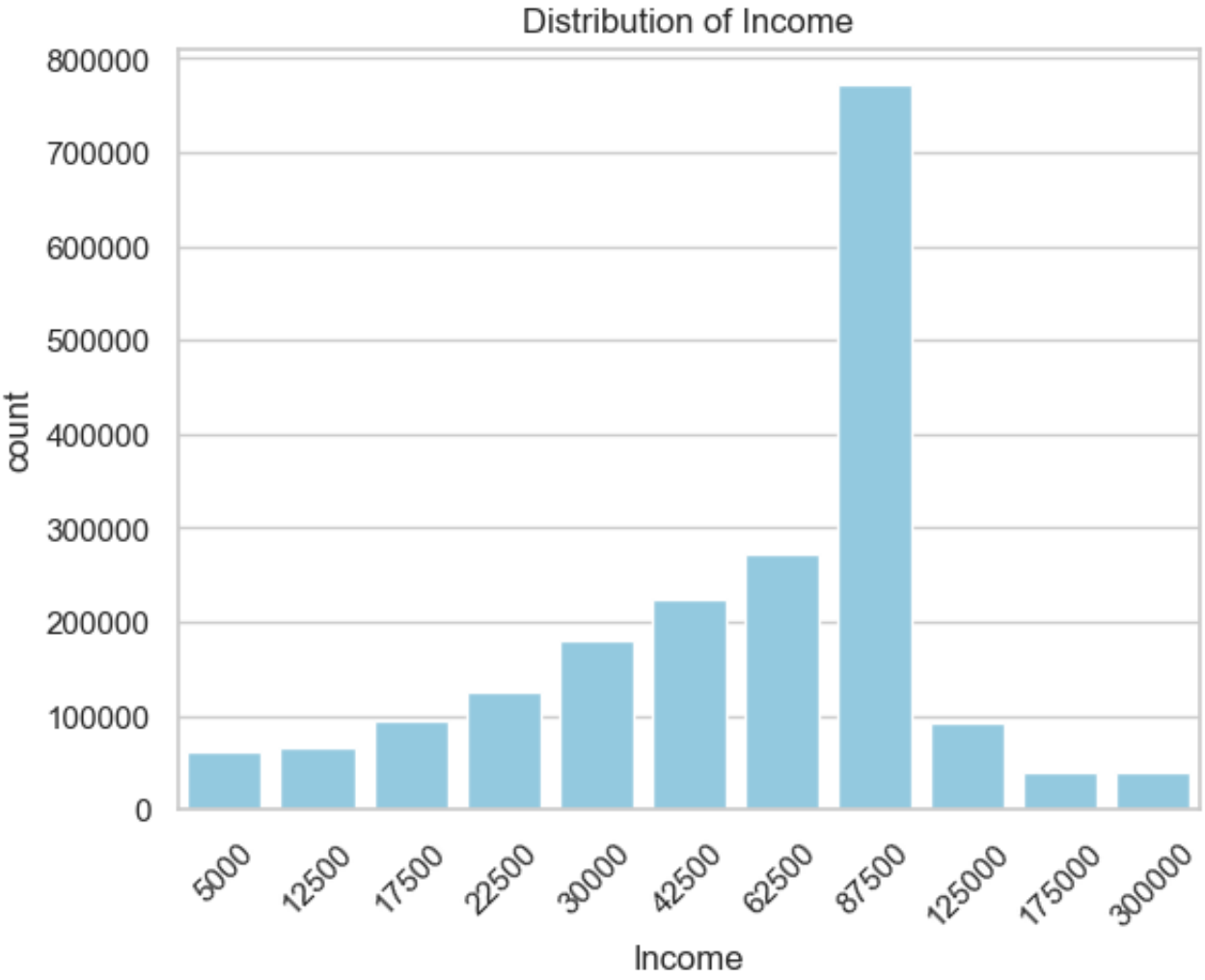


Creating Income from Income Category

We'll calculate the income per group using median number. For the last income level (income greater than \$200k), since income range is wider for this category, we use \$300k as a proxy.

In [27]:
```
dict_income = {1: 5000, 2: 12500, 3: 17500, 4: 22500, 5: 30000, 6: 42500,
               7: 62500, 8: 87500, 9: 125000, 10: 175000, 11: 300000}

df['Income'] = df['Income Group'].map(dict_income)
```

```
sns.countplot(data=df, x='Income', color='skyblue')
plt.xticks(rotation=45)
plt.title('Distribution of Income')
plt.show()
```



Converting Date to years

We will convert the date to its associated year to reduce complexity and allow a more aggregated and logical approach.

In [28]:
```
df['Date'] = df['Date'].dt.year
```

Converting float to int

Some of our categories are of float type while they are discrete integers. We can easily correct that without any data truncations with the astype(int) method

In [29]:
```
col_to_int=['State', 'Age Group', 'Race', 'BMI Category', 'Drinking', 'Smoki
for column in col_to_int:
    df[column] = df[column].astype(int)
```

Aggregating response variable to binary

Our response variable can have 5 categories:

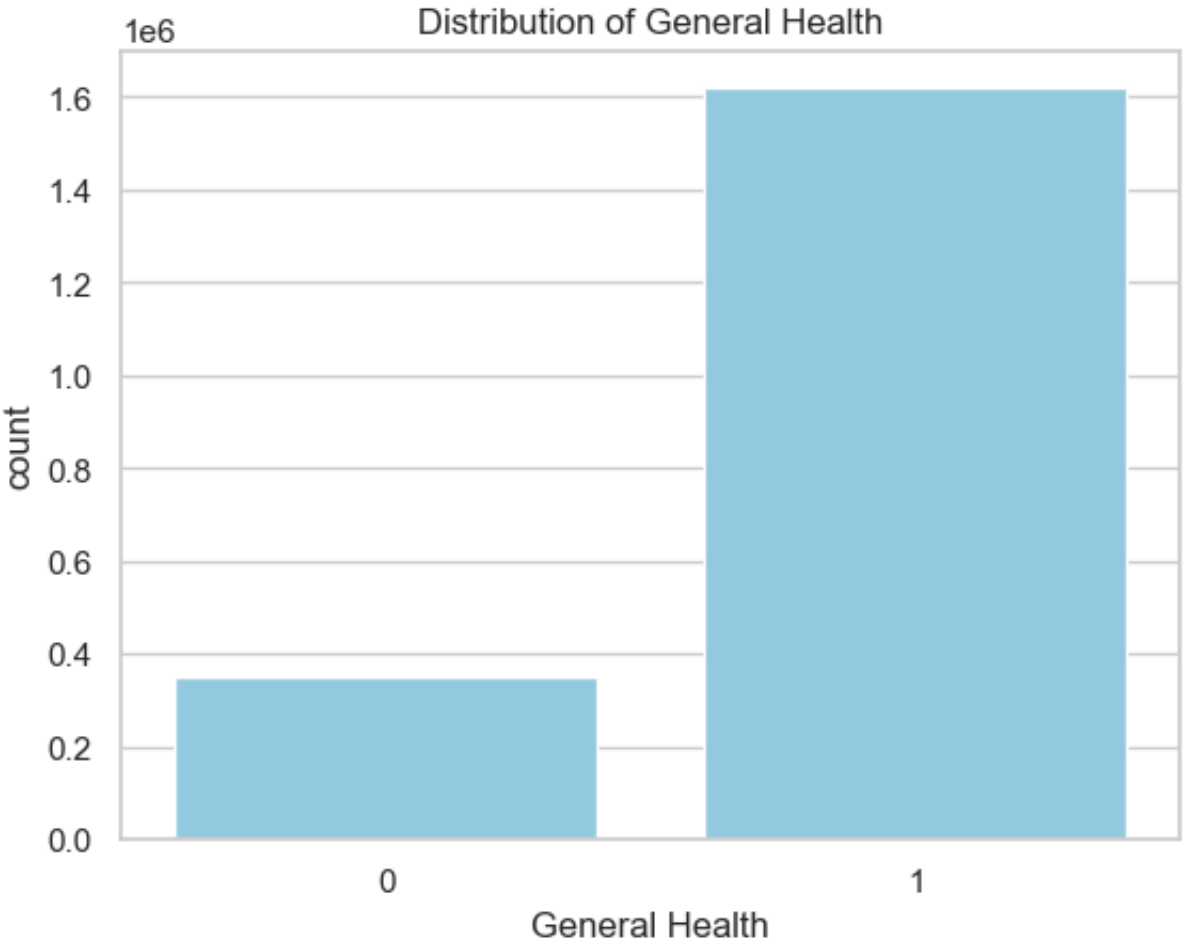
- 1 Excellent
- 2 Very good
- 3 Good
- 4 Fair
- 5 Poor

We will reclassify it as GOOD (=Good, Very Good and Excellent) for values below 3 and NOT GOOD for values above 3 (=Fair and Poor): 0 is NOT GOOD, 1 is GOOD.

In [30]:
```
threshold = 3  # below 3 means good health, above means not good

# Create a new binary variable based on the threshold
df['General Health'] = df['General Health'].apply(lambda x: 1 if x <= thresh

# Count the occurrences of 0 and 1 in the General Health column
count = df['General Health'].value_counts()

# Display the counts
print(count)
```

```
General Health
1    1619499
0     348624
Name: count, dtype: int64
```

In [31]:
```
sns.countplot(data=df, x='General Health', color='skyblue')
plt.title('Distribution of General Health')
plt.show()
```



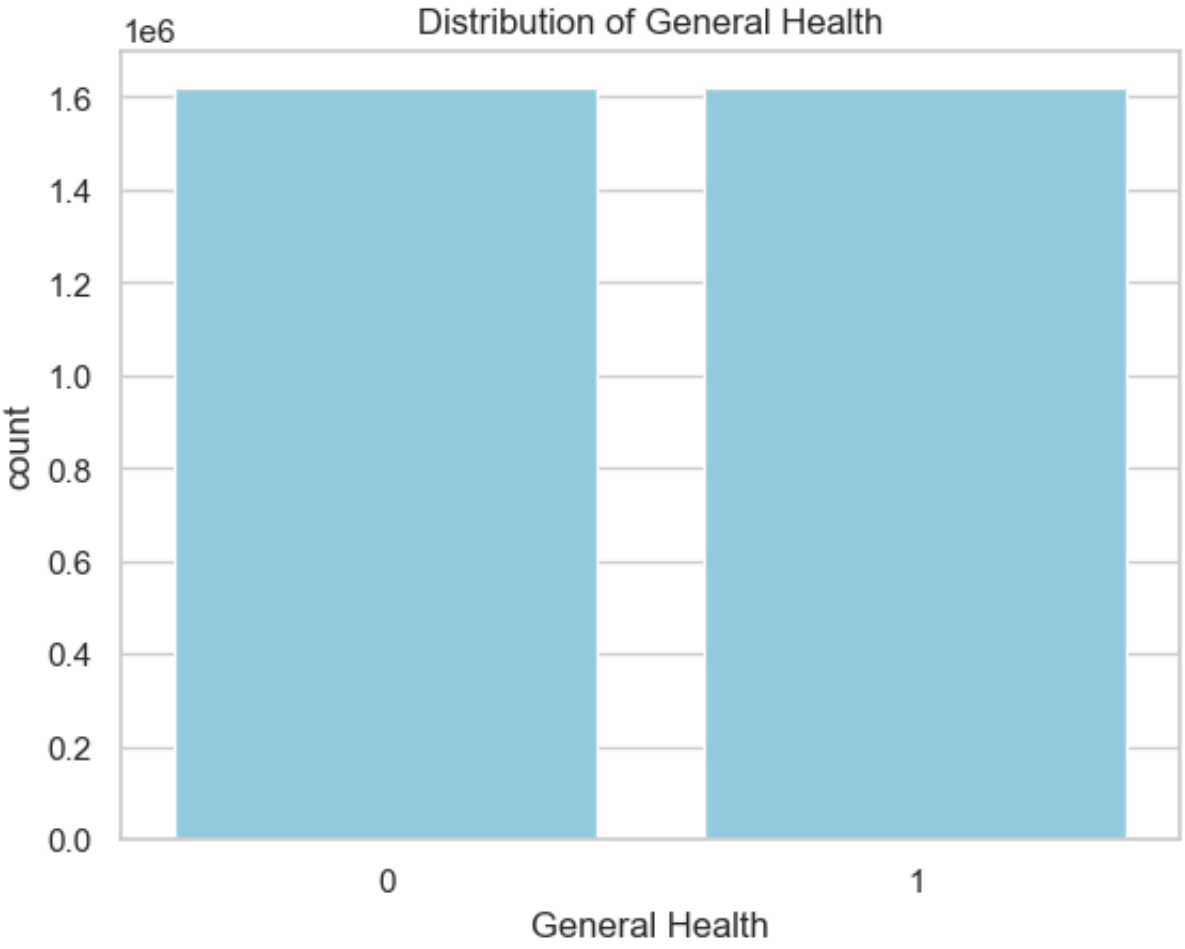
Imbalance treatment by resampling

The previous operations leads to an imbalance of data, meaning we have to treat it. We will oversample the 'NOT GOOD' health status.

In [32]:
```
# Separate the majority and minority classes
data_majority = df[df['General Health'] == 1]
data_minority = df[df['General Health'] == 0]

# Upsample minority class
data_minority_upsampled = resample(data_minority,
                                   replace=True,     # sample with replaceme
                                   n_samples=len(data_majority),    # to mat
                                   random_state=123) # reproducible results

# Combine majority class with upsampled minority class
data_upsampled = pd.concat([data_majority, data_minority_upsampled])

# New class counts
print(data_upsampled['General Health'].value_counts())

# replace the original df
df=data_upsampled.copy()
del data_upsampled # free memory
```

```
General Health
1    1619499
0    1619499
Name: count, dtype: int64
```

In [33]:
```
sns.countplot(data=df, x='General Health', color='skyblue')
plt.title('Distribution of General Health')
plt.show()
```



In [34]:
```
# drop the below features as we engineered new ones related to them
df = df.drop(['State', 'Age Group', 'Income Group'], axis=1)
```

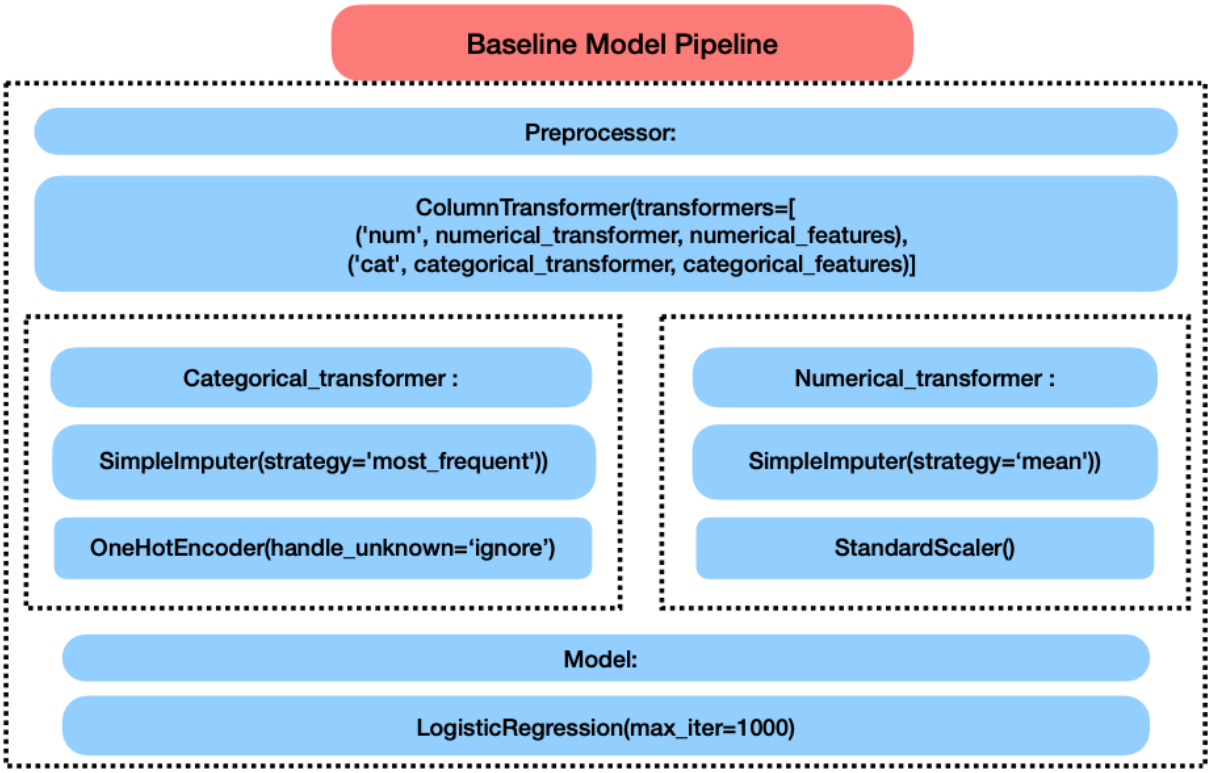
Baseline model

Baseline model selection

Given the binary nature of the response variable, a logistic regression model can be a good starting point for our baseline model. It's simple, interpretable, and effective for binary classification tasks.

Model training

We will first define the pre-processing of our features and define our baseline model to create our pipeline. Then we will run the pipeline to create our baseline model (called log_model). Then we will split the data into training and testing sets, train the logistic regression model on the training set, and evaluate it using appropriate metrics such as accuracy, precision, recall, and the ROC-AUC score.



In [35]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, r
from sklearn.impute import SimpleImputer

# Splitting the dataset into features (X) and target variable (y)
X = df.drop('General Health', axis=1)
y = df['General Health']

# Encoding categorical variables
categorical_features = ['Race', 'BMI Category', 'Education Category', 'Emplo
binary_features = ['Gender(Male)', 'Pregnant', 'Physical Activity', 'Insurar
numerical_features = ['Age', 'Income']

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

```
# Preprocessing for numerical data
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Define the model
model = LogisticRegression(max_iter=1000)

# Create and evaluate the pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('model', model)])

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

# Training the model
clf.fit(X_train, y_train)

# Predictions
y_train_pred = clf.predict(X_train)
y_test_pred = clf.predict(X_test)

# Evaluation metrics
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
F1 = f1_score(y_test, y_test_pred)
roc_auc = roc_auc_score(y_test, clf.predict_proba(X_test)[:, 1])

# Create a table of evaluation metrics
eval_metrics = pd.DataFrame({
    'baseline model logistic regression': [train_accuracy, test_accuracy, pr
}, index=['Training Accuracy', 'Test Accuracy', 'Precision', 'Recall', 'F1',

display(eval_metrics)
```

	baseline model logistic regression
Training Accuracy	0.703526
Test Accuracy	0.703873
Precision	0.692717
Recall	0.732493
F1	0.712050
ROC AUC	0.778334

Model evaluation

- The training accuracy of 70.36% is decent but not exceptionally high, indicating that there may be room for improvement. It suggests that the model is capturing some, but not all, of the patterns in the data. This could be due to the model's simplicity (logistic regression is a linear model) or the lack of separability in the data itself.
- The testing accuracy of 70.29% indicates that the model correctly predicts the general health status (Good or Not Good) in about 70% of the cases which is a good starting point.
- From training and testing accuracy analysis, we can deduct that our baseline model is not subject to overfitting (as the scores are nearly the same).
- Precision suggests that when the model predicts a person to be in good health, it is correct about 69.15% of the time.
- Recall indicates that the model identifies 73.18% of the actual cases of good health.
- F1 score is a metric commonly used in binary classification that fairly assesses the model's performance, by taking the mean of precision and recall and provides a balance between the two. Here the F1 score of 0.711 is relatively high and indicates a good balance between minimizing false positives (precision) and false negatives (recall).
- ROC-AUC Score is relatively high at 77.74%, suggesting the model has a good measure of separability between the two health classes.

As a conclusion this baseline model is a good starting point, but there is definitely room for improvements as we suspect that more complex models will allow us to improve our scoring metrics and therefore our predictive capabilities.

Baseline model interpretation

We will now analyze the coefficients of the logistic regression baseline model to understand the impact of different predictors on the likelihood of having GOOD health. We will do so by extracting the values of the top 10 coeficients to see how they impact the response variable.

(see code below)

- Employment ('Unable to Work'): A large negative coefficient suggests that being unable to work is significantly associated with poorer health. This is the most significant coefficient of the model.
- Education Category (Lowest level = not finished high school): A negative coefficient indicates that lower education levels are associated with poorer health.
- Employment ('Employed'): Being employed is positively associated with good health.
- Employment ('Student'): A positive coefficient implies that students are more likely to be in good health.
- Education Category (Highest level): Higher education levels show a positive relationship with good health.
- BMI Category (Obese): Obesity (BMI Category 4) has a negative impact on health.
- Smoking (Heavy smoker): Heavy smoking is associated with not good health.
- BMI Category (Underweight): Underweight individuals are more likely to be in good health, compared to the reference BMI category.
- BMI Category (Normal weight): Normal weight shows a negative association with good health compared to the reference category, which is interesting and may warrant further investigation.
- Income: Higher income is positively correlated with good health.

In conclusion our baseline model is telling us that employment and education category are the most important predictors to determine the health status of a person. This might seem a bit surprising at first, we would have expected BMI, smoking and alcohol to be in the first positions.

In [36]:
```
# Getting the names of the columns after one-hot encoding
ohe_columns = list(clf.named_steps['preprocessor'].named_transformers_['cat'
all_features = numerical_features + ohe_columns

# Extracting coefficients
coefficients = clf.named_steps['model'].coef_[0]

# Creating a DataFrame for easier interpretation
feature_importance = pd.DataFrame({'Feature': all_features, 'Coefficient': c

# Sorting the features by the absolute values of their coefficients
feature_importance['Absolute Coefficient'] = feature_importance['Coefficient
sorted_feature_importance = feature_importance.sort_values(by='Absolute Coef

sorted_feature_importance.head(10)  # Display the top 10 features in terms o
```

Out[36]:

	Feature	Coefficient	Absolute Coefficient
20	Employment_Unable to Work	-1.575648	1.575648
12	Education Category_1	-0.662039	0.662039
16	Employment_Employed	0.635999	0.635999
19	Employment_Student	0.592603	0.592603
15	Education Category_4	0.453552	0.453552
11	BMI Category_4	-0.433173	0.433173
42	Smoking_2	-0.381454	0.381454
9	BMI Category_2	0.328153	0.328153
8	BMI Category_1	-0.308038	0.308038
1	Income	0.277417	0.277417

Final model selection

As a next step, we want to improve upon our baseline model by selecting a model that will be better suited in the context of our classification of general health prediction based on our selected predictors:

- We saw that while our scores were not bad, we would like to explore additional approaches that may capture the full complexity of the dataset.
- Also, given the number of features, we also need a robust model that will be able to manage potential collinearities.
- The reporting specificities (ie telephonic survey) also requires a model that can handle noise properly without overfitting
- We also saw that some predictors were quite strong like educations and employment, potentially leading to less exploration of the other features

Therefore, the list of models considered are:

- decision trees
- random forest
- boosting

We compared the below pro's and con's to see which one was the most appropriate based on our context:

1. Decision Trees: Pros: They are simple, easy to interpret, and can capture non-linear patterns. Cons: Prone to overfitting, especially with a lot of features like we have in our current project They might not be as powerful as ensemble methods (like

Random Forest or Boosting). Also, Given their greedy approach, they might be less efficient when in presence of strong features (like in the case of employment and education) that will be systematically choosen in priority in the initial depths.

2. Random Forest: Pros: As ensemble of decision trees, typically more powerful than a single decision tree. It can handle a large number of features and is less likely to overfit than individual decision trees. Given its random nature to select features to build its trees, it may be particularly usefull here given the presence of some strong features while dozens are part of the set. Cons: More complex, less interpretable than a single decision tree, and the training time can be longer. Note that even if less interpretable, they still offer many ways to be interpreted like feature importance...
3. Boosting (w/ decision trees as weak learners) (e.g., Gradient Boosting, XGBoost, AdaBoost): Pros: Often provides higher predictive accuracy than Random Forest. It builds trees sequentially, each one correcting the errors of the previous, which can lead to better performance in a context like ours (many potential noise and inacurate answers from the telephonic nature of the survey) Cons: Can be more prone to overfitting compared to Random Forest, especially if the data is noisy. It also typically requires more parameter tuning.

Decision (Random Forest)

Given the potential inaccuracies or inconsistencies of our dataset, (remember those data were mostly collected via telephonic survey), the number of features, the presence of strong predictors, and the likely complexity of the required model, we decided that Random Forest may offer a more balanced approach because of its inherent nature of averaging over many decision trees that can make it more tolerant to such data issues, reducing the risk of overfitting to noise.

Note that Boosting methods (which can also be very relevant in this project), while powerful, might require more careful handling to ensure that they don't overfit by learning the noise in the data. This doesn't mean they can't be used effectively; it simply implies a need for more cautious parameter tuning and validation. We can however, in the next step, and for the sake of curiosity, make a comparison with it.

Next steps

In the final step of our project, we will so focus on answering our key questions based on a random forest model. To do so we will first have to tune its parameters to reach the best performances. Parameters we will consider are the below:

- Number of Trees (n_estimators): The number of trees in the forest. Increasing this number generally improves the model's performance but also increases the computational cost.
- Maximum Depth of Trees (max_depth): The maximum depth of each tree. Deeper

trees can model more complex patterns but can also lead to overfitting.
- Minimum Samples Split (min_samples_split): The minimum number of samples required to split an internal node. Higher values prevent creating nodes that might be too specific, thus avoiding overfitting.
- Minimum Samples Leaf (min_samples_leaf): The minimum number of samples required to be at a leaf node. This can smooth the model, especially in regression.
- Maximum Features (max_features): The number of features to consider when looking for the best split. Choices include "auto", "sqrt", "log2", or a fraction of the features.
- Bootstrap (bootstrap): Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

Final model

Model finetuning

As our final model will use random forest with hyperparameters obtained by finetuning, we will start by building an accurate sample (using stratification) of the original data (remember original set is +- 2m).

We will make a 10% sample.

We will focus our optimization on "depth of trees" and "number of estimators".

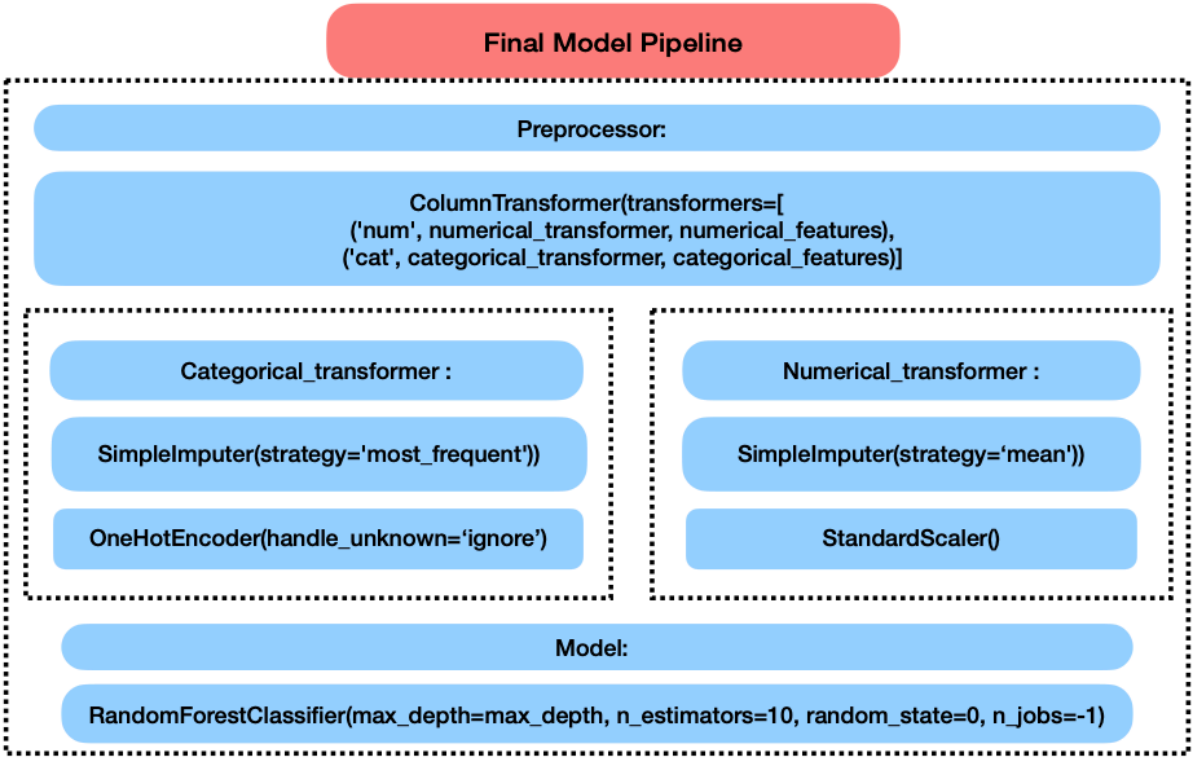
Due to computing resource limitations, we will first iterate on the tree_depth and keep the n_estimators fixed to 5, then we will use our best tree_depth found and make the n_estimators move along a bigger range. This approach is a compromise to allow us to fastly find a good approximation of the hyperparameters (even if more computing resource can surely lead to slightly better choice).

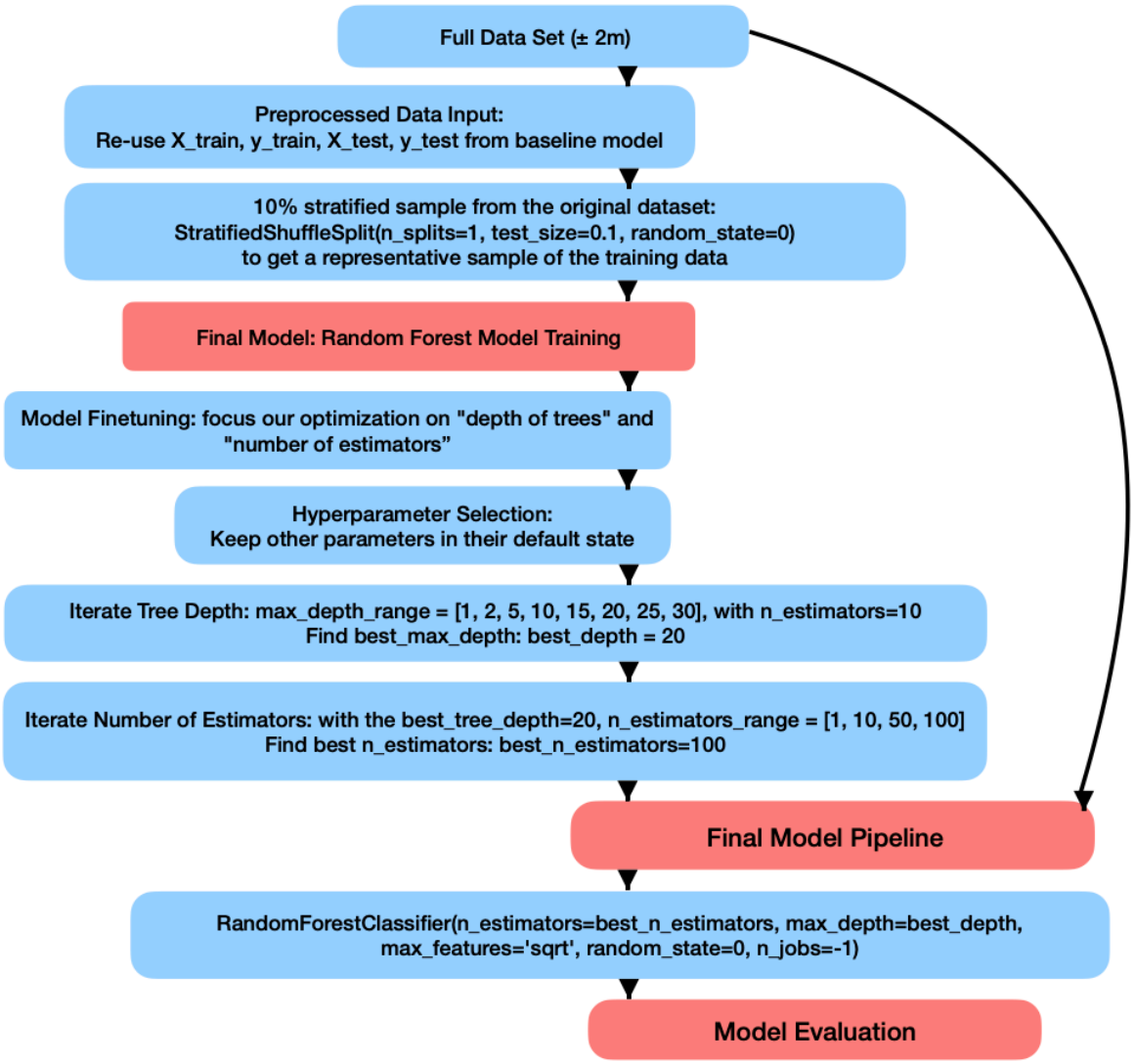
The other parameters will be left in their default state:

- min_samples_split: float | int = 2
- min_samples_leaf: float | int = 1
- criterion: Literal['gini', 'entropy', 'log_loss'] = "gini",
- max_features: float | int | Literal['sqrt', 'log2'] = "sqrt"
- bootstrap: bool = True,

Note we Re-use X_train, y_train, X_test, y_test from baseline model

Note we will use n_jobs=-1 to make sure we use all core available from the processor


Final Model Pipeline
Preprocessor:
ColumnTransformer(transformers=[
('num', numerical_transformer, numerical_features),
('cat', categorical_transformer, categorical_features)]
Categorical_transformer :
SimpleImputer(strategy='most_frequent'))
OneHotEncoder(handle_unknown='ignore')
Numerical_transformer :
SimpleImputer(strategy='mean'))
StandardScaler()
Model:
RandomForestClassifier(max_depth=max_depth, n_estimators=10, random_state=0, n_jobs=-1)



In [37]:
```
# Build a sub-sample X_train_sample and y_train_sample
from sklearn.model_selection import StratifiedShuffleSplit

# we Re-use X_train, y_train, X_test, y_test from baseline model

# Initialize StratifiedShuffleSplit
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.1, random_state=0)

# Applying StratifiedShuffleSplit to get a representative sample of the trai
for train_index, sample_index in sss.split(X_train, y_train):
    X_train_sample = X_train.iloc[sample_index]
    y_train_sample = y_train.iloc[sample_index]

print("X_train size=", X_train_sample.shape)
print("y_train size=", y_train_sample.shape)
```

```
X_train size= (259120, 20)
y_train size= (259120,)
```

In [38]:
```
from sklearn.ensemble import RandomForestClassifier
```

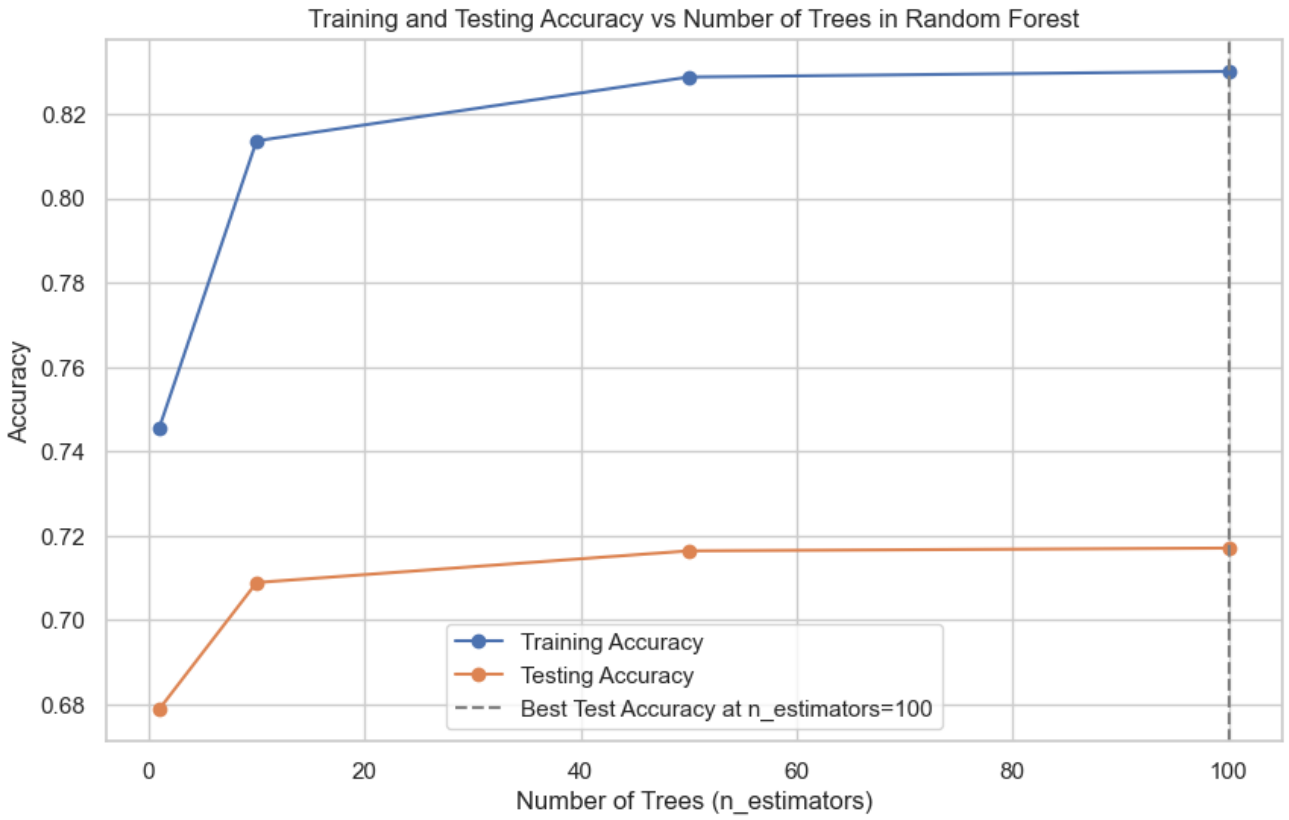
```python
# Assuming preprocessor, X_train_sample, X_test, y_train_sample, y_test are

max_depth_range = [1, 2, 5, 10, 15, 20, 25, 30]
train_accuracies_depth = []
test_accuracies_depth = []

for max_depth in max_depth_range:
    model = RandomForestClassifier(max_depth=max_depth, n_estimators=10, ran
    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', moc

    pipeline.fit(X_train_sample, y_train_sample)

    # Training accuracy
    y_train_pred = pipeline.predict(X_train_sample)
    train_accuracy = accuracy_score(y_train_sample, y_train_pred)
    train_accuracies_depth.append(train_accuracy)

    # Testing accuracy
    y_test_pred = pipeline.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_accuracies_depth.append(test_accuracy)

# Find the max depth with the highest test accuracy
max_test_accuracy = max(test_accuracies_depth)
best_max_depth = max_depth_range[test_accuracies_depth.index(max_test_accura

# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(max_depth_range, train_accuracies_depth, label='Training Accuracy',
plt.plot(max_depth_range, test_accuracies_depth, label='Testing Accuracy', m
plt.axvline(x=best_max_depth, color='gray', linestyle='--', label=f'Best Tes
plt.xlabel('Max Depth of Trees')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy vs Max Depth in Random Forest')
plt.legend()
plt.grid(True)
plt.show()
```

As from now, we will consider 20 as being our best depth to use. We will now fit different model based on number of estimators.

Note we are still using a sample of 10%

In [39]:
```
best_depth = 20
n_estimators_range = [1, 10, 50, 100]
train_accuracies = []
test_accuracies = []

for n_estimators in n_estimators_range:
    model = RandomForestClassifier(n_estimators=n_estimators, max_depth=best
    pipeline = Pipeline(steps=[('preprocessor', preprocessor), ('model', moc

    pipeline.fit(X_train_sample, y_train_sample)

    # Training accuracy
    y_train_pred = pipeline.predict(X_train_sample)
    train_accuracy = accuracy_score(y_train_sample, y_train_pred)
    train_accuracies.append(train_accuracy)

    # Testing accuracy
    y_test_pred = pipeline.predict(X_test)
    test_accuracy = accuracy_score(y_test, y_test_pred)
    test_accuracies.append(test_accuracy)

# Find the number of estimators with the highest test accuracy
max_test_accuracy = max(test_accuracies)
best_n_estimators = n_estimators_range[test_accuracies.index(max_test_accura
```

```
# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, train_accuracies, label='Training Accuracy', ma
plt.plot(n_estimators_range, test_accuracies, label='Testing Accuracy', mark
plt.axvline(x=best_n_estimators, color='gray', linestyle='--', label=f'Best
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy vs Number of Trees in Random Forest
plt.legend()
plt.grid(True)
plt.show()
```



Best hyperparameter selection

Based on the above analysis, we decided to select:

- max_depth=20
- n_estimators=100

We see from the graph that the test accuracy continue to improve after 100 estimators, however for the sake of computing resource management, we decided to stick to 100 as we see that the marginal improvement after this value is very low.

Final model pipeline

We wil now build our final pipeline which is based on the same data pre-processing as

the baseline model.

For this final model, we will train on the entire dataset. Remember that we had to finetune our model based on a sample of 10% of the original dataset. As this was only used to approximate the best hyperparameters, we assume this approach was relatively safe for the final training.

In [40]:
```
import pickle

# as the training of the model is intensive and time consuming, we first loc
model_file = 'final_random_forest_model.pkl'
z_file = 'final_random_forest_model.7z'

if not os.path.exists(model_file) and os.path.exists(z_file):
    with py7zr.SevenZipFile(z_file, mode='r') as z:
        z.extract()

if os.path.exists(model_file):
    # Load the trained model
    with open(model_file, 'rb') as file:
        clf_rf = pickle.load(file)

else:
    best_depth = 20
    best_n_estimators= 100

    # Encoding categorical variables
    categorical_features = ['Race', 'BMI Category', 'Education Category', 'E
    binary_features = ['Gender(Male)', 'Pregnant', 'Physical Activity', 'Ins
    numerical_features = ['Age', 'Income']

    # Preprocessing for categorical data
    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])

    # Preprocessing for numerical data
    numerical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='mean')),
        ('scaler', StandardScaler())
    ])

    # Bundle preprocessing for numerical and categorical data
    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numerical_transformer, numerical_features),
            ('cat', categorical_transformer, categorical_features)
        ])

    # Define the Random Forest model
    rf_model = RandomForestClassifier(n_estimators=best_n_estimators, max_de
```

```
    # Create the pipeline
    clf_rf = Pipeline(steps=[('preprocessor', preprocessor),
                             ('model', rf_model)])

    # Training the Random Forest model
    clf_rf.fit(X_train, y_train)

    # Save the trained model to a file
    with open(model_file, 'wb') as file:
        pickle.dump(clf_rf, file)

# Predictions
y_train_pred_rf = clf_rf.predict(X_train)
y_test_pred_rf = clf_rf.predict(X_test)

# Evaluation metrics
train_accuracy_rf = accuracy_score(y_train, y_train_pred_rf)
test_accuracy_rf = accuracy_score(y_test, y_test_pred_rf)
precision_rf = precision_score(y_test, y_test_pred_rf, average='weighted')
recall_rf = recall_score(y_test, y_test_pred_rf, average='weighted')
F1_rf = f1_score(y_test, y_test_pred)
roc_auc_rf = roc_auc_score(pd.get_dummies(y_test), clf_rf.predict_proba(X_te

eval_metrics['finetuned random forest model'] = [train_accuracy_rf, test_acc
eval_metrics
```

Out[40]:

	baseline model logistic regression	finetuned random forest model
Training Accuracy	0.703526	0.779235
Test Accuracy	0.703873	0.756394
Precision	0.692717	0.756459
Recall	0.732493	0.756394
F1	0.712050	0.719203
ROC AUC	0.778334	0.837799

From the above we see that our finetuned model clearly improved the performance of our baseline model.

With those metrics, we are now in a better position to make more accurate predictions.

Note that we will save the above model in a pickle file to avoid having to retrain each time.

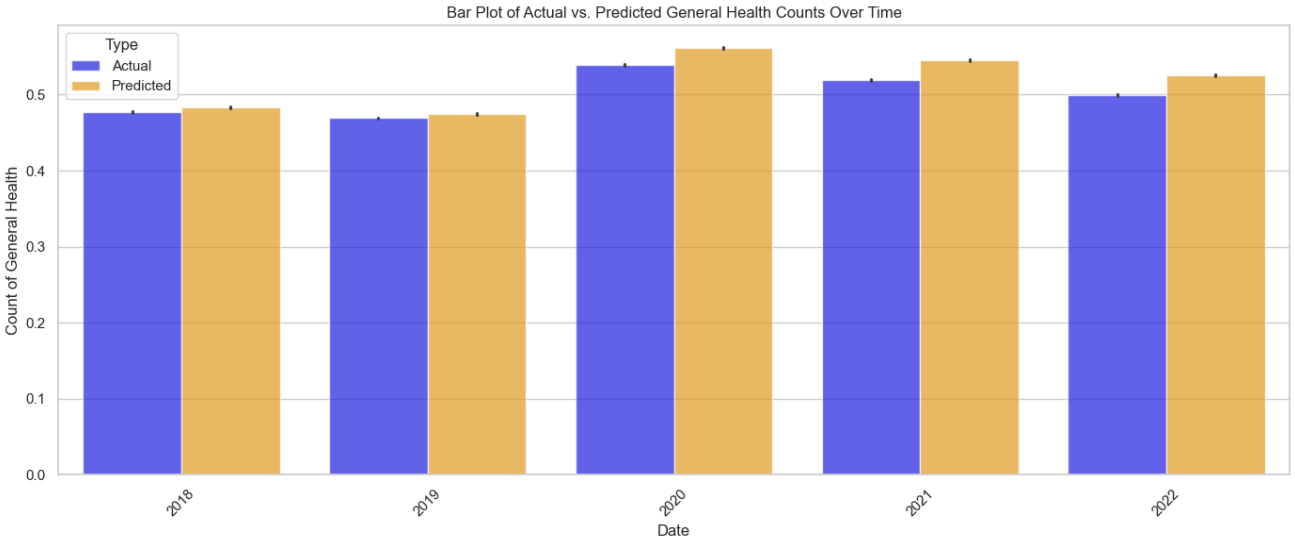
In [41]:
```
# Get feature names after one-hot encoding
feature_names_transformed = clf_rf.named_steps['preprocessor'].transformers_

# Combine with numerical feature names
```

```
feature_names = list(numerical_features) + list(feature_names_transformed)
importances = clf_rf.named_steps['model'].feature_importances_

# Now you can create the DataFrame
feature_importances = pd.DataFrame({'feature': feature_names, 'importance':
feature_importances = feature_importances.sort_values(by='importance', ascer

# display(feature_importances)
results = {key: value for key, value in zip(feature_names, importances)}
sorted_results = dict(sorted(results.items(), key=lambda item: item[1], reve
pretty_df('Feature Importance', sorted_results, ['feature', 'importance'], 4
```

Feature Importance

feature	importance	feature	im
Income	0.15919854639082867	Employment_Unable to Work	0.147007148
Employment_Employed	0.11176417153941724	Age	0.0814117508
Education Category_4	0.058720695310246054	BMI Category_4	0.042178730
Education Category_1	0.03434791533574887	Marital Status_1	0.022858831
Employment_Retired	0.02242873109831821	Smoking_2	0.021391546
BMI Category_2	0.0168262130022746	Education Category_2	0.0149505192
Race_1	0.014783428719172216	Smoking_1	0.013861251
BMI Category_3	0.012876952733001603	Education Category_3	0.010864806
Region_South	0.010771820804674325	Race_5	0.010668413
Date_2020	0.01065672684281385	Region_East	0.010384109
Region_Midwest	0.010367125038371632	Region_West	0.010028999
Date_2022	0.009978765420855193	Date_2021	0.009458803
Date_2018	0.009386569555255263	Drinking_1	0.009224542
Marital Status_3	0.009192793600709057	Date_2019	0.009065549
Marital Status_5	0.008703444787801991	Marital Status_2	0.008214348
Employment_Student	0.00794277503898227	Drinking_2	0.007685583
Employment_Unemployed	0.007659543211451124	Race_2	0.006870898
Drinking_9	0.006787903166441448	Employment_Homemaker	0.006580218
Race_6	0.005483644730131144	Marital Status_6	0.004821489
Region_U.S. Territory	0.004817639502567767	Marital Status_4	0.004463906
Race_4	0.004368113178419281	Race_3	0.003865004
BMI Category_1	0.0035872171979255763	Smoking_9	0.003492814

In [42]:
```
# Make predictions on the entire dataset
y_all_pred_rf = clf_rf.predict(X)  # X_all represents the entire dataset

# Create a DataFrame for visualization & sort
results_all = pd.DataFrame({'Actual': y, 'Predicted': y_all_pred_rf, 'Date':
results_all.sort_values(by='Date', inplace=True)

plt.figure(figsize=(14, 6))
sns.set(style="whitegrid")

# Reshape the DataFrame
results_all_melted = results_all.melt(id_vars=['Date'], value_vars=['Actual'

# Create a bar plot
sns.barplot(x='Date', y='General Health', hue='Type', data=results_all_melte

plt.title('Bar Plot of Actual vs. Predicted General Health Counts Over Time'
plt.xlabel('Date')
plt.ylabel('Count of General Health')
plt.legend(title='Type')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Conclusions

As we see from the actual vs predicted General Health Plots, the model's predictive capabilities are shown here.

- Random Forest model performed the best with scores besting the baseline model (logistic regression), performing relatively well for all 2018-2022 years analyzed, despite assumed disruptions during covid years.
- Predictions are slightly weaker still across 2020-2022 (bigger gap between actual health value and predicted).
    - Survey collection trends may have been impacted from covid.

    - It could also be because we combined the general health data from categorical to binary.

Many interesting insights were gathered from this analysis that surprised us.

- When analyzing important features, Income and Employment highest were the highest influencers on General Health.
- Drinking and smoking were not used in top five predictors as we predicted.
- Interesting to see general health increased during covid years, as seen in EDA

Appendix

Variable definitions

Variable name
SAS variable name in CDC data
Survey question:

Interview Date
IDATE
Survey: Date of the phone interview
Converted to datetime yyyy-mm-dd

State
_STATE
Survey: Respondent's US State

Value	State	Value	State	Value	State	Value	State
1	Alabama	2	Alaska	4	Arizona	5	Arkansas
6	California	8	Colorado	9	Connecticut	10	Delaware
11	District of Columbia	12	Florida	13	Georgia	15	Hawaii
16	Idaho	17	Illinois	18	Indiana	19	Iowa
20	Kansas	21	Kentucky	22	Louisiana	23	Maine
24	Maryland	25	Massachusetts	26	Michigan	27	Minnesota
28	Mississippi	29	Missouri	30	Montana	31	Nebraska
			New				New

32	Nevada	33	Hampshire	34	New Jersey	35	Mexico
36	New York	37	North Carolina	38	North Dakota	39	Ohio
40	Oklahoma	41	Oregon	42	Pennsylvania	44	Rhode Island
45	South Carolina	46	South Dakota	47	Tennessee	48	Texas
49	Utah	50	Vermont	51	Virginia	53	Washington
54	West Virginia	55	Wisconsin	56	Wyoming	66	Guam
72	Puerto Rico	78	Virgin Islands				

Gender

GENDER

Survey: Sex at birth

Value	Value Label
1	Male
2	Female

Age

_AGE_G

Survey: Six-level imputed age category

Value	Value Label
1	Age 18 to 24
2	Age 25 to 34
3	Age 35 to 44
4	Age 45 to 54
5	Age 55 to 64
6	Age 65 or older

Race

IMPRACE

Imputed race/ethnicity value (This value is the reported race/ethnicity or an imputed race/ethnicity, if the respondent refused to give a race/ethnicity. The value of the imputed race/ethnicity will be the most common race/ethnicity

Value	Value Label
1	White, Non-Hispanic
2	Black, Non-Hispanic
3	Asian, Non-Hispanic
4	American Indian/Alaskan Native, Non-Hispanic
5	Hispanic
6	Other race, Non-Hispanic

BMI

_BMI5CAT

Four-categories of Body Mass Index (BMI)

Value	Value Label
1	Underweight: _BMI5 < 1850 (_BMI5 has 2implied decimal places)
2	Normal Weight: 1850 <= _BMI5 < 2500
3	Overweight: 2500 <= _BMI5 < 3000
4	Obese: 3000 <= _BMI5 < 9999
BLANK	Don't know/Refused/Missing

Physical Activity

_TOTINDA

Adults who reported doing physical activity or exercise during the past 30 days other than their regular job

Value	Value Label
1	Had physical activity or exercise
2	No physical activity or exercise inlast 30 days
9	Don't know/Refused/Missing

Pregnant

PREGNANT

To your knowledge, are you now pregnant?

Value	Value Label
1	Yes

2	No
7	Don't know/Not Sure
9	Refused
BLANK	Not asked or Missing

Drinking

_RFDRHV8

Heavy drinkers (adult men having more than 14 drinks per week and adult women having more than 7 drinks per week)?

Value	Value Label
1	No
2	Yes
9	Don’t know/Refused/Missing

Smoking

_RFSMOK3

Smoking GroupAdults who are current smokers

Value	Value Label
1	No
2	Yes
9	Don't know/Refused/Missing

Education Category

_EDUCAG

Level of education completed

Value	Value Label
1	Did not graduate High School
2	Graduated High School
3	Attended College or Technical School
4	Graduated from College or Technical School
9	Don't know/Not sure/Missing

Employment

EMPLOY1

Current employment situation

Value	Value Label
1	Employed for wages
2	Self-employed
3	Out of work for 1 year or more
4	Out of work for less than 1 year
5	A homemaker
6	A student
7	Retired
8	Unable to work
9	Refused

Income Category

INCOME3

Annual income from all sources (income in $USD)

Value	Value Label
1	Less than 10,000
2	Less than 15,000 (10,000 to < 15,000)
3	Less than 20,000 (15,000 to < 20,000)
4	Less than 25,000 (20,000 to < 25,000)
5	Less than 35,000 (25,000 to < 35,000)
6	Less than 50,000 (35,000 to < 50,000)
7	Less than 75,000 (50,000 to < 75,000)
8	Less than 100,000? (75,000 to < 100,000)
9	Less than 150,000? (100,000 to < 150,000)
10	Less than 200,000? (150,000 to < 200,000)
11	$200,000 or more

Veteran

VETERAN
Have you ever served on active duty in the United States Armed Forces, either in the regular military or in a National Guard or military reserve unit?

Value	Value Label
1	Yes
2	No
7	Don't know/Not Sure
9	Refused

Marital Status
MARITAL
Respondent's marital status

Value	Marital Status
1	Married
2	Divorced
3	Widowed
4	Separated
5	Never married
6	A member of an unmarried couple
9	Refused

Adult in Household
NUMADULT
Excluding adults living away from home, such as students away at college, how many members of your household, including yourself, are 18 years or older

Value	Label
1	Number of adults in the household
2	Number of adults in the household
3	Number of adults in the household
4	Number of adults in the household
5	Number of adults in the household
6 - 99	Number of adults in the household

BLANK	Missing

Children in Household

CHILDREN

How many children less than 18 years of age live in your household?

Value	Label
0	
1 - 87	Number of children
88	None
99	Refused
BLANK	Not asked or Missing

Rent Home

RENTHOM1

Do you own or rent your home?

Value	Label
1	Own
2	Rent
3	Other arrangement
7	Don't know/Not Sure
9	Refused

Mental Health

MENTHLTH

Now thinking about your mental health, which includes stress, depression, and problems with emoti ons, for how many days during the past 30 days was your mental health not good?

Value	Label
1 - 30	Number of days
88	None
77	Don't know/Not sure
99	Refused

BLANK	Not asked or Missing

Physical Health

PHYSHLTH

Now thinking about your physical health, which includes physical illness and injury, for how many days during the past 30 days was your physical health not good?

Value	Label
1 - 30	Number of days
88	None
77	Don't know/Not sure
99	Refused
BLANK	Not asked or Missing

General Health

GENHLTH

Would you say that in general your health is:

Value	Label
1	Excellent
2	Very good
3	Good
4	Fair
5	Poor
7	Don’t know/Not Sure
9	Refused
BLANK	Not asked or Missing

Create data file for analysis

Import libraries

In [43]:
```
import glob
import re
import csv
```

Combine downloaded XPT (SAS) files

In [44]:
```
%%time
column_mapping = {
    'SEX1': 'GENDER',
    'SEXVAR': 'GENDER',
    'INCOME2': 'INCOME3',
    'HLTHPLN1': '_HLTHPLN',
    '_RFDRHV6': '_RFDRHV8',
    '_RFDRHV7': '_RFDRHV8'
}
features, dfs = [], []
for yr in range(2018, 2023):
    print(f'Reading year {yr}...')
    file_path = f'../raw_data/XPT/LLCP{yr}.XPT_'
    df = pd.read_sas(file_path, format='xport')
    df.rename(columns=column_mapping, inplace=True)
    features.append(df.columns)
    dfs.append(df)

features
```

```
Reading year 2018...
Reading year 2019...
Reading year 2020...
Reading year 2021...
Reading year 2022...
CPU times: total: 1min 34s
Wall time: 1min 47s
```

Out[44]:
```
[Index(['_STATE', 'FMONTH', 'IDATE', 'IMONTH', 'IDAY', 'IYEAR', 'DISPCODE',
       'SEQNO', '_PSU', 'CTELENM1',
       ...
       '_MAM5022', '_RFPAP34', '_RFPSA22', '_RFBLDS3', '_COL10YR', '_HFOB3
YR',
       '_FS5YR', '_FOBTFS', '_CRCREC', '_AIDTST3'],
      dtype='object', length=275),
 Index(['_STATE', 'FMONTH', 'IDATE', 'IMONTH', 'IDAY', 'IYEAR', 'DISPCODE',
       'SEQNO', '_PSU', 'CTELENM1',
       ...
       '_VEGESU1', '_FRTLT1A', '_VEGLT1A', '_FRT16A', '_VEG23A', '_FRUITE1
',
       '_VEGETE1', '_FLSHOT7', '_PNEUMO3', '_AIDTST4'],
      dtype='object', length=342),
 Index(['_STATE', 'FMONTH', 'IDATE', 'IMONTH', 'IDAY', 'IYEAR', 'DISPCODE',
       'SEQNO', '_PSU', 'CTELENM1',
       ...
       '_RFPSA23', '_CLNSCPY', '_SGMSCPY', '_SGMS10Y', '_RFBLDS4', '_STOLD
NA',
       '_VIRCOLN', '_SBONTIM', '_CRCREC1', '_AIDTST4'],
      dtype='object', length=279),
 Index(['_STATE', 'FMONTH', 'IDATE', 'IMONTH', 'IDAY', 'IYEAR', 'DISPCODE',
       'SEQNO', '_PSU', 'CTELENM1',
       ...
       '_FRTRES1', '_VEGRES1', '_FRUTSU1', '_VEGESU1', '_FRTLT1A', '_VEGLT
1A',
       '_FRT16A', '_VEG23A', '_FRUITE1', '_VEGETE1'],
      dtype='object', length=303),
 Index(['_STATE', 'FMONTH', 'IDATE', 'IMONTH', 'IDAY', 'IYEAR', 'DISPCODE',
       'SEQNO', '_PSU', 'CTELENM1',
       ...
       '_SMOKGRP', '_LCSREC', 'DRNKANY6', 'DROCDY4_', '_RFBING6', '_DRNKWK
2',
       '_RFDRHV8', '_FLSHOT7', '_PNEUMO3', '_AIDTST4'],
      dtype='object', length=328)]
```

Over the years, the CDC changed the name of the features, but the data and description remain the same. The column mapping ensures the changing feature names over the year will have the same name for the final CSV files used for analysis. Over 14 years of data were downloaded, only five years (2018-2022) will be used in the study.

Find common features among the data files

In [45]:
```
def recursive_set_intersection(sets, index=0):
    if index == len(sets) - 1:
        return sets[index]
    else:
        common_elements = sets[index].intersection(recursive_set_intersectio
        return common_elements

sets = []
```

```
for i in range(0, len(features)):
    sets.append(set(features[i]))

common_features = list(recursive_set_intersection(sets))
print(common_features)
```

```
['_STATE', 'CRGVHRS1', 'FMONTH', 'CSRVINSR', 'CASTHNO2', '_RAWRAKE', '_ASTHM
S1', 'QSTVER', 'COLGHOUS', '_TOTINDA', '_AGE_G', 'CDSOCIAL', 'GENDER', 'DECI
DE', 'SOFEMALE', 'CSRVDEIN', 'PVTRESD3', '_CHLDCNT', 'WEIGHT2', 'HTIN4', '_S
TRWT', 'NUMMEN', '_EDUCAG', 'LCSNUMCG', 'HOWLONG', 'STOPSMK2', 'RENTHOM1', '
_SMOKER3', 'NUMWOMEN', 'PSATEST1', 'CRGVEXPT', 'GENHLTH', 'EYEEXAM1', '_DUAL
USE', 'CSRVSUM', 'SHINGLE2', '_BMI5', 'CHECKUP1', 'HTM4', 'DRNK3GE5', 'TETAN
US1', 'CHILDREN', 'CRGVLNG1', 'MSCODE', '_CASTHM1', '_HLTHPLN', '_RFSMOK3',
'HADMAM', 'LASTSMK2', '_URBSTAT', '_AGE80', 'POORHLTH', 'PNEUVAC4', 'SEQNO',
'MARIJAN1', '_RFHLTH', 'QSTLANG', '_PNEUMO3', 'RCSRLTN2', 'LCSLAST', 'IDATE'
, 'DIFFDRES', 'VETERAN3', 'HEIGHT3', 'STATERE1', 'HPVADSHT', '_WT2RAKE', 'MA
RITAL', 'LCSFIRST', '_METSTAT', 'SMOKE100', '_RFDRHV8', 'EDUCA', 'CNCRDIFF',
'CDASSIST', 'CSRVCLIN', '_MICHD', 'MENTHLTH', '_DUALCOR', 'IYEAR', 'CVDINFR4
', 'DISPCODE', 'CSRVPAIN', 'SMOKDAY2', 'USENOW3', 'CASTHDX2', 'CVDCRHD4', 'C
STATE1', 'PVTRESD1', 'HADHYST2', 'IMONTH', 'CHKHEMO3', '_HISPANC', 'PREGNANT
', 'HIVTSTD3', 'WTKG3', 'CTELNUM1', 'CVDSTRK3', 'CDHOUSE', 'NUMADULT', '_RFB
MI5', '_PHYS14D', '_STSTR', '_MENT14D', 'MAXDRNKS', 'CAREGIV1', 'DEAF', 'EMP
LOY1', 'CTELENM1', 'SOMALE', '_LTASTH1', '_BMI5CAT', 'CNCRAGE', '_LLCPWT2',
'INCOME3', '_CLLCPWT', 'CIMEMLOS', 'CDHELP', 'SAFETIME', 'BLIND', '_PSU', 'C
SRVDOC1', '_IMPRACE', 'PHYSHLTH', 'CCLGHOUS', 'HHADULT', 'ASTHMA3', 'LANDLIN
E', 'TRNSGNDR', 'DIFFWALK', 'CELLFON5', '_AGEG5YR', '_CHISPNC', 'DIFFALON',
'_AGE65YR', '_LLCPWT', 'CDDISCUS', 'IDAY', 'CSRVINST', 'ASTHNOW', 'CSRVRTRN'
, 'EXERANY2']
```

Each year has hundreds of features to consider. Not all of the featurtes appear in each year. This function makes sets of each datasets features and finds common features across all of the years to be used in the study. Any feature not common across all five years will not be included in the final CSV file.

Combine all data frames

In [46]:
```
df_combined = pd.DataFrame()
for df in dfs:
    df = df[common_features]
    print(df.shape)
    df_combined = pd.concat([df_combined, df], axis=0)
df_combined.shape
```

```
(437436, 142)
(418268, 142)
(401958, 142)
(438693, 142)
(445132, 142)
```

Out[46]: (2141487, 142)

Create CSV file of features

In [47]:
```
column_names = df_combined.columns.tolist()
csv_file_path = '../raw_data/features.csv'
with open(csv_file_path, 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(column_names)
```

Save the CSV file so it can be read prior to analysis. Downloading and combining the data files is a time consumiong process that just needs to be done once.

Data statistics

In [48]:
```
%%time
df_combined.describe()
```

```
CPU times: total: 10.6 s
Wall time: 12 s
```

Out[48]:

	_STATE	CRGVHRS1	FMONTH	CSRVINSR	CASTHNO2	_F
count	2.141487e+06	74812.000000	2.141487e+06	32147.000000	28281.000000	2.14
mean	3.023119e+01	2.408344	6.462997e+00	1.128068	1.480110	1.20
std	1.592214e+01	1.850787	3.450432e+00	0.764005	1.012913	5.31
min	1.000000e+00	1.000000	1.000000e+00	1.000000	1.000000	3.33
25%	1.800000e+01	1.000000	3.000000e+00	1.000000	1.000000	1.00
50%	2.900000e+01	2.000000	7.000000e+00	1.000000	1.000000	1.00
75%	4.200000e+01	4.000000	9.000000e+00	1.000000	2.000000	1.00
max	7.800000e+01	9.000000	1.200000e+01	9.000000	9.000000	5.00

8 rows × 137 columns

In [49]:
```
df_combined.head()
```

Out[49]:

	_STATE	CRGVHRS1	FMONTH	CSRVINSR	CASTHNO2	_RAWRAKE	_ASTHMS1	Q
0	1.0	NaN	1.0	NaN	NaN	1.0	3.0	
1	1.0	NaN	1.0	NaN	NaN	1.0	3.0	
2	1.0	NaN	1.0	NaN	NaN	1.0	3.0	
3	1.0	NaN	1.0	NaN	NaN	1.0	3.0	
4	1.0	NaN	1.0	NaN	NaN	2.0	3.0	

5 rows × 142 columns

In [50]:
```
df_combined['IDATE'] = df_combined['IDATE'].astype(str)
df_combined['IDATE'] = pd.to_datetime(df_combined['IDATE'], format='%m%d%Y',
display(df_combined['IDATE'].head(10))
```

```
0   2018-01-05
1   2018-01-12
2   2018-01-08
3   2018-01-03
4   2018-01-12
5   2018-01-11
6   2018-01-10
7   2018-01-13
8   2018-01-09
9   2018-01-10
Name: IDATE, dtype: datetime64[ns]
```

The date field is used for EDA and analysis. It needs to be defined as a date object in PANDAS dataframe. There are some date errata scattered across all of the observations. The `coerce` argument will identify those missing values so the observations can be dropped - if decided - when the CSV file is loaded into a new data frame.

Save final data frame as CSV file

In [51]:
```
df_combined.to_csv('../raw_data/full_dataset.csv', index=False)
```