Title: Visualising NHANES dataset

Name: Lakshya Cheema **Student Id:** 224717273 **Email Id:** s224717273@deakin.edu.au **Undergraduate SIT220 student**

Introduction:

In this task, I explored data from the National Health and Nutrition Examination Survey (NHANES), a large-scale survey conducted to assess the health and nutritional status of adults and children in the United States. The datasets I worked with included various modules such as demographics, body measurements, alcohol use, dietary intake, and blood test results. These files were provided in .xpt format, which I imported using the pyreadstat library. After importing the datasets, I merged them on the unique participant identifier SEQN to create a unified DataFrame for analysis. To prepare the data, I cleaned the merged DataFrame by removing duplicate records, dropping columns with missing values, and eliminating unnecessary variables based on a careful review of the NHANES documentation. I also renamed and recoded multiple columns to make them more human-readable and suitable for analysis. This included converting coded values (such as gender, race, and participation status) into descriptive labels, which helped make the analysis more intuitive and easier to interpret. Following data preparation, I performed exploratory analysis using value_counts() and crosstab functions to understand the distribution and relationships among key variables like age group, gender, language of interview, country of birth, and proxy/interpreter usage. These insights guided the development of five visualisations using the Bokeh library, which allowed me to create interactive, aesthetically appealing plots such as heatmaps, scatter plots, and more. Throughout the task, I also reflected on the considerations and limitations of working with survey data, particularly when analyzing sensitive variables such as race, accessibility needs, and participation patterns.

In [2]:
```
import pandas as pd
```

Installing required package

The following command installs the pyreadstat package, a Python interface to the ReadStat C library. This library allows for fast and efficient reading Stata data files directly into pandas DataFrames.

In [3]:
```
pip install pyreadstat
```

```
Requirement already satisfied: pyreadstat in c:\anaconda3\lib\site-packages (1.2.8)
Requirement already satisfied: pandas>=1.2.0 in c:\anaconda3\lib\site-packages (from pyreadstat) (2.2.2)
Requirement already satisfied: numpy>=1.26.0 in c:\anaconda3\lib\site-packages (from pandas>=1.2.0->pyreadstat)
(1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\anaconda3\lib\site-packages (from pandas>=1.2.0->pyre
adstat) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\anaconda3\lib\site-packages (from pandas>=1.2.0->pyreadstat) (2
024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\anaconda3\lib\site-packages (from pandas>=1.2.0->pyreadstat)
(2023.3)
Requirement already satisfied: six>=1.5 in c:\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pandas>=
1.2.0->pyreadstat) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [64]:
```
pip install panel
```

```
Requirement already satisfied: panel in c:\anaconda3\lib\site-packages (1.5.2)
Requirement already satisfied: bleach in c:\anaconda3\lib\site-packages (from panel) (4.1.0)
Requirement already satisfied: bokeh<3.7.0,>=3.5.0 in c:\anaconda3\lib\site-packages (from panel) (3.6.0)
Requirement already satisfied: linkify-it-py in c:\anaconda3\lib\site-packages (from panel) (2.0.0)
Requirement already satisfied: markdown in c:\anaconda3\lib\site-packages (from panel) (3.4.1)
Requirement already satisfied: markdown-it-py in c:\anaconda3\lib\site-packages (from panel) (2.2.0)
Requirement already satisfied: mdit-py-plugins in c:\anaconda3\lib\site-packages (from panel) (0.3.0)
Requirement already satisfied: packaging in c:\anaconda3\lib\site-packages (from panel) (24.1)
Requirement already satisfied: pandas>=1.2 in c:\anaconda3\lib\site-packages (from panel) (2.2.2)
Requirement already satisfied: param<3.0,>=2.1.0 in c:\anaconda3\lib\site-packages (from panel) (2.1.1)
Requirement already satisfied: pyviz-comms>=2.0.0 in c:\anaconda3\lib\site-packages (from panel) (3.0.2)
Requirement already satisfied: requests in c:\anaconda3\lib\site-packages (from panel) (2.32.3)
Requirement already satisfied: tqdm in c:\anaconda3\lib\site-packages (from panel) (4.66.5)
Requirement already satisfied: typing-extensions in c:\anaconda3\lib\site-packages (from panel) (4.11.0)
Requirement already satisfied: Jinja2>=2.9 in c:\anaconda3\lib\site-packages (from bokeh<3.7.0,>=3.5.0->panel)
(3.1.4)
Requirement already satisfied: contourpy>=1.2 in c:\anaconda3\lib\site-packages (from bokeh<3.7.0,>=3.5.0->panel)
(1.2.0)
Requirement already satisfied: numpy>=1.16 in c:\anaconda3\lib\site-packages (from bokeh<3.7.0,>=3.5.0->panel)
(1.26.4)
Requirement already satisfied: pillow>=7.1.0 in c:\anaconda3\lib\site-packages (from bokeh<3.7.0,>=3.5.0->panel)
(10.4.0)
Requirement already satisfied: PyYAML>=3.10 in c:\anaconda3\lib\site-packages (from bokeh<3.7.0,>=3.5.0->panel)
(6.0.1)
Requirement already satisfied: tornado>=6.2 in c:\anaconda3\lib\site-packages (from bokeh<3.7.0,>=3.5.0->panel)
(6.4.1)
Requirement already satisfied: xyzservices>=2021.09.1 in c:\anaconda3\lib\site-packages (from bokeh<3.7.0,>=3.5.0
->panel) (2022.9.0)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\anaconda3\lib\site-packages (from pandas>=1.2->panel)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\anaconda3\lib\site-packages (from pandas>=1.2->panel) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\anaconda3\lib\site-packages (from pandas>=1.2->panel) (2023.
3)
Requirement already satisfied: six>=1.9.0 in c:\anaconda3\lib\site-packages (from bleach->panel) (1.16.0)
Requirement already satisfied: webencodings in c:\anaconda3\lib\site-packages (from bleach->panel) (0.5.1)
Requirement already satisfied: uc-micro-py in c:\anaconda3\lib\site-packages (from linkify-it-py->panel) (1.0.1)
Requirement already satisfied: mdurl~=0.1 in c:\anaconda3\lib\site-packages (from markdown-it-py->panel) (0.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\anaconda3\lib\site-packages (from requests->panel)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\anaconda3\lib\site-packages (from requests->panel) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\anaconda3\lib\site-packages (from requests->panel) (2.2.
3)
Requirement already satisfied: certifi>=2017.4.17 in c:\anaconda3\lib\site-packages (from requests->panel) (2024.
8.30)
Requirement already satisfied: colorama in c:\anaconda3\lib\site-packages (from tqdm->panel) (0.4.6)
Requirement already satisfied: MarkupSafe>=2.0 in c:\anaconda3\lib\site-packages (from Jinja2>=2.9->bokeh<3.7.0,>
=3.5.0->panel) (2.1.3)
Note: you may need to restart the kernel to use updated packages.
```

In [73]:
```
pip install panel bokeh
```

```
Requirement already satisfied: panel in c:\anaconda3\lib\site-packages (1.5.2)
Requirement already satisfied: bokeh in c:\anaconda3\lib\site-packages (3.6.0)
Requirement already satisfied: bleach in c:\anaconda3\lib\site-packages (from panel) (4.1.0)
Requirement already satisfied: linkify-it-py in c:\anaconda3\lib\site-packages (from panel) (2.0.0)
Requirement already satisfied: markdown in c:\anaconda3\lib\site-packages (from panel) (3.4.1)
Requirement already satisfied: markdown-it-py in c:\anaconda3\lib\site-packages (from panel) (2.2.0)
Requirement already satisfied: mdit-py-plugins in c:\anaconda3\lib\site-packages (from panel) (0.3.0)
Requirement already satisfied: packaging in c:\anaconda3\lib\site-packages (from panel) (24.1)
Requirement already satisfied: pandas>=1.2 in c:\anaconda3\lib\site-packages (from panel) (2.2.2)
Requirement already satisfied: param<3.0,>=2.1.0 in c:\anaconda3\lib\site-packages (from panel) (2.1.1)
Requirement already satisfied: pyviz-comms>=2.0.0 in c:\anaconda3\lib\site-packages (from panel) (3.0.2)
Requirement already satisfied: requests in c:\anaconda3\lib\site-packages (from panel) (2.32.3)
Requirement already satisfied: tqdm in c:\anaconda3\lib\site-packages (from panel) (4.66.5)
Requirement already satisfied: typing-extensions in c:\anaconda3\lib\site-packages (from panel) (4.11.0)
Requirement already satisfied: Jinja2>=2.9 in c:\anaconda3\lib\site-packages (from bokeh) (3.1.4)
Requirement already satisfied: contourpy>=1.2 in c:\anaconda3\lib\site-packages (from bokeh) (1.2.0)
Requirement already satisfied: numpy>=1.16 in c:\anaconda3\lib\site-packages (from bokeh) (1.26.4)
Requirement already satisfied: pillow>=7.1.0 in c:\anaconda3\lib\site-packages (from bokeh) (10.4.0)
Requirement already satisfied: PyYAML>=3.10 in c:\anaconda3\lib\site-packages (from bokeh) (6.0.1)
Requirement already satisfied: tornado>=6.2 in c:\anaconda3\lib\site-packages (from bokeh) (6.4.1)
Requirement already satisfied: xyzservices>=2021.09.1 in c:\anaconda3\lib\site-packages (from bokeh) (2022.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\anaconda3\lib\site-packages (from Jinja2>=2.9->bokeh) (2.1.
3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\anaconda3\lib\site-packages (from pandas>=1.2->panel)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\anaconda3\lib\site-packages (from pandas>=1.2->panel) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\anaconda3\lib\site-packages (from pandas>=1.2->panel) (2023.
3)
Requirement already satisfied: six>=1.9.0 in c:\anaconda3\lib\site-packages (from bleach->panel) (1.16.0)
Requirement already satisfied: webencodings in c:\anaconda3\lib\site-packages (from bleach->panel) (0.5.1)
Requirement already satisfied: uc-micro-py in c:\anaconda3\lib\site-packages (from linkify-it-py->panel) (1.0.1)
Requirement already satisfied: mdurl~=0.1 in c:\anaconda3\lib\site-packages (from markdown-it-py->panel) (0.1.0)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\anaconda3\lib\site-packages (from requests->panel)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in c:\anaconda3\lib\site-packages (from requests->panel) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\anaconda3\lib\site-packages (from requests->panel) (2.2.
3)
Requirement already satisfied: certifi>=2017.4.17 in c:\anaconda3\lib\site-packages (from requests->panel) (2024.
8.30)
Requirement already satisfied: colorama in c:\anaconda3\lib\site-packages (from tqdm->panel) (0.4.6)
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
import pyreadstat
```

Loading files (in .xpt format) into dataframe

The following code uses the pyreadstat library to load multiple NHANES .xpt (SAS transport) files into pandas DataFrames. Each of these files contains data from a different NHANES survey module, including demographics, body measurements, alcohol use, complete blood counts, and dietary intake. The function pyreadstat.read_xport() returns a tuple where the first element is the actual dataset in the form of a pandas DataFrame, and the second element contains metadata such as column labels and data types. By appending [0] to each function call, we extract only the DataFrame portion, which is what we need for further data analysis.

In [5]:
```
df1= pyreadstat.read_xport(r"C:\Users\YAKSH CHEEMA\Downloads\P_DEMO.xpt")[0]
df2= pyreadstat.read_xport(r"C:\Users\YAKSH CHEEMA\Downloads\P_BMX.xpt")[0]
df3= pyreadstat.read_xport(r"C:\Users\YAKSH CHEEMA\Downloads\P_ALQ.xpt")[0]
df4= pyreadstat.read_xport(r"C:\Users\YAKSH CHEEMA\Downloads\P_CBC.xpt")[0]
df5= pyreadstat.read_xport(r"C:\Users\YAKSH CHEEMA\Downloads\P_DR1TOT.xpt")[0]
```

Merging dataframes

This code merges the five individual NHANES DataFrames into a single combined dataset called dfs. Each merge is performed on the SEQN column, which serves as the unique participant identifier across all NHANES modules. The how="left" parameter ensures that all participants present in the main demographics file (df1) are retained, even if corresponding records from the

other modules are missing. This approach preserves the full sample while integrating additional information from body measurements, alcohol use, blood tests, and dietary intake into one unified DataFrame, ready for analysis.

In [6]:
```
dfs= df1.merge(df2, on= "SEQN", how= "left")\
     .merge(df3, on= "SEQN", how= "left")\
     .merge(df4, on= "SEQN", how= "left")\
     .merge(df5, on= "SEQN", how= "left")
```

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

Out[7]: (15560, 247)

In [8]:
```
dfs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15560 entries, 0 to 15559
Columns: 247 entries, SEQN to DRD370V
dtypes: float64(247)
memory usage: 29.3 MB
```

Filtering Data

This line removes any duplicate rows from the merged dataset dfs to ensure that each participant record is unique.

In [9]:
```
dfs.drop_duplicates(inplace=True)
```

This line removes all columns from the dfs DataFrame that contain any missing values. This step is useful for simplifying the dataset by retaining only complete variables, especially when preparing data for visualisation or modeling where missing values could cause errors or skew results.

In [10]:
```
dfs.dropna(axis= 1, how= 'any', inplace= True)
```

In [11]:
```
dfs.shape
```

Out[11]: (15560, 15)

In [12]:
```
dfs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15560 entries, 0 to 15559
Data columns (total 15 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   SEQN      15560 non-null  float64
 1   SDDSRVYR  15560 non-null  float64
 2   RIDSTATR  15560 non-null  float64
 3   RIAGENDR  15560 non-null  float64
 4   RIDAGEYR  15560 non-null  float64
 5   RIDRETH1  15560 non-null  float64
 6   RIDRETH3  15560 non-null  float64
 7   DMDBORN4  15560 non-null  float64
 8   SIALANG   15560 non-null  float64
 9   SIAPROXY  15560 non-null  float64
 10  SIAINTRP  15560 non-null  float64
 11  WTINTPRP  15560 non-null  float64
 12  WTMECPRP  15560 non-null  float64
 13  SDMVPSU   15560 non-null  float64
 14  SDMVSTRA  15560 non-null  float64
dtypes: float64(15)
memory usage: 1.8 MB
```

In [13]:
```
dfs
```

Out[13]:

	SEQN	SDDSRVYR	RIDSTATR	RIAGENDR	RIDAGEYR	RIDRETH1	RIDRETH3	DMDBORN4	SIALANG	SIAPROXY
0	109263.0	66.0	2.0	1.0	2.0	5.0	6.0	1.0	1.0	1.0
1	109264.0	66.0	2.0	2.0	13.0	1.0	1.0	1.0	1.0	1.0
2	109265.0	66.0	2.0	1.0	2.0	3.0	3.0	1.0	1.0	1.0
3	109266.0	66.0	2.0	2.0	29.0	5.0	6.0	2.0	1.0	2.0
4	109267.0	66.0	1.0	2.0	21.0	2.0	2.0	2.0	1.0	2.0
...	...	...	...	...	...	...	...	...	...	...
15555	124818.0	66.0	2.0	1.0	40.0	4.0	4.0	1.0	1.0	2.0
15556	124819.0	66.0	2.0	1.0	2.0	4.0	4.0	1.0	1.0	1.0
15557	124820.0	66.0	2.0	2.0	7.0	3.0	3.0	1.0	1.0	1.0
15558	124821.0	66.0	2.0	1.0	63.0	4.0	4.0	1.0	1.0	2.0
15559	124822.0	66.0	2.0	1.0	74.0	2.0	2.0	2.0	2.0	2.0

15560 rows × 15 columns

I deleted the columns SEQN and SDDSRVYR from the dfs DataFrame because, after reading the documentation on the NHANES website, I found that these columns were not useful for my analysis. SEQN is just a participant identifier used for merging the files, and SDDSRVYR indicates the survey cycle, which wasn't relevant for my current task since all the data comes from the same cycle. Removing them helped simplify the dataset and focus only on the variables that actually matter for my visualisations and insights.

In [14]:
```
dfs.drop(columns= ['SEQN','SDDSRVYR'], inplace= True)
```

In [15]:
```
dfs.shape
```

Out[15]:
```
(15560, 13)
```

I deleted the column RIDRETH1 from the dfs DataFrame because, after comparing it with RIDRETH3, I found that the only difference was that RIDRETH1 did not include the Asian population category, whereas RIDRETH3 provides a more complete representation of race/ethnicity, including Asians. Since RIDRETH3 is more comprehensive, I decided to keep it and remove RIDRETH1 to avoid redundancy in the dataset.

In [16]:
```
dfs.drop(columns= 'RIDRETH1', inplace= True)
```

Decoding

Changing column names: I carefully checked the NHANES website and read through the documentation for each dataset to understand what the columns represent. Based on that, I renamed several columns in the dfs DataFrame to make them more readable and meaningful for my analysis. For example, I renamed RIDSTATR to ParticipationStatus, RIAGENDR to Gender, and RIDAGEYR to AgeGroup. I followed this approach for all key variables like race, country of birth, interview language, interpreter and proxy use, and weight variables.

In [17]:
```
dfs.rename(columns= {
    'RIDSTATR': 'ParticipationStatus',
    'RIAGENDR': 'Gender',
    'RIDAGEYR': 'AgeGroup',
    'RIDRETH3': 'Race',
    'DMDBORN4': 'CountryofBirth',
    'SIALANG': 'LanguageofInterview',
    'SIAPROXY': 'ProxyUsed',
    'SIAINTRP': 'InterpreterUsed',
    'WTINTPRP': 'InterviewWeight',
    'WTMECPRP': 'ExamWeight',
    'SDMVPSU': 'VariancePSU',
```

```
    'SDMVSTRA': 'VarianceStratum'
}, inplace= True)
```

Decoding column values I performed these mappings after checking the NHANES documentation, where I found that many of the columns used numeric codes to represent categorical values. For example, ParticipationStatus used 1 and 2 to indicate whether a person was only interviewed or both interviewed and examined, and Gender used 1.0 and 2.0 for male and female. To make the dataset more readable and easier to interpret, I converted these coded values into clear labels like "Male", "Female", "English", "Yes", "No", and so on.

In [18]:
```
dfs['ParticipationStatus']= dfs['ParticipationStatus'].map({1: 'Interviewed only', 2: 'Interviewed and Examined'}
dfs['Gender']= dfs['Gender'].map({1.0: 'Male', 2.0: 'Female'})
dfs['AgeGroup']= dfs['AgeGroup'].apply(lambda x: '80 or above' if x== 80 else 'Below 80' )
dfs['Race']= dfs['Race'].map({1.0: 'Mexican American', 2.0: 'Other Hispanic', 3.0: 'Non- Hispanic Black', 4.0: 'N
dfs['CountryofBirth']= dfs['CountryofBirth'].map({1.0: 'US', 2.0: 'Others', 77.0: 'Refused', 99.0: 'Unknown'})
dfs['LanguageofInterview']= dfs['LanguageofInterview'].map({1.0: 'English', 2.0: 'Spanish'})
dfs['ProxyUsed']=dfs['ProxyUsed'].map({1.0: 'Yes', 2.0: 'No'})
dfs['InterpreterUsed']= dfs['InterpreterUsed'].map({1.0: 'Yes', 2.0: 'No'})
```

In [31]:
```
dfs
```

Out[31]:

	ParticipationStatus	Gender	AgeGroup	Race	CountryofBirth	LanguageofInterview	ProxyUsed	InterpreterUsed
0	Interviewed and Examined	Male	Below 80	Non-Hispanic Asian	US	English	Yes	No
1	Interviewed and Examined	Female	Below 80	Mexican American	US	English	Yes	No
2	Interviewed and Examined	Male	Below 80	Non-Hispanic Black	US	English	Yes	No
3	Interviewed and Examined	Female	Below 80	Non-Hispanic Asian	Others	English	No	No
4	Interviewed only	Female	Below 80	Other Hispanic	Others	English	No	No
...	...	...	...	...	...	...	...	...
15555	Interviewed and Examined	Male	Below 80	Non-Hispanic White	US	English	No	No
15556	Interviewed and Examined	Male	Below 80	Non-Hispanic White	US	English	Yes	No
15557	Interviewed and Examined	Female	Below 80	Non-Hispanic Black	US	English	Yes	No
15558	Interviewed and Examined	Male	Below 80	Non-Hispanic White	US	English	No	No
15559	Interviewed and Examined	Male	Below 80	Other Hispanic	Others	Spanish	No	No

15560 rows × 13 columns

Exploring Categorical Variable Distributions

I used the value_counts() function to quickly examine the distribution of values in each of the key categorical columns. This helped me understand how the data is spread across different categories such as participation status, gender, age group, race, country of birth, language of interview, and whether a proxy or interpreter was used.

In [19]:
```
print(dfs['ParticipationStatus'].value_counts())
print("---------------------------------------")
print(dfs['Gender'].value_counts())
print("---------------------------------------")
print(dfs['AgeGroup'].value_counts())
print("---------------------------------------")
print(dfs['Race'].value_counts())
print("---------------------------------------")
print(dfs['CountryofBirth'].value_counts())
print("---------------------------------------")
print(dfs['LanguageofInterview'].value_counts())
print("---------------------------------------")
print(dfs['ProxyUsed'].value_counts())
print("---------------------------------------")
print(dfs['InterpreterUsed'].value_counts())
```

```
ParticipationStatus
Interviewed and Examined    14300
Interviewed only             1260
Name: count, dtype: int64
---------------------------------------
Gender
Female    7839
Male      7721
Name: count, dtype: int64
---------------------------------------
AgeGroup
Below 80       14878
80 or above      682
Name: count, dtype: int64
---------------------------------------
Race
Non- Hispanic Black    5271
Non- Hispanic White    4098
Mexican American       1990
Non- Hispanic Asian    1638
Other Hispanic         1544
Other Race             1019
Name: count, dtype: int64
---------------------------------------
CountryofBirth
US         12525
Others      3028
Refused        6
Unknown        1
Name: count, dtype: int64
---------------------------------------
LanguageofInterview
English    14003
Spanish     1557
Name: count, dtype: int64
---------------------------------------
ProxyUsed
No     10041
Yes     5519
Name: count, dtype: int64
---------------------------------------
InterpreterUsed
No     15159
Yes      401
Name: count, dtype: int64
```

Examining relationships between different attributes

I used a crosstab to examine the relationship between AgeGroup and ProxyUsed. This allowed me to see how proxy usage varies between participants who are below 80 years old and those who are 80 or above. The crosstab displays the count of participants in each age group who either did or did not use a proxy, helping me identify any patterns or significant differences in accessibility needs based on age. I used the same approach for three additional comparisons: Race vs LanguageofInterview, CountryofBirth vs InterpreterUsed, and ParticipationStatus vs Gender.

In [20]:
```
pd.crosstab(dfs['AgeGroup'], dfs['ProxyUsed'])
```

Out[20]:

ProxyUsed	No	Yes
AgeGroup		
80 or above	623	59
Below 80	9418	5460

Surprisingly, proxy usage was more common among participants below 80 years of age, with nearly 36% requiring assistance during their interviews. In contrast, participants aged 80 or above had a much lower rate of proxy use, accounting for only 59 out of 682 individuals. This contradicts common assumptions that older adults are more likely to need assistance, and may reflect differences in survey engagement or living situations across age groups.

In [21]:
```
pd.crosstab(dfs['Race'], dfs['LanguageofInterview'])
```

Out[21]:

LanguageofInterview	English	Spanish
Race		
Mexican American	1115	875
Non- Hispanic Asian	1638	0
Non- Hispanic Black	5261	10
Non- Hispanic White	4097	1
Other Hispanic	873	671
Other Race	1019	0

Language needs varied significantly across racial groups. Spanish was frequently used among Mexican American and Other Hispanic participants, with these two groups having hundreds of interviews conducted in Spanish. Meanwhile, Non-Hispanic White, Black, Asian, and Other Race participants almost exclusively conducted their interviews in English, indicating that Spanish-language resources are particularly critical for Hispanic subpopulations.

In [22]:
```
pd.crosstab(dfs['Gender'], dfs['ParticipationStatus'])
```

Out[22]:

ParticipationStatus	Interviewed and Examined	Interviewed only
Gender		
Female	7215	624
Male	7085	636

Participation in both the interview and examination components of the survey was high across genders, with only a small portion of individuals completing the interview alone. Interestingly, female participants were slightly more likely to complete the full examination process compared to males, with 92% of females and 91% of males participating fully.

In [23]:
```
pd.crosstab(dfs['CountryofBirth'], dfs['InterpreterUsed'])
```

Out[23]:

InterpreterUsed	No	Yes
CountryofBirth		
Others	2680	348
Refused	5	1
US	12473	52
Unknown	1	0

Interpreter assistance was used far more often by participants born outside the United States, with 348 non-U.S. born individuals needing interpreters compared to only 52 among U.S.-born respondents. This highlights the significant impact of language barriers among the immigrant population and underscores the importance of providing interpretation services to ensure accessibility and accurate data collection in public health research.

The following code generates a grouped bar chart using Bokeh to visualize the estimated population coverage of NHANES participants who completed both the interview and physical examination, broken down by race and participation status. It uses the ExamWeight variable to reflect population representation, not individual counts, and includes hover tooltips and value labels for enhanced readability.

Visualization 1: Weighted Participation Analysis by Racial Group

Objective To visualize the estimated U.S. population representation of NHANES participants, categorized by racial group and participation status, using appropriate sample weights. This visualization distinguishes between individuals who only completed the interview and those who completed both the interview and physical examination, ensuring both groups are fairly represented in the chart regardless of the original weighting limitations.

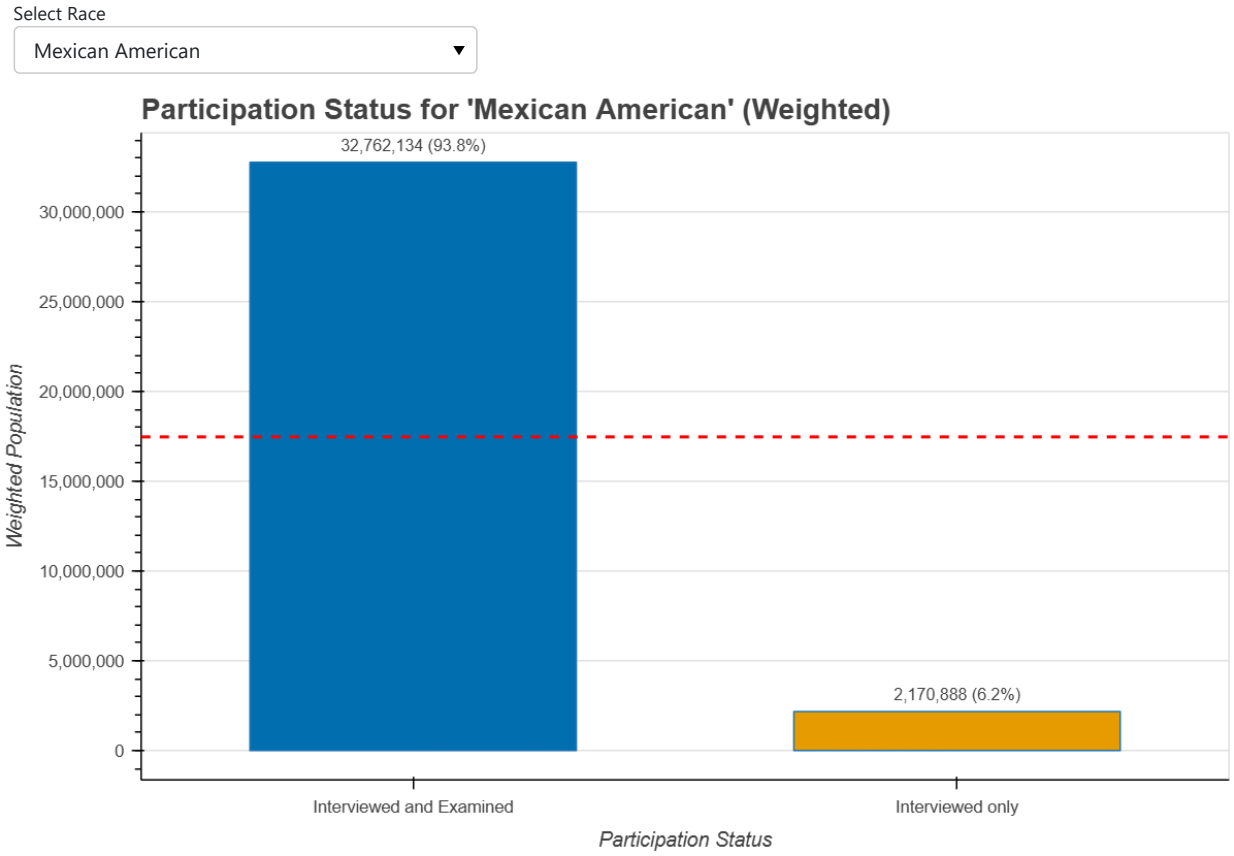
In [69]:
```
dfs['WeightForChart'] = dfs.apply(
    lambda row: row['InterviewWeight'] if row['ParticipationStatus'] == 'Interviewed only' else row['ExamWeight']
    axis=1
)
```

In [71]:
```
import pandas as pd
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource, HoverTool, LabelSet, NumeralTickFormatter, Span
from bokeh.transform import factor_cmap
from bokeh.palettes import Colorblind
from bokeh.io import show
import panel as pn
pn.extension()

# STEP 1: Smart weight selection based on ParticipationStatus
dfs['WeightForChart'] = dfs.apply(
    lambda row: row['InterviewWeight'] if row['ParticipationStatus'] == 'Interviewed only'
    else row['ExamWeight'],
    axis=1
)

# STEP 2: Group by Race and ParticipationStatus using combined weight
grouped = (
    dfs.groupby(['Race', 'ParticipationStatus'])['WeightForChart']
    .sum()
    .reset_index()
)

# STEP 3: Calculate % within each Race group
grouped['TotalByRace'] = grouped.groupby('Race')['WeightForChart'].transform('sum')
grouped['Percentage'] = (grouped['WeightForChart'] / grouped['TotalByRace']) * 100

# STEP 4: Formatting
grouped['DisplayWeight'] = grouped['WeightForChart'].apply(lambda x: x if x > 100000 else 100000)
grouped['LabelText'] = grouped.apply(
```

```
    lambda row: f"{int(row['WeightForChart']):,} ({row['Percentage']:.1f}%)", axis=1
)

# STEP 5: Dynamic ColumnDataSource (used inside plot function)
source = ColumnDataSource(grouped)

# STEP 6: Function to make a plot for a selected Race
def make_plot(selected_race):
    filtered = grouped[grouped['Race'] == selected_race].copy()
    filtered['x'] = filtered['ParticipationStatus']
    src = ColumnDataSource(filtered)

    # Create figure
    p = figure(
        x_range=filtered['x'],
        height=500,
        width=800,
        title=f"Participation Status for '{selected_race}' (Weighted)",
        toolbar_location=None,
        tools=""
    )

    # Draw bars
    p.vbar(
        x='x',
        top='DisplayWeight',
        width=0.6,
        source=src,
        fill_color=factor_cmap('x',
            palette=Colorblind[3],
            factors=filtered['ParticipationStatus'].unique().tolist()
        )
    )

    # Add labels
    labels = LabelSet(
        x='x',
        y='DisplayWeight',
        text='LabelText',
        y_offset=5,
        text_font_size="8pt",
        text_align="center",
        source=src
    )
    p.add_layout(labels)

    # Add mean reference line
    mean_weight = filtered['WeightForChart'].mean()
    mean_line = Span(location=mean_weight, dimension='width',
                     line_color='red', line_dash='dashed', line_width=2)
    p.add_layout(mean_line)

    # Hover tool
    hover = HoverTool(tooltips=[
        ("Participation", "@ParticipationStatus"),
        ("Weighted Count", "@WeightForChart{0,0}"),
        ("Percent", "@Percentage{0.0}%"),
        ("Race Total", "@TotalByRace{0,0}")
    ])
    p.add_tools(hover)

    # Styling
    p.yaxis.formatter = NumeralTickFormatter(format="0,0")
    p.xaxis.axis_label = "Participation Status"
    p.yaxis.axis_label = "Weighted Population"
    p.title.text_font_size = "14pt"
    p.xgrid.grid_line_color = None

    return p

# STEP 7: Interactive dropdown filter
race_select = pn.widgets.Select(name='Select Race', options=sorted(dfs['Race'].unique()))
interactive_plot = pn.bind(make_plot, selected_race=race_select)
```

```
# STEP 8: Show interactive panel
pn.Column(race_select, interactive_plot).servable()
```

Out[71]:



Description:

This enhanced interactive visualization presents the population representation of NHANES participants by race and participation status using a weighted metric that adapts to each participant's completion level. The dataset originally contained two key weighting columns:

InterviewWeight: used for all participants, including those who only completed the interview.

ExamWeight: used only for participants who completed both the interview and the physical exam.

However, NHANES assigns an ExamWeight of 0.0 to "Interviewed only" participants, which caused them to disappear in previous visualizations based solely on ExamWeight. To address this and provide a more accurate visual summary, a new column WeightForChart was created. This column assigns:

InterviewWeight to participants with "Interviewed only" status

ExamWeight to those with "Interviewed and Examined" status

This approach ensures all participant groups are visually represented proportionally to their estimated population coverage.

The visualization groups the data by race and participation status, then sums the WeightForChart to compute total representation. A percentage column was also added to reflect each group's contribution relative to its racial category. A minimum display threshold (DisplayWeight) is applied to ensure visibility of small values. The chart includes hover tooltips, numerical labels on each bar, and a mean reference line for analytical context. An interactive dropdown lets users filter the chart by race, making the visualization more engaging and customizable.

Conslusion:

This refined visualization effectively addresses the limitations of using a single weight column by dynamically selecting the appropriate weight (InterviewWeight or ExamWeight) based on participant completion status. It reveals that while some racial groups have a significant number of participants who only completed the interview, their representation was previously

masked due to a lack of assigned exam weights. By using WeightForChart, the visualization provides a more inclusive and comprehensive view of the NHANES population structure.

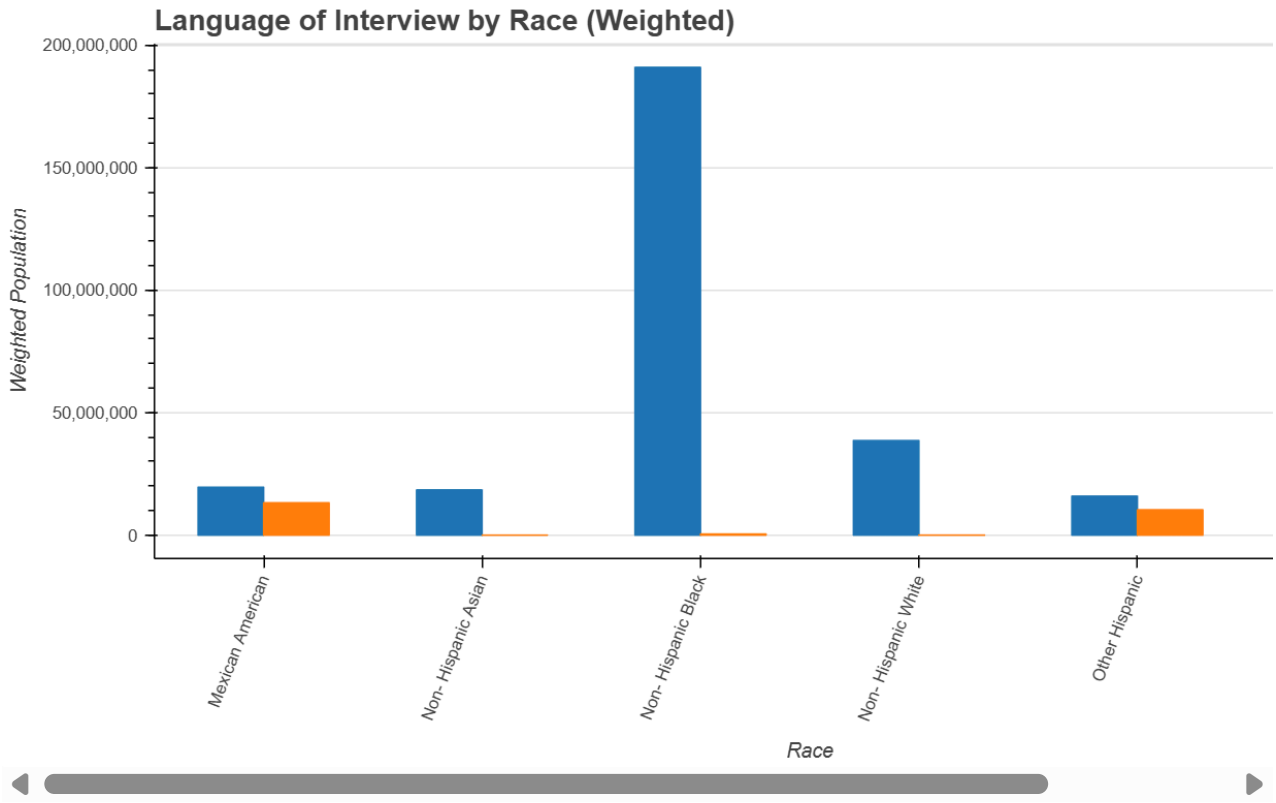
This approach reaffirms that NHANES weights are crucial not for counting individual participants, but for scaling their data to be representative of the U.S. population. Consequently, the height of each bar illustrates how many people in the general population a group represents — not just how many individuals responded. This enriched visualization offers deeper insight and fairness in representing all types of participation in the NHANES dataset.

Visualization 2: Language of Interview by Race (Weighted)

Objective To explore the distribution of interview languages (English vs. Spanish) among different racial groups in the NHANES dataset, using population-weighted values. This helps assess linguistic accessibility and the need for multilingual survey support across racial demographics.

In [60]:
```
import pandas as pd
from bokeh.io import output_notebook, show
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource, HoverTool, NumeralTickFormatter
from bokeh.transform import dodge
from bokeh.palettes import Category10

output_notebook()

# Group data and sum InterviewWeight
lang_group = (
    dfs.groupby(['Race', 'LanguageofInterview'])['InterviewWeight']
    .sum()
    .unstack(fill_value=0)
    .reset_index()
)

# Create a ColumnDataSource
source = ColumnDataSource(lang_group)

# Create figure
races = lang_group['Race'].tolist()
p = figure(x_range=races, height=500, width=950,
           title="Language of Interview by Race (Weighted)",
           toolbar_location=None, tools="")

# Plot stacked bars
p.vbar(x=dodge('Race', -0.15, range=p.x_range), top='English', width=0.3,
       source=source, color=Category10[3][0], legend_label="English")

p.vbar(x=dodge('Race', 0.15, range=p.x_range), top='Spanish', width=0.3,
       source=source, color=Category10[3][1], legend_label="Spanish")

# Add hover tool
hover = HoverTool(tooltips=[
    ("Race", "@Race"),
    ("English (Weighted)", "@English{0,0}"),
    ("Spanish (Weighted)", "@Spanish{0,0}")
])
p.add_tools(hover)

# Styling
p.yaxis.axis_label = "Weighted Population"
p.xaxis.axis_label = "Race"
p.xaxis.major_label_orientation = 1.2
p.yaxis.formatter = NumeralTickFormatter(format="0,0")
p.xgrid.grid_line_color = None
p.legend.location = "top_right"
p.title.text_font_size = "14pt"

# Show the plot
show(p)
```

Loading BokehJS ...



Description:

This visualization explores the relationship between race and language preference during interviews in the NHANES dataset. Specifically, it displays the estimated population-level distribution of participants who were interviewed in either English or Spanish, broken down by racial groups. To generate the chart, the dataset was grouped by Race and LanguageofInterview, and the InterviewWeight for each combination was summed. InterviewWeight reflects how many people in the broader U.S. population each participant represents — making it appropriate for measuring national-level language accessibility needs. The chart is rendered as a grouped bar chart, with each racial group displaying two bars: one for English and one for Spanish. Hover tooltips reveal the weighted population figures for each language, and the axes and labels are formatted for readability and clarity.

Conclusion:

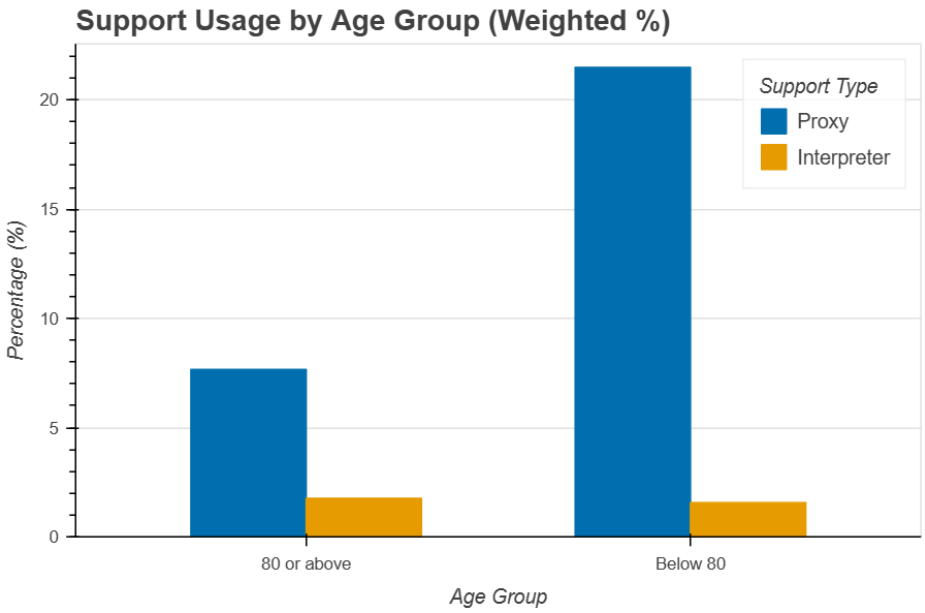
The visualization reveals clear and meaningful patterns in language use across racial groups. Spanish was predominantly used among Mexican American and Other Hispanic participants, reflecting the linguistic diversity and accessibility requirements of these communities. In contrast, nearly all participants identified as Non-Hispanic White, Non-Hispanic Black, Non-Hispanic Asian, and Other Race were interviewed in English, with negligible use of Spanish. These findings suggest that while English remains the default language of interview for the majority of racial groups, Spanish-language resources are essential for effective communication and inclusion of Hispanic subpopulations. This has important implications for the design of health surveys and public health services, emphasizing the need for multilingual support to ensure equitable participation and accurate data collection.

Visualization 3: Support Service Utilization by Age Group (Weighted %)

Objective To analyze and compare the utilization of accessibility support services—namely, proxy assistance and interpreter services—among different age groups of NHANES participants. By using interview weights, this visualization provides a population-level understanding of which age groups rely more heavily on support during the data collection process, highlighting potential accessibility gaps across demographics.

In [77]:
```
import pandas as pd
from bokeh.plotting import figure, show
from bokeh.models import ColumnDataSource, HoverTool, LabelSet
from bokeh.io import output_notebook
from bokeh.transform import dodge
from bokeh.palettes import Colorblind

output_notebook()

# STEP 1: Calculate weighted percentage of 'Yes' for Proxy and Interpreter by AgeGroup
data = []
for support_col, support_label in [('ProxyUsed', 'Proxy'), ('InterpreterUsed', 'Interpreter')]:
    grouped = dfs.groupby(['AgeGroup', support_col])['InterviewWeight'].sum().unstack(fill_value=0)
    total = grouped.sum(axis=1)
    percent_yes = (grouped.get('Yes', 0) / total) * 100

    for age_group in percent_yes.index:
        data.append({
            'AgeGroup': age_group,
            'SupportType': support_label,
            'Percentage': percent_yes.loc[age_group]
        })

# STEP 2: Convert to DataFrame and reshape
df_support = pd.DataFrame(data)
df_pivot = df_support.pivot(index='AgeGroup', columns='SupportType', values='Percentage').reset_index()

# STEP 3: Prepare data for Bokeh
source = ColumnDataSource(df_pivot)
age_groups = list(df_pivot['AgeGroup'])

# STEP 4: Create figure
p = figure(x_range=age_groups, height=400, width=600,
           title="Support Usage by Age Group (Weighted %)",
           toolbar_location=None, tools="")

bar_width = 0.3

# Proxy bars
p.vbar(x=dodge('AgeGroup', -bar_width/2, range=p.x_range), top='Proxy',
       width=bar_width, source=source, color=Colorblind[3][0], legend_label="Proxy")

# Interpreter bars
p.vbar(x=dodge('AgeGroup', bar_width/2, range=p.x_range), top='Interpreter',
       width=bar_width, source=source, color=Colorblind[3][1], legend_label="Interpreter")

# Hover tool
hover = HoverTool(tooltips=[
    ("Age Group", "@AgeGroup"),
    ("Proxy %", "@Proxy{0.0}%"),
    ("Interpreter %", "@Interpreter{0.0}%")
])
p.add_tools(hover)

# Style
p.x_range.range_padding = 0.1
p.xgrid.grid_line_color = None
p.y_range.start = 0
p.yaxis.axis_label = "Percentage (%)"
p.xaxis.axis_label = "Age Group"
p.legend.location = "top_right"
p.legend.title = "Support Type"
p.title.text_font_size = "14pt"

show(p)
```

Loading BokehJS ...



Description:

This side-by-side bar chart visualizes the percentage of participants who required proxy or interpreter services, segmented by age group. The chart is constructed using NHANES's InterviewWeight to reflect population-level estimates rather than raw counts. For each age group, two bars are presented: one showing the weighted percentage of participants who used a proxy, and the other showing those who used an interpreter. The data was grouped by AgeGroup and support type columns (ProxyUsed and InterpreterUsed), and the percentage of "Yes" responses was calculated within each age group. Tooltips and color-coded bars help highlight support differences clearly, making the chart intuitive and informative.

Conslusion:

The visualization reveals a surprising trend: participants below 80 years of age tend to rely more heavily on proxy support than those aged 80 or above. This may reflect broader participation of younger individuals with cognitive, literacy, or communication barriers that necessitate assistance, or it could be influenced by differences in data collection protocols or proxy eligibility criteria across age groups. In contrast, interpreter usage remains relatively low across all age categories but appears to be slightly higher among older participants, suggesting a modest increase in language-related accessibility needs with age.

This insight challenges the common assumption that older adults are uniformly more dependent on support services and instead points to a more nuanced relationship between age and accessibility requirements. Understanding these differences is essential for tailoring health survey methodologies to ensure equitable participation and accurate representation across demographic segments. It also underscores the importance of data-driven evaluation when allocating support resources or adjusting public health outreach strategies.

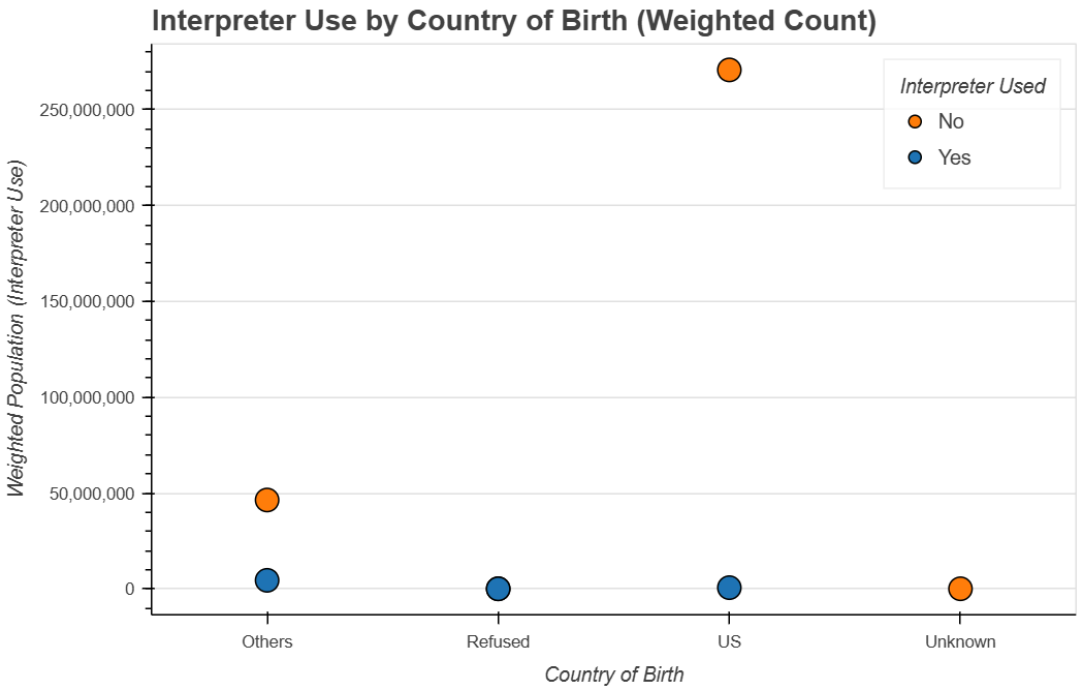
Visualization 4: Interpreter Use by Country of Birth (Weighted Count)

Objective To visualize the relationship between interpreter usage and participants' country of birth using weighted population data. This helps identify accessibility needs based on linguistic background and origin.

In [62]:
```
import pandas as pd
from bokeh.io import output_notebook, show
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource, HoverTool
from bokeh.palettes import Category10

output_notebook()
```

```
# Group data by CountryOfBirth and InterpreterUsed
grouped = (
    dfs.groupby(['CountryofBirth', 'InterpreterUsed'])['InterviewWeight']
    .sum()
    .reset_index()
)

# Create data source
source = ColumnDataSource(grouped)

# Create scatter plot
p = figure(x_range=grouped['CountryofBirth'].unique().tolist(),
           height=450,
           width=700,
           title="Interpreter Use by Country of Birth (Weighted Count)",
           toolbar_location=None, tools="")

# Draw circles
colors = {'Yes': Category10[3][0], 'No': Category10[3][1]}
grouped['color'] = grouped['InterpreterUsed'].map(colors)
source.data['color'] = grouped['color']

p.scatter(x='CountryofBirth',
          y='InterviewWeight',
          size=15,
          source=source,
          fill_color='color',
          line_color='black',
          legend_field='InterpreterUsed')


# Add hover
hover = HoverTool(tooltips=[
    ("Country", "@CountryofBirth"),
    ("Interpreter Used", "@InterpreterUsed"),
    ("Weighted Count", "@InterviewWeight{0,0}")
])
p.add_tools(hover)

# Style axes
p.yaxis.axis_label = "Weighted Population (Interpreter Use)"
p.xaxis.axis_label = "Country of Birth"
p.yaxis.formatter = NumeralTickFormatter(format="0,0")
p.title.text_font_size = "14pt"
p.legend.location = "top_right"
p.legend.title = "Interpreter Used"
p.xgrid.grid_line_color = None

# Show plot
show(p)
```

BokehJS 3.6.0 successfully loaded.



Description: This scatter plot visualizes the relationship between interpreter usage and participants' country of birth using NHANES data. The x-axis displays the participant's reported country of birth (US, Others, Refused, and Unknown), while the y-axis shows the total population each group represents, weighted by InterviewWeight. Each data point represents whether participants from that group used an interpreter during the interview, with color indicating interpreter usage (Yes or No). Hovering over each point reveals the exact weighted population. This format provides a clear and minimalistic way to observe trends in accessibility support based on country of origin.

Conslusion: The plot highlights a significant difference in interpreter usage between participants born in the United States and those born elsewhere. Interpreter use among U.S.-born participants was extremely low, whereas participants from other countries showed a substantially higher need for interpreter support, reinforcing the role of language as a key accessibility factor for foreign-born populations. This finding emphasizes the importance of providing interpreter services in national health surveys to ensure equitable participation and accurate data collection for linguistically diverse communities.

Visualization 5: Distribution of Interview Weights by Gender and Age Group

Objective To analyze how the interview weight (which reflects the estimated U.S. population each participant represents) is distributed across combinations of gender and age group. This helps assess balance and sampling design in the NHANES dataset.

In [63]:
```
import plotly.express as px

# Create a combined group label
dfs['Group'] = dfs['Gender'] + ' - ' + dfs['AgeGroup']

# Filter rows with InterviewWeight present
filtered_df = dfs.dropna(subset=['Group', 'InterviewWeight'])

# Create refined box plot
fig = px.box(
    filtered_df,
    x='Group',
    y='InterviewWeight',
    title='Distribution of Interview Weights by Gender and Age Group',
    labels={'InterviewWeight': 'Interview Weight'},
    color='Group',
    color_discrete_sequence=px.colors.qualitative.Set3,
    points=False, # Hide individual points for cleaner visual
```

```
    height= 600
)

# Show plot
fig.update_layout(showlegend=False)
fig.show()
```

Description: This box plot visualizes the variation in InterviewWeight across different combinations of gender and age group. The x-axis groups participants into categories such as “Male - Below 80” and “Female - 80 or above,” while the y-axis represents their assigned interview weight — a metric that estimates how many individuals each respondent represents in the overall U.S. population. The plot displays the median, interquartile range (IQR), and any statistical outliers for each demographic group. Colors differentiate each category, and the layout is optimized for readability by hiding individual data points and increasing figure height.

Conclusion: The distribution of interview weights reveals some variation between demographic groups. Generally, the IQR and median values are relatively consistent across categories, suggesting balanced population representation in the NHANES sample design. However, slight differences in spread or outliers may reflect specific subpopulations that were either oversampled or underrepresented in certain age or gender segments. This visualization reinforces the importance of understanding the weight variable not as a direct count, but as a reflection of national-level representation — key for drawing generalizable health conclusions from the survey data.

In [ ]: